ERA GLONASS Terminal
Communications Protocol
Transport Level

Version: 1.6

Code: ERA GLONASS

Contents

1. INTRODUCTION **4**
2. ABBREVIATIONS **5**
3. REFERENCES **6**
4. PURPOSE OF THE TRANSPORT LEVEL PROTOCOL **7**
4.1 ROUTING SUPPORT 7
4.2 DATA INTEGRITY CHECK MECHANISM 7
4.3 ENSURING DELIVERY RELIABILITY 7
5. DESCRIPTION OF THE PROTOCOL-BASED SYSTEM CONSTRUCTION PRINCIPLE **9**
6. NGTP-BASED PROTOCOL ANALYSIS **11**
7. DATA TYPE DESCRIPTION **12**
8. DATA STRUCTURE DESCRIPTION **14**
8.1 GENERAL PACKAGE STRUCTURE 14
8.1.1 PACKAGE COMPOSITION 14
8.1.2 PACKAGE FORMAT 14
8.1.3 PRV parameter 15
8.1.4 SKID Parameter 15
8.1.5 PRF Parameter 15
8.1.6 RTE (Route) Field 15
8.1.7 ENA (Encryption Algorithm) Field 15
8.1.8 CMP (Compressed) Field 15
8.1.9 PR (Priority) Field 15
8.1.10 HL Field 16
8.1.11 HE Field 16
8.1.12 FDL Field 16
8.1.14 PT Field 16
8.1.15 PRA Field 16
8.1.16 RCA Field 16
8.1.17 TTL Field 17
8.1.18 HCS Field 17
8.1.19 SFRD Field 17
8.1.20 SFRCS Field 17
8.2 DATA STRUCTURES DEPENDING ON THE PACKAGE TYPE 19

8.2.1 EGTS_PT_APPDATA PACKAGE DATA STRUCTURE 19
8.2.1.1 SDR 1, SDR 2, SDR n STRUCTURES 19
8.2.2 EGTS_PT_RESPONSE PACKAGE DATA STRUCTURE 19
8.2.2.1 RPID PARAMETER 20
8.2.2.2 PR PARAMETER 20
8.2.2.3 SDR 1, SDR 2, SDR n STRUCTURES 20
8.2.3 EGTS_PT_SIGNED_APPDATA PACKAGE DATA STRUCTURE 20
8.2.3.1 SIGL PARAMETER 20
8.2.3.2 SIGD PARAMETER 20
8.2.3.3 SDR 1, SDR 2, SDR n STRUCTURES 21
9. DESCRIPTION OF THE DATA STRUCTURE WHEN USING SMS AS A RESERVE DATA COMMUNICATION CHANNEL 22
9.1 SMS MESSAGE STRUCTURE 22
9.2 COMMUNICATION INFORMATION FORMAT DESCRIPTION 28
10. TEMPORAL AND QUANTITATIVE PARAMETERS OF THE TRANSPORT LEVEL PROTOCOL WHEN USING PACKAGE DATA TRANSMISSION 30
11. APPENDIX 1 – PROCESSING RESULT CODES 31
12. APPENDIX 2 – EXAMPLE OF IMPLEMENTATION OF CRC-16 CHECK SUM CALCULATION ALGORITHM USING C LANGUAGE 33
13. APPENDIX 3 – EXAMPLE OF IMPLEMENTATION OF CRC-8 CHECK SUM CALCULATION ALGORITHM USING C LANGUAGE 35

1. INTRODUCTION

This document describes the Information Transport Level Protocol between the ERA GLONASS terminal and the Operator infrastructure. The document gives all necessary data on the format and rules of message transfer of the Transport Level and is to be used together with the document “ERA GLONASS Terminal, Communications Protocol, Service Support Level” for the development of data transfer subsystems on the side of the ERA GLONASS terminals and the System Operator.

The document is intended for use by:

- terminal equipment manufacturers

- car manufacturers

- service developers and providers

- System Operator.

The OSI network model defines the following levels: physical, channel, network, transport, session, data presentation and applications. Within the OSI network model in the ERA GLONASS system the following protocols are used for data communication between the ERA GLONASS terminals and the System Operators: Transport Level – TCP protocol, network level –IP protocol. The correlation of the OSI network model, TCP/IP protocol stack and data transfer protocols of the ERA GLONASS system is presented in Table 1.

Table 1: Conformity of the levels of the OSI model, TCP/IP protocol stack and ERA GLONASS protocols

OSI Model		TCP/IP protocol stack		TCP/IP protocols	ERA GLONASS protocols
Level No.	Level description	Level No.	Level description		
7	Applications	4	Applications	FTP, HTTP, POP3, IMAP, telnet, SMTP, DNS, TFTP	Service Support Protocol
6	Data presentation				
5	Session				Transport Level
4	Transport	3	Transport	TCP, UDP	TCP
3	Network	2	Internetwork	IP	IP
2	Channel	1	Network access		
1	Physical				

2. ABBREVIATIONS

Abbreviation	Description
ST	Subscriber Terminal
GLONASS	GLObal NAvigation Satellite System
RA	Road Accident
TUI	Terminal User Interface
NIS	Navigation Information Systems
RAM	Random Access Memory
SW	Software
TP	Telematic Platform
ERA	Emergency Accident Response
EGTS	Era Glonass Telematics Standard
GSM	Global System for Mobile communications
ISDN	Integrated Services Digital Network
NGTP	Next Generation Telematics Protocol. Architecture and concept.
PDU	Protocol Description Unit
SC	Service Center
SIM	Subscriber Identification Module
SME	Short Message Entity - objects able to transmit and receive SMS messages
SMS	Short Message Service
SMSC	Short Message Service Center

3. REFERENCES

[1] RFC1323: “TCP Extensions for High Performance”

[2] GSM 03.38 (ETS 300 628): "Digital cellular telecommunication system (Phase 2); Alphabets and language-specific information".

[3] GSM 03.40 (ETS 300 536): "Digital cellular telecommunication system (Phase 2); Technical realization of the Short Message Service (SMS) Point to Point (PP)".

4. PURPOSE OF THE TRANSPORT LEVEL PROTOCOL

The Transport Level Protocol is intended for routing of information contained in the Service Support Protocol between infrastructure points and ST using the Protocol, checking of integrity and proper sequence of data, as well as ensuring reliability of delivery to the point of destination.

4.1 ROUTING SUPPORT

The Protocol is based on the principle of flexible routing of data packages between interconnected elements of a TP distributed network using the Protocol. Routing addresses are TP identifiers which are to be unique within one interconnected network.

4.2 DATA INTEGRITY CHECK MECHANISM

Checking transmitted information for integrity is based on the application of check sums of the Transport Level protocol, and the Service Support Level data. The receiver calculates the check sums and compares them with the corresponding values entered by the dispatcher in the relevant fields of the package. If the check sums do not match, the integrity is considered violated, a message thereof is sent as a processing result error code.

In order to minimize the use of system resources when processing packets of the Protocol of the Transport Layer and Data Layer Support Services uses a variety of fields and algorithms to ensure integrity. It uses a mechanism based on the calculation of the checksum byte transmitted (CRC).

CRC algorithms provide a lower probability of erroneous reception of information than, for example, the algorithm is XOR. CRC-8 provides an equal probability of 1e-19 and the algorithm XOR8 - 0.11-0.12. Use the same table the CRC calculation algorithm provides a relatively high speed of calculation.

The use of one-way functions, "hash" achieves lower levels of probability of erroneous reception of information than the algorithms of CRC, but the implementation of this method requires large computational resources in an efficient outcome is given a sequence of bytes significant length of 16-32.

Based on the foregoing, the decision to use the algorithm for CRC.

For part of the Transport Layer packet used algorithm for computing the cyclic redundancy check CRC-8.

For part of a package of Level Support Services uses an algorithm for computing cyclic redundancy code CRC-16.

4.3 ENSURING DELIVERY RELIABILITY

The mechanism to ensure safe delivery is based on the use of confirmations of packages sent earlier. The dispatcher transmitting the package expects its confirmation in the form of a package of a certain type which contains the identifier of the package transmitted earlier, and the code of the result of its processing by the receiver. The expectation continues for a certain period of time specified by the Protocol, which depends on the type of the lower level transport protocol (TL_RESPONSE_TO parameter in Table 4). After the confirmation is received, the dispatcher analyses the result code. The processing result codes are also specified by the Protocol, these are presented in Appendix 1. Further, depending on the analysis result, the package is considered delivered or not delivered. The package is also considered not delivered in case the confirmation does not arrive on the expiration of the TL_RESPONSE_TO interval. Non-delivered packages are sent again (the number of the dispatch attempts is specified by the Protocol and is defined by the TL_RESEND_ATTEMPTS parameter from Table 4). On achieving the limited number of dispatch attempts the data link is considered unreliable, and the established session is cancelled (disconnection in case the TCP/IP protocol is used as the transport one) and an attempt to create a new session (connection) is made after a delay defined by the TL_RECONNECT_TO parameter (see Table 4).

5. DESCRIPTION OF THE PROTOCOL-BASED SYSTEM CONSTRUCTION PRINCIPLE

The minimum and sufficient element of the system using the Protocol is TP. As the basic component of TP performing the functions of coordination of intraplatform interaction and routing the concept of the Dispatcher is used.

The protocol differentiates between the logical level of interplatform routing, in which data (information packages) are transmitted at the level of individual TPs, and the level of intraplatform routing, where information is transmitted between separate Services of one TP. A Service is a separate component of TP which ensures functional performance of the algorithm of a particular service using the Protocol described. Interaction in all specified types of routing occurs through the Dispatcher.

The generators and consumers of data in the protocol-based system are Services which create packages on one side – the dispatcher – and on the other side – the receiver – process the packages received from other Services. Each service implements a different business logic depending on the specific Service functionality. The Service type is its main functional characteristic and is used by the Dispatcher for intraplatform data routing. As a rule, there is a complementary pair of Services participating in the interaction: one of them located on the Terminal, for example, generates the packages with co-ordinate data and sensor indications, and the other – on TP – processes such data.

All Services within the framework of one TP are connected with the Dispatcher and have no direct connections among themselves.

The telematic platform can communicate with other TPs and exchange data on the basis of the routing data. For routing, the Dispatcher accesses the local databank containing the data about the neighboring TPs and the Services available on them, and also the information on the Services functioning within the framework of their TPs. When establishing communication between Dispatchers of different TPs, information is shared on the types of the Services accessible on each of the sides, and also their status. Route search is reduced to searching a direction (connection) for the requested Service type. If the requested Service is on the same TP as the Dispatcher, interaction occurs with the use of intraplatform routing only. Otherwise, where so authorized, the Service is searched for according to the routing data on the neighboring TPs, and when such route is found and is available, the request is translated to the TP found, and the identifier of the remote TP Dispatcher is used as the address.

The terminal also uses a Dispatcher to communicate with TP Services. In doing so, it is identified by special packages containing a unique Terminal number assigned during registration in the system, and also other account data and information on the internal infrastructure and status of modules and units.

A block diagram of interaction of the system elements based on the Protocol described is shown in Fig. 1.
Each Service has a certain type which in Fig. 1 is defined by the SID parameter.

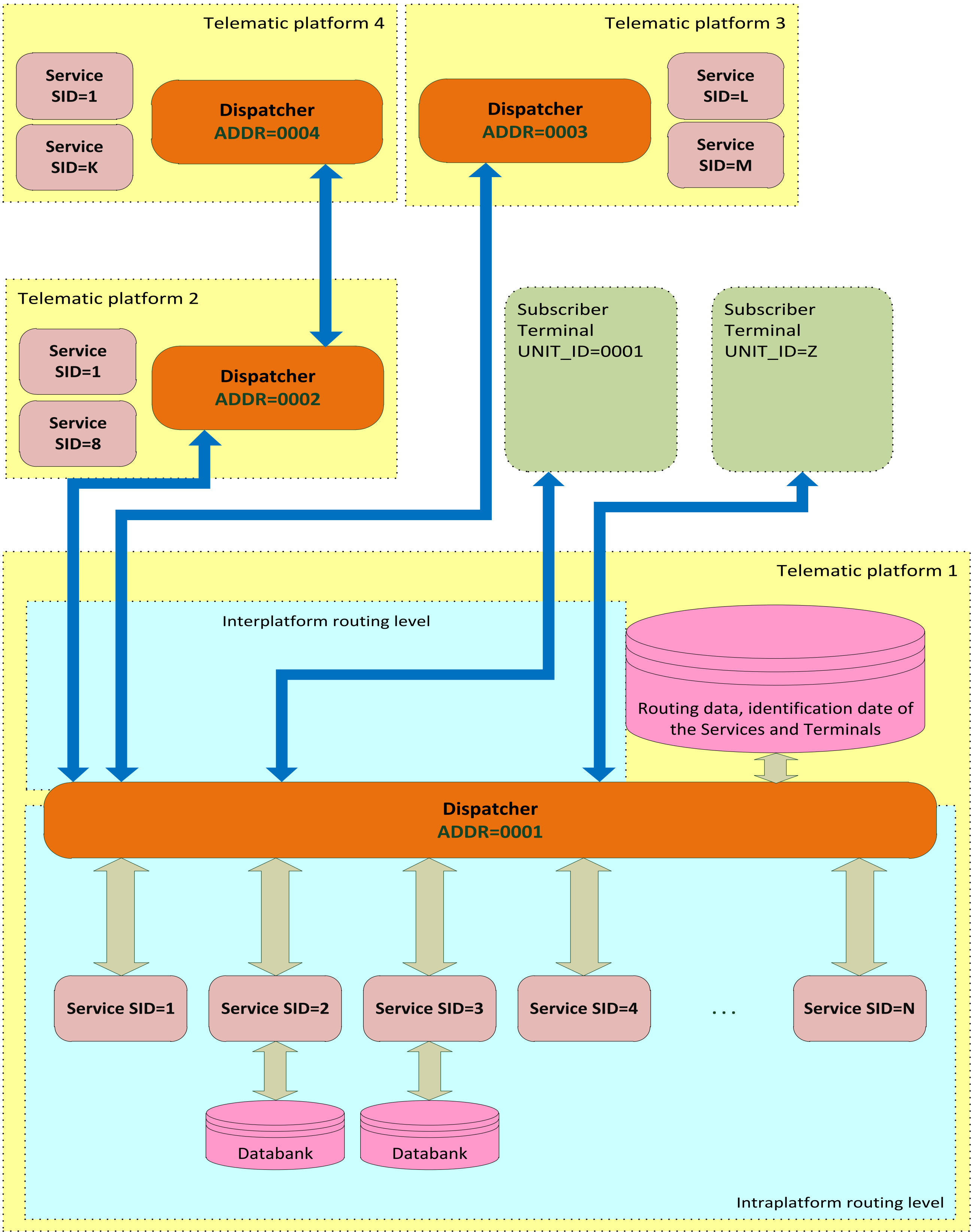


Fig. 1: Structural scheme of element interaction in a Protocol-based system

6. NGTP-BASED PROTOCOL ANALYSIS

According to the NGTP-based telematic systems concept, there are three basic elements of communication: telematic unit (TU), telematic service provider (TSP) and dispatcher. All these entities communicate through standard interfaces and are elements of the Protocol interaction, with the exception of TSP which is combined with the Dispatcher in the Protocol.

As a rule, a TU is integrated into a vehicle (ST), but can also be a personal navigating device or a mobile phone.

TSP has two tasks: data transfer from Services to TUs, and data transfer from TUs to Services.

According to NGTP, the dispatcher is an intermediary between a TSP and a TU, and provides a standard communication interface for a TU with other components of the system ensuring Services functionality performance. The dispatcher only handles data of its level and does not analyze the composition of Service level data.

The NGTP header completely coincides with the first bytes of the Transport Level Protocol header: Protocol Version (1 byte), Security Context (2 bytes), NGTPHeaderLength (1 byte), NGTPHeaderEncoding (1 byte)

In NGTP, the TU identifier is the VIN/DriveID, in the Protocol described it is the UNIT_ID.

To identify a TU installed as part of the standard equipment configuration, the VIN is used.

The same as NGTP, the Protocol is aimed at flexible routing of the Service data from ST to TP, and from TP to ST. Furthermore, introduction of a new Service does not require the Protocol update as the Protocol only performs the data routing, and processing is conducted directly by the Service. It is only necessary to correctly set up the routing of the Dispatcher to the new type of Service, which is done by means of administering the Protocol-based system.

NGTP uses the concept of “Event” to define some general characteristic of data, and it is intended for integration of information of different types into a certain generalized data array. Each event identifier also corresponds to an attribute identifying the event generation time. The use of such mechanism of generalization is embedded in the Protocol in which each Record of the Service Support Protocol can contain the event identifier generated by the source of such records in a certain period of time, for example, when a road accident occurs.

Unlike NGTP which uses various interfaces between TU and the dispatcher, the dispatcher and TSP and between TSP and Services, the Protocol uses one interface for communication between components.

NGTP uses the concept of the trigger which means a certain notice of the system components that there is information received for them. On accepting such trigger, the recipient is to request information and process it. The Protocol does not use triggers, and information is directly transferred to the recipient.

7. DATA TYPE DESCRIPTION

The Protocol defines and uses a number of different types of field and parameter data. A description of the types of data in the Protocol is given in Table 2.

Table 2: Composition of Protocol data types

Data type	Length, bytes	Value range	Description
BOOLEAN	1	TRUE=1, FALSE=0	Logical type which can have only two values: TRUE or FALSE
BYTE	1	0 … 255	Unsigned integer
USHORT	2	0 … 65535	Unsigned integer
UINT	4	0 … 4294967295	Unsigned integer
ULONG	8	0 … 18446744073709551615	Unsigned integer
SHORT	2	-32768 … + 32767	Signed integer
INT	4	-2147483648 … +2147483647	Signed integer
FLOAT	4	±1.2 E – 38 … 3.4 E + 38	Fractional signed number
DOUBLE	8	±2.2 E – 308 … 1.7 E + 308	Fractional signed number
STRING	Variable. The length is defined by external parameters or use of a special terminator character (code 0x00)		Contains a sequence of printed characters in the default coding (CP-1251), unless a different coding is expressly stated (by an additional parameter)
BINARY	Variable. The length is defined by external parameters		Contains a sequence of BYTE type data
ARRAY OF TYPE	Variable. The length is defined by external		Can contain a sequence of one of the above types (TYPE), except BINARY. The sequence order and length of each element of the type used is defined by the type itself. Type copies follow each other sequentially. For

	parameters		instance: ARRAY OF STRING [10] contains 10 copies of the STRING type, the length of each copy defined by the separator (code 0x00), to be present between the copies.

Multibyte data types USHORT, UINT, ULONG, FLOAT and DOUBLE use a little - endian byte (lower byte forward) sequence. The bytes making the sequence in the STRING and BINARY types should be interpreted as is, i.e. to be processed when received.

Here and hereinafter the following types of fields and parameters are described:

M – mandatory parameter. The parameter is to be always transferred;

O – optional. The parameter can be excluded, and its presence is defined by other parameters in the package.

8. DATA STRUCTURE DESCRIPTION

8.1 GENERAL PACKAGE STRUCTURE

8.1.1 PACKAGE COMPOSITION

The Transport Level protocol composition is shown in Fig. 1.

Transport Level protocol header	Service Support Level Data	Check sum for the Service Support Level Data

Fig. 2: Transport Level Protocol package composition

The Transport Level protocol package consists of a header, Service Support Level Data field and Check sum field for the Service Support Level Data.

8.1.2 PACKAGE FORMAT

The total length of the Transport Level Protocol package does not exceed 65535 bytes, which corresponds to the maximum value of the Window Size parameter (maximum size of the entire package received by the recipient) of the TCP Protocol header. This maximum package size value optimizes the use of data transmission channels, based only on the standard method of data flow management embedded in the TCP/IP protocol [1]. The composition of the Transport Level package is shown in Table 3.

Table 3: Composition of the Transport Level Protocol package

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Type	Data type	Size, bytes
PRV (Protocol Version)								M	BYTE	1
SKID (Security Key ID)								M	BYTE	1
PRF (Prefix)		RTE	ENA		CMP	PR		M	BYTE	1
HL (Header Length)								M	BYTE	1
HE (Header Encoding)								M	BYTE	1
FDL (Frame Data Length)								M	USHORT	2
PID (Packet Identifier)								M	USHORT	2
PT (Packet Type)								M	BYTE	1
PRA (Peer Address)								O	USHORT	2
RCA (Recipient Address)								O	USHORT	2
TTL (Time To Live)								O	BYTE	1

HCS (Header Check Sum)	M	BYTE	1
SFRD (Services Frame Data)	O	BINARY	0 … 65517
SFRCS (Services Frame Data Check Sum)	O	USHORT	0, 2***

The Transport Level protocol header includes the following fields: PRV, PRF, PR, CMP, ENA, RTE, HL, HE, FDL, PID, PT, PRA, RCA, TTL, HCS. The Service Support protocol is represented by the SFRD field, the check sum for the Service Support Level is contained in the SFRCS field.

8.1.3 PRV parameter

The parameter indicates the version of the header structure used and should contain the value 0x01. The value of this parameter is incremented each time the header structure is modified.

8.1.4 SKID Parameter

The parameter defines the identifier of the key used in encoding.

8.1.5 PRF Parameter

The given parameter defines the prefix of the Transport Level header, and for the given version should contain the value 00.

8.1.6 RTE (Route) Field

The bit field defines the need for further routing of the package to a remote telematic platform, and also the presence of optional parameters PRA, RCA, TTL, necessary for the package routing. If the field contains value 1, routing is required, and the PRA, RCA, TTL fields are used in the package. The field is set by the Dispatcher of the TP on which the package is generated, or ST which generated the package for sending to TP, in case it has the «HOME_DISPATCHER_ID» parameter defining the TP address at which the given ST is registered.

8.1.7 ENA (Encryption Algorithm) Field

The bit field defines the algorithm code used to encode the data from the SFRD field. If the field value is 0 0, the data in the SFRD field are not encrypted. The composition and codes of the algorithms are not defined in this version of the Protocol

8.1.8 CMP (Compressed) Field

The bit field defines whether compression of data from the SFRD field is used. If the field value is 1, the data in the SFRD field are considered compressed. The algorithm of compression is not defied in this version of the Protocol.

8.1.9 PR (Priority) Field

The bit field defines the routing priority of the package and can have the following values:

0 0 – highest

0 1 – high

1 0 – medium

1 1 – low

Setting a higher priority allows transmission of packages with urgent data, such as a package with a minimum ERA GLONASS service data set or data on the vehicle alarm system actuation. When receiving a package, the Dispatcher, analyzing the given field, routes a package with a higher priority faster, than a package with a lower priority, thereby enhancing the processing efficiency when critical events occur.

8.1.10 HL Field

The Transport Level header length in bytes including the check sum byte (HCS field);

8.1.11 HE Field

The field defines the method for encoding the Transport Level header part following the given parameter.

8.1.12 FDL Field

The filed defines the byte length of the SFRD field containing Service Support Protocol information.

8.1.13 PID Field

The field contains the Transport Level package number which is increased by 1 when each new package is sent on the dispatcher side. Values in this field change by the cyclic counter rules from 0 to 65535, i.e. when the 65535 value is achieved, the following value should be 0.

8.1.14 PT Field

Transport Level type package. The PT Field can have the following values:

0 – EGTS_PT_RESPONSE (confirmation of the Transport Level package);

1 – EGTS_PT_APPDATA (package containing Service Support Protocol data);

2 – EGTS_PT_SIGNED_APPDATA (package containing Service Support Protocol data with a digital signature);

8.1.15 PRA Field

TP address to which the given package is generated. This address is unique for an interconnected network and is used to generate a confirmation package for the receiver.

8.1.16 RCA Field

TP address for which the package is intended. Based on this address, the package appurtenance to a certain TP is identified during its routing when intermediate TPs are used.

8.1.17 TTL Field

Life time of a package during its routing between TPs. The use of this parameter prevents cycling of retransmitted packages in systems with complex topology of address points. Originally, TTL is established by TP which generated the package. The TTL value is set to the maximum permissible number of TPs between dispatching and receiving TPs . The TTL value decreases by 1 during package transmission through each TP, with the check sum of the Transport Level header recalculated. When the parameter becomes 0, and the need for further routing of the package is identified, it shall be destroyed and a confirmation with the corresponding code shall be issued (PC_TTLEXPIRED – see Appendix 1).

8.1.18 HCS Field

Check sum for the Transport Level header (from the «PRV» field to the «HCS» field, the latter excluded). To calculate the value of the HCS field the CRC-8 algorithm is used. The example of programming code for calculation of CRC-8 algorithm is represented in Appendix 3.

8.1.19 SFRD Field

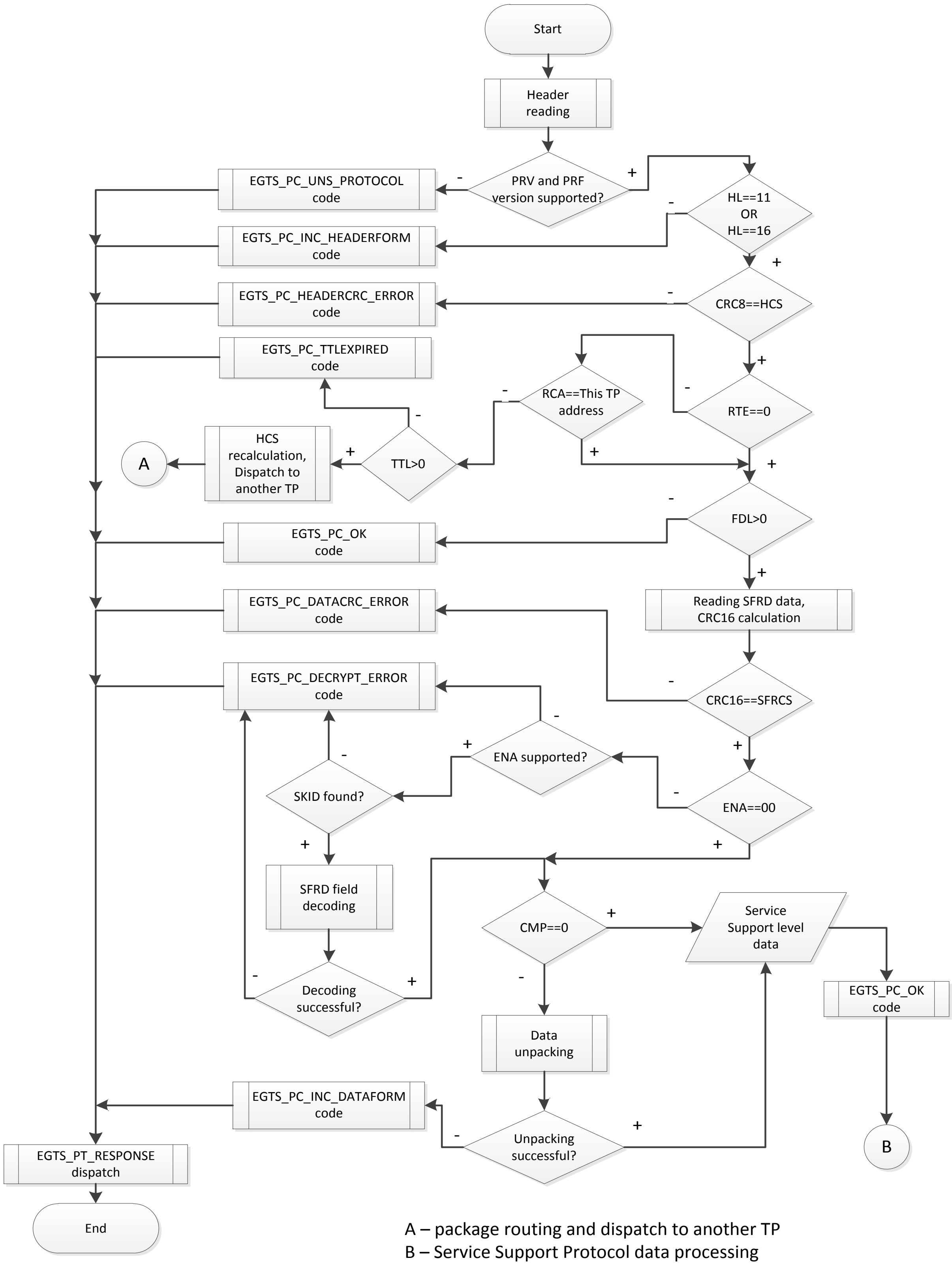
Data structure which is determined by the Package type and contains Service Support Protocol information. The format of the data structure depending on the Package type is described in p. 6.25.

8.1.20 SFRCS Field

Check sum of the Data communication protocol level field. The check sum is calculated based on data from the SFRD field, using the CRC-16 algorithm. The example of programming code for calculation of CRC-16 algorithm is represented in Appendix 2.

The flow chart of the Transport Level Protocol Package assembly algorithm at the receipt stage is presented in Figure 3.

The packet parser’s implementation may not send the appropriate responses on packets are containing the format structure errors or invalid values of fields. Such behavior may be given by security politics or another causes. In those cases (if the connection oriented protocol is used, TCP/IP) the packet parser may break the established connection.



A – package routing and dispatch to another TP
B – Service Support Protocol data processing

Fig. 3: Flow chart of the Transport Level Protocol Package assembly algorithm at the receipt stage

8.2 DATA STRUCTURES DEPENDING ON THE PACKAGE TYPE

Depending on the Transport Level package type, the SFRD field structure can have a different format.

8.2.1 EGTS_PT_APPDATA PACKAGE DATA STRUCTURE

This type of package is intended to transmit one or several structures containing the Service Support Protocol information. The SFRD field data structure for a EGTS_PT_APPDATA package is shown in Table 4.

Table 4: SFRD field format for the EGTS_PT_APPDATA type package

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Type	Data type	Bytes
SDR 1(Service Data Record)								O	BINARY	9 … 65517
SDR 2								O	BINARY	9 … 65517
. . .										
SDR n								O	BINARY	9 … 65517

8.2.1.1 SDR 1, SDR 2, SDR n STRUCTURES

Structures containing the Service Support Protocol information. There may be one or more successive structures of this type in the SFRD field. The internal composition of these structures is described in ERA GLONASS terminal, Communications Protocol, Service Support Level.

8.2.2 EGTS_PT_RESPONSE PACKAGE DATA STRUCTURE

This package makes it possible to confirm a Transport Level package and, in addition to the structures containing the Service Support Level information, includes information on the Transport Level Protocol data processing result received earlier. The SFRD field data structure of the EGTS_PT_RESPONSE package is shown in Table 5

Table 5: SFRD field format for the EGTS_PT_RESPONSE type package

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Type	Data type	Bytes
RPID (Response Packet ID)								M	USHORT	2
PR (Processing Result)								M	BYTE	1
SDR 1(Service Data Record)								O	BINARY	9 … 65517
SDR 2								O	BINARY	9 … 65517
. . .										

SDR n	O	BINARY	9 … 65517

8.2.2.1 RPID PARAMETER

Transport Level package identifier, confirmation to which is generated.

8.2.2.2 PR PARAMETER

Processing result code for the Package part related to the Transport Level (calculation of the check sums of the Transport Level header and the Service Support Level data, checking the package size, identifying the need for further routing of the Package, etc.). The list of possible codes of the processing results is given in Appendix 1.

8.2.2.3 SDR 1, SDR 2, SDR n STRUCTURES

Structures containing the Service Support Level information. There may be one or more successive structures of this type.

8.2.3 EGTS_PT_SIGNED_APPDATA PACKAGE DATA STRUCTURE

The package of this type is used to transmit not only structures containing the Service Support Level information, but also information on the so-called «digital signature» identifying the package dispatcher. The SFRD field data structure of the EGTS_PT_ SIGNED_APPDATA package is shown in Table 6.

Table 6: SFRD field format for the EGTS_PT_SIGNED_APPDATA type package

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Type	Data type	Bytes
SIGL(Signature Length)								M	SHORT	2
SIGD (Signature Data)								O	BINARY	0 … 512
SDR 1(Service Data Record)								O	BINARY	9 … 65515
SDR 2								O	BINARY	9 … 65515
. . .										
SDR n								O	BINARY	9 … 65515

8.2.3.1 SIGL PARAMETER

It defines the length of the «digital signature» data from the SIGD field

8.2.3.2 SIGD PARAMETER

It contains the «digital signature» data.

8.2.3.3 SDR 1, SDR 2, SDR n STRUCTURES

Structures containing the Service Support Level information. There may be one or more successive structures of this type.

For each package of the EGTS_PT_APPDATA or EGTS_PT_SOGNED_APPDATA type delivered from ST to TP or from TP to ST, there shall be a EGTS_PT_RESPONSE type package sent, containing the package number from the EGTS_PT_APPDATA or EGTS_PT_SOGNED_APPDATA package in the PID field. Fig. 2 shows the sequence of package exchange in ST and TP communication.

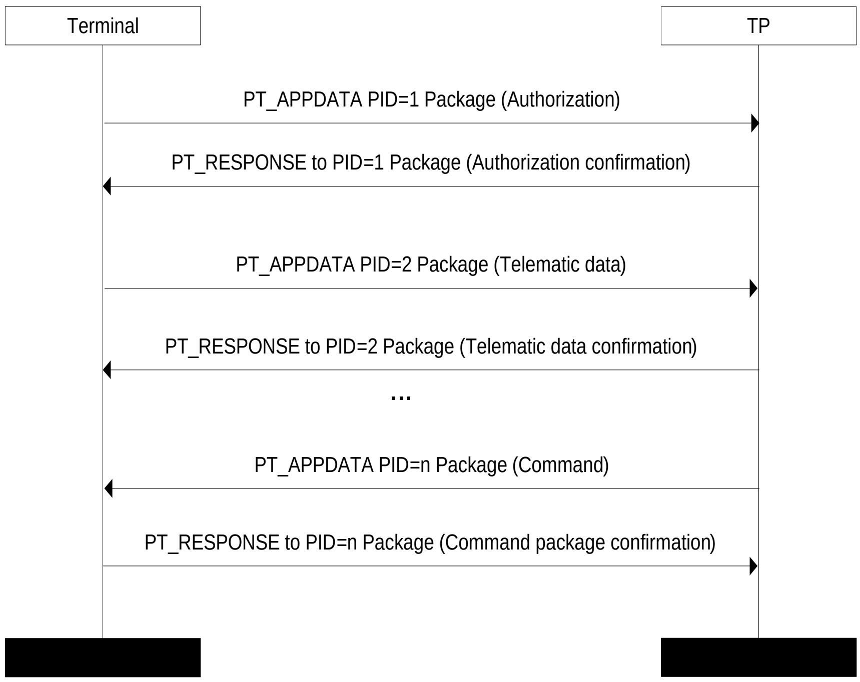


Fig. 4: Communication of the Terminal and TP at the Transport package level

9. DESCRIPTION OF THE DATA STRUCTURE WHEN USING SMS AS A RESERVE DATA COMMUNICATION CHANNEL

9.1 SMS MESSAGE STRUCTURE

When SMS is used to transmit the Protocol data packages the PDU mode is used [2], [3]. The PDU mode enables transmission of not only verbal, but also binary information through the SMS service of the GSM cellular communication operator. The Protocol described operates with binary data, that is why the PDU mode is best suited to use SMS as a reserve Transport level transmission channel.

Transmission is performed using 8-bit encoding. The SMS message format for dispatch in the PDU mode is given in Table 7 and uses the structure SMS-SUBMIT described in p. 9.2.2.2 [3].

Table 7: SMS format using the PDU mode (SMS-SUBMIT)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Type	Bytes
SMSC_AL (SMSC Address Length)								M	1
SMSC_AT (SMSC Address Type)								O	0,1
SMSC_A (SMSC Address)								O	0,6
TP_RP	TP_UDHI	TP_SRR	TP_VPF		TP_RD	TP_MTI		M	1
TP_MR (Message Reference)								M	1
TP_DA_L (Destination Address Length)								M	1
TP_DA_T (Destination Address Type)								M	1
TP_DA (Destination Address)								M	6
TP_PID (Protocol Identifier)								M	1
TP_DCS (Data Coding Scheme)								M	1
TP_VP (Validity Period)								O	0, 1, 7
TP_UDL (User Data Length)								M	1

TP_UD (User Data)	O	0…140

Description of parameters included in SMS in the PDU mode:

- SMSC_AL – Length of useful data of the SMSC address in octets plus 1 octet of SMSC_AT
- SMSC_AT – Type of the SMSC address format. The possible values of SMSC_AT parameters are presented in table 9. The field is optional, its use depends on the SMSC_AL parameter value (if SMSC_AL> 0, the field is used)
- SMSC_A – SMSC address. Each decimal figure of the number is presented in the form of 4 bits (lower 4 bits – a higher order digit, upper 4 bits – a lower order digit); provided that if the number of the digits in the number is odd, bits 4 to 7 of the last byte of the number are set to 0xF (1111b). The parameter is optional, its use depends on the SMSC_AL parameter value. In case the SMSC_A parameter is not used, SMSC from the SIM card will be used
- TP_MTI– Message Type Indicator (should contain binary value 01)
- TP_RD – (Reject Duplicates) the field defines whether SMSC should accept the given message for processing if a previous non-processed message sent from the given number with the same TP_MR field value and the same number of the recipient in the TP_DA field exist
- TP_VPF– (Validity Period Format) TP_VP parameter format. Possible values of the TP_VPF parameter are presented in Table 8
- TP_SRR– (Status Report Request) the field defines the need for sending a confirmation from SMSC to the given message (If the bit value is 1 , confirmation is required)
- TP_UDHI – (User Data Header Indicator). The field defines whether the header of user data TP_UD_HEADER is to be transmitted (if the field value is 1, the header is present).
- TP_RP – (Reply Path). The field defines whether the RP field is present in the message
- TP_MR – Message Identifier (increases by 1 at each dispatch of a new message)
- TP_DA_L – Length of useful data of the recipient address (the quantity of symbols in recipient address). For example, if address is “79991234567” the TP_DA_L=0Bh ;
- TP_DA_T – Type of the recipient address format. Possible values of the TP_DA_T and SMSC_AT parameters are shown in table 9
- TP_DA – Address of the recipient. The number is encoded using the same rules as in the SMSC_A parameter.
- TP_PID – Protocol identifier (should contain value 00);
- TP_DCS– Type of data encoding (should contain value 0x04, defining 8 bit encoding of the message, absence of compression)
- TP_VP – Time of validity of the message. The format of the given parameter is defined by the value from table 8. The parameter is optional. Its presence and size depend on the TP_VPF field value.

- TP_UDL – Length of data of the message from the TP_DL field, in bytes for the 8 bit encoding used
- TP_UD – Directly transferred user data. The field format and corresponding TP_UDHI field values are presented in Table 10.

Table 8: TP_VP field format depending on the TP_VPF field value

Bit value		Description
0	0	TP_VP field not transmitted
1	0	TP_VP field has a «relative time» format and 1 byte length
0	1	TP_VP field has an «extended time» format and 7 byte length
1	1	TP_VP field has an «absolute time» format and 7 byte length

Table 9: TP_DA_T and SMSC_AT field format (address type)

bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0	Bytes
1	TON			NPI				1

- TON – Type Of Number. TON can have the following values:

  000 – unknown
  001 – international
  010 – national
  011 – special network-defined number
  100 – subscriber number
  101 – alphanumeric (codes according to [2] with7-bit default encoding)
  110 – short
  111 – reserved

- NPI – Numeric Plan Identification (applicable for TON = 000,001,010). NPI can have the following values:

  0000 – unknown
  0001 – ISDN telephony numeric plan
  0011 – data transmission numeric plan
  0100 – cable

1000 – national
1001 – private
1111 – reserved

Table 10: TP_UD field format

bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0	Type	Bytes
LUDH (Length of User Data Header)								O	1
IEI «A» (Information-Element-Identifier «A»)								O	1
LIE «A» (Length of Information-Element «A»)								O	1
IED «A» (Information-Element-Data of «A»)								O	1 … n
IEI «B» (Information-Element-Identifier «B»)								O	1
LIE «B» (Length of Information-Element «B»)								O	1
IED «B» (Information-Element-Data of «B»)								O	1 … n
IEI «N» (Information-Element-Identifier «N»)								O	1
LIE «N» (Length of Information-Element «N»)								O	1
IED «N» (Information-Element-Data of «N»)								O	1 … n
UD (User Data)								M	1…140

- LUDH – Length of the user data header in bytes, the field size not taken into account
- IEI A, IEI B , IEI N are A, B и N information element identifier, correspondingly, which defines the type of the information element, and can have the following values (in the hexadecimal system):

00 = part of a concatenated SMS message
01 = indicator of a special SMS message
02 = reserved
03 = not used
04 – 7F = reserved
80 – 9F = SME special use
A0 – BF = reserved
C0 – DF = SC special use

E0 – FF = reserved

LIE A, LIE B , LIE N are parameters defining the data size of the A, B и N information elements, correspondingly, in bytes, the field size not taken into account

- IED A, IED B , IED N are data of information elements A, B и N, correspondingly
- UD is user data. The size of this field is defined by the presence of the PT_UD_HEADER user data header consisting of the LUDH, IEI, LIE, IED fields. If the header is not transmitted, the size equals the TP_UDL field value from Table 6. If the header is transmitted, the field size is calculated as a difference of (TP_UDL – LUDH -1)

If the information element identifier of the TP_UD_HEADER user data header equals 00, the IED field structure will be as shown in Table 11.

Table 11: Data field format of the information element characterizing the part of a concatenated SMS message

bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0	Type	Size, bytes
CSMRN (Concatenated Short Message Reference Number)								M	1
MNSM (Maximum Number of Short Messages)								M	1
SNCSM (Sequence Number of Current Short Message)								M	1

- CSMRN is the concatenated SMS message number. Value is to be the same for all parts of a long SMS message.
- MNSM is the total number of messages to generate a long SMS. Values are to be in the range of 1 to 255.
- SNCSM is the number of the transmitted part of a long SMS message. Incremented during dispatch of each new part of a long message. Value is to be in the range of 1 to 255. If the field value exceeds that of the MNSM field or equals zero, the recipient shall ignore the entire information element.

Incoming SMS has the format SMS-DELIVER [3] and 8-bit encoding is used. The SMS message format SMS-DELIVER in the PDU mode is given in Table 12 and uses the structure described in p. 9.2.2.1 [3].

Table 12: Format of incoming SMS using the PDU mode (SMS-DELIVER)

bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0	Type	Size, bytes
SMSC_AL (SMSC Address Length)								M	1
SMSC_AT (SMSC Address Type)								O	0,1
SMSC_A (SMSC Address)								O	0,6
TP_RP	TP_UDHI	TP_SRI	-		TP_MMS	TP_MTI		M	1
TP_OA_L (Originating Address Length)								M	1
TP_OA_T (Originating Address Type)								M	1
TP_OA (Originating Address)								M	0-10
TP_PID (Protocol Identifier)								M	1
TP_DCS (Data Coding Schema)								M	1
TP_SCTS (SMSC Time Stamp)								M	7
TP_UDL (User Data Length)								M	1
TP_UD (User Data)								O	0…140

Description of parameters included in SMS (SMS_DELIVER) in the PDU mode:

- SMSC_AL – Length of useful data of the SMSC address in octets plus 1 octet of SMSC_AT
- SMSC_AT – Type of the SMSC address format. The possible values of SMSC_AT parameters are presented in table 9. The field is optional, its use depends on the SMSC_AL parameter value (if SMSC_AL> 0, the field is used)
- SMSC_A – SMSC address. Each decimal figure of the number is presented in the form of 4 bits (lower 4 bits – a higher order digit, upper 4 bits – a lower order digit); provided that if the number of the digits in the number is odd, bits 4 to 7 of the last byte of the number are set to 0xF (1111b). The parameter is optional its use depends on the SMSC_AL parameter value.
- TP_MTI– Message Type Indicator (should contain binary value 00)

- TP_MMS – (More Messages to Send) Parameter indicating whether or not there are more messages to send

  0 = more messages are waiting for delivery to this recipient
  1 = no more messages are waiting for delivery to this recipient

- TP_SRI – (Status Report Indication) parameter indicating if the status report is required by originator. Possible values are the following:

  0 = a status report will not be returned
  1 = a status report will be sent

- TP_UDHI– (User Data Header Indicator). The field defines whether the header of user data TP_UD_HEADER is to be transmitted (if the field value is 1, the header is present).
- TP_RP – (Reply Path). The field defines whether the RP field is present in the message
- TP_OA_L– Length of useful data of the originating address (the quantity of symbols in address). For example, if address is “79991234567” the TP_OA_L=0Bh ;
- TP_OA_T– Type of the originating address format. Possible values of the TP_OA_T and SMSC_AT parameters are shown in table 9
- TP_OA – Originating address. The number is encoded using the same rules as in the SMSC_A parameter.
- TP_PID – Protocol identifier (should contain value 00);
- TP_DCS– Type of data encoding (should contain value 0x04, defining 8 bit encoding of the message, absence of compression)
- TP_SCTS– Time stamp given in semi-octet representation, and represents the local time when the message arrived at the SMSC with the accuracy of a second. Format of this parameter is the same as described in Table8.
- TP_UDL – Length of data of the message from the TP_DL field, in bytes for the 8 bit encoding used
- TP_UD – Directly transferred user data. The field format and corresponding TP_UDHI field values are presented in Table 10.

9.2 COMMUNICATION INFORMATION FORMAT DESCRIPTION

When using SMS for data exchange between ST and TP, packages generated by the rules of the Transport Level protocol and the Service Support Level are placed in the TP_UD field (Table 7), provided the full size of the Protocol package may not exceed 140 bytes. In this case the authorization mechanism is not used and no confirmation of the transmitted packages is required. After successful dispatch of SMS, the information is considered delivered.

For sending SMS containing a «digital signature» the Transport Level package of the EGTS_PT_SIGNED_APPDATA type is used.

If the Protocol data package exceeds 140 bytes, the SMS message concatenation mechanism will be used as defined in item 9.2.3.24.1 [3]. The essence of this mechanism is that the user data transmitted are segmented and sent by separate SMS messages. Provided that each such message contains a special structure defining the total number of the parts of the data transmitted and the order of their assembly by the recipient. The structure used for the purpose is the TP_UD_HEADER field which contains an information element characterizing the relevant part of the concatenated SMS message. Provided that based on the size of the user data header and the maximum number of parts in a long message equaling 255, the largest possible size of a package when using 8 bit encoding can make 255 * (140-6) = 34170 bytes.

10. TEMPORAL AND QUANTITATIVE PARAMETERS OF THE TRANSPORT LEVEL PROTOCOL WHEN USING PACKAGE DATA TRANSMISSION

The temporal and quantitative parameters of the Transport Level protocol are described in Table 12.

Table 13: Temporal and quantitative parameters of the Transport Level Protocol

Name	Data type	Value range	Default value	Description
TL_RESPONSE_TO	BYTE	0 … 255	5	Interval for the package confirmation at the Transport Level, sec
TL_RESEND_ATTEMPTS	BYTE	0 … 255	3	Number of repeated attempts to send a non-confirmed package
TL_RECONNECT_TO	BYTE	0 … 255	30	Time interval after which a repeated attempt to establish a communication channel after disconnection will be made.

11. APPENDIX 1 – PROCESSING RESULT CODES

Value	Definition	Description
0	EGTS_PC_OK	Processing successful
1	EGTS_PC_IN_PROGRESS	Processing in progress (no processing result is yet known)
128	EGTS_PC_UNS_PROTOCOL	Protocol not supported
129	EGTS_PC_DECRYPT_ERROR	Decoding error
130	EGTS_PC_PROC_DENIED	Processing denied
131	EGTS_PC_INC_HEADERFORM	Incorrect header format
132	EGTS_PC_INC_DATAFORM	Incorrect data format
133	EGTS_PC_UNS_TYPE	Type not supported
134	EGTS_PC_NOTEN_PARAMS	Incorrect number of parameters
135	EGTS_PC_DBL_PROC	Repeated processing attempt
136	EGTS_PC_PROC_SRC_DENIED	Data processing from the source denied
137	EGTS_PC_HEADERCRC_ERROR	Header check sum error
138	EGTS_PC_DATACRC_ERROR	Data check sum error
139	EGTS_PC_INVDATALEN	Invalid data length
140	EGTS_PC_ROUTE_NFOUND	Route not found
141	EGTS_PC_ROUTE_CLOSED	Route closed
142	EGTS_PC_ROUTE_DENIED	Route denied
143	EGTS_PC_INVADDR	Invalid address
144	EGTS_PC_TTLEXPIRED	Data retransmission quota exceeded
145	EGTS_PC_NO_ACK	No confirmation
146	EGTS_PC_OBJ_NFOUND	Object not found
147	EGTS_PC_EVNT_NFOUND	Event not found

148	EGTS_PC_SRVC_NFOUND	Service not found
149	EGTS_PC_SRVC_DENIED	Service denied
150	EGTS_PC_SRVC_UNKN	Unknown service type
151	EGTS_PC_AUTH_DENIED	Authorization denied
152	EGTS_PC_ALREADY_EXISTS	Object already exists
153	EGTS_PC_ID_NFOUND	Identifier not found
154	EGTS_PC_INC_DATETIME	Incorrect date and time
155	EGTS_PC_IO_ERROR	Input/output error
156	EGTS_PC_NO_RES_AVAIL	No resources available
157	EGTS_PC_MODULE_FAULT	Internal module fault
158	EGTS_PC_MODULE_PWR_FLT	Module power fault
159	EGTS_PC_MODULE_PROC_FLT	Module microcontroller fault
160	EGTS_PC_MODULE_SW_FLT	Module SW fault
161	EGTS_PC_MODULE_FW_FLT	Module internal SW fault
162	EGTS_PC_MODULE_IO_FLT	Module Input/output unit fault
163	EGTS_PC_MODULE_MEM_FLT	Internal module memory fault
164	EGTS_PC_TEST_FAILED	Test failed

12. APPENDIX 2 – EXAMPLE OF IMPLEMENTATION OF CRC-16 CHECK SUM CALCULATION ALGORITHM USING C LANGUAGE

```
/*
  Name  : CRC-16 CCITT
  Poly  : 0x1021    x^16 + x^12 + x^5 + 1
  Init  : 0xFFFF
  Revert: false
  XorOut: 0x0000
  Check : 0x29B1 ("123456789")
*/
const unsigned short Crc16Table[256] = {
    0x0000, 0x1021, 0x2042, 0x3063, 0x4084, 0x50A5, 0x60C6, 0x70E7,
    0x8108, 0x9129, 0xA14A, 0xB16B, 0xC18C, 0xD1AD, 0xE1CE, 0xF1EF,
    0x1231, 0x0210, 0x3273, 0x2252, 0x52B5, 0x4294, 0x72F7, 0x62D6,
    0x9339, 0x8318, 0xB37B, 0xA35A, 0xD3BD, 0xC39C, 0xF3FF, 0xE3DE,
    0x2462, 0x3443, 0x0420, 0x1401, 0x64E6, 0x74C7, 0x44A4, 0x5485,
    0xA56A, 0xB54B, 0x8528, 0x9509, 0xE5EE, 0xF5CF, 0xC5AC, 0xD58D,
    0x3653, 0x2672, 0x1611, 0x0630, 0x76D7, 0x66F6, 0x5695, 0x46B4,
    0xB75B, 0xA77A, 0x9719, 0x8738, 0xF7DF, 0xE7FE, 0xD79D, 0xC7BC,
    0x48C4, 0x58E5, 0x6886, 0x78A7, 0x0840, 0x1861, 0x2802, 0x3823,
    0xC9CC, 0xD9ED, 0xE98E, 0xF9AF, 0x8948, 0x9969, 0xA90A, 0xB92B,
    0x5AF5, 0x4AD4, 0x7AB7, 0x6A96, 0x1A71, 0x0A50, 0x3A33, 0x2A12,
    0xDBFD, 0xCBDC, 0xFBBF, 0xEB9E, 0x9B79, 0x8B58, 0xBB3B, 0xAB1A,
    0x6CA6, 0x7C87, 0x4CE4, 0x5CC5, 0x2C22, 0x3C03, 0x0C60, 0x1C41,
    0xEDAE, 0xFD8F, 0xCDEC, 0xDDCD, 0xAD2A, 0xBD0B, 0x8D68, 0x9D49,
    0x7E97, 0x6EB6, 0x5ED5, 0x4EF4, 0x3E13, 0x2E32, 0x1E51, 0x0E70,
    0xFF9F, 0xEFBE, 0xDFDD, 0xCFFC, 0xBF1B, 0xAF3A, 0x9F59, 0x8F78,
    0x9188, 0x81A9, 0xB1CA, 0xA1EB, 0xD10C, 0xC12D, 0xF14E, 0xE16F,
    0x1080, 0x00A1, 0x30C2, 0x20E3, 0x5004, 0x4025, 0x7046, 0x6067,
    0x83B9, 0x9398, 0xA3FB, 0xB3DA, 0xC33D, 0xD31C, 0xE37F, 0xF35E,
    0x02B1, 0x1290, 0x22F3, 0x32D2, 0x4235, 0x5214, 0x6277, 0x7256,
    0xB5EA, 0xA5CB, 0x95A8, 0x8589, 0xF56E, 0xE54F, 0xD52C, 0xC50D,
```

```
    0x34E2, 0x24C3, 0x14A0, 0x0481, 0x7466, 0x6447, 0x5424, 0x4405,
    0xA7DB, 0xB7FA, 0x8799, 0x97B8, 0xE75F, 0xF77E, 0xC71D, 0xD73C,
    0x26D3, 0x36F2, 0x0691, 0x16B0, 0x6657, 0x7676, 0x4615, 0x5634,
    0xD94C, 0xC96D, 0xF90E, 0xE92F, 0x99C8, 0x89E9, 0xB98A, 0xA9AB,
    0x5844, 0x4865, 0x7806, 0x6827, 0x18C0, 0x08E1, 0x3882, 0x28A3,
    0xCB7D, 0xDB5C, 0xEB3F, 0xFB1E, 0x8BF9, 0x9BD8, 0xABBB, 0xBB9A,
    0x4A75, 0x5A54, 0x6A37, 0x7A16, 0x0AF1, 0x1AD0, 0x2AB3, 0x3A92,
    0xFD2E, 0xED0F, 0xDD6C, 0xCD4D, 0xBDAA, 0xAD8B, 0x9DE8, 0x8DC9,
    0x7C26, 0x6C07, 0x5C64, 0x4C45, 0x3CA2, 0x2C83, 0x1CE0, 0x0CC1,
    0xEF1F, 0xFF3E, 0xCF5D, 0xDF7C, 0xAF9B, 0xBFBA, 0x8FD9, 0x9FF8,
    0x6E17, 0x7E36, 0x4E55, 0x5E74, 0x2E93, 0x3EB2, 0x0ED1, 0x1EF0
};

unsigned short Crc16(unsigned char * pcBlock, unsigned short len)
{
    unsigned short crc = 0xFFFF;
     while (len--)
        crc = (crc << 8) ^ Crc16Table[(crc >> 8) ^ *pcBlock++];
    return crc;
}
```

13. APPENDIX 3 – EXAMPLE OF IMPLEMENTATION OF CRC-8 CHECK SUM CALCULATION ALGORITHM USING C LANGUAGE

```
/*
  Name  : CRC-8
  Poly  : 0x31    x^8 + x^5 + x^4 + 1
  Init  : 0xFF
  Revert: false
  XorOut: 0x00
  Check : 0xF7 ("123456789")
*/
const unsigned char CRC8Table[256] = {
    0x00, 0x31, 0x62, 0x53, 0xC4, 0xF5, 0xA6, 0x97,
    0xB9, 0x88, 0xDB, 0xEA, 0x7D, 0x4C, 0x1F, 0x2E,
    0x43, 0x72, 0x21, 0x10, 0x87, 0xB6, 0xE5, 0xD4,
    0xFA, 0xCB, 0x98, 0xA9, 0x3E, 0x0F, 0x5C, 0x6D,
    0x86, 0xB7, 0xE4, 0xD5, 0x42, 0x73, 0x20, 0x11,
    0x3F, 0x0E, 0x5D, 0x6C, 0xFB, 0xCA, 0x99, 0xA8,
    0xC5, 0xF4, 0xA7, 0x96, 0x01, 0x30, 0x63, 0x52,
    0x7C, 0x4D, 0x1E, 0x2F, 0xB8, 0x89, 0xDA, 0xEB,
    0x3D, 0x0C, 0x5F, 0x6E, 0xF9, 0xC8, 0x9B, 0xAA,
    0x84, 0xB5, 0xE6, 0xD7, 0x40, 0x71, 0x22, 0x13,
    0x7E, 0x4F, 0x1C, 0x2D, 0xBA, 0x8B, 0xD8, 0xE9,
    0xC7, 0xF6, 0xA5, 0x94, 0x03, 0x32, 0x61, 0x50,
    0xBB, 0x8A, 0xD9, 0xE8, 0x7F, 0x4E, 0x1D, 0x2C,
    0x02, 0x33, 0x60, 0x51, 0xC6, 0xF7, 0xA4, 0x95,
    0xF8, 0xC9, 0x9A, 0xAB, 0x3C, 0x0D, 0x5E, 0x6F,
    0x41, 0x70, 0x23, 0x12, 0x85, 0xB4, 0xE7, 0xD6,
    0x7A, 0x4B, 0x18, 0x29, 0xBE, 0x8F, 0xDC, 0xED,
    0xC3, 0xF2, 0xA1, 0x90, 0x07, 0x36, 0x65, 0x54,
    0x39, 0x08, 0x5B, 0x6A, 0xFD, 0xCC, 0x9F, 0xAE,
    0x80, 0xB1, 0xE2, 0xD3, 0x44, 0x75, 0x26, 0x17,
    0xFC, 0xCD, 0x9E, 0xAF, 0x38, 0x09, 0x5A, 0x6B,
```

```
    0x45, 0x74, 0x27, 0x16, 0x81, 0xB0, 0xE3, 0xD2,
    0xBF, 0x8E, 0xDD, 0xEC, 0x7B, 0x4A, 0x19, 0x28,
    0x06, 0x37, 0x64, 0x55, 0xC2, 0xF3, 0xA0, 0x91,
    0x47, 0x76, 0x25, 0x14, 0x83, 0xB2, 0xE1, 0xD0,
    0xFE, 0xCF, 0x9C, 0xAD, 0x3A, 0x0B, 0x58, 0x69,
    0x04, 0x35, 0x66, 0x57, 0xC0, 0xF1, 0xA2, 0x93,
    0xBD, 0x8C, 0xDF, 0xEE, 0x79, 0x48, 0x1B, 0x2A,
    0xC1, 0xF0, 0xA3, 0x92, 0x05, 0x34, 0x67, 0x56,
    0x78, 0x49, 0x1A, 0x2B, 0xBC, 0x8D, 0xDE, 0xEF,
    0x82, 0xB3, 0xE0, 0xD1, 0x46, 0x77, 0x24, 0x15,
    0x3B, 0x0A, 0x59, 0x68, 0xFF, 0xCE, 0x9D, 0xAC
};

unsigned char CRC8(unsigned char *lpBlock, unsigned char len)
{
    unsigned char crc = 0xFF;
    while (len--)
        crc = CRC8Table[crc ^ *lpBlock++];
    return crc;
}
```