

Active Router Transport Protocol (ARTP) Reference  
Manual  
1.0

Generated by Doxygen 1.2.15

Fri May 21 08:02:47 2004



---

# Contents

<b>1</b>	<b>Active Router Transport Protocol (ARTP) Data Structure Index</b>	<b>1</b>
1.1	Active Router Transport Protocol (ARTP) Data Structures . . . . .	1
<b>2</b>	<b>Active Router Transport Protocol (ARTP) File Index</b>	<b>3</b>
2.1	Active Router Transport Protocol (ARTP) File List . . . . .	3
<b>3</b>	<b>Active Router Transport Protocol (ARTP) Data Structure Documentation</b>	<b>5</b>
3.1	artp_dgram Struct Reference . . . . .	5
3.2	artp_receiver Union Reference . . . . .	7
3.3	control_type Struct Reference . . . . .	8
3.4	dead_zero_sessions Struct Reference . . . . .	9
3.5	ext_item Struct Reference . . . . .	10
3.6	fragment_item Struct Reference . . . . .	11
3.7	item Struct Reference . . . . .	12
3.8	payload_CTRL Struct Reference . . . . .	15
3.9	payload_DATA Struct Reference . . . . .	16
3.10	payload_type Union Reference . . . . .	17
3.11	session_item Struct Reference . . . . .	18
3.12	Tabuffers Struct Reference . . . . .	22
3.13	Tduple Struct Reference . . . . .	24
3.14	Trcvng_buffers Struct Reference . . . . .	25
3.15	Tsending_buffers Struct Reference . . . . .	27
3.16	Tsetting Struct Reference . . . . .	29
<b>4</b>	<b>Active Router Transport Protocol (ARTP) File Documentation</b>	<b>31</b>
4.1	abuffers.c File Reference . . . . .	31

---

4.2	abuffers.h File Reference . . . . .	35
4.3	artp.c File Reference . . . . .	38
4.4	artp.h File Reference . . . . .	44
4.5	config.h File Reference . . . . .	51
4.6	errors.c File Reference . . . . .	55
4.7	errors.h File Reference . . . . .	56
4.8	net.c File Reference . . . . .	59
4.9	net.h File Reference . . . . .	61
4.10	options.c File Reference . . . . .	63
4.11	options.h File Reference . . . . .	66
4.12	packet.h File Reference . . . . .	70
4.13	rbuffers.c File Reference . . . . .	72
4.14	rbuffers.h File Reference . . . . .	76
4.15	rwlocks.h File Reference . . . . .	80
4.16	sbuffers.c File Reference . . . . .	83
4.17	sbuffers.h File Reference . . . . .	89
4.18	setting.c File Reference . . . . .	95
4.19	setting.h File Reference . . . . .	98
4.20	structs.h File Reference . . . . .	100
4.21	types.h File Reference . . . . .	102

---

# Chapter 1

## Active Router Transport Protocol (ARTP) Data Structure Index

### 1.1 Active Router Transport Protocol (ARTP) Data Structures

Here are the data structures with brief descriptions:

artp_dgram	5
artp_receiver	7
control_type	8
dead_zero_sessions	9
ext_item	10
fragment_item	11
item	12
payload_CTRL	15
payload_DATA	16
payload_type	17
session_item	18
Tabuffers	22
Tdupple	24
Trcvng_buffers	25
Tsending_buffers	27
Tsetting	29



---

## Chapter 2

# Active Router Transport Protocol (ARTP) File Index

### 2.1 Active Router Transport Protocol (ARTP) File List

Here is a list of all documented files with brief descriptions:

<a href="#">abuffers.c</a>	31
<a href="#">abuffers.h</a>	35
<a href="#">artp.c</a>	38
<a href="#">artp.h</a>	44
<a href="#">config.h</a>	51
<a href="#">errors.c</a>	55
<a href="#">errors.h</a>	56
<a href="#">net.c</a>	59
<a href="#">net.h</a>	61
<a href="#">options.c</a>	63
<a href="#">options.h</a>	66
<a href="#">packet.h</a>	70
<a href="#">rbuffers.c</a>	72
<a href="#">rbuffers.h</a>	76
<a href="#">rwlocks.h</a>	80
<a href="#">sbuffers.c</a>	83
<a href="#">sbuffers.h</a>	89
<a href="#">setting.c</a>	95
<a href="#">setting.h</a>	98
<a href="#">structs.h</a>	100
<a href="#">types.h</a>	102





---

## Chapter 3

# Active Router Transport Protocol (ARTP) Data Structure Documentation

### 3.1 artp\_dgram Struct Reference

```
#include <packet.h>
```

#### Data Fields

- enum [packet\\_type](#) type
- [payload\\_type](#) payload

#### 3.1.1 Detailed Description

ARTP packet structure

#### 3.1.2 Field Documentation

##### 3.1.2.1 union [payload\\_type](#) artp\_dgram::payload

packet payload

##### 3.1.2.2 enum [packet\\_type](#) artp\_dgram::type

packet type

The documentation for this struct was generated from the following file:

---

- [packet.h](#)

## 3.2 artp\_receiver Union Reference

```
#include <net.h>
```

### Data Fields

- [sockaddr\\_in](#) [ip](#)
- [sockaddr\\_in6](#) [ip6](#)

### 3.2.1 Detailed Description

Supported address families structure. This structure is used for storing senders and receivers - it MUST contain all supported address families structures (alocated size is the maximal size of all specified structures).

### 3.2.2 Field Documentation

#### 3.2.2.1 struct `sockaddr_in` `artp_receiver::ip`

IPv4 address family

#### 3.2.2.2 struct `sockaddr_in6` `artp_receiver::ip6`

IPv6 address family

The documentation for this union was generated from the following file:

- [net.h](#)

## 3.3 control\_type Struct Reference

```
#include <packet.h>
```

### Data Fields

- enum [ctrl\\_type](#) `type`
- CTRL\_OPTID\_TYPE [optid](#)
- char \* [value](#)
- CTRL\_VALUE\_SZ\_TYPE [valuesize](#)

### 3.3.1 Detailed Description

ARTP control packet structure (its items)

### 3.3.2 Field Documentation

#### 3.3.2.1 CTRL\_OPTID\_TYPE control\_type::optid

the option identification

#### 3.3.2.2 enum [ctrl\\_type](#) control\_type::type

the option type

#### 3.3.2.3 char\* control\_type::value

the option value

#### 3.3.2.4 CTRL\_VALUE\_SZ\_TYPE control\_type::valuesize

the option value size

The documentation for this struct was generated from the following file:

- [packet.h](#)

## 3.4 dead\_zero\_sessions Struct Reference

```
#include <structs.h>
```

### Data Fields

- [SID\\_TYPE](#) `sid`
- [artp\\_receiver](#) `receiver`
- `int` `invalid`
- `dead_zero_sessions *` `next`

### 3.4.1 Detailed Description

Structure for undeliverable sessions. This structure is used to store sessions whose initial packets cannot be delivered.

### 3.4.2 Field Documentation

#### 3.4.2.1 `int` `dead_zero_sessions::invalid`

slipper sign

#### 3.4.2.2 `struct` `dead_zero_sessions*` `dead_zero_sessions::next`

next [item](#) in this structure

#### 3.4.2.3 `union` [artp\\_receiver](#) `dead_zero_sessions::receiver`

session's receiver

#### 3.4.2.4 `SID_TYPE` `dead_zero_sessions::sid`

session identification number

The documentation for this struct was generated from the following file:

- [structs.h](#)

## 3.5 ext\_item Struct Reference

### Data Fields

- [item main\\_item](#)
- [artp\\_receiver sender](#)
- `SID_TYPE sid`
- [item main\\_item](#)
- [artp\\_receiver receiver](#)

### 3.5.1 Detailed Description

Structure for buffer identified by 0. This structure is used for saving information about non-established sessions - we must remember some more information about each datagram.

### 3.5.2 Field Documentation

#### 3.5.2.1 struct [item](#) ext\_item::main\_item

generic packet structure

#### 3.5.2.2 struct [item](#) ext\_item::main\_item

generic datagram structure

#### 3.5.2.3 union [artp\\_receiver](#) ext\_item::receiver

packet's receiver

#### 3.5.2.4 union [artp\\_receiver](#) ext\_item::sender

datagram's sender

#### 3.5.2.5 `SID_TYPE` ext\_item::sid

datagram's session identifier

The documentation for this struct was generated from the following files:

- [rbuffers.c](#)
- [sbuffers.c](#)

## 3.6 `fragment_item` Struct Reference

### Data Fields

- `FRAGMENTS_TYPE` [frag\\_id](#)
- `char *` [payload](#)
- `int` [payload\\_size](#)
- `fragment_item *` [next\\_fragment](#)
- `fragment_item *` [last\\_fragment](#)

### 3.6.1 Detailed Description

Structure for saving incoming data fragments

### 3.6.2 Field Documentation

#### 3.6.2.1 `FRAGMENTS_TYPE` `fragment_item::frag_id`

fragment id (it's equal to 1 if we're saving control packet)

#### 3.6.2.2 `struct fragment_item*` `fragment_item::last_fragment`

last fragment

#### 3.6.2.3 `struct fragment_item*` `fragment_item::next_fragment`

next fragment

#### 3.6.2.4 `char*` `fragment_item::payload`

fragment payload

#### 3.6.2.5 `int` `fragment_item::payload_size`

fragment payload size

The documentation for this struct was generated from the following file:

- [rbuffers.c](#)

## 3.7 item Struct Reference

### Data Fields

- DSEQ\_TYPE [dseq](#)
- FRAGMENTS\_TYPE [act\\_frag](#)
- FRAGMENTS\_TYPE [frag\\_count](#)
- enum [packet\\_type](#) [dgram\\_type](#)
- int [invalid](#)
- [fragment\\_item](#) \* [fragments\\_start](#)
- [fragment\\_item](#) \* [fragments\\_end](#)
- item \* [next](#)
- item \* [last](#)
- char \* [packet](#)
- T[sending\\_buffers](#) \* [buffer\\_info](#)
- int [length](#)
- SEQ\_TYPE [seq](#)
- double [first\\_send\\_time](#)
- double [time\\_to\\_resend](#)
- int [retr\\_counter](#)
- item \* [next](#)
- item \* [last](#)
- item \* [next\\_to\\_resend](#)
- item \* [last\\_to\\_resend](#)

### 3.7.1 Detailed Description

Structure for saving whole datagram information.

### 3.7.2 Field Documentation

#### 3.7.2.1 FRAGMENTS\_TYPE item::act\_frag

actual fragments count (count of saved fragments from this datagram).

#### 3.7.2.2 struct T[sending\\_buffers](#)\* item::buffer\_info

the pointer to the buffer whose packet it is

#### 3.7.2.3 enum [packet\\_type](#) item::dgram\_type

datagram type



**3.7.2.4 DSEQ\_TYPE item::dseq**

data sequence number

**3.7.2.5 double item::first\_send\_time**

packet first sending time

**3.7.2.6 FRAGMENTS\_TYPE item::frag\_count**

total fragments count

**3.7.2.7 struct [fragment\\_item](#)\* item::fragments\_end**

fragments end

**3.7.2.8 struct [fragment\\_item](#)\* item::fragments\_start**

fragments start

**3.7.2.9 int item::invalid**

item validity sign

**3.7.2.10 struct item\* item::last**

last packet in buffer

**3.7.2.11 struct item\* item::last**

last item in buffer

**3.7.2.12 struct item\* item::last\_to\_resend**

last packet in structure for retransmissions

**3.7.2.13 int item::length**

packet payload length (size)

**3.7.2.14 struct item\* item::next**

next packet in buffer

**3.7.2.15 struct item\* item::next**

next item in buffer

**3.7.2.16 struct item\* item::next\_to\_resend**

next packet in structure for retransmissions

**3.7.2.17 char\* item::packet**

packet payload

**3.7.2.18 int item::retr\_counter**

packet retransmit counter

**3.7.2.19 SEQ\_TYPE item::seq**

packet sequence number

**3.7.2.20 double item::time\_to\_resend**

packet time to resend

The documentation for this struct was generated from the following files:

- [rbuffers.c](#)
- [sbuffers.c](#)

## 3.8 payload\_CTRL Struct Reference

```
#include <packet.h>
```

### Data Fields

- `int count`
- `control_type * control`

### 3.8.1 Detailed Description

Structure describing ARTP control payload.

### 3.8.2 Field Documentation

#### 3.8.2.1 `struct control_type* payload_CTRL::control`

control elements

#### 3.8.2.2 `int payload_CTRL::count`

total count of control elements

The documentation for this struct was generated from the following file:

- `packet.h`

## 3.9 payload\_DATA Struct Reference

```
#include <packet.h>
```

### Data Fields

- DSEQ\_TYPE [dseq](#)
- char \* [sigdata](#)
- SIGSZ\_TYPE [sigsz](#)
- char \* [encdata](#)
- ENCDATA\_SIZE\_TYPE [encsz](#)

### 3.9.1 Detailed Description

Structure describing ARTP data payload.

### 3.9.2 Field Documentation

#### 3.9.2.1 DSEQ\_TYPE payload\_DATA::dseq

data sequence number

#### 3.9.2.2 char\* payload\_DATA::encdata

encrypted data

#### 3.9.2.3 ENCDATA\_SIZE\_TYPE payload\_DATA::encsz

encrypted data size

#### 3.9.2.4 char\* payload\_DATA::sigdata

data signature

#### 3.9.2.5 SIGSZ\_TYPE payload\_DATA::sigsz

data signature size

The documentation for this struct was generated from the following file:

- [packet.h](#)

## 3.10 payload\_type Union Reference

```
#include <packet.h>
```

### Data Fields

- [payload\\_DATA](#) data
- [payload\\_CTRL](#) ctrl

### 3.10.1 Detailed Description

ARTP packet payload

### 3.10.2 Field Documentation

#### 3.10.2.1 struct [payload\\_CTRL](#) payload\_type::ctrl

control payload

#### 3.10.2.2 struct [payload\\_DATA](#) payload\_type::data

data payload

The documentation for this union was generated from the following file:

- [packet.h](#)

## 3.11 session\_item Struct Reference

```
#include <structs.h>
```

### Data Fields

- SID\_TYPE [session\\_sid](#)
- [artp\\_receiver](#) [session\\_receiver](#)
- SEQ\_TYPE [current\\_seq](#)
- int [buffers\\_id](#)
- enum [Tsession\\_status](#) [session\\_status](#)
- enum [Tsession\\_type](#) [session\\_type](#)
- void \* [options](#) [OPTIONS\_COUNT]
- void \* [partner\\_options](#) [OPTIONS\_COUNT]
- unsigned long int [cwnd](#)
- unsigned long int [flight](#)
- MSS\_TYPE [mss](#)
- unsigned int [max\\_acks\\_count](#)
- unsigned long int [sbuffer\\_max\\_size](#)
- unsigned long int [rbuffer\\_max\\_size](#)
- unsigned long int [rbuffer\\_red\\_limit](#)
- int [rbuffer\\_red\\_prob](#)
- double [rtt](#)
- double [srtt](#)
- double [rto](#)
- double [ts\\_delta](#)
- RETR\_TIMEOUT\_TYPE [retries\\_timeout](#)
- TS\_TYPE [expiration\\_time](#)
- double [last\\_send\\_time](#)
- unsigned int [ref\\_counter](#)
- pthread\_mutex\_t [ref\\_counter\\_mutex](#)
- pthread\_mutex\_t [session\\_mutex](#)

### 3.11.1 Detailed Description

Structure for storing session information.

### 3.11.2 Field Documentation

#### 3.11.2.1 int session\_item::buffers\_id

the identification number of session buffers

**3.11.2.2 SEQ\_TYPE session\_item::current\_seq**

next packet sequence number

**3.11.2.3 unsigned long int session\_item::cwnd**

session congestion window size

**3.11.2.4 TS\_TYPE session\_item::expiration\_time**

packets expiration time that belong to this session

**3.11.2.5 unsigned long int session\_item::flight**

the size of unacknowledged packets for this session

**3.11.2.6 double session\_item::last\_send\_time**

last link activity time

**3.11.2.7 unsigned int session\_item::max\_acks\_count**

current maximal acks count

**3.11.2.8 MSS\_TYPE session\_item::mss**

current maximal segment size

**3.11.2.9 void\* session\_item::options[OPTIONS\_COUNT]**

options which this session sends

**3.11.2.10 void\* session\_item::partner\_options[OPTIONS\_COUNT]**

options which this session receives

**3.11.2.11 unsigned long int session\_item::rbuffer\_max\_size**

current maximal receive buffer size

**3.11.2.12 unsigned long int session\_item::rbuffer\_red\_limit**

current R.E.D. limit

**3.11.2.13 int session\_item::rbuffer\_red\_prob**

current R.E.D. dropping probability

**3.11.2.14 unsigned int session\_item::ref\_counter**

session reference counter

**3.11.2.15 pthread\_mutex\_t session\_item::ref\_counter\_mutex**

mutex used for mutual exclusion of threads changing reference counter

**3.11.2.16 RETR\_TIMEOUT\_TYPE session\_item::retries\_timeout**

maximal retries timeout for session

**3.11.2.17 double session\_item::rto**

current retransmit timeout

**3.11.2.18 double session\_item::rtt**

current session round trip time

**3.11.2.19 unsigned long int session\_item::sbuffer\_max\_size**

current maximal send buffer size

**3.11.2.20 pthread\_mutex\_t session\_item::session\_mutex**

mutex used for mutual exclusion of threads changing some session critical parameters.

**3.11.2.21 union [artp\\_receiver](#) session\_item::session\_receiver**

session's receiver

**3.11.2.22 SID\_TYPE session\_item::session\_sid**

session identification number



### 3.11.2.23 enum [Tsession\\_status](#) session\_item::session\_status

session status

### 3.11.2.24 enum [Tsession\\_type](#) session\_item::session\_type

session type

### 3.11.2.25 double session\_item::srtt

current sessin smooth round trip time

### 3.11.2.26 double session\_item::ts\_delta

current difference between session's time and its partner's time

The documentation for this struct was generated from the following file:

- [structs.h](#)

## 3.12 Tabuffers Struct Reference

### Data Fields

- char [acks](#) [MAX\_ARTP\_PACKET\_SIZE]
- int [acks\\_count](#)
- double [latest\\_send\\_time](#)
- SID\_TYPE [sid](#)
- [artp\\_receiver](#) receiver
- Tabuffers \* [next](#)
- Tabuffers \* [last](#)

### 3.12.1 Detailed Description

Ack buffers' main structure.

### 3.12.2 Field Documentation

#### 3.12.2.1 char Tabuffers::acks[MAX\_ARTP\_PACKET\_SIZE]

Place for storing sequence numbers to be acknowledged.

#### 3.12.2.2 int Tabuffers::acks\_count

Actual acks (sequence numbers to be acknowledged) count

#### 3.12.2.3 struct Tabuffers\* Tabuffers::last

Last element in structure for sending ack packets (last ack buffer).

#### 3.12.2.4 double Tabuffers::latest\_send\_time

The latest time when this acknowledgement must be sent.

#### 3.12.2.5 struct Tabuffers\* Tabuffers::next

Next element in structure for sending ack packets (next ack buffer).

#### 3.12.2.6 union [artp\\_receiver](#) Tabuffers::receiver

Receiver of the session which this acknowledgement belongs to.

### 3.12.2.7 SID\_TYPE Tabuffers::sid

Identification number of the session which this acknowledgement belongs to.

The documentation for this struct was generated from the following file:

- [abuffers.c](#)

## 3.13 Tdupple Struct Reference

### Data Fields

- SEQ\_TYPE [min\\_seq](#)
- SEQ\_TYPE [max\\_seq](#)
- double [max\\_seq\\_time](#)
- Tdupple \* [last](#)
- Tdupple \* [next](#)

### 3.13.1 Detailed Description

Structure for determining duplicities. The items of this structure consist of minimal and maximal incoming sequence number (it means that all sequence numbers between this values already came). These items are joined to the list ordered by their minimal sequence number.

### 3.13.2 Field Documentation

#### 3.13.2.1 struct Tdupple\* Tdupple::last

last [item](#)

#### 3.13.2.2 SEQ\_TYPE Tdupple::max\_seq

maximal incoming seq

#### 3.13.2.3 double Tdupple::max\_seq\_time

incoming time of maximal seq

#### 3.13.2.4 SEQ\_TYPE Tdupple::min\_seq

minimal seq

#### 3.13.2.5 struct Tdupple\* Tdupple::next

next [item](#)

The documentation for this struct was generated from the following file:

- [rbuffers.c](#)

## 3.14 Trcvng\_buffers Struct Reference

### Data Fields

- `item * complete_start`
- `item * complete_end`
- `item * incomplete_start`
- `item * incomplete_end`
- `Tdupple * dupple_start`
- `Tdupple * dupple_end`
- unsigned long int `buffer_size`
- `pthread_mutex_t buffer_size_mutex`
- `pthread_mutex_t complete_start_mutex`

### 3.14.1 Detailed Description

receive buffers structure

### 3.14.2 Field Documentation

#### 3.14.2.1 unsigned long int Trcvng\_buffers::buffer\_size

counter of buffer size

#### 3.14.2.2 pthread\_mutex\_t Trcvng\_buffers::buffer\_size\_mutex

mutex used for mutual exclusion of threads manipulating with buffer size counter

#### 3.14.2.3 struct `item*` Trcvng\_buffers::complete\_end

complete buffer end

#### 3.14.2.4 struct `item*` Trcvng\_buffers::complete\_start

complete buffer start

#### 3.14.2.5 pthread\_mutex\_t Trcvng\_buffers::complete\_start\_mutex

mutex used for mutual exclusion of threads reading from complete buffer

#### 3.14.2.6 struct `Tdupple*` Trcvng\_buffers::dupple\_end

end of structure for determining duplicities

**3.14.2.7 struct [Tdupple](#)\* Trcvng\_buffers::dupple\_start**

start of structure for determining duplicities

**3.14.2.8 struct [item](#)\* Trcvng\_buffers::incomplete\_end**

incomplete buffer end

**3.14.2.9 struct [item](#)\* Trcvng\_buffers::incomplete\_start**

incomplete buffer start

The documentation for this struct was generated from the following file:

- [rbuffers.c](#)

## 3.15 Tsending\_buffers Struct Reference

### Data Fields

- [artp\\_receiver](#) receiver
- [item](#) \* sent\_start
- [item](#) \* sent\_end
- [item](#) \* to\_send
- [item](#) \* end
- unsigned long int [buffer\\_size](#)
- pthread\_mutex\_t [buffer\\_size\\_mutex](#)
- pthread\_mutex\_t [sent\\_start\\_mutex](#)
- pthread\_mutex\_t [sent\\_end\\_mutex](#)
- pthread\_mutex\_t [to\\_send\\_mutex](#)
- pthread\_mutex\_t [end\\_mutex](#)

### 3.15.1 Detailed Description

Send buffers main structure

### 3.15.2 Field Documentation

#### 3.15.2.1 unsigned long int Tsending\_buffers::buffer\_size

counter of buffer size

#### 3.15.2.2 pthread\_mutex\_t Tsending\_buffers::buffer\_size\_mutex

mutex used for mutual exclusion of threads manipulating with buffer size counter

#### 3.15.2.3 struct [item](#)\* Tsending\_buffers::end

the end of buffer for packets to be sent

#### 3.15.2.4 pthread\_mutex\_t Tsending\_buffers::end\_mutex

mutex used for mutual exclusion of threads writing to to send packets buffer end

#### 3.15.2.5 union [artp\\_receiver](#) Tsending\_buffers::receiver

session receiver

**3.15.2.6 struct [item](#)\* Tsending\_buffers::sent\_end**

buffer for sent packets (its end)

**3.15.2.7 pthread\_mutex\_t Tsending\_buffers::sent\_end\_mutex**

mutex used for mutual exclusion of threads writing to sent packets buffer end

**3.15.2.8 struct [item](#)\* Tsending\_buffers::sent\_start**

buffer for sent packets (its start)

**3.15.2.9 pthread\_mutex\_t Tsending\_buffers::sent\_start\_mutex**

mutex used for mutual exclusion of threads reading from sent packets buffer.

**3.15.2.10 struct [item](#)\* Tsending\_buffers::to\_send**

the beginning of buffer for packets to be sent

**3.15.2.11 pthread\_mutex\_t Tsending\_buffers::to\_send\_mutex**

mutex used for mutual exclusion of threads reading from to send packets buffer

The documentation for this struct was generated from the following file:

- [sbuffers.c](#)



## 3.16 Tsetting Struct Reference

```
#include <setting.h>
```

### Data Fields

- unsigned int [initial\\_mss](#)
- TS\_TYPE [initial\\_exp\\_time](#)
- double [initial\\_rto\\_time](#)
- unsigned int [default\\_retries\\_timeout](#)
- unsigned long int [initial\\_sbuffers\\_max\\_size](#)
- unsigned long int [initial\\_rbuffers\\_max\\_size](#)
- unsigned long int [initial\\_rbuffers\\_red\\_limit](#)
- int [default\\_red\\_drop\\_probability](#)
- unsigned int [default\\_max\\_acks\\_count](#)

### 3.16.1 Detailed Description

Type of a structure for saving global ARTP settings

### 3.16.2 Field Documentation

#### 3.16.2.1 unsigned int Tsetting::default\_max\_acks\_count

maximal sequence numbers count in one acknowledgement packet

#### 3.16.2.2 int Tsetting::default\_red\_drop\_probability

R.E.D. dropping probability

#### 3.16.2.3 unsigned int Tsetting::default\_retries\_timeout

retries (for retransmissions) timeout

#### 3.16.2.4 TS\_TYPE Tsetting::initial\_exp\_time

expiration time

#### 3.16.2.5 unsigned int Tsetting::initial\_mss

maximum segment size

**3.16.2.6 unsigned long int Tsetting::initial\_rbuffers\_max\_size**

maximal receiving buffer size

**3.16.2.7 unsigned long int Tsetting::initial\_rbuffers\_red\_limit**

the random early detection (R.E.D.) limit

**3.16.2.8 double Tsetting::initial\_rto\_time**

retransmit timeout

**3.16.2.9 unsigned long int Tsetting::initial\_sbuffers\_max\_size**

maximal sending buffer size

The documentation for this struct was generated from the following file:

- [setting.h](#)

---

## Chapter 4

# Active Router Transport Protocol (ARTP) File Documentation

### 4.1 abuffers.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <pthread.h>
#include "abuffers.h"
#include "config.h"
#include "net.h"
```

#### Data Structures

- struct [Tabuffers](#)

#### Functions

- int [abuffers\\_init](#) (void)
  - int [abuffers\\_create](#) (int id\_buffer, SID\_TYPE sid, struct sockaddr \*receiver)
  - int [abuffers\\_destroy](#) (int id\_buffer)
  - int [abuffers\\_add\\_seq](#) (int id\_buffer, SID\_TYPE sid, struct sockaddr \*receiver, SEQ\_TYPE seq, double latest\_send\_time, int max\_count)
  - int [abuffers\\_get\\_ack](#) (double actual\_time, SID\_TYPE \*sid, struct sockaddr \*\*receiver, char \*\*value, int \*size)
-

### 4.1.1 Detailed Description

ARTP library for ack buffers.

**Author:**

Tomas Rebok

**Date:**

2004

### 4.1.2 Function Documentation

**4.1.2.1 int abuffers\_add\_seq (int *id\_buffer*, SID\_TYPE *sid*, struct sockaddr \*  
*sender*, SEQ\_TYPE *seq*, double *latest\_send\_time*, int *maximal\_count*)**

Store incoming sequence number to relevant buffer. This function adds sequence number of incoming packet to given buffer identified by its id. If it's the first sequence number saving for this session, it inserts the pointer of this session to the right place in structure for sending ack packets (depending on its latest time to send).

**Parameters:**

*id\_buffer* the identification number of ack buffer.

*sid* the identification number of session that this sequence number relates to (used when this session isn't created).

*sender* pointer to the place where the information about actual packet's sender is stored (it's the same as above - used when this session isn't created).

*seq* the sequence number that has to be stored.

*latest\_send\_time* the latest time when the ack packet must be sent.

*maximal\_count* the maximal count of sequence numbers in one ack packet (after reaching this count the ack packet is immediately sent).

**Returns:**

zero success

**Returns:**

nonzero related error code if something failed (for further information see documentation of file `errors.h`).

**4.1.2.2 int abuffers\_create (int *id\_buffer*, SID\_TYPE *sid*, struct sockaddr \*  
*sender*)**

Create ack buffer. This function allocates place for relevant buffer and initializes its parameters.

**Parameters:**

*id\_buffer* the identification number of buffer to be created.

*sid* the identification number of session which this buffer will be created for.  
*sender* pointer to the place where the sender of received packets is stored.

**Returns:**

zero success

**Returns:**

nonzero related error code if something failed (for further information see documentation of file `errors.h`).

**4.1.2.3 int abuffers\_destroy (int *id\_buffer*)**

Destroy ack buffer. This function destroys relevant ack buffer. It unallocates place used by that buffer and makes some other clearing steps (e.g. removes this buffer from structure for sending ack packets, etc.).

**Parameters:**

*id\_buffer* identification number of buffer to be destroyed.

**Returns:**

zero success

**Returns:**

nonzero related error code if something failed (for further information see documentation of file `errors.h`).

**4.1.2.4 int abuffers\_get\_ack (double *time*, SID\_TYPE \* *sid*, struct sockaddr \*\* *receiver*, char \*\* *value*, int \* *size*)**

Get next ack packet which should be sent. This function looks whether there's any ack packet which should be sent in specified time. If there's any it returns all information about that packet and removes related buffer from structure for sending ack packets.

**Parameters:**

*time* actual time

*sid* the pointer to the place where session's identification number will be stored.

*receiver* the relevant pointer which will be moved to the place where the receiver of returned ack packet will be stored.

*value* the relevant pointer which will be moved to the place where the payload of returned ack packet will be stored.

*size* the pointer to the place where the size of ack packet payload will be stored.

**Returns:**

zero success

**Returns:**

nonzero related error code if something failed (for further information see documentation of file `errors.h`).

#### 4.1.2.5 `int abuffers_init (void)`

Initialize ack buffers. This function doesn't include any code this time. It is defined for further purposes.

**Returns:**

zero success.

**Returns:**

nonzero related error code if something failed (for further information see documentation of file `errors.h`).

## 4.2 abuffers.h File Reference

```
#include <netinet/in.h>
#include <sys/socket.h>
#include "types.h"
#include "errors.h"
```

### Defines

- #define ARTP\_ABUFFERS\_H 1

### Functions

- int [abuffers\\_init](#) (void)
- int [abuffers\\_create](#) (int id\_buffer, SID\_TYPE sid, struct sockaddr \*sender)
- int [abuffers\\_destroy](#) (int id\_buffer)
- int [abuffers\\_add\\_seq](#) (int id\_buffer, SID\_TYPE sid, struct sockaddr \*sender, SEQ\_TYPE seq, double latest\_send\_time, int maximal\_count)
- int [abuffers\\_get\\_ack](#) (double time, SID\_TYPE \*sid, struct sockaddr \*\*receiver, char \*\*value, int \*size)

### 4.2.1 Detailed Description

ARTP library for ack buffers.

**Author:**

Tomas Rebok

**Date:**

2004

### 4.2.2 Function Documentation

#### 4.2.2.1 int [abuffers\\_add\\_seq](#) (int *id\_buffer*, SID\_TYPE *sid*, struct sockaddr \**sender*, SEQ\_TYPE *seq*, double *latest\_send\_time*, int *maximal\_count*)

Store incoming sequence number to relevant buffer. This function adds sequence number of incoming packet to given buffer identified by its id. If it's the first sequence number saving for this session, it inserts the pointer of this session to the right place in structure for sending ack packets (depending on its latest time to send).

**Parameters:**

*id\_buffer* the identification number of ack buffer.

*sid* the identification number of session that this sequence number relates to (used when this session isn't created).

*sender* pointer to the place where the information about actual packet's sender is stored (it's the same as above - used when this session isn't created).

*seq* the sequence number that has to be stored.

*latest\_send\_time* the latest time when the ack packet must be sent.

*maximal\_count* the maximal count of sequence numbers in one ack packet (after reaching this count the ack packet is immediately sent).

**Returns:**

zero success

**Returns:**

nonzero related error code if something failed (for further information see documentation of file `errors.h`).

**4.2.2.2 int abuffers\_create (int *id\_buffer*, SID\_TYPE *sid*, struct sockaddr \**sender*)**

Create ack buffer. This function allocates place for relevant buffer and initializes its parameters.

**Parameters:**

*id\_buffer* the identification number of buffer to be created.

*sid* the identification number of session which this buffer will be created for.

*sender* pointer to the place where the sender of received packets is stored.

**Returns:**

zero success

**Returns:**

nonzero related error code if something failed (for further information see documentation of file `errors.h`).

**4.2.2.3 int abuffers\_destroy (int *id\_buffer*)**

Destroy ack buffer. This function destroys relevant ack buffer. It unallocates place used by that buffer and makes some other clearing steps (e.g. removes this buffer from structure for sending ack packets, etc.).

**Parameters:**

*id\_buffer* identification number of buffer to be destroyed.

**Returns:**

zero success

**Returns:**

nonzero related error code if something failed (for further information see documentation of file `errors.h`).



**4.2.2.4 int abuffers\_get\_ack (double *time*, SID\_TYPE \* *sid*, struct sockaddr \*\*  
*receiver*, char \*\* *value*, int \* *size*)**

Get next ack packet which should be sent. This function looks whether there's any ack packet which should be sent in specified time. If there's any it returns all information about that packet and removes related buffer from structure for sending ack packets.

**Parameters:**

*time* actual time

*sid* the pointer to the place where session's identification number will be stored.

*receiver* the relevant pointer which will be moved to the place where the receiver of returned ack packet will be stored.

*value* the relevant pointer which will be moved to the place where the payload of returned ack packet will be stored.

*size* the pointer to the place where the size of ack packet payload will be stored.

**Returns:**

zero success

**Returns:**

nonzero related error code if something failed (for further information see documentation of file `errors.h`).

**4.2.2.5 int abuffers\_init (void)**

Initialize ack buffers. This function doesn't include any code this time. It is defined for further purposes.

**Returns:**

zero success.

**Returns:**

nonzero related error code if something failed (for further information see documentation of file `errors.h`).

### 4.3 artp.c File Reference

```
#include <stdio.h>
#include <string.h>
#include <time.h>
#include <sys/time.h>
#include <sys/types.h>
#include <pthread.h>
#include <unistd.h>
#include <sys/wait.h>
#include <stdarg.h>
#include <stdlib.h>
#include "artp.h"
#include "structs.h"
#include "options.h"
#include "setting.h"
#include "config.h"
#include "rbuffers.h"
#include "sbuffers.h"
#include "abuffers.h"
#include "rwlocks.h"
#include "net.h"
```

#### Functions

- int [artp\\_prepare\\_connection](#) (SID\_TYPE sid, struct sockaddr \*receiver)
- int [artp\\_destroy\\_connection](#) (SID\_TYPE sid, struct sockaddr \*receiver)
- int [artp\\_send\\_dgram](#) (struct [artp\\_dgram](#) \*dgram, SID\_TYPE sid, struct sockaddr \*receiver)
- int [artp\\_free\\_dgram](#) (struct [artp\\_dgram](#) \*dgram)
- int [artp\\_receive\\_dgram](#) (SID\_TYPE sid, struct sockaddr \*sender, struct [artp\\_dgram](#) \*dgram)
- int [artp\\_receive\\_any\\_dgram](#) (SID\_TYPE \*sid, struct sockaddr \*\*sender, struct [artp\\_dgram](#) \*dgram)
- int [artp\\_get\\_undlvr\\_session](#) (SID\_TYPE \*sid, struct sockaddr \*receiver)
- int [artp\\_get\\_sid](#) (struct sockaddr \*receiver, SID\_TYPE \*sid)
- int [artp\\_set\\_session\\_options](#) (SID\_TYPE sid, struct sockaddr \*receiver, int use, enum [artp\\_session\\_options](#) option,...)

- int [artp\\_set\\_session\\_params](#) (SID\_TYPE sid, struct sockaddr \*receiver, enum [artp\\_session\\_params](#) param,...)
- int [artp\\_init](#) (int socket, char \*filename)

### 4.3.1 Detailed Description

Active router transport protocol's (ARTP) main library.

**Author:**

Tomas Rebok

**Date:**

2004

### 4.3.2 Function Documentation

#### 4.3.2.1 int artp\_destroy\_connection (SID\_TYPE sid, struct sockaddr \* receiver)

Destroy session. This function destroys previously created session (all its structures and buffers are destroyed, too). This session destroying is made like a garbage collector - each session knows the count of pointers that points to it. When this count is equal to 0, the session is deleted.

**Parameters:**

*sid* session identification number of deleted session

*receiver* the pointer to the place where destroying session's receiver is stored

**Returns:**

zero success.

**Returns:**

nonzero related error code if something failed (for further information see documentation of file `errors.h`).

#### 4.3.2.2 int artp\_free\_dgram (struct artp\_dgram \* dgram)

Free ARTP datagram. This function unallocates space taken by ARTP dgram (the space where points ARTP dgram pointers).

**Parameters:**

*dgram* the pointer to the space where ARTP dgram is stored

**Returns:**

zero success.

**Returns:**

nonzero related error code if something failed (for further information see documentation of file `errors.h`).

**4.3.2.3 int artp\_get\_sid (struct sockaddr \* receiver, SID\_TYPE \* sid)**

Get available session identification number. This function finds out available session identification number for given receiver.

**Parameters:**

*receiver* the pointer to the place where receiver address is stored

*sid* the pointer to the place where found session identification number could be stored

**Returns:**

zero success

**Returns:**

nonzero related error code if something failed (for further information see documentation of file `errors.h`).

**4.3.2.4 int artp\_get\_undlvr\_session (SID\_TYPE \* sid, struct sockaddr \* receiver)**

Get undeliverable session. This function returns the non-established session whose packet couldn't be sent.

**Parameters:**

*sid* the pointer to the place where session identification number could be stored

*receiver* the relevant pointer which will be moved to the place where the packet's receiver is stored.

**Returns:**

zero success.

**Returns:**

nonzero related error code if something failed (for further information see documentation of file `errors.h`).

**4.3.2.5 int artp\_init (int sckt, char \* filename)**

Initialize ARTP library. This function makes necessary initialization steps for proper function of ARTP protocol (starts threads, buffers and other structures initialization, etc.). It MUST be called as a first function! (Before any using of this protocol).

**Parameters:**

*sckt* socket where this protocol should listen on.

*filename* the name of a config file (or NULL if no config file is required).

**Returns:**

zero success.

**Returns:**

nonzero related error code if something failed (for further information see documentation of file `errors.h`).

**4.3.2.6 int artp\_prepare\_connection (SID\_TYPE *sid*, struct sockaddr \* *receiver*)**

Prepare new connection. This function prepares new connection to be established (all necessary structures and buffers are created). Then this new session is inserted to its right position in array of pointers to sessions.

**Parameters:**

- sid* session identification number of creating session.
- receiver* the pointer to the place where new session's receiver is stored

**Returns:**

zero success.

**Returns:**

nonzero related error code if something failed (for further information see documentation of file `errors.h`).

**4.3.2.7 int artp\_receive\_any\_dgram (SID\_TYPE \* *sid*, struct sockaddr \*\* *sender*, struct artp\_dgram \* *dgram*)**

Receive ARTP datagram from non-established sessions. This function receives ARTP dgram from non-established session.

**Parameters:**

- sid* the pointer to the place where session identification number could be stored
- sender* the relevant pointer which will be moved to the place where the dgram sender is stored.
- dgram* the pointer to the place where ARTP dgram information could be saved.

**Returns:**

zero success.

**Returns:**

nonzero related error code if something failed (for further information see documentation of file `errors.h`).

**4.3.2.8 int artp\_receive\_dgram (SID\_TYPE *sid*, struct sockaddr \* *sender*, struct artp\_dgram \* *dgram*)**

Receive ARTP datagram from established sessions. This function receives ARTP datagram from given sender (sent through given session). This session must be previously established.

**Parameters:**

- sid* session identification number
- sender* the pointer to the place where session's sender/receiver is stored

*dgram* the pointer to the place where ARTP dgram information could be saved.

**Returns:**

zero success.

**Returns:**

nonzero related error code if something failed (for further information see documentation of file `errors.h`).

**4.3.2.9 int artp\_send\_dgram (struct artp\_dgram \* dgram, SID\_TYPE sid, struct sockaddr \* receiver)**

Send ARTP datagram. This function sends ARTP datagram to specified receiver through given session. It creates packet header and copies there its payload (the payload is fragmented if necessary). Then it's stored to proper session buffer.

**Parameters:**

*dgram* the pointer to the sending datagram.

*sid* session identification number

*receiver* the pointer to the place where session receiver is stored

**Returns:**

zero success.

**Returns:**

nonzero related error code if something failed (for further information see documentation of file `errors.h`).

**4.3.2.10 int artp\_set\_session\_options (SID\_TYPE sid, struct sockaddr \* receiver, int use, enum artp\_session\_options option, ...)**

Set session options. This function sets defined session options. The option value is taken from variable options list (there's taken the first one only). By proper parameter we can say if we want to use this option or not.

**Parameters:**

*sid* the identification number of proper session

*receiver* the pointer to the place where session's receiver is stored

*use* indicates whether we have to use this option or not. (0 = do not use, 1 = use)

*option* the option identifier that we want to set

... option value

**Returns:**

zero success.

**Returns:**

nonzero related error code if something failed (for further information see documentation of file `errors.h`).

**4.3.2.11** `int artp_set_session_params (SID_TYPE sid, struct sockaddr * sender, enum artp_session_params param, ...)`

Set session parameters. This function sets defined session parameters. The parameter value is taken from variable options list (there's taken the first one only).

**Parameters:**

- sid* the identification number of proper session
- receiver* the pointer to the place where session's receiver is stored
- param* the parameter identifier that we want to set
- ... parameter value

**Returns:**

zero success.

**Returns:**

nonzero related error code if something failed (for further information see documentation of file `errors.h`).

## 4.4 artp.h File Reference

```
#include <sys/socket.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include "packet.h"
#include "options.h"
#include "errors.h"
```

### Defines

- #define **ARTP\_H** 1

### Enumerations

- enum **artp\_session\_params** { **RETRIES\_TIMEOUT**, **MSS**, **EXP\_TIME**, **MAX\_ACKS\_COUNT**, **MAX\_SBUFFER\_SIZE**, **MAX\_RBUFFER\_SIZE**, **MAX\_RBUFFER\_RED\_LIMIT**, **MAX\_RBUFFER\_RED\_PROBABILITY** }

### Functions

- int **artp\_init** (int sckt, char \*filename)
- int **artp\_prepare\_connection** (SID\_TYPE sid, struct sockaddr \*receiver)
- int **artp\_destroy\_connection** (SID\_TYPE sid, struct sockaddr \*receiver)
- int **artp\_send\_dgram** (struct **artp\_dgram** \*dgram, SID\_TYPE sid, struct sockaddr \*receiver)
- int **artp\_receive\_dgram** (SID\_TYPE sid, struct sockaddr \*sender, struct **artp\_dgram** \*dgram)
- int **artp\_receive\_any\_dgram** (SID\_TYPE \*sid, struct sockaddr \*\*sender, struct **artp\_dgram** \*dgram)
- int **artp\_free\_dgram** (struct **artp\_dgram** \*dgram)
- int **artp\_get\_undlvr\_session** (SID\_TYPE \*sid, struct sockaddr \*receiver)
- int **artp\_get\_sid** (struct sockaddr \*receiver, SID\_TYPE \*sid)
- int **artp\_set\_session\_options** (SID\_TYPE sid, struct sockaddr \*receiver, int use, enum **artp\_session\_options** option,...)
- int **artp\_set\_session\_params** (SID\_TYPE sid, struct sockaddr \*sender, enum **artp\_session\_params** param,...)

### 4.4.1 Detailed Description

Active router transport protocol's (ARTP) main library.



**Author:**

Tomas Rebok

**Date:**

2004

## 4.4.2 Enumeration Type Documentation

### 4.4.2.1 enum artp\_session\_params

Available session's parameters (which could be set by application)

**Enumeration values:**

**RETRIES\_TIMEOUT** maximal retries timeout Defines the maximal retries timeout for each packet – when acknowledgement doesn't come until this time is reached, session is said to be dead. (in seconds)

**MSS** maximal segment size Maximal size of packet that could be sent to the network. (in bytes)

**EXP\_TIME** expiration time The expiration time for all sent packets. (in microseconds)

**MAX\_ACKS\_COUNT** maximal seq. numbers count (in acks) Defines the maximal count of sequence numbers that could be sent in one ack packet.

**MAX\_SBUFFER\_SIZE** maximal send buffer size Maximal size that could be occupied by send buffer. If it's set to 0, buffer size is unlimited. (in bytes)

**MAX\_RBUFFER\_SIZE** maximal receive buffer size Maximal size that could be occupied by receive buffer. If it's set to 0, buffer size is unlimited. (in bytes)

**MAX\_RBUFFER\_RED\_LIMIT** maximal receive buffer R.E.D. limit The receive buffer size when random early detection (R.E.D.) takes place. Every incoming packet is then dropped by defined probability. If it's set to 0 and buffer size is limited, buffer's behavior is like normal tail drop buffer.

**MAX\_RBUFFER\_RED\_PROBABILITY** maximal receive buffer R.E.D. probability of packet's dropping Each incoming packet is dropped by this probability (if R.E.D. buffer is used).

## 4.4.3 Function Documentation

### 4.4.3.1 int artp\_destroy\_connection (SID\_TYPE *sid*, struct sockaddr \* *receiver*)

Destroy session. This function destroys previously created session (all its structures and buffers are destroyed, too). This session destroying is made like a garbage collector - each session knows the count of pointers that points to it. When this count is equal to 0, the session is deleted.

**Parameters:**

*sid* session identification number of deleted session

*receiver* the pointer to the place where destroying session's receiver is stored

**Returns:**

zero success.

**Returns:**

nonzero related error code if something failed (for further information see documentation of file `errors.h`).

#### 4.4.3.2 `int artp_free_dgram (struct artp_dgram * dgram)`

Free ARTP datagram. This function unallocates space taken by ARTP dgram (the space where points ARTP dgram pointers).

**Parameters:**

*dgram* the pointer to the space where ARTP dgram is stored

**Returns:**

zero success.

**Returns:**

nonzero related error code if something failed (for further information see documentation of file `errors.h`).

#### 4.4.3.3 `int artp_get_sid (struct sockaddr * receiver, SID_TYPE * sid)`

Get available session identification number. This function finds out available session identification number for given receiver.

**Parameters:**

*receiver* the pointer to the place where receiver address is stored

*sid* the pointer to the place where found session identification number could be stored

**Returns:**

zero success

**Returns:**

nonzero related error code if something failed (for further information see documentation of file `errors.h`).

#### 4.4.3.4 `int artp_get_undlvr_session (SID_TYPE * sid, struct sockaddr * receiver)`

Get undeliverable session. This function returns the non-established session whose packet couldn't be sent.

**Parameters:**

*sid* the pointer to the place where session identification number could be stored  
*receiver* the relevant pointer which will be moved to the place where the packet's receiver is stored.

**Returns:**

zero success.

**Returns:**

nonzero related error code if something failed (for further information see documentation of file `errors.h`).

**4.4.3.5 int artp\_init (int *sckt*, char \* *filename*)**

Initialize ARTP library. This function makes necessary initialization steps for proper function of ARTP protocol (starts threads, buffers and other structures initialization, etc.). It MUST be called as a first function! (Before any using of this protocol).

**Parameters:**

*sckt* socket where this protocol should listen on.  
*filename* the name of a config file (or NULL if no config file is required).

**Returns:**

zero success.

**Returns:**

nonzero related error code if something failed (for further information see documentation of file `errors.h`).

**4.4.3.6 int artp\_prepare\_connection (SID\_TYPE *sid*, struct sockaddr \* *receiver*)**

Prepare new connection. This function prepares new connection to be established (all necessary structures and buffers are created). Then this new session is inserted to its right position in array of pointers to sessions.

**Parameters:**

*sid* session identification number of creating session.  
*receiver* the pointer to the place where new session's receiver is stored

**Returns:**

zero success.

**Returns:**

nonzero related error code if something failed (for further information see documentation of file `errors.h`).

**4.4.3.7 int artp\_receive\_any\_dgram (SID\_TYPE \* sid, struct sockaddr \*\* sender, struct artp\_dgram \* dgram)**

Receive ARTP datagram from non-established sessions. This function receives ARTP dgram from non-established session.

**Parameters:**

- sid* the pointer to the place where session identification number could be stored
- sender* the relevant pointer which will be moved to the place where the dgram sender is stored.
- dgram* the pointer to the place where ARTP dgram information could be saved.

**Returns:**

zero success.

**Returns:**

nonzero related error code if something failed (for further information see documentation of file `errors.h`).

**4.4.3.8 int artp\_receive\_dgram (SID\_TYPE sid, struct sockaddr \* sender, struct artp\_dgram \* dgram)**

Receive ARTP datagram from established sessions. This function receives ARTP datagram from given sender (sent through given session). This session must be previously established.

**Parameters:**

- sid* session identification number
- sender* the pointer to the place where session's sender/receiver is stored
- dgram* the pointer to the place where ARTP dgram information could be saved.

**Returns:**

zero success.

**Returns:**

nonzero related error code if something failed (for further information see documentation of file `errors.h`).

**4.4.3.9 int artp\_send\_dgram (struct artp\_dgram \* dgram, SID\_TYPE sid, struct sockaddr \* receiver)**

Send ARTP datagram. This function sends ARTP datagram to specified receiver through given session. It creates packet header and copies there its payload (the payload is fragmented if necessary). Then it's stored to proper session buffer.

**Parameters:**

- dgram* the pointer to the sending datagram.

*sid* session identification number

*receiver* the pointer to the place where session receiver is stored

**Returns:**

zero success.

**Returns:**

nonzero related error code if something failed (for further information see documentation of file `errors.h`).

**4.4.3.10 int artp\_set\_session\_options (SID\_TYPE *sid*, struct sockaddr \* *receiver*, int *use*, enum artp\_session\_options *option*, ...)**

Set session options. This function sets defined session options. The option value is taken from variable options list (there's taken the first one only). By proper parameter we can say if we want to use this option or not.

**Parameters:**

*sid* the identification number of proper session

*receiver* the pointer to the place where session's receiver is stored

*use* indicates whether we have to use this option or not. (0 = do not use, 1 = use)

*option* the option identifier that we want to set

... option value

**Returns:**

zero success.

**Returns:**

nonzero related error code if something failed (for further information see documentation of file `errors.h`).

**4.4.3.11 int artp\_set\_session\_params (SID\_TYPE *sid*, struct sockaddr \* *sender*, enum artp\_session\_params *param*, ...)**

Set session parameters. This function sets defined session parameters. The parameter value is taken from variable options list (there's taken the first one only).

**Parameters:**

*sid* the identification number of proper session

*receiver* the pointer to the place where session's receiver is stored

*param* the parameter identifier that we want to set

... parameter value

**Returns:**

zero success.

**Returns:**

nonzero related error code if something failed (for further information see documentation of file `errors.h`).

## 4.5 config.h File Reference

### Defines

- #define **ARTP\_CONFIG\_H** 1
- #define **ARTP\_VERSION** 1
- #define **LINK\_IDLE**(rto) (2 \* (rto))
- #define **INITIAL\_WINDOW\_SIZE**(mss) (3 \* (mss))
- #define **CWND\_ACK**(mss, old\_cwnd) ((mss) \* (mss)) / (old\_cwnd)
- #define **CWND\_DOWN\_MULTPLIER** 0.7
- #define **CWND\_RETRANS**(mss, old\_cwnd)
- #define **RTO\_COMPUTE**(srtt, init\_resend\_time) (srtt > 0) ? (4 \* (srtt)) : init\_resend\_time
- #define **LATEST\_ACKS\_SEND**(srtt) ((srtt) / 2.0)
- #define **SRTT\_ALPHA** 0.875
- #define **SRTT\_COMPUTE**(rtt, old\_srtt)
- #define **TS\_DELTA\_ALPHA** 0.875
- #define **TS\_DELTA\_COMPUTE**(current\_time, sent\_time, srtt, old\_ts\_delta)
- #define **MAX\_ARTP\_PACKET\_SIZE** 2048
- #define **DUPPLE\_SEQ\_COUNT** 100000
- #define **MAX\_CONFIG\_LINE\_LENGTH** 1000

### 4.5.1 Detailed Description

ARTP's configuraton file.

**Author:**

Tomas Rebok

**Date:**

2004

### 4.5.2 Define Documentation

#### 4.5.2.1 #define **ARTP\_VERSION** 1

Constant that define the ARTP's version.

#### 4.5.2.2 #define **CWND\_ACK**(mss, old\_cwnd) ((mss) \* (mss)) / (old\_cwnd)

Congestion window change after incoming acknowledgement. This macro says how to change congestion window size when any acknowledgement came.

**Parameters:**

*mss* current maximum segment size

*old\_cwnd* old size of congestion window

**4.5.2.3 #define CWND\_DOWN\_MULTIPLIER 0.7**

Congestion window multiplier after packet loss. This constant defines multiplier when retransmission took place (packet loss was detected).

**4.5.2.4 #define CWND\_RETRANS(mss, old\_cwnd)****Value:**

```
(CWND_DOWN_MULTIPLIER * (old_cwnd) >= INITIAL_WINDOW_SIZE(mss)) \
? CWND_DOWN_MULTIPLIER * (old_cwnd) \
: INITIAL_WINDOW_SIZE(mss);
```

Congestion window change after packet loss. This macro says how to compute new congestion window size after retransmission of any packet.

**Parameters:**

*mss* current maximum segment size  
*old\_cwnd* old size of congestion window

**4.5.2.5 #define DUPPLE\_SEQ\_COUNT 100000**

Maximal count of sequence numbers in one [item](#) of structure for searching duplicities.

**4.5.2.6 #define INITIAL\_WINDOW\_SIZE(mss) (3 \* (mss))**

Initial congestion window size

**Parameters:**

*mss* actual maximum segment size

**4.5.2.7 #define LATEST\_ACKS\_SEND(srft) ((srft) / 2.0)**

Latest acknowledgement packet send time. This macro says how to compute time when given acknowledgement packet has to be sent (lately).

**Parameters:**

*srft* actual smooth round trip time

**4.5.2.8 #define LINK\_IDLE(rto) (2 \* (rto))**

Link idle computing. This macro says how to compute time when we want to decrease congestion window size when connection is idle more than computed time.

**Parameters:**

*rto* actual retransmit timeout



**4.5.2.9 #define MAX\_ARTP\_PACKET\_SIZE 2048**

Maximal incoming ARTP packet size

**4.5.2.10 #define MAX\_CONFIG\_LINE\_LENGTH 1000**

Maximal read line length from config file

**4.5.2.11 #define RTO\_COMPUTE(srtd, init\_resend\_time) (srtd > 0) ? (4 \* (srtd)) : init\_resend\_time**

Retransmit timeout computing. This macro says how to compute packets' retransmit timeout after incoming acknowledgement packet.

**Parameters:**

*srtd* actual smooth round trip time

*init\_resend\_time* initial retransmit timeout

**4.5.2.12 #define SRTT\_ALPHA 0.875**

Smooth round trip time computing constant.

**4.5.2.13 #define SRTT\_COMPUTE(rtd, old\_srtd)****Value:**

```
((old_srtd) == 0) ? rtd \  
                : (SRTT_ALPHA * (old_srtd)) + ((1 - SRTT_ALPHA) * (rtd));
```

Smooth round trip time computing. This macro says how to compute smooth round trip time when actual round trip time and old smooth round trip time are given.

**Parameters:**

*rtd* actual round trip time

*old\_srtd* old value of smooth round trip time

**4.5.2.14 #define TS\_DELTA\_ALPHA 0.875**

Time stamp delta computing constant (see below)

**4.5.2.15 #define TS\_DELTA\_COMPUTE(current\_time, sent\_time, srtt, old\_ts\_delta)****Value:**

```
((old_ts_delta) == 0) ? (current_time) - ((sent_time) + (srtt) / 2.0) \  
: (TS_DELTA_ALPHA * (old_ts_delta)) + \  
  ((1 - TS_DELTA_ALPHA) * ((current_time) \  
    - ((sent_time) + (srtt) / 2.0)))
```

Timestamp delta computing. This macro says how to compute the difference between our time and our partner's time (timestamp delta).

**Parameters:**

*current\_time* actual time

*sent\_time* sent time of given acknowledgement packet

*srtt* actual smooth round trip time

*old\_ts\_delta* old value of timestamp delta

## 4.6 errors.c File Reference

```
#include "stdlib.h"  
#include "errors.h"
```

### Functions

- char \* [artp\\_error\\_str](#) (enum [artp\\_errors](#) error\_code)

#### 4.6.1 Detailed Description

ARTP possible error codes.

**Author:**

Tomas Rebok

**Date:**

2004

#### 4.6.2 Function Documentation

##### 4.6.2.1 char\* [artp\\_error\\_str](#) (enum [artp\\_errors](#) *error\_code*)

Returns string describing given error. This function finds out relevant description for given ARTP error code and returns it as a string.

**Parameters:**

*error\_code* relevant error code

**Returns:**

NULL given error code is bad

**Returns:**

string error description

## 4.7 errors.h File Reference

### Defines

- #define **ARTP\_ERRORS\_H** 1

### Enumerations

- enum **artp\_errors** { **E\_GENERIC\_ERROR** = -1, **E\_MEMORY\_FAIL** = -2, **E\_EMPTY\_BUFFER** = -3, **E\_FULL\_BUFFER** = -4, **E\_DEAD\_SESSION** = -5, **E\_START\_THREADS** = -6, **E\_BUF\_INIT\_ERROR** = -7, **E\_OPENING\_FILE** = -8, **E\_SYNTAX\_ERROR** = -9, **E\_BAD\_VALUE** = -10, **E\_INVALID\_BUFFER\_ID** = -11, **E\_NO\_AVAIL\_SID** = -12, **E\_NONEST\_SESSION** = -13, **E\_SESSION\_EXISTS** = -14, **E\_PACKET\_NOT\_FOUND** = -15, **E\_DUPLICITY\_PACKET** = -16, **E\_INVALID\_OPTION** = -17, **E\_INVALID\_OPT\_SIZE** = -18, **E\_INVALID\_OPT\_VALUE** = -19, **E\_BAD\_OPT\_USE** = -20, **E\_SENDING\_ERROR** = -21, **E\_FULL\_CWND** = -22, **E\_INVALID\_PARAMETER** = -23, **E\_BAD\_PARAMETER\_VALUE** = -24, **E\_NO\_ACK** = -25, **E\_NULL\_PTR** = -26, **E\_UNSUPPORTED\_AF** = -27, **E\_DIFF\_ADDR** = -28, **E\_BAD\_DGRAM** = -29, **E\_BIG\_DGRAM** = -30, **E\_SMALL\_MSS** = -31, **E\_PARTNER\_MSS** = -32, **E\_BAD\_PACKET\_TYPE** = -33 }

### Functions

- char \* **artp\_error\_str** (enum **artp\_errors** error\_code)

#### 4.7.1 Detailed Description

ARTP possible error codes.

**Author:**

Tomas Rebok

**Date:**

2004

#### 4.7.2 Enumeration Type Documentation

##### 4.7.2.1 enum artp\_errors

Enumeration of possible ARTP functions error codes

**Enumeration values:**

**E\_GENERIC\_ERROR** Generic error (not specified)

**E\_MEMORY\_FAIL** Memory error (error in allocation or something else)

**E\_EMPTY\_BUFFER** Empty buffer

**E\_FULL\_BUFFER** Full buffer

**E\_DEAD\_SESSION** Session is dead (connection lost)

**E\_START\_THREADS** Error in starting ARTP necessary threads

**E\_BUF\_INIT\_ERROR** Error in buffer initialization

**E\_OPENING\_FILE** Error in opening config file

**E\_SYNTAX\_ERROR** Syntax error in config file

**E\_BAD\_VALUE** Bad value in config file

**E\_INVALID\_BUFFER\_ID** Invalid buffer identification number

**E\_NO\_AVAIL\_SID** There's no available session identification for that receiver

**E\_NONEST\_SESSION** Referring to non-established session

**E\_SESSION\_EXISTS** Wanted session already exists

**E\_PACKET\_NOT\_FOUND** Referring packet wasn't found

**E\_DUPLICITY\_PACKET** Duplicity packet

**E\_INVALID\_OPTION** Invalid option

**E\_INVALID\_OPT\_SIZE** Invalid option size

**E\_INVALID\_OPT\_VALUE** Invalid option value

**E\_BAD\_OPT\_USE** Badly specified option using

**E\_SENDING\_ERROR** Error in packet sending

**E\_FULL\_CWND** Full congestion window

**E\_INVALID\_PARAMETER** Invalid parameter

**E\_BAD\_PARAM\_VALUE** Invalid parameter value

**E\_NO\_ACK** No acknowledgement packet to send

**E\_NULL\_PTR** Given pointer points to NULL

**E\_UNSUPPORTED\_AF** Unsupported address family

**E\_DIFF\_ADDR** Different addresses

**E\_BAD\_DGRAM** Given datagram is bad

**E\_BIG\_DGRAM** Given datagram is too big

**E\_SMALL\_MSS** Set MSS is too small - packet cannot be sent

**E\_PARTNER\_MSS** Our partner wishes smaller MSS than we want

**E\_BAD\_PACKET\_TYPE** Badly specified packet type

### 4.7.3 Function Documentation

#### 4.7.3.1 `char* artp_error_str (enum artp\_errors error_code)`

Returns string describing given error. This function finds out relevant description for given ARTP error code and returns it as a string.

**Parameters:**

*error\_code* relevant error code

**Returns:**

NULL given error code is bad

**Returns:**

string error description

## 4.8 net.c File Reference

```
#include <string.h>
#include "net.h"
#include "errors.h"
```

### Functions

- int [rcvrcmp](#) (struct sockaddr \*r1, struct sockaddr \*r2)
- int [rcvrcpy](#) (struct sockaddr \*dest\_rcvr, struct sockaddr \*source\_rcvr)
- int [rcvrsvz](#) (struct sockaddr \*rcvr)

### 4.8.1 Detailed Description

ARTP socket address manipulating functions.

**Author:**

Tomas Rebok

**Date:**

2004

### 4.8.2 Function Documentation

#### 4.8.2.1 int rcvrcmp (struct sockaddr \* r1, struct sockaddr \* r2)

Compare two socket addresses. This function is used for comparing two socket addresses. Structures are compared depending on their address family.

**Parameters:**

*r1* the pointer to the place where the first address information is stored

*r2* the pointer to the place where the second address information is stored

**Returns:**

zero addresses are equal

**Returns:**

nonzero addresses are different or some error happened (for further information see documentation of file `errors.h`).

#### 4.8.2.2 int rcvrcpy (struct sockaddr \* dest\_rcvr, struct sockaddr \* source\_rcvr)

Copy socket address. This function copies socket address pointed by `source_rcvr` parameter into the place pointed by `dest_rcvr` parameter.

**Parameters:**

*source\_rcvr* the pointer to the place where the source socket address is stored.

*dest\_rcvr* the pointer to the place where the source socket address will be copied to.

**Returns:**

zero success

**Returns:**

nonzero related error code if something failed (for further information see documentation of file `errors.h`).

**4.8.2.3 int rcvrsz (struct sockaddr \* rcvr)**

Find out the size of the socket address. This function finds out the size occupied by given socket address.

**Parameters:**

*rcvr* the pointer to the place where the socket address information is stored.

*rcvr\_len* the pointer to the place where the size of given socket address will be stored.

**Returns:**

below zero related error code if something failed (for further information see documentation of file `errors.h`).

**Returns:**

above zero the size occupied by given socket address.



## 4.9 net.h File Reference

```
#include <sys/socket.h>
#include <sys/types.h>
#include <stdlib.h>
#include <netinet/in.h>
#include <arpa/inet.h>
```

### Data Structures

- union [artp\\_receiver](#)

### Defines

- #define `ARTP_NET_H 1`

### Functions

- int [rcvrcmp](#) (struct sockaddr \*r1, struct sockaddr \*r2)
- int [rcvrcpy](#) (struct sockaddr \*dest\_rcvr, struct sockaddr \*source\_rcvr)
- int [rcvrsvz](#) (struct sockaddr \*rcvr)

#### 4.9.1 Detailed Description

ARTP socket address manipulating functions.

**Author:**

Tomas Rebok

**Date:**

2004

#### 4.9.2 Function Documentation

##### 4.9.2.1 int rcvrcmp (struct sockaddr \* r1, struct sockaddr \* r2)

Compare two socket addresses. This function is used for comparing two socket addresses. Structures are compared depending on their address family.

**Parameters:**

*r1* the pointer to the place where the first address information is stored

*r2* the pointer to the place where the second address information is stored

**Returns:**

zero addresses are equal

**Returns:**

nonzero addresses are different or some error happened (for further information see documentation of file `errors.h`).

**4.9.2.2 int rcvrcpy (struct sockaddr \* dest\_rcvr, struct sockaddr \* source\_rcvr)**

Copy socket address. This function copies socket address pointed by `source_rcvr` parameter into the place pointed by `dest_rcvr` parameter.

**Parameters:**

*source\_rcvr* the pointer to the place where the source socket address is stored.

*dest\_rcvr* the pointer to the place where the source socket address will be copied to.

**Returns:**

zero success

**Returns:**

nonzero related error code if something failed (for further information see documentation of file `errors.h`).

**4.9.2.3 int rcvrsz (struct sockaddr \* rcvr)**

Find out the size of the socket address. This function finds out the size occupied by given socket address.

**Parameters:**

*rcvr* the pointer to the place where the socket address information is stored.

*rcvr\_len* the pointer to the place where the size of given socket address will be stored.

**Returns:**

below zero related error code if something failed (for further information see documentation of file `errors.h`).

**Returns:**

above zero the size occupied by given socket address.

## 4.10 options.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "options.h"
#include "config.h"
#include "errors.h"
```

### Functions

- int [options\\_init](#) ()
- int [set\\_global\\_option](#) (char \*option\_name, char \*option\_value, int use)
- int [set\\_default\\_options](#) (struct [session\\_item](#) \*session)
- int [get\\_session\\_options](#) (struct [session\\_item](#) \*session, struct [artp\\_dgram](#) \*dgram, char \*\*options, int \*size)
- int [parse\\_session\\_options](#) (struct [session\\_item](#) \*session, char \*options, int size)
- int [use\\_options](#) (struct [session\\_item](#) \*session, int use, enum [artp\\_session\\_options](#) option\_id, va\_list \*ap)

### 4.10.1 Detailed Description

definitions of ARTP options and their manipulating functions.

**Author:**

Tomas Rebok

**Date:**

2004

### 4.10.2 Function Documentation

#### 4.10.2.1 int [get\\_session\\_options](#) (struct [session\\_item](#) \* *session*, struct [artp\\_dgram](#) \* *dgram*, char \*\* *options*, int \* *size*)

Find out session options. This function finds out options which has to be sent as a part of ARTP packet. There're returned no options for non-established sessions.

**Parameters:**

*session* the pointer to the place where session information is stored.

*packet* the pointer to the place where currently send packet is stored (currently unused - for further purposes).

*options* the relevant pointer which will be moved to the place where returned options will be stored.

*size* the pointer to the place where size of returned options will be stored.

**Returns:**

zero success

**Returns:**

nonzero related error code if something failed (for further information see documentation of file `errors.h`).

#### 4.10.2.2 `int options_init ()`

Initialize options. This function initializes options - creates necessary structure and sets default options which has to be set for each session.

**Returns:**

zero success

**Returns:**

nonzero related error code if something failed (for further information see documentation of file `errors.h`).

#### 4.10.2.3 `int parse_session_options (struct session\_item * session, char * options, int size)`

Parse session options. This function is used for parsing received options and saving their values to relevant structure. Previously saved (and allocated) options which wasn't covered in parsed string will be unallocated.

**Parameters:**

*session* the pointer to the place where session information is stored.

*options* the pointer to the place where options to parse are stored.

*size* given options size.

**Returns:**

zero success

**Returns:**

nonzero related error code if something failed (for further information see documentation of file `errors.h`).

#### 4.10.2.4 `int set_default_options (struct session\_item * session)`

Set default session options. This function is used for setting defaultly set options for given session.

**Parameters:**

*session* the pointer to the place where session information is stored.

**Returns:**

zero success

**Returns:**

nonzero related error code if something failed (for further information see documentation of file `errors.h`).

**4.10.2.5 int set\_global\_option (char \* option\_name, char \* option\_value, int use)**

Set global sessions options. This function sets global sessions options as read from config file.

**Parameters:**

*option\_name* the pointer to the space where option name is stored.

*option\_value* the pointer to the space where option value is stored.

*use* indicates whether this option has to be used or not (0 = don't use, 1 = use).

**4.10.2.6 int use\_options (struct session\_item \* session, int use, enum artp\_session\_options option\_id, va\_list \* ap)**

Set options using. This function is used for setting using of defined options (and their values).

**Parameters:**

*session* the pointer to the place where session information is stored.

*use* indicates whether this option will be used or not (0 = don't use, 1 = use).

*option\_id* option identification number.

*ap* the pointer to the place where variable parameters list is stored (the option value is obtained from this list).

**Returns:**

zero success

**Returns:**

nonzero related error code if something failed (for further information see documentation of file `errors.h`).

## 4.11 options.h File Reference

```
#include <stdarg.h>
#include "structs.h"
#include "packet.h"
```

### Defines

- #define `ARTP_OPTIONS_H` 1
- #define `OPTIONS_COUNT` 3

### Enumerations

- enum `artp_session_options` { `MAX_MSS`, `MAX_DGRAM_LEN`, `RETRANS-MITS_TIMEOUT` }

### Functions

- int `options_init` ()
- int `set_global_option` (char \*option\_name, char \*option\_value, int use)
- int `set_default_options` (struct `session_item` \*session)
- int `get_session_options` (struct `session_item` \*session, struct `artp_dgram` \*dgram, char \*\*options, int \*size)
- int `parse_session_options` (struct `session_item` \*session, char \*options, int size)
- int `use_options` (struct `session_item` \*session, int use, enum `artp_session_options` option\_id, va\_list \*ap)

#### 4.11.1 Detailed Description

ARTP options definition and their manipulating functions.

**Author:**

Tomas Rebok

**Date:**

2004

#### 4.11.2 Define Documentation

##### 4.11.2.1 #define `OPTIONS_COUNT` 3

The total options count.

## 4.11.3 Enumeration Type Documentation

### 4.11.3.1 enum artp\_session\_options

The list of available options.

#### Enumeration values:

**MAX\_MSS** maximal MSS Send this option to your partner when you want to receive packets that has to be lesser than given size.

**MAX\_DGRAM\_LEN** maximal datagram size Send this option to your partner when you want to receive datagrams that has to be lesser than given size.

**RETRANSMITS\_TIMEOUT** retransmits' timeout This option is used for receiver's determining too old information in his packets history (history used for determining duplicities).

NOTE: If you're sure that you will send less than  $2^{32}$  packets you may not use this option.

## 4.11.4 Function Documentation

### 4.11.4.1 int get\_session\_options (struct session\_item \* session, struct artp\_dgram \* dgram, char \*\* options, int \* size)

Find out session options. This function finds out options which has to be sent as a part of ARTP packet. There're returned no options for non-established sessions.

#### Parameters:

*session* the pointer to the place where session information is stored.

*packet* the pointer to the place where currently send packet is stored (currently unused - for further purposes).

*options* the relevant pointer which will be moved to the place where returned options will be stored.

*size* the pointer to the place where size of returned options will be stored.

#### Returns:

zero success

#### Returns:

nonzero related error code if something failed (for further information see documentation of file `errors.h`).

### 4.11.4.2 int options\_init ()

Initialize options. This function initializes options - creates necessary structure and sets default options which has to be set for each session.

#### Returns:

zero success

**Returns:**

nonzero related error code if something failed (for further information see documentation of file `errors.h`).

**4.11.4.3 int parse\_session\_options (struct session\_item \* session, char \* options, int size)**

Parse session options. This function is used for parsing received options and saving their values to relevant structure. Previously saved (and allocated) options which wasn't covered in parsed string will be unallocated.

**Parameters:**

*session* the pointer to the place where session information is stored.

*options* the pointer to the place where options to parse are stored.

*size* given options size.

**Returns:**

zero success

**Returns:**

nonzero related error code if something failed (for further information see documentation of file `errors.h`).

**4.11.4.4 int set\_default\_options (struct session\_item \* session)**

Set default session options. This function is used for setting defaultly set options for given session.

**Parameters:**

*session* the pointer to the place where session information is stored.

**Returns:**

zero success

**Returns:**

nonzero related error code if something failed (for further information see documentation of file `errors.h`).

**4.11.4.5 int set\_global\_option (char \* option\_name, char \* option\_value, int use)**

Set global sessions options. This function sets global sessions options as read from config file.

**Parameters:**

*option\_name* the pointer to the space where option name is stored.

*option\_value* the pointer to the space where option value is stored.

*use* indicates whether this option has to be used or not (0 = don't use, 1 = use).



**4.11.4.6 int use\_options (struct [session\\_item](#) \* session, int use, enum [artp\\_session\\_options](#) option\_id, va\_list \* ap)**

Set options using. This function is used for setting using of defined options (and their values).

**Parameters:**

*session* the pointer to the place where session information is stored.

*use* indicates whether this option will be used or not (0 = don't use, 1 = use).

*option\_id* option identification number.

*ap* the pointer to the place where variable parameters list is stored (the option value is obtained from this list).

**Returns:**

zero success

**Returns:**

nonzero related error code if something failed (for further information see documentation of file `errors.h`).

## 4.12 packet.h File Reference

```
#include "types.h"
```

### Data Structures

- struct [artp\\_dgram](#)
- struct [control\\_type](#)
- struct [payload\\_CTRL](#)
- struct [payload\\_DATA](#)
- union [payload\\_type](#)

### Defines

- #define [ARTP\\_PACKET\\_H](#) 1

### Enumerations

- enum [packet\\_type](#) { [DATA](#), [CTRL](#), [ACK](#) }
- enum [ctrl\\_type](#) { [REQUEST](#), [REPLY](#) }

#### 4.12.1 Detailed Description

ARTP packet structure definition.

**Author:**

Tomas Rebok

**Date:**

2004

#### 4.12.2 Enumeration Type Documentation

##### 4.12.2.1 enum ctrl\_type

ARTP control packet possible types.

**Enumeration values:**

**REQUEST** request packet

**REPLY** reply packet

#### 4.12.2.2 enum packet\_type

ARTP packet possible types.

**Enumeration values:**

**DATA** data packet

**CTRL** control packet

**ACK** acknowledgement packet

## 4.13 rbuffers.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <pthread.h>
#include "rbuffers.h"
#include "setting.h"
#include "net.h"
#include "errors.h"
#include "config.h"
```

### Data Structures

- struct [ext\\_item](#)
- struct [fragment\\_item](#)
- struct [item](#)
- struct [Tdupple](#)
- struct [Trcvng\\_buffers](#)

### Functions

- int [rbuffers\\_init](#) (void)
- int [rbuffers\\_create](#) (int id\_buffer)
- int [rbuffers\\_destroy](#) (int id\_buffer)
- int [rbuffers\\_get\\_size](#) (int id\_buffer, unsigned long int \*buffer\_size)
- int [rbuffers\\_add\\_packet](#) (int id\_buffer, char \*value, int payload\_size, struct sock-addr \*sender, SID\_TYPE sid, enum [packet\\_type](#) type, SEQ\_TYPE seq, DSEQ-  
TYPE dseq, FRAGMENTS\_TYPE frag\_id, FRAGMENTS\_TYPE frag\_count, double current\_time, unsigned int retransmits\_timeout)
- int [rbuffers\\_get\\_dgram](#) (int id\_buffer, char \*\*value, int \*payload\_size, enum [packet\\_type](#) \*type, DSEQ\_TYPE \*dseq, struct sockaddr \*\*sender, SID\_TYPE \*sid)

### 4.13.1 Detailed Description

ARTP library for receive buffers.

**Author:**

Tomas Rebok

**Date:**

2004

## 4.13.2 Function Documentation

**4.13.2.1** `int rbuffers_add_packet (int id_buffer, char * value, int payload_size, struct sockaddr * sender, SID_TYPE sid, enum packet_type type, SEQ_TYPE seq, DSEQ_TYPE dseq, FRAGMENTS_TYPE frag_id, FRAGMENTS_TYPE frag_count, double current_time, unsigned int retransmits_timeout)`

Add incoming packet payload into receive buffer. This function adds packet payload into receive buffer. If the whole datagram consists of 1 fragment, it's immediately saved into complete datagrams buffer. If that datagram consists of more than one fragment, the packet is saved into incomplete datagrams buffer. After coming of the last fragment the whole datagram is moved to complete datagrams buffer.

### Parameters:

*id\_buffer* identification number of receive buffer.

*value* the pointer to the place where the packet payload is stored.

*payload\_size* payload size of adding packet.

*sender* the pointer to the place where the information about packet's sender is stored (used for non-established sessions).

*sid* session identification number

*type* packet type

*seq* packet sequence number

*dseq* packet data sequence (or arbitrary value if adding control packet).

*frag\_id* data datagram fragment identification number (if adding control packet, it MUST be set to 1).

*frag\_count* data datagram fragments count (if adding control packet, it MUST be set to 1).

*current\_time* current time (it's equal to packet incoming time)

*retransmits\_timeout* retransmits.timeout sent by our partner to determine old packets in structure for duplicity check. Set it to 0 if our partner doesn't send this option.

### Returns:

zero success

### Returns:

nonzero related error code if something failed (for further information see documentation of file `errors.h`).

**4.13.2.2** `int rbuffers_create (int id_buffer)`

Create receive buffer. This function allocates place for relevant buffer and initializes its parameters.

**Parameters:**

*id\_buffer* identification number of creating buffer.

**Returns:**

zero success

**Returns:**

nonzero related error code if something failed (for further information see documentation of file `errors.h`).

**4.13.2.3 int rbuffers\_destroy (int id\_buffer)**

Destroy receive buffer. This function destroys relevant receive buffer. It unallocates place used by that buffer and makes some other clearing steps (deletes all completely and incompletely received datagrams, destroys all its structures, etc.).

**Parameters:**

*id\_buffer* identification number of destroyed buffer.

**Returns:**

zero success

**Returns:**

nonzero related error code if something failed (for further information see documentation of file `errors.h`).

**4.13.2.4 int rbuffers\_get\_dgram (int id\_buffer, char \*\* value, int \* size, enum packet\_type \* type, DSEQ\_TYPE \* dseq, struct sockaddr \*\* sender, SID\_TYPE \* sid)**

Obtain completely received datagram. This functions gets one completely received datagram (if any). If that datagram consists of more than one fragment it will be assembled from all its fragments.

**Parameters:**

*id\_buffer* identification number of receive buffer.

*value* the relevant pointer which will be moved to the place where the datagram payload will be saved.

*size* the pointer to the place where the payload size could be stored.

*type* the pointer to the place where the datagram type could be stored.

*dseq* the pointer to the place where the payload data sequence number could be stored.

*sender* the relevant pointer which will be moved to the place where the datagram's sender is stored.

*sid* the pointer to the place where the session identification number could be stored.

**Returns:**

zero success

**Returns:**

nonzero related error code if something failed (for further information see documentation of file `errors.h`).

**4.13.2.5 int rbuffers\_get\_size (int *id\_buffer*, unsigned long int \* *buffer\_size*)**

Return identified receive buffer size. This function finds out receive buffer size and returns it.

**Parameters:**

*id\_buffer* identification number of receive buffer.

*buffer\_size* the pointer to the place where buffer size could be stored.

**Returns:**

zero success.

**Returns:**

nonzero related error code if something failed (for further information see documentation of file `errors.h`).

**4.13.2.6 int rbuffers\_init (void)**

Initialize receive buffers. This function doesn't include any code this time. It is defined for further purposes.

**Returns:**

zero success

**Returns:**

nonzero related error code if something failed (for further information see documentation of file `errors.h`).

## 4.14 rbuffers.h File Reference

```
#include <netinet/in.h>
#include <sys/socket.h>
#include "packet.h"
```

### Defines

- #define ARTP\_RBUFFERS\_H 1

### Functions

- int [rbuffers\\_init](#) (void)
- int [rbuffers\\_create](#) (int id\_buffer)
- int [rbuffers\\_destroy](#) (int id\_buffer)
- int [rbuffers\\_get\\_size](#) (int id\_buffer, unsigned long int \*buffer\_size)
- int [rbuffers\\_add\\_packet](#) (int id\_buffer, char \*value, int payload\_size, struct sockaddr \*sender, SID\_TYPE sid, enum [packet\\_type](#) type, SEQ\_TYPE seq, DSEQ\_TYPE dseq, FRAGMENTS\_TYPE frag\_id, FRAGMENTS\_TYPE frag\_count, double current\_time, unsigned int retransmits\_timeout)
- int [rbuffers\\_get\\_dgram](#) (int id\_buffer, char \*\*value, int \*size, enum [packet\\_type](#) \*type, DSEQ\_TYPE \*dseq, struct sockaddr \*\*sender, SID\_TYPE \*sid)

### 4.14.1 Detailed Description

ARTP library for receive buffers.

**Author:**

Tomas Rebok

**Date:**

2004

### 4.14.2 Function Documentation

**4.14.2.1** int [rbuffers\\_add\\_packet](#) (int *id\_buffer*, char \* *value*, int *payload\_size*, struct sockaddr \* *sender*, SID\_TYPE *sid*, enum [packet\\_type](#) *type*, SEQ\_TYPE *seq*, DSEQ\_TYPE *dseq*, FRAGMENTS\_TYPE *frag\_id*, FRAGMENTS\_TYPE *frag\_count*, double *current\_time*, unsigned int *retransmits\_timeout*)

Add incoming packet payload into receive buffer. This function adds packet payload into receive buffer. If the whole datagram consists of 1 fragment, it's immediately saved into complete datagrams buffer. If that datagram consists of more than one fragment, the packet is saved into incomplete datagrams buffer. After coming of the last fragment the whole datagram is moved to complete datagrams buffer.



**Parameters:**

- id\_buffer* identification number of receive buffer.
- value* the pointer to the place where the packet payload is stored.
- payload\_size* payload size of adding packet.
- sender* the pointer to the place where the information about packet's sender is stored (used for non-established sessions).
- sid* session identification number
- type* packet type
- seq* packet sequence number
- dseq* packet data sequence (or arbitrary value if adding control packet).
- frag\_id* data datagram fragment identification number (if adding control packet, it MUST be set to 1).
- frag\_count* data datagram fragments count (if adding control packet, it MUST be set to 1).
- current\_time* current time (it's equal to packet incoming time)
- retransmits\_timeout* retransmits\_timeout sent by our partner to determine old packets in structure for duplicity check. Set it to 0 if our partner doesn't send this option.

**Returns:**

zero success

**Returns:**

nonzero related error code if something failed (for further information see documentation of file `errors.h`).

**4.14.2.2 int rbuffers\_create (int id\_buffer)**

Create receive buffer. This function allocates place for relevant buffer and initializes its parameters.

**Parameters:**

- id\_buffer* identification number of creating buffer.

**Returns:**

zero success

**Returns:**

nonzero related error code if something failed (for further information see documentation of file `errors.h`).

#### 4.14.2.3 `int rbuffers_destroy (int id_buffer)`

Destroy receive buffer. This function destroys relevant receive buffer. It unallocates place used by that buffer and makes some other clearing steps (deletes all completely and incompletely received datagrams, destroys all its structures, etc.).

**Parameters:**

*id\_buffer* identification number of destroyed buffer.

**Returns:**

zero success

**Returns:**

nonzero related error code if something failed (for further information see documentation of file `errors.h`).

#### 4.14.2.4 `int rbuffers_get_dgram (int id_buffer, char ** value, int * size, enum packet_type * type, DSEQ_TYPE * dseq, struct sockaddr ** sender, SID_TYPE * sid)`

Obtain completely received datagram. This functions gets one completely received datagram (if any). If that datagram consists of more than one fragment it will be assembled from all its fragments.

**Parameters:**

*id\_buffer* identification number of receive buffer.

*value* the relevant pointer which will be moved to the place where the datagram payload will be saved.

*size* the pointer to the place where the payload size could be stored.

*type* the pointer to the place where the datagram type could be stored.

*dseq* the pointer to the place where the payload data sequence number could be stored.

*sender* the relevant pointer which will be moved to the place where the datagram's sender is stored.

*sid* the pointer to the place where the session identification number could be stored.

**Returns:**

zero success

**Returns:**

nonzero related error code if something failed (for further information see documentation of file `errors.h`).

#### 4.14.2.5 int rbuffers\_get\_size (int *id\_buffer*, unsigned long int \* *buffer\_size*)

Return identified receive buffer size. This function finds out receive buffer size and returns it.

**Parameters:**

*id\_buffer* identification number of receive buffer.

*buffer\_size* the pointer to the place where buffer size could be stored.

**Returns:**

zero success.

**Returns:**

nonzero related error code if something failed (for further information see documentation of file `errors.h`).

#### 4.14.2.6 int rbuffers\_init (void)

Initialize receive buffers. This function doesn't include any code this time. It is defined for further purposes.

**Returns:**

zero success

**Returns:**

nonzero related error code if something failed (for further information see documentation of file `errors.h`).

## 4.15 rwlocks.h File Reference

### Defines

- #define **ARTP\_RWLOCKS\_H** 1
- #define **RW\_INIT**(x, y, z, wsem, rsem, wcount, rcount)
- #define **READERS\_LOCK**(x, y, z, wsem, rsem, wcount, rcount)
- #define **READERS\_UNLOCK**(x, y, z, wsem, rsem, wcount, rcount)
- #define **WRITERS\_LOCK**(x, y, z, wsem, rsem, wcount, rcount)
- #define **WRITERS\_UNLOCK**(x, y, z, wsem, rsem, wcount, rcount)

### 4.15.1 Detailed Description

Help macros for readers/writers problem.

**Author:**

Tomas Rebok

**Date:**

2004

### 4.15.2 Define Documentation

#### 4.15.2.1 #define READERS\_LOCK(x, y, z, wsem, rsem, wcount, rcount)

**Value:**

```
{\n    pthread_mutex_lock(&z);\n    pthread_mutex_lock(&rsem);\n    pthread_mutex_lock(&x);\n    (rcount)++;\n    if ((rcount) == 1)\n        pthread_mutex_lock(&wsem);\n    pthread_mutex_unlock(&x);\n    pthread_mutex_unlock(&rsem);\n    pthread_mutex_unlock(&z);\n}
```

Readers lock. This macro is used for readers. There can be more than one reader simultaneously, but first writer stops new readers till the end of the writer(s) code.

#### 4.15.2.2 #define READERS\_UNLOCK(x, y, z, wsem, rsem, wcount, rcount)

**Value:**

```
{\n    pthread_mutex_lock(&x);\n}
```

```

        (rcount)--;\
        if ((rcount) == 0)\
            pthread_mutex_unlock(&wsem);\
        pthread_mutex_unlock(&x);\
    }

```

Readers unlock. This macro unlocks previously locked readers lock. It decreases the count of readers, too.

#### 4.15.2.3 #define RW\_INIT(x, y, z, wsem, rsem, wcount, rcount)

**Value:**

```

{\
    if ((pthread_mutex_init(&x, NULL) != 0)\
        || (pthread_mutex_init(&y, NULL) != 0)\
        || (pthread_mutex_init(&z, NULL) != 0)\
        || (pthread_mutex_init(&wsem, NULL) != 0)\
        || (pthread_mutex_init(&rsem, NULL) != 0))\
        return E_MEMORY_FAIL;\
    wcount = 0;\
    rcount = 0;\
}

```

Initialize all necessary structures. This macro initializes all necessary mutexes and counters.

#### 4.15.2.4 #define WRITERS\_LOCK(x, y, z, wsem, rsem, wcount, rcount)

**Value:**

```

{\
    pthread_mutex_lock(&y);\
    (wcount)++;\
    if ((wcount) == 1)\
        pthread_mutex_lock(&rsem);\
    pthread_mutex_unlock(&y);\
    pthread_mutex_lock(&wsem);\
}

```

Writers lock. This macro is used for writers to lock. There can be only one writer at the moment, no readers are accepted, too. Waiting writer stops all new readers till its end.

#### 4.15.2.5 #define WRITERS\_UNLOCK(x, y, z, wsem, rsem, wcount, rcount)

**Value:**

```

{\
    pthread_mutex_unlock(&wsem);\
}

```

```
pthread_mutex_lock(&y);\n    (wcount)--;\n    if ((wcount) == 0)\n        pthread_mutex_unlock(&rsem);\n    pthread_mutex_unlock(&y);\n}
```

Writers unlock. This macro unlocks previously locked writers lock. It decreases the count of writers, too.

## 4.16 sbuffers.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <pthread.h>
#include "sbuffers.h"
#include "setting.h"
#include "net.h"
#include "errors.h"
```

### Data Structures

- struct [ext\\_item](#)
- struct [item](#)
- struct [Tsending\\_buffers](#)

### Functions

- int [sbuffers\\_init](#) (void)
- int [sbuffers\\_create](#) (struct sockaddr \*receiver)
- int [sbuffers\\_destroy](#) (int id\_buffer)
- int [sbuffers\\_get\\_size](#) (int id\_buffer, unsigned long int \*buffer\_size)
- int [sbuffers\\_add\\_packet](#) (int id\_buffer, struct sockaddr \*receiver, SID\_TYPE sid, char \*value, int size, SEQ\_TYPE seq)
- int [sbuffers\\_get\\_packet](#) (int id\_buffer, char \*\*value, int \*size, struct sockaddr \*\*receiver)
- int [sbuffers\\_send\\_event](#) (int id\_buffer, char \*bitstream, int size, double sent\_time, double time\_to\_resend)
- int [sbuffers\\_resend\\_event](#) (double time\_to\_resend)
- int [sbuffers\\_ack\\_event](#) (int id\_buffer, struct sockaddr \*sender, SID\_TYPE sid, SEQ\_TYPE seq, int \*size, double \*time)
- double [sbuffers\\_get\\_top\\_rsnd\\_time](#) (void)
- int [sbuffers\\_get\\_rsnd\\_packet](#) (char \*\*value, int \*size, struct sockaddr \*\*receiver, double \*first\_send\_time)
- int [sbuffers\\_ignore\\_first\\_rsnd](#) (char \*value)

#### 4.16.1 Detailed Description

ARTP library for send buffers.

**Author:**

Tomas Rebok

**Date:**

2004

**4.16.2 Function Documentation****4.16.2.1 int sbuffers\_ack\_event (int *id\_buffer*, struct sockaddr \* *sender*, SID\_TYPE *sid*, SEQ\_TYPE *seq*, int \* *size*, double \* *time*)**

Delete packet from sent packets buffer. This function deletes packet from sent buffer after incoming acknowledgement packet. Packet is searched depending on its sequence number (for established sessions) or depending on its sequence number, sender and session identifier (for non-established sessions). It's deleted from structure for retransmissions, too. If deleted packet wasn't resent, its sent time is returned, too (for computing round trip time). Its size is returned, too.

**Parameters:**

*id\_buffer* the identification number of send buffer.

*sender* the pointer to the place where sender's address is stored.

*sid* the session identification number which this packet belongs to.

*seq* packet sequence number.

*size* the pointer to the place where deleted packet size will be stored.

*time* the pointer to the place where deleted packet sent time could be stored.

**Returns:**

zero success.

**Returns:**

nonzero related error code if something failed (for further information see documentation of file `errors.h`).

**4.16.2.2 int sbuffers\_add\_packet (int *id\_buffer*, struct sockaddr \* *receiver*, SID\_TYPE *sid*, char \* *value*, int *size*, SEQ\_TYPE *seq*)**

Add packet to send. This function adds packet which has to be sent. This adding is a bit complicated because of slipper (this is made for mutual exclusion of reading and adding threads). First of all new [item](#) is inserted to the end of complete buffer (after slipper) and its signed as a new slipper. Then all necessary information is copied to the old slipper and then it's signed (the old slipper) as a new valid [item](#).

**Parameters:**

*id\_buffer* the identification number of send buffer.

*receiver* the pointer to the place where the packet's receiver is stored. It's used for non-established sessions only. Otherwise it could be NULL.

*sid* the session identification number. It's used for non-established sessions only. Otherwise it could be any value.



*value* the pointer to the place where the packet value is stored.

*size* the size of packet payload.

*seq* the packet sequence number.

**Returns:**

zero success.

**Returns:**

nonzero related error code if something failed (for further information see documentation of file `errors.h`).

**4.16.2.3 int sbuffers\_create (struct sockaddr \* receiver)**

Create send buffer. At the beginning this function searches first free space for creating new buffer in the array of pointers to all allocated buffers. If there's any, it's used. If there isn't, the array is resized and last pointer is used. Then all slippers are made and all buffer's parameters are set to its default values.

**Parameters:**

*receiver* the pointer to the place where session receiver is stored.

**Returns:**

above or equal zero success. Creating buffer identification number is returned.

**Returns:**

below zero related error code if something failed (for further information see documentation of file `errors.h`).

**4.16.2.4 int sbuffers\_destroy (int id\_buffer)**

Destroy send buffers. This function destroys relevant send buffer. It unallocates place used by that buffer and makes some other clearing steps (deletes all sent and unsent packets, destroys all its help structures and semaphores, etc.).

**Parameters:**

*id\_buffer* the identification number of destroying buffer.

**Returns:**

zero success

**Returns:**

nonzero related error code if something failed (for further information see documentation of file `errors.h`).

#### 4.16.2.5 `int sbuffers_get_packet (int id_buffer, char ** value, int * size, struct sockaddr ** receiver)`

Get packet dedicated to send. This function returns packet which has to be sent (and its other information if necessary).

**Parameters:**

*id\_buffer* the identification number of send buffer.

*value* the relevant pointer which will be moved to the place where the packet payload will be saved.

*size* the pointer to the place where the packet size will be stored.

*receiver* the relevant pointer which will be moved to the place where the packet receiver is stored.

**Returns:**

zero success.

**Returns:**

nonzero related error code if something failed (for further information see documentation of file `errors.h`).

#### 4.16.2.6 `int sbuffers_get_rsnd_packet (char ** value, int * size, struct sockaddr ** receiver, double * first_send_time)`

Return packet to resend. This function returns packet which has to be resent (it's taken from the head of structure for retransmissions).

**Parameters:**

*value* the relevant pointer which will be moved to the place where the packet payload will be saved.

*size* the pointer to the place where the packet size will be stored.

*receiver* the relevant pointer which will be moved to the place where the packet's receiver is stored.

*first\_send\_time* the pointer to the place where the packet's first send time could be stored (used for detecting connection errors).

**Returns:**

zero success.

**Returns:**

nonzero related error code if something failed (for further information see documentation of file `errors.h`).

**4.16.2.7 int sbuffers\_get\_size (int *id\_buffer*, unsigned long int \* *buffer\_size*)**

Return identified send buffer size. This function finds out send buffer size and returns it.

**Parameters:**

*id\_buffer* the identification number of send buffer.

*buffer\_size* the pointer to the place where buffer size could be stored.

**Returns:**

zero success.

**Returns:**

nonzero related error code if something failed (for further information see documentation of file `errors.h`).

**4.16.2.8 double sbuffers\_get\_top\_rsnd\_time (void)**

Return the smallest time to resend. This function looks to the head of structure for retransmissions and returns the minimal time when some packet has to be resend (if any).

**Returns:**

above zero the minimal packet resend time.

**Returns:**

below zero related error code if something failed (for further information see documentation of file `errors.h`).

**4.16.2.9 int sbuffers\_init (void)**

Initialize send buffers. This function makes some initial steps - currently it initializes mutex used for retransmit structure.

**Returns:**

zero success

**Returns:**

nonzero related error code if something failed (for further information see documentation of file `errors.h`).

**4.16.2.10 int sbuffers\_resend\_event (double *time\_to\_resend*)**

Move last resend packet to its new position in retransmit structure. This function has to be called after retransmitting some packet. It moves that packet to its new position (depending on its next time to resend) in structure for retransmissions.

**Parameters:**

*time\_to\_resend* time when was this packet should be resend next.

**Returns:**

zero success.

**Returns:**

nonzero related error code if something failed (for further information see documentation of file `errors.h`).

**4.16.2.11 int sbuffers\_send\_event (int id\_buffer, char \* bitstream, int size, double sent\_time, double time\_to\_resend)**

Move packet from send packets buffer to sent packets buffer. This function moves defined acket from send packets buffer to sent packets buffer. This function has to be called after packet sending. If the packet is different from the one stored in buffer, it won't do any steps (this situation may appear when two threads sent the same packet simultaneously). The adding is a bit complicated again (because of threads mutual exclusion). The moved packet is inserted at the end of sent packets buffer (after the old slipper) and is signed as a slipper, too. All its values are copied to the old one (pointers are moved only). Now we have to insert this new packet into the structure for retransmissions (it's sorted by packet next retransmission time). Then the moved packet is signed as a valid [item](#).

**Parameters:**

*id\_buffer* the identification number of send buffer.

*bitstream* the pointer to the place where the moved packet value is stored.

*size* the size of moved packet payload.

*sent\_time* the time when moved packet was sent.

*time\_to\_resend* the time when this packet should be resent (if no acknowledge will come).

**Returns:**

zero success.

**Returns:**

nonzero related error code if something failed (for further information see documentation of file `errors.h`).

## 4.17 sbufers.h File Reference

```
#include <netinet/in.h>
#include <sys/socket.h>
#include "types.h"
```

### Defines

- #define **ARTP\_SBUFFERS\_H** 1

### Functions

- int [sbufers\\_init](#) (void)
- int [sbufers\\_create](#) (struct sockaddr \*receiver)
- int [sbufers\\_destroy](#) (int id\_buffer)
- int [sbufers\\_get\\_size](#) (int id\_buffer, unsigned long int \*buffer\_size)
- int [sbufers\\_add\\_packet](#) (int id\_buffer, struct sockaddr \*receiver, SID\_TYPE sid, char \*value, int size, SEQ\_TYPE seq)
- int [sbufers\\_get\\_packet](#) (int id\_buffer, char \*\*value, int \*size, struct sockaddr \*\*receiver)
- int [sbufers\\_send\\_event](#) (int id\_buffer, char \*bitstream, int size, double sent\_time, double time\_to\_resend)
- int [sbufers\\_resend\\_event](#) (double time\_to\_resend)
- int [sbufers\\_ack\\_event](#) (int id\_buffer, struct sockaddr \*sender, SID\_TYPE sid, SEQ\_TYPE seq, int \*size, double \*time)
- double [sbufers\\_get\\_top\\_rsnd\\_time](#) (void)
- int [sbufers\\_get\\_rsnd\\_packet](#) (char \*\*value, int \*size, struct sockaddr \*\*receiver, double \*first\_send\_time)
- int [sbufers\\_ignore\\_first\\_rsnd](#) ()

### 4.17.1 Detailed Description

ARTP library for send buffers.

**Author:**

Tomas Rebok

**Date:**

2004

## 4.17.2 Function Documentation

### 4.17.2.1 `int sbufs_ack_event (int id_buffer, struct sockaddr * sender, SID_TYPE sid, SEQ_TYPE seq, int * size, double * time)`

Delete packet from sent packets buffer. This function deletes packet from sent buffer after incoming acknowledgement packet. Packet is searched depending on its sequence number (for established sessions) or depending on its sequence number, sender and session identifier (for non-established sessions). It's deleted from structure for retransmissions, too. If deleted packet wasn't resent, its sent time is returned, too (for computing round trip time). Its size is returned, too.

#### Parameters:

*id\_buffer* the identification number of send buffer.

*sender* the pointer to the place where sender's address is stored.

*sid* the session identification number which this packet belongs to.

*seq* packet sequence number.

*size* the pointer to the place where deleted packet size will be stored.

*time* the pointer to the place where deleted packet sent time could be stored.

#### Returns:

zero success.

#### Returns:

nonzero related error code if something failed (for further information see documentation of file `errors.h`).

### 4.17.2.2 `int sbufs_add_packet (int id_buffer, struct sockaddr * receiver, SID_TYPE sid, char * value, int size, SEQ_TYPE seq)`

Add packet to send. This function adds packet which has to be sent. This adding is a bit complicated because of slipper (this is made for mutual exclusion of reading and adding threads). First of all new [item](#) is inserted to the end of complete buffer (after slipper) and its signed as a new slipper. Then all necessary information is copied to the old slipper and then it's signed (the old slipper) as a new valid [item](#).

#### Parameters:

*id\_buffer* the identification number of send buffer.

*receiver* the pointer to the place where the packet's receiver is stored. It's used for non-established sessions only. Otherwise it could be NULL.

*sid* the session identification number. It's used for non-established sessions only. Otherwise it could be any value.

*value* the pointer to the place where the packet value is stored.

*size* the size of packet payload.

*seq* the packet sequence number.

**Returns:**

zero success.

**Returns:**

nonzero related error code if something failed (for further information see documentation of file `errors.h`).

**4.17.2.3 int sbuffers\_create (struct sockaddr \* receiver)**

Create send buffer. At the beginning this function searches first free space for creating new buffer in the array of pointers to all allocated buffers. If there's any, it's used. If there isn't, the array is resized and last pointer is used. Then all slippers are made and all buffer's parameters are set to its default values.

**Parameters:**

*receiver* the pointer to the place where session receiver is stored.

**Returns:**

above or equal zero success. Creating buffer identification number is returned.

**Returns:**

below zero related error code if something failed (for further information see documentation of file `errors.h`).

**4.17.2.4 int sbuffers\_destroy (int id\_buffer)**

Destroy send buffers. This function destroys relevant send buffer. It unallocates place used by that buffer and makes some other clearing steps (deletes all sent and unsent packets, destroys all its help structures and semaphores, etc.).

**Parameters:**

*id\_buffer* the identification number of destroying buffer.

**Returns:**

zero success

**Returns:**

nonzero related error code if something failed (for further information see documentation of file `errors.h`).

**4.17.2.5 int sbuffers\_get\_packet (int id\_buffer, char \*\* value, int \* size, struct sockaddr \*\* receiver)**

Get packet dedicated to send. This function returns packet which has to be sent (and its other information if necessary).

**Parameters:**

*id\_buffer* the identification number of send buffer.

*value* the relevant pointer which will be moved to the place where the packet payload will be saved.

*size* the pointer to the place where the packet size will be stored.

*receiver* the relevant pointer which will be moved to the place where the packet receiver is stored.

**Returns:**

zero success.

**Returns:**

nonzero related error code if something failed (for further information see documentation of file `errors.h`).

**4.17.2.6 int sbuffers\_get\_rsnd\_packet (char \*\* value, int \* size, struct sockaddr \*\* receiver, double \* first\_send\_time)**

Return packet to resend. This function returns packet which has to be resent (it's taken from the head of structure for retransmissions).

**Parameters:**

*value* the relevant pointer which will be moved to the place where the packet payload will be saved.

*size* the pointer to the place where the packet size will be stored.

*receiver* the relevant pointer which will be moved to the place where the packet's receiver is stored.

*first\_send\_time* the pointer to the place where the packet's first send time could be stored (used for detecting connection errors).

**Returns:**

zero success.

**Returns:**

nonzero related error code if something failed (for further information see documentation of file `errors.h`).

**4.17.2.7 int sbuffers\_get\_size (int id\_buffer, unsigned long int \* buffer\_size)**

Return identified send buffer size. This function finds out send buffer size and returns it.

**Parameters:**

*id\_buffer* the identification number of send buffer.

*buffer\_size* the pointer to the place where buffer size could be stored.



**Returns:**

zero success.

**Returns:**

nonzero related error code if something failed (for further information see documentation of file `errors.h`).

**4.17.2.8 double sbuffers\_get\_top\_rsnd\_time (void)**

Return the smallest time to resend. This function looks to the head of structure for retransmissions and returns the minimal time when some packet has to be resend (if any).

**Returns:**

above zero the minimal packet resend time.

**Returns:**

below zero related error code if something failed (for further information see documentation of file `errors.h`).

**4.17.2.9 int sbuffers\_ignore\_first\_rsnd ()**

Skip the top packet in structure for retransmissions. This function skips the leading packet in structure for retransmissions. It's used when connection fail is detected - no other packets which belongs to that connection are retransmitted any more.

**Returns:**

zero success.

**Returns:**

nonzero related error code if something failed (for further information see documentation of file `errors.h`).

**4.17.2.10 int sbuffers\_init (void)**

Initialize send buffers. This function makes some initial steps - currently it initializes mutex used for retransmit structure.

**Returns:**

zero success

**Returns:**

nonzero related error code if something failed (for further information see documentation of file `errors.h`).

**4.17.2.11 int sbuffers\_resend\_event (double *time\_to\_resend*)**

Move last resend packet to its new position in retransmit structure. This function has to be called after retransmitting some packet. It moves that packet to its new position (depending on its next time to resend) in structure for retransmissions.

**Parameters:**

*time\_to\_resend* time when was this packet should be resend next.

**Returns:**

zero success.

**Returns:**

nonzero related error code if something failed (for further information see documentation of file `errors.h`).

**4.17.2.12 int sbuffers\_send\_event (int *id\_buffer*, char \* *bitstream*, int *size*, double *sent\_time*, double *time\_to\_resend*)**

Move packet from send packets buffer to sent packets buffer. This function moves defined acket from send packets buffer to sent packets buffer. This function has to be called after packet sending. If the packet is different from the one stored in buffer, it won't do any steps (this situation may appear when two threads sent the same packet simultaneously). The adding is a bit complicated again (because of threads mutual exclusion). The moved packet is inserted at the end of sent packets buffer (after the old slipper) and is signed as a slipper, too. All its values are copied to the old one (pointers are moved only). Now we have to insert this new packet into the structure for retransmissions (it's sorted by packet next retransmission time). Then the moved packet is signed as a valid [item](#).

**Parameters:**

*id\_buffer* the identification number of send buffer.

*bitstream* the pointer to the place where the moved packet value is stored.

*size* the size of moved packet payload.

*sent\_time* the time when moved packet was sent.

*time\_to\_resend* the time when this packet should be resent (if no acknowledge will come).

**Returns:**

zero success.

**Returns:**

nonzero related error code if something failed (for further information see documentation of file `errors.h`).

## 4.18 setting.c File Reference

```
#include <stdio.h>
#include <ctype.h>
#include <string.h>
#include <stdlib.h>
#include "setting.h"
#include "options.h"
#include "errors.h"
#include "config.h"
```

### Defines

- #define [INITIAL\\_MSS](#) 2048
- #define [INITIAL\\_EXP\\_TIME](#) 10000000
- #define [INITIAL\\_RTO\\_TIME](#) 3
- #define [INITIAL\\_SBUFFERS\\_MAX\\_SIZE](#) 0
- #define [INITIAL\\_RBUFFERS\\_MAX\\_SIZE](#) 0
- #define [INITIAL\\_RBUFFERS\\_RED\\_LIMIT](#) 0
- #define [DEFAULT\\_RED\\_DROP\\_PROBABILITY](#) 20
- #define [DEFAULT\\_RETRIES\\_TIMEOUT](#) 15
- #define [DEFAULT\\_MAX\\_ACKS\\_COUNT](#) 50

### Functions

- int [setting\\_set\\_defaults](#) (void)
- int [setting\\_read\\_file](#) (char \*filename)

#### 4.18.1 Detailed Description

ARTP library for reading and setting ARTP main settings.

**Author:**

Tomas Rebok

**Date:**

2004

#### 4.18.2 Define Documentation

##### 4.18.2.1 #define [DEFAULT\\_MAX\\_ACKS\\_COUNT](#) 50

initial maximal sequence numbers count in one acknowledgement packet

**4.18.2.2 #define DEFAULT\_RED\_DROP\_PROBABILITY 20**

initial R.E.D. dropping probability (in percent)

**4.18.2.3 #define DEFAULT\_RETRIES\_TIMEOUT 15**

default timeout for retransmissions (maximal time for tries) (in seconds)

**4.18.2.4 #define INITIAL\_EXP\_TIME 1000000**

Initial expiration time (in microseconds)

**4.18.2.5 #define INITIAL\_MSS 2048**

Initial maximum segment size

**4.18.2.6 #define INITIAL\_RBUFFERS\_MAX\_SIZE 0**

initial maximal receive buffers size. When set to 0, buffer size will be unlimited. (in bytes)

**4.18.2.7 #define INITIAL\_RBUFFERS\_RED\_LIMIT 0**

initial random early detection (R.E.D.) limit. This limit says when we want to applicate random packet dropping. It has no sense when maximal count is unlimited. When it's set to 0 or it's greater than maximal receive buffer size, it's not applied (buffer works as usual tail drop buffer).

**4.18.2.8 #define INITIAL\_RTO\_TIME 3**

initial retransmit timeout (in seconds)

**4.18.2.9 #define INITIAL\_SBUFFERS\_MAX\_SIZE 0**

initial maximal send buffers size. When set to 0, buffer size will be unlimited. (in bytes)

**4.18.3 Function Documentation****4.18.3.1 int setting\_read\_file (char \*filename)**

Read setting from a file. This function reads setting from given file.

**Parameters:**

*filename* the file name to be read (or NULL if no config file is required)

**Returns:**

zero success.

**Returns:**

nonzero related error code if something failed (for further information see documentation of file `errors.h`).

**4.18.3.2 int setting\_set\_defaults (void)**

Set default setting. This functions sets the default setting to the relevant structure (`global_setting`).

**Returns:**

zero success.

**Returns:**

nonzero related error code if something failed (for further information see documentation of file `errors.h`).

## 4.19 setting.h File Reference

```
#include <stdint.h>
#include "types.h"
```

### Data Structures

- struct [Tsetting](#)

### Defines

- #define `ARTP_SETTING_H 1`

### Functions

- int [setting\\_set\\_defaults](#) (void)
- int [setting\\_read\\_file](#) (char \*filename)

### Variables

- [Tsetting global\\_setting](#)

#### 4.19.1 Detailed Description

ARTP library for reading and setting ARTP settings.

**Author:**

Tomas Rebok

**Date:**

2004

#### 4.19.2 Function Documentation

##### 4.19.2.1 int [setting\\_read\\_file](#) (char \**filename*)

Read setting from a file. This function reads setting from given file.

**Parameters:**

*filename* the file name to be read (or NULL if no config file is required)

**Returns:**

zero success.

**Returns:**

nonzero related error code if something failed (for further information see documentation of file `errors.h`).

**4.19.2.2 int setting\_set\_defaults (void)**

Set default setting. This functions sets the default setting to the relevant structure (`global_setting`).

**Returns:**

zero success.

**Returns:**

nonzero related error code if something failed (for further information see documentation of file `errors.h`).

**4.19.3 Variable Documentation****4.19.3.1 struct [Tsetting](#) global\_setting**

Structure for saving global ARTP settings

## 4.20 structs.h File Reference

```
#include <sys/socket.h>
#include <sys/types.h>
#include <stdlib.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <pthread.h>
#include "net.h"
#include "options.h"
```

### Data Structures

- struct [dead\\_zero\\_sessions](#)
- struct [session\\_item](#)

### Defines

- #define `ARTP_STRUCTS_H 1`

### Enumerations

- enum `Tsession_status` { `LIVE`, `DEAD` }
- enum `Tsession_type` { `NON_EST`, `EST` }

#### 4.20.1 Detailed Description

ARTP structures for storing session information.

**Author:**

Tomas Rebok

**Date:**

2004

#### 4.20.2 Enumeration Type Documentation

##### 4.20.2.1 enum `Tsession_status`

Enumeration of available session status states.

**Enumeration values:**

`LIVE` connection lives



**DEAD** connection lost

#### 4.20.2.2 enum Tsession\_type

Enumeration of available session types

**Enumeration values:**

**NON\_EST** non-established session

**EST** established session

## 4.21 types.h File Reference

### Defines

- #define **ARTP\_TYPES\_H** 1
- #define **OPTS\_OPTID\_TYPE** uint16\_t
- #define **OPTS\_SZ\_TYPE** uint16\_t
- #define **SIGSZ\_TYPE** uint32\_t
- #define **ENCDATA\_SIZE\_TYPE** uint32\_t
- #define **DSEQ\_TYPE** uint32\_t
- #define **CTRL\_OPTID\_TYPE** uint8\_t
- #define **CTRL\_SZ\_TYPE** uint16\_t
- #define **CTRL\_VALUE\_SZ\_TYPE** int
- #define **SID\_TYPE** uint8\_t
- #define **SID\_MAX** UINT8\_MAX
- #define **SEQ\_TYPE** uint32\_t
- #define **SEQ\_MAX** UINT32\_MAX
- #define **TS\_TYPE** uint32\_t
- #define **OPTSZ\_TYPE** uint16\_t
- #define **FRAGMENTS\_TYPE** uint16\_t
- #define **MSS\_TYPE** uint16\_t
- #define **MAX\_DGRAM\_LEN\_TYPE** uint32\_t
- #define **RETR\_TIMEOUT\_TYPE** uint16\_t

### 4.21.1 Detailed Description

ARTP datagram's types.

**Author:**

Tomas Rebok

**Date:**

2004

### 4.21.2 Define Documentation

#### 4.21.2.1 #define CTRL\_OPTID\_TYPE uint8\_t

control packet option identifier type

#### 4.21.2.2 #define CTRL\_SZ\_TYPE uint16\_t

one control [item](#) size type

**4.21.2.3 #define CTRL\_VALUE\_SZ\_TYPE int**

control value size type

**4.21.2.4 #define DSEQ\_TYPE uint32\_t**

packet data sequence type

**4.21.2.5 #define ENCDATA\_SIZE\_TYPE uint32\_t**

encrypted packet payload size type

**4.21.2.6 #define FRAGMENTS\_TYPE uint16\_t**

packet fragments count type

**4.21.2.7 #define MAX\_DGRAM\_LEN\_TYPE uint32\_t**

packet option - maximum datagram length

**4.21.2.8 #define MSS\_TYPE uint16\_t**

packet option - maximum segment size

**4.21.2.9 #define OPTS\_OPTID\_TYPE uint16\_t**

option identification type (options sent in headers of packets)

**4.21.2.10 #define OPTS\_SZ\_TYPE uint16\_t**

option size type (options sent in headers of packets)

**4.21.2.11 #define OPTSZ\_TYPE uint16\_t**

packet option size type

**4.21.2.12 #define RETR\_TIMEOUT\_TYPE uint16\_t**

packet option - retransmits timeout

**4.21.2.13 #define SEQ\_MAX UINT32\_MAX**

sequence numbers maximal value

**4.21.2.14 #define SEQ\_TYPE uint32\_t**

sequence numbers type

**4.21.2.15 #define SID\_MAX UINT8\_MAX**

session identification max value

**4.21.2.16 #define SID\_TYPE uint8\_t**

session identification type

**4.21.2.17 #define SIGSZ\_TYPE uint32\_t**

signed packet payload size type

**4.21.2.18 #define TS\_TYPE uint32\_t**

packet timestamps type

---

# Index

- abuffers.c, 31
    - abuffers\_add\_seq, 32
    - abuffers\_create, 32
    - abuffers\_destroy, 33
    - abuffers\_get\_ack, 33
    - abuffers\_init, 33
  - abuffers.h, 35
    - abuffers\_add\_seq, 35
    - abuffers\_create, 36
    - abuffers\_destroy, 36
    - abuffers\_get\_ack, 36
    - abuffers\_init, 37
    - ARTP\_ABUFFERS\_H, 35
  - abuffers\_add\_seq
    - abuffers.c, 32
    - abuffers.h, 35
  - abuffers\_create
    - abuffers.c, 32
    - abuffers.h, 36
  - abuffers\_destroy
    - abuffers.c, 33
    - abuffers.h, 36
  - abuffers\_get\_ack
    - abuffers.c, 33
    - abuffers.h, 36
  - abuffers\_init
    - abuffers.c, 33
    - abuffers.h, 37
  - ACK
    - packet.h, 71
  - acks
    - Tabuffers, 22
  - acks\_count
    - Tabuffers, 22
  - act\_frag
    - item, 12
  - artp.c, 38
    - artp\_destroy\_connection, 39
    - artp\_free\_dgram, 39
    - artp\_get\_sid, 39
    - artp\_get\_undlvr\_session, 40
    - artp\_init, 40
    - artp\_prepare\_connection, 40
    - artp\_receive\_any\_dgram, 41
    - artp\_receive\_dgram, 41
    - artp\_send\_dgram, 42
    - artp\_set\_session\_options, 42
    - artp\_set\_session\_params, 42
  - artp.h, 44
    - artp\_destroy\_connection, 45
    - artp\_free\_dgram, 46
    - artp\_get\_sid, 46
    - artp\_get\_undlvr\_session, 46
    - ARTP\_H, 44
    - artp\_init, 47
    - artp\_prepare\_connection, 47
    - artp\_receive\_any\_dgram, 47
    - artp\_receive\_dgram, 48
    - artp\_send\_dgram, 48
    - artp\_session\_params, 45
    - artp\_set\_session\_options, 49
    - artp\_set\_session\_params, 49
    - EXP\_TIME, 45
    - MAX\_ACKS\_COUNT, 45
    - MAX\_RBUFFER\_RED\_LIMIT,  
45
    - MAX\_RBUFFER\_RED\_-  
PROBABILITY, 45
    - MAX\_RBUFFER\_SIZE, 45
    - MAX\_SBUFFER\_SIZE, 45
    - MSS, 45
    - RETRIES\_TIMEOUT, 45
  - ARTP\_ABUFFERS\_H
    - abuffers.h, 35
  - ARTP\_CONFIG\_H
    - config.h, 51
  - artp\_destroy\_connection
    - artp.c, 39
    - artp.h, 45
  - artp\_dgram, 5
    - payload, 5
    - type, 5
-

- artp\_error\_str
  - errors.c, [55](#)
  - errors.h, [57](#)
- artp\_errors
  - errors.h, [56](#)
- ARTP\_ERRORS\_H
  - errors.h, [56](#)
- artp\_free\_dgram
  - artp.c, [39](#)
  - artp.h, [46](#)
- artp\_get\_sid
  - artp.c, [39](#)
  - artp.h, [46](#)
- artp\_get\_undlvr\_session
  - artp.c, [40](#)
  - artp.h, [46](#)
- ARTP\_H
  - artp.h, [44](#)
- artp\_init
  - artp.c, [40](#)
  - artp.h, [47](#)
- ARTP\_NET\_H
  - net.h, [61](#)
- ARTP\_OPTIONS\_H
  - options.h, [66](#)
- ARTP\_PACKET\_H
  - packet.h, [70](#)
- artp\_prepare\_connection
  - artp.c, [40](#)
  - artp.h, [47](#)
- ARTP\_RBUFFERS\_H
  - rbuffers.h, [76](#)
- artp\_receive\_any\_dgram
  - artp.c, [41](#)
  - artp.h, [47](#)
- artp\_receive\_dgram
  - artp.c, [41](#)
  - artp.h, [48](#)
- artp\_receiver, [7](#)
  - ip, [7](#)
  - ip6, [7](#)
- ARTP\_RWLOCKS\_H
  - rwlocks.h, [80](#)
- ARTP\_SBUFFERS\_H
  - sbuffers.h, [89](#)
- artp\_send\_dgram
  - artp.c, [42](#)
  - artp.h, [48](#)
- artp\_session\_options
  - options.h, [67](#)
- artp\_session\_params
  - artp.h, [45](#)
- artp\_set\_session\_options
  - artp.c, [42](#)
  - artp.h, [49](#)
- artp\_set\_session\_params
  - artp.c, [42](#)
  - artp.h, [49](#)
- ARTP\_SETTING\_H
  - setting.h, [98](#)
- ARTP\_STRUCTS\_H
  - structs.h, [100](#)
- ARTP\_TYPES\_H
  - types.h, [102](#)
- ARTP\_VERSION
  - config.h, [51](#)
- buffer\_info
  - item, [12](#)
- buffer\_size
  - Trcvng\_buffers, [25](#)
  - Tsending\_buffers, [27](#)
- buffer\_size\_mutex
  - Trcvng\_buffers, [25](#)
  - Tsending\_buffers, [27](#)
- buffers\_id
  - session\_item, [18](#)
- complete\_end
  - Trcvng\_buffers, [25](#)
- complete\_start
  - Trcvng\_buffers, [25](#)
- complete\_start\_mutex
  - Trcvng\_buffers, [25](#)
- config.h, [51](#)
  - ARTP\_CONFIG\_H, [51](#)
  - ARTP\_VERSION, [51](#)
  - CWND\_ACK, [51](#)
  - CWND\_DOWN\_MULTIPLIER, [51](#)
  - CWND\_RETRANS, [52](#)
  - DUPPLE\_SEQ\_COUNT, [52](#)
  - INITIAL\_WINDOW\_SIZE, [52](#)
  - LATEST\_ACKS\_SEND, [52](#)
  - LINK\_IDLE, [52](#)
  - MAX\_ARTP\_PACKET\_SIZE, [52](#)
  - MAX\_CONFIG\_LINE\_LENGTH, [53](#)
  - RTO\_COMPUTE, [53](#)
  - SRTT\_ALPHA, [53](#)

- SRTT\_COMPUTE, 53
- TS\_DELTA\_ALPHA, 53
- TS\_DELTA\_COMPUTE, 53
- control
  - payload\_CTRL, 15
- control\_type, 8
  - optid, 8
  - type, 8
  - value, 8
  - valuesize, 8
- count
  - payload\_CTRL, 15
- CTRL
  - packet.h, 71
- ctrl
  - payload\_type, 17
- CTRL\_OPTID\_TYPE
  - types.h, 102
- CTRL\_SZ\_TYPE
  - types.h, 102
- ctrl\_type
  - packet.h, 70
- CTRL\_VALUE\_SZ\_TYPE
  - types.h, 102
- current\_seq
  - session\_item, 18
- cwnd
  - session\_item, 19
- CWND\_ACK
  - config.h, 51
- CWND\_DOWN\_MULTIPLIER
  - config.h, 51
- CWND\_RETRANS
  - config.h, 52
- DATA
  - packet.h, 71
- data
  - payload\_type, 17
- DEAD
  - structs.h, 100
- dead\_zero\_sessions, 9
  - invalid, 9
  - next, 9
  - receiver, 9
  - sid, 9
- DEFAULT\_MAX\_ACKS\_COUNT
  - setting.c, 95
- default\_max\_acks\_count
  - Tsetting, 29
- DEFAULT\_RED\_DROP\_PROBABILITY
  - setting.c, 95
- default\_red\_drop\_probability
  - Tsetting, 29
- DEFAULT\_RETRIES\_TIMEOUT
  - setting.c, 96
- default\_retries\_timeout
  - Tsetting, 29
- dgram\_type
  - item, 12
- dseq
  - item, 12
  - payload\_DATA, 16
- DSEQ\_TYPE
  - types.h, 103
- dupple\_end
  - Trcving\_buffers, 25
- DUPPLE\_SEQ\_COUNT
  - config.h, 52
- dupple\_start
  - Trcving\_buffers, 25
- E\_BAD\_DGRAM
  - errors.h, 57
- E\_BAD\_OPT\_USE
  - errors.h, 57
- E\_BAD\_PACKET\_TYPE
  - errors.h, 57
- E\_BAD\_PARAM\_VALUE
  - errors.h, 57
- E\_BAD\_VALUE
  - errors.h, 57
- E\_BIG\_DGRAM
  - errors.h, 57
- E\_BUF\_INIT\_ERROR
  - errors.h, 57
- E\_DEAD\_SESSION
  - errors.h, 57
- E\_DIFF\_ADDR
  - errors.h, 57
- E\_DUPLICITY\_PACKET
  - errors.h, 57
- E\_EMPTY\_BUFFER
  - errors.h, 56
- E\_FULL\_BUFFER
  - errors.h, 56
- E\_FULL\_CWND
  - errors.h, 57
- E\_GENERIC\_ERROR

- errors.h, 56
- E\_INVALID\_BUFFER\_ID
  - errors.h, 57
- E\_INVALID\_OPT\_SIZE
  - errors.h, 57
- E\_INVALID\_OPT\_VALUE
  - errors.h, 57
- E\_INVALID\_OPTION
  - errors.h, 57
- E\_INVALID\_PARAMETER
  - errors.h, 57
- E\_MEMORY\_FAIL
  - errors.h, 56
- E\_NO\_ACK
  - errors.h, 57
- E\_NO\_AVAIL\_SID
  - errors.h, 57
- E\_NONEST\_SESSION
  - errors.h, 57
- E\_NULL\_PTR
  - errors.h, 57
- E\_OPENING\_FILE
  - errors.h, 57
- E\_PACKET\_NOT\_FOUND
  - errors.h, 57
- E\_PARTNER\_MSS
  - errors.h, 57
- E\_SENDING\_ERROR
  - errors.h, 57
- E\_SESSION\_EXISTS
  - errors.h, 57
- E\_SMALL\_MSS
  - errors.h, 57
- E\_START\_THREADS
  - errors.h, 57
- E\_SYNTAX\_ERROR
  - errors.h, 57
- E\_UNSUPPORTED\_AF
  - errors.h, 57
- encdata
  - payload\_DATA, 16
- ENCDATA\_SIZE\_TYPE
  - types.h, 103
- encsz
  - payload\_DATA, 16
- end
  - Tsending\_buffers, 27
- end\_mutex
  - Tsending\_buffers, 27
- errors.c, 55
  - artp\_error\_str, 55
- errors.h, 56
  - artp\_error\_str, 57
  - artp\_errors, 56
  - ARTP\_ERRORS\_H, 56
  - E\_BAD\_DGRAM, 57
  - E\_BAD\_OPT\_USE, 57
  - E\_BAD\_PACKET\_TYPE, 57
  - E\_BAD\_PARAM\_VALUE, 57
  - E\_BAD\_VALUE, 57
  - E\_BIG\_DGRAM, 57
  - E\_BUF\_INIT\_ERROR, 57
  - E\_DEAD\_SESSION, 57
  - E\_DIFF\_ADDR, 57
  - E\_DUPLICITY\_PACKET, 57
  - E\_EMPTY\_BUFFER, 56
  - E\_FULL\_BUFFER, 56
  - E\_FULL\_CWND, 57
  - E\_GENERIC\_ERROR, 56
  - E\_INVALID\_BUFFER\_ID, 57
  - E\_INVALID\_OPT\_SIZE, 57
  - E\_INVALID\_OPT\_VALUE, 57
  - E\_INVALID\_OPTION, 57
  - E\_INVALID\_PARAMETER, 57
  - E\_MEMORY\_FAIL, 56
  - E\_NO\_ACK, 57
  - E\_NO\_AVAIL\_SID, 57
  - E\_NONEST\_SESSION, 57
  - E\_NULL\_PTR, 57
  - E\_OPENING\_FILE, 57
  - E\_PACKET\_NOT\_FOUND, 57
  - E\_PARTNER\_MSS, 57
  - E\_SENDING\_ERROR, 57
  - E\_SESSION\_EXISTS, 57
  - E\_SMALL\_MSS, 57
  - E\_START\_THREADS, 57
  - E\_SYNTAX\_ERROR, 57
  - E\_UNSUPPORTED\_AF, 57
- EST
  - structs.h, 101
- EXP\_TIME
  - artp.h, 45
- expiration\_time
  - session\_item, 19
- ext\_item, 10
  - main\_item, 10
  - receiver, 10
  - sender, 10
  - sid, 10



- first\_send\_time
  - item, 13
- flight
  - session\_item, 19
- frag\_count
  - item, 13
- frag\_id
  - fragment\_item, 11
- fragment\_item, 11
  - frag\_id, 11
  - last\_fragment, 11
  - next\_fragment, 11
  - payload, 11
  - payload\_size, 11
- fragments\_end
  - item, 13
- fragments\_start
  - item, 13
- FRAGMENTS\_TYPE
  - types.h, 103
- get\_session\_options
  - options.c, 63
  - options.h, 67
- global\_setting
  - setting.h, 99
- incomplete\_end
  - Trcving\_buffers, 26
- incomplete\_start
  - Trcving\_buffers, 26
- INITIAL\_EXP\_TIME
  - setting.c, 96
- initial\_exp\_time
  - Tsetting, 29
- INITIAL\_MSS
  - setting.c, 96
- initial\_mss
  - Tsetting, 29
- INITIAL\_RBUFFERS\_MAX\_SIZE
  - setting.c, 96
- initial\_rbuffers\_max\_size
  - Tsetting, 29
- INITIAL\_RBUFFERS\_RED\_LIMIT
  - setting.c, 96
- initial\_rbuffers\_red\_limit
  - Tsetting, 30
- INITIAL\_RTO\_TIME
  - setting.c, 96
- initial\_rto\_time
  - Tsetting, 30
- INITIAL\_SBUFFERS\_MAX\_SIZE
  - setting.c, 96
- initial\_sbuffers\_max\_size
  - Tsetting, 30
- INITIAL\_WINDOW\_SIZE
  - config.h, 52
- invalid
  - dead\_zero\_sessions, 9
  - item, 13
- ip
  - artp\_receiver, 7
- ip6
  - artp\_receiver, 7
- item, 12
  - act\_frag, 12
  - buffer\_info, 12
  - dgram\_type, 12
  - dseq, 12
  - first\_send\_time, 13
  - frag\_count, 13
  - fragments\_end, 13
  - fragments\_start, 13
  - invalid, 13
  - last, 13
  - last\_to\_resend, 13
  - length, 13
  - next, 13
  - next\_to\_resend, 14
  - packet, 14
  - retr\_counter, 14
  - seq, 14
  - time\_to\_resend, 14
- last
  - item, 13
  - Tabuffers, 22
  - Tdupple, 24
- last\_fragment
  - fragment\_item, 11
- last\_send\_time
  - session\_item, 19
- last\_to\_resend
  - item, 13
- LATEST\_ACKS\_SEND
  - config.h, 52
- latest\_send\_time
  - Tabuffers, 22
- length
  - item, 13

- LINK\_IDLE
  - config.h, 52
- LIVE
  - structs.h, 100
- main\_item
  - ext\_item, 10
- MAX\_ACKS\_COUNT
  - artp.h, 45
- max\_acks\_count
  - session\_item, 19
- MAX\_ARTP\_PACKET\_SIZE
  - config.h, 52
- MAX\_CONFIG\_LINE\_LENGTH
  - config.h, 53
- MAX\_DGRAM\_LEN
  - options.h, 67
- MAX\_DGRAM\_LEN\_TYPE
  - types.h, 103
- MAX\_MSS
  - options.h, 67
- MAX\_RBUFFER\_RED\_LIMIT
  - artp.h, 45
- MAX\_RBUFFER\_RED\_-  
PROBABILITY
  - artp.h, 45
- MAX\_RBUFFER\_SIZE
  - artp.h, 45
- MAX\_SBUFFER\_SIZE
  - artp.h, 45
- max\_seq
  - Tdupple, 24
- max\_seq\_time
  - Tdupple, 24
- min\_seq
  - Tdupple, 24
- MSS
  - artp.h, 45
- mss
  - session\_item, 19
- MSS\_TYPE
  - types.h, 103
- net.c, 59
  - rcvrcmp, 59
  - rcvrncpy, 59
  - rcvrpsz, 60
- net.h, 61
  - ARTP\_NET\_H, 61
  - rcvrcmp, 61
  - rcvrncpy, 62
  - rcvrpsz, 62
- next
  - dead\_zero\_sessions, 9
  - item, 13
  - Tabuffers, 22
  - Tdupple, 24
- next\_fragment
  - fragment\_item, 11
- next\_to\_resend
  - item, 14
- NON\_EST
  - structs.h, 101
- optid
  - control\_type, 8
- options
  - session\_item, 19
- options.c, 63
  - get\_session\_options, 63
  - options\_init, 64
  - parse\_session\_options, 64
  - set\_default\_options, 64
  - set\_global\_option, 65
  - use\_options, 65
- options.h, 66
  - ARTP\_OPTIONS\_H, 66
  - artp\_session\_options, 67
  - get\_session\_options, 67
  - MAX\_DGRAM\_LEN, 67
  - MAX\_MSS, 67
  - OPTIONS\_COUNT, 66
  - options\_init, 67
  - parse\_session\_options, 68
  - RETRANSMITS\_TIMEOUT, 67
  - set\_default\_options, 68
  - set\_global\_option, 68
  - use\_options, 68
- OPTIONS\_COUNT
  - options.h, 66
- options\_init
  - options.c, 64
  - options.h, 67
- OPTS\_OPTID\_TYPE
  - types.h, 103
- OPTS\_SZ\_TYPE
  - types.h, 103
- OPTSZ\_TYPE
  - types.h, 103

- packet
  - item, 14
- packet.h, 70
  - ACK, 71
  - ARTP\_PACKET\_H, 70
  - CTRL, 71
  - ctrl\_type, 70
  - DATA, 71
  - packet\_type, 70
  - REPLY, 70
  - REQUEST, 70
- packet\_type
  - packet.h, 70
- parse\_session\_options
  - options.c, 64
  - options.h, 68
- partner\_options
  - session\_item, 19
- payload
  - artp\_dgram, 5
  - fragment\_item, 11
- payload\_CTRL, 15
  - control, 15
  - count, 15
- payload\_DATA, 16
  - dseq, 16
  - encdata, 16
  - encsz, 16
  - sigdata, 16
  - sigsz, 16
- payload\_size
  - fragment\_item, 11
- payload\_type, 17
  - ctrl, 17
  - data, 17
- rbuffer\_max\_size
  - session\_item, 19
- rbuffer\_red\_limit
  - session\_item, 19
- rbuffer\_red\_prob
  - session\_item, 19
- rbuffers.c, 72
  - rbuffers\_add\_packet, 73
  - rbuffers\_create, 73
  - rbuffers\_destroy, 74
  - rbuffers\_get\_dgram, 74
  - rbuffers\_get\_size, 75
  - rbuffers\_init, 75
- rbuffers.h, 76
  - ARTP\_RBUFFERS\_H, 76
  - rbuffers\_add\_packet, 76
  - rbuffers\_create, 77
  - rbuffers\_destroy, 77
  - rbuffers\_get\_dgram, 78
  - rbuffers\_get\_size, 78
  - rbuffers\_init, 79
- rbuffers\_add\_packet
  - rbuffers.c, 73
  - rbuffers.h, 76
- rbuffers\_create
  - rbuffers.c, 73
  - rbuffers.h, 77
- rbuffers\_destroy
  - rbuffers.c, 74
  - rbuffers.h, 77
- rbuffers\_get\_dgram
  - rbuffers.c, 74
  - rbuffers.h, 78
- rbuffers\_get\_size
  - rbuffers.c, 75
  - rbuffers.h, 78
- rbuffers\_init
  - rbuffers.c, 75
  - rbuffers.h, 79
- rcvrcmp
  - net.c, 59
  - net.h, 61
- rcvrncpy
  - net.c, 59
  - net.h, 62
- rcvrsz
  - net.c, 60
  - net.h, 62
- READERS\_LOCK
  - rwlocks.h, 80
- READERS\_UNLOCK
  - rwlocks.h, 80
- receiver
  - dead\_zero\_sessions, 9
  - ext\_item, 10
  - Tabuffers, 22
  - Tsending\_buffers, 27
- ref\_counter
  - session\_item, 20
- ref\_counter\_mutex
  - session\_item, 20
- REPLY
  - packet.h, 70
- REQUEST

- packet.h, 70
- retr\_counter
  - item, 14
- RETR\_TIMEOUT\_TYPE
  - types.h, 103
- RETRANSMITS\_TIMEOUT
  - options.h, 67
- RETRIES\_TIMEOUT
  - artp.h, 45
- retries\_timeout
  - session\_item, 20
- rto
  - session\_item, 20
- RTO\_COMPUTE
  - config.h, 53
- rtt
  - session\_item, 20
- RW\_INIT
  - rwlocks.h, 81
- rwlocks.h, 80
  - ARTP\_RWLOCKS\_H, 80
  - READERS\_LOCK, 80
  - READERS\_UNLOCK, 80
  - RW\_INIT, 81
  - WRITERS\_LOCK, 81
  - WRITERS\_UNLOCK, 81
- sbuffer\_max\_size
  - session\_item, 20
- sbuffer.c, 83
  - sbuffer\_ack\_event, 84
  - sbuffer\_add\_packet, 84
  - sbuffer\_create, 85
  - sbuffer\_destroy, 85
  - sbuffer\_get\_packet, 85
  - sbuffer\_get\_rsnd\_packet, 86
  - sbuffer\_get\_size, 86
  - sbuffer\_get\_top\_rsnd\_time, 87
  - sbuffer\_ignore\_first\_rsnd, 83
  - sbuffer\_init, 87
  - sbuffer\_resend\_event, 87
  - sbuffer\_send\_event, 88
- sbuffer.h, 89
  - ARTP\_SBUFFERS\_H, 89
  - sbuffer\_ack\_event, 90
  - sbuffer\_add\_packet, 90
  - sbuffer\_create, 91
  - sbuffer\_destroy, 91
  - sbuffer\_get\_packet, 91
  - sbuffer\_get\_rsnd\_packet, 92
  - sbuffer\_get\_size, 92
  - sbuffer\_get\_top\_rsnd\_time, 93
  - sbuffer\_ignore\_first\_rsnd, 93
  - sbuffer\_init, 93
  - sbuffer\_resend\_event, 93
  - sbuffer\_send\_event, 94
- sender
  - ext\_item, 10
- sent\_end
  - Tsending\_buffers, 27
- sent\_end\_mutex
  - Tsending\_buffers, 28
- sent\_start
  - Tsending\_buffers, 28

- sent\_start\_mutex
  - Tsending\_buffers, 28
- seq
  - item, 14
- SEQ\_MAX
  - types.h, 103
- SEQ\_TYPE
  - types.h, 103
- session\_item, 18
  - buffers\_id, 18
  - current\_seq, 18
  - cwnd, 19
  - expiration\_time, 19
  - flight, 19
  - last\_send\_time, 19
  - max\_acks\_count, 19
  - mss, 19
  - options, 19
  - partner\_options, 19
  - rbuffer\_max\_size, 19
  - rbuffer\_red\_limit, 19
  - rbuffer\_red\_prob, 19
  - ref\_counter, 20
  - ref\_counter\_mutex, 20
  - retries\_timeout, 20
  - rto, 20
  - rtt, 20
  - sbuffer\_max\_size, 20
  - session\_mutex, 20
  - session\_receiver, 20
  - session\_sid, 20
  - session\_status, 20
  - session\_type, 21
  - srtt, 21
  - ts\_delta, 21
- session\_mutex
  - session\_item, 20
- session\_receiver
  - session\_item, 20
- session\_sid
  - session\_item, 20
- session\_status
  - session\_item, 20
- session\_type
  - session\_item, 21
- set\_default\_options
  - options.c, 64
  - options.h, 68
- set\_global\_option
  - options.c, 65
  - options.h, 68
- setting.c, 95
  - DEFAULT\_MAX\_ACKS\_COUNT, 95
  - DEFAULT\_RED\_DROP\_PROBABILITY, 95
  - DEFAULT\_RETRIES\_TIMEOUT, 96
  - INITIAL\_EXP\_TIME, 96
  - INITIAL\_MSS, 96
  - INITIAL\_RBUFFERS\_MAX\_SIZE, 96
  - INITIAL\_RBUFFERS\_RED\_LIMIT, 96
  - INITIAL\_RTO\_TIME, 96
  - INITIAL\_SBUFFERS\_MAX\_SIZE, 96
  - setting\_read\_file, 96
  - setting\_set\_defaults, 97
- setting.h, 98
  - ARTP\_SETTING\_H, 98
  - global\_setting, 99
  - setting\_read\_file, 98
  - setting\_set\_defaults, 99
- setting\_read\_file
  - setting.c, 96
  - setting.h, 98
- setting\_set\_defaults
  - setting.c, 97
  - setting.h, 99
- sid
  - dead\_zero\_sessions, 9
  - ext\_item, 10
  - Tabuffers, 22
- SID\_MAX
  - types.h, 104
- SID\_TYPE
  - types.h, 104
- sigdata
  - payload\_DATA, 16
- sigsz
  - payload\_DATA, 16
- SIGSZ\_TYPE
  - types.h, 104
- srtt
  - session\_item, 21
- SRTT\_ALPHA
  - config.h, 53
- SRTT\_COMPUTE
  - config.h, 53

- structs.h, 100
  - ARTP\_STRUCTS\_H, 100
  - DEAD, 100
  - EST, 101
  - LIVE, 100
  - NON\_EST, 101
  - Tsession\_status, 100
  - Tsession\_type, 101
- Tabuffers, 22
  - acks, 22
  - acks\_count, 22
  - last, 22
  - latest\_send\_time, 22
  - next, 22
  - receiver, 22
  - sid, 22
- Tdupple, 24
  - last, 24
  - max\_seq, 24
  - max\_seq\_time, 24
  - min\_seq, 24
  - next, 24
- time\_to\_resend
  - item, 14
- to\_send
  - Tsending\_buffers, 28
- to\_send\_mutex
  - Tsending\_buffers, 28
- Trcvng\_buffers, 25
  - buffer\_size, 25
  - buffer\_size\_mutex, 25
  - complete\_end, 25
  - complete\_start, 25
  - complete\_start\_mutex, 25
  - dupple\_end, 25
  - dupple\_start, 25
  - incomplete\_end, 26
  - incomplete\_start, 26
- ts\_delta
  - session\_item, 21
- TS\_DELTA\_ALPHA
  - config.h, 53
- TS\_DELTA\_COMPUTE
  - config.h, 53
- TS\_TYPE
  - types.h, 104
- Tsending\_buffers, 27
  - buffer\_size, 27
  - buffer\_size\_mutex, 27
  - end, 27
  - end\_mutex, 27
  - receiver, 27
  - sent\_end, 27
  - sent\_end\_mutex, 28
  - sent\_start, 28
  - sent\_start\_mutex, 28
  - to\_send, 28
  - to\_send\_mutex, 28
- Tsession\_status
  - structs.h, 100
- Tsession\_type
  - structs.h, 101
- Tsetting, 29
  - default\_max\_acks\_count, 29
  - default\_red\_drop\_probability, 29
  - default\_retries\_timeout, 29
  - initial\_exp\_time, 29
  - initial\_mss, 29
  - initial\_rbuffers\_max\_size, 29
  - initial\_rbuffers\_red\_limit, 30
  - initial\_rto\_time, 30
  - initial\_sbuffers\_max\_size, 30
- type
  - artp\_dgram, 5
  - control\_type, 8
- types.h, 102
  - ARTP\_TYPES\_H, 102
  - CTRL\_OPTID\_TYPE, 102
  - CTRL\_SZ\_TYPE, 102
  - CTRL\_VALUE\_SZ\_TYPE, 102
  - DSEQ\_TYPE, 103
  - ENCDATA\_SIZE\_TYPE, 103
  - FRAGMENTS\_TYPE, 103
  - MAX\_DGRAM\_LEN\_TYPE, 103
  - MSS\_TYPE, 103
  - OPTS\_OPTID\_TYPE, 103
  - OPTS\_SZ\_TYPE, 103
  - OPTSZ\_TYPE, 103
  - RETR\_TIMEOUT\_TYPE, 103
  - SEQ\_MAX, 103
  - SEQ\_TYPE, 103
  - SID\_MAX, 104
  - SID\_TYPE, 104
  - SIGSZ\_TYPE, 104
  - TS\_TYPE, 104
- use\_options
  - options.c, 65

options.h, [68](#)

value

control\_type, [8](#)

valuesize

control\_type, [8](#)

WRITERS\_LOCK

rwlocks.h, [81](#)

WRITERS\_UNLOCK

rwlocks.h, [81](#)