Active Router Transport Protocol (ARTP) Reference Manual

1.0

Generated by Doxygen 1.2.15

Fri May 21 08:02:47 2004

Contents

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:

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:

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
- Tsending_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 Tsending_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 Trcving_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 Trcving_buffers::buffer_size

counter of buffer size

3.14.2.2 pthread_mutex_t Trcving_buffers::buffer_size_mutex

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

3.14.2.3 struct item* Trcving_buffers::complete_end

complete buffer end

3.14.2.4 struct item* Trcving_buffers::complete_start

complete buffer start

3.14.2.5 pthread_mutex_t Trcving_buffers::complete_start_mutex

mutex used for mutual exclusion of threads reading from complete buffer

3.14.2.6 struct Tdupple* Trcving_buffers::dupple_end

end of structure for determining duplicities

3.14.2.7 struct Tdupple* Trcving_buffers::dupple_start

start of structure for determining duplicities

3.14.2.8 struct item* Trcving_buffers::incomplete_end

incomplete buffer end

3.14.2.9 struct item* Trcving_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_MULTIPLIER 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(srtt) ((srtt) / 2.0)

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

Parameters:

srtt 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(srtt, init_resend_time) (srtt > 0) ? (4 * (srtt)) : init_resend_time

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

Parameters:

srtt 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(rtt, old_srtt)

Value:

```
((old_srtt) == 0) ? rtt \
                     : (SRTT_ALPHA * (old_srtt)) + ((1 - SRTT_ALPHA) * (rtt));
```

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:

rtt actual round trip time

old_srtt 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_PARAM_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 Refering to non-established session

E_SESSION_EXISTS Wanted session already exists

E_PACKET_NOT_FOUND Refering 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 rcvrsz (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 rcvrsz (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, RETRANSMITS_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 Trcving_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 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 *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:

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

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:

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

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

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 sbuffers.h File Reference

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

Defines

- #define **ARTP_SBUFFERS_H** 1

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 ()

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 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`).

4.17.2.2 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`).

4.17.2.3 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 10000000

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