RTP Protocol Modules for TTCN-3 Toolset with TITAN DESCRIPTION

Gábor Szalai

Version 1551-CNL 113 392, Rev. B, 2015-08-14

Table of Contents

How to Read This Document

This is the Function Description for the RTP protocol module. The RTP protocol module is developed for the TTCN-3 Toolset with TITAN.

Scope

The purpose of this document is to specify the content of the RTP protocol modules. Basic knowledge of TTCN-3 [1] and TITAN TTCN-3 Test Executor [2] is valuable when reading this document.

System Requirements

Protocol modules are a set of TTCN-3 source code files that can be used as part of TTCN-3 test suites only. Hence, protocol modules alone do not put specific requirements on the system used. However, in order to compile and execute a TTCN-3 test suite using the set of protocol modules, the following system requirements must be satisfied:

- TITAN TTCN-3 Test Executor Executor version R7A (1.7.pl0) or higher installed. For installation guide see [2]

NOTE	This version of the protocol module is not compatible with TITAN releases earlier than R7A.

Protocol Modules

Overview

Protocol modules implement the message structure of the related protocol in a formalized way, using the standard specification language TTCN-3. This allows defining of test data (templates) in the TTCN-3 language [1] and correct encoding/decoding messages when executing test suites using the Titan TTCN-3 test environment.

Additionally, there are some implemented functions, which are able to read/write media files. The return value of the reading function is in the RTP payload format according to the media. The incoming parameter of the writing function is an octetstring (an RTP payload format packet).

Protocol module uses Titan's RAW encoding attributes and hence is usable with the Titan test toolset only.

Installation

- The set of protocol modules can be used for developing TTCN-3 test suites using any text editor. However to make the work more efficient a TTCN-3-enabled text editor is recommended

(e.g.`nedit`, `xemacs`). Since the RTP protocol is used as a part of a TTCN-3 test suite, this requires TTCN-3 Test Executor be installed before the module can be compiled and executed together with other parts of the test suite. For more details on the installation of TTCN-3 Test Executor see the relevant section of [2].

Configuration

None.

Functional Specification

Protocol Version Implemented

This set of protocol modules implement protocol messages and constants of the RTP protocol [3] with the modifications specified in Modifications/Deviations Related to the Protocol Specification.

The RTCP messages described in RFC 4585 & RFC 5104 have been implemented

Modifications/Deviations Related to the Protocol Specification

Unimplemented Messages, Information Elements and Constants

None.

Protocol Modifications/Deviations

Protocol modules contain the following deviations from :

- The protocol module does not support the encryption of the messages. SRTP_CNL113769 can encrypt and decrypt the messages. See [8].

Encoding/Decoding Functions

This product contains encoding/decoding functions, which assure correct encoding of messages when sent from Titan and correct decoding of messages when received by Titan. Implemented encoding/decoding functions:

Name	Type of formal parameters	Type of return value
`f_RTP_enc`	(in RTP_messages_union pdu)	octetstring;

Name	Type of formal parameters	Type of return value
f_RTP_dec	(in octetstring data)	RTP_messages_union;
f_RTP_packet_enc	(in RTP_packet pdu)	octetstring;
f_RTP_packet_dec	(in octetstring data)	RTP_packet;

Media Stream Handling in RTP Protocol

General

Each RTP packet, starts with a fixed RTP header followed by payload format [3].

Supported Media Files

- JPEG - JPEG video codec [5] (Note2)
- H263 - H263 video codec ([4] mode: A)
- MPEG4 - MPEG4 video codec (Note1)
- GIF - GIF image codec (Note1)
- f3GP - 3gp file format (Note1)
- fMPEG4 - MPEG4 file format (Note1)
- UNKNOWN - other (Note1)
  - Note1: The payload is handled as an octetstream.
  - Note2: The f_Put_Media_Content function sets the fragment_offset field of the payload header structure to zero when writing the payload header into the file.

Reading/Writing and Other Related Functions

This product also contains read/write functions so that we can read any blocks and the function returns RTP payload format, and write received RTP payload packets into a file. Implemented encoding/decoding functions:

Types of formal parameters:

Type	Description
InitOperType	enumerated (OPEN, CREATE)
RTP_MediaType	enumerated (JPEG, H263, MPEG4, GIF, f3GP, fMPEG4, UNKNOWN)

RTP_FileInfo record with the following fields:

```
integer fd, // file description

integer block_size, // size of block

integer block_no, // starting block position

integer nof_blocks_to_read, // number of blocks to read

RTP_MediaType mediaType, // type of media

boolean headerOp, // true: get or put header

integer headerOffset, // size of media header

Media_RTP_Header mediaHeader // header of the media
```

Implemented functions:

Name	Type of formal parameters	Type of return value
f_Init_Media_Fileinfo	(in charstring pl_filename in integer pl_block_size in InitOperType pl_init_type in RTP_MediaType pl_media_type in integer pl_blockno in integer pl_nof_blocks inout RTP_FileInfo pl_fileinfo)	none

The f_Init_Media_Fileinfo opens (OPEN) or creates (CREATE) the file to read or write the blocks. This function sets the FileInfo parameter that contains the fd, block_size, block_no, nof_blocks_to_read, Header_Offset, etc. values.

Name	Type of formal parameters	Type of return value
f_Get_Media_Content	(inout RTP_FileInfo pl_filename)	octetstring

This function reads the blocks from the file and encapsulates it according to RTP payload format. The function encodes this packet to octetstring.

Name	Type of formal parameters	Type of return value
f_Put_Media_Content	(in RTP_FileInfo pl_fileinfo, in octetstring data)	integer

This function decodes the incoming octetstring (RTP payload format packet) and writes the appropriate data into the file.

These functions may be useful when we want to manipulate the files directly.

Name	Type of formal parameters	Type of return value
f_INIT_CODEC	(in charstring pl_filename, in integer pl_block_size, in InitOperType pl_init_type)	integer

The f_INIT_CODEC opens (OPEN) or creates (CREATE) the file to read or to write the blocks and sets the size of the blocks. It returns the identifier of the file.

Name	Type of formal parameters	Type of return value
f_GET_CONTENT	(in integer pl_fd, in integer pl_blockno, in integer pl_nof_blocks_to_read, in integer pl_header_offset)	octetstring

The f_GET_CONTENT reads the blocks from the file. It reads nof_blocks_to_read blocks starting from (blockno + header_offset).

Name	Type of formal parameters	Type of return value
f_PUT_CONTENT	(in integer pl_fd, in integer pl_blockno, in octetstring pl_stream, in integer pl_header_offset)	octetstring

This function writes the blocks into the file. It writes nof_blocks_to_read blocks starting from (blockno + header_offset).

Name	Type of formal parameters	Type of return value
f_CLOSE_CODEC	(in integer pl_fd)	none

The f_CLOSE_CODEC closes the file.

There are further auxiliary inside functions that are not for direct use of the user (they are used by the other functions).

Encoding/decoding functions for RTP payload formats of media:

Name	Type of formal parameters	Type of return value
f_RTP_Hdr_enc	(in Media_RTP_Header hdr)	octetstring
f_JPEG_RTP_Hdr_dec	(in octetstring stream)	JPEG_RTP_Header
f_H263_RTP_Hdr_dec	(in octetstring stream)	H263_RTP_Header
f_RTP_Data_enc	(in Media_RTP_Data rtp_data)	octetstring
f_JPEG_RTP_Data_dec	(in octetstring stream)	JPEG_RTP
f_H263_RTP_Data_dec	(in octetstring stream)	H263_RTP

Other inside functions:

```
f_Count_JPEG_Header_Offset( in FileInfo_t fi);

void log_info_list()

int f_Fileinfo_Check(const int& fd, const Operation& OPERATION)

int f_Operation_Check(const int& fd, const int& blockno, const int& nof_b, const
Operation& OPERATION, const int& hdr_off)
```

Error Messages

ERROR: Wrong media type setting!

ERROR: INIT__CODEC: empty filename is not allowed

ERROR: INIT__CODEC: Block size must be a positive integer

ERROR: INIT__CODEC: There is not enough memory.

ERROR: "INIT__CODEC: Cannot open file '%s'", filename

ERROR: "INIT__CODEC: Cannot create file '%s'", filename

ERROR: INIT__CODEC: Wrong init_type setting! Available: OPEN, CREATE.

ERROR: INIT__CODEC: Cannot gather file info

ERROR: GET_CONTENT: There is not enough memory.

ERROR: GET_CONTENT: unsuccesful read (%d), read_data

ERROR: Get_Media_Header: Header offset must be a non-negative integer!

ERROR: Get_Media_Header: Size of file %s is smaller than the size of header!, filename

ERROR: "Count_JPEG_Header_Offset: Cannot set the starting position in file %s", filename

ERROR: Count_JPEG_Header_Offset: There is not enough memory.

ERROR: Count_JPEG_Header_Offset: unsuccesful read (%d), read_data

ERROR: PUT_CONTENT: unsuccesful write to file

ERROR: CLOSE__CODEC: There is not enough memory

ERROR: CLOSE__CODEC: Unknown file descriptor (%d), fd

ERROR: Fileinfo_Check: Unknown file descriptor (%d), fd

ERROR: Fileinfo_Check: file info list is empty

ERROR: Fileinfo_Check: inconsistent file info list (filename is missing)

ERROR: Fileinfo_Check: inconsistent file info list

ERROR: Fileinfo_Check: Cannot gather file info

```
ERROR: Operation_Check: The number of blocks to read must be a non-negative integer

ERROR: Operation_Check: Starting block position must be a non-negative integer

ERROR: Operation_Check: Header offset must be a non-negative integer

ERROR: Operation_Check: Size of file %s is smaller than the starting block position, filename

ERROR: Operation_Check: Cannot set the starting position in file %s, filename

ERROR: Operation_Check: Wrong OPERATION setting! Available: READ, WRITE.

ERROR: RTP_Hdr_enc: The incoming parameter (hdr) is unbound!

ERROR: RTP_Data_enc: The incoming parameter (rtp_data) is unbound!
```

Warning Messages

```
WARNING: "INIT__CODEC: File %s contains uncomplete blocks", filenam

WARNING: "Operation_Check: This is an uncomplete block. Size: %d byte/bytes."",
bytes_to_operation
```

Examples

The "demo" directory of the deliverable contains the following examples and functions:

Mapping Module

The mapping module provides the connection between the RTP protocol module and the UDP test port. It encodes and decodes the RTP messages and manages the opening and closing of RTP sessions.

Open Session

New session is requested by the `ASP_RTP_Open_session` message. The `session_id` contains the requested parameter of the new session.

- `id`:
  The unique identifier of the session. It must be omitted. It will be assigned by the mapping module
- `local_address`:
  The local ip address. If it is omitted the default is any address.
- `local_port`:
  The local port number. If omitted a random port number will be used.
- `dest_address` and `dest_port`:
  Contains the address and port number of the remote host. If specified this address will be the default remote address for the session.

The mapping module answers the open request with `ASP_RTP_Open_session_result` message. That message contains the parameters of the new session. The *session_id.id* is the unique identifier of the session. It will identify the session during sending and receiving data.

Close Session

The closing of the session is requested by the `ASP_RTP_Close_session` message. The *session_id.id* contains the session identifier.

Payload Generation

Comfort Noise

The following function generates a comfort noise payload according to [5].

Name	Type of formal parameters	Type of return value
`f_generate_comfort_noise`	(in integer level, in Coefficient_list coefficients)	octetstring;

Parameters:

- level: Noise level value
- coefficients: List of reflection coefficients

Telephony Events and DTMF Codes

The following functions generates a telephony event and DTMF codes payload according to [6].

Name	Type of formal parameters	Type of return value
`f_generate_tones_events`	(in Tones_DTMFs events_dtmfs)	octetstring;

Parameter:

- events_dtmfs: List of DTMF digits, events or tones.

Codec Handling

The demo program (*example.ttcn*) introduces many examples of payload generation between two UDP testports. The following functions read and write samples from/to files:

Name	Type of formal parameters	Type of return value
f_Init_Media_Fileinfo	(in charstring pl_filename, in integer pl_block_size, in InitOperType pl_init_type, in RTP_MediaType pl_media_type, in integer pl_blockno, in integer pl_nof_blocks, inout RTP_FileInfo pl_fileinfo)	none
f_Get_Media_Content	(inout RTP_FileInfo pl_filename)	octetstring
f_Put_Media_Content	(in RTP_FileInfo pl_fileinfo, in octetstring data)	integer
f_INIT_CODEC	(in charstring pl_filename, in integer pl_block_size, in InitOperType pl_init_type)	integer
f_GET_CONTENT	(in integer pl_fd, in integer pl_blockno, in integer pl_nof_blocks_to_read, in integer pl_header_offset)	octetstring
f_PUT_CONTENT	(in integer pl_fd, in integer pl_blockno, in octetstring pl_stream, in integer pl_header_offset)	octetstring
f_CLOSE_CODEC	(in integer pl_fd)	none
f_RTP_Hdr_enc	(in Media_RTP_Header hdr)	octetstring

Example Code

```
module example{
    modulepar {
        integer BLOCK_SIZE := 4;
        integer BLOCK_NO := 0;
        integer NOF_BLOCKS_TO_READ := 6;
    }
    import from UDPasp_Types all;
    import from UDPasp_PortType all;
    import from RTP_Types all;
    import from RTP_Mapping all;
    import from RTP_File_Types all;
    import from RTP_Media all;

    template ASP_RTP_Open_session_result t_open_res:=?;
    template ASP_RTP_message t_message:=?;

    function TWAIT( in integer sec ) runs on test_comp
    {
        timer T_WAIT;
        T_WAIT.start(int2float(sec));
```

```
        T_WAIT.timeout;
    }
    type component test_comp{
        var RTP_mapping_CT v_mapping_comp;
        var RTP_mapping_CT v_mapping_comp2;

        port RTPasp_PT RTP1;
        port RTPasp_PT RTP2;
    }

    type component system_comp{
        port UDPasp_PT UDP1;
        port UDPasp_PT UDP2;
    }

    function make_pattern_file(in charstring FileName1,
                               in charstring FileName2,
                               in Media_RTP_Header mrh,
                               inout ASP_RTP_message v_message)
                               runs on test_comp
    {
        var ASP_RTP_message v_message1;
    // Init codec
        var integer v_codec1:=f_INIT_CODEC(FileName1, BLOCK_SIZE, OPEN);
        var integer v_codec2:=f_INIT_CODEC(FileName2, BLOCK_SIZE, CREATE);

    // Send and receive message
        var integer hdr_size := f_PUT_CONTENT(v_codec2,0,f_RTP_Hdr_enc(mrh),0)
        var integer block_no := BLOCK_NO;
        var boolean next := true;
        do {
            // read samples from file
            v_message.data.rtp.data :=
                f_GET_CONTENT(v_codec1, block_no, NOF_BLOCKS_TO_READ, 0);
            if ( v_message.data.rtp.data!=''O  ) {
                RTP1.send(v_message);
                RTP2.receive(t_message) -> value v_message1;
            if (f_PUT_CONTENT(v_codec2, block_no,
                              v_message1.data.rtp.data, hdr_size) <
                NOF_BLOCKS_TO_READ*BLOCK_SIZE) {
                    next := false;
                }
                else {
                    block_no := block_no + NOF_BLOCKS_TO_READ;
                }
            }
            else { next := false; }
        } while (next);

    // Close codec
        f_CLOSE_CODEC(v_codec1);
```

```
        f_CLOSE_CODEC(v_codec2);
    }

    function send_receive_file( in charstring FileName1,
                                in charstring FileName2,
                                in boolean HEADER,
                                in RTP_MediaType mt,
                                inout ASP_RTP_message v_message)
                                runs on test_comp
    {
        var ASP_RTP_message v_message1;
        var RTP_FileInfo FileInfo1, FileInfo2;
        f_Init_Media_Fileinfo( FileName1, BLOCK_SIZE, OPEN,  mt, BLOCK_NO,
                               NOF_BLOCKS_TO_READ, FileInfo1);
        f_Init_Media_Fileinfo( FileName2, BLOCK_SIZE, CREATE, mt, BLOCK_NO,
                               NOF_BLOCKS_TO_READ, FileInfo2);
        var boolean next := true;
        do {
            v_message.data.rtp.data := f_Get_Media_Content( FileInfo1 );
            if ( v_message.data.rtp.data!=''O  ) {
                RTP1.send(v_message);
                RTP2.receive(t_message) -> value v_message1;
                // HEADER == 0 : get file without payload header (.org)
                if ( HEADER==false ) {
                    FileInfo2.headerOffset := 0;
                    FileInfo2.headerOp := false;
                }
                if (f_Put_Media_Content(FileInfo2,v_message1.data.rtp.data)==0){
                    next := false;
                }
                else {
                FileInfo1.block_no :=
                    FileInfo1.block_no + FileInfo1.nof_blocks_to_read;
                FileInfo2.block_no :=
                    FileInfo2.block_no + FileInfo2.nof_blocks_to_read;
                }
            }
            else { next := false; }
        } while (next);
        f_CLOSE_CODEC(FileInfo1.fd);
        f_CLOSE_CODEC(FileInfo2.fd);
    }

    testcase TC() runs on test_comp system system_comp {
        var RTP_session_par v_session_par;
        var ASP_RTP_Open_session v_open, v_open2;
        var ASP_RTP_Open_session_result v_open_res, v_open_res2;
        var ASP_RTP_message v_message,v_message1,v_message2;
        var ASP_RTP_Close_session v_close;

    // Create and start mapping components
```

```
        v_mapping_comp:=RTP_mapping_CT.create;
        map(v_mapping_comp:UDP_PCO, system:UDP1);
        connect(self:RTP1,v_mapping_comp:RTP_SP_PCO);
        v_mapping_comp.start(f_RTP_EncDec_Mapping());

        v_mapping_comp2:=RTP_mapping_CT.create;
        map(v_mapping_comp2:UDP_PCO, system:UDP2);
        connect(self:RTP2,v_mapping_comp2:RTP_SP_PCO);
        v_mapping_comp2.start(f_RTP_EncDec_Mapping());

    // Open Session
        v_session_par.id:=omit;
        v_session_par.local_address:="localhost";
        v_session_par.local_port:=5679;
        v_session_par.dest_address:="localhost";
        v_session_par.dest_port:=5060;
        v_open.session_id:=v_session_par;

        RTP1.send(v_open);
        RTP1.receive(t_open_res) -> value v_open_res;

    // Open Session2
        v_session_par.id:=omit;
        v_session_par.local_address:="localhost";
        v_session_par.local_port:=5060;
        v_session_par.dest_address:="localhost";
        v_session_par.dest_port:=5679;
        v_open2.session_id:=v_session_par;

        RTP2.send(v_open2);
        RTP2.receive(t_open_res) -> value v_open_res2;

    // Set message
        v_session_par:=v_open_res.session_id;
        v_session_par.local_address:=omit;
        v_session_par.local_port:=omit;
        v_session_par.dest_address:="localhost";
        v_session_par.dest_port:=5060;

        v_message.session_id:=v_session_par;
        v_message.data:={
            rtp:={ version:=2,
                   padding_ind:= '0'B,
                   extension_ind:='0'B,
                   CSRC_count:=0,
                   marker_bit:='0'B,
                   payload_type:=11,
                   sequence_number:=52134,
                   time_stamp:='11110000101010101100110000001111'B,
                   SSRC_id:='00001111010101010011001111110000'B,
                   CSRCs:={'11110000101010101100110000001111'B},
```

```
                    ext_header:=omit
                }
            };

/*************************************************************************/
    // make a pattern JPEG file with RTP payload header
        var Media_RTP_Header mrh1 := {
            jpeg_rtp_hdr := {{1,0,3,2,51,6},
                             {1,'0'B,'1'B,2},
                             {9,8,11,{250,134,255,99,1,23,99,45,32,2,8}}
                            }};
        make_pattern_file("sample.media", "jpeg_pattern.dat", mrh1, v_message);

    // Send and receive JPEG media file
        send_receive_file("jpeg_pattern.dat", "jpeg_rtp.dat",
                          true, JPEG, v_message);
        send_receive_file("jpeg_pattern.dat", "jpeg_rtp.org",
                          false, JPEG, v_message);

/*************************************************************************/
    // make a pattern H263 file with RTP payload header
        var Media_RTP_Header mrh2 := {
                             h263_rtp_hdr := {'0'B,'1'B,'011'B,'100'B,
                                              '001'B,'0'B,'1'B,'1'B,'0'B,
                                              '0110'B,'10'B,'101'B,
                                              '01101011'B}
                                    };
        make_pattern_file("sample.media", "h263_pattern.dat", mrh2, v_message);
    // Send and receive H263 media file
        send_receive_file("h263_pattern.dat", "h263_rtp.dat",
                          true, H263, v_message);
        send_receive_file("h263_pattern.dat", "h263_rtp.org",
                          false, H263, v_message);

/*************************************************************************/
    // Send and receive MPEG4 media file
        send_receive_file("sample.media", "mpeg4_rtp.dat",
                          true, MPEG4, v_message);
/*************************************************************************/

    // Close the session
        v_close.session_id:=v_session_par;
        RTP1.send(v_close);
        RTP2.send(v_close);

        TWAIT(1);

        v_mapping_comp.stop;
        disconnect(self:RTP1,v_mapping_comp:RTP_SP_PCO);
        unmap(v_mapping_comp:UDP_PCO, system:UDP1);
```

```
        v_mapping_comp2.stop;
        disconnect(self:RTP2,v_mapping_comp2:RTP_SP_PCO);
        unmap(v_mapping_comp:UDP_PCO, system:UDP2);
    }

    control{
        execute(TC());
    }
}
```

Terminology

No specific terminology is used.

Abbreviations

ASP
: Abstract Service Primitive

RTP
: Real-time Transport Protocol

RTCP
: RTP Control Protocol

TTCN-3
: Testing and Test Control Notation version 3

UDP
: User Datagram Protocol

References

[1] ETSI ES 201 873-1 v.3.1.1 (2005-06)
The Testing and Test Control Notation version 3. Part 1: Core Language

[2] User Guide for TITAN TTCN–3 Test Executor

[3] RFC 3550
RTP: A Transport protocol for Real-Time Applications

[4] RFC 3389
Real-time Transport Protocol (RTP) Payload for Comfort Noise (CN)

[5] RFC 2833

RTP Payload for DTMF Digits, Telephony Tones and Telephony Signals

[6] RFC 2190
RTP Payload Format for H.263 Video Streams

[7] RFC 2435
RTP Payload Format for JPEG-compressed Video

[8] Function Description for the SRTP Protocol Module