IP Protocol Modules for TTCN-3 Toolset with TITAN, Description

Gábor Szalai

Version 1551-CNL 113 418, Rev. C, 2015-07-30

Table of Contents

General

Protocol modules implement the message structures 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 [3] and correctly encoding/decoding messages when executing test suites using the TITAN TTCN-3 test environment.

Protocol modules are using TITAN's RAW encoding attributes [4] and hence are usable with the TITAN test toolset only.

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 version R7A (1.7.pl0) or higher installed. For installation guide see [4].

NOTE	This version of the test port is not compatible with TITAN releases earlier than R7A.

Functionality

Protocol Version Implemented

This set of protocol modules implements protocol messages and constants of the IP protocol, version 4 (see [1]) and version 6 (see [2]) with the modifications specified in Modifications/Deviations Related to the Protocol Specification.

The following IPv4 extension headers are supported:

- authentication header (***AH***), see [6]
- encapsulating security payload (***ESP***), see [7]
- generic routing encapsulation (***GRE*** and ***GRE2***), see [8] and [9]

Modifications/Deviations Related to the Protocol Specification

The IPv4 options at the end of the header are not supported.

Encoding/Decoding and Other Related Functions

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

Function `f_IPv4_enc_eth` puts padding zeros to the end of the encoded IP message if shorter than 46 bytes, it assures the minimal length of payload in case of Ethernet frame is used as lower layer.

Implemented encoding/decoding functions:

Name	Type of formal parameters	Type of return value
`f_IPv4_enc`	(IPv4_packet)	returns octetstring
`f_IPv4_enc_eth`	(IPv4_packet)	returns octetstring
`f_IPv4_dec`	(octetstring)	returns IPv4_packet
`f_IPv4_dec_backtrack`	(octetstring, IPv4_packet,boolean)	returns integer
`f_IPv6_enc`	(IPv6_packet)	returns octetstring
`f_IPv6_dec`	(octetstring)	returns IPv6_packet
`f_IPv6_dec_backtrack`	(octetstring, IPv6_packet,boolean)	returns integer

NOTE In general, the length values are automatically calculated regardless of user input. One exception is the `exthdr_length` field where the value is automatically calculated only if the tester uses the dummy value `-1`. If the tester uses values from `0` to `255` for `exthdr_length` then the user-defined value will be encoded.

The 3rd parameter of the backtrack decoders control the payload length check. If the parameter is `true`, the length of the payload part and the payload length in the IP header are compared and the decoding will fail if they are not equal. If the parameter is `false` the length check if not enforced, which is useful for handling truncated IP datagrams.

The default value of the parameter is controlled by the `tsp_use_strict_length_check` module parameter, which default value is `true`.

The product also provides some additional functionality to the user via the following functions:

- The `f_IPv4_checksum()` can be used to calculate the IPv4 checksum over an already encoded IPv4 packet.
- The `f_IPv4_addr_enc()` and `f_IPv4_addr_dec()` functions can be used to convert IPv4 addresses from character string format to encoded octetstring format and vice versa.
- The `f_IPv6_addr_enc` and `f_IPv6_addr_dec` can be used for IPv6 addresses. The functions with backtrack postfix are giving back `0` integer if the decoding failed and `1` if it was successful. The decoded value will be in the second parameter.

The functions return an empty string if the conversion of the address is not possible.

Name	Type of formal parameters	Type of return value
`f_IPv4_checksum`	(octetstring)	returns OCT2
`f_IPv4_addr_enc`	(IPV4ADDR)	returns octetstring
`f_IPv4_addr_dec`	(octetstring)	returns IPV4ADDR
`f_IPv6_addr_enc`	(charstring)	returns octetstring
`f_IPv6_addr_dec`	(octetstring)	returns charstring

Usage

Installation

The set of protocol modules can be used in 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 IP 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 [4].

Configuration

None.

Implementation Specifics

The `f_IPv4_checksum()` can be used to calculate the value of the IPv4 checksum field. The parameter of the function is the encoded IP packet. The checksum is calculated over the ***IP*** header and the `return` value is the value of the IP checksum field. The length of the checksum field is always 2 octets.

The `f_IPv4_addr_enc()` and `f_IPv4_addr_dec()` functions can be used to convert IPv4 addresses from character string format to encoded octetstring format and vice versa. The `return` value is the value of the source or destination IPv4 address field. The length of the address field is always 4 octets.

The `IPv4_ASP` ASP is a very basic ASP, containing:

- the IPv4 packet
- a boolean flag, whether the IPv4 checksum should be calculated or not. The flag can be used to perform the IPv4 checksum calculation, when sending an IP packet.

Function `f_IPv4_enc_eth` puts padding zeros to the end of the encoded IP message if shorter than 46 bytes, it assures the minimal length of payload in case of Ethernet frame is used as lower layer.

Examples

IPv4 Packet Encoding and Decoding

The following example shows how an IPv4 packet can be encoded and decoded:

```
var IPv4_ASP v_ipv4_asp;
var IPv4_packet v_ipv4_packet;
var octetstring data;

data:= f_IPv4_enc(v_ipv4_asp.ipv4_packet);
if (v_ipv4_asp.cksum_calc) {
var OCT2 cksum := f_IPv4_checksum(data);
// Copy the calculated checksum into the encoded data.
// The checksum field is on the 11th and 12nd octet.
  data[10] := cksum[0];
  data[11] := cksum[1];
}

v_ipv4_packet := f_IPv4_dec(data);
```

IPv4 Packet Encoding for Ethernet Support

The following example shows how an IPv4 packet can be encoded to ensure the minimal payload length for Ethernet:

```
var IPv4_ASP v_ipv4_asp;
var IPv4_packet v_ipv4_packet;
var octetstring data;

data:= f_IPv4_enc_eth(v_ipv4_asp.ipv4_packet);
var OCT2 cksum := f_IPv4_checksum(data);
data[10] := cksum[0];
data[11] := cksum[1];
```

IPv6 Packet Encoding and Decoding

The following example shows how an IPv4 packet can be encoded and decoded:

```
var IPv6_packet v_ipv6_packet;
var octetstring data;

data:= f_IPv6_enc(v_ipv6_packet);

v_ipv6_packet := f_IPv6_dec(data);
```

IPv4 Address Encoding and Decoding

The following example shows how the IPv4 address fields can be filled up:

```
var IPv4_packet v_ipv4_packet;
var charstring v_address := "192.168.0.1";

v_ipv4_packet.header.srcaddr := f_IPv4_addr_enc(v_address);

v_address := f_IPv4_addr_dec(v_ipv4_packet.header.srcaddr);
```

IPv6 Address Encoding and Decoding

The following example shows how the IPv6 address fields can be filled up:

```
var IPv6_packet v_ipv6_packet;
var charstring v_address := "2001:3ab5:5566:1234::1";

v_ipv6_packet.header.srcaddr := f_IPv6_addr_enc(v_address);

v_address := f_IPv6_addr_dec(v_ipv6_packet.header.srcaddr);
```

Terminology

No specific terminology is used.

Abbreviations

IP

Internet Protocol

IPv4

Internet Protocol version 4

IPv6

Internet Protocol version 6

RFC

Request For Comments

TTCN-3

Testing and Test Control Notation version 3

References

[1] RFC 791
Internet Protocol, Version 4 (IPv4)

[2] RFC 2460
Internet Protocol, Version 6 (IPv6)

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

[4] User Documentation for the TITAN TTCN-3 Test Executor

[5] RFC 2004
Minimal Encapsulation within IP

[6] RFC 2402
IP Authentication Header

[7] RFC 2406
IP Encapsulating Security Payload (ESP)

[8] RFC 2784
Generic Routing Encapsulation (GRE)

[9] RFC 2890
Key and Sequence Number Extension to GRE