|  |  |
| --- | --- |
|  |  |

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

# Abstract

This is the description for the IP protocol module. The protocol module is developed for the TTCN-3 Toolset with Titan. This document should be read together with Product Revision Information [4].

**Contents**

[1 General 2](#_Toc426023713)

[1.1 System Requirements 2](#_Toc426023714)

[2 Functionality 2](#_Toc426023715)

[2.1 Protocol version implemented 2](#_Toc426023716)

[2.2 Modifications/deviations related to the protocol specification 2](#_Toc426023717)

[2.3 Encoding/Decoding and Other Related Functions 2](#_Toc426023718)

[3 Usage 4](#_Toc426023719)

[3.1 Installation 4](#_Toc426023720)

[3.2 Configuration 4](#_Toc426023721)

[3.3 Implementation specifics 4](#_Toc426023722)

[3.4 Examples 5](#_Toc426023723)

[3.4.1 IPv4 packet encoding and decoding 5](#_Toc426023724)

[3.4.2 IPv4 packet encoding for Ethernet support 5](#_Toc426023725)

[3.4.3 IPv6 packet encoding and decoding 5](#_Toc426023726)

[3.4.4 IPv4 address encoding and decoding 5](#_Toc426023727)

[3.4.5 IPv6 address encoding and decoding 6](#_Toc426023728)

[4 Terminology 6](#_Toc426023729)

[5 References 6](#_Toc426023730)

[6 Abbreviations 6](#_Toc426023731)

# 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 ‎[5] 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 [5] ‎. Please 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 2.2‎.

The following IPv4 extension headers are supported:

* authentication header (AH), see ‎[7]
* encapsulating security payload (ESP), see ‎[8]
* generic routing encapsulation (GRE and GRE2), see ‎[9] and ‎[10]

## 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 that 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. For IPv6 addresses, you can use f\_IPv6\_addr\_enc and f\_IPv6\_addr\_dec. 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 [5].

## 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.

# 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. 109 21-CNL 113 418-10  
   IP Protocol Modules for TTCN-3 Toolset with TITAN, Product Revision Information
5. 1/1553-CRL 113 200/5 Uen  
   User Documentation for the TITAN TTCN-3 Test Executor
6. RFC 2004 – Minimal Encapsulation within IP
7. RFC 2402 – IP Authentication Header
8. RFC 2406 – IP Encapsulating Security Payload (ESP)
9. RFC 2784 – Generic Routing Encapsulation (GRE)
10. RFC 2890 – Key and Sequence Number Extension to GRE

# 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