



## **GTP Tunnel Encapsulation Example**

The following example demonstrates GTPv1 encapsulation which must be declared as a custom header:

```
header gtp_v1_h {
    bit<3>      version;          /** For GTPv1, this has a value of 1. */
    bit        protocol_type;    /** GTP (value 1) from GTP' (value 0) */
    bit        reserved;
    bit        extension_header_flag; /** extension header optional field. */
    bit        seq_number_flag;  /** Sequence Number optional field */
    bit        n_pdu_number_flag; /** N-PDU number optional field */
    bit<8>     message_type;     /** types of messages are defined in
3GPP TS 29.060 section 7.1 */
    bit<16>    message_length;   /** length of the payload in bytes */
    bit<32>    teid;            /** Tunnel endpoint identifier */
    bit<16>    sequence_number;  /** optional */
    bit<8>     n_pdu_number;     /** optional */
    bit<8>     next_extension_hdr_type; /** optional if any of the E, S, or PN
bits are on. The field must be interpreted only if the E bit is on */
}
```

The custom GTPv1 header can optionally be added to the parse graph (see [Custom Parser States](#)) is optional, and is only required if any of the custom header fields need to be read from or modified after encapsulation.

```
struct headers_t {
    NV_FIXED_HEADERS
    gtp_v1_h    gtpv1;
}

parser packet_parser(packet_in packet, out headers_t headers) {
    NV_FIXED_PARSER(packet, headers)

    @nv_transition_from("nv_parse_udp", GTP_U_PORT)
    state parse_gtp
```

```

    {
        packet.extract(headers.gtpv1);
        transition nv_parse_inner_ipv4;
    }
}

```

The extern object `NvTunnelTemplate<HEADER_TYPE>` requires a struct type, for which the entire underlay header is defined as a struct:

```

struct tunnel_headers_t {
    nv_ethernet_h ethernet;
    nv_ipv4_h      ipv4;
    nv_udp_h       udp;
    gtp_v1_h       gtpv1;
}

```

Finally, when declaring a `NvTunnelTemplate` object, the annotation `@nv_tunnel_fields` must be present. This annotation contains a key-value entry for each header in the specified struct type. Under each header, each header field must be specified in the order that the field appears in the header. For the GTPv1 example, a `NvTunnelTemplate` would be declared in the main control:

```

@nv_tunnel_fields(
    ethernet = {
        dst_addr = "variable",
        src_addr = "variable",
        ether_type = 0x0800
    },
    ipv4 = {
        version = 0x4,
        ihl = 0x5,
        diffserv = 0,
        ecn = "ignore", // Cannot set
    }
)

```

```

total_len = "ignore",          // reparse will calculate, cannot set
identification = "ignore",    // Cannot set
flags = 0,
frag_offset = 0,
ttl = 64,
protocol = 17,
hdr_checksum = "ignore",      // reparse will calculate, cannot set
src_addr = "variable",
dst_addr = "variable"
},
udp = {
    src_port = "ignore",        // reparse will set, cannot be set manually
    dst_port = "variable",
    length = "ignore",          // reparse will calculate, cannot set
    checksum = "ignore"        // Cannot set
},
gtpv1 = {
    version = 1,
    protocol_type = 1,
    reserved = 0,
    extension_header_flag = 1,
    seq_number_flag = 1,
    n_pdu_number_flag = 1,
    message_type = 0xFF,       // T-PDU
    message_length = "variable",
    teid = "variable",
    sequence_number = 0,
    n_pdu_number = 0,
    next_extension_hdr_type = 0
}
)
NvTunnelTemplate<tunnel_headers_t>() gtpv1Tunnel;

```

The type `tunnel_headers_t` determines the required structure of the annotation. Fields marked "variable" will be determined according to the arguments passed to

`nv_set_l2tunnel_underlay` or `nv_set_l3tunnel_underlay`. Some fields will always be overwritten by reparse, such as IPv4 length, IPv4 checksum, UDP length, etc which must be calculated and cannot be set by the user. Such fields can be marked "ignore". Marking other fields "ignore", would set them to zero.

Finally, the instantiated `NvTunnelTemplate` object can be used in one of the custom tunnel externs. For example:

```
action forward(nv_ipv4_addr_t src_addr) {
    nv_set_l3tunnel_underlay(headers, gtpv1Tunnel, {
        48w0x001A2B3C4D5E, // ethernet.dst_addr
        48w0x00F1E2D3C4B5, // ethernet.src_addr
        src_addr,           // ipv4.src_addr
        32w0xC07B7B01,     // ipv4.dst_addr
        16w0x868,           // udp.dst_port for GTP-U
        16w0x4,             // gtpv1.message_length
        32w0x1234567        // gtpv1.teid
    });
    nv_send_to_port(1);
}
```

The third argument to `nv_set_l3tunnel_underlay` is required to be a list expression where each element corresponds to a header field marked "variable" in the `NvTunnelTemplate` annotation. The order of the values in the list is the order in which the header fields appear in the packet.

[Full example of gtp\\_tunnel.p4:](#)

```
/*
 * SPDX-FileCopyrightText: Copyright (c) 2025 NVIDIA CORPORATION & AFFILIATES. All rights reserved.
 * SPDX-License-Identifier: LicenseRef-NvidiaProprietary
 *
 * NVIDIA CORPORATION, its affiliates and licensors retain all intellectual
 * property and proprietary rights in and to this material, related
 * documentation and any modifications thereto. Any use, reproduction,
 * disclosure or distribution of this material and related documentation
 * without an express license agreement from NVIDIA CORPORATION or
```

```
* its affiliates is strictly prohibited.
```

```
*/
```

```
#include <doca_model.p4>
```

```
#include <doca_headers.p4>
```

```
#include <doca_extensns.p4>
```

```
#include <doca_parser.p4>
```

```
#define GTP_U_PORT 2152
```

```
header gtp_v1_h {
```

```
    bit<3>          version;                /** For GTPv1, this has a value of 1. */
```

```
    bit            protocol_type;          /** GTP (value 1) from GTP' (value 0) */
```

```
    bit            reserved;
```

```
    bit            extension_header_flag;  /** extension header optional field. */
```

```
    bit            seq_number_flag;       /** Sequence Number optional field */
```

```
    bit            n_pdu_number_flag;     /** N-PDU number optional field */
```

```
    bit<8>         message_type;          /** types of messages are defined in
```

```
3GPP TS 29.060 section 7.1 */
```

```
    bit<16>        message_length;        /** length of the payload in bytes */
```

```
    bit<32>        teid;                   /** Tunnel endpoint identifier */
```

```
    bit<16>        sequence_number;       /** optional */
```

```
    bit<8>         n_pdu_number;          /** optional */
```

```
    bit<8>         next_extension_hdr_type; /** optional if any of the E, S, or PN
```

```
bits are on. The field must be interpreted only if the E bit is on */
```

```
}
```

```
struct headers_t {
```

```
    NV_FIXED_HEADERS
```

```
    gtp_v1_h    gtpv1;
```

```
}
```

```
parser packet_parser(packet_in packet, out headers_t headers) {  
    NV_FIXED_PARSER(packet, headers)
```

```
    @nv_transition_from("nv_parse_udp", GTP_U_PORT)
```

```
    state parse_gtp
```

```

    {
        packet.extract(headers.gtpv1);
        transition nv_parse_inner_ipv4;
    }
}

struct tunnel_headers_t {
    nv_ethernet_h ethernet;
    nv_ipv4_h      ipv4;
    nv_udp_h       udp;
    gtp_v1_h       gtpv1;
}

control c(
    inout headers_t headers,
    in  nv_standard_metadata_t std_meta,
    inout nv_empty_metadata_t user_meta,
    inout nv_empty_metadata_t pkt_out_meta
) {
    @nv_tunnel_fields(
        ethernet = {
            dst_addr = "variable",
            src_addr = "variable",
            ether_type = 0x0800
        },
        ipv4 = {
            version = 0x4,
            ihl = 0x5,
            diffserv = 0,
            ecn = "ignore", // cannot set
            total_len = "ignore", // HW will calculate, cannot set
            identification = "ignore", // cannot set
            flags = 0,
            frag_offset = 0,
            ttl = 64,
            protocol = 17,

```

```

        hdr_checksum = "ignore", // HW will calculate, cannot set
        src_addr = "variable",
        dst_addr = "variable"
    },
    udp = {
        src_port = "ignore", // HW will calculate entropy, cannot set
        dst_port = GTP_U_PORT,
        length = "ignore", // HW will calculate, cannot set
        checksum = "ignore" // cannot set
    },
    gtpv1 = {
        version = 1,
        protocol_type = 1,
        reserved = 0,
        extension_header_flag = 1,
        seq_number_flag = 1,
        n_pdu_number_flag = 1,
        message_type = 0xFF,
        message_length = "variable",
        teid = "variable",
        sequence_number = 123,
        n_pdu_number = 255,
        next_extension_hdr_type = 0
    }
)
NvTunnelTemplate<tunnel_headers_t>() gtpv1Tunnel;

action drop() {
    nv_drop();
}

action forward(nv_logical_port_t port) {
    nv_send_to_port(port);
}

action tunnel(nv_ipv4_addr_t dst_addr, bit<32> teid) {

```



```

    nv_set_l3tunnel_underlay(headers, gtpv1Tunnel, {
        48w0x1, 48w0x2, 32w0x3, dst_addr,
        16w4, /* 4 bytes of optional GTP-U payload */
        teid
    });
}

table ip_as_key {
    key = {
        headers.ipv4.src_addr : exact;
    }
    actions = {
        NoAction;
        drop;
        forward;
        tunnel;
    }
    size = 128;
    default_action = NoAction;
    const entries = {
        (32w0x01010101) : tunnel(32w0x11111111, 1);
        (32w0x02020202) : tunnel(32w0x22222222, 2);
        (32w0x03030303) : tunnel(32w0x33333333, 3);
    }
}

table teid_as_key {
    key = {
        headers.gtpv1.teid : exact;
    }
    actions = {
        NoAction;
        drop;
        forward;
    }
    size = 128;
}

```

```

        default_action = forward(3);
    const entries = {
        (32w0x01) : forward(1);
        (32w0x02) : forward(2);
    }
}

apply {
    ip_as_key.apply();
    if (headers.gtpv1.isValid()) {
        teid_as_key.apply();
    }
    forward(3);
}
}

NvDocaPipeline(
    packet_parser(),
    c()
) main;

```

**Notice**  
This document is provided for information purposes only and shall not be regarded as a warranty of a certain functionality, condition, or quality of a product. NVIDIA Corporation (“NVIDIA”) makes no representations or warranties, expressed or implied, as to the accuracy or completeness of the information contained in this document and assumes no responsibility for any errors contained herein. NVIDIA shall have no liability for the consequences or use of such information or for any infringement of patents or other rights of third parties that may result from its use. This document is not a commitment to develop, release, or deliver any Material (defined below), code, or functionality. NVIDIA reserves the right to make corrections, modifications, enhancements, improvements, and any other changes to this document, at any time without notice. Customer should obtain the latest relevant information before placing orders and should verify that such information is current and complete. NVIDIA products are sold subject to the NVIDIA standard terms and conditions of sale supplied at the time of order acknowledgement, unless otherwise agreed in an individual sales agreement signed by authorized representatives of NVIDIA and customer (“Terms of Sale”). NVIDIA hereby expressly objects to applying any customer general terms and conditions with regards to the purchase of the NVIDIA product referenced in this document. No contractual obligations are formed either directly or indirectly by this document. NVIDIA products are not designed, authorized, or warranted to be suitable for use in medical, military, aircraft, space, or life support equipment, nor in applications where failure or malfunction of the NVIDIA product can reasonably be expected to result in personal injury, death, or property or environmental damage. NVIDIA accepts no liability for inclusion and/or use of NVIDIA products in such equipment or applications and therefore such inclusion and/or use is at customer’s own risk. NVIDIA makes no representation or warranty that products based on this document will be suitable for any specified use. Testing of all parameters of each product is not necessarily performed by NVIDIA. It is customer’s sole responsibility to evaluate and determine the applicability of any information contained in this document, ensure the product is suitable and fit for the application planned by customer, and perform the necessary testing for the application in order to avoid a default of the

application or the product. Weaknesses in customer's product designs may affect the quality and reliability of the NVIDIA product and may result in additional or different conditions and/or requirements beyond those contained in this document. NVIDIA accepts no liability related to any default, damage, costs, or problem which may be based on or attributable to: (i) the use of the NVIDIA product in any manner that is contrary to this document or (ii) customer product designs.

No license, either expressed or implied, is granted under any NVIDIA patent right, copyright, or other NVIDIA intellectual property right under this document. Information published by NVIDIA regarding third-party products or services does not constitute a license from NVIDIA to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property rights of the third party, or a license from NVIDIA under the patents or other intellectual property rights of NVIDIA.

Reproduction of information in this document is permissible only if approved in advance by NVIDIA in writing, reproduced without alteration and in full compliance with all applicable export laws and regulations, and accompanied by all associated conditions, limitations, and notices.

THIS DOCUMENT AND ALL NVIDIA DESIGN SPECIFICATIONS, REFERENCE BOARDS, FILES, DRAWINGS, DIAGNOSTICS, LISTS, AND OTHER DOCUMENTS (TOGETHER AND SEPARATELY, "MATERIALS") ARE BEING PROVIDED "AS IS." NVIDIA MAKES NO WARRANTIES, EXPRESSED, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE MATERIALS, AND EXPRESSLY DISCLAIMS ALL IMPLIED WARRANTIES OF NONINFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A PARTICULAR PURPOSE. TO THE EXTENT NOT PROHIBITED BY LAW, IN NO EVENT WILL NVIDIA BE LIABLE FOR ANY DAMAGES, INCLUDING WITHOUT LIMITATION ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL, PUNITIVE, OR CONSEQUENTIAL DAMAGES, HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF ANY USE OF THIS DOCUMENT, EVEN IF NVIDIA HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. Notwithstanding any damages that customer might incur for any reason whatsoever, NVIDIA's aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms of Sale for the product.

**Trademarks**

NVIDIA and the NVIDIA logo are trademarks and/or registered trademarks of NVIDIA Corporation in the U.S. and other countries. Other company and product names may be trademarks of the respective companies with which they are associated.

Copyright 2025. PDF Generated on 04/24/2025