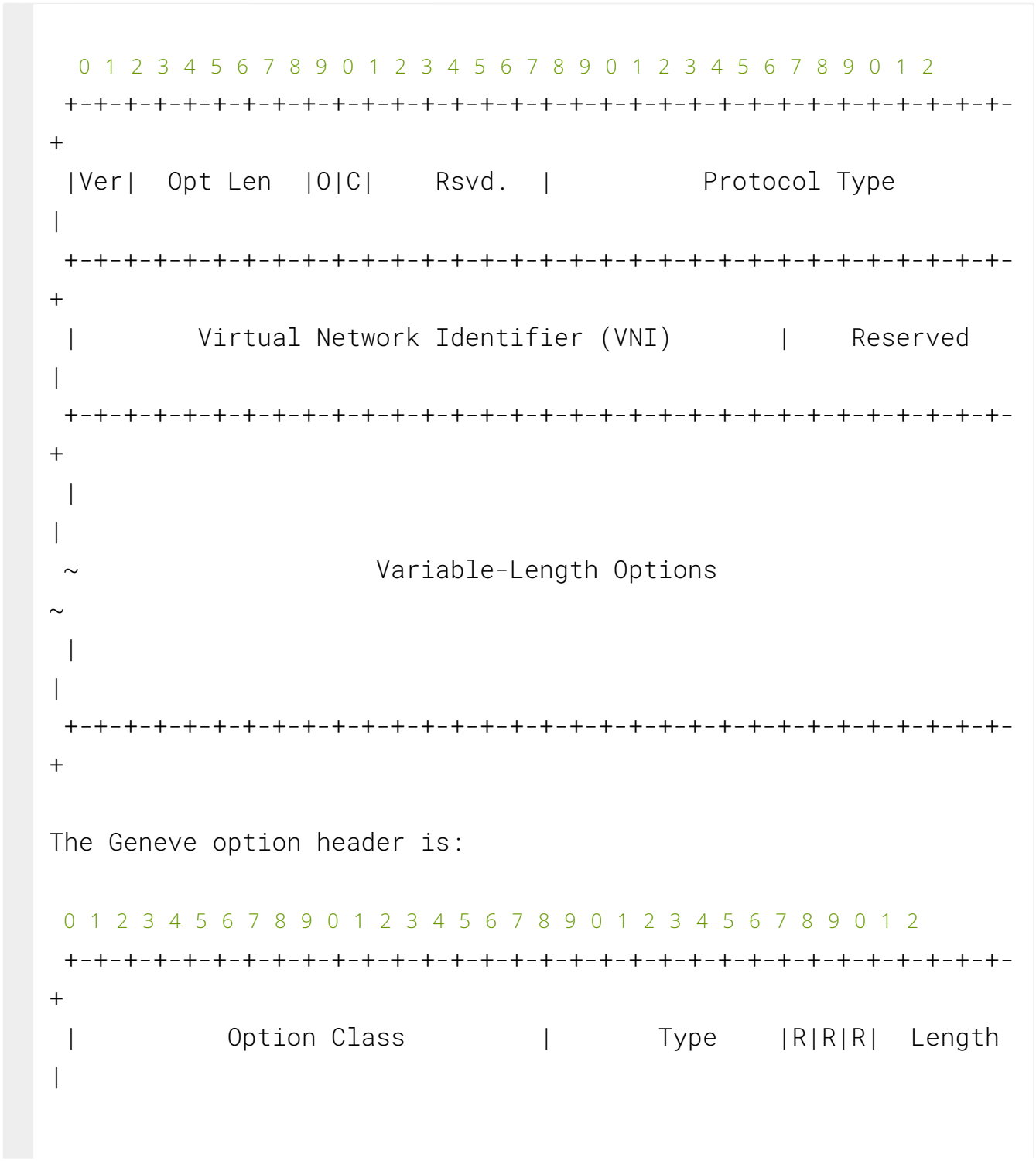
Geneve TLV Parsing Example

Table of contents

This example demonstrates how to add a custom protocol header that supports optional Type-Length-Value (TLV) fields. In the P4-16 specification, there is no first class support in the parser primitives, and the user needs to manually.

Geneve is short for Generic Network Virtualization Encapsulation, a common L2 tunnel header. The Geneve header format is:



```
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
+
 |               ~ Variable-Length Option Data ~                 |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
+
```

Sample Code

First we define the Geneve header, per RFC 8926.

```
header geneve_t {
    bit<2> ver;
    bit<6> opt_len;
    bit<1> o;
    bit<1> c;
    bit<6> reserved1;
    bit<16> protocol_type;
    bit<24> vni;
    bit<8> reserved2;
};
```

Next we define a struct that contains the base Geneve Option header, shared by all option types.

```
// Struct so it can be a field within other headers
struct geneve_option_t {
    bit<24> option_class_type;
    bit<3> reserved;
    bit<5> length;
};
```

Lastly, depending on the option class and type, the variable length option data can be defined. In this example, we model the Geneve options used by the P4 Inband Telemetry (INT) specification.

```
// Intermediate nodes (INT Transit Hops) must process this type of INT Header.
header geneve_option_int_md_t {
    geneve_option_t base;
    bit<4> version;
    bit<1> discard;
    bit<1> exceeded_max_hops;
    bit<1> mtu_exceeded;
    bit<12> reserved;
    bit<5> hop_ml;
    bit<8> remaining_hop_count;
    bit<16> instruction_bitmap;
    bit<16> ds_id;
    bit<16> ds_instruction;
    bit<16> ds_flags;
};
// Destination headers can be used to enable Edge-to-Edge communication between
// the INT Source and INT Sink.
header geneve_option_int_destination_t {
    geneve_option_t base;
    // Destination headers must only be consumed by the INT Sink
};

// Intermediate nodes (INT Transit Hops) must process this type of INT Header
// and generate reports to the monitoring system as instructed.
header geneve_option_int_mx_t {
    geneve_option_t base;
    bit<4> version;
    bit<1> discard;
    bit<27> reserved1;
    bit<16> instruction_bitmap;
    bit<16> ds_id;
    bit<16> ds_instruction;
```

```
    bit<16> ds_flags;
};
```

All the above headers and structs must be added to the application's headers struct, along with the NV_FIXED_HEADERS.

```
struct app_headers {
    NV_FIXED_HEADERS

    geneve_t custom_geneve;
    // geneve_option_t is not explicitly used by the P4 program, so a reference
    // placed in the headers the type is not optimized out. The NvOptionParser
    // instantiation references this type it by string name.
    geneve_option_t geneve_opt;
    geneve_option_int_md_t geneve_opt_int_md;
    geneve_option_int_destination_t geneve_opt_int_destination;
    geneve_option_int_mx_t geneve_opt_int_mx;
};
```

The last step for the parser is to define the parser state that will perform Geneve parsing. An extern object, NvOptionsParser, is used to further parse into the Geneve options. In this example, the TLV "type" is a combination of Option class and type:

- Class 0x103, type 1 (INT-MD)
- Class 0x103, type 2 (Destination-type)
- Class 0x103, type 3 (INT-MX)

```
parser geneve_parser(
    packet_in packet,
    out app_headers headers
) {
    NvOptionParser<bit<24>, _>(
```

```
            "opt_len",                          // options_length_field
            2,                                      // options_length_shift
            0,                                      // options_length_add
            "geneve_option_t",                  // option_layout_header_type
            "length",                           // option_length_field
            0,                                      // option_length_shift
            4,                                      // option_length_add
            "option_class_type",                // option_type_field
            (list<tuple<bit<24>, _>>){      // options
                {24w0x010301, "headers.geneve_opt_int_md"},
                {24w0x010302, "headers.geneve_opt_int_destination"},
                {24w0x010303, "headers.geneve_opt_int_mx"}
            }
    ) geneveOptions;

    NV_FIXED_PARSER(packet, headers)

    @nv_transition_from("nv_parse_udp", GENEVE_PORT)
    state parse_geneve {
        // Fixed geneve is only an example; "base" header must be flex.
        packet.extract(headers.custom_geneve);
        geneveOptions.parseOptions(packet, headers);
        transition select(headers.custom_geneve.protocol_type) {
            NV_TYPE_IPV4 : nv_parse_inner_ipv4;
            NV_TYPE_IPV6 : nv_parse_inner_ipv6;
            NV_TYPE_MAC : nv_parse_inner_ethernet;
            default : accept;
        }
    }
}
```

The control below shows how a match action table can be configured to now match on a specific INT domain, and perform per ID forwarding.

```
control int_over_geneve(
    inout app_headers headers,
    in nv_standard_metadata_t std_meta,
    inout nv_empty_metadata_t user_meta,
    inout nv_empty_metadata_t pkt_out_meta
) {
    action forward(bit<32> port) {
        nv_send_to_port(port);
    }

    action drop() {
        nv_drop();
    }

    table geneve_option_int_md_table {
        key = {
            headers.geneve_opt_int_md.ds_id : exact;
        }
        actions = {
            forward;
            NoAction;
        }
        default_action = NoAction();
        const entries = {
            (16w0x100) : forward(1);
        }
    }

    table geneve_option_int_destination_table {
        key = {
            headers.custom_geneve.vni : exact;
        }
        actions = {
            forward;
            NoAction;
```

```
        }
        default_action = NoAction();
        const entries = {
            (24w0x200) : forward(2);
        }
    }

    table geneve_option_int_mx_table {
        key = {
            headers.geneve_opt_int_mx.ds_id : exact;
        }
        actions = {
            forward;
            NoAction;
        }
        default_action = NoAction();
        const entries = {
            (16w0x300) : forward(3);
        }
    }

    apply {
        if (headers.custom_geneve.isValid()) {
            geneve_option_int_md_table.apply();
            geneve_option_int_destination_table.apply();
            geneve_option_int_mx_table.apply();
        }
        drop();
    }
}

NvDocaPipeline(
    geneve_parser(),
    int_over_geneve()
) main;
```

See the full DPL example geneve_tlv.p4