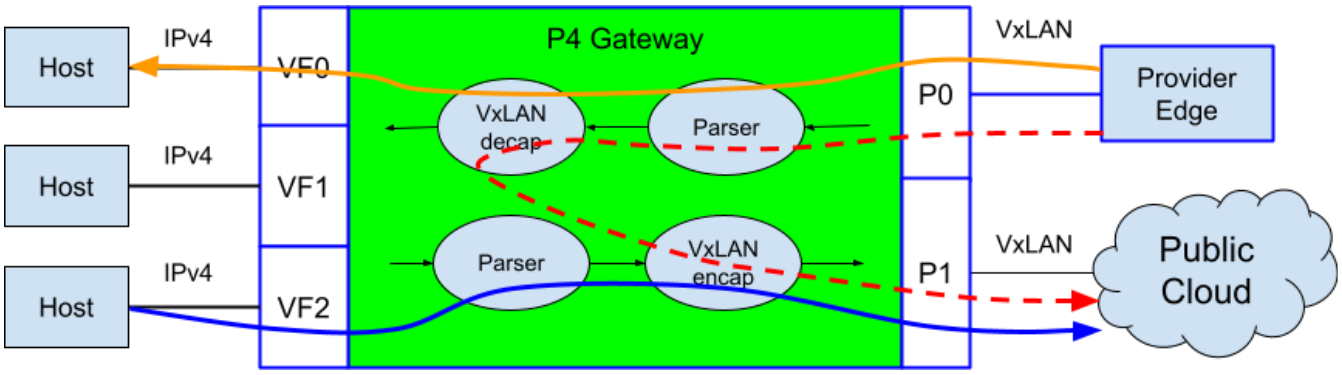
VXLAN Tunnel Gateway Example

Table of contents

This example illustrates how to write a basic VXLAN tunnel gateway using the DOCA target architecture. A tunnel gateway allows programmatic control over how VXLAN traffic can be "stitched" packets across tenant domains. In this example, end point traffic destined to local bare metal hosts can be decapsulated and forwarded to a VF, while gateway traffic can be decapsulated, re-encapsulated and sent back to the wire. For example, this program can be easily extended to be a gateway connecting legacy NVGRE networks to a VXLAN-GPE network.



Sample Code

This example users the native parser and 2 tables, of size 32K each. The wire port is configured with P4 port ID 0. A single bit in the user metadata structure is used to keep the decapsulation state.

```
#include <doca_model.p4>
#include <doca_headers.p4>
#include <doca_externs.p4>
#include <doca_parser.p4>

/*
* Table sizes.
*/
const bit<32> DECAP_TABLE_SIZE = 32768;
const bit<32> ENCAP_TABLE_SIZE = 32768;

/* The directionality is based on network to host
 * The user will configure the P4 port IDs in the OVS configuration
 */
const bit<32> WIRE_PORT = 32w0;
```

```
struct metadata_t {
    bit<1> was_decapped;
}

struct headers_t {
    NV_FIXED_HEADERS
}

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

The encapsulation control has a single table, matching on IPv4 destination address. If an entry matches, the packet is VXLAN encapsulated and forwarded to the specified port. If the packet does not hit any entry, then the packet is dropped. It is simple to add more complex policy rules such as a 5 tuple ACL.

```
/*
* This control performs the overlay policy including L2 encap with VXLAN
*/
control overlay_encap(
    inout headers_t headers,
    in nv_standard_metadata_t std_meta,
    inout metadata_t user_meta,
    inout nv_empty_metadata_t pkt_out_meta
) {
    NvDirectCounter(NvCounterType.PACKETS_AND_BYTES)
encap_counter;

    action deny() {
        nv_drop();
    }
```

```
    action vxlan_v4_encap(nv_mac_addr_t underlay_src_mac,
nv_mac_addr_t underlay_dst_mac,
                  nv_ipv4_addr_t underlay_sip, nv_ipv4_addr_t
underlay_dip, bit<24> vni, nv_logical_port_t port) {
        encap_counter.count();
        nv_set_vxlan_v4_underlay(headers, false, underlay_dst_mac,
underlay_src_mac, 0, underlay_sip, underlay_dip, vni);
        nv_send_to_port(port);
    }
    table encap_v4_table {
        key = {
            headers.ipv4.dst_addr : exact;
        }
        actions = {
            vxlan_v4_encap;
            deny;
        }
        size = ENCAP_TABLE_SIZE;
        default_action = deny;
        direct_counter = encap_counter;
    }

    apply {
       if (headers.ipv4.isValid() && (user_meta.was_decapped ==
1)) {
            encap_v4_table.apply();
       }
    }
}
```

The decapsulation control simply checks if the packet is VXLAN, and decapsulates it. From there the packet can be sent directly to a port, or hair-pinned back to the wire.

```
/*
* This control is for packets from wire to host (RX)
```

```
* and includes policy for L2 decap
*/
control decap_flow(
    inout headers_t headers,
    in nv_standard_metadata_t std_meta,
    inout metadata_t user_meta,
    inout nv_empty_metadata_t pkt_out_meta
) {
    NvDirectCounter(NvCounterType.PACKETS_AND_BYTES)
decap_counter;

    action deny() {
        nv_drop();
    }

    action decap() {
        decap_counter.count();
        nv_l2_decap(headers);
        user_meta.was_decapped = 1;
    }

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

    action decap_to_port(nv_logical_port_t port) {
        decap_counter.count();
        user_meta.was_decapped = 1;
        nv_l2_decap(headers);
        nv_send_to_port(port);
    }

    table decap_v4_table {
        key = {
            headers.vxlan.vni : exact;
        }
```

```
        actions = {
            decap;
            to_port;
            decap_to_port;
            deny;
            NoAction;
        }
        size = DECAP_TABLE_SIZE;
        direct_counter = decap_counter;
        default_action = deny;
    }

    apply {
        if (headers.vxlan.isValid()) {
            decap_v4_table.apply();
        }
    }
}
```

The main control checks the ingress port to determine if the packet is Network to Host, or Host to Network. Depending on the direction, it applies the decap_flow control, or performs an overlay encapsulation.

```
control gateway(
    inout headers_t headers,
    in nv_standard_metadata_t std_meta,
    inout metadata_t user_meta,
    inout nv_empty_metadata_t pkt_out_meta
) {
    overlay_encap() over;
    decap_flow() decap;

    /* user should add entries that correspond to the wire ports
     * A hit means this is an RX packet, miss means a TX packet
     */
```

```
    table direction_table {
        key = {
            std_meta.ingress_port : exact;
        }
        actions = {
            NoAction;
        }
        default_action = NoAction;
        const entries = {
            (WIRE_PORT) : NoAction();
        }
    }

    apply {
        user_meta.was_decapped = 0;
        if (direction_table.apply().hit) {
            decap.apply(headers, std_meta, user_meta,
pkt_out_meta);
        }
        over.apply(headers, std_meta, user_meta, pkt_out_meta);
    }
}

NvDocaPipeline(
    packet_parser(),
    gateway()
) main;
```

See the full DPL example gateway.p4

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.
