

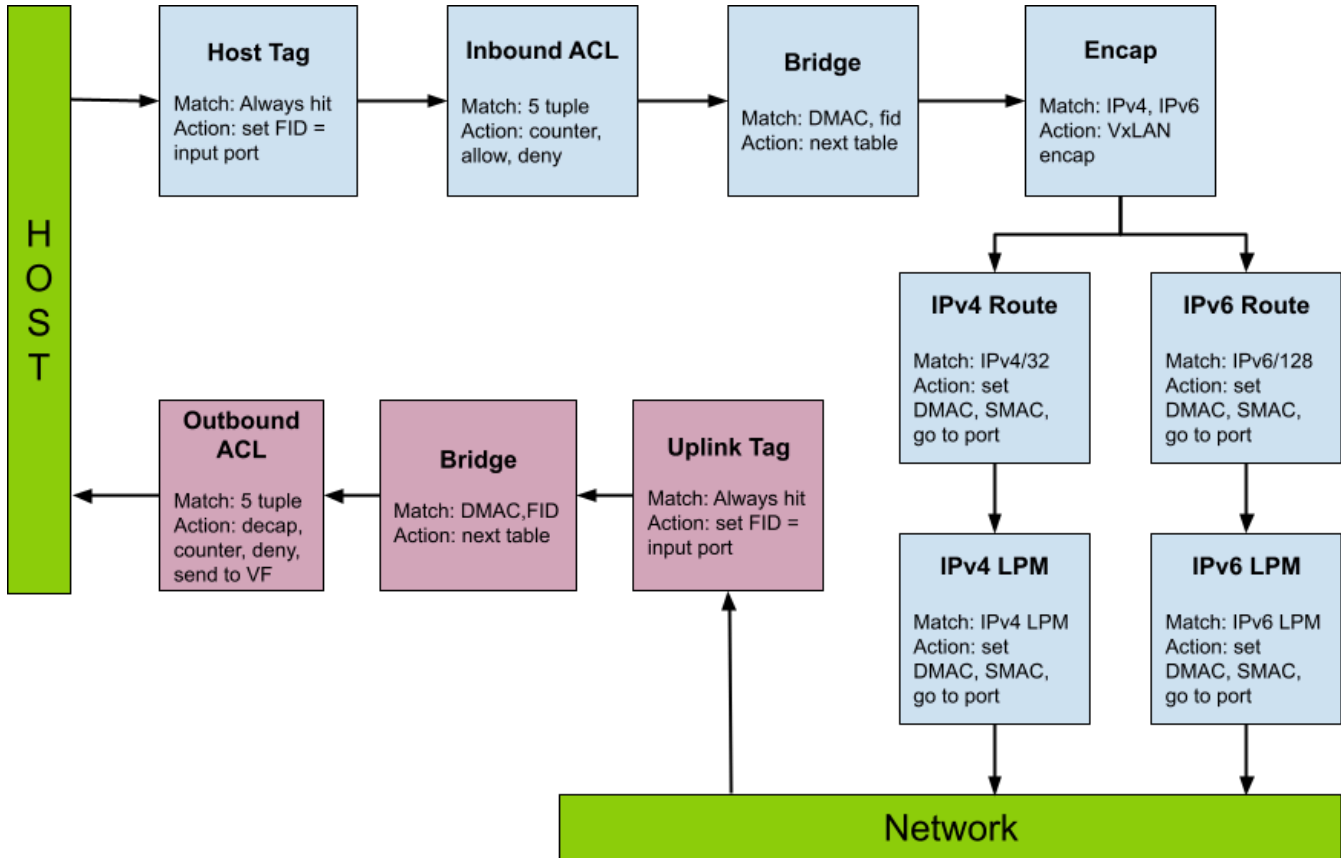


Host-based Networking Example

Table of contents

Sample Code

HBN is service that provides controller-less VPC networking solution for bare metal as a service (BMaaS), featuring tenant isolation with EVPN VXLAN and accelerated routing on the host. This example demonstrates how to build a DPL datapath that supports host-based networking (HBN).



Sample Code

This example uses indirect counters to monitor the number of packets that are allowed, denied and encapsulated/decapsulated. A user defined forwarding ID is used for determining how the bridge behaves.

```

enum bit<32> AdmissionCounter_t {
    DENY = 0,
    ALLOW = 1,
    VXLAN_V4_ENCAP = 2,
    VXLAN_V6_ENCAP = 3
}
  
```

```

enum bit<32> DecapFlowCounter_t {
    DENY = 0,
    ALLOW = 1,
}

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

/* fid is 32 bits
 */
typedef bit<32> fid_t;

struct metadata_t {
    fid_t fid;
}

```

The default parser is used for this example:

```

struct headers_t {
    NV_FIXED_HEADERS
}

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

```

Finally, the main control body invokes three separate sub-controls:

- `overlay_encap` – defines the logic and policies for applying VXLAN encapsulations along with ACLs
- `underlay_route` – applies a custom IPv4/IPv6 routing tables with exact and LPM matching. The next hop is set, TTL decremented, and the packet is routed.

```

/**
 * 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) fid_counter;
    NvCounter(4, NvCounterType.PACKETS_AND_BYTES)
admission_counter;

    action set_fid(fid_t fid) {
        user_meta.fid = fid;
    }
    action deny() {
        admission_counter.count(AdmissionCounter_t.DENY);
        nv_drop();
    }
    action allow() {
        admission_counter.count(AdmissionCounter_t.ALLOW);
    }
    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_set_vxlan_v4_underlay(headers, false, underlay_dst_mac,
underlay_src_mac, 0, underlay_sip, underlay_dip, vni);

admission_counter.count(AdmissionCounter_t.VXLAN_V4_ENCAP);
    }
    action vxlan_v6_encap(nv_mac_addr_t underlay_src_mac,
nv_mac_addr_t underlay_dst_mac,

```

```

        nv_ipv6_addr_t underlay_sip, nv_ipv6_addr_t
underlay_dip, bit<24> vni) {
    nv_set_vxlan_v6_underlay(headers, false, underlay_dst_mac,
underlay_src_mac, 0, underlay_sip, underlay_dip, vni);

admission_counter.count(AdmissionCounter_t.VXLAN_V6_ENCAP);
    }
    // A FID could be as simple as ingress port to FID mapping
    // e.g. ports 1 and 2 are in FID 1, ports 3 and 4 are in FID 2
    table fid_table {
        key = {
            std_meta.ingress_port : exact;
        }
        actions = {
            set_fid;
            NoAction;
        }
        size = FID_TABLE_SIZE;
        default_action = NoAction;
        direct_counter = fid_counter;
    }

    table admit_v4_table {
        key = {
            headers.ipv4.src_addr : exact;
            headers.ipv4.dst_addr : exact;
            headers.ipv4.protocol : exact;
            headers.tcp.src_port : exact;
            headers.tcp.dst_port : exact;
        }
        actions = {
            allow;
            deny;
            NoAction;
        }
        size = ADMIT_TABLE_SIZE;
    }

```

```

        default_action = NoAction;
    }

table admit_v6_table {
    key = {
        headers.ipv6.src_addr : exact;
        headers.ipv6.dst_addr : exact;
        headers.ipv6.next_header : exact;
        headers.tcp.src_port : exact;
        headers.tcp.dst_port : exact;
    }
    actions = {
        allow;
        deny;
        NoAction;
    }
    size = ADMIT_TABLE_SIZE;
    default_action = NoAction;
}

table bridge {
    key = {
        headers.ethernet.dst_addr : exact;
        user_meta.fid : exact;
    }
    actions = {
        NoAction;
    }
    size = BRIDGE_TABLE_SIZE;
    default_action = NoAction;
}

table encap_v4_table {
    key = {
        headers.ipv4.dst_addr : lpm;
    }
}

```

```

    actions = {
        vxlan_v4_encap;
    }
    size = ENCAP_TABLE_SIZE;
    default_action = vxlan_v4_encap(DEFAULT_DST_MAC,
DEFAULT_SRC_MAC,
                                DEFAULT_SRC_IPV4,
DEFAULT_DST_IPV4, DEFAULT_VNI);
}

table encap_v6_table {
    key = {
        headers.ipv6.dst_addr : lpm;
    }

    actions = {
        vxlan_v6_encap;
    }
    size = ENCAP_TABLE_SIZE;
    default_action = vxlan_v6_encap(DEFAULT_DST_MAC,
DEFAULT_SRC_MAC,
                                DEFAULT_SRC_IPV6,
DEFAULT_DST_IPV6, DEFAULT_VNI);

}
apply {
    user_meta.fid = 0;
    fid_table.apply();
    if (headers.ipv4.isValid()) {
        admit_v4_table.apply();
        if (bridge.apply().hit) {
            encap_v4_table.apply();
        }
    }
    else if (headers.ipv6.isValid()) {
        admit_v6_table.apply();
    }
}

```



```

        if (bridge.apply().hit) {
            encap_v6_table.apply();
        }
    }
}

/**
 * This control performs the underlay policy with routing
 */
control underlay_route(
    inout headers_t headers,
    in nv_standard_metadata_t std_meta,
    inout metadata_t user_meta,
    inout nv_empty_metadata_t pkt_out_meta
) {
    action set_port_and_route(nv_logical_port_t port,
nv_mac_addr_t src_mac, nv_mac_addr_t dst_mac) {
        headers.ethernet.src_addr = src_mac;
        headers.ethernet.dst_addr = dst_mac;
        nv_dec_ip_ttl(headers, 1);
        nv_send_to_port(port);
    }

    table exact_v4_route {
        key = {
            headers.ipv4.dst_addr : exact;
        }
        actions = {
            set_port_and_route;
            NoAction;
        }
        size = ROUTE_TABLE_SIZE;
        default_action = NoAction;
    }
    table lpm_v4_route {

```

```

    key = {
        headers.ipv4.dst_addr : lpm;
    }
    actions = {
        set_port_and_route;
        NoAction;
    }
    size = ROUTE_TABLE_SIZE;
    default_action = NoAction;
}
table exact_v6_route {
    key = {
        headers.ipv6.dst_addr : exact;
    }
    actions = {
        set_port_and_route;
        NoAction;
    }
    size = ROUTE_TABLE_SIZE;
    default_action = NoAction;
}
table lpm_v6_route {
    key = {
        headers.ipv6.dst_addr : lpm;
    }
    actions = {
        set_port_and_route;
        NoAction;
    }
    size = ROUTE_TABLE_SIZE;
    default_action = NoAction;
}

apply {
    if (headers.ipv4.isValid()) {
        if (exact_v4_route.apply().miss) {

```

```

        lpm_v4_route.apply();
    }
}
else if (headers.ipv6.isValid()) {
    if (exact_v6_route.apply().miss) {
        lpm_v6_route.apply();
    }
}
}
}
}

```

- `decap_flow` – defines the policies for forwarding ID, L2 bridging, decapsulation of VXLAN and VXLAN-GPE tunnels, and forwarding to the VF

```

/**
 * 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) fid_counter;
    NvCounter(2, NvCounterType.PACKETS_AND_BYTES) decap_counter;

    action deny() {
        decap_counter.count(DecapFlowCounter_t.DENY);
        nv_drop();
    }
    action set_fid(fid_t fid) {
        user_meta.fid = fid;
    }
    action decap_l2_and_send(nv_logical_port_t port) {

```

```

    decap_counter.count(DecapFlowCounter_t.ALLOW);
    nv_l2_decap(headers);
    nv_send_to_port(port);
}
action decap_ipv4_and_send(nv_logical_port_t port) {
    decap_counter.count(DecapFlowCounter_t.ALLOW);
    nv_l3_decap(headers, false, 0xffffffff,
0x112233445566, NV_TYPE_IPV4, 0);
    nv_send_to_port(port);
}
action decap_ipv6_and_send(nv_logical_port_t port) {
    decap_counter.count(DecapFlowCounter_t.ALLOW);
    nv_l3_decap(headers, false, 0xffffffff,
0x112233445566, NV_TYPE_IPV6, 0);
    nv_send_to_port(port);
}

table fid_table {
    key = {
        std_meta.ingress_port : exact;
    }
    actions = {
        set_fid;
        NoAction;
    }
    size = FID_TABLE_SIZE;
    default_action = NoAction;
    direct_counter = fid_counter;
}

table bridge {
    key = {
        headers.ethernet.dst_addr : exact;
        user_meta.fid : exact;
    }
    actions = {

```

```

        deny;
        NoAction;
    }
    size = BRIDGE_TABLE_SIZE;
    default_action = deny;
}

table decap_v4_table {
    key = {
        headers.ipv4.src_addr : exact;
        headers.ipv4.dst_addr : exact;
        headers.ipv4.protocol : exact;
        headers.udp.src_port : exact;
        headers.udp.dst_port : exact;
    }
    actions = {
        decap_l2_and_send;
        decap_ipv4_and_send;
        deny;
    }
    size = DECAP_TABLE_SIZE;
    default_action = deny;
}

table decap_v6_table {
    key = {
        headers.ipv6.src_addr : exact;
        headers.ipv6.dst_addr : exact;
        headers.ipv6.next_header : exact;
        headers.udp.src_port : exact;
        headers.udp.dst_port : exact;
    }
    actions = {
        decap_l2_and_send;
        decap_ipv6_and_send;
        deny;
    }
}

```

```

    }
    size = DECAP_TABLE_SIZE;
    default_action = deny;
}

apply {
    user_meta.fid = (fid_t) 0;
    fid_table.apply();
    if (bridge.apply().hit) {
        if (headers.ipv4.isValid()) {
            decap_v4_table.apply();
        }
        else if (headers.ipv6.isValid()) {
            decap_v6_table.apply();
        }
    }
}
}
}

```

The main control checks which direction the packet came from.

- if the packet is from network to host, it invokes the decap_flow control
- if the packet is from host to network, it invokes the overlay_encap and routing controls

```

control host_based_networking(
    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() overlay;
    underlay_route() route;
    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 {
        if (direction_table.apply().miss) {
            overlay.apply(headers, std_meta, user_meta,
pkt_out_meta);
            route.apply(headers, std_meta, user_meta,
pkt_out_meta);
        }
        else {
            decap.apply(headers, std_meta, user_meta,
pkt_out_meta);
        }
    }
}

NvDocaPipeline(
    packet_parser(),
    host_based_networking()
) main;

```

See the full DPL example [hbn.p4](#)

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, NVIDIA. PDF Generated on 04/24/2025