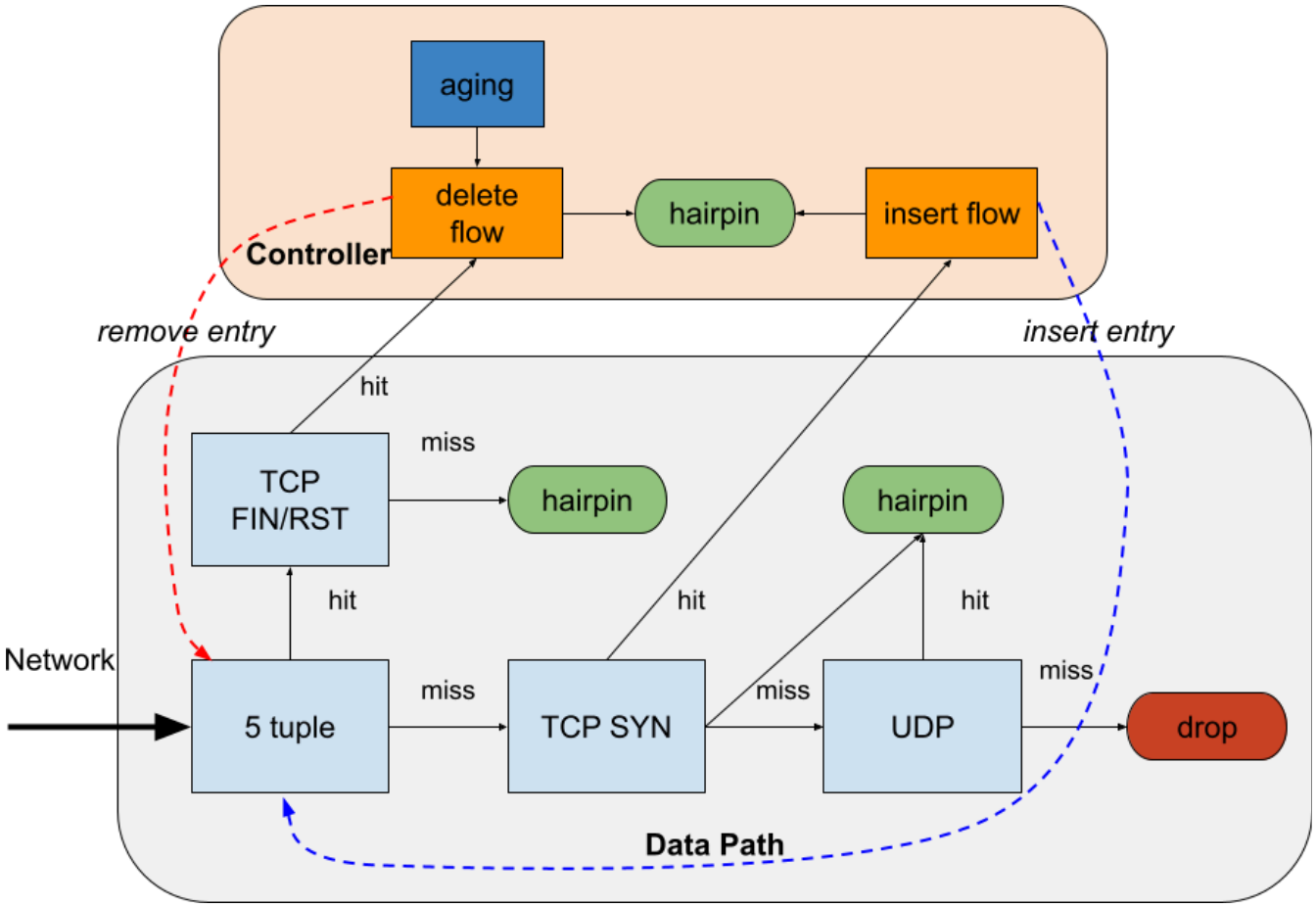
Connection Tracking Example

Table of contents

Sample Code

Connection tracking is a stateful process that follows the a TCP/UDP session from establishment to termination. It consists of a 5 tuple match on <Source IP address, Destination IP address, IP protocol, L4 Source port, L4 Destination port>. The DPL datapath relies on a controller to insert entries for new 5 tuple flows that have not been seen yet.



Sample Code

Connection metadata is a struct that sent with a packet that is sent to the P4Runtime controller. It requires a special annotation with the label "packet_in". The user can define up to 32 bits of data in this data structure.

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

```
@nv_controller_metadata("packet_in")
struct connection_meta_t {
    bit<2> type;
    bit<2> _reserved;
    bit<28> zone;
}
```

In this sample, the packet processing pipeline examines the state of the connection based on the TCP flags and set in the connection metadata an enum type describing the state of the flow.

```
/* Meta connection type */
enum bit<2> ct_type {
    PASS = 0,
    NEW = 1,
    RST = 2,
    FIN = 3
}
```

For readability, constants are used to assign P4 port IDs to more a more meaningful symbol name. The mapping of PFs and VFs to P4 port IDs is done separately in the Service Configuration.

```
/*
  VF port to SW entity that manages the flow insertion,
  deletion and expiry times
*/
const bit<32> WIRE_PORT = 32w0;
const bit<32> HAIRPIN_PORT = 32w1;
```

```
control conn_track(
    inout nv_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
) {
    connection_meta_t ctrl_meta;

    /* 5-tuple matching for L4 TCP/UDP flows */
    NvDirectCounter(NvCounterType.PACKETS_AND_BYTES)
table_t5_counter;

    action set_connection_id_ct_table_t5(bit<28> zone) {
        table_t5_counter.count();
        ctrl_meta.zone = zone;
        ctrl_meta.type = ct_type.PASS;
    }

    action no_action_ct_table_t5() {
        table_t5_counter.count();
    }

    table ct_table_t5 {
        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 = {
            set_connection_id_ct_table_t5;
            no_action_ct_table_t5;
        }
        direct_counter = table_t5_counter;
        // on hit table ct_table_known
        // on miss table ct_table_tcp_miss
        default_action = no_action_ct_table_t5;
```

```
        size = 1048576;
    }

    /*
 * Known connections handling.
 * Precondition - must be a TCP packet
 *  RST:    TYPE_RST
 *  FIN:    TYPE_FIN
 *  FINRST: TYPE_RST
 *
 * Match: tcp.flags
 * Actions: meta.type, next PIPE
 * MISS: next PIPE
 */
    NvDirectCounter(NvCounterType.PACKETS_AND_BYTES)
table_known_counter;

    action set_connection_type_ct_table_known(bit<2> type) {
        table_known_counter.count();
        ctrl_meta.type = type;
    }

    table ct_table_known {
        key = {
            headers.tcp.flags : exact;
        }
        actions = {
            set_connection_type_ct_table_known;
        }
        direct_counter = table_known_counter;
        default_action =
set_connection_type_ct_table_known(ct_type.PASS);
        const entries = {
            ( 0x1) :
set_connection_type_ct_table_known(ct_type.FIN);
            ( 0x4) :
set_connection_type_ct_table_known(ct_type.RST);
```

```
            (/*NV_TCP_PROTOCOL,*/ 0x5) :
set_connection_type_ct_table_known(ct_type.RST); /* RST+FIN */
        }
    }

    /*
  * Unknown TCP connections handling
  * Precondition - must be a TCP packet
  *  SYN: TYPE_NEW
  *
  * Match: tcp.flags
  * Actions: PIPE, meta.type=NEW
  * MISS: NEXT PIPE
  */
    NvDirectCounter(NvCounterType.PACKETS_AND_BYTES)
table_tcp_miss_counter;

    action set_connection_type_ct_table_tcp(bit<2> type) {
        table_tcp_miss_counter.count();
        ctrl_meta.type = type;
    }

    action no_action_ct_table_tcp() {
        table_tcp_miss_counter.count();
    }

    table ct_table_tcp {
        key = {
            headers.tcp.flags : exact;
        }
        actions = {
            set_connection_type_ct_table_tcp;
            no_action_ct_table_tcp;
        }
        direct_counter = table_tcp_miss_counter;
        default_action = no_action_ct_table_tcp;
        const entries = {
```

```
            (0x02) :
set_connection_type_ct_table_tcp(ct_type.NEW);   /* SYN=1 ACK=0 */
        }
    }

    /*
  * Unknown UDP connections handling
  *
  * Match: UDP
  * Actions: PIPE, meta.type=NEW
  * MISS: DROP
  */
    NvDirectCounter(NvCounterType.PACKETS_AND_BYTES)
table_udp_miss_counter;

    action set_connection_type_ct_table_udp(bit<2> type) {
        table_udp_miss_counter.count();
        ctrl_meta.type = type;
    }

    action drop_ct_table_udp() {
        table_udp_miss_counter.count();
        nv_drop();
    }

    table ct_table_udp {
        key = {
            std_meta.l4_type : exact;
        }
        actions = {
            set_connection_type_ct_table_udp;
            drop_ct_table_udp();
        }
        direct_counter = table_udp_miss_counter;
        default_action = drop_ct_table_udp();
        const entries = {
```

```
            (L4_TYPE_UDP) :
set_connection_type_ct_table_udp(ct_type.NEW);   /* SYN=1 ACK=0 */
        }
    }

    /* 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 {
        ctrl_meta.type = ct_type.PASS;
        ctrl_meta._reserved = 0;
        ctrl_meta.zone = 0;

        if (direction_table.apply().hit) {
            if (std_meta.is_l4_ok == 1w1) {
                if (ct_table_t5.apply().hit) {
                    if (ct_table_known.apply().hit) {
                        nv_send_to_controller(ctrl_meta);
                    }
                } else if (headers.tcp.isValid()) {
                    if (ct_table_tcp.apply().hit) {
                        nv_send_to_controller(ctrl_meta);
                    }
```

```
                } else {
                    if (ct_table_udp.apply().hit) {
                        nv_send_to_controller(ctrl_meta);
                    }
                }
            }
        }
        nv_send_to_port(HAIRPIN_PORT);
    }
}

// Instantiate the top-level DOCA Rx package
NvDocaPipeline(
    nv_fixed_parser(),
    conn_track()
) main;
```

See the full DPL example conn_track.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.
