Data Path Features and Enhancements

Application Note

Table of Contents

Chapter 1: Overview

This document contains details and application notes about miscellaneous data path features/enhancements.

NOTE: This document will contain features applicable to a combination of platforms and will be updated in following releases as the feature is added to other platforms.

Chapter 2: [Flow-cache] TCP Pure-ACK Flows

Supported Platforms: *ALL*

Earlier TCP-ACK flow prioritization feature was based on detecting *N* number of consecutive TCP-ACK packets and then assigning the high priority queue (priority#1) to the flow. Once the flow has been identified and marked as TCP-ACK, it will never change to data and vice-versa.

Based on the nature of TCP, a flow starts as ACK/data in one direction and subsequently can change the direction (example: drop-box, and so on).

The flow-cache has been modified to create a separate flow for TCP pure-ACK, if pure-ACKs are received in either direction. This would mean that a TCP session could have up to four flows based on the interaction and flow of pure-ACK and data packets. Flows that are idle for a period of *Flow Timer Interval* are candidates for eviction to create space for new flows if the flow table becomes congested (as per the already existing mechanism).

The flow-cache configuration option is provided to enable/disable this feature. If disabled, flow-cache would fall back to the old TCP-ACK flow prioritization mechanism.

Command to check the status of TCP Pure-ACK prioritization:

```
# fcctl status
        Flow Timer Interval = 10000 millisecs
        Pkt-HW Activate Deferral = 1
        Pkt-HW Idle Deactivate = 0
        Acceleration Mode: <L2 & L3>
        MCast Learning <Disabled>
        MCast Acceleration IPv4<Enabled> IPv6<Enabled>
        IPv6 Learning <Enabled>
        GRE Learning <Enabled> Mode<Tunnel>
        TCP Ack Prioritization <Enabled>
        HW Acceleration <Enabled>
        Flow Learning Enabled : Max<16384>, Active<0>, Cummulative [ 0 - 0 ]
```

Command to enable/disable TCP Pure-ACK Prioritization:

```
# fcctl config --tcp-ack-mflows 1
Broadcom Packet Flow Cache TCP ACK Multi-Flow <Enabled>
#
# fcctl config --tcp-ack-mflows 0
Broadcom Packet Flow Cache TCP ACK Multi-Flow <Disabled>
```

Flow-cache flows classified as ACK will display TCP_PURE_ACK flag set, as below:

```
# cat /proc/fcache/nflist
Broadcom Packet Flow Cache v4.0
FlowObject          idle:+swhit SW_TotHits:          TotalBytes HW_tpl HW_TotHits DlConntrack
    PdConntrack L1-Info Prot   SourceIpAddress:Port    DestinIpAddress:Port   Vlan0(mcast)
    Vlan1(mcast) tag# IqPrio SkbMark    TCP_PURE_ACK

0xd94744a0@002977    0:     4         72:                5574 0xffffffff          0    (null)
    d65c6a70 EPHY  8    6  <192.168.001.250:61480> <192.168.001.001:07681>  0x0000ffff  0x0000ffff
    0      0 0xd662e880    0

0xd9474540@002978    0:     3         55:                3364 0xffffffff          0    (null)
    d65c6a70 EPHY  8    6  <192.168.001.250:61480> <192.168.001.001:07681>  0x0000ffff  0x0000ffff
    0      0 0xd662e880    1
```

Similarly, hardware accelerator flows would show the *tcp_pure_ack* flag set, as below:

```
            flow[990] :
{key={src_mac=e2:11:68:58:00:00,dst_mac=78:9e:ae:aa:00:00,vtag0=0x0,vtag1=0x0,vtag_num=0,eth_type
=0x0,tos=0xc0,dir=ds,ingress_if=wan0,lookup_port=1,wan_flow=0,tcp_pure_ack=1},result={egress_if=l
an2,queue_id=2,service_queue_id=0,wan_flow=0,wan_flow_mode=0,is_routed=0,is_l2_accel=1,is_hit_tra
p=0,is_wred_high_prio=0,drop=0,mtu=1500,is_tos_mangle=0,tos=0xc0,wl_metadata=0,pathstat_idx=2,cmd
_list_length=32,cmd_list=08450100ffff00109400000100109400000438000004080 0[5]ff[76]}}
```

iptables/ebtables Extension

Blog tcp-pure-ack filter is added to iptables/ebtables to benefit the TCP-pure-ACK classification by flow-cache. The user must add corresponding rules based on blog filters as shown below.

```
iptables -A FORWARD -m blog --tcp-pureack …
ip6tables -A FORWARD -m blog --tcp-pureack …
ebtables -A FORWARD --tcp-pureack …
```

In order to stay backward compatible, the user must add the appropriate rules for TCP-ACK packets to be forwarded to the correct egress priority queue.

Backward Compatibility

When the TCP-pure-ACK multi-flow feature is disabled in flow-cache (`fcctl config --tcp-ack-mflows 0`), the system falls back to an earlier implementation (based on *N* number of consecutive TCP-ACK packets) but without automatically selecting the egress queue.

The user must configure the required iptables/ebtables rules to prioritize the ACK packets, in other words, select the appropriate egress queue.

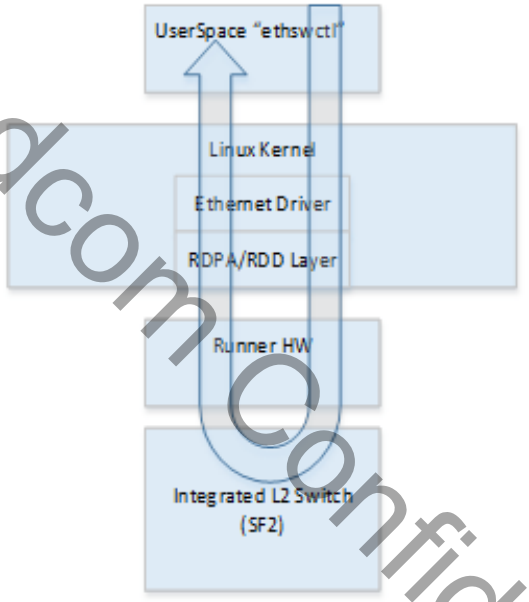
Chapter 3: [Runner/SF2] Runner Keep-Alive Mechanism

Supported Platforms: *BCM63138/BCM63148/BCM4908/BCM62118*

In order to verify that the Runner data path is working (alive), a Runner keep-alive mechanism is added. The userspace interface is provided as library API/CLI for customers to build their own application and take action as appropriate upon detecting failure.

A dummy Linux device, *bcmswlpbk0*, is created by Ethernet driver. This device is used to send and receive the loopback packet that verifies the runner/switch data path, as shown below.

Figure 1: bcmswlpbk0 Sends and Receives the Loopback Packet that Verifies the Runner/Switch Data Path



Using the *ethswctl* CLI command:

```
ethswctl {-c|--command} keepalive -x <timeout(sec)> -y <retries> -z <iterations> -v <sleep(sec)>
    -x <timeout(sec)>  :  Timeout value in seconds to wait for loopback packet reception.
    -y <retries>       :  Number of retries before declaring runner data path failure.
    -z <iterations>    :  Number of times to run the loopback test.
                          Iterations = 0 means run forever (every <sleep> seconds).
    -v <sleep(sec)>    :  Seconds to sleep between iterations. Minimum 60 seconds.
```

Using the library API from a customer application:

```
int bcm_KeepAlive(unsigned int timeout)
```

Parameters:

```
    - unsigned int timeout = Seconds to wait until declaring failure (similar to -x in CLI)
```

Returns:

```
    - BCM_E_NONE     = Success.
    - ETIME          = Timed out without receiving looped back packet
    - Other errors as defined in errno.h.
```

Chapter 4: [Kernel] Connection Tracking Regardless-Drop Improvements

Supported Platforms: *ALL*

Existing *regardless_drop* maintains the Least Recently Used (LRU) connection lists and when the conntrack (connection tracking) table is full, drops the connection at the head of the list to make room for a new connection. There are two such lists to hold High and Low priority connections based on the Ingress QoS classification.

The following limitations are noticed with the above implementation:

1. Non-accelerated connections get refreshed with packet hits but accelerated connections are refreshed only when the CT timer expires. This puts accelerated connections at a disadvantage in the LRU list and can be evicted even when there is a lot of traffic.
2. The current design does not allow low priority connections to evict high priority. Traffic flows that do not go through Ingress QoS classification (including local traffic) are treated as low priority, thus they run the risk of not finding a place in conntrack if all the entries are occupied by high priority connections.

To address the above issues, the following enhancements are done:

3. Added a mechanism to notify software and hardware accelerated flow activation/deactivation events to conntrack.
4. Two LRU lists are extended to six as described below:
   a. Low (not accelerated)
   b. Low_sw_accel (Flow-cache accelerated)
   c. Low_hw_accel (Hardware accelerated)
   d. High (not accelerated)
   e. High_sw_accel (Flow-cache accelerated)
   f. High_hw_accel (Hardware accelerated)
5. Three compile-time selectable connection drop policies are defined:
   1. LoHiLoHi Policy (default) — **Policy#1**
      - Drop candidacy order is Low→High→Low_sw_accel→High_sw_accel→Low_hw_accel→High_hw_accel
      - Avoids dropping accelerated flows to reduce the cost (of flow eviction from software and then hardware)
      - In general, the number of conntrack entries is greater than the accelerated flows.
   2. HW-accelerated-last Policy — **Policy#2**
      - Drop candidacy order is Low→Low_sw_accel→ High→High_sw_accel→Low_hw_accl→High_hw_accel
      - This policy prefers to keep flows in HW accelerator.
   3. Strict Priority Policy — **Policy#3**
      - Drop candidacy order is Low→Low_sw_accel→Low_hw_accl→High→High_sw_accel→High_hw_accel
      - This policy prefers high priority flows/connections over low priority.
6. Irrespective of the connection priority and/or acceleration mode, a new connection will always drop “an” existing connection based on the policy, when the conntrack table is full.

NOTE: Change the drop policy using “menuconfig→”… Networking Features”→”..Netfilter…” → “Netfilter Regardless Connection drop order”

Chapter 5: [Flow-cache] Half-Closed TCP Connection Acceleration

Supported Platforms: *ALL*

As per TCP RFC-1122, it is possible that the data continues in the other direction even when one side of the connection is closed (termed *Half-Closed TCP connection*). The existing flow-cache behavior is to evict the flows in both directions upon receiving FIN in either direction. This implementation does not continue the acceleration (flow-cache and/or hardware) of the traffic and could potentially impact the throughput.

Flow-cache is enhanced to support acceleration of *Half-Closed* TCP connections. When FIN is received, instead of closing the flows in both directions, the flow cache will only close the flow for the direction in which FIN is received. This will ensure the data coming in the other direction is accelerated util FIN or RST is received.

Chapter 6: [Flow-cache] Interface Stats Collection Enhancement

Supported Platforms: *ALL*

The existing Linux interface statistics query mechanism includes traversing all the accelerated flows going through the device and retrieving the statistics from accelerators (flow-cache and hardware). With support for thousands of flows, this process takes time, blocks the flow-cache from packet processing, and holds the CPU, thus preventing other critical tasks to run, e.g., Voice, for the duration of querying all the flows.

The interface statistics management module of the flow-cache has been redesigned to maintain *interface-flow-path* entries for quick retrieval of statistics for a given interface. The interface-flow-path logic maintains statistics per different, i.e., unique, interface paths that the flows are going through and accumulates them to retrieve the interface statistics efficiently.

Interface-flow-path entries are a limited resource in software and hardware. the flow-cache and hardware accelerator could support a different number of entries depending upon available hardware/platform resources, but the flow-cache must always support greater/equal entries to hardware.

Due to the hardware/platforms dependency, the hardware capability is not supported on all platforms (though flow-cache enhancement is supported on ALL).

The following command should be used to find the supported configuration per platform:

```
# cat /proc/fcache/stats/path

Max # of Path = 128
Max # of HWACC cntr = 64
Active # of Path = 3
Active # of vt device = 4


<Path_44> key=0x1b1c287eb, active_flows=2, sw_pathstat_idx=44, hw_pathstat_idx=1
         sw_pkt_count=0, sw_pkt_bytes=0, hw_pkt_count=80554, hw_pkt_bytes=5961408
         eth3.0_rx [-18] -> eth0.1_tx [-4] -> End

<Path_88> key=0x1b1c287f3, active_flows=2, sw_pathstat_idx=88, hw_pathstat_idx=2
         sw_pkt_count=0, sw_pkt_bytes=0, hw_pkt_count=426966, hw_pkt_bytes=647880826
         eth0.1_rx [-14] -> eth3.0_tx [0] -> End

<Path_108> key=0x294f2dfdd, active_flows=2, sw_pathstat_idx=108, hw_pathstat_idx=0
         sw_pkt_count=56932, sw_pkt_bytes=3786005, hw_pkt_count=0, hw_pkt_bytes=0
         eth1.0_rx [-18] -> br0_rx [-18] -> br0_tx [0] -> End

HWACC cntr table
<1>, valid=1, hit=80554, byte=5961408
<2>, valid=1, hit=426966, byte=647880826
```

Chapter 7: [XTM] Dynamic Upstream Queue Thresholds

Supported Platforms: *ALL xDSL*

DSL interfaces have a wide range of modes and link speeds. In ADSL mode, upstream rates could be just a few Kbps, whereas G.fast reaches in Gbps. These varied modes/speeds require appropriate upstream queue sizes based on the mode.

The XTM driver is updated to provide different upstream queue sizes based on the line mode. Default queue sizes are defined in the following file.

```
bcmdrivers/opensource/include/bcm963xx/bcmxtmcfg.h
#define XTM_ADSL_QUEUE_DROP_THRESHOLD          64    /* ADSL ATM,PTM, PTM_BONDED   */
#define XTM_VDSL_QUEUE_DROP_THRESHOLD         128    /* VDSL PTM, PTM_BONDED       */
#define XTM_VDSL_RTX_QUEUE_DROP_THRESHOLD     512    /* VDSL PTM, PTM_BONDED, ReTx */
#define XTM_GFAST_QUEUE_DROP_THRESHOLD       1024    /* GFAST PTM, PTM_BONDED      */
```

NOTE: These queue thresholds are applied to all the queues for a given mode.

The following CLI commands are provided to change the default queue sizes at runtime (effective on subsequent line retrain).

```
xtmctl threshold -adsl <threshold>
xtmctl threshold -vdsl <threshold>
xtmctl threshold -vdslRtx <threshold>
xtmctl threshold -gfast <threshold>
```

Chapter 8: [GRE] Hardware Acceleration Support for GRE

Supported Platforms: ***All Runner based platforms***

Hardware/Runner acceleration support for GREv0 (not for PPTP/GREv1) traffic is added for the above mentioned platforms. GRE tunnel pass-through and tunnel termination are supported as described below.

Runner Accelerated GRE Traffic Types:

1. L2GRE + 4in4
2. L3GRE + 4in4/4in6 (excluding BCM6838X/6848X)
3. L3GRE + 6in4 (excluding BCM63138/63148/62118/4908)
4. GRE Tunnel-in-Tunnel-Out + 4in4
5. GRE Tunnel-in-Tunnel-Out + 6in4/4in6/6in6 (excluding BCM63138/63148/62118/4908/63158)
6. GRE header with optional Key field (excluding BCM63138/63148/62118/4908/63158/6838X/6848X)

Non-Runner Accelerated GRE Traffic Types:

1. L3GRE + 6in6
2. L2GRE + 6in4/4in6/6in6
3. GRE header with optional fields (Sequence number and Checksum)
4. L2GRE tunnel of any type from LAN/WLAN side.

Additional Limitations:

1. [BCM6838X/6848X] GRE tunnel and GRE pass-through functionality are mutually excluded
2. [BCM6838X/6848X/6858X/6836X/6846X/6856X] Single GRE tunnel is supported

Chapter 9: [OVS] Integration with Packet Accelerators

Supported Platforms: ***BCM63138/BCM63148/BCM62118/BCM4908/BCM47189***

Broadcom packet accelerators are integrated with OpenVSwitch (OVS) datapath to support both unicast and multicast/IPTV traffic acceleration.

- Latest release supports OVS version 2.8.2.
- Flow-cache packet classification has not changed due to OVS integration, in other words, it continues to support the same L2/L3/L4 header fields for classification as it does for Linux bridge.
- Flow-cache packet modification for L2/L3/L4 header fields has not changed due to OVS integration, in other words, it continues to support the same header field modifications as it does for Linux bridge.

Unicast Flow Acceleration

- For unicast flows, Broadcom packet accelerator supports L3/L4 5-tuple classification for routed traffic and L2-header (MAC/VLAN) + DSCP based classification for bridged traffic. OVS flows that are classified based on the subset of these header fields are candidates for acceleration.
- By default, LAN-2-LAN bridging/switching is enabled in the hardware (switch or Runner). Customer must disable hardware switching if LAN-2-LAN traffic must be controlled by OVS.
- Flow-cache interfaces with OVS datapath using Mega-flows only, in other words, it does not directly interface with other OVS modules (for example, OVS connection tracking, and so on).

Multicast Flow Acceleration

- Learning based multicast flow acceleration is recommended to make sure OVS datapath modifications are applied correctly.
- Multicast implementation relies on OVS snooping capabilities.
- OVS is integrated with Broadcom MCPD Userspace daemon to provide multicast proxy capabilities. Customers not using BRCM MCPD are required to adjust their proxy accordingly.
- Broadcom reference software CMS and WebGUI do not support multicast configuration when using OVS Bridge. This implies that mcpd.conf file does not reflect the configuration for OVS Bridge and customers are required to make necessary adjustments in their CMS. The following provided sample mcpd.conf could be used as reference. Explanation of configuration attributes is available in docs/data_customer_docs/IGMPUserGuide.pdf.

Sample mcpd.conf for Bridged Ethernet WAN Interface (eth0.1)

```
#
#Begin IGMP configuration
#
igmp-default-version 3
igmp-query-interval 125
igmp-query-response-interval 10
igmp-last-member-query-interval 30
igmp-robustness-value 2
igmp-max-groups 25
igmp-max-sources 10
igmp-max-members 25
igmp-fast-leave 0
igmp-admission-required 0
igmp-proxy-interfaces
igmp-snooping-interfaces brsdn
igmp-mcast-interfaces eth0.1
#
#End IGMP configuration
#
#
#Begin MLD configuration
#
mld-default-version 2
mld-query-interval 125
mld-query-response-interval 10
mld-last-member-query-interval 10
mld-robustness-value 2
mld-max-groups 10
mld-max-sources 10
mld-max-members 10
mld-fast-leave 0
mld-admission-required 0
mld-proxy-interfaces
mld-snooping-interfaces brsdn
mld-mcast-interfaces eth0.1
#
#End MLD configuration
#
#
#Begin mcast configuration
#
mcast-max-groups-port-list
mcpd-strict-wan 0
igmp-mcast-snoop-exceptions 239.255.255.250/255.255.255.255 224.0.255.135/255.255.255.255
mld-mcast-snoop-exceptions ff05::0001:0003/ffff:ffff:ffff:ffff:ffff:ffff:ffff:ffff
#
#End mcast configuration
```

Revision History

96XXX-AN202-R, July 31, 2019

- Updated Chapter 4, [Kernel] Connection Tracking Regardless-Drop Improvements
- Added Chapter 8, [GRE] Hardware Acceleration Support for GRE
- Added Chapter 9, [OVS] Integration with Packet Accelerators

96XXX-AN201-R, December 14, 2018

- Updated Chapter 2, [Flow-cache] TCP Pure-ACK Flows (Supported Platforms)
- Added iptables/ebtables Extension
- Added Backward Compatibility
- Added Chapter 7, [XTM] Dynamic Upstream Queue Thresholds

96XXX-AN200-RI, July 11, 2018

- Initial release