Event Capture Subsystem

1 Introduction

In order to record the development of buffer sizes in switches, a system was needed to record arrival and departure of packets into the queue. The event capture subsystem was designed to meet this need. The functional requirements of the system are stated in detail in the document entitled “Requirements”.

The subsystem works by monitoring events such as arrival/departure/drop of a packet on the queues, then recording a Timestamp as well as the length of the packet. These events are then output in a packet with a certain format called the “event packet”.

2 Using the system

All of the functionality (except for the tentative requirements) has been met. In addition, the tentative requirement that the timer precision could be specified was also implemented. However, it might be found that this functionality consumes too much logic and is not necessary. It was included for the initial phases of the design where exact timer resolution needed is still unknown. It is possible to remove it later on without too much change to the logic.

The system has two software components: `unetwbs_ctrl` and `rcv_evts`. These two control the event capture system and record the events at the receiver respectively.

2.1 `unetwbs_ctrl`

The unetwbs_ctrl program is found in NF2/sw/unet/unet-switch4-w-b-s/. It implements all the functions to control and view the status of the event capture system. Again, these functions are documented in the “Requirements”.

The program works by reading and changing the control register of the event capture system. In addition, several read-only registers were added to observe the behavior of the subsystem. Executing “unetwbs_ctrl -h” will show available options. “unetwbs_ctrl -p” will print the status. A typical example of using the software is shown below:

```
Set the event packet ethertype:
# unetwbs_ctrl -i nf2c4 -e 0x8888

Set the destination MAC address:
# unetwbs_ctrl -i nf2c4 -D ffffffffffff

Set the source MAC address:
# unetwbs_ctrl -i nf2c4 -S 112233445566

Set the ports to monitor to ports 1,2, and 4:
# unetwbs_ctrl -i nf2c4 -c 0xb

Set the ports to send the event packet on to port 3:
# unetwbs_ctrl -i nf2c4 -s 0x4

Set the timer resolution to 2^2*16 ns = 64 ns:
# unetwbs_ctrl -i nf2c4 -t 0x2
```

```
Print the settings:
# unetwbs_ctrl -i nf2c4 -p

Found net device: nf2c4
Ethertype           : 0x8888
Send now            : 0
Enable events       : 0
Reset Timers        : 0
Timer Resolution    : 2
Event Out Ports     : 0x4
Event Capture Ports : 0xb
Destination MAC     : ff:ff:ff:ff:ff:ff
Source Mac          : 11:22:33:44:55:66
Events dropped      : 0
Events recorded     : 0
current_in_fifo     : 0
current_out_fifo    : 0
evt_wr_rdy          : 1
almost_full_0       : 0
almost_full_1       : 0
closed_fifo_0       : 0
closed_fifo_1       : 0
oq_pkt_avail        : 0
time difference out : 0
pkts sent out       : 0
almost full fifo 0  : 0
almost full fifo 1  : 0

Enable event collection:
# unetwbs_ctrl -i nf2c4 -b 1
```

Notes:

- All these commands could be done in one line as such:

```
unetwbs_ctrl -i nf2c4 -e 0x8888 -D ffffffffffff -S 112233445566
-c 0xb -s 0x4 -t 0x2 -b 1 -p
```

- The user needs to set all the values in the packet headers to get valid data.

2.2 rcv_evts

The `rcv_evts` program is supposed to record incoming evts, parse them, and display the output to stdout. It can be used as a basis on which to do more interesting things with the event packet information (such as build the queue occupancy in time). To use the program, simply run:

```
# rcv_evts -i nf2c3 -v
```

where nf2c3 is the interface receiving the event packets. The -v switch produces a verbose output like this:

```
Packet length    : 367
Packet seq num   : 0
Queue 1 size     : 0
Queue 2 size     : 0
Queue 3 size     : 0
Queue 4 size     : 0
Timestamp Event  : 0xc000000000001
Timestamp Event  : 0xc000000080000
Store  Event     : Q: 1, Pkt len: 134, Rel. Time: 121406
Remove Event     : Q: 1, Pkt len: 134, Rel. Time: 153
Store  Event     : Q: 1, Pkt len: 23, Rel. Time: 15770
Remove Event     : Q: 1, Pkt len: 23, Rel. Time: 36
Store  Event     : Q: 0, Pkt len: 355, Rel. Time: 74080
Remove Event     : Q: 0, Pkt len: 355, Rel. Time: 387
Store  Event     : Q: 1, Pkt len: 200, Rel. Time: 15485
Remove Event     : Q: 1, Pkt len: 200, Rel. Time: 222
Store  Event     : Q: 1, Pkt len: 185, Rel. Time: 2109
Remove Event     : Q: 1, Pkt len: 185, Rel. Time: 207
```

A script `run_test` is included that will send some packets and then receives them. The script assumes that the interfaces nf2c0, nf2c1, and nf2c3 are connected to the switch on ports 0, 1, and 3 respectively.

NOTE: The rcv_evts is able to parse evt packets captured to a pcap-format file offline. To parse such a file, rename the file to “teth_file” and do `./rcv_evts -o -v`

In order to capture the event packets to a file, we can tethereal as such:

```
tethereal -i nf2c3 -w teth_file
```

However, the live capture should work just fine. A problem might arise due to the network data not being flushed, but a hack using send_pkts to send some extra packets from the port packets are being captured on should fix that. Look at the run_test script to see how that is done.

3 Design Details

The system is composed of two main modules: evt_rcrdr and evt_pkt_wrtr. The evt_rcrdr module records individual events as they happen and serializes them to be sent out to the evt_pkt_wrtr module. The evt_pkt_wrtr then reads each event and stores it, and then when the send_egress_pkt is ready, it sends out a complete event packet.

3.1 evt_rcrdr

On every clock cycle, there are 4 types of possible events: A packet stored, a packet removed, a packet dropped, and a Timestamp event. The first three event types we call “short events” because they only need 32 bits. Whereas the Timestamp is a “long event” since it uses 64 bits.

The short events carry a time difference field which is the time since the last event. If this time does not fit into the field, a Timestamp event is signaled and recorded and the time difference is then recorded since the Timestamp event. The evt_rcrdr maintains both the Timestamp counter and the time difference counter.

The evt_rcrdr rearranges the events as they come in and stores them in a single clock cycle into the event fifo. The event fifo is a shallow fifo (depth=8) with a variable input size. The input is composed of five 32-bit words of which we can store a variable number of words (the first x words are stored). This is the number of events at the current clock cycle (the Timestamp event takes 2 words).

Note that the evt_rcrdr assumes that the events are all independent. The maximum sustained event recording capability is 62.5 million events per second, whereas the peak event recording capability is 8 events in any 32 ns interval (after which events should not go over the average or events would be lost). It is possible to adapt the evt_rcrdr to record any signal that meets these requirements with an additional field of 9-bits (usually the length field) to record any additional data.

In addition, the evt_rcrdr make no assumption on the sizes of the fields. These could be modified by changing the sizes in unet_defines.v. However, note that the evt_pkt_wrtr *does* make the assumption that the word sizes are 32 bits since that would simplify writing to the send_egress_pkt. However, this could be easily adapted as well.

3.2 evt_pkt_wrtr

This module reads events from the evt_rcrdr module and delineates them into packets. The module also monitors the datapath for activity and only injects event packets when the datapath is idle.

3.3 Schematic

The system follows the reference user data path.