Latency Test Report

June 2016 (4bd3371-dirty)

Alessandro Rubini for GSI

Table of Contents

Introduction 1

1 The Test Environment 1

2 The ltest Mechanism 1

3 Log Files 2

4 Interpreting the Data 2

4.1 Idle Time 3
4.2 Moderate Traffic (up to 97% nominal) 3
4.3 High Traffic (98% nominal and more) 3
4.4 Misbehavior of the VME devices 5

Introduction

This is a report of the latency tests performed on the lab WR network at GSI on 2016-06-09, after fixing a problem with the *minic* DMA buffer.

1 The Test Environment

We performed the tests in a network of 4 WR switches and a number of nodes. The switches are connected in a chain, where switch "1" is the timing master of "2", which is the master of "3", which is the master of "4".

There were three types of nodes connected to the network. 2 SPEC cards in PCI Express slots, 6 Pexaria cards in PCI express slots, and 4 Vetar cards in VME slots. The log of VME is collected with USB ports.

A traffic generator is used to injected random multicast traffic into switch 1. The latency test frames are injected into switch 2. Receivers are connected to each switch in the following way:

```
WRS 1:   traffic, U1, P2, S1

WRS 2:   ltest,   P3, P5

WRS 3:            U2, U3, P1

WRS 4:            U0, P4, S2
```

In the ascii-art above, the SPEC cards are labeled "S" and the USB ports to access the Vetar cards are labeled "U". The six Pexarias are labeled P0 through P6, where board P0 is the ltest source (and is thus labeled "ltest" above).

2 The ltest Mechanism

The latency-test mechanism is implemented in the WR node. As I write this it is in the *proposed_master* branch, but a few fixes are still pending. The version used in the test is commit "`38b1ae4 dev/minic: cleanup for overflow management`.

Latency is measured by timestamping the egress time of frames sent by an *ltest sender* and the ingress time of the frames as received by all *ltest receiver*. The test frames are periodically sent as broadcast using a custom-chosen Ethertype. At each iteration the master sends three frames, one at priority 7, one at priority 6, and the last at priority 0. The last frame broadcasts the egress timestamp of the other two frames. The test was performed with no VLAN, and thus no priority. The latency of the two frames is thus roughly the same. The ltest mechanism relies on the WR nodes to be synchronized (in the tests we performed they didn't miss WR track-phase while recording ltest events).

Every node can be a sender or a receiver. The user must ensure there is one sender only. The Ethertype for *ltest* is chosen at build time, and the packet-filter rules are changed to route *ltest* frames to the CPU in the node.

Each receiver reports about lost frames (or out of order ones: they require the three frames to be properly ordered), and the two delays for every tuple of three frames that is correctly receiver. For example:

```
lat:     22009  55142.730  53990.730
lat: unexpected 22010.3 after 22010.1
lat:     22011  53558.735  54310.735
```

From the lines above we see that tuple 22009 and 22011 were received with a delay of 53-55 microseconds, while the second frame of tuple 22010 was lost.

The sender can be configured to periodically send tuples using the `ltest` command. During the test we sent them 10 times per second, with the command “`ltest 0 100`” (0 seconds, 100 millliseconds).

3 Log Files

Based on previous tests, we performed the test twice. During first round we instructed the traffic generator to only sen frames between 100 and 800 bytes. During the second round the generator sent frames of all possible sizes: 64 to 1500 bytes long; it also spanned a wider range of loads. We configured the generator to make bursts of 5 minutes traffic, with a gigabit traffic of 70%, 80%, 90%, and then 91% to 100% with 1% steps, bu the 100% burst is shorter for a configuration mishap. The first test was configured to stop after 90% load. At 10 tuples per second, we thus sent 3000 ltest tuples for each traffic burst.

The name I used for the log files is of the form “log-mmddHHMM-id”, where `mmddHHMM` are month, day, hour, minute. And `id` is the name of the node: p1..p5, s1, s2, u1..u4 as listed in Chapter 2 [The ltest Mechanism], page 1.

The names are not very handy after the test is over, but the automatically-enforced creation of those names is useful on the test size, to ease removal of non-test-log files. The final files are committed under `ltest-files/log-160609/round1` and `ltest-files/log-160609/round2`.

4 Interpreting the Data

I’m now looking at data in `round2`, because `round1` is shorter and less interesting (we’ll pick back some `round1` data in Section 4.4 [Misbehavior of the VME devices], page 5.

The data points are split as follows (there is a gap between bursts, usually 13s):

- 70%: 7580 to 10580
- 80%: 10710 to 13710
- 90%: 13840 to 16840
- 91%: 16970 to 19970
- 92%: 20100 to 23100
- 93%: 23235 to 26235
- 94%: 26365 to 29365
- 95%: 29495 to 32495
- 96%: 32625 to 35625
- 97%: 35760 to 38760
- 98%: 38890 to 41890
- 99%: 42020 to 45020
- 100%: 45150 to 46330

Unfortunately, we don’t know exactly the bandwidth injected by the traffic generator. Its own application reports a little more bandwidth than the configured one; most likely we configured the payload, excluding ethernet headers. The feeling from data is that we reached 100% already at nominal 98% traffic. We must also considered that there is other traffic on the network, though limited.

4.1 Idle Time

The first thing to note is how each switch takes 2.5 microseconds to route an ltest frame (which is a minimum-size frame). Nodes connected to the same switch as the ltest sender see a delay 2500 nanoseconds, those connected to a nearby switch see 5000 nanoseconds and those connected to switch 4 see 7500 nanoseconds. We consistently observe 10 microseconds when injecting ltest in switch 1 and receiving from switch 4.

4.2 Moderate Traffic (up to 97% nominal)

Note: I won't consider VME nodes here, as they are very misbehaving

There is a little frame loss even at moderate traffic. Lines of the form “unexpected 11990.3 after 11990.1” signal a frame loss (in the example, the second frame of a tuple).

20 different ltest frames were lost during the 80% burst. WRS2 dropped one ltest frame towards node p3, and one towards p5. Both were overflows of the output queue. We lost 6 frames from WRS2 to WRS3, either in the output or input queue. Some frames were lost in WRS3 and WRS4, nothing in WRS1.

Only 2 frames were lost during the 90% burst (WRS3 and WRS3-to-WRS4), 3 during the 91% burst, 1 during the 92% burst.

Nothing was lost in the 93%-96% bursts. 5 frames were lost in the 97% burst.

The measured latency is evenly spread between the baseline (2.5, 5, 7.5 usec) and 20 microseconds. No event is reported with more than 24 microseconds. Latencies more than 22 microseconds are only reported by nodes in WRS4 during the 97% burst.

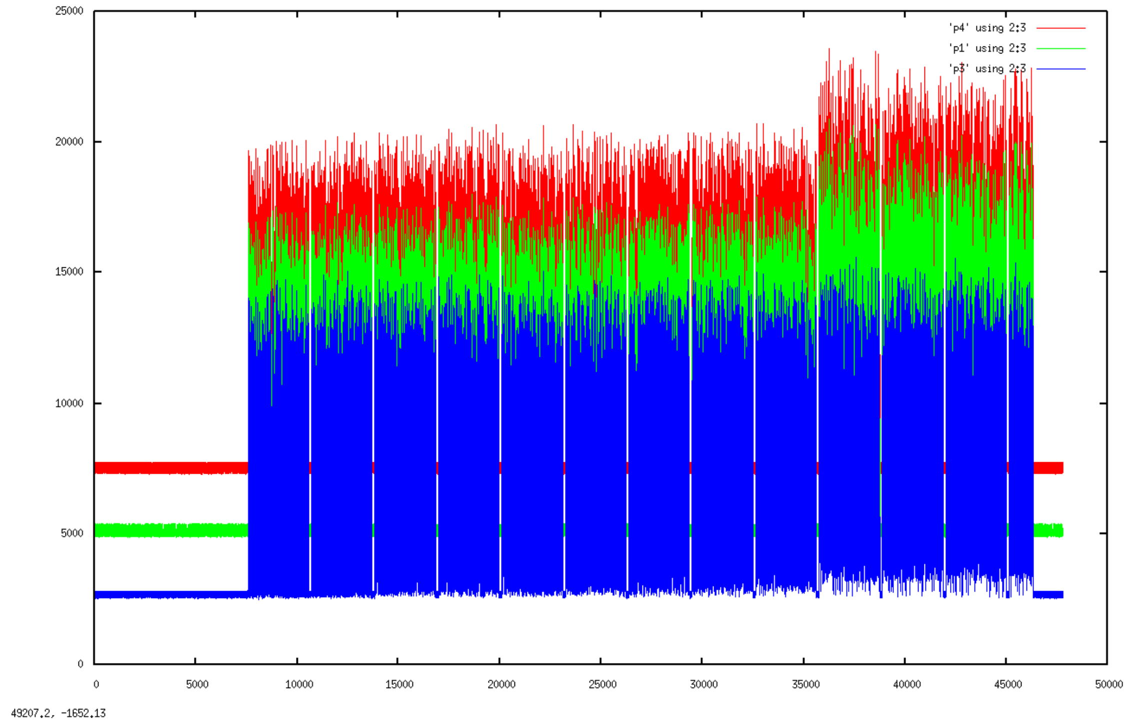
4.3 High Traffic (98% nominal and more)

Note: I won't consider VME nodes here, as they are very misbehaving

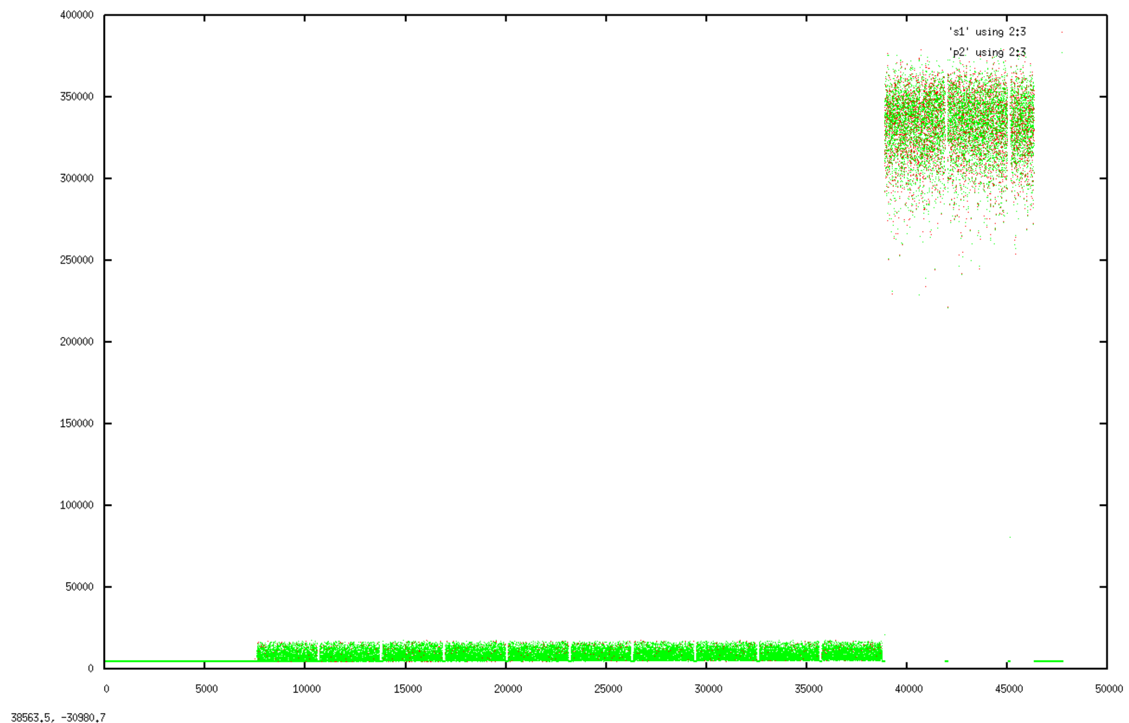
The situation changes abruptly when the reported load is 98%. It looks like we already saturated the gigabit channel, because there is no difference at all in frame loss and latency between the three bursts.

It is interesting to note how the behavior differs between the nodes connected to WRS 1 and those connected to the other nodes. The latency from the ltest sender connected to WRS2 to the receivers in WRS1 jumps up to 380 microseconds, while The latency towards WRS2, WRS3 and WRS4 increases but remains bound to around 22 microseconds, only 15 microseconds more than the situation in the idle case – excluding the lost frames.

As shown in the figure, most of this extra delay is gained within WR2, where both the ltest source and the p3 receiver are connected.



Nodes “p2” and “s1”, connected to WRS1, feature a latency of 225-380 microseconds: never less than 225 and never more than 382. As shown in the following figure (this is drawn with dots instead of lines, to avoid red items to be completely overlaid by green items; the two data sets are quite overlapping)



At least 350 of these 380 microseconds are spent in the WRS2-to-WRS1 link and the input queue of the port of the node. Interestingly, the WRS2-to-WRS1 link is completely empty.

This is the result for the worst tuple ever, from all nodes that are still alive (which includes “u1”.

```
p5:lat:     44834   3939.495  11587.495
p3:lat:     44834   3949.331  11645.331
```

```
p1:lat:     44834   9046.831  14502.831
p4:lat:     44834  11829.184  17285.184
s2:lat:     44834  11948.873  17420.873
p2:lat:     44834 371187.596 373123.596
s1:lat:     44834 378901.223 380837.223
u1:lat:     44834 379555.553 381491.553
```

Here, going up to WRS1 and into the node takes more than 360 microseconds more than just going into a node. This amounts to a “queue time” of 360000 bytes. With a queue memory size of 64kB (`g_mpm_mem_size = 65536` in `scb_top_bare.vhd`, this accounts for 6 queue sizes.

Even the “fastest” frame in the 98% burst takes 216000 bytes of “queue time” to go up one switch. I have no explanation for this behavior.

```
p5:lat:     40581  13299.494  10819.494
p3:lat:     40581  13309.341  10845.341
p1:lat:     40581  18022.834  15030.834
p4:lat:     40581  20917.183  17845.183
s2:lat:     40581  21020.883  17932.883
s1:lat:     40581 229109.228 238165.228
p2:lat:     40581 229123.598 238179.598
```

Lost frames during the three bursts are:

	98% burst (3000 frames)	99% (3000 frames)	100% (1180 frames)
s1 (WRS1)	59	44	36
p2 (WRS1)	60	45	36
u1 (WRS1)	60	49	23
p3 (WRS2)	0	0	0
p5 (WRS2)	0	0	0
p1 (WRS3)	5	6	2
p4 (WRS4)	5	6	2
s2 (WRS4)	5	6	2

The lost frames in WRS3 and WRS4 are the same, so they were clearly dropped in the path from WRS2 to WRS3.

4.4 Misbehavior of the VME devices

Three out of 4 of the Vetar devices stopped receiving data during `round2` of the test. In particular, u2 and u3 (both connected to WRS3), stopped receiving at the same time, after tuple 10708. U0, connected to WRS4, stopped after tuple 20102 Both events fall at the very beginning of a traffic burst. The nodes are not really dead: the CPU is still running and responding to console commands. But they stopped receiving any frame.

Both u2 and u3, which died at the beginning of the 80% traffic burst, featured massive data loss during the 70% data burst, counting holes of a few hundred to a few thousand ltest tuples. Similarly, u0, which died at the beginning of the 91% burst, features similar rx holes in previous bursts, up to loosing the whole of the 90% traffic burst:

```
lat: unexpected 16841.2 after 13839.3
```

When making post-mortem analysis, I found that the last received frame in all three nodes is a multcast frame from the traffic generator, with destination mac address 01:aa:bb:cc:dd:ee, which should not reach the CPU according to packet-filter rules. So it looks like the pfilter misbehaves in some way. The fact the two nodes stopped receiving at the same time hints at a data-dependent problem.

In the first test (“`round1`), u1, u2 and u3 all stopped after 28178 events, during idle time. Unfortunately, I didn’t collect memory images for post-mortem analysis.

The fact u1 is connected upstream of the ltest sender (i.e. it lives towards the traffic generator) is not an interesting fact, because the mishap happened while the traffic generator is idle.

Another concerning behaviour of the Vetars during the first test is a complete stop in receiving frames for the duration of the whole bursts of traffic. This is u3 for example (connected to WRS3):

```
lat: unexpected 16712.3 after 13711.3
lat: unexpected 19844.1 after 16842.3
lat: unexpected 22975.1 after 19973.3
```

Each of these bursts of lost messages (the only ones reported by u3), is 3000 tuples long, and the exactly correspond to the three bursts of the traffic generator. U2 (also connected to WRS3) shows an almost identical behaviour (it differs by 1-2 frames from the above).

U0, which is connected to WRS4, thus downstream for u2 and u3 described above, after loosing 1180 tuples at the beginning of the first traffic bursts, recovers and only misses a few frames later one (i.e. it returns back to behaving like pexarias and specs):

```
lat: unexpected 14894.1 after 13711.3
lat: unexpected 14923.2 after 14922.3
lat: unexpected 15043.1 after 15042.2
[...]
```

Actually, some of the messages about individual frames lost match messages from pexarias and specs, confirming those frames were discarded by upper switches due to traffic.

All of this makes me thing that WRS3 has been correctly forwarding the frames, and the problem leading to RX losses is in the vetars.

At the same time, u1, which is upstream towards the traffic generator, only reports losses of individual frames, like other devices. Again, this hints at a data-driven problem.