libosmocore - Bug #5671
rate_ctr: strange per-minute/hour/day e1 tx/rx frame counts

08/30/2022 07:53 PM - cquirin

Status: New
Priority: Normal
Assignee: laforge
Category: Target version:
Spec Reference:

Description
I am seeing this in osmo-e1d

| El Tx Frames multiplexed: 899771261 (8000/s 712004/m 35520752/h 892818365/d) |
| El Rx Frames demultiplexed: 899771232 (8000/s 712000/m 35520752/h 892818352/d) |

8000 * 60 = 480000 right?
712000 / 8000 = 89

History
#1 - 08/30/2022 07:54 PM - laforge
this is the vty output of the libosmocore rate_ctr API. so it may very well be a bug in libosmocore affecting all our counters.

There's one global timer in libosmocore which iterates over all rate_ctr and computes the per-minute/hour/day aggregate stats.

#2 - 08/31/2022 10:18 AM - laforge
Looking at the output of some of my long-running osmo-e1d instances:

| tx:frames_muxed: 80257523645 (8011/s 512011/m 49440128/h 748802000/d) El Tx Frames multiplexed |
| rx:frames_demuxed: 80257529819 (8008/s 512008/m 49440136/h 748802000/d) El Rx Frames demultiplexed |

that's also odd:

- 512011/60 = 8533/s
- 49440128/3600 = 13733/s
- 748802000/86400 = 8667/s

So clearly something odd is going on here.

#3 - 08/31/2022 10:27 AM - laforge
So the libosmocore/rate_ctr works like this:

- schedule a libosmocore timer 1s in the future
- timer call-back is rate_ctr_group_intv()
  - increment timer_ticks
  - update the per-second interval stats
  - if timer_ticks % 60 == 0, update the per-minute stats
  - etc. for the higher intervals
- re-arm the timer 1s in the future

So this has the following problems:

- the timer expiration might be delayed by the scheduler
- the processing of the timer takes some time
- thus, in reality, the timer expiration is likely always "some amount more" than 1s, where "some amount" is the sum of the scheduler delay and the sum of the processing time of the timer callback function

so ideally, we would use a periodic timer expiring regularly at 1s intervals rather than multiple consecutive relative 1s timers. That way, the error doesn't accumulate over time.

When rate_ctr was written, we didn't yet have support for periodic timerfd in libosmocore yet, so we had to use what was available (osmo_timer,
which are not repeating).

So this explains why there is some amount of error. However, it does not explain the magnitude of error we're seeing (13733/s is way more than 8533/s).

#4 - 08/31/2022 02:47 PM - cquirin

How about doing a gettimeofday() each time the counter get actually updated? Some math should then allow for the correct values to be deduced. That can be done with or without FP.

#5 - 09/01/2022 06:10 AM - laforge

On Wed, Aug 31, 2022 at 02:47:48PM +0000, cquirin wrote:

> How about doing a gettimeofday() each time the counter get actually updated? Some math should then allow for the correct values to be deduced. That can be done with or without FP.

using a timerfd avoids the need for any of this. Even if the system is stalled for several seconds, it will tell you how many timer intervals (1s in our case) have expired meanwhile. We use this in all the critical timing paths of other osmocom software, just not for the rate_ctr.

However, I still don't think this is the root cause of what we're seeing.

#6 - 09/01/2022 07:47 AM - laforge

- Project changed from osmo-e1d to libosmocore
- Subject changed from strange per-minute/hour/day e1 tx/rx frame counts to rate_ctr: strange per-minute/hour/day e1 tx/rx frame counts

#7 - 09/01/2022 02:59 PM - cquirin

rate_ctl.c:307 contains the following code

```c
/* update the rate of the next bigger interval. This will
* be overwritten when that next larger interval expires */
if (intv + 1 < ARRAY_SIZE(ctr->intv))
    ctr->intv[intv+1].rate += ctr->intv[intv].rate;
```

This code is reasonable when the very first interval has not yet elapsed, as it incrementally fills the higher values with some kind of estimates.

However, as soon as the rate has been computed the regular way, by differentiation, the code should no longer run.

I commented the code to check. I have rates of 0 for the first interval, of course, (meaning that the per day value stays 0 for a whole day) but I do have reasonable values as soon as the first interval has elapsed.