libosmocore - Feature #4591

Export FSM states and statistics with counters

06/07/2020 02:40 AM - ipse

Status:	New	**Start date:**	06/07/2020
Priority:	Normal	**Due date:**	
Assignee:		**% Done:**	0%
Category:			
Target version:			
Spec Reference:			

Description

Working on borken stats export and thinking about adding monitoring to OsmoSTP and MGCP code, I realized that a lot of useful information can be acquired from FSMs if they supported generic statistics export.

This export can be made configurable through VTY on a per-FSM class level to tweak it to specific monitoring needs.

Examples of generic counters/gauges I can think of immediately:

1. Number of instances of a given FSM (gauge for current and counter for allocation/deallocation). Useful for short living FSMs like SCCP connections.
2. Current FSM instance state (gauge). Useful for long-living FSMs like A-interface connections.
3. Counters for transitions to/from states (per state) and received events (per event). Useful for long-living FSMs like lchan's.

History

#1 - 06/07/2020 06:40 PM - laforge

On Sun, Jun 07, 2020 at 02:40:30AM +0000, ipse [REDMINE] wrote:

> Working on borken stats export and thinking about adding monitoring to OsmoSTP and MGCP code, I realized that a lot of useful information can be acquired from FSMs if they supported generic statistics export.

Intresting idea. You most likely know that the FSMs already are auto-added to the CTRL and VTY interface in a generic way, but currently I think only their state as well as the current timer+timeout (if any) are exported?

> 1. Counters for transitions to/from states (per state) and received events (per event). Useful for short-living FSMs like lchan's.

how would you handle such counters? Normally, the approach would be to allocate a new set of counters for every lchan, but then if that lchan gets destroyed, you'd loose the history...

#2 - 06/07/2020 08:08 PM - ipse

- Description updated

laforge wrote:

> On Sun, Jun 07, 2020 at 02:40:30AM +0000, ipse [REDMINE] wrote:
>
> > Working on borken stats export and thinking about adding monitoring to OsmoSTP and MGCP code, I realized that a lot of useful information can be acquired from FSMs if they supported generic statistics export.
>
> Intresting idea. You most likely know that the FSMs already are auto-added to the CTRL and VTY interface in a generic way, but currently I think only their state as well as the current timer+timeout (if any) are exported?

Yes. Plus general information like parent and child FSMs, IDs, list of events, etc.

My idea is mostly about exporting the gauges and counters to statsd but they can be used to enrich the VTY output as well, of course.

> 1. Counters for transitions to/from states (per state) and received events (per event). Useful for short-living FSMs like lchan's.

> how would you handle such counters? Normally, the approach would be to allocate a new set of counters for every lchan, but then if that lchan gets destroyed, you'd loose the history...

I was rather thinking about linking state transition counters to the FSM classes, not instances - so that they survive FSM instance death. But now I'm not sure this is a correct approach.

lchan's and timeslots are actually quite long-leaving FSMs. lchan's live while the OML link is up, and timeslots are never destroyed at all. At the same time, we want to see stats *at least* on the per-BTS granularity. Better yet - even with timeslot/subslot granularity in some cases. Which means these stats should be at the FSM instance level.

re: losing history
At least in our environment, we don't care too much about losing history for long-living FSMs like lchan's because we export it to statsd and store/analyze externally. And there we have a full history, including from previous app instances. From our perspective, VTY is useful for digging deeper once we notice something wrong in the exported stats. So having a full history in VTY is not a goal in our use case - it's more important for us to be able to get access to detailed internal status/state via VTY to assist debugging.