Robotics Service Bus - Enhancement #545
Check if pure TCP Connector would improve performance for simpler RSB setups

09/05/2011 11:37 AM - S. Wrede

Status:	Resolved	**Start date:**	09/05/2011
Priority:	Normal	**Due date:**	
Assignee:	J. Moringen	**% Done:**	100%
Category:	Performance Optimization	**Estimated time:**	0.00 hour
Target version:			

Description

A typical usage pattern is to have some components running in an in-process composite on an embedded PC (usually part of the robot) and another set of components running on a connected cluster or workstation ideally also running in an in-process composite. Probably, in this configuration using spread introduces unnecessary overhead.

Related issues:

Related to Robotics Service Bus - Feature # 711: Implement socket-based trans...	**Resolved**	**11/10/2011**
Related to Robotics Service Bus - Feature # 712: Implement socket-based trans...	**Resolved**	**11/10/2011**
Related to Robotics Service Bus - Feature # 713: Implement socket-based trans...	**Resolved**	**07/06/2012**
Related to Robotics Service Bus - Tasks # 715: Integrate socket-based transpo...	**Resolved**	**11/10/2011**
Blocked by Robotics Service Bus - Feature # 710: Implement socket-based trans...	**Resolved**	**11/10/2011**

History

#1 - 10/13/2011 10:39 PM - S. Wrede

- Status changed from New to Feedback

- Assignee set to J. Moringen

- % Done changed from 0 to 20

According to our initial benchmarking, this definitively is the case. Hence, we should check what would be necessary to finalize it in C++ and how complex corresponding implementations in Java and Python would be.

Jan: Could you provide us with an estimate of the complexity?

#2 - 10/13/2011 11:21 PM - J. Moringen

Implementing the basic functionality (one client process, one server process, multiple connectors in each) is not too hard. I actually did most of it for Python already.
However, my experience from the complete implementation in CL is as follows: beyond the basic functionality, it is rather hard to get things right. Examples of harder problems include:

- Closing sockets properly (e.g. errors during closing a socket while already handling an error)
- Handling errors properly
- Managing connecting and disconnecting clients
  - In a thread-safe manner
  - Ensuring ordering constraints for events when connections change
- Disposal of resources and objects was rather difficult in C++ and CL as well

If the main focus of having the connector is performance, optimizations would probably also be required. However, these would likely make the abofementioned problems even harder.
In summary, I think the amount of work strongly depends on the desired

- robustness
- functionality

- degree of optimization

Robust, complete und fully optimized implementations in all languages would be very time consuming. Minimal, unoptimized implementations are not that hard (as the recent benchmarking experiments have shown).

#3 - 11/10/2011 02:46 AM - J. Moringen

- % Done changed from 20 to 70

We can say with reasonable certainty that a socket-based transport is more efficient than Spread as long as the number of clients is small. This is particularly true for the latency (which should not come as a suprise since one hop is eliminated).

Note: the TCP_NODELAY socket option can be used (and is in the CL and C++ implementations) to make a tradeoff between troughput and latency.

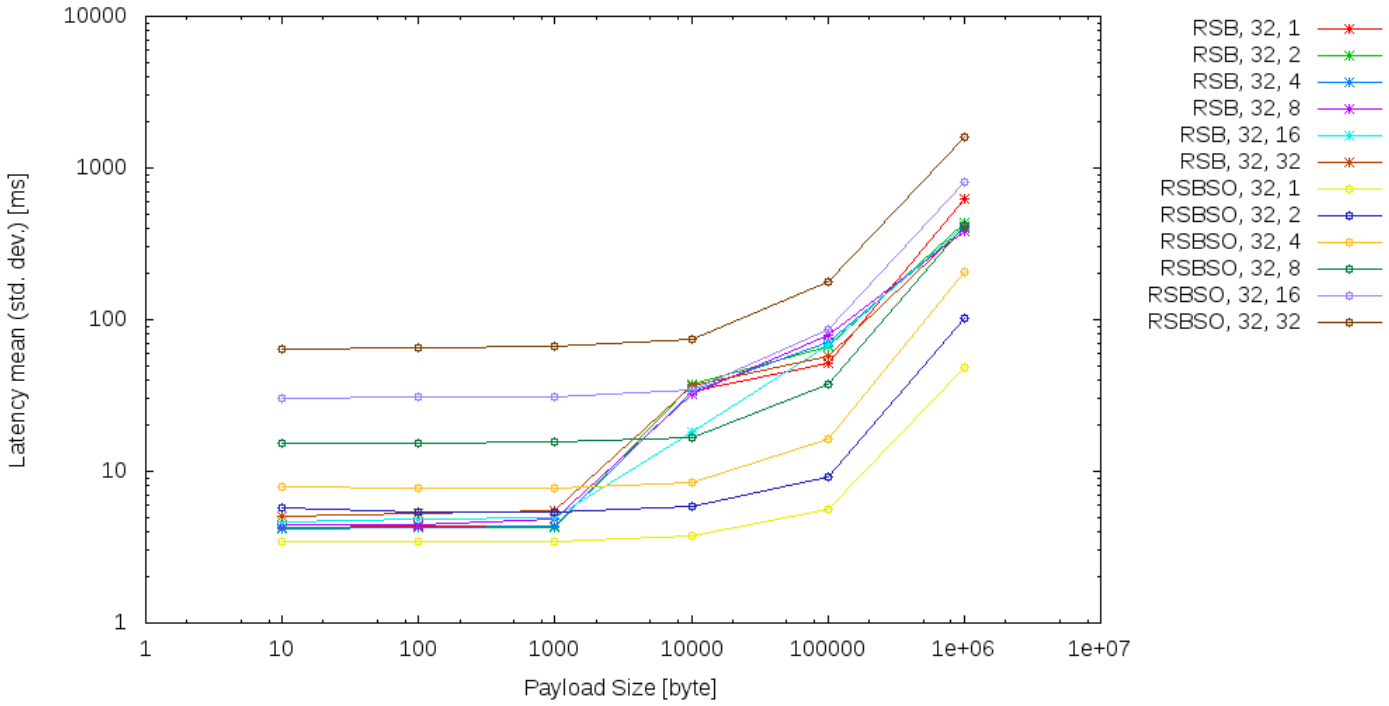
#4 - 11/16/2011 02:56 AM - J. Moringen

- File jaspis-remote-fullspeed.png added

- Status changed from Feedback to Resolved

- % Done changed from 70 to 100

Benchmarks support the hypothesis that a socket-based connector can improve performance for use cases involving a small number of processes:



Files

jaspis-remote-fullspeed.png	10.5 KB	11/16/2011	J. Moringen