OpenSplice | DDS

Empirical Evaluation of RMI Frameworks

Nawel Hamouche
Senior Engineer
PrismTech
Nawel.Hamouche@prismtech.com

Agenda

- RMI fundamentals
- DDS RMI introduction
- OpenSplice RMI implementation
- OpenSplice RMI Performance
- DDS RMI use cases
- Conclusion

RMI Fundamentals

What is RMI ?

- The general concept of invoking a remote object operation
- A powerful and popular technology for developing distributed service-oriented applications
- A complementary paradigm to data centricity used beyond Client/Server Systems
- A client invokes (calls), across the network, a remote method (procedure) transparently as if it was local bypassing network burden

Genesis …

mid 80's ➢ Sun MicroSystems extends BSD unix with **RPC** facilities to build NFS and NIS

1988 ➢ The Open Software Foundation (OSF) specifies the distributed Computing Environment with **DCE RPC** as the basic communication mechanism

90's ➢ The Object Management Group (OMG) specifies **CORBA** for distributed OO applications. Many commercial and open source implementations has emerged like **TAO**

➢ Sun issued **Java RMI** for distributed Java applications

Since 2000 ➢ ZeroC issued **ICE** framework as an enhanced derivative of CORBA

➢ RMI paradigm widely used in Component-based and Service-oriented architectures

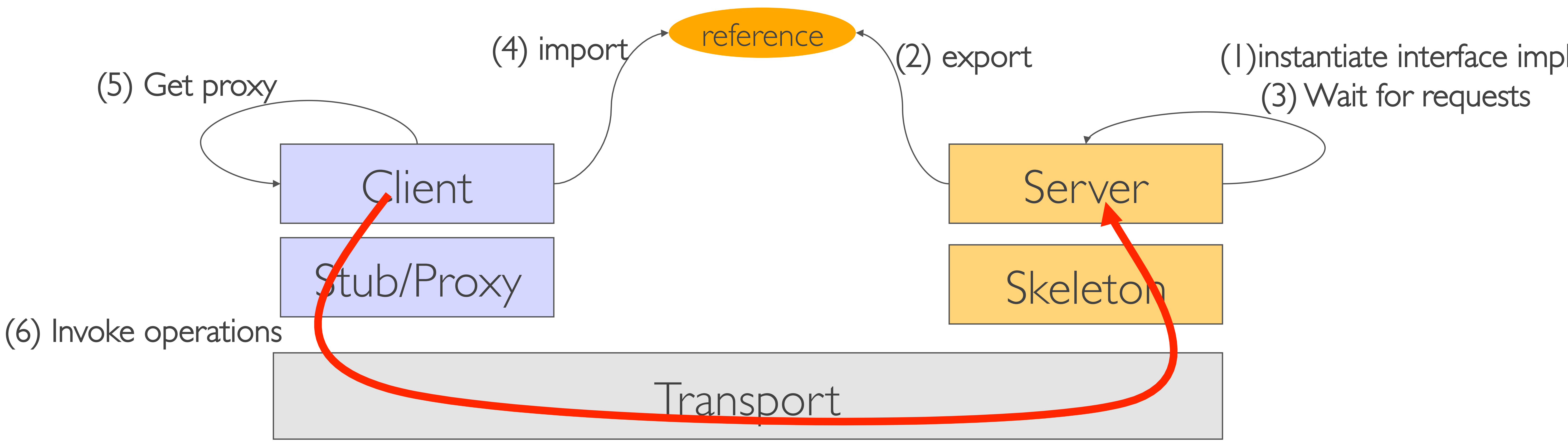
➢ Emergence of the **Data Distribution Service** for loosly-coupled asynchronous systems

RMI Concepts

- Server interface
  - Client/Server contract
- Interface endpoint reference
  - Network and interface addressing information
- Common communication protocol
  - Connectionless or connection-oriented protocol
- Programming languages and API

TAO	ICE	Java RMI
OMG IDL	Slice	Java
IIOR	host:port + oid	name
IIOP, …	Ice Protocol	RMI-IIOP
C++	C++, Java, C#, …	Java

RMI Execution Model



DDS RMI introduction

OpenSplice | DDS

DDS RMI as a future standard

- Remote Method Invocation over DDS
- Using DDS as a Distributed Services Space
- Using DDS mechanisms to export, find and invoke services
- Mapping Client-to-server exchanges to DDS topics
- Takes benefit of DDS for discovery, fault tolerance and one-to-many invocations
- OMG DDS RMI RFP (MARS/2012-03-33) currently in progress

DDS RMI benefit

- Provides a higher abstraction level than achieving such paradigm manually through topic exchanges and applications synchronization
- A unique middleware technology to mix the Global Services and Data Spaces with an easy and dynamic services registration, data declaration and same discovery mechanisms
- Allows data-centric applications to use RMI without the burden of an additional middleware
- Strong services location transparency (services can be referenced by name only)
- Simple API and Easy deployment process
- A solid foundation for :
  - Distributed Administration tools: Deployment, Supervision, (Persistent) Naming service, …
  - Full DDS-based component platforms
  - Replicated servers
  - … even RPCs !

OpenSplice RMI

- Initial implementation of DDS RMI, available as an add-on in the OpenSplice DDS product of PrismTech
- C++ and Java implementation
- Services interfaces are specified in IDL2
- Synchronous and asynchronous communication modes supported
- Service invocation framework on top of DCPS (and DDSI)
- Simple and intuitive client/server programming model
- Ability to tune request/reply DDS Qos policies via XML
- Future versions may associate RMI QoSs at the service level and map them on DDS level (ex : operation priority, …)

OpenSpliceRMI example

Step 1

- **Describing the service interface in IDL**

```
local interface HelloWorld : ::DDS_RMI::Services {
    void sayHello(in string msg);
}
```

- DDS Qos policies can be associated to each operation request and/or reply in an XML file

Step 2

- **Compiling the service description**
  - ***rmipp*** pre-processor generates corresponding request and reply topics as well as corresponding stub/skeleton to handle the operations invocation

```
      <DDS topics>
struct sayHello_request {
  RequestHeader header;
  string msg;
};
struct sayHello_reply {
   RequestHeader header;
};
```

```
<stub>
HelloWorldInterfaceProxy
```

```
<skeleton>
HelloWorldInterface
```

Step 3

- Implementing the service interface

```
Class HelloWorld_impl : public virtual HelloWorldInterface
{
  public:
   virtual void sayHello(DDS::String msg);
}
```

OpenSplice | DDS

Step 4

- Writing the Server code

```
// RMI runtime init
CRuntime_ref runtime = CRuntime::getDefaultRuntime();
Runtime->start(argc, argv);

// interface implementation instanciation
HelloWorld_impl * impl = new HelloWorld_impl();

// interface registration
DDS_Service::register_interface<HelloServiceInterface,
    HelloService_impl> (impl, “HelloServer”, server_id);

// interface activation
DDS_Service::run(“HelloServer”);
```

OpenSplice | DDS

Step 5

- Writing the Client code

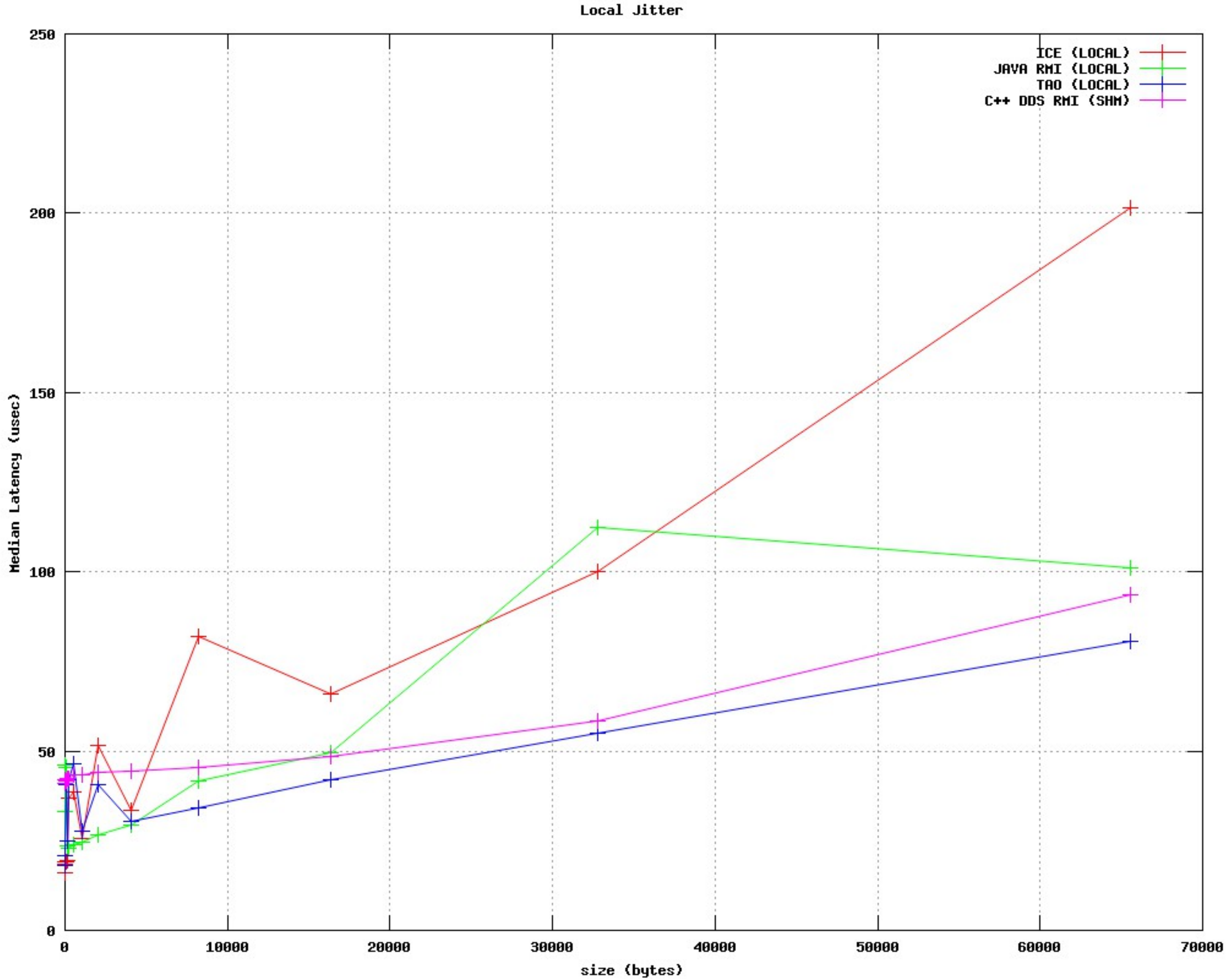
```
// RMI runtime init
CRuntime_ref runtime = CRuntime::getDefaultRuntime();
Runtime->start(argc, argv);

// Getting the interface proxy
shared_ptr<HelloServiceInterfaceProxy> proxy;

DDS_Service::getServerProxy<HelloServiceInterfaceProxy>
    (“HelloServer”, proxy_id, proxy);

// Calling the remote interface
proxy->sayHello(“Bonjour !”);
```

OpenSpliceRMI Performance

Reference Platform

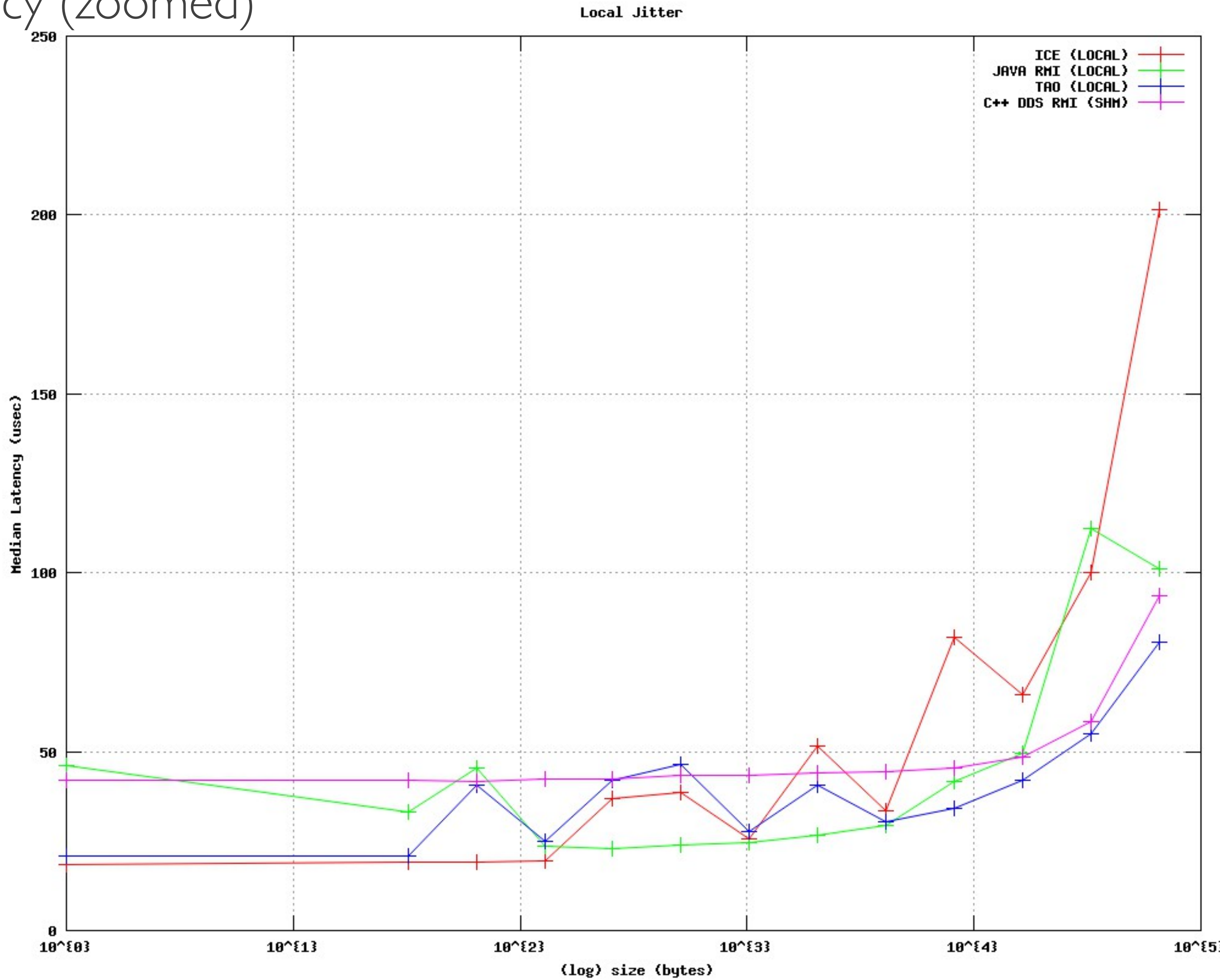
- Hardware and Systems
  - 2 Nodes : DELL Latitude E5410, Intel Core i7, 2.67Ghz
  - OS: Mandriva Linux 2.6.33.7-desktop-2mnb, x86_64 GNU/Linux
  - Network : Ethernet cross cable 1000 Mbps
- Software
  - **DDS RMI** , OpenSpliceDDS, version 6.1.0p3
  - **TAO 6.0.0 , ICE 3.3.1, Java RMI JDK 1.6.0_26**
- Benchmark
  - Remote and local Client + Server applications
  - Two-way operation : `OctetSeq test_method_octetseq(in OctetSeq data);`
  - Single threaded configuration
  - Borrowed the "TAO/performance-tests/Latency/Single_Threaded" configuration
  - Latency and jitter

Local Latency



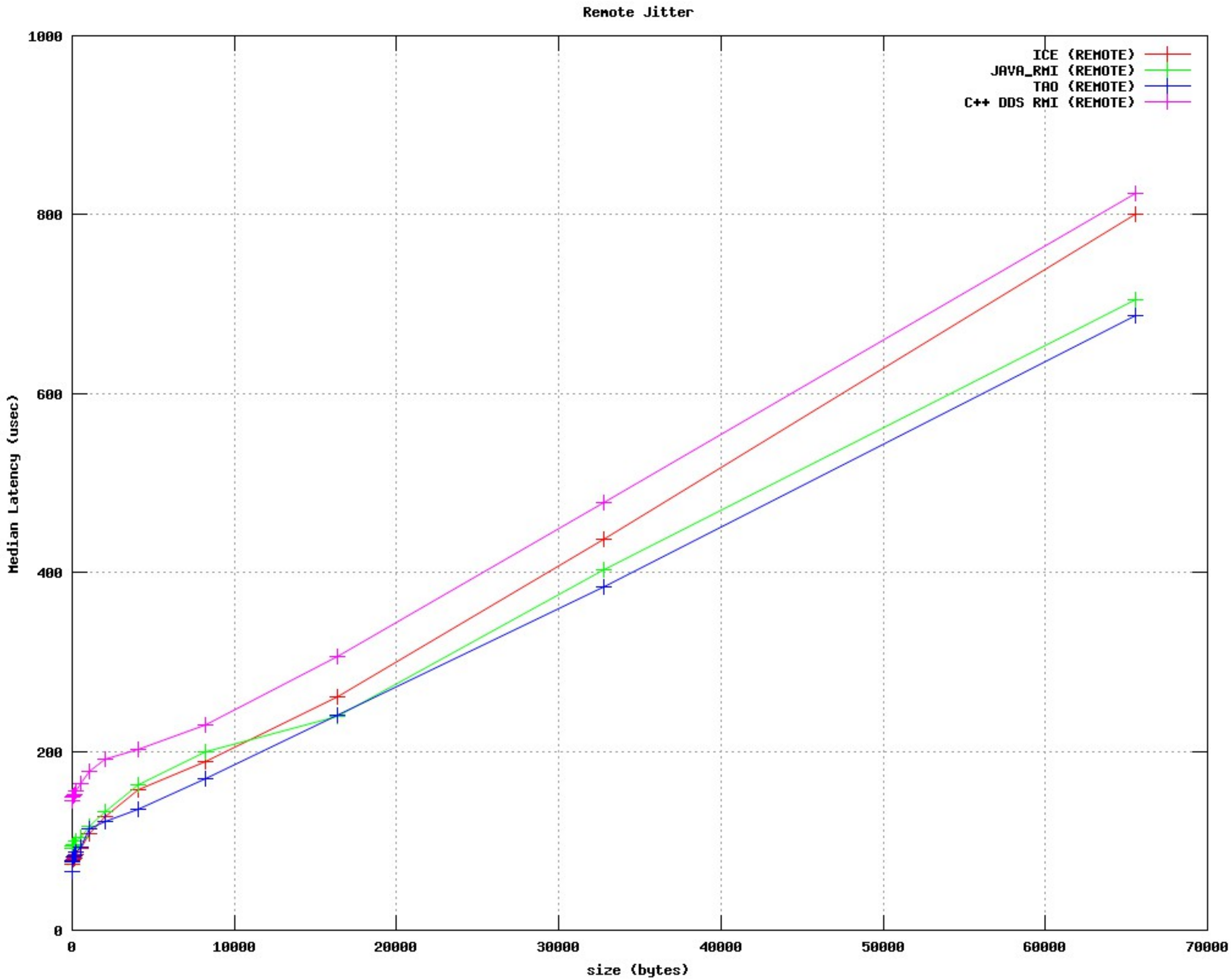
OpenSplice | DDS

Local Latency (zoomed)

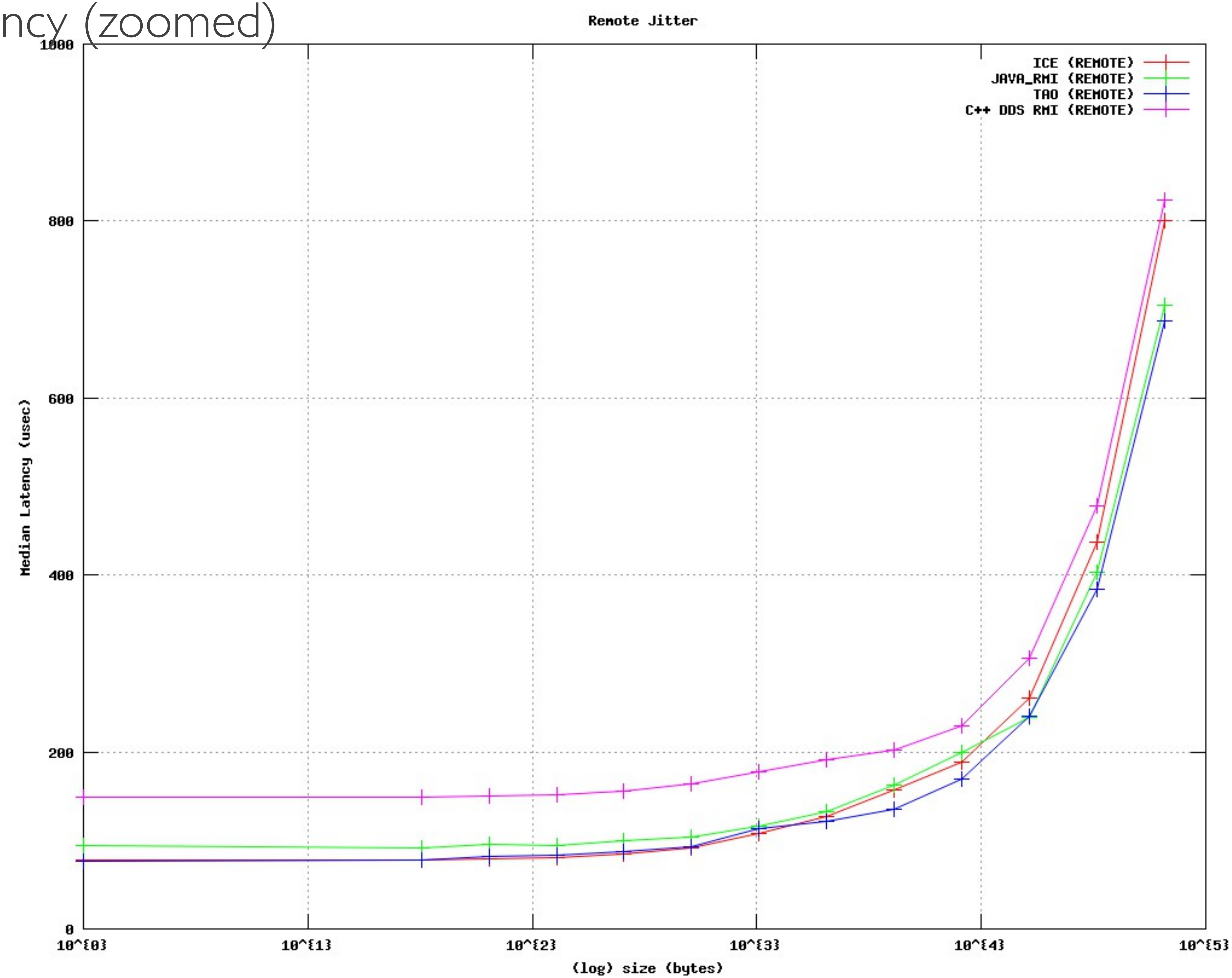


OpenSplice | DDS

Remote Latency



Remote Latency (zoomed)



OpenSplice | DDS

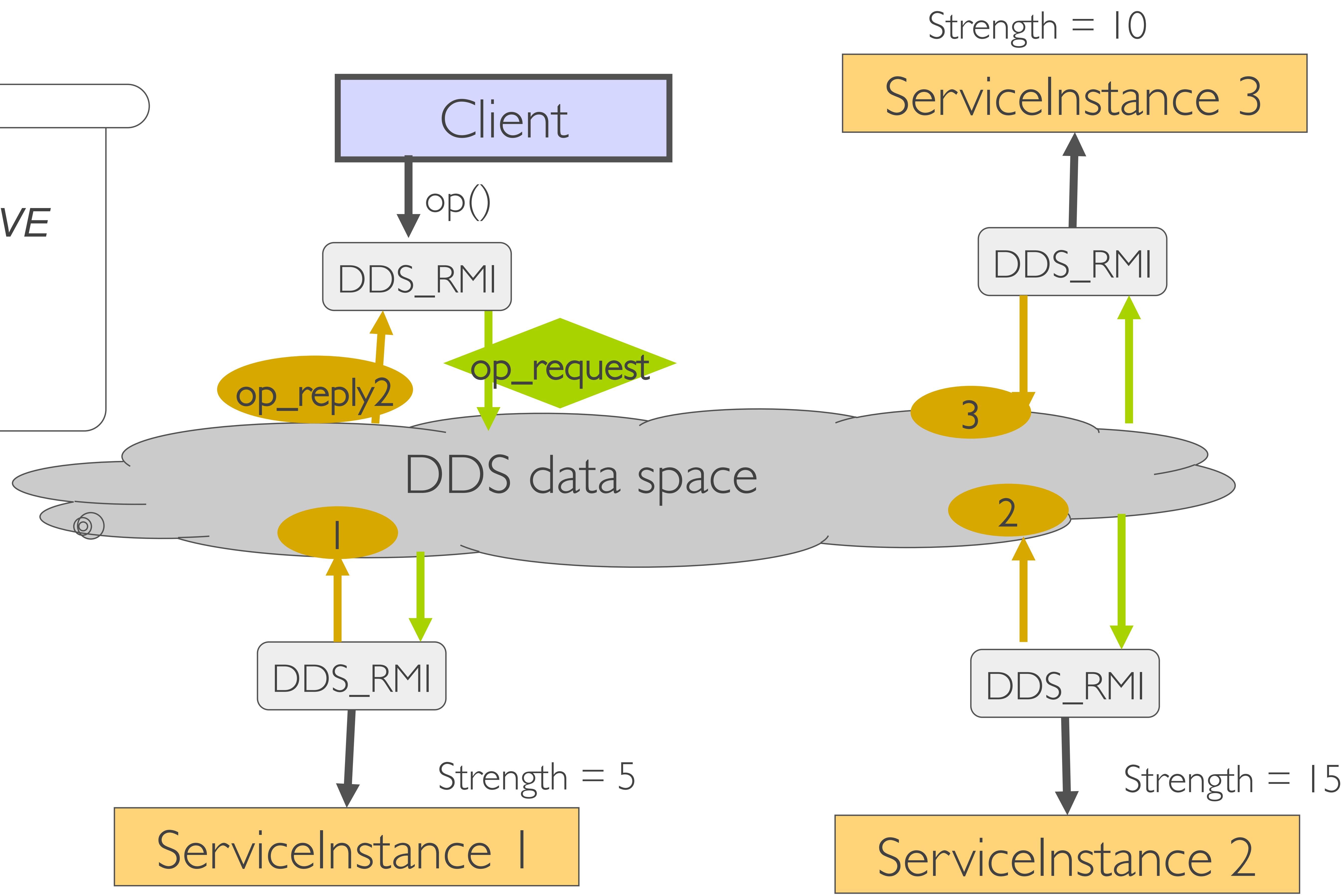
Performance comparison results

- Globally, TAO is the fastest framework but remains close to ICE and Java RMI.
- OpenSplice RMI is 20-10 µs slower than TAO locally (resp. remotely).
- Considering that the performance of competing technologies have been optimized over the past 10+ years, OpenSplice RMI shows some initial very good performance!

Some DDS RMI use cases

Active Replication with DDS RMI

Relevant DDS Qos':
- *Ownership EXCLUSIVE*
- *FT Reliability (*)*
- *Single Client*



Towards a Real Time CBA

- A growing interest in a **CORBA-less Component Model**
- CORBA ORB in CCM can be swapped by a DDS RMI connector (DDS_RMI4CCM) to perform receptacle to remote facet invocations.
- DDS4CCM and DDS_RMI4CCM can provide support to a Real Time Component-Based Architecture inspired from CCM thanks to DDS features.
- Need to review the CCM specification to remove the built-in support to CORBA.

Transparent State Sharing with DDS RMI and DLRL

- DLRL provides a simple and transparent object-oriented view to state dissemination
- Mixing DLRL and DDS RMI allows to expose the DLRL object to remote clients
- RMI interface maps on the DLRL's local interface
- Any RMI invocation that would change the DLRL object state can be automatically disseminated
- Applications that subscribe to the DLRL-associated-topics will get transparently all state changes.
- A full object-oriented DDS-based service-oriented applications.

Conclusion

- Distributed applications still need to communicate via Request/Reply in combination with Publish/Subscribe paradigms
- Many existing and mature frameworks already provide RMI, but they do not support data centricity.
  - In best case , data centricity is emulated (e.g Notification in CORBA)
- DDS RMI provides a 2-in-1 middleware to satisfy both data-centric and service-oriented applications with real-time, fault tolerance and performance Qos'
- OpenSpliceRMI performance is acceptable wrt the fastest RMI frameworks in the market. Future enhancements are planned.