High-performance Distributed Object Computing with Real-time CORBA

Douglas C. Schmidt

Washington University, St. Louis

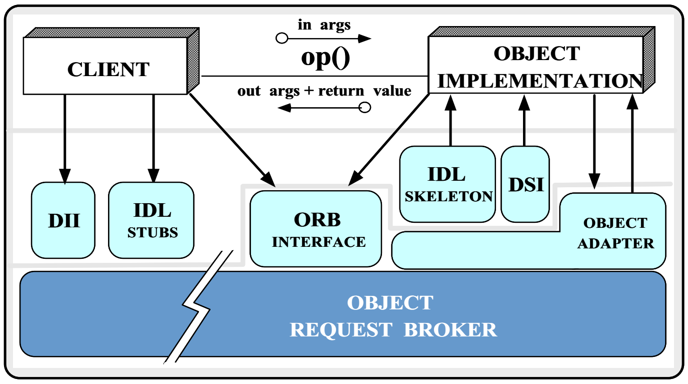
http://www.cs.wustl.edu/~schmidt/

schmidt@cs.wustl.edu

Motivation

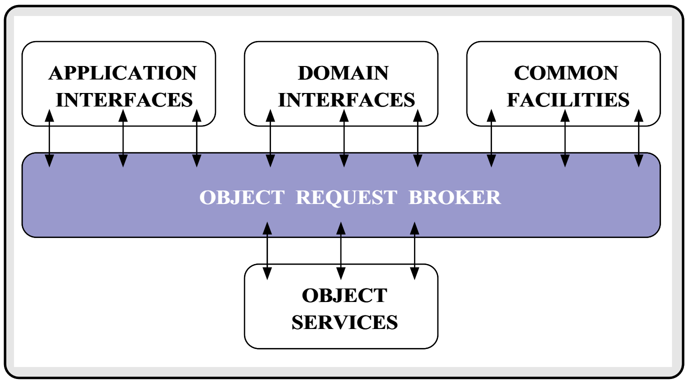
- Typical state of affairs today is the "Distribution Crisis"
  - Computers and networks get faster and cheaper
  - Communication software gets slower, buggier, more expensive
- *Accidental complexity* is one source of problems, *e.g.*,
  - Incompatible software infrastructures
  - Continuous rediscovery and reinvention of core concepts and components
- *Inherent complexity* is another source of problems
  - *e.g.*, latency, partial failures, partitioning, causal ordering, etc.

Candidate Solution: CORBA

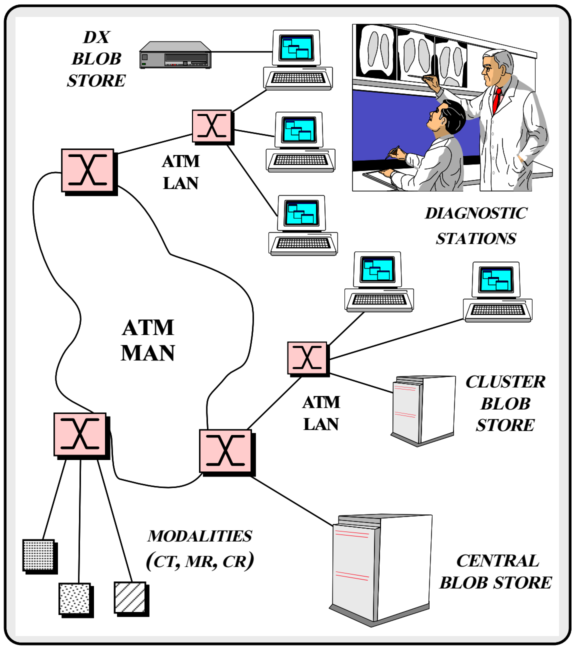


- Goals
  1. Simplify development of distributed applications
  2. Provide flexible foundation for higher-level services

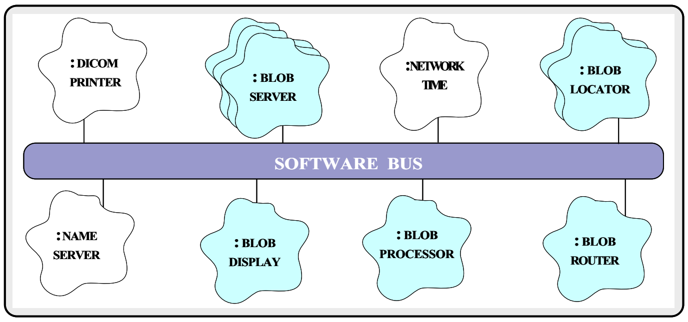
OMA Reference Model Interface Categories



Example 1: Distributed Medical Imaging

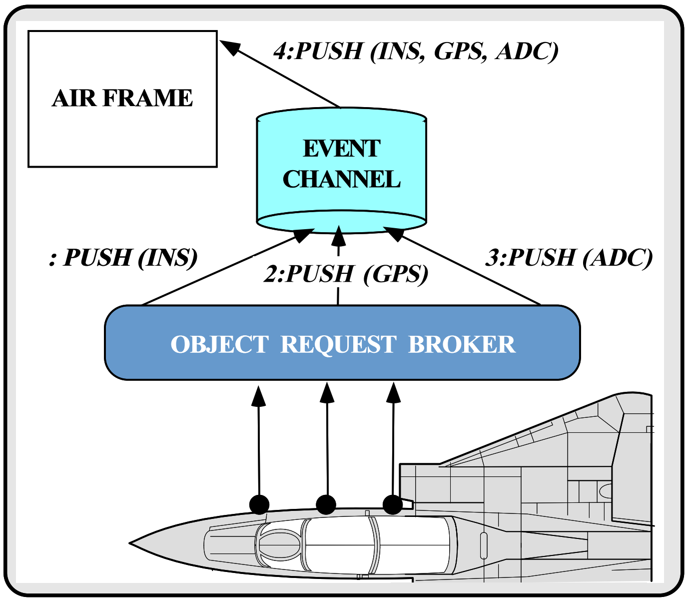


Distributed Objects in Medical Imaging



- "Blob" == Binary Large OBject

Example 2: Real-time Avionics



Observations

- CORBA is well-suited for certain *communication requirements* and certain *network environments*
  - *e.g.*, request/response or oneway messaging over low-speed Ethernet or Token Ring
- However, current CORBA implementations exhibit high overhead for other types of *requirements* and *environments*
  - *e.g.*, bandwidth-intensive and delay-sensitive applications over high-speed networks
- Performance limitations will ultimately impede adoption of CORBA

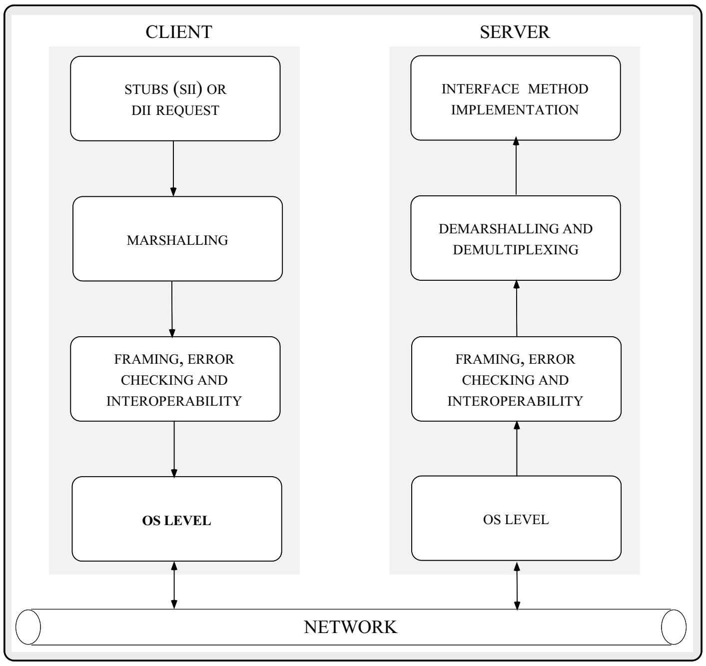
Key Research Questions

- "Can CORBA be used for performance-sensitive applications on high-speed networks?"
  - Goal is to determine this empirically
- "What are the strategic optimizations for Gigabit CORBA"?
  - Goal is to maintain strict CORBA compliance
- "What changes are required to provide Real-time CORBA?"
  - Goal is to provide end-to-end QoS guarantees

Pinpointing CORBA Overhead

- *Presentation layer overhead*
  - *e.g.*, typed and untyped data
- *Data manipulation and data copying overhead*
  - *e.g.*, message management
- *Demultiplexing and operation dispatching overhead*
  - *e.g.*, layered and de-layered demultiplexing
- *OS/network/protocol integration*
  - *e.g.*, ATM/host adapters, resource reservation and scheduling

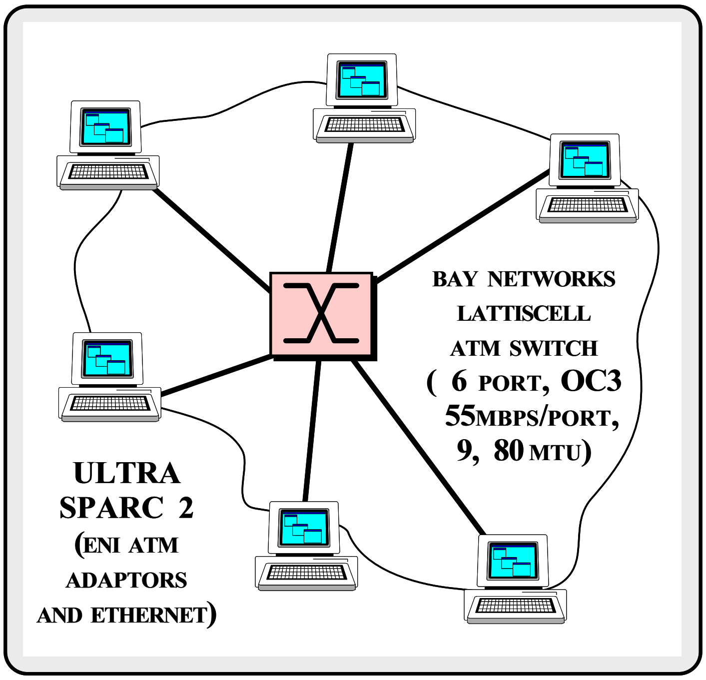
General Path of CORBA Requests



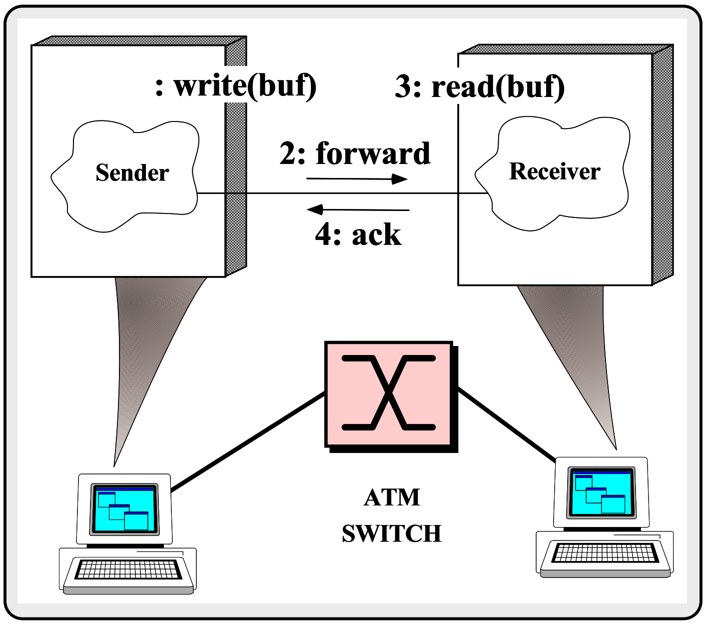
Experimental Setup

- Enhanced version of TTCP
  - TTCP measures end-to-end data/request transfer
  - Enhanced version compares C, ACE C++ wrappers, Orbix 2.0.1 and VisiBroker 2.0, and Blob Streaming
- Parameters varied
  - 100 Mbytes of typed data
    - Types included scalars, floats, structs, and sequences
  - Sender buffer sizes ranged from 1K to 128K
  - Socket queues were 8k (default) and 64k (maximum)
  - Network was 155 Mbps ATM and "loopback"

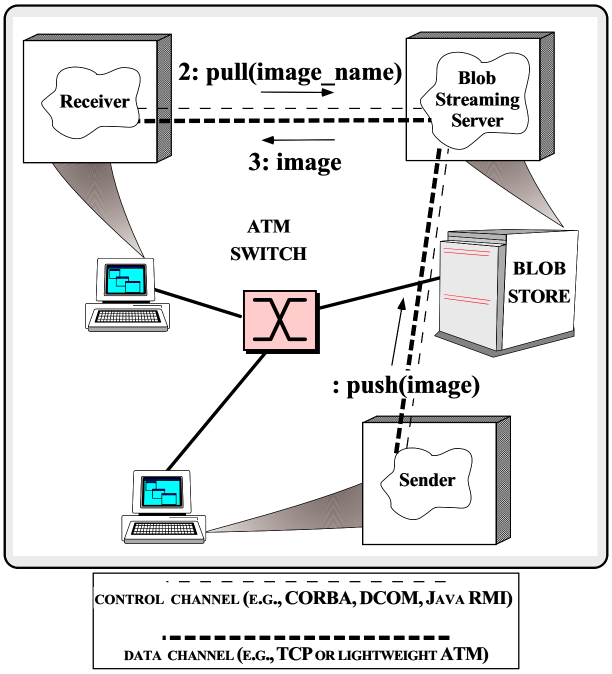
Network/Host Environment



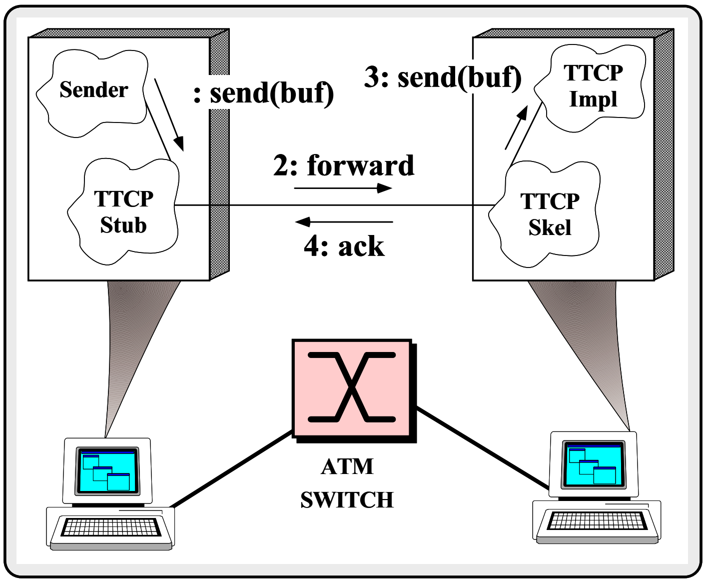
TTCP Configuration for C and ACE C++ Wrappers



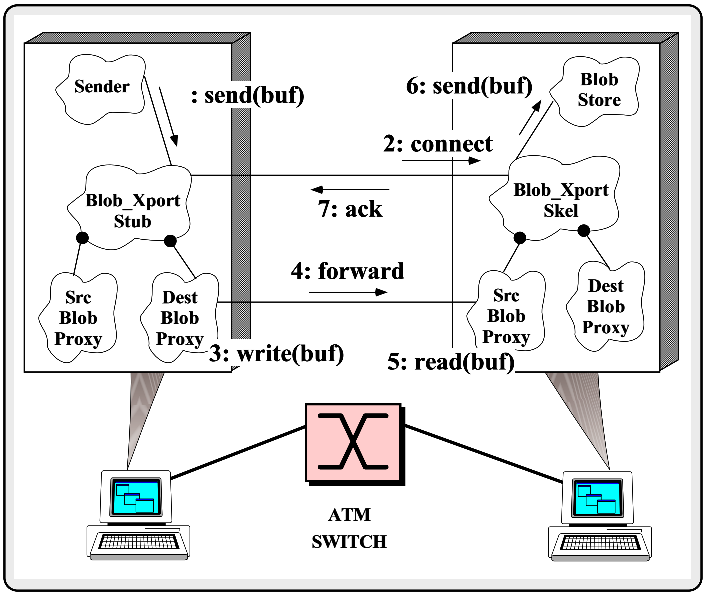
Blob Streaming System Architecture



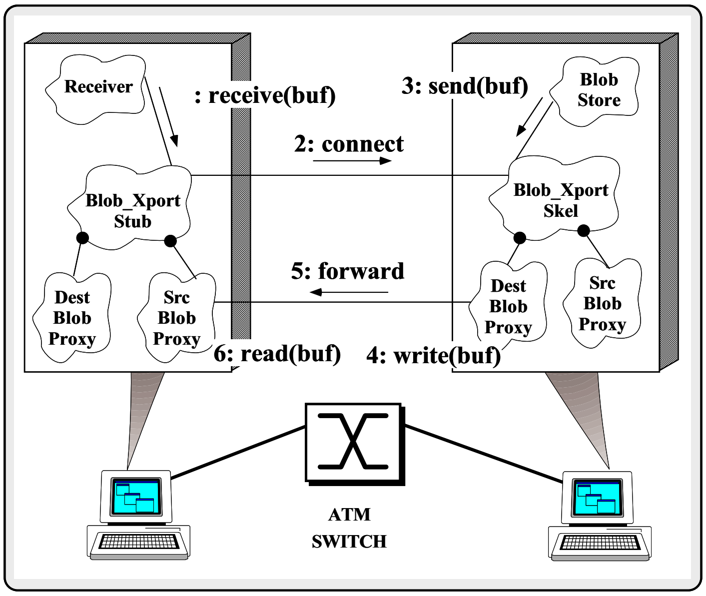
TTCP Configuration for CORBA Implementation



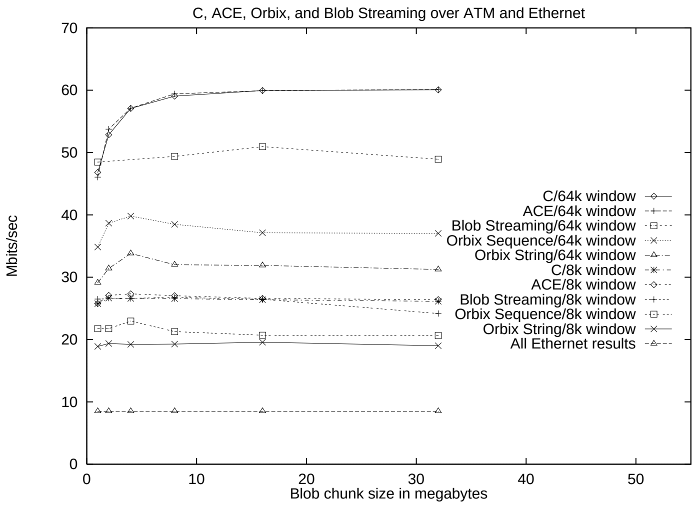
TTCP Configuration for Blob Streaming (Push Model)



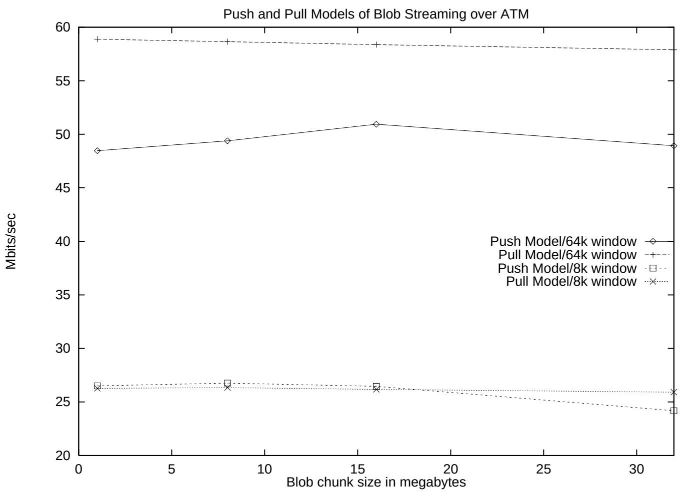
TTCP Configuration for Blob Streaming (Pull Model)



Push Model Performance over ATM and Ethernet



Pull Model Performance over ATM



High-Cost Functions

- C and ACE C++ Tests
  - Transferring 100 Mbytes with 1 Mbyte buffers

Test	%Time	#Calls	Name
C sockets (sender)	93.9	112	write
	3.6	110	read
C sockets (receiver)	93.2	13,085	read
	4.5	102	write
ACE C++ wrapper (sender)	94.4	112	write
	3.2	110	read
ACE C++ wrapper (receiver)	93.9	12,984	read
	5.6	102	write

High-Cost Functions (cont'd)

- Orbix String and Sequence

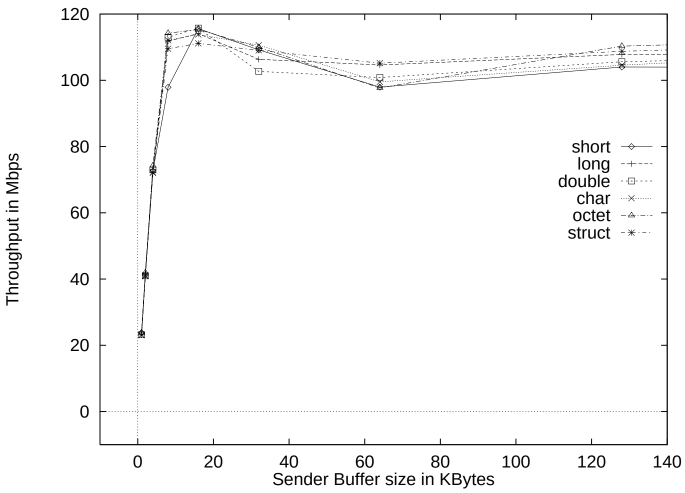
Test	%Time	#Calls	Name
Orbix Sequence (sender)	53.5	127	write
	35.1	223	read
	7.3	1,108	memcpy
Orbix Sequence (receiver)	84.6	12,846	read
	12.4	1,064	memcpy
	3.2	101	write
Orbix String (sender)	45.0	127	write
	35.1	223	read
	10.8	1,315	strlen
	6.0	1,108	memcpy
Orbix String (receiver)	70.7	12,443	read
	16.1	2,142	strlen
	10.0	1,064	memcpy
	3.0	101	write

High-Cost Functions (cont'd)

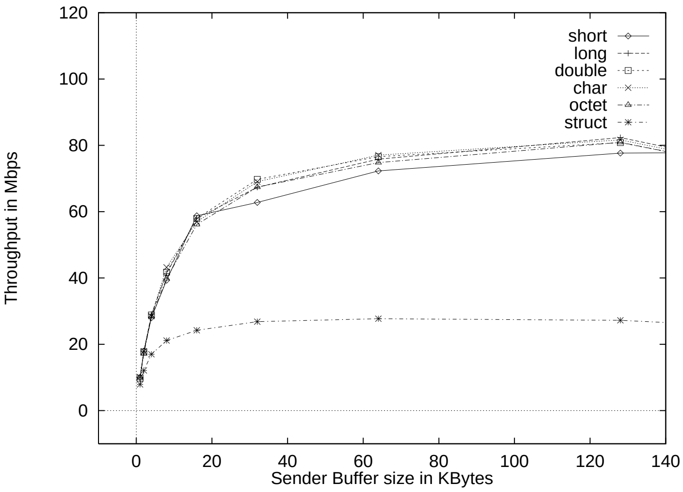
- Blob Streaming

Test	%Time	#Calls	Name
BlobStreaming (sender)	48.8	327	write
	44.8	232	read
	1.3	2,055	memcpy
BlobStreaming (receiver)	77.2	12,546	read
	16.4	12,734	memcpy
	1.4	102	write

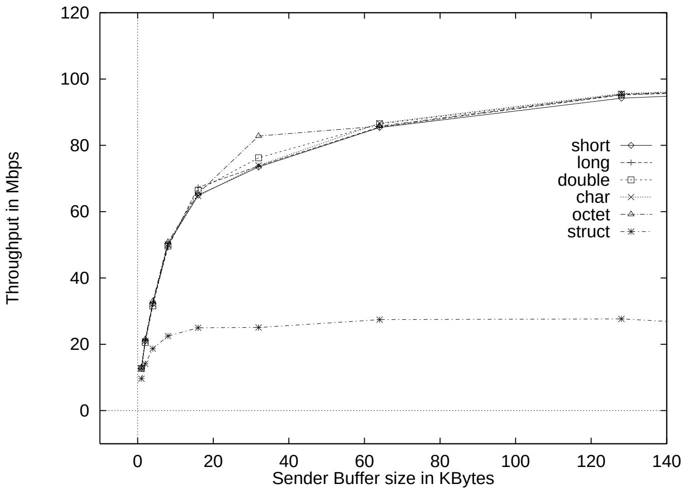
C/C++ Remote Data Transfer Results



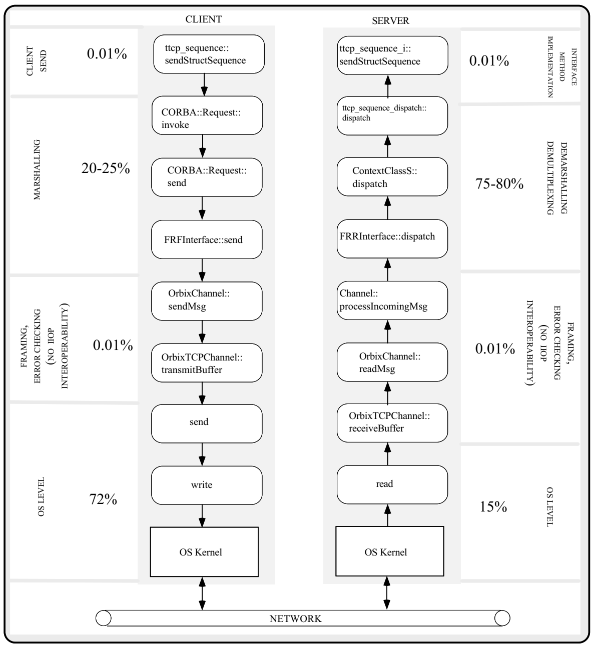
Orbix Remote Data Transfer Results



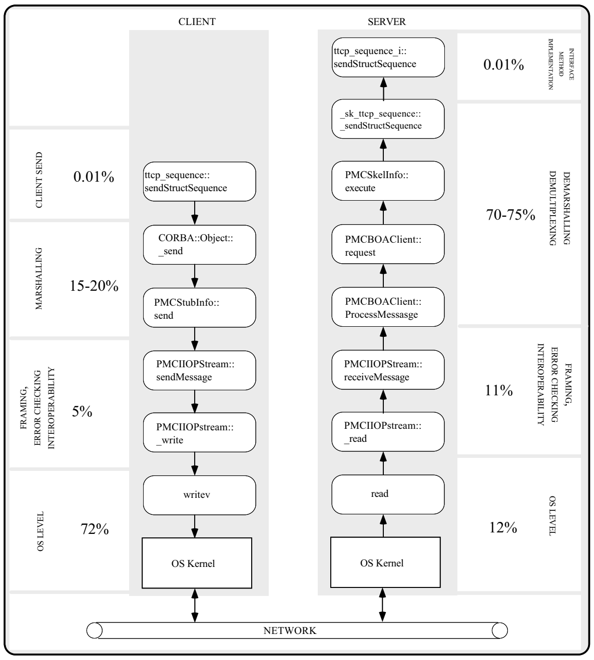
VisiBroker Remote Data Transfer Results



Analysis of Orbix Overhead



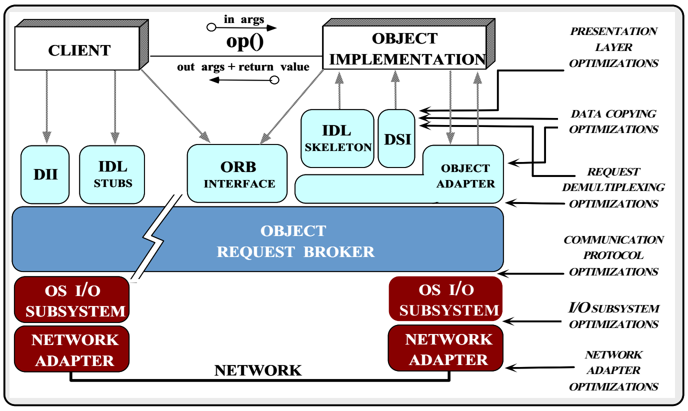
Analysis of VisiBroker Overhead



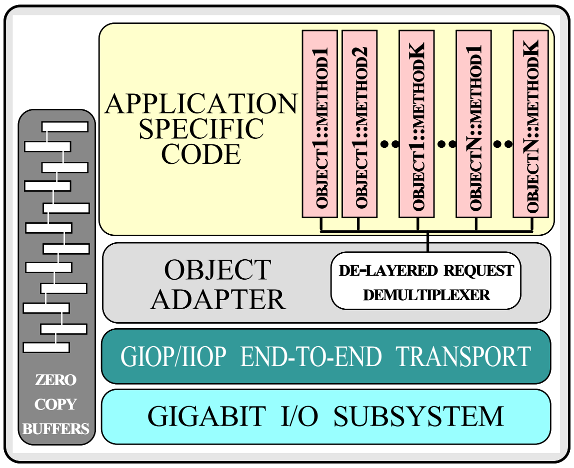
Summary of Throughput Results

- For bytestreams
  - VisiBroker performed > 80% *cf* C/C++ versions
  - Orbix performed 70% *cf* C/C++ versions
- For scalars and floats
  - *Remote*: 75–80% *cf* C/C++ versions
  - *Loopback*: Orbix performed 65–68% *cf* C/C++ versions, VisiBroker achieved similar throughput for large sender buffer sizes
- For structures
  - *Remote*: 31% *cf* C/C++ versions
  - *Loopback*: 16% *cf* C/C++ versions

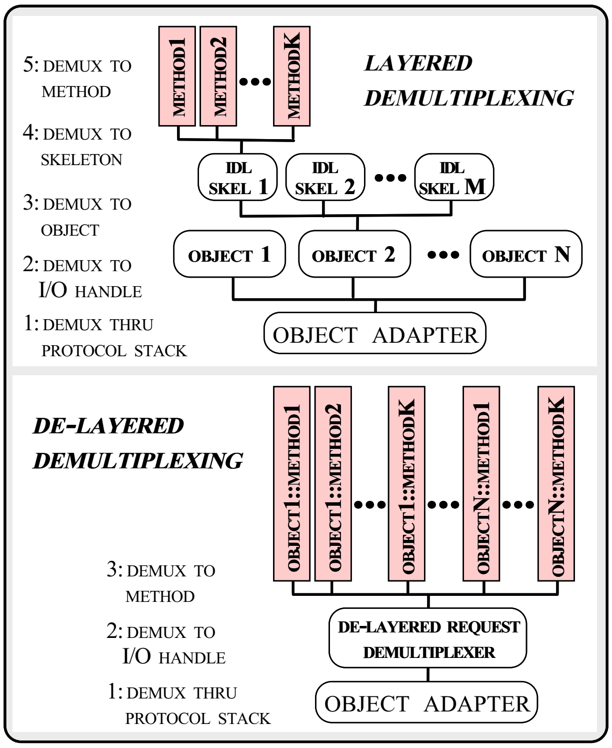
Gigabit CORBA Optimizations



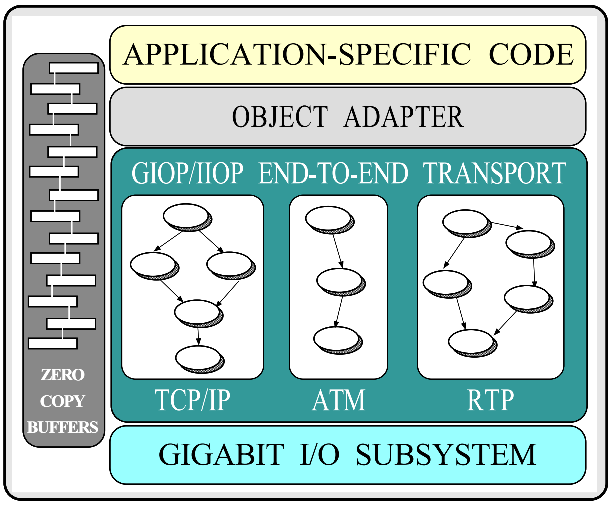
Real-time CORBA



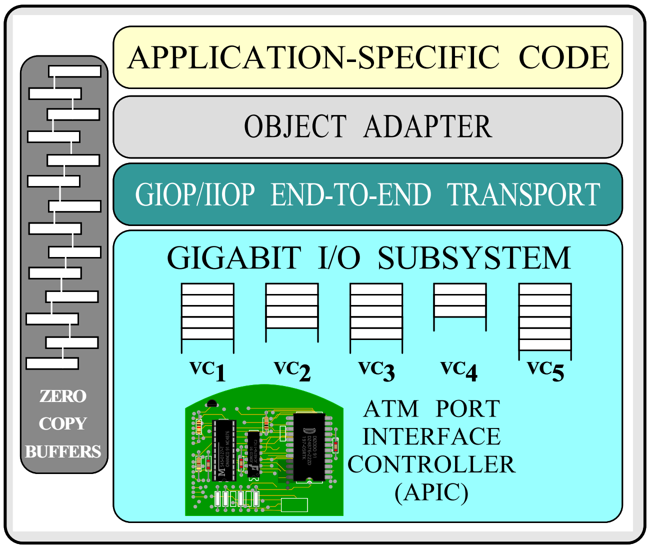
Demultiplexing Optimizations



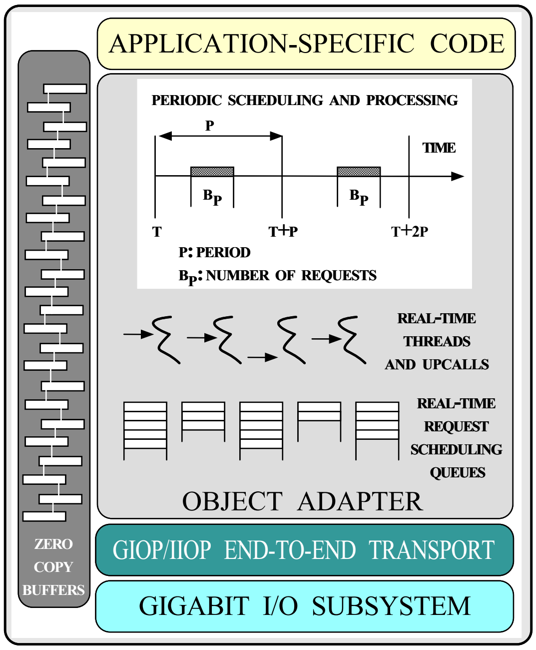
Multi-Protocol Support



I/O Subsystem Optimizations



Real-time Scheduling Optimizations



Concluding Remarks

- CORBA is a promising architecture for distributed computing
- Conventional CORBA implementations are not tuned for high-performance or real-time systems
  - Note, low-speed networks often hide performance overhead
- Ultimately, an integrated approach is the best solution
- Optimizations must be applied at multiple layers
  - *e.g.*, network/OS/protocol/ORB