Optimizations for High Performance ORBs

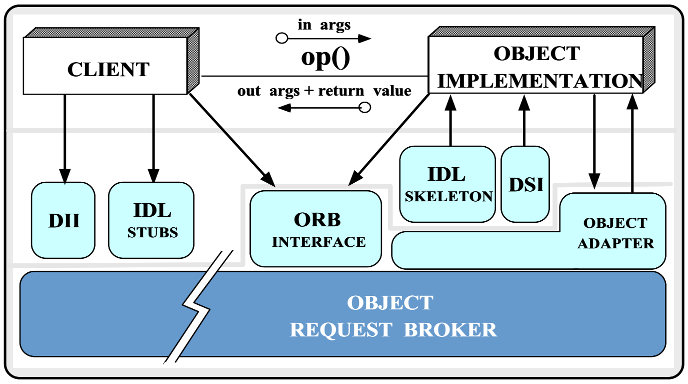
Douglas C. Schmidt
(www.cs.wustl.edu/~schmidt)
Aniruddha S. Gokhale
(www.cs.wustl.edu/~gokhale)

Washington University, St. Louis, USA.

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

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

Performance Problems with Existing ORBs

- Existing ORBs lack certain features (*e.g.*, end-to-end QoS)
- Reasons for poor performance
  - *Inefficient server demultiplexing techniques*
  - *Long chains of intra-ORB function calls*
  - *Excessive presentation layer conversions and data copying*
  - *Non-optimized buffering algorithms used for network reads and writes*
  - *Improper choice of underlying operating system interfaces*
  - *Lack of integration with advanced "OS" and network features*

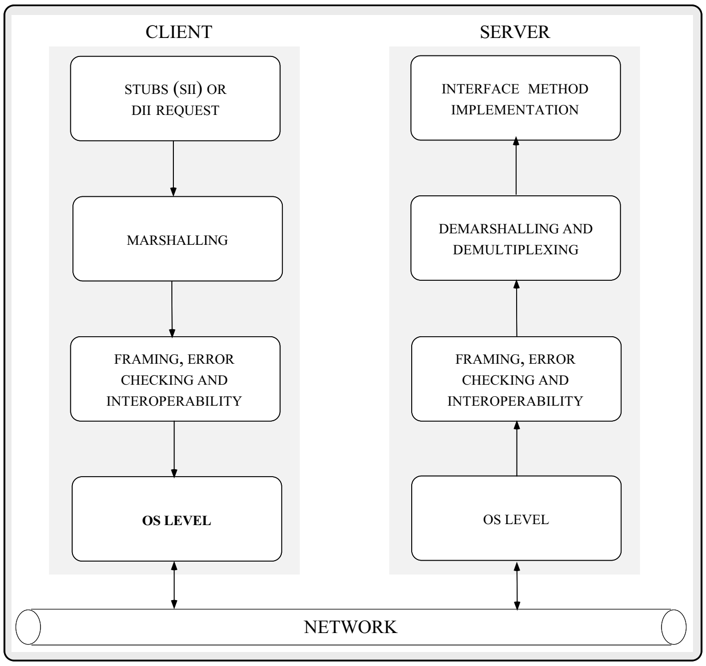
Key Research Questions

- "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

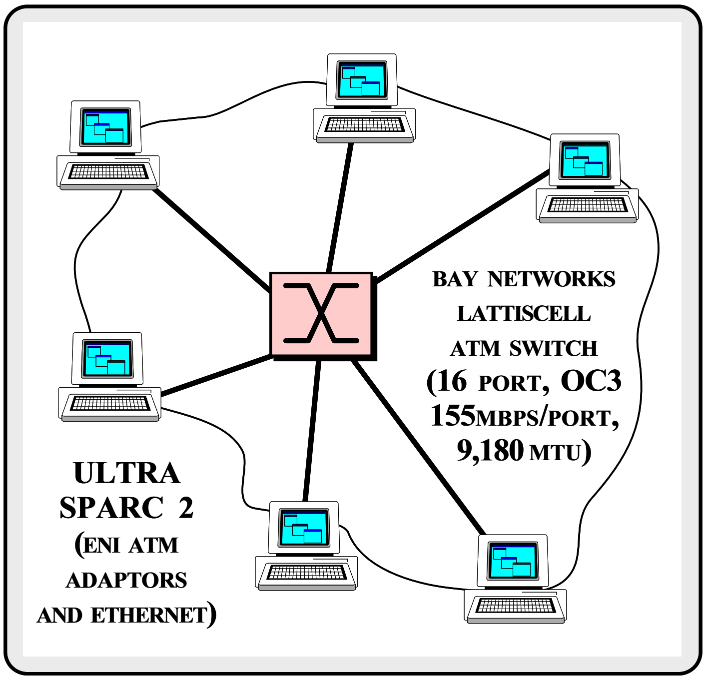
General Path of CORBA Requests



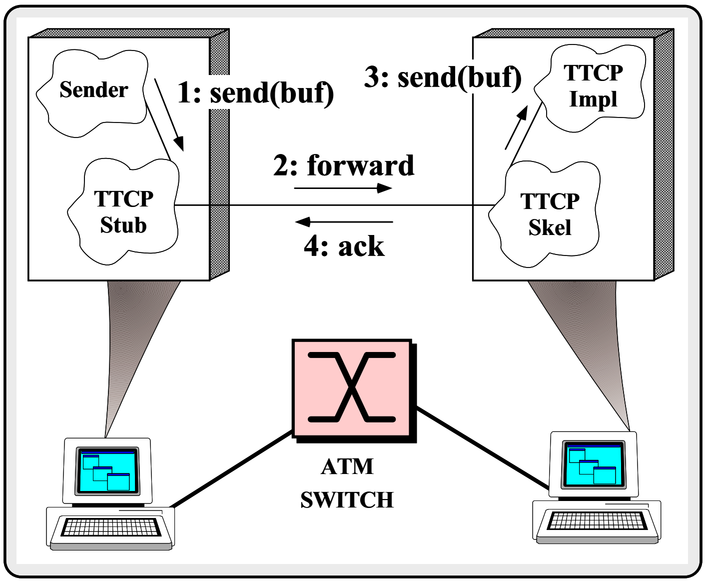
Experiments Performed

- *Throughput measurements*
  - *i.e.*, using static and dynamic invocation policies
- *Receiver-side request demultiplexing*
- *Latency measurements*
- *Scalability*
  - *i.e.*, under varying number of objects per server

Network/Host Environment



TTCP Configuration for CORBA Implementation



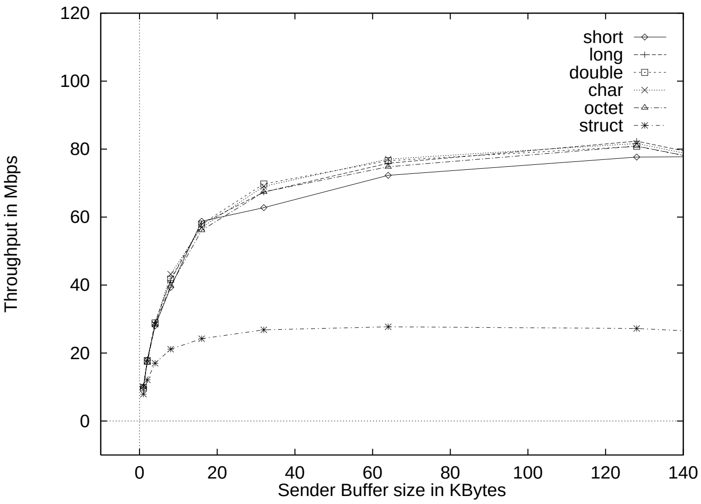
Experimental Setup for Throughput Measurements

- Enhanced version of TTCP
  - TTCP measures end-to-end data/request transfer
  - Enhanced version compares Orbix 2.1 and Visi-Broker 2.0
- Parameters varied
  - 64MB of typed data
    - ▹ Types included sequences of scalars and structs
  - Sender buffer sizes ranged from 1K to 1024K
  - Socket queues were 8k (default) and 64k (maximum)
  - Network was 155 Mbps ATM and "loopback"

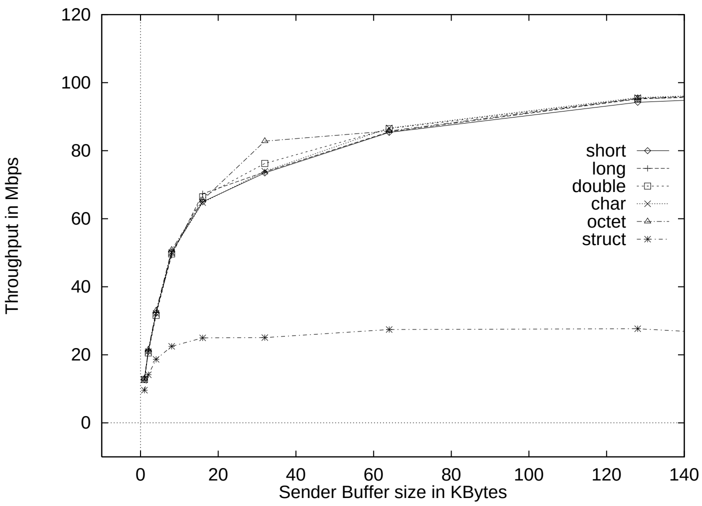
Throughput Measurements using Static Invocation Interface

- www.cs.wustl.edu/~schmidt/SIGCOMM−96.ps.gz

Orbix SII 'Blackbox' Performance



VisiBroker SII 'Blackbox' Performance

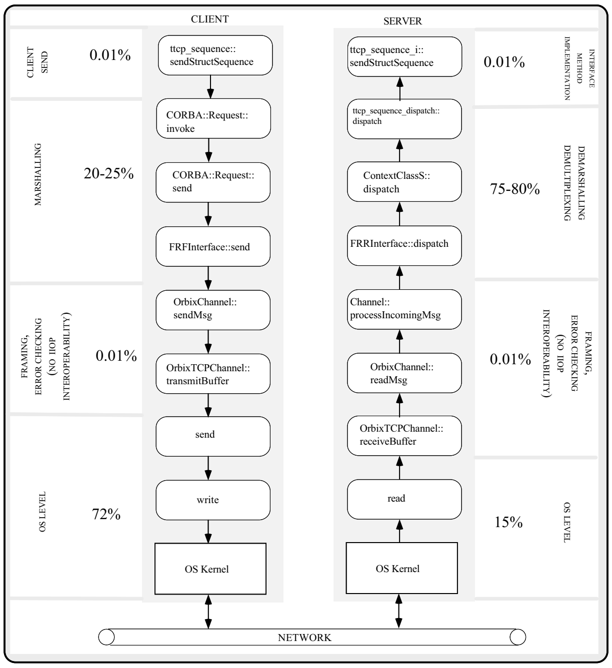


Orbix High-Cost Functions (sender side)

- Orbix sequences for 128K user buffer

Test	Time (msec)	%Exec	Name
scalars	6,091	86.00	write
	895	13.00	memcpy
struct	32,713	72.00	write
	1,177	2.58	NullCoder::codeLongArray
	978	2.15	Request::op<<(double&)
	952	2.09	BinStruct::encodeOp
	952	2.09	Request::encodeLongArray
	922	2.02	Request::insertOctet
	922	2.02	Request::op<<(short&)
	922	2.02	Request::op<<(long&)
	922	2.02	Request::op<<(char&)
	838	1.84	NullCoder::codeDouble
	755	1.66	NullCoder::codeLong
	671	1.47	memcpy

Orbix Whitebox Analysis of Throughput using SII

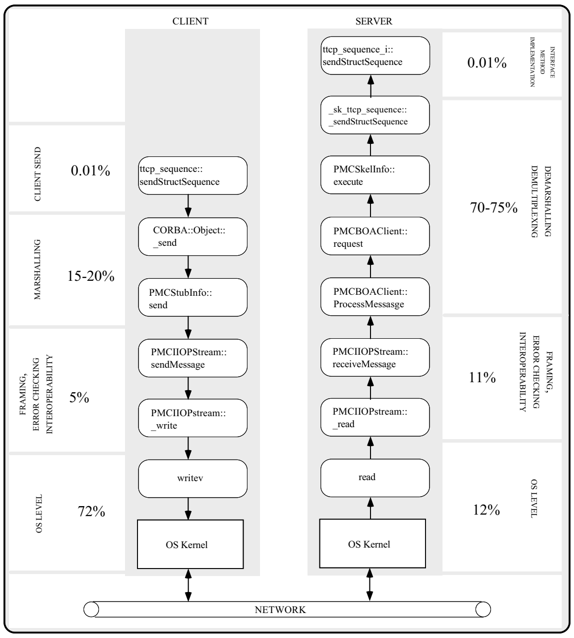


VisiBroker High-Cost Functions (sender side)

- VisiBroker sequences for 128K user buffer

Test	Time (msec)	%Exec	Name
scalars	6,214	99.00	write
struct	37,960	72.00	write
	3,831	7.28	op<<(NCostream&,BinStruct&)
	3,594	6.83	memcpy
	979	1.86	PMCIIOPStream::op<<(double)
	951	1.81	PMCIIOPStream::put
	951	1.81	PMCIIOPStream::op<<(long)
	666	1.27	PMCIIOPStream::op<<(short)

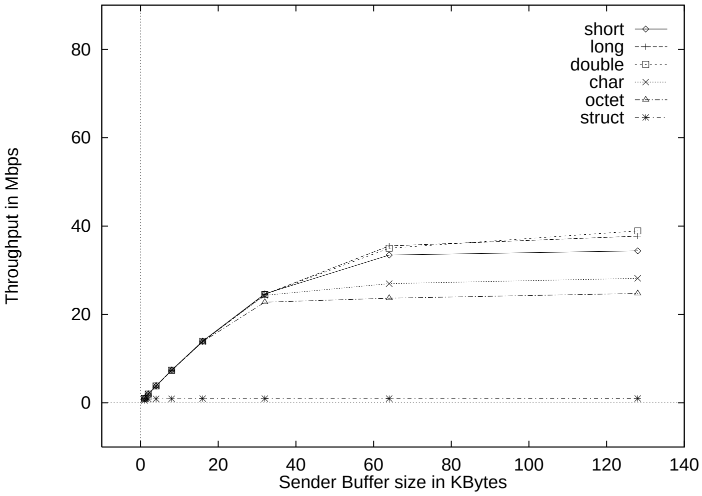
VisiBroker Whitebox Analysis of Throughput using SII



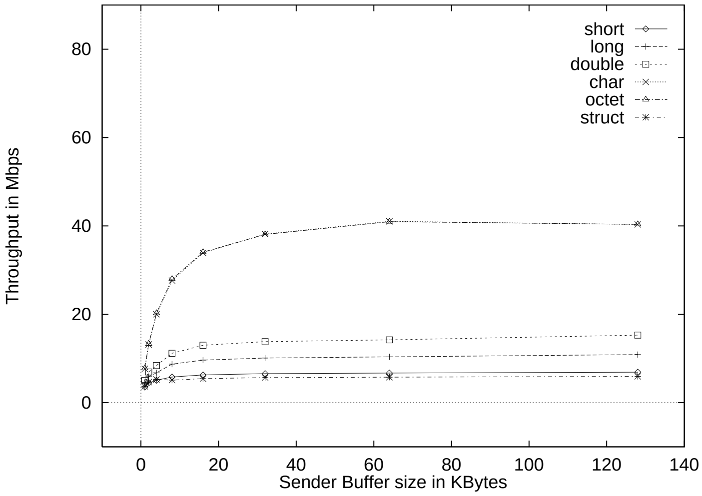
Throughput Measurements using Dynamic Invocation and Dynamic Skeleton Interface

- www.cs.wustl.edu/~schmidt/GLOBECOM−96.ps.gz

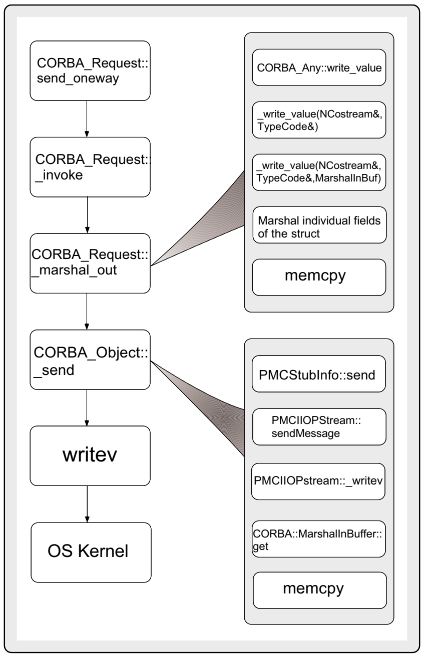
Orbix DII 'Blackbox' Performance



VisiBroker DII 'Blackbox' Performance



VisiBroker DII Client Datapath



Orbix High-Cost Functions

- Orbix sequences for 128K user buffer

Test	Time (msec)	%Exec	#Calls	Name
scalars	27,864	98.14	513	_libc_write
	352	1.24	2,560	memcpy
struct	25,740	12.38	44,085,834	cleanfree
	18,893	9.09	46,190,107	_free_unlocked
	14,328	6.89	44,092,964	realfree
	9,710	4.67	4,195,328	count
	7,522	3.62	46,190,159	malloc
	6,461	3.11	46,190,158	new
	4,855	2.34	2,097,664	CORBA::typeCode ::putValue

VisiBroker High-Cost Functions

- VisiBroker sequences for 128K user buffer

Time (msec)	%Exec	#Calls	Name
		octets/chars	
3,630	90.50	512	writev
353	8.81	8,195	memcpy
		doubles	
1,413	27.81	8,414,220	memcpy
1,229	24.21	512	writev
1,173	23.09	8,388,608	CORBA::MarshallInBuffer ::operator>>(double&)
880	17.32	8,389,632	CORBA::MarshallInBuffer ::get(char*)
352	6.93	512	CORBA::MarshallInBuffer ::get(double*)

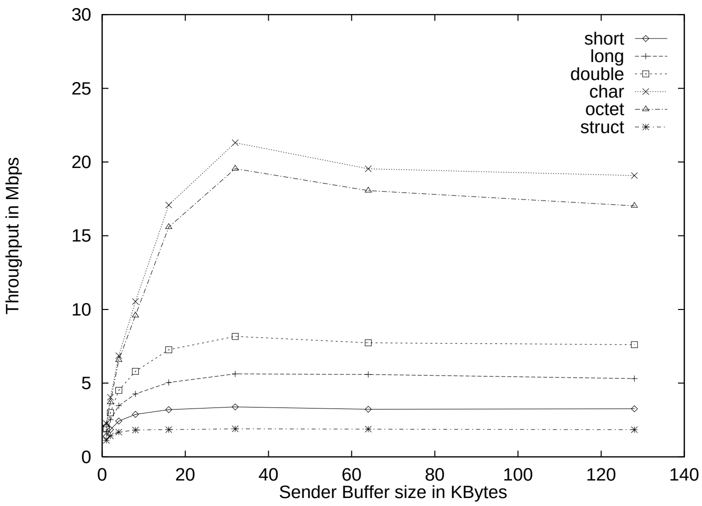
VisiBroker High-Cost Functions (cont,d.)

Time (msec)	%Exec	#Calls	Name
		longs	
2,346	29.41	16,777,216	CORBA::MarshallInBuffer ::operator>>(long&)
1,883	23.60	16,817,163	memcpy
1,760	22.06	16,777,728	CORBA::MarshallInBuffer ::get(char*)
1,249	15.65	512	writev
704	8.82	512	CORBA::MarshallInBuffer ::get(long*)
		shorts	
4,693	33.87	33,554,422	CORBA::MarshallInBuffer operator>>(short&)
3,520	25.40	33,554,944	CORBA::MarshallInBuffer ::get(char*)
2,824	20.38	33,627,659	memcpy
1,408	10.16	512	CORBA::MarshallInBuffer ::get(short*)
1,367	9.86	512	writev

VisiBroker High-Cost Functions (cont,d.)

Time (msec)	%Exec	#Calls	Name
		structs	
7,279	25.96	9,216	writev
5,200	20.00	2,796,032	various marshalling functions
4,080	14.55	39,321,438	memcpy
3,188	11.37	16,776,192	write_value(NCOstream&, CORBA:::TypeCode*, CORBA::MarshallInBuffer)
2,053	7.32	19,572,736	CORBA::MarshalInBuffer ::get(char*)
1,702	6.07	2,796,032	_write_struct_value (NCOstream, TypeCode)
1,466	5.23	13,980,160	CORBA::TypeCode:: member_type
1,408	5.02	16,776,192	CORBA::TypeCode:: member_count

VisiBroker DSI 'Blackbox' Performance



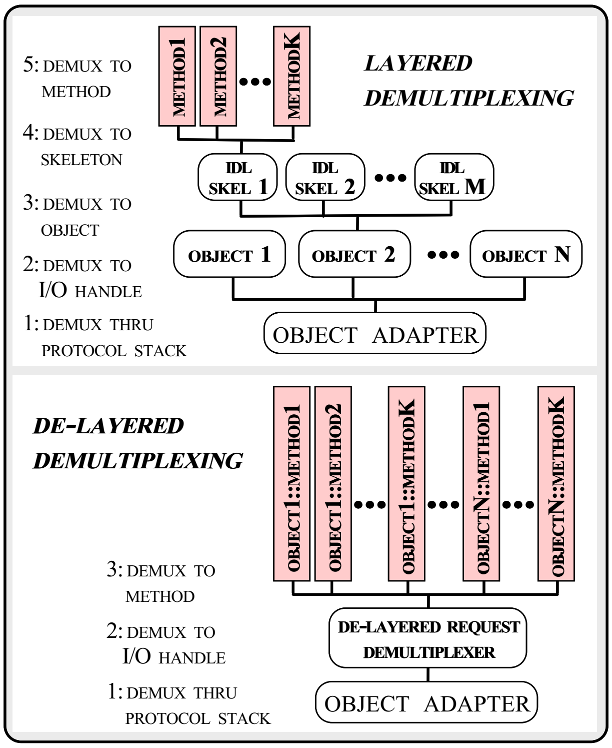
Measuring Server-side Request Demultiplexing Overhead

- An interface with N (N is large, say 100) methods defined
- Method names chosen arbitrarily
- In each iteration, client invokes the final method M times (M is large, say 100)
- Repeat the experiment 100, 500, 1,000 times

Client-side Latency in seconds

Version	Iterations 1	100	500	1,000
Orbix	0.032	5.23	31.86	64.21
VisiBroker	0.010	4.53	24.44	50.06

Demultiplexing in ORBs



Demultiplexing and Dispatching Overhead in ORBs

- Both Orbix and VisiBroker pass the method name in the request
- Passing a string adds to the amount of information carried in the request header
- Orbix uses *strcmp* and linear search on the server side to demultiplex incoming request
- For very large interfaces, this is undesirable

Hand-crafted Optimizations

- For both the ORBs, hash the method name to a numeric value and pass this information in the request header
- For Orbix, use numeric comparisons using the hashed value for method name lookup

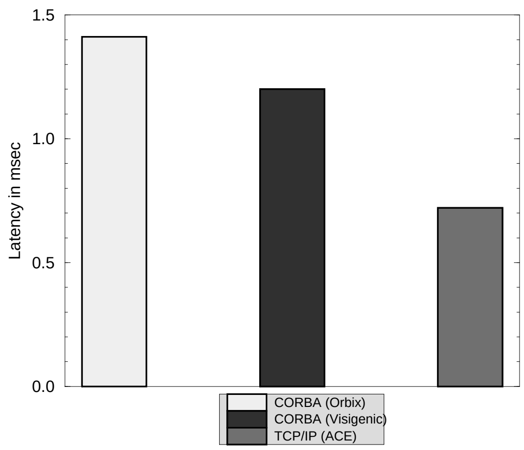
Client-side Latency in seconds

Version	Iterations 1	100	500	1,000
Orbix	0.032	5.23	31.86	64.21
Optimized Orbix	0.028	4.21	29.97	60.67
VisiBroker	0.010	4.53	24.44	50.06
Optimized VisiBroker	0.010	3.86	24.04	49.36

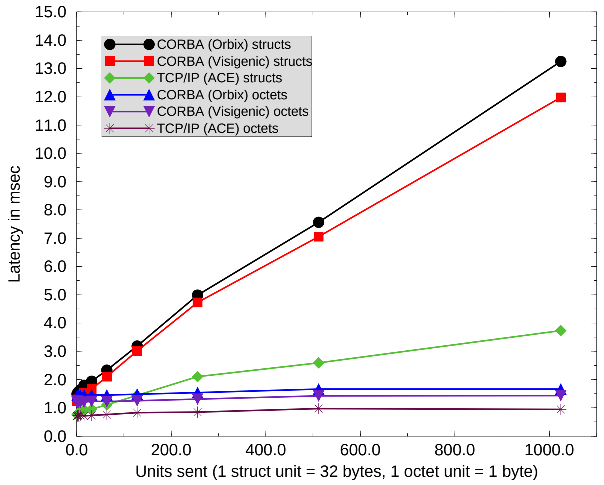
Latency Measurements over ATM

- Latency measurements for *twoway* methods
- Methods are of two types:
  1. Parameterless methods
  2. Methods that send a sequence of *octets* and *structs* of primitive data types
- Sequence parameter takes range of buffer sizes from 1 to 1,024 units of the specific data type

Latency for Parameterless Method Invocation



Latency for Methods Sending Sequences



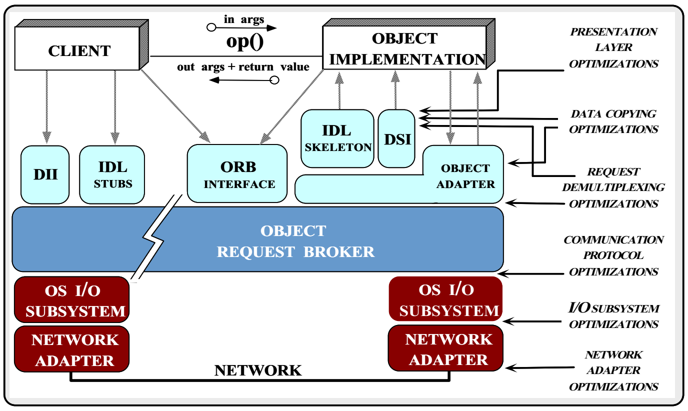
Optimizations for High Performance ORBs

- Lack of integration with advanced OS and network features:
  - Provide hooks to utilize OS features such as Real-Time scheduling, Multi-threading, and zero-copy buffer management
- Inefficient server demultiplexing techniques:
  - Use delayered demultiplexing and efficient packet filters
- Long chains of intra-ORB function calls:
  - Use Integrated Layer Processing and Inlining
  - Use compiler techniques to automate this

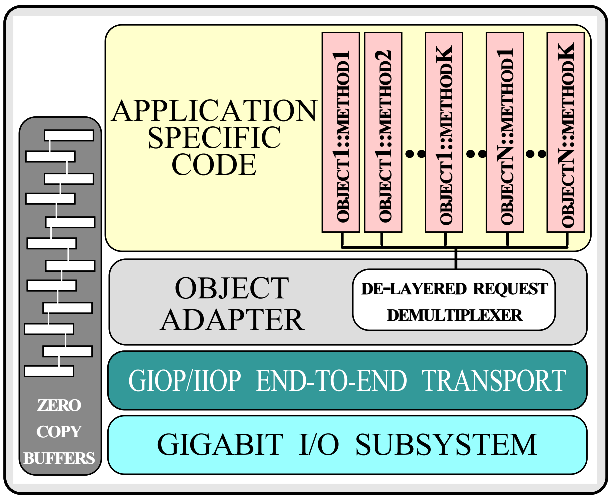
Optimizations for High Performance ORBs

- Excessive presentation layer conversions and data copying:
  - Use efficient stub compilers
  - Achieve an optimal tradeoff between compiled versus interpreted stubs
- Non-optimized buffering algorithms used for network reads and writes:
  - Use optimal buffer sizes and flow control

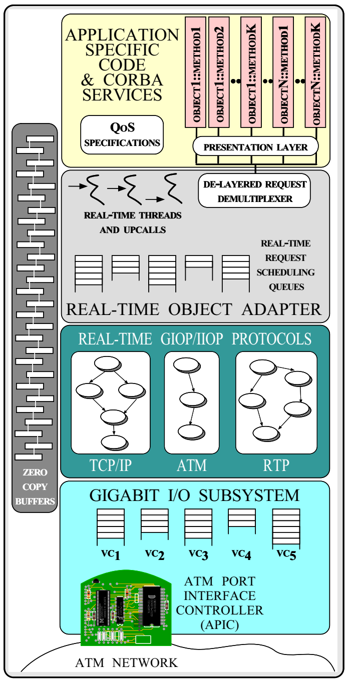
Gigabit CORBA Optimizations



Real-time CORBA



Integrated View of CORBA 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