Measuring the Performance of CORBA for High-speed Networking

Douglas C. Schmidt

schmidt@cs.wustl.edu

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

Washington University, St. Louis

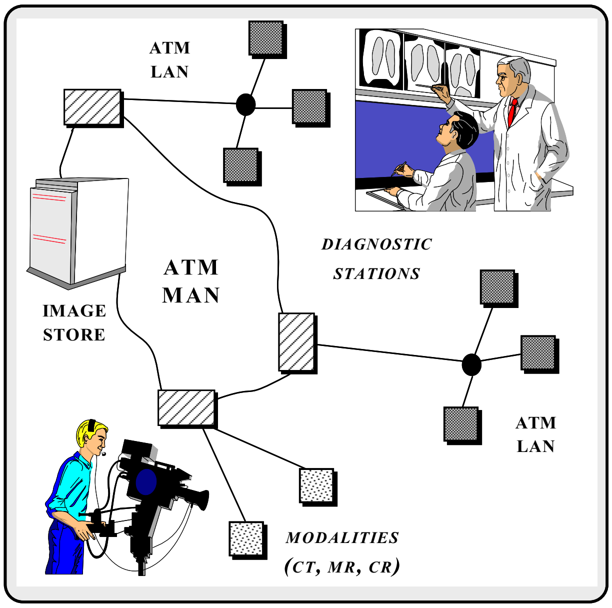
Introduction

- Distributed object computing (DOC) frameworks are 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 DOC implementations exhibit high overhead for other types of *requirements* and *environments*
  - *e.g.*, bandwidth-intensive and delay-sensitive streaming applications over high-speed ATM or FDDI

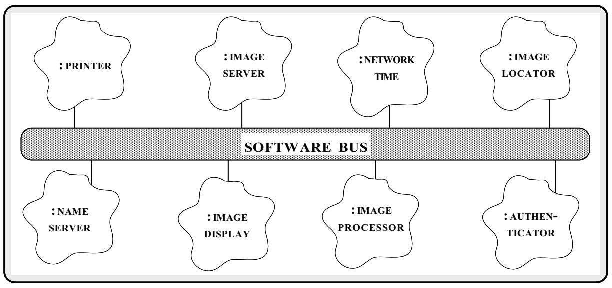
Outline of Talk

- Outline communication requirements of distributed medical imaging domain
- Compare performance of several network programming mechanisms:
  - Sockets
  - ACE C++ wrappers
  - Two CORBA implementations (ORBeline and Orbix)
- Discuss how to use distributed object computing frameworks efficiently and effectively

Distributed Medical Imaging



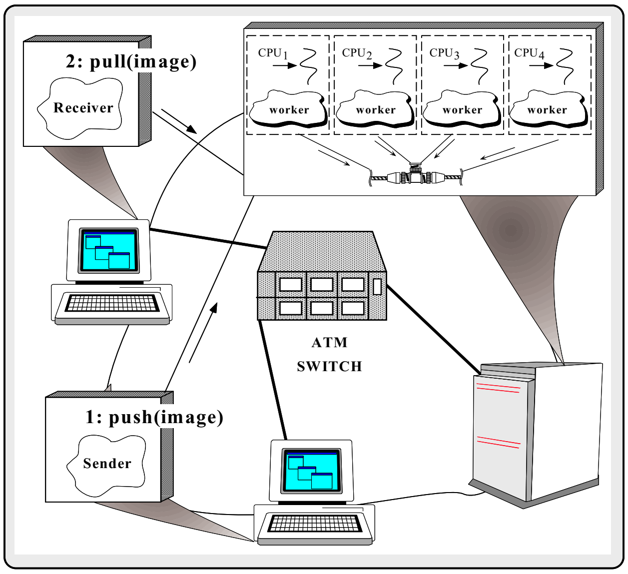
Distributed Objects in Medical Imaging Systems



- Image servers have the following responsibilities and requirements:

  * *Efficiently store/retrieve large medical images*
  * *Respond to queries from Image Locator Servers*
  * *Manage short-term and long-term image persistence*

Image Server System Architecture



Motivation for CORBA

- Simplifies application interworking
  - CORBA provides higher level integration than traditional "untyped TCP bytestreams"
- Provides a foundation for higher-level distributed object collaboration
  - *e.g.*, Windows OLE and the OMG Common Object Service Specification (COSS)
- Benefits for distributed programming similar to OO languages for non-distributed programming
  - *e.g.*, encapsulation, interface inheritance, and object-based exception handling

CORBA Overview

- CORBA specifies the following functions of an *Object Request Broker* (ORB)
  - *Interface Definition Language* (CORBA IDL)
  - *A mapping from CORBA IDL onto C, C++, and Smalltalk*
  - *An Interface Repository*
    - ▷ Contains meta-info that can be queried at run-time
  - *A Dynamic Invocation Interface*
    - ▷ Used to compose method requests at run-time
  - *A Basic Object Adaptor* (BOA)
    - ▷ Allows developers to integrate their objects with an ORB

CORBA Services

- CORBA provides the following mechanisms
  - *Parameter marshalling*
  - *Object location*
  - *Object activation*
  - *Replication and fault tolerance*
- COSS extends CORBA to provide services like
  - *Event services*
  - *Naming services*
  - *Transactions*
  - *Object lifecycle management*

Key Research Question

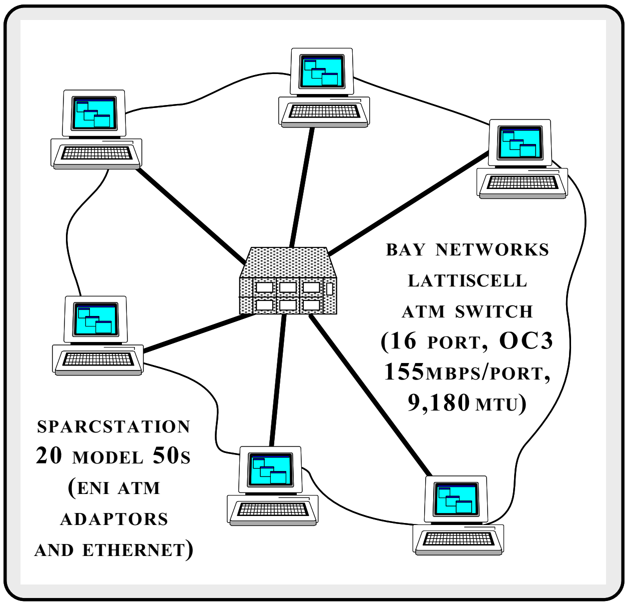
Can CORBA be used to transfer medical images efficiently over high-speed networks?

- Our goal was to determine this empirically *before* adopting the CORBA communication model wholesale

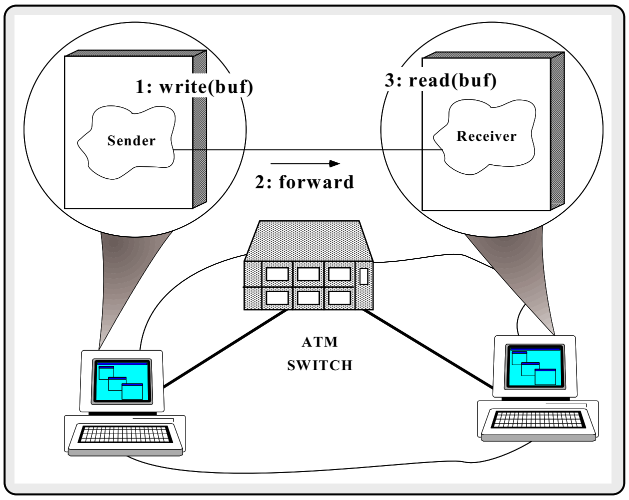
Performance Experiments

- Enhanced version of TTCP
  - TTCP measures end-to-end, oneway bulk data transfer
  - Enhanced version tests C, ACE C++ wrappers, and CORBA
- Parameters varied
  - 64 Mbytes of data buffers ranging from 1 Kbyte to 128 Kbyte (by powers of 2)
  - Socket queues were 8k (default) and 64k (maximum)
  - Networks were 155 Mbps ATM and 10 Mbps Ethernet
- Compiler was SunC++ 4.0.1 using highest optimization level

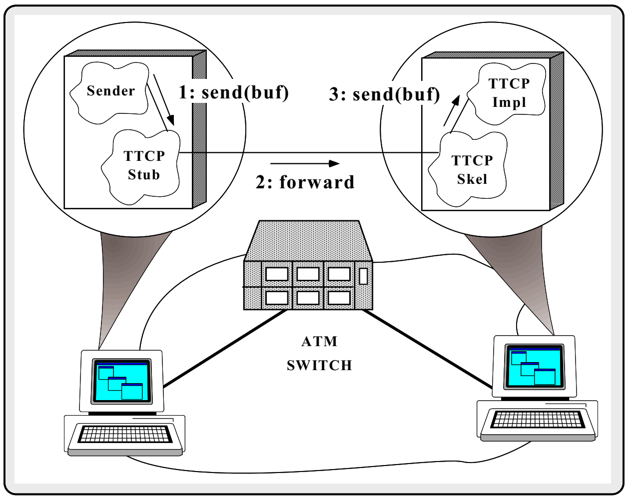
Network/Host Environment



TTCP Configuration for C and ACE C++ Wrappers



TTCP Configuration for CORBA Implementations



CORBA Implementations

- 2 implementations of TTCP using 2 versions of CORBA
  - IDL `string` and IDL `sequence`

```
typedef sequence<char> ttcp_sequence;

interface TTCP_Sequence
{
  oneway void send (in ttcp_sequence ttcp_seq);
};

interface TTCP_String
{
  oneway void send (in string ttcp_string);
};
```

  - Orbix 1.3 and ORBeline 1.2
    - ▷ Couldn't directly reuse source code since neither ORB supported same IDL → C++ mapping
    - ▷ Also, neither ORB supported CORBA 2.0

CORBA Sender Implementation

- Obtain reference to target objects via `_bind` factory:

```
// Use locator service to acquire bindings.
TTCP_String *t_str = TTCP_String::_bind ();
TTCP_Sequence *t_seq = TTCP_Sequence::_bind ();

// ...

// String transfer.

char *buffer = new char[buffer_size];
// Initialize data in char * buffer...

while (--buffers_sent >= 0)
  t_str->send (buffer);

// Sequence transfer.

ttcp_sequence sequence_buffer;
// Initialize data in TTCP_Sequence buffer...

while (--buffers_sent >= 0)
  t_seq->send (sequence_buffer);
```

CORBA Receiver Implementation

- Implementation class for IDL interface that inherits from automatically-generated CORBA skeleton class

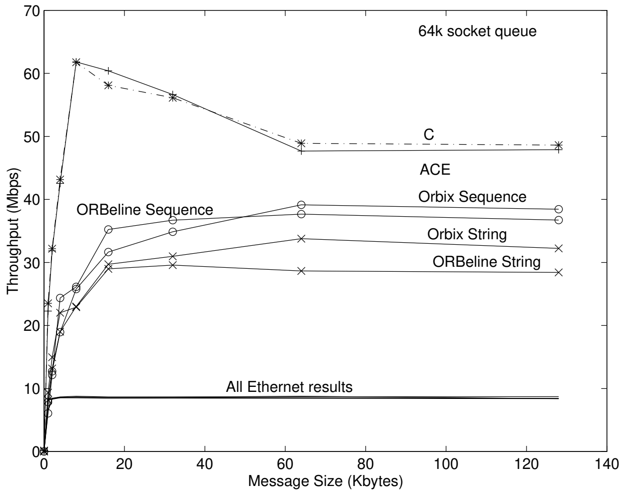
```
class TTCP_Sequence_i
  : virtual public TTCP_SequenceBOAImpl
{
public:
  TTCP_Sequence_i (void): nbytes_ (0) {}

  // Upcall invoked by the CORBA skeleton.
  virtual void send (const ttcp_sequence &ttcp_seq,
                     CORBA::Environment &IT_env)
  {
    this->nbytes_ += ttcp_seq._length;
  }
  // ...

private:
  // Keep track of bytes received.
  u_long nbytes_;
};
```


CORBA Receiver Main

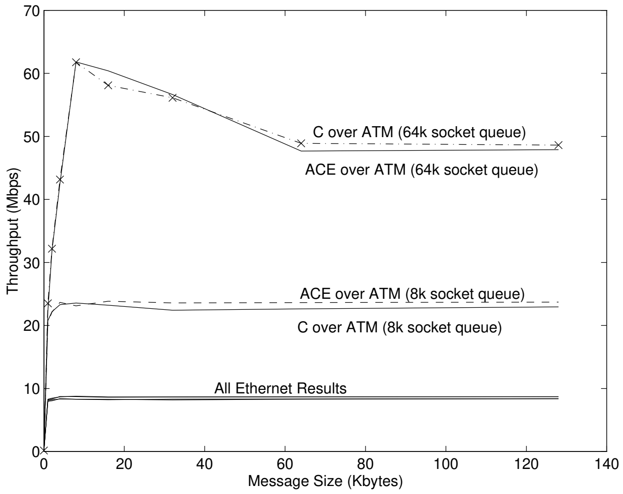
- Initializes object implementations and goes into CORBA event loop

```
int main (int argc, char *argv[])
{
  // Implements the Sequence object.
  TTCP_Sequence_i ttcp_sequence;

  // Implements the String object.
  TTCP_String_i ttcp_string;

  // Tell the ORB that the objects are active.
  CORBA::BOA::impl_is_ready ();

  /* NOTREACHED */
  return 0;
}
```

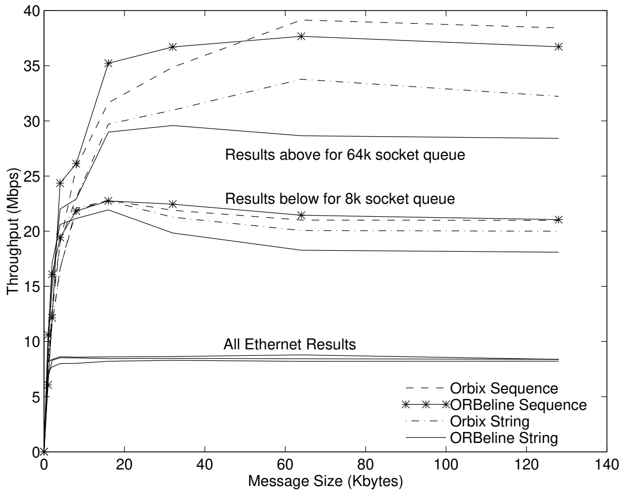

Performance over ATM and Ethernet



C and ACE Performance over ATM and Ethernet



Orbix and ORBeline Performance over ATM and Ethernet



Primary Sources of Overhead for CORBA

- *Data copying*
- *Demultiplexing*
- *Memory allocation*
- *Presentation layer formatting*

High-Cost Functions

- C and ACE C++ Tests
  - Transferring 64 Mbytes with 128 Kbyte buffers

Test	%Time	#Calls	msec/call	Name
C sockets (sender)	99.6	527	92.8	_write
C sockets (receiver)	99.3	7201	6.2	_read
ACE C++ wrapper (sender)	99.4	527	87.3	_write
ACE C++ wrapper (receiver)	99.6	7192	6.2	_read

High-Cost Functions (cont'd)

- Orbix String and Sequence Tests

Test	%Time	#Calls	msec/call	Name
Orbix Sequence	94.6	532	89.1	_write
(sender)	4.1	2121	1.0	memcpy
Orbix Sequence	92.7	7860	6.1	_read
(receiver)	4.8	2581	0.6	memcpy
Orbix String	89.0	532	85.6	_write
(sender)	4.6	2121	1.1	memcpy
	4.1	2700	0.7	strlen
Orbix String	86.3	7744	5.7	_read
(receiver)	5.5	6740	0.4	strlen
	4.5	2581	0.9	memcpy

High-Cost Functions (cont'd)

- ORBeline String and Sequence Tests

Test	%Time	#Calls	msec/call	Name
ORBeline Sequence	91.0	551	74.9	_write
(sender)	5.2	6413	0.4	memcpy
	1.8	1032	0.8	__sigaction
ORBeline Sequence	89.0	7568	5.8	_read
(receiver)	5.1	7222	0.3	memcpy
	3.3	1071	1.5	_poll
ORBeline String	83.8	551	83.9	_write
(sender)	5.4	920	3.2	strcpy
	4.3	5901	0.4	memcpy
	3.9	1728	1.2	strlen
	1.1	1032	0.6	__sigaction
ORBeline String	85.4	7827	5.5	_read
(receiver)	4.6	6710	0.3	memcpy
	4.2	1702	1.3	strlen
	2.8	1071	1.3	_poll

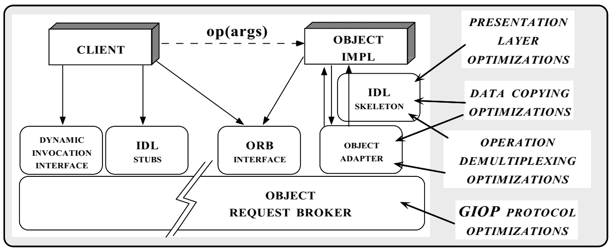
Evaluation and Recommendations

- Understand communication requirements and network/host environments
- Measure performance empirically before adopting a communication model
  - Low-speed networks often hide performance overhead
- Insist CORBA implementors provide hooks to manipulate options
  - *e.g.*, setting socket queue size with ORBeline was hard
- Increase size of socket queues to largest value supported by OS
- Tune the size of the transmitted data buffers to match MTU of the network

Evaluation and Recommendations (cont'd)

- Use IDL `sequences` rather than IDL `strings` to avoid unnecessary data access and copying
- Use `write/read` rather than `send/recv` on SVR4 platforms
- Long-term solution:
  - Optimize DOC frameworks
  - Add streaming support to CORBA specification
- Near-term solution for CORBA overhead on high-speed networks:
  - Integrate DOC frameworks with OO encapsulation of network programming interfaces

Concluding Remarks



- To be effective for use with performance-critical applications over high-speed networks, CORBA implementations must be optimized
- Key optimization points are illustrated above