CSC 7700: Scientific Computing
Module B: Networks and Data
Lecture 3: Middleware

Dr. Andrei Hutanu

Introduction

- Previous lectures
  - Basic networking, transport protocols (TCP, UDP), performance
  - Simple applications: bulk file transfer (GridFTP), videoconferencing
- This and next two lectures
  - Middleware
  - Increasingly complex network applications
    - Data Management
    - Distributed Visualization

Middleware

- What is middleware?
  - How many have heard of or understand the term?
  - Broadly – what connects applications to distributed run-time infrastructure
- We are interested applications that run on more than one machine
  - But machines may have different architectures, not necessary binary compatible

Communication

- Lowest layer: sockets
  - Socket API is not the same on each operating system: Windows, Linux
  - Application should be able to run on multiple operating systems, without needing to be coded against each API
  - Middleware provides translation from a standard communication API to the lower layer (OS-dependent) data transmission API

Data Models/formats

- How to represent data in memory
- ANSI C about int types size:
  - short int <= int <= long int
  - same for floating point data types
  - Not consistent across machines
- In some cases, communication between programming languages is also needed
- To achieve communication, the communicating parties need to use the same language
  - Rules to represent data types

Endianness

- Similar issue with endianness
  - Ordering of bytes in memory for a multy-byte datatype (for example 32 bit integer).
  - Big endian: most significant byte has the lowest address, the next significant one is at the following address, etc
  - Little endian: reverse, least significant byte has lowest address

```
int i = 1;
char* c = (char*) &i;
printf("little endian: %d\n", (*c == 1));
```

Naming

- From previous lecture: we can identify a communicating party by host name/IP and port
  - This can be insufficient (when too many entities run on the same host)
  - Also, the binding between an application component and a hostname/port is inherently dynamic
    - A component may run on a different machine each time
- Naming/mapping services are needed
  - To find a particular component
  - Translate from a “known” abstract or virtual name, to a real physical location

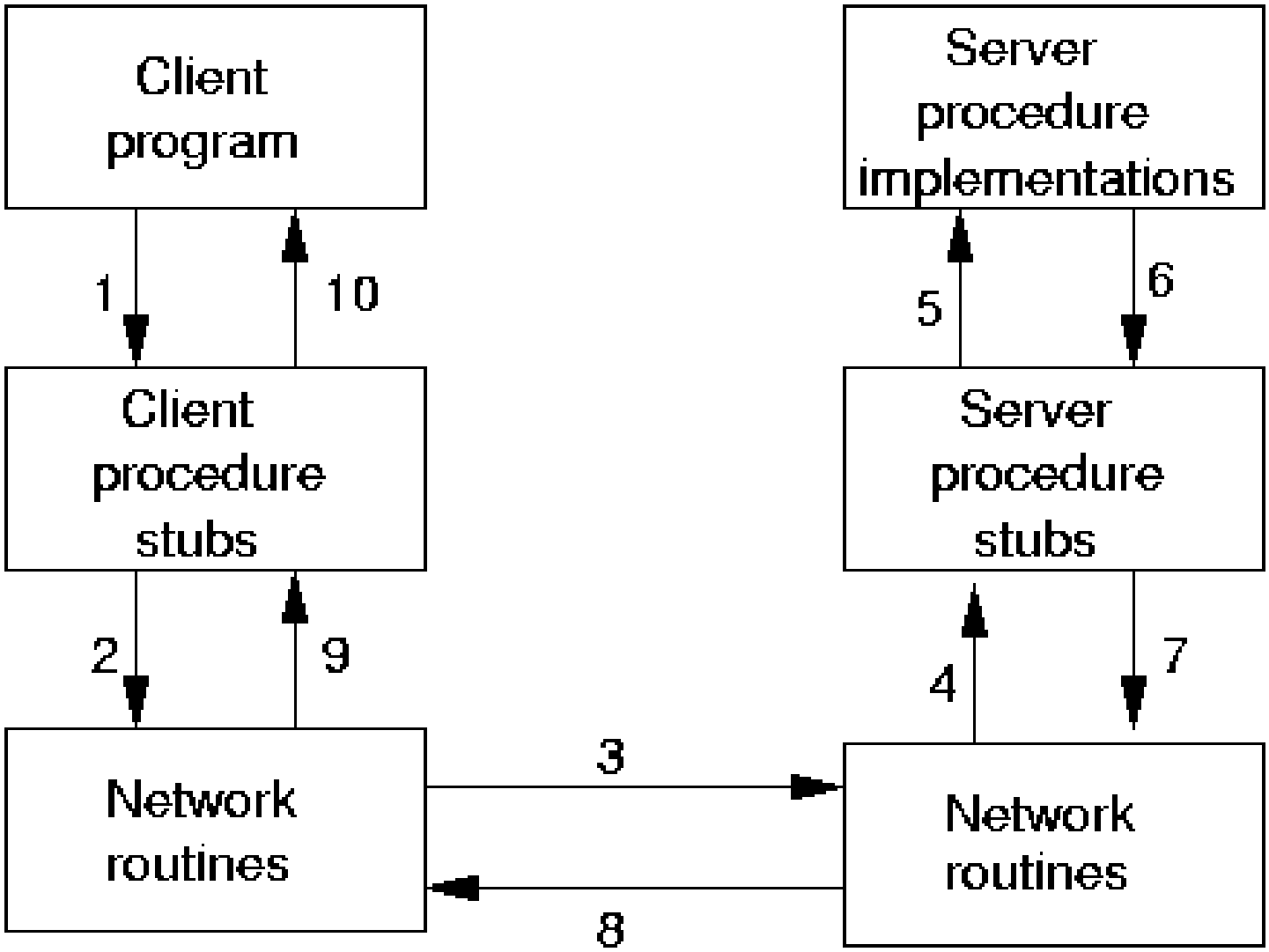
Ordinary Procedure Call

- Calling procedure pushes parameters on the stack
- Also pushes the return address
- Jumps to address of the procedure
- Procedure pops parameters, return address, does computation, pushes return value on stack, jumps to return address
- Need to implement these same operations differently for remote procedure calls

Remote Procedure Calls

- Execute a procedure in a different address space
- Usually across a network
- Workflow
  - Client specifies method name, parameter list
  - Server parses parameters
  - Server executes method
  - Serializes response and sends back to client
  - Client parses response and returns the value to the caller

RPC mechanism



RPC specifications

- Specifications provide
  - List of data types
  - Wire formats
  - IDL (Interface Description Language) – programming language independent interface description
  - Naming standard
- Implementations usually provide:
  - Client and Server stub compiler (from IDC)
  - linking facilities (link the runtime libraries into the code)
  - Container for running servers

RPC systems

- Procedural
  - Sun RPC, XML-RPC, SOAP
- Object-oriented
  - Direct support for object instances (multiple instances implementing the same interface)
  - Java RMI, CORBA

CORBA

- Starting from 1989
- Goal
  - Development of a generic framework for distributed applications
  - Integration of heterogeneous systems
- Object Management Group
  - Developer of standards (no implementation)

CORBA

- Objects communicate over the Object Request Broker (ORB)
- CORBA framework
  - Implements ORB, communication between ORBs
  - defines the Interface Definition Language (IDL)
- IDL
  - Language-independent description of an object
  - Mapped to concrete implementation languages (C++, Java)

CORBA IDL Example

- A simple one:

```
module HelloApp
{
  interface Hello
  {
  string sayHello();
  oneway void shutdown();
  };
};
```

IDL Example

```
module StockObjects {
  struct Quote {
    string symbol;
    long at_time;
    double price;
    long volume;
  };
```

(from http://java.sun.com/developer/onlineTraining/corba/)

IDL Example (cont)

```
exception Unknown{};
interface Stock {
 Quote get_quote() raises(Unknown);
 void set_quote(in Quote stock_quote);
readonly attribute string description;
};
interface StockFactory {
   Stock create_stock( in string symbol,  in string description );
};
};
```

IDL Datatypes

- Basic types:
  - short, long, long long, float, double
- Constructed types
  - struct, sequence (array – variable size), enum, array [] - fixed size
- any
  - keeping things interesting (placeholder for any data type)
- Qualifiers: in, out, inout
  - Describe how parameters are passed client/server

more IDL

- module -> namespace (C++), package (Java)
- interface -> class
- exceptions

Method invocation

- Client -> Client Stub -> Client ORB -> Server ORB -> Server Stub (Skeleton) -> Implementation
- Object references
  - Need to find an object (host system, object adapter, implementation)
  - IOR – Interoperable Object Reference
- How to get the name references
  - Naming service
  - Ask the naming service for an object with a particular name

Naming Service

- Ok, but how do we find the naming service
- Part of the CORBA bootstrap process
- Interoperable Naming Service
  - First start the naming service (can be started as a program, or as part of the ORB_Init initialization)
  - Find the naming service in your code using ORB::resolve_initial_references(“NameService”)
- If INS is not implemented, bootstrap is implementation specific

Some implementations

- Java:
  - http://java.sun.com/developer/onlineTraining/corba/
  - JacORB: http://www.jacorb.org/
- C++
  - OmniORB, MICO, TAO
- See more here:
  - http://www.yolinux.com/TUTORIALS/CORBA.html

Java RMI

- Interface extends java.rmi.Remote (public interface)
  - means it can be invoked from another Java Virtual Machine
- Methods need to be declared to throw RemoteException
- Arguments passed to remote methods or returned as return value must be basic types or implement java.io.Serializable
- public interface Hello extends Remote {

  String sayHello() throws RemoteException; }

Java RMI server implementation

- Your server class could extend UnicastRemoteObject (that has all the default values for socket transport and runs all the time, and is by default “exported” - accepts incoming connections) and needs to implement the previously defined remote interface
- public class HelloImpl extends UnicastRemoteObject implements Hello
- Other options available: java.rmi.activation.Activatable
- If not extending any of the predefined types, object must be explicitly “exported”
  - UnicastRemoteObject.exportObject

main (server)

- Load a SecurityManager

```
if (System.getSecurityManager() == null) {
    System.setSecurityManager(new
RMISecurityManager());}
```

- Create an object instance of your class
  - HelloImpl obj = new HelloImpl();
- Just as with CORBA, you will need to register the object for other programs to find it

```
Registry registry = LocateRegistry.getRegistry();
registry.rebind("my_super_object", obj);
```

Client

- Constructor must throw RemoteException
- Client:
  - Load a security manager; Look-up object in the registry
  - Invoke method

```
try {
        Registry registry = LocateRegistry.getRegistry(<hostname>);
        Hello obj = (Hello)registry.lookup(“my_super_object");
        message = obj.sayHello();
    } catch (Exception e) {
        System.err.println("Exception ");
        e.printStackTrace(); }
```

Registry

- Same issue – where is the registry?
- Need to run the registry on the machine where the RMI server is running
- rmiregistry
- The look-up name in the client code is used to contact the registry, use <hostname> on lookup
- Port is default (1099) but can be changed, by adding the port number to registration and look-up methods parameter lists

Web Services

- WSDL – Web Services Description Language – is the IDL for web services
- SOAP – Simple Object Access Protocol – wire format for web service messages
- UDDI – Universal Description, Discovery and Integration – the naming service for web services

WSDL

```
<?xml version="1.0"?>
<definitions name="StockQuote"

targetNamespace="http://example.com/stockquote.wsdl"
        xmlns:tns="http://example.com/stockquote.wsdl"
        xmlns:xsd1="http://example.com/stockquote.xsd"
        xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
        xmlns="http://schemas.xmlsoap.org/wsdl/">

    <types>
      <schema targetNamespace="http://example.com/stockquote.xsd"
            xmlns="http://www.w3.org/2000/10/XMLSchema">
          <element name="TradePriceRequest">
            <complexType>
```

WSDL

```
            <all>
                <element name="tickerSymbol" type="string"/>
            </all>
        </complexType>
    </element>
    <element name="TradePrice">
        <complexType>
            <all>
                <element name="price" type="float"/>
            </all>
        </complexType>
    </element>
  </schema>
</types>
```

WSDL

```
<message name="GetLastTradePriceInput">
    <part name="body" element="xsd1:TradePriceRequest"/>
</message>

<message name="GetLastTradePriceOutput">
    <part name="body" element="xsd1:TradePrice"/>
</message>

<portType name="StockQuotePortType">
    <operation name="GetLastTradePrice">
        <input message="tns:GetLastTradePriceInput"/>
        <output message="tns:GetLastTradePriceOutput"/>
    </operation>
</portType>
```

WSDL

```
<binding name="StockQuoteSoapBinding" type="tns:StockQuotePortType">
  <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
  <operation name="GetLastTradePrice">
    <soap:operation soapAction="http://example.com/GetLastTradePrice"/>
    <input>
      <soap:body use="literal"/>
    </input>
    <output>
      <soap:body use="literal"/>
    </output>
  </operation>
</binding>
```

WSDL

```
  <service name="StockQuoteService">
    <documentation>My first service</documentation>
    <port name="StockQuotePort" binding="tns:StockQuoteBinding">
      <soap:address location="http://example.com/stockquote"/>
    </port>
  </service>


</definitions>
```

WSDL parts

- types – data type definitions used for messages (XML Schema – or XSD)
- message – abstract definition of the data being transmitted. Messages have logical parts (elements), each associated with types. element = name + type
- portType – abstract operation. Input message and output message.
  - name is unique in the WSDL, it is the operation name
  - types- one-way, request-response, solicit-response, notification
  - one input, one output, optional fault (for two-way)

WSDL parts

- binding – specification of protocol and data format specifications for operations and messages of a portType
  - can have multiple bindings for a portType
  - SMTP binding, HTTP binding, MIME binding
- port – defines the address/communication endpoint for a binding;
- service – set of ports

SOAP

- XML-based protocol for RPC invocation
- SOAP message parts
- Envelope
  - Identifies the message as SOAP message
- Header
- Body
  - Call and response information
- Fault
  - Error and status

SOAP message structure

```
<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
<soap:Header>
...
</soap:Header>
<soap:Body>
...
  <soap:Fault>
  ...
  </soap:Fault>
</soap:Body>
</soap:Envelope>
```

More about message structure

- Envelope
  - Must use the exact xmlns:soap namespace from example
  - Defines encoding – can be defined on any element, and it is used for all children also
  - There are rules for defining encodings, just use the default, or SOAP encoding unless you really want to go in deep
- Header: Optional
- Body
  - Type names, Parameters (name, type, value)

SOAP (cont)

- Fault
  - Predefined sub-elements
- SOAP Method
  - SOAP message (XML) + HTTP
  - Destination is implicitly in the HTTP destination(host, port)
- SOAP RPC
  - Can be used without HTTP
  - Need to specify destination (URI), method name
  - In SOAP body: method invocation is a struct

SOAP (cont)

- SOAP RPC
  - the method invocation struct is named after the method name
    - Each parameter has the name the same as the parameter name and the type the same as the parameter type
  - method response is also a struct (name not important)
    - Similar as above + parameter names not important

UDDI

- Registry of web services
- Designed for public dissemination of web services
- Mostly used for dynamic placement and location of services
- As it stands, most web services usually are statically located and are heavy-weight systems
  - Not so much used for nimble dynamic distributed object placement

(Optional) Homework

- You did get a lot of homework/projects
- This is **ONLY** for those that are unhappy with their grade in the network performance report, to reduce the weight of that report in the final grade (which is – 10%, or half of the module grade; the other 10% in the final exam) to 5%
  - As far as I know only one group expressed interest
- Assignment (groups of at most 2)
  - for 5% weight (and 5% for the first report) write a simple client/server app in CORBA with a single remote (server) class that is instantiated twice (with two different names)
  - Need code + running demonstration
  - Use any CORBA&language – you will need to read the doc.

Dealing with network issues

- RPC systems make remote invocations look like local invocations
- But we can't forget that there is one important difference
  - They are not local!
- Issues to deal with
  - Network errors
  - Transport protocol performance
  - Network latency
  - Authentication/security

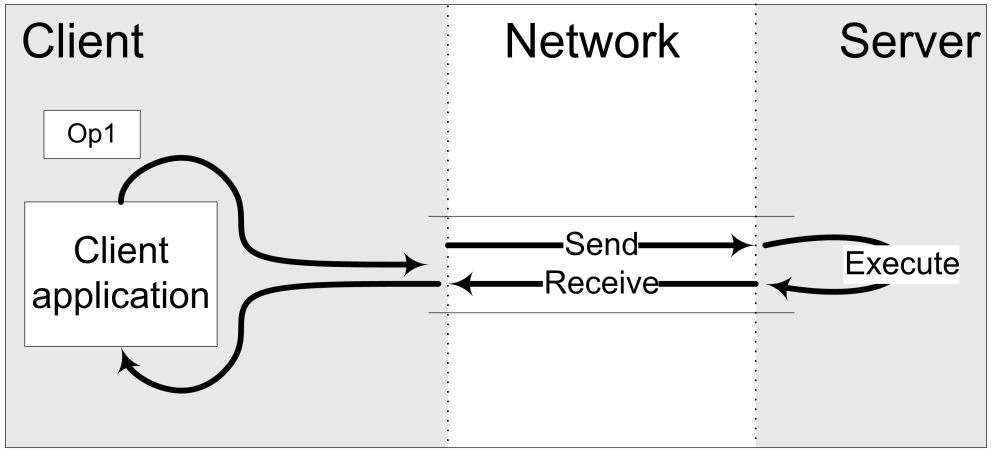
Network issues

- Remote method invocations can fail
  - No route to destination, network down
  - Data corruption if using unreliable data transmission
- Transport protocol performance
  - Discussed in previous lectures
  - Default protocol may not have good performance
  - Ideally should support a variety of protocols, esp. if large data transfers are used
- Authentication/security
  - No implicit trust as in a local invocation, need ID, permissions

Latency

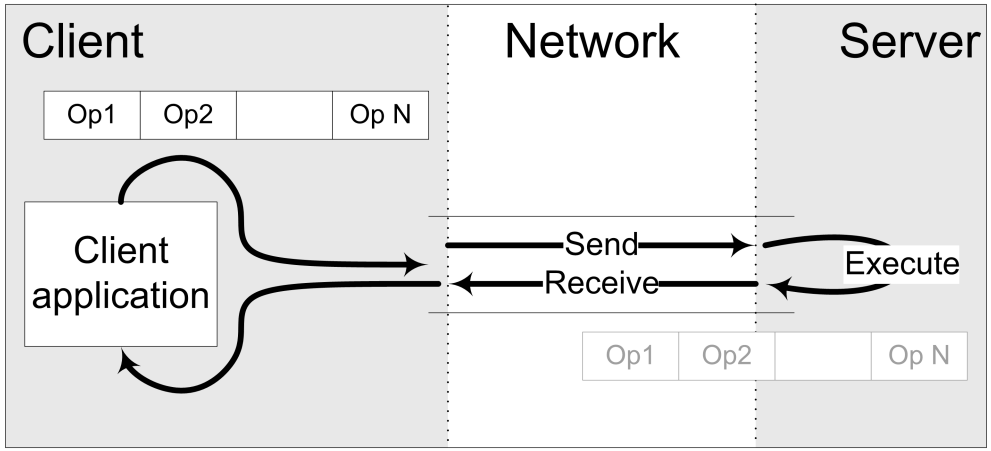
- Not a concern if the call is local
  - There's a cost associated with invoking a procedure, but most of the time it is ignored (nanoseconds)
- Network latency much higher
  - Tens of milliseconds
  - Makes frequent remote invocations impractical
- For many applications, the additional latency is not a major issue
  - Only few remote invocations, overhead insignificant compared to total running time

Synchronous invocation



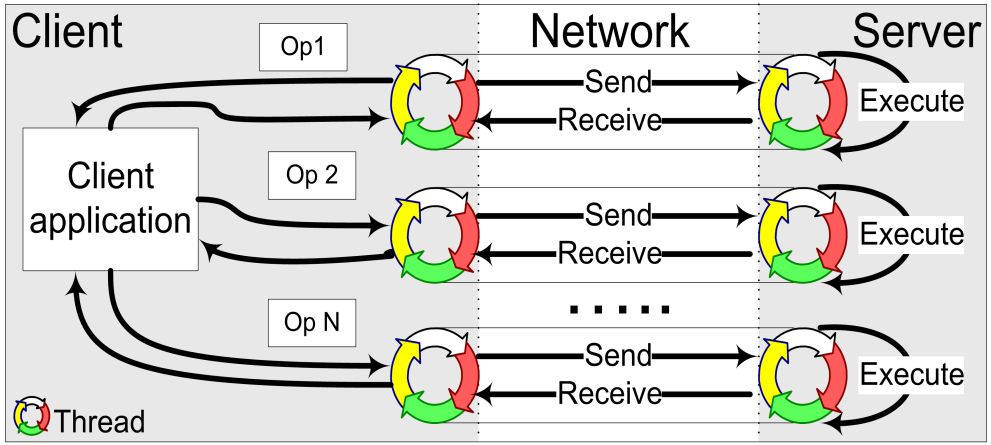
- Serial execution of all operations
- Low throughput
  - time(n operations) = n x time(one operation)
- Easy to use

Bulk/batch operations



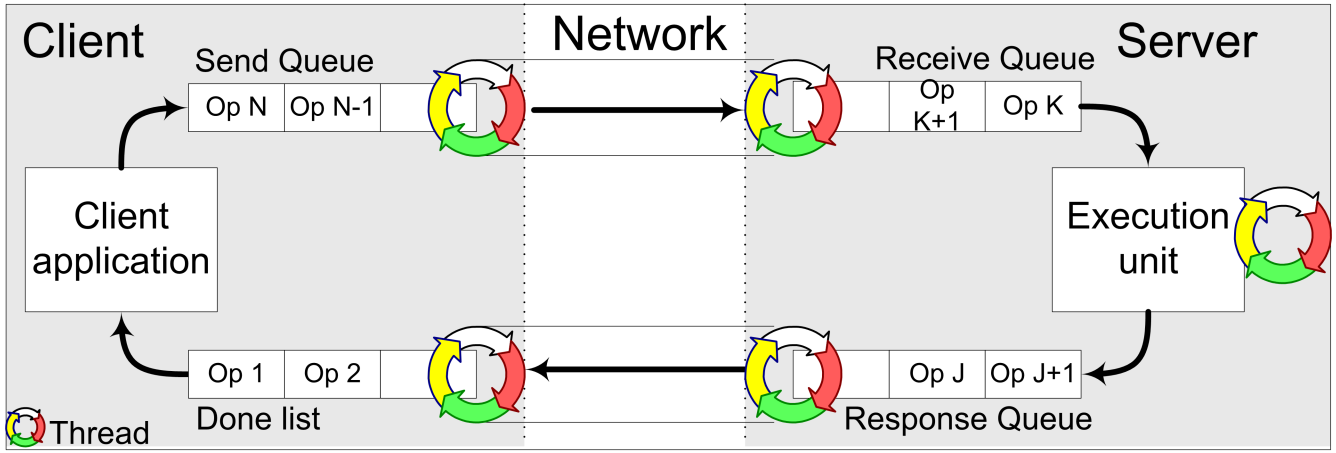
- Combine multiple operations in a single remote call
- Latency is “hidden” - accounted for once for many operations
- Need to know all operations at the same time

Threaded/Pool execution



- Threaded: Start a tread for each remote operation
- Pool: Fixed pool of threads established at set-up, queue (waiting for an available thread)
- Non-blocking, can overlap communication with computation

Pipeline execution



- Two persistent connections (commands, data)
- Network segments (send, receive), execution segment
- Complex implementation requirements

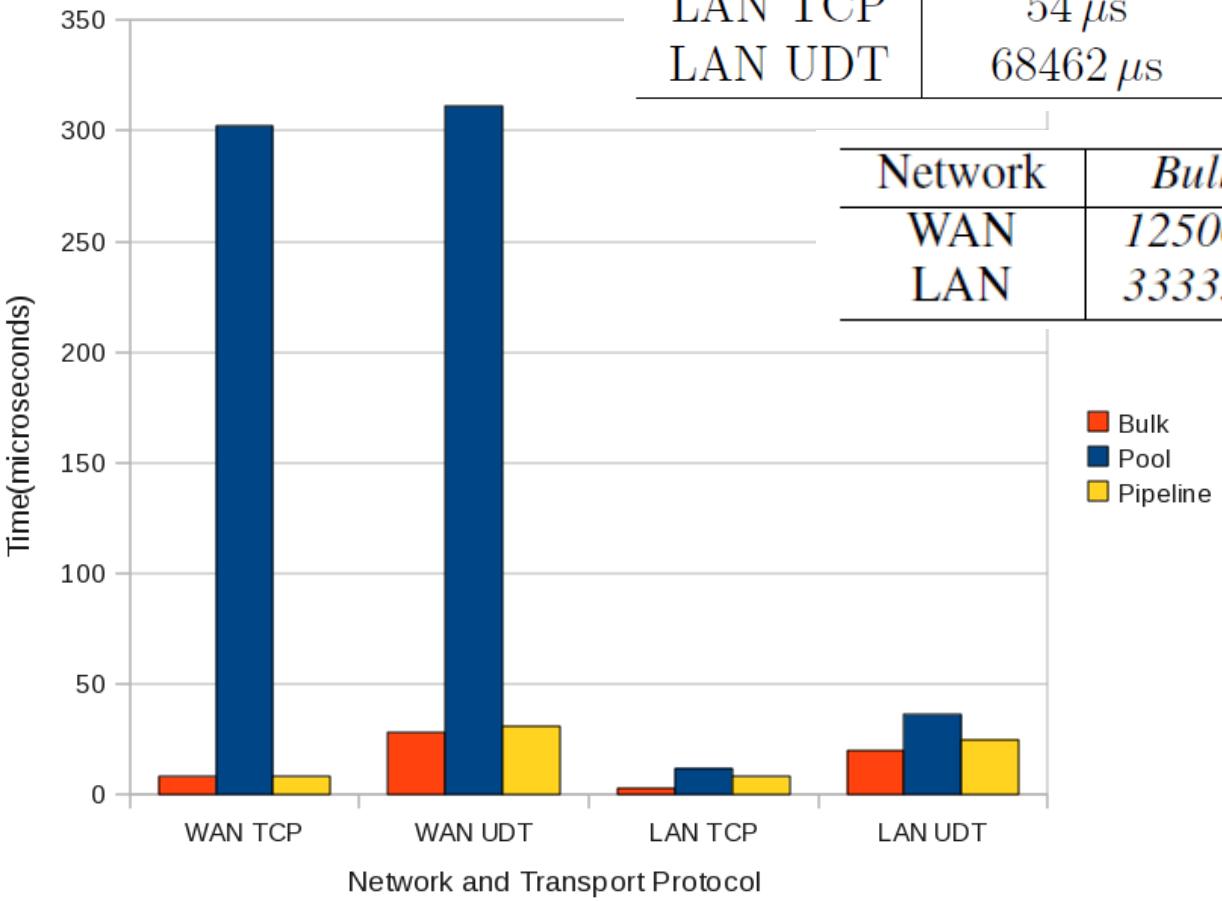
Evaluation

- How many operations can execute/second
  - Throughput
- What is the time needed to execute a single operation
  - Overhead
- Other factors
  - Usability (blocking/non-blocking), implementation effort
- As with most things, expect some trade-off

Benchmarks: Throughput

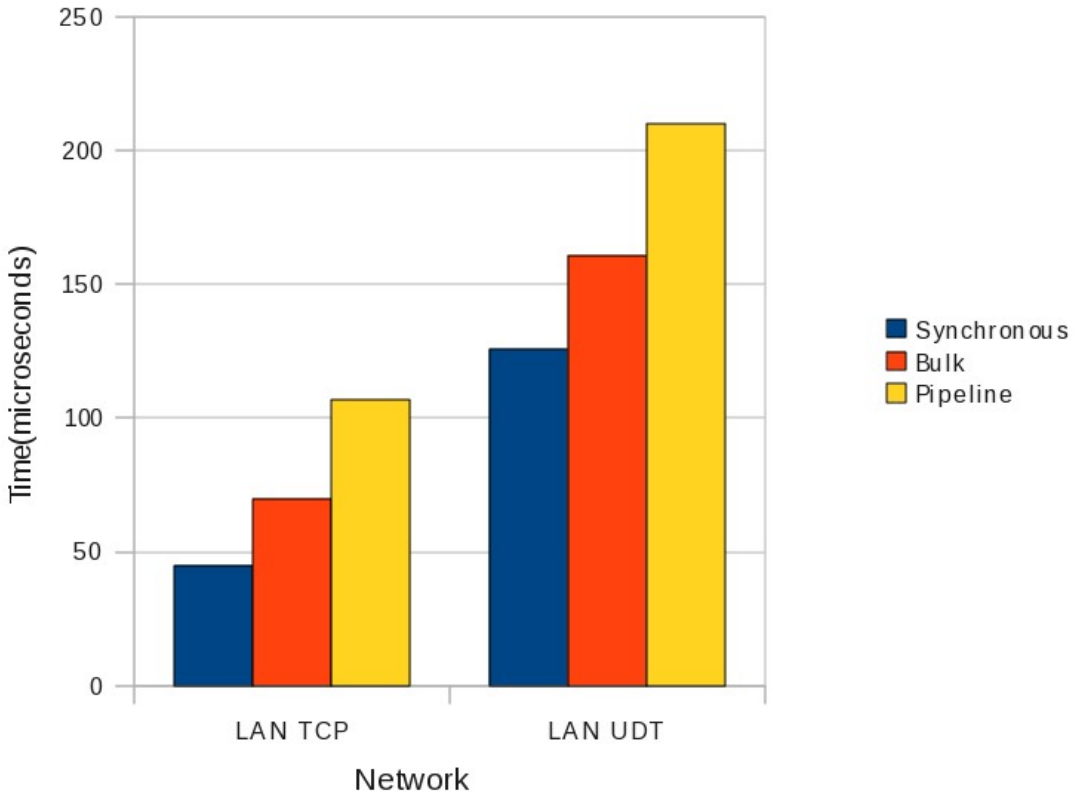
Network	Synchronous	*Bulk*	Threaded	Pool	*Pipeline*
WAN TCP	151040 μs	*8 μs*	3275 μs	302 μs	*8 μs*
WAN UDT	151103 μs	*28 μs*	8962 μs	311 μs	*31 μs*
LAN TCP	54 μs	*3 μs*	108 μs	12 μs	*8 μs*
LAN UDT	68462 μs	*20 μs*	2687 μs	36 μs	*25 μs*

Network	*Bulk*	Thread	Pool	*Pipe*
WAN	*125000*	305	3311	*125000*
LAN	*333333*	9259	83333	*125000*



Benchmarks: Overhead

Network	Synchronous	Bulk	Threaded	Pool	Pipeline
WAN TCP	151040 μs	151064 μs	302488 μs	151257 μs	151084 μs
WAN UDT	151059 μs	151854 μs	492458 μs	151344 μs	151120 μs
LAN TCP	*45 μs*	*70 μs*	199 μs	449 μs	*107 μs*
LAN UDT	*126 μs*	*161 μs*	58603 μs	3904 μs	*210 μs*



End-to-end arguments in system design

- Paper linked from wiki page of this module
- In a system that includes network communication, can draw a line between communication and the rest of the system
  - For a particular function, it can be implemented in several ways
    - By the communication system
    - By its client
    - By both, in cooperation, or redundantly

- *“The function in question can completely and correctly be implemented only with the knowledge and help of the application standing at the end points of the communication system. Therefore, providing that questioned function as a feature of the communication system itself is not possible. (Sometimes an incomplete version of the function provided by the communication system may be useful as a performance enhancement.)”*
- this is the “end-to-end argument”

Examples

- File copy across networks
- Possible sources of errors
  - Source file system
  - Data transmission software
  - Network cards
  - Data transmission networks
  - Hosts may crash
- Taking care of carefully reducing the possibility of error at each step will reduce the error rate
  - But with great effort, and not providing total certainty

Discussion

- Alternative
  - Verify checksum after file transfer complete
  - Retry everything if it failed
- Does network transmission reliability guarantee success?
  - It eliminates one source of error
  - Not if the goal is reliable end-to-end file transfer
  - App still needs to do the work
- But still, some reliability can help
  - Don't retransmit the whole file, just lost packets

Discussion

- That retransmission does not need to happen at the lowest layer though
  - App can do it also
- Careful decision about where to put functionality
- Don't want app to be stuck with costly “feature” it doesn't need
- More examples
  - Reliability for video transmission increases latency
  - Congestion control on dedicated links reduces bandwidth utilization. More examples in the paper