Network Programming with Boost

Jeff Garland

jeff@crystalclearsoftware.com

Tutorial Goals

- Describe the basics of writing networked programs
- Writing network enabled programs including:
  - understanding of some of the issues in network programming
  - an understanding of 'asynchronous programming'
  - basics of TCP based sockets
  - integration of sockets and i/o streams
  - understanding of using timers in network programming
- Using serialization with asio to build a general object messaging frameworks

Topics Covered

- Network programming
  - Terminology and theory
  - Tools
- Boost Libraries
  - asio
  - serialization
  - bind
  - regex
  - smart_ptr
  - date-time

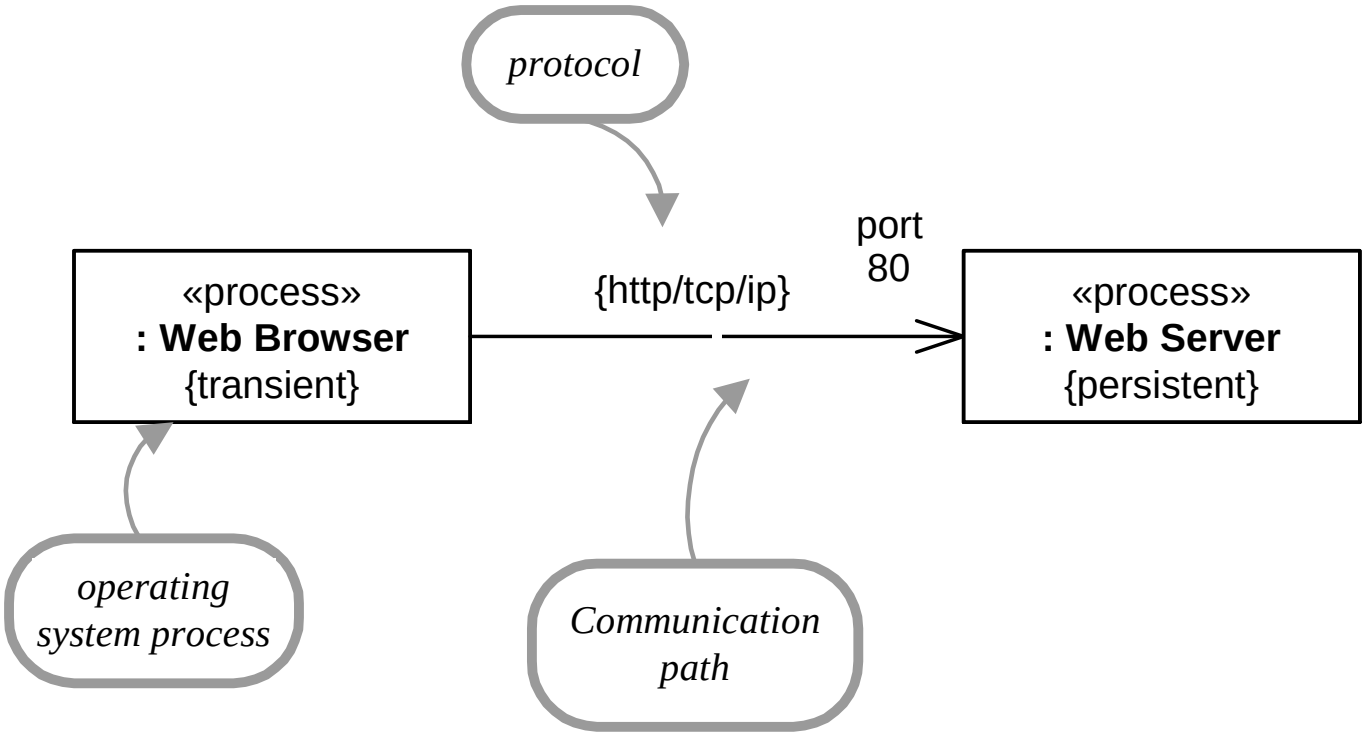
Session Agenda

- Introduction
- Boost.asio Overview
- Basic Synchronous Client Program
- Boost.asio in depth
- Asynchronous Server Program
- Boost.serialization – short overview
- Pitfalls of Network Programming
- Network Programming Tools
- Resources and Conclusion

What's the need?

- Many modern applications are now distributed applications
- Networking Advancements
  - Ubiquitous networks - opens possibility of making more apps networked
  - 'Fast' networks
    - Gigabit ethernet
    - Wireless N
- Protocol Development
  - Continued advancements in 'application protocols' (example RSS)

Basic Client Server Communication

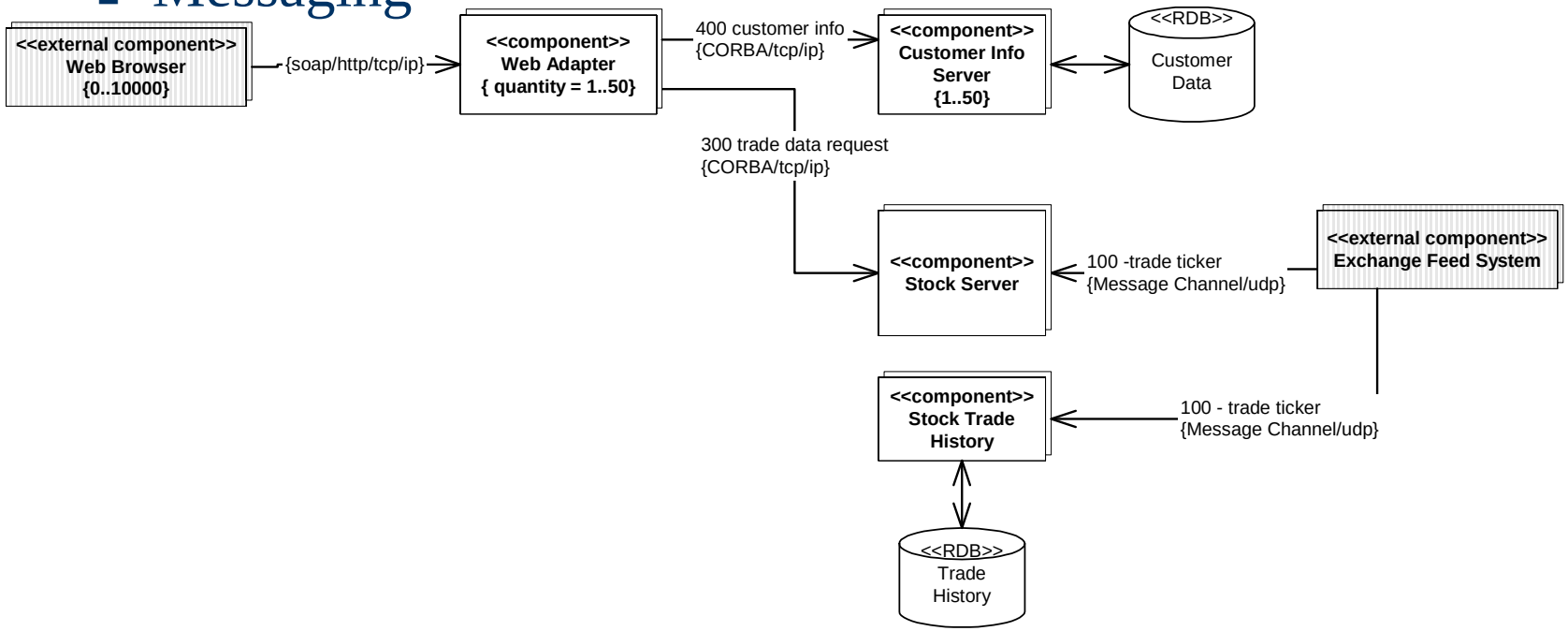


Networking Terminology

- **Client** – Process that accesses a remote service.
- **Server** – Process that provides a service for remote computers.
- **Runtime Component** - a construct that groups runtime entities and provides/consumes a set of interfaces.
- **Process** - an instance of an operating system process

Types of Network Communication - Example

- Web page to display current price of stock
  - Client – server type
- Exchange feeds
  - Messaging



Types of Network Communications

- Service
  - Client/Server
  - RPC
  - Client connects to server, makes a request, optionally receives response
  - Suitable for larger data transfers
  - Point to Point
- Messaging
  - Publish subscribe, ‘Multi-cast’

Networking Concepts

- Connectionless Protocol
  - UDP/IP
    - UDP – User Datagram Protocol
    - IP – Internet Protocol
- Connection-oriented Protocol
  - TCP/IP
    - TCP - Transmission Control Protocol
  - SCTP/IP
    - SCTP – Stream Control Transmission Protocol

IP – Internet Protocol

- Base addressing scheme
  - Used for almost all communications now
- IPv4
  - Typical address: 192.168.1.100
  - 32 bits
- IPv6
  - Typical address: 3ffe:0501:0008:0000:0260:97ff:fe40:efab
  - 128 bits
- Special addresses
  - Loopback 127.0.0.1 – localhost
  - 192.xxx.xxx.xxx

Session Agenda

- Introduction
- Boost.asio Overview
- Basic Synchronous Client Program
- Boost.asio in depth
- Asynchronous Server Program
- Boost.serialization – short overview
- Pitfalls of Network Programming
- Network Programming Tools
- Resources and Conclusion

asio 101 – What is asio?

- Boost cross-platform system OS access library
  - Networking is just one aspect
  - Timers
  - May eventually support file access and others
- What does asio mean?
  - Asychronous Input-Output…nope
  - Actually - play on words Australian Security Intelligence Organisation
- Pronounced - ay-zee-oh
- Library written by Christopher M. Kohlhoff

asio 101 – history / info

- Started outside of Boost
  - Was quite successful as a standalone project
- Reviewed and accepted into Boost Dec/Jan 2005
- In Boost 1.35 – just released

asio 101 – Design Goals

- Efficient - Allow for minimal data copying
- Scalable – Up to thousands of concurrent connections
- Integrate well with standard library
  - streambuf integration
  - Foundation for TR2 Networking Proposal
    - http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2006/n2054.pdf
- Easy to learn
  - Network concepts based on Berkley sockets
- Portable

asio 101 - Platforms

- Win32 with VC++ 7.1 and 8.x
- Win32 with Borland C++ builder
- Win32 with MinGW or Cygwin / g++
- Linux with g++ 3.3 or later
- Solaris with g++3.3 or later
- Mac OS X 10.4 using g++ 3.3 or later
- FreeBSD with g++3.3 or later
- QNX Neutrino 6.3 with g++ 3.3. or later

asio 101 – lib dependencies

- Other boost library dependencies
  - ‘system’ used for integrated error handling
  - date_time used for time specifications
    - Technically optional

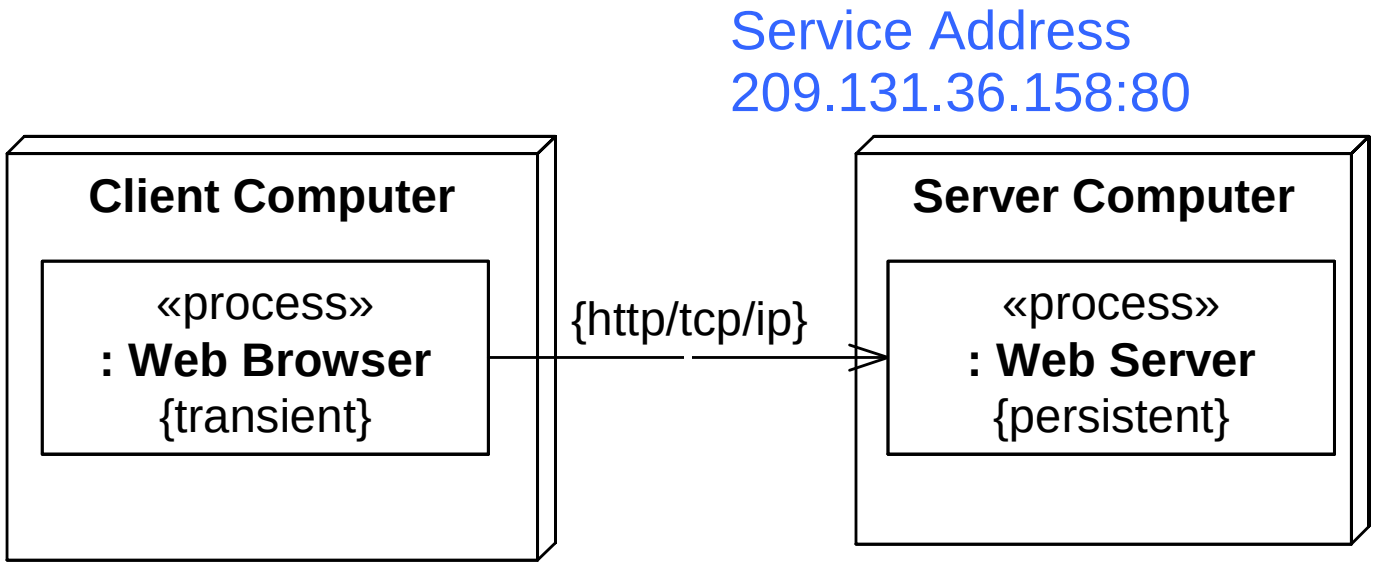
asio 101 - header Only?

- Review version of asio was header-only
- Newer versions depend on boost.system
  - Compiled library
- List discussion of merits

Session Agenda

- Introduction
- Boost.asio Overview
- Basic Synchronous Client Program
- Boost.asio in depth
- Asynchronous Server Program
- Boost.serialization – short overview
- Pitfalls of Network Programming
- Network Programming Tools
- Resources and Conclusion

Fetch a web page



Example: fetch a web page

- Simple form of ‘wget’
- Steps:
  - Step 1: Includes and setup stuff
  - Step 2: Get the address of the server
  - Step 3: Make a connection to server
  - Step 4: Write http request to server
  - Step 5: Read resulting data
    - 5a – read the header return
    - 5b – process the data

Functions needed in Networked Apps

- Find address of service
- If connection oriented:
  - Connect to service, or wait for connections
  - Reconnect on failures
- Deal with Data
  - Format request or response data to transmit on wire
  - Protocol support
  - Buffering
- Handle Failures
- Multiplexing and Timing
- Setup connection and other options

Fetch Web Page 1: Setup

```
#include <boost/asio.hpp>
#include <boost/regex.hpp> //not needed in latest
#include <boost/system/system_error.hpp> //latest
    only
#include <string>
#include <iostream>
using namespace boost::asio;
using namespace boost::asio::ip;
using std::cout;
using std::endl;
using std::string;
```

Fetch Web Page 1:Setup - Main

```
int main (int argc, char* argv[])  {
  try
 {
    boost::asio::io_service io_service;
    //To Be expanded
  }
  catch (std::exception& e)
  {
     std::cout << "Exception: " << e.what() << std::endl;
  }

   return 0;
}
```

Fetch Web Page 2: Get Address

```
boost::asio::io_service io_service;

//eg: www.boost.org
string host_name = "localhost";

tcp::resolver resolver(io_service);
tcp::resolver::query query(host_name, "http");
tcp::resolver::iterator endpoint_iterator = resolver.resolve(query);
tcp::resolver::iterator end;

// To be expanded
while (endpoint_iterator != end)
```

Fetch Web Page 3: Connect Socket

```
tcp::socket socket(io_service);

//connect error is exception type!
//boost::asio::error connect_error = boost::asio::error::host_not_found;
boost::system::error_code connect_error = boost::asio::error::host_not_found;
while (connect_error && endpoint_iterator != end)
{
    socket.close();
    socket.connect(*endpoint_iterator++, connect_error);
}

if (connect_error)
    throw connect_error;
//To Be expanded
```

Fetch Web Page 4: Write Fetch Request

```
string page_path = "/index.html";

boost::asio::streambuf request;
std::ostream request_stream(&request);
//http protocol stuff…
request_stream << "GET " << page_path << " HTTP/1.0\r\n";
request_stream << "Host: " << host_name << "\r\n";
request_stream << "Accept: */*\r\n";
request_stream << "Connection: close\r\n\r\n";

// Send the request.
boost::asio::write(socket, request);

//To Be expanded
```

Fetch Web Page 5: Receive Data – Check Header

Read the response status line.

```
boost::asio::streambuf response;
boost::asio::read_until(socket, response, "\r\n");
//boost::asio::read_until(socket, response, boost::regex("\r\n"));  //v 0.3.7

// Check that response is OK.
std::istream response_stream(&response);
std::string http_version;
response_stream >> http_version;
unsigned int status_code;
response_stream >> status_code;
std::string status_message;
std::getline(response_stream, status_message);
```

Fetch Web Page 5: Receive Data – Check Header

```
//….continued…
if (!response_stream || http_version.substr(0, 5) != "HTTP/")
{
    std::cout << "Invalid response\n";
    return 1;
}
if (status_code != 200)
{
    std::cout << "Response returned with status code " << status_code << "\n";
    return 1;
}

//To Be expanded
```

Fetch Web Page 6: Receive Data

```
// Read the response headers, which are terminated by a blank line.
//boost::asio::read_until(socket, response, boost::regex("\r\n\r\n")); //v 0.3.7
boost::asio::read_until(socket, response, "\r\n\r\n");


// Process the response headers.
string header;
cout << "**** Header ******" << endl;
while (std::getline(response_stream, header) && header != "\r")
{
    cout << header << "\n";
}
cout << "\n";
```

Fetch web Page 6: Receive Data

```
cout << "***** Part 1 ******" << endl;
// Write whatever content we already have to output.
if (response.size() > 0)
      cout << &response;

cout << "***** Rest ******" << endl;
boost::system::error read_error;
// Read until EOF, writing data to output as we go.
while (boost::asio::read(socket,
                                response,
                                boost::asio::transfer_at_least(1),
                                read_error)) {
                               // boost::asio::assign_error(read_error))) {
    cout << &response;
}
if (read_error != boost::asio::error::eof)
  throw read_error;
```

Session Agenda

- Introduction
- Boost.asio Overview
- Basic Synchronous Client Program
- Boost.asio in depth
- Asynchronous Server Program
- Boost.serialization – short overview
- Pitfalls of Network Programming
- Network Programming Tools
- Resources and Conclusion

asio in depth - namespace structure

- Base Namespace: boost::asio
  - Core concept templates
    - basic_socket, basic_deadline_timer, etc
  - Contains io_service, buffer, others
  - Various free functions (read/write)
- boost::asio::ip - concrete ip related implementations
  - Defines core protocol level classes
    - tcp – for tcp related capabilities
      - acceptor, endpoint, iostream, resolver, socket
    - udp – for udp related capabilities
      - endpoint, socket, resolver
  - Sub namespace for multicast and unicast options
  - Some concept templates as well -- basic_endpoint (moved since v0.3.7)
- boost::asio::ssl – Secure sockets
  - context, context_base, stream_base (etc)
- boost::asio::placeholders – bind placeholders
  - error, bytes_transferred, iterator

asio in depth – sync read

- Synchronous read functions
  - Attempt to read a certain amount of data from a stream before returning.
  - Many variants:
    - read
    - read_until

```
template<typename Sync_Read_Stream,
         typename Mutable_Buffers>
std::size_t  boost::asio::read (Sync_Read_Stream &s,
                                const Mutable_Buffers &buffers)
```

asio in depth – sync write

- Write all of the supplied data to a stream before returning.
- Several variants with different buffer types

```
template<typename Sync_Write_Stream,
         typename Const_Buffers>
std::size_t  boost::asio::write (Sync_Write_Stream &s,
                                 const Const_Buffers &buffers)
```

Asio : ip address

- Class that represents a v4 or v6 ip address
- Example: classic v4 address:
  - 192.168.100.1
- Couple variants if you know the ip version
  - address_v4
  - address_v6

Key ip_address functions

Method	Parameters	Return	Description
address			Default constructor.
to_string		std::string	Convert to string.
from_string	const char*	address	Construct address from a string.
operator== operator!= operator<	Address	bool	Less than comparable functions

- Several other variants of from_string
- To_v4 and to_v6 if you need specific address type

Other functions: Get our Hostname

```
#include <boost/asio.hpp>
#include <iostream>
using namespace boost::asio;
int main()
{
  try
  {
    std::string host_name = ip::host_name();
    std::cout << "hostname: " << host_name << std::endl;
  }
  catch (std::exception&)
  {}
return 0;
}
```

asio concepts: Endpoint

- Combines protocol and address of a service
- Use by asio in variety of functions:
  - Server - listening for clients
  - Client - connecting to a server

Key endpoint functions

Method	Description
Constructor	basic_endpoint(const Protocol &protocol, unsigned short port_num)
Constructor for connect	basic_endpoint(const boost::asio::ip::address &addr, unsigned short port_num)

asio concepts: Resolver

- Resolver
  - Provides the ability to retrieve an endpoint(s) from name
  - Convert yahoo.com to an ip address
- Provides endpoint iterator

Key resolver functions

Method	Description
Constructor	basic_resolver(io_service&)
Resolve	iterator resolve (const endpoint_type &e) Resolve an endpoint to a list of entries.
Resolve query	iterator resolve(const query & q, boost::system::error_code & ec)

asio concepts: basic_socket

- Provides socket capabilities
- Concrete types for tcp and udp
  - boost::asio::ip::tcp::socket
  - boost::asio::ip::udp::socket

Socket – Getting - Setting Options

```
//set nodelay option
boost::asio::tcp::socket socket (io_service);
boost::asio::tcp::no_delay option(true);
socket.set_option(option);

//get no_delay setting
boost::asio::tcp::no_delay option;
socket.get_option(option);
bool is_set = option.get();
```

Key Socket Functions -

Method	Description
async_connect	Start a connect to an endpoint
connect	Start a connection - synchronous
cancel	Cancel all asynchronous operations associated with the socket.
close	Close the socket.
open	Open the socket using the specified protocol.
native	Get the underlying socket representation (non portable)

Key Socket Functions - Info

Method	Description
remote_endpoint	Get the other ip information from the other side of the socket.
local_endpoint	IP information from local side.

asio concepts: io_service

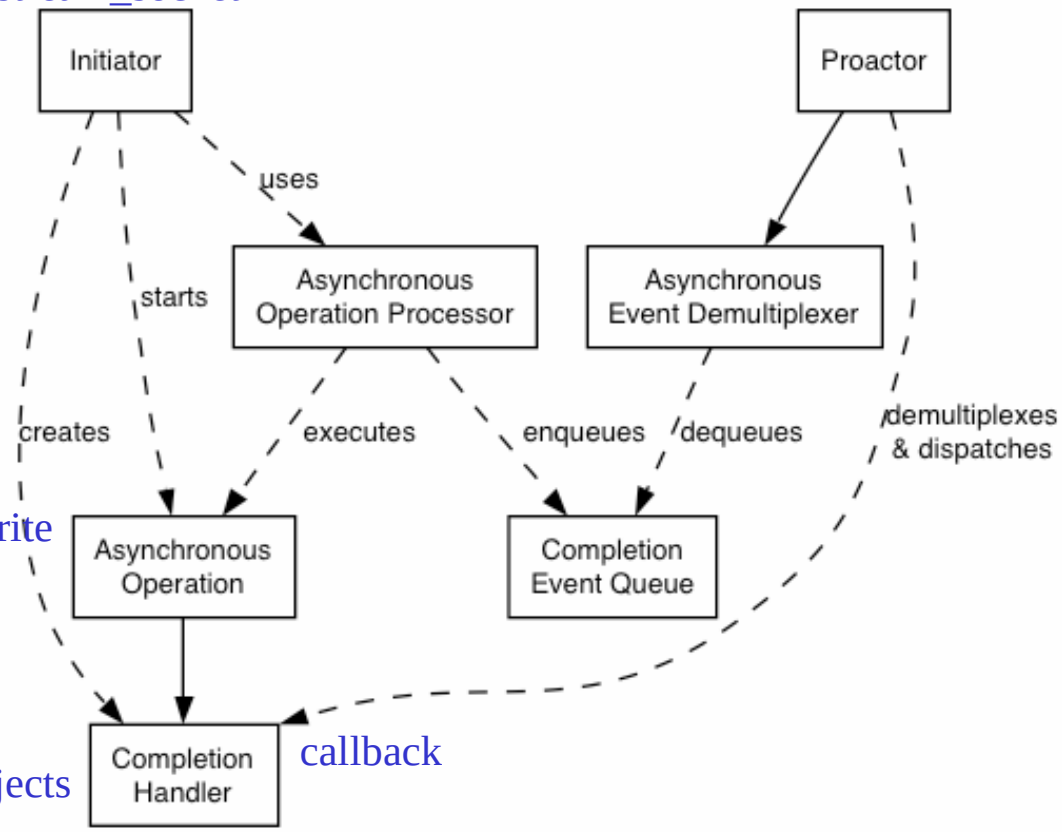
- Provides an ‘event loop’
  - Callback functions are registered
  - Requests initiated
  - Callbacks de-multiplexed and dispatched to handlers
- Allows for multi threaded operations
  - Calling run from multiple threads creates pool
  - Handler registration and thread NOT associated

Event De-multiplexing

Your application

Eg: asio::basic_stream_socket

boost::asio::io_service



eg: read or write on a socket

Function objects

callback

Key io_service functions

Method	Description
io_service	Default constructor.
run	Start event processing - blocks until all work has finished and there are no more handlers to be dispatched or stop is called.
Stop (was interrupt)	Stop the even loop. Returns immediately, there may be a delay before run returns.

asio concepts: timing – deadline_timer

- Timers a frequent need in network programming
  - Example: retry connection on failure
- Deadline timer provides this capability
  - Sync and async versions

Deadline timer sync

```
using boost::posix_time::seconds;
using boost::posix_time::milliseconds;
 // Construct a timer without setting an expiration time.
 boost::asio::deadline_timer timer(io_service);

 // Set an expiration time relative to now.
 timer.expires_from_now(seconds(5));
  // timer.expires_from_now(milliseconds(50));
 // Wait for the timer to expire.
 timer.wait();
```

Deadline Timer - async

```
void handler(const boost::asio::error& error) {
  if (!error)
  {
     // The current time is:
      boost::posix_time::ptime now(microsec_clock::local_time());
      std::cout << “Timeout callback: “ << now << std::endl;
     // Timer expired.
  }
}
// Construct a timer with an absolute expiration time.
 boost::asio::deadline_timer timer(io_service,
                                   boost::posix_time::time_from_string("2005-12-07 23:59:59.000"));
// boost::asio::deadline_timer timer(io_service, seconds(3)); //alternative for relative time
 // Start an asynchronous wait.
 timer.async_wait(handler);
```

Session Agenda

- Introduction
- Boost.asio Overview
- Basic Synchronous Client Program
- Boost.asio in depth
- Asynchronous Server Program
- Boost.serialization – short overview
- Pitfalls of Network Programming
- Network Programming Tools
- Resources and Conclusion

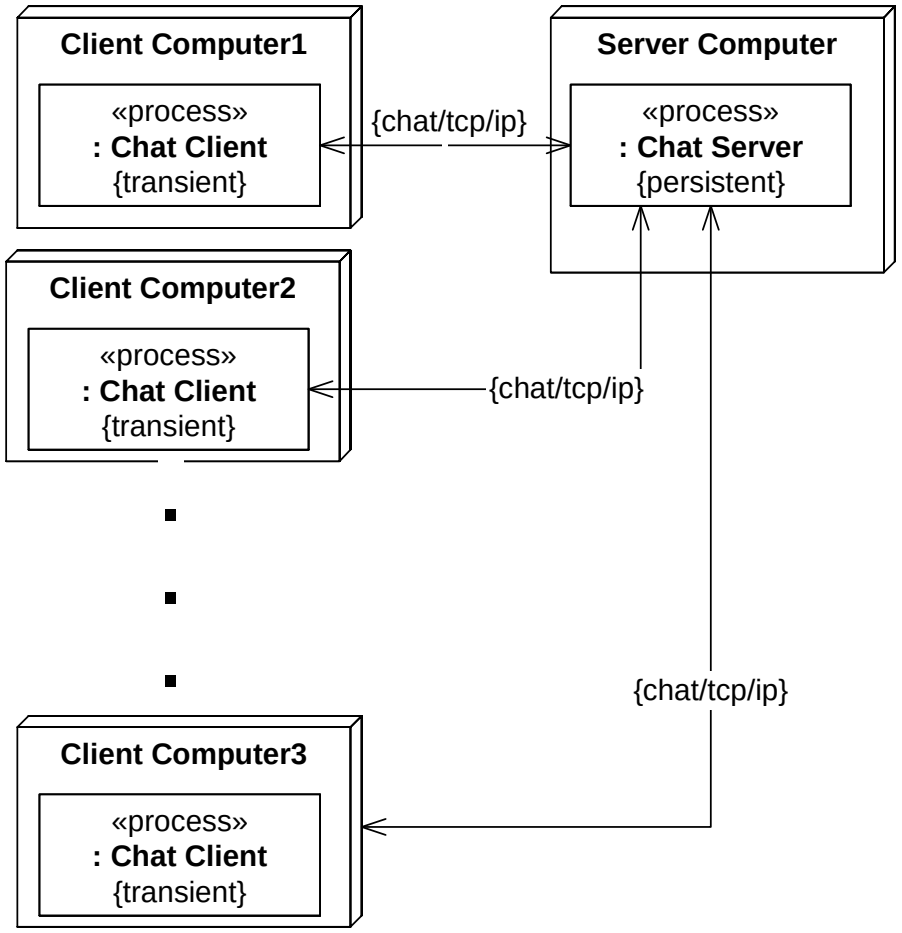
Building Chat Server

- Tasks include
  - Listen for a client to connect
  - ‘accept’ connection requests
  - Handle data from connections
    - When received send messages back to all clients
- Use async functions
- Use Chat program

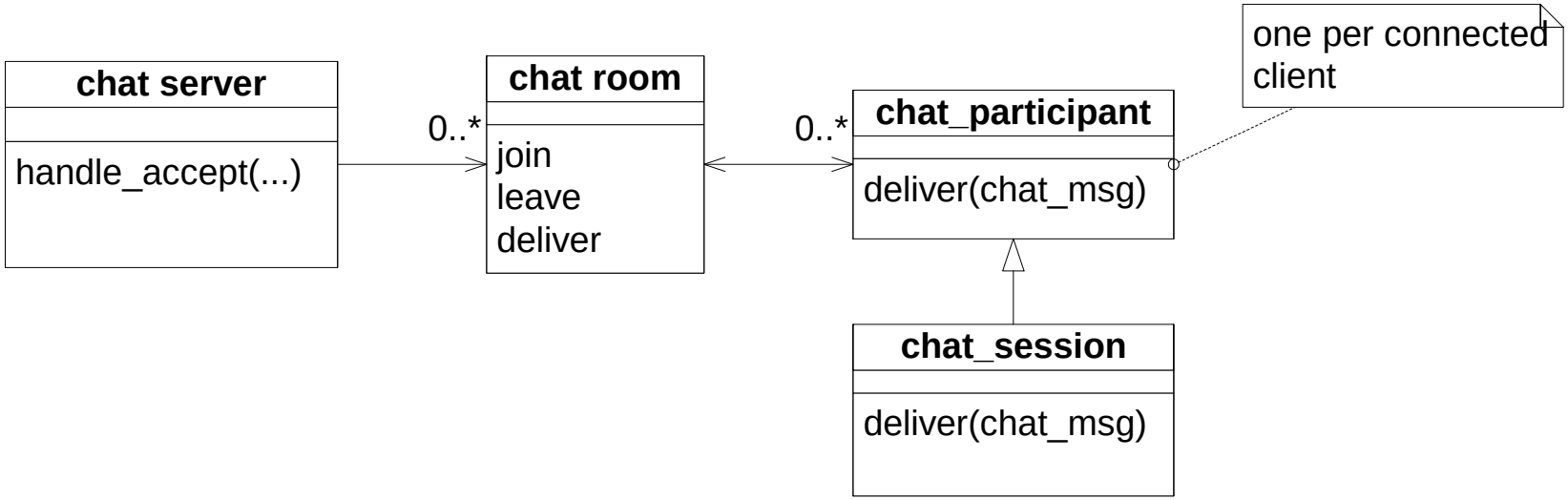
Acceptor pattern

- Separates ‘listening behavior’ from client behavior
- New ‘ServiceHandler’for each client
- Acceptor class is ‘factory’ for service handler
- May also handle any application specific setup protocol

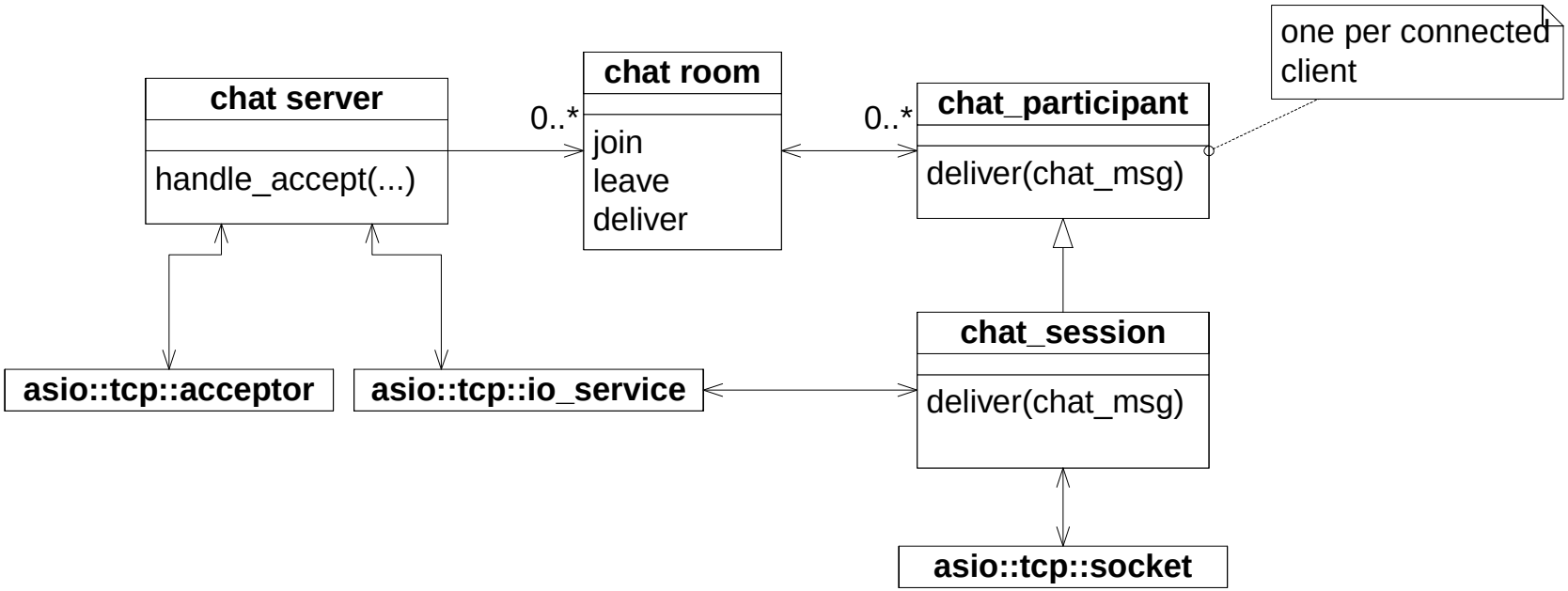
Chat Example – Network View



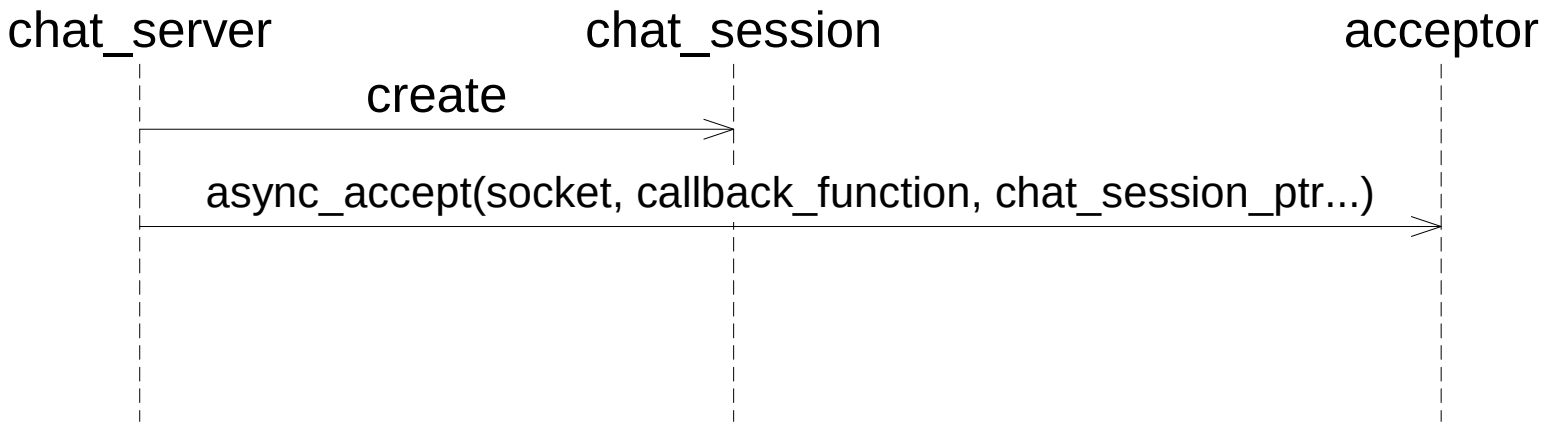
Chat Classes Overview



Chat Classes Detail



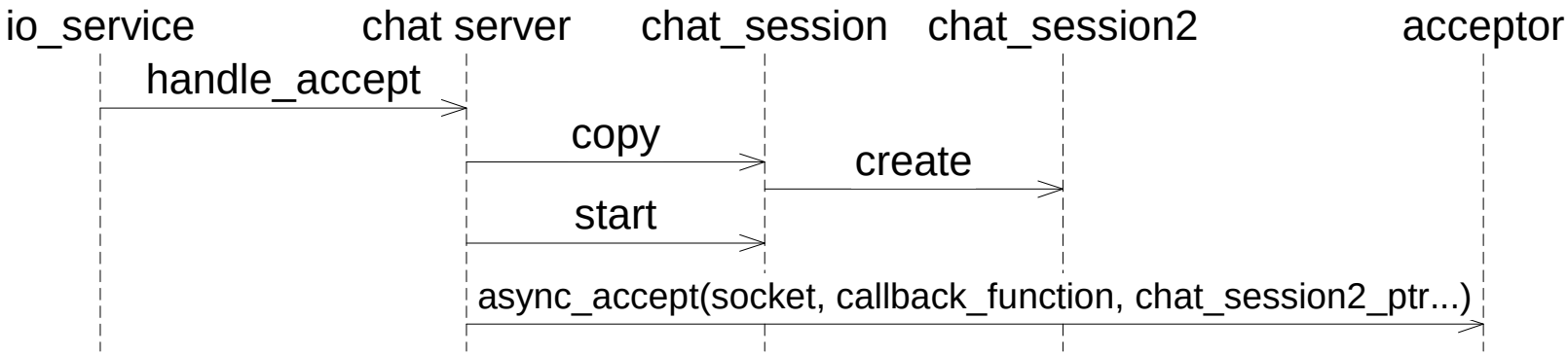
Listening for Client Connection



Listening for Client Connection

```
class chat_server
{
public:
  chat_server(boost::asio::io_service& io_service,
      const tcp::endpoint& endpoint)
    : io_service_(io_service),
      acceptor_(io_service, endpoint)
  {
    chat_session_ptr new_session(new chat_session(io_service_, room_));
    acceptor_.async_accept(new_session->socket(),
        boost::bind(&chat_server::handle_accept, this, new_session,
          boost::asio::placeholders::error));
  }
```

Session Setup Sequence



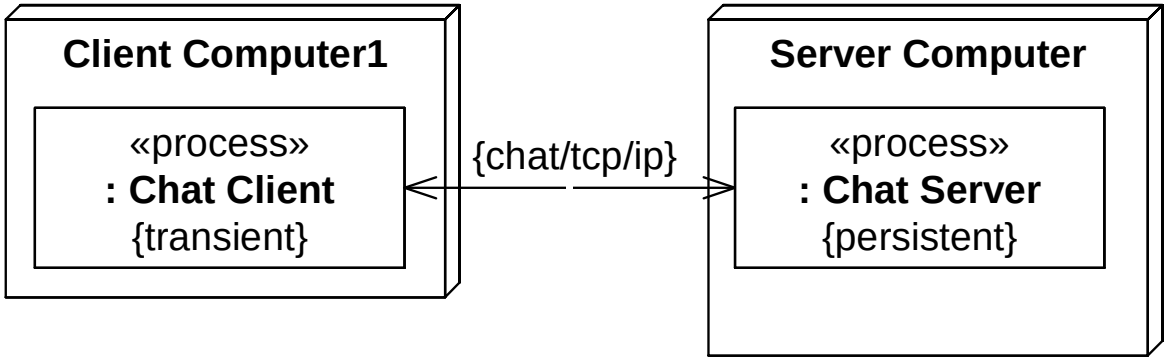
Session Setup Code

```
class chat_server
{

void handle_accept(chat_session_ptr session,
                   const boost::system::error_code& error)
  {
   if (!error)
   {
    session->start();
    chat_session_ptr new_session(new chat_session(io_service_, room_));
    acceptor_.async_accept(new_session->socket(),
                   boost::bind(&chat_server::handle_accept, this, new_session,
                   boost::asio::placeholders::error));
   }
  }
```

Chat - Current State

- Single established socket connection



Chat Session Start

- Wait for inbound socket data
- Setup async read for header

Chat – Handling Inbound Message – header

```
void handle_read_header(const boost::system::error_code& error)
{
  if (!error && read_msg_.decode_header())
  {
    boost::asio::async_read(socket_,
        boost::asio::buffer(read_msg_.body(), read_msg_.body_length()),
        boost::bind(&chat_session::handle_read_body, shared_from_this(),
          boost::asio::placeholders::error));
  }
  else
  {
    room_.leave(shared_from_this());
  }
}
```

Chat – Handling Inbound Message - Body

```
void handle_read_body(const boost::system::error_code& error)
 {
  if (!error)
  {
    room_.deliver(read_msg_);
    boost::asio::async_read(socket_,
        boost::asio::buffer(read_msg_.data(), chat_message::header_length),
        boost::bind(&chat_session::handle_read_header, shared_from_this(),
          boost::asio::placeholders::error));
  }
  else
  {
    room_.leave(shared_from_this());
  }
 }
```

Chat – sending outbound

```
class chat_room {
//…
void deliver(const chat_message& msg)
  {
    //…
      std::for_each(participants_.begin(), participants_.end(),
                    boost::bind(&chat_participant::deliver, _1, boost::ref(msg)));
  }
```

Chat – sending outbound

```
void deliver(const chat_message& msg)
 {
  bool write_in_progress = !write_msgs_.empty();
  write_msgs_.push_back(msg);
  if (!write_in_progress)
  {
   boost::asio::async_write(socket_,
       boost::asio::buffer(write_msgs_.front().data(),
         write_msgs_.front().length()),
       boost::bind(&chat_session::handle_write, shared_from_this(),
         boost::asio::placeholders::error));
  }
 }
```

Chat – Sending Outbound

```
void handle_write(const boost::system::error_code& error)
 {
  if (!error)
  {
    write_msgs_.pop_front();
    if (!write_msgs_.empty())
    {
      boost::asio::async_write(socket_,
        boost::asio::buffer(write_msgs_.front().data(),
          write_msgs_.front().length()),
        boost::bind(&chat_session::handle_write, shared_from_this(),
          boost::asio::placeholders::error));
    }
  }
  else
  {
    room_.leave(shared_from_this());
  }
 }
```

Chat Server – Main Program

```
//main…
try
  {
    //…snip…
    boost::asio::io_service io_service;

   chat_server_list servers;
   for (int i = 1; i < argc; ++i)
   {
     using namespace std; // For atoi.
     tcp::endpoint endpoint(tcp::v4(), atoi(argv[i]));
     chat_server_ptr server(new chat_server(io_service, endpoint));
     servers.push_back(server);
   }

   io_service.run();
  }
 //catch, exit, etc
```

Notes on Chat Concurrency

- Single threaded
- Will support many active clients
- Callback functions are ‘quick’
- Excellent example of how to build many basic servers

Other ways to implement Chat

- Wouldn't need to use point to point communication
- Could use UDP multi-cast
- Might not need a 'server' at all – peer to peer

Session Agenda

- Introduction
- Boost.asio Overview
- Basic Synchronous Client Program
- Boost.asio in depth
- Asynchronous Server Program
- Boost.serialization – short overview
- Pitfalls of Network Programming
- Network Programming Tools
- Resources and Conclusion

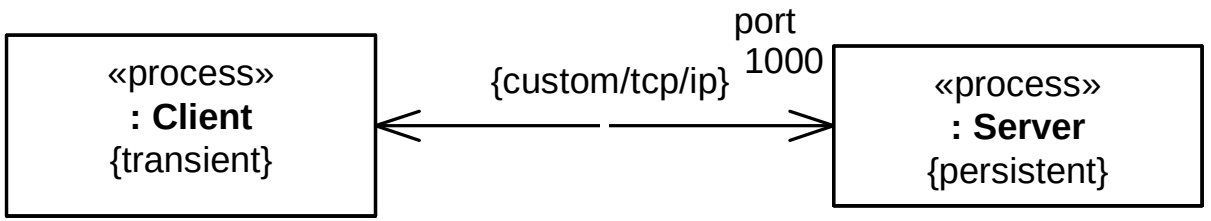
Task: Send arbitrary Graph of Objects Between Processes

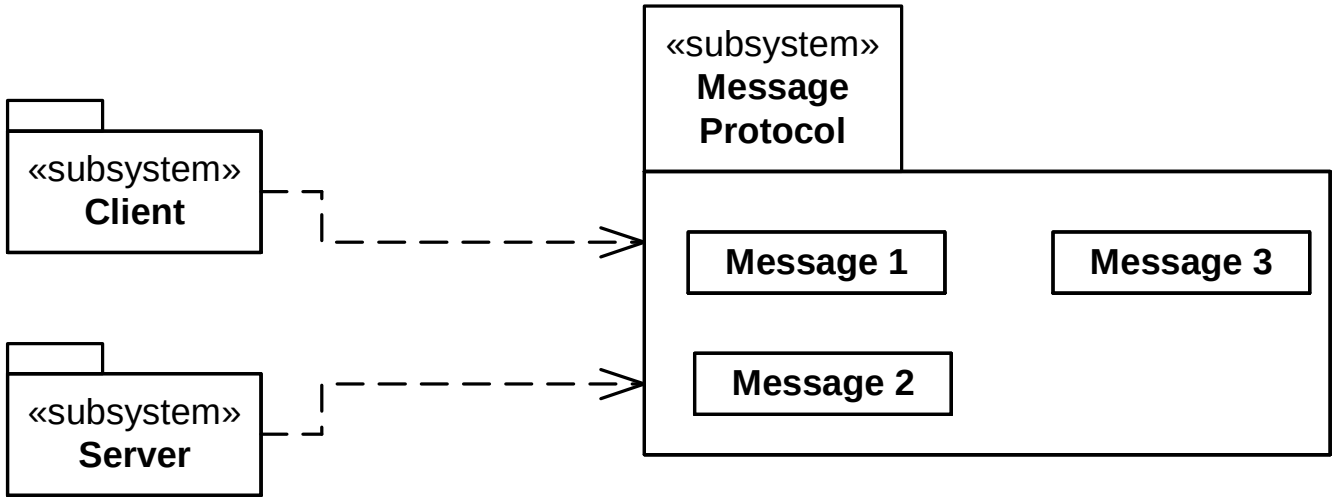
- Ultimately this is the task required in many network programs
- Implications
  - User defined types must be supported
  - Collections support
  - Must work with various platforms/compilers

Serialization Design

- Serialization is Design pattern from long ago
- Separation of Concerns:
  - Object layout from format of ‘stream’
- Useful for network programs where data layout is a problem

Messaging Architecture with Serialization





serialization 101 – What is Boost.Serialization?

- Provides the ability to take data structures and make into a sequence of bytes
- Users can provide functions to serialize their own data types
- Automatically handles STL collections
- Many Boost libraries support
  - Date-time, multi-index, shared_ptr, etc
- Author: Robert Ramey

serialization 101 – history

- Initial version submitted in Feb 2002
- Initial review Nov 2002 – rejected
- Second review April 2004 – accepted
- Included in 1.32 release
- Details: http://www.boost.org/libs/serialization/doc/index.html

serialization 101 – design goals

- Non-intrusive serialization
- Support for different archive formats (xml, binary)
- Archives can be extended by users
- Support for different stream types
- Versioning of classes

serialization 101 – platforms

- Summary – modern compilers prefered
- VC6 (limits), VC7.1, VC8
- Intel 8+
- Borland (some limits)
- g++ 3.x + (full support – wchar issues on mingw)
- Sun 5.8
- HP-UX acc6 v2

serialization 101 – Built Library

- Serialization is built into libraries
- To utilize will need to link appropriate libs

Serializaton 101 - Archive Concept: Renders the data into an iostream

- Several different formats
  - Text based: ‘text’, ‘xml’
  - Binary: ‘binary’
- Archives can be ‘loading’ or ‘saving’
- Archive class naming:
  - ‘format’_’chartype’’direction’archive
  - Where ‘chartype == blank (narrow) || ‘w’ wide
  - Where ‘direction’ == ‘i’ in || ‘o’ out
  - Xml, wide char, output == boost::archive::xml_woarchive

serialization 101 – save a value

```
#include <boost/archive/text_oarchive.hpp>
#include <boost/date_time/gregorian/greg_serialize.hpp>

const date d(2007, May, 14);
std::ostringstream archive_stream;
boost::archive::text_oarchive archive(archive_stream);
archive << d; //2007-May-14 now in archive_stream buffer
```

serialization 101 – serialization for custom types

```
#include <boost/date_time/gregorian/greg_serialize.hpp>
#include <boost/serialization/vector.hpp>
#include <boost/serialization/string.hpp>

class MyMessageType {
  vector<int> vi;
  date d;
  string s;
  friend class boost::serialization::access;
  template<class Archive>
  serialize(Archive& ar, const unsigned int version) const
  {
        ar & s;
        ar & vi;
        ar & d; //d has custom serialization method
 }
  public:
   //constructors and useful functions
};
```

serialization 101 – serialization for custom types

```
template<class Archive>
serialize(Archive& ar, MyMessageType m &
        const unsigned int version) const
{
    ar & m.get_vi();
    ar & m.get_d();
    ar & m.get_s();
}
```

serialization 101 – serialization for custom types - xml

```
template<class Archive>
serialize(Archive& ar, MyMessageType m &
        const unsigned int version) const
{
    ar & m.get_vi();
    ar & m.get_d();
    ar & m.get_s();
}
```

Boost Serialization for Networked Programs

- Boost.serialization doesn't have a 'cross-platform' binary archive
- Need to stick to text-based archives such as xml or text archives
- Serialization could be extend to support true binary formats
  - CDR (from CORBA)
  - XDR (an IETF binary data standard)
    - http://www.faqs.org/rfcs/rfc1014.html
  - YAML SOC project

Tradeoffs

- Advantages of Serialization Approach
  - Separates rendering from data types
  - Handles complex cases including pointers
  - Ensures client/server ‘automatically’ in sync
- Disadvantages
  - Not as fast as custom code

Inbound Message Handling Design

- Problem: Connection supports multiple types of messages – how do you serialize and call the write handler?

Stock Trade
string symbol double price time sale_time long quantity

Stock Quote
string symbol double ask_price double bid_price

Inbound Message Design

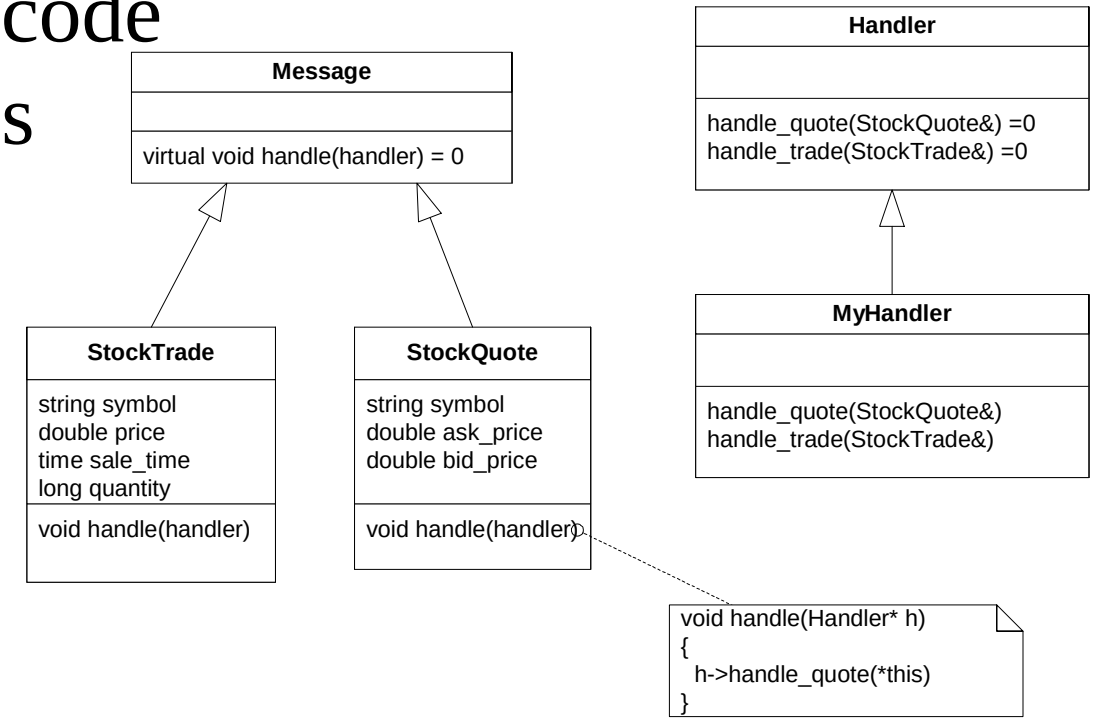
- Simple – add a message id to header then serialize correct type:

```
if (type == trade) {
    StockTrade st;
    ar >> st;
    handle_trade(st);
else if (type == quote) {
    StockQuote sq;
    ar >> sq;
    handle_quote(sq);
}
```

Inbound Message Design

- Use polymorphism
- Serialization code never changes

```
Message m;
ar >> m;
m.handle();
```



Session Agenda

- Introduction
- Boost.asio Overview
- Basic Synchronous Client Program
- Boost.asio in depth
- Asynchronous Server Program
- Boost.serialization – short overview
- Pitfalls of Network Programming
- Network Programming Tools
- Resources and Conclusion

Pitfalls of Network Programming

- Assuming zero latency
  - But the speed of light is 186,000 mps?
- Assuming a reliable network
  - But TCP is reliable, right?
- Hard-coding addresses
  - But it's only a prototype?

Pitfall 1: Latency in-depth

- Simple LAN
  - Latency is minimal (< 0-10 milliseconds)
  - Most apps tested on local setup - dangerous
- WAN applications
  - Much larger latency – rule of thumb (< 10-100 milliseconds)
  - US/Europe theoretical min latency is ~ 30 milliseconds
- WAN over Satellite
  - Geostationary orbit - 22,300 miles (roundtrip 44, 600)
  - One hop == ~ .25 second delay – 250 milliseconds!
  - One round trip TCP sequence (send/ack) ~0.5 second delay
  - If one interaction requires 10 TCP request/responses that's 5 seconds!

Latency Measurements – LAN

rtt min/avg/max/mdev = 38.058/41.519/51.352/4.112 ms

jeff@jeffdev2:~$ **ping 192.168.1.100**

PING 192.168.1.100 (192.168.1.100) 56(84) bytes of data.

64 bytes from 192.168.1.100: icmp_seq=1 ttl=128 **time=1.37 ms**

64 bytes from 192.168.1.100: icmp_seq=2 ttl=128 **time=1.25 ms**

64 bytes from 192.168.1.100: icmp_seq=3 ttl=128 **time=3.11 ms**

Latency Measurements – Internet – 'close'

jeff@jeffdev2:~$ ping -c 10 www.yahoo.com

PING www.yahoo-ht3.akadns.net (209.131.36.158) 56(84) bytes of data.

64 bytes from f1.www.vip.sp1.yahoo.com (209.131.36.158): icmp_seq=1 ttl=52 time=40.4 ms

64 bytes from f1.www.vip.sp1.yahoo.com (209.131.36.158): icmp_seq=2 ttl=52 time=42.9 ms

64 bytes from f1.www.vip.sp1.yahoo.com (209.131.36.158): icmp_seq=3 ttl=52 time=37.3 ms

64 bytes from f1.www.vip.sp1.yahoo.com (209.131.36.158): icmp_seq=4 ttl=52 time=50.6 ms

64 bytes from f1.www.vip.sp1.yahoo.com (209.131.36.158): icmp_seq=5 ttl=52 **time=64.7 ms**

64 bytes from f1.www.vip.sp1.yahoo.com (209.131.36.158): icmp_seq=6 ttl=52 time=44.4 ms

64 bytes from f1.www.vip.sp1.yahoo.com (209.131.36.158): icmp_seq=7 ttl=52 time=37.6 ms

64 bytes from f1.www.vip.sp1.yahoo.com (209.131.36.158): icmp_seq=8 ttl=52 time=40.0 ms

64 bytes from f1.www.vip.sp1.yahoo.com (209.131.36.158): icmp_seq=9 ttl=52 time=51.5 ms

64 bytes from f1.www.vip.sp1.yahoo.com (209.131.36.158): icmp_seq=10 ttl=52 time=40.4 ms

--- www.yahoo-ht3.akadns.net ping statistics ---

10 packets transmitted, 10 received, 0% packet loss, time 9055ms

rtt min/avg/max/mdev = 37.357/**45.025**/64.710/8.030 ms

Latency Measurements – Internet – Arizona to Austrialia

jeff@jeffdev2:~$ **ping -c 10 au.yahoo.com**

PING p1.www.vip.aue.yahoo.com (203.84.217.32) 56(84) bytes of data.

64 bytes from p1.www.vip.aue.yahoo.com (203.84.217.32): icmp_seq=1 ttl=236 time=260 ms

64 bytes from p1.www.vip.aue.yahoo.com (203.84.217.32): icmp_seq=2 ttl=236 time=197 ms

64 bytes from p1.www.vip.aue.yahoo.com (203.84.217.32): icmp_seq=3 ttl=235 time=230 ms

64 bytes from p1.www.vip.aue.yahoo.com (203.84.217.32): icmp_seq=4 ttl=236 time=200 ms

64 bytes from p1.www.vip.aue.yahoo.com (203.84.217.32): icmp_seq=5 ttl=236 time=280 ms

64 bytes from p1.www.vip.aue.yahoo.com (203.84.217.32): icmp_seq=6 ttl=235 **time=284 ms**

64 bytes from p1.www.vip.aue.yahoo.com (203.84.217.32): icmp_seq=7 ttl=235 time=284 ms

64 bytes from p1.www.vip.aue.yahoo.com (203.84.217.32): icmp_seq=8 ttl=235 time=228 ms

64 bytes from p1.www.vip.aue.yahoo.com (203.84.217.32): icmp_seq=9 ttl=236 time=263 ms

64 bytes from p1.www.vip.aue.yahoo.com (203.84.217.32): icmp_seq=10 ttl=236 time=199 ms

--- p1.www.vip.aue.yahoo.com ping statistics ---

10 packets transmitted, 10 received, 0% packet loss, time 10549ms

rtt min/avg/max/mdev = 197.213/**242.937**/284.684/34.321 ms

Latency Measurements – Internet – Arizona to UK

jeff@jeffdev2:~$ **ping -c 10 uk.yahoo.com**

PING www.euro.yahoo-eu1.akadns.net (217.146.186.51) 56(84) bytes of data.

64 bytes from www.vip.ird.yahoo.com (217.146.186.51): icmp_seq=1 ttl=49 time=185 ms

64 bytes from www.vip.ird.yahoo.com (217.146.186.51): icmp_seq=2 ttl=49 time=166 ms

64 bytes from www.vip.ird.yahoo.com (217.146.186.51): icmp_seq=3 ttl=49 **time=174 ms**

64 bytes from www.vip.ird.yahoo.com (217.146.186.51): icmp_seq=4 ttl=49 time=169 ms

64 bytes from www.vip.ird.yahoo.com (217.146.186.51): icmp_seq=5 ttl=49 time=163 ms

64 bytes from www.vip.ird.yahoo.com (217.146.186.51): icmp_seq=6 ttl=49 time=166 ms

64 bytes from www.vip.ird.yahoo.com (217.146.186.51): icmp_seq=7 ttl=49 time=164 ms

64 bytes from www.vip.ird.yahoo.com (217.146.186.51): icmp_seq=8 ttl=49 time=165 ms

64 bytes from www.vip.ird.yahoo.com (217.146.186.51): icmp_seq=9 ttl=49 time=167 ms

64 bytes from www.vip.ird.yahoo.com (217.146.186.51): icmp_seq=10 ttl=49 time=167 ms

--- www.euro.yahoo-eu1.akadns.net ping statistics ---

10 packets transmitted, 10 received, 0% packet loss, time 9042ms

rtt min/avg/max/mdev = 163.422/**169.112**/185.240/6.168 ms

Designing for Latency

- New hardware isn't going to fix it – it's physics
- Consider using UDP (no acknowledgement)
- Minimize the number of application calls in protocol
- Example –
  - Client needs to retrieve 'user information'
  - Design 1: Remote User Object
    - Each user attribute (name, address) requires remote call
  - Design 2: Locally Cached User Object
    - One remote call, proxy makes remaining local

Pitfall 2: Reliability in depth

- Nothing about network is really reliable
  - Theoretically impossible
    - Node crashes, router overloads, software bugs
  - tcp behaviors
    - Buffering
    - Window sizing (nagle algorithm) – cuts bandwidth automatically
- 'Guaranteed message delivery'…not
  - Practically all schemes have limits
  - What if messages are 'out of order'?
- Design for failure – it is inevitable!

Designing for Failure

- Provide ‘graceful degredation mode’ if possible
  - Web browsers (cannot connect, offline mode)
- Work thru error cases
  - Connection based
    - Test connects when server not available
    - Disconnect processes at various times
    - Send periodic messages – ping messages
  - Messaging
    - Application level message sequencing
    - Servers may require retransmit facilities

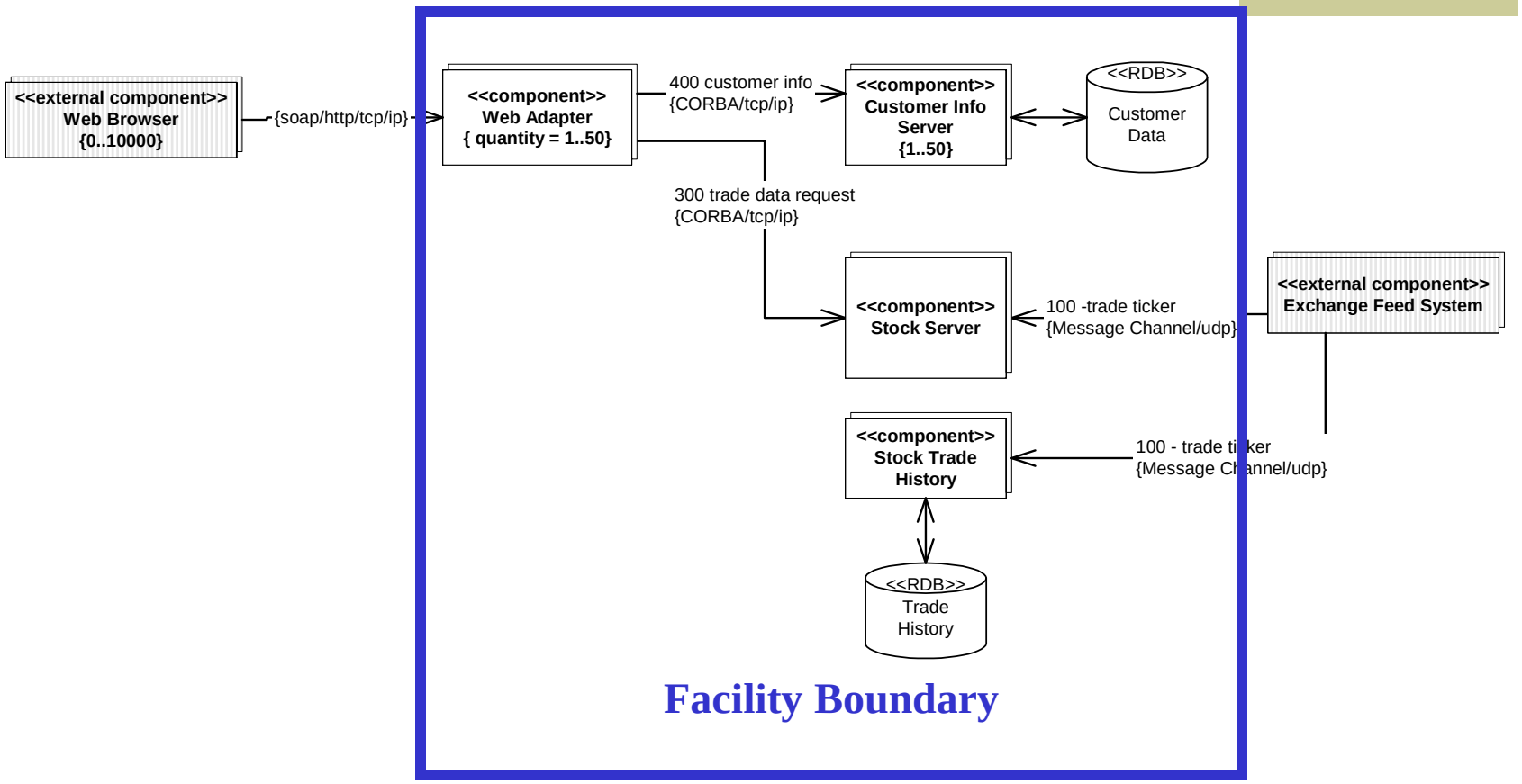
Example: OMG DCPS QoS Specification

- DCPS – Data Centric Publish Subscribe
- Suitable for ‘real-time’ machines
- 23 different quality of service options
  - History (keep last, keep all) – kind of replay
  - Presentation – coherent data transmission
  - Reliability – kind of sequencing
  - etc

Design for Failure – Startup Problem

- Design guideline
  - Order of process startup is completely arbitrary
  - Minimal amount of logic in each process
- Some functions may not be available until connectivity is complete
- Persistence often needed to all effective startup

Startup Problem – An Example



Session Agenda

- Introduction
- Boost.asio Overview
- Basic Synchronous Client Program
- Boost.asio in depth
- Asynchronous Server Program
- Boost.serialization – short overview
- Pitfalls of Network Programming
- Network Programming Tools
- Resources and Conclusion

Tools: Traceroute Example

$ traceroute www.yahoo.com

Tracing route to www.yahoo-ht3.akadns.net [209.131.36.158]

over a maximum of 30 hops:

1	**5 ms**	**9 ms**	**2 ms**	**f1.www.vip.sp1.yahoo.com [209.131.36.158]**
2	**29 ms**	**14 ms**	**14 ms**	**10.118.128.1**
3	**24 ms**	**10 ms**	**7 ms**	**ip68-2-6-110.ph.ph.cox.net [68.2.6.110]**
4	**16 ms**	**15 ms**	**13 ms**	**68.2.13.94**
5	**13 ms**	**17 ms**	**17 ms**	**68.2.13.9**
6	**12 ms**	**32 ms**	**89 ms**	**68.2.13.5**
7	**10 ms**	**13 ms**	**29 ms**	**68.2.13.1**
8	**15 ms**	**13 ms**	**27 ms**	**chnddsrj02-ae2.0.rd.ph.cox.net [68.2.14.5]**
9	**32 ms**	**43 ms**	**32 ms**	**langbbr01-ae0.r2.la.cox.net [68.1.0.232]**
10	**30 ms**	**25 ms**	**31 ms**	**exchange-cust1.la1.equinix.net [206.223.123.16]**
11	**40 ms**	*****	**38 ms**	**ge-1-3-4-p142.pat1.pao.yahoo.com [216.115.96.42]**
12	**38 ms**	**42 ms**	**40 ms**	**g-1-0-0-p150.msr2.sp1.yahoo.com [216.115.107.77]**
13	**41 ms**	**38 ms**	**42 ms**	**UNKNOWN-209-131-32-23.yahoo.com [209.131.32.23]**
14	**45 ms**	**41 ms**	**43 ms**	**f1.www.vip.sp1.yahoo.com [209.131.36.158]**

Trace complete.

Tools: netstat (windows)

$ netstat --help

Displays protocol statistics and current TCP/IP network connections.

NETSTAT [-a] [-b] [-e] [-n] [-o] [-p proto] [-r] [-s] [-v] [interval]

-a Displays all connections and listening ports.

-b Displays the executable involved in creating each connection or listening port. In some cases well-known executables host multiple independent components, and in these cases the sequence of components involved in creating the connection or listening port is displayed. In this case the executable name is in [] at the bottom, on top is the component it called, and so forth until TCP/IP was reached. Note that this option can be time-consuming and will fail unless you have sufficient permissions.

-e Displays Ethernet statistics. This may be combined with the -s option.

-n Displays addresses and port numbers in numerical form.

-o Displays the owning process ID associated with each connection.

-v When used in conjunction with -b, will display sequence of components involved in creating the connection or listening port for all executables.

Tools: netstat (unix)

```
usage: netstat [-veenNcCF] [<Af>] -r         netstat {-V|--version|-h|--help}
       netstat [-vnNcaeol] [<Socket> ...]
       netstat { [-veenNac] -i | [-cnNe] -M | -s }

        -i, --interfaces        display interface table
        -g, --groups            display multicast group memberships
        -v, --verbose           be verbose
        -n, --numeric           don't resolve names
        --numeric-hosts         don't resolve host names
        --numeric-ports         don't resolve port names
        --numeric-users         don't resolve user names
        -N, --symbolic          resolve hardware names
        -p, --programs          display PID/Program name for sockets
        -l, --listening         display listening server sockets
        -a, --all, --listening  display all sockets (default: connected)
        -o, --timers            display timers

  <Socket>={-t|--tcp} {-u|--udp} {-w|--raw} {-x|--unix} --ax25 --ipx --netrom
```

Tools: netstat (windows)

$ netstat -b

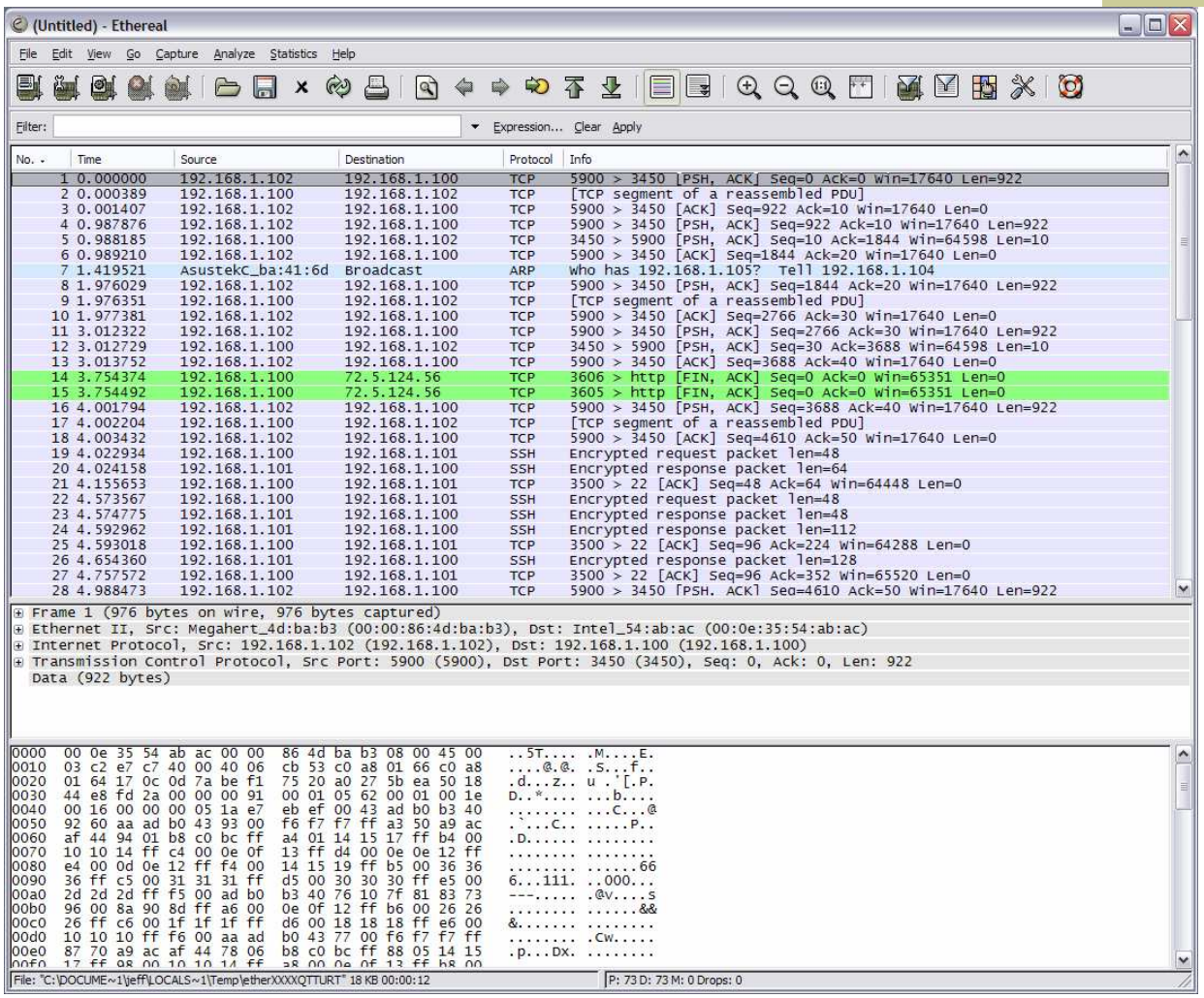
Active Connections

Proto	Local Address	Foreign Address	State	PID
TCP	jeff_tablet:3314	localhost:3315	ESTABLISHED	9944
[firefox.exe]				
TCP	jeff_tablet:3315	localhost:3314	ESTABLISHED	9944
[firefox.exe]				
TCP	jeff_tablet:3322	localhost:3323	ESTABLISHED	9944
[firefox.exe]				
TCP	jeff_tablet:3323	localhost:3322	ESTABLISHED	9944
[firefox.exe]				
TCP	jeff_tablet:3450	192.168.1.102:5900	ESTABLISHED	11592
[vncviewer.exe]				
TCP	jeff_tablet:3487	cg-in-f167.google.com:http	ESTABLISHED	9944
[firefox.exe]				

Ethereal/Wireshark

- Cross-platform network capture tool
  - Captures packets at os level
  - Great help in protocol debugging
  - Great learning tool
- Called ‘wireshark’ on some *nix platforms
- http://www.ethereal.com/

Tools: ethereal (windows)



Tools: ping

```
jeff@jeffdev2:~$ ping -c 10 www.yahoo.com
PING www.yahoo-ht3.akadns.net (209.131.36.158) 56(84) bytes of data.
64 bytes from f1.www.vip.sp1.yahoo.com (209.131.36.158): icmp_seq=1 ttl=52 time=38.3 ms
64 bytes from f1.www.vip.sp1.yahoo.com (209.131.36.158): icmp_seq=2 ttl=52 time=39.7 ms
64 bytes from f1.www.vip.sp1.yahoo.com (209.131.36.158): icmp_seq=3 ttl=52 time=40.6 ms
64 bytes from f1.www.vip.sp1.yahoo.com (209.131.36.158): icmp_seq=4 ttl=52 time=41.3 ms
64 bytes from f1.www.vip.sp1.yahoo.com (209.131.36.158): icmp_seq=5 ttl=52 time=47.2 ms
64 bytes from f1.www.vip.sp1.yahoo.com (209.131.36.158): icmp_seq=6 ttl=52 time=38.8 ms
64 bytes from f1.www.vip.sp1.yahoo.com (209.131.36.158): icmp_seq=7 ttl=52 time=38.0 ms
64 bytes from f1.www.vip.sp1.yahoo.com (209.131.36.158): icmp_seq=8 ttl=52 time=40.0 ms
64 bytes from f1.www.vip.sp1.yahoo.com (209.131.36.158): icmp_seq=9 ttl=52 time=51.3 ms
64 bytes from f1.www.vip.sp1.yahoo.com (209.131.36.158): icmp_seq=10 ttl=52 time=39.3 ms

--- www.yahoo-ht3.akadns.net ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 9034ms
rtt min/avg/max/mdev = 38.058/41.519/51.352/4.112 ms
```

Session Agenda

- Introduction
- Boost.asio Overview
- Basic Synchronous Client Program
- Boost.asio in depth
- Asynchronous Server Program
- Boost.serialization – short overview
- Pitfalls of Network Programming
- Network Programming Tools
- Resources and Conclusion

Other Libraries of Interest

- Channel - Yigong Liu
  - Built on asio for Message Passing
  - http://channel.sourceforge.net/
- Signal Network (2007 SoC) – Stjepan Rajko
  - Has ‘remote signal’ concept
  - http://dancinghacker.com/code/signet/index.html
- Marshal/RPC - Stjepan Rajko
  - Factory of RPC/Marshalling part from above
  - http://dancinghacker.com/code/marshal/index.html

Other Libraries of Interest

- RCF (Remote Call Framework) - **Jarl Lindrud**
  - Built on asio for C++ to C++ remote calls
  - http://www.codeproject.com/threads/Rcf_Ipc_For_Cpp.asp

Other Libraries of Interest

- ACE/TAO
  - Mature, portable C++ library includes threading, networking, shared memory, and CORBA implementations
  - ACE == Adaptive Communications Environment
  - TAO == Open source Object Request Broker
  - http://www.cs.wustl.edu/~schmidt/ACE.html

Boost.MPI – Message Passing Interface

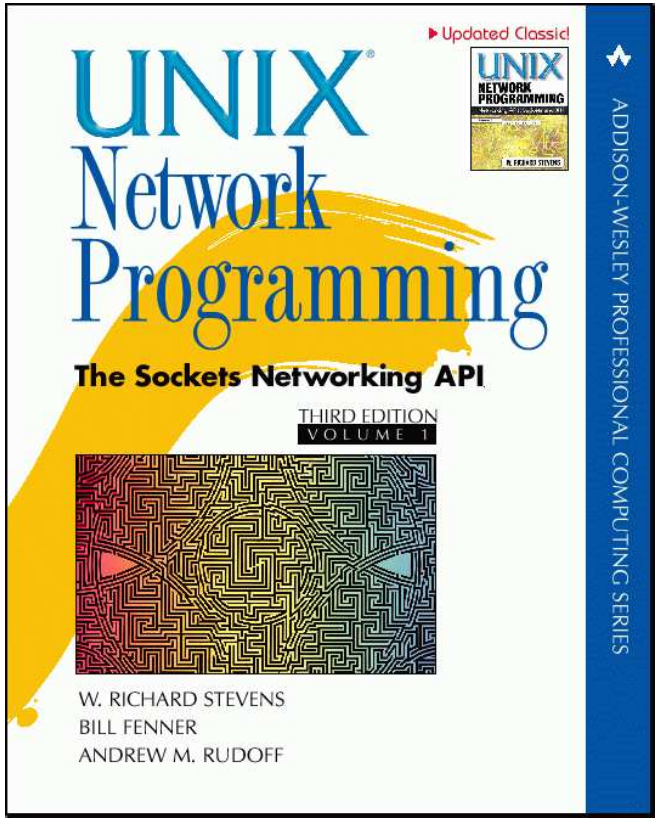
- Message Passing Interface
  - C++ binding to MPI framework
  - Requires external MPI library
  - MPI supports many languages
  - MPI is traditionally used for ‘cluster computing’
- In Boost 1.35
- Uses boost.serialization for user defined types
- Authors: Douglas Gregor, Matthias Troyer

Latency – Some Resources

- Read Waldo et. al. http://research.sun.com/techrep/1994/smli_tr-94-29.pdf
- AJAX Latency http://richui.blogspot.com/2005/09/ajax-latency-problems-myth-or-reality.html

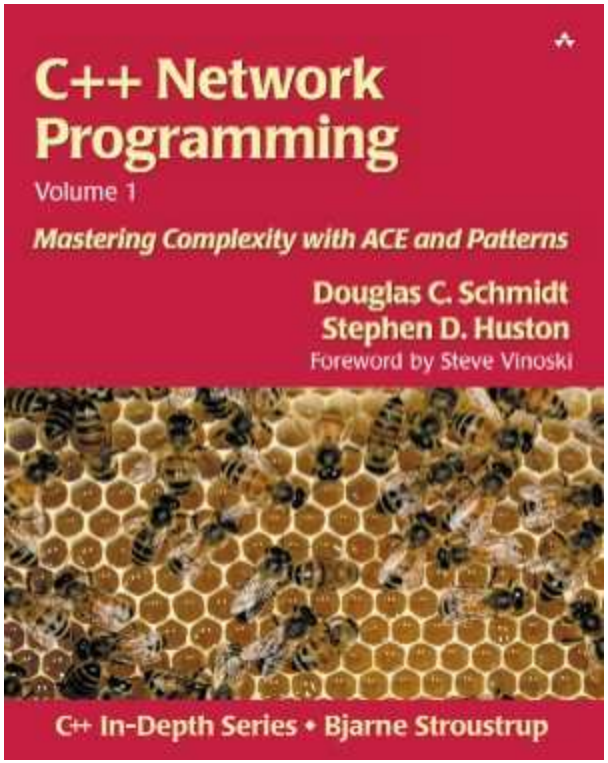
Network Programming Resources

- Unix Network Programming
  - http://www.unpbook.com
- Other Stevens books
  - TCP Illustrated
  - 3 volumes



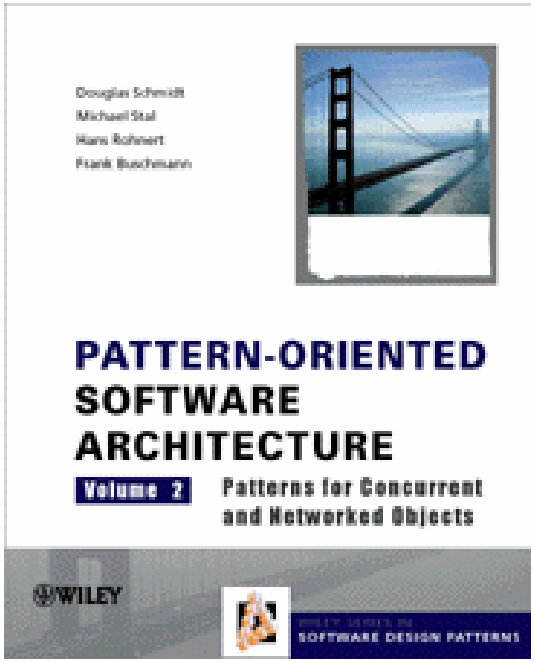
Schmidt C++ Books

- 2 volumes
- Describes details of using ACE
- Recommended if you're using ACE



Schmidt Patterns

- Vol2 of Pattern Oriented Software Architectures
- Application design level networking descriptions



Summary

- Network programming is ubiquitous
- New Boost libraries are making it easier than ever
- Go do it!