Introduction to Middleware

Petr Tůma

Department of Distributed and Dependable Systems
Faculty of Mathematics and Physics
Charles University

2017 – 2018

Part I

Sockets: The Hard Way

Outline

1. Berkeley Socket Interface
2. Assignment Part I
3. Marshalling Implementation
4. Assignment Part II

Interface Overview

Socket

An abstraction representing a (network) communication channel.

Both stream oriented and message oriented channels.

Spectrum of supported protocols.

Stream Oriented Channel

Socket on *client side* initiates outgoing connections.

Socket on *server side* waits for incoming connections.

Data flows in both directions after connection established.

Message Oriented Channel

No connection established.

Sender and receiver roles symmetrical.

Stream Oriented Channel

Client Side Pseudocode

```
socket = CreateSocket (comms_domain, socket_type);
ConnectToServer (socket, server_address);
... Write (socket, data);
... Read (socket, data);
Shutdown (socket);
Close (socket);
```

Server Side Pseudocode

```
server_socket = CreateSocket (comms_domain, socket_type);
BindToLocalAddress (socket, address);
PermitListeningOnSocket (socket, backlog);
client_socket, client_address = AcceptIncomingConnection (socket);
... Write (client_socket, data);
... Read (client_socket, data);
Shutdown (client_socket);
Close (client_socket);
```

Outline

1. Berkeley Socket Interface
2. Assignment Part I
3. Marshalling Implementation
4. Assignment Part II

Assignment

Server

Implement a server that will:

- Listen for incoming connections.
- Provide information on current time to connected clients.

Client

Implement a client that will:

- Connect to the server described above.
- Query information on current time.
- Wrap all this in a local function.
- Print the time.

C Local Function

```
/**
 * Return server time in standard structure.
 * \param result Caller allocated structure to fill.
 * \return Zero for success, non zero error code otherwise.
 */
int server_time (struct tm *result);

struct tm {
    int tm_sec;    // Seconds (0-60)
    int tm_min;    // Minutes (0-59)
    int tm_hour;   // Hours (0-23)
    int tm_mday;   // Day of the month (1-31)
    int tm_mon;    // Month (0-11)
    int tm_year;   // Year - 1900
    int tm_wday;   // Day of the week (0-6, Sunday = 0)
    int tm_yday;   // Day in the year (0-365, 1 Jan = 0)
    int tm_isdst;  // Daylight saving time
};
```

… man localtime

Java Local Function

```
/**
 * Access server time in standard structure.
 */
public interface ServerTime {
    int getSecond ();           // Gets the second-of-minute field.
    int getMinute ();           // Gets the minute-of-hour field.
    int getHour ();             // Gets the hour-of-day field.
    int getDayOfMonth ();       // Gets the day-of-month field.
    Month getMonth ();          // Gets the month-of-year field.
    int getYear ();             // Gets the year field.
    DayOfWeek getDayOfWeek ();  // Gets the day-of-week field.
    int getDayOfYear ();        // Gets the day-of-year field.
}
```

… javadoc `LocalDateTime`

Python Local Function

```
def server_time ():
    """Returns server time in datetime.datetime class."""
    ...

# Instance attributes (read-only):
#
#     datetime.year
#         Between MINYEAR and MAXYEAR inclusive.
#     datetime.month
#         Between 1 and 12 inclusive.
#     datetime.day
#         Between 1 and the number of days in the given month of the given year.
#     datetime.hour
#         In range(24).
#     datetime.minute
#         In range(60).
#     datetime.second
#         In range(60).
```

… help (datetime.datetime)

Examples To Begin With …

```
> git clone http://github.com/D-iii-S/teaching-introduction-middleware.git
```

C

```
> cd teaching-introduction-middleware/src/sockets-basic-server/c
> cat README.md
```

Java

```
> cd teaching-introduction-middleware/src/sockets-basic-server/java
> cat README.md
```

Python

```
> cd teaching-introduction-middleware/src/sockets-basic-server/python
> cat README.md
```

Outline

1. Berkeley Socket Interface
2. Assignment Part I
3. Marshalling Implementation
4. Assignment Part II

C Marshalling

Textual Stream ?

```
int sprintf (char *str, const char *format, ...);
int sscanf (const char *str, const char *format, ...);
```

Network Order Binary Stream ?

```
uint32_t htonl (uint32_t hostlong);
uint16_t htons (uint16_t hostshort);
uint32_t ntohl (uint32_t netlong);
uint16_t ntohs (uint16_t netshort);
```

Native Order Binary Stream ?

```
char buffer [1024];
int *address = (int *) &buffer [16];
*address = 1234;
```

Java Marshalling

Serialized Stream ?

```
output_stream = socket.getOutputStream ();
object_stream = new ObjectOutputStream (output_stream);
object_stream.writeInt (1234);
object_stream.writeObject (...);
```

Textual Stream ?

```
PrintWriter writer = new PrintWriter (output_stream, true);
writer.println ("...");
```

Byte Stream ?

```
ByteBuffer buffer = ByteBuffer.allocate (4);
buffer.putInt (1234);
output_stream.write (buffer.array ());
```

Python Marshalling

Pickled Stream ?

```
with socket.makefile () as file_object:
    pickle.dump (..., file_object)
```

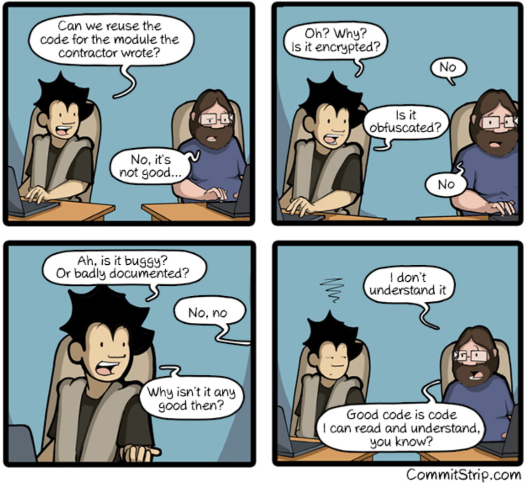
JSON Stream ?

```
with socket.makefile () as file_object:
    json.dump (..., file_object)
```

Byte Stream ?

```
data = 1234;
socket.send (data.to_bytes (4, 'little'))
```

Code Now ...


Can we reuse the code for the module the contractor wrote?
No, it's not good...
Oh? Why? Is it encrypted?
No
Is it obfuscated?
No
Ah, is it buggy? Or badly documented?
No, no
Why isn't it any good then?
I don't understand it
Good code is code I can read and understand, you know?
CommitStrip.com

`http://www.commitstrip.com/en/2016/06/07/good-code`

Outline

1. Berkeley Socket Interface
2. Assignment Part I
3. Marshalling Implementation
4. Assignment Part II

Assignment

Languages

Implement Part I in at least two programming languages.

Interoperability

Make sure your clients and servers in both languages are interchangeable:

- Run any client with any server.
- Basic fields are enough (YYYY-MM-DD HH:MM:SS).
- Use sensible defaults for other fields (TZ, DOW, DOY).

Part II

Protocol Buffers: Marshalling

Outline

5. Technology Overview
6. Assignment Part I
7. Message Encoding
8. Message Specification
9. Message Manipulation
10. Assignment Part II

Technology Overview

Goals

Provide platform independent structured data serialization framework.

Features

- Platform independent data description language.
- Serialization code generation for multiple languages (C++, Java, Python, Go, Ruby, JavaScript, Objective C, C# …).
- Binary transport format with compact data representation.

… `http://developers.google.com/protocol-buffers`

Installation

Fedora Packages

```
> dnf install protobuf-devel
> dnf install python3-protobuf
```

Source Distribution

```
> git clone -b 3.6.x http://github.com/protocolbuffers/protobuf
> cd protobuf
> ./autogen.sh
> ./configure --prefix=${HOME}/.local
> make -j 8
> make install
> export PATH=${HOME}/.local/bin:${PATH}
> export LD_LIBRARY_PATH=${HOME}/.local/lib:${LD_LIBRARY_PATH}
> export PKG_CONFIG_PATH=${HOME}/.local/lib/pkgconfig:${PKG_CONFIG_PATH}
```

... use supplied install-protobuf.sh script

Message Specification Example

```
syntax = "proto3";

package example;

message AnExampleMessage {
    uint32 some_integer = 1;
    sint32 another_integer = 2;
    string some_string = 8;
    repeated string some_more_strings = 11;
}

message MoreExampleMessages {
    repeated AnExampleMessage messages = 1;
}
```

Version Summary

	proto2	proto3
Fields	required and optional	optional
Default values	custom values	zero or null
Missing values	indicated	same as default
Unknown fields	preserved	version dependent
Message extensions	extension fields	any type fields
Encoding	binary	binary and JSON

Outline

5 Technology Overview

6 Assignment Part I

7 Message Encoding

8 Message Specification

9 Message Manipulation

10 Assignment Part II

Assignment

Server

Implement a server that will provide information on current time.

- The server should accept a spec of what fields to return.
- Fields should be standard YYYY-MM-DD HH:MM:SS.

Client

Implement a client that will query server time:

- Pick a random combination of fields.
- Query information on current time.
- Print the time.

Interoperability

Implement compatible clients and servers in two languages.

Examples To Begin With ...

```
> git clone http://github.com/D-iii-S/teaching-introduction-middleware.git
```

C

```
> cd teaching-introduction-middleware/src/protocol-buffers-basic-usage/c
> cat README.md
```

Java

```
> cd teaching-introduction-middleware/src/protocol-buffers-basic-usage/java
> cat README.md
```

Python

```
> cd teaching-introduction-middleware/src/protocol-buffers-basic-usage/python
> cat README.md
```

Outline

5 Technology Overview

6 Assignment Part I

7 Message Encoding

8 Message Specification

9 Message Manipulation

10 Assignment Part II

Message Encoding

Goals

Compact structure with support for field removal and addition.

Features

- Sequence of field key value pairs.
- Key is field index and type indication.
  - One of variable integer, explicit length, fixed length.
  - Not enough to tell the exact field type !
- Primitive repeated fields packed.

Variable Length Encoding

Goals

Support integers clustered around zero more efficiently.

Features

- Integer stored as variable number of 7 bit values.
- High bit set to zero for last byte.
- Little endian byte order.
- Signed variant.

Outline

5 Technology Overview

6 Assignment Part I

7 Message Encoding

8 Message Specification

9 Message Manipulation

10 Assignment Part II

Primitive Field Types

Integer Types

(s)fixed(32|64)	Integers with fixed length encoding.
(u)int(32|64)	Integers with variable length encoding.
sint(32|64)	Integers with sign optimized variable length encoding.

Floating Point Types

float	IEEE 754 32 bit float.
double	IEEE 754 64 bit float.

Additional Primitive Types

bool	Boolean.
bytes	Arbitrary sequence of bytes.
string	Arbitrary sequence of UTF-8 characters.

More Field Types

Oneof Type

```
message AnExampleMessage {
    oneof some_oneof_field {
        int32 some_integer = 1;
        string some_string = 2;
    }
}
```

Enum Type

```
enum AnEnum {
    INITIAL = 0;
    RED = 1;
    BLUE = 2;
    GREEN = 3;
    WHATEVER = 8;
}
```

More Field Types

Any Type

```
import "google/protobuf/any.proto”;
message AnExampleMessage {
    repeated google.protobuf.Any whatever = 8;
}
```

Map Type

```
message AnExampleMessage {
    map<int32, string> keywords = 8;
}
```

Outline

5 Technology Overview

6 Assignment Part I

7 Message Encoding

8 Message Specification

9 Message Manipulation

10 Assignment Part II

C++ Message Basics

Construction

```
AnExampleMessage message;
AnExampleMessage message (another_message);
message.CopyFrom (another_message);
```

Singular Fields

```
cout << message.some_integer ();
message.set_some_integer (1234);
```

Repeated Fields

```
int size = messages.messages_size ();
const AnExampleMessage &message = messages.messages (1234);
AnExampleMessage *message = messages.mutable_messages (1234);
AnExampleMessage *message = messages.add_messages ();
```

C++ Message Serialization

Byte Array

```cpp
char buffer [BUFFER_SIZE];
message.SerializeToArray (buffer, sizeof (buffer));
message.ParseFromArray (buffer, sizeof (buffer));
```

Standard Stream

```cpp
message.SerializeToOstream (&stream);
message.ParseFromIstream (&stream);
```

Java Message Basics

Construction

```
AnExampleMessage.Builder messageBuilder;
messageBuilder = AnExampleMessage.newBuilder ();
messageBuilder = AnExampleMessage.newBuilder (another_message);
AnExampleMessage message = messageBulder.build ();
```

Singular Fields

```
System.out.println (message.getSomeInteger ());
messageBuilder.setSomeInteger (1234);
```

Repeated Fields

```
int size = messages.getMessagesCount ();
AnExampleMessage message = messages.getMessages (1234);
List<AnExampleMessage> messageList = messages.getMessagesList ();
messages.addMessages (messageBuilder);
messages.addMessages (message);
```

Java Message Serialization

Byte Array

```
byte [] buffer = message.toByteArray ();
try {
    AnExampleMessage message = AnExampleMessage.parseFrom (buffer);
} catch (InvalidProtocolBufferException e) {
    System.out.println (e);
}
```

Standard Stream

```
message.writeTo (stream);
AnExampleMessage message = AnExampleMessage.parseFrom (stream);
```

Python Message Basics

Construction

```
message = AnExampleMessage ()
message.CopyFrom (another_message)
```

Singular Fields

```
print (message.some_integer)
message.some_integer = 1234
```

Repeated Fields

```
size = len (messages.messages)
message = messages.messages [1234]
message = messages.messages.add ()
```

Python Message Serialization

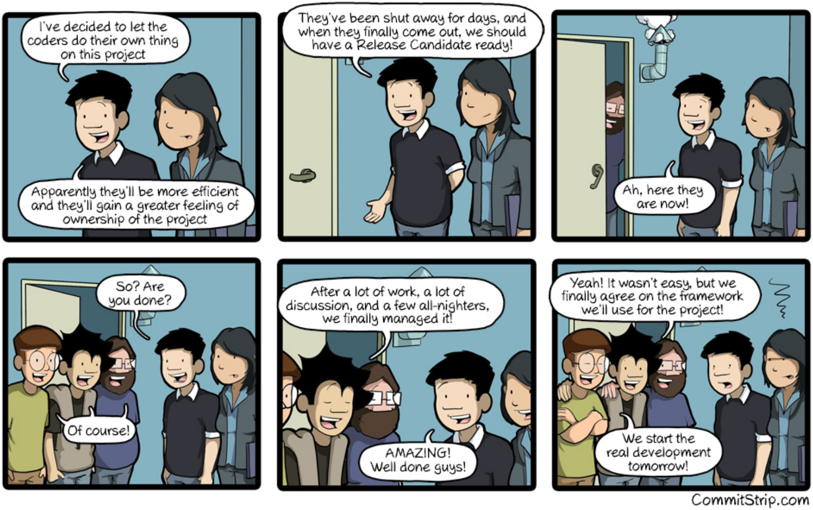
Byte Array

```
buffer = message.SerializeToString ()
message.ParseFromString (buffer)
message = AnExampleMessage.FromString (buffer)
```

Standard Stream

```
file.write (message.SerializeToString ())
message.ParseFromString (file.read ())
AnExampleMessage.FromString (file.read ())
```

Code Now ...


I've decided to let the coders do their own thing on this project
Apparently they'll be more efficient and they'll gain a greater feeling of ownership of the project
They've been shut away for days, and when they finally come out, we should have a Release Candidate ready!
Ah, here they are now!
So? Are you done?
Of course!
After a lot of work, a lot of discussion, and a few all-nighters, we finally managed it!
AMAZING! Well done guys!
Yeah! It wasn't easy, but we finally agree on the framework we'll use for the project!
We start the real development tomorrow!
CommitStrip.com

http://www.commitstrip.com/en/2017/03/16/when-we-leave-coders-to-do-their-own-thing

Outline

5. Technology Overview
6. Assignment Part I
7. Message Encoding
8. Message Specification
9. Message Manipulation
10. Assignment Part II

Assignment

Performance

Measure the performance of your implementation.

Experiment Design

Stick to the following, or provide arguments for why not:

- Random field mix, each field with probability 1/2.
- Measure at least two minutes long traffic.
- Report average invocation throughput.
- No printing during measurement.
- Compare with first assignment.

Measuring Time

C++

```
#include <time.h>
#include <stdint.h>
struct timespec time;
clock_gettime (CLOCK_MONOTONIC_RAW, &time);
uint64_t nanoseconds =
    (uint64_t) time.tv_sec * 1000000000 +
    (uint64_t) time.tv_nsec;
```

Java

```
long nanoseconds = System.nanoTime ();
```

Python

```
import time
nanoseconds = time.clock_gettime (time.CLOCK_MONOTONIC_RAW) * 1000000000
```

Part III

gRPC: Remote Procedure Call

Outline

11 Technology Overview

12 Assignment Details

Technology Overview

Goals

Provide platform independent remote procedure call mechanism.

Features

- Protocol buffers as interface description language.
- Stub code generation for multiple languages (C++, Java, Python, Go, Ruby, JavaScript, PHP, C# ...).
- Binary transport format with compact data representation.
- Supports streaming arguments during remote call.
- Synchronous and asynchronous invocation code.
- Compression support at transport level.
- Security support at transport level.

... `http://www.grpc.io`

Installation

Source Distribution

```
> git clone -b v1.7.x http://github.com/grpc/grpc
> cd grpc
> git submodule update --init
> make -j 8 prefix=${HOME}/.local
> make install
> export PATH=${HOME}/.local/bin:${PATH}
> export LD_LIBRARY_PATH=${HOME}/.local/lib:${LD_LIBRARY_PATH}
> export PKG_CONFIG_PATH=${HOME}/.local/lib/pkgconfig:${PKG_CONFIG_PATH}
```

... use supplied install-grpc.sh script

Service Specification Example

```
syntax = ”proto3”;

message AnExampleRequest { ... }
message AnExampleResponse { ... }

service AnExampleService {

    rpc OneToOneCall (AnExampleRequest) returns (AnExampleResponse) { }

    rpc OneToStreamCall (AnExampleRequest)
        returns (stream AnExampleResponse) { }

    rpc StreamToStreamCall (stream AnExampleRequest)
        returns (stream AnExampleResponse) { }
}
```

Outline

11 Technology Overview

12 Assignment Details

Assignment

Server

Implement a server that will provide information on current time.

- The server should accept a spec of what fields to return.
- Fields should be standard YYYY-MM-DD HH:MM:SS.

Client

Implement a client that will query server time:

- Pick a random combination of fields.
- Query information on current time.
- Print the time.

Interoperability

Implement compatible clients and servers in two languages.

C++ Service Basics

Implementation

```
class MyService : public AnExampleService::Service {
    grpc.Status OneToOne (grpc.ServerContext *context,
        const AnExampleRequest *request, AnExampleResponse *response) {
        // Method implementation goes here ...
        return (grpc.Status::OK);
    }
    ...
```

Execution

```
MyService service;
grpc.ServerBuilder builder;
builder.AddListeningPort ("localhost:8888", grpc.InsecureServerCredentials ());
builder.RegisterService (&service);
std::unique_ptr<grpc.Server> server (builder.BuildAndStart ());

server->Wait ();
```

C++ Client Basics

Connection

```
std::shared_ptr<grpc.Channel> channel = grpc.CreateChannel (
    "localhost:8888", grpc.InsecureChannelCredentials ());
```

Invocation

```
grpc.ClientContext context;
AnExampleResponse response;
std::shared_ptr<AnExampleService::Stub> stub = AnExampleService::NewStub (channel);
grpc.Status status = stub->OneToOne (&context, request, &response);
if (status.ok ()) {
    // Response available here ...
}
```

Java Service Basics

Implementation

```
class MyService extends AnExampleServiceGrpc.AnExampleServiceImplBase {
    @Override public void OneToOne (
        AnExampleRequest request,
        io.grpc.stub.StreamObserver<AnExampleResponse> responseObserver) {
        // Method implementation goes here ...
        responseObserver.onNext (response);
        responseObserver.onCompleted ();
    }
    ...
```

Execution

```
io.grpc.Server server = io.grpc.ServerBuilder
    .forPort (8888).addService (new MyService ()).build ().start ();

server.awaitTermination ();
```

Java Client Basics

Connection

```
io.grpc.ManagedChannel channel = io.grpc.ManagedChannelBuilder
    .forAddress ("localhost", 8888)
    .usePlaintext (true)
    .build ();
```

Invocation

```
AnExampleServiceGrpc.AnExampleServiceBlockingStub stub =
    AnExampleServiceGrpc.newBlockingStub (channel);
AnExampleResponse response = stub.oneToOne (request);
// Response available here ...
```

Python Service Basics

Implementation

```
class MyServicer (AnExampleServiceServicer):
    def OneToOne (self, request, context):
        # Method implementation goes here ...
        return response
```

Execution

```
server = grpc.server (
    futures.ThreadPoolExecutor (
        max_workers = SERVER_THREAD_COUNT))
add_AnExampleServiceServicer_to_server (MyServicer (), server)
server.add_insecure_port ("localhost:8888")
server.start ()
```

Python Client Basics

Connection

```
channel = grpc.insecure_channel (”localhost:8888”)
```

Invocation

```
stub = AnExampleServiceStub (channel)
response = stub.OneToOne (request)
# Response available here ...
```

Part IV

JGroups: Multicast Messaging

Outline

13. Technology Overview

14. Assignment Details

Technology Overview

Goals

Provide reliable group messaging mechanism.

Features

- Basic group messaging interface.
- Groups identified by names.
- Messages are byte arrays.
- Configurable protocol stack.
  - Multiple underlying transports.
  - Multiple reliability mechanisms.
  - Multiple membership discovery mechanisms.
  - Multiple error recovery mechanisms.
  - ...

... `http://www.jgroups.org`

JChannel Class

```
public class JChannel implements Closeable {
    public JChannel ();
    public JChannel (File file);
    public JChannel (URL properties);
    public JChannel (Element properties);

    public void connect (String cluster_name);
    public void disconnect ();

    public void send (Message msg);
    public void send (Address dst, byte [] buf);
    public void send (Address dst, Object obj);

    public void setReceiver (Receiver r);
    public Receiver getReceiver ();

    public View getView ();

    public void addChannelListener (ChannelListener listener);
    public void removeChannelListener (ChannelListener listener);

    ...
}
```

Message Class

```
public class Message ... {
    public Message (Address dest);
    public Message (Address dest, byte [] buf);
    public Message (Address dest, Object obj);

    public Address getDest ();
    public Message setDest (Address new_dest);
    public Address getSrc ();
    public Message setSrc (Address new_src);

    public int getOffset ();
    public int getLength ();
    public byte [] getBuffer ();
    public Message setBuffer (byte[] b);
    public Message setBuffer (byte[] b, int offset, int length);

    ...
}
```

Outline

13 Technology Overview

14 Assignment Details

Assignment

Peer

Implement a process that will update shared hash map.

- The shared hash map is available through SharedHashMap channel.
- The updates are transmitted through UpdateEvent class.

```java
import java.io.Serializable;

public class UpdateEvent implements Serializable {
    private static final long serialVersionUID = 0xBAADBAADBAADL;

    public int key;
    public String value;
}
```

ReceiverAdapter Class

```
public class ReceiverAdapter implements Receiver {
    public void receive (Message msg);
    public void receive (MessageBatch batch);

    public void block ();
    public void unblock ();

    public void getState (OutputStream output);
    public void setState (InputStream input);

    public void suspect (Address mbr);
    public void viewAccepted (View view);
}
```

ChannelListener Interface

```java
public interface ChannelListener {
    public void channelClosed (JChannel channel);
    public void channelConnected (JChannel channel);
    public void channelDisconnected (JChannel channel);
}
```

Assignment Extension

Peer

Implement a process that will track shared hash map state.

- The shared hash map is available through `SharedHashMap` channel.
- The updates are transmitted through `UpdateEvent` class.

```java
import java.io.Serializable;

public class UpdateEvent implements Serializable {
    private static final long serialVersionUID = 0xBAADBAADBAADL;

    public int key;
    public String value;
}
```

Quiz

- How would you go about measuring the cluster throughput ?
- Will the entire cluster see the same state ?

Part V

Google Cloud: Secure Communication

Outline

15 Technology Overview

16 Assignment Details

RSA Refresher

Public Key Cryptography

A key pair where data encrypted with one key (private or public) can be decrypted with the other one (public or private).

- Public key available, private key kept secret
- Encrypting with public key, signing with private key

$x^{(p-1)(q-1)} = 1 \ (\mathit{modulo}\ pq)$ ... for p, q prime and x not commensurable with pq

pick p, q
have $n = pq$ and $\phi = (p-1)(q-1)$
pick e, d such that $ed = 1 \ (\mathit{modulo}\ \phi)$
then $(m^e)^d = m^{1+k(p-1)(q-1)} = m \cdot m^{k(p-1)(q-1)} = m \ (\mathit{all\ modulo}\ n)$

... Martin Ouwehand: The (simple) Mathematics of RSA

DH Refresher

Shared Secret Agreement

A process through which parties can agree on a shared secret without actually transmitting the shared secret itself.

have p and g where g is a generator of multiplicative integer group modulo p

Alice: pick a and publish g^a $(modulo\ p)$
Bob: pick b and publish g^b $(modulo\ p)$
then $(g^a)^b = (g^b)^a$ is a shared secret

TLS Technology Overview

Goals

Provide privacy and integrity guarantees in network communication.

Features

- Ciper suite negotiation
  - Key exchange (RSA, DHE, PSK ...)
  - Encryption (AES GCM, AES CCM, AES CBC ...)
  - Message authentication (MD5, SHA1, SHA256 ...)
- Secure session key exchange
- Server authentication
- Data encryption
- Data integrity

... TLS 1.2 RFC 5246

TLS RSA Handshake Sketch

```
[CLT] Hello, I support these cipher suites,
      and here is my CLIENT RANDOM number

[SRV] Hello, I have picked cipher suite AES256-SHA256,
      here is my SIGNED SERVER CERTIFICATE
      and here is my SERVER RANDOM number

[CLT] Here is a random PRE MASTER SECRET encrypted with your RSA key

MASTER SECRET = function (PRE MASTER SECRET, CLIENT RANDOM, SERVER RANDOM)
various session keys = function (MASTER SECRET)

[CLT] Finished and here is encrypted hash of exchanged messages

[SRV] Finished and here is encrypted hash of exchanged messages
```

TLS DH Handshake Sketch

```
[CLT] Hello, I support these cipher suites,
      and here is my CLIENT RANDOM number

[SRV] Hello, I have picked cipher suite AES256-SHA256,
      here is my SIGNED SERVER CERTIFICATE
      and here is my SERVER RANDOM number

[SRV] Here is my signed SERVER DH PUBLIC KEY

[CLT] Here is my CLIENT DH PUBLIC KEY

PRE MASTER SECRET = function (CLIENT DH PUBLIC KEY, SERVER DH PUBLIC KEY)
MASTER SECRET = function (PRE MASTER SECRET, CLIENT RANDOM, SERVER RANDOM)
various session keys = function (MASTER SECRET)

[CLT] Finished and here is encrypted hash of exchanged messages

[SRV] Finished and here is encrypted hash of exchanged messages
```

Outline

15 Technology Overview

16 Assignment Details

Assignment

Server

Implement a server that will provide information on current time.

- The server should accept a spec of what fields to return.
- Fields should be standard YYYY-MM-DD HH:MM:SS.

Client

Implement a client that will query server time:

- Pick a random combination of fields.
- Query information on current time.
- Print the time.

Security

The connection between the client and the server should be encrypted.

Python Secure Connection Basics

Server

```
key_data = open ('server.key', 'rb').read ()
crt_data = open ('server.crt', 'rb').read ()
credentials = grpc.ssl_server_credentials ([( key_data, crt_data )])

server = grpc.server (...)
server.add_secure_port (SERVER_ADDR, credentials)
```

Client

```
crt_data = open ('server.crt', 'rb').read ()
credentials = grpc.ssl_channel_credentials (root_certificates = crt_data)
channel = grpc.secure_channel (SERVER_ADDR, credentials)

stub = AnExampleServiceStub (channel)
```

OAuth Technology Overview

Goals

Standard protocol for granting third party applications limited access to HTTP accessible resources.

Features

- Considers multiple client types
  - Applications running in browser
  - Server hosted applications acting on own behalf
  - Server hosted applications acting on user behalf
- Heavily uses browser request redirection
- Requires (mostly) encrypted communication
- Authentication represented by (secret) access token

… OAuth 2.0 RFC 6749

Authorization Process Participants

Resource Owner

This is the end user who authorizes third party clients to access resources.
The resource owner accesses the third party client through a browser.

Resource Server

This is the server that provides access to resources
when shown authorization in the form of access token.

Third Party Client

This is the application that needs to access
resources on behalf of resource owner.

Authorization Server

This is the server that can authenticate the resource owner
and issues access tokens as directed by the resource owner.

Authorization Process Sketch

```
[OWN] Accesses an application link that needs authorization.

[APP] Responds with REDIRECT sending the browser to authorization server.
      The link includes CLIENT ID and SCOPE and arbitrary STATE.

[OWN] The browser follows the link to the authorization server.

[AUT] The server authenticates the user behind the browser.
      The user is then asked to grant authorization for SCOPE.
      The server concludes with REDIRECT back to the application.
      The link includes AUTHORIZATION CODE and associated application STATE.

[OWN] The browser follows the link to the application.

[APP] The application gets the AUTHORIZATION CODE from the link.
      The application asks the authorization server to convert
      the AUTHORIZATION CODE into an ACCESS TOKEN.

[AUT] The server generates the ACCESS TOKEN as requested.

[APP] The application accesses the resource server
      with the ACCESS TOKEN included in request header.
```

Google Cloud Platform Technology Overview

Goals

Computing platform build on Google infrastructure resources and services.

Features

- Tons of services
  - Compute services (IaaS and PaaS and FaaS)
  - Storage services (SQL, tables, documents, raw block storage)
  - Networking (private networks, load balancing, content delivery)
  - Big data processing
  - Machine learning
  - Management
- Accessible through public interfaces
- Libraries for multiple languages

… `http://cloud.google.com`

Installation

Browser

- Register for free trial at http://cloud.google.com
- Log in to console at http://console.cloud.google.com
- Create a new project
- Enable required libraries
- Create and download a service account key

Shell

```
> export GOOGLE_APPLICATION_CREDENTIALS=/path/to/service-account-key.json
```

Cloud Speech API

```
from google.cloud import speech as google_cloud_speech
from google.cloud.speech import enums as google_cloud_speech_enums
from google.cloud.speech import types as google_cloud_speech_types

client = google_cloud_speech.SpeechClient ()

content = read_data_from_file (...)
audio = google_cloud_speech_types.RecognitionAudio (content = content)
config = google_cloud_speech_types.RecognitionConfig (language_code = 'en-US')

result = client.recognize (config, audio)
```

… http://cloud.google.com/speech/docs

Cloud Translate API

```
from google.cloud import translate as google_cloud_translate

client = google_cloud_translate.Client ()

# Get a list of all supported languages.
languages = client.get_languages ()

# Translate a sentence.
result = client.translate ('some␣text', target_language = 'en')
```

… http://cloud.google.com/translate/docs

Assignment

Goal

Create a client that translates input speech.

- An audio file with speech in English on input
- A text with speech translated into Czech on output

Implementation

Use the client libraries rather than generated stub code.

Part VI

Swagger: REST API Generation

Outline

17 Technology Overview

18 Assignment Details

REST: Representational State Transfer

Features

REST compliant web services allow requesting systems to access and manipulate textual representations of web resources using a uniform and predefined set of stateless operations.

... Wikipedia

Practically: each object (for example each database record) has its own URL and each action on the object a specific method or a specific child URL.

- Add new person with POST at `http://example.com/person/add`
- Get person info with GET at `http://example.com/person/42`
- Update person info with POST at `http://example.com/person/42`
- Delete person info with DELETE at `http://example.com/person/42`

REST: Motivation

Motivation

Strike balance between
need for *explicit interfaces*
and need for *loose coupling*.

- Standard communication protocol (HTTP)
  - Already defines CRUD operations
  - Provides security and reliability
  - Is easy to deploy across internet
- Encourages separating model from view
- Supports independent implementation technology between client and server

REST and CRUD

CRUD

Create	to create an object
Read	to query object attributes
Update	to update object attributes
Delete	to delete an object

- The recommended minimum set of operations
- Corresponds reasonably well to HTTP methods
- Anything beyond CRUD is not considered pure REST

REST: Data Transfer

Data exchange format is application specific but there are obvious choices

- JSON because of JavaScript in the browser
- XML because of existing library support

```
{
    "name": "Jane Doe",
    "email":  "jane.doe@example.com",
    "url": [
        "http://example.com/~jane.doe",
        "http://example.com/people/jane.doe"
    ],
    "address": {
        "street1": "Our Street One",
        "street2": "Street Line Two",
        "city": "The City",
        "postal": "12345"
    },
    "room": 123
}
```

Swagger: API Development for REST

Interface Description

URLs	to identify data model classes
Actions	to operate on class instances
Attributes	with types to describe class instances
Security	defines access rules
Comments	provide human readable description

- Code generation
  - Stubs wrap communication in language or framework specific constructs
  - RPC style with futures for client
  - Callback style for server
  - Over 80 targets supported
- Editor at `http://editor.swagger.io`.

Outline

17 Technology Overview

18 Assignment Details

Assignment

Inventory Application

Keeps track of *users* and *assets*.

Basic user related operations are already defined.

Define similar operations for assets and implement everything.

- Interface
  - Elementary CRUD operations for assets
  - One to many relationship between users and assets
- Server
  - Python implementation using Flask, or
  - Java implementation using Spring
- Client
  - TypeScript implementation using Angular, or
  - R and bash helper scripts

Assignment Interface: Prologue

```
swagger: 2.0

info:
  description: Inventory database service
  version: 1.0.0
  title: Inventory
  termsOfService: ""
  license:
    name: Apache 2.0
    url: "http://www.apache.org/licenses/LICENSE-2.0.html”

host: localhost:8080

basePath: /v1

schemes:
  - http
```

Assignment Interface: Listing Users

```
paths:
  /users:
    get:
      operationId: readUsers
      produces:
        - "application/json"
      responses:
        200:
          schema:
            type: array
            items:
              $ref: "#/definitions/UserBase"

definitions:
  UserBase:
    type: object
    properties:
      id:
        type: integer
      firstname:
        type: string
      lastname:
        type: string
```

Assignment Interface: Querying User Data

```
/user/{id}:
  get:
    summary: Query user information.
    operationId: readUser
    parameters:
      - in: path
        name: id
        description: ID of the user.
        required: true
        type: integer
    produces:
      - "application/json”
    responses:
      200:
        description: Successful operation
        schema:
          type: object
          $ref: "#/definitions/User”
```

Assignment Interface: Updating User Data

```
post:
  summary: Update user information.
  operationId: updateUser
  consumes:
    - "application/json”
  produces:
    - "application/json”
  parameters:
    - in: path
      name: id
      description: ID of the user.
      required: true
      type: integer
    - in: body
      name: body
      description: Updated data.
      required: true
      schema:
        $ref: "#/definitions/User”
  responses:
    405:
      description: Invalid input
```

Assignment Interface: Inheritance

```
definitions:
  UserBase:
    type: object
    properties:
      id:
        type: integer
      firstname:
        type: string
      lastname:
        type: string
      email:
        type: string
  User:
    allOf:
      - $ref: "#/definitions/UserBase"
      - type: object
        properties:
          homepage:
            type: string
          department:
            type: string
```

Code Generation

```
> swagger-codegen generate -i api.yaml -o <path> -l <framework>
```

Assignment

Use scripts `build-{server,client}-*.sh`
after updating `api.yaml` to invoke code generator.

Flask-Based and Spring-Based Servers

General

- No real database (data kept in memory)
- Data dump to JSON at termination for debugging
- See `README` for instructions how to run

Flask-Based Server

swagger_server/controllers/default_controller.py

```python
def create_user(body):  # noqa: E501
    """Creates a new user.

    :param body: User to be added.
    :type body: dict | bytes

    :rtype: None
    """
    if connexion.request.is_json:
        body = User.from_dict(connexion.request.get_json())
    return 'do␣some␣magic!'
```

controllers/users.py

Actual implementation with data kept in memory.

Spring-Based Server

src/gen/java/io/swagger/api/UsersApiController.java

```
public ResponseEntity<Void> createUser (
    @ApiParam (value = "User␣to␣be␣added." ,required=true)
    @Valid
    @RequestBody
    User body)
{
    String accept = request.getHeader("Accept");
    return new ResponseEntity<Void> (HttpStatus.NOT_IMPLEMENTED);
}
```

src/main/java/io/swagger/api/UsersApiController.java

Actual implementation with data kept in memory.

Angular-Based Client

Goal

Add interface components for listing complete inventory.
Extend user detail page with asset list.

General

- Sources are under `src/app`
- `*.component.html` contains web page snippets of the component
- `*.component.ts` contains TypeScript implementation of the component

Angular-Based Client

app-routing.module.ts

- Import all your components
- Add new routes to routes

app.component.html

- Items in the topbar

Angular-Based Client: Reading Server Data

users/users.component.ts

```
export class UsersComponent implements OnInit {
    users: User [];

    constructor (private api: DefaultService) {}

    ngOnInit () {
        this.api.readUsers ().subscribe (u => this.users = u);
    }
}
```

users/users.component.html

```
<ul>
    <li *ngFor="let␣user␣of␣users">
        <a routerLink="/user/{{user.id}}">{{user.lastname}}, {{user.firstname}}</a>
    </li>
</ul>
```

Angular-Based Application: Writing Server Data

users/user.component.html

```html
<form (ngSubmit)="save();">
    <label for="user-first-name">First name:</label>
    <input [(ngModel)]="user.firstname" id="user-first-name" />
    ...
    <button type="submit">Save</button>
</form>
```

users/user.component.ts

```typescript
export class UserComponent {
    save (): void {
        const id = +this.route.snapshot.paramMap.get ('id');
        this.api.updateUser (id, this.user).subscribe ();
    }
}
```

Bash Client: Overview

Generated

The generated script `client.sh` is a thin wrapper on top of `curl` doing the actual requests. Useful to check that the server works as expected.

`make-check-lists` and `add-employees.sh`

Downloads list of employees, creates printable version of the inventory. Reads employee list from a CSV, adds them to the database.

Task

Extend the `make-check-lists` to include assets listing and create a similar script for adding assets.

```
asset,price,acquired,owner
Magic Wand,42,2017,harry.potter@example.com
...
```

Bash Client: Usage

```
> ./client.sh --silent readUsers | json_reformat

> ./client.sh --silent readUser id=1

> ./client.sh createUser \
    firstname==Horatio lastname==Hornblower \
    email==horatio.hornblower@royalnavy.mod.uk \
    department==Navy \
    homepage==https://www.royalnavy.mod.uk/hornblower
```

R Client: Overview

`dept-plot.r`

Draws a barplot showing number of employees in each department.

Task

Create a similar script that will show total price of assets across departments and for each employee.

dept-plot.r

```
source (”init.r”)

api <- DefaultApi$new ()

all.users.id <- api$read_users ()$content$id

department.people.count <- list ()

for (i in all.users.id) {
    u <- api$read_user (i)$content
    dept <- u$department
    if (!(dept %in% names (department.people.count))) {
        department.people.count [[ dept ]] <- 0
    }

    department.people.count [[ dept ]] <- department.people.count [[ dept ]] + 1
}

barplot (unlist (department.people.count), main=”Employee␣count␣per␣department”)
```

Assignment Summary

- Extend `api.yaml` with assets-related operations and data definitions
- Extend one of the servers (Flask or Spring)
  - Implement all CRUD operations and listing (all and per-user)
- Extend one of the clients (Angular or R and bash)
  - Angular: allow all of CRUD operations on assets and per-user listing
  - R and bash: asset adding script, printable version of asset listing and two plotting scripts