Protocol Buffer (ProtoBuf) Tutorial

Jian Li

ONOS/CORD Ambassador Steering Team, Open Networking Foundation (ONF), US

ONOS/CORD Working Group, SDN/NFV Forum, Korea

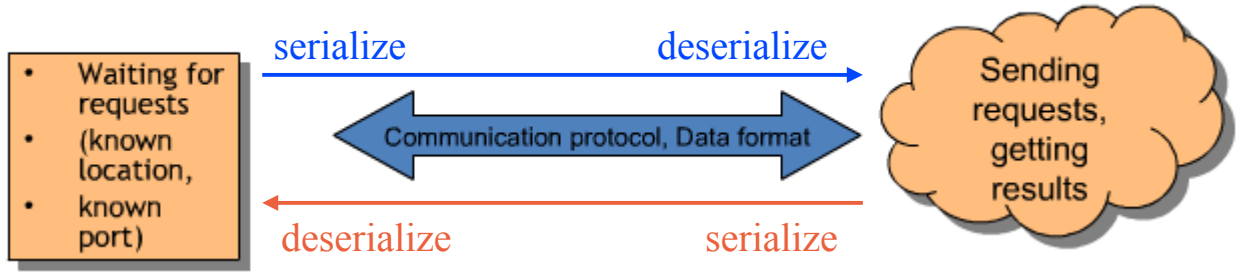
jian@{opennetworking.org, onlab.us}

Agenda

- Problem statement
- Background
- Serialization Frameworks
- Protocol Buffer

Problem Statement

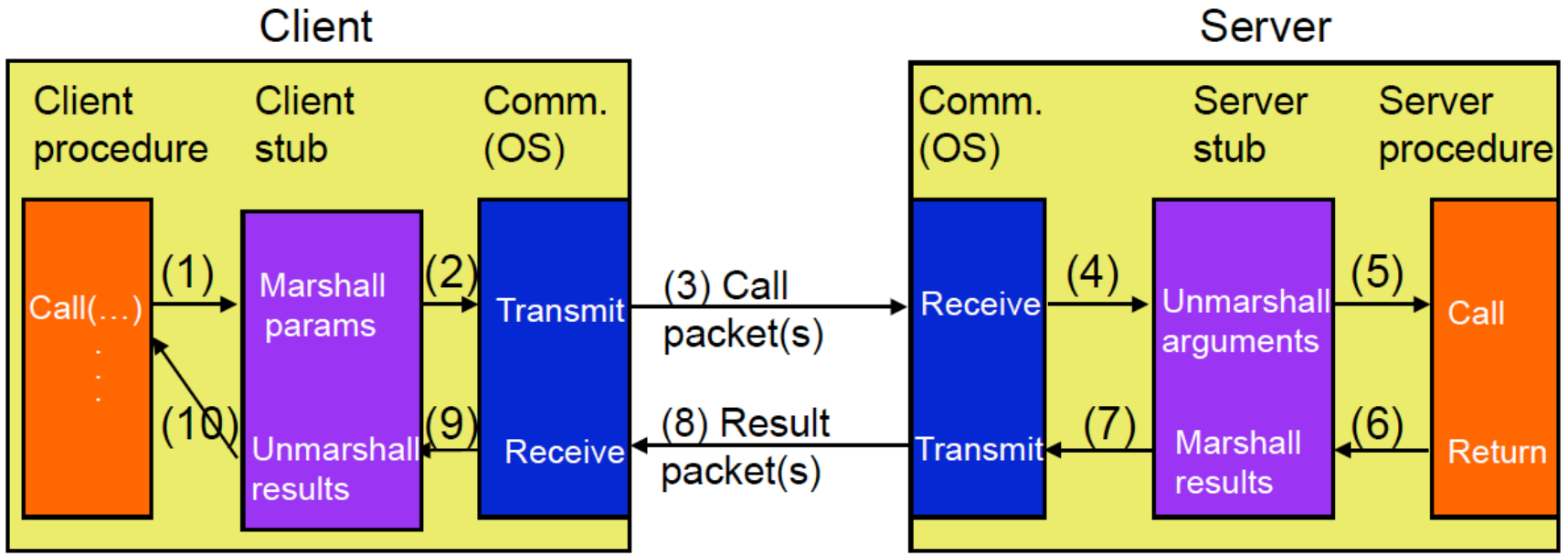
- Simple Distributed Architecture
  - What kind of protocol to use, what data to transmit?
  - Is there an efficient mechanism for storing and exchanging data?
  - What to do with requests on the server side?



- Requirements
  - Flexibility
  - Extensibility
  - Latency
  - Simplicity

Background

- Remote Procedure Call (RPC)
  - Objective: make inter-process communication among remote processes similar to local ones
  - RPC achieves its transparency in a way analogous to the read operation in a traditional single processor system



Background

- Existing RPC Technologies
  - SOAP (Simple Object Access Protocol)
    - XML
  - CORBA
    - Amazing idea! But overdesigned and heavyweight!
  - DCOM, COM+
    - Embraced mainly in windows client software
  - HTTP/JSON/XML/etc.
    - Lack of protocol description
    - Have to maintain both client and server code
    - Have to write your own wrapper to the protocol
    - XML has high parsing overhead

High Level Goals

- Transparent Interaction between Multiple Programming Languages
  - A language and platform neutral way of serializing structured data for use in communications protocols, data storage etc.

- Maintain Right Balance between
  - Efficiency
    - Time and space
  - Ease and speed of development
  - Availability of existing libraries, documentations and etc.

Consideration: Protocol Space

- Text vs. Binary
  - Binary takes less space!

{"deposit_money": "12345678"}	
JSON	Binary
'0x6d', '0x6f', '0x6e', '0x65', '0x79', '0x31', '0x32', '0x33', '0x34', '0x35', '0x36', '0x37', '0x38'	'0x01', '0xBC614E'

  - Binary is way faster!

JSON	Binary
Push down automata (PDA) parser (LL(1), LR(1)) -- 1 character lookahead. Then, final translation from characters to native types (int, float, etc)	No parser needed. The binary representation IS [as close as to] the machine representation.

Consideration: Protocol Ease of Use

- Text vs. Binary
  - Text is easier, while binary is a pain!

JSON	Binary
Brainless to learn Popular	Need to manually write code to define message packets (total pain and error prone!!!) or Use a code generator like Thrift (oh noes, I don't want to learn something new!)

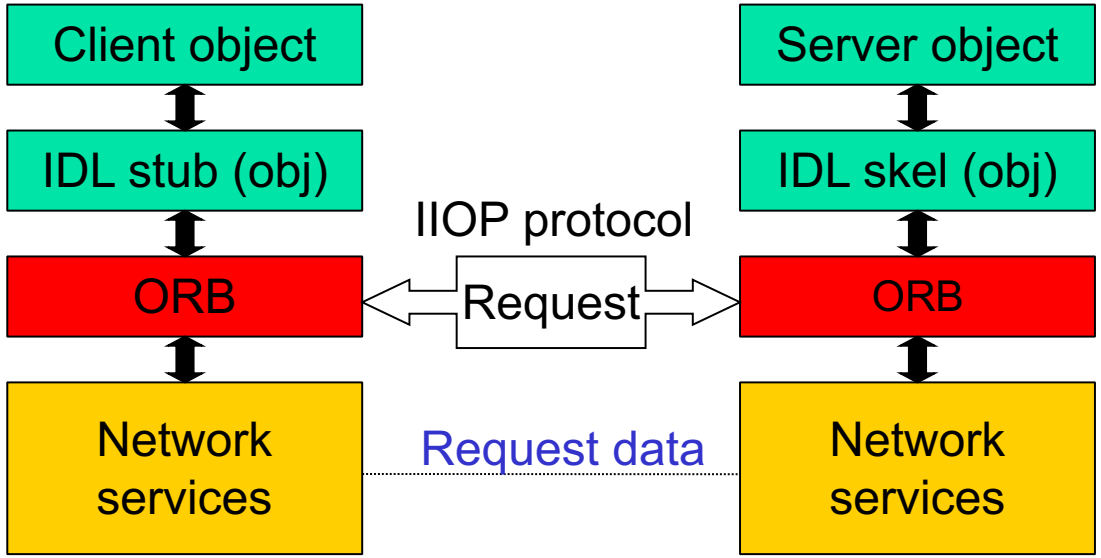
Serialization Frameworks (SF)

- Serialization Framework
  - Provides a mechanism for “translating” data models into other formats
- Popular SFs
  - XML, JSON
  - **Protocol Buffers**, BERT
  - BSON, **Apache Thrift**, Message Pack
  - Etch, Hessian, ICE, Apache Avro, etc.
- Common Properties in SFs
  - Interface Description (IDL)
  - Performance
  - Versioning
  - Binary Format

Apache Thrift ™

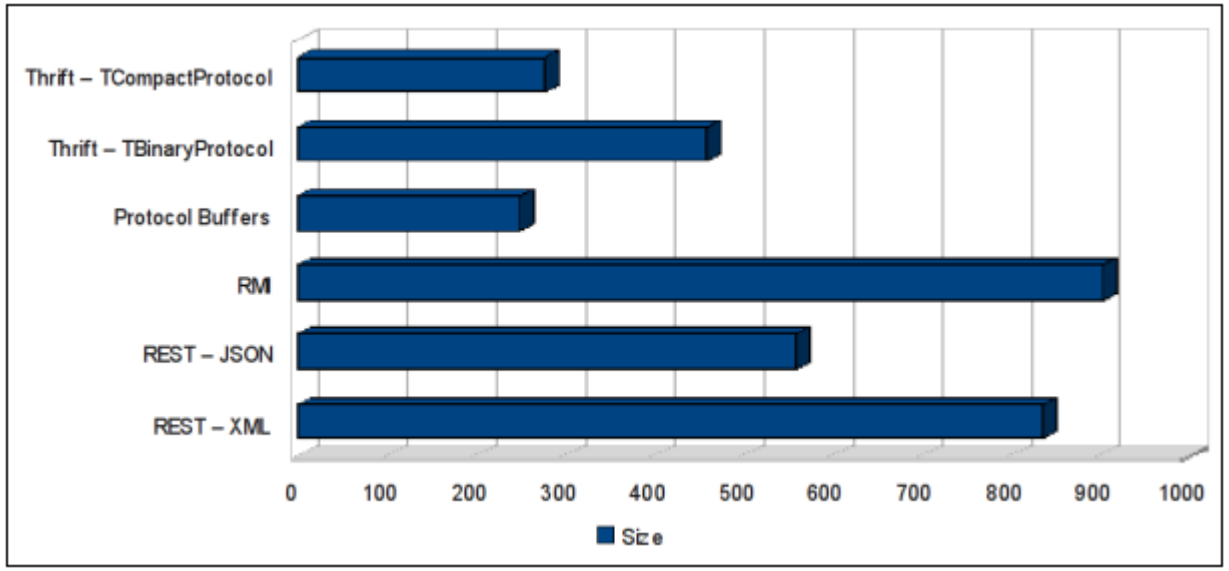
Interface Definition Language (IDL)

- Interface Definition Language
  - WS interfaces are described using Web Service Definition Language
    - E.g., SOAP, WSDL - XML based language
  - New SFs use their own languages to define the interfaces
    - Not rely on XML
  - New languages are very similar to the IDL known from CORBA



Performance Comparison (1/2)

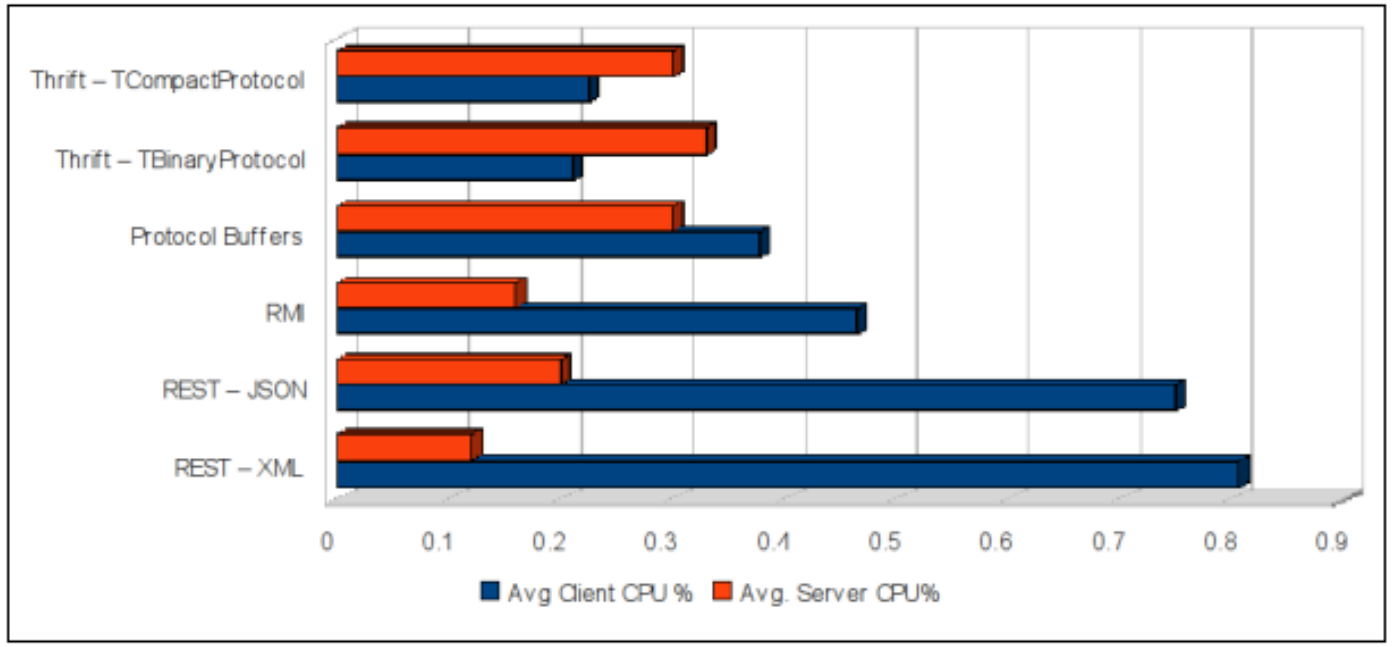
- Size
  - Each write includes one Course object with 5 Person objects, and one Phone object
    - TBinaryProtocol
      - Not optimized for space efficiency, faster to process than the text protocol, more difficult to debug
    - TCompactProtocol
      - More compact binary format; typically more efficient to process as well



RPC Type	Size (smaller is better)
Thrift — TCompactProtocol	278 (not bad)
Thrift — TBinaryProtocol	460
Protocol Buffers	**250 (winner!)**
RMI	905
REST — JSON	559
REST — XML	836

Performance Comparison (2/2)

- Runtime Performance
  - Query the list of Course numbers
  - Fetch the course for each course number
  - Executed 10k times



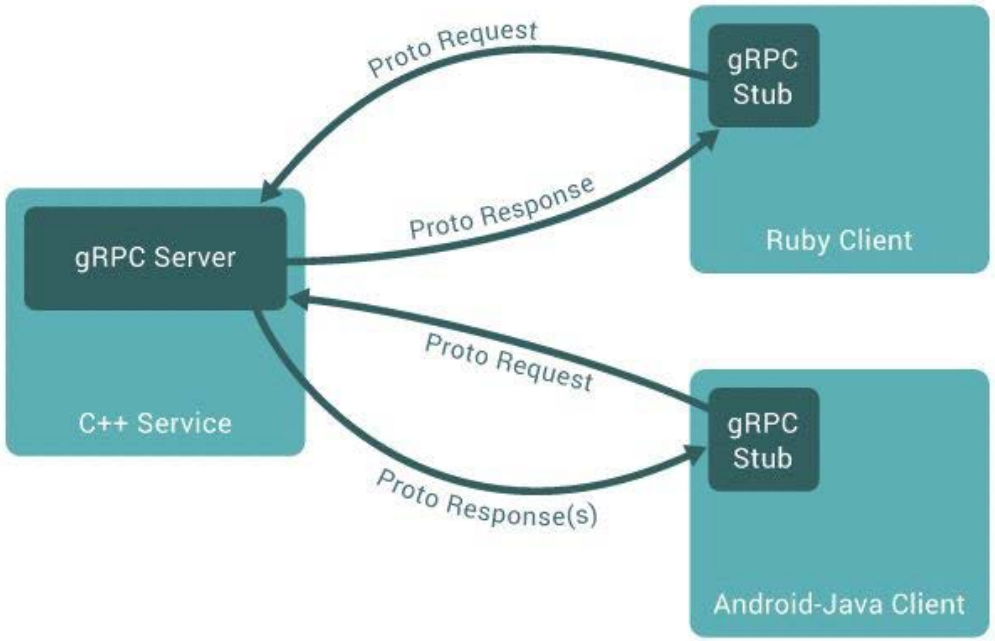
RPC Type	Server CPU %	Avg. Client CPU %	Avg. Time
REST — XML	12.00%	80.75%	05:27.45
REST — JSON	20.00%	75.00%	04:44.83
RMI	16.00%	46.50%	02:14.54
Protocol Buffers	30.00%	37.75%	01:19.48
Thrift — TBinaryProtocol	33.00%	21.00%	01:13.65
Thrift — TCompactProtocol	30.00%	22.50%	01:05.12

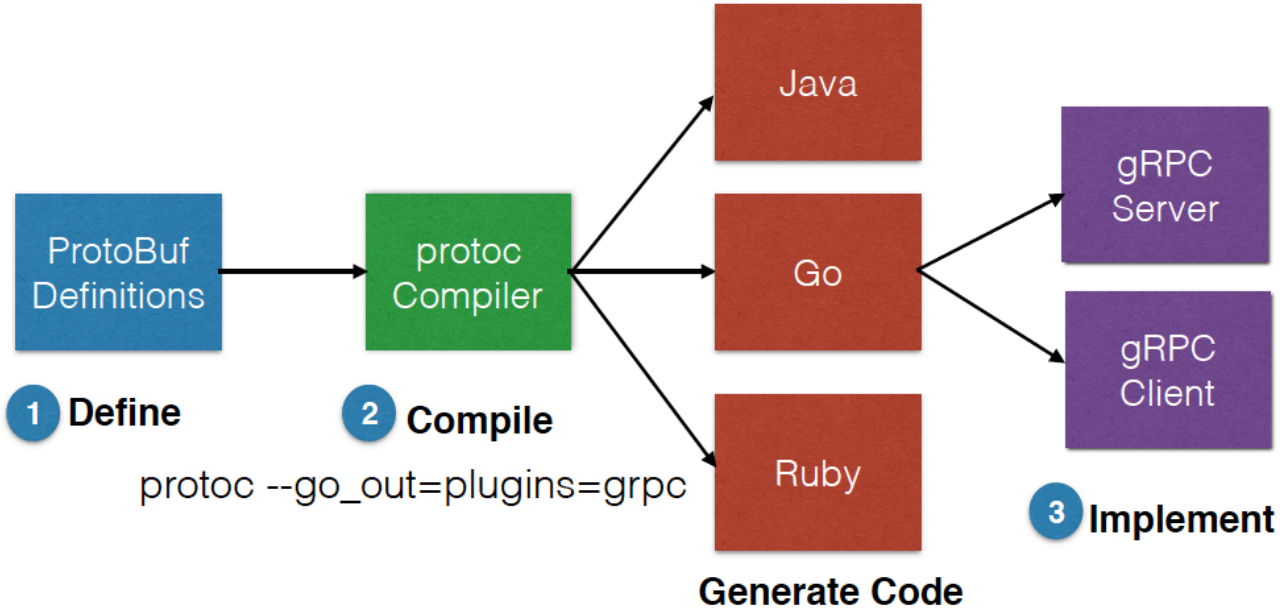
Protocol Buffer (ProtoBuf)

- What is Protocol Buffer
  - Google's language-neutral, platform-neutral, extensible mechanism for serializing structured data – think XML, but smaller, faster, and simpler
- Descriptions
  - Production, proprietary in Google from 2001-2008, open sourced since 2008
  - Very stable and well trusted
  - ProtoBuf is the glue to all Google services
  - Officially support for four languages
    - C++, Java, Python and Objective-C
  - New ProtoBuf 3 support more languages
    - Go, JavaNano, Ruby and C#
  - Simple definition as a form of IDL: portable + class or struct design
  - BSD License

ProtoBuf Work Flow

- Work Flow
  - Define IDL
    - Describe service and message exchanged between machine
  - Compile
    - Generate interfaces for multiple languages
  - Implement server and client interfaces





Define a Message Type

- Each Message May Have
  - Messages
  - Enums
  - Fields
- Specifying Field Type
  - Primitive and composite types
- Specifying Field Rules
  - Singular
    - Zero or one of this field (but not more than one)
  - Repeated
    - Repeated any # of times (incl. zero)
    - Order of the values will be preserved

```
syntax = “proto3”;
```

```
message <messageName> {
    ...
}
```

```
enum <name> {
    valueName = value;
}
```

```
<rule> <type> <name> = <id> {[<options>]};
```

Define a Message Type

- Scalar Value Types

.proto Type	Java Type	Notes
double	double	
float	float	
int32	int	Inefficient for encoding negative numbers
int64	long	Inefficient for encoding negative numbers
uint32	int	Variable length encoding
uint64	long	Variable length encoding
sint32	int	More efficiently encode negative numbers
sint64	long	More efficiently encode negative numbers

.proto Type	Java Type	Notes
fixed32	int	Four bytes encoding, efficient when encode value greater than 2^{28}
fixed64	long	Eight bytes encoding, efficient when encode value greater than 2^{56}
sfixed32	int	
sfixed64	long	
bool	boolean	boolean
string	String	UTF-8 encoding
bytes	ByteString	Any arbitrary seq. of bytes

Define a Message Type

- Assigning Tags (Field ID)
  - Unique numbered tag (e.g., 1, 2, 3, …)
  - Used to identify fields in msg binary format
  - Field names are **NOT** used in encoded data
  - Value ranges
    - 1 ~ 15: one byte
      use for very frequently occurring msg
    - 16 ~ 2047: two bytes
    - Largest: 2^{29} -1, 536,870,911
    - Cannot use 19,000 ~ 19,999, reserved for protocol buffer implementation
  - Cannot use previously reserved tags
- Commenting
  - Use C/C++ style "//" syntax

```
message Person {

  // name of person
  string name  = 1;
  // id of person
  int32  id    = 2;
  // email address
  string email = 3;

  message PhoneNumber {
    string number = 1;
  }

  repeated PhoneNumber phone = 4;
}
```

Define a Message Type

- Reused Fields can Causes Issues
  - Data corruption, privacy bugs, etc.
- Solution → Reserved Fields
  - Specify the fields tags of the removed fields as reserved
  - ProtoBuf compiler complains if users try to use these fields ID

```
message Foo {
  reserved 2, 15, 9 to 11;
  reserved “foo“, “bar“;
  reserved 12, 13, “test“; // wrong
  string name = 2;         // wrong
  string foo = 3;          // wrong
}
```

Note: cannot mix field names and tag numbers in the same reserved statement

Define an Enumeration (ENUM)

- Enumeration
  - A field that mapped to a predefined list of values
  - A constant maps to *zero* as its first element
  - Place of enum type declaration
    - Inside a message: MessageType.EnumType
    - Outside a message (.proto), can be reused in other msg
  - Alias
    - Assign the same value to different enum constants
    - allow_alias = true

```
message Person {

  ...

  enum PhoneType {
    option allow_alias = true;
    MOBILE = 0;
    HOME   = 1;
    WORK   = 2;
    JOB    = 2;
  }

  message PhoneNumber {
    PhoneType type= 2;
  }

  ...

}
```

Other Message Types

- Importing
  - Used to importing a message defined in other files
- Nested Types
  - Define and use message types inside other message types
  - Refer the nested message type
    - Parent.Type
    - E.g.;
      - Outer.MiddleAA.Inner
      - Outer.MiddleBB.Inner

```
import “myproject/other_protos.proto”
```

```
message Outer {          // Level 0
  message MiddleAA {     // Level 1
    message Inner {      // Level 2
      int64 ival = 1;
      bool booly = 2;
    }
  }
  message MiddleBB {     // Level 1
    message Inner {      // Level 2
      int32 ival = 1;
      bool booly = 2;
    }
  }
}
```

Versioning

- Definition
  - System must be able to support reading of old data and requests from out-of-date clients to new servers and vice versa

- State-of-the-art
  - Versioning in Thrift and ProtoBuf is implemented via field identifiers
  - The combination of the field identifiers and its type specifier is used to uniquely identify the field
  - A new compilation is NOT necessary
  - Statically typed system (i.e., CORBA and RMI) would require an update of all clients

Forward and Backward Compatibility (1/2)

- Four Cases
  - Added field: old client ←→ new server
  - Removed field: old client ←→ new server
  - Added field: new client ←→ old server
  - Removed field: new client ←→ old server

- No Field Changes
  - Client sends a message to a server
  - Operated properly without any issue!

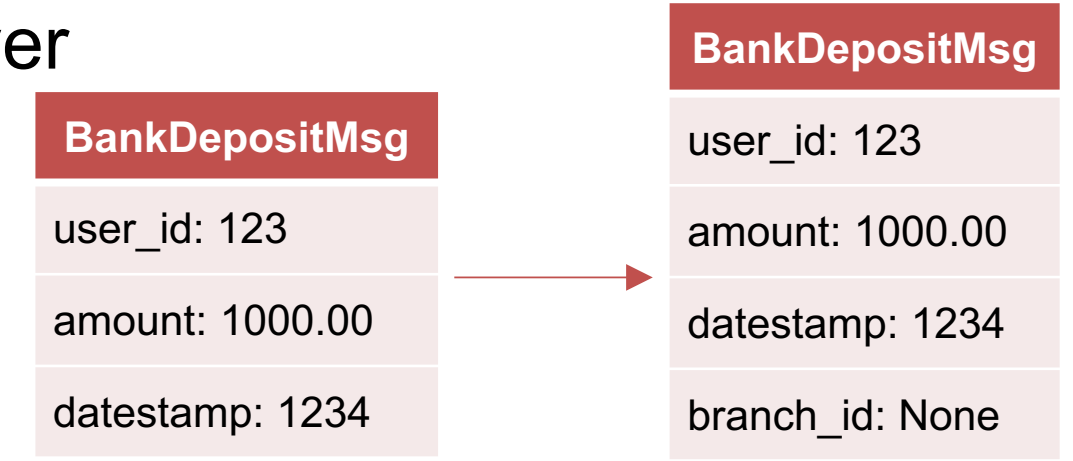
BankDepositMsg
user_id: 123
amount: 1000.00
datestamp: 82912323

→

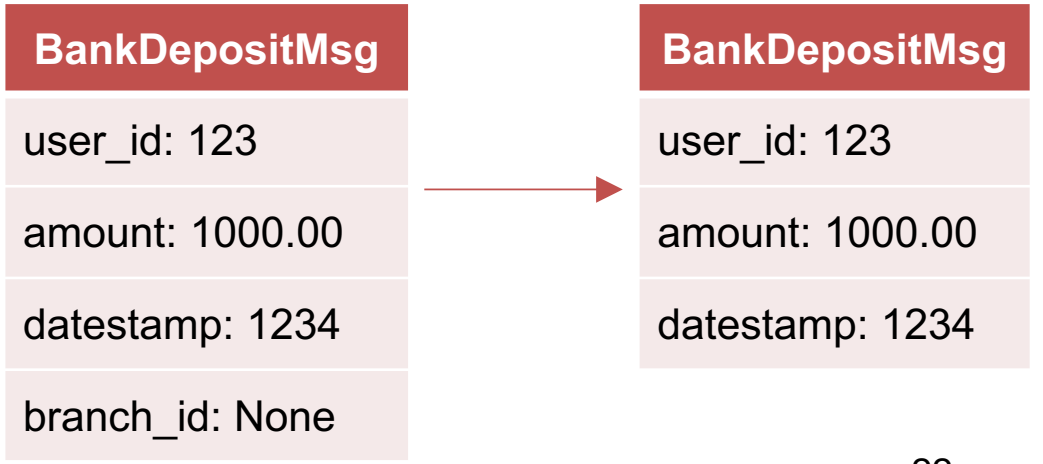
BankDepositMsg
user_id: 123
amount: 1000.00
datestamp: 82912323

Forward and Backward Compatibility (2/2)

- Added Field: Old Client → New Server
- Removed Field: New Client → Old Server
  - Server recognizes the field is not set, and implements default behavior for Out-Of-Date requests



- Added Field: New Client → Old Server
- Removed Field: Old Client → New Server
  - The server simply ignores the newly added fields and processes as normal



Update a Message Type

- Rules for Defining New Fields
  - As repeated or optional
  - Set sensible default values
  - Do NOT change tags/ids
  - Do NOT recycle tags/ids
    - Add the prefix “OBSOLETE_”, or make the tag reserved
  - Change between compatible types
    - int32, uint32, int64, unit64
    - sint32 and sint64 are compatible with each other but are not compatible with the others
    - string and bytes are compatible as long as the bytes are valid UTF-8
    - fixed32 is compatible with sfixed32; fixed64 is compatible with sfixed64
  - Unknown fields
    - Fields that the parser does not recognize
    - Old parser treats newly added fields as unknown fields

Any and Oneof Type

- Any
  - Allow to use messages as embedded types without having their .proto def.
  - Contain an arbitrary serialized message as bytes, along with a URL
    - URL acts as a globally unique identifier
  - Need to import “google/protobuf/any.proto”
- Oneof
  - At most one field can be set at the same time
  - Only the last field you set will still have a value
  - An oneof cannot be repeated

```
import "google/protobuf/any.proto";

message ErrorStatus {
  string message = 1;
  repeated google.protobuf.Any
           details = 2;
}
```

```
message SampleMessage {
  oneof test_oneof {
    string name = 4;
    SubMessage sub_message = 9;
  }
}
```

```
SampleMessage message;
message.set_name("name");
CHECK(message.has_name());
message.mutable_sub_message();
CHECK(!message.has_name());
```

Maps

- Maps
  - Key, value paired data type
  - key_type: integral or string type
  - value_type: any type except another map
    - Nested map is **NOT** permitted
  - Map fields cannot be repeated
  - Ordering of map values is undefined
  - For duplicated map key
    - From wire: only the last key seen is used
      From text format: parsing may fail
  - Backwards compatibility
    - Map syntax is equivalent to combination of key and value message type that was annotated with repeated notation

```
map<key_type, value_type> map_field = N;
```

```
message MapFieldEntry {
  key_type key = 1;
  value_type value = 2;
}

repeated MapFieldEntry map_field = N;
```

Defining Services

- Define an RPC Service Interface
  - Inside a .proto file
  - gRPC → the most straightforward RPC system that uses ProtoBuf
  - Comprise of an input and an output interface

```
service <serviceName> {
    rpc stubName1 (inputMessageName1) returns (outputMessageName1);
    rpc stubName2 (inputMessageName2) returns (outputMessageName2);
    ...
}
```

  - Drawback
    - Do NOT support method overloading
      - We can only specify a message name for each service interface as an input

Packages

- Packages
  - Prevent name clashes between protocol message types
  - The package name will also be used as the Java package
  - To customize the Java package name we need to use `java_package` option
- Name Resolution
  - Name resolution works like C++
  - The innermost scope is searched, then the next-innermost scope is searched, and so on
  - Compiler resolves all type names by parsing the imported .proto files
  - The code generator knows how to refer to each type in that language

```
package foo.bar;
message Open { ... }
```

```
message Foo {
  ...
  foo.bar.Open
  open = 1;
  ...
}
```

Options

- Options
  - Individual declarations in a .proto file can be annotated with a # of options
  - Options do **NOT** change the overall meaning of a declaration
  - Java specific options
    - option java_package = “package.name”;
    - option java_multiple_files = true;
    - option java_outer_classname = “outer.class.name”;
  - Commonly used options
    - [default = value] → sets a default value (default values are not encoded!)
    - [packed = **false**/true] → better encoding of repeated
    - [deprecated = **false**/true] → marks a field as obsolete
    - [optimize_for = **SPEED**/CODE/LITE_RUNTIME]

Java Code Generation (1/10)

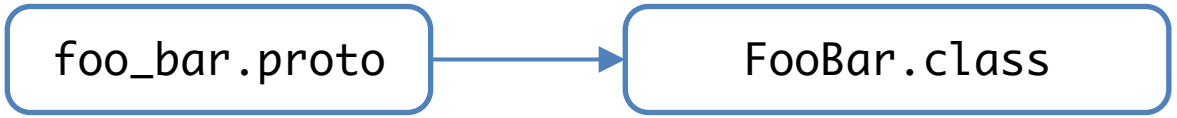
- Compiler Invocation
  - ***--java_out=*** option is the directory where you want to write your Java output
  - Compiler creates a single .java for each .proto file input
    - Contains a single outer class and several nested classes and static fields

```
protoc --proto_path=IMPORT_PATH --java_out=DST_DIR path/to/file.proto
```

  - Compiler can output directly to JAR archives

```
protoc --proto_path=IMPORT_PATH --java_out=DST_DIR/abc.jar path/to/file.proto
```

  - Outer class name
    - Can be specified using **`java_outer_classname`** option
    - Otherwise, the name is determined by converting the .proto file name to camel case



Java Code Generation (2/10)

- Messages
  - Optimized for SPEED
    - Overrides many as many methods as possible
    - Maximize the LOC with detailed impl.
  - Optimized for CODE_SIZE
    - Overrides only the minimum set of methods necessary to function
    - Relies on GeneratedMessage's reflection-based impl.
  - Optimized for LITE_RUNTIME
    - Includes fast impl. of all methods
    - Implements MessageLite interface
    - Appropriate for resource-constrained systems such as mobile phones

```
Message Foo {}
```

```
public final class Foo extends GeneratedMessage {

  // a set of accessor methods
  public XXX getXXX() { ... }
  public final class Builder extends
                        GeneratedMessage.Builder {

    // a set of accessor methods
    public XXX getXXX() { ... }
    ...
    // a set of modification methods
    public Builder setXXX() { ... }
    ...
    // build method to spawn an object
    public Foo build() { ... }
  }
}
```

Java Code Generation (3/10)

- Messages
  - Builders
    - Allow all instances of the Message class are immutable
    - All setter methods are chained using builder
    - Once the object is instantiated, there is no way to set the attribute of the instance

```
builder.mergeFrom(obj).setFoo(1).setBar("abc").clearBaz();
```

  - Sub builders
    - Compiler also generates sub builders for sub-messages

```
baz =
baz.toBuilder().setBar( baz.getBar().toBuilder().setFoo( baz.getBar
().getFoo().toBuilder().setVal(10).build() ).build()).build();
```

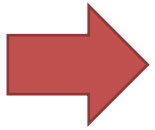
```
Baz.Builder builder = baz.toBuilder();
builder.getBarBuilder().getFooBuilder().setVal(10);
baz = builder.build();
```

```
message Foo {
  int32 val = 1;
}

message Bar {
  Foo foo = 1;
}

message Baz {
  Bar bar = 1;
}
```

Java Code Generation (4/10)

- Fields
  - Generates a set of accessor methods for each field
  - Methods that read the field value are defined both in msg and its corresponding builder classes
  - Methods that modify the value are defined in the builder only
  - Method names always use camel-case naming
  - Singular fields (proto3)

```
message Foo {
  int32 foo = 1;
}
```

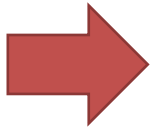


```
public final class Foo extends GeneratedMessage {
  public int getFoo() { ... }
  public Builder getFooBuilder() { ... }
  public final class Builder extends GeneratedMessage.Builder {
    // same accessor methods in message class
    Builder setFoo(int value) { ... }
    Builder clearFoo() { ... }
    Foo build() { ... }
  }
}
```

Java Code Generation (5/10)

- Fields
  - Enum fields

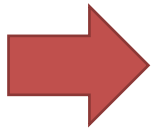
```
message Foo {
  Bar bar = 1;
}
```



```
public final class Foo extends GeneratedMessage {
...
  public int getBarValue() { ... }
  public final class Builder extends GeneratedMessage.Builder {
    // same accessor methods in message class
    Builder setFooValue(int value) { ... }
...
```

  - Repeated fields

```
message Foo {
  repeated int32 foo = 1;
}
```

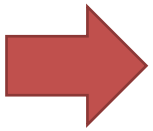


```
public final class Foo extends GeneratedMessage {
  public int getFooCount() { ... }
  public int getFoo(int index) { ... }
  public List<Integer> getFooList() { ... }
  public final class Builder extends GeneratedMessage.Builder {
    // same accessor methods in message class
  }
}
```

Java Code Generation (6/10)

- Fields
  - Repeated fields

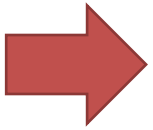
```
message Foo {
  repeated int32 foo = 1;
}
```



```
public final class Foo extends GeneratedMessage {
  ...
  public final class Builder extends GeneratedMessage.Builder {
    ...
    Builder setFoo(int index, int value) { ... }
    Builder addFoo(int value) { ... }
    Builder addAllFoo(Iterable<? Extends Integer> value) { ... }
    Builder clearFoo() { ... }
    ...
  }
}
```

  - Repeated enum fields

```
message Foo {
  repeated Bar bar = 1;
}
```

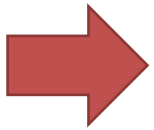


```
public final class Foo extends GeneratedMessage {
  public int getBarValue(int index) { ... }
  public List getBarValueList() { ... }
  public final class Builder extends GeneratedMessage.Builder {
    // same accessor methods in message class
    Builder setBarValue(int index, int value) { ... }
    ...
```

Java Code Generation (7/10)

- Fields
  - Map fields
    - getXXXorDefault: returns the value for key or the default value if not present
    - getXXXorThrow: returns the value for key or throws IlliegalArgumentException if not present
    - Map<Integer, Integer> getMutableWeight() → returns a mutable Map (deprecated)

```
message Foo {
  map<int32, int32> weight= 1;
}
```

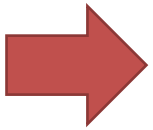


```
public final class Foo extends GeneratedMessage {
  public Map<Integer, Integer> getWeightMap() { ... }
  public int getWeightOrDefault(int key, int default) { ... }
  public int getWeightOrThrow(int key) { ... }
  public boolean containsWeight(int key) { ... }
  public int getWeightCount() { ... }

  public final class Builder extends GeneratedMessage.Builder {
    // same accessor methods in message class
    Builder putWeight(int key, int value) { ... }
    Builder putAllWeight(Map<Integer, Integer> value) { ... }
    Builder removeWeight(int key) { ... }
    Builder clearWeight() { ... }
    ...
  }
}
```

Java Code Generation (8/10)

- Enumerations
  - Compiler generates a Java enum type with the same set of values
  - For proto3, compiler also addes **UNRECOGNIZED** value to the enum type

```
enum Bar {
  VALUE_A = 0;
  VALUE_B = 5;
  VALUE_C = 5;
}
```



```
public final class Foo extends GeneratedMessage {
  ...
  public enum Bar implements ProtocolMessageEnum {
    VALUE_A(0),
    VALUE_B(5),
    UNRECOGNIZED(-1),;
  }
  public static final Bar VALUE_C = VALUE_B;

  public int getNumber() { ... }
  public EnumValueDescriptor getValueDescriptor() { ... }
  public EnumDescriptor getDescriptorForType() { ... }

  public static Bar forNumber(int value) { ... }
  public static Bar valueOf(int value) { ... }
  public static Bar valueOf(EnumValueDescriptor descriptor) { ... }
  public static EnumDescriptor getDescriptor() { ... }
}
```

Java Code Generation (9/10)

- Services
  - Interface
    - Compiler generates RPC service code based on service definitions

```
service Foo {
  rpc Bar(FooRequest) returns (FooResponse);
}
```

```
public final class FooProviderServiceRpc {
  ...
  private FooProviderServiceRpc
  public static abstract class FooProviderRegistryRpcImplBase implements BindableService {

    // this method should be implemented by developers
    public FooResponse bar(FooRequest request, FooResponse response) {
        ...
    }
    public final ServerServiceDefinition bindService() { ... } // fine with the default impl.
  }
}
```

Java Code Generation (10/10)

- Services
  - Stub
    - Used by clients wishing to send requests to servers implementing the service
    - Blocking stub
    - Future stub

```
public final class FooProviderServiceRpc {
  ...
  public static final class FooProviderServiceRpcStub extends AbstractStub<?> {
    ...
    public FooResponse bar(FooRequest request, StreamObserver<FooResponse> responseObserver) { ... }
  }
  public static final class FooProviderServiceRpcBlockingStub extends AbstractStub<?> {
    ...
    public FooResponse bar(FooRequest request) { ... }
  }
  public static final class FooProviderServiceRpcFutureStub extends AbstractStub<?> {
    ...
    public ListenableFuture<FooResponse> bar(FooRequest request, FooResponse response) { ... }
  }
}
```

References

- ProtoBuf language guide: https://developers.google.com/protocol-buffers/docs/proto3
- Reference for Java Generated Code: https://developers.google.com/protocol-buffers/docs/reference/java-generated
- PB vs. Thrift vs. Avro
- Protocol Buffer Overview
- Distributed System