Intro to ZMQ and Google Protobuf in C++

@diegorlosada

@t3chfest C3M Feb-2015

Serialization

- JSON, XML, CSV…
  - Parsing efficiency (vs human readable)
  - Robust to changes
    - Adding fields
    - Removing fields? Ignoring fields
  - SW engineering possible:
    - Composable, hierarchies…
- Necessary for:
  - Persistence
  - Transport (network)
    - RPC/ distributed systems

Protocol Buffers (Protobuf)

- Serialization format by Google
  used by Google for almost all internal RPC protoc ols and file formats
  - (currently 48,162 different message types defined in t he Google code tree across 12,183 .proto files. They're used both in RPC systems and for persistent storage o f data in a variety of storage systems.)
- XML vs Protobuf
  - Speed 20-100x
  - Size 10x
  - Not human readable

Protocol Buffers (Protobuf)

- Define .proto
- Compile with “protoc” (different languages)
- Lets do it!

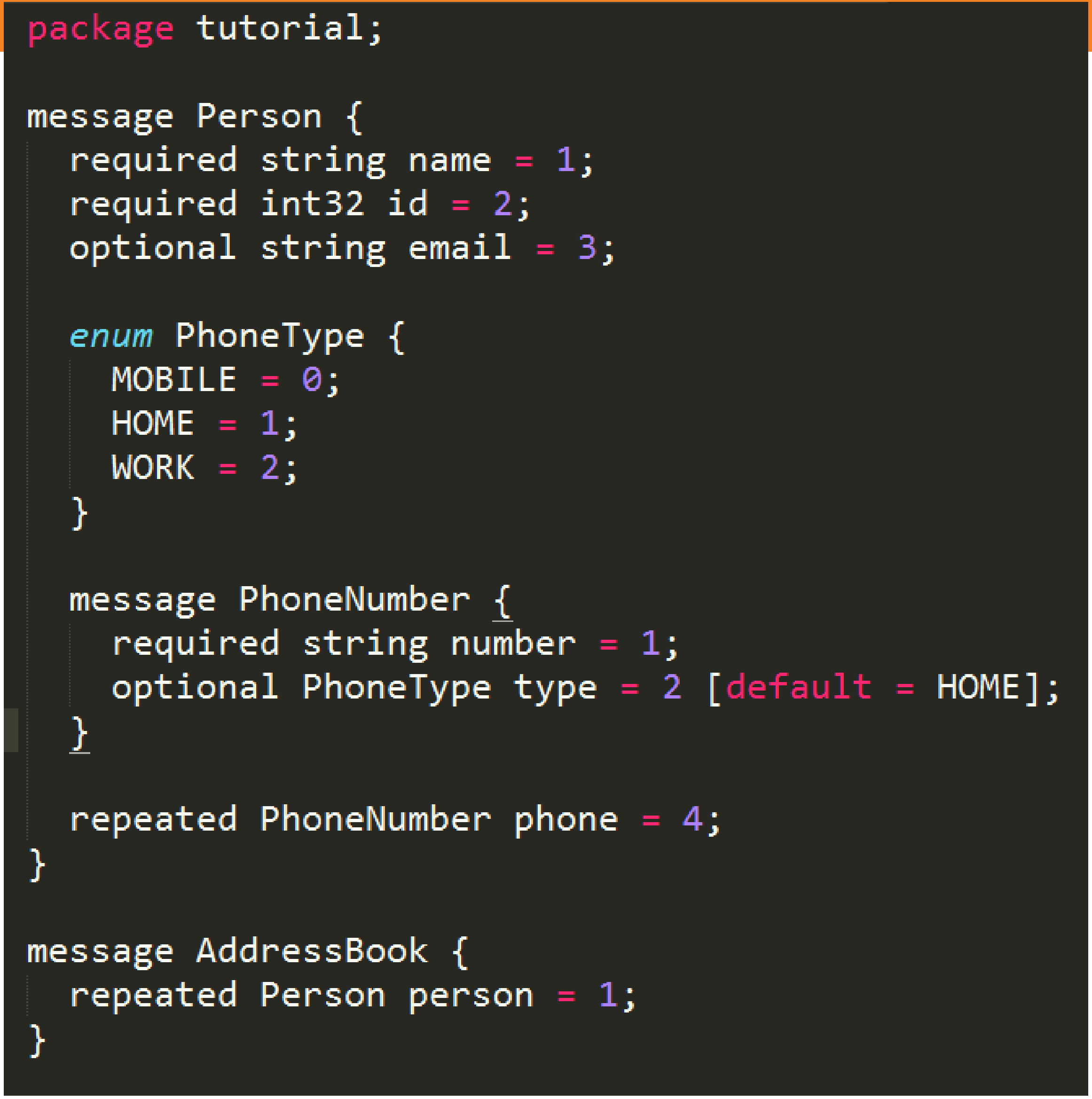
```
$ bii init myproject
$ cd myproject
$ bii open examples/protobuf
$ bii cpp:build
```

OR

http://www.biicode.com/examples/protobuf

Protocol Buffers IDL

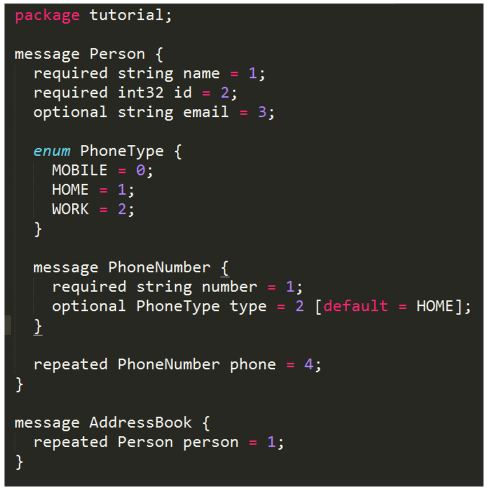
message.proto



```
package tutorial;

message Person {
  required string name = 1;
  required int32 id = 2;
  optional string email = 3;

  enum PhoneType {
    MOBILE = 0;
    HOME = 1;
    WORK = 2;
  }

  message PhoneNumber {
    required string number = 1;
    optional PhoneType type = 2 [default = HOME];
  }

  repeated PhoneNumber phone = 4;
}

message AddressBook {
  repeated Person person = 1;
}
```

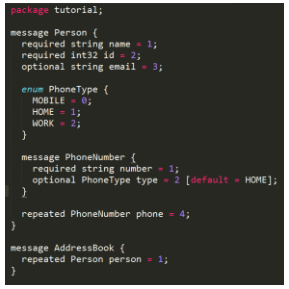
Generate code

message.proto



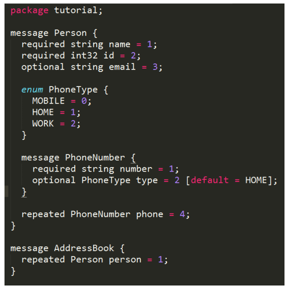
```
package tutorial;

message Person {
  required string name = 1;
  required int32 id = 2;
  optional string email = 3;

  enum PhoneType {
    MOBILE = 0;
    HOME = 1;
    WORK = 2;
  }

  message PhoneNumber {
    required string number = 1;
    optional PhoneType type = 2 [default = HOME];
  }

  repeated PhoneNumber phone = 4;
}

message AddressBook {
  repeated Person person = 1;
}
```

message.pb.h



```
package tutorial;

message Person {
  required string name = 1;
  required int32 id = 2;
  optional string email = 3;

  enum PhoneType {
    MOBILE = 0;
    HOME = 1;
    WORK = 2;
  }

  message PhoneNumber {
    required string number = 1;
    optional PhoneType type = 2 [default = HOME];
  }

  repeated PhoneNumber phone = 4;
}

message AddressBook {
  repeated Person person = 1;
}
```

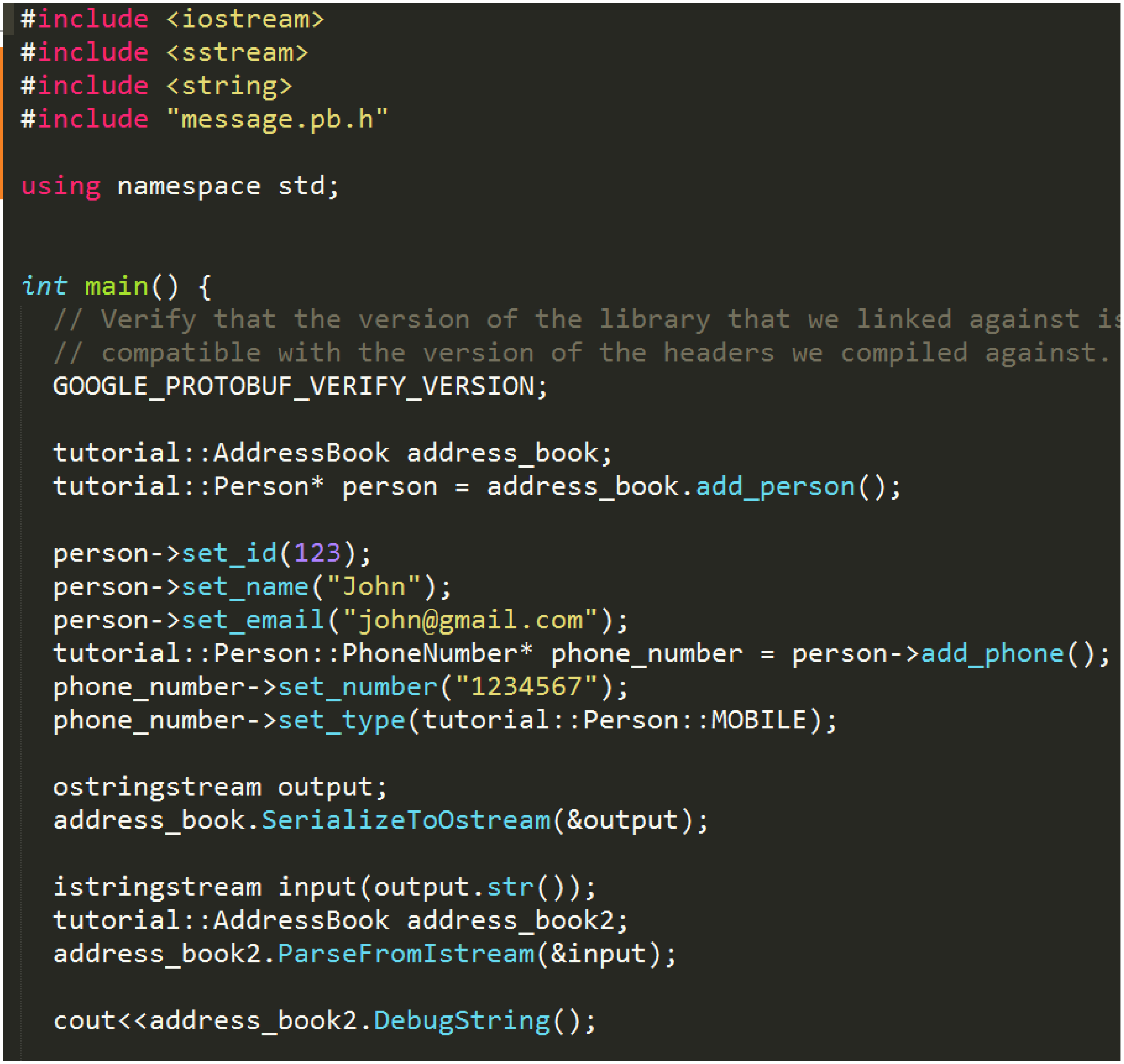
message.pb.cc



```
package tutorial;

message Person {
  required string name = 1;
  required int32 id = 2;
  optional string email = 3;

  enum PhoneType {
    MOBILE = 0;
    HOME = 1;
    WORK = 2;
  }

  message PhoneNumber {
    required string number = 1;
    optional PhoneType type = 2 [default = HOME];
  }

  repeated PhoneNumber phone = 4;
}

message AddressBook {
  repeated Person person = 1;
}
```

```
$ protoc message.proto --cpp_out=“.”
```

Use



```cpp
#include <iostream>
#include <sstream>
#include <string>
#include "message.pb.h"

using namespace std;


int main() {
  // Verify that the version of the library that we linked against is
  // compatible with the version of the headers we compiled against.
  GOOGLE_PROTOBUF_VERIFY_VERSION;

  tutorial::AddressBook address_book;
  tutorial::Person* person = address_book.add_person();

  person->set_id(123);
  person->set_name("John");
  person->set_email("john@gmail.com");
  tutorial::Person::PhoneNumber* phone_number = person->add_phone();
  phone_number->set_number("1234567");
  phone_number->set_type(tutorial::Person::MOBILE);

  ostringstream output;
  address_book.SerializeToOstream(&output);

  istringstream input(output.str());
  tutorial::AddressBook address_book2;
  address_book2.ParseFromIstream(&input);

  cout<<address_book2.DebugString();
```

Protocol Buffers IDL

- bool SerializeToString(string* output) const;
- bool ParseFromString(const string& data);
- bool SerializeToOstream(ostream* output) const;
- bool ParseFromIstream(istream* input);

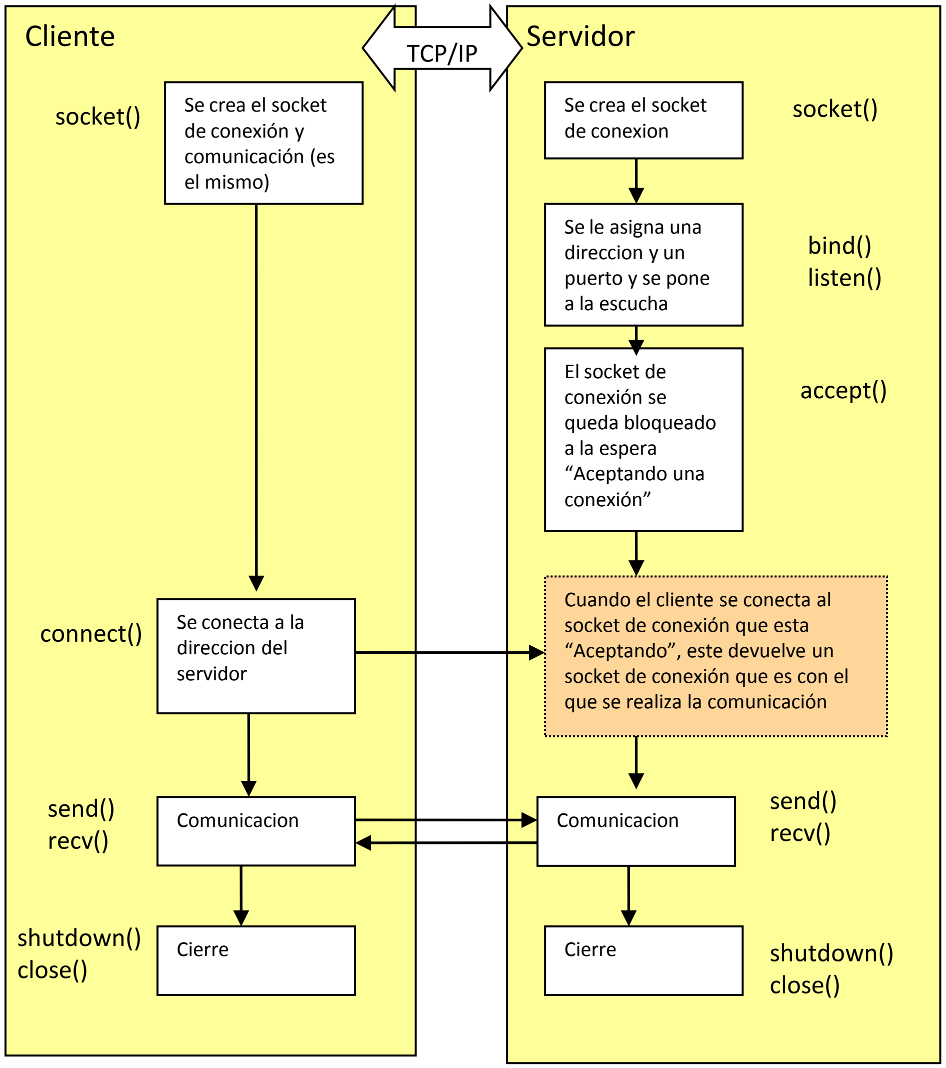
ZeroMQ

High level messaging: RabbitMQ

ØMQ

Low level sockets

Raw Sockets

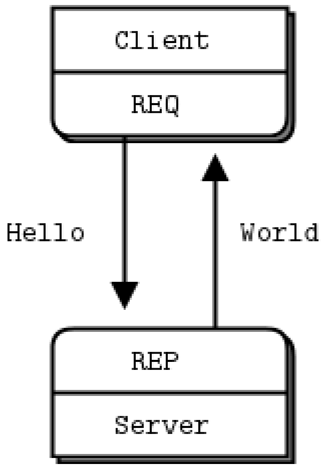


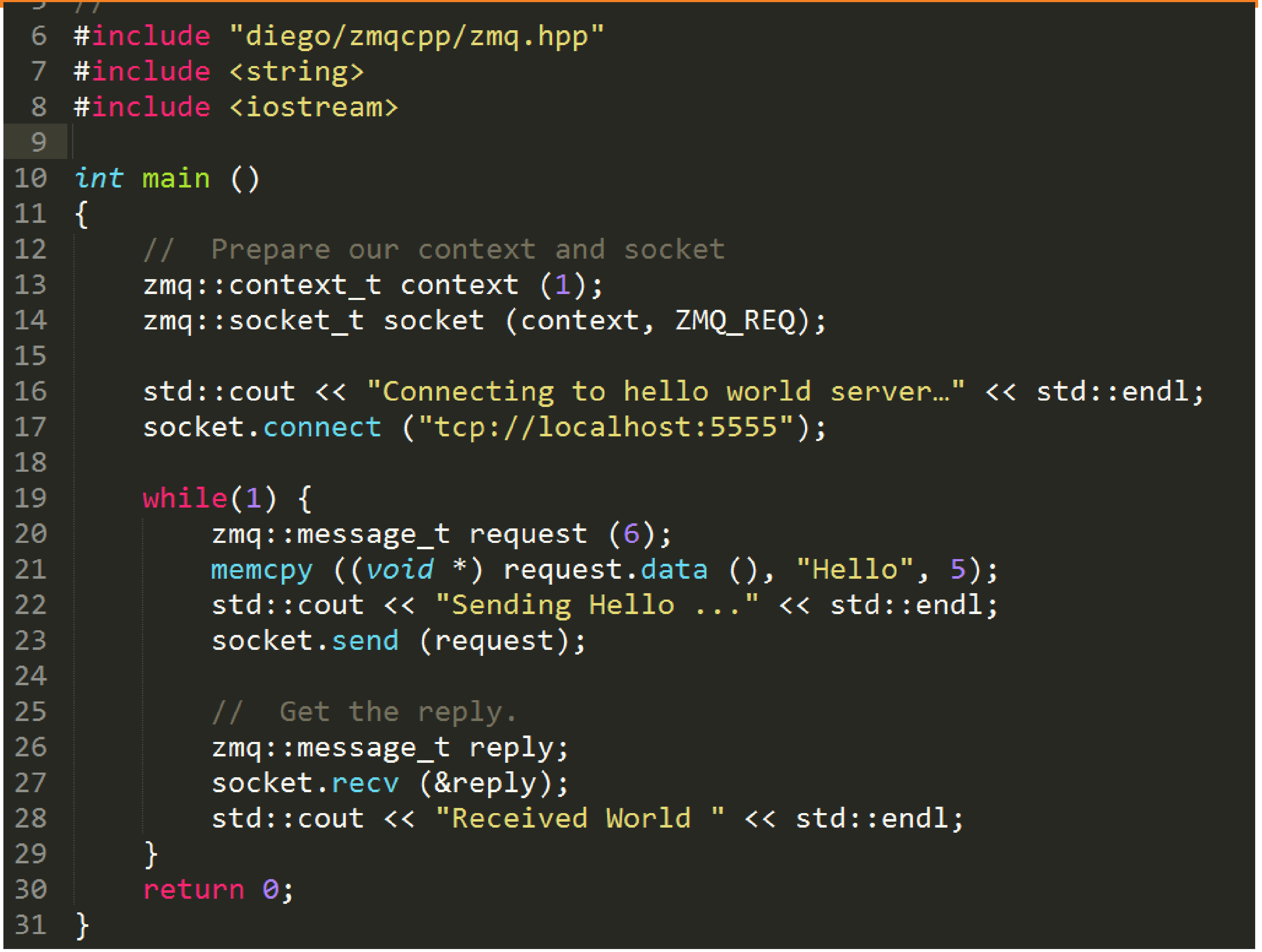
Cliente
TCP/IP
Servidor
socket()
Se crea el socket de conexión y comunicación (es el mismo)
connect()
Se conecta a la direccion del servidor
send()
recv()
Comunicacion
shutdown()
close()
Cierre
Se crea el socket de conexion
socket()
Se le asigna una direccion y un puerto y se pone a la escucha
bind()
listen()
El socket de conexión se queda bloqueado a la espera “Aceptando una conexión”
accept()
Cuando el cliente se conecta al socket de conexión que esta “Aceptando”, este devuelve un socket de conexión que es con el que se realiza la comunicación
Comunicacion
send()
recv()
Cierre
shutdown()
close()

ZeroMQ

– No neutral carrier
– No protocol

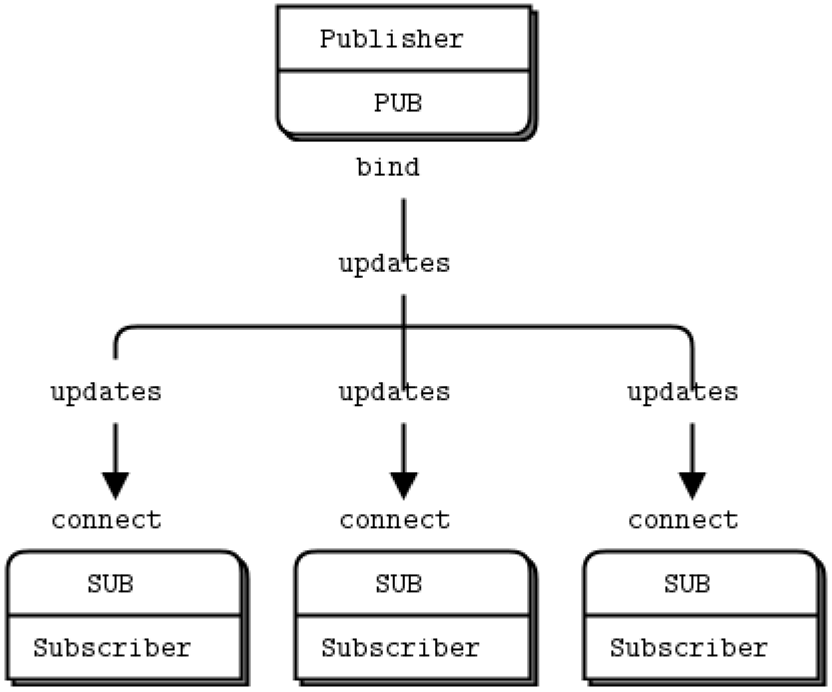
REQ-REP => RPC

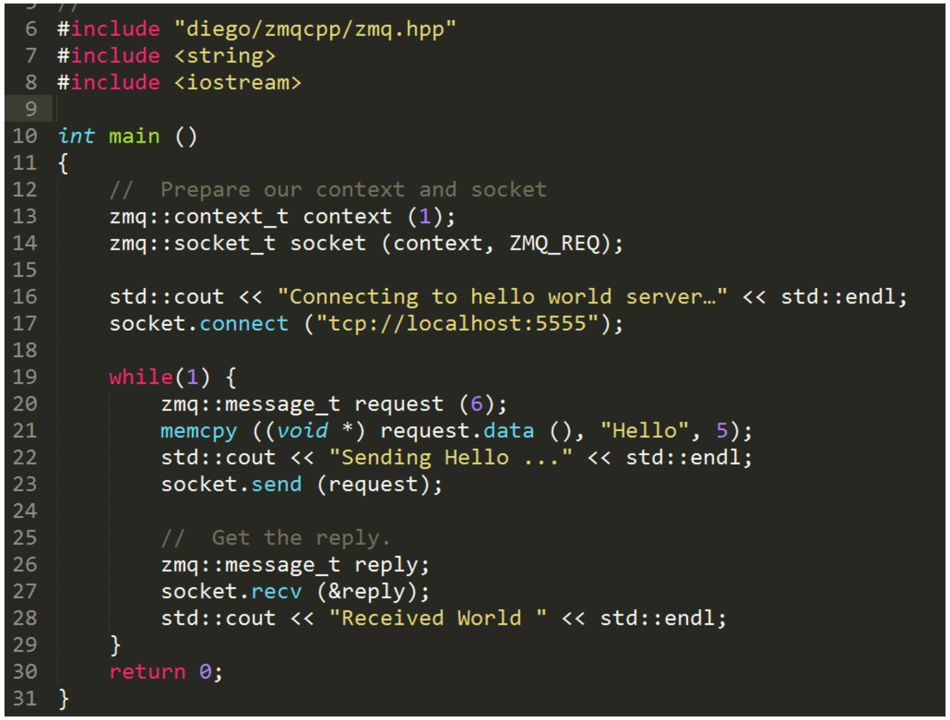




```cpp
//
#include "diego/zmqcpp/zmq.hpp"
#include <string>
#include <iostream>

int main ()
{
    //  Prepare our context and socket
    zmq::context_t context (1);
    zmq::socket_t socket (context, ZMQ_REQ);

    std::cout << "Connecting to hello world server…" << std::endl;
    socket.connect ("tcp://localhost:5555");

    while(1) {
        zmq::message_t request (6);
        memcpy ((void *) request.data (), "Hello", 5);
        std::cout << "Sending Hello ..." << std::endl;
        socket.send (request);

        //  Get the reply.
        zmq::message_t reply;
        socket.recv (&reply);
        std::cout << "Received World " << std::endl;
    }
    return 0;
}
```

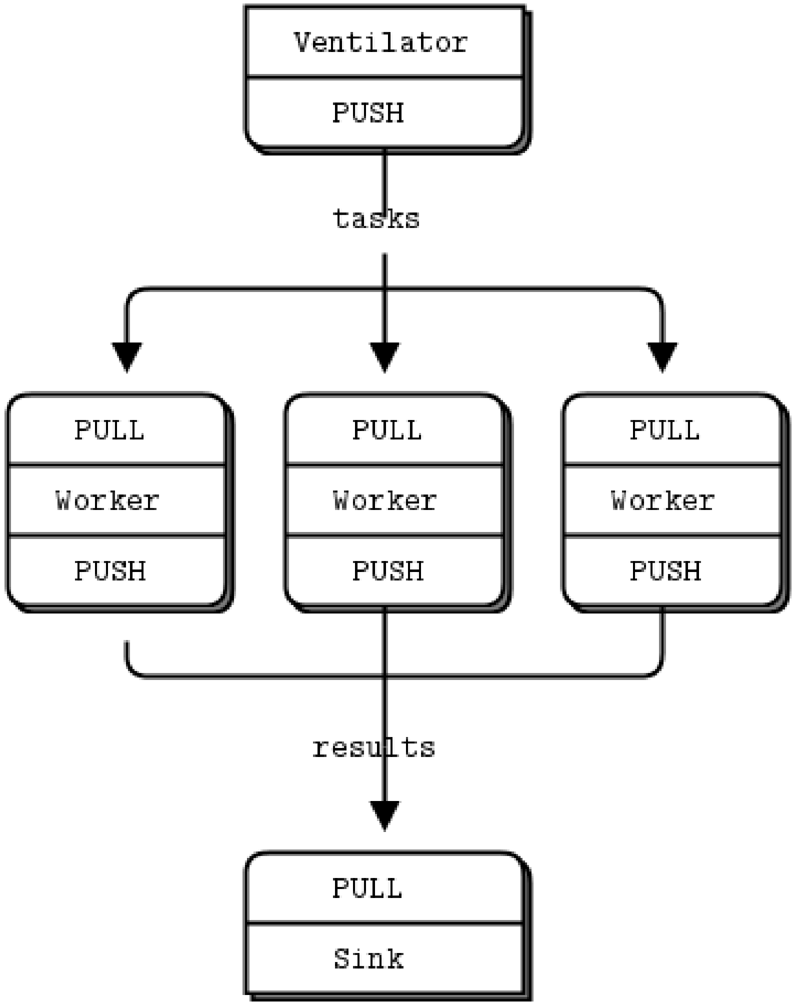
PUB-SUB => Broadcast

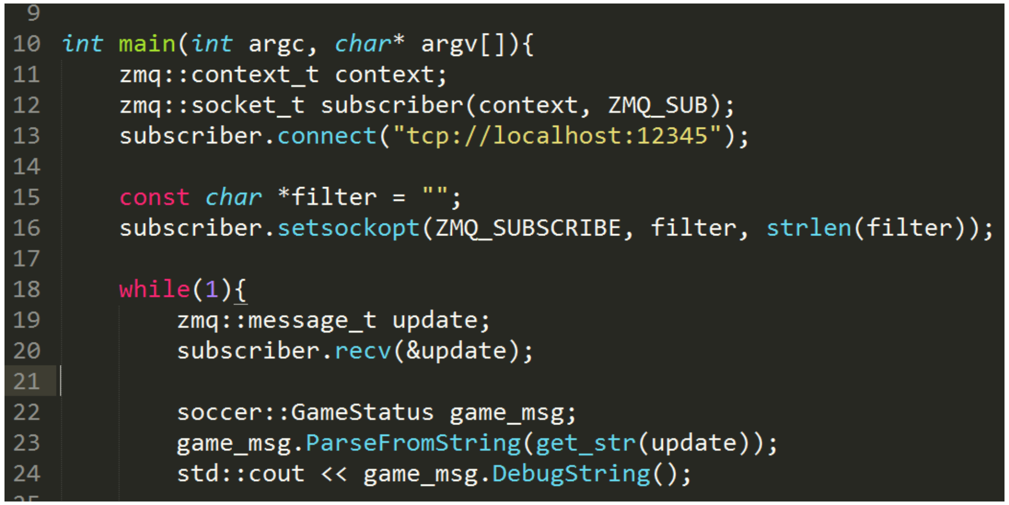




```
#include "diego/zmqcpp/zmq.hpp"
#include <string>
#include <iostream>

int main ()
{
    //  Prepare our context and socket
    zmq::context_t context (1);
    zmq::socket_t socket (context, ZMQ_REQ);

    std::cout << "Connecting to hello world server…" << std::endl;
    socket.connect ("tcp://localhost:5555");

    while(1) {
        zmq::message_t request (6);
        memcpy ((void *) request.data (), "Hello", 5);
        std::cout << "Sending Hello ..." << std::endl;
        socket.send (request);

        //  Get the reply.
        zmq::message_t reply;
        socket.recv (&reply);
        std::cout << "Received World " << std::endl;
    }
    return 0;
}
```

PUSH-PULL => Load Balance, Fair Queueing





```
int main(int argc, char* argv[]){
    zmq::context_t context;
    zmq::socket_t subscriber(context, ZMQ_SUB);
    subscriber.connect("tcp://localhost:12345");

    const char *filter = "";
    subscriber.setsockopt(ZMQ_SUBSCRIBE, filter, strlen(filter));

    while(1){
        zmq::message_t update;
        subscriber.recv(&update);

        soccer::GameStatus game_msg;
        game_msg.ParseFromString(get_str(update));
        std::cout << game_msg.DebugString();
```

Build sha password distributed analyzer

SOCCER TIME

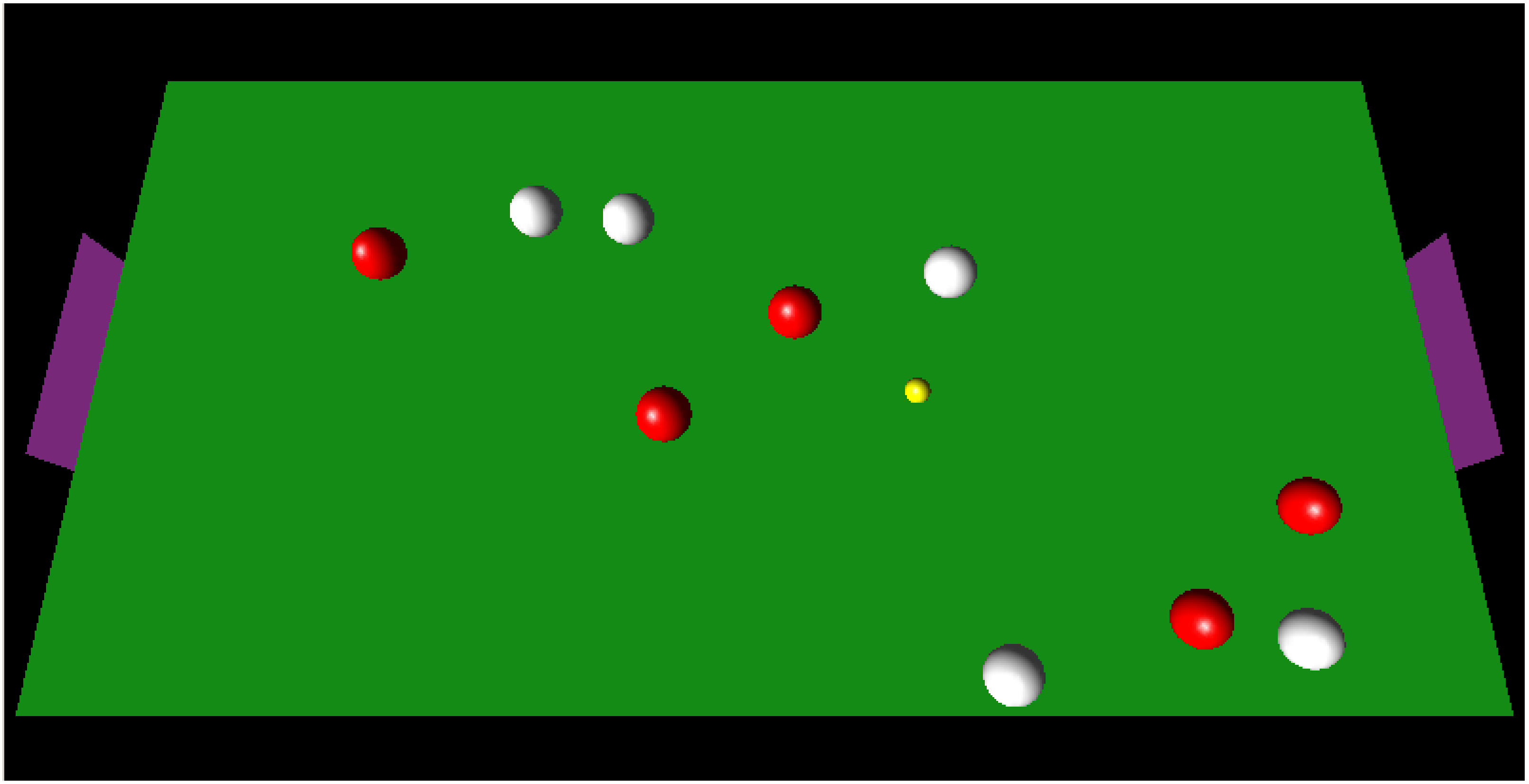
```
$ bii init myproject
$ cd myproject
$ bii open
diego/soccer(enunciado)
$ bii cpp:build
```

OR

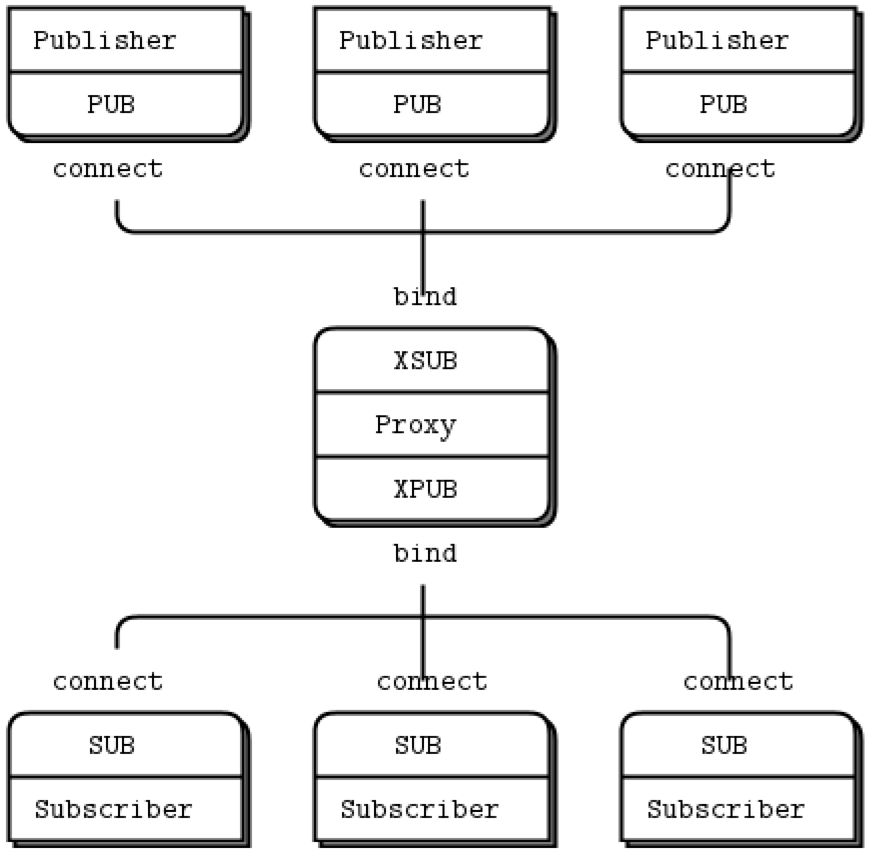
http://www.biicode.com/diego/diego/soccer/enunciado

Github.com/drodri/soccer -> branch enunciado

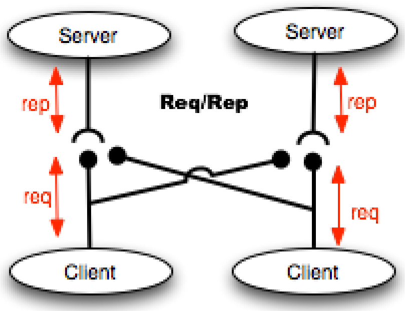
SOCCER TIME

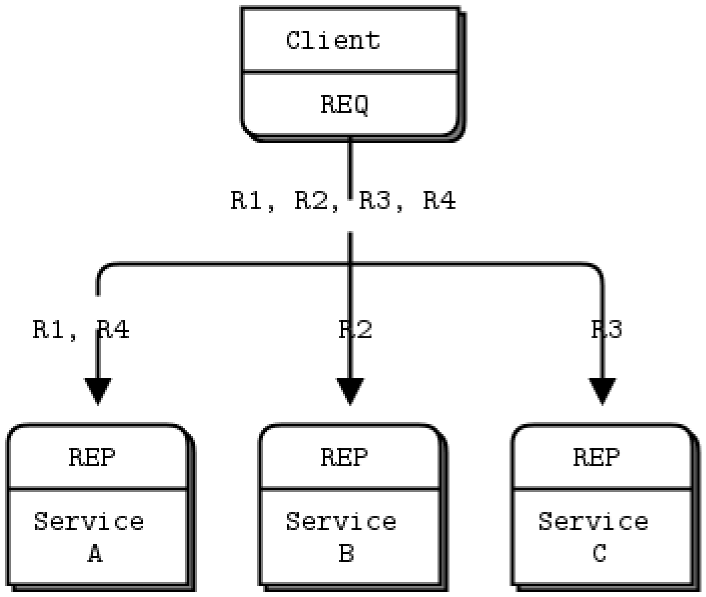


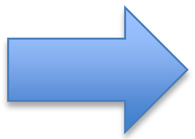
PUB-SUB with proxy

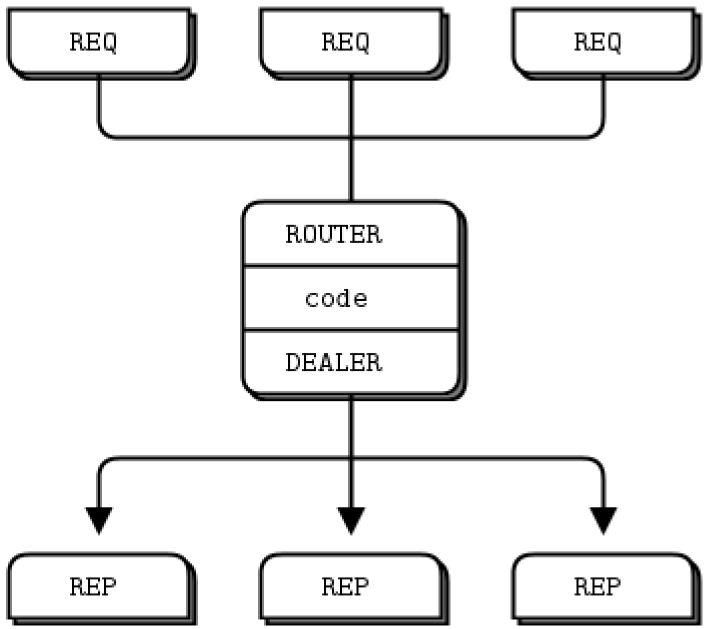


Extend REQ-REP with ROUTER-DEALER









Intro to ZMQ and Google Protobuf in C++

@diegorlosada

@t3chfest C3M Feb-2015