

A Simple Peer To Peer Network Implementation

Hoor Abootalebi
Nariman Aryan
Amin Isaai
Amirhossein Khajepour
Mahdis Tajdari
Ali Zeynali

November 2018

Contents

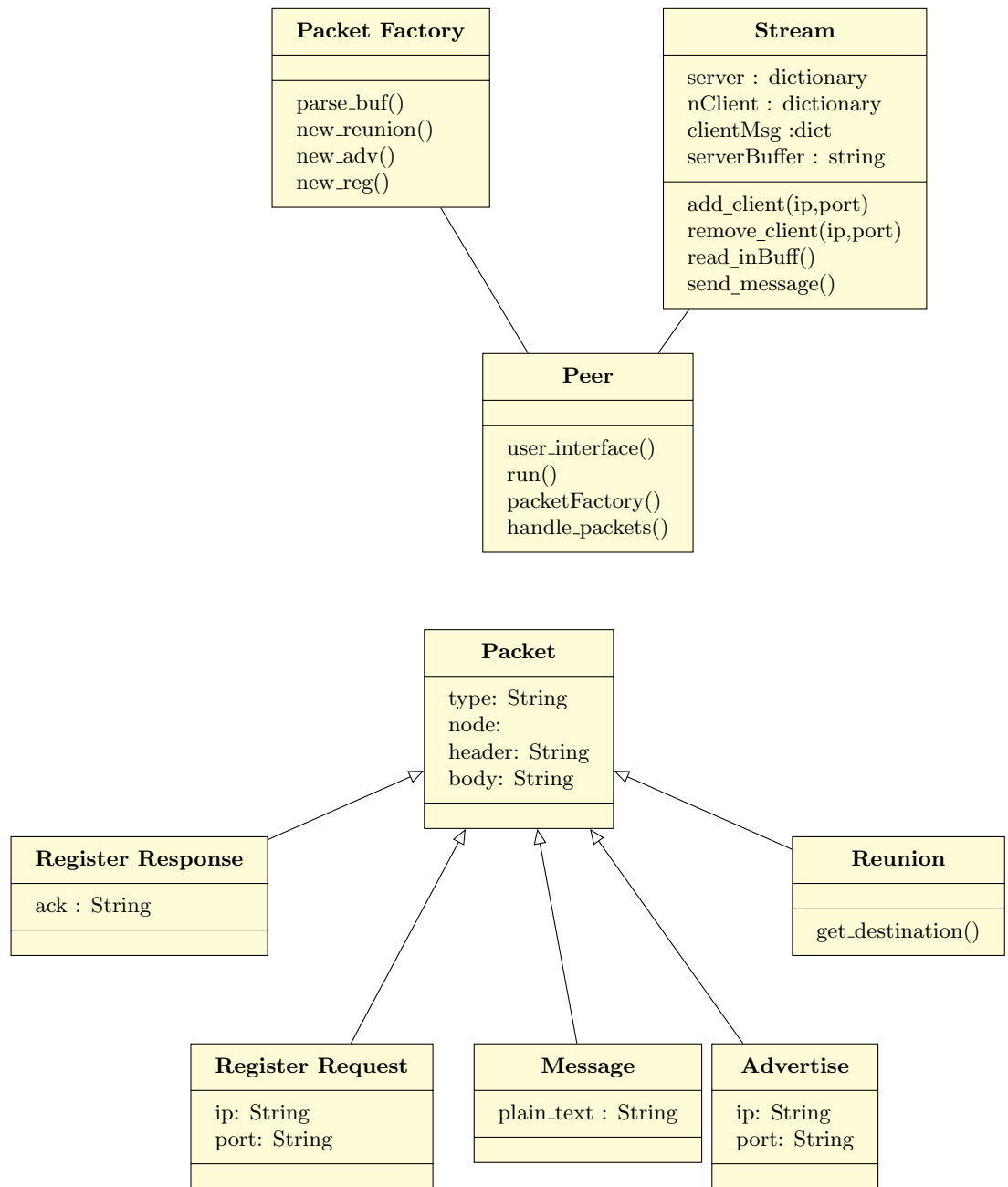
1	Introdeuction	3
2	UML model	3
3	Objects	4
3.1	Stream	5
3.2	Peer	5
3.3	Packet Factory	6
3.4	Packet	6
3.5	Reunion	7
3.6	Node	7
3.7	Resgister Request	7
3.8	Register Response	7
3.9	Advertise	7
3.10	Mesasge	7

1 Introdeuction

This project aims to implement a peer to peer network. In the first step we design UML model and then we are going to explain each objects' attributes and methods.

2 UML model

We design the UML model in order to make the project more understandable, clearer and professional.



3 Objects

Now it's time to explain every object's duty.

3.1 Stream

This object has one server and n clients. Servers are always open for reading and clients will be open whenever we want to write on a socket.

There is a clientMsg dictionary in this object to handle messages. This means that there is an array assigned to a specific client for all clients. If we need to add a new client to this object, we use add_client() method. Consequently, an array will be assigned to this new client in clientMsg.

There is also a remove_client() method for the times when we want to remove a client from this object. This method is mostly used when reunion fails.

The read_inBuf returns the buffer of the server.

The send_msg() method is used when peer wants to send a message to a specific client.

Byte_ack() is used to reply to the received messages. We must reply all of the received messages by sending Ack (which is a string).

```
#stream()
serverBuffer
add_client(ip,port)
remove_client(ip,port)
send_message()
read_inBuff()
```

We also need to add a dictionary to specify every client's message(s)

3.2 Peer

This object is the main object that we are working with. It must have a Stream object which provides the connection to the socket; This means that reading and writing are done using the Stream object. Peer must also have a userInterface object in order to facilitate commanding by users.(e.g. for connecting or sending message to a specific node)

The run method handles all of the events included in stream in an infinite loop and it also handles the received messages; This means

that it does a certain action based on the type of the received packet. The `handle_packet` function is a wrapper for doing each packets action; this function uses several internal functions that implemented for each type of packets as listed below:

1. `handle_advertise_packet`
2. `handle_reunion_packet`
3. `handle_register_packet`
4. `handle_message_packet`

```
#Peer()
stream()
user_interface() #Which the user or client sees and works with.
run()           #This method runs every time to see
                #whether there is new messages or not.
packetFactory()
handle_packets()
```

3.3 Packet Factory

`packetFactory()` The main functionality of this object is to create different types of packet and return them. To be more specific, we read data from buffer and we pass it through `pars.buff()` method in order to get a packet.

In addition this object has some methods to create our specified packet type like: `advertise`, `reunion`, `register`, `message`; for each packet type we have a separate function.

```
#packetFactory
parse_buf()
new_reunion()
new_adv() #makes a new advertise packet.
new_reg() #makes a new register packet.
```

3.4 Packet

Every packet consists seven differntes parts: **plain_text** which is the raw text message in the packet.

Node: Specifies to which node the packet sent to. **Sender** specifies

who sent the packet **Validator** which makes the packet valid.
Header where the information such as type of the packet and etc.
are going to be there.
Body body of our packet .

3.5 Reunion

`reunion(packet)` checks the connection of the nodes to the root.

```
#reunion(packet)
get_destination()
```

3.6 Node

Every node has two parameters: **IP** and **Port**.

3.7 Resgister Request

`reg_req()` sends IP/Port of a node to the root to ask if it can register it.

3.8 Register Response

`reg_res()` should just send an from the root *Ack* to inform a node that it has been registerd in the root if the `reg_req()` was successful.

3.9 Advertise

`adv(packet)`

3.10 Mesasge

`msg(packet)`