Message Queuing Telemetry Transport

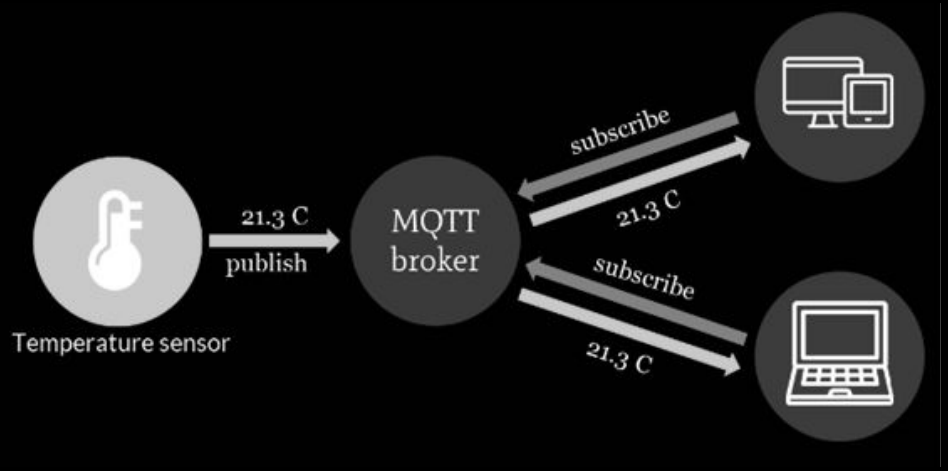
Armando Palacio - Ingeniería en Telecomunicaciones

Lab. 6 - Instituto Balseiro

Introducción

¿Qué es MQTT?

- Protocolo de comunicación que se distingue por su ligereza y sencillez.
- En un principio, se creó para conectar dispositivos y enviar la información de un sensor a servidores remotos.
- Hoy en día, se usa mayormente en la internet de las cosas (IoT).



Características

- Ligero y eficiente (clientes pequeños con recursos mínimos).
- Comunicación bidireccional.
- Puede aceptar millones de clientes.
- Tres niveles de QoS.
- Compatibilidad con Arduino, ESP8266 y Raspberry Pi.
- Modo de “publicación/suscripción” en lugar de “request/response”
- Al igual que HTTP, está implementado en la capa de aplicación sobre TCP/IP.

Protocolo

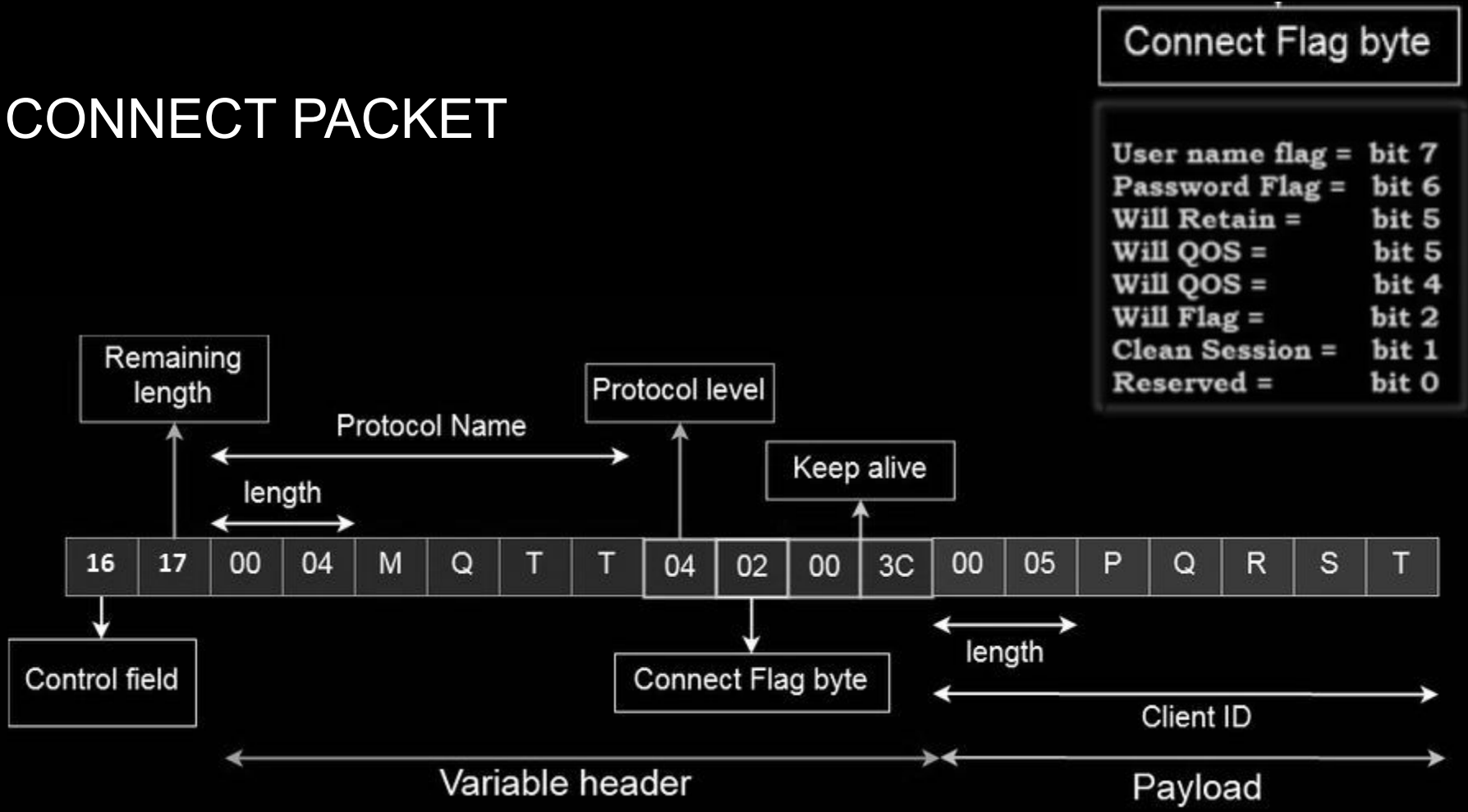
Tipos de Paquetes

```
class Header{ ···
class CONNECT_Msg:public Header{ ···
class CONNACK_Msg:public Header{ ···
class PUBLISH_Msg:public Header{ ···
class SUBSCRIBE_Msg:public Header{ ···
class SUBACK_Msg:public Header{ ···
class UNSUBSCRIBE_Msg:public Header{ ···
class UNSUBACK_Msg:public Header{ ···
class PINGREQ_Msg:public Header{ ···
class PINGRESP_Msg:public Header{ ···
class DISCONNECT_Msg:public Header{ ···
```


```
class Header{
    protected:
        Type type;
        uint8_t flags;
        int rem_len;

    public:
        Header() = default;
        virtual ~Header() = default;

        void setType(uint8_t* buffer) const;
        void setHFlags(uint8_t* buffer) const;
        int setRemLength(uint8_t* buffer) const;

        Type getType(uint8_t* buffer);
        int getHFlags(uint8_t* buffer);
        int getRemLength(uint8_t* buffer);

        virtual int pack (uint8_t * buffer) const{ return 0;}
        virtual int unpack (uint8_t * buffer) const{ return 0;}
};
```

Bit	7	6	5	4	3	2	1	0
Byte 1	MQTT Control Packet type				Flags specific to each MQTT Control Packet type			
Byte 2	Remaining Length							

CONNECT PACKET



CONNECT PACKET

```
class CONNECT_Msg:public Header{
    private:
        //Variable header
        uint8_t protocol_name[6];
        uint8_t protocol_level;
        uint8_t connect_flags;
        uint16_t keep_alive;

        //Payload
        uint16_t ID_len;
        string client_ID;
        uint16_t will_topic_len;
        string will_topic;
        uint16_t will_message_len;
        string will_message;

        //User Name Flag, Password Flag y Will QoS, deben fijarse a '0' en esta implementación.
        bool will_retain;
        bool will_flag;


    public:
        CONNECT_Msg(){};
        CONNECT_Msg(uint16_t keep_alive_, string client_ID_);
        CONNECT_Msg(uint16_t keep_alive_, string client_ID_, string will_topic_, string will_message_, bool retain_);
        ~CONNECT_Msg(){};

        int pack (uint8_t * buffer) const;
        int unpack (uint8_t * buffer);

        int check_protocol_name_level (uint8_t* buffer) const;
        int set_header (uint8_t* buffer) const;
        int get_header (uint8_t* buffer);

        int set_payload(uint8_t* buffer) const;
        int get_payload(uint8_t* buffer);

        bool is_will_retain() const;
        bool is_will_flag() const;
};
```

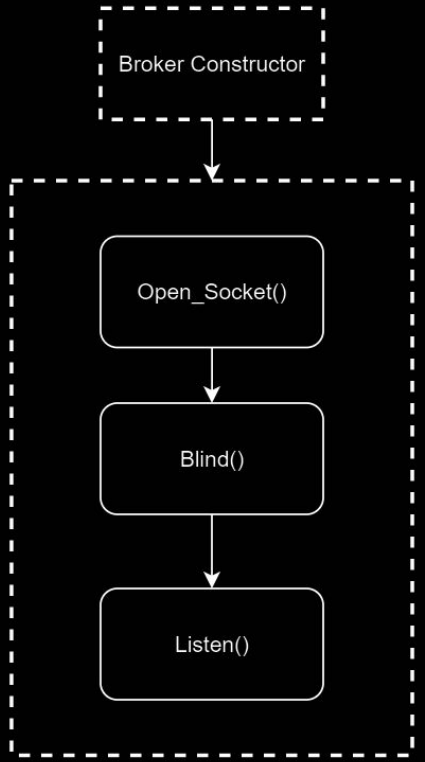

Broker

Broker

```
class Broker{
    private:
        int sockfd;
        struct sockaddr_in server_addr;
        list<Client*>  list_active_clients;
        Queue<PUBLISH_Msg> queue_msg;
        map<string, unordered_set<Client*>> subscriptions;
        map<string, PUBLISH_Msg*> retain_msg;
        mutex mutex_list_active_clients;
        mutex mutex_subscriptions;
        mutex mutex_retain_msg;
    public:
        Broker();
        ~Broker();
        int get_server_socket() const;
        void add_client_to_server(Client* c);
        void remove_client_from_server(Client* c);
        int get_number_of_clients() const;
        void add_client_to_topics(const list<string> &topics, Client* c);
        void remove_client_from_topics(list<string> &topics, Client* c);
        void get_clients_by_topic(const string& topic, list<Client*> &clients);
        bool check_client_ID(const string &client_ID);
        void new_publish(PUBLISH_Msg* msg);
        void publish_will_msg(Client* c);
        PUBLISH_Msg get_publish();
        PUBLISH_Msg get_retain_msg_by_topic(const string &topic);
};
```

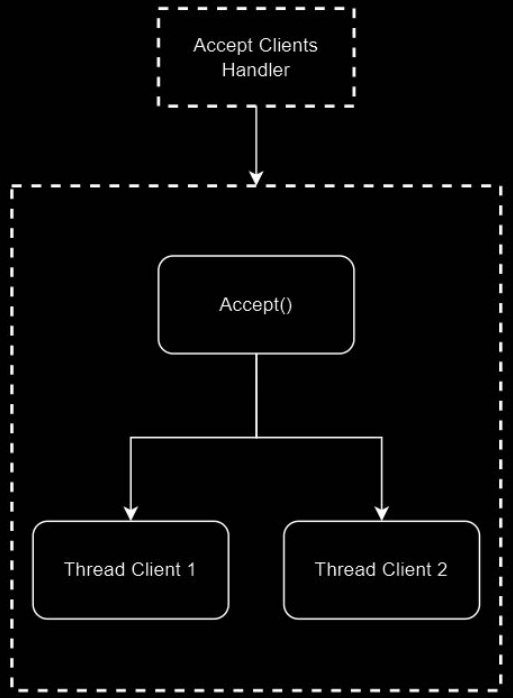
Broker

```cpp
#include "broker.h"

using namespace std;


int main(){
    Broker* Mqtt = new Broker();

    thread consumer_thr(consumer, Mqtt);
    thread accept_client_handler_thr(accept_client_handler, Mqtt);

    consumer_thr.join();
    accept_client_handler_thr.join();

    return 0;
}
```



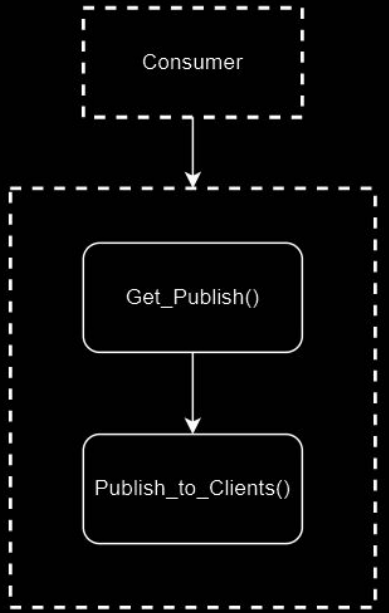
Broker

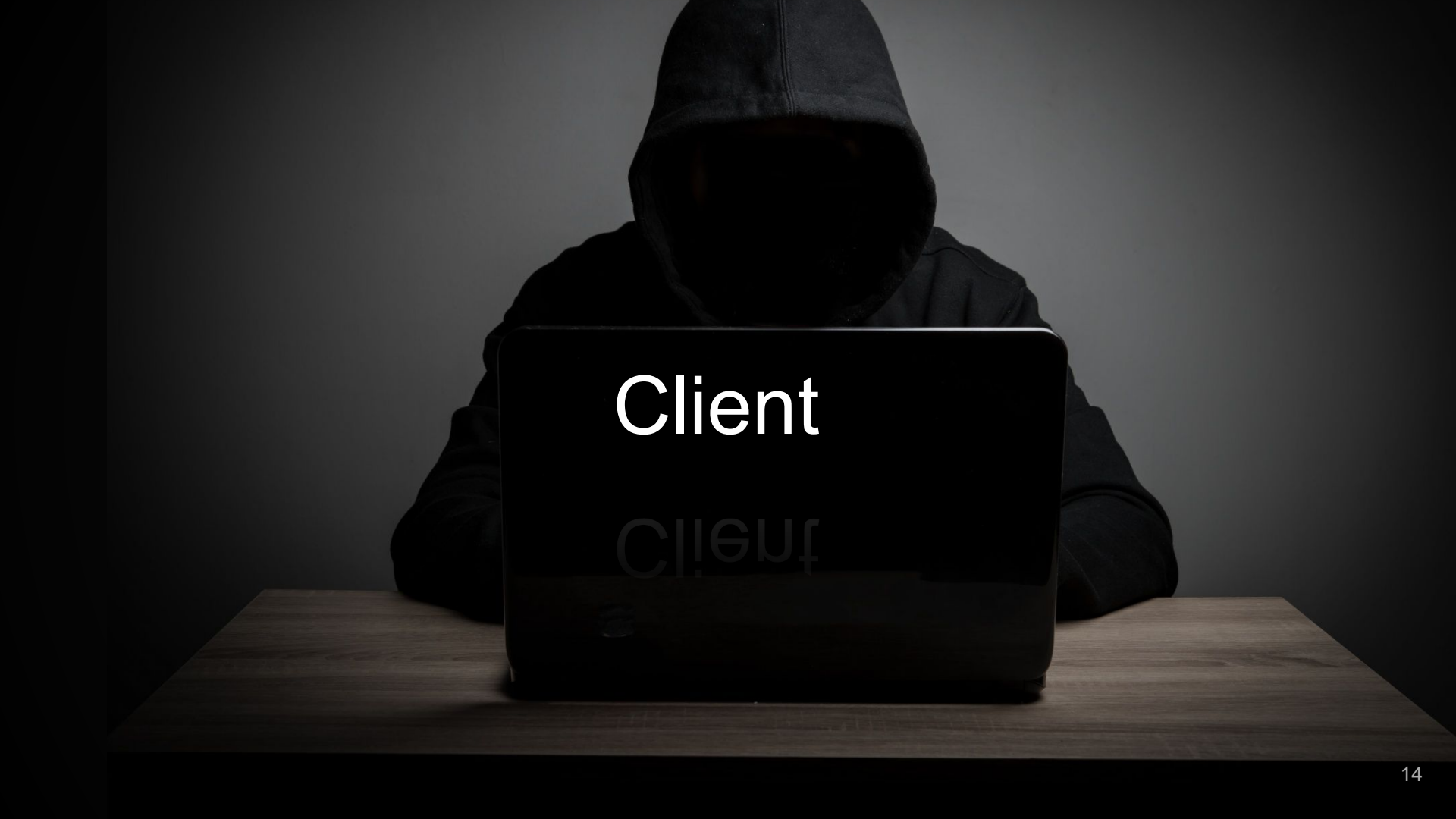
```cpp
#include "broker.h"

using namespace std;


int main(){
    Broker* Mqtt = new Broker();

    thread consumer_thr(consumer, Mqtt);
    thread accept_client_handler_thr(accept_client_handler, Mqtt);

    consumer_thr.join();
    accept_client_handler_thr.join();

    return 0;
}
```



Broker

```cpp
#include "broker.h"

using namespace std;


int main(){
    Broker* Mqtt = new Broker();

    thread consumer_thr(consumer, Mqtt);
    thread accept_client_handler_thr(accept_client_handler, Mqtt);

    consumer_thr.join();
    accept_client_handler_thr.join();

    return 0;
}
```



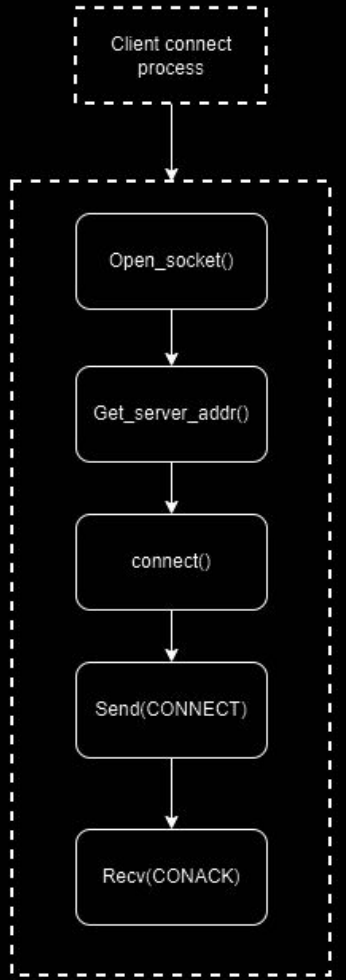

Client

Client

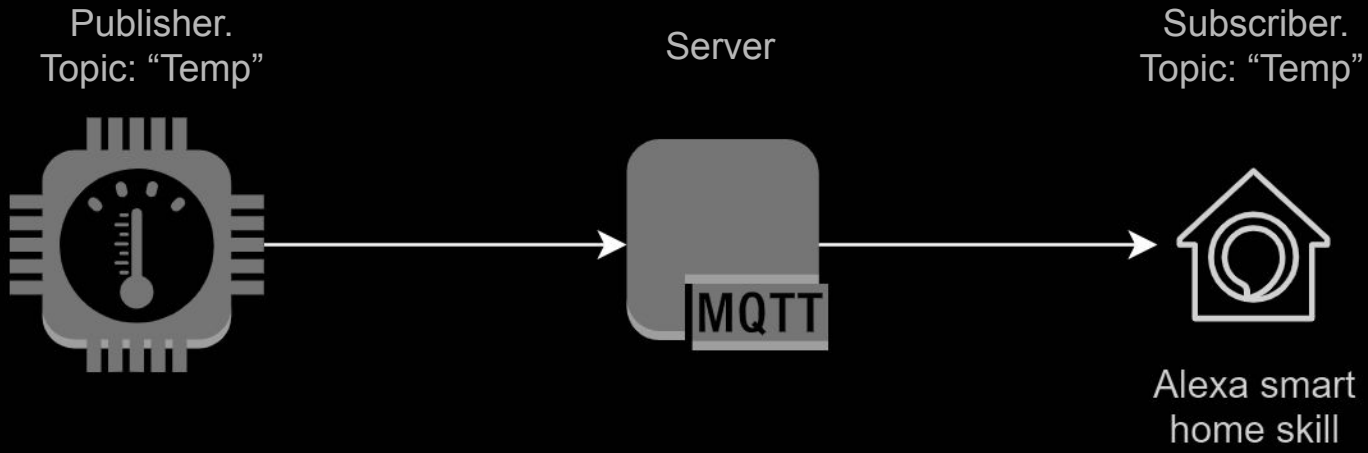
```
class Client{
    private:
        int sockfd;
        struct sockaddr_in serv_addr;
        string client_id;
        uint16_t keep_alive;
    public:
        Client(uint16_t keep_alive, string client_id);
        ~Client();
        int get_sockfd() const;
        string get_client_id() const;
        uint16_t get_keep_alive() const;
        void get_server_addr(char* hostname);
        void CONNECT();
};
```

Client

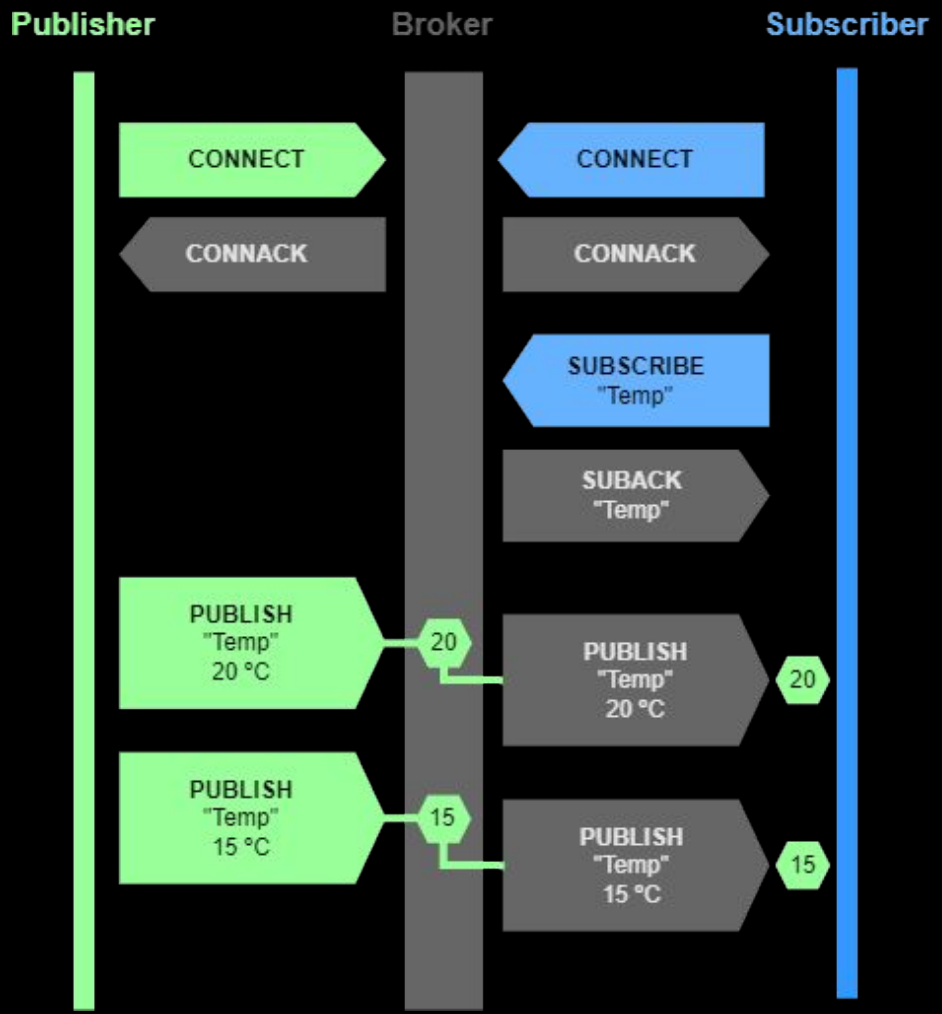
```cpp
#include "client.hpp"
#include "proto.h"
#include <signal.h>

int main(int argc, char *argv[])
{
    char* hostname = argv[1];

    uint16_t keep_alive = 0;
    string client_id = "client1";

    Client* client = new Client(keep_alive, client_id);
    client->get_server_addr(hostname);
    client->CONNECT();

    CONNECT_Msg msg (client->get_keep_alive(), client->get_client_id());
    if(sendMsg(client->get_sockfd(), msg) == -1)
        error("Error al enviar CONNECT");

    CONNACK_Msg msg2(0x00);
    if(recvMsg(client->get_sockfd(), msg2) == -1)
        error("Conección rechazada!");
    print("Conexión aceptada!");
```



Test


Publisher.
Topic: “Temp”
Server
Subscriber.
Topic: “Temp”
MQTT
Alexa smart
home skill

Secuencia de Paquetes



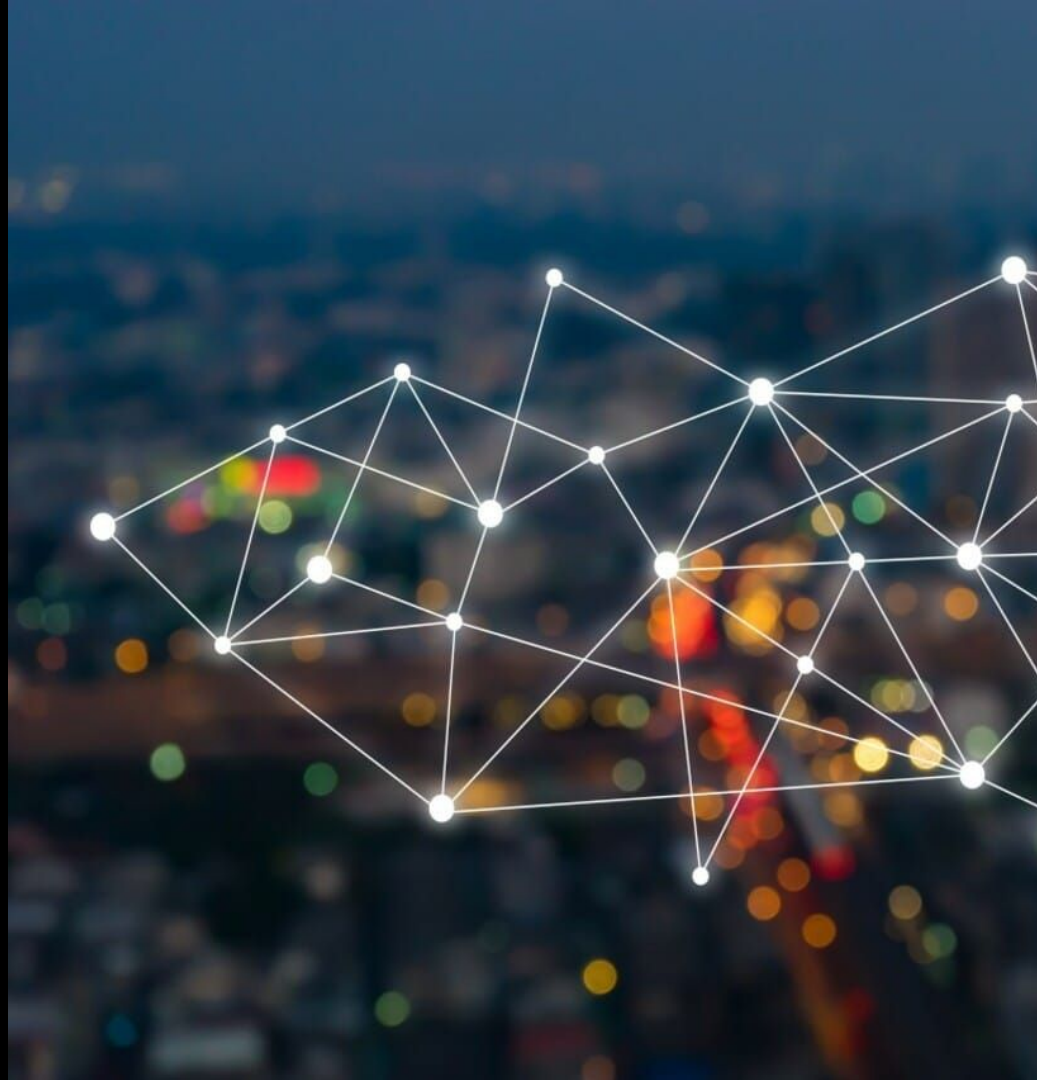
Client Publisher

```
    string message;
    bool retain = 0;

    while(!DISCONNECTED){
        message = to_string(15 + rand()%10);
        PUBLISH_Msg msg5(topic, message, retain);

        if(sendMsg(client->get_sockfd(), msg5) == -1){
            close(client->get_sockfd());
            error("Error al enviar PUBLISH");
        }
        print("sended: " + message);

        sleep(1);
    }

    DISCONNECT_Msg msg6;
    if(sendMsg(client->get_sockfd(), msg6) == -1){
        close(client->get_sockfd());
        error("Error al enviar DISCONNECT");
    }
    print("Desconectado del broker!");

    close(client->get_sockfd());
    return 0;
}
```

Client Subscriber

```
    uint16_t packet_id = 12;
    SUBSCRIBE_Msg msg3(packet_id, (list<string>) {topic});
    if(sendMsg(client->get_sockfd(), msg3) == -1)
        error("Error al enviar SUBSCRIBE");

    SUBACK_Msg msg4;
    if(recvMsg(client->get_sockfd(), msg4) == -1)
        error("Error al recibir SUBACK");
    msg4.checkReturnCode();

    while(!DISCONNECTED){
        PUBLISH_Msg msg5;

        if(recvMsg(client->get_sockfd(), msg5) == -1){
            close(client->get_sockfd());
            error("Error al enviar PUBLISH");
        }

        print("Temperatura: " + msg5.getMessage());
    }

    DISCONNECT_Msg msg7;
    if(sendMsg(client->get_sockfd(), msg7) == -1){
        close(client->get_sockfd());
        error("Error al enviar DISCONNECT");
    }
    print("Desconectado del broker!");

    close(client->get_sockfd());
    return 0;
}
```

Conclusiones



Conclusiones

- El broker funciona correctamente aunque no implementa QoS y keep alive.
- Los Clientes realizan correctamente la conexión, suscripción, publicaciones, de-suscripción y desconexión.
- Tanto el Broker como los Clientes fueron probados con “mosquitto” y funcionan correctamente.

END