ReTRo – UDP

Background

ReTRo – UDP (Real Time Reliable over UDP) is a communication protocol based on UDP. This protocol is designed for communication between IoT Devices. The principles behind the protocol are:

* A large number of IoT devices sends to each others small messages
* The frequency of messages can be high or low
* Usually a message represents a real time information
* Messages must arrive at their destination as soon as possible
* Each device can send messages to many devices simultaneously
* A device usually send different types of messages (i.e. status updates, actions, negotiation, requests)
* The system’s topology can be client-server, p2p or hybrid and can change dinamically

So the communication needs to be fast and reliable, it must also allow exchange of messages between many different devices.

Implementation choices

Given the need for fast and possibly frequent communication of small messages, it was decided to use the UDP protocol instead of TCP.

Definitions

* A *User* is defined by the pair (ip,port)
* When a *User* receives or sends a message to another *User*,not yet met, he creates a *Connection*: a set of *Mailbox* each identified by a number. A *Connection* allows bidirectional communication with a single user (ip, port).
* A *Mailbox* is defined by two *Channels*, one for the messages to be sent and the other for those received.
* A *Channel* is a special queue: it has a maximum size (max\_size) and contains only the max\_size most recent messages.

It was chosen to use the *Channels* because given the real time nature of the information contained in the messages, it would not make sense to store too old information.

The choice of using multiple *Mailboxes* for communication with the same *User* allows to differentiate already at the communication level different messages’s type. Furthermore, the management of messages in different *Mailboxes* is completely independent.

The ReTRo protocol has the following characteristics to ensure the reliability:

* each message has a sequence number and a mailbox number
* when a message is received an ack message is sent in response
* if an ack is not received within a certain time limit the message is sent back
* if the sequence number is less than or equal to the last message received in the same *Mailbox*, the message is discarded
* for each *Mailbox:* as long as you do not receive an ack message for the last message sent, no other messages are sent from the same mailbox until the ack is received or the message is discarded from the *Channel* because it is too old

These mechanisms allow to have a real time, fast and reliable protocol.

The multi-*Mailbox* system allows you to manage messages to the same *User* independently: while waiting for an ack in a specific mailbox, the others can continue communication independently.

The sequence number also allows you to manage the sorting of messages from the same mailbox.

Furthermore, if the last message arrived is for example 42, and arrives 44 before 43, it is not important to manage the situation, because a correct use of *Mailboxes* allows to manage only one type of messages for each *Mailbox*. So message 44 will be an "update" of 43, therefore the information lost in message 43, which even if it arrives at its destination will be discarded, is not important.

Characteristics

ReTRo is:

* Bidirectional
* Multiply-Connected (1 Port – N Connection)
* Reliable (allows ordering and acknowledge)
* Message oriented
* P2P Friendly
* W/o initialisation phase or closing phase
* With flow control (RTO)
* With implicit window control (Channel)
* With Implicit Flow label (Mailbox)
* With parallel management of flows
* With 32 bit header

The protocol does not require an initialisation phase, buffers aren’t used to handle unordered messages. In fact, all these feauters would slow down communication and considering the real time nature of the messages and their small size, they would not improve performance.

Header (32 bit)

SYN -> 1 bit

ACK -> 1 bit

Sequence/Ack number -> 16 bit

Global number -> 10 bit

Mailbox number -> 4 bit