**Monitor and Control Garage Protocol (MCGP)**

Group 2

Kenneth Balogh

Arudra Venkat

Xiaofeng Zhou

Daniel Speichert

# 1.Service Description

The Monitor and Control Garage Protocol (MCGP) provides a mechanism for users to manage, monitor and control a garage through the services defined by the protocol. MCGP provides a communication specification for applications, and is a protocol defined at the application layer utilizing TCP/IP. MCGP uses TCP as its transport layer protocol in order to have a reliable and ordered connection. It adopts client-server mode and provide safety mechanism, the mechanism is based by user authentication which is realized by message signing via public-private key and embedding hash-signature in PDU.

After the client passes the server’s authentication, users can have a remote connection with their garage. Applications that implement MCGP will provide two kinds of services for users. Clients will connect to a central server that will provide several services. First is the monitor service, server may connect to many digital devices in the garage such as thermometer, barometer and hygrometer. This allows a user to watch various environment parameters through the client. Second is the control service, in which a user can remotely control the door of the garage or the light inside it through client. Figure 1 is the schematic diagram of the MCGP.

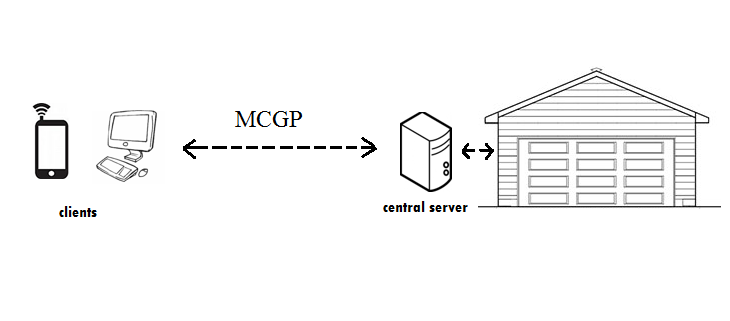


Figure 1.

# 2 Message Definition – PDU

## 2.1 Addressing

The protocol is designed to operate over any reliably ordered transport layers. It is recommended that TCP/IP be used for the implementation. The client will establish a connection with a server using its IP address and a designated port number. In order to allow multiple connections to the server port numbers will used to allow different devices to be connected at any given time. An unused port of 9070 was selected because it does not require registration as is currently not being used by any other services at this time.

## 2.2 Flow Control

By utilizing TCP/IP flow control is handled at the TCP/IP Layer. This includes features such as QoS and Flow control.

## 2.3 PDU Definition

Communication is divided as follows:

1. Handshake –client and server doing authentication and version check.
2. Initialization – Server Sends a Check Garage Status Message to the client
3. Client/Server Communication –
   1. Client sends messages at any time and the server responds based on the message type. Table 1 below shows the initial message types.
   2. Server Responds according to the message type and action.

All messages are an ordered byte stream with a fixed size. When multiple packets of devices need to be sent a ByteCode of 0xA is sent. The client will then read each device or item as it is sent. The following convention will be used for this document C designates communication from the client and S is communication from the Server. [MessageByte] will be used to designate the message. An example is C:[0x00] which is the client sending the request connection.

Message Byte Code Message Type Sent By

0x00 Request Connection Client

0x01 Error Client/Server

0x02 Connection Ready Server

0x03 Read Device List/Status Server

0x04 Control Device Client

0x05 Done Client

0x06 Confirm Server

0x07 Version Server/Client

0x08 Authenticate Server

0x09 Service Done Client

0x0A Number of Devices Server

Table 1 Message Codes

Error Byte Error Reason

0x00 Connection Refused

0x01 Authentication Error

0x02 Server Error

0x03 Checking Status Error

0x04 Server Control Error

0x05 Client Command Error

0x06 Server Execution Error

Table 2: Error Bytes

### 2.3.0 PDU format

0 1 2 3

0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1

+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

| version (1) | op (1) | errno (1) | payloadlen (1)|

+---------------+---------------+---------------+---------------+

| |

| payload (variable) |

| |

+---------------+---------------+---------------+---------------+

| sig ver (1) | challenge (2) | sig len (1) |

+---------------+-------------------------------+---------------+

| signature (variable) |

+---------------------------------------------------------------+

Figure 2.1: Format of a MGCP message

Field descriptions:

\* version (1 byte) is now 0x00

\* op (1 byte) is according to Table 1

\* errno (1 byte) is according to Table 2

\* payload length (1 byte) is the size (in bits) of payload (following)

\* signature version (1 byte) is used to differentiate between two available signature generation methods

\* challenge (2 bytes) is sent by the server to the client, to be used when signing next message

\* signature length (1 byte) is the size (in bits) of signature

### 2.3.1 Handshake

The handshake phase is used to accomplish two tasks:

1. Version
2. Authentication

A client initiates this by sending a request connection command.

C:[0x00]

The server will then respond with the version byte code followed by the version it supports. This implementation will be version 1 so 0x01 will be sent.

S:[0x07][0x01]

The client will respond with the version it supports. A server can except the client by initiating authentication.

S:[0x08][authtication challenge]

The Authentication process will be discussed in more depth in Section 5.

If the authentication is successful, the server responds with a connection ready.

S:[0x02]

If there was an error in the client version or authentication, then the server responds with an error message and an error code.

S:[0x01][ErrorByteCode]

### 2.3.2 Discover Phase/Check Garage Status

Once the Handshake Phase has been completed the client will send a check garage status command. The server will respond with the list of devices and their current status.

All device actions are predefined in this version of the protocol.

S:[0x03][Number of Devices to Be Sent]

S:[D0][DeviceCode][CurrentState]

..

S:[DX][DeviceCode][CurrentState]

Until All Devices Are Sent (DX represents last device, for example, if there are 5 Devices then D4 is last device sent]

### 2.3.3 Client to Server/ Server to Client Messages

The client can send messages to control the device or read a device’s parameter.

C: [0x04][DeviceID][Action]

An example of this if the client wants to turn a light D1 on. A client will send

C: [0x04][D1][Action]

Future versions could support additional parameters or custom controls.

Table 3 below shows the initial list of devices that are supported.

Initial Device Types

Code Device Type States Action

0x00 Garage Door open(0x00)/closed(0x01) turn on(0x01) turn off(0x00)

0x01 Light on(0x01)/off(0x00) open (1) close(0x00)

0x02 Temp Sensor on(0x01)/off(0x00) read temperature(0x02)

0x03 BaroPres Sensor on(0x01)/off (0x00) read pressure(0x03)

0x04 Hygro Sensor on(0x01)/off (0x00) read humidity

At any time if an error is received the error message and error byte will be sent by either the client or server.

## 2.4 Error Control

There are several different types of error control as mentioned table 2. At this stage of protocol design, when an error occurs the error message and error code will be sent, the connection is then terminated and everyone goes back to the IDLE state. The client must then reinitiate the connection with a Request Connection as described above. In the future, the reaction to different kinds of error may be different, varying from directly terminate the connection to roll back to server ready state or adding some timeout mechanisms, the error control will be update with the evolve of the version of the protocol.

## 2.5 Quality of Service

The protocol is simplistic to use but still allows for authentication with the server. It provides a simple method to control and read the status of various devices. This is an advantage when most devices will be small resource constrained devices. Having a protocol that is simple to implement but still provides control and security is important in an ever connected world.

# 3. DFA

Figure 3.1 is the DFA of MCGP protocol.

As described in the beginning of section 2.3. There are three main stage of MCGP protocol. The version checks and authentication states form the handshake stage. Client and server determine the version of protocol they use in version check state, if agreement can be made, it will move to the authentication state, otherwise, either client or server will send an error message and terminates the connection. If they succeed, server and client will then do the authentication process, as mentioned in section 1, it is done by utilizing private and public key, details can be found in section 5.

After connection has been successfully established, the client will ask the server to send all environmental parameters of the garage and the states of all devices. This is done automatically every time the connection is established. After the server sends all information, the protocol will move to server ready state and the server will keep on running and wait for client’s command. Commands type sent by clients can be found in table 1 of Section 2.3.

The bottom right part of the DFA is the control service part. Client can control devices such light and door of the garage based on states defined by the MCGP protocol.

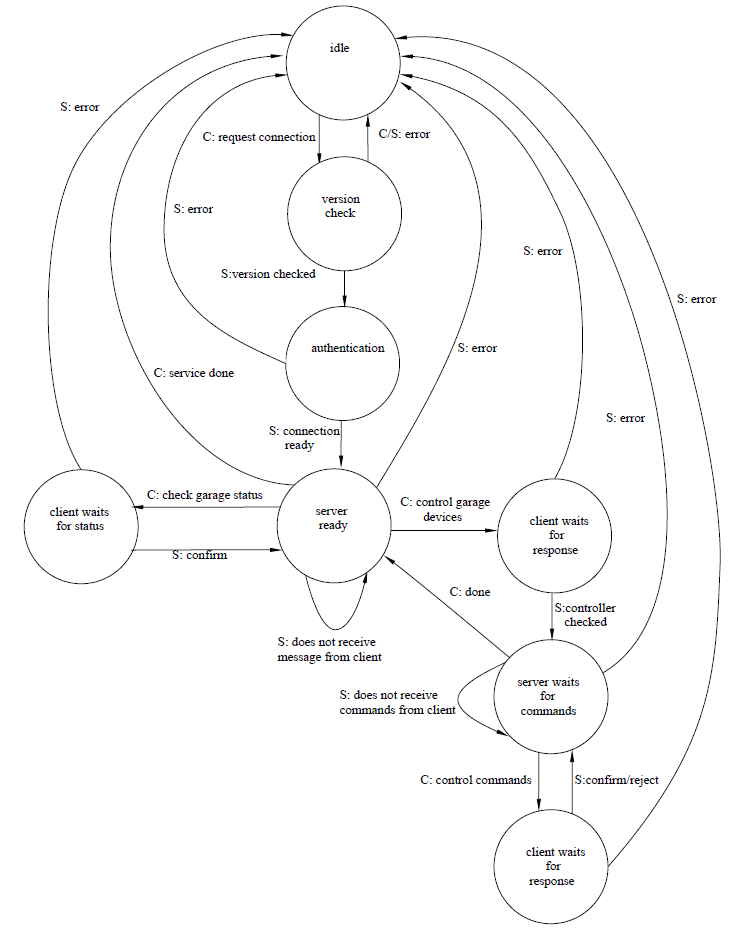


Figure 3.1 DFA of MCGP

# 4. Extensibility

MCGP is built to be an extensible application level protocol. The first reason is that it is meant to run on top of existing protocols like TCP. Since TCP itself is built to be extensible, we inherit that benefit. We can add a layer of security using TLS. We could use it with Websocket(WS) or Secure Web Socket (WSS). We could make further security enhancements by adding secure tokens like OAuth in the payload.

In addition to extensibility with sublayers, there is also the ability to extend on an application layer. We could implement a version to be more specific that would be built on top of MCGP. It could validate certain types of devices by massages in the payload.

We could also build on top to add additional device types with new device codes to the PDU in future versions.

Additional protocols also extend this one and define custom messages and build a more specific message spec that certain devices and server applications could honor, for instance say Company X builds garage doors and other automated devices, they could define a protocol XMCGP which has specific messages that X devices and X Server implementation understand. This MCGP serves as a building block for future home automation product protocols

# 5. Security

## 5.1 Security model

The MCGP protocol's security model assumes that MCGP protocol will be used primarily over a local, trusted network. Second assumption is that the data carried by MCGP protocol (such as temperature) is not sensitive. Therefore, MCGP focuses on authentication rather than encryption.

Much of MCGP's security depends, therefore, on the security of physical network. While on the public Internet, the threat of a malicious intermediary intercepting or modifying the information on-the-fly is high, such attacks are not expected in (W)LANs. A properly secured local home network isolate outsiders and any party able to listen in on the unicast traffic on the network is therefore probably the network owner.

MCGP specifically assumes that:

- TCP connections cannot be hijacked

- TCP/IP packets cannot be rerouted (or ARP-spoofed) to a malicious server

- Unicast packets cannot be intercepted

## 5.2 Authentication method

Despite these naive assumptions, MCGP takes measures to validate incoming client connections. At this moment, only servers can authenticate clients. The clients have to implicitly trust that the server they connect to is not an imposter (see security assumptions above).

### 5.2.1 Message hashing & signing

For the purpose of identification and authentication, every packet is signed by the client. The PDU includes a field (last field) that carries a secure signature of a hash of the message. The message signature can be obtained in one of two supported methods:

* Shared-secret signature

The hash-signature is obtained by concatenating all fields of the PDU (in the order they are defined and sent) together, appending the byte-representation of the shared-secret (key) and computing an sha256 sum. The sum is then transmitted as the last field of the PDU.

* Public-key signature

The hash-signature is obtained by concatenating all fields of the PDU (in the order they are defined and sent) together and computing an sha256 sum. The sum is then signed using the RSA 2048-bit private-key. The signature is used as the last field of the PDU.

### 5.2.2 Message authentication

The server can authenticate a message by computing a hash of the packet received from a client using same rules as defined above. In case of shared-secret signature, the server knows the shared secret and can apply it. In case of the public-key signature, the server knows the public key of the client and can verify the signature.

### 5.2.3 Connection authentication

The client can authenticate a connect in its early stage by sending its ID (e.g. a hostname or username) as the authentication payload and signing the packet properly. The server can then determine if the signature matches the claimed name.

## 5.3 Security risks / Vulnerabilities

* Server spoofing / MITM attacks - the protocol does NOT provide any resiliency against MITM attacks.
* Replay attacks - the protocol defends against replay attacks by sending a new randomly-generated one-time-challenge from the server to the client, to be included (and signed) in the next packet. The security of this solution depends on the randomness of the server.
* Packet tampering - the protocol's message signing method provides sufficient protection against any message tampering.