# 

# Monitor and Control Garage Protocol (MCGP)

Group 2  
Daniel Speichert  
Kenneth Balogh  
Arudra Venkat  
Xiaofeng Zhou

# 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, secure and ordered connection.

The protocol works in a client-server mode and provides authentication and encryption using TLS. After the client passes the server’s authentication, users can have a remote connection with their garage. Applications that follow MCGP will provide two services to users:

1. 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.
2. 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.

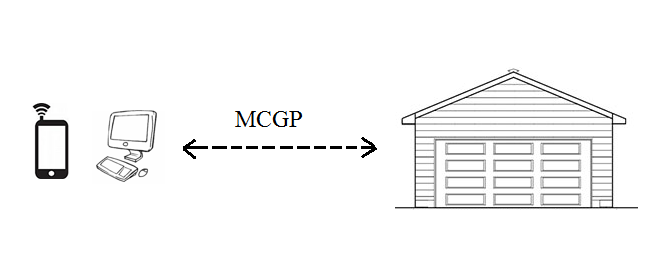


Figure 1 MCGP Picture

# 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 6666 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

Flow control is handled at the TCP/IP Layer.

# 2.3 PDU schema

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) | /reserved/ (1)|  
 +---------------+---------------+---------------+---------------+  
 | ident (authentication identifier) (8 bytes) |  
 | |  
 +---------------+---------------+---------------+---------------+  
 | dev\_id[0] (1) |dev\_type[0] (1)| dev\_status[0] | dev\_action[0] |  
 +---------------+---------------+---------------+---------------+  
 | dev\_value (4) |  
 +---------------+---------------+---------------+---------------+  
 | dev\_id[1] (1) | dev\_type (1) | dev\_status[1] | dev\_action[1] |  
 +---------------+---------------+---------------+---------------+  
 | dev\_value (4) |  
 +---------------+---------------+---------------+---------------+  
 | dev\_id[2] (1) | dev\_type (1) | dev\_status[2] | dev\_action[2] |  
 +---------------+---------------+---------------+---------------+  
 | dev\_value (4) |  
 +---------------+---------------+---------------+---------------+  
 | dev\_id[3] (1) | dev\_type (1) | dev\_status[3] | dev\_action[3] |  
 +---------------+---------------+---------------+---------------+  
 | dev\_value (4) |  
 +---------------+---------------+---------------+---------------+  
 | dev\_id[4] (1) | dev\_type (1) | dev\_status[4] | dev\_action[4] |  
 +---------------+---------------+---------------+---------------+  
 | dev\_value (4) |  
 +---------------------------------------------------------------+  
  
 Figure 2: Format of a MGCP message

Field descriptions:

* version (1 byte) unsigned integer is now 0x01 (MUST be set on every packet)
* op (1 byte) unsigned integer according to Table 1
* errno (1 byte) unsigned integer according to Table 2
* ident (4 bytes) UTF-8 encoded string
* dev\_id (1 byte) unsigned integer (device unique identifier on the server) dev\_id MUST NOT be zero (reserved for empty)
* dev\_type (1 byte) unsigned integer according to Table 3
* dev\_status (1 byte) unsigned integer according to Table 3
* dev\_action (1 byte) unsigned integer according to Table 3
* dev\_value is a 32 bit (4 bytes) floating point number

All messages are an ordered byte stream with a fixed size with values coded in big-endian format.

Unused (unapplicable) fields SHOULD be set to their zero value and MUST be discarded by recipient.

Op Code Message Type  
0x00 /Reserved/  
0x01 Version Check  
0x02 Authentication  
0x03 List Devices / List complete  
0x04 List Devices / Continued  
0x05 Control with Action  
  
Table 1 Message Codes

Error Byte Error Reason  
0x00 NO ERROR  
0x01 Version not available  
0x02 Authentication Error  
0x03 List Devices Error  
0x04 Control error  
0x05 Unexpected operation  
  
Table 2: Error Bytes

------------------------- Initial Device Types --------------------------------  
  
Device Code Device Type States Available Actions  
0x00 -> /empty/  
0x01 -> Garage Door open(0x00)/closed(0x01) open(0x01)/close(0x00)  
0x02 -> Light on(0x01)/off(0x00) turn on(0x01)/off(0x00)  
0x03 -> Temp Sensor on(0x01)/off(0x00) (no actions)  
0x04 -> BaroPres Sensor on(0x01)/off(0x00) (no actions)  
  
Table 3: List of supported devices

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 Communication protocol

The protocol flow is as follows:

1. Connect with TLS Handshake
2. Handshake – Client sends the protocol version
3. Authentication - Client sends the identity it wants to use
4. Listing - Client sends a query to list all devices and states
5. Control - Client sends tuple
6. Disconnect

All fields that are unused should be set to zero.

## 2.4.1 Handshake

Client sends to server a single packet: \* version=X \* op=0x01 (version check)

Successful server response: \* version=X \* op=0x01 (version check) \* errno=0x00 (no error)

Erronous server response: \* version=X \* op=0x02 (version check) \* errno=0x01 (version not available)

## 2.4.2 Authentication

Client sends to server a single packet: \* version=X \* op=0x02 (authentication) \* ident=Y

Successful server response: \* version=X \* op=0x02 (authentication) \* errno=0x00 (no error)

Erronous server response: \* version=X \* op=0x02 (authentication) \* errno=0x02 (authentication error)

## 2.4.3 Listing

Client sends to server a single packet:

\* version=X

\* op=0x03 (list devices)

Successful server response (there are more than five):

\* version=X

\* op=0x04 (list devices / continued)

\* errno=0x00 (no error)

\* server sets dev\_id, dev\_type, dev\_status as appropriate

\* server MUST send next packet

Successful server response (this is the last 5):

\* version=X

\* op=0x03 (list devices / complete)

\* errno=0x00 (no error)

\* server sets dev\_id, dev\_type, dev\_status as appropriate

\* no more packets in this listing from server

Erronous server response: \* version=X \* op=0x03 (list devices) \* errno=0x03 (list devices error)

## 2.4.4 Control

Client sends to server a single packet: \* version=X \* op=0x05 (control with action) \* device\_id=Y, device\_action=Z (up to 5 pairs)

Successful server response: \* version=X \* op=0x05 (control with action) \* errno=0x00 (no error)

Erronous server response:

\* version=X

\* op=0x05 (control with action)

\* errno=0x04 (control error)

# 2.5 Quality of Service / Error Control

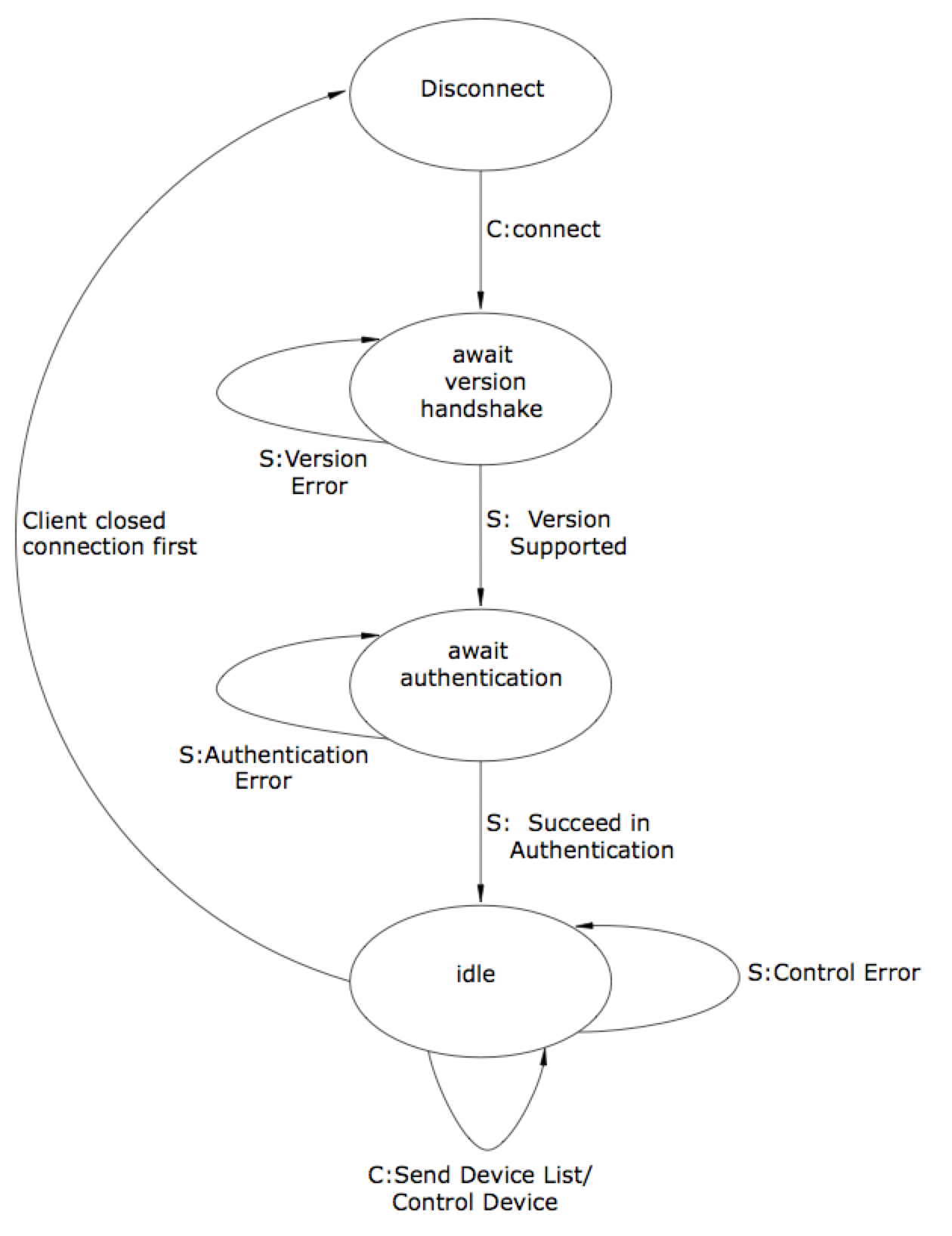
The are several different types of error control as mentioned above. When an error occurs the error message and error code are sent. The connection is then terminated and everyone goes back to the IDLE state. The client must then reinitiate the connection with a Request Connection as described above.

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

Figure3.1 is the new DFA of the MCGP protocol. It has 4 states, after the connection between client and server successfully been set up. The server will check whether the client's version is supported or not. If not, server will response with a version error and it will keep on waiting for the client's action. If the version is supported by the server, it will send back a response and move to the await authentication state.

In the authentication state, the server will check whether or not the identifier sent by the client matches its own. If not, the server will send back an authentication error and keep on waiting for the client's new identifier. If succeed, the server will move to idle state.  
In the idle state, the server will keep on listening request sent by the client. It supposed to deal with two kinds of operation, list and control. When the server receives list command, it will send the statuses of all the devices in the garage back to the client and wait for the next command. When the server receives a control command, it will change the status of a device depends on the command. The connection will be closed if the server receives a close connection command from the client. Any error happens in the idle state will cause the server to send a control error back to the client, the server will then keep on listening for the coming request.

  
Figure 3.1 DFA

# 4. Extensibility

MCGP is build to be an extensible application level protocol. There are many fields in the PDU that support expansion in future version. Some areas of extensibility are listed below.

4.1 Version The first byte in the PDU is a protocol version field. In initial handshake, the client sends the version number to the server, which checks if it supports it and then responds. Future protocol versions may increment the protocol version on the client and server side when newer protocol versions are established. Also a server set to run at a higher version might optionally support lower protocol versions and switch to that mode when sending messages depending on the value it receives during the handshake.

4.2 Operations Since only the first 6 values of the opcode byte are used, future protocol versions might support future operations (say for instance publish / subscribe operations) which could occupy unused values in that field.

4.3 Devices The server may list up to 5 devices in a single message. However, the opcode defines if the list is complete or more devices are to follow. This allows for the protocol to support many devices and scale up depending on the need.

Now the device type byte currently has only 3 possible device types, but the other values are left for future device types to be used with this protocol. Furthermore, many different device actions may be defined. Similarly, the device value has 4 bytes which may be specifically defined or reserved for certain messages in future protocol versions.

4.4 Configurability MCGP is meant to run on top of existing protocols like TCP. Since TCP itself is built to be extensible, we inherit that benefit. One could implement a version to be more specific that would be built on top of MCGP. It could validate certain types of devices by messages in the device values in the Application Layer.

# 5. Security

# 5.1 Security model

The MCGP protocol's security model relies on SSL/TLS with client certificate authentication. The use of SSL/TLS over the TCP connection established by MGCP ensures confidentiality and authenticity, as well as protects against replay attacks (as does TLS).

# 5.2 Authentication method

The MGCP server only accepts connections from clients that present a client certificate during TLS handshake that is signed by the Certificate Authority that the server knows and explicitly allows. The server is then able to make access-decisions based on the Common Name (CN) of the client certificate.

The client is given an option to authenticate in the protocol handshake (not TLS handshake). This authentication phase usually repeats the identifier from Common Name and the server MAY require them to match. However, the server MAY allow certain certificate holders (identified by CN) to impersonate other users. This authentication field allows for this possibility without the need to create extra certificates.

An example scenario would include a special-purpose certificate with CN such as admin that would allow the user of this certificate to provide any identity to the server and have it accepted. The protocol definition is flexible in the implementation of the meaning of these authentication fields and can be made very powerful. A minimum reference implementation SHALL check that the CN of the certificate matches the ident field in the authentication phase.

# 5.3 Solved security challenges

* Server spoofing / MITM attacks - the use of TLS provides sufficient resiliency against MITM attacks.
* Packet tampering - the use of TLS provides sufficient protection against any message tampering.

# 6. Differences

# 6.1 PDU Changes

The payload section was detailed so that each device was send in the payload.  
This allowed the payload to parsed quicker and required less changes.

# 6.2 Security

The authentication and encryption model was reenginerred to benefit from SSL/TLS encryption and authentication instead of the previous, custom-designed MAC algorithm that did not provide any confidentiality.

# 6.2 DFA

As mentioned in our feedback the DFA was simplied to better show the states. It was also updated to match the changes for security.