

# COAP implementation

Elias Berglin

16<sup>th</sup> November 2021

## 1 Creating a message

I created four different function for the different message types GET, POST, PUT, DELETE. Each function takes a uri path and POST and PUT also take a payload as a string. It returns a byte array this array is the finished message.

```
1 func createPut(path string, payload string) []byte
```

The first part of every function is the creation of the header. The process of creating the header is always the same for the given method. Here we can see the creation process of a POST request:

```
1 ret := make([]byte, 0)
2 firstByte := byte(0b01010000)
3 ret = append(ret, firstByte)
4 code := byte(0x02)
5 ret = append(ret, code)
6 r := rand.Uint32()
7 id := make([]byte, 4)
8 binary.BigEndian.PutUint32(id, r)
9 id = id[:2]
10 ret = append(ret, id...)
```

The variable ret is the byte array that is returned. first byte creates the first byte of the message containing the version, message type and token length. In this case we have version 1, type 1 and token length 0. I then append the byte to ret.

The variable code is next containing the method. This gets a value of 2 that is a POST message. This is then appended to the array.

The message id is always randomized. The smallest unsigned int that can be randomized in go is 32bits long and thus is 4 bytes. So I first create a 4 byte long array and put the integer there and last i remove the 2 last bytes creating a 2 byte long random int that is then appended to the array. As I do not include a token this is skipped

Next part is creating options. There is two options that is required first is the URI and then the content type. I created a separate function for this.

```
1 func createOption(name string, data []byte, lastDelta int) ([]byte,
   int) {
2     switch name {
3     case "uri":
4         delta := (11 - lastDelta) << 4
5         lastDelta = 11 - lastDelta
6         l := len(data)
7         header := delta + l
8         return append([]byte{byte(header)}, data...), lastDelta
9     case "contentType":
10        delta := (12 - lastDelta) << 4
11        lastDelta = 12 - lastDelta
12        l := len(data)
13        header := delta + l
14        return append([]byte{byte(header)}, data...), lastDelta
15    }
16    return nil, 0
17 }
```

This function first takes the name of the option. Then it takes the option data as a byte array and lastly it takes the last delta from the previous option. I then have a switch case statement for creating the different options depending in the option name I took as input. The function returns the finished option as an array and an integer representing the last delta. The first thing I do is calculate the option delta and bit shift it 4 times to move the 4 bits to the last 4 bits I then get the length of the option data and add that to the delta finally I return a byte array with the delta and option byte followed by the option data and the calculated delta.

After the options are returned they are appended to the message array in creation function for the message. Here is an example from the POST request:

```
1 lastDelta := 0
2 option, lastDelta := createOption("uri", []byte(path), lastDelta)
3 ret = append(ret, option...)
4 option, _ = createOption("contentType", make([]byte, 0), lastDelta)
5 ret = append(ret, option...)
```

Here we can see the user input as both path and payload. They are a string but here I convert them to byte arrays before sending them in to the createOption function. You can also see that the last delta starts at 0 and is overwritten by the returning int from the function.

Lastly I append the delimiter and as we can see in the following code snippet it is FF in hexa decimal and after that I append the payload that is determined by the user input

```
1 ret = append(ret, byte(0xFF))
2 ret = append(ret, []byte(payload)...)
```

The other functions for GET, PUT and DELETE are similar, the difference is the method bits in the header and the fact that option and delete does not contain a payload.

## 2 Parse a message

## Appendix A Code

```
1      package main
2
3  import (
4      "encoding/binary"
5      "fmt"
6      "math/rand"
7      "net"
8      "os"
9      "os/exec"
10     "strconv"
11     "time"
12 )
13
14 //40 01 04 d2 b4 74 65 73 74
15
16 type COAPOption struct {
17     Delta    int
18     Leangth  int
19     Value    []byte
20 }
21
22 type COAPMessage struct {
23     Version   int
24     T         int
25     TKL      int
26     Code     int
27     MessageID int
28     Token     string
29     Options   []COAPOption
30     Payload   []byte
31     Format     string
32 }
33
34 func parseMethodCode(c int) string {
35     switch c {
36     case 1:
37         return "GET"
38     case 2:
39         return "POST"
40     case 3:
41         return "PUT"
42     case 4:
43         return "DELETE"
44     case 65:
45         return "Created"
46     case 66:
47         return "Deleted"
48     case 67:
49         return "Valid"
50     case 68:
51         return "Changed"
52     case 69:
53         return "Continue"
54     case 132:
55         return "Not Found"
```

```

56 }
57
58 return "not in list: " + strconv.Itoa(c)
59 }
60
61 func parseOptionCode(c int) (string, string) {
62     switch c {
63     case 4:
64         return "Etag", "opaque"
65     case 8:
66         return "Location Path", "string"
67     case 11:
68         return "Uri-path", "string"
69     case 12:
70         return "Content Format", "string"
71     }
72
73     return "Number " + strconv.Itoa(c) + " is not in list", "null"
74 }
75
76 func createOption(name string, data []byte, lastDelta int) ([]byte,
77     int) {
78     switch name {
79     case "uri":
80         delta := (11 - lastDelta) << 4
81         lastDelta = 11 - lastDelta
82         l := len(data)
83         header := delta + 1
84         return append([]byte{byte(header)}, data...), lastDelta
85     case "contentType":
86         delta := (12 - lastDelta) << 4
87         lastDelta = 12 - lastDelta
88         l := len(data)
89         header := delta + 1
90         return append([]byte{byte(header)}, data...), lastDelta
91     }
92
93     return nil, 0
94 }
95
96 func parseOptionsHeader(header byte) (int, int) {
97     delta := int((header & 0b11110000) >> 4)
98     len := int(header & 0b00001111)
99     return delta, len
100 }
101
102 func parseOptions(arr []byte) []COAPOption {
103     var options []COAPOption
104     cursor := 0
105     lastDelta := 0
106     for cursor < len(arr) {
107         delta, len := parseOptionsHeader(arr[cursor])
108         lastDelta += delta
109         cursor++
110         var val []byte
111         if len != 0 {
112             val = arr[cursor : cursor+len]

```

```

112     }
113     temp := COAPOption{
114         Delta:    lastDelta,
115         Leangth:  len,
116         Value:    val,
117     }
118     options = append(options, temp)
119     cursor += len
120
121 }
122 return options
123 }
124
125 func parseMessage(arr []byte, n int) COAPMessage {
126     message := COAPMessage{
127         Version:  int((arr[0] & 0b11000000) >> 6),
128         T:         int((arr[0] & 0b00110000) >> 4),
129         TKL:       int(arr[0] & 0b00001111),
130         Code:      int(arr[1]),
131         MessageID: int(binary.BigEndian.Uint16([]byte{arr[2], arr[3]})),
132     }
133     var index int
134     for i, b := range arr {
135         if b == 0xFF {
136             index = i
137         }
138     }
139     if index == 0 {
140         index = n
141     }
142
143     optionStart := 4
144
145     if message.TKL != 0 {
146         message.Token = string(arr[4 : 4+message.TKL])
147         optionStart = 5 + message.TKL
148     }
149
150     optionByte := arr[optionStart:index]
151     message.Options = parseOptions(optionByte)
152
153     if index != n {
154         message.Payload = arr[index+1 : n]
155     }
156
157     return message
158 }
159
160 func createGet(path string) []byte {
161     var ret []byte
162     firstByte := byte(0b01010000)
163     ret = append(ret, firstByte)
164     code := byte(0x01)
165     ret = append(ret, code)
166     r := rand.Uint32()
167     id := make([]byte, 4)
168     binary.BigEndian.PutUint32(id, r)

```

```

169 id = id[:2]
170 ret = append(ret, id...)
171 lastDelta := 0
172 option, lastDelta := createOption("uri", []byte(path), lastDelta)
173 ret = append(ret, option...)
174 option, _ = createOption("contentType", make([]byte, 0), lastDelta
175 )
176 ret = append(ret, option...)
177 return ret
178 }
179
180 func createPost(path string, payload string) []byte {
181 ret := make([]byte, 0)
182 firstByte := byte(0b01010000)
183 ret = append(ret, firstByte)
184 code := byte(0x02)
185 ret = append(ret, code)
186 r := rand.Uint32()
187 id := make([]byte, 4)
188 binary.BigEndian.PutUint32(id, r)
189 id = id[:2]
190 ret = append(ret, id...)
191 lastDelta := 0
192 option, lastDelta := createOption("uri", []byte(path), lastDelta)
193 ret = append(ret, option...)
194 option, _ = createOption("contentType", make([]byte, 0), lastDelta
195 )
196 ret = append(ret, option...)
197 ret = append(ret, byte(0xFF))
198 ret = append(ret, []byte(payload)...)
199 return ret
200 }
201
202 func createPut(path string, payload string) []byte {
203 ret := make([]byte, 0)
204 firstByte := byte(0b01010000)
205 ret = append(ret, firstByte)
206 code := byte(0x03)
207 ret = append(ret, code)
208 r := rand.Uint32()
209 id := make([]byte, 4)
210 binary.BigEndian.PutUint32(id, r)
211 id = id[:2]
212 ret = append(ret, id...)
213 lastDelta := 0
214 option, lastDelta := createOption("uri", []byte(path), lastDelta)
215 ret = append(ret, option...)
216 option, _ = createOption("contentType", make([]byte, 0), lastDelta
217 )
218 ret = append(ret, option...)
219 ret = append(ret, byte(0xFF))
220 ret = append(ret, []byte(payload)...)
221 return ret
222 }

```

```

223
224 func createDelete(path string) []byte {
225     ret := make([]byte, 0)
226     firstByte := byte(0b01010000)
227     ret = append(ret, firstByte)
228     code := byte(0x04)
229     ret = append(ret, code)
230     r := rand.Uint32()
231     id := make([]byte, 4)
232     binary.BigEndian.PutUint32(id, r)
233     id = id[:2]
234     ret = append(ret, id...)
235     lastDelta := 0
236     option, lastDelta := createOption("uri", []byte(path), lastDelta)
237     ret = append(ret, option...)
238     option, _ = createOption("contentType", make([]byte, 0), lastDelta)
239     ret = append(ret, option...)
240
241     return ret
242 }
243
244 func printOptions(options []COAPOption) string {
245     format := ""
246     for _, option := range options {
247         optionName, optionFormat := parseOptionCode(option.Delta)
248         if optionName == "Content Format" {
249             if len(option.Value) == 0 {
250                 format = "text/plain"
251                 optionFormat = "CF"
252             }
253         }
254         fmt.Print("\t" + optionName + ": ")
255         switch optionFormat {
256             case "string":
257                 fmt.Println(string(option.Value))
258             case "CF":
259                 fmt.Println(format)
260             case "opaque":
261                 fmt.Println(option.Value)
262         }
263     }
264     return format
265 }
266
267 func printCOAP(c COAPMessage) {
268     fmt.Println("Version:", c.Version, "Message Type:", c.T, "Token",
269         "length:", c.TKL)
270     fmt.Println("Metod/Response:", "\""+parseMethodCode(c.Code)+"\"",
271         "Message id:", c.MessageID)
272     fmt.Println("Token:", c.Token)
273     fmt.Println("Options:")
274     c.Format = printOptions(c.Options)
275     if c.Format != "text/plain" {
276         fmt.Println("FIX FORMAT")
277     } else {

```



```

277     fmt.Println("Payload:\n", "\t"+string(c.Payload))
278 }
279 }
280
281 func sendCOAP(method string) {
282     conn, err := net.Dial("udp", "coap.me:5683")
283     if err != nil {
284         panic(err)
285     }
286     var msg []byte
287     var uri string
288     fmt.Println("Type endpoint")
289     fmt.Scanln(&uri)
290     switch method {
291     case "GET":
292         msg = createGet(uri)
293         break
294     case "POST":
295         var payload string
296         fmt.Println("Type payload")
297         fmt.Scanln(&payload)
298         msg = createPost(uri, payload)
299         break
300     case "PUT":
301         var payload string
302         fmt.Println("Type payload")
303         fmt.Scanln(&payload)
304         msg = createPut(uri, payload)
305         break
306     case "DELETE":
307         msg = createDelete(uri)
308         break
309     default:
310         msg = make([]byte, 0)
311         break
312     }
313
314     fmt.Println("Created following Message:")
315     fmt.Println("-----")
316     printCOAP(parseMessage(msg, len(msg)))
317     fmt.Println("-----")
318     _, err = conn.Write(msg)
319     if err != nil {
320         panic(err)
321     }
322     response := make([]byte, 1024)
323     n, err := conn.Read(response)
324     if err != nil {
325         panic(err)
326     }
327     response = response[:n]
328     fmt.Print("\n")
329     fmt.Println("Got following response:")
330     fmt.Println("-----")
331     printCOAP(parseMessage(response, n))
332     fmt.Println("-----")
333     fmt.Println("Press any key to continue")

```

```
334 fmt.Scanln()
335 cmd := exec.Command("clear") //Linux example, its tested
336 cmd.Stdout = os.Stdout
337 cmd.Run()
338
339 }
340
341 func main() {
342     rand.Seed(time.Now().UnixMicro())
343     var option string
344     run := true
345     /* msg := createGet()c
346     printCOAP(parseMessage(msg, len(msg))) */
347     for run {
348         fmt.Println("Select message to send :")
349         fmt.Println("1: GET")
350         fmt.Println("2: POST")
351         fmt.Println("3: PUT")
352         fmt.Println("4: DELETE")
353         fmt.Println("Any other key to exit")
354         fmt.Scanln(&option)
355
356         switch option {
357             case "1":
358                 sendCOAP("GET")
359             case "2":
360                 sendCOAP("POST")
361             case "3":
362                 sendCOAP("PUT")
363             case "4":
364                 sendCOAP("DELETE")
365             default:
366                 run = false
367             }
368         option = ""
369     }
370 }
```