COAP implementation

Elias Berglin

14th December 2021

1 Creating a message

I created four different functions 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.

```
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:

```
ret := make([]byte, 0)
firstByte := byte(0b01010000)
ret = append(ret, firstByte)
code := byte(0x02)
ret = append(ret, code)
r := rand.Uint32()
id := make([]byte, 4)
binary.BigEndian.PutUint32(id, r)
id = id[:2]
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 are two options that is required first is the URI and then the content type. I created a separate function for this.

```
func createOption(name string, data []byte, lastDelta int) ([]byte,
    int) {
   switch name {
    case "uri":
     delta := (11 - lastDelta) << 4
     lastDelta = 11 - lastDelta
     l := len(data)
     header := delta + l
     return append([]byte{byte(header)}, data...), lastDelta
    case "contentType":
     delta := (12 - lastDelta) << 4
     lastDelta = 12 - lastDelta
     l := len(data)
     header := delta + l
     return append([]byte{byte(header)}, data...), lastDelta
 }
 return nil, 0
}
```

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:

```
lastDelta := 0
option, lastDelta := createOption("uri", []byte(path), lastDelta)
ret = append(ret, option...)
option, _ = createOption("contentType", make([]byte, 0), lastDelta)
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 thus is determined by the user input

```
    ret = append(ret, byte(0xFF))
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

To parse the message, I use two functions. One for parsing the message and one for parsing the options. For the main parsing function, I receive a byte array containing the message and the number of bytes it contains. I return an instance of a COAP message struct.

```
message := COAPMessage{
 Version:   int((arr[0] & 0b11000000) >> 6),
 T:         int((arr[0] & 0b00110000) >> 4),
 TKL:       int(arr[0] & 0b00001111),
 Code:      int(arr[1]),
 MessageID: int(binary.BigEndian.Uint16([]byte{arr[2], arr[3]})),
}
```

First, I create an instance of the message and assign the fixed values. I do this by doing an and on the byte with a bit string containing ones on the bits where the data I am interested in getting. I then shift the resulting bits to place them so their value is correct.

When the set bits are done, I then try to find the delimiter byte (FF in hex). I store the value so I can take out the subset of bits containing the options.

```
var index int
for i, b := range arr {
 if b == 0xFF {
  index = i
 }
}
if index == 0 {
 index = n
}
```

If I do not find the delimiter, I set the index of the final bit of the options to the end of the whole message because I know the rest of the message is only options.

If token length is 0 then I set the start of options to the fifth byte otherwise I set it to the byte after token. I then send the subset of the array to a function that decodes the options. In that function I start the delta on 0 and add to it with every option and then save that to a struct representing an option. For now I do nothing with the option data and just save as a byte array.

3 Printing the message

The printing of a message is done by a void function taking an instance of the COAP message class.

```
func printCOAP(c COAPMessage) {
 fmt.Println("Version:", c.Version, "Message Type:", c.T, "Token
     leangth:", c.TKL)
 fmt.Println("Metod/Response:", "\""+parseMethodCode(c.Code)+"\"",
     "Message id:", c.MessageID)
 fmt.Println("Token:", c.Token)
 fmt.Println("Options:")
 c.Format = printOptions(c.Options)
 if c.Format != "text/plain" {
  fmt.Println("FIX FORMAT")
 } else {
  fmt.Println("Payload:\n", "\t"+string(c.Payload))
 }
}
```

Helper functions help the main function handle options and converting identification number to human readable strings. The complete code can be found in Appendix A

Appendix A Code

```
            package main

import (
 "encoding/binary"
 "fmt"
 "math/rand"
 "net"
 "os"
 "os/exec"
 "strconv"
 "time"
)

//40 01 04 d2 b4 74 65 73 74

type COAPOption struct {
 Delta   int
 Leangth int
 Value   []byte
}

type COAPMessage struct {
 Version   int
 T         int
 TKL       int
 Code      int
 MessageID int
 Token     string
 Options   []COAPOption
 Payload   []byte
 Format    string
}

func parseMethodCode(c int) string {
 switch c {
 case 1:
  return "GET"
 case 2:
  return "POST"
 case 3:
  return "PUT"
 case 4:
  return "DELETE"
 case 65:
  return "Created"
 case 66:
  return "Deleted"
 case 67:
  return "Valid"
 case 68:
  return "Changed"
 case 69:
  return "Continue"
 case 132:
  return "Not Found"
```

```
 }

 return "not in list: " + strconv.Itoa(c)
}

func parseOptionCode(c int) (string, string) {
 switch c {
 case 4:
  return "Etag", "opaque"
 case 8:
  return "Location Path", "string"
 case 11:
  return "Uri-path", "string"
 case 12:
  return "Content Format", "string"
 }

 return "Number " + strconv.Itoa(c) + " is not in list", "null"
}

func createOption(name string, data []byte, lastDelta int) ([]byte,
     int) {
 switch name {
 case "uri":
  delta := (11 - lastDelta) << 4
  lastDelta = 11 - lastDelta
  l := len(data)
  header := delta + l
  return append([]byte{byte(header)}, data...), lastDelta
 case "contentType":
  delta := (12 - lastDelta) << 4
  lastDelta = 12 - lastDelta
  l := len(data)
  header := delta + l
  return append([]byte{byte(header)}, data...), lastDelta
 }

 return nil, 0
}

func parseOptionsHeader(header byte) (int, int) {
 delta := int((header & 0b11110000) >> 4)
 len := int(header & 0b00001111)
 return delta, len
}

func parseOptions(arr []byte) []COAPOption {
 var options []COAPOption
 cursor := 0
 lastDelta := 0
 for cursor < len(arr) {
  delta, len := parseOptionsHeader(arr[cursor])
  lastDelta += delta
  cursor++
  var val []byte
  if len != 0 {
   val = arr[cursor : cursor+len]
```

```
  }
  temp := COAPOption{
   Delta:   lastDelta,
   Leangth: len,
   Value:   val,
  }
  options = append(options, temp)
  cursor += len

 }
 return options
}

func parseMessage(arr []byte, n int) COAPMessage {
 message := COAPMessage{
  Version:   int((arr[0] & 0b11000000) >> 6),
  T:         int((arr[0] & 0b00110000) >> 4),
  TKL:       int(arr[0] & 0b00001111),
  Code:      int(arr[1]),
  MessageID: int(binary.BigEndian.Uint16([]byte{arr[2], arr[3]})),
 }
 var index int
 for i, b := range arr {
  if b == 0xFF {
   index = i
  }
 }
 if index == 0 {
  index = n
 }

 optionStart := 4

 if message.TKL != 0 {
  message.Token = string(arr[4 : 4+message.TKL])
  optionStart = 5 + message.TKL
 }

 optionByte := arr[optionStart:index]
 message.Options = parseOptions(optionByte)

 if index != n {
  message.Payload = arr[index+1 : n]
 }

 return message
}

func createGet(path string) []byte {
 var ret []byte
 firstByte := byte(0b01010000)
 ret = append(ret, firstByte)
 code := byte(0x01)
 ret = append(ret, code)
 r := rand.Uint32()
 id := make([]byte, 4)
 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  ret = append(ret, option...)
176
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  ret = append(ret, option...)
196  ret = append(ret, byte(0xFF))
197  ret = append(ret, []byte(payload)...)
198
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  ret = append(ret, option...)
218  ret = append(ret, byte(0xFF))
219  ret = append(ret, []byte(payload)...)
220
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  }
265  return format
266 }
267
268 func printCOAP(c COAPMessage) {
269  fmt.Println("Version:", c.Version, "Message Type:", c.T, "Token
         leangth:", c.TKL)
270  fmt.Println("Metod/Response:", "\""+parseMethodCode(c.Code)+"\"",
         "Message id:", c.MessageID)
271  fmt.Println("Token:", c.Token)
272  fmt.Println("Options:")
273  c.Format = printOptions(c.Options)
274  if c.Format != "text/plain" {
275   fmt.Println("FIX FORMAT")
276  } 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 }
```