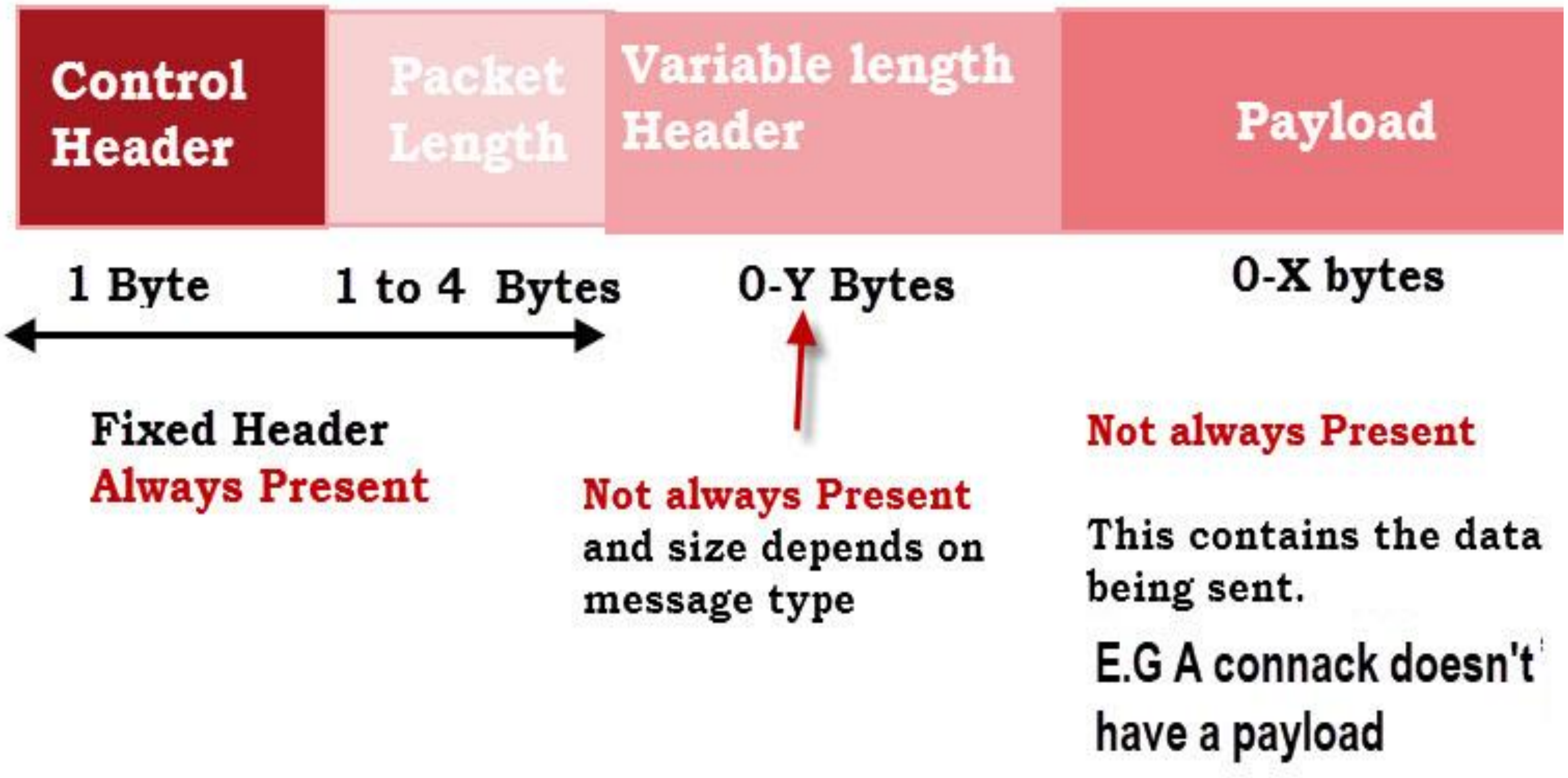
Internet of Things
IO 4041
Application Layer Protocols

MQTT packet structure



Control Header

Packet Length

Variable length Header

Payload

1 Byte

1 to 4 Bytes

0-Y Bytes

0-X bytes

Fixed Header
Always Present

Not always Present
and size depends on
message type

Not always Present

This contains the data being sent.

E.G A connack doesn't have a payload

MQTT Standard Packet Structure

MQTT: Remaining Length

Digits	From	To
1	0 (0x00)	127 (0x7F)
2	128 (0x80, 0x01)	16 383 (0xFF, 0x7F)
3	16 384 (0x80, 0x80, 0x01)	2 097 151 (0xFF, 0xFF, 0x7F)
4	2 097 152 (0x80, 0x80, 0x80, 0x01)	268 435 455 (0xFF, 0xFF, 0xFF, 0x7F)

MQTT: Remaining Length

The algorithm for encoding a decimal number (X) into the variable length encoding scheme is as follows

```
do
digit = X MOD 128 (% in C)
X = X DIV 128 (/ in C)
// if there are more digits to encode, set the top bit of this digit
if(X>0)
        digit = digit OR 0x80 (bitwise or | in C)
endif
Output digit
while(X>0)
```

Encoding Example:

Encoding of a number X = 8560

X	digit (X % 128)	X = X / 128	if (X>0) digit = digit OR 0x80	Print digit	while(X> 0)
8560	112	66	(240) 11110000	(240) 11110000	yes
66	66	0	NA	(66) 01000010	No

MQTT: Remaining Length

The algorithm for decoding the Remaining Length field

```
multiplier = 1
value = 0
do
    digit = 'next digit from stream’
    value += (digit AND 127) * multiplier
    multiplier *= 128
while((digit AND 128) != 0)
```

Decoding: Remaining Length

decoding remaining length i.e., (212) **11110000** (66) **01000010 ---------**8560

Multiplier	value	digit = 'next digit from stream	value += (digit AND 127) * multiplier	multiplier *= 128	while ((digit AND 128) != 0)
1	0	(240) 11110000	(112) 01110000	(128) = 10000000	yes
(128) = 10000000	(112) 01110000	(66) 01000010	112 +66 * 128 = 112+ 8448 = 8560	16384	No

MQTT: Variable (Optional) header

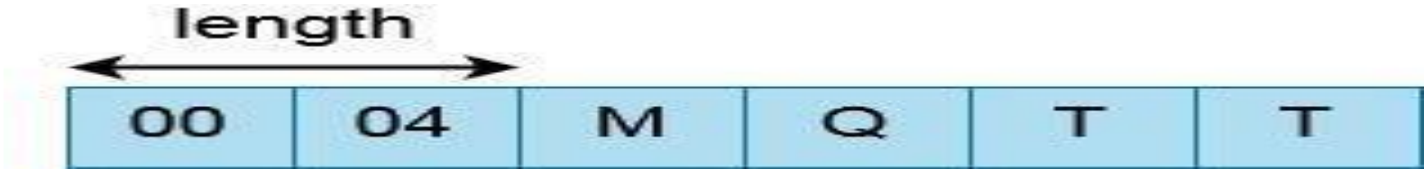
- Resides between the fixed header and the payload
- Contained in some types of MQTT command messages

Many fields like

  - Protocol name
  - Protocol version
  - Connect flags
  - Username and password flags

MQTT: Variable (Optional) header

- **Protocol name**
  - 1st 2 bytes mention the length of protocol name followed by protocol name
  - MQTT has length equal to 4, so it will be



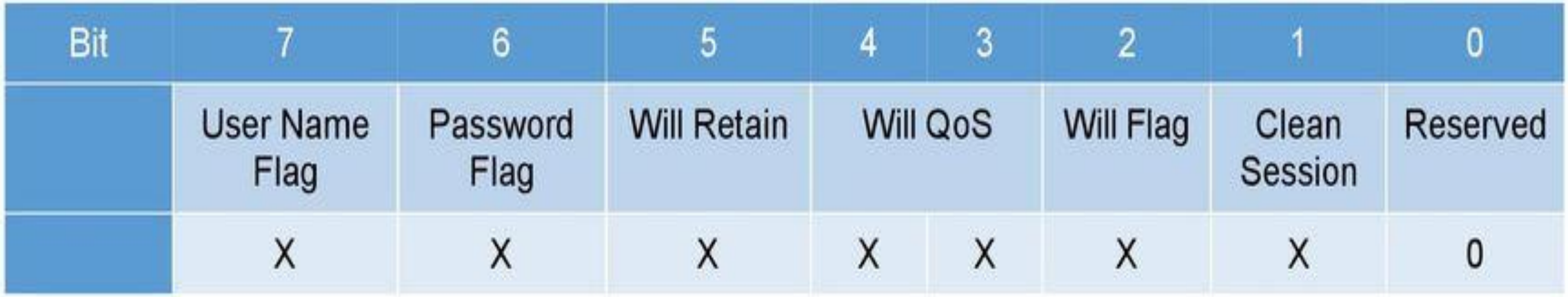
00	04	M	Q	T	T

- **Protocol level**
  - Shows the version of the protocol
  - For MQTT. 3.1.1: level is 4

MQTT: Variable (Optional) header

Connect flags byte

- **Bit 0** is reserved for future use
- The Clean session (**bit 1**):
  - If not set (0), server must store subscription of clients after it disconnects (QoS 1 and QoS 2)
  - If set (1), then the server must discard any previously maintained information about the client and treat the connection as "clean"

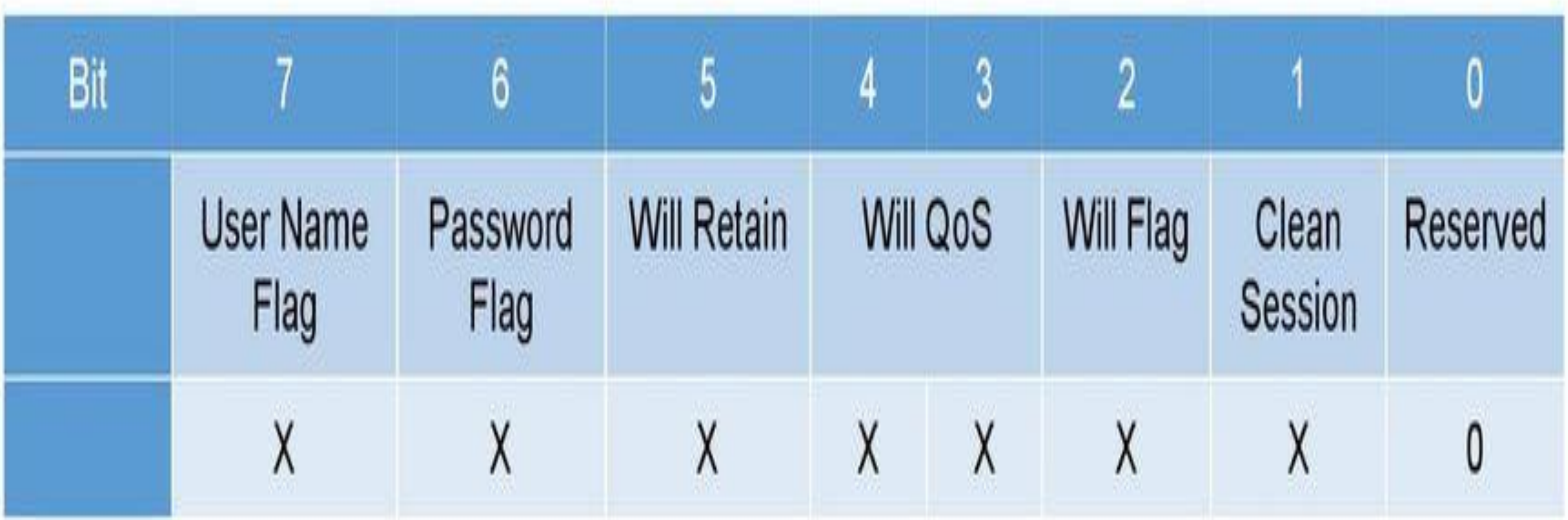


Bit	7	6	5	4	3	2	1	0
	User Name Flag	Password Flag	Will Retain	Will QoS		Will Flag	Clean Session	Reserved
	X	X	X	X	X	X	X	0

MQTT: Variable (Optional) header

Connect flags byte

- Will, Will QoS, and Retain flags are present in the variable header of a CONNECT message (**bit 2 to 5)**



Bit	7	6	5	4	3	2	1	0
	User Name Flag	Password Flag	Will Retain	Will QoS		Will Flag	Clean Session	Reserved
	X	X	X	X	X	X	X	0

MQTT: Variable (Optional) header

Keep alive

- two bytes are used to mention the keep alive duration in seconds.
    - For 60 seconds, the value will be 003C in hex.
- defines the maximum time interval between messages received from a client
- The client has a responsibility to send a message within each Keep Alive time period
- In the absence of a data-related message during the time period,
    - the client sends a PINGREQ message,
    - which the server acknowledges with a PINGRESP message.

MQTT: Variable (Optional) header

Keep alive

- If the server does not receive a message from the client within one and a half times the Keep Alive time period
  - it disconnects the client as if the client had sent a DISCONNECT message.
- If a client does not receive a PINGRESP message within a Keep Alive time period after sending a PINGREQ,
  - it should close the TCP/IP socket connection.

MQTT: payload

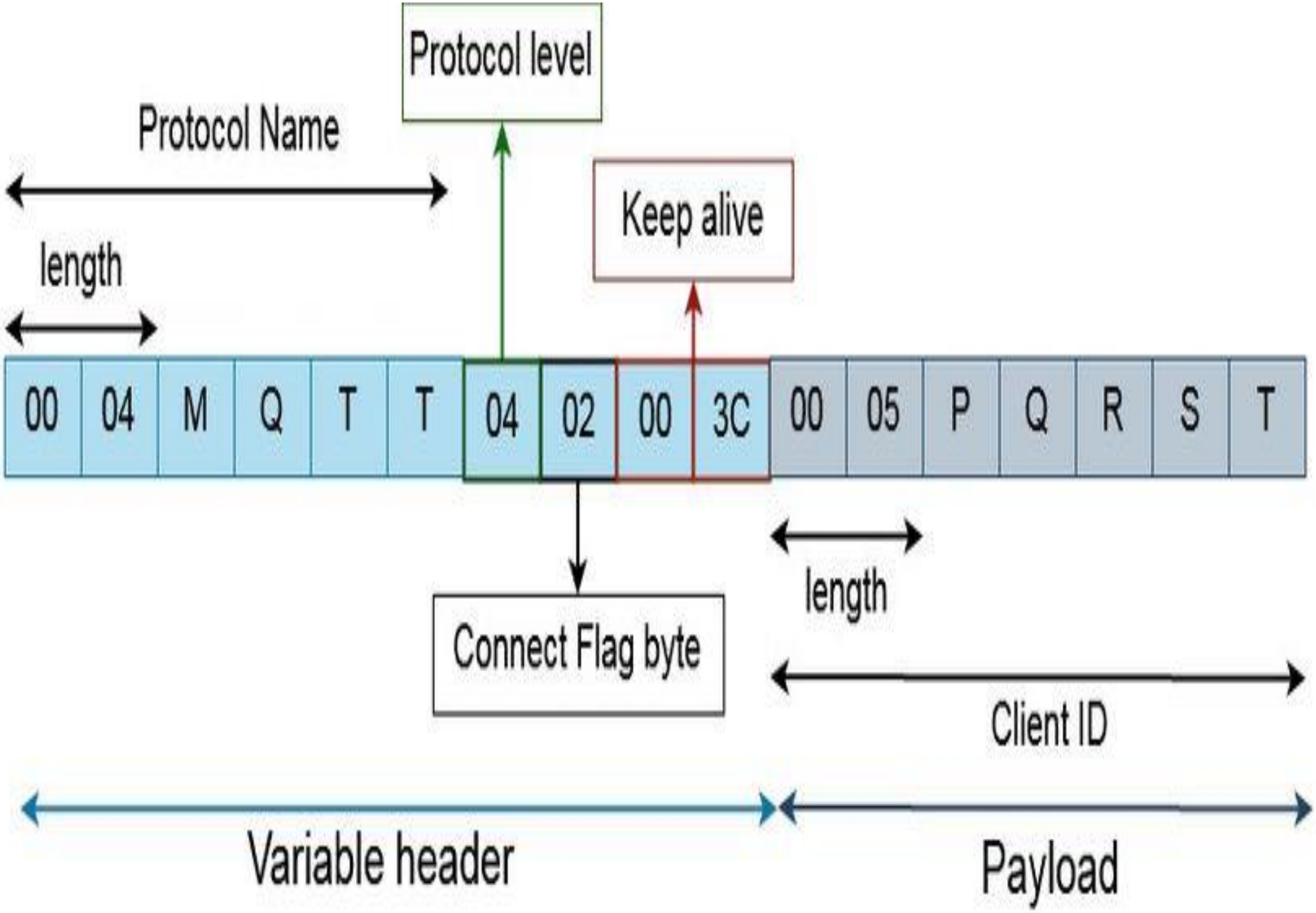
The following types of MQTT command message have a payload

- CONNECT
  - Here, payload is client ID and
  - ‘username and password’ if they are present
- SUBSCRIBE
- SUBACK
- PUBLISH
  - message to be published

Example: Connect Packet

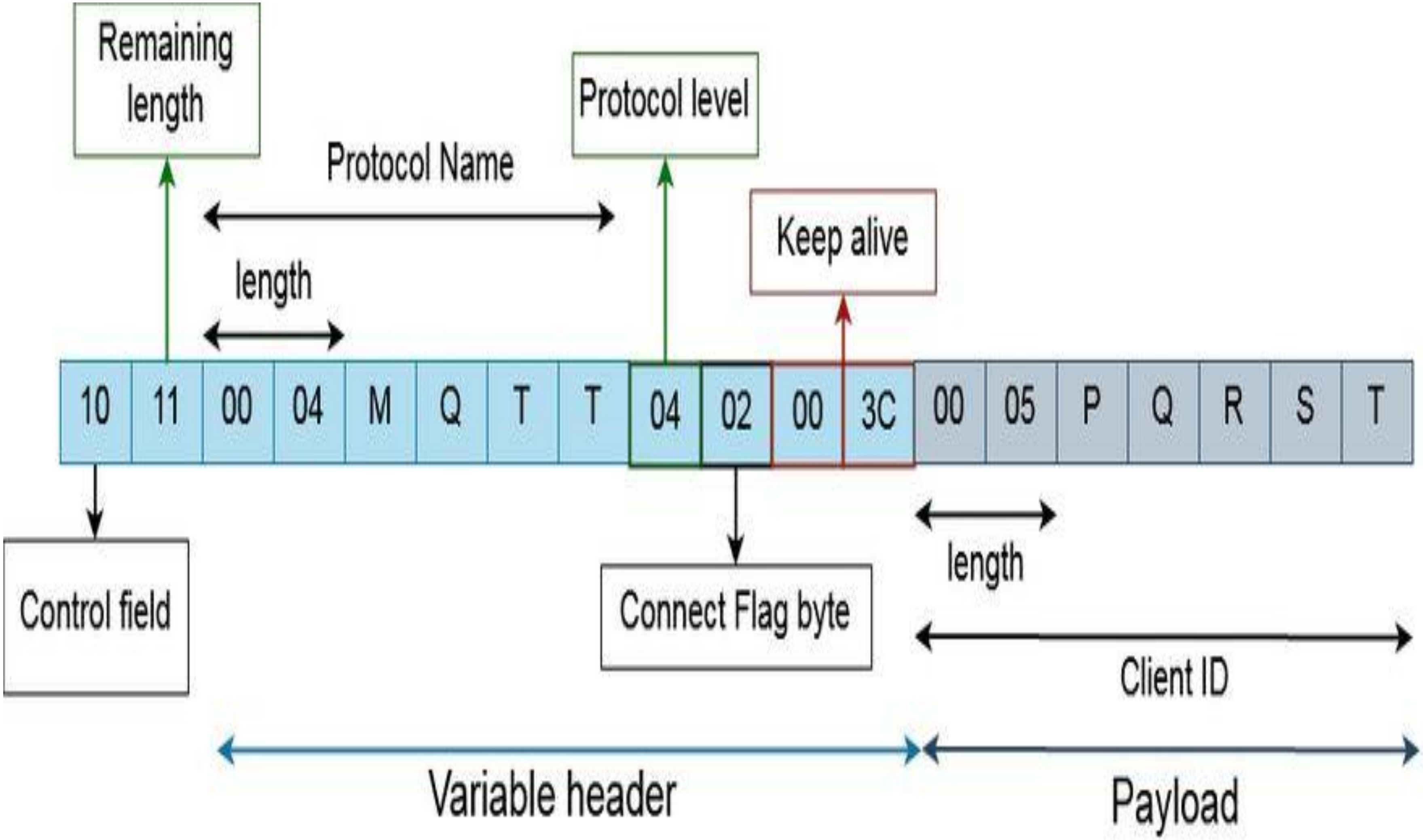
Frist byte will be 10 i.e., 00010000
Second byte will be remaining length (to be calculated from variable header and payload)

Example: Connect Payload



00	04	M	Q	T	T	04	02	00	3C	00	05	P	Q	R	S	T

Example: Connect Packet



Remaining length

Protocol Name

Protocol level

length

Keep alive

10	11	00	04	M	Q	T	T	04	02	00	3C	00	05	P	Q	R	S	T

Control field

Connect Flag byte

length

Client ID

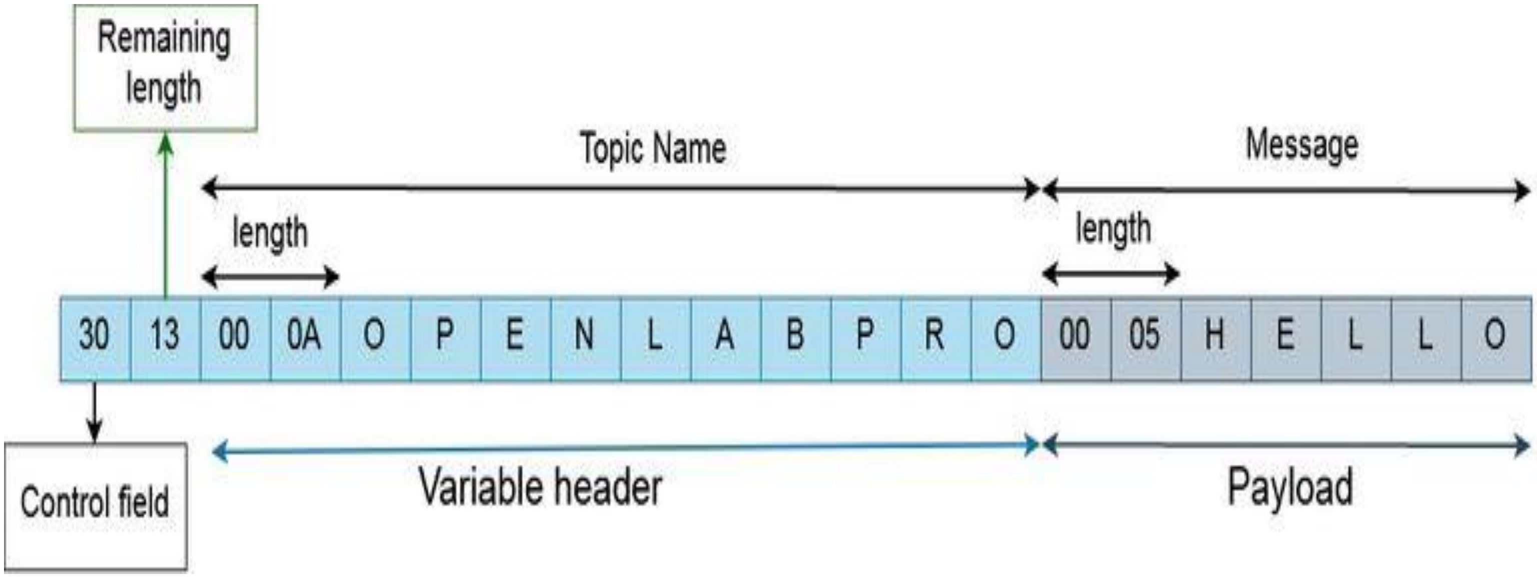
Variable header

Payload

Example: Publish Packet

publish the message “HELLO” to the topic OPENLABPRO

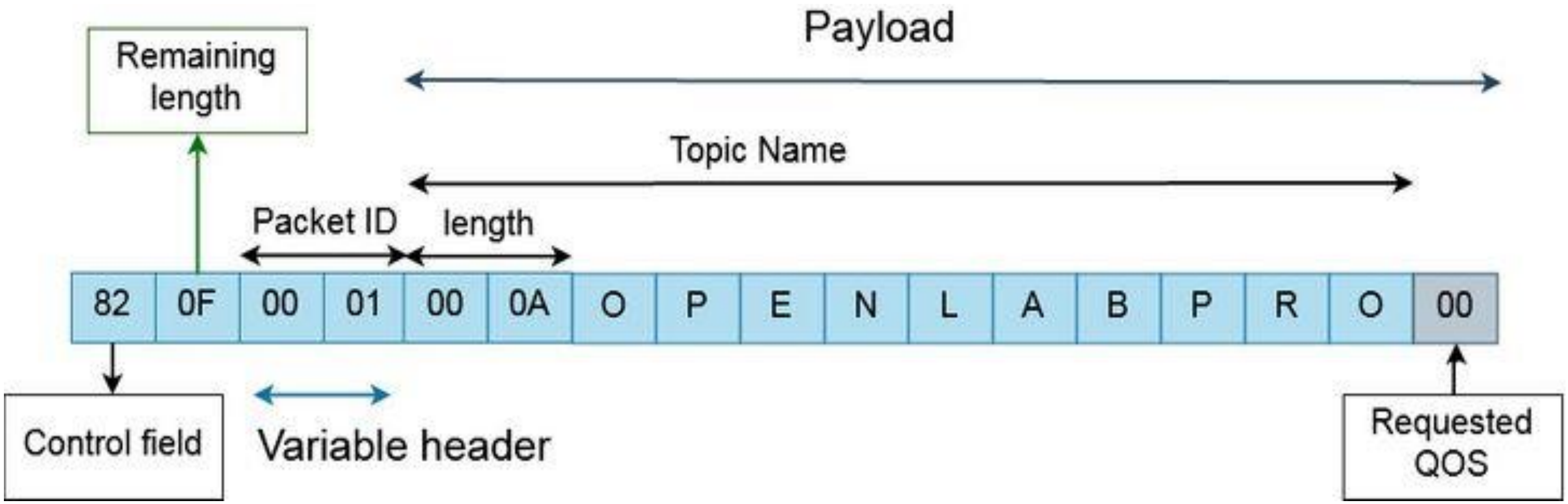
- In the variable header section first 2 bytes will denote the length of the topic and then followed by topic.
- In payload section first 2 bytes will denote the length of the message which is followed by the message.



30	13	00	0A	O	P	E	N	L	A	B	P	R	O	00	05	H	E	L	L	O

Example: SUBSCRIBE Packet

- Command value of Subscribe packet is 8 and the Control flag is reserved and should be 2
- The variable header will contain a non-zero 16-bit packet ID
- As payload, there will be the topic to subscribe followed by requested QOS level



82	0F	00	01	00	0A	O	P	E	N	L	A	B	P	R	O	00

Summary

- Works mainly as a pipe for binary data and provides a flexibility in communication patterns
- A publish-subscribe messaging protocol with most possible minimal bandwidth requirements
- Uses TCP for transport
- MQTT is an open standard, giving a mechanisms to asynchronous communication,
- Have a range of implementations, and it is working on IP.
- Widely used protocol
    - used by famous corporations such as, Amazon and Facebook

MQTT

- used in devices with restricted memory capabilities and limited processing power
- connects the networks and devices with middleware and applications
- This connection uses machine-to server (M2S), server-to-server (S2S), machine-to-machine communication patterns, and
- routing mechanism (one-to-many, one-to-one, many-to-many).
- most favorable connection protocol for M2M and IoT

MQTT: Broker

- The default MQTT port that worked on is TCP/IP port.1883
- Has different types, such as, mosquito, hivemq, and paho MQTT
- MQTT built on the upper of TCP protocol as the Hypertext Transfer Protocol (HTTP).
- However, it is designed to have a less protocol overhead than HTTP
- delivers the messages using three QoS levels

COnstrained Application Protocol (CoAP)

CoAP

- A specialized web transfer protocol for use with constrained nodes and constrained (e.g., low-power, lossy) networks.
- Nodes often have 8-bit microcontrollers with small amounts of ROM and RAM, while
- constrained networks such 6LoWPANs often have high packet error rates and a typical throughput of 10s of kbit/s.
- designed for machine- to-machine (M2M) applications such as smart energy and building automation

CoAP

- provides a request/response interaction model between application endpoints,
- supports built-in discovery of services and resources, and includes key concepts of the Web such as URIs and Internet media types.
- designed to easily interface with HTTP for integration with the Web while meeting specialized requirements
  - such as multicast support, very low overhead, and simplicity for constrained environments.
  - supports the basic methods of **GET, POST, PUT, DELETE**, which are easily mapped to HTTP