Security tool for IOT and IMAGE compression techniques

[a]S.Ramana, M Pavan Kumar[b], N Bhaskar[c], S. China Ramu[d], G.R. Ramadevi[e]
[a]Research Scholar, Dept. of Computer Science, Osmania University, Hyderabad, Telangana, India
[b]Student, Dept. of Electronics and communication Engineering, MVSR Engineering College, Nahergul, Telangana
[c]Research Scholar, Dept. of Computer Science, Rayalaseema University Kurnool, A.P, India.
[d]Dept. of CSE, CBIT Hyderabad, Telangana, India
[e]Dept. of CSE, CBIT Hyderabad, Telangana, India

Abstract

A new era of computation has began with wide spread because of its ease of use and advantages in human kind that is IoT(Internet of Things). IoT is used in many applications like greenhouse, telemedicine monitoring, smart farming etc.

Construction of IoT systems requires a perfect infrastructure planning. Moreover, management and security of these systems are considered to be the most primary and vital challenges by system developers.

IoT is the interconnection of electronic devices and software. The devices which are connected in the network will have different sensors which are used for data collection. Each sensor will monitor a specific condition such as location, vibration, motion, temperature and visual data. Sensors of a device communicate over an IP Network with other devices. IoT-enabled devices will share information about their conditions with software systems, and other machines. This information can be shared in real time or they can be collected and shared at desired intervals. Due to IoT enabled devices, everything will have a digital identity and connectivity, which means that, one can identify, track and communicate with the devices.

Machine-to-Machine (M2M) communication is drawn from the IOT-enabled devices in the network to allow business to automate certain basic tasks without depending on central or cloud-based applications and services. The number of devices, or nodes, that are connected in the network are bulk in IoT than in traditional systems.

This paper presents the Security solutions for overcoming the challenges faced in storage and transmission of big data images through compression which are used for IoT networks through a lightweight protocol called as MQTT (Message Queuing Telemetry Transport) protocol.

KEYWORDS : Compression , Big Data, Images, Internet of Things (IoT), Machine-to-Machine Communication, MQTT

----

1. Introduction

The Internet of Things (IoT) is turning out to be an emerging discussion in the field of research and practical implementation in the recent years. IoT is a model that includes ordinary entities with the capability to sense and communicate with fellow devices using Internet. In the present scenario, the broadband Internet is generally accessible and its cost of connectivity is also reduced, more gadgets and sensors are getting connected to it . Such conditions are providing suitable ground for the growth of IoT. There is a great deal of complexities around the IoT, since we wish to approach each and every object from any where in the world . The sophisticated chips and sensors are embedded in the physical things that surround us to transmit the valuable data. The process of sharing such large amount of data begins with the devices themselves that must securely communicate with the IoT devices. This platform integrates the data from many devices.
IoT has five main features which are as follows:

a) sensing
b) information processing
c) heterogeneous access
d) services
e) security and privacy

Recently, the IoT term is also being called as machine to-machine communication or cyber-physical systems.

The architecture of IoT contains an important data communication tool, which is called as Radio Frequency Identification (RFID) along with some complex computational items.

Another definition of IoTcan be defined as an universal network infrastructure communicating with different types of objects through the utilization of sensing data and communication capabilities. Existing Internet and network tools are embedded in this infrastructure. The system will provide the features like object detection, sensor, actuator and connection capability as the basis for the development of independent services and applications .

With reference to the security issue, several challenges obstruct the progress of IoT applications. They are :

a) extension of IoT to collect recent technologies such as sensor network and mobile network

b) the internet will comprise of the passive and active things
c) It is compulsory to communicate these things.

Upon these issues of IoT, new security problems will arise. More attention to the research is required on IoT authenticity, confidentiality, and integrity of data should be considered.

Image compression has become a primary responsibility for transmitting data over the network to overcome the congestion problems. The Compressed Image should be securely transmitted across different IoT devices in the network.

MQTT (Message Queuing Telemetry Transport) protocol is best for communication between constrained devices, because it is lightweight messaging protocol, and can be transmitted in low-bandwidth, high-latency or unreliable networks. It uses publish/subscribe communication pattern. This protocol is ideal for the emerging “machine-to-machine” (M2M) or “Internet of Things” world of connected devices, and for mobile applications where bandwidth and battery power are at a premium.

2. MQTT Protocol

MQTT protocol was invented by Andy Stanford-Clark (IBM) and Arlen Nipper during 1999, when their aim was to create a protocol for minimal battery loss and minimal bandwidth connecting oil pipelines over satellite connection. They specified the following goals, the future protocol should have:

- Simple to implement
- Provide a Quality of Service Data Delivery
- Lightweight and Bandwidth Efficient
- Data Agnostic
- Continuous Session Awareness

MQTT is a publish/subscribe and Client/Server messaging protocol. It is used to transport the message in light weight, open source, simple, and designed to implement in easy manner. These characteristics make it unique for use in many circumstances including constrained environments such as for communication in Machine-to-Machine (M2M) and Internet of Things (IoT) contexts where a small code is required and/or network bandwidth is at a premium.

The protocol uses the architecture of publish/subscribe to compare with HTTP with its request and response paradigm. Publish/Subscribe is event-driven and enables messages to be pushed to clients. The centric communication is MQTT broker, it is in charge of dispatching all messages between the senders and the receivers. Each client that wants to receive messages subscribes to a certain topic and the broker delivers all messages with relation to the topic of the client. So, the clients don’t have to know each other, they only communicate over the topic. This architecture enables highly scalable solutions without dependent on data producers and the data consumers.

Each client that publishes a message to the broker, includes a topic into the message. The topic is the routing information related to the broker.

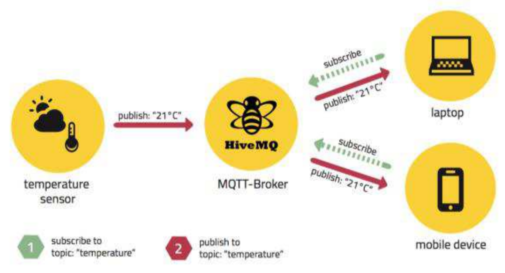


Fig. 2.1 MQTT Publish/Subscribe Architecture

The difference to HTTP is that a client doesn't have to pull the information it needs, but the broker pushes the information to the client, in the case there is something new. Therefore each MQTT client has a permanently open TCP connection to the broker. If this connection is interrupted by any circumstances, the MQTT broker can buffer all messages and send them to the client when it is back to online. As mentioned before the central concept in MQTT to dispatch message as topic. A topic is a simple string that can have more hierarchy levels, which are separated by a slash. A sample topic for sending temperature data of the living room could be ***house/living-room/temperature***. On one hand the client can subscribe to the exact topic or on the other hand use a wildcard. The

subscription to ***house/+/temperature*** would result in all message send to the previously mentioned topic ***house/living-room/temperature*** as well as any topic with an arbitrary value in the place of living room. The plus sign is a single level wild card and only allows arbitrary values for one hierarchy. If you need to subscribe to more than one level, for example to the entire subtree, there is also a multilevel wildcard (***#***). It allows to subscribe to all underlying hierarchy levels. For example ***house/#*** is subscribing to all topics beginning with ***house***.

In order to make the subsequent code more understandable, we will use the transferring of sensor data from a temperature and brightness sensor to a control center over the internet as an example. The sensors will be connected to a Raspberry Pi, which acts as gateway to the MQTT broker, which resides in the cloud. On the other side is a second device, the control center, that also has an MQTT client and receives the data. Additionally we will implement a notification, which alerts the control center if the sensor is disconnected.

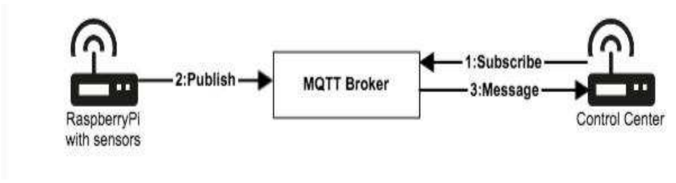


Fig. 2.2 Communication between the sensor client and the control center over MQTT

Client

When talking about a client it almost always means an MQTT client. This includes publisher or subscribers, both of them label an MQTT client that is only doing publishing or subscribing. (In general a MQTT client can be both a publisher & subscriber at the same time). A MQTT client is any device from a micro controller up to a full fledged server, that has a MQTT library running and is connecting to an MQTT broker over any kind of network. This could be a really small and resource constrained device, that is connected over a wireless network and has a library strapped to the minimum or a typical computer running a graphical MQTT client for testing purposes, basically any device that has a TCP/IP stack and speaks MQTT over it. The client implementation of the MQTT protocol is very straight-forward and really reduced to the essence. That's one aspect, why MQTT is ideally suitable for small devices. MQTT client libraries are available for a huge variety of programming languages, for example Android, Arduino, C, C++, C#, Go, iOS, Java, JavaScript, .NET. A complete list can be found on the MQTT wiki.

Broker

The counterpart to a MQTT client is the MQTT broker, which is the heart of any publish/subscribe protocol. Depending on the concrete implementation, a broker can handle up to thousands of concurrently connected MQTT clients. The broker is primarily responsible for receiving all messages, filtering them, decide who is interested in it and then sending the message to all subscribed clients. It also holds the session of all persisted clients including subscriptions and missed messages (More details). Another responsibility of the broker is the authentication and authorization of clients. And at most of the times a broker is also extensible, which allows to easily integrate custom authentication, authorization and integration into backend systems. Especially the integration is an important aspect, because often the broker is the component, which is directly exposed on the internet and handles a lot of clients and then passes messages along to downstream analyzing and processing systems. As we described in one of our early blog post subscribing to all message is not really an option. All in all the broker is the central hub, which every message needs to pass. Therefore it is important, that it is highly scalable, integratable into backend systems, easy to monitor and of course failure-resistant. For example HiveMQ solves this challenges by using state-of-the-art event driven network processing, an open plugin system and standard providers for monitoring.

MQTT Connection

The MQTT protocol is based on top of TCP/IP and both client and broker need to have a TCP/IP stack.

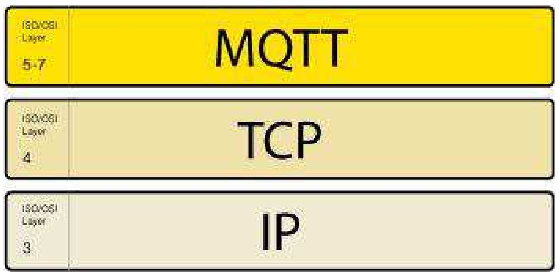
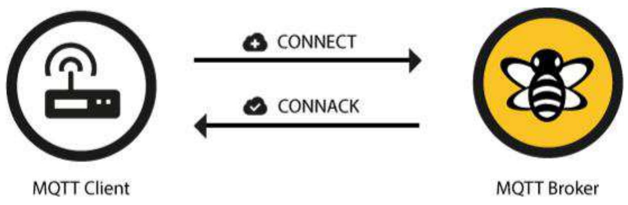


Fig. 2.3 MQTT protocol stack

The MQTT connection itself is always between one client and the broker, no client is connected to another client directly. The connection is initiated through a client sending a CONNECT message to the broker. The broker response with a CONNACK and a status code. Once the connection is established, the broker will keep it open as long as the client doesn't send a disconnect command or it looses the connection.

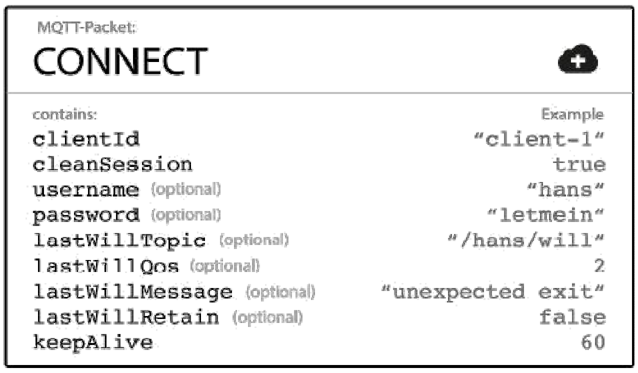


MQTT connection through a NAT

It is a common use case that MQTT clients are behind routers, which are using network address translation (NAT) in order to translate from a private network address (like 192.168.x.x, 10.0.x.x) to a public facing one. As already mentioned the MQTT client is doing the first step by sending a CONNECT message. So there is no problem at all with clients behind a NAT, because the broker has a public address and the connection will be kept open to allow sending and receiving message bidirectional after the initial CONNECT.

Client initiates connection with the CONNECT message

So let's look at the MQTT CONNECT command message. As already mentioned this is sent from the client to the broker to initiate a connection. If the CONNECT message is malformed (according to the MQTT spec) or it takes too long from opening a network socket to sending it, the broker will close the connection. This is a reasonable behavior to avoid that malicious clients can slow down the broker. A good-natured client will send a connect message with the following content among other things:


MQTT-Packet:
CONNECT

contains:	Example
clientId	"client-1"
cleanSession	true
username (optional)	"hans"
password (optional)	"letmein"
lastWillTopic (optional)	"/hans/will"
lastWillQos (optional)	2
lastWillMessage (optional)	"unexpected exit"
lastWillRetain (optional)	false
keepAlive	60

Additionally there are other information included in a CONNECT message, which are more a concern to the implementer of a MQTT library than to the user of a library. If you are interested in the details have a look at the MQTT3.1.1specification.

So let's go through all these options one by one:

ClientId

The client identifier (short ClientId) is an identifier of each MQTT client connecting to a MQTT broker. As the word identifier already suggests, it should be unique per broker. The broker uses it for identifying the client and the current state of the client. If you don't need a state to be hold by the broker, in MQTT 3.1.1 (current standard) it is also possible to send an empty ClientId, which results in a connection without any state. A condition is that clean session is true, otherwise the connection will be rejected.

Clean Session

The clean session flag indicates the broker, whether the client wants to establish a persistent session or not. A persistent session (CleanSession is false) means, that the broker will store all subscriptions for the client and also all missed messages, when subscribing with Quality of Service (QoS) 1 or 2. If clean session is set to true, the broker won't store anything for the client and will also purge all information from a previous persistent session.

Username/Password

MQTT allows to send a username and password for authenticating the client and also authorization. However, the password is sent in plaintext, if it isn’t encrypted or hashed by implementation or TLS is used underneath. We highly recommend to use username and password together with a secure transport of it. In brokers like HiveMQ it is also possible to authenticate clients with an SSL certificate, so no username and password is needed.

Will Message

The will message is part of the last will and testament feature of MQTT. It allows to notify other clients, when a client disconnects ungracefully. A connecting client will provide his will in form of an MQTT message and topic in the CONNECT message. If this clients gets disconnected ungracefully, the broker sends this message on behalf of the client. We will talk about this in detail in an individual post.

KeepAlive

The keep alive is a time interval, the clients commits to by sending regular PING Request messages to the broker. The broker response with PING Response and this mechanism will allow both sides to determine if the other one is still alive and reachable. We’ll talk about this in detail in a future post.

That are basically all information that are necessary to connect to a MQTT broker from a MQTT client. Often each individual library will have additional options, which can be configured. They are most likely regarding the specific implementation, for example how should queued message be stored.

Broker responds with the CONNACK message

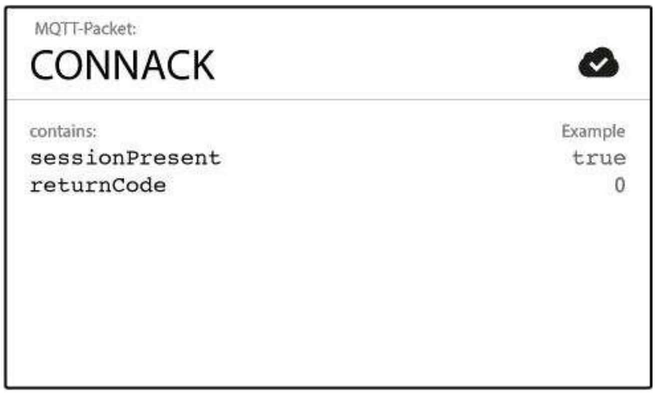
When a broker obtains a CONNECT message, it is obligated to respond with a CONNACK message. The CONNACK contains only two data entries: session present flag, connect return code.

Session Present flag

The session present flag indicate, whether the broker already has a persistent session of the client from previous interactions. If a client connects and has set CleanSession to true, this flag is always false, because there is no session available. If the client has set CleanSession to false, the flag is depending on, if there are session information available for the ClientId. If stored session information exist, then the flag is true and otherwise it is false. This flag was added newly in MQTT 3.1.1 and helps the client to determine, whether it has to subscribe to topics or if these are still stored in his session.

Connect acknowledge flag

The second flag in the CONNACK is the connect acknowledge flag. It signals the client, if the connection attempt was successful and otherwise it will fail.



MQTT-Packet:

CONNACK

contains:	Example
sessionPresent	true
returnCode	0

The table-1 shows the return codes at a glance.

Return Code	Return Code Response
0	Connection Accepted
1	Connection Refused, unacceptable protocol version
2	Connection Refused, Identifier rejected
3	Connection Refused, Server unavailable
4	Connection Refused, bad user name or password
5	Connection Refused, not authorized

Table 1: Return code and response description

3. Image Compression

The captured image through Secure Digital Camera (SDC) Must be compressed and transmitted securely across different IoT devices. The Image Compression can be done using **SPIHT algorithm** in the following manner.

The traditional image coding technology utilizes the redundant data in an image to compress the transmitting data. But these methods have been replaced by digital wavelet transform. These methods have peak speed, low memory requirements and absolute reversibility. Now in this work we are considering SPIHT algorithm. We convert the analog data with wavelet encoding scheme and examining the results in terms of bit error rate, PSNR and MSE.

The SPIHT algorithm is more efficient implementation than Embedded Zero Wavelet (EZW) algorithm. This compression conspiracy is based on wavelet coding technique. The given image is transformed by using a discrete wavelet transform. In the originating, the image is deteriorated into four sub-bands by cascading horizontal and vertical two-channel critically sampled filter-banks. After implementing wavelet transform to an image, the SPIHT algorithm partitions the decomposed wavelet into significant and insignificant partitions.

There are two passes in the algorithm- they are i-the sorting pass ii-the refinement pass. The SPIHT encoding technique utilizes lists of LIP (List of Insignificant Pixels). It contains individual coefficients that have magnitudes smaller than thethresholds.LIS (List of Insignificant Sets) contains set of wavelet coefficients that are described by tree structures and are found to have magnitudes smaller than the threshold. LSP (List of Significant Pixels). It is a list of pixels found to have magnitude value greater than the threshold (significant) fixed for the problem. The sorting pass is performed on the above three lists.

In the refinement pass, the nth MSB of the coefficients in the LSP is the final output. The value of n is decremented. These passes will keep on pursuing until either the desired rate is reached or n =0. The latter case will give an almost perfect renovation since all the coefficients have been processed completely. The bit rate can be controlled absolutely in the SPIHT algorithm as the output generated is in single bits and the algorithm can be completed at any time.

4. Proposed Solution

In this paper we provide a model fig. 4.1 for transmitting compressed big data images across different IoT devices securely using a lightweight publish/subscribe protocol namely MQTT(Message Queuing Telemetry Transport) Protocol.

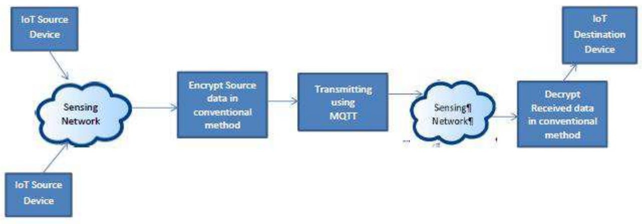


Fig : 4.1 Proposed Model for Transmitting Compressed

Image across different IoT devices using MQTT Protocol. The model works as follows

Step 1 : The image is captured by an IoT device.
Step 2 : The Captured image will be compressed.

Step 3 : The Compressed image is encrypted using any of the conventional algorithms.

Step 4 : The encrypted image will be transferred to the receiving IoT device using the MQTT protocol.

Step 5 : The receiving IoT device will decrypt the encrypted compressed message of sending device.

Step 6 : The decrypted message is now decompressed and read by the receiving IoT device.

5. Conclusion

The IoT is the future of the world, where everything in human mankind is revolving around advanced gadgets and for a secure transmission of data using a lightweight protocol MQTT is providing the solution.

The future work of this paper is to simulate the proposed model and do a comparative study across different conventional encryption algorithms and public-key/private-key encryption algorithms for a set of different key sizes and image sizes.

6.References

1. J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, “Internet of things (iot): A vision, architectural elements, and future directions,” Future Generation

   Computer Systems, vol. 29, no. 7, pp. 1645–1660, 2013.

2. R. Want and S. Dustdar, “Activating the internet of things [guest editors’ introduction],” Computer, vol. 48, no. 9, pp. 16–20, 2015. Fig. 6: Correlation comparison

3. J. Romero-Mariona, R. Hallman, M. Kline, J. San

   Miguel, M. Major, and L. Kerr, “Security in the industrial internet of things,” 2016.

4. H. Suo, J. Wan, C. Zou, and J. Liu, “Security in the internet of things: a review,” in Computer Science and

   Electronics Engineering (ICCSEE), 2012 International Conference on, vol. 3. IEEE, 2012, pp. 648–651.

5. G. Ho, D. Leung, P. Mishra, A. Hosseini, D. Song, and

   D. Wagner, “Smart locks: Lessons for securing commodity internet of things devices,” in Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security. ACM, 2016, pp. 461–472.

6. D. Airehrour, J. Gutierrez, and S. K. Ray, “Secure routing for internet of things: A survey,” Journal of Network and Computer Applications, vol. 66, pp. 198– 213, 2016.

7. D. Miorandi, S. Sicari, F. De Pellegrini, and I. Chlamtac, “Internet of things: Vision, applications and research challenges,” Ad Hoc Networks, vol. 10, no. 7, pp. 1497–1516, 2012.