Shanghai Jiao Tong University
University of Michigan- Shanghai Jiao Tong University Joint Institute

Design and Implementation of MAC, Routing, and Application Protocols for Internet of Things

by

Lingyang Ma

A thesis submitted in partial satisfaction of the
requirements for the degree of Master of Science in
Electrical and Computer Engineering at Shanghai Jiao Tong University

Committee in charge:
Professor Xudong Wang, Chair
Assistant Professor Weikang Qian
Assistant Professor Chong Han

Shanghai
March, 2017

上海交通大学

交大密西根学院

物联网 MAC 与路由协议及应用的研究与开发

马龄阳

上海交通大学密西根学院硕士学位论文

信息与通信工程专业

委员会成员：

王旭东 （主席）

钱炜慷

韩充

上海

2017 年 3 月

Abstract

Internet of Things (IoT) has become an important research area in recent years. From the perspective of networking, IoT is a type of heterogeneous wireless networks. So far it still has many problems to be solved in different protocol layers. This thesis is mainly focused on the media access control(MAC), network, and application layers.

In the MAC and network layers, a new IoT system model is proposed based on hierarchical mesh networks. Based on this model, a new MAC protocol and an efficient routing algorithm are developed to support an IoT system that are equipped with many devices forming a large scale heterogeneous multi-hop network. Simulations are conducted to evaluate the performance of the new schemes, and the results show that the new MAC protocol can achieve much improved performance with reduced collision possibility and delay. Moreover, the new routing algorithm significantly reduces the overhead of route table setup and thus improves the efficiency of the entire routing process. The new schemes are also prototyped and validated on testbed.

In the application layer, a lightweight messaging protocol called Message Queuing Telemetry Transport(MQTT) is adopted for message delivery among IoT devices, because it is much more efficient than other application layer protocols such as HyperText Transfer Protocol(HTTP). Based on MQTT, an IoT application system is developed for typical IoT application scenarios. It supports efficient data exchange among IoT devices. Moreover, it provides users an interface to manage the whole network through a management system. More specifically, users cannot only check data sent from different devices but can also change the configuration parameters of each device. To evaluate the new IoT application, prototype software is implemented on both PCs and mobile phones. Experiments successfully demonstrate the distinct features of the new IoT application.

摘要

近年来，物联网是业界的一个研究热点。从通信网络的角度来看，物联网是一种无线异构网络。在物联网研究领域，许多问题依然有待解决。根据网络分层模型，这些问题同时存在于各层中。在本论文中，我们主要关注介质访问控制层，网络层和应用层。

在介质访问控制层，针对异构 Mesh 网络，本文提出了一种新的系统模型。基于该模型，我们设计了新的介质访问控制协议和路由协议，用于支持大规模的异构多跳物联网。论文中给出了仿真用于评估协议性能。仿真结果表明，在介质访问控制层，我们的协议能在更小的碰撞率和时延情况下达到相似或者更好的表现；而在网络层，我们的路由协议能有效减少路由表建立的开销，从而提高效率。最后，基于我们实验室的嵌入式系统平台，我们实现了系统的一部分功能。

在应用层，首先我们分析了现有的超文本传输协议协议的缺点，然后介绍了一种更适用于物联网场景的轻量级消息推送协议：消息队列遥测传输（MQTT）。基于 MQTT，我们针对典型应用场景开发了一套物联网应用系统。使用该系统，物联网设备不仅能够彼此交换数据，同时用户还能够对整个系统进行管理，例如，用户可以通过管理中心查看每一个设备的数据发送情况，并修改设备的配置信息等。我们已经基于 PC 端和移动端搭建了一套原型系统，完整地实现了我们设计的系统功能。

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Existing Problems and Related Work

As electronic and communication technologies develop, the dream to connect everything together is becoming more and more close to reality, which makes Internet of Things(IoT) a hot research area in recent years(Atzori et al. (2010)). There have been some applications whose concepts are similar to IoT, such as Wireless Sensor Networks(Akyildiz et al. (2002)). However, these applications can hardly be regarded as IoT, because they only work under certain scenarios, while one of IoT's most important features is the ability to fit various conditions.

A lot of problems stay unsolved in IoT. For example, in practical systems, there are different kind of networks, how to make these networks coexist together is a big problem. Thus, from a network point of view, the key research topic in IoT is heterogenous wireless network.

There have been some researches about heterogenous wireless networks. Stevensnavarro et al. (2007) and Lee et al. (2009) proposed some vertical handoff algorithms when a mobile terminal switches between different networks. In Ferrus et al. (2010) and Niyato and Hossain (2007), new frameworks are studied to connect different networks such as WiMAX(Eklund et al. (2002)) and 3G/4G so that ubiquitous access and seamless mobility can be achieved. Han et al. (2009) addressed the problem of deploying relay nodes to provide fault tolerance with higher network connectivity in heterogeneous wireless networks. There are also some papers study on topology control and resource allocation problems. Li and Hou (2006) presented two localized topology control algorithms. Wang (2005) proposed a minimum-power allocation algorithm for wireless wide area multimedia networks, and based on which, a multimedia wideband CDMA generalized processor sharing (GPS) scheduling scheme is designed. Lv et al. (2012) developed a virtual access network embedding framework which supports flexible

resource allocation. What's more, communication security in heterogenous networks is also very important, Lu et al. (2008) and Wang et al. (2008) discussed this problem and proposed some methods to guarantee security.

However, most of these works can't be applied directly to IoT. On one hand, these works focus on solving problems between mobile terminals and stations rather than problems between terminals and terminals. On the other hand, these works mainly studied networks such as WiMAX and 3G/4G, which are quite different from some simpler networks such as wifi(Crow et al. (1997)), zigbee(Gislason (2008)) and bluetooth(Haartsen (2000)). Considering these problems, we mainly study on heterogenous wireless mesh networks(Akyildiz et al. (2005)). In fact, choosing mesh network as the system model is quite reasonable, since in practical systems, wireless mesh network is one of the most widely used network form, which fits a lot of scenarios, such as wireless sensor network, smart home(Chan et al. (2009)) and smart city.

What's more, IoT is not only a kind of network, it's also a kind of web service, thus application systems should be built for IoT.

In our life, we are using different kind of web services. For example, when we receive or send a email, we are using mail service. When we download some files from a file server, we are using FTP(Reynolds (1985)) service. Among all these services, HTTP(Totty et al. (2002)) service is the most widely used service. However, due to the complexity of HTTP protocol, it's not suitable for IoT system. The main problem to be solved in IoT system is how to send a message to a certain device efficiently. Using HTTP protocol, the procedure is like this: when a client wants to receive messages from a server, it has to initiate a connection to the server to ask for messages. If the server has message for the client, it will send the message immediately, otherwise it just disconnects the connection. Then the client needs to initiate a connection again, and the same process will be repeated periodically. Fig 1.1 shows the procedure.

We know that HTTP is based on tcp. To build an HTTP connection, a tcp connection needs to be built first, thus every time the three-way handshake needs to be applied, which will cause high overhead. Also, due to tcp's congestion control algorithm, the transmission rate at the beginning of each tcp connection is very low. Because of these two reasons, the efficiency of this procedure is very low, which will consume a lot of power. Another shortcoming of HTTP is that it's very heavy, which means to use this protocol, a lot of data need to be transmitted, which also increases the power consumption. Thus in IoT system, we need to choose other protocols which is light-weighted and efficient.

There are some messaging protocols which are designed for message pushing, for example, Websocket(February (2011)), XMPP(Liuhto and Mantysaari (2004)) and MQTT. The transmitting procedure of these protocols are quite different from HTTP. Fig 1.2 shows the basic procedure. When client A wants to receive data from

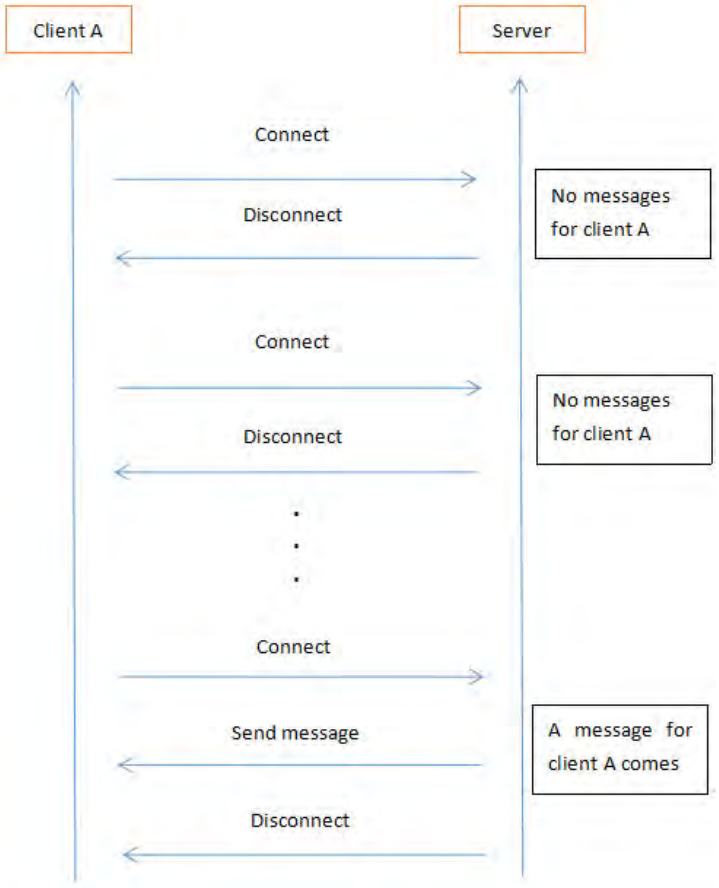


Figure 1.1: Process of data transmission using HTTP

the server, it will initiate a connection to the server. But instead of asking for messages, client A and server will just keep the connection active. When a message for client A comes to the server, the server will send this message to client A. Through this procedure, client A doesn't have to build connections again and again to ask for messages, thus the power consumption will decrease, also due to the simplicity of these protocols, less data need to be transmitted.

Among these message pushing protocols, Websocket is a part of HTML5(Hickson (2014)) standard, which is built upon HTTP. To initiate a Websocket link, first client and server need to use HTTP, after the link is established, HTTP is then abandoned. For some simple devices in IoT, HTTP is too heavy and complex after all, this shortcoming limits Websocket's application in IoT. As for XMPP, it also has a constraint: all packages in XMPP use XML as their data format. XML is easy to read, and is very capable of representing structural data, however, the data format itself has a lot of redundancy, making its efficiency very low. Compared with Websocket and XMPP, MQTT doesn't have their shortcomings, it's a totally new application layer protocol based on tcp, and its data format is utf-8 strings, which is much simpler than XML. What's more, it's based on

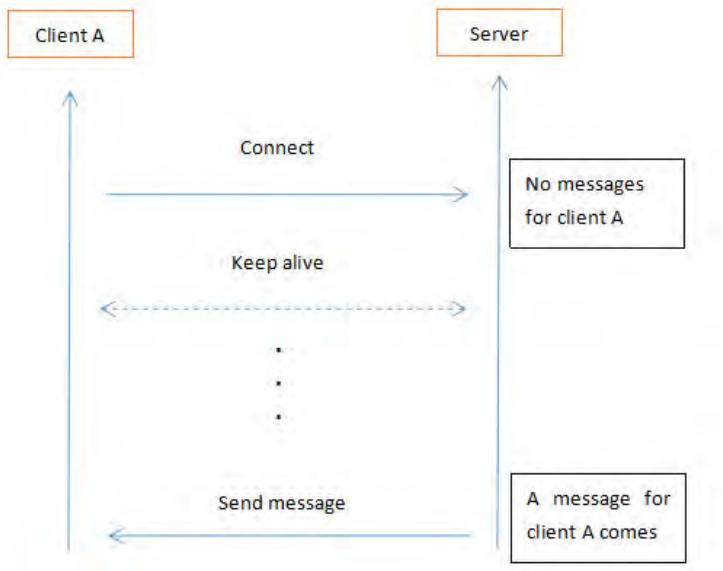


Figure 1.2: Process of data transmission using messaging protocols

publish/subscribe mechanism, which is very suitable for IoT application scenarios. Thus, in our IoT application development, we will use MQTT.

1.2 Organization of the Thesis

The rest of the proposal is organized as follows. In chapter 2, from typical application scenarios of IoT, we analyze existing problems in MAC layer and network layer. To solve these problems, we propose a new system model, and based on our model, new MAC protocols and routing algorithms are described respectively. Simulation results are given in to evaluate the performance of our proposed protocols and algorithms, a simple test is also done using our lab's embedded systems. In chapter 3, MQTT protocol is first introduced.Based on MQTT, we describe the functions of our IoT application system, and then the implementations are presented. Finally in chapter 4, the future research plan is scheduled.

Chapter 2

MAC Protocol and Routing Algorithm Design

2.1 Existing Problems

IoT's target is to connect everything, which means each device should have ability to communicate with other members in IoT. A most typical application scenario of IoT is smart city, which can be described by Fig 2.1. This system has two characteristics. First, the scale of system is very large. In Fig 2.1, the circle can be a campus, a community, or even a whole city, and all devices form a wireless mesh network. Second, there are different kind of network devices allocated in the same area. For example, some of them are WiFi devices while others are Zigbee or Bluetooth devices, and they are represented by different colors in the figure.

There are several problems existing in this system.

In MAC layer, we know that different kind of networks use different standards, still take WiFi, Zigbee and Bluetooth as examples, they use 802.11, 802.15.4 and 802.15.1 respectively. These differences will make the whole system hard to manage. Also, in this system, there can be a lot of devices in the same area, making the communication traffics crowded. Although we can try to solve this problem by allocating different channels to different devices, but when the device density is high enough, the traffic in each channel will still be very crowded. From Bianchi (2000) we know that in MAC layer, as device number in one channel increases, system's performance is bad. First, the packet collision probability will be very high, which is quite unacceptable considering that a lot of devices in IoT are energy-limited. Also, due to high collision probability, the network's throughput will drop quickly.

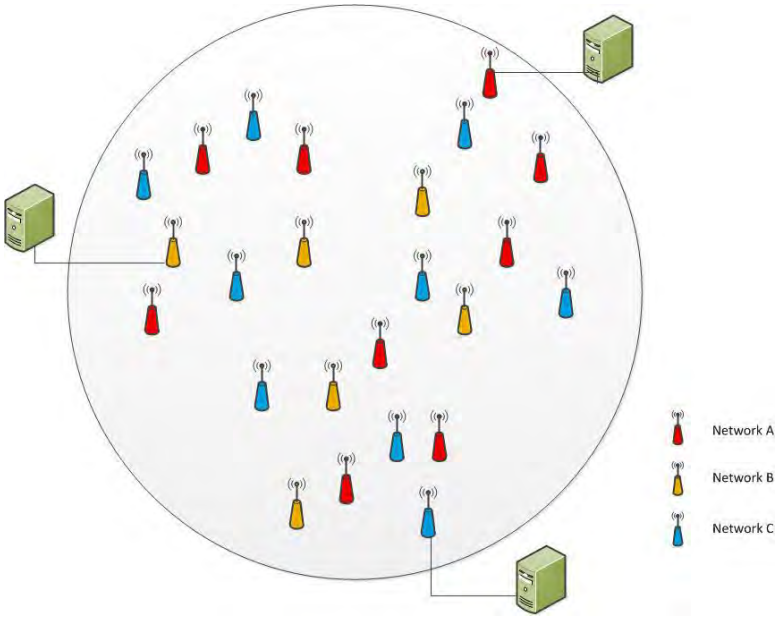


Figure 2.1: Typical scenario

In network layer, there are also some problems. First, in wireless mesh networks, each node needs to establish a route table so that it can know how to transmit its data packets to other nodes. To establish route tables, nodes broadcast topology control(TC) messages containing their own topology information to the whole network, which can be a very costly process. Flooding algorithm is the simplest broadcasting algorithm but the performance is bad. There have been a lot of other algorithms to improve the efficiency of broadcasting process, for example, in optimal link state routing(Clausen (2003)) algorithm, every node only selects some of its one hop neighbors rather than all of them to forward its TC packets, these selected nodes are called Multipoint Relay(MRP) nodes. But still, when the scale of network gets large, the overhead of establishing route tables is quite high. The second problem caused by large network scale is that for each data packet, the average transmitting hop number will be big, since the distance between source node and destination node can be very long and data packet has to be relayed by other nodes many times. Third, there usually exist some data packets need to be exchanged between different networks. Since devices in different networks use different technologies, they can't communicate directly. Data packets have to be uploaded to the internet by source nodes and then downloaded by destination nodes, this process is quite inefficient.

2.2 A New System Model

To solve problems mentioned in Section 2.1, we propose a new system model which is shown in Fig 2.2. In this system, we add a new device called mesh router, who has several characteristics. First, a mesh router has

different kind of network interfaces so that it can communicate with different devices. Second, a mesh router has larger transmission power so that it can support long distance communications. Third, it has stronger computing ability to run complicated algorithms.

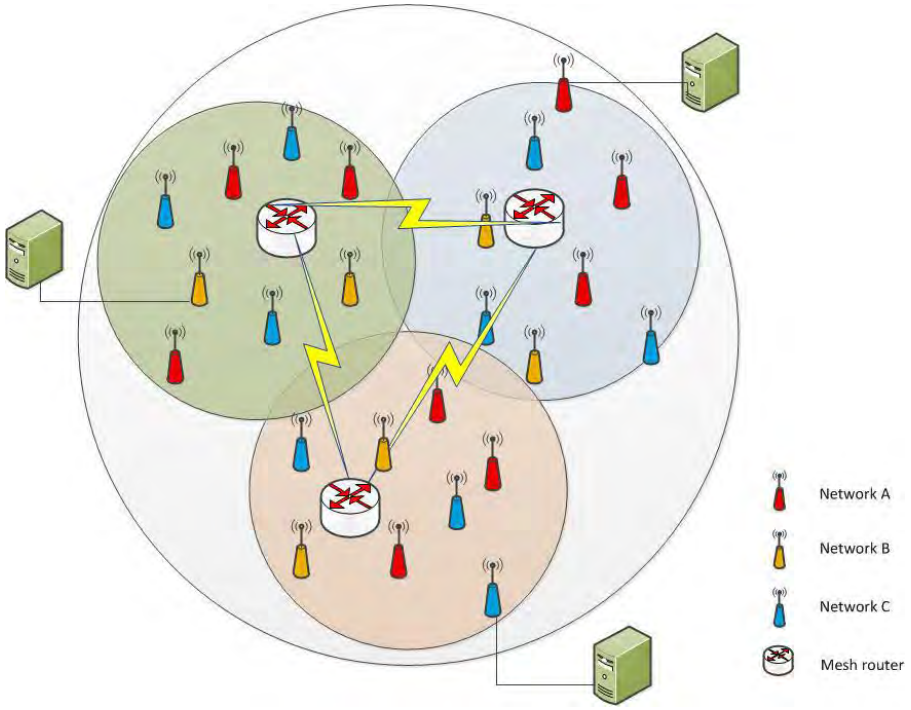


Figure 2.2: New system model

And with these new added mesh routers, we divide the whole system into two levels. Level one is formed by groups, which is actually subnetworks whose centers are mesh routers. Level two is a backbone network formed by mesh routers themselves. When a mesh router communicates with ordinary nodes in its own group, it just acts like an ordinary node too. As for the communication between mesh routers, a certain wireless channel is allocated to them so that they won't interfere ordinary nodes. Thus we can see, this system now has a hierarchical structure.

2.3 MAC Protocol Design

As we mentioned in Section 2.1, too many devices in the same channel will make the system performance very bad. To solve this problem, a basic idea is to allocate different active periods to different networks, as shown in Fig 2.3, devices in different networks only transmit data in their own active periods and in other periods they just keep silent. This can be regarded as a kind of Time Division Multiplexing(TDM) method. In fact, there

have already been some similar ideas proposed(Wang et al. (2005)).

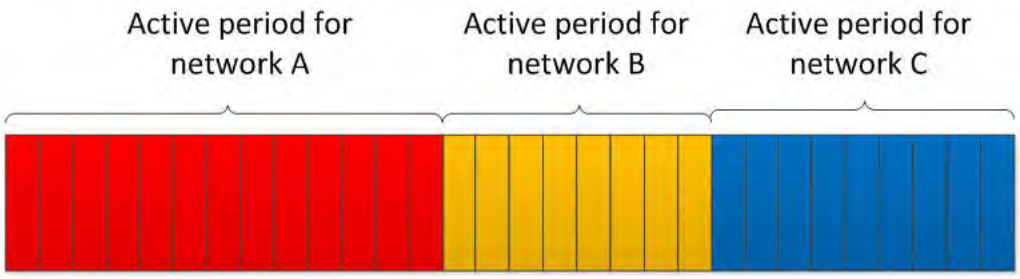


Figure 2.3: Basic idea to solve traffic problem in MAC layer

Since there are different kind of networks in this system, to implement this TDM method, we need to design different MAC protocols in order to conform to different standards. Here we just consider three kind of devices that are most widely used: zigbee, wifi and bluetooth.

For zigbee devices, according to 802.15.4 standards, there exist two kind of modes in MAC layer: Beacon mode and non beacon mode. Under beacon mode, there exists a specific zigbee device called coordinator who sends beacon packets periodically to control the whole zigbee network. Every beacon interval is divided into two parts: active period and inactive period, as shown in Fig 2.4. In active period, zigbee devices use slotted CSMA/CA algorithm to contend for transmission opportunities. In inactive period, they just sleep to save energy. The lengths of a beacon interval's active period and inactive period are decided by two parameters in beacon packets: BO and SO. The calculation functions are also shown in Fig 2.4, here *aBaseSuperframeDuratoin* is a constant parameter. In our system, mesh routers can play the role of zigbee coordinator since they have zigbee interfaces. Then we can find that it's straightforward to use beacon mode to control other zigbee devices in each group.

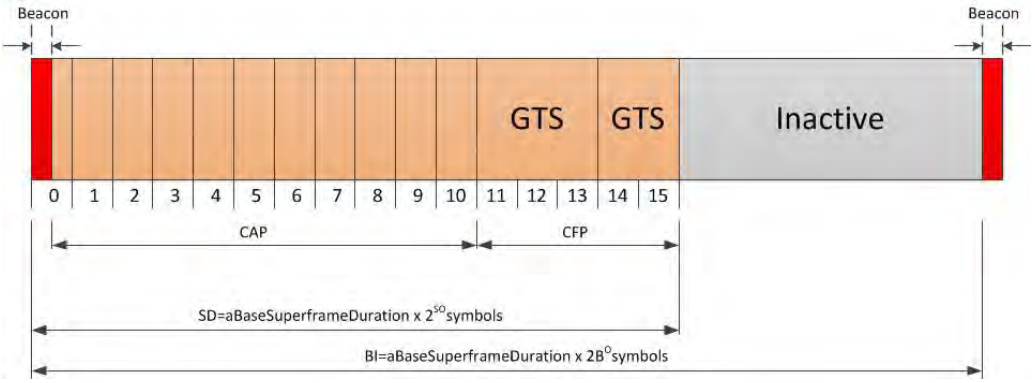


Figure 2.4: Process of zigbee's beacon mode

For wifi devices, 802.11 standard is used. There is no such straightforward method like zigbee's beacon mode

in 802.11, so another way is needed to allocate active and inactive periods to wifi devices. We notice that in 802.11 there exists an optional mechanism called CTS-to-self, devices who choose this mechanism will work like this: before a device starting to transmit data packet, it first transmits a CTS frame, with a receive address(RA) field set as its own MAC address. There is also a field containing a duration time, when other devices receive this CTS frame, they will set their NAVs according to this field. When this device begins to transmit data, other devices will just keep silent due to their NAVs, by this way, collisions can be avoided. We can easily find that by using CTS-to-self, we can allocate inactive and active periods to wifi devices. In our system, a mesh router can be the device who sends CTS-to-self periodically to control other wifi devices in its own group. The whole process can be shown in Fig 2.5. In the beginning of every beacon interval, after sending beacon packet, a mesh router will send a CTS-to-self after SIFS. When other devices receive this packet, they will keep silent according to NAVs. Once NAV runs out, they will begin to contend for transmission opportunities. Notice that there may be two modes in wifi devices, some of them are under Ap-client mode, and others are under ad-hoc mode. No matter in which mode, this CTS-to-self mechanism will work.

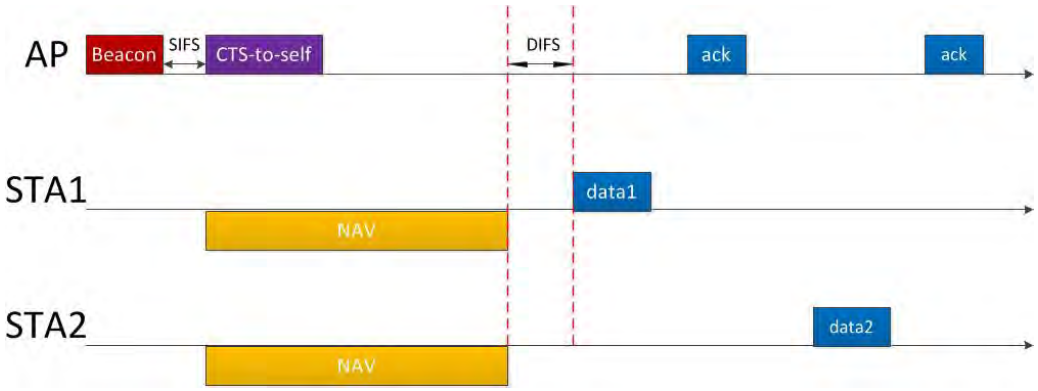


Figure 2.5: Process of wifi's CTS-to-self

As for bluetooth, some works have been done to solve the coexistence problem between bluetooth and other devices. One of most widely used method is called adaptive frequency hopping(Golmie et al. (2003)), which works like this: before bluetooth devices transmitting, they will check channel conditions, then by using channels which don't collide with wifi or zigbee, the coexistence problem is solved. Instead of using MAC layer method. But in this paper we mainly consider time division method. In fact, for bluetooth, it's very convenient to allocate active and inactive period, because bluetooth devices use master/slave mode, slaves can only transmit data when they are asked by a master. In our system, mesh router can be the master.

Now from the above, we can figure out how the whole system works. Fig 2.6 shows the process. First, a mesh router uses wifi interface to sends a beacon packet, and after SIFS, it sends a CTS-to-self packets to

allocate NAVs to other wifi devices in its group. Then it use zigbee interface to send a beacon packet, allocating active and inactive periods to zigbee devices in its group. Here the lengths of wifi devices' NAVs and zigbee devices active periods are the same, so that when zigbee devices are transmitting, wifi devices just keep silent due to NAVs, and when wifi devices begin to transmit, zigbee devices will enter inactive periods and begin to sleep. And then this process repeat again. As for bluetooth devices, we also use the same mechanism. In this way, we finally achieve the target to let different network transmit in different periods so that devices' collision probabilities can be decreased.

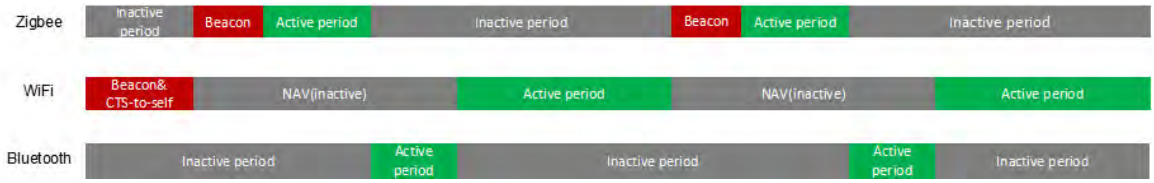


Figure 2.6: MAC process of the whole system

2.4 Routing Algorithm Design

As we analyzed in Section 2.1, the performance of traditional routing algorithms in large-scale networks is not good: the route table establishment process will cause a lot of overhead and the average hop number will be big. Thus new mechanisms are needed for IoT.Fortunately, based on our new system model, we can make some changes to traditional routing algorithms to improve their efficiencies.

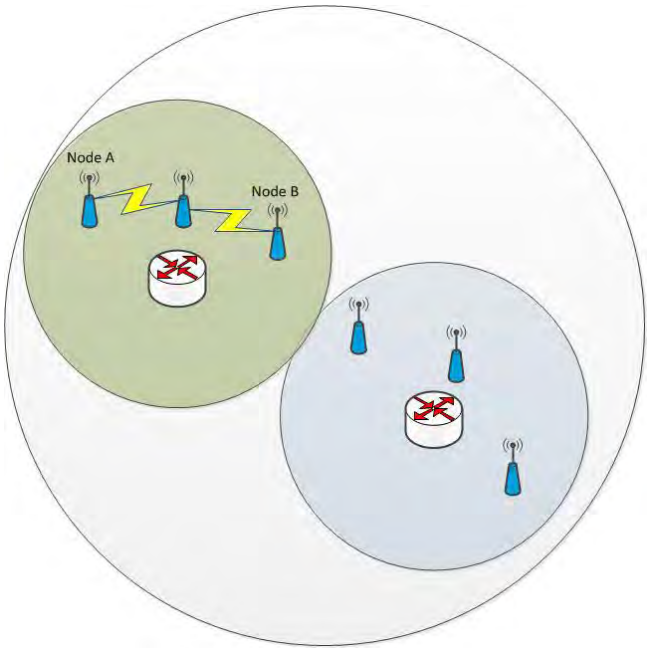
The basic idea is that each node only broadcasts its topology control messages to its own group rather than broadcasting them to the whole network. As for the topology information outside its group, mesh routers will get it. The detailed process is listed as follows:

1. Each node maintains a group number got from the mesh router of its group.

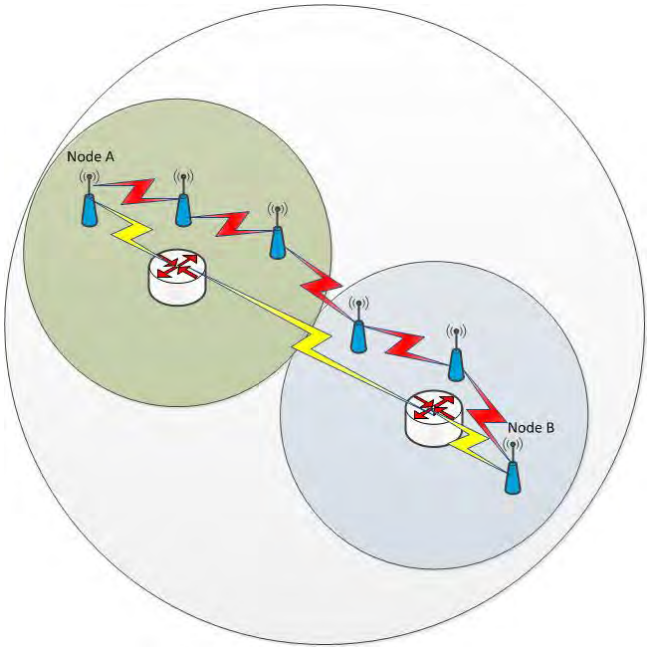
2. HELLO message is used for neighbor discovery, which is only transmitted to one's neighbors and should be never forwarded. A node's address and its one-hop neighbors' addresses are contained in HELLO messages. By receiving HELLO messages, a node can know the topology of its one hop and two hop neighbors. Then it chooses its Multipoint Relay(MPR) neighbors just using the same algorithm used in OLSR.

3. A node inserts its topology information into TC and broadcasts it. This node's group number should also be contained in TC.

4. When a node receives a TC packet, it will compare the group number contained in the packet. If the group number is equal to its own, it will forward this TC, otherwise it won't forward.



(a) Collision possibility when load is 10 pks/s



(b) Delay when load is 10 pks/s

Figure 2.7: Routing process of two cases

5. Mesh routers should exchange their own route tables using a specific channel.

Following these principles, finally the situation will be like this: normal nodes only have route tables of their own groups, while mesh routers have route tables of the whole network. Under this situation, when a node, say, node A, wants to transmit data to another node called node B, there will be two cases:

1. If node A and B are in the same group, node B's address will be in node A's route table, node A just transmits data packet according to route table.

2. If node A and B are in different groups, there will not be node B's address in node A's route table. Node A just transmits data packet to the mesh router of its own group. Since the mesh router has a route table of the whole network, it will find which group node B is and forward data packet to the mesh router of node B's group, then finally, the data packet will be forwarded to node B.

Fig 2.7 shows the routing processes of these two cases.

Now we can compare our routing algorithm with traditional algorithms. In the route table establishment process, since the broadcasting range of TC messages are bounded within one group, the overhead and complexity to maintain route tables are reduced dramatically. In the data transmission process, when node A and B are in the same group, the transmission process is the same to traditional algorithms. When node A and B are far away from each other, the average hop number from A to B will be smaller than traditional algorithms since the communication distances between mesh routers are longer than normal nodes. Still take Fig 2.7 as an example, if node A want to transmit data to node B, in traditional algorithms, the data packet will go through 3 hops while in our system only 5 hops are needed.

Since our mesh routers have multiple interfaces to support different networks, the data transmission efficiency between different network devices can also be improved. Suppose a wifi device A wants to transmit data to a zigbee device B, it can send data packets to a mesh router, letting this mesh router forward data packets to device B. In this way, data packets don't need to go through the internet anymore and the overhead is reduced dramatically.

2.5 Simulation and Implementation

In this section, we design some simulations to evaluate the performance of our new system. Simulation results of both new system and traditional system are given to see whether the performance is improved. A demo is also done based on our lab's embedded system to implement the data exchanging process between different networks through mesh routers.

2.5.1 MAC Layer Simulations

The scenario we design for our MAC layer simulations can be saw in Fig 2.8: The range of the whole network is one hop, which means every node can interfere others. In this network exist two kind of nodes: wifi nodes and

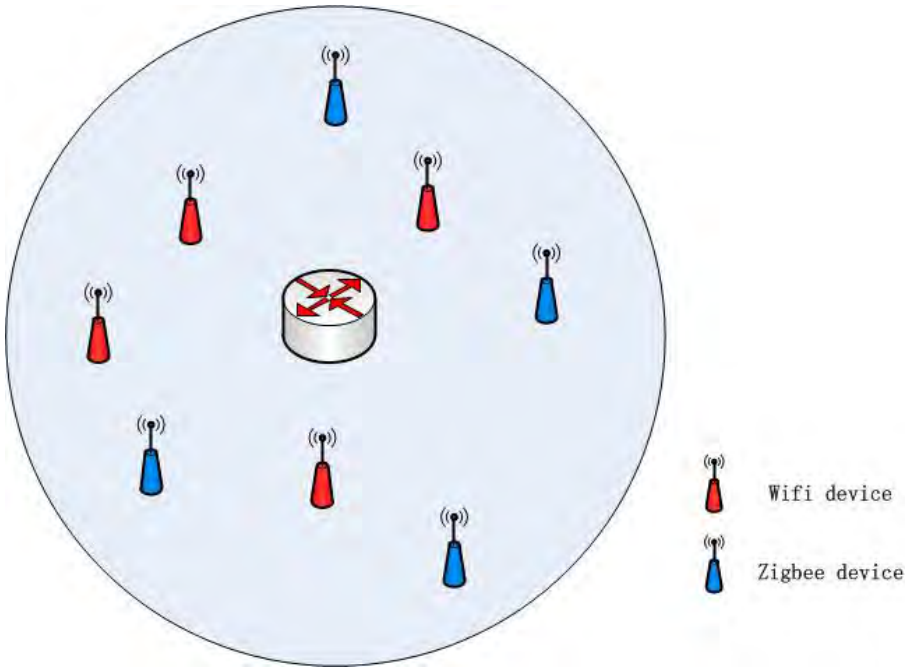


Figure 2.8: scenario of MAC layer simulations

SIFS	16 μsec
DIFS	34 μsec
slot unit	9 μsec
CWmin	31
CWmax	1023
packet length	2500 Bytes

Table 2.1: Parameters used for wifi devices

zigbee nodes, with each kind's number equal to another kind. All nodes have the same load, here load means the number of packets generated by each node per second. We keep the load fixed while we increase the number of nodes(still keep each kind's number equal to another kind) in this network to see how will performance change. Here we use collision probability, end-to-end packet delay and throughput to evaluate the performance. First we do simulations using traditional MAC protocols, which means wifi and zigbee nodes contend freely for transmission opportunities. Then we do simulations using new MAC protocols presented in section 2.3.

The parameters we use in our simulations are shown in table 2.1 and 2.2. We design three groups of

aBaseSuperframeDuration	960 symbols
SO	1
BO	5
BEmin	3
BEmax	5

Table 2.2: Parameters used for zigbee devices

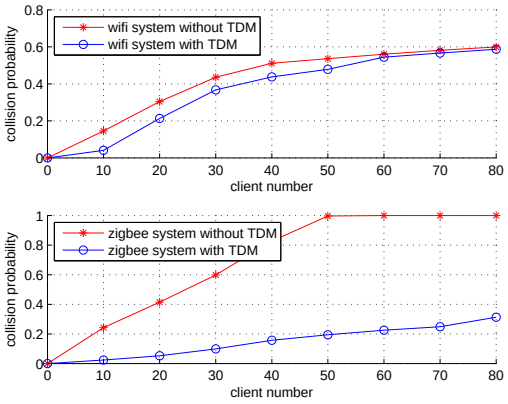
simulations, in group one the load is 10 packets/s, in group two the load is 50 packets/s and in group three the load is 100 packets/s. Simulation results of them are shown in Fig 2.9, Fig 2.10 and Fig 2.11.

First we analyze wifi devices. From figures we can see that keeping load fixed, when wifi client number in a network increases, the average data collision possibility and data delay will increase too. This result conforms to what we analyze, since more clients means more transmission contentions at the same time, the possibility for them to collide is higher. And more collisions means more retransmissions, which will make bigger average packet delay.

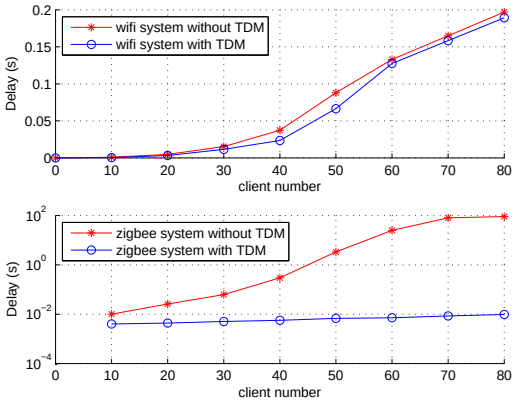
So no matter what kind of MAC protocols we use, this tendency stays similar, the only difference is that when using our new MAC protocol, the average collision possibility and packet delay are always smaller than what are in traditional system. This is reasonable because in new system, when wifi devices transmit, there are no zigbee device to interfere them. As for network throughput, in figures we can see as client number increases, first the throughput will increase too, then when the client number reaches a threshold, the throughput will begin to drop. Comparing two wifi networks using different MAC protocols, we can see the throughput in new system is almost the same as what is in traditional system, but remember that we achieve this in new system with smaller collision possibility and packet delay.

Second let's see zigbee devices. Keeping load fixed, the average data collision possibility and data delay will increase as zigbee client number increases. But here we can see in traditional system, collision possibility and data delay are far bigger than what are in new system, especially when the client number is big, the collision possibility can even be close to 100%. But in new system, even when the client number is big, the collision possibility and data delay can still be within an acceptable range. Why in traditional system the performance will be so bad when client number is big? This is mainly caused by the interference of wifi devices.

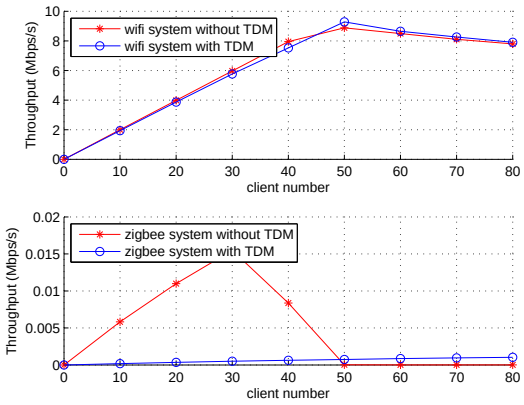
We know that zigbee devices use 802.15.4 standard, in this standard, a mechanism called slotted csma/ca is proposed. But different from the csma/ca in 802.11, in slotted csma/ca, zigbee devices don't always do clear channel assessment(CCA) because CCA will cost a lot of energy while 802.15.4 is mainly designed for energy limited devices.Instead, when a zigbee device contends for transmission opportunities, first it will do random backoff process no matter the channel is idle or not, when backoff process ends, it will then do CCA at the beginning of a time slot, if the channel is "idle", in next time slot, it will begin to transmit data. But here is a problem, because the CCA is only done at the beginning of a time slot then it will stop CCA and wait for next time slot, when it begins to transmit, the channel can already be busy due to other zigbee devices or wifi devices, but the data will still be transmitted, and a collision then happens. The same problem will never happen to wifi devices since they always do CCA before they transmit data. From this point of view, when wifi



(a) Collision possibility when load is 10 pks/s



(b) Delay when load is 10 pks/s

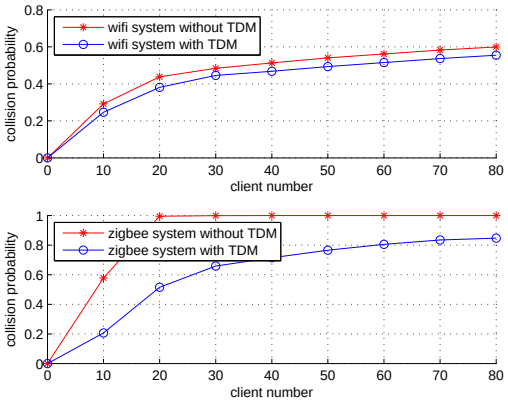


(c) Throughput when load is 10 pks/s

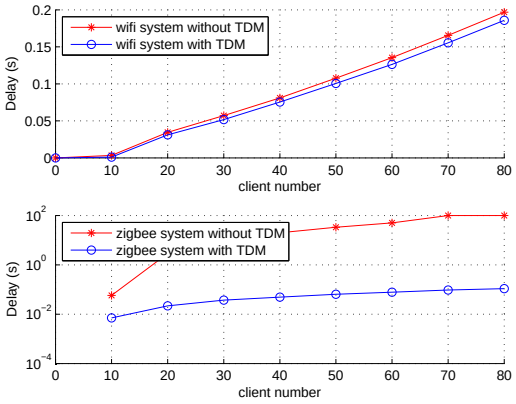
Figure 2.9: Simulation results when load is 10 pks/s

devices and zigbee devices exist together, wifi devices will usually occupy more resources. This can be reflected by throughput results in traditional system, we see when client number reaches a threshold, the throughput of zigbee network will drop very quickly, especially when client number is big enough, zigbee network's throughput will be very low. Fortunately we see in our new system, as client number increases, the throughput of zigbee network won't drop too fast, it can still be kept in an acceptable range even when client number is big. This is because the wifi devices are silent when zigbee devices transmit due to our new MAC protocol. So although when data load and client number are small at same time, the throughput in traditional system is bigger, in most situations, the new system has bigger throughput.

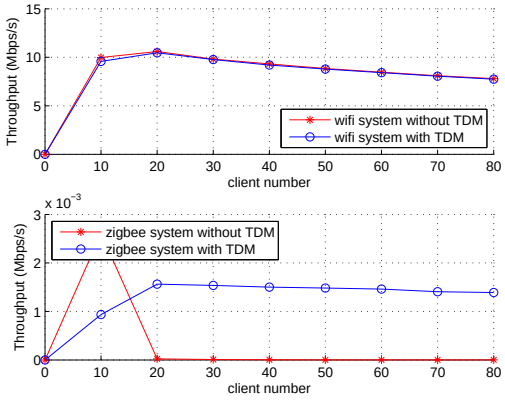
From above analysis, we can see in a system with both wifi and zigbee devices, using our new MAC protocol, we can achieve bigger or similar throughput with smaller collision possibility and data delay in most situations. Considering there exist a lot of energy limited devices in IoT, our new MAC protocol can bring a big performance



(a) Collision possibility when load is 50 pks/s



(b) Delay when load is 50 pks/s



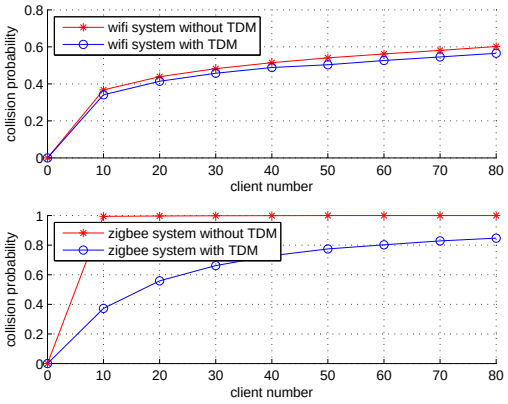
(c) Throughput when load is 50 pks/s

Figure 2.10: Simulation results when load is 50 pks/s

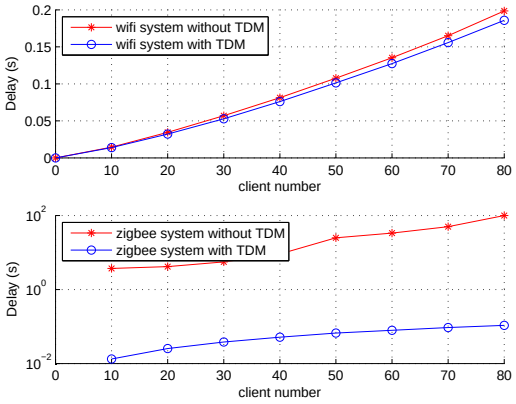
improvement for the whole system.

2.5.2 Network Layer Simulations

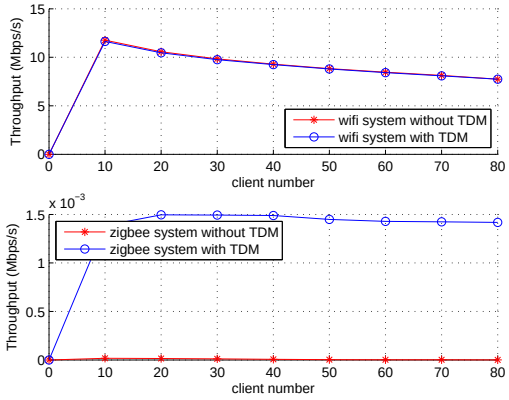
Fig 2.12 shows the scenario we design for our first network layer simulation. The system is using hierarchical structure we proposed in section 2.2: there are four groups in this system, and the radius of each group is two hops. The density of nodes in this system is 60 clients/group, and they are allocated randomly in system. Each node transmits two kind of packets: TC packets and data packets. The TC packet of each node is generated randomly every five seconds one time on average. In traditional system, each TC packet should be broadcasted to the whole network, while in our new system, it only needs to be broadcasted within its own group. We will do simulations in both traditional system and new system to see how system performance will change as data packet load increases. Again, here we use data collision possibility, data delay and data throughput to evaluate



(a) Collision possibility when load is 100 pks/s



(b) Delay when load is 100 pks/s



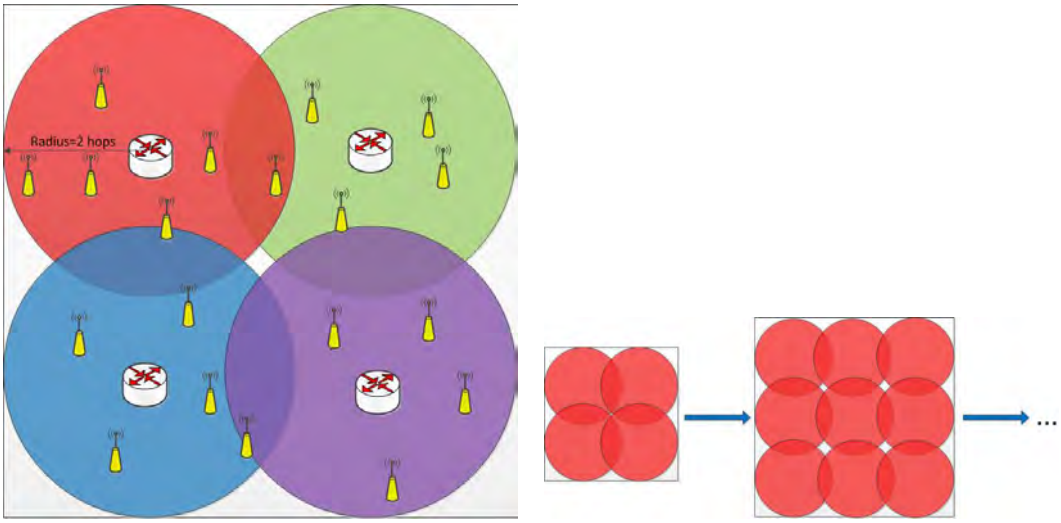
(c) Throughput when load is 100 pks/s

Figure 2.11: Simulation results when load is 100 pks/s

performance.

Fig 2.13 gives the simulation results, from which we can see compared to traditional system, the data collision possibility and delay of our new system are smaller, and the throughput is bigger. This is easy to explain, since in traditional system, more resources are needed to broadcast TC packets and they will interfere the transmissions of data packets. We can make a simple calculation, the whole system has 240 nodes, suppose every node only has one TC packet to be broadcasted, then to forward all TC packets to the whole system using flooding algorithm, each node needs to forward 240 packets. But if TC packets are only broadcasted within one group, then each node only needs to forward 60 packets on average, which is much smaller than what is needed in traditional system.

From Fig 2.13 we can see that as data load increases, the differences between two systems' data collision possibility and delay will become smaller and smaller. The reason is as data load increases, the ratio of TC



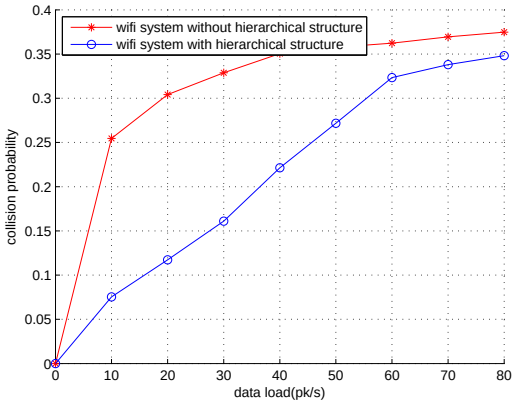
(a) Scenario of the first network layer simulation (b) Scenario of the second network layer simulation

Figure 2.12: Scenarios of network layer simulations

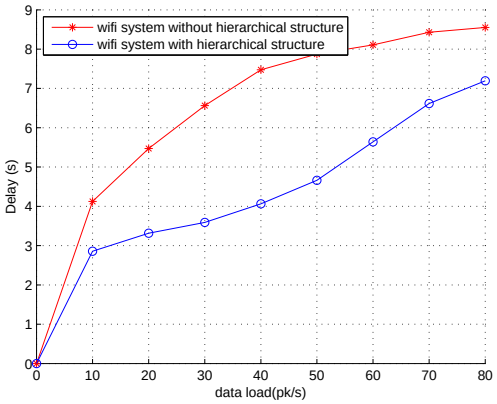
packets in all transmitted packets will drop, so the influence of TC packets will also become less obvious. Another characteristic we can see is when data load is very small or very large, the differences between two systems' throughput will be smaller. When data load is small enough, there are not so many data packets to transmit in a certain time range, so alghough in traditional system the data collision possibility and delay are bigger due to interference of more TC packet transmissions, all data packets can still always be transmitted successfully within this range, causing the throughput in two systems closed to each other. When data load is large enough, the influence of TC packets will become less obvious as we analyzed just now, so the throughput in two systems become closed to each other again. But still, our new system using hierarchical structure always has better performance.

Except to reducing overhead of broadcasting TC packets, our new system can also improve the efficiency of routing process. Fig x shows the scenario we design for our second network layer simulation. In a system which has a square area, nodes are allocated randomly with a density of 60 clients/group. Each node generates data packets whose destination are other random nodes in this system. We will see how the average hop number will change in both traditional system and new system as the system range increases. Here we use the diameter of one group as the unit to calculate the length of system's side.

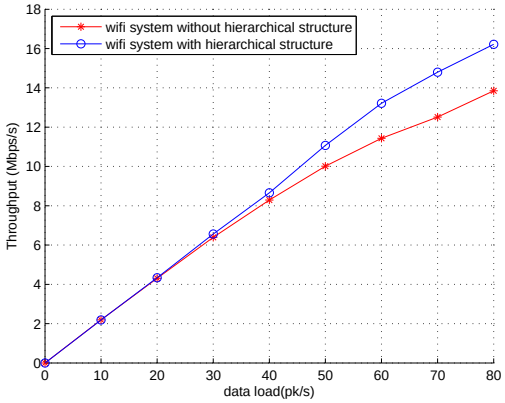
Fig 2.14 shows the simulation result. We can see as the length of system's side increases, the differences between two systems' average hop number will become larger and larger, which is straightforward to understand. We know the reason our new system can improve routing efficiency is that we use mesh routers that have larger communicating range to forward data packets among groups. Larger system will contain more groups, and



(a) Collision possibility



(b) Delay



(c) Throughput

Figure 2.13: Simulation results of network layer

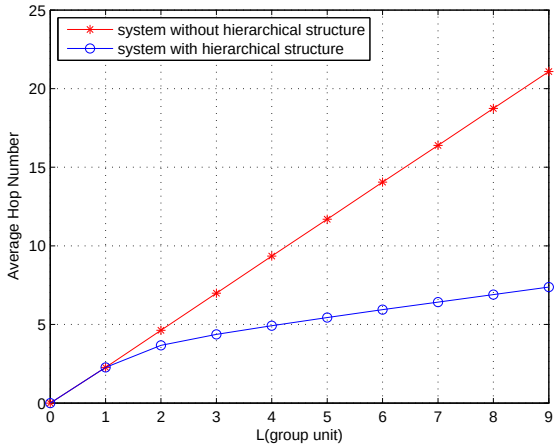


Figure 2.14: Average hop number in routing process

mesh routers will be used more frequently in data forwarding, making the improvement of routing efficiency more obvious.

2.5.3 Implementation

Fig 2.15 is an embedded system designed by our lab. It has several different interfaces so that different network cards such as wifi, zigbee and bluetooth can be integrated on it. The operating system we use is openWrt, which is an open source linux release, and we can implement MAC protocols and routing algorithms based on it.

Figure 2.15: Our lab's experiment platform

Just as we analyzed in Section 2.3, a mesh router can forward data packets between different networks. Based on our lab's embedded system we implement this function and test its performance. Fig 2.16 shows the setup of our demo. Network A and Network B are two wifi networks using different ESSIDs and orthogonal channels. In openwrt, to set up such a network, we need to write different script files to different boards. For example, for a device in wifi network, it's network configuration script should be written like this:

```
ifconfig wlan0 down
iwconfig wlan0 mode ad-hoc
iwconfig wlan0 essid "wifi"
iwconfig wlan0 channel 11
ifconfig wlan0 up
ifconfig wlan0 192.168.2.1
```

In such situation, data packets can not be exchanged between network A and B directly. The mesh router in this demo has two wifi interfaces, one is in network A and another is in network B. To enable the mesh router's forwarding function, we need to run the following script in mesh router:

```
iptables -A forwarding_rule -i wlan0 -o wlan0 -j ACCEPT
iptables -A forwarding_rule -i wlan1 -o wlan1 -j ACCEPT
iptables -A forwarding_rule -i wlan0 -o wlan1 -j ACCEPT
iptables -A forwarding_rule -i wlan1 -o wlan0 -j ACCEPT
```

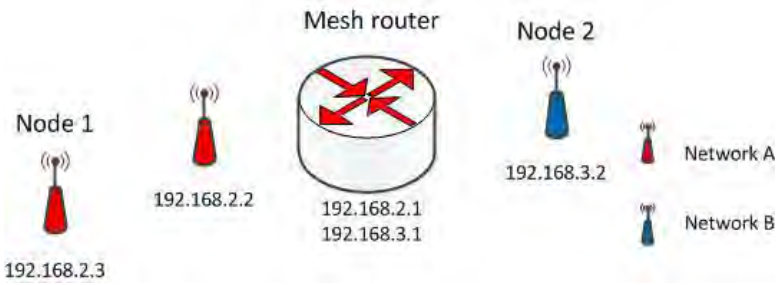


Figure 2.16: The setup of our demo

Also, in a mesh network, to let a device know the next hop of its destination, a routing program is needed. Here in our system, we use an open source routing program called OLSRD. By editing a configuration file, we can run OLSRD properly. A configuration file looks like this:

```
DebugLevel 0
IpVersion 4
Hna4{
192.168.3.0 255.255.255.0
192.168.2.0 255.255.255.0
}
Interface "wlan0"
{
AutoDetectChanges yes
}
```

Now we let node 1 transmit TCP packets to node 2, and use iperf tool to test whether the transmission can be done and what the bandwidth will be. 802.11g is used in this test.

If the bandwidth of a one hop link is x, the bandwidth in our demo should be x/2 if the system works, because at the same time only one device in network A can transmit, and since network B use another orthogonal channel, it won't influent bandwidth.

Interval	Transfer	Bandwidth
0.0-5.0 sec	6.90 MBytes	11.5 Mbits/sec
5.0-10.0 sec	6.5 MBytes	10.9 Mbits/sec
10.0-15.0 sec	7.2 MBytes	12.1 Mbits/sec
15.0-20.0 sec	6.2 MBytes	10.4 Mbits/sec
0.0-20.0 sec	6.8 MBytes	11.2 Mbits/sec

Table 2.3: TCP bandwidth of the demo

Table 2.3 shows the testing result. We test the bandwidth for 20 seconds and finally the average bandwidth is 11.2 Mbits/sec. Considering one hop bandwidth using 802.11g is usually around 20 25 Mbits/sec, this result can prove that our system works well.

2.6 Summary

In this chapter, a new system model with hierarchical structure is proposed for heterogenous wireless mesh network , and based on this model, new MAC protocol and routing algorithm are designed. In MAC layer, different active periods are allocated to different network devices so that system can achieve similar or even higher throughput with smaller collision possibility and delay. In network layer, we fully utilize system's hierarchical structure to reduce the overhead of route table establishing process while improving routing efficiency. Simulation results are given to prove our system's performance, and a demo is also done based on our lab's platform.

Chapter 3

IoT Application Development Based on MQTT Protocol

3.1 Introduction to MQTT

MQTT(Message Queuing Telemetry Transport) is a messaging protocol proposed by IBM. Just like HTTP, it's an application layer protocol but more lightweight, which requires lower transmission rate and consumes less power when transmitting data. Thus it's very suitable for IoT devices who are usually very simple and power limited.

Recent years, MQTT has been widely used in many applications. There is an open source project which implements MQTT called Mosquitto, it has a lot of versions, for example, python version, node version and java version. So using mosquitto, we can develop all kind of MQTT based applications.

MQTT is based on publish/subscribe mechanism, in an MQTT network, there are two kind of devices: clients and brokers. Fig 3.1 shows the basic structure of an MQTT network.

A client is usually a sensor device or user device, it pushes its own data to a broker and also receives certain data whose topic names are what it subscribes from broker. A broker is a server who receives data from clients and pushes these data to clients who subscribe them. For example, in Fig 3.1, client B and C subscribe topic "shanghai/temperature" to broker, when client A publishes a data packet whose topic name is "shanghai/temperature", the broker will find that B and C have subscribed this topic , then it will push this packet to these two clients.

Since the design principle of MQTT is to be lightweight, the packets of MQTT should also be short and

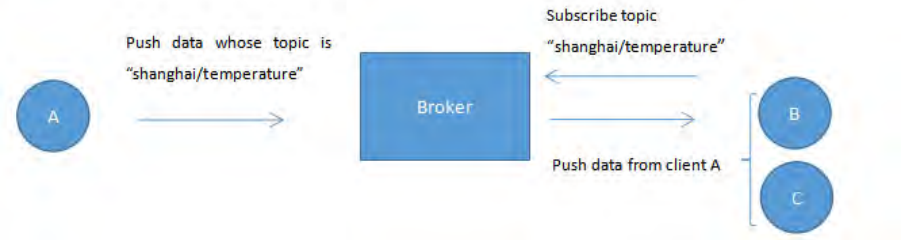


Figure 3.1: Basic structure of an MQTT network

simple. The basic structure of an MQTT packet is described by Fig 3.2. It is composed by three parts: Fixed header part, Variable header part and Payload part.

Fixed header
Variable header
Payload

Figure 3.2: Basic structure of an MQTT packet

Every packet must contain a Fixed header part. It's format is shown by Fig 3.3.

Bit	7	6	5	4	3	2	1	0
Byte 1	MQTT packet type				Flags			
Byte 2	Remaining Length							

Figure 3.3: Basic structure of an MQTT fixed header

In this part the most important field is MQTT packet type filed, which is a 4-bit area. Its different values represent different packet types respectively, the following table shows all packet types in MQTT and their type value:

From table 3.1 we can see there are only 14 kind of packets in MQTT.

Variable header part is not in all kind of packets, also among those who have Variable header, their contents are quite different due to their types.

Finally, the payload part is not in all kind of packet either. This part mainly contains application information, for example, in a publish packet, the data part is contained in this part.

In the follow sections, we will introduce the functions of different type of packets, then we will introduce the principle of some mechanisms such as QoS and topic matching in MQTT.

Packet Type	Value	Packet Type	Value
Reserved	0	SUBSCRIBE	8
CONNECT	1	UNSUBSCRIBE	9
CONNACK	2	SUBACK	10
PUBLISH	3	UNSUBACK	11
PUBACK	4	PINGREQ	12
PUBREC	5	PINGRESP	13
PUBREL	6	DISCONNECT	14
PUBCOMP	7	Reserved	15

Table 3.1: MQTT packet types and type values

3.1.1 CONNECT/CONNACK and DISCONNECT

When a client wants to communicate with a broker, first a connection link should be built. In MQTT, CONNECT packet is used to establish a link. This packet can only be sent from a client to a broker, in another word, only a client can initiate a link. The type value of a CONNECT packet is 1, and variable header is present.

In CONNECT's variable header, four fileds are contained: Protocol Name, Protocol Level, Connect Flags and Keep Alive.

Protocol Name is just a utf-8 encoded string whose content is "MQTT", which represents that the protocol type used is MQTT.

Protocol Level is a value that represents the revision level of the protocol used by client, then the broker will use this field to decide whether the client's revision level is supported.

Connect Flags field consists of some parameters that will influence the behaviors of this connection, for example, one of flags is "Will QoS", it's a 2-bit area which represents the QoS level of this connection.

Keep Alive is a 16-bit area which represents a time interval in seconds. To keep a link alive, a client must transmit at least one packet in every Keep Alive time, this packet can be any type, for example, SUBSCRIBE or PUBLISH, if a client doesn't have data to transmit, then it need to transmit PINGREQ to keep a link alive.

In CONNECT's Payload part, it contains several fields which are determined by flags in variable headers, for example, if in variable header, "User Name" flag is set, then in payload part a user name field must be contained which has a string represent user name. What's more, among all fileds in payload, Client Identifier field must be contained. This field is an ID to help broker distinguish different links, thus each link session should have a unique Client Identifier. A Client Identifier must be a utf-8 string with a length between 1 and 23.

Related to CONNECT, CONNACK should only be sent from a client to a broker. The fixed header is similar to CONNECT except for the type value. The structure of variable header is very simple, as Fig 3.4 shows:

The most important field is Connect return code field, it represent a value. Sometimes although the broker

	Description	7	6	5	4	3	2	1	0
Connect Ack Flags		Reserved							
Byte1		0	0	0	0	0	0	0	SP
Connect Return Code									
Byte2		X	X	X	X	X	X	X	X

Figure 3.4: Variable header part

has received a CONNECT, for some reasons broker can't approve the establishment of the link, and to inform clients the reasons, it set connect return code field to certain value. So when a client sends a CONNECT to a broker, it must wait for CONNACK, and then check the connect return code field. Only when the code shows that broker has approved this link, a link is successfully built.

Disconnecting is a contrary procedure. A client sends a DISCONNECT packet to tell the broker to disconnect the link cleanly.

3.1.2 PUBLISH

A PUBLISH packet can both be transmitted from a client to a broker or from a broker to a client. It's used to transmit data information, for example after a link is established, if a client wants to transmit data to a broker, it just sends a PUBLISH packet containing the data it wants to transmit.

Fig 3.5 shows the structure of PUBLISH's fixed header. Except for type value field, there are three other fields: DUP flag, QoS level and RETAIN. DUP flag indicates whether this packet is sent the first time. If DUP flag is 1, this packet may be a re-transmitted packet due to last time's failure. QoS field decides the QoS level of this packet.

Bit	7	6	5	4	3	2	1	0
Byte 1	MQTT Control Packet type				DUP flag	QoS Level		RETAIN
	0	0	1	1	X	X	X	X
Byte 2	Remaining Length							

Figure 3.5: Structure of the fixed header of PUBLISH packet

Since this packet is used to transmit data, and MQTT is based on topic publish/subscribe, then every publish packet should have a topic name, it is contained in "topic name" field of variable header part. A topic name must be a utf-8 encoded string, and in MQTT, topic names are in a format like "xxx/xxx/xx..". For example, a topic name can be "shanghai/temperature", or "shanghai/minhang area/weather".

Finally, the data are contained in payload part, all encoded in utf-8 format.

After a device send a PUBLISH packet, it will wait for a response so that can decide whether the transmission has succeeded. Different QoS levels expect different type of responses.

3.1.3 SUBSCRIBE/SUBACK and UNSUBSCRIBE/UNSUBACK

A SUBSCRIBE packet can only be transmitted from a client to a broker. The fixed header part of SUBSCRIBE is similar to CONNECT except type value. In payload part, topic filters are contained to inform broker what kind of data it is interested in.

A topic filter is used to match topic names, and the matching rules are simple. There are two mainly used wildcards in MQTT, "+" and "#".

"+" is a single level wildcard which can match any strings in one level. For example, if a client¡¯s filter is "sports/+/player1", then the following topic names will match this filter:

```
"sports/player1"
"sports/football/player1"
```

And the following topic names will not match:

```
"sports/football/player2"
"sports/football/China/player1"
```

"#" is a multiple level wildcard which can match any strings in multiple levels. For example, if a client's filter is "sports/football/#", then the following topic names will match:

```
"sports/football/player1"
"sports/football/China/player1"
```

And the following topic names will not match:

```
"sports/basketball/player1"
"sports/player1"
```

After sending a SUBSCRIBE, a client needs to wait for a SUBACK. A SUBACK packet will contain return code in its payload to tell a client whether the subscription has succeeded, according to different QoS level, the return code may also be different.

Unsubscribing some topic names is a contrary procedure. A client will send an UNSUBSCRIBE packet which contains topic filters, then a broker will cancel subscriptions of those who match filters, if succeed, it will return a UNSUBACK with a certain return code.

3.1.4 PINGREQ and PINGRESP

In every link there is a time interval, between each time interval, at least one packet should be transmitted otherwise the broker will regard this link as inactive and then disconnect it. Any kind of packets can be used to keep a link alive, for example, if during a time interval, a PUBLISH packet is transmitted, then the broker will think this link is alive. However, sometimes clients don's have data to transmit, then PINGREQ should be used in order to keep link alive. When received a PINGREQ, the broker should respond with a PINGRESP packet.

PINGREQ and PINGRESP are both very short packets so that it won't cost too much bandwidth and energy to keep a link alive.

3.1.5 Quality of Service

To make MQTT fit more application scenarios, QoS is also defined in this protocol. According to different QoS level, MQTT package will be delivered in different procedures.There are three QoS levels in MQTT: 0(at most once), 1(at least once), 2(only once). Two flag field in a package is related to QoS, they are QoS field and DUP field. QoS field must be 0, 1 or 2, representing different levels. DUP represents how many times this package has been retransmitted, thus at first it must be 0.

In QoS 0, a sender only sends a package once without waiting to get any response from receivers. For a receiver, when it receives a QoS 0 package, it will not reply anything. Thus QoS 0 package's delivery is unreliable.

In QoS 1, when a sender sends a package, it will wait for a PUBACK pakcage. If after timeout, PUBACK still doesn't arrive, the sender will retransmit this package with the same package id, and also set the DUP flag to non-zero. As for the receiver, once it receives a QoS 1 package, it must reply a PUBACK package. After sending this PUBACK package, next time if it receives another package with the same package id, it will treat this package as a new package. Thus in QoS 1, there is a possibility that a package is delivered more than once, that's why it's called at least once.

In QoS 2, when a sender sends a package, it will wait for a PUBREC package from the receiver. After receiving a PUBREC, it will send a PUBREL, and then wait for a PUBCOMP. Until a PUBCOMP is received,

the whole delivery process is finished, otherwise if timeout is over and a PUBREC or PUBCOMP still doesn't arrive, the sender will retransmit package. For the receiver, when it receives a QoS 2 package, it first replies a PUBREC, and wait for a PUBREL. In this process, if it receives a package with the same package id, it must treat it as the same package rather than a new package, and reply a PUBREC. Then after PUBREL has arrived, it replies a PUBCOMP, and then next time if another package with the same package id arrives, it will treat this package as a net package. In this way, MQTT protocol can assure that each package will only be delivered exactly once.

Fig 3.6 shows the three kind of QoS delivery procedures.

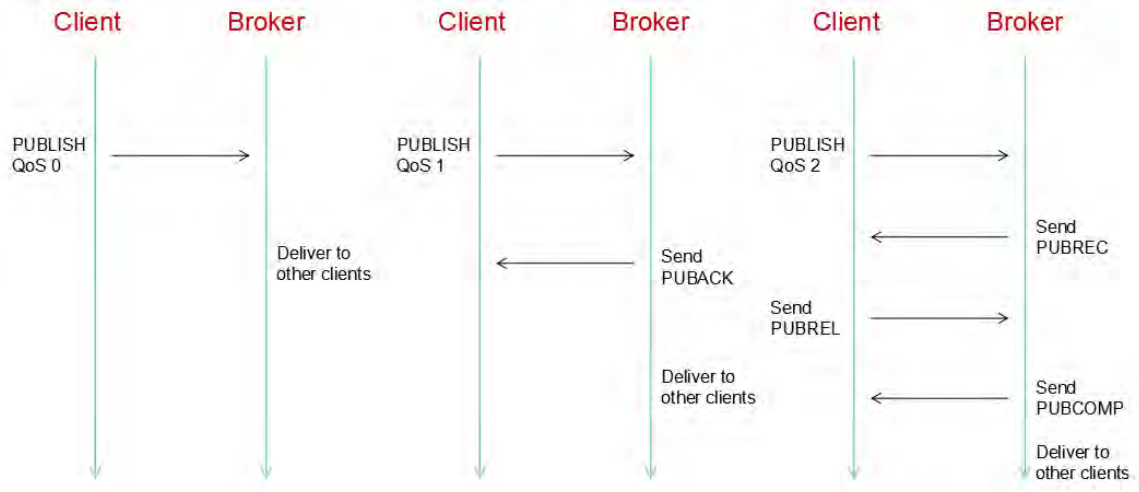


Figure 3.6: QoS flows

3.2 Motivation

MQTT is simple and lightweight, however it's just a messaging protocol, what it provides is just an efficient way for devices to communicate. We can build our IoT system using MQTT, but to make a system useful, we need to add more functions. So first based on typical IoT application scenario, we need to consider what functions we need in an IoT system.

Fig 3.7 shows a typical IoT application scenario. In this system, sensor nodes form a wireless mesh network, they collect data using their sensors, and push these data to the center. A center is not only a broker which receives and dispatches data, but also a management system which can let users monitor the network situation in this network, store and check each node's data, and manage each node's publications and subscriptions. To implement such a system, both client side software and server side software are needed.

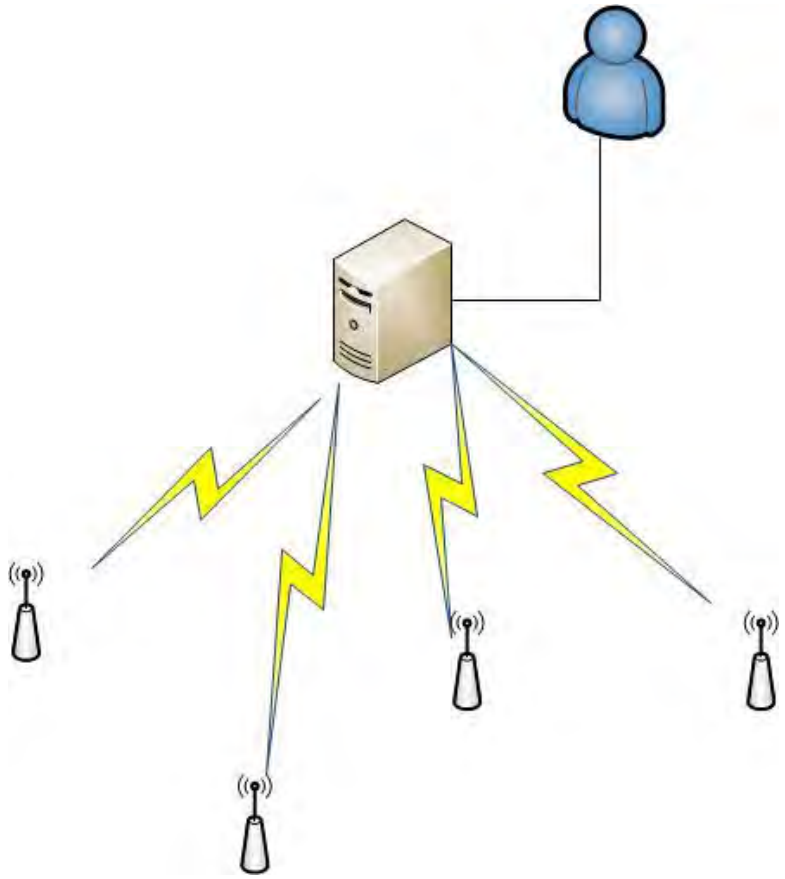


Figure 3.7: Typical IoT application scenario

3.3 Function Design

3.3.1 Device Binding

MQTT is a topic subscribe/publish based protocol, which means "topic" is its key concern rather than devices. Clients publish data using certain topic names, and brokers push data to links who subscribe certain topic names without knowing what the devices are on the other side of these links. However in practical scenarios, users usually not only concern data, but also concern who send these data. For example, suppose in a smart home application scenario, when a user receives a data packet of topic "temperature", he also wants to know is this packet from the thermometer in his bedroom or sitting-room. To implement this function requires our system to bind different device information with different links.

As the former section introduces, in MQTT, each CONNECT packet should has a field named "client identifier", it's a unique id for a link. What we need to do is adding some remark information which can represent this client device to each link on broker side. In a network, what can represent a device? There is

no doubt that IP and mac address are most widely used to distinguish different devices in a network. So every time when a link is established, that means when a client receives CONNACK from the broker, it needs to send an extra packet to the broker which contains its IP and mac address information. This packet can be a PUBLISH packet with a reserved topic , say, "deviceInfo", then the broker should bind these information with the client identifier of this link. Next time when a data packet is sent, the broker will know it's sender's IP and mac address.

What's more, to make a remark information more straightforward, we not only want to know IP and mac, but also want to know what kind of this device is. The extra information should be configured by users , for example, the device can be a air-condition, then we just bind a string "air-condition" with the IP , which requires the server side software to provide a plane for users to edit.

Fig 3.8 shows the whole device binding procedure.

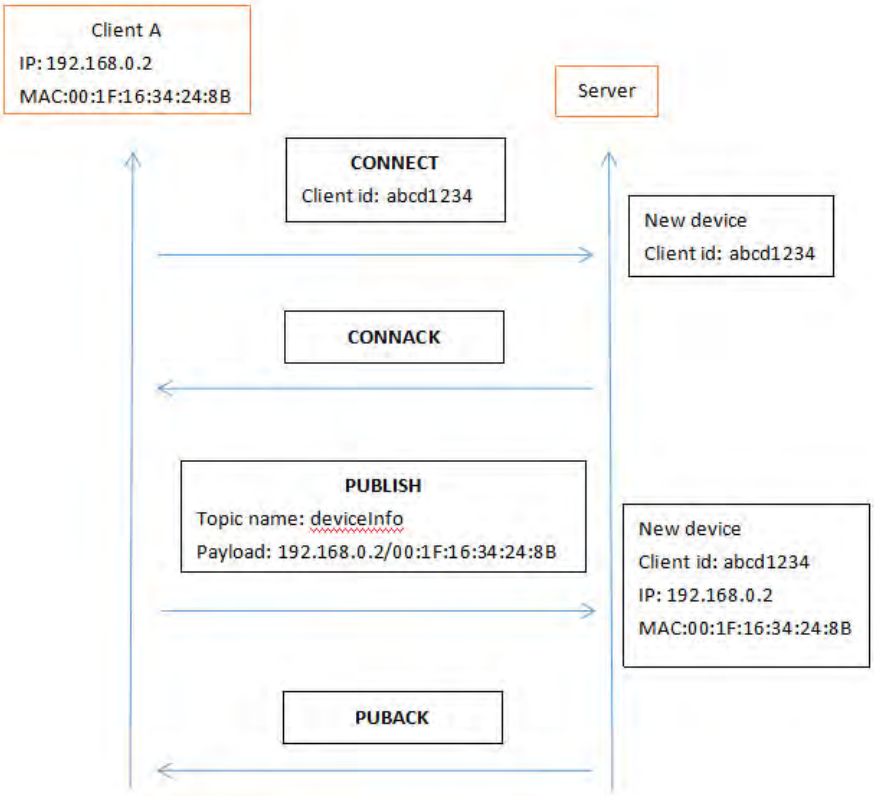


Figure 3.8: Device binding process

3.3.2 Data Query

In MQTT, broker is just a server to forward data packets to certain subscribers. But in a practical IoT system, a center should also provide data query function, which means to let users be able to check data sent by certain devices, or under certain topic names. For example, a user may want to know the temperature records in the last seven days sent by device A. This function requires server side to not only store data records, but only remember the relationships among devices, topic names and these records. Files or database should be used on server side to store these records and their relationships. Also, these records should be displayed in a friendly format, say, a chart.

3.3.3 Publication and Subscription Management

In MQTT PUBLISH and SUBSCRIBE are used to transmit data using certain topic names. But what kind of topic names should be subscribed or published is a problem in practical systems. Since our goal is to make the whole system easy to manage and monitor, we should provide functions for users to check and decide the topic names of each device's subscriptions and publications.

Since topic names of publications and subscriptions are configurations of clients, to change these configurations on server side, we need to design some APIs which conform to RESTful standard. RESTful standard is a kind of software development framework which is mainly used to implement interactions between servers and clients. It provides some design principles and constraints to make codes more simple and organized. In REST web services, every resource has a URL and a method, these two components' combination will define every resouce itself. There are mainly four kind of method in RESTful standard: GET, POST, PUT, DELETE. HTTP web application is a typical case of RESTful system, so we can take the API designs in HTTP applications as a reference. In HTTP applications, the simplest APIs are contained in URL. For example, suppose an HTTP server's URL is "http://example.com", and this server contains many users' data, say, each user's member points. Now a user bought something, and consumes 5 points, to change the record on server side, the user should send an HTTP message to the server using a URL like this :
"http://example.com?userId=someone&pointReduce=5". Here character "?" means the string behind it is command parameters, and server will recognize it. Then after parsing the parameters, it will know user "someone" consumed 5 points, then it will reduce 5 points from this user¡¯s record.

In IoT system based on MQTT, we can use special topic names to design APIs.

To let a client subscribe or unsubscribe topics, a broker can push a PUBLISH packet with topic name "subscribe/client IP" or "unsubscribe/client IP", and in the payload area contains the topic filters that the

broker want the client to subscribe or unsubscribe. Once a client receives a packet with such topic name, it will response with a SUBSCRIBE or UNSUBSCRIBE packet to execute the actions.

Fig 3.9 shows an example. Broker sends a PUBLISH with topic name "subscribe/192.168.0.2" to the related client, the client will fetch the topic filter from payload and then send a SUBSCRIBE to broker using this topic filter.

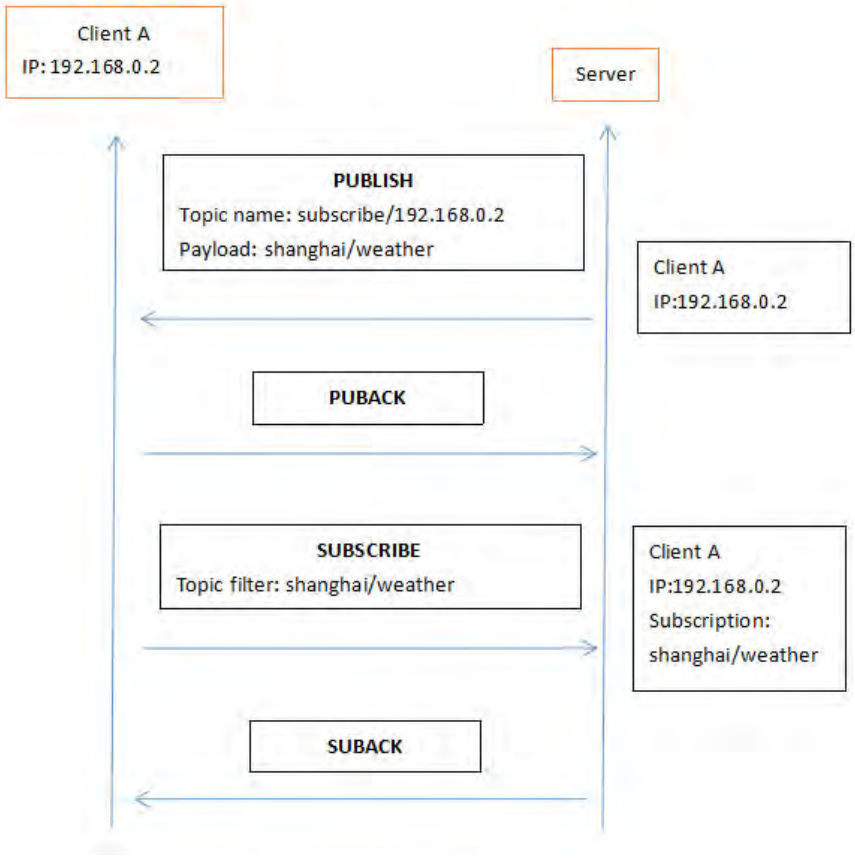


Figure 3.9: Subscription management process

To check all topics a client has subscribed, a broker can push a PUBLISH packet with topic name "query_sub/client IP" to this client. A client will then return a PUBLISH packet with a payload area containing all topic filters it subscribed. Through this procedure, broker can get to know every client's subscriptions.

To let a client publish certain data using another topic name, a broker can push a PUBLISH packet with topic name "change_pub/client IP", and in payload, the content should be in this format: "old topic&new topic". Fig 3.10 shows an example of changing publishing topic name , client A used to publish data using topic name "shanghai/weather", but when it receives a PUBLISH packet with topic name "change_pub/192.168.0.2" and a payload whose content is "shanghai/weather&beijing /weather", then next time it will publish data with

new topic name "beijing/weather".

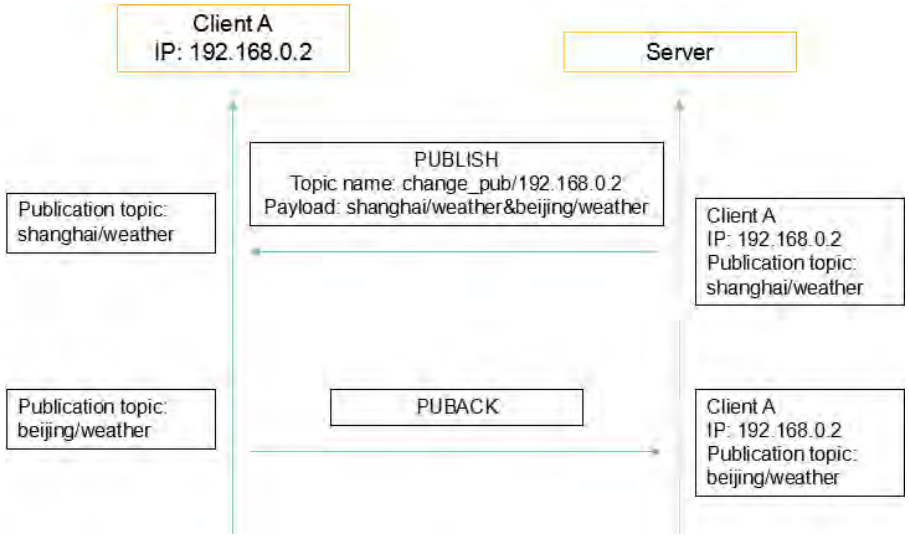


Figure 3.10: Process of changing publication topic names

3.3.4 User Authority Control

Some of the above functions are sensitive, for example, changing subscriptions of clients. Thus not all users have the authority to use all these functions. Thus in our management system, we should also add a user log in function. By giving different user accounts different authorities, we can only allow those who are root users to execute some sensitive actions.

3.3.5 User Interface

All the above functions should be hidden under a friendly user interface, so that users can use this system conveniently without knowing APIs.

Here we should notice that our MQTT management system has two application scenarios, one is smart home, and another is industrial device management. According to the characteristics of these two scenarios, different kind of softwares should be developed, which means two different user interfaces are needed.

In smart home case, sensor devices are users' furniture, and users may want to monitor and control their furniture no matter where they are, for example, before a user coming back to home, he may want to turn on the air conditioner in his house, so here an app on smart phone is more suitable to provide user interface. In our system, we plan to develop an android app using java.

In industrial device management case, the amount of devices is usually very large, thus more information

need to be displayed, for example, devices can form a very complex mesh network, and a user may want to see the topology of this network so that it's more straightforward to assess the whole network's status. Also in such case, users don't have a very high requirement on softwares mobility, thus a PC version of web application is more suitable to provide user interface. To build such a PC web application, in back end, we choose to use python and a famous framework called webpy. In front end, we just use standard html, css and javascript. Notice in this system, a difficult problem is to draw topology figure. In back end, the system needs to collect the whole network's information, we can use pulling mechanism to ask every node one by one, and when asked, each node just sends its own neighbors to server. After collecting all topology information, in front end, we need to map these information into a figure in browser. In HTML5, a new technology called canvas is proposed, which can let developers draw beautiful figures without using flash plugins. Also here we use a graphic library called Echarts to help us implement fancy pictures.

3.4 Implementation

The whole system can be divided into two parts: client side part and server side part.

3.4.1 Client Side

The client side software should be run on sensor devices. Fig 3.11 shows the basic structure of client side software. Three modules should be included: data collecting module, MQTT module and logical processing module.

According to nodes' different sensors, data collecting modules are also different. The main function of this module is to get data from sensors, and then give these data to logical processing module.

MQTT module is a implementation of MQTT client. There are a lot of open source MQTT implementations, one the the most famous is mosquitto, we use a python version as our MQTT communication module. It's function is to receive data from broker and give these data to logical processing module, also it provide APIs for processing module to transmit data to broker. To import this module in python is easy, you just need to install it and use the following codes:

```
import paho.mqtt.client as mqtt
client = mqtt.Client()
```

Logical processing module's job is to respond to the broker according to different APIs and packet types.

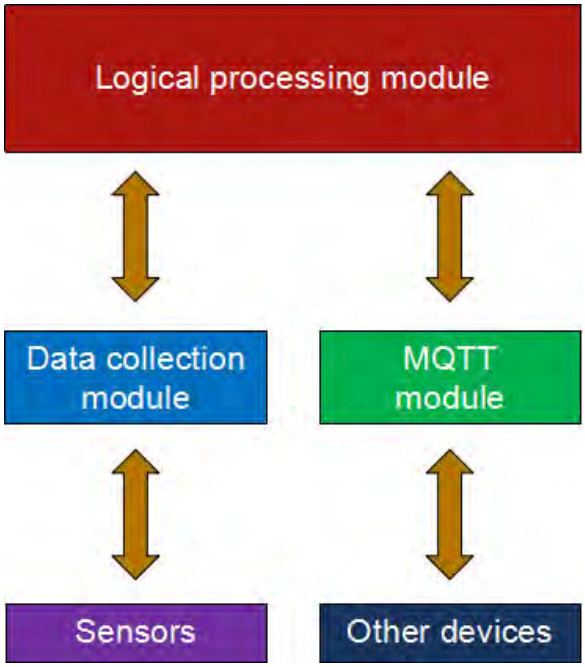


Figure 3.11: Basic structure of client side software

When it first connects to a broker, it need to send its IP and mac address so that broker can bind these information to the link. Using python, we can implement this part in the following codes:

```
def on_connect(client, userdata, rc):
  client.publish("dataInfo", payload)
client.on_connect = on_connect
pubTopic=""
```

Here "dataInfo" is a reserved topic name in our system, which indicates that this packet is used to bind device information. "payload" is a dict type variable containing this client¡¯s IP and mac address.

When it receives API packets from the broker, it need to execute different actions. The following codes implement this part:

```
def on_message(client, userdata, msg):
  if(msg.topic=='subscribe/clientIP'):
    client.subscribe(msg.payload)
  elif(msg.topic=='unsubscribe/clientIP'):
    client.unsubscribe(msg.payload)
  elif(msg.topic=='query_sub/clientIP'):
```

```
            client.publish(topicSubs)
        elif(msg.topic=='change_pub/clientIP'):
            changePub(msg.payload)

    def changePub(pubString):
        str=pubString.split('/')
        oldPub=str[0]
        newPub=str[1]
        if(i=pubTopic.index(oldPub)):
            pubTopic.pop(i)
            pubTopic.append(newPub)

    client.on_message=on_message
```

These codes use some functions implemented in other parts.

Thus through this logic flow, sensor nodes can communicate with broker, and do different logical procedures according to different APIs.

3.4.2 Server Side

On server side, the whole software includes two modules: MQTT module and User management module. Just like client side software, we still choose mosquitto as our server side MQTT module. As to User management module, we need to provide a friendly user interface to users, and according to users' operations, the user interface will call different functions to execute certain actions. Here in our system, in industrial scenario, we develop a web application to interact with users, which means users can access User management system through a browser. And in smart home case, we develop an app on android system to let users use our system on their smart phone. Fig 3.12 shows the structure of server side software. We willl introduce its functions one by one.

(1) User login

To use management system, first a user needs to log in. Open browser and type in the URL of our management system, the first page is just the login page, as Fig 3.13 shows:

After logging in, an overview page will be displayed to show a basic situation of the whole network, from which you can see all devices' basic information, including their IP and mac addresses, their device names, etc.

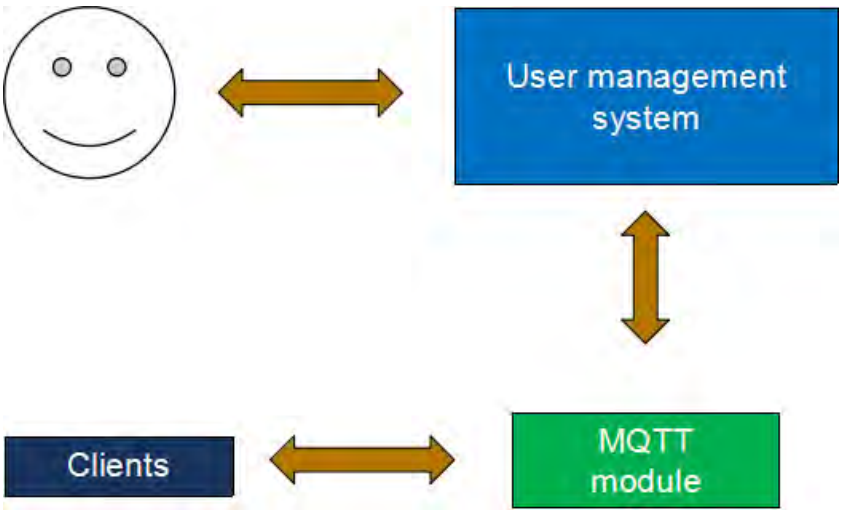


Figure 3.12: Structure of server side software

Figure 3.13: User login

In industrial management case, users will see an interface in their browsers as Fig 3.14 shows.

And in smart home case, Fig 3.15 is what users will see on their smart phones.

There are different type of user accounts, which can be divided into administer, user and guest. Guests can only check data records of limited devices, while users can check data and do some configurations to limited devices. Only administers can check data and do configurations to all devices.

In the back of user interface, the codes to implement such kind of login logic are also written in python:

Figure 3.14: Basic information displaying(PC version)

```
def Authenticate(func):
  def wrapper(*args,**kw):
    flag=web.cookies().get('username')
      if flag==None:
        raise web.seeother('/')
      else:
        return func(*args,**kw)
    return wrapper


class loginHandler:
  def GET(self):
    if web.cookies().get('username')!=None:
      raise web.seeother('/index')
    else:
      return render.login(error='false')


  def POST(self):
    account=web.input()
```

Figure 3.15: Basic information displaying(Mobile version)

```
username=account.get('username')
password=account.get('password')
if password=='admin':
  web.setcookie('username','admin',3600)
  raise web.seeother('/index')
else:
  return render.login(error='true')
```

(2) Device binding

Every time a new link is established, an alert window will pump out in management system just as Fig 3.16 shows:

IP and mac address has been bound through the procedure explained in previous sections. What the alert window want users to do is to add a remark so that the device's function will be more obvious. For example, if a user knows this device is from a air-condition, then he can type in "air-condition" in remark item. Then we

Figure 3.16: Configuration window for device binding

Figure 3.17: Data displaying(PC version)

will return back to overview page, and we can see the new device has already been added to the list.

(3) Data query

Every time the server receives a data packet from a link, it will add a data record to a device's item which is bound to this link. The basic structure of an item is an object in json format which records a device's IP

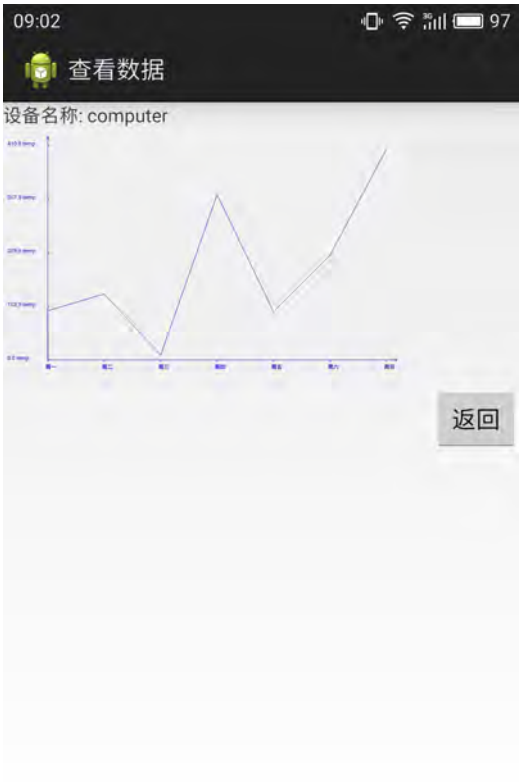


Figure 3.18: Data displaying(Mobile version)

address and its data records. For example, a device whose IP is "192.168.0.2" should have an item like this:

```
{ deviceIP: "192.168.0.2",
data: [22.2, 23.4, 28.5, 29.1, 30.0, 31.2.....] }
```

All devices' data items are saved in a database table or a file using json format with name "deviceData". In industrial scenario, there are a lot of devices, so we need to use database, here we can choose sqlite or mysql. In smart home case, because the data amount is not very large, we can just use file to store these data:

```
import os
import json

class fileReader:
  def __init__(self,fileName):
    self.path=os.path.join(os.path.abspath('.'),fileName)
    self.content={}
```

Figure 3.19: Subscription/publication management window(PC version)

```
def readData(self,lineNum):
  with open(self.path,'r') as f:
    lineContents=f.readlines()
    line=lineContents[lineNum].strip()
    self.content=json.loads(line)
  return self.content


def writeData(self,data,lineNum):
  with open(self.path,'r') as f:
    lineContents=f.readlines()
    maxLine=len(lineContents)
    line=self.readData(lineNum)
    line.append(data)
    lineContents[lineNum]=json.dumps(line)+'\n'
    tmp=''
    for ele in lineContents:
      tmp+=ele
```

Figure 3.20: Subscription/publication management window(Mobile version)

```
with open(self.path,'w') as f:
  f.write(tmp)
```

In the overview page, each row has a button named "check", when a user clicks it, a data query request will be submitted to the system, and a page which displays this device's data in a diagram format will be shown. In PC version, a data page looks just like Fig 3.17, while Fig 3.18 shows the mobile version.

(4) Subscription/publication management

Through our user management system, we can not only check data, but also configure subscriptions and publications of devices. To do these configurations to a device, just click the "config" button of the device's row, then a configuring window will pump out as Fig 3.19 shows in PC version and 3.20 in mobile version. PC version is implemented using javascript and python, while mobile version is implemented using java.

In this window, we can see the topics subscribed by this device, this information is queried by sending an API packet with topic "query_sub/client IP" to the device. Also, in "subscribe" item, we can type in new topic filters to let this device add new subscriptions, and in "unsubscribe" item, we can choose topic filters that

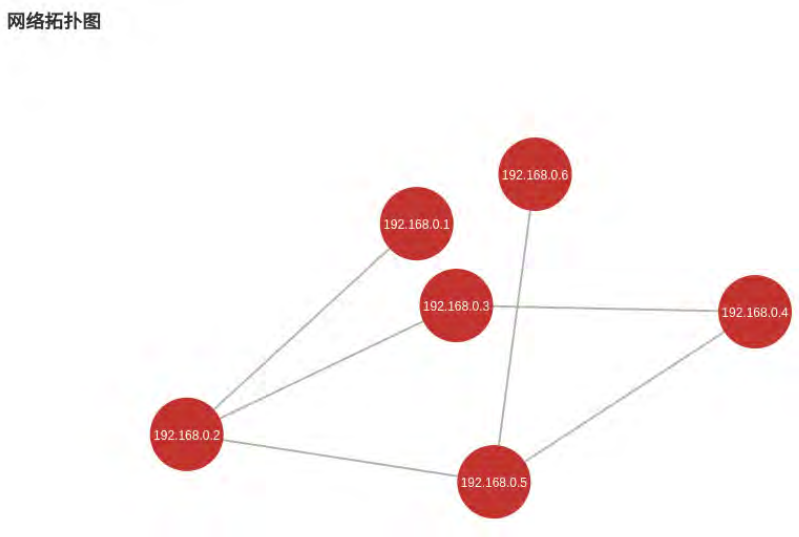


Figure 3.21: Topology figure of a mesh network

we want the device to unsubscribe. What's more, in "change publication" item, we can type in both old and new topic to let the device publish certain data using new topic name instead of old name. Finally, when we finished editing, we can click "submit" button to submit our changes.

What's more, in industry case, considering there can be a large amount of devices which form a complex mesh network, users may want to see the topology of the whole network, thus in PC version, we also implement a function which collects the network's topology information and draws a figure in browser using HTML5's newest technology called Canvas. Fig 3.21 shows an example of a network's topology figure. By looking at this figure, it's straightforward for us to know that there are 6 nodes in this network, and the link relationships between these nodes are also very clear.

3.5 Summary

In this chapter, a lightweight message pushing protocol called MQTT is introduced. Based on MQTT, we present an IoT application system. Function designs are first presented, including device binding, data query, publication and subscription management, etc. According to our function designs, we implement our system both on PC platform and mobile platform. A practical demo system has been set up in our lab, and it works well.

Chapter 4

Conclusion

4.1 Contributions

The contributions of this thesis are listed below.

- A new system model is proposed to solve existing problems in typical application scenarios of IoT system.
- Based on our new system model, we design a new MAC protocol to make different devices coexist with minimum interferences to each other. Additionally, a new routing algorithm is developed to reduce the overhead in establishing route tables.
- Simulations are done to evaluate performance of our newly designed protocols. A simple implementation of mesh router is also done based on our lab's embedded system.
- Based on MQTT, an IoT application system is developed which can not only make devices exchange data with each other, but also let users manage the whole network through a management system. A prototype system both based on PC platform and mobile platform has been implemented.

4.2 Future Work

In spite of the work we have done, there are still many interesting problems to be solved.

In chapter 2, using our lab's embedded system, we test the data forwarding function of our mesh router, howerver, for MAC protocols and routing algorithms, we only do some simulations. In the future, we hope to build a demo system which implements all functions.

In chapter 3, we have already built a prototype system. This system is suitable for some simple scenarios like smart home, but in industry scenario, more clients are in the same network, to manage such a complicated network, new functions and softwares need to be developed. When the network is complicated, users may want to see the topology of the whole network. In addition, the way users manage the network should also be changed, for example, we need to add automatic configuration function to the system, using this function, users just need to create some rules, then all clients in the system will maintain their own configuration information automatically by following these rules. We are still working on such a new system, some functions have already been implemented.

What's more, in practical scenarios, there are a lot of legacy IoT devices which can't run MQTT protocol. How to enable these devices to access MQTT network is an interesting problem. Our basic idea is to design a gateway. Legacy devices can connect to this gateway, through which their data will be transformed into MQTT format, and then forwarded to MQTT network. Users can also manage these devices using this gateway's management system. Fig 4.1 shows the application structure. A patent Wang and Ma (2017) based on this idea is in application.

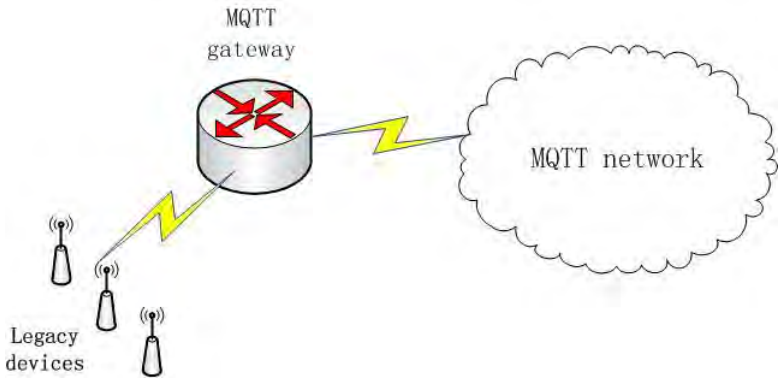


Figure 4.1: MQTT gateway

Acknowledgments

I would like to express my gratitude to all the people who give me a lot of help during my master study.

First, I gratefully acknowledge support and instructions from my advisor, Prof. Xudong Wang. I learned a lot and made solid progress in different aspects with his help. Prof. Wang not only taught me how to do research, but also showed me what a true researcher should be with his own behaviors. He told me I should have a strict mind and positive attitude no matter in normal life or in study. He also encouraged me to keep eyes open on other areas, which will help me a lot in my future career.

I also want to thank Prof. Chong Han and Prof. Weikang Qian who gave me a lot of valuable suggestions to improve my thesis.

What's more, I would like to thank all friends in my lab, Aimin Tang, Chao Xu, Bangzhao Zhai, Cheng Huang, Mengxin Yu, Peng Shi, Dianhan Xie and Jiawei Zhang. It has really been great experience to work with them.

Finally, I would like to give my deepest gratitude to my parents for their support and love. Without their support, I will never be successful.

Appendix A

Codes and Patents

A.1 Codes

- Source code of simulations in chapter 2: http://wanglab.sjtu.edu.cn/userfiles/files/Simulation_program.zip
- Source code of android app in chapter 3: http://wanglab.sjtu.edu.cn/userfiles/files/mqtt_app.zip

Bibliography

Akyildiz, I. F., W. Su, Y. Sankarasubramaniam, and E. Cayirci (2002). Wireless sensor networks: a survey. *Computer Networks 38*(4), 393–422.

Akyildiz, I. F., X. Wang, and W. Wang (2005). Wireless mesh networks: a survey. *Computer Networks 47*(4), 445–487.

Atzori, L., A. Iera, and G. Morabito (2010). The Internet of Things: A survey. *Computer Networks 54*(15), 2787–2805.

Bianchi, G. (2000). Performance analysis of the IEEE 802.11 distributed coordination function. *IEEE Journal on Selected Areas in Communications 18*(3), 535–547.

Chan, M., E. Campo, D. Est¨¨ve, and J. Y. Fourniols (2009). Smart homes - current features and future perspectives. *Maturitas 64*(2), 90–97.

Clausen, T. (2003). Optimized Link State Routing (OLSR). *Ietf Rfc*.

Crow, B. P., I. Widjaja, L. G. Kim, and P. T. Sakai (1997). IEEE 802.11 Wireless Local Area Networks. *IEEE Communications Magazine 35*(9), 116–126.

Eklund, C., R. B. Marks, K. L. Stanwood, and S. Wang (2002). IEEE Standard 802.16: A technical overview of the WirelessMAN Air Interface for broadband wireless access. *World Telecommunications 40*(6), 98–107.

February (2011). The WebSocket protocol.

Ferrus, R., O. Sallent, and R. Agusti (2010). Interworking in heterogeneous wireless networks: comprehensive framework and future trends. *IEEE Wireless Communications 17*(2), 22–31.

Gislason, D. (2008). *Zigbee Wireless Networking*.

Golmie, N., O. Rebala, and N. Chevrollier (2003). Bluetooth adaptive frequency hopping and scheduling. In *IEEE Conference on Military Communications*, pp. 1138–1142.

Haartsen, J. C. (2000). The Bluetooth radio system. *IEEE Personal Communications 7*(1), 28–36.

Han, X., X. Cao, E. L. Lloyd, and C. C. Shen (2009). Fault-Tolerant Relay Node Placement in Heterogeneous Wireless Sensor Networks. *IEEE Transactions on Mobile Computing 9*(5), 643–656.

Hickson, I. (2014). HTML5: A vocabulary and associated APIs for HTML and XHTML.

Lee, S. K., K. Sriram, K. Kim, Y. H. Kim, and N. Golmie (2009). Vertical Handoff Decision Algorithms for Providing Optimized Performance in Heterogeneous Wireless Networks. *IEEE Transactions on Vehicular Technology 58*(2), 865–881.

Li, N. and J. C. Hou (2006). Localized topology control algorithms for heterogeneous wireless networks. *IEEE/ACM Transactions on Networking 13*(6), 1313–1324.

Liuhto, L. and V. Mantysaari (2004). Extensible Messaging and Presence Protocol (XMPP). *University of Helsinki Department of Computer Science*.

Lu, K., Y. Qian, M. Guizani, and H. H. Chen (2008). A framework for a distributed key management scheme in heterogeneous wireless sensor networks. *IEEE Transactions on Wireless Communications 7*(2), 639–647.

Lv, P., X. Wang, and M. Xu (2012). Virtual access network embedding in wireless mesh networks. *Ad Hoc Networks 10*(7), 1362–1378.

Niyato, D. and E. Hossain (2007). A Noncooperative Game-Theoretic Framework for Radio Resource Management in 4G Heterogeneous Wireless Access Networks. *IEEE Transactions on Mobile Computing 7*(3), 332–345.

Reynolds, J. (1985). Rfc959 file transfer protocol (ftp). *www 2*(78), 312¨C319.

Stevensnavarro, E., V. W. S. Wong, and Y. Lin (2007). A Vertical Handoff Decision Algorithm for Heterogeneous Wireless Networks. *Vehicular Technology IEEE Transactions on 57*(2), 3199–3204.

Totty, B., D. Gourley, M. Sayer, A. Aggarwal, and S. Reddy (2002). HTTP: The Definitive Guide. *Oreilly Media 215*(11), 403–410(8).

Wang, X. (2005). An FDD Wideband CDMA MAC Protocol with Minimum-Power Allocation and GPS-Scheduling for Wireless Wide Area Multimedia Networks. *IEEE Transactions on Mobile Computing 4*(1), 16–28.

Wang, X. and L. Ma (2017). A gateway design which enables legacy devices to access IoT network. *A Chinese patent in application*.

Wang, X., W. Wang, and M. P. Nova (2005). Distributed TDMA for wireless mesh network.

Wang, Y., X. Wang, B. Xie, D. Wang, and D. P. Agrawal (2008). Intrusion Detection in Homogeneous and Heterogeneous Wireless Sensor Networks. *IEEE Transactions on Mobile Computing 7*(6), 698–711.