MQTT 协议在移动互联网即时通信中的应用①

马 跃[1], 孙 翱[1,2], 贾军营[1], 孙建伟[1], 于碧辉[1], 杨雪华[3]

[1](中国科学院 沈阳计算技术研究所, 沈阳 110168)
[2](中国科学院大学, 北京 100049)
[3](沈阳师范大学 软件学院, 沈阳 110034)

摘 要: 移动互联网下的即时通信应用现今已经成为人们日常沟通必不可少的工具, 然而作为其开发基础的即时通信协议却始终没有一个统一的标准, 已有的即时通信协议都不能够很好的适应移动互联网网络环境不稳定, 低带宽高延迟, 设备计算能力差等特点. MQTT 协议作为一种基于发布/订阅模型的轻量级消息传输协议, 在移动平台具有节省流量和能耗, 可扩展性强的优点. 本文首先介绍了当前一些主流的即时通信协议, 指出了它们在移动互联网环境下存在哪些缺陷; 之后研究了 MQTT 协议的消息格式与使用方式, 并与已有的即时通信协议进行了对比; 最后基于 MQTT 协议, 对即时通信应用的两项核心功能 IM 和 Presence 进行了设计和实现, 并经过测试表明使用 MQTT 协议能够在移动互联网环境下提供比传统即时通信协议更少的带宽耗费和更良好的用户体验.

关键词: MQTT 协议; 移动互联网; 即时通信; 系统设计; 带宽耗费

Application of MQTT Protocol to Instant Communicationin Mobile Internet

MA Yue[1], SUN Ao[1,2], JIA Jun-Ying[1], SUN Jian-Wei[1], YU Bi-Hui[1], YANG Xue-Hua[3]

[1](Shenyang Institute of Computer Technology, Chinese Academy of Sciences, Shenyang 110168, China)
[2](University of Chinese Academy of Sciences, Beijing 100049, China)
[3](College of Software, Shenyang Normal University, Shenyang 110034, China)

Abstract: Instant communication applications in Mobile Internet have become an indispensable part for people's daily communication nowadays. The real-time communication protocol, which is the basis for the development of instant communication applications, however, has no unified standard. The existing real-time communication protocols are not able to perfectly adapt to mobile Internet, which has a constrained network environment with the characteristics of low-bandwidth, limited processing capabilities and high latency. MQTT is a publish/subscribe based, extremely simple and lightweight messaging protocol, whose design principle is to minimize the network bandwidth and device resource requirements. This article first introduces some existing mainstream instant communication protocols and points out their shortcomings in mobile Internet environment. The second part introduces MQTT protocol briefly and studies the format and interaction process of MQTT message. Finally, based on MQTT, the article puts forward an implementation of IM and Presence, which are the core functions of Instant Communication. As a conclusion, the experiment shows that using the MQTT protocol can provide less bandwidth cost and better user experience for instant communication in mobile Internet.

Key words: MQTT protocol; mobile internet; instant communication; system design; bandwidth cost

即时通信, 泛指能够实时发送和接收互联网消息的业务. 发展到现在, 即时通信应用已经成为以通信录管理, 聊天, 用户状态管理为核心的综合化信息平台. 随着移动互联网近些年来的飞速发展, 即时通信的应用平台已经从传统的桌面个人计算机扩展至手机等移动设备, 成为人们日常生活中不可或缺的一部分.

① 收稿时间:2015-06-29;收到修改稿时间:2015-09-06

即时通信协议作为即时通信应用实现的基础, 其重要性不言而喻. 为了规范化即时通信协议, 在 2000 年 2 月份, IETF(Internet Engineering Task Force)发布了由 IMPP(Instant Messaging and Presence Protocol)工作组提出的即时通信与在线状态管理系统模型(RFC 2778)[1], 以及该系统模型中的各项需求(RFC 2779)[2]. 在此基础上衍生出了 SIMPLE 和 XMPP 两个当前主流的即时通信协议. 然而由于移动互联网天然的具有网络环境不稳定, 低带宽高延迟的特点, 加上移动设备对能耗和流量消耗较为敏感, 使得上述两种协议都不能够很好的适应移动互联网下即时通信的应用场景.

MQTT(MQ Telemetry Transport)协议作为 IBM 公司提出的一种专门为物联网设计的协议, 其设计理念就在于最小化通信的网络带宽耗费以及设备资源的占用率. 本文将分为四个部分, 第一部分分析当前主流的即时通信协议, 并阐释它们在移动互联网环境下存在的不足. 第二部分简单分析 MQTT 的消息格式, 指出 MQTT 协议的优势和缺点, 阐述了将 MQTT 协议应用到即时通信中时的信息交互流程. 第三部分将在第二部分的基础上, 对即时通信的两个核心功能: Presence(出席状态管理)和 IM(实时聊天), 提出设计方案和具体实现. 第四部分将对实现的性能进行测试, 并与 SIMPLE 与 XMPP 在同等测试条件下的表现进行对比, 得出结论.

1 背景

除去一些私有协议外, 目前较为主流的即时通信协议分别为: 即时通信对话初始协议和表示扩展协议(SIMPLE)以及基于 XML 的可扩展通讯和表示协议(XMPP). 下面对这两种协议进行一个简单的介绍.

SIP 协议是一个用于信令控制的应用层协议[3]. 用于创建、修改和释放一个或多个参与者之间的会话. 其只对管理会话的方式进行了定义, 并不强制要求建立会话的类型. SIMPLE 作为一个基于 SIP 协议的即时消息通信协议族, 在原有信令的基础上还增加了多种方式的请求, 如 MESSAGE 和 NOTIFY 信号.

SIMPLE 协议的优点在于继承了 SIP 协议原有的会话建立流程, 规范性强, 便于扩展和实现. 然而 SIMPLE 协议存在的问题在于, 首先 SIP 的信令和消息传送是基于文本的平面化的数据表达, 解析起来缺少规律性. 其次, SIP 的连接建立通道在 SIP 客户端与服务器之间, 而数据传送通道是在客户端之间直接建立的. 当面对类似多人聊天或会议这种业务需求时就会大大增加客户端的工作负载.

XMPP[4]于 2004 年由 IETF 完成了标准化工作, 形成了核心的 XML 流传输应用层协议. 它继承了 XML 的灵活性和扩展性, 并以 TCP/IP 传输为基础, 定义了客户端、服务器和网关三种角色[5]. 服务器承担了客户端信息记录、连接管理和信息的路由功能. 网关负责各种异构即时通信系统之间的互通.

XMPP 与 SIMPLE 协议对比, 天然的具有易于解析和阅读的特性, 能方便的表达层次化的内容和内在逻辑, 易于根据需求进行扩展并实现代码复用. 另外, XMPP 的连接建立和数据传输通道是一体的, 虽然一定程度上增加了服务器的负载, 然而也能够比较方便的实现如多人聊天, 广播, 数据同步等业务需求.

然而在移动互联网环境下, 这两种主流的即时通信协议都存在有一些不容忽视的问题. 首先, 这两种协议都是基于消息体中的内容进行寻址, 极大的减少了带宽利用率; 其次, 这两种协议都是文本协议, 没有二进制数据传输, 降低了内容传输效率; 最后, 这两种协议都是基于相对可靠的网络环境, 对于移动互联网复杂的网络环境并不能够提供很好的容忍机制.

2 MQTT协议介绍

MQTT 协议对比前两种协议, 拥有以下两个优点:

①MQTT 协议是一个二进制协议, 相比文本协议而言其消息体编码更为紧凑, 可以传递各种类型的数据并对数据内容能够进行有效的屏蔽[6].

②客户端通过向服务器上的某个主题发送消息来转发, 从而避免了根据消息体寻址而导致消息体过大的问题, 有效的提高了带宽利用率.

MQTT 协议通过 MQTT 控制消息实现客户端与服务器之间的交互. 控制消息分为固定头部、可变头部和 Payload 三部分. 其中固定头部格式如表 1 所示.

表 1 MQTT 控制消息固定头部格式

Bit	7	6	5	4	3	2	1	0
Byte1	Packet Type				Flags			
Byte2…	Remaining Length							

在固定头部第一个字节的前四位定义了控制消息的类型, 目前 MQTT 协议共定义了 16 个控制消息[7], 主要可以分为五类, 如表 2 所示.

表 2　MQTT 控制消息类别

MQTT 连接建立和断开	CONNCETCONNACK,DISCONNECT
MQTT 消息发布	PUBLISH PUBACK(QoS=1) PUBREC,PUBREL,PUBCOMP(QoS=2)
MQTT 消息订阅	SUBSCRIBE,SUBACK
MQTT 消息退订	UNSUBSCRIBE,UNSUBACK
MQTT 消息保活	PINGREQ,PINGRRESP

Flags 中的 QoS 位定义消息交付的三种服务质量, 0 表示至多发送一次,可能会发生消息丢失; 1 表示至少发送一次, 能够确保消息到达但可能有重复; 2 表示精确发送一次. 对 QoS 的设置能够保证消息传输的可靠性. 剩余长度部分采用可变长编码, 这使固定头部的长度最短仅为两个字节, 有效节省了带宽. 总体上看, 客户端与服务器之间的交互如图 1 所示.

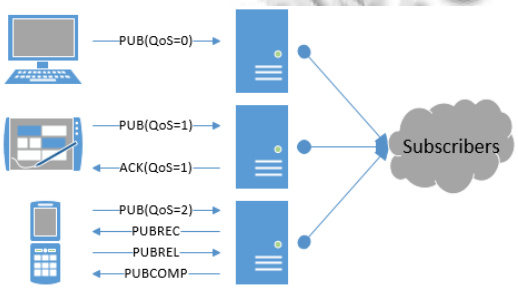

图 1　MQTT 客户端与服务器交互流程图

具体到即时通信的实际应用场景中, 客户端与服务器的交互流程如图 2 所示.

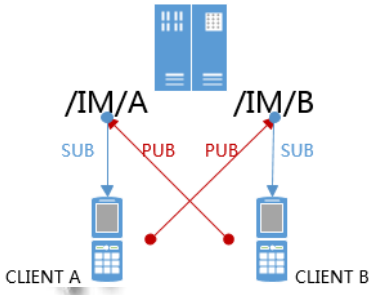

图 2　IM 交互流程简图

图中客户端 A 和 B 会分别订阅 IM 主题/IM/A 和 /IM/B. A 和 B 向对方的主题上发布消息再通过服务器转发. 由于 MQTT 消息不携带路由信息和消息内容类型. 这就需要对载荷域中携带的消息格式根据业务需求进行明确规范, 使客户端能够从中获取消息类别, 发送者信息以及载荷域的内容格式等信息.

由此可见, 使用 MQTT 协议实现即时通信应用的关键点在于话题的设计, 载荷域的编解码, 以及消息的发布和订阅流程. 下部分将对上述问题进行详细的分析, 并对即时通信中的两个核心功能: Presence 和 IM[8]进行设计和实现.

3　核心功能实现

Presence(出席状态管理)和 IM(实时聊天)作为即时通信应用中的两大核心功能, 通常会和通信录功能相结合. 前者在于对用户的实时状态进行呈现. 最常见的用户状态为在线和离线两种, 还可以包括忙碌、离开等多种自定义用户状态. 后者主要在于实现两个或多个用户之间的实时消息通信, 消息中的主要媒体为文本、图片、音视频等. 本部分将从客户端和服务器两方面对这两个功能进行分析与实现.

3.1 话题格式设计

Presence 话题: /env/pres/client_uid

IM 话题: /env/im/client_uid

MQTT 中每个话题至少需要一个字符的长度并且是大小写敏感的. ”/”用来区分话题的层级. 通常情况下, 以”/”开头的话题为普通话题, 以”$”开头的话题为系统话题. env 为用户所在的组织名称, 可以为企业或学校的标识名. 中间部分用来表示话题类别, pres 表示 Presence 话题, im 表示 IM 话题. 可见在即时通信中使用话题订阅这种方式十分便于业务的扩展. 最后的 client_uid 代表客户端的唯一标示符.

3.2 Presence 功能实现

客户端获取其他用户 Presence 状态的思路与 IM 实现思想类似, 具体实现时有两种方法可以采用:

①所有的客户端只需订阅一个属于自己的 Presence 话题, 某个客户端的状态发生改变时需要向其通讯录中所有联系人的 Presence 话题发送消息.

②客户端会在登陆成功后订阅自己通信录中每个用户的 Presence 话题, 这样当某个用户状态发生改变时, 只需向其自身的 Presence 话题上发布消息即可.

然而无论是订阅大量状态话题或是向大量状态话题发送消息, 这期间产生的流量耗费是十分可观的. 为了解决该问题, 采用了一种预先订阅的方法[9].

3.3.1 预先订阅

预先订阅的基本思想是在服务器侧构造一个预订阅客户端, 它与服务器同时启动并订阅话题 /connect/new. 当服务器检测到某个客户端是初次连接

时，就会向/connect/new 上发布消息，其中包含 client_uid. 预订阅客户端进而会“伪装”成 client_uid 代表的真正客户端发起一系列订阅请求. 这里的重点有两个，第一是“预订阅客户端”如何获得需要预先订阅的话题. 第二是其如何伪装成为真正的客户端向 MQTT 服务器发送订阅请求.

本文提出的一种处理方式为，通信录服务器提供一个预先订阅的 WebService 接口，预订阅客户端会向该接口发送包含 client_uid 的 POST 请求，通信录服务器会查询数据库中对应的联系人，按话题格式组装后，同其他需要订阅的话题一起封装成一个消息通过消息总线传回给预订阅客户端，消息格式如下:

Client_uid-<sub>-QoS-<topic1>[,topic2]

预订阅客户端在收到该消息后会把需要预先订阅的话题及其 QoS 值解析出来，并封装到 MQTT 订阅请求 Payload 中. 自定义封装格式如表 3 所示.

表 3 预订阅请求 Payload 格式

Bit	7	6	5	4	3	2	1	0
Byte 1	Client uid							
Byte 2	(00000100)Presub Flag							
Byte 3…	Topic List							

在 MQTT 服务器接收到订阅消息时，会检查 Payload 中的第二个字节是否为 Presub Flag，如果是，则会以首字节中的 client_uid 作为发起订阅的客户端，为其订阅从第三个字节开始的话题列表. 预先订阅的优点在于保证了客户端实现的简洁性，同时消除了订阅/发布大量话题造成的带宽耗费.

3.3.2 Presence 消息 Payload 格式

用户状态消息 Payload 格式如表 4 所示.

表 4 Presence 消息 Payload 格式

Bit	7	6	5	4	3	2	1	0
Byte1	Type		Flag		Status			
Byte2	Context(Optional)							

Type 字段定义 00 为保留位，01 表示基本状态，10 表示详细状态. 基本状态即为在线，离线，忙碌等. 详细状态可在可选的 Context 中携带一些状态信息. Status 字段定义实际的状态类别，0 为保留位，数字 1-4 顺次表示在线、离线、忙碌和离开四种状态，15 表示自定义状态，剩余留待扩展. Flag 字段定义 01 表示非自定义状态，10 表示为自定义状态.

综上所述，大体流程如下，首先是话题的预订阅，流程如图 3 所示.

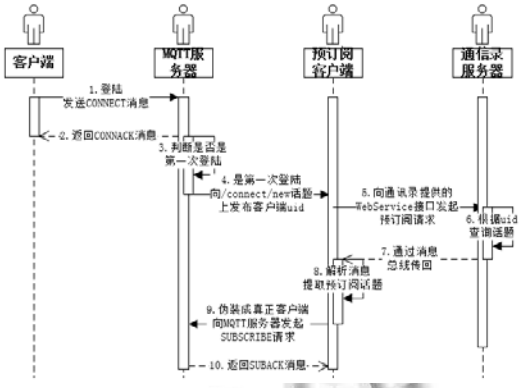

图 3 话题预订阅序列图

之后是用户基本状态的发布，不失一般性，图 4 中客户端 B 为客户端 A 的通信录中任意一个联系人. 以客户端 A 从离线变为在线状态为例:

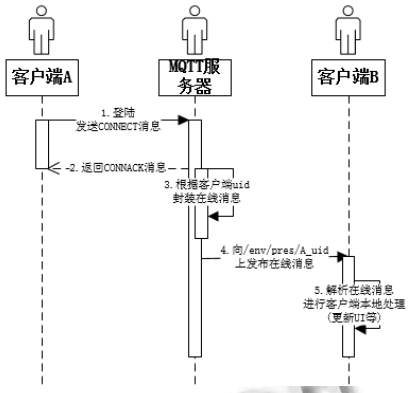

图 4 在线状态更新序列图

3.4 IM 功能实现

IM 的实现思想在之前已经进行了大致阐述，每个客户端都有一个属于自己的 IM 话题，IM 话题的订阅可以在预先订阅步骤里顺带完成. 为了提供离线消息支持，IM 话题的订阅和发布需要将 QoS 值设置为 1，保证客户端能收到离线状态下发来的消息.

IM 消息的 Payload 格式如表 5 所示.

表 5 IM 消息 Payload 格式

Bit	7	6	5	4	3	2	1	0
Byte1-4	Time Stamp							
Byte 5	IM Type				IM Count			
Byte 6…	IM Content							
Byte n	Sender Client UID							

其中前四个字节代表 IM 消息的创建时间，第五个字节包含 IM 消息的类型和数量，消息最多支持 16 种类型，具有很强的扩展性. IM Type 字段为 1 时表示

点对点 IM. 需要扩展业务需求, 添加如群组聊天等功能时可以对消息类型做相应的扩展.

IM 消息的具体内容编码格式如表 6 所示.

表 6　IM 消息内容格式

Bit	7	6	5	4	3	2	1	0
Byte 1	Content Type							
Byte 2-5	Content Length							
Byte 6…	Content							

第一个字节为 IM 消息内容的类型, 0 为文本类型, 1 为图片类型, 2 和 3 分别代表音频和视频类型, 4 代表文本文件类型. 随后是消息长度, 最后是消息的具体内容. 由于非文本内容的媒体类型可能还包含有其他信息, 比如音频编码类型, 文件长度等. 为了方便解析, 将消息的具体内容使用 JSON 格式进行包装, 转化为二进制存入 Content 之中[10].

综上所述, 客户端发送或接收 IM 消息的处理流程如图 5 所示.

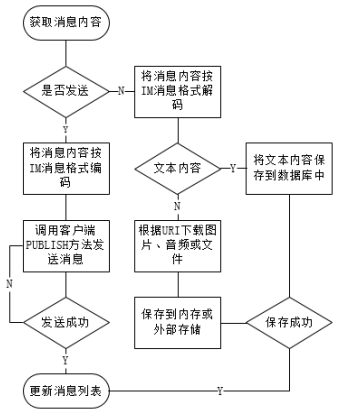

图 5　客户端对 IM 消息处理流程

整体交互流程如下, 仍不失一般性, 图 6 中客户端 B 为客户端 A 的通信录中任意一个联系人:

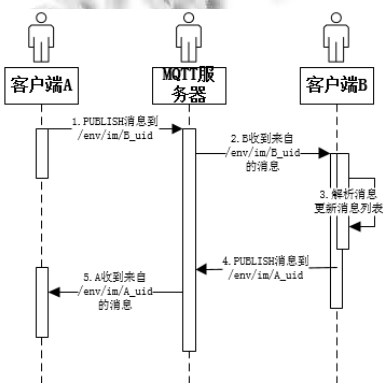

图 6　IM 序列图

4　实验验证

为了对前文提出的解决方案进行验证, 对实验所需要的环境进行了搭建,如表 7 所示.

表 7　服务器配置表

CPU	Intel(R) Pentium(R) 4 3.00GHz
内存	2GB
硬盘	500GB
操作系统	CentOS release 6.0
网卡	100Mbps

MQTT 服务器端采用了使用 C 语言编写的 Mosquitto. 客户端已经有许多健壮可用的实现, 能够在 8bit 位处理器上运行良好的 C/Java 的库分别只有 30/100KB, 由此也可以看出 MQTT 是一款十分轻量级的协议. 基于之前提出的思想, 笔者基于 EclipsePaho 项目提供的 MQTT 客户端 API 以及本实验室基于 LDAP 协议实现的企业通信录服务器, 对 Android 以及 IOS 平台的移动客户端进行了设计与实现, Presence 以及 IM 界面分别如图 7 和图 8 所示.

图 7　用户 Presence 状态管理界面

图 8　用户 IM 界面

用户登录成功后会设置包括用户名, 密码和保活间隔在内的连接参数, 之后调用 API 中 MqttClient 的

connect 函数连接服务器. 为了方便开发, 将 IM 具体内容封装成为 IMContent 自定义类, 发送之前需要对其使用 JSON 格式进行封装, 具体格式如表 8 所示.

表 8 IM 具体内容 JSON 格式

文本类型	{“co” : IMContent.getText()} 文本内容
音频类型	{“co” : IMContent.getLink(), 音频链接 “si” : IMContent.getLength()} 音频长度
图片类型	{“co” : IMContent.getLink(), 图片链接 “th” : IMContent.getThumUrl(), 缩略图链接 “or” : IMContent.getOriUrl()} 原图链接
文件类型	{“co” : IMContent.getLink(), 文件链接 “fn” : IMContent.getContentName(), 文件名称 “si” : IMContent.getContentSize()} 文件长度

之后按照表 5 给出的格式对消息 Payload 进行封装, 转化为二进制后作为参数来初始化 API 中的 MqttMessage 对象, 调用 MqttMessage 的 setQos 函数将消息的 Qos 设置为 1. 最后按照上文给出的话题格式构造消息接收方的话题, 作为参数传入 MqttClient 中的 getTopic 函数获得 MqttTopic 对象, 通过调用其 publish 函数完成对 IM 消息的发送. 对于 Presence 消息的发布与 IM 类似, 不再赘述.

介于 MQTT 协议是基于话题订阅树的形式来实现即时通信功能, 使得其能够很好的在逻辑上契合类似于企业内部的这种层级感较强的通讯录, 降低了开发的复杂度. 同时, 使用 JSON 格式对消息内容进行封装也有效的增强了 MQTT 协议的消息扩展性.

此外, 为了能够和 SIMPLE、XMPP 协议在移动即时通信上的性能表现进行对比, 也对这两种协议的测试环境进行了搭建. 实验分别对三种协议的 Presence 和 IM 两个项目的开销进行了对比测试. 主要统计指标为平均每秒发送的 packet 数以及字节数.

5.1 IM 测试

IM 测试结果如表 9 所示.

表 9 IM 测试结果

Protocol	Packets	Bytes	Packets/s	Bytes/s	Duration(s)
SIMPLE	424	216497	0.651	332.35	651.399
XMPP	426	63662	0.647	96.689	658.418
MQTT	428	50664	0.335	35.947	654.516

对于 IM 的测试, 采用的方法是在相同网络环境下发送相同的文本内容. 从上表可以看出 MQTT 协议发送的总字节数只有 SIMPLE 协议的四分之一不到, 也比 XMPP 协议少了大约 20%左右的流量.

5.2 Presence 测试

Presence 测试结果如表 10 所示.

表 10 Presence 测试结果

Protocol	Packets	Bytes	Packets/s	Bytes/s	Duration/s
SIMPLE	403	141346	0.346	121.323	1165.038
XMPP	261	44721	0.144	24.625	1816.089
MQTT	415	36337	0.334	29.279	1241.034

在 Presence 的测试中, SIMPLE 和 MQTT 每分钟进行一次状态更新, XMPP 每两分钟进行一次状态更新, 所以 XMPP 的测试结果要翻倍来看. 由于 MQTT 的状态更新包采用二进制方式封装, 内容编码紧凑, 发送的总字节数和平均字节数都明显的少于 SIMPLE 和 XMPP.

从上面三组测试可以看出, MQTT 协议在待机效率, IM 效率和用户状态更新效率方面远远要优于 SIMPLE 协议, 对比 XMPP 协议也有显著的提升, 大大减少了带宽的耗费和移动设备的电量消耗.

5 结语

移动互联网即时通信应用已经成为人们日常沟通的重要工具, 然而当前已有的即时通信协议都不能够很好的满足移动互联网网络环境不稳定, 流量代价高, 设备处理能力有限等特点. MQTT 协议作为一种基于发布/订阅模型的轻量级消息传输协议, 在移动互联网中具有带宽利用率高, 耗电量较少等特点. 本文首先阐释了 MQTT 协议存在的优势, 之后分析了 MQTT 协议的消息格式和交互流程, 并在此基础上对即时通信的两个核心功能, Presence 与 IM 进行了设计和实现. 使用预先订阅的方法进行了优化, 大幅减少了带宽消耗. 最后对 MQTT 协议与 XMPP、SIMPLE 协议在 IM 和 Presence 更新两方面的表现进行了对比测试, 结果表明使用 MQTT 协议确实能够更有效的节约网络成本. 然而使用 MQTT 作为移动互联网即时通信协议, 缺点在于消息可读性变差, 需要客户端和服务器预先设计并协商好消息的封装格式, 开发难度较大.

参考文献

1 Day M, Rosenberg J, Sugano H. A model for presence and instant messaging. IETF, 2000, 2: RFC2778.

2 Day M, Aggarwal S, Mohr G, et al. Instant messaging/

presence protocol requirements. IETF, 2000, 2: RFC2779.

3 Protocol S. SIP: Session Initiation Protocol. Retrieved May, 2003, (2): 382–388.

4 Extensible Messaging and Presence Protocol http://xmpp.org/

5 Saint-Andre P, et al. Extensible messaging and presence protocol (XMPP): Core. University of Helsinki Department of Computer Science, 2004.

6 Tang K, Wang Y, Liu H, et al. Design and implementation of push notification system based on the MQTT protocol. Proc. International Conference on Informationence& Computer Applications. 2013, 92. 116–119.

7 Banks A, Gupta R. OASIS Standard MQTT Version 3.1.1. http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/mqtt-v3.1.1.html. [2014-10-29].

8 杨海波,王默涵,贾正锋,卜立平.面向移动互联网的 Presence/IM 机制研究.小型微型计算机系统,2015,36.

9 任亨,马跃,杨海波,贾正锋.基于 MQTT 协议的消息推送服务器.计算机系统应用,2014,23(3):78–79.

10 关庆余,李鸿彬,于波.MQTT 协议在 Android 平台上的研究和应用.计算机系统应用,2014,23(4):199–200.