**有关用户在线时长的统计**

|  |  |  |  |
| --- | --- | --- | --- |
| **版本** | **时间** | **作者** | **备注** |
| **V1.0** | **2024.6.25** | **刘亚鹏** |  |

1. **目前方案：**

1.通过MQTT组件

**原理：**后端服务器通过客户端与MQTT连接建立、断开连接操作触发的回调请求来知道用户登录与退出；连接建立更新用户登录时间、连接断开更新用户下线时间、并以这两项数据计算用户的在线时间。

**统计结果（办公室手机测试）：**

IOS将应用推到后台-间隔大约2分钟的时间断开MQTT连接

杀掉应用-直接断开MQTT连接

开启飞行模式-间隔大约2分钟的时间断开MQTT连接

Android将应用推到后台-间隔大约30秒的时间断开MQTT连接

杀掉应用-直接断开MQTT连接  
 开启飞行模式-间隔大约2分钟的时间断开MQTT连接

也就是说只要不是杀掉APP的退出，这两个平台的用户在线时长实际统计都是不准确，多统计出的时间是关联了MQTT的连接断开时间。

**问题：**

直观感受就是统计结果不统一，统计结果受不同设备的影响。

并且，线上用户存在在线时间过长的问题，也就是MQTT不断开的情况

**思路：**

统计结果受前端差异化影响越小越好

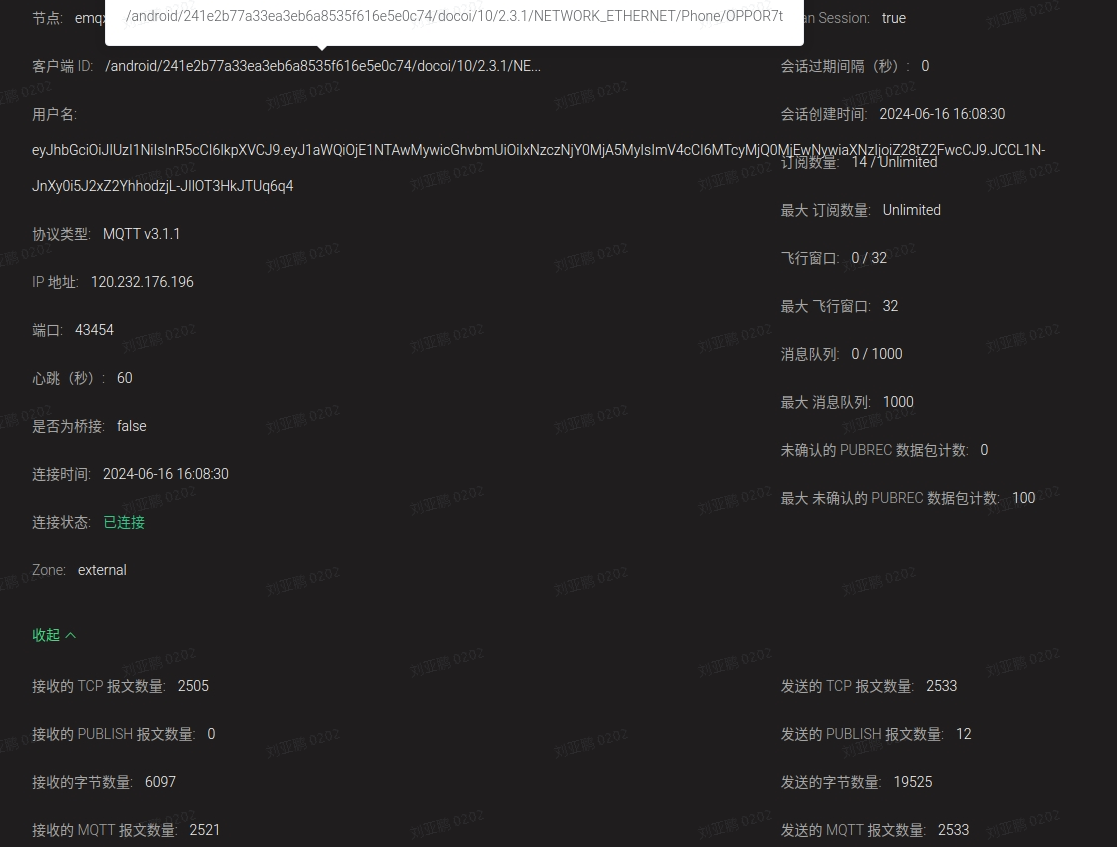
**二、在线时间过长问题排查：**

1. 用户打开梦游社APP后，由于种种原因，将梦游社APP推到后台。如下为排查到的单个用户的埋点上传数据图

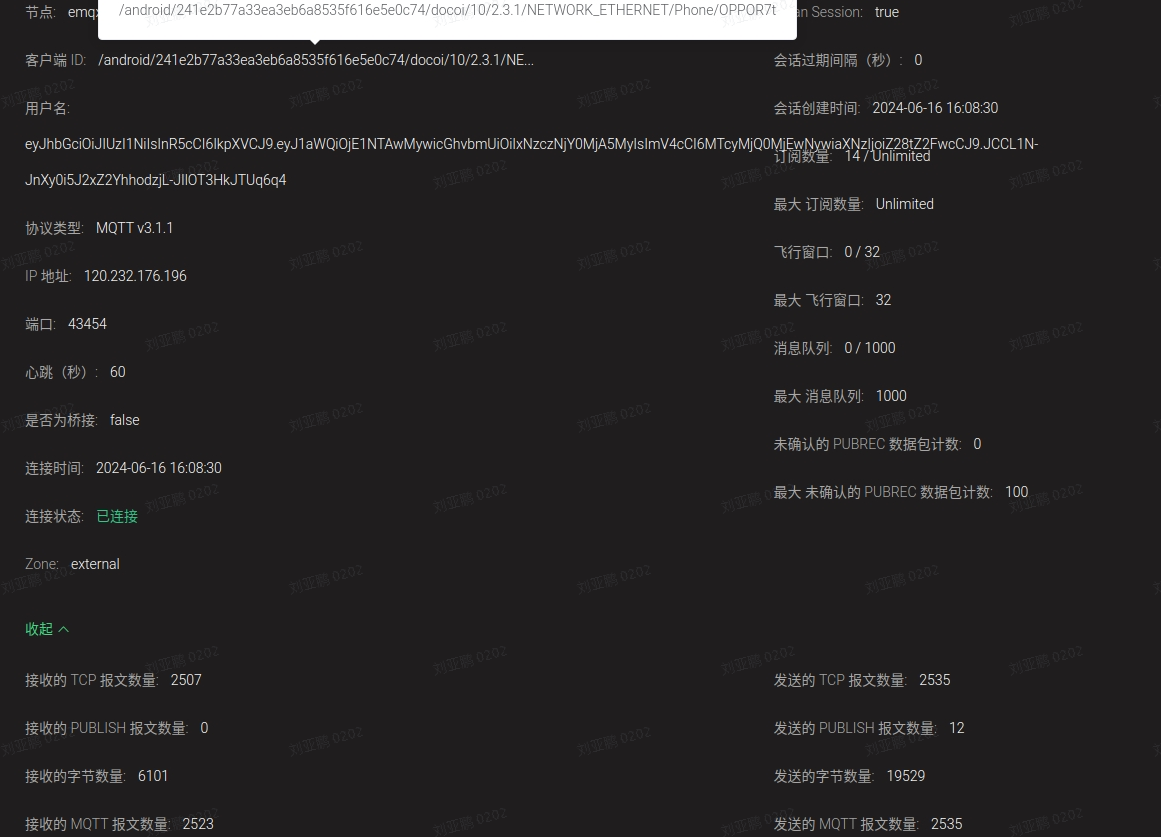


1.1 数据显示用户将APP推到后台时间为6.17号凌晨00:35

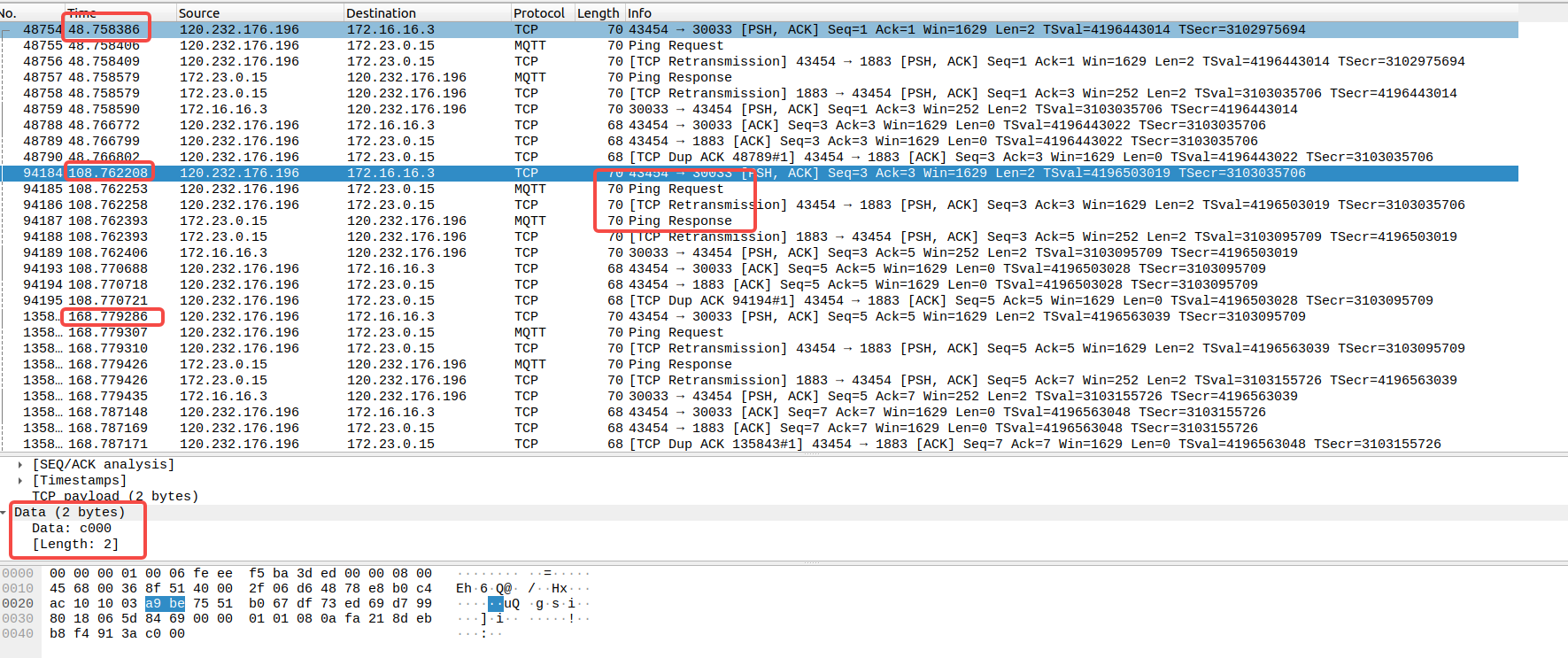
1. 前端还是会保持与MQTT的连接（目前看大部分设备是android未断开）。如下为排查到的MQTT后台网页的实际TCP连接。



2.1 第一次截图 接收到的TCP包文数量为2505 发送的TCP报文数量为2533

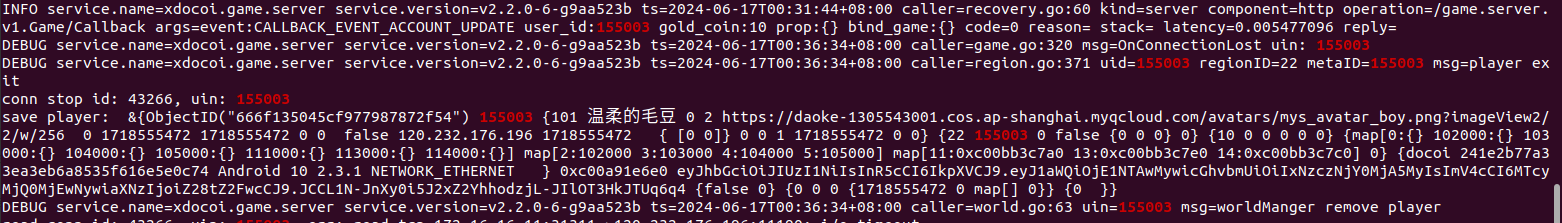


2.2 第二次截图 接收到的TCP包文数量为2507 发送的TCP报文数量为2535



2.3抓包使用WireShark分析实际的TCP传输数据，确实为心跳数据

1. 查看APP与游戏服TCP长连接断开时间，如下为游戏服日志信息图



3.1数据显示APP与游戏服连接超时断开时间为为6.17号凌晨00:36，与用户将APP推到后台时间相差1分钟，这个时间差是建立TCP连接时设置的超时时间

3.2 这个时间点是和用户将APP推到后台时间相仿的

3.3 也就是说连接游戏服TCP链接断开了，但是连接MQTT的TCP链接未断开。原因在第五项的具体方案有说明。

**三、统计用户在线时长难点：**

前后端短链接场景，服务器端无法准确判断用户的退出时间

**四、明确要点：**

将APP推到后台算不算退出？

为了应对频繁切后台场景，不能处理为立即断开MQTT连接。

1. **解决方案：**

**方案一：（不改变现有的计算方式，解决现有问题。）**

**说明：**APP被推到后台。如果超过1分钟没有将应用切回前台，则手动断开MQTT连接；超过1分钟后将应用切回前台，则需要重新建立MQTT连接。

**优点：**不改变现有计算方式

**缺点：**旧版本客户端不能兼顾到改动。前端需要做MQTT的超时断开和重新切回前台的重连处理；

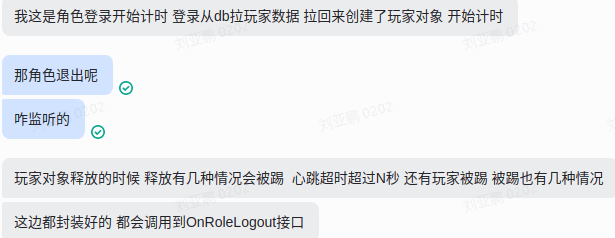
**统计结果差异：**将应用推到后台导致退出的用户的在线时长统计相差1分钟，网络不好的用户导致超时退出在线时长统计相差2分钟，整体数据偏差需要统计这部分人数的占比

**方案二：（组件差异？使用其它第三方组件回调？如：IM等。）**



2.1 在APP被切到后台的情况下，不管什么第三方组件，只要它们的连接被保活，就有可能出现一直在线的问题。

**方案三：（用自己的长链接统计。）**



**说明：**后端调整通过游戏服的长链接来统计用户登录与退出登录；询问X3项目，也是长链接方式，在用户登录接口中开始计时，在用户退出接口完成计时。

**优点：**可控的链接，设置心跳超时时间，监听到用户切后台则不再发送心跳

**缺点：**由于业务调整，后续需求是否在登录APP的时候就开始建立游戏服的链接？

**统计结果差异：**可判断链接断开原因，统计的结果是比较接近用户真实的在线时长。

**方案四：（客户端本地记录。）**

**说明：**通过客户端的埋点上传

**优点：**现有功能，客户端能准确反应用户的真实状态

**缺点：**不能即时上报用户退出时间，要等到客户端下一次上线才能上报完成

**统计结果差异：**统计结果准确反应用户真实的在线时长

**方案五：（活跃时间）**

**说明：**后端通过用户请求接口来统计用户在线时长，没有接口请求表示用户下线，这种实现起来较为复杂

**方案六：（前后端定时请求）**

**说明：**类似心跳数据的定时请求，如果没有请求则表示用户下线，这种会增加网络传输、带宽占用，不是很合理

1. **总结分析**

**总结：**

**对于长链接场景：后端所监听的客户端都只有两种状态：在线和不在线。明确用户在线的定义：用户真实打开应用在使用。而那些将应用切到后台或者由于网络等问题导致链接断开，状态就是不在线。后端进一步明确链接断开原因，就可以进行相应的处理，从而计算更精准的客户端在线时长；**

**对于短链接场景：没有太多的参考资料，没有更多能适配场景的方案。**

**现在的用户时长统计数据报表排除一个小时以上的数据其实是比较符合我们APP的使用场景的，毕竟不涉及游戏。。。但是为了数据准确还是需要解决问题。**

**结论：**

**综上几种方案，个人觉得方案三是目前最合适的。但还是以业务需求调整为主。**

**第一点是开发量：只需要后端修改，且开发时间适中；**

**第二点是兼容性：所有版本客户端都能被统计，并且不需要区分客户端平台；**

**第三点是时间准确性：统计的结果是最接近用户在线时长的；**

**其次方案一。**

**方案统计：**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **开发成本** | **版本兼容性** | **统计结果准确性** | **推荐值** | **备注** |
| **方案一** | **适中** | **最新版本** | **1-2分钟误差** | **5** |  |
| **方案二** | **较高** | **所有版本** | **误差<1分钟** | **3** |  |
| **方案三** | **低** | **所以版本** | **误差<1分钟** | **6** |  |
| **方案四** | **无** | **所有版本** | **误差<1分钟，但是数据上报有延迟** | **6** |  |
| **方案五** | **高** | **所有版本** | **误差<设置的活动时间** | **3** |  |
| **方案六** | **适中** | **最新版本** | **误差<设置的定时时间** | **4** |  |

1. **引申问题**

**目前统计用户实时在线人数的上报同样是由游戏服的长链接数量统计，如果之后业务或需求调整，这部分功能也需要重新评估。**

1. **讨论结果**

**前端不推荐使用方案一，断开重新建立连接流程，需要重新订阅，流程调整多，可能导致消息丢失。  
 使用方案四，使用数数上报中客户端原有的统计用户在线时长事件，统计保证准确，主要关注大部分的用户的数据上报。**