**利用Fiddler模拟恶劣网络环境**

2015年06月29日 15:51:06

<https://blog.csdn.net/baidu_zhongce/article/details/46683323>

在解决日常的支持需求中，经常会遇到一些用户反馈一些无法简单复现的bug，有很大一部分的bug是由于用户自身的网络环境波动，或者是本身网络环境就较为恶劣，而服务在面对这种恶劣的网络环境的健壮性不够，导致会出现一些意想不到的bug。而在正常的开发自测过程中很难去营造出这种恶劣的网络环境，使得这些bug较难被提前发现和修复。另外一些服务在恶劣网络环境下虽然不会出现不可用的情况，但是用户体检很差，为了优化这个情况下的用户体验，也需要去在本地模拟这种环境来进行调优。

所以要去复现这些bug，甚至是去提前发现这些bug，就需要能够在开发环境中模拟出恶劣的网络环境，从而看到在这种恶劣的网络环境下的服务的表现等。当前模拟恶劣网络环境主要可以通过以下这些手段实现：

1. 通过应用层或者传输层的代理服务器，通过在代理服务器上设置一些模拟恶劣网络环境的参数，使得通过这些代理服务器的流量都被转化为恶劣网络环境下的流量。如利用Fiddler，Charles等具有代理服务器功能的网络流量分析软件来实现。
2. 通过利用一些更底层的驱动层面的服务，通过控制网卡的收包发包的行为，来模拟恶劣的网络环境。如dummynet的ipfw驱动等。
3. 通过建立一个可控的网关，在网关上部署模拟恶劣环境的相关程序，所有需要借助该网关进行转发的流量都会被模拟为恶劣网络条件。Linux下的netem就提供了这类支持。

这里主要先讲的是第一种手段，即利用Fiddler来模拟恶劣的网络环境，对服务进行测试，这个手段实现简单，较为直观，但是缺点是只能支持那些利用HTTP进行通信和交互的服务。在之后的文章中也会进一步说一下后两种手段。

**【Fiddler是啥】**

Fiddler的官网上是这样描述它自己的：The free web debugging proxy for any browser, system or platform，即跨浏览器、跨系统、跨平台的免费Web Debug代理服务器。当你的HTTP浏览经过Fiddler时，Fiddler可以监视流量，查看HTTP通讯的各种信息，设置断点查看和修改HTTP数据，甚至可以构造各种测试用的HTTP包以及重放已记录的包等。其官网是<http://www.fiddler2.com/fiddler2/>，上面详细地介绍了Fiddler到底是什么。

**【简单地利用Fiddler限速模拟恶劣网络环境】**

Fiddler本身已经预置提供了模拟Modem速度的选项，其位置位于：

Rules – Performances – Simulate Modem Speeds

勾选该选项后，所有通过Fiddler代理的流量都会变得和多年前的56k小猫时上网一般的慢。

由于Fiddler只是一个HTTP代理，要直观地看出限速效果，最好是运行在浏览器中的测速工具，这里选用speedtest.net提供的测速工具进行测试。

首先是开启该选项之前的速度：

打开了Simulate Modem Speeds后：

速度已经回到了当年那种无法忍受的低速了，注意到这里PING值也有了显著的提高，而事实上ping值是ICMP层的控制报文，并不会被Fiddler影响，理论上ping值并不会出现提高的情况，进一步分析Fiddler中的报文则可以看出端倪：

事实上网页插件并不能实现发送ICMP包并得到ping值的功能，而是用多次较小的HTTP GET请求的响应时间来计算PING值，这里实际算出来的是一个平均的HTTP的RTT值，所以受到Fiddler模拟恶劣环境的影响就是正常的了。

**【调整模拟恶劣网络环境的参数】**

直接模拟Modem速度实在是慢爆了，事实上就算是在很差信号的情况下，手机移动网络的速度都已经超过了当年的56k Modem速度了，所以采用默认的配置模拟出来的环境过于恶劣，并不一定符合需求，此时就需要对限速的参数进行调整。

Fiddler本身就提供了一个配置文件供调整这些参数，点击：

Rules – Customize Rules…

就会用文本编辑器打开CustomRules.js文件，其默认位于用户目录的文档目录下的\Fiddler2\Scripts 位置，后缀名是js，其内容实质是JScript.NET——微软对ECMAScript规范的实现，与日常使用的javascript是属于同一个规范下的，但是在扩展的细节实现存在一定的不同。

打开该文件后，可以找到一个m\_SimulateModem标志位：

if (m\_SimulateModem) {  
 // Delay sends by 300ms per KB uploaded.  
 oSession["request-trickle-delay"] = "300";  
 // Delay receives by 150ms per KB downloaded.  
 oSession["response-trickle-delay"] = "150"  
}

* 1
* 2
* 3
* 4
* 5
* 6

该标志位控制着oSession的两个参数值的设置，当勾选了Simulate Modem Speeds时，request-trickle-delay与response-trickle-delay就会被设置，其中request-trickle-delay中的值代表每KB的数据被上传时会被延时多少毫秒，response-trickle-delay则对应下载时每KB的数据会被延时多少毫秒，如果本身网速已经相当快的话，这里设置的值就可以近似地推算出开启模拟后的上传和下载带宽了，比如默认设置下下载延时为150ms，上传延时为300ms，对应可以推算出大致的模拟带宽为：

上传带宽=(1\*8/1000)/0.300≈0.053Mbps

下载带宽=(1\*8/1000)/0.150≈0.027Mbps

然而实际情况下却得到了两倍于这个值的带宽，推测可能是Fiddler的内部实现上有一些和描述上的不同，为何为造成这个现象现在还不是很清楚，所以上述公式最后还需要修正一个2.0的系数，即：

上传带宽=((1\*8/1000)/0.300)\*2.0≈0.106Mbps

下载带宽=((1\*8/1000)/0.150)\*2.0≈0.053Mbps

假设我们将两个参数都设置为50，则会得到上下载带宽均为0.32Mbps，测速结果如下所示：

让我们来分析一下这几行代码：

* 首先来判断m\_SimulateModem是否为true，也就是是否设置了弱网模式。
* 如果为弱网模式。则分析代码
* oSession[“request-trickle-delay”] = “300”; 注释的也很明白，Delay sends by 300ms per KB uploaded.上传1KB需要300ms，转化一下上传速度：1Kb/0.3s = 10/3(KB/s)
* 如果你想设置上传的速度为50KB/s，你则需要设置Delay 时间为 20ms
* 同样的方法，也可以限制上传的速度，调整oSession[“response-trickle-delay”]即可。

*来自 <*[*https://www.cnblogs.com/pachongshangdexuebi/p/6601332.html*](https://www.cnblogs.com/pachongshangdexuebi/p/6601332.html)*>*

**【编写自定义脚本】**

进一步地，我们可以扩展CustomRules.js里的逻辑，参照Jscript的文档可以在模拟恶劣环境中加入更多自定义的逻辑，这里实现了一个随机延时量设置，使得网络带宽不是恒定为一个低速的值，而是会在一定范围内随机抖动：

static function randInt(min, max) {  
 return Math.round(Math.random()\*(max-min)+min);  
}  
if (m\_SimulateModem) {  
 // Delay sends by 300ms per KB uploaded.  
 oSession["request-trickle-delay"] = ""+randInt(1,50);  
 // Delay receives by 150ms per KB downloaded.  
 oSession["response-trickle-delay"] = ""+randInt(1,50);  
}

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9

得到的测试结果如下：

在测速过程中的瞬时速度的趋势图如下：

可以看到整体的网络限速存在了一定程度的抖动。

通过进一步扩展CustionRules.js可以实现很多需要的恶劣环境模拟场景，如果场景较为复杂的话，也可以通过编写Fiddler的插件的方式，编写C#插件代码来进一步控制Fiddler的行为，在这里就不多做赘述了。详细可以参照：<http://docs.telerik.com/fiddler/extend-fiddler/extendwithdotnet>

**【Fiddler模拟恶劣网络环境的局限性】**

Fiddler进行限速较为简单和灵活，配置也较为方便，但是由于它是一个应用层的HTTP的代理，只能模拟该层上的行为，对于一些复杂的网络层的丢包、重传等恶劣情况就不能很好的模拟出来，而且对于其他协议的应用也不支持，后续会介绍一些其他的模拟恶劣环境的方法和软件来弥补这些缺失。