# 多机弱网络环境模拟

项目地址： <https://gitlab.gz.cvte.cn/Geek/netem>

核心技术简述： 本文通过让被测程序的server和client运行在docker容器中，来实现1对60路程序在弱网络环境下的性能测试，主要使用以下技术：使用docker的bridge网络驱动创建虚拟网络接口，使用Linux Traffic Control(tc)中的htb/tbf和netem等qdisc来对docker的网络环境进行配置，使用ncat+sh-redirect来进程多机同步控制，使用docker-compose来进行client数量的动态伸缩管理。由于内容太多，因此本文先介绍大体的实现方法，后续会详细介绍其中的主要技术。

## 一、背景介绍

有一天产品经理找上门问我，在WIFI局域网中将1M的文件发送到60台设备上需要多久？作为一个严谨的程序员，立马发现这个问题有隐藏变量，那就是网络环境，显然在不同的网络环境下发送文件需要的时间是不同的，因此我们需要控制网络环境的相关指标然后才能测量出发送所需的时长。

因此我们将这个问题分解为三步：

1. 搭建受控的网络环境；
2. 部署60个终端的测试程序；
3. 收集并统计数据；

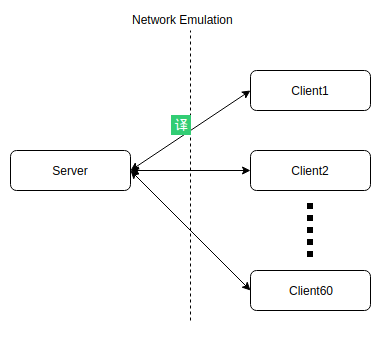
第1个问题目前看来还不是那么棘手，因为现在有很多模拟网络环境的设备和工具，比如网络损伤仪，Network Emulator for Windows Toolkit，Fiddler，tc等。

第2和第3个问题看起来是人力成本，无非就是辛苦一些，但实际上它是目前最棘手的问题，因为它非常麻烦，会大大降低工作的效率和积极性，比如我今天优化了一下代码，我就需要重新部署测试程序，这导致我不能及时了解到改动的效果，虽然我非常想知道优化的效果，但是由于部署麻烦现实中经常会导致我不愿意去测试。所以我们要重点解决这个问题。

总结一下，我们渴望一个这样的模拟环境：

1. 可以方便的对网络参数进行设置；
2. 可以方便的部署多机测试环境；
3. 可以方便的收集结果数据；
4. 可以方便的进行对比测试（比如和其他同类程序进行对比）；
5. 可以方便的为不同客户端模拟不同的网络环境；
6. 可以方便的迁移到其他项目中，用于测试其他项目；
7. 可以在自己的工位上完成整个测试（不需要去搬运机器，配置设备等）；

显然，这样的环境肯定是要用软件模拟的，用图形表示大概是下面这样：



在现实中Server和每一个Client都运行在不同的设备上，如果要把这个环境移到一台电脑上最直接的方式就是将Server和Client全部放到虚拟机中，但是虚拟机比较重，在一台电脑上同时运行超过60个虚拟机显然不是一个明智的选择，所以就只剩下2种方法了：

1. Server和Clients都作为一个进程运行，对这些进程进行网络限制； 这种方案有几种实现方法，比如通过给应用程序设置网络代理，或者通过注入的方式修改进程所使用socket API接口地址，或者在linux上通过netfilter/queuing-discipline框架/在windows上通过WFP(Windows Filtering Platform)框架开发专门的过滤程序来实现。虽然这些方法都可以实现，但都有一定的缺陷。比如在linux上通过so劫持方式实现给进程网络限速的trickle程序，它不支持fork，也不支持静态链接的程序；比如linux上非常强大的tc（后文重点讲解）程序，它只能针对网卡进行限速，不能针对进程进行限速。
2. Server和Clients都运行在独立的容器中，对每个容器进行网络限制； 这种方案可以通过创建容器环境来隔离各个端，看起来是一种理想的方式，但是对docker不熟悉的同学可能会怀疑能在一台机器上同时运行超过60个container么？答案是肯定的，由于docker采用linux内核的namespace机制实现，理论上来讲在docker中运行一个进程并不会比在容器外运行一个进程耗费更多的资源，只要这个进程可以在主机上同时运行超过60个实例那么也就（几乎）肯定可以运行在60个容器中。 只要进程运行在了容器中，我们就可以通过限制容器的网络环境来控制程序运行的网络环境了，这个时候tc就派上用场了。

经过分析，我们决定采用方法2。

由于目前项目中的主要网络SDK都是跨平台的，而linux对我们的测试场景更加友好，因此我们只在linux上搭建测试环境。

## 二、具体实现

为了让镜像足够精致，不会引入太多没必要的内容，我们决定选用docker官方的alpine作为base image，虽然网上有很多负面评价，但因为其精致我愿意给它一次机会。制作镜像的Dockerfile如下：

这个文件放在image文件夹里，后续会在docker-compose文件中使用。 需要注意的是以上镜像中并没有对网络环境进行设置，这是因为网络环境是运行时参数，可能需要经常调整，因此就没有将这些配置写死在镜像中。

下面是镜像启动后运行数据发送端的脚本server.sh，这个脚本会挂载到镜像中的/data目录下。

下面是镜像启动后运行数据接收端的脚本client.sh，这个脚本会挂载到镜像中的/data目录下。

我们使用docker-compose来管理容器，docker-compose.yml文件如下：

最后就是使用以下命令启动server和client两个容器：

sudo docker-compose up --force-recreate

不出意外的话这会启动两个容器，一个是server，一个是client，并且日志会显示client已经连接到server了，接下来我们需要通过docker attach命令附加到server容器中，然后输入sh -c "$CLIENT" 来让client开始接收文件。当文件传输完毕之后会看到time命令打印的耗时信息。至此，1v1场景下的模拟环境已经搭建完毕，那怎么扩展到60个接收端呢？其实非常简单，只要给docker-compose命令添加一些额外参数即可，如下：

sudo docker-compose up --scale file-client=60 --force-recreate

通过指定scale参数，我们让file-client启动了60个实例，启动后我们会在日志中看到已经有60个client连接到了server上，同样执行sh -c "$CLIENT" 即可同时向60个实例发送文件。传输完毕之后时间信息会同样打印出来。

我们可以将自己要测试的程序挂载到data目录，然后在file-server容器中启动自己的服务端，在file-client容器中启动自己的客户端进行真实场景的测试。

## 三、成果

通过以上的模拟系统，我们对项目中使用到的文件传输功能进行了性能测试，发现自己的程序性能比我们预想的要差很多，后来经过多次的代码优化，反复的测试最终使得我们程序的性能达到了一个我们比较满意的水平。这同时也验证了这个模拟系统的有效性。

## 四、总结

以上方式模拟出来的测试环境已经可以非常方便的扩容，以及对网络参数进行方便的调整，已经基本达到了可用的程度，但是作为一个有追求的程序员，我当然不能就此打住，因为它还有一些缺陷：

1. 无法对接收的数据包进行限制，是什么原因呢？有什么办法做到么？；
2. 需要在镜像中安装tc工具，而这个工具本身并不是我们测试的payload，因此有一点多余，有没有办法不需要在镜像中安装tc工具就可以进行限速呢？

以上问题的答案是肯定的，具体的解决办法大家可以先想一想，我也会在接下来文章里分享出来。

参考链接：

[Compose file version 3 reference | Docker Documentation](https://docs.docker.com/compose/compose-file/)

[Running glibc programs - Alpine Linux](https://wiki.alpinelinux.org/wiki/Running_glibc_programs)

[The best Docker base image for your Python application (July 2019)](https://pythonspeed.com/articles/base-image-python-docker-images/)

[linux - docker networking namespace not visible in ip netns list - Stack Overflow](https://stackoverflow.com/questions/31265993/docker-networking-namespace-not-visible-in-ip-netns-list)

[Networking with standalone containers | Docker Documentation](https://docs.docker.com/network/network-tutorial-standalone/)

[Use Macvlan networks | Docker Documentation](https://docs.docker.com/v17.12/network/macvlan/#use-an-ipvlan-instead-of-macvlan)

[Traffic shaping - Wikipedia](https://en.wikipedia.org/wiki/Traffic_shaping)

[networking:netem](https://wiki.linuxfoundation.org/networking/netem)

[Linux Advanced Routing & Traffic Control HOWTO: Queueing Disciplines for Bandwidth Management](http://www.fifi.org/doc/HOWTO/en-html/Adv-Routing-HOWTO-9.html)

[tc-htb(8) - Linux manual page](http://man7.org/linux/man-pages/man8/tc-htb.8.html)

[Understanding tc qdisc and iperf - Unix & Linux Stack Exchange](https://unix.stackexchange.com/questions/38029/understanding-tc-qdisc-and-iperf)