作者：dog250-anonymous  
链接：https://zhuanlan.zhihu.com/p/362049294  
来源：知乎  
著作权归作者所有。商业转载请联系作者获得授权，非商业转载请注明出处。

UDP越来越被重视了，人们期望它和TCP一样，但是呢？...

UDP能不能像TCP那样做高并发？

内核早在3.9版本就支持REUSEPORT了，但直到2015年底，我也只能做个表面上(看起来像那么回事，但实际上就是垃圾的东西)的东西： [https://blog.csdn.net/dog250/article/details/17061277](https://link.zhihu.com/?target=https%3A//blog.csdn.net/dog250/article/details/17061277) 但也只是看起来像，除此之外，我当时的想法相当于垃圾。

UDP做高并发非常简单，和TCP的accept模型似乎没有什么区别，下面是一个随手撸的代码，主要是跟着小小学python：

#!/usr/bin/python3

import select

import socket

sd = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)

sd.bind(('192.168.56.102', 1234))

sd.setsockopt(socket.SOL\_SOCKET, socket.SO\_REUSEPORT, 1)

inputs = [sd]

while True:

readable = select.select(inputs, [], [], 1.0)[0]

for s in readable:

if s == sd: # Accept 逻辑在此if分支实现

data, addr = s.recvfrom(1024)

# 创建per-客户端socket

csd = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)

csd.setsockopt(socket.SOL\_SOCKET, socket.SO\_REUSEPORT, 1)

csd.bind(('192.168.56.102', 1234))

csd.connect(addr)

inputs.append(csd)

print('accept connection from %s:%s' % addr, 'sd[%d] created' % csd.fileno())

else: # 这个else分支处理per-客户端的socket

data, addr = s.recvfrom(1024)

if data:

print('received "%s" from %s' % (data.splitlines()[0].decode('utf-8'), addr), 'sd[%d]' % s.fileno())

s.sendto(data, addr)

是不是超级简单，比我之前长篇大论的垃圾强多了。

根本就一点， ***只要socket数量上去了，并发也就是上去了，*** 因为一个socket一个读队列一个写队列，你再分身文件描述符也没鸟用，唯一的办法就是创建多个socket，这样队列操作就可以彼此分开并行处理了。至于[accept逻辑](https://www.zhihu.com/search?q=accept%E9%80%BB%E8%BE%91&search_source=Entity&hybrid_search_source=Entity&hybrid_search_extra=%7B%22sourceType%22%3A%22article%22%2C%22sourceId%22%3A%22362049294%22%7D)，那只是让UDP看起来像TCP而已，这是次要的。

由于UDP的无连接性，在实际client发送数据前，server是不知道四元组信息的，并且也不知道UDP通信的模式，如果是oneshot，pingpong形式的数据，那谈不上什么连接跟踪的必要，但是对于类似QUIC这种 ***长连接*** 的[流式数据](https://www.zhihu.com/search?q=%E6%B5%81%E5%BC%8F%E6%95%B0%E6%8D%AE&search_source=Entity&hybrid_search_source=Entity&hybrid_search_extra=%7B%22sourceType%22%3A%22article%22%2C%22sourceId%22%3A%22362049294%22%7D)传输而言，UDP只是承载数据的一个协议而已，这个时候只要知道了四元组，当然也就可以专门创建一个socket，用connect来保持住这个 ***连接*** 了。

问题是，当有数据进入UDP处理的时候，内核协议栈如何查找到对应的socket呢？如果socket数量数以万计，查找的开销还是可观的，这里的问题有两点： - ***对于connected的UDP socket，无法使用REUSEPORT的四元组hash模数取余机制。*** REUSEPORT的四元组hash模数取余机制会导致同一个四元组随着socket数量增减而对应到不同的socket，无法和线程，epoll模型协同。详情看： [https://lore.kernel.org/patchwork/patch/1129552/](https://link.zhihu.com/?target=https%3A//lore.kernel.org/patchwork/patch/1129552/) - ***如果不能用REUSEPORT的模数取余机制，所有socket只能二元组hash来组织。*** 所有的socket由于bind同一个地址和端口，将会位于同一个hash桶中，退化成链表。

所以说呢？这里就有优化空间了，如何用四元组来组织hash表呢？看这里： [https://blog.csdn.net/dog250/article/details/104219341](https://link.zhihu.com/?target=https%3A//blog.csdn.net/dog250/article/details/104219341)

UDP在QUIC的催化作用下，近些年已经逐渐规模化了，UDP很多隐藏的实现问题逐步暴露，优化点也越来越多，事情才刚刚开始。

[杏花村](https://www.zhihu.com/search?q=%E6%9D%8F%E8%8A%B1%E6%9D%91&search_source=Entity&hybrid_search_source=Entity&hybrid_search_extra=%7B%22sourceType%22%3A%22article%22%2C%22sourceId%22%3A%22362049294%22%7D)不让买了，也就不喝了，哎...

浙江温州皮鞋湿，下雨进水不会胖。