

Forum:[Hack Forums](#)

Thread: [TCP send and forget](#)

RUMRUN - August 16, 2019, 2:29 pm

Is it possible to send and forget a TCP packet? If so, what is the fastest and easiest way?

I'm working on a project and need to contact 10000 ipv4 addresses through 7500 socks proxies every 6-10 hours ($10000 * 7500 = 75\,000\,000$ sockets/6h = 12 500 000 sockets/hr, 208 333/minute or 3472/second)

It speaks for itself that opening a socket with a proxy which is bad already takes a couple of seconds,

I am running 100 redis workers right now on a 4 core, 16 gb ram server which more or less maxes it out.

I'm not even close to scratching the surface yet (estimate is 200 proxies have been handled the last 8h)

Which would be 3% of the total amount, leading me to believe I need to scale 30x.

Preferably I'd optimize first before going big on the servers -- so back to the beginning: is it feasible?

How does zmap accomplish this and how can I accomplish it in python?

Thanks, if you want to put some time into this we can work something out, just dm me

f1re_w1re - August 16, 2019, 7:39 pm

Well sounds like you know how many packets to send out a second, but i am still unclear on how much you can actually achieve without running some baselines to get a rate. The results may vary depending on what your computer is doing and how much bandwidth is sitting on your line.

I also think you should try to use UDP instead of TCP because it is faster. That may help your cause a bit. You can also do this in Python (look around).

What exactly are you trying to do anyway?

Nomad.Null - August 17, 2019, 8:13 am

The problem with your use case is that it requires extremely high concurrency which python does not do well because of its Global Interpreter Lock (GIL) limitation.

You can certainly end a TCP connection prematurely by sending a close (FIN) or reset (RST) signal packet in the middle of a stream to achieve what you want. It's just a question of whether python can run enough concurrent processes to achieve the goal. It sounds very doable, but you'll need to implement your own solution. Redis workers won't cut it.

I recommend looking into socket programming with C or Golang or something. Your own solution is the best you can expect

Prawnster - August 17, 2019, 8:15 am

(Aug 17, 2019 08:13 AM)True Demon Wrote: [\[CITE\]](#) I recommend looking into socket programming

with C or Golang or something. Your own solution is the best you can expect

Golang 100% with this imo, it's great for concurrency and it has a "fasthttp" package which is even faster than it's normal net/http package :)

CPyvoid - August 18, 2019, 2:24 pm

As one member mentioned above, python have concurrency problems because of it's GIL, which means that your python code can't run on more than 1 thread at a time.

Although this doesn't apply to I/O(networking and files), you still wont be able to take advantage of multiple cores.

Now easy workaround for this problem is to spawn N separate python processes where N is a number of processor cores you have.

Each of them can be responsible for one part of those 75mil sockets.

For spawning multiple processes you can use multiprocessing module, and of course they should be multithreaded.

* As long as your are not doing heavy computation stuff on that threads, you will be fine, and you won't have problems of 1 thread blocking all others from doing anything.

CyberStrike - August 19, 2019, 7:07 am

I love seeing that GoLang is suggested for such a project, Golang has a multi-threading API, Semaphore allows you to setup multiple workers in parallel on one task. Also GoLang tends to make imports quite easy, for example let's say that you need to denote what servers have timed out you don't have to import say net.dailtimeout this is done for you when you import net. Now of course if you are familiar with say C/CPP then that's fine as well, I just wanted to mention a couple of important aspects to using GoLang which can make your job hopefully easier.