A Census Of Minecraft Servers

Still, IPv6's sluggish rollout has some benefits; namely, because almost 100% of the Internet supports IPv4, I can do something I've always been curious about: scan the entire public Internet for Minecraft servers.

That can't possibly be possible!, you might say. Well, it's actually not as Herculean a task as it seems. IPv4 addresses are 32 bits long, so there are around four billion possible IP addresses. Some of these are reserved for things like private networks or multicast, leaving us with about 3,970,693,888 hosts to scan. At 20,000 hosts/second, we'll have scanned every publicly accessible IP in under three days.

Building the Scanner §

I'm not a very patient person, so I knew that much of my time spent working on this project would be devoted towards making it go fast. I had actually built a primitive Minecraft server scanner using NodeJS about a year ago; however, trying to scale it up to whole-Internet scanning rates quickly ended up becoming a nightmare. I did end up scanning the entirety of 0.0.0.0/0 (even though it took five days); however, I accidentally deleted half the data in a stupid blunder which I will never forgive myself for, prompting me to fall into a deep depression. This time, I started by finding the fastest TCP port scanner available, which brought me to MASSCAN.

MASSCAN is a TCP port scanning tool created by Robert David Graham. It's distinguished from its port-scanning siblings like nmap by the fact that it can go really fast: up to 25 million packets per second, enough to scan the entire IPv4 Internet in just five minutes. It achieves this ludicrously high speed by implementing a stripped-down TCP/IP stack in userspace, tailored for massively concurrent port scanning. (Userspace networking is one of the many ingredients in the C10M sauce.) MASSCAN can also conduct basic interactions with a scanned server to extract information about the services running on that port, retrieving what masscan refers to as a "banner". This makes it perfect for our purposes.

Unsurprisingly, masscan doesn't come with Minecraft support out of the box. No problem, we can implement it ourselves. I started by brushing up on the Minecraft protocol, for which wiki.vg is an invaluable reference.

Thankfully, the Minecraft protocol is simple enough that all we need to do is send one packet to collect all the available info about a server. Interestingly enough, this means that you can actually use netcat to ping a Minecraft server from the console. Here's what that looks like:

With that, I started coding. The process was rather uneventful, save for a couple brief moments of frustration which have since been memorialized in my commit history.

You can check out my finished code here. The magic occurs pretty much entirely in proto-

minecraft.c. First, we declare the probe which we'll send to every scanned server:

Now, all that's left is to parse the response from the server.

It's not the best C ever written, but whatever. Sue me.

With that done, all that was left to do was start the scan...

...and we were off to the races!

Data Processing §

The next day, while the scan was still running, I began thinking about how to process the data. One challenge I quickly noticed was that the data is strewn with false positives, i.e. the server responded with a SYN/ACK, but the Minecraft server ping couldn't be completed. While a tiny fraction of these are probably represented by servers with enable-status set to false, most are probably not running Minecraft at all. Hits like these represent around 90% of the data collected, inflating file size considerably. Thankfully, this didn't end up being too much of a problem besides making download times annoyingly long.

Another issue is that our data is way too large for NodeJS to swallow in one gulp. I naively tried to import the enormous JSON file, only for Node to spit out this error message:

Bummer. Oh well, it was foolish of me to expect that to work. My goal is to store the data in an SQLite database, which will hopefully make it easier to work with. So, I whipped up a short script:

Great, but it's running slow as a dog. A little profiling reveals the problem, and it's SQLite. If we comment out the line which inserts the record into the database, we can blast through the dataset at around 600,000 lines per second. Uncomment it, and that rate absolutely plummets. I took the liberty of making a chart to illustrate the difference.

Yeah... not very promising. MY GAMES It starts out slow, and quickly falls off a cliff into the absolutely abysmal range. At this rate, I calculated that it would take almost two hours for the data to be fully processed, and I wasn't even 50% done with the scan! We need to figure out how to drastically increase our SQLite insert rate, or I may have to ditch SQLite altogether.

With that, I embarked on my optimization journey, anticipating another sleepless night of running tests and creating charts. However, it turns out that my mistake was actually very simple. Some Googling brought me to the official SQLite FAQ, which has this to say about INSERT performance:

SQLite will easily do 50,000 or more INSERT statements per second on an average desktop computer. But it will only do a few dozen transactions per second.

By default, each INSERT statement is its own transaction. But if you surround multiple INSERT statements with BEGIN...COMMIT then all the inserts are grouped into a single transaction. The time needed to commit the transaction is amortized over all the enclosed insert statements and so the time per insert statement is greatly reduced.

Sounds easy enough to implement. I first collected a baseline measurement, where I measured the time it took to insert 1,000 rows into the table. This yielded an incredible rate of 8.2 rows/second.

Next, I modified the program to wrap our INSERTs in a single transaction. The change only involved two lines of code:

Lo and behold, our INSERT rate had jumped to over 8,400 rows/second! When the number came onto my screen, I sat there for a moment, mouth agape, pupils dilated, sweat running down my forehead, my feeble brain struggling to comprehend the tremendous difference that two measly lines of code could have. Then, disbelief set in. I opened up the database and ran some queries, but the data had been flawlessly imported.

I was suddenly overcome by emotion. A sensation of immense relief washed over me, as I realized that in just two lines I had made the program over 1,000 times faster. I felt as though I had just witnessed the hell-fire of the underworld, only to be told that I would be spared. At the same time, my earlobes burned with embarassment-after all, how could I have missed something so obvious? Am I even worthy of SQLite? No. I can't be. I am a lowly, dirty, JSON lover, and that is all I will ever be.

Okay, my reaction obviously wasn't as melodramatic as my reenactment suggests, but I was pretty relieved that the solution was so simple. Now I had a different problem on my hands: lots of seemingly invalid responses. About 4% of all the banners collected were not valid JSON. A vanishingly small percentage, sure, but I wasn't about to let those precious datapoints slip through my fingers. So, I began combing through the banners which failed to parse.

There were a couple low-hanging fruit which I chose to tackle first. For starters, MASSCAN doesn't have any understanding of character encodings. When it encounters a byte that isn't printable ASCII, it directly converts the byte to a Unicode codepoint. Thus, multibyte UTF-8 characters get absolutely mangled. Luckily, the fix was easy:

Much nicer!

There was another, bigger problem, though. In my attempts to fix my hopelessly broken packet parser, I ended up getting rid of the code which actually reads the response length, instead opting to log everything received from the server after the packet header as part of the banner. This resulted in a lot of JSON with extra garbage at the end, which greatly upsets JSON.parse. I needed a way to figure out where the actual JSON ended... but how?

Before I got to work on that, I decided to first sift through the data and identify any oddities-things that definitely weren't Minecraft servers. These were quite numerous; about 68% of the responses did not begin with an opening brace . Of these, the vast majority just sent back the probe baked into our custom build of MASSCAN. Three Minecraft servers had somehow made their way into this dejected pile of rejects, far too few for me to go about diagnosing the issue. There were also plenty of MySQL/MariaDB servers, and whatever this is:

Ok, back to work. My first attempt at extracting the JSON actually made things much worse, but a couple stabs at the problem brought me to this:

This takes us down to around 4,530 unparseable responses. Of these, 891 responses start with a '' yet cannot be extracted currently. A cursory look at these misfits suggests that most of them were cut off-maybe re-scan these later? That'll have to wait until another day. (Keep in mind that the scan still hasn't finished yet, but since MASSCAN iterates through IPs randomly, I can be fairly confident that these numbers will be representative of our final dataset.)

Analyzing the Data §

I woke up on Monday to some good news: the scan was done! Finally, we're ready to start dirtying our fingers with some data. My concerns about losing responses to my crappy parser turned out to be unfounded, because only 0.7% of the collected hits were recognized as incomplete JSON. Anyways, to answer the original question: how many Minecraft servers are out there? Drumroll, please...

160,992

Give or take. There's probably some pretty big error bars on that number. Shodan reports a very similar count, while bStats says that there are about 175,000 online servers. The difference probably represents servers that aren't publicly accessible.

Here are the ten greatest player counts:

99,999,999 20,220,320 20,220,320 20,220,319 20,220,319
9,128,312 114,516 114,515 114,514 114,514

and the ten lowest player counts:

-1337 -46 -2 -1 -1 -1 -1 -1 -1 -1

Minecraft servers also send a sample of the online players by default, letting us make an incomplete list of who's online at any given moment. I couldn't think of anything to do with this data, but maybe your name's in the list!

There are mods/plugins which allow you to summon fake players under any name; technical

players often use these fake players to keep farms loaded. As a result, many spoofed names like jeb_ and notch show up in the list. However, the vast majority of these usernames probably represent real players.

Geographical Distribution §

Where are most Minecraft servers located? We can make a map of which countries have the most Minecraft servers:

The US is quite Minecraft server-dense, with over one server per 10,000 people. However, Germany ends up taking the prize for most Minecraft servers per capita, with a whopping four servers for every 10,000 people. This is probably thanks to cheap hosting offerings from companies like Hetzner.

Did you know this type of map is called a choropleth? Well, now you do!

Server Softwares §

The ping response includes a "version" field, which can be used to determine what software a server is running. There's a problem, though: you can use plugins to display a custom version in place of the default version message, resulting in thousands of unique "versions" being recorded. I had to manually go through and remove all of these while taking care to not discard any legitimate version strings. In the process, I discovered dozens of obscure server softwares that I had no idea existed (mostly shitty Spigot forks), and learned how to say "maintenance" in at least five languages. Anyways, here's a look at the most common servers:

I also matched the strings for certain keywords to identify which specific server "brands" were most common.

I also compared the popularity of different server software "brands".

DDoS Protection §

One statistic that I unwittingly collected during the scan was the number of servers with DDoS protection. Most Minecraft DDoS protection services operate by filtering traffic through a proxy to the actual server, allowing the server host to take use of the provider's beefier network infrastructure. The two dominant players in this field are TCPShield and Cosmic Guard.

We're able to detect IPs representing proxies for these two providers thanks to the fact that when pinging a server, a Minecraft client will send the hostname used to connect to the server. However, in our case, our probe uses the same hostname ("example.com") every time; normal Minecraft servers ignore this value, but DDoS proxies will detect that an invalid hostname was used and deny the connection. Here's what it looks like when you try to join a

DDoS-protected server using its IP:

We can search the dataset for ping responses which look like this. Turns out there are about 742 TCPShield proxies and 1,139 Cosmic Guard proxies on the public Internet.

Modded Servers §

Let's see what mods people are playing with! Here's the top 20 most popular mods (Forge only).

Mods marked with a † are library mods, which are used as dependencies by other mods. As expected, they make up a lot of the most popular mods.

Server Hosts §

Before we start, here's some eye candy:

What you're looking at is every online Minecraft server in IPv4 space, plotted along a Hilbert curve (a style of visualization that was pioneered by xkcd). I won't go into too much detail here, but essentially the Hilbert curve is a way of arranging linear points in 2D space such that a sequence of points will always end up in a compact region (unlike other mappings, where two adjacent points might end up far apart because of wrapping).

Each IP is associated with an autonomous system; by grouping IPs with ASNs, we can rank hosts by how many Minecraft servers they're running. Here are our top 10 contenders:

Oh, and just for fun, we can label the servers on the Hilbert map according to ASN. (I ran out of colors after about 15, unfortunately.)

Epilogue §

This blogpost ended up being one of the most exciting and interesting ones I've worked on so far. If you have any ideas for interesting ways to analyze the data I've collected, please don't hestitate to leave a comment or contact me directly. Unfortunately, I probably won't be releasing the dataset since it will inevitably be used to target unprotected servers.