



Contribution ID: 303

Type: **not specified**

How to share IPv4 addresses by partitioning the port space

Tuesday 13 September 2022 12:00 (30 minutes)

When establishing connections, a client needs a source IP address. For better or worse, network and service operators often assign traits to client IP addresses such as a reputation score, geolocation or traffic category, e.g. mobile, residential, server. These traits influence the way a service responds.

Transparent Web proxies, or VPN services, obfuscate true client IPs. To ensure a good user experience, a transparent proxy service should carefully select the egress IPs to mirror the traits of the true-client IP.

However, this design is hard to scale in IPv4 due to the scarcity of IP addresses. As the price of IPv4 addresses rise, it becomes important to make efficient use of the available public IPv4 address pool.

The limited pool of IPv4 addresses, coupled with a desire to express traits known to be used by other services, presented Cloudflare with a challenge: The number of server instances in a single Point of Presence exceed the number of IPv4 egress addresses available – a disconnect that is exacerbated by the need to partition available addresses further according to traits.

This has led us to search for ways to share a scarce resource. The result is a system where a single egress IPv4 address, with given traits, is assigned to not one, but multiple hosts. We make it possible by partitioning ephemeral TCP/UDP port space and dividing it among the hosts. Such a setup avoids use of stateful NAT, which is undesirable due to scalability and single-point-of-failure concerns.

From previous work [1] we know that the Linux Sockets API is poorly suited to a task of establishing connections from a given source port range. Opening a TCP connection from a port range is only possible if the user re-implements the free port search - a task that the Linux TCP stack already performs when auto-binding a socket.

On UDP sockets, selecting source port range for a connected socket turns out to be very difficult. Correctly dealing with connected sockets is important because they are a desirable tool for egress traffic, despite their memory overhead. Currently, the Linux API forces the user to choose: Either use a single connected UDP socket owning a local port, which greatly limits the number of concurrent UDP flows; or, alternatively, somehow detect a connected-socket conflict when creating connected UDP sockets, which share the local address.

We previously built a detection mechanism with a combination of querying `sock_diag` and toggling the port sharing on and off after binding the socket [1]. Depending on perspective, the process might be described by some as arduous, or by others as an ingenious hack that works.

Recent innovations such as these demonstrate that sharing the finite set of ports and addresses among larger sets of distributed processes is a problem not yet completely solved for the Linux Sockets API. At Cloudflare we have come up with a few different ideas to address the shortcomings of the Linux API. Each of them makes the task of sharing an IPv4 address between servers and/or processes easier, but the degree of user-friendliness varies.

In no particular order, the ideas we have evaluated are:

1. Introduce a per-socket configuration option for narrowing down the IP ephemeral port range.
2. Introduce a flag to enforce unicast semantics for connected UDP sockets, when sharing the local address (`SO_REUSEADDR`). With the flag set, it should not be possible to create two connected UDP sockets with conflicting 4-tuples (`{local IP, local port, remote IP, remote port}`).

3. Extend the late-bind feature (IP_BIND_ADDRESS_NO_PORT) to UDP sockets, so that dynamically-bound connected UDP sockets can share a local address as long as the remote address is unique.
4. Extend Linux sockets API to let the user atomically bind a socket to a local and a remote address with conflict detection. Akin to what the Darwin connectx() syscall provides.
5. Introduce a post-connect() BPF program to allow user-space processes to prevent creation of connected UDP sockets with conflicting 4-tuples.

During the talk, we will go over the challenges of designing a distributed proxy system that mirrors client IP traits, which led us to look into IP sharing and port space partitioning.

Then, we will shortly explain production tested implementation of TCP/UDP port space partitioning using only existing Linux API features.

Finally, we will describe the proposed API improvement ideas, together with their pros and cons and implementation challenges.

We will accompany the most promising, according to our judgment, ideas with a series of RFC patches posted prior to the talk for the upstream community consideration.

[1] <https://blog.cloudflare.com/how-to-stop-running-out-of-ephemeral-ports-and-start-to-love-long-lived-connections/>

I agree to abide by the anti-harassment policy

Yes

Primary authors: SITNICKI, Jakub (Cloudflare); MAJKOWSKI, Marek (Cloudflare)

Presenters: SITNICKI, Jakub (Cloudflare); MAJKOWSKI, Marek (Cloudflare)

Session Classification: eBPF & Networking

Track Classification: eBPF & Networking Track