September 13, 2022

How to share IPv4 addresses by partitioning the port space

Jakub Sitnicki, Cloudflare
Marek Majkowski, Cloudflare

Linux Plumbers Conference
Dublin, Ireland
September 12-14, 2022

Part I

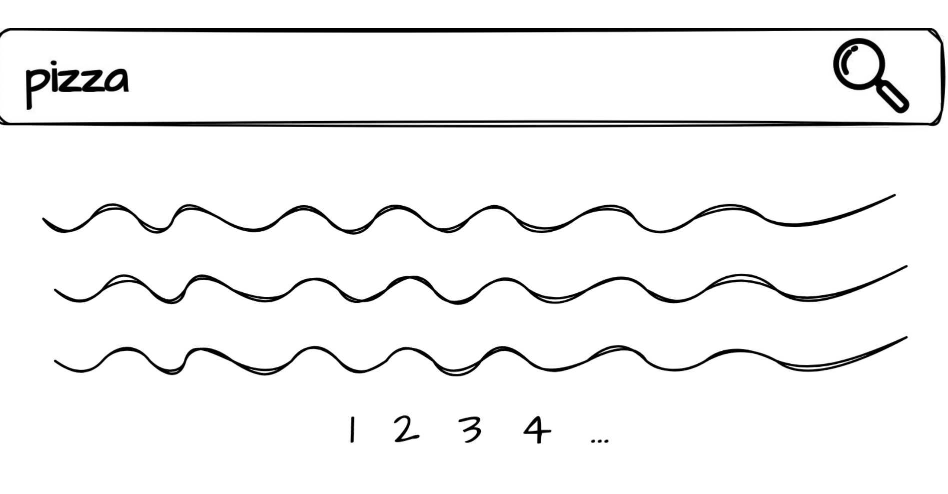
The Situation

Not all public IP addresses are created equal

```
198.51.100.183 != 203.0.113.72
```

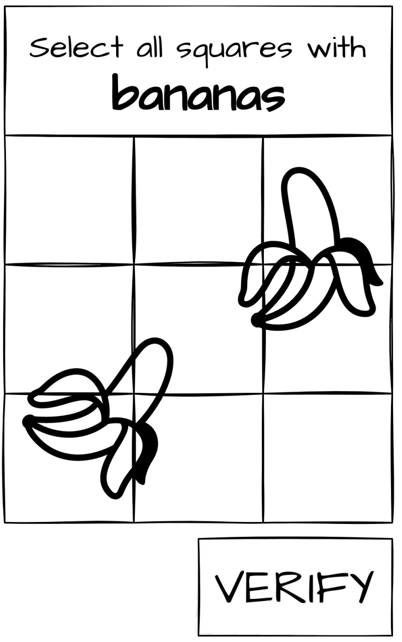
IP addresses have traits

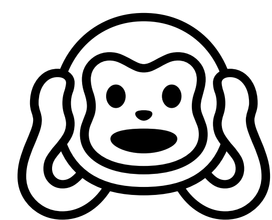
198.51.100.183 -> geo-location



IP addresses have traits

198.51.100.183 -> reputation





I, ~~Robot~~ Proxy

We run a VPN called WARP

https://blog.cloudflare.com/1111-warp-better-vpn/

We run an anycast network

AMSTERDAM
192.0.2.0/24

SAN JOSE
192.0.2.0/24

LONDON
192.0.2.0/24

INTERNET

LOS ANGELES
192.0.2.0/24

FRANKFURT
192.0.2.0/24

NEW YORK
192.0.2.0/24

We run an anycast network

AMSTERDAM
192.0.2.0/24

SAN JOSE
192.0.2.0/24

203.0.113.72

LONDON
192.0.2.0/24

198.51.100.183

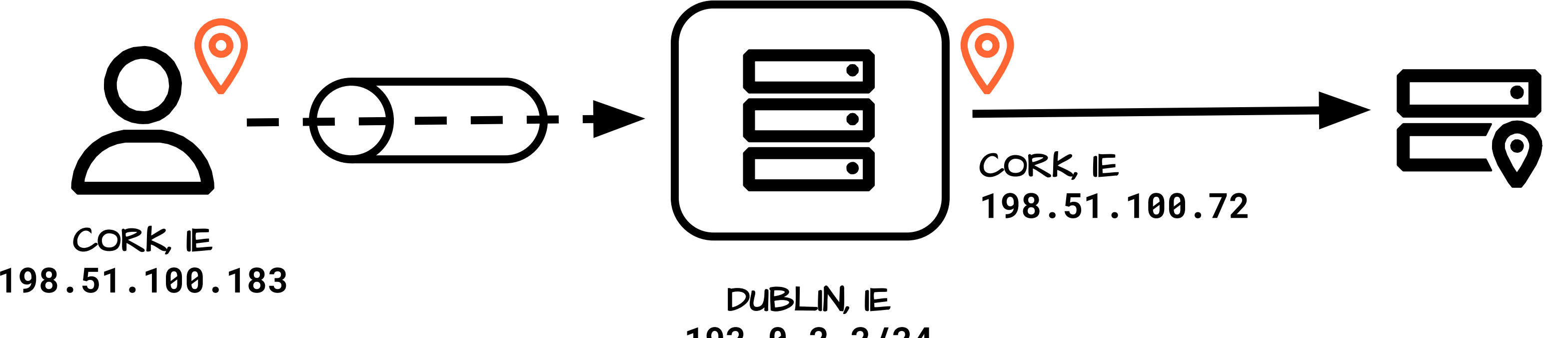
LOS ANGELES
192.0.2.0/24

FRANKFURT
192.0.2.0/24

NEW YORK
192.0.2.0/24

I, ~~Robot~~ Proxy

-> (location, reputation, ...)



Part II

The Problem

Cork IP on every CF server	Cork IP on one server
Need lots of IPs	Need just one IP
Egress always local	Forward for egress
Excellent reliability	Poor reliability

$$$ ⟷ $

Cork IP on every server	Cork IP per DC	Cork IP on one server
Need lots of IPs	Need some IPs	Need just one IP
Egress always local	Egress mostly local	Forward for egress
Excellent reliability	Good reliability	Poor reliability
$$$	$$	$

Local egress is happy user

Local egress is happy SRE

SRE

Local egress is happy SRE

SRE

SNAT on router?

ext	int
198.51.100.72:xyz	10.0.0.1:abc

CORK, IE
198.51.100.72

10.0.0.1

10.0.0.2

10.0.0.3

Distributed SNAT?

SNAT

CORK, IE
198.51.100.72

SNAT

CORK, IE
198.51.100.72

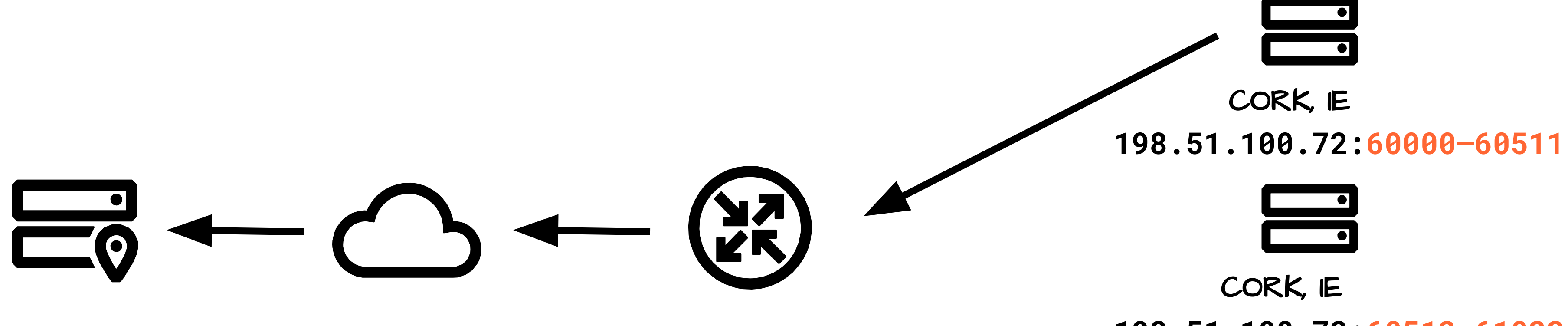
SNAT

CORK, IE
198.51.100.72

Part III

The Solution

Share the IP, partition the port range



Each server gets 512 egress ports per IP
Return traffic routed by (dst IP, dst port) !

How to open connections from a given port range?

```
system("sysctl -w net.ipv4.ip_local_port_range='60000 60511'")  ☹

s = socket(AF_INET, SOCK_STREAM)
s.setsockopt(SOL_IP, IP_BIND_ADDRESS_NO_PORT, 1) # share the local port
s.bind(("192.0.2.1", 0)) # let the kernel find a free 4-tuple
s.connect(("1.1.1.1", 53))
```

https://kernelnewbies.org/Linux_4.2#Networking

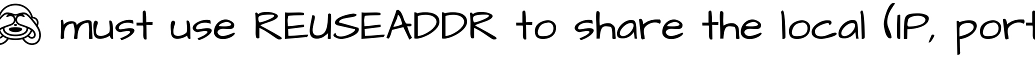
Sample the port range, try each port

{ ~~60,000~~ ; 60,002 ; 60,003 ; 60,007 ; 60,009 }

```
s1 = socket(AF_INET, SOCK_STREAM)
s1.setsockopt(SOL_SOCKET, SO_REUSEADDR, 1)
s1.bind(("192.0.2.1", 60_000))
s1.connect(("1.1.1.1", 53)) # EADDRNOTAVAIL
```

4-tuple is busy
try next port

```
s2 = socket(AF_INET, SOCK_STREAM)
s2.setsockopt(SOL_SOCKET, SO_REUSEADDR, 1)
s2.bind(("192.0.2.1", 60_002))
s2.connect(("1.1.1.1", 53)) # OK
```



Sample the port range, try each port

{ ~~60,000~~ ; 60,002 ; 60,003 ; 60,007 ; 60,009 }

```
s1 = socket(AF_INET, SOCK_STREAM)
s1.setsockopt(SOL_SOCKET, SO_REUSE
s1.bind(("192.0.2.1", 60_00
s1.connect(("1.1.1.1", 53))   ADD  TAVAI


s2 = socket(AF_INET, SOCK_STREAM)
s2.setsockopt(SOL_SOCKET, SO_REUSEADDR   )
s2.bind(("192.0.2.1", 60_002))
s2.connect(("1.1.1.1", 53))
```

TCP ONLY

4-tuple is busy
try next port

Sample the port range, try each port

What about UDP?

TCP & UDP semantics differ

https://blog.cloudflare.com/doom-multiplayer-workers/

TCP & UDP connect( ) semantics differ

```
s1.setsockopt(SOL_SOCKET, SO_REUSEADDR, 1)
s2.setsockopt(SOL_SOCKET, SO_REUSEADDR, 1)
```

	TCP	UDP
`s1.bind(192.0.2.1, 60000)`	⇒ OK	⇒ OK
`s1.connect(1.1.1.1, 53)`	⇒ OK	⇒ OK
`s2.bind(192.0.2.1, 60000)`	⇒ OK	⇒ OK
`s2.connect(1.1.1.1, 53)`	**⇒ EADDRINUSE**	**⇒ OK**

Must use REUSEADDR to share the local (IP, port)
Otherwise -> only as many concurrent flows as local ports

UDP port conflict detection in user-space

```
s = socket(AF_INET, SOCK_DGRAM)
s.setsockopt(SOL_SOCKET, SO_REUSEADDR, 1)
s.bind(("192.0.2.1", 60_000))

cookie = s.getsockopt(SOL_SOCKET, SO_COOKIE, 8)
s.setsockopt(SOL_SOCKET, SO_REUSEADDR, 0) # block port
c = netlink_udp_lookup(("192.0.2.1", 60_000), ("1.1.1.1", 53))
if c != cookie: # conflict?
    raise OSError(EADDRNOTAVAIL)
s.connect(("1.1.1.1", 53))
s.setsockopt(SOL_SOCKET, SO_REUSEADDR, 1) # unblock port
```

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

Part IV

The Proposal

Again - How to open connections from a given port range?

```
system("sysctl -w net.ipv4.ip_local_port_range='60000 60511'")  ☹

s = socket(AF_INET, SOCK_STREAM)
s.setsockopt(SOL_IP, IP_BIND_ADDRESS_NO_PORT, 1) # share the local port
s.bind(("192.0.2.1", 0)) # let the kernel find a free 4-tuple
s.connect(("1.1.1.1", 53))
```

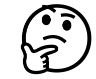
https://kernelnewbies.org/Linux_4.2#Networking

Ephemeral port range per socket

```
system("sysctl -w net.ipv4.ip_local_port_range='60000 60009'")

s = socket(AF_INET, SOCK_STREAM)
s.setsockopt(SOL_IP, IP_BIND_ADDRESS_NO_PORT, 1)
s.setsockopt(SOL_IP, IP_LOCAL_PORT_RANGE, [60000, 60009])
s.bind(("192.0.2.1", 0)) # let the kernel find a free 4-tuple
s.connect(("1.1.1.1", 53))
```

What if we could do this?



Ephemeral port range per socket

📨 RFC posted to netdev

```
--- a/include/net/inet_sock.h
+++ b/include/net/inet_sock.h
@@ -238,6 +238,10 @@ struct inet_sock {
       __be32                         mc_addr;
       struct ip_mc_socklist __rcu    *mc_list;
       struct inet_cork_full   cork;
+      struct {
+              __u16 lo;
+              __u16 hi;
+      }                       local_port_range;
 };
```

Set in ip_sockglue.c by IP_LOCAL_PORT_RANGE option handler

Ephemeral port range per socket

```
void inet_sk_get_local_port_range(const struct sock *sk, int *low, int *high)
{
    const struct inet_sock *inet = inet_sk(sk);
    const struct net *net = sock_net(sk);
    int lo, hi;

    inet_get_local_port_range(net, &lo, &hi);

    if (unlikely(inet->local_port_range.lo))
        lo = clamp_val(inet->local_port_range.lo, lo, hi);
    if (unlikely(inet->local_port_range.hi))
        hi = clamp_val(inet->local_port_range.hi, lo, hi);

    *low = lo;
    *high = hi;
}
```

Called in ->connect() ⇒ tcp_v4_connect() ⇒ inet_hash_connect()

What about UDP?

Does it also work for UDP? 🤔

```
system("sysctl -w net.ipv4.ip_local_port_range='60000 60009'")

s = socket(AF_INET, SOCK_DGRAM)
s.setsockopt(SOL_IP, IP_BIND_ADDRESS_NO_PORT, 1)
s.setsockopt(SOL_IP, IP_LOCAL_PORT_RANGE, [60000, 60009])
s.bind(("192.0.2.1", 0)) # let the kernel find a free 4-tuple
s.connect(("1.1.1.1", 53))
```

What about UDP?

NOPE

NEW GAME
CHOOSE SKILL LEVEL:
I'M TOO YOUNG TO DIE.
HEY, NOT TOO ROUGH.
HURT ME PLENTY.
ULTRA-VIOLENCE.
NIGHTMARE!

9 AMMO
82% HEALTH
2 3 4 5 6 7 ARMS
83% ARMOR
BULL 49 / 200
SHEL 9 / 50
RCKT 0 / 50
CELL 0 / 300

No source port sharing with late autobind

IP_BIND_ADDRESS_NO_PORT works
but not as you expect it

```
# if there is just one ephemeral port
system("sysctl -w net.ipv4.ip_local_port_range='60000 60000'")

s1 = socket(AF_INET, SOCK_DGRAM)
s1.setsockopt(SOL_IP, IP_BIND_ADDRESS_NO_PORT, 1)
s1.bind(("192.0.2.1", 0))
s1.connect(("1.1.1.1", 53))

s2 = socket(AF_INET, SOCK_DGRAM)
s2.setsockopt(SOL_IP, IP_BIND_ADDRESS_NO_PORT, 1)
s2.bind(("192.0.2.1", 0))
s2.connect(("1.0.0.1", 53)) # ⇒ EAGAIN
```

Can we make it work?

```
int inet_dgram_connect(struct socket *sock, struct sockaddr *uaddr,
                       int addr_len, int flags)
{
        // …
        if (data_race(!inet_sk(sk)->inet_num) && inet_autobind(sk))
                return -EAGAIN;
        return sk->sk_prot->connect(sk, uaddr, addr_len);
}
```

sk->sk_prot->get_port(sk, snum)

https://elixir.bootlin.com/linux/v6.0-rc4/source/net/ipv4/af_inet.c#L557

Can we make it work?

sk->sk_prot->get_port(sk, snum)

```
int udp_lib_get_port(struct sock *sk, unsigned short snum,
                     unsigned int hash2_nulladdr)
{
        if (!snum) {
                // (A) port not specified - find a free one
        } else {
                // (B) port given - check if free
        }
found:
        // publish the new binding
}
```

https://elixir.bootlin.com/linux/v6.0-rc4/source/net/ipv4/udp.c#L232

UDP get_port algorithm

u16 snum

0110 010111 0110

high bits

low bits

udp_hashfn()

λ

udptable->hash[]

sock{}

?

udp_lib_lport_inuse()

bitmap

UDP get_port algorithm

udp_lib_lport_inuse()

↓

↓

{ netns, src addr, src port, SO_BINDTODEVICE, SO_REUSEADDR }

↓

conflict?

port free

port busy

https://elixir.bootlin.com/linux/v6.0-rc4/source/net/ipv4/udp.c#L134

How can we detect 4-tuple conflicts?

u16 snum

0110 | 0101110110

high bits

low bits

udp_hashfn()

λ

udptable->hash[]

sock{}

some new func

bitmap

How can we detect 4-tuple conflicts?

some_new_func()

↓

↓

{ netns, src addr, src port, SO_BINDTODEVICE, SO_REUSEADDR,
dst addr, dst port }

↓

conflict?

port free

port busy

We could do it like so…

```
int (*bind_add)(struct sock *sk, struct sockaddr *addr, int addr_len);
```

```
@@ -2937,6 +3028,7 @@ struct proto udp_prot = {
        .sendmsg                = udp_sendmsg,
        .recvmsg                = udp_recvmsg,
        .sendpage               = udp_sendpage,
+       .bind_add               = udp_v4_bind_add,
        .release_cb             = ip4_datagram_release_cb,
        .hash                   = udp_lib_hash,
        .unhash                 = udp_lib_unhash,
```

We could do it like so…

```
int inet_dgram_connect(struct socket *sock, struct sockaddr *uaddr,
                       int addr_len, int flags)
{
        // …
        if (data_race(!inet_sk(sk)->inet_num) && inet_autobind(sk))
                return -EAGAIN;
        return sk->sk_prot->connect(sk, uaddr, addr_len);
}
```

https://elixir.bootlin.com/linux/v6.0-rc4/source/net/ipv4/af_inet.c#L557

We could do it like so…

```
int inet_dgram_connect(struct socket *sock, struct sockaddr *uaddr,
                       int addr_len, int flags)
{
        // …
        if (data_race(!inet_sk(sk)->inet_num)) { // late bind
                if (some condition) {
                        if (sk->sk_prot->bind_add(sk, uaddr, addr_len))
                                return -EAGAIN;
                } else {
                        if (inet_autobind(sk)) // sk_prot->get_port()
                                return -EAGAIN;
                }
        }
        return sk->sk_prot->connect(sk, uaddr, addr_len);
}
```

We could do it like so…

```
if (some condition) {
        if (sk->sk_prot->bind_add(sk, uaddr, addr_len))
                return -EAGAIN;
}
```

```
some condition = source IP != wildcard &&
                 SO_REUSEADDR &&
                 IP_BIND_ADDRESS_NO_PORT
```

📨 RFC posted to netdev

User API change

IF 4-tuple is not unique AND we are out of ephemeral ports THEN

SO_REUSEADDR	IP_BIND_ADDRESS_NO_PORT	socket(AF_INET, SOCK_DGRAM) bind(192.0.2.1, 0) connect(1.1.1.1, 53)
F	F	bind ⇒ EADDRINUSE
F	T	bind ⇒ OK connect ⇒ EAGAIN
T	F	bind ⇒ OK connect ⇒ OK conflict!
T	T	bind ⇒ OK **connect ⇒ ~~OK~~ EAGAIN**

Part V

The Alternatives

Yet Another Socket Option

```
s = socket(AF_INET, SOCK_DGRAM)
s.setsockopt(SOL_SOCKET, SO_REUSEADDR, 1) # I'm a multicast socket!
s.setsockopt(SOL_UDP, UDP_UNICAST, 1)     # No, I'm a unicast socket!
s.bind(('127.1.1.1', 1111))
s.connect(('127.2.2.2', 2222)) # EADDRINUSE if 4-tuple conflict
```

PoC code available on request

Yet Another Syscall – connectx() like

NAME

connectx -- initiate a connection on a socket

SYNOPSIS

```
#include <sys/socket.h>

int
connectx(int socket, const sa_endpoints_t *endpoints,
    sae_associd_t associd, unsigned int flags, const struct iovec *iov,
    unsigned int iovcnt, size_t *len, sae_connid_t *connid);
```

DESCRIPTION

The parameter *socket* is a socket. In general, **connectx**() may be used as a substitute for cases when **bind(2)** and **connect(2)** are issued in succession, as well as a mechanism to transmit data at connection establishment time.

https://www.manpagez.com/man/2/connectx/

Yet Another BPF Hook – post-connect()

```
struct bpf_sock_tuple tuple = {};
tuple.ipv4.saddr = ctx->user_ip4;
tuple.ipv4.sport = ctx->user_port;
tuple.ipv4.daddr = sk->src_ip4;
tuple.ipv4.dport = bpf_htons(sk->src_port);

struct bpf_sock *nsk = bpf_sk_lookup_udp(ctx, &tuple, sizeof(tuple.ipv4),
                                          BPF_F_CURRENT_NETNS, 0);

if (!nsk) {
        return 1; /* Conflict with existing socket. Fail connect(). */
}
```

https://github.com/cloudflare/cloudflare-blog/blob/master/2022-02-connectx/ebpf_connect4/connect4_ebpf.c

Thank you

Jakub
jakub@cloudflare.com
@jkbs0

Marek
marek@cloudflare.com
@majek04