connect() - why you so slow?

Frederick Lawler

Systems Engineer @ Cloudflare

- security_create_user_ns()
- CVE-2022-47929: traffic control noqueue no problem?
- pci_(alert|crit|dbg|emerg|err|info|notice|warn) printk macros

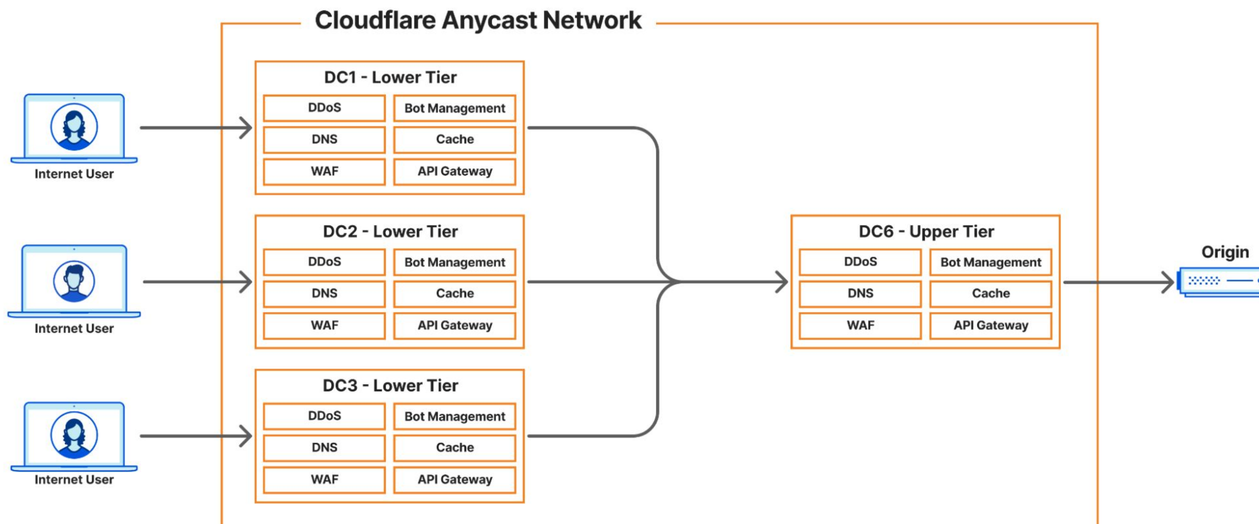
50k egress unicast connections to a single destination... Who does that?

CLOUDFLARE

Who does that?

CLOUDFLARE

CDN request flow for uncached assets



CLOUDFLARE

```
$ sysctl net.ipv4.ip_local_port_range
net.ipv4.ip_local_port_range = 9024    65535
```

CLOUDFLARE

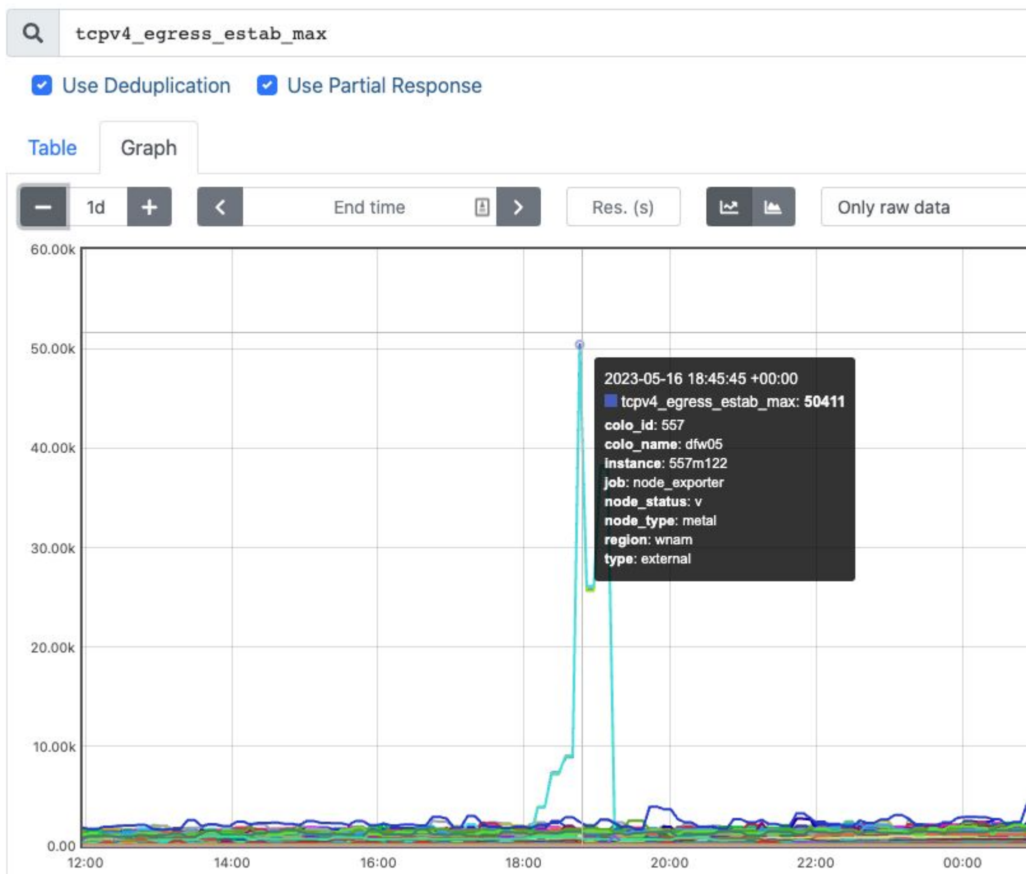
bind() before connect()

```
sk = socket(AF_INET, SOCK_STREAM)
sk.setsockopt(IPPROTO_IP, IP_BIND_ADDRESS_NO_PORT, 1)
sk.bind((src_ip, 0))
sk.connect((dest_ip, dest_port))
```

How to stop running out of ephemeral ports and start to love long-lived connections

Who does that?

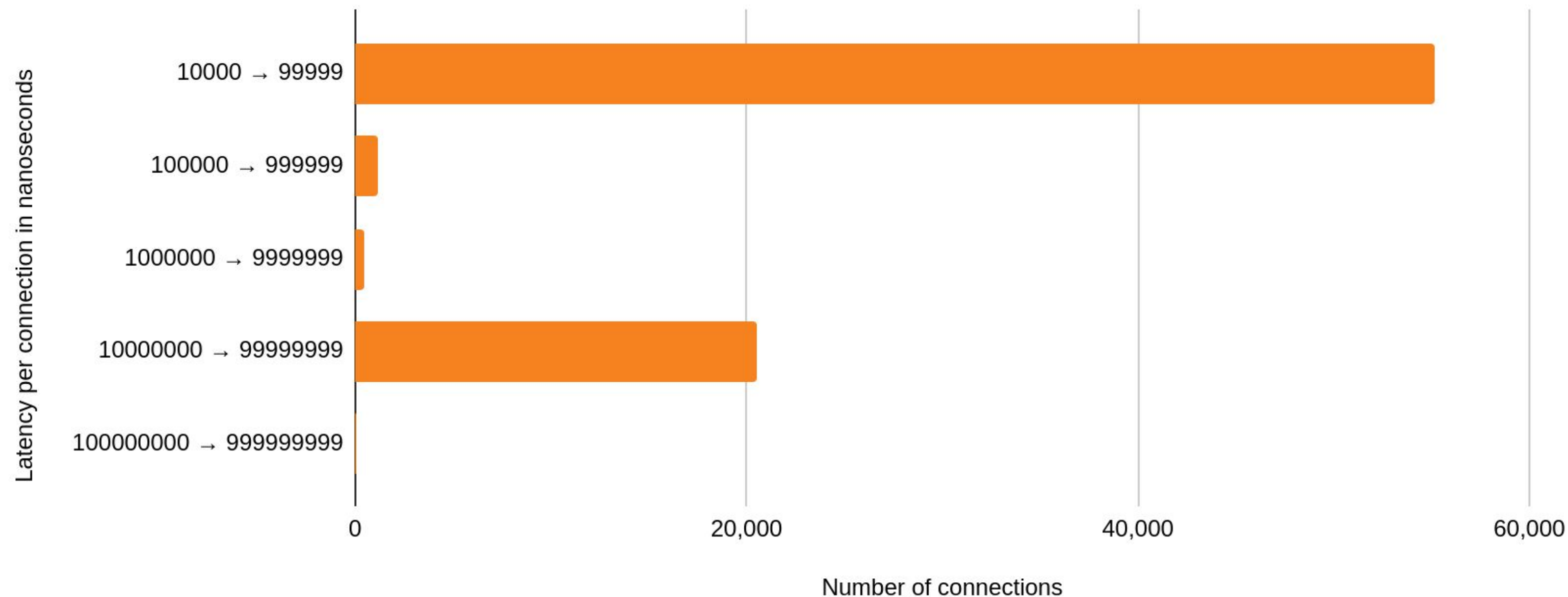
CLOUDFLARE



CLOUDFLARE

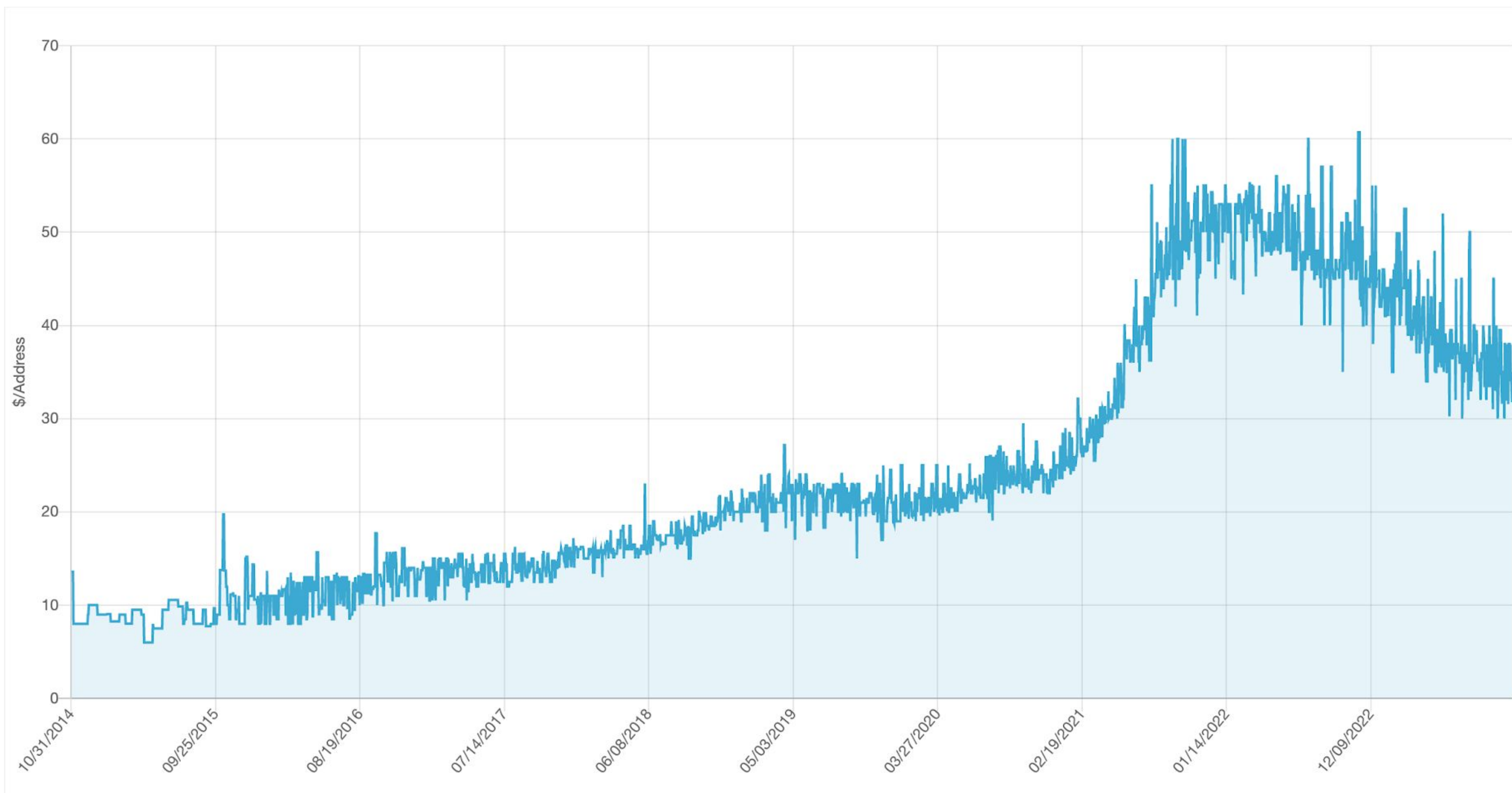
```
2 IPv4 addresses for this service
```

tcp_v4_connect() func latency 2 IPv4 address



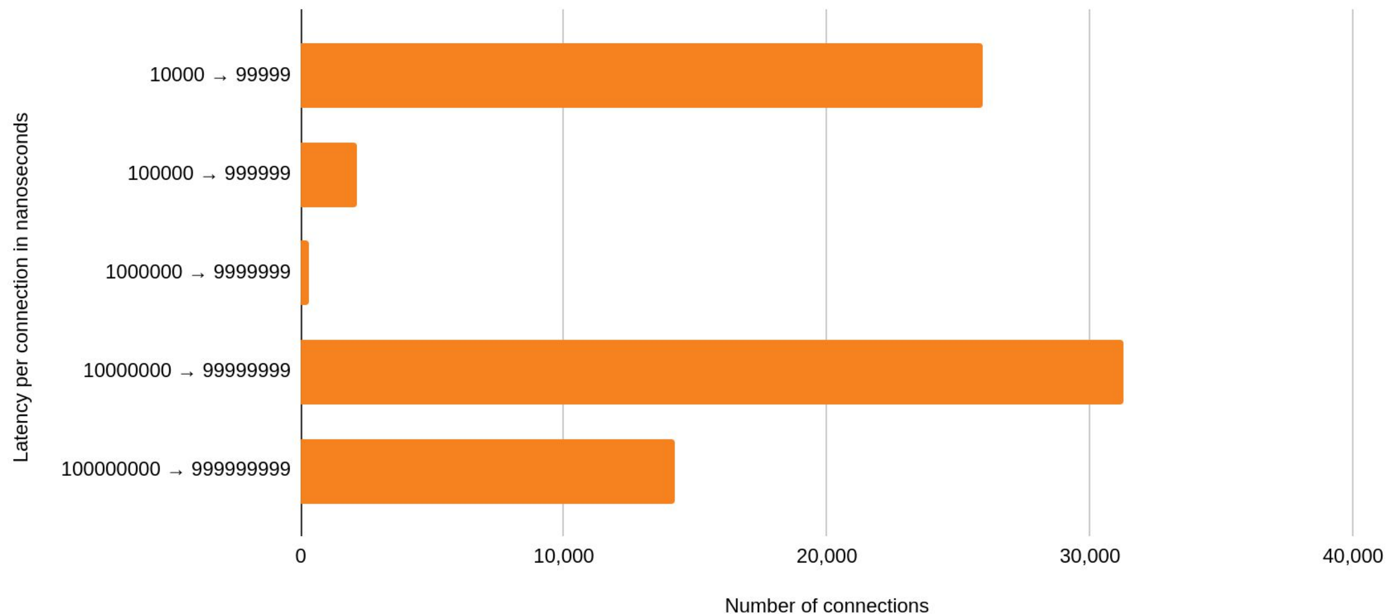
CLOUDFLARE

`1 IPv4 addresses for this service`

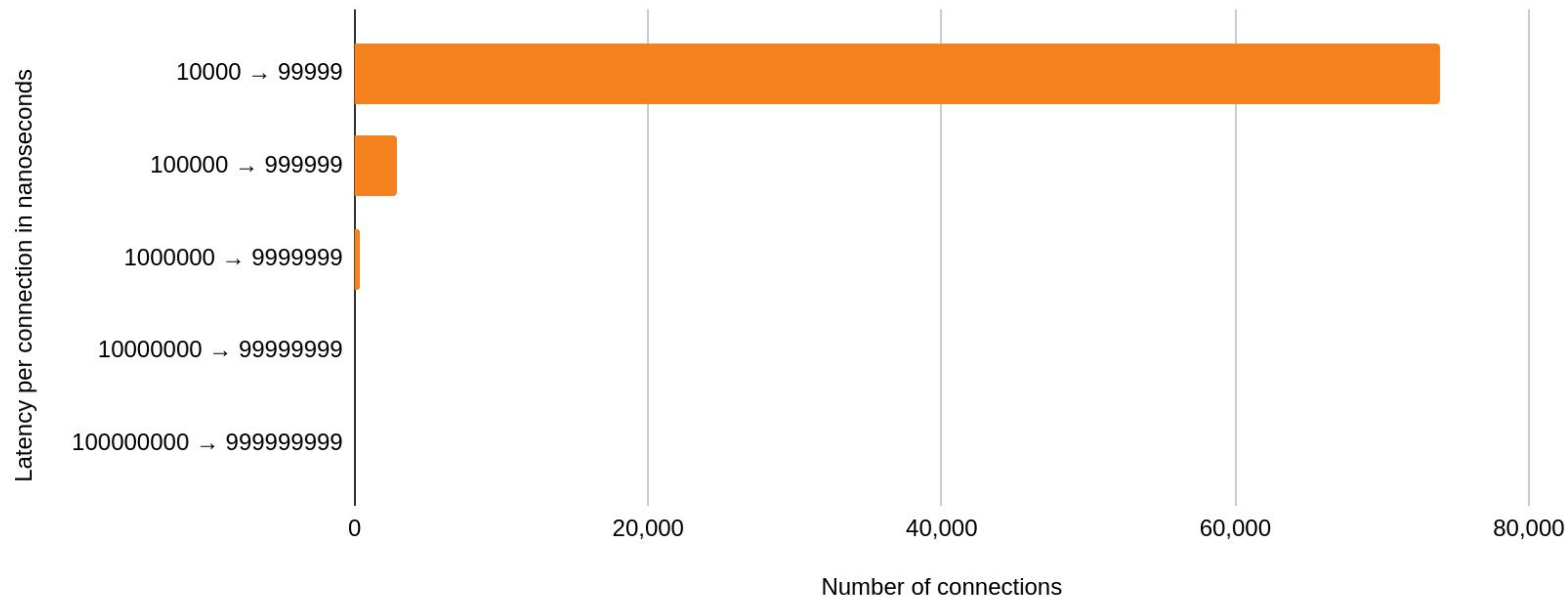


IPv4 sales data. Source: Hilco Streambank.

tcp_v4_connect() func latency 1 IPv4 address



tcp_v4_connect() func latency 3 IPv4 address (for fun)



CLOUDFLARE

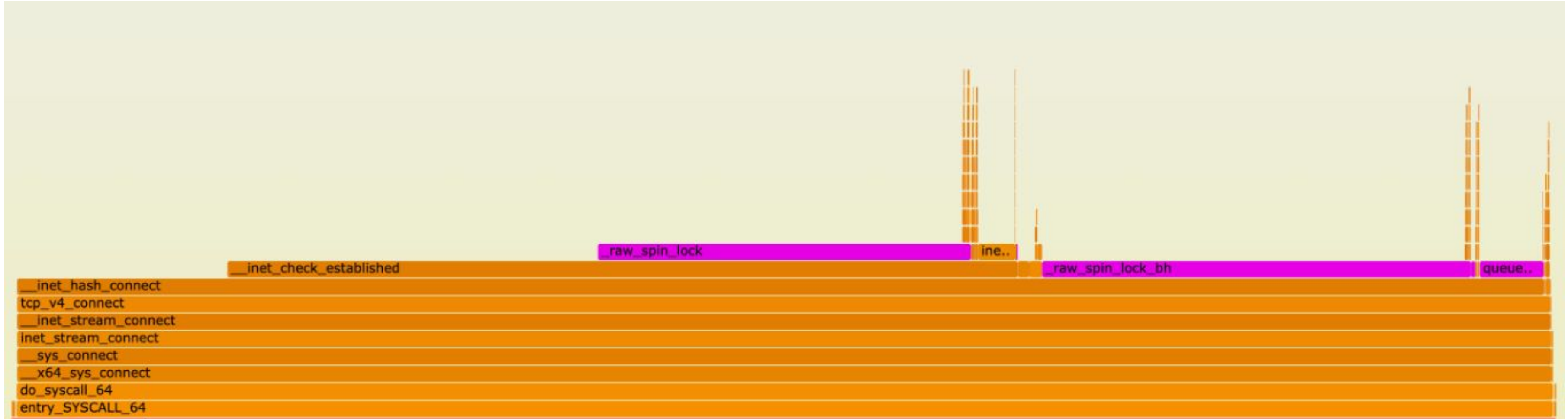
This is fine for most workloads, but for Cloudflare...

- Customers largely still leverage IPv4
- Similar performance with 1 IPv4's as we'd see with 3
- Leverage our infrastructure to lazily hand off excess connections
  ie. fail fast

Time to investigate: TCP connect() why you so slow?

CLOUDFLARE

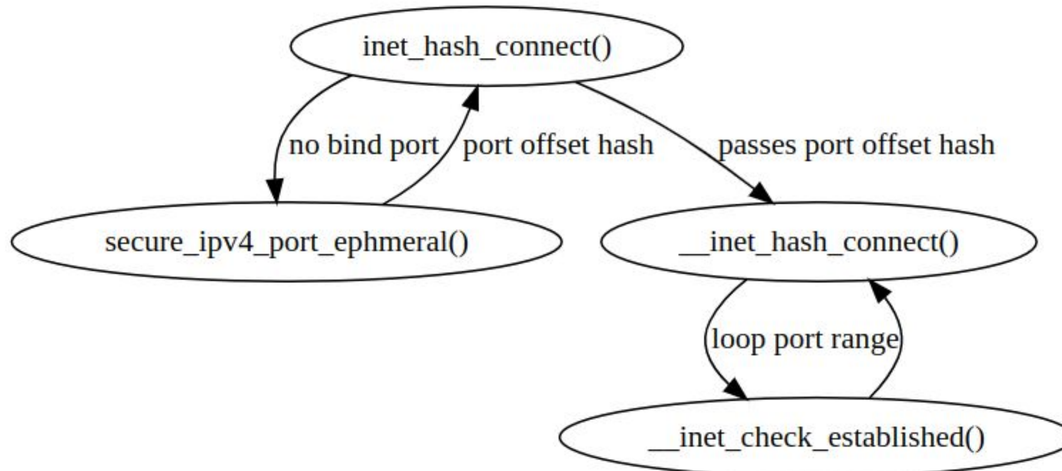
Time to investigate: TCP connect() why you so slow?



CLOUDFLARE

inet_hash_connect() overview

- Called in both TCP IPv4 & IPv6 contexts; but we'll be focusing on IPv4
- We assume the kernel has to pick a port



CLOUDFLARE

__inet_hash_connect() overview

[net/ipv4/inet_hashtables.c: __inet_hash_connect]

```
   offset &= ~1U;

other_parity_scan:
    port = low + offset;
    for (i = 0; i < remaining; i += 2, port += 2) {
        if (unlikely(port >= high))
            port -= remaining;

        inet_bind_bucket_for_each(tb, &head->chain) {
            if (inet_bind_bucket_match(tb, net, port, l3mdev)) {
                if (!check_established(death_row, sk,
                               port, &tw))
                    goto ok;
                goto next_port;
            }
        }
    }

    offset++;
    if ((offset & 1) && remaining > 1)
        goto other_parity_scan;
```

CLOUDFLARE

__inet_hash_connect() overview: initial port selection

```
   offset &= ~1U;

other_parity_scan:
    port = low + offset;
    for (i = 0; i < remaining; i += 2, port += 2) {
        if (unlikely(port >= high))
            port -= remaining;

        inet_bind_bucket_for_each(tb, &head->chain) {
            if (inet_bind_bucket_match(tb, net, port, l3mdev)) {
                if (!check_established(death_row, sk,
                                       port, &tw))
                    goto ok;
                goto next_port;
            }
        }
    }

    offset++;
    if ((offset & 1) && remaining > 1)
        goto other_parity_scan;
```

- Offset is randomly generated
- Offset is set to an even number
- Picked port is either “even” or “odd” based on net.ipv4.ip_local_port_range’s low port eg. 9024

CLOUDFLARE

Are we starting our loop at the same offset each connect()?

2011

net: Compute protocol sequence numbers and fragment IDs using MD5.

introduced secure_ipv4_port_ephemeral() with md5 hashing

2017

secure_seq: use SipHash in place of MD5

2021

tcp: change source port randomizarion at connect() time

table_perturb introduced for more randomization + fingerprint mitigation

2022

tcp: resalt the secret every 10 seconds

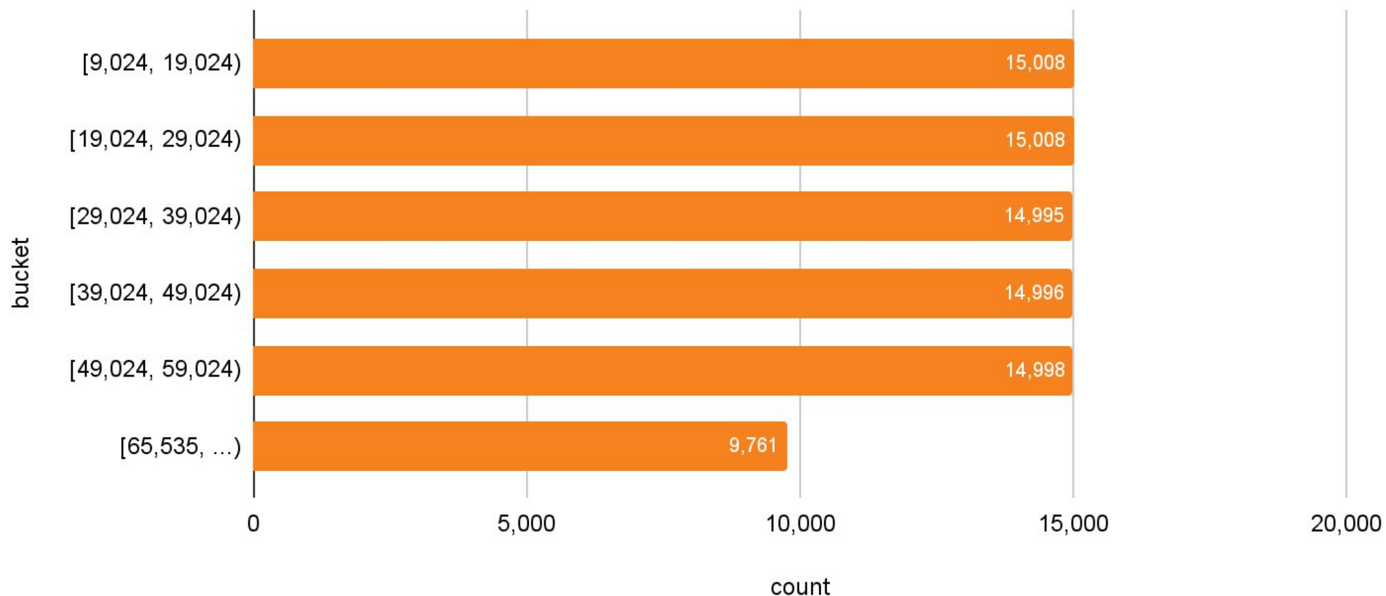
CLOUDFLARE

Are we starting our loop at the same offset each connect()?

```
#!/usr/bin/env bpftrace
/*
 // net/ipv4/inet_hashtables.c:__inet_hash_connect
 <+211>:    and    edx,0xfffffffe              // offset &= ~1U;
 <+214>:    mov    DWORD PTR [rsp+0x28],edx
 // other_parity_scan:
 <+218>:    add    r14d,DWORD PTR [rsp+0x28]   // port = low + offset;
 <+223>:    test   r13d,r13d
*/
kprobe:__inet_hash_connect+223 {
    $port = reg("r14");
    @port_buckets = lhist($port, 9024, 65535, 10000);
}
```

CLOUDFLARE

Are we starting our loop at the same offset each connect()?



CLOUDFLARE

__inet_hash_connect() overview: the loop

```
   offset &= ~1U;

other_parity_scan:
    port = low + offset;
    for (i = 0; i < remaining; i += 2, port += 2) {
        if (unlikely(port >= high))
            port -= remaining;

        inet_bind_bucket_for_each(tb, &head->chain) {
            if (inet_bind_bucket_match(tb, net, port, l3mdev)) {
                if (!check_established(death_row, sk,
                                       port, &tw))
                    goto ok;
                goto next_port;
            }
        }
    }

    offset++;
    if ((offset & 1) && remaining > 1)
        goto other_parity_scan;
```

- Check if the socket is unique
- check_established() == __inet_check_established()

CLOUDFLARE

Is __inet_check_established() a problem?

- Tested benchmarks on a quiet virtual machine
- No other connections were established for the same src/dest ip + dest port
- Therefore, negligible impact
- Bind buckets will fill up eventually!

The quantum state of a TCP port

CLOUDFLARE

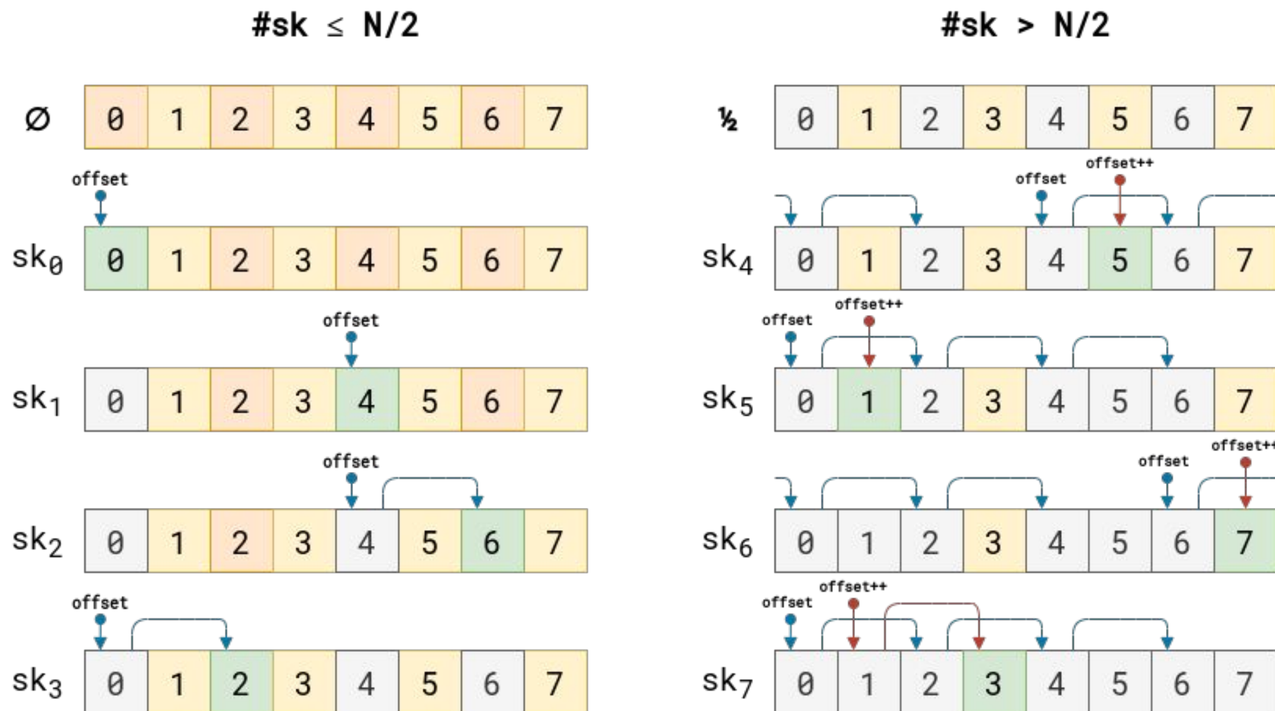
__inet_hash_connect() overview: the loop

```
    offset &= ~1U;

other_parity_scan:
    port = low + offset;
    for (i = 0; i < remaining; i += 2, port += 2) {
        if (unlikely(port >= high))
            port -= remaining;

        inet_bind_bucket_for_each(tb, &head->chain) {
            if (inet_bind_bucket_match(tb, net, port, l3mdev)) {
                if (!check_established(death_row, sk,
                                       port, &tw))
                    goto ok;
                goto next_port;
            }
        }
    }

    offset++;
    if ((offset & 1) && remaining > 1)
        goto other_parity_scan;
```

- Loop through first half of the ephemeral range then second
- Every other port is tested in sequence

CLOUDFLARE

__inet_hash_connect() overview: the loop

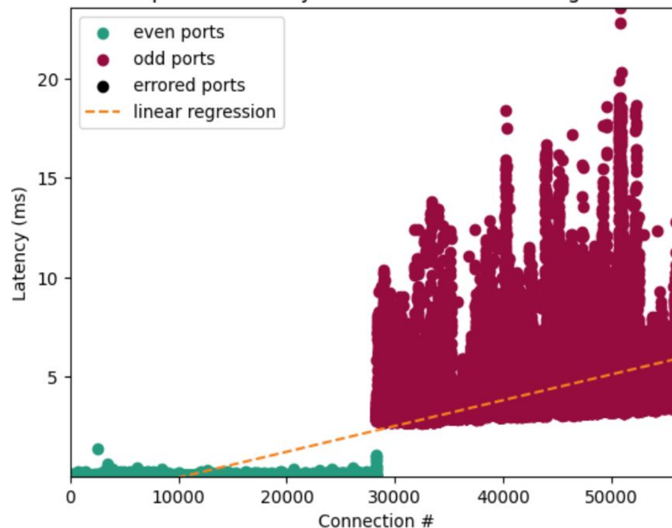


CLOUDFLARE

Is the loop the problem?

- Via experimentation
- Counted the even ports green, odd ports red
- Our port range dictates we always loop through even ports first

2.1 min @ 56k connections



CLOUDFLARE

Tracking port parity switches

```
#!/usr/bin/env bpftrace

kretfunc:vmlinux:inet_hash_connect /retval == 0/ {
    $port = args->sk->__sk_common.skc_num;
    @procs[comm,cgroup] += $port & 1;
}
```

```
rate(connect_port_parity_switches_total[1m])
```

Prometheus exporter for eBPF metrics

CLOUDFLARE

Our conclusion

- Exhausting half the net.ipv4.ip_local_port_range is fast
- The port looping appears to be our primary bottleneck
- Evidenced by a previous attempt [PATCH] tcp: avoid unnecessary loop if even ports are used up and was not merged

What do?

CLOUDFLARE

CLOUDFLARE

Some feasible, but not viable solutions for our case

1. Split egress unicast connections over 2..N IP addresses
2. Introduce a sysctl to manipulate connect
3. Pick a random port in userspace, and bind() with that
4. Leverage the new IP_LOCAL_PORT_RANGE socket option (v6.3.y)*

CLOUDFLARE

Split egress unicast connections over 2..N IP addresses

- Leaks networking configuration to user space
- No ability to tell the interface to balance between assigned IP's or IP blocks
- Requires IP_BIND_ADDRESS_NO_PORT socket option + bind() before connect() pattern
- We do this strategy now, but want to reduce to 1 IP

CLOUDFLARE

Introduce a sysctl to manipulate connect

- Kernel modification
- [PATCH] tcp: avoid unnecessary loop if even ports are used up

CLOUDFLARE

Pick a random port in userspace, and bind() before connect()

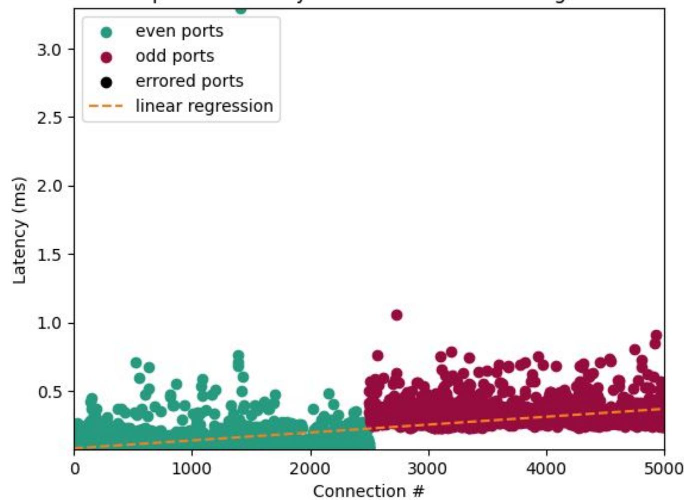
- Requires bind() before connect()
- Syscall overhead and ~8-12 attempts per connect closer to exhaustion
- Good up to ~70-80% port range utilization

```
sys = get_ip_local_port_range()
estab = 0
i = sys.hi
while i >= 0:
    if estab >= sys.hi:
        break

    random_port = random.randint(
        sys.lo, sys.hi)
    connection = attempt_connect(random_port)
    if connection is None:
        i += 1
        continue

    i -= 1
    estab += 1
```

Leverage the new IP_LOCAL_PORT_RANGE socket option (v6.3.y)

- Max # of connect() as range
- Pre-allocation of partitions to balance between
- Loop problem still persists

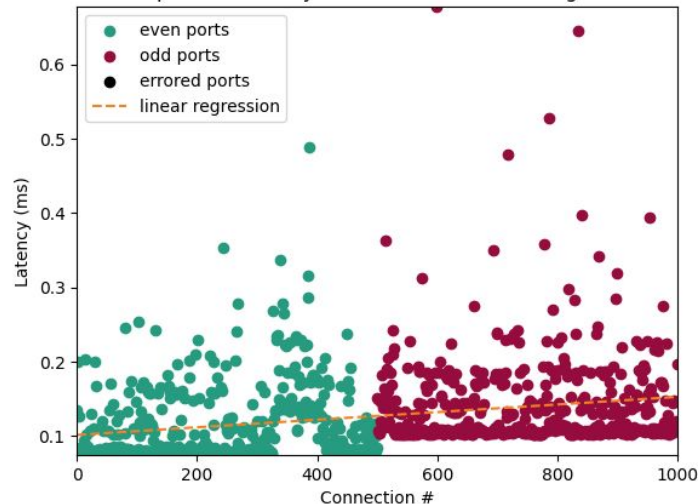
5k window @ 1.1 sec



Leverage the new IP_LOCAL_PORT_RANGE socket option (v6.3.y)

- Lower range works better
- Overlapping ranges is possible
- Overlap is determined by implementation

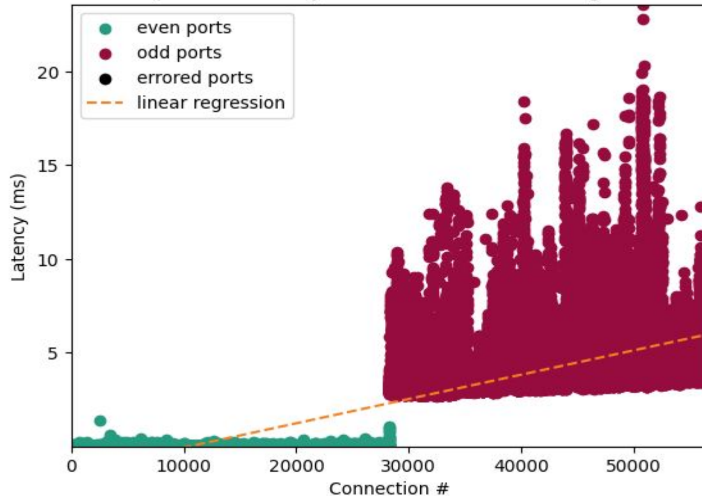
1k window @ 0.1 sec

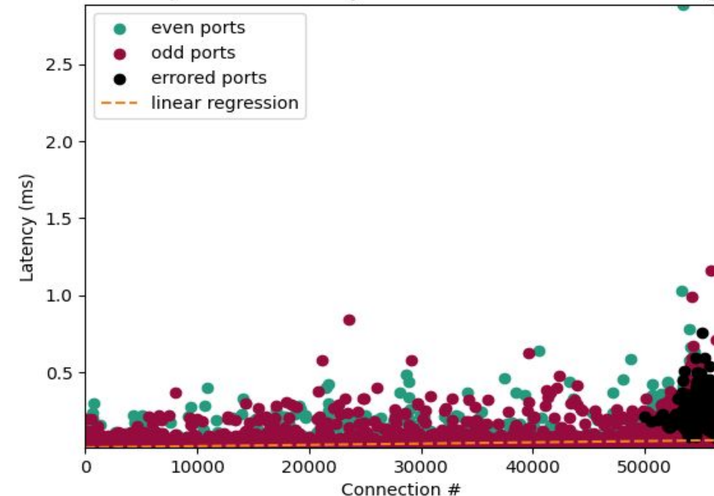


CLOUDFLARE

Leverage the new IP_LOCAL_PORT_RANGE socket option (v6.3.y) + random offset

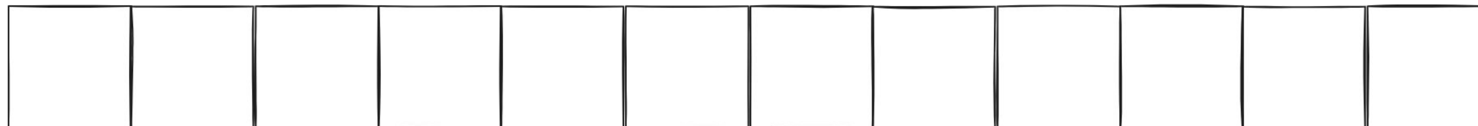
2.1 min → 1.8 sec!





Implementation details

sys.lo = 9024; sys.hi = 65535



CLOUDFLARE

Implementation details

```
window.lo = 0;  window.hi = 500
range = window.hi - window.lo
offset = randint(sys.lo, sys.hi - range)
window.lo = offset;  window.hi = offset + range
setsockopt(SOL_IP, IP_LOCAL_PORT_RANGE, window.lo | (window.hi << 16))
```

CLOUDFLARE

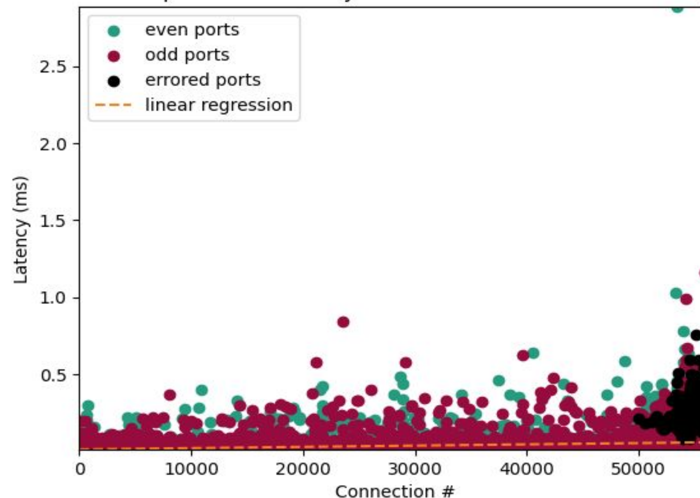
Implementation details

- Overlap is OK
- Reattempts may be necessary depending on use case
- Larger net.ipv4.ip_local_port_range is better with smaller selection window

In summary

- Leverages a random port offset + random low port in range to be even or odd
- Allows kernel to perform loop over a small + configurable local port range
- Overlaps windows on top of another

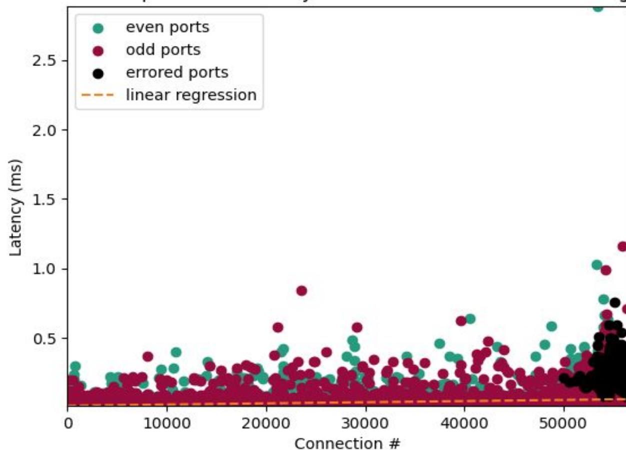
2.1 min → 1.8 sec @ 56k connections 500 window



Performance 56k unicast egress connections

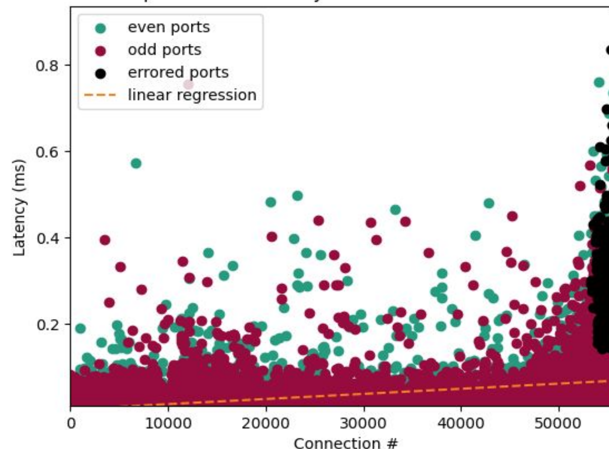
2.1 min → 1.8 sec

500 window



2.1 min → 2.0 sec

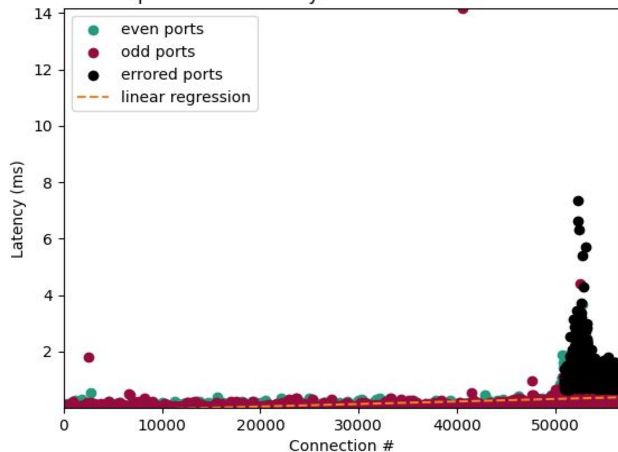
1000 window



Performance 56k unicast egress connections

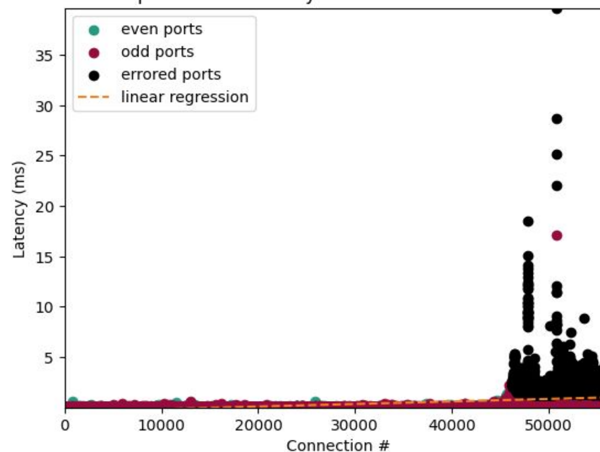
2.1 min → 6.7 sec

5k window



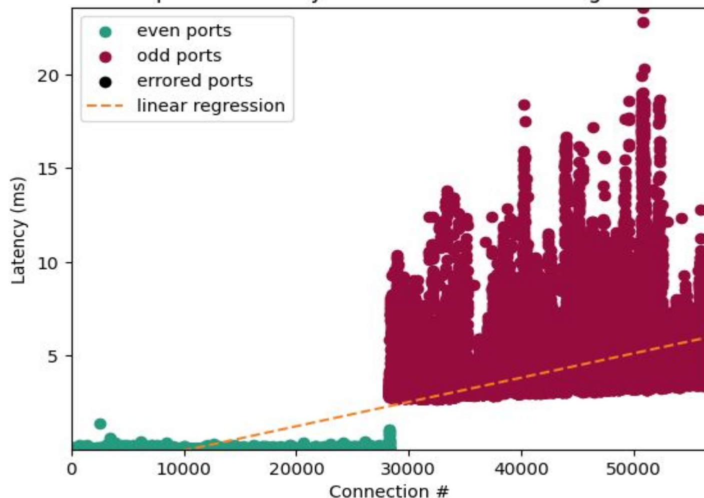
2.1 min → 17.7 sec

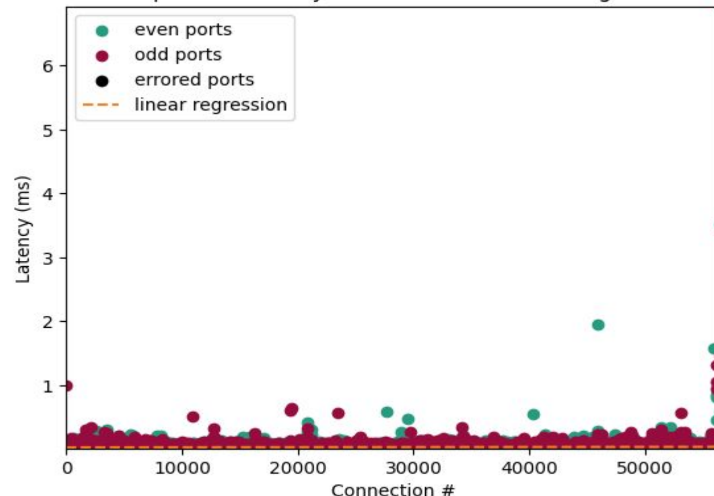
10k window



Leverage the new IP_LOCAL_PORT_RANGE socket option (v6.8.y)

2.1 min → 1.6 sec!



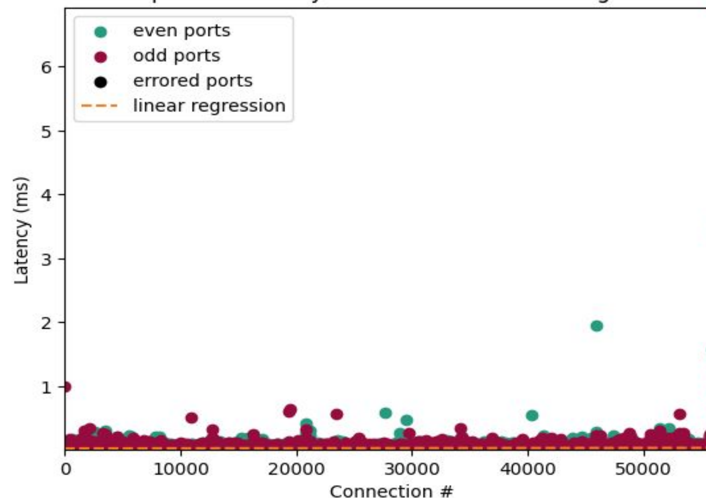


CLOUDFLARE

New in Linux 6.8.y

- Just requires IP_LOCAL_PORT_RANGE from userspace
- Faster performance with other 6.8.y features
- Guaranteed to find a port
- Patch: tcp/dccp: change source port selection at connect() time

2.1 min → 1.6 sec @ 56k connections



CLOUDFLARE

>= Linux 6.8.y implementation

```
IP_BIND_ADDRESS_NO_PORT = 24
IP_LOCAL_PORT_RANGE = 51
sys = get_local_port_range()
sk = socket(AF_INET, SOCK_STREAM)
sk.setsockopt(IPPROTO_IP, IP_BIND_ADDRESS_NO_PORT, 1)
range = pack("@I", sys.lo | (sys.hi << 16))
sk.setsockopt(IPPROTO_IP, IP_LOCAL_PORT_RANGE, range)
sk.bind((src_ip, 0))
sk.connect((dest_ip, dest_port))
```

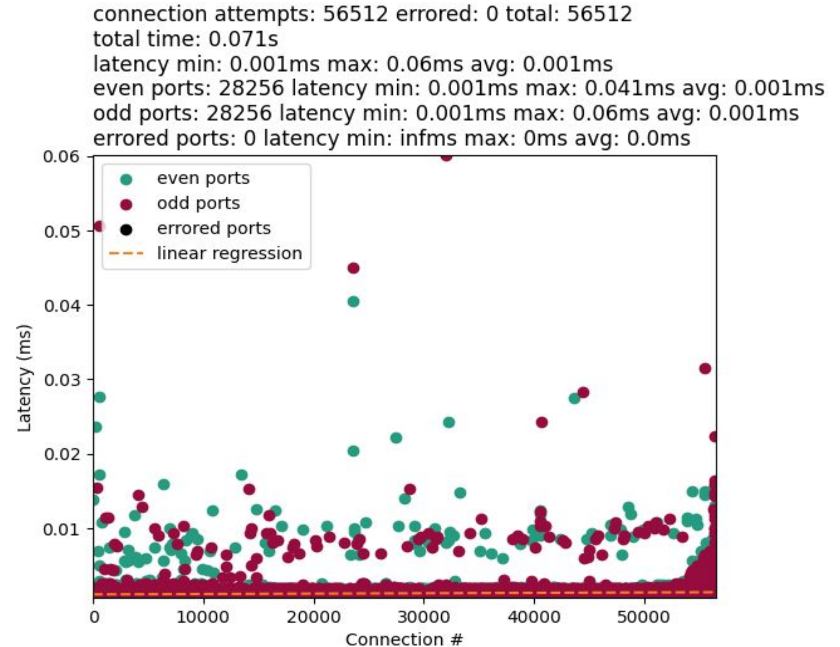
CLOUDFLARE

What about UDP?

CLOUDFLARE

Completely different algorithm!

- Still uses a tight loop
- Does not check one half of the range, then the next
- A port is randomly picked, a loop increments that port by a fixed-random number until integer overflow back to original port
- then, increment port by 1 and repeat until port is found



Takeaways

- Current implementation guarantees a port is selected
- Current implementation is not great at extreme egress workloads
- Random offset + 500-1k window coupled with kernel random port picking ensures we start looping at both odd and even ports with small-N
- Backport patches or update to at least 6.8.y
- Purely user space implementation

CLOUDFLARE

Questions?

fred@cloudflare.com

connect() - why are you so slow?

Minimal code that generated the charts