

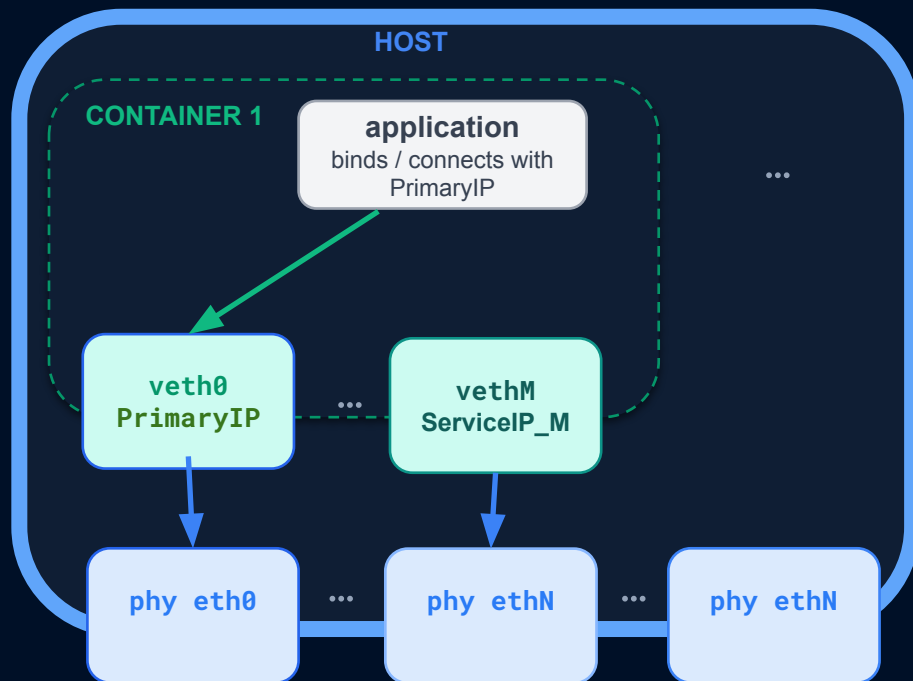
TCP Multipath implementation with BPF

Takshak Chahande, Sargun Dhillon, Jakub Kicinski, Raman Shukhau

Agenda

- Containers with multiple NICs
- Ingress/Egress
- Public vs On-Prem setup
- Solutions for On-Prem
- Multipath with BPF
- Pitfalls and learnings
- Demo

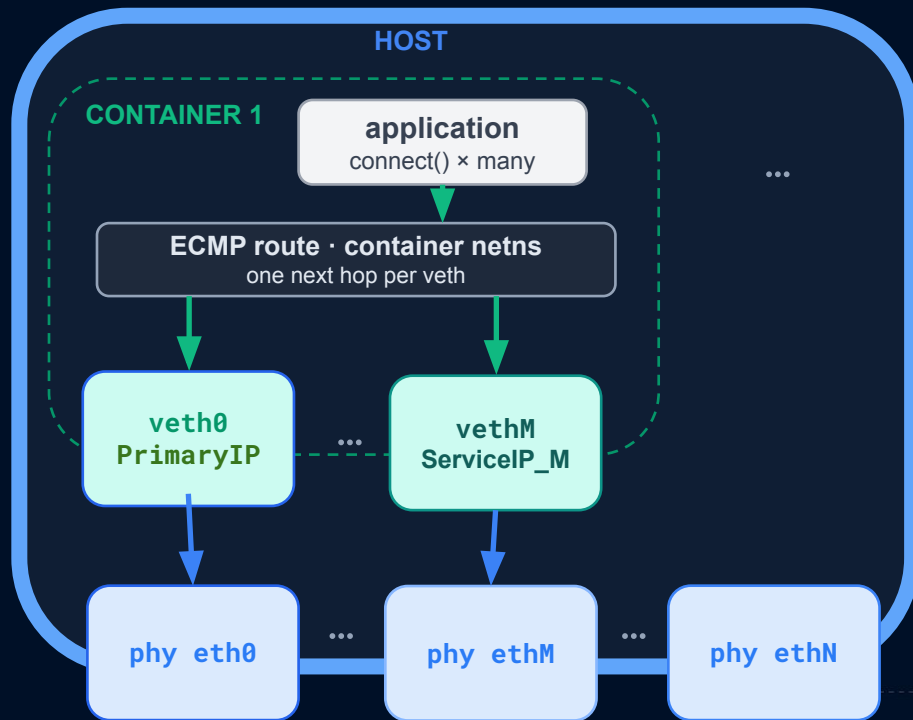
Container with multiple NICs



PrimaryIP is allocated from eth0's subnet;
every other NIC gets its own ServiceIP

- A GPU host has N NICs — separate subnets, separate addresses, ~100 Gb/s each
- Container dynamically allocates M(1 to N) NICs on the host.
- Connections should be distributed evenly across allocated NICs
- Throughput goes from one NIC to the whole box:
 $100 \text{ Gb/s} \rightarrow \sim(M \times 100) \text{ Gb/s}$
- Applications/Services want to use single IP
- No application changes necessary

Multipath: Egress



outbound flows hash across every NIC
decided inside the host; no fabric help needed

What the host does

one ECMP route in the container netns

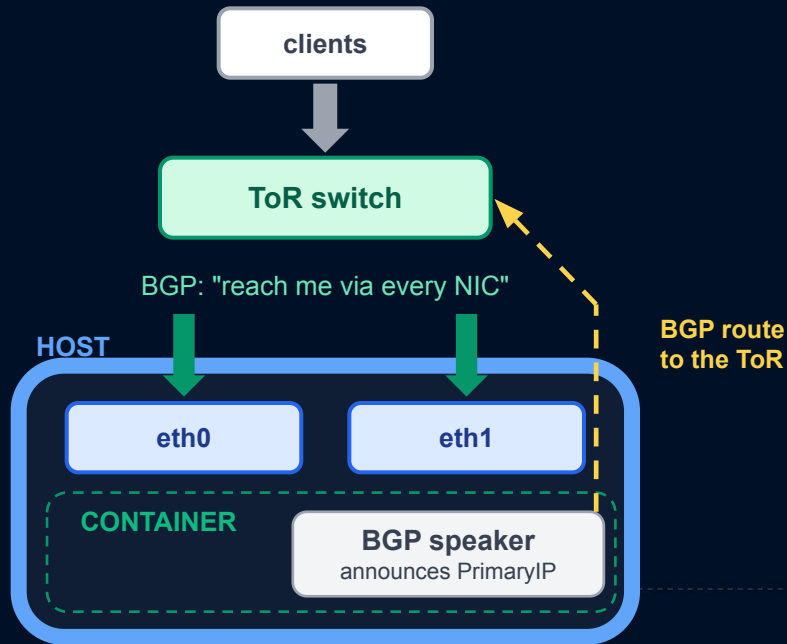
```
default src PrimaryIP
  nexthop via <gw> dev veth0
...
  nexthop via <gw> dev vethM
```

5-tuple hash:

protocol, src/dst IP and port
each connection uses one NIC

On-prem: Ingress

ON-PREM — this works today



HOW WE DO IT TODAY

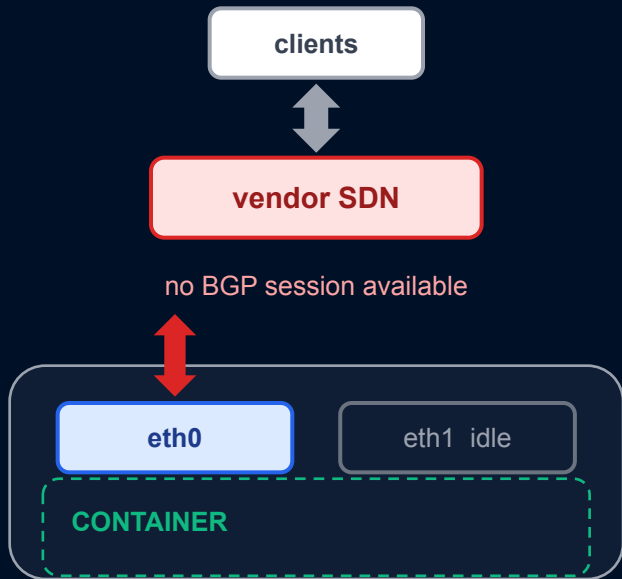
ingress — BGP Injection

starts inside the container and opens a BGP session with the rack switch: "reach my PrimaryIP via any of my NICs". The ToR installs an ECMP route and spreads inbound flows across them. Route is not propagated beyond ToR.

one PrimaryIP, both directions, application untouched

Public cloud: Ingress half breaks

PUBLIC CLOUD — this doesn't work



WHAT WE LOSE

nothing to peer with

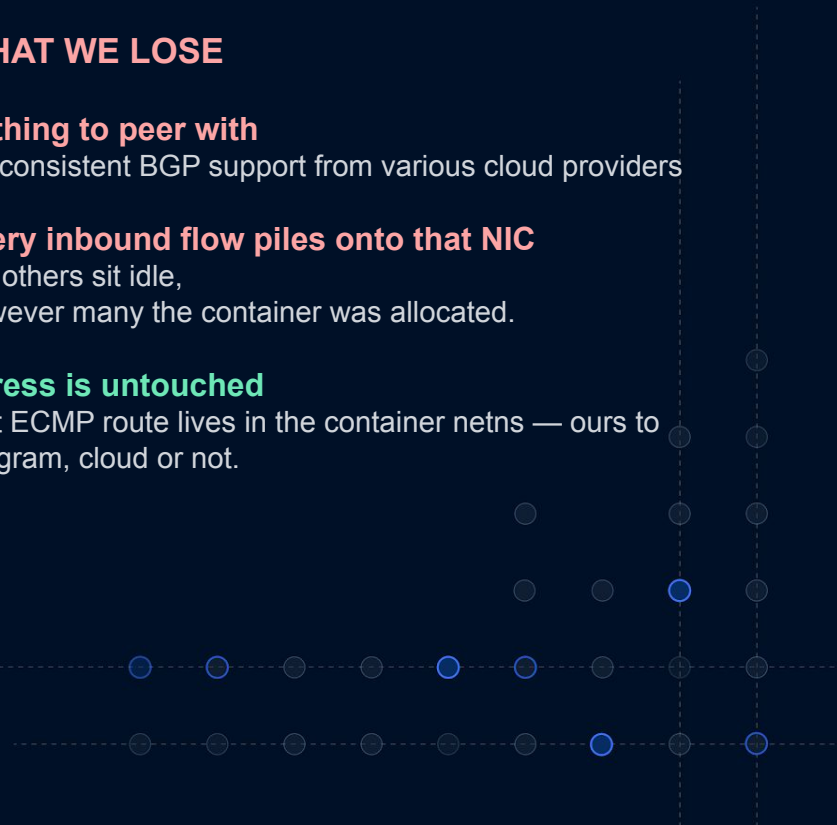
No consistent BGP support from various cloud providers

every inbound flow piles onto that NIC

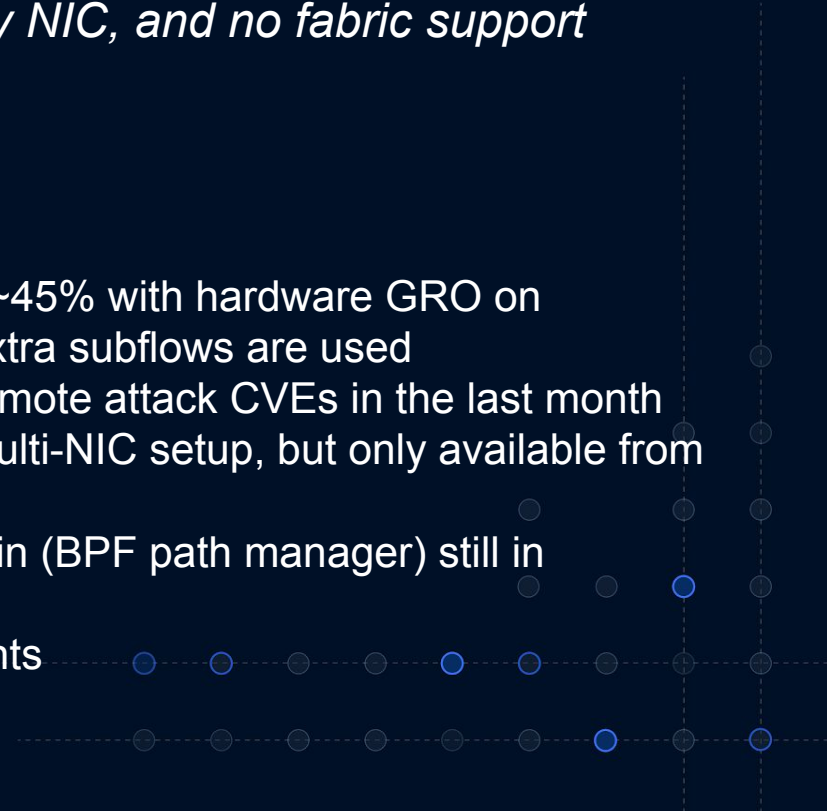
the others sit idle,
however many the container was allocated.

egress is untouched

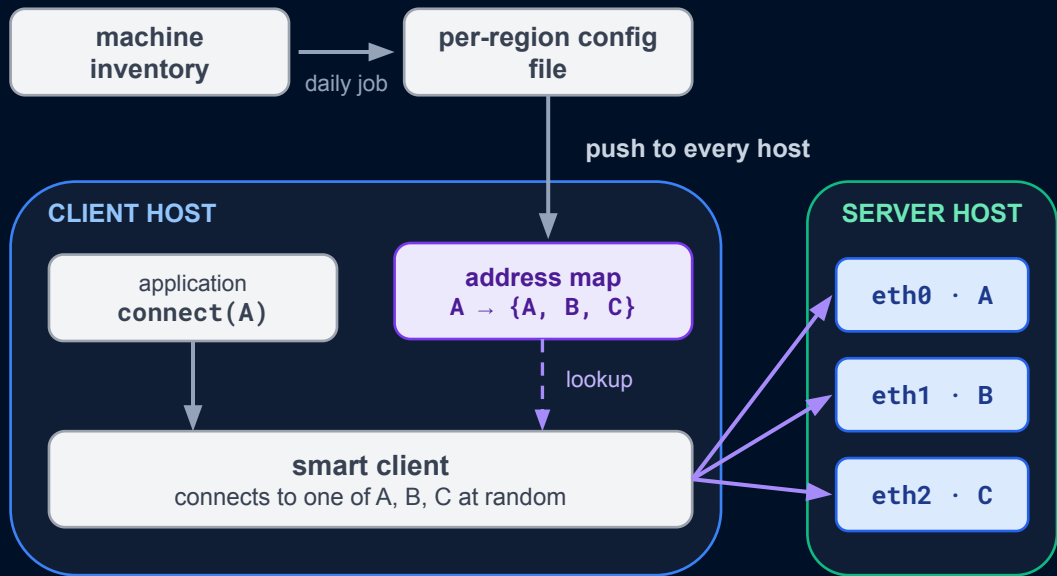
that ECMP route lives in the container netns — ours to
program, cloud or not.



First option: MPTCP

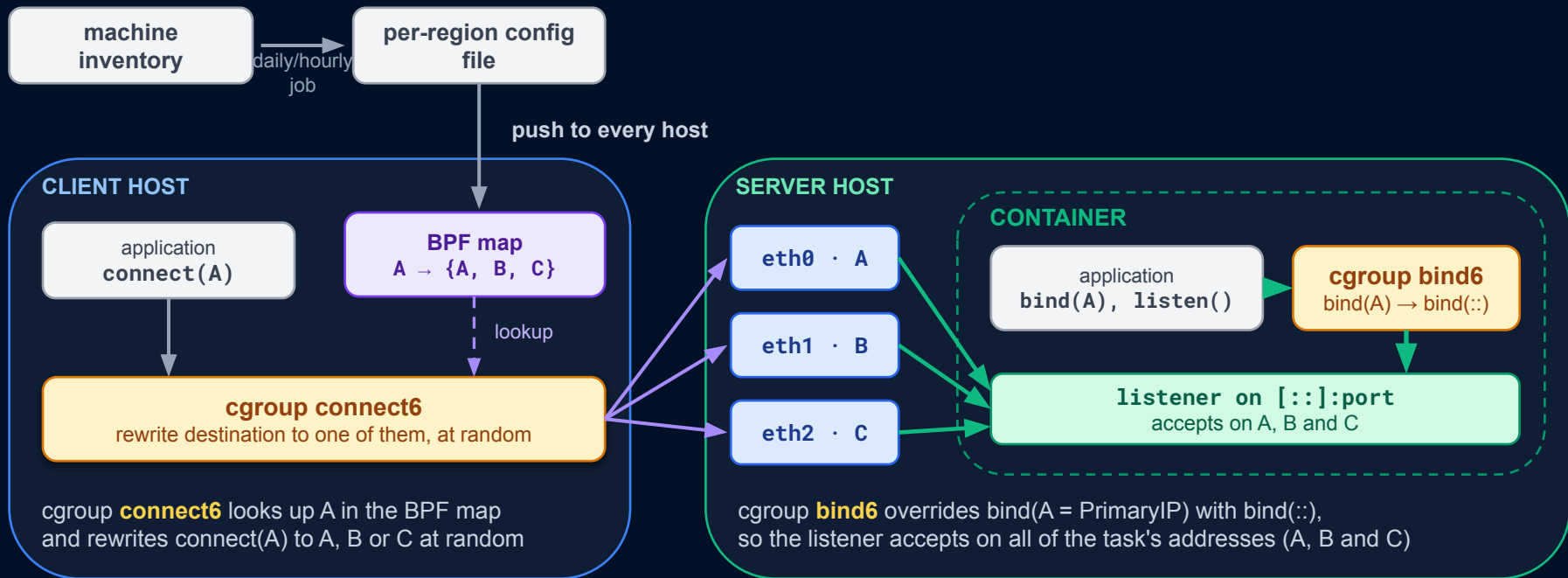
- *Real per-flow aggregation over every NIC, and no fabric support needed*
 - Kernel support 6.9+
 - Observed Issues
 - A single connection is slower up to ~45% with hardware GRO on
 - High number of retransmits when extra subflows are used
 - Security concern, with 2 potential remote attack CVEs in the last month
 - laminar endpoints are needed for multi-NIC setup, but only available from 6.18+
 - NUMA awareness with mptcpd plugin (BPF path manager) still in development
 - Hard limit of 8 subflows and endpoints
- 

Proposal: Multipath BPF



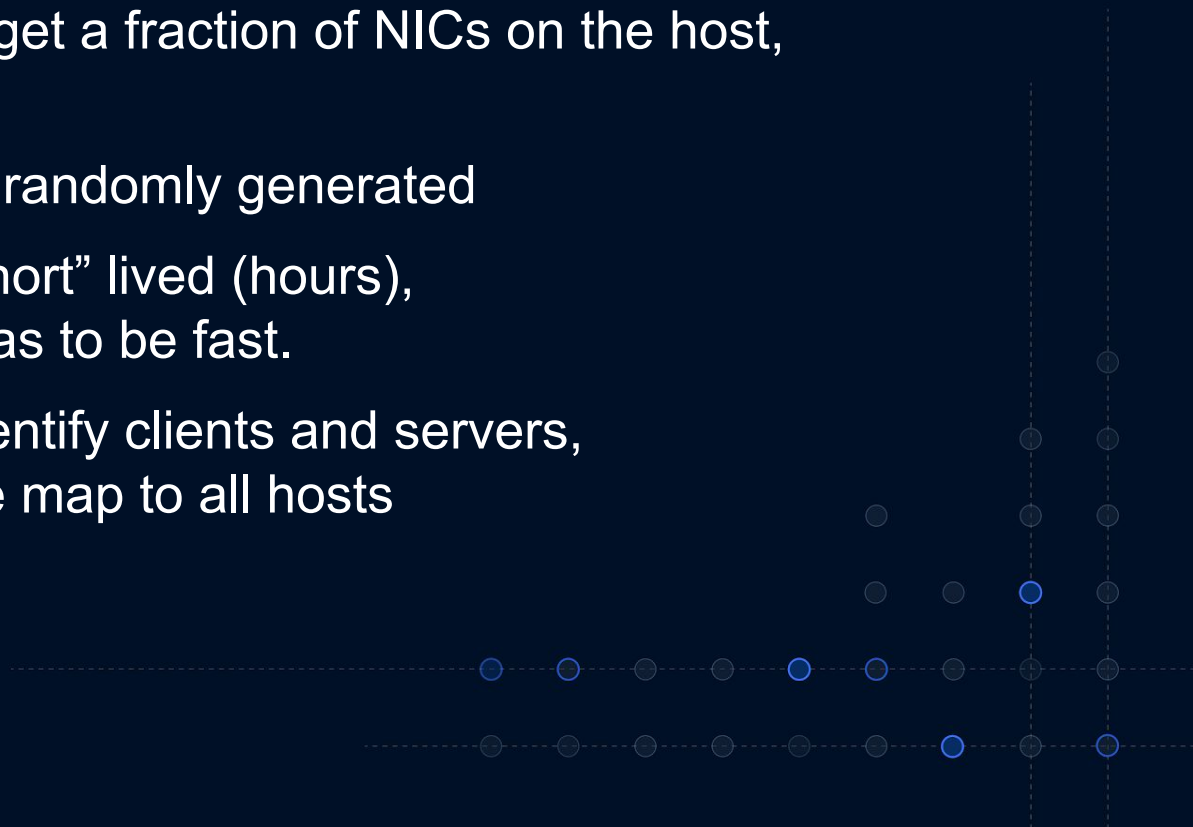
- A smart client: it knows the server host has multiple NICs/IPs
 - Picks one of them at random for each connection; the application still calls `connect(A)`
- Initial plan: push a map of every multi-NIC host's addresses to every host
 - Daily/Hourly job: machine inventory -> per-region config file -> address map on every host
 - A full-fleet map is ~50 MB

Multipath BPF: Hooks



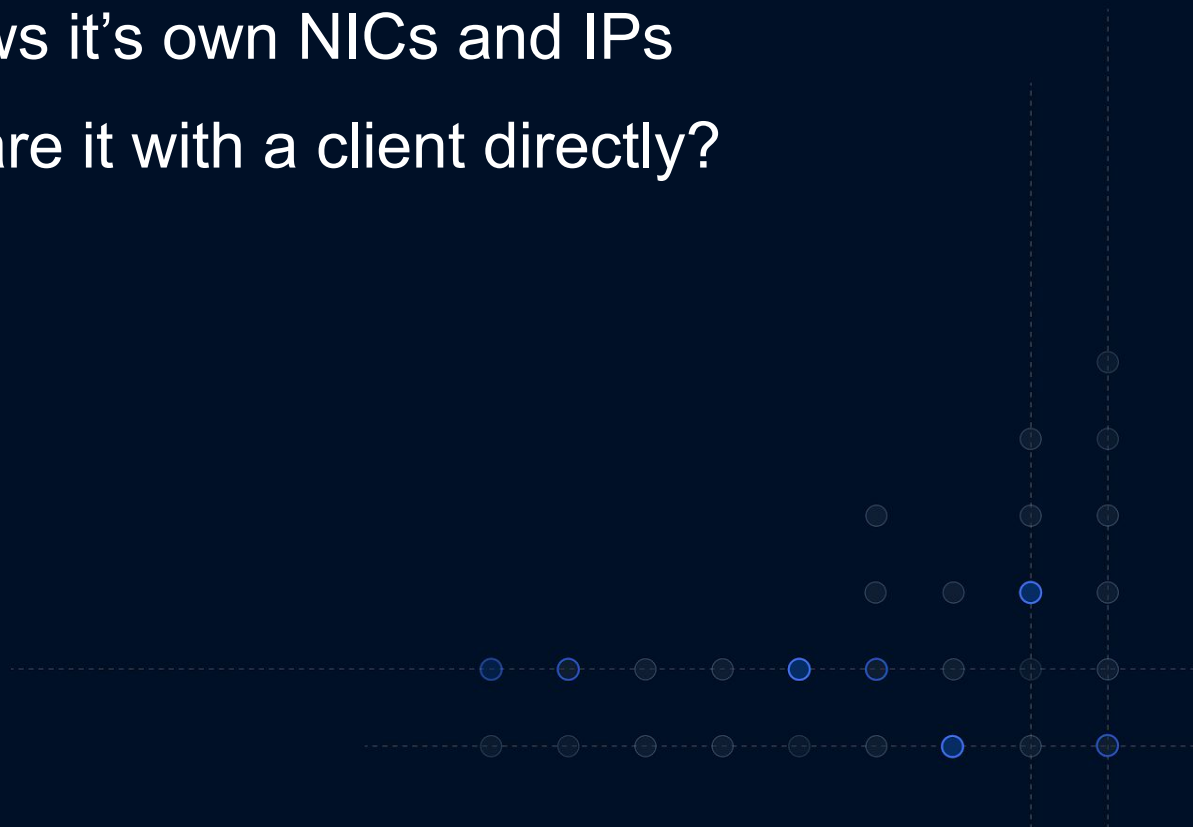
Multipath BPF: Issues

- Containers might get a fraction of NICs on the host, e.g. 2 out of N.
- Container IPs are randomly generated
- Containers are “short” lived (hours), synchronisation has to be fast.
- Not possible to identify clients and servers, need to propagate map to all hosts



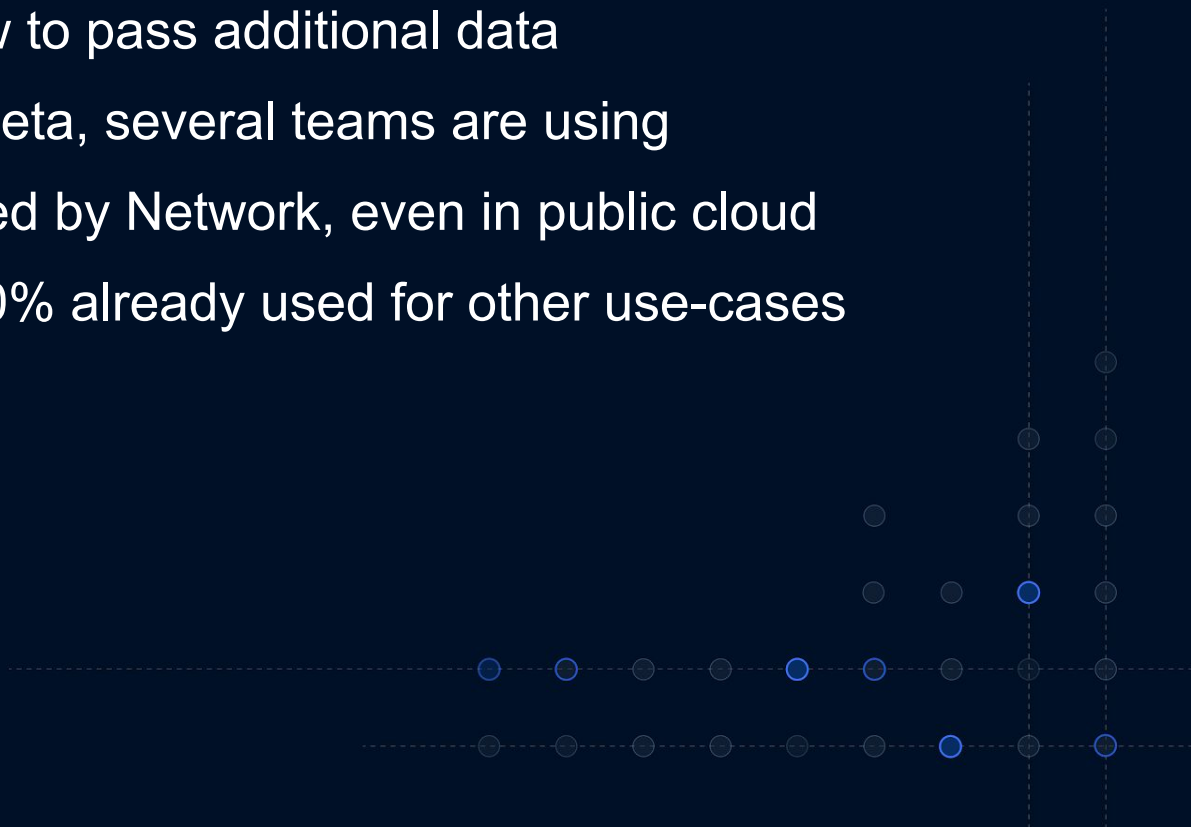
Multipath BPF: Ask directly

- Container knows it's own NICs and IPs
- Can server share it with a client directly?



Multipath BPF: TCP Options

- TCP Options allow to pass additional data
- Already used at Meta, several teams are using
- Not filtered/dropped by Network, even in public cloud
- 40 bytes max, ~50% already used for other use-cases



Multipath BPF: TCP Options format

TCP gives you 40 bytes of option space — for every option, together

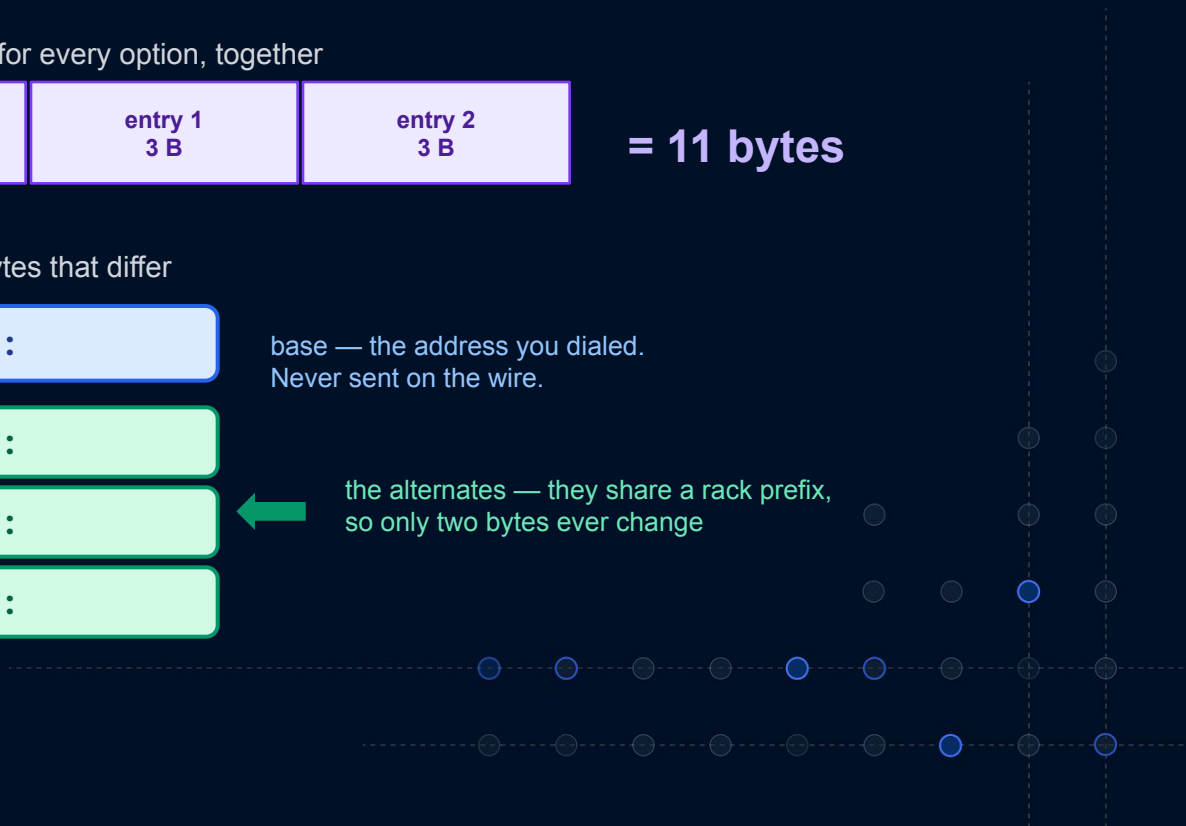


each entry = one position byte + only the bytes that differ

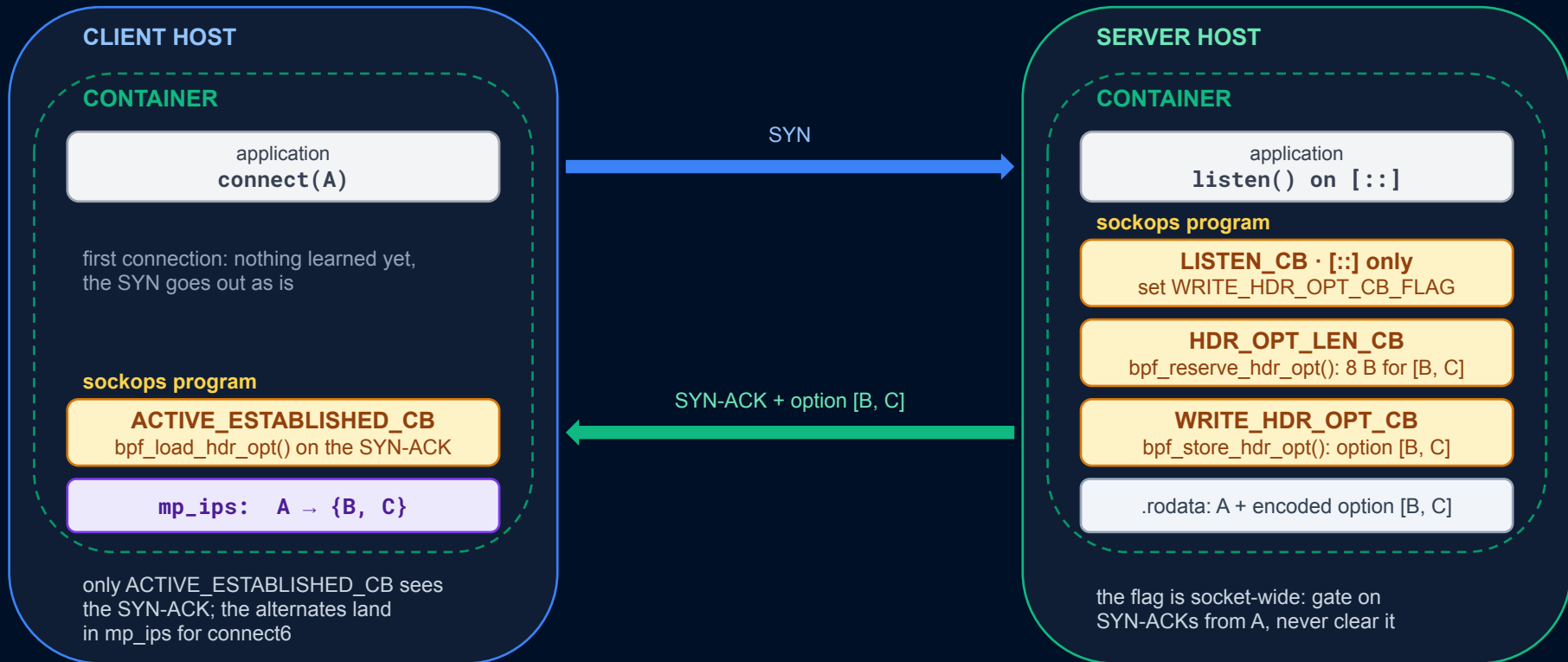


base — the address you dialed.
Never sent on the wire.

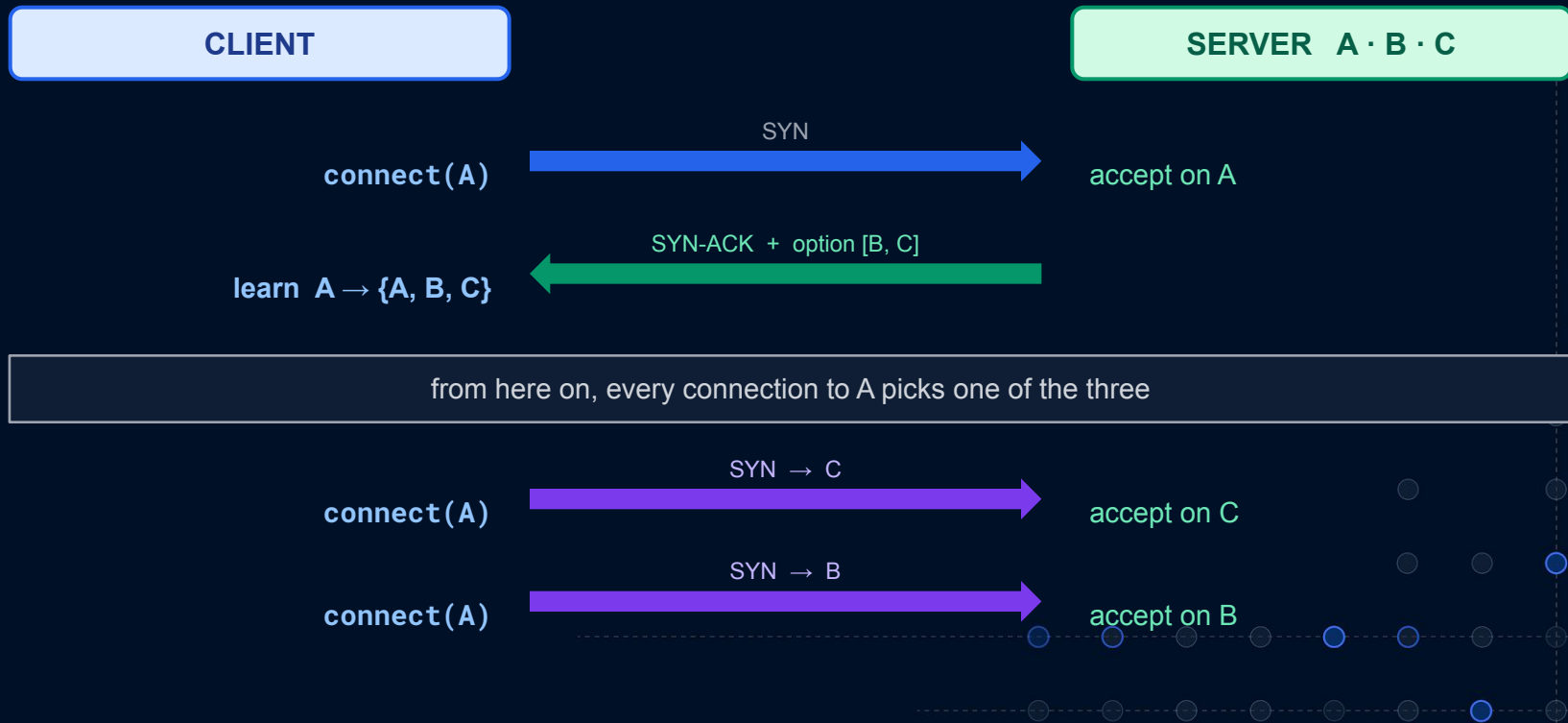
the alternates — they share a rack prefix,
so only two bytes ever change



Multipath BPF: TCP option hooks

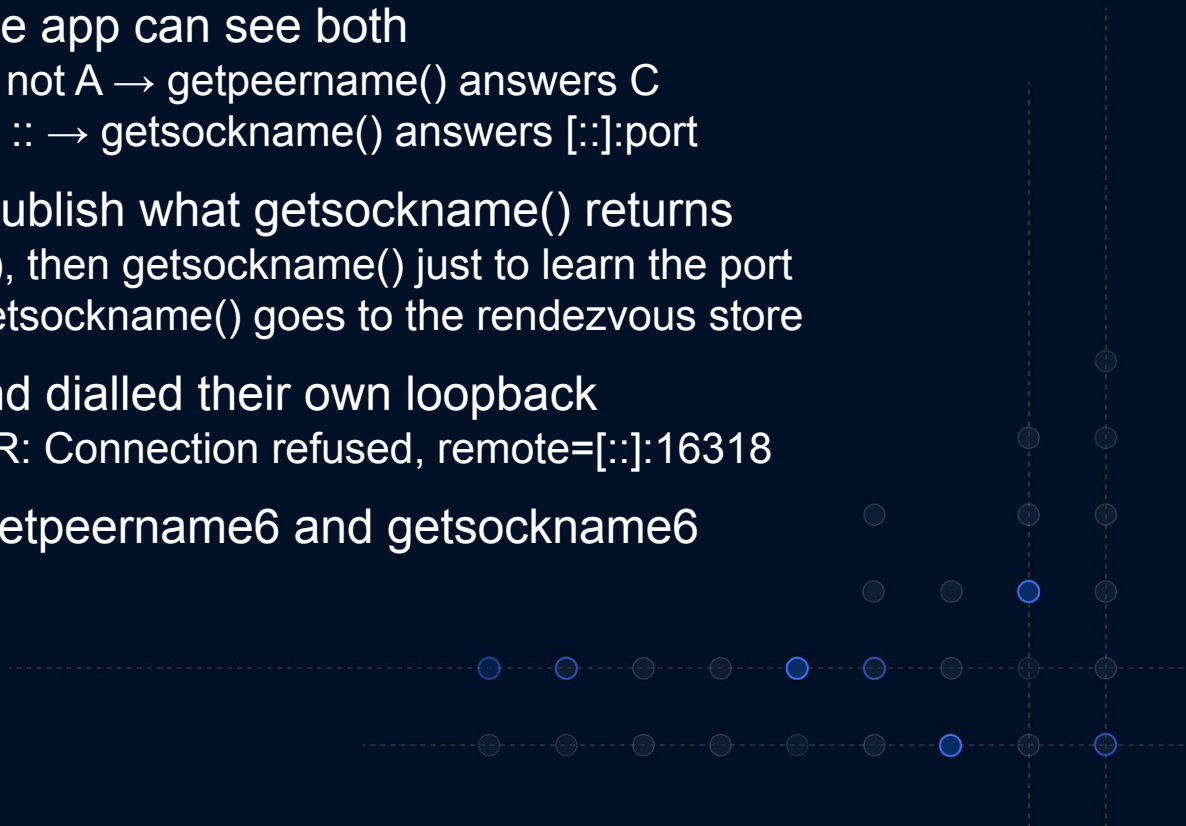


Multipath BPF: Connection flow

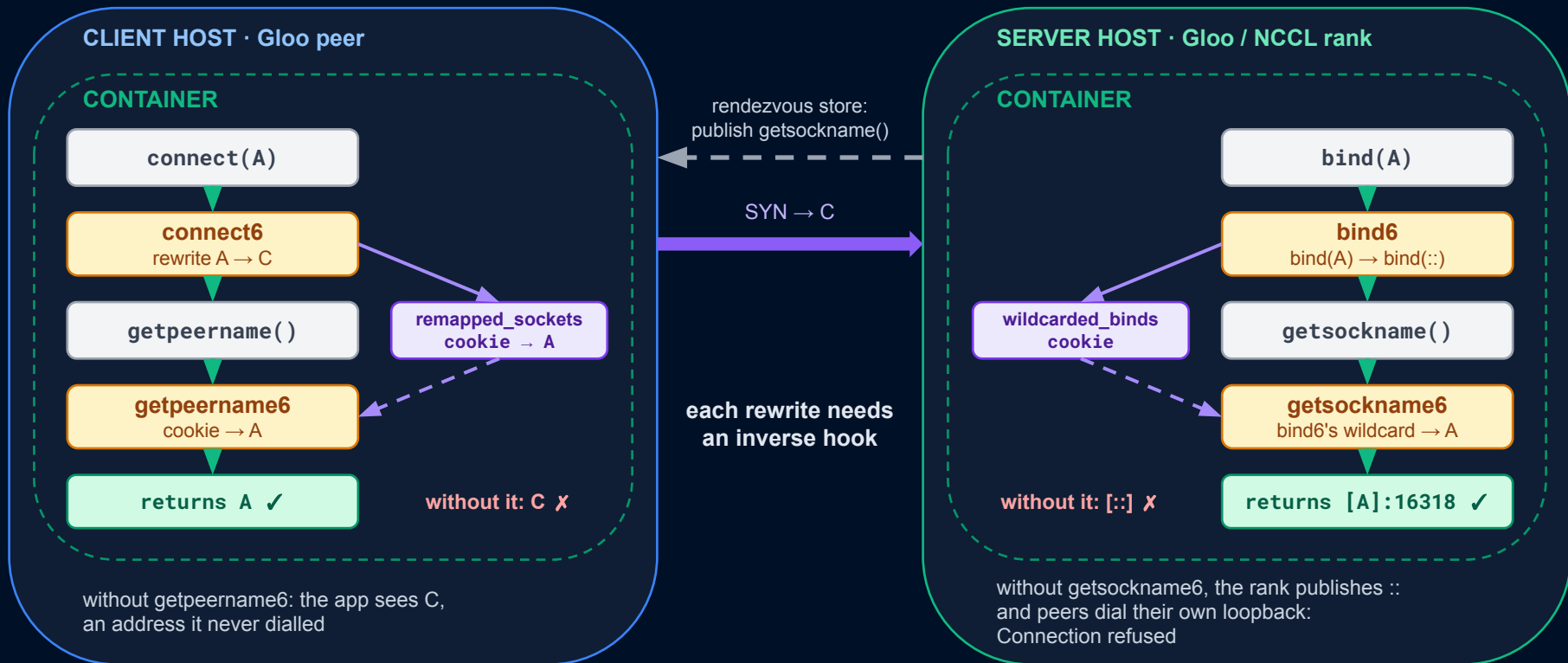


Multipath BPF: a rewrite you can see

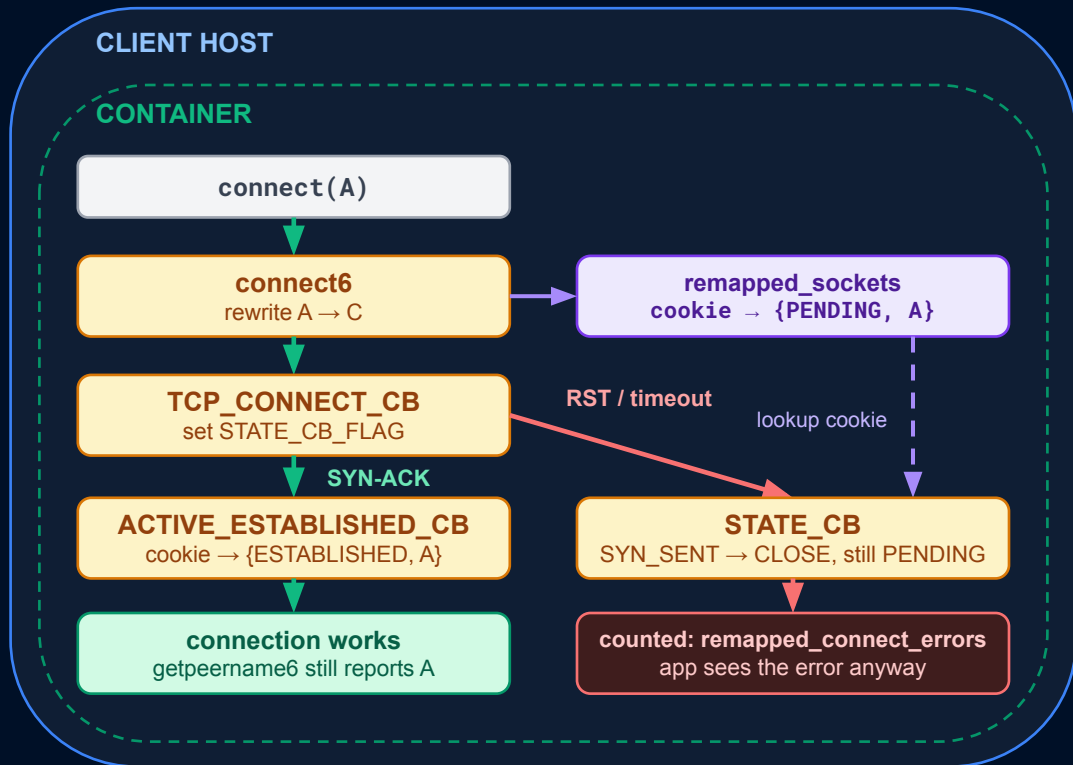
- Two rewrites, and the app can see both
 - connect6 dials C, not A → getpeername() answers C
 - bind6 widens A to :: → getsockname() answers [::]:port
- Collective libraries publish what getsockname() returns
 - NCCL: bind([A]:0), then getsockname() just to learn the port
 - Gloo: listener's getsockname() goes to the rendezvous store
- Peers got [::]:port and dialled their own loopback
 - Gloo: SO_ERROR: Connection refused, remote=[::]:16318
- Fix: inverse hooks getpeername6 and getsockname6



Multipath BPF: **getpeername** & **getsockname**

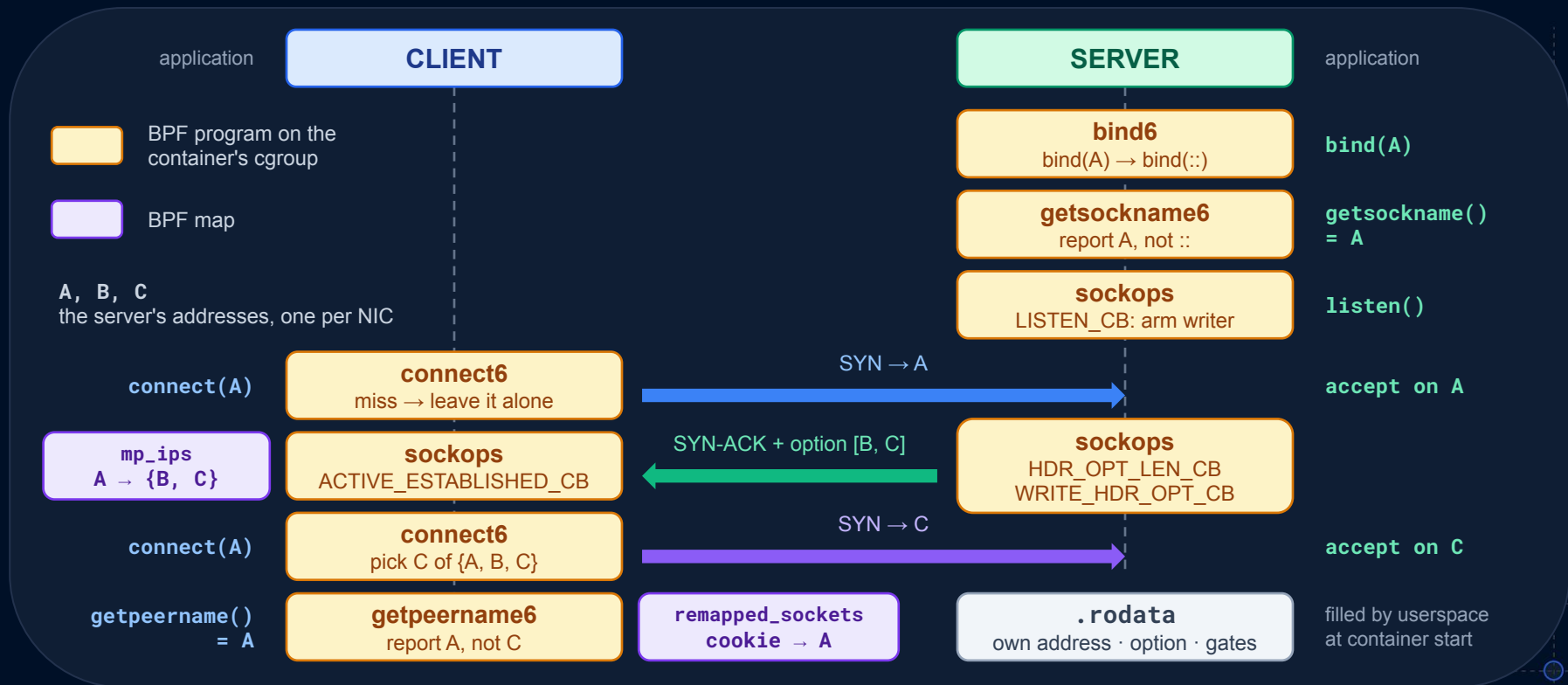


Multipath BPF: Error metrics



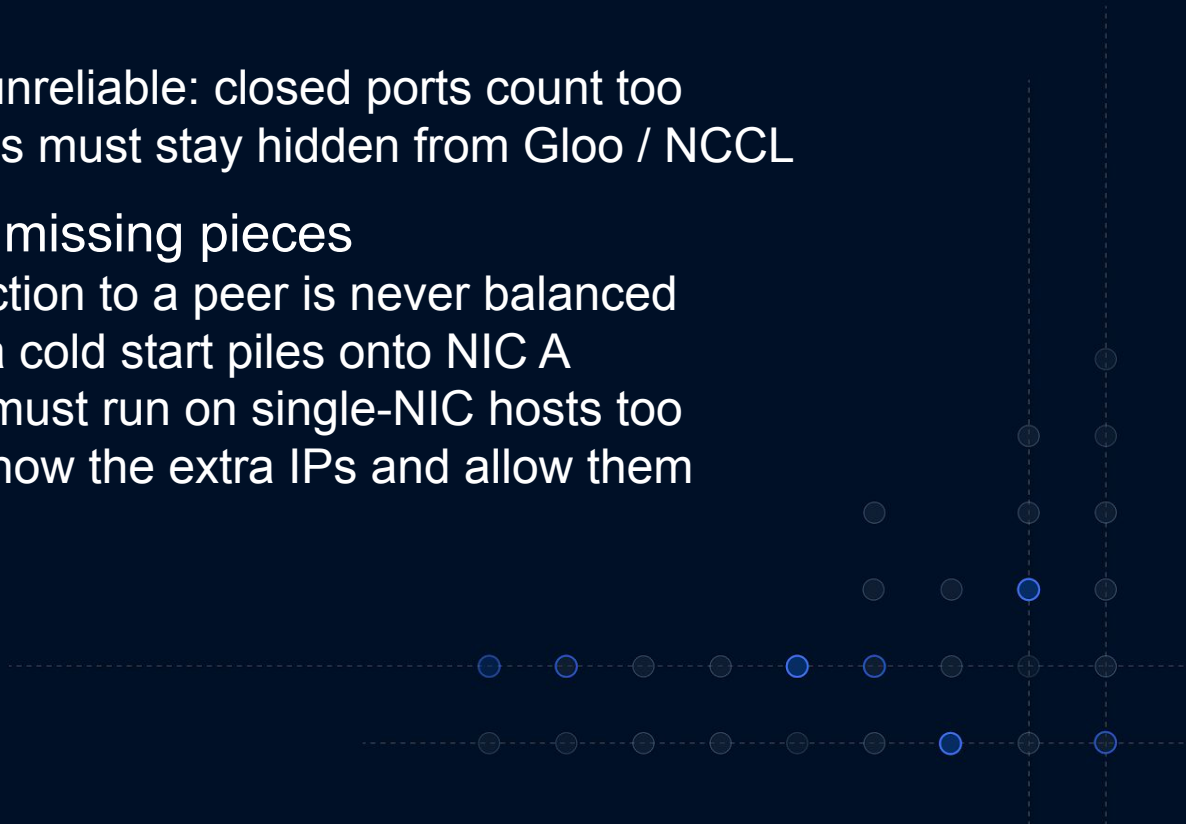
- Tag: `connect6` stores cookie → `{PENDING, A}`
- Wait: `STATE_CB` fires on each state change
- SYN-ACK: ESTABLISHED, remap worked
- `SYN_SENT` → `CLOSE` while `PENDING`: never connected
- Counted, not fixed: `connect()` has no undo

The v2 proposal: all BPF hooks



Learnings

- BPF
 - Error metric is unreliable: closed ports count too
 - Address rewrites must stay hidden from Gloo / NCCL
- Other: issues and missing pieces
 - The first connection to a peer is never balanced
 - Flock of birds: a cold start piles onto NIC A
 - The client part must run on single-NIC hosts too
 - Firewall must know the extra IPs and allow them



Demo



Q&A

Thank you

