Operating System-Level Load Distribution for Network Telemetry Data Collection

Eduard Bachmakov
2021-11 @ IETF112 grow
Traditional networks maintain distributed state across many devices and applications. Observability requires joining relevant bits of state across devices and from different perspectives.

“perspectives”? control plane, forwarding plane, device topology

“relevant”? obtain via specialized network telemetry protocols
Obtaining network telemetry

Device configuration is hard, fragile, and comes with a long tail in actuation. Preferred solution: globally consistent endpoint address.

Usable for load balancing!

1. Anycast for regional routing
2. ECMP for flow balancing
3. On-host balancing across processes
Background
Goal & setup

Setting: network with some number of routers configured to push network telemetry to a specific host.

Goal:

- combining control plane and forwarding plane information, per source device
  - IPFIX & BMP (and more)
- collector daemon: *pmacct*’s *nfacctd*
- (later steps: message broker → storage system → OLAP queries)
To combine the data we need to ensure that both network telemetry data streams hit the same collector daemon.

Solution: proxies.

- *HAProxy* for TCP-based telemetry protocols (i.e. BMP)
- *nfacctd* for UDP-based ones (i.e. IPFIX)
Data flow routing: visualization
Data flow routing: issues

- Reliability
- Overhead
- Large configuration space
Design
Multiple processes bind to the same IP:port combination, forming a “reuseport group”.

During local delivery of TCP/UDP packets, on socket lookup,
  - all *new* TCP connections and
  - all UDP datagrams are distributed among the reuseport group.

socket assignment determined by hash of 5-tuple

Good start, however, uncontrolled assignment is not acceptable …

… but we can influence this using eBPF!
eBPF is a Linux kernel subsystem allowing users to attach custom logic at specific hooks at runtime.

Properties:

- virtual machine with restricted instruction set optimized for JIT-compilation
- limited functionality (not Turing-complete)
- on-load safety/security verification
- well defined interface to userspace ("eBPF maps")
- not a kernel modification, not a kernel module
- public API/ABI → stability

One of the available hooks is right at the SO_REUSEPORT socket selection!
Now the pieces are in place. We can now

1. register all participating collector daemons in a lookup table (i.e. a special eBPF map),
2. create an algorithm to find the appropriate socket for a given, incoming packet,
3. bundle that logic in a “SK_REUSEPORT” eBPF program, and
4. attach a SK_REUSEPORT eBPF program to the reuseport group.
We have now achieved balancing that is

- stateless,
- based on device identity only,
- stable across restarts,
- prevents cascading failures,
- requires virtually no configuration.
Conclusion
Network telemetry aggregation has additional, special requirements for load balancing.

We designed and implemented a system addressing these requirements using in-kernel loadbalancing via eBPF.

Balancing is device scoped and cross-protocol balancing across an arbitrary number of collection endpoints.

Enables reliable network monitoring by ensuring robust correlation of network telemetry data at the collection endpoint.

Requires less maintenance effort and much less configuration overhead than the previously existing architecture.

Evaluation shows more efficiently than the previously existing system.
Questions? Answers!

Same stuff—many, many more words: doi.org/10.3929/ethz-b-000507440
Backup
Resource usage: CPU

Proxy-based

BPF-based

CPU usage (ms/s)

Time (normalized)
Resource usage: memory

Proxy-based

BPF-based

Usage (GB)

Time (normalized)