enfabrica

Zero / Low Cost Counters and Linux TCP for ML

David Ahern

September 2023

Zero (or Extremely Low) Cost Counters for Userpace Monitoring

Goal

enfabrica

Fast, low overhead, end-to-end metrics to userspace

- Hardware to networking stack and back (ie., Rx and Tx)
- Lockless, no atomics, no memory barriers, no copies
- Always enabled counters
- Constant overhead for a counter irrespective of number of readers
  - e.g., same overhead for 10 readers as with 1 reader
- Inconsistency in a single read is ok with eventual consistency
  - e.g, byte counter and packet counter out of sync on a given read

Flexible Pull of Statistics

- per flow
- per queue
- per device
- in any combination

Stats from networking stack, vendor unique stats in driver, vendor unique stats in h/w

Current APIs for Networking Stats

enfabrica

ethtool -S (or directly invoking ethtool uAPI)

- Vendor / driver unique stats (e.g., per queue, hardware)
- All or nothing scheme (ie., all stats tracked by driver are returned)
- Overhead on every read: rtnl, generating response, copying to userspace, parsing in userspace

ip -s

- Standard device tracked stats (rtnl_link_stats{,64})
- Overhead: same as ethtool

/proc, /sysfs files

- Overhead: system call, data rendering (snprintf), copy to userspace, parsing in userspace

Outline of Infrastructure

enfabrica

Extend netdev-genl or add new stats focused genl

QUERY command to get list of supported counters for specific kernel + driver + H/W

- Returns name and description for each counter group
- Returns name, description, datatype, and optional collection arguments for each counter

REGISTER command for userspace to opt-in to counters using a descriptive name

- TCP socket: tcp/<socket-id>
- L3 stats: ip, ip6
- core stats: core/page_pool
- qdisc stats:
- Driver stats: <netdev>/queue/rx/<id>
- H/W stats: <netdev>/hwqueue/tx/<id>/<collection options>

Names and collection options map to nested attributes

Outline of Infrastructure, cont'd

enfabrica

Counters are mapped to process as read-only page(s)

Successful response returns:

- Base address of read-only mapping
- For each counter: string with a name, data type enum, offset within the mapping

Single counter instance in kernel is observable by many processes - ie., constant cost

- Counters are updated once by kernel
- Counters are read with zero system cost

UNREGISTER command to remove mapping

Notifications to userspace if memory is unmapped by kernel operation

- e.g., monitored socket is closed, netdevice is deleted
- Race here, so may have to convert mapping to zero page to avoid faults

Stat Groups

Existing networking stack statistics moved to a separate page

- All stats in the page are related and mapped as a group
- socket stats, IP, core networking layer, qdisc
- page_pool / generic buffer pool stats

Core code has direct access to stats handlers exported by each layer

Example:

- TCP stats in tcp_sock

```
struct tcp_stats {
        u64     bytes_received; /* RFC4898 tcpEStatsAppHCThruOctetsReceived
                                 * sum(delta(rcv_nxt)), or how many bytes
                                 * were acked.
                                 */
        u32     segs_in;        /* RFC4898 tcpEStatsPerfSegsIn
                                 * total number of segments in.
                                 */
        u32     data_segs_in;   /* RFC4898 tcpEStatsPerfDataSegsIn
                                 * total number of data segments in.
                                 */
        u32     rcv_nxt;        /* What we want to receive next         */
        u32     copied_seq;     /* Head of yet unread data              */
        u32     rcv_wup;        /* rcv_nxt on last window update sent   */
        u32     snd_nxt;        /* Next sequence we send                */
        u32     segs_out;       /* RFC4898 tcpEStatsPerfSegsOut
                                 * The total number of segments sent.
                                 */
        u32     data_segs_out;  /* RFC4898 tcpEStatsPerfDataSegsOut
                                 * total number of data segments sent.
                                 */
        u64     bytes_sent;     /* RFC4898 tcpEStatsPerfHCDataOctetsOut
                                 * total number of data bytes sent.
                                 */
        u64     bytes_acked;    /* RFC4898 tcpEStatsAppHCThruOctetsAcked
                                 * sum(delta(snd_una)), or how many bytes
                                 * were acked.
                                 */
        u32     dsack_dups;     /* RFC4898 tcpEStatsStackDSACKDups
                                 * total number of DSACK blocks received
                                 */
        u32     snd_una;        /* First byte we want an ack for        */
};

struct tcp_sock {
      ...
      struct tcp_stats *stats;   // page allocation that can be mapped
};
```

Driver and Hardware Counters

enfabrica

Stats per queue maintained by driver

Stats per queue from hardware

System level stats from hardware

New ndo indicates support for new infrastructure

Hardware stats can have collection options to indicate how counters are accessed

- e.g., updated by timer every N seconds

Support for hardware dependent options

- e.g., hardware push vs software pull to update stats in the page

Depending on H/W details and support - ability to map BAR space to process for some counters

Derived Counters

enfabrica

Computed values

- Callback handler

Timer to update - collection option

Selectable Counters

enfabrica

Ultimate flexibility (with accompanying complexity)

Userspace opts into specific counters

Infrastructure

- Allocates a page of memory for counters
- Assigns slots in the page as counters are added and removed
- Updates stat location to do accounting at a given address
  - ‘struct some_object { unsigned int counter; }’ -> ‘struct some_object { unsigned int *counter; }’
  - Set counter address

More refined but limited to app-specific structs

Why ebpf is Not the Answer

enfabrica

ebpf allows custom counters across the stack with data going to a userspace map

- Tracepoint + kprobe or various networking hooks

Requires multiple attach points to get end-to-end accounting

Too much overhead for high performance datapath

- Enable tracepoint, probe or bpf attach hook + run bpf program + write to map
- Tracepoints, probes and bpf hook points are code based, not flow based
  - Either limited to 1 system wide stats daemon or take overhead of multiple programs on a tracepoint
- Nanoseconds matter for > 100Gbps flows

Need new hooks to get hardware or driver stats

- See the many, many discussions on ebpf and hardware offloads

Many of the counters duplicate existing system accounting

- e.g., Counters bumped multiple times depending on design

Linux TCP for ML Use Cases

RFC Summary

enfabrica

mempool for huge pages

- expands to custom memory provider

Rx ZC direct to GPU

- devmem patch set - packet payload written direct to GPU memory
- based on the memory providers

io_uring support

- extends devmem set to host memory

genl to manage H/W queues

- requirements around use of userspace and GPU memory and flushing references

What do people believe is the end goal of this approach?

- Still significant overhead in the datapath which is not relevant for GPU direct use case

ZC Direct to/from GPU

enfabrica

Should TCP window consider buffers available in buffer pool?

- Data will only land in flow specific buffers
- Slow application == no buffers == dropped packets == waste of network resources

Flushing S/W queues when devmem or host memory is invalidated

- retransmit, ofo and write queues
- references to released (or being released) memory

H/W Queues in Userspace

- Allow application to post buffers directly to hardware
  - Avoids unneeded system calls and page pool overhead proposed in current design
  - Process knows what buffers are available

Other

enfabrica

Big TCP for IPv6

- Drop use of extension header
- Added at L3, removed by driver or needs H/W support
  - Really only needed for tcpdump and it has a solution now
- Make IPv6 equivalent to IPv4 design

Thank You