VPP overview

Shwetha Bhandari
Developer@Cisco
Scalar Packet Processing

• A fancy name for processing one packet at a time
• Traditional, straightforward implementation scheme
• Interrupt, a calls b calls c … return return return
• Issues:
  • thrashing the I-cache (when code path length exceeds the primary I-cache size)
  • Dependent read latency (packet headers, forwarding tables, stack, other data structures)
  • Each packet incurs an identical set of I-cache and D-Cache misses
Packet Processing Budget

14 Mpps on 3.5 GHz CPU = 250 cycles per packet
## Memory Read/Write latency

<table>
<thead>
<tr>
<th></th>
<th>Sandy Bridge Ivy Bridge</th>
<th>Haswell</th>
<th>Skylake</th>
</tr>
</thead>
<tbody>
<tr>
<td>L1 data access (cycles)</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>L1 Peak Bandwidth (bytes/cycle)</td>
<td>2x16</td>
<td>2x32 load</td>
<td>2x32 load</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1x32 store</td>
<td>1x32 store</td>
</tr>
<tr>
<td>L2 data Access (cycles)</td>
<td>12</td>
<td>11</td>
<td>12</td>
</tr>
<tr>
<td>L2 peak bandwidth (bytes/cycle)</td>
<td>1x32</td>
<td>64</td>
<td>64</td>
</tr>
<tr>
<td>Shared L3 Access (cycles)</td>
<td>26-31</td>
<td>34</td>
<td>44</td>
</tr>
<tr>
<td>L3 peak bandwidth (bytes/cycle)</td>
<td>32</td>
<td>-</td>
<td>32</td>
</tr>
<tr>
<td>Data hit in L2 or L1D Dcache of another core</td>
<td>43 – clean hit 60 – modified hit</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- BUT memory is ~70+ ns away (i.e. 2.0 GHz = 140+ cycles)

Introducing VPP: the vector packet processor
Introducing VPP

Accelerating the dataplane since 2002

Fast, Scalable and consistent

- 14+ Mpps per core
- Tested to 1TB
- Scalable FIB: supporting millions of entries
- 0 packet drops, ~15µs latency

Optimized

- DPDK for fast I/O
- ISA: SSE, AVX, AVX2, NEON ..
- IPC: Batching, no mode switching, no context switches, non-blocking
- Multi-core: Cache and memory efficient
Introducing VPP

Extensible and Flexible modular design
• Implement as a directed graph of nodes
• Extensible with plugins, plugins are equal citizens.
• Configurable via CP and CLI

Developer friendly
• Deep introspection with counters and tracing facilities.
• Runtime counters with IPC and errors information.
• Pipeline tracing facilities, life-of-a-packet.
• Developed using standard toolchains.
Introducing VPP

**Fully featured**
- **L2**: VLAN, Q-in-Q, Bridge Domains, LLDP ...
- **L3**: IPv4, GRE, VXLAN, DHCP, IPSEC …
- **L3**: IPv6, Discovery, Segment Routing …
- **CP**: CLI, IKEv2 …

**Integrated**
- Language bindings
- Open Stack/ODL (Netconf/Yang)
- Kubernetes/Flanel (Python API)
- OSV Packaging
VPP in the Overall Stack

- Application Layer / App Server
- VM/VIM Management Systems
- Orchestration
- Network Controller
- Operating Systems
- Data Plane Services
  - Packet Processing
  - Network IO
- Hardware
VPP: Dipping into internals..
VPP Graph Scheduler

- Always process as many packets as possible
- As vector size increases, processing cost per packet decreases
- Amortize l-cache misses
- Native support for interrupt and polling modes
- Node types:
  - Internal
  - Process
  - Input
Sample Graph
Packet processing is decomposed into a directed graph node …

* approx. 173 nodes in default deployment
How does it work?

… instruction cache is warm with the instructions from a single graph node …

While packets in vector:
- Get pointer to vector
- While 4 or more packets:
  - PREFETCH #3 and #4
  - PROCESS #1 and #2
  - ASSUME next_node same as last packet
  - Update counters, advance buffers
  - Enqueue the packet to next_node
- While any packets:
  - <as above but single packet>

… data cache is warm with a small number of packets …

… packets are processed in groups of four, any remaining packets are processed on by one …
How does it work?

Microprocessor

ethernet-input

Packet 1
Packet 2

while packets in vector

Get pointer to vector

while 4 or more packets

PREFETCH #1 and #2

PROCESS #1 and #2

ASSUME next_node same as last packet

Update counters, advance buffers

Enqueue the packet to next_node

while any packets

<as above but single packet>

… prefetch packets #1 and #2 …
How does it work?

While packets in vector:
- Get pointer to vector
- While 4 or more packets:
  - Prefetch #3 and #4
  - Process #1 and #2
  - Assume next_node same as last packet
  - Update counters, advance buffers
  - Enqueue the packet to next_node
- While any packets:
  - <as above but single packet>

... process packet #3 and #4 ...
... update counters, enqueue packets to the next node ...
Modularity Enabling Flexible Plugins

Plugins can:
- Introduce new graph nodes
- Rearrange packet processing graph
- Can be built independently of VPP source tree
- Can be added at runtime (drop into plugin directory)
- All in user space

Enabling:
- Ability to take advantage of diverse hardware when present
- Support for multiple processor architectures (x86, ARM, PPC)
- Few dependencies on the OS (clib) allowing easier ports to other Oses/Env
VPP: performance
VPP Performance at Scale

IPv6, 24 of 72 cores

- 480Gbps zero frame loss

IPv4+, 2k Whitelist, 36 of 72 cores

- IMIX => 342 Gbps, 1518B => 462 Gbps

Zero-packet-loss Throughput for 12 port 40GE

Hardware:
Cisco UCS C460 M4
- Intel® C610 series chipset
- 4 x Intel® Xeon® Processor E7-8890 v3 (18 cores, 2.5GHz, 45MB Cache)
- 2133 MHz, 512 GB Total
- 9 x 2p40GE Intel XL710
- 18 x 40GE = 720GE !!

Latency
- 18 x 7.7 trillion packets soak test
- Average latency: <23 usec
- Min Latency: 7…10 usec
- Max Latency: 3.5 ms

Headroom
- Average vector size ~24-27
- Max vector size 255
- Headroom for much more throughput/features
- NIC/PCI bus is the limit not vpp
VPP: integrations
FD.io Integrations

Integration work done at

Control Plane

- OPEN DAYLIGHT
  - LISP flow mapping app
  - SFC
  - GBP app
  - VBD app

Data Plane

- Honeycomb
- FD.io ML2 Agent
- Felixv2 (Calico Agent)
- VPP

Openstack

- ODL Neutron
  - Neutron
  - FD.io Plugin

LISP Mapping Protocol

Netconf/Yang

REST
Summary

- VPP is a fast, scalable and low latency network stack in user space.
- VPP is trace-able, debug-able and fully featured layer 2, 3, 4 implementation.
- VPP is easy to integrate with your data-centre environment for both NFV and Cloud use cases.
- VPP is always growing, innovating and getting faster.
- VPP is a fast growing community of fellow travellers.

ML: vpp-dev@lists.fd.io  Wiki: wiki.fd.io/view/VPP
THANK YOU