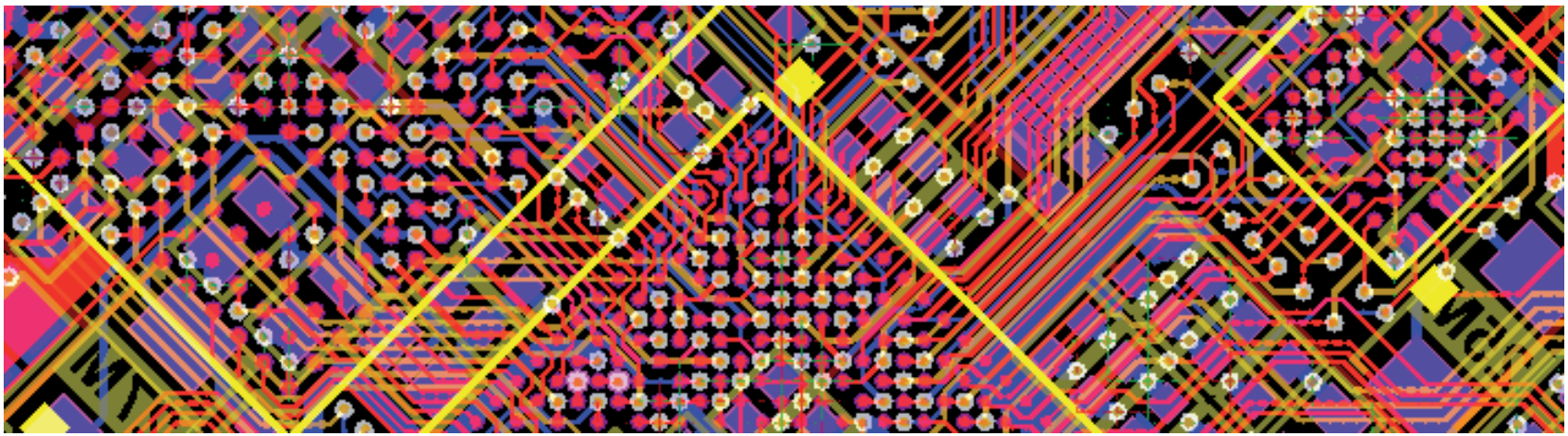
Introduction to VPP

Szymon Sliwa

11.05.2018



1. What is VPP?
2. Cross compilation
3. Debug capabilities
4. Useful CLI commands
5. Example: CLI commands for debugging
6. Running VPP
7. VPP Architecture
8. Useful resources

What is VPP?

The VPP platform is an extensible framework that provides out-of-the-box production quality switch/router functionality. It is the open source version of Cisco's Vector Packet Processing (VPP) technology: a high performance, packet-processing stack that can run on commodity CPUs.
The benefits of this implementation of VPP are its high performance, proven technology, its modularity and flexibility, and rich feature set... -
`https://wiki.fd.io/view/VPP/What_is_VPP%3F`

You can find braging about performance and a summary of the project aims there

VPP vs. FD.io: VPP $\in$ FD.io

VPP project roadmap: Haven't seen such thing, maybe joining weekly call (`https://wiki.fd.io/view/VPP/Meeting`), reading the summary of the previous calls and joining the mailing list may give some insight

Comparison to other frameworks (OVS, OFP?): VPP appears to be much more actively developed, and has quite good performance comparing to other frameworks (at least the project authors claim so)

Cross compilation

As cross compilations for Marvell platforms is quite cumbersome, a script has been created to automate things. The script and a basic instructions is located at `https://github.com/Semihalf/marvell-vpp/tree/vpp-build-tool`

When changes were made only in dpdk or musdk, it may be enough to just recompile dpdk and/or musdk and *dpdk_plugin*, detailed instructions are included in the readme of the above repository.

Dependences: Problems may raise if less than 1G of ram is avaliable. Lots of continuous memory for hugepages.

Adding platforms: Platforms are located in *build-data/platforms/*. To add a new platform, one should probably look at other files in that directory.

Debug capabilites

If compiled natively, VPP has useful makefile targets which can be listed using **make** without arguments, amongst others, there are
make build which builds debug version of VPP
make build-release which builds release version of VPP
make run
make run-release
make debug
make debug-release
The main difference between *debug* and *release* versions are the compile time flags, *release* has "-O2"
and *debug* has "-O0 -DCLIB_DEBUG"
("-g" is included in both, more information in *build-data/platforms/vpp.mk*)
How to set log level for specific module: I don't think that is possible, there are some defines enabling debug in parts of code, but they have to be found and enabled manually, probably (look TCP_DEBUG).

Useful CLI commands

VPP aids new user in choosing commands by *tab* autocompletion and CLI help.

To use the CLI help append **?** to a paritaly completed command and it will print all the possible expansions of the command with the needed arguments.

Also, some command abbreviations work
e.g. **show interface** can be abbreviated to **sh int**

Example: CLI commands for debugging

- **show interface** shows the avaliable interface and some packet counters
- **show hardware** shows more information about the interfaces and the hardware counters
- **show error** shows error counters, and all ipsec packets
- **trace add dpdk input** *num* adds *num* of packets to trace
- **show trace** prints the log of the packets traversing the graph

  very useful debugging configurational issues
- **clear trace** clears trace buffers the amount of packets to trace is also set to 0
- **show version** to be sure the right version of software is tested

Running VPP

VPP normally takes parameters from *startup.conf*, but on Armada platforms we provide it directly as command line parameter. So things like ports, worker amount, worker to port mapping and memory amount can be changed in the script *start_with_crypto_a3k.sh*.
CLI command **show int rx** shows which worker polls on which port

Multicore/multi-queue configuraton: *src/vpp/conf/startup.conf* shows how to place multiple queues per port, assign port to worker etc.

Different use cases: ipsec, bridging, routing, etc.: How to setup VPP for ipsec, bridging and routing can be found in the how-to prepared after the performance testing. The VPP wiki also contatins lots of use cases.

VPP Architecture

"The VPP platform is built on a packet processing graph. This modular approach means that anyone can plugin new graph nodes. This makes extensibility rather simple, and it means that plugins can be customized for specific purposes..." - `https://wiki.fd.io/view/VPP/What_is_VPP%3F`

Modules, plugins, etc.: VPP tries to make it's code modular, thus divides the base code into *layers* (vppinfra, vlib, vnet), and tries to add as much as possible as *plugins*, which are compiled to shared object files, and loaded at runtime

Support for hardware accelerations: Haven't heard of any except dpdk crypto, but they could probably easily be added as plugins. The code responsible for dpdk crypto PMD utilization resides in src/plugins/dpdk/ipsec/ .

VPP Architecture

Integration with other frameworks for IO

- VPP is able to communicate with Unix OS'es via TAP interfaces
- VPP is able to communicate with VM via Vhost-user interfaces
- There was an idea to port VPP to ODP, in a FD.io project named ODP4VPP, but it does not support the newest version of VPP, nor the newest version of ODP, and seems pretty much abandoned (as well as the rest of the ODP)
- There appears to be a plugin working directly on the Marvell PP2 driver in *src/plugins/marvell*

Useful resources

Generic VPP related stuff (non Marvell specific)

- `https://wiki.fd.io/view/VPP`
- `https://wiki.fd.io/view/Presentations`
- `https://www.youtube.com/channel/UCIJ2OP6_i1npoHM39kxvwyg` - fd.io youtube channel, the videos are quite lengthy, but contain lots of information, sometimes not avaliable elsewhere