VPP overview

fd.io Foundation

Agenda

- Overview
- Structure, layers and features
- Anatomy of a graph node
- Integrations
- FIB 2.0
- Future Directions
- New features
- Performance
- Continuous Integration and Testing
- Summary

Introducing VPP: the *vector packet processor*

fd.io Foundation

Introducing VPP (the vector packet processor)

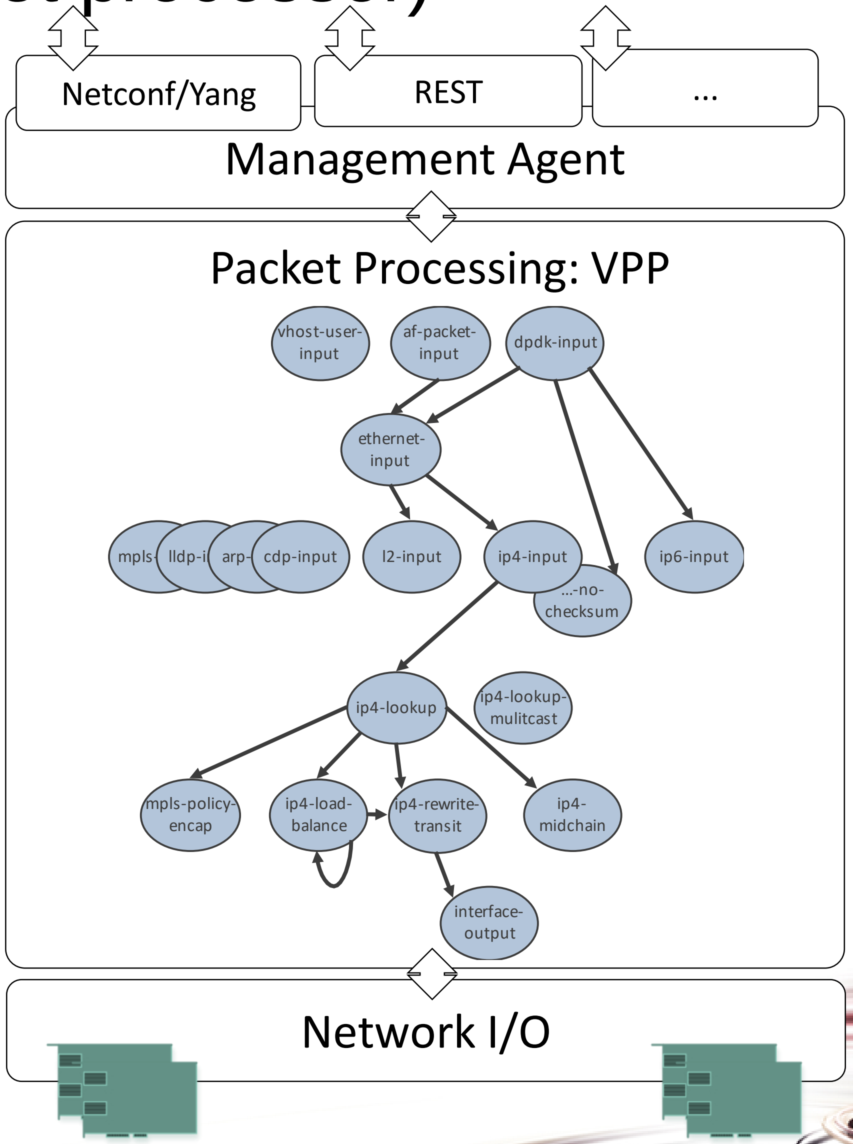
Accelerating the dataplane since 2002

Fast, Scalable and Determinisic

- 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



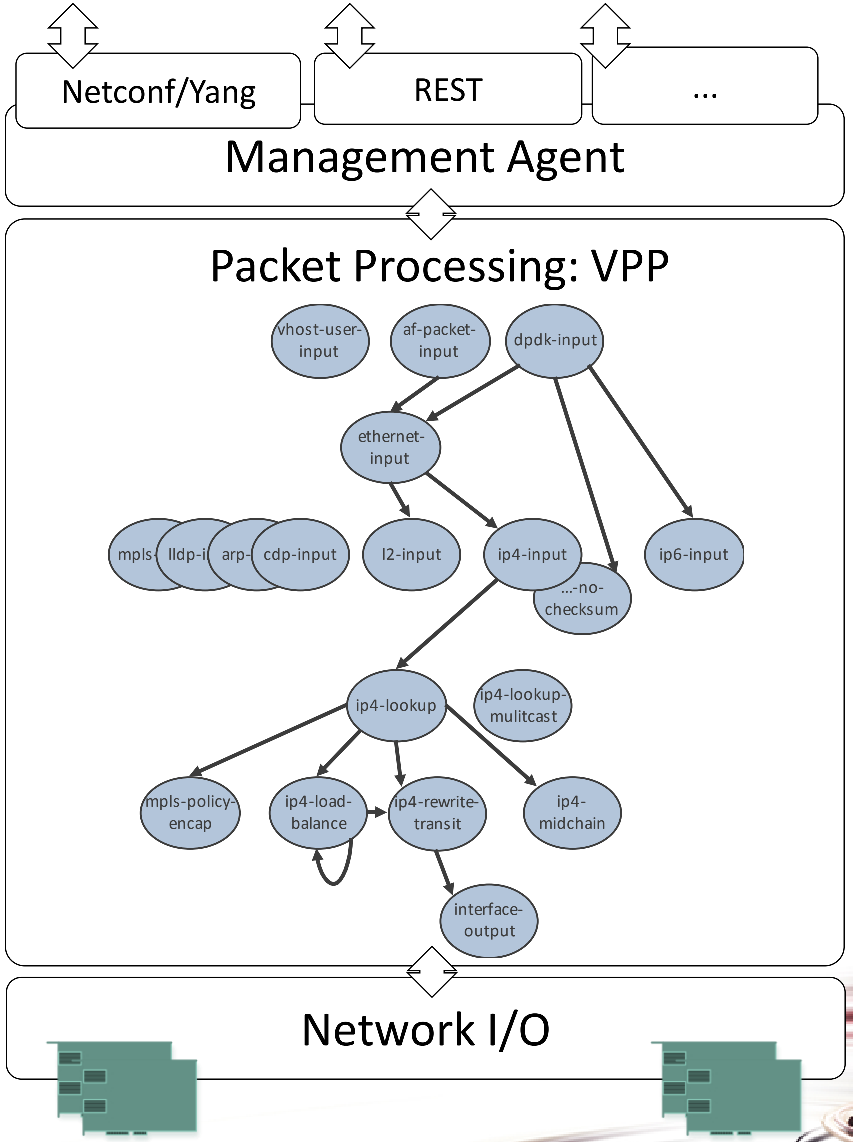
Introducing VPP

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.



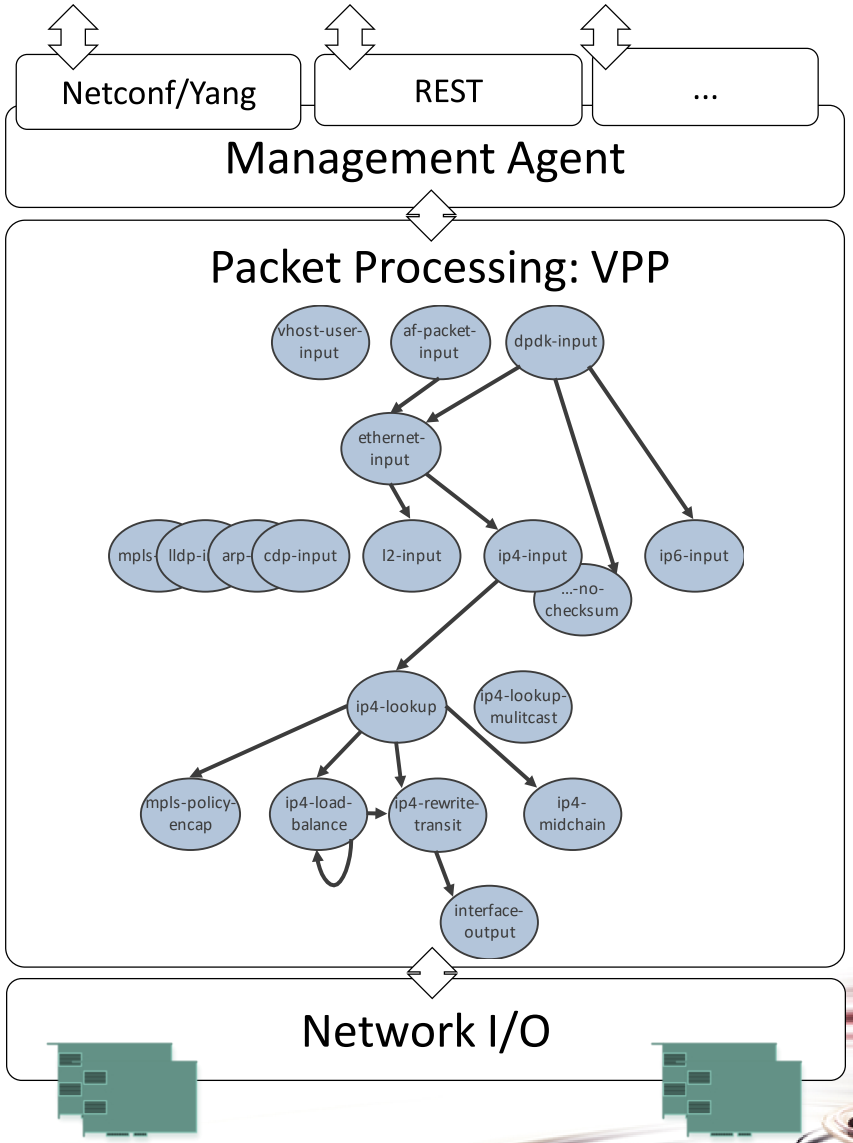
Introducing VPP

Fully featured

- **L2:** VLan, Q-in-Q, Bridge Domains, LLDP ...
- **L3:** IPv4, GRE, VXLAN, DHCP, IPSEC …
- **L3:** IPv6, Discovery, Segment Routing …
- **L4:** TCP, UDP …
- **CP:** API, CLI, IKEv2 …

Integrated

- Language bindings
- Open Stack/ODL (Netconf/Yang)
- Kubernetes/Flanel (Python API)
- OSV Packaging



VPP: structure, layers and features

VPP: VPP Layering

VNET

VPP networking source

- Devices
- Layer [2, 3, 4]
- Session Management
- Overlays
- Control Plane
- Traffic Management

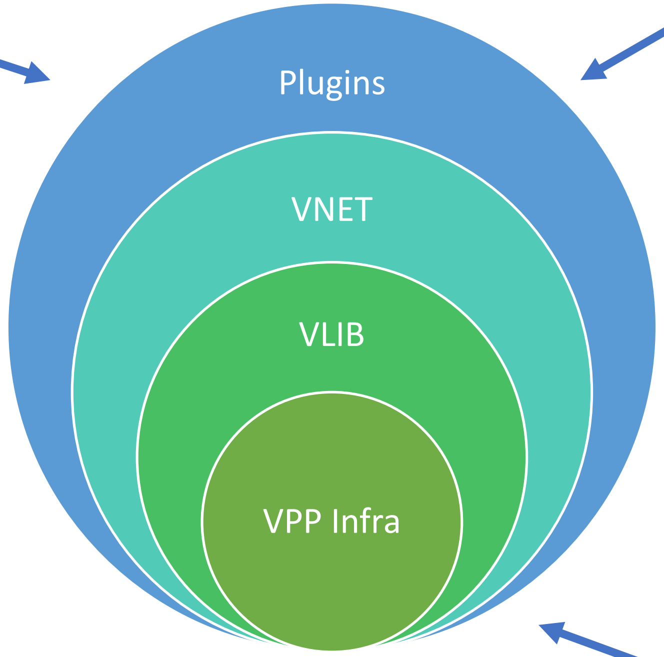
Plugins

- Plugins can be in-tree:

  SNAT,
  Policy ACL,
  Flow Per Packet,
  ILA,
  IOAM,
  LB,
  SIXRD,
  VCGN

- Separate fd.io project:

  NSH_SFC



VLIB

VPP application management

- buffer, buffer management
- graph node, node management
- tracing, counters
- threading
- CLI

and most importantly …

- main()

VPP INFRA

Library of function primitives, for

- memory management
- memory operations
- vectors
- rings
- hashing
- timers

VPP: VNET Features

Devices

- AF_PACKET
- DPDK v16.11, HQOS, CryptoDev
- NETMAP
- SSVM
- vhost-user

Layer 2

- Ethernet, MPLS over Ethernet
- HDLC, LLC, SNAP, PPP, SRP, LLDP
- VLAN, Q-in-Q
- MAC Learning
- Bridging
  - Split-horizon group support/EFP Filtering
  - VTR – push/pop/Translate (1:1,1:2, 2:1,2:2)
- ARP : Proxy, termination
- IRB: BVI Support with Router/MAC assignment
- Flooding
- Input ACLs
- Interface cross-connect

Layer 3

- Source RPF
- Thousands of VRFs
  - Controlled cross-VRF lookups
- Multipath – ECMP and Unequal Cost
- IPSec
- IPv6
  - Neighbor discovery
  - Router Advertisement
  - Segment Routing
- FIB 2.0
  - Multimillion scalable FIBs
  - Recursive FIB lookup, failure detection
  - IP MPLS FIB
  - Shared FIB adjacencies

Layer 4

- UDP
- TCP
- Sockets: FIFO, Socket PreLoad

Overlays

- GRE
  - MPLS-GRE
  - NSH-GRE
- VXLAN
- VXLAN-GPE
- L2TPv3

Traffic Management

- Mandatory Input Checks:
  - TTL expiration, Header checksum, L2 length < IP length, ARP resolution/snooping, per interface whitelists
- Multiple million Classifiers, arbitrary N-tuple
- Lawful Intercept
- Policer
- GBP/Security Groups classifier support
- Connection tracking
- MAP/LW46
- SNAT
- MagLev-like Load Balancer
- Identifier Locator Addressing (ILA)
- High performance port range ingress filtering

Control Plane

- LISP
- NSH
- iOAM
- DHCP
- IKEv2

fd.io Foundation

VPP: anatomy of a graph node

VPP: anatomy of a graph node

```
#define VLIB_REGISTER_NODE (  x,
                              ...
                           )
```

Value:

```
__VA_ARGS__ vlib_node_registration_t x;                              \
static void __vlib_add_node_registration_##x (void)                      \
    __attribute__((__constructor__)) ;                                   \
static void __vlib_add_node_registration_##x (void)                      \
{                                                                        \
    vlib_main_t * vm = vlib_get_main();                                  \
    x.next_registration = vm->node_main.node_registrations;              \
    vm->node_main.node_registrations = &x;                               \
}                                                                        \
__VA_ARGS__ vlib_node_registration_t x
```

Definition at line 143 of file node.h.

- VLIB_REGISTER_NODE, macro to declare a graph node.
- Creates:-
  - a graph node registration vlib_node_registration_t <graph node>
  - initializes values in <graph node>
  - a construction function __vlib_add_node_registration_<graph node> to register the graph node at startup.

vlib_node_registration_t			
Type	Name	Description	User visible?
vlib_node_function_t *	function	Vector processing function for this node	
char *	name	Node name	see `show run`
u16	n_errors	Number of error codes used by this node.	
char **	error_strings	Error strings indexed by error code for this node.	see `show error`
u16	n_next_nodes	Number of next node names that follow.	
char *	next_nodes[]	Names of next nodes which this node feeds into.	

VPP: anatomy of a VXLAN graph node

- VLIB_REGISTER_NODE registers `vxlan4_input_node` node
  - vxlan4_input is the vector processing function.
  - vxlan errors strings.
    - no such tunnel.
  - add next node strings.
    - error-drop – no such tunnel.
    - l2-input – layer 2 input.
    - IPv4-input – IPv4 input
    - IPv6-input – IPv6 input

#define foreach_vxlan_input_next

Value:

```
_(DROP, "error-drop")                     \
_(L2_INPUT, "l2-input")                   \
_(IP4_INPUT, "ip4-input")                 \
_(IP6_INPUT, "ip6-input")
```

Definition at line 99 of file vxlan.h.

vlib_node_registration_t vxlan4_input_node

Initial value:

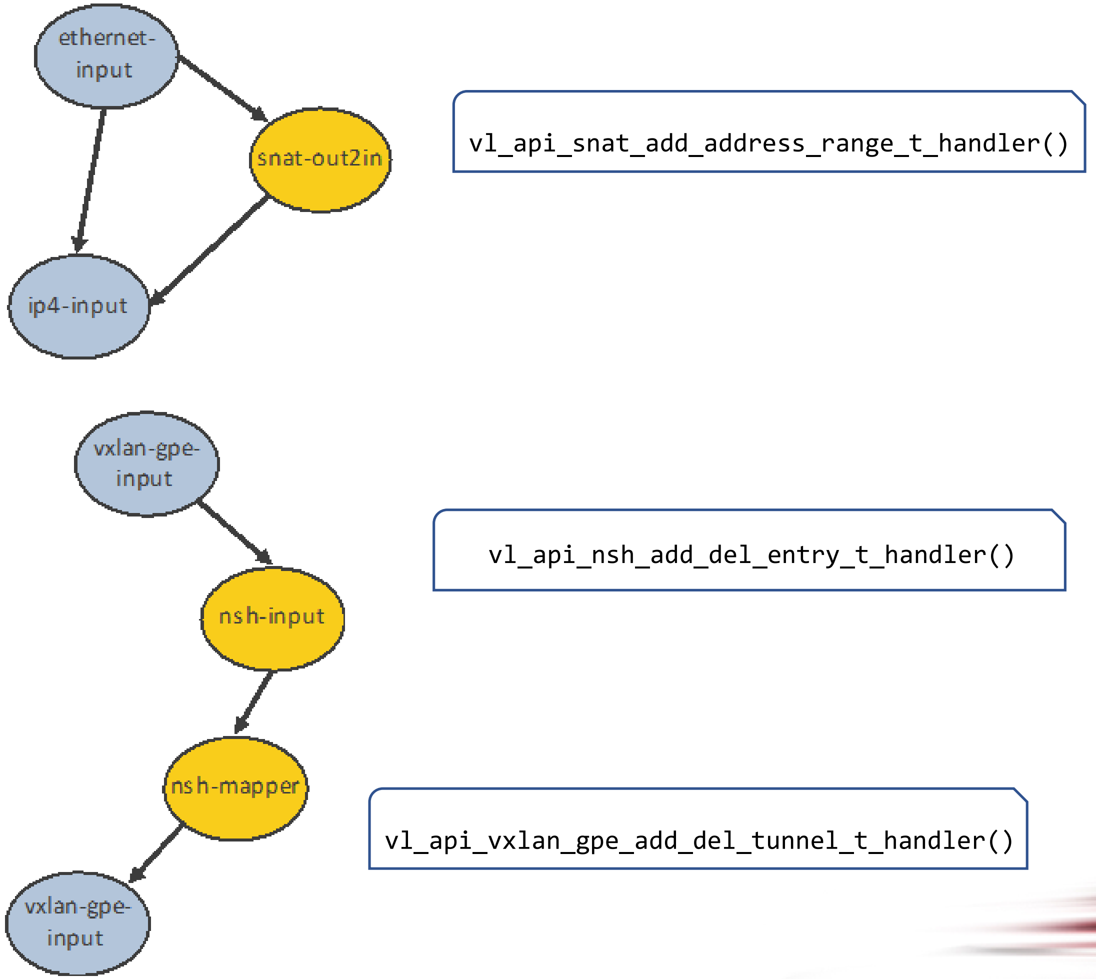
```
= {
  .function = vxlan4_input,
  .name = "vxlan4-input",

  .vector_size = sizeof (u32),

  .n_errors = VXLAN_N_ERROR,
  .error_strings = vxlan_error_strings,

  .n_next_nodes = VXLAN_INPUT_N_NEXT,
  .next_nodes = {
#define _(s,n)
    foreach_vxlan_input_next

  },


  .format_trace = format_vxlan_rx_trace,

}
```

(*constructor*) VLIB_REGISTER_NODE (vxlan4_input_node)

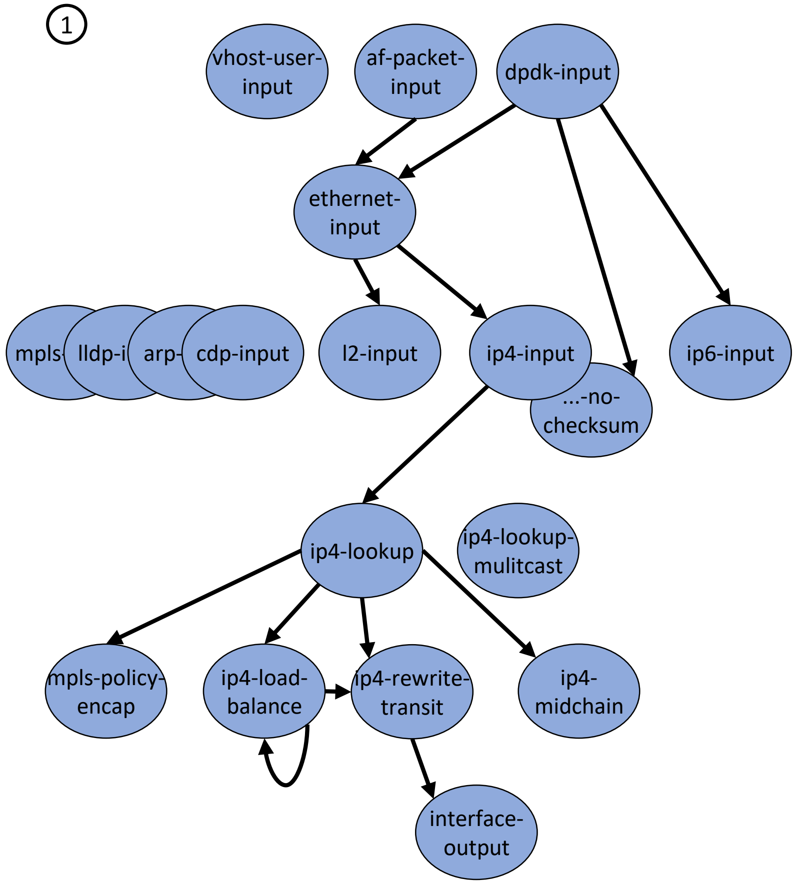
Definition at line 22 of file decap.c.

VPP: graph node interfaces

- Each feature; collection of graph nodes expose a binary API and CLI
- API
  - Multiple language bindings
    - Python
    - Java
    - LUA
  - Implementation
    - High-performance shared-memory ring-buffer
    - Asynchronous callback
- CLI
  - Accessible via UNIX Channel, see vpp_api_test (VAT).
  - Runtime composed list of commands.
  - Typical CLI features; help system (?), command history etc.

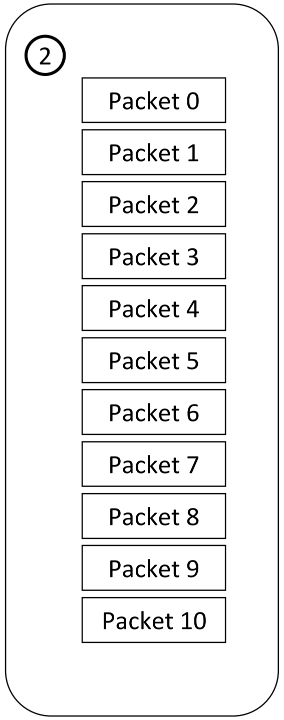


VPP: How does it work?



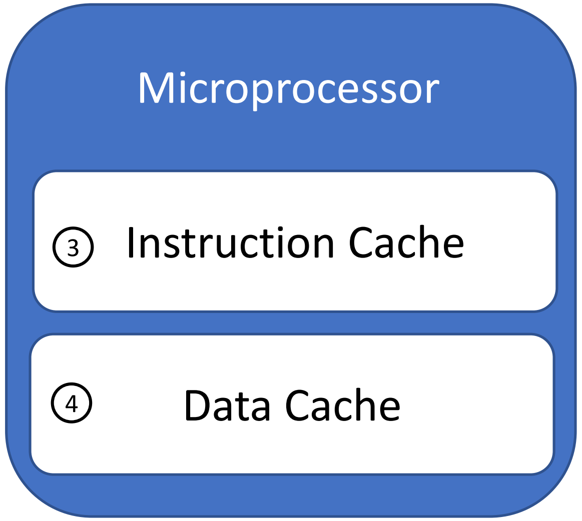
Packet processing is decomposed into a directed graph node …

* approx. 173 nodes in default deployment



… packets moved through graph nodes in vector …

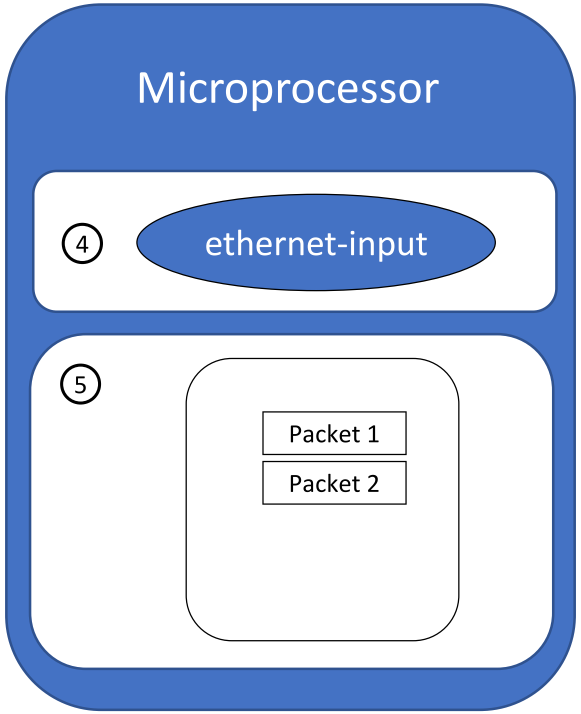
… graph nodes are optimized to fit inside the instruction cache …



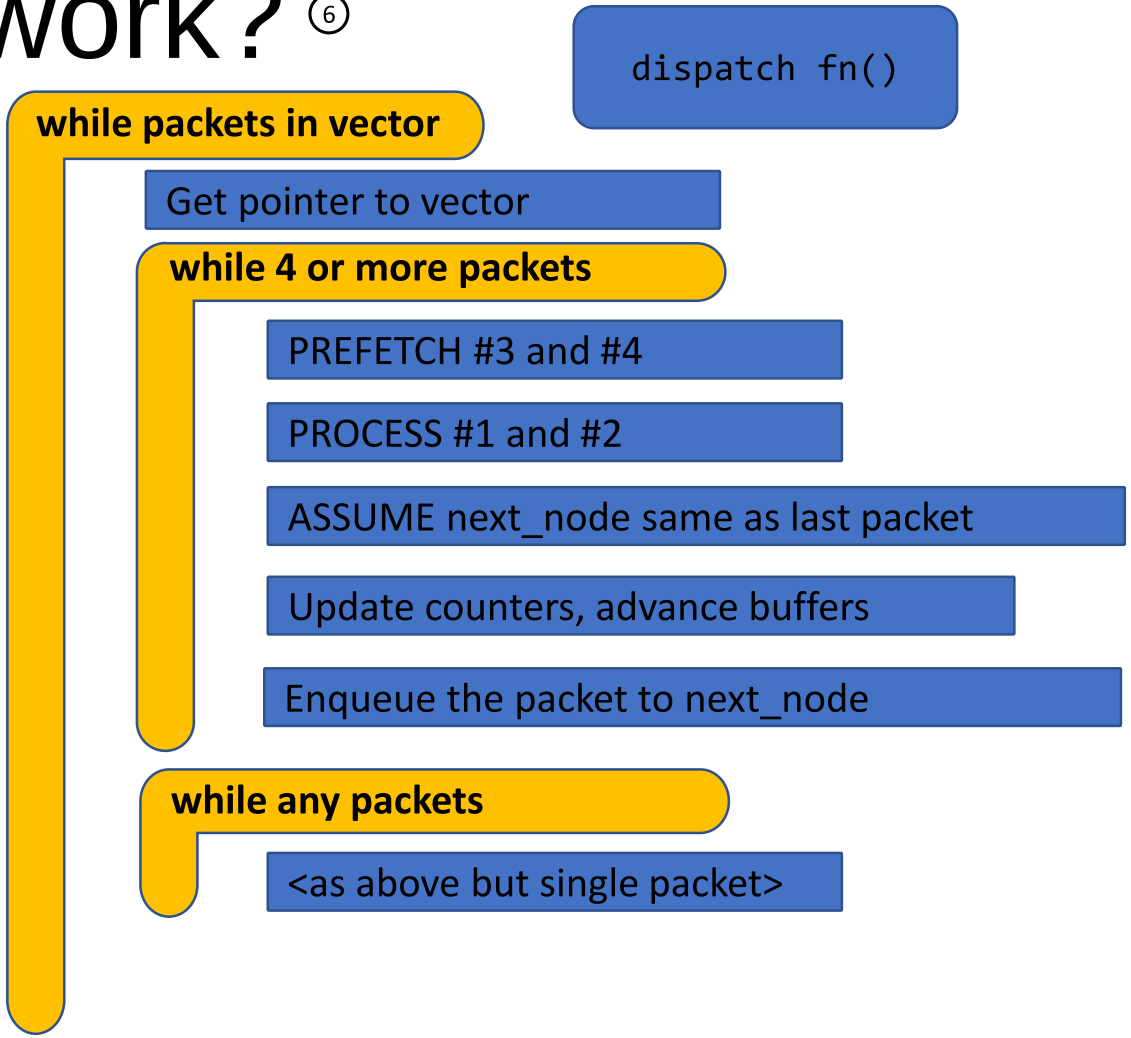
… packets are pre-fetched, into the data cache …

VPP: How does it work? 6

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

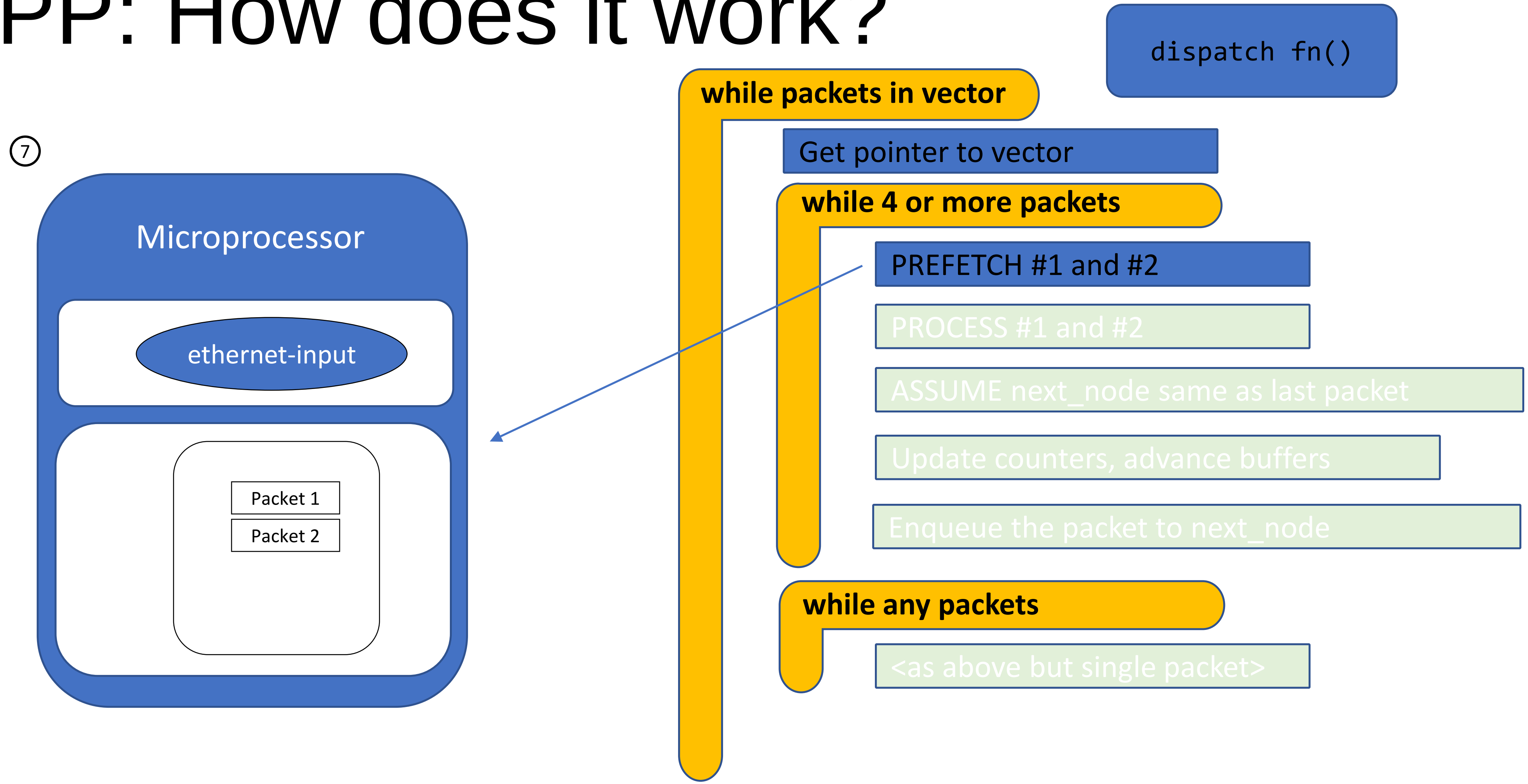


… data cache is warm with a small number of packets ..



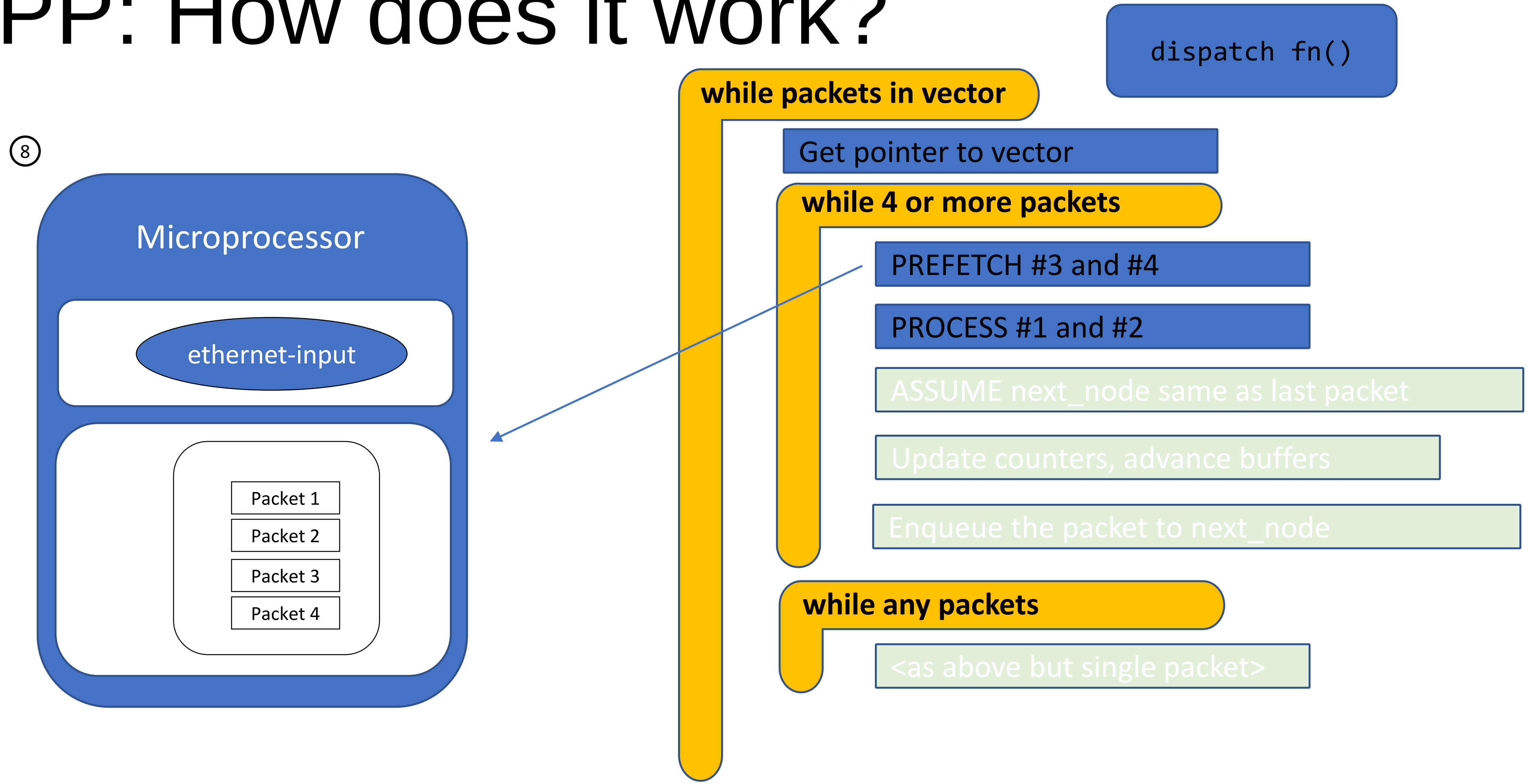
… packets are processed in groups of four, any remaining packets are processed on by one …

VPP: How does it work?



… prefetch packets #1 and #2 …

VPP: How does it work?

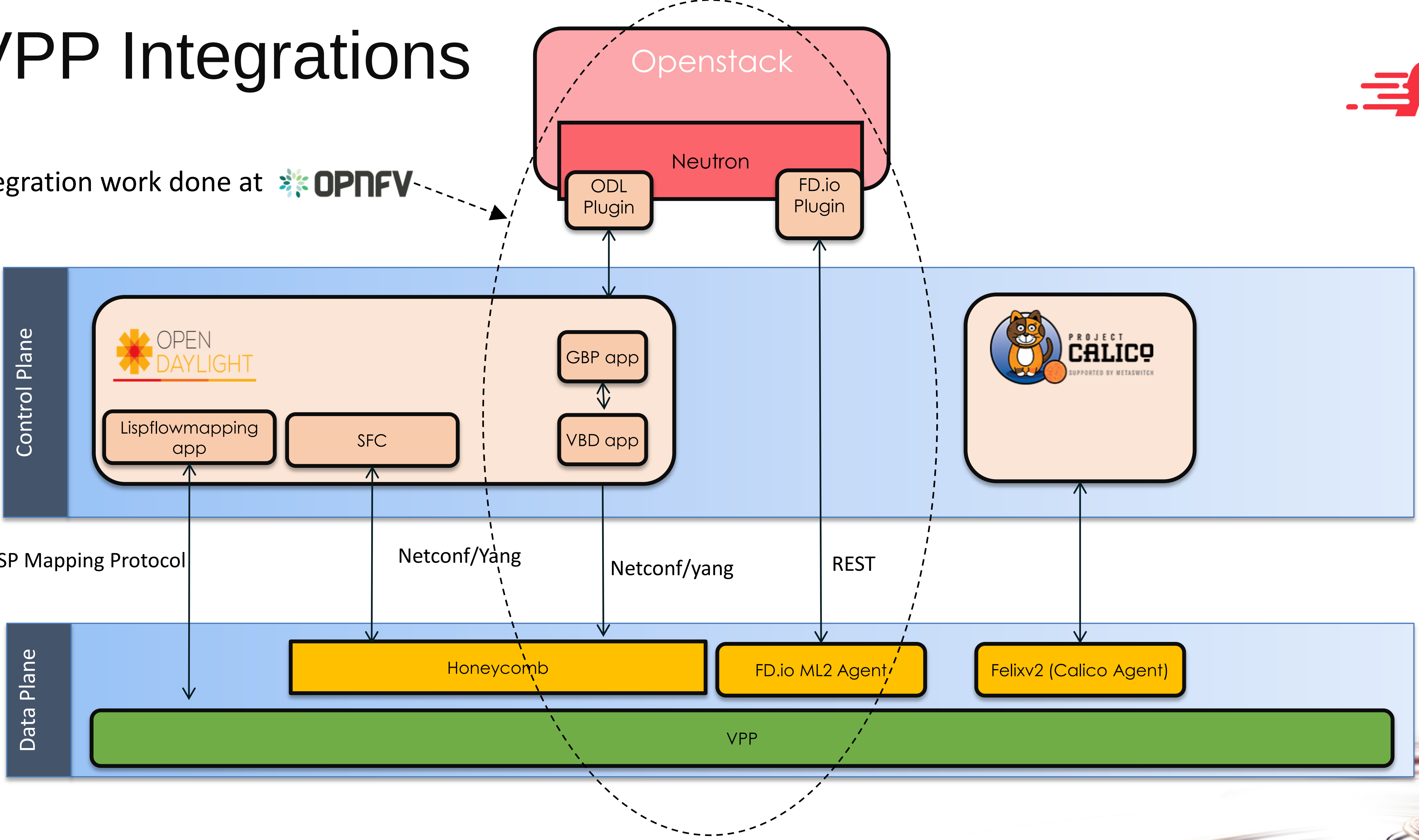


… process packet #3 and #4 …
… update counters, enqueue packets to the next node …

VPP: integrations

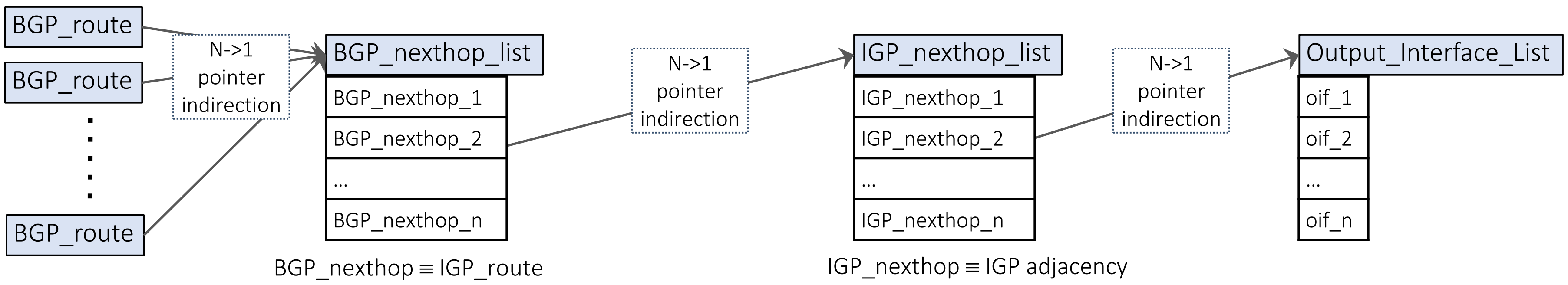
VPP Integrations

Integration work done at OPNFV



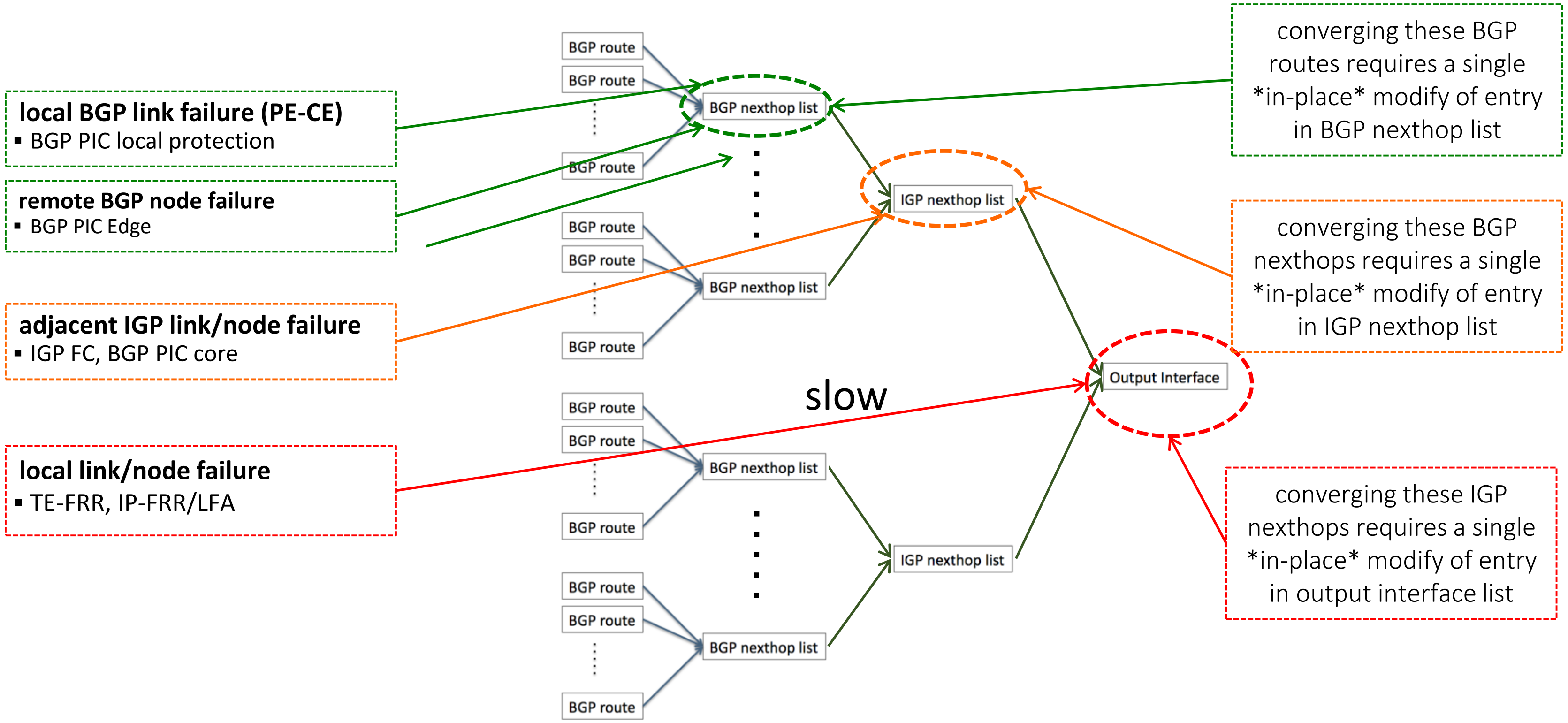
VPP: FIB 2.0

Why hierarchical FIB in data plane – recap



- Data plane FIB pointer indirection structure between BGP, IGP and adjacency entries enables scale independent fast convergence

VPP v17.01: FIB 2.0 (Hierarchical FIB)

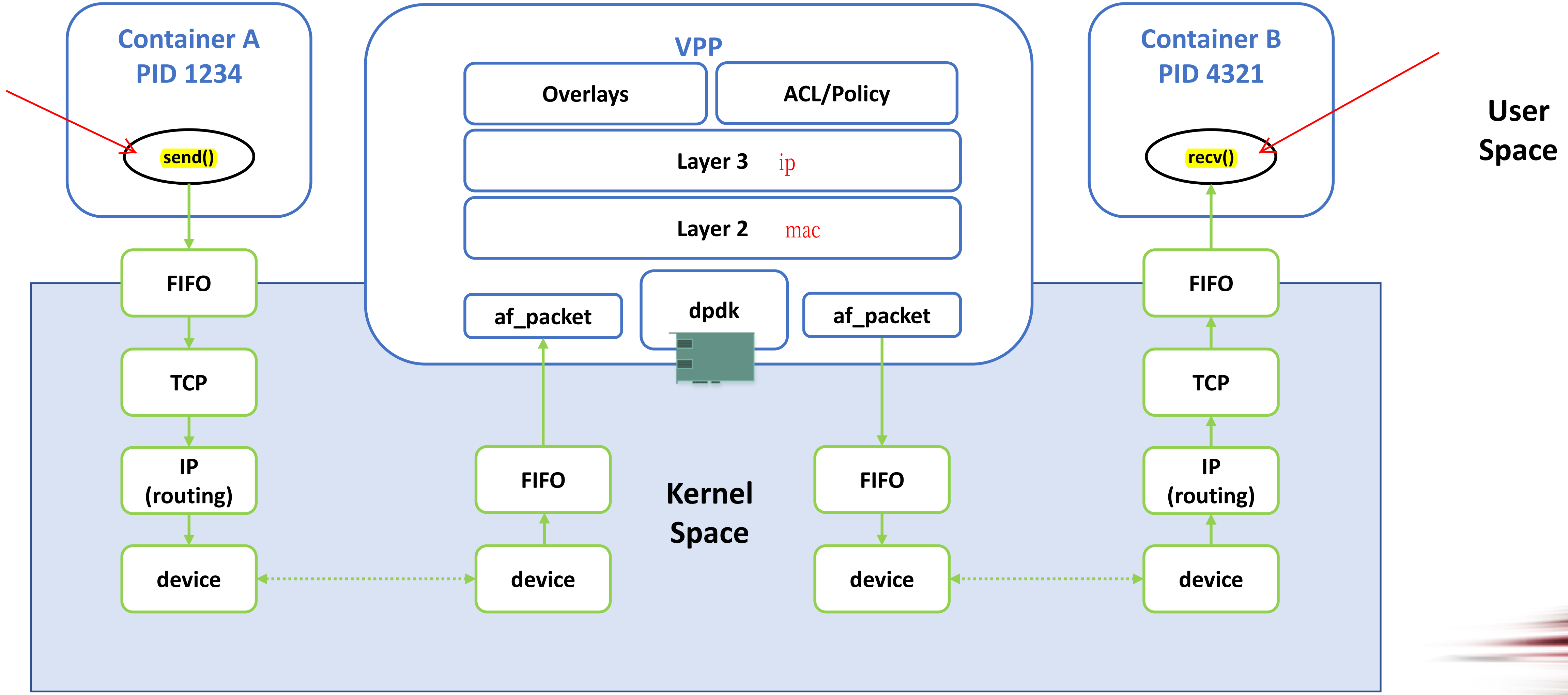


- VPP FIB 2.0 is a fast FIB implementation that provides for fast route updates, robust failure and fast routing decisions.
- It provides an optimal implementation without the trade-off's associated with a collapsed FIB (faster but slow updates) or a decoupled FIB/RIB (slow but faster updates)
- Data plane FIB indirection enables route scale independent failure handling

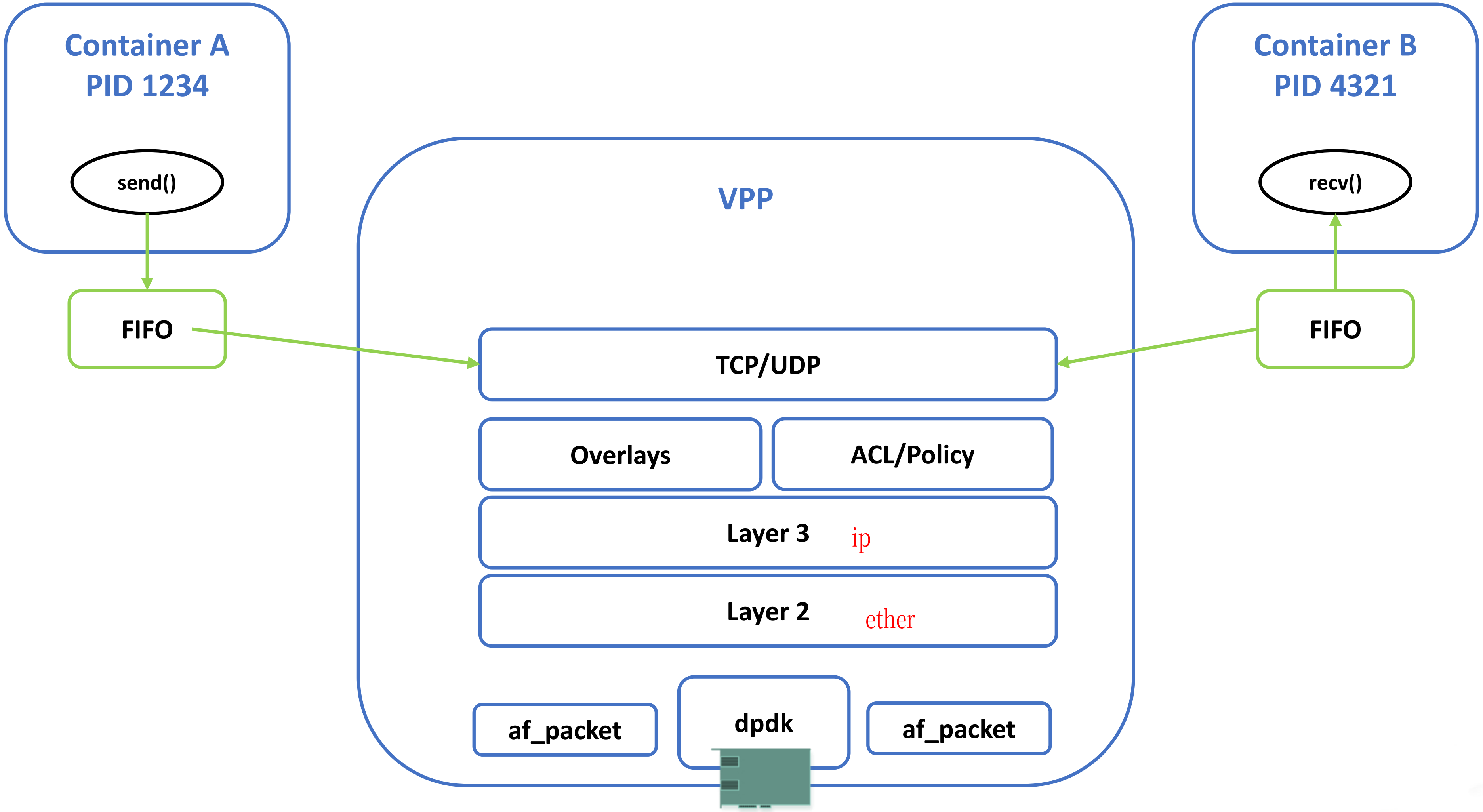
VPP: future directions

- Accelerating container networking
- Accelerating IPSEC

Container networking: Current State



Smarter networking: Future State



User Space

全都在用户态做？

fd.io Foundation

Accelerating IPSEC

VPP

Cryptodev API

DPDK

Vector/Crypto Instructions

Quick Assist

Future On-core Accel

Future Accelerators

2017

2018

2019

VPP: new features

Rapid Release Cadence – ~3 months

16-06
Release- VPP

16-09
Release:
VPP, Honeycomb,
NSH_SFC, ONE

17-01
Release:
VPP, Honeycomb,
NSH_SFC, ONE

17-04
Release:
VPP, Honeycomb,
NSH_SFC, ONE…

16-06 New Features

- Enhanced Switching & Routing
  - SRv6 spray use-case (IP-TV)
  - LISP xTR support
  - VXLAN over IPv6 underlay
  - per interface whitelists
  - shared adjacencies in FIB
- Improves interface support
  - vhost-user – jumbo frames
  - Netmap interface support
  - AF_Packet interface support
- Improved programmability
  - Python API bindings
  - Enhanced JVPP Java API bindings
  - Enhanced debugging cli
- Hardware and Software Support
  - Support for ARM 32 targets
  - Support for Raspberry Pi
  - Support for DPDK 16.04

16-09 New Features

- Enhanced LISP support for
  - L2 overlays
  - Multitenancy
  - Multihoming
  - Re-encapsulating Tunnel Routers
  - Map-Resolver failover algorithm
- New plugins for
  - SNAT
  - MagLev-like Load
  - Identifier Locator Addressing
  - NSH SFC SFF's & NSH Proxy
- Port range ingress filtering
- Dynamically ordered subgraphs

17-01 New Features

- Hierarchical FIB
- Performance Improvements
  - DPDK input and output nodes
  - L2 Path
  - IPv4 lookup node
- IPSEC
  - SW and HW Crypto Support
- HQoS support
- Simple Port Analyzer (SPAN)
- BFD
- IPFIX Improvements
- L2 GRE over IPSec tunnels
- LLDP
- LISP Enhancements
  - Source/Dest control plane
  - L2 over LISP and GRE
  - Map-Register/Map-Notify
  - RLOC-probing
- ACL
- Flow Per Packet
- SNAT – Multithread, Flow Export
- LUA API Bindings

17-04 New Features

….

Future Release Plans – 17.04 (Due Apr 19)

VPP Userspace Host Stack

TCP stack
DHCPv4 relay multi-destination
DHCPv4 option 82
DHCPv6 relay multi-destination
DHPCv6 relay remote-id
ND Proxy

SNAT

CGN: Configurable port allocation
CGN: Configurable Address pooling
CPE: External interface
DHCP support
NAT64, LW46

Security Groups

Routed interface support
L4 filters with IPv6 Extension Headers

API

Move to CFFI for Python binding
Python Packaging improvements
CLI over API
Improved C/C++ language binding

Segment Routing v6

SRv6 Network Programming
SR Traffic Engineering
SR LocalSIDs
Framework to expand LocalSIDs w/ plugins

iOAM

UDP Pinger w/path fault isolation
IOAM as type 2 metadata in NSH
IOAM raw IPFIX collector and analyzer
Anycast active server selection

IPFIX

Collect IPv6 information
Per flow state

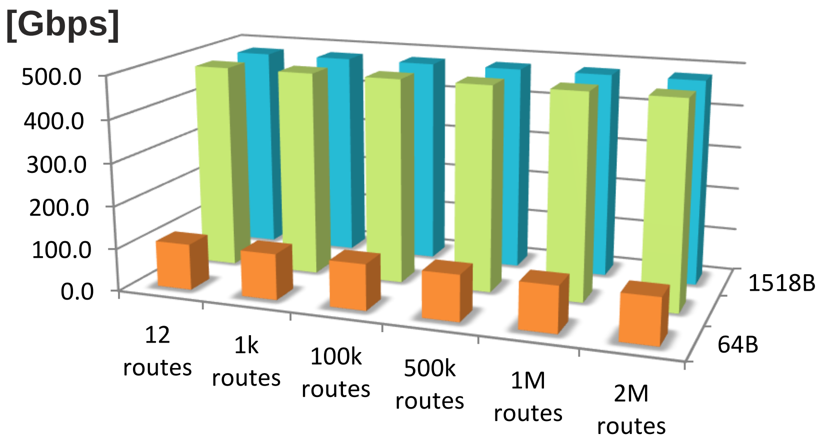
VPP: performance

CSIT NDR Throughput VPP 16.09 v 17.01

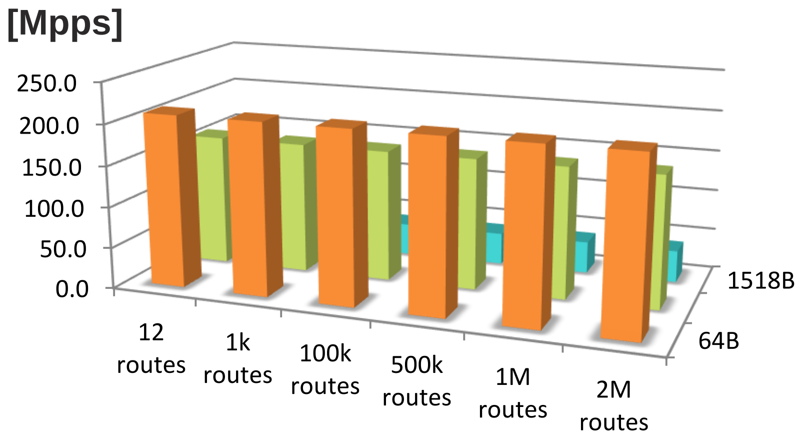
VPP Performance at Scale

Phy-VS-Phy

IPv6, 24 of 72 cores

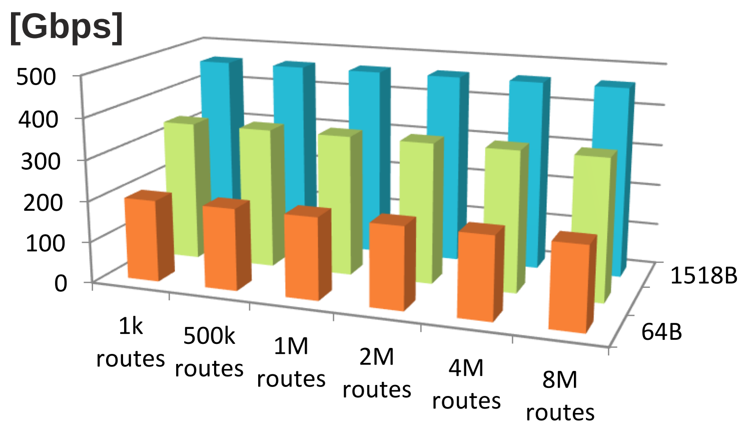


480Gbps zero frame loss

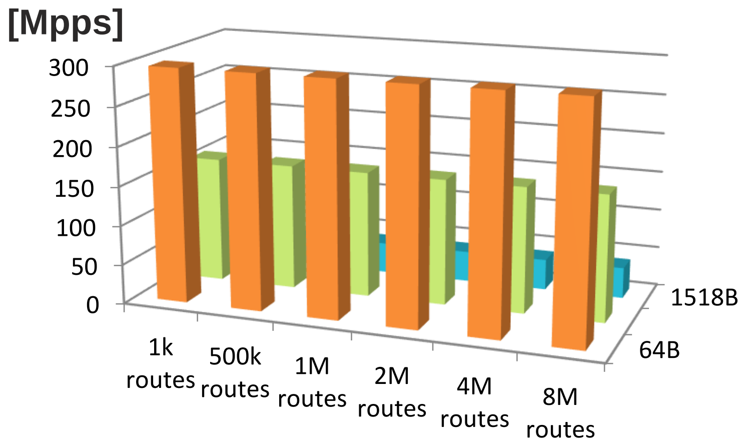


200Mpps zero frame loss

IPv4+ 2k Whitelist, 36 of 72 cores



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



64B => 238 Mpps

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 7.7trillion 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: Continuous Integration and Testing

fd.io Foundation

Continuous Quality, Performance, Usability

Built into the development process – patch by patch

Submit → Automated Verify → Code Review → Merge → Publish Artifacts

Build/Unit Testing
120 Tests/Patch

Build binary packaging for
- Ubuntu 14.04
- Ubuntu 16.04
- Centos 7

Automated Style Checking

Unit test :

IPFIX	IPv6
BFD	IP Multicast
Classifier	L2 FIB
DHCP	L2 Bridge Domain
FIB	MPLS
GRE	SNAT
IPv4	SPAN
IPv4 IRB	VXLAN
IPv4 multi-VRF	

System Functional Testing
252 Tests/Patch

DHCP – Client and Proxy
GRE Overlay Tunnels
L2BD Ethernet Switching
L2 Cross Connect Ethernet Switching
LISP Overlay Tunnels
IPv4-in-IPv6 Softwire Tunnels
Cop Address Security
IPSec
IPv6 Routing – NS/ND, RA, ICMPv6
uRPF Security
Tap Interface
Telemetry – IPFIX and Span
VRF Routed Forwarding
iACL Security – Ingress – IPv6/IPv6/Mac
IPv4 Routing
QoS Policer Metering
VLAN Tag Translation
VXLAN Overlay Tunnels

Performance Testing
144 Tests/Patch, 841 Tests

L2 Cross Connect
L2 Bridging
IPv4 Routing
IPv6 Routing
IPv4 Scale – 20k,200k,2M FIB Entries
IPv4 Scale - 20k,200k,2M FIB Entries
VM with vhost-userr
- PHYS-VPP-VM-VPP-PHYS
- L2 Cross Connect/Bridge
- VXLAN w/L2 Bridge Domain
- IPv4 Routing

COP – IPv4/IPv6 whiteless
iACL – ingress IPv4/IPv6 ACLs
LISP – IPv4-o-IPv6/IPv6-o-IPv4
VXLAN
QoS Policer
L2 Cross over
L2 Bridging

Usability

Merge-by-merge:
- apt installable deb packaging
- yum installable rpm packaging
- autogenerated code documentation
- autogenerated cli documentation

Per release:
- autogenerated testing reports
  - report perf improvements

Puppet modules
Training/Tutorial videos
Hands-on-usecase documentation

Merge-by-merge packaging feeds
Downstream consumer CI pipelines

Run on real hardware in fd.io Performance Lab

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**

Join us in FD.io & VPP - f*ellow travellers are <u>always</u> welcome.*
Please reuse and contribute!

Questions?

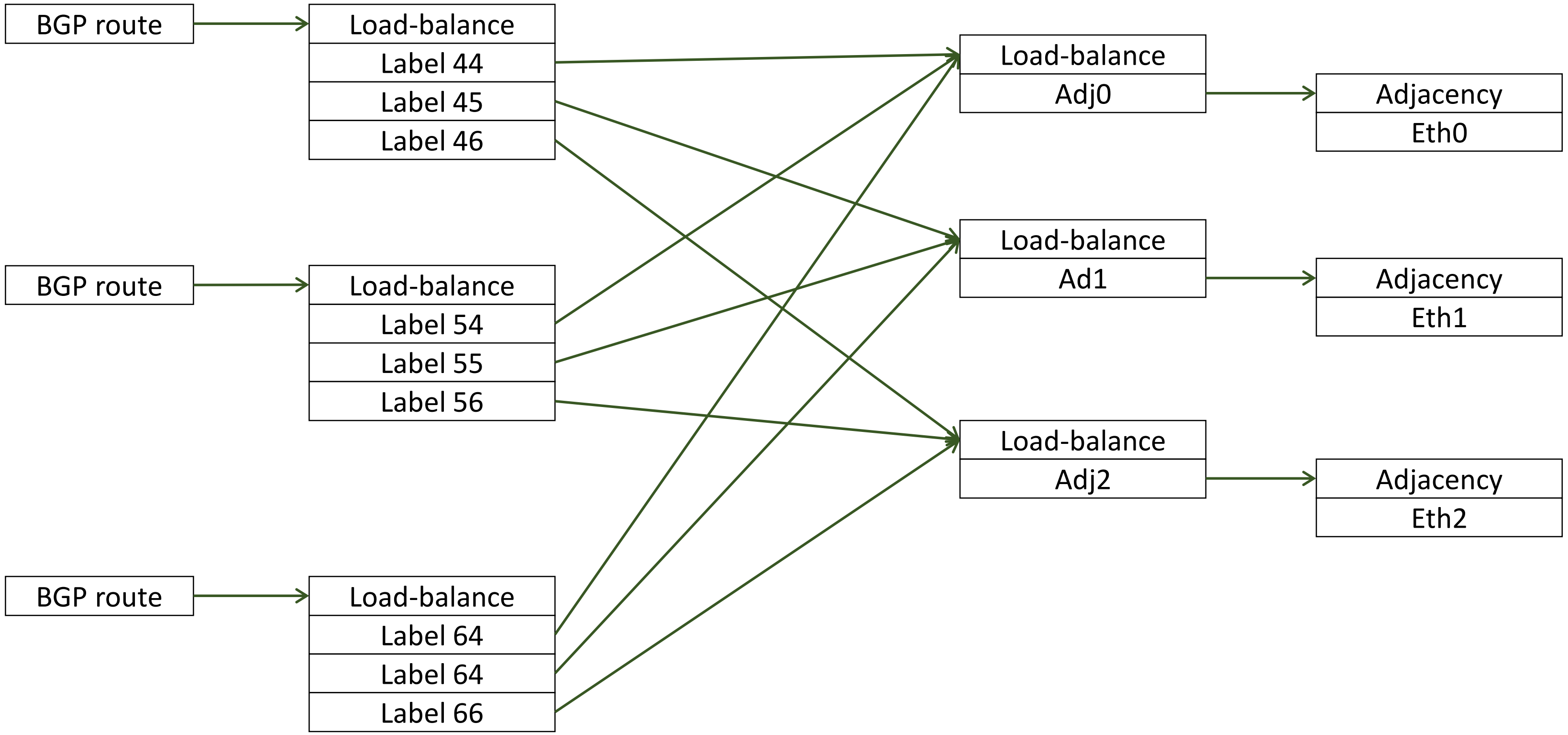
VPP: Source Structure

Directory name	Description
build-data	Build metadata – package and platform specific build settings. e.g. vpp_lite, x86, cavium etc.
build-root	Build output directory • build-vpp_lite_debug-native - build artifacts for vpp_lite, built with symbols. • install-vpp_lite_debug-native – fakeroot for vpp_lite installation, built with symbols. • deb – debian packages • rpm – rpm packages • vagrant – bootstrap a development environment
src/plugins	VPP bundled plugins directory - ila-plugin: Identifier Locator Addressing (ILA) - flowperpkt-plugin: Per-packet IPFIX record generation plugin - lb-plugin: MagLev-like Load Balancer, similar to Google's Maglev Load Balancer - snat-plugin: Simple ip4 NAT plugin - sample-plugin: Sample macswap plugin
src/vnet	VPP networking source - device: af-packet, dpdk pmd, ssvm - l2 : ethernet, mpls, lldp, ppp, l2tp, mcast - l3+: ip[4,6], ipsec, icmp, udp - overlays: vxlan, gre
src/vpp	VPP application source
src/vlib	VPP application library source;
src/vlib-api	VPP API library source
src/vpp-api	VPP application API source
src/vppapigen	VPP API generator source
src/vppinfra	VPP core library source

VPP: Build System

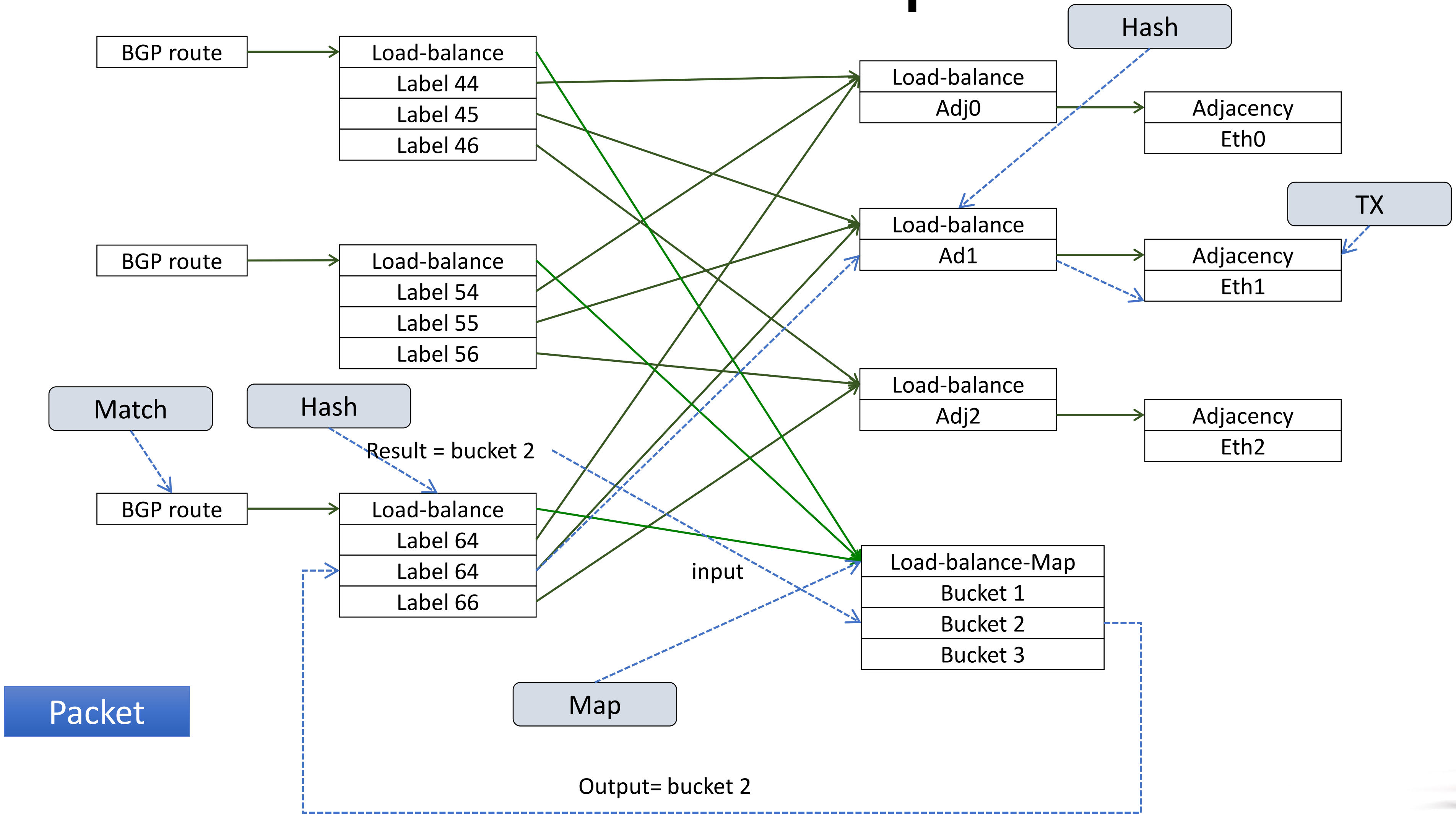
Make Targets	Description
bootstrap	prepare tree for build, setup paths and compilers etc
install-dep	install software dependencies, automatically apt-get build dependencies, used by vagrant provisioning scripts.
wipe, wipe-release	wipe all products of debug/release build
build, build-release	build debug/release binaries
plugins, plugins-release	build debug/release plugin binaries
rebuild, rebuild-release	wipe and build debug/release binaries
run, run-release	run debug/release binary in interactive mode
debug	run debug binary with debugger (gdb)
test, test-debug	build and run functional tests
build-vpp-api	build vpp-api
pkg-deb, pkg-rpm	build packages, build debian and rpm packaging for VPP, can be dpkg'ed or rpm'ed afterward.
ctags, gtags, cscope	(re)generate ctags/gtags/cscope databases
doxygen	(re)generate documentation
Make Variables	**Description**
V	1 or 0, to switch on verbose builds
PLATFORM	Platform specific build, e.g. vpp_lite

Hierarchical FIB in Data plane – VPN-v4



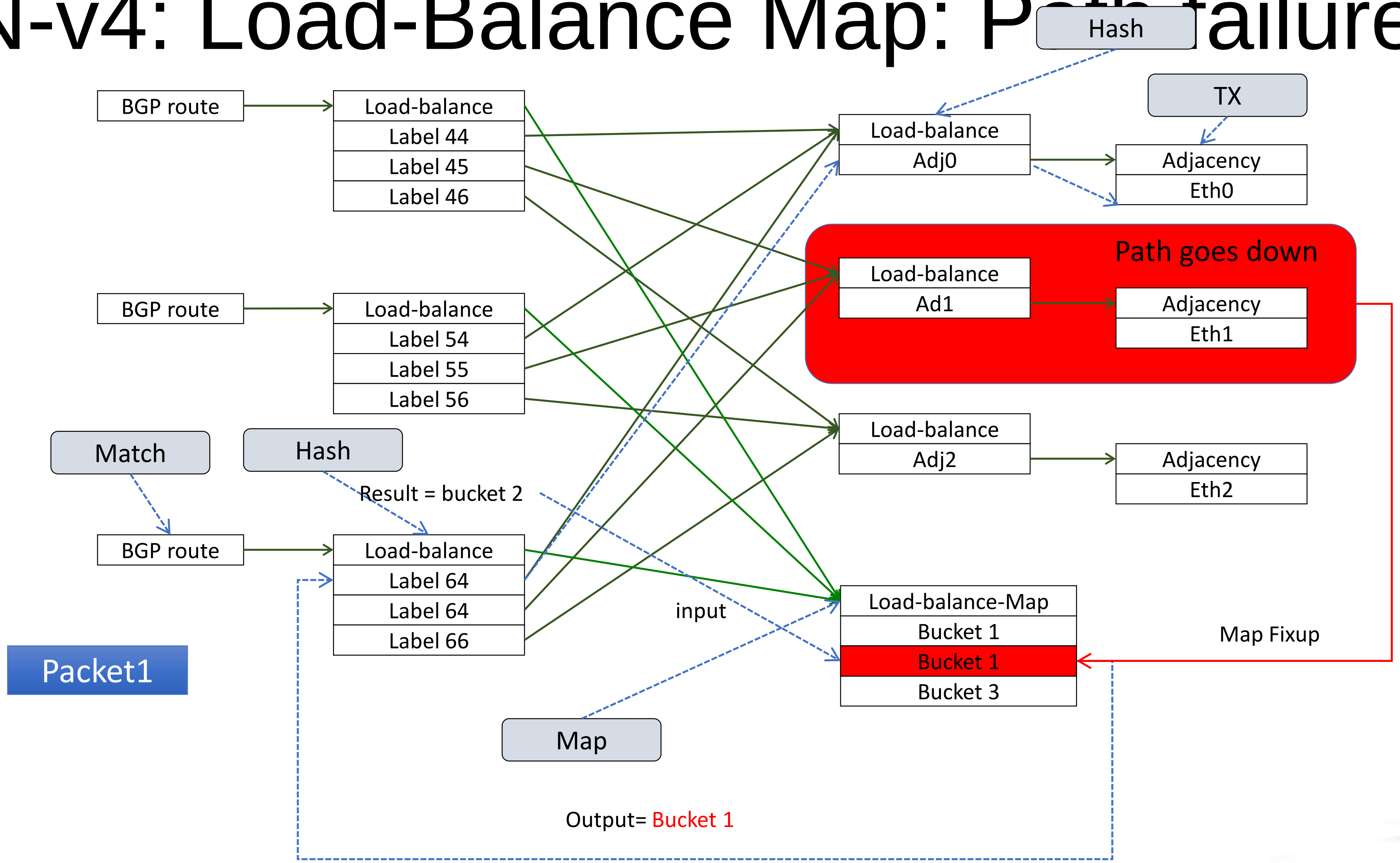
- A unique output label for each route on each path means that the load-balance choice for each route is different.
- Different choices mean the load-balance objects are not shared.
- No sharing means there is no common location where an in-place modify will affect all routes.
- PIC is broken.

VPN-v4: Load-Balance Map



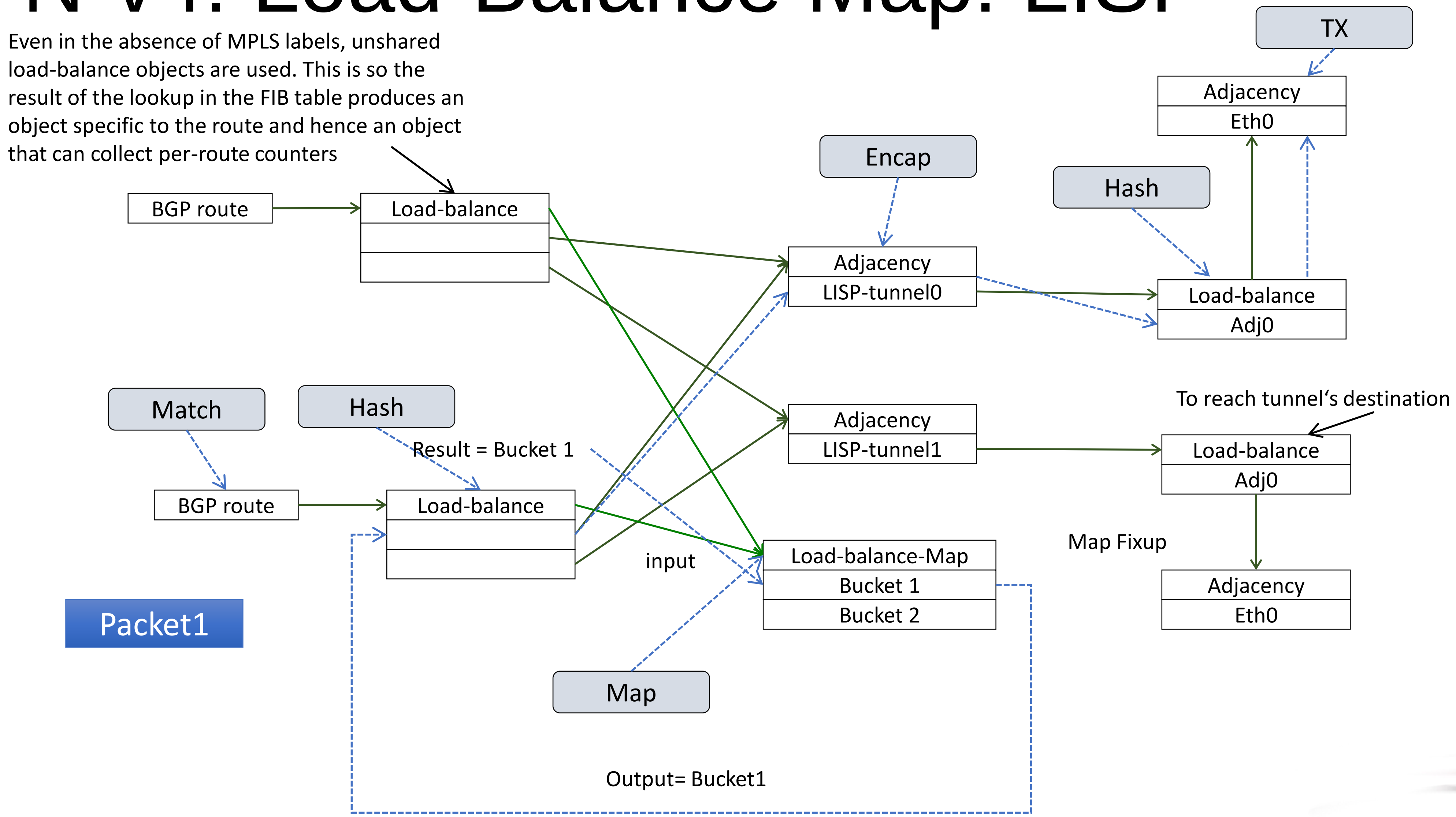
- Load-Balance Map translates from bucket indices that are unusable to bucket indices that are usable.

VPN-v4: Load-Balance Map: Path failure



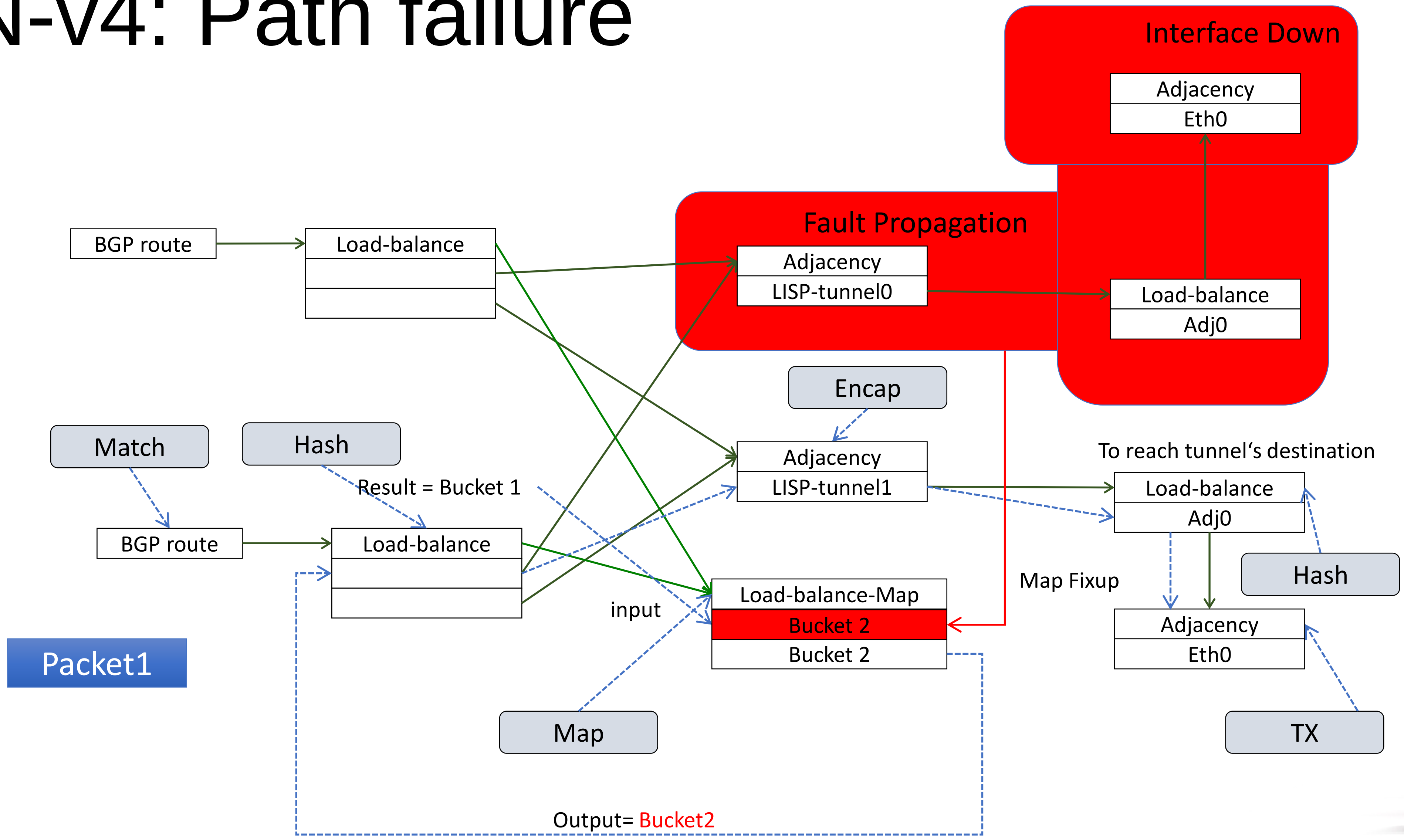
- When a path becomes unusable the load-balance map is updated to replace that path with one that is usable.
- Since it is a shared structure, this has the effect of making the path unusable for each route

VPN-v4: Load-Balance Map: LISP



- Adjacencies on the LISP tunnel apply the LISP tunnel encapsulation.
- They point to the load-balance object to reach the tunnel's destination in the underlay.

VPN-v4: Path failure



- A failure of an interface in the LISP underlay is propagated up the hierarchy.
- The underlay failure results in the LISP tunnel going down and the Map is updated to remove that tunnel from the ECMP set.