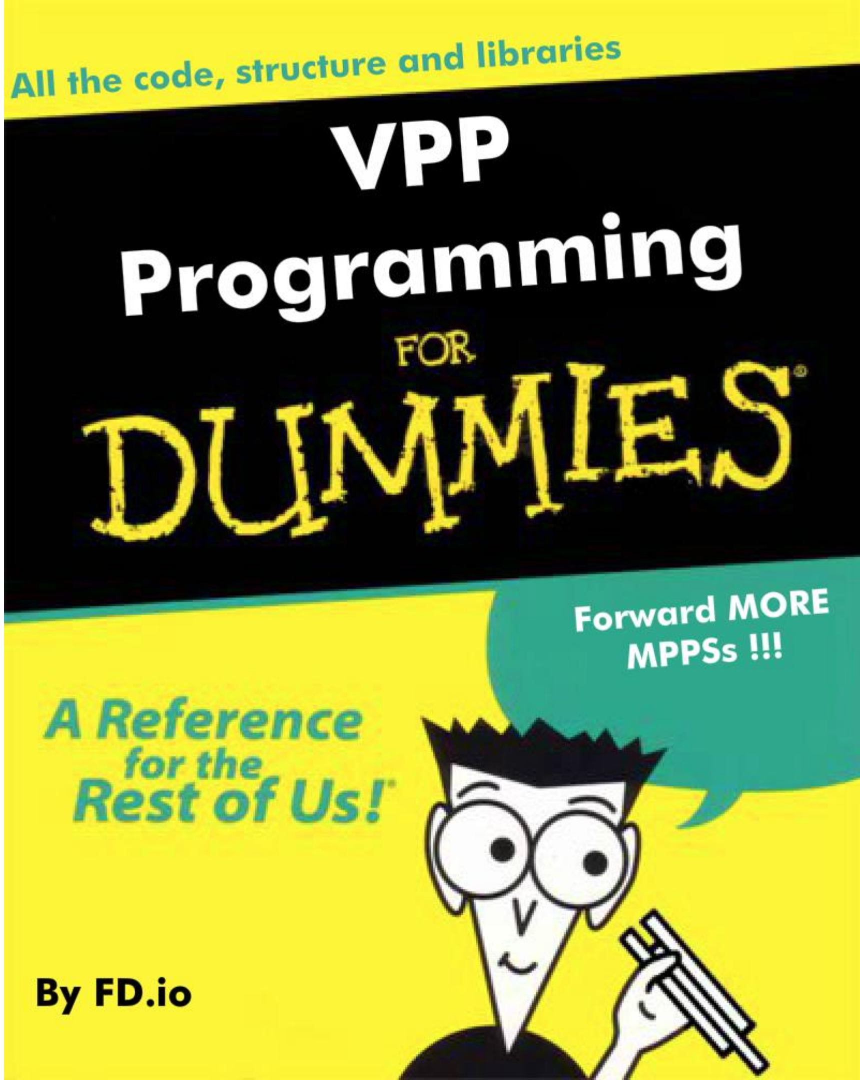
/dev/boot

Pierre Pfister | VPP Infrastructure Libraries

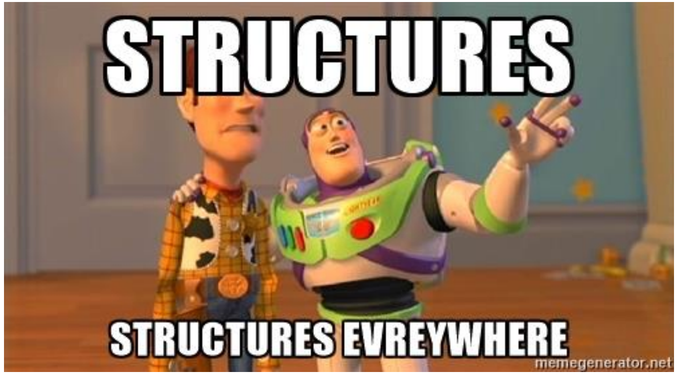
FD.io /dev/boot Training
31 May – 03 June | Télécom ParisTech

VPP Infrastructure Libraries

Warning: Extreme C coding


All the code, structure and libraries
VPP Programming FOR DUMMIES®
Forward MORE MPPSs !!!
A Reference for the Rest of Us!®
By FD.io

vppinfra structures



Sub-agenda

types.h

We use:

u8, u16, u32, u64, u32x4 (intrinsics)

i8, …

f32, f64

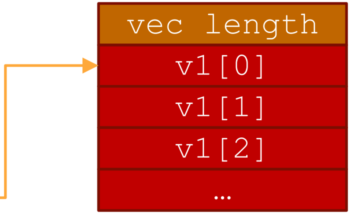
uword

vec.h

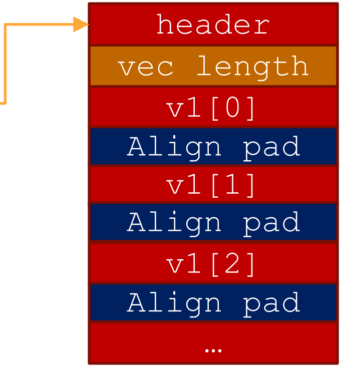
A vector is an auto-resized C array with some meta data:

Parameters: Type T, header size, element alignment

```
T e[5];
//Default alignment
//No header
T *v1 = 0; //Init
vec_add1(v1, &e[0]); //Add 1
vec_add2(v1, &e[1], 4); //Add 4
v1
```



```
//Header of 8 bytes
//Alignement of 16 bytes
T *v2 = 0; //Init
vec_add2_ha(v2, &e[0], 5, 8, 16);
vec_header(v2, 8);
```



```
vec_len(v1); //length safe when v1==0
_vec_len(v1) = 0; //length unsafe

//increase length to Ith elmt.
//Set new elements to 0.
vec_validate(V,I);

// Allocate for N more elements (but does not change leng
vec_alloc(V,N);

vec_free(v); //Free memory and set v to 0
vec_reset_length(v); //set length to 0 (safe)
```

Allocation only increases.

Vector origin pointer may changer !

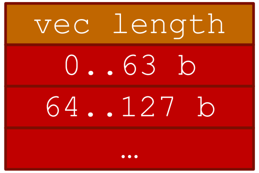
Store Indexes (not pointers) !

bitmap.h

Set and get bits with indexes. Lots of them.

Implemented as a uword vector.

```
uword *bitmap = 0;

//Allocate bitmap for 100 bits
clib_bitmap_alloc(bitmap, 100);

//Make room for 201 bits
clib_bitmap_validate(bitmap, 200);

clib_bitmap_set(bitmap, 33, 1);
clib_bitmap_get(bitmap, 33);

//Get bits from 3 to 12 included
uword a = clib_bitmap_get_multiple
            (bitmap, 3, 10);

clib_bitmap_set_multiple …
```



vec length

0..63 b

64..127 b

…

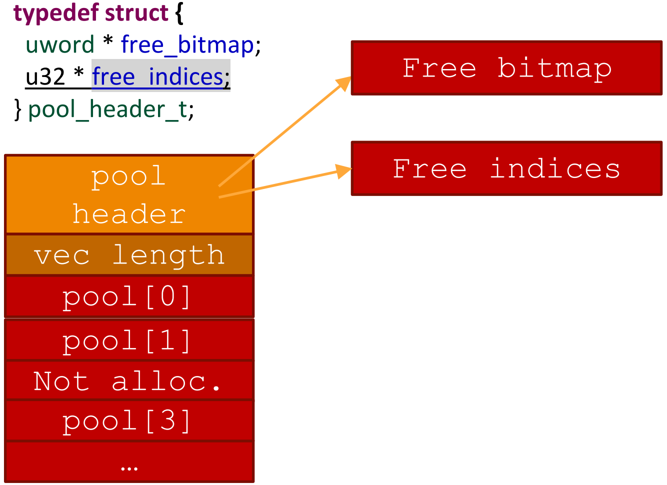
```
uword clib_bitmap_first_set (uword * ai);
uword clib_bitmap_first_clear (uword * ai);

uword clib_bitmap_next_set (uword * ai, uword i);
uword clib_bitmap_next_clear (uword * ai, uword i);

uword clib_bitmap_count_set_bits (uword * ai);

…
```

pool.h

Fixed sized element allocator.

Based on a vec and a bitmap.

```
T *pool = 0;
T *e;

pool_get(pool, e);
//pool_get_aligned(pool, e, 4);

//Query if element is free
pool_is_free(pool,e);
pool_is_free_index(pool, e – pool);

//Unalloc element
pool_put(pool,e);

//Free the whole pool
pool_free(pool);

// Allocated element count
pool_elts(pool);
```



```
typedef struct {
  uword * free_bitmap;
  u32 * free_indices;
} pool_header_t;
```

Keeps track of element allocation

Used for fast re-allocation

Sparse memory assignment.

Avoids memory fragmentation.

Fast.

heap.h

Variable element size allocator.

```
T *heap = 0;
T *e;

//Allocate 4xT.
U32 handle;
u32 offset = heap_alloc(heap, 4, handle);
//Allocated heap[offset] - heap[offset + 4]

//Object size
heap_size(heap, handle);

//Deallocate
heap_dealloc(heap, handle);
```

Rarely used (pools are faster).

Still efficient.

To be used if you need variably sized allocations.

e.g. classifier

bihash.h

Hashing table on steroïd.

Heart of the ipv6 fib

Key size

Value size

bihash_8_8.h

bihash_24_8.h

bihash_template.h

Code with macro magic to add your own bihash
(define BIHASH_TYPE as X_Y and include the header)

vppinfra
Parse’n’print

```
                      (  ) (@@) ( )  (@)  ()    @@    O     @     O
                 (@@@)
             (    )
          (@@@@)

        (   )
      ====        ________                ___________
  _D _|  |_______/        \__I_I_____===__|_________|
   |(_)---  |   H\________/ |   |        =|___ ___|      _________________
   /     |  |   H  |  |     |   |         ||_| |_||     _|
  |      |  |   H  |__--------------------| [___] |   =|
  | ________|___H__/__|_____/[][]~\_______|       |   -|
  |/ |   |-----------I_____I [][] []  D   |=======|____|________________________
__/ =| o |=-~O=====O=====O=====O\ ____Y___________|__|__________________________
 |/-=|___|=    ||    ||    ||    |_____/~\___/          |_D__D__D_|  |
  \_/      \__/  \__/  \__/  \__/      \_/               \_/   \_/
```

Sub-agenda

format.h

Printing and Scanning has never been so easy.

Format == write from data to strings

```
typedef u8 * (format_function_t) (u8 * s, va_list * args);

u8 * va_format (u8 * s, char * format, va_list * args);
u8 *    format (u8 * s, char * format, ...);
```

```
struct {
   u32 v1, v2;
} example;

u8 *format_example (u8 *s, va_list *args) {
   struct example *e = va_arg (*va, struct example *);
   return format(s, “(%d, %d)”, e->v1, e->v2);
}

u8 *format_example_with_u32(u8 *s, va_list *args) {
   struct example *e = va_arg (*va, struct example *);
   u32 v3 = va_arg (*va, u32);
   return format(s, “%U with %d”, format_example, e, v3);
}

struct example e = {1, 2};
u8 *str = format(0, “%U“ , format_example_with_u32, &e, 5);
clib_warning(“%s“, str);
vec_free(str);
```

Supports ‘most’ of printf syntax.

format.h

Printing and Scanning has never been so easy.

Unformat == get from string to data

```
typedef uword (unformat_function_t) (unformat_input_t * input, va_list * args);

uword unformat (unformat_input_t * i, char * fmt, ...);
uword va_unformat (unformat_input_t * i, char * fmt, va_list * args);
```

```
struct {
   u32 v1, v2;
} example;

uword unformat_example (unformat_input_t *i, va_list *args) {
   struct example *e = va_arg (*va, struct example *);
   return unformat(i, “(%d, %d)”, &e->v1, &e->v2);
}

uword unformat_example_with_u32(unformat_input_t *i, va_list *args) {
   struct example *e = va_arg (*va, struct example *);
   u32 *v3 = va_arg (*va, u32 *);
   return unformat(i, “%U with %d”, unformat_example, e, v3);
}

struct example e; u32 v;
const char *str = “(1, 2) with 3 foo bar”;
unformat_input_t i;
unformat_init_string(&i, str, strlen(str));
unformat(i, “%U“ , unformat_example_with_u32, &e, &v);
```

Supports ‘most’ of scanf syntax.
Returns wether unformat was succesfull.

In case of success, the data is consumed in the unformat_input.
Otherwise, the data is left untouched.

format.h

Printing and Scanning has never been so easy.

Parsing CLI input example

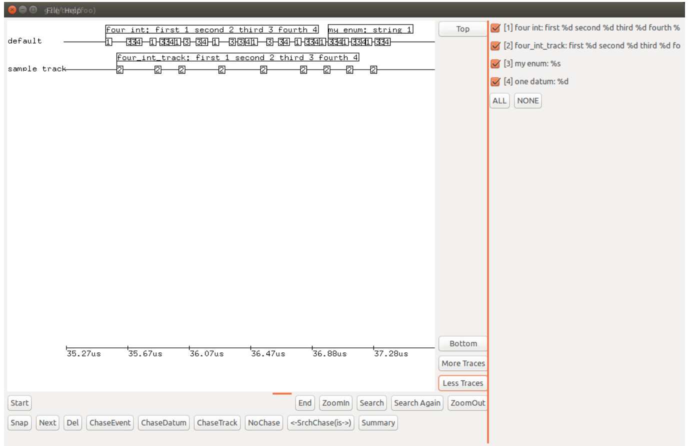
From vnet/vnet/l2tp/l2tp.c

Arguments can be in any order.

client <ip6_addr> our <ip6_addr> local-cookie <long> remote-cookie <long> …

```
if (! unformat_user (input, unformat_line_input, line_input))
  return 0;

 while (unformat_check_input (line_input) != UNFORMAT_END_OF_INPUT) {
  if (unformat (line_input, "client %U",
               unformat_ip6_address, &client_address))
   client_address_set = 1;
  else if (unformat (line_input, "our %U",
                unformat_ip6_address, &our_address))
   our_address_set = 1;
  else if (unformat (line_input, "local-cookie %llx", &local_cookie))
   ;
  else if (unformat (line_input, "remote-cookie %llx", &remote_cookie))
   ;
  else if (unformat (line_input, "local-session-id %d",
                &local_session_id))
   ;
  else if (unformat (line_input, "remote-session-id %d",
                &remote_session_id))
   ;
  else if (unformat (line_input, "l2-sublayer-present"))
   l2_sublayer_present = 1;
  else
   return clib_error_return (0, "parse error: '%U'",
                     format_unformat_error, line_input);
 }
```

vppinfra
Event Logger



```
four int: first 1 second 2 third 3 fourth 4
my enum: string 1
four_int_track: first 1 second 2 third 3 fourth 4
```

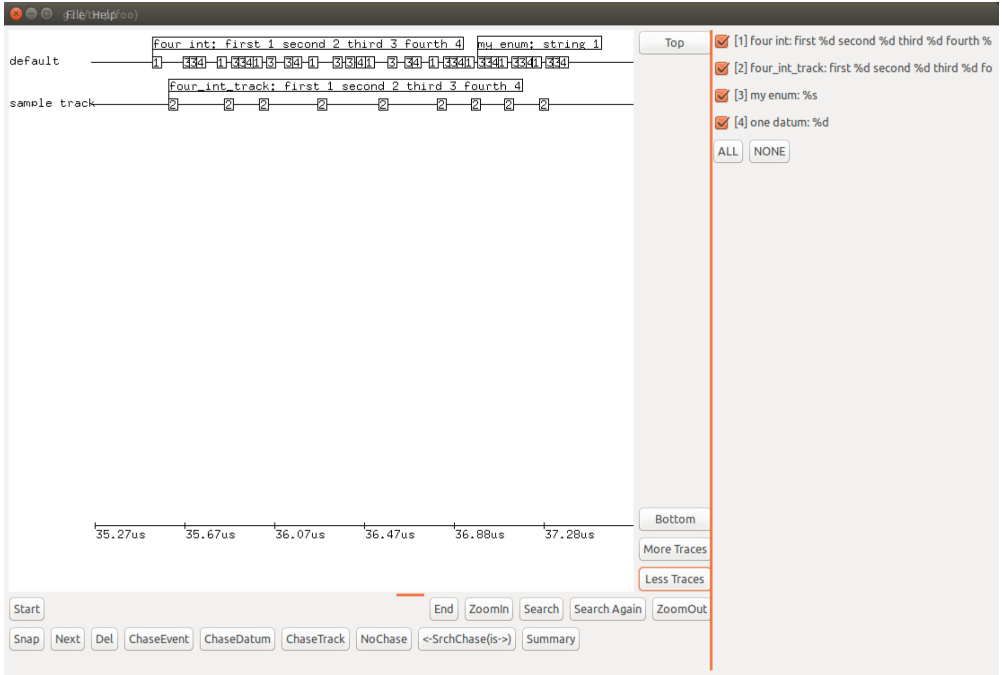
- ☑ [1] four int: first %d second %d third %d fourth %
- ☑ [2] four_int_track: first %d second %d third %d fo
- ☑ [3] my enum: %s
- ☑ [4] one datum: %d

Sub-agenda

elog.h

High speed event logging

https://wiki.fd.io/view/VPP/elog



☑ [1] four int: first %d second %d third %d fourth %

☑ [2] four_int_track: first %d second %d third %d fo

☑ [3] my enum: %s

☑ [4] one datum: %d

- Event-logging enabled in .../vlib/vlib/main.c:vlib_main(...)
- Use the elog_main_t in vlib_global_main, aka &vlib_global_main.elog_main
- Default ring size 128K events
- Thread safe—lock-free atomic increment to dole out event slots
- Each event-slot is 32 bytes: u64 time-in-cpu-clocks, u16 event-id, u16 track, 20 bytes of data
- Logging an event costs less than 100ns
- Observer effect: at most a couple of events per node, per frame at speed

Cf. Dave Barach’s presentation:
https://docs.google.com/presentation/d/1C_1zM5Z3sTibOj1e2pe_YDbiMCytwZspK541fPElyWM

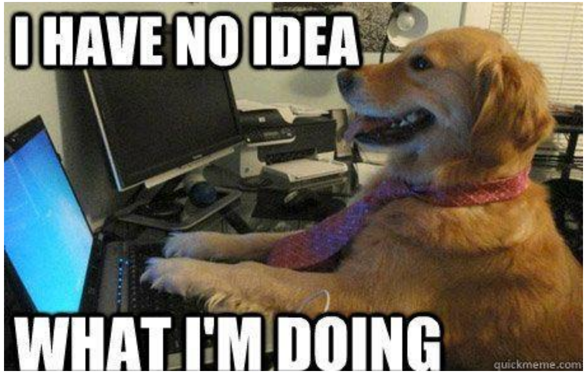
vppinfra
The rest

Sub-agenda

Things you may or may not learn by yourself

fheap.h	Fibonacci heap…
graph.h	Graph Implementation
hash.h	Hash table
phash.h	Again
qhash.h	Again
longjmp.h	We actually use it !

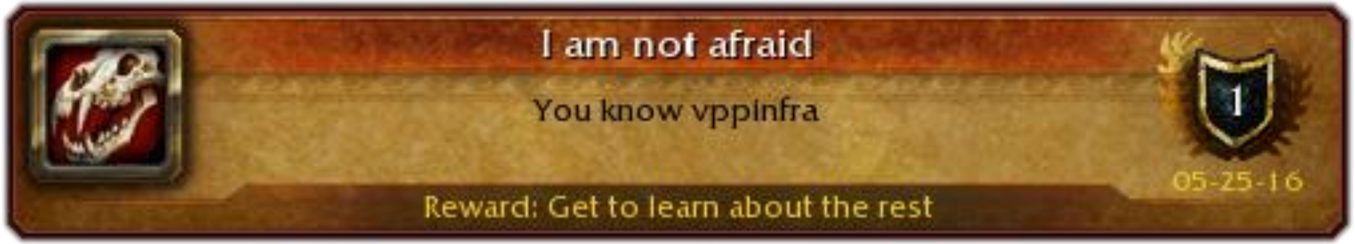
Etc…



Lessons from vppinfra

1. Store indexes, not pointers (arrays get resized).
2. Macros do change parameters (and not only pointers).
3. Format and unformat functions are usefull.
4. Alignement can be important for multi-thread efficiency.

Congratulation !



I am not afraid

You know vppinfra

Reward: Get to learn about the rest

05-25-16

vlib

Sub-agenda

Vlib Overview

VPP is based on 2 main ideas:

1. Vectors (0 to 256 buffers bundled together)
2. Nodes (in a graph) performing actions and passing the vectors.

vlib is where all this abstract model is defined. Outside of any networking considerations (No IP, no ethernet, etc…).

It also includes:

- Counters
- CLI
- Scheduler (main loop).
- A few other things...

Vlib nodes

i-am-a-vlib-node

Sub-agenda

Vlib nodes - node.h – node_funcs.h

Nodes have:

• A name	dpdk-rx, ip6-forward, tapcli-tx, …
• An index	Uniquely identifies a node
• A function	Node callback to operate on vectors
• A type	
• A set of next nodes	Identified by index & counters
• A set of errors	With names & counters

Each node has a registration data (`vlib_node_registration_t`) and a runtime data (`vlib_node_t`).

Nodes are created with constructors at initialization.
It is possible to dynamically add next nodes.

```
always_inline uword
vlib_node_add_next (vlib_main_t * vm, uword node,
uword next_node);
```

```
typedef enum {
  VLIB_NODE_TYPE_INTERNAL,
  VLIB_NODE_TYPE_INPUT,
  VLIB_NODE_TYPE_PRE_INPUT,
  VLIB_NODE_TYPE_PROCESS,
  VLIB_N_NODE_TYPE,
} vlib_node_type_t;
```

```
VLIB_REGISTER_NODE (l2input_node) = {
  .function = l2input_node_fn,
  .name = "l2-input",
  .vector_size = sizeof (u32),
  .format_trace = format_l2input_trace,
  .format_buffer = format_ethernet_header_with_length,
  .type = VLIB_NODE_TYPE_INTERNAL,
  .n_errors = ARRAY_LEN(l2input_error_strings),
  .error_strings = l2input_error_strings,
  .n_next_nodes = L2INPUT_N_NEXT,
  .next_nodes = {
      [L2INPUT_NEXT_LEARN] = "l2-learn",
      [L2INPUT_NEXT_FWD]   = "l2-fwd",
      [L2INPUT_NEXT_DROP]  = "error-drop",
  },
};
```

Vlib nodes - *VLIB_NODE_TYPE_INTERNAL*

Most typical node receiving buffer vectors, performing operations.

Includes tx nodes.

Vlib nodes - *VLIB_NODE_TYPE_INPUT*

Typically device input nodes.
Create frames from scratch and dispatch to internal nodes.

```
typedef enum {
 VLIB_NODE_STATE_POLLING,
 VLIB_NODE_STATE_INTERRUPT,
 VLIB_NODE_STATE_DISABLED,
 VLIB_N_NODE_STATE,
} vlib_node_state_t;
```

Constantly called

Called once when interrupted

Not called

```
u32
input_main_loops_per_call;
```

Throttle polling

Vlib nodes - *VLIB_NODE_TYPE_PRE_INPUT*

Called before input nodes.

Not used as far as I know.

Vlib nodes - *VLIB_NODE_TYPE_PROCESS*

Thread-like function.
Can be suspended, wait for events, be resumed…
(based on setjump/longjump).

Wait for an event

```
always_inline f64
vlib_process_wait_for_event_or_clock (
                                vlib_main_t * vm, f64 dt)
```

Send an event

```
always_inline void
vlib_process_signal_event (vlib_main_t * vm,
                uword node_index,
                uword type_opaque, uword data);
```

```
static uword ip6_icmp_neighbor_discovery_event_process (vlib_main_t * vm,
vlib_node_runtime_t * node, vlib_frame_t * frame)
{
  uword event_type;
  ip6_icmp_neighbor_discovery_event_data_t * event_data;
  while (1) {
     vlib_process_wait_for_event_or_clock (vm,  1. /* seconds */);
     event_data = vlib_process_get_event_data (vm,  &event_type);
     if(!event_data) {
                        ip6_neighbor_process_timer_event (vm,  node,  frame);
                      } else {
                        switch (event_type) {
                        case ICMP6_ND_EVENT_INIT:
                          break;
                        case ~0:
                          break;
                        default:
                          ASSERT (0);
                        }
                        if (event_data)
                          _vec_len (event_data) = 0;
                      }
    }
  return frame->n_vectors;
}
```

Vlib/unix

Sub-agenda

Vlib and file descriptors

Vlib implements a poll – event-loop-like node.
Listens for fd events and execute callbacks.
Somehow slow due to `node->input_main_loops_per_call`

```
typedef struct unix_file {
  u32 file_descriptor;

  u32 flags;
#define UNIX_FILE_DATA_AVAILABLE_TO_WRITE (1 << 0)
#define UNIX_FILE_EVENT_EDGE_TRIGGERED   (1 << 1)

 uword private_data;

  unix_file_function_t * read_function, * write_function, *
error_function;
} unix_file_t;
```

Example from vnet/unix/tapcli.c

```
 unix_file_t template = {0};
   template.read_function = tapcli_read_ready;
   template.file_descriptor = dev_net_tun_fd;
   ti->unix_file_index = unix_file_add (&unix_main,
&template);
```

```
static clib_error_t * tapcli_read_ready (unix_file_t * uf)
{
 vlib_main_t * vm = vlib_get_main();
 tapcli_main_t * tm = &tapcli_main;
 uword * p;
 vlib_node_set_interrupt_pending (vm, tapcli_rx_node.index);
 …
 return 0;
}
```

```
uword unix_file_add (unix_main_t * um, unix_file_t * template);
```

See more at vlib/unix/unix.h

Plugins vlib/unix/plugin.h

Plugins are loaded at VPP init:
$ vpp plugin_path <path>

Vpp tries to load all libraries present in the path.

`vlib_plugin_register` must exist and return 0.

’constructor’ functions are also executed.

```
clib_error_t * constr_example (vlib_main_t * vm);
VLIB_INIT_FUNCTION (constr_example);
```

```
static int
load_one_plugin (plugin_main_t *pm, plugin_info_t *pi, int from_early_init)
{
  void *handle, *register_handle;
  clib_error_t * (*fp)(vlib_main_t *, void *, int);
  clib_error_t * error;
  void *handoff_structure;

  handle = dlopen ((char *)pi->name, RTLD_LAZY);
  […]
  pi->handle = handle;

  register_handle = dlsym (pi->handle, "vlib_plugin_register");
  if (register_handle == 0) {
     dlclose (handle);
     return 0;
   }
  fp = register_handle;
  handoff_structure = vnet_get_handoff_structure();
  if (handoff_structure == 0)
    error = clib_error_return (0, "handoff structure callback returned 0");
  else
    error = (*fp)(pm->vlib_main, handoff_structure, from_early_init);

  if (error) {
     clib_error_report (error);
     dlclose (handle);
     return 1;
   }

  clib_warning ("Loaded plugin: %s", pi->name);
  return 0;
}
```

CLI

To be explained in more details in CLI and API session

Vlib buffers

DPDK
DATA PLANE DEVELOPMENT KIT

Or not DPDK DATA PLANE DEVELOPMENT KIT

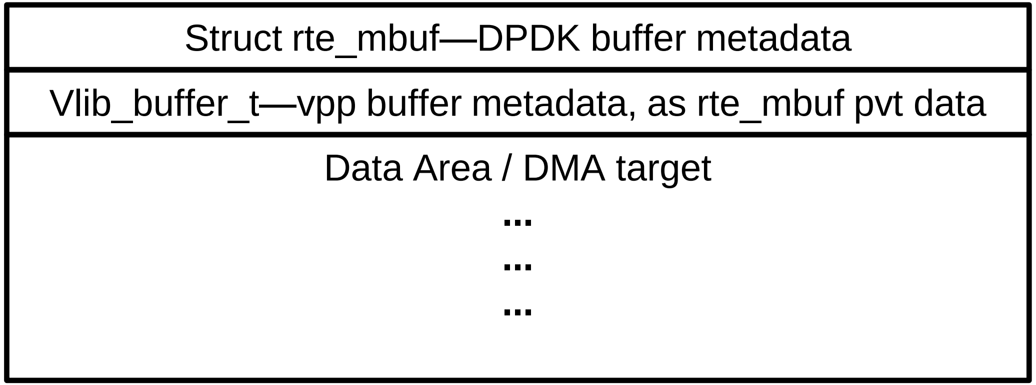
That is the question

Sub-agenda

Vlib buffers – vlib/buffer.[ch] vlib/dpdk_buffer.c

`vlib_buffer_t` is VPP’s ‘packet’ basic structure (equivalent to DPDK’s `rte_mbuf`).

When DPDK == 1, `vlib_buffer_t` is encapsulated within `rte_mbuf`.



Struct rte_mbuf—DPDK buffer metadata

Vlib_buffer_t—vpp buffer metadata, as rte_mbuf pvt data

Data Area / DMA target

…

…

…

```
#define vlib_buffer_from_rte_mbuf(x) ((vlib_buffer_t *)(x+1))

#define rte_mbuf_from_vlib_buffer(x) (((struct rte_mbuf *)x) - 1)
```

Vlib buffers

i16 current_data;	Signed offset in data[], pre_data[]
u16 current_length;	Nbytes between current data and the end of this buffer.
u32 flags;	#define **VLIB_BUFFER_IS_TRACED** (1 << 0) #define VLIB_BUFFER_LOG2_NEXT_PRESENT (1) #define **VLIB_BUFFER_NEXT_PRESENT** \\ (1 <<VLIB_BUFFER_LOG2_NEXT_PRESENT) #define VLIB_BUFFER_IS_RECYCLED (1 << 2) #define **VLIB_BUFFER_TOTAL_LENGTH_VALID** \\ (1 << 3) #define VLIB_BUFFER_HGSHM_USER_INDEX_VALID (1 << 4) #define VLIB_BUFFER_REPL_FAIL (1 << 5) #define LOG2_VLIB_BUFFER_FLAG_USER(n) (32 - (n))

Vlib buffers

u32 free_list_index;	Barely used
u32 total_length_not_including_first_buffer;	End of vlib_buffer_init_for_free_list() init
u32 next_buffer;	Buffer index of next buffer (if present)
u32 trace_index;	Valid if buffer traced
u32 clone_count;	If non-zero, xmit copy and recycle the original buffer
vlib_error_t error;	u16 error_node, error_code – set to arrange counter bump
u32 opaque[8];	subgraph metadata, see .../vnet/vnet/buffer.h

Vlib buffers

u32 opaque2[16];	Barely used
u8 pre_data[VLIB_BUFFER_PRE_DATA_SIZE];	Rewrite space, 128 bytes
u32 data[0];	Aligned DMA target. Certain hardware devices DMA into data[n], e.g. n=6.

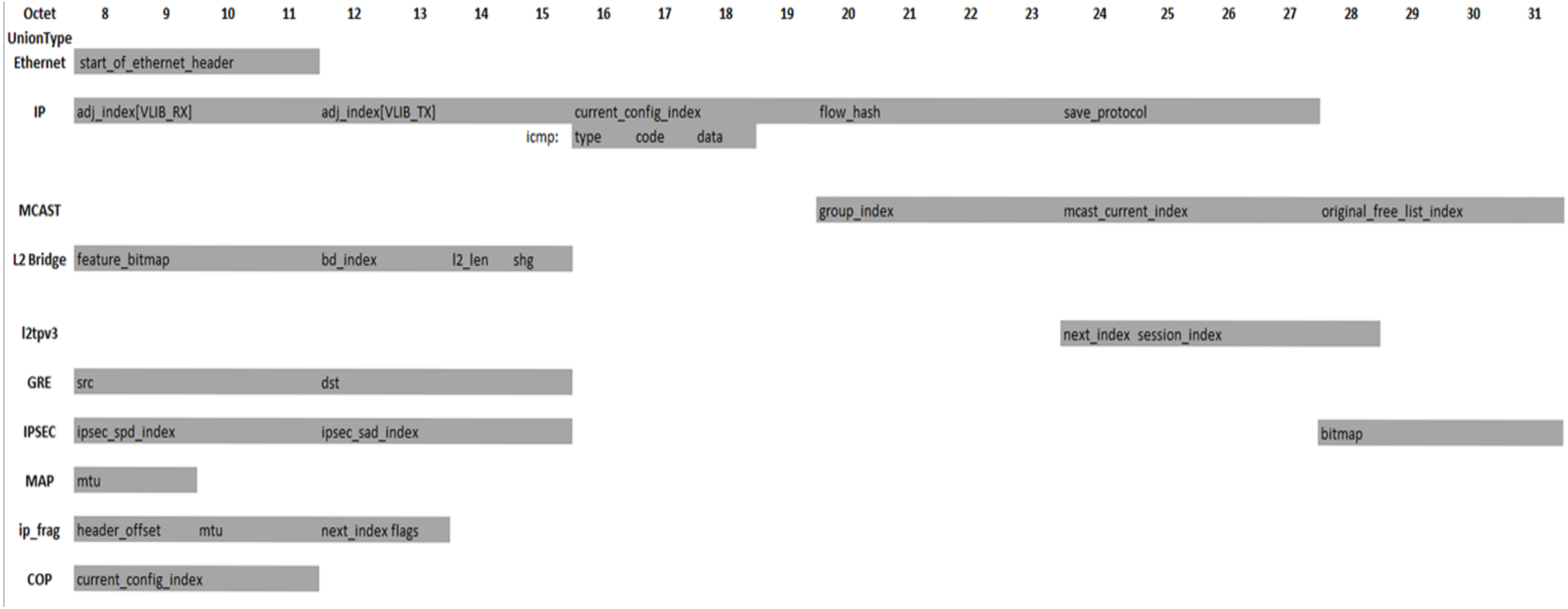
Vlib buffers

vnet_buffer_opaque_t – union of types which contain ‘subgraph’ metadata

- 8x32 bits
- vnet_buffer(b)->sw_if_index[VLIB_RX, VLIB_TX] are common
- From driver level: vnet_buffer(b)->sw_if_index[VLIB_RX] set to RX (physical) interface
  vnet_buffer(b)->sw_if_index[VLIB_TX] = ~0

- ip4/6-lookup use VLIB_TX as FIB index override.

- These are like node input/output metadata (e.g. ip-frag).



Octet	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
UnionType																								
Ethernet	start_of_ethernet_header																							
IP	adj_index[VLIB_RX]				adj_index[VLIB_TX]				current_config_index				flow_hash				save_protocol							
								icmp:	type	code	data													
MCAST													group_index				mcast_current_index				original_free_list_index			
L2 Bridge	feature_bitmap				bd_index		l2_len	shg																
l2tpv3																	next_index	session_index						
GRE	src				dst																			
IPSEC	ipsec_spd_index				ipsec_sad_index																bitmap			
MAP	mtu																							
ip_frag	header_offset		mtu		next_index flags																			
COP	current_config_index																							

Vnet_buffer_opaque_t intersection analysis

You nailed it !