# OvS and Socket Maps

## Beyond ports as endpoints

Aaron Conole
aconole@redhat.com



Red Hat

# What we'll discuss today

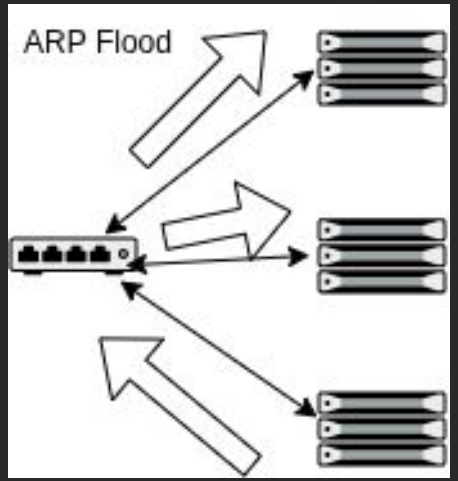
- Layers - How they (don't) matter
- Bypasses - Intelligently skipping stuff
- Direct forwarding
- Numbers - performance related stuff
- (Apologies - I didn't use AI and I'm bad at pictures)



# Open vSwitch - Is it a L2 switch?

Sometimes we need ot know whether things are what they

- Not exactly, BUT
- NORMAL action - usually just L2
- Packet comes in, ARP flood, etc.
- FDB only (No routing details)

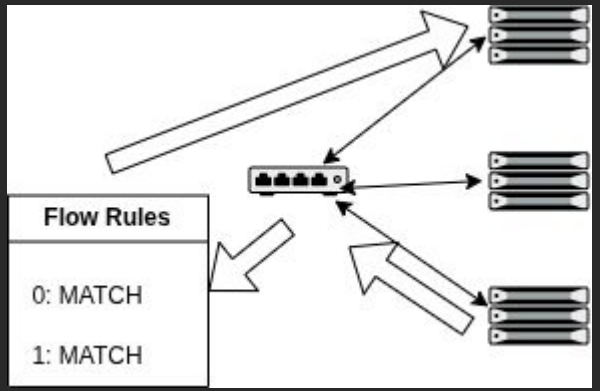





# Open vSwitch - Layer 3, it's there!

ever Forget the 'routing' we can do!

- CONTROLLER/learn action, and advanced flow rules - matching and rewriting!
- Can fake out multiple subnets
- Acts similar to a router - IPs can be set on bridge ports or patch ports, etc.

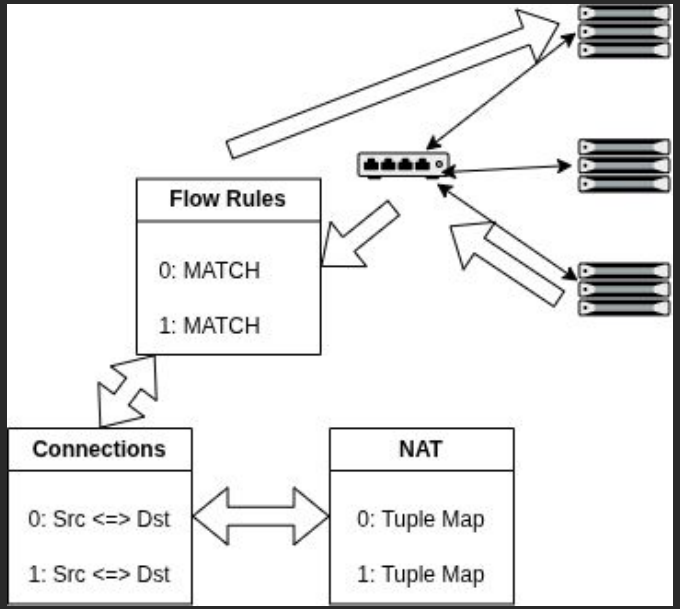





# Open vSwitch - “Now with CT” (for the last 10 years)

ever Forget the 'routing' we can do!

- CT action lets us do Stateful NAT, firewalling, etc.
- Connection tracking tables with windowing etc.
- L4 kindof (helpers, related, etc.)



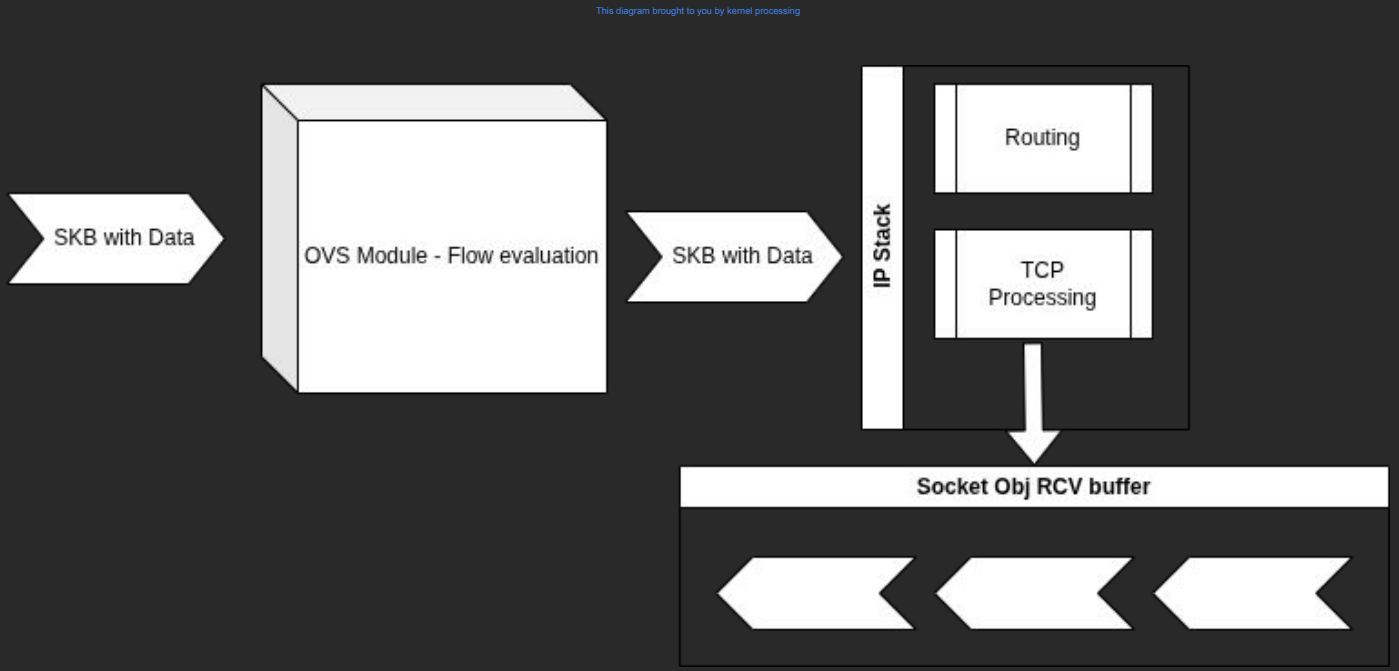
# Open vSwitch - Layers Are Useful for Diagramming

Is it a sandwich? Philosophers hate this one simple question

- Did you like the previous images? They look reasonable!
- When we implement packet movement using those concepts, it makes a big sandwich!



# What's happening really (why layers aren't useful for throughput)



# Current pipeline - For every packet, we go through layers

Is it a sandwich? Philosophers hate this one simple question

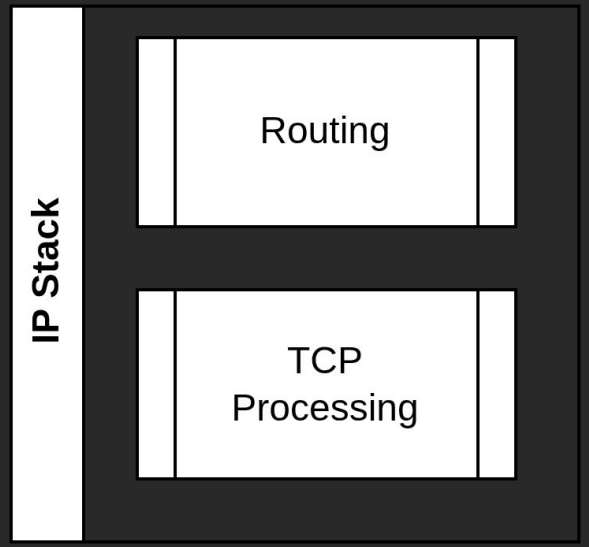
- Downsides - every packet gets recirculated a bunch.
- Overhead for re-parsing when we need to extract more layer-related fields.
- Protocol work that has to happen
- Pods are on the same host, same memory, etc.
- All the memory is “local”
- Why do all the protocol work that has to happen?



# What is the output providing us for the price?

Besides output - which is nice, but too generic for me

- The vswitch can already “know” where a packet is destined from tuple details
- ovs-vswitchd knows about IP/TCP information - ct tables, etc.
- What does the stack provide “extra”?



# How much does it cost?

Is it a sandwich? Philosophers hate this one simple question

- Measurement: roughly 1000ns (of a total 7395ns) are spent processing the the protocol overhead - 1 uS (more numbers coming) - these are 65k pkts
- Netns to netns, using simple output action (pre-programmed with ovs-dpctl.py)

```
[core at localhost ~]$ sudo ip netns exec left ./git/iperf3/src/iperf3 -s
-----------------------------------------------------------
Server listening on 5201 (test #1)
-----------------------------------------------------------
Accepted connection from 172.31.110.2, port 55942
[  5] local 172.31.110.1 port 5201 connected to 172.31.110.2 port 55956
[ ID] Interval           Transfer     Bitrate
[  5]   0.00-1.00   sec  8.40 GBytes  72.0 Gbits/sec
[  5]   1.00-2.00   sec  8.66 GBytes  74.4 Gbits/sec
[  5]   2.00-3.00   sec  8.52 GBytes  73.2 Gbits/sec
[  5]   3.00-4.00   sec  8.48 GBytes  72.9 Gbits/sec
[  5]   4.00-5.00   sec  8.60 GBytes  73.9 Gbits/sec
[  5]   5.00-6.00   sec  8.46 GBytes  72.7 Gbits/sec
[  5]   6.00-7.00   sec  8.46 GBytes  72.7 Gbits/sec
[  5]   7.00-8.00   sec  8.45 GBytes  72.6 Gbits/sec
[  5]   8.00-9.00   sec  8.39 GBytes  72.1 Gbits/sec
[  5]   9.00-10.00  sec  8.40 GBytes  72.2 Gbits/sec
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bitrate
[ 5] 0.00-10.00 sec 84.8 GBytes 72.9 Gbits/sec receiver
-----------------------------------------------------------
Server listening on 5201 (test #2)
-----------------------------------------------------------
```



# Alternate approach(es) - skip the IP stack

Skipping things is all the rage now!

- RFC to generate some discussion at

https://mail.openvswitch.org/pipermail/ovs-dev/2025-June/424390.html

- 4 different approaches (maybe there are more?) proposed

```
                if (unlikely(err)) {
diff --git a/net/openvswitch/datapath.h b/net/openvswitch/datapath.h
index cfeb817a1889..90279cb0adbd 100644
--- a/net/openvswitch/datapath.h
+++ b/net/openvswitch/datapath.h
@@ -67,6 +67,50 @@  struct dp_nlsk_pids {
        u32 pids[];
 };

+enum ovs_sk_map_key_select {
+       OVS_SK_MAP_KEY_UNSET,
+       OVS_SK_MAP_KEY_INPUT_SOCKET_BASED,
+       OVS_SK_MAP_KEY_TUPLE_BASED,
+
+       OVS_SK_MAP_KEY_MAX__
+};
+
+/**
+ * struct ovs_skb_sk_map_data - OVS SK Map lookup data
+ * @key_type: Select whether to use input_socket based map or use the 5-tuple.
+ * @key: Union of input_socket vs 5-tuple.
+ */
+struct ovs_skb_sk_map_data {
+       enum ovs_sk_map_key_select key_type;
+       union {
+               struct sock *input_socket;
+               struct {
+                       union {
+                               struct {
+                                       __be32 src;     /* IP4 source address. */
+                                       __be32 dst;     /* IP4 destination address. */
+                               } ipv4;
+                               struct {
+                                       struct in6_addr src;    /* IP6 source address. */
+                                       struct in6_addr dst;    /* IP6 destination address. */
+                                       __be32 label;           /* IP6 flow labe. */
+                               } ipv6;
+                       } ip;
+                       struct {
+                               __be16 src;     /* TCP/UDP/SCTP src port. */
+                               __be16 dst;     /* TCP/UDP/SCTP dst port. */
+                       } tp;
+                       u8 protocol;            /* IPPROTO_*. */
+               } tuple;
+       } key;
+};
```



# How much does it cost NOW?

Is it a sandwich? Philosophers hate this one simple question

- Measurement: 6190ns per packet, and ~23,000 65k packets more per-second (actually more than that, this is slightly napkin'd)
- Netns to netns, using simple output action (pre-programmed with ovs-dpctl.py) this time with **sock(try)**

```
[core at localhost ~]$ sudo ip netns exec left ./git/iperf3/src/iperf3 -s
-----------------------------------------------------------
Server listening on 5201 (test #1)
-----------------------------------------------------------
Accepted connection from 172.31.110.2, port 50794
[  5] local 172.31.110.1 port 5201 connected to 172.31.110.2 port 50806
[ ID] Interval           Transfer     Bitrate
[  5]   0.00-1.00   sec  9.57 GBytes  82.1 Gbits/sec
[  5]   1.00-2.00   sec  9.49 GBytes  81.6 Gbits/sec
[  5]   2.00-3.00   sec  9.71 GBytes  83.4 Gbits/sec
[  5]   3.00-4.00   sec  9.75 GBytes  83.8 Gbits/sec
[  5]   4.00-5.00   sec  10.0 GBytes  86.3 Gbits/sec
[  5]   5.00-6.00   sec  9.95 GBytes  85.4 Gbits/sec
[  5]   6.00-7.00   sec  9.97 GBytes  85.7 Gbits/sec
[  5]   7.00-8.00   sec  10.0 GBytes  86.1 Gbits/sec
[  5]   8.00-9.00   sec  9.84 GBytes  84.5 Gbits/sec
[  5]   9.00-10.00  sec  9.95 GBytes  85.5 Gbits/sec
[  5]  10.00-10.00  sec   512 KBytes  11.0 Gbits/sec
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bitrate
[ 5] 0.00-10.00 sec 98.3 GBytes 84.4 Gbits/sec receiver
-----------------------------------------------------------
Server listening on 5201 (test #2)
-----------------------------------------------------------
```



# What about Conntrack?

Is it a sandwich? Philosophers hate this one simple question

- Measurement: 8254ns per packet.

  LOL. LMAO.

- The results with the socket map like approach (using sock-try to skip the ct calls) were almost identical to the output case.

```
>   [core@localhost ~]$ sudo ip netns exec right ./git/iperf3/src/iperf3
>   [  5] local 172.31.110.1 port 5201 connected to 192.168.0.21 port 58960
>   [ ID] Interval           Transfer     Bitrate         Retr  Cwnd
>   [  5]   0.00-1.00   sec  6.99 GBytes  59.9 Gbits/sec    0   4.11 MBytes
>   [  5]   1.00-2.00   sec  7.32 GBytes  62.9 Gbits/sec    0   4.11 MBytes
>   [  5]   2.00-3.00   sec  7.25 GBytes  62.2 Gbits/sec    0   4.11 MBytes
>   [  5]   3.00-4.00   sec  7.37 GBytes  63.4 Gbits/sec   41   4.11 MBytes
>   [  5]   4.00-5.00   sec  7.30 GBytes  62.6 Gbits/sec    0   4.11 MBytes
>   [  5]   5.00-6.00   sec  7.52 GBytes  64.7 Gbits/sec    0   4.11 MBytes
>   [  5]   6.00-7.00   sec  7.05 GBytes  60.5 Gbits/sec    0   4.11 MBytes
>   [  5]   7.00-8.00   sec  7.38 GBytes  63.4 Gbits/sec    0   4.11 MBytes
>   [  5]   8.00-9.00   sec  7.45 GBytes  63.9 Gbits/sec    0   4.11 MBytes
>   [  5]   9.00-10.00  sec  7.38 GBytes  63.5 Gbits/sec    0   4.11 MBytes
>   - - - - - - - - - - - - - - - - - - - - - - - - -
>   [ ID] Interval           Transfer     Bitrate         Retr
>   [  5]   0.00-10.00  sec  73.4 GBytes  63.0 Gbits/sec   41             sender
>   [  5]   0.00-10.00  sec  73.4 GBytes  63.0 Gbits/sec                  receiver
```



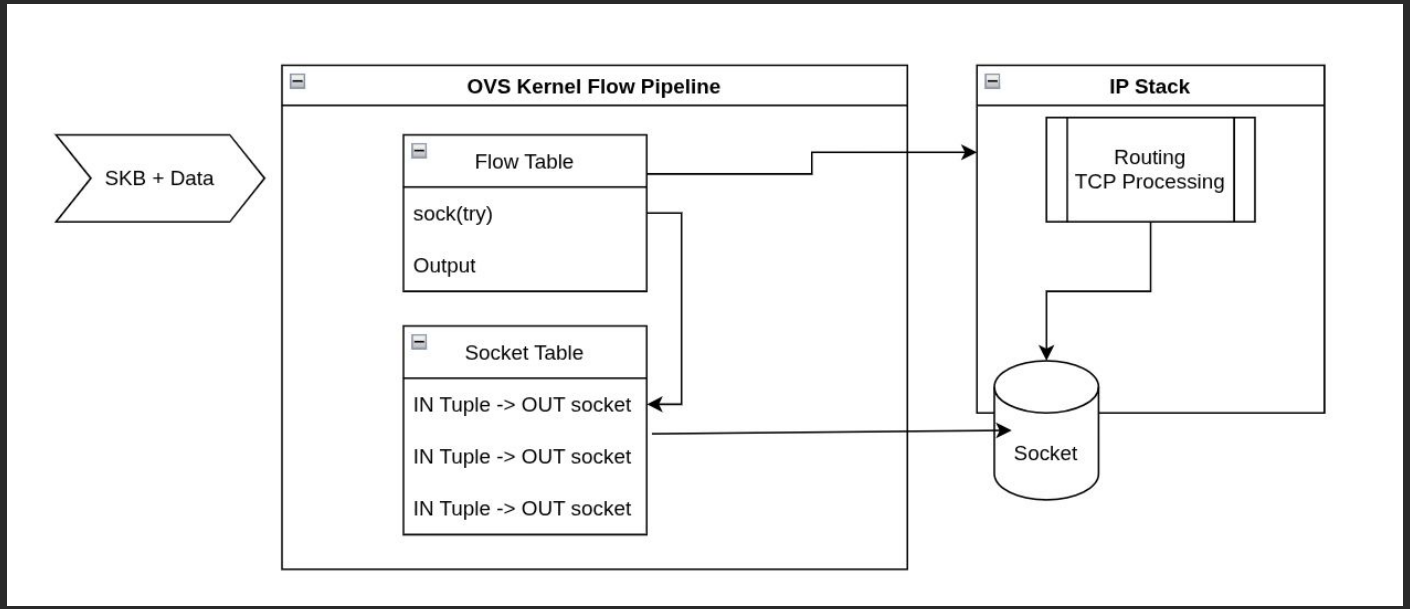
# Sample Datapath Flow Rules

Just a sample, yo!

```
in_port(1),eth(),eth_type(0x806),arp() actions=2
in_port(2),eth(),eth_type(0x806),arp() actions=1
recirc_id(0),in_port(1),eth(),eth_type(0x800),ipv4(),tcp_flags()
        sock(tuple),sock(try,recirc=0x1)
recirc_id(1),in_port(1),eth(),eth_type(0x800),ipv4(),tcp_flags() sock(commit,2),2
```

# In-Datapath Table approach

I tried this - it works






# In-Kernel Approach

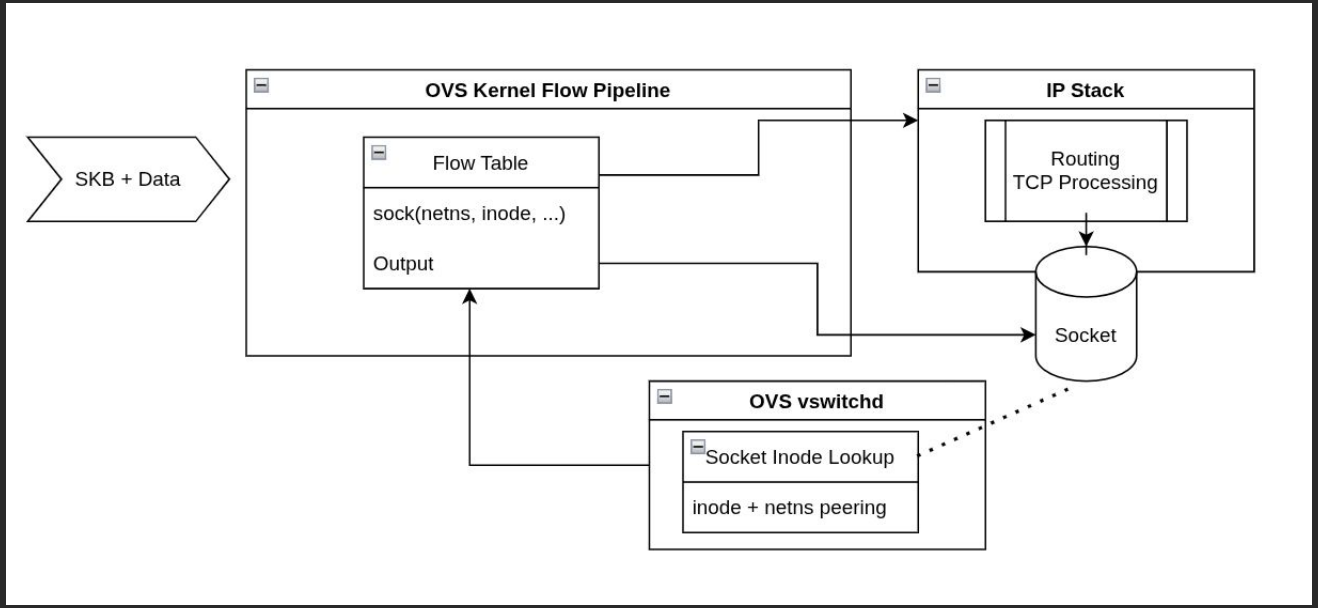
Good and bad

- Upside - easy to think about.
- Upside - Not too much work in userspace (can work easily with megaflows)
- Upside - Preserves flow centric nature of OVS
- Downside - Need to have the kernel manage a table
- Downside - Need to also manage socket object references
- Downside - Yet another cache
- Downside - Getting table state



# Userspace Centric Approach

Pulling this off is hard






# Userspace Centric Approach

On the plus side, it's cool!

- Upside - simple, can always see the sockets in-use from a flow dump
- Upside - Kernel doesn't need to do extra management, the lifecycles are regulated
- Upside - Looks "closer" to eBPF Socket Map

- Downside - Userspace implementation is HARD!
- Downside - Tracking socket references is HARD
- Downside - Need narrow tuples - are flows more expensive than socket entries?



# The userspace modification side really is HARD

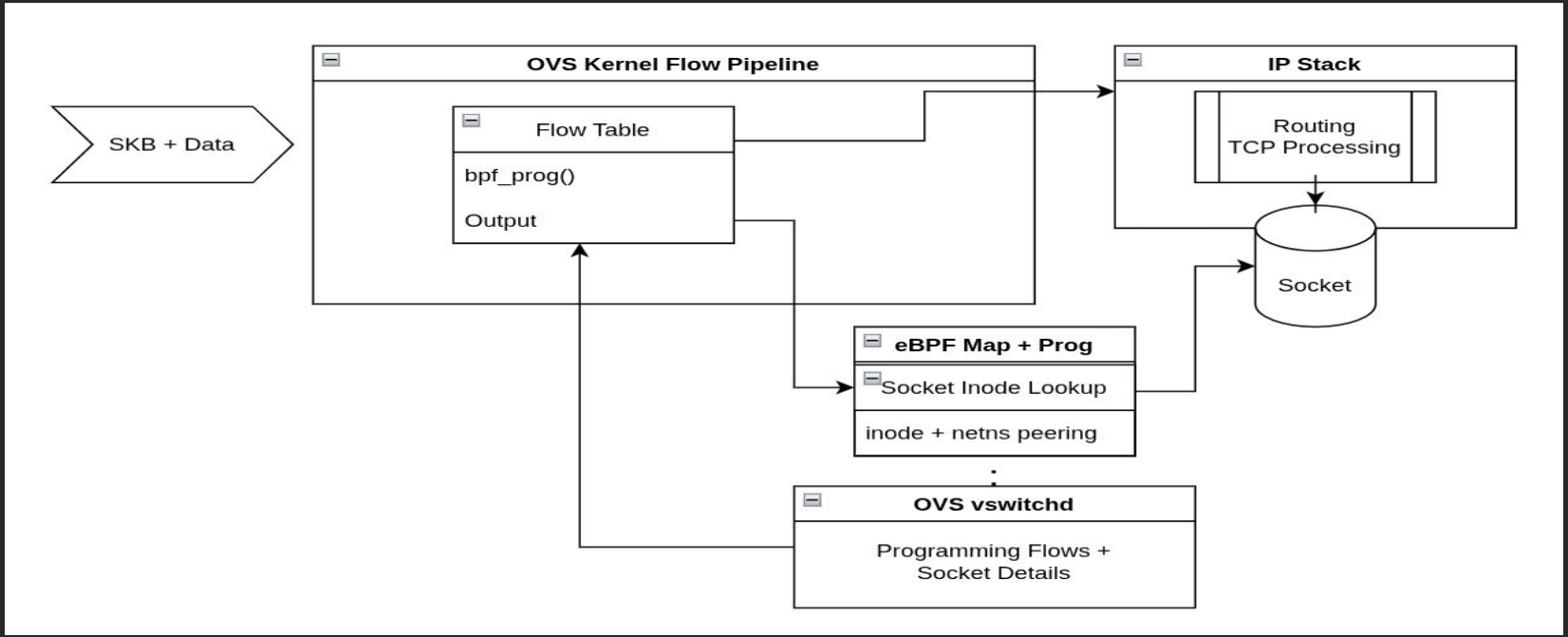
I didn’t emphasize it enough in previous slide, it’s very much a nightmare:

1. Tracking sockets - we can’t cross the netns boundary easily without introducing new primitives

2. Making sure sockets are in the right state actually requires *rewriting* the recirc states in the code (or adding some kind of metadata for use by the classifier…)

# eBPF based approach

This one is controversioal



# eBPF Based Approach

It is 'future' thinking, though

- Upside - Builds on psock + eBPF
- Upside - Programming might be easier - and maybe we can preserve "megaflow" like behavior
- Upside - eBPF is constantly getting cool stuff (tm)
- Downside - bpf action is a big can of worms to add - doesn't play as well with OVS userspace
- Downside - May be harder to debug



# Where do we go?

Where is my mind?

# Where do we go?

Where is my mind?

- Preliminary patches for some of these approaches are posted
- Not too much opinion (Adrian and Cong gave two opinions)
- Difficult to know what the best approach moving forward is

# One more crazy thing to consider!

What else can I throw in here?

Should the flow writers be aware of socket concept as a way of programming application destination endpoints?

# Acknowledgements

Where is my mind?


- Minxi Hou - Helping with benchmarking and testing
- Adrian Moreno - Giving feedback
- Mike Pattrick - Internal code review
- Eric Dumazet, Stephen Hemminger, Cong Wang, Eelco Chaudron - Feedback on the kernel side RFC




# Questions?

Where is my mind?