Open vSwitch

Prague, November 19-20, 2025

OpenFlow Classifier: Arcane knowledge and common pitfalls

Ilya Maximets, Red Hat

OpenFlow Classifier: Rules

```
priority=300,arp                       actions=NORMAL
priority=200,ip,nw_dst=192.168.0.1     actions=output:port1
priority=200,ip,nw_dst=192.168.0.2     actions=output:port2
priority=100,ip,nw_dst=192.168.0.0/24  actions=output:port3
```

OpenFlow Classifier: Goals

1. Given a set of packet headers, as quickly as possible find the highest-priority rule that matches those headers.

2. “Un-wildcarding”

“Un-wildcarding”

Goal:

- Produce a wildcard mask that indicates which bits of the packet headers were essential to the classification result.

OpenFlow Classifier (trace)

```
$ ovs-appctl ofproto/trace br-int
'in_port=port1,tcp,nw_src=192.168.0.1,nw_dst=192.168.0.7'

Flow: tcp,in_port=port1,<...>,nw_src=192.168.0.1,nw_dst=192.168.0.7,<...>

bridge("br-int")
----------------
 0. ip,nw_dst=192.168.0.0/24, priority 100
    output:port3

Final flow: unchanged
Megaflow: recirc_id=0,eth,ip,in_port=port1,nw_dst=192.168.0.4/30,nw_frag=no
Datapath actions: port3
```

OpenFlow Classifier (trace)

```
$ ovs-appctl ofproto/trace br-int
'in_port=port1,tcp,nw_src=192.168.0.1,nw_dst=192.168.0.7'

Flow: tcp,in_port=port1,<...>,nw_src=192.168.0.1,nw_dst=192.168.0.7,<...>

bridge("br-int")
----------------
 0. ip,nw_dst=192.168.0.0/24, priority 100
    output:port3

Final flow: unchanged
Megaflow: recirc_id=0,eth,ip,in_port=port1,nw_dst=192.168.0.4/30,nw_frag=no
Datapath actions: port3
```

OpenFlow Classifier (trace)

```
$ ovs-appctl ofproto/trace br-int
'in_port=port1,tcp,nw_src=192.168.0.1,nw_dst=192.168.0.7'

Flow: tcp,in_port=port1,<...>,nw_src=192.168.0.1,nw_dst=192.168.0.7,<...>

bridge("br-int")
----------------
 0. ip,nw_dst=192.168.0.0/24, priority 100
    output:port3

Final flow: unchanged
Megaflow: recirc_id=0,eth,ip,in_port=port1,nw_dst=192.168.0.4/30,nw_frag=no
Datapath actions: port3
```

“Un-wildcarding”

Goal:

- Produce a wildcard mask that indicates which bits of the packet headers were essential to the classification result.

Specifics:

- 1-bit in any position of this mask means that, if the corresponding bit in the packet header were flipped, then the classification result might change.

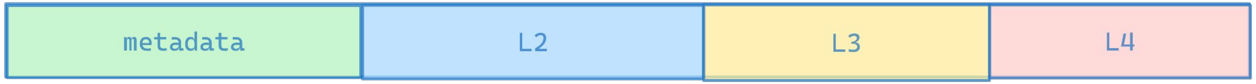
- 0-bit means that changing the packet header bit would have no effect.

“Un-wildcarding” - turning a wildcarded 0-bit into an exact-match 1-bit.

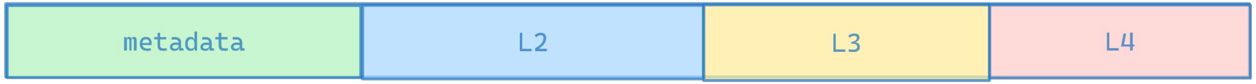
Properties of the wildcard mask

- “False 1-bits”: acceptable, but not desirable.
- “False 0-bits”: not acceptable.
- 0-bits are desirable in most cases - more autonomous datapath.
- Wildcard masks for lookups in a given classifier yield a non-overlapping set of rules.

Basic Classifier Design



Basic Classifier Design



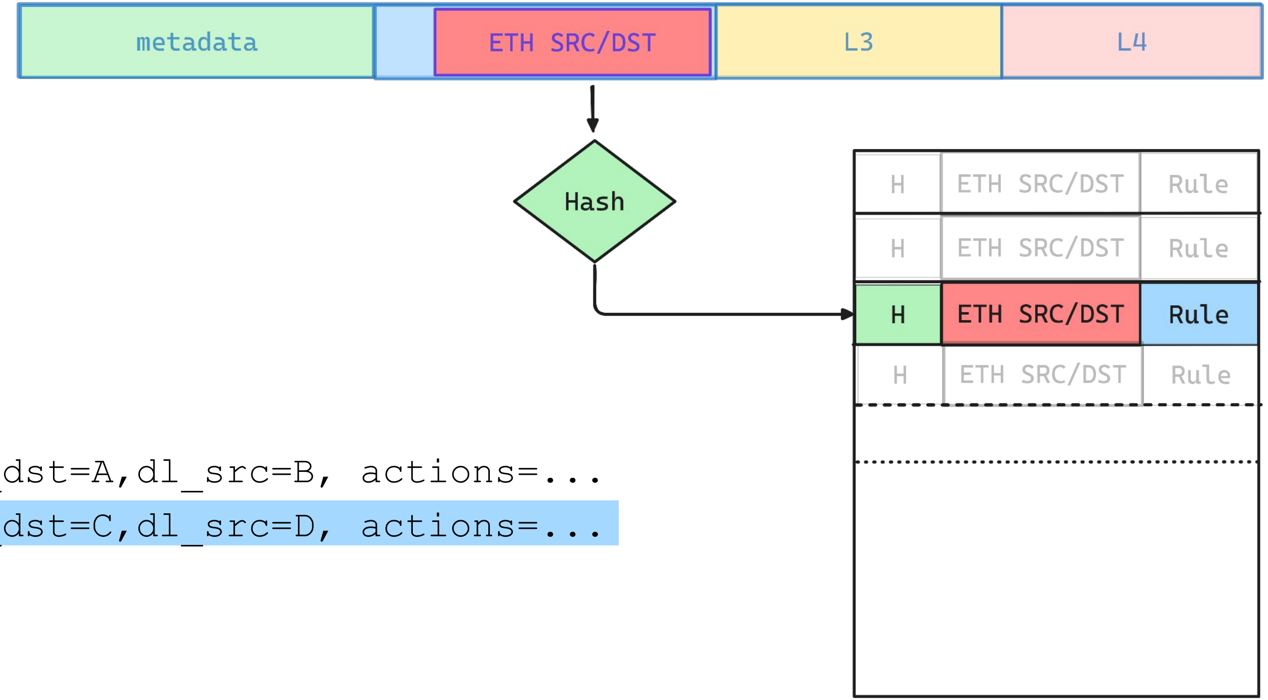
- Let's assume all OpenFlow rules are only matching on the L2 source and destination.

Basic Classifier Design

Let’s assume all OpenFlow rules are only matching on the L2 source and destination:

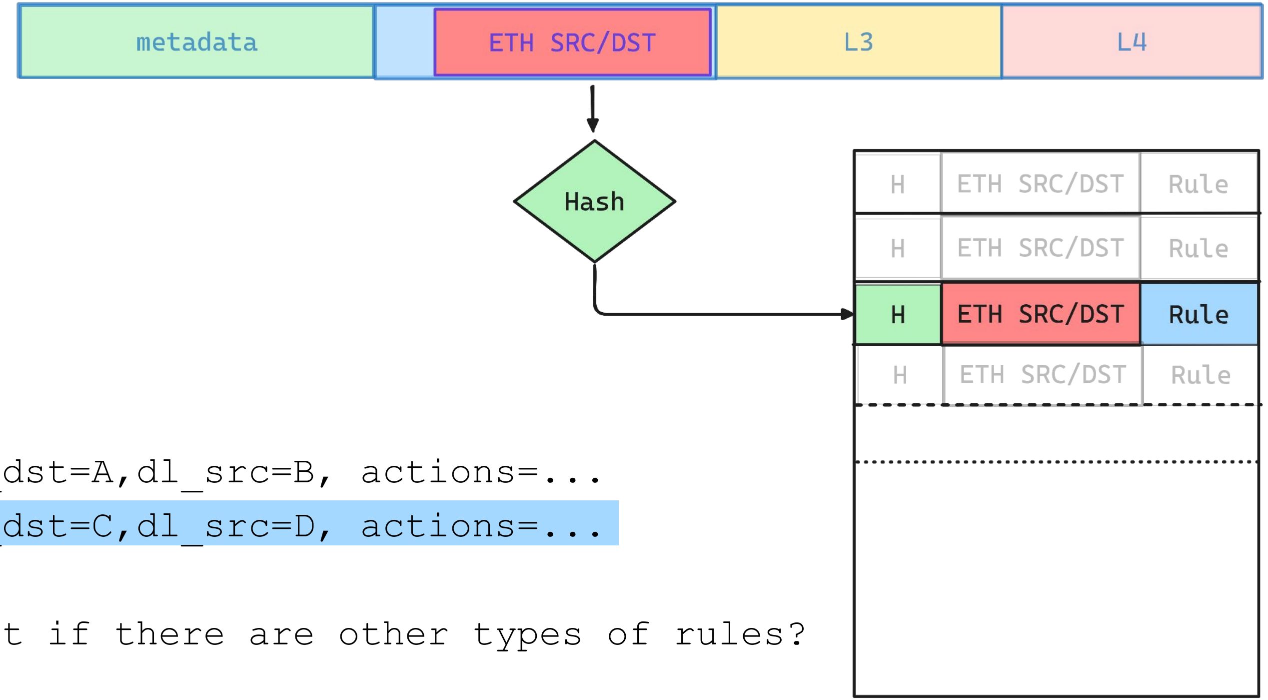
```
dl_dst=A,dl_src=B, actions=...
dl_dst=C,dl_src=D, actions=...
```

Basic Classifier Design: Hash Table




```
dl_dst=A,dl_src=B, actions=...
dl_dst=C,dl_src=D, actions=...
```

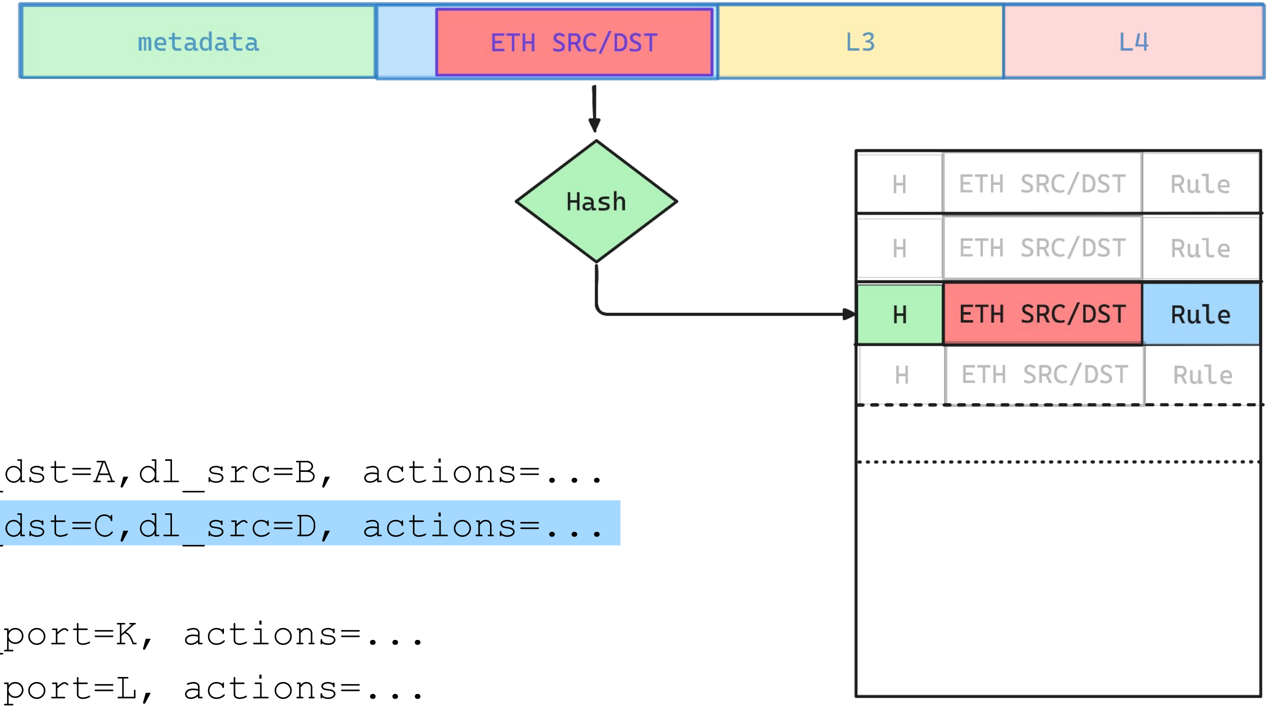
Basic Classifier Design: Hash Table




```
dl_dst=A,dl_src=B, actions=...
dl_dst=C,dl_src=D, actions=...
```

What if there are other types of rules?

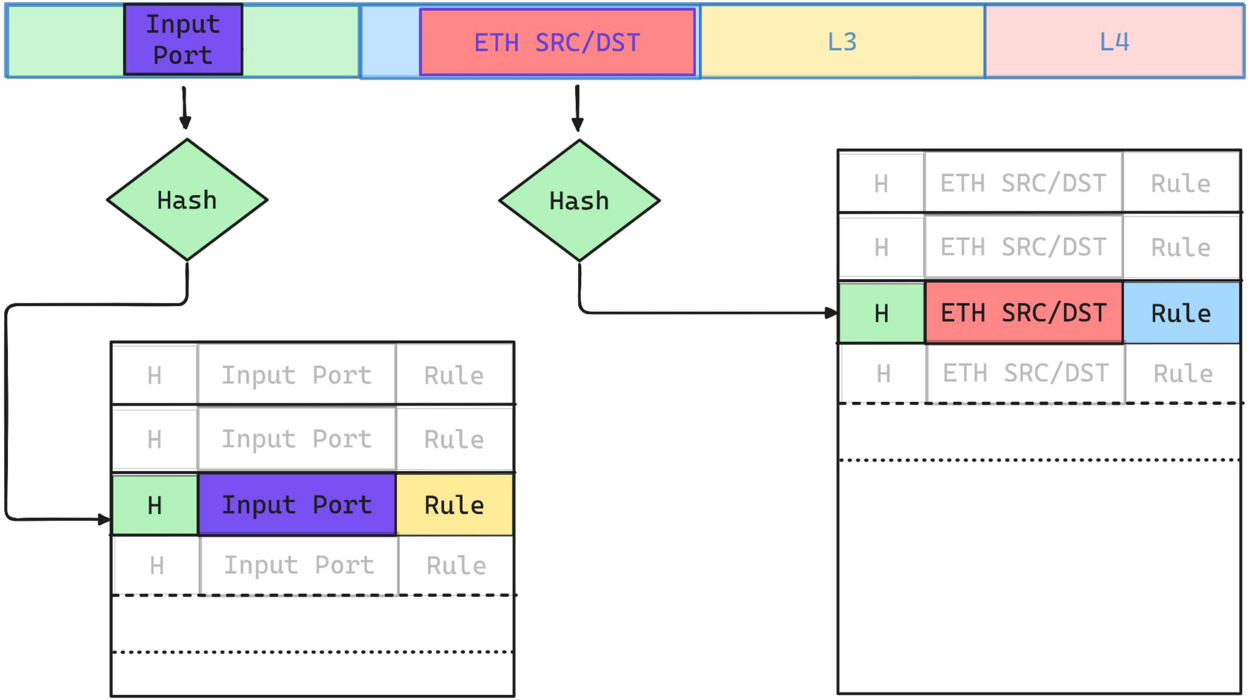
Basic Classifier Design: Hash Table




```
dl_dst=A,dl_src=B, actions=...
dl_dst=C,dl_src=D, actions=...

in_port=K, actions=...
in_port=L, actions=...
```

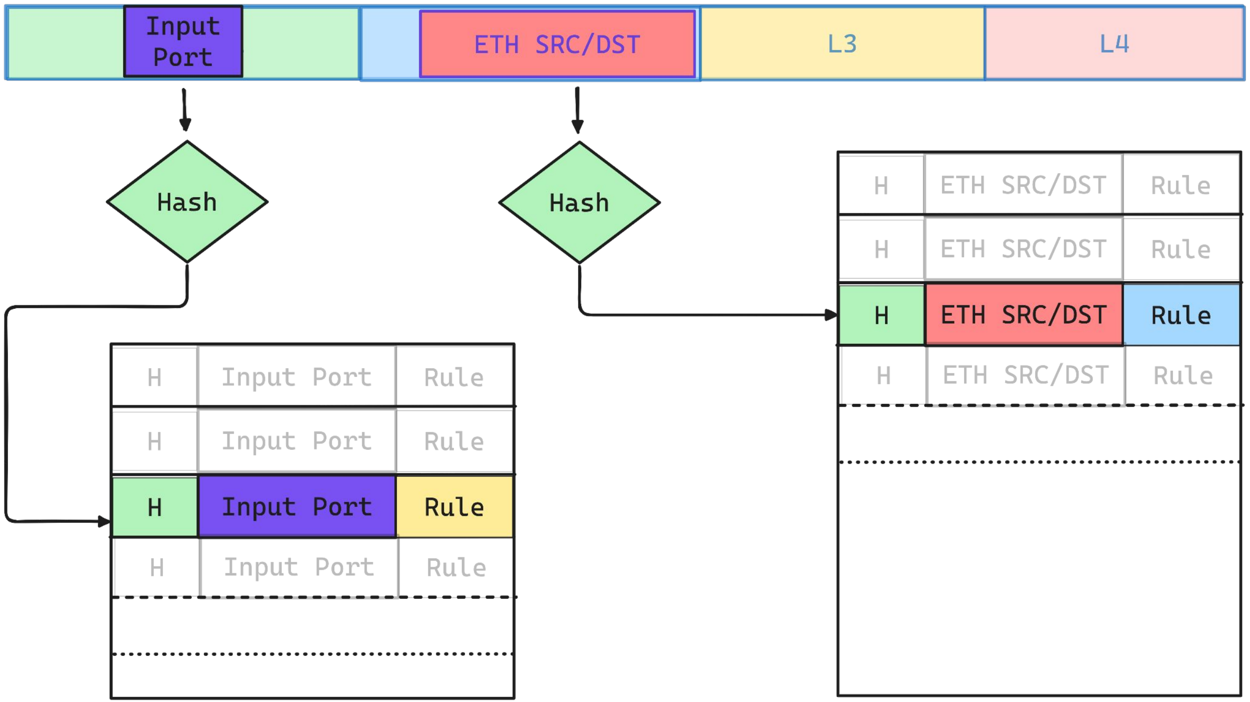
Basic Classifier Design: Hash Table(s)




```
dl_dst=A,dl_src=B, act…
dl_dst=C,dl_src=D, act…

in_port=K, actions=...
in_port=L, actions=...
```

Basic Classifier Design: Hash Table(s)

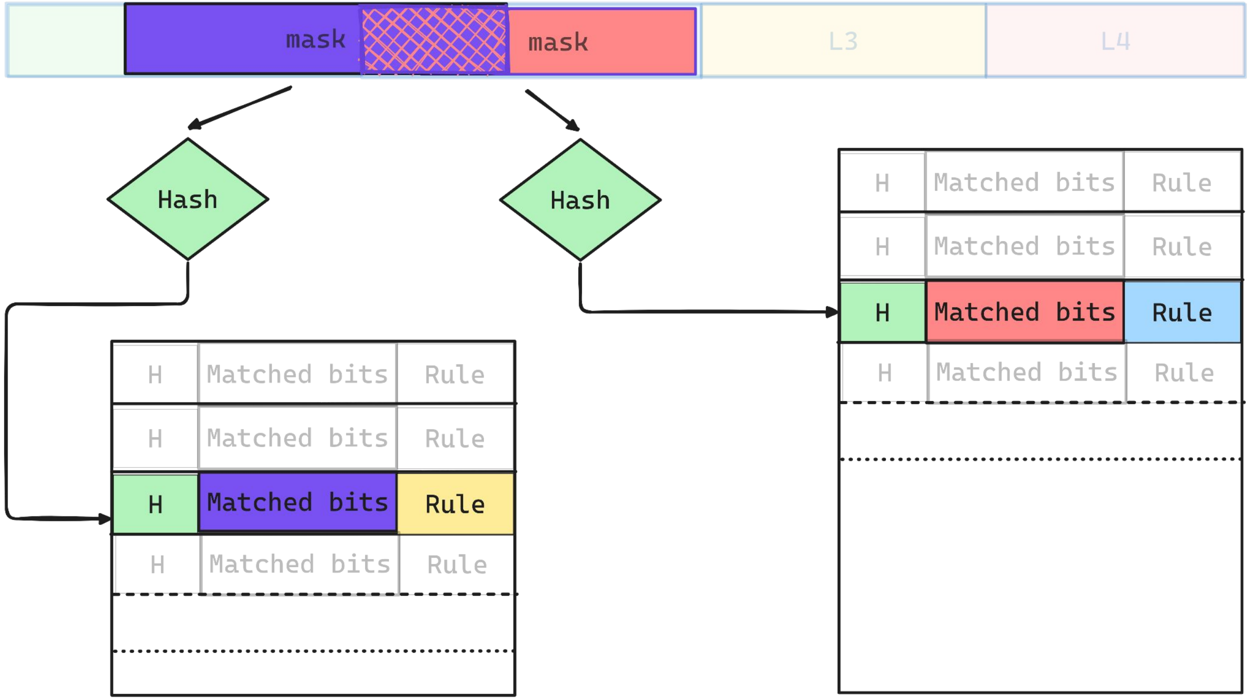


Out of two found rules, choose the one with the highest priority.

```
dl_dst=A,dl_src=B, act…
dl_dst=C,dl_src=D, act…

in_port=K, actions=...
in_port=L, actions=...
```

Basic Classifier Design: Hash Table(s)

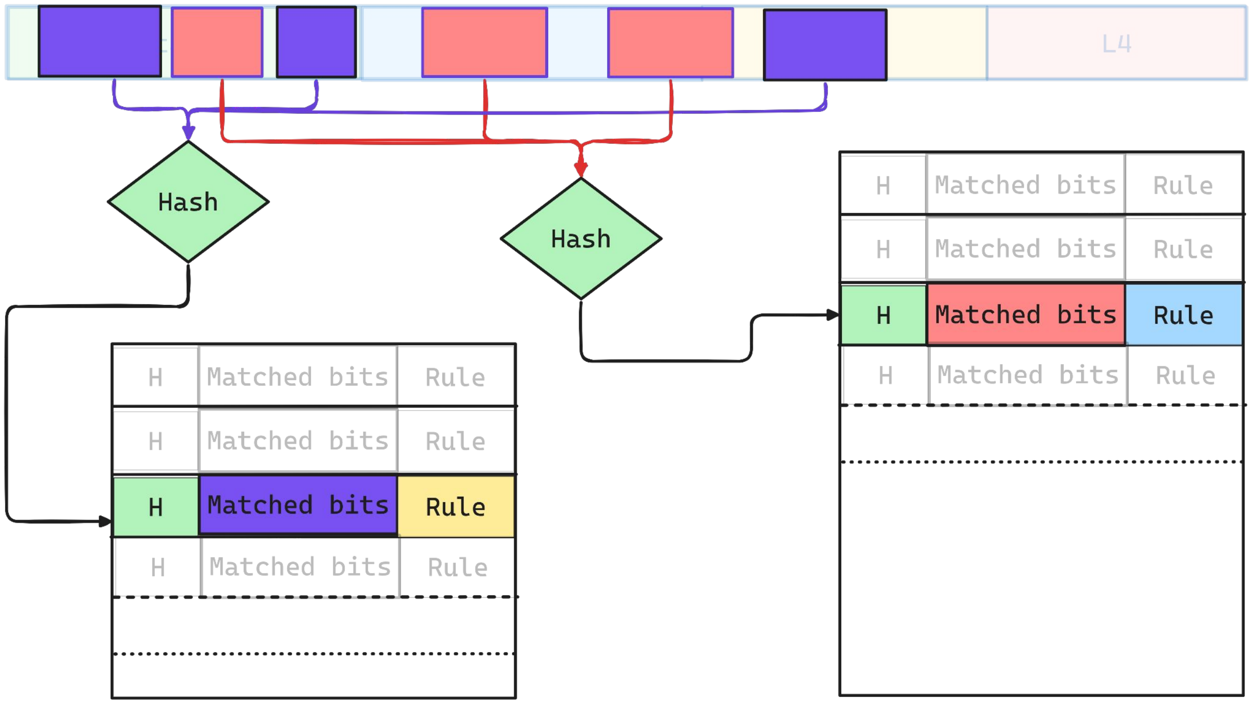


Overlapping ✅

```
Match pattern #1.a, act…
Match pattern #1.b, act…

Match pattern #2.a, act…
Match pattern #2.b, act…
```

Basic Classifier Design: Hash Table(s)

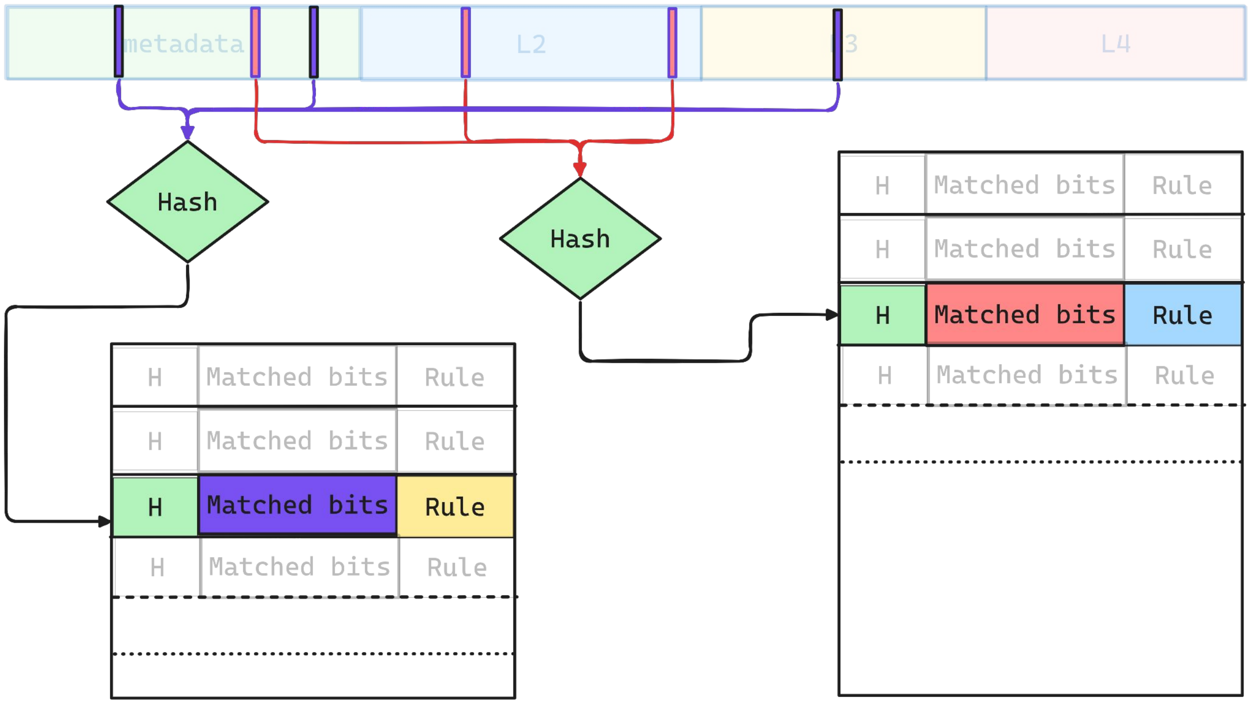


Overlapping ✅
Non-contiguous ✅

```
Match pattern #1.a, act…
Match pattern #1.b, act…

Match pattern #2.a, act…
Match pattern #2.b, act…
```

Basic Classifier Design: Hash Table(s)

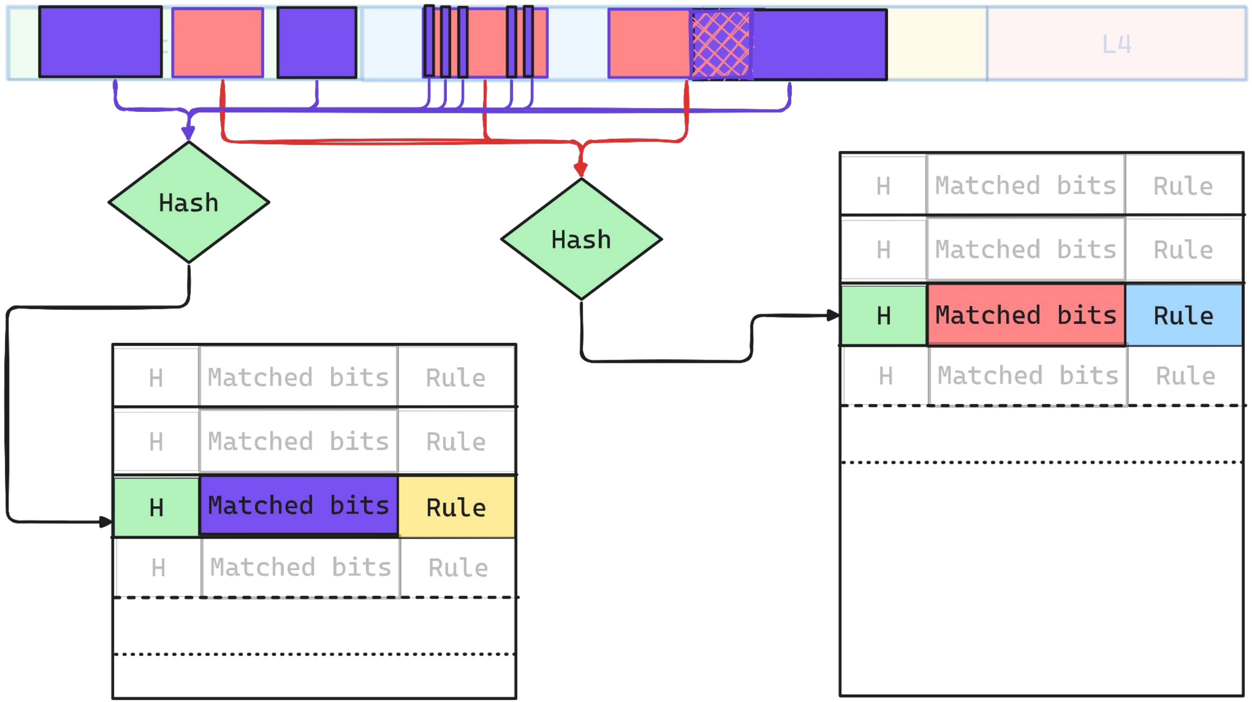


Overlapping ✅
Non-contiguous ✅
Arbitrary bits ✅

```
Match pattern #1.a, act…
Match pattern #1.b, act…

Match pattern #2.a, act…
Match pattern #2.b, act…
```

Basic Classifier Design: Hash Table(s)



Overlapping ✅
Non-contiguous ✅
Arbitrary bits ✅
All of the above ✅

```
Match pattern #1.a, act…
Match pattern #1.b, act…

Match pattern #2.a, act…
Match pattern #2.b, act…
```

Classifier workflow

- On OpenFlow rule insertion:
    - Find all unique rule patterns (“masks”).

Classifier workflow

- On OpenFlow rule insertion:
  - Find all unique rule patterns (“masks”).
  - Create a hash table (“subtable”) for each of these masks.

Classifier workflow

- On OpenFlow rule insertion:
  - Find all unique rule patterns (“masks”).
  - Create a hash table (“subtable”) for each of these masks.
  - Put all the rules to their specific subtables
    (one rule cannot be in more than one subtable).

Classifier workflow

- On lookup:
    - Go over all the subtables.

Classifier workflow

- On lookup:
  - Go over all the subtables.
  - Calculate a packet hash from bits included in the subtable mask.

Classifier workflow

- On lookup:
  - Go over all the subtables.
  - Calculate a packet hash from bits included in the subtable mask.
  - Lookup the rule in the hash table.

Classifier workflow

- On lookup:
  - Go over all the subtables.
  - Calculate a packet hash from bits included in the subtable mask.
  - Lookup the rule in the hash table.
  - Out of all the rules found in different subtables, choose one with the highest priority.

Classifier workflow

- On lookup:
  - Go over all the subtables.
  - Calculate a packet hash from bits included in the subtable mask.
  - Lookup the rule in the hash table.
  - Out of all the rules found in different subtables, choose one with the highest priority.
- Optimization:
  - Subtables are stored in order of highest priority of included rules.
  - Once we got a match in one subtable, all the subtables that do not contain higher priority rules, can be skipped.

Classifier workflow

- Lookup complexity:
  - O(1) for a hash map lookup.
  - **O(n)** for going over all the subtables in the worst case, where ‘n’ is the number of different match patterns in the OpenFlow table.

Classifier workflow: Performance

- Lookup complexity:
  - O(1) for a hash map lookup.
  - **O(n)** for going over all the subtables in the worst case, where ‘n’ is the number of different match patterns in the OpenFlow table.
- Potential performance concern: **too many different match patterns within the same table.**

Classifier workflow: Performance

- Potential performance concern: **too many different match patterns within the same table.**
- Potential sources:

Classifier workflow: Performance

- Potential performance concern: **too many different match patterns within the same table.**
- Potential sources:
  - Users. :)

Classifier workflow: Performance

- Potential performance concern: **too many different match patterns within the same table.**
- Potential sources:
  - Users. :)
  - Older OVN converting inequality matches into a lot of different single or a few-bit matches.

Classifier workflow: Performance

- Potential performance concern: **too many different match patterns within the same table.**
- Potential sources:
  - Users. :)
  - Older OVN converting inequality matches into a lot of different single or a few-bit matches.

    Not anymore!

    `[PATCH ovn] expr: Use prefixes instead of single bit masks for inequality.`
    https://mail.openvswitch.org/pipermail/ovs-dev/2025-October/427300.html

Classifier workflow: Performance

- Potential performance concern: **too many different match patterns within the same table.**

- Potential sources:

  – Users. :)

  – Older OVN converting inequality matches into a lot of different single or a few-bit matches.

    Not anymore!

    `[PATCH ovn] expr: Use prefixes instead of single bit masks for inequality.`
    https://mail.openvswitch.org/pipermail/ovs-dev/2025-October/427300.html

  – But a pair of IPv6 addresses is still a possibility for 128*128 unique masks.
    So, watch out!
    Though OVN shouldn't be generating that many different prefix masks normally.

Classifier workflow: Missing Parts

- There is an extra functionality in the classifier to support conjunctive matches, but we’ll not cover it here.

Wildcards: Optimizations

- Classifier MUST un-wildcard the mask of the subtable that was examined.
(It made decisions based on those bits.)

Wildcards: Optimizations

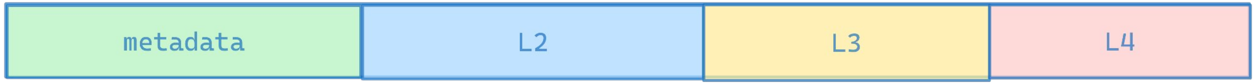
- Classifier MUST un-wildcard the mask of the subtable that was examined.
  (It made decisions based on those bits.)

- Issue: A lot of bits getting un-wildcarded that are not strictly necessary for the decision.

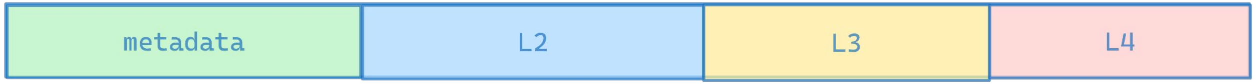
Wildcards: Optimizations

- Classifier MUST un-wildcard the mask of the subtable that was examined.
  (It made decisions based on those bits.)
- Issue: A lot of bits getting un-wildcarded that are not strictly necessary for the decision.
- Optimizations:
  - Staged Lookup.
  - Prefix Tracking.

Staged Lookup



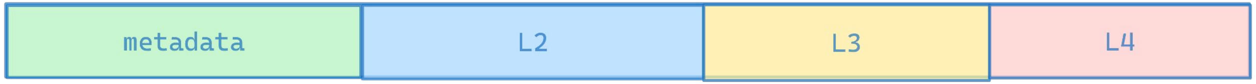
Staged Lookup



- Lines are a little blurry inside the `struct flow`.
- But packet fields in one stage cannot depend on fields from the next.
  We had some problems with this in the past:

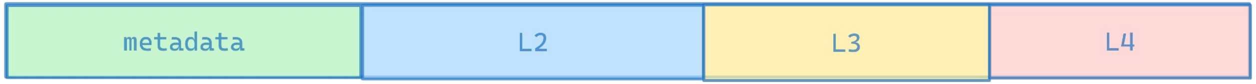
  `ca44218515f0 ("classifier: Adjust segment boundary to execute prerequisite processing.")`

Staged Lookup: Workflow



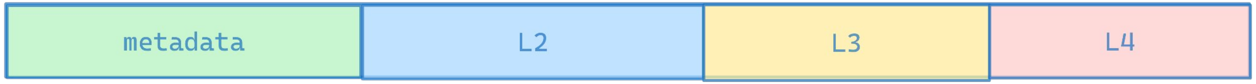
- The mask of each subtable is split into 4 segments.

Staged Lookup: Workflow



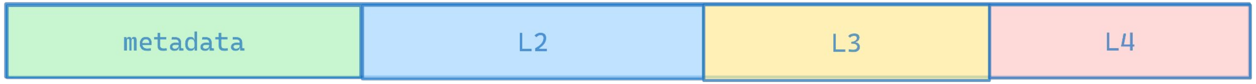
- The mask of each subtable is split into 4 segments.
- Each segment is hashed separately and we practically have 4 hash maps now.

Staged Lookup: Workflow



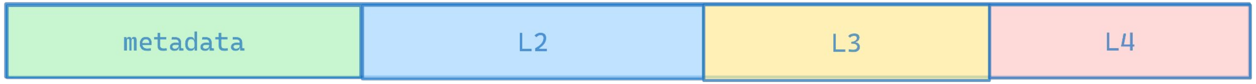
- The mask of each subtable is split into 4 segments.
- Each segment is hashed separately and we practically have 4 hash maps now.
- If the lookup for one segment doesn’t find any rules, it means that lookup failed and there is no point in checking further segments or the whole mask.

Staged Lookup: Workflow



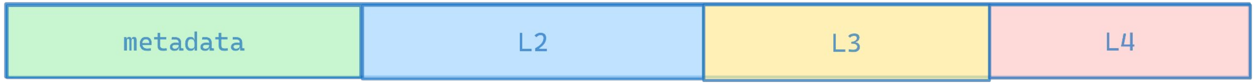
- The mask of each subtable is split into 4 segments.
- Each segment is hashed separately and we practically have 4 hash maps now.
- If the lookup for one segment doesn’t find any rules, it means that lookup failed and there is no point in checking further segments or the whole mask.
- Only the mask segments that were looked at need to be un-wildcarded.

Staged Lookup: Pros and Cons



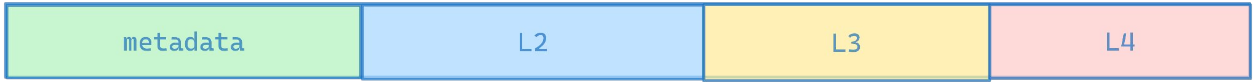
- Likely a bit slower due to 4 hash map lookups instead of 1 in case there is a match.

Staged Lookup: Pros and Cons



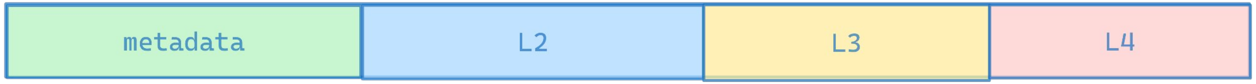
- Likely a bit slower due to 4 hash map lookups instead of 1 in case there is a match.
- Likely faster detection if the packet doesn’t match the subtable in the common case.

Staged Lookup: Pros and Cons



- Likely a bit slower due to 4 hash map lookups instead of 1 in case there is a match.
- Likely faster detection if the packet doesn’t match the subtable in the common case.
- Significantly smaller amount of un-wildcarded bits for packets that do not match.

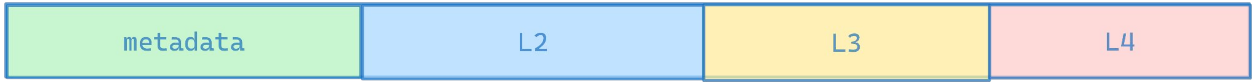
Staged Lookup: Pros and Cons



- Likely a bit slower due to 4 hash map lookups instead of 1 in case there is a match.
- Likely faster detection if the packet doesn't match the subtable in the common case.
- Significantly smaller amount of un-wildcarded bits for packets that do not match.

Warning: Be extra careful modifying `struct flow` in regards to segment boundaries!
Dependencies must never escape into a later segment!
(also, segments must be 64-bit aligned - implementation detail)

Staged Lookup: Take Advantage!



- If there is a possibility to match on fields from earlier stages along with the fields from the later ones without sacrificing logic of your OpenFlow pipeline - do so!

```
03ef56f9cf6f ("northd: Add missing multicast match to DHCPv6 options flows.")
2e7f318c9b54 ("Reply only for the multicast ND solicitations.")
43c34f2e6676 ("logical-fields: Add missing multicast matches for MLD and IGMP.")

--------
    /* MLDv2 packets are sent to ff02::16 (RFC 3810, 5.2.14) */
    expr_symtab_add_predicate(symtab, "mldv2",
-                                 "ip6.dst == ff02::16 && icmp6.type == 143");
+                                 "eth.mcastv6 && ip6.dst == ff02::16 && "
+                                 "icmp6.type == 143");
```

Before this change - exact match on ipv6_dst for majority of IPv6 traffic!

Prefix Tracking

- People prefer to use prefix matches for certain types of fields like:

  - nw_dst=192.168.0.0/24
  - ipv6_src=fe80::/64

Prefix Tracking

- People prefer to use prefix matches for certain types of fields like:

  - `nw_dst=192.168.0.0/24`
  - `ipv6_src=fe80::/64`

- We can use this to our advantage to avoid un-wildcarding unnecessary bits!

Prefix Tracking

```
priority=300,arp                              actions=NORMAL
priority=200,ip,nw_dst=192.168.0.1            actions=output:port1
priority=200,ip,nw_dst=192.168.0.2            actions=output:port2
priority=100,ip,nw_dst=192.168.0.0/24         actions=output:port3
```

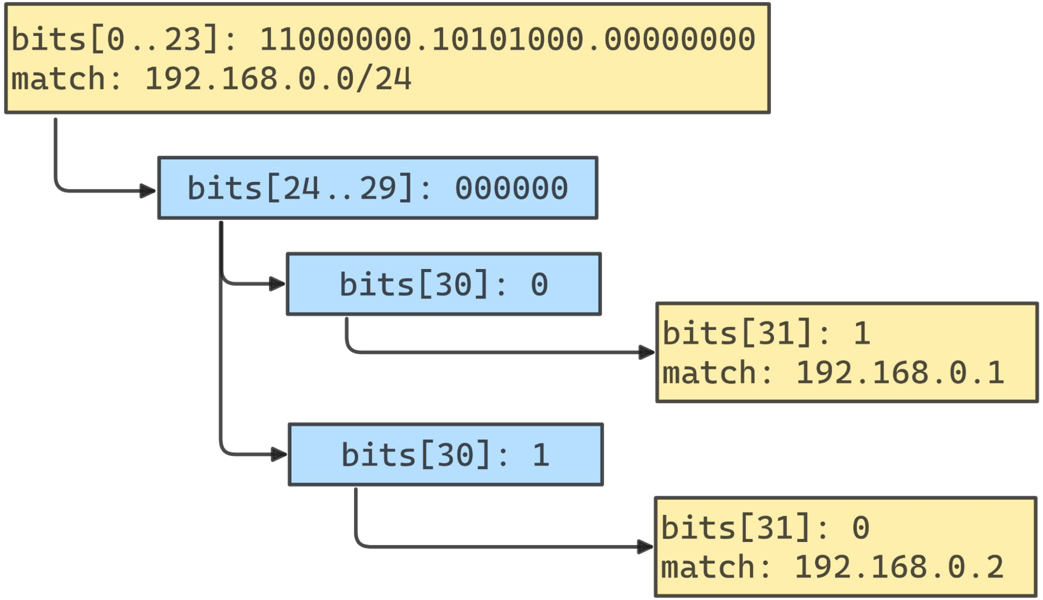
Prefix Tracking

- Let’s build a Prefix Tree (trie) for the nw_dst field using values for this field from all the OpenFlow rules we have.

```
Binary representation:
 - 192.168.0.0/24  = 11000000.10101000.00000000.00000000 /24
 - 192.168.0.1/32  = 11000000.10101000.00000000.00000001 /32
 - 192.168.0.2/32  = 11000000.10101000.00000000.00000010 /32
```

Prefix Tracking

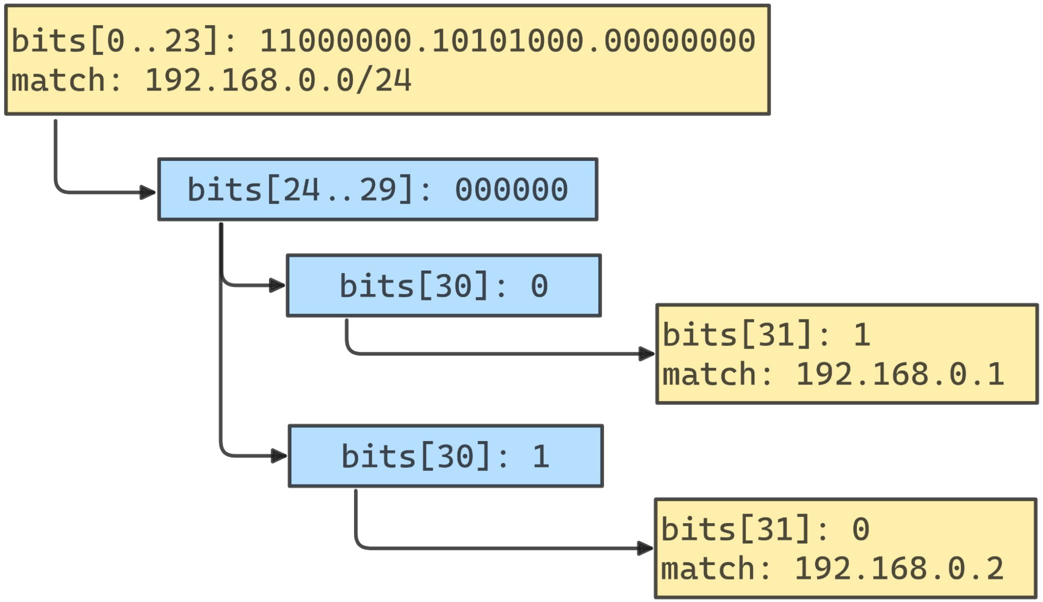
```
Binary representation:
 - 192.168.0.0/24  = 11000000.10101000.00000000.00000000 /24
 - 192.168.0.1/32  = 11000000.10101000.00000000.00000001 /32
 - 192.168.0.2/32  = 11000000.10101000.00000000.00000010 /32
```



Prefix Tracking

```
Binary representation:
 - 192.168.0.0/24  = 11000000.10101000.00000000.00000000 /24
 - 192.168.0.1/32  = 11000000.10101000.00000000.00000001 /32
 - 192.168.0.2/32  = 11000000.10101000.00000000.00000010 /32
```

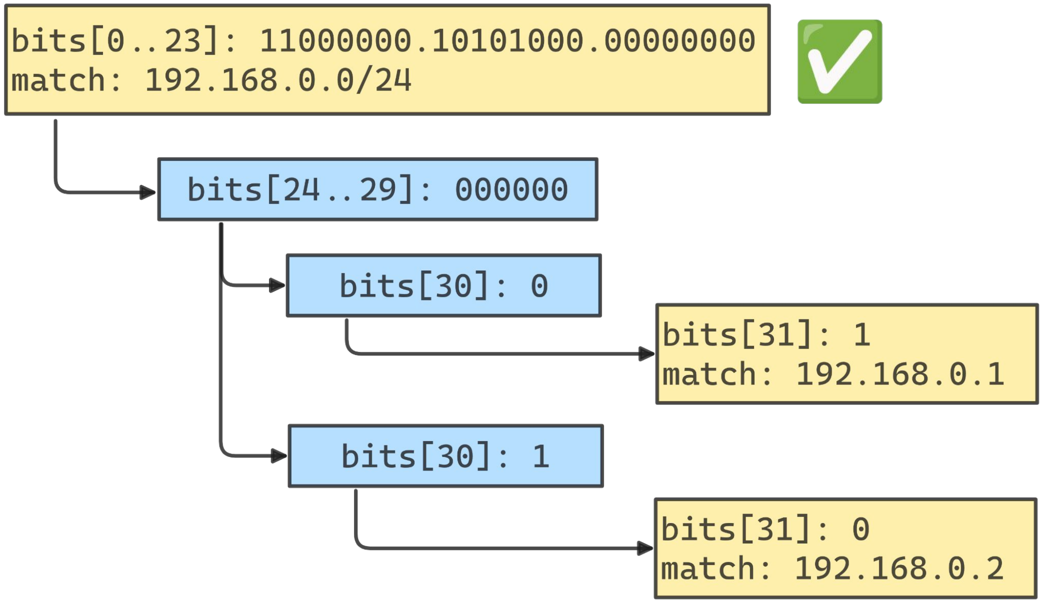
Let’s lookup 192.168.0.7



Prefix Tracking

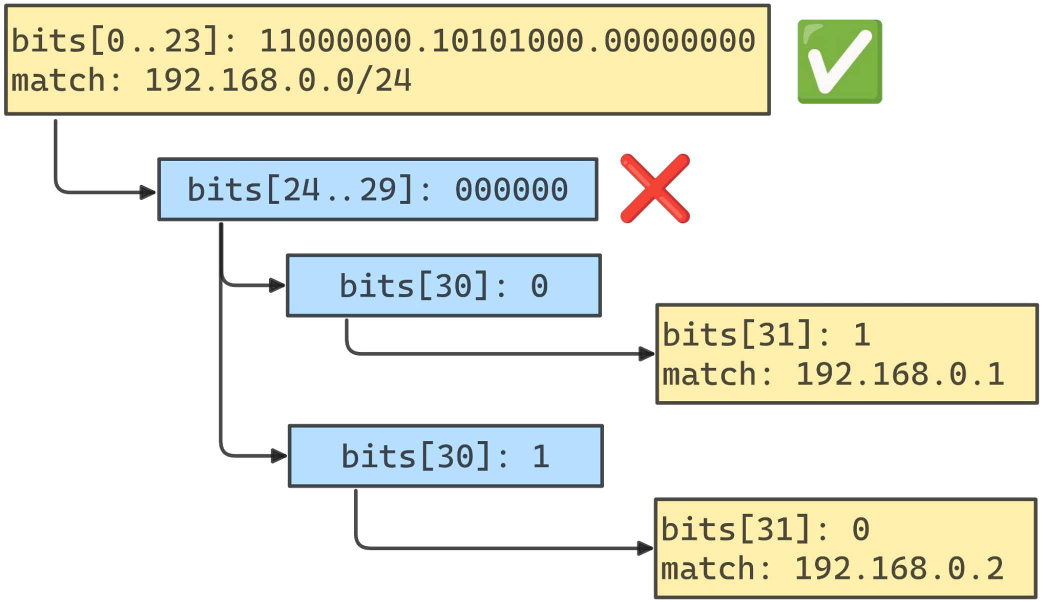
```
Binary representation:
 - 192.168.0.0/24  = 11000000.10101000.00000000.00000000 /24
 - 192.168.0.1/32  = 11000000.10101000.00000000.00000001 /32
 - 192.168.0.2/32  = 11000000.10101000.00000000.00000010 /32
```

Let’s lookup 192.168.0.7



Prefix Tracking

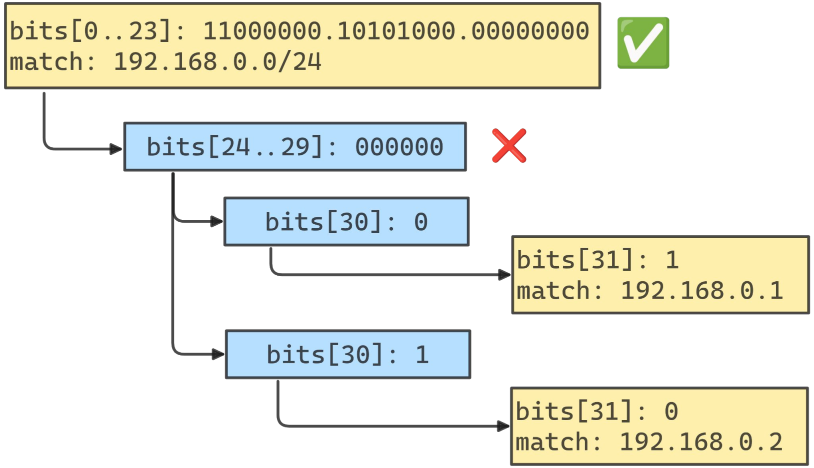
```
Binary representation:
 - 192.168.0.0/24  = 11000000.10101000.00000000.00000000 /24
 - 192.168.0.1/32  = 11000000.10101000.00000000.00000001 /32
 - 192.168.0.2/32  = 11000000.10101000.00000000.00000010 /32
```



Let's lookup 192.168.0.7

Prefix Tracking

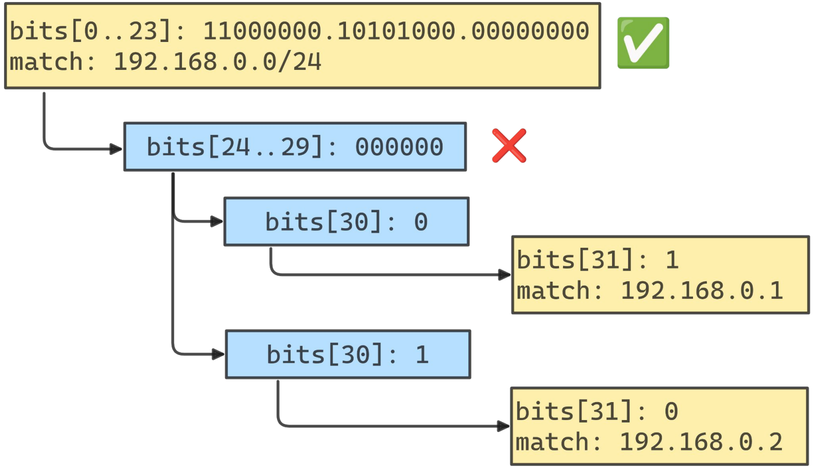
- So, we had 3 OpenFlow rules.
- One of them had /24 mask.
- Two of them had /32 masks on the address.



Prefix Tracking

- Since masks are different, we'll have two subtables:
  - One for the rule with /24 match.
  - One for two rules with /32 matches.

- Our packet has address 192.168.0.7.

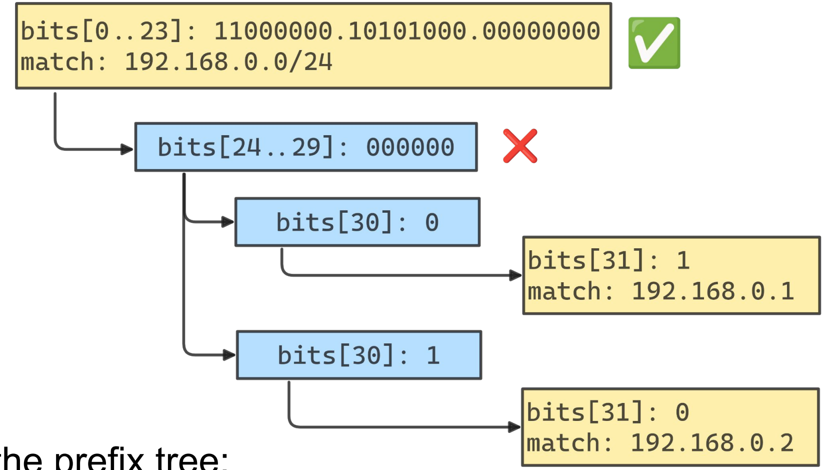


Prefix Tracking

- Since masks are different, we'll have two subtables:
  - One for the rule with /24 match.
  - One for two rules with /32 matches.

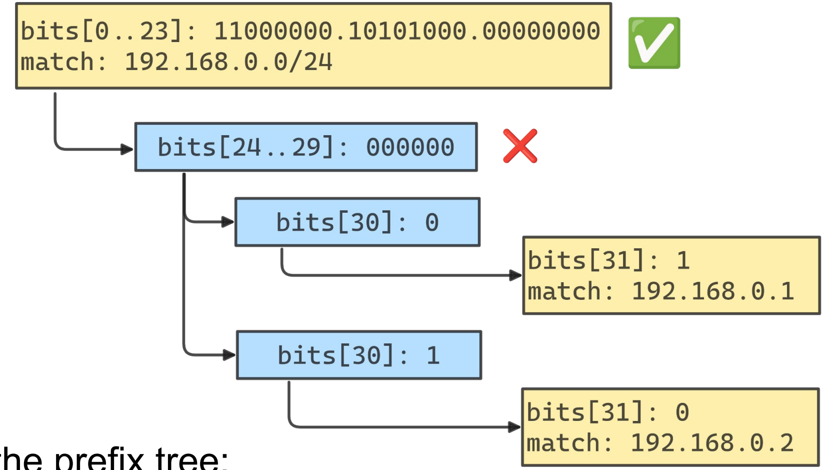
- Our packet has address 192.168.0.7.

- Before looking it up in the first subtable - look it up in the prefix tree:
  - It's necessary to check first 30 bits (0-29) before the lookup fails.
  - But our subtable has only 24 bits in the mask and there was a match.
  - No luck, need to do a full staged lookup.



Prefix Tracking

- Since masks are different, we'll have two subtables:
  - One for the rule with /24 match.
  - One for two rules with /32 matches.



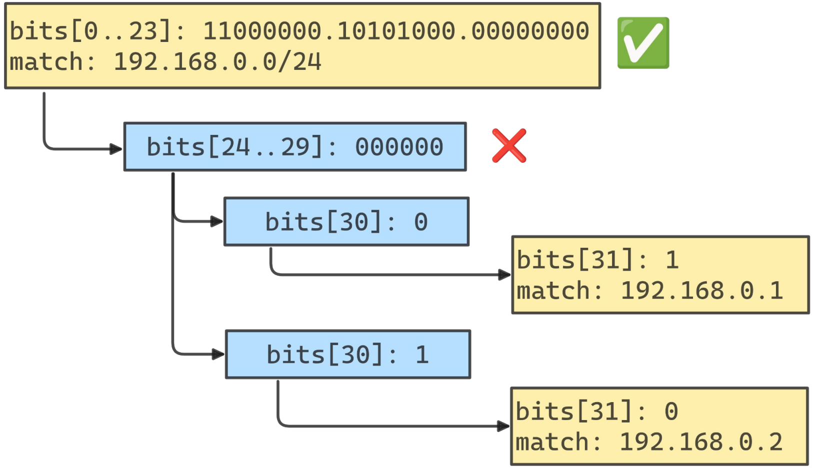
- Our packet has address 192.168.0.7.

- Before looking it up in the first subtable - look it up in the prefix tree:
  - It's necessary to check first 30 bits (0-29) before the lookup fails.
  - But our subtable has only 24 bits in the mask and there was a match.
  - No luck, need to do a full staged lookup.

- Result: un-wildcarded first 24 bits.

Prefix Tracking

- Since masks are different, we'll have two subtables:
  - One for the rule with /24 match.
  - One for two rules with /32 matches.
- Our packet has address 192.168.0.7.
- Let's see what is happening in the second subtable:



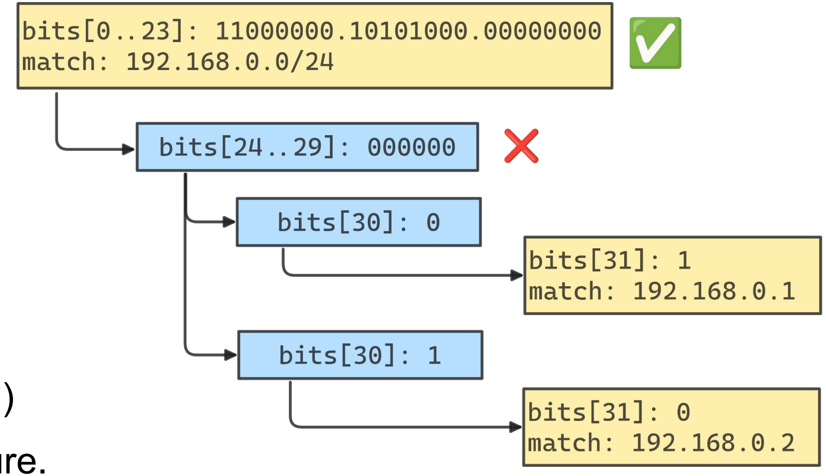
Prefix Tracking

- Since masks are different, we'll have two subtables:
  - One for the rule with /24 match.
  - One for two rules with /32 matches.

- Let's see what is happening in the second subtable:
  - Prefix tree lookup is the same (the tree is global)
  - It still requires to check the bits 0-29 before failure.
  - But our subtable mask has 32 bits in it!

    So, we didn't get to any of the rules that are relevant to this subtable!

- Result: It's enough to un-wildcard only the first 30 bits to know that this packet doesn't match anything in this subtable!



Prefix Tracking (trace)

```
$ ovs-appctl ofproto/trace br-int
'in_port=port1,tcp,nw_src=192.168.0.1,nw_dst=192.168.0.7'

Flow: tcp,in_port=port1,<...>,nw_src=192.168.0.1,nw_dst=192.168.0.7,<...>

bridge("br-int")
----------------
 0. ip,nw_dst=192.168.0.0/24, priority 100
    output:port3

Final flow: unchanged
Megaflow: recirc_id=0,eth,ip,in_port=port1, nw_dst=192.168.0.4/30,nw_frag=no
Datapath actions: port3
```

Prefix Tracking: Pros and Cons

- Need to pre-build prefix trees and maintain them during OpenFlow rule updates.

Prefix Tracking: Pros and Cons

- Need to pre-build prefix trees and maintain them during OpenFlow rule updates.
- Need to perform prefix tree lookups in addition to subtable lookups (at most 5):
  - Only looked up when a field is included in the current stage.
  - Since the tree is per-classifier and not per-subtable, lookups can be cached.

Prefix Tracking: Pros and Cons

- Need to pre-build prefix trees and maintain them during OpenFlow rule updates.
- Need to perform prefix tree lookups in addition to subtable lookups (at most 5):
  - Only looked up when a field is included in the current stage.
  - Since the tree is per-classifier and not per-subtable, lookups can be cached.

- Allows to eliminate un-wildcarding of entire stage, replacing it with a shorter prefix mask on one field, when trie lookup detects no-match for the current subtable.

Prefix Tracking: Limitations

- Current implementation relies on fields to be multiple of 32-bit:
  - Trie for transport ports includes both ports, so it’s a prefix of two ports together.

Prefix Tracking: Limitations

- Current implementation relies on fields to be multiple of 32-bit:
  - Trie for transport ports includes both ports, so it’s a prefix of two ports together.

- Needs to be enabled!
  (Enabled by default for ipv4 and ipv6 addresses and transport ports since OVS 3.5)

  ```
  89e43f7528b0 ("controller: Fix IPv6 dp flow explosion by setting flow table prefixes.")
  ```

Prefix Tracking: Limitations

- Current implementation relies on fields to be multiple of 32-bit:
  - Trie for transport ports includes both ports, so it’s a prefix of two ports together.

- Needs to be enabled!
  (Enabled by default for ipv4 and ipv6 addresses and transport ports since OVS 3.5)

  ```
  89e43f7528b0 ("controller: Fix IPv6 dp flow explosion by setting flow table prefixes.")
  ```

- Can only be enabled for specific fields: nw_src/dst, ipv6_src/dst, and corresponding tunnel metadata + tun_id (Transport ports trie is always enabled).

Prefix Tracking: Limitations (more)

- Doesn't work for subtables with non-contiguous masks for the field.

```
ab19375413cf ("logical-fields: Fix IPv6 dp flow explosion caused by ip6.mcast_rsvd.")

-------
    /* Predefined IPv6 multicast groups (RFC 4291, 2.7.1). */
    expr_symtab_add_predicate(symtab, "ip6.mcast_rsvd",
-                             "ip6.dst[116..127] == 0xff0 && "
-                             "ip6.dst[0..111] == 0x0");
+                             "ip6.dst == { "
+                                 "ff00::0, ff01::0, ff02::0, ff03::0, "
+                                 "ff04::0, ff05::0, ff06::0, ff07::0, "
+                                 "ff08::0, ff09::0, ff0a::0, ff0b::0, "
+                                 "ff0c::0, ff0d::0, ff0e::0, ff0f::0 "
+                              "}");
```

- Without this change we have practically an exact match on ipv6_dst for every IPv6 packet traversing OVN logical router!

Thank you!

Ilya Maximets, i.maximets@ovn.org