Source-specific routing implementation on Linux

Matthieu Boutier, joint work with Juliusz Chroboczek

IRIF (ex Laboratoire PPS) - Université Paris Diderot
boutier@pps.univ-paris-diderot.fr
jch@pps.univ-paris-diderot.fr

Virtual interim meeting
RTGWG

May 2016

Reminders: source-specific routing

Source-specific routing
(or SADR, or dst/src routing)

forwards packets based on their
destination and source addresses

a source-specific routing table

destination	source	next-hop
2001:db8:2::/48	::/0	…
::/0	2001:db8:1::/48	…
…	…	…

Reminders: expected behaviour

destination	source	next-hop
2001:db8:2::/48	::/0	…
::/0	2001:db8:1::/48	…
…	…	…

How to route (2001:db8:2::1, 2001:db8:1::1)?

There is an ambiguity when two entries match a single packet, without one being more specific than the other on both the destination and the source address of the packet.

In case of ambiguity, there is consensus to prefer entries:

- with the most specific destination prefix
- if equal, with the most specific source prefix

Context: general case

In most cases, it looks like source specific routes are:

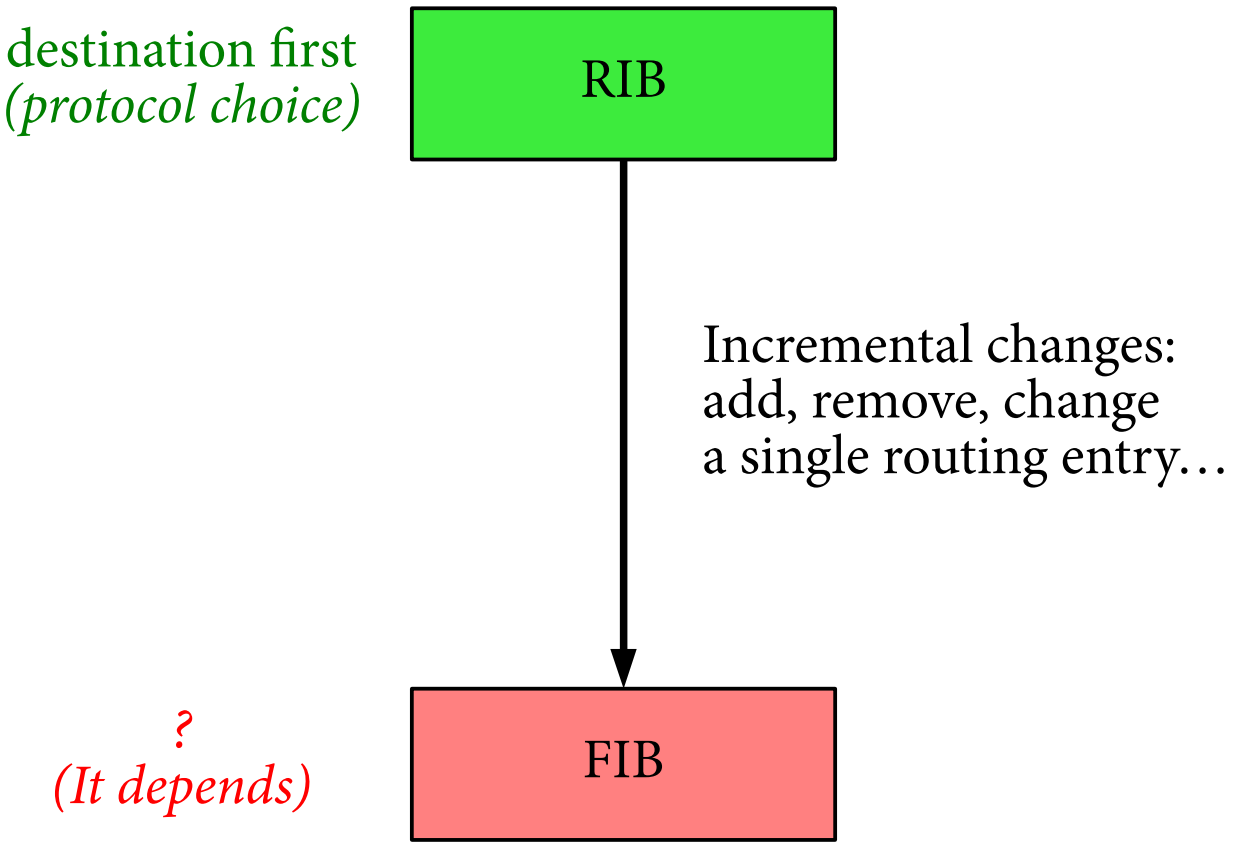
- with default (::/0) destinations,
- with disjoint or default sources.

We look at the general case, where
both the source and the destination may not be ::/0

destination	source	next-hop
2001:db8:3::/48	2001:db8:1::/48	…

- What about futures applications?
- Is it really worth it?

Implementation depends on the Forwarding plane



Linux APIs for source-specific routing

In Linux, there is two APIs (both through *Netlink*):

- IPv6 subtrees,
  → *native destination first source-specific routing tables*
  → *not available everywhere*

- Traffic engineering.
  → *multiple classical routing tables selected by traffic engineering rules (source first)*
  → *available everywhere*

Our implementation can use either.

Linux IPv6 subtrees

→ install(dest, src, next-hop)

	destination	source	next-hop
	2001:db8:2::/48	::/0	B
+	::/0	2001:db8:1::/48	C

GOOD BEHAVIOUR

(destination first)

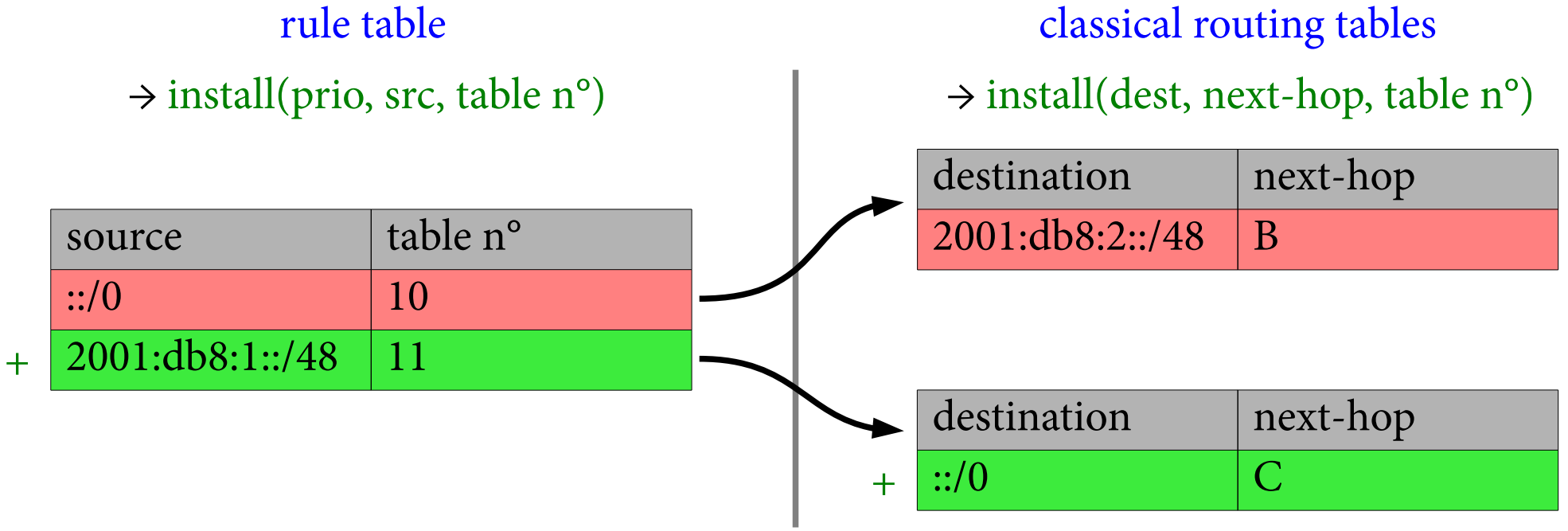
But not available everywhere:

→ it's *only available on recent Linux kernels,*

→ *Linux must be compiled with the right option,*

→ *it works only for IPv6.*

Linux traffic engineering



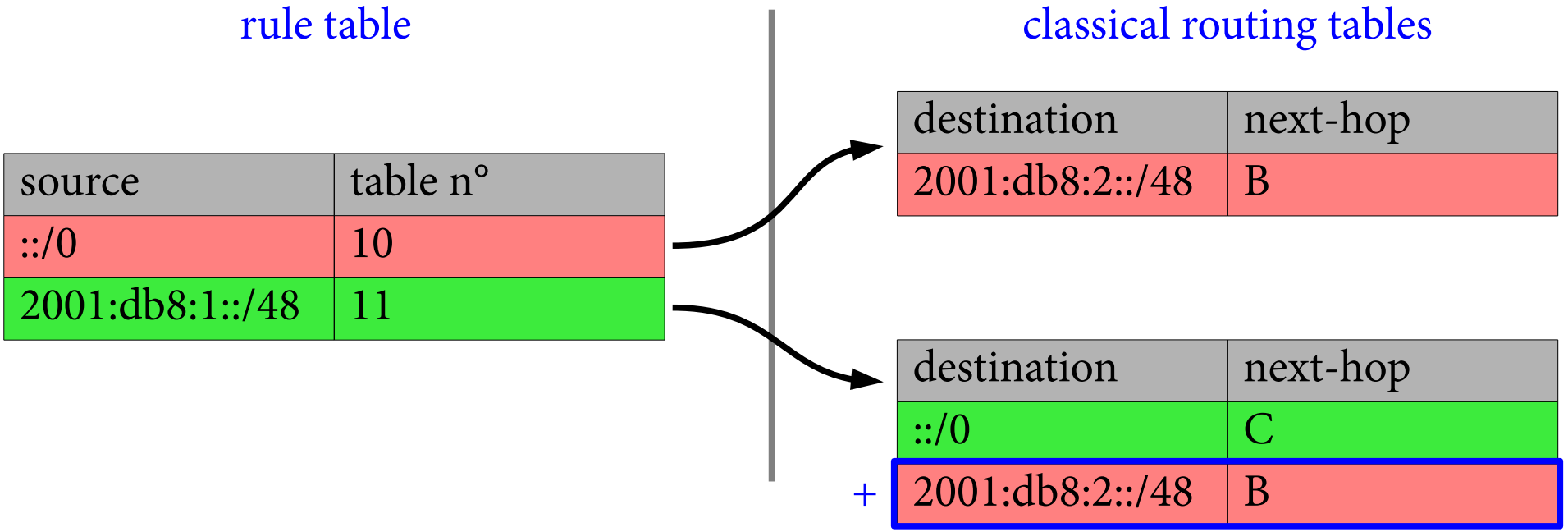
WRONG BEHAVIOUR

(source first)

But it works on every Linux distribution we met.

Similar interfaces exist on other systems.

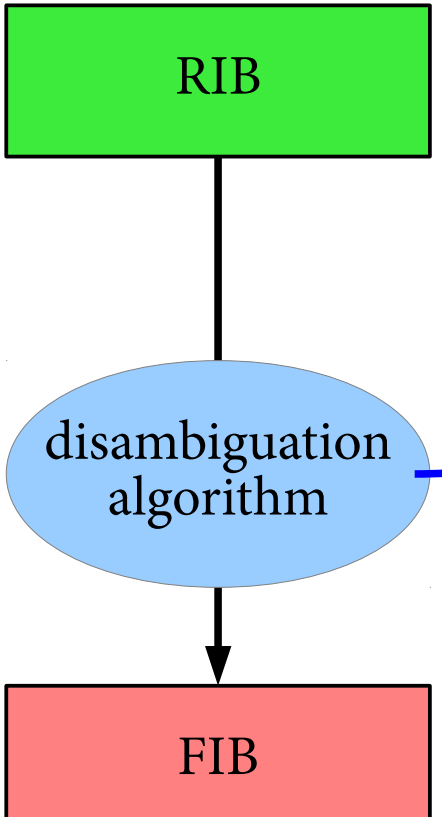
Most specific entries are preferred



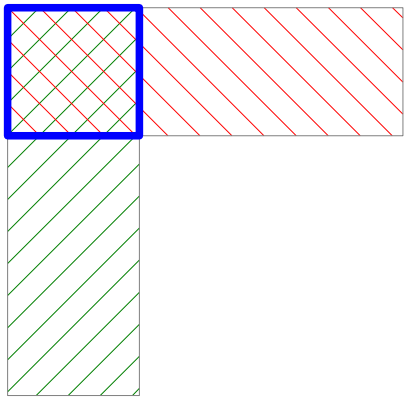
→ This behaves the same than the native source-specific FIB.
→ This FIB is not ambiguous.

Disambiguation algorithm (idea) *(disambiguation.c)*

destination first
(protocol choice)



Main idea: for each ambiguity, we maintain more specific entries (kernel only)

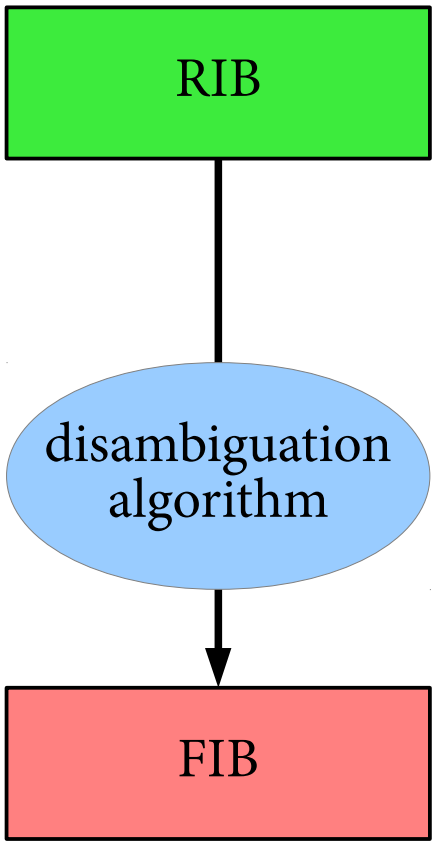


source or destination first:
we don't care, there is no ambiguity left

The algorithm is:

- incremental
- state less
  (don't remember additional routes)

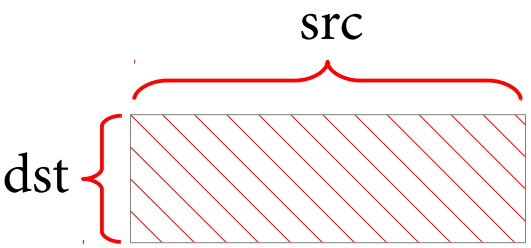
Example: initial state



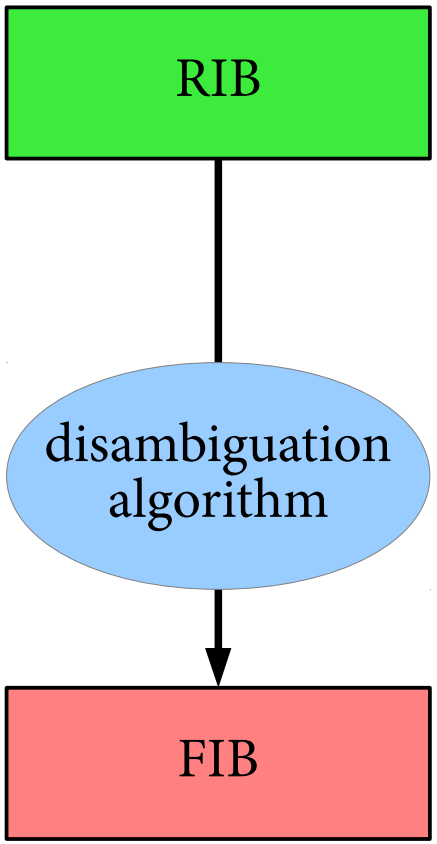
destination	source
2001:db8:2::/48	::/0

No ambiguity here:
the two tables are the same.

destination	source
2001:db8:2::/48	::/0



Example: adding a new route

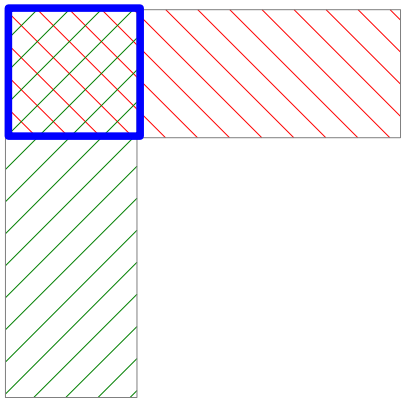


destination	source
2001:db8:2::/48	::/0
::/0	2001:db8:1::/48

1) The protocol receive an update for a new route.

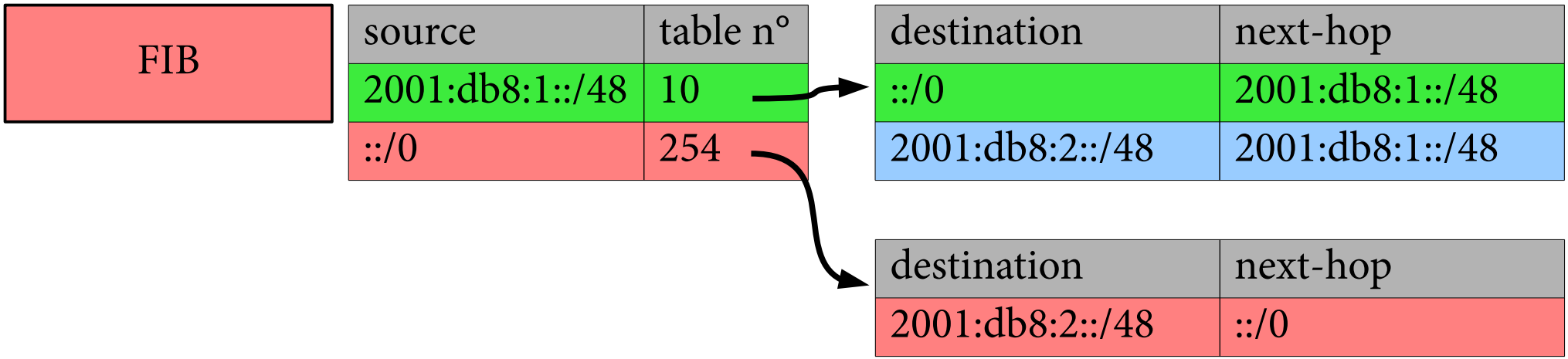
2) Before inserting it, we insert an additional route.

destination	source
2001:db8:2::/48	::/0
::/0	2001:db8:1::/48
2001:db8:2::/48	2001:db8:1::/48



(remark: with traffic engineering, it's multiple tables with rules)

Explicit traffic engineering



$ ip rule show
0: from all lookup local
100: from 192.168.4.0/24 lookup 10
32766: from all lookup main
32767: from all lookup default

$ ip route show table 10
default via 192.168.4.39 dev gre-omicron proto babel onlink
192.168.4.30 via 192.168.4.39 dev gre-omicron proto babel onlink
192.168.4.31 via 192.168.4.39 dev gre-omicron proto babel onlink
…

$ ip route show
default via 172.23.47.254 dev eth0
169.254.0.0/16 dev eth0 scope link metric 1000
172.23.32.0/20 dev eth0 proto kernel scope link src 172.23.36.45
192.168.4.30 via 192.168.4.39 dev gre-omicron proto babel onlink
192.168.4.31 via 192.168.4.39 dev gre-omicron proto babel onlink
…

(remark: with traffic engineering, it's multiple tables with rules)

Conclusion

There is two ways to achieve source-specific routing in Linux:

- Both are doable,
- Prefer native destination-first FIB (IPv6 subtrees),
- Otherwise, disambiguate with traffic engineering rules.

Our disambiguation algorithm is:

- protocol and kernel agnostic (layer between RIB and FIB),
- incremental,
- state less,
- proved correct.

  (more details in our article — source-specific routing, IFIP Networking 2015)

Feel free to use our code (MIT licensed):

- disambiguation.c: the disambiguation algorithm,
- rule.c: traffic engineering rules management.

  → https://github.com/jech/babeld