What makes the panda sad?

Jakub Sitnicki
Systems Engineer
Cloudflare

Linux Plumbers Conference
Networking Track
September 18, 2024
Vienna, Austria

What makes the panda sad?
... in the Linux network stack

Jakub Sitnicki
Systems Engineer
Cloudflare

Linux Plumbers Conference
Networking Track
September 18, 2024
Vienna, Austria

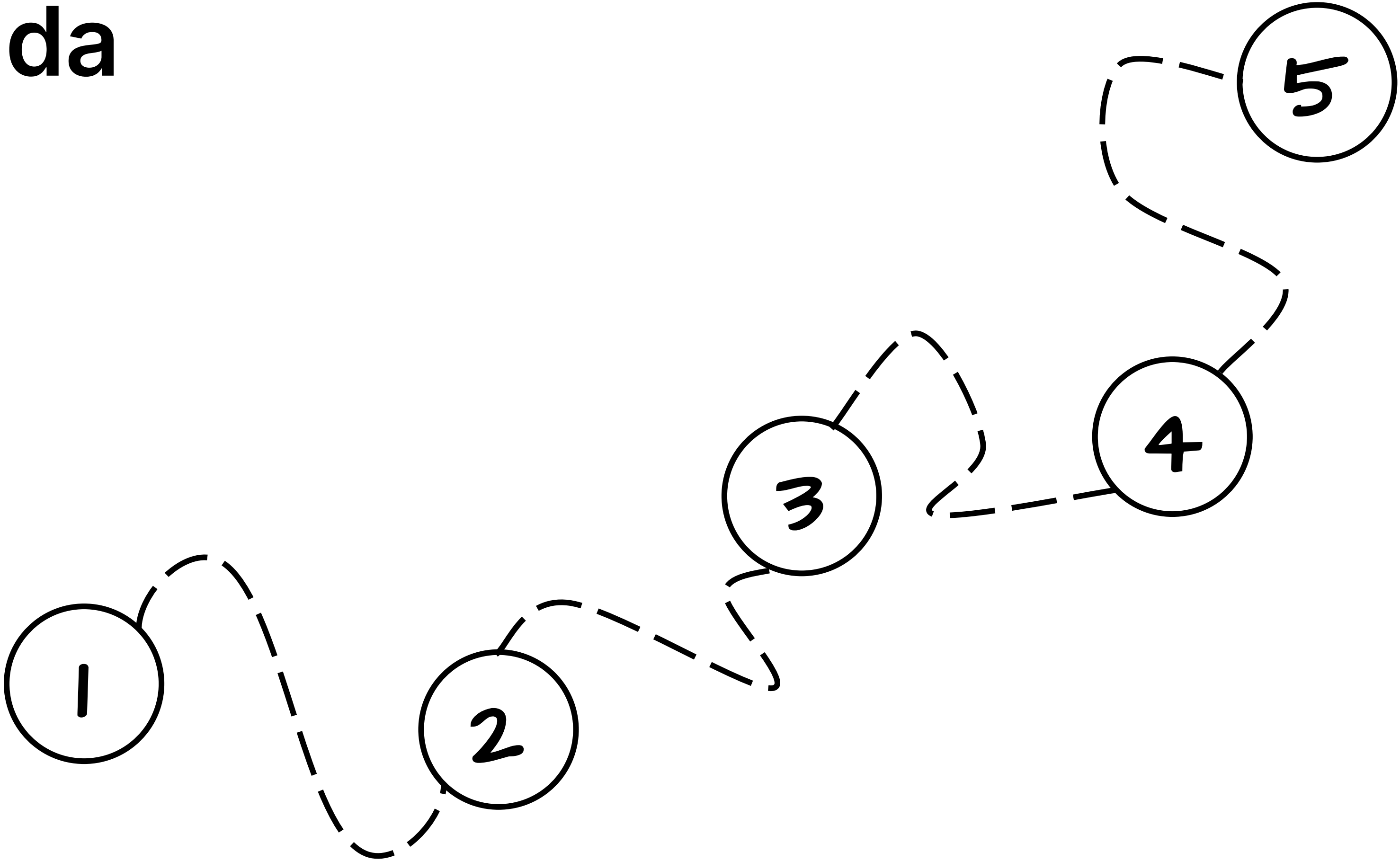
CLOUDFLARE

What makes the panda sad?
... in the Linux network stack
... today

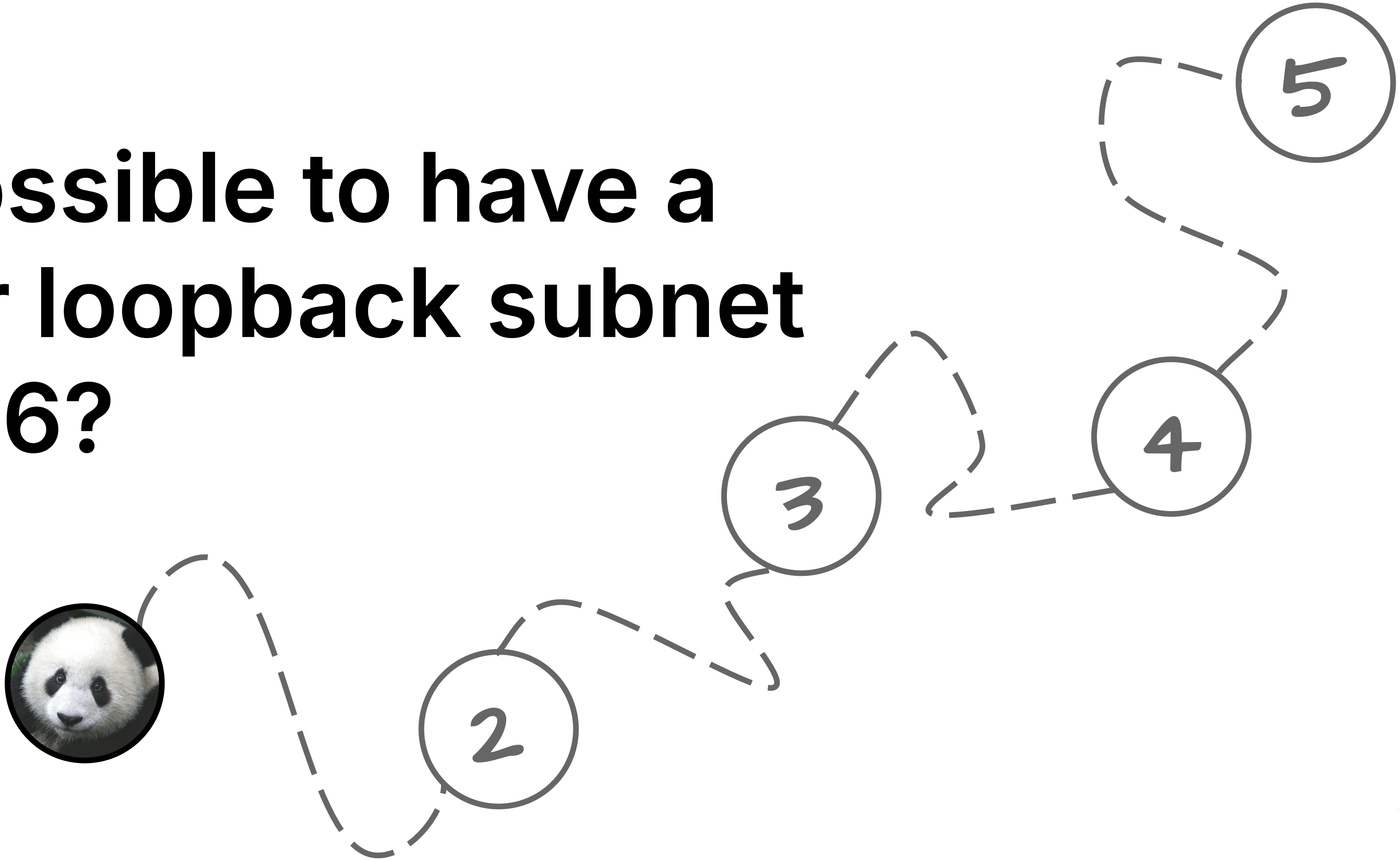
Jakub Sitnicki
Systems Engineer
Cloudflare

Linux Plumbers Conference
Networking Track
September 18, 2024
Vienna, Austria

Agenda



❶ Is it possible to have a proper loopback subnet for IPv6?



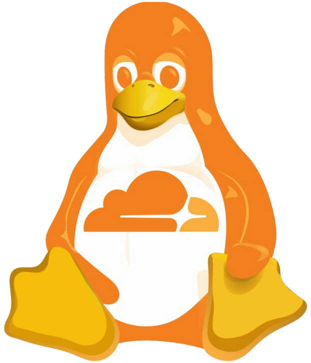
```
 ~ # ip -4 address show dev lo
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever

~ # ip -4 route show table local
local 127.0.0.0/8 dev lo proto kernel scope host src 127.0.0.1
local 127.0.0.1 dev lo proto kernel scope host src 127.0.0.1
broadcast 127.255.255.255 dev lo proto kernel scope link src 127.0.0.1

 ~ # ipcalc-ng --addresses 127.0.0.0/8
ADDRESSES=16777214
```

2^24 node-local addresses

```
 ~ # ip -6 address show dev lo
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever


 ~ # ip -6 route show table local
local ::1 dev lo proto kernel metric 0 pref medium
```

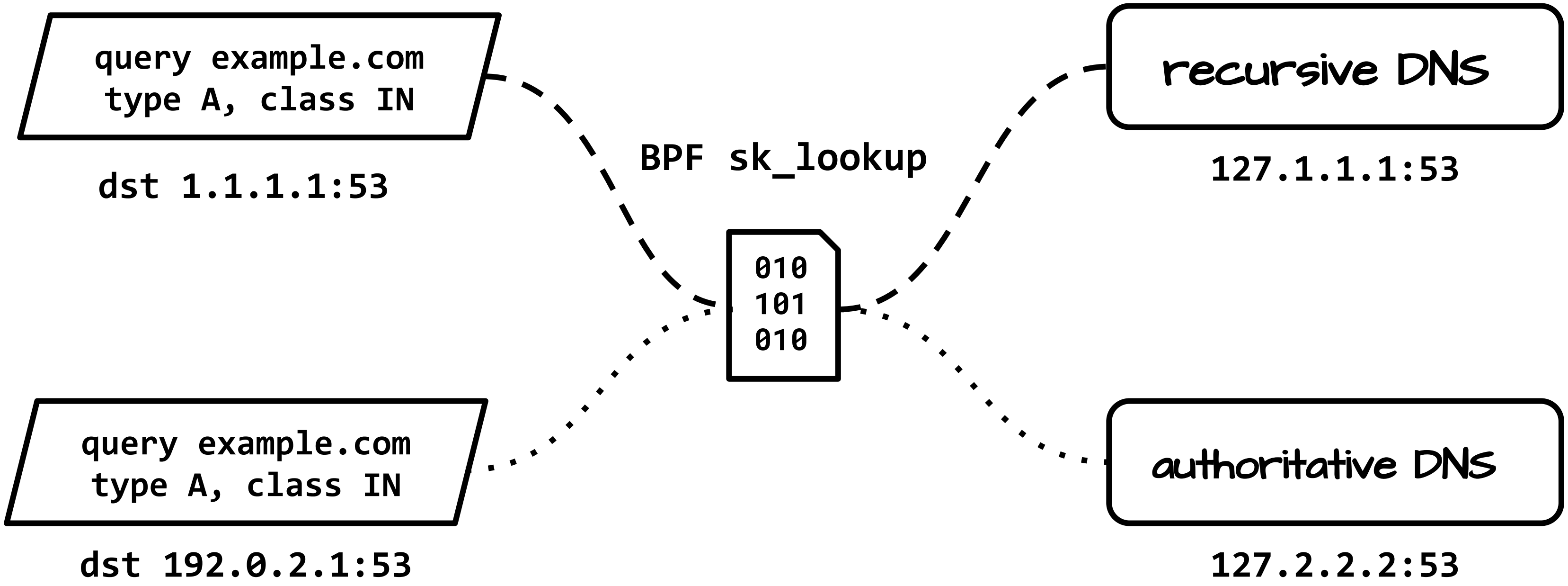
one node-local address

… but not Linux net stack fault



🫶 **AF_UNIX**

but...



query example.com
type A, class IN

dst 1.1.1.1:53

query example.com
type A, class IN

dst 192.0.2.1:53

BPF sk_lookup

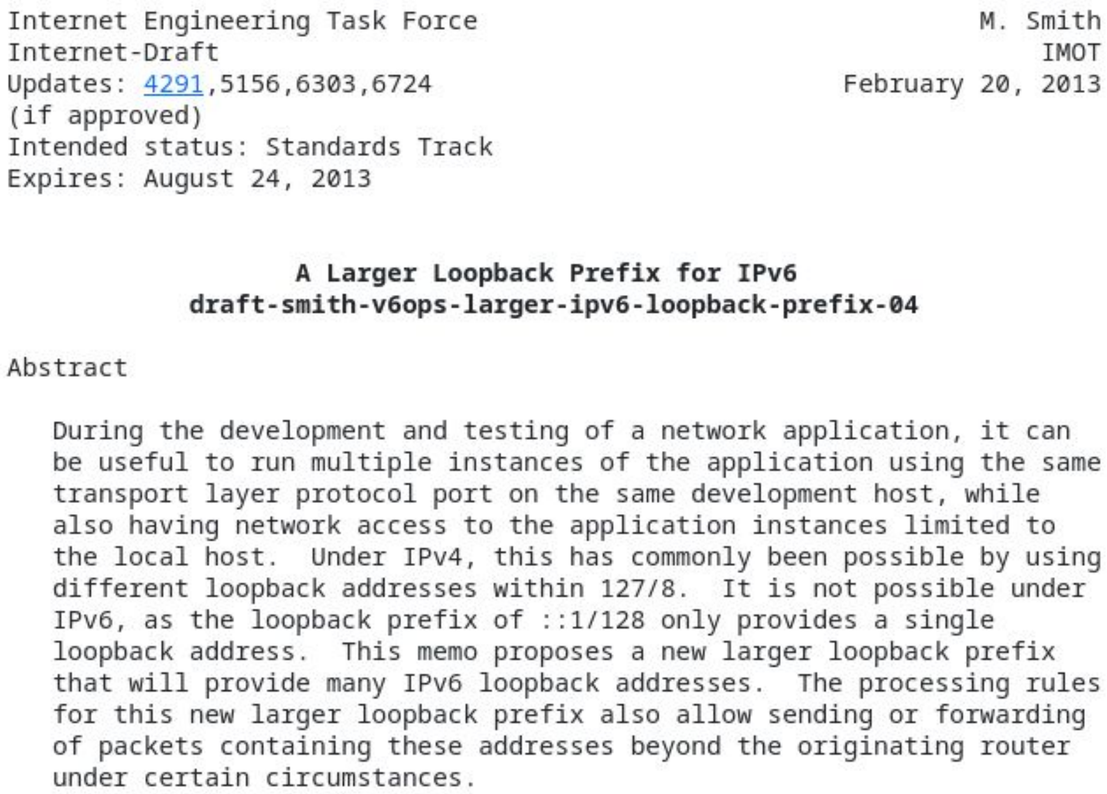
010
101
010

recursive DNS

127.1.1.1:53

authoritative DNS

127.2.2.2:53



Internet Engineering Task Force   M. Smith
Internet-Draft   IMOT
Updates: 4291,5156,6303,6724   February 20, 2013
(if approved)
Intended status: Standards Track
Expires: August 24, 2013

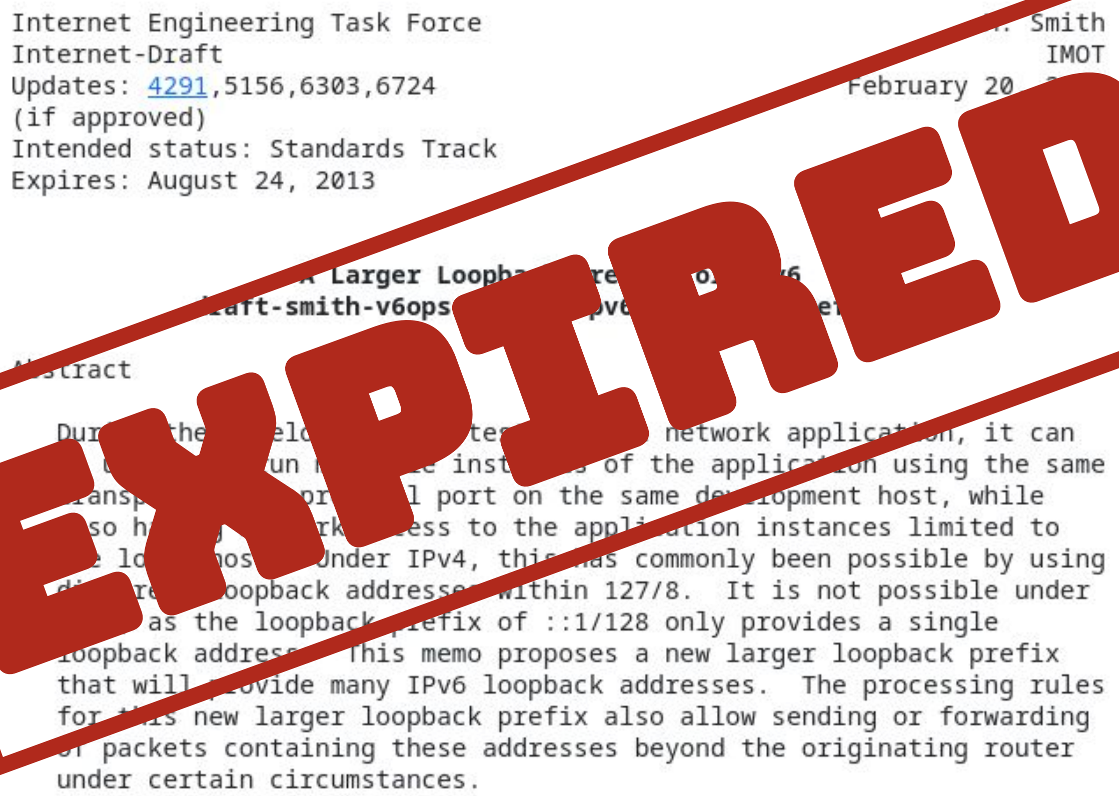
A Larger Loopback Prefix for IPv6
draft-smith-v6ops-larger-ipv6-loopback-prefix-04

Abstract

During the development and testing of a network application, it can be useful to run multiple instances of the application using the same transport layer protocol port on the same development host, while also having network access to the application instances limited to the local host. Under IPv4, this has commonly been possible by using different loopback addresses within 127/8. It is not possible under IPv6, as the loopback prefix of ::1/128 only provides a single loopback address. This memo proposes a new larger loopback prefix that will provide many IPv6 loopback addresses. The processing rules for this new larger loopback prefix also allow sending or forwarding of packets containing these addresses beyond the originating router under certain circumstances.

https://datatracker.ietf.org/doc/html/draft-smith-v6ops-larger-ipv6-loopback-prefix-04

CLOUDFLARE



Internet Engineering Task Force [illegible] Smith
Internet-Draft IMOT
Updates: 4291,5156,6303,6724 February 20, [illegible]
(if approved)
Intended status: Standards Track
Expires: August 24, 2013

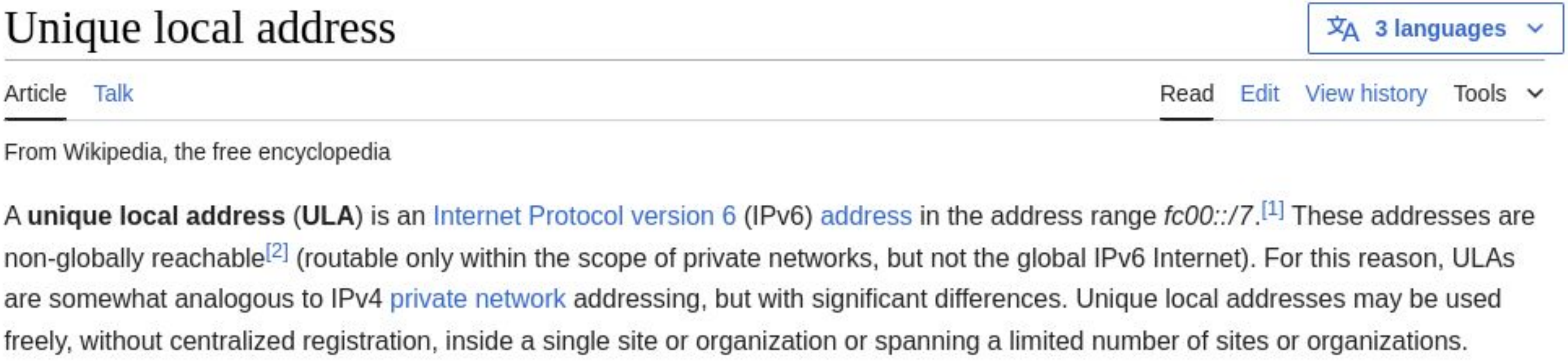
A Larger Loopba[illegible]re[illegible]o[illegible]6
[illegible]aft-smith-v6ops[illegible]pv6[illegible]e[illegible]

[illegible]stract

Dur[illegible] the [illegible]elo[illegible]te[illegible] network applica[illegible]on, it can [illegible] u[illegible]un [illegible]e inst[illegible]s of the applica[illegible]on using the same [illegible]ansp[illegible]or[illegible]l port on the same de[illegible]opment host, while [illegible]so h[illegible]rk[illegible]ess to the appl[illegible]ation instances limited to [illegible]e lo[illegible]os[illegible] Under IPv4, thi[illegible]as commonly been possible by using di[illegible]re[illegible]oopback addresses within 127/8. It is not possible under [illegible] as the loopback prefix of ::1/128 only provides a single loopback address. This memo proposes a new larger loopback prefix that will provide many IPv6 loopback addresses. The processing rules for this new larger loopback prefix also allow sending or forwarding of packets containing these addresses beyond the originating router under certain circumstances.

EXPIRED

https://datatracker.ietf.org/doc/html/draft-smith-v6ops-larger-ipv6-loopback-prefix-04

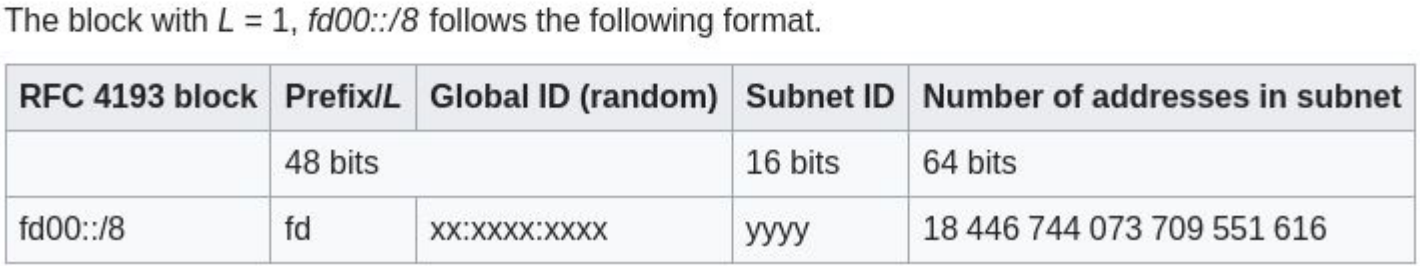


Unique local address

From Wikipedia, the free encyclopedia

A **unique local address** (**ULA**) is an Internet Protocol version 6 (IPv6) address in the address range *fc00::/7*.[1] These addresses are non-globally reachable[2] (routable only within the scope of private networks, but not the global IPv6 Internet). For this reason, ULAs are somewhat analogous to IPv4 private network addressing, but with significant differences. Unique local addresses may be used freely, without centralized registration, inside a single site or organization or spanning a limited number of sites or organizations.

https://en.wikipedia.org/wiki/Unique_local_address



The block with *L* = 1, *fd00::/8* follows the following format.

RFC 4193 block	Prefix/*L*	Global ID (random)	Subnet ID	Number of addresses in subnet
	48 bits		16 bits	64 bits
fd00::/8	fd	xx:xxxx:xxxx	yyyy	18 446 744 073 709 551 616

https://en.wikipedia.org/wiki/Unique_local_address

```
fdXX:XXXX:XXXX::/48
```

```
fdXX:XXXX:XXXX::/48
```

↓

```
fd00:1009:bacc::/48
```

```
fdXX:XXXX:XXXX::/48
```

⇩

```
fd00:1009:bacc::/48
```

⇩

F - D - double - 0 - LOOPBACK

`fdXX:XXXX:XXXX::/48`

⇩

`fd00:1009:bacc::/48`

⇩

F - D - double - O - LOOPBACK

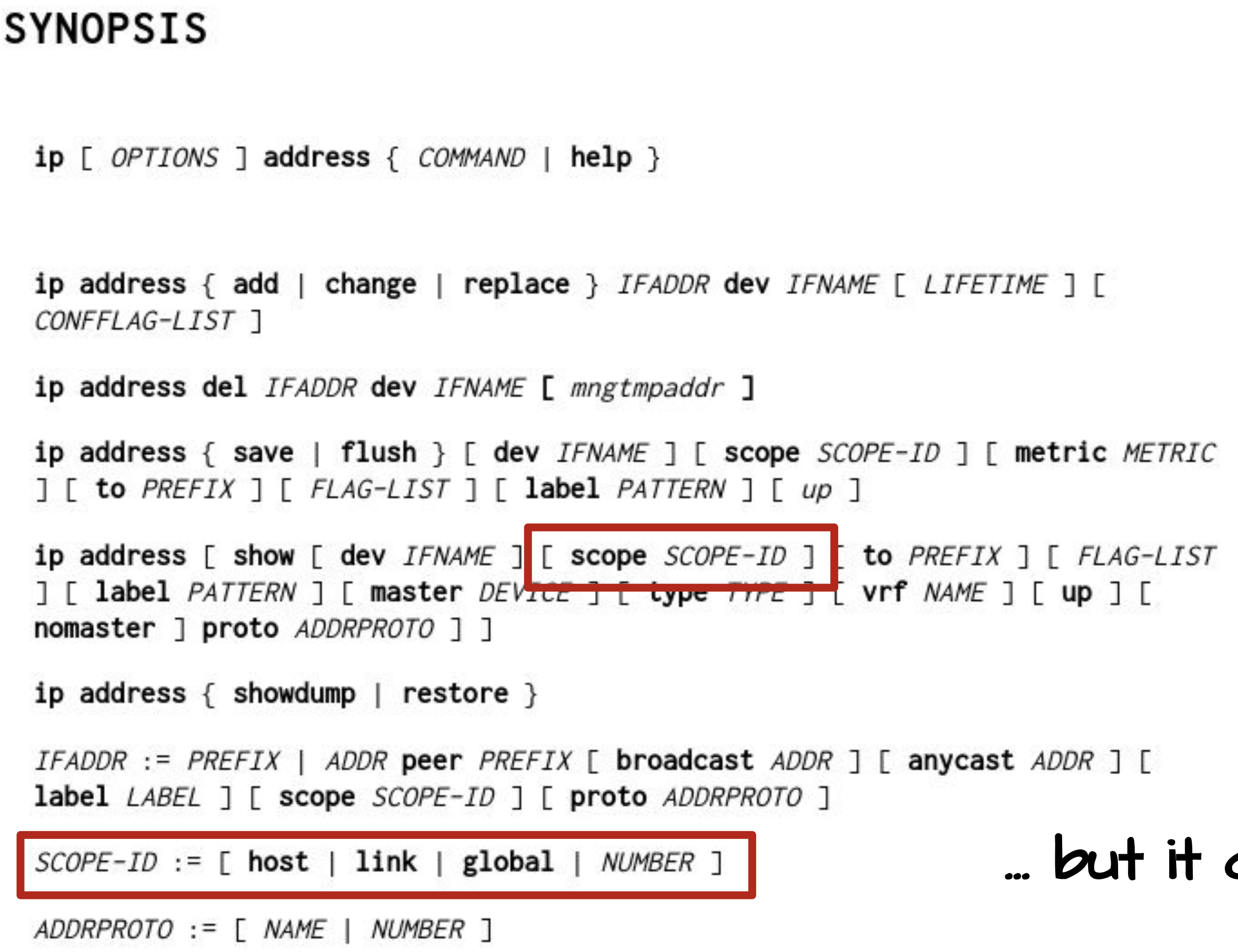
⇩

We want it to be:

1. locally assigned
2. node-local

```
 ~ # ip address add fd00:1009:bacc::1/48 dev lo
 ~ # ip -6 address show dev lo
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    inet6 fd00:1009:bacc::1/48 scope global
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
 ~ #
```

default scope is wrong...



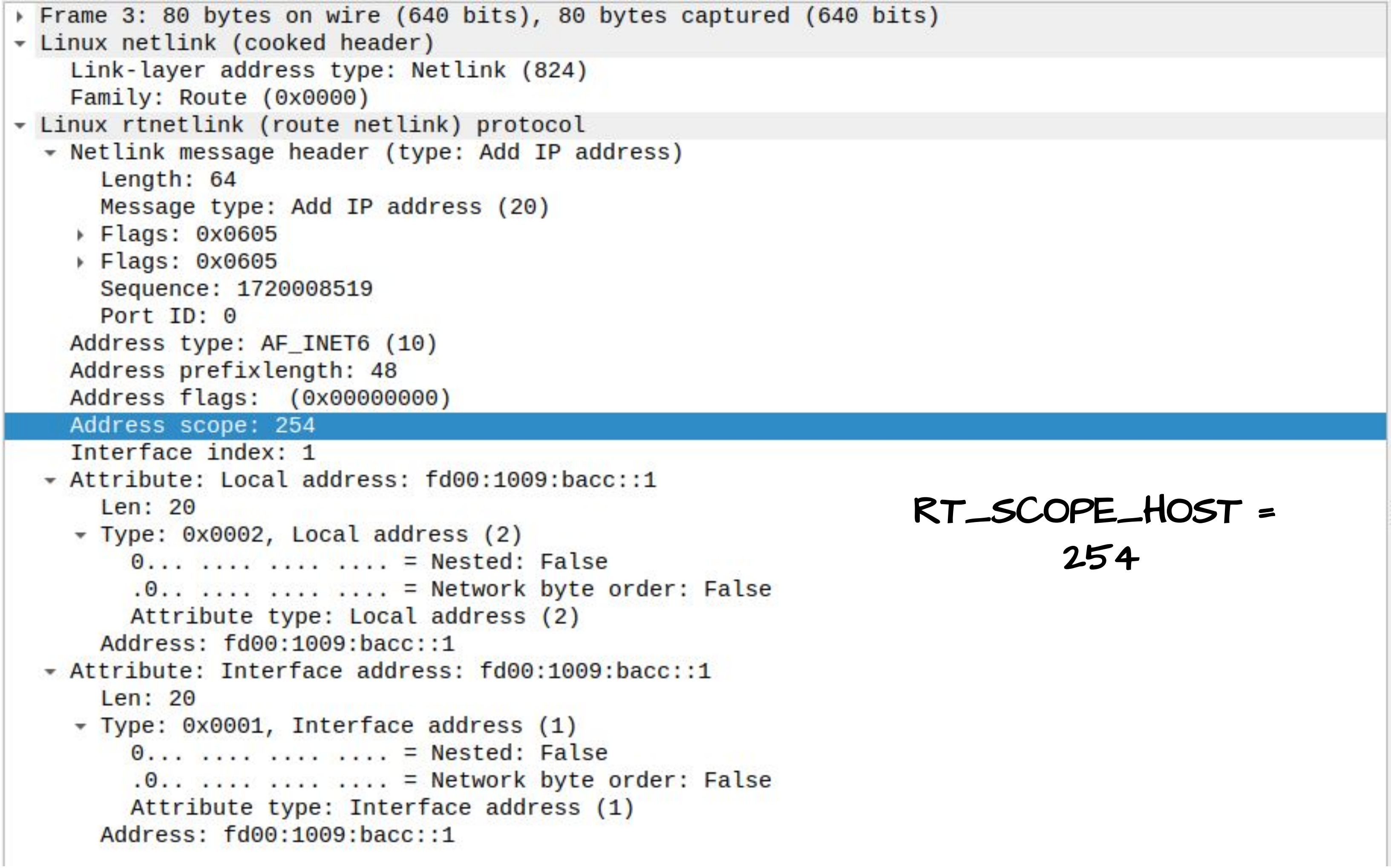
SYNOPSIS

```
ip [ OPTIONS ] address { COMMAND | help }


ip address { add | change | replace } IFADDR dev IFNAME [ LIFETIME ] [
CONFFLAG-LIST ]

ip address del IFADDR dev IFNAME [ mngtmpaddr ]

ip address { save | flush } [ dev IFNAME ] [ scope SCOPE-ID ] [ metric METRIC
] [ to PREFIX ] [ FLAG-LIST ] [ label PATTERN ] [ up ]

ip address [ show [ dev IFNAME ] [ scope SCOPE-ID ] [ to PREFIX ] [ FLAG-LIST
] [ label PATTERN ] [ master DEVICE ] [ type TYPE ] [ vrf NAME ] [ up ] [
nomaster ] proto ADDRPROTO ] ]

ip address { showdump | restore }

IFADDR := PREFIX | ADDR peer PREFIX [ broadcast ADDR ] [ anycast ADDR ] [
label LABEL ] [ scope SCOPE-ID ] [ proto ADDRPROTO ]

SCOPE-ID := [ host | link | global | NUMBER ]

ADDRPROTO := [ NAME | NUMBER ]
```

… but it can be set 🤩

```
 ~ # ip address add fd00:1009:bacc::1/48 scope host dev lo
 ~ # ip -6 address show dev lo
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    inet6 fd00:1009:bacc::1/48 scope global
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
 ~ #
```

:"-(



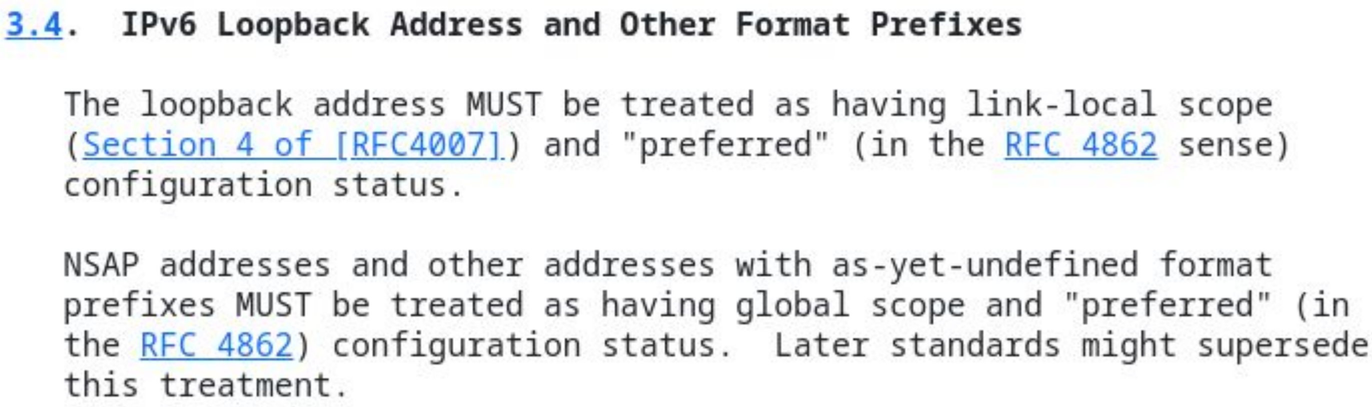
```
▸ Frame 3: 80 bytes on wire (640 bits), 80 bytes captured (640 bits)
▾ Linux netlink (cooked header)
    Link-layer address type: Netlink (824)
    Family: Route (0x0000)
▾ Linux rtnetlink (route netlink) protocol
  ▾ Netlink message header (type: Add IP address)
      Length: 64
      Message type: Add IP address (20)
    ▸ Flags: 0x0605
    ▸ Flags: 0x0605
      Sequence: 1720008519
      Port ID: 0
    Address type: AF_INET6 (10)
    Address prefixlength: 48
    Address flags:  (0x00000000)
    Address scope: 254
    Interface index: 1
  ▾ Attribute: Local address: fd00:1009:bacc::1
      Len: 20
    ▾ Type: 0x0002, Local address (2)
        0... .... .... .... = Nested: False
        .0.. .... .... .... = Network byte order: False
        Attribute type: Local address (2)
      Address: fd00:1009:bacc::1
  ▾ Attribute: Interface address: fd00:1009:bacc::1
      Len: 20
    ▾ Type: 0x0001, Interface address (1)
        0... .... .... .... = Nested: False
        .0.. .... .... .... = Network byte order: False
        Attribute type: Interface address (1)
      Address: fd00:1009:bacc::1
```

RT_SCOPE_HOST =
254

```
inet6_rtm_newaddr()            handles RTM_NEWADDR (20) request
  inet6_addr_add()             sets cfg->scope = ipv6_addr_scope(cfg->pfx)
    ipv6_addr_scope()
      __ipv6_addr_type()       determines address scope
    ipv6_add_addr()            sets ifa->scope = cfg->scope
```

passed scope is ignored ¯_(ツ)_/¯

__ipv6_addr_type() follows RFC 6724



3.4. IPv6 Loopback Address and Other Format Prefixes

The loopback address MUST be treated as having link-local scope (Section 4 of [RFC4007]) and "preferred" (in the RFC 4862 sense) configuration status.

NSAP addresses and other addresses with as-yet-undefined format prefixes MUST be treated as having global scope and "preferred" (in the RFC 4862) configuration status. Later standards might supersede this treatment.

https://datatracker.ietf.org/doc/html/rfc6724#section-3.4

1. we can't set address scope, AND
2. we want an address for local communication only

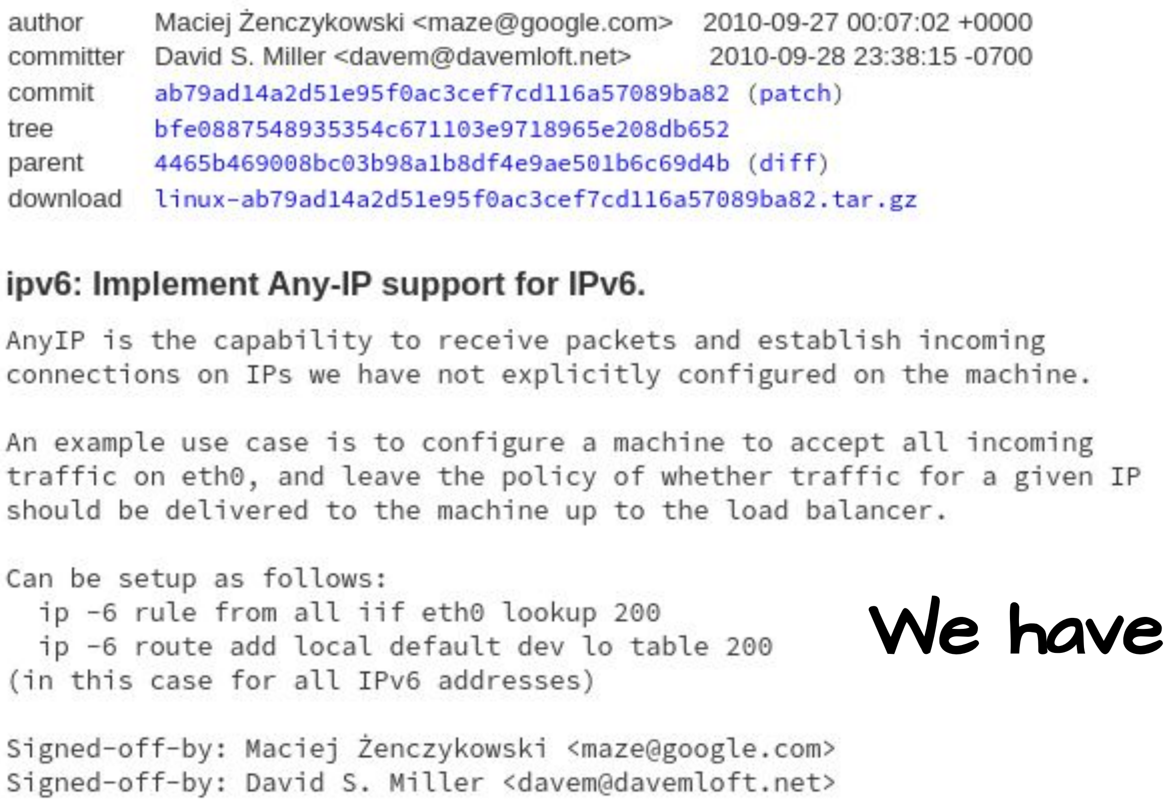
Workaround?

If we want a strict setup like the default
`net.ipv4.conf.all.route_localnet = 0`

```
nft add rule ip6 filter input ip6 saddr fd00:1009:bacc::/48 iifname != "lo" drop
nft add rule ip6 filter output ip6 daddr fd00:1009:bacc::/48 oifname != "lo" drop
```

☑ one local address

□ whole local subnet



author	Maciej Żenczykowski <maze@google.com>	2010-09-27 00:07:02 +0000
committer	David S. Miller <davem@davemloft.net>	2010-09-28 23:38:15 -0700
commit	ab79ad14a2d51e95f0ac3cef7cd116a57089ba82 (patch)	
tree	bfe0887548935354c671103e9718965e208db652	
parent	4465b469008bc03b98a1b8df4e9ae501b6c69d4b (diff)	
download	linux-ab79ad14a2d51e95f0ac3cef7cd116a57089ba82.tar.gz	

ipv6: Implement Any-IP support for IPv6.

```
AnyIP is the capability to receive packets and establish incoming
connections on IPs we have not explicitly configured on the machine.

An example use case is to configure a machine to accept all incoming
traffic on eth0, and leave the policy of whether traffic for a given IP
should be delivered to the machine up to the load balancer.

Can be setup as follows:
  ip -6 rule from all iif eth0 lookup 200
  ip -6 route add local default dev lo table 200
(in this case for all IPv6 addresses)

Signed-off-by: Maciej Żenczykowski <maze@google.com>
Signed-off-by: David S. Miller <davem@davemloft.net>
```

We have AnyIP

```
 ~ # ip -6 route add local fd00:1009:bacc::/48 dev lo src fd00:1009:bacc::1

 ~ # ip -6 route show table local
local ::1 dev lo proto kernel metric 0 pref medium
local fd00:1009:bacc::1 dev lo proto kernel metric 0 pref medium
local fd00:1009:bacc::/48 dev lo src fd00:1009:bacc::1 metric 1024 pref medium
```

1. routing treat all addresses from the subnet as local, on ingress and egress

2. ipv6 stack responds to ND requests on all of these addresses

🤩

Except you can't bind() to them…

```
~ # strace -e bind nc -6l fd00:1009:bacc::1 1111
bind(3, {sa_family=AF_INET6, sin6_port=htons(1111), sin6_flowinfo=htonl(0),
inet_pton(AF_INET6, "fd00:1009:bacc::1", &sin6_addr), sin6_scope_id=0}, 28) = 0
^Cstrace: Process 2208695 detached
~ #

~ # strace -e bind nc -6l fd00:1009:bacc::dead 1111
bind(3, {sa_family=AF_INET6, sin6_port=htons(1111), sin6_flowinfo=htonl(0),
inet_pton(AF_INET6, "fd00:1009:bacc::dead", &sin6_addr), sin6_scope_id=0}, 28) = -1
EADDRNOTAVAIL (Cannot assign requested address)
nc: Cannot assign requested address
+++ exited with 1 +++
~ #
```

:"-(

```
~ # perf ftrace -C3 -G inet6_bind --graph-opts=noirqs | cat
# tracer: function_graph
#
# CPU  DURATION                  FUNCTION CALLS
# |     |   |                     |   |   |   |
  3)               |  inet6_bind() {
  3)               |    __inet6_bind() {
  3)   0.244 us    |      __ipv6_addr_type(); /* = 0xe0001 */
  3)               |      ipv6_chk_addr() {
  3)               |        __ipv6_chk_addr_and_flags() {
  3)   1.106 us    |        } /* __ipv6_chk_addr_and_flags = 0x0 */
  3)   1.607 us    |      } /* ipv6_chk_addr = 0x0 */ 👎
  3)   7.635 us    |    } /* __inet6_bind = -99 */
  3) + 19.603 us   |  } /* inet6_bind = -99 */
~ #
~ # perf ftrace -C3 -G inet6_bind --graph-opts=noirqs | cat
# tracer: function_graph
#
# CPU  DURATION                  FUNCTION CALLS
# |     |   |                      |   |   |   |
  3)               |  inet6_bind() {
  3)               |    __inet6_bind() {
  3)   0.244 us    |      __ipv6_addr_type(); /* = 0xe0001 */
  3)               |      ipv6_chk_addr() {
  3)               |        __ipv6_chk_addr_and_flags() {
  3)   1.106 us    |        } /* __ipv6_chk_addr_and_flags = 0x0 */
  3)   1.607 us    |      } /* ipv6_chk_addr = 0x0 */ 👎
  3)   7.635 us    |    } /* __inet6_bind = -99 */
  3) + 19.603 us   |  } /* inet6_bind = -99 */
~ #
```

__inet6_bind()

```
if (!(addr_type & IPV6_ADDR_MULTICAST)) {
    if (!ipv6_can_nonlocal_bind(net, inet) &&
        !ipv6_chk_addr(net, &addr->sin6_addr,
               dev, 0)) {
        err = -EADDRNOTAVAIL;
        goto out_unlock;
    }
}
```

```
~ # perf ftrace -C3 -G inet6_bind --graph-opts=noirqs | cat
# tracer: function_graph
#
# CPU  DURATION                  FUNCTION CALLS
# |     |   |                     |   |   |   |
  3)               |  inet6_bind() {
  3)               |    __inet6_bind() {
  3)   0.244 us    |      __ipv6_addr_type(); /* = 0xe0001 */
  3)               |      ipv6_chk_addr() {
  3)               |        __ipv6_chk_addr_and_flags() {
  3)   1.106 us    |        } /* __ipv6_chk_addr_and_flags = 0x0 */
  3)   1.607 us    |      } /* ipv6_chk_addr = 0x0 */ 👎
  3)   7.635 us    |    } /* __inet6_bind = -99 */
  3) + 19.603 us  |  } /* inet6_bind = -99 */
~ #
```

__inet6_bind()

```
if (!(addr_type & IPV6_ADDR_MULTICAST)) {
    if (!ipv6_can_nonlocal_bind(net, inet) &&
        !ipv6_chk_addr(net, &addr->sin6_addr,
               dev, 0)) {
        err = -EADDRNOTAVAIL;
        goto out_unlock;
    }
}
```

ipv6_can_nonlocal_bind()

```
static inline bool ipv6_can_nonlocal_bind(struct net *net,
                                          struct inet_sock *inet)
{
    return net->ipv6.sysctl.ip_nonlocal_bind ||
        test_bit(INET_FLAGS_FREEBIND, &inet->inet_flags) ||
        test_bit(INET_FLAGS_TRANSPARENT, &inet->inet_flags);
}
```

ip_nonlocal_bind
sysctl
IPV6_FREEBIND
sockopt
imgflip.com
JAKE-CLARK.TUMBLR

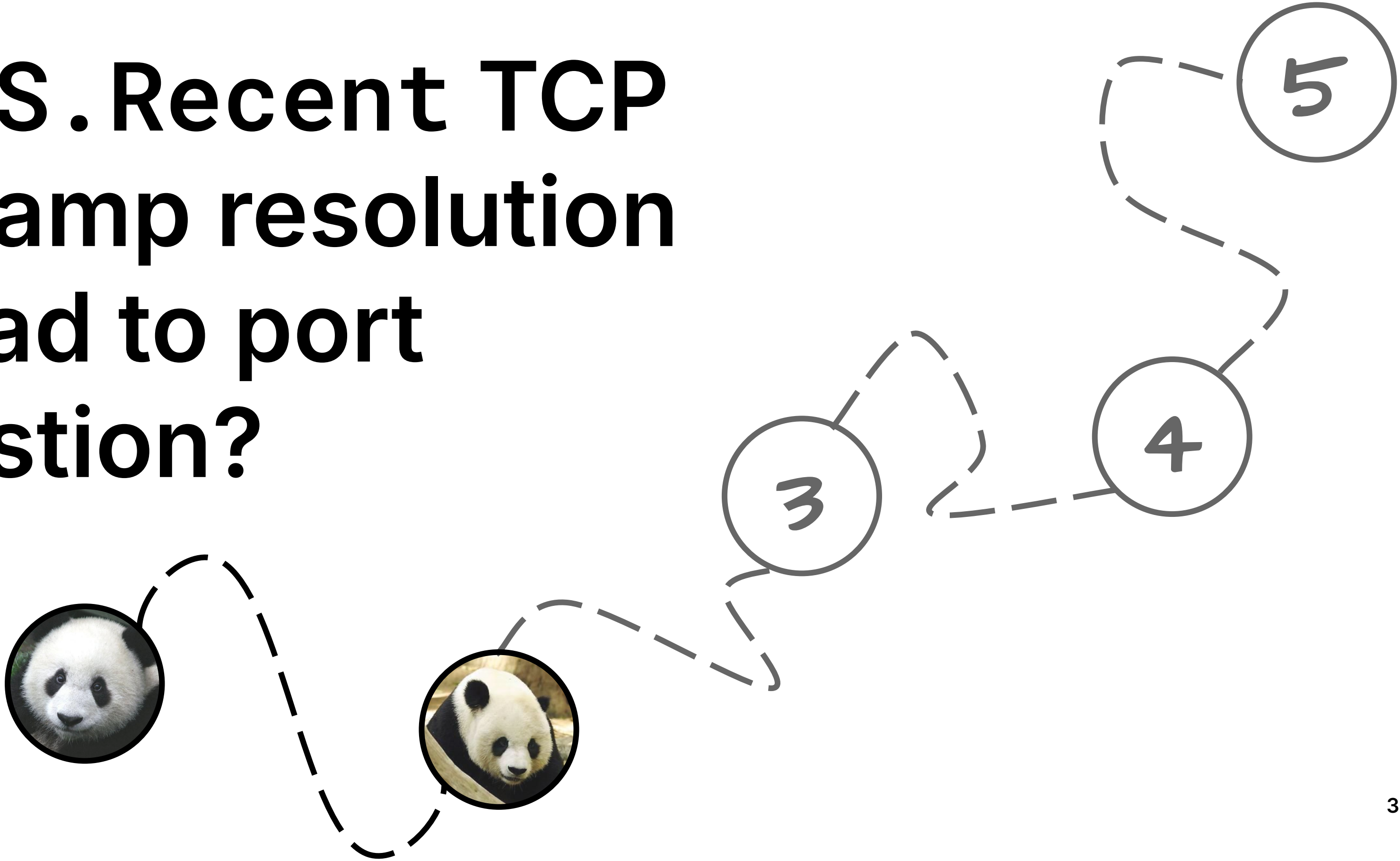
💡 IDEA

- Enable setting `IPV6_FREEBIND` with `bpf_setsockopt()`
- Call it from `BPF_CGROUP_INET[46]_BIND` hook when addr matches

Allows for a finer policy than `ip_non_local_bind` sysctl

❷

How TS.Recent TCP timestamp resolution can lead to port exhaustion?



```
#!/bin/env python3
# this_works.py

from socket import *
from time import sleep

ln = socket(AF_INET, SOCK_STREAM)
ln.setsockopt(SOL_SOCKET, SO_REUSEADDR, 1)
ln.bind(('127.1.1.1', 1111))
ln.listen(SOMAXCONN)

for _ in range(1000):
    s = socket(AF_INET, SOCK_STREAM)
    s.setsockopt(SOL_SOCKET, SO_REUSEADDR, 1)
    s.bind(('127.2.2.2', 2222))
    s.connect_ex(('127.1.1.1', 1111))
    s.close()

    sleep(0.010) # wait 10 msec
```

```
~ # strace -c -e connect ./this_works.py
% time     seconds  usecs/call     calls    errors syscall
------ ----------- ----------- --------- --------- ----------------
100.00    0.044373          44      1000           connect
------ ----------- ----------- --------- --------- ----------------
100.00    0.044373          44      1000           total
~ #
```

This works

👍

```python
#!/bin/env python3
# this_doesnt.py

from socket import *
from time import sleep

IP_LOCAL_PORT_RANGE = 51

# listener setup as last time…

for _ in range(1000):
    s = socket(AF_INET, SOCK_STREAM)
    s.setsockopt(SOL_IP, IP_BIND_ADDRESS_NO_PORT, 1)
    s.setsockopt(SOL_IP, IP_LOCAL_PORT_RANGE, 44_444 << 16 | 44_444)
    s.bind(("127.2.2.2", 0))
    s.connect_ex(("127.1.1.1", 1111))  # ignore errors
    s.close()

    sleep(0.010)  # wait 10 msec
```

use single-port ephemeral range

```
# strace -c -e connect ./this_doesnt.py
% time     seconds  usecs/call     calls    errors syscall
------ ----------- ----------- --------- --------- ----------------
100.00    0.008996           8      1000       988 connect
------ ----------- ----------- --------- --------- ----------------
100.00    0.008996           8      1000       988 total
#
```

This
doesn’t 👎

```
# strace -tt -z -e connect ./this_doesnt.py
22:01:34.839231 connect(4, {sa_family=AF_INET, sin_port=htons(1111), sin_addr=inet_addr("127.1.1.1")}, 16) = 0
22:01:34.911347 connect(4, {sa_family=AF_INET, sin_port=htons(1111), sin_addr=inet_addr("127.1.1.1")}, 16) = 0
22:01:35.913553 connect(4, {sa_family=AF_INET, sin_port=htons(1111), sin_addr=inet_addr("127.1.1.1")}, 16) = 0
22:01:36.916284 connect(4, {sa_family=AF_INET, sin_port=htons(1111), sin_addr=inet_addr("127.1.1.1")}, 16) = 0
22:01:37.908789 connect(4, {sa_family=AF_INET, sin_port=htons(1111), sin_addr=inet_addr("127.1.1.1")}, 16) = 0
22:01:38.909912 connect(4, {sa_family=AF_INET, sin_port=htons(1111), sin_addr=inet_addr("127.1.1.1")}, 16) = 0
22:01:39.909908 connect(4, {sa_family=AF_INET, sin_port=htons(1111), sin_addr=inet_addr("127.1.1.1")}, 16) = 0
22:01:40.915194 connect(4, {sa_family=AF_INET, sin_port=htons(1111), sin_addr=inet_addr("127.1.1.1")}, 16) = 0
22:01:41.911430 connect(4, {sa_family=AF_INET, sin_port=htons(1111), sin_addr=inet_addr("127.1.1.1")}, 16) = 0
22:01:42.910067 connect(4, {sa_family=AF_INET, sin_port=htons(1111), sin_addr=inet_addr("127.1.1.1")}, 16) = 0
22:01:43.912075 connect(4, {sa_family=AF_INET, sin_port=htons(1111), sin_addr=inet_addr("127.1.1.1")}, 16) = 0
22:01:44.908675 connect(4, {sa_family=AF_INET, sin_port=htons(1111), sin_addr=inet_addr("127.1.1.1")}, 16) = 0
22:01:45.507782 +++ exited with 0 +++
#
```

We succeed once every second

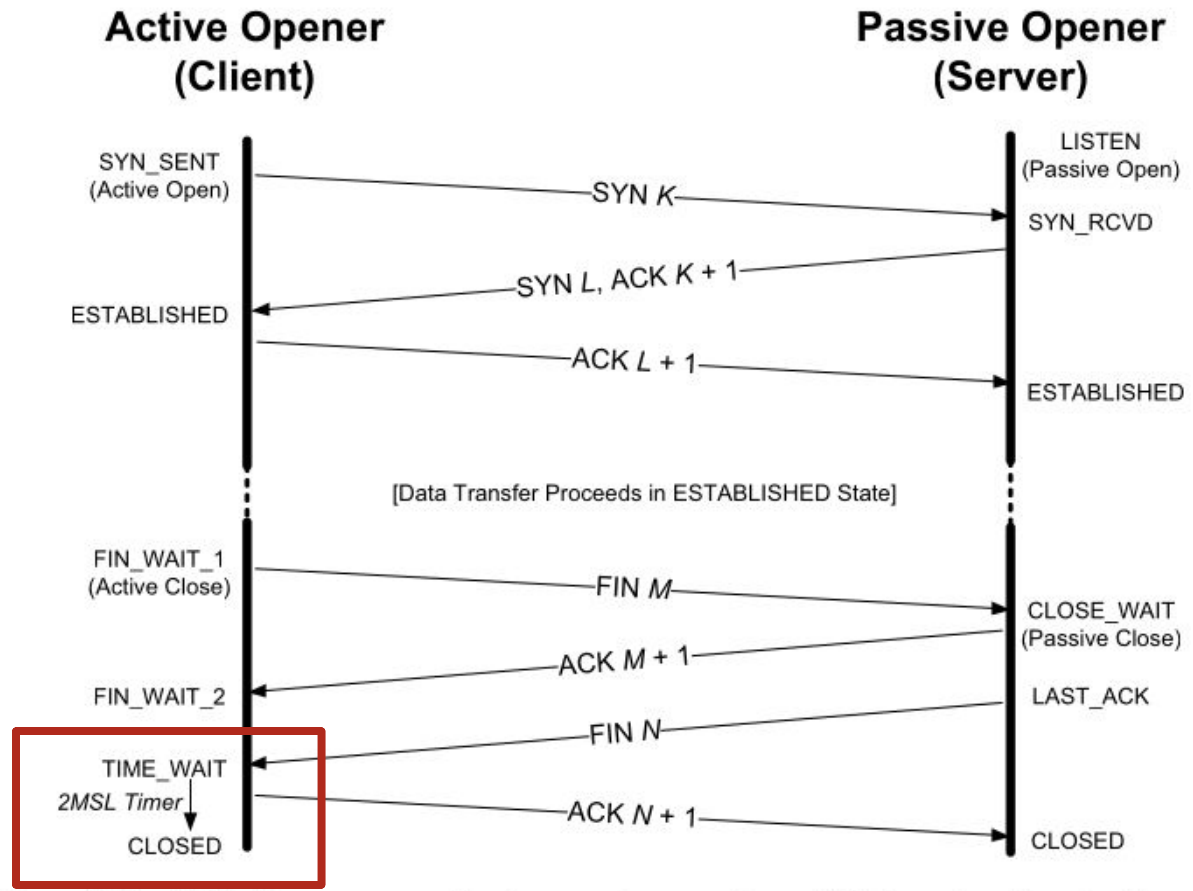
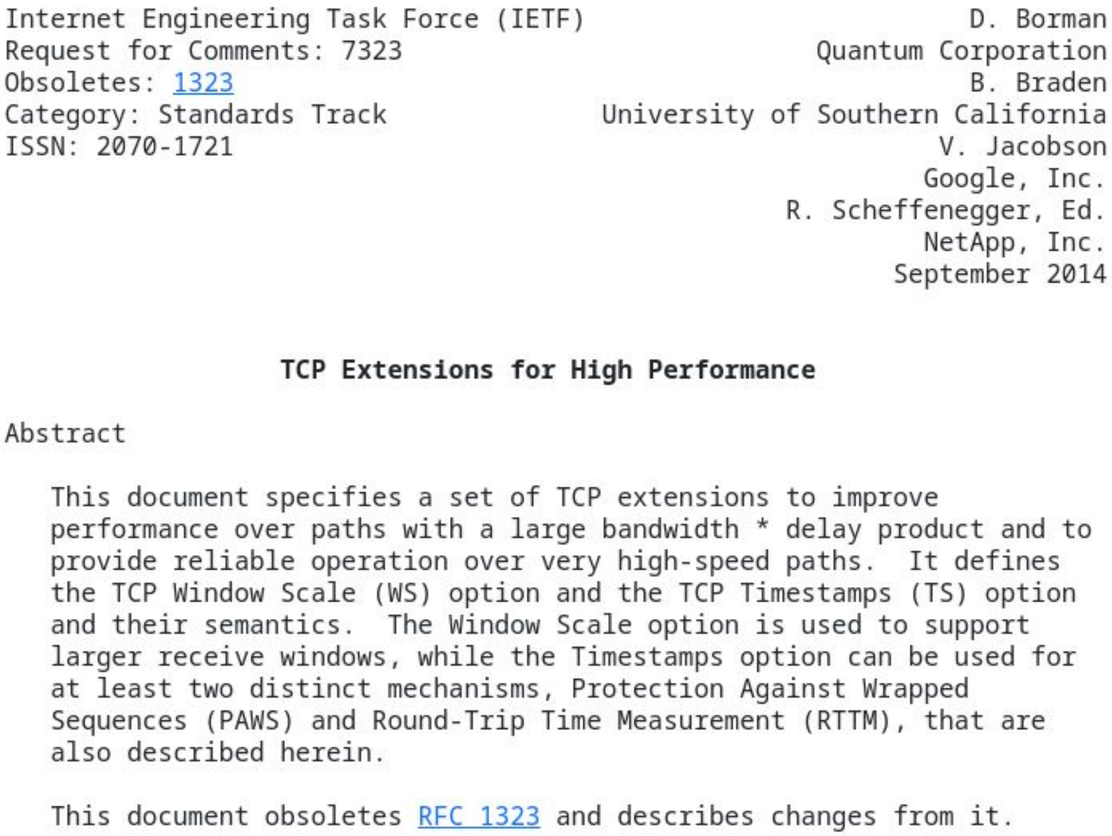


Figure 13-9 TCP states corresponding to normal connection establishment and termination

Stevens, W. Richard; Fall, Kevin R.; Wright, Gary R. (1994). TCP/IP illustrated (Volume 1): The Protocols

```
# sysctl net.ipv4.tcp_fin_timeout
net.ipv4.tcp_fin_timeout = 60
```

☜ ***2MSL timeout in seconds***



Internet Engineering Task Force (IETF)
Request for Comments: 7323
Obsoletes: 1323
Category: Standards Track
ISSN: 2070-1721

D. Borman
Quantum Corporation
B. Braden
University of Southern California
V. Jacobson
Google, Inc.
R. Scheffenegger, Ed.
NetApp, Inc.
September 2014

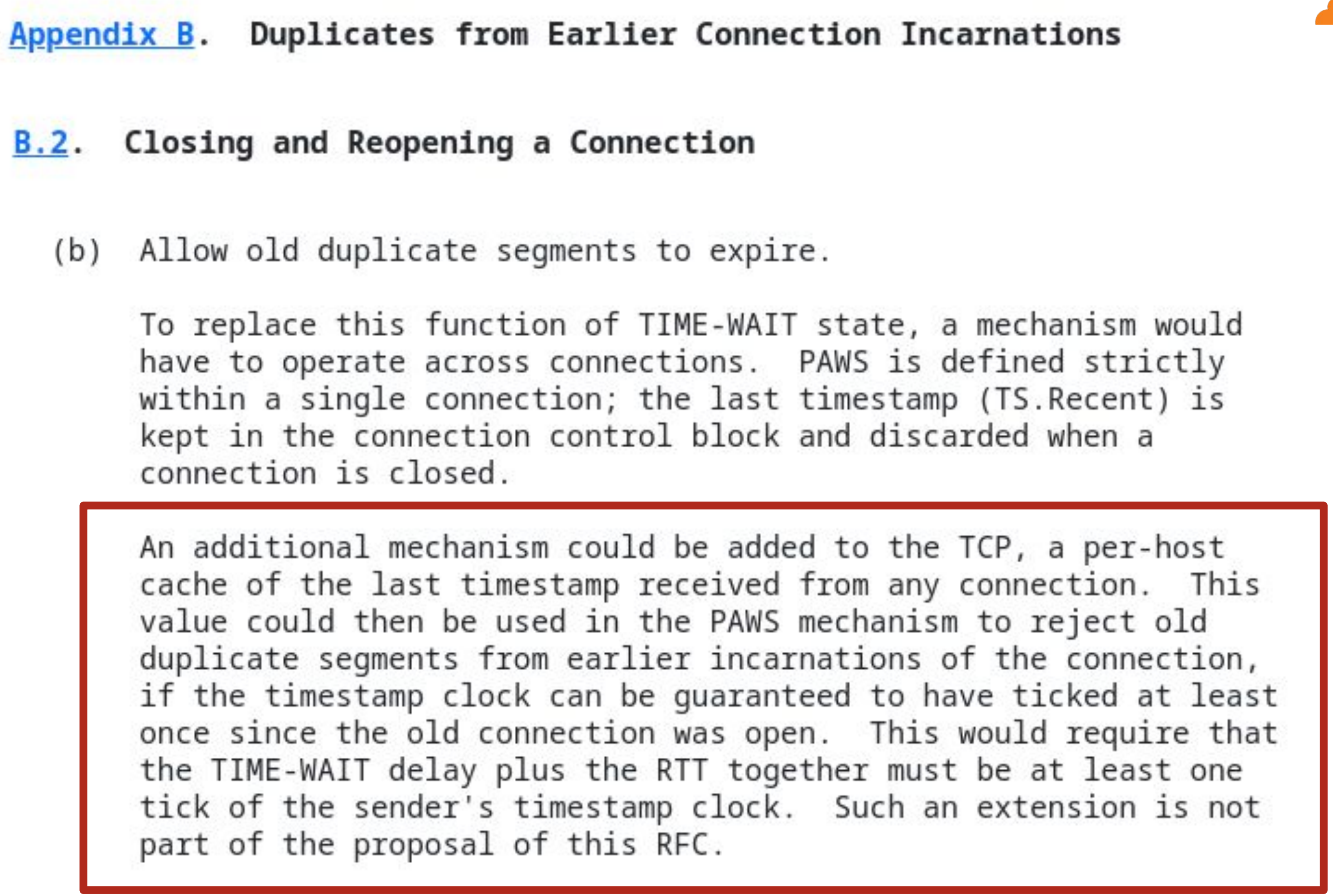
TCP Extensions for High Performance

Abstract

This document specifies a set of TCP extensions to improve performance over paths with a large bandwidth * delay product and to provide reliable operation over very high-speed paths. It defines the TCP Window Scale (WS) option and the TCP Timestamps (TS) option and their semantics. The Window Scale option is used to support larger receive windows, while the Timestamps option can be used for at least two distinct mechanisms, Protection Against Wrapped Sequences (PAWS) and Round-Trip Time Measurement (RTTM), that are also described herein.

This document obsoletes RFC 1323 and describes changes from it.

https://datatracker.ietf.org/doc/html/rfc7323



Appendix B. Duplicates from Earlier Connection Incarnations

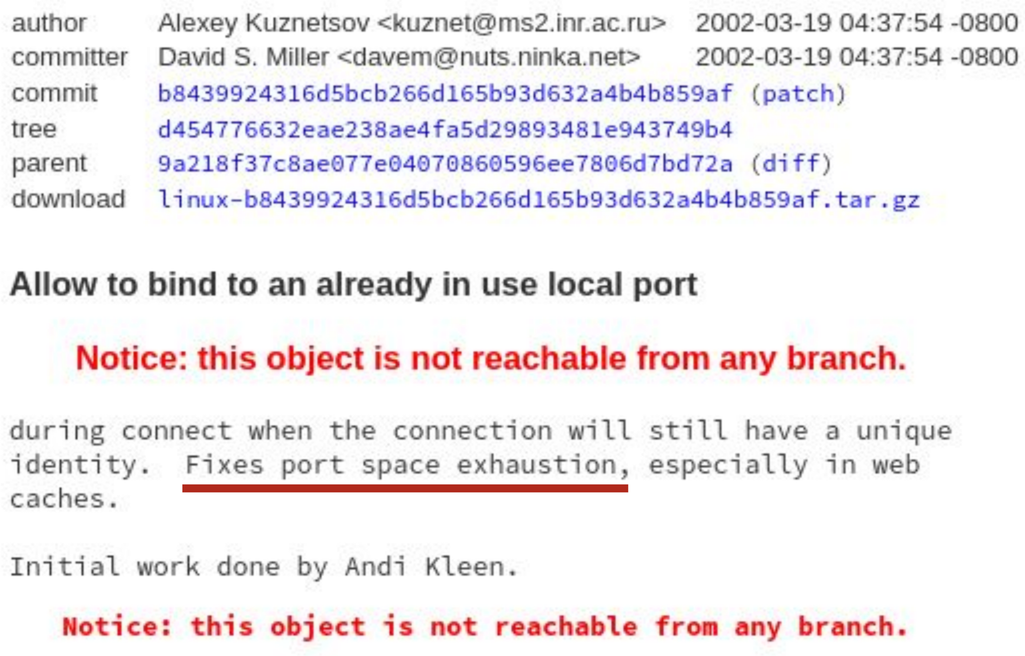
B.2. Closing and Reopening a Connection

(b) Allow old duplicate segments to expire.

To replace this function of TIME-WAIT state, a mechanism would have to operate across connections. PAWS is defined strictly within a single connection; the last timestamp (TS.Recent) is kept in the connection control block and discarded when a connection is closed.

An additional mechanism could be added to the TCP, a per-host cache of the last timestamp received from any connection. This value could then be used in the PAWS mechanism to reject old duplicate segments from earlier incarnations of the connection, if the timestamp clock can be guaranteed to have ticked at least once since the old connection was open. This would require that the TIME-WAIT delay plus the RTT together must be at least one tick of the sender's timestamp clock. Such an extension is not part of the proposal of this RFC.

https://datatracker.ietf.org/doc/html/rfc7323#appendix-B.2



author	Alexey Kuznetsov <kuznet@ms2.inr.ac.ru>	2002-03-19 04:37:54 -0800
committer	David S. Miller <davem@nuts.ninka.net>	2002-03-19 04:37:54 -0800
commit	b8439924316d5bcb266d165b93d632a4b4b859af (patch)	
tree	d454776632eae238ae4fa5d29893481e943749b4	
parent	9a218f37c8ae077e04070860596ee7806d7bd72a (diff)	
download	linux-b8439924316d5bcb266d165b93d632a4b4b859af.tar.gz	

Allow to bind to an already in use local port

Notice: this object is not reachable from any branch.

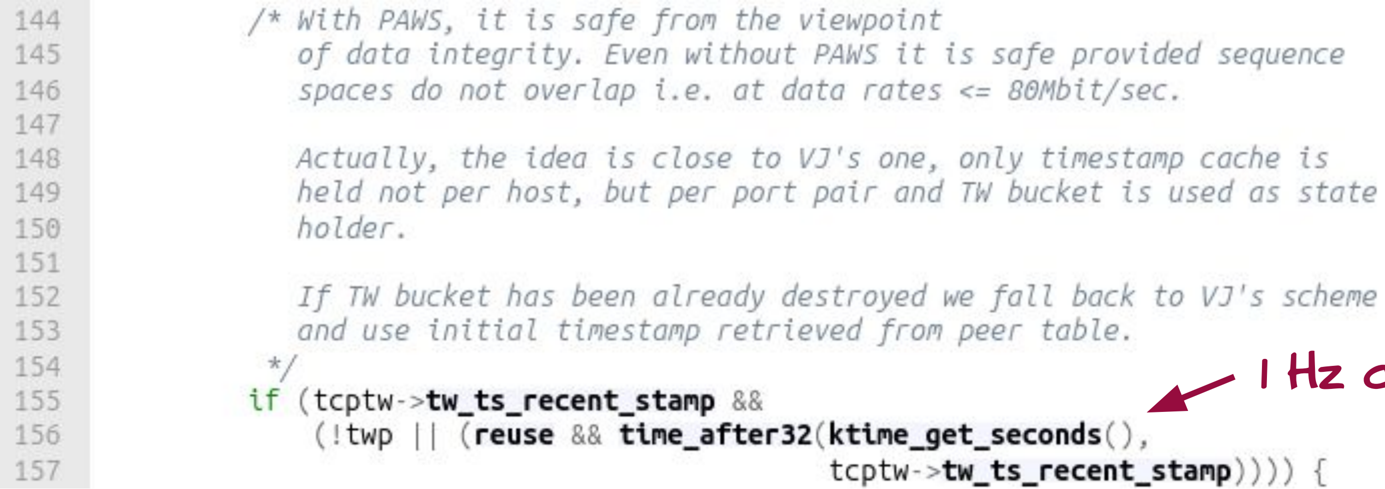
```
during connect when the connection will still have a unique
identity.  Fixes port space exhaustion, especially in web
caches.

Initial work done by Andi Kleen.
```

Notice: this object is not reachable from any branch.

https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/commit/?id=b8439924316d5bcb266d165b93d632a4b4b859af

```
tcp_tw_reuse - INTEGER
        Enable reuse of TIME-WAIT sockets for new connections when it is
        safe from protocol viewpoint.

        - 0 - disable
        - 1 - global enable
        - 2 - enable for loopback traffic only

        It should not be changed without advice/request of technical
        experts.

        Default: 2
```

tcp_twsk_unique()

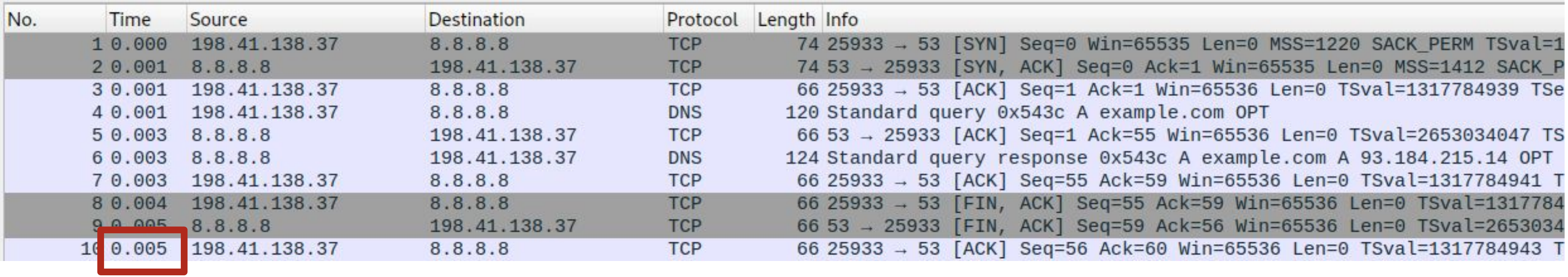


```
/* With PAWS, it is safe from the viewpoint
   of data integrity. Even without PAWS it is safe provided sequence
   spaces do not overlap i.e. at data rates <= 80Mbit/sec.

   Actually, the idea is close to VJ's one, only timestamp cache is
   held not per host, but per port pair and TW bucket is used as state
   holder.

   If TW bucket has been already destroyed we fall back to VJ's scheme
   and use initial timestamp retrieved from peer table.
 */
if (tcptw->tw_ts_recent_stamp &&
    (!twp || (reuse && time_after32(ktime_get_seconds(),
                                    tcptw->tw_ts_recent_stamp)))) {
```

1 Hz clock

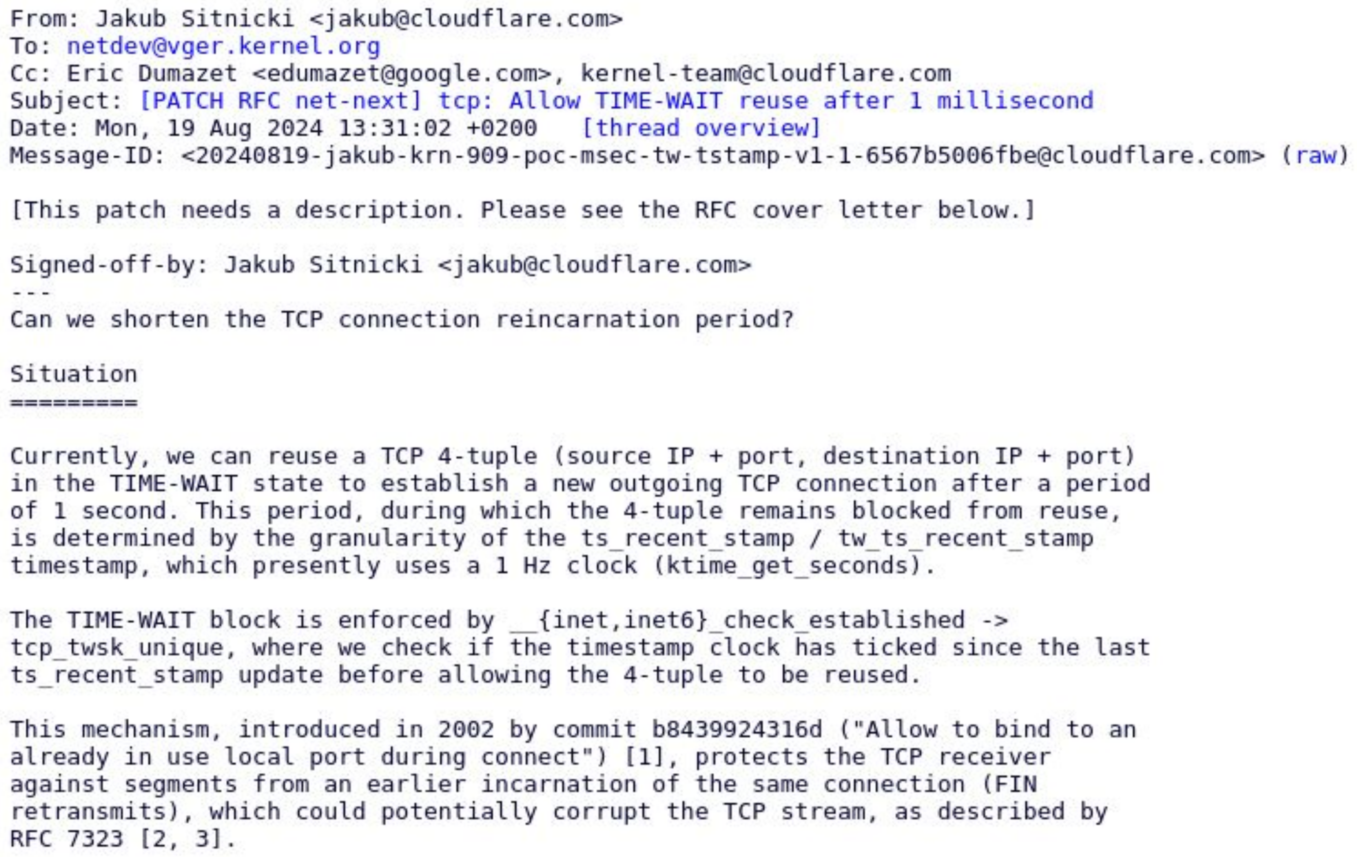
https://elixir.bootlin.com/linux/v6.10/source/net/ipv4/tcp_ipv4.c#L144

```
dig +short +tcp @8.8.8.8 example.com A
```



No.	Time	Source	Destination	Protocol	Length	Info
1	0.000	198.41.138.37	8.8.8.8	TCP	74	25933 → 53 [SYN] Seq=0 Win=65535 Len=0 MSS=1220 SACK_PERM TSval=1
2	0.001	8.8.8.8	198.41.138.37	TCP	74	53 → 25933 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1412 SACK_P
3	0.001	198.41.138.37	8.8.8.8	TCP	66	25933 → 53 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=1317784939 TSe
4	0.001	198.41.138.37	8.8.8.8	DNS	120	Standard query 0x543c A example.com OPT
5	0.003	8.8.8.8	198.41.138.37	TCP	66	53 → 25933 [ACK] Seq=1 Ack=55 Win=65536 Len=0 TSval=2653034047 TS
6	0.003	8.8.8.8	198.41.138.37	DNS	124	Standard query response 0x543c A example.com A 93.184.215.14 OPT
7	0.003	198.41.138.37	8.8.8.8	TCP	66	25933 → 53 [ACK] Seq=55 Ack=59 Win=65536 Len=0 TSval=1317784941 T
8	0.004	198.41.138.37	8.8.8.8	TCP	66	25933 → 53 [FIN, ACK] Seq=55 Ack=59 Win=65536 Len=0 TSval=1317784
9	0.005	8.8.8.8	198.41.138.37	TCP	66	53 → 25933 [FIN, ACK] Seq=59 Ack=56 Win=65536 Len=0 TSval=2653034
10	0.005	198.41.138.37	8.8.8.8	TCP	66	25933 → 53 [ACK] Seq=56 Ack=60 Win=65536 Len=0 TSval=1317784943 T

From SYN to last ACK - few milliseconds
TIME-WAIT reuse after - (up to) a second



```
From: Jakub Sitnicki <jakub@cloudflare.com>
To: netdev@vger.kernel.org
Cc: Eric Dumazet <edumazet@google.com>, kernel-team@cloudflare.com
Subject: [PATCH RFC net-next] tcp: Allow TIME-WAIT reuse after 1 millisecond
Date: Mon, 19 Aug 2024 13:31:02 +0200   [thread overview]
Message-ID: <20240819-jakub-krn-909-poc-msec-tw-tstamp-v1-1-6567b5006fbe@cloudflare.com> (raw)

[This patch needs a description. Please see the RFC cover letter below.]

Signed-off-by: Jakub Sitnicki <jakub@cloudflare.com>
---
Can we shorten the TCP connection reincarnation period?

Situation
=========

Currently, we can reuse a TCP 4-tuple (source IP + port, destination IP + port)
in the TIME-WAIT state to establish a new outgoing TCP connection after a period
of 1 second. This period, during which the 4-tuple remains blocked from reuse,
is determined by the granularity of the ts_recent_stamp / tw_ts_recent_stamp
timestamp, which presently uses a 1 Hz clock (ktime_get_seconds).

The TIME-WAIT block is enforced by __{inet,inet6}_check_established ->
tcp_twsk_unique, where we check if the timestamp clock has ticked since the last
ts_recent_stamp update before allowing the 4-tuple to be reused.

This mechanism, introduced in 2002 by commit b8439924316d ("Allow to bind to an
already in use local port during connect") [1], protects the TCP receiver
against segments from an earlier incarnation of the same connection (FIN
retransmits), which could potentially corrupt the TCP stream, as described by
RFC 7323 [2, 3].
```

https://lore.kernel.org/all/20240819-jakub-krn-909-poc-msec-tw-tstamp-v1-1-6567b5006fbe@cloudflare.com/

Initial feedback

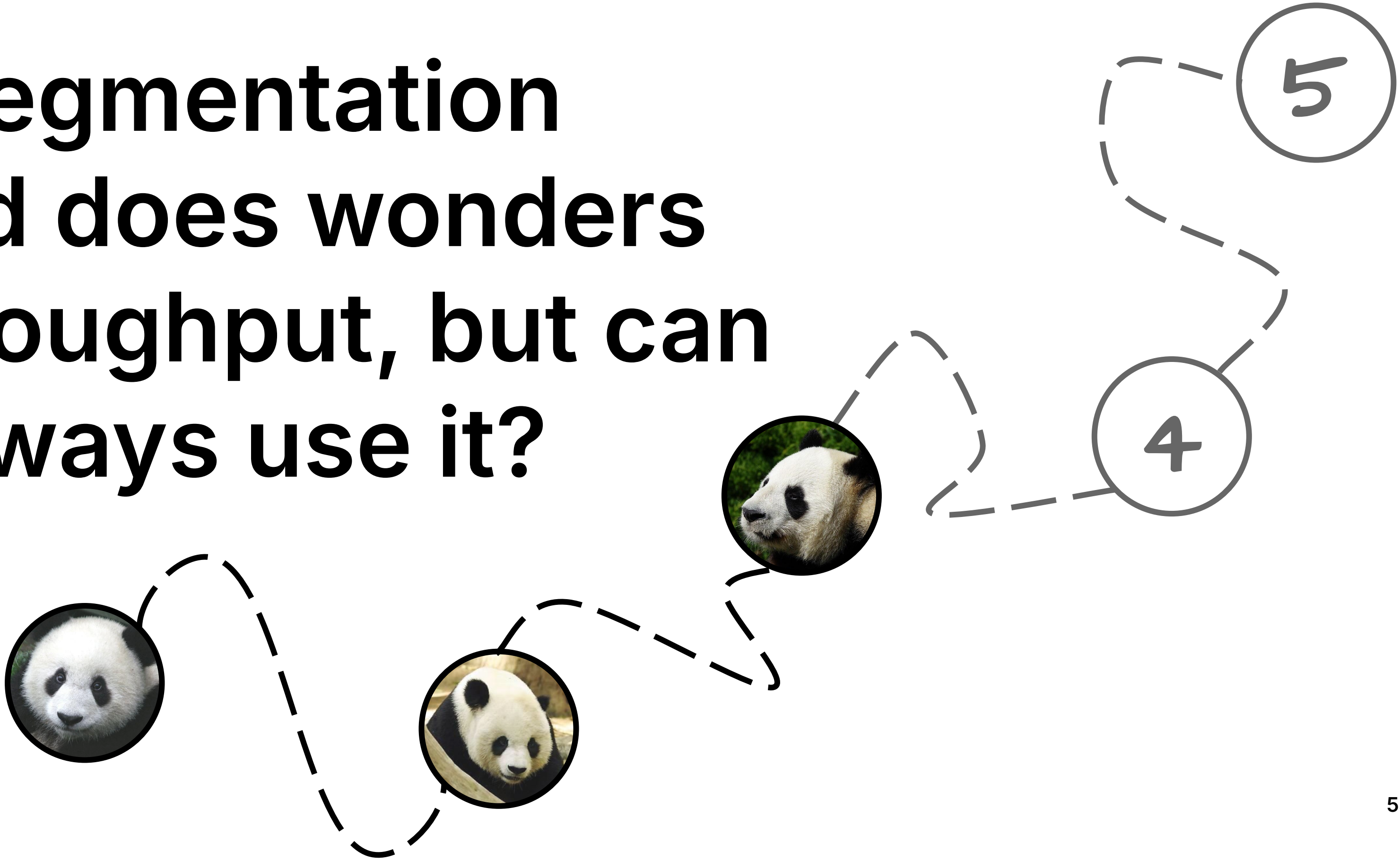
* don't use jiffies for timestamps (where possible)
* account for RTT in reuse threshold
* watch out for integer roundoff
* make it configurable

🙏 Eric Dumazet

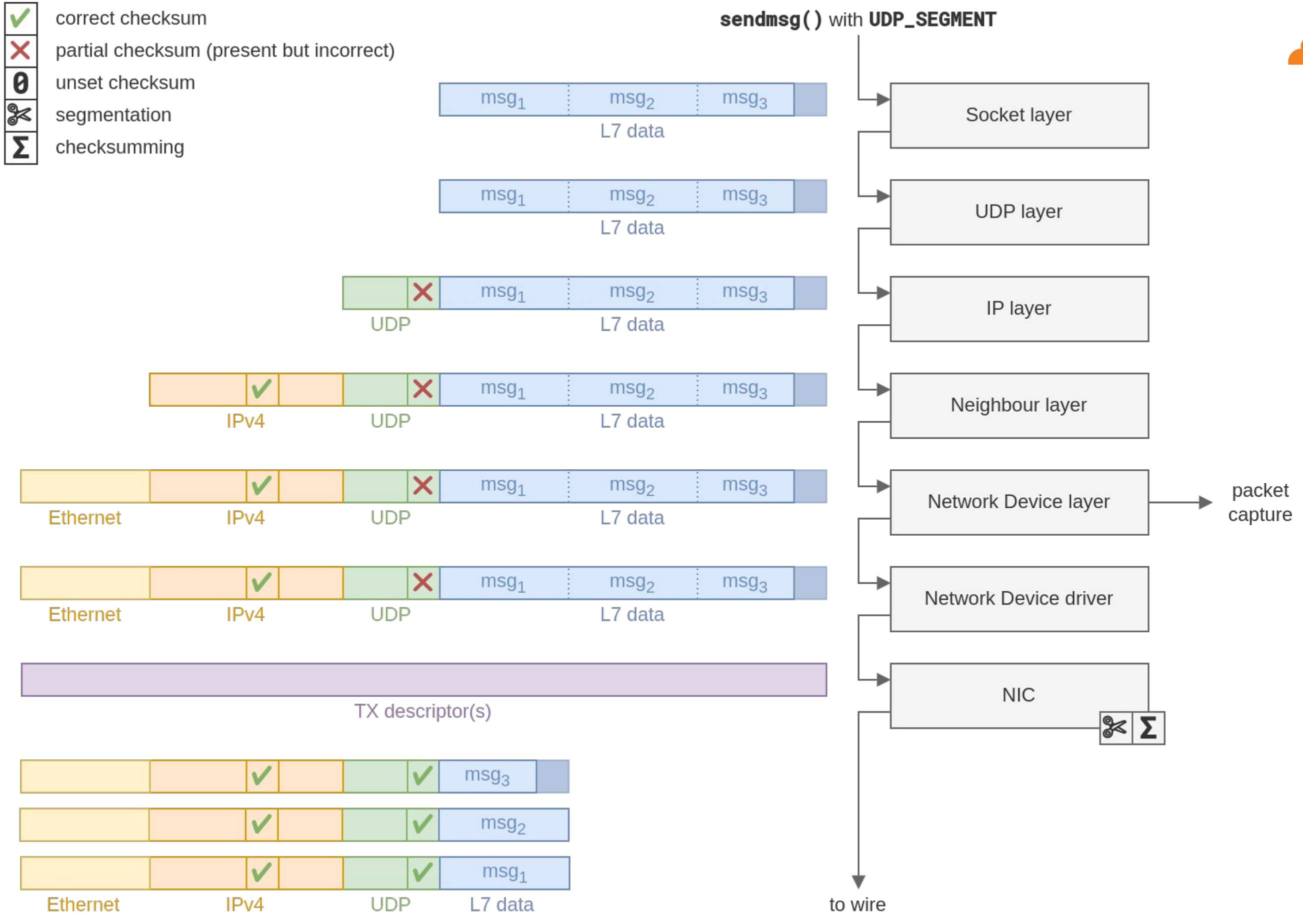
patch series TBC

❸

UDP segmentation offload does wonders for throughput, but can you always use it?



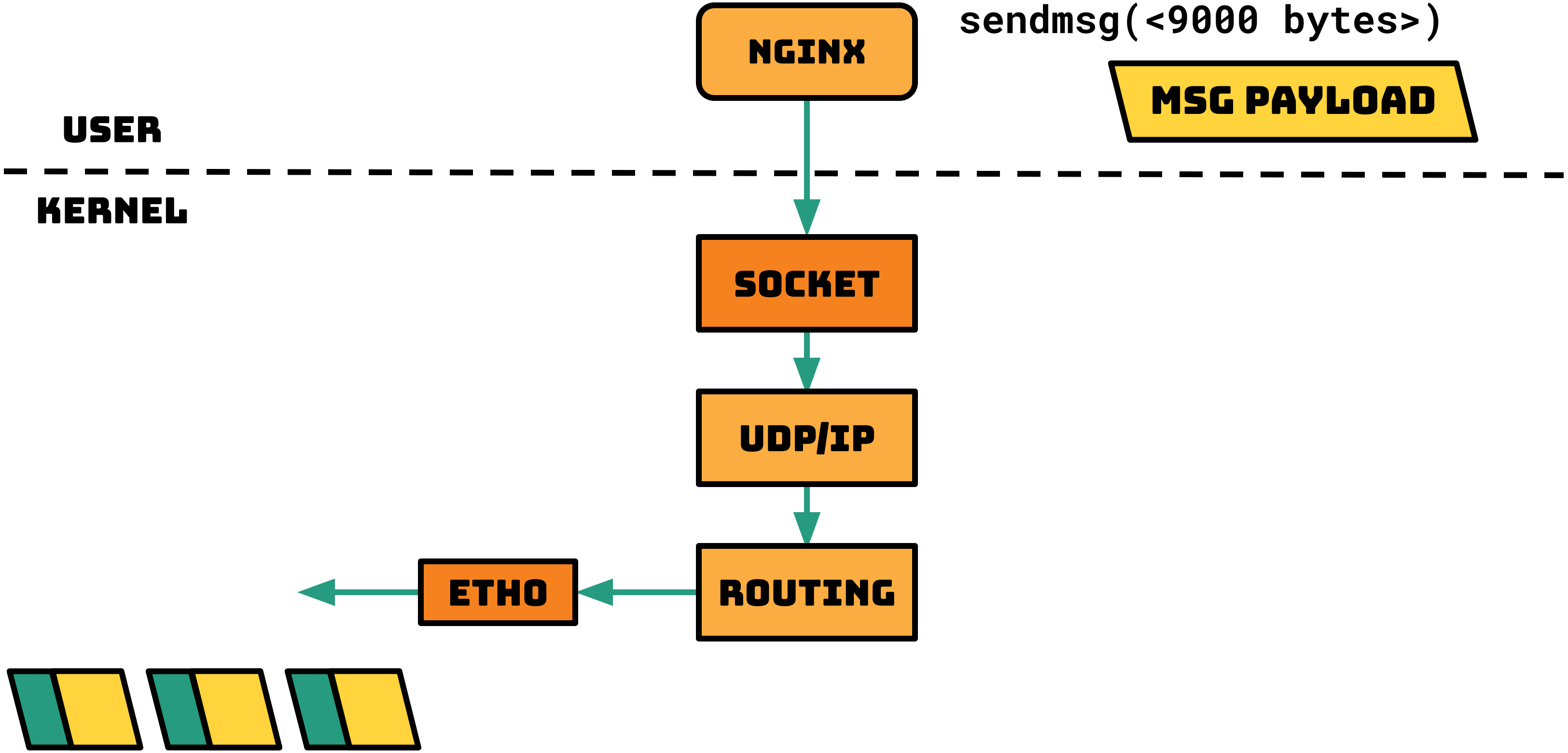
- ✓ correct checksum
- ✗ partial checksum (present but incorrect)
- 0 unset checksum
- ✂ segmentation
- Σ checksumming



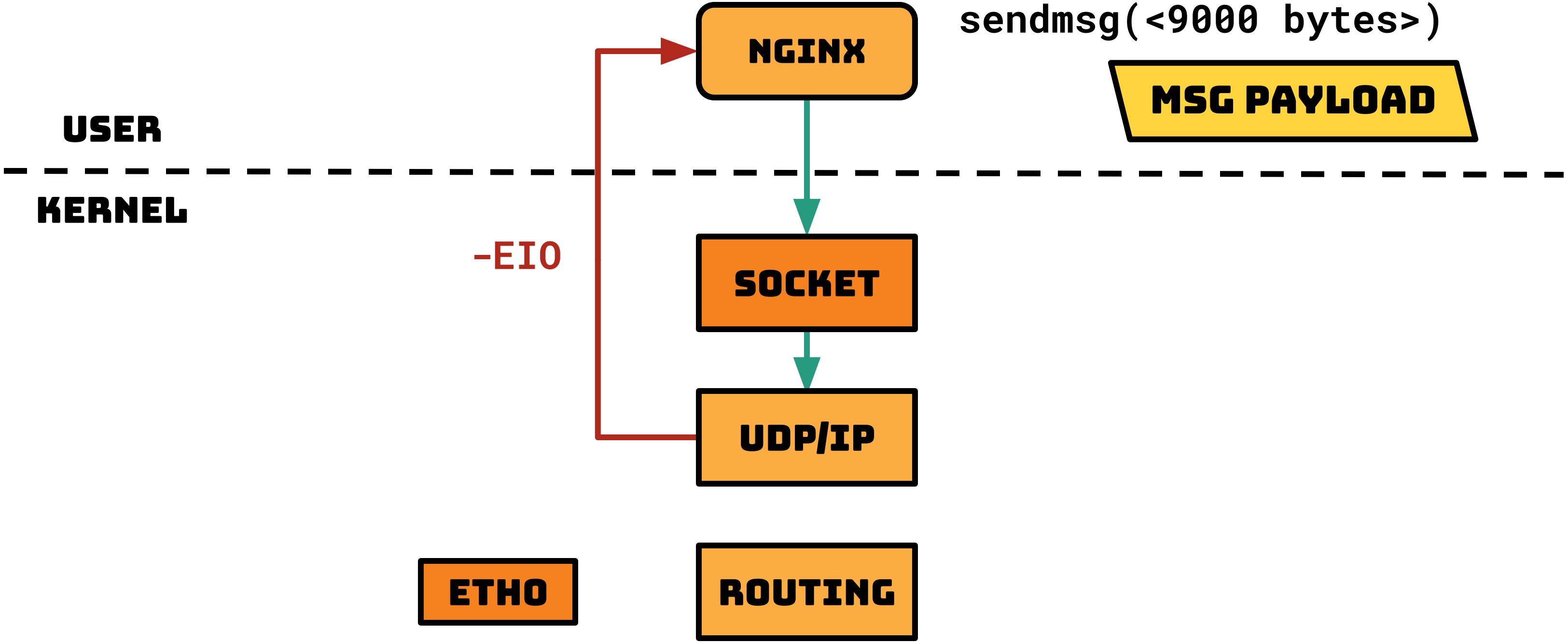
Hardware UDP Segmentation Offload

`tx-udp-segmentation=on`, `tx-checksum-ip-generic=on`

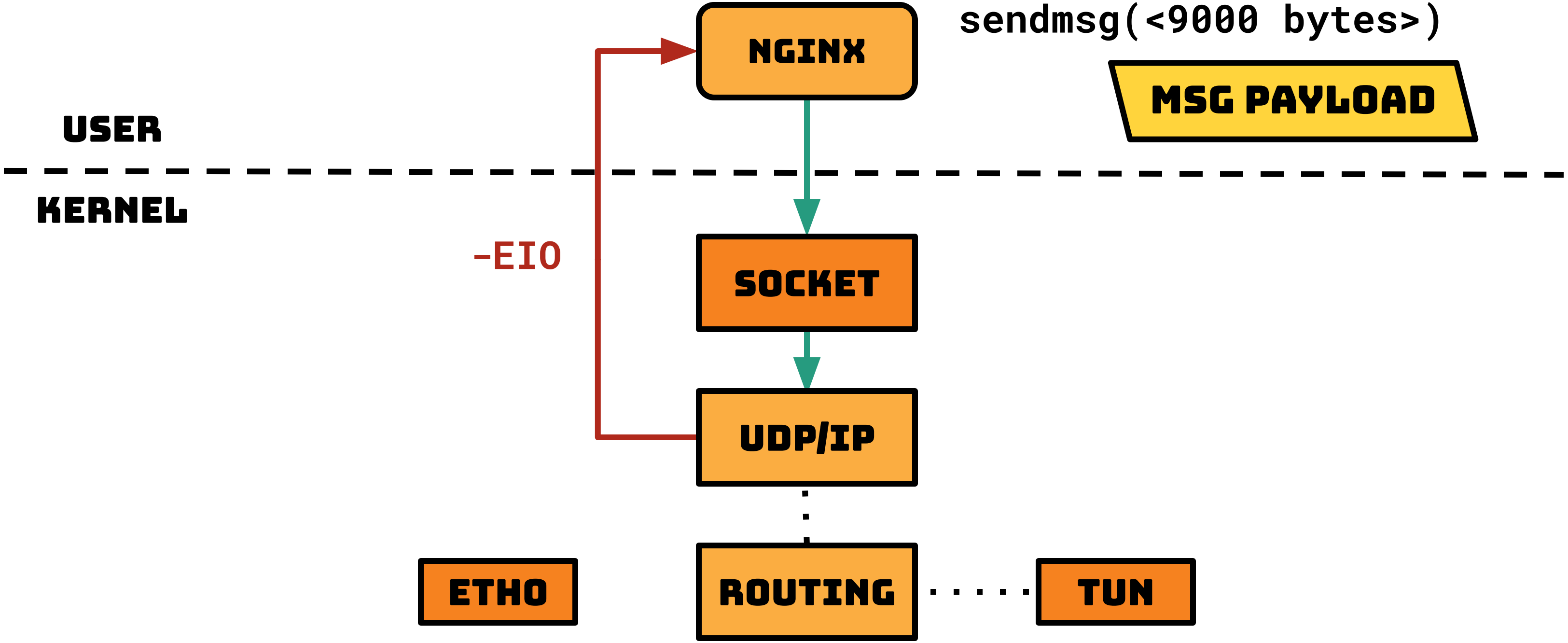
```
setsockopt(…, UDP_SEGMENT, 1350)
sendmsg(<9000 bytes>)
```



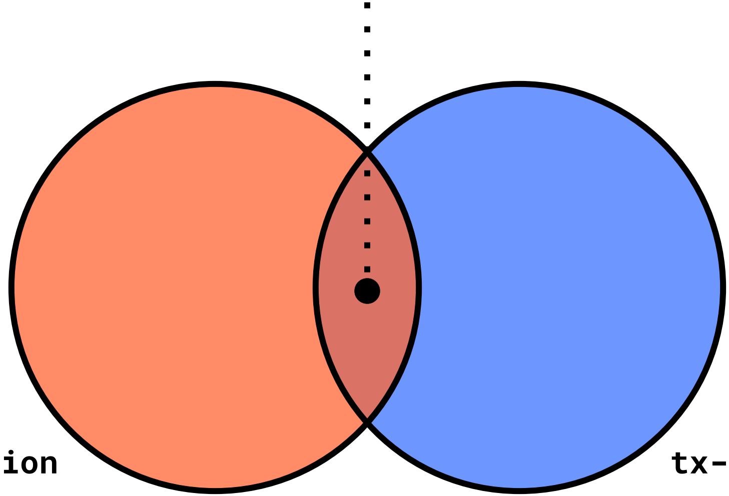
```
setsockopt(…, UDP_SEGMENT, 1350)
sendmsg(<9000 bytes>)
```



```
setsockopt(…, UDP_SEGMENT, 1350)
sendmsg(<9000 bytes>)
```

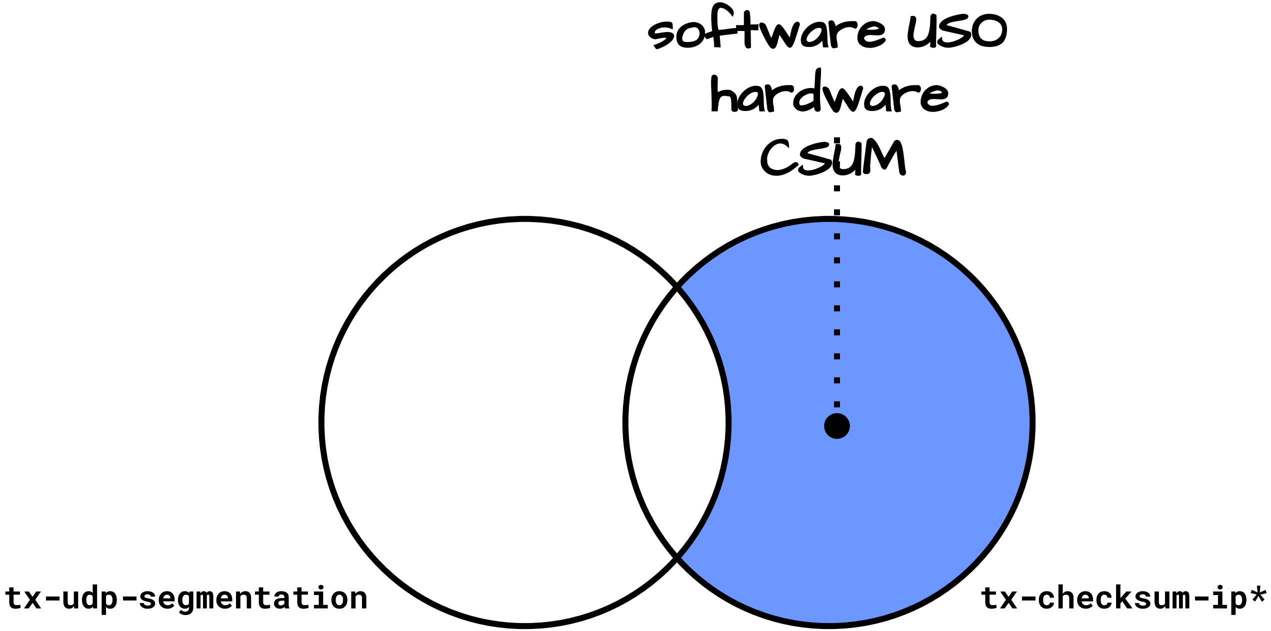


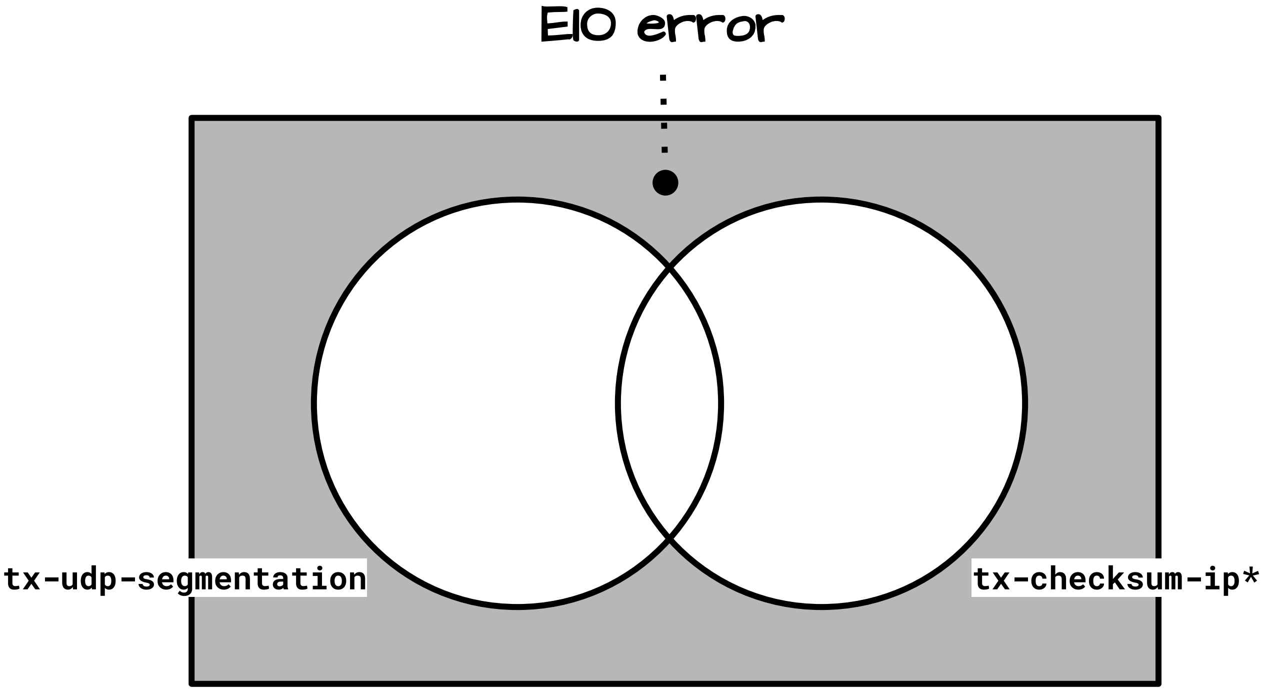
hardware USO & CSUM



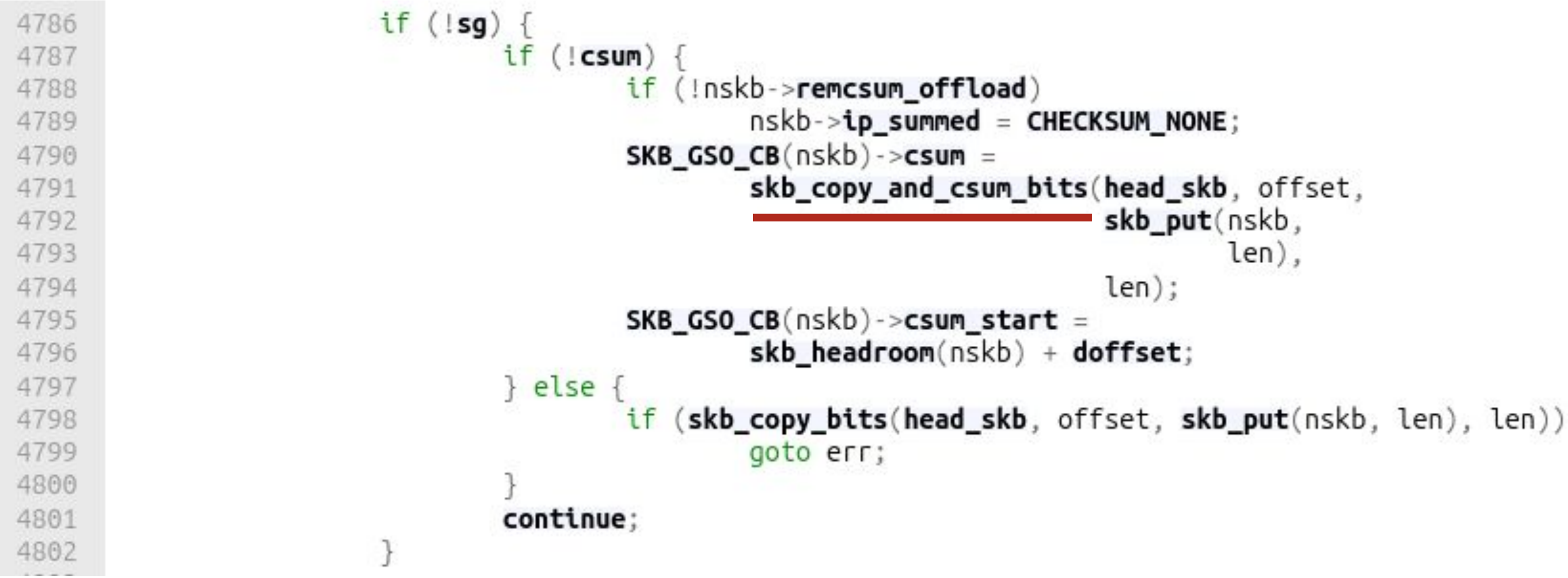
tx-udp-segmentation

tx-checksum-ip*


software USO
hardware
CSUM
tx-udp-segmentation
tx-checksum-ip*


EIO error
tx-udp-segmentation
tx-checksum-ip*

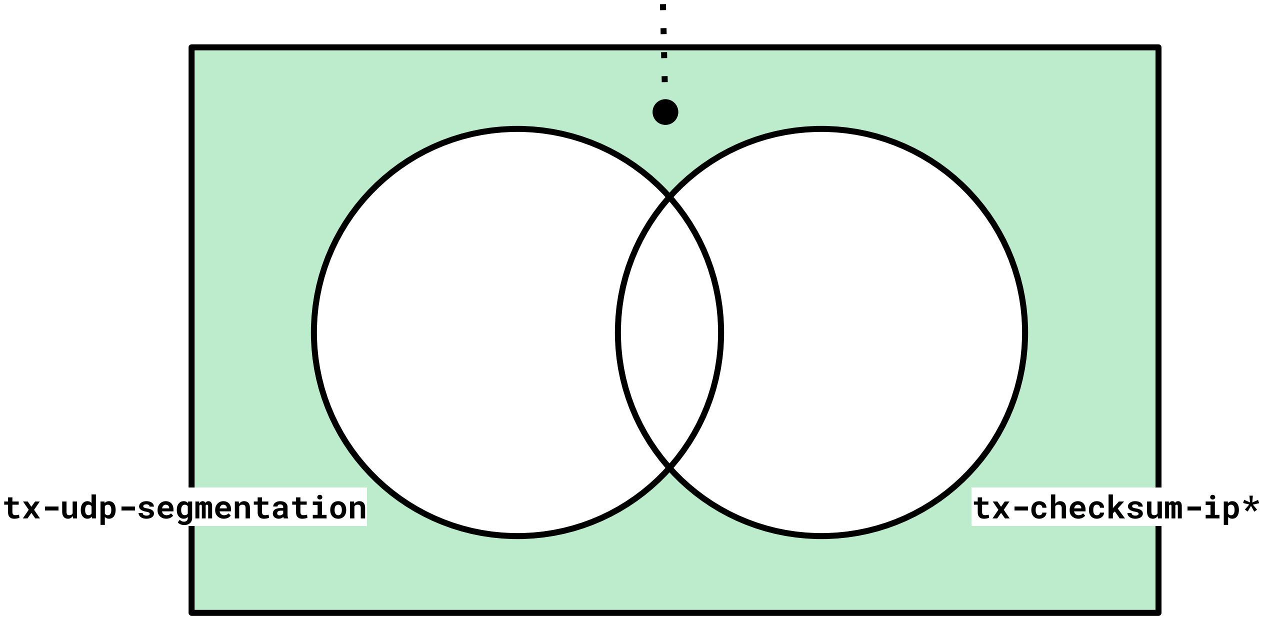
skb_segment()

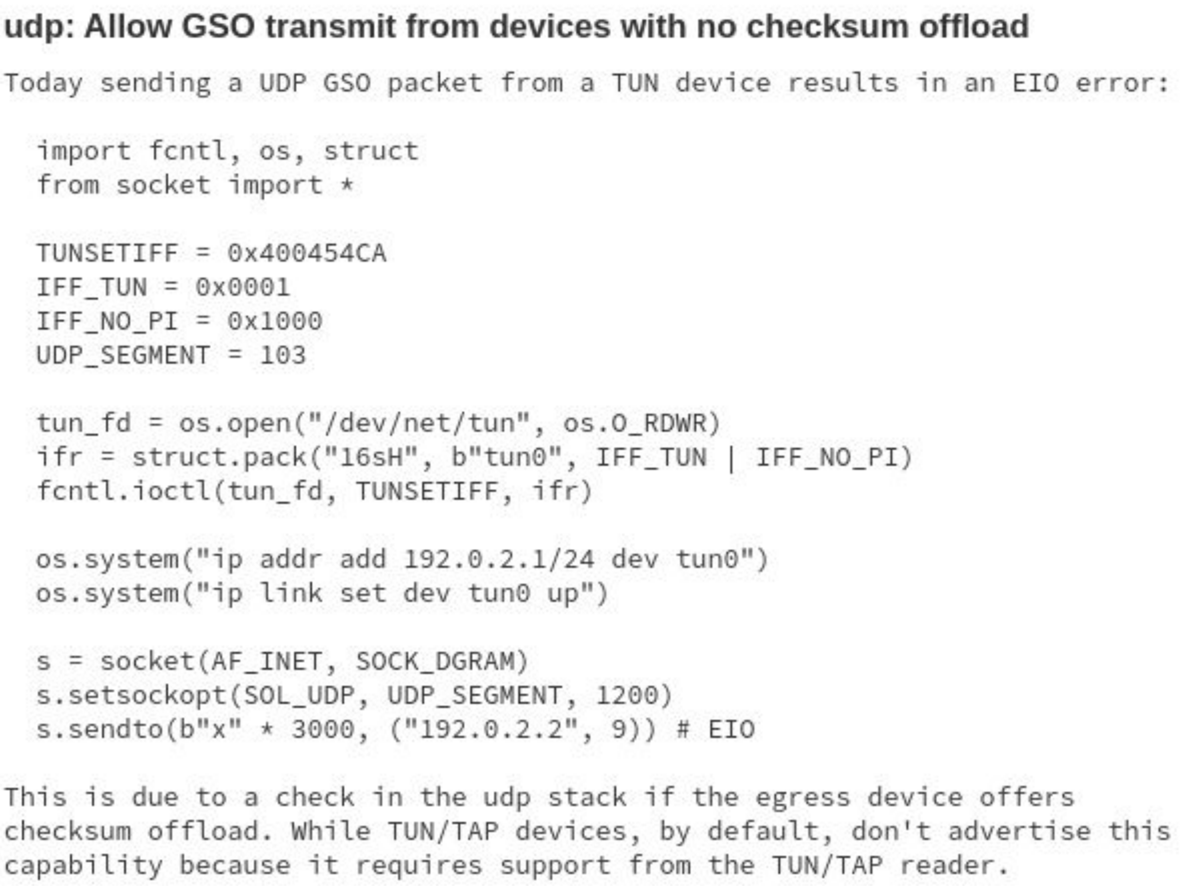


```
if (!sg) {
        if (!csum) {
                if (!nskb->remcsum_offload)
                        nskb->ip_summed = CHECKSUM_NONE;
                SKB_GSO_CB(nskb)->csum =
                        skb_copy_and_csum_bits(head_skb, offset,
                                               skb_put(nskb,
                                                       len),
                                               len);
                SKB_GSO_CB(nskb)->csum_start =
                        skb_headroom(nskb) + doffset;
        } else {
                if (skb_copy_bits(head_skb, offset, skb_put(nskb, len), len))
                        goto err;
        }
        continue;
}
```

https://elixir.bootlin.com/linux/v6.10/source/net/core/skbuff.c#L4791

~~EIO error~~
software USO & CSUM





udp: Allow GSO transmit from devices with no checksum offload

Today sending a UDP GSO packet from a TUN device results in an EIO error:

```
  import fcntl, os, struct
  from socket import *

  TUNSETIFF = 0x400454CA
  IFF_TUN = 0x0001
  IFF_NO_PI = 0x1000
  UDP_SEGMENT = 103

  tun_fd = os.open("/dev/net/tun", os.O_RDWR)
  ifr = struct.pack("16sH", b"tun0", IFF_TUN | IFF_NO_PI)
  fcntl.ioctl(tun_fd, TUNSETIFF, ifr)

  os.system("ip addr add 192.0.2.1/24 dev tun0")
  os.system("ip link set dev tun0 up")

  s = socket(AF_INET, SOCK_DGRAM)
  s.setsockopt(SOL_UDP, UDP_SEGMENT, 1200)
  s.sendto(b"x" * 3000, ("192.0.2.2", 9)) # EIO
```

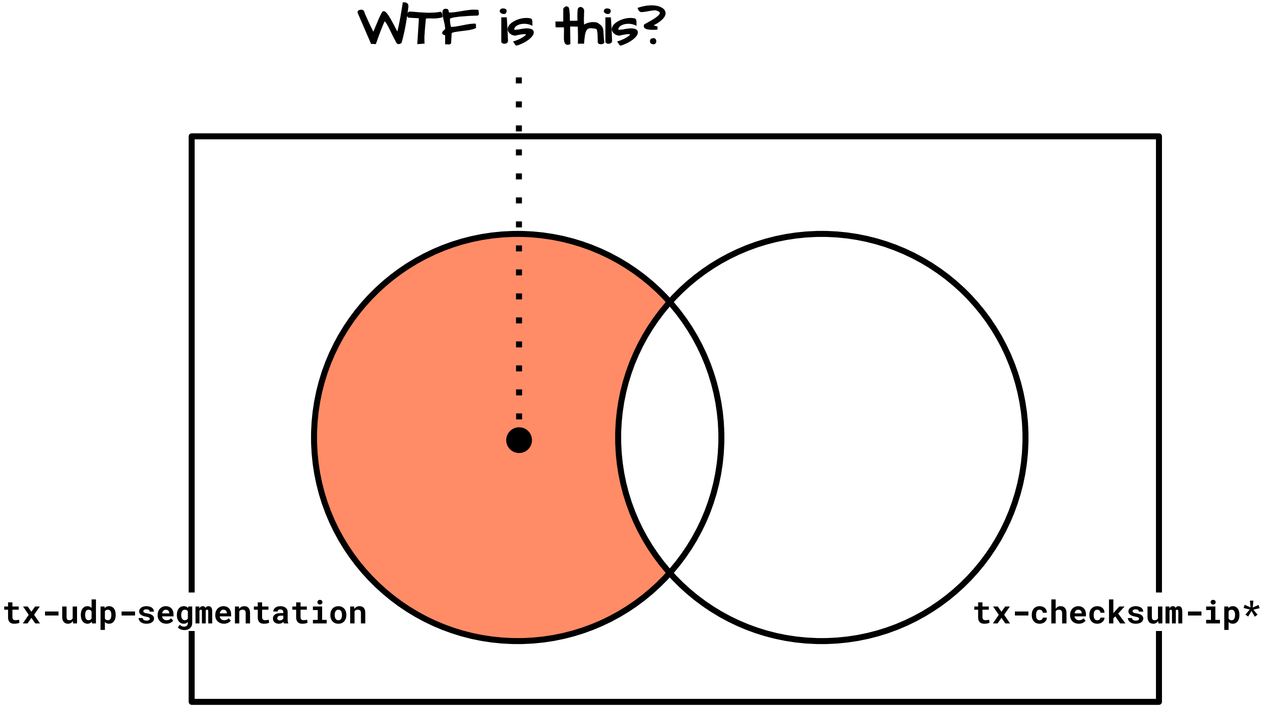
This is due to a check in the udp stack if the egress device offers checksum offload. While TUN/TAP devices, by default, don't advertise this capability because it requires support from the TUN/TAP reader.

available in v6.11

https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/commit/?id=10154dbded6d6a2fecaebdfda206609de0f121a9

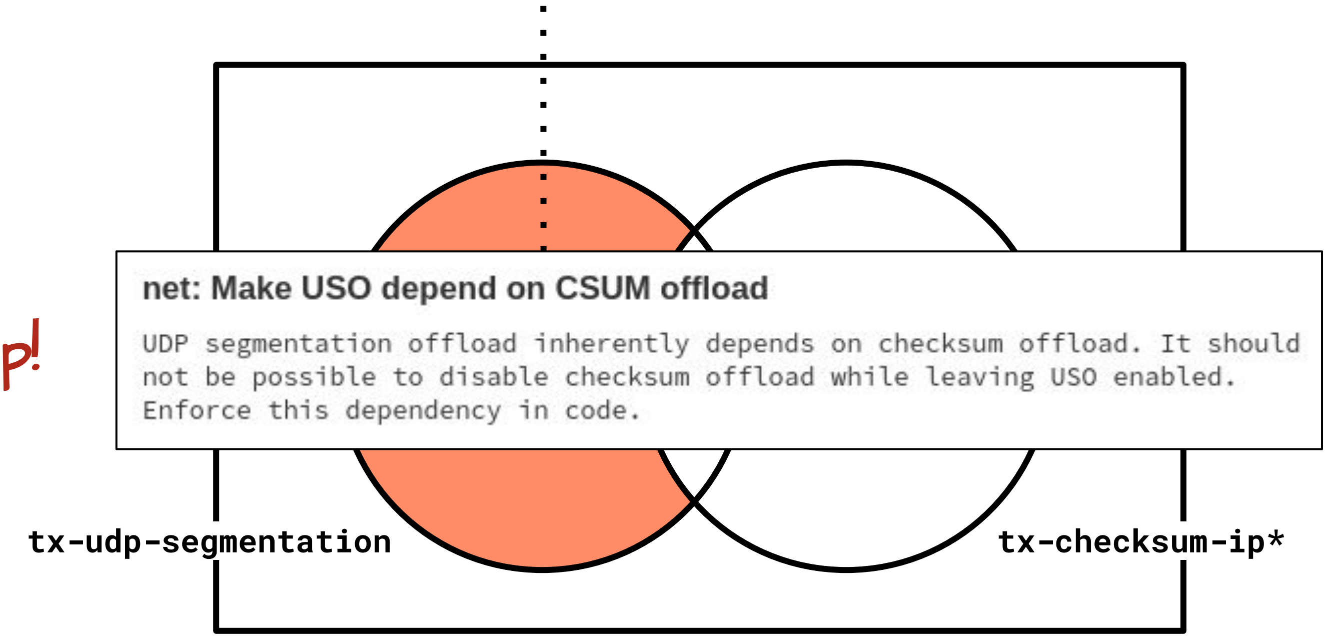
“there is always a but in this imperfect world!”

— Anne Brontë, The Tenant of Wildfell Hall


WTF is this?
tx-udp-segmentation
tx-checksum-ip*

WTF is this?

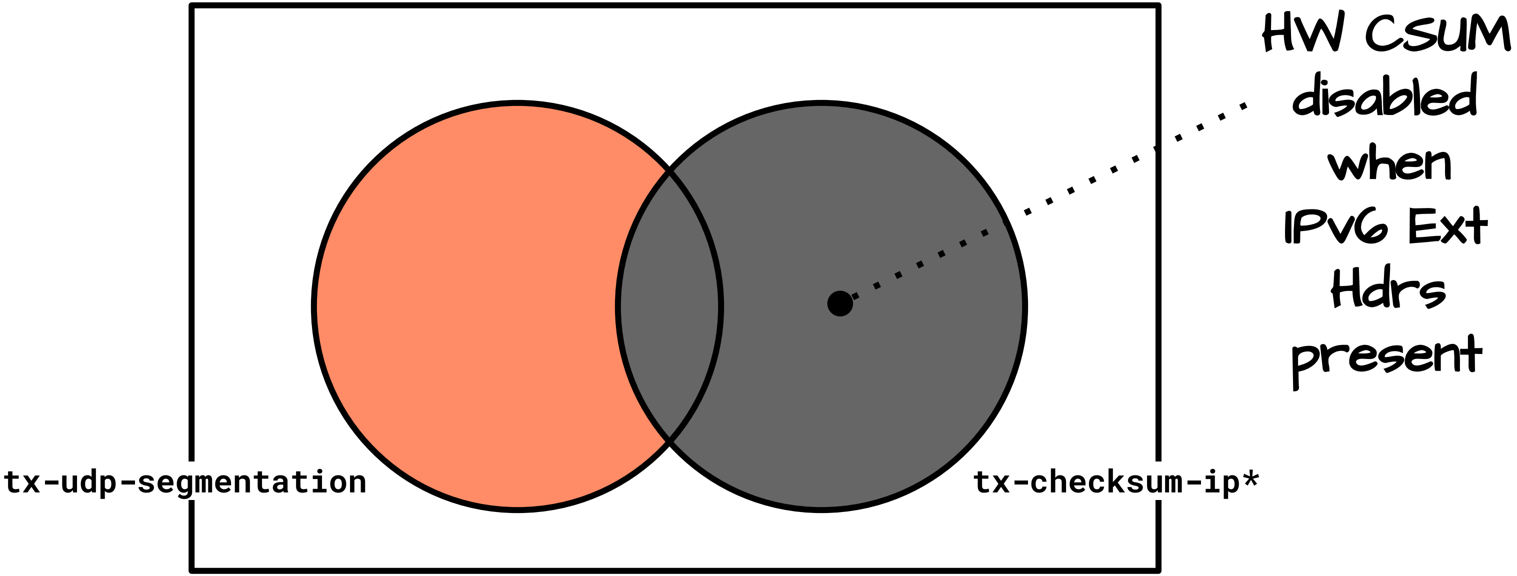
fixup!



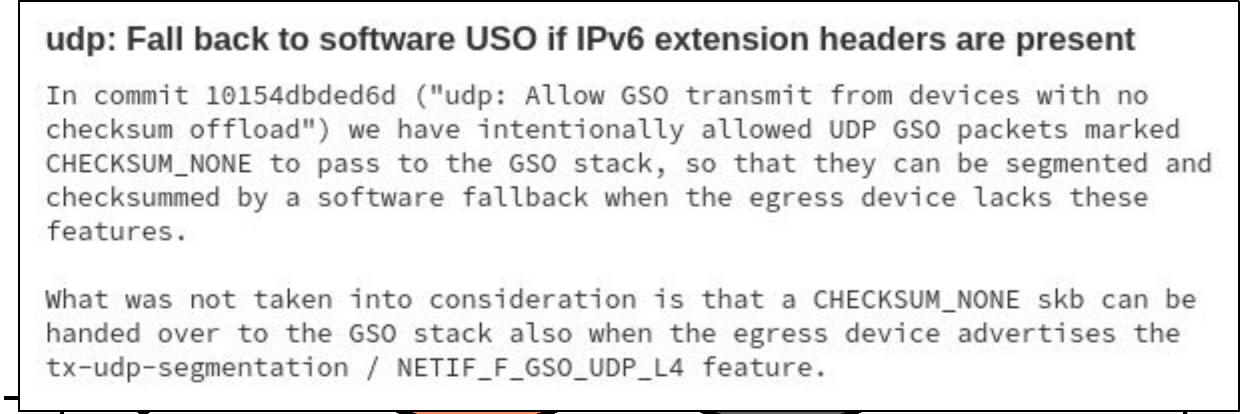
net: Make USO depend on CSUM offload

```
UDP segmentation offload inherently depends on checksum offload. It should
not be possible to disable checksum offload while leaving USO enabled.
Enforce this dependency in code.
```

https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/commit/?id=2b2bc3bab158b7e036508742b16cd8a3c2f59a12


HW CSUM disabled when IPv6 Ext Hdrs present
tx-udp-segmentation
tx-checksum-ip*

fixup!



udp: Fall back to software USO if IPv6 extension headers are present

In commit 10154dbded6d ("udp: Allow GSO transmit from devices with no checksum offload") we have intentionally allowed UDP GSO packets marked CHECKSUM_NONE to pass to the GSO stack, so that they can be segmented and checksummed by a software fallback when the egress device lacks these features.

What was not taken into consideration is that a CHECKSUM_NONE skb can be handed over to the GSO stack also when the egress device advertises the tx-udp-segmentation / NETIF_F_GSO_UDP_L4 feature.

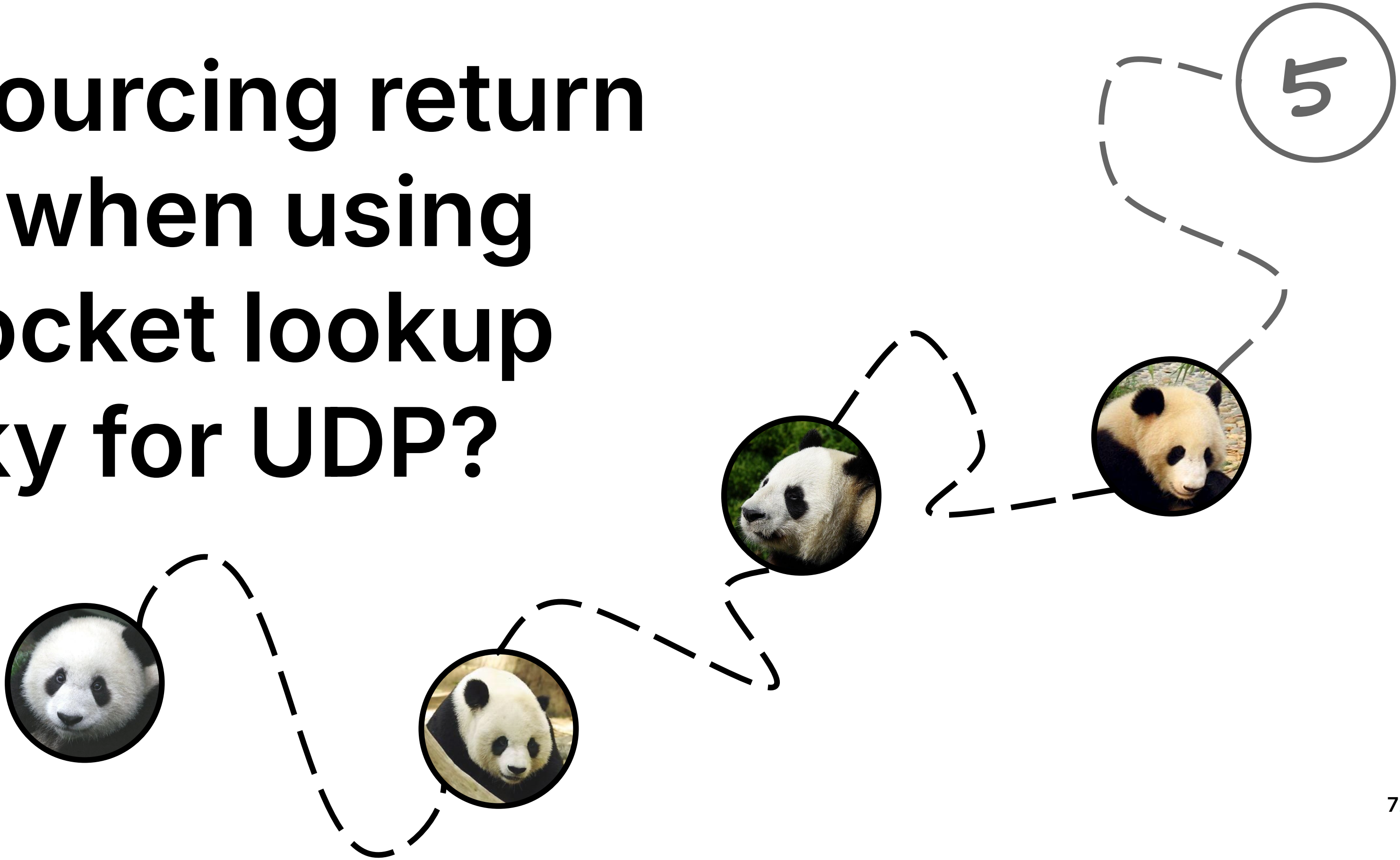
tx-

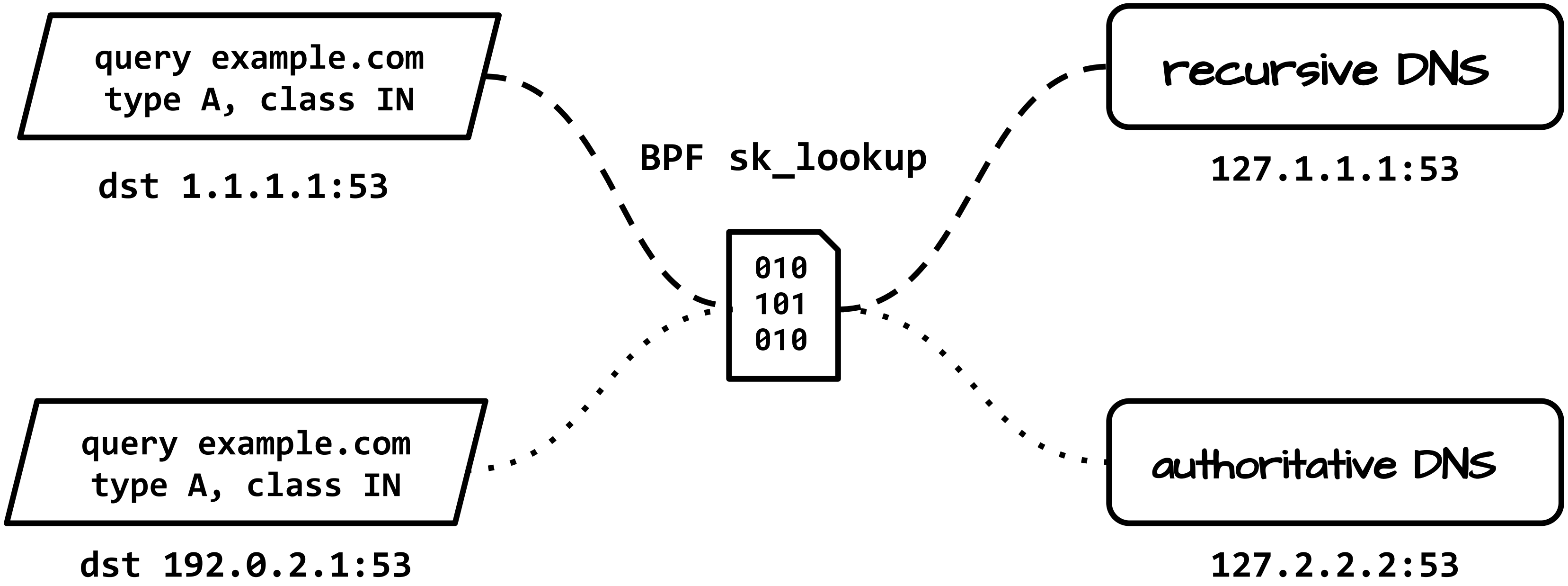
HW CSUM disabled when IPv6 Ext Hdrs present

🙏 Willem de Bruijn

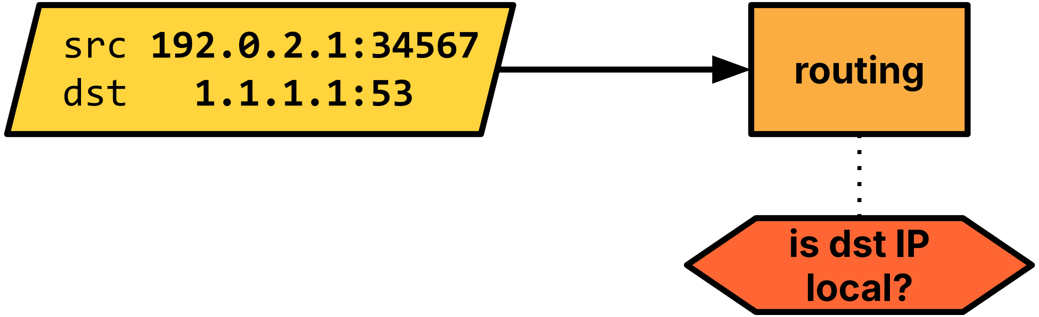
❹

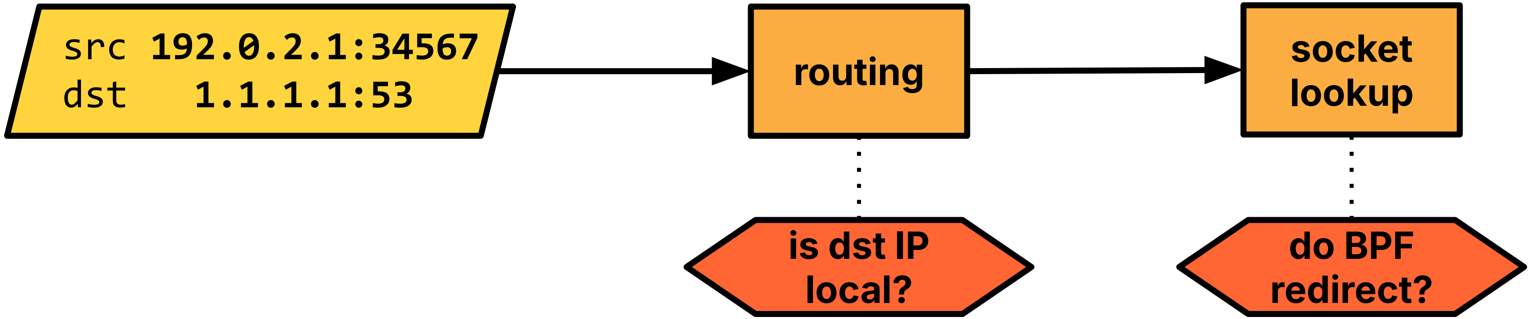
Why sourcing return traffic when using BPF socket lookup is tricky for UDP?



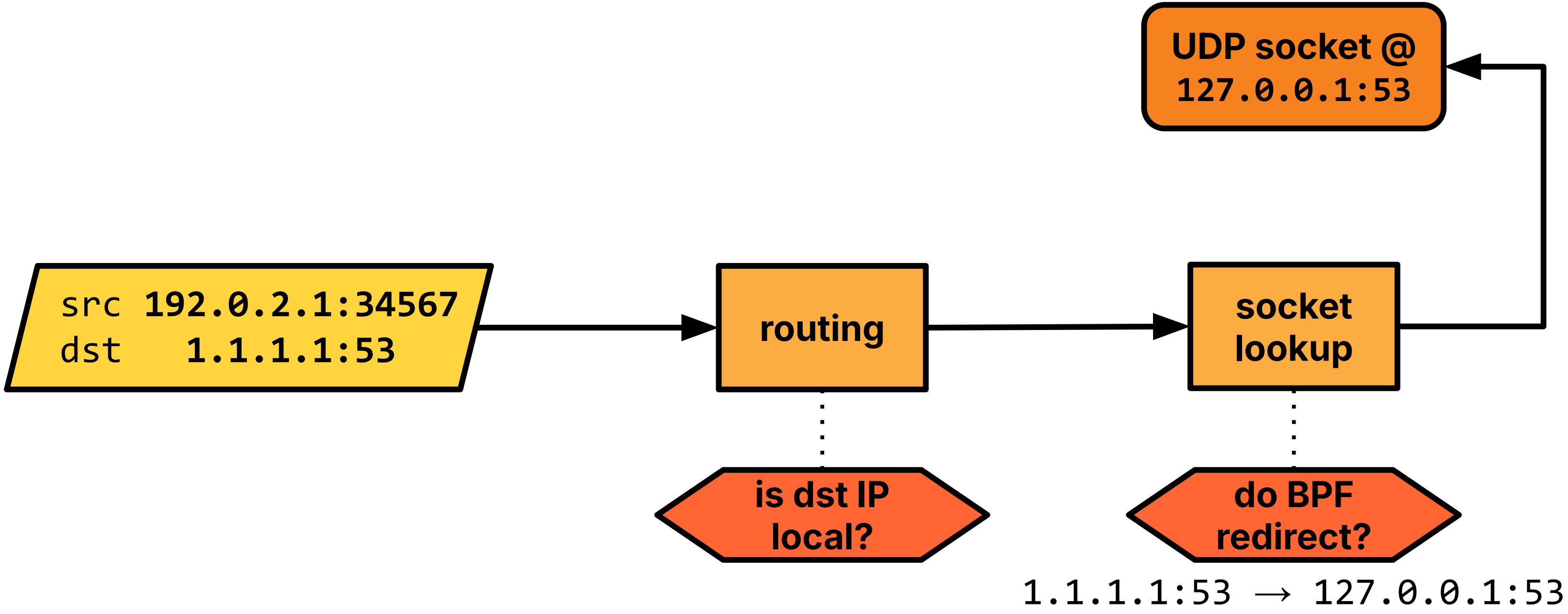

query example.com
type A, class IN
dst 1.1.1.1:53
query example.com
type A, class IN
dst 192.0.2.1:53
BPF sk_lookup
010
101
010
recursive DNS
127.1.1.1:53
authoritative DNS
127.2.2.2:53

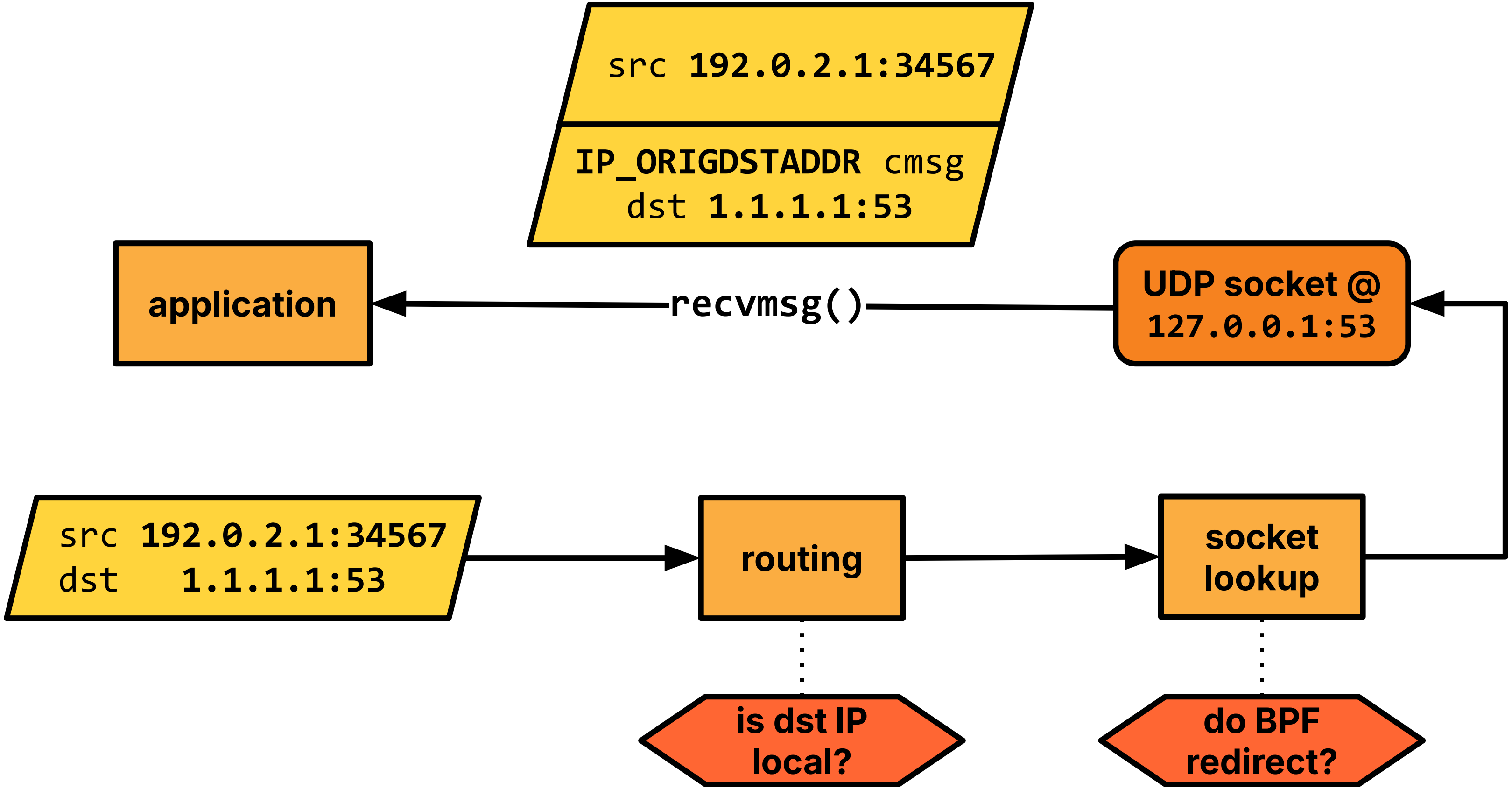
```
src 192.0.2.1:34567
dst   1.1.1.1:53
```


src 192.0.2.1:34567
dst 1.1.1.1:53
routing
is dst IP local?

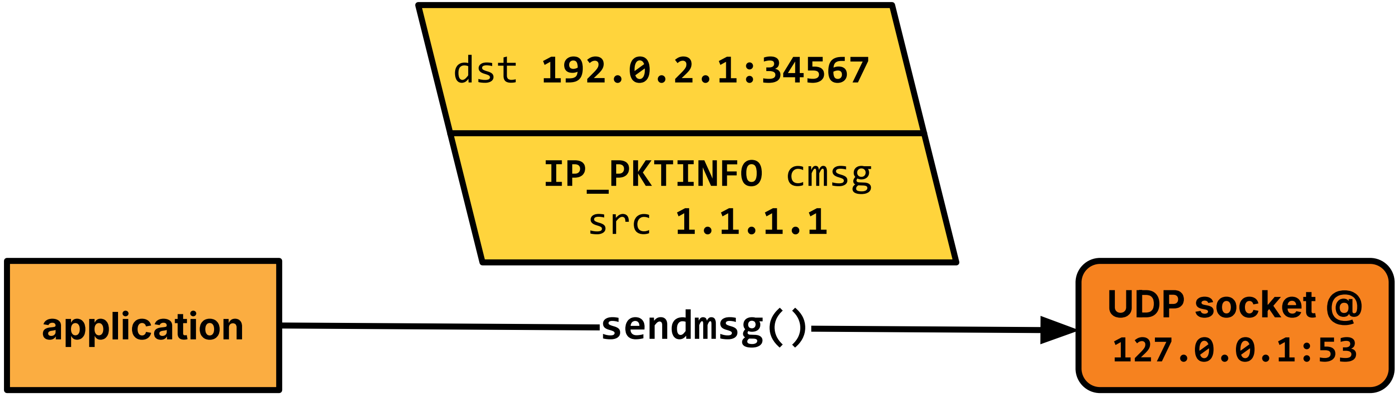


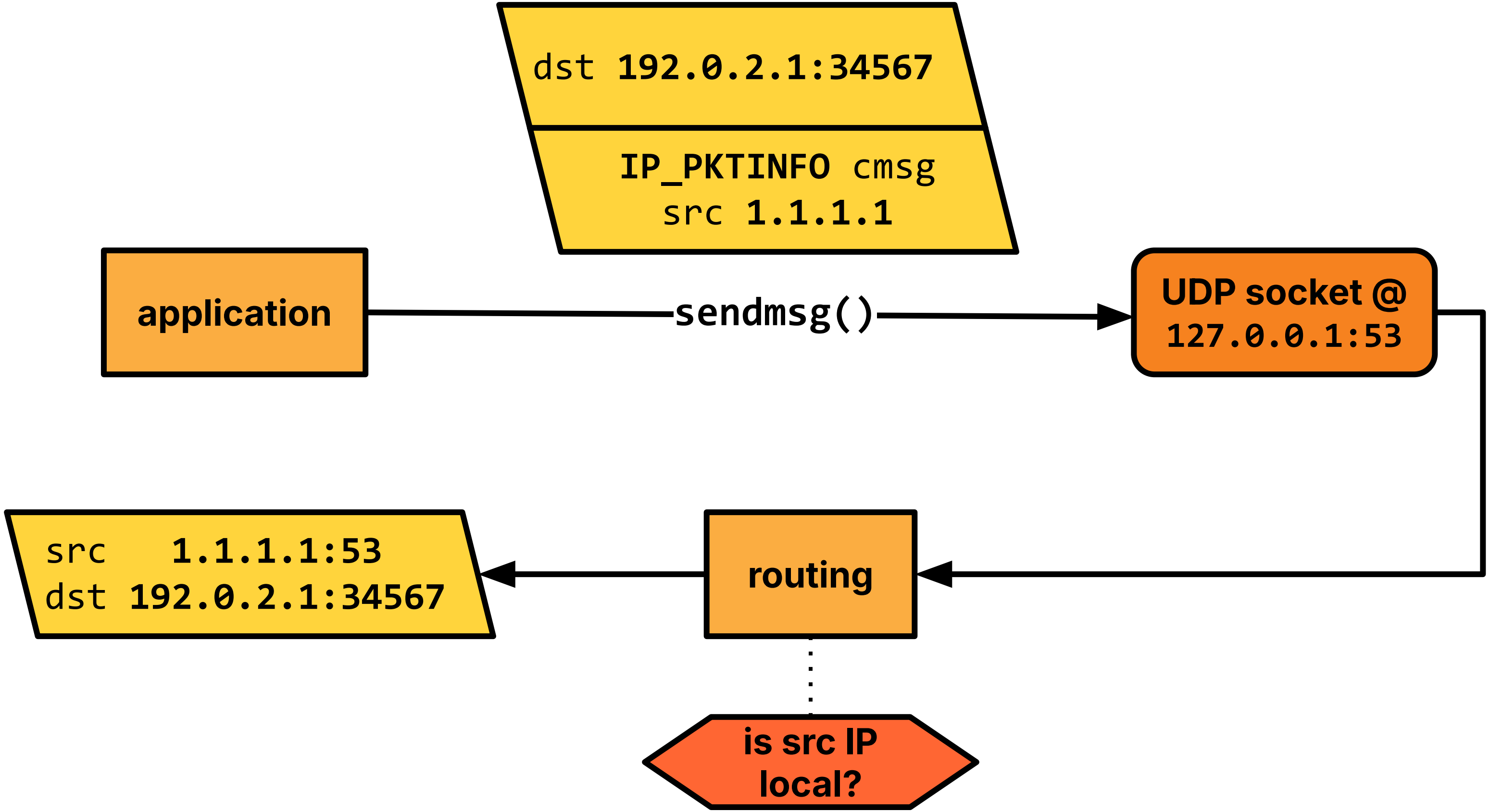
src 192.0.2.1:34567
dst 1.1.1.1:53
routing
socket lookup
is dst IP local?
do BPF redirect?

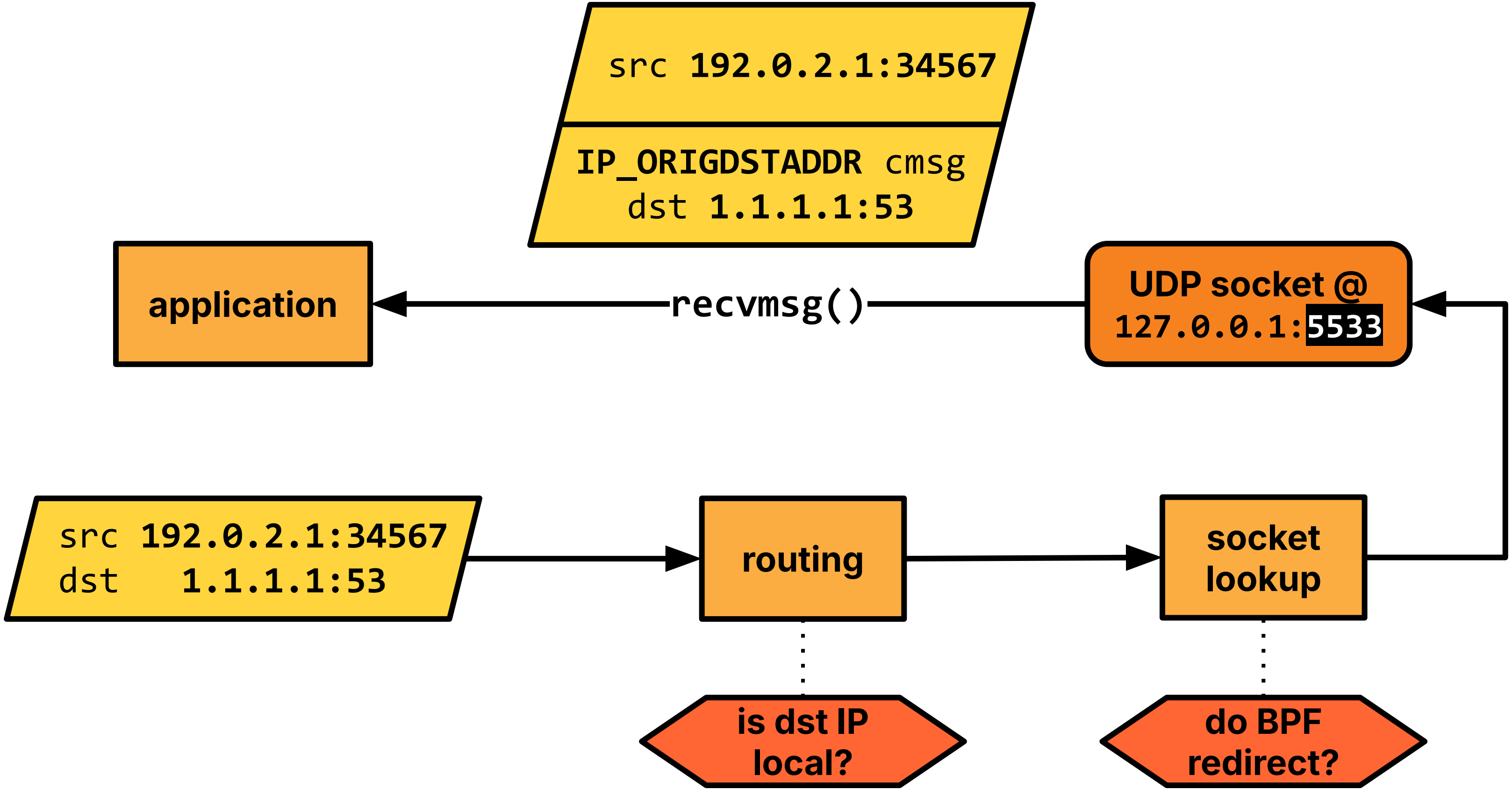

src 192.0.2.1:34567
dst 1.1.1.1:53
routing
is dst IP local?
socket lookup
do BPF redirect?
1.1.1.1:53 → 127.0.0.1:53
UDP socket @ 127.0.0.1:53



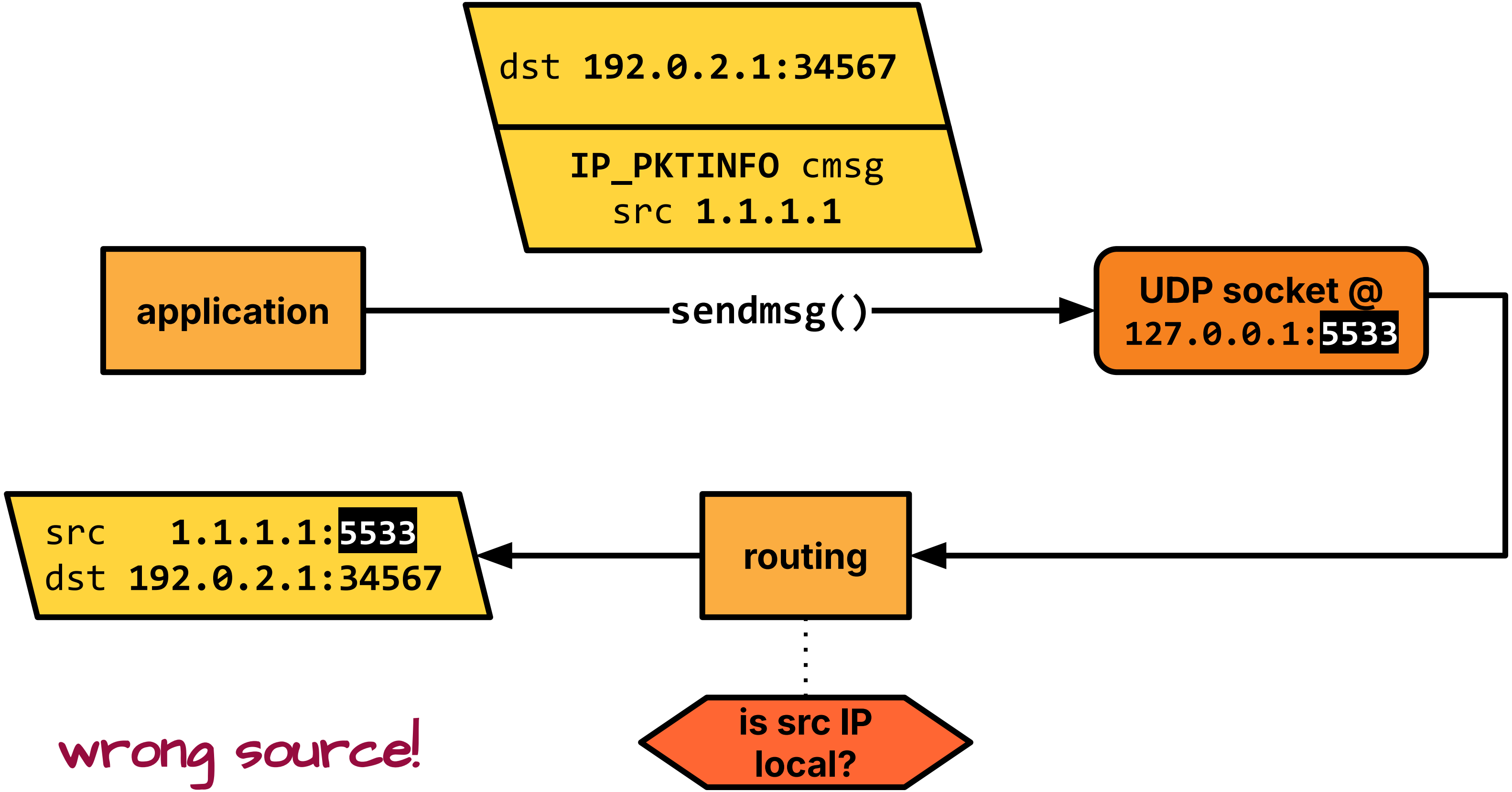
1.1.1.1:53 → 127.0.0.1:53


dst 192.0.2.1:34567
IP_PKTINFO cmsg
src 1.1.1.1
application
sendmsg()
UDP socket @
127.0.0.1:53

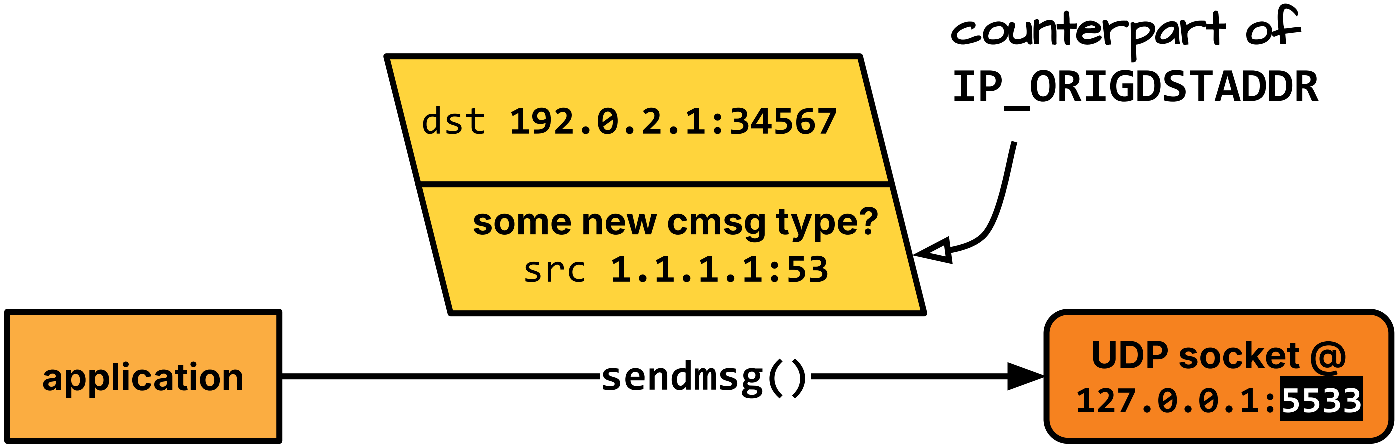

dst 192.0.2.1:34567
IP_PKTINFO cmsg
src 1.1.1.1
application
sendmsg()
UDP socket @
127.0.0.1:53
src 1.1.1.1:53
dst 192.0.2.1:34567
routing
is src IP
local?



1.1.1.1:53 → 127.0.0.1:5533



wrong source!



counterpart of IP_ORIGDSTADDR

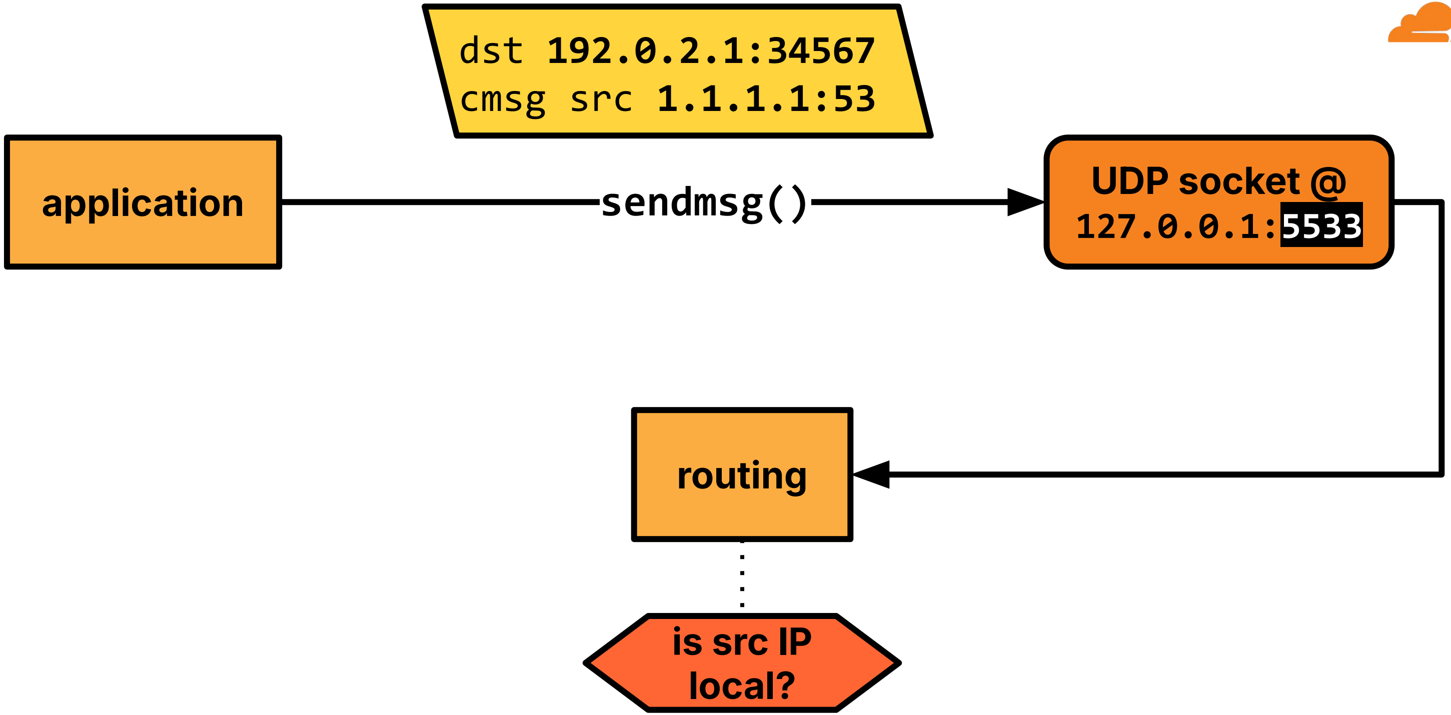
dst 192.0.2.1:34567

some new cmsg type?
src 1.1.1.1:53

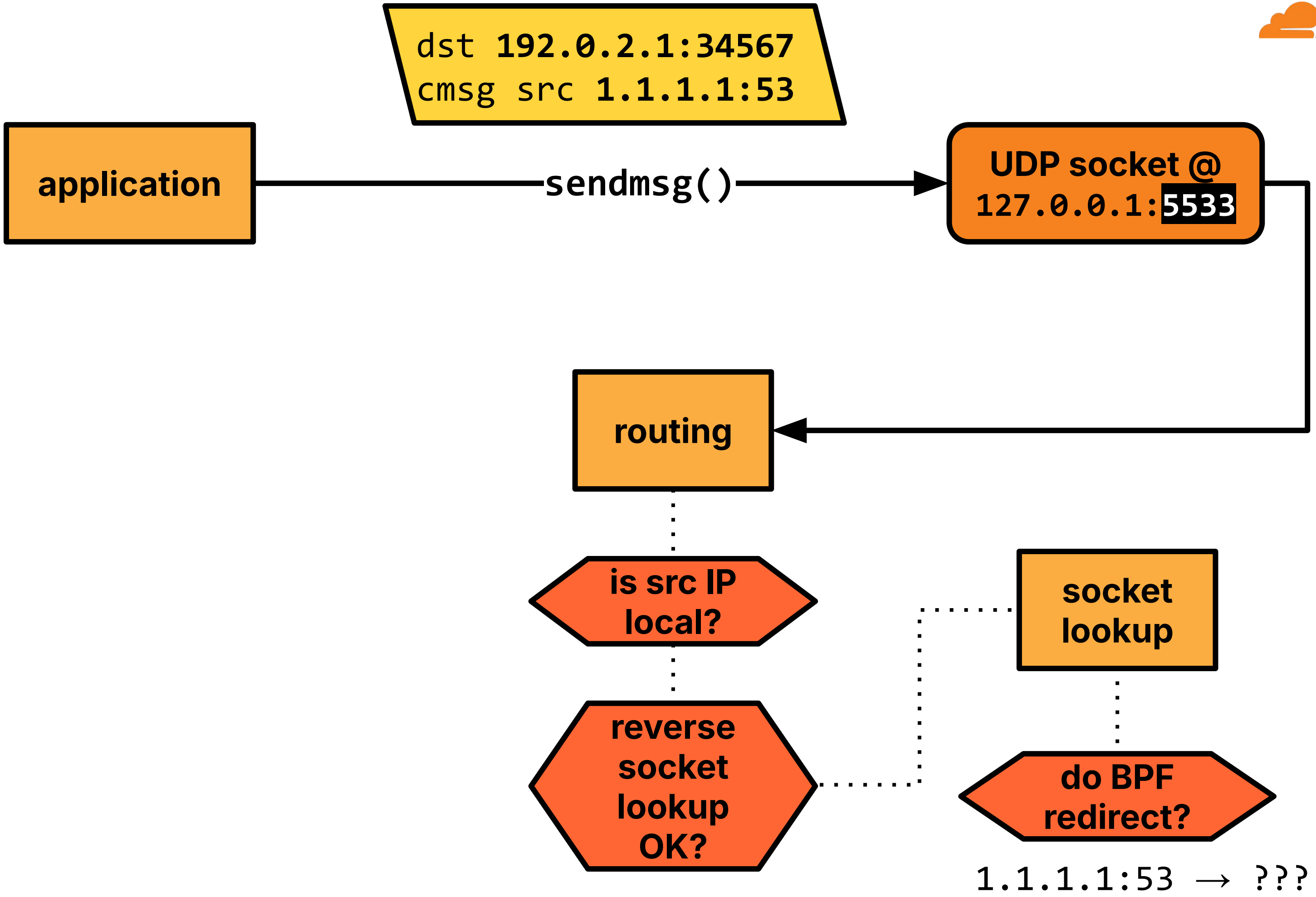
application

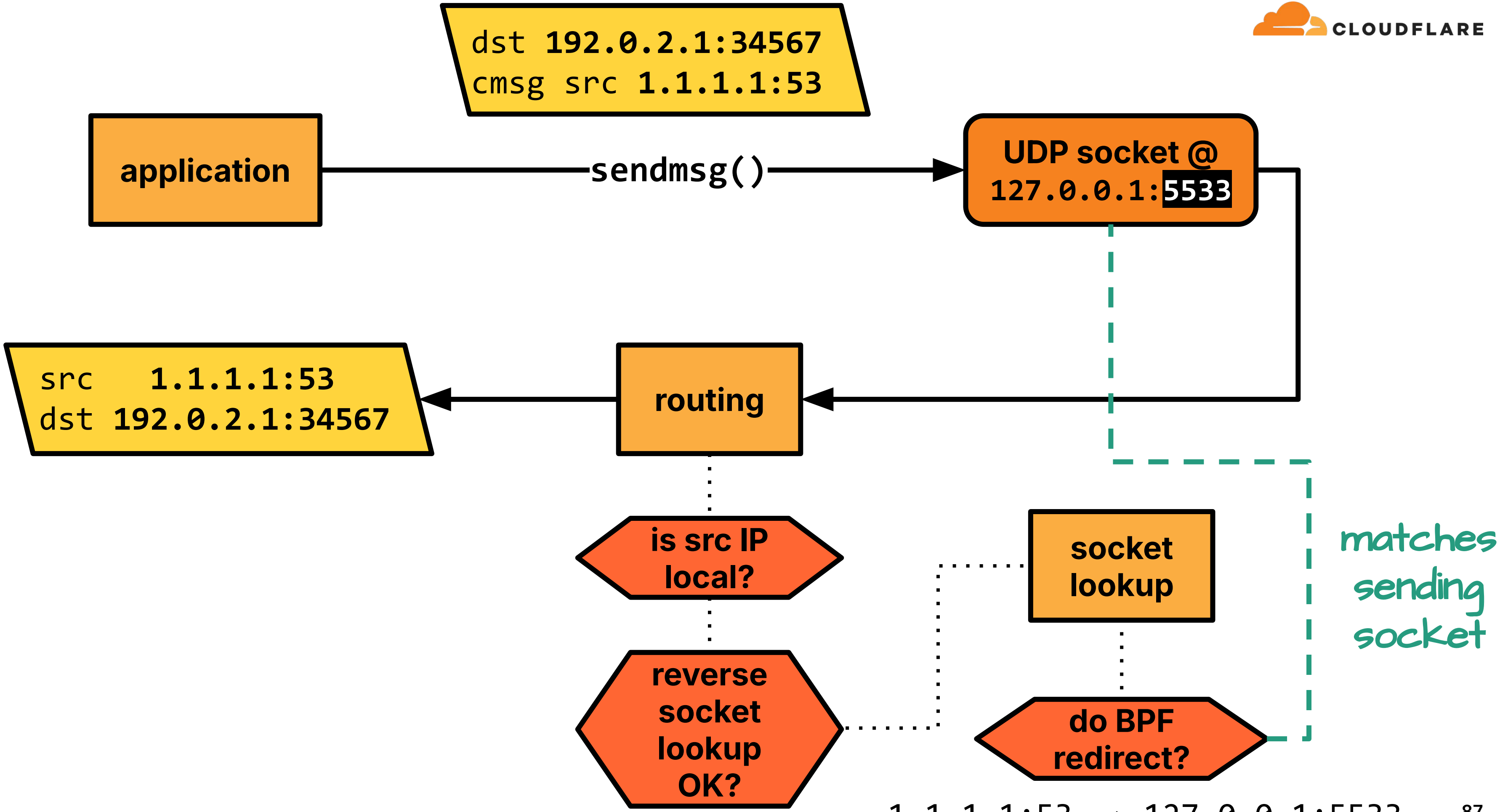
sendmsg()

UDP socket @
127.0.0.1:5533



dst 192.0.2.1:34567
cmsg src 1.1.1.1:53
application
sendmsg()
UDP socket @
127.0.0.1:5533
routing
is src IP local?


dst 192.0.2.1:34567
cmsg src 1.1.1.1:53
application
sendmsg()
UDP socket @
127.0.0.1:5533
routing
is src IP local?
reverse socket lookup OK?
socket lookup
do BPF redirect?
1.1.1.1:53 → ???



dst 192.0.2.1:34567
cmsg src 1.1.1.1:53

application

sendmsg()

UDP socket @
127.0.0.1:5533

src 1.1.1.1:53
dst 192.0.2.1:34567

routing

is src IP local?

reverse socket lookup OK?

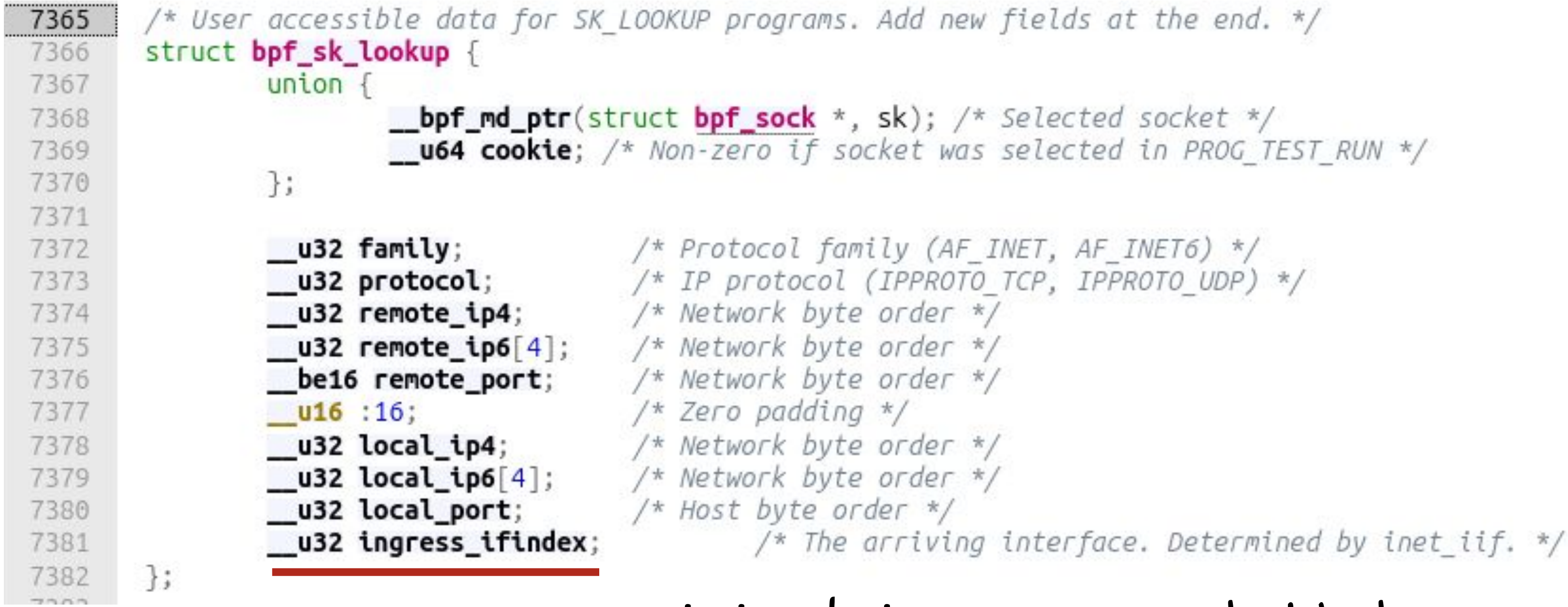
socket lookup

do BPF redirect?

matches sending socket

1.1.1.1:53 → 127.0.0.1:5533

Caveats?



```
/* User accessible data for SK_LOOKUP programs. Add new fields at the end. */
struct bpf_sk_lookup {
        union {
                __bpf_md_ptr(struct bpf_sock *, sk); /* Selected socket */
                __u64 cookie; /* Non-zero if socket was selected in PROG_TEST_RUN */
        };

        __u32 family;           /* Protocol family (AF_INET, AF_INET6) */
        __u32 protocol;         /* IP protocol (IPPROTO_TCP, IPPROTO_UDP) */
        __u32 remote_ip4;       /* Network byte order */
        __u32 remote_ip6[4];    /* Network byte order */
        __be16 remote_port;     /* Network byte order */
        __u16 :16;              /* Zero padding */
        __u32 local_ip4;        /* Network byte order */
        __u32 local_ip6[4];     /* Network byte order */
        __u32 local_port;       /* Host byte order */
        __u32 ingress_ifindex;          /* The arriving interface. Determined by inet_iif. */
};
```

missing during reverse socket lookup
do we just fill it with egress ifindex?
what if we have asymmetric routing?

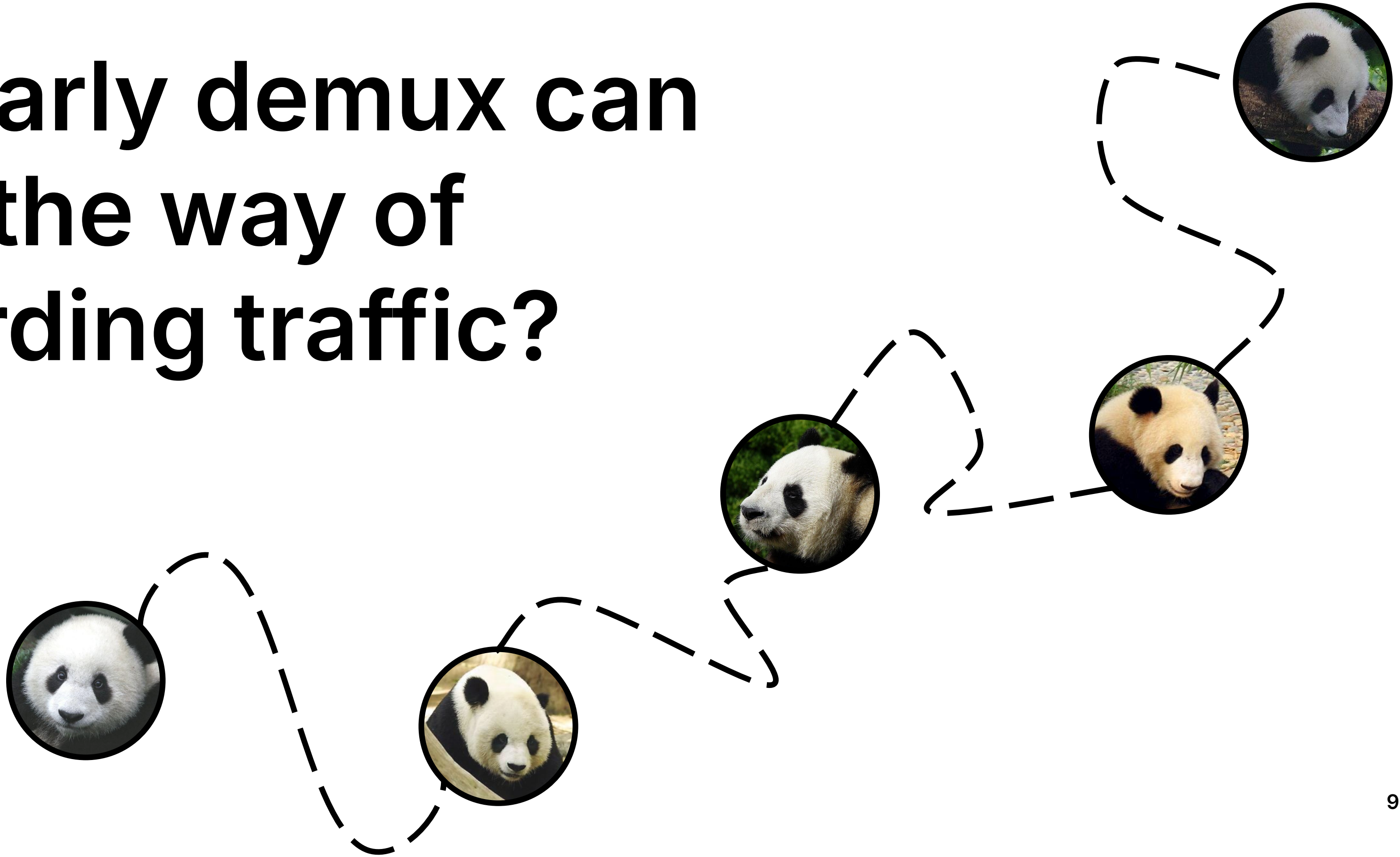
https://elixir.bootlin.com/linux/v6.10/source/include/uapi/linux/bpf.h#L7365

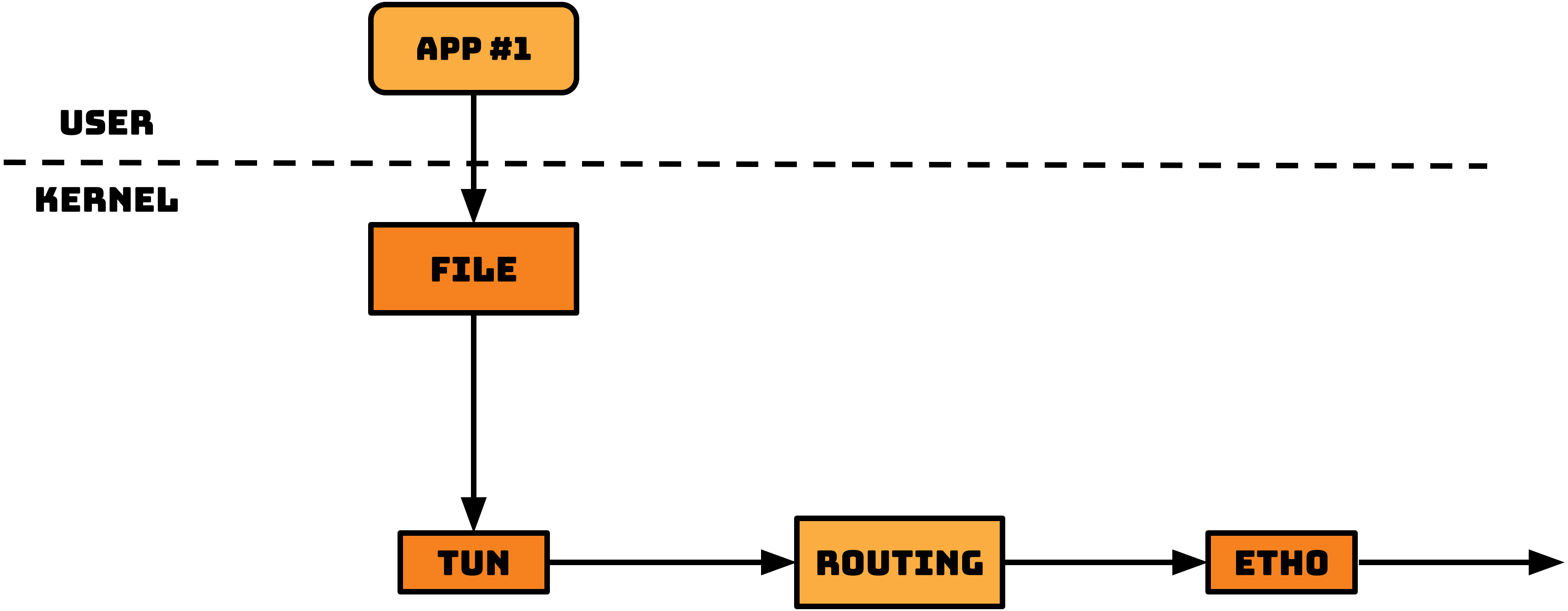
📨 RFC posted to netdev

[RFC PATCH 0/3] Allow sk_lookup UDP return traffic to egress.
https://lore.kernel.org/r/20240913-reverse-sk-lookup-v1-0-e721ea003d4c@cloudflare.com

🙏 Tiago Lam

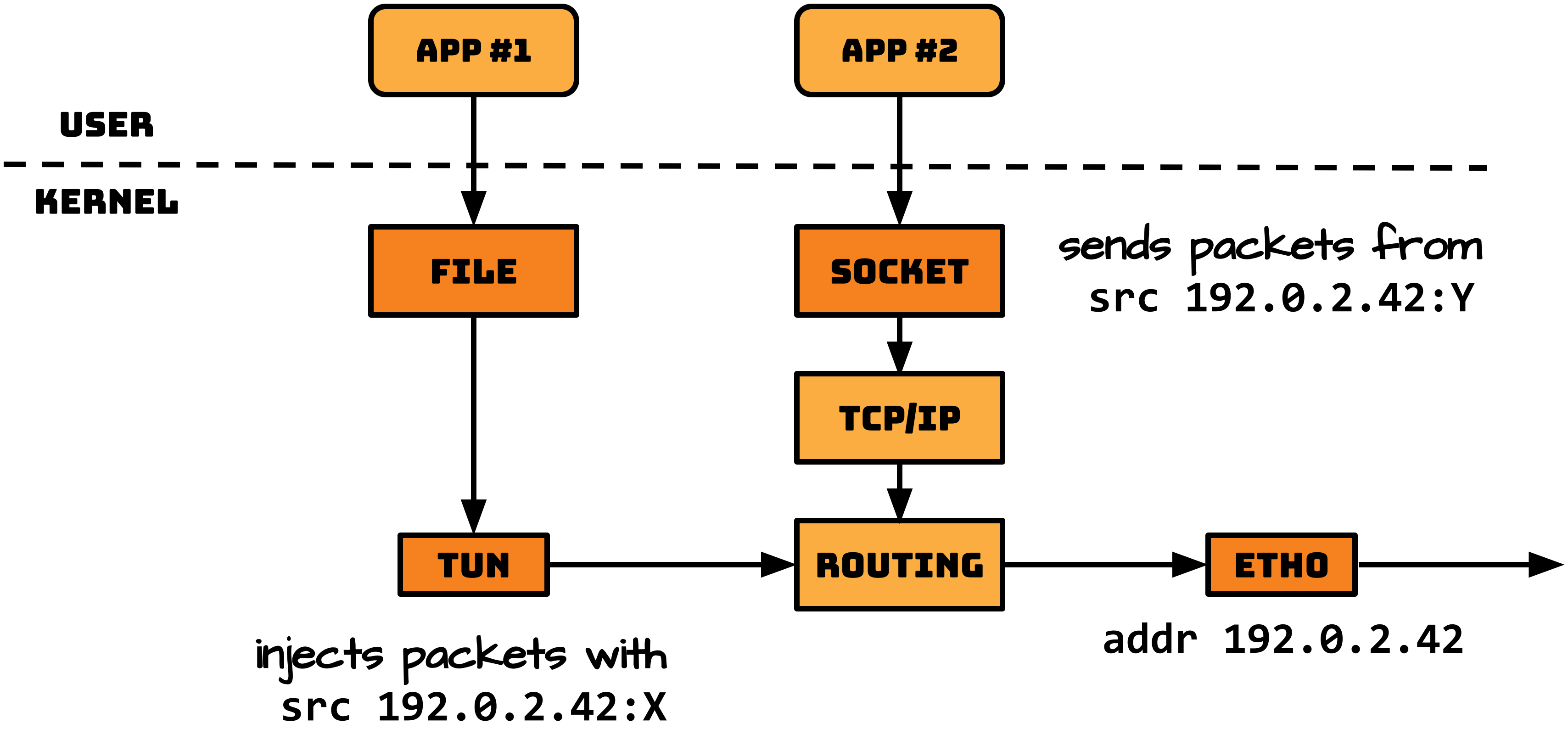
❺ How early demux can get in the way of forwarding traffic?





injects packets with
src 192.0.2.42:X

addr 192.0.2.42



sends packets from
src 192.0.2.42:Y

injects packets with
src 192.0.2.42:X

addr 192.0.2.42

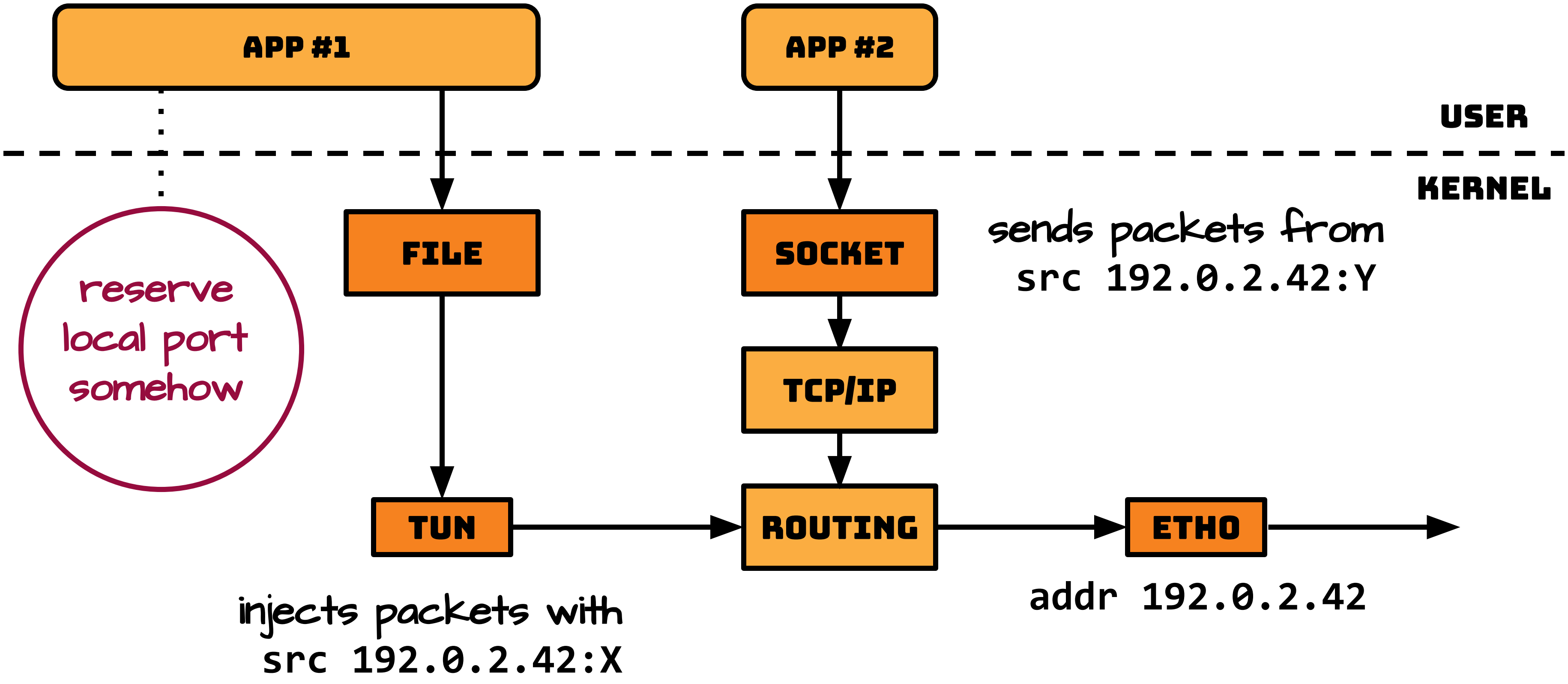
Goals:

1. guarantee that we won't have a local port clash

```
src port X != src port Y
```

2. app #2 delegates local port search to the kernel

```
setsockopt(IP_BIND_ADDRESS_NO_PORT)
bind(192.0.2.42, 0)
connect(...)
```



sends packets from
src 192.0.2.42:Y

injects packets with
src 192.0.2.42:X

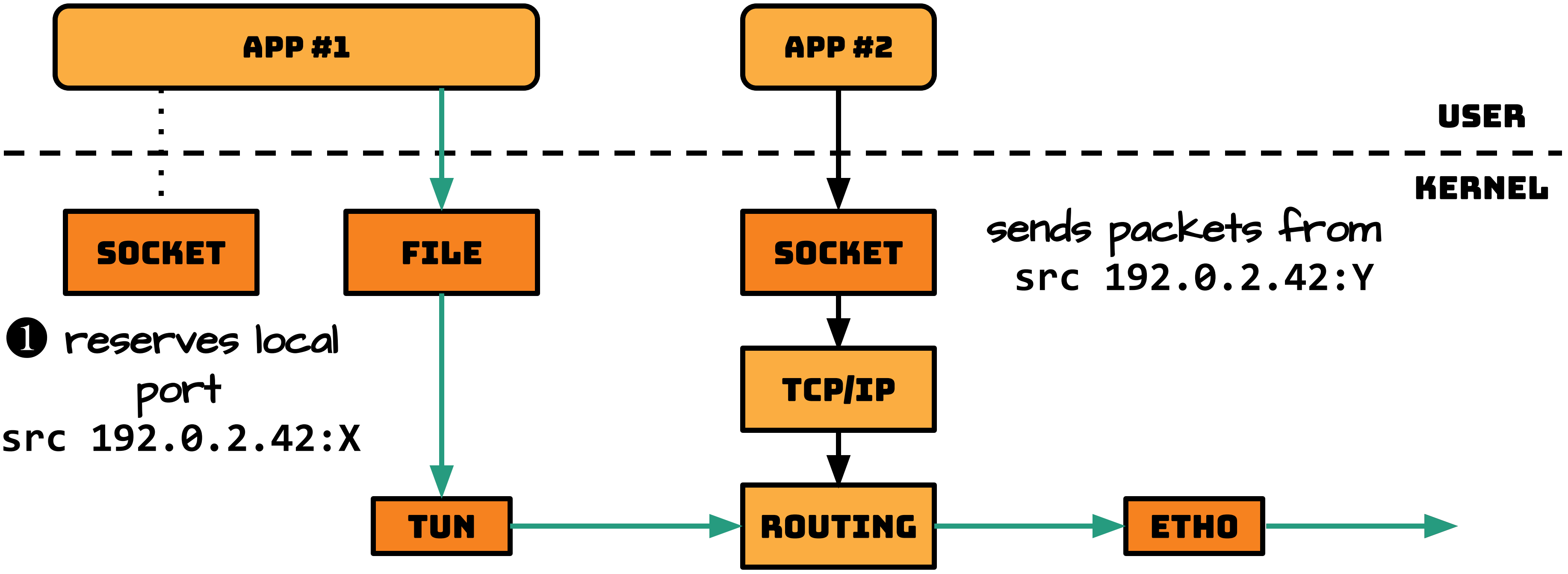
addr 192.0.2.42

How to reserve a local TCP port without sending anything?

```
IP_BIND_ADDRESS_NO_PORT = 24
TCP_FASTOPEN_CONNECT = 30
TCP_FASTOPEN_NO_COOKIE = 34

s = socket(AF_INET, SOCK_STREAM)
s.setsockopt(SOL_IP, IP_BIND_ADDRESS_NO_PORT, 1)
s.setsockopt(SOL_TCP, TCP_FASTOPEN_CONNECT, 1)
s.setsockopt(SOL_TCP, TCP_FASTOPEN_NO_COOKIE, 1)

s.bind(('192.0.2.42', 0))
s.connect(('1.1.1.1', 53))
```

3WHS delayed until first `send()`

```
~ # ss -tanp dst 1.1.1.1
State    Recv-Q Send-Q      Local Address:Port  Peer Address:Port Process
SYN-SENT 0      [REDACTED]     192.0.2.42:54378      1.1.1.1:53   users:(("python3",pid=894397,fd=3))
~ #
```

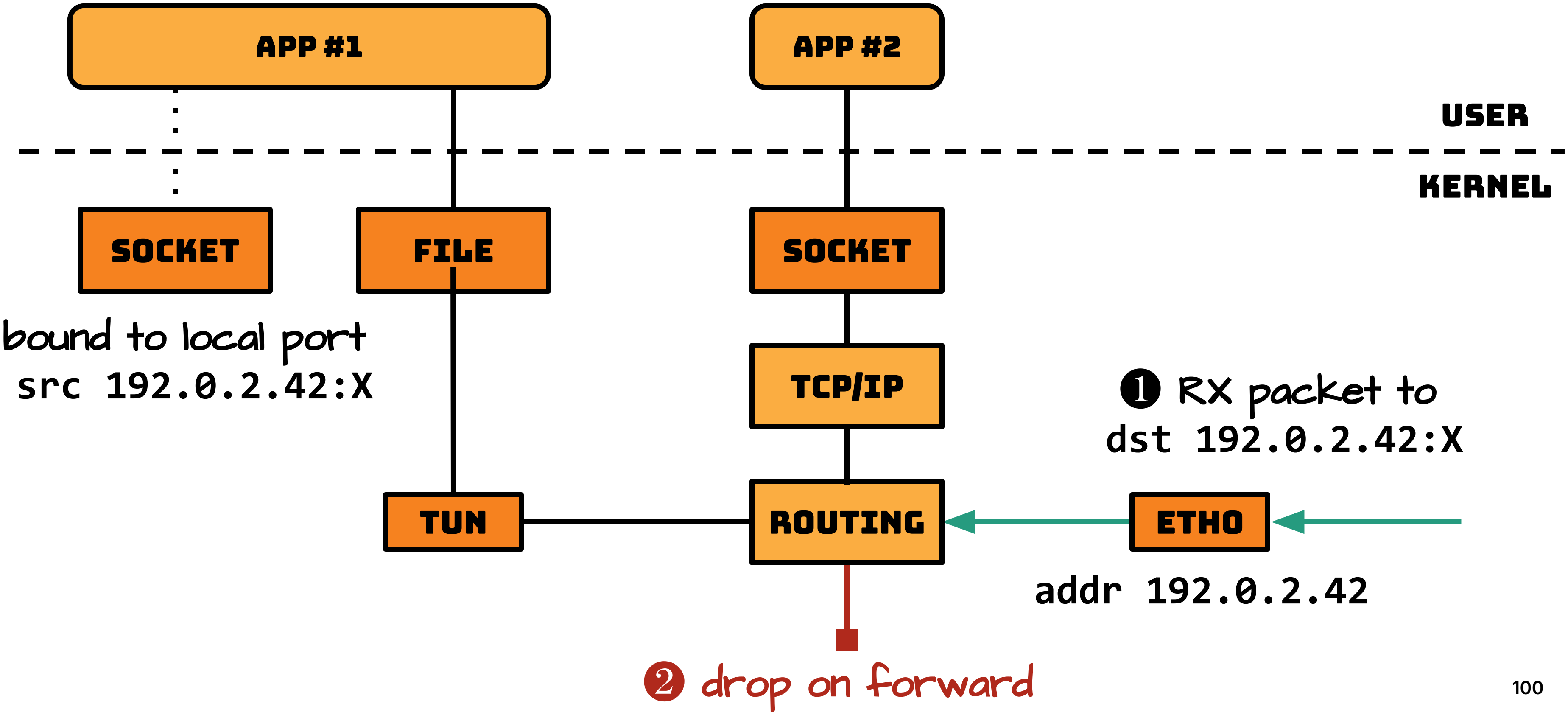


sends packets from
src 192.0.2.42:Y

❶ reserves local port
src 192.0.2.42:X

addr 192.0.2.42

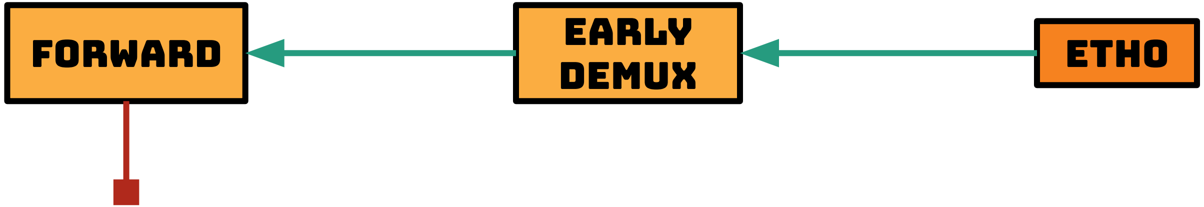
❷ injects packets with
src 192.0.2.42:X


APP #1
APP #2
USER
KERNEL
SOCKET
FILE
SOCKET
bound to local port
src 192.0.2.42:X
TCP/IP
❶ RX packet to
dst 192.0.2.42:X
TUN
ROUTING
ETH0
addr 192.0.2.42
❷ drop on forward

What is happening?

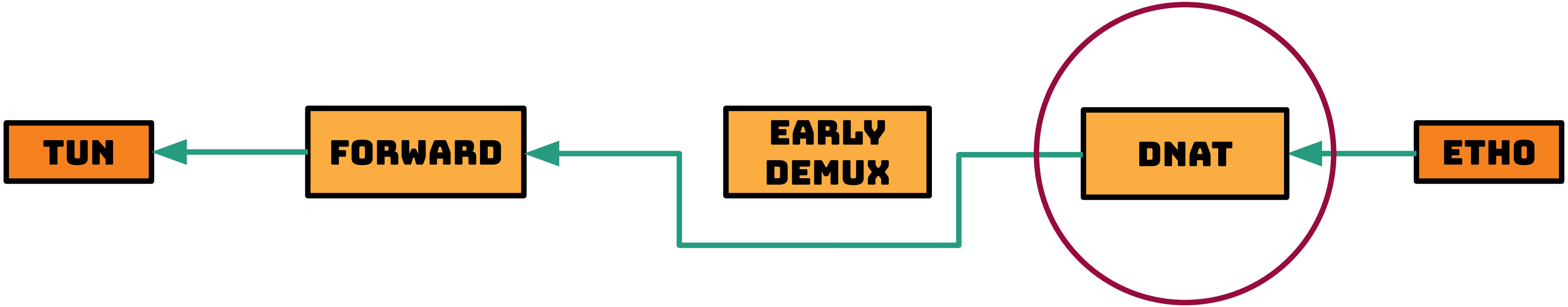
```
if skb->sk is set:
     drop skb
```

```
skb->sk = TCP socket
          (port holder)
```

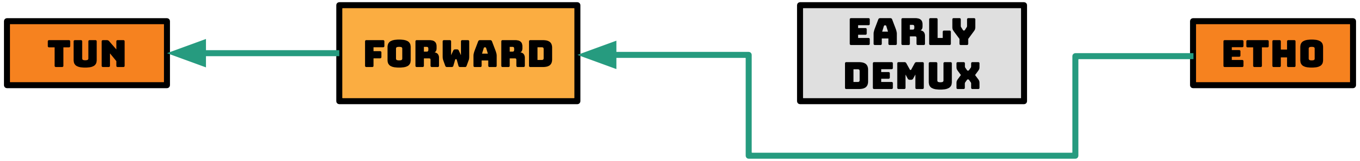


NOTE: Setup also requires an `ip rule` to override local delivery

Workaround #1 - Hide flows from early demux

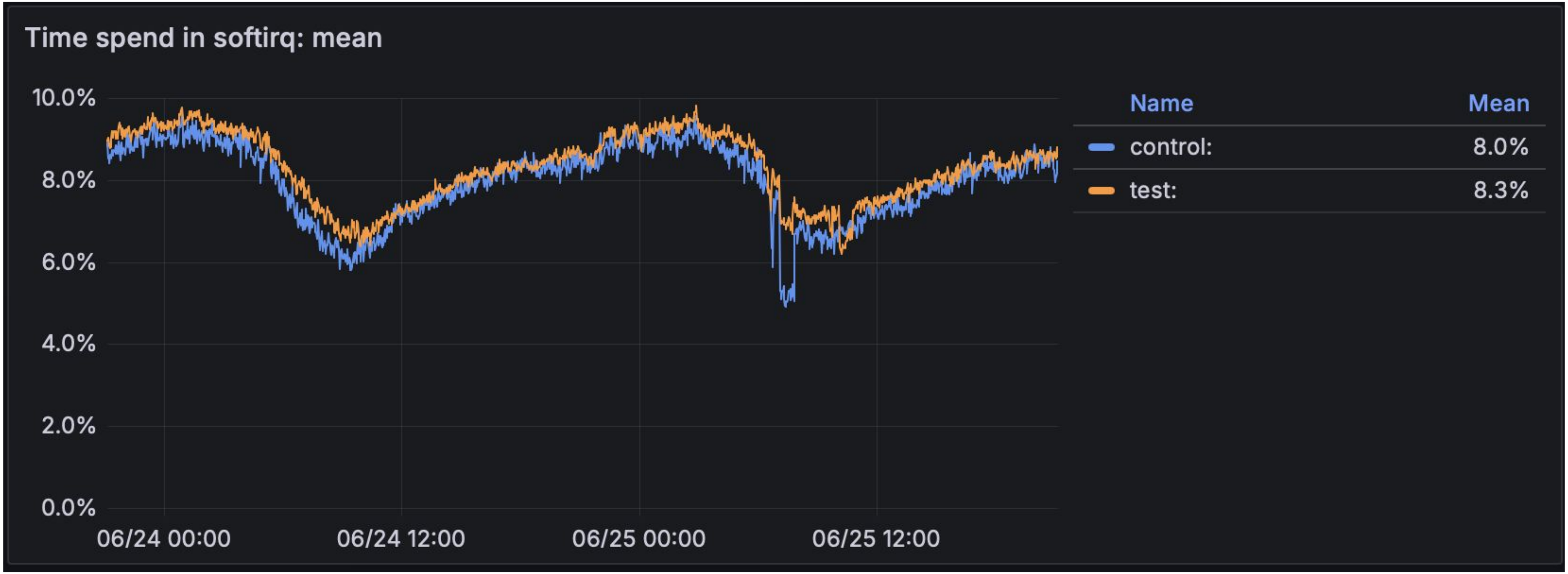


Workaround #2 - Disable early demux



```
sysctl -w net.ipv4.tcp_early_demux=0
```

Workaround #2 - Disable early demux



Name	Mean
▬ control:	8.0%
▬ test:	8.3%

+0.5% CPU time penalty

💡 IDEA

Add `setsockopt(SOL_IP, IP_EARLY_DEMUX, 0)`

Allow for a finer control than `ip_early_demux` sysctl

🙏 Matt Oswalt
🙏 Chris Branch
🙏 Yan Zhai

CLOUDFLARE

Thank you

✉ jakub@cloudflare.com

Section

Image Attribution

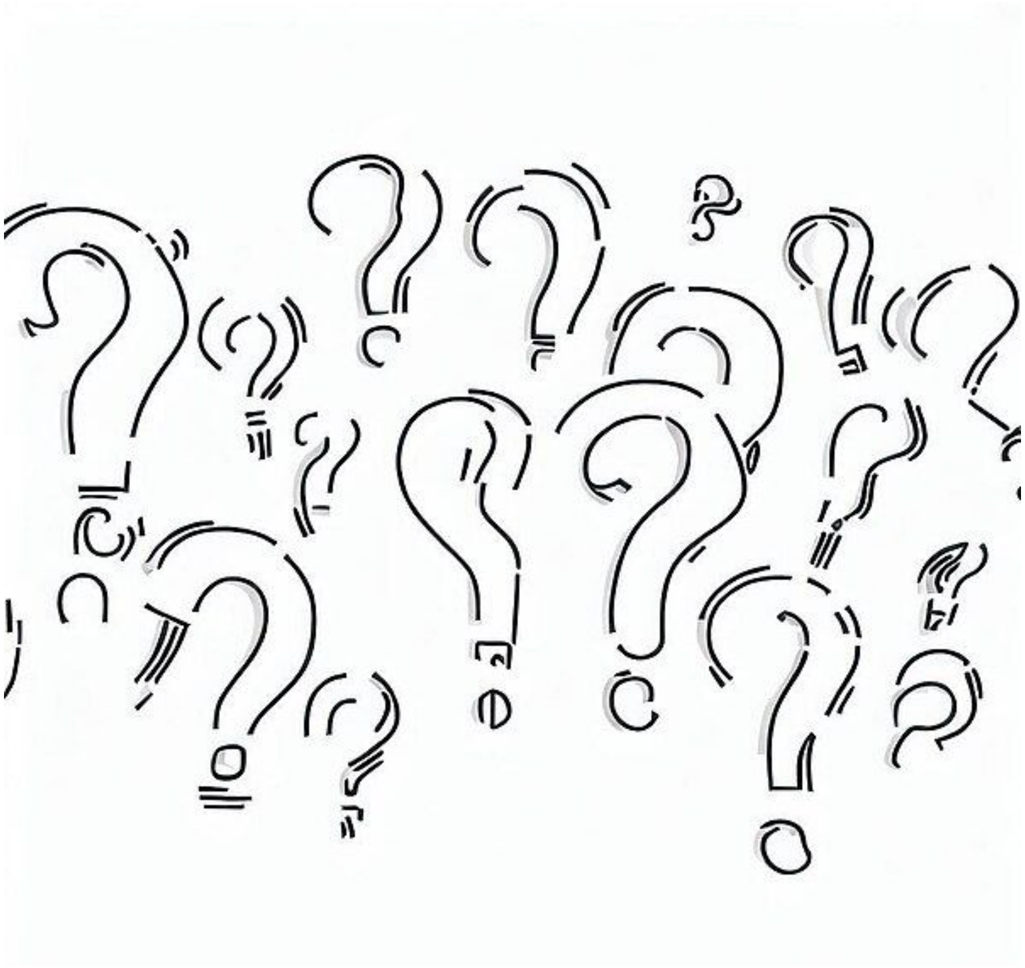
Panda Cub from Wolong, Sichuan, China.JPG - Wikipedia, Public Domain

Lightmatter panda.jpg - Wikipedia, Aaron Logan, CC-BY-1.0

Panda géant - tête (Ailuropoda melanoleuca) (2).jpg - Wikimedia Commons, Gzen92, CC BY-SA 4.0

Großer Panda – Wikipedia, Colegota, CC BY-SA 2.5

2017-07-09 AT Wien 13 Hietzing, Tiergarten Schönbrunn, Ailuropoda melanoleuca - Wikimedia Commons, Paul Korecky, CC BY-SA 2.0



A bunch of question marks, Wikimedia Commons, RMHare, CC BY-SA 4.0 DEED