Linux Forwarding Stack Fastpath

Nishit Shah & Jagdish Motwani

Netdev 1.2, Tokyo, Japan

SOPHOS

Agenda

- Objectives & Challenges
- Proposed Solution
- Stateful Firewall With Proposed Solution
- Performance Numbers
- Future Work / Discussion
- Q & A

Objective & Challenges

Objective & Challenges

- Focus of this work is on linux deployments as routers/firewalls and ideas to improve throughput of forwarding path

- There has been a lot of work going on in the area of linux networking throughput enhancements. Frameworks studied,

  - Netchannel (https://lwn.net/Articles/169961/)
  - Packet Shader (shader.kaist.edu/packetshader/io_engine/index.html)
  - Intel DPDK (dpdk.org/)
  - Netmap (info.iet.unipi.it/~luigi/netmap/)

Objective & Challenges

- Fast Packet Processing Techniques
  - I/O Batching
  - Pre-allocated packet buffers
  - Packet processing without skb (meta-data) allocation
  - Forward cache prefetching
  - Reduce Locking / Lockless operations
  - Memory mapped buffers

- These frameworks are talking about moving the stuff into user-space
  - This might look good for server application(s)
  - Not a practical choice for routers/firewalls as it requires network stack to be written/ported in userspace (Linux is already there ☺)

Proposed Solution

SOPHOS

Proposed Solution

- Proposal is to integrate and enhance networking stack with the fast packet processing techniques mentioned earlier

- In order to evaluate these techniques, we started comparing network stack with a similar application running on Netmap/DPDK
    - A linux device was configured as a simple router by keeping only require modules and unloading other modules
    - When this was compared with a Netmap/DPDK application, there was a good amount of difference between them

Proposed Solution – Fastpath

- As a first step, instead of using standard Rx/Tx path, we used Netmap rings
  - having pre-allocated rx/tx buffers and batch I/O capabilities
  - To use the Netmap rings, network interface card is required to be put in Netmap mode (A small patch in driver is required to support Netmap) in which kernel will see the interface using normal netdevice structure but rx/tx functions are disconnected from network stack

- On Receive side, Netmap framework adds a hook (*netmap_rx_irq*) in driver's napi callback, that is used to wake up the userspace process, which in turns calls the *netmap_rxsync* function to receive the packets in the Netmap ring

- In order to transmit the packet, the application fills Netmap's tx-ring and calls *netmap_txsync* that in turn calls driver specific Tx function

Proposed Solution – Fastpath

- In the modified approach, instead of waking up the userspace process from napi callback, *netmap_rxsync* is called to get the packets in netmap ring and packets are processed in kernel space
- For each received packet in the batch, a top level fastpath function (*do_fastpath*) is called, which does the routing lookup using packet data
- Respective functions of routing code are modified to use packet data instead of skb structure
- On Transmit side, only the packet pointers are moved from rx-rings to tx-rings, transmit signals are issued to NIC for batch mode transmit through *netmap_txsync* function

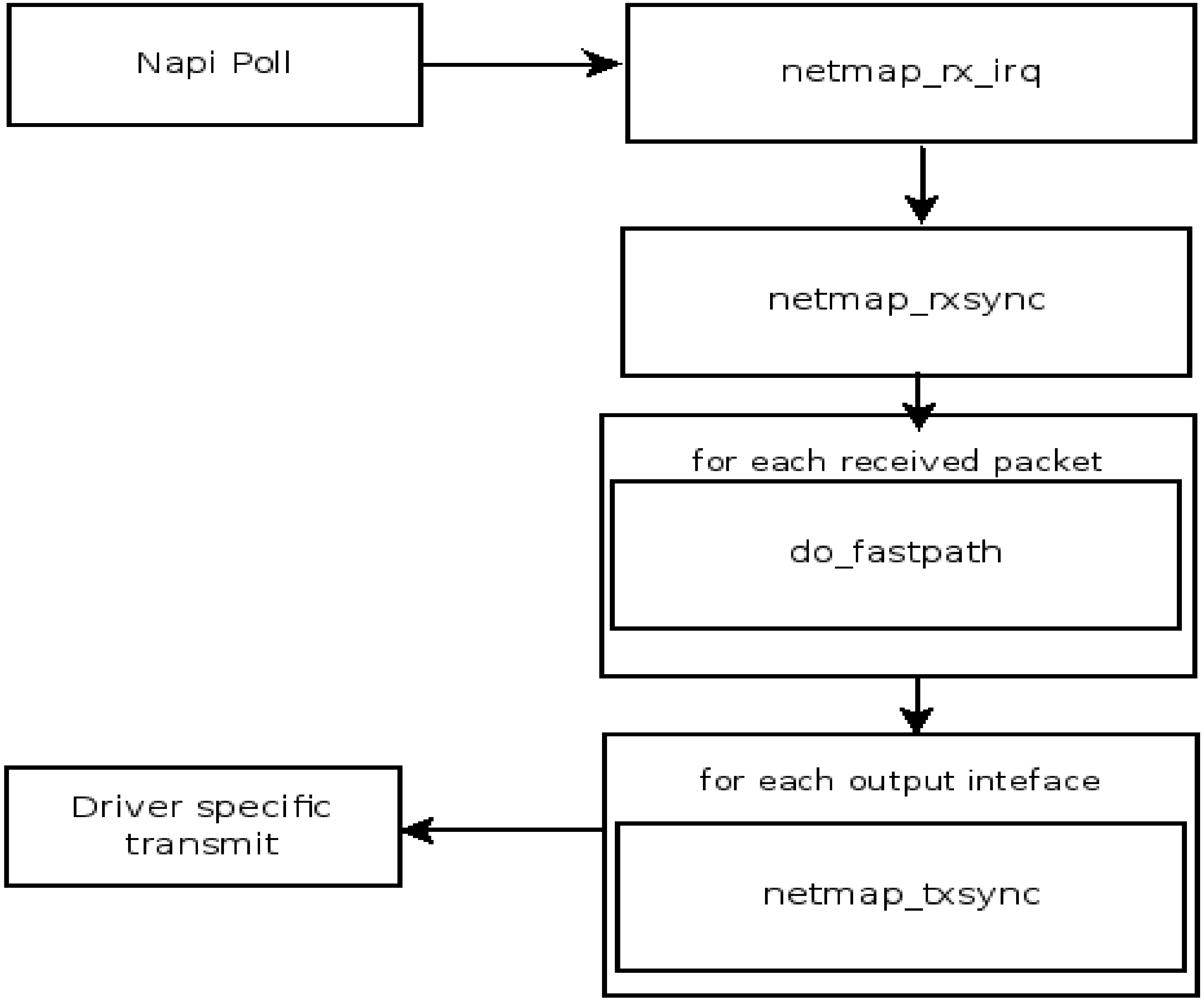

Figure: Modified Packet RX/TX

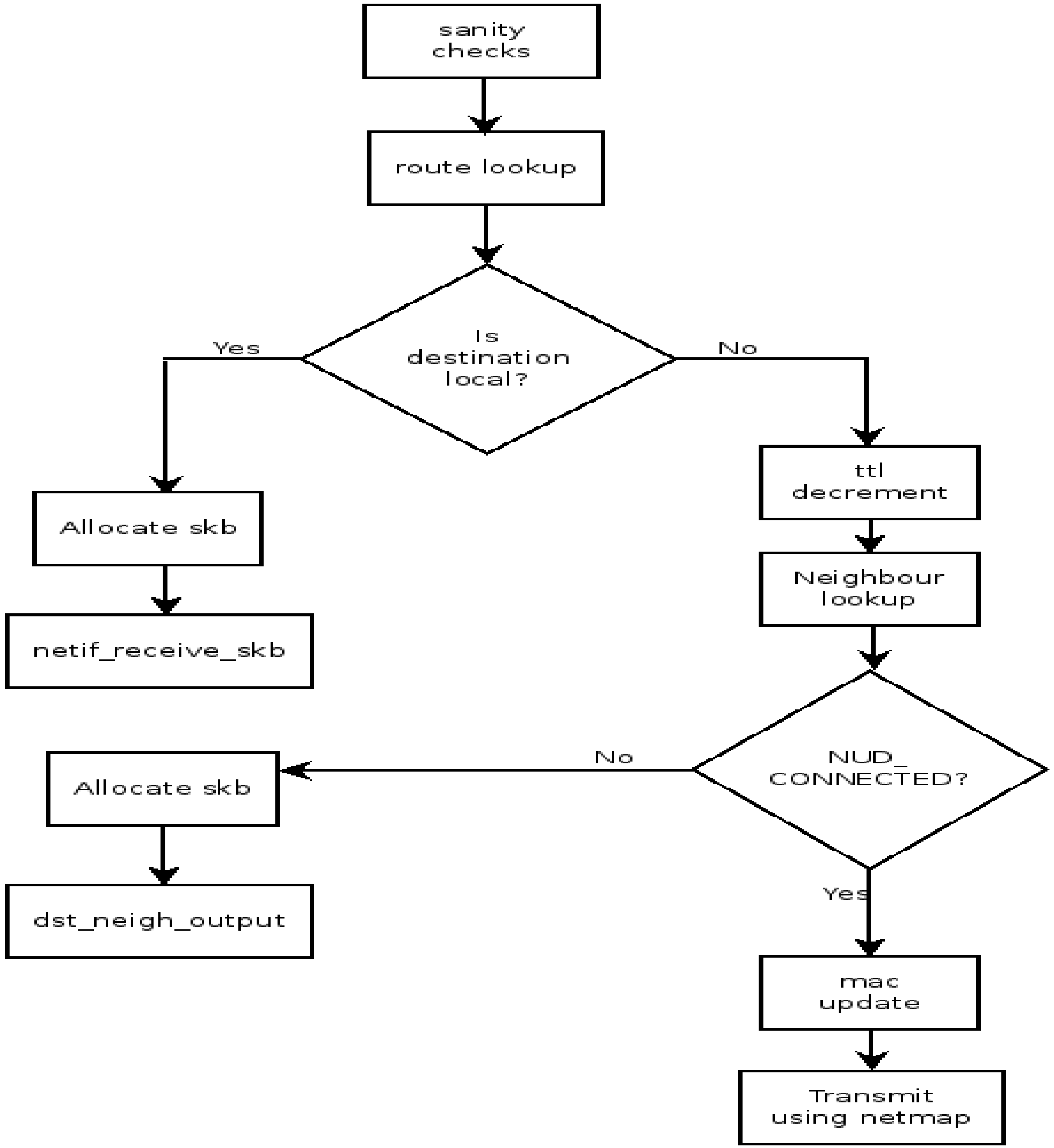

Figure: do_fastpath function

Proposed Solution – Co-existence with network stack

- As we are in kernel, kernel stack is used for packets which are,
  - Either not supported/ported on fastpath (i.e. ARP packets, Fragmented packets, Packets without valid neighbour cache entry etc.)
  - Or not required to be moved on fastpath (i.e. Control plane traffic)
  - For such packets, skbs are allocated, data is copied & respective stack functions are called
- For device originated control plane traffic, Netmap changes the driver specific transmit function to *netmap_transmit*. This function is modified to copy the skb data into netmap tx-ring, followed by call to netmap txsync function to transmit the packet

Proposed Solution – Forward Cache Prefetching

- While going through some sample applications of DPDK, we saw the use of forward cache prefetching

- As we have batch of packets in Netmap rings, forward prefetching can be used here. That is,
  - On having N packets in NAPI callback,
  - Prefetch first 3 packets
  - Process 1st packet & prefetch 4th packet, process 2nd packet & prefetch 5th packet and so on until Nth packet

- It has helped significantly on x86 and x86_64 platforms where DDIO is not supported

Stateful Firewall with Fastpath

- Like we converted routing code to use packet data instead of skb, we also modified connection tracking and NAT code in the same way
- From *do_fastpath* function, conntrack lookup has been done and if it was the first packet of a connection, it was sent to network stack to complete the full journey
- CONNMARK target was used in iptables rules to set FAST_PATH mark to indicate the connection in fastpath
- For all subsequent packets of the same connection, FAST_PATH mark was checked during conntrack lookup and if set, it was transmitted through fastpath

Stateful Firewall – Avoiding Two Lookups

- Florian Westphal proposed a patch[3] "[RFC] netfilter: conntrack: cache route for forwarded connections" that caches dst entries in conntrack structure
- Using it, we avoided route lookups once we do the connection lookups in fastpath

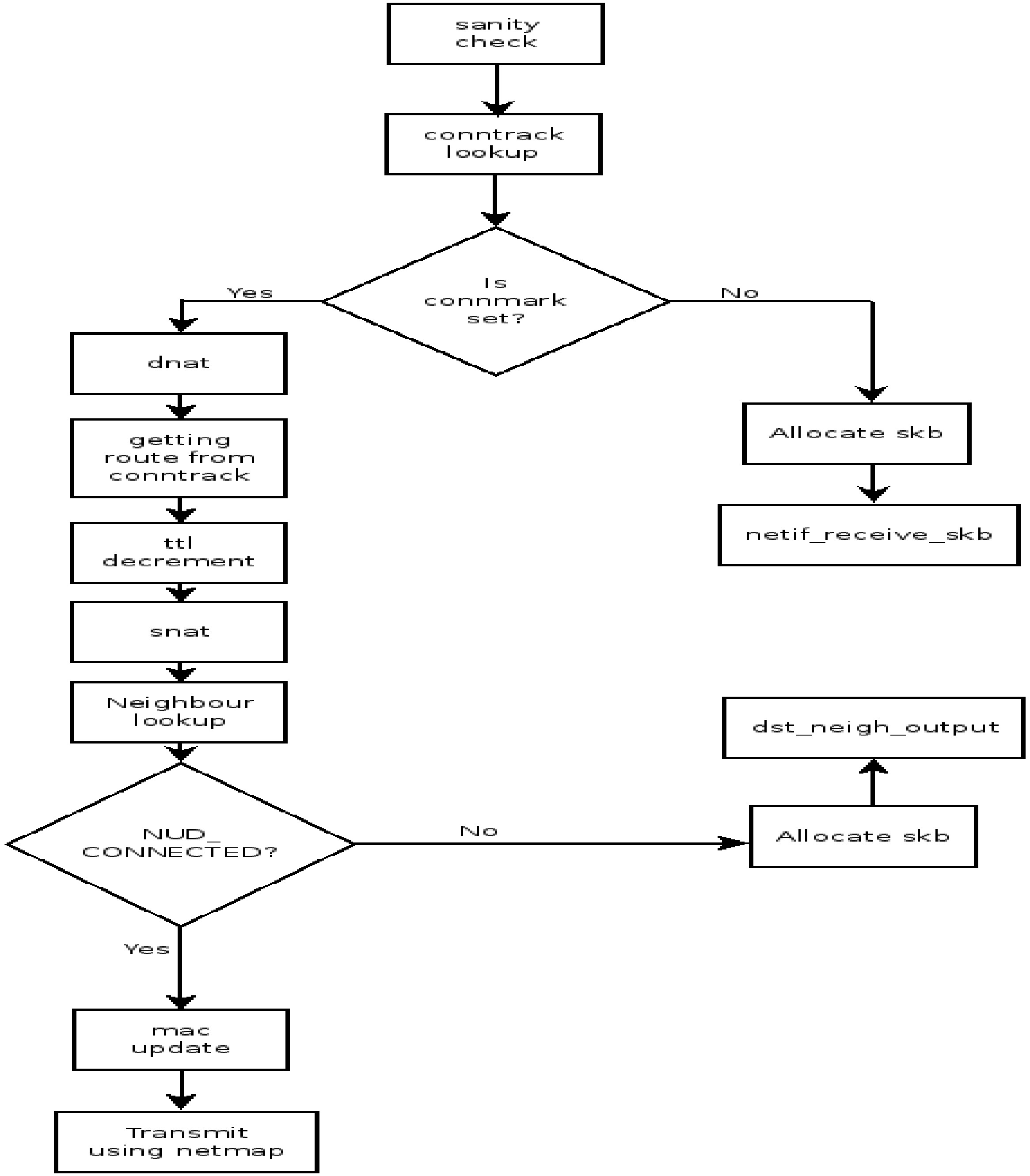

Figure: Firewall Fastpath

Performance Numbers

Performance Numbers

- Table shows some performance numbers taken on linux kernel version 3.14
- Results were taken on a single core of Intel(R) Xeon(R) CPU E5-2680 v3@2.50GHz processor with two 10G ports connected to Ixia Breaking point systems

UDP (Packet Size)	Stateful Firewall	Stateful Firewall With Fastpath	% Improvement
64 bytes	645 Mbps 1320960 pps	3114 Mbps 6377472 pps	4.8x
128 bytes	1155 Mbps 1182720 pps	6240 Mbps 6389760 pps	5.4x
256 bytes	2165 Mbps 1108480 pps	11690 Mbps 5985280 pps	5.3x
512 bytes	4170 Mbps 1067520 pps	20000 Mbps 5120000 pps	4.7x*

Future Work / Discussion

- Packet/Buffer holding support in Netmap rings
- Avoid data copies when sending packets to kernel network stack (Jesper's page-pool)
- XDP (eXpress Data Path) Possibility
- Fastpath Porting: xfrm, bridge, iptables/ipset/nftables, QoS

Thank You

SOPHOS