OPENFABRICS ALLIANCE

Network Adapter Flow Steering

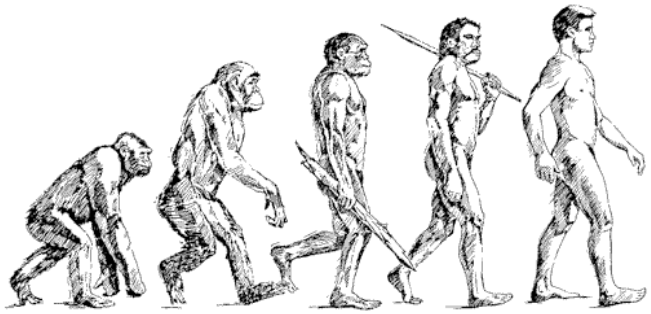
OFA 2012

Author: Tzahi Oved

Date: March 2012

Receive Steering Evolution

- *The traditional Single Ring*
  - All ingress traffic to land on a single receive ring
- *Kernel threads / DPC de-multiplexing*
  - Interrupt to de-mux ingress handling to multiple Kernel threads / DPCs
  - Still use single HW ring
  - May be aided with flow based HW hash
- *Receive Side Scaling (RSS)*
  - Multiple RX rings and interrupts, typically per core
  - Ingress flow hash based distribution to cores (QP->CQ->EQ->MSI->Core)
- **Flow Steering**
  - Network adapters to support TCP/UDP/IP transports flow steering
  - Create as many receive rings as you need
  - Adapter to steer ingress traffic to dedicated receive rings according to 5-tuple classification rule
  - Rich specification of flow to ring mapping
  - And… do it from user mode too!



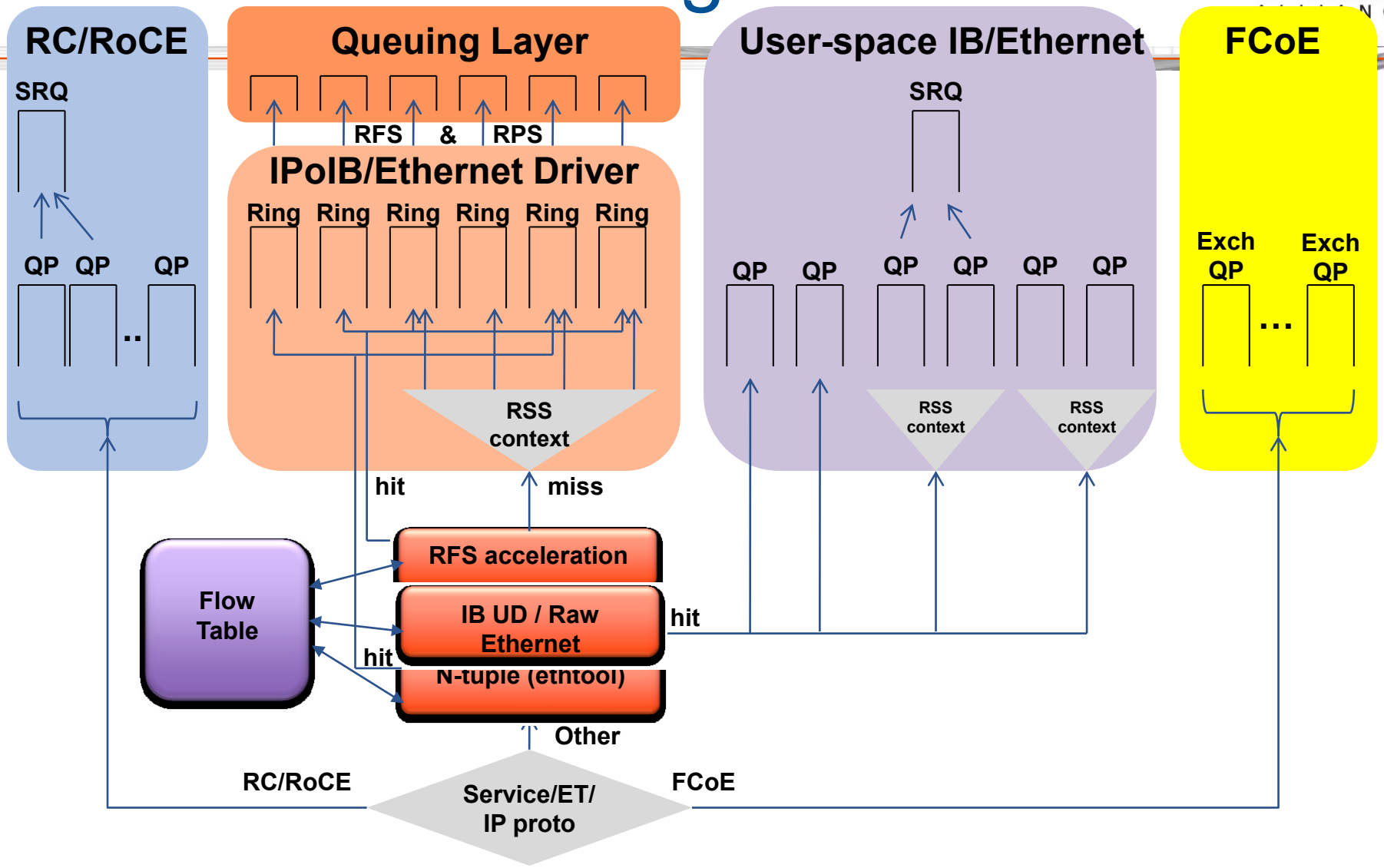
Receive Steering – Why?

- Network stack acceleration
  - NUMA optimized
    - Receive buffer is allocated close to the flow handling core
  - Granular flow to core processing
    - Ingress flows are directed to the matching receiving application core
      - Don't interrupt other cores
    - OS/bypass, application ingress processing run on the same cache
  - Receive filtering at hardware level
- User mode application acceleration
  - TCP/UDP offload applications can steer its flows directly to user mode
    - Hijacking flows from kernel driver
  - Building block of user mode stack

Implementation Overview

- Hierarchical Steering Capabilities
  - Transport: Ethernet, RoCE, and FCoE / IB-UD
  - Flow
    - MAC/VLAN/priority/ET or d.QP
    - IP protocol
    - TCP/UDP ports
  - RSS contexts, QPs, and SRQs
- Concurrent Support for Multiple steering Interfaces
  - NIC Driver
    - ethtool n-tuple settings
    - RPS, RFS acceleration
    - RSS
  - Verbs Consumers (Kernel and User)
    - RC, RoCE QPs
    - UD, Raw Ethernet QPs
  - FCoE
- Extends Naturally to Virtualized Environment (SR-IOV)
  - Steering rules table is maintained per VF

Receive Flow Diagram



Flows and Priorities

- Steering rules ordered in a layered hierarchy
  - Domains
    - ethtool
    - Verbs (offloaded traffic)
    - RFS, Netdev (network stack)
  - Priority
- Multiple rules can exist in the same domain+priority
  - For different flows
    - Insertion order applies, last wins
  - For same flows
    - Adapter to distribute ingress to multiple rings
- Rule processing order
  - Scan:
    - (1) domain, (2) priority, (3) location
  - First match takes
- Rule actions
  - Steer to QP
    - Multiple QPs are also allowed
  - Drop

Domain	Priority	Pattern	QP
ethtool	1	tcp flow f	ring1
	2	src+dst ip g	ring2
	3	dst ip h	drop
Verbs & RDMA-CM	1	tcp flow a	qp1
	1	tcp flow b	qp2
	1	udp mcast c	qp2
	2	tcp listener d	qp3
	2	tcp listener e	qp3
Linux RFS	999	dst_ip i	ring3
	999	tcp flow j	ring4
	999	tcp flow k	ring4
Net_dev	999	tcp flow l	ring4
	999	tcp flow m	ring5
	999	udp flow n	ring6

ethtool rules

Flows

Listeners

RFS acceleration

attach_mcast

The API - Kernel Verbs

- flow_id ib_attach_flow(target_qp, domain, prio, flow_spec*)
  - Attach QP ***target_qp*** to flow ***flow_spec*** in
    - Domain ***domain***
    - Priority ***prio***
  - Attaching multiple QPs to the same flow+domain+prio allowed
    - Adds the QPs to the same rule
    - Replicates traffic to all QPs
- ib_detach_flow(flow_id)
  - Detach QP ***target_qp*** from flow ***flow_spec*** in
    - Domain ***domain***
    - Priority ***prio***
  - Rule is actually removed once all QPs are removed
- Flow specification: IPv4, IPv6, IB-UD/RoCE-UD
  - Built from L2, L3, L4 protocol headers ordered filters and masks
  - L2: DMAC, VID, ET
  - L3: src IP, dst IP, protocol
  - L4: src port, dst port
  - IB/RoCE: dst QP, dst GID

Flow Filter Specification

- Extendible flow specification structure
- Preceded by a flow specification header
- struct flow_spec
  - uint filters_num; // Number of filters describing the flow
  - void *next_flow_spec; // pointer to next flow spec filter
- General flow specification structure:
  - struct flow_spec_xx
    - enum flow_spec_type // eth/arp/ipv4/ipv6/udp/tcp/ud
    - struct flow_spec_xx // flow filter characteristics + mask
    - void* next_flow_spec; // header specific flow characteristics

Flow Filter Specification – An Example

- struct flow_spec_eth
  - enum flow_spec_type type;
  - struct flow_spec_eth_filter filter;
  - struct flow_spec_eth_filter filter_mask;
  - void * next_flow_spec;
- struct flow_spec_eth_filter
  - u8 d.addr[ETH_ALEN];
  - u8 s.addr[ETH_ALEN];
  - u16 protocol;
  - u16 vid;
  - u8 priority;
- The same for rest of header types:
  - struct flow_spec_arp, struct flow_spec_arp_filter
  - struct flow_spec_ipv4, struct flow_spec_ipv4_filter
  - struct flow_spec_ipv6, struct flow_spec_ipv6_filter
  - struct flow_spec_udp, struct flow_spec_udp_filter
  - struct flow_spec_tcp, struct_ flow_spec_tcp_filter
  - struct flow_spec_ud, struct flow_spec_ud_filter

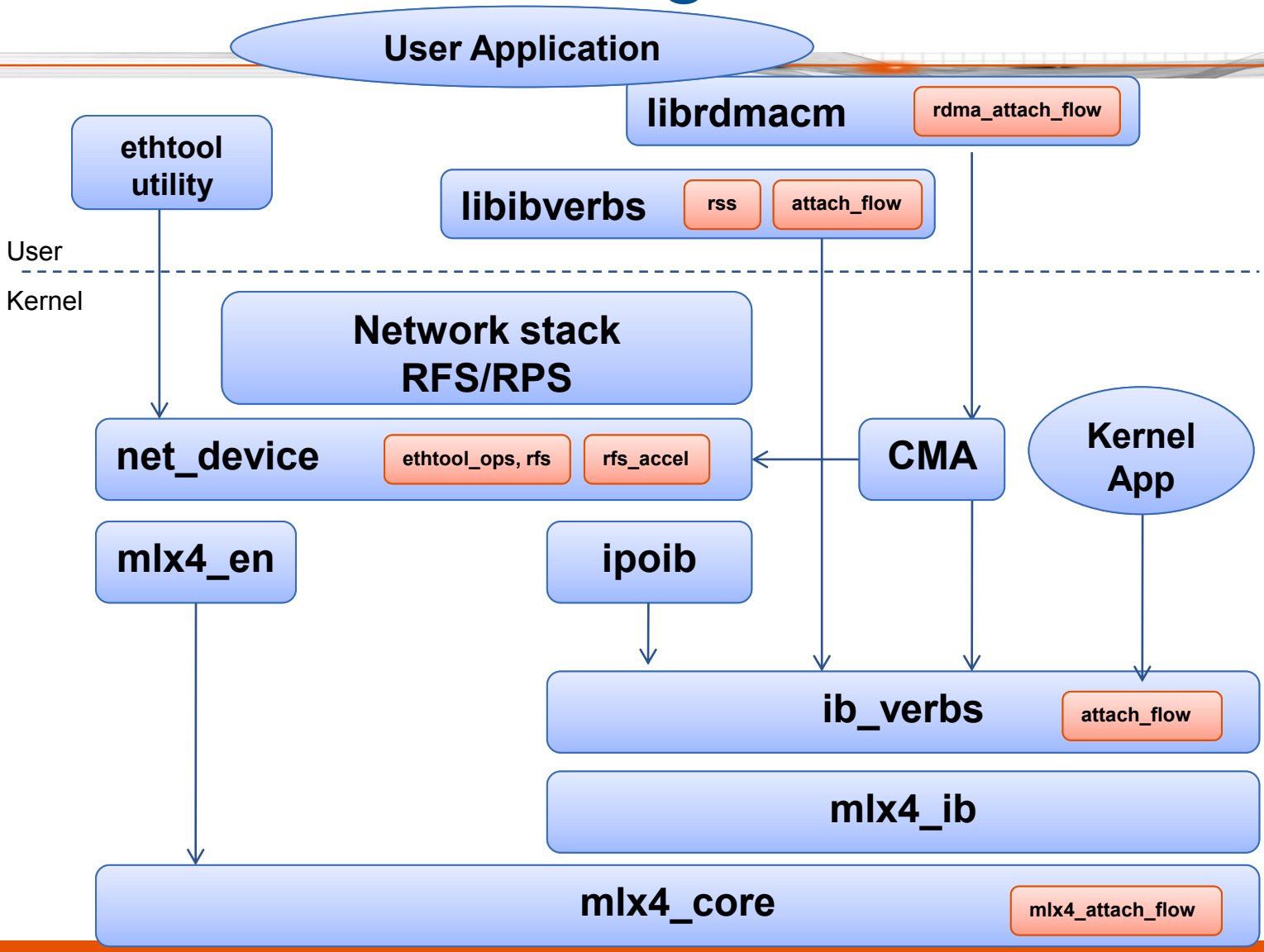
User Verbs Provider Interface

- QP type: IBV_QPT_RAW_ETH, IBV_QPT_UD
- Similar to kernel (except for fixed domain = UVERBS_DOMAIN)
  - flow_handle ibv_attach_flow(target_qp, prio, flow_spec*)
  - ibv_detach_flow(flow_handle)
- Requires CAP_NET_RAW privileges
- Resources tracked by uverbs module
  - Flows are tracked as process resource
  - When terminates, all it's remaining allocated flows are detached

rdmacm Abstraction

- RDMA-CM will use rdma_cm_id identifier for flow characteristic
  - rdma_id is the “handle” to the target QP
- API calls:
  - flow_handle rdma_attach_flow(flow_rdma_id, dest_rdma_id)
    - flow_rdma_id includes enough info to build the flow_spec
    - For L2 params:
      - Extract local link layer params from local interface (IP address to MAC+VLAN / GID+QP)
      - Depending on rule type, extract remote link layer address through neigh_lookup()
    - Will use 3 rule priorities according to flow_rdma_id
      - 1st (highest) - For 5-tuple rule (connect)
      - 2nd - For 3-tuple rule (listen)
      - 3rd – For 2-tuple rule (fragment receive)
    - Use kernel verb ib_attach_flow() call with UVERBS_DOMAIN
  - rdma_detach_flow(flow_handle)

Solution block diagram



Quick Example

rdma_create_id(target_id, PS_UDP)

rdma_create_qp(target_id)

// Create target id->qp

rdma_create_id(source_id, PS_UDP)

// Create source flow spec, set protocol

rdma_bind_addr(source_id, 2-tuple of local IP address + port)

// Set rx flow spec characteristics through sockaddr

Or rdma_resolve_addr(source_id, 2 or 4 tuple)

// src sockaddr to point to local interface+port, dst sockaddr to remote, protocol was set in create_id()

rdma_attach_flow(source_id, target_id)

// Insert our steering rule

Status and What's Next

- Kernel API done
- User API under definition, implementation to follow
- IPv4 support only for now
- Next
  - Bitwise flow spec mask support
  - IPv6
  - Extend flow spec filters for L7 application level data inspection
  - TCP/UDP payload based steering

OPENFABRICS ALLIANCE

Thank You