Lecture 26: Networking

Fall 2018
Jason Tang

Slides based upon Operating System Concept slides,
http://codex.cs.yale.edu/avi/os-book/OS9/slide-dir/index.html

Topics

- OSI Model
- Linux Kernel Networking
- Receiving Packets

Computer Networking

- Computer sends stream of octets (8-bit bytes) over networks
- Each computer on a network has a unique address (or addresses)
- Bytes are often grouped into packets with a destination address
- When an OS receives a packet addressed to it, the OS can either accept it, reject it, or drop it
- When an OS receives a packet destined to a different computer, the OS can either reject it, drop it, or forward (route) it to the correct computer

Networking Layers

OSI (Open Source Interconnection) 7 Layer Model

Layer	Application/Example		Central Device/ Protocols		DOD4 Model
Application (7) Serves as the window for users and application processes to access the network services.	**End User layer** Program that opens what was sent or creates what is to be sent Resource sharing • Remote file access • Remote printer access • Directory services • Network management		**User Applications** SMTP	**GATEWAY** Can be used on all layers	Process
Presentation (6) Formats the data to be presented to the Application layer. It can be viewed as the "Translator" for the network.	**Syntax layer** encrypt & decrypt (if needed) Character code translation • Data conversion • Data compression • Data encryption • **Character Set Translation**		JPEG/ASCII EBDIC/TIFF/GIF PICT		
Session (5) Allows session establishment between processes running on different stations.	**Synch & send to ports** (logical ports) Session establishment, maintenance and termination • Session support - perform security, name recognition, logging, etc.		**Logical Ports** RPC/SQL/NFS NetBIOS names		
Transport (4) Ensures that messages are delivered error-free, in sequence, and with no losses or duplications.	**TCP** Host to Host, Flow Control Message segmentation • Message acknowledgement • Message traffic control • Session multiplexing	PACKET FILTERING	TCP/SPX/UDP		Host to Host
Network (3) Controls the operations of the subnet, deciding which physical path the data takes.	**Packets** ("letter", contains IP address) Routing • Subnet traffic control • Frame fragmentation • Logical-physical address mapping • Subnet usage accounting		**Routers** IP/IPX/ICMP		Internet
Data Link (2) Provides error-free transfer of data frames from one node to another over the Physical layer.	**Frames** ("envelopes", contains MAC address) [NIC card — Switch — NIC card] (end to end) Establishes & terminates the logical link between nodes • Frame traffic control • Frame sequencing • Frame acknowledgment • Frame delimiting • Frame error checking • Media access control		**Switch Bridge WAP** PPP/SLIP	Land Based Layers	Network
Physical (1) Concerned with the transmission and reception of the unstructured raw bit stream over the physical medium.	**Physical structure** Cables, hubs, etc. Data Encoding • Physical medium attachment • Transmission technique - Baseband or Broadband • Physical medium transmission Bits & Volts		**Hub**		

https://buildingautomationmonthly.com/what-is-the-osi-model/

Open Systems Interconnection (OSI) Model

- Separates the various parts of a communication network into multiple layers
- Layer 1, Physical Layer: Handles mechanical and electrical details of physical transmission of a bit stream
- Layer 2, Data Link Layer: Handles frames, or fixed-length parts of packets, including error detection and recovery that occurred in physical layer
- Layer 3, Network Layer: Provides connections and routes packets in the communication network, decoding the address of incoming packets, and maintaining routing information for proper response to changing load levels

Open Systems Interconnection (OSI) Model

- Layer 4, Transport Layer: Responsible for low-level network access and for message transfer between clients, including partition messages into packets, maintaining packet order, controlling flow, and generating physical addresses

- Layer 5, Session Layer: Implements sessions, or process-to-process communications protocols

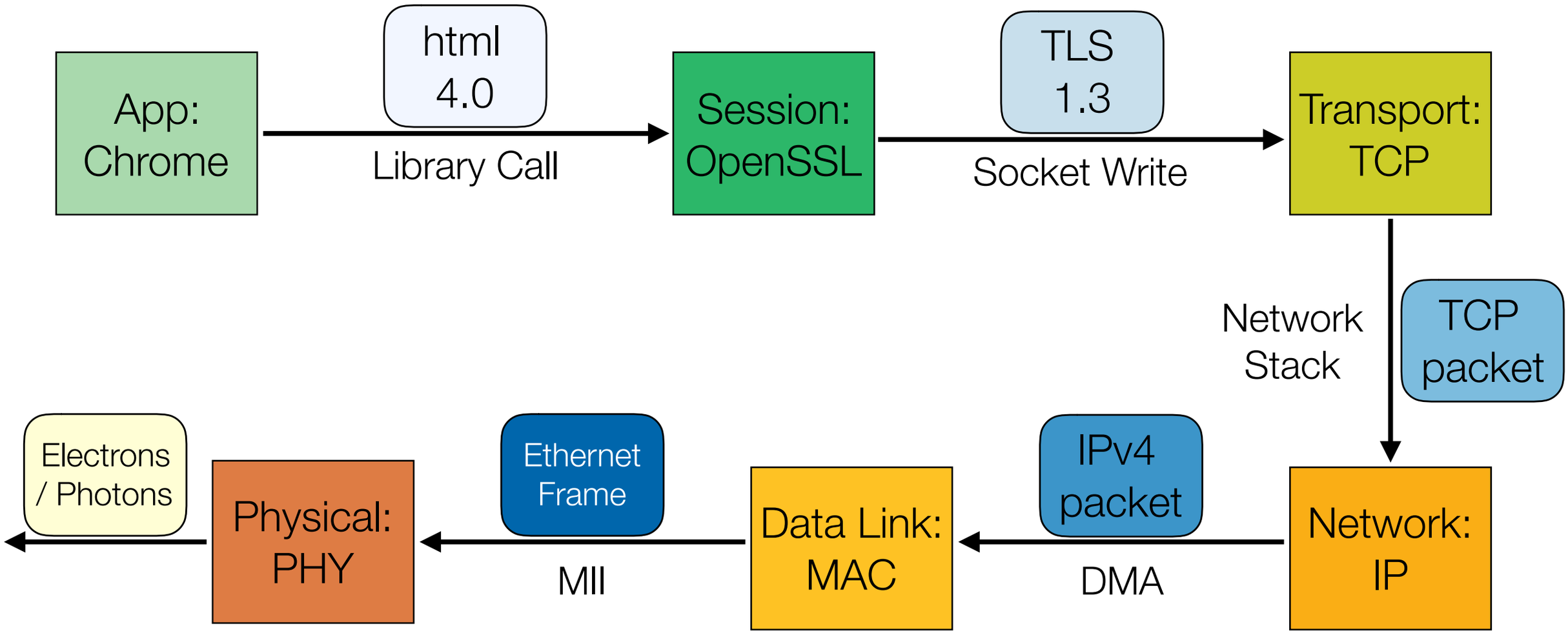
- Layer 6, Presentation Layer: Resolves differences in formats among various sites in the network, including character conversions, and half duplex/full duplex

- Layer 7, Application Layer: Interacts directly with user space, such as file transfer, email, and web pages

Transmission Control Protocol (TCP)

- Common OSI layer 4 protocol, usually via an Ethernet network
- Many higher-order protocols built on top of TCP
- Every host has a unique Internet Protocol (IP) address and a corresponding Ethernet Media Access Control (MAC) address
- Communication requires both IP and MAC addresses
  - Kernel inspects all incoming TCP packets, checking if the packet's destination IP and MAC addresses match that computer

Networking Layers



Ethernet Frame

bytes		
7	preamble—start of packet	each byte pattern 10101010
1	start of frame delimiter	pattern 10101011
2 or 6	destination address	Ethernet address or broadcast
2 or 6	source address	Ethernet address
2	length of data section	length in bytes
0–1500	data	message data
0–46	pad (optional)	message must be > 63 bytes long
4	frame checksum	for error detection

Linux Networking

- Linux kernel designed to handle numerous network configurations
- Kernel primarily deals with layers 2, 3, and 4
- Most important kernel data structures are `struct sk_buff` and `struct net_device`
  - Kernel routes instances of `struct sk_buff` from the various `struct net_device`s throughout its network stack
  - The `sk_buff` can be changed as it works its way through kernel

sk_buff

- Linked list of network data
- Contains both header(s) and payload data

From include/linux/skbuff.h

```
struct sk_buff {
    union {
        struct {
            /* These two members must be first. */
            struct sk_buff      *next;
            struct sk_buff      *prev;

            union {
                ktime_t         tstamp;
                struct skb_mstamp skb_mstamp;
            };
        };
        struct rb_node  rbnode; /* used in netem & tcp stack */
    };
    struct sock         *sk;
    struct net_device   *dev;
```

net_device

- Represents a network interface card (NIC)

From include/linux/netdevice.h

```
struct net_device {
    char                    name[IFNAMSIZ];
    struct hlist_node       name_hlist;
    char                    *ifalias;
    /*
     *  I/O specific fields
     *  FIXME: Merge these and struct ifmap into one
     */
    unsigned long           mem_end;
    unsigned long           mem_start;
    unsigned long           base_addr;
    int                     irq;
...
    const struct net_device_ops *netdev_ops;
    const struct ethtool_ops *ethtool_ops;
```

netdev_ops

- Whereas a `struct miscdevice` has a pointer to a `struct file_operations` containing callbacks, a `struct net_device` has a pointer to a `struct net_dev_ops` for callbacks

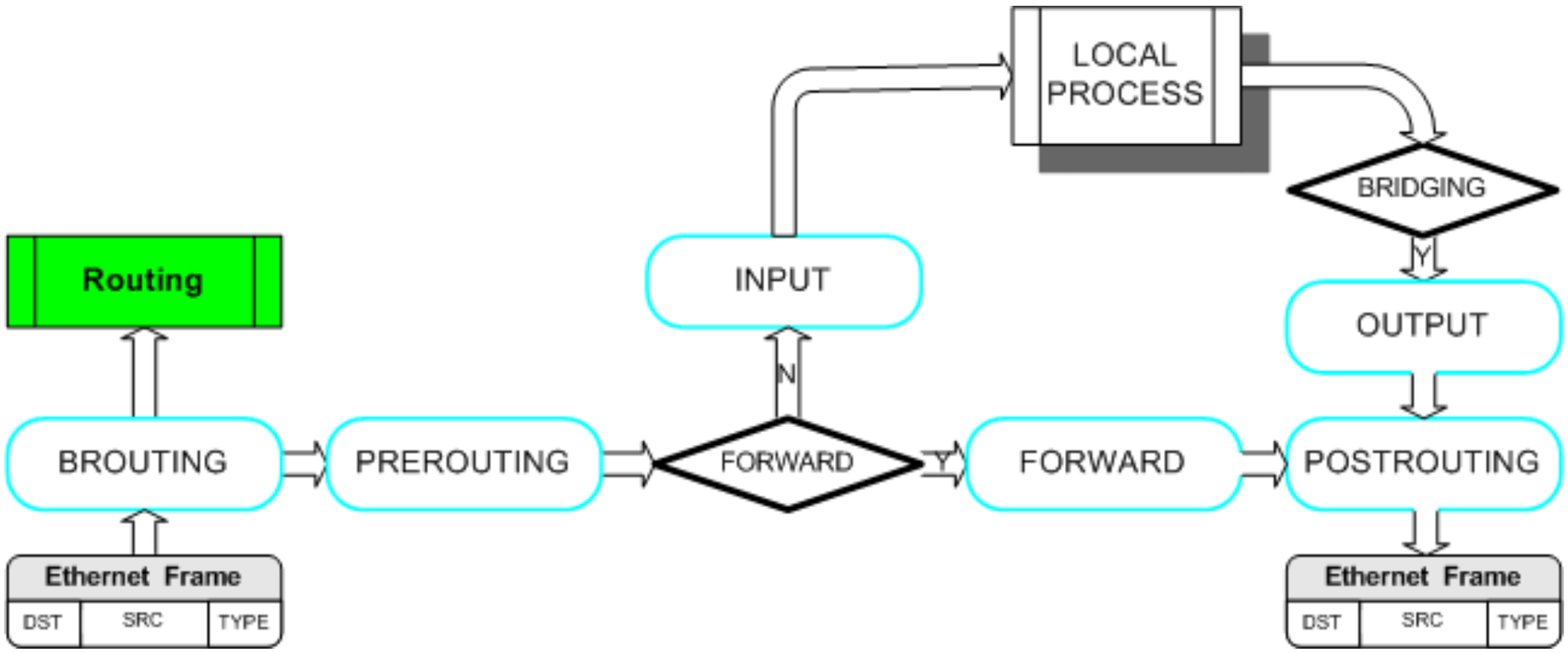
From include/linux/netdevice.h

```
struct net_device_ops {
    int     (*ndo_init)(struct net_device *dev);
    void    (*ndo_uninit)(struct net_device *dev);
    int     (*ndo_open)(struct net_device *dev);
    int     (*ndo_stop)(struct net_device *dev);
    netdev_tx_t  (*ndo_start_xmit) (struct sk_buff *skb,
                                    struct net_device *dev);
    netdev_features_t (*ndo_feature_check)(struct sk_buff *skb,
                                struct net_device *dev,
                                netdev_features_t features);
    u16     (*ndo_select_queue)(struct net_device *dev,
                                struct sk_buff *skb,
                                void *accel_priv,
                                select_queue_fallback_t fallback);
```

Receiving Packets Overview

- When a NIC detects an incoming packet, it raises an interrupt
- Kernel then jumps to the top-half of the device driver that had registered an ISR for that IRQ
- Device driver fetches data from NIC, and then allocates and fills in a `sk_buff`
- Driver then injects `sk_buff` into network stack (Linux's Netfilter subsystem)
- Netfilter has a policy that determines fate of that `sk_buff`

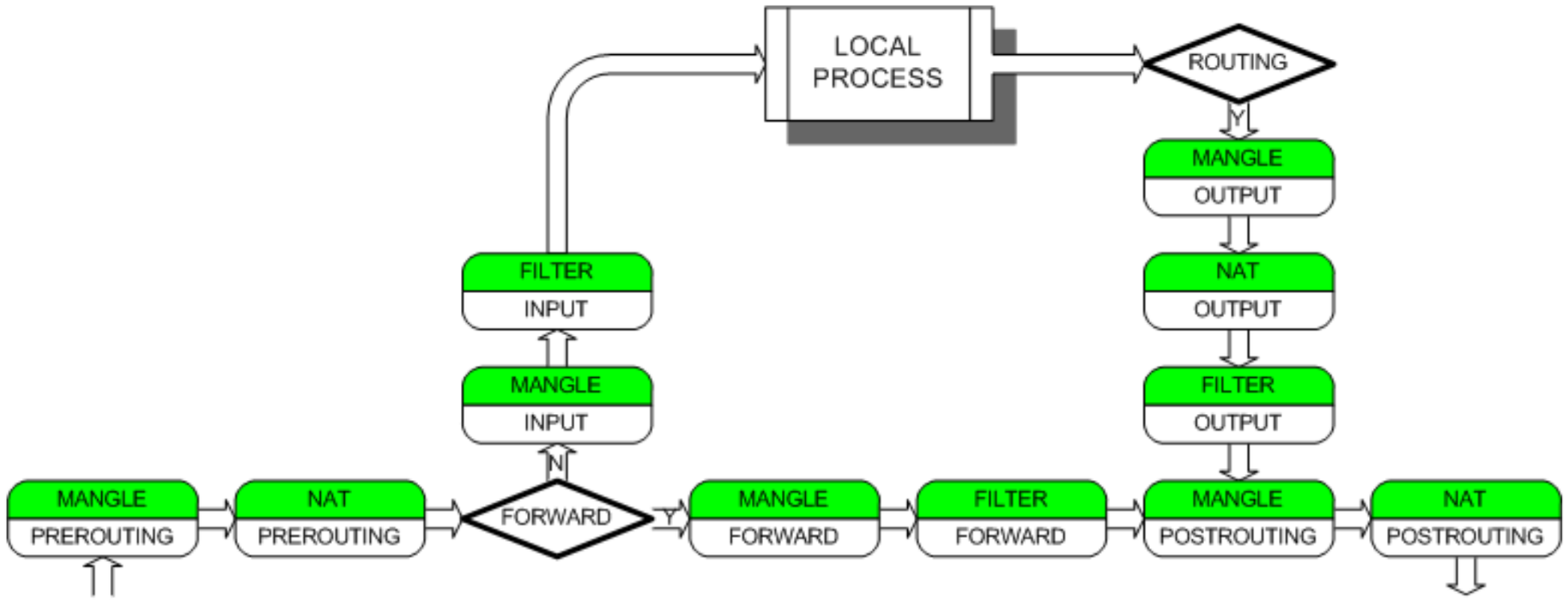
Receiving Packets Overview



- Network stack uses various rules to decide what to do with `sk_buff`
- These rules can be used to configure a router and/or a firewall
  - Each of these rules belong to a Netfilter chain

http://ebtables.netfilter.org/br_fw_ia/br_fw_ia.html

Netfilter Chains



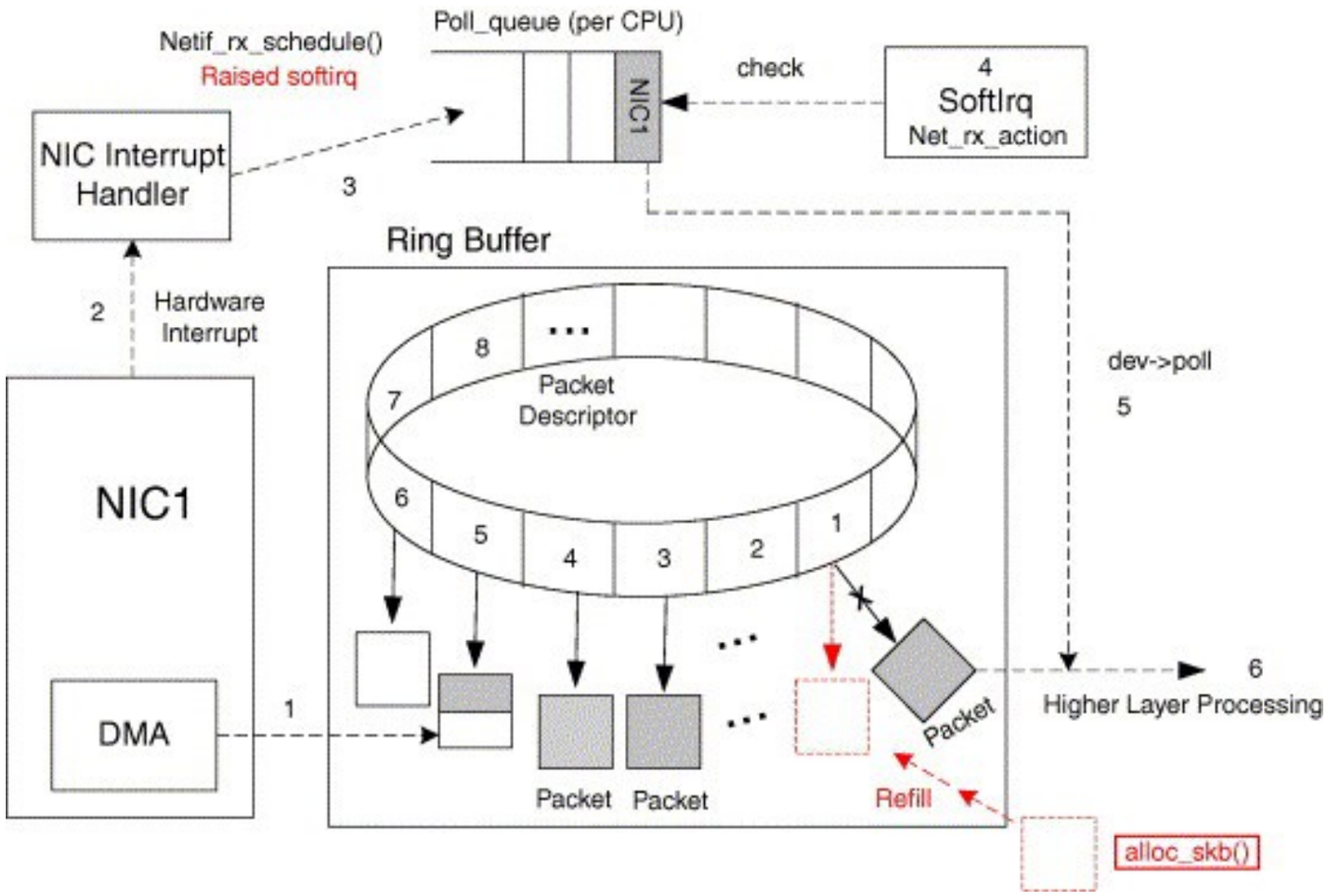
Chain	Use
INPUT	Reject or accept packet destined for this machine
OUTPUT	Modify packet prior to broadcasting out
FORWARD	Reroute packet to a different machine

http://ebtables.netfilter.org/br_fw_ia/br_fw_ia.html

Networking Hardware

- NICs have one of two ways to store incoming octets:
    - Device memory: NIC has a small amount of RAM (on the order of a few kibibytes) to store data
    - DMA: NIC writes directly to main RAM via DMA descriptors
- Either way, NICs have a ring buffer
    - Every time it receives a packet, NIC writes to next entry in buffer, replacing oldest entry (like a fixed-size queue)
    - Buffer capacity is limited, sometimes as few as 16 entries

DMA Ring Buffer



- When ring buffer is in RAM, during initialization device driver writes to device the starting address and number of entries within the ring

- Device stores incoming data in next ring buffer entry, then raises an interrupt, notifying driver that the next entry is valid

https://medium.com/@oscar.eriks/case-study-network-bottlenecks-on-linux-part-1-the-nic-c24d62826a64

Ring Buffer Issues

- Because the ring buffer has finite capacity, entries can be overridden if NIC is experiencing a packet flood (such as from a DDOS)
    - If kernel does not copy packets out of ring buffer fast enough, NIC will overwrite entries and thus those packets will be lost

- Traditionally, NIC will raise an interrupt for each incoming packet
    - Top-half copies entry into a `sk_buff` and then awakens network stack
    - Network stack runs in bottom-half
    - During a flood, bottom-half will never get a chance to run

Linux New API (NAPI)

- New-ish network design to handle heavy loads

- During initialization, device driver registers itself to be NAPI-aware, by registering a function pointer table of type `struct napi_struct`

- When driver's top-half is invoked, indicating that a packet has arrived, instead of processing the packets, the driver disables any further "packet available" interrupts and instead notify core kernel that a packet arrived

- A NAPI kthread will awake (as a bottom-half) and then polls hardware to process all pending packets

NAPI Polling

```
struct napi_struct {
    /* The poll_list must only be managed by the entity which
     * changes the state of the NAPI_STATE_SCHED bit.  This means
     * whoever atomically sets that bit can add this napi_struct
     * to the per-cpu poll_list, and whoever clears that bit
     * can remove from the list right before clearing the bit.
     */
    struct list_head poll_list;

    unsigned long state;
    int           weight;
    unsigned int  gro_count;
    int           (*poll)(struct napi_struct *, int);
```

From include/linux/netdevice.h

- `poll()` callback invoked when NAPI is ready to process packets from the `net_device`

- Second parameter passed to callback is the budget

NAPI Polling

- Budget is a limit to number of packets that driver may process
- Return value from `poll()` callback is number of packets actually processed
  - If this return value is equal to the budget, NAPI kthread will call `poll()` again
- When driver processes fewer packets than the budget, it should reenable interrupts on the NIC
  - NAPI kthread will stop calling driver's `poll()`

NAPI Polling

- With NAPI, system can continue processing packets under high load, without being interrupted constantly by NIC
    - Fewer interrupt handling means more CPU time spent processing packets
    - After bottom-half has caught up to all available packets, then driver will reenable interrupts
- Disadvantage to NAPI is increased latency when processing initial packet, and additional overhead when packets arrive slowly