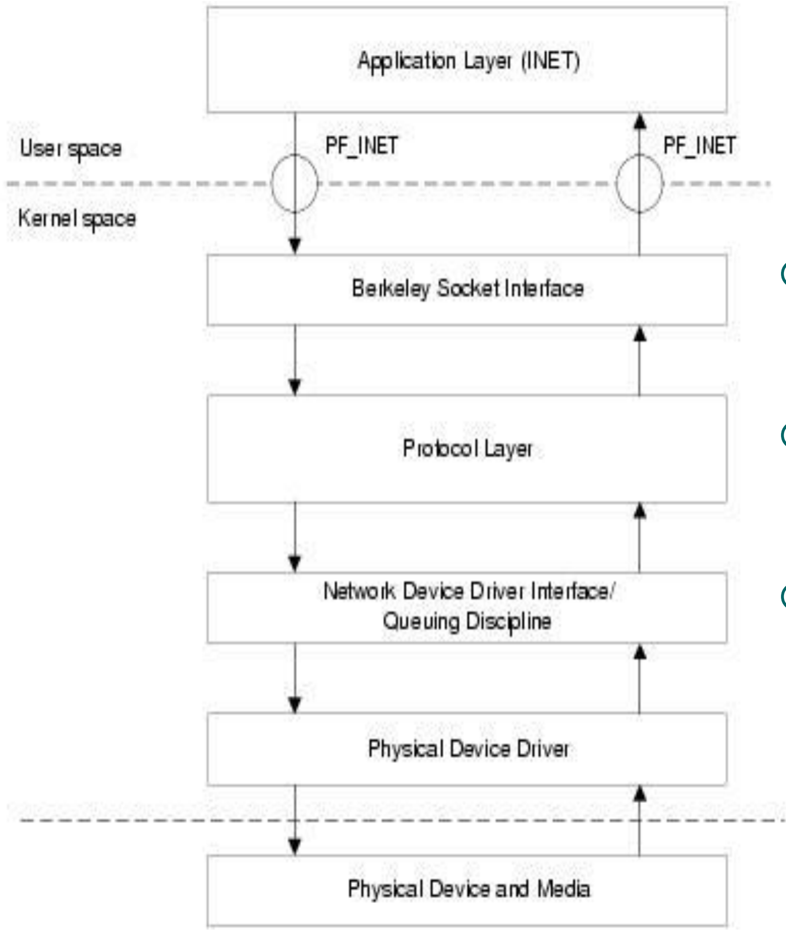
The Journey of a Packet Through the Linux Network Stack

… plus hints on Lab 9

Some Words

- Assume IP version 4
- Codes are from Kernel 2.6.9.EL (use in Lab 9)
- Ideas are similar

Linux High-Level Network Stack



- Interface to users
- TCP/UDP/IP etc...
- Queue for device

Image from http://affix.sourceforge.net/affix-doc/c190.html

Receiving a Packet (Device)

- **Network card**
  - receives a frame

issues an *interrupt*

- **Driver**
  - handles the *interrupt*

•Frame → RAM

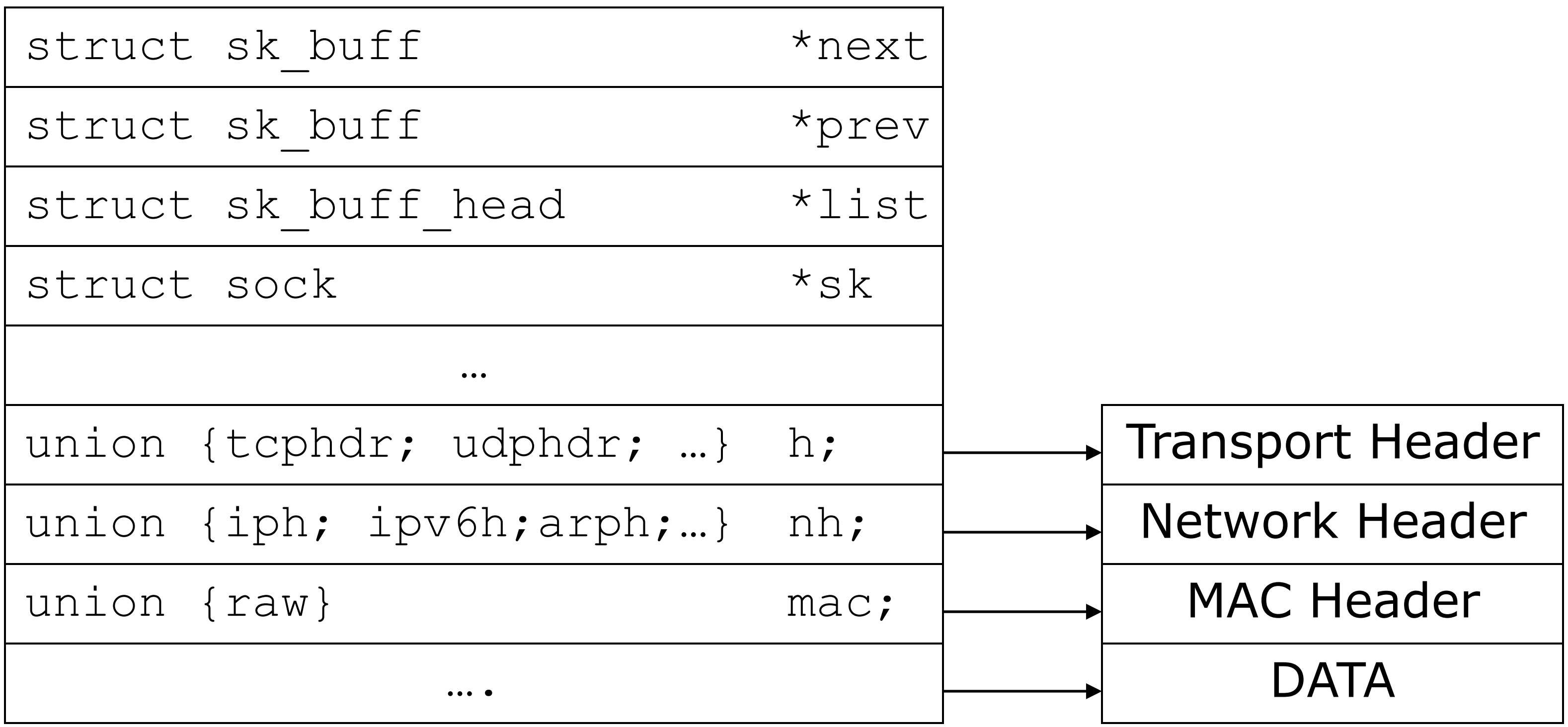
•Allocates **sk_buff** (called **skb**)

•Frame → **skb**

Aside: `sk_buff` `(skbuff.h)`

- Generic buffer for all packets
- Pointers to `skb` are passed up/down
- Can be linked together

sk_buff (cont.)


struct sk_buff *next
struct sk_buff *prev
struct sk_buff_head *list
struct sock *sk
…
union {tcphdr; udphdr; …} h;
union {iph; ipv6h;arph;…} nh;
union {raw} mac;
….
Transport Header
Network Header
MAC Header
DATA

`sk_buff` (cont.)

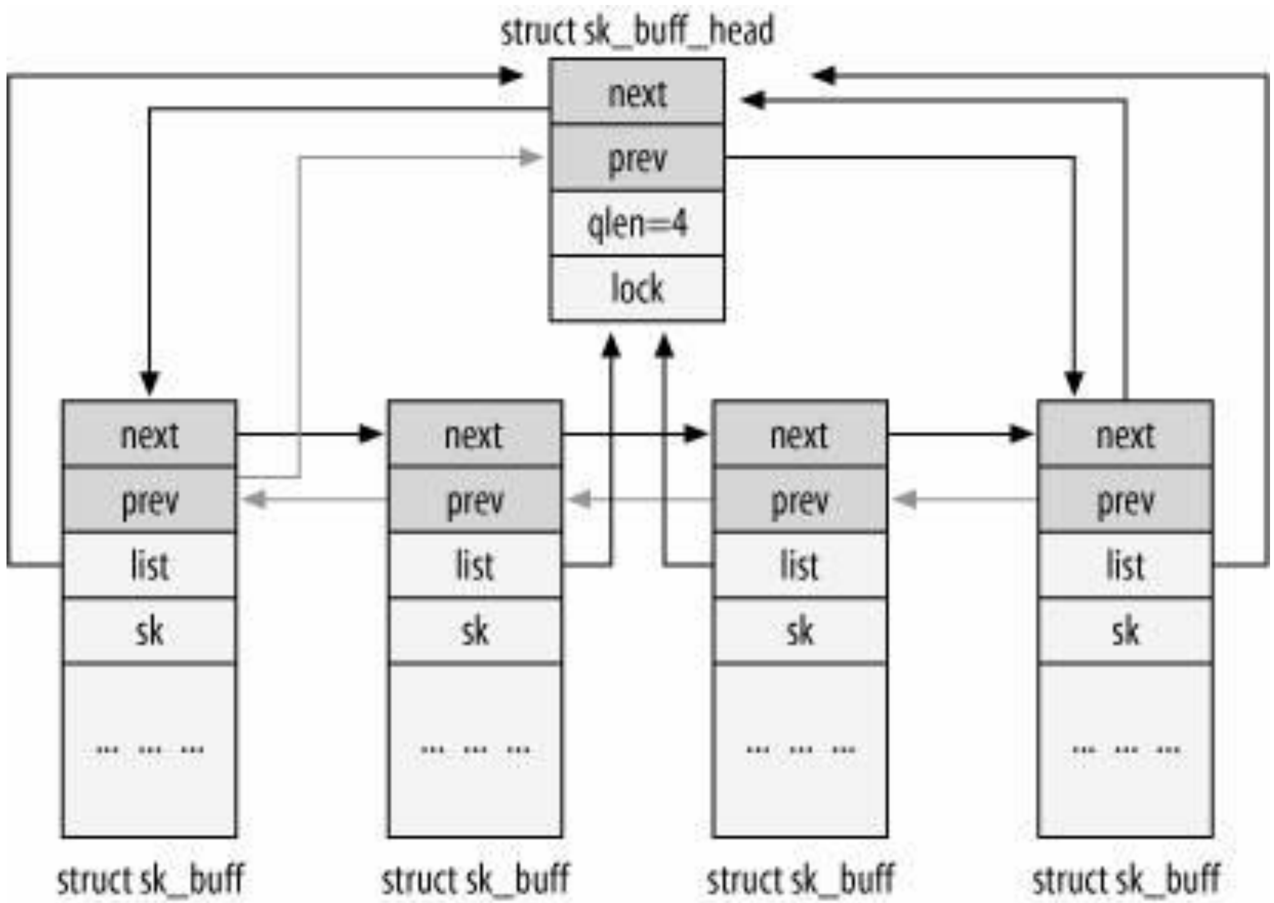


Image from “Understanding Linux Network Internals”, Christian Benvenuti

Receiving a Packet (Device)

- Driver (cont.)
  - calls device independent `core/dev.c:netif_rx(skb)`

    - puts **`skb`** into CPU queue
    - issues a "soft" interrupt

- CPU
  - calls `core/dev.c:net_rx_action()`

    - removes **`skb`** from CPU queue
    - passes to network layer e.g. ip/arp
    - In this case: IPv4 `ipv4/ip_input.c:ip_rcv()`

Receiving a Packet (IP)

- `ip_input.c:ip_rcv()`

checks

- Length >= IP Header *(20 bytes)*
- Version == 4
- Checksum
- Check length again

calls

`ip_rcv_finish()`

calls

`route.c:ip_route_input()`

Aside: Finish/Slow suffix

- Division into two stages is common
- Usually called “slow”

The first stage	cache
The second stage	table

Receiving a Packet (routing)

- **`ipv4/route.c:ip_route_input()`**

Destination == me?	
YES	**`ip_input.c:ip_local_deliver()`**
NO	Calls **`ip_route_input_slow()`**

- **`ipv4/route.c:ip_route_input_slow()`**

Can forward?	
•Forwarding enabled? •Know route?	
NO	Sends ICMP

Forwarding a Packet

- Forwarding is per-device basis
  - Receiving device!
- Enable/Disable forwarding in Linux:
  - Kernel
  - `/proc` file system ↔ Kernel
  - read/write normally (in most cases)

```
•/proc/sys/net/ipv4/conf/<device>/forwarding
•/proc/sys/net/ipv4/conf/default/forwarding
•/proc/sys/net/ipv4/ip_forwarding
```

Forwarding a Packet (cont.)

- **`ipv4/ip_forward.c:ip_forward()`**

IP TTL > 1	
YES	Decreases TTL
NO	Sends ICMP

- **`core/dev.c:dev_queue_xmit()`**
- Default queue: priority FIFO
  `sched/sch_generic.c:pfifo_fast_enqueue()`
- Others: FIFO, Stochastic Fair Queuing, etc.

Priority Based Output Scheduling

- **`pfifo_fast_enqueue()`**
- Again, per-device basis
- Queue Discipline (Qdisc: **`pkt_sched.c`**)
  - Not exactly a priority queue
  - Uses three queues (bands)
    - 0 “interactive”
    - 1 “best effort”
    - 2 “bulk”
- Priority is based on IP Type of Service (ToS)
  - Normal IP packet → 1 “best effort”

Queue Discipline: Qdisc

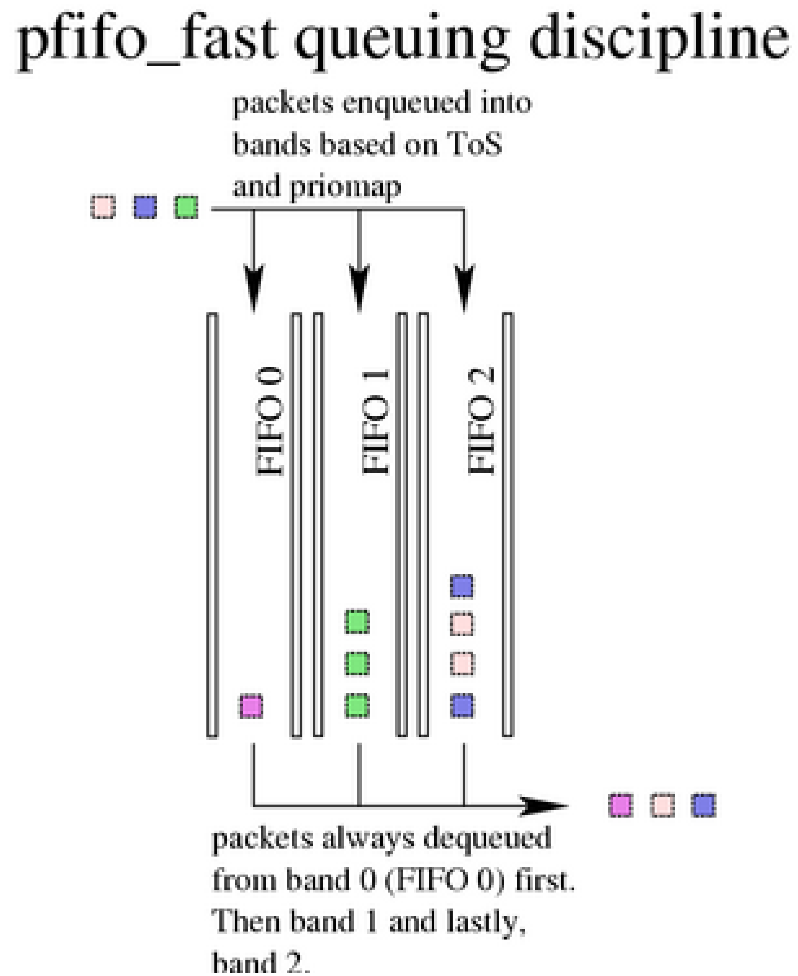


Image from http://linux-ip.net/articles/Traffic-Control-HOWTO/classless-qdisc.html

Mapping IP ToS to Queue

- IP ToS: PPPDTRCX
  - PPP → Precedence
    - Linux = ignore!
    - Cisco = Policy-Based Routing (PBR)
  - D → Minimizes Delay
  - T → Maximizes Throughput
  - R → Maximizes Reliability
  - C → Minimizes Cost
  - X → Reserved

Mapping IP ToS to Queue (cont.)

IP ToS	Band
0x0	**1**
0x2	**2**
0x4	**2**
0x6	2
0x8	**1**
0xA	2
0xC	**0**
0xE	0
0x10	1
0x12	1
0x14	1
0x16	1
0x18	1
0x1A	1
0x1C	1
0x1E	1

- **`pfifo_fast_enqueue()`** maps IP ToS to one of three queues
- IP ToS: PPPDTRCX
- Mapping array: prior2band

Queue Selection

sch_generic.c

Mapping array

```c

static const u8 prio2band[TC_PRIO_MAX+1] =
{ 1, 2, 2, 2, 1, 2, 0, 0 , 1, 1, 1, 1, 1, 1, 1, 1 };

/* 3-band FIFO queue: old style, but should be a bit faster than
   generic prio+fifo combination.
 */

static int
pfifo_fast_enqueue(struct sk_buff *skb, struct Qdisc* qdisc)
{
  struct sk_buff_head *list;

  list = ((struct sk_buff_head*)qdisc->data) +
    prio2band[skb->priority&TC_PRIO_MAX];

```

Band “0” (first in Qdisc)

Change band

Queue Selection (cont.)

- Kernel 2.6.9.EL

`Qdisc`

…
`sk_buff_head band 0`
`sk_buff_head band 1`
`sk_buff_head band 2`
…

list = ((struct sk_buff_head*)qdisc→data +prior2band[skb->priority&TC_PRIOR_MAX]

Sending Out a Packet

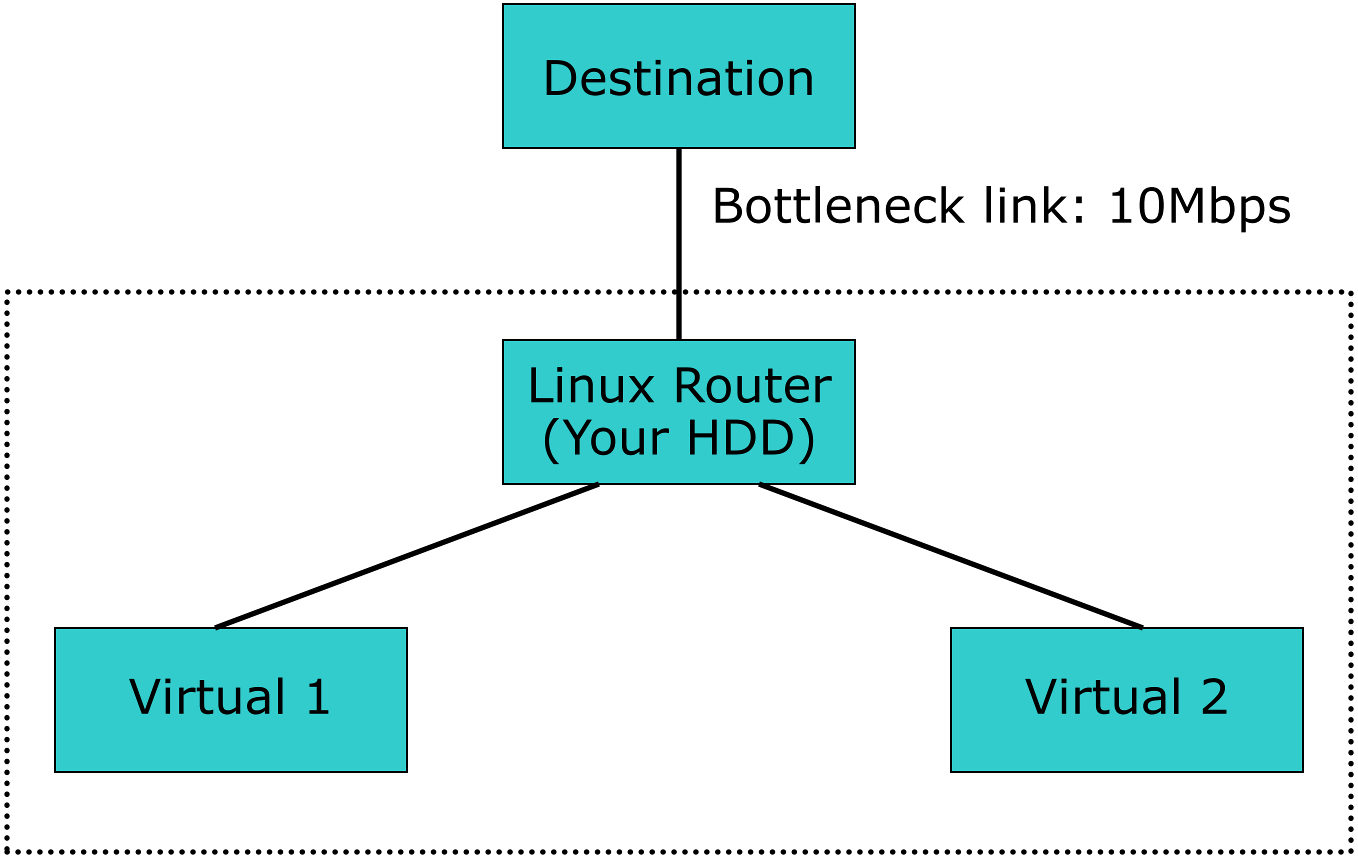
- **`pfifo_fast_dequeue()`**
  - Removes the oldest packet from the highest priority band
  - The packet that was just enqueued!
  - Passes it to the device driver

Lab 9

Scenarios, hints, etc.

Lab 9 Part 1&2

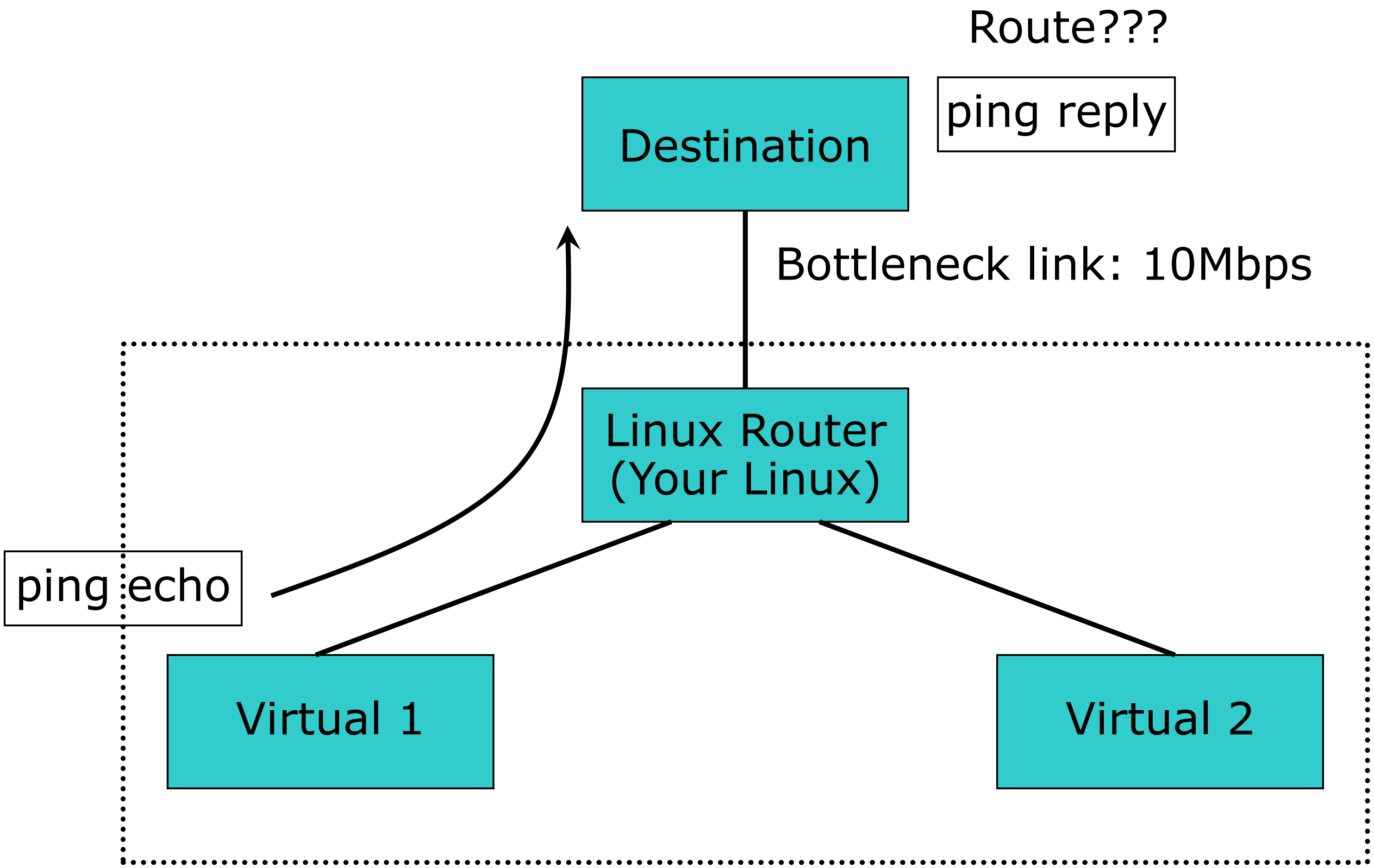
- Scenario



Lab 9 Part 2

- Default: no IP forwarding
  - Enable it! **`/proc/`**…
- Only one router
- Default route on “destination”

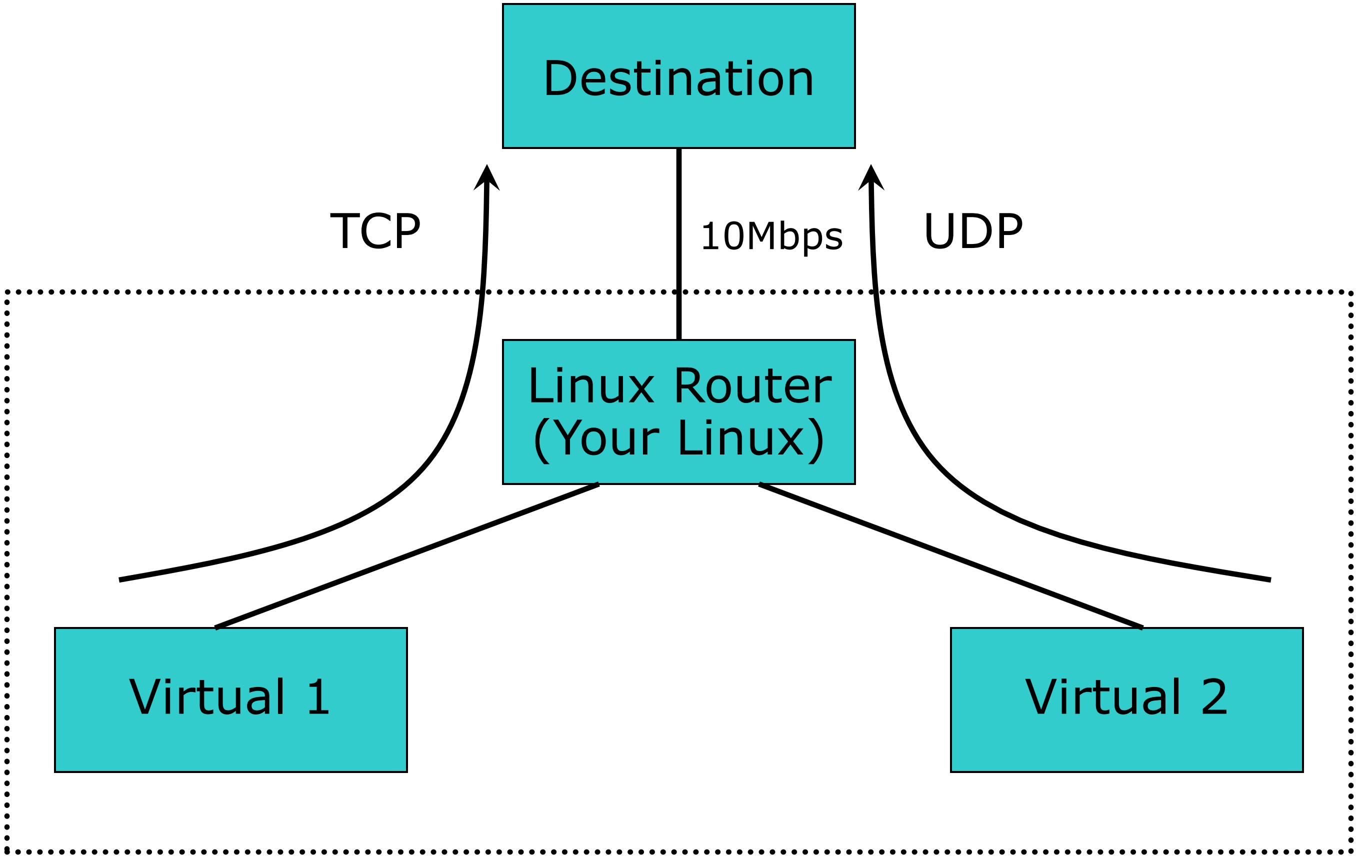
Lab 9 Part 2



Route???
ping reply
Destination
Bottleneck link: 10Mbps
Linux Router
(Your Linux)
ping echo
Virtual 1
Virtual 2

Lab 9 Part 3

- Scenario



Lab 9 Part 3 (cont.)

- Problem with TCP v.s. UDP?
- TCP is too “nice”
- Proposed solution:

  Modify kernel TCP → higher priority

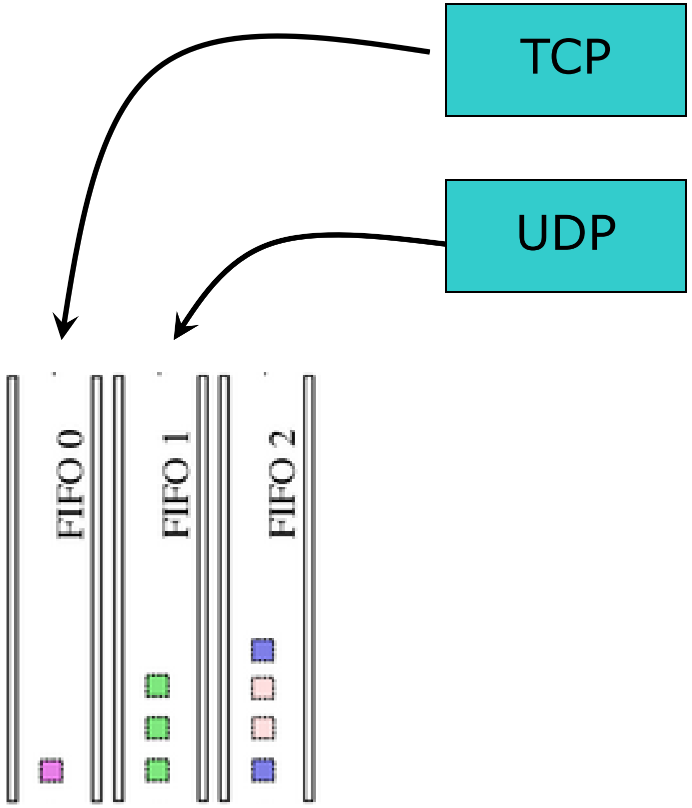
Lab 9 Part 4

- Goal: compile the modified kernel
- Print out TCP/UDP when sending or forwarding a packet
- **`/proc/sys/kernel/printk`**
- Start up with the new kernel!
  - Press any key on boot → OS list
  - Select 2.6.9

Lab 9 Part 5

- Goal: change the kernel scheduling
- Idea: place TCP in the higher priority band
- **`pfifo_fast_enqueue()`**
  - Default → IP ToS
  - Change it to TCP v.s. UDP (+others)
  - Options: UDP++ or TCP--
  - Do NOT change IP ToS!

Lab 9 Part 5 (cont.)



Lab 9 Part 5 (cont.)

```

static const u8 prio2band[TC_PRIO_MAX+1] =
{ 1, 2, 2, 2, 1, 2, 0, 0 , 1, 1, 1, 1, 1, 1, 1, 1 };

/* 3-band FIFO queue: old style, but should be a bit faster than
   generic prio+fifo combination.
 */

static int
pfifo_fast_enqueue(struct sk_buff *skb, struct Qdisc* qdisc)
{
  struct sk_buff_head *list;

  list = ((struct sk_buff_head*)qdisc->data) +
    prio2band[skb->priority&TC_PRIO_MAX];

```

Lab 9 Part 5 (cont.)

- Remember: take **`printk()`** out!
  boot into 2.6.9
  enable forwarding
- What should happen?
- Different from Part 2?

Interesting Links

Linux Networking

- http://www.linuxfoundation.org/collaborate/workgroups/networking/kernelflow
- http://ftp.gnumonks.org/pub/doc/packet-journey-2.4.html
- Understanding Linux Network Internals, Christian Benvenuti
- http://lartc.org/lartc.html

Queue Discipline

- http://linux-ip.net/articles/Traffic-Control-HOWTO/classless-qdiscs.html

/proc file system

- http://www.gentoo.org/doc/en/security/security-handbook.xml?part=1&chap=9
- http://oatv.com/pub/a/linux/2000/11/16/LinuxAdmin.html