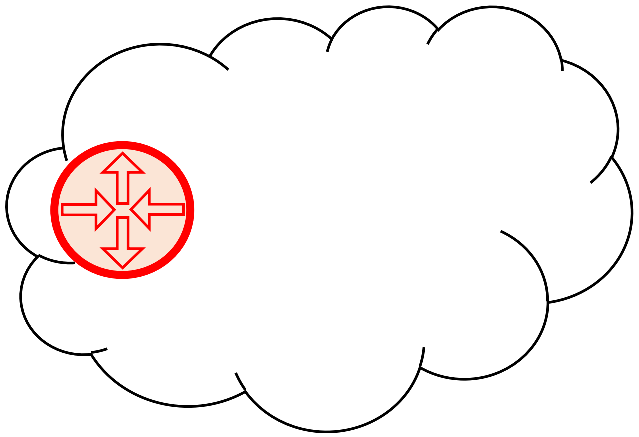
Problem / Overview

Course: Networking Principles in Practice – Linux Networking
Module: Creating a Gateway with Linux

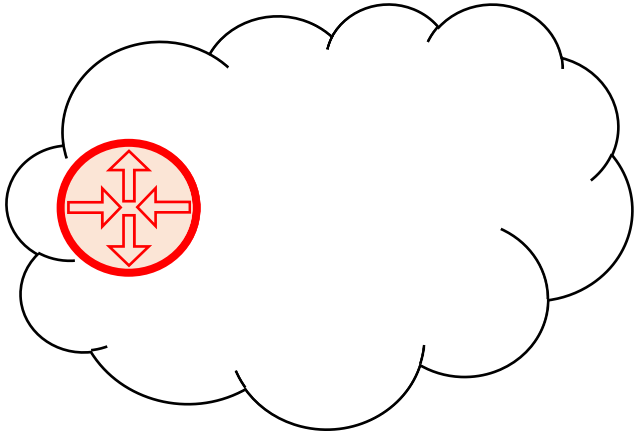
University of Colorado Boulder

What is a Gateway

- We’ll define it as something that sits at the edge of a network and provides some functionality beyond routing and forwarding.
  - May be public Internet / private network
  - May be two segments of a network under the same admin

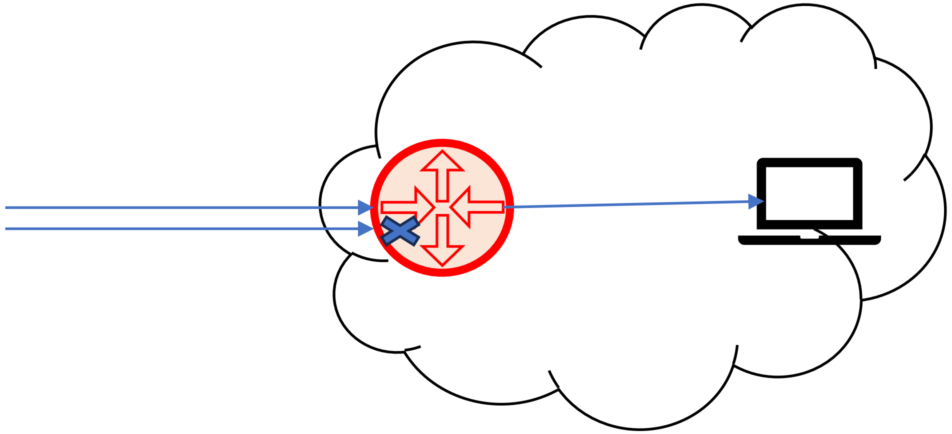


What Functionality



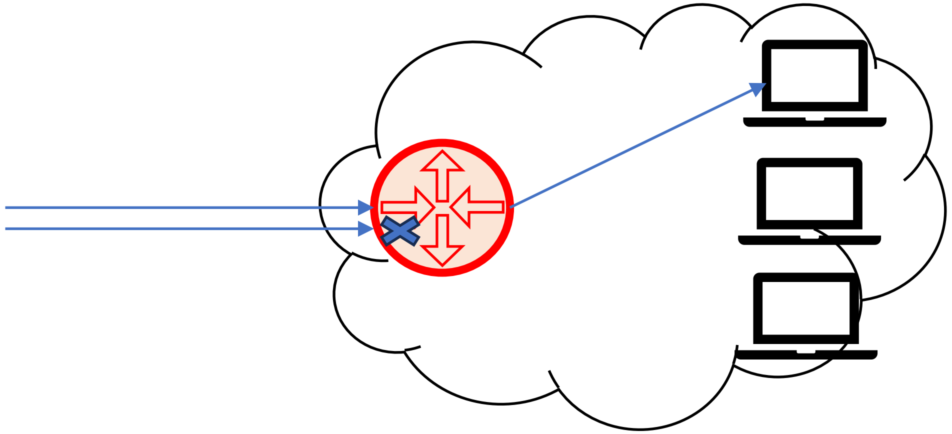
What Functionality

- Filtering / Address Translation – have a policy for what traffic is allowed in and out, and possibly translating addresses



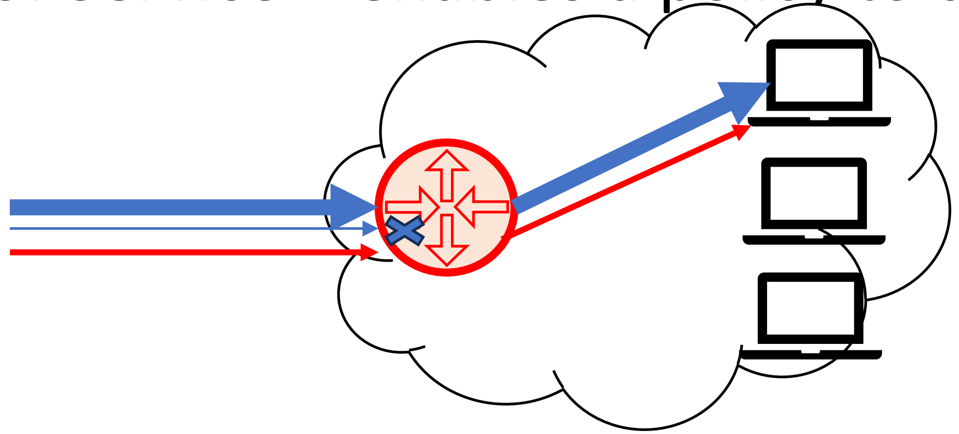
What Functionality

- Filtering / Address Translation – have a policy for what traffic is allowed in and out, and possibly translating addresses
- Load balance – say you're running a replicated service, load balancing enables forwarding to the backend servers algorithmically



What Functionality

- Filtering / Address Translation – have a policy for what traffic is allowed in and out, and possibly translating addresses
- Load balance – say you're running a replicated service, load balancing enables forwarding to the backend servers algorithmically
- Quality of Service – enables a policy to describe limits and priority on traffic



Linux covers them all

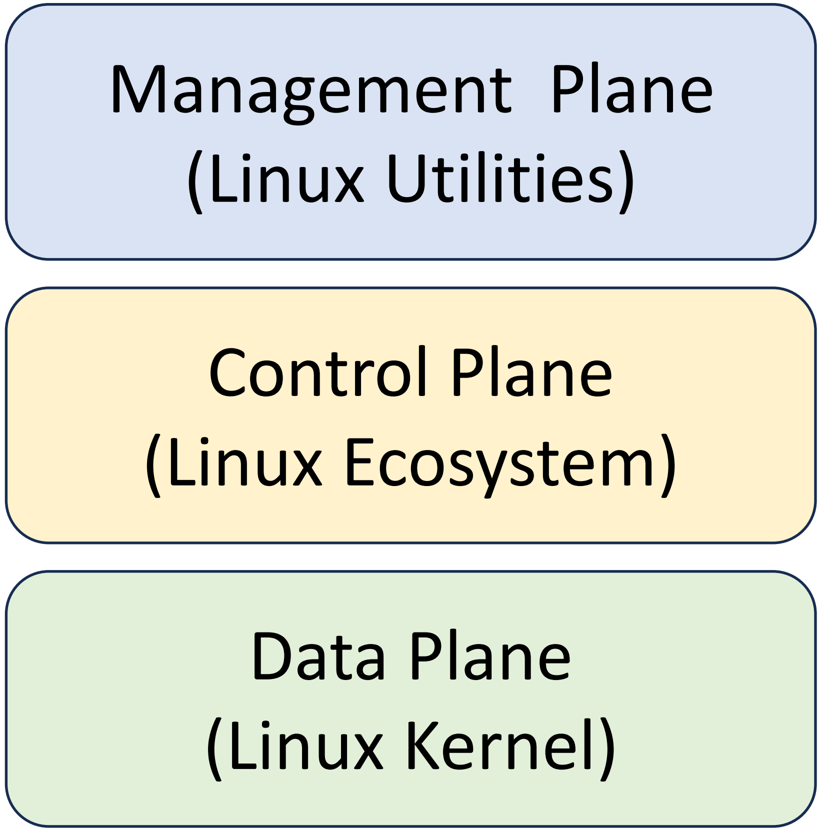
- Filtering – iptables
- Load balancing – ipvs
- QoS – tc

For each, we will cover:

- Foundational background
- Overview of the Linux utility

And provide:

- Practice exercises in a github repo



Management Plane
(Linux Utilities)

Control Plane
(Linux Ecosystem)

Data Plane
(Linux Kernel)

University of Colorado **Boulder**

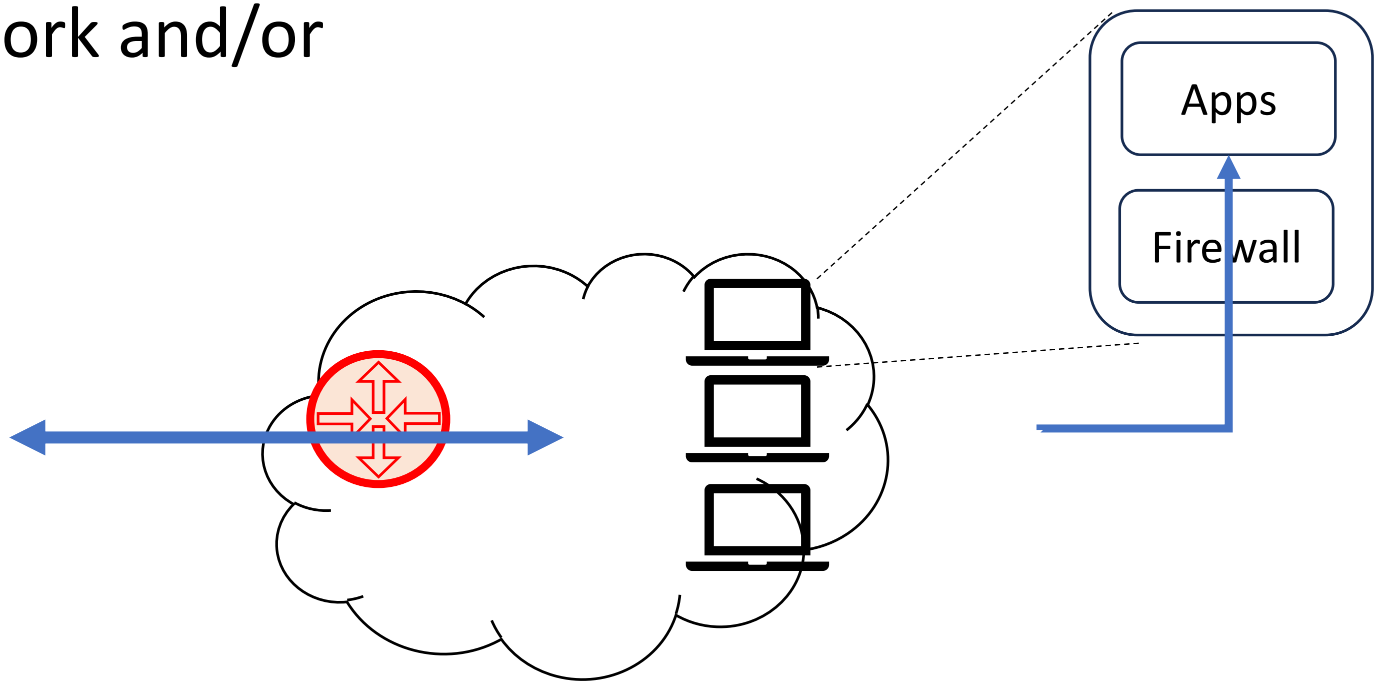
Filtering / Address Translation

Course: Networking Principles in Practice – Linux Networking
Module: Creating a Gateway with Linux

University of Colorado Boulder

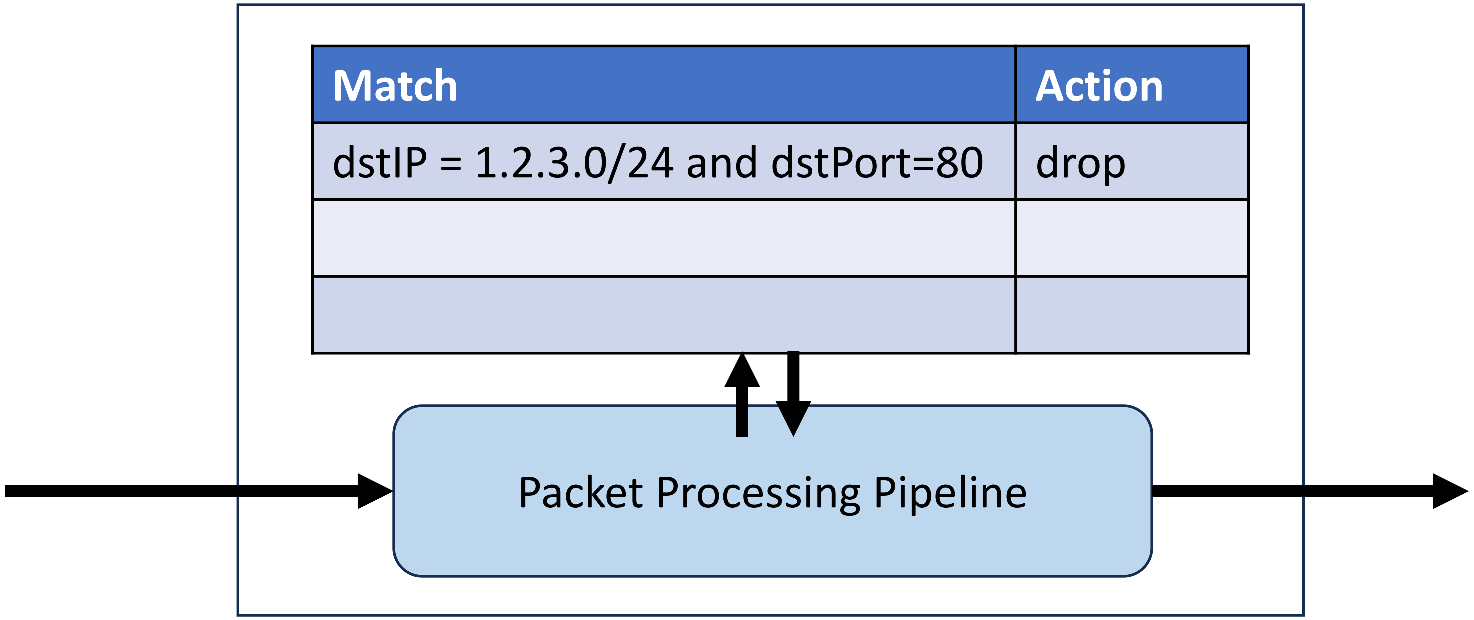
Filtering

- Want to block/allow traffic according to some policy
  - Web server: block all traffic not for port 80
  - Home gateway: block all traffic except for connections initiated from internal
- Can be at edge of some network and/or
  On the end hosts themselves



Match-Actions Tables

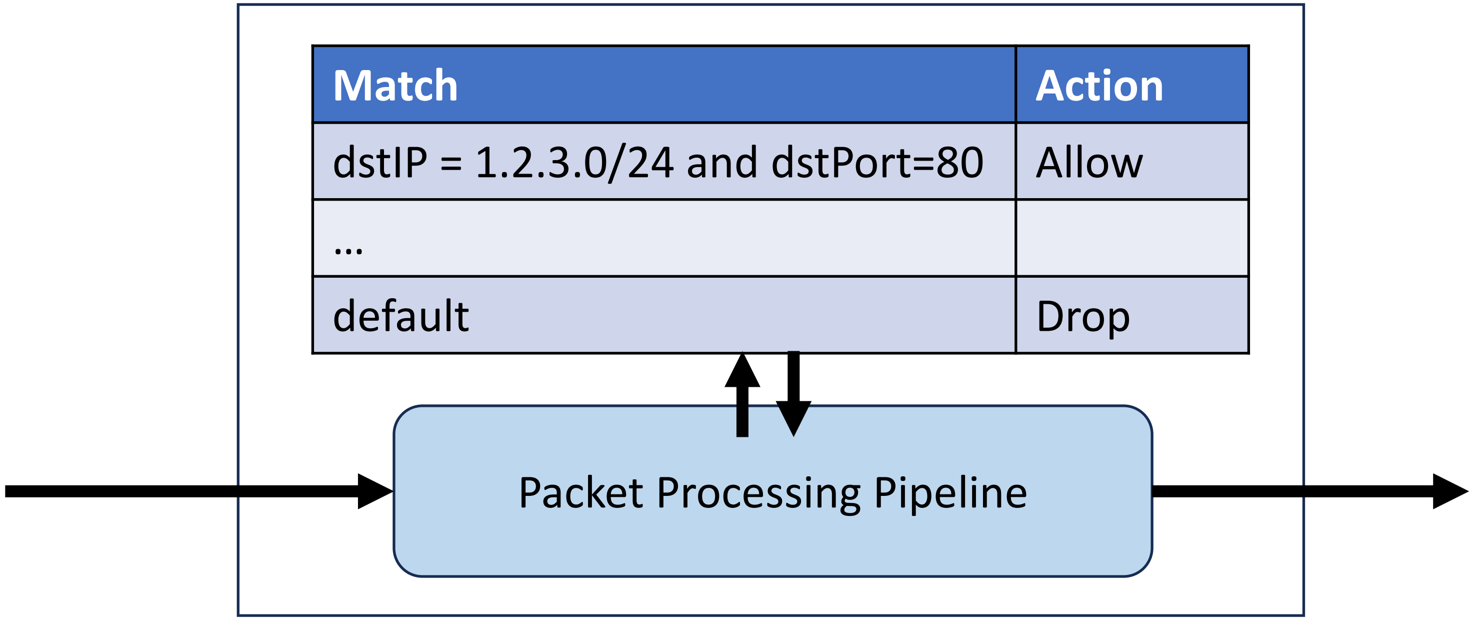
- Match - compare input packets to the rule to see if there’s a match
  - e.g., dstIP = 1.2.3.0/24 and dstPort=80
- Action – what to do with the packet if there is a match
  - E.g., drop, allow



Match	Action
dstIP = 1.2.3.0/24 and dstPort=80	drop

Good Practice – Default Drop

- Set the default action to drop
- Add rules to have an action other than drop

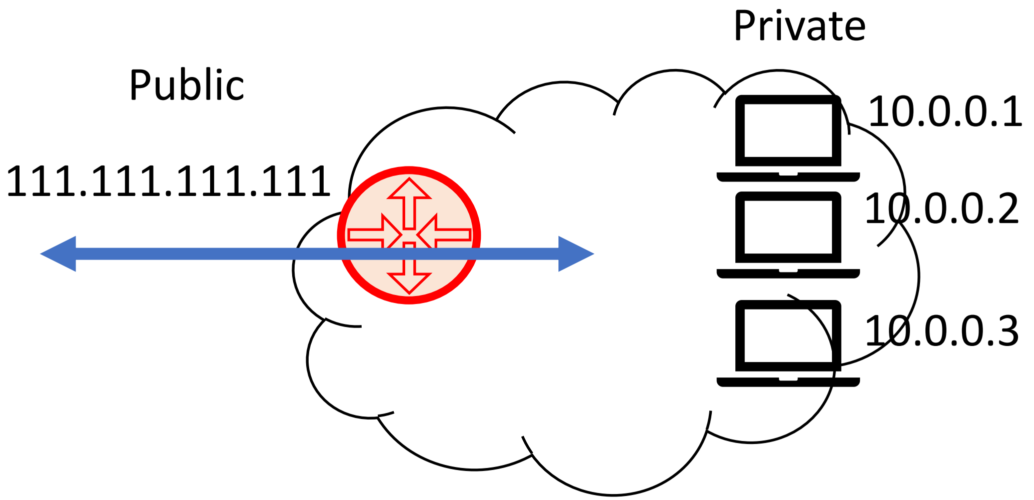


Match	Action
dstIP = 1.2.3.0/24 and dstPort=80	Allow
…	
default	Drop

Packet Processing Pipeline

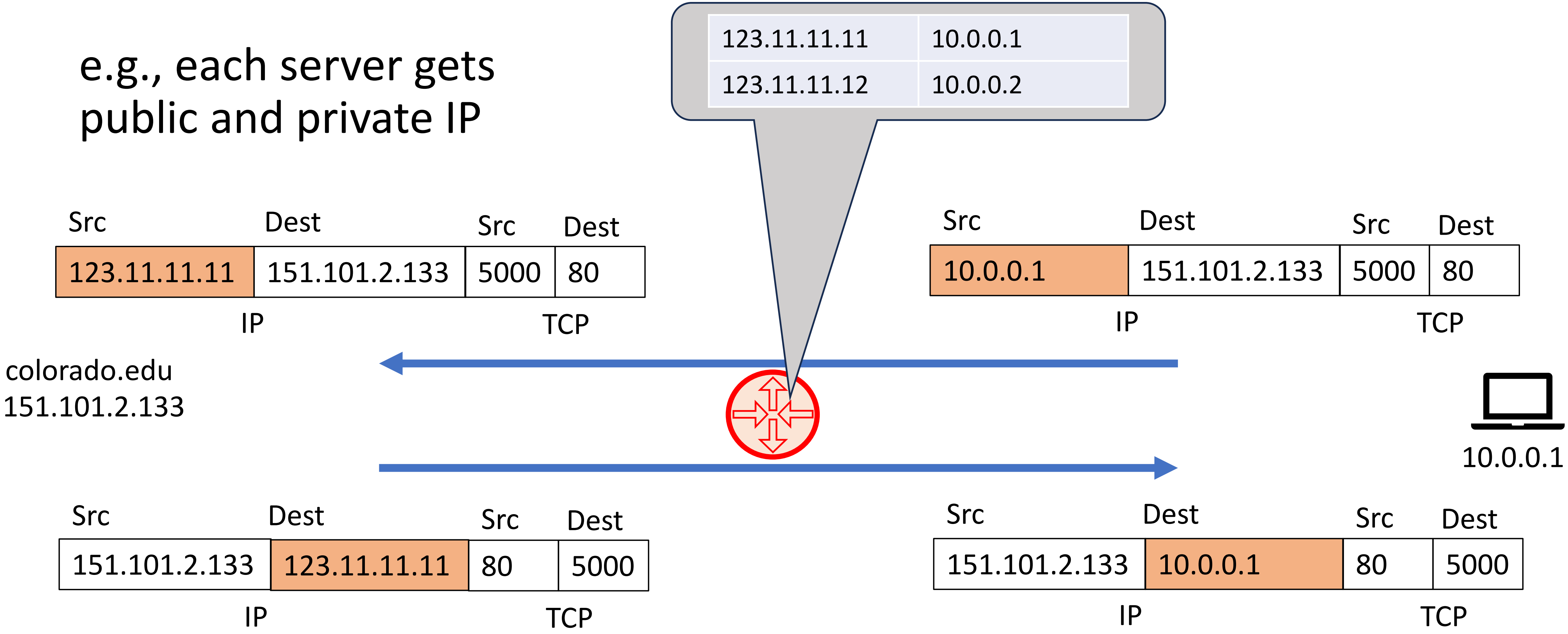
Address translation

- Actions can also manipulate the address
  - source/destination IP addresses
  - source/destination ports
- Example: use private addressing on the LAN, with a single shared public IP address.
  - Packets destined to public address are translated to a private one



Use Case 1: Static Mapping - Internal to External

e.g., each server gets
public and private IP



123.11.11.11	10.0.0.1
123.11.11.12	10.0.0.2

Src	Dest	Src	Dest
123.11.11.11	151.101.2.133	5000	80
IP		TCP	

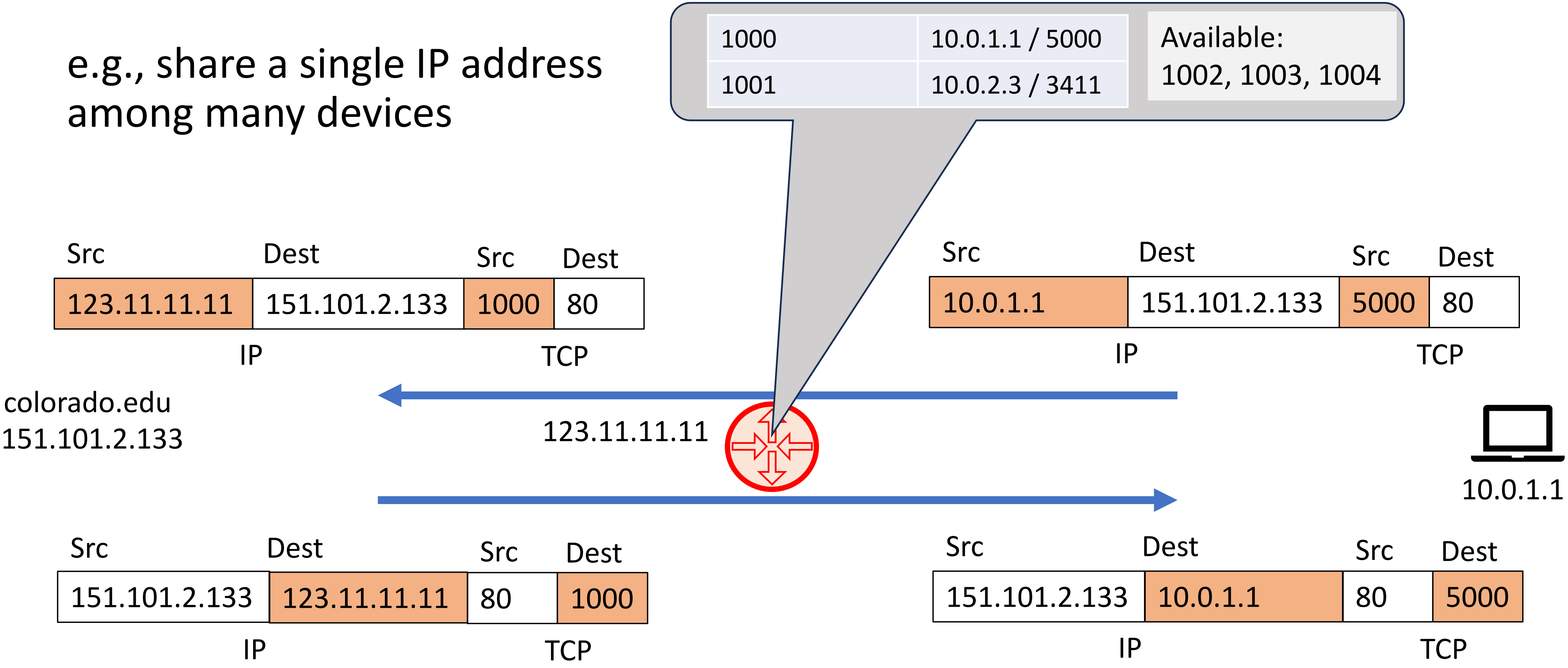
Src	Dest	Src	Dest
10.0.0.1	151.101.2.133	5000	80
IP		TCP	

Src	Dest	Src	Dest
151.101.2.133	123.11.11.11	80	5000
IP		TCP	

Src	Dest	Src	Dest
151.101.2.133	10.0.0.1	80	5000
IP		TCP	

Use Case 2: Dynamic Mapping Single External IP (dynamically select from available TCP ports)

e.g., share a single IP address among many devices



1000	10.0.1.1 / 5000
1001	10.0.2.3 / 3411

Available:
1002, 1003, 1004

Src	Dest	Src	Dest
123.11.11.11	151.101.2.133	1000	80
IP		TCP	

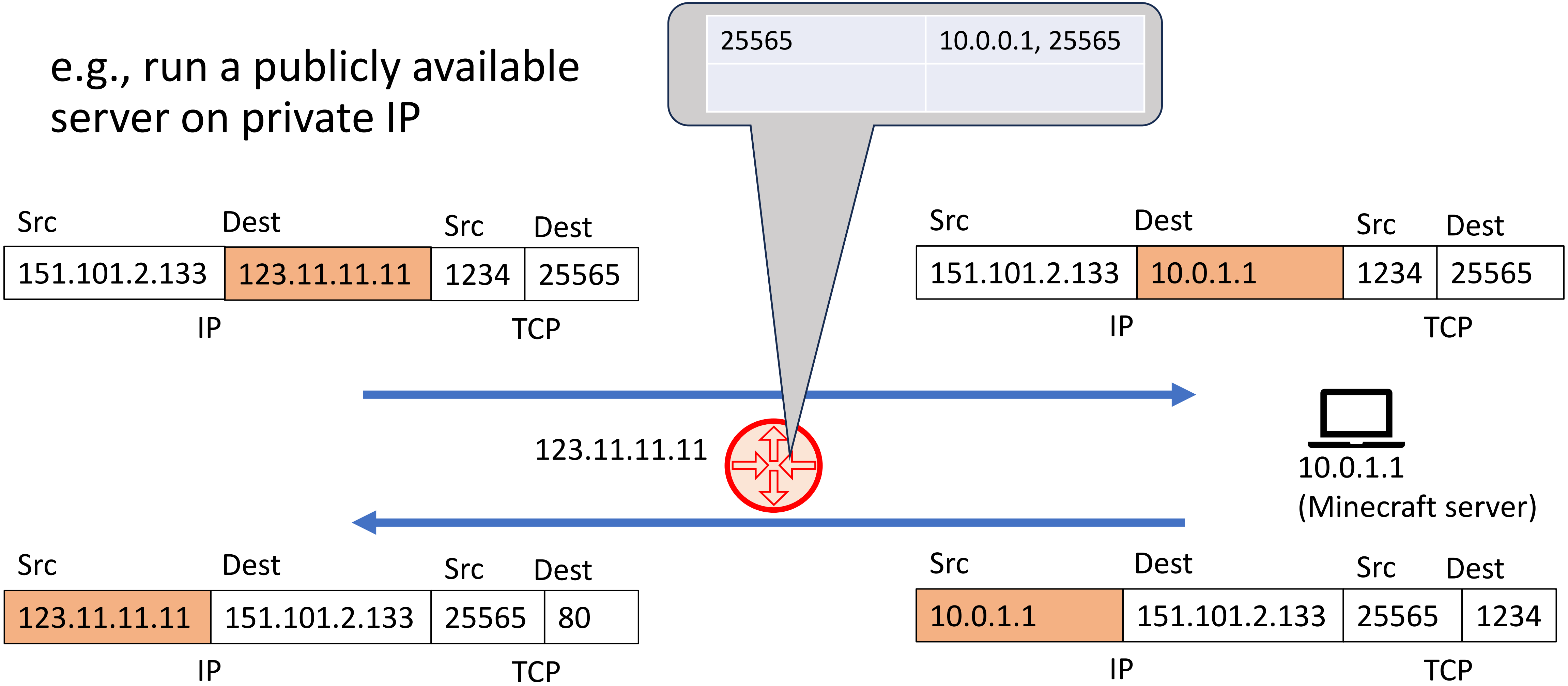
Src	Dest	Src	Dest
10.0.1.1	151.101.2.133	5000	80
IP		TCP	

Src	Dest	Src	Dest
151.101.2.133	123.11.11.11	80	1000
IP		TCP	

Src	Dest	Src	Dest
151.101.2.133	10.0.1.1	80	5000
IP		TCP	

Use Case 3: Port Forwarding

e.g., run a publicly available server on private IP



25565	10.0.0.1, 25565

Src	Dest	Src	Dest
151.101.2.133	123.11.11.11	1234	25565
IP		TCP	

Src	Dest	Src	Dest
151.101.2.133	10.0.1.1	1234	25565
IP		TCP	

Src	Dest	Src	Dest
123.11.11.11	151.101.2.133	25565	80
IP		TCP	

Src	Dest	Src	Dest
10.0.1.1	151.101.2.133	25565	1234
IP		TCP	

University of Colorado **Boulder**

iptables

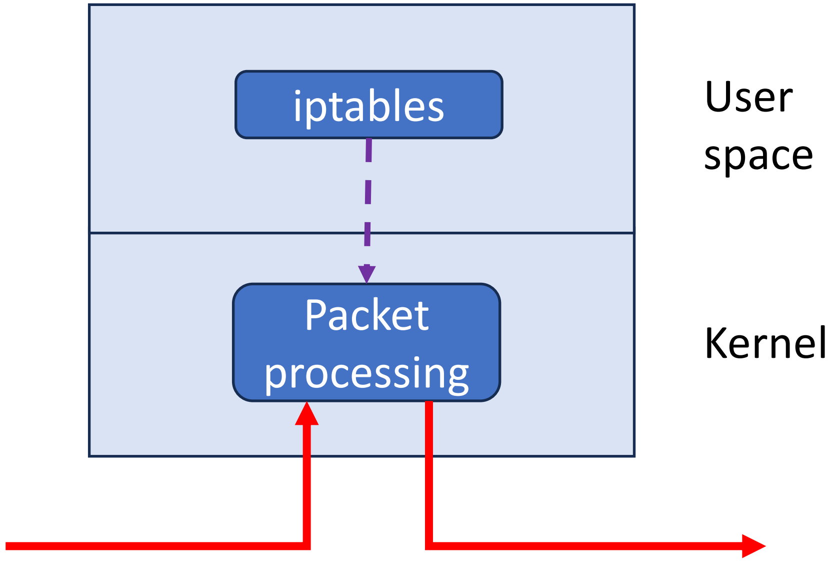
Course: Networking Principles in Practice – Linux Networking
Module: Creating a Gateway with Linux

University of Colorado Boulder

Linux netfilter framework (https://www.netfilter.org/)

User space utilities (iptables, nftables) that can configure the Linux kernel's filtering framework

netfilter
firewalling, NAT, and packet mangling for linux



Examples

```
iptables -A FORWARD -m state --state ESTABLISHED,RELATED -j ACCEPT

iptables -A INPUT -p tcp --dport 80 -j DROP

iptables -A INPUT -i eth1 -s 10.0.0.0/8 -j LOG --log-prefix "IP_SPOOF A: "
```

Tables, Chains, Rules

From: https://linux.die.net/man/8/iptables

iptables is used to set up, maintain, and inspect the tables of IP packet filter rules in the Linux kernel. Several different tables may be defined.

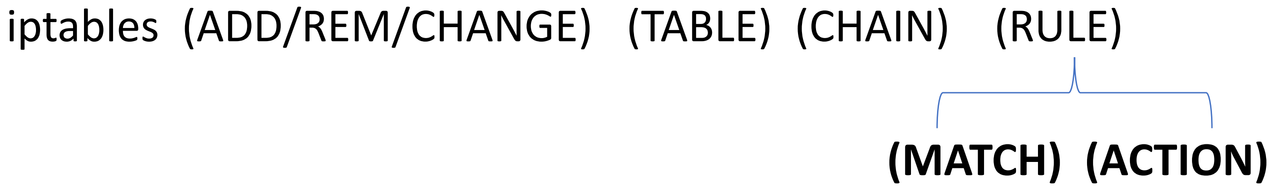
Each table contains a number of built-in chains and may also contain user-defined chains.

Each chain is a list of rules which can match a set of packets.

```
iptables  (ADD/REM/CHANGE)   (TABLE)   (CHAIN)  (RULE)
```

Rules

- Match on some criteria
- Take some action



iptables (ADD/REM/CHANGE) (TABLE) (CHAIN) (RULE)

(MATCH) (ACTION)

Rules: Matching - Basic

-p, --protocol [!] *protocol*
-s, --source [!] *address*[/*mask*]
-d, --destination [!] *address*[/*mask*]
-i, --in-interface [!] *name*
-o, --out-interface [!] *name*

Rules: Matching - Extensions

- If -p or --protocol is used, protocol specific extensions get loaded.
  - tcp
    --destination-port [!] *port*[:*port*]
    -p tcp --destination-port 8080

    --tcp-flags [!] *mask comp*
    -p tcp --tcp-flags SYN,ACK,FIN,RST SYN

- If -m [module] or --match [module] is used, match extensions get loaded
  - connlimit
    -m connlimit --connlimit-above 16 --connlimit-mask 24
  - conntrack
    -m conntrack --ctstate ESTABLISHED,RELATED

Rules: Actions

- Jump to a target
  -j, --jump *target*

- Default: ACCEPT, DROP (RETURN, QUEUE)
  -j DROP

Rules: Actions - Extensions

- REJECT
  -j REJECT --reject-with icmp-port-unreachable
- *LOG*
  -j LOG --log-level 4 --log-prefix ’IPTABLES LOG:’
- SET
- SNAT
- DNAT
- MASQUERADE

Reminder: Tables, Chains, Rules

From: https://linux.die.net/man/8/iptables

iptables is used to set up, maintain, and inspect the tables of IP packet filter rules in the Linux kernel. Several different tables may be defined.

Each table contains a number of built-in chains and may also contain user-defined chains.

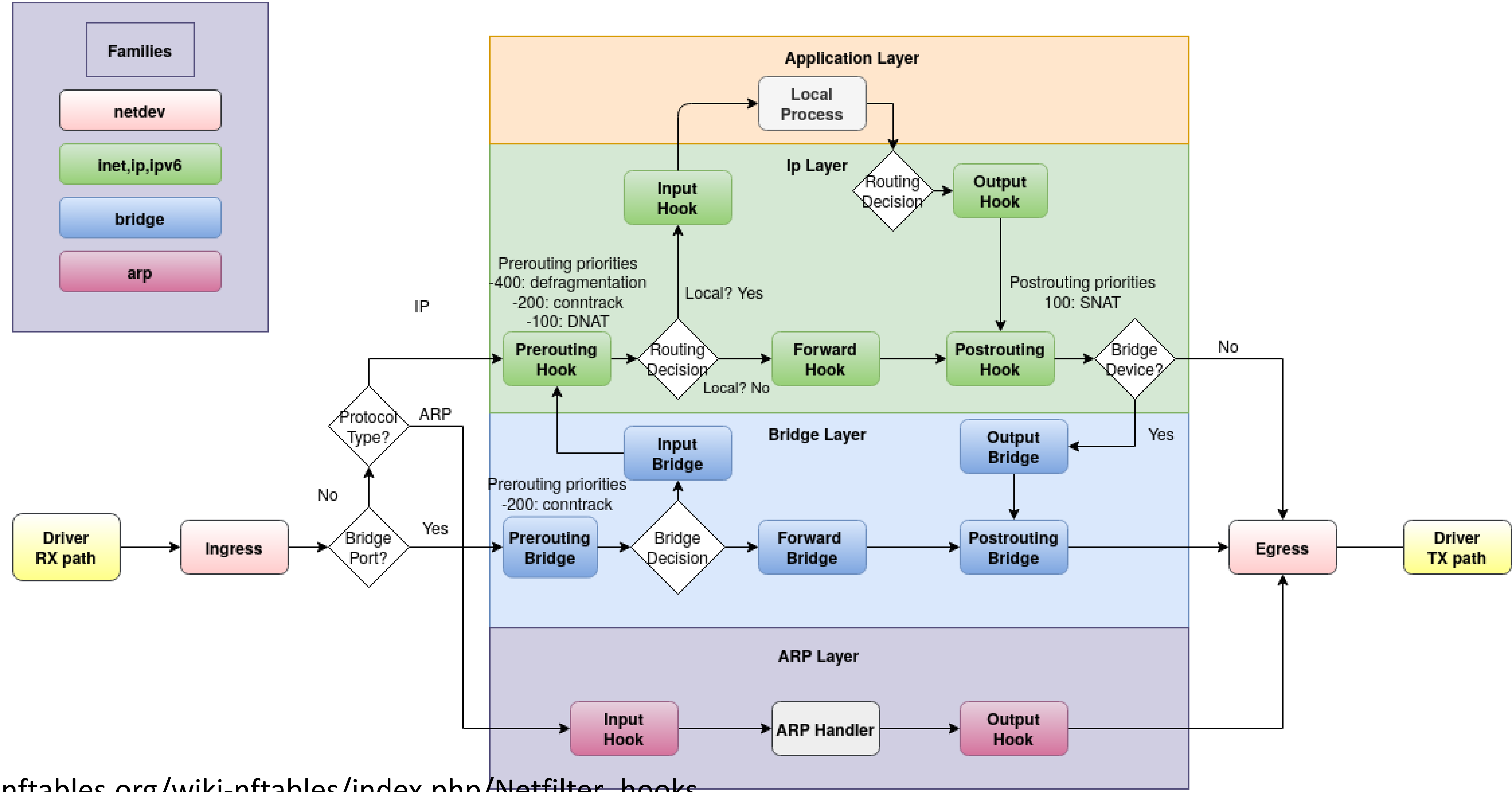
Each chain is a list of rules which can match a set of packets.

```
iptables  (ADD/REM/CHANGE)   (TABLE)   (CHAIN)  (RULE)
```

Tables – 4 defined Tables

- filter – default table
- nat - This table is consulted when a packet that creates a new connection is encountered.
- mangle -used for specialized packet alteration
- raw - used mainly for configuring exemptions from connection tracking

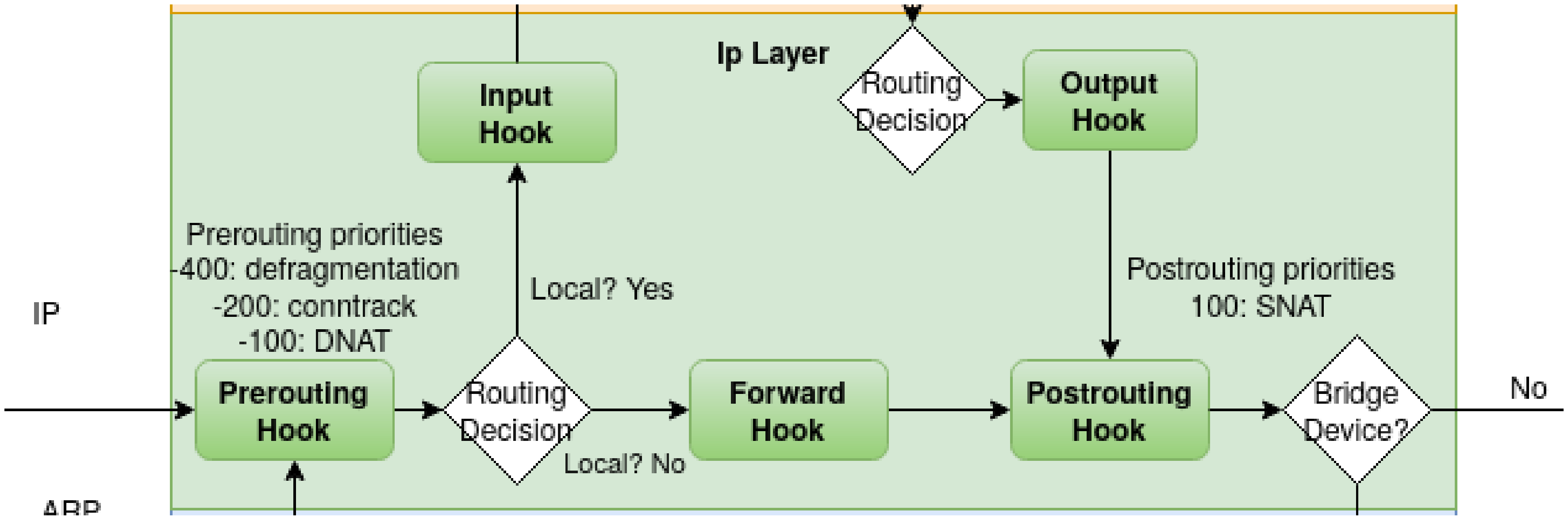
Packet Traversal in Kernel



https://wiki.nftables.org/wiki-nftables/index.php/Netfilter_hooks

Packet Traversal in Kernel

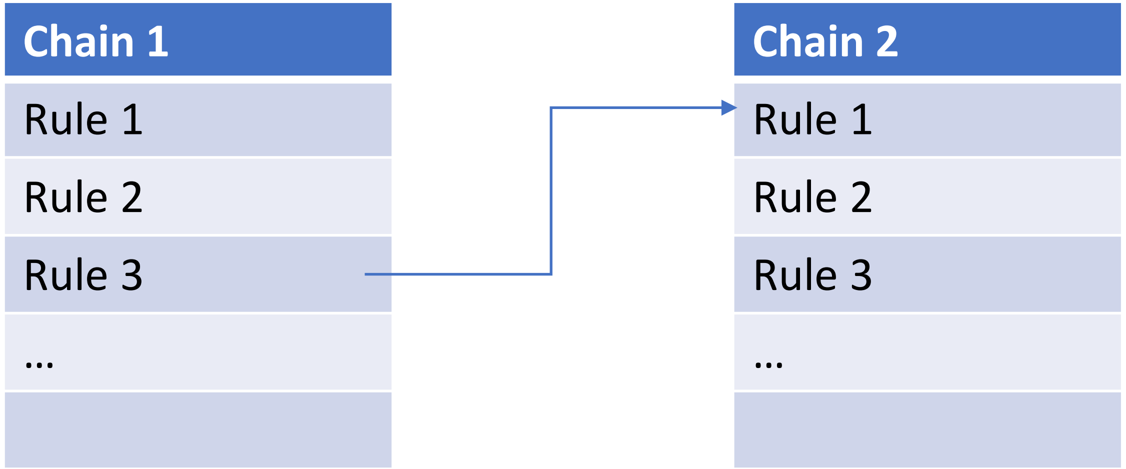
zoom in on IP layer



https://wiki.nftables.org/wiki-nftables/index.php/Netfilter_hooks

Chains

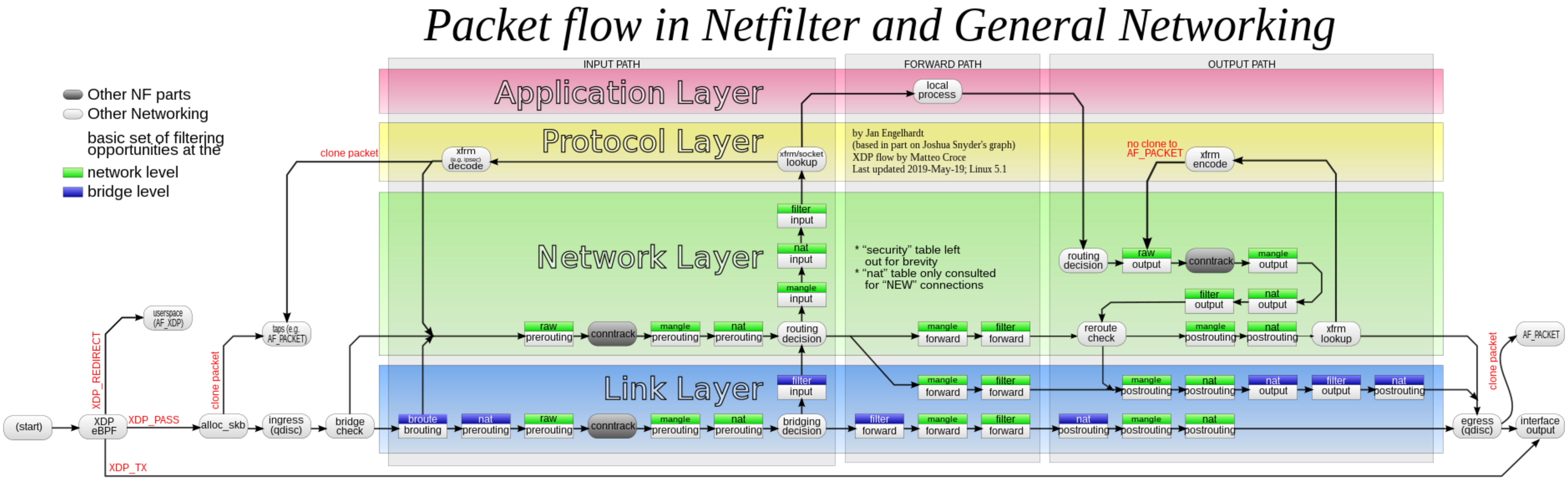
- Chain is just a set of rules that are evaluated sequentially
- Rules can be terminating or non-terminating
- Recall: **-j, --jump** *target*
  The target can be a user-defined chain (other than the one this rule is in), one of the special builtin targets which decide the fate of the packet immediately, or an extension



Chain 1
Rule 1
Rule 2
Rule 3
…

Chain 2
Rule 1
Rule 2
Rule 3
…

At each hook point, there's pre-defined chains

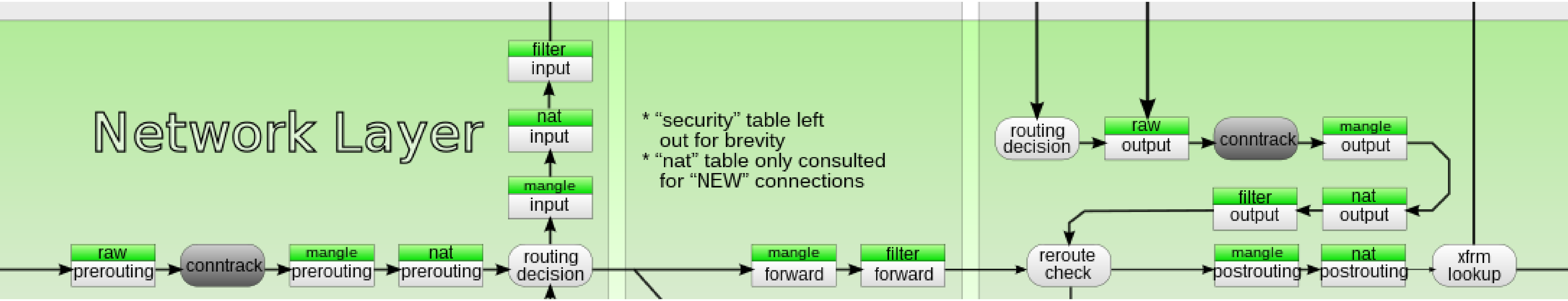


Packet flow in Netfilter and General Networking

by Jan Engelhardt
(based in part on Joshua Snyder's graph)
XDP flow by Matteo Croce
Last updated 2019-May-19; Linux 5.1

* "security" table left out for brevity
* "nat" table only consulted for "NEW" connections

By Jan Engelhardt - Own work, Origin SVG PNG, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=8575254



By Jan Engelhardt - Own work, Origin SVG PNG, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=8575254

Each Table Consists of Multiple Chains (named for the hook point)

- Filter – input, output, forward
- Nat – pre-routing, input, output, post-routing
- Mangle – all

Specifying Tables / Chains

- -t <table>
- -A <chain>   (to add an entry)

```
iptables -t filter -A FORWARD …
```

(same as: iptables -A FORWARD …)

```
iptables -t nat -A PREROUTING …
```

```
iptables -t mangle -A FORWARD …
```

Revisiting the Examples

What is the Table? Chain? Match Condition? Action?

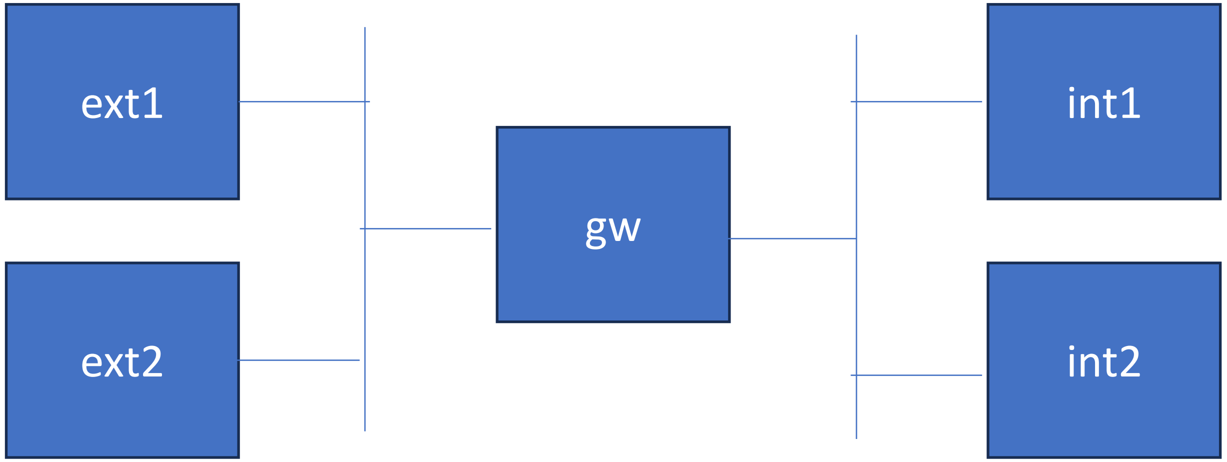
```
iptables -A FORWARD -m state --state ESTABLISHED,RELATED -j ACCEPT
```

```
iptables -A INPUT -p tcp --dport 80 -j DROP
```

```
iptables -A INPUT -i eth1 -s 10.0.0.0/8 -j LOG --log-prefix "IP_SPOOF A: "
```

Practice on your own

- Vagrant
- Container Lab
- Set of commands
  e.g., block traffic from ext2, but allow traffic from ext1



University of Colorado **Boulder**

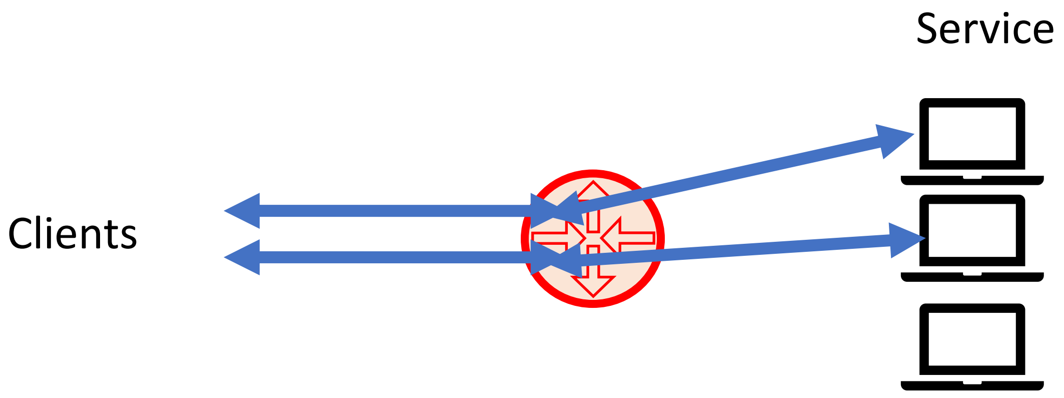
Load Balancing

Course: Networking Principles in Practice – Linux Networking
Module: Creating a Gateway with Linux

University of Colorado Boulder

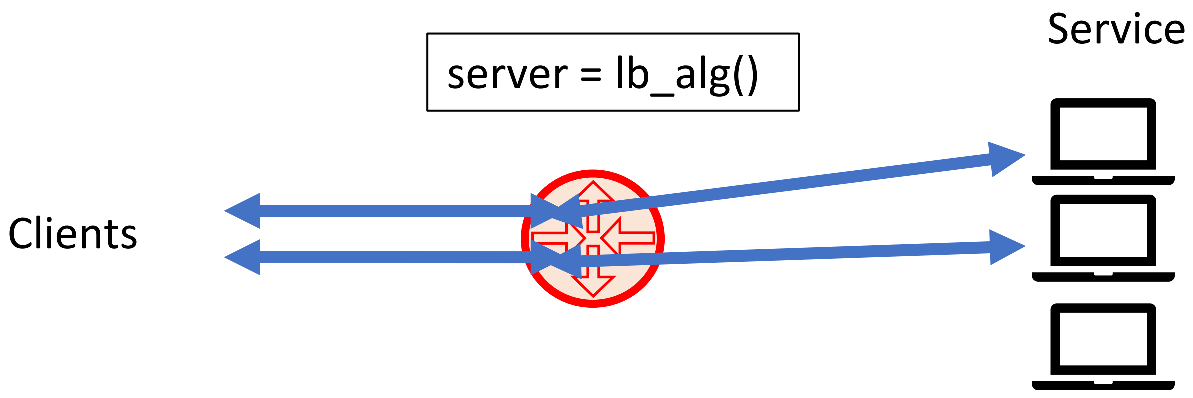
What is Load Balancing

- A load balancer serves as the point of entry for a service, and directs traffic to one of N servers that can handle the request
- Used for both scaling and for resilience
- Likely involves NAT



Load Balancing Algorithm

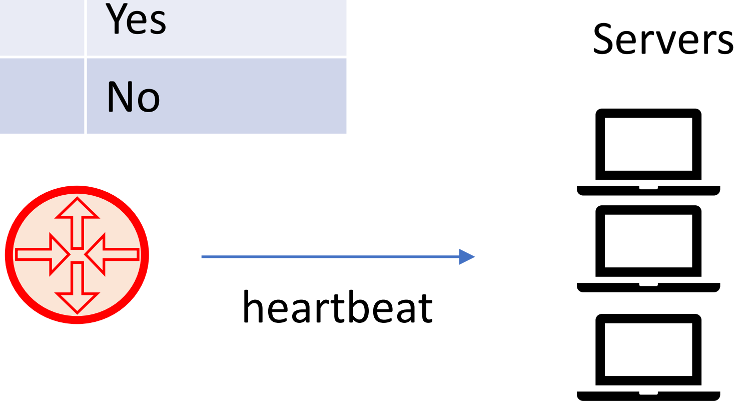
- Client initiates (TCP) connection
- Load balancer must select which server to forward the request to



Important Considerations: Server Set

- Which servers are part of the set to choose from
- Set is statically assigned
- May include liveness checks (through heartbeats)

Servers	Live
10.0.0.1	Yes
10.0.0.2	Yes
10.0.0.3	No



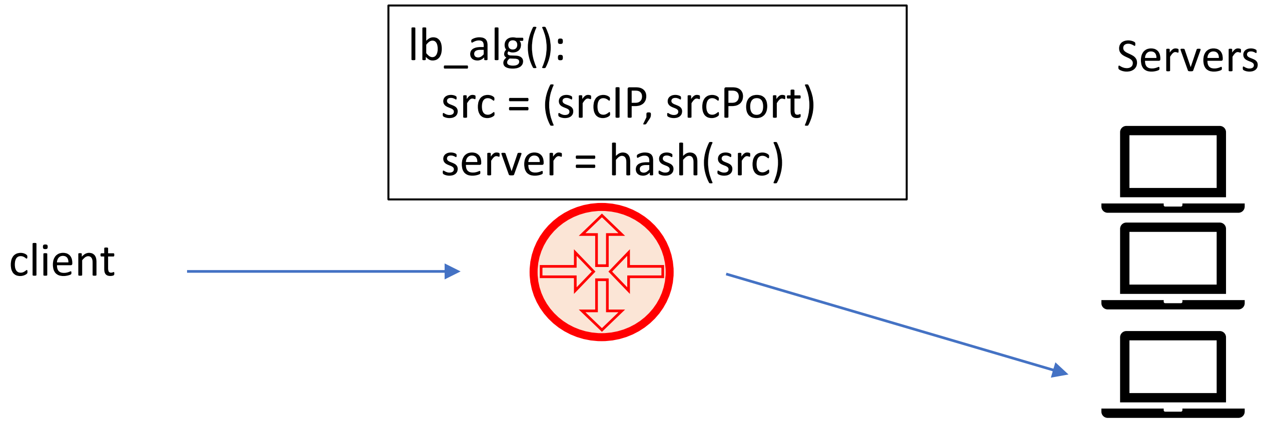
Important Considerations: Flow Affinity

- TCP is stateful, so need to ensure all packets from the same flow (TCP connection) go to the same server
- First packet – algorithm will select server
  - IP of server called VIP (virtual IP)
- Subsequent packets – look in table

Flow (src IP, port)	VIP
111.11.11.11, 1234	10.0.0.1
222.22.22.22, 2345	10.0.0.2

Alg 1: Source Hash

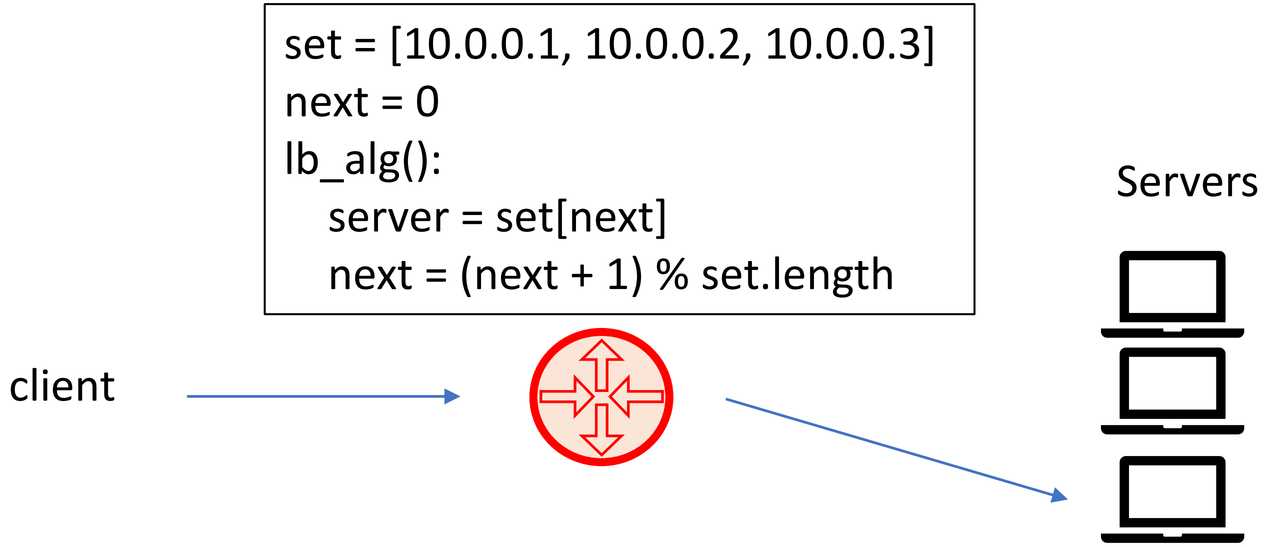
- For a source IP and Port, a hash function determines server
- Balances connections among servers
  (assuming a distribution of source IP and Ports)
- Stateless –flow affinity is taken care of by the hash function



```
lb_alg():
  src = (srcIP, srcPort)
  server = hash(src)
```

Alg 2: Round Robin

- Server selection iterates through each server in order (1, 2, 3, 1, 2,…)
- Ensures balanced load, assuming each request is roughly same load
- Load balancer needs to keep state for flow affinity

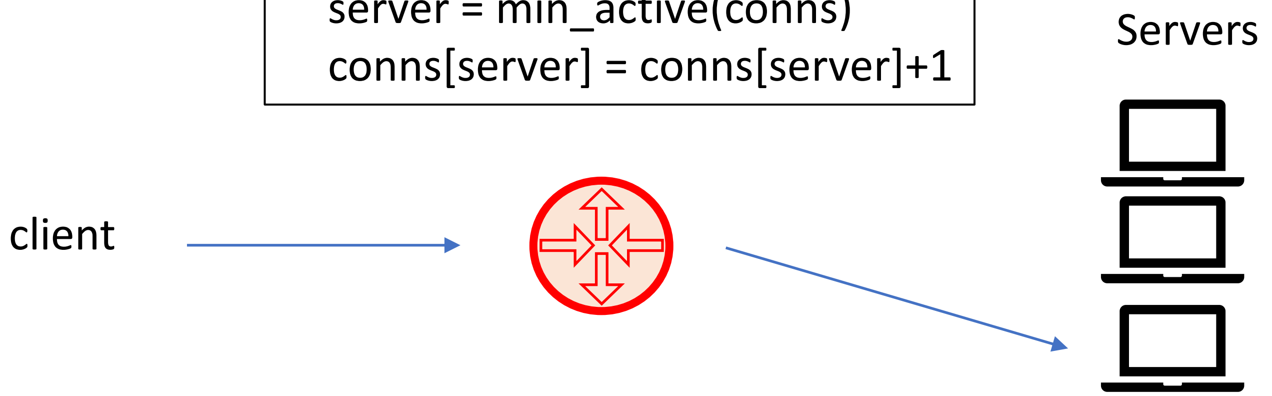


```
set = [10.0.0.1, 10.0.0.2, 10.0.0.3]
next = 0
lb_alg():
   server = set[next]
   next = (next + 1) % set.length
```

Alg 3: Least Connections

- Select server based on which has least number of active connections
- Takes into account that some requests may be longer, but assumes each connection imposes similar load on server
- Load balancer needs to keep state (flow affinity and algorithm)

```
lb_alg():
   server = min_active(conns)
   conns[server] = conns[server]+1
```



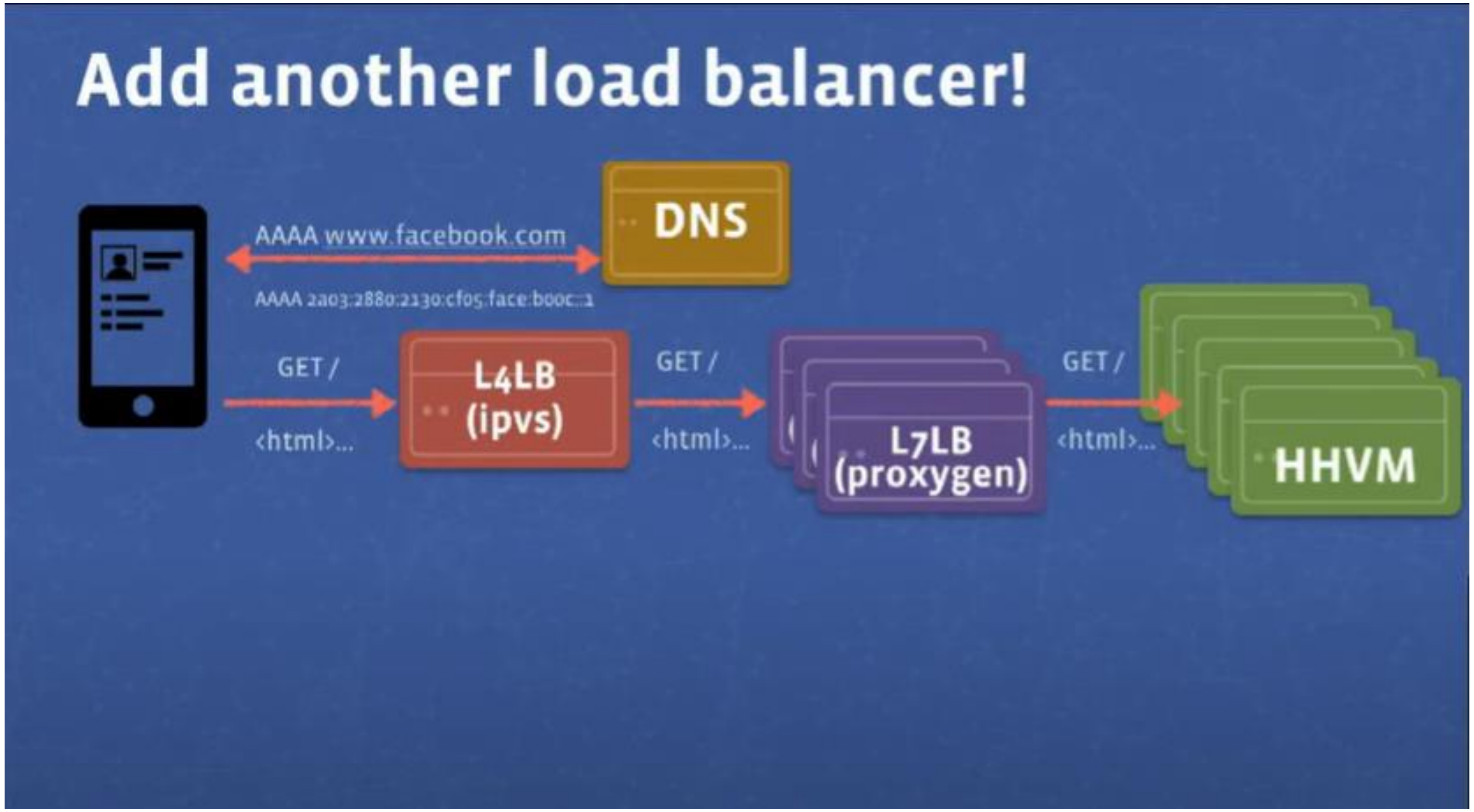
Layer 4 vs Layer 7 Load Balancing

- Layer 4 considers network (IP) and transport (TCP) headers
  - Also called a network load balancer

- Layer 7 also considers application (HTTP) header
  - Also called web proxy
  - e.g., http://<domain>/signup goes to server 1, http://<domain>/purchase goes to server 2 or 3

L4 and L7 Load Balancers can be used together

https://www.usenix.org/conference/lisa16/conference-program/presentation/shuff



Add another load balancer!

University of Colorado **Boulder**

Linux ipvs

Course: Networking Principles in Practice – Linux Networking
Module: Creating a Gateway with Linux

University of Colorado Boulder

ipvs and ipvsadm

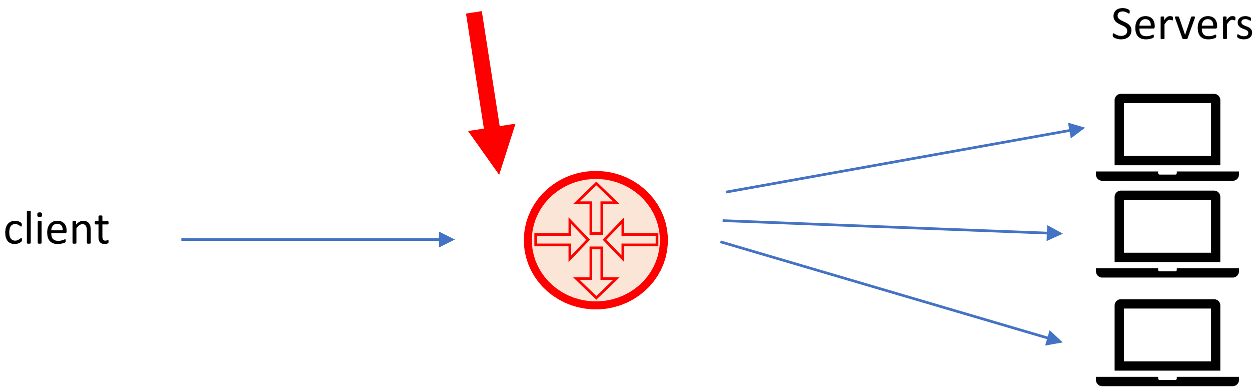
https://linux.die.net/man/8/ipvsadm

- IPVS (IP Virtual Server) implements transport-layer load balancing inside the Linux kernel
- ipvsadm is used to set up, maintain or inspect the virtual server table in the Linux kernel.

- Must have the ipvs kernel module loaded:
  `modprobe ip_vs`

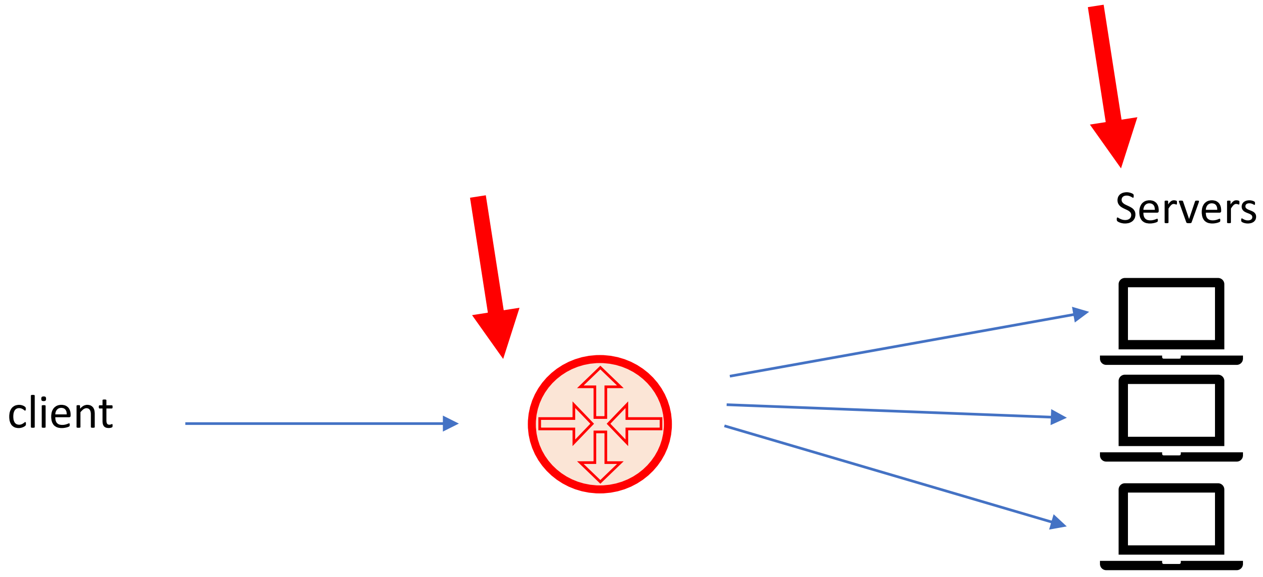
Two Parts

- Part 1: define the service



Two Parts

- Part 1: define the service
- Part 2: define the servers



Define the Service

ipvsadm *COMMAND* **[***protocol***]** *service-address* **[***scheduling-method***]** **[***persistence options***]**

- COMMAND
  - -A or --add-service
  - -E or --edit-service
  - -D or --delete-service

Running example:

ipvsadm -A

Define the Service

ipvsadm *COMMAND* **[***protocol***]** *service-address* **[***scheduling-method***]** **[***persistence options***]**

- Protocol
    - -t or --tcp-service
    - -u or --udp-service

Running example:

ipvsadm -A -t

Define the Service

ipvsadm *COMMAND* **[***protocol***]** *service-address* **[***scheduling-method***]** **[***persistence options***]**

- service-address: The publicly facing address.
  - The service-address is of the form host[:port].
  - Host may be one of a plain IP address or a hostname.
  - Port may be either a plain port number or the service name of port.

Running example:

```
ipvsadm -A -t 207.175.44.110:80
```

Define the Service

ipvsadm *COMMAND* **[***protocol***]** *service-address* **[***scheduling-method***]** **[***persistence options***]**

- scheduling-method
  - -s or --scheduler *scheduling-method*
  - rr – round robin
  - lc – least connections
  - sh – source hashing

Running example:

```
ipvsadm -A -t 207.175.44.110:80 -s rr
```

Adding a Server

ipvsadm *command* **[***protocol***]** *service-address server-address* **[***packet-forwarding-method***]** **[***weight options***]**

Command:

- -a or --add-server
- -e or --edit-server
- -d or --delete-server

Running example:

ipvsadm -a

Adding a Server

ipvsadm *command* **[***protocol***]** *service-address server-address* **[***packet-forwarding-method***]** **[***weight options***]**

service-address:

- Match whatever specified for adding the service

Running example:

```
ipvsadm -a -t 207.175.44.110:80
```

Adding a Server

ipvsadm *command* **[***protocol***]** *service-address server-address* **[***packet-forwarding-method***]** **[***weight options***]**

server-address: Address of backend server

- **-r, --real-server** *server-address*
- (need to talk about packet-forwarding-method first)

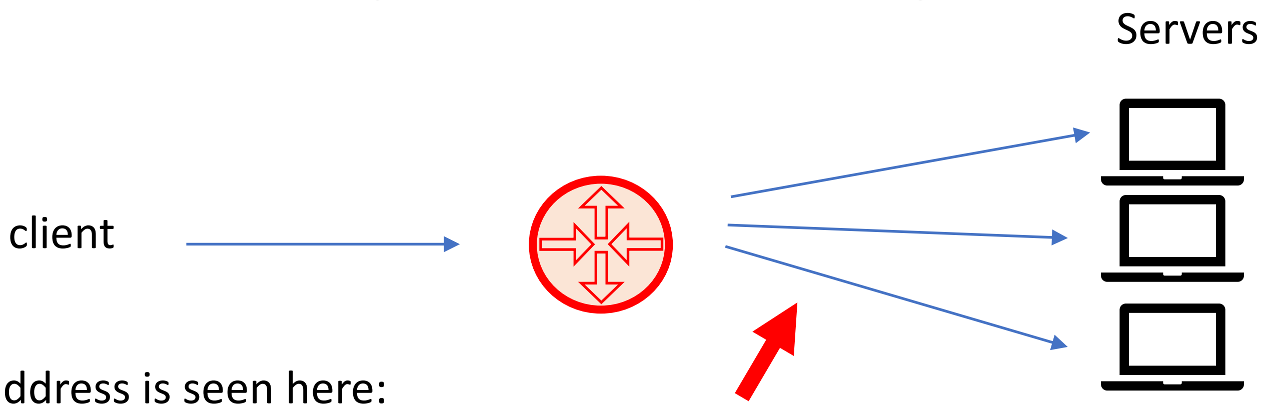
Running example:

ipvsadm -a -t 207.175.44.110:80

Adding a Server – Packet Forwarding Method

packet-forwarding-method

- -g, --gatewaying = Use gatewaying (direct routing). This is the default.
- -i, --ipip = Use ipip encapsulation (tunneling).
- -m, --masquerading = Use masquerading (network access translation, or NAT).

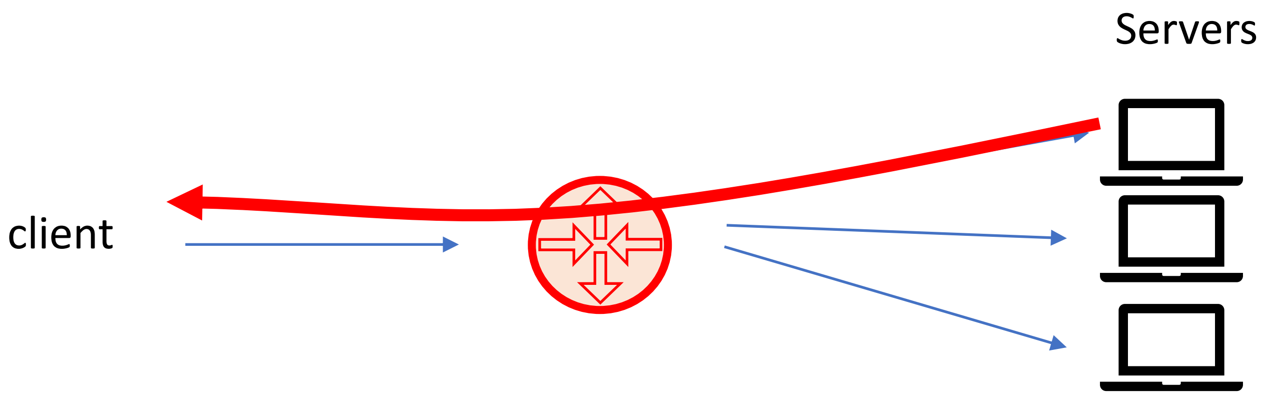


What address is seen here:
- Gatewaying – no change in address (same as from client)
- ipip – put outer IP header (src=LB addr, dest=server address)
- Masquerading – use NAT to translate dest address to that of server, and keep state for the translation

Aside: Direct Server Return (DSR)

When a server replies to the request how is traffic forwarded:

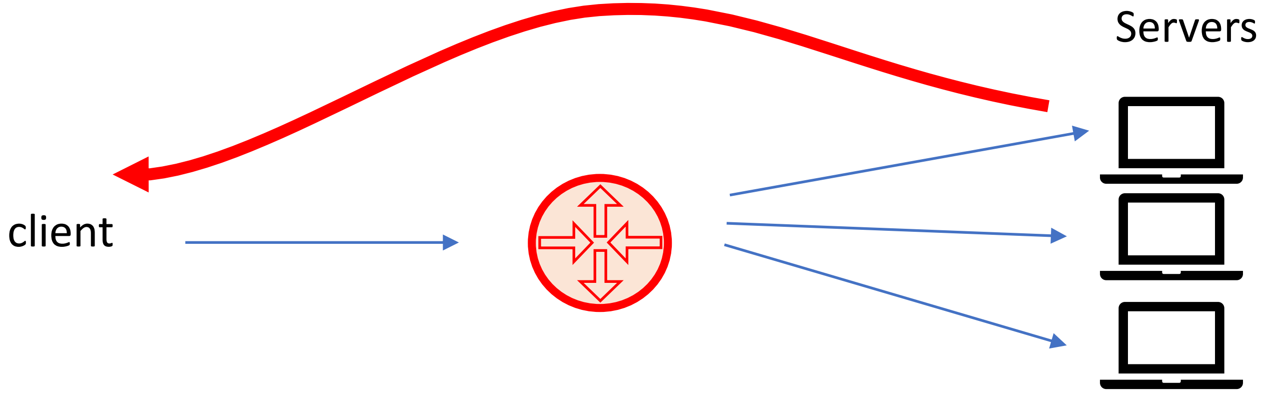
- Option 1: via the Load Balancer



Aside: Direct Server Return (DSR)

When a server replies to the request how is traffic forwarded:

- Option 1: via the Load Balancer
- Option 2: directly to the client (bypassing the load balancer)



Option 2 is known as DSR.

Adding a Server

ipvsadm *command* **[***protocol***]** *service-address server-address* **[***packet-forwarding-method***]** **[***weight options***]**

server-address: Address of backend server

- **-r, --real-server** *server-address*
- With Masquerading method, the port can be different from the service.
- With the tunneling (ipip) and direct routing (gatewaying) methods, port must be equal to that of the service address.

Running example:

```
ipvsadm -a -t 207.175.44.110:80 -r 192.168.10.1:80 -m
```

Adding a Server

ipvsadm *command* **[***protocol***]** *service-address server-address* **[***packet-forwarding-method***]** **[***weight options***]**

Weight options

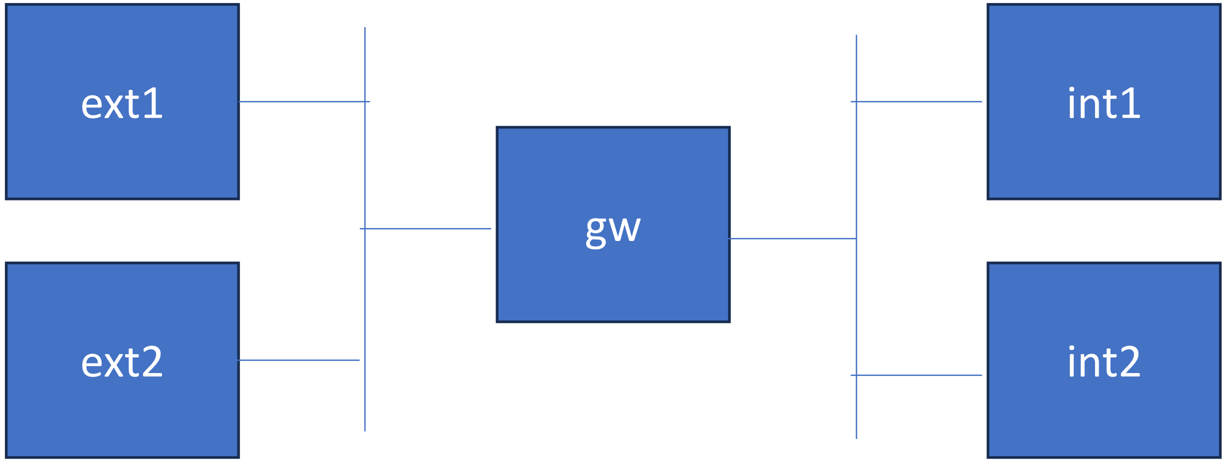
- -w, --weight *weight*
- Weight is an integer specifying capacity of server (used in scheduling algorithms with weight – e.g., weighted round robin)

Running example:

```
ipvsadm -a -t 207.175.44.110:80 -r 192.168.10.1:80 -m
```

Practice on your own

- Vagrant
- Container Lab
- Set of commands
  e.g., run a service on int1 and int2 (using nc), balance between them



University of Colorado **Boulder**

Quality of Service

Course: Networking Principles in Practice – Linux Networking
Module: Creating a Gateway with Linux

University of Colorado Boulder

Quality of Service

Quality of Service (QoS) refers to the overall performance of a service

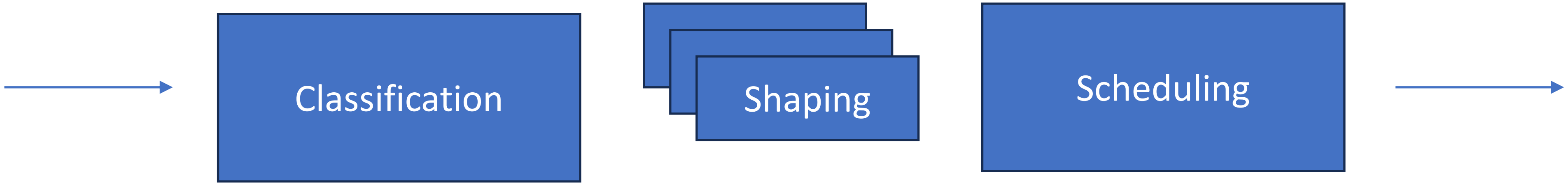
What might applications care about?

- Bandwidth
- Latency
- Jitter
- Loss

Different services will have different QoS requirements

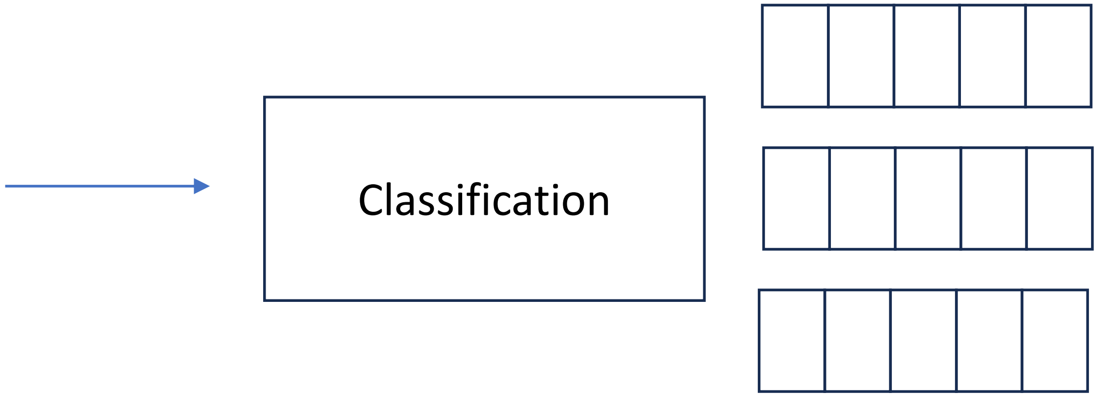
QoS Management

- Classification
  - Inspecting packets to identify what class of traffic they belong to
- Shaping
  - Ensuring a given class of traffic conforms to desired properties such as rates
- Scheduling
  - Determining which packet to transmit next and which packets to drop



Classification

- Given a packet, place it into a queue representing a traffic class
  - E.g., Low (e.g., email), Medium (web), and High priority traffic (video conf.)

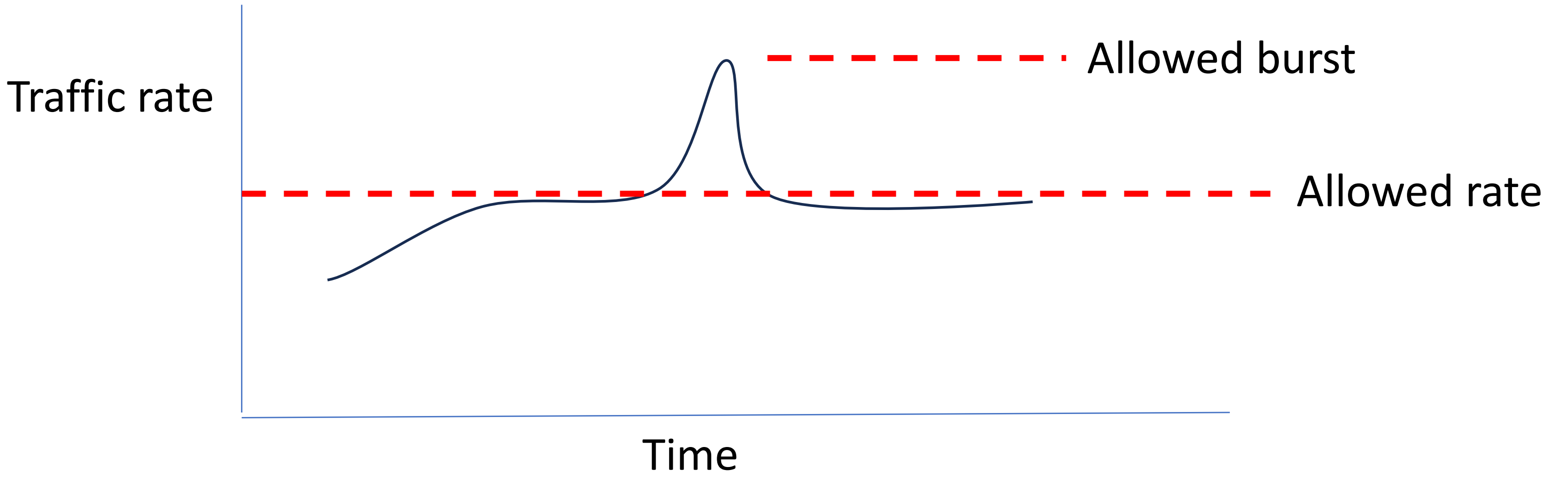


Classification Mechanisms

- Based on packet headers
  - Look at Layer 3 and 4 headers to identify traffic
  - e.g., port 80 = web traffic
  - e.g., IP source 1.2.3.0/24 pays more for service
- Based on Deep Packet Inspection (inspect packet payload)
  - Look at the payload for know headers or bit patterns
  - Computationally expensive, but helps identify traffic that can’t be at L3/4
- Based on fingerprinting
  - Protocols, applications, operating systems all have fingerprints, such as distribution of packet sizes, inter-packet gap, etc. so performing statistical analysis over many packets can identify traffic

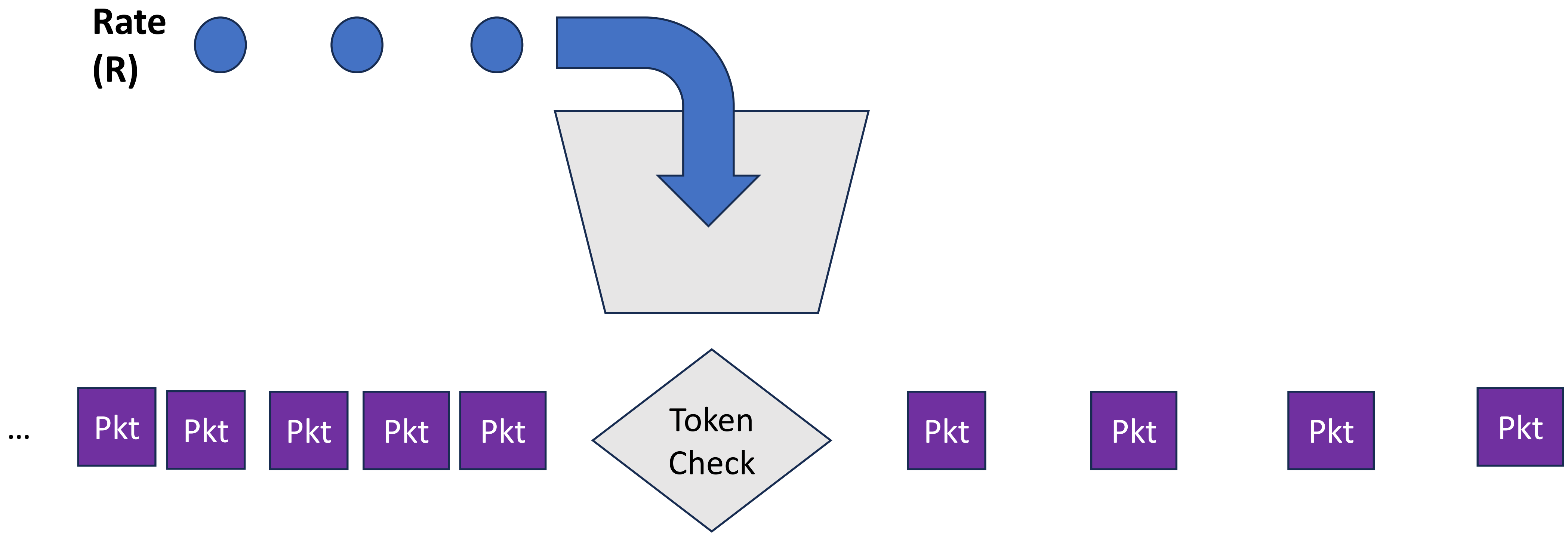
Traffic Shaping

- Goal is for traffic of a given class to conform to some shape
- Two key properties:
  - Traffic rate
  - Burst rate



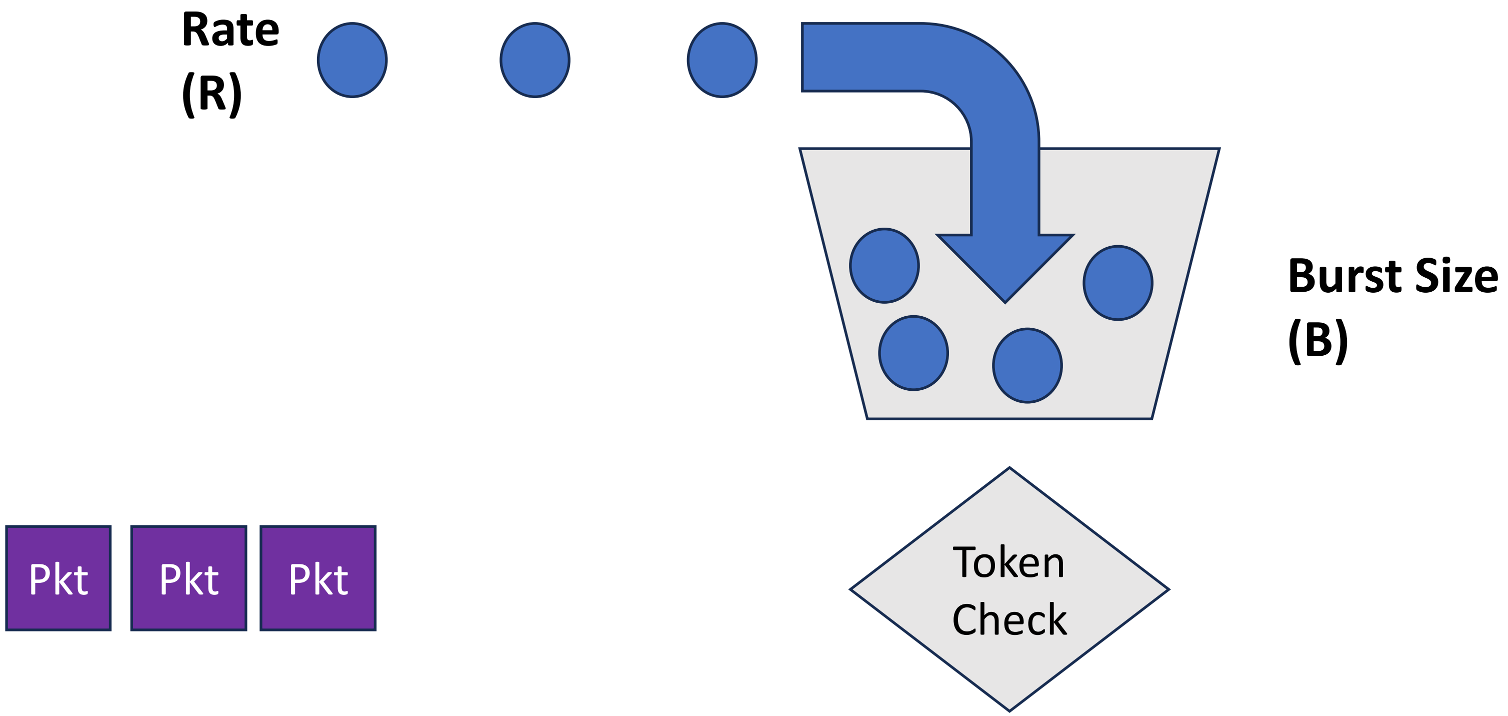
Token Bucket – Rate Limiting

- Tokens are added to a bucket at rate R
- A packet can be transmitted if there is a token in the bucket



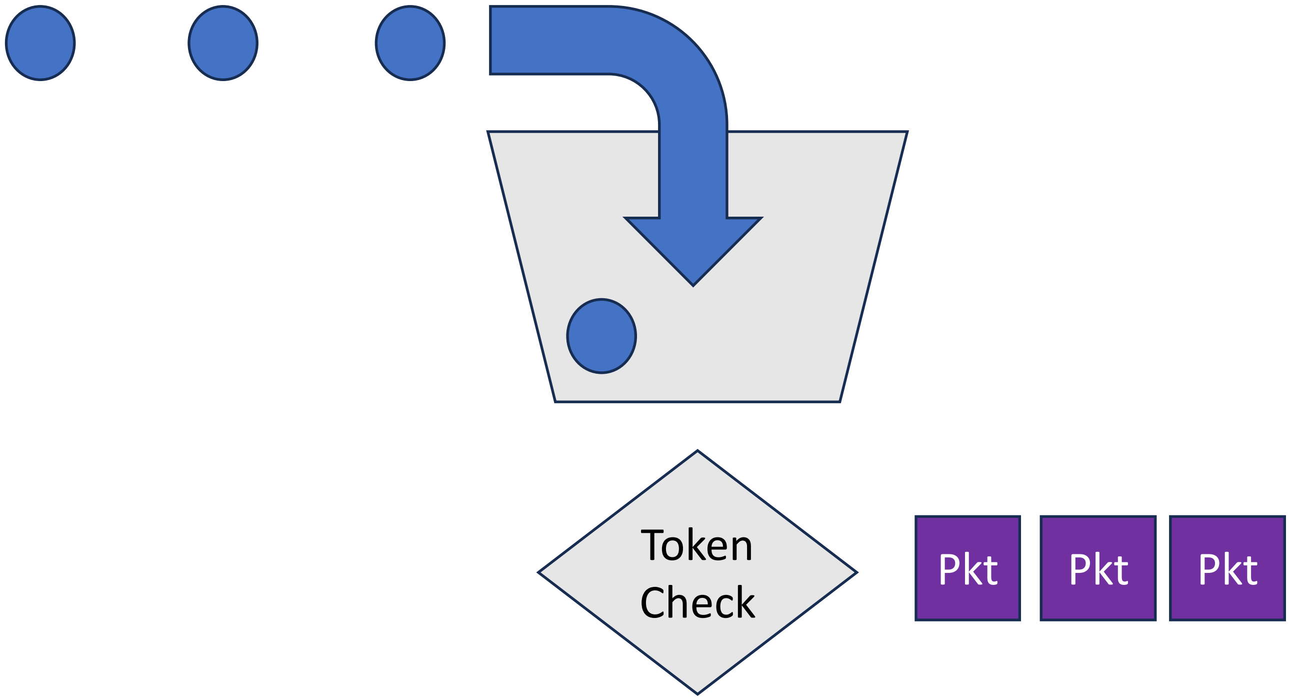
Token Bucket - Bursts

- The Bucket size (B), determines the burst rate
- e.g., say there’s no traffic for a while, tokens start filling up the bucket



Token Bucket - Bursts

- When we do get traffic (a burst), there are tokens for the entire burst
- The bucket will eventually overflow, which limits the burst size



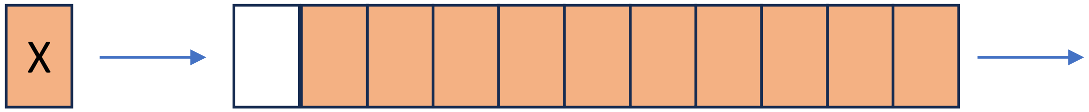
Scheduling

Two main functions:

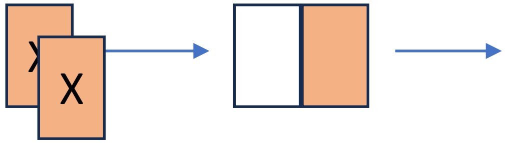
- Given a queue that is starting to fill up, determine what/when to drop
- Given multiple queues, determine which packet to transmit next

Queues

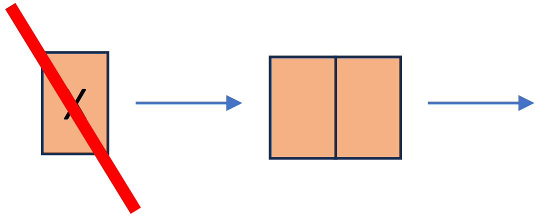
- Queue Size:
  - Too big – long delays with congestion



  - Too short – lead to a lot of drops

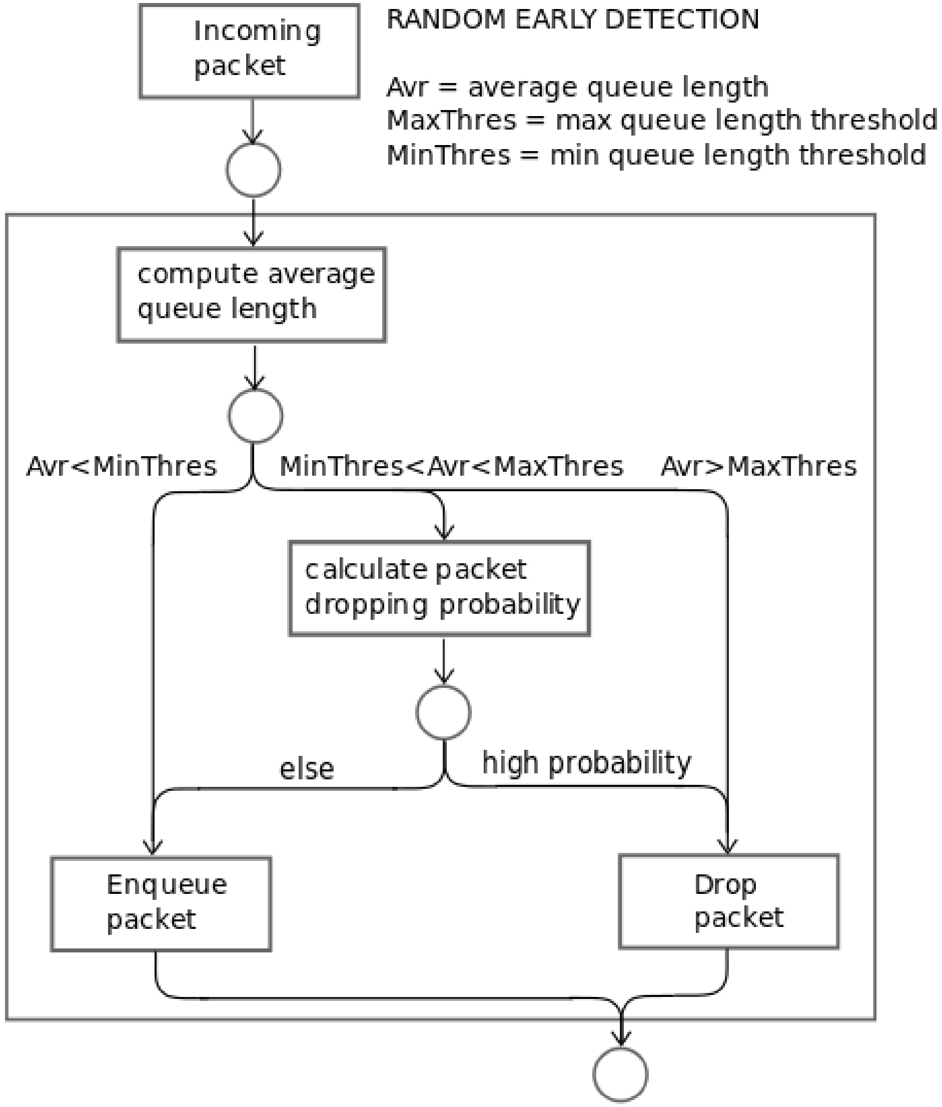


- What to Drop?
  - Tail Drop – when queue fills up, just drop from the tail



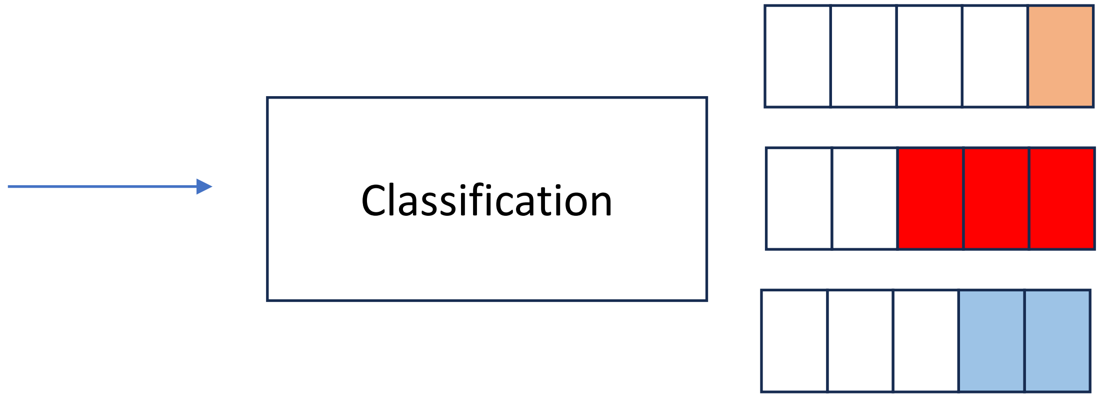
Problem with Tail Drop

- Consider TCP – it uses packet loss as an indication of congestion, and then slows sending rate.
- With tail-drop, packet loss would occur when congestion reached peak.
    - Then a round trip time or a timeout amount of time would be when TCP detects… in the meantime, many packets were sent (and likely dropped)
- RED (Random Early Detection), CoDel, FQ-CoDel, etc.
    - Drop packets earlier to signal to TCP earlier


RANDOM EARLY DETECTION
Avr = average queue length
MaxThres = max queue length threshold
MinThres = min queue length threshold
Incoming packet
compute average queue length
Avr<MinThres
MinThres<Avr<MaxThres
Avr>MaxThres
calculate packet dropping probability
else
high probability
Enqueue packet
Drop packet

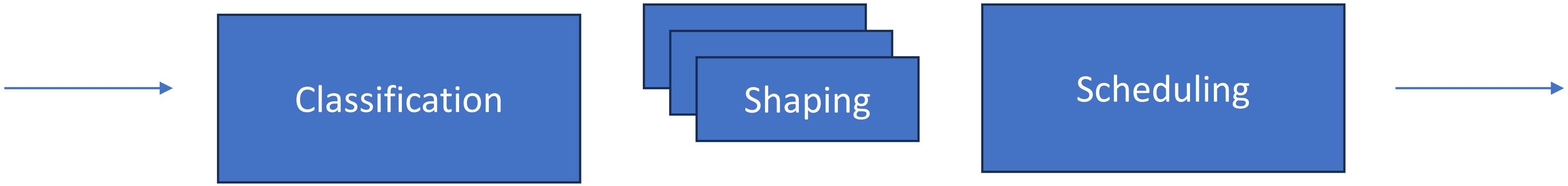
What Queue to Transmit From

- Strict Priority – If top queue has a packet send that, else look in next…
- Round Robin – Cycle through each queue, transmitting one packet from each
- Fair Queuing - mimic a bit-per-bit multiplexing by computing theoretical departure date for each packet



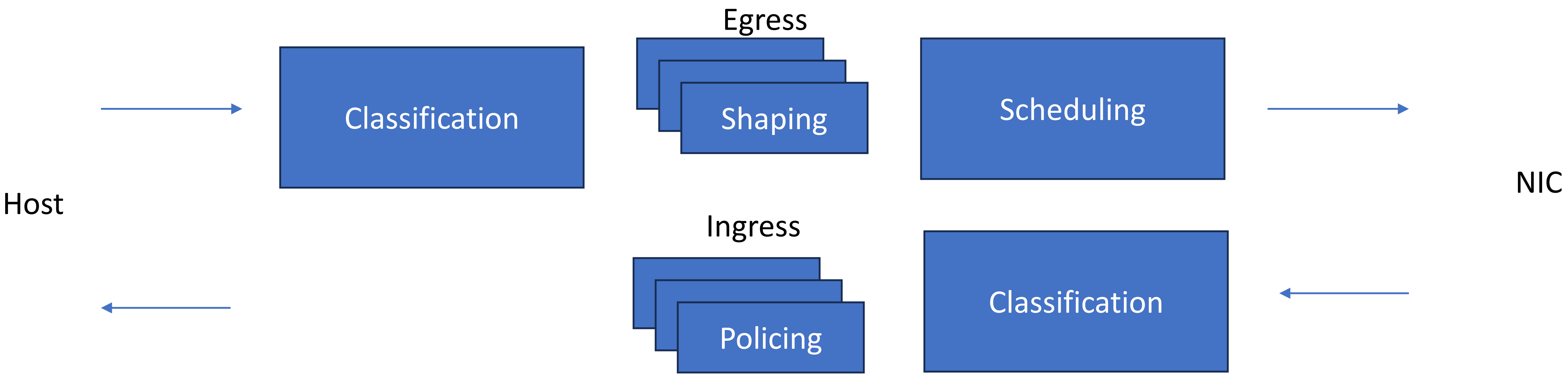
Recap: QoS Management

- Classification
  - Inspecting packets to identify what class of traffic they belong to
- Shaping
  - Ensuring a given class of traffic conforms to desired properties such as rates
- Scheduling
  - Determining which packet to transmit next and which packets to drop



Ingress - Policing

- Policing is similar to shaping – enforce a rate limit
- Difference is it can only take one corrective action:
  - Dropping a received packet



University of Colorado **Boulder**

tc

Course: Networking Principles in Practice – Linux Networking
Module: Creating a Gateway with Linux

University of Colorado Boulder

tc – Linux Traffic Control Utility

https://man7.org/linux/man-pages/man8/tc.8.html

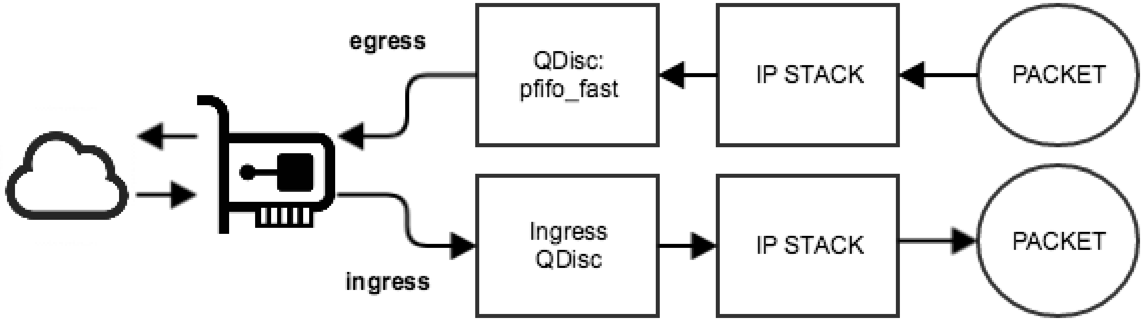
- Used for QoS setup
- But, actually can do much more

Key constructs:

- qdisc
- class
- filter

Queuing Discipline (qdisc)

- Main object for shaping / scheduling
- Every interface must have an ingress and egress qdisc
- Packets go into a qdisc and then goes out the other side
- Simple example: pfifo_fast – first in first out
  - Default used by Linux (if you don’t configure anything)



Pic: https://medium.com/criteo-engineering/demystification-of-tc-de3dfe4067c2

Adding/Removing a qdisc

```
tc qdisc [add | delete | replace…] dev DEV \
```

- Add, delete, or replace qdisc
- Each qdisc must be associated with a device

Example:

```
tc qdisc add dev eth0 …
```

Adding/Removing a qdisc

```
tc qdisc [add | delete | replace…] dev DEV \
          [ parent qdisc-id | root ] [ handle qdisc-id ]  \
```

- qdisc can be hierarchical, so need to specify parent (or root if root of hierarchy)
- Can also specify a handle

Example:

```
tc qdisc add dev eth0 root handle 1: …
```

Adding/Removing a qdisc

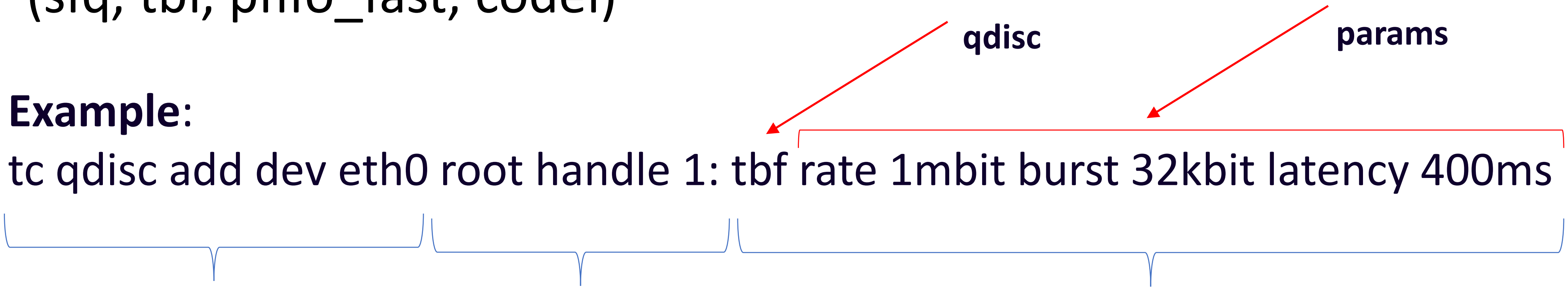
tc qdisc [**add** | **delete** | **replace**…] **dev** *DEV* \

[parent *qdisc-id* **| root] [handle** *qdisc-id*] \

qdisc [qdisc specific parameters]

- Then need to specify the qdisc to add along with its parameters (sfq, tbf, pfifo_fast, codel)

Example:



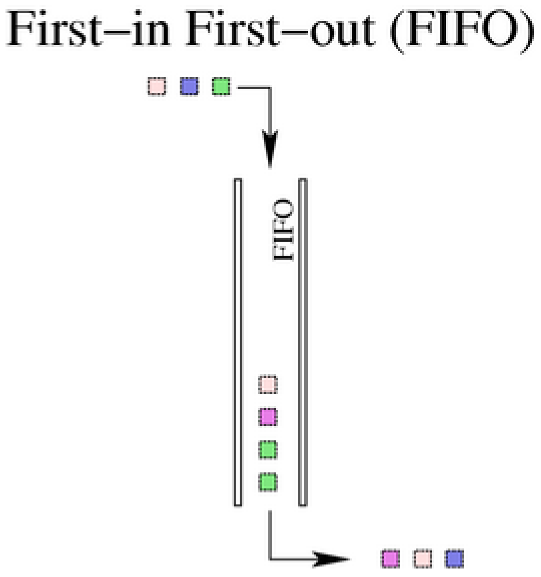
qdisc

params

```
tc qdisc add dev eth0 root handle 1: tbf rate 1mbit burst 32kbit latency 400ms
```

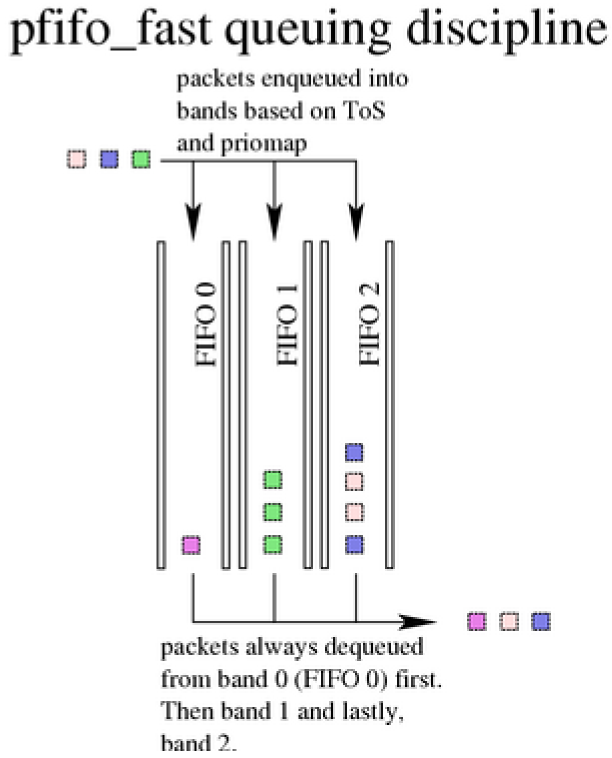
Example qdisc (1)

pfifo
bfifo



https://man7.org/linux/man-pages/man8/tc-bfifo.8.html

pfifo_fast



packets enqueued into bands based on ToS and priomap

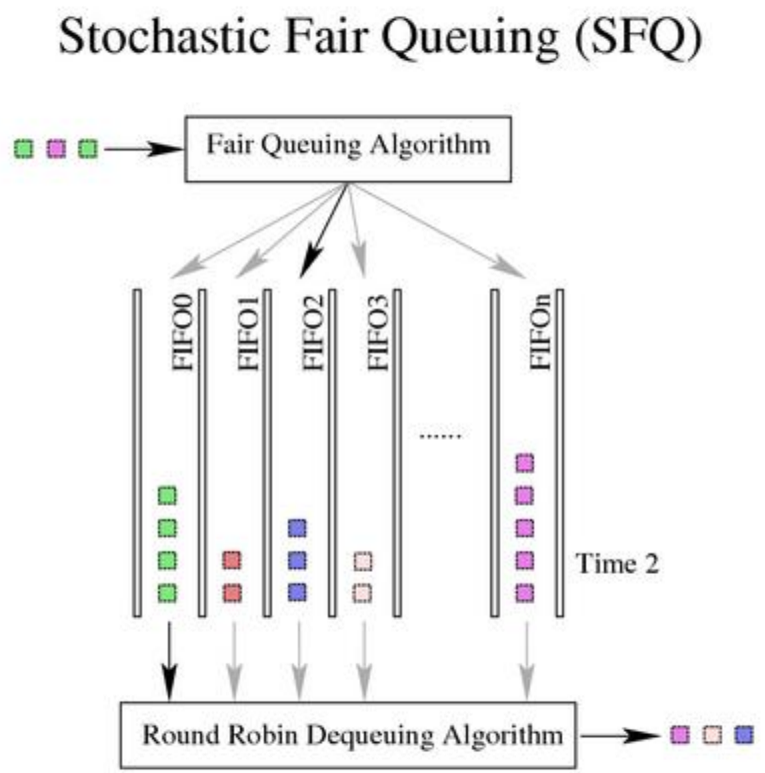
packets always dequeued from band 0 (FIFO 0) first. Then band 1 and lastly, band 2.

https://man7.org/linux/man-pages/man8/tc-pfifo_fast.8.html

Pics: https://tldp.org/HOWTO/Traffic-Control-HOWTO/classless-qdiscs.html

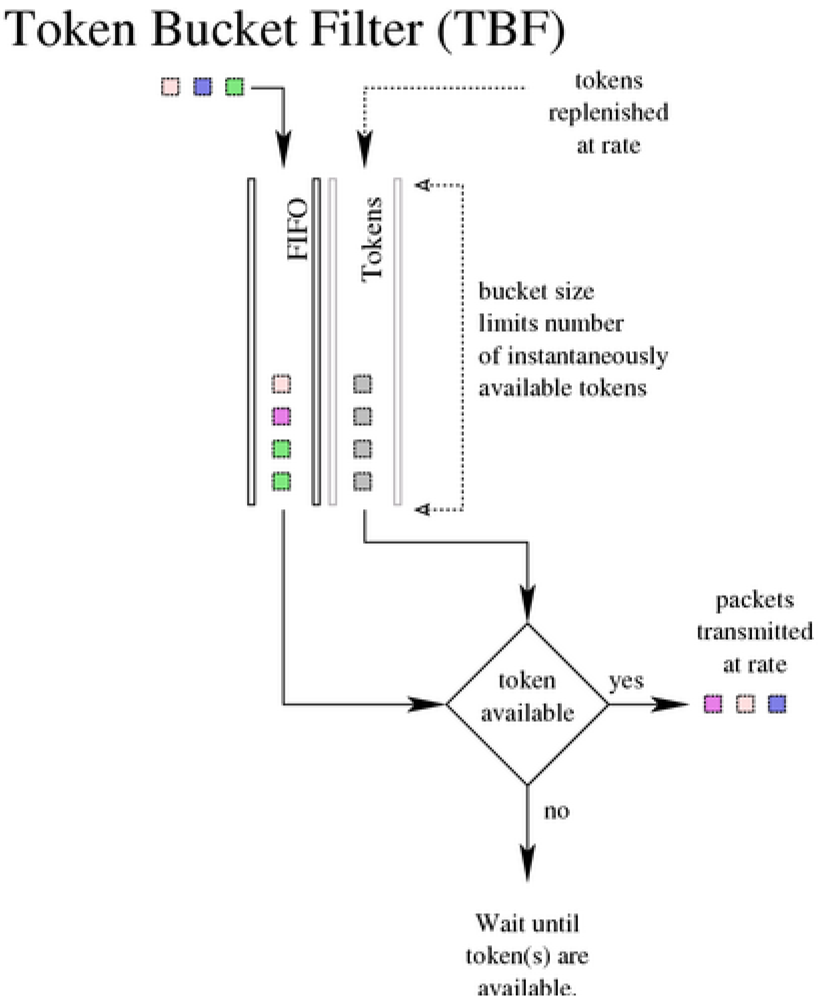
Example qdisc (2)

sfq


Stochastic Fair Queuing (SFQ)
Fair Queuing Algorithm
FIFO0
FIFO1
FIFO2
FIFO3
......
FIFOn
Time 2
Round Robin Dequeuing Algorithm

https://man7.org/linux/man-pages/man8/tc-sfq.8.html

tbf


Token Bucket Filter (TBF)
tokens replenished at rate
FIFO
Tokens
bucket size limits number of instantaneously available tokens
token available
yes
packets transmitted at rate
no
Wait until token(s) are available.

https://man7.org/linux/man-pages/man8/tc-tbf.8.html

Pics: https://tldp.org/HOWTO/Traffic-Control-HOWTO/classless-qdiscs.html

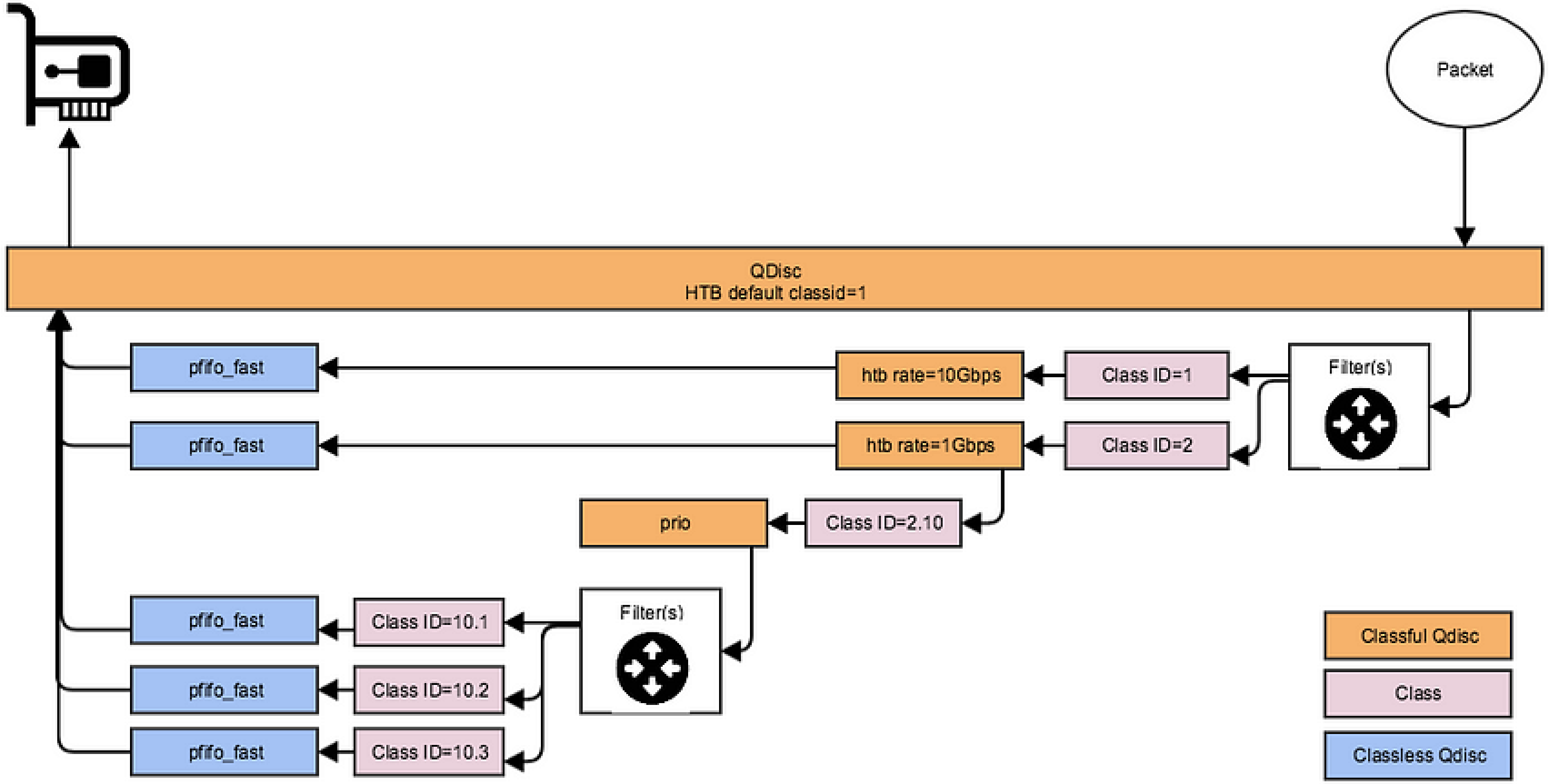
Classes

- qdiscs can be classless or classful
    - Examples on previous two slides were all classless

- “Some qdiscs can contain classes, which contain further qdiscs - traffic may then be enqueued in any of the inner qdiscs, which are within the **classes.”**

tc [*OPTIONS*] **class [add | change | replace | delete | show] dev** *DEV* **parent** *qdisc-id* **[classid** *class-id*] qdisc [qdisc specific parameters]

Classes - Illustration



Pic: https://medium.com/criteo-engineering/demystification-of-tc-de3dfe4067c2

Adding / Removing classes

tc [*OPTIONS*] **class [add | change | replace | delete | show] dev** *DEV* **parent** *qdisc-id* **[classid** *class-id*] qdisc [qdisc specific parameters]

- Note: nearly identical to qdisc:
  - can’t be root
  - classid instead of handle – same basic type
  - the type (qdisc) must be the same as parent qdisc

Classful qdisc Ex: Hierarchical Token Bucket (HTB) (allows tokens to be shared)

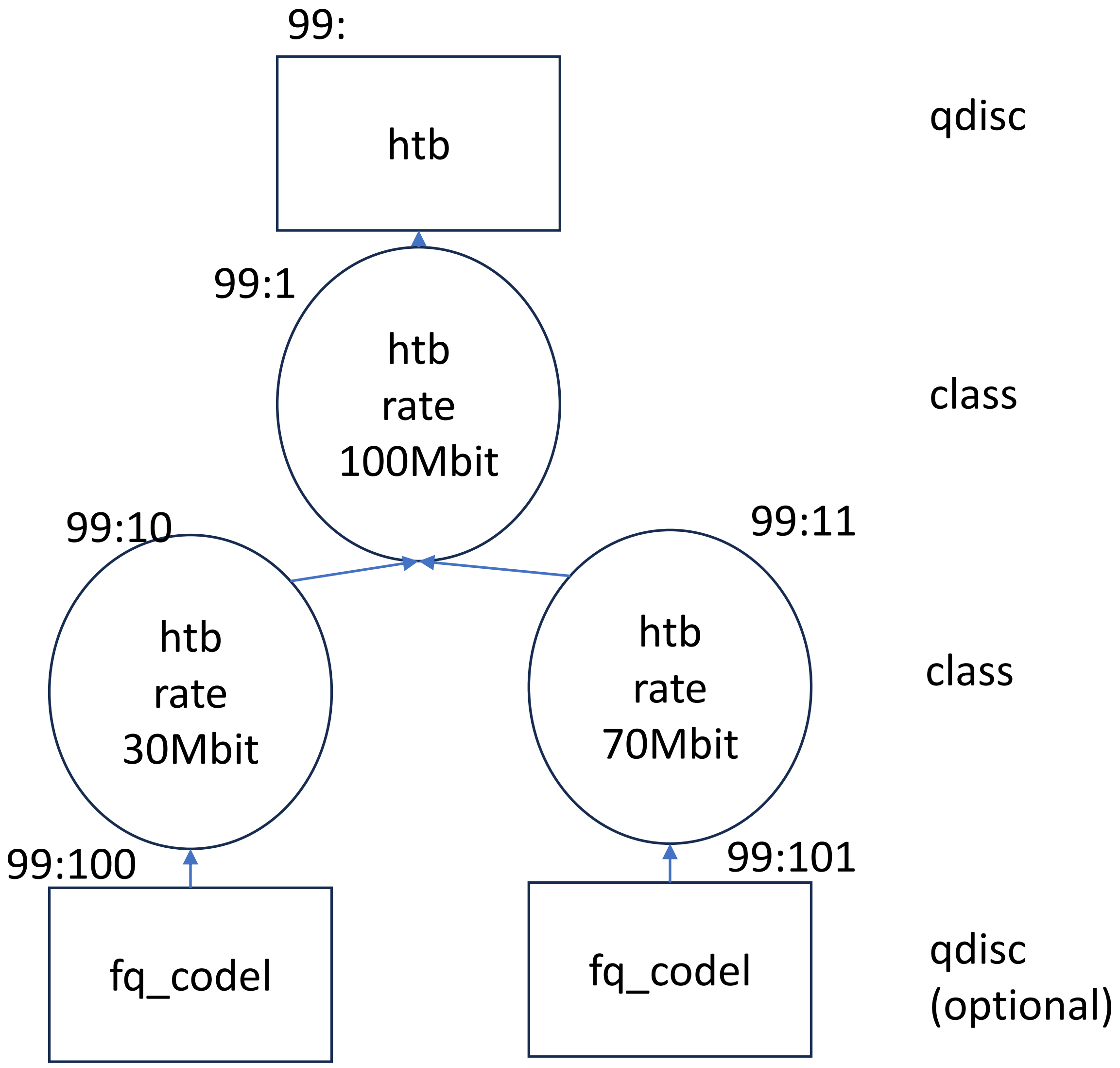
http://luxik.cdi.cz/~devik/qos/htb/manual/userg.htm

```
tc qdisc add dev eth0 root handle 99: htb
```

```
tc class add dev eth0 parent 99: classid 99:1 htb rate 100mbit
```

```
tc class add dev eth0 parent 99:1 classid 99:10 htb rate 30mbit
tc class add dev eth0 parent 99:1 classid 99:11 htb rate 70mbit
```

```
tc qdisc add dev eth0 parent 99:10 fq_codel
tc qdisc add dev eth0 parent 99:11 fq_codel
```



Filters: specifying which traffic is in which class

```
tc filter [ add | replace | delete | … ] dev DEV  \
        [ parent qdisc-id | root ] [ handle filter-id ] \
        protocol protocol prio priority \
        filtertype [ filtertype specific parameters ]
```

- Enable specifying which traffic is in which class

Filters: specifying which traffic is in which class

tc filter [add | replace | delete | …] dev *DEV* \
[parent *qdisc-id* **| root] [handle** *filter-id* **]** \
protocol *protocol* **prio** *priority* \
filtertype [filtertype specific parameters]

- Need to specify the device to associate with
- Also need to specify the parent qdisc

Example:

tc filter add dev eth0 parent 1: protocol ip prio 1 u32 match ip src 1.2.3.4 classid 99:10

Filters: specifying which traffic is in which class

tc filter [add | replace | delete | …] dev *DEV* \

[parent *qdisc-id* **| root] [handle** *filter-id* **]** \

protocol *protocol* **prio** *priority* \

filtertype [filtertype specific parameters]

- Specify the protocol this applies to (all, ip, ipv6)
- Priority will specify the order of evaluation (lower prio number means higher priority)

Example:

tc filter add dev eth0 parent 1: protocol ip prio 1 u32 match ip src 1.2.3.4 classid 99:10

Filters: specifying which traffic is in which class

tc filter [add | replace | delete | …] dev *DEV* \\
[parent *qdisc-id* **| root] [handle** *filter-id* **]** \\
protocol *protocol* **prio** *priority* \\
filtertype [filtertype specific parameters]

- There are many different types of filters
- Shown is a basic one u32

Example:

```
tc filter add dev eth0 parent 1: protocol ip prio 1 u32 match ip src 1.2.3.4 classid 99:10
```

Match Filters

- u32 Generic filtering on arbitrary packet data, assisted by syntax to abstract common operations. https://man7.org/linux/man-pages/man8/tc-u32.8.html

- fw Filter based on fwmark. Directly maps fwmark value to traffic class. https://man7.org/linux/man-pages/man8/tc-fw.8.html

- flow, flower Flow-based classifiers, filtering packets based on their flow (identified by selectable keys) https://man7.org/linux/man-pages/man8/tc-flower.8.html

Filters – Match / Action

- Each has syntax to specify the match condition
- Each has ability to specify the action of the form:
  - classid <class-id> - pushes packet to specified class
  - action <action> – performs some action specific to the match filter https://man7.org/linux/man-pages/man8/tc-actions.8.html

Example:

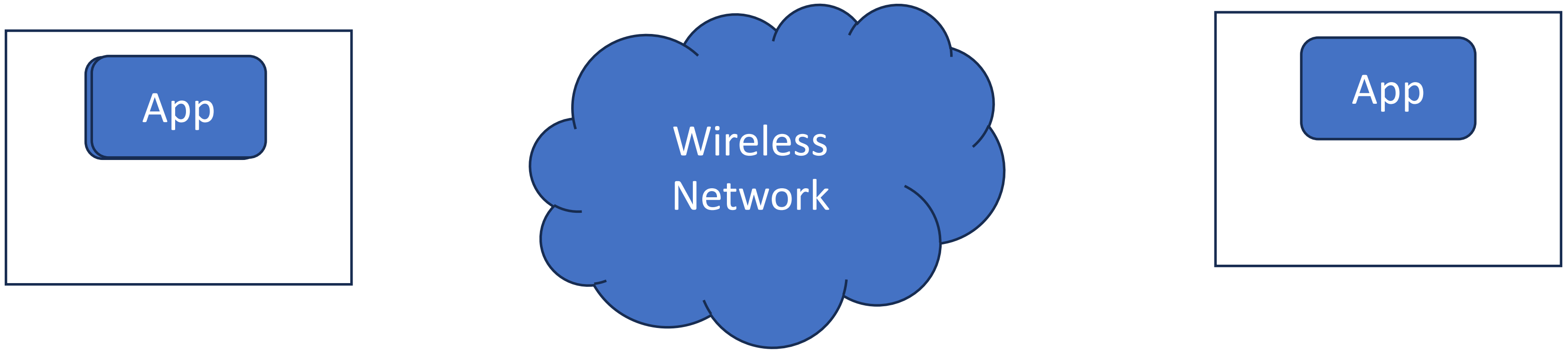
```
tc filter add dev eth0 parent 1: protocol ip prio 1 u32 match ip src 1.2.3.4 classid 99:10
```

Another Interesting qdisc: netem

- Provides Network Emulation functionality for testing protocols by emulating the properties of real-world networks. https://man7.org/linux/man-pages/man8/tc-netem.8.html
- Add delays to packets
- Add packet loss with some probability
- …

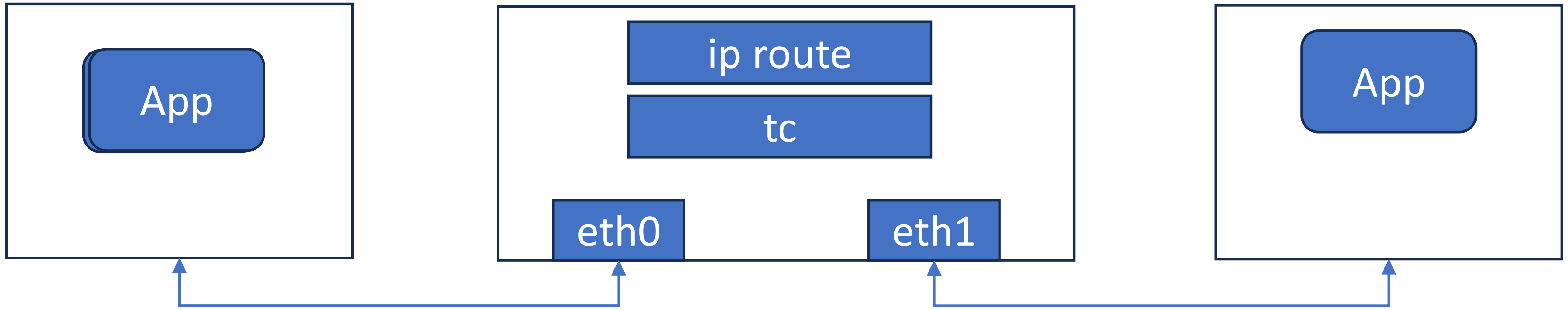
Example use of netem

- Say I created a new application or protocol and I want to test out how it would behave under different conditions on a wireless network



Example use of netem

- Say I created a new application or protocol and I want to test out how it would behave under different conditions on a wireless network

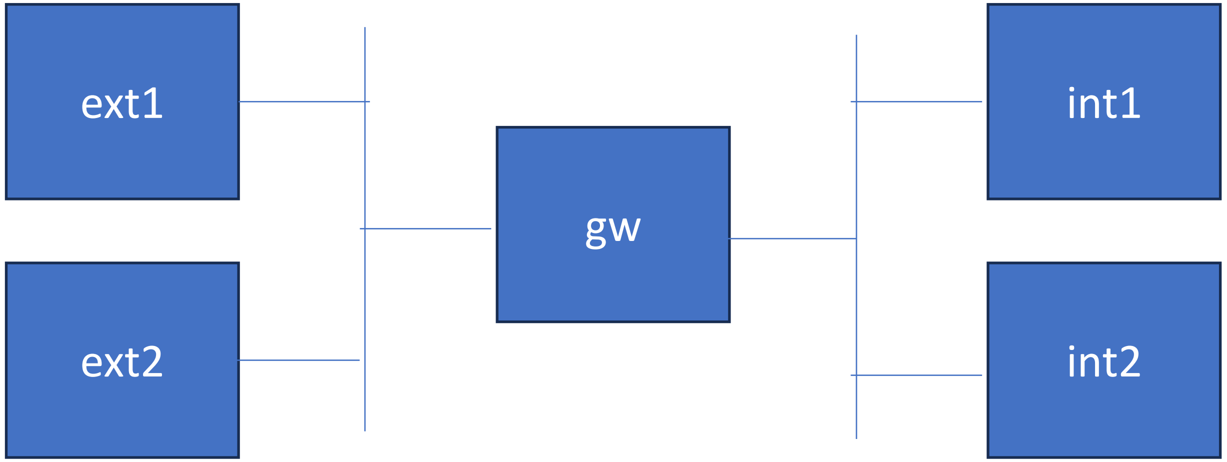


```
tc qdisc replace dev eth0 root netem delay 100ms 12ms 10%

tc qdisc replace dev eth1 root netem delay 100ms 12ms 10%
```

Practice on your own

- Vagrant
- Container Lab
- Set of commands
  e.g., run iperf on ext1 and int1 and control the rate of traffic



University of Colorado **Boulder**