Netfilter and iptables

Network Infrastructures labs

Marco Spaziani Brunella

SAPIENZA
UNIVERSITÀ DI ROMA

Lecture details

- **Readings:**
  - Linux iptables Pocket Reference; Gregor N. Purdy; O'Reilly
- **Lecture outline:**
  - Netfilter
  - iptables

Netfilter

- **NETFILTER** is a framework that provides hook handling within the Linux kernel for intercepting and manipulating network packets
- A **hook** is an "entry point" within the Linux Kernel IP (v4|v6) networking subsystem that allows packet mangling operations
  - Packets traversing (incoming/outgoing/forwarded) the IP stack are **intercepted** by these hooks, **verified** against a given set of matching rules and **processed** as described by an **action** configured by the user
- 5 built-in hooks:
  - PRE_ROUTING, LOCAL_INPUT, FORWAD, LOCAL_OUT, POST_ROUTING

Netfilter basics 1/2

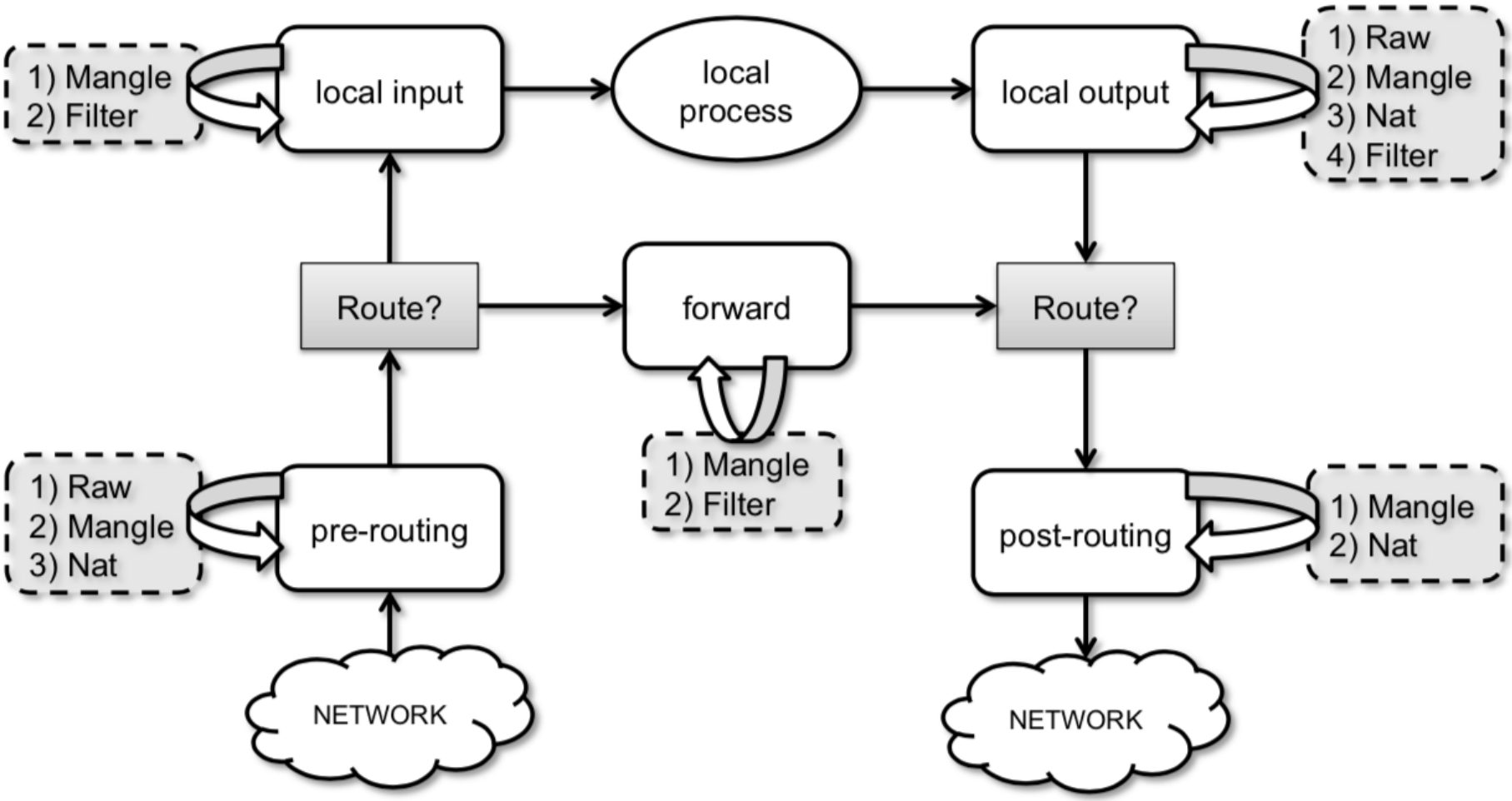
- All packet intercepted by the hooks pass through a sequence of built-in **tables** (queues) for processing. Each of these queues is dedicated to a particular type of packet activity and is controlled by an associated packet transformation/filtering chain

- 4 built-in tables
  - **Filter**: packet filtering (accept, drop)
  - **Nat**: network address translation (snat, dnat, masquerade)
  - **Mangle**: modify the packet header (tos, ttl)
  - **Raw**: used mainly for configuring exemptions from connection tracking in combination with the NOTRACK target

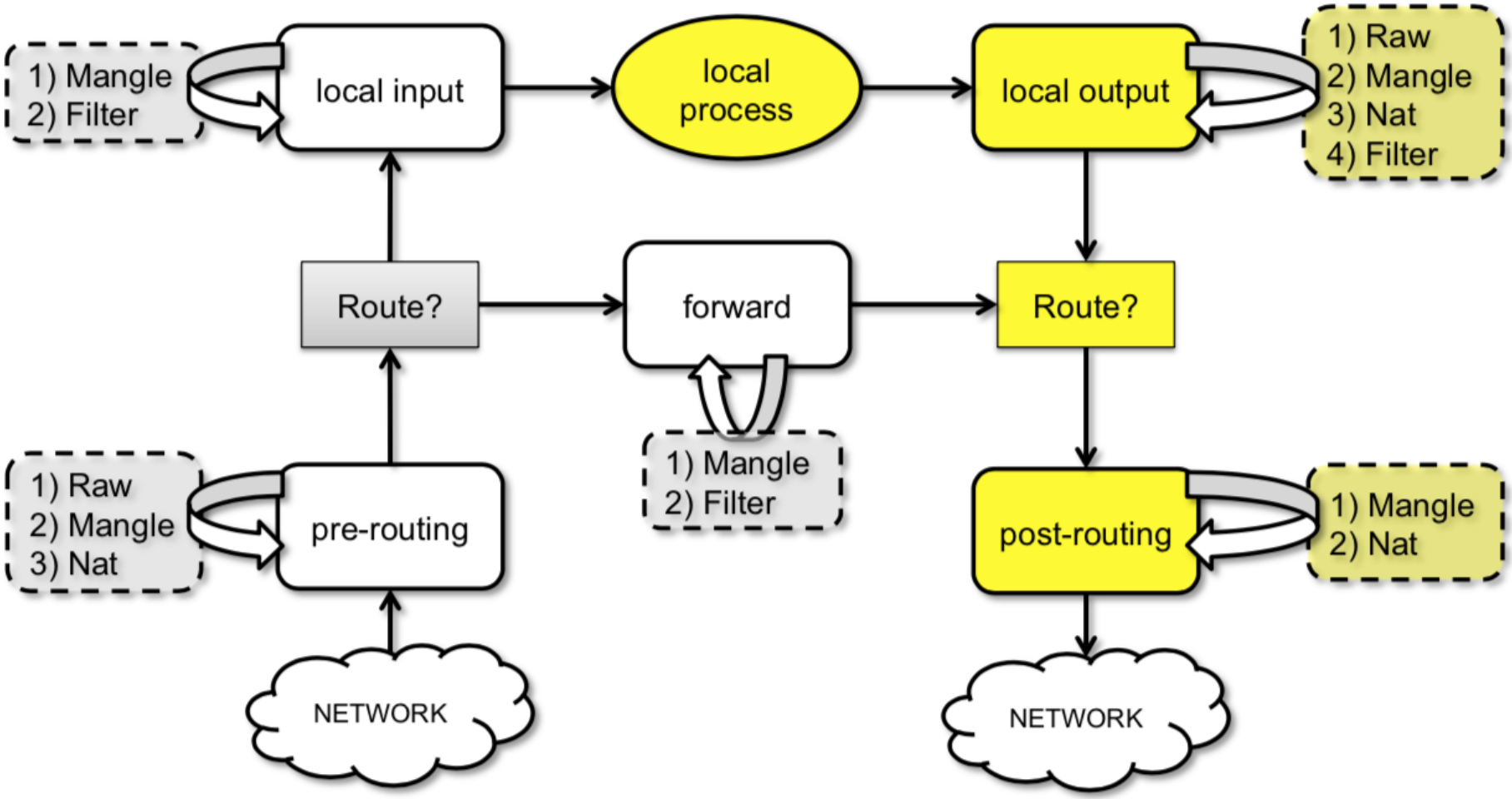
Netfilter basics 2/2

- The **matching rules** and the **actions (targets)** are implemented by different kernel modules and provides a powerful packet filtering system
- Common matches
  - Protocol, source/destination address or network, input/output interface, source/destination TCP/UDP port
- Common targets
  - ACCEPT, DROP, MASQUERADE, DNAT, SNAT, LOG
- NETFILTER is extensible
  - You can register your custom HOOK
  - You can write your own “matching module” and “action module”
  - http://jengelh.medozas.de/documents/Netfilter_Modules.pdf

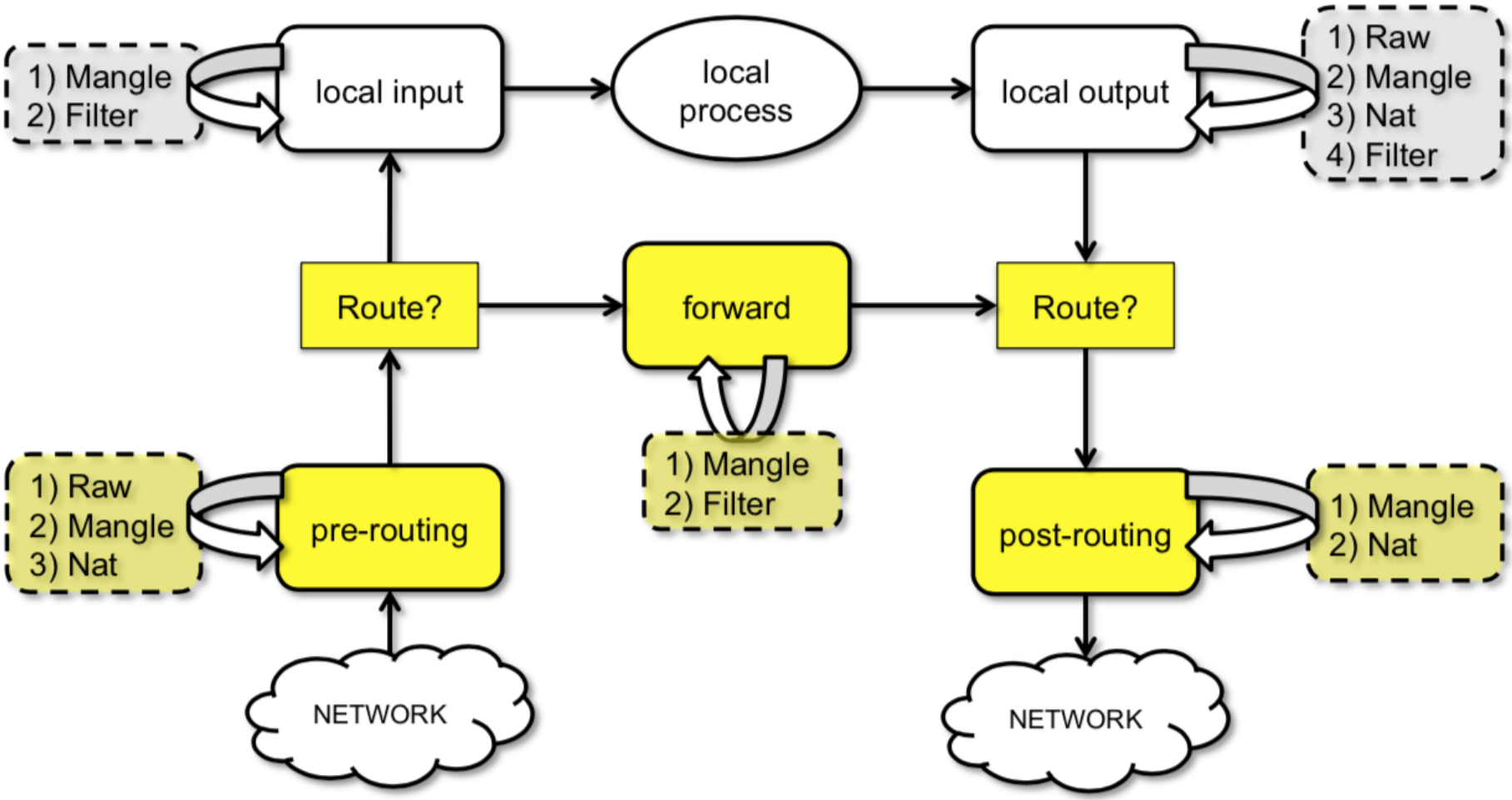
Netfilter - The Big Picture



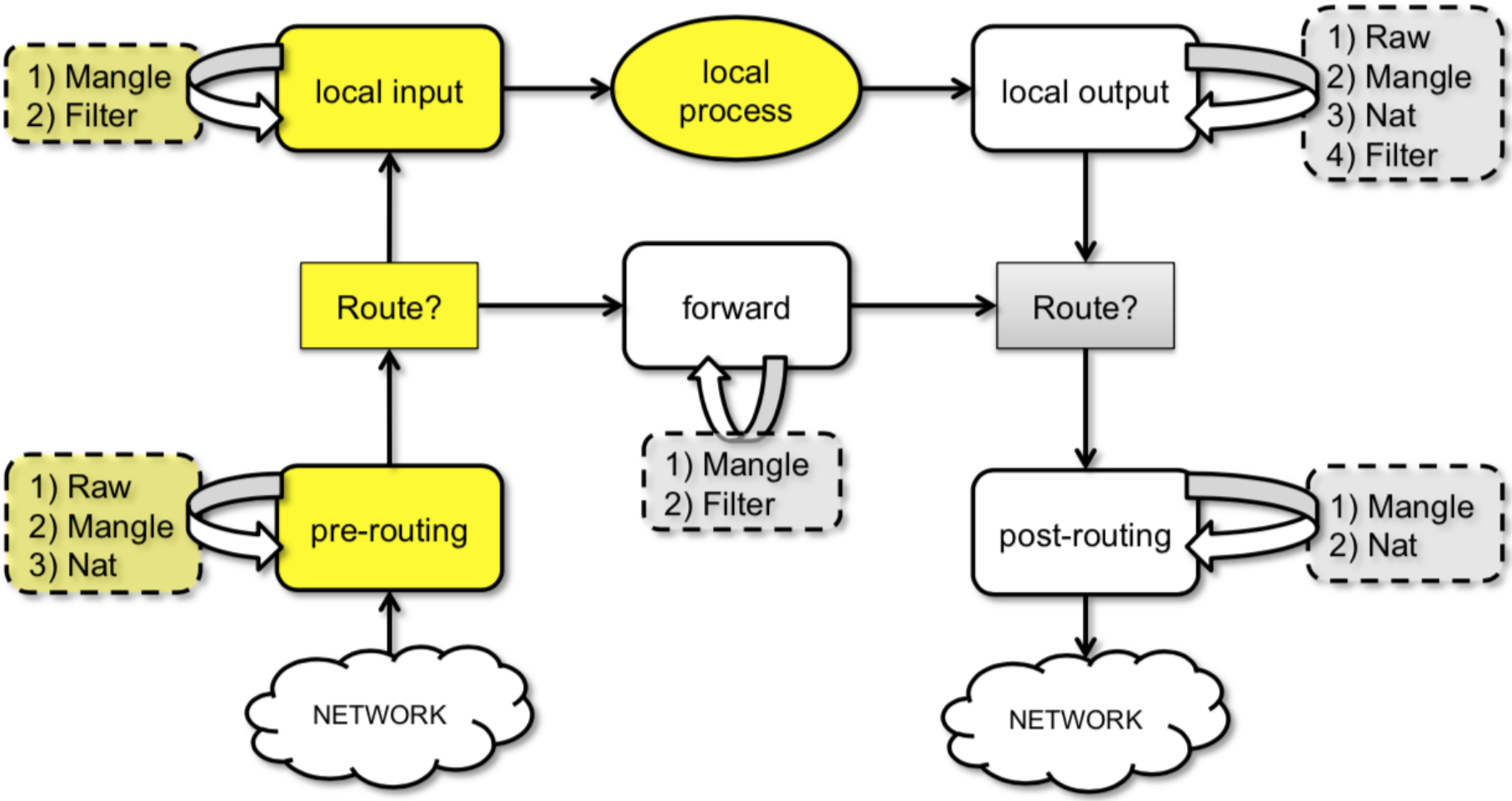
Netfilter - Locally Generated Packets



Netfilter - Forwarded Packets



Netfilter - Locally Addressed Packets



Connection tracking system 1/2

- Within NETFILTER packets can be related to tracked connections in four different so called states
  - **NEW**, **ESTABLISHED**, **RELATED**, **INVALID**
- With the “state” match we can easily control who or what is allowed to initiate new sessions. More later on…
- To load the conntrack module
  - modprobe `ip_conntrack`
- `/proc/net/ip_conntrack` gives a list of all the current entries in your conntrack database

Connection tracking system 2/2

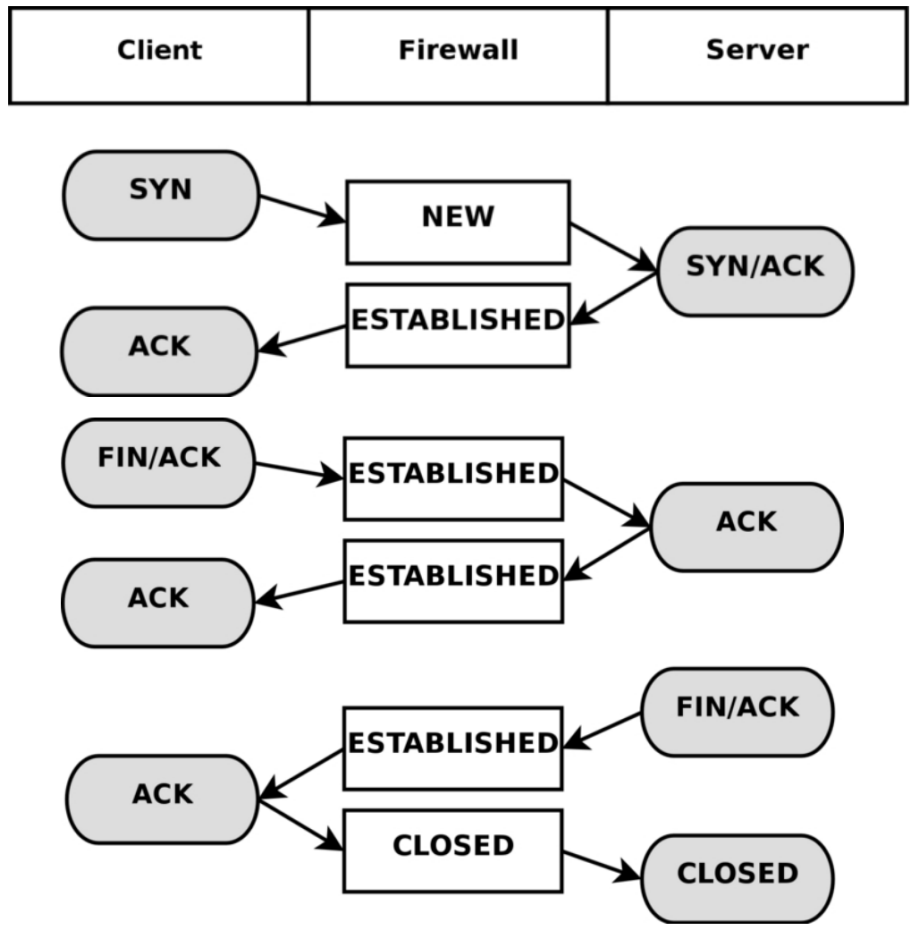
- `conntrack` util provides a full featured userspace interface to the netfilter connection tracking system that is intended to replace the old /proc/net/ip_conntrack interface
  - Commands: dump, create, get, delete, update, event, flush, stats…
  - `man conntrack`
  - `apt-get install contrack`

- Two internal tables
  - **conntrack**: it contains a list of all currently tracked connections through the system
  - **expect**: it is the table of expectations. Connection tracking expectations are the mechanism used to "expect" RELATED connections to existing ones. Expectations are generally used by "connection tracking helpers" (sometimes called application level gateways [ALGs]) for more complex protocols such as FTP, SIP, H.323

TCP Finite State Machine

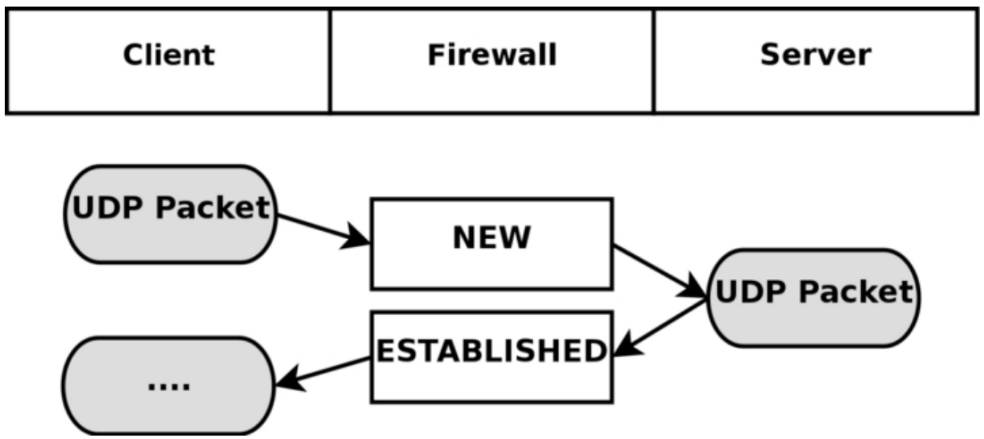
Opening a connection

Closing a connection



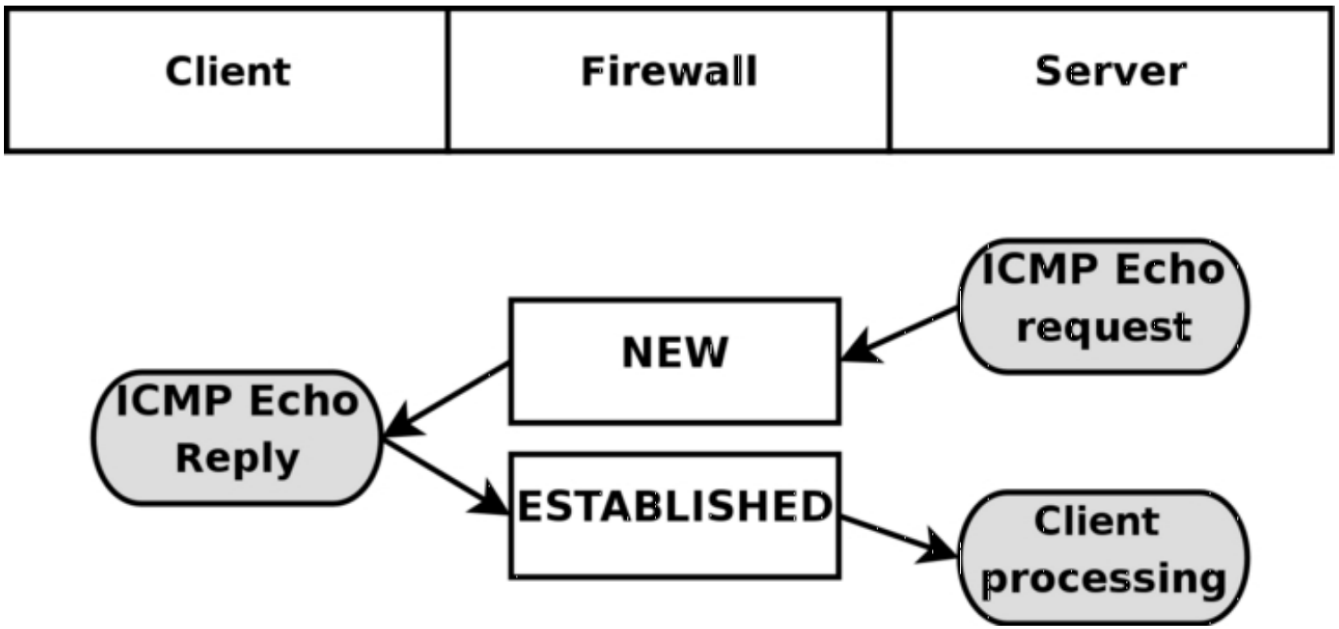
UDP Finite State Machine

Opening a connection

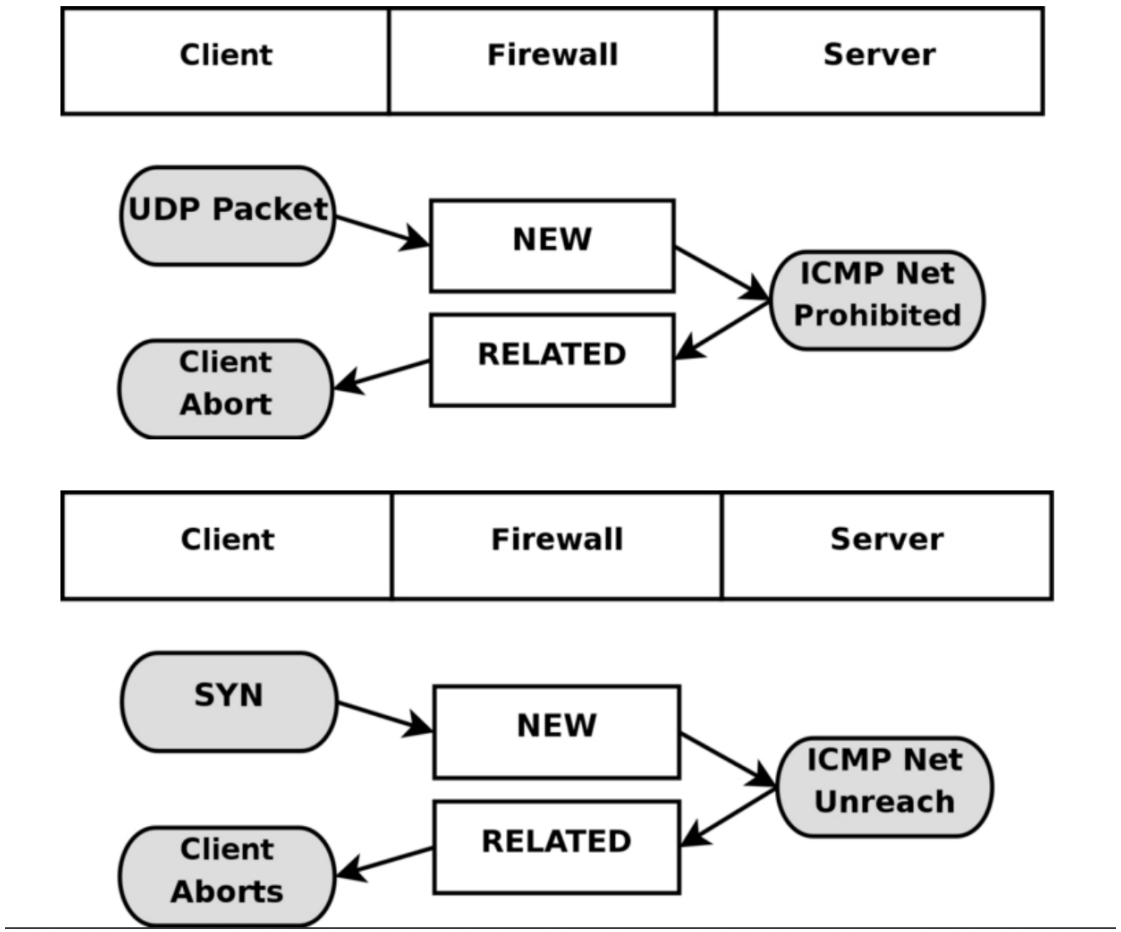


Closing a connection?

ICMP Finite State Machine



ICMP Related Finite State Machine



iptables intro 1/3

- **iptables** is the frontend of NETFILTER
- In other words, iptables is the userspace application used to configure the NETFILTER tables
- It is mainly used to add/remove rules to a chain (mapping of NETFILETR hooks) within a table

- General structure for adding remove a rule

```
iptables <command> <chain> <table> <match> <target>
iptables -A POSTROUTING -t nat -o eth0 -j MASQUERADE
```

iptables intro 2/3

- **`iptables`** is used to set up, maintain, and inspect the tables of IPv4 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
- Each rule specifies what to do with a packet that matches. This is called a **target**, which may be a jump to a user-defined chain in the same table

iptables intro 3/3

- Append, delete, insert, replace rules
  - `iptables [-t table] {-A|-D} chain rule-specification`
  - `iptables [-t table] -D chain rulenum`
  - `iptables [-t table] -I chain [rulenum] rule-specification`
  - `iptables [-t table] -R chain rulenum rule-specification`
- List, flush rules
  - `iptables [-t table] -S [chain [rulenum]]`
  - `iptables [-t table] -{F|L} [chain [rulenum]] [options...]`
- Create, delete, rename chains and set policy to a chain
  - `iptables [-t table] -N chain`
  - `iptables [-t table] -X [chain]`
  - `iptables [-t table] -E old-chain-name new-chain-name`
  - `iptables [-t table] -P chain target`
- Where:
  - `rule-specification = [matches...] [target]`
  - `match = -m matchname [per-match-options]`
  - `target = -j targetname [per-target-options]`

iptables targets

- A firewall rule specifies criteria for a packet and a target. If the packet does not match, the next rule in the chain is the examined;
- if the packet does match, then the next rule is specified by the value of the target (option -j), which can be the name of a user-defined chain or one of the special (standard) values
  - **ACCEPT** means to let the packet through (no other rules will be checked)
  - **DROP** means to drop the packet on the floor
  - **QUEUE** means to pass the packet to userspace
  - **RETURN** means stop traversing this chain and resume at the next rule in the previous (calling) chain. If the end of a built-in chain is reached or a rule in a built-in chain with target RETURN is matched, the target specified by the chain policy determines the fate of the packet

- More targets with target extensions. More later on...

iptables tables & chains 1/2

- **filter**: This is the default table (if no -t option is passed). It contains the built-in chains **INPUT** (for packets destined to local sockets), **FORWARD** (for packets being routed through the box), and **OUTPUT** (for locally-generated packets)
- **nat**: This table is consulted when a packet that creates a new connection is encountered. It consists of three built-ins: **PREROUTING** (for altering packets as soon as they come in), **OUTPUT** (for altering locally-generated packets before routing), and **POSTROUTING** (for altering packets as they are about to go out)
- **mangle**: This table is used for specialized packet alteration. Until kernel 2.4.17 it had two built-in chains: **PREROUTING** (for altering incoming packets before routing) and **OUTPUT** (for altering locally-generated packets before routing). Since kernel 2.4.18, three other built-in chains are also supported: **INPUT** (for packets coming into the box itself), **FORWARD** (for altering packets being routed through the box), and **POSTROUTING** (for altering packets as they are about to go out)
- **raw**: This table is used mainly for configuring exemptions from connection tracking in combination with the NOTRACK target. It registers at the netfilter hooks with higher priority and is thus called before ip_conntrack, or any other IP tables. It provides the following built-in chains: **PREROUTING** (for packets arriving via any network interface) **OUTPUT** (for packets generated by local processes)

iptables tables & chains 2/2

Queue Type	Queue Function	Packet Transformation Chain in Queue	Chain Function
Filter	Packet filtering	`FORWARD`	Filters packets to servers accessible by another NIC on the firewall.
		`INPUT`	Filters packets destined to the firewall.
		`OUTPUT`	Filters packets originating from the firewall
Nat	Network Address Translation	`PREROUTING`	Address translation occurs before routing. Facilitates the transformation of the destination IP address to be compatible with the firewall's routing table. Used with NAT of the destination IP address, also known as **destination NAT** or **DNAT**.
		`POSTROUTING`	Address translation occurs after routing. This implies that there was no need to modify the destination IP address of the packet as in pre-routing. Used with NAT of the source IP address using either one-to-one or many-to-one NAT. This is known as **source NAT**, or **SNAT**.
		`OUTPUT`	Network address translation for packets generated by the firewall. (Rarely used in SOHO environments)

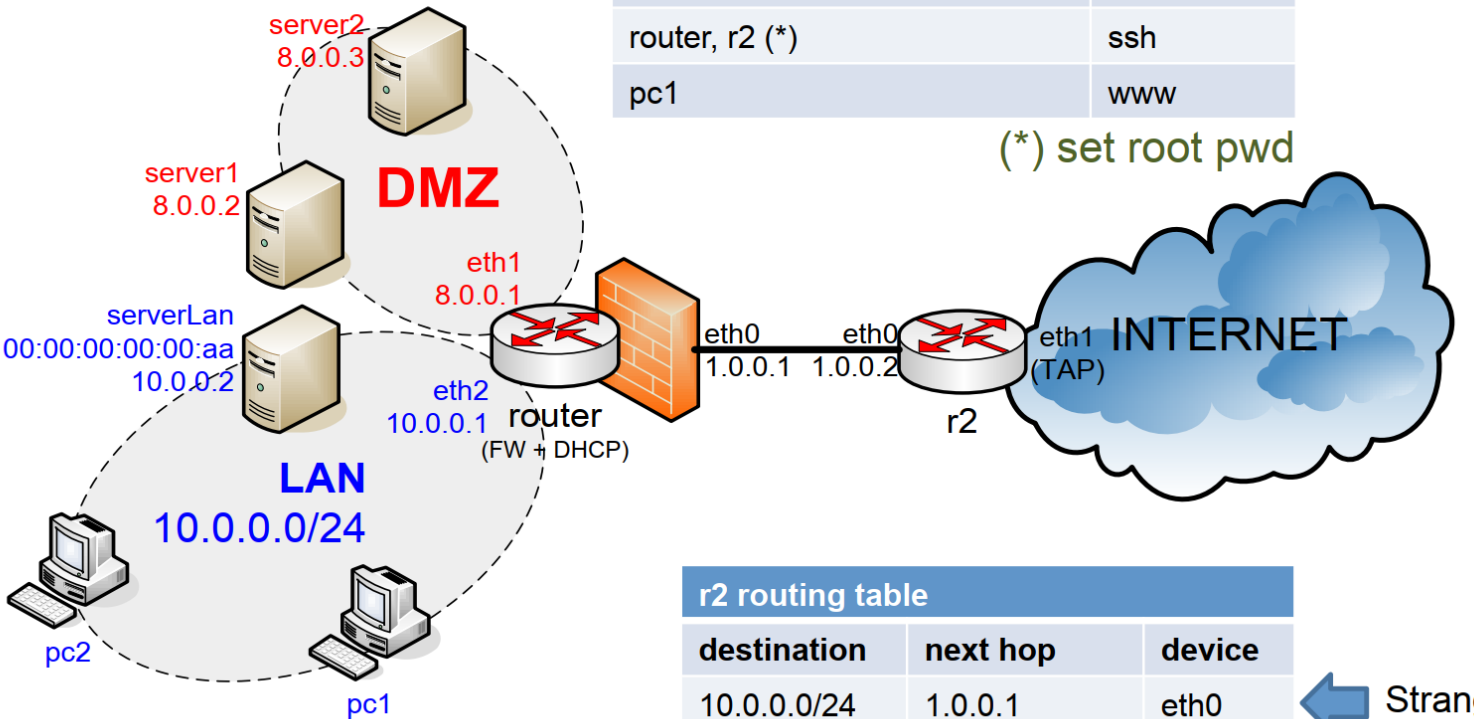
Mangle	TCP header modification	`PREROUTING`
		`POSTROUTING`
		`OUTPUT`
		`INPUT`
		`FORWARD`
			Modification of the TCP packet quality of service bits before routing occurs. (Rarely used in SOHO environments)

Basic match

The following parameters make up a match specification: (Iptables also support match extensions that provides many more match specification. More later on…)

- **`[!] -p, --protocol protocol`**: The protocol of the rule or of the packet to check. The specified protocol can be one of **tcp**, **udp**, **udplite**, **icmp**, **esp**, **ah**, **sctp** or **all**, or it can be a **numeric value**, representing one of these protocols or a different one. A protocol name from /etc/protocols is also allowed. The number zero is equivalent to all. The character “!” inverts the test
- **`[!] -s, --source address[/mask]`**: Source specification. Address can be either a network name, a hostname, a network IP address (with /mask), or a plain IP address. Hostnames will be resolved once only, before the rule is submitted to the kernel. Please note that specifying any name to be resolved with a remote query such as DNS is a really bad idea. The character “!” inverts the test
- **`[!] -d, --destination address[/mask]`**: Destination specification. See the description of the -s (source)
- **`[!] -i, --in-interface name`**: Name of an interface via which a packet was received (only for packets entering the INPUT, FORWARD and PREROUTING chains). When the "!" argument is used before the interface name, the sense is inverted. If the interface name ends in a "+", then any interface which begins with this name will match. If this option is omitted, any interface name will match.
- **`[!] -o, --out-interface name`**: Name of an interface via which a packet is going to be sent (for packets entering the FORWARD, OUTPUT and POSTROUTING chains). When the "!" argument is used before the interface name, the sense is inverted. If the interface name ends in a "+", then any interface which begins with this name will match. If this option is omitted, any interface name will match
- **`[!] -f, --fragment`**: This means that the rule only refers to second and further fragments of fragmented packets. Since there is no way to tell the source or destination ports of such a packet (or ICMP type), such a packet will not match any rules which specify them. When the "!" argument precedes the "-f" flag, the rule will only match head fragments, or unfragmented packets

Topology



running servers	
server1, server2, serverLan (*)	ssh, www
router, r2 (*)	ssh
pc1	www

(*) set root pwd

r2 routing table		
destination	**next hop**	**device**
10.0.0.0/24	1.0.0.1	eth0
8.0.0.0/24	1.0.0.1	eth0
1.0.0.0/24	*	eth0
default	host-machine	eth1

Strange…
Just to make things work, for now…

Basic commands

- Show rules in the filter table with numeric output and rule numbers
```
router# iptables -L -n --line-numbers
```
- Flush rules in the filter table
```
router# iptables -F
```
- Set the policy for the FORWARD chain in the filter table
```
router# iptables -P FORWARD DROP
```
- Allow all packets coming from LAN toward anywere
```
router# iptables -A FORWARD -i eth2 -j ACCEPT
```
- Allow all packets from the Internet to DMZ
```
router# iptables -A FORWARD -i eth0 -o eth1 -j ACCEPT
```
- Allow packets from server1 to LAN
```
router# iptables -A FORWARD -o eth2 -s 8.0.0.2 -i eth1 -j ACCEPT
```

Rule order is important!

- DROP all incoming packets from LAN except from serverLAN

```
router# iptables -A INPUT -s 10.0.0.0/24 -i eth2 -j DROP
router# iptables -A INPUT -s 10.0.0.2 -i eth2 -j ACCEPT
```

It won't work! Why?

Because the second rule is appended and then a packet from 10.0.0.2 will be dropped anyway as it will match the first rule

- Solutions:
  - Change the order ☺
  - Insert the second rule instead of appending it
  - set DROP as default policy for the INPUT chain and append the 1 ACCEPT rule

Match extensions

- iptables can use extended packet matching modules
  - implicitly loaded when -p or --protocol is specified
  - or with -m or --match options, followed by the matching module name
- After an extended match is specified, various extra command line options become available, depending on the specific module
  - iptables -A INPUT -p tcp --sport 9000 -j DROP
  - iptables -A INPUT -m addrtype --dst-type MULTICAST -j DROP
- You can specify multiple extended match modules in one line, and you can use the -h or --help options after the module has been specified to receive help specific to that module
- Man iptables for all match extensions

Match extensions

- -p tcp: matches IP packets with protocol=TCP
  - --sport: matches the TCP source port
  - --dport: matches the TCP destination port
  - --tcp-flags: matches the TCP header flags…
  - --syn: matches packets with the SYN flag set

Example:

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

- -p udp: matches IP packets with protocol=UDP
  - --sport: matches the UDP source port
  - --dport: matches the UDP destination port

Example:

```
iptables -A OUTPUT -p udp --dport 53 -j DROP
```

Example: Forward SSH traffic

Allow forwarding of SSH traffic from clients inside the LAN and servers on the internet

```
router# iptables -A FORWARD -i eth2 -p tcp --dport
22 -j ACCEPT
```

```
router# iptables -A FORWARD -i eth0 -p tcp --sport
22 -j ACCEPT
```

We would like to accept traffic from the internet with source port 22 only if related to a previously established connection from LAN…

State match extensions

This module, when combined with connection tracking, allows access to the connection tracking state for this packet

`-m state [!] --state` *`state`*

Where state is a comma separated list of the connection states to match.

- **INVALID** meaning that the packet could not be identified for some reason which includes running out of memory and ICMP errors which don't correspond to any known connection
- **ESTABLISHED** meaning that the packet is associated with a connection which has seen packets in both directions
- **NEW** meaning that the packet has started a new connection, or otherwise associated with a connection which has not seen packets in both directions
- **RELATED** meaning that the packet is starting a new connection, but is associated with an existing connection, such as an FTP data transfer, or an ICMP error

-m ctstate provides additional features. Man iptables for more…

Multiport match

- Match multiple destination ports (e.g. tcp)
  - -p tcp -m multiport --dports *port1,port2,…,portn*
- Match multiple source ports (e.g. udp)
  - -p udp -m multiport --sports *port1,port2,…,portn*
- Match multiple ports (both src and dst) (tcp)
  - -p tcp -m multiport --ports *port1,portb:portc*