University of Rome Tor Vergata
ICT and Internet Engineering

Network and System Defense

Alessandro Pellegrini, Angelo Tulumello

A.A. 2023/2024

Lecture 5: Firewall and Packet Classification Algorithms

Angelo Tulumello

Slides by Marco Bonola

Table of Contents

Firewall Overview

Source: William Stallings and Lawrie Brown, "Computer Security: Principles and Practice", chapter 9

The Need For Firewalls

- Internet connectivity is essential
  - However it creates a threat
- Effective means of protecting LANs
- Inserted between the premises network and the Internet to establish a controlled link
  - Can be a single computer system or a set of two or more systems working together
- Used as a perimeter defense
  - Single choke point to impose security and auditing
  - Insulates the internal systems from external network

Tramite Firewall proteggiamo la LAN da problematiche provenienti da INTERNET.
L'obiettivo è controllare ciò che si interfaccia con la nostra rete.

Firewall Characteristics: Design goals

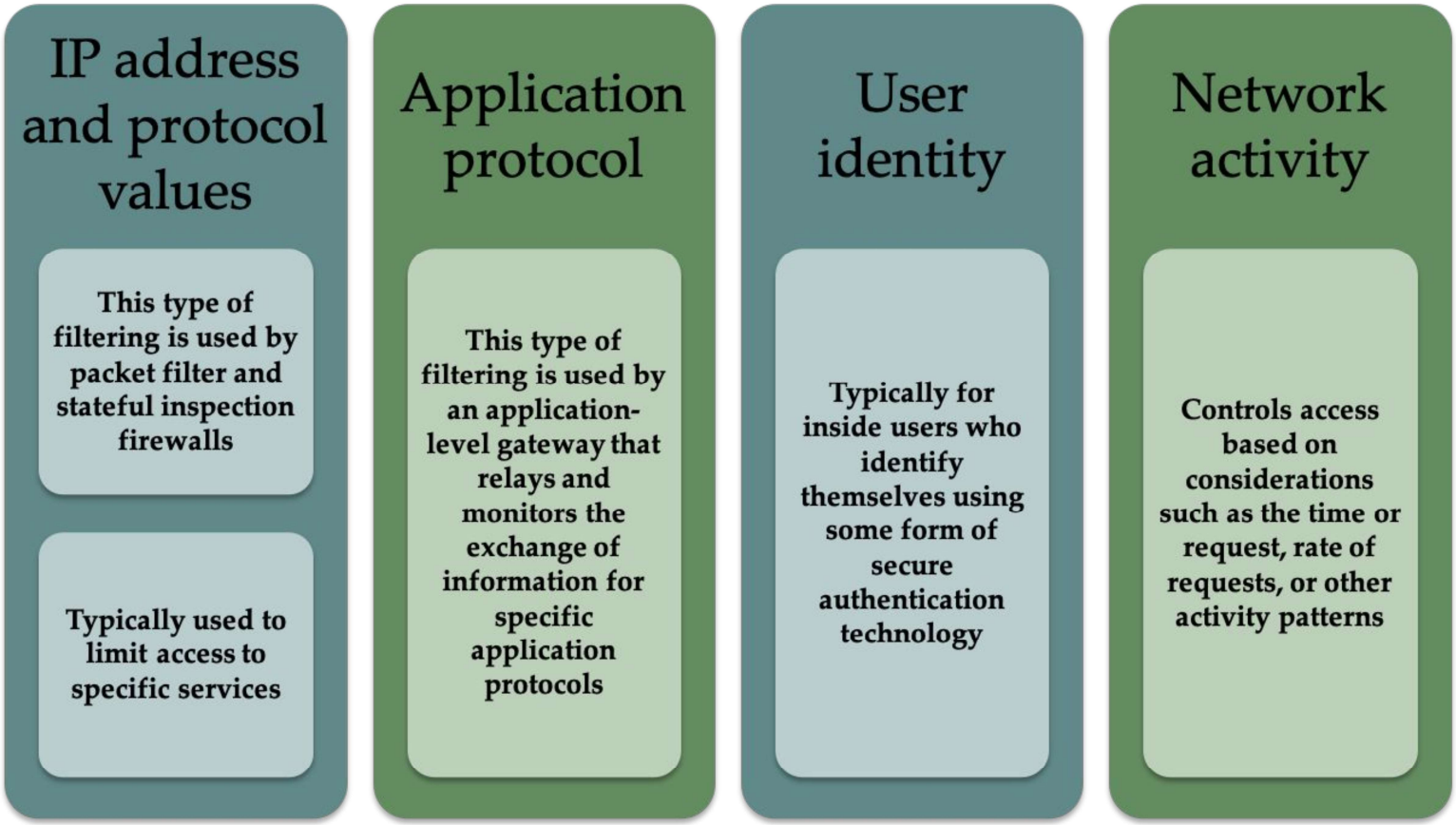
- All traffic from inside to outside, and vice versa, must pass through the firewall
- Only authorized traffic as defined by the local security policy will be allowed to pass
- The firewall itself is immune to penetration

Firewall Access Policy

- A critical component in the planning and implementation of a firewall is specifying a suitable access policy
  - This lists the types of traffic authorized to pass through the firewall
  - Includes address ranges, protocols, applications and content types
- This policy should be developed from the organization's information security risk assessment and policy
- Should be developed from a broad specification of which traffic types the organization needs to support
  - Then refined to detail the filter elements which can then be implemented within an appropriate firewall topology

Voglio una descrizione del traffico che consento e che non consento, in entrambe le direzioni.

Firewall Filter Characteristics



Firewall Capabilities And Limits

Capabilities

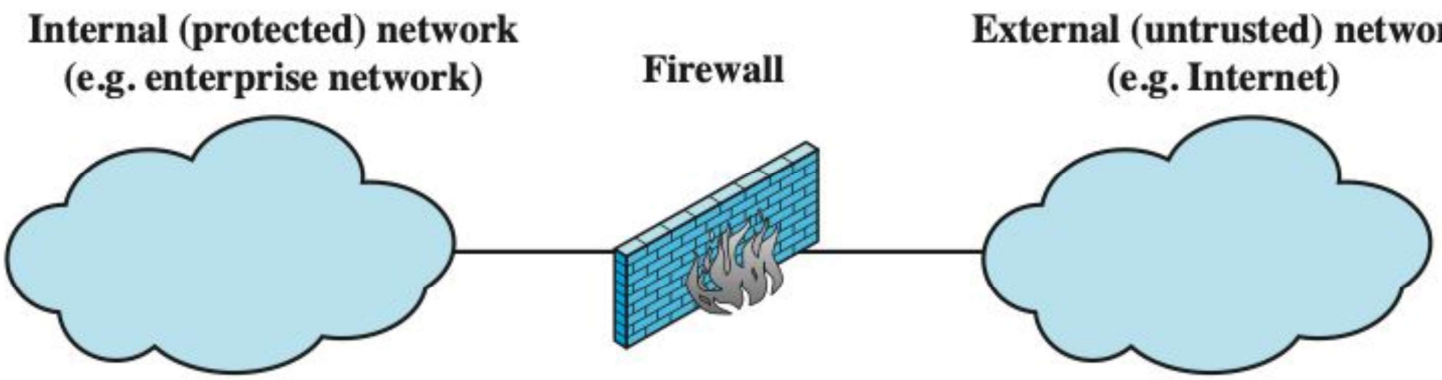
tra i vantaggi principali, abbiamo un unico punto di accesso (quindi o si entra da lì o non si entra), il monitoraggio degli eventi, e la compatibilità con IPSec

- ❏ Defines a single choke point
- ❏ Provides a location for monitoring security events
- ❏ Convenient platform for several Internet functions that are not security related
- ❏ Can serve as the platform for IPSec

Limitations

- ❏ Cannot protect against attacks bypassing firewall
- ❏ May not protect fully against internal threats
- ❏ Improperly secured wireless LAN can be accessed from outside the organization
- ❏ Laptop or portable storage device may be infected outside the corporate network then used internally

Tra i limiti principali:
- se bypasso il firewall, ad esempio con un tunnel, sono dentro.
- non ho protezione da attacchi INTERNI.



(a) General model

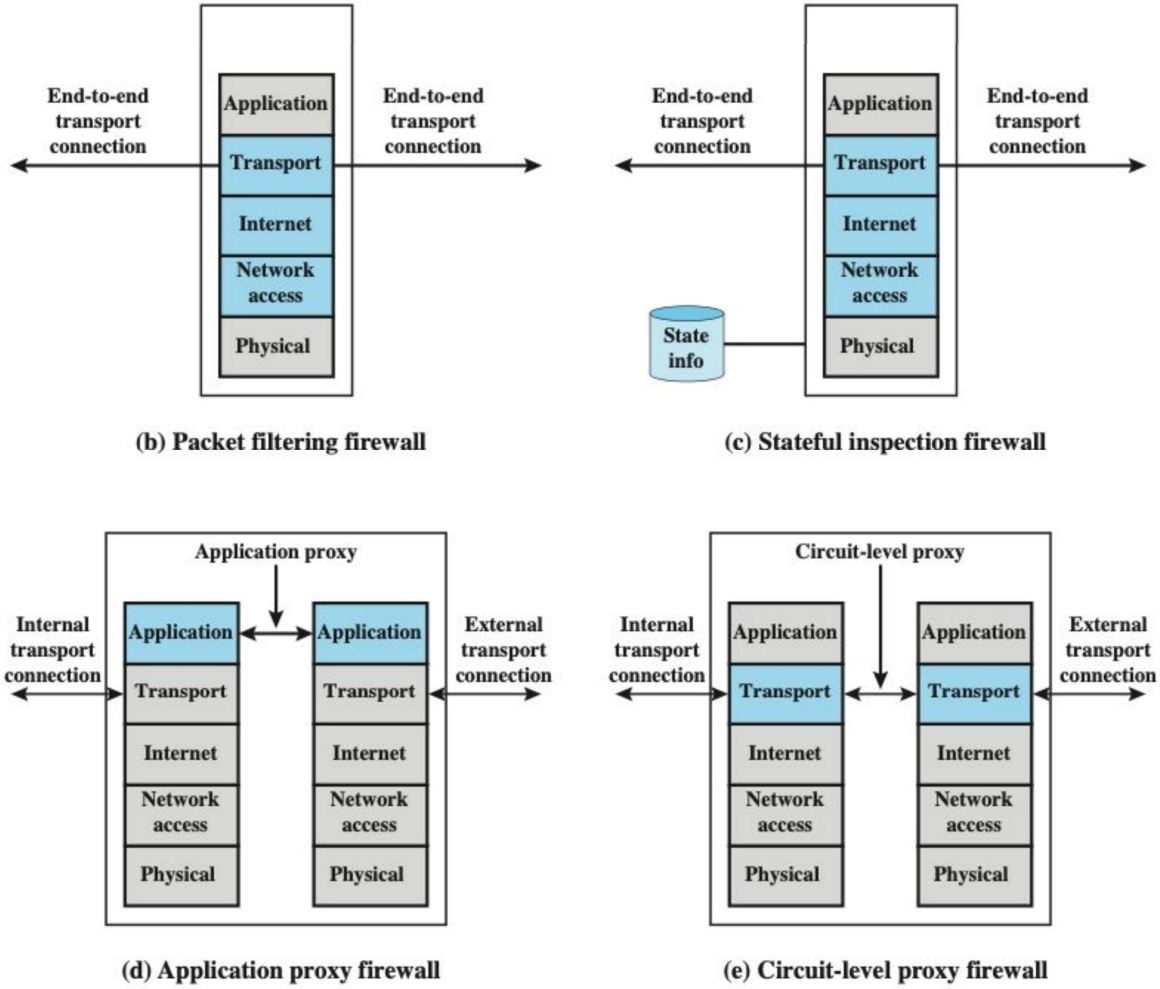


Figure 9.1 Types of Firewalls

Packet Filtering Firewall

- Applies rules to ***each incoming and outgoing IP packet***
  - Typically a list of rules based on matches in the IP or transport header
  - Forwards or discards the packet based on rules match:
    - Source IP address, Destination IP address, Source and destination transport-level address, IP protocol field, Interface
- Two default policies:
  - ***Discard*** - prohibit unless expressly permitted
    - More conservative, controlled, visible to users
  - ***Forward*** - permit unless expressly prohibited
    - Easier to manage and use but less secure

Con la prima, blocco tutto ciò che non è espressamente scritto.
Con la seconda, accetto tutto ciò che non è espressamente scritto.

Packet-Filtering Examples

Rule	Direction	Src address	Dest addresss	Protocol	Dest port	Action
1	In	External	Internal	TCP	25	Permit
2	Out	Internal	External	TCP	>1023	Permit
3	Out	Internal	External	TCP	25	Permit
4	In	External	Internal	TCP	>1023	Permit
5	Either	Any	Any	Any	Any	Deny

Packet Filter: Advantages And Weaknesses

- ***Advantages***
  - Simplicity
  - Typically transparent to users and are very fast
- ***Weaknesses***
  - Cannot prevent attacks that employ application specific vulnerabilities or functions
  - Limited logging functionality
  - Do not support advanced user authentication
  - Vulnerable to attacks on TCP/IP protocol bugs
  - Improper configuration can lead to breaches

Stateful Inspection Firewall

Regole per il traffico TCP a partire da una directory di connessioni TCP in uscita.

Tightens rules for TCP traffic by creating a ***directory of outbound TCP connections***

- ***There is an entry for each currently established connection***
- Packet filter allows incoming traffic to high numbered ports only for those packets that fit the profile of one of the entries in this directory

Reviews packet information but also records information about TCP connections

- **Keeps track of TCP sequence numbers** to prevent attacks that depend on the sequence number
- ***Inspects data for higher protocols*** like FTP, IM and SIPS commands

Stateful firewalls are applied also to ***connectionless protocols (e.g. UDP)***

Connection State Table

Source Address	Source Port	Destination Address	Destination Port	Connection State
192.168.1.100	1030	210.9.88.29	80	Established
192.168.1.102	1031	216.32.42.123	80	Established
192.168.1.101	1033	173.66.32.122	25	Established
192.168.1.106	1035	177.231.32.12	79	Established
223.43.21.231	1990	192.168.1.6	80	Established
219.22.123.32	2112	192.168.1.6	80	Established
210.99.212.18	3321	192.168.1.6	80	Established
24.102.32.23	1025	192.168.1.6	80	Established
223.21.22.12	1046	192.168.1.6	80	Established

Application-Level Gateway

- Also called ***application proxy***
- Acts as a relay of application-level traffic
  - User contacts gateway using a TCP/IP application
  - User is authenticated
  - Gateway contacts application on remote host and relays TCP segments between server and user
- Must have proxy code for each application
  - May restrict application features supported
- Tend to be more secure than packet filters
- Disadvantage is the additional processing overhead on each connection

Gestisco il traffico a livello di applicazione, devo usare un proxy per ogni applicazione che ho, e questo può essere dispendioso se ho molte applicazioni.

Circuit-Level Gateway

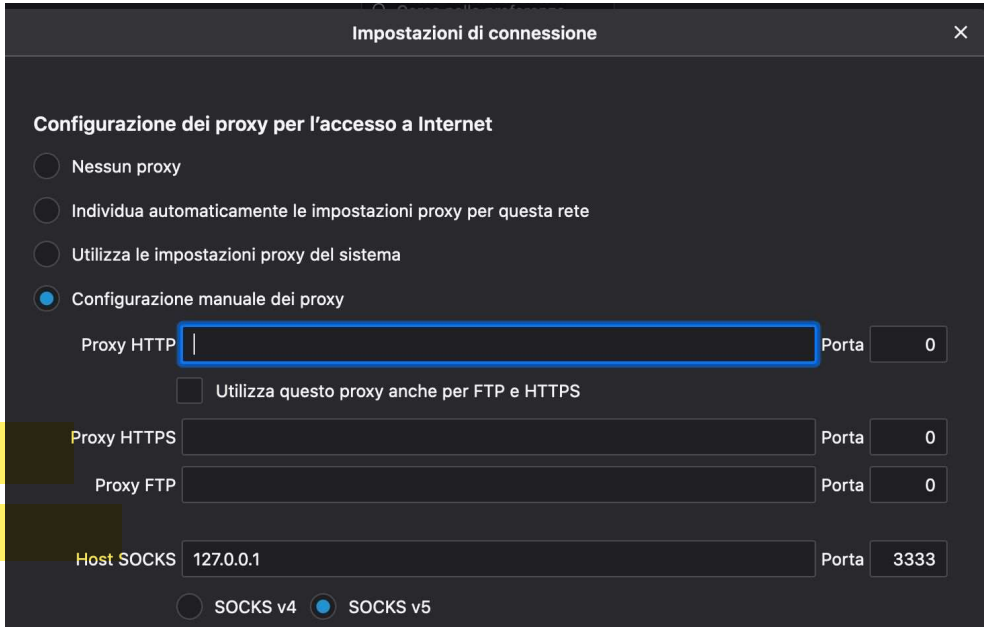
- AKA ***Circuit level proxy***
  - Sets up two TCP connections, one between itself and a TCP user on an inner host and one on an outside host
  - Relays TCP segments from one connection to the other without examining contents
  - Security function consists of determining which connections will be allowed
- Typically used when inside users are trusted
  - May use application-level gateway inbound and circuit-level gateway outbound
  - Lower overheads

Chiamato anche Circuit level PROXY, perchè è come il proxy ma a livello di trasporto!

SOCKS Circuit-Level Gateway

(il nonno di NordVPN)

- ***SOCKS v5*** defined in RFC1928
- Designed to provide a framework for client-server applications in TCP/UDP domains to conveniently and securely use the services of a network firewall
- Client application contacts SOCKS server, authenticates, sends relay request
- Server evaluates and either establishes or denies the connection



firefox proxy SOCKS configuration

Host-Based Firewalls

- Used to secure ***an individual host***
- Available in operating systems or can be provided as an add-on package
- Filter and restrict packet flows
- Common location is a ***server***
- Advantages (su misura)
  - Filtering rules can be tailored to the host environment
  - Protection is provided independent of topology
  - Provides an additional layer of protection

Personal Firewall

- Controls traffic between ***a personal computer or workstation and the Internet or enterprise network***
- For both home or corporate use
- Typically is a software module on a personal computer
- Can be housed in a router that connects all of the home computers to a DSL, cable modem, or other Internet interface
- Typically much less complex than server-based or standalone firewalls
- Primary role is to deny unauthorized remote access
- May also monitor outgoing traffic to detect and block ***worms and malware activity***

Ad esempio, se installo un programma su Windows che richiede l'uso di Internet, ci viene chiesto di consentire il traffico internet per questo programma, ovvero creare una regola nel firewall per consentire ciò!

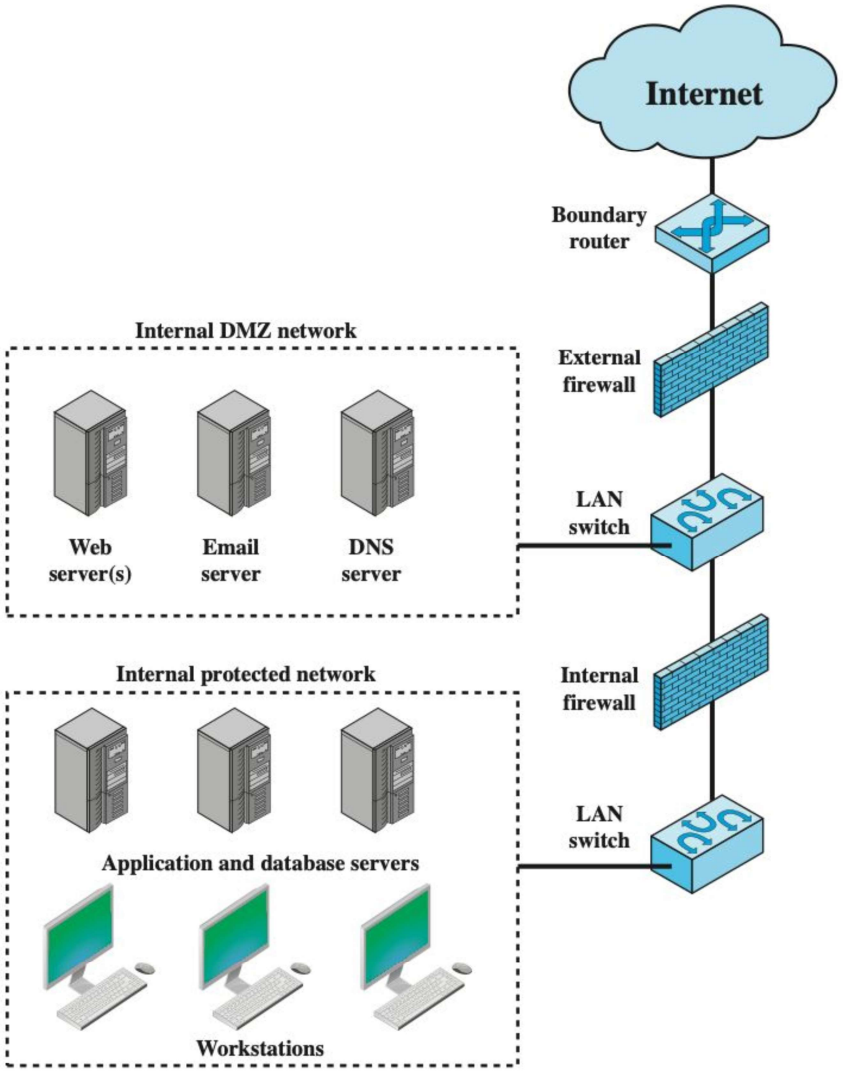


Figure 9.2 Example Firewall Configuration

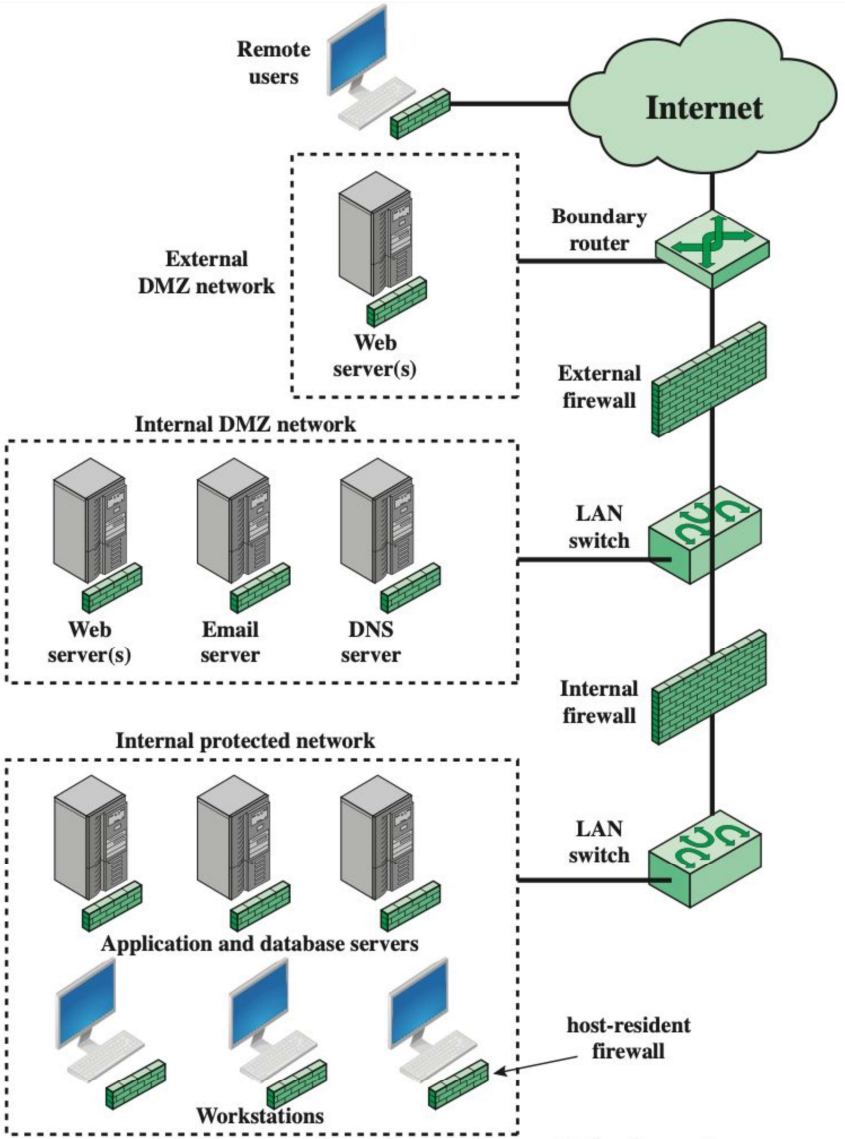


Figure 9.4 Example Distributed Firewall Configuration

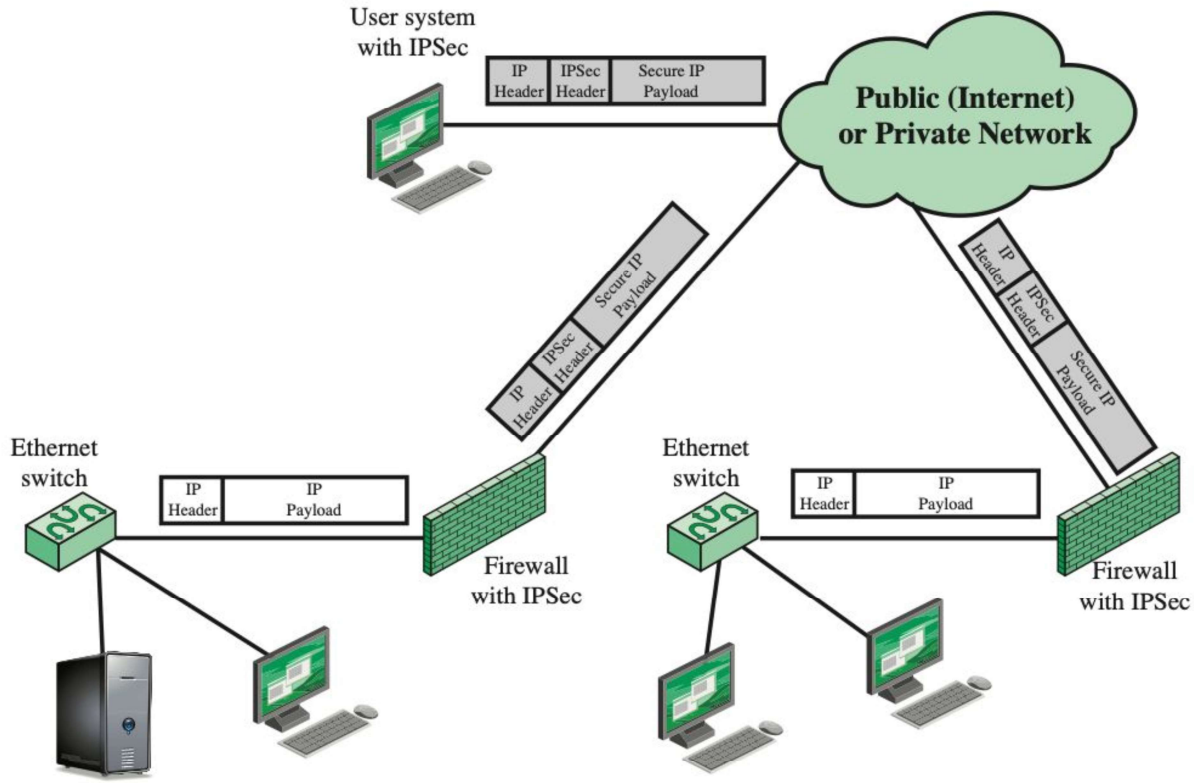


Figure 9.3 A VPN Security Scenario

A “real world” implementation: Linux’s NETFILTER and iptables

Hook: un insieme di tecniche utilizzate per intercettare chiamate di funzioni, messaggi o eventi scambiati tra componenti software. Il codice che gestisce tali chiamate di funzioni, eventi o messaggi intercettati è chiamato hook.

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***, ***FORWARD***, ***LOCAL_OUT*** ***POST_ROUTING***

Uso NETFILTER per policy legate ai Firewall. Espongo Hooks/Entry Point per applicare il " mangling", cioè delle modifiche, sull'header.

hook invocation from linux/net/ipv4/ip_input.c

```c
/*
 *      Deliver IP Packets to the higher protocol layers.
 */
int ip_local_deliver(struct sk_buff *skb)
{
        /*
         *      Reassemble IP fragments.
         */
        struct net *net = dev_net(skb->dev);

        if (ip_is_fragment(ip_hdr(skb))) {
                if (ip_defrag(net, skb, IP_DEFRAG_LOCAL_DELIVER))
                        return 0;
        }

        return NF_HOOK(NFPROTO_IPV4, NF_INET_LOCAL_IN,
                       net, NULL, skb, skb->dev, NULL,
                       ip_local_deliver_finish);
}
EXPORT_SYMBOL(ip_local_deliver);
```

LOCAL_INPUT

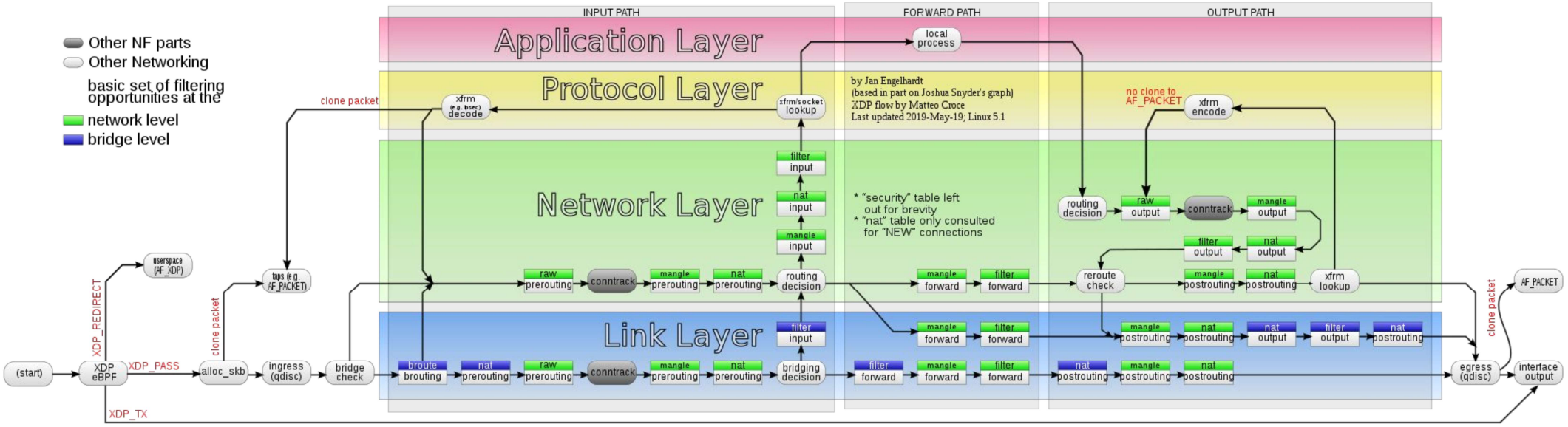
```c
int ip_rcv(struct sk_buff *skb, struct net_device *dev, struct packet_type *pt,
           struct net_device *orig_dev)
{
        struct net *net = dev_net(dev);

        skb = ip_rcv_core(skb, net);
        if (skb == NULL)
                return NET_RX_DROP;

        return NF_HOOK(NFPROTO_IPV4, NF_INET_PRE_ROUTING,
                       net, NULL, skb, dev, NULL,
                       ip_rcv_finish);
}
```

PRE_ROUTING

NETFILTER

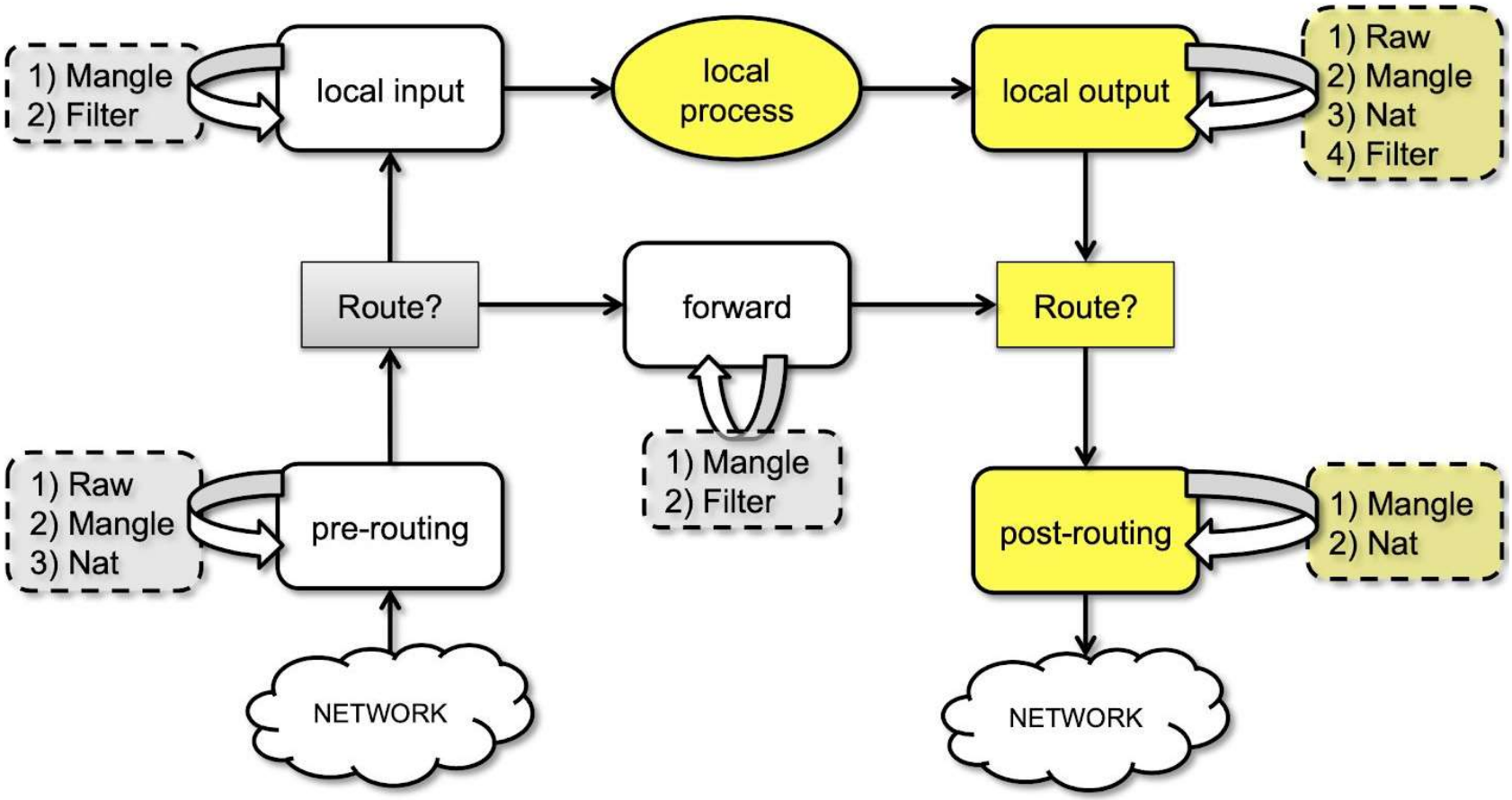
- All packet intercepted by the hooks pass through a sequence of ***built-in tables*** (queues) for processing
- Each of these queues is dedicated to a particular type of packet activity and is controlled by an associated packet transformation/filtering chain
- ***4 built-in tables***
  - **Filter**: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

Packet Flow in NETFILTER and General Networking

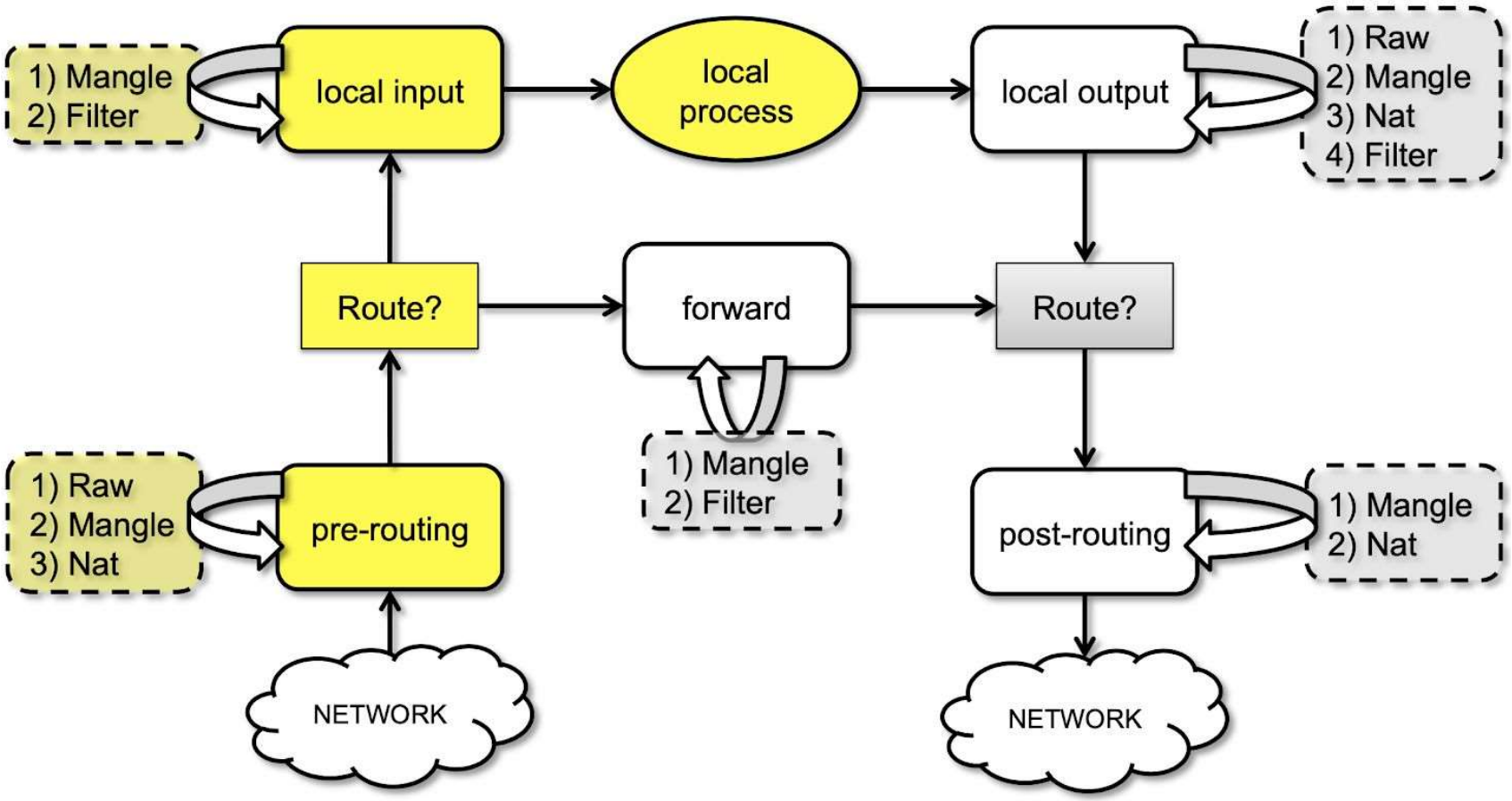


https://upload.wikimedia.org/wikipedia/commons/3/37/Netfilter-packet-flow.svg

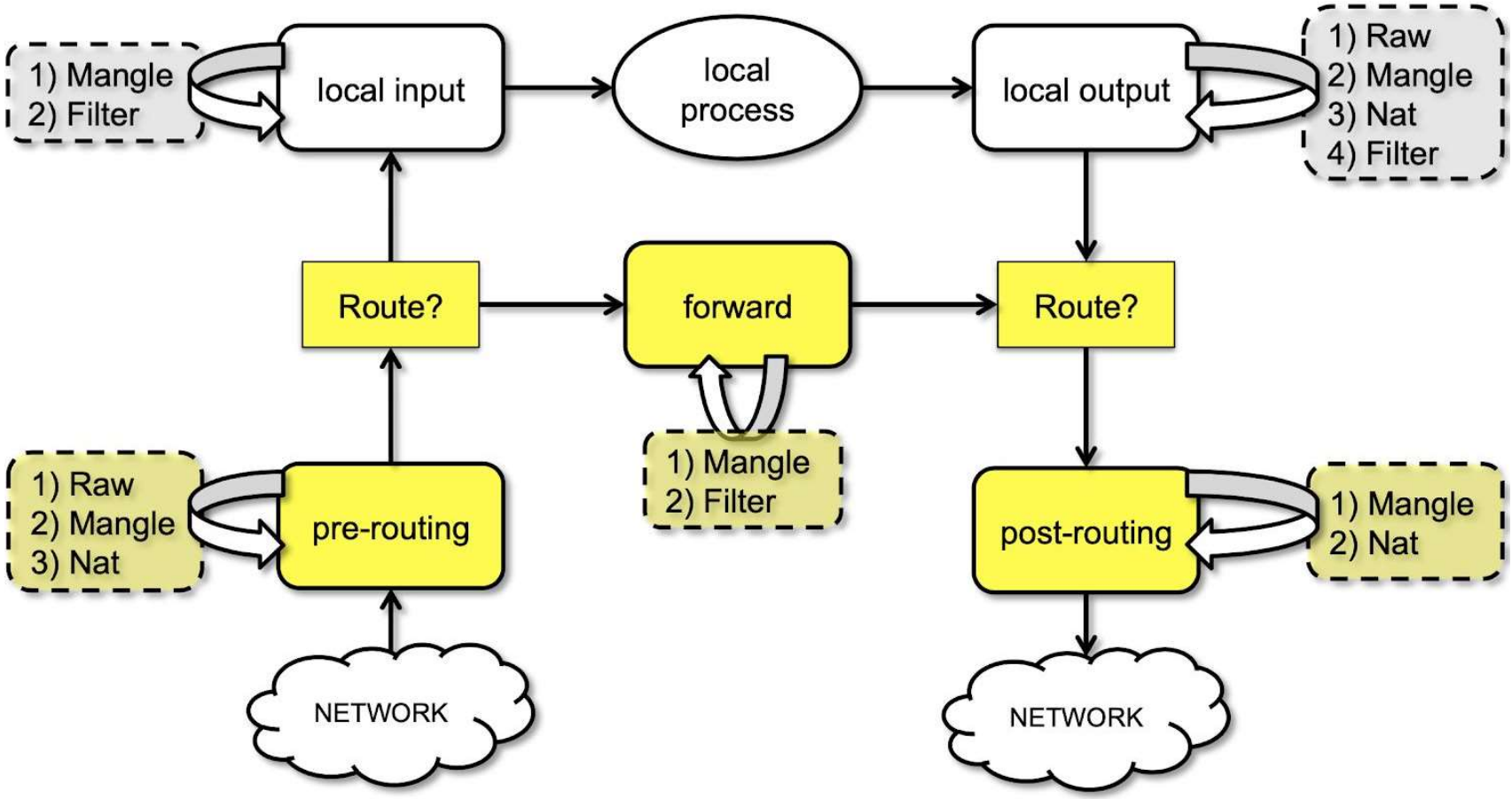
Packets sent from the local host



Packets sent to a local address



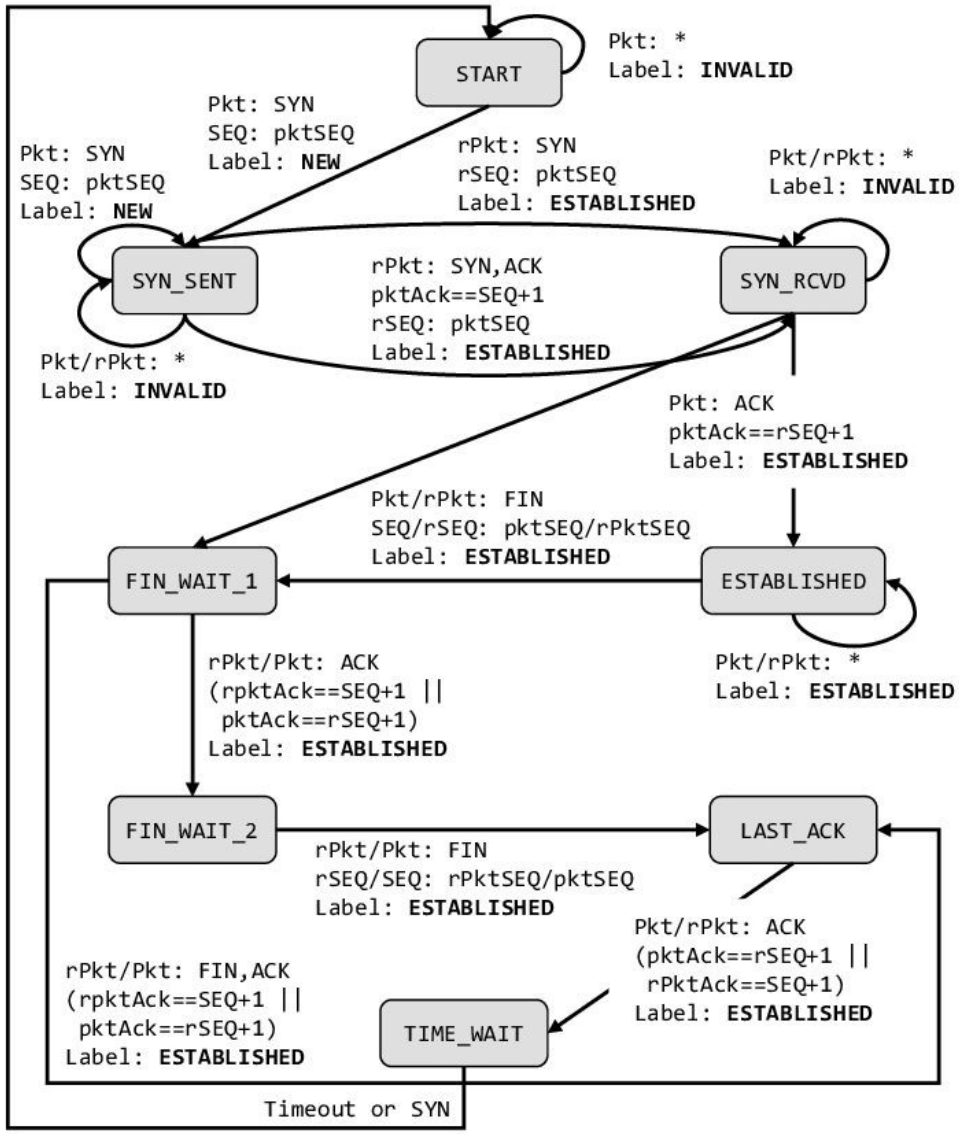
Forwarded Packets



Connection Tracking

- 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
  - newer distributions does not have this file, use `conntrack -L` instead

TCP state machine



Linux conntrack

```
marlon@marlon-vmxbn:~$ sudo cat /proc/net/ip_conntrack
[sudo] password for marlon:

udp      17 28 src=172.16.166.156 dst=172.16.166.2 sport=43716 dport=53 src=172.16.166.2
dst=172.16.166.156 sport=53 dport=43716 mark=0 use=2

tcp      6 431951 ESTABLISHED src=172.16.166.156 dst=172.16.166.2 sport=48680 dport=9999
src=172.16.166.2 dst=172.16.166.156 sport=9999 dport=48680 [ASSURED] mark=0 use=2

udp      17 28 src=172.16.166.156 dst=172.16.166.2 sport=44936 dport=53 src=172.16.166.2
dst=172.16.166.156 sport=53 dport=44936 mark=0 use=2

udp      17 19 src=172.16.166.156 dst=224.0.0.251 sport=5353 dport=5353 [UNREPLIED]
src=224.0.0.251 dst=172.16.166.156 sport=5353 dport=5353 mark=0 use=2

tcp      6 431487 ESTABLISHED src=172.16.166.156 dst=172.16.166.1 sport=43733 dport=139
src=172.16.166.1 dst=172.16.166.156 sport=139 dport=43733 [ASSURED] mark=0 use=2

udp      17 28 src=172.16.166.156 dst=172.16.166.2 sport=43581 dport=53 src=172.16.166.2
dst=172.16.166.156 sport=53 dport=43581 mark=0 use=2
```

conntrack events

```
root@marlon-vmxbn:/home/marlon# conntrack --event
[NEW] udp      17 30 src=172.16.166.156 dst=172.16.166.156 sport=47282 dport=4444 [UNREPLIED]
src=172.16.166.156 dst=172.16.166.156 sport=4444 dport=47282

[DESTROY] udp      17 src=172.16.166.2 dst=172.16.166.156 sport=5353 dport=5353 [UNREPLIED]
src=172.16.166.156 dst=172.16.166.2 sport=5353 dport=5353

[DESTROY] udp      17 src=172.16.166.156 dst=224.0.0.251 sport=5353 dport=5353 [UNREPLIED]
src=224.0.0.251 dst=172.16.166.156 sport=5353 dport=5353

[DESTROY] udp      17 src=172.16.166.1 dst=224.0.0.251 sport=5353 dport=5353 [UNREPLIED]
src=224.0.0.251 dst=172.16.166.1 sport=5353 dport=5353

[NEW] tcp      6 120 SYN_SENT src=172.16.166.156 dst=160.80.103.147 sport=45696 dport=80
[UNREPLIED] src=160.80.103.147 dst=172.16.166.156 sport=80 dport=45696

[UPDATE] tcp      6 60 SYN_RECV src=172.16.166.156 dst=160.80.103.147 sport=45696 dport=80
src=160.80.103.147 dst=172.16.166.156 sport=80 dport=45696

[UPDATE] tcp      6 432000 ESTABLISHED src=172.16.166.156 dst=160.80.103.147 sport=45696
dport=80 src=160.80.103.147 dst=172.16.166.156 sport=80 dport=45696 [ASSURED]
```

Application Level Gateway (ALG) related state

- protocols like FTP, IRC, and others carry information within the actual data payload of the packets, and hence requires special connection tracking helpers to enable it to function correctly
- For example, FTP first opens up a single connection that is called the FTP control session and negotiate the opening of the data session over a different socket
- When a connection is done actively, the FTP client sends the server a port and IP address to connect to. After this, the FTP client opens up the port and the server connects to that specified port from a random unprivileged port (>1024) and sends the data over it
- A special NETFILTER conntrack helper can read the FTP control payload and read the “data port”
- The new data socket will be considered as RELATED

iptables

- ***iptables is the front end 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 NETFILTER hooks) within a table
- General structure for adding a rule:

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

iptables

- **`iptables`** is used to set up, maintain, and inspect the tables of IPv4 packet filter rules in the Linux kernel
- Several different ***chains*** may be defined. Each chain contains a number of built-in ***tables***
  - a chain could be also custom, i.e. defined by the user
- In each table, there 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
  - or a standard *action* (DROP, ACCEPT…)

Posso creare Chains, ma non HOOKS, i quali sono interni al kernel.

iptables commands

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]
```

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)

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 matches

- in this class we are going to use the following iptables matches
  - IP source address (also net supported): **`-s $address`**
  - IP destination address (also net supported): **`-d $address`**
  - IP protocol: **`-p tcp|udp`** [**`--sport $sp]`** [**`--dport $dp]`**
  - input interface: **`-i $iif`**
  - output interface: **`-o $oif`**
  - state: **`-m state --state NEW|ESTABLISHED`**

Example (not all matches are mandatory. matches can be arbitrary combined. all matches in the same rule are in logical AND)

```
iptables -A INPUT -s 10.0.0.100 -d 10.0.0.1 -p tcp --dport
80 --sport 54400 -j ACCEPT
```

that's all with the theory

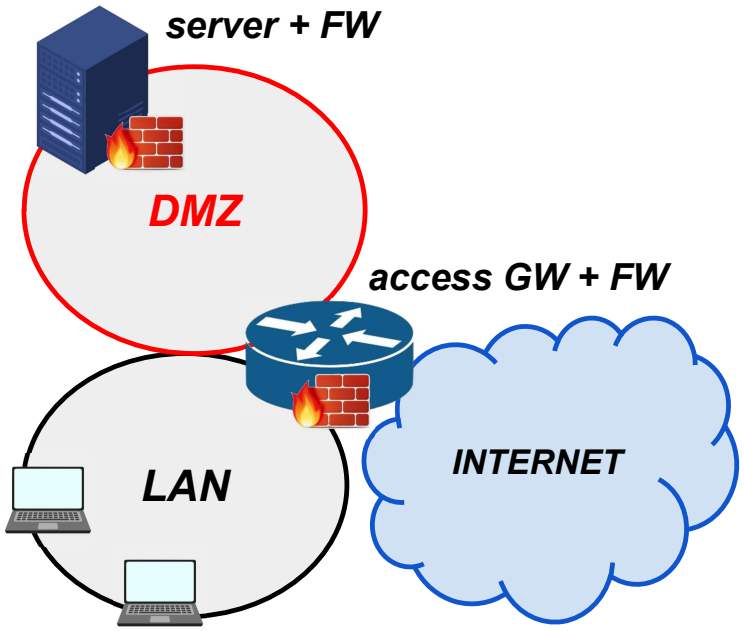
- a lot of further details should be discussed…
- ***… better learn by doing!***

Access GW security policy plan

- **Permit** traffic between DMZ and the INTERNET
- **Permit** traffic between LAN and DMZ only if initiated from LAN
- **Permit** SSH, HTTP, HTTPS and DNS traffic between LAN and INTERNET only if initiated from LAN
- **Deny** all traffic to GW except ssh and expect reply packets for locally initiated flows
- **Deny** all traffic initiated from DMZ to GW
- **Permit** traffic from GW to anywhere
- **Permit** all ICMP traffic (e.g. echo request/reply, port unreach, etc..)

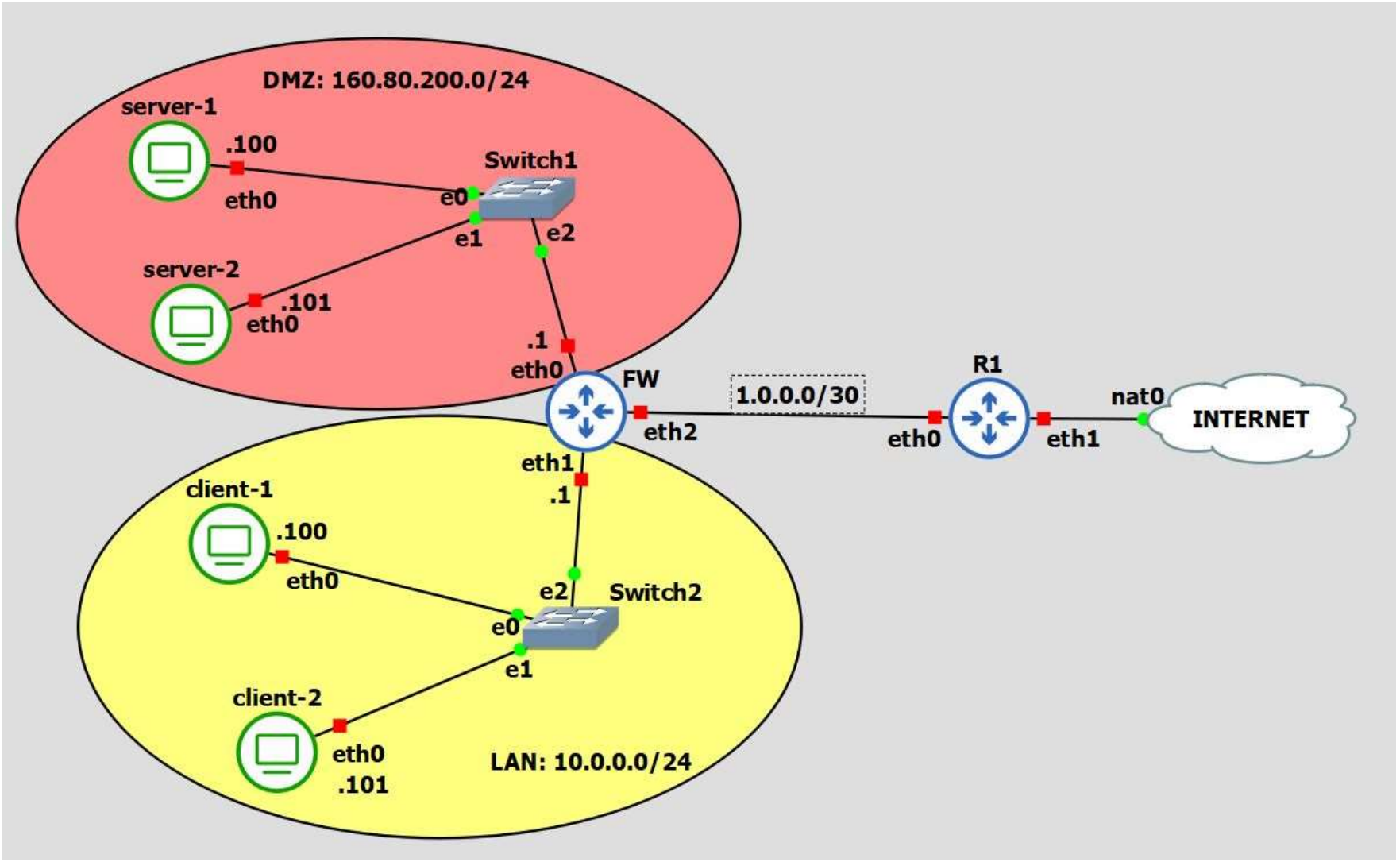
Server security policy plan

- **Permit** incoming traffic only for HTTP, HTTPS, SSH and all traffic related to connections locally initiated
- **Permit** all outgoing traffic

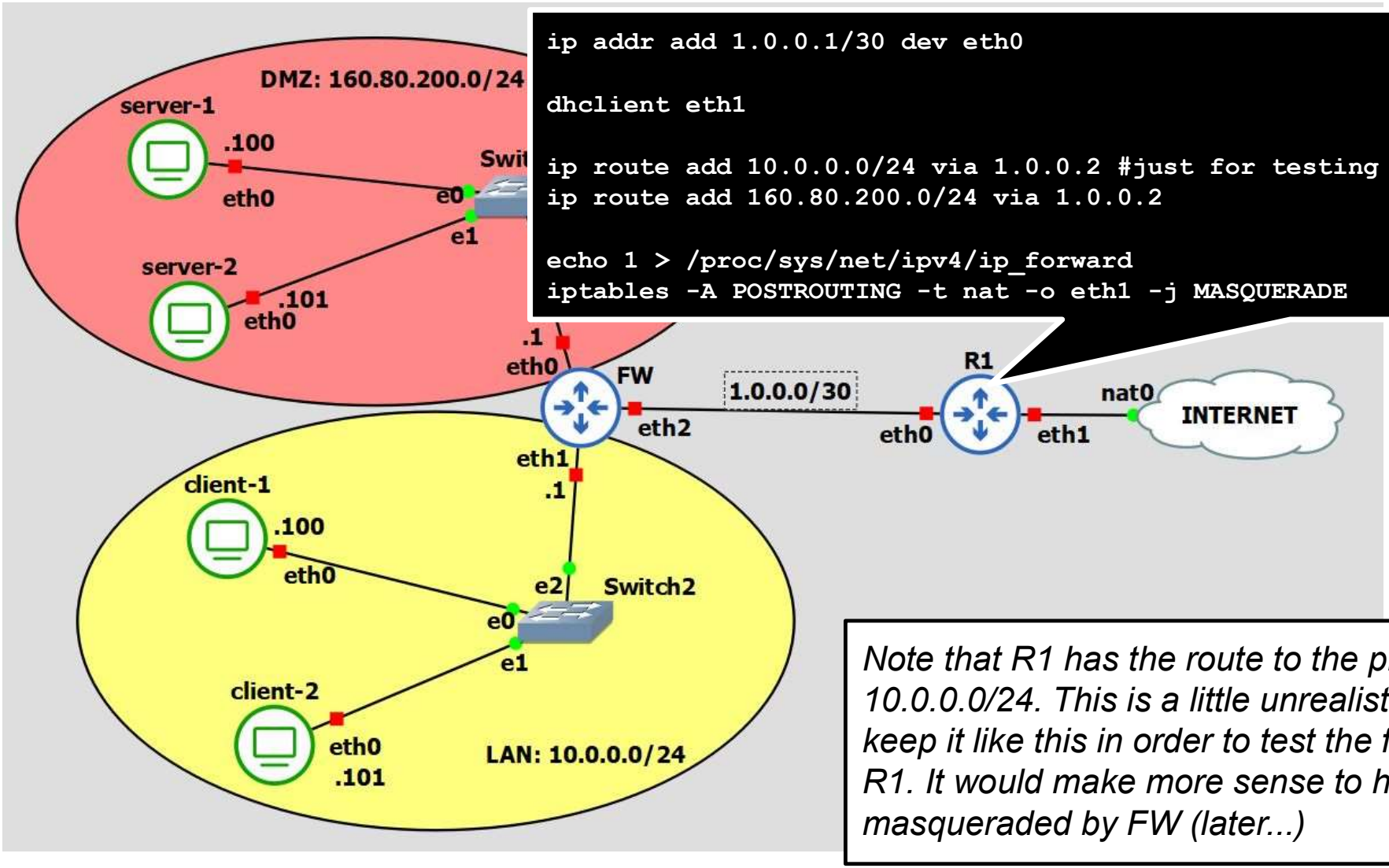


high-level topology

Detailed Topology

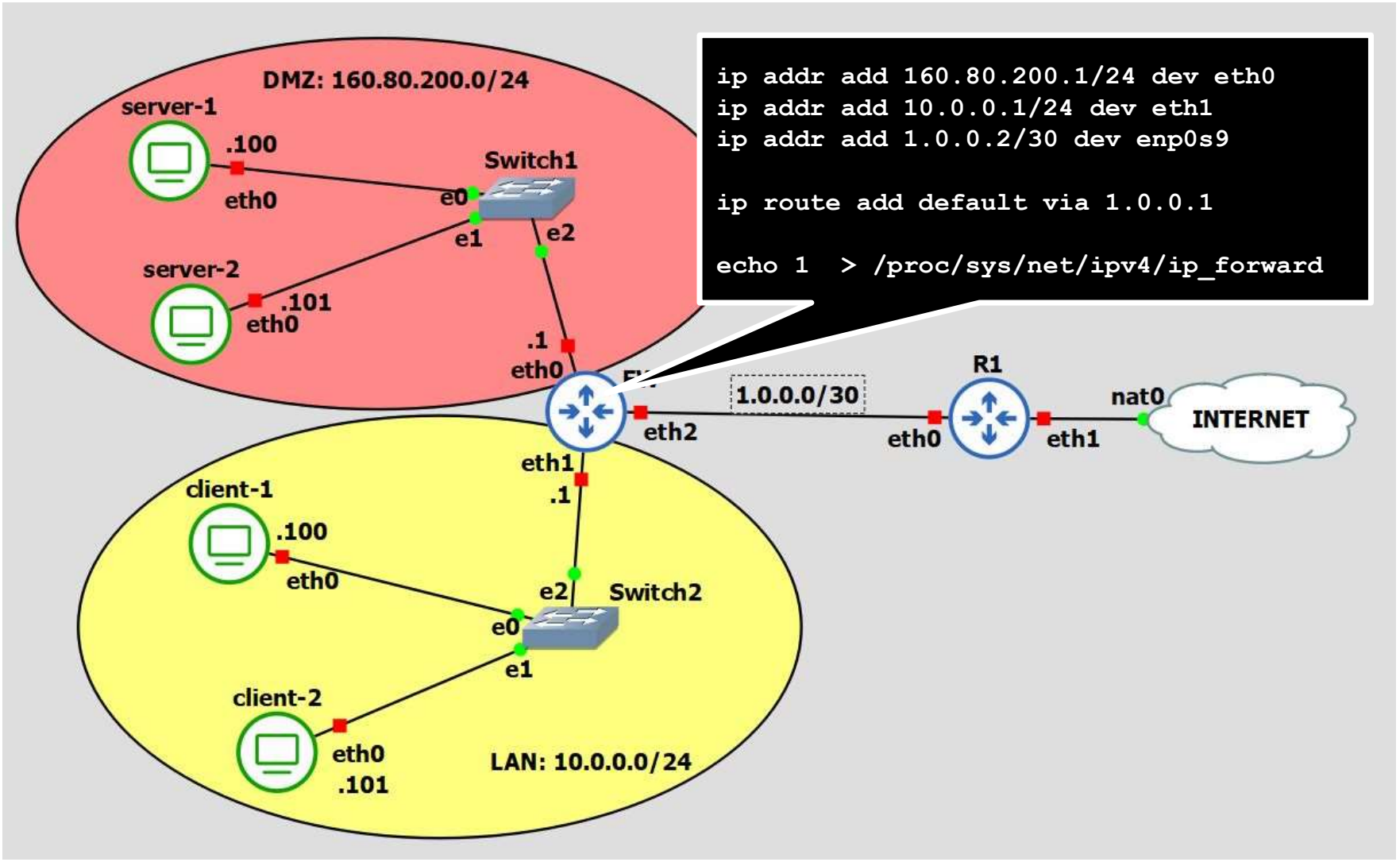


Detailed Topology

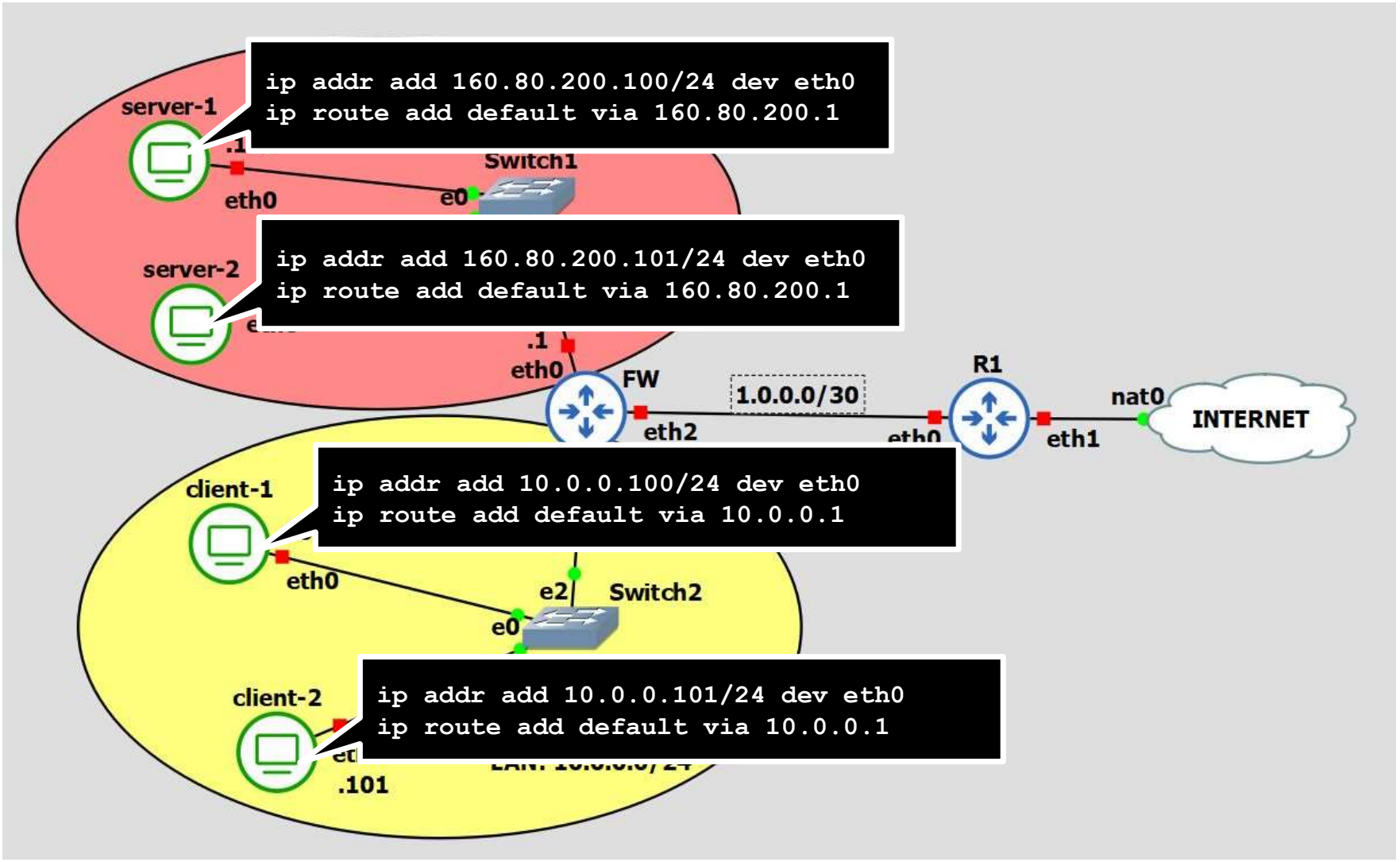


Note that R1 has the route to the private net 10.0.0.0/24. This is a little unrealistic but let's keep it like this in order to test the firewall from R1. It would make more sense to have this LAN masqueraded by FW (later...)

Detailed Topology



Detailed Topology



```
iptables -F  # flush already present entries
iptables -P FORWARD DROP
iptables -P INPUT DROP
iptables -P OUTPUT ACCEPT

iptables -A FORWARD -m state --state ESTABLISHED -j ACCEPT

iptables -A FORWARD -i $LAN -p tcp --dport 80 -j ACCEPT
iptables -A FORWARD -i $LAN -p tcp --dport 443 -j ACCEPT
iptables -A FORWARD -i $LAN -p tcp --dport 22 -j ACCEPT
iptables -A FORWARD -i $LAN -p udp --dport 53 -j ACCEPT

iptables -A FORWARD -i $LAN -o $DMZ -j ACCEPT
iptables -A FORWARD -i $WAN -o $DMZ -j ACCEPT
iptables -A FORWARD -i $DMZ -o $WAN -j ACCEPT


iptables -A INPUT -m state --state ESTABLISHED -j ACCEPT
iptables -A INPUT -i $LAN -p tcp --dport 22 -j ACCEPT
iptables -A INPUT -i $WAN -p tcp --dport 22 -j ACCEPT
iptables -A INPUT -p icmp -j ACCEPT

iptables -A FORWARD -p icmp -j ACCEPT
```

GW config

```
#interfaces

export LAN=eth1
export DMZ=eth0
export WAN=eth2
```

```
iptables -F
iptables -P INPUT DROP
iptables -P OUTPUT ACCEPT

iptables -A INPUT -m state --state ESTABLISHED -j ACCEPT
iptables -A INPUT -p tcp --dport 443 -j ACCEPT
iptables -A INPUT -p tcp --dport 80 -j ACCEPT
iptables -A INPUT -p tcp --dport 22 -j ACCEPT
```

server config

How to test the firewall

- the simplest way is tot use the network tool **`nc`** (**`netcat`**)
- **`nc`** allows to open client and server socket (TCP and UDP) with arbitrary port
- syntax for clients
  - **`nc [-uv] $addr $port`**
    - -v for verbose, -u for UDP (default TCP)
- syntax for servers
  - **`nc [-uv] -l -p $port`**
    - -v for verbose, -u for UDP (default TCP)
- For incoming connections we can use R1 as a client (that's why we have the route to 10.0.0.0/24)
- For outgoing connections we may also connect to real servers on the internet

Network Address Translation

- NETFILTER also support network address translations
- Dynamic source address translation:
  - `-j MASQUERADE`
- Static destination address translation (port forwarding)
  - `-j DNAT --to-destination $addr:[$port]`
- Local redirect (remember the DNS spoofing LAB??)
  - `-j REDIRECT`

Packet Classification

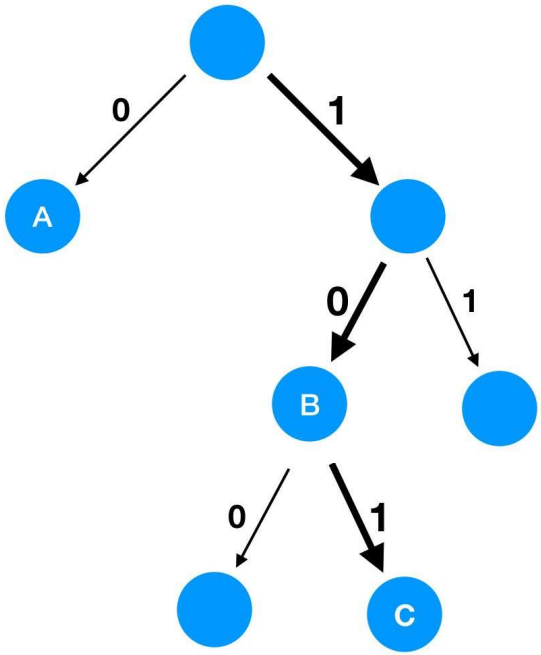
Source: George Varghese, "Network Algorithmics. An Interdisciplinary Approach to Designing Fast Networked Devices", Chapters 11 and 12

Why packet classification?

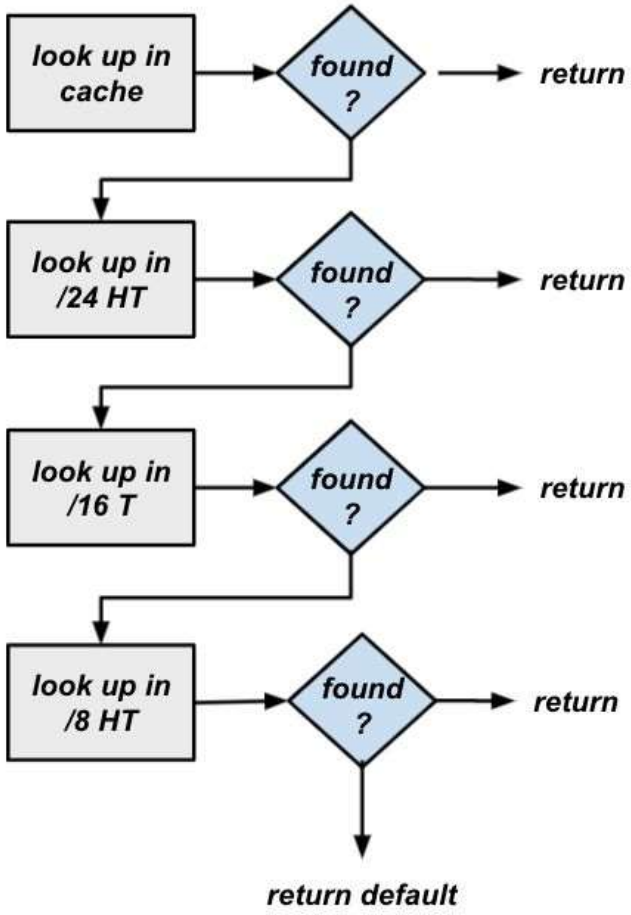
- ***Critical operation not only for firewalls***
  - Resource reservations, QoS routing, unicast routing, multicast routing, IDS, NAT, monitoring, traffic engineering, etc..
- Packet forwarding based on a longest-matching-prefix lookup of destination IP addresses is fairly well understood
  - LPM can be seen as a specific packet classification primitive
  - Efficiently solved with algorithmic solutions (basic variants of tries) and CAM pipelines
- Unfortunately, the Internet is becoming more complex because of its use for mission critical functions executed by organizations
  - Both QoS and security guarantees require a finer discrimination of packets, based on fields other than the destination
  - Examples of other fields a router may need to examine include source addresses (to forbid or provide different service to some source networks), port fields (to discriminate between traffic types, such as Napster and E-mail), and even TCP flags (to distinguish between externally and internally initiated connections)

Algorithmic LPM Approaches

Prefix	Router
0*	A
10*	B
101*	C



LPM Trie



LPM with multiple HTs (basic idea)

The Packet Classification Problem

- The matching rules for classifying a message are called matching **rules** and the packet-classification problem is to determine the ***lowest-cost matching rule for each incoming message***
- Information relevant to a lookup is contained in ***K*** distinct ***header fields***
- Header fields are denoted ***H[1], H[2], . . . , H[K]***
- The ***classifier*** consists of a finite set of rules, $R_1, R_2, \ldots, R_N$
- Each field in a rule is allowed three kinds of matches: ***(i) exact match***, ***(ii) prefix match***, and ***(iii) range match***
- Each rule R_i has an associated directive $disp_i$, which specifies how to process the packet matching this rule
  - in a (basic) firewall ***ACCEPT***, ***DROP***

The Packet Classification Problem

- A packet ***P*** is said to match a rule ***R*** if each field of P matches the corresponding field of R
- The match type is implicit in the specification of the field
  - if the destination field is specified as 1010∗, then it requires a prefix match
  - if the protocol field is UDP, then it requires an exact match
  - if the port field is a range, such as 1024–1100, then it requires a range match
- For instance, let ***R = (1010∗, ∗, TCP, 1024:1080, ∗)*** be a rule with ***disp=DROP***
- a packet with header ***(10101 . . . 111, 11110 . . . 000, TCP, 1050, 3)*** matches R and is therefore blocked
- The packet ***(10110 . . . 000, 11110 . . . 000, TCP, 80, 3)*** doesn't match R

The Packet Classification Problem

- Since a packet may match multiple rules in the database, each rule R in the database is associated with a nonnegative number, ***cost(R)***.
- Ambiguity is avoided by returning the ***least-cost rule matching the packet's header***.
- The cost function generalizes the implicit precedence rules that are used in practice to choose between multiple matching rules.
- In firewall applications or Cisco ACLs, for instance, rules are placed in the database in a specific linear order, where each rule takes precedence over a subsequent rule.
- Thus, the goal there is to find the first matching rule.
- Of course, the same effect can be achieved by making cost(R) equal to the position of rule R in the database.

Firewall example

Destination	Source	Destination Port	Source Port	Flags	Comments
M	*	25	*	*	Allow inbound mail
M	*	53	*	UDP	Allow DNS access
M	S	53	*	*	Secondary access
M	*	23	*	*	Incoming telnet
TI	*TO*	123	123	UDP	NTP time info
*	Net	*	*	*	Outgoing packets
Net	*	*	*	TCP ack	Return ACKs OK
*	*	*	*	*	Block everything!

Requirements and Metrics

- The requirements for rule matching are similar to those for IP lookups
- We wish to do packet classification at wire speed for minimum-size packets, and thus ***speed is the dominant metric***
- To allow the database to fit in high-speed memory it is useful to ***reduce the amount of memory needed***
- For most firewall databases, insertion speed is not an issue because rules are rarely changed
  - However, this is not always true for dynamic or stateful packet rules. E.g.: ***UDP "connection" tracking***
  - In some products the solution is to have the outgoing request packet dynamically trigger the insertion of a rule (which has addresses and ports that match the request) that allows the inbound response to be passed
  - This requires very ***fast update times*** (a third metric)

netfilter/iptables is not really scalable...

- **`iptables`** has been the primary tool to implement firewalls and packet filters on Linux for many years
- Over the years, **`iptables`** has been a blessing and a curse
  - A blessing for its flexibility and quick fixes
  - A curse during times debugging a 5K rules iptables setup in an environment where multiple system components are fighting over who gets to install what iptables rules
- Back then, network speeds were slow (in ‘95 the most common router would provide 56 kbps throughput). ***Today: 802.11ax (theoretically) up to 10 gbps***
- The standard practice of implementing access control lists (ACLs) as implemented by iptables was to use ***sequential list of rules***
  - i.e. every packet received or transmitted is matched against a list of rules, one by one
- ***However, linear processing has an obvious massive disadvantage, the cost of filtering a packet can increase linearly with the number of rules added***

nice reading: https://cilium.io/blog/2018/04/17/why-is-the-kernel-community-replacing-iptables

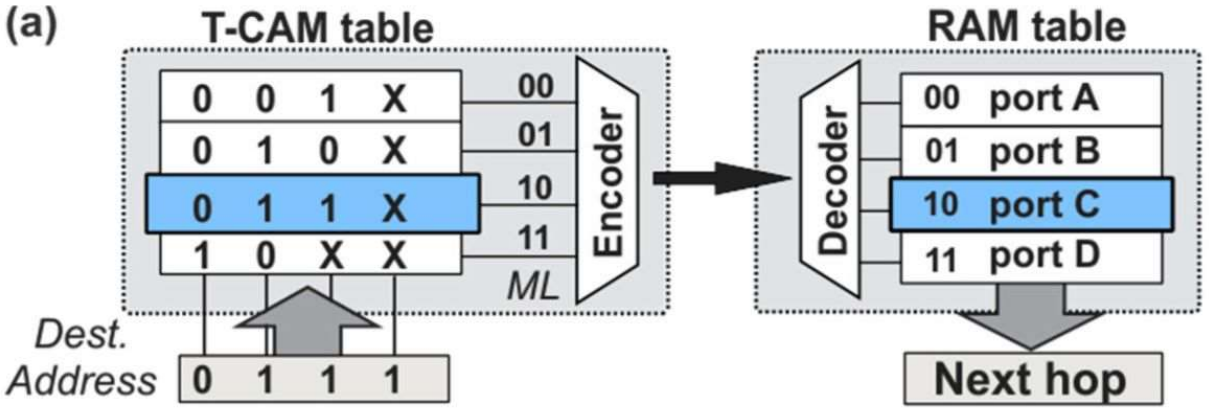
A simple solution: Caching

- Some implementations cache the result of the search keyed against the whole header
  - usually the socket 5-tuple is used as cache key
- The ***cache hit rate of caching full IP addresses*** in the backbones is typically ***at most 80–90%*** [Par96][NMH97]
  - Web accesses and other flows that send only a small number of packets
  - Caching full headers takes a lot more memory
  - In reality an eviction policy (e.g. LRU) is most likely enforced
  - ***So the cache hit rate is even worse***
- Even if the hit rate was 90%, the 10% in the slow match path still inexorably degrades the performance
  - suppose that a search of the cache costs 100 nsec (one memory access) and that a linear search of 10,000 rules costs 1,000,000 nsec = 1 msec (one memory access per rule). Then the average search time with a cache hit rate of 90% is still 0.1 msec, which is rather slow
- ***Even with caching, we need a fast matching algorithms***

A better solution: Content-Addressable Memories (CAM)

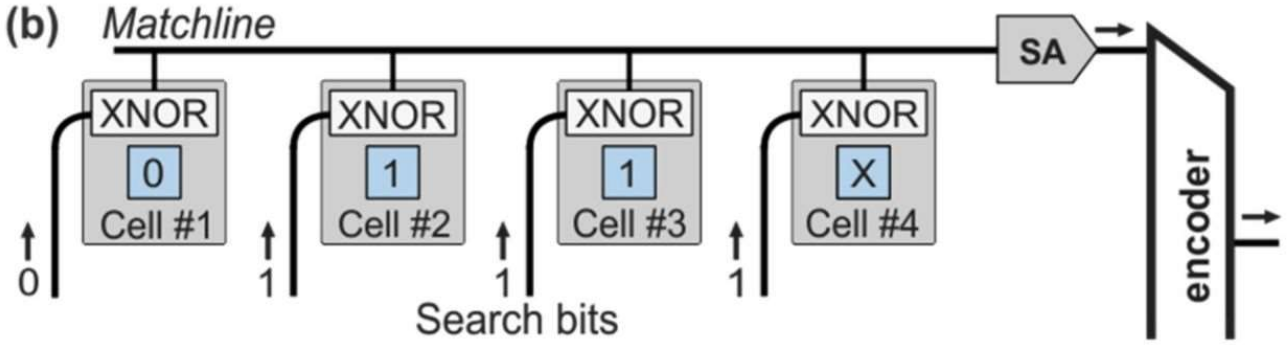
- A ***CAM*** is a ***content-addressable memory***, where the first cell that matches a data item will be returned using a parallel lookup in hardware
- A ***ternary CAM*** (TCAM) allows each bit of data to be either a 0, a 1, or a wildcard
  - Clearly, ternary CAMs can be used for rule matching as well as for prefix matching
  - However, the CAMs must provide wide lengths
    - ipv4 → 32 ip.src + 32 ip.dst + 16 sport + 16 dport + 8 ip.proto = **104 bits**
    - ipv6 → 128 ip.src + 128 ip.dst + 16 sport + 16 dport + 8 ip.proto = **296 bits**

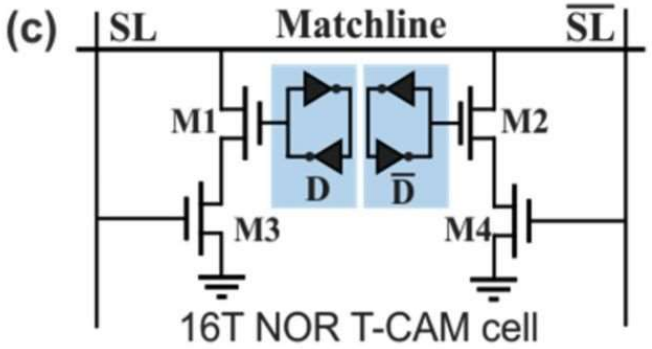
TCAMs for routing look-up



Router RIB Table

RIB entry	Destination address	Next hop port
1	001X	Port A
2	010X	Port B
3	011X	Port C
4	10XX	Port D





CAMs: not always a viable solution

- Several reasons to consider ***algorithmic alternatives to ternary CAMs***
  - Smaller density and larger power of CAMs versus SRAMs
  - Difficulty of integrating forwarding logic with the CAM
  - Rule multiplication caused by ranges
  - Several CAM vendors were also considering algorithmic solutions
  - No CAMs in SW packet processing
    - we still need fast SW implementations (especially with new NFV programming models)
- **Examples of algorithmic approaches:**
  - Tries: Multi-dimensional schemes
  - Binary search

An alternative approach: divide-and-conquer

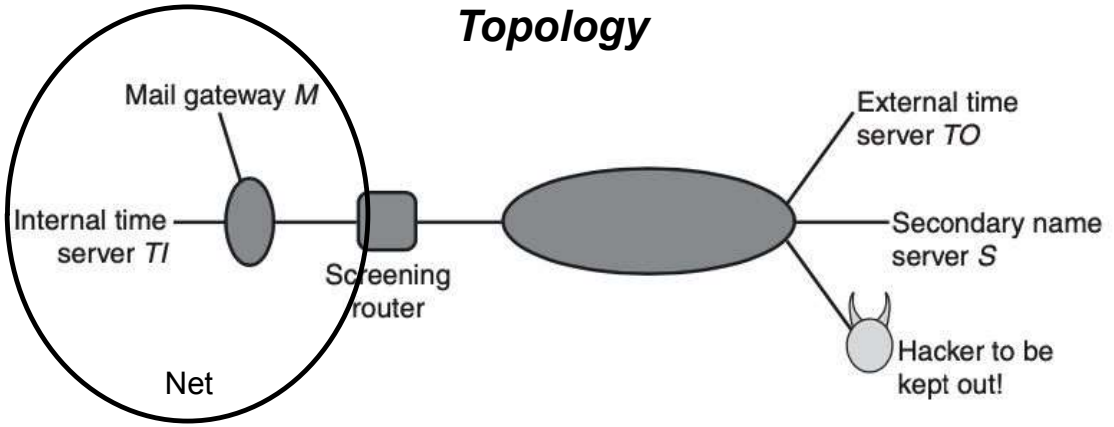
- ***Divide-and-conquer*** refers to dividing a problem into simpler pieces and then efficiently combining the answers from the pieces
- Start by slicing the rule database into columns, with the ***ith*** column storing all distinct prefixes (or ranges) in field ***i***
- Given a packet ***P***, determine the matching prefixes for each of its fields separately
- Finally, combine the results of the best-matching-prefix lookups on individual fields
- 3 methods that differ from each other in how the lookup of individual fields is combined into a single compound lookup
  - ***(i) Bit Vector Linear Search***; ***(ii) Cross-producting***; ***(iii) Decision Tree Approach***

Reference example

Firewall rule database

	Destination	Source	Destination Port	Source Port	Flags	Comments
1	*M*	*	25	*	*	Allow inbound mail
2	*M*	*	53	*	UDP	Allow DNS access
3	*M*	S	53	*	*	Secondary access
4	*M*	*	23	*	*	Incoming telnet
5	*TI*	*TO*	123	123	UDP	NTP time info
6	*	Net	*	*	*	Outgoing packets
7	Net	*	*	*	TCP ack	Return ACKs OK
8	*	*	*	*	*	Block everything!

Topology

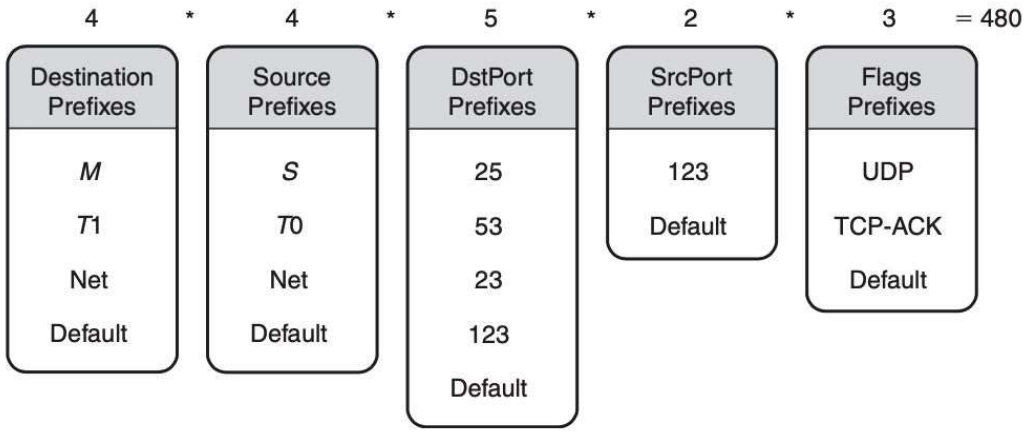


Reference example

Firewall rule database

	Destination	Source	Destination Port	Source Port	Flags	Comments
1	*M*	*	25	*	*	Allow inbound mail
2	*M*	*	53	*	UDP	Allow DNS access
3	*M*	S	53	*	*	Secondary access
4	*M*	*	23	*	*	Incoming telnet
5	*TI*	*TO*	123	123	UDP	NTP time info
6	*	Net	*	*	*	Outgoing packets
7	Net	*	*	*	TCP ack	Return ACKs OK
8	*	*	*	*	*	Block everything!

"Sliced" database



Bit Vector Linear Search

- Any match X in a given single field DB excludes the entries that don't match X in the other single field DBs
  - e.g.: a packet with destination address in T1 matches rules 5, 6, 7, 8 in DB 1
  - rules 1, 2, 3 and 4 can be "ignored" also in the other DBs
- The search algorithm needs to search only the intersection of the remaining sets obtained by each field lookup
- This would clearly be a good heuristic for optimizing the average case if the remaining sets are typically small
- one can guarantee performance even in the worst case
  - for each single DB we store a bitmask indicating which rules match the input packet
  - the intersection is obtained by simply ***ANDing*** the k bitmasks (k = number of match fields)
  - ***if the rules are ordered by descending priority, the best rule is the one associated to the first bit from the left set to 1***

Destination field DB extraction from full DB

	Destination	Source	Destination Port	Source Port	Flags	Comments
1	*M*	*	25	*	*	Allow inbound mail
2	*M*	*	53	*	UDP	Allow DNS access
3	*M*	S	53	*	*	Secondary access
4	*M*	*	23	*	*	Incoming telnet
5	*TI*	*TO*	123	123	UDP	NTP time info
6	*	Net	*	*	*	Outgoing packets
7	Net	*	*	*	TCP ack	Return ACKs OK
8	*	*	*	*	*	Block everything!

Field1 DB

field	b0	b1	b2	b3	b4	b5	b6	b7
M								
T1								
Net								
*								

4 different values, 8 bit bitmask

Destination field DB extraction from full DB

	Destination	Source	Destination Port	Source Port	Flags	Comments
1	M	*	25	*	*	Allow inbound mail
2	M	*	53	*	UDP	Allow DNS access
3	M	S	53	*	*	Secondary access
4	M	*	23	*	*	Incoming telnet
5	T1	TO	123	123	UDP	NTP time info
6	*	Net	*	*	*	Outgoing packets
7	Net	*	*	*	TCP ack	Return ACKs OK
8	*	*	*	*	*	Block everything!

Field1 DB

field	b0	b1	b2	b3	b4	b5	b6	b7
M								
T1								
Net								
*								

if destination matched M, rules 5 never matches

Destination field DB extraction from full DB

	Destination	Source	Destination Port	Source Port	Flags	Comments
1	M	*	25	*	*	Allow inbound mail
2	M	*	53	*	UDP	Allow DNS access
3	M	S	53	*	*	Secondary access
4	M	*	23	*	*	Incoming telnet
5	~~T1~~	~~TO~~	~~123~~	~~123~~	~~UDP~~	~~NTP time info~~
6	*	Net	*	*	*	Outgoing packets
7	Net	*	*	*	TCP ack	Return ACKs OK
8	*	*	*	*	*	Block everything!

Field1 DB

field	b0	b1	b2	b3	b4	b5	b6	b7
M	1	1	1	1	0	1	1	1
T1								
Net								
*								

resulting bitmask = 11110111

Destination field DB extraction from full DB

	Destination	Source	Destination Port	Source Port	Flags	Comments
1	M	*	25	*	*	Allow inbound mail
2	M	*	53	*	UDP	Allow DNS access
3	M	S	53	*	*	Secondary access
4	M	*	23	*	*	Incoming telnet
5	TI	TO	123	123	UDP	NTP time info
6	*	Net	*	*	*	Outgoing packets
7	Net	*	*	*	TCP ack	Return ACKs OK
8	*	*	*	*	*	Block everything!

Field1 DB

field	b0	b1	b2	b3	b4	b5	b6	b7
M	1	1	1	1	0	1	1	1
T1								
Net								
*								

if destination matches T1, rules 1,2, 3 and 4 never match

Destination field DB extraction from full DB

	Destination	Source	Destination Port	Source Port	Flags	Comments
1	*M*	*	25	*	*	Allow inbound mail
2	*M*	*	53	*	UDP	Allow DNS access
3	*M*	S	53	*	*	Secondary access
4	*M*	*	23	*	*	Incoming telnet
5	*TI*	*TO*	123	123	UDP	NTP time info
6	*	Net	*	*	*	Outgoing packets
7	Net	*	*	*	TCP ack	Return ACKs OK
8	*	*	*	*	*	Block everything!

Field1 DB

field	b0	b1	b2	b3	b4	b5	b6	b7
M	1	1	1	1	0	1	1	1
T1	0	0	0	0	1	1	1	1
Net								
*								

resulting bitmask = 00001111

Destination field DB extraction from full DB

	Destination	Source	Destination Port	Source Port	Flags	Comments
1	*M*	*	25	*	*	Allow inbound mail
2	*M*	*	53	*	UDP	Allow DNS access
3	*M*	S	53			Secondary access
4	*M*	*	23	*	*	Incoming telnet
5	*TI*	*TO*	123	123	UDP	NTP time info
6	*	Net	*	*	*	Outgoing packets
7	Net	*	*	*	TCP ack	Return ACKs OK
8	*	*	*	*	*	Block everything!

Field1 DB

field	b0	b1	b2	b3	b4	b5	b6	b7
M	1	1	1	1	0	1	1	1
T1	0	0	0	0	1	1	1	1
Net	0	0	0	0	0	1	1	1
*								

and so on...

Destination field DB extraction from full DB

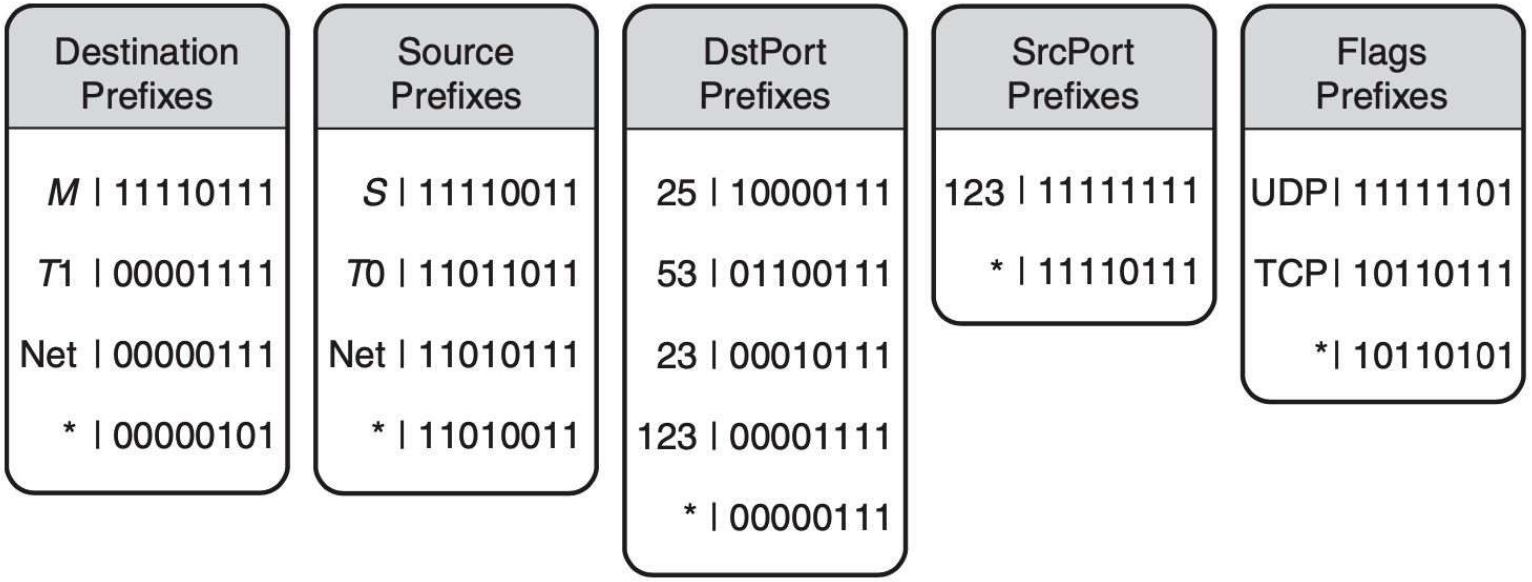
	Destination	Source	Destination Port	Source Port	Flags	Comments
1	*M*	*	25	*	*	Allow inbound mail
2	*M*	*	53	*	UDP	Allow DNS access
3	*M*	S	53	*	*	Secondary access
4	*M*	*	23	*	*	Incoming telnet
5	*TI*	*TO*	123	123	UDP	NTP time info
6	*	Net	*	*	*	Outgoing packets
7	Net	*	*	*	TCP ack	Return ACKs OK
8	*	*	*	*	*	Block everything!

Field1 DB

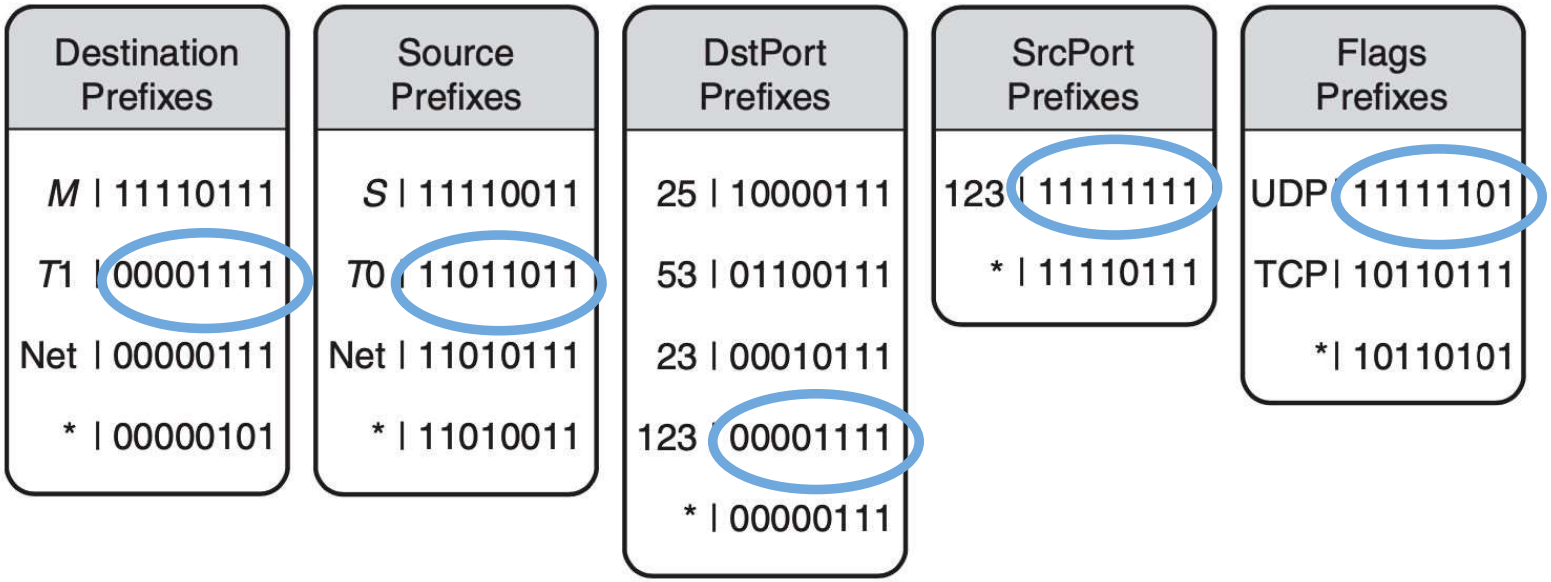
field	b0	b1	b2	b3	b4	b5	b6	b7
M	1	1	1	1	0	1	1	1
T1	0	0	0	0	1	1	1	1
Net	0	0	0	0	0	1	1	1
*	0	0	0	0	0	1	0	1

and so on...

Bit Vector Linear Search example ("disjoint" prefix ranges)



Bit Vector Linear Search example ("disjoint" prefix ranges)



look up keys: T1, T0, 123, 123, UDP

Bit Vector Linear Search example (“disjoint” prefix ranges)

```
0 0 0 0 1 1 1 1
      AND
1 1 0 1 1 0 1 1
      AND
0 0 0 0 1 1 1 1
      AND
1 1 1 1 1 1 1 1
      AND
1 1 1 1 1 1 0 1
```

Bit Vector Linear Search example (“disjoint” prefix ranges)

```
0 0 0 0 1 1 1 1
      AND
1 1 0 1 1 0 1 1
      AND
0 0 0 0 1 1 1 1
      AND
1 1 1 1 1 1 1 1
      AND
1 1 1 1 1 1 0 1
        =
0 0 0 0 1 0 0 1
```

Bit Vector Linear Search example ("disjoint" prefix ranges)

```
0 0 0 0 1 1 1 1
      AND
1 1 0 1 1 0 1 1
      AND
0 0 0 0 1 1 1 1
      AND
1 1 1 1 1 1 1 1
      AND
1 1 1 1 1 1 0 1
        =
0 0 0 0 1 0 0 1
```

matched rule: #5

Destination	Source	Destination Port	Source Port	Flags	Comments
M	*	25	*	*	Allow inbound mail
M	*	53	*	UDP	Allow DNS access
M	S	53	*	*	Secondary access
M	*	23	*	*	Incoming telnet
TI	*TO*	123	123	UDP	NTP time info
*	Net				Outgoing packets
Net	*	*	*	TCP ack	Return ACKs OK
*	*	*	*	*	Block everything!

But what if I found multiples matching rules?

- In case of a simple DROP/ACCEPT firewall, we only need to verify if the final bitvector result is != 0
- But in more complex cases, we need to understand what is the lowest cost rule (or the highest priority matching rule)
- Assuming a descending order, the highest priority rule is the one related to the leftmost bit set to 1 in the final bitvector result
- Searching for this has a linear complexity in a naive approach
- ***We need something more efficient!***
- ***Possible Solution:*** Leiserson et al. "Using de Bruijn sequences to index a 1 in a computer word"
  - https://www.academia.edu/download/41176510/0fcfd5107691fdedff000000.pdf20160115-19908-1ocbbzi.pdf
  - inspired by Miano, Sebastiano, et al. "Securing Linux with a faster and scalable iptables." ACM SIGCOMM Computer Communication Review 49.3 (2019). ***(Further details in the next laboratory)***

How do I search for the leftmost bit set to 1?

- Naive approach: linear search
  - linear complexity
- Better approach:
  - indexing all possible combination of the substrings for each index n
  - lookup time constant
- Shortcoming
  - exponential memory explosion

word	index
000	0
100	1
110	1
101	1
111	1
010	2
011	2
001	3

Optimization

Look for the position of the first bit to 1 in the bitvector using the de Bruijn sequences to find the index of the first bit set in a single word [Mia19]

Algorithm at a glance (input word x, len = n bits)

1. isolate the 1 (*)

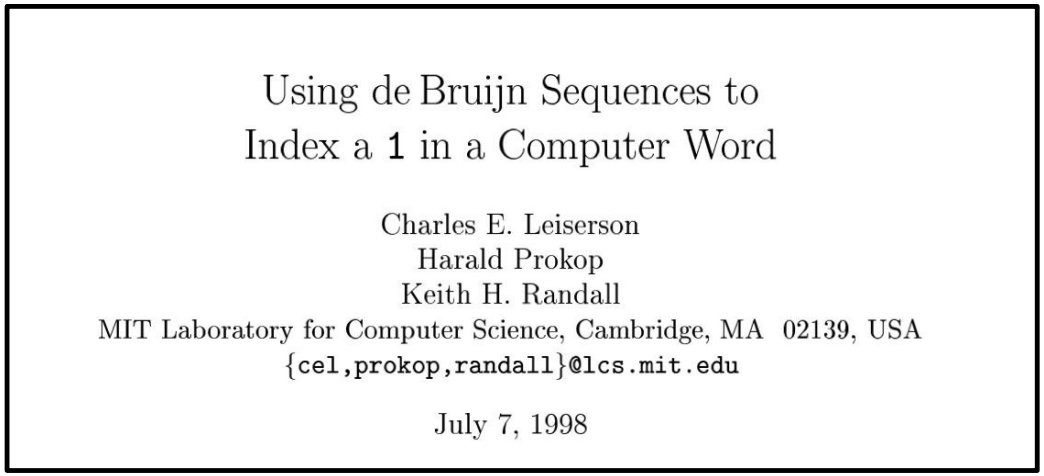
   ```
   y = x & (-x)
   ```

2. define a hash function for indexing the n different “one-1 words”

   ```
   h(z)=z*DeBruijn>>(n-log2(n))
   ```

3. find the index of the lowest order 1 in x

   ```
   index = h(x & (-x))
   ```



Using de Bruijn Sequences to
Index a 1 in a Computer Word

Charles E. Leiserson
Harald Prokop
Keith H. Randall
MIT Laboratory for Computer Science, Cambridge, MA 02139, USA
{cel,prokop,randall}@lcs.mit.edu

July 7, 1998

(*) with this operation, we isolate the rightmost bit (i.e. in reverse order w.r.t. previous examples)

32-bit C implementation of the DeBruijn strategy

```
#define debruijn32 0x077CB531UL
/* debruijn32 = 0000 0111 0111 1100 1011 0101 0011 0001 */

/* table to convert debruijn index to standard index */
int index32[32];

/* routine to initialize index32 */
void setup( void )
{
  int i;
  for(i=0; i<32; i++)
    index32[ (debruijn32 << i) >> 27 ] = i;
}

/* compute index of rightmost 1 */
int rightmost_index( unsigned long b )
{
  b &= -b;
  b *= debruijn32;
  b >>= 27;
  return index32[b];
}
```

Bit Vector Linear Search: performance

- N rules, the intersected bitmaps are N bits long.
  - Hence, computing the AND ***requires O(N) operations***
  - ***Why not do simple linear search instead?***
- Computing the AND of K bit vectors N-bit long is ***O(N*K)***
  - but general purpose 64 bit CPUs do 64 bit AND and memory look up in 1 clock cycle
  - and specialized HW can do even better...
- Thus, in reality, we have ***O(N*K/W)*** where ***W*** is the is the ***memory width***
- For example, using W = 1000 and k = 5 fields, the number of memory accesses for 5000 rules is $5000 * 6/1000 = 30$. Using 10-nsec SRAM, this allows a rule lookup in 300 nsec, which is sufficient to process minimum-size (40-byte) packets at wire speed on a gigabit link.
  - but specialized HW can do even better → ***k-fold parallelism***