Ch. 11 – Access Control Lists

Cabrillo College

CCNA 2 version 3.0

Rick Graziani

Cabrillo College

Note to instructors

- If you have downloaded this presentation from the Cisco Networking Academy Community FTP Center, this may not be my latest version of this PowerPoint.
- For the latest PowerPoints for all my CCNA, CCNP, and Wireless classes, please go to my web site:

  http://www.cabrillo.cc.ca.us/~rgraziani/

  - The username is *cisco* and the password is *perlman* for all of my materials.
- If you have any questions on any of my materials or the curriculum, please feel free to email me at graziani@cabrillo.edu (I really don't mind helping.) Also, if you run across any typos or errors in my presentations, please let me know.
- I will add "(Updated – *date*)" next to each presentation on my web site that has been updated since these have been uploaded to the FTP center.

Thanks! Rick

Part 1: ACL Fundamentals

Overview

- Network administrators must figure out how to deny unwanted access to the network while allowing internal users appropriate access to necessary services.
- Although security tools, such as passwords, callback equipment, and physical security devices are helpful, they often lack the flexibility of basic traffic filtering and the specific controls most administrators prefer.
- For example, a network administrator may want to allow users access to the Internet, but not permit external users telnet access into the LAN.
- Routers provide basic traffic filtering capabilities, such as blocking Internet traffic, with **access control lists (ACLs).**
- An ACL is a sequential list of permit or deny statements that apply to addresses or upper-layer protocols.
- This module will introduce ***standard*** and ***extended ACLs*** as a means to control network traffic, and how ACLs are used as part of a security solution.

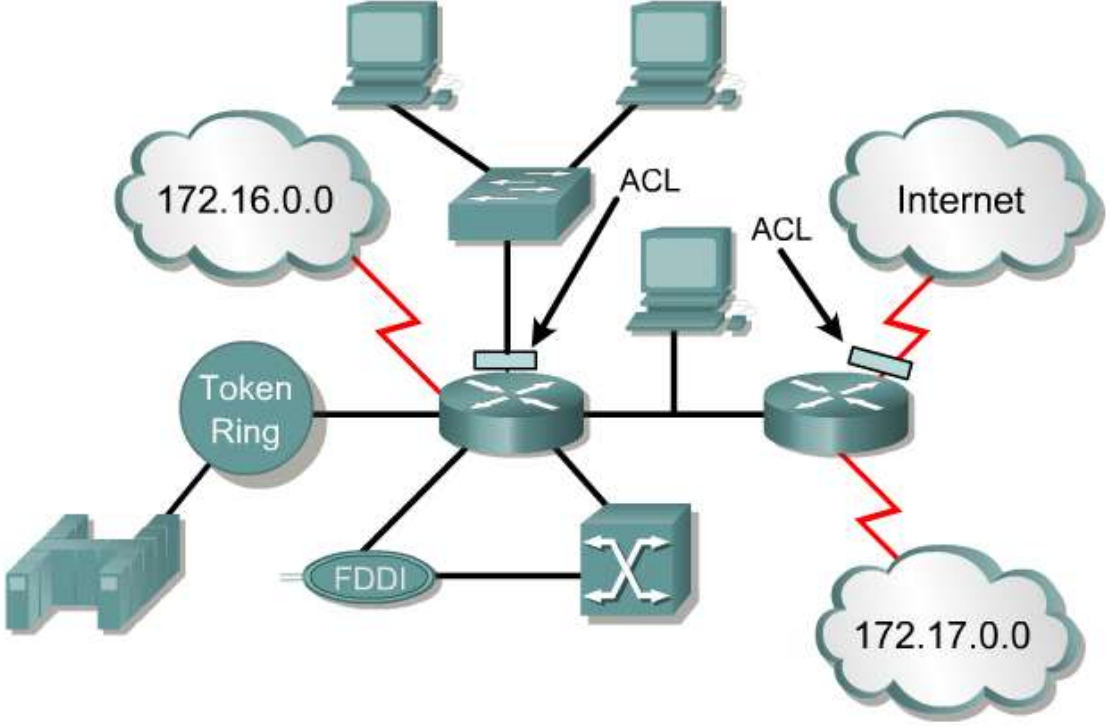
Overview

- In addition, this chapter includes:
  - Tips, considerations, recommendations, and general guidelines on how to use ACLs,
  - Commands and configurations needed to create ACLs.
  - Examples of standard and extended ACLs
  - How to apply ACLs to router interfaces.

Doyle:

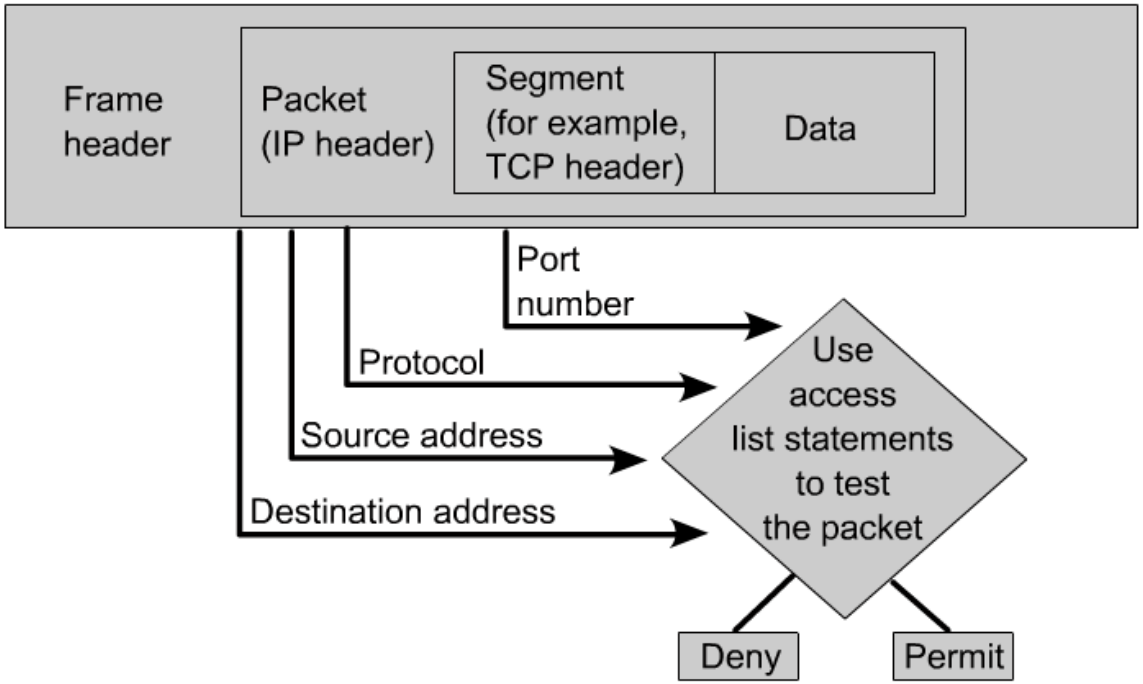
- Access Lists have become powerful tools for controlling the behavior of packets and frames.
- Their uses fall into three categories.
  1. Security Filters protect the integrity of the router and the networks to which it is passing traffic. (CCNA)
  2. Traffic Filters prevent unnecessary packets from passing onto limited-bandwidth links. (CCNP)
  3. Other Filters such as dialer lists, route filters, route maps, and queuing lists, must be able to identify certain packets to function properly. (CCNP)

What are ACLs?



- **Note**: Much of the beginning of this module are concepts. These concepts will become much clearer once we begin configuring ACLs.
- An access list is a sequential series of commands or filters.
- These lists tell the router what types of packets to:
  - accept or
  - deny
- Acceptance and denial can be based on specified conditions.
- ACLs applied on the router's interfaces.

What are ACLs?



- The router examines each packet to determine whether to forward or drop it, based on the conditions specified in the ACL.
- *Some* ACL decision points are:
  - IP source address
  - IP destination addresses
  - UDP or TCP protocols
  - upper-layer (TCP/UDP) port numbers

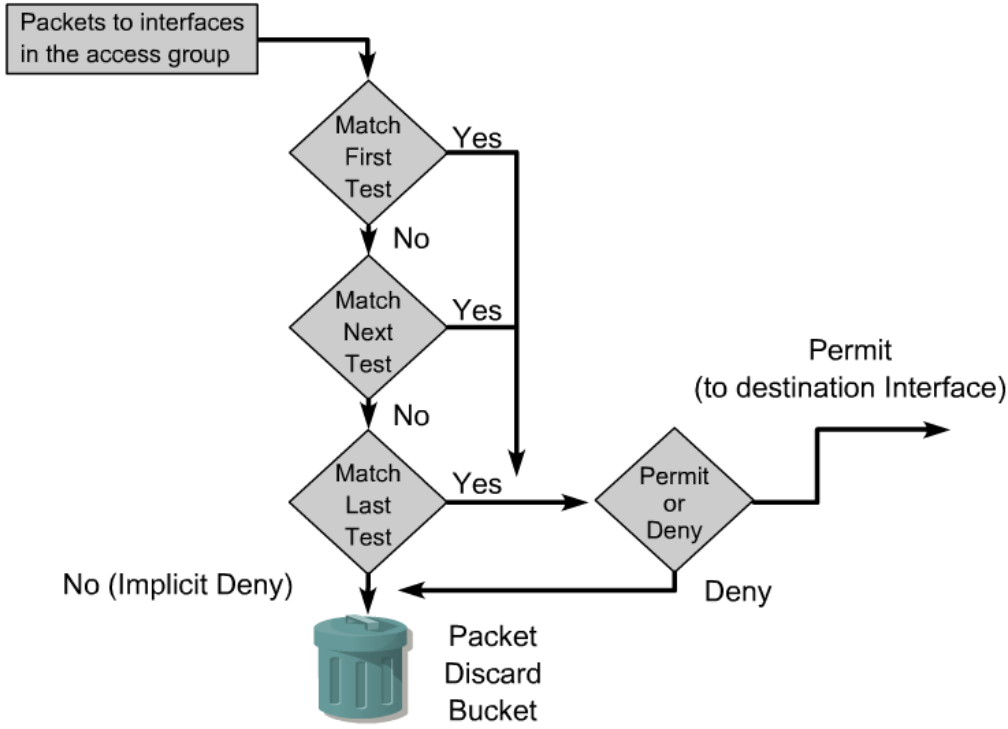
What are ACLs?

One list, per port, per direction, per protocol

With two interfaces and three protocols running, this router could have a total of 12 separate ACLs applied.

- ACLs must be defined on a:
  - per-protocol (IP, IPX, AppleTalk)
  - per direction (in or out)
  - per port (interface) basis.
- ACLs control traffic in one direction at a time on an interface.
- A separate ACL would need to be created for each direction, one for inbound and one for outbound traffic.
- Finally every interface can have multiple protocols and directions defined.

How ACLs work



- An ACL is a group of statements that define whether packets are accepted or rejected coming into an interface or leaving an interface.
- ACL statements operate in sequential, logical order.
- If a condition match is true, the packet is permitted or denied and the rest of the ACL statements are not checked.
- If all the ACL statements are unmatched, an implicit ***"deny any"*** statement is placed at the end of the list by ***default***. (not visible)
- When first learning how to create ACLs, it is a good idea to add the **`implicit deny`** at the end of ACLs to reinforce the dynamic presence of the command line..

How ACLs work

- **Access list statements** operate in sequential, logical order.
- They evaluate packets from the **top down**.
- Once there is an access list statement **match**, the packet skips the rest of the statements.
  - If a condition **match is true**, the packet is permitted or denied.
- There can be **only one access list** per protocol per interface.
- There is an implicit “deny any” at the end of every access list.
- **ACLs do not block packets that originate within the router. (ie. pings, telnets, etc.)**

Two types of ACLs

- Standard IP ACLs
  - Can only filter on source IP addresses

- Extended IP ACLs
  - Can filter on:
    - Source IP address
    - Destination IP address
    - Protocol (TCP, UDP)
    - Port Numbers (Telnet – 23, http – 80, etc.)
    - *and other parameters*

Creating Standard ACLs – 2 Steps

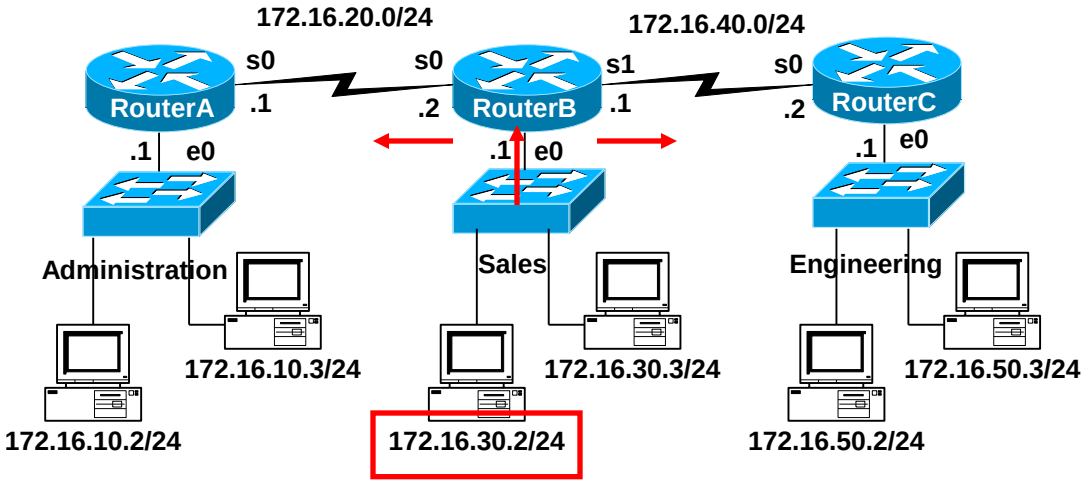
Step 1	Define the ACL by using the following command: `Router(config)#access-list access-list-number {permit | deny} {test-conditions}` A global statement identifies the ACL. Specifically, the 1-99 range is reserved for standard IP. This number refers to the type of ACL. In Cisco IOS Release 11.2 or newer, ACLs can also use an ACL name, such as education_group, rather than a number. The **permit** or **deny** term in the global ACL statement indicates how packets that meet the test conditions are handled by Cisco IOS software. **permit** usually means the packet will be allowed to use one or more interfaces that you will specify later.The final term or terms specifies the test conditions used by the ACL statement.
Step 2	Next, you need to apply ACLs to an interface by using the **access-group** command, as in this example: `Router(config-if)#{protocol} access-group access-list-number` All the ACL statements identified by *access-list-number* are associated with one or more interfaces. Any packets that pass the ACL test conditions can be permitted to use any interface in the access group of interfaces.

Creating ACLs – 2 Steps

Step 1	Define the ACL by using the following command: `Router(config)#access-list access-list-number {permit | deny} {test-conditions}` A global statement identifies the ACL. Specifically, the 1-99 range is reserved for standard IP. This number refers to the type of ACL. In Cisco IOS Release 11.2 or newer, ACLs can also use an ACL name, such as education_group, rather than a number. The **permit** or **deny** term in the global ACL statement indicates how packets that meet the test conditions are handled by Cisco IOS software. **permit** usually means the packet will be allowed to use one or more interfaces that you will specify later.The final term or terms specifies the test conditions used by the ACL statement.

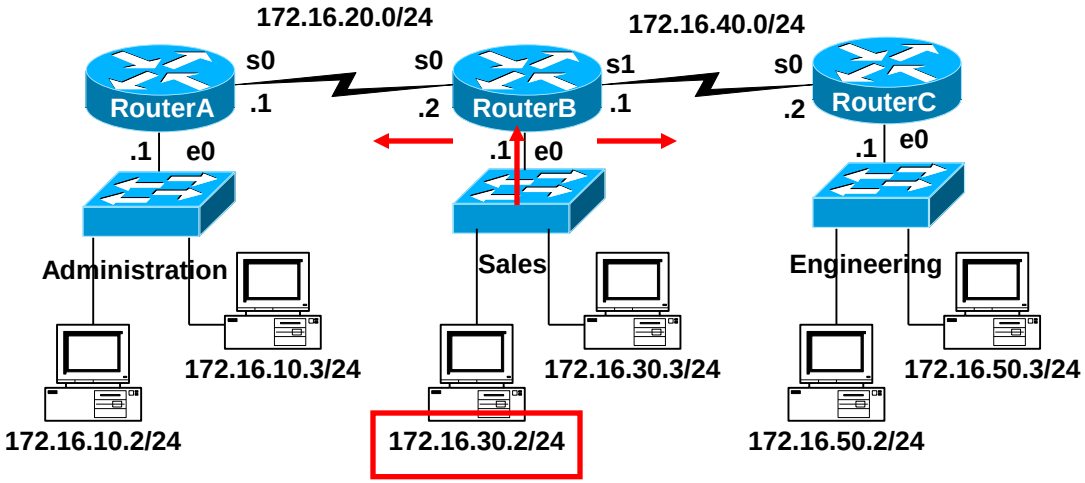
Protocol	Range
IP (Standard IP)	1-99
Extended IP	100-199
AppleTalk	600-699
IPX	800-899
Extended IPX	900-999
IPX Service Advertising Protocol	1000-1099

Learn by example!



- Task:
  - Permit only the host 172.16.30.2 from exiting the Sales network.
  - Deny all other hosts on the Sales network from leaving the 172.16.30.0/24 network.

Learn by example!



Step 1 – ACL statements Implicit deny any, which is automatically added.

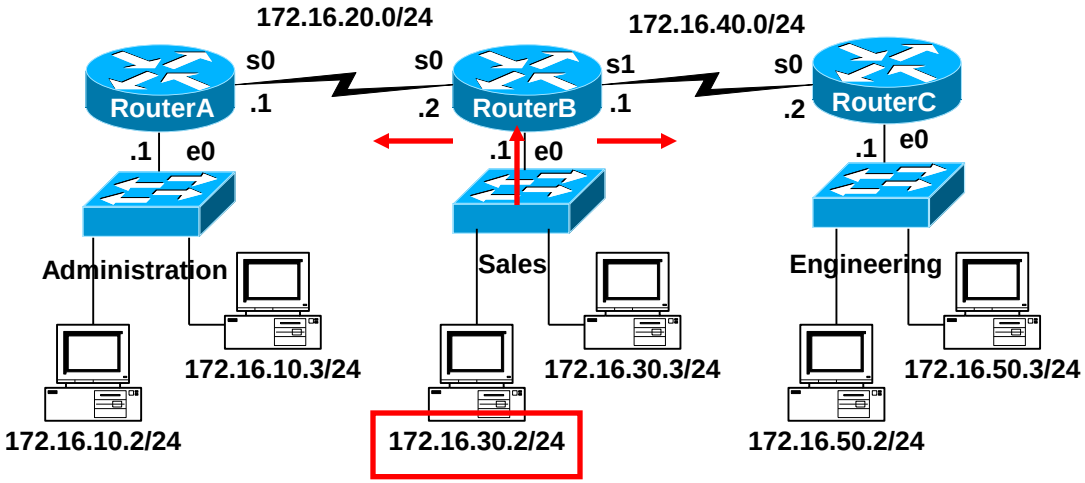
Test Condition

```
RouterB(config)#access-list 10 permit 172.16.30.2
Implicit "deny any" -do not need to add this, discussed later
RouterB(config)#access-list 10 deny 0.0.0.0 255.255.255.255
```

```
Router(config)#access-list access-list-number
   {permit | deny} {test-conditions}
```

Protocol		Range
IP	(Standard IP)	1-99

From Cisco Web Site



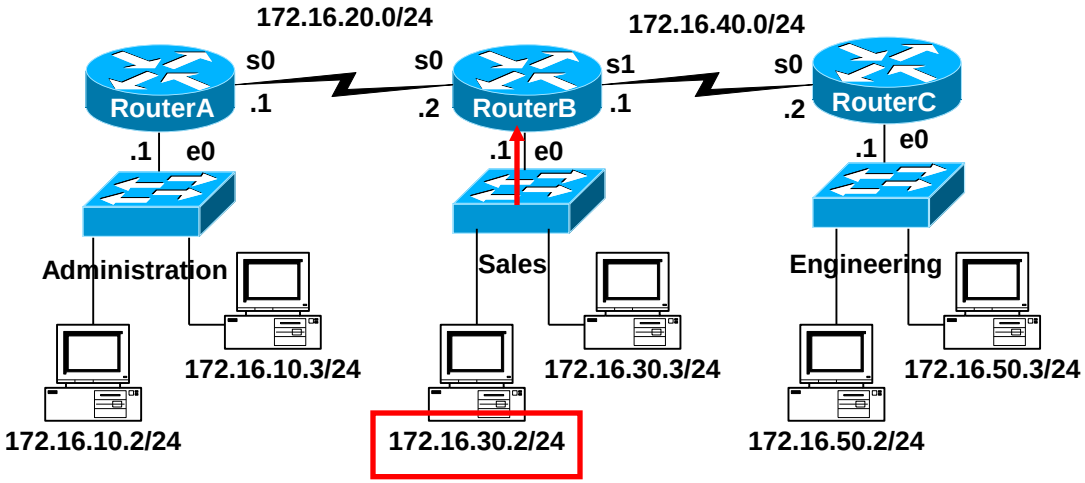
Applying ACLs

- You can define ACLs without applying them.
- However, the ACLs will have no effect until they are applied to the router's interface.
- It is a good practice to apply the Standard ACLs on the interface closest to the destination of the traffic and Extended ACLs on the interface closest to the source. (coming later)

Defining In, Out, Source, and Destination

- **Out** - Traffic that has already been routed by the router and is leaving the interface
- **In** - Traffic that is arriving on the interface and which will be routed router.

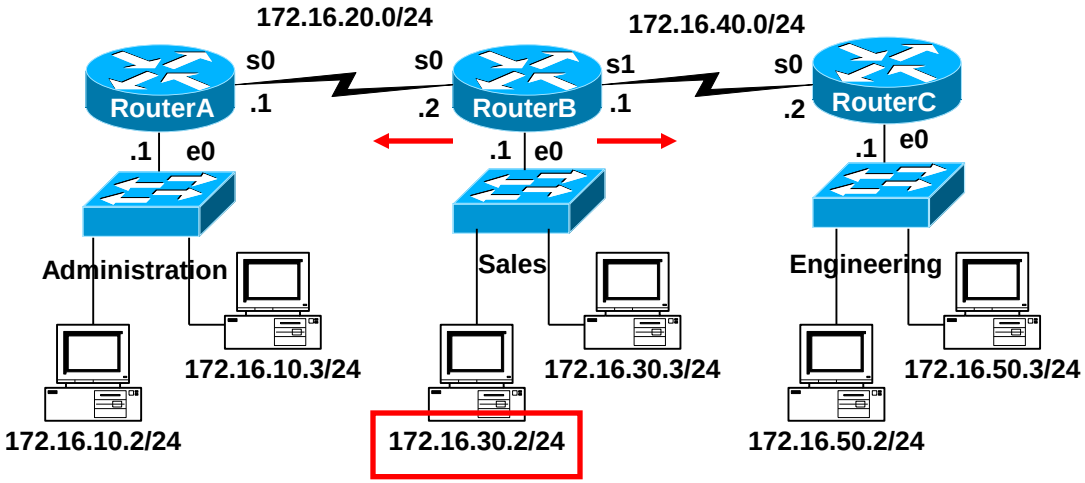
Learn by example!



Step 2 – Apply to an interface(s)

```
RouterB(config)#access-list 10 permit 172.16.30.2
Implicit "deny any" -do not need to add this, discussed later
RouterB(config)#access-list 10 deny 0.0.0.0 255.255.255.255

RouterB(config)# interface e 0
RouterB(config-if)# ip access-group 10 in
Router(config-if)#{protocol} access-group access-list-
number
```

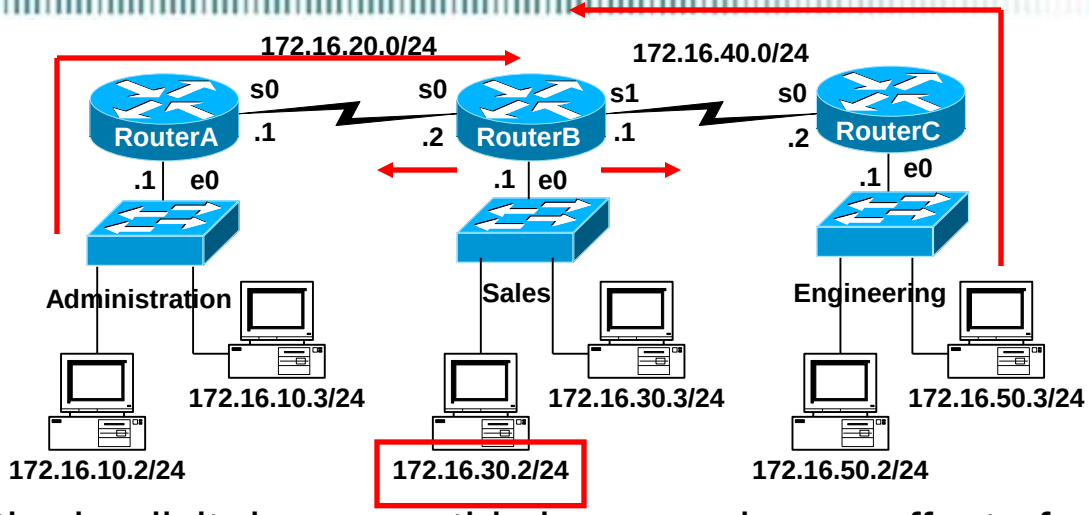
Learn by example!



Step 2 – Or the outgoing interfaces… Which is preferable and why?

```
RouterB(config)#access-list 10 permit 172.16.30.2
Implicit "deny any" -do not need to add this, discussed later
RouterB(config)#access-list 10 deny 0.0.0.0 255.255.255.255

RouterB(config)# interface s 0
RouterB(config-if)# ip access-group 10 out
RouterB(config)# interface s 1
RouterB(config-if)# ip access-group 10 out
```

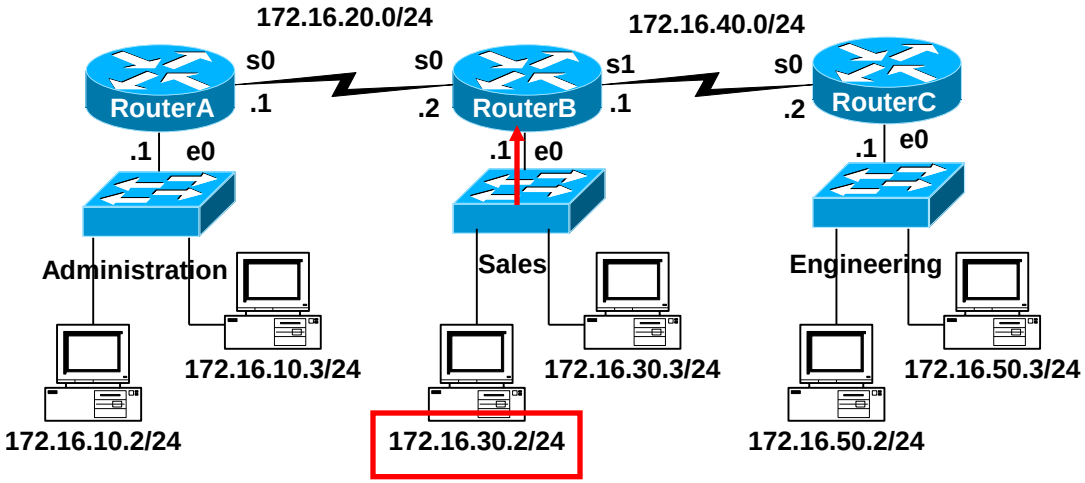
Learn by example!



Because of the implicit deny any, this has an adverse affect of also denying packets from Administration from reaching Engineering, and denying packets from Engineering from reaching Administration.

```
RouterB(config)#access-list 10 permit 172.16.30.2
Implicit "deny any" -do not need to add this, discussed later
RouterB(config)#access-list 10 deny 0.0.0.0 255.255.255.255

RouterB(config)# interface s 0
RouterB(config-if)# ip access-group 10 out
RouterB(config)# interface s 1
RouterB(config-if)# ip access-group 10 out
```

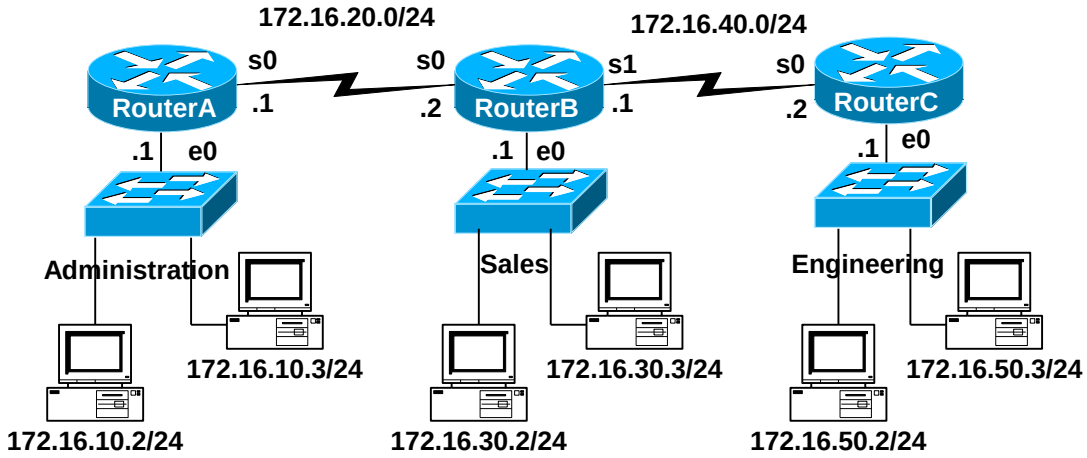
Learn by example!



Preferred, this access list will work to all existing and new interfaces on RouterB.

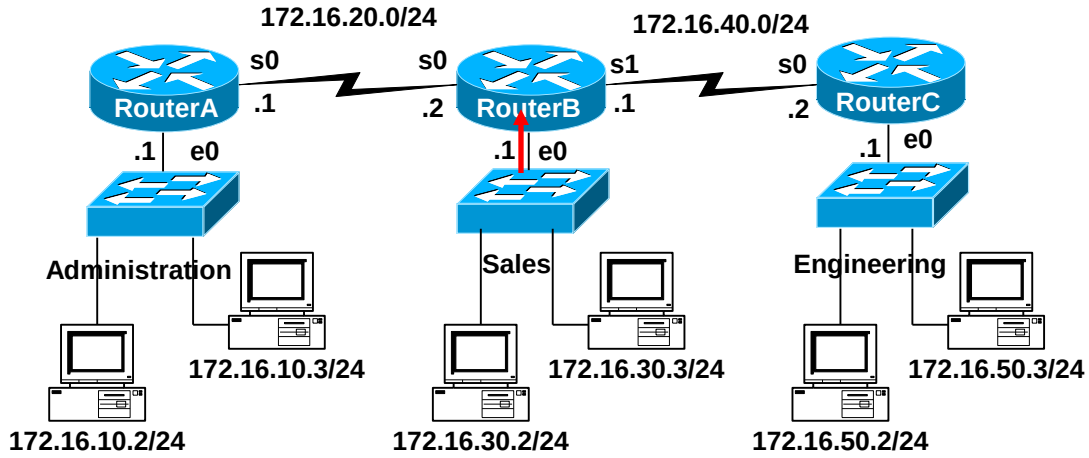
```
RouterB(config)#access-list 10 permit 172.16.30.2
Implicit "deny any" -do not need to add this, discussed later
RouterB(config)#access-list 10 deny 0.0.0.0 255.255.255.255

RouterB(config)# interface e 0
RouterB(config-if)# ip access-group 10 in
```

Example 2



- Task:
  - Permit only the hosts 172.16.30.2, 172.16.30.3, 172.16.30.4, 172.16.30.5 from exiting the Sales network.
  - Deny all other hosts on the Sales network from leaving the 172.16.30.0/24 network.

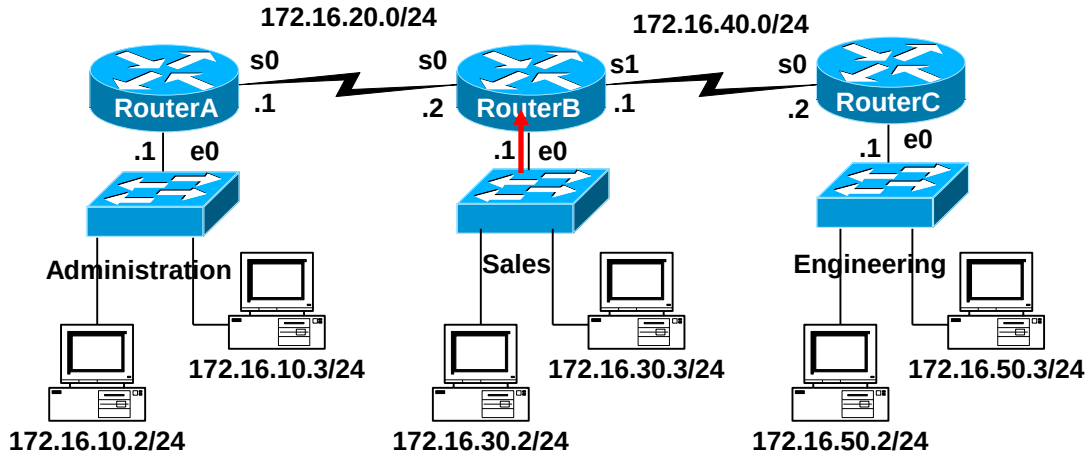
Example 2



Once a condition is met, all other statements are ignored, so the implicit *deny any* only applies to not-matched packets.

```
RouterB(config)#access-list 10 permit 172.16.30.2
RouterB(config)#access-list 10 permit 172.16.30.3
RouterB(config)#access-list 10 permit 172.16.30.4
RouterB(config)#access-list 10 permit 172.16.30.5
Implicit "deny any" -do not need to add this, discussed later
RouterB(config)#access-list 10 deny 0.0.0.0 255.255.255.255

RouterB(config)# interface e 0
RouterB(config-if)# ip access-group 10 in
```

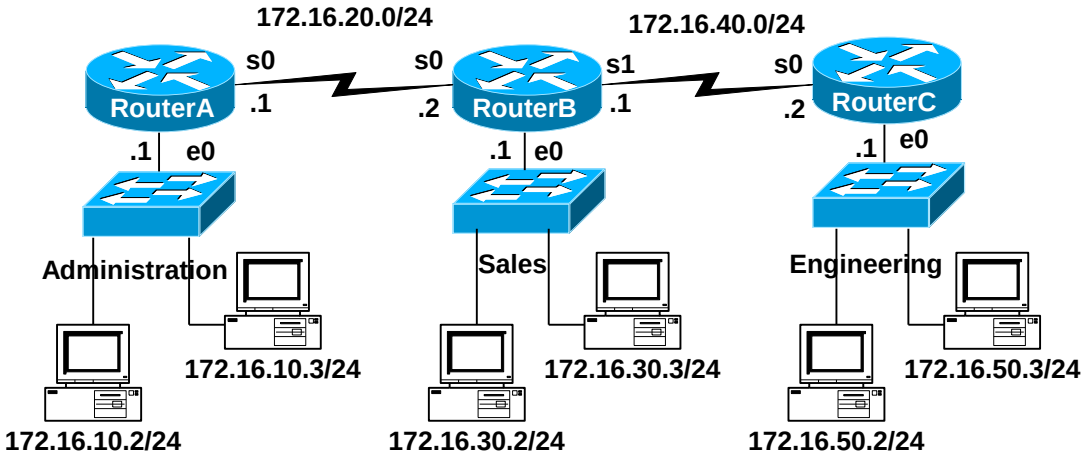
Example 2



To remove an Access List, use the no access-list command. Removing the access-group only from from the interface leaves the access list, but they are not currently being applied. Usually, best to remove it from both.

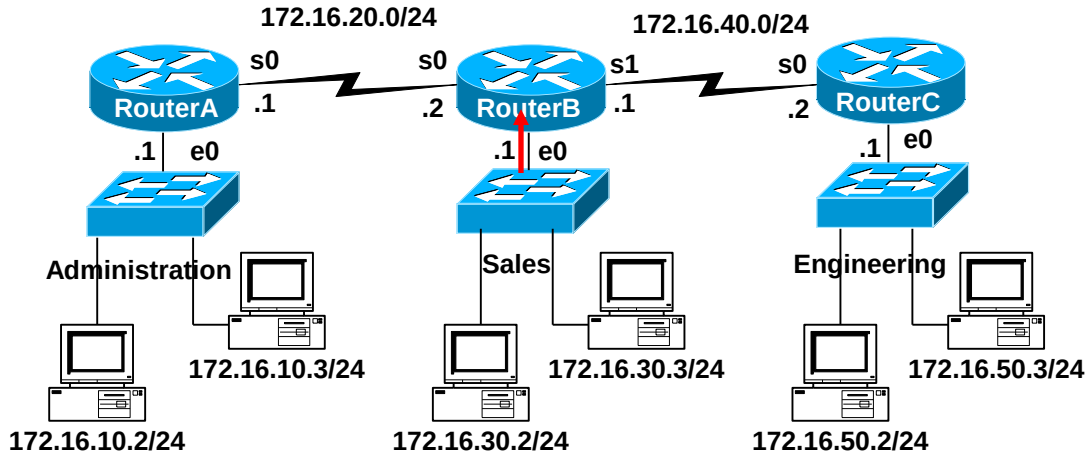
```
RouterB(config)#no access-list 10

RouterB(config)# interface e 0
RouterB(config-if)# no ip access-group 10 in
```

Example 3



- Task:
  - Deny only the host 172.16.30.2 from exiting the Sales network.
  - Permit all other hosts on the Sales network to leave the 172.16.30.0/24 network.
- Keyword “any” can be used to represent all IP Addresses.

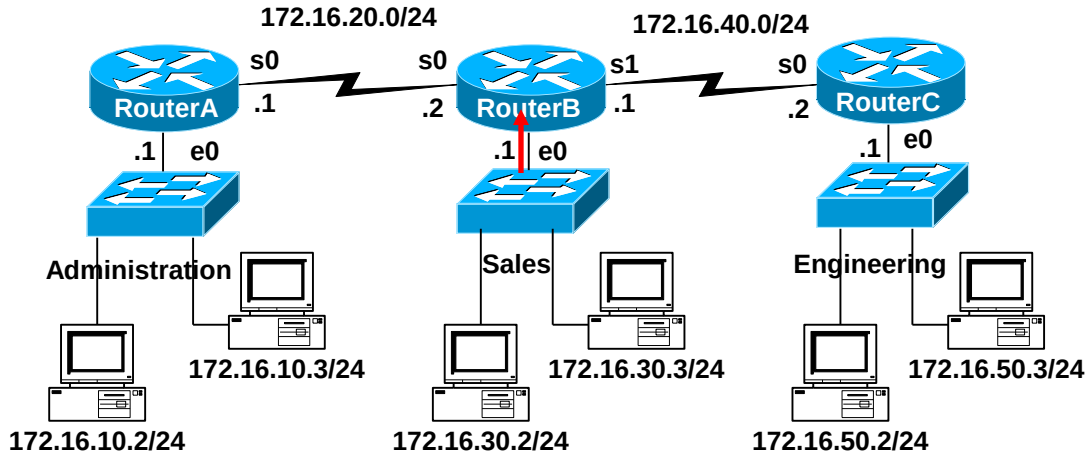
Example 3



Order matters! What if these two statements were reversed? Does the *implicit deny any* ever get a match? No, the permit any will cover all other packets.

```
RouterB(config)#access-list 10 deny 172.16.30.2
RouterB(config)#access-list 10 permit any
Implicit "deny any" -do not need to add this, discussed later
RouterB(config)#access-list 10 deny 0.0.0.0 255.255.255.255

RouterB(config)# interface e 0
RouterB(config-if)# ip access-group 10 in
```

Example 3



Order matters! In this case all packets would be permitted, because all packets would match the first access list statement. Once a condition is met, all other statements are ignored. The second access list statement and the implicit deny any would never be used. This would not do what we want.

```
RouterB(config)#access-list 10 permit any
RouterB(config)#access-list 10 deny 172.16.30.2
Implicit "deny any" -do not need to add this, discussed later
RouterB(config)#access-list 10 deny 0.0.0.0 255.255.255.255

RouterB(config)# interface e 0
RouterB(config-if)# ip access-group 10 in
```

Note on inbound access lists

- When an access lists applied to an **inbound** interface, the packets are checked against the access list before any routing table lookup process occurs.
- We will see how **outbound access list** work in a moment, but they are applied after the forwarding decision is made, after the routing table lookup process takes place and an exit interface is determined.
- Once a packet is denied by an ACL, the router sends an **ICMP** "Destination Unreachable" message, with the code value set to "Administratively Prohibited" to the source of the packet.

```
RouterB(config)#access-list 10 deny 172.16.30.2
RouterB(config)#access-list 10 permit any
Implicit "deny any" (do not need to add this, discussed later):
RouterB(config)#access-list 10 deny 0.0.0.0 255.255.255.255

RouterB(config)# interface e 0
RouterB(config-if)# ip access-group 10 in
```

Notes from www.cisco.com

- Traffic coming into the router is compared to ACL entries based on the order that the entries occur in the router.
- New statements are added to the end of the list.
- The router keeps looking until it has a match.
- If no matches are found when the router reaches the end of the list, the traffic is denied.
- For this reason, you should have the frequently hit entries at the top of the list.
- There is an "implied deny" for traffic that is not permitted.
- A single-entry ACL with only one "deny" entry has the effect of denying all traffic.
- You must have at least one "permit" statement in an ACL or all traffic will be blocked.

```
access-list 10 permit 10.1.1.1 0.0.0.255
access-list 10 deny ip any   (implicit)
```

Time for Wildcard Masks!

Access-list 1 permit 172.16.0.0 0.0.255.255

A **wildcard mask** address:

- Tells how much of the packet's source IP address (or destination IP address) needs to match for this condition to be true.

Time for Wildcard Masks!

Access-list 1 permit 172.16.0.0 0.0.255.255

- A **wildcard mask** is a 32-bit quantity that is divided into four octets.
- A wildcard mask is paired with an IP address.
- The numbers one and zero in the mask are used to identify how to treat the corresponding IP address bits.
- The term wildcard masking is a nickname for the ACL mask-bit matching process and comes from of an analogy of a wildcard that matches any other card in the game of poker.
- Wildcard masks have no functional relationship with subnet masks.
  - They are used for different purposes and follow different rules.
- Subnet masks start from the left side of an IP address and work towards the right to extend the network field by borrowing bits from the host field.
- Wildcard masks are designed to filter individual or groups of IP addresses permitting or denying access to resources based on the address.

Wildcard Masks!

Access-list 1 permit 172.16.0.0 0.0.255.255

- “Trying to figure out how wildcard masks work by relating them to subnet masking will only confuse the entire matter. The only similarity between a wildcard mask and a subnet mask is that they are both thirty-two bits long and use ones and zeros for the mask.”
- This is not entirely true.
- Although it is very important that you understand how a wildcard mask works, it can also be thought as an **inverse subnet mask**.
- We will see examples in a moment…

Wildcard Masks!

Test Condition

Cabrillo College

Access-list 1 permit 172.16.0.0 0.0.255.255

Test Conditon

172.16.0.0 10101100.00010000.00000000.00000000

0.0.255.255 00000000.00000000.11111111.11111111

A Match…

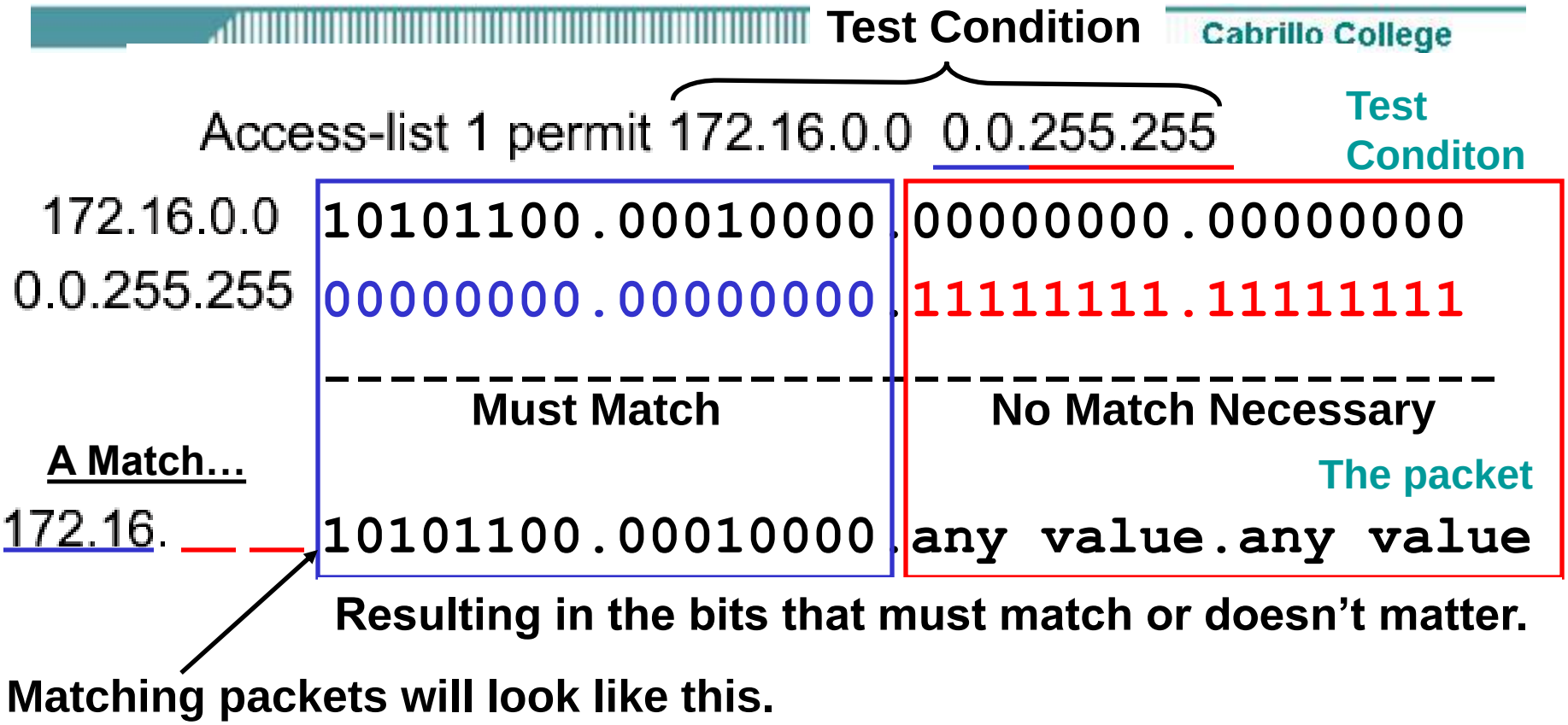
Matching packets will look like this…

The packet

172.16. __ __ 10101100.00010000.any value.any value

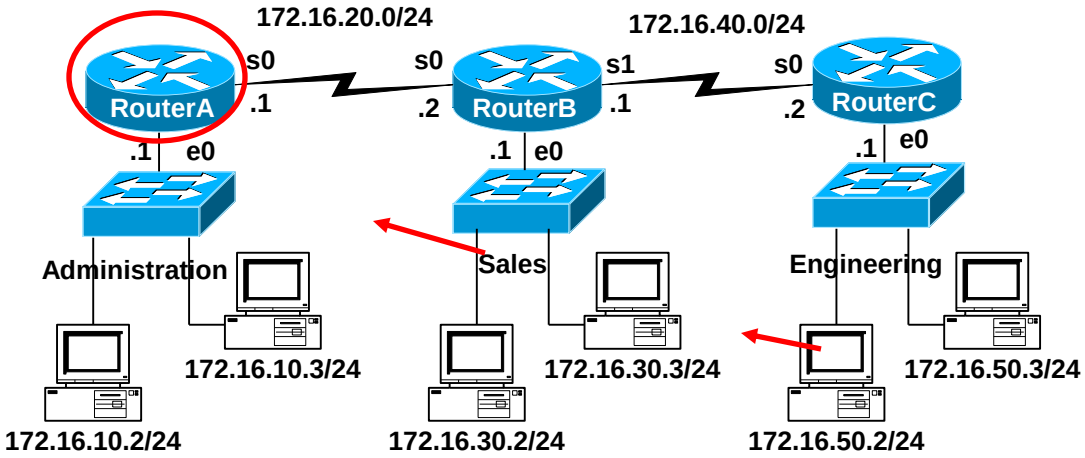
- Wildcard masking used to identify how to treat the corresponding IP address bits.
  - **0** - "**check** the corresponding bit value."
  - **1** - "**do not check** (ignore) that corresponding bit value."
- A **zero** in a bit position of the access list mask indicates that the corresponding bit in the address must be checked and must match for condition to be true.
- A **one** in a bit position of the access list mask indicates the corresponding bit in the address is not "interesting", does not need to match, and can be ignored.

Wildcard Masks!



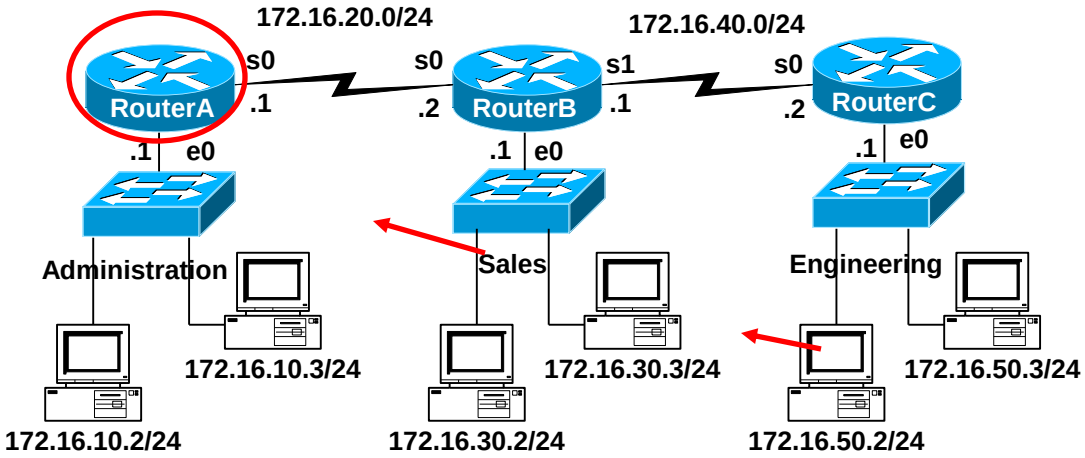
- **0** - "**check** the corresponding bit value."
- **1** - "**do not check** (ignore) that corresponding bit value."

Example 4 – Using Wildcard Masks



- Task:
  - Want RouterA to permit entire sales network and just the 172.16.50.2 station.
  - Deny all other traffic from entering Administrative network.

Example 4 – Using Wildcard Masks




```
RouterA(config)#access-list 11 permit 172.16.30.0 0.0.0.255
RouterA(config)#access-list 11 permit 172.16.50.2 0.0.0.0
```

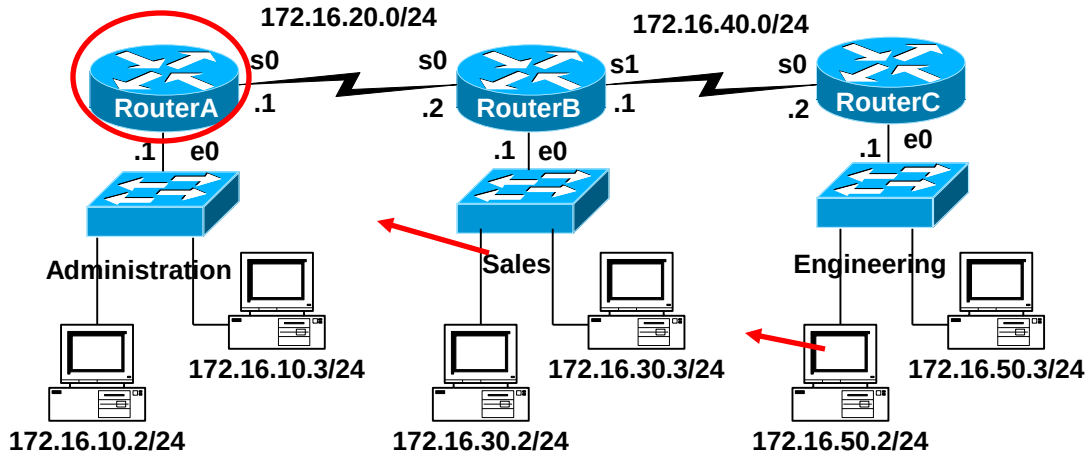
172.16.30.0 **0.0.0.255**

- 0 check - make sure first octet is 172
- 0 check - make sure second octet is 16
- 0 check - make sure third octet is 30
- 255 - don't check (*permit* any fourth octet)

172.16.50.2 **0.0.0.0**

- 0 check - make sure first octet is 172
- 0 check - make sure second octet is 16
- 0 check - make sure third octet is 50
- 0 check - make sure fourth octet is 2

Example 4 – Using Wildcard Masks



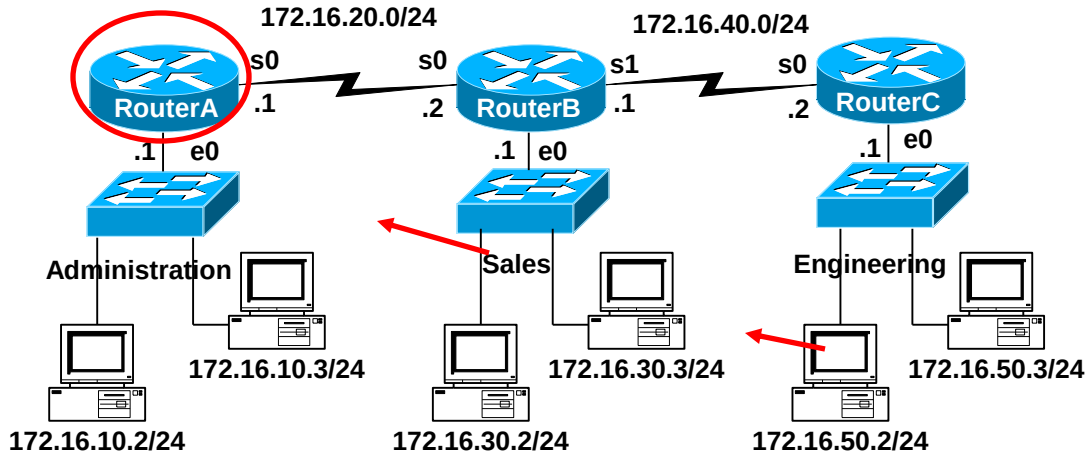

```
RouterA(config)#access-list 11 permit 172.16.30.0 0.0.0.255
0 = check, we want this to match, 1 = don't check (don't care)
```

```
172.16.30.0     10101100 . 00010000 . 00011110 . 00000000
0.0.0.255       00000000 . 00000000 . 00000000 . 11111111
                ---------------------------------------

172.16.30.0     10101100 . 00010000 . 00011110 . 00000000
172.16.30.1     10101100 . 00010000 . 00011110 . 00000001
                                                 ... (through)
172.16.30.255   10101100 . 00010000 . 00011110 . 11111111
```

Test Conditon

The packet(s)

Example 4 – Using Wildcard Masks



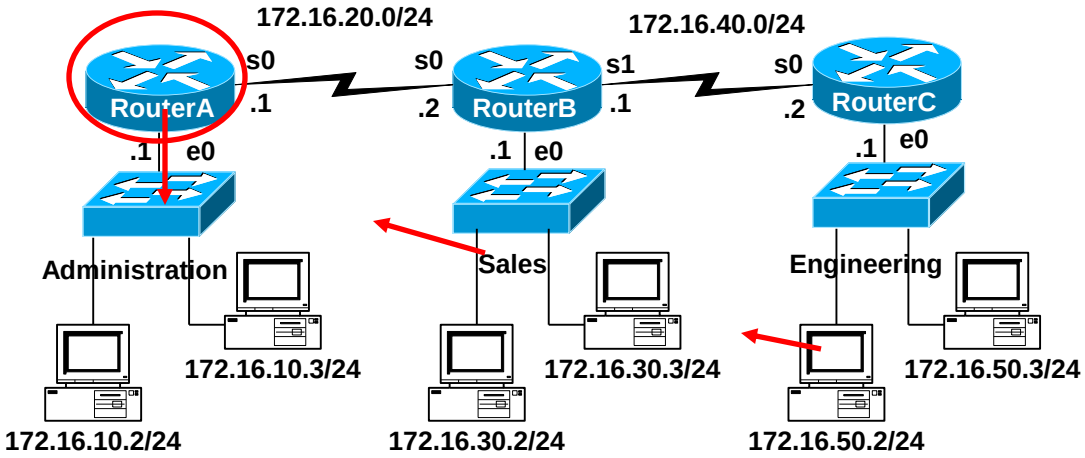

```
RouterA(config)#access-list 11 permit 172.16.50.2 0.0.0.0
0 = check, we want this to match, 1 = don't check (don't care)
```

```
172.16.50.2     10101100 . 00010000 . 00110010 . 00000010
0.0.0.0         00000000 . 00000000 . 00000000 . 00000000
                ------------------------------------------
172.16.50.2     10101100 . 00010000 . 00110010 . 00000010
```

Test Conditon

The packet(s)

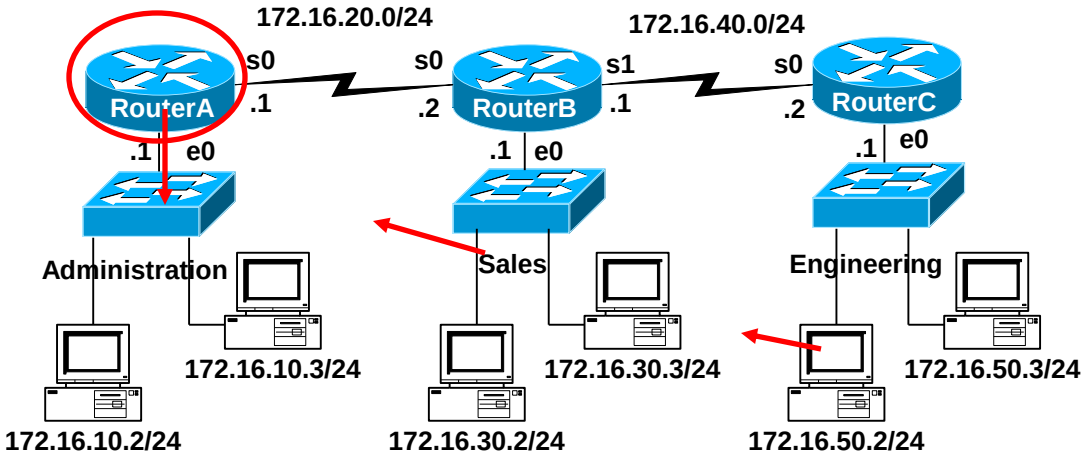
Example 4 – Using Wildcard Masks



Don't forget to apply the access-list to an interface.

```
RouterA(config)#access-list 11 permit 172.16.30.0 0.0.0.255
RouterA(config)#access-list 11 permit 172.16.50.2 0.0.0.0

RouterA(config)# interface e 0
RouterA(config-if)#ip access-group 11 out
```

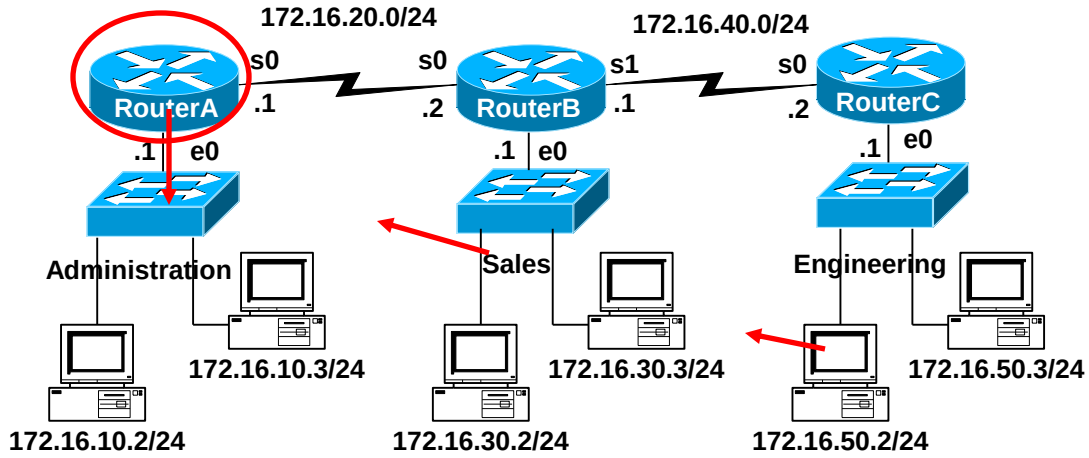
Example 4 – Using Wildcard Masks



Remember that implicit deny any? It's a good idea for beginners to include the deny any statement just as a reminder.

```
RouterA(config)#access-list 11 permit 172.16.30.0 0.0.0.255
RouterA(config)#access-list 11 permit 172.16.50.2 0.0.0.0
RouterA(config)#access-list 11 deny 0.0.0.0 255.255.255.255

RouterA(config)# interface e 0
RouterA(config-if)#ip access-group 11 out
```

Example 4 – Using Wildcard Masks



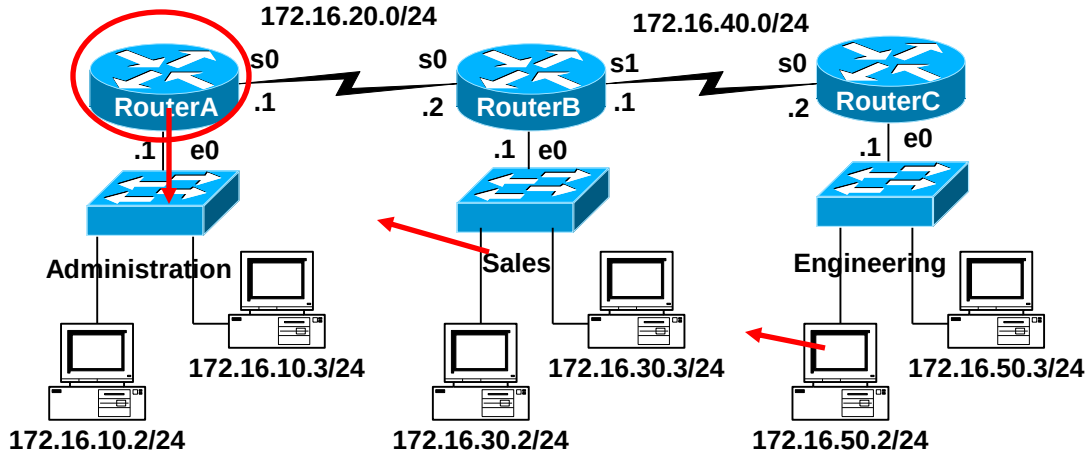

```
RouterA(config)#access-list 11 deny 0.0.0.0 255.255.255.255
0 = check, we want this to match, 1 = don't check (don't care)
```

```
0.0.0.0           00000000 . 00000000 . 00000000 . 00000000
255.255.255.255   11111111 . 11111111 . 11111111 . 11111111
                  ------------------------------------------

0.0.0.0           00000000 . 00000000 . 00000000 . 00000000
0.0.0.1           00000000 . 00000000 . 00000000 . 00000001
                                 ... (through)
255.255.255.255   11111111 . 11111111 . 11111111 . 11111111
```

Test Conditon

The packet(s)

“any” keyword

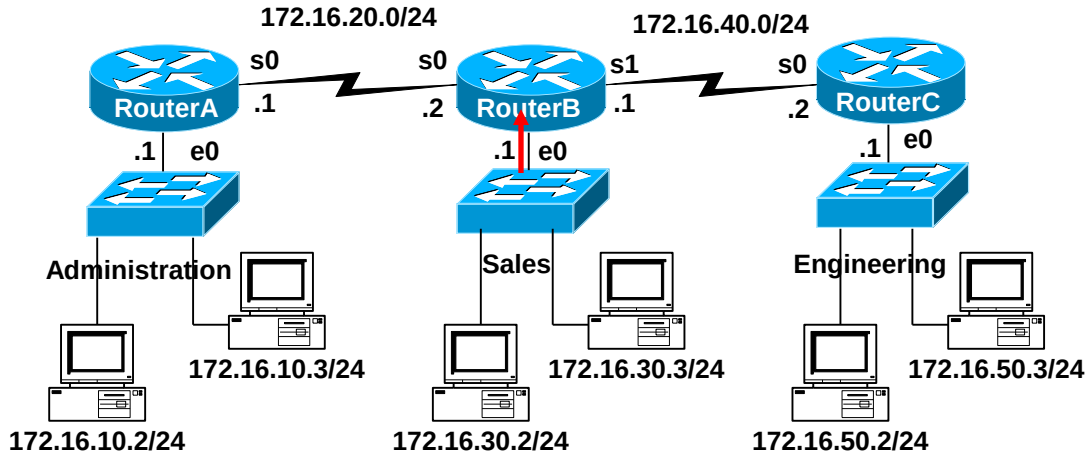



```
RouterA(config)#access-list 11 deny 0.0.0.0 255.255.255.255
Or
RouterA(config)#access-list 11 deny any
```

any = 0.0.0.0 255.255.255.255

- Simply put, the **any** option substitutes 0.0.0.0 for the IP address and 255.255.255.255 for the wildcard mask.
- This option will match any address that it is compared against.

“any” keyword – From Example 3

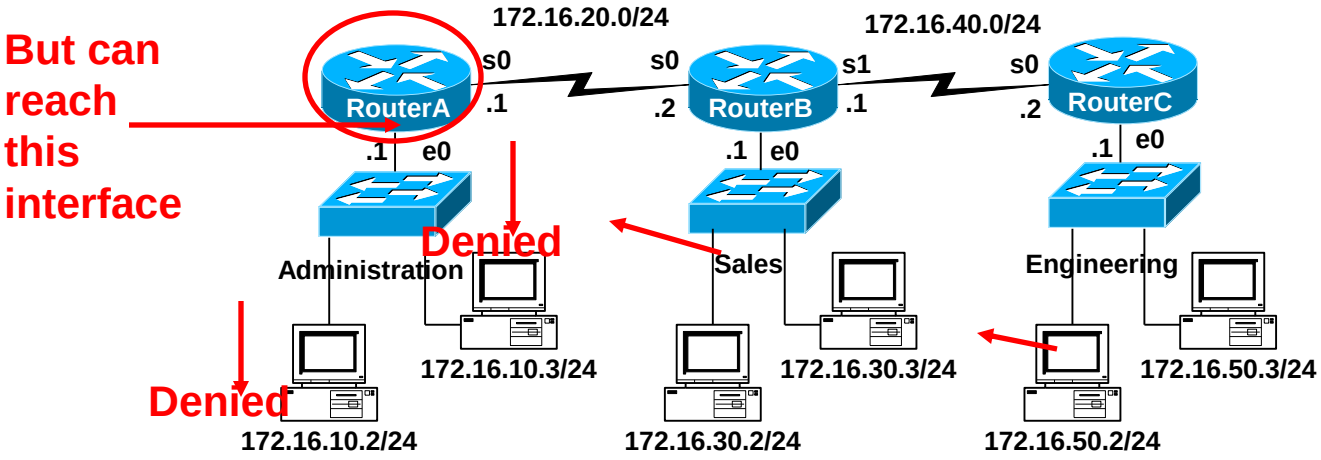



```
RouterB(config)#access-list 10 deny 172.16.30.2
RouterB(config)#access-list 10 permit any
or
RouterB(config)#access-list 10 permit 0.0.0.0 255.255.255.255
```

Previous example:

- Deny only the host 172.16.30.2 from exiting the Sales network.
- Permit all other hosts on the Sales network to leave the 172.16.30.0/24 network.
- Keyword “any” can be used to represent all IP Addresses.

A note about outbound access lists




```
RouterA(config)#access-list 11 permit 172.16.30.0 0.0.0.255
RouterA(config)#access-list 11 permit 172.16.50.2 0.0.0.0
RouterA(config)#access-list 11 deny 0.0.0.0 255.255.255.255
RouterA(config)# interface e 0
RouterA(config-if)#ip access-group 11 out
```

This will deny packets from 172.16.30.0/24 from reaching all devices in the 172.16.10.0/24 Administration LAN, except RouterA's Ethernet 0 interface, of 172.16.10.1. The access list will need to be applied on Router A's Serial 0 interface for it to be denied on RouterA's Ethernet 0 interface. A better soluton is to use an Extended Access list. (coming)

Practice


```
RouterB(config)#access-list 10 permit _____ _____
```

Permit the following networks:

	Network/Subnet Mask	Address/Wildcard Mask
A.	172.16.0.0	255.255.0.0
B.	172.16.1.0	255.255.255.0
C.	192.168.1.0	255.255.255.0
D.	172.16.16.0	255.255.240.0 (hmmm . . .?)
E.	172.16.128.0	255.255.192.0 (hmmm . . .?)

Permit the following hosts:

	Network/Subnet Mask	Address/Wildcard Mask
A.	172.16.10.100	
B.	192.168.1.100	
C.	All hosts	

Practice – Do you see a relationship?


```
RouterB(config)#access-list 10 permit _____ _____
```

Permit the following networks:

	Network/Subnet Mask	Address/Wildcard Mask
A.	172.16.0.0 255.255.0.0	172.16.0.0 0.0.255.255
B.	172.16.1.0 255.255.255.0	172.16.1.0 0.0.0.255
C.	192.168.1.0 255.255.255.0	192.168.1.0 0.0.0.255
D.	172.16.32.0 255.255.240.0	172.16.32.0 0.0.15.255
E.	172.16.128.0 255.255.192.0	172.16.128 0.0.63.255

Permit the following hosts:

	Network/Subnet Mask	Address/Wildcard Mask
A.	172.16.10.100	172.16.10.100 0.0.0.0
B.	192.168.1.100	192.168.1.100 0.0.0.0
C.	All hosts	0.0.0.0 255.255.255.255

Answers Explained

A. `172.16.0.0 0.0.255.255`

```
RouterB(config)#access-list 10 permit 172.16.0.0 0.0.255.255
```

0 = check, we want this to match

1 = don't check, this can be any value, does not need to match

Test Conditon

```
172.16.0.0      10101100 . 00010000 . 00000000 . 00000000
0.0.255.255     00000000 . 00000000 . 11111111 . 11111111
                ---------------------------------------
172.16.0.0      10101100 . 00010000 . 00000000 . 00000000
172.16.0.1      10101100 . 00010000 . 00000000 . 00000001
172.16.0.2      10101100 . 00010000 . 00000000 . 00000010
                                          ... (through)
172.16.255.255  10101100 . 00010000 . 11111111 . 11111111
```

Matching packets will look like this.

The packet(s)

Answers Explained

D. 172.16.32.0 255.255.240.0

RouterB(config)#**access-list 10 permit 172.16.32.0 0.0.15.255**

0 = check, we want this to match

1 = don't check, this can be any value, does not need to match

Test Conditon

```
172.16.16.0     10101100 . 00010000 . 00100000 . 00000000
0.0.15.255      00000000 . 00000000 . 00001111 . 11111111
                ---------------------------------------

172.16.16.0     10101100 . 00010000 . 00100000 . 00000000
172.16.16.1     10101100 . 00010000 . 00100000 . 00000001
172.16.16.2     10101100 . 00010000 . 00100000 . 00000010
                                          ... (through)
172.16.16.255   10101100 . 00010000 . 00101111 . 11111111
```

The packet(s)

Packets belonging to the 172.16.32.0/20 network will match this condition because they have the same 20 bits in common.

There is a relationship!
Bitwise-not on the Subnet Mask

D. 172.16.32.0 255.255.240.0

RouterB(config)#**access-list 10 permit 172.16.32.0 0.0.15.255**

```
Subnet Mask:          255 . 255 . 240 .   0
Wildcard Mask:   +      0 .   0 .  15 . 255
                      -----------------------
                      255 . 255 . 255 . 255
```

So, we could calculate the Wildcard Mask by:

```
                      255 . 255 . 255 . 255
Subnet Mask:      -   255 . 255 . 240 .   0
                      -----------------------
Wildcard Mask:         0 .   0 .  15 .  255
```

255.255.255.255 – Subnet = Wildcard


```
RouterB(config)#access-list 10 permit __________ __________
```

Permit the following networks:

	255.255.255.255. - Subnet Mask	=	Wildcard Mask
A.	255.255.255.255 – 255.255.0.0	=	0.0.255.255
B.	255.255.255.255 – 255.255.255.0	=	0.0.0.255
C.	255.255.255.255 – 255.255.255.0	=	0.0.0.255
D.	255.255.255.255 – 255.255.240.0	=	0.0.15.255
E.	255.255.255.255 – 255.255.192.0	=	0.0.63.255

Permit the following hosts: (host routes have a /32 mask)

	255.255.255.255. - /32 Mask	=	Wildcard Mask
A.	255.255.255.255 – 255.255.255.255	=	0.0.0.0
B.	255.255.255.255 – 255.255.255.255	=	0.0.0.0

255.255.255.255 – Subnet = Wildcard


```
RouterB(config)#access-list 10 permit _____ _____
```

Permit the following networks:

	Network/Subnet Mask	Address/Wildcard Mask
A.	172.16.0.0 255.255.0.0	172.16.0.0 0.0.255.255
B.	172.16.1.0 255.255.255.0	172.16.1.0 0.0.0.255
C.	192.168.1.0 255.255.255.0	192.168.1.0 0.0.0.255
D.	172.16.32.0 255.255.240.0	172.16.32.0 0.0.15.255
E.	172.16.128.0 255.255.192.0	172.16.128 0.0.63.255

Permit the following hosts:

	Network/Subnet Mask	Address/Wildcard Mask
A.	172.16.10.100	172.16.10.100 0.0.0.0
B.	192.168.1.100	192.168.1.100 0.0.0.0
C.	All hosts or “any”	0.0.0.0 255.255.255.255

“host” option


```
RouterB(config)#access-list 10 permit 192.168.1.100 0.0.0.0
RouterB(config)#access-list 10 permit host 192.168.1.100
```

Permit the following hosts:

	Network/Subnet Mask	Address/Wildcard Mask
A.	172.16.10.100	172.16.10.100 0.0.0.0
B.	192.168.1.100	192.168.1.100 0.0.0.0

- The **host** option substitutes for the 0.0.0.0 mask.
- This mask requires that all bits of the ACL address and the packet address match.
- The host keyword precedes the IP address.
- This option will match just one address.

172.16.10.100 0.0.0.0 ***replaced by*** **host 172.16.10.100**

192.168.1.100 0.0.0.0 ***replaced by*** **host 192.168.1.100**

Ranges with Wildcard Masks - Extra

- Wildcard masks can be used to define “some” ranges of IP address.
- Note:
  - It is possible to get overly complicate your access lists when trying to do a range.
  - Many times using multiple access lists are easier to configure, easier to understand, and you are less likely to make a mistake.
  - We will do our best to understand this, but it is not imperative that you do.
  - If you are with me so far, but I lose you here, don’t worry about it.
- For example:
  - The administrator wants to use IP wildcard masking bits to permit, match subnets **172.30.16.0 to 172.30.31.0**.
  - access-list 20 permit 172.30.16.0 0.0.15.255

Ranges with Wildcard Masks

Match subnets **172.30.16.0 *to* 172.30.31.0**

```
access-list 20 permit 172.30.16.0 0.0.15.255
```

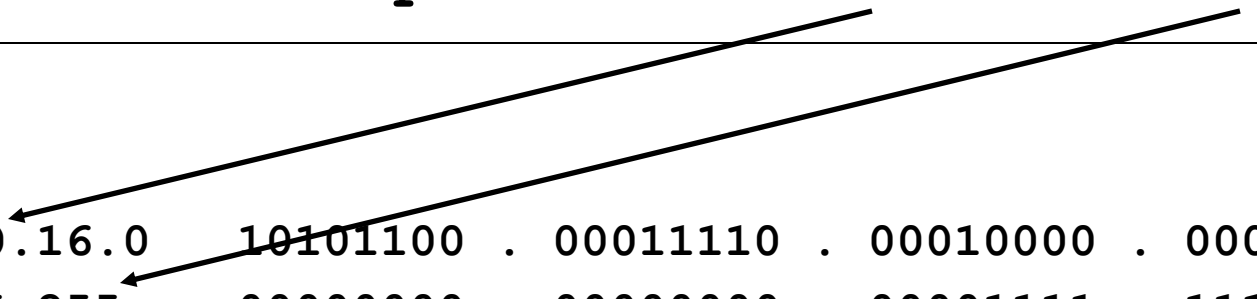
What's happening (*we'll see its easier than this*):

- The easiest way to see how we did this is to show it in binary…

Ranges with Wildcard Masks

Match subnets **172.30.16.0 *to* 172.30.31.0**

```
access-list 20 permit 172.30.16.0 0.0.15.255
```



```
172.30.16.0     10101100 . 00011110 . 00010000 . 00000000
0.0.15.255      00000000 . 00000000 . 00001111 . 11111111
                ---------------------------------------
172.30.16.0     10101100 . 00011110 . 00010000 . 00000000
172.30.16.1     10101100 . 00011110 . 00010000 . 00000001
                                                through . . .
172.30.31.254   10101100 . 00011110 . 00011111 . 11111110
172.30.31.255   10101100 . 00011110 . 00011111 . 11111115
```

Ranges with Wildcard Masks

Match subnets **172.30.16.0 *to* 172.30.31.0**

```
access-list 20 permit 172.30.16.0 0.0.15.255
```

Must match

```
172.30.16.0     10101100 . 00011110 . 00010000 . 00000000
0.0.15.255      00000000 . 00000000 . 00001111 . 11111111
                ---------------------------------------
172.30.16.0     10101100 . 00011110 . 00010000 . 00000000
172.30.16.1     10101100 . 00011110 . 00010000 . 00000001
                                                through . . .
172.30.31.254   10101100 . 00011110 . 00011111 . 11111110
172.30.31.255   10101100 . 00011110 . 00011111 . 11111115
```

Ranges with Wildcard Masks

Match subnets **172.30.16.0 *to* 172.30.31.0**

```
access-list 20 permit 172.30.16.0 0.0.15.255
```

Any Value

```
172.30.16.0     10101100 . 00011110 . 00010000 . 00000000
0.0.15.255      00000000 . 00000000 . 00001111 . 11111111
                ---------------------------------------
172.30.16.0     10101100 . 00011110 . 00010000 . 00000000
172.30.16.1     10101100 . 00011110 . 00010000 . 00000001
                                                through . . .
172.30.31.254   10101100 . 00011110 . 00011111 . 11111110
172.30.31.255   10101100 . 00011110 . 00011111 . 11111115
```

Ranges with Wildcard Masks

Match subnets **172.30.16.0 *to* 172.30.31.0**

```
access-list 20 permit 172.30.16.0 0.0.15.255
```

Must match **Any Value**

```
172.30.16.0     10101100 . 00011110 . 00010000 . 00000000
0.0.15.255      00000000 . 00000000 . 00001111 . 11111111
                ---------------------------------------
172.30.16.0     10101100 . 00011110 . 00010000 . 00000000
172.30.16.1     10101100 . 00011110 . 00010000 . 00000001
                                                through . . .
172.30.31.254   10101100 . 00011110 . 00011111 . 11111110
172.30.31.255   10101100 . 00011110 . 00011111 . 11111111
```

Ranges with Wildcard Masks

Match subnets **172.30.16.0 *to* 172.30.31.0**

```
access-list 20 permit 172.30.16.0 0.0.15.255
```

	Must match	Any Value
172.30.16.0	10101100 . 00011110 . 00010000	. 00000000
0.0.15.255	00000000 . 00000000 . 00001111	. 11111111

172.30.16.0	10101100 . 00011110 . 00010000	. 00000000
through . . .		
172.30.31.255	10101100 . 00011110 . 00011111	. 11111111

- The subnets 172.30.16.0 *through* 172.30.31.0 have the subnet mask 255.255.240.0 in common.
- This gives us the wildcard mask: 0.0.15.255 (255.255.255.255 – 255.255.240.).
- Using the first permitted subnet, 172.30.16.0, gives us the address for our test condition.
- This will not work for all ranges but does in some cases like this one.

Verifying Access Lists


```
Router#show ip interface
FastEthernet0/0 is up, line protocol is down
  Internet address is 192.168.1.1/24
  Broadcast address is 255.255.255.255
  MTU is 1500 bytes
  Helper address is not set
  Directed broadcast forwarding is disabled
  Outgoing access list is not set
  Inbound  access list is 2
Serial0/0 is down, line protocol is down
  Internet address is 200.200.2.1/24
  Broadcast address is 255.255.255.255
  MTU is 1500 bytes
  Helper address is not set
  Directed broadcast forwarding is disabled
  Outgoing access list is not set
  Inbound access list is 101
```

Verifying Access Lists


```
Router#show access-lists
Standard IP access list 2
 deny   172.16.1.1
 permit 172.16.1.0, wildcard bits 0.0.0.255
 deny   172.16.0.0, wildcard bits 0.0.255.255
 permit 172.0.0.0, wildcard bits 0.255.255.255
Extended IP access list 101
 permit tcp 192.168.6.0 0.0.0.255 any eq telnet
 permit tcp 192.168.6.0 0.0.0.255 any eq ftp
 permit tcp 192.168.0.0 0.0.0.255 any eq ftp-data
Router#
```

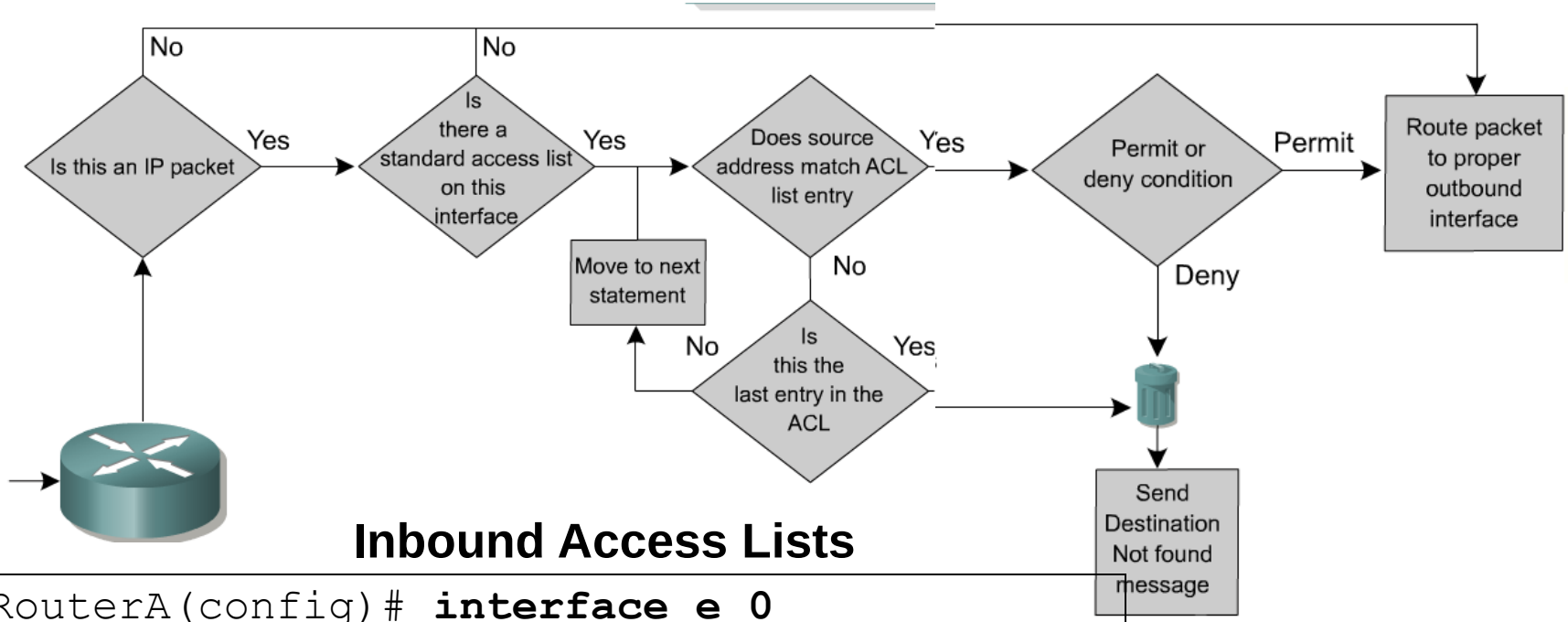
Verifying Access Lists


```
Router#show running-config
Current configuration : 953 bytes
!
interface FastEthernet0/0
 ip address 192.168.1.1 255.255.255.0
 ip access-group 2 in
!
interface Serial0/0
 ip address 200.200.2.1 255.255.255.0
 ip access-group 101 in
!
access-list 2 deny   172.16.1.1
access-list 2 permit 172.16.1.0 0.0.0.255
access-list 2 deny   172.16.0.0 0.0.255.255
access-list 2 permit 172.0.0.0 0.255.255.255
access-list 101 permit tcp 192.168.6.0 0.0.0.255 any
eq telnet
access-list 101 permit tcp 192.168.6.0 0.0.0.255 any
eq ftp
!
```

- Note: More than one interface can use the same access-list.

Part 2: ACL Operations

Inbound Standard Access Lists



Inbound Access Lists

```
RouterA(config)# interface e 0
RouterA(config-if)#ip access-group 11 in
```

- With **inbound** Access Lists the IOS checks the packets ***before*** it is sent to the Routing Table Process.
- With **outbound** Access Lists, the IOS checks the packets ***after*** it is sent to the Routing Table Process, except destined for the router's own interface.
  - This is because the output interface is not known until the forwarding decision is made.

Standard ACL


```
access-list 2 deny   172.16.1.1
access-list 2 permit 172.16.1.0  0.0.0.255
access-list 2 deny   172.16.0.0  0.0.255.255
access-list 2 permit 172.0.0.0   0.255.255.255
```

- Access list number range of 1-99
- Filter only on source IP address
- Wildcard masks
- Applied to port closest to destination ← **We will see why in a moment.**

The **full syntax** of the standard ACL command is:

`Router(config)#`**`access-list`** *`access-list-number`* `{deny | permit} source [`*`source-wildcard`*` ] [`**`log`**`]`

The **no form** of this command is used to remove a standard ACL. This is the syntax: (Deletes entire ACL!)

`Router(config)#`**`no access-list`** *`access-list-number`*

Extended Access Lists


```
Router(config)#access-list access-list-number {permit | deny}
protocol source
[source-mask destination destination-mask operator operand]
[established]
```

Paramter	Description
access-list-number	Identifies the list using a number in the range 100 to 199.
permit | **deny**	Indicates whether this entry allows or blocks the specified address.
protocol	The protocol, such as IP, TCP, UDP, ICMP, GRE, or IGRP.
source and **destination**	Identifies source and destination addresses.
source-mask and *destination-mask*	Wildcard mask; zeros indicate positions that must match, ones indicate do not care positions.
operator *operand*	lt, gt, eq, neq (less than, greater than, equal, not equal), and a port number.
established	Allows TCP traffic to pass if the packet uses an established connection (for example, has ACK bits set).

Extended Access Lists


```
Router(config)#access-list access-list-number {permit | deny}
protocol source
[source-mask destination destination-mask operator operand]
[established]
```

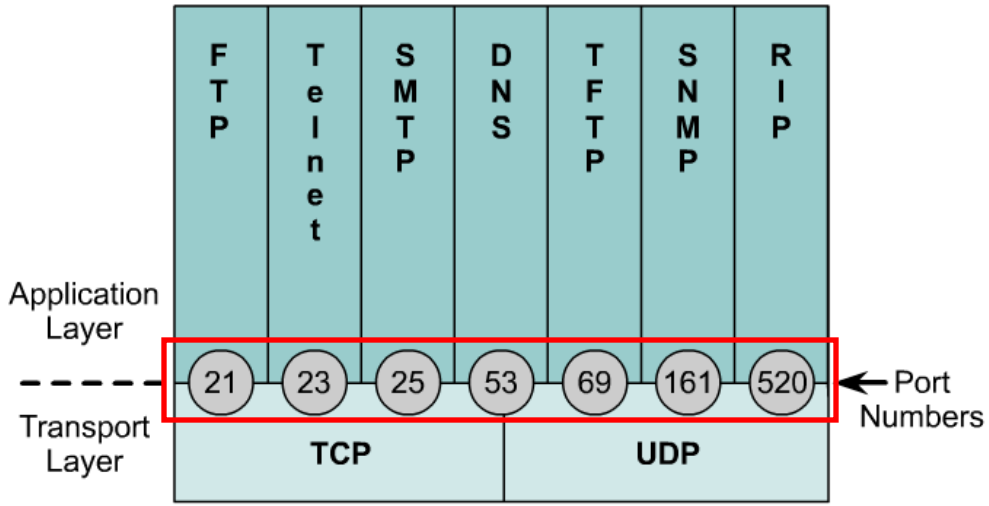
Protocol	Range
IP	1-99
Extended IP	100-199
AppleTalk	600-699
IPX	800-899
Extended IPX	900-999
IPX Service Advertising Protocol	1000-1099

- Extended ACLs are used more often than standard ACLs because they provide a **greater range of control**.
- Extended ACLs **check the source and destination packet addresses** as well as being able to **check for protocols and port numbers**.
- This gives **greater flexibility** to describe what the ACL will check.
- Packets can be permitted or denied access based on where the packet originated and its destination as well as protocol type and port addresses.

Extended Access Lists


```
Router(config)#access-list access-list-number {permit | deny}
protocol source
[source-mask destination destination-mask operator operand]
[established]
```

protocol	The protocol, such as IP, TCP, UDP, ICMP, GRE, or IGRP.
operator operand	lt, gt, eq, neq (less than, greater than, equal, not equal), and a port number.



- **Operator and operand** can also refer to ICMP Types and Codes or whatever the **protocol** is being checked.
- If the **operator and operand** follow the **source address** it refers to the **source port**
- If the **operator and operand** follow the **destination address** it refers to the **destination port**.

Extended Access Lists - Examples

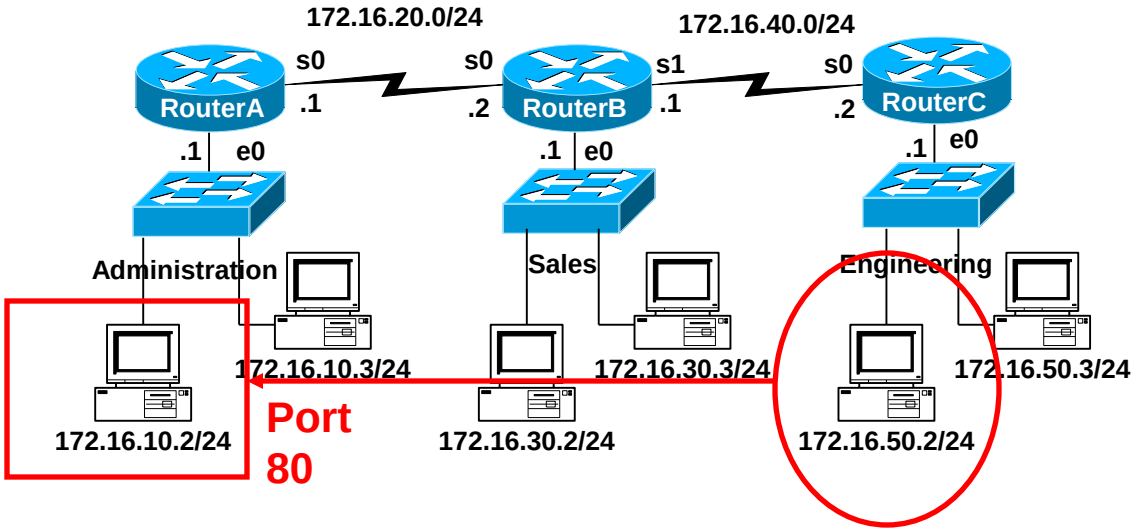

```
access-list 114 permit tcp 172.16.6.0 0.0.0.255 any eq telnet
access-list 114 permit tcp 172.16.6.0 0.0.0.255 any eq ftp
access-list 114 permit tcp 172.16.6.0 0.0.0.255 any eq ftp-data
```

port number or protocol name

- Access list number range of 100-199
- Source destination IP address
- Layer 4 protocol number
- Applied to port closest to source host

- The **`ip access-group`** command links an existing extended ACL to an interface.
- Remember that only one ACL per interface, per direction, per protocol is allowed. The format of the command is:
- `Router(config-if)#` **`ip access-group`** *`access-list-number`* `{in | out}`

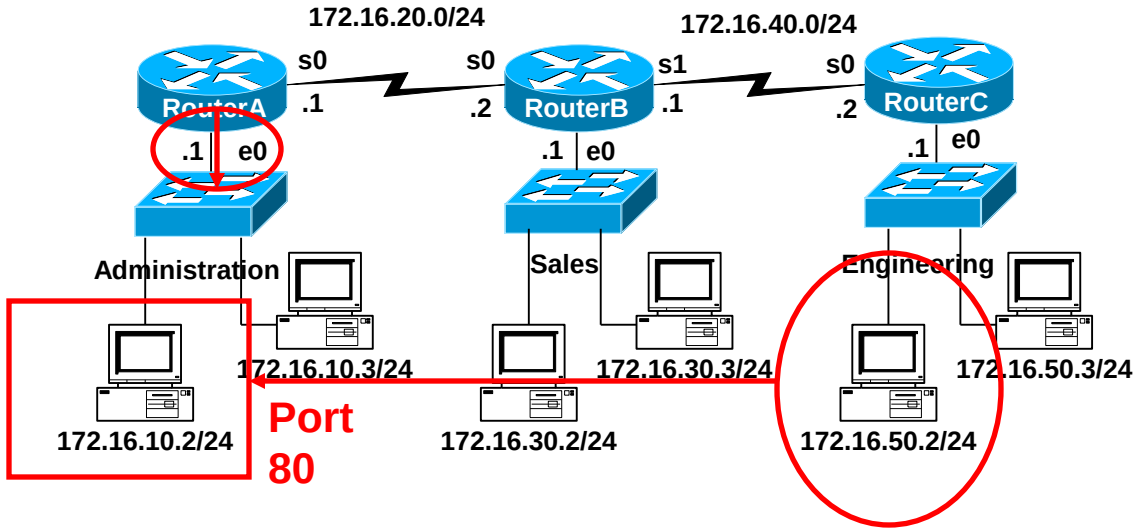
Example 1



Task

- What if we wanted Router A to permit only the Engineering workstation 172.16.50.2 to be able to access the web server in Administrative network with the IP address 172.16.10.2 and port address 80.
- All other traffic is denied.

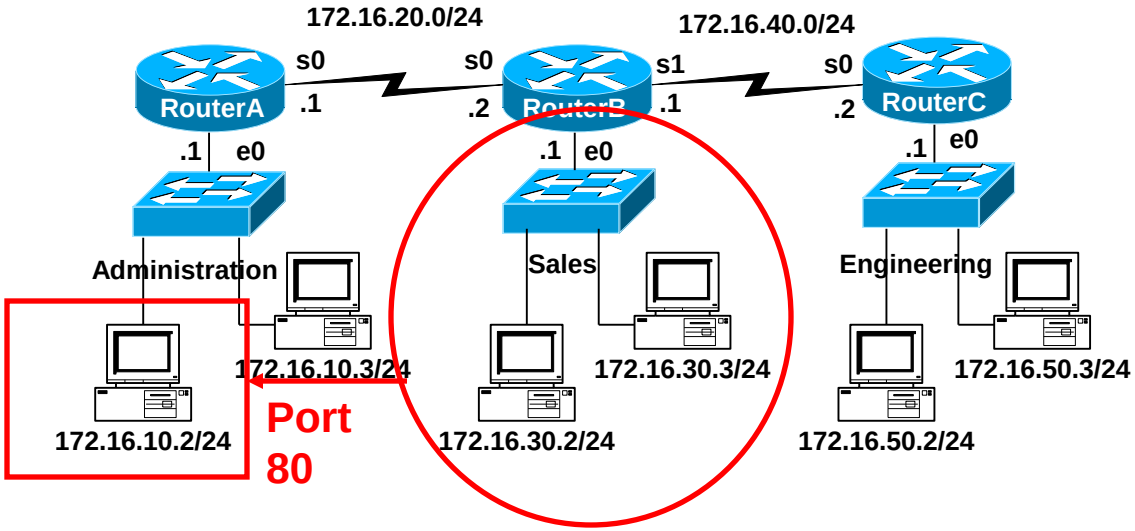
Example 1




```
RouterA(config)#access-list 110 permit tcp host 172.16.50.2
   host 172.16.10.2 eq 80

RouterA(config)#inter e 0
RouterA(config-if)#ip access-group 110 out
```

- Why is better to place the ACL on RouterA instead of RouterC?
- Why is the e0 interface used instead of s0 on RouterA?
- We'll see in a moment!

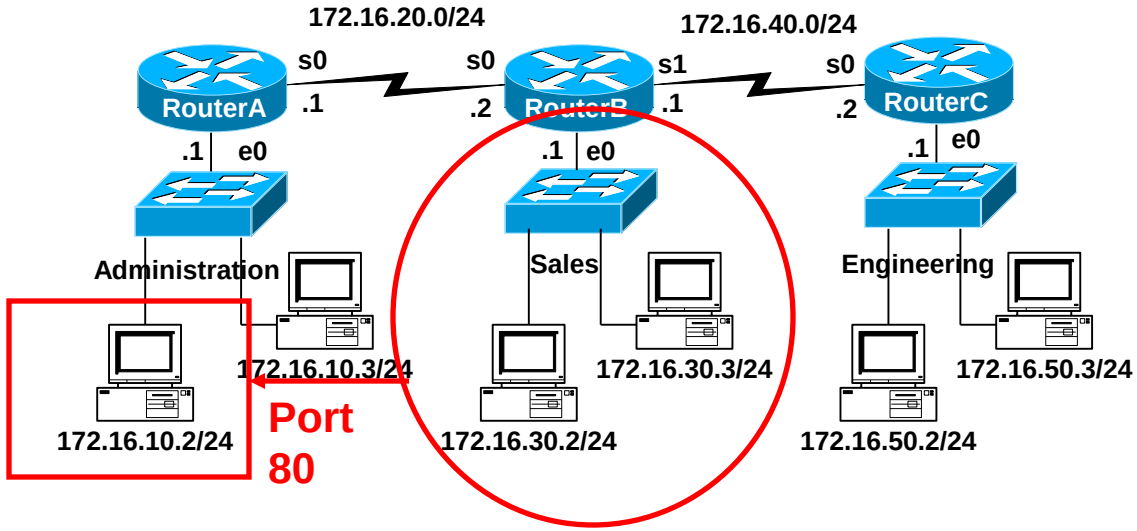
Example 2



Task

- What if we wanted Router A to permit any workstation on the Sales network be able to access the web server in Administrative network with the IP address 172.16.10.2 and port address 80.
- All other traffic is denied.

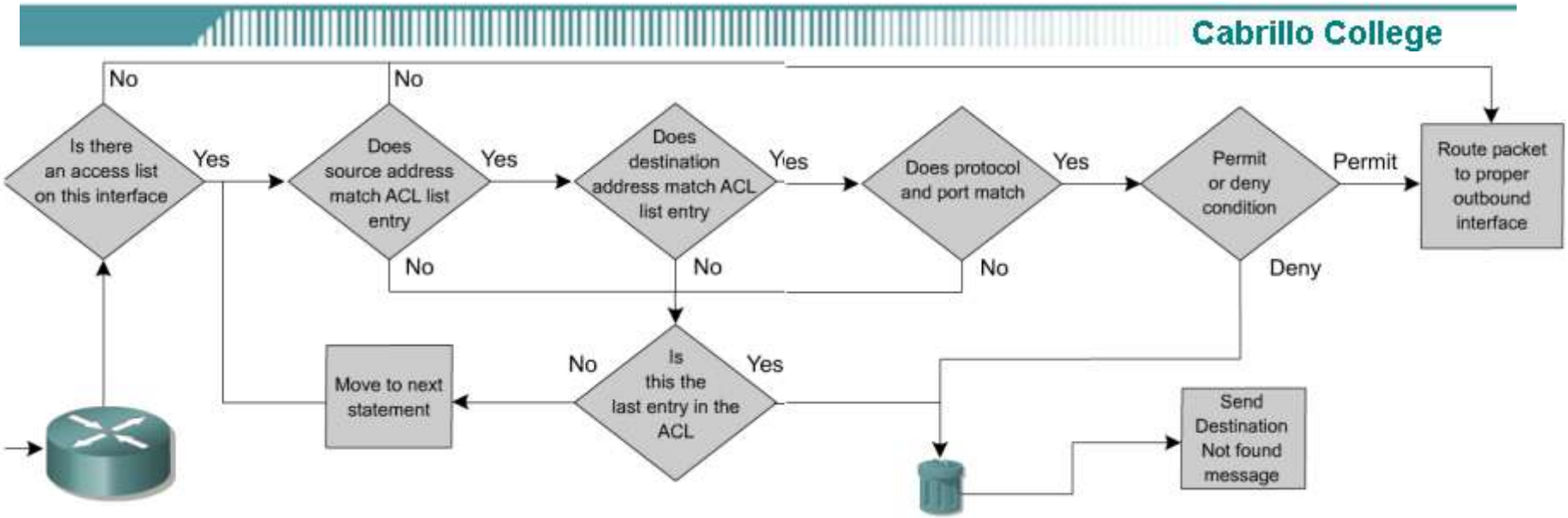
Example 2




```
RouterA(config)#access-list 110 permit tcp 172.16.30.0
   0.0.0.255 host 172.16.10.2 eq 80

RouterA(config)#inter e 0
RouterA(config-if)#ip access-group 110 out
```

- When configuring access list statements, use the **"?"** to walk yourself through the command!

Inbound Extended Access Lists



Inbound Access Lists

```
RouterA(config)# interface e 0
RouterA(config-if)#ip access-group 11 in
```

- With **inbound** Access Lists the IOS checks the packets ***before*** it is sent to the Routing Table Process.
- With **outbound** Access Lists, the IOS checks the packets ***after*** it is sent to the Routing Table Process.
  - This is because the output interface is not known until the forwarding decision is made.

Notes from www.cisco.com

- In the following example, the last entry is sufficient.
- You do not need the first three entries because TCP includes Telnet, and IP includes TCP, User Datagram Protocol (UDP), and Internet Control Message Protocol (ICMP).

```
access-list 101 permit tcp host 10.1.1.2 host 172.16.1.1 eq
   telnet
access-list 101 permit tcp host 10.1.1.2 host 172.16.1.1
access-list 101 permit udp host 10.1.1.2 host 172.16.1.1
access-list 101 permit ip 10.1.1.0 0.0.0.255 172.16.1.0
   0.0.0.255
```

Named ACLs


```
ip access-list {extended|standard} name
```

- IP named ACLs were introduced in Cisco IOS Software Release 11.2.
- Allows standard and extended ACLs to be given names instead of numbers.
- The advantages that a named access list provides are:
  - Intuitively identify an ACL using an alphanumeric name.
  - Eliminate the limit of 798 simple and 799 extended ACLs
  - Named ACLs provide the ability to modify ACLs without deleting and then reconfiguring them.
  - It is important to note that a named access list will allow the deletion of statements but will only allow for statements to be inserted at the end of a list.
  - Even with named ACLs it is a good idea to use a text editor to create them.

Named ACLs


```
Rt1(config)#ip access-list extended server-access
Rt1(config-ext-nacl)#permit TCP any host 131.108.101.99 eq
smtp
Rt1(config-ext-nacl)#permit UDP any host 131.108.101.99 eq
domain
Rt1(config-ext-nacl)#deny ip any any log
Rt1(config-ext-nacl)#^Z
Applying the named list:
Rt1(config)#interface fastethernet 0/0
Rt1(config-if)#ip access-group server-access out
Rt1(config-if)#^Z
```

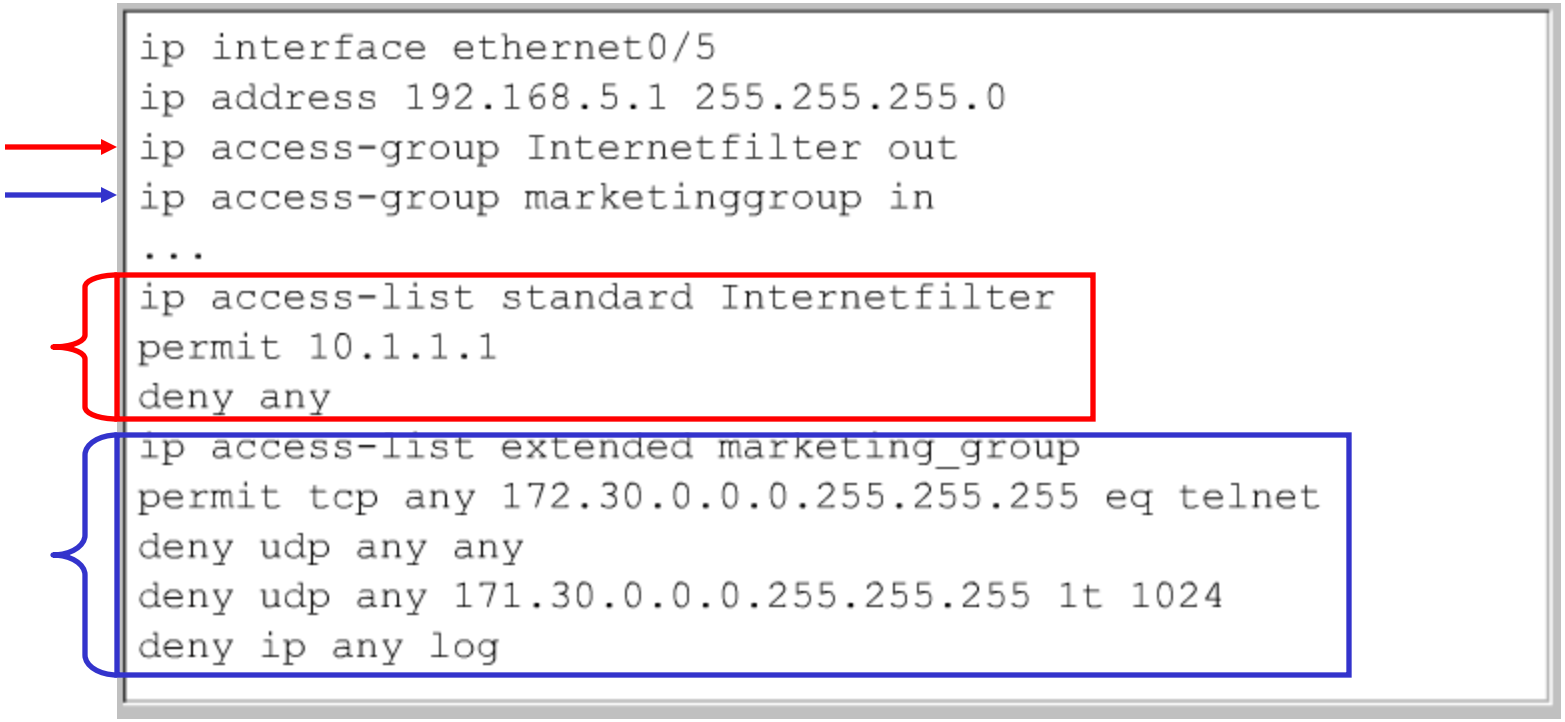
- A named ACL is created with the **`ip access-list`** command.
- This places the user in the ACL configuration mode.

Named ACLs


```
router(config-ext-nacl)#permit|deny protocol source
source-wildcard [operator [port]] destination
destination-wildcard [operator [port]] [established]
[precedence precedence] [tos tos] [log] [time-range time-
range-name]
```

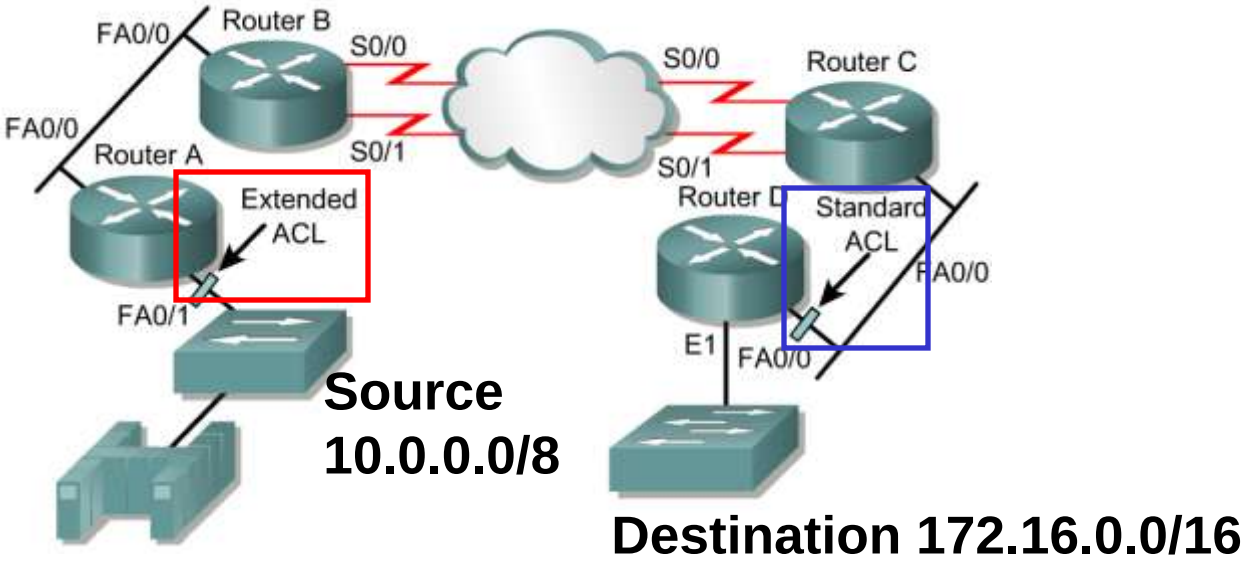
- In ACL configuration mode, specify one or more conditions to be permitted or denied.
- This determines whether the packet is passed or dropped when the ACL statement matches.

Named ACLs



The configuration shown creates a standard ACL named internetfilter and an extended ACL named marketing_group. The lists are applied to the Ethernet Interface 0/5.

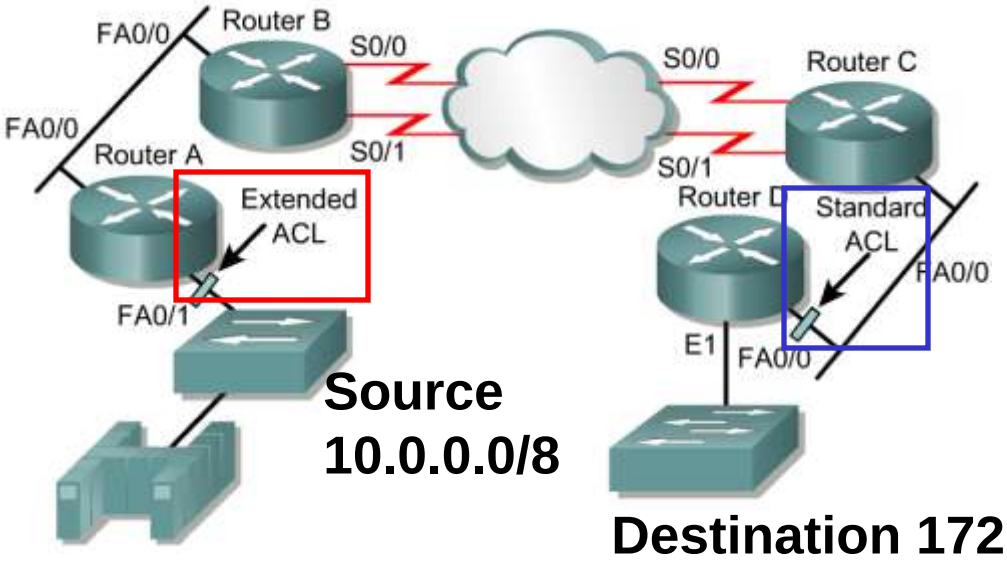
Placing ACLs



The general rule:

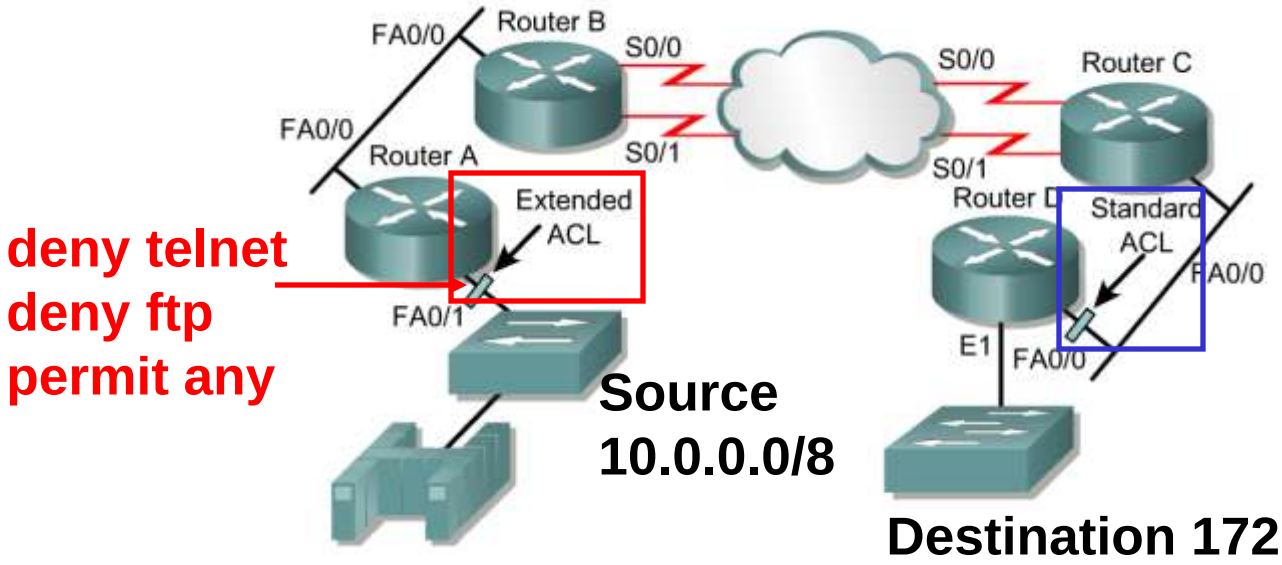
- **Standard ACLs** do not specify destination addresses, so they should be placed as close to the destination as possible.
- Put the **extended ACLs** as close as possible to the source of the traffic denied.

Placing ACLs



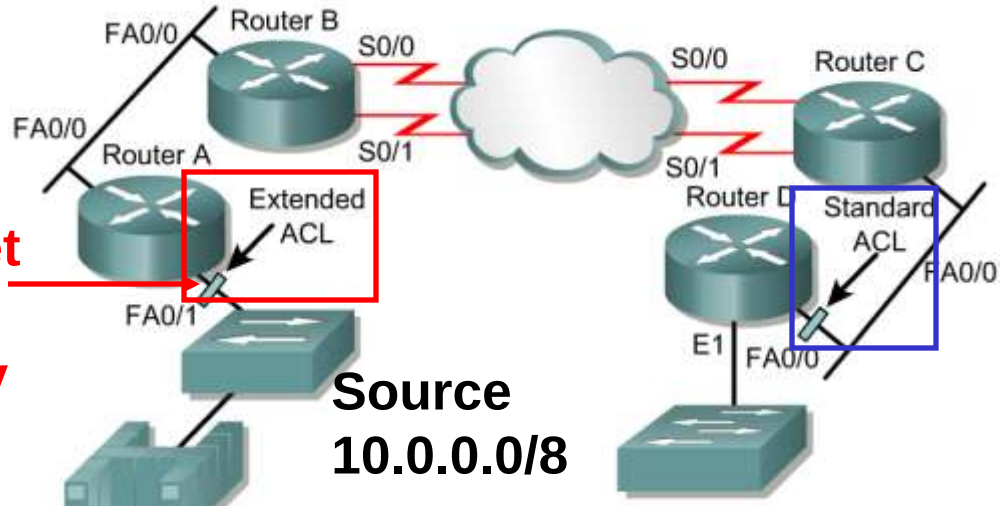
- If the ACLs are placed in the proper location, not only can traffic be filtered, but it can make the whole network more efficient.
- If traffic is going to be filtered, the ACL should be placed where it has the greatest impact on increasing efficiency.

Placing ACLs – Extended Example



- Policy is to deny telnet or FTP Router A LAN to Router D LAN.
- All other traffic must be permitted.
- Several approaches can accomplish this policy.
- The recommended approach uses an extended ACL specifying both source and destination addresses.

Placing ACLs – Extended Example



deny telnet
deny ftp
permit any

Source
10.0.0.0/8

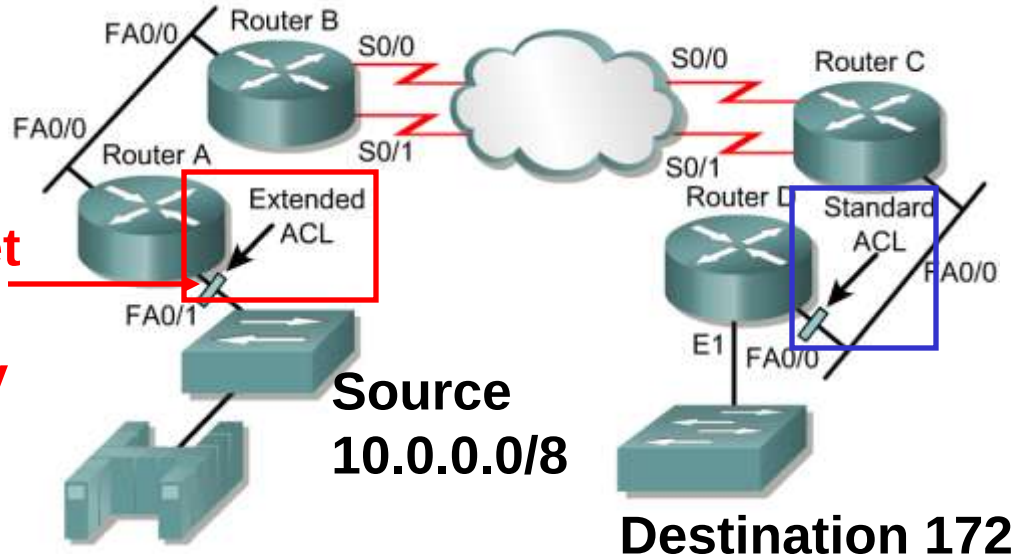
RouterA

Destination 172.16.0.0/16

```
interface fastethernet 0/1
    access-group 101 in
access-list 101 deny tcp any 172.16.0.0 0.0.255.255 eq telnet
access-list 101 deny tcp any 172.16.0.0 0.0.255.255 eq ftp
access-list 101 permit ip any any
```

- Place this extended ACL in Router A.
- Then, packets do not cross Router A's Ethernet, do not cross the serial interfaces of Routers B and C, and do not enter Router D.
- Traffic with different source and destination addresses will still be permitted.

Placing ACLs – Extended Example



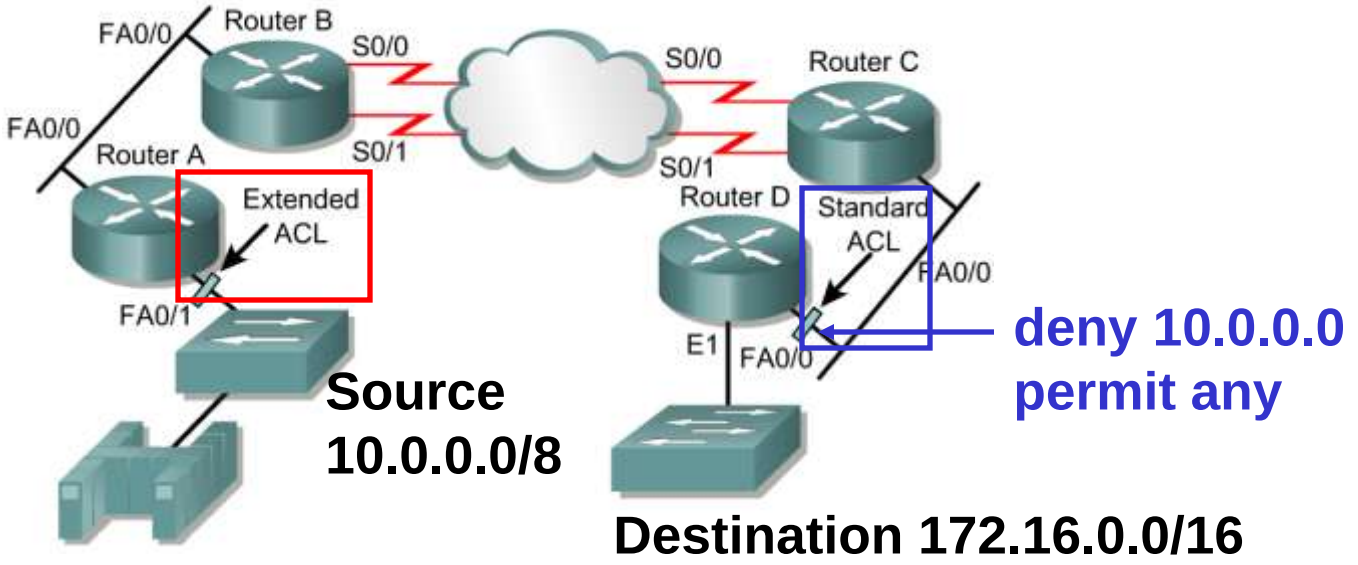
deny telnet
deny ftp
permit any

RouterA

```
interface fastethernet 0/1
    access-group 101 in
access-list 101 deny tcp any 172.16.0.0 0.0.255.255 eq telnet
access-list 101 deny tcp any 172.16.0.0 0.0.255.255 eq ftp
access-list 101 permit ip any any
```

- If the **`permit ip any any`** is not used, then no traffic is permitted.
- Be sure to **`permit ip`** and not just tcp or all udp traffic will be denied.

Placing ACLs – Standard Example

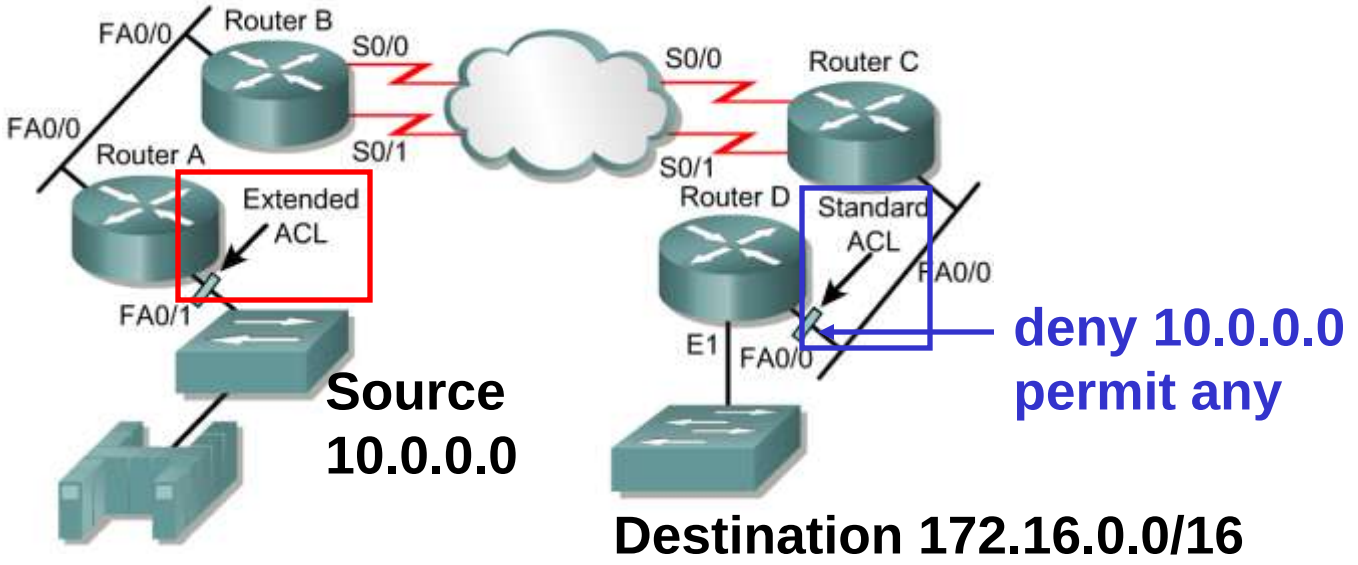


RouterD

```
interface fastethernet 0/0
    access-group 10 in
access-list 10 deny 10.0.0.0 0.255.255.255
access-list 10 permit any
```

- Standard ACLs do not specify destination addresses, so they should be placed as close to the destination as possible.
- If a **standard** ACL is put too close to the source, it will not only deny the intended traffic, but all other traffic to all other networks.

Placing ACLs – Standard Example

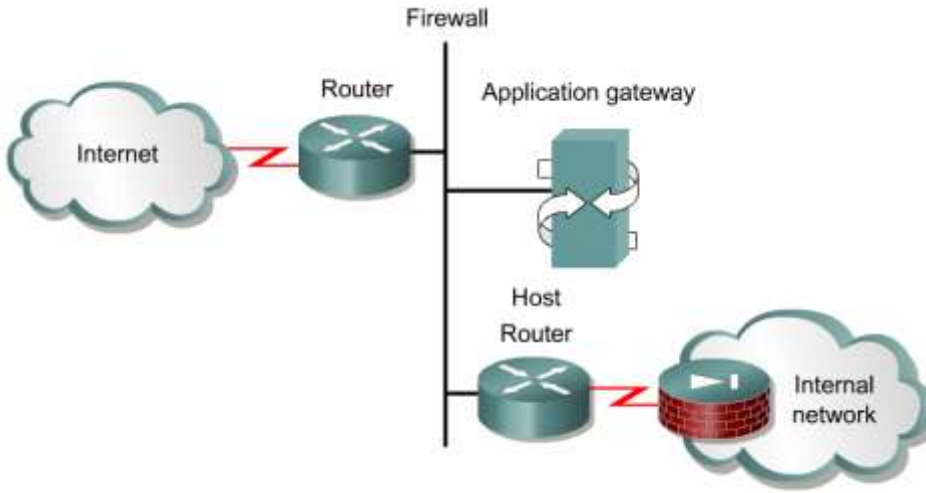


RouterD

```
interface fastethernet 0/0
    access-group 10 in
access-list 10 deny 10.0.0.0 0.255.255.255
access-list 10 permit any
```

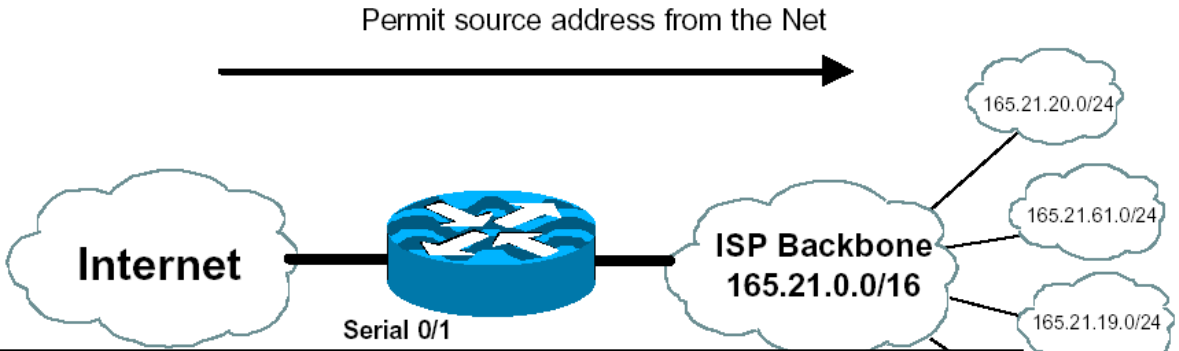
- Better to use extended access lists, and place them close to the source, as this traffic will travel all the way to RouterD before being denied.

Firewalls



- A firewall is an architectural structure that exists between the user and the outside world to protect the internal network from intruders.
- In most circumstances, intruders come from the global Internet and the thousands of remote networks that it interconnects.
- Typically, a network firewall consists of several different machines that work together to prevent unwanted and illegal access.
- ACLs should be used in firewall routers, which are often positioned between the internal network and an external network, such as the Internet.
- The firewall router provides a point of isolation so that the rest of the internal network structure is not affected.
- ACLs can be used on a router positioned between the two parts of the network to control traffic entering or exiting a specific part of the internal network.

Firewalls



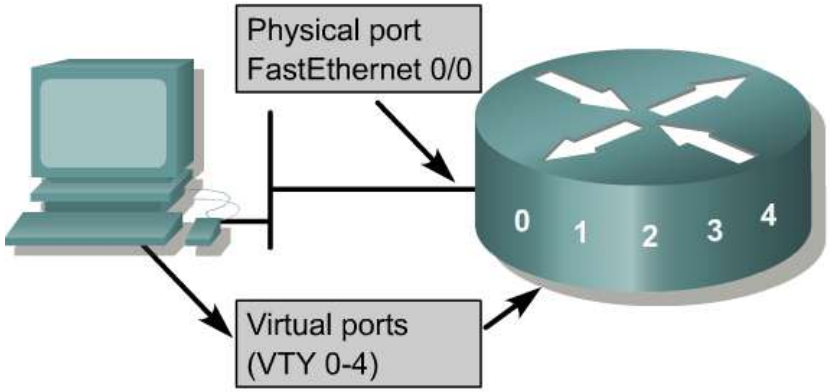

```
access-list 111 deny ip host 0.0.0.0 any log
access-list 111 deny ip 127.0.0.0 0.255.255.255 any log
access-list 111 deny ip 10.0.0.0 0.255.255.255 any log
access-list 111 deny ip 172.16.0.0 0.15.255.255 any log
access-list 111 deny ip 192.168.0.0 0.0.255.255 any log
access-list 111 deny ip 165.21.0.0 0.0.255.255 any log
access-list 111 permit ip any any
!
interface serial 1/0
 ip access-group 111 in
!
```

- ISPs use ACLs to deny RFC 1918 addresses into their networks as these are non-routable Internet addresses.
- IP packets coming into your network should never have a source addresses that belong to your network. (This should be applied on all network entrance routers.)
- There are several other simple access lists which should be added to network entrance routers.
- See Cisco IP Essentials White Paper for more information.

Restricting Virtual Terminal Access to a Router


```
Creating the standard list:

Rt1(config)#access-list 2 permit 172.16.1.0 0.0.0.255
Rt1(config)#access-list 2 permit 172.16.2.0 0.0.0.255
Rt1(config)#access-list 2 deny any

Applying the access list:

Rt1(config)#line vty 0 4
Rt1(config)#login
Rt1(config)#password secret
Rt1(config)#access-class 2 in
```

Rt1(config-line)#

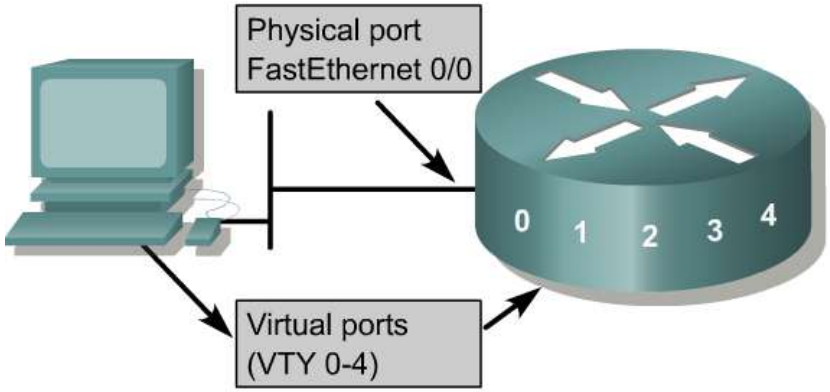


- The purpose of restricted vty access is increased network security.
- Access to vty is also accomplished using the Telnet protocol to make a nonphysical connection to the router.
- As a result, there is only one type of vty access list. Identical restrictions should be placed on all vty lines as it is not possible to control which line a user will connect on.

Restricting Virtual Terminal Access to a Router


```
Creating the standard list:

Rt1(config)#access-list 2 permit 172.16.1.0 0.0.0.255
Rt1(config)#access-list 2 permit 172.16.2.0 0.0.0.255
Rt1(config)#access-list 2 deny any

Applying the access list:

Rt1(config)#line vty 0 4
Rt1(config)#login
Rt1(config)#password secret
Rt1(config)#access-class 2 in
```

Rt1(config-line)#



- Standard and extended access lists apply to packets traveling through a router.
- **ACLs do not block packets that originate within the router.**
- An outbound Telnet extended access list does not prevent router initiated Telnet sessions, by default.

Ch. 11 – Access Control Lists

Cabrillo College

CCNA 2 version 3.0

Rick Graziani

Cabrillo College