TRS2251
Routing and Switching

# Access Control Lists (ACLs)

Lecture 9

Source: Rick Graziani
Cabrillo College

# Lecture 8

1. IP ACL Operation
2. Standard IPv4 ACLs
3. Extended IPv4 ACLSs
4. Contextual Unit: Debug with ACLs
5. Troubleshoot ACLs
6. Contextual Unit: IPv6 ACLs
7. Summary

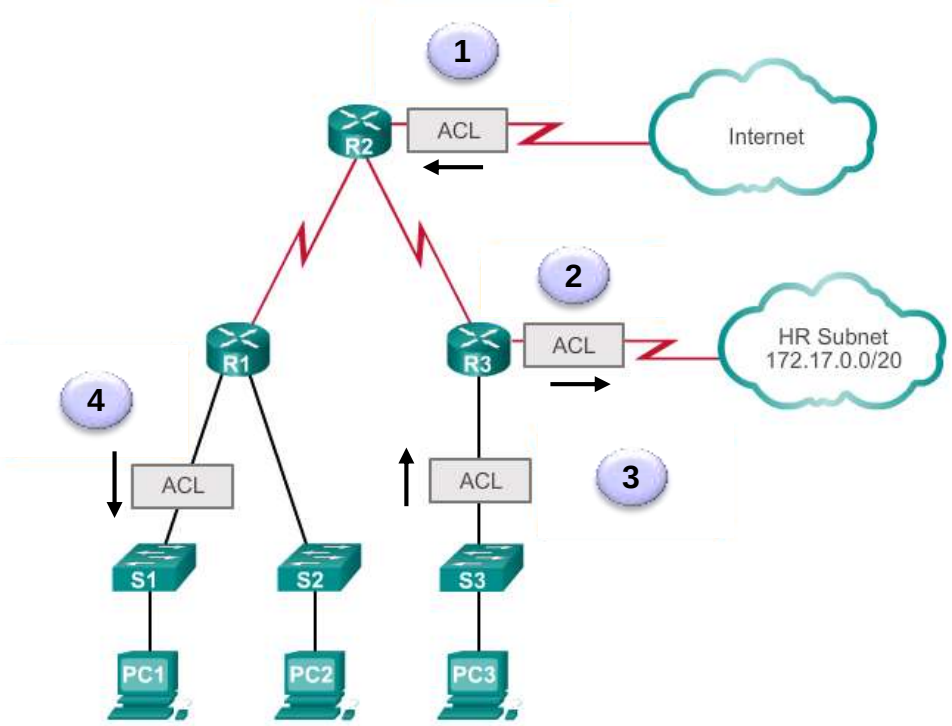
# Lecture 8: Objectives

- After completing this chapter, students will be able to:
  - Explain how ACLs are used to filter traffic.
  - Compare standard and extended IPv4 ACLs.
  - Explain how ACLs use wildcard masks.
  - Explain the guidelines for creating ACLs.
  - Explain the guidelines for placement of ACLs.
  - Configure standard IPv4 ACLs to filter traffic according to networking requirements.
  - Modify a standard IPv4 ACL using sequence numbers.
  - Configure a standard ACL to secure vty access.
  - Explain the structure of an extended access control entry (ACE).
  - Configure extended IPv4 ACLs to filter traffic according to networking requirements.
  - Configure an ACL to limit debug output.
  - Explain how a router processes packets when an ACL is applied.
  - Troubleshoot common ACL errors using CLI commands.
  - Compare IPv4 and IPv6 ACL creation.
  - Configure IPv6 ACLs to filter traffic according to networking requirements.

Access Control Lists (ACLs)

# Access Control Lists (ACLs)

- By default, a router does not filter traffic.
- When an ACL is applied to an interface:
  - performs the additional task of evaluating all network packets
  - determines if the packet can be forwarded in or out of the interface.



# What are ACLs?

- A router acts as a packet filter when it forwards or denies packets according to filtering rules.
  - Packet filtering controls access to a network by:
    - analyzing the incoming and outgoing packets
    - passing or dropping them based on given criteria

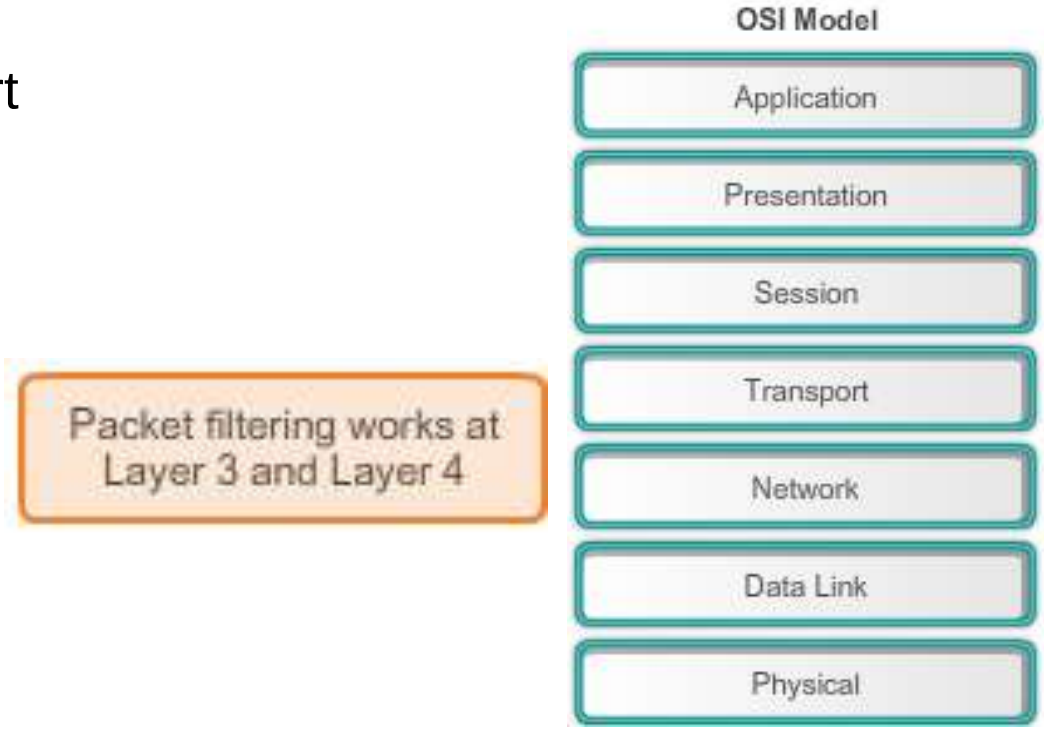
- An ACL is a sequential list of of **permit** or **deny** statements, known as **access control entries (ACEs)**.
  - ACEs are also commonly called ACL statements.

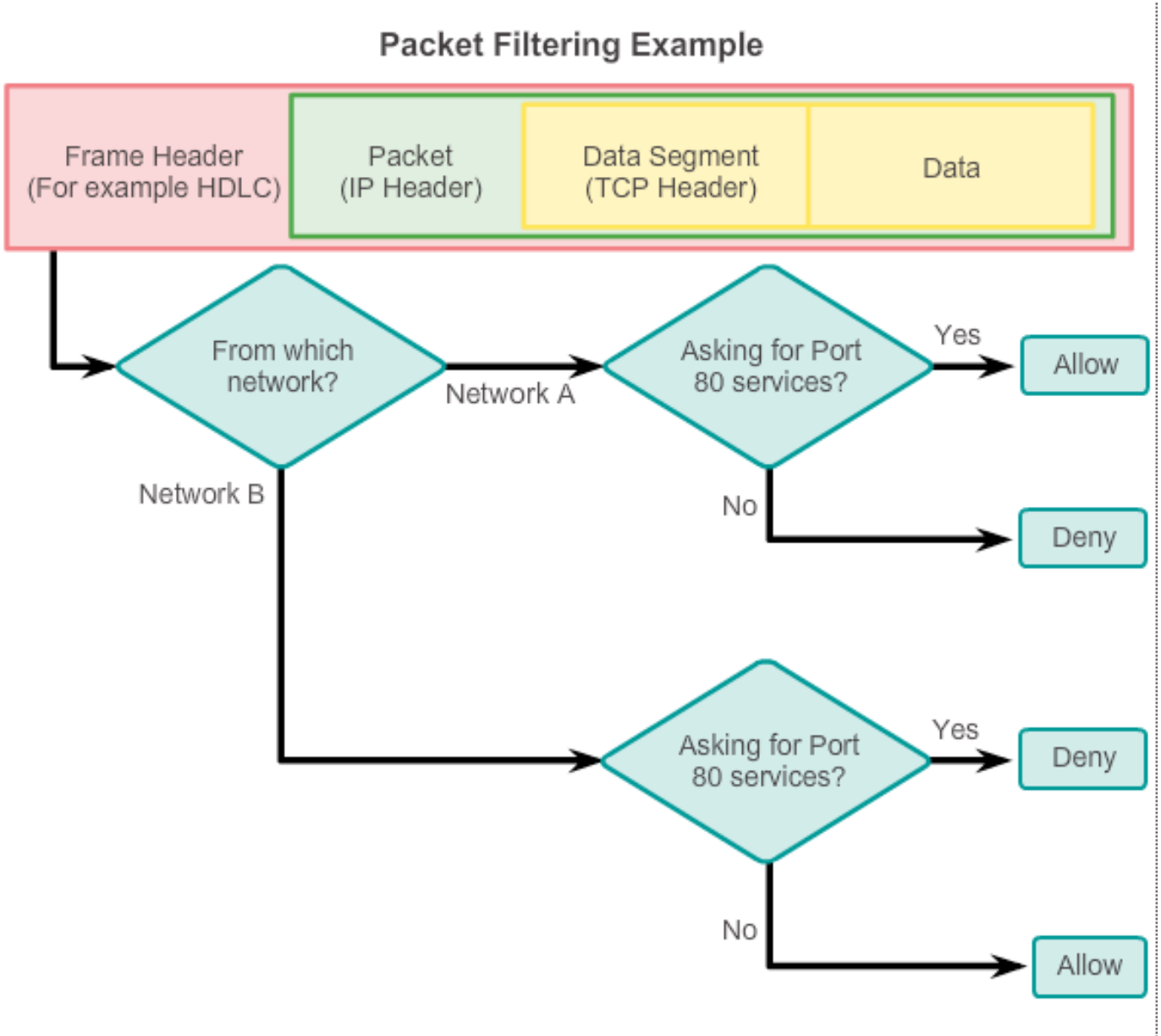
# Packet Filtering

- ACLs control whether a router permits or denies packets based on criteria in the header that identifies the:
  - Source IP address
  - Destination IP address
  - IP protocols (ICMP, TCP, UDP, EIGRP, …)
  - TCP/UDP source port
  - TCP/UDP destination port

# Packet Filtering

- ACLs control whether a router permits or denies packets based on criteria in the header that identifies the:
  - Source IP address
  - Destination IP address
  - IP protocols (ICMP, TCP, UDP, EIGRP, …)
  - TCP/UDP source port
  - TCP/UDP destination port




Packet Filtering Example
Frame Header
(For example HDLC)
Packet
(IP Header)
Data Segment
(TCP Header)
Data
From which network?
Network A
Asking for Port 80 services?
Yes
Allow
No
Deny
Network B
Asking for Port 80 services?
Yes
Deny
No
Allow


An inbound ACL filters packets coming into a specific interface and before they are routed to the outbound interface.

An outbound ACL filters packets after being routed, regardless of the inbound interface.

# Types of ACLs

- **Standard ACLs**
  - Can **permit** or **deny** traffic for Source IP addresses … only!

```
access-list 10 permit 192.168.30.0 0.0.0.255
```

- **Extended ACLs**
  - Can **permit** or **deny** traffic for:
    - Protocol type IP (IP, ICMP, EIGRP, OSPF, TCP, UDP, …)
    - Source IP address
    - Source TCP or UDP ports
    - Destination IP address
    - Destination TCP or UDP ports

```
access-list 103 permit tcp 192.168.30.0 0.0.0.255 any eq 80
```

# Numbered and Named ACLs

**Numbered ACL:**

Assign a number based on protocol to be filtered.

- (1 to 99) and (1300 and 1999): Standard IP ACL
- (100 to 199) and (2000 to 2699): Extended IP ACL

**Named ACL:**

Assign a name to identify the ACL.

- Names can contain alphanumeric characters.
- It is suggested that the name be written in CAPITAL LETTERS.
- Names cannot contain spaces or punctuation.
- Entries can be added or deleted within the ACL.

# Wildcard Mask

- Standard and Extended ACLs both use wildcard masks.
  - Wildcard masks and subnet masks differ in the way they match binary 1s and 0s.

- Wildcard masks use the following rules to match binary 1s and 0s:
  - Wildcard mask bit 0 - Match the corresponding bit value in the address
  - Wildcard mask bit 1 - Ignore the corresponding bit value in the address

# Calculating Wildcard Masks #1

- Calculating wildcard masks can be difficult, but you can do it easily by subtracting the subnet mask from 255.255.255.255.

- For example, assume you wanted to permit access to all users in the 192.168.3.0 /24
  - Subtract the subnet mask (255.255.255.0) from the subnet mask 255.255.255.255.

```
  255.255.255.255
-255.255.255.000
  000.000.000.255
```

```
access-list  1  permit  192.168.3.0 0.0.0.255
```

# Calculating Wildcard Masks #2

- Assume you wanted to permit access to all users in the 192.168.3.32 /28
  - Subtract the subnet mask (255.255.255.240) from the subnet mask 255.255.255.255.

```
 255.255.255.255
-255.255.255.240
 000.000.000.015
```

```
access-list  1  permit  192.168.3.32 0.0.0.15
```

# Host keyword

```
access-list  1  permit  192.168.1.1 0.0.0.0
```

| | Decimal | Binary |
|---|---|---|
| IP Address | 192.168.1.1 | 11000000.10101000.00000001.00000001 |
| Wildcard Mask | 0.0.0.0 | 00000000.00000000.00000000.00000000 |
| Result | 192.168.1.1 | 11000000.10101000.00000001.00000001 |

- The **host** keyword can be used to substitute for the 0.0.0.0 wildcard mask.
  - This mask states that all IPv4 address bits must match or only one host is matched.

```
access-list 1 permit host 192.168.1.1
```

**Note**: The **host** keyword can also be used in IPv6 ACLs.

# Any Keyword

```
access-list  1  permit  0.0.0.0 255.255.255.255
```

| | Decimal | Binary |
|---|---|---|
| IP Address | 0.0.0.0 | 00000000.00000000.00000000.00000000 |
| Wildcard Mask | 255.255.255.255 | 11111111.11111111.11111111.11111111 |
| Result | 0.0.0.0 | 00000000.00000000.00000000.00000000 |

- The **any** keyword substitutes for the 255.255.255.255 wildcard mask.
  - This mask says to ignore the entire IPv4 address or to accept any addresses.

```
access-list  1  permit  any
```

**Note**: The **any** keyword can also be used in IPv6 ACLs.

# Any and Host Keywords

**Example 1:**

```
R1(config)#access-list 1 permit 0.0.0.0 255.255.255.255
R1(config)#access-list 1 permit any
```

**Example 2:**

```
R1(config)#access-list 1 permit 192.168.10.10 0.0.0.0
R1(config)#access-list 1 permit host 192.168.10.10
```

# Placement of ACLs

# Any Traffic filtering on a Router


One list per interface, per direction, and per protocol

With two interfaces and two protocols running, this router could have a total of 8 separate ACLs applied.

**The three Ps for using ACLs**

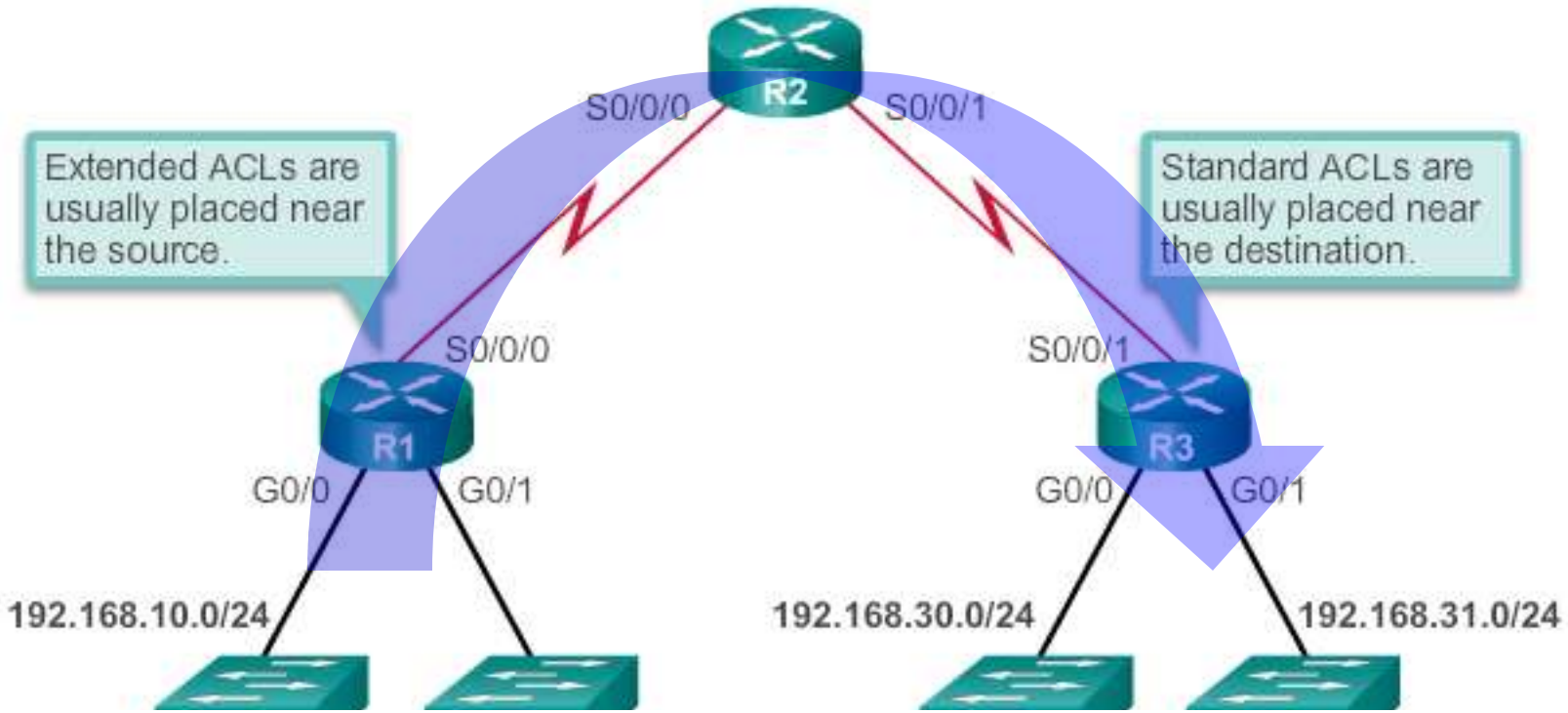
You can only have one ACL per protocol, per interface, and per direction:

- One ACL per protocol (e.g., IPv4 or IPv6)
- One ACL per direction (i.e., IN or OUT)
- One ACL per interface (e.g., FastEthernet0/0)

# Any Best Practices

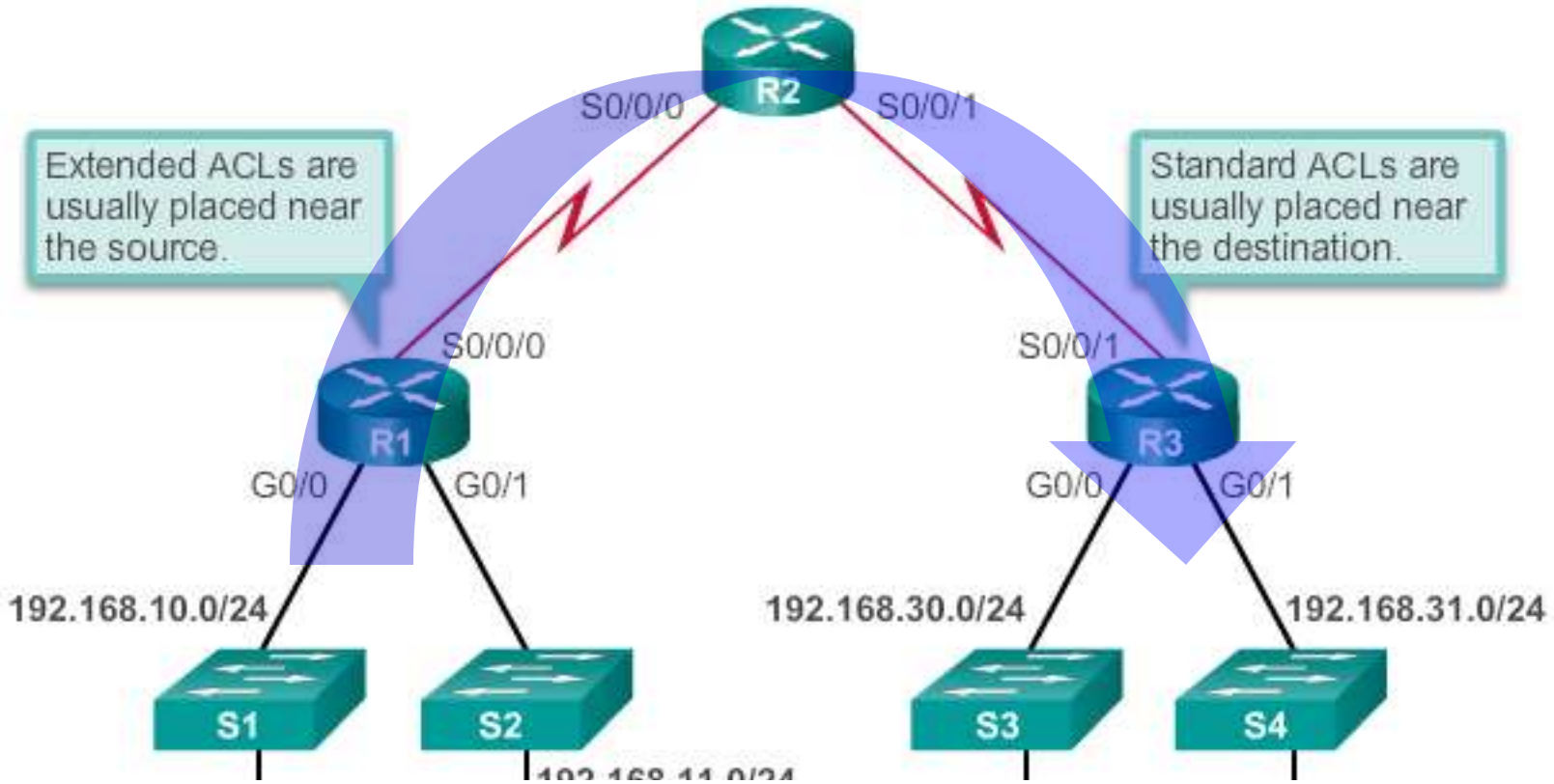
| Guideline | Benefit |
|---|---|
| Base your ACLs on the security policy of the organization. | This will ensure you implement organizational security guidelines. |
| Prepare a description of what you want your ACLs to do. | This will help you avoid inadvertently creating potential access problems. |
| Use a text editor to create, edit, and save ACLs. | This will help you create a library of reusable ACLs. |
| Test your ACLs on a development network before implementing them on a production network. | This will help you avoid costly errors. |

# ACL Placement



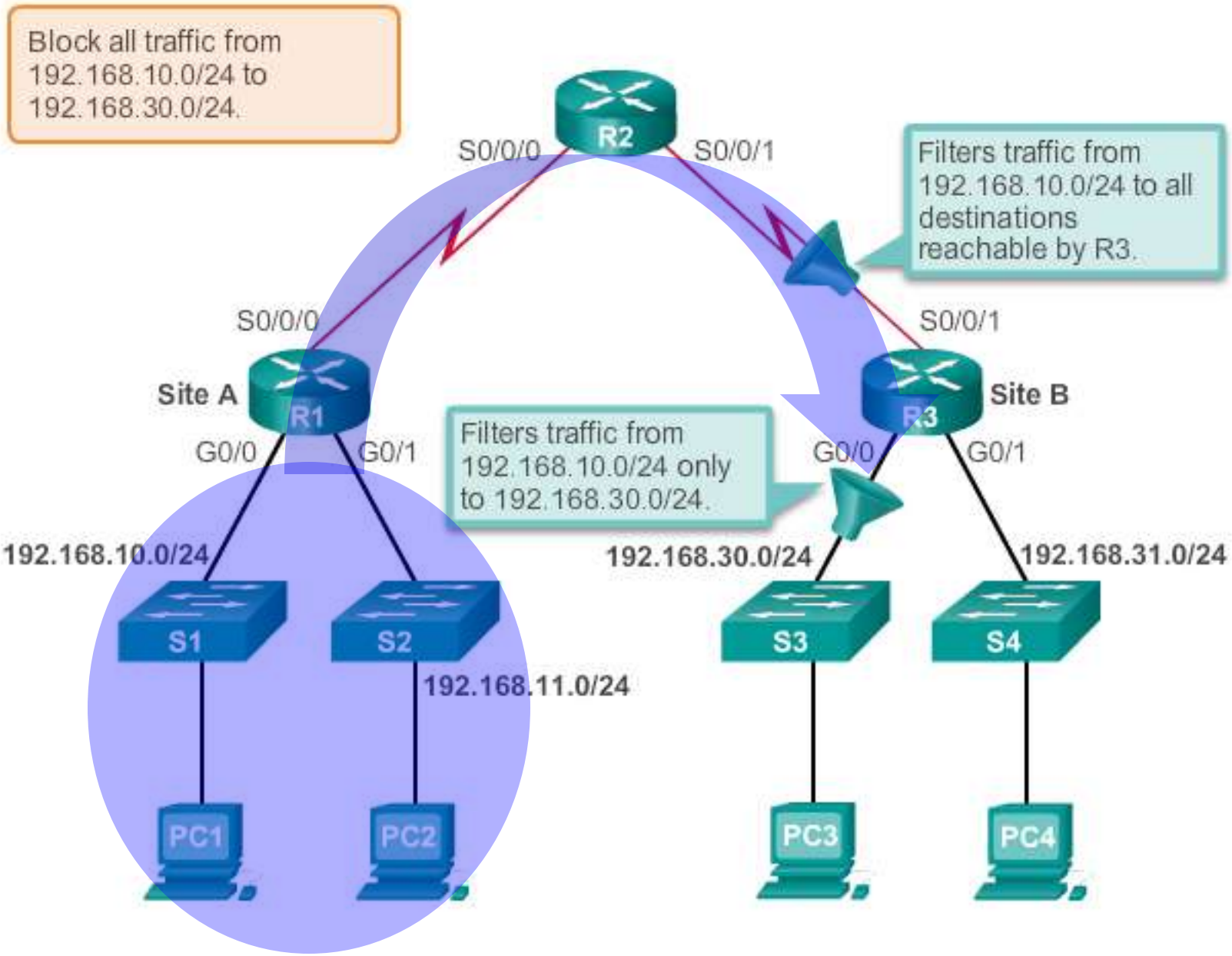

- **Extended ACLs** - This way, undesirable traffic is denied close to the source network without crossing the network infrastructure.
- **Standard ACLs** - Because standard ACLs do not specify destination addresses, place them as close to the destination as possible.
- Placing a standard ACL at the source of the traffic will effectively prevent that traffic from reaching any other networks through the interface where the ACL is applied.

# ACL Placement

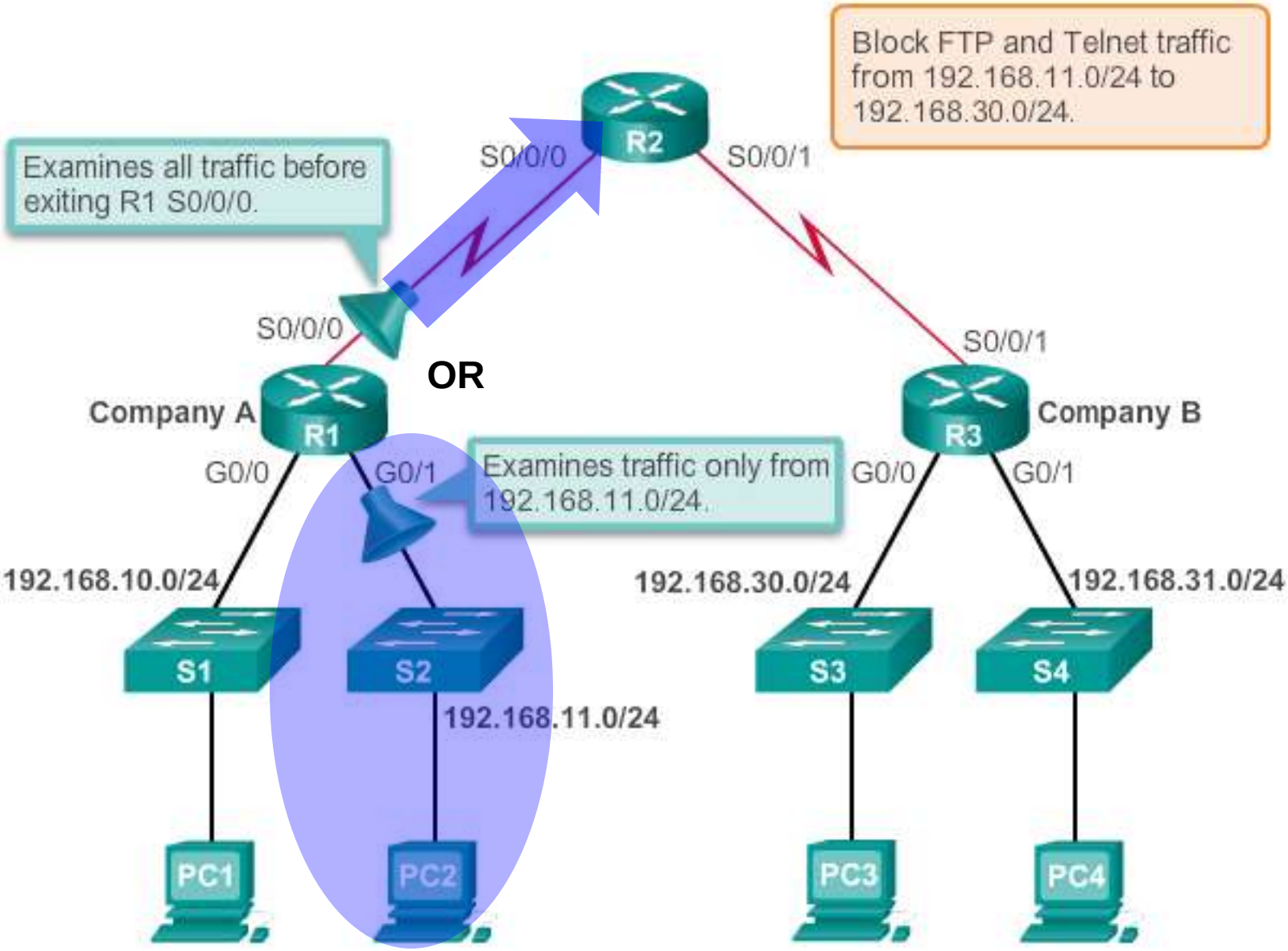



- Placement of the ACL and therefore the type of ACL used may also depend on:
  - **The extent of the network administrator's control**
  - **Bandwidth of the networks involved**
  - **Ease of configuration**

# Standard ACL Placement

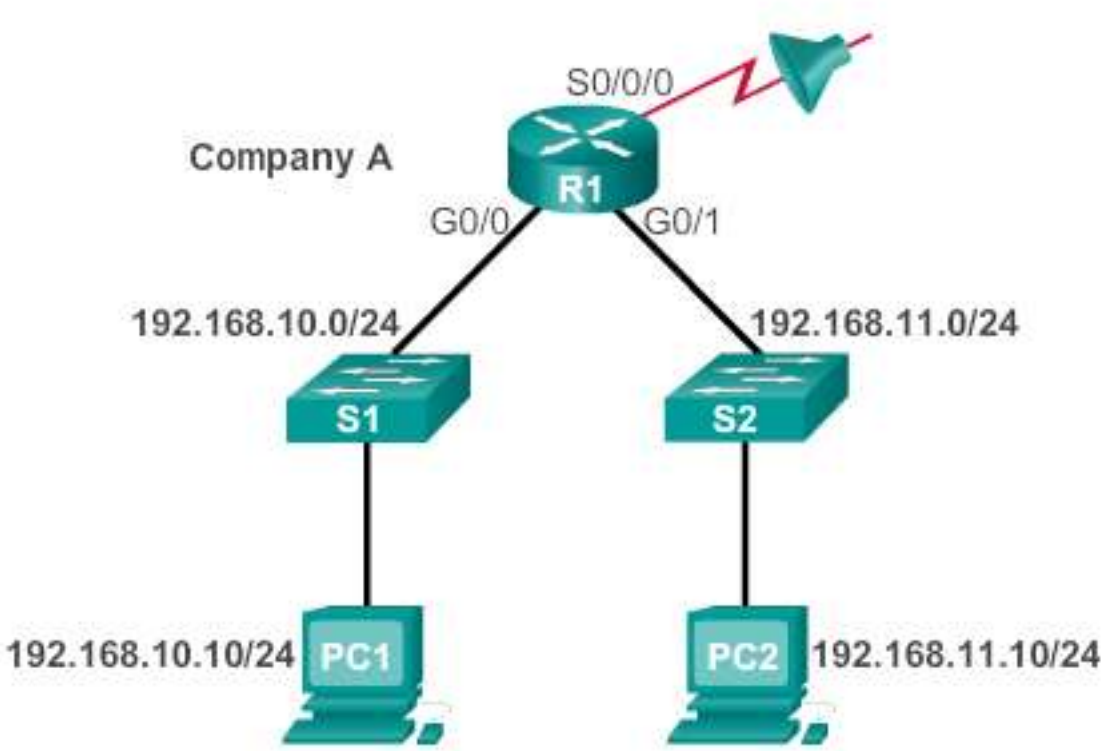


# Extended ACL Placement



# Configuring Standard IPv4 ACLs Numbered and Named

# How are ACLs Created? In Two Steps!




**1. Create an ACL definition**.

- Enter global configuration mode.
- Define statements of what to filter.

**2. Apply the ACL to an interface**.

- Enter interface configuration mode.
- Identify the ACL and the direction to filter.

# 1. Create a Standard ACL

| RTR(config)# | **access-list** *ACL#* **{permit\|deny}** **{** *test-conditions* **}** |
|---|---|
| | **access-list 5 permit 172.34.54.34 0.0.0.0** |

- *ACL-#* is a unique identifier.
  - The # range identifies the type of ACL.

- **permit** | **deny** are terms to specify how the packets which meet the condition will be handled.
  - **permit**: Implies the packet will not be filtered.
  - **deny**: Implies the packet will be filtered.
  - **remark**: Allows you to enter a description of the ACL

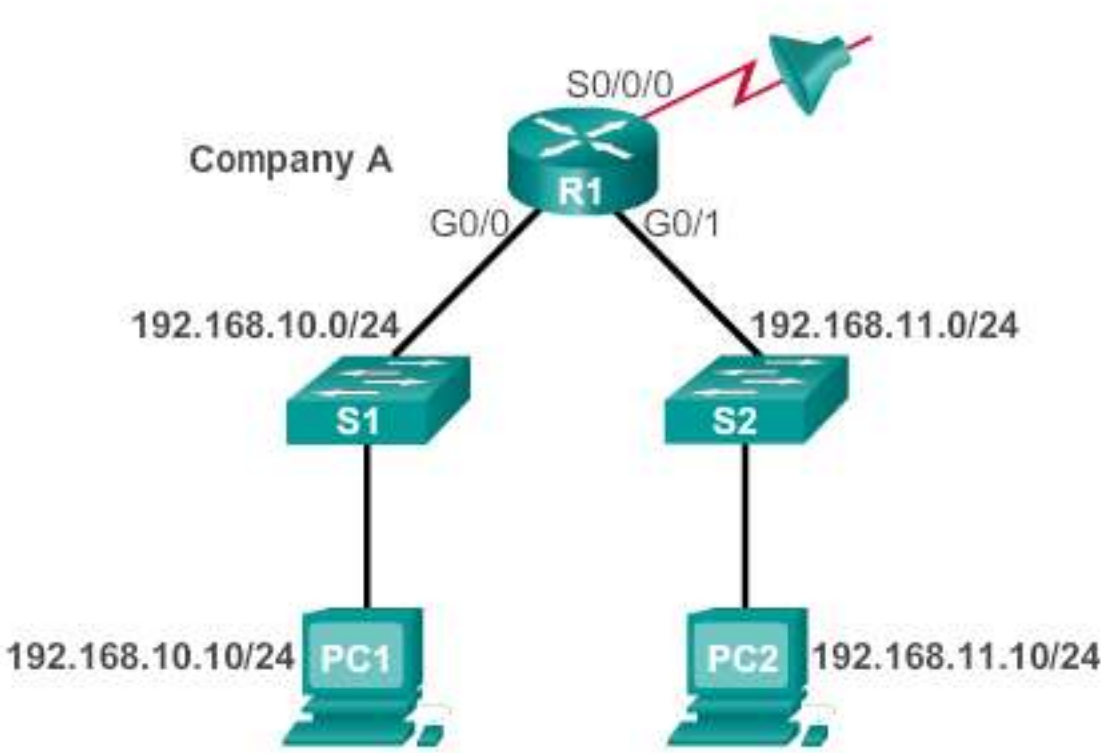
# 2. Apply the ACL to an interface

| RTR(config-if)# **{*protocol*} access-group *list-#* {in\|out}** |
|---|
| **ip access-group 5 out** |

- **in** | **out** identifies if the ACL is for incoming or outgoing traffic.
  - **in** means that packets are filtered as they enter the interface, before the routing decision.
  - **out** means that packets are filtered as they leave the interface, after the routing decision.

- "**out**" is the default.
  - Outbound ACLs are generally more efficient, and are preferred.
  - Inbound ACLs must check every packet.

By default, there is an **implied deny** at the end of all ACLs for traffic that was not matched to a configured entry.

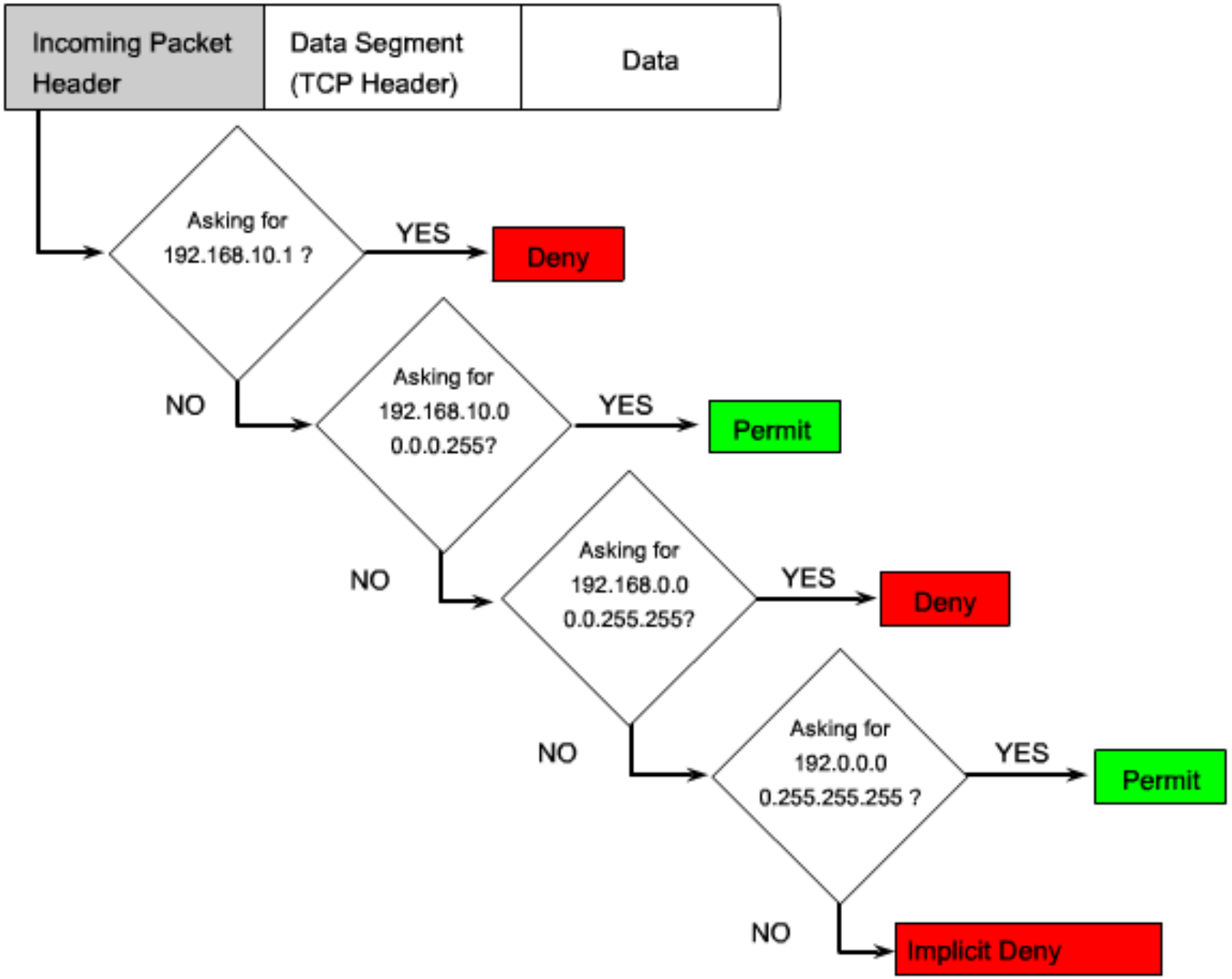



```
R1(config)# access-list 1 permit ip 192.168.10.0 0.0.0.255
```

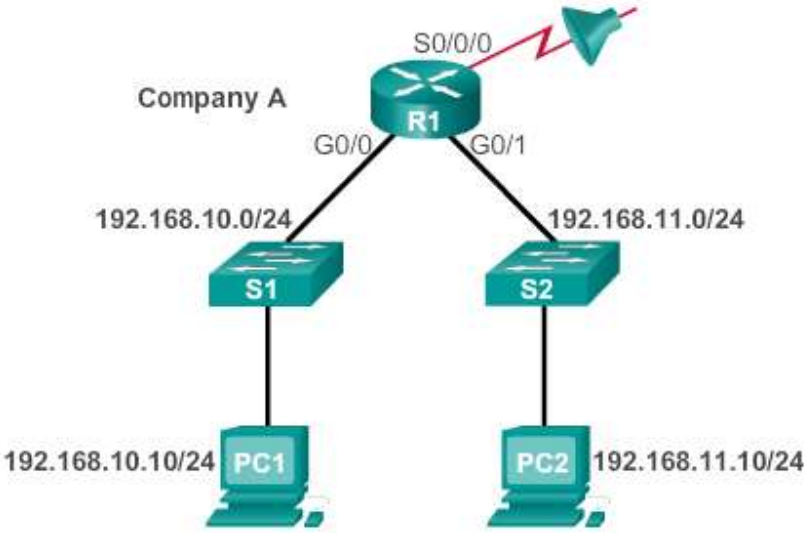
**SAME AS**

```
R1(config)# access-list 2 permit ip 192.168.10.0 0.0.0.255
R1(config)# access-list 2 deny any
```

# Standard ACL Logic

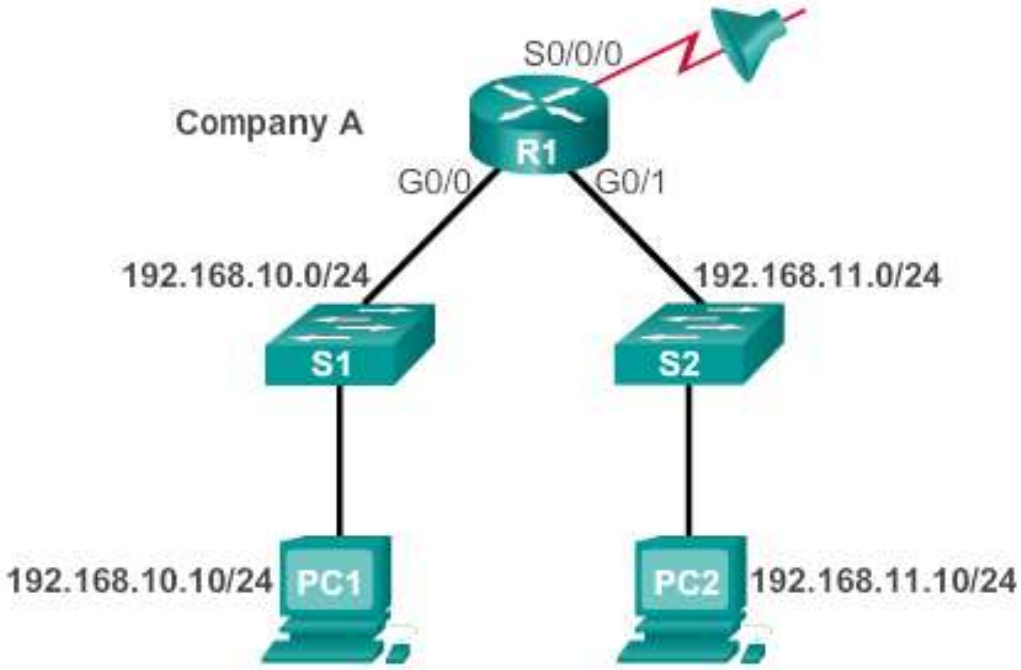


# Removing an ACL



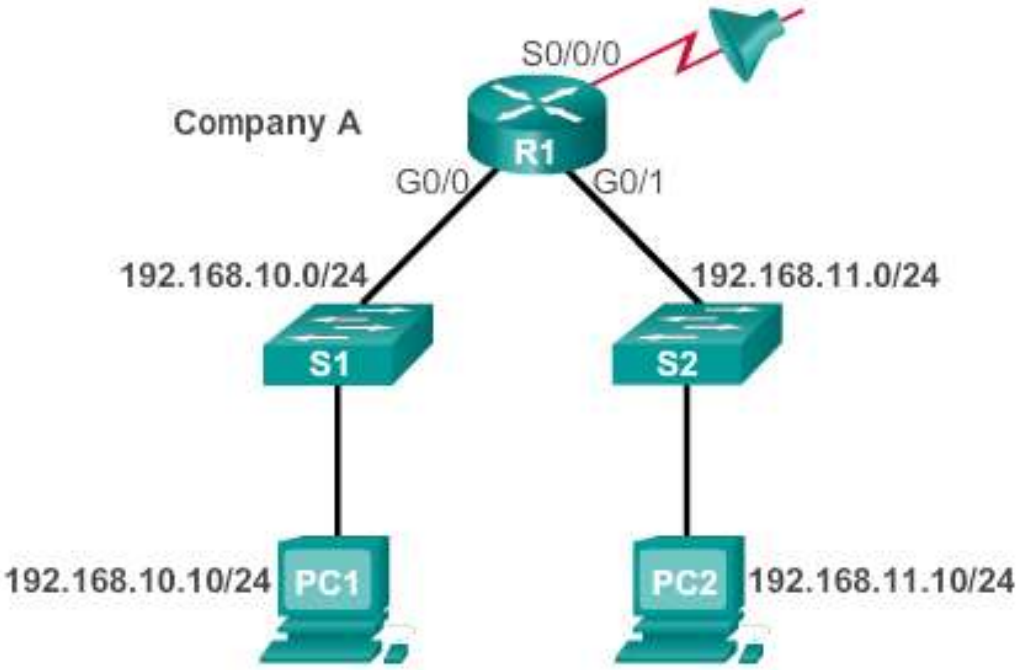

```
R1(config)# access-list 10 permit 192.168.10.0 0.0.0.255
R1(config)# exit
R1# show access-lists
Standard IP access list 10
    10 permit 192.168.10.0, wildcard bits 0.0.0.255
R1# conf t
Enter configuration commands, one per line.  End with CNTL/Z.
R1(config)# no access-list 10
R1(config)# exit
R1 #show access-lists
R1#
```

# Comments - remark




```
R1(config)# access-list 10 remark Permit hosts from the
192.168.10.0 LAN
R1(config)# access-list 10 permit 192.168.10.0 0.0.0.255
R1(config)# exit
R1# show running-config | include access-list 10
access-list 10 remark Permit hosts from the 192.168.10.0 LAN
access-list 10 permit 192.168.10.0 0.0.0.255
R1#
```

# Internal Logic
# Order matters

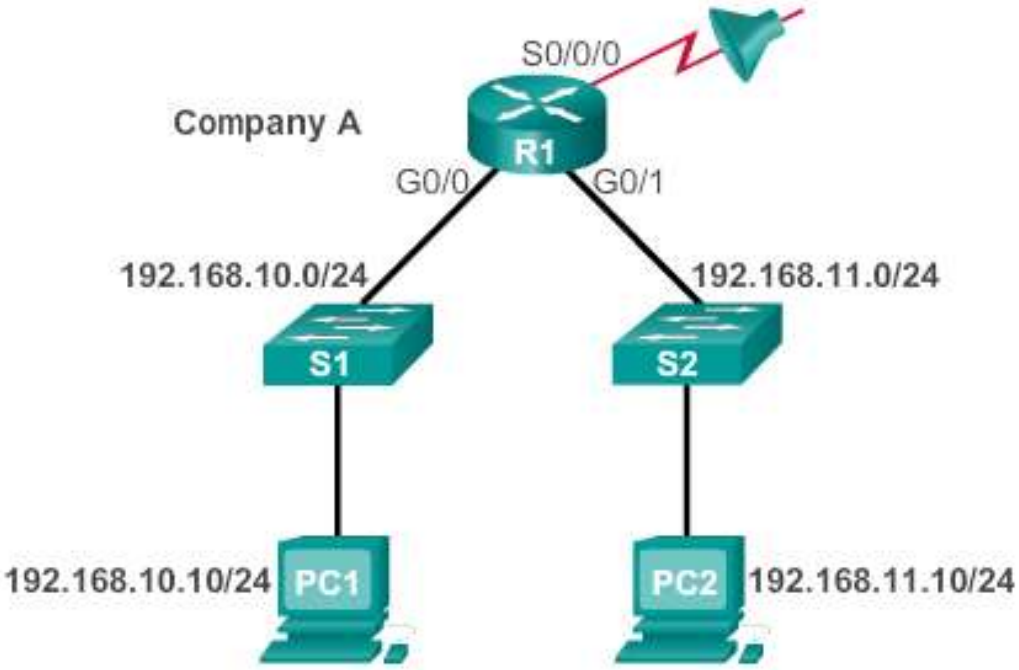



**Conflict with Statements**

**ACL 3: Host statement conflicts with previous range statement**

```
R1(config)# access-list 3 deny 192.168.10.0 0.0.0.255
R1(config)# access-list 3 permit host 192.168.10.10
% Access rule can't be configured at higher sequence num as
it is part of the existing rule at sequence num 10
R1(config)#
```

# Internal Logic
# Order matters

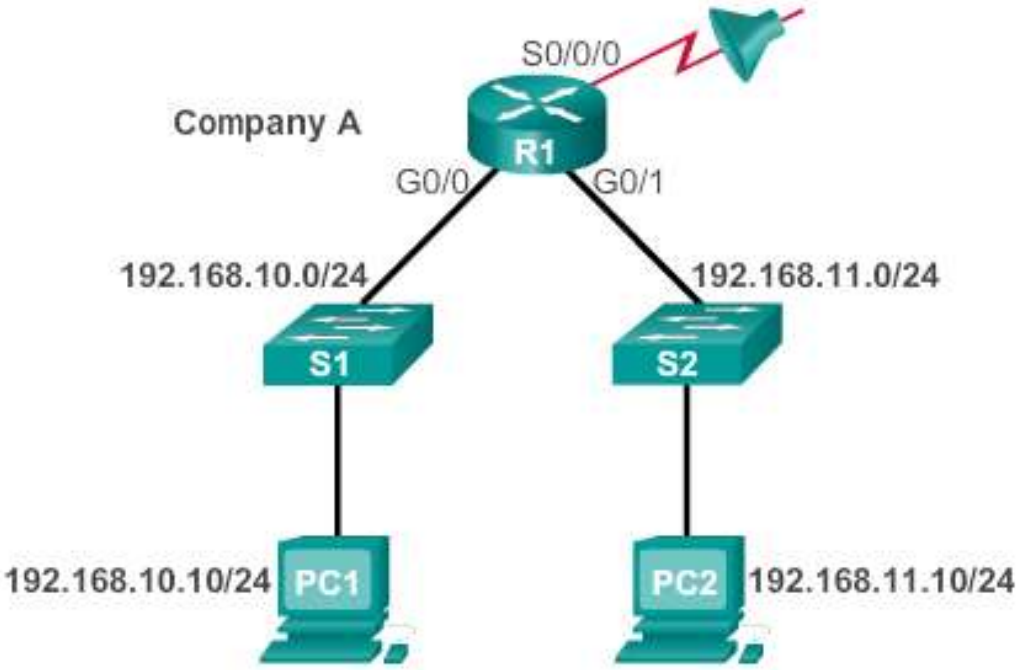



**Host Statement Entered Before Range**

**ACL 4: Host statement can always be configured before range statements**

```
R1(config)# access-list 4 permit host 192.168.10.10
R1(config)# access-list 4 deny 192.168.10.0 0.0.0.255
R1(config)#
```

# Internal Logic Order matters




**Host Configured Before Range with no Conflict**

**ACL 5: Host statement can be configured after range statement if there is no conflict**

```
R1(config)# access-list 5 deny 192.168.10.0 0.0.0.255
R1(config)# access-list 5 permit host 192.168.11.10
R1(config)#
```

# Applying Standard ACLs to Interfaces

Step 1: Configure the ACL statements

```
R1(config)# access-list 1 permit 192.168.10.0 0.0.0.255
```

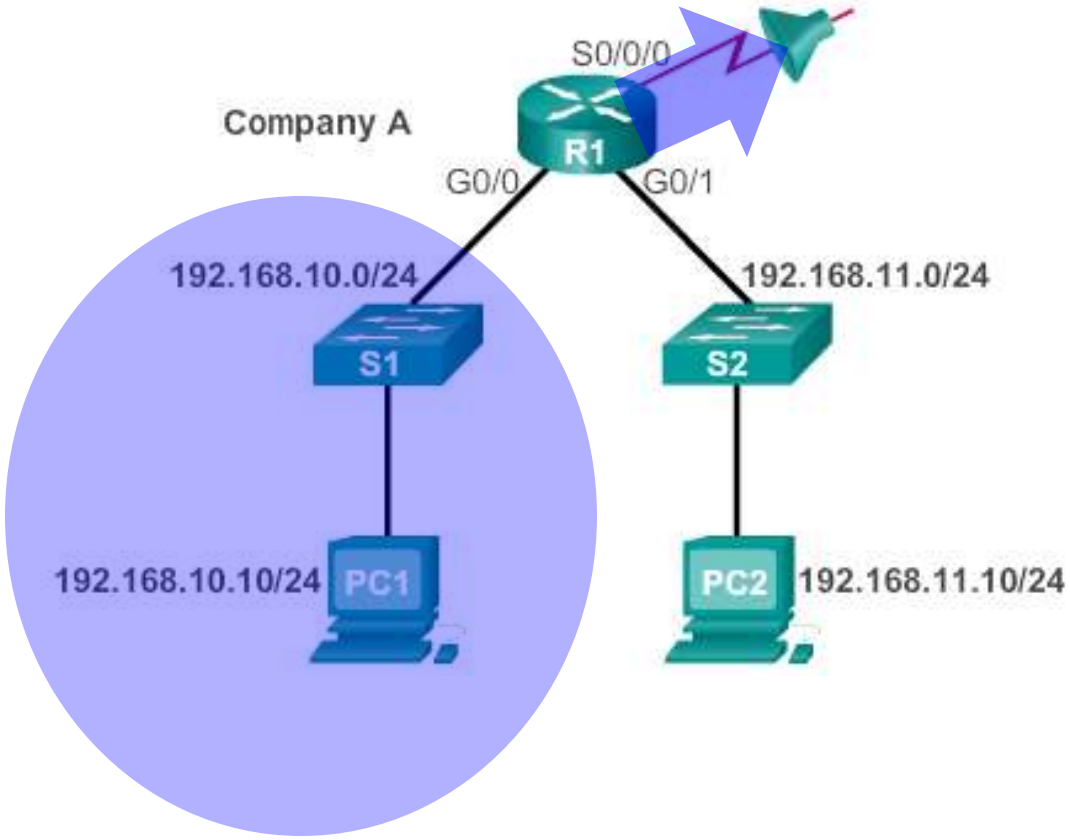
Step 2: Select the interface to apply the ACL

```
R1(config)# interface serial 0/0/0
```

Step 3: Apply the ACL to the interface using the **ip access-group** command

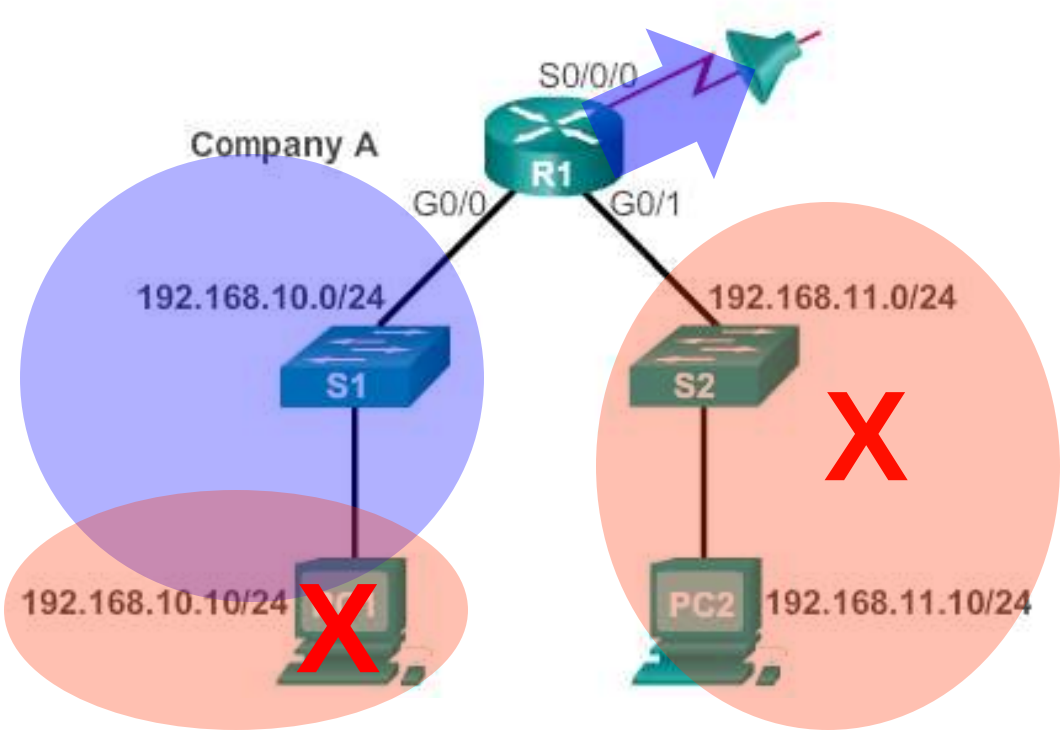
```
R1(config-if)# ip access-group 1 out
```

# Permit a Specific Subnet



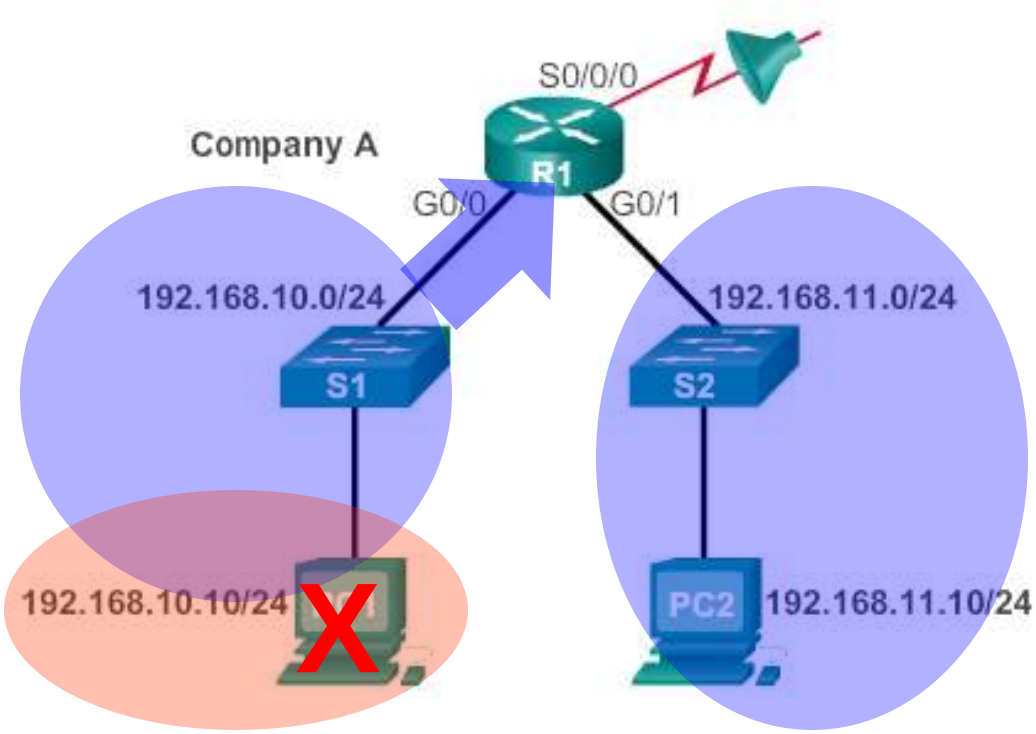

```
R1(config)# access-list 1 permit 192.168.10.0 0.0.0.255
R1(config)# interface s0/0/0
R1(config-if)# ip access-group 1 out
```

# Deny a Specific Host and Permit a Specific Subnet




```
R1(config)# access-list 1 deny host 192.168.10.10
R1(config)# access-list 1 permit 192.168.10.0 0.0.0.255
R1(config)# interface s0/0/0
R1(config-if)# ip access-group 1 out
```

# Deny a Specific Host




```
R1(config)# access-list 1 deny host 192.168.10.10
R1(config)# access-list 1 permit any
R1(config)# interface g0/0
R1(config-if)# ip access-group 1 in
```

# Named ACL

```
Router(config)# ip access-list [standard | extended] name
```

Alphanumeric name string must be unique and cannot begin with a number.

```
Router(config-std-nacl)# [permit | deny | remark] {source
[source-wildcard]} [log]
```

```
Router(config-if)# ip access-group name [in | out]
```

Activates the named IP ACL on an inteface.

# Named ACLs

| RTR(config)# **ip access-list {standard\|extended} {** *NAME* **}** |
| --- |
| **ip access-list extended TELNET-FILTER** |

- Named ACLs allow standard and extended IP ACLs to be identified with an alphanumeric string (name) instead of the current numeric representation.
  - Name cannot start with a number.
- Named ACLs help identify the function of the ACL.
- The actual names used must be unique across all named access lists of all protocols and types on an individual router.
  - Names can be duplicated on different routers.
- ACLs of different types cannot have the same name.
  - For example, it is illegal to specify a standard ACL named BOB and an extended ACL with the same name.

# Named ACLs Syntax

| RTR(config)# **ip access-list {standard\|extended} {** *NAME* **}** |
|---|
| **ip access-list extended TELNET-FILTER** |

- You create the named ACL in global configuration mode.
- Notice that the **access-list** command has changed to:
  - **ip access-list**
- You then enter named ACL configuration mode.
  - The sub config mode prompt varies between standard and extended ACLs.

```
R1(config)# ip access-list standard STANDARD-ACL
R1(config-std-nacl)# exit
R1(config)# ip access-list extended EXT-ACL
R1(config-ext-nacl)# exit
```

# Named ACLs Syntax

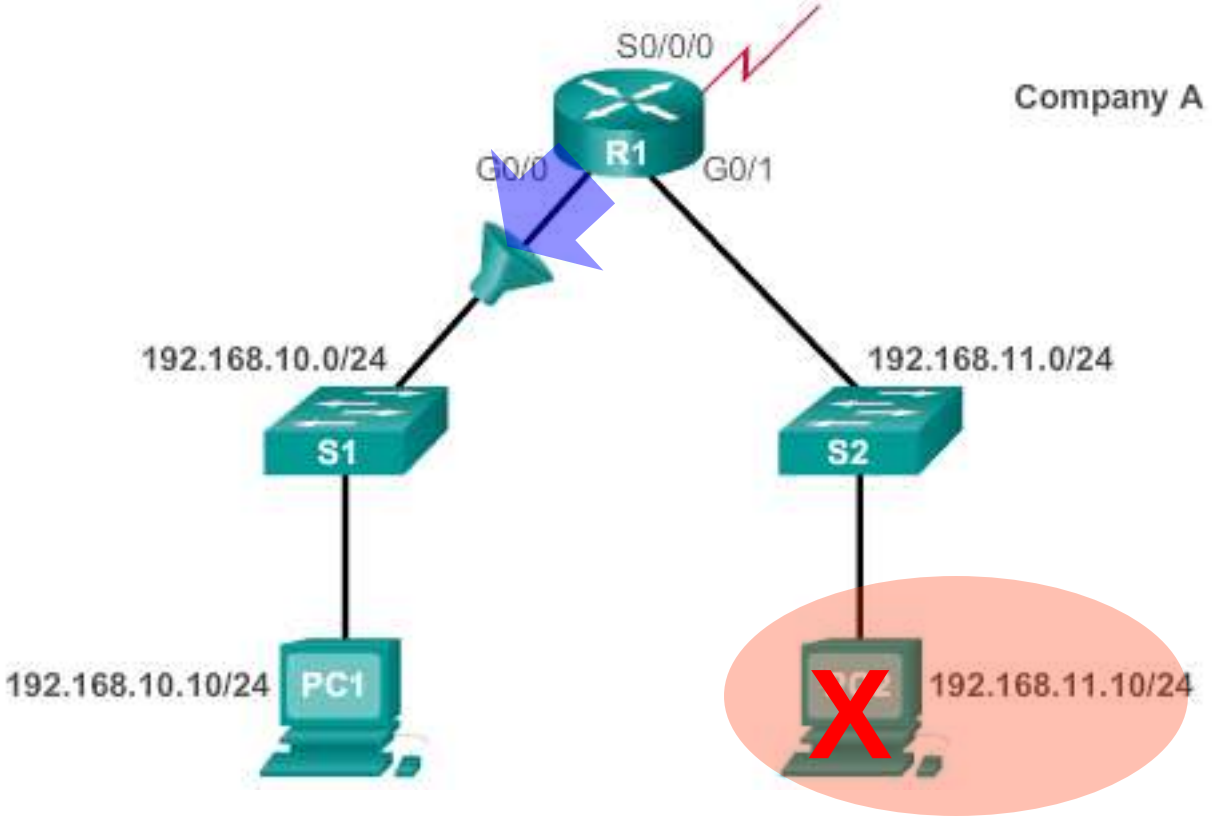
- In ACL configuration mode, specify one or more conditions permitted or denied.
  - This determines whether the packet is passed or dropped.

```
RTR(config {std- | ext-}nacl)# deny {source [source wildcard] | any}
```

```
RTR(config {std- | ext-}nacl)# permit {source [source wildcard] | any}
```

```
RTR(config {std- | ext-}nacl)# remark [comment]
```

# Named ACL Example




```
R1(config)# ip access-list standard NO_ACCESS
R1(config-std-nacl)# deny host 192.168.11.10
R1(config-std-nacl)# permit any
R1(config-std-nacl)# exit
R1(config)# interface g0/0
R1(config-if)# ip access-group NO_ACCESS out
```

# Commenting ACLs

## Example 1 – Commenting a numbered ACL

```
R1(config)# access-list 1 remark Do not allow Guest workstation through
R1(config)# access-list 1 deny host 192.168.10.10
R1(config)# access-list 1 remark Allow devices from all other 192.168.x.x subnets
R1(config)# access-list 1 permit 192.168.0.0 0.0.255.255
R1(config)# interface s0/0/0
R1(config-if)# ip access-group 1 out
R1(config-if)#
```

## Example 2 – Commenting a named ACL

```
R1(config)# ip access-list standard NO_ACCESS
R1(config-std-nacl)# remark Do not allow access from Lab workstation
R1(config-std-nacl)# deny host 192.168.11.10
R1(config-std-nacl)# remark Allow access from all other networks
R1(config-std-nacl)# permit any
R1(config-std-nacl)# interface G0/0
R1(config-if)# ip access-group NO_ACCESS out
R1(config-if)#
```

# Editing Numbered ACLs Using Sequence Numbers

**Configuration**

```
R1(config)# access-list 1 deny host 192.168.10.99
R1(config)# access-list 1 permit 192.168.0.0 0.0.255.255
```

**Step 1**

```
R1# show access-lists 1
Standard IP access list 1
    10 deny   192.168.10.99
    20 permit 192.168.0.0, wildcard bits 0.0.255.255
R1#
```

**Step 2**

```
R1# conf t
R1(config)# ip access-list standard 1
R1(config-std-nacl)# no 10
R1(config-std-nacl)# 10 deny host 192.168.10.10
R1(config-std-nacl)# end
R1#
```

**Step 3**

```
R1# show access-lists
Standard IP access list 1
    10 deny   192.168.10.10
    20 permit 192.168.0.0, wildcard bits 0.0.255.255
R1#
```

# Editing Named ACLs – Adding a Line

```
R1# show access-lists
Standard IP access list NO_ACCESS
    10 deny   192.168.11.10
    20 permit 192.168.11.0, wildcard bits 0.0.0.255
R1# conf t
Enter configuration commands, one per line.  End with CNTL/Z.
R1(config)# ip access-list standard NO_ACCESS
R1(config-std-nacl)# 15 deny host 192.168.11.11
R1(config-std-nacl)# end
R1# show access-lists
Standard IP access list NO_ACCESS
    10 deny   192.168.11.10
    15 deny   192.168.11.11
    20 permit 192.168.11.0, wildcard bits 0.0.0.255
R1#
```

**Note**: The **no** *sequence-number* named-ACL command is used to delete individual statements.

# Verifying ACLs

```
R1# show ip interface s0/0/0
Serial0/0/0 is up, line protocol is up
  Internet address is 10.1.1.1/30
 <output omitted>
  Outgoing access list is 1
  Inbound  access list is not set
 <output omitted>

R1# show ip interface g0/0
GigabitEthernet0/1 is up, line protocol is up
  Internet address is 192.168.10.1/24
 <output omitted>
  Outgoing access list is NO_ACCESS
  Inbound  access list is not set
 <output omitted>
```

# Verifying ACLs

```
R1# show access-lists
Standard IP access list 1
    10 deny   192.168.10.10
    20 permit 192.168.0.0, wildcard bits 0.0.255.255
Standard IP access list NO_ACCESS
    15 deny   192.168.11.11
    10 deny   192.168.11.10
    20 permit 192.168.11.0, wildcard bits 0.0.0.255
R1#
```

# Viewing ACL Statistics

```
R1# show access-lists
Standard IP access list 1
    10 deny   192.168.10.10 (4 match(es))
    20 permit 192.168.0.0, wildcard bits 0.0.255.255
Standard IP access list NO_ACCESS
    15 deny   192.168.11.11
    10 deny   192.168.11.10 (4 match(es))
    20 permit 192.168.11.0, wildcard bits 0.0.0.255
R1#
```

**Output after pinging PC3 from PC1.**

```
R1# show access-lists
Standard IP access list 1
    10 deny   192.168.10.10 (8 match(es))
    20 permit 192.168.0.0, wildcard bits 0.0.255.255
Standard IP access list NO_ACCESS
    15 deny   192.168.11.11
    10 deny   192.168.11.10 (4 match(es))
    20 permit 192.168.11.0, wildcard bits 0.0.0.255
R1#
```

Matches have been incremented.

# Clearing ACL Statistics

```
R1#show access-lists
Standard IP access list 1
    10 deny   192.168.10.10 (8 match(es))
    20 permit 192.168.0.0, wildcard bits 0.0.255.255
Standard IP access list NO_ACCESS
    15 deny   192.168.11.11
    10 deny   192.168.11.10 (4 match(es))
    20 permit 192.168.11.0, wildcard bits 0.0.0.255
R1#
R1#clear access-list counters 1
R1#
R1#show access-lists
Standard IP access list 1
    10 deny   192.168.10.10
    20 permit 192.168.0.0, wildcard bits 0.0.255.255
Standard IP access list NO_ACCESS
    15 deny   192.168.11.11
    10 deny   192.168.11.10 (4 match(es))
    20 permit 192.168.11.0, wildcard bits 0.0.0.255
```

Matches have been cleared.

# Standard ACL Sequence Numbers and Internal Logic

```
R1(config)#access-list 1 deny 192.168.10.0 0.0.0.255
R1(config)#access-list 1 deny 192.168.20.0 0.0.0.255
R1(config)#access-list 1 deny 192.168.30.0 0.0.0.255
R1(config)#access-list 1 permit 10.0.0.1
R1(config)#access-list 1 permit 10.0.0.2
R1(config)#access-list 1 permit 10.0.0.3
R1(config)#access-list 1 permit 10.0.0.4
R1(config)#access-list 1 permit 10.0.0.5
R1(config)#end
R1#show running-config | include access-list 1
access-list 1 permit 10.0.0.2
access-list 1 permit 10.0.0.3
access-list 1 permit 10.0.0.1
access-list 1 permit 10.0.0.4
access-list 1 permit 10.0.0.5
access-list 1 deny   192.168.10.0 0.0.0.255
access-list 1 deny   192.168.20.0 0.0.0.255
access-list 1 deny   192.168.30.0 0.0.0.255
R1#
```

Range (network) statements

Host statements (no conflicts

Host statements (no conflicts

Range (network) statements

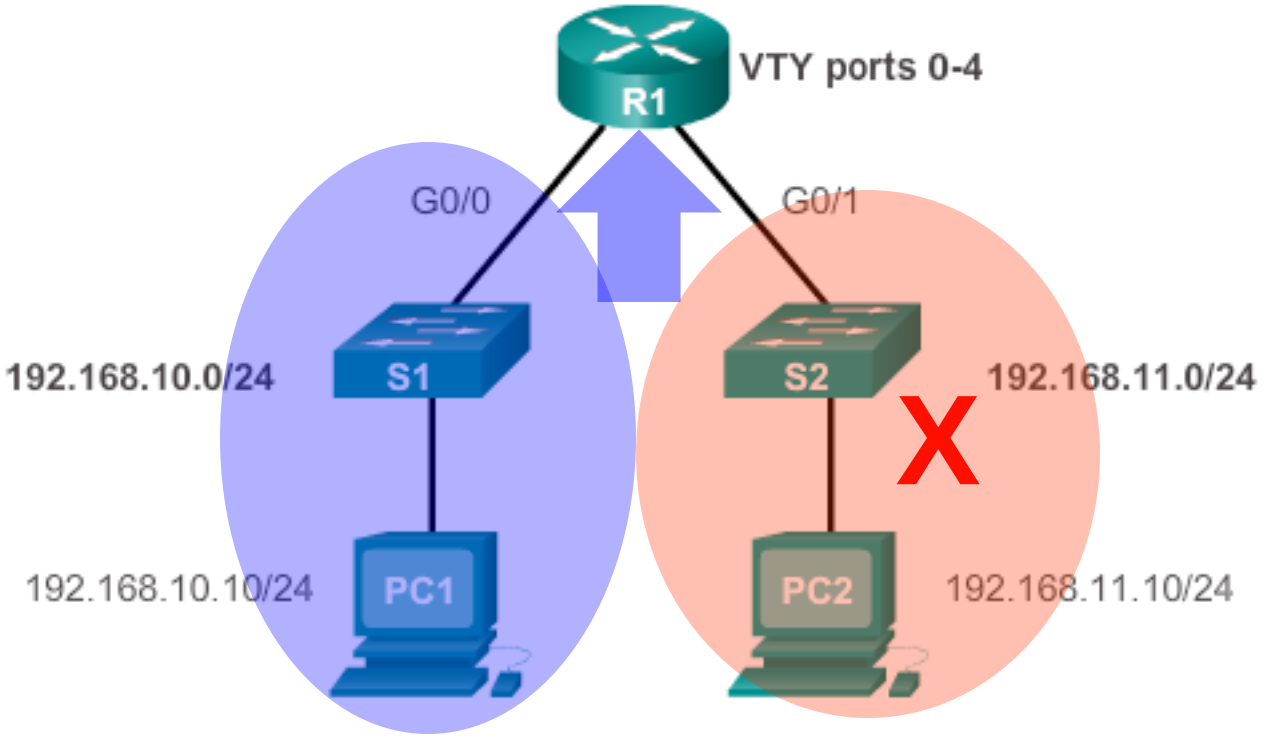
# Standard ACL Sequence Numbers After Reload

```
R1#show access-lists 1
Standard IP access list 1
    50 permit 10.0.0.2
    60 permit 10.0.0.3
    40 permit 10.0.0.1
    70 permit 10.0.0.4
    80 permit 10.0.0.5
    10 deny   192.168.10.0, wildcard bits 0.0.0.255
    20 deny   192.168.20.0, wildcard bits 0.0.0.255
    30 deny   192.168.30.0, wildcard bits 0.0.0.255
R1#copy running-config startup-config
R1#reload
R1#show access-lists 1
Standard IP access list 1
    10 permit 10.0.0.2
    20 permit 10.0.0.3
    30 permit 10.0.0.1
    40 permit 10.0.0.4
    50 permit 10.0.0.5
    60 deny   192.168.10.0, wildcard bits 0.0.0.255
    70 deny   192.168.20.0, wildcard bits 0.0.0.255
    80 deny   192.168.30.0, wildcard bits 0.0.0.255
R1#
```

**Host statements are listed first, in an order to be efficiently processed by the IOS keeping the original sequence numbers assigned by IOS**

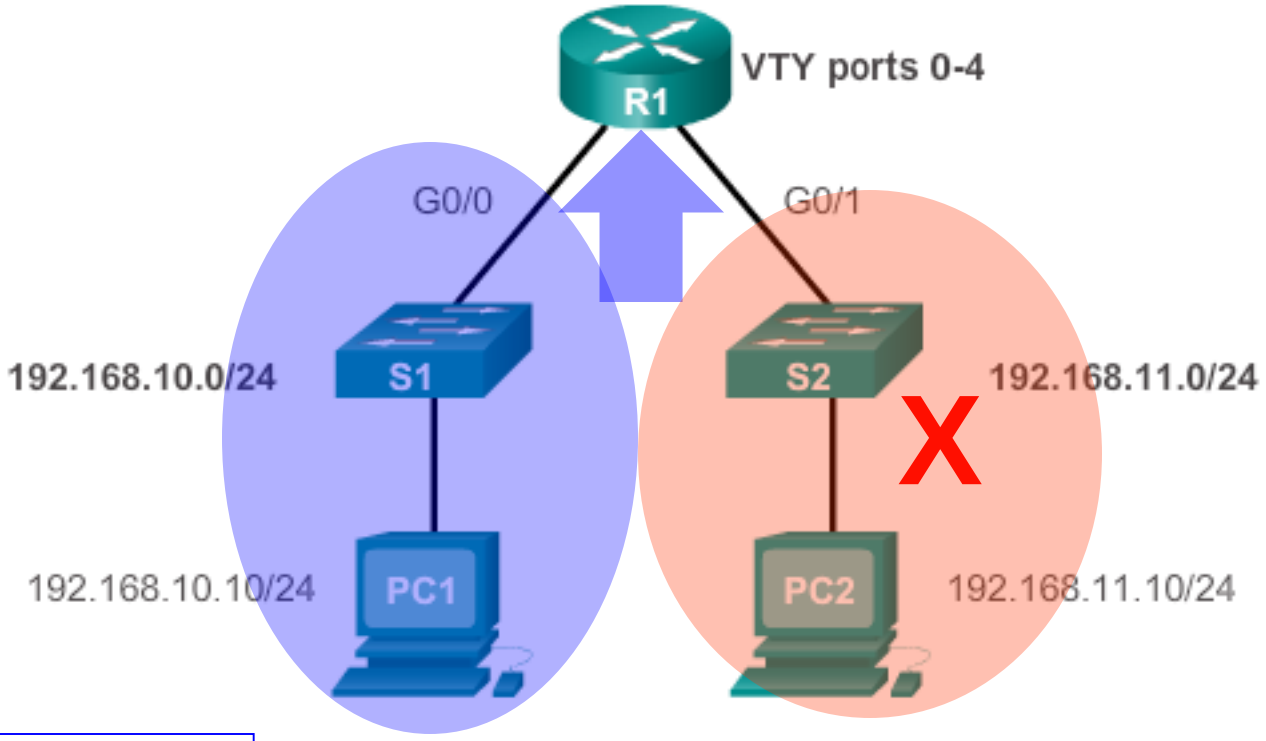
**Range statements are listed after host statements, in the order they were entered, also with original sequence numbers**

# Securing VTY Ports with Standard IPv4 ACLs




```
R1(config)# line vty 0 4
R1(config-line)# login local
R1(config-line)# transport input ssh
R1(config-line)# access-class 21 in
R1(config-line)# exit
R1(config)# access-list 21 permit 192.168.10.0 0.0.0.255
R1(config)# access-list 21 deny any
```

```
R1#show access-lists
Standard IP access list 21
    10 permit 192.168.10.0, wildcard bits 0.0.0.255 (2 matches)
    20 deny   any (1 match)
R1#
```




```
PC1>ssh 192.168.10.1

Login as: admin
Password: *****
R1>
```

```
PC2>ssh 192.168.11.1
ssh connect to host
192.168.11.1 port 22:
Connection refused

PC2>
```

# Configuring Extended IPv4 ACLs Numbered and Named


**Extended ACLs can filter on:**

- Source address
- Destination address
- Protocol
- Port numbers

## Using Port Numbers

```
access-list 114 permit tcp 192.168.20.0 0.0.0.255 any eq 23
access-list 114 permit tcp 192.168.20.0 0.0.0.255 any eq 21
access-list 114 permit tcp 192.168.20.0 0.0.0.255 any eq 20
```

## Using Keywords

```
access-list 114 permit tcp 192.168.20.0 0.0.0.255 any eq telnet
access-list 114 permit tcp 192.168.20.0 0.0.0.255 any eq ftp
access-list 114 permit tcp 192.168.20.0 0.0.0.255 any eq ftp-data
```

```
R1(config)# access-list 101 permit tcp any any eq ?
  <0-65535>     Port number
  bgp           Border Gateway Protocol (179)
  chargen       Character generator (19)
  cmd           Remote commands (rcmd, 514)
  daytime       Daytime (13)
  discard       Discard (9)
  domain        Domain Name Service (53)
  drip          Dynamic Routing Information Protocol (3949)
  echo          Echo (7)
  exec          Exec (rsh, 512)
  finger        Finger (79)
  ftp           File Transfer Protocol (21)
  ftp-data      FTP data connections (20)
  gopher        Gopher (70)
  hostname      NIC hostname server (101)
  ident         Ident Protocol (113)
  irc           Internet Relay Chat (194)
  klogin        Kerberos login (543)
  kshell        Kerberos shell (544)
  login         Login (rlogin, 513)
  lpd           Printer service (515)
  nntp          Network News Transport Protocol (119)
  pim-auto-rp   PIM Auto-RP (496)
  pop2          Post Office Protocol v2 (109)
  pop3          Post Office Protocol v3 (110)
```

# Extended ACLs

- Extended ACLs are used more often than standard ACLs because they provide a greater degree of control. Extended ACLs provide more precise traffic-filtering control.
  - Also referred to as “increased granular control”.

- All extended ACLs filter on Source IP address AND **Destination IP address**.

- But what make them really special is that they can also filter based on:
  - Upper layer protocols (e.g., IP, TCP, UDP, ICMP, EIGRP, …)
  - Source port
  - Destination port

# Extended ACLs Syntax

Extended ACLs also filter on **Protocol** and **Destination** address.

- All extended ACLs follow this basic syntax.

The choice of **Protocol** adds various other options.

| access-list | list-# | permit<br>deny<br>remark | Protocol | Source | | Destination | |
|---|---|---|---|---|---|---|---|
| | | | | IP | Wildcard | IP | Wildcard |
| | 100-199<br><br>2000 to 2699 | | IP<br>TCP<br>UDP<br>ICMP<br>EIGRP<br>OSPF | any<br>host | | any<br>host | |

These options change depending which Protocol is selected.

# Port Names versus Port Number

```
access-list 114 permit tcp 192.168.20.0 0.0.0.255 any eq 23
access-list 114 permit tcp 192.168.20.0 0.0.0.255 any eq 21
access-list 114 permit tcp 192.168.20.0 0.0.0.255 any eq 20
```

```
access-list 114 permit tcp 192.168.20.0 0.0.0.255 any eq telnet
access-list 114 permit tcp 192.168.20.0 0.0.0.255 any eq ftp
access-list 114 permit tcp 192.168.20.0 0.0.0.255 any eq ftp-data
```

Port/protocol after destination address refers to the destination port

- **Note**:
  - Not all protocols have a port name assigned
  - Port numbers always work.
  - E.g., SSH and HTTPS do not have port names assigned and must therefore be assigned using their respective port numbers (22 and 443)

Use ? examples.

# Extended IP ACLs Examples

- IP is for the entire protocol suite so port numbers cannot be specified.

```
access-list 101 permit ip any any
```

Permit all packets

```
access-list 101 deny ip any host 10.1.1.1
```

Deny all packets from any source address going specifically to host 10.1.1.1.

```
access-list 101 deny ip host 10.1.1.1 any
```

Deny all packets from host 10.1.1.1 going to any destination address.

# Extended TCP ACLs Examples

```
access-list 101 deny tcp any any eq telnet
```

Deny packets from any source address telnetting to anywhere.

```
access-list 101 deny tcp any host 10.1.1.1 eq 23
```

Deny packets from any source address telnetting to 10.1.1.1.

```
access-list 101 deny tcp any host 10.1.1.1 eq telnet
```

Same function as last example; except it denies using the keyword **telnet**.

# Extended TCP ACLs Examples

Port/protocol after source address refers to the source port

```
access-list 101 deny tcp any eq telnet any
```

Any TCP packets whose source port is 23 are denied access to any destination.

Port/protocol after destination address refers to the destination port

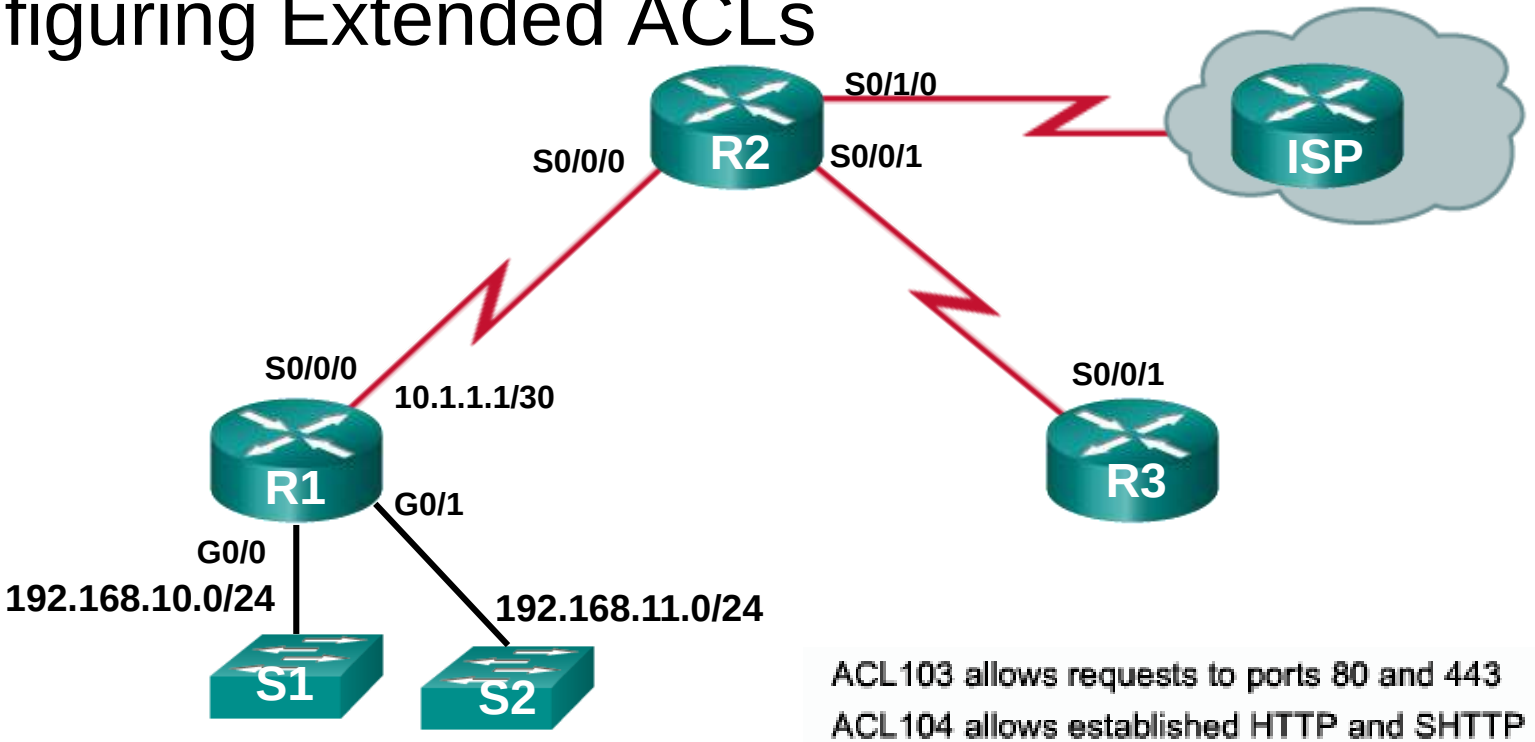
```
access-list 101 permit tcp 192.168.32.0 0.0.31.255 any eq 20
access-list 101 permit tcp 192.168.32.0 0.0.31.255 any eq 21
```

Packets from subnets 192.168.**32**.0 to 192.168.**63**.0 are permitted FTP access to any destination.

FTP requires both ports to be permitted.

- Port 20 = ftp-data
- Port 21 = ftp (commands)

# Configuring Extended ACLs

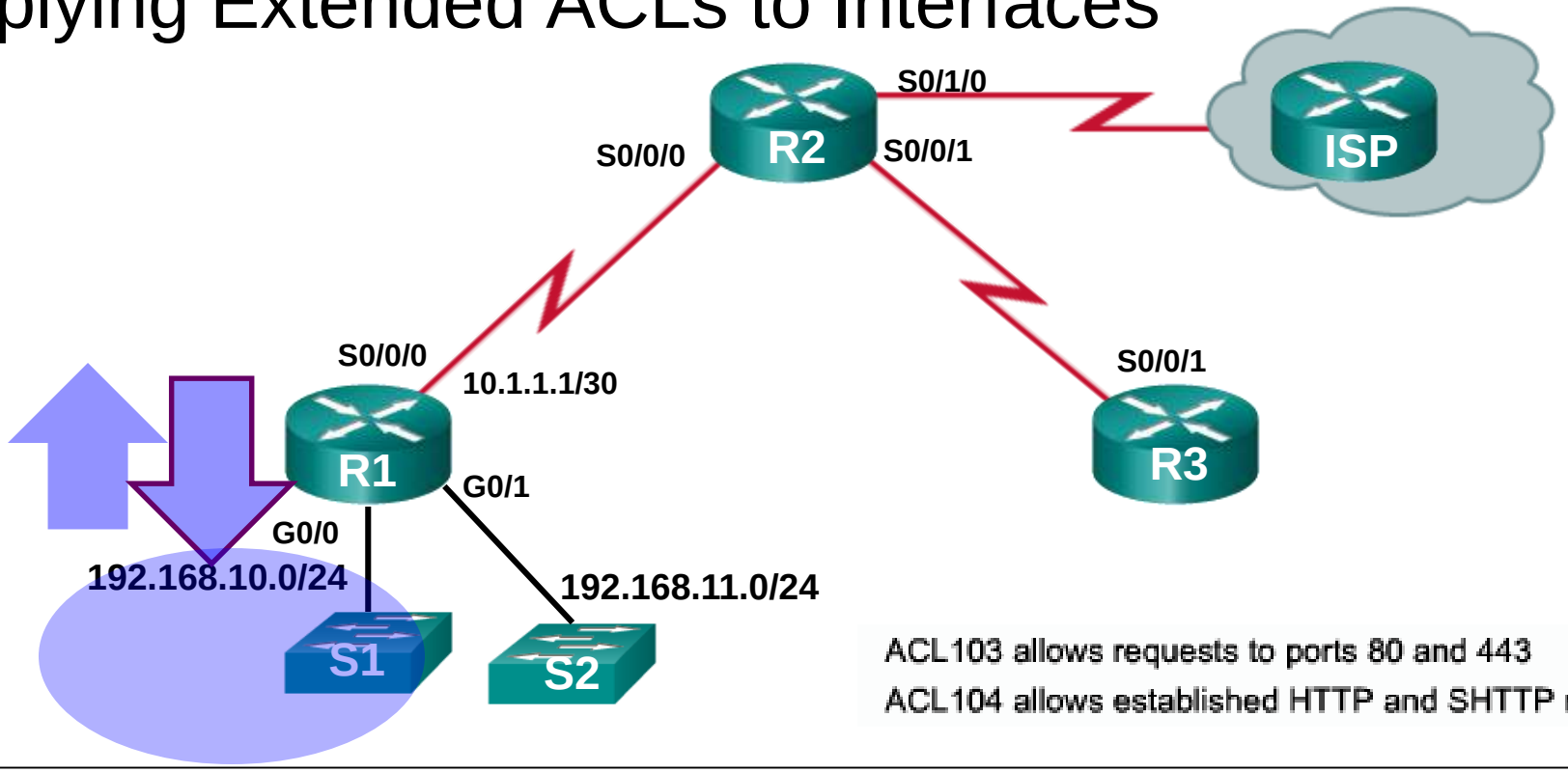



```
R1(config)# access-list 103 permit tcp 192.168.10.0 0.0.0.255 any eq 80
R1(config)# access-list 103 permit tcp 192.168.10.0 0.0.0.255 any eq 443
R1(config)# access-list 104 permit tcp any 192.168.10.0 0.0.0.255
established
```

The **established** parameter allows only responses to traffic that originates from the 192.168.10.0/24 network to return to that network.

Without the **established** parameter in the ACL statement, clients could send traffic to a web server, but not receive traffic returning from the web server.

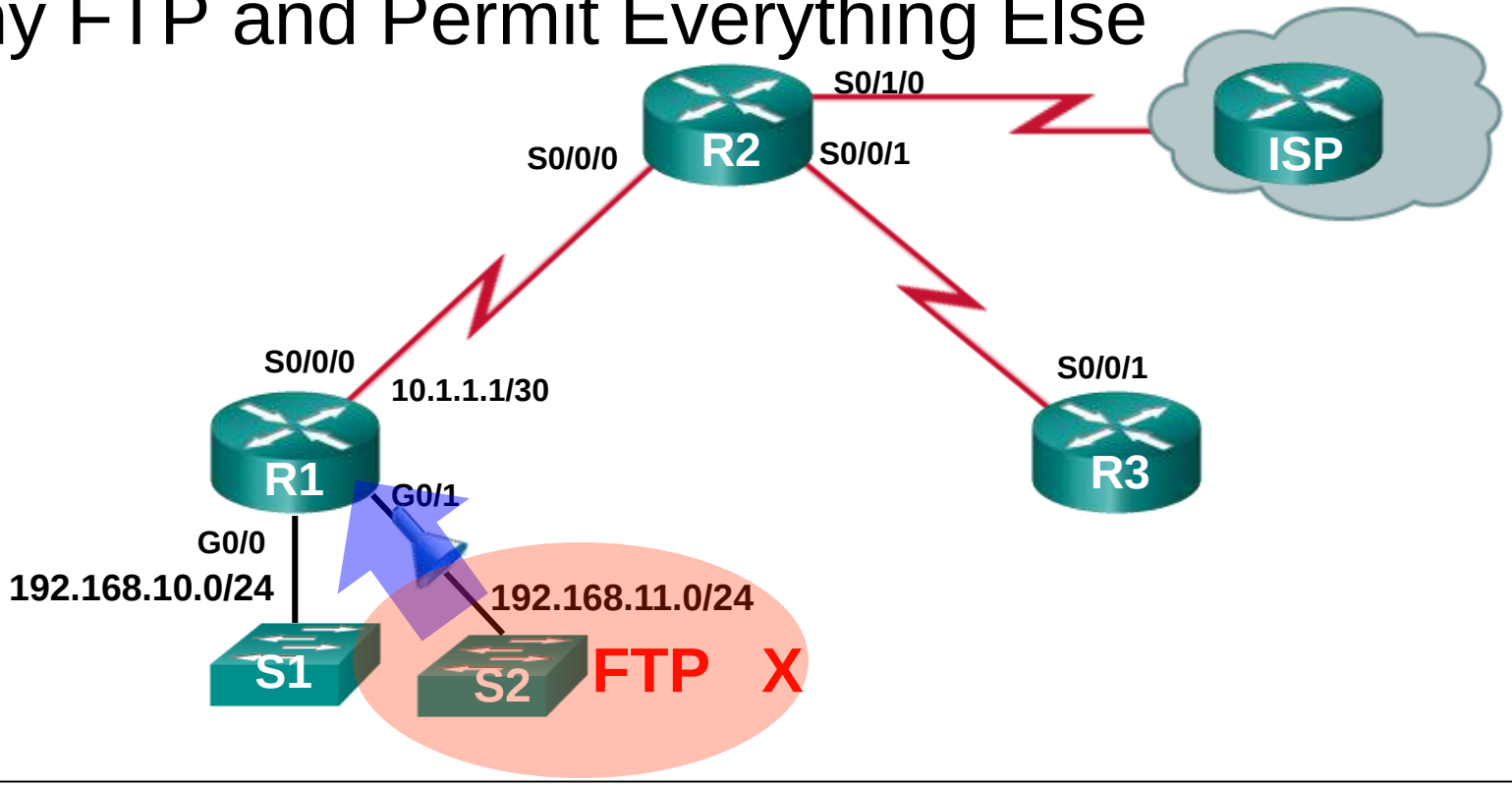
# Applying Extended ACLs to Interfaces




```
R1(config)# access-list 103 permit tcp 192.168.10.0 0.0.0.255 any eq 80
R1(config)# access-list 103 permit tcp 192.168.10.0 0.0.0.255 any eq 443
R1(config)# access-list 104 permit tcp any 192.168.10.0 0.0.0.255
established

R1(config)#interface g0/0
R1(config-if)# ip access-group 103 in
R1(config-if)# ip access-group 104 out
```

# Deny FTP and Permit Everything Else

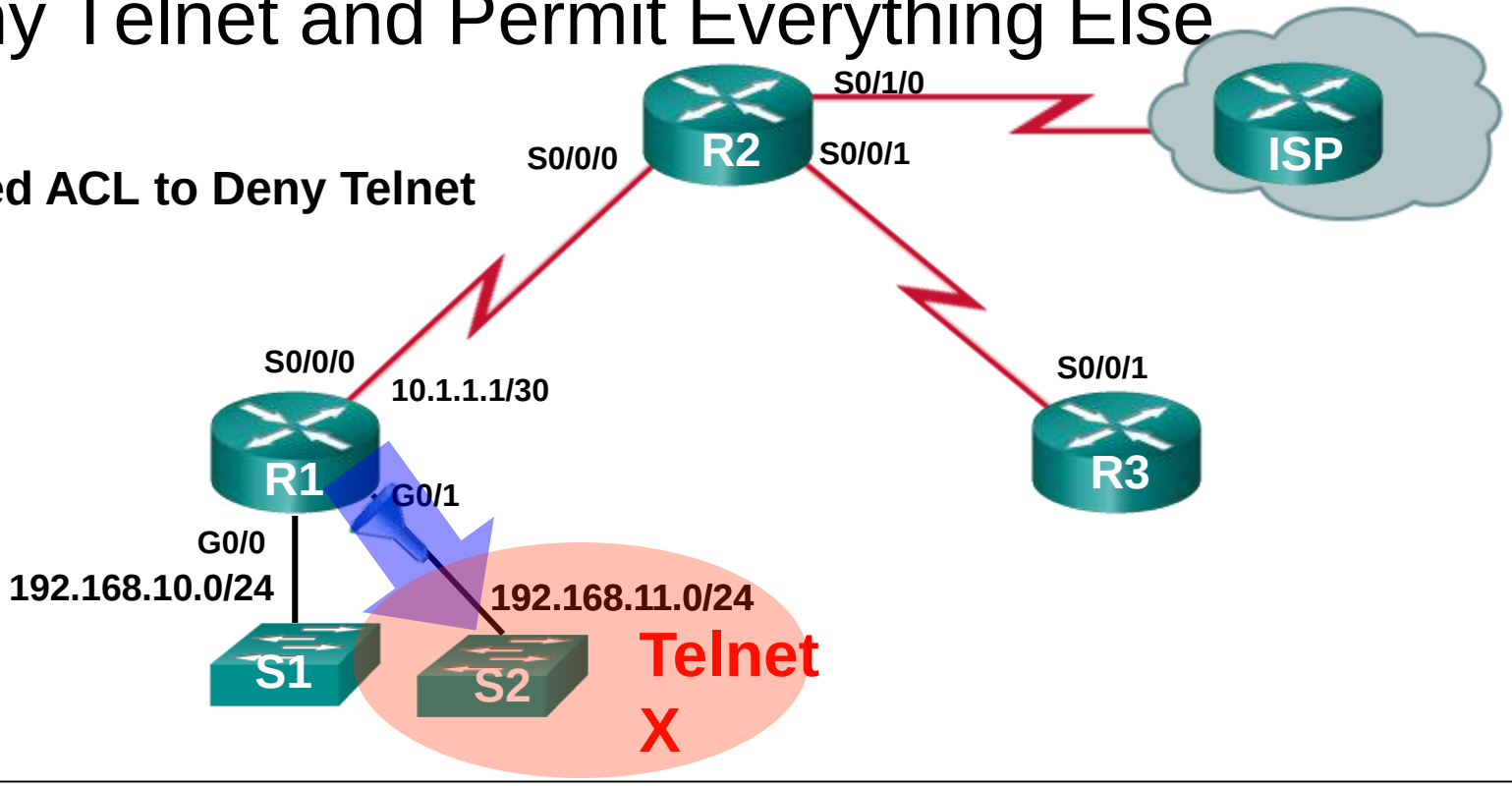



```
R1(config)# access-list 101 deny tcp 192.168.11.0 0.0.0.255
192.168.10.0 0.0.0.255 eq ftp
R1(config)# access-list 101 deny tcp 192.168.11.0 0.0.0.255
192.168.10.0 0.0.0.255 eq ftp-data
R1(config)# access-list 101 permit ip any any

R1(config)# interface g0/1
R1(config-if)# ip access-group 101 in
```

# Deny Telnet and Permit Everything Else

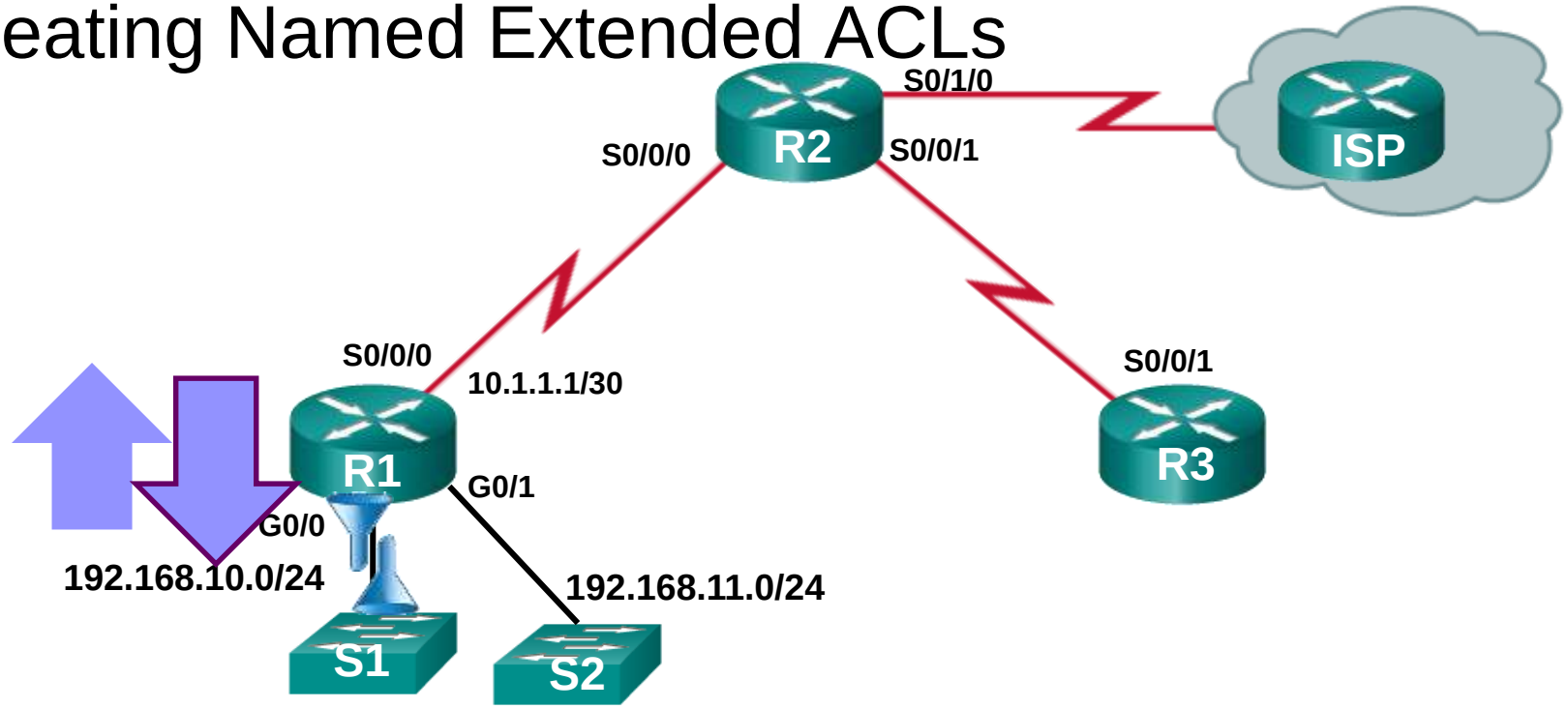
**Extended ACL to Deny Telnet**




```
R1(config)# access-list 102 deny tcp 192.168.11.0 0.0.0.255 any eq 23
R1(config)# access-list 102 permit ip any any

R1(config)# interface g0/1
R1(config-if)# ip access-group 102 out
```

# Creating Named Extended ACLs




```
R1(config)# ip access-list extended SURFING
R1(config-ext-nacl)# permit tcp 192.168.10.0 0.0.0.255 any eq 80
R1(config-ext-nacl)# permit tcp 192.168.10.0 0.0.0.255 any eq 443
R1(config-ext-nacl)# exit
R1(config)# ip access-list extended BROWSING
R1(config-ext-nacl)# permit tcp any 192.168.10.0 0.0.0.255 established
R1(config-ext-nacl)# exit
R1(config)# interface g0/0
R1(config-if)# ip access-group SURFING in
R1(config-if)# ip access-group BROWSING out
```

# Verifying Extended ACLs

```
R1#show access-lists
Extended IP access list BROWSING
    10 permit tcp any 192.168.10.0 0.0.0.255 established
Extended IP access list SURFING
    10 permit tcp 192.168.10.0 0.0.0.255 any eq www
    20 permit tcp 192.168.10.0 0.0.0.255 any eq 443
R1#
R1#show ip interface g0/0
GigabitEthernet0/0 is up, line protocol is up
  Internet address is 192.168.10.1/24
  <output omitted for brevity>
  Outgoing access list is BROWSING
  Inbound  access list is SURFING
  <rest of output omitted for brevity>
```

# Editing Extended ACLs

```
R1# show access-lists
Extended IP access list BROWSING
    10 permit tcp any 192.168.10.0 0.0.0.255 established
Extended IP access list SURFING
    10 permit tcp 192.168.11.0 0.0.0.255 any eq www
    20 permit tcp 192.168.10.0 0.0.0.255 any eq 443
R1#
R1# configure terminal
R1(config)# ip access-list extended SURFING
R1(config-ext-nacl)# no 10
R1(config-ext-nacl)# 10 permit tcp 192.168.10.0 0.0.0.255
any eq www
R1(config-ext-nacl)# exit
R1#
R1#show access-lists
Extended IP access list BROWSING
    10 permit tcp any 192.168.10.0 0.0.0.255 established
Extended IP access list SURFING
    10 permit tcp 192.168.10.0 0.0.0.255 any eq www
    20 permit tcp 192.168.10.0 0.0.0.255 any eq 443
```

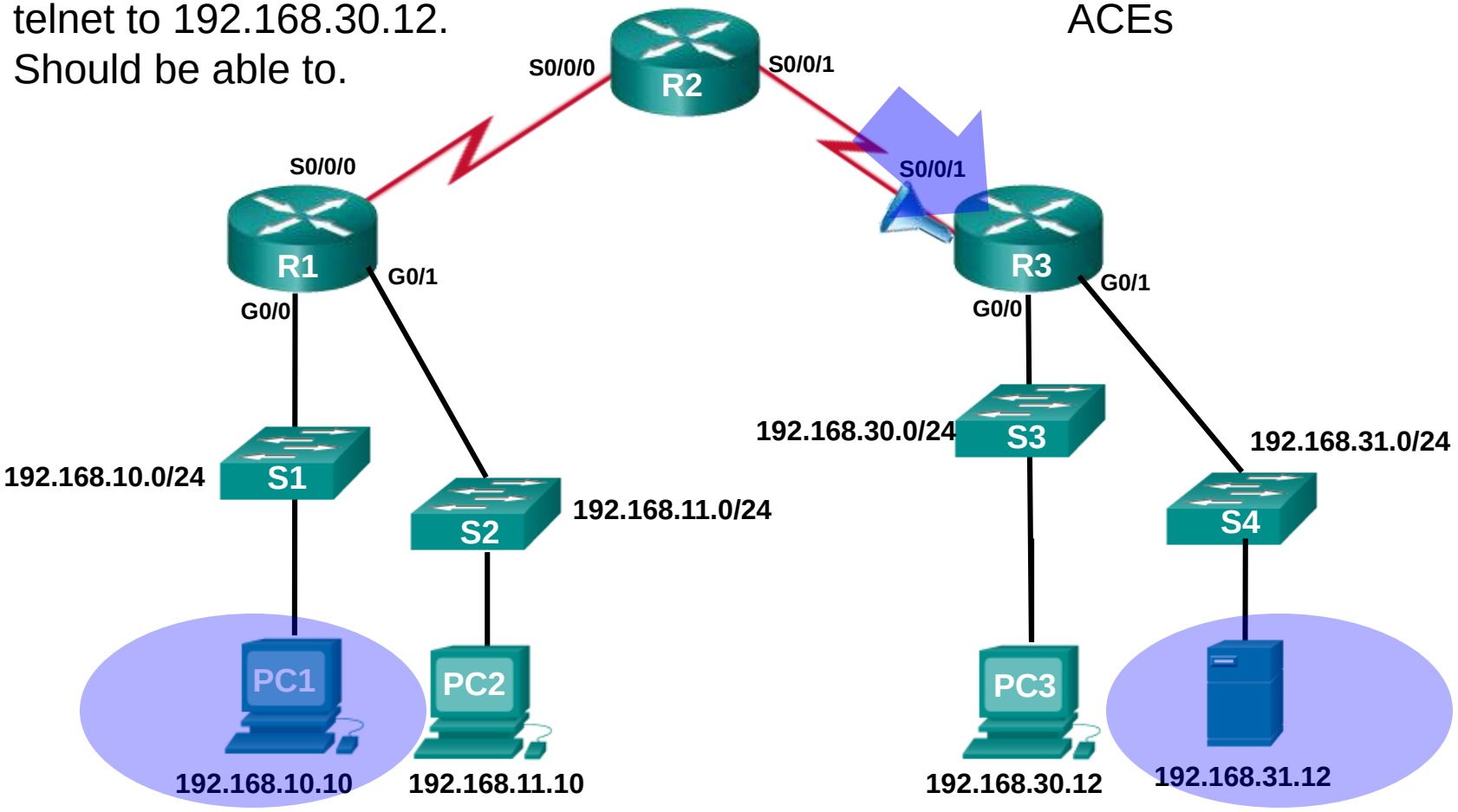
Should be 192.168.10.0

# Troubleshooting Common ACL Operations – Error 1

```
R3# show access-lists
Extended IP access list 110
    10 deny tcp 192.168.10.0 0.0.0.255 any (12 match(es))
    20 permit tcp 192.168.10.0 0.0.0.255 any eq telnet
    30 permit ip any any
```

192.168.10.10 cannot telnet to 192.168.30.12. Should be able to.

Look at the order of ACEs

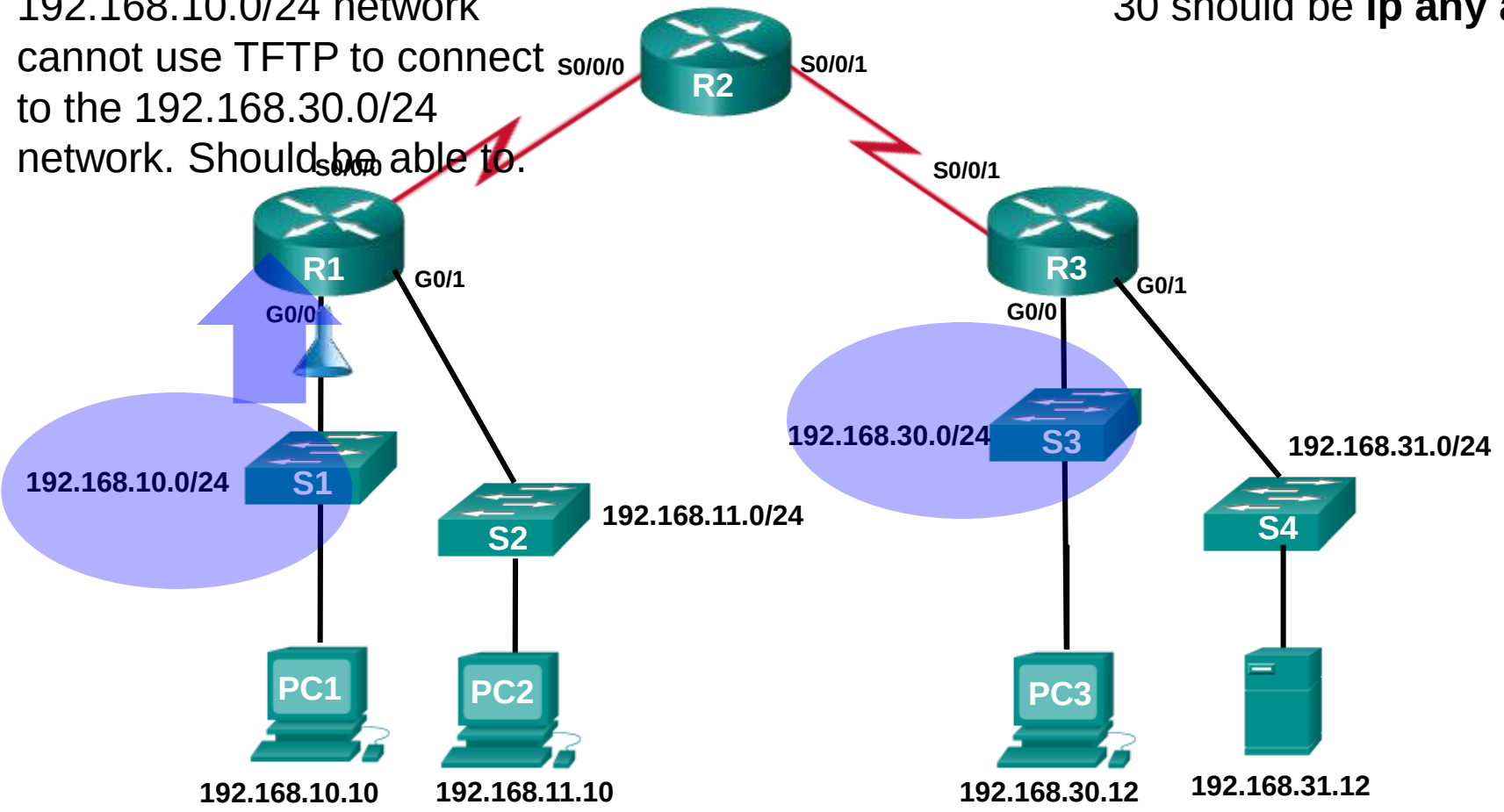


# Troubleshooting Common ACL Operations – Error 2

```
R1# show access-lists 120
Extended IP access list 120
    10 deny tcp 192.168.10.0 0.0.0.255 any eq telnet
    20 deny tcp 192.168.10.0 0.0.0.255 host 192.168.31.12 eq smtp
    30 permit tcp any any
```

192.168.10.0/24 network cannot use TFTP to connect to the 192.168.30.0/24 network. Should be able to.

30 should be **ip any any**

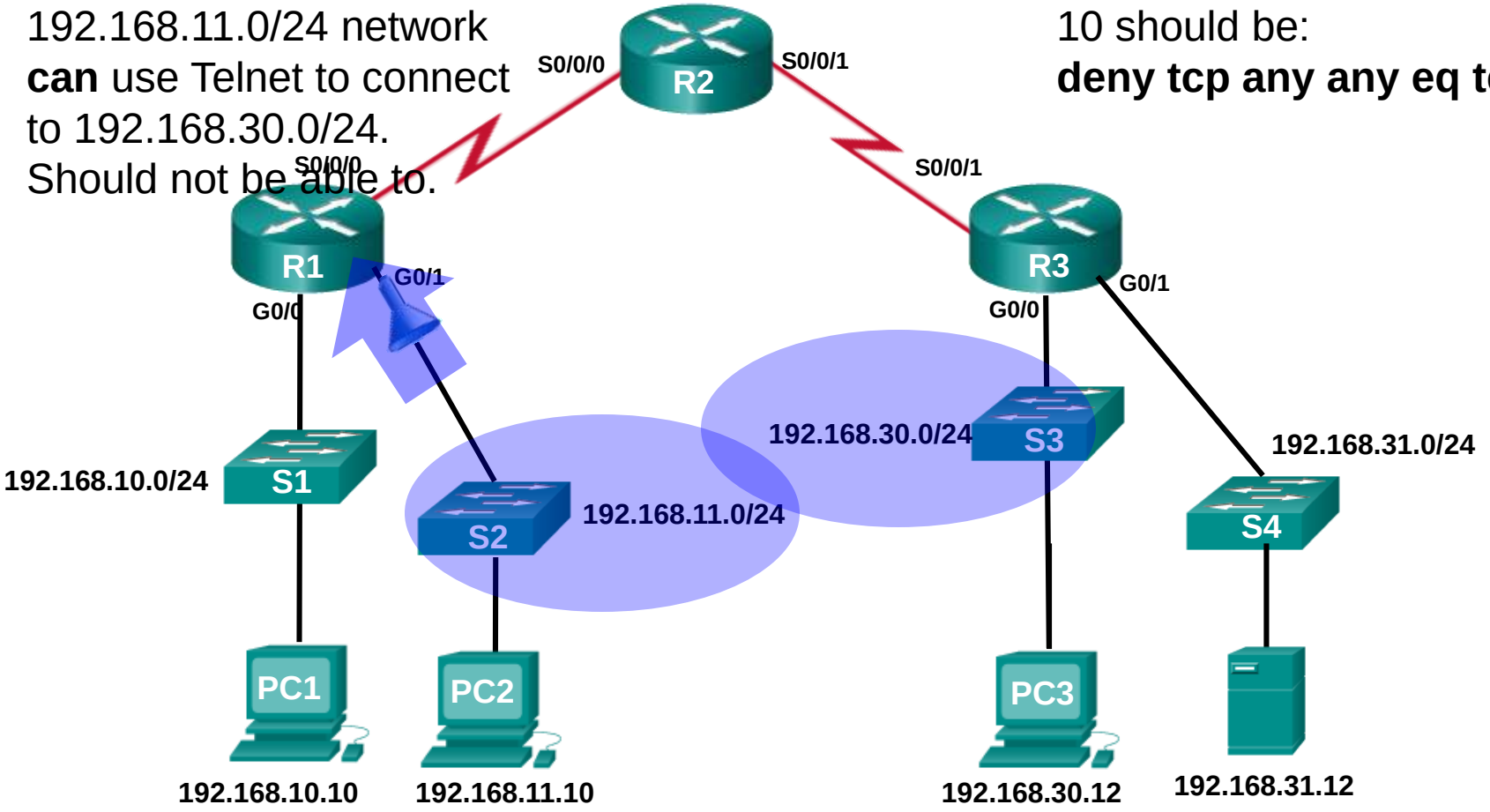


# Troubleshooting Common ACL Operations – Error 3

```
R1# show access-lists 130
Extended IP access list 130
    10 deny tcp any eq telnet any
    20 deny tcp 192.168.11.0 0.0.0.255 host 192.168.31.12 eq smtp
    30 permit tcp any any (12 match(es))
```

192.168.11.0/24 network **can** use Telnet to connect to 192.168.30.0/24. Should not be able to.

10 should be:
**deny tcp any any eq telnet**

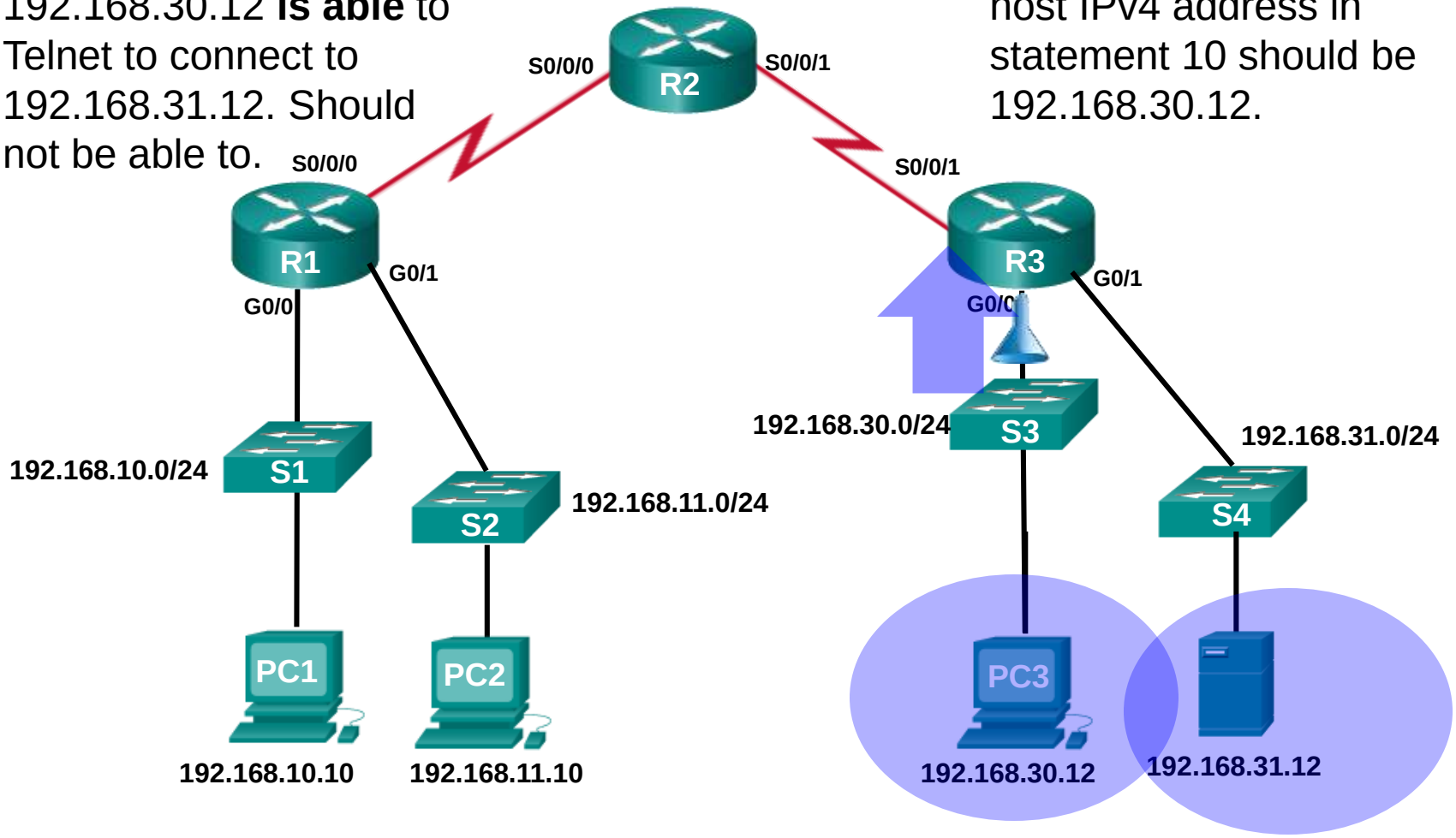


# Troubleshooting Common ACL Operations – Error 4

```
R3# show access-lists 140
Extended IP access list 140
    10 deny tcp host 192.168.30.1 any eq telnet
    20 permit ip any any (5 match(es))
```

192.168.30.12 **is able** to Telnet to connect to 192.168.31.12. Should not be able to.

host IPv4 address in statement 10 should be 192.168.30.12.

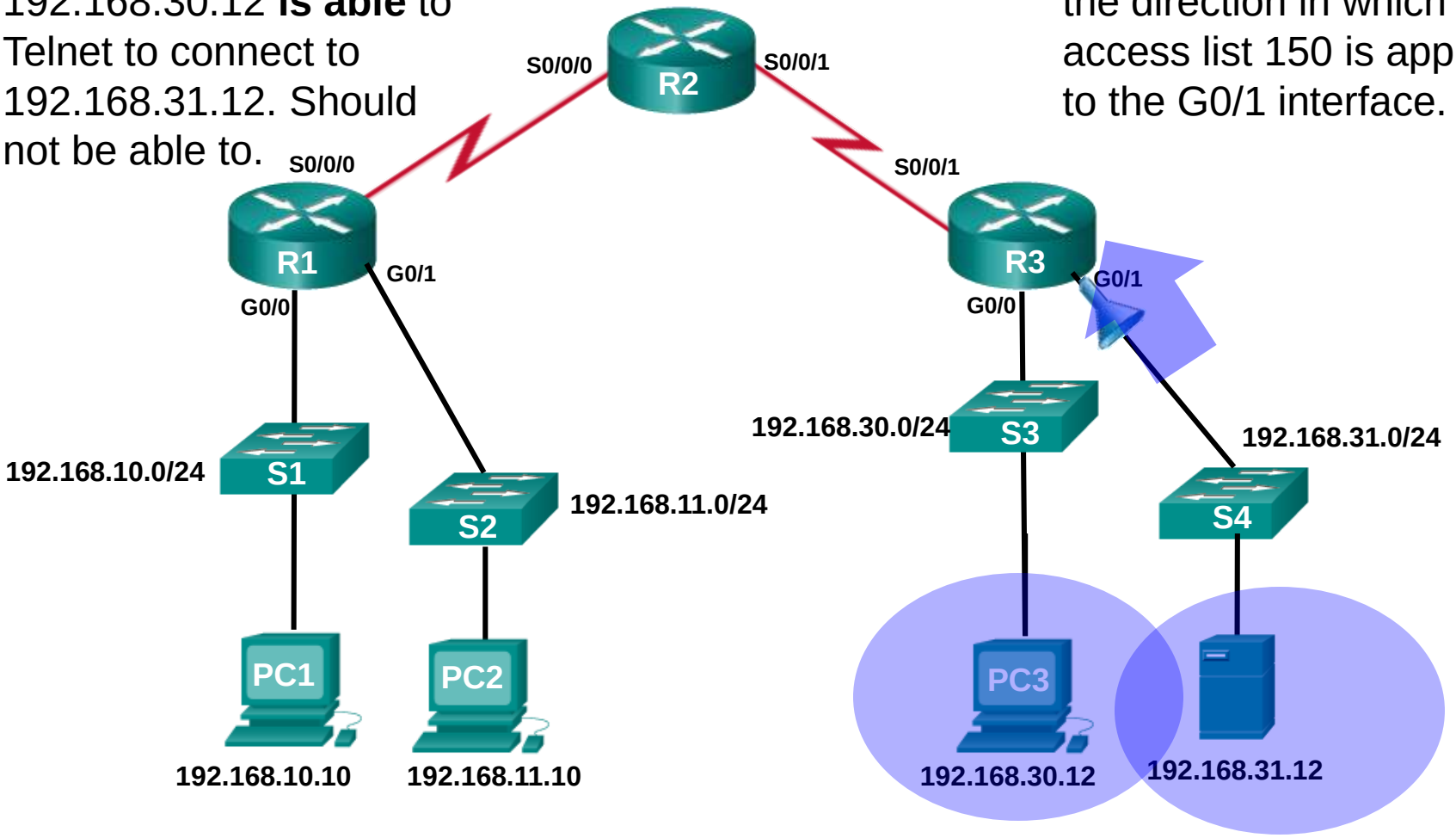


# Troubleshooting Common ACL Operations – Error 5

```
R2# show access-lists 150
Extended IP access list 150
    10 deny tcp any host 192.168.31.12 eq telnet
    20 permit ip any any
```

192.168.30.12 **is able** to Telnet to connect to 192.168.31.12. Should not be able to.

the direction in which access list 150 is applied to the G0/1 interface.



# Configuring IPv6 ACLs

# IPv6 ACL

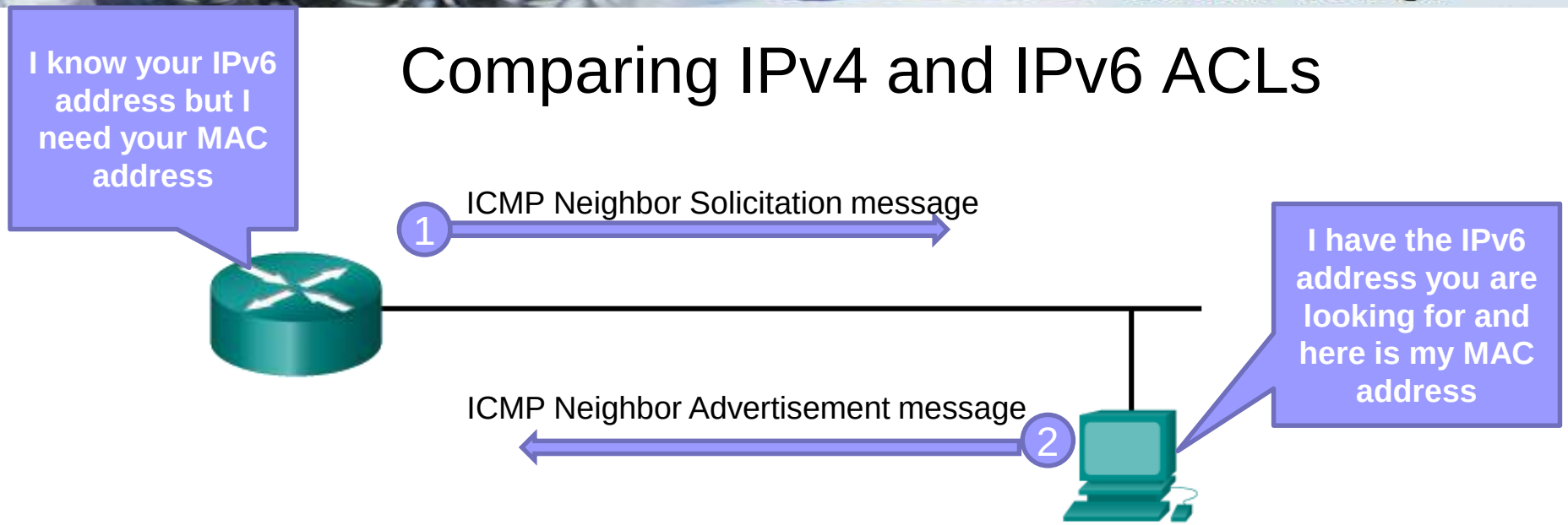
**IPv4 ACLs**

- Standard
  - Numbered
  - Named
- Extended
  - Numbered
  - Named

**IPv6 ACLs**

- Named only
- Similar features to Extended ACLs

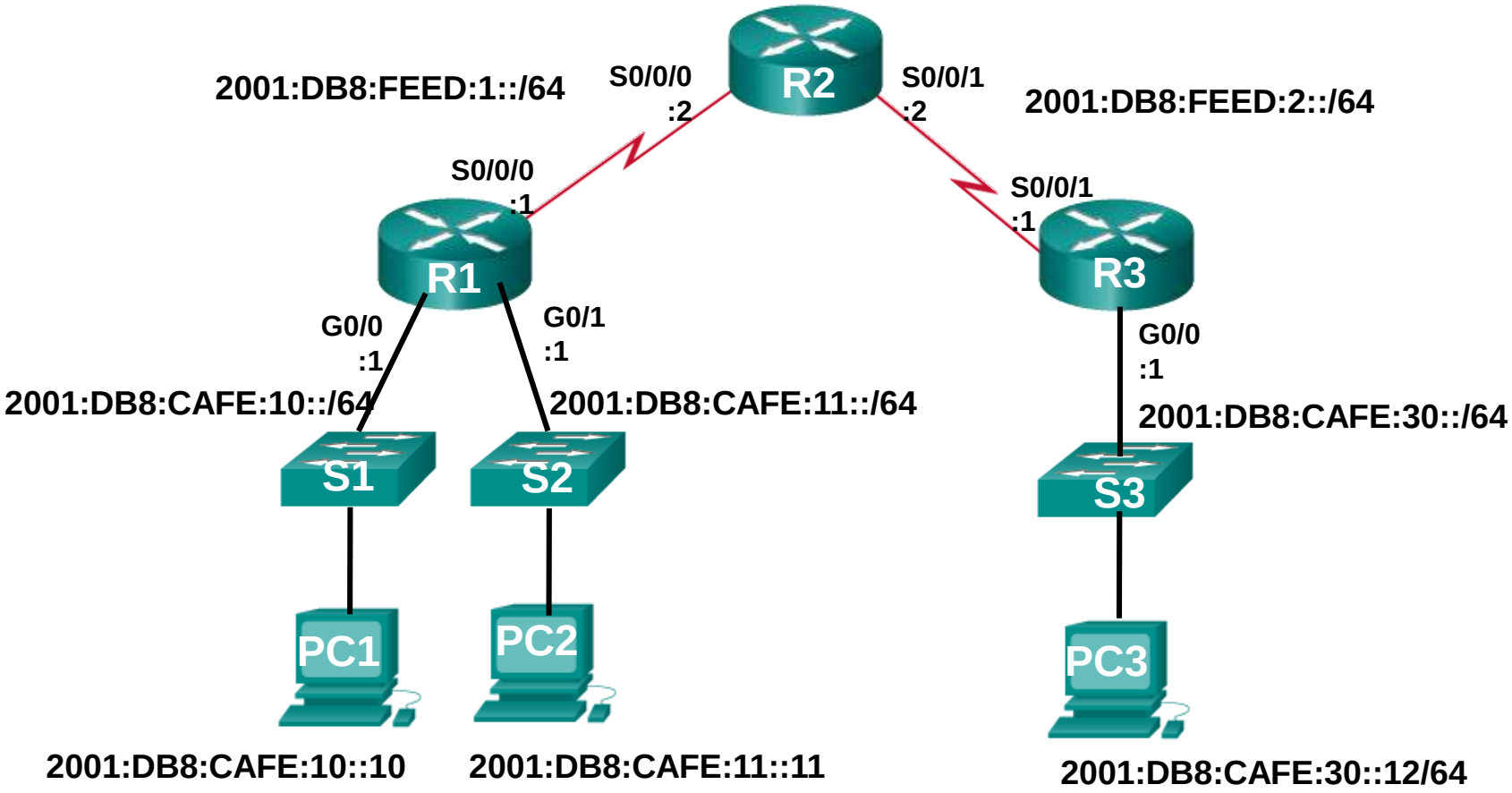
# Comparing IPv4 and IPv6 ACLs




Very similar, but there are three significant differences

- **Applying an IPv6 ACL**
  - IPv4 - **ip access-group**
  - IPv6 - **ipv6 traffic-filter**
- **No Wildcard Masks** - Instead, the prefix-length is used
- **Additional Default Statements**
  - **permit icmp any any nd-na**
  - **permit icmp any any nd-ns**
  - These two statements allow the router to participate in the IPv6 equivalent of ARP for IPv4.

# IPv6 Topology



# Configuring the IPv6 Topology

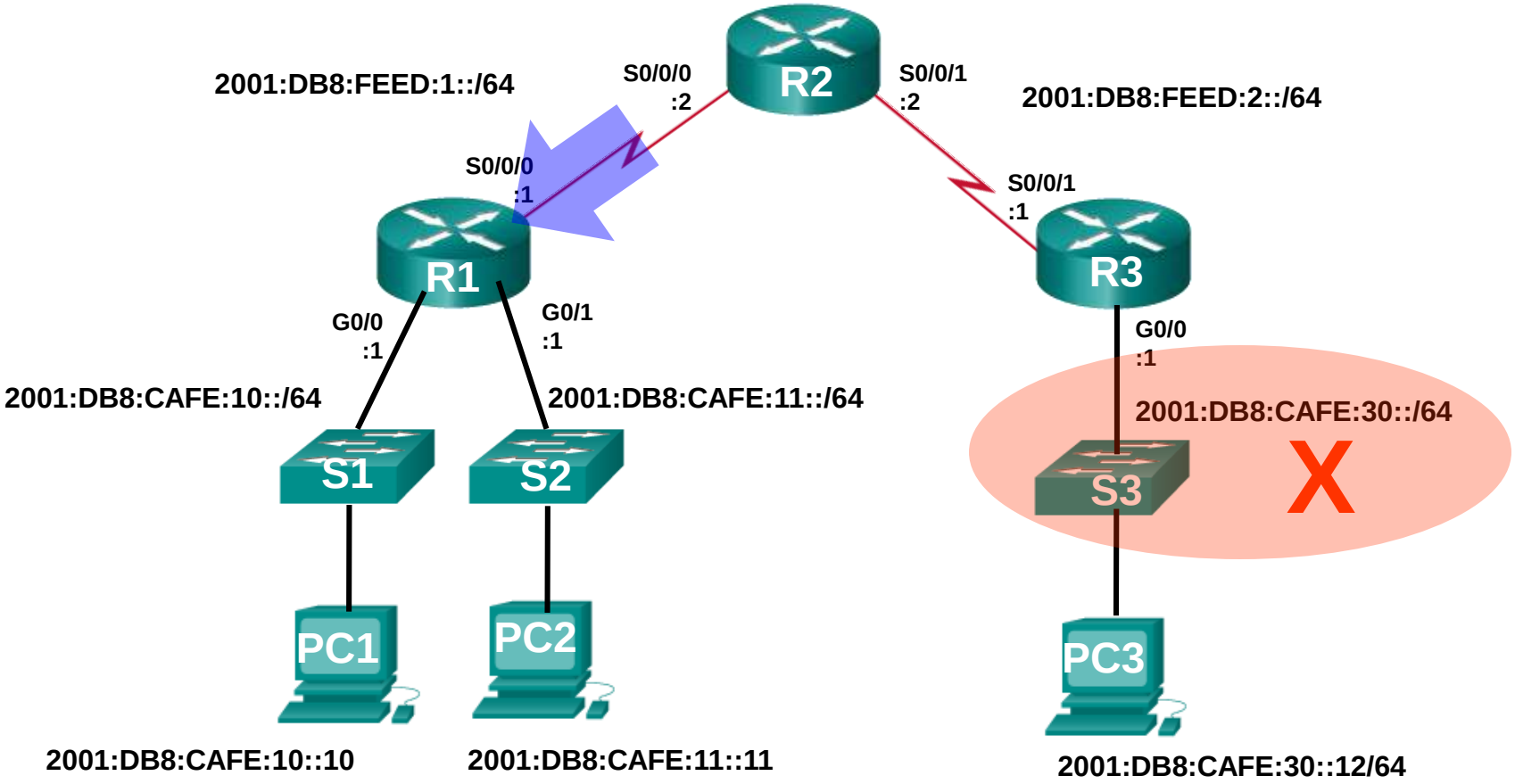
```
R1(config)#interface g0/0
R1(config-if)#ipv6 address 2001:db8:cafe:10::1/64
R1(config-if)#exit
R1(config)#interface s0/0/0
R1(config-if)#ipv6 address 2001:db8:feed:1::1/64
R1(config-if)#exit
R1(config)#interface g0/1
R1(config-if)#ipv6 address 2001:db8:cafe:11::1/64
R1(config-if)#end
R1#show ipv6 interface brief
GigabitEthernet0/0     [up/up]
    FE80::FE99:47FF:FE75:C3E0
    2001:DB8:CAFE:10::1
GigabitEthernet0/1     [up/up]
    FE80::FE99:47FF:FE75:C3E1
    2001:DB8:CAFE:11::1
Serial0/0/0            [up/up]
    FE80::FE99:47FF:FE75:C3E0
    2001:DB8:FEED:1::1
<some output omitted for brevity>
R1#
```

```
R2(config)#interface s0/0/0
R2(config-if)#ipv6 address 2001:db8:feed:1::2/64
R2(config-if)#exit
R2(config)#interface s0/0/1
R2(config-if)#ipv6 address 2001:db8:feed:2::2/64
R2(config-if)#end
R2#show ipv6 interface brief
Serial0/0/0             [up/up]
    FE80::FE99:47FF:FE71:78A0
    2001:DB8:FEED:1::2
Serial0/0/1             [up/up]
    FE80::FE99:47FF:FE71:78A0
    2001:DB8:FEED:2::2
<some output omitted for brevity>
R2#
```

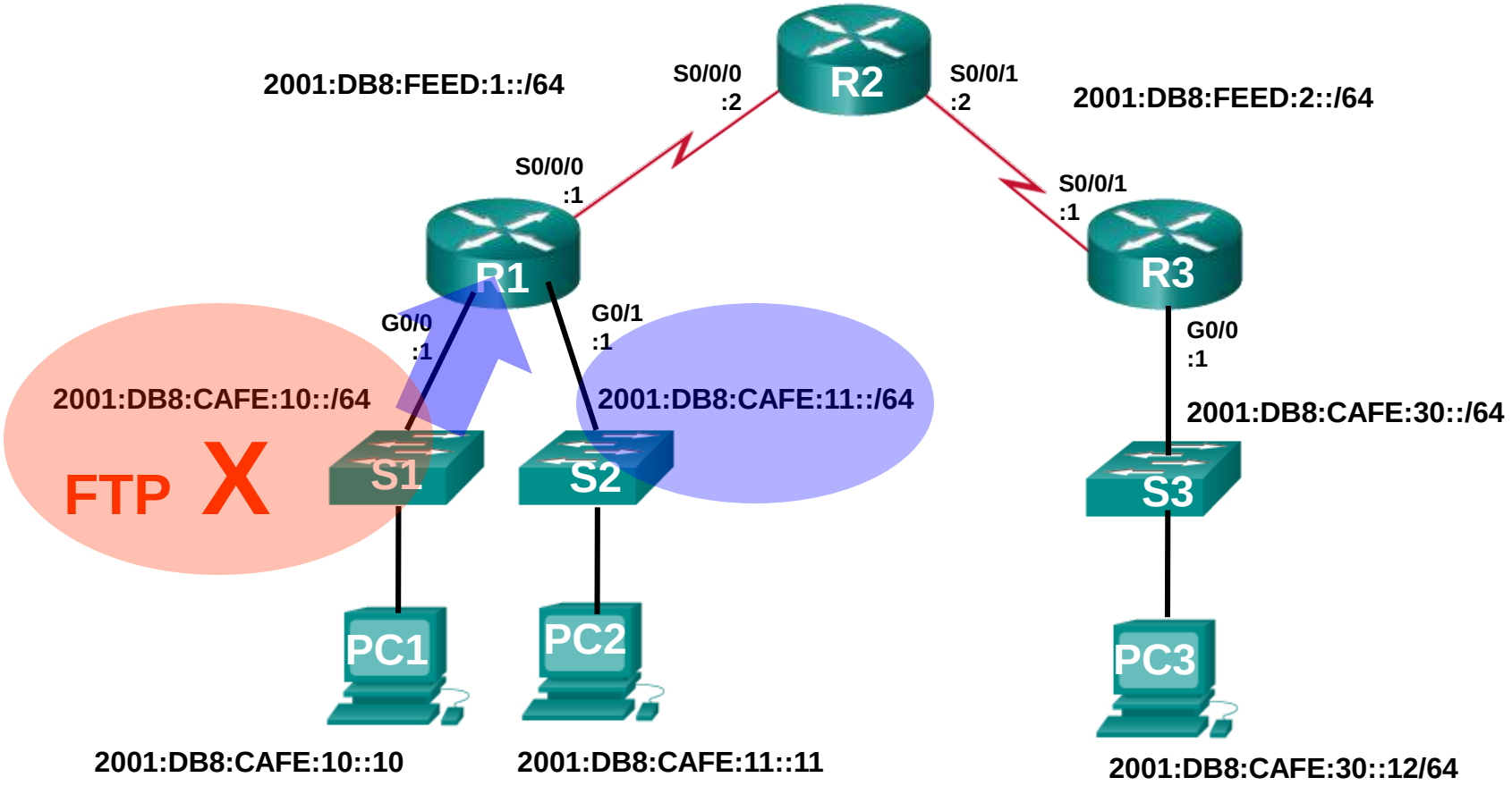
```
R3(config)#interface s0/0/1
R3(config-if)#ipv6 address 2001:db8:feed:2::1/64
R3(config-if)#exit
R3(config)#interface g0/0
R3(config-if)#ipv6 address 2001:db8:cafe:30::1/64
R3(config-if)#end
R3#show ipv6 interface brief
GigabitEthernet0/0     [up/up]
    FE80::FE99:47FF:FE71:7A20
    2001:DB8:CAFE:30::1
Serial0/0/1            [up/up]
    FE80::FE99:47FF:FE71:7A20
    2001:DB8:FEED:2::1
R3#
```

```
R1(config-ipv6-acl)# deny | permit protocol {source-ipv6-prefix/prefix-length | any
| host source-ipv6-address} [operator [port-number]] {destination-ipv6-prefix/
prefix-length | any | host destination-ipv6-address} [operator [port-number]]
```

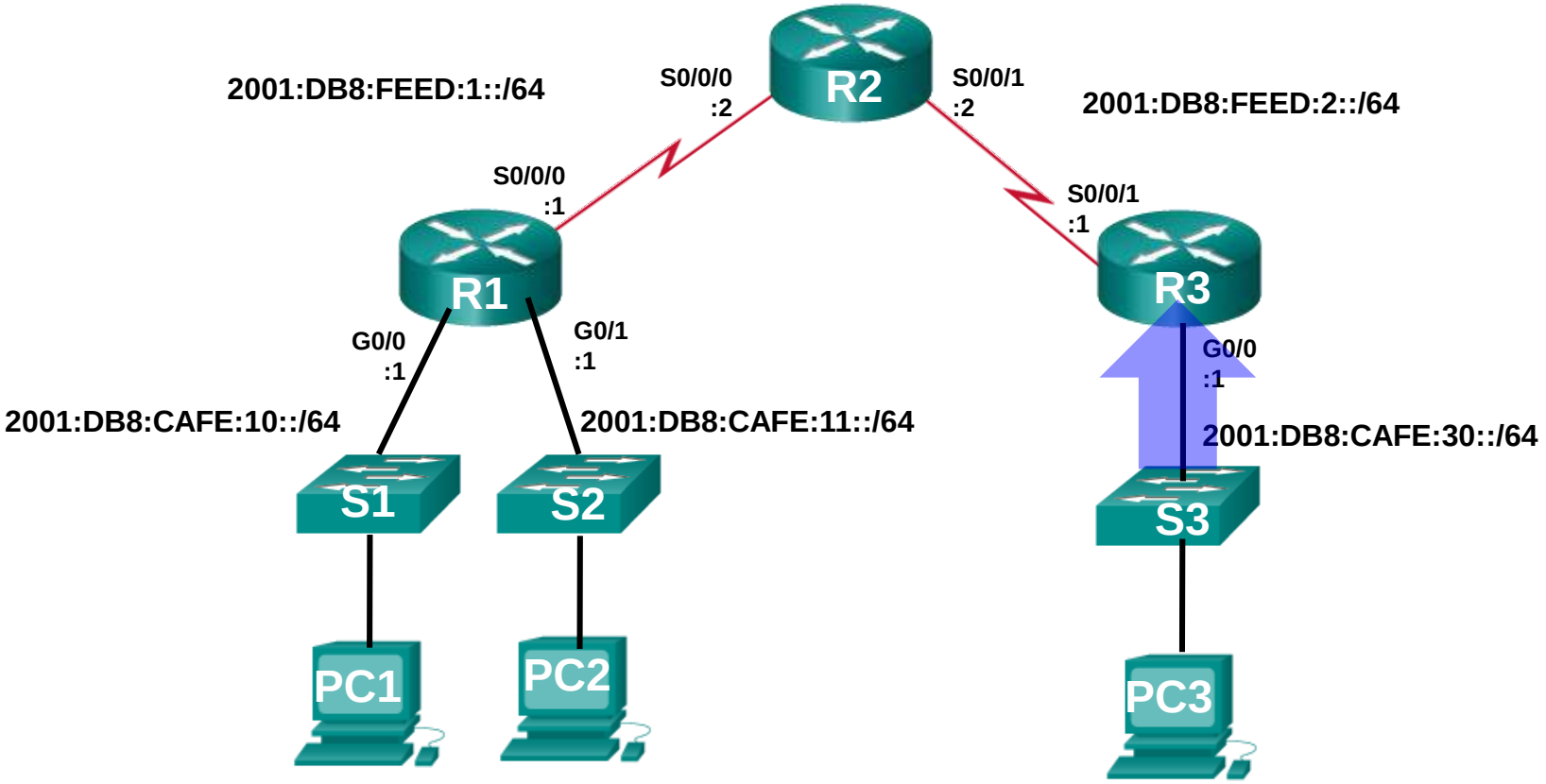
| Parameter | Description |
|---|---|
| **deny** \| **permit** | Specifies whether to deny or permit the packet. |
| *protocol* | Enter the name or number of an Internet protocol, or an integer representing an IPv6 protocol number. |
| *source-ipv6-prefix*/*prefix-length*<br><br>*destination-ipv6-address* | The source or destination IPv6 network or class of networks for which to set deny or permit conditions |
| **any** | Enter **any** as an abbreviation for the IPv6 prefix ::/0. This matches all addresses. |
| **host** | For **host** *source-ipv6-address* or *destination-ipv6-address*, enter the source or destination IPv6 host address for which to set deny or permit conditions |
| *operator* | (Optional) An operand that compares the source or destination ports of the specified protocol. Operands are lt (less than), gt (greater than), eq (equal), neq (not equal), and range. |
| *port-number* | (Optional) A decimal number or the name of a TCP or UDP port for filtering TCP or UDP, respectively. |




```
R1(config)# ipv6 access-list NO-R3-LAN-ACCESS
R1(config-ipv6-acl)# deny ipv6 2001:db8:cafe:30::/64 any
R1(config-ipv6-acl)# permit ipv6 any any
R1(config-ipv6-acl)# end
R1#
R1(config)# interface s0/0/0
R1(config-if)# ipv6 traffic-filter NO-R3-LAN-ACCESS in
```



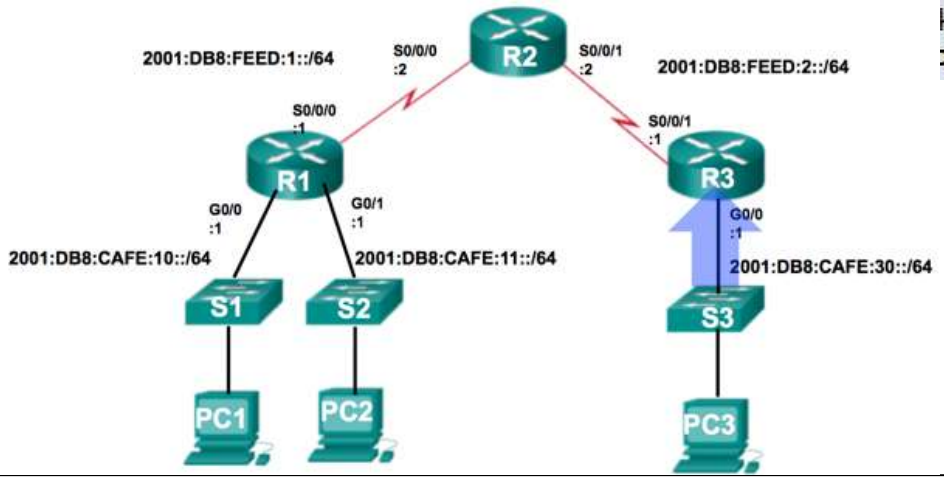

```
R1(config)# ipv6 access-list NO-FTP-TO-11
R1(config-ipv6-acl)# deny tcp any 2001:db8:cafe:11::/64 eq ftp
R1(config-ipv6-acl)# deny tcp any 2001:db8:cafe:11::/64 eq ftp-data
R1(config-ipv6-acl)# permit ipv6 any any
R1(config-ipv6-acl)# exit
R1(config)# interface g0/0
R1(config-if)# ipv6 traffic-filter NO-FTP-TO-11 in
```




- Permit access only HTTP and HTTPS to Network 10
- Deny all other traffic to Network 10

- Permit PC3 telnet access to PC2
- Deny telnet access to PC2 for all other devices

- Permit access to everything else




```
R3(config)# ipv6 access-list RETRICTED-ACCESS
R3(config-ipv6-acl)# remark Permit access only HTTP and HTTPS to Network 10
R3(config-ipv6-acl)# permit tcp any host 2001:db8:cafe:10::10 eq 80
R3(config-ipv6-acl)# permit tcp any host 2001:db8:cafe:10::10 eq 443

R3(config-ipv6-acl)# remark Deny all other traffic to Network 10
R3(config-ipv6-acl)# deny ipv6 any 2001:db8:cafe:10::/64

R3(config-ipv6-acl)# remark Permit PC3 telnet access to PC2
R3(config-ipv6-acl)# permit tcp host 2001:DB8:CAFE:30::12 host 2001:DB8:CAFE:11::11
eq 23

R3(config-ipv6-acl)# remark Deny telnet access to PC2 for all other devices
R3(config-ipv6-acl)# deny tcp any host 2001:db8:cafe:11::11 eq 23

R3(config-ipv6-acl)#remark Permit access to everything else
R3(config-ipv6-acl)#permit ipv6 any any
R3(config-ipv6-acl)#exit

R3(config)#interface g0/0
R3(config-if)#ipv6 traffic-filter RESTRICTED-ACCESS in
```

# Verifying IPv6 ACLs

```
R3# show ipv6 interface g0/0
GigabitEthernet0/0 is up, line protocol is up
  Global unicast address(es):
    2001:DB8:CAFE:30::1, subnet is 2001:DB8:CAFE:30::/64
  Input features: Access List
  Inbound access list RESTRICTED-ACCESS
<some output omitted for brevity>
```

```
R3# show access-lists
IPv6 access list RESTRICTED-ACCESS
    permit tcp any host 2001:DB8:CAFE:10::10 eq www sequence 20
    permit tcp any host 2001:DB8:CAFE:10::10 eq 443 sequence 30
    deny ipv6 any 2001:DB8:CAFE:10::/64 sequence 50
    permit tcp host 2001:DB8:CAFE:30::12 host 2001:DB8:CAFE:11::11
eq telnet sequence 70
    deny tcp any host 2001:DB8:CAFE:11::11 eq telnet sequence 90
    permit ipv6 any any sequence 110
R3#
```

# Verifying IPv6 ACLs

```
R3# show running-config
<some output omitted for brevity>
ipv6 access-list RESTRICTED-ACCESS
 remark Permit access only HTTP and HTTPS to Network 10
 permit tcp any host 2001:DB8:CAFE:10::10 eq www
 permit tcp any host 2001:DB8:CAFE:10::10 eq 443
 remark Deny all other traffic to Network 10
 deny ipv6 any 2001:DB8:CAFE:10::/64
 remark Permit PC3 telnet access to PC2
 permit tcp host 2001:DB8:CAFE:30::12 host 2001:DB8:CAFE:11::11 eq telnet
 remark Deny telnet access to PC2 for all other devices
 deny tcp any host 2001:DB8:CAFE:11::11 eq telnet
 remark Permit access to everything else
 permit ipv6 any any
```

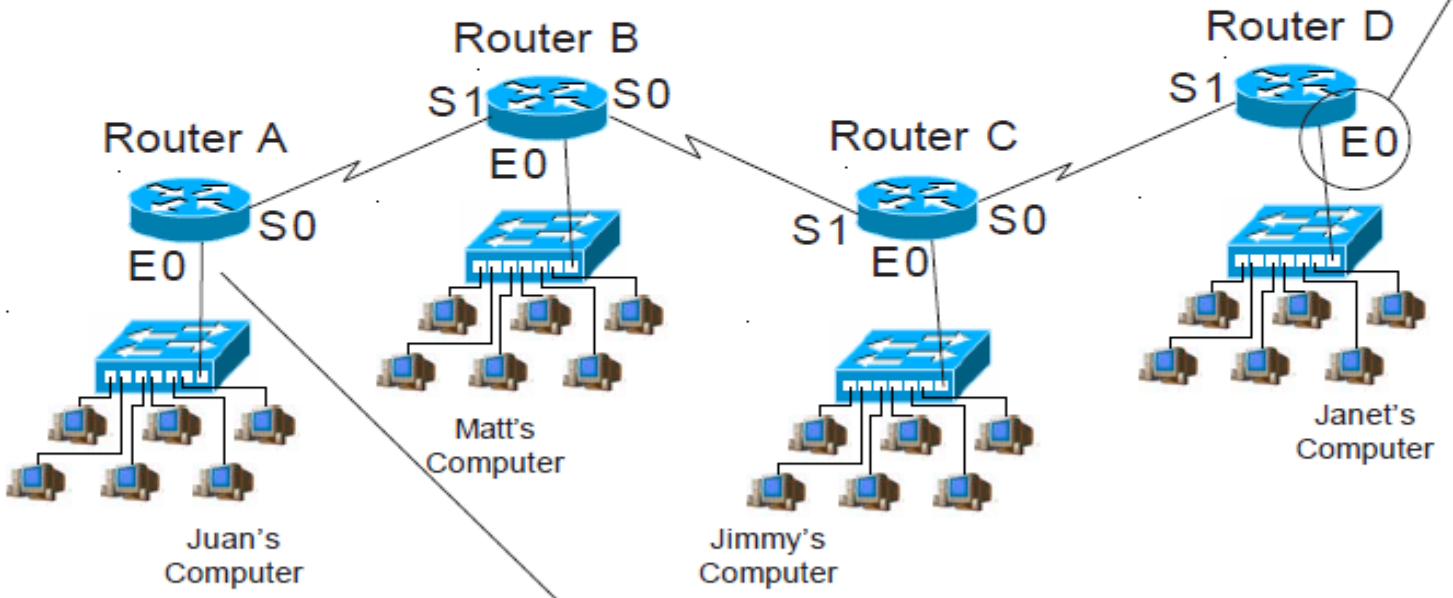
## Standard Access Lists

Standard Access Lists...

- ...are numbered from 1 to 99.
- ...filter (permit or deny) only source addresses.
- ...do not have any destination information so it must placed as close to the destination as possible.
- ...work at layer 3 of the OSI model.

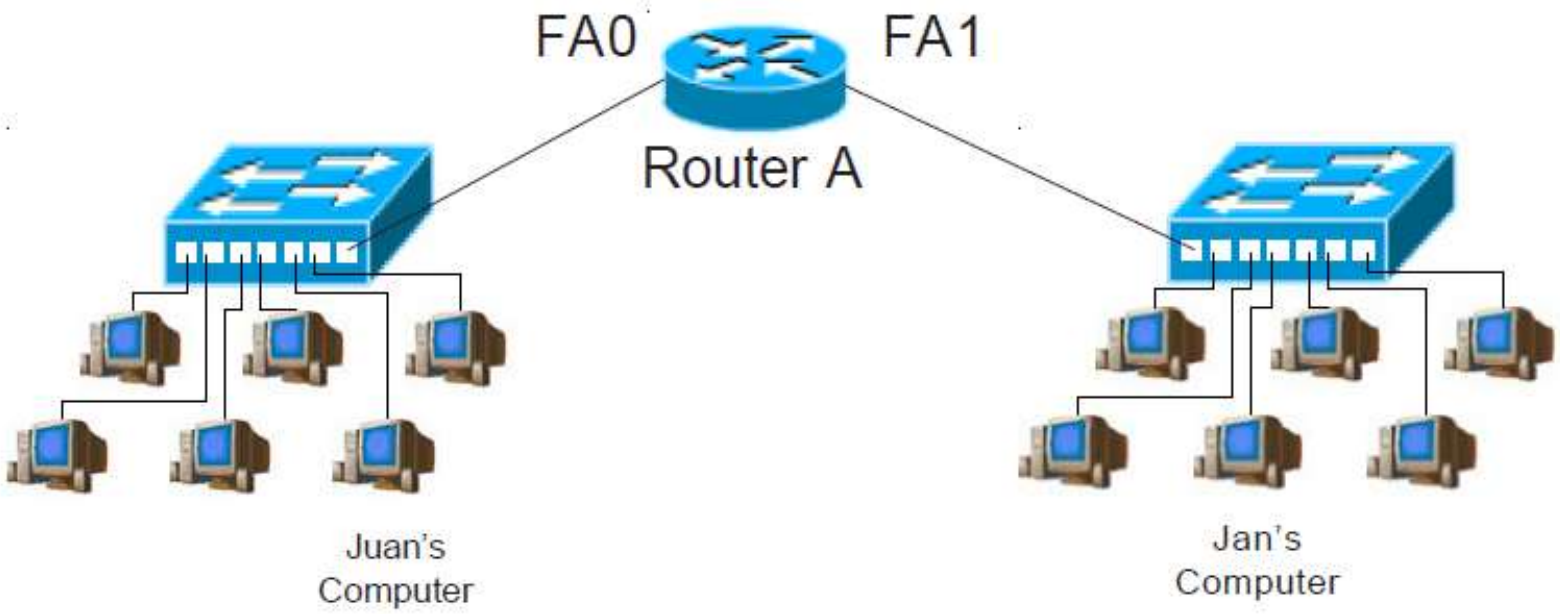
# Why standard ACLs are placed close to the destination.

If you want to block traffic from Juan's computer from reaching Janet's computer with a standard access list you would place the ACL **close to the destination** on Router D, interface E0. Since its using only the source address to permit or deny packets the ACL here will not effect packets reaching Routers B, or C.

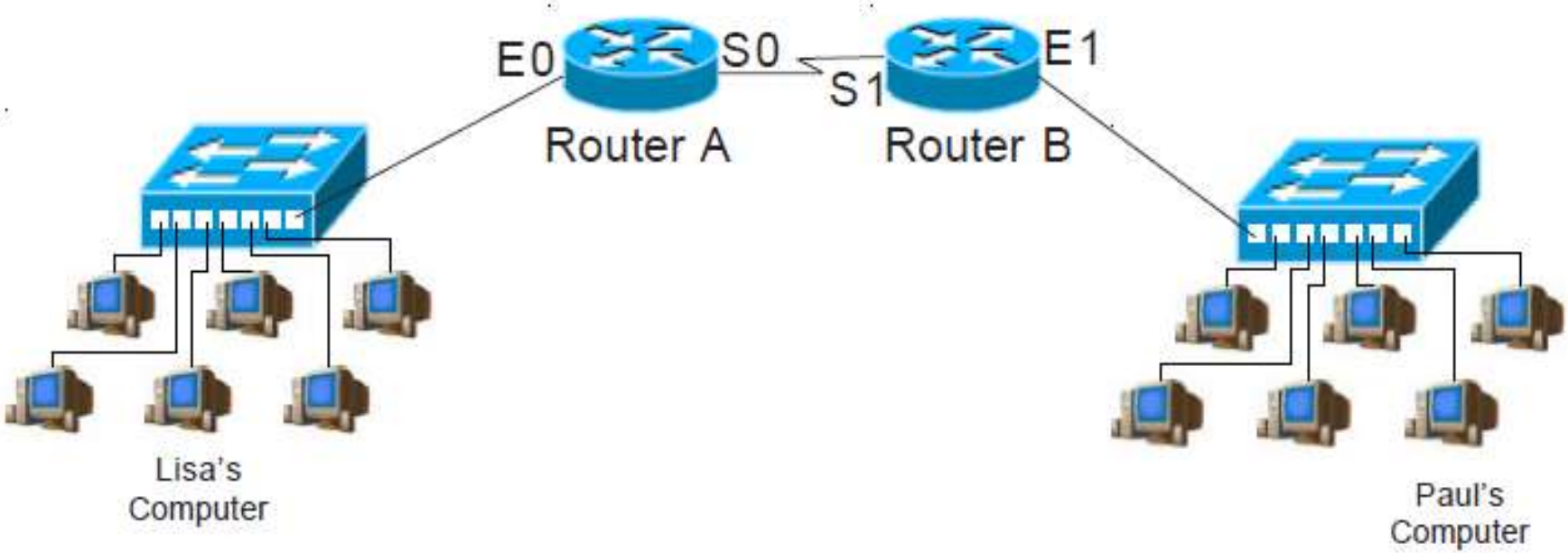



If you place the ACL on router A to block traffic to Router D it will also block all packets going to Routers B, and C; because all the packets will have the same source address.

# Standard Access List Placement Sample Problems




In order to permit packets from Juan's computer to arrive at Jan's computer you would place the standard access list at router interface FA1.




Lisa has been sending unnecessary information to Paul. Where would you place the standard ACL to deny all traffic from Lisa to Paul?
Router Name Router B Interface E1

Where would you place the standard ACL to deny traffic from Paul to Lisa?
Router Name Router A Interface E0

# Extended Access Lists

Extended Access Lists...

...are numbered from 100 to 199.

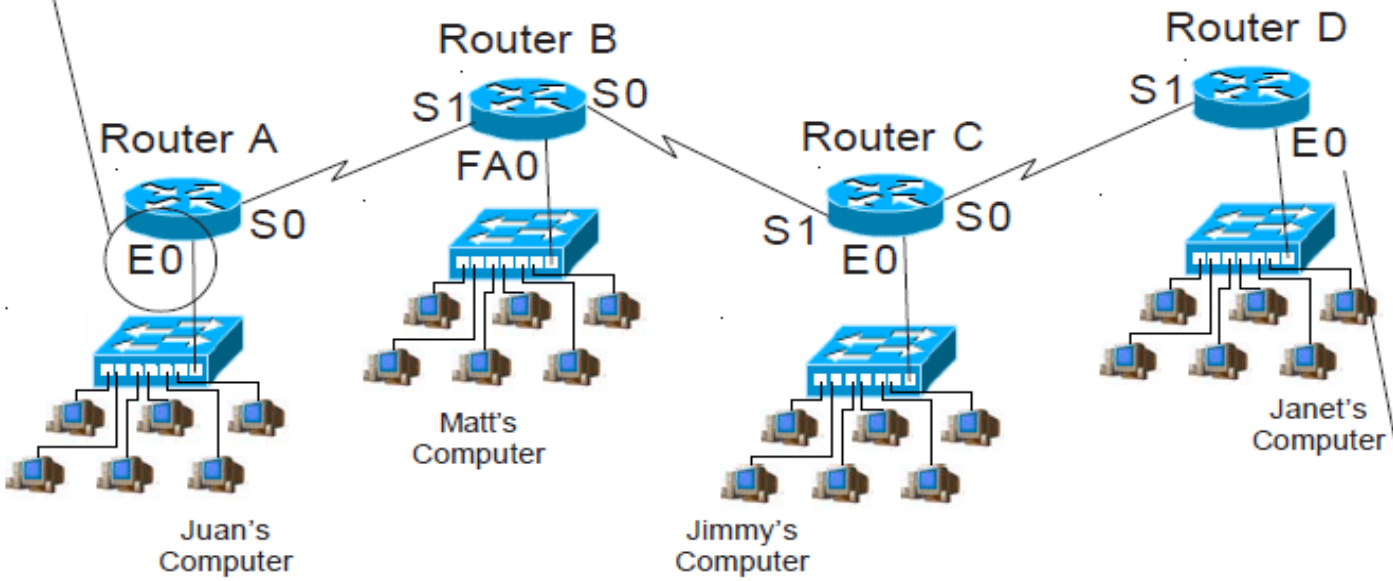
...filter (permit or deny) based on the:
- source address
- destination address
- protocol
- port number

... are placed close to the source.

...work at both layer 3 and 4 of the OSI model.

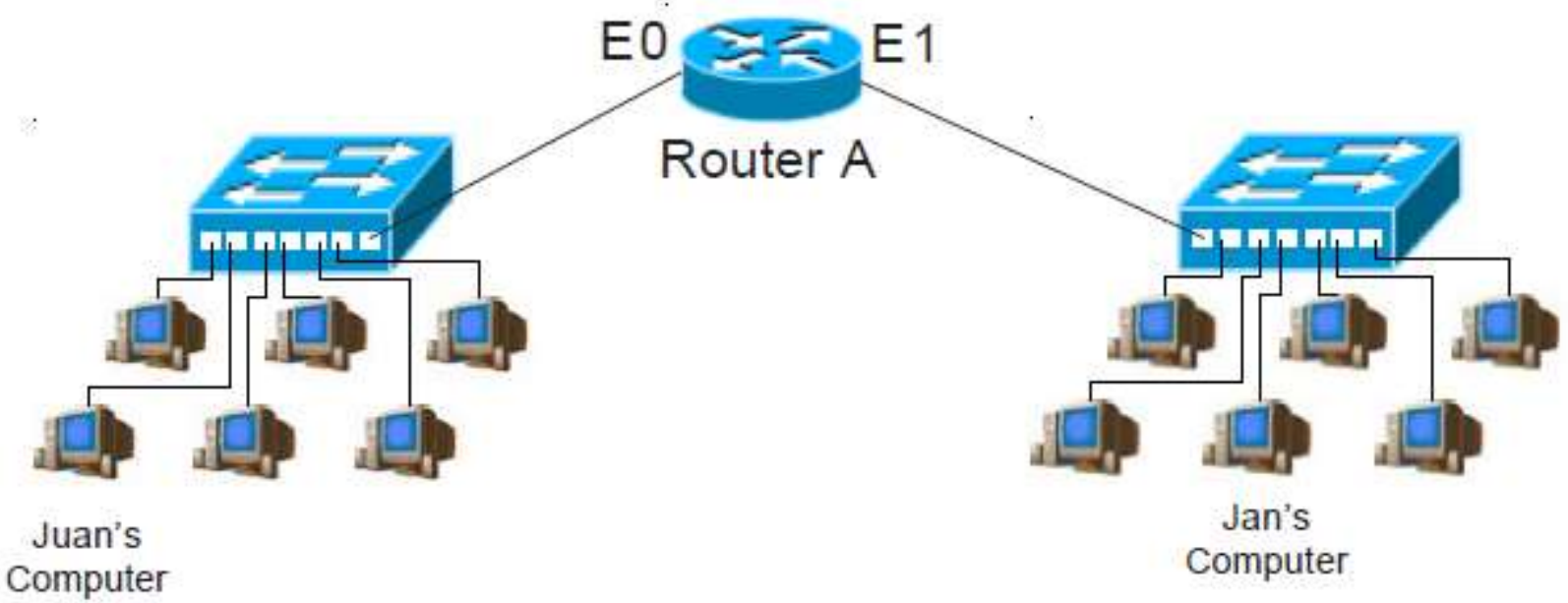
# Why extended ACLs are placed close to the source.

If you want to deny traffic from Juan's computer from reaching Janet's computer with an extended access list you would place the ACL **close to the source** on Router A, interface E0. Since it can permit or deny based on the destination address it can reduce backbone overhead and not effect traffic to Routers B, or C.

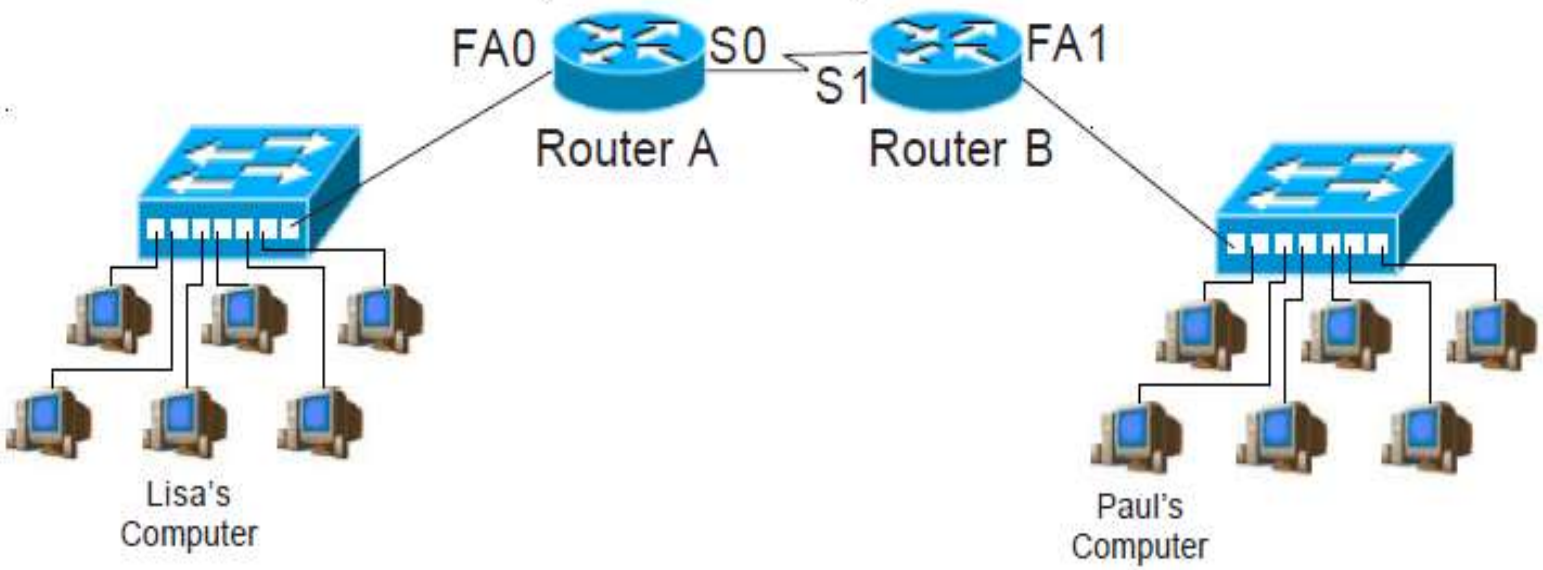



If you place the ACL on Router E to block traffic from Router A, it will work. However, Routers B, and C will have to route the packet before it is finally blocked at Router E. This increases the volume of useless network traffic.

# Extended Access List Placement Sample Problems




In order to permit packets from Juan's computer to arrive at Jan's computer you would place the extended access list at router interface E0.




Lisa has been sending unnecessary information to Paul. Where would you place the extended ACL to deny all traffic from Lisa to Paul?
Router Name Router A Interface FA0

Where would you place the extended ACL to deny traffic from Paul to Lisa?
Router Name Router B Interface FA1