

# LACNIC XIII

## Using BGP for Traffic Engineering in an ISP



# Program

- Using BGP Attributes
- Implementing IBGP
- Implementing EBGP
- Emphasis in Stability, Scalability and Configuration Examples



# BGP Review

Why use BGP?



# What we want to achieve

- Implement routing policies that are:
  - *Scalable*
  - *Stable*
  - *Simple*



# More Details ...

- You need to scale your IGP
- You are a client with two external connections
- You need to receive all Internet routes
- You need to implement a consistent routing policy or expand your QoS policy



# BGP Updates

Withdrawals

Attributes

Prefixes  
(NLRI - Network-Layer  
Reachability Information)



# BGP Attributes for Routing Policy Definition

- 1: ORIGIN
- 2: AS-PATH
- 3: NEXT-HOP
- 4: MED
- 5: LOCAL\_PREF
- 6: ATOMIC\_AGGREGATE
- 7: AGGREGATOR
- 8: COMMUNITY
- 9: ORIGINATOR\_ID
- 10: CLUSTER\_LIST
- 14: MP\_REACH\_NLRI
- 15: MP\_UNREACH\_NLRI



# External BGP (eBGP)

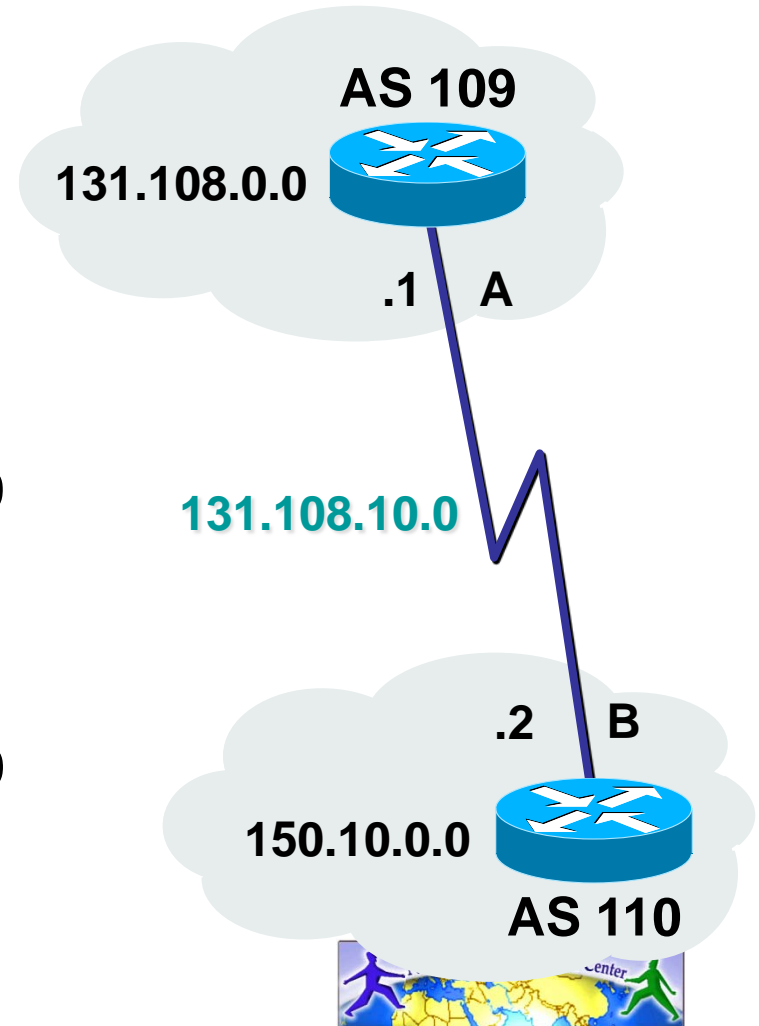
- Between routers in different ASNs
- Usually with a direct connexion
- With next-hop pointing to itself

- Router B

```
router bgp 110  
neighbor 131.108.10.1 remote-as 109
```

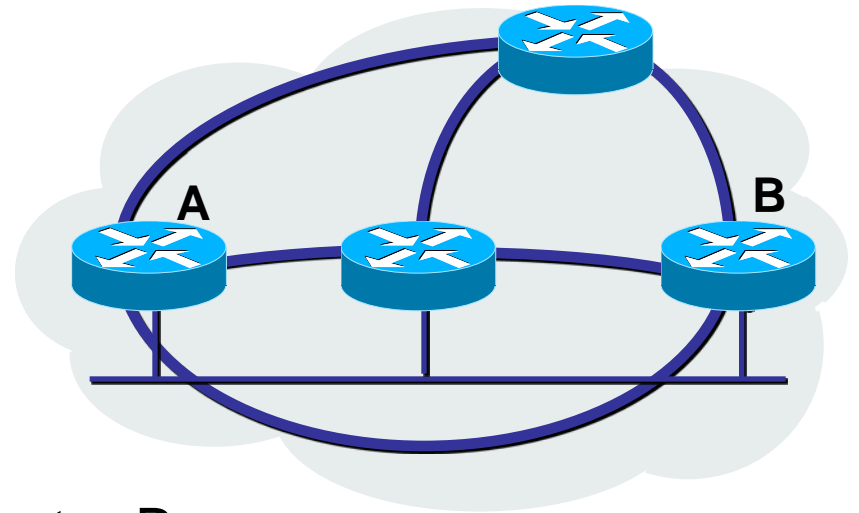
- Router A

```
router bgp 109  
neighbor 131.108.10.2 remote-as 110
```



# Internal BGP

- Neighbors within the same ASN
- Don't modify next-hop
- Not necessarily with a direct connection
- Do announce routes learned by other iBGP peers



Router B:

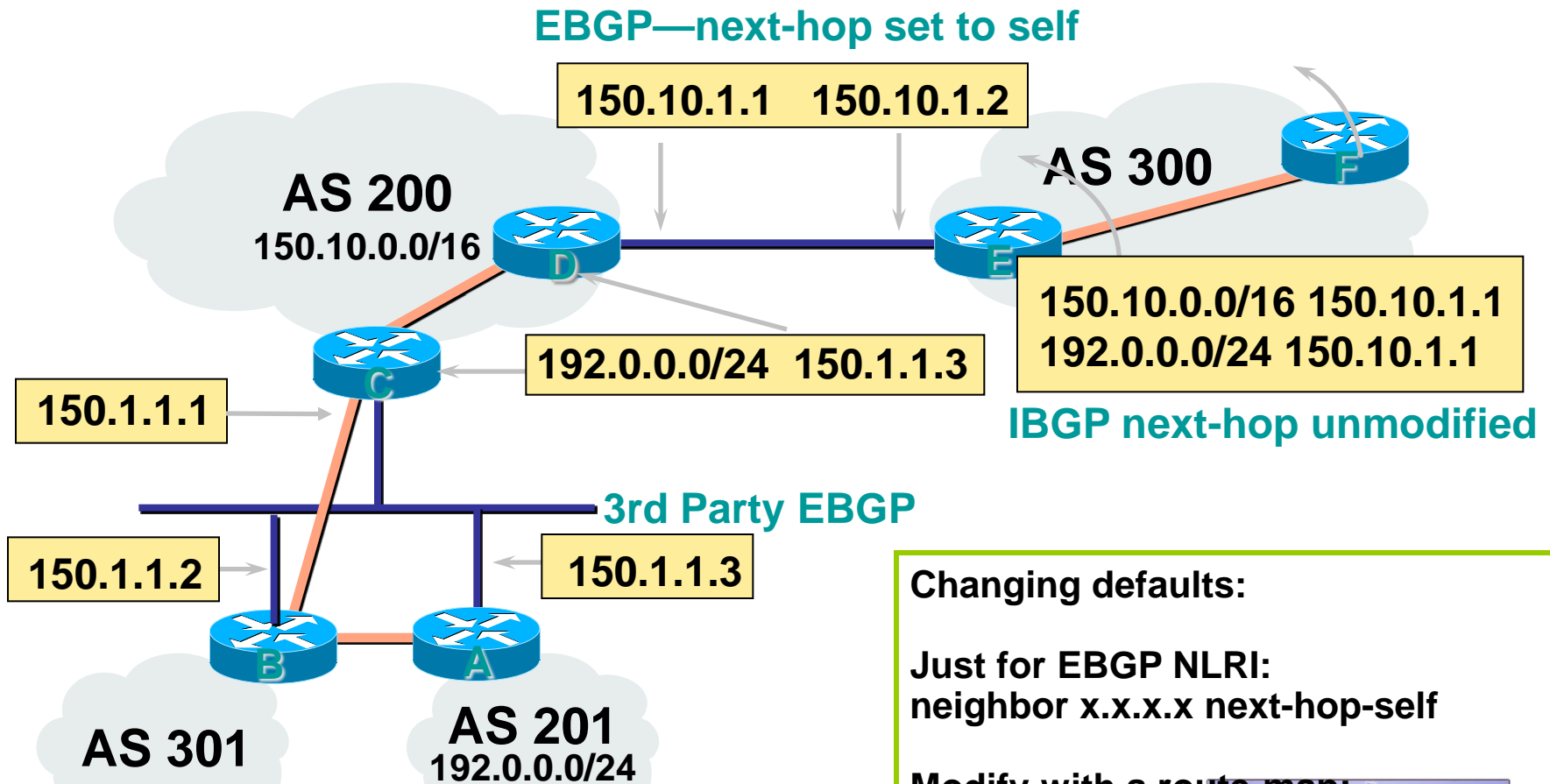
```
router bgp 109
neighbor 131.108.30.2 remote-as 109
```

Router A:

```
router bgp 109
neighbor 131.108.20.1 remote-as 109
```



# BGP Attributes: NEXT\_HOP



**Changing defaults:**

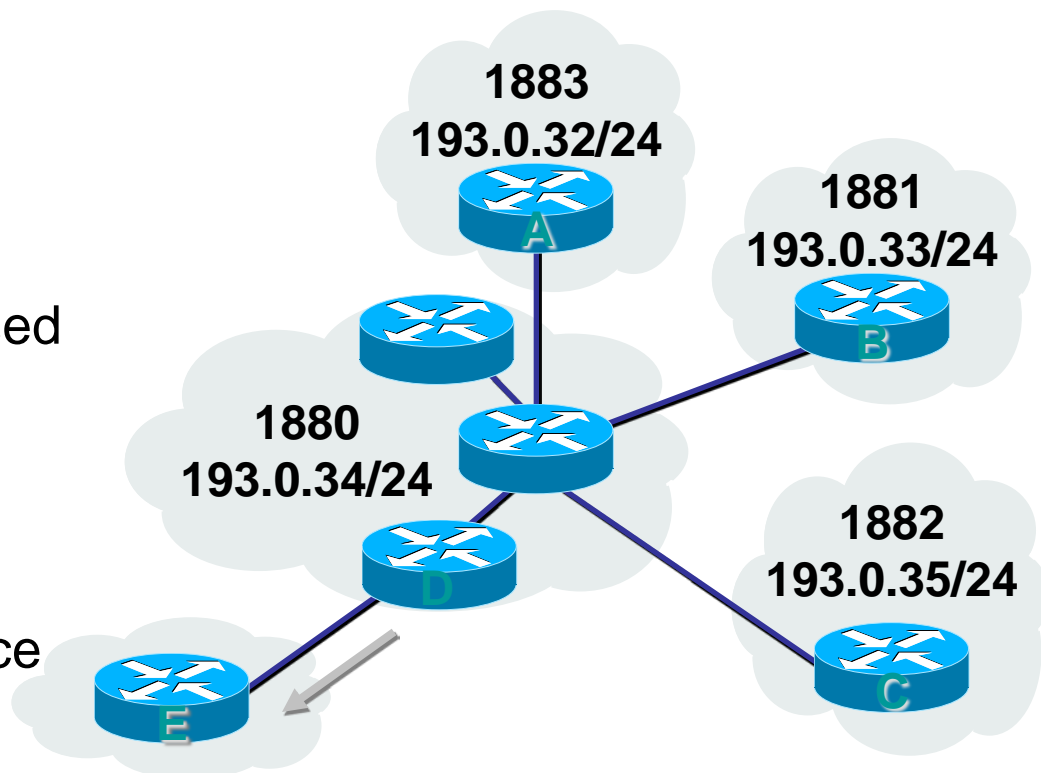
**Just for EBGP NLRI:**  
 neighbor x.x.x.x next-hop-self

**Modify with a route-map:**  
 set ip next-hop { A.B.C.D | peeraddress }

# Problem: Loop detection, Policies

## Solution: AS-PATH

- AS Sequence
  - List of ASN the advertisement has traversed
- AS Set
  - Summarizes an AS Sequence
  - The order in the sequence is lost
- Modify with route-map:  
*set as-path*

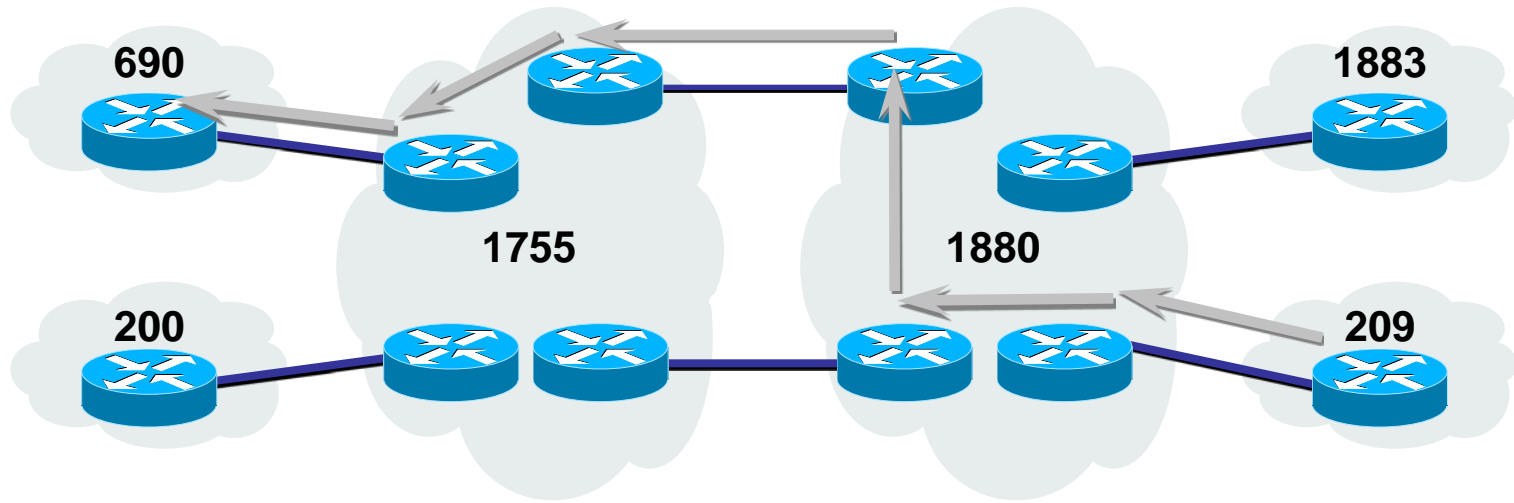


**A: 193.0.33/24 1880 1881**  
**B: 193.0.34/24 1880**  
**C: 193.0.32/24 1880 1883**  
**E: 193.0.32/22 1880 {1881, 1882, 1883}**



# Problem: Indicate the best path to an AS

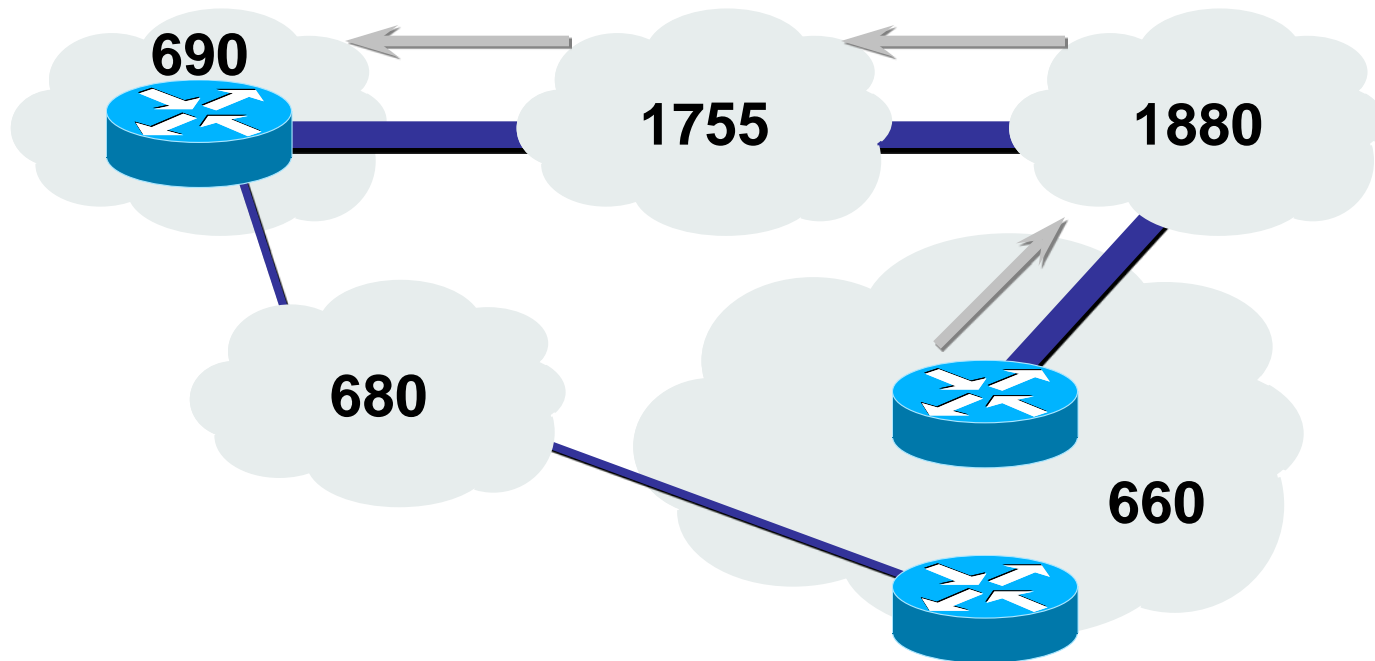
## Solution: MED



- Informs about an entry point preference
- Is compared if the path is to the same AS
  - Unless you use “bgp always-compare-med”
- Is a non-transitive attribute
- In a route-map: *set metric*



## Problem: Overriding MED/AS-PATH Solution: Local Preference

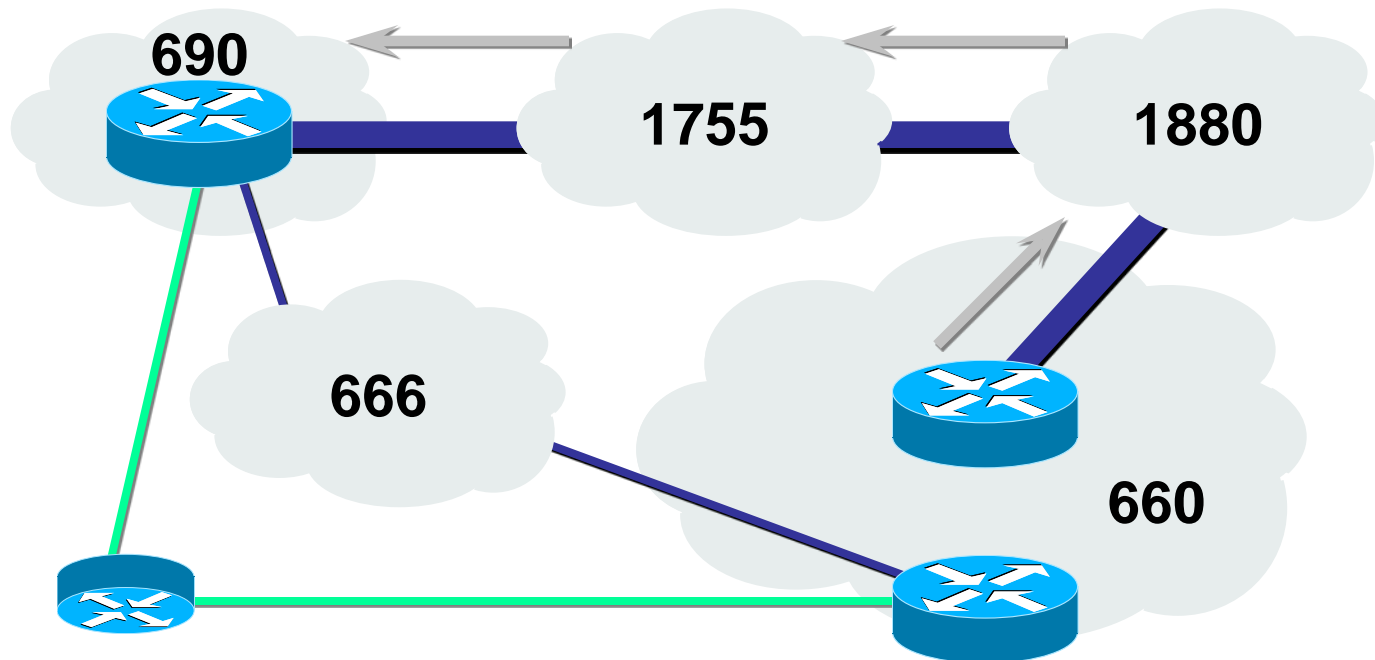


- Attribute is local to the AS – mandatory for iBGP updates
- route-map: *set local-preference*



# Problem: Overriding Local Preference

## Solution: Weight



- Local to the router where is configured
- route-map: *set weight*
- The highest weight wins over all the valid paths

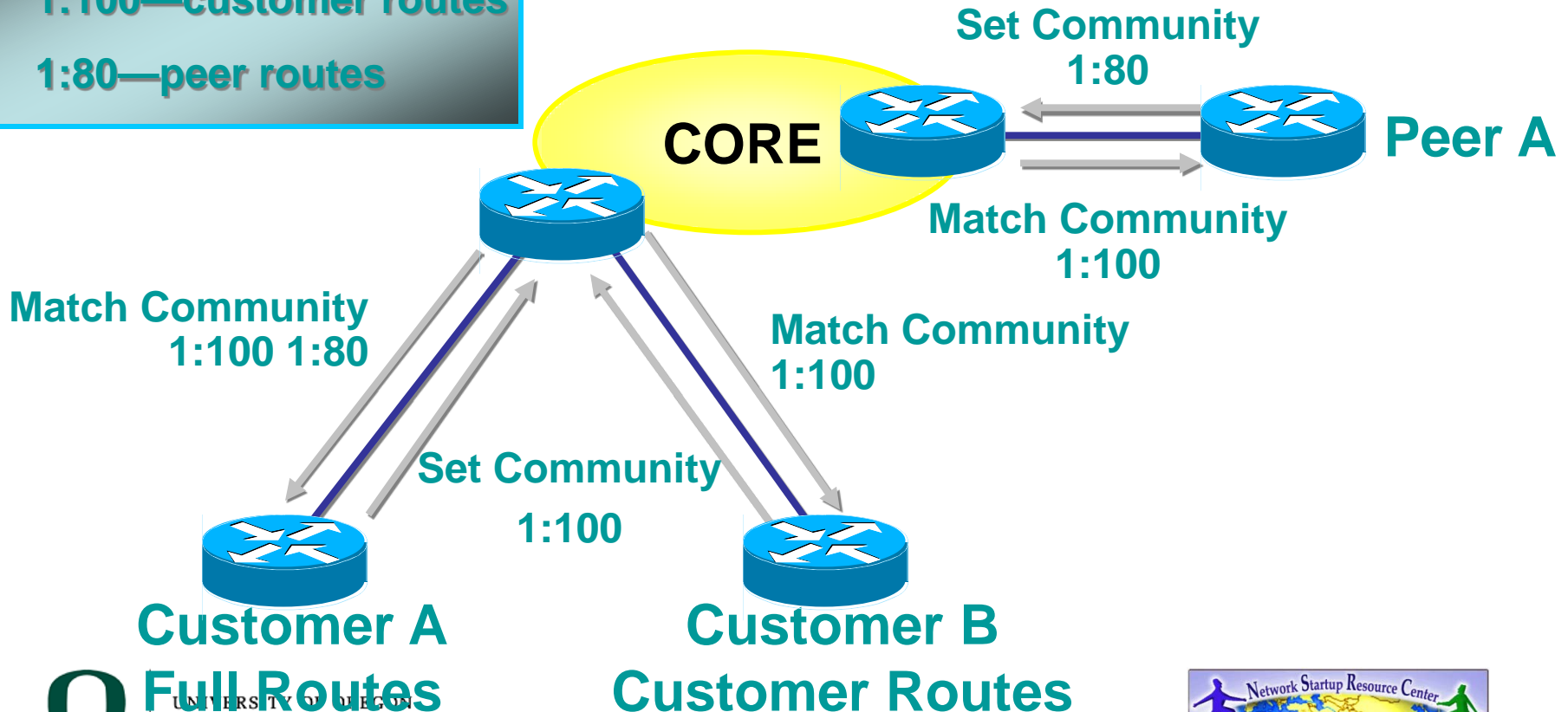
# Problem: Scaling Routing Policies

## Solution: COMMUNITY

### Communities:

1:100—customer routes

1:80—peer routes



# BGP Attributes: COMMUNITY

- Groups destinations to help scale policy application
- Typical Communities:
  - Prefixes learned from customers
  - Prefixes learned from peers
  - Prefixes in a VPN
  - Prefixes with preferential treatment in queuing



# BGP Attributes: COMMUNITY

- Activated per neighbor/peer-group:
  - *neighbor {peer-address | peer-group-name} send-community*
- Transitive across AS boundaries
- Common format is a 4-bytes string `<AS>:[0-65536]`



# BGP Attributes: COMMUNITY

- Each prefix can be a member of several communities
- Route-map: *set community*
  - <1-4294967295> community number
  - aa:nn community number in aa:nn format
  - additive* *Adds to a list of existing communities*
  - local-AS* *Do not send to EBGP neighbors (well-known community)*
  - no-advertise* *Do not send to any peers (well-known community)*
  - no-expert* *Do not export outside of the AS/Confederation (well-known community)*
  - *community*
  - none* *No community attribute*



# Least Used Attribute: ORIGIN

- IGP – created with a *network* command in the BGP configuration
- EGP – redistributed from an EGP
- Incomplete – redistributed from an IGP in the BGP configuration
- **NOTE** – always use a route-map to modify the origin: *set origin igp*



# Set command in a route-map

- as-path                      Prepends a string of AS to the AS-PATH attribute
- comm-list                    Sets BGP community list (for deletion)
- community                   BGP community attribute
- dampening                   Sets BGP dampening formeters
- local-preference            BGP local preference attribute
- metric                        Metric value for the destination routing protocol
- origin                        BGP origin code
- weight                       BGP weight for routing table
- ip next-hop                 { A.B.C.D | peer-address }



# Atributos de BGP

```
router1#sh ip bgp 10.0.0.0
```

```
BGP routing table entry for 10.0.0.0/24, version 139267814
```

```
Paths: (1 available, best #1)
```

```
Not advertised to any peer
```

```
! AS-PATH
```

```
AS ID
```

```
65000 64000 {100 200}, (aggregated by 64000 16.0.0.2)
```

```
! NEXT-HOP
```

```
IGP METRIC
```

```
PEER-IP
```

```
PEER-ID
```

```
10.0.10.4 (metric 10) from 10.0.0.1 (10.0.0.2)
```

```
Origin IGP, metric 100, localpref 230, valid, aggregated  
internal (or external or local),
```

```
atomic-aggregate, best
```

```
Community: 64000:3 100:0 200:10
```

```
Originator: 10.0.0.1, Cluster list: 16.0.0.4, 16.0.0.14
```



# Decision Algorithm

Only consider synchronized routes without AS loops and a valid next-hop, and then prefer:

Highest WEIGHT

Highest LOCAL PREFERENCE

LOCALLY ORIGINATED (eg network/aggregate)

Shortest AS-PATH

Lowest ORIGIN (IGP < EGP < incomplete)

Lowest MED

EBGP

IBGP

Lowest IGP METRIC to next-hop

Oldest external path

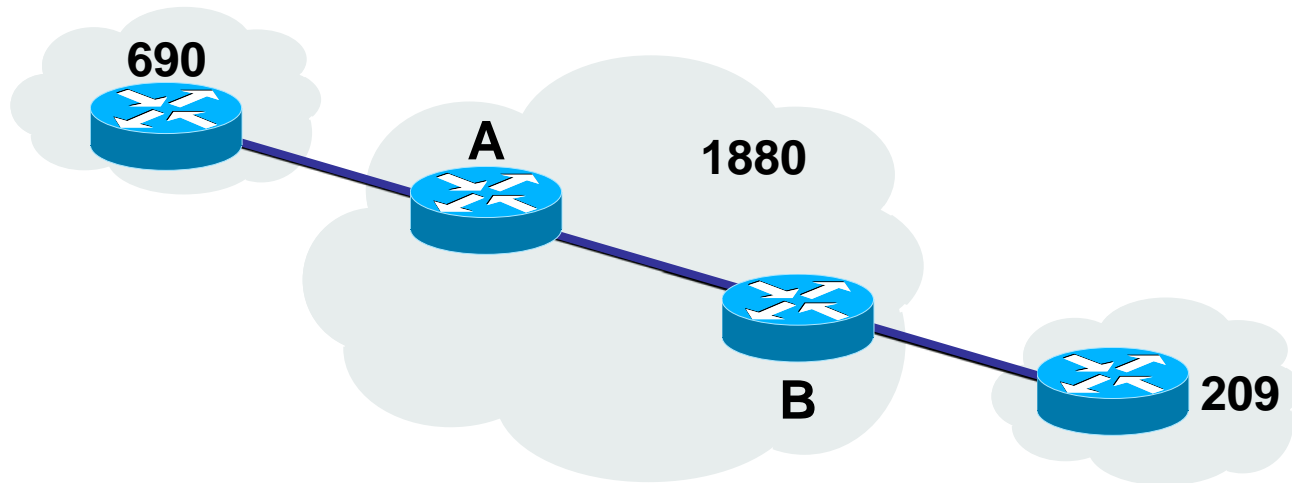
Router with lowest Router ID

Shortest CLUSTER\_LIST length

Lowest Neighbor IP address



# Synchronization



- Router A won't announce prefixes to AS209 until its IGP has converged
- Make sure that iBGP next-hops are reachable via the IGP, and then:

```
router bgp 1880  
no synchronization
```



# General Considerations

- Synchronization is not required if you have a full iBGP mesh
- Don't let BGP override your IGP
- auto-summary: avoid. Instead use aggregation commands:

```
router bgp 100  
no synchronization  
no auto-summary  
distance 200 200 200
```



# Until now ...

- We can apply policies on a per AS basis
- Can group prefixes using communities
- Can chose entry and exit points for large policy groups using MED and local preference attributes

## But, can the policies scale?



# Implementing iBGP

## Route Reflectors, Peer Groups



# Guidelines for a Stable iBGP

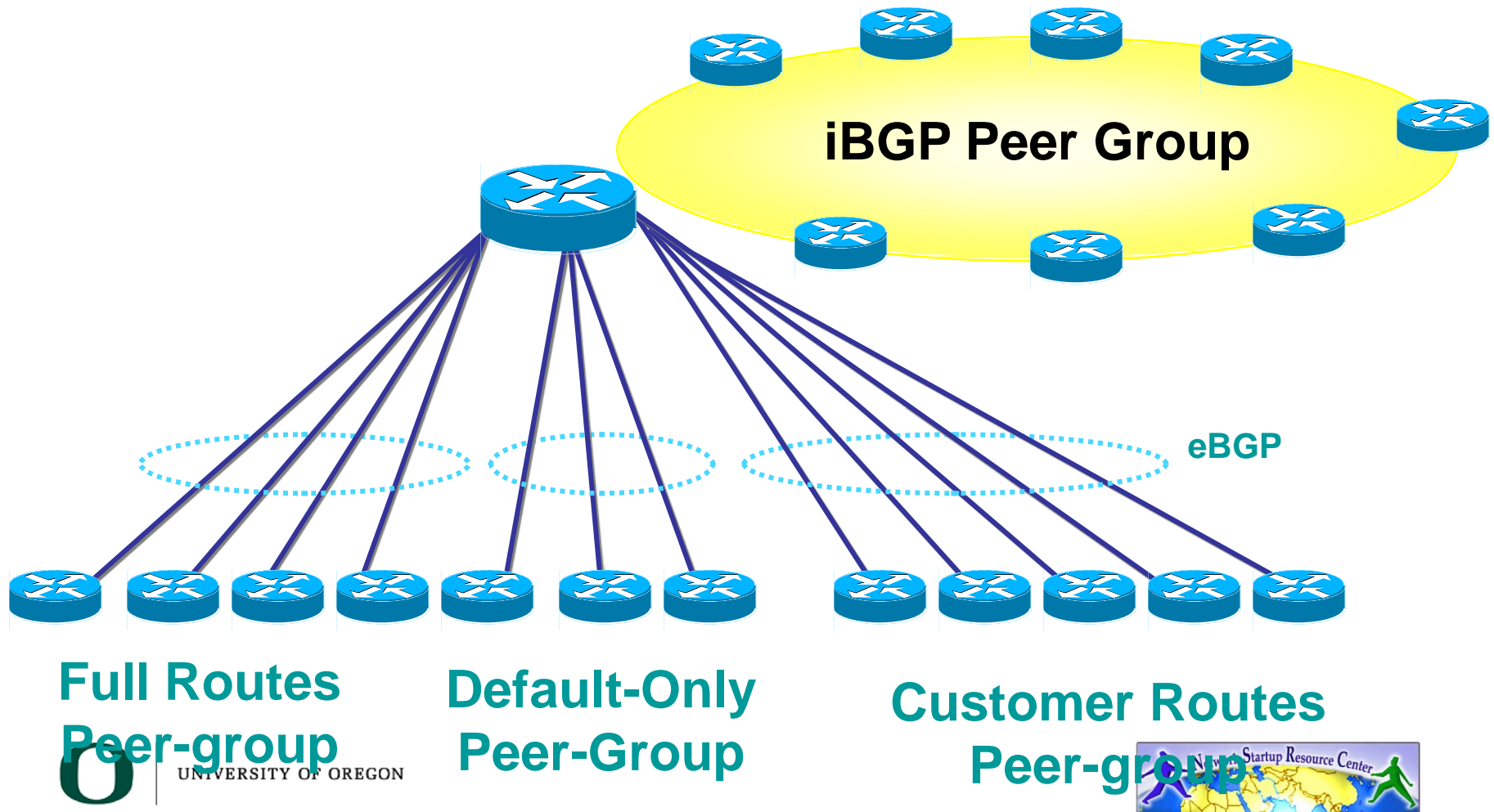
- Peer using the loopback address
  - neighbor { ip address | peer-group }  
update-source loopback0
- Independent from physical interface failures
- Takes advantage of any IGP load-sharing

# Guidelines for a Stable iBGP

- Use *peer-group* and *route-reflector*
- Only carry the next-hops in the IGP
- Carry full routes in BGP if it is necessary
- DO NO redistribute BGP into IGP



# Using *Peer-Groups*



# What is a *peer-group*?

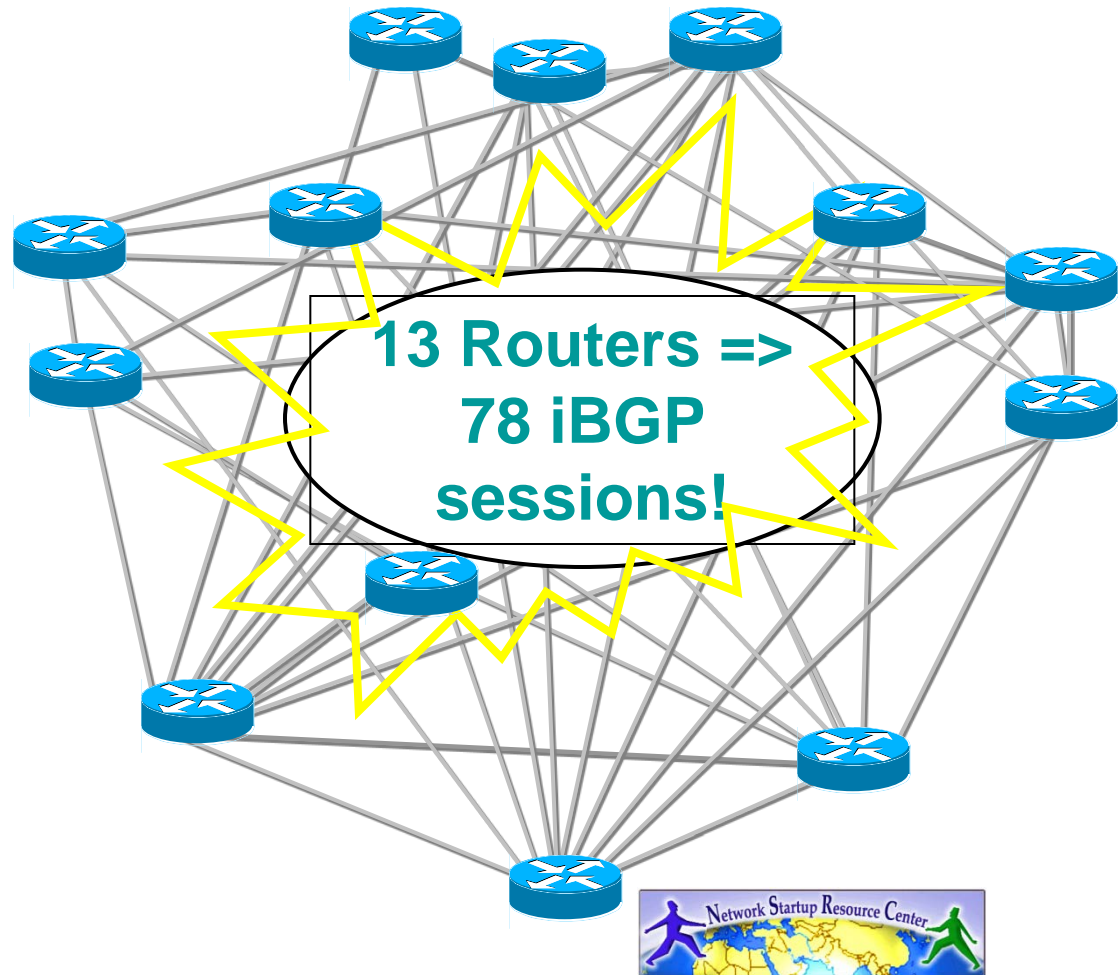
- All members of a peer-group have a common outbound policy
- Updates are generated only once per peer-group
- Simplifies configuration
- Members can have different inbound policies



# Why use a Route-reflector?

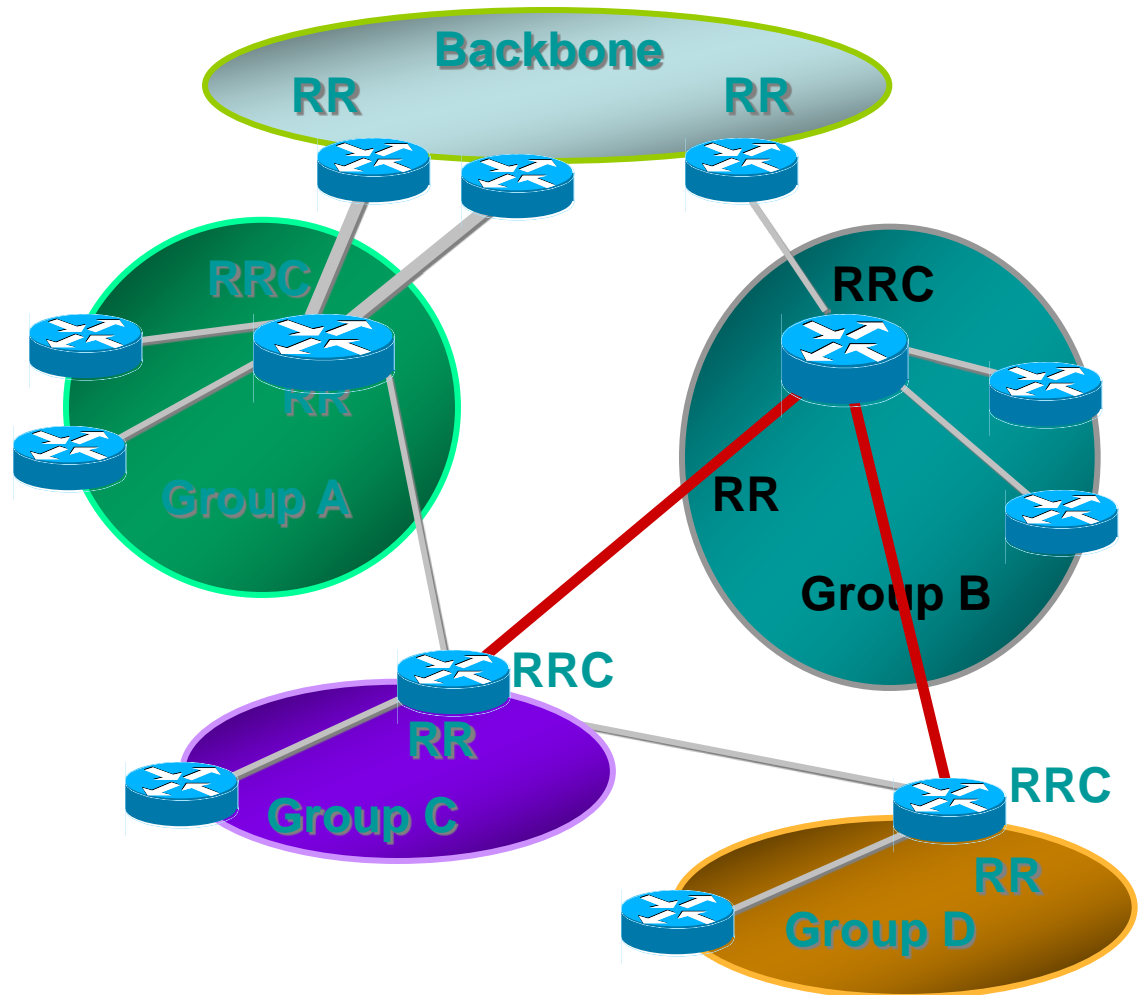
To avoid having a full mesh with  $N(n-1)/2$  sessions

$n=1000 \Rightarrow$  almost half a million iBGP sessions!



# Using *Route-Reflectors*

Rule for RR Loop Avoidance: RR topology should follow the physical topology



# What is a *Route-Reflector*?

- The reflector receives path updates from clients and non-clients
- If the path is from a client, reflect it to clients and non-clients
- If the best path is from a non-client, reflect it only to the clients



# Deploying *Route-Reflectors*

- Split the backbone into different groups
- Each group contains at least one RR (multiple for redundancy), and multiple clients
- Build a iBGP full mesh for the RRs
- Utilize single IGP - next-hop is not modified by the RR



# Hierarchical *Route-Reflector*

- Example:

```
RouterB>sh ip bgp 198.10.0.0
```

```
BGP routing table entry for 198.10.10.0/24
```

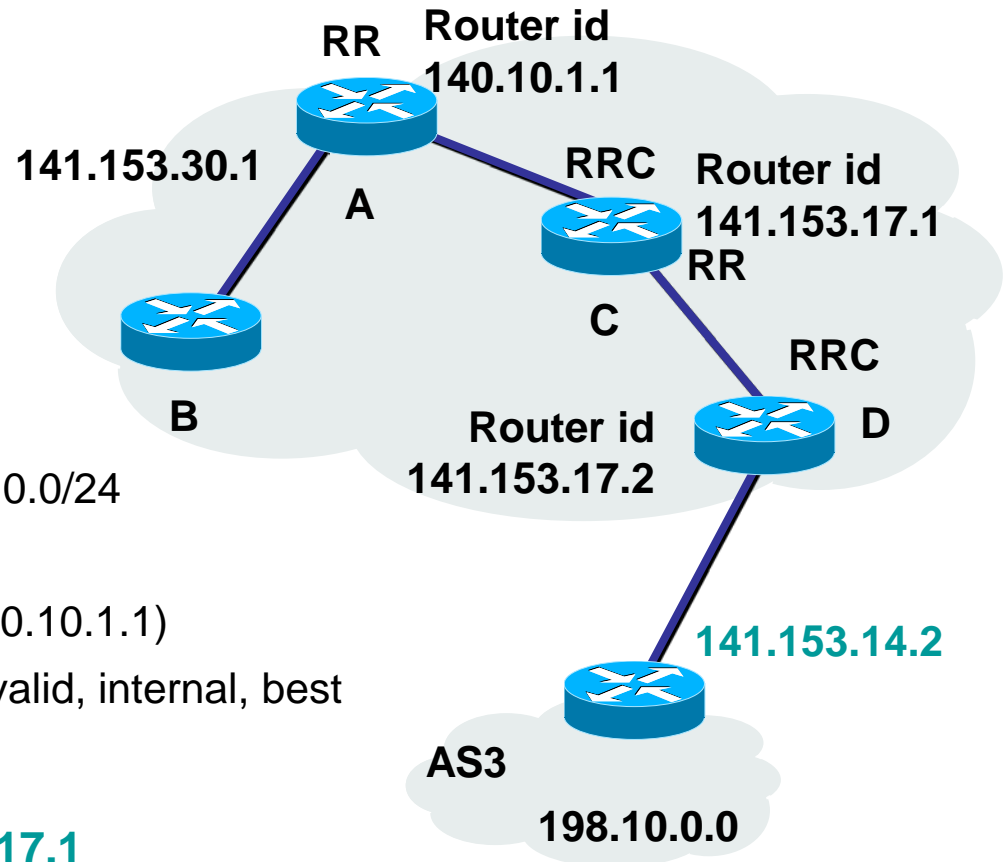
```
3
```

```
141.153.14.2 from 141.153.30.1 (140.10.1.1)
```

```
Origin IGP, metric 0, localpref 100, valid, internal, best
```

**Originator: 141.153.17.2**

**Cluster list: 144.10.1.1, 141.153.17.1**



# BGP Attributes: ORIGINATOR\_ID

- ORIGINATOR\_ID
  - Router ID of iBGP speaker that reflects the RR client routes to non-clients
  - Overriden by: *bgp cluster-id x.x.x.x*
- Useful for troubleshooting and loop detection



# BGP Attributes: CLUSTER\_LIST

- CLUSTER\_LIST
  - String of ORIGINATOR\_IDs through which the prefix has traversed
- Useful for troubleshooting and loop detection



# Until now ...

- Is the iBGP peering **S**table?
  - Use of loopbacks for the connection
- Will it **S**cale?
  - Use *peer-groups*
  - Use *route-reflectors*
- **S**imple, hierarchical configuration?



# Deploying eBGP

## Customer & ISP Issues



# Customer Issues

- Procedure
  - Configure BGP (use session passwords!)
  - Generate a stable aggregate route
  - Configure Inbound Policy
  - Configure Outbound Policy
  - Configure loadsharing/multihoming



# Connecting to an ISP

- **AS 100 is a customer of AS 200**
- **Usually with a direct connection**

Router B:

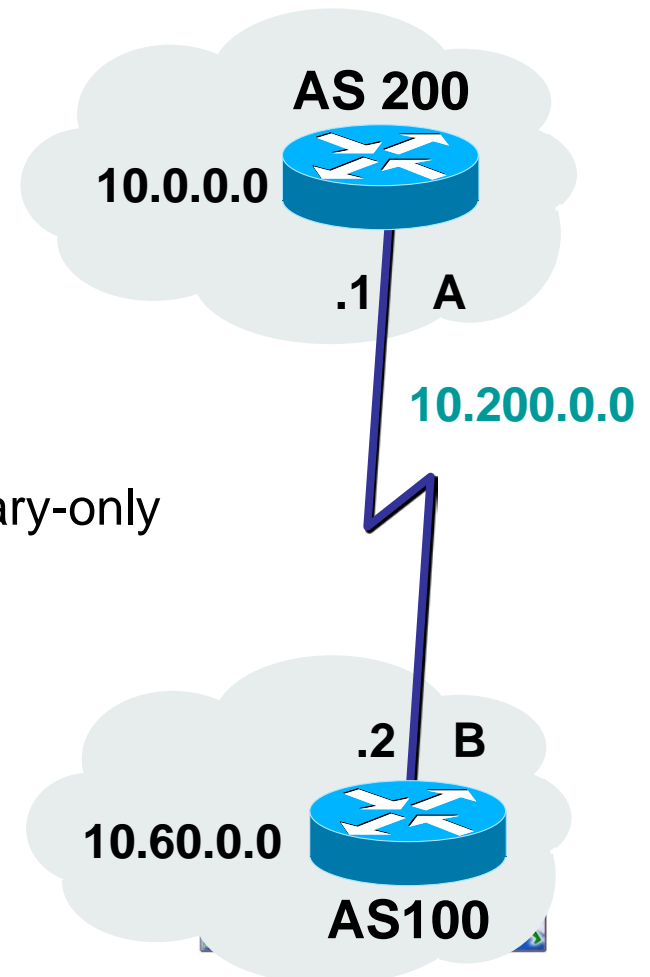
```
router bgp 100
```

```
  aggregate-address 10.60.0.0 255.255.0.0 summary-only
```

```
  neighbor 10.200.0.1 remote-as 200
```

```
  neighbor 10.200.0.1 route-map isp-out out
```

```
  neighbor 10.200.0.1 route-map isp-in in
```



# What is Aggregation?

- Summarization based on specific routes *from the BGP routing tables*
  - 10.1.1.0 255.255.255.0
  - 10.2.0.0 255.255.0.0
  - => 10.0.0.0 255.0.0.0



# How to Aggregate?

- `aggregate-address 10.0.0.0 255.0.0.0 {as-set} {summary-only} {route-map}`
- Use `as-set` to include path and community information from specific routes
- *summary-only* suppresses specific routes
- Use `route-map` to configure other attributes



# Why Aggregate?

- Reduce the number of prefixes to announce
- Increase stability — aggregate routes are maintained even when specifics disappear
- How to generate stable aggregates:
  - router bgp 100
  - aggregate-address 10.0.0.0 255.0.0.0 as-set summary-only
  - network 10.1.0.0 255.255.0.0
  - :
  - ip route 10.1.0.0 255.255.0.0 null0



## BGP Attributes: ATOMIC\_AGGREGATE

- Indicates the loss of AS-PATH information
- Must not be removed once configured
- Configuration: *aggregate-address x.x.x.x*
- Is not set if the *as-set* keyword is used, however, *AS-SET* and *COMMUNITY* then carry information about the specifics



# BGP Attributes: AGGREGATOR

- AS number and IP of router generating the aggregate
- Useful for troubleshooting



# Attributes of the Aggregate

- NEXT\_HOP = local (0.0.0.0)
- WEIGHT = 32768
- LOCAL\_PREF = none (assumes 100)
- AS\_PATH = AS\_SET or nothing
- ORIGIN = IGP
- MED = none



# Why an Inbound Policy?

- So we can apply a recognizable COMMUNITY that can be used in outbound filters and other policies
- Configure local-preference to override the default of 100
- Multihoming loadsharing
- Example:  
route-map isp-in permit 10  
set local-preference 200  
set community 100:2



# Why an Outbound Policy?

- Outbound prefix filters help protect against errors (can also apply as-path and community filters)
- Send communities based on agreements with ISP
- Example

```
route-map isp-out permit 10  
  match ip address prefix-list outgoing  
  set community 100:1 additive
```



# Load-Sharing – One Path

Router A:

```
interface loopback 0
```

```
  ip address 10.60.0.1 255.255.255.255
```

```
!
```

```
router bgp 100
```

```
  neighbor 10.200.0.2 remote-as 200
```

```
  neighbor 10.200.0.2 update-source loopback0
```

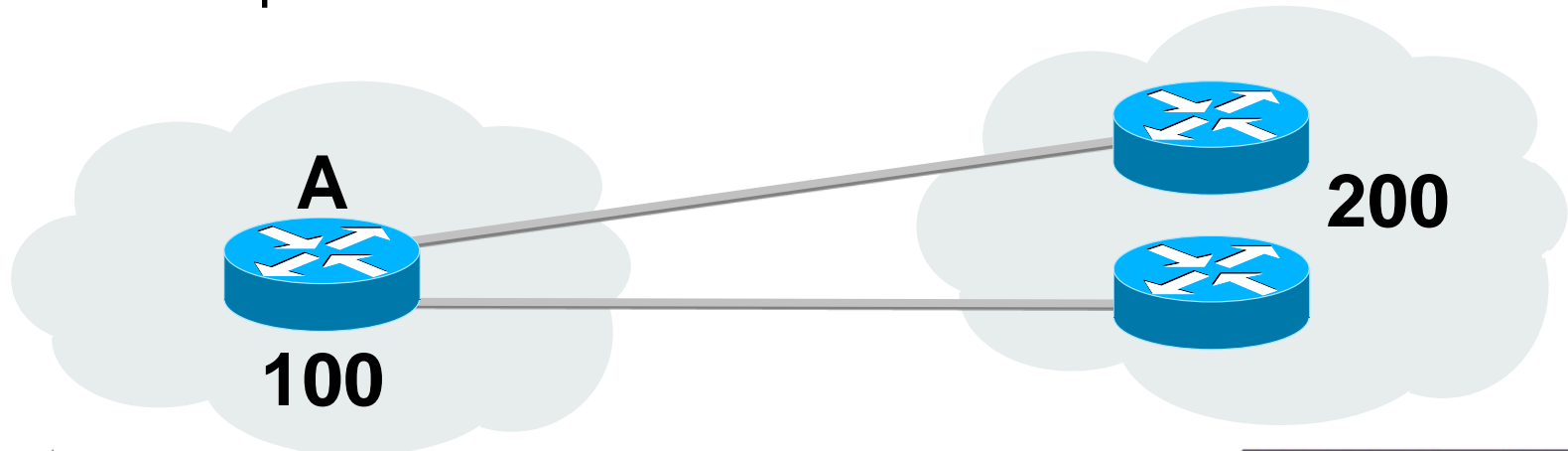
```
  neighbor 10.200.0.2 ebgp-multi-hop 2
```



# Load-sharing – Multiple Paths/Same AS

Router A:

```
router bgp 100  
neighbor 10.200.0.1 remote-as 200  
neighbor 10.300.0.1 remote-as 200  
maximum-paths 6
```



# What is Multihoming?

- Connecting to two or more ISPs to increase:
  - **Reliability** – if one ISP fails, still have others
  - **Performance** – better paths to common Internet destinations



# Types of Multihoming

- Three common cases:
  - Default route from all providers
  - Customer plus Default from all providers
  - Full routes from all providers

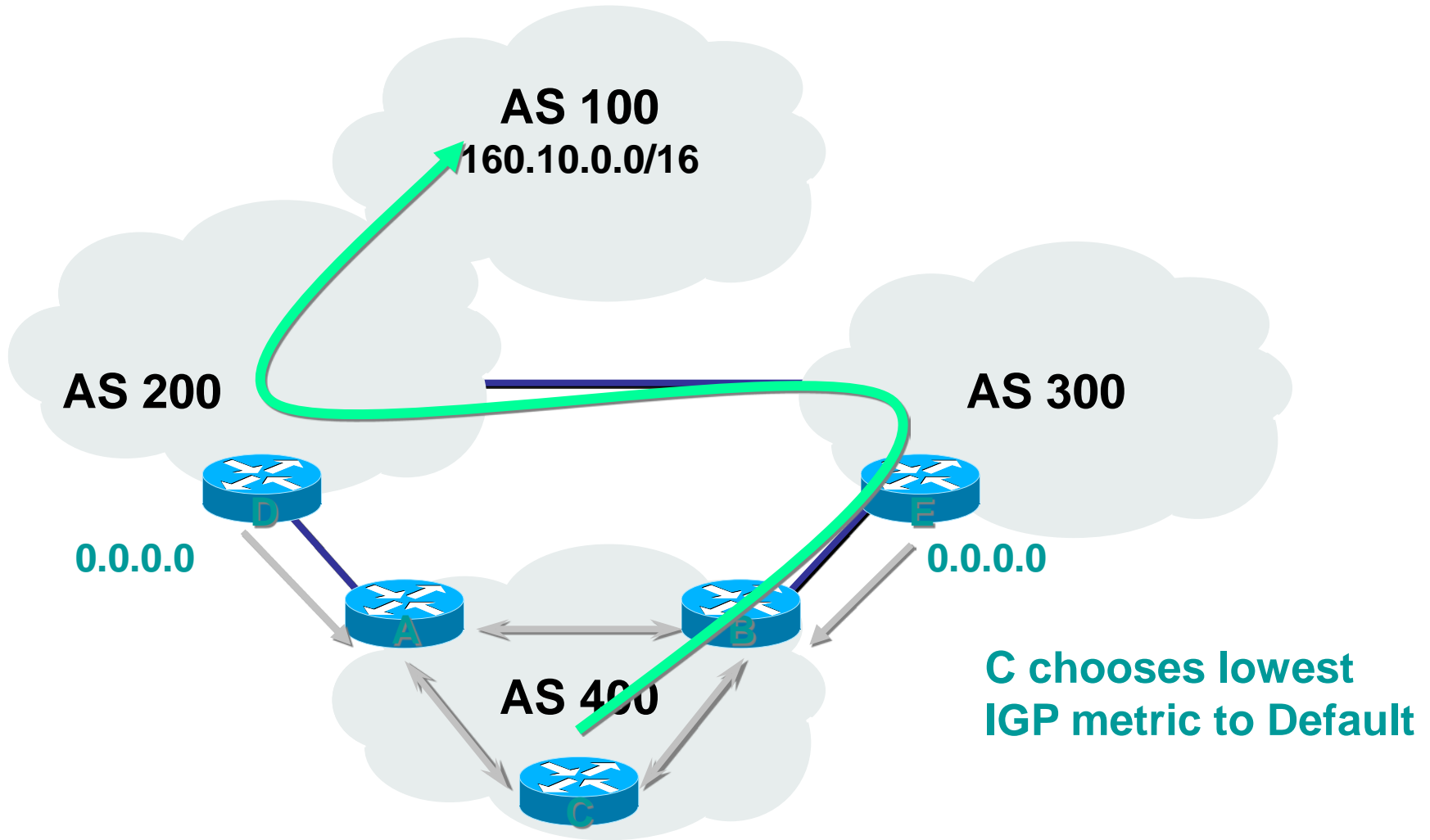


# Default Route from All Providers

- Low memory and CPU requirements
- Provider sends BGP default => provider decides based on IGP metrics to reach default
- You send all your routes to the provider => inbound path decided by Internet
  - You can influence using AS-PATH prepend



# Default Route from All Providers

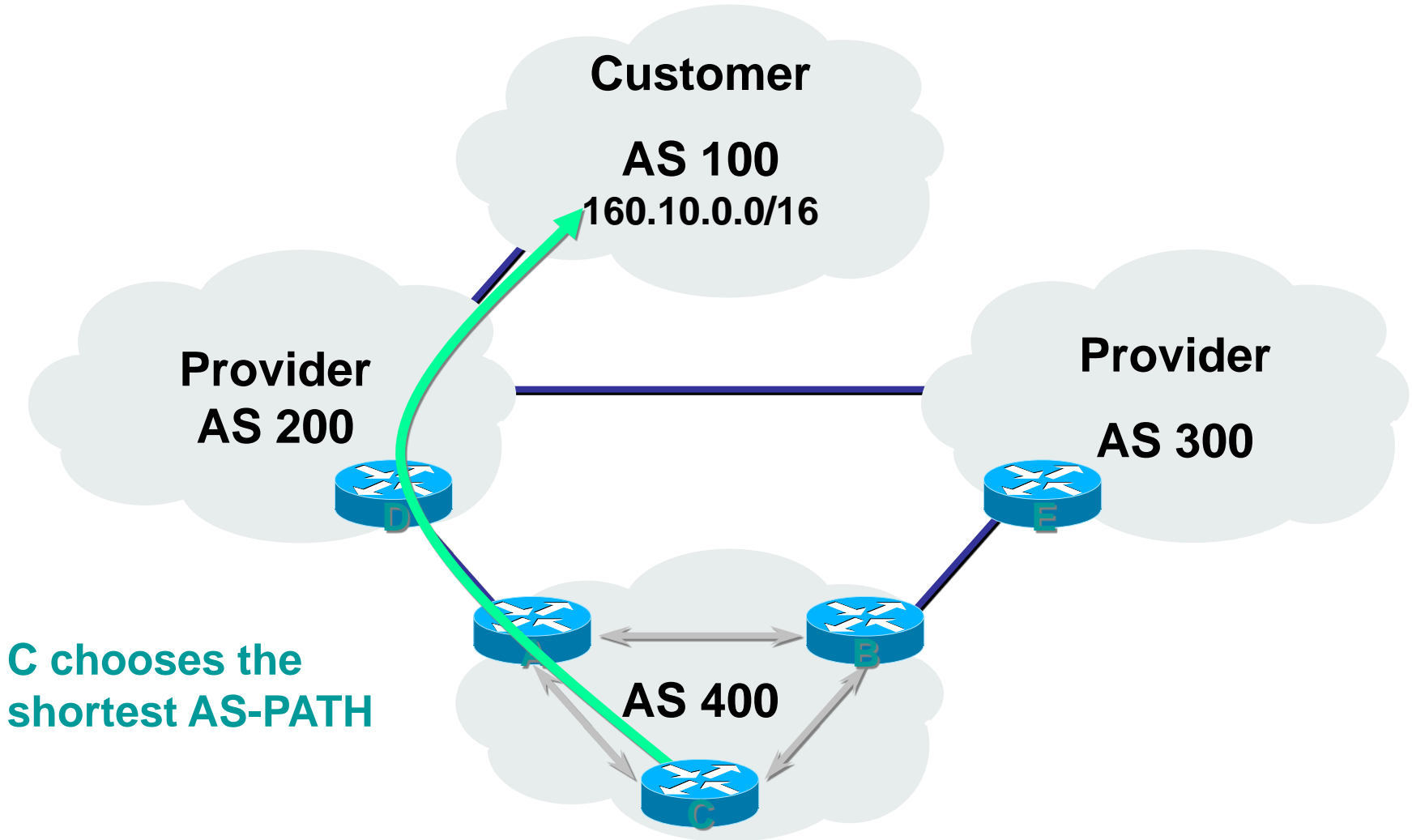


# Customer+Default from All Providers

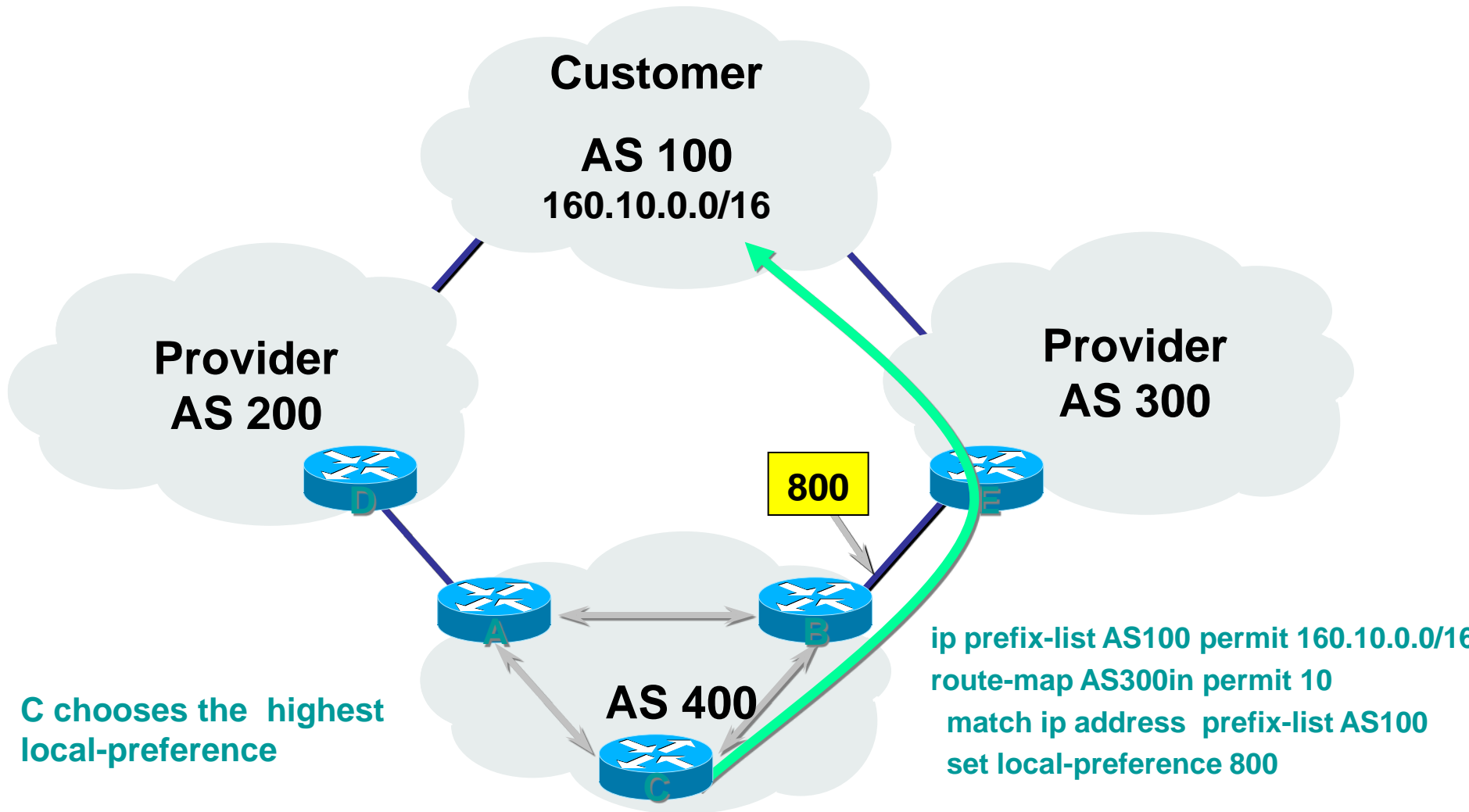
- Medium memory/CPU requirements
- “Best” path – usually the shortest AS-PATH
- Use local-preference to override based on prefix, as-path, or community
- IGP metric to default used for all other destinations



# Customer+Default from All Providers



# Customer Routes from All Providers

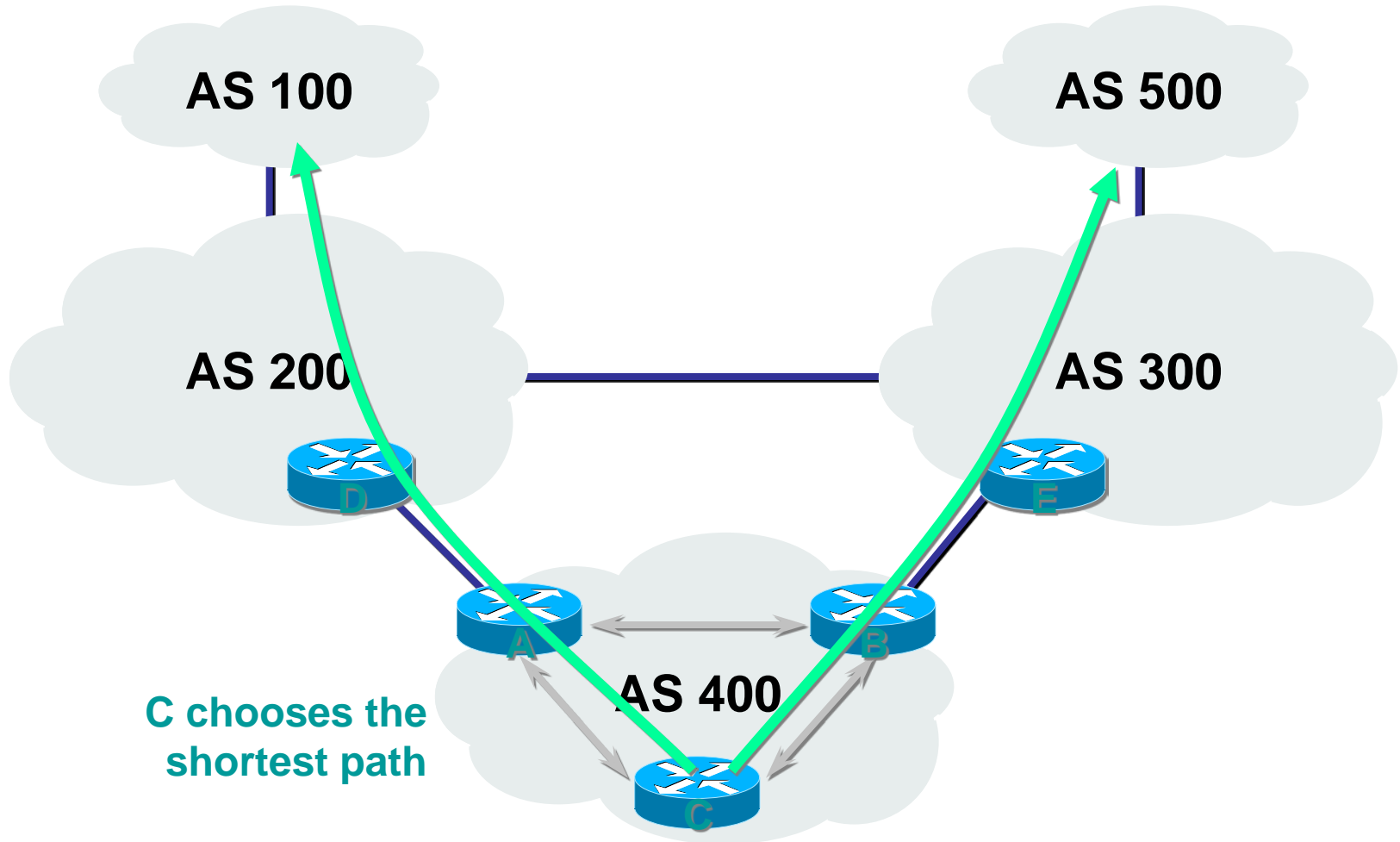


# Full Routes from All Providers

- Higher memory/CPU requirements
- Reach all destinations based in the “best” path – usually the one with the shortest path
- Still can adjust manually using local-preference and comparing as-path, communities and prefix-lists



# Full Routes from All Providers



# Controlling Inbound Traffic?

- Controlling inbound traffic is very difficult due to lack of a transitive metric
- You can split your prefix announcements among the providers, but then, what happens to redundancy?

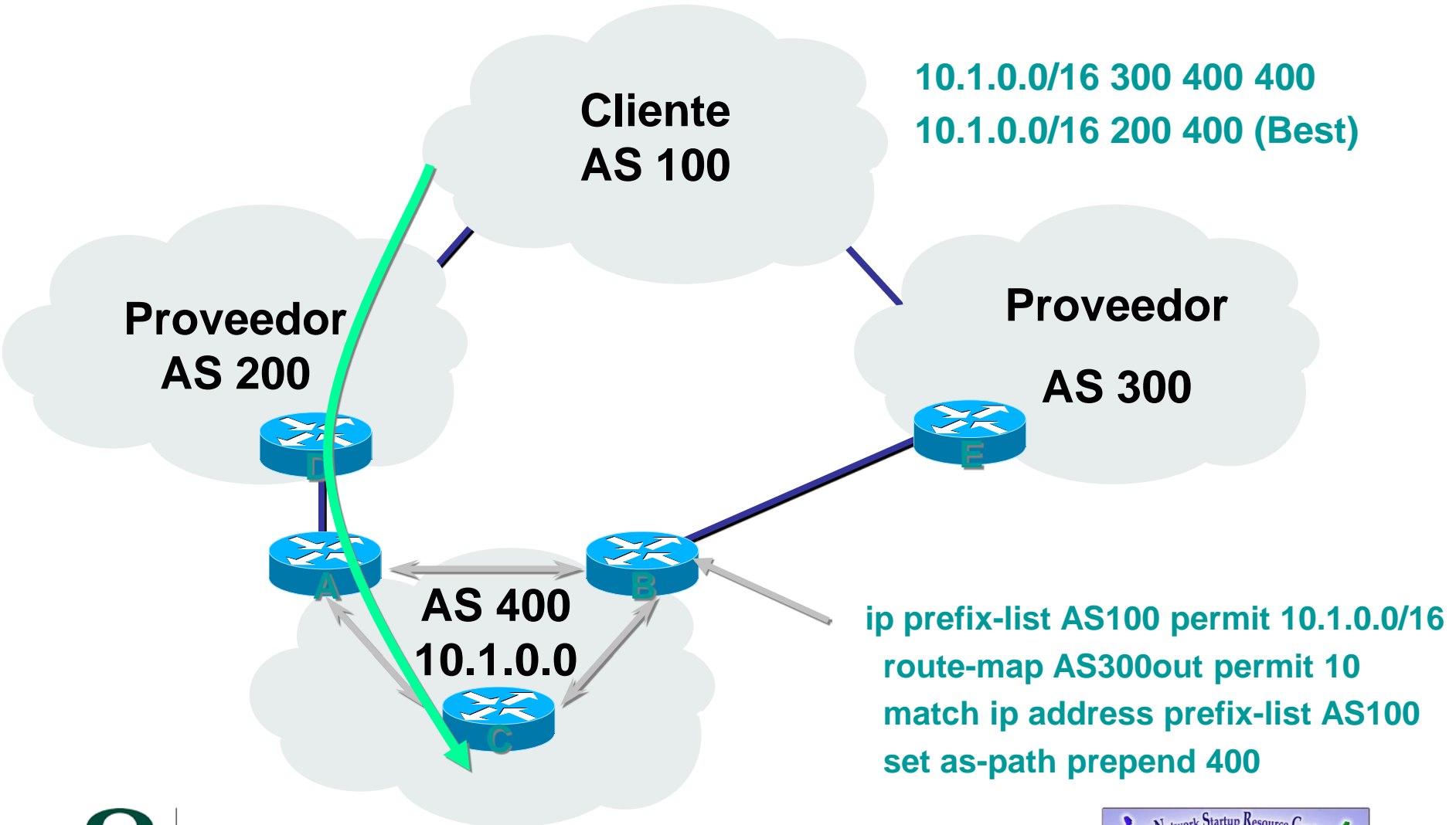


# Controlling Inbound Traffic?

- **Bad Internet Citizen:**
  - Splits the address space
  - Uses “as-path prepend”
  
- **Good Internet Citizen:**
  - Splits address space
  - Uses “advertise maps”



# Using “AS-PATH prepend”

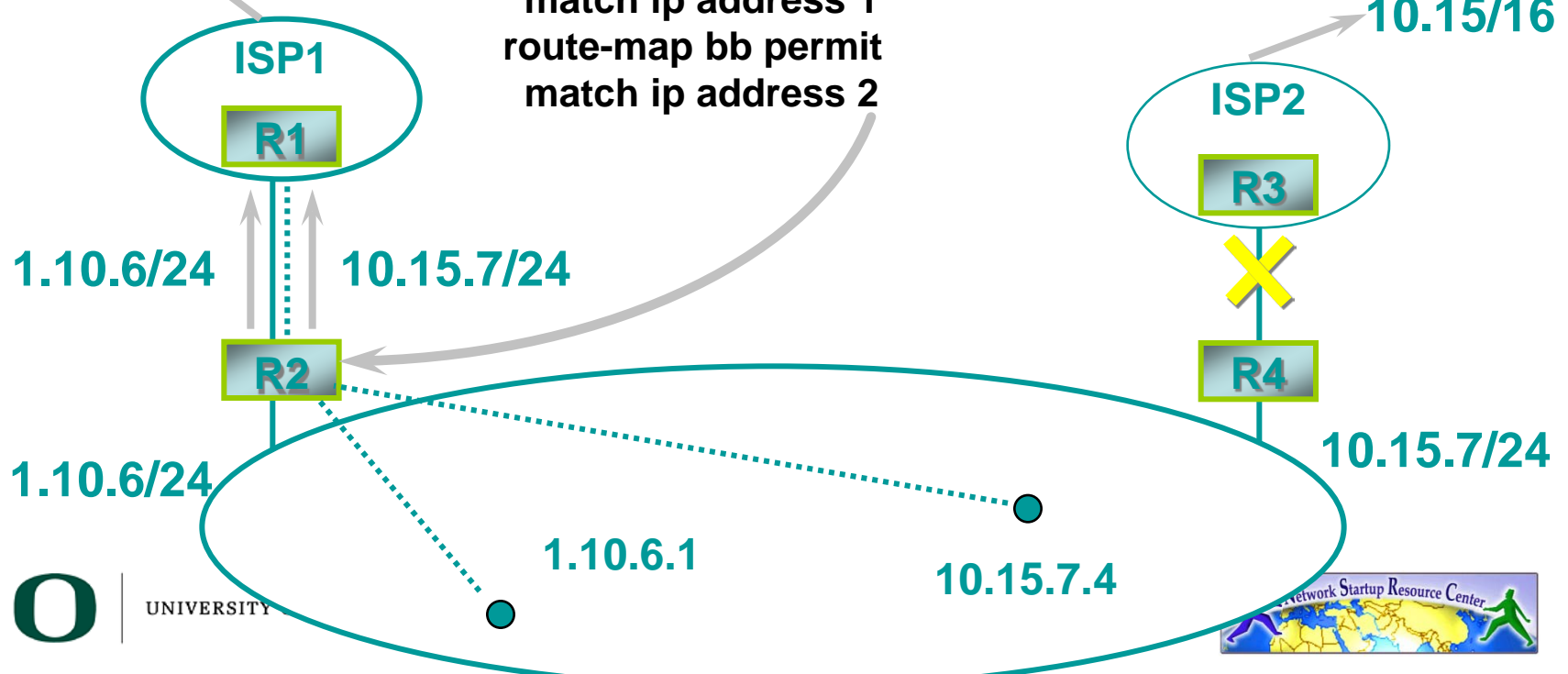


# Using an “*Advertise-Map*”

1.10/16  
10.15.7/24 auto-inject

```

access-list 1 permit 10.15.7.0    !Announces when ...
access-list 2 permit 10.15.0.0    !... this one disappears
neighbor <R1> advertise-map am non-exist-map bb
route-map am permit 10
  match ip address 1
route-map bb permit
  match ip address 2
  
```



# Until Now ...

- **S**tability via:
  - Aggregation
  - Multihoming
  - Inbound/Outbound Filtering
- **S**calability of Memory/CPU:
  - Default, customer routes, full routes
- **S**implicity using “standard” solutions



# ISP Issues

- Scale customer aggregation using BGP
- Offer a choice of route feeds
- Peer with other providers
- Minimize BGP activity and protect against customer's misconfigurations
- Provide a backup service
- Propagate a QoS policy

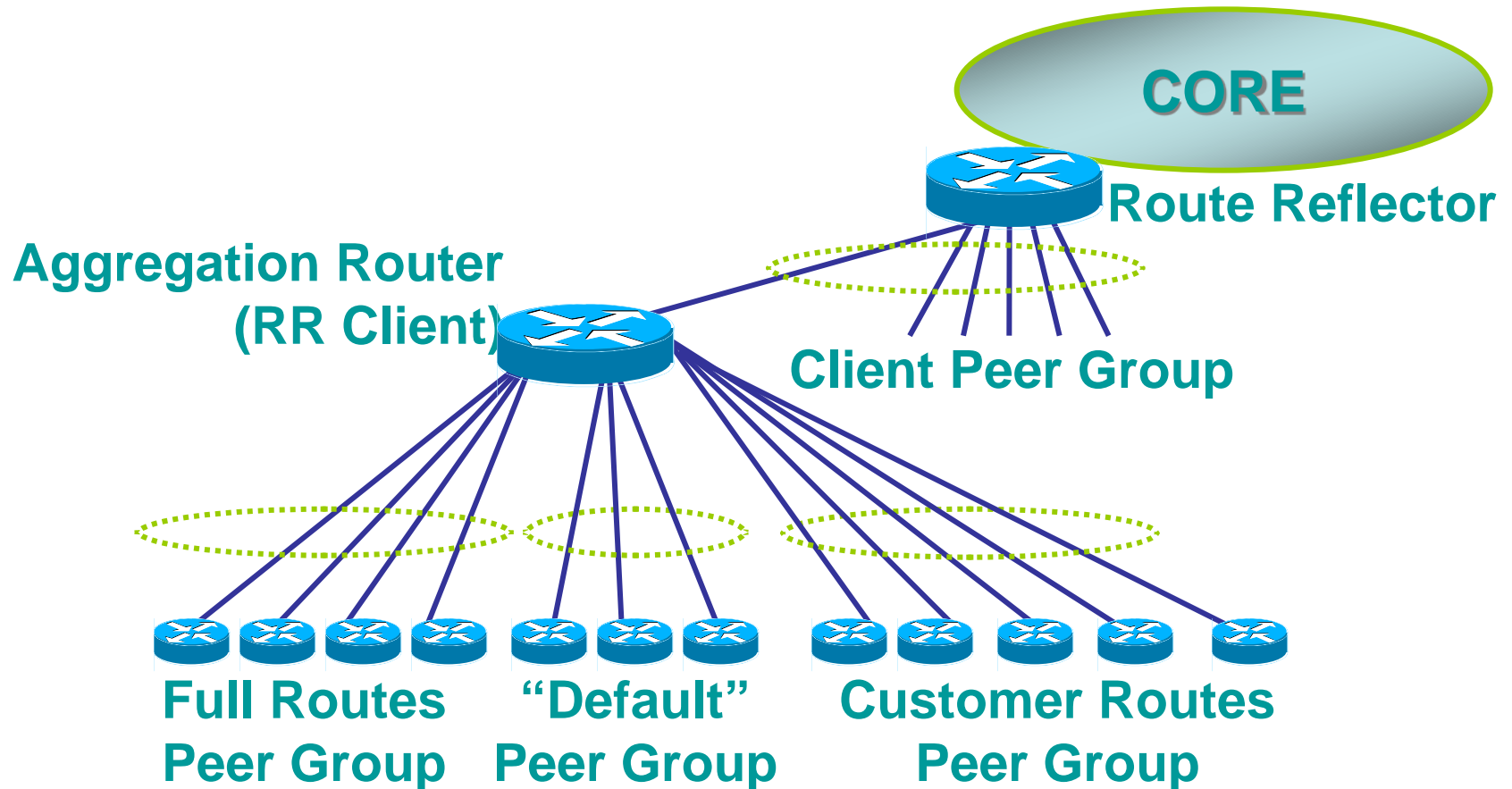


# Guidelines for Customer Aggregation

- Define at least three “*peer-groups*”:
  - cust-default – send default route only
  - cust-customer – send customer’s routes only
  - cust-full – send all routes
- Identify prefixes using communities
  - 2:100=customers;                      2:80=peers
- Apply passwords and an inbound prefix-list on a per neighbor basis



# Customer Aggregation



**NOTE: Apply passwords and inbound prefix list to each customer**



# cust-full Peer-group

neighbor cust-full peer-group

neighbor cust-full description Send all routes

neighbor cust-full remove-private-as

neighbor cust-full version 4

neighbor cust-full route-map cust-in in

neighbor cust-full prefix-list cidr-block out

neighbor cust-full route-map full-routes out

.

ip prefix-list cidr-block seq 5 deny 10.0.0.0/8 ge 9

ip prefix-list cidr-block seq 10 permit 0.0.0.0/0 le 32

# cust-full outgoing route-map

```
ip community-list 1 permit 2:100
```

```
ip community-list 80 permit 2:80
```

```
.
```

```
route-map full-routes permit 10
```

```
  match community 1 80          ; customers & peers
```

```
  set metric-type internal      ; MED = IGP metric
```

```
  set ip next-hop peer-address ; ours
```



# cust-in route-map

```
route-map cust-int permit 10  
  set metric 4294967294 ; ignore MED  
  set ip next-hop peer-address  
  set community 2:100 additive
```



# cust-customer peer-group

neighbor cust-customer peer-group

neighbor cust-customer description Customer Routes

neighbor cust-customer remove-private-as

neighbor cust-customer version 4

neighbor cust-customer route-map cust-in in

neighbor cust-customer prefix-list cidr-block out

neighbor cust-customer route-map cust-routes out



# cust-routes route-map

```
route-map cust-routes permit 10
```

```
  match community 1 ; customers only
```

```
  set metric-type internal ; MED = igp metric
```

```
  set ip next-hop peer-address ; ours
```



# default-route peer-group

**neighbor cust-default peer-group**

**neighbor cust-default description Send Default**

**neighbor cust-default default-originate route-map default-route**

**neighbor cust-default remove-private-as**

**neighbor cust-default version 4**

**neighbor cust-default route-map cust-in in**

**neighbor cust-default prefix-list deny-all out**

**ip prefix-list deny-all seq 5 deny 0.0.0.0/0 le 32**



# default-route route-map

```
route-map default-route permit 10
```

```
set metric-type internal ; MED = igp-metric
```

```
set ip next-hop peer-address ; ours
```



# *Peer Groups* for IXPs & NAPs

- Similar to eBGP customer aggregation except inbound prefix filtering is rarely used
- Instead use *maximum-prefix* and prefix sanity checking
- Continue to use passwords for each neighbor!

# *Peer Groups* for IXPs & NAPs

neighbor nap peer-group

neighbor nap description from ISP A

neighbor nap remove-private-as

neighbor nap version 4

neighbor nap prefix-list sanity-check in

neighbor nap prefix-list cidr-block out

neighbor nap route-map nap-out out

neighbor nap maximum prefix 30000



# *Peer Groups* for IXPs & NAPs

route-map nap-out permit 10

match community 1 ; customers only

set metric-type internal ; MED = IGP metric

set ip next-hop peer-address ; ours



# *Peer Groups* for IXPs & NAPs : Prefix-List sanity-check

# First filter our own address space!!

#deny default

ip prefix-list sanity-check seq 5 deny 0.0.0.0/32

#deny anything beginning with 0

ip prefix-list sanity-check seq 10 deny 0.0.0.0/8 le 32

#deny masks > 20 for all class A networks (1-127)

ip prefix-list sanity-check seq 15 deny 0.0.0.0/1 ge 20

#deny 10/8 per RFC1918

ip prefix-list sanity-check seq 20 deny 10.0.0.0/8 le 32

# reserved by IANA – loopback address

ip prefix-list sanity-check seq 25 deny 127.0.0.0/8 le 32

#deny masks >= 17 for all class B networks (129-191)

ip prefix-list sanity-check seq 30 deny 128.0.0.0/2 ge 17

#deny network 128.0 – reserved by IANA

ip prefix-list sanity-check seq 35 deny 128.0.0.0/16 le 32



# Peer Groups for IXPs & NAPs: Prefix-List sanity-check

```
#deny 172.16 perRFC1918
ip prefix-list sanity-check seq 40 deny 172.16.0.0/12 le 32
#deny class C 192.0.20.0 reserved by IANA
ip prefix-list sanity-check seq 45 deny 192.0.2.0/24 le 32
#deny class C 192.0.0.0 reserved by IANA
ip prefix-list sanity-check seq 50 deny 192.0.0.0/24 le 32
#deny 192.168/16 per RFC1918
ip prefix-list sanity-check seq 55 deny 192.168.0.0/16 le 32
#deny 191.255.0.0 – reserved by IANA (Creo ??)
ip prefix-list sanity-check seq 60 deny 191.255.0.0/16 le 32
#deny masks > 25 for class C (192-222)
ip prefix-list sanity-check seq 65 deny 192.0.0.0/3 ge 25
#deny anything in 223 – reserved by IANA
ip prefix-list sanity-check seq 70 deny 223.255.255.0/24 le 32
#deny class D/Experimental
ip prefix-list sanity-check seq 75 deny 224.0.0.0/3 le 32
```



# Summary

- **Scalability:**
  - Use attributes, specially COMMUNITY
  - Use peer-groups and route-reflectors
- **Stability:**
  - Use loopback addresses for iBGP
  - Generate Aggregates
  - Use passwords per BGP session
  - Always filter inbound and outbound announcements



# Summary

- **S**implicity – use of standard solutions:
  - Three options for multihoming
  - Group customers using communities
  - Apply standard policies at the edge
  - Avoid “special configurations”
  - Automate configuration generation (RR & RtConfig)



# References:

- Cisco ([www.cisco.com](http://www.cisco.com))
- Dave Meyer (dmm@cisco.com)
- John Stewart, BGP4, Addison Wesley
- Sam Halabi, “Internet Routing Architectures”, Cisco Press
- RFCs

## Examples for Customer Filters

```
ip prefix-list announce-my-prefix seq 10 permit <network>/<prefix_mask> ge 23
```

```
ip prefix-list announce-my-prefix seq 100 deny 0.0.0.0/32 le 32
```

```
ip prefix-list accept-default seq 10 permit 0.0.0.0/0 ge 32
```

```
ip prefix-list accept-default seq 100 deny 0.0.0.0/0 le 31
```

```
access-list 10 permit <network> <wildcard_mask>
```

```
access-list 10 deny any
```

```
access-list 20 permit 0.0.0.0 0.0.0.0
```

```
access-list 20 deny any
```

