



Juniper
NETWORKS

Juniper Networks Advanced Policy

Release 5.0, Revision 1

Module 1: Policy Creation

Copyright © 2001, Juniper Networks, Inc.

Objectives

In this module, students will:

- Describe the routing policy framework
- Explain policy match conditions and actions
- Explain the role of the rename and insert CLI commands in routing policy
- Describe the steps needed to test a policy and verify proper policy operation

Copyright © 2001, Juniper Networks, Inc.

This module discusses:

- What the JUNOS software policy framework is and how it is used.
- The operation of policy match conditions and action statements.
- How to alter the names of policies through the use of the rename and insert commands.
- How to test the operation of a policy without applying it to a production routing protocol.

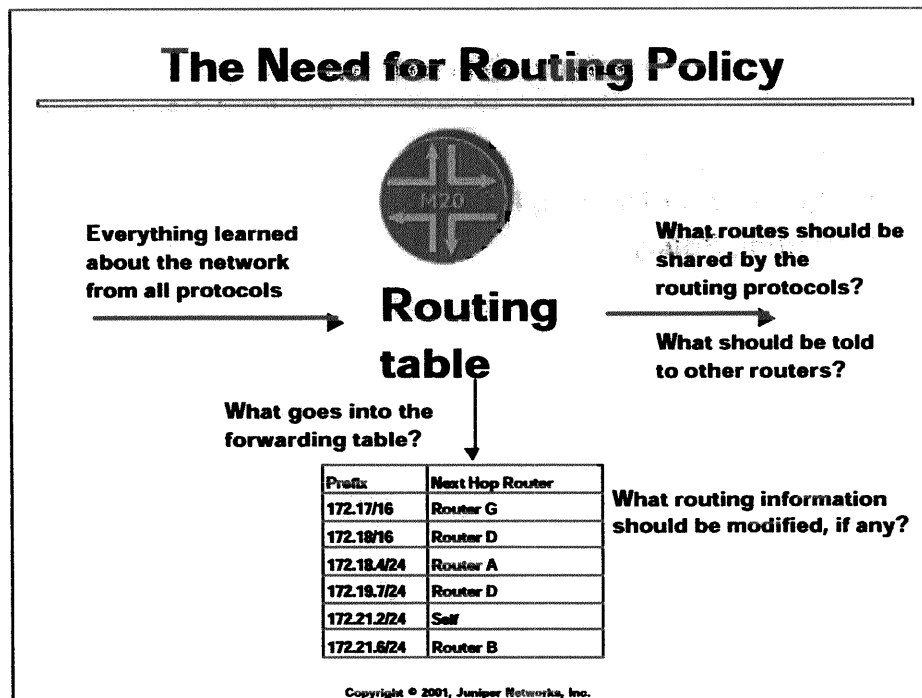
Policy Creation

Where we are going...

- **The Need for Routing Policy**
- **Routing Policy**

Copyright © 2001, Juniper Networks, Inc.

Subsequent slides will discuss the need for routing policy and routing policies.



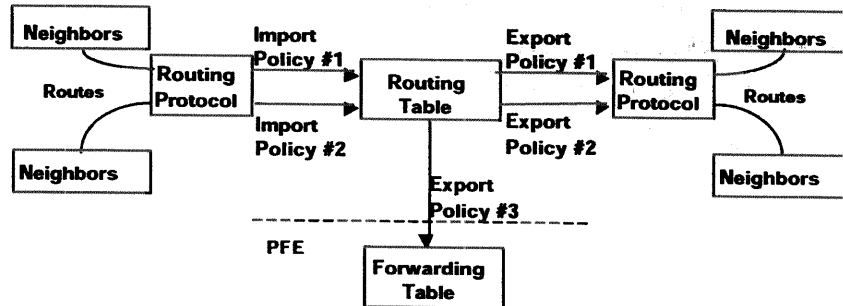
The Routing Policy Framework in the JUNOS Software is the central means for controlling routing information. As depicted in the slide, this control can affect what routes are accepted into the routing table, what routes are advertised from the routing table, and what information gets placed into the forwarding table.

The routing table is the focus of routing protocol activity in the Juniper Networks router. Everything learned about the structure of the network from all of the routing protocols is placed into the routing table (actually, there are several routing tables).

Routing policy is used to determine what routes become active routes. That is, which routes' information is actually installed into the forwarding table as the next hop for a particular destination. This information can be modified by policy action, and policy also determines which routing information is shared by the routing protocols.

Routing Policy

- Applies only to active routes in the routing table
- Controls and filters route information entering or leaving the router
- Are separate for different routing protocols



Copyright © 2001, Juniper Networks, Inc.

The use of routing policies within the JUNOS software is a two step process. The policy must first be created and then it must be applied. Since these are two separate steps, a single policy can be applied to multiple protocols.

All policies are configured and applied with respect to the routing table. Import policies affect what information goes into the routing table and export policies affect what information is advertised out of the routing table.

Multiple policies can be applied to a single protocol. In this case, the evaluation of these policies is accomplished in a pre-defined manner, as can be seen on the next slide. Alternatively, a single policy can be applied to multiple protocols. The idea is to allow for maximum flexibility in policy application.

Only routes that are considered active routes in the routing table are eligible to be evaluated by a policy. Since the routing table contains all possible route knowledge for the router, it is possible that a route exists in the routing table, but does not get evaluated by a policy since it is not considered the current active route.

Note that an export policy can even be applied to routing table information installed into the forwarding table. Several key JUNOS software features such as load balancing take advantage of this application of routing policy. There is no import policy that can be applied to the forwarding table, since this table is always incrementally updated by the routing table and the forwarding table never is used to load information into the routing table.

Policy Construction

Where we are going...

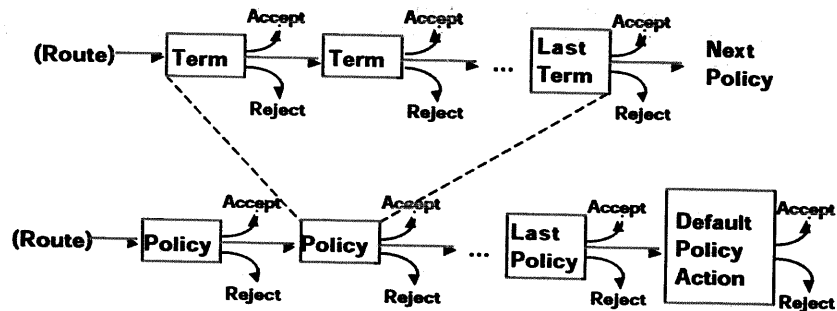
- **Policy Structure**
- **Policy Configuration**
- **Policy Match Conditions**
- **Multiple Match Conditions Defined**
- **Policy Match Actions**
- **Default Policy Actions**
- **Applying Policies**

Copyright © 2001, Juniper Networks, Inc.

Subsequent slides will discuss policy structure, policy configuration, policy match conditions, multiple match conditions defined, policy match actions, default policy actions, and applying policies.

Policy Structure

- A policy has one or more terms
 - Terms have match conditions and actions
- Each protocol can have several policies applied
- Result is both powerful and flexible



Copyright © 2001, Juniper Networks, Inc.

The basic building block of a routing policy is a term that contains a match and action pair. These terms can then be combined together into a single policy. In addition, multiple policies can be combined together into a policy string. The evaluation of a single multi-term policy or a policy string is accomplished in a pre-determined fashion. This evaluation mirrors the JUNOS software configuration where a multi-term policy is displayed and evaluated in a top-down fashion and a policy string is displayed and evaluated in a left-to-right fashion.

The evaluation begins with the first term in the first policy encountered. Within that term, the match criteria are checked to determine if the candidate route being evaluated does indeed meet that criteria. If it does, then the associated action is taken. If it does not meet the criteria, then the associated actions are skipped and the next term in the policy is evaluated. If there is no next term in the policy, then the next policy in the string is evaluated. If there is no next policy in the string, then the protocol's default policy is evaluated. All routing protocols have a default routing policy, which is always applied to the routes imported (received) and exported (advertised) by that protocol.

In the slide, two possible actions are shown - *accept* and *reject*. These two actions are known as *terminating actions* and immediately halt the policy evaluation process. Note that whenever terminating actions are encountered, the terms and/or policies that follow are NOT evaluated.

Also notice that the default policy only has the option to either *accept* a route or *reject* it. This allows all possible routes to "reach" a terminating action.

Policy Configuration

Policies can have multiple terms (one term shown here)

```
[edit policy-options]
policy-statement policy-name {
  term term-name {
    from {
      match-conditions;
      route-filter destination-prefix match-type <actions>;
      source-address-filter destination-prefix match-type <actions>;
      prefix-list name;
    }
    to {
      match-conditions;
    }
    then {
      action;
    }
  }
  final-action;
}
```

If there is no final-action configured,
the default policy action is taken

Copyright © 2001, Juniper Networks, Inc.

The slide shows the basic configuration syntax for a routing policy. As shown on a proceeding slide, policy terms are sets of match and action pairs.

The *from* and *to* sections of the policy configuration make up the match criteria that a route must meet in order for the action to be taken. The *then* portion of the policy is where an action or multiple actions are defined to take on an individual route.

While the syntax examples will always display the concept of a "final action", you can think of this as a final "unnamed" term. As such, you can configure a *from* section for some match criteria as well as an action to take.

It is possible to omit the *from* section in a policy term. When the *from* is omitted, all possible routes are considered to match the criteria (since there are none defined) and the specified action is taken. For example, the following syntax shows a policy term that will accept all routes:

```
[edit policy-options]
policy-statement accept-all-routes {
  term accept-all {
    then accept;
  }
}
```

Policy Match Conditions

- **Route characteristics**
 - 4 metrics, 2 preferences, 2 colors
- **Interface name**
- **Neighbor address**
- **Next-hop address**
- **Protocol (*Source of information*)**
 - bgp, direct, dvmrp, isis, local, mpls, ospf, pim-dense, pim-sparse, rip, static, aggregate
- **For OSPF: Area ID, Tag and Tag2 fields**
- **For IS-IS: Level number**
- **For BGP: AS Path, Community name, Local preference, Origin**
- **Any can be used to find the route of interest**

Copyright © 2001, Juniper Networks, Inc.

There are numerous match criteria that can be used within the configuration of a policy term. The slide shows a subset of those criteria.

There are generic levels of criteria such as the interface the route was received on, the metric of the route, the neighbor or next hop address, and the protocol (more accurately, the term *protocol* indicates the source of the information) that placed the route into the routing table.

There are also match criteria that are used specifically for specific protocols. For example, only ISIS routes contain level information and only OSPF routes contain area information.

One of the powerful things about the JUNOS software policy framework is that any of the match criteria can be configured within the *from* section of a policy. As such, you can be very granular in the method by which you select routes. On the other hand, you can also be too selective in your match criteria which might result in no routes matching your policy.

Multiple Match Conditions Defined

- Within a policy term, there can be multiple match conditions defined
- ALL of the match criteria must be true for a route to take the action
 - This is similar to a logical AND operation

```
policy-statement find-specific-routes {  
  term get-routes {  
    from {  
      protocol static;  
      metric 20;  
    }  
    then accept;  
  }  
}
```

Both
must
match

Copyright © 2001, Juniper Networks, Inc.

There can be multiple match conditions defined with a single policy term. That is, a single *from* statement might have many match conditions within it.

When there are multiple match criteria defined within a single term, the JUNOS software evaluates **ALL** of the criteria to determine if a route matches the policy term. This is very similar in function to a logical Boolean "AND" operation.

The slide shows just a *from* section containing multiple criteria. It is possible to configure a term which has a single *from* and a single *to* statement. In this case, the JUNOS software interprets this as multiple match criteria and the AND operation will be performed.

When one of the match criteria is a **route-filter** (use to select a route based on its prefix and bit-mask length), the AND operation is performed in a slightly different manner. The use of this match criteria will be covered in more detail when we discuss **route-filters** in a subsequent module.

Policy Match Actions

- A route can be subjected to these actions:
 - "Terminate" action: accept or reject (suppress) route
 - Flow Control: skip to next term or policy
 - Modify action: change a route's attributes
 - Trace: log the match to trace file, continue term

```
policy-statement find-specific-routes {  
  term get-good-routes {  
    from {  
      protocol static;  
      metric 10;  
    }  
    then accept;  
  }  
  term stop-bad-routes {  
    from protocol static;  
    then reject;  
  }  
}
```

Copyright © 2001, Juniper Networks, Inc.

After a route has been determined to match all of the defined criteria, then any configured actions can be taken against that route.

There are four main types of actions that can be taken:

- Terminating actions like *accept* and *reject* which halt the policy processing
- Flow control actions like *next term* or *next policy* which alter the default policy processing within and between terms
- Modifying actions that change a route's attributes. Some examples might be altering the metric of a route or changing the BGP AS Path information
- Tracing actions can update a trace file to assist in logging information for network administrators

Much like the match criteria in a policy, there can be multiple actions defined as well. The processing of these multiple actions functions similarly to the processing of multiple terms in a policy -- in a top-down fashion. Each action is taken in turn and, if appropriate, the next action is taken until all have been completed.

This flexibility does come at a price, however. It is entirely possible to configure both an *accept* and a *reject* action within a single term. In a case such as this, the action defined first within the configuration is taken and processing stops at that point. (Recall that both *accept* and *reject* are terminating actions!)

More Policy Match Actions

- A route can also have these parameters modified:
 - Route characteristics (metric, preference, color)
 - The route's Next-Hop Address advertised to neighbors
- For OSPF: modify Type 1/2 external link advertisement
- For BGP: modify AS Path (prepend), add, delete, or set community, change damping (import only), local preference, or origin

```

policy-statement change-things {
  term set-metric {
    from protocol static;
    then {
      metric 20;
      accept;
    }
  }
}

```

Copyright © 2001, Juniper Networks, Inc.

When evaluating an individual route, many of the attributes of that route can be changed via a policy. These attributes can include "protocol neutral" attributes such as metric and preference, as well as the route's next-hop address that is advertised to neighbors.

In addition, attributes specific to an individual protocol can be modified. For example, OSPF can alter the type of external route sent into the network and BGP can alter just about all of its attributes. These include such things as Local Preference, AS Path, Origin, and Communities.

Since multiple actions can be defined within a single term, the modifying actions get applied one after the other in a top-down fashion until either a terminating action is encountered or all of the actions have been completed. The downside to this flexibility is that it is entirely possible to modify an attribute and then immediately change that attribute's value again. For example, the policy term below sets the route metric to 10 and then changes the route metric to 40.

```

[edit policy-options]
policy-statement multiple-actions {
  term alter-metric-twice {
    from protocol static;
    then {
      metric 10;
      metric 40;
    }
  }
}

```


Default Policy Actions

- Each routing protocol being imported (into routing table) or exported (out of routing table) has a different default policy
- Default policy is invoked if there is no match in a policy (or chain of policies) for a route
- For IS-IS and OSPF:
 - Import: policies cannot be used on import
 - Export: export all routes learned by that protocol, and direct routes for the interfaces protocol is configured on
- For BGP:
 - Import: accept all routes learned from BGP neighbors
 - Export: send all routes learned from BGP neighbors to all BGP neighbors, but only the active routes are exported

Copyright © 2001, Juniper Networks, Inc.

A previous slide showed the policy string evaluation sequence, ending with a final policy -- the default policy. The default policy does not have to be defined and is unique for each routing protocol.

Each route being imported (brought into routing table) or exported (advertised from the routing table) has a default policy that applies if there is no match for a route in a policy term or policy string.

The default policy for the routing protocols mirrors the normal operation of that protocol. The slide details the default policy for OSPF, ISIS, and BGP.

It is important to note that since link state routing protocols such as OSPF and ISIS must have consistent databases, no import policies can be applied to these routing protocols.

There is also a default policy for RIP which is as follows:

- Import: accept all routes received on interfaces running RIP
- Export: do not advertise any RIP routes

Applying Policies

- Policies are configured under **[edit policy-options]**
- To take effect, they must be applied to a routing protocol (protocols)
 - This policy allows BGP to advertise routes learned from OSPF

```
policy-options {  
  policy-statement advertise-ospf-routes {  
    term find-ospf {  
      from protocol ospf;  
      then accept;  
    }  
  }  
}  
protocols bgp {  
  export advertise-ospf-routes  
}
```

Copyright © 2001, Juniper Networks, Inc.

A previous slide pointed out that use of the routing policy framework is a two step process. Policies must first be configured under the **[edit policy-options]** section of the JUNOS software configuration.

Then to truly make use of routing policies, they must be applied to one or more protocols configured the router. The slide shows the definition of a policy called **advertise-ospf-routes** that matches all OSPF routes in the routing table and accepts them. The policy is then applied to BGP as an export policy. This combination of creation and application will effectively redistribute all OSPF routes into BGP.

Modifying Existing Policies

Where we are going...

- **Adding Terms**
- **Reordering with Insert**
- **Terms and Policies Can Be Renamed**

Copyright © 2001, Juniper Networks, Inc.

Subsequent slides will discuss adding terms, reordering with insert, and renaming terms and policies.

Adding Terms

- Order counts with policy terms, but terms are always added at the end of the policy statement
- The intent is to reject Area 51 routes, but all OSPF routes have already been accepted by the first term!

```
policy-statement advertise-ospf-routes {  
  term find-ospf {  
    from protocol ospf;  
    then accept;  
  }  
  term reject-area-51 {  
    from {  
      protocol ospf;  
      area 51;  
    }  
    then reject;  
  }  
}
```

Copyright © 2001, Juniper Networks, Inc.

Since policy terms are usually evaluated in a top-down order, the sequence of terms within a given policy is very important. Simply put, *order counts*. However, when terms are added to a policy, the terms are always added in sequence after all previously configured terms. If no terms have been previously defined in a policy, and terms are added, then any initial *from* and/or *then* statements essentially become final actions. Either situation can change the action of the policy and produce unintended results.

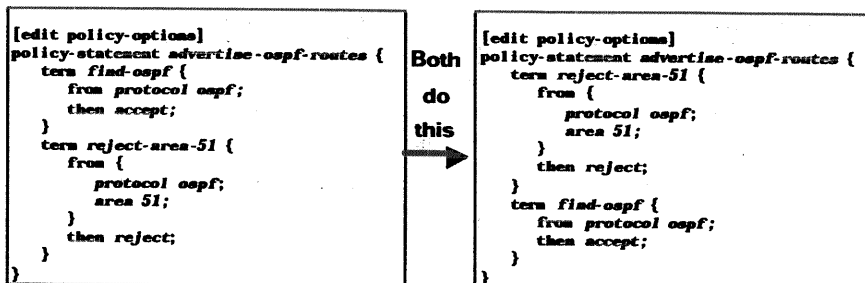
One of the most flexible features of creating a multi-term policy, or an initial policy with named terms, is the ability to add additional terms to the policy at a later time. This allows a network administrator to easily alter the behavior of a policy based on network performance or network changes.

However, it is important to remember that all new terms are added to the end of the policy and that the policy evaluation happens in a top-down fashion. If the newly added term should be applied elsewhere within the policy (at the beginning, for example), it must be moved.

The slide shows the effects of configuring the term **reject-area-51** after the term **find-ospf**. There are no area 51 routes to reject, since the first term has already accepted all OSPF routes.

Reordering with Insert

- **Insert** <statement-path> identifier1 (before | after) identifier2
- Insert policy-statement advertise-ospf-routes term reject-area-51 before term find-ospf
- Insert policy-statement advertise-ospf-routes term find-ospf after term reject-area-51



Copyright © 2001, Juniper Networks, Inc.

To move terms within an existing policy, you can use the **insert** command. This command also works on altering the order of policies within a policy string.

The slide shows the basic syntax of the command and an example of its use. The basic idea is that you tell the configuration CLI where to find a variable and then where to move it to. The location to move the variable to must be within the same configuration hierarchy directory as where it is moving it from. In essence, you cannot move a term from one directory to another using the **insert** command.

To copy a configuration snippet from one hierarchy directory to another you should use the **copy** command. The format of the **copy** command is quite similar to the **insert** command. You inform the configuration CLI where to find the snippet and then where to copy it to. For example, the following command copies a term from one policy (number-1) to another (number-2).

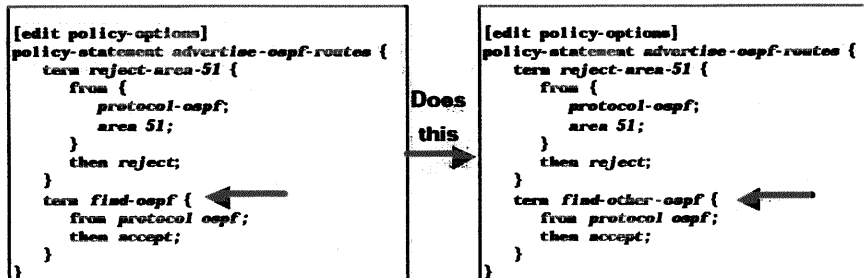
```

[edit policy-options]
user@host# copy policy-statement number-1 term to-be-copied to
policy-statement number-2
  
```

Note that the copy command leaves the original term in place in the original policy. Some further editing might be needed to either delete the original term, or to change the term to ensure proper operation in the new location.

Terms and Policies Can Be Renamed

- **Rename <statement-path> identifier1 to identifier2**
- **Rename policy-statement advertise-ospf-routes term find-ospf to term find-other-ospf**



Copyright © 2001, Juniper Networks, Inc.

In addition to moving policies and terms around, you are also able to rename what you have called the policies and terms. The format of the **rename** command is quite similar to the **insert** command. You tell the configuration where to find a variable and then what to rename it to.

The slide shows a term being renamed within a single policy. You are also able to rename individual policies as well. The command below shows how to change a policy name from **please-change-me** to **policy-is-cool**.

```
[edit policy-options]
```

```
user@host# rename policy-statement please-change-me to policy-
statement policy-is-cool
```

The **rename** command is also commonly used to change other variables (strings) within the router's configuration. For example, the following command can be used to change the IP address on an interface.

```
[edit interfaces so-0/0/0 unit 0 family inet]
```

```
user@host# rename address 192.168.1.1/24 to address 10.10.1.1/24
```

Viewing and Testing Policies

Where we are going...

- **Viewing Policies**
- **Testing A Routing Policy**
- **Policy Testing Example**
- **Policy Testing All Routes**
- **Policy Testing Some Routes**

Copyright © 2001, Juniper Networks, Inc.

Subsequent slides will discuss viewing policies, testing a routing policy, and policy testing examples.

Viewing Policies

- The **show policy** command is used to view the configured policies

```
user@host> show policy
Configured policies:
reject-unwanted-routes
```

- Individual policies can also be viewed

```
user@host> show policy reject-unwanted-routes
Policy reject-unwanted-routes:
  from {
    route-filter 0/0 exact;
    route-filter 127/8 orlonger;
    route-filter 10/8 orlonger;
    route-filter 172.16/12 orlonger;
    route-filter 192.168/16 orlonger;
    route-filter 224/3 orlonger;
  }
  then reject;
```

Copyright © 2001, Juniper Networks, Inc.

In addition to viewing configured policies within the configuration hierarchy, users can also view that information from within the operational command mode. There are two commands that are used to view configured policies.

The **show policy** command will show the names of all the configured policies within the active router configuration.

To actually see the match and action pairs within a specific policy, the command **show policy <policy-name>** can be used.

Testing a Routing Policy

- The **test policy** command can be used to see how a configured policy handles individual prefixes
- Can be a very useful tool before applying the policy in a live environment
- Only active routes in inet.0 are evaluated
- Not all match conditions are supported
 - Most helpful for testing policies that use *route-filter* match conditions
- Default protocol policies are not evaluated
- Default action of **test policy** is **accept**

Copyright © 2001, Juniper Networks, Inc.

The JUNOS software provides a **test policy** command that can be used to see how a particular policy handles the prefixes in the routing table. Traditionally, network administrators have had to apply policies to an operational router to test their effectiveness. Within the JUNOS software, this need for an operational test is not an issue since all configured policies can be tested against the existing routing table. The only requirement for using this function is simply that the policy be configured within [edit policy-options] in the active router configuration.

Once the policy is configured, the policy can be tested against the routing table. As with any applied policy, only the active routes in the routing table get evaluated against the tested policy. Not all match conditions are supported by the **test policy** command, and it is most useful when **route-filter** match conditions are used in a policy term.

In addition, the **test policy** command also has a default policy associated with it that will accept all routes. The **test policy** command default policy can be thought of in the following manner:

```
policy-statement test-policy-default-policy {  
    term accept-all {  
        then accept;  
    }  
}
```

The results the **test policy** command might surprise you. But remember that when the policy you are testing is actually applied to a routing protocol, the default policy for that protocol will be in effect. That protocol's default policy will most likely reject some of the routes that the **test policy** command accepted.

Policy Testing Example

```
policy-statement reject-unwanted-routes {  
  term drop-these-routes {  
    from {  
      route-filter 0/0 exact;  
      route-filter 127/8 orlonger;  
      route-filter 10/8 orlonger;  
      route-filter 172.16/12 orlonger;  
      route-filter 192.168/16 orlonger;  
      route-filter 224/3 orlonger;  
    }  
    then reject;  
  }  
}
```

- Test this policy against all routes in the routing table
- Test this policy against a subset of routes

Copyright © 2001, Juniper Networks, Inc.

When using the **test policy** command, you can specify which routes in the routing table get evaluated by the policy being tested. This is accomplished with the prefix/bit-mask notation found in the slide.

When anything other than 0/0 is used, only a subset of routes will get processed. This subset will contain all routes that share the most-significant bits specified. For example, all routes that start with 192.168 will be evaluated when 192.168/16 is used within the command.

Policy Testing Route Table

```
user@host> show route terse
```

```
inet.0: 100 destinations, 100 routes (100 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both
```

A	Destination	P	Prf	Metric 1	Metric 2	Next hop	AS path
*	10.20.20.0/24	D	0			>so-0/0/3.0	
*	10.49.100.0/24	S	5			Reject	
*	10.222.2.0/24	I	18	20		>10.222.4.1	
*	10.222.4.0/24	D	0			>so-0/0/1.0	
*	155.24.1.0/24	S	5			Reject	
*	155.25.1.0/24	S	5			Reject	
*	155.26.1.0/24	S	5			Reject	
*	172.16.1.0/24	S	5			Reject	
*	172.20.1.0/24	S	5			Reject	
*	172.31.1.0/24	S	5			Reject	
*	192.168.32.0/24	S	5	5		Reject	
*	192.168.33.0/24	S	5	5		Reject	
*	192.168.34.0/24	S	5	5		Reject	
*	192.168.35.0/24	S	5	10		Reject	
*	192.168.48.0/24	B	170	100	5	>10.20.20.2	(65001) I
*	192.168.49.0/24	B	170	100	5	>10.20.20.2	(65001) I
*	192.168.50.0/24	B	170	100	5	>10.20.20.2	(65001) I
*	192.168.51.0/24	B	170	100	10	>10.20.20.2	(65001) I
*	224.10.10.0/24	S	5			Reject	

Copyright © 2001, Juniper Networks, Inc.

The next few slides will show the results of using the **test policy** command. This slide represents the routing table that is currently operational on a router. The policy below will be tested against this table.

```
policy-statement reject-unwanted-routes {
  term drop-these-routes {
    from {
      route-filter 0/0 exact;
      route-filter 127/8 orlonger;
      route-filter 10/8 orlonger;
      route-filter 172.16/12 orlonger;
      route-filter 192.168/16 orlonger;
      route-filter 224/3 orlonger;
    }
    then reject;
  }
}
```

Although we have not discussed the specifics of the **route-filter** command, the routing table above does indeed have routes that should be rejected by the policy.

Policy Testing All Routes

Test this policy against all routes in the routing table

```
user@host> test policy reject-unwanted-routes 0/0
```

```
user@host> test policy reject-unwanted-routes 0/0
```

```
inet.0: 103 destinations, 103 routes (103 active, 0 holdown, 0 hidden)  
Prefixes passing policy:
```

```
155.24.1.0/24      *[Static/5] 00:01:31  
                   Reject  
155.25.1.0/24      *[Static/5] 00:01:31  
                   Reject  
155.26.1.0/24      *[Static/5] 00:01:31  
                   Reject
```

```
Policy reject-unwanted-routes: 3 prefixes accepted, 100 prefixes rejected
```

Copyright © 2001, Juniper Networks, Inc.

Here we are testing the policy against the entire routing table.

The routes displayed after running the command are the routes that have been accepted by the specific policy or the **test policy** default policy.

Policy Testing Some Routes

Test this policy against a subset of routes

```
user@host> test policy reject-unwanted-routes 10.49/16
```

```
user@host> test policy reject-unwanted-routes 10.49/16
```

```
Policy reject-unwanted-routes: 0 prefixes accepted, 1 prefix rejected
```

Copyright © 2001, Juniper Networks, Inc.

Here we are testing the policy against a subset of the entire routing table.

No routes are shown since the only route in the routing table that is a subset within 10.49.0.0/16 gets rejected by the policy. Note that the output at the bottom of the screen states that no routes (prefixes) were accepted and that one route was rejected .

Review Questions

- Are terms required to be configured in a policy statement?
- Why is there not a single default policy for every routing protocol?
- What are the two terminating actions that bring policy processing to a halt?
- What happens when a route attribute (e.g. metric) is changed in a *then* statement, and then changed again?
- What *from* condition will match each and every route?
- How are terms reordered within a policy?
- How is a policy tested against *all* of the entries in the routing table?

Copyright © 2001, Juniper Networks, Inc.

During this module we discussed:

- What the JUNOS software policy framework was and how it was used.
- The operation of policy match conditions and action statements.
- How to alter the names of policies through the use of the rename and insert commands.
- How to test the operation of a policy without applying it to a production routing protocol.



Juniper Networks Advanced Policy

Release 5.0, Revision 1

Module 2: Route Tables, Configured Routes, and Route Filters

Copyright © 2001, Juniper Networks, Inc.

Objectives

In this module, students will:

- **Describe the default JUNOS Software route tables**
- **Configure static, aggregate, and generated routes**
- **Explain how route filters operate on prefixes**

Copyright © 2001, Juniper Networks, Inc.

This module discusses:

- What the default JUNOS software route tables are and how they are populated.
- How to configure static, aggregate, and generated routes.
- The operation of route filters within a policy and how those filters affect routing information.

Routing Tables

Where we are going...

- Routing Table and Forwarding Table
- Routing Table inet.0
- Routing Table inet.1
- Routing Table inet.2
- Routing Table inet.3
- Routing Table mpls.0
- Synchronizing The Tables
- Determining the Active Route
- Default Route Preference

Copyright © 2001, Juniper Networks, Inc.

Subsequent slides will discuss the separate routing tables in the JUNOS software, the synchronization of those tables, and the selection of an active route.

Routing Table and Forwarding Table

- **Juniper Networks routers have distinct terminology**
- **Routing table: *all* of the routing information learned by *all* routing protocols**
 - May have multiple routes to a single destination
- **Forwarding table: Routes actually used to forward packets (active routes)**
 - Only *one* route to each possible destination
- **Supported routing and signaling protocols:**
 - Unicast: BGP, IS-IS, OSPF, RIP
 - Multicast: DVMRP, IGMP, MSDP, PIM-SM, PIM-DM, SAP/SDP
 - MPLS: LDP, RSVP
- **Default routing tables:**
 - inet.0 (unicast), inet.1 (multicast), inet.2 (unicast RPF lookup)
 - inet.3 (MPLS LSPs), mpls.0

Copyright © 2001, Juniper Networks, Inc.

Routing tables in Juniper Networks routers use a distinct terminology that might vary from that of other vendors. So a quick look at what is meant by these terms is in order.

The routing table in the JUNOS software is the total amount of information known to the router. Routing protocols place all of their "best" routes into the routing table, so there might be many routes to the same destination prefix. In addition, *configured routes* (such as static and aggregate routes) also get placed into the routing table. It is then up to the routing table to decide on a single *best route* to any single destination. These single best routes (or *active routes*) get placed into the *forwarding table* which gets downloaded into the Packet Forwarding Engine. The supported routing and signaling protocols are listed in the slide.

The routing table in the router actually contains multiple, separate tables which get created as the router needs them. Some of the default tables are listed in the slide. In addition, other routing tables get created when their features are enabled in the JUNOS software. For example, if IPv6 information has been configured, routing information gathered via this protocol is placed into the inet6.0 table.

When a Juniper Networks router is participating in an RFC 2547bis Virtual Private Network (VPN) environment, a number of new routing tables get created. Some of these include:

- *instance.inet.0* - For each VPN Routing and Forwarding (VRF) instance created, a separate routing table is created to hold information specific to that VRF. The *instance* name is the string name given to the VRF within the configuration
- i3vpn.bgp.inet.0
- i2vpn.bgp.inet.0

The use of these routing tables by a VPN is beyond the scope of this course. This topic is covered by a different course.

Routing Table inet.0

Used for unicast routes

```

user@host> show route table inet.0

inet.0: 49 destinations, 49 routes (49 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

10.0.11.0/24      *[Direct/0] 1d 08:19:20
                  > via at-0/1/0.100
10.0.11.1/32      *[Local/0] 1d 08:19:20
                  Local
192.168.1.0/24    *[BGP/170] 00:06:08, localpref 100
                  AS path: 1 I
                  > to 10.0.11.2 via at-0/1/0.100
192.168.16.0/21   *[Static/5] 00:02:40
                  Discard
                  [Aggregate/130] 00:36:17
                  Reject
192.168.20.0/24   *[Static/5] 00:06:12
                  Reject

```

Copyright © 2001, Juniper Networks, Inc.

This is an example of the contents of the basic inet.0 routing table. As the router's interfaces get configured with IP addresses, the inet.0 table gets created and the interface routes (Direct and Local) get placed into it. Local routes are always 32 bit masked addresses.

The inet.0 routing table is the main unicast routing table for the router. The inet.0 table contains the information used to route most of the user packets through the network.

The active route to any particular destination is usually shown by the asterisk (*) next to the protocol which placed the route into the table. The asterisk is really a sort of combination of the plus (+) sign, which is the actual active route flag, and the minus (-) sign, which indicates the route that was the last active route loaded into the forwarding table when the forwarding table was incrementally updated. Asterisks are a sign of network stability.

For example, in the slide, the route 192.168.16/21 which was placed into the inet.0 routing table by the protocol "static" is *preferred* over the same route placed in the routing table by the protocol "aggregate." The specifics behind how this decision is made to make one route active and not another will be covered in a future slide.

Routing Table inet.1

Used for multicast route information

```
user@host> show route table inet.1

inet.1: 1 destinations, 1 routes (1 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

232.232.2.2,10.10.200.200/32*[PIN/105] 00:01:58
      Multicast
232.232.1.1,10.10.201.200/32*[PIN/105] 00:08:46
      Multicast
```

Copyright © 2001, Juniper Networks, Inc.

This is an example of the contents of the inet.1 routing table. This is the multicast forwarding cache for the router. Multicast routes, in contrast to unicast routes, consist of pairs of IP addresses indicating the source (multicast server) and group (receiving clients) address of the multicast content.

As valid multicast (Source, Group) pairs are discovered, they are placed into this routing table. Multicast user traffic is then forwarded based on the information contained in this table. The inet.1 table is organized by multicast group address.

Routing Table inet.2

Used for unicast Reverse Path Forwarding (RPF)

```

user@host> show route table inet.2

inet.2: 9 destinations, 9 routes (9 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

10.20.20.0/24      *[Direct/0] 2d 19:39:14
                  > via so-0/0/3.0
10.20.20.1/32     *[Local/0] 2d 19:39:14
                  Local
10.222.4.0/24     *[Direct/0] 2d 19:39:14
                  > via so-0/0/1.0
10.222.4.2/32     *[Local/0] 2d 19:39:14
                  Local
10.222.6.0/24     *[Direct/0] 2d 19:39:14
                  > via so-0/0/2.0
10.222.6.1/32     *[Local/0] 2d 19:39:14
                  Local
192.168.32.1/32  *[Direct/0] 2d 19:39:14
                  > via lo0.0

```

Copyright © 2001, Juniper Networks, Inc.

Here is an example of the contents of the inet.2 routing table. This routing table has been set aside within the JUNOS software for use with multicast applications. Its function is to assist in a Reverse Path Forwarding (RPF) decision.

RPF is an important concept in a multicast network. This is the loop avoidance mechanism used to keep multicast traffic from constantly circulating on a network. The concept is quite simple. When a router receives traffic for a multicast destination, it first finds the source of the traffic using the (Source, Group) pair listed in the inet.1 routing table. The router then determines if it knows about that source via its unicast inet.0 routing table. In addition, the router further determines if the interface that the multicast traffic was received on is along the shortest path from the router back to the source. If *both* of these conditions are met then the traffic can be forwarded to downstream routers and user nodes. If either of these conditions is not met, then there is a potential for a multicast loop and the traffic will be dropped.

By default, the JUNOS software uses the inet.0 routing table for RPF checks. However, the inet.2 table has been reserved for this purpose if enabled so that multicast and unicast traffic are not required to follow the same network topology (helpful in heavy multicast environments). To enable the use of the inet.2 table, configure a **rib-group** under the [edit routing-options] section of the configuration hierarchy.

Routing Table inet.3

Used for MPLS traffic-engineered LSPs

```
user@host> show route table inet.3

inet.3: 1 destinations, 1 routes (1 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

192.168.96.1/32    *[RSVP/7] 00:05:12, metric 20, metric2 0
                  > via so-0/0/3.0, label-switched-path to-the-egress
```

Copyright © 2001, Juniper Networks, Inc.

This is an example of the contents of the inet.3 routing table. This table contains information about operational Multi-protocol Label Switching (MPLS) Label Switched Paths (LSPs). Once an LSP has been configured, signaled, and is ready for use the IP address of the LSP egress router gets placed into the inet.3 routing table.

No actual user traffic gets forwarded based on the inet.3 table directly. Instead, other routing protocols can use the information in inet.3 for resolving next-hop information. If one of these routing protocols, BGP for example, decides to use an LSP found in inet.3, it then places its route into inet.0 with a next-hop of the LSP. User traffic is then forwarded based on the inet.0 route via the LSP.

Routing Table mpls.0

Used for MPLS label switching

```

user@host> show route table mpls.0

mpls.0: 4 destinations, 4 routes (4 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

0          *[MPLS/0] 00:06:40, metric 1
            Receive
1          *[MPLS/0] 00:06:40, metric 1
            Receive
100000     *[RSVP/7] 00:06:18, metric 1
            > via so-0/0/1.0, label-switched-path to-the-egress

```

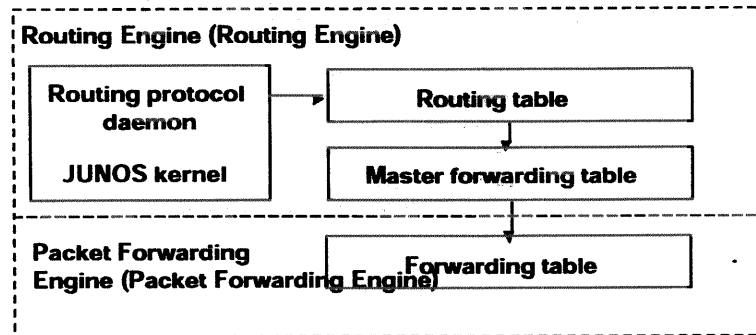
Copyright © 2001, Juniper Networks, Inc.

This is an example of the contents of the mpls.0 table. Technically speaking, the mpls.0 table is a switching table, not a routing table. But since the mpls.0 table gets displayed via the **show route** command, it is useful to talk about it here.

As user packets traverse an MPLS LSP, they get forwarded on a hop-by-hop basis based on the information contained in the MPLS header instead of the IP header. The incoming LSP label information in the MPLS header gets read and a lookup is performed in the mpls.0 table. If the incoming label is found in the mpls.0 switching table, the packet gets forwarded out the specified interface with a new MPLS label attached.

Synchronizing the Tables

- Routing protocol daemon calculates active routes from routing table and installs them into forwarding table
- Copy of forwarding table is installed into Packet Forwarding Engine



Copyright © 2001, Juniper Networks, Inc.

As discussed previously, the routing table within the JUNOS Software contains all of the routing knowledge known to the router at any point in time. From this total set of information, a single active route to any destination is chosen and this active route gets placed into the forwarding table.

The slide shows that there are actually two copies of the forwarding table. The active routes from the routing table get placed into the Master Forwarding Table on the Routing Engine.

If changes need to be made to the Master Forwarding Table on the Packet Forwarding Engine, only those incremental changes get placed into the Packet Forwarding Engine forwarding table.

Determining the Active Route

- Only *one* active route per destination is loaded from the routing table into the forwarding table
- How is the active route chosen from among many?
- For all routing protocols *except BGP*, the choice is:
 - Prefer path with lowest routing protocol preference
 - Prefer path with lowest preference2 value
 - Prefer path with lowest color value
 - Prefer path with lowest color2 value
 - Prefer path with lowest metric value
 - Prefer path with lowest metric2 value
 - Prefer path with lowest metric3 value
 - Prefer path with lowest metric4 value
- Routing policy can change these values

Copyright © 2001, Juniper Networks, Inc.

Only one route for each destination can be active in the forwarding table at any one time. However, if there are several routing protocols running in the router, there can be many routes to the same destination in the routing table.

So how is the active route chosen from among many possible ways to get to a given destination prefix?

Each route in the JUNOS software routing table contains eight possible values that can be used to determine which route becomes the active route when there are multiple possible routes to choose from. The slide shows the values and the order of preference among those values for all routing protocols *except* BGP. Generally speaking, the preference value of a route is usually the determining factor in deciding upon the active route.

The Border Gateway Protocol (BGP) adds to the variables above for a more comprehensive decision process. BGP specifics will be discussed in a later module in this course.

Each of these eight values can be changed by a routing policy.

Default Route Preference

- Which route source to install is a matter of *preference*
- Preference of Direct & Local routes cannot change

Route learned by:	Preference:
Direct	0
Local	0
Static	5
RSVP	7
LDP	9
OSPF internal	10
IS-IS L1 int.	15

Route learned by:	Preference:
IS-IS L2 int.	18
RIP	100
Aggregate	130
OSPF AS ext.	150
IS-IS L1 ext.	160
IS-IS L2 ext.	165
BGP	170

Copyright © 2001, Juniper Networks, Inc.

The tables above show almost all of the default route preferences for the JUNOS software. With the exception of the directly connected routes (Direct and Local), these preference values can be changed either via a configuration knob or a routing policy.

Generated routes (discussed in a future slide) are placed into the routing table as protocol *aggregate* with a preference value of 130. This is due to the fact that these routes are similar to aggregate routes in almost every single respect except one.

In addition to the information above, routes created as part of an RFC 2547bis VPN environment also have preference values assigned to them. For example, if the VPN has been established with RSVP, the preference of the VPN is 7.

Static Routes

Where we are going...

- **Configuring Static Routes**
- **Static Routes**
- **Static Routes In an ISP Network**

Copyright © 2001, Juniper Networks, Inc.

Subsequent slides will discuss how static routes are configured within the JUNOS software and why these routes are needed in an ISP network.

Configuring Static Routes

- Manually configured routes added to the routing table
- Once active, remain in the routing table until deleted
- Configured at the routing-options hierarchy level

```
[edit]
routing-options {
  static {
    defaults {
      static-options;
    }
    route destination-prefix {
      next-hop;
      static-options;
    }
  }
}
```

Copyright © 2001, Juniper Networks, Inc.

Unlike dynamic routing protocols, routing information provided by static routes is manually configured on each router in the network.

In addition, these routes are permanently in the routing table until they are removed by a network administrator or they become "non-active". One possible way for a static route to not be active is for the IP address of the next-hop to not be reachable across a directly connected interface.

Static routes are configured at the [routing-options] level of the CLI hierarchy. Options applied at the **default** level apply to all static routes except those that have the same option configured for a particular route. Options applied to a particular static route apply *only* to that route.

Possible options that can be assigned to static routes include:

- **AS-Path** – Used if this route is intended to be redistributed into BGP and you wish to manually add values to the AS_PATH attribute.
- **Community** – Used if this route is intended for BGP and you wish to add community values to the route for use in your AS.
- **Metric** – If multiple routes share the same preference value, then the route with the best metric will become active in the routing table. Use this value to prefer one route over another in the case of a preference tie.
- **Preference** – The default preference value of static routes is 5. This makes them more likely to be active than OSPF, ISIS, or BGP for matching prefixes. Use this option to increase the value of the static routes to prefer other sources of routing information.

Static Routes

- Static routes require a next-hop to be configured.
 - Valid options are IP address, Discard, Reject, and Receive
- Defaults section affects all static routes

```
routing-options {  
  static {  
    defaults {  
      preference 250;  
    }  
    route 192.168.20.0/24 next-hop 10.0.0.1;  
    route 192.168.21.0/24 discard;  
    route 192.168.22.0/24 reject;  
  }  
}
```

Copyright © 2001, Juniper Networks, Inc.

In order to be committed to the active configuration, static routes must have a next-hop value configured. In many cases that next-hop value is the IP address of the neighboring router headed towards the ultimate destination. But another possibility is that the next-hop value should be a configured null value. This null value is used to indicate that the router has dropped the packet off of the network. Within the JUNOS software, the way to represent the dropping of packets is with the keywords *reject* or *discard*. While both options will drop the packet from the network, the difference lies in the action the router takes after the drop action. If *reject* is specified as the next-hop value, the router will send an ICMP message (network unreachable) back to the source of the IP packet. If *discard* is specified as the next-hop, the router does NOT send back an ICMP message, the packet is dropped silently.

Once in the configuration, static routes will appear in the routing table if they are active. Active static routes have a valid next-hop option. Routes with *reject* or *discard* as next-hops will always be active and present in the routing table. Routes with an IP address as a next hop will be present only if that address is reachable across a directly connected interface on the router.

fax: 408/745-2100
main: 408/745-2000
www.juniper.net

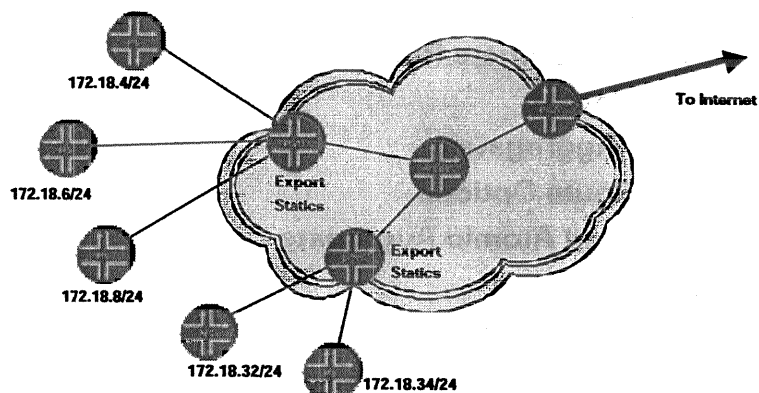
Within the `[edit routing-options static]` configuration hierarchy, the **defaults** section can also contain static route options. Any options configured within this section are applied to all static routes on the router. In the example on the slide, the preference value has been changed to 250. This means that all static routes configured on the router will have that preference value.

Any single static route can override the defaults section by configuring the same option within its specific route section of the configuration hierarchy. Any option defined specifically for a route overrides the same option defined within the defaults section.

While a single static route can have multiple IP next-hops configured, there can only be one static route configured per prefix. Traffic for that specific route can be load balanced over the multiple next-hops through the use of a policy.

Static Routes in an ISP Network

- Static routes can be used to connect customers
- Statics are then advertised into ISP network



Static routes are useful in an Internet Service Provider (ISP) network for a number of things. Perhaps the most useful function of static routes is to represent customer networks within the ISP's Autonomous System (AS).

On the edge router facing the customer, a static route is configured for the IP address block assigned to that customer. The next-hop for this static route is the IP address of the Customer Premises Equipment (CPE) interface that connects the customer to the ISP. This static route, as well as other customer static routes on the router, are then redistributed into the ISP routing protocols (shown as *export statics*). In this fashion, the ISP now has connectivity to the customer network throughout its AS.

Aggregate Routes

Where we are going...

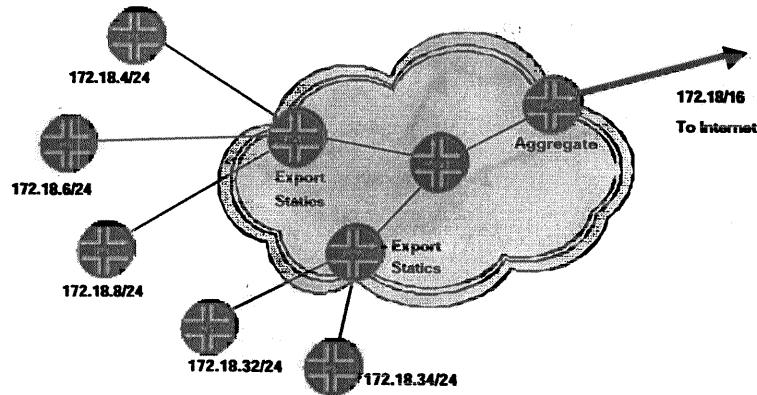
- **Route Aggregation**
- **Aggregates In The Internet**
- **Longest Match Consequences**
- **Configuring Aggregate Routes**
- **Aggregate Routes**
- **Advertising Aggregates**
- **Aggregate Route Options**
- **Aggregator and Atomic Aggregate**
- **Contributing Routes**
- **Controlling Contributing Routes**

Copyright © 2001, Juniper Networks, Inc.

Subsequent slides will discuss route aggregation and its consequences, configuring aggregate routes, advertising of those routes, and contributing routes in the routing table.

Route Aggregation

- Internet routes have grown very quickly
- Aggregation can cut down on table sizes



The number of Internet routes has grown dramatically. This has meant that router table sizes have grown in proportion, and the amount of bandwidth needed to distribute this routing information has grown as well.

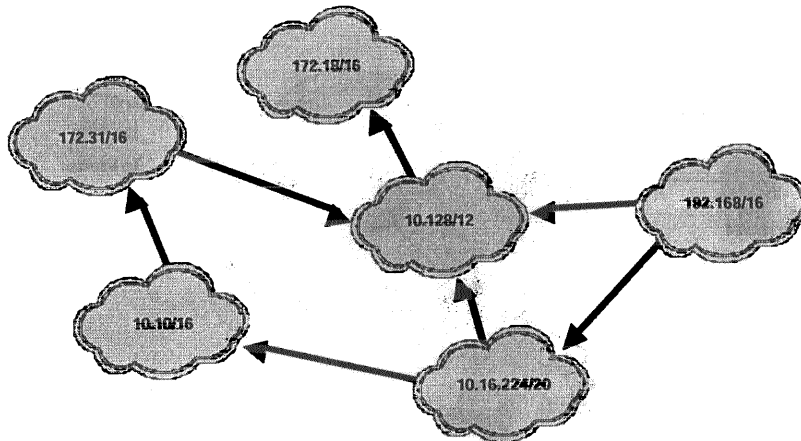
Route aggregation (also often called *route summarization*) can cut down on the size of the tables required to store routing information.

Generally speaking, there is always a trade off within a network environment. Such is the case with route aggregation. On the one hand, aggregate routes reduce the amount of information stored within the routing tables and, even more importantly today, sent between routers using the routing protocols. On the other hand, each router now has less granular information from which to make a routing decision which could result in suboptimal routing. The details of an individual route become blurred when this route is aggregated.

In the scope of the Internet as a whole, route aggregation is viewed as a good thing. The possibility of suboptimal routing is considered a small price to pay when routing tables are over 100,000 routes.

Aggregates In The Internet

- Most ISPs would prefer to only advertise aggregates
- Not always possible



Copyright © 2001, Juniper Networks, Inc.

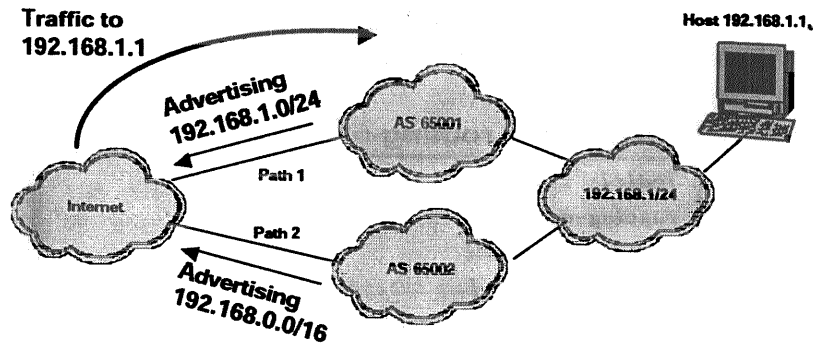
In a perfect world, all ISPs would like to advertise only a single aggregated route that represents all of its assigned IP addresses. If this were always possible, the number of Internet routes would equal the total number of ISPs in the world.

Of course, we don't live in a perfect world, so this is not always possible. There are a number of reasons why an ISP's routes cannot be neatly aggregated into one route.

In today's networking environment, many businesses are relying on the Internet for basic connectivity and take network downtime very seriously. In an attempt to avoid such issues, business customers may purchase service from multiple ISPs. These businesses would then like their IP address space to be announced through both ISPs for connectivity reasons.

But what makes perfect sense for the customer complicates things for the ISPs. There are potential issues with this type of a configuration when it comes to seeking the best route to a particular destination.

Longest Match Consequences



Destinations connected to multiple domains must always be explicitly announced (nonaggregate form)

Copyright © 2001, Juniper Networks, Inc.

In this example, a customer has existing service from an ISP using AS 65002 and has been assigned the IP address block of 192.168.1/24 from AS 65002. The customer then purchases service from an ISP using AS 65001 and wishes to have connectivity through that ISP as well.

AS 65001 then starts to advertise the /24 subnet into the public Internet. However, AS 65002 is still only advertising its aggregate route (192.168/16) to the Internet. This will cause all traffic destined to the customer to traverse AS 65001, since routers in the Internet will perform a longest-match lookup in their routing tables and choose the /24 route over the /16 aggregate. This might not be what either the ISPs nor the customer wants. For example, the main ISP for the customer might be AS 65002 and AS 65001 is intended only as a backup path.

To avoid this situation, AS 65002 must "punch a hole" in its aggregate and start announcing two routes -- 192.168/16 and 192.168.1/24. This is current best practice for multi-homed customer's addresses: destinations connected to multiple routing domains must always be explicitly announced in non-aggregate form to make sure that the longest-match rule will apply to them.

Configuring Aggregate Routes

- Route prefixes in the network can be combined into a single entry in the Routing Table
- Becomes active once one or more contributing routes are active
- Configured at the routing-options hierarchy level

```
[edit]
routing-options {
  aggregate {
    defaults {
      aggregate-options;
    }
    route destination-prefix {
      policy policy-name;
      aggregate-options;
    }
  }
}
```

Copyright © 2001, Juniper Networks, Inc.

Like static routes, aggregate routes are manually configured on each router in the network. Aggregate routes become active in the routing table when at least one of the contributing routes (the more specific routes under the aggregate range) for the aggregate is also active in the routing table.

Options that can be configured for an aggregate route include:

- AS-Path—Used if this route is intended to be redistributed into BGP and you wish to manually add values to the AS_PATH attribute.
- Community—Used if this route is intended for BGP and you wish to add community values to the route for use in your AS.
- Metric—If multiple routes share the same preference value, then the route with the best metric will become active in the routing table. Use this value to prefer one route over another in this case.
- Policy—By default, all possible more specific contributing routes can be used to activate an aggregate route. To alter this default, you can use a policy to accept or reject certain routes that should or should not be used.
- Preference—The default preference value of aggregate routes is 130. Use this option to alter the value of aggregate routes.

Aggregate Routes

- The default next-hop for an aggregate is **Reject**
 - Discard is also a valid option
- Defaults section affects all aggregate routes
- Aggregates rely on more specific routes to be active

```
routing-options {  
  aggregate {  
    defaults {  
      community 1:888;  
    }  
    route 192.168.16.0/21;  
    route 192.168.24.0/21 discard;  
  }  
}
```

Copyright © 2001, Juniper Networks, Inc.

The default next-hop value for aggregate routes is *reject*. This means that the router will drop the packet from the network and will send an ICMP message (network unreachable) back to the source of the IP packet. The other possible next-hop value for aggregate routes is *discard*. As usual with discard, the router does NOT send back an ICMP message and the packet is dropped silently. It might sound odd that aggregates are configured without "real" next-hops. An "aggregate" route configured in the JUNOS software with a valid IP address next-hop is called a *generated route*.

Within the [edit routing-options aggregate] configuration hierarchy, the defaults section can contain aggregate route options. Any options configured within this section are applied to all aggregate routes on the router. In the example on the slide, the community value has been set to 1:888. This means that all aggregate routes configured on the router will have that community value.

Any single aggregate route can override the defaults section by configuring the same option within its specific route section of the configuration hierarchy. Any option defined specifically for a route overrides the same option defined within the defaults section.

Aggregate routes rely on the more specific routes that fall under them to be active. Once a more specific route becomes active, the aggregate route that includes it becomes active. Aggregate routes that become active can in turn activate even more inclusive aggregate routes "above" them. For example, the aggregate route 192.168.16/21 when active can in turn activate the aggregate route 192.168/16.

NOTE

You can only configure one aggregate route per prefix.

Advertising Aggregates

A policy can be used to advertise the aggregate routes via a routing protocol

```

policy-options {
  policy-statement advertise-aggregates {
    term find-aggregates {
      from protocol aggregate;
      then accept;
    }
  }
}
protocols {
  bgp {
    export advertise-aggregates;
  }
}

```

Copyright © 2001, Juniper Networks, Inc.

Within the JUNOS software, aggregate routes are considered to be *configured routes*. Configured routes are never automatically advertised into a dynamic routing protocol. To have aggregate routes redistributed into a routing protocol, such as BGP, it is necessary to configure a policy similar to the example in the slide.

This example defines a policy called **advertise-aggregates**. The match condition for this policy is **protocol aggregate** and the action is **accept**. The effect of this policy is that all active aggregate routes in the inet.0 routing table will be accepted.

The policy is then applied as an **export** policy within BGP. This has the desired end result of advertising the aggregate routes on this router into BGP and potentially to the Internet.

Aggregate Route Options

- **Aggregate routes can have added information stored with them in the routing table**
 - route level overrides default level options
- **Route metric: one of the four metrics**
- **Route preference: two preferences and two colors**
- **BGP Community Values**
- **AS Path: controls path information for route**
- **AS numbers to include in aggregate route**
 - Controls which AS numbers to include in aggregate path
- **OSPF tag: associates tag or tag2 with aggregate**
- **Aggregates sound complex, but usually very simple**

Copyright © 2001, Juniper Networks, Inc.

Aggregate routes can have additional information stored with them in the routing table. This added information is optional and options configured at the **route** level overrides options configured at the **default** level.

Options that can be configured for an aggregate route include:

- **Metric**—If multiple routes share the same preference value, then the route with the best metric will become active in the routing table. Use this value to prefer one route over another in this case.
- **Preference**—The default preference value of aggregate routes is 130. Use this option to alter the value of aggregate routes.
- **Community**—Used if this route is intended for BGP and you wish to add community values to the route for use in your AS.
- **AS-Path**—Used if this route is intended to be redistributed into BGP and you wish to manually add values to the AS_PATH attribute.
- **OSPF tag**—Used to associate the value of an OSPF tag (or tag2) with the aggregate route.

Aggregate routes sound very complicated, but in practice they are usually quite simple to configure and use.

Aggregator and Atomic Aggregate

- **Aggregator is an optional BGP path attribute**
 - Includes the AS number and IP address of the BGP system that formed the aggregate
 - `<aggregator as-number in-address>`
- **Atomic Aggregate is an option within the AS Path section of an aggregated route**
 - `ATOMIC_AGGREGATE` is a BGP path attribute
 - Can be set when aggregation results in "loss of information"
 - JUNOS software includes all information by default, so this is not usually set

Copyright © 2001, Juniper Networks, Inc.

While it is generally agreed upon in the Internet that route aggregation is a good thing, it is also helpful to know where that aggregation took place and if information was "lost" in the aggregation. "Lost" information could include details about which specific AS_Path a route had prior to being aggregated, for instance.

In order to let other routers know when this situation occurs, the Border Gateway Protocol (BGP) has two optional attributes defined. The first optional attribute is called the *Aggregator* which simply places the router-id and AS number of the BGP router that performed the aggregation into the attribute list.

The second attribute is called the *Atomic Aggregate* which is a "flag" that represents a loss of information occurred during the aggregation. These two attributes usually appear together within a route advertisement.

The default behavior for the JUNOS software during route aggregation is to include all of the more specific information into the aggregate. Because of this default behavior, the aggregator and atomic aggregate attributes are *not* set within the aggregate route. Only when a condition occurs when information is truly lost do these attributes get set.

Contributing Routes

- All routes that fall within an aggregate route
- Any active route in the routing table can contribute
- A single route can contribute to only one aggregate route
- View with **show route extensive**

```

172.16.64.0/20 (1 entry, 1 announced)
*Aggregate      Preference: 130
                 Next hop type: Discard ←
                 State: <Active Int Ext>
                 Age: 11:02
                 Task: Aggregate
                 Announcement bits (1): 0-KRT
                 AS path: I
                 Flags: Generate Discard Depth: 0      Active
                 Contributing Routes (7):
                   172.16.65.0/24      proto Static
                   172.16.66.0/24      proto Static
                   172.16.67.0/24      proto Static
                   172.16.68.0/24      proto Static
                   172.16.69.0/24      proto BGP
                   172.16.70.0/24      proto BGP
                   172.16.71.0/24      proto BGP
    
```

Copyright © 2001, Juniper Networks, Inc.

The concept of a *contributing route* is important to aggregate routes. By definition, all routes that share the most common bits with the aggregate and have a longer bit-mask length are eligible to contribute to the aggregate.

Any active route can be a contributing route to an aggregate, but this of course depends on the configuration details.

NOTE

While all active routes in the routing table are eligible to contribute to an aggregate route, a single route can contribute to only one aggregate route at a time.

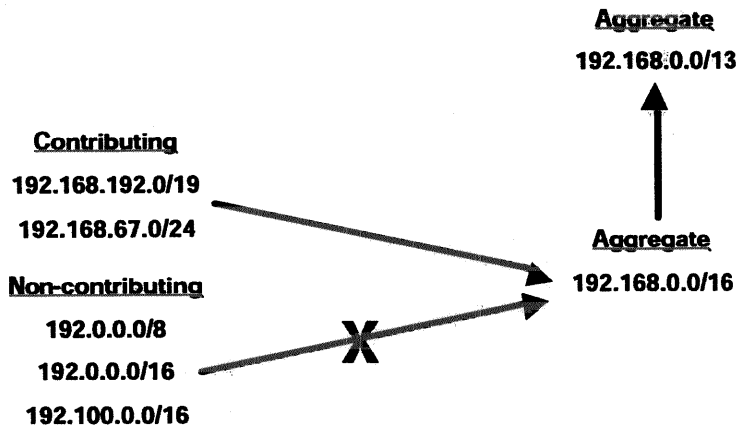
The contributing routes for a particular aggregate route can be seen by using the **show route extensive** CLI command. This will show the aggregate route and all of the currently contributing routes. In addition to listing what prefixes are actively contributing, you can also see what protocol placed that contributing route into the routing table.

In the example on the slide:

- The aggregate route 172.16.64/20 is active in the routing table (arrow).
- It has seven current contributing routes.
 - Some of the contributing routes are configured as static routes (proto Static).
 - Some of the contributing routes have been learned through BGP (proto BGP).

Aggregates Can Contribute

Aggregate routes can contribute to other less-specific aggregates



Copyright © 2001, Juniper Networks, Inc.

As previously mentioned, an active route can only contribute to a single aggregate route. This could potentially cause a problem where a number of /24 routes all contribute to a /16 route. If a less-specific /13 route were then configured, it would then have no /24 contributing routes since they have already been "assigned" to the /16 route.

With the JUNOS software, this potential issue is resolved by allowing an aggregate route to act as a contributing route to a less-specific aggregate route. In the slide, routes 192.168.192.0/19 and 192.168.67.0/24 have been "assigned" to aggregate route 192.168.0.0/16. If the aggregate route 192.168.0.0/13 is now configured, how can it ever become active? In this case, the 192.168.0.0/16 aggregate route contributes to the 192.168.0.0/13 aggregate route, activating it.

Controlling Contributing Routes

A policy can be used to determine what routes contribute to an aggregate

```

routing-options {
  aggregate {
    route 192.168/16 {
      policy only-certain-routes;
    }
  }
}

policy-options {
  policy-statement only-certain-routes {
    term find-a-route {
      from route-filter 192.168.0/24 orlonger;
      then reject;
    }
  }
}

```

Copyright © 2001, Juniper Networks, Inc.

By default, the JUNOS software will automatically assign routes to contribute to certain aggregate routes. But the network administrator has control over this default function through the use of a policy. Should it be desired that all possible contributing routes not be considered for assignment to a particular aggregate route, then a policy can be defined and applied to the aggregate route configuration.

In the example on the slide, a policy called **only-certain-routes** is created that rejects all routes within the 192.168.0/24 subnet. This policy is shown on the lower portion of the slide.

This policy is then applied to the aggregate route of 192.168/16 within the [edit routing-options aggregate] configuration hierarchy. The effect of this example would be that any potential contributing routes from within the 192.168.0/24 subnet would not be used.

Any other potential contributing route not rejected by the policy, however, would still be eligible to make the 192.168/16 aggregate active in the inet.0 routing table.

Generated Routes

Where we are going...

- **Configuring Generated Routes**
- **Generated Routes**
- **Primary Contributing Route**
- **Generated Route Example**

Copyright © 2001, Juniper Networks, Inc.

Subsequent slides will discuss generated routes and their configuration in the JUNOS software.

Configuring Generated Routes

- **Generated routes are similar to aggregate routes**
 - **Becomes active once one or more contributing routes are active**
- **Configured at the routing-options hierarchy level**

```
[edit]
routing-options {
  generate {
    defaults {
      generate-options;
    }
    route destination-prefix {
      policy policy-name;
      generate-options;
    }
  }
}
```

Copyright © 2001, Juniper Networks, Inc.

Generated routes are similar in nature and configuration to aggregate routes. They are manually configured on each router in the network. Generated routes become active in the routing table when at least one of the contributing routes (more specific routes) for the generated route is also active in the routing table.

Generated routes are often used as the *route of last resort* and differ from aggregate mainly in that generated routes are allowed to have an IP address next-hop.

Options that can be configured for a generated route include :

- **AS-Path**—Used if this route is intended to be redistributed into BGP and you wish to manually add values to the AS_PATH attribute.
- **Community**—Used if this route is intended for BGP and you wish to add community values to the route for use in your AS.
- **Metric**—If multiple routes share the same preference value, then the route with the best metric will become active in the routing table. Use this value to prefer one route over another in this case.
- **Policy**—By default, all possible more specific contributing routes can be used to activate a generated route. To alter this default, you can use a policy to accept or reject certain routes that should or should not be used.
- **Preference**—The default preference value of generated routes is 130. Use this option to alter the value of the generated routes.

Generated Routes

- The default next-hop is the next-hop of the primary contributing route
 - Discard is also a valid option
- Defaults section affects all generated routes
- Appear in the routing table as an aggregate route

```
routing-options {
  generate {
    defaults {
      metric 5;
    }
    route 172.16.64.0/20;
    route 172.16.80.0/20 discard;
  }
}
```

Copyright © 2001, Juniper Networks, Inc.

Unlike aggregate routes, the default next-hop value for a generated route is the IP next-hop address of the primary contributing route. The other possible next-hop value for generated routes is *discard*, which drops the packet from the network and does NOT send back an ICMP message.

So the difference between a generated route and an aggregate route is the default next-hop assigned. In every other respect, the two are identical. For this reason, generated routes appear in the inet.0 routing table simply as an aggregate route, also with a preference value of 130. In other words, there is no special name in the routing table for a generated route: a generated route is an aggregate route to the routing table.

Within the [edit routing-options generate] configuration hierarchy, the **defaults** section can contain generated route options. Any options configured within this section are applied to all generated routes on the router. In the example on the slide, the metric value has been set to 5. This means that all generated routes on the router will have that metric value.

Any single generated route can override the defaults section by configuring the same option within its specific route section of the configuration hierarchy. Any option defined specifically for a route overrides the same option defined within the **defaults** section.

NOTE

You can only configure one generated route per prefix.

Primary Contributing Route

- For generated routes, the primary contributor is the numerically smallest prefix value
- IP next-hop of the primary is inherited as the next-hop of the generated route

```
172.16.64.0/20 (1 entry, 1 announced)
*Aggregate Preference: 130
Nexthop: 10.222.9.2 via so-0/0/1.0, selected
State: <Active Int Ext>
Age: 21
Task: Aggregate
Announcement bits (2): 0-KRT 4-BGP_Sync_Any
AS path: I
Flags: Generate Depth: 0 Active
Contributing Routes (13):
    172.16.64.1/32 proto IS-IS
    172.16.65.0/24 proto BGP
    172.16.66.0/24 proto BGP
```

```
172.16.64.1/32 *[IS-IS/18] 03:26:35, metric 10, tag 2
> to 10.222.9.2 via so-0/0/1.0
```

Copyright © 2001, Juniper Networks, Inc.

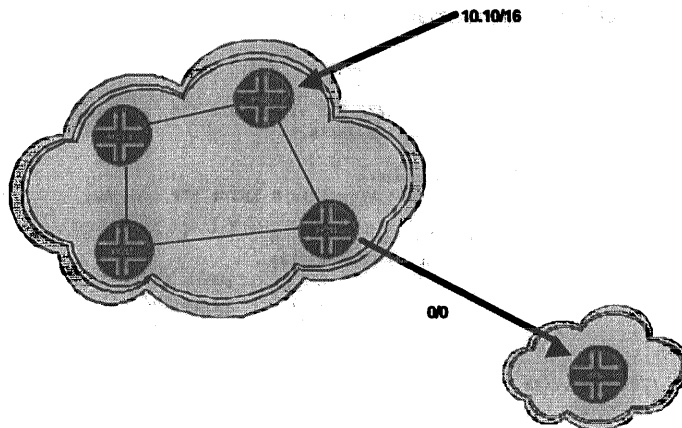
The primary contributing route is an important concept for a generated route. This is because the IP next-hop address of the primary contributing route is inherited by the generated route as its IP next-hop address.

The primary contributing route can be found by using the **show route extensive** CLI command and examining the output. It will be the contributing route with the smallest numerical prefix value. This primary contributor is *always listed first* in the command output.

In the example above, the generated route 172.16.64/20 has a primary contributing route of 172.16.64.1/32. Notice that both the primary contributor and the generated route share the same IP next-hop address of 10.222.9.2 via so-0/0/1.0.

Generated Route Example (I)

- ISP wants to send a default route to its customer
- What's the best way to get 0/0 to appear on the edge router?



Copyright © 2001, Juniper Networks, Inc.

One example of using a generated route would be if an ISP wants to send a default route to one of its customers. The use of a generated route here is better than using a simple static route to establish this default route.

For example, the ISP could configure a static route for the 0/0 network on the edge router leading to the customer and redistribute that route to the customer.

The problem with doing this with a static route is that the static route will always be active on the router even if there is no upstream connection for the ISP to actually route the traffic to. This method gets the customer traffic to the ISP even if there is no real place for the ISP to pass the 0/0 traffic off to.

What would be the easiest way to associate the 0/0 route with upstream connectivity?

Generated Route Example (II)

- Static is probably not the best solution
- A generated route with a policy might work
 - Becomes a "dynamic" static route with an IP next-hop

```
routing-options {  
  generate {  
    route 0.0.0.0/0 {  
      policy only-certain-routes;  
    }  
  }  
}  
policy-options {  
  policy-statement only-certain-routes {  
    term find-a-route {  
      from route-filter 10.10/16 exact;  
      then accept;  
    }  
    term nothing-else {  
      then reject;  
    }  
  }  
}
```

Copyright © 2001, Juniper Networks, Inc.

Static routes are not the best way to distribute 0/0 default routes to customers, since statics are always active regardless of connectivity.

A generated route with a policy applied might just do the job, however. In this case, the generated route becomes a type of "dynamic" static route with an IP next-hop.

By default, a generated route of 0/0 will use all routes in the routing table as potential contributing routes. This gives you the same net effect as a static route to 0/0, but without the drawbacks of the 0/0 static.

We can use a policy to control what routes from the routing table can contribute to a specific generated route. In the previous slide, there was an upstream route of 10.10/16 being received from the Internet. We can match only that one route in a policy called **only-certain-routes**. That policy can then be applied to the generated route within the [edit routing-options generate] section of the configuration hierarchy.

Once this is done, the 0/0 route will only appear in the inet.0 routing table when the 10.10/16 upstream route is also present. Should the 10.10/16 route disappear, the 0/0 route will also disappear. We have created a sort of "dynamic" static route.

Martian Routes

Where we are going...

- **Martian Filtering Policy**

Copyright © 2001, Juniper Networks, Inc.

Subsequent slides will discuss how the JUNOS software handles martian routing information.

Martian Filtering Policy

- Policies can often be used to reject unwanted routes
- These prefixes are not intended for Internet IP traffic

```
[edit policy-options]
policy-statement reject-unwanted-routes {
  term drop-these-routes {
    from {
      route-filter 0/0 exact;           # default
      route-filter 127/8 orlonger;      # Loopbacks
      route-filter 10/8 orlonger;       # RFC 1918 Reserved
      route-filter 172.16/12 orlonger;  # RFC 1918 Reserved
      route-filter 192.168/16 orlonger; # RFC 1918 Reserved
      route-filter 224/3 orlonger;      # Multicast
    }
    then reject;
  }
}
```

Copyright © 2001, Juniper Networks, Inc.

Policies are often used to reject unwanted routes and prevent them from appearing in the routing table. One important class of unwanted routes are known as *martian addresses* (as in "out of this world").

The keyword *martian* in the terminology of the Internet means routes that should never be present in a routing table and are not intended for live Internet traffic. In addition to reserved test addresses, private IP address from RFC 1918, and the default route of 0/0 are also usually considered to be martian routes.

Within the JUNOS software, there is a set of default martian routes configured which contains the following information:

- 0.0.0.0/8
- 127.0.0.0/8
- 128.0.0.0/16
- 191.255.0.0/16
- 192.0.0.0/24
- 223.255.255.0/24
- 240.0.0.0/4

Notice that none of the default martian routes contain the private IP addresses from RFC 1918. This is due to the way in which the JUNOS software uses the martian list. When a route is listed in the martians, it is never allowed to appear in the inet.0 routing table. Since many ISPs use the RFC 1918 address internally for network connectivity, they need to be allowed in the routing table. However, those private routes should not be accepted from a peer router in another AS. This is where a policy can be useful.

Advanced Policy

In the slide, we have created a policy that matches on the 0/0 default route, multicast routes, and the RFC 1918 private address ranges. The policy rejects those routes. When this policy gets applied as an import policy on a BGP session, it will effectively filter out these routes such that they can not enter the ISP network.

Route Filters

Where we are going...

- **Route Filters and Prefixes**
- **The Match Type Options**
- **Match Type Examples**
- **What Matches?**
- **Multiple Route Filters Defined**
- **Longest Match Lookup**
- **Route Filters & Other Match Criteria**
- **Route Filter Match Actions**
- **Test Your Knowledge**
- **When Order Does Count**

Copyright © 2001, Juniper Networks, Inc.

Subsequent slides will discuss route filter match options and actions, how to determine which route filter is used for a match, and exceptions to the route filter rules.

Route Filters and Prefixes

- An important role in routing policy is played by the *route filter*
- Route filters act on prefixes (IP addresses/ranges)
- Multiple route filters in a single policy term possible
- Special rules apply to route filter evaluation

```
policy-statement policy-name {  
  term term-name {  
    from {  
      route-filter prefix/prefix-length match-type <actions>;  
    }  
    then actions;  
  }  
}
```

Copyright © 2001, Juniper Networks, Inc.

Up to this point, we have described various ways to use match criteria in a policy. We have generally talked about match configuration statements such as protocols, interfaces, and so on. These will generally apply to and match a large number of routes. At times, it is very useful to match a specific route or a limited set of routes. This is the job of the route filter.

Route filters match specific IP address prefixes or ranges of prefixes.

It is possible to have multiple route filters in a single policy term. Since normal longest-match rules apply to route filters, it is usually not important in which order the route filters are configured as a match condition. As usual, new route filters are just added to the end of the list of other route filters.

Route filters are used in the **from** section of a policy term. The configuration syntax is shown in the slide. Unlike the other match criteria we have discussed, route filters have a special set of evaluation rules which we will discuss on the following slides.

The Match Type Option

Specifies type of match applied to destination prefix

Match Type	Match if...
exact	Prefix-length is <i>equal</i> to route's prefix length
orlonger	Prefix-length is <i>equal to or greater than</i> route's prefix length
longer	Prefix-length is <i>greater than</i> route's prefix length
upto	Route shares most significant bits (as set in prefix-length) and route's prefix length falls between prefix-length and prefix-length2
prefix-length-range	Route shares most significant bits and the prefix length is between the two lengths specified
through	Route falls exactly between first prefix/prefix-length and second prefix/prefix-length (list of exact matches)

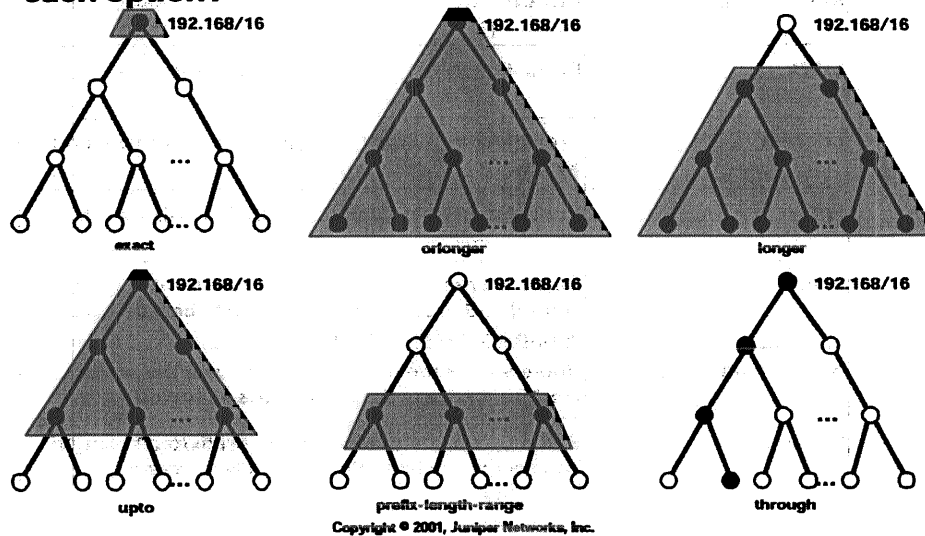
Copyright © 2001, Juniper Networks, Inc.

One of the required pieces of the route filter configuration is the *match-type*. There are currently six defined match-types which are displayed on the slide. Any one of these match-types can be used with a route filter.

The **match-type** keyword is what allows the route filter to distinguish between a single route or a grouping of routes. Each match-type performs a specific function as detailed in this slide and in the following slides.

What Matches?

Given a starting prefix of 192.168/16, what matches with each option?



Sometimes it is helpful to examine the potential routes in the IP address space in a graphical manner. This slide shows a portion of what is known as the "J-Tree." It is very similar in function to a *radix tree* that is used to represent many bit-oriented things in computing and communications.

The way to "read" the J-Tree is as follows: Start at the top of the tree with the 0.0.0.0/0 route. Evaluate the first bit position in the IP address -- it will either be a 0 or a 1. If it is a 0, move to the *left* on the tree. This is now the representation of 0.0.0.0/1. If the first bit is a 1, move to the *right* on the tree. This is now the representation of 128.0.0.0/1.

From each of these two established points, examine the next bit position in line (second overall bit). Again, move left or right depending on whether the next bit is a 0 or a 1. You will then have four data points: 0.0.0.0/2, 64.0.0.0/2, 128.0.0.0/2, 192.0.0.0/2. Continue in this fashion thirty more times and you will have mapped out all of the IP address space.

The slide above shows only a portion of the J-Tree which starts at 192.168.0.0/16 to show how each of the route filters in the previous slide finds routes to match.

Match Type Examples

Prefix	192.168/16 exact	192.168/16 or longer	192.168/16 longer	192.168/16 upto /24	192.168/16 prefix-length- range /18-20	192.168/16 through 192.168.16/20
192.0.0.0/8						
192.168.0.0/16	Passes	Passes		Passes		Passes
192.168.0.0/17		Passes	Passes	Passes		Passes
192.168.0.0/18		Passes	Passes	Passes	Passes	Passes
192.168.0.0/19		Passes	Passes	Passes	Passes	Passes
192.168.4.0/24		Passes	Passes	Passes		
192.168.5.4/30		Passes	Passes			
192.168.12.4/30		Passes	Passes			
192.168.12.128/32		Passes	Passes			
192.168.16.0/20		Passes	Passes	Passes	Passes	Passes
192.168.192.0/18		Passes	Passes	Passes	Passes	
192.168.224.0/19		Passes	Passes	Passes	Passes	
192.169.1.0/24						
192.170.0.0/16						

Copyright © 2001, Juniper Networks, Inc.

Using the match-type definitions from the previous slide, we can apply some routes to those match-types to see what routes match or do not match the route filter.

The routes displayed along the left side of the slide appear as they would in the output of a show route command. If a particular route has a "Passes" noted in its row, that signifies the route filter in the column matches that route. If the column is blank, the route-filter does **NOT** match the specified route.

Multiple Route Filters Defined

- Multiple route filters can be defined within a term
- A longest match is performed on the defined prefixes
 - Only that one filter is used as the match criteria

```

policy-statement try-this-one {
  term always-a-longest-match {
    from {
      route-filter 0.0.0.0/0 exact;
      route-filter 172.16.32/24 longer;
      route-filter 192.168.128/22 orlonger;
    }
    then accept;
  }
}
    
```

Longest
match
lookup

Copyright © 2001, Juniper Networks, Inc.

It is possible to define multiple route filters within a single policy term. Unlike the other match criteria discussed previously, in which routes match a particular attribute such as protocol, the evaluation of multiple route filters within a policy term uses special rules.

Basically, only *one* of the route filters is used as a match criteria for any particular term. The route filter that gets used is found by performing a longest-match lookup similar to what occurs during a route lookup. The route filter with the "best" match of prefix and bit-mask length to the candidate route is the **one** route-filter used as a match criteria.

Longest Match Lookup

- When performing a longest match lookup, only the prefixes defined are evaluated
- Do not evaluate the match-types
- How does 151.140.10.0/24 get evaluated by this policy?

```
policy-statement only-prefixes-count {  
  term dont-look-at-match-type {  
    from {  
      route-filter 151.140.0.0/16 upto /24;  
      route-filter 151.140.10.0/23 exact;  
    }  
    then accept;  
  }  
}
```

Copyright © 2001, Juniper Networks, Inc.

The longest-match lookup is performed *only* on the prefixes and bit-masks defined, and NOT on the match-types configured. The match-types are "Part 2" of the evaluation. This is worth examining in more detail.

In the slide, there are two route filters defined. Now, the candidate route of 151.140.10.0/24 is being evaluated by this policy and the multiple route filters. The first step of the evaluation is the longest-match lookup on the prefixes defined. In this policy, the prefixes are 151.140.0.0/16 and 151.140.10.0/23. The longest match results in 151.140.10.0/23 being the "best."

It is at this point that the defined match-types come into play. The longest-match route filter is then evaluated against the candidate route using the match-type. In this case, **151.140.10.0/23 exact** does NOT match the candidate route of 151.140.10.0/24. So this policy has no matches and the actions are skipped.

Route Filters & Other Match Criteria

- All match condition statements and the one longest match route-filter must apply for the action to be taken
- Below, a BGP prefix will never match the term
 - If that is desired, use `protocol [bgp ospf]`

```

policy-statement no-bgp-here {
  term more-complex-example {
    from {
      protocol ospf;
      route-filter 0.0.0.0/0 exact;
      route-filter 172.16.32/24 longer;
      route-filter 192.168.128/22 orlonger;
    }
    then accept;
  }
}

```

Longest
match
lookup

Copyright © 2001, Juniper Networks, Inc.

When multiple route filters are combined with the other match criteria previously discussed, the two evaluation concepts must be used together. First, the one longest-match route filter is found. Then the one longest-match is combined into the logical AND with the other match criteria defined. As before, *all* of the match criteria must be true, including the single selected route filter, for the candidate route to have any specified actions taken against it.

In the slide, only OSPF routes will ever have the chance to match the policy using the route filters. In order to have multiple protocols be evaluated, use the syntax shown above of `protocol [ospf bgp]`. When used as a match criteria in a policy, this type of syntax using the square braces is considered to be a logical OR statement. Now the route's protocol can be either OSPF or BGP, but not IS-IS or anything else.

Of course, even if the route is BGP or OSPF, the route filter longest match lookup might still not apply to the route and no actions taken in this term with regard to the candidate route.

Route Filter Match Actions

- If the route matches a filter, and an action is specified, the action applies and the "then" is ignored
- If the route matches a filter and no action is specified, the "then" action is taken

```
policy-statement stop-one-of-these {
  term multiple-filters-used {
    from {
      route-filter 192.168.18.0/24 exact reject;
      route-filter 192.168.100.0/24 longer {
        metric 10;
        accept;
      }
      route-filter 192.168.0.0/16 orlonger;
    }
    then accept;
  }
}
```

Copyright © 2001, Juniper Networks, Inc.

Route filters have the option of having actions defined specifically for that route filter. Typically, these are actions that differ from the actions found in the `then` section of a policy term, but this is not a requirement. These "special" or *immediate* actions are used differently than `then` actions applied previously.

When a particular route is found to have matched a route filter statement, the route filter is examined to determine if there are any specific actions defined along with the route filter. If there are specific actions defined, then those actions are taken immediately and **ANY** other actions specified within the policy term are skipped and ignored.

On the other hand, if the route matches a route filter and no immediate action is specified, the actions following the `then` statement are applied.

This type of configuration can be useful when there are a number of route filters defined in a policy term and where some sets of prefixes are to be accepted and while others are to be rejected. Of course, this can also be done by using multiple policy terms. The particular method you use is up to you. The JUNOS software policy framework is flexible enough to support both methods.

Test Your Knowledge

Apply these prefixes to the following policy route-filters.

What happens in each case?

- 10.0.67.43/32
- 10.0.55.2/32
- 192.168.1/17

```
term testing-1-2-3 {
  from {
    route-filter 10.0.0.0/16 orlonger accept;
    route-filter 10.0.67.0/24 orlonger;
    route-filter 10.0.0.0/8 orlonger reject;
    route-filter 192.168.0/17 orlonger reject;
    route-filter 192.168/16 longer accept;
  }
  then {
    metric 10;
    accept;
  }
}
```

Copyright © 2001, Juniper Networks, Inc.

When evaluating the routes in the slide against the policy shown, remember to first perform the longest-match lookup on the route filters using only the prefixes defined. After the longest-match is found, evaluate whether the candidate route matches the longest-match route filter AND the match-type defined. If the candidate route does match these conditions, then look to see if the route filter has any immediate actions defined for it specifically. If the route filter does, take those actions and stop processing this policy term. If the route filter does not have any immediate actions specified, take any actions defined in the **then** portion of the term.

The route 10.0.67.43/32 matches the **route-filter 10.0.67.0/24 orlonger** in the policy. It is the longest-match and the match-type contains the candidate route. There is no action specified in the route-filter, so the **then** actions are taken. This route gets **accepted and has the metric set to 10**.

The route 10.0.55.2/32 matches the **route-filter 10.0.0.0/16 orlonger accept** in the policy. It is the longest-match and the match-type contains the candidate route. There is an action specified in the route filter, so the **then** actions are skipped. This route gets **accepted**.

The route 192.168.1/17 matches the **route-filter 192.168.0/17 orlonger reject** in the policy. It is the longest-match and the match-type contains the candidate route. There is an action specified in the route filter, so the **then** actions are skipped. This route gets **rejected**.

When Order *Does* Count

- The order of some prefixes *is* important!
- Occurs when identical prefixes are defined
- Match types are then checked in order for a match
- This router-filter rejects the 10.0.0.0/8 route, but the 10.0.0.0/16 route has the next-hop changed to self
 - Since no "terminating" action, process the next term/policy

```

policy-statement this-is-different {
  term order-counts-here {
    from {
      route-filter 10.0.0.0/8 exact reject;
      route-filter 10.0.0.0/8 longer next-hop self;
    }
  }
}
    
```

Order
matters
here

Copyright © 2001, Juniper Networks, Inc.

Up to this point, the order of the route-filter statements in the term was not important since a longest-match lookup was performed. However, what happens when the longest-match lookup matches multiple route filters? This is the case in the policy in the slide. Both of the **route-filter** statements have a prefix defined of 10.0.0.0/8. In this case, the order of the route filters ***IS*** important. When the match-types are evaluated, the route filters get processed in a top-down fashion.

While this is a simplistic example, the concept holds true for a more complicated policy. Suppose that we had a policy with route-filters defined as:

```

route-filter 192.168.1/24 exact
route-filter 172.16.24/24 orlonger
route-filter 10.0/16 longer
route-filter 192.168.1/24 orlonger
route-filter 172.16.24/24 exact
route-filter 10.0/16 exact
    
```

The longest match is still performed and the multiple returned matches are then processed in order.

Review Questions

- **What is the function of the following routing tables?**
 - inet.0
 - inet.1
 - inet.2
 - inet.3
- **What is the difference between an aggregate route and a generated route?**
- **What are the Aggregator and Atomic Aggregator attributes used for in BGP?**
- **How is the primary contributing route of a generated route determined?**
- **When is the order of route-filter statements important?**

Copyright © 2001, Juniper Networks, Inc.

During this module we discussed:

- What the default JUNOS software route tables were and how they were populated.
- How to configure static, aggregate, and generated routes.
- The operation of route filters within a policy and how those filters affected routing information in an Autonomous System.



Juniper Networks Advanced Policy

Release 5.0, Revision 1

Module 3: Load Balancing

Copyright © 2001, Juniper Networks, Inc.

Objectives

- **JUNOS software load balancing options**
- **Configuring a policy to load balance all traffic**
- **Configuring a policy to load balance specific routes**
- **Default BGP load balancing behavior**

Copyright © 2001, Juniper Networks, Inc.

This module discusses:

- The JUNOS software load balancing options, including how a routing policy can be written to install two equal cost routes into the Packet Forwarding Engine forwarding table
- Configuring a routing policy that will load balance (e.g. distribute) all traffic passing through the router onto equal cost paths when such paths exist
- Configuring a routing policy that will load balance (e.g. distribute) only the traffic to specific destinations (prefixes) onto equal cost paths when such paths exist.
- The behavior of default BGP load balancing, which differs slightly from the load balancing behavior of other routing protocols.

JUNOS Software Load Balancing

Where we are going...

- **Why Choose Just One Route?**
- **Load Balancing Behavior**
- **Network Flows**
- **Per-Packet Traffic With Internet Processor I**
- **Per-Flow Traffic With Internet Processor II**
- **Load Balancing Example Network**
- **router1 Routing Tables**

Copyright © 2001, Juniper Networks, Inc.

Subsequent slides will discuss the default JUNOS software load balancing behavior, and the difference between per-packet and per-flow traffic.

Why Choose Just One Route?

- What if routing table has multiple, but equal cost, routes to the same destination prefix?
- Only one active route gets to go into the forwarding table
- JUNOS software chooses randomly one of the next-hop addresses to install into the forwarding table
- If destination "next-hop set" changes, random choice repeats
- JUNOS software can be configured to put all destination next-hop addresses into forwarding table as active route
 - This is called *per-packet load balancing*

Copyright © 2001, Juniper Networks, Inc.

Since the routing tables can learn of a route through multiple interfaces, it is possible to have more than one equal cost route to the same destination prefix. What should happen then?

Only one next-hop will be installed in the forwarding table as an active path.

The default load balancing behavior for the JUNOS software is to "load balance" across one route chosen at random from among the equal next-hops. This comes down to an issue of control. Network administrators know exactly where traffic is flowing on their networks at any point in time. So when there are multiple equal-cost paths in the routing-table, the JUNOS software randomly chooses one of the next-hops from the available next-hop set to be placed into the forwarding-table.

Should the information in the next-hop set change, the random choice gets repeated. The next-hop set could change due to the addition or subtraction of a next-hop. It could change due to the actual next-hop value changing for any of the possible paths.

However, network administrators are given policy controls to alter this load balancing default. A policy can be created to match all routes or only certain routes and an action of **load-balance per-packet** can be defined. This policy, when applied, will cause the routing table to place multiple next-hop values into the forwarding table for a given route. This is called per-packet load balancing.

Load Balancing Behavior

- Precise behavior of load balancing depends on IP ASIC version used in the router
- For Internet Processor (IP) ASIC, traffic is spread in a *random fashion* between the routers
 - Forwarding table does the balancing
 - Round-robin among maximum of 8 equal-cost paths
 - Load-balanced on a per-packet basis
- Internet Processor II ASIC divides traffic into individual *traffic flows*
 - Flows balanced among maximum of 16 equal-cost paths
 - Packets for each flow are kept on same output interface
 - Avoids issues with packet arrival sequence

Copyright © 2001, Juniper Networks, Inc.

The load-balancing policy that gets created in the JUNOS software will always contain the same configuration syntax of **load-balance per-packet**. But the actual behavior of the load balancing is determined by the Internet Processor (IP) ASIC found in the router.

For older routers still using the Internet Processor I ASIC, the balancing behavior is truly per-packet. The traffic is spread across the active routes in a random fashion, with the forwarding table itself enforcing the balancing. Each packet that matches a destination route gets forwarded across a different outbound interface in a round-robin fashion. The Internet Processor I ASIC can support up to 8 different equal-cost paths per destination route.

The Internet Processor II ASIC has different balancing characteristics. The maximum number of supported equal-cost paths rises to 16. The behavior of the load balancing is also different. With an Internet Processor II ASIC, the packets are forwarded based on traffic flows. All packets headed for a destination route that contain the same flow characteristics will all be forwarded out the same outbound interface. This avoids issues of packet arrival sequence and the need to Routing Engine-sequence packets, which can slow TCP operation considerably.

Network Flows

- **Maintaining a flow results in generally better network performance**
 - End-User Applications experience less delay
 - Common path through the network for implementing QoS
 - All protocols benefit: TCP, UDP, etc.
- **By default, flows in JUNOS software are defined by the uniqueness of three Layer 3 parameters**
 - Source IP address
 - Destination IP address
 - Incoming router interface
- **Additional Layer 4 parameters can be included in the calculation if desired**
 - Protocol
 - Source port
 - Destination port

Copyright © 2001, Juniper Networks, Inc.

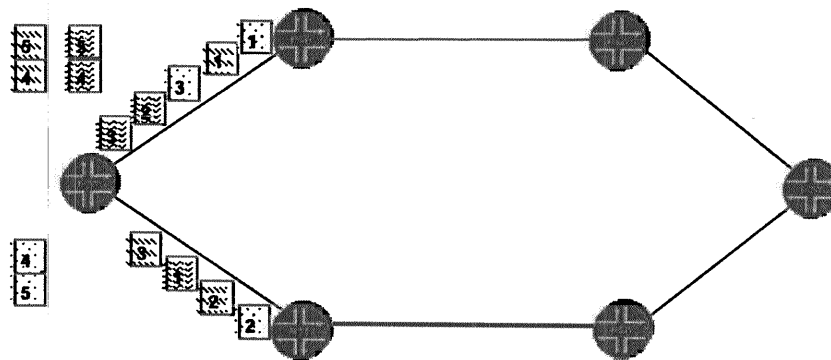
Within a networking environment, maintaining a flow of traffic between end stations results in a number of benefits. One benefit is that the packets generally arrive at the destination in the order they were sent. That way, the end station does not have to Routing Engine-order the packets and the applications experience less delay. In addition, since similar user traffic is using the same path through the network, network-wide policies such as Quality of Service (QoS) are easier to implement. All protocols, not just TCP, can benefit from the establishment of flows through a network.

The default JUNOS software characteristics that define a network flow are the IP Layer 3 parameters of Source IP Address, Destination IP Address, and the incoming interface on the router.

In addition, network administrators can also enable the router to include IP Layer 4 parameters such as Source port, Destination port, and Protocol in the flow calculation. To have both Layer 3 and Layer 4 information used, configure the **hash-key** under the [edit forwarding-options] section of the configuration hierarchy.

```
[edit forwarding-options]
hash-key {
  family inet {
    layer-3;
    layer-4;
  }
}
```

Per-Packet Traffic with Internet Processor I



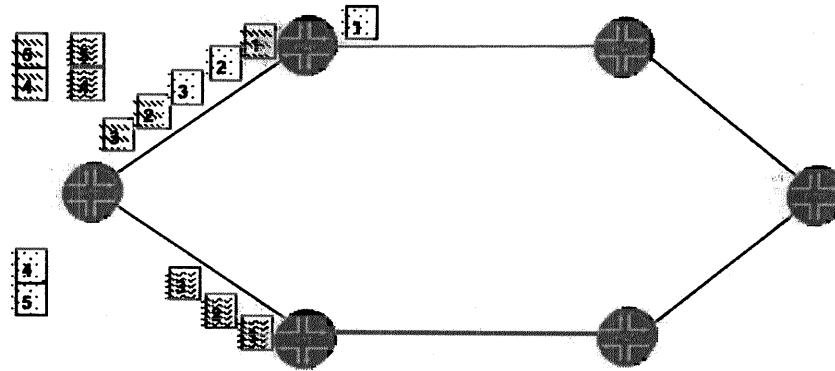
Copyright © 2001, Juniper Networks, Inc.

The slide shows the default per-packet load balancing behavior of the Internet Processor I ASIC found in older Juniper Networks routers. Notice that traffic from the three different network flows gets forwarded across both of the outbound interfaces.

On the slide, there are three sequences of five packets each destined for the same prefix. The packets arrived on different interfaces on the leftmost router and are to be sent to a destination beyond the rightmost router. Because Internet Processor I load balancing has been enabled, the packets can be sent out on both of the two equal cost paths, which are now both active and in the forwarding table.

Note that there is no attempt on the part of the router to output the associated packets on the same interface. The first packet of the sequence goes "up" and the next goes "down" in true round-robin fashion. So packets can easily arrive out of sequence, causing a reduction in throughput.

Per-Flow Traffic with Internet Processor II



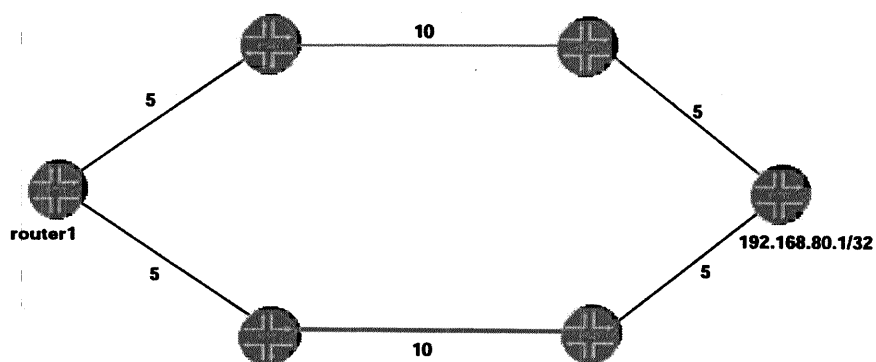
Copyright © 2001, Juniper Networks, Inc.

The slide shows the default *per-flow* load balancing behavior of the Internet Processor II ASIC found in newer Juniper Networks routers. Notice that traffic from the three different network flows gets forwarded across both of the outbound interfaces, as before, but in a different fashion.

On the slide, there are three sequences of five packets each destined for the same prefix, as on the previous slide. Because Internet Processor II load balancing has been enabled, the packets can be sent out on both of the two equal cost paths, which are now both active and in the forwarding table.

Note that in contrast to Internet Processor I default load balancing, the packets are now associated by flow with a particular interface. If the first packet of one of the sequences goes "up" then the rest of the packets associated with that flow must go "up." So packets must arrive in sequence, since they all follow the same path through the network. The net result is better throughput, more stable delays, and more efficient network operation.

Load Balancing Example Network



Copyright © 2001, Juniper Networks, Inc.

The slide shows a sample network that will be used in the following sequence of slides to show the effects of the JUNOS software load-balancing behavior.

We will always be looking at the network from the perspective of router1 on the left-hand side of the slide.

Notice that there are two possible paths to destination prefix 192.168.80.1/32 and that both of the paths have a total metric value of 20.

router1 Routing Tables

```
user@router1> show route 192.168.80/20 terse
```

```
inet.0: 84 destinations, 84 routes (84 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both
```

A Destination	P	Prf	Metric 1	Metric 2	Next hop	AS path
* 192.168.80.1/32	I	18		20	10.222.10.2	
					>10.20.20.1	

```
user@router1> show route forwarding-table
```

```
Routing table:: inet
```

```
Internet:
```

Destination	Type	RtRef	Nexthop	Type	Index	NhRef	Netif
192.168.80.1/32	user	0	10.20.20.0	wcst	26	30	so-0/0/3.0

Copyright © 2001, Juniper Networks, Inc.

This slide shows the routing and forwarding table entries for the destination prefix in the example, 192.168.80/20.

Note that router1 has an IS-IS route in inet.0 for 192.168.80.1/32 with a metric of 20. This particular route has two possible next-hop values (10.222.10.2 and 10.20.20.1) that can be used to forward traffic towards the destination.

The default JUNOS software behavior (that is, without load balancing) is to randomly choose one of the next-hops to be included in the forwarding table. In this case, that single next-hop is 10.20.20.1. How can you tell?

Notice that in inet.0 the 10.20.20.1 next-hop is marked with a ">" symbol. This signifies that this next-hop was the random choice placed into the forwarding table. This choice can be verified by looking at the forwarding table using the **show route forwarding-table** CLI command.

In the forwarding table, 192.168.80.1/32 does in fact have 10.20.20.1 using output interface so-0/0/3.0 as a next-hop value. And 10.222.10.2 is nowhere to be found in the forwarding table associated with 192.168.80.1/32.

Load Balancing Policies

Where we are going...

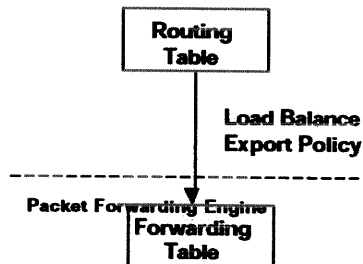
- **Forwarding Table Export**
- **Load Balancing For All Routes**
- **router1 Tables After Export**
- **Load Balancing For Some Routes**

Copyright © 2001, Juniper Networks, Inc.

Subsequent slides will discuss how a policy can be used to alter the default JUNOS software load balancing behavior, and provide some example policies that accomplish that goal.

Forwarding Table Export

- Active routes in the routing table are *exported* to the forwarding table in the Packet Forwarding Engine (Packet Forwarding Engine)
- A policy to control the load balancing behavior is applied as an export policy



Copyright © 2001, Juniper Networks, Inc.

Information in the forwarding table is derived from the total set of data in the routing table. Since this information is "removed" from the routing table it is said to be *exported* from the routing table (recall that export and import are from the respect of the routing table) to the forwarding table in the Packet Forwarding Engine.

Although it is never seen in a configuration, there is actually a type of default policy in effect between routing table and forwarding table that matches all destination routes to be installed in the forwarding table with an *accept one next-hop value* action. To alter the JUNOS software load balancing defaults, create a load balancing policy and apply this policy as an export policy to the forwarding table.

Load Balancing For All Routes

- Create a policy with no *from* statement to load balance all routes
- No "accept" action needed since it is the default action

```
[edit policy-options]
policy-statement load-balance-me {
  term balances-all-routes {
    then {
      load-balance per-packet;
    }
  }
}
```

```
[edit routing-options]
forwarding-table {
  export load-balance-me ;
}
```

Copyright © 2001, Juniper Networks, Inc.

To load balance traffic for all routes in the routing table, create a policy similar to the one in the slide above. Since no *from* statement was configured, all possible routes will match the policy. The action for the policy is **load-balance per-packet**. This action is the same for both the Internet Processor I ASIC and the Internet Processor II ASIC. However, the actual balancing behavior is determined by the ASIC in use as discussed previously. In other words, the Internet Processor II ASIC will still load balance by flow, even though the action is the same in both cases.

To perform load balancing, this policy must be applied as an *export* policy under the `[edit routing-options]` configuration hierarchy to the forwarding table. Once committed, this configuration will start to load balance traffic for all routes which have multiple equal-cost paths through the network.

This load balancing policy is only effected on this single router. To have all routers in the network performing similar function, create and apply a similar policy on each router.

router1 Tables After Export

```
user@router1> show route 192.168.80/20 terse
```

```
inet.0: 84 destinations, 84 routes (84 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both
```

A	Destination	P	Prf	Metric 1	Metric 2	Next hop	AS path
*	192.168.80.1/32	I	18		20	10.222.10.2	
						>10.20.20.1	

```
user@router1> show route forwarding-table
```

```
Routing table:: inet
```

```
Internet:
```

Destination	Type	RtRef	Nexthop	Type	Index	NhRef	Netif
192.168.80.1/32	user	0		ulst	30	14	
			10.222.10.0	ucst	20	19	so-0/0/0.0
			10.20.20.0	ucst	26	22	so-0/0/3.0

Copyright © 2001, Juniper Networks, Inc.

The slide shows the routing table and forwarding tables for router1 after the load balancing policy has been applied. The routing table still has an IS-IS route in inet.0 for 192.168.80.1/32 with a metric of 20. The route has two equal-cost paths through the network that can be used, as before.

However, when the forwarding table is viewed, the 192.168.80.1/32 route has both 10.20.20.1 on interface so-0/0/3.0 and 10.222.10.2 on interface so-0/0/0.0 as next-hop values. We are now load balancing traffic across the two equal cost paths.

There is a very important point to be made about the load balancing and the routing table. Look at the inet.0 routing table in more detail. Notice that it still has the 10.20.20.1 next-hop marked with a ">" symbol. Also notice that the 10.222.10.2 next-hop is not marked. While this might seem strange at first, remember where the policy is located in this process – between the routing table and the forwarding table. The routing table has no knowledge of the policy and still performs its default action of a random next-hop choice. It is only when the forwarding table is viewed (after the policy has been evaluated) that the effects of the load balancing policy can be seen.

Load Balancing For Some Routes

The *from* will apply load balancing only to the routes that match the conditions

```
[edit policy-options]
policy-statement load-balance-me {
  term balances-all-routes {
    from {
      route-filter 192.168.10/24 orlonger;
      route-filter 10.114/16 orlonger;
    }
    then {
      load-balance per-packet
    }
  }
}
```

```
[edit routing-options]
forwarding-table {
  export load-balance-me
}
```

Copyright © 2001, Juniper Networks, Inc.

The load balancing policy on the earlier slide can be modified so that only certain routes, not all routes, in the routing table get load balanced. All routes are balanced when there is no **from** statement in the policy, so all routes match the policy. To selectively load balance, include a **from** statement with one or more route filters. Of course, it is still necessary to apply the policy as an export policy to the forwarding table.

In this example, two route filters have been defined. Any routes matching these route filters will get load balanced. All other routes will have the *single* next-hop placed in the forwarding table according to the default behavior of the router – random selection of one next-hop.

The policy created here is just that—a policy. **ANY** of the policy match criteria can be used in this policy to decide what routes get load balanced and what routes do not get load balanced. Again there is maximum control and flexibility over the behavior of the router.

BGP Load Balancing

Where we are going...

- **Default BGP Load Balancing**
- **BGP Load Balancing Network**
- **router1 inet.0**
- **router1 Forwarding Table**
- **Combining The Concepts**
- **router1 After Policy**

Copyright © 2001, Juniper Networks, Inc.

Subsequent slides will discuss the default JUNOS software load balancing behavior for BGP routes, and how that default can be combined with the destination load balancing behavior from previous slides.

Default BGP Load Balancing

- Default load balancing for BGP is a little different
- If multiple equal-cost paths exist to the BGP Next-Hop attribute, then the total amount of prefixes received from that peer will be balanced across the multiple paths (per-prefix load balancing)
- Each BGP prefix still uses *only one* path for all traffic
- Example:
 - 100 routes are advertised from Peer A to Peer B
 - Peer A is performing next-hop-self
 - Peer B has multiple equal cost paths to Peer A lo0 address
 - The 100 routes are evenly distributed across the equal cost paths to Peer A

Copyright © 2001, Juniper Networks, Inc.

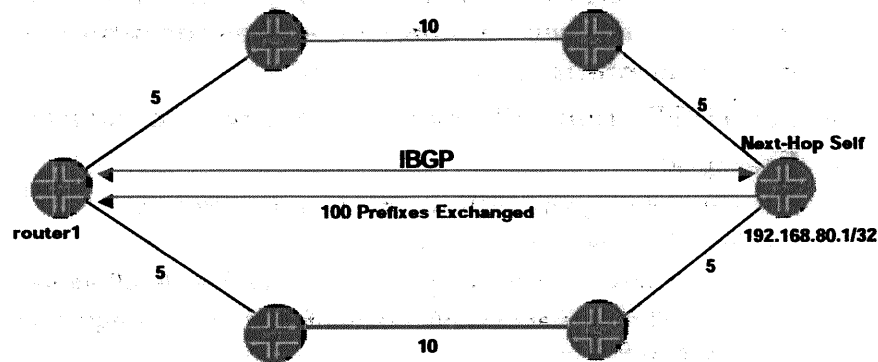
When looking at BGP routes in a network, there is another JUNOS software default load balancing action at work. The default BGP load balancing is slightly different than the load balancing examined earlier, which applies to IGPs.

The default BGP load balancing is known as *per-prefix load balancing*. Pre-prefix load balancing occurs when routes are received from an internal BGP peer and some of those routes have identical BGP Next-Hop attributes (an IP address). When the router performs the usual recursive route table lookup to find the BGP Next-Hop attribute value, the router might find that there are multiple equal-cost paths available to that IP address. In this case, the total number of received BGP routes that fit this criteria are spread across the available network paths.

However, each individual BGP route still uses *only one* path through the network.

For example, suppose that 100 routes are advertised by IBGP from router Peer A to router Peer B. Peer A is performing a next-hop-self, so Peer B will now have multiple equal cost paths to Peer A's lo0 IP address. In this case, the 100 routes on Peer B to Peer A's lo0 address will be "evenly" distributed across the equal cost paths to Peer A (a completely numerically equal distribution of the routes is not guaranteed, especially if routes are withdrawn).

BGP Load Balancing Network



Copyright © 2001, Juniper Networks, Inc.

The slide shows the same basic load balancing network with IBGP running between the leftmost and rightmost routers. The leftmost router1 still has two possible network paths to 192.168.80.1/32. The **load-balance per-packet** policy configured and applied previously is **NOT** in effect at this time.

The router on the right side of the network is sending to router1 its BGP routes with the BGP Next-Hop attribute set to 192.168.80.1 (the loopback address of the router on the right hand side of the slide). Since router1 has multiple equal cost paths to 192.168.80.1/32, the 100 BGP routes will be somewhat evenly distributed across those two paths.

For example, applying BGP load balancing might result in 52 routes using the upper path to the rightmost router, while 48 routes are using the bottom path to the rightmost router.

router1 inet.0

```
user@router1> show route 192.168.80/20 terse
```

```
inet.0: 84 destinations, 84 routes (84 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both
```

A	Destination	P	Prf	Metric 1	Metric 2	Next hop	AS path
*	192.168.80.0/24	B	170	100	5	>10.222.10.2 10.20.20.1	(65000) I
*	192.168.80.1/32	I	18	20		10.222.10.2 >10.20.20.1	
*	192.168.81.0/24	B	170	100	5	10.222.10.2 >10.20.20.1	(65000) I
*	192.168.82.0/24	B	170	100	5	10.222.10.2 >10.20.20.1	(65000) I
*	192.168.83.0/24	B	170	100	10	>10.222.10.2 10.20.20.1	(65000) I
*	192.168.84.0/24	B	170	100	10	10.222.10.2 >10.20.20.1	(65000) I
*	192.168.85.0/24	B	170	100	10	10.222.10.2 >10.20.20.1	(65000) I

Copyright © 2001, Juniper Networks, Inc.

The slide shows the inet.0 routing table of router1 after the BGP routes have been received, but before any *per-packet* load balancing has been applied.

Notice that some of the BGP routes are using 10.222.10.2 for a next-hop and some of the routes are using 10.20.20.1 as a next-hop. Each of the BGP routes knows about the multiple possible next-hops, but each individual route is only using one of them to forward traffic. This is the default behavior of the router. The next-hop choice is random and is independent of the next-hop used to reach 192.168.80.1/32. Notice that the IS-IS route is using 10.20.20.1 for its next-hop.

router1 Forwarding Table

```
user@router1> show route forwarding-table
```

```
Routing table:: inet
```

```
Internet:
```

Destination	Type	RtRef	Nexthop	Type	Index	NhRef	Netif
192.168.80.0/24	user	0	10.222.10.0	ucst	20	23	so-0/0/0.0
192.168.80.1/32	user	0	10.20.20.0	ucst	26	30	so-0/0/3.0
192.168.81.0/24	user	0	10.20.20.0	ucst	26	30	so-0/0/3.0
192.168.82.0/24	user	0	10.20.20.0	ucst	26	30	so-0/0/3.0
192.168.83.0/24	user	0	10.222.10.0	ucst	20	23	so-0/0/0.0
192.168.84.0/24	user	0	10.20.20.0	ucst	26	30	so-0/0/3.0
192.168.85.0/24	user	0	10.20.20.0	ucst	26	30	so-0/0/3.0

Copyright © 2001, Juniper Networks, Inc.

This slide shows the forwarding table on router1 after the BGP routes have been received, again before any *per-packet* load balancing has been applied.

Each of the destinations in the forwarding table is using a *single* next-hop to forward traffic, even though the next-hops are using both 10.222.10.0 and 10.20.20.0 at random. This is not yet really load balancing for each individual prefix..

Combining The Concepts

The load balancing policy for all routes can be used to forward traffic for *all* received BGP routes across *all* of the possible next-hop interfaces

```
[edit policy-options]
policy-statement load-balance {
  term all-routes {
    then {
      load-balance per-packet;
    }
  }
}

[edit routing-options]
forwarding-table {
  export load-balance;
}
```

Copyright © 2001, Juniper Networks, Inc.

In order to load balance traffic for either a single BGP prefix or all of the BGP prefixes, it is necessary to configure and apply a load balancing policy similar to the one used earlier. In fact, this policy is essentially identical to the earlier policy used to balance all routes.

The difference is load balancing behavior is strictly with regard to how BGP load balances, and not policy configuration or application issues.

So this policy will cause *all* possible routes on the router, including all the received BGP routes, to be load balanced across *all* of the possible next-hop values on the router.

router1 After Policy

```
user@router1> show route forwarding-table
```

```
Routing table:: inet
```

```
Internet:
```

Destination	Type	RtRef	Nexthop	Type	Index	NhRef	Netif
192.168.80.0/24	user	0	10.222.10.0	ulst	30	14	
			10.20.20.0	ucst	20	19	so-0/0/0.0
192.168.80.1/32	user	0	10.222.10.0	ucst	26	22	so-0/0/3.0
			10.20.20.0	ulst	30	14	
192.168.81.0/24	user	0	10.222.10.0	ucst	20	19	so-0/0/0.0
			10.20.20.0	ucst	26	22	so-0/0/3.0
192.168.82.0/24	user	0	10.222.10.0	ulst	30	14	
			10.20.20.0	ucst	20	19	so-0/0/0.0
192.168.83.0/24	user	0	10.222.10.0	ucst	26	22	so-0/0/3.0
			10.20.20.0	ulst	30	14	
192.168.84.0/24	user	0	10.222.10.0	ucst	20	19	so-0/0/0.0
			10.20.20.0	ucst	26	22	so-0/0/3.0

Copyright © 2001, Juniper Networks, Inc.

The slide shows the forwarding table (not the routing table) on router1 after the BGP routes have been received and the **load-balance per-packet** action has been defined in a policy.

Each of the destinations in the forwarding table is using all possible next-hop values to forward traffic instead of just one as before. In this example, there are only two paths, but the Internet Processor II ASIC will balance up to 16 paths.

Review Questions

- What is the default "load balancing" done by the Juniper Networks router?
- How does per-packet load balancing work with the *Internet Processor I* ASIC?
- How does per-packet load balancing work with the *Internet Processor II* ASIC?
- How does load balancing differ with regard to BGP routes?
- If 200 routes are load balanced over two paths with BGP, would the split always be 100 and 100 for each?

Copyright © 2001, Juniper Networks, Inc.

During this module we discussed:

- The JUNOS software load balancing options, including how a routing policy could be written to install two equal cost routes into the Packet Forwarding Engine forwarding table
- Configuring a routing policy that would load balance (e.g. distribute) all traffic passing through the router onto equal cost paths when such paths exist
- Configuring a routing policy that would load balance (e.g. distribute) only the traffic to specific destinations (prefixes) onto equal cost paths when such paths exist.
- The behavior of default BGP load balancing, which differs slightly from the load balancing behavior of other routing protocols.



Juniper Networks Advanced Policy

Release 5.0, Revision 1

Module 4: Policy Processing

Copyright © 2001, Juniper Networks, Inc.

Objectives

- Describe the JUNOS software policy processing
- Configure a prefix list
- Explain the operation of configuration groups
- Describe how policy subroutines operate
- Explain how policy expressions alter the default policy processing sequence

Copyright © 2001, Juniper Networks, Inc.

This module discusses:

- The way that the JUNOS software processes policies when there is more than one policy to apply.
- How to configure a prefix list that can be used to ease the maintenance of customer routes.
- How configuration groups can be used to make the application of prefix lists simpler and easier.
- The behavior of routing policy subroutines, which can be used in some instances to make the maintenance of complex policies easier to perform.
- The use of policy expressions such as a logical AND or OR to alter the default flow of policy application.

Policy Chains

Where we are going...

- **The Need For Multiple Policies**
- **Multiple Policies**
- **Invoking Multiple Policies**

Copyright © 2001, Juniper Networks, Inc.

Subsequent slides will discuss why multiple policies might be needed in an ISP network and how the JUNOS software evaluates those policies.

The Need For Multiple Policies

- **BGP routers at the edge of an ISP must usually implement more than one import policy filter**
- **Guarding against "martians" is a must**
 - Defined in RFC 1812
 - No 0.0.0.0/0 exact, loopbacks, reserved blocks, Multicast or Class E, etc. should be in a unicast routing table
- **Do not accept routes to local address space**
- **Good idea not to accept many "long prefix" routes**
 - Aggregates save update bandwidth and route table space
 - For example, reject anything more specific than /20
- **How can all three policies be applied properly?**

Copyright © 2001, Juniper Networks, Inc.

The use of the JUNOS software policy framework in an individual network will vary greatly from its use in a different network. Each network design and policy goals will factor into the decision of what pieces of the policy framework to use and how to best use them. Some networks might find a need for multiple different policies in their network, especially for BGP routers at the edge of the network. Each individual policy accomplishes a specific task. These policies can then be combined together on an as-needed basis to accomplish a larger policy goal in the network. The slide lists some possible uses for individual policies in a network. While these may not be used in every network in the Internet, these policies (or variations thereof) are quite popular.

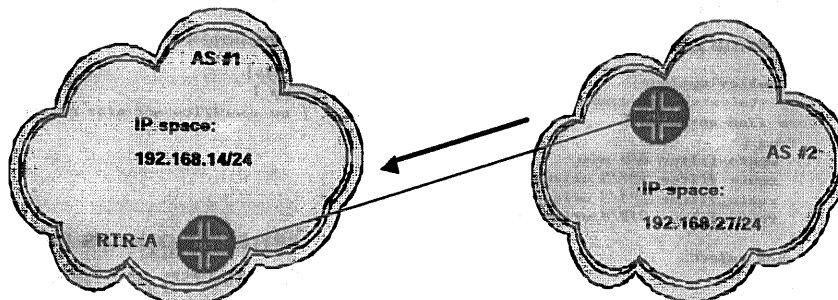
Guarding against martian routes, as defined in RFC 1812, is required. Also, there should be no 0.0.0.0/0 exact route, nor loopbacks, nor reserved blocks (RFC 1918), nor multicast addresses (Class D), nor Class E (experimental) addresses in a unicast routing table on any router.

Routers should never accept routes to their own local IP address space. How could another router or ISP have a better route to a network that is local or a customer?

It is also a good idea not to accept many "long prefix" routes (just what is "long" is left up to the network administrator). Aggregation to shorter prefixes saves bandwidth for updates and space in the routing tables. For example, a routing policy might reject any routes more specific (longer) than a /20.

So it is quite common to ask a router to reject martians and related addresses, reject local routes, and reject specifics when a shorter aggregate is also used. But how is it possible to formulate routing policy within the framework to apply all three of these separate actions correctly?

Multiple Policies



RTR A will not accept on this interface:

1. Any routes to 192.168.14/24 (or longer)
2. Routes more specific than 192.168.27/24
3. Martians

Could be combined into one multi-term policy, but using multiple policies gives flexibility for multiple interfaces and applications

Copyright © 2001, Juniper Networks, Inc.

In this example of multiple policy use, the network administrators of Autonomous System (AS) 1 have decided on some routing policy goals for their network. They are:

- Not to accept any routes that belong to the 192.168.14/24 subnet block, which is their own.
- Not to accept any routes with a prefix length of /24 or longer that also belong to the 192.168.27/24 subnet block.
- Not to accept any martian routes into the network such as the default route, the loopback address space, multicast networks, and the experimental networks of the 240.0.0.0/4 network space.

While these policy goals could certainly be applied and configured as a single, multi-term policy, the administrators of AS 1 made them three separate policies for reasons of flexibility and ease of maintenance.

Invoking Multiple Policies

Routing protocol can invoke any defined policies

#1. Define policies

```
[edit policy-options]
policy-statement no-unwanted {
  term find-unwanted {
    from {
      route-filter 0/0 exact;
      route-filter 127/8 orlonger;
      route-filter 224/3 orlonger;
      route-filter 240/4 orlonger;
    }
    then reject;
  }
}
policy-statement not-mine {
  term find-mine {
    from route-filter 192.168.14/24 orlonger;
    then reject;
  }
}
policy-statement no-specifics {
  term find-specifics {
    from route-filter 192.168.27/24 longer;
    then reject;
  }
}
```

#2. Apply policies

```
[edit protocols]
protocols bgp {
  import [ no-specifics not-mine no-unwanted ];
}
```

What happens when the following prefixes arrive?

- 0/0
- 127/8
- 192.168.14/24
- 192.168.27/24

Copyright © 2001, Juniper Networks, Inc.

This is the configuration syntax for the three policy goals listed in a previous slide. Once defined, these three policies are all applied as an *import* BGP policy at the global BGP level. The evaluation of these policies will be accomplished using the JUNOS software default policy evaluation of *left to right* processing as the policies appear in the *import* statement.

In other words, when the prefixes on the slide arrive in the order listed:

- The 0.0.0.0/0 default route will be rejected by the **no-unwanted** policy.
- The 127.0.0.0/8 route will be rejected by the **no-unwanted** policy.
- The 192.168.14/24 route will be rejected by the **not-mine** policy.
- The 192.168.27/24 route does not match any of the defined policies so it gets evaluated by the BGP default policy. The default policy for BGP will accept the route.

Prefix Lists

Where we are going...

- **Prefix Lists in Policy**
- **Configuring and Applying Prefix Lists**
- **Configuration Groups in Policy**
- **Configuring Groups**
- **Applying Groups**

Copyright © 2001, Juniper Networks, Inc.

Subsequent slides will discuss how a prefix list can be used in a policy as well as an alternative method known as a configuration group.

Prefix Lists in Policy

- **[edit policy-options]** allows for the creation of prefix lists of IP addresses
- Prefixes are interpreted by the policy as a list of exact match conditions
- Action applies to all prefixes that cause a match
- Can be used in many policies (ease of maintenance)
- Provides a single place to maintain customer routes

Copyright © 2001, Juniper Networks, Inc.

When an Internet Service Provider (ISP) wants to perform common policy actions on their customer routes, it needs to be able to identify those routes throughout the AS. There are a number of different ways to accomplish this identification with one of them being the use of route filters. After all, the ISP administrators know who the customers are and can identify them by their assigned address space. The problem here is one of scalability — adding a new route filter statement to five or ten policies can be time consuming.

The JUNOS software routing policy framework offers network administrators multiple different ways to ease this scalability issue. One of those ways is the *prefix list*. A prefix list is a set of routes defined by the network administrator. The big difference here is that those routes get defined one time — in the prefix list. This prefix list is then referenced from within multiple policies.

The prefix list gets defined within the **[edit policy-options]** configuration hierarchy. It is given a name and is then assigned the prefixes that "belong" to it. When a prefix list is referenced from within a policy, the routes in the list are evaluated by the policy as a *series of exact route filter statements*.

A match on any one of the routes within the prefix list will cause the action specified in the *then* portion of the policy term to be taken. So the action applies to *all* prefixes in the list that cause a match.

A prefix list can be used in many policies. This allows for easy maintenance of routes that might require special handling (such as customer routes) and gives network administrators a single place to maintain their customer routes.

Configuring and Applying Prefix List

- Prefix lists are not route filters!
- No match-types get defined
- Can only apply to *from* policy sections

```
[edit policy-options]
prefix-list known-dir-bcast-sites {
  10.3.4.6;
  10.2.0.0/16;
  192.168.1.0/24;
}
prefix-list known-okay-sites {
  172.8.0.3;
  10.10.0.0/16;
  192.168.12.0/24;
}
```

```
[edit policy-options]
policy-statement allow-ok-policy {
  term okay-from-25 {
    from {
      prefix-list known-okay-sites;
    }
    then accept;
  }
  term reject-bcasts {
    from {
      prefix-list known-dir-bcast-sites;
    }
    then reject;
  }
}
```

Copyright © 2001, Juniper Networks, Inc.

This slide defines two separate prefix lists. A few example routes have been configured under each of the prefix lists. As in all other aspects of the JUNOS software, the absence of a bit-mask is interpreted as a /32 bit-mask.

These prefix lists are then referenced from within two separate policy terms as match criteria. The policy terms see the prefix lists as a series of **route-filter** statements with a match type of *exact*. When any of the routes in the prefix list are matched, the action in the **then** section of the policy term is taken.

However, prefix lists are not route filters. There are significant differences.

For example, no **match-types** can be defined for prefix lists. Prefix lists always apply an exact match to the prefix.

Also, prefix lists can only be applied to the **from** section of a policy or policy term.

Configuration Groups in Policy

- Configuration groups can be used to replicate the functionality of a prefix list
- Groups are a function of the JUNOS software configuration CLI
- Represents a list of route filters and match-types within any policy, so differs from prefix list
- Snippets of configuration used in multiple places
- Policy matching, actions, and processing are not changed when groups are used
- Can be used in many policies (ease of maintenance)
- Provides a single place to maintain customer routes

Copyright © 2001, Juniper Networks, Inc.

There is another method within the JUNOS software for performing route maintenance tasks similar to a prefix list. This feature is known as a *configuration group*.

While configuration groups are not a JUNOS policy framework specific feature, but a function of the JUNOS software CLI, the routing policy framework can make use of them.

Like a prefix list, a configuration group is defined and given a name. The group is then referenced from within a policy. Unlike the prefix list, however, the configuration group is not evaluated as a series of exact route filters. The configuration group can have any combination of route filter statements and match-types it wishes. This is due to the way in which the configuration groups operate.

Configuration groups are quite simply snippets of the JUNOS software configuration hierarchy used in multiple places. When the group gets applied, the hierarchy searches through the group definition looking for statements applicable for use. Only statements within the group that "belong" to the current hierarchy level or below are used. All other group statements are ignored.

The use of configuration groups in no way changes the routing policy matching, actions, or processing behavior.

The major advantage of using configuration groups is that once defined, the groups can be used in many policies. This allows for easy maintenance of routes that might require special handling (such as customer routes) and gives network administrators a single place to maintain their customer routes.

Configuring Groups

- Configuration groups must be defined and then applied
- Regular route-filter statements are used
- The wildcard (*) is used to represent any policy or term name

```
[edit groups]
customer-routes {
  policy-options {
    policy-statement <*> {
      term <*> {
        from {
          route-filter 192.168.1.0/24 exact;
          route-filter 172.16.0.0/16 orlonger;
          route-filter 10.10.0.0/16 upto /24;
        }
      }
    }
  }
}
```

Copyright © 2001, Juniper Networks, Inc.

As expected, configuration groups must be first defined and then applied. Configuration groups are defined under the [edit groups] portion of the CLI hierarchy.

Regular **route-filter** statements (which include match-types) are used in configuration groups. So **exact**, **orlonger**, **upto**, and so on are all allowed.

The wildcard symbol – the * (asterisk) within angled brackets – is used to represent any policy or term name in which the configuration group is applied. This might sound confusing, so an example of the use of the wildcard symbol will be helpful.

The slide shows a configuration group defined called **customer-routes**. This group looks very similar to a routing policy and contains some route filters with varying match-types defined. The policy and term names have been configured as <*>, which is the wildcard notation. When the configuration group is applied and gets evaluated, the wildcard will be seen as any possible policy and any possible term. This allows this particular configuration group to be used in any policy where the customer routes defined within the configuration group are needed.

Applying Groups

- Groups are applied with **apply-groups <name>** command
- Configuration inherits statements found within the group
- To "see" the actual statements, use a pipe (**|**) command

```
[edit policy-options]
policy-statement allow-good-routes {
  term customers-okay {
    apply-groups customer-routes;
    then accept;
  }
}
```

```
[edit]
user@host# show policy-options | display inheritance | except ##
policy-statement allow-good-routes {
  term customers-okay {
    from {
      route-filter 192.168.1.0/24 exact;
      route-filter 172.16.0.0/16 orlonger;
      route-filter 10.10.0.0/16 upto /24;
    }
    then accept;
  }
}
```

Copyright © 2001, Juniper Networks, Inc.

After the configuration group is defined, the group can be applied and referenced from within a policy. The syntax for applying a configuration group is by using the **apply-groups** command. Here, the **customer-routes** group from the previous slide is applied to the **allow-good-routes** policy.

Once the group is applied to the routing policy, the policy will look for the configuration group referenced and will search for applicable statements for use in the policy. So the configuration itself inherits the statements that are found in the configuration group. In this case, the three route-filter statements are found. The policy then logically uses those statements in its running operation.

Notice that the candidate configuration simply references the applied group *by name* and never shows the routes that are actually used in the policy. While you can always look at the group itself and decide what should or should not be used, the configuration CLI gives another option. By using the pipe (**|**) feature, we can ask for more information. The command to use after the pipe is **display inheritance**. This will show you what statements from the group have been used in a particular application of the configuration group. This command will also comment the output to explain what configuration group was used for each application. To view the applied statements as if it were a "normal" configuration, use two piped commands together. The commands are **display inheritance** and **except ##**. (The **except** command eliminates lines matching the **##** regular expression from the display.) The complete string to use with the two pipes is shown on the slide.

Subroutines

Where we are going...

- **Policy Subroutines**
- **Policy Subroutine Guidelines**
- **Subroutine Examples**

Copyright © 2001, Juniper Networks, Inc.

Subsequent slides will discuss how the JUNOS software uses a policy subroutines, provide guidelines for its usage, and illustrate the operation using examples.

Policy Subroutines

- One policy can appear as a "subroutine" to another
- Useful when "inner" policy is a set of route-filters

```
[edit policy-options]
policy-statement this-is-the-subroutine { ←
  term what-to-match {
    from {
      match-conditions;
      route-filter destination-prefix match-type <actions>;
      prefix-list name;
    }
    then action;
  }
}
policy-statement main-policy {
  term i-hope-this-works {
    from {
      policy this-is-the-subroutine; ←
    }
    then action;
  }
}
```

Copyright © 2001, Juniper Networks, Inc.

In addition to the use of prefix lists and configuration groups, there is yet another option for defining a common set of routes to be used in a policy. This method relies on the use of a "regular" policy for the definition of these common routes. This policy can then be used as a match criteria from another policy through the use of JUNOS policy *subroutines*. So one policy can appear to be a type of "subroutine" of another policy, although a "subroutine" in the JUNOS software routing policy framework is quite different from the subroutines used in programming languages.

Policy subroutines are most useful when the "inner" or "subroutine" policy is simply a list of route filters and their associated match-types. This is not the only allowable use of a policy subroutine, but route filters are by far the most common and useful application of policy subroutines.

The slide shows (with arrows) both the definition of a subroutine policy (**this-is-the-subroutine**) and the "calling" of this subroutine policy from within the policy **main-policy**. Since the names used for policies and terms are totally up to the person doing the configuration, considerable confusion is possible when policy subroutines are used. Care is always needed, but there are a few guidelines to follow when using policy subroutines.

Policy Subroutine Guidelines

- **Never evaluate a policy within itself! No prefixes will ever match this configuration.**
- **If the subroutine policy matches a prefix:**
 - Actions specified in the *then* that modify attributes are taken immediately
 - Can cause unexpected results (side effects)
- **Subroutine policies return a TRUE or FALSE value**
 - TRUE is returned if there was a match on the route and an accept action was specified (perform main *then* actions)
 - FALSE is returned if there was a match on the route and a reject action was specified (no *then* actions, continue...)
- **The routing protocol's default policy is ALWAYS evaluated as part of the policy subroutine!**
- **Careful planning is needed when using subroutines**

Copyright © 2001, Juniper Networks, Inc.

Policy subroutines are **not** at all like programming language subroutines! Policy subroutines cannot be nested "recursively" to force evaluation of a policy within itself, although there is nothing to prevent this type of configuration. However, no prefixes will ever match this type of "policy subroutine within itself" structure.

The policy subroutine evaluation is very similar to the evaluation of policies in general. The subroutine is checked for match criteria and if a match is found, the actions specified in the *then* section of the term or policy that modify attributes of the route are taken immediately. This immediate action can cause some unexpected and unintended side effects, especially when terminating actions such as **accept** or **reject** are considered.

The main difference between a "normal" policy and a policy subroutine is what the terminating actions mean to the subroutine. The policy subroutine actions do not actually accept or reject a route on their own. They only assist the policy that used the subroutine in determining if a match is found. The policy subroutine does this by changing the actions of accept and reject into the logic answers of TRUE and FALSE.

If the subroutine has an action of **accept**, then it returns a value of TRUE to the "main" policy that referenced the policy subroutine. When the main policy sees a value of TRUE, it sees the policy subroutine as a match and takes any specified actions.

If the subroutine has an action of **reject**, then it returns a value of FALSE to the main policy. When the main policy sees a value of FALSE, it does not see the policy subroutine as a match and skips any specified actions and continues processing its own terms. So a **reject** in a subroutine policy might *still result in an accepted route*!

The policy subroutine is never viewed "alone." The default policy for the particular protocol using the policy is **ALWAYS** evaluated. That means if the main policy is applied to BGP, the BGP default policy will be evaluated *as part of the subroutine*.

All in all, extreme care is needed when configuring and using policy subroutines.

Subroutine Example

What happens when the BGP route of 10.10.10.0/24 is evaluated by the policy subroutine?

```
[edit policy-options]
policy-statement subroutine-policy {
  term accept-exact {
    from route-filter 192.168/16 exact;
    then accept;
  }
}
policy-statement main-policy {
  term only-want-exact {
    from {
      policy subroutine-policy;
    }
    then accept;
  }
}

[edit protocols]
bgp {
  export main-policy;
}
```

Copyright © 2001, Juniper Networks, Inc.

The slide shows a policy called **main-policy** applied to BGP as an export policy. A candidate BGP route of 10.10.10.0/24 is being evaluated by this policy. The match criteria for **main-policy** is the subroutine policy of **subroutine-policy**.

What happens? First, the main policy "calls" the subroutine policy and so the match criteria of **subroutine-policy** are evaluated. The BGP route 10.10.10.0/24 does not match the defined subroutine, so *the BGP default policy is evaluated* as part of the subroutine. The BGP default policy will match the 10.10.10.0/24 route in this instance and has an action of **accept**. The subroutine logic then translates that **accept** into a TRUE value and returns this TRUE to the **main-policy**.

The **main-policy** now has a match for the term **accept-exact** (because of the TRUE returned by the policy sub-routine). So **main-policy** takes the specified action for this BGP route, which is **accept**. Therefore, this BGP route is accepted for advertising to BGP peers.

Note that the TRUE for the BGP route 10.10.10.0/24 was generated **not** by the policy subroutine itself, but by the BGP default policy that "follows" the policy subroutine evaluation (the BGP default policy follows because the BGP route does not match the policy subroutine match condition of **192.168/16 exact**). And this BGP default policy, as usual, never appears in the configuration.

Another Subroutine Example

What happens when the *static* route of 10.10.10.0/24 is evaluated by the policy subroutine?

```
[edit policy-options]
policy-statement subroutine-policy {
  term accept-exact {
    from route-filter 192.168/16 exact;
    then accept;
  }
}
policy-statement main-policy {
  term only-want-exact {
    from {
      policy subroutine-policy;
    }
    then accept;
  }
}

[edit protocols]
bgp {
  export main-policy;
}
```

Copyright © 2001, Juniper Networks, Inc.

Policy subroutines can be very tricky to "see" properly, so another example is called for. The slide shows a policy called **main-policy** applied to BGP as an export policy, exactly the same as in the first example. But this time a candidate *static* route (not BGP route, as before) of 10.10.10.0/24 is being evaluated by this policy. The match criteria for **main-policy** is the subroutine policy of **subroutine-policy**.

What happens? First, the main policy "calls" the subroutine policy and so the match criteria of **subroutine-policy** are evaluated. The static route 10.10.10.0/24 does not match the defined subroutine, so *the BGP default policy is evaluated* as part of the subroutine. The BGP default policy does *not* match the 10.10.10.0/24 route in this instance and so has an action of **reject**. The subroutine logic then translates that **reject** into a FALSE value and returns this FALSE to the **main-policy**.

The **main-policy** now does *not* have a match for the term **accept-exact** (because of the FALSE returned by the policy sub-routine). Since there are no more terms in **main-policy** and the BGP configuration has no more policies applied, the default BGP policy is evaluated. The BGP default policy does not match the static route (BGP will not export static routes by default) and therefore has an action of **reject**. So this static route is rejected for advertising to BGP peers.

Note that the FALSE for the static route 10.10.10.0/24 was generated by the policy subroutine itself.

More Subroutine Examples

Consider route 10.10/16 in all examples below

Example #1:

```
policy-statement outer1 {
  from policy called1;
  then accept;
}
policy-statement called1 {
  from {
    route-filter 10.10.0.0/16 exact;
  }
  then accept;
}
```

Example #2:

```
policy-statement outer2 {
  from policy called1;
  then reject;
}
policy-statement called1 {
  from {
    route-filter 10.10.0.0/16 exact;
  }
  then accept;
}
```

Example #3:

```
policy-statement outer3 {
  from policy called2;
  then accept;
}
policy-statement called2 {
  from {
    route-filter 10.10.0.0/16 exact;
  }
  then reject;
}
```

Example #4:

```
policy-statement outer4 {
  from policy called3;
  then {
    community add comm2;
    accept;
  }
}
policy-statement called3 {
  from {
    route-filter 10.10.0.0/16 exact;
  }
  then
    community add comm1;
    accept;
}
```

Copyright © 2001, Juniper Networks, Inc.

The route 10.10.0.0/16 is being evaluated against four different subroutine examples shown on the slide. What happens in each case?

Example # 1: The route matches the subroutine and the action is **accept**. This returns a TRUE to the outer policy which is seen as a match. The outer policy then has an action of **accept**, so the route is **accepted**.

Example # 2: The route matches the subroutine and the action is **accept**. This returns a TRUE to the outer policy which is seen as a match. The outer policy then has an action of **reject**, so the route is **rejected**.

Example # 3: The route matches the subroutine and the action is **reject**. This returns a FALSE to the outer policy which is seen as "no match." The outer policy then skips the actions and the route is passed to a default policy (not shown here) for further processing. **More information is needed** about the routing protocol in this case.

Example # 4: The route matches the subroutine and the action is to add a community value of **comm1** and to **accept**. This adds the appropriate community value and also returns a TRUE to the outer policy which is seen as a match. The outer policy then has an action of adding a community value of **comm2** and an **accept** action, so the second community is added to the route and the route is **accepted with both community values added**.

Policy Expressions

Where we are going...

- **Controlling Policy Order**
- **Policy Expression Operators**
- **Policy Expression Logic**
- **OR/AND Operator Logic and Example**
- **NOT/Group Operator Logic and Example**
- **Expression Logic Pitfalls**

Copyright © 2001, Juniper Networks, Inc.

Subsequent slides will discuss how the JUNOS software uses a policy expression, provide some guidelines for its usage, and illustrate the operation through some examples.

Controlling Policy Order

- By default, policies are evaluated in order, from left to right, as follows:
 - The terms of each policy are evaluated in order, from first to last
 - The first matching policy term is applied to the route
 - If the policy action contains an **accept** or **reject** control action, no more policy terms or policies are evaluated
 - If there is no **accept** or **reject** action and if there are more terms in the policy, the next term is evaluated
 - If there are no more terms in the current policy, the next policy listed in the **import** or **export** statement is evaluated
- Usually, the policy operation order is sufficient
- To change the default order, use policy expressions

Copyright © 2001, Juniper Networks, Inc.

By default, the JUNOS software routing policy framework evaluates multiple policies applied to a protocol in a left-to-right fashion in the order they were configured. In each policy, the terms (if any) in the policy are evaluated in order, from first to last. The first matching policy condition in a term is then applied to the route. If this policy action contains an **accept** or **reject**, no more policy terms or policies are evaluated (there is an exception for policy subroutines). If there are no terminating actions, and there are more terms in the policy, then the next term is evaluated. If there are no more terms in the current policy, then the next policy listed in the **import** or **export** statement is evaluated until a terminating action is reached or there are no more policies.

In most cases, this default policy order is fine. Policies are seldom intertwined enough that close attention to specific order is necessary. But there are times when policy order is important.

So for flexibility purposes, this default evaluation order can be altered through the use of *policy expressions*. A policy expression is a logical evaluation of multiple policies using Boolean AND/OR operators. The configuration views a policy expression as a single large policy whose *logical expression result* (TRUE or FALSE) must be evaluated before actually accepting or rejecting a route.

Policy Expression Operators

Used when specifying multiple policy names

Highest to lowest precedence	Description
!	Logical NOT. Changes a <i>TRUE</i> result to <i>FALSE</i> and a <i>FALSE</i> result to <i>TRUE</i>
&&	Logical AND. If the first policy returns <i>TRUE</i> , the next policy is evaluated. If the first policy returns <i>FALSE</i> , the next policy is skipped.
	Logical OR. If the first policy returns <i>TRUE</i> , the next policy is skipped. If the first policy returns <i>FALSE</i> , the next policy is evaluated.
(...)	Group operator. Used to override default precedence order.

Copyright © 2001, Juniper Networks, Inc.

The slide shows the defined policy expression operators that can be used when specifying the application of multiple routing policies. These are listed from highest to lowest precedence.

When configuring policy expressions, the logical operators of NOT, AND, and OR are translated into the symbols !, &&, and || respectively. In addition, the operators and the policies they act upon can be grouped together through the use of the parenthesis (...).

The logical NOT (!) applies to a single policy and changes a *TRUE* to a *FALSE* and a *FALSE* to a *TRUE* in terms of the value returned.

The logical AND (&&) is used between policies. If the first policy evaluated returns a *TRUE*, then the next policy is evaluated. If the first policy returns a *FALSE*, the next policy is skipped.

The logical OR (||) is also used between policies. If the first policy returns a *TRUE*, then the next policy is skipped. If the first policy returns a *FALSE*, then the next policy is evaluated.

When the group operator (parentheses) is used, the innermost set of policies is evaluated, then the next set, and so on until the outermost set of policies is reached.

Policy Expression Logic

- There are 2 separate steps to consider – evaluation and action
- Evaluation is the True/False logic portion
 - A policy action of "accept" returns a TRUE
 - A policy action of "next policy" returns a TRUE
 - A policy action of "reject" returns a FALSE
- The logic evaluation has nothing to do with the actual accept/reject decision on a route, it only serves to determine the TRUE/FALSE nature of the expression
- The action is taken from the policy that caused the expression result to either be TRUE or FALSE
- The actual "action" portion of the policy is only applied after the logic evaluation
- Unlike policy subroutines, no default is applied

Copyright © 2001, Juniper Networks, Inc.

There are two very important steps to consider when using policy expressions. The first is the logical *evaluation* of the expression and its TRUE/FALSE result. The second is the translation of that result into an *action*. These are always separate steps and should always be viewed as such.

Much like the policy subroutines, the policy expressions evaluate a configured policy for match criteria and translate appropriate actions into TRUE/FALSE logical results. Within the policy expression logic, an action of **accept** translates into a TRUE value. An action of **next policy** translates into a TRUE value. An action of **reject** translates into a FALSE value.

But simple logic evaluation has nothing to do with the ultimate acceptance or rejection of a route. Using the policy expression operators, the policies are evaluated and the actions turned into TRUE/FALSE until a guaranteed result is achieved (that is, this sequence of policies **must** be TRUE or **must** be FALSE). Then that policy that caused the policy expression to be TRUE or FALSE has its actions applied to the route.

For example, if the policy that caused the expression as a whole to be TRUE had an action of **accept**, then the route would be accepted. If the policy that caused the expression to be FALSE had an action of **reject**, then the route would be rejected. Lastly, if the policy that caused the expression to be TRUE had an action of **next policy**, then the next policy in the chain (which could be the default policy) would be evaluated.

It is only after that logic result is achieved that the "real" action on the route can be taken. First comes logic evaluation, then the "action" is applied.

The big difference between the policy subroutines and policy expressions is that policy expressions do **NOT** have any concept of the default policy. They are solely evaluated on their own terms.

OR/AND Operator Logic

- **(Policy1 || Policy2)**
 - If Policy1 returns a TRUE, skip Policy2 and take action specified in Policy1
 - If Policy1 returns a FALSE, evaluate Policy2 for TRUE/FALSE and take action specified in Policy2
- **(Policy1 && Policy2)**
 - If Policy1 returns a TRUE, evaluate Policy2 for TRUE/FALSE and take action specified in Policy2
 - If Policy1 returns a FALSE, skip Policy2 and take action specified in Policy1

Copyright © 2001, Juniper Networks, Inc.

Policy expression evaluation always continues until the outcome is guaranteed. Depending on the operator, this might mean that a single policy is evaluated, two policies are evaluated, or even more, depending on the complexity of the expression.

Consider the first example on the slide using the OR. Following the usual rules of logic, the logical OR requires that only one of the arguments (policies) be TRUE for the expression to be TRUE. So if **policy1** returns a TRUE result through an **accept** or a **next policy**, then **policy2** does not have to be evaluated (since the expression result is guaranteed). The action specified in **policy1** (**accept** or **next policy**) is taken.

If **policy1** returns a FALSE result through a **reject**, then **policy2** does have to be evaluated. If **policy2** then returns a TRUE result via an **accept** or a **next policy**, then the action specified in **policy2** is taken.

Now consider the second example using the AND. Following the usual rules of logic, the logical AND requires that *both* of the arguments (policies) be TRUE for the whole expression to be TRUE. So if **policy1** returns a TRUE result through an **accept** or a **next policy**, then **policy2** has to be evaluated (since the expression result is not guaranteed at that point). If **policy2** then returns a TRUE result through an **accept** or a **next policy**, then that action is taken. If **policy2** returns a FALSE result through a **reject** then that action is taken.

What if **policy1** returns a FALSE? The logical AND requires that both of the arguments (policies) be TRUE for the expression to be TRUE. If **policy1** returns a FALSE result through a **reject**, then **policy2** does not have to be evaluated (since the expression is guaranteed at that point). The action of **reject** from **policy1** is taken.

OR/AND Operator Example

- What is the result for (PolicyA || PolicyB)?
 - 172.24.68/24
 - 172.20.145/24
 - 172.20.100/24
- What is the result for (PolicyA && PolicyB)?
 - 172.24.68/24
 - 172.20.145/24
 - 172.20.100/24

```

policy-statement PolicyA {
  term routes-from-customer-A {
    from {
      route-filter 172.24.67/24 exact;
      route-filter 172.24.68/24 exact;
      route-filter 172.24.69/24 exact;
    }
    then accept;
  }
  term nothing-else
  then reject;
}

policy-statement PolicyB {
  term routes-from-customer-B {
    from {
      route-filter 172.20.145/24 exact;
      route-filter 172.20.146/24 exact;
      route-filter 172.20.147/24 exact;
    }
    then accept;
  }
}

```

Copyright © 2001, Juniper Networks, Inc.

The slide shows two configured policies, PolicyA and PolicyB. These policies are applied to a routing protocol (not shown). What happens when each of the routes listed is applied to the logical OR (||) and AND (&&) expressions shown combining the two policies?

PolicyA || PolicyB

- 172.24.68/24 - Matches the first term in **PolicyA** and the action is **accept**. This means that **PolicyA** is TRUE and the entire expression must be TRUE. The action from **PolicyA** is taken and the route is **accepted**.
- 172.20.145/24 - Matches the second term in **PolicyA** and the action is **reject**. This means that **PolicyA** is FALSE, so **PolicyB** is evaluated (since the logic result for the whole expression is still in doubt). The route matches the first term in **PolicyB** and the action is **accept**. The action from **PolicyB** is taken and the route is **accepted**.
- 172.20.100/24 - Matches the second term in **PolicyA** and the action is **reject**. This means that **PolicyA** is FALSE, so **PolicyB** is evaluated (result in doubt). The route matches no terms in **PolicyB** so the default action is **next policy**. The action from **PolicyB** is taken (**next policy**) because **PolicyB** "determined the expression result." So the **next policy** in the chain (which could be the default policy) is evaluated.

Advanced Policy

PolicyA && PolicyB

- 172.24.68/24 - Matches the first term in **PolicyA** and the action is **accept**. This means that **PolicyA** is TRUE, so **PolicyB** must be evaluated (since the logic result for the whole expression is still in doubt). The route matches no terms in **PolicyB** so the default action is **next policy** (TRUE). The action from **PolicyB** is taken because **PolicyB** "determined the expression result." So the **next policy** in the chain (which could be the default policy) is evaluated.
- 172.20.145/24 - Matches the second term in **PolicyA** and the action is **reject**. This means that **PolicyA** is FALSE and the entire expression is then FALSE (result is now determined). The action from **PolicyA** is taken because **PolicyA** "determined the expression result." So the route is **rejected**.
- 172.20.100/24 - Matches the second term in **PolicyA** and the action is **reject**. This means that **PolicyA** is FALSE and the entire expression is then FALSE (result is now determined). The action from **PolicyA** is taken because **PolicyA** "determined the expression result." So the route is **rejected**.

NOT/Group Operator Logic

- **!Policy1**
 - If **Policy1** returns a **TRUE**, reverse that to **FALSE** and reject the route
 - If **Policy1** returns a **FALSE**, reverse that to **TRUE** and accept the route
- **Group (...): export [(Policy1 && Policy2) Policy3]**
 - If **Policy1** returns a **FALSE**, skip all other policies and take action specified in **Policy1** ("reject")
 - If **Policy1** returns a **TRUE**, evaluate **Policy2** and take action specified.
 - If **Policy2** action is "accept", accept the route and skip all other policies
 - If **Policy2** action is "reject", reject the route and skip all other policies
 - If **Policy2** action is "next policy", evaluate **Policy3**

Copyright © 2001, Juniper Networks, Inc.

The logical NOT operation reverses the logical result prior to converting that result into an actual action. If a policy evaluation result is **TRUE** either through an **accept** or a **next policy**, then that **TRUE** is reversed to a **FALSE**. The **FALSE** then is translated into an action of **reject**. If a policy evaluation result is **FALSE** through a **reject**, then that **FALSE** is reversed to a **TRUE**. The **TRUE** then is translated into an action of **accept**.

When using multiple logical operators, it is possible to group those operators to be evaluated in a specific order. This is accomplished with the use of the parentheses. In the example on the slide, **Policy1** and **Policy2** are combined into a policy expression. This expression is then added to a policy chain with **Policy3** in the **export** statement **(Policy1 && Policy2) Policy3**. The router interprets this as two logical policies to evaluate in the default left-to-right fashion.

The policy expression of **(Policy1 && Policy 2)** is evaluated first according the rules previously discussed. If **Policy1** returns a **FALSE**, the route is rejected and all policy processing stops.

If **Policy1** returns a **TRUE**, then **Policy2** is evaluated. If **Policy2** returns **FALSE**, then the route is rejected and all policy processing stops. If **Policy2** returns **TRUE** through an **accept** action, then the route is accepted and all policy processing stops. If **Policy2** returns **TRUE** through a **next policy** action, then the policy route is passed to **Policy3** for further processing in accordance with the default policy evaluation rules.

NOT Logic Example

- If a route is from Customer A it is sent via BGP
- If a route is from Customer B it is suppressed by BGP

```
[edit policy-options]
policy-statement CustomerA {
  term routes-from-A {
    from {
      route-filter 172.24.67/24 exact;
      route-filter 172.24.68/24 exact;
      route-filter 172.24.69/24 exact;
    }
    then reject;
  }
}
policy-statement CustomerB {
  term routes-from-B {
    from {
      route-filter 172.20.145/24 exact;
      route-filter 172.20.146/24 exact;
      route-filter 172.20.147/24 exact;
    }
    then accept;
  }
}
[edit protocols]
protocols bgp {
  export [ !CustomerA !CustomerB ];
}
```

Copyright © 2001, Juniper Networks, Inc.

In this example of the NOT logic for policy expressions, two policies are defined. The **CustomerA** policy lists some routes with an action of **reject**. The **CustomerB** policy lists some routes with an action of **accept**. The policies are combined as a BGP **export** policy as **[!CustomerA !CustomerB]**.

When one of the routes specified in **CustomerA** is evaluated by that policy, the configured action of **reject** causes a logical result of FALSE. Since the export statement has a NOT expression defined for **CustomerA**, that FALSE is reversed to a TRUE and the route is accepted for advertisement to BGP peers.

When one of the routes specified in **CustomerB** is evaluated by that policy, the configured action of **accept** causes a logical result of TRUE. Since the export statement has a NOT expression defined for **CustomerB**, that TRUE is reversed to a FALSE and the route is rejected for advertisement to BGP peers.

As written on the slide, the **export [!CustomerA !CustomerB]** will not allow any routes other than those in **CustomerA** to be sent to BGP peers. For all routes that do not match **CustomerA**, the action from **CustomerB** will always return a TRUE. The TRUE is then reversed to a FALSE and all routes are rejected.

To allow **CustomerA** routes to be sent, reject **CustomerB** routes, and allow all BGP transit traffic to be evaluated by the BGP default policy, the export statement should be rewritten as **export [(!CustomerA || !CustomerB)]**.

Group Logic Example

If a route is from Customer A or B and route has a BGP community of send-to-Internet, then export route in BGP

```
[edit policy-options]
policy-statement customer-A {
  term routes-from-A {
    from {
      route-filter 172.24.67/24 exact;
      route-filter 172.24.68/24 exact;
      route-filter 172.24.69/24 exact;
    }
    then accept;
  }
  term nothing-else
  then reject;
}
policy-statement customer-B {
  term routes-from-B {
    from {
      route-filter 172.20.145/24 exact;
      route-filter 172.20.146/24 exact;
      route-filter 172.20.147/24 exact;
    }
    then accept;
  }
  term nothing-else
  then reject;
}

[edit policy-options]
policy-statement comm-values {
  term check-community {
    from community send-to-Internet;
    then accept;
  }
  term nothing-else
  then reject;
}

[edit protocols]
protocols bgp {
  export [(customer-A||customer-B)&&comm-values];
}
```

Copyright © 2001, Juniper Networks, Inc.

There is still policy expression grouping logic to explore. The slide shows a policy expression that groups the policies **CustomerA** and **CustomerB** together in an OR statement. The result of this logical OR is then combined with the policy **comm-values** in a logical AND statement.

Since the OR expression is enclosed in parentheses, this inner expression must be evaluated first. Since this is a logical OR, this inner group will return a TRUE value when a route matches **either** of the two policies. That TRUE in turn causes an evaluation of the **comm-values** policy since the outcome of the logical AND policy expression is not determined. If the route has a community value of **send-to-Internet** defined, this route will get a TRUE value from the **comm-values** policy and will be accepted. If the community attribute is not defined on the route, the route will receive a FALSE and will be rejected.

If the OR expression ever returns a FALSE (due to a route not matching either of the policies), then the **comm-values** policy will not be evaluated and the route will be rejected.

Expression Logic Pitfalls

- **[Policy1 Policy2 Policy3] is not the same as (Policy1 && Policy2 && Policy3) or (Policy1 || Policy2 || Policy3)**
 - **[Policy1 Policy2 Policy3]:** if Policy1 has a match, and the action is *accept* or *reject*, no more policies are evaluated
 - If Policy2 matches and has *accept* or *reject*, Policy3 is not evaluated
 - **(Policy1 && Policy2 && Policy3):** if Policy1 has a TRUE, Policy2 is still evaluated!
 - **(Policy1 || Policy2 || Policy3):** if Policy1 has a TRUE, take the action specified. Policy 2 and Policy3 are not evaluated
- **Watch out for "lazy logic": (Policy1 || Policy2)**
 - Does not mean "apply Policy1 OR Policy2"
 - If Policy1 *ever* produces a TRUE, Policy2 is not evaluated!

Copyright © 2001, Juniper Networks, Inc.

The evaluation procedures for the default, left-to-right policy evaluation in a simple policy chain are different than when the policy expressions OR and AND are used. Each needs to be evaluated separately in context.

For example, assume that a route matches **Policy1** and has an action of **accept**. The simple policy chain without logic expressions will accept the route and stop processing. The OR expression will also accept the route and stop processing. The AND expression, however, will need to process **Policy2** and possibly **Policy3**.

If a route matches **Policy1** and has an action of **next policy**, then the simple policy chain without logic expressions will pass the route to **Policy2** for more processing. The OR expression, however, will pass the route to the default policy for more processing. The AND expression, as before, will need to process **Policy2** and possibly **Policy3** before taking action.

If a route matches **Policy1** and has an action of **reject**, then the simple policy chain without logic expressions will reject the route and stop processing. The OR expression, however, will need to process **Policy 2** and possibly **Policy3** before taking action. The AND expression will reject the route and stop processing.

So a logic expression like **(Policy1 || Policy2)** never means "apply **Policy1** OR **Policy2**)." There is much more to be considered. If **Policy1** in this expression ever produces a TRUE, then **Policy2** is never evaluated at all.

The actions taken by complex policy logic expressions, policy subroutines, and combinations of the two are never easy to figure out. Careful planning and execution and testing is necessary before complex policy strings can be used with confidence.

Review Questions

- **Why are multiple routing policies often configured and used?**
- **How do prefix lists differ from route filters?**
- **Do policy configuration groups allow the use of route filters and match-types?**
- **What is the role of the configuration group wildcard <*>?**
- **What is the logic value of a policy that applies the next policy action?**
- **Is the policy expression !(Policy1 && Policy2) allowed?**
- **Is the policy expression !(Policy1) the same as !Policy1?**

Copyright © 2001, Juniper Networks, Inc.

During this module we discussed:

- The way in which the JUNOS software processed policies when there was more than one policy to apply.
- How to configure a prefix list that could be used to ease the maintenance of customer routes.
- How configuration groups could be used to make the application of prefix lists simpler and easier.
- The behavior of routing policy subroutines, which could be used in some instances to make the maintenance of complex policies easier to perform.
- The use of policy expressions such as a logical AND or OR to alter the default flow of policy application.



Juniper Networks Advanced Policy

Release 5.0, Revision 1

Module 5: Border Gateway Protocol

Copyright © 2001, Juniper Networks, Inc.

Objectives

- Describe the configuration options for BGP peers
- Explain the default "movement" of BGP routes through a router
- Describe the role of BGP attributes in the process of choosing routes to use and advertise
- Describe how BGP Next-hop reachability is achieved
- Explain options for handling IBGP scalability issues

Copyright © 2001, Juniper Networks, Inc.

This module discusses:

- The JUNOS software configuration options for BGP.
- The default behavior of the BGP routing protocol with regard to the "movement" (use and advertising) of BGP routes through the router.
- How the BGP attributes play a role in the process of choosing routes to use and advertise through BGP.
- When BGP next-hop reachability is an issue and how BGP routes find their next-hops.
- The available options for handling IBGP scalability issues such as the number of peer sessions that must be maintained.

BGP Operation

Where we are going...

- **EBGP Peering**
- **IBGP Peering**
- **BGP Explicit Configuration**
- **BGP Configuration Options**
- **BGP Policy**
- **BGP Import Policy & Example**
- **BGP Export Policy & Example**
- **BGP Path Selection Flow**
- **Path Selection Notes**

Copyright © 2001, Juniper Networks, Inc.

Subsequent slides will review the basic operation of BGP including EBGP connections, IBGP connections, route movement through BGP memory tables, and BGP path selection.

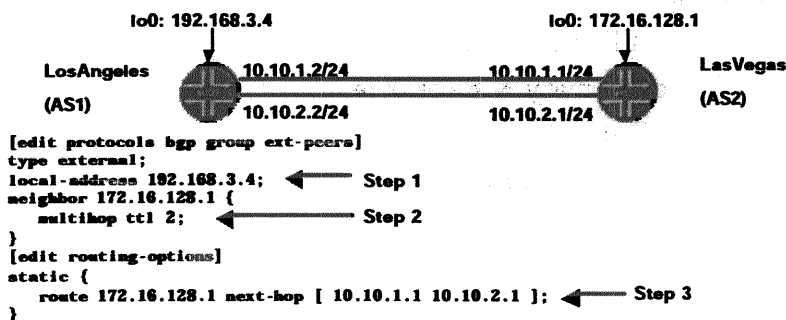
EBGP Peering

EBGP sessions peer with physical interface addresses



If the point to point link to the AS is down, reachability is lost

EBGP sessions may peer with non-physical addresses



Copyright © 2001, Juniper Networks, Inc.

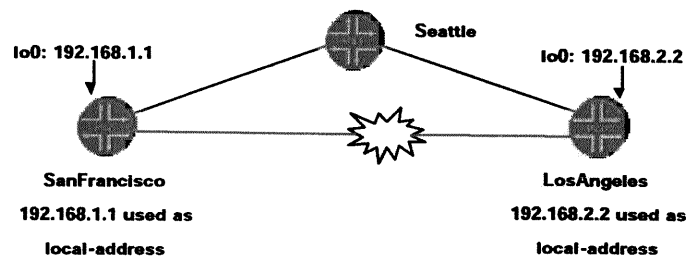
The default for an External Border Gateway Protocol (EBGP) connection is to peer over a single physical hop using the physical interface address of the peer. This is because typically the connection between EBGp peers is a point-to-point WAN link that connects two different Autonomous Systems (AS). If the point-to-point link is unavailable for traffic, reachability is lost to the remote router anyway, and the EBGp session should be lost to prevent conflict between the EBGp view of the network and the physical network itself. Use of the physical interface address for EBGp peers assures that the EBGp session fails when the WAN link fails. Furthermore, the default Time-To-Live (TTL) value in the IP header for an EBGp packet is 1. Any attempt to configure an EBGp session over multiple hops (using default settings) will fail.

There are some cases where it is advantageous to alter this default, one-hop, physical peering EBGp behavior. One such case is when there are multiple physical links connecting two routers who are to be EBGp peers. In this case, if one of the point-to-point links fails, there is still reachability on the alternate link. There are three extra configuration steps that need to be taken to do this, as shown on the slide.

First, each router needs to establish the peering session with the loopback address of the remote router. This is accomplished through use of the **local-address** command which alters the peer address header information in the BGP packets. Second, the **multihop** command is used to alter the default TTL value in the BGP packets. Administrators can specify what TTL value to use in place of the default value of 1. On the slide, a TTL value of 2 is used to ensure that the session can not be established across any other "backdoor" links in the network. Third, each router needs to have IP routing capability to the remote router's loopback address. As the slide shows, this is often accomplished by using a static route to "map" the loopback address to the interface physical addresses.

IBGP Peering

IBGP sessions should peer with loopback addresses for resiliency



Copyright © 2001, Juniper Networks, Inc.

For Internal BGP (IBGP) sessions, the philosophy is different than with EBGp. Instead of typically having a single point-to-point link between Autonomous Systems (ASs) to deal with as in EBGp, IBGP runs within an AS and usually can use multiple links and paths between the routers forming the endpoint of the IBGP sessions. So the best thing to do here is to allow the IBGP sessions to try to stay up even if the physical interface that the IBGP session is running on has failed. Only the BGP **local-address** need be configured to retain these IBGP sessions, in contrast to EBGp.

No change to the TTL is needed, since for IBGP sessions between peers the default TTL value is set to 64. This allows these sessions to be established over multiple links and paths within the AS and also allows the IBGP sessions to remain established in the event of a network failure within the AS. You can still route around the problem if there is some way to reach the peer IBGP router. So static routes are not needed or wanted either.

However, the default BGP header information, and the default peering arrangement for IBGP is to still use physical IP addresses for connectivity. While this is okay when a network failure occurs in the middle of the AS somewhere, it does cause connectivity problems when the failure involves one of those router interfaces of the IBGP peers themselves. Quite simply, if the interface is unavailable, the BGP session will not establish itself. This issue is resolved through the use of the **local-address** command.

All IBGP routers should establish peering sessions to the loopback addresses of all other IBGP routers instead of the default (physical interface) address. Since the loopback address is a virtual interface inside of the router, it will only be unavailable when the router itself is unavailable. Both ends of the BGP session need to know that the defaults are being altered so each IBGP router also tells its neighbors that its **local-address** is its own loopback address. This is considered a best practice within many Internet Service Providers (ISPs).

BGP Explicit Configuration

- BGP requires explicit configuration between neighbors
- Details about the peer's IP address and AS are required

```
[edit protocols bgp]
group ext-peers {
  type external;
  peer-as 2;
  neighbor 10.10.10.1;
}
group int-peers {
  type internal;
  local-address 172.16.1.1;
  neighbor 172.16.2.2;
}

[edit routing-options]
autonomous-system 1;
```

Copyright © 2001, Juniper Networks, Inc.

BGP sessions will only establish between BGP peers when both peers know the IP address and Autonomous System (AS) number of each other. This information is explicitly configured on each peer. Should any of this information not match, the BGP session will be unusable.

The slide shows a BGP router that has a single IBGP session and a single EBGP session configured. The IBGP session is configured within the peer group of **int-peers**. The **type** command tells the local router that the remote router is in the same AS as itself (AS 1 in this case). The session is established to neighbor 172.16.2.2 and the **local-address** command is used to alter the header information for the session, which is established from 172.16.1.1 instead of from the physical interface address.

The EBGP session is configured within the peer group of **ext-peers**. The **type** command in this case tells the local router that the far end router is in a different AS. The remote AS number is supplied via the **peer-as** command.

BGP Configuration Options

- **passive** keeps BGP from sending OPEN message

```
[edit protocols bgp]
group ext-peers {
  type external;
  peer-as 2;
  neighbor 10.10.10.1 {
    passive;
  }
}
```

- **allow** accepts OPEN messages from any peer within the configured IP address range

```
[edit protocols bgp]
group ext-peers {
  type external;
  allow 10.10/16;
}
```

- **MD5 authentication** can be enabled

```
[edit protocols bgp]
group ext-peers {
  type external;
  peer-as 2;
  neighbor 10.10.10.1 {
    authentication-key "$9$.mQn/9pR2SAp7VY0j1Ap001h";
  }
}
```

Copyright © 2001, Juniper Networks, Inc.

Many default parameters for an explicit BGP configuration can be altered by the network administrator as desired. Some examples are the use of the **passive** command, the **allow** command, and enabling Message Digest 5 (MD5) authentication between BGP peers.

When using the default BGP configuration parameters, the local router will initiate a BGP OPEN message to the remote router to establish the session. The **passive** command stops this default action and no OPEN message will be sent. The IP address of the remote peer is still configured, but the remote router must initiate the BGP session.

The related option of **allow** will also stop the sending of a BGP OPEN message to the remote router. In addition, the **allow** command also relaxes the requirement of explicitly configuring the remote router's IP address by allowing the network administrator to define a subnet range for connections. Any BGP OPEN message received from an IP address within the configured range will be processed and a session will be initiated with that remote router.

Another BGP peer configuration option accomplishes a different goal. Instead of making it *easier* to establish a session (less explicit configuration with **allow**), this option makes it *harder* by requiring peer authentication. When authentication is configured for BGP, the default authentication option is the MD5 hash algorithm. Hash authentication is similar to using a cyclical redundancy check (CRC) on frames to see if bits have been altered in transit (through errors in the CRC case). Since MD5 is the default, the administrator only needs to configure the key (password) using the **authentication-key** command. Within the configuration, the password is automatically encrypted (actually, it is hashed) for security. This authentication password is sent in every BGP message from the router. Only remote peers who have the same password configured will be able to authenticate the messages and use the BGP message content.

BGP Policy

- **Import and export policy determines BGP behavior**
- **BGP attributes are available to use/adjust**
- **Attributes can be changed by import/export policy**
- **BGP stores routes in three main Routing Information Base (RIB) memory tables**
 - **RIB-IN: All received routes get placed here**
 - **RIB-LOCAL: Routes the local router is using to forward traffic**
 - **RIB-OUT: All advertised routes get placed here**
- **Only active BGP routes in the local routing table may be advertised to peers**
 - **Single best BGP path is advertised**
 - **advertise-inactive can be used when BGP route is not active, but only the single best inactive BGP path is advertised**

Copyright © 2001, Juniper Networks, Inc.

BGP is a very policy oriented protocol, in the sense that import and export policies largely determine BGP behavior.

All of the BGP attributes are available to the router for use and can be adjusted to influence the behavior of the local router as well as other routers receiving the route.

Each of the BGP attributes can be used as a match criteria for a policy and each can be modified via a policy action. To better understand where BGP import and export policies are applied to BGP routes, the process of how a router uses BGP routes should be detailed.

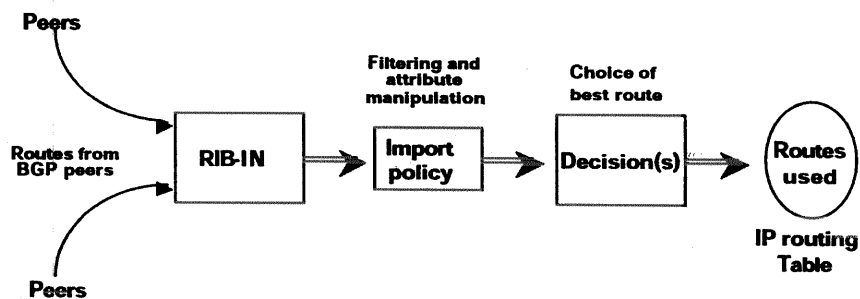
BGP uses three different storage tables known as Routing Information Bases (RIB) as "databases" to maintain routing knowledge. There is a separate RIB-IN for each established BGP peer to store all routes received from that peer. There is a RIB-LOCAL where BGP stores routes used for traffic forwarding. Then there is a separate RIB-OUT for each established BGP peer to store routes to be advertised to that peer.

Only active BGP routes in the routing table can be moved into the RIB-OUT tables and are advertised to BGP peers. In addition, only the single best BGP path to each separate IP route destination is placed in the RIB-LOCAL and RIB-OUT tables.

At times, it is possible that the best BGP path is not advertised to a peer due to the local router's routing table rules. For example, if the router knows about a particular route through both IS-IS and BGP, the IS-IS route will be active in the local routing table. This is due to the default JUNOS software protocol preference values. Therefore, the BGP "version" of that route will not be sent to any peers, since BGP advertises only active routes (routes used by BGP). To override this default action, network administrators can use the **advertise-inactive** command. This command will always force the advertisement of the single best BGP path to any destination regardless of whether the route is currently active in the local routing table or not.

BGP Import Policy

Import policies are enforced between the RIB-IN & RIB-LOCAL tables



Copyright © 2001, Juniper Networks, Inc.

BGP stores the information about routes received from BGP peers in the RIB-IN table. No policies are applied yet: all information is stored in this table.

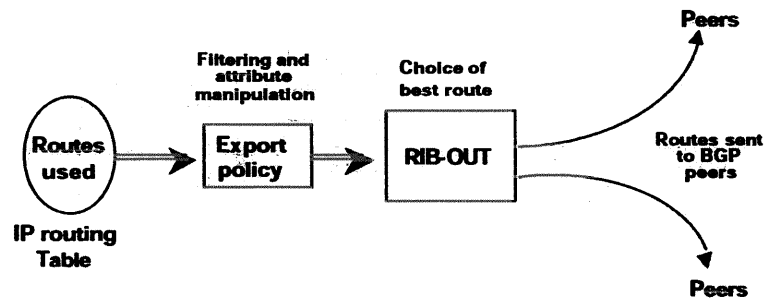
As BGP moves the routes that it received from peers to the RIB-LOCAL table, the JUNOS software routing policy framework can apply import policies. These policies can reject (filter) routes or can change attributes and affect what the BGP route selection process uses to pick the best route.

After the BGP import policy or policies (if any are configured and applied) has filtered and manipulated the BGP attributes, the BGP decision process chooses the "best" route to use and installs that route into the IP routing table.

It should be noted that even if no routing policies have been configured, the default (and unseen) BGP import policy will always be applied.

BGP Export Policy

Export policies are enforced between the RIB-LOCAL & RIB-OUT tables

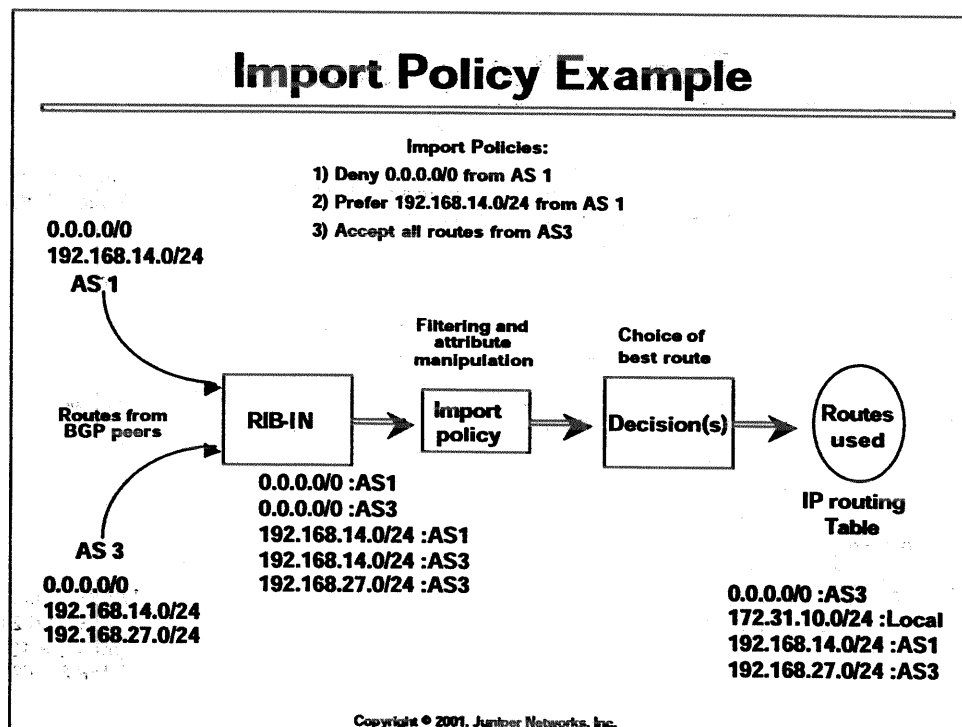


Copyright © 2001, Juniper Networks, Inc.

BGP stores the information about routes to be advertised to BGP peers in the RIB-OUT table.

As BGP moves the routes from the RIB-LOCAL table to the RIB-OUT table, the JUNOS software routing policy framework can apply export policies. These policies can reject (filter) routes and affect what BGP routes are advertised to BGP peers or can change the BGP attributes of advertised routes.

In addition, new routing information can be injected into the BGP routing process at this point.

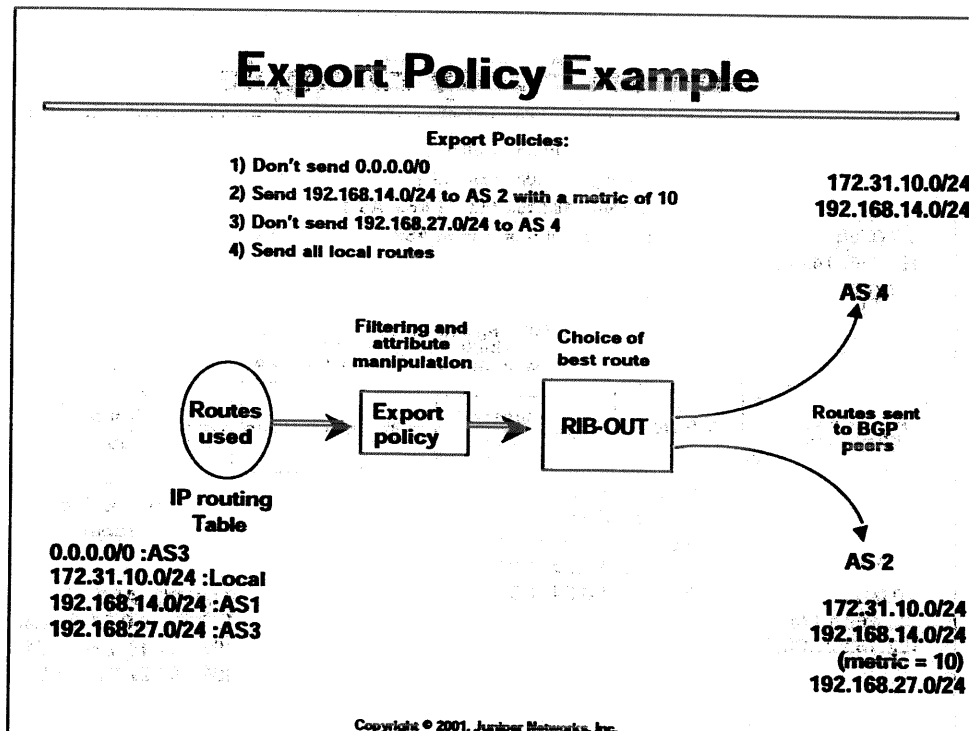


The slide shows an example of some BGP import policies in action.

Some policies have been configured that reject the default route (0/0) if received from AS 1, prefer the AS 1 version of 192.168.14.0/24, and accept all other routes from AS 3. These three policies, which might be three separate policies or one policy with three terms, are then configured as import policies within BGP.

As the BGP process attempts to move those routes from the RIB-IN table to the BGP decision process to select the best BGP route, the import policies are applied. The net result of this policy application is shown on the slide and as follows:

- The 0.0.0.0/0:AS1 route has been **rejected**
- The 192.168.14.0/24:AS3 route has been **rejected**
- All other AS3 routes have been **accepted**



The slide shows an example of some BGP export policies in action.

Some policies have been configured that reject the default (0/0) route, do not send some routes to AS4, alter a BGP attribute on routes sent to AS2, and inject new route information into the BGP process. These four policies, which might be four separate policies or one policy with four terms, are then configured as export policies within BGP.

As the BGP routing process attempts to move those routes from the RIB-LOCAL table to the RIB-OUT tables, the export policies are applied. The net result of this policy application is shown on the slide and as follows:

- The 0.0.0.0/0:AS3 route has been **rejected and is not advertised**
- The 192.168.27.0/24:AS3 route has only been **sent to AS 2**
- The 192.168.14.0/24 route has been **sent to both AS2 (with a metric of 10) and AS 4**
- The 172.31.10.0/24 has been **sent to both AS 2 and AS4**

BGP Path Selection (1 of 2)

1. Can the BGP Next-Hop (BNH) be reached?
 - If yes, proceed.
 - If no, stop processing.
2. Prefer the highest LOCAL-PREF value.
3. Prefer the shortest AS-PATH length.
4. Prefer the lowest ORIGIN value.
5. Prefer the lowest MED value.
6. Prefer paths learned via EBGp over routes via IBGP.

Copyright © 2001, Juniper Networks, Inc.

This is a high-level look at the BGP path selection process to determine the active path when BGP knows about more than one way to reach a destination. This is not an exhaustive look at the process at the microcode level, but it is representative.

After the router has verified that it has a current route to the IP address in the BGP Next-Hop attribute (next-hop reachable?) for that path, the BGP process performs these steps in order until it has the single best BGP path to every route destination. At any point when a single path remains for a route, the process will stop and select that path as the active route.

Paths are first compared for the *highest* local preference (the only choice based on a higher rather than lower value). Next comes the shortest AS path, then lowest origin code, and lowest multi-exit discriminator (MED). These four BGP attributes are so important to BGP performance that each will be explored in detail in modules to come.

If all is exactly the same to this point, EBGp paths are preferred over IBGP paths. Why? Because EBGp's AS path prepending offers better loop detection and prevention than available in IBGP. If the paths were all learned through EBGp, the next four steps are skipped.

BGP Path Selection (2 of 2)

7. Prefer paths with the lowest IGP metric.
8. Prefer paths where BNH is resolved in inet.3 over inet.0
9. Prefer paths where BNH has greater number of equal-cost paths
10. Prefer paths with the shortest Cluster-List length
11. Prefer paths from the peer with the lowest RID.
12. Prefer paths from the peer with the lowest peer ID.

Copyright © 2001, Juniper Networks, Inc.

If the paths were learned through IBGP, then the lowest IGP cost to the peer is used as the next tie-breaker. The theory here is that the IGP should be authoritative when it comes to routes within the AS.

Should all the remaining paths have the same IGP cost to the peer, the BGP process will examine both the inet.0 and the inet.3 route tables. If a route to the peer is found in the inet.3 table (via an MPLS LSP), then that version of the path will be selected. The theory behind this decision is that an MPLS LSP is a traffic engineered connection and it should be used if available.

If all available paths are still known via a single route table (inet.0 OR inet.3), then any next-hops reachable via multiple equal-cost paths is chosen. This allows the default BGP per-prefix load balancing to occur and spreads BGP transit traffic across multiple links within the AS.

Next, if no active path has yet been determined, the shortest cluster list is selected. Clusters are used with BGP route reflectors and are discussed later in this course.

Then the lowest router ID (usually the loopback IP address) is used as a tie-breaker.

The last step is used when there are multiple links between adjacent BGP peers that are used for the EBGP sessions. Each session will have a peer ID associated with it. The lowest peer ID is used as the last active path selection step.

Path Selection Notes

- **Peer-ID is used when there are multiple peering sessions between two routers**
 - Only one session will be used to forward traffic
 - ID is the physical IP address on the neighboring interface
- **Router-ID and Peer-ID comparisons can both be ignored when multipath is configured within BGP**
 - Two peering sessions to the same router can be used
 - Two peering sessions to different routers can be used
 - Two peering sessions to different AS networks can be used
- **Multipath cannot be used with multihop**

Copyright © 2001, Juniper Networks, Inc.

We discussed the case where two routers have multiple links between them for EBGp peering. In that case there is still only a single peering session running between the routers. So routes from the remote router would be seen as advertised from the same peer.

It is possible in such a multi-link configuration to setup two peering sessions between the two routers, one on each link, for redundancy. In this situation, the remote router would send two copies of the same route to the local router. The local router would see both copies as being advertised by the same remote router with the same router-id. Since BGP can only use one path to every route destination, there needs to be a tie-breaker to pick between those two copies. That tie-breaker is the peer-id associated with each copy of the route. The peer-id in this case would be the physical IP address of the far end of the link. The lower of the two IP addresses would be selected and the copy of the route from that lower IP address would be installed into the local routing table.

However, both the peer-id and the router-id selection criteria can be ignored by the selection process for EBGp received routes. This is accomplished via the **multipath** statement. Once configured, routes from multiple peering sessions can be installed into the routing table. This allows multiple copies of a route from the same remote router. It also allows multiple copies of a route from two different routers in the same neighboring AS. It even can use multiple copies of a route from two routers in two different AS networks. The entire concept centers around resiliency and redundancy.

It is important to note that these multiple EBGp connections **MUST** be over single hop links. The **multipath** statement can not be used with the **multihop** statement. They are mutually exclusive BGP options.

BGP Next_hop Attribute

Where we are going...

- How BGP Uses Next_hop
- BGP Next-hop Example
- BGP Next_hop Resolution
- Controlling Next-hop Options
- Next Hop Self

Copyright © 2001, Juniper Networks, Inc.

Subsequent slides will discuss the BGP attribute of Next-hop, why it is important, and how potentially solve reachability problems associated with the attribute.

How BGP Uses Next_hop

- **Next_hop concept in IGP is straightforward**
- **Next_hop in BGP is more elaborate**
- **Default forms of BGP Next_hop information:**
 - **EBGP sessions: next hop is IP address of neighbor that announced the route**
 - **IBGP sessions: for routes originating inside the AS, next hop is IP address of neighbor that announced the route**
 - **IBGP sessions: for routes injected into AS via EBGp, next hop from EBGp session is placed unchanged into IBGP**
 - **IP address of EBGp neighbor from which the route was learned**

Copyright © 2001, Juniper Networks, Inc.

The next-hop concept in IGPs such as IS-IS and OSPF is relatively straightforward. IGPs basically exist to give adjacent routers next-hop reachability information, which is then flooded (in one form or another) throughout the AS. But BGP does not flood: BGP peers. And BGP cannot peer unless there is an underlying IGP route to the peer, since BGP requires a TCP session in order for routes to be exchanged. So BGP cannot "bootstrap" its own next-hops as an IGP does.

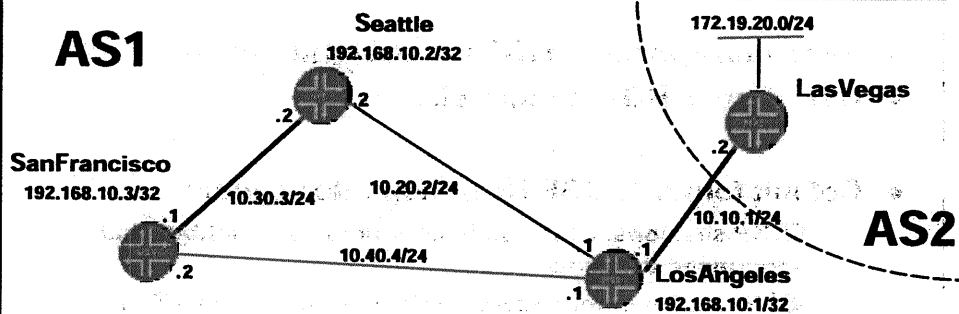
So a next-hop in BGP is more elaborate than in any IGP. The concept of the BGP Next-hop attribute is an important one. Without reachability to the IP address that is listed in this BGP attribute, a BGP router can not use the advertised route. This is a very common problem to be solved in any ISP network.

The default actions for changing the next-hop attribute are detailed below. Later slides will discuss the various ways to provide the required reachability.

The BGP next-hop attribute value is only changed when a route is advertised across an EBGp link. In this situation, the IP address of the remote router is placed into the attribute. Should the local router choose to advertise this EBGp learned route to any IBGP peers, it does so without modifying that IP address value. So the EBGp next-hop advertised into a local AS is an address from the remote AS. How is the local IGP supposed to know how to reach this external address?

When new routing information is injected into the BGP process, the default attribute value is set to the local router's peer-id. For IBGP sessions, this is typically the local router's loopback address. For EBGp sessions, this is usually a physical link address on a point-to-point link.

BGP Next_hop Example (I)



```

user@LosAngeles> show bgp summary
Peer      AS      InPkt  OutPkt  OutQ  Flaps  Last Up/Dn  State|#Active/Rec..
192.168.10.2  1      10      12      0      0      4:13 0/0/0
192.168.10.3  1       2       4      0      0      14 0/0/0
10.10.1.2     2      18      20      0      0      8:23 1/1/0

user@LosAngeles> show route terse
inet.0: 28 destinations, 28 routes (28 active, 0 holdown, 0 hidden)
A Destination P Prf Metric 1 Metric 2 Next hop AS path
* 172.19.20.0/24 B 170 100 >10.10.1.2 2 I
* 192.168.10.2/32 I 18 10 >10.20.2.2
* 192.168.10.3/32 I 18 10 >10.40.4.2
  
```

Copyright © 2001, Juniper Networks, Inc.

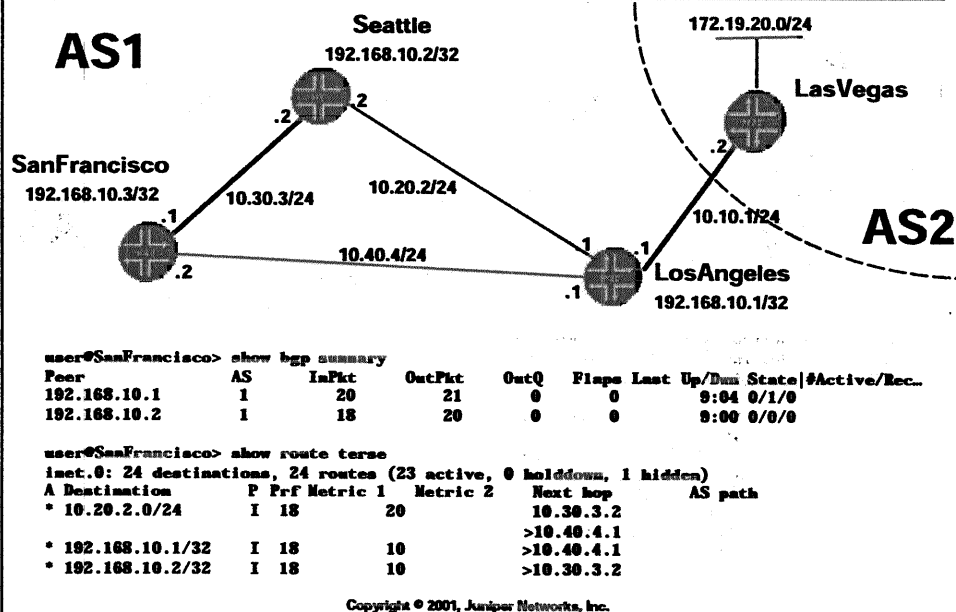
The next few slides graphically show the default BGP behavior with respect to the next-hop attribute. The physical interface addresses are shown along with the loopback addresses of each router.

The slide shows the details for the *LosAngeles* router. The *LosAngeles* router has an EBGP session with the *LasVegas* router using 10.10.1.2.

The output of the **show bgp summary** command is listed on the slide. *LosAngeles* is receiving one route from *LasVegas* (peer 10.10.1.2) and is actively using that route (last column is 1/1/0).

This can be verified with the **show route terse** output where 172.19.20.0/24 is listed as an active route from protocol BGP and a next-hop of 10.10.1.2.

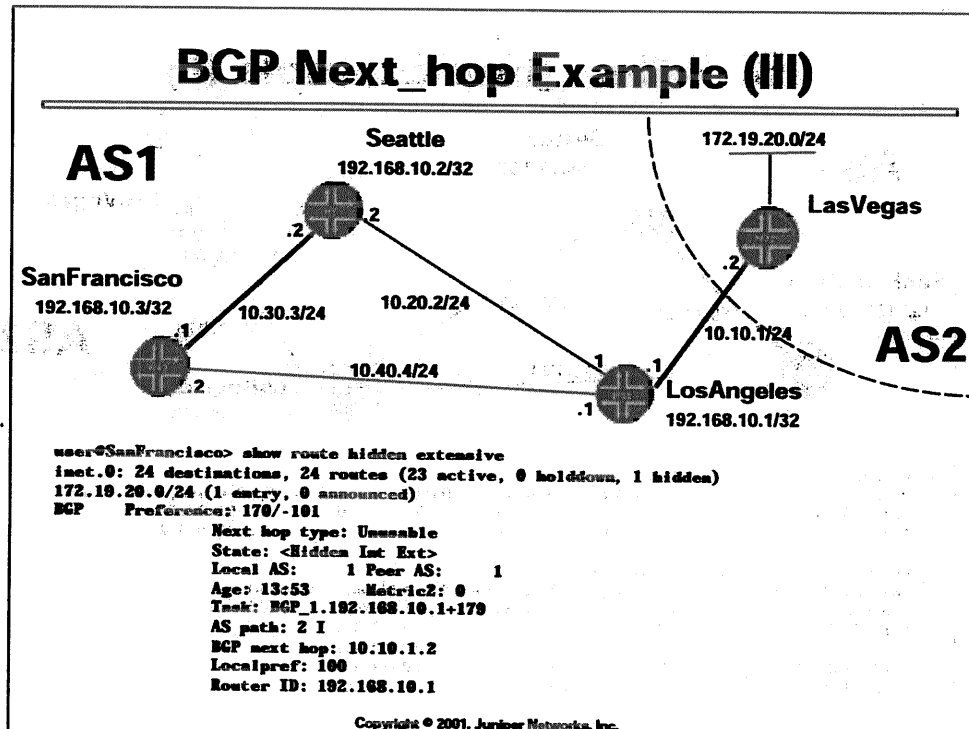
BGP Next_hop Example (II)



This slide moves over to the *SanFrancisco* router. The *SanFrancisco* router has an IBGP session with *LosAngeles* using 192.168.10.1.

The output of the **show bgp summary** command shows that *SanFrancisco* is receiving one route from *LosAngeles* (peer 192.168.10.2), but it is not being used to forward traffic (last column is 0/1/0).

In fact, a look at the **show route** output does not show 172.19.20.0/24 (the active BGP route on the *LosAngeles* router) listed at all. If the 172.19.20.0/24 route is being received by *SanFrancisco* from *LosAngeles*, then the route should appear as the active route (there is no chance of an AS1 IGP also supplying this AS2 route). There must be some problem here.



That problem with the absent route to 172.19.20.0/24 on the *SanFrancisco* router can be seen here with the help of the **show route hidden extensive** command run on *SanFrancisco* and shown on the slide.

The output from this command shows the 172.19.20.0/24 route, but the route is hidden. By examining the output a little more closely, we can determine that the next-hop is unusable (Next hop type: Unusable).

Why is this route unusable? Obviously, there is connectivity from *SanFrancisco* to *LosAngeles*. But notice that the current BGP Next-hop attribute for the 172.19.20.0/24 route is set to 10.10.1.2 (the physical interface IP address of *LasVegas*). This is because the BGP Next-hop attribute is not changed by *LosAngeles* before the route is advertised to *SanFrancisco*. This is the default EBGP-to-IBGP behavior: do not change the advertised Next-hop attribute value.

The **show route terse** output on the previous slide did not show a route to 10.10.1.2. Since *SanFrancisco* does not have reachability to the IP address listed as the BGP Next-hop attribute, it can not use the received BGP route. On a Juniper Networks router, this type of unusable route is marked as a hidden route.

BGP Next_hop Resolution

- **Next-Hop Self**
 - Use a policy to alter the Next-hop value
 - Change the BGP Next-hop to be the address of the IBGP peer
- **Export Direct routes into the IGP**
 - Use a policy to send interface prefixes to IBGP peers
 - Adds interface prefixes to the IGP routing tables
- **IGP Passive Interface**
 - IGP advertises interface prefixes to EBGp peers, no adjacency
 - Adds interface prefixes to the IGP routing tables
- **Static routes**
- **IGP adjacency formed on inter-AS links to EBGp peers**

Copyright © 2001, Juniper Networks, Inc.

There are numerous ways to solve this BGP Next-hop reachability problem and five examples are listed on the slide. Some of these examples are **NOT** best practices in a networking environment, but will technically solve the reachability issue. Some are in the category of "you can use a hammer to swat a fly on a window, but you might want to use something else..."

Perhaps the most commonly used (and recommended) solution is known as *Next-hop self*. This means that when a BGP router advertises an EBGp learned route to an IBGP peer, it alters the BGP Next-hop attribute. The Next-hop attribute's IP address of the remote EBGp peer is replaced with the IP address of the BGP router itself. Since the IBGP session was most likely established using the peer's loopback address, this new BGP Next-hop value will be reachable and the advertised BGP route can be used. Next-hop self is done using a policy that matches on specific routes with an action of changing the Next-hop attribute value. This policy is then applied as an export policy to any IBGP peers.

The next two options listed (*export direct routes* and *IGP passive*) are almost identical in their results. The difference between the two is in the approach that each takes to provide reachability. With *export direct*, the address assigned to the point-to-point link between the two EBGp peers is advertised to all IBGP peers by the Interior Gateway Protocol (IGP) operating in the AS with a routing policy. With *IGP passive*, the IGP is configured on the inter-AS link and advertises the interface addresses, but forms no adjacency (it's "passive"). Both methods inject the interface addresses into the local routing table for the IGP to use.

Export direct uses a JUNOS software routing policy to retrieve the subnet information from the local routing table. Within *inet.0*, these networks are known as protocol *direct*. The policy matches on these *direct* routes and accepts them. This policy is then applied as an export policy to the local IGP.

An *IGP passive* interface allows the local IGP to advertise the subnet on a particular interface without forming an adjacency at the IGP level to the remote EBGp peer. This has the advantage of not using a policy, but requires explicit configuration for each interface and subnet address that the administrator wishes to advertise.

The last two options listed on the slide (*static routes* and forming an *IGP adjacency* relationship with the remote EBGp router) have some severe disadvantages, but they both will work.

Static routes have an inherent scalability problem. Each IBGP router in the network needs to be configured for a single static route for each remote EBGp peer. The more EBGp peers in the network, the more static routes are required. The more IBGP peers in the AS, the more places that additions and changes have to be made. Clearly, this is not a "real world" option.

With regard to the *full IGP adjacency* between AS networks, while reachability information can be provided by forming an IGP relationship with the remote EBGp peer, this is not a recommended practice. This is because of the very trusting nature of the IGP protocols. Once this adjacency has been formed, the protocol will accept ANY routing information told to it by the remote EBGp peer. This is very dangerous since the remote AS might inject bad information into your network. In addition, this method potentially violates the entire idea of having *autonomous* (independent of the IGP) systems in the first place.

Controlling Next_hop Options

- How should BGP resolve its next hops?
- When MPLS is used, BGP examines both inet.0 and inet.3 by default
 - Best longest-match route (based on Preference) is picked as the next hop
 - inet.3 (MPLS) is preferred when there is a preference tie
- **protocol** keyword can control which IGP route (IS-IS, OSPF, or RIP) in inet.0 is used as the next hop

```
[edit protocols bgp]
group int-peers {
  type internal;
  protocol IS-IS;
  local-address 172.16.1.1;
  neighbor 172.16.2.2;
}
```

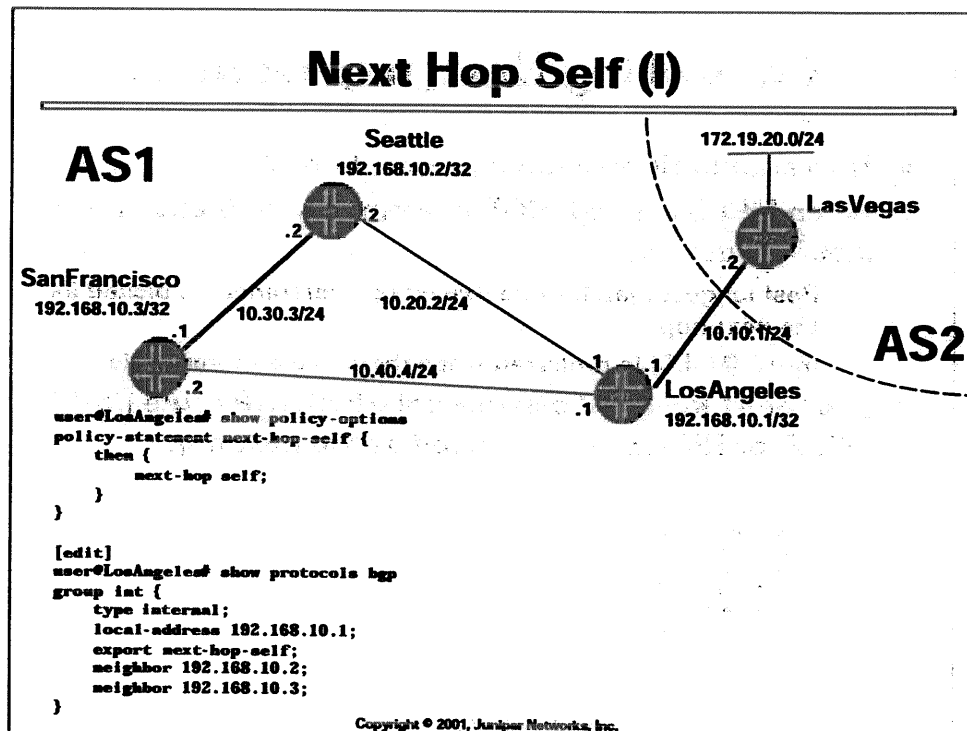
Copyright © 2001, Juniper Networks, Inc.

Once reachability to the BGP Next-hop has been achieved, most simply and easily with Next-hop self, the JUNOS software provides additional features for telling the router how to use that information. This is important to BGP since BGP relies on the IGP to resolve its next hops (BGP cannot "bootstrap" its own next hops like an IGP can).

One option for BGP next hop resolution is through the use of Multi-Protocol Label Switching (MPLS). MPLS establishes paths through an AS by way of a Label Switched Path (LSP). Once this LSP has been established and is operational, information about this new network path is placed into the inet.3 routing table.

When a BGP router examines its local routing table to determine if the BGP Next-hop is reachable, it examines the information in **BOTH** inet.0 and inet.3. The first task is to perform a longest-match for the next-hop value. If there is only one match found, then that next-hop is used for the BGP route. If multiple matches are found, then the protocol preference values are used as a tiebreaker. By default, the LSP preference value is 9, which is more preferred than OSPF (10) and IS-IS (18). So the LSP next-hop value will be used for the BGP route. Should the protocol preference values be the same in inet.0 and inet.3, then the next-hop information from inet.3 will be chosen by default.

Another option for BGP next hop resolution involves multiple longest matches within inet.0. The information causing the multiple matches will be primarily from IGP protocols such as OSPF, IS-IS, or RIP. Which IGP should BGP use to resolve next-hops? To override the default action of using protocol preference values, network administrators can use the **protocol** keyword and specify OSPF or IS-IS or RIP. This statement effectively limits the BGP next-hop resolution to routes learned by a specific protocol. For example, if a needed BGP next-hop value is learned by OSPF but **protocol IS-IS** has been configured, then the OSPF route is ignored and BGP next-hop reachability is not achieved.

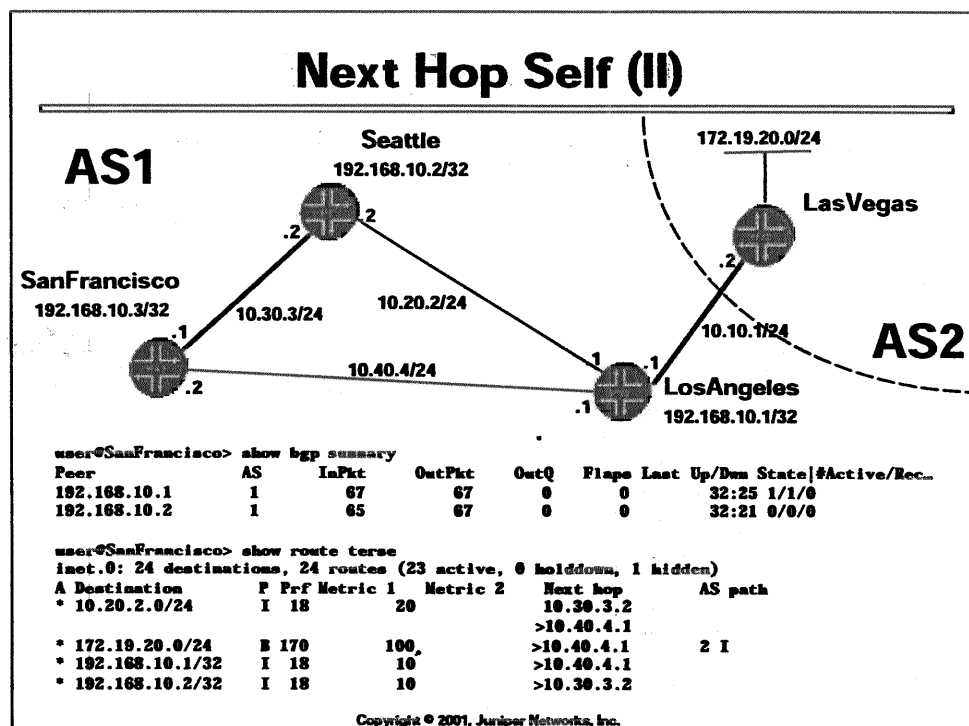


The next few slides will show how the use of **next-hop self** in a routing policy provides reachability for IBGP peers.

The slide details the situation on the *LosAngeles* router. The *LosAngeles* router has an EBGP connection to *LasVegas* using 10.10.1.2 and two IBGP connections to *Seattle* and to *SanFrancisco*.

LosAngeles is now configured with the **next-hop-self** policy shown on the slide as the output of the **show policy-options** configuration command. The policy matches all routes (no **from**) and will replace the BGP Next-hop attribute (since only BGP has a Next-hop attribute) with the value **self** (a keyword for the local router's physical interface address that the route is advertised on).

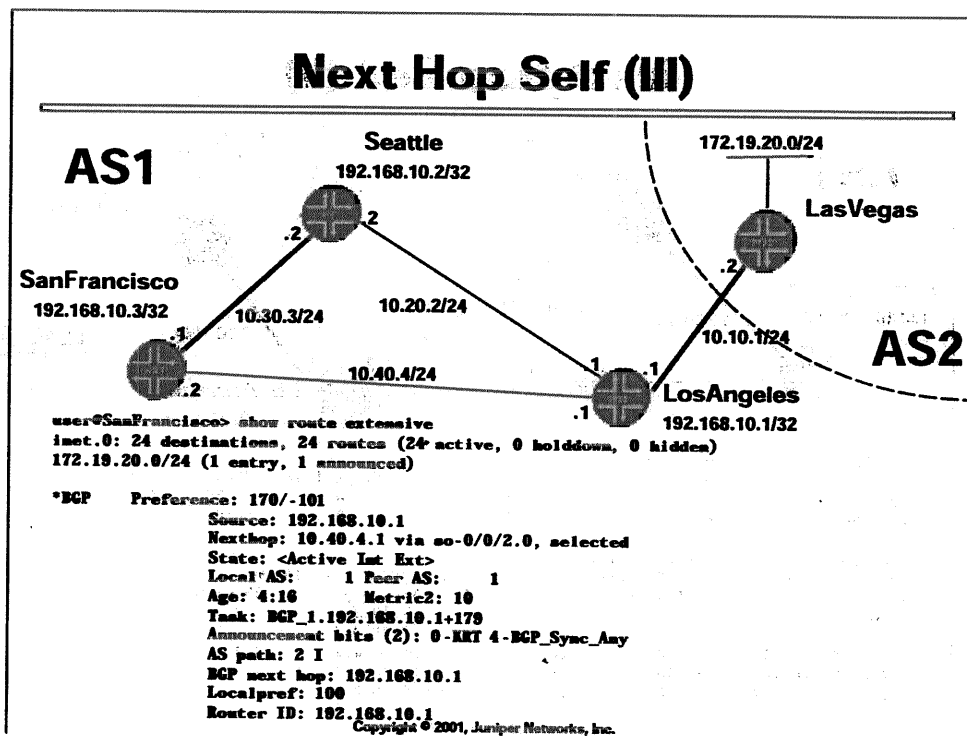
This **next-hop-self** policy is applied as an **export** policy to the BGP group **int**, which includes the *SanFrancisco* router.



This slide shifts the focus of attention to the *SanFrancisco* router.

The output of the **show bgp summary** command now shows that *SanFrancisco* is receiving one route from *LosAngeles* (192.168.10.1) and that this route is actively being used to forward traffic (last column is 1/1/0).

A look at the **show route terse** output now shows the 172.19.20.0/24 listed as a BGP route with a next hop of 10.40.4.1. This next hop address is the "self" next hop address advertised for the route to *SanFrancisco* by *LosAngeles*, since the route was advertised on the 10.40.4.1 interface.



How do we know that the BGP Next-hop attribute was really changed with the policy?

To verify that the policy on the *LosAngeles* router has indeed changed the BGP Next-hop attribute, we can look at the output of the **show route extensive** command on the *SanFrancisco* router.

The output lists the BGP Next-hop 172.19.20.0/24 (which is in AS2) as 192.168.10.1, the loopback address of the *LosAngeles* router. Since *SanFrancisco* has IGP reachability to *LosAngeles*, *SanFrancisco* also has reachability to the BGP Next-hop value and can use the BGP route.

Large-Scale BGP Routing

Where we are going...

- Scaling BGP
- Scaling BGP with Route Reflection
- Basic Route Reflection
- Hierarchical Route Reflection
- Scaling BGP - Confederations
- Confederations

Copyright © 2001, Juniper Networks, Inc.

Subsequent slides will discuss how BGP can be adapted to a large-scale ISP network through the use of Route Reflection and Confederations.

Scaling BGP

- **IBGP full-mesh peer requirement has an N-squared problem**
 - Addition of a new router requires new peering with all current IBGP speakers
 - Current IBGP speakers must update configurations
- **Two methods are available for scaling IBGP connectivity**
 - Route Reflection (RFC 2796)
 - Confederations (RFC 3065)

Copyright © 2001, Juniper Networks, Inc.

BGP is not a flooding routing protocol as many IGPs are (like OSPF, IS-IS, etc.), but a peering routing protocol that exchanges routes directly with meshed peers. The requirement for all IBGP peers to be fully meshed within an Autonomous System has inherent scalability problems. When a new router is added to the AS, each current IBGP router needs to have their configurations updated. This can become quite an issue when there are 100, 200, or even 1000 routers in an AS.

There are two main methods for assisting in the scalability of IBGP. These are the use of Route Reflection (described in RFC 2796) and Confederations (sometimes called Sub-confederations, and described in RFC 3065).

Scaling BGP with Route Reflection

- Allows an IBGP speaker to re-advertise an IBGP learned route to another IBGP speaker
- Route Reflector (RR) only re-advertises the best route to "clients"

- Clients are configured in a separate peer group
- Each peer group uses the **cluster** keyword

```
[edit protocols bgp]
group int-peers {
  type internal;
  local-address 172.16.1.1;
  cluster 172.16.1.1;
  neighbor 172.16.2.2;
  neighbor 172.16.3.3;
  neighbor 172.16.4.4;
}
```

- IBGP attributes are not changed by reflection
- Routing Loops are prevented by new BGP attributes of Cluster-id, Cluster-list, and Originator-id

Copyright © 2001, Juniper Networks, Inc.

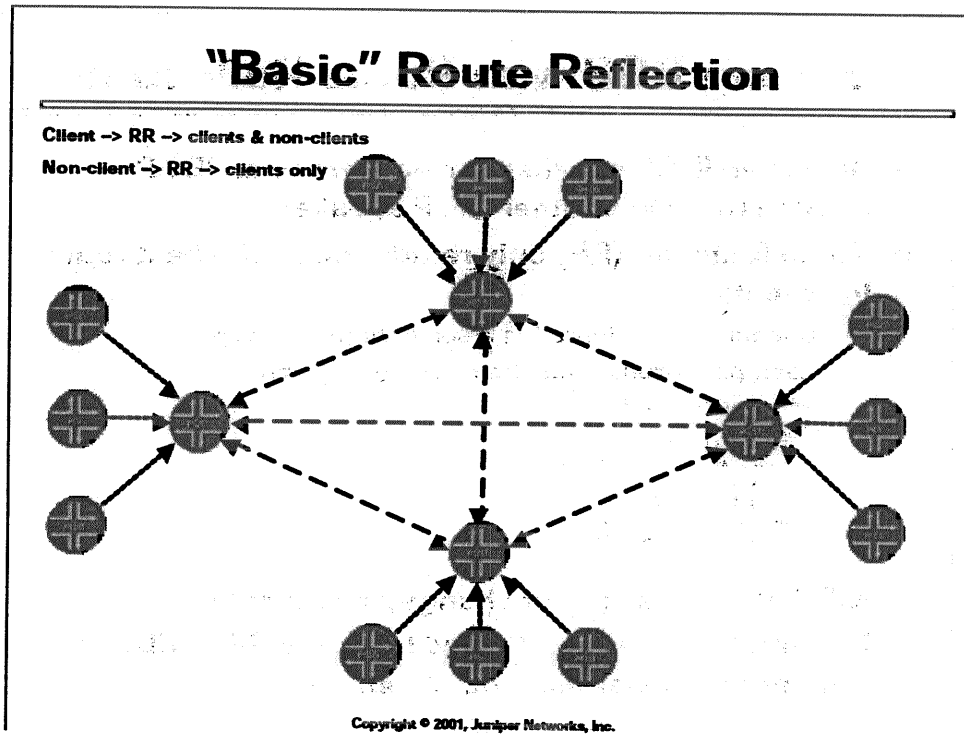
BGP route reflection relaxes the default IBGP peer action of not re-advertising an IBGP learned route (which is what flooding does). The routers that are allowed to override this default are known as Route Reflectors (RR).

RRs only re-advertise the best routes to their "clients." An RR is activated and clients configured through the use of the **cluster** statement within an IBGP peer-group. Each of the peers configured within that peer-group are considered clients of the RR. The RR clients do not have any knowledge of the presence of the RR, they simply see the RR as an IBGP peer.

The basic operation of the RR is that any routes the RR receives from a client get re-advertised to all other clients of the RR and all non-clients of the RR. All routes received by the RR from non-client peers get re-advertised to all RR clients. In this fashion, route reachability is achieved for all BGP speakers in the AS.

One of the primary drivers behind requiring the IBGP full-mesh in the first place was loop prevention since the AS-Path attribute does not get modified within an AS. Route Reflection does not change that behavior. In fact, *none* of the BGP attributes change when Route Reflection is used in an AS. But loop prevention is still a critical part of BGP, so new BGP attributes were introduced to assist a Route Reflection network.

The new attributes are the *cluster-id*, the *cluster-list*, and the *originator-id*. All of these attributes are modified by the RR when it re-advertises the routes to both clients and non-clients. The *cluster-id* is very similar to an AS number and should be unique within an individual AS. When a route is reflected by the RR, it adds its *cluster-id* to the *cluster-list*. The *cluster-list* is analogous to the AS-Path attribute and is used to prevent loops. If a RR receives a route with its own *cluster-id* in the *cluster-list*, it drops the route. The *originator-id* is the peer address of the first router to advertise the route in the AS. It is also used for loop prevention in the rare case where the *cluster-list* does not prevent a loop.



The slide shows an AS network using a common route reflection topology. BGP speaking routers along the edge of the network all have a single peer configured. This peer is the RR for the local cluster.

The RRs are then fully meshed together via standard IBGP peering. In this fashion, all routes received by any BGP router will reach all other BGP routers in the AS.

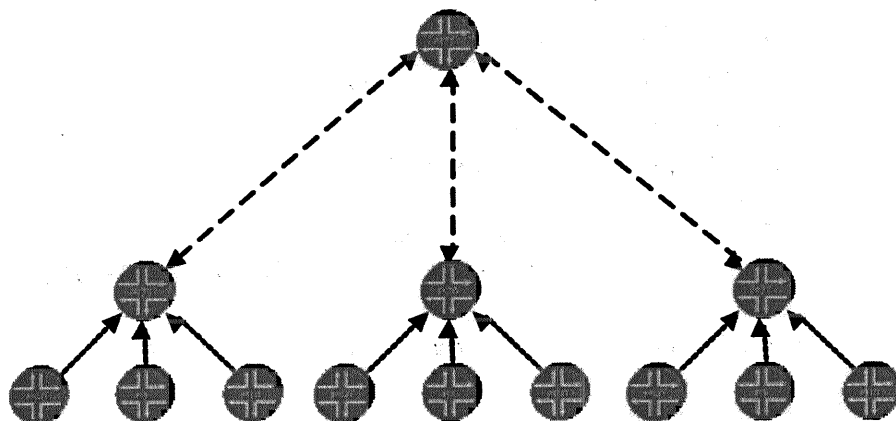
The slide also shows the basic operation of the RR. Any routes the RR receives from a client get re-advertised to all other clients of the RR and all non-clients of the RR. All routes received by the RR from non-client peers get re-advertised to all RR clients.

Naturally, a Route Reflection configuration should be guided by the physical topology of the network.

Hierarchical Route Reflection

Client → RR → clients & non-clients

Non-client → RR → clients only



Copyright © 2001, Juniper Networks, Inc.

The slide shows an AS network using a more complex route reflection topology known as hierarchical route reflection.

Hierarchical route reflection is when the RRs for some clusters are themselves clients in another route reflection cluster. Very often AS networks evolve to this type of a setup when the RR full mesh shown on the previous slide becomes too large. The internal RR full mesh then is transformed into a RR cluster.

The slide again shows the basic operation of the RR. Any routes the RR receives from a client get re-advertised to all other clients of the RR and all non-clients of the RR. All routes received by the RR from non-client peers get re-advertised to all RR clients.

Again, a Route Reflection configuration should be guided by the physical topology of the network.

Scaling BGP - Confederations

- Breaks a global AS into multiple pieces (Sub-AS)
- Within each sub-AS:
 - Use private AS numbers
 - An IBGP full-mesh is still required
- Between each sub-AS:
 - EBGP-type configurations (CBGP) are required (multihop, etc.)
 - Most IBGP attributes are not changed
 - AS_Path is modified to prevent loops (but sub-AS is NOT a hop)
- Global AS is still viewed externally as a single AS

```
[edit routing-options]
autonomous-system 65000;
confederation 2 members [ 65000 65001 65002 ];
```

Copyright © 2001, Juniper Networks, Inc.

The second method for easing scalability issues with BGP is through the use of a BGP *Confederation*. Quite simply, a BGP Confederation takes an Autonomous System and breaks it up into smaller sub-Autonomous Systems. These sub-AS networks are then connected together to form the unified AS that the Internet (all other ASs) sees.

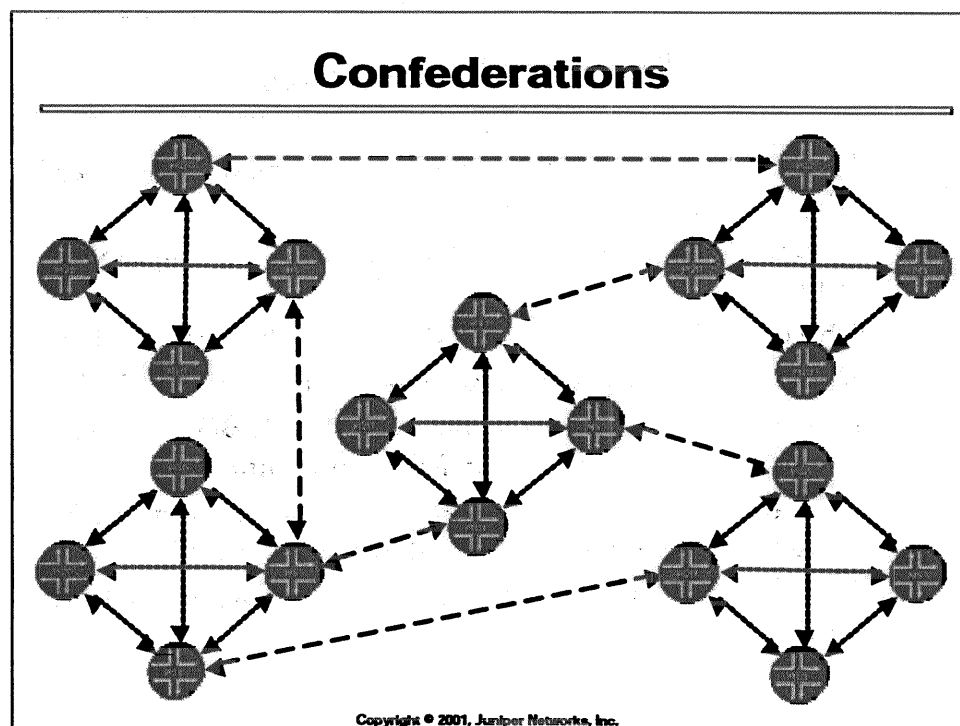
The confederation sub-AS networks act just a "real" AS. Confederations must provide full mesh IBGP connectivity within themselves. Confederations must have a unique AS number defined. Most sub-AS networks use an AS number out of the private AS range of 64512-65535.

The sub-AS networks are then connected via an EBGP-like connection that is often referred to as Confederation BGP (CBGP). The CBGP links only modify the AS_Path attribute to add in the sub-AS numbers to assist with loop prevention. All other BGP attributes like Local_Pref and MED remain unchanged. The CBGP links can also traverse multiple hops since they are within a single global AS. However, the **multihop** statement needs to be used since the router views the connection as an EBGP link.

To the Internet, the Confederation is viewed as a single Autonomous System. The AS_Path received by other AS networks in the Internet only shows the globally assigned AS number, not the sub-AS numbers. This is because the sub-AS numbers are removed as the route is advertised out of the global AS.

Although the sub-AS numbers appear within the AS_Path attribute inside of the Confederation, all BGP routers in the global AS see these sub-AS values as a single AS hop.

To effectively operate a Confederation network, all BGP routers in the global AS need to know the globally unique AS number as well as all the configured sub-AS numbers. This is done with the **confederation** statement, as shown on the slide.



The slide shows an AS network using a Confederation topology. Within each of the sub-AS networks, all of the BGP speaking routers are fully meshed with IBGP.

The sub-AS networks are then connected with EBGP-like connections (sometimes called CBGP). Since the CBGP links behave like EBGP, there is no need for a full-mesh. Hence, the CBGP connections can be used wherever it is convenient.

Review Questions

- To what IP addresses to EBGp sessions usually peer?
Why?
- To what IP addresses to IBGP sessions usually peer?
Why?
- Why does BGP rely on an IGP for next hop resolution?
- What is "next hop self" and when is it used in BGP?
- What does a BGP route reflector do with BGP routes received from a configured RR client router?
- What three BGP attributes are introduced when BGP route reflection is used and what is their role?
- What form of BGP is run between the routers in a BGP Confederation?

Copyright © 2001, Juniper Networks, Inc.

During this module we discussed:

- The JUNOS software configuration options for BGP.
- The default behavior of the BGP routing protocol with regard to the "movement" (use and advertising) of BGP routes through the router.
- How the BGP attributes play a role in the process of choosing routes to use and advertise through BGP.
- When BGP next-hop reachability was an issue and how BGP routes find their next-hops.
- The available options for handling IBGP scalability issues such as the number of peer sessions that must be maintained.



Juniper Networks Advanced Policy

Release 5.0, Revision 1

Module 6: BGP Attributes: Origin and Multi-Exit Discriminator (MED)

Copyright © 2001, Juniper Networks, Inc.

Objectives

- **Describe the use of the BGP Origin attribute**
- **Configure a policy to alter the BGP Origin**
- **Explain the role played by the Multi-Exit Discriminator (MED) in BGP operation**
- **Describe the operation of the BGP MED attribute**
- **Configure a policy to change the MED value**

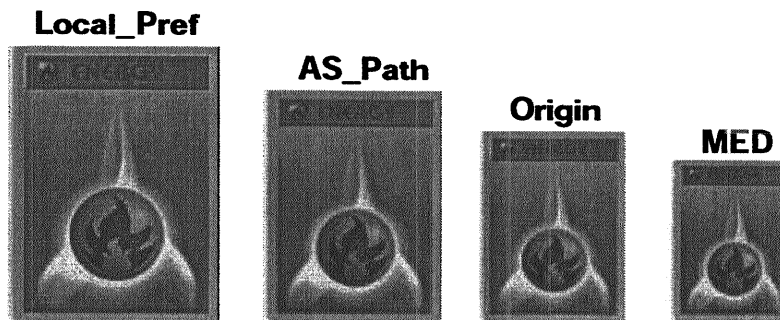
Copyright © 2001, Juniper Networks, Inc.

This module discusses:

- The use of the BGP Origin attribute.
- Configuring a routing policy to alter the BGP Origin attribute.
- The role played by the Multi-Exit Discriminator (MED) BGP attribute in BGP operation.
- The operation of the MED attribute and how it influences BGP route selection.
- Configuring a routing policy to change the MED value and the effects this has on AS to AS traffic flows.

Origin and MED Attributes

- After Local Preference and AS Path Length, BGP looks at Origin Code and MED to select a path to use
- This is relatively high in the BGP decision process
- Both Origin and MED can be used to attempt to influence inbound traffic



Copyright © 2001, Juniper Networks, Inc.

In many cases, routers running BGP learn about routes to a particular destination from more than one source over more than one interface. By default, only one route from among those alternatives can become the active route or path to a destination. BGP has a series of decisions to make for path selection. Four BGP attributes are considered as part of this decision process. In order, the attributes are: *Local Preference*, *AS Path*, *Origin*, and *Multi-Exit Discriminator (MED)*.

Both the Origin and the MED BGP attributes are relatively high in the path selection process to choose active routes from among BGP alternatives. As shown in the slide, Origin and MED fall only behind Local Preference and the AS Path in order of BGP attribute importance.

Both Origin and MED can be set by using various configuration statements and both can be modified by using the JUNOS software policy framework. The modification of these attributes is most often applied as a way to influence the path that user traffic takes as it returns to the Autonomous System (AS) through the Internet. That is, the Origin and MED are modified on routes sent (exported) to another AS, where it is hoped that these attributes will influence the path that traffic takes *inbound* to the original AS.

Both Origin and MED attempt to do this by making some routes leading back to the original AS look less attractive than others when considered by the remote AS. Rather than making one route appear *better* than another, Origin and MED alterations make a route back to the original AS look *worse* than another. As such, both Origin and MED are said to exert a *negative bias* (negative influence) on routes exported to remote AS.

BGP Origin Attribute

Where we are going...

- The Origin Code
- Use of the Origin Attribute
- Origin Example
- Changing the Origin Code

Copyright © 2001, Juniper Networks, Inc.

Subsequent slides will explore how the BGP Origin code is used and how to configure a policy to alter the attribute value.

The Origin Code

- Installed by the originating router for the prefix (route)
- A tag of "believability" as to the origin of the route information (*Where did you get it from?*)
- BGP ORIGIN code is a well-known mandatory attribute (Type Code 1)
- Origin can be internal, external, or unknown
 - I: Internal (0) – Learned from an IGP
 - E: External (1) – Learned from EGP
 - ?: Incomplete (2) – NLRI found by some other means
- "I" (0) is better than "E" (1) which is better than "?" (2)
- All JUNOS software BGP routes have origin IGP by default

Copyright © 2001, Juniper Networks, Inc.

The Origin attribute is attached to a prefix (route) by the first router to inject the route into BGP. Other routers can change this value, of course, as the route makes its way through an AS and on to other ASs.

The intent of the Origin code was a measure of "believability" as to the origin of a particular route. In other words, the intent was to provide a kind of "where did you get this from?" clue for other routers seeing the route.

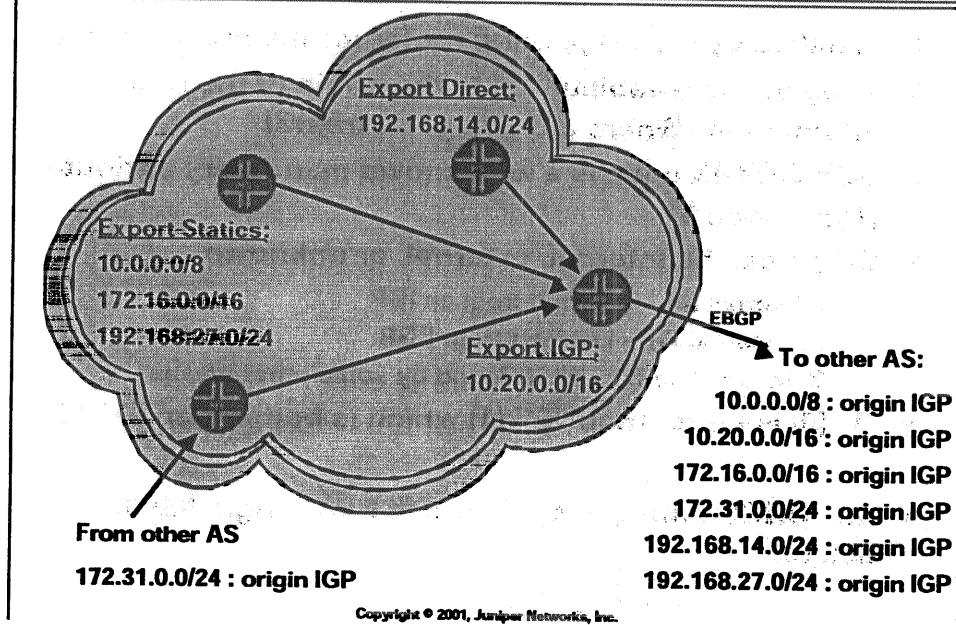
The Origin code is a well-known, mandatory BGP attribute (the Type Code is 1), meaning that each route associated with BGP must have an Origin code assigned to it.

The values available for origins include internal, external, or incomplete. The internal (abbreviated I) origin was a tag designated for all routes learned through a traditional Interior Gateway Protocol (IGP) such as OSPF, IS-IS, or RIP. These types of routes were typically seen as "best" sources of information due to their stability at the time BGP was created. The external (abbreviated E) origin was a tag designated for routes learned through the original Exterior Gateway Protocol (EGP) called EGP. This was the precursor to BGP but was not as robust and was generally less reliable than the IGP routes. The last origin of Incomplete (abbreviated ?) was a tag designated for all routes who did not fall into either the internal or external categories.

Each of the origin tags are assigned a value for use in transmitting the attribute to other BGP speakers. The values are 0 for internal origin (I), 1 for external origin (E), and 2 for unknown (incomplete) origin (?). A lower value is better, so routes learned from an IGP are preferred over routes learned from an EGP. EGP routes are better than incomplete routes.

Since all routing information eligible to be injected into BGP on a Juniper Networks router resides in inet.0, all possible routes are seen as internal routes. These internal routes all receive a BGP Origin code of Internal when placed into BGP.

Use of the ORIGIN Attribute



The slide shows the default BGP behavior within the JUNOS software with regard to the Origin code.

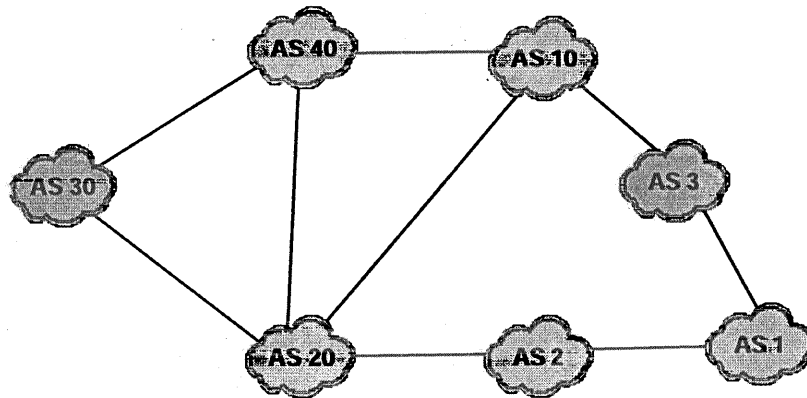
Within the AS shown, there are some static routes of 10.0.0.0/8, 172.16.0.0/16, and 192.168.27.0/24. These routes have been placed into BGP by an export policy. There is a direct route of 192.168.14.0/24 that has been exported into BGP. There is also a route to 172.31.0.0/24 that has been learned from another AS altogether. Finally, there is an IGP learned route of 10.20.0.0/16 in the network. The router does not know whether this is an OSPF route or an IS-IS route, but the appropriate export policy was put in place and the route was placed into BGP.

The point is that it does not matter to the Juniper Networks router when these routes are advertised to another AS using EBGP. The BGP Origin code is the same for all of these routes: statics, direct, IGP, and external to the advertising AS. All are set to Internal.

So on the basis of Origin alone, all of these routes will appear to be equally attractive to the other AS.

ORIGIN Example (I)

- Using the defaults, how does AS 40 reach AS 1?
- Using the defaults, how do the other remote networks reach AS 1?



Copyright © 2001, Juniper Networks, Inc.

In the collection of AS networks on the slide, for all routes originated by AS1, the BGP origin code has been left at its default setting of Internal. These routes have been sent to each of the Autonomous Systems (AS) networks on the slide.

How will AS40 send traffic to AS1? Through AS30, or AS20, or AS10? Assume that the Local Preference BGP attribute has been left as the default, and that the AS Path BGP attribute accurately reflects the actual path for the route.

Routers in AS40 see the AS1 routes advertised from both AS20 and AS10. Since the Local_Pref values and AS_Path lengths are the same for both sets of routes, the Origin code is examined.

For both sets of routes, the Origin code is the same as well. Some other method will need to be used to determine which path the AS40 routers use. That determination is outside the scope of this slide. The important thing here is that AS1 has no way to control how the AS40 routers reach the AS1 networks, based on the default value of the Origin code alone.

Note that routers in AS30 will use AS20 to reach the networks in AS1 due to a shorter AS_Path length through AS20.

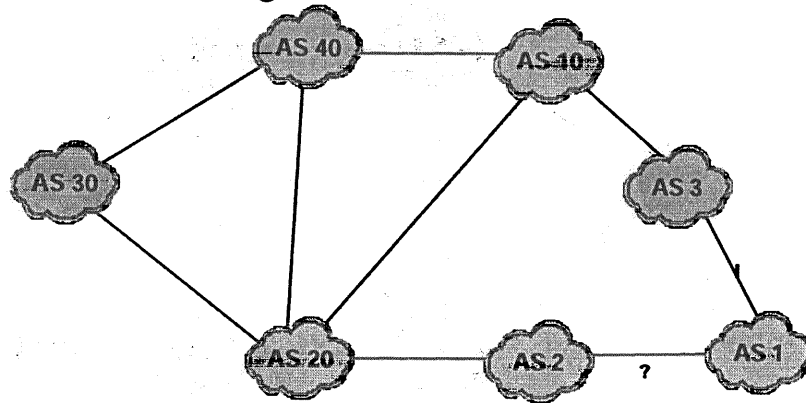
Also, routers in AS20 will use AS2 to reach the networks in AS1 due to a shorter AS_Path length.

Finally, routers in AS10 will use AS3 to reach the networks in AS1 due to a shorter AS_Path length.

Why should AS40 be the only AS that is confused about the "best" way to reach AS1?

ORIGIN Example (II)

- AS 1 sends origin "?" To AS 2. How does AS 40 reach AS 1 now?
- How do the other remote networks reach AS 1 after this attribute change?



Copyright © 2001, Juniper Networks, Inc.

On the previous slide, only AS40 had multiple path options after examining the AS_Path length, since both AS10 and AS20 are the same "distance" away from AS1. At this point, the value of any other tiebreakers that AS40 might use to pick one of those two paths through AS10 or AS20 is unknown to AS1.

But perhaps AS1 has now decided that traffic sent to it from AS40 should use AS10 rather than AS20 (for reasons of economics, politics, or even something else).

By using a routing policy, AS1 has altered the BGP Origin code to Incomplete (?) on all routes advertised to AS2. Consider the effect of this policy on AS40.

Routers in AS40 still see the AS1 routes advertised from both AS20 and AS10. Since the Local-Pref and AS_Path lengths are the same for both sets of routes, the Origin code is examined. The routes received by AS40 from AS10 have an Origin of Internal (0) while the routes received from AS20 have an Origin of Incomplete (2). Internal is better (lower) than Incomplete, so the routes from AS10 are used to reach the networks in AS1. By altering the Origin code, AS1 now has a way to influence the routing decisions in AS40.

Notice that this involved making the path through AS20 look "worse" than the route through AS10, and not making the AS10 path look "better" than the Path through AS20. This reflects the negative bias of Origin adjustments.

Note that routers in AS30 will still use AS20 to reach the networks in AS1 due to a shorter AS_Path length. Routers in AS20 will still use AS2 to reach the networks in AS1 due to a shorter AS_Path length. Routers in AS10 will still use AS3 to reach the networks in AS1 due to a shorter AS_Path length.

Notice that all of the other AS networks on the slide (besides AS40) still use the AS_Path length for route selection. The Origin code is truly only effective when the AS_Path lengths are equal.

Changing the Origin Code

Find IGP Origin codes and change these Origin codes to INCOMPLETE

```
[edit policy-options]
policy-statement change-all-igps {
  term igp-to-incomplete {
    from {
      protocol bgp;
      origin igp;
    }
    then origin incomplete;
  }
}

[edit protocols]
bgp {
  export change-all-igps;
}
```

Copyright © 2001, Juniper Networks, Inc.

The slide shows an example of a BGP export policy that changes the Origin code. This example finds all BGP routes and changes the Origin code from IGP (0) to Incomplete (2).

The match criteria for the policy are all active BGP routes that currently have an Origin code of Internal. The action for the policy is to change the Origin code to Incomplete. Once applied as a BGP export policy and committed, this policy will start altering the origin code for all BGP routes advertised to all BGP peers.

The JUNOS software has specific keywords to represent the different BGP Origin codes. They are:

- **igp** = Internal (value 0)
- **egp** = External (value 1)
- **incomplete** = Incomplete (value 2)

BGP MED Attribute

Where we are going...

- Multiexit Discriminator
- Simple MED Example
- More Complex Use of MED
- Path Selection and MEDs
- Controlling the MED with Policy
- Coordinating MED and IGP Metrics
- MEDs and Aggregates

Copyright © 2001, Juniper Networks, Inc.

Subsequent slides will explore how the BGP MED attribute is used, how to configure a policy to alter the attribute value, how to associate the value with IGP metrics, and how route aggregation affects the attribute value.

Mutliexit Discriminator (MED)

- An optional, non-transitive attribute (Code Type 4), it is never passed through one AS to another AS
- MED can be used by a neighboring AS to prefer one of several paths to the local AS
- Informs neighboring AS which of the ingress paths should be used to reach the local AS in an attempt to influence inbound traffic
- Can perform some primitive load balancing
- MED values are often translated from IGP metric
- Other AS networks can always preempt MED via other BGP attributes

Copyright © 2001, Juniper Networks, Inc.

The BGP Multi-Exit Discriminator (MED) attribute is an optional, non-transitive attribute of BGP. This means that a BGP implementation does not have to understand or use MEDs at all, and that a MED is not sent through one AS and on to another AS. In other words, MEDs are only exchanged between pairs of directly connected ASs. So by default, MED values are transmitted along with BGP routes within the AS where the MED first originated and to all neighboring Autonomous Systems. The MED travels no further without some intervention by means of a policy or alternate configuration.

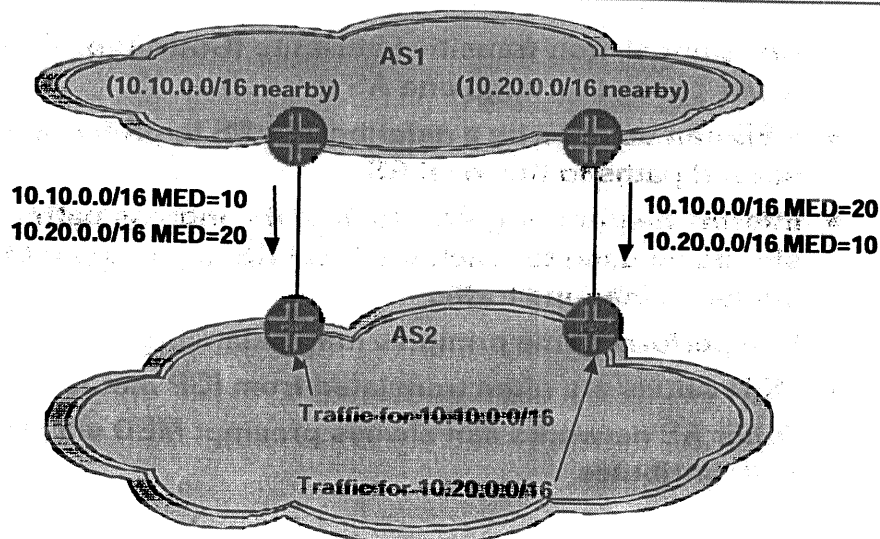
The MED is a type of routing metric assigned to BGP routes. The function of the MED is to assist a neighboring AS to pick which of multiple links connecting the remote AS to the local AS (ingress paths) to use for traffic to a particular route (prefix). So MEDs are an attempt by the local AS to influence the routing decisions in the remote AS for traffic *inbound* to the local AS, just as Origin. And just like Origin, MEDs are a negative bias mechanism to make some paths look worse than others.

MED values can be used to perform some form of primitive load balancing between ASs with multiple links between them. But the use of MEDs for load balancing is neither efficient nor particularly effective compared to more sophisticated mechanisms available.

MED values can be set from multiple locations including administrator configuration or IGP metrics. However, it is very common to take MED values from the metric values in the IGP.

In spite of the best efforts on the part of a local AS to manipulate MEDs to influence inbound traffic flows to the local AS, other ASs can always preempt, or even ignore, the MED. This is not only because the MED is an optional BGP attribute, but also because there are several other BGP attributes more important than MED in the BGP route selection process. For example, an altered Local_Pref attribute will always override the MED.

Simple MED Example



Copyright © 2001, Juniper Networks, Inc.

The slide shows a very basic example in the use of the BGP MED attribute to influence inter-AS traffic flows.

AS1 has assigned its IP address spaces so that it can summarize its network into two major segments. Furthermore, AS1 is relatively cleanly divided into networks that are near the leftmost router (10.10.0.0/16 networks are nearby) and networks that are near the rightmost router (10.20.0.0/16 networks are nearby). Perhaps the split is between Eastern and Western operations, but there are many other alternatives.

AS1 has two EBGPs sessions to AS2 and will be advertising both the 10.10.0.0/16 and the 10.20.0.0/16 networks to AS2 on each EBGPs session, as shown on the slide.

Naturally, AS1 would like AS2 to return traffic to the closest point in the AS1 network so that "timely" packet delivery and low latency can be achieved. Ordinarily, AS1 would have no real way to convey this desire to AS2, and AS2 would simply send traffic to AS1 over whichever router AS2 decided to use based on its own routing policies.

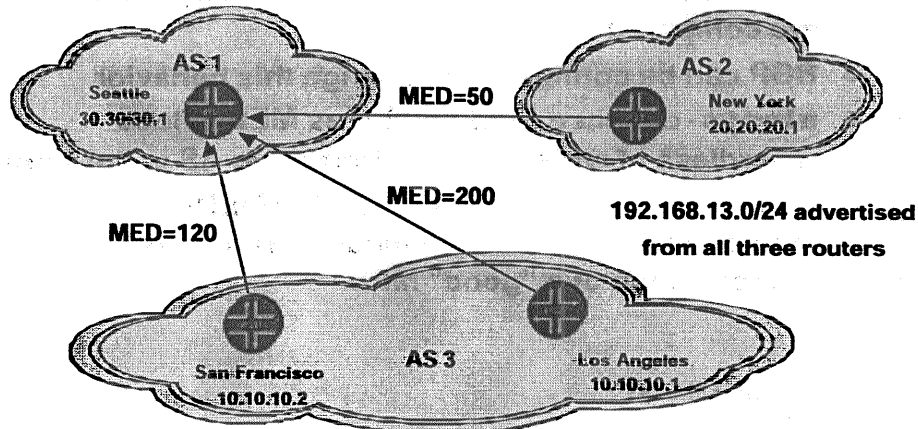
But MED offers a way for AS1 to try and influence the routing policy on AS2 for traffic sent to AS1. To try and accomplish this "closest point" goal, AS1 has altered the MED values on the routes that it advertises to AS2 with a BGP export routing policy.

Both of the networks in AS1 are advertised across both of the links for redundancy. All things being equal, the routers in AS2 will still see multiple network paths to the routes in AS1 as the AS1 routes are passed along throughout AS2. But the AS2 routers will use the MED values of 10 and 20 (10 being preferred) to choose the BGP path to install in their local routing tables.

So within AS2, traffic to 10.10.0.0/16 networks flows to the leftmost router and out to AS1, while traffic to 10.20.0.0/16 networks flows to the rightmost router and out to AS1. AS1 has influenced AS2, and at the same time achieved a primitive type of load balancing.

More Complex Use of MED

Both AS 2 and AS 3 want to influence AS 1 traffic



Choice of SF, LA, or NY to reach 192.168.13.0 is up to AS 1

Chances are that SF will be picked. Why?

To use NY, AS 1 should use **always-compare-med** path selection

Copyright © 2001, Juniper Networks, Inc.

The use of the MED attribute is straightforward when adjacent pairs of Autonomous Systems (ASs) are considered. The use of the MED attribute becomes a little more complicated when more than one AS is involved.

Consider the AS networks on the slide. Both AS2 and AS3 are advertising the 192.168.13.0/24 route to AS1 and want to influence the way that AS1 sends traffic to 192.168.13.0/24. All three of the advertisements have identical Local Preferences, AS_Path lengths and BGP Origin codes. The other IP addresses on the slide represent the router IDs of the three routers in New York, San Francisco, and Los Angeles.

The MED value from the AS2 router is the lowest among the three advertisements. Yet when the router in AS1 chooses the BGP path to use in its local routing table, the AS1 router most likely chooses San Francisco in AS3. Why should this be?

The problem in this scenario is that the default evaluation of the MED attribute only happens when two route advertisements come from the *same* neighboring AS. In this scenario, only two of the three advertisements come from the same AS: those from Los Angeles and San Francisco. Between those two advertisements, the route from San Francisco is the best due to its lower MED.

The route from New York in AS2 can not be compared to the two routes from AS3 since it is from a different AS. At this point, AS1 is left with the San Francisco and the New York routes. The San Francisco route will most likely be selected as the active path because this router has a lower Router ID than New York.

However, the routers in AS1 can compare the MED values for all three of these routes with the use of the **always-compare-med** configuration statement. With this configuration, the path to 192.168.13.0/24 through New York should be chosen based on the lowest MED.

Path Selection and MEDs

- By default, only MEDs from the same neighboring AS are compared
- BGP can be configured to change this behavior
- **always-compare-med** compares MED values regardless of whether the neighboring AS is the same
- Caution is needed when comparing MEDs from more than one AS since every network has a different interpretation of a "good" MED

```
[edit protocols]
bgp {
    path-selection always-compare-med;
}
```

Copyright © 2001, Juniper Networks, Inc.

By default, only the MED values from the same neighboring AS are compared to select a BGP path.

The **always-compare-med** configuration statement allows a network administrator to override the default BGP behavior for MED evaluation.

When configured, **ALL** routes that have the shortest AS_Path length will be compared to each other to determine the route with the smallest MED value, not just routes from the same AS. The route with the lowest MED value will then be selected as the active BGP path regardless of the AS the route came from. The lowest MED value is selected as long as other path selection values for the route, such as Local Preference, are the same.

Caution is needed when comparing MEDs from different ASs. There is some inherent danger when using the **always-compare-med** configuration option to compare MEDs from more than one AS. This is due to the fact that every Autonomous System in the Internet can set their own "good" and "bad" values for MEDs.

One AS may consider a MED of 50 as the best, while another AS might consider a MED of 5 to be good. To complicate matters further, some AS networks may not set the MED value at all (MEDs are optional), which essentially sets the MED value to 0.

Controlling the MED With Policy

- The action of **metric** corresponds to the MED value
- Can be set to a number or added to or subtracted from

```
[edit policy-options]
policy-statement change-the-MED {
  term set-the-med {
    from route-filter 172.31.25.0/24 exact;
    then metric 50;
  }
  term add-to-med {
    from route-filter 192.168.32.0/20 orlonger;
    then metric add 50;
  }
  term subtract-from-med {
    from route-filter 10.124.0.0/16 upto /24;
    then metric subtract 50;
  }
}
```

Copyright © 2001, Juniper Networks, Inc.

The JUNOS software routing policy framework can use the BGP MED attribute as both a match condition and as an action. The **metric** statement applied to BGP is used to indicate the MED value. That is, a policy can match on a certain MED value, or set the MED value to a certain value as an action.

Moreover, the MED value can be set not only to a specific value, but manipulated mathematically ("add 100 to MED" or "subtract 50 from MED").

The example on the slide shows the MED attribute being modified for certain routes. As mentioned, the **metric** statement represents the MED value. Within a policy, the metric can be set to a value, it can have its current value added to, or it can have its current value subtracted from.

In the policy shown:

- The 172.31.25.0/24 route will have its MED set to 50.
- All routes within the 192.168.32.0/20 network will have their current MED value increased by 50.
- All routes within the 10.124.0.0/16 network with a mask shorter than /24 will have their current MED value decreased by 50. Should the current value be less than 50, the result of this policy action will be a MED value of 0.

Coordinating MED and IGP Metrics

- BGP can set the MED value on route announcements based on the IGP metric to the peer the route was received from
- Use the **metric-out** command with a group or neighbor
 - Can be set to a specific value
 - Can be set to the current IGP metric
 - Can be set to the minimum IGP metric ever learned
 - Can add to or subtract from the IGP metric

```
[edit protocols bgp]
group as-100-peers {
  type external;
  peer-as 100;
  neighbor 192.168.2.2 metric-out 10;
  neighbor 192.168.3.3 metric-out igp;
  neighbor 192.168.4.4 metric-out minimum-igp;
  neighbor 192.168.5.5 metric-out igp 5;
}
```

Copyright © 2001, Juniper Networks, Inc.

MED values do not have to be arbitrary. In many cases, the MED values are coordinated with the metric values used by an IGP. So BGP can set the MED value on routes that are advertised based on the IGP metric leading to the BGP peer that the route was received from.

In addition to using a policy to alter the MED values, the JUNOS software also allows an administrator to configure a MED value for all BGP routes to an individual peer, peer-group, or all peers. The configuration statement used the keyword **metric-out** and has several optional parameters.

To set the MED to a static numeric value, use the **metric-out <value>** statement. This option, setting the MED to 10, can be seen for the 192.168.2.2 peer above.

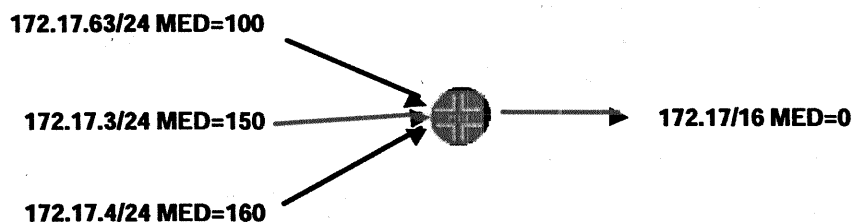
The MED can be set to match the internal IGP metric to reach the IBGP that advertised the route. The **metric-out igp** command is used to do this. As the IGP metric to this peer changes, the MED value associated with these routes will also change. This option can be seen on the 192.168.3.3 peer above.

The MED will then change every time the IGP metric to the peer changes. If this is undesirable, the MED can be associated to the lowest possible IGP metric ever known for the specific IBGP peer. The MED may decrease if the IGP metric lowers, but a network failure that increases the IGP cost will not increase the MED value. The **metric-out minimum-igp** command is used to do this. This option can be seen on the 192.168.4.4 peer above.

Lastly, the addition and subtraction functions from the policy framework can be used in conjunction with both the **igp** and **minimum-igp** options. To alter the MED in this fashion, use the **metric-out igp <offset>** command. This option can be seen on the 192.168.5.5 peer above, which adds 5 to the MED based on the IGP metric. To subtract 5, use -5.

MEDs and Aggregates

When routing aggregation occurs, the MEDs associated with the more granular routes are no longer available



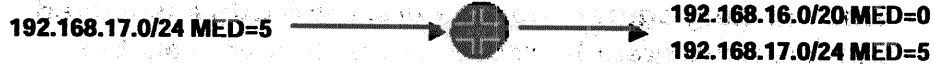
Copyright © 2001, Juniper Networks, Inc.

MED values have, by default, a limited scope of operation. For example, MEDs are not propagated through one AS and on to another AS by default (non-transitive). This limiting concept also applies when route aggregation is examined.

When a new aggregate route is created, any MED values currently assigned to any of the contributing routes remain only with those routes. The aggregate route has no MED assigned to it, which is a MED of 0. While at first this might seem to be a contradiction, since 0 *is* a MED, the aggregate route has no method for determining which **ONE** MED value to choose, so MED = 0 is used.

There is really no other alternative. Since a BGP route can have only one MED value, the aggregate would need to choose what that value should be. Should the aggregate take the worst MED value from the contributors and be conservative? Should the aggregate take the best MED value to not "penalize" that contributing route? Should the aggregate average the contributors MED values together? None of these would adequately represent all of the contributing information, so the aggregate route takes the ultimate conservative approach: MED = 0, or no MED at all.

More on MEDs and Aggregates



```
user@host> show route protocol bgp
```

```
inet.0: 31 destinations, 31 routes (31 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both
```

```
192.168.17.0/24    *[BGP/170] 00:20:38, MED 5, localpref 100, from 192.168.48.1
                  AS path: (65001) 1 I
                  > to 10.40.40.1 via so-0/0/0.0
```

```
user@host> show route advertising-protocol bgp 10.222.11.1
```

```
inet.0: 31 destinations, 31 routes (31 active, 0 holddown, 0 hidden)
Prefix           Nexthop      MED      Lc1pref AS path
192.168.16.0/20   Self         0         100 (65001) 1 I
192.168.17.0/24   192.168.0.1  5         100 (65001) 1 I
```

Copyright © 2001, Juniper Networks, Inc.

The slide shows the default MED behavior regarding aggregate routes. The router receives the 192.168.17.0/24 route with a MED of 5. This is seen from the output of the **show route protocol bgp** command as shown on the slide.

The router has a locally defined aggregate route which is being injected into BGP by means of a policy (the policy not shown on the slide).

By examining the **show route advertising-protocol bgp** output, we can see that both the aggregate route and the more specific received BGP route are being advertised. The 192.168.17.0/24 route maintains its MED of 5, while the aggregate route of 192.168.16.0/20 has no MED value (MED = 0) assigned. Of course, the MED on the aggregate can always be changed with a routing policy, but this lack of an aggregate MED value is the default behavior.

So it is ironic that MEDs, which are most useful between AS pairs, are useless by default on aggregates, which are exactly the types of routes we want to send between AS pairs!

Review Questions

- Put the following BGP route selection attributes in order from highest to lowest priority:
 - Origin
 - Local Preference
 - MED
 - AS Path
- Why do Juniper Networks routers set all Origins to Internal regardless of actual source of the route?
- Why is the default not to compare MEDs that come from two different ASs?
- What is the value of the MED for an aggregate route?

Copyright © 2001, Juniper Networks, Inc.

During this module we discussed:

- The use of the BGP Origin attribute.
- How to configure a routing policy to alter the BGP Origin attribute.
- The role played by the Multi-Exit Discriminator (MED) BGP attribute in BGP operation.
- The operation of the MED attribute and how it influenced BGP route selection.
- How to configure a routing policy to change the MED value and the effect that has on AS to AS traffic flows.



Juniper Networks Advanced Policy

Release 5.0, Revision 1

Module 7: BGP Attributes: AS Path

Copyright © 2001, Juniper Networks, Inc.

Objectives

- **Describe the different methods for altering the AS_Path attribute**
- **Explain the operators and use of regular expressions for AS Paths in a routing policy**
- **Configure a routing policy using regular expressions to change AS-Paths and alter traffic flows**
- **Explain the role played by the Null AS_Path in routing policy**

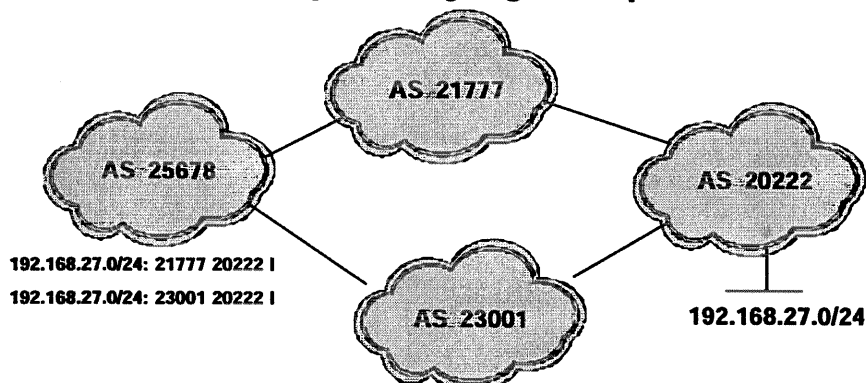
Copyright © 2001, Juniper Networks, Inc.

This module discusses:

- The different methods that can be used to change the BGP AS-Path attribute, and why this is useful.
- The regular expressions and their operators that are used in a routing policy to find and alter AS Paths.
- Configuring a routing policy that uses these regular expressions to change the BGP AS-Path attribute and so alter the default traffic flows between AS networks.
- The special role played by the Null AS_Path in a routing policy.

AS Paths

- BGP AS Path is the route to a given destination
- Consists of a list of the AS numbers of all routers (path) a packet must go through
- The path can be parsed by regular expressions



Copyright © 2001, Juniper Networks, Inc.

The BGP attribute of AS_Path contains the data that all BGP routers use to find a route back to a route's source. As a route passes through each AS, the BGP routers add path information in the form of the AS number to this attribute. This AS number is *prepended*, or added at the beginning of the AS_Path attribute. So each AS basically counts as a single "hop" from the perspective of the BGP AS_Path attribute.

By default, the total amount of information in the AS_Path attribute gives each router the entire list of AS numbers (the path) that the user data must pass through to reach a particular destination. It is important to note that when the AS_Path is "changed" by a routing policy, the AS Path information in the BGP AS_Path attribute can be added to, but a existing individual AS number should never actually be changed or deleted.

The AS_Path attribute can be used as a single object, or the string of AS numbers can be parsed by a regular expression to find specific AS information.

In the slide, AS 20222 is originating the route 192.168.27.0/24. AS 20222 advertises that route to both AS 21777 and AS 23001. In turn, each of those AS systems advertises the same route to AS 25678. The BGP routers in AS 25678 see this route as having two possible paths through the Internet, as shown on the slide. They can then use the AS_Path attribute to determine the shortest path back to the route's source. (In this simple example, the paths are of equal length, but of course this will not always be the case.)

Modifying AS Path Information

Where we are going...

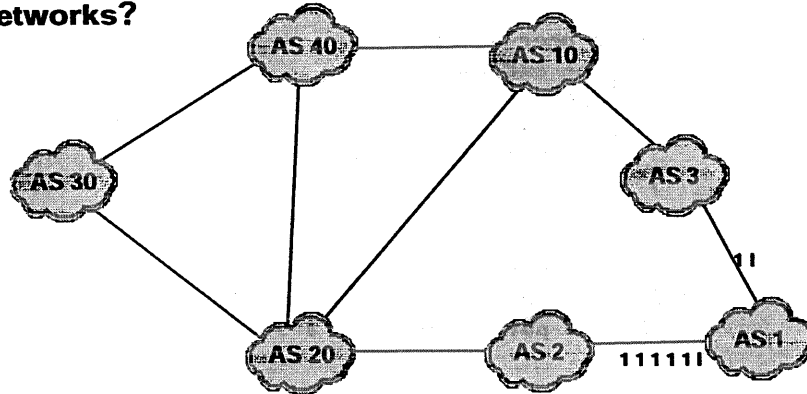
- **AS Path Prepend & Example**
- **Remove-Private**
- **Local-AS**

Copyright © 2001, Juniper Networks, Inc.

Subsequent slides will explore how modify the default behavior of the BGP AS_Path attribute including prepending, the remove-private option, and the local-as configuration knob.

Modifying AS Path: Prepend

- Manipulating the AS_Path attribute is a major way to favor or disfavor BGP routes
- AS 1 prepends its AS number 3 times to AS 2. How does AS 40 reach AS 1? How about the other remote networks?



Copyright © 2001, Juniper Networks, Inc.

If the AS_Path attribute is changed before the route is re-advertised to other BGP routers, this can make a route through the local AS look *less* attractive to another AS. Note that it should not really be possible to make the route more attractive by shortening the AS_Path. But the path can be lengthened to make the AS_Path attribute another type of negative bias path selection mechanism in the sense that "un-lengthened" paths will look more attractive for other AS networks to use than artificially lengthened AS Paths.

The only standard way to alter the AS_Path attribute is to add information to it by prepending. BGP routers, according to the RFC, are **NOT** allowed to remove or change information from or in the AS_Path attribute. But a routing policy can be used to artificially extend (by prepending) AS information onto an existing AS Path. This type of a policy is often to attempt to influence traffic coming *into* an AS from another AS. In this respect, AS_Path is similar in function to MED.

In the slide, AS 1 is announcing its routes to both AS 2 and AS 3. Using a routing policy, AS 1 is prepending its own AS number four times (shown as 1 1 1 1 on the slide) onto all route announcements to AS 2. This action will cause the following:

- AS 2 will use AS 20 to forward packets to AS 1
- AS 10 will use AS 3 to forward packets to AS 1
- AS 20 will use AS 10 to forward packets to AS 1
- AS 30 will use AS 40 to forward packets to AS 1
- AS 40 will use AS 10 to forward packets to AS 1

Note that this behavior on the part of AS 2 (using AS 20 instead of sending directly to AS 1) is unexpected and would not occur without the routing policy. This behavior is extended to AS 20 as well, since AS 2 cannot shorten the AS_Path advertised by AS 1 even if AS 2 would like to.

AS Path Prepend Example

```
[edit routing-options]
autonomous-system 1;

[edit protocols]
bgp {
  group peer-AS2 {
    type external;
    export longer-as-path;
    peer-as 2;
    neighbor 10.10.10.2;
  }
}

[edit policy-options]
policy-statement longer-as-path {
  then as-path-prepend "1 1 1 1";
}
```

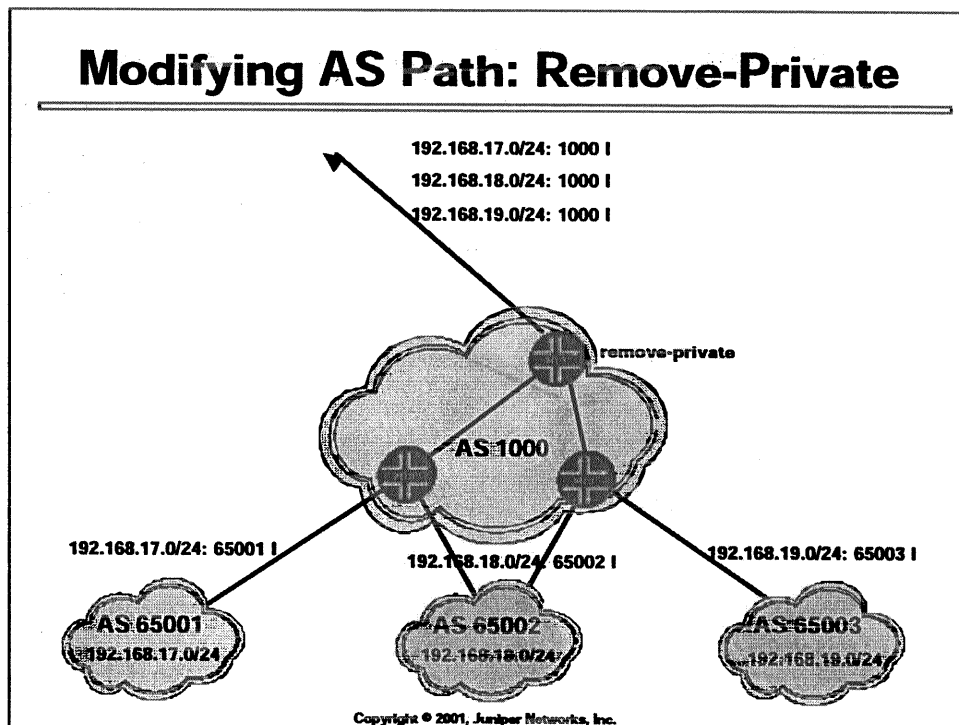
Copyright © 2001, Juniper Networks, Inc.

This slide shows the routing policy used by AS 1 in the previous example.

A policy called **longer-as-path** has been created on the AS 1 router. Since there is no **from** statement, all candidate routes will match the policy. The action taken on all of the matched routes will be to add AS 1 to each route four times. This is done with the **as-path-prepend** statement.

This routing policy then is applied for routes exported by BGP to the external BGP peer in AS 2.

Modifying AS Path: Remove-Private



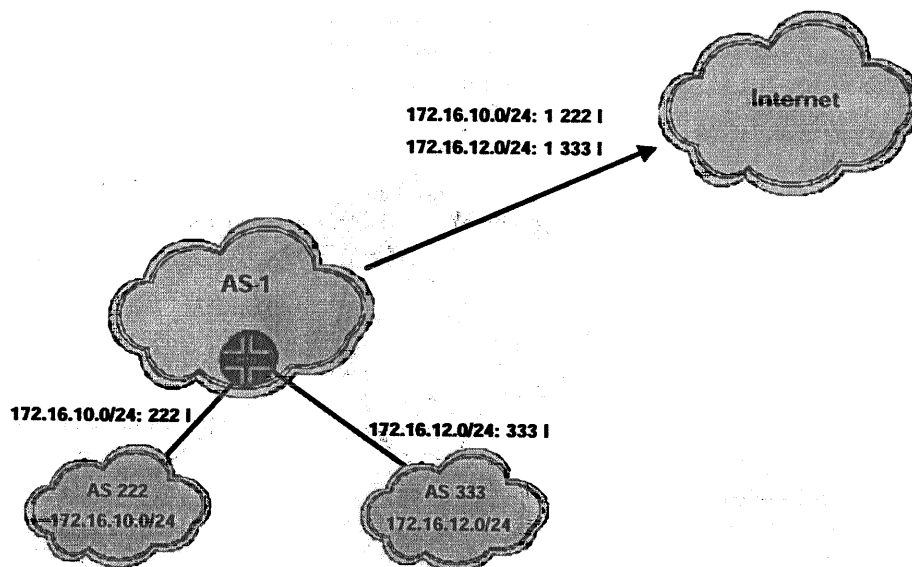
In spite of the wording in the BGP RFC, many vendors have included configuration options in their BGP implementations to actually remove information from the AS_Path, which is technically not allowed. This removal, however, only operates on specific information in the AS_Path attribute, and does not apply to making arbitrary changes to the actual AS Path. Typically, the information removed was placed there by the AS itself or by other routers within the administrative control of the AS. So it is not a question of one AS trampling on the path information another AS has put into the AS Path.

One example of this type of configuration option is the **remove-private** configuration statement. This keyword allows an ISP to remove private AS numbers from paths received from BGP customers when those customers are using private AS numbers. Since the customers are effectively within the administrative scope of the ISP, the provider is allowed to remove the private AS numbers from the path.

In the slide, AS 1000 has three different customers connected via BGP. The customers are using AS 65001, AS 65002, and AS 65003 for the BGP peer communications. Within AS 1000, each of the BGP routers see the private AS numbers within the path.

The **remove-private** option is enabled on the edge router in AS 1000 that faces the Internet or other EBP peers. As the routes from the customer AS systems are advertised out of AS 1000, the private AS numbers are removed from the AS_Path attribute. In this case, all customer networks are seen to have originated within AS 1000.

Modifying AS Path: Local-AS (I)



Copyright © 2001, Juniper Networks, Inc.

Another option for removing AS_Path information is the **local-as** configuration statement. The purpose of the **local-as** keyword is to aid an ISP in migrating BGP customers to a new AS number.

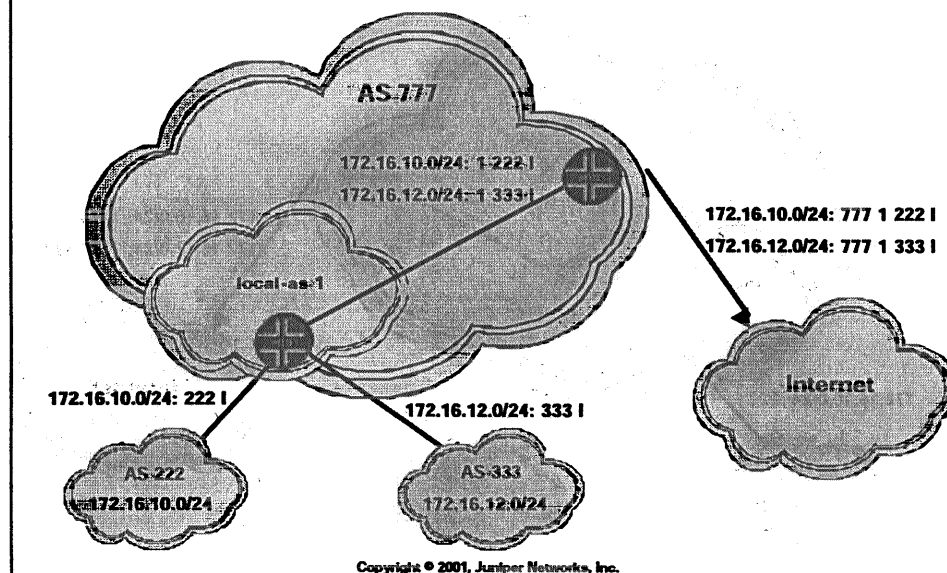
Here is an example covering when using **local-as** is helpful.

Consider the normal BGP AS_Path operation first. AS 1 has two customers that it is providing service for: AS 222 and AS 333.

As AS 1 announces those routes from AS 222 and AS 333 on to the Internet, AS 1 injects its own AS number (1) into the AS_Path as the BGP RFC expects.

So far, there is nothing unusual about the AS_Path operation.

Modifying AS Path: Local-AS (II)



Next, consider what happens if AS 1 then merges with AS 777. This situation is shown on the slide.

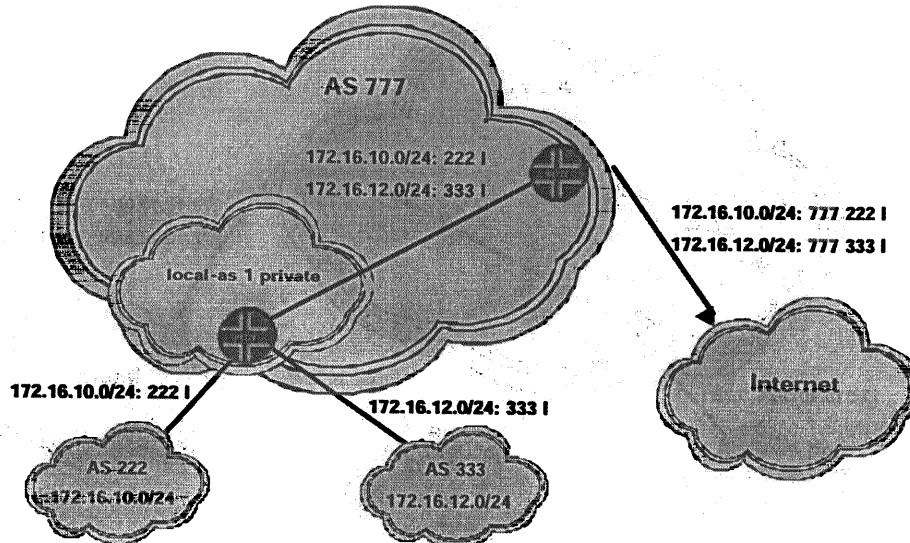
Suppose the resulting merged organization decides to use AS 777 as the official AS to represent both AS numbers on the Internet. To ease in the migration of the customer BGP configurations from AS 1 to AS 777, the edge routers can use the **local-as 1** configuration option.

The effect of this option is that the customer routes within AS 777 see **both** AS 1 and the customer AS numbers (AS 222 and AS 333). As AS 777 advertises those routes to the Internet, it prepends its own value of AS 777 onto each of the routes.

The point is that even though AS 1 has been merged into AS 777, AS 1 still shows up on the paths sent to the Internet.

So far, this is still just prepending AS numbers to the BGP AS_Path attribute. How does **local-as** remove AS Path information?

Modifying AS Path: Local-AS (III)



Copyright © 2001, Juniper Networks, Inc.

There is an optional parameter to be used with the **local-as** configuration statement that actually does remove AS Path information from the BGP AS_Path attribute. This optional parameter restricts knowledge of AS 1 to the edge router connected to the customer (AS 222 and AS 333) only. This situation is shown on the slide.

In the slide, the edge router in the formerly intact AS 1 is configured with the option of **local-as 1 private**. As the customer routes are advertised into AS 777 by the edge router, AS 1 information is removed from the AS_Path attribute, as shown on the slide. At this point the AS 777 routers, as well as the Internet, have no knowledge of AS 1.

The **local-as 1 private** statement has now indeed removed AS_Path information. Again, this applies to a type of special case and should not be used to arbitrarily attempt to change AS Path information received from another AS.

AS Path Regular Expressions

Where we are going...

- Regular Expressions and BGP Routes
- Regular Expressions
- Regular Expression Terms
- Regular Expression Operators
- Regular Expression Examples
- Regular Expression Formation
- Regular Expression Examples
- Test Your Knowledge

Copyright © 2001, Juniper Networks, Inc.

Subsequent slides will explore how the JUNOS software forms AS Path regular expressions, how those expressions are used in a policy, and provides of examples of that usage.

Regular Expressions and BGP Routes

- Often, BGP policy relies on finding prefixes based on their AS Path information.
 - Used to enforce administrative policy
 - Sometimes easier than looking for specific prefixes
- Common policy requirements:
 - Find all routes that originated in AS 1
 - Find all routes that have transited AS 100
 - Find all routes that came from AS 1000
 - Find all the routes that originated in my own AS
- AS Path Regular Expressions allow the proper prefixes to be selected

Copyright © 2001, Juniper Networks, Inc.

Many times, administrative BGP routing policies require that route announcements or prefixes be found and acted on based on the information contained within the AS_Path attribute. This might be required to enforce some administrative policy regarding other AS networks. Sometimes it is just easier to find routes based on their AS_Path than by looking for many specific prefixes and routes individually.

Some examples of the types of information that need to be found in the AS_Path might be:

- All routes that originated from a specific AS
- All routes that have transited a specific AS
- All routes that were announced by a specific neighbor AS
- All of the routes that have originated from within the local AS

This type of information can be found quite easily in the AS_Path attribute through the use of *regular expressions*.

Regular Expressions

- Regular expressions are a powerful *pattern matching* engine
- Works not only on fixed strings (as do wildcards), but on variable patterns of text
- It is the combination of text and special operators that make up a *regular expression*
- Regular expressions allow for things to be found in context, not as isolated instances

Copyright © 2001, Juniper Networks, Inc.

A regular expression (often seen as Regex or RE) is a powerful *pattern matching* tool that can be applied in a routing policy to act on AS_Path strings. This pattern matching engine can find specific strings of text or textual patterns.

When used in a routing policy, regular expressions work not only on fixed strings like wildcard operators such as *, but regular expressions act on variable patterns of text. This is done through the combination of basic text patterns and special operators.

This combination of basic text patterns and special operators are what make up the regular expression itself.

Regular expressions allow information in a string to be found within a specific context, not just in isolated instances. The use of regular expressions in conjunction with the BGP AS_Path attribute can be used to match routes within a policy.

Regular Expression Terms

- Regular expressions take form *term* <operator>
- *Terms* are mandatory, and identify the AS number:
 - Can be a single AS number
 - "1024"
 - Can be a complete AS path
 - "1024 2685 3957"
 - Can be a wildcard "." character which represents a single AS
 - "1024 . 3957"
- Each AS number (not a character) represents a single "entity" to the regular expression parser

Copyright © 2001, Juniper Networks, Inc.

Regular expressions have two main components to them – the term and the operator. These take the form **term<operator>**.

The regular expression *term* is the core matching component to be found by the pattern matching engine. The *term* is a mandatory object and each regular expression must have at least one term. Terms identify the AS number. This AS could be a single number (1024), a complete AS Path (1024 2685 3957), or a wildcard character (the dot or .) representing any single AS number (1024 . 3957)

Within the JUNOS software AS_Path regular expression syntax, the term is a complete AS value. The wildcard character of the "dot" (.) can be used to represent a single AS number as well. This means that an AS number of 1024 (for example) is seen by the JUNOS software not as the four character text string of 1, 0, 2, 4, but as the single "entity" of 1024. To the JUNOS software, AS numbers are not sequences of characters.

Regular Expression Operators

- Regular expressions take form *term* <operator>
- Operator is an optional pattern matching character that applies to a single term:
 - Operators immediately follow the term referenced
 - "1024? 2685"
 - The pipe (|) operator is used between terms
 - "1024 | 2685"
 - The dash (-) operator is used between terms
 - "1024 - 2685"
- One or more term-operator pairs can appear in an AS Path Regular Expression

Copyright © 2001, Juniper Networks, Inc.

All regular expressions take the form **term<operator>**. Use of the *term* is mandatory. However, the regular expression <operator> is an optional component of the pattern matching engine. An operator can be associated with a single regular expression term. If used, the operator appears immediately after the term it is operating on.

There are two special regular expression operators that can appear between regular expression terms. Those special characters are the pipe (|) and the dash (-). The pipe (|) operator is used between terms to indicate OR ("1024 OR 2685" is "1024 | 2685"). The dash (-) operator is used between terms to indicate range ("1024 through 2685" is "1024-2685").

An individual regular expression can contain multiple term-operator pairs.

AS Path Regex Operators

{m,n}	Match at least <i>m</i> and at most <i>n</i> repetitions of <i>term</i>
{m}	Match exactly <i>m</i> repetitions of <i>term</i>
{m,}	Match <i>m</i> or more repetitions of <i>term</i>
*	Match 0 or more repetitions of <i>term</i> , same as {0,}
+	Match 1 or more repetitions of <i>term</i> , same as {1,}
?	Match 0 or 1 repetitions of <i>term</i> , same as {0,1}
 	Match one of the two <i>terms</i> on either side of the pipe
-	Used to represent a range
(...),0	Used to group <i>terms</i> , or indicate null with no space

Copyright © 2001, Juniper Networks, Inc.

The table on the slide shows a subset of the possible regular expression operators that can be used with routing policies. Some operators are a kind of "shorthand" for their longer equivalents.

Of special note is the use of the parentheses. Typically, the (...) operator is used to group multiple terms together in conjunction with an operator. For example, the regular expression of "(1|2)?" is translated as either AS 1 or AS 2 can be present in the AS Path zero (absent) or at most one time (this is what the ? operator means).

The parenthesis operator also has another special use. When used with no spaces in between, parentheses represent a *Null AS_Path* value. The concept of the Null AS_Path will be covered shortly.

Regular Expression Examples

AS Path pattern to match:	Regex:	Example matches:
Exactly one instance of AS1234	1234	1234
0 or more instances of AS1234	1234*	1234, 1234 1234, etc., or Null AS_Path
0 or 1 instances of AS1234	1234?	1234 Null AS_Path
1 to 4 instances of AS1234	1234{1,4}	1234, 1234 1234, 1234 1234 1234, 1234 1234 1234 1234
1 to 3 instances of AS12 followed by 1 instance of AS34	"12{1,3} 34"	12 34, 12 12 34, 12 12 12 34
Range of AS numbers to match a single AS	"123 - 125"	123 or 124 or 125

Copyright © 2001, Juniper Networks, Inc.

The table on the slide shows some examples of regular expression pattern matching as applied to AS Paths.

The first column shows the AS Path string that the routing policy is trying to match. The second column shows the regular expression that is used to match that pattern. The last column shows examples of values of the BGP AS_Path attribute that the regular expression will match. In some cases the list is not exhaustive, so there are more AS-Paths that will match the pattern.

More Regex Examples

AS Path pattern to match:	Regex:	Example matches:
Second AS <i>must</i> be 56 or 78	". (56 78)"	1234 56, 34 78
Second AS <i>might</i> be 56 or 78	". (56 78)?"	1234, 1234 56, 34 78
All paths from neighbor AS1234	"1234 ."	1234, 1234 5678, 1234 5 6 7 8
1 followed by 2, followed by one or more instances of 3	"1 2 3+"	1 2 3, 1 2 3 3, 1 2 3 3 3, etc.
One or more instances of 1, then 2, then 3	"1+ 2+ 3+"	1 2 3, 1 1 2 3, 1 1 2 2 3, 1 1 2 2 3 3, etc.
Any length path that contains the list 4, 5, 6 in it anywhere	".* 4 5 6 ."	1 2 3 4 5 6, 1 2 3 4 5 6 7 8 9, 4 5 6 7 8 9

Copyright © 2001, Juniper Networks, Inc.

The table on the slide shows more examples of regular expression pattern matching as applied to AS Paths.

As before, the first column shows the AS Path string that the routing policy is trying to match. The second column shows the regular expression that is used to match that pattern. The last column shows examples of values of the BGP AS_Path attribute that the regular expression will match. In some cases the list is not exhaustive, so there are more AS-Paths that will match the pattern.

Regular Expression Formation

- Regular Expressions are defined at the **policy-options** level
 - [edit policy-options]
 - user@host# **set as-path name regular-expression**
- Where:
 - The name identifies the regular expression
 - The regular-expression consists of the pattern to match in **term <operator>** format
- The name can be up to 255 characters long
- To include spaces in the name, enclose the entire name in double quote quotation marks
- The defined AS Path regex can then be used as a policy match condition

Copyright © 2001, Juniper Networks, Inc.

The whole point in this regular expression exercise is that AS_Path regular expressions can be used as the match criteria within a routing policy. This application of the regular expression is similar in concept to the application of a policy. As such, AS_Path regular expressions follow the JUNOS software abstraction concept of first defining the entity and then applying the entity.

Both the definition and application of the AS_Path regular expressions occurs within the **policy-options** configuration hierarchy. The regular expression in proper **term<operator>** format is given a name with the **as-path** statement.

That regular expression name can then be referenced within a policy. The name can be up to 255 characters long and can even include spaces, as long as the name is enclosed in double quotation marks. In practice, this is rarely done.

Once defined, the AS Path regular expression can be applied as a policy match condition.

Regex Example

Accept only routes with the exact AS path of 1234 56 78 9

```
[edit policy-options]
as-path digits-route "1234 56 78 9"
```

```
policy-statement from-digits-route {
  term digits {
    from as-path digits-route;
    then accept;
  }
  term reject-others {
    then reject;
  }
}
```

```
[edit protocols]
bgp {
  import from-digits-route;
}
```

Copyright © 2001, Juniper Networks, Inc.

The example on the slide shows an AS_Path regular expression being used as a policy match condition. The goal is to have a BGP routing policy that accepts only routes with the exact AS Path of 1234 56 78 9.

An **as-path** called **digits-route** has been defined within the double quotation marks. The **as-path** contains only *terms* (no *operators*) and defines an exact AS_Path match of "1234 56 78 9".

The **digits-route** path definition is then used within the **from-digits-route** policy to match routes and accept them. A second term in the policy rejects all other routes.

When this policy is applied as an import BGP policy, only routes matching the AS_Path defined in **digits-route** will be accepted. All other routes seen by BGP will get rejected.

Another Regex Example

Reject any routes with AS 123, 124, or 125 anywhere in the AS path

```
[edit policy-options]
as-path not-a-good-route ".* 123-125 .*"

policy-statement from-not-a-good-route {
  term not-good {
    from as-path not-a-good-route;
    then reject;
  }
}

[edit protocols]
bgp {
  import from-not-a-good-route;
}
```

Copyright © 2001, Juniper Networks, Inc.

The example on the slide also shows an AS_Path regular expression being used as a policy match condition. This time, however, the **as-path** uses both terms and operators.

The goal this time is to reject routes that either have originated in AS 123-125, transited through AS 123-125, or been directly advertised from AS 123-125.

To do this, an **as-path** called **not-a-good-route** has been defined. The **as-path** contains both terms and operators. So **not-a-good-route** defines an AS_Path match as follows:

- The AS Path starts off with any AS value zero or more times, followed by
- any AS value between 123 and 125, followed by
- any AS value zero or more times.

The **not-a-good-route** path definition is then used within the **from-not-a-good-route** policy to match routes and reject them.

When this policy is applied as an import BGP policy, only routes matching an AS_Path defined in **not-a-good-route** will be rejected.

Test Your Knowledge

- Consider this policy statement:

```
policy-statement testing-as-paths {
  term as-paths {
    from as-path testing-1-2-3;
    then accept;
  }
  then reject;
}
```

- Will I accept a route with the path "1024 44 44 2685," given the following as-path statements?

- set as-path testing-1-2-3 ".* 1024"
- set as-path testing-1-2-3 "1024 ."
- set as-path testing-1-2-3 ".* 1024 ."
- set as-path testing-1-2-3 ".* 44{1,2} ."
- set as-path testing-1-2-3 "2685 44{1,3} 1024"
- set as-path testing-1-2-3 "1024 44{1,3} 2685"

Copyright © 2001, Juniper Networks, Inc.

Consider the routing policy shown on the slide. Will this policy, when properly applied, accept a route with the path "1024 44 44 2685" given the as-path statements listed on the slide?

Using the different as-path definitions within the testing-as-paths policy gives the following results:

- ".* 1024" – Starts with any AS zero or more times, followed by 1024. The route will not match the term as-paths. This definition does not allow for AS numbers after 1024. Therefore it will be **rejected** by the final action.
- "1024 ." – Starts with 1024, followed by zero or more AS numbers. The route does match the term as-paths. It will be **accepted**.
- ".* 1024 ." – Starts with any AS zero or more times, followed by 1024, followed by zero or more AS numbers. The route does match the term as-paths. It will be **accepted**.
- ".* 44{1,2} ." – Starts with any AS zero or more times, followed by 44 once or twice, followed by zero or more AS numbers. The route does match the term as-paths. It will be **accepted**.
- "2685 44{1,3} 1024" – Start with AS 2685, followed by 44 one to three times, followed by 1024. The route will not match the term as-paths. Therefore it will be **rejected** by the final action.
- "1024 44{1,3} 2685" – Start with AS 1024, followed by 44 one to three times, followed by 2685. The route does match the term as-paths. It will be **accepted**.

Null AS Path Information

Where we are going...

- Null AS_Path
- Null AS_Path to Stop Transit
- Null AS_Path in Action

Copyright © 2001, Juniper Networks, Inc.

Subsequent slides will explore the concept of the Null AS Path and how to use it in a real world example.

Null AS_Path

- Routes that have originated in your own AS have no AS numbers in the path yet
- To reference the Null AS_Path within a policy, use the "0" (no space) regular expression

192.168.48.0/24
192.168.49.0/24
192.168.50.0/24
192.168.51.0/24

IBGP



```
user@host> show route protocol bgp terse
```

```
inet.0: 42 destinations, 42 routes (42 active, 0 holdown, 0 hidden)
+ = Active Route, - = Last Active, * = Both
```

A Destination	P Prf	Metric 1	Metric 2	Next hop	AS path
* 192.168.48.0/24	B 170	100	5	>10.222.9.1	I
* 192.168.49.0/24	B 170	100	5	>10.222.9.1	I
* 192.168.50.0/24	B 170	100	5	>10.222.9.1	I
* 192.168.51.0/24	B 170	100	10	>10.222.9.1	I

Copyright © 2001, Juniper Networks, Inc.

The concept of the Null AS_Path is quite important for the Internet. Routes that have been originated within a particular AS have yet to prepend the BGP AS_Path attribute. Therefore, there is no information contained in the AS_Path attribute for routes originating within the AS and the AS_Path is assumed to be null (empty).

So how are routes originating within the AS that have been advertised with BGP, to be found with a routing policy using an **as-path** policy? By creating a match condition based on the Null AS_Path.

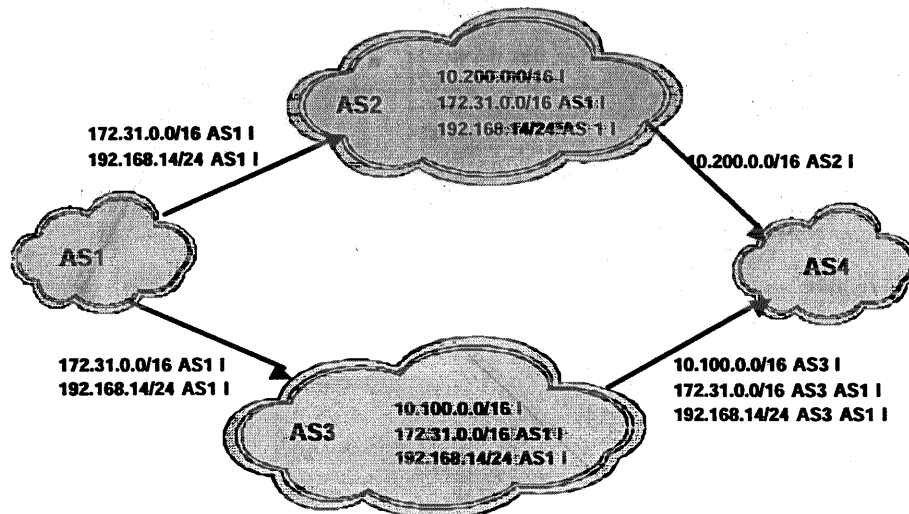
To specify the Null AS_Path using a regular expression, use the parentheses with no space in between them.

In the example, the router is receiving four routes from IBGP. By examining the routing table, we can see that the AS Path column is empty ("I" is for the Origin code).

Therefore, these routes have been sourced within the router's own AS. The Null AS_Path will find these routes.

Null AS_Path to Stop Transit

AS 2 does not want to provide transit service to AS 1



Copyright © 2001, Juniper Networks, Inc.

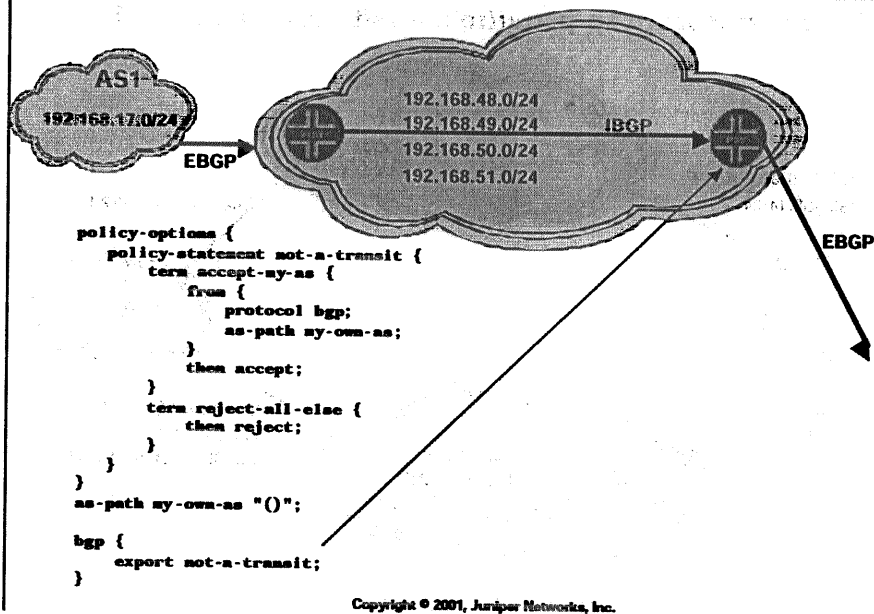
The Null AS_Path can be used to find routes advertised by BGP that originated within the local AS. When would this be helpful? Usually, when one AS does not want to carry transit traffic for another AS.

In the example on the slide, the Null AS_Path will be used to halt BGP transit traffic from AS 1 through AS2.

AS 2 does not want to re-advertise the routes from AS 1 to AS 4. That could lead to the possibility that AS 4 would route traffic for AS 1 through AS 2. However, AS 2 still has to advertise its own routes to AS 4 so that these prefixes are reachable.

AS 2 defines an **as-path** applied as a BGP export policy towards AS 4 that rejects all routes except those with a Null AS_Path.

Null AS_Path in Action (I)

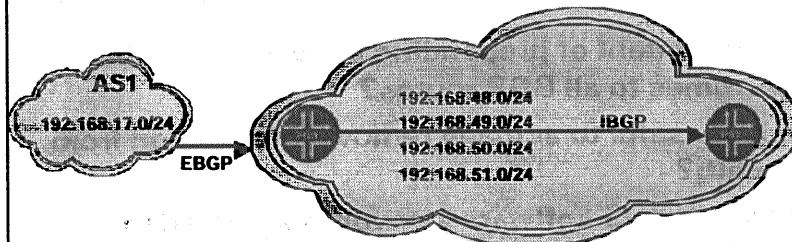


Here is an example of Null AS_Path in action.

The **not-a-transit** policy in the slide is being applied as an export policy towards the EBGP peer.

The policy has a term called **accept-my-as** that matches on BGP routes with an **as-path** regular expression of **()**. All other routes are rejected via the **reject-all-else** term.

Null AS_Path in Action (II)



```
user@host> show route protocol bgp terse
inet.0: 43 destinations, 43 routes (43 active, 0 holddown, 0 hidden)

A Destination      P Prf Metric 1   Metric 2   Next hop      AS path
* 192.168.17.0/24  E 170      100         5 >10.40.40.1  I I
* 192.168.48.0/24  E 170      100         5 >10.40.40.1  I
* 192.168.49.0/24  E 170      100         5 >10.40.40.1  I
* 192.168.50.0/24  E 170      100         5 >10.40.40.1  I
* 192.168.51.0/24  E 170      100        10 >10.40.40.1  I
```

```
user@host> show route advertising-protocol bgp 10.222.11.1
inet.0: 43 destinations, 43 routes (43 active, 0 holddown, 0 hidden)
Prefix      Nexthop      MED      Lclpref AS path
192.168.48.0/24  192.168.48.1      5        100 I
192.168.49.0/24  192.168.48.1      5        100 I
192.168.50.0/24  192.168.48.1      5        100 I
192.168.51.0/24  192.168.48.1     10        100 I
```

Copyright © 2001, Juniper Networks, Inc.

The slide shows the results of the policy configured and applied on the previous slide.

The **show route protocol bgp** output above shows that the router on the right hand side of the slide is receiving five BGP routes from its IBGP peer. One of those routes is from AS 1 and the other four are sourced from within the AS.

The **show route advertising-protocol bgp** output shows that only the four routes sourced from within the AS are being sent to the EBGP peer. Transit routes have been rejected by the Null AS_Path regular expression in the routing policy.

Review Questions

- What is the intent of prepending a local AS number multiple times to all BGP routes?
- When is it useful to actually remove information from an AS Path?
- What two JUNOS software keywords are used to remove AS Path information?
- What do the following special regex operators match?
 - ?
 - +
 - *
- When is the AS Path for a route considered to be Null?
- Will "0?" also match the Null AS_Path?

Copyright © 2001, Juniper Networks, Inc.

During this module we discussed:

- The different methods that could be used to change the BGP AS-Path attribute, and why this was useful.
- The regular expressions and their operators that were used in a routing policy to find and alter AS Paths.
- Configuring a routing policy that used regular expressions to change the BGP AS-Path attribute and so alter the default traffic flows between AS networks.
- The special role played by the Null AS_Path in a routing policy.



Juniper Networks Advanced Policy

Release 5.0, Revision 1

Module 8: BGP Attributes:

Local Preference

Copyright © 2001, Juniper Networks, Inc.

Objectives

- Describe the importance of the BGP Local_Pref attribute in routing policy
- Compare Local_Pref "positive bias" operation with "negative bias" BGP attribute operation
- Contrast Local_Pref "exit point" behavior with attempts to influence traffic inbound to an AS
- Explain how Local_Pref is used in real-world situations
- Alter the Local_Pref BGP attribute with a policy

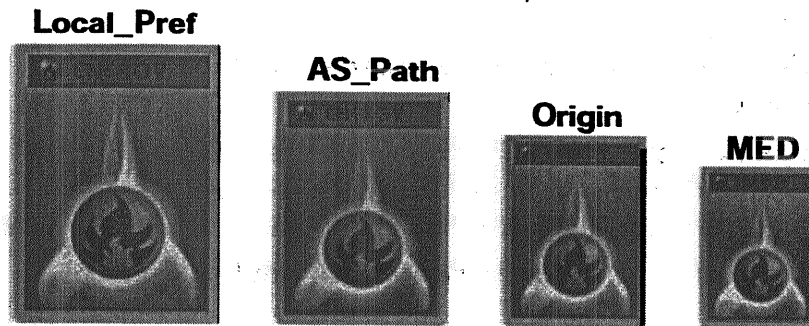
Copyright © 2001, Juniper Networks, Inc.

This module discusses:

- The importance of the BGP Local Preference (Local_Pref) attribute in formulating and using routing policy.
- How the Local_Pref attribute makes routes look more attractive ("positive bias") rather than less attractive ("negative bias"), as do attributes like AS_Path.
- How Local_Pref is used to set an "exit point" for an AS, rather than try to influence traffic inbound to the AS as MED (for example) would do.
- The key role played by the Local_Pref in real-world situations.
- Configuring a routing policy to alter the Local_Pref attribute value.

The Power of Local Preference

- **Local_Pref** is the first BGP attribute used to favor one route over another
- **A route with higher Local_Pref always wins – regardless of AS_Path length**



Copyright © 2001, Juniper Networks, Inc.

The BGP attribute of Local Preference (Local_Pref) is the highest tiebreaker in the BGP path selection process. If a BGP Next-hop is found to be reachable, and there are multiple routes known to BGP, the route with the highest Local_Pref is always chosen. So Local_Pref is the first BGP attribute that is used to favor one path over another.

Because of the position of the BGP Local_Pref, it does not matter what the AS_Path length is for the route, or the Origin code, or the MED. The route with the highest Local Preference value will always be chosen as the exit point of the AS – the end.

Local_Pref is *not* Preference

- Local_Pref is a priority applied *within* BGP itself
- If the Local_Pref for a route is not set, a default value of 100 is used
- Local_Pref is set in `[edit protocols bgp]`
- Local_Pref is changed with `local -preference` in a routing policy
- Preference is a priority applied to routing protocols themselves
- For example, the BGP preference is 170
 - Preference is set in `[edit protocols bgp]`
 - Preference is changed with `preference` in a routing policy
- Make sure to change Local_Pref, not Preference!

Copyright © 2001, Juniper Networks, Inc.

The BGP attribute of Local Preference should not be confused with the JUNOS software concept of route preference. The JUNOS software preference is local to an individual router and assists the routing table in choosing the active route among many possible paths.

The BGP Local Preference attribute is used only within BGP itself. The Local_Pref gets transmitted among the BGP routers within an AS. If a value for Local_Pref is not explicitly configured, the default value used on BGP routes is 100.

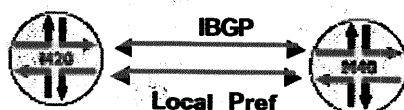
This value can be changed on a per-peer basis using the `local-preference` command within the `[edit protocols bgp]` configuration hierarchy directory. In addition, the value can be altered via a policy on a per-route basis. The policy action also uses the `local-preference` command to alter the attribute value.

The default Local_Pref applied to a BGP route is 100. But the default route preference applied to BGP itself is 170. The JUNOS software preference applies to the entire routing protocol. Preference is also set in the `[edit protocols bgp]` configuration hierarchy directory, but as `preference`, not `local-preference` (which applies only to BGP).

When it comes to routing policy configurations, make sure to change Local_Pref on the BGP routes, not the preference of the BGP protocol as a whole!

Local Preference Advantages

- Setting the Local_Pref does have advantages
- Local_Pref is highest BGP path selection criteria
- No matter how AS_Path is manipulated, Local_Pref can always override the longest contrived AS path
- Not a *negative bias* mechanism, such as AS_Path, but a *positive bias*. The highest Local_Pref is always chosen
- The Local_Pref of the route is distributed to all IBGP routers within the AS



Copyright © 2001, Juniper Networks, Inc.

By default, the Local Preference value is transmitted between all IBGP peers in an AS. The default Local Preference value is 100, but there are advantages to using and setting Local_Pref values within an AS.

Local_Pref is used primarily as a method for defining the *exit point* outbound from the AS. Since the Local Preference value is evaluated first during the path selection process, Local_Pref will override the values set in any other BGP attributes.

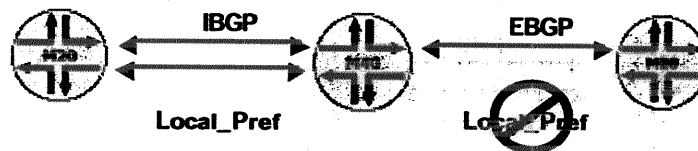
For example, a route with an AS_Path length of 5 and a Local_Pref value of 200 will be chosen over a route with an AS_Path length of 2 and a Local_Pref value of 100. In fact, no matter how the AS_Path is manipulated, Local_Pref will always override the AS_Path.

AS_Path and many other BGP attributes such as Origin or MED (in many cases) form what are known as *negative bias* path selection mechanisms. This means that these attributes function mainly by making some routes look worse than others. In contrast, Local_Pref is a *positive bias* path selection mechanism. That is, there is a default value and a maximum value to assign, and there is no question about how "good" the route looks. The route with the highest Local_Pref is always chosen, period.

The Local_Pref of a route is always distributed to all IBGP peers within an AS. The Local_Pref value can be changed by any router by means of a routing policy.

Local Preference Disadvantages

- Setting the Local_Pref does have some drawbacks
- Local_Pref is exactly that: local to the AS
- Not sent outside the AS in which it applies
- Cannot be used to affect how other AS areas handle the route



Copyright © 2001, Juniper Networks, Inc.

Using Local_Pref to try and influence BGP path selection does have some drawbacks. The biggest disadvantage to the Local Preference attribute is that it is local to the AS. The Local_Pref is distributed to IBGP peers, but only to IBGP peers inside the AS. The Local_Pref, being local, does not get transmitted to adjacent neighbors with EBGP. The Local_Pref attribute is well-known (all BGP implementations must understand Local_Pref), but *discretionary*. Discretionary BGP attributes are only used and understood by one BGP configuration or another (for instance, IBGP or EBGP, but not both). Local_Pref applies to IBGP.

So while Local Preference is very effective at controlling routing decisions within the AS, Local Preference does not assist in affecting routing decisions in other networks. How other AS systems handle the route is still up to them.

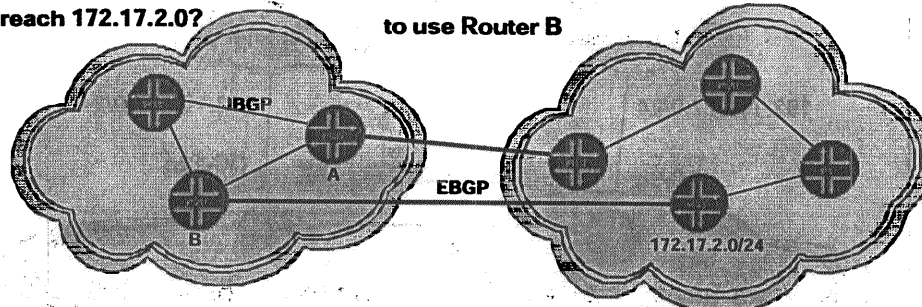
In order to influence traffic *inbound* to an AS from other ASs, other BGP attributes of AS_Path, Origin, and MED must be used. Local_Pref is used to influence *outbound* AS traffic.

Local_Pref Notes

- Exchanged by IBGP peers only
- Usually used to set the *exit point* from an AS
- IBGP propagates information throughout the AS

Which router to reach 172.17.2.0?

It makes sense to use Router B



IBGP makes sure each peer knows to use Router B through Local_Pref

This AS neither knows nor cares about the other AS' Local_Pref

Copyright © 2001, Juniper Networks, Inc.

The values of the Local Preference BGP attribute is exchanged by IBGP peers only. EBGP peers never see an Local Preference set on route information sent between AS networks.

The Local Preference is typically used to set the preferred *exit point* for a particular route from an AS. This can be important when there are several links between two ASs.

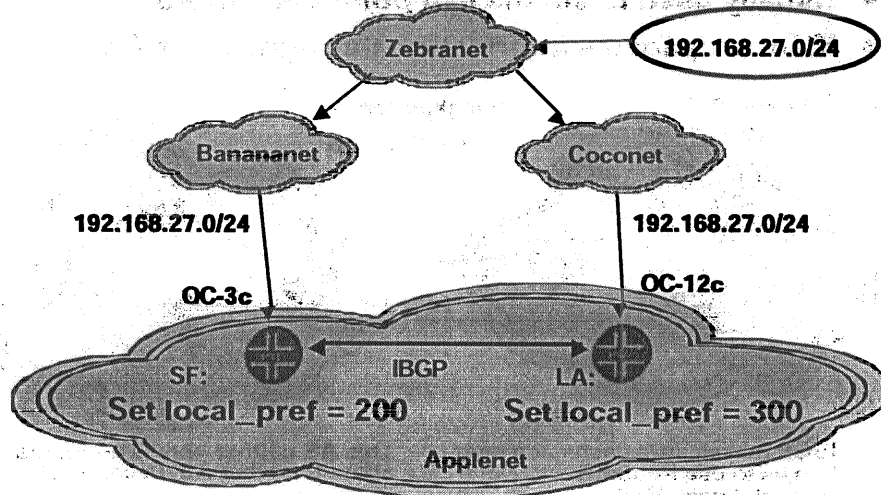
Once the Local Preference for a route is set, IBGP propagates that information throughout the entire AS.

This slide shows the concept of how Local Preference affects traffic leaving an AS. The administrators of the AS on the left have knowledge that Router B should always be used to reach the 172.17.2.0/24 network in the AS on the right. Therefore, the administrators of the AS on the left can configure their edge routers to set the Local Preference value higher on the copy of the route received on router B than the copy received on router A. IBGP will make sure that every router in the AS knows that the preferred exit point for this route is Router B. Note that the AS on the right neither knows nor cares about the value of the Local_Pref assigned to the route by Router A or Router B.

The AS on the left still has fail over capability since it is receiving the route from the AS on the right through both routers. Although router B is the primary exit point for the route, user traffic can use router A to reach the AS on the right in the event of a failure.

Local Preference Example

Applenet wants to use OC-12c outbound, but have OC-3c available for inbound traffic and backup outbound traffic



Copyright © 2001, Juniper Networks, Inc.

Here is an example of how Local Preference is used within an AS.

As shown on the slide, the network administrators in Applenet have decided that the routers in Applenet should use the Los Angeles router to reach the 192.168.27.0/24 network in Zebranet. This is because of the greater bandwidth capacity available on that link: an OC-12c runs at 622 Mbps while an OC-3c runs at 155 Mbps.

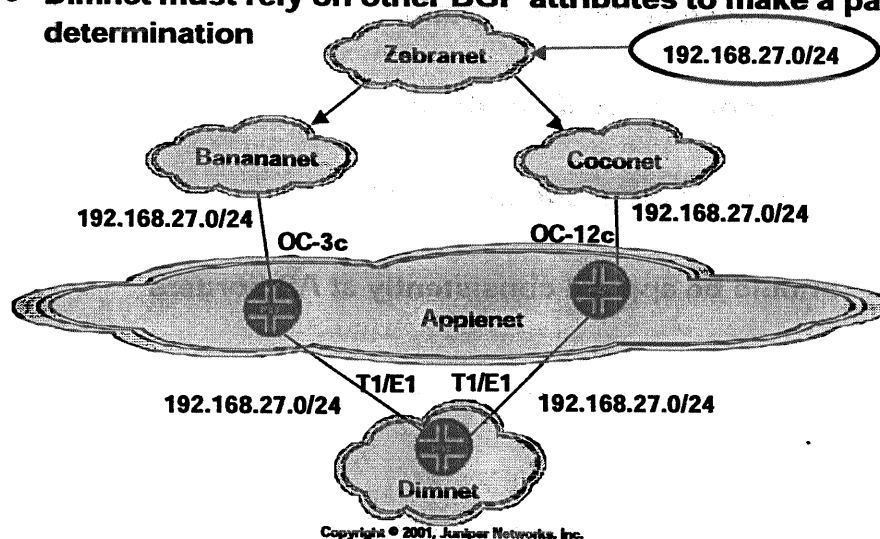
To do this, the Applenet network administrators set the Local_Pref on the route to 192.168.27.0/24 advertised to the San Francisco router by Banananet to 200, and set the Local_Pref on the route to 192.168.27.0/24 advertised to the Los Angeles router by Coconet to 300.

In contrast to almost every other metric associated with routing protocols, the highest Local Preference is better. So for this route, the exit point of the AS is through Los Angeles. IBGP makes these values known to all other routers in Applenet.

The link on the San Francisco router can still be used for inbound traffic flows from Zebranet and for outbound fail over traffic if the OC-12c is not useable because of a router or link failure.

Another Local_Pref Example

- Local_Pref is not propagated to Dimnet
- Dimnet must rely on other BGP attributes to make a path determination



The example on the previous slide is extended further to point out the local nature of Local Preference. Consider another AS called Dimnet linked by two lower, but equal, speed links to Applenet. Which link should Dimnet use to reach 192.168.27.0/24 in Zebranet?

Since the Local Preference values used inside Applenet are not propagated by EBGP, the Dimnet AS has no knowledge of the Local Preference routing decisions made within Applenet with regard to the 192.168.27.0/24 route.

Applenet, of course, would still like to have traffic from Dimnet flow towards the Los Angeles router to avoid shuttling all of this traffic across its backbone.

However, there is no way for Applenet to do this with Local Preference. To accomplish this goal, Applenet must use other BGP attributes instead of Local_Pref such as AS_Path or MED to try and influence Dimnet's flow of traffic to Applenet. This is because of the strictly local nature of the Local_Pref attribute.

Local Preference Issues

- Determines *outbound* AS traffic flow towards the highest Local_Pref exit point
- Most powerful BGP attribute
- Very common usage with an AS
- Should be applied consistently at AS borders

Copyright © 2001, Juniper Networks, Inc.

This slide summarizes the important points with regard to the Local Preference attribute:

- It is a guarantee of *outbound* traffic flows. In fact, there is little else that can be done with BGP to establish an exit point for an AS.
- It is the most powerful BGP path selector and is commonly used to override other BGP attributes in the path selection process. This means that Local_Pref will select a BGP route regardless of the values of AS_Path, Origin, or MED.
- It is very commonly used within an AS, but is local to the AS only. So the Local Preference is never sent to another AS with EBGp.
- It should be applied consistently at the edge of the AS for maximum efficiency. That is, the Local_Pref should be set on a route advertised from another AS at the border router and not changed haphazardly within the AS.

Changing Local_Pref

Change the *Local_Pref* of routes with an *AS_Path* of 124 44 13 to 200 (instead of the default 100)

```
[edit policy-options]
as-path look-for-path "124 44 13"
policy-statement check-the-path {
  term got-path {
    from as-path look-for-path;
    then {
      local-preference 200;
      accept;
    }
  }
}
```

Copyright © 2001, Juniper Networks, Inc.

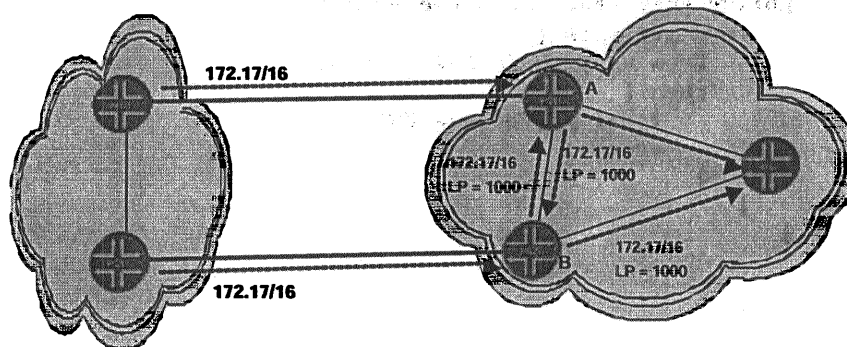
In many cases when it comes to *Local_Pref*, a routing policy will not consider routes in isolation, but will consider other BGP attributes such as the *AS_Path* to select routes for preferred local handling.

The example on the slide alters the Local Preference value for all received routes that match the *AS_Path* of **look-for-path**. Within the term **got-path**, an action of **local-preference 200** is specified. This action will change the *Local_Pref* attribute value for those routes to 200.

All other received BGP routes will not be affected by this routing policy.

Local_Pref Creating Routing Loops?

- Since Local_Pref overrides all other attributes, is it possible to create a routing loop in your network?
 - Both Router A and Router B are setting Local_Pref to 1000 to all IBGP peers (including each other).
 - Will Router A and Router B see each other as the best way to get to the remote network and so form a loop?



Copyright © 2001, Juniper Networks, Inc.

The case study shown on the slide is an interesting look at the operation of BGP in general, and how BGP uses Local Preference specifically.

Since the Local Preference attribute overrides all other BGP attributes in the path selection process, is it possible to set up a routing loop? Routing loops are especially bad since the destination is technically reachable, but some or even all routers within an AS cannot find the proper path to the destination.

In this example, both Router A and Router B have a policy in place that takes received EBGP routes and sets the Local Preference to 1000 as the routes are then advertised to IBGP peers, which include each other. So Router A advertises the route to Router B, and Router B advertises the route to Router A.

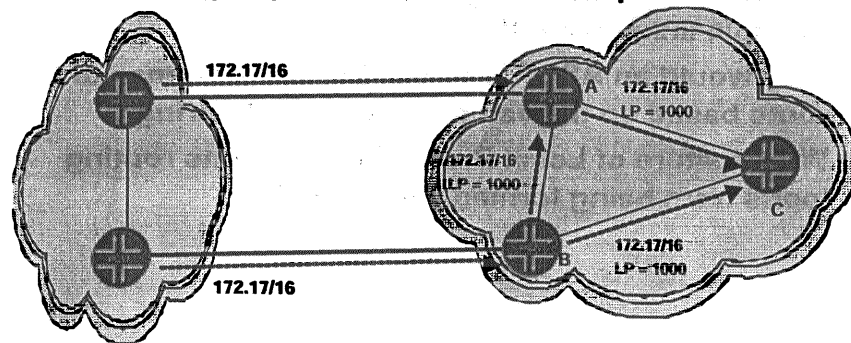
Theoretically, both Router A and Router B will set the value of 1000 from the other and realize that it is better than the *local* copy of the route! This is because the Local Preference on the copy of the route within the router is still set to 100.

Therefore, Router A and Router B will see *each other* as the best route to 172.17.0.0/16 and a routing loop will occur as the two routers shuttle traffic back and forth.

Right?

No Loop in This Case

- IBGP sessions implement split horizon, so only Router A OR B will send out Local_Pref values into their AS
 - Assume Router B sets Local_Pref to 1000 "first"
 - Router A will install the Router B routes as active and will not advertise those routes BACK to Router B
 - A routing loop will not be formed in this particular case



Copyright © 2001, Juniper Networks, Inc.

It turns out that BGP is a little too smart in this case. The following chain of events assumes that Router B received the EBGP route of 172.17.0.0/16 first and advertises the route with a Local Preference of 1000 to its IBGP peers. (If Router A accomplishes this first, the roles are easily reversed, but the point is the same.)

First, Router B receives the EBGP route of 172.17.0.0/16 from its peer in the other AS. Since Router B sees this route as the current best, Router B installs the route in its routing table.

Then Router B alters the Local Preference to 1000 and advertises this route to its IBGP peers with this Local_Pref value. At this point, both Router A and Router C receive the 172.17.0.0/16 route through IBGP.

Router A then also receives the EBGP route for 172.17.0.0/16 from the other AS. Router A finds at this point that it already has a route to 172.17.0.0/16 from Router B with a Local_Pref of 1000. The presence of this route to Router B overrides the EBGP received route.

Since the current version of the route on Router A is from Router B, and since IBGP does implement split horizon, then the routing loop never forms in this case. Router A just sends traffic for 172.17.0.0/16 to Router B. And because only active BGP routes are advertised, there is no confusion on Router C with regard to the best path to reach 172.17.0.0/16: Router B is the choice.

Review Questions

- What is the difference between Preference and Local Preference when it comes to BGP?
- What is the default value the JUNOS software assigns to Local_Pref?
- Is Local Preference only useful when an AS has multiple links to another AS?
- Why would an AS change the Local Preference of a route based on the value of an AS Path string?
- What feature of Local Preference prevents routing loops from being formed within an AS?

Copyright © 2001, Juniper Networks, Inc.

During this module we discussed:

- The importance of the BGP Local Preference (Local_Pref) attribute in formulating and using routing policy.
- How the Local_Pref attribute makes routes look more attractive ("positive bias") rather than less attractive ("negative bias"), as do attributes like AS_Path.
- How Local_Pref is used to set an "exit point" for an AS, rather than try to influence traffic inbound to the AS as MED (for example) would do.
- The key role played by the Local_Pref in real-world situations.
- Configuring a routing policy to alter the Local_Pref attribute value.



Juniper Networks Advanced Policy

Release 5.0, Revision 1

Module 9: BGP Attributes: Communities

Copyright © 2001, Juniper Networks, Inc.

Objectives

- **Explain the basic use of the BGP Community attribute**
- **Describe the importance of well-known BGP communities**
- **Use BGP communities in real-world examples and routing policies**
- **Describe how regular expressions are used with BGP communities**
- **Explain the function of BGP extended communities**

Copyright © 2001, Juniper Networks, Inc.

This module discusses:

- The basic function and use of the BGP Community attribute.
- The importance of the well-known BGP communities that are pre-defined.
- How to use BGP communities in real-world examples and in routing policies to perform certain key routing tasks.
- The key role played by the use of regular expressions in routing policies with regard to BGP communities.
- The function and operation of BGP extended communities.

Community

Where we are going...

- BGP Community
- Notes on Community
- More Community Notes

Copyright © 2001, Juniper Networks, Inc.

Subsequent slides will explore the purpose behind and the operation of BGP communities.

BGP Community

- **An easy way to group routes**
- **BGP attribute (optional) passed along to other BGP peers (transitive)**
- **A group of destinations that share a common property**
- **Are not bounded by a network, AS, or any other physical constraint**
- **Simplify routing policies by identifying routes based on logical bounds established by network administrator**
 - AS number is very broad (lots of routes)
 - IP prefix is very granular (route filter for each route?)
- **Used with other attributes to accept, prefer, or advertise BGP routes**

Copyright © 2001, Juniper Networks, Inc.

The BGP Community attribute is not complex. The main role of the BGP Community attribute is to make it easy to group routes that should be treated the same with a routing policy. BGP communities are very flexible: a BGP route can belong to many BGP communities.

The BGP Community attribute is an optional attribute, so not all implementations of BGP must recognize and use communities. However, if BGP communities are associated with a BGP route, then the BGP Community attribute must be passed along to all other BGP peers, both within an AS and between AS networks (transitive).

The BGP Community attribute is an administrative "tag" that network administrators can use to associate routes together that share common properties.

The nice thing about using BGP communities to group routes is that the communities are globally known, even in other AS networks, and so are not bounded by network, AS, or any other physical constraint.

The attraction of BGP communities is that they can be used to simplify routing policies. BGP communities identify routes based on the logical boundaries established by the network administrator, and not by AS number (too broad in most cases) or individual IP prefix (too granular in most cases).

The community attribute can be used within routing policies to accept or reject routes based on the values of the BGP Community tags. In addition, the community attribute values can be used with other BGP attributes to implement routing policies that accept, prefer, or advertise BGP routes.

Notes on Community

- Establishes *categories* for routes/prefixes
- Often used to set Local_Pref for a group
- Cuts down on manual reconfiguration and complexity of maintenance
- If a new prefix can be placed in a community, no other changes to routing policy are necessary
- Too many communities requires more manual maintenance
- Too many overlapping communities can be a nightmare

Copyright © 2001, Juniper Networks, Inc.

BGP communities are used to establish various logical *categories* for routes (prefixes). Think of BGP communities as "communities of interest" or even "clubs" for routes. And just as people can belong to more than one club, routes can fit into more than one BGP community.

The BGP community attribute value is often used to implement policies for customer networks, such as altering the Local-Pref attribute on incoming routes.

The community attribute can be used to assist network administrators in the configuration and maintenance of policies. This cuts down on the time needed for manual reconfiguration and the complexity of the overall maintenance task.

As an example, consider the addition of a new prefix for a new customer within an AS. If communities are used to enforce routing for customer networks, and the new prefix can be placed into an existing community, no changes to overall routing policies are needed. When new customers are brought into the network, the new routes only need to be assigned the proper community attribute. Without the use of communities, each policy in the network might need to be updated to include the new customer information.

However, care is required when establishing communities for the first time. If there are too many communities, this defeats the whole purpose. Maintenance of the communities then becomes as tedious as maintaining a whole list of route filters.

Network administrators should also avoid having too many overlapping communities. When customer routes belongs to multiple communities, this too can be a nightmare.

When it comes to communities, simplicity is always a worthwhile goal.

More Community Notes

- **Well-known communities have a global meaning**
- **Local-use communities must be defined**
- **Community attribute is a list of four-octet individual community attribute values associated with a route**
 - Route can belong to many communities
 - 2 high order octets represent an AS number
 - 2 low order octets represent a value unique to that AS
 - Represented in decimal form of *AS:number* (e.g. 200:666)
 - All values in AS 0 & 65535 are reserved: 0x00000000 to 0x0000FFFF and 0xFFFF0000 to 0xFFFFFFFF

Copyright © 2001, Juniper Networks, Inc.

There are certain *well-known communities* that have a global meaning and special purposes. All BGP implementations that understand communities (communities are optional, but transitive) must know these well-known communities and respect their functions.

BGP communities that are not well-known are for local use. Local-use communities must be defined locally, but since the BGP Community attribute follows the BGP route wherever it goes, even local-use communities are circulated outside the AS. So care is needed in using BGP Community attribute values arriving from other AS networks. BGP communities are best thought of as a category for a group of routes (such as "all customers").

The BGP Community attribute itself is just a list of the individual community attribute values associated with the route ("tags"). Since there is no real limit on the number of tags in the list, a route can belong to many communities. There is no predefined upper limit on the number of communities allowed on a route.

The community attribute values themselves are designed so the values can be uniquely assigned in the global Internet. The BGP Community attribute itself is a 32-bit (four octet) value that has two parts. The two high order octets (16-bits) form the first part and are set aside for the AS number of the network that assigned the community to the route in the first place. The two low order octets (16-bits) form the second part and are set aside for use by the specific AS networks. Since the AS value is included in the community, the value is guaranteed to be unique on the whole Internet.

When written in bits or hex, communities can look very odd. So communities are usually represented in decimal form as *AS:number* (for example 200:666). This is community 666 in AS 200. One restriction on possible communities values is that the AS values of 0 and 65535 are reserved and can not be assigned to a route. So in combination, this restriction covers values 0x00000000 to 0x0000FFFF (AS 0) and 0xFFFF0000 to 0xFFFFFFFF (AS 65535).

Well-Known Community Values

Where we are going...

- Well-Known Communities
- NO_ADVERTISE
- NO_EXPORT_SUBCONFED
- NO_EXPORT
- Examples of Well-Known Communities
- Backup Routes
- Transit and No-Transit

Copyright © 2001, Juniper Networks, Inc.

Subsequent slides will explore the BGP well-known community values, their usage, and provide some real-world examples.

Well-Known Communities

- **Three standard values are established:**
 - **NO_EXPORT (0xFFFFF01):** distribute these routes within confederation (or AS), but no farther
 - **NO_ADVERTISE (0xFFFFF02):** These routes must not be advertised to other BGP peers
 - **NO_EXPORT_SUBCONFED (0xFFFFF03):** These routes must not be advertised to external BGP peers (confined to sub-AS)
- **NO_EXPORT typically used to keep aggregation optimal**
 - **NO_EXPORT_SUBCONFED** extends this to the sub-AS
- **NO-ADVERTISE has narrower scope**
 - **Often used in LAN-connected router environment**

Copyright © 2001, Juniper Networks, Inc.

There are three community attribute values that have been designated as well-known communities. These well-known communities share the AS value of 65535 (0xFFFFxxx). The three communities are NO_EXPORT, NO_ADVERTISE, and NO_EXPORT_SUBCONFED.

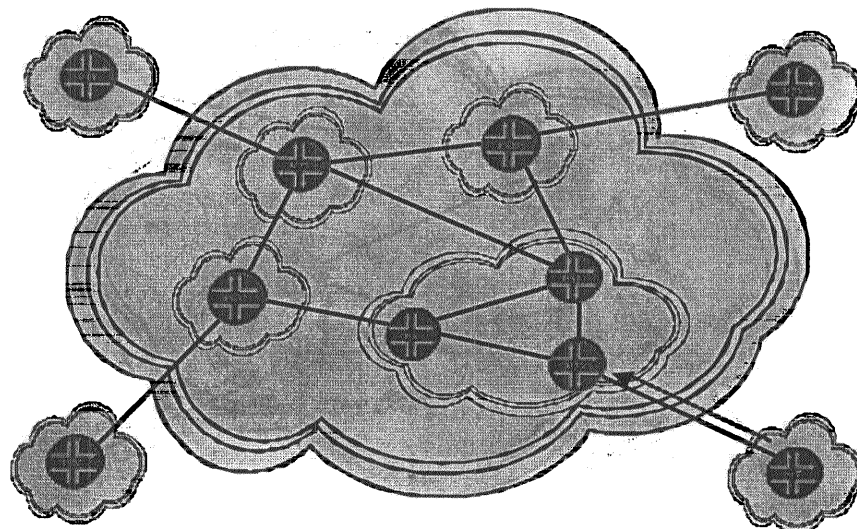
- **NO_EXPORT (0xFFFFF01)** tells routers to distribute routes with this community tag within the confederation or AS, but no farther.
- **NO_ADVERTISE (0xFFFFF02)** tells routers not to advertise these routes to other BGP peers at all. (In spite of appearances, this BGP Community is quite useful.)
- **NO_EXPORT_SUBCONFED (0xFFFFF03)** tells routers not to distribute routes with this community tag to external BGP peers (so they are confined to the sub-AS).

The NO_EXPORT community is typically used to make sure that route aggregation is optimal by suppressing more specific routes. The NO_EXPORT_SUBCONFED just extends this aggregate concept to the sub-AS.

The NO_ADVERTISE community has a very narrow scope. Routes go to a BGP peer and no farther, usually because the routes are known to peers through other means. This community is often used in a LAN-connected router environment or when 2 BGP peers have multiple links between them.

NO_ADVERTISE

Routes go no further than the next-hop



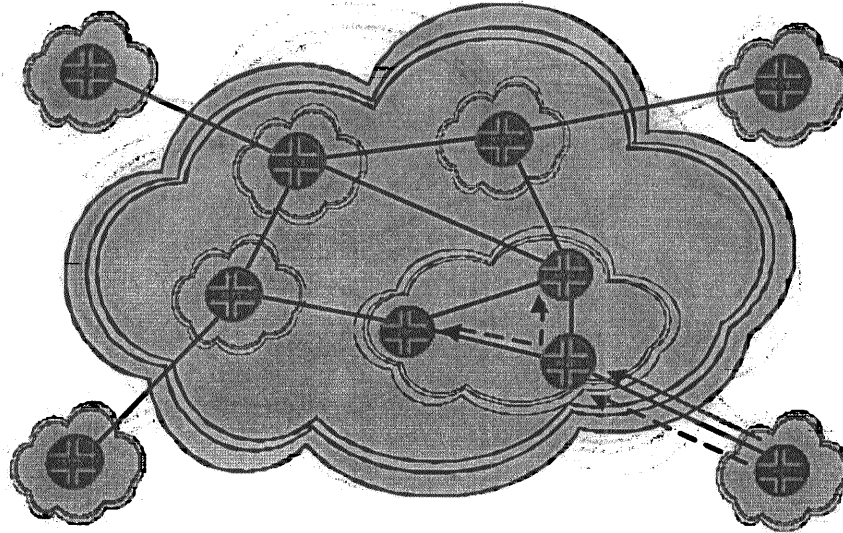
Copyright © 2001, Juniper Networks, Inc.

The slide shows an example of the scope of the NO_ADVERTISE community. The arrow at the lower right shows a BGP route with the NO_ADVERTISE community injected into an AS with sub-confederations.

The well-known community attribute value of NO_ADVERTISE is designed such that a route can be sent to a single BGP peer and be advertised no further. Routes are restricted to the next-hop router.

NO_EXPORT_SUBCONFED

Routes go no further than the sub-AS

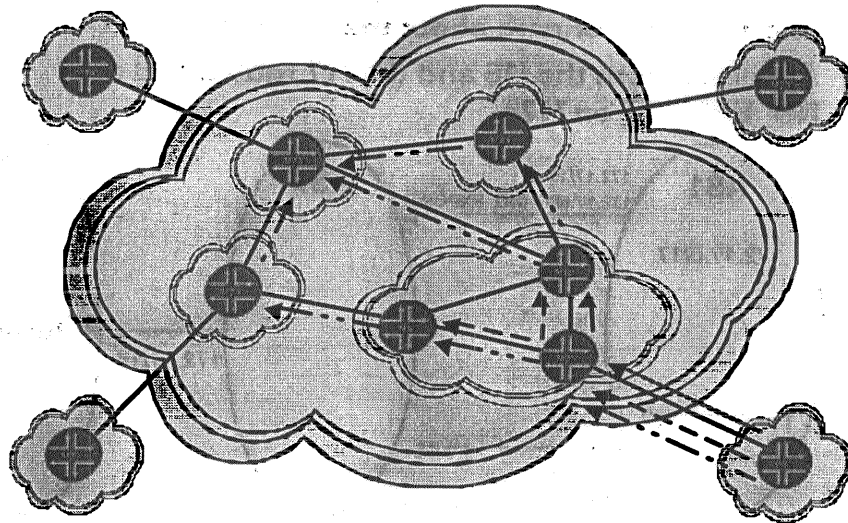


Copyright © 2001, Juniper Networks, Inc.

The slide shows an example of the scope of the NO_EXPORT_SUBCONFED community. The arrow at the lower right shows a BGP route with the NO_EXPORT_SUBCONFED community injected into an AS with sub-confederations. The well-known community attribute value of NO_EXPORT_SUBCONFED is designed such that a route can be sent into a BGP confederation network and have the information remain with a particular sub-AS. The routing information will be advertised no further than the sub-AS, as shown on the slide.

NO_EXPORT

Routes go no further than the entire AS



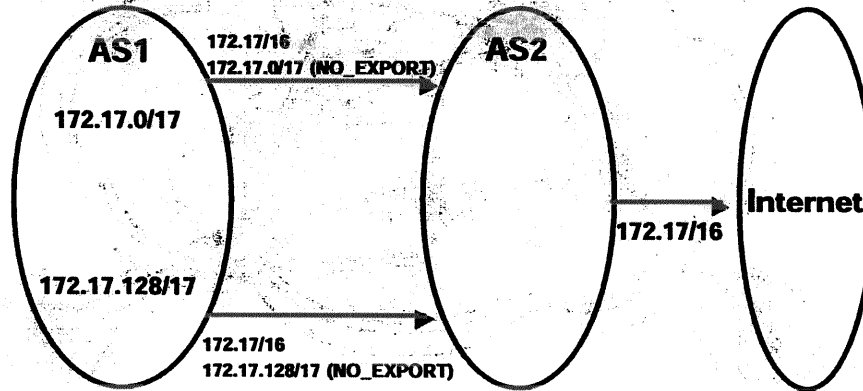
Copyright © 2001, Juniper Networks, Inc.

The slide shows an example of the scope of the NO_EXPORT community. The arrow at the lower right shows a BGP route with the NO_EXPORT community injected into an AS with sub-confederations.

The well-known community attribute value of NO_EXPORT is designed such that a route can be sent into a neighboring AS and have the information remain within that neighboring AS. Normally, BGP communities are transitive and are passed from each AS to all others, even if the BGP community option is not supported and used by the router. But the routing information with the NO_EXPORT community tag will not be advertised beyond the AS, as shown on the slide.

Example Using NO_EXPORT

- AS2 should use "load sharing," but why should the Internet know or care about the /17 routes?
- Advertise both the /16 and the /17 routes and use NO_EXPORT on /17s



Copyright © 2001, Juniper Networks, Inc.

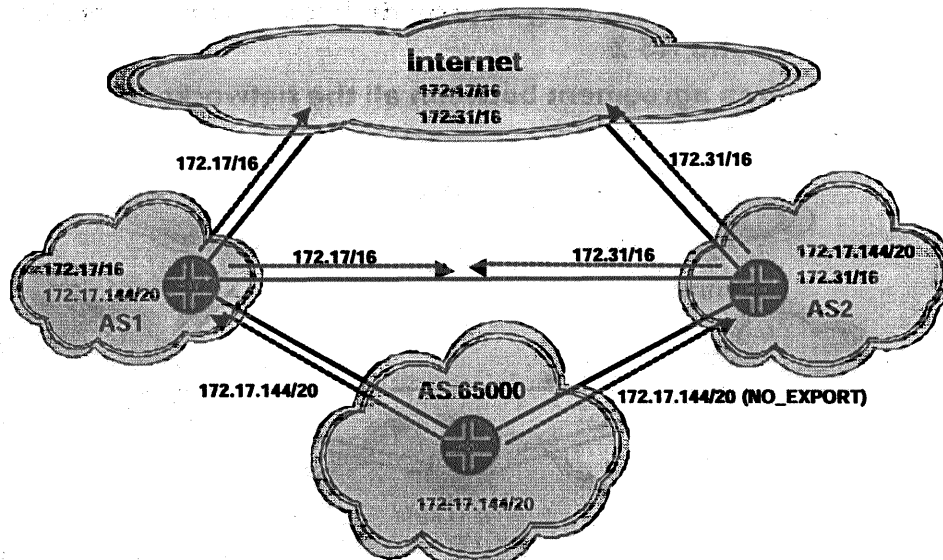
The slide shows a typical use of the BGP NO_EXPORT well-known community.

AS 1 has multiple BGP sessions to its neighbor AS 2. AS 1 is also using AS 2 as a transit AS for connectivity to the Internet. AS 1 would like to advertise 172.17.0.0/16 to the Internet because AS 1 owns that entire address space. In addition, AS 1 would also like to advertise more specific route information (shown as 172.17.0/17 and 172.17.128/17) only to its peer, AS 2.

Advertising the specifics as well as the aggregate to AS 2 would assist AS 2 to more efficiently route user traffic into AS 1, since "load sharing" could be used on the more specific routes as shown on the slide. But why should the whole Internet know or care about these specifics?

To assist AS 2 in finding and rejecting the more specific routes, AS 1 assigns the NO_EXPORT community to the 172.17.0.0/17 and the 172.17.128.0/17 routes. The BGP edge routers in AS 2 that connect to the Internet will automatically suppress and not readvertise the /17 routes. Only the /16 route is readvertised to the Internet.

Multi-Homing using NO_EXPORT



Copyright © 2001, Juniper Networks, Inc.

The BGP well-known community NO_EXPORT is also useful when a customer is multi-homed to two ISPs. In this example, the customer would like to receive Internet traffic through one main ISP, but be able to receive locally originating traffic from the other ISP (perhaps a major trading partner uses AS 2 as its ISP).

One way to do this (there are other ways) is with the BGP NO_EXPORT Community. In this example, the customer has AS 1 as its main ISP.

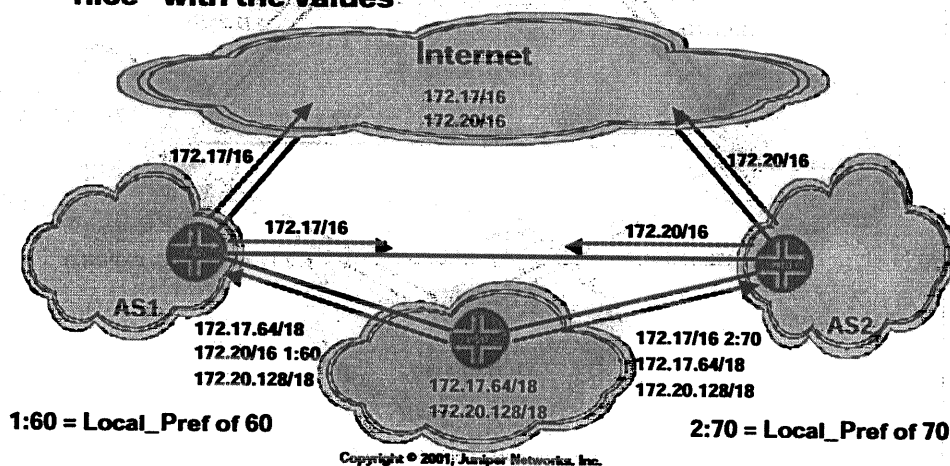
Customer AS 65000 has address space 172.17.144.0/20, taken from AS 1's address space of 172.16.0.0/16. The 172.17.144.0/20 route is advertised with BGP to both AS 1 and AS 2. Since AS 2 is not the primary ISP and only traffic originating in AS 2 should reach AS 65000 directly, the route advertised to AS 2 is given the BGP NO_EXPORT community. This route goes no farther than AS 2. AS 2 will advertise 172.31.0.0/16 to AS 1 and the Internet, but not 172.17.144.0/20.

AS 1 covers the 172.17.144/20 address space with aggregate 172.17.0.0/16, as shown on the slide. This is advertised to AS 2 and the Internet. So AS 65000 is reachable through AS 1 from the Internet, but AS 65000 is only reachable by local AS 2 users.

A drawback of this scenario is that if the link from AS 65000 to AS 1 fails, the 172.17.144.0/20 route is not reachable from the Internet through AS 2. But since AS 2 is not the primary ISP for AS 65000, perhaps this is acceptable.

Backup Routes

- Customer is agreeing to provide backup transit service to AS1 and AS 2
- Requires agreement between all the networks to "play nice" with the values



One of the most common uses of the BGP Community attribute addresses one major limitation of the BGP Local Preference attribute. Local Preference is only distributed within an AS and no Local Preference information is ever sent between two AS networks. Yet Local Preference is a valuable way to establish proper exit points for a route within an AS, especially when the AS has multiple links. How can another AS be informed of the Local Preferences of a route learned from another AS? The most popular way is with BGP communities.

In this example, the customer AS at the bottom has agreed to provide backup transit service to both AS 1 and AS 2 in case their links to each other are lost. The address spaces used are shown on the slide. The customer AS uses 172.17.64.0/18 and 172.20.128.0/18.

What the customer wants to accomplish is to make the 172.20.0.0/16 route sent to AS 1 to have a *lower* Local Preference than the default of 100, and to make the 172.17.0.0/16 route sent to AS 2 to have a *lower* Local Preference than the default of 100 there as well.

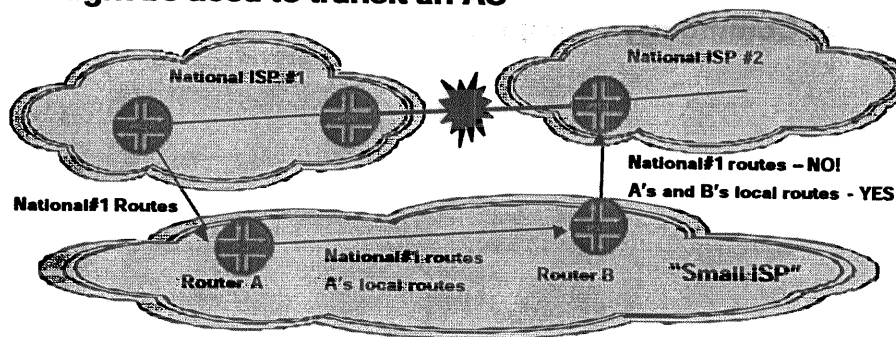
The key to making this work is with the BGP communities on routes sent to AS 1 and AS 2. Route 172.20.0.0/16 sent to AS 1 is tagged as community 1:60. This says to AS 1, "AS 1, within your AS, this route should have a Local Preference of 60." Naturally, route 172.17.0.0/16 sent to AS 2 is tagged as community 1:70. This says to AS 2, "AS 2, within your AS, this route should have a Local Preference of 70." In this example, AS 1 and AS 2 only advertise aggregates to each other and the Internet.

If AS 1 and AS 2 set the Local Preferences on these routes as requested, the exit points through the customer's AS will only be active when there is no "normal" peering route available (Local Preference = 100).

Of course, setting Local Preferences in other AS networks with communities requires that all of the AS administrators "play nice" and cooperate. Nothing makes an AS respect the community attribute value.

Transit and Non-transit

- Policies can be used in BGP to make an AS appear as a transit AS or non-transit AS for a route
- Communities make it easier to "hold back routes" that might be used to transit an AS



"Small ISP" needs to advertise own routes, but never be transit AS!

Copyright © 2001, Juniper Networks, Inc.

A key area for the application of the BGP Community attribute is with regard to transit and non-transit AS networks. A transit AS carries traffic that neither originates nor is destined for hosts within an AS. A non-transit AS only carries traffic that has its own addresses appearing as either the source or the destination. Non-transit AS networks must be careful when it comes to advertising BGP routes outside of the AS. Only local routes can be advertised by a non-transit AS.

Routing policies can be used in combination with BGP communities to make an AS appear as a transit AS or a non-transit AS. Communities make it much easier to "hold back routes" that might be advertised and attract transit traffic.

Generally, a small ISP needs to advertise its own local routes, but never be a transit AS for larger ISPs. Such a situation could easily swamp the operation of a small ISP.

The slide shows a small ISP linked to two larger, national ISPs: National ISP #1 and National ISP #2. As an example of what could happen, consider that National ISP #1 advertises BGP routes into Router A of the small ISP as shown on the left. Router A advertises not only its own local routes to Router B in the small ISP, but also National ISP #1's routes so that all users can reach these routes.

But Router B should never, ever advertise National ISP #1's routes to National ISP #2! Router A and Router B's local routes are okay to send to National ISP #2. But if Router B ever advertises the National ISP #1 routes to National ISP #2 and the link (or links) between the two national ISPs ever fail, National ISP #2 will think that a good way to reach National ISP #1 is through the small ISP! Hence, the small ISP has now become a transit network.

Community Configurations

Where we are going...

- **Configuring a BGP Community**
- **Community Actions: Add, Delete, Set**
- **Community Examples**

Copyright © 2001, Juniper Networks, Inc.

Subsequent slides will explore to configure and use BGP communities within the JUNOS software policy framework.

Configuring a BGP Community

- Done at **[edit policy-options]** level
 - Multiple **community-ids** mean a logical AND

```
[edit policy-options]  
community name members [community-ids];
```
- Where **community-ids** have the format:
 - **as-number:community-value**
 - **community-id** can also be:
 - **no-export**
 - **no-advertise**
 - **no-export-subconfed**
- Used in a policy as a match condition or an action
 - Actions include add, delete, and set

Copyright © 2001, Juniper Networks, Inc.

BGP communities are configured in the JUNOS software at the **[edit policy-options]** CLI hierarchy level. The community is simply given a name and a number of "members" in the form of the **community-id**. When multiple community members are defined, it means there is a logical AND between them. So a given name represents Community1 AND Community2 AND Community3 and so on.

The **community-id** has an **as-number:community-value** format, with a colon (:) separating the high order and low order octets. The keywords **no-export**, **no-advertise**, and **no-export-subconfed** can be used to specify the well-known community values.

When used in a routing policy, the community name can be used as a match condition ("find these BGP communities") or as an action. Actions applied to communities include the adding, deleting, or setting of community attribute values.

Community Actions: Add

Leave existing communities alone and add in the specified value

192.168.0.0/24 (2 entries, 1 announced)

Communities: 64512:666 100:20 50:70 1234:66

[edit policy-options]

policy-statement community-actions {

term add-a-community

then community add test-comm;

}

}
community test-comm members 65001:1234;

192.168.0.0/24 (2 entries, 1 announced)

Communities: 64512:666 100:20 50:70 1234:66 65001:1234

Copyright © 2001, Juniper Networks, Inc.

The slide shows the definition and application of a community in a routing policy. This policy is used to leave the existing community tags on the route in place, but also add the specified community attribute value to the route.

Route 192.168.0.0/24 currently has four community tags on the route: 64512:666, 100:20, 50:70, and 1234:66.

Since the policy **community-actions** has no **from**, all routes are matched. It is not necessary to check for just the BGP routes, since only BGP has a Community attribute to change. (Including a **from protocol bgp** statement will not change the action of the routing policy.

All BGP routes will have the community tag **test-comm** value of 65001:1234 added to the existing community tags on the route. This is done by the action of **then community add test-comm**.

After this routing policy has been correctly applied, the 192.168.0.0/24 route now has five community tags on the route: 64512:666, 100:20, 50:70, 1234:66, and the added 65001:1234.

Community Actions: Delete

Remove only the specified values and leave other existing communities alone

192.168.0.0/24 (2 entries, 1 announced)
Communities: 64512:666 100:20 50:70 1234:66

```
[edit policy-options]
policy-statement community-actions {
  term add-a-community
    then community delete test-comm;
}
community test-comm members 64512:666;
```

192.168.0.0/24 (2 entries, 1 announced)
Communities: 100:20 50:70 1234:66

Copyright © 2001, Juniper Networks, Inc.

This slide also shows a policy that defines and applies a community in a routing policy. This policy is used to remove only the specified values of the existing community tags, and leaves other existing community tags in place.

Route 192.168.0.0/24 currently has the same four community tags on the route: 64512:666, 100:20, 50:70, and 1234:66.

Since the policy **community-actions** has no **from**, all routes are matched.

All BGP routes will have the community tag **test-comm** value of 64512:666 *deleted* from the existing community tags on the route. This is done by the action of **then community delete test-comm**.

After this routing policy has been correctly applied, the 192.168.0.0/24 route now has only *three* community tags on the route: 100:20, 50:70, and 1234:66.

Community Actions: Set

Remove ALL existing communities and add the specified values

192.168.0.0/24 (2 entries, 1 announced)
Communities: 64512:666 100:20 50:70 1234:66

```
[edit policy-options]
policy-statement community-actions {
  term add-a-community
    then community set test-comm;
}
community test-comm members 65001:1234;
```

192.168.0.0/24 (2 entries, 1 announced)
Communities: 65001:1234

Copyright © 2001, Juniper Networks, Inc.

This slide also shows a policy that defines and applies a community in a routing policy. This policy is used to remove **all** of the values of the existing community tags, and adds (sets) the new community tag (or tags) in place.

Route 192.168.0.0/24 currently has the same four community tags on the route: 64512:666, 100:20, 50:70, and 1234:66.

Since the policy **community-actions** has no **from**, all routes are matched.

All BGP routes will have the community tag **test-comm** value of 64512:666 set as the existing community tag on the route. This is done by the action of **then community set test-comm**.

After this routing policy has been correctly applied, the 192.168.0.0/24 route now has only **one** community tag on the route: 65001:1234.

Community Example #1

- Create and apply a community named *customers*
 - Set MED to 20
 - Set BGP local preference to 200 (instead of 100)

```
[edit policy-options]
community customers members [56:2379 23:46944];
policy-statement from-customers {
    from community customers;
    then {
        metric 20;
        local-preference 200;
        next policy;
    }
}
```

Copyright © 2001, Juniper Networks, Inc.

The slide shows a realistic application of the BGP Community attribute.

The goal here is create a community named **customers**. The community is defined as having two members: 56:2379 AND 23:46944.

When this community is used in a routing policy to find (match) routes, the community will match routes that have **both** 56:2379 AND 23:46944 as a community tag value.

Once found, these routes are given a MED (the BGP **metric**) of 20 and a **local-preference** of 200 (instead of the default 100).

The slide shows how this community is defined and applied in the routing policy.

Community Example #2

- Create and apply a community named **my-accept**
 - Reject routes more specific than /19 for Class A, /16 for Class B, and /24 for Class C
- ```
[edit policy-options]
community my-accept members 666:1;
policy-statement drop-specifics {
 from {
 route-filter 0.0.0.0/1 upto /19
 community add my-accept
 next policy;
 route-filter 128.0.0.1/2 upto /16
 community add my-accept
 next policy;
 route-filter 0.0.0.0/0 upto /24
 community add my-accept
 next policy;
 }
 then reject;
}
```

Copyright © 2001, Juniper Networks, Inc.

The slide shows another realistic example of a BGP community.

The goal here is to create and apply a community named **my-accept**. This community will be used to drop specific routes and tag all other routes with the community value of 666:1.

The routing policy shown will accept the desired routes with a series of route filters, while at the same time using the **community add** statement to add the community attribute value of 666:1 to any existing communities on the accepted routes.

## Communities in Routing Policy

- Some routing policies *delete* communities, which might be useful at AS boundaries
- By default, communities are sent to peers
- To stop community advertising, delete the community
- This policy will delete all communities. The (\*) wildcard matches any possible value

```
[edit policy-options]
community wild-match members "::*";
...
policy-statement delete-all-communities {
 term all-gone {
 then community delete wild-match;
 }
}
```

Copyright © 2001, Juniper Networks, Inc.

In contrast to other BGP attributes like AS\_Path, it is perfectly allowable to delete some or all of the community attribute values on a BGP route. In fact, this is very useful to do at the boundaries of an AS, since there is no guarantee that any other AS will understand or respect the values of the communities established in one AS.

If this is not done, by default all BGP communities are sent to all peers inside an AS and outside the AS. Why clutter up the routing updates with useless and potentially harmful information?

In order to stop the community from being advertised beyond the local AS, it is necessary to delete the community.

The slide shows a routing policy that will delete *all* communities from a BGP route. This example uses the wildcard character (\*) to match all communities. This is done with the action of **community delete wild-match**.

Note that the wildcard must be applied to *both* "halves" of the community: the AS number as well as the community-value. The syntax is therefore "::\*" to match all communities.

## Pop Quiz (I)

What happens when the route is evaluated?

```
192.168.0.0/24 (2 entries, 1 announced)
Nexthop: 10.222.3.2
MRD: 5
AS path: 10 I
Communities: 64512:666
```

```
[edit policy-options]
policy-statement incoming-routes {
 term can-you-get-this
 from community nap-peers;
 then {
 community delete all-wild;
 next policy;
 }
}
community all-wild members *:*;
community nap-peers members 64512:666;
```

Copyright © 2001, Juniper Networks, Inc.

What will happen when the route 192.168.0.0/24 is evaluated by the routing policy on the slide? The communities used in the routing policy are defined at the bottom of the slide.

The 192.168.0.0/24 route has one community tag already: 64512:666. This value is also defined on the slide as **nap-peers**. The **from community nap-peers** statement will match this route based on this community attribute value.

The routing policy then deletes all communities that match the definition of **all-wild**. Since **all-wild** is defined to match all communities ("**\*:\***") the 64512:666 community is **deleted** from the 192.168.0.0/24 route.

## Pop Quiz (II)

### What happens when the route is evaluated?

192.168.128.0/24 (2 entries, 1 announced)  
 Nexthop: 10.222.4.2  
 MED: 20  
 AS path: 30 20 10 I  
 Communities: 64512:1234 65001:10 65002:10 65001:11

```
[edit policy-options]
policy-statement clear-comm {
 term i-dont-like-comms
 from community [as111-routes as222-routes]; ← OR
 then {
 community delete all-comms;
 next policy;
 }
}
community as111-routes members [65001:10 65001:11]; ← AND
community as222-routes members [65002:10 65002:11];
community all-comms members [65001:* 65002:*];
```

Copyright © 2001, Juniper Networks, Inc.

What will happen when the route 192.168.128.0/24 is evaluated by the routing policy on the slide? The communities used in the routing policy are defined at the bottom of the slide.

The 192.168.128.0/24 route has four community tags already: 64512:1234, 65001:10, 65002:10, and 65001:11.

Two communities, 65001:10 AND 65001:11 are defined as **as111-routes**. Two others, 65002:10 AND 65002:11 are defined as **as222-routes**.

Also, a community named **all-comms** is defined as all community-values from AS numbers 65001 AND 65002 (65001:\* and 65002:\*).

Note that when these named communities are applied in a routing policy, the normal matching rules apply and the matches are considered to be a logical OR. So the routing policy will match and perform the action if **as111-routes** OR **as222-routes**, as defined in the community, match the communities.

However, community *members* are always defined as a logical AND. This means that any route **must** have both 65001:10 **and** 65001:11 present as community attribute values to produce a match with **as111-routes**. The same applies to **as222-routes** a route **must** have both the 65002:10 **and** 65001:11 present on the route to produce a match.

So this routing policy matches the 65001:10 and 65001:11 tags on 192.168.128.0/24. Since **all-comms** is defined to match all communities values having AS numbers 65001 AND 65002 (65001:\* and 65002:\*), all of these communities are **deleted** from the 192.168.128.0/24 route, leaving only 64512:1234 remaining.

Note that 65002:10 was deleted, even though the routing policy did not explicitly match on this community.

## Pop Quiz (III)

**What happens when the route is evaluated?**

```
192.168.240.0/24 (2 entries, 1 announced)
Nexthop: 10.222.7.2
MED: 15
AS path: 20 10 I
Communities: 512:1234 6500:10 1002:66 65001:1111
```

```
[edit policy-options]
policy-statement interesting-comm-example {
 term what-happens-here {
 then {
 community set set-comms-1;
 community set set-comms-2;
 }
 }
}
community set-comms-1 members 65550:6789;
community set-comms-2 members 10:10;
```

Copyright © 2001, Juniper Networks, Inc.

What will happen when the route 192.168.240.0/24 is evaluated by the routing policy on the slide? The communities used in the routing policy are defined at the bottom of the slide.

The 192.168.240.0/24 route has four community tags already: 512:1234, 65001:10, 1002:66, and 65001:1111.

Two communities are defined. One is **set-comms-1** with the single member of 65550:6789, and the other is **set-comms-2** with the single member of 10:10.

Since there is no **from** in the routing policy, this routing policy matches all routes, including BGP routes with community tags.

The actions in the policy first sets the community attribute value to 65550:6789, *deleting* all four of the existing tags because of the way in which the **set** action works. The **set** removes all existing communities and attaches only the community in the action, in this case 65550:6789.

However, the second action of **community set set-comms-2** immediately deletes the "new" community of 65550:6789 and replaces it with 10:10.

As a result of the combined action of the two **set** statements, the 192.168.240.0/24 route will have only one community, 10:10.



## Regular Expressions

### Where we are going...

- **Regex and Communities**
- **Community Matching**
- **More Complex Regex**
- **Regular Expression Operators**
- **Regular Expression Examples**

Copyright © 2001, Juniper Networks, Inc.

Subsequent slides will explore how regular expressions can be used in a routing policy to match BGP communities.

## Regex and Communities

- **"Simple" community regex expressions can use wildcard values of ( \* ) or ( . )**
  - The ( \* ) matches any single AS number or community-value
  - The ( . ) matches any single digit within the AS or community-value
  - The combination of these wildcards ( . \* ) means something different and is considered a "complex" community regex
- **Examples of "simple" community regex matches**
  - All communities from particular AS: "as-number:"
  - Example: "600:" matches all AS 600 communities
  - All AS networks with the same value: ":"community-value"
  - Example: ":"20" matches value 20 from all AS areas
  - All community-values where the 3<sup>rd</sup> digit is any number: (5000, 5010, 5020, etc. up to 5090 from AS 1111)
  - "1111:50.0"

Copyright © 2001, Juniper Networks, Inc.

A regular expression (regex or RE), first introduced in an earlier section on AS Paths, can also be used with BGP communities to produce a powerful pattern matching system for finding communities that match a given regex. Regular expressions used with communities implement the full capabilities of a complete regex implementation, unlike the AS\_Path regex syntax, which used a subset.

It will be helpful to consider two forms of regular expressions used with routing policies concerned with BGP communities. These two forms are "simple" and "complex" regular expressions. These are informal terms that are defined in this course as follows. "Simple" community regular expressions contain only the \* (asterisk) or . (dot) wildcard characters separately. "Complex" community regular expressions can use the \* (asterisk) and . (dot) in conjunction with each other. Further, the "complex" regex statements can use additional operator syntax characters.

The \* (asterisk) matches any single AS number or community-value. The . (dot) matches any single digit within the AS number or community-value. Note that the combination of these characters (.\* ) is a "complex" community regex and will be considered on a later slide.

Some examples of "simple" community regex matches include:

- **\*:1000** = Any possible AS number with a value of 1000
- **65001:\*** = AS 65001 with any possible community value
- **65001:100.** = AS 65001 with community values of 1000, 1001, 1002, 1003, 1004, 1005, 1006, 1007, 1008, or 1009
- **11.1:1000** = AS 1101, 1111, 1121, 1131, 1141, 1151, 1161, 1171, 1181, or 1191 with a community value of 1000

## Community Matching

```

user@host> show route community *:20 terse

inet.0: 123 destinations, 123 routes (123 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

A Destination P Prf Metric 1 Metric 2 Next hop AS path
* 192.168.128.0/24 S 5 5 Reject

```

```

user@host> show route community *:20 detail

inet.0: 123 destinations, 123 routes (123 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

192.168.128.0/24 (1 entry, 1 announced)
 *Static Preference: 5
 Next hop type: Reject
 State: <Active Int Ext>
 Age: 3d 6:07:37 Metric: 5
 Task: RT
 Announcement bits (3): 0-KRT 1-BGP.0.0.0.0+179 5-Aggregate
 AS path: I
 Communities: 1:20

```

Copyright © 2001, Juniper Networks, Inc.

Standard CLI commands can be used with simple regular expressions to find the routes (if any) associated with any community. A few examples are shown on the slide.

To find all routes that have a community-value of 20 regardless of AS number, use **show route community \*:20 terse**. This command shows the routes, but not the complete community attribute values associated with the routes. To see the communities and more detailed information, use **show route community \*:20 detail**.

## More Complex Regex

- More complex regular expressions can be used with communities
  - Community regex is *character based* (not like AS\_Path)
  - Format is still **term<operator>**
  - Regex anchors ( **^** ) and ( **\$** ) are not required, but can be helpful
  - Used in both **show route** and within a policy as a match-condition
    - **show route community <regex>**
    - **community match-this members <regex>**

Copyright © 2001, Juniper Networks, Inc.

More complex regular expressions (regex) can be used with communities. A community regular expression is character based, unlike the regular expressions used with AS Paths, which match entire AS numbers.

The format for the community regular expression is still **term<operator>** as in AS Path regular expressions, but the application of the term and operator is different.

When formulated for use with communities, the regular expression anchors of **^** (start) and **\$** (end) are not required, but can be helpful to organize and clearly represent the regular expression.

Complex regular expressions can be used in both the **show route** CLI commands and within a policy as a match condition.

## Regular Expression Operators

|                |                                                                               |
|----------------|-------------------------------------------------------------------------------|
| <b>{m,n}</b>   | <b>Match a least <i>m</i> and at most <i>n</i> repetitions of <i>term</i></b> |
| <b>{m}</b>     | <b>Match exactly <i>m</i> repetitions of <i>term</i></b>                      |
| <b>{m,}</b>    | <b>Match <i>m</i> or more repetitions of <i>term</i></b>                      |
| <b>*</b>       | <b>Match 0 or more repetitions of <i>term</i>, same as {0,}</b>               |
| <b>+</b>       | <b>Match 1 or more repetitions of <i>term</i>, same as {1,}</b>               |
| <b>?</b>       | <b>Match 0 or 1 repetitions of <i>term</i>, same as {0,1}</b>                 |
| <b> </b>       | <b>Match one of the two <i>terms</i> on either side of the pipe</b>           |
| <b>^</b>       | <b>Match character at start of community</b>                                  |
| <b>\$</b>      | <b>Match character at end of community</b>                                    |
| <b>[ ]</b>     | <b>Match range or array of letters or digits</b>                              |
| <b>(...),0</b> | <b>Used to group <i>terms</i>, or indicate null with no space</b>             |

Copyright © 2001, Juniper Networks, Inc.

The table on the slide shows a list of the possible regular expression operators that can be used with BGP communities. Some operators are a kind of "shorthand" for their longer equivalents. For example, the + (plus) is the same as {1,}. Both will match one or more repetitions of the **term** preceding the operator.

The square brackets ([ ]) will match a range [2-8] or array [256] of numbers. So the first regular expression in the previous sentence matches 2 through 8, and the second one matches 2 or 5 or 6.

Of special note is the use of the parentheses. Typically, the (...) operator is used to group multiple terms together in conjunction with an operator.

The parenthesis operator also has another special use. When used with no spaces in between, parentheses represent a *null* value.

## "Complex" Regex Examples

| Community pattern to match:                                                   | Regex:                      | Matches:                        |
|-------------------------------------------------------------------------------|-----------------------------|---------------------------------|
| AS is 56 or 78                                                                | "^((56) (78)):(.*)\$"       | 56:1000,<br>78:65000            |
| AS is 56 and value starts with a 2                                            | "^56:(2.*)\$"               | 56:234, 56:2,<br>56:222         |
| AS can be anything and value ends with either a 5, 7, or 9                    | "^(.*):(.?[579])\$"         | 1234:5,<br>78:2357,<br>34:65005 |
| AS is 56 or 78, value starts with a 2 and ends with any value between 2 and 8 | "^((56) (78)):(2.*[2-8])\$" | 56:22,<br>56:21197,<br>78:2678  |

Copyright © 2001, Juniper Networks, Inc.

The slide shows some examples of quite complex regular expressions that can be used to match communities in routing policies.

The first column shows the BGP Community string that the routing policy is trying to match. The second column shows the community regular expression that is used to match that pattern. The last column shows examples of values of the BGP community attribute that the regular expression will match.

In some cases the list is not exhaustive, so there are more possible communities that will match the pattern.

Note the presence of the : (colon) to separate the AS number of community-value sections of the community.

## Another Regex Example

- Does the following community regex meet the criteria?
  - The AS can be either 105, or 207, or 309
  - The value must be a 4 or 5 digit number starting with a 1
  - Or, the value must start with a 2, 5, or 6
  - Or, the value must end with a 3, 4, 7, or 9

```
"^((105) | (207) | (309)) : ((1.{3,4}) | ([256].*) | (. *[3479]))$"
```

Copyright © 2001, Juniper Networks, Inc.

The answer to the question on the slide is **yes**, the community regular expression in the slide above does in fact meet the criteria outlined. This complex regular expression utilizes multiple term-operator pairs in its definition. To better understand how the expression is operating, let's reconstruct it from the ground up.

First, we have a basic expression of `^() : ()$`. This just sets the foundation for the expression. Loosely speaking, we will be searching for an exact match of an AS number followed by a colon (:) followed by an exact match of a community value.

Next, the AS value is assigned. This AS value is actually a regular expression in itself that states the AS is either 105, 207, or 309. The pipe (|) operator is used to separate the 3 AS numbers into logical OR groupings. The extra parenthesis are used to explicitly set aside each AS number from the pipe operator. The regular expression now appears as `^((105) | (207) | (309)) : ()$`.

Now the community values can be defined. As with the AS numbers, the values can be one of three separate entities. Again, we will use the pipe operator to separate the values and the parenthesis to explicitly define the possible values. The regular expression is now `^((105) | (207) | (309)) : (( ) | ( ) | ( ))$`. Each of the possible community values in this example are individual expressions themselves. We'll examine each one in turn.

The first possible value is a 4 or 5 digit number where the first character is a 1. Since the community expressions are a character based match, the expression starts simply with a 1. The following characters in the value can be any number so the wildcard of dot (.) is used to represent that. Since the total value should be 4 or 5 digits long, the wildcard can be operated upon with a {3,4} which means at least 3 instances of any number can appear but no more than 4 instances. Combined with the character of 1 to start the value, the wildcard-operator pair makes the value a 4 or 5 digit number. The regular expression now appears as:

```
^((105)|(207)|(309)):((1.{3,4})|(0)|(0))$.
```

The second possible value can be any length but must start with either a 2, 5, or a 6. Again, the community expressions are character based, so this value should also start with a character. These are actually 3 possible characters in this single position, so the brackets are used ([256]) to signify a range of possible values. Following that, there can be more characters in the value, or there could be no characters in the value. To represent this possibility, the wildcard dot (.) is once again used. In this case, the wildcard is operated on by the asterisk (\*) which results in a .\* notation. This represents any possible value present 0 or more times. Combined with the [256], this wildcard-operator pair gives any possible value of any length as long as it starts with a 2, 5, or a 6. The regular expression now appears as:

```
^((105)|(207)|(309)):((1.{3,4})|([256].*)|(0))$.
```

The final possible value can again be any length. This time, it must end with either a 3, 4, 7, or a 9. The logic for this value is exactly the opposite of the logic for the second value so the same operators can be used. In this case, the value starts with the .\* which again represents any possible value present 0 or more times. This is followed by the bracket notation of [3479] to represent a single character in that single position. When combined, the result is any possible value of any possible length as long as it ends with a 3, 4, 7, or a 9. The regular expression now appears as:

```
^((105)|(207)|(309)):((1.{3,4})|([256].*)|(.*[3479]))$.
```



## Extended Communities

### Where we are going...

- Extended Communities
- Extended Communities Format
- Extended Communities Example

Copyright © 2001, Juniper Networks, Inc.

Subsequent slides will explore the BGP extended communities attribute and provide an example on its use..

## Extended Communities

- Regular BGP communities have four octet values
  - Usually as-number:community-value
  - Numbers limited to 0-65535 for each
- Have a larger range of eight octets
  - Format is *type:administrator:assigned-number*
  - Type: *target* or *origin* (destination or source of route)
  - Administrator: AS number or IPv4 prefix
  - Assigned-number: Used to identify the local provider
- Used for VPNs and VRF to specify uniqueness and implement routing policy
- Regular expressions are not supported for extended communities

Copyright © 2001, Juniper Networks, Inc.

There are now two types of BGP communities. The newer type is the *extended community*. The difference between them is in the format of the attribute and how it is represented in its decimal format.

Regular BGP communities have four octet (32 bit) values. This is represented as **as-number:community-value**. The numeric range for each half is 0-65535.

This might seem like enough for almost anything. But extended communities establish more fields, and more flexible use of the fields, to hold more information about a route.

Extended communities have eight octet (64 bit) values. The format is *type:administrator:assigned-number*. The type can be *target* or *origin* (the destination or source of the route) to better serve applications. The *administrator* can be an AS number or an IPv4 prefix. The *assigned-number* can be used to identify the local service provider.

Extended communities are used in virtual private networks (VPNs) within the VPN Routing and Forwarding (VRF) instances to ensure uniqueness of routing information and implement routing policies in these complex situations.

It is important to note that the use of regular expressions is not currently supported for extended communities.

## Extended Communities Format

- 2 octets for type and 6 octets for a value
- High order type octet of 0x00 specifies a value of:
  - Administrator = 2 octets; Assigned number = 4 octets
- High order type octet of 0x01 specifies a value of:
  - Administrator = 4 octets; Assigned number = 2 octets
- Low order type octet of 0x02 is a route target
  - Routers that should receive this information
- Low order type octet of 0x03 is a route origin
  - Routers that have sourced this information
- Type codes can be mixed in any order required
  - 0x0002 = target:AS #:Value (4 octet)
  - 0x0102 = target:IPv4 Prefix:Value (2 octet)
  - 0x0003 = origin:AS #:Value (4 octet)
  - 0x0103 = origin:IPv4 Prefix:Value (2 octet)

Copyright © 2001, Juniper Networks, Inc.

The extended communities format is quite complicated. Basically, there are two octets to designate the type and 6 octets for the value of the community.

The high order (first) type octet value of 0x00 means that the value has 2 octets for the administrator and 4 octets for the assigned number.

The high order (first) type octet value of 0x01 means that the value has 4 octets for the administrator and 2 octets for the assigned number.

In contrast, the low order (second) type octet value of 0x02 means that the community is a *route target*. The value that follows determines which routers should receive this community information.

The low order (second) type octet value of 0x03 means that the community is a *route origin*. The value that follows tells which routers have been the source of this community information.

The type codes can be combined in any order needed to establish the community type and values. For example:

- 0x0002: this is a target, with a 2 octet AS number and a 4 octet value.
- 0x0102: this is a target, with an IPv4 prefix of 4 octets and a 2 octet value.
- 0x0003: this is an origin, with a 2 octet AS number and a 4 octet value.
- 0x0103: this is an origin, with an IPv4 prefix of 4 octets and a 2 octet value.

## Extended Community Examples

See *draft-ramachandra-bgp-ext-communities-04.txt*

```
[edit policy-options]
community ext-test-a members [target:10458:20];
```

```
[edit policy-options]
community ext-test-b members [target:192.168.1.1:20];
```

```
[edit policy-options]
community ext-test-c members [origin:192.168.1.1:20];
```

Copyright © 2001, Juniper Networks, Inc.

The slide shows some examples of how extended communities are defined for use in a routing policy.

Within the [edit policy-options] configuration hierarchy directory, extended communities are first defined and can then be applied to a policy. With the exception of regular expressions, the extended communities can be used as a match criteria or as an action with the **add**, **delete**, and **set** commands.

In the slide above, three separate extended communities have been defined:

- **ext-test-a** is a target community that uses a 2 byte administrator field (AS number) and a 4 byte assigned-number field.
- **ext-test-b** is a target community that uses a 4 byte administrator field (IP address) and a 2 byte assigned-number field.
- **ext-test-c** is an origin community that uses a 4 byte administrator field (IP address) and a 2 byte assigned-number field.

## Review Questions

- Why are BGP communities often used to set Local Preference on a route?
- What is the scope (router?, AS?, confederation?) of these well-known BGP communities?
  - NO\_EXPORT
  - NO\_ADVERTISE
  - NO\_EXPORT\_SUBCONFED
- What is the difference between the ADD and SET community actions?
- Consider the definition `community some-routes members [65300:100 65300:120];`
  - In a routing policy, will a route have to have both communities present to produce a match?
- What is the function of the `target` and `origin` types in an extended community?

Copyright © 2001, Juniper Networks, Inc.

During this module we discussed:

- The basic function and use of the BGP Community attribute.
- The importance of the well-known BGP communities that are pre-defined.
- How BGP communities are used in real-world examples and in routing policies to perform certain key routing tasks.
- The key role played by the use of regular expressions in routing policies with regard to BGP communities.
- The function and operation of BGP extended communities.





## **Juniper Networks Advanced Policy**

**Release 5.0, Revision 1**

### ***Module 10: BGP Route Damping***

**Copyright © 2001, Juniper Networks, Inc.**

## **Objectives**

- **Explain the causes for route instability**
- **Describe the effect of damping on BGP routing**
- **Explain the default behavior of damping on links**
- **Control damping using the routing policy framework**
- **View damped routes using CLI commands**

Copyright © 2001, Juniper Networks, Inc.

This module discusses:

- The causes of route instability and route flapping.
- The effect that route damping has on BGP routes with regard to route flapping.
- The default behavior of damping on links that fail and restore "rapidly."
- How to control damping using the routing policy framework to alter and tune damping parameters.
- How to view the behavior of damped routes with CLI commands.



## Route Instabilities

- Routes can appear and disappear rapidly
- This effect is called *route flapping* or just *flapping*
- Rapid sequence of UPDATE and WITHDRAWN BGP messages
- Every BGP router that gets an UPDATE or WITHDRAWN message must propagate it to its peers
- Effects quickly cascade and affect router performance
- Consumes processing power and bandwidth

Copyright © 2001, Juniper Networks, Inc.

In any real network, routes can appear and disappear rapidly if a link fails and restores itself repeatedly in a short period of time. This is because any routes (and there could be thousands) that use the failed interface as a next hop must respond to the failure and the change in next hop must propagate to all other routers on the network.

This rapid changing of routing next hops is called *route flapping* or just *flapping* as the link "flap" up and down.

Flapping results in a rapid sequence of BGP UPDATE or WITHDRAWN messages. Recall that BGP routers must maintain separate memory tables for inbound and outbound traffic on a per-peer basis. In addition, the BGP routing protocol propagates information on an as-needed basis. These two factors make BGP unstable in the face of a flapping link.

Every BGP router that receives one of these UPDATE or WITHDRAWN messages has to send this information on to all of its BGP router peers. Much like the link-state IGPs of OSPF and ISIS, BGP must also recalculate its routing tables and databases every time a new update is received. If the new information alters the path selection process, a new route is chosen for the RIB\_LOCAL and the new route must be set downstream to all BGP peers.

The effect of route flapping quickly cascades and affects router performance. One intermittently failing link can adversely affect a whole network.

If this type of update or withdrawal occurs on a very frequent basis, valuable resources in the router such as processing power normally used to forward packets and bandwidth now needed for routing updates are consumed.

## Causes of Route Instability

- **Faulty circuits (#1)**
- **IGP instability: dynamic injection of IGP routes into BGP (static definition and aggregation helps)**
- **Human Error: Incorrect routing policy (!)**
- **Link Congestion: overloaded links drop BGP sessions**
- **In the past, some routers themselves have had problems:**
  - **Software problems: bugs, especially after upgrades**
  - **Insufficient power: busy CPU drops BGP sessions...**
  - **Insufficient memory: tables are kept in memory**
  - **Network upgrades and maintenance: adding equipment**

Copyright © 2001, Juniper Networks, Inc.

Routes in a network can flap for any number of reasons. Quite possibly, the most frequent reason for a link flap is due to a faulty circuit that is on the brink of outright failure. Any link that rapidly thinks it is okay and then fails is a potential source of route flapping.

Route flapping is not totally a BGP phenomenon. IGP's that are unstable due to faulty links can affect BGP when IGP routes are injected into BGP for advertising. BGP stability is always desirable and can be enhanced with careful use of static definitions and aggregates instead of injecting raw IGP routes into BGP.

Human error can cause route flaps as well. This can happen when an incorrectly configured routing policy causes routes to be first rejected, then accepted due to the change on the route.

Link congestion can cause route flaps if the overloaded links drop the BGP sessions that keep the BGP routes "fresh."

In the past, sometimes the routers themselves contributed to the flapping problem. Older routers were filled with software bugs (mostly after an upgrade to a new release), had insufficient power (a busy CPU in a software-based router would drop BGP sessions), and often had insufficient memory (routing tables must be kept in memory). Sometimes just adding equipment for routine network upgrades and maintenance caused route flaps.

## BGP Stability Features

- **Route damping is the main BGP mechanism to control the route flapping effects**
- **Route damping is defined in RFC 2439 "BGP Route Flap Damping" (November 1998)**
- **Damping is ignored on IBGP sessions**
  - IGP routes must be allowed to come and go so that IBGP sessions do not drop
  - The IGP supplies vital reachability information for IBGP
- **Damping applies to EBGp sessions**
  - EBGp sessions can carry thousands of routes
  - EBGp must update or withdraw these routes as required

Copyright © 2001, Juniper Networks, Inc.

Since route flapping can be so harmful to BGP, the protocol was extended to support route damping.

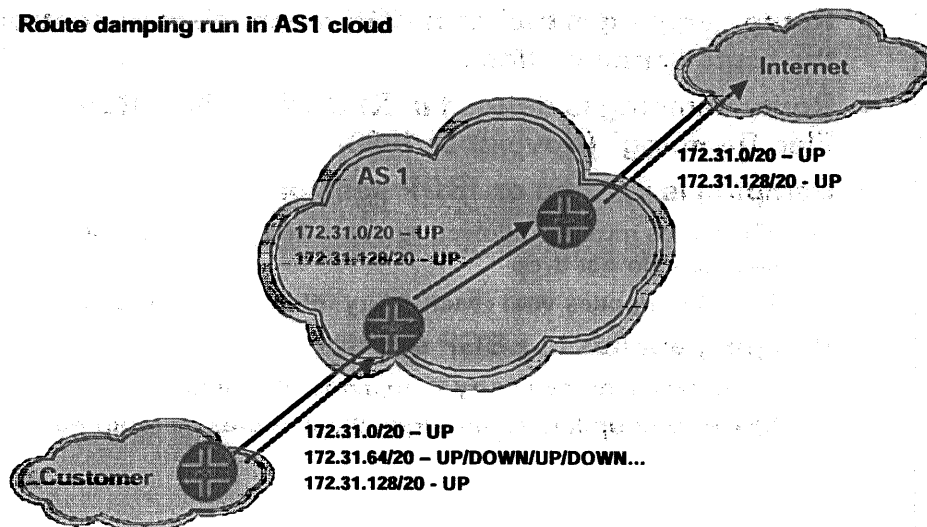
Route damping is defined in RFC 2439 "BGP Route Flap Damping" (November 1998). Sometimes the term "dampening" is seen and used.

There is a difference between how damping is applied in BGP for internal and external peers. IBGP sessions ignore damping and flap as they please. There a very good reason for this IBGP behavior. IBGP sessions usually peer to loopbacks; so IGP routes must be able to come and go so IBGP sessions always have a way to reach the loopback. Recall that BGP has no reachability information of its own and relies on the IGP to resolve next-hops.

Route damping is only applied to routes received from an EBGp peer. EBGp sessions can carry information about thousands of routes. Each EBGp session must update or withdraw these routes as required. Route damping seeks to reward route stabilities while penalizing route flapping. Once damping is enabled, the router begins to maintain a database of instability. If an EBGp received route experiences enough flaps, the local BGP process will ignore information about that route. This has the effect of **NOT** including this information in the route selection process and not advertising route changes to downstream BGP peers.

## ISPs and Route Damping

Route damping run in AS1 cloud



Copyright © 2001, Juniper Networks, Inc.

The slide shows an example of when BGP route damping is useful.

A customer of AS 1 is connected to the AS by a link running EBGP and is advertising three routes: 172.31.0.0/20, 172.31.64.0/20, and 172.31.128.0/20.

AS 1 is providing transit service for this customer to the Internet, so the three routes are re-advertised within AS 1 and further to the Internet.

But look what happens when the 172.31.64.0/20 route starts to experience stability problems causing multiple update and withdraw message to be sent to AS 1 (UP/DOWN/UP/DOWN, etc.). Without route damping enabled, this flap action would cause the router in AS 1 to send new updates to other routers in AS 1. These IBGP peers would then also send new updates to their Internet peers.

This wave of instability can be halted at the edge of AS 1 by enabling route damping. Once enabled, the edge router in AS 1 starts maintaining statistics for the routes received from the customer. Once the 172.31.64.0/20 route has been deemed unstable, the AS 1 router stops generating new update message to its IBGP peers. The IBGP peers, in turn, also have no need to send updates to the Internet. This makes the Internet as a whole more stable.

## Figure of Merit

- Type of "point value" given to a route that becomes a "penalty" if it exceeds some value (*suppress*)
- New route gets a figure of merit of 0
- Point value of given incidents:
  - 1000 points when route is withdrawn
  - 1000 points when route is re-advertised
  - 500 points when the path attribute is changed
- Points "decay" (reduce) at a certain rate (*half-life*)
- If points exceed cutoff (*suppress*) threshold, route is suppressed (must be less than or equal to *merit ceiling* value)
- When points fall below certain value (*reuse*), route is used again
- Maximum "penalty" imposed by *max-suppress* value

Copyright © 2001, Juniper Networks, Inc.

The point at which a route is deemed to be too unstable is calculated by the damping *figure of merit*. In this context, the term *figure* means a *number*, not a picture. The figure of merit is a type of "point value" given to a route. The value becomes a "penalty" if the figure of merit exceeds some predetermined value (the route is *suppressed*). It is often said that damping puts a route into a "penalty box" for a given period of time.

When a previously unknown (new) route arrives at a BGP router that has damping enabled, the new route gets assigned a figure of merit value of 0. Should the route experience any instability, the figure-of-merit gets incremented according to the following:

- As the route is withdrawn by the EBGp peer, the figure of merit is increased by a value of 1000.
- As the route is re-advertised from the EBGp peer, the figure of merit is increased by a value of 1000.
- As attributes for the route change through new update messages from the EBGp peer, the figure of merit is increased by a value of 500.

The points given to a route "decay" (reduce in value) at a certain rate, known as the *half-life*. As long as points decay faster than they accumulate, the route is not suppressed.

Should the figure of merit value increase beyond a configured cutoff (*suppress*) threshold, the route will be considered unusable and new information about the route from the EBGp peer will be ignored. The *suppress* value is configurable, but must be less than or equal to the *merit ceiling* value explained below.

The route can once again be considered usable after the figure of merit decreases below a configured threshold. The figure of merit is decreased on a time schedule set by the network administrator. Should the figure of merit not decrease below the bottom threshold in a configured amount of time, the route can automatically be usable again (*reuse*).

There is a maximum time that a route can be suppressed. This maximum is established by the configurable *max-suppress* parameter.

Also, the figure of merit can only increase to the maximum value, called the *merit ceiling*. The symbol used for the merit ceiling is  $\epsilon_c$ . This maximum value is calculated from a combination of the components listed above and is determined by the following formula:

$$\epsilon_c \leq \epsilon_r e^{(t/\lambda)(\ln 2)}$$

Where  $\epsilon_r$  is the figure of merit reuse threshold,  $t$  is the maximum suppression (hold-down) time in minutes, and  $\lambda$  is the half-life in minutes.

## Damping Parameters

- **Suppress: Cutoff (suppression) threshold**
  - Value of "penalty" when damping is initiated
  - Default value 3000 (if changed, must be less than or equal to merit ceiling value  $e_c$  or no damping will occur )
- **Reuse: reuse threshold**
  - Value "penalty" must decay to for route to be used again
  - Default threshold is 750
- **Half-life: decay half-life, in minutes**
  - Time it takes to reduce damping "penalty" by half
  - Default is 15 minutes
- **Max-suppress: Maximum hold-down time, in minutes**
  - Longest time to suppress the route activity
  - Default is 60 minutes

Copyright © 2001, Juniper Networks, Inc.

The figure of merit value interacts with the damping parameters. These parameters are listed on the slide.

The **suppress** variable is the configured threshold where a BGP route is considered instable and will not be used. This represents the value of the "penalty" that establishes the point at which damping is initiated. When this value is reached, the route is cutoff (suppressed). The default value of **suppress** is 3000. Possible values range from 1-20000. If changed, this value must be less than or equal to the merit ceiling  $e_c$ , or damping never occurs. The merit ceiling calculation was given earlier.

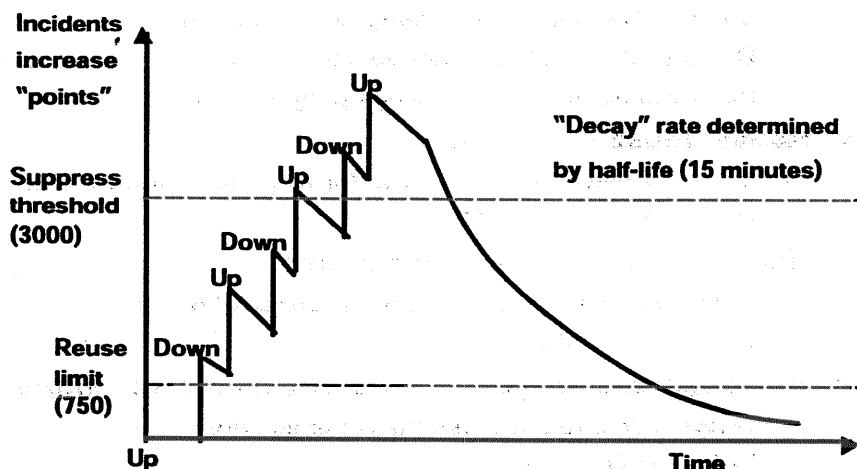
The **reuse** variable is the configured threshold where a BGP route is considered usable once again. This is the value the "penalty" must decay to for the route to go before the router will consider the route in its path selection. The default value of the **reuse** is 750. Possible values range from 1-20000.

The **half-life** variable is the rate at which the figure of merit is decreased to half its value once the value is larger than 0. The default value of the **half-life** is 15 minutes. Possible values range from 1-45 minutes.

The **max-suppress** variable is the configured maximum amount of time that a BGP route can be deemed unusable. This is the longest time that the route can be suppressed until the route is "given another chance to behave." The default value of the **max-suppress** is 60 minutes. Possible values range from 1-720 minutes. At the end of the max-suppress interval, all is forgiven and the route becomes active again.

## Figure of Merit Figure

- Exponential decay, and there is a fixed ceiling



Copyright © 2001, Juniper Networks, Inc.

The slide shows a graphic representation of the figure of merit in use for a BGP route. The default values for **suppress** (3000), **reuse** (750), and **half-life** (15 minutes) are being used. The important points here is that there is an *exponential decay* on the figure of merit, and there is a fixed ceiling on the figure of merit value (the merit ceiling).

After receiving a new BGP route, the figure of merit is 0 for some period of time. As soon as the route is withdrawn (or the link is down), the figure of merit increments to 1000. As long as the route stays down, the figure of merit decays somewhat. As the route is re-advertised, the figure of merit is incremented by another 1000. Again, the figure of merit starts to decay. Now the route is withdrawn a second time and again the figure of merit is increased. Now when the route is re-advertised, the figure of merit is increased by another 1000. This time, since not enough time has elapsed between these events in this example, the route is now over the suppress limit of 3000 and is considered unusable. In short order, the route is withdrawn and re-advertised yet again. Each time, the figure-of-merit increases 1000 for each action. Notice that the route is still damped and considered unusable, but the figure of merit is still increasing as well as decreasing even while the route is suppressed.

Whenever the figure of merit is greater than 0, the value is constantly and consistently decayed using the configured **half-life** value. The **half-life** is always configured in increments of minutes. The purpose of the **half-life** is to exponentially decay the figure of merit such that the value from any point in time is reduced by half at the end of the configured **half-life** (this half-life behavior is the essence of an exponential decay). The decay is made an exponential process so that routes can be reused as they gradually cross the **reuse** threshold and not in large groups as a timer expires. This has the effect of not overloading BGP routers with large amounts of updates at once and causing further route damping.



## Configuring Damping

- Damping parameters are defined within policy options
- Can be used as a policy action

- half-life: 1-45 minutes, default 15
- max-suppress: 1-720 minutes, default 60
- reuse: 1-20000, default 750
- suppress: 1-20000, default 3000
- disable = "do not damp"

```
[edit policy-options]
damping name {
 half-life minutes;
 max-suppress minutes;
 reuse number;
 suppress number;
 disable;
}
```

Copyright © 2001, Juniper Networks, Inc.

Once damping has been enabled within the BGP portion of the Juniper Networks router configuration, the default values introduced on previous slides will be used for figure of merit calculations.

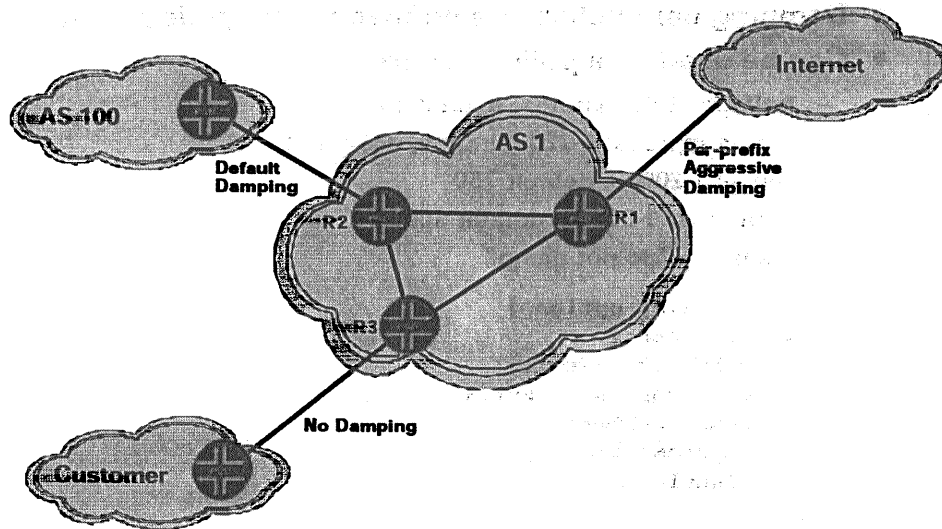
To alter those default values, a damping profile can be created and defined within the [edit policy-options] configuration hierarchy directory.

Much like the AS\_Path and community attributes, a damping profile is named and defined first. Then it can be used within a policy as an action. The five damping parameters are shown on the slide.

The presence of the **disable** keyword deserves a few words of explanation. The keyword of **disable** can be used within a damping profile to NOT have the figure of merit be calculated for certain routes. This is often useful to exempt certain routes that should never be damped and unusable. One good example of these types of routes are the root DNS servers in the Internet. Should these servers become unreachable due to damping, the ISP and its customers would experience DNS lookup failures.

For example, DNS routes could have a damping profile of **no-damping** defined that contains a single statement: **disable**.

## Damping Example (I)



Copyright © 2001, Juniper Networks, Inc.

Here is a fairly sophisticated example of route damping in action.

On the slide, AS 1 would like to enable damping for all of its EBGP peers based on the following administrative decisions:

- AS 1 would like to operate the default damping values for routes from AS 100
- AS 1 would like to NOT damp any routes from its customer
- AS 1 would like to aggressively damp all routes from the Internet except for certain prefixes

The next few slides in this sequence examine how these damping policies can be implemented.

The routing policies that can be used to accomplish these goals are outlined on the following slide. The application of those routing policies is detailed on a later slide.

## Damping Example (II)

```
[edit policy-options]
policy-statement customer-inbound {
 then damping do-not-damp;
}
policy-statement internet-inbound {
 term let-some-through {
 from {
 route-filter 192.168.10.0/24 orlonger damping do-not-damp;
 route-filter 172.16.240.0/24 orlonger damping do-not-damp;
 route-filter 172.27.32.64/26 orlonger damping do-not-damp;
 route-filter 192.168.100.0/24 orlonger damping do-not-damp;
 }
 then accept;
 }
 term damp-all-others {
 then damping aggressive-damp;
 }
}
damping do-not-damp {
 disable;
}
damping aggressive-damp {
 half-life 30;
 reuse 200;
 suppress 1500;
 max-suppress 120;
}
```

Copyright © 2001, Juniper Networks, Inc.

This slide shows all the damping profiles and policies to be used in AS 1 in the damping example.

The profile **do-not-damp** has a variable of **disable** defined.

The profile **aggressive-damp** has defined 4 variables as follows:

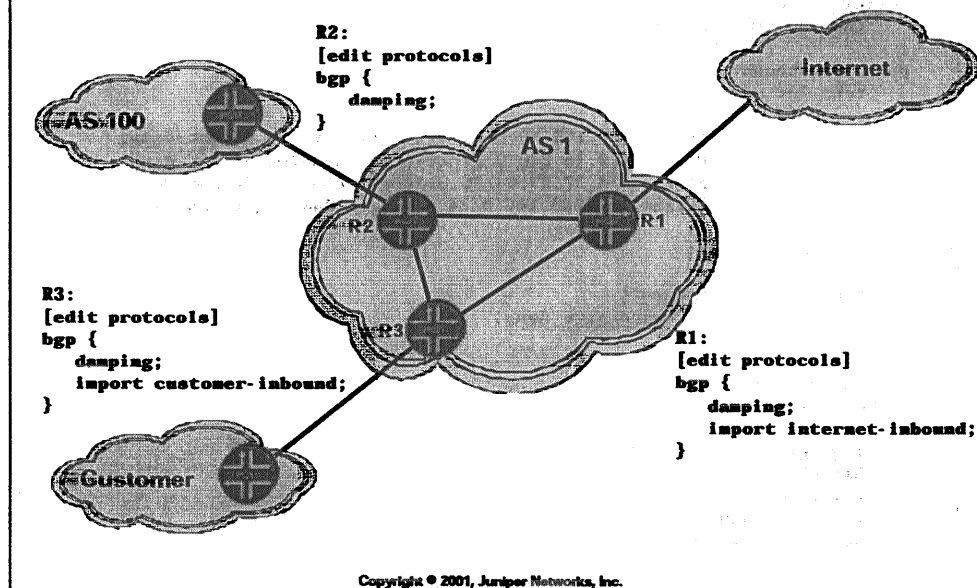
- **suppress** is 1500
- **reuse** is 200
- **half-life** is 30
- **max-suppress** is 120

A policy called **customer-inbound** has been defined with no **from** statement, so all possible routes will match the policy. The policy has an action of **damping do-not-damp**. This action sets the profile of **do-not-damp** to all routes.

A policy called **internet-inbound** has been defined with two terms. The **let-some-through** term has several **route-filter** statements with actions defined of **damping do-not-damp**. This term further lists an action of **then accept**. A second term of **damp-all-others** has no **from** statement defined, so all routes are subjected to the **damping aggressive-damp** action.

**WARNING:** This version of the **internet-inbound** policy contains a logic flaw and does not work. Please do not use this policy in your network. A corrected version will be presented on a later slide. Can you spot the flaw?

## Damping Example (III)



The routers in AS 1 are next updated to apply the policies created on the previous slide.

Router R1 defines **damping** in BGP and an import policy of **internet-inbound**. This will enable damping on the router and apply profile parameters as per the policy. As mentioned on the previous slide, the currently configured policy contains a logic flaw which will cause it to not meet the administrative requirements.

Router R2 simply defines **damping** within its BGP configuration. This will both enable damping and operate with the default parameters on routes from AS 100. No policy is needed on this router.

Router R3 defines **damping** within BGP and an import policy of **customer-inbound**. This will enable damping on the router and apply profile parameters as per the policy.

## Damping Example (IV)

```
[edit policy-options]
policy-statement internet-inbound {
 term let-some-through {
 from {
 route-filter 192.168.10.0/24 orlonger;
 route-filter 172.16.240.0/24 orlonger;
 route-filter 172.27.32.64/26 orlonger;
 route-filter 192.168.100.0/24 orlonger;
 }
 then {
 damping do-not-damp;
 accept;
 }
 }
 term damp-all-others {
 then damping aggressive-damp;
 }
}
```

Copyright © 2001, Juniper Networks, Inc.

This slide shows a corrected version of the **internet-inbound** policy. As mentioned before, previous versions contained a logic flaw. The flaw is a result of the way that *immediate actions* are applied to a route filter.

The issue is the **do-not-damp** action defined for the **route-filter** statements. When a candidate route that matches one of the filters appears, the action of **damping do-not-damp** is taken. Since this action is defined within the **route-filter** statement itself (it is an immediate action), any *further* actions on the route specified within a **then** statement are skipped. In this case, the **then accept** is skipped within the **let-some-through** term. But the **route-filter** statements do not also define a terminating action (accept or reject). So the damping profile is applied, but the route is passed to the next term in the policy for further evaluation. When the route enters the **damp-all-others** term, the route matches the term since no **from** statement is defined. The route is then applied the damping profile of **aggressive-damp**. This will cause the specified "exempt" routes to actually be damped at the aggressive rate! This violates the administrative policies of the AS.

To correct the policy, the **damping do-not-damp** actions are removed from the individual **route-filter** statements and are instead placed within the **then** portion of the term. After making this change, the exempt routes will match the **let-some-through** term, will have the correct damping profile applied, and will be accepted.

## Show Route Damping History

**show route damping history** displays routes that have been withdrawn and have a figure of merit

```
user@host> show route damping history extensive
inet.0: 15 destinations, 15 routes (14 active, 0 holddown, 1 hidden)
+ = Active Route, - = Last Active, * = Both

200.200.200.0/24 (1 entry, 0 announced)
 BGP Preference: /-101
 Nexthop: 172.16.10.1 via fe-0/0/0.0, selected
 State: <Hidden Ext>
 Local AS: 2 Peer AS: 1
 AS path: 1 I
 Localpref: 100
 Router ID: 192.168.1.1
 Merit (last update/now): 2777/2454
 Default damping parameters used
 Last update: 00:02:45
 First update: 00:04:35
 Flaps: 3
 History entry. Expires in: 00:54:20
```

Copyright © 2001, Juniper Networks, Inc.

The slide shows the output from the **show route damping history** command.

Any routes displayed by this command have been withdrawn from the router. However, the router is retaining a record of these routes should they be re-advertised to the local router. Some things to note in the display above are:

- The route is currently hidden. This can be seen in both the **State: <Hidden Ext>** field as well as the **Preference: /-101** field. Notice that there is no JUNOS software protocol preference value defined.
- There is a field (**Merit**) for the current figure of merit value. The two values that follow describe what the value was after the last BGP update (or withdrawal) and what the current value is after experiencing some decay. For this route the values are **Merit: 2777/2454**. So the value at last update/withdrawal was 2777 (note this value need not necessarily exceed the default suppress threshold of 3000) and the current value is 2454.
- The default parameters are being used (**Default damping parameters used**). If this route had been evaluated by a policy with a **damping** action, the new damping profile name would appear in the output.

## Show Route Damping Decayed

**show route damping decayed** displays routes that are active and have a figure of merit

```
user@host> show route damping decayed extensive
inet.0: 15 destinations, 15 routes (15 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

200.200.200.0/24 (1 entry, 1 announced)
 *BGP Preference: 170/-101
 Nexthop: 172.16.10.1 via fe-0/0/0.0, selected
 State: <Active Ext>
 Local AS: 2 Peer AS: 1
 AS path: 1 I
 Localpref: 100
 Router ID: 192.168.1.1
 Merit (last update/now): 2000/1954
 Default damping parameters used
 Last update: 00:00:35
 First update: 00:00:40
 Flaps: 2
```

Copyright © 2001, Juniper Networks, Inc.

The slide shows the output from the **show route damping decayed** command.

Any routes displayed by this command have been advertised to the router and are currently usable routes, but these routes have a figure of merit greater than 0. Some things to note in the display above are:

- The route is currently active. This can be seen both by the asterisk (\*) in the output as well as the State: <Active Ext> field.
- There is a field (Merit) for the current figure-of-merit value. The two values describe what the value was after the last update (or withdrawal) and what the current value is after experiencing some decay. For this route the values are Merit: 2000/1954.
- The default parameters are being used (Default damping parameters used). If this route had been evaluated by a policy with a **damping** action, the new damping profile name would appear in the output.

## Show Route Damping Suppressed

- **show route damping suppressed** displays routes that have been damped due to the figure of merit
- **clear bgp damping**
  - Suppressed routes can have the figure of merit reduced to 0
  - All routes (as well as individual routes) can be cleared

```
user@host> show route damping suppressed
inet.0: 15 destinations, 15 routes (14 active, 0 holddown, 1 hidden)
+ = Active Route, - = Last Active, * = Both

200.200.200.0/24 [BGP] 00:01:21, localpref 100
 AS path: 1 I
 > to 172.16.10.1 via fe-0/0/0.0

user@host> clear bgp damping
inet.0: 15 destinations, 15 routes (15 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both
```

Copyright © 2001, Juniper Networks, Inc.

The slide shows the output from the **show route damping suppressed** command.

Any routes displayed by this command have been advertised to the router, but these routes have a figure of merit that is currently above the **suppress** threshold and the route is unusable.

The route will remain in this state until the figure of merit crosses below the **reuse** threshold. A route can have the figure of merit reduced to 0 administratively by using the **clear bgp damping** command.

In the slide, the route 200.200.200.0/24 is currently suppressed and hidden. After the **clear bgp damping** command is issued, the route is no longer hidden.



## Review Questions

- Why is route flapping especially bad for EBGP?
- Why is damping ignored on IBGP routes?
- What is the *half-life* as this term applies to route damping?
- Why is the decay function set as an exponential decay rather than as a simple timeout?
- What is the function of the **max - suppress** parameter?
- What happens if the **suppress** threshold is set *higher* than the merit ceiling?
- What is the routing protocol preference of a hidden damped route?
- What kind of routes are shown with the **show route damping decayed** command?

Copyright © 2001, Juniper Networks, Inc.

During this module we discussed:

- The causes of route instability and route flapping.
- The effect that route damping had on BGP routes with regard to route flapping.
- The default behavior of damping on links that fail and restore "rapidly."
- How to control damping through the use of the routing policy framework to alter and tune damping parameters.
- How to view the behavior of damped routes with CLI commands.

