

Chapter 10

Simplified Interinstance Route Sharing

When support for multiple virtual private network (VPN) routing and forwarding (VRF) instances was added to the JUNOS software, the import and export of routes to other instances and routing protocols from individual instances caused an issue. Interinstance route sharing required configuration of routing table groups (sometimes referred to as routing information base [RIB] groups) for every routing instance that exported routes to other tables.

Policy-based interinstance export in JUNOS Release 5.4 and later simplifies the configuration requirements for users, maintains existing functionality, and, when possible, eliminates the use of routing table groups. This chapter explores in detail the streamlined configuration hierarchy that has been created for interinstance route sharing.

This feature guide covers these topics:

- Overview on page 418

- System Requirements on page 419

- Terms and Acronyms on page 419

- Simplified Interinstance Configuration on page 419

- Instance Export Using an IGP Export Policy on page 422

- Configuring Overlapping VPNs on page 423

 - Example: Overlapping VPNs Configuration on page 426

 - Checking Your Work on page 434

- Configuring Nonforwarding Instances on page 437

 - Example: Nonforwarding Instances Configuration on page 439

 - Checking Your Work on page 443

- For More Information on page 444

- Revision History on page 445

Overview

In JUNOS Release 5.3 and earlier, interinstance route sharing often required configuration of routing table groups by means of the `rib-group` statement. Although these configurations performed well, the routing table group technique had several limitations:

Lack of intuitiveness—A routing table group is an unfamiliar configuration construct for many users.

Complex configuration requirements—Routing table groups specify a primary import routing table that must match the routing table of the VRF instance on which they are applied. Thus, a different routing table group is defined for each of the instances that participate in interinstance route export.

Redundancy—The information imported and exported by the routing table groups is already present in the router or can be deduced from most configurations (for example, overlapping VPNs).

Per-protocol configuration—Routing table groups must be applied to every protocol containing routes designated for export.

There are two typical situations in which interinstance export is used:

Overlapping VPNs—VPN configurations where more than one VRF instance lists the same community route target in a `vrf-import` policy. In this case, the use of routing table groups is particularly tricky. Incoming routes from other provider edge (PE) routers are automatically imported according to the community route targets, but local VRFs require additional configuration.

Nonforwarding instances—Multilevel interior gateway protocols (IGPs) that have multiple routing instances and perform route-sharing through interinstance route export. The IGP export policy contains the specific instances that are permitted to advertise routes.

These two scenarios differ in the way that policy clauses are specified (route targets in the VRF case; instances in the IGP routing instance case), but are similar in that import and export routes can be deduced by examining the policy configuration. In this guide, you can learn about new hierarchy statements that simplify interinstance route sharing, such as `auto-export`, `instance-import`, and `instance-export`.

System Requirements

To implement simplified interinstance route sharing, your system must meet these minimum requirements:

JUNOS Release 5.4 or later

Four Juniper Networks M-series or T-series routing platforms for the overlapping VPNs example: Two routers act as PE routers and two act as customer edge (CE) routers

Six Juniper Networks M-series or T-series routing platforms for the nonforwarding instance example: Two act as PE routers and four act as CE routers

Terms and Acronyms

VPN routing and forwarding (VRF) instance—A private routing table created for an individual VPN customer. For more information about VRFs, see the *JUNOS VPNs Configuration Guide*.

RIB group—A routing table group. The group is a master routing table for individual routing tables and stores information about routes that are shared between instances.

Simplified Interinstance Configuration

By changing the configuration format of interinstance export policies, JUNOS Release 5.4 software makes it easier to share routes between VRF instances, other types of instances (such as nonforwarding instances), and IGPs.

VRF instances can share routes with the statement `auto-export`. When you configure `auto-export`, the `vrf-import` and `vrf-export` policies are compared across all VRF instances. If there is a common route target community between the instances, the routes are shared.

For VRF instances, such as overlapping VPNs, the basic hierarchy levels for `auto-export` are as follows:

```
[edit]
routing-instances {
  instance-name {
    routing-options {
      auto-export;
    }
  }
}
```

For nonforwarding instances, routes are imported into the instance so routing protocols can announce them. For more information, see “Configuring Nonforwarding Instances” on page 437.

A third option for interinstance export is using an Interior Gateway Protocol (IGP), such as Open Shortest Path First (OSPF) or Intermediate System-to-Intermediate System (IS-IS). An example of the hierarchy used for IGP import and export is listed below.

```
[edit]
routing-options {
  instance-import;
  instance-export;
  auto-export {
    (enable | disable);
    family inet {
      unicast {
        (enable | disable);
        rib-group;
      }
      multicast {
        (enable | disable);
        rib-group;
      }
    }
  }
}
```

When configuring interinstance route sharing, keep this information in mind:

The `instance-import` and `instance-export` commands cannot be used with VRF instances. They are equivalent to the `vrf-import` and `vrf-export` VRF-specific commands.

Traceoptions in the master instance apply to the routing table export task. Consequently, such traceoptions are propagated to all other instances, although they can be modified as needed.

The `auto-export` statement applies to VRF and non-VRF instances.

Use of the command `instance-import` automatically enables `auto-export` for non-VRF instances.

Some network administrators use the `instance-import` functionality to create communities of interest. By setting up different VPNs and sharing routes between instances as needed, administrators can tailor services to the needs of their customers. For an example, see “Configuring Nonforwarding Instances” on page 437.

To save time when configuring interinstance parameters on multiple instances, you can configure auto-export to be the default behavior for all your routing instances by means of a configuration group.

```
[edit]
groups {
  vrf-export-on {
    routing-instances {
      routing-options {
        auto-export;
      }
    }
  }
}
```

At the desired hierarchy level, you apply the configuration group and all members of the group receive the same policy:

```
apply-groups vrf-export-on;
```

Instance Export Using an IGP Export Policy

Current configurations that use routing table groups define a policy with a `from instance` statement to select routes in an IGP export policy. However, no policy controls the export process itself. Therefore, the configuration has been simplified so that you do not need to specify additional policies to control the export process. Many current interinstance implementations use an IGP export policy model. The policy model has been extended to support both interinstance route export and IGP export.

If the `[edit routing-options auto-export]` hierarchy is enabled when an instance-import policy has not been defined, OSPF and IS-IS export policies are automatically examined for the presence of `from instance` statements. If these statements are present, the instance-import policy is selected.

This verification process prevents attribute changes from being applied twice. It also prevents the second policy (IGP export policy) from causing conflicting routing choices. The following is an example of a configuration using an IGP export policy:

```
[edit]
policy-options {
  policy-statement {
    red-ospf-export {
      from instance blue;
      then {
        tag 1;
        accept;
      }
    }
  }
}
routing-instances {
  red {
    routing-options {
      auto-export;
    }
    protocols ospf {
      export red-ospf-export;
    }
  }
  blue {
  }...
}
```

Configuring Overlapping VPNs

Policy-based instance export automatically exports routes between VRF instances that refer to the same route target community. If this feature is enabled, a VRF-target tree is constructed by examining the vrf-import and vrf-export policies configured on the system. When an instance refers to a given target in its vrf-import policy, this instance is added to the import list of the target. Similarly, if the instance refers to a specific route target in its vrf-export policy, the instance is added to the export list for that target. Route targets that contain a single importer that matches a single exporter, or that lack importers and exporters altogether, are ignored by the router when policies are evaluated.

The “rt-export” module tracks changes in routing tables that export a specified route target. When routing changes occur, the vpn-export policy of an instance is applied to the route. Also, if allowed, the route will be imported to all the import tables (subject to vrf-import policy) of the route targets set by the export policy.

The auto-export statement is particularly useful for configuring overlapping VPNs. The auto-export statement determines which routing tables to export routes from and import routes to by examining the existing policy configuration, which can include vrf-target configuration. (For more information on the vrf-target statement, see the *JUNOS VPNs Configuration Guide*.)

When you use the auto-export statement, the behavior varies significantly from the behavior of the rib-groups statement. With the auto-export statement, only the primary route from the originating routing table is exported. In addition, routes exported from the originating VRF to another on the same PE router honor the export policy changes to route attributes. As a result, you must add each originating route target to the exported routes when you use the auto-export statement.

The next example configuration uses a Border Gateway Protocol (BGP) session between a PE and a CE router. It shows the configuration changes required when you use the autoexport feature. Text marked in italics indicates the rib-group statements from JUNOS Release 5.4 and earlier that can be omitted, while bold text highlights the new, simplified style of configuration.

```
[edit]
## routing-options {
## rib-groups {
##   vpna-vpnab {
##     import-rib [VPN-A.inet.0 VPN-AB.inet.0];
##   }
##   vpnab-vpna_and_vpnb {
##     import-rib [VPN-AB.inet.0 VPN-A.inet.0 VPN-B.inet.0];
##   }
## }
##}
```

```

routing-instances {
  VPN-A {
    instance-type vrf;
    interface fe-1/0/0.0;
    route-distinguisher 10.255.255.175:3;
    vrf-import vpna-import;
    vrf-export vpna-export;
    routing-options {                                # New method
      auto-export;                                    # New method
    }
  }
  protocols {
    bgp {
      group vpna-site1 {
        family inet {                                # Old method
        unicast {                                     # Old method
        rib-group vpna-vpnab;                         # Old method
        }
      }
      peer-as 1;
      neighbor 192.255.197.141;
    }
  }
}
VPN-AB {
  instance-type vrf;
  interface fe-1/1/0.0;
  route-distinguisher 10.255.255.175:9;
  vrf-import vpnab-import;
  vrf-export vpnab-export;
  routing-options {                                # New method
    auto-export;                                    # New method
  }
  protocols {
    bgp {
      group vpnab-site1 {
        family inet {                                # Old method
        unicast {                                     # Old method
        rib-group vpnab-vpna_and_vpnab;             # Old method
        }
      }
      peer-as 9;
      neighbor 192.255.197.178;
    }
  }
}
}

```

In some overlapping VPN cases, additional configuration information is required:

When `vrf-import` and `vrf-export` policies are configured on a per-instance basis, you must enable or disable the policies individually for unicast or multicast, especially when a multicast network layer reachability information (NLRI) is configured.

When you use `auto-export` between overlapping VPNs and require a subset of the routes learned from an instance to be installed into the `inet.0` or `instance.inet.2` routing tables, you must install the routes with additional configuration statements.

To support scenarios in which not all the required information is present in the `vrf-import` and `vrf-export` policies, you can configure additional routing tables with a routing table group. For example, if you wish to add routes from VPN A and VPN AB to the `inet.0` routing table, the following additional configuration parameters are required:

```
[edit]
routing-options {
  rib-groups {
    inet-access {
      import-rib inet.0;
    }
  }
}
routing-instances {
  VPN-A {
    routing-options {
      auto-export {
        family inet {
          unicast {
            rib-group inet-access;
          }
        }
      }
    }
  }
  VPN-AB {
    routing-options {
      auto-export {
        family inet {
          unicast {
            rib-group inet-access;
          }
        }
      }
    }
  }
}
```

There is a significant difference in how routing table groups are used in this case and how they are used more generally. Typically, routing table groups require the exporting routing table to be referenced as the primary import routing table in the `rib-group` configuration. In this case, the restriction is lifted and the routing table group functions as an additional list of tables that export routes.



NOTE: When upgrading to JUNOS Release 5.4 or later, be aware that route export behavior differs when using the auto-export command instead of rib-group export:

When routes are exported between routing tables by using the rib-group statements, both primary routes (routes in the originating routing table) and secondary routes (routes imported from other routing tables) are exported to the remote PE routers. When the auto-export statement is used, only the primary routes from the originating routing table are exported.

Routes exported from an originating VRF instance to another on the same PE now honor export policy changes to route attributes. When you use auto-export, you must add the originating route target(s) to the exported routes. With rib-group statements, no additional configuration is necessary.

Example: Overlapping VPNs Configuration

Figure 40: Overlapping VPNs Topology Diagram

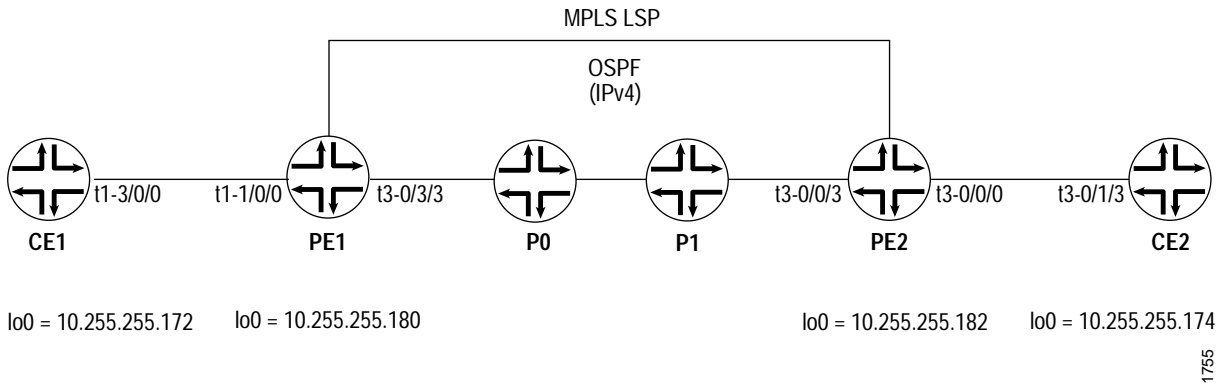


Figure 40 shows a standard Multiprotocol Label Switching (MPLS) VPN topology. Routers PE1 and PE2 are acting as PE routers, CE1 and CE2 are CE routers, and P0 and P1 are core provider routers. You will establish three VRF instances: A, B, and AB. You will also configure auto-export as the method of sharing routing information between instances.

This example focuses on the interinstance and policy statements. As a result, some information has been omitted.

Because PE1 uses static routing instances, the router configuration for CE1 is not included in this example.

Most routers display a minimal configuration. Interface addresses and loopback addresses are assumed to have been enabled properly.

For more information about VPNs, see the *JUNOS VPNs Configuration Guide*.

Routers PE1 and PE2 contain the bulk of the configuration. At PE1, initiate an IBGP connection to PE2 and open a VPN connection to CE Router CE1 through three VRF instances: A, B, and AB.

The auto-export policy is applied to all instances simultaneously by means of a configuration group. Another method of enabling this option is to configure auto-export individually on each VRF instance.

Finally, the policy statements add the appropriate communities to each instance and accept traffic coming from the desired community. For example, the policy for VRF A sets community A on all outbound traffic leaving the instance, and only accepts traffic from PE2 that is tagged with community A.

```
Router PE1 [edit]
groups {
  vrf-export on {
    routing-instances {
      <*> {
        routing-options {
          auto-export;
        }
      }
    }
  }
}
interfaces {
  t1-0/0/0
    description "to vpn02 t1-3/0/0";
    dce;
    encapsulation frame-relay;
    unit 0 {
      dlsi 100;
      family inet {
        address 192.255.197.38/30;
      }
    }
    unit 1 {
      dlsi 101;
      family inet {
        address 10.3.0.1/30;
      }
    }
    unit 2 {
      dlsi 102;
      family inet {
        address 10.3.0.5/30;
      }
    }
  }
}
lo0
  unit 0
    family inet {
      address 10.255.255.180/32;
    }
}
}
```

```
protocols {
  mpls {
    interface all;
  }
  bgp {
    group pepe {
      type internal;
      neighbor 10.255.255.182 {
        family inet-vpn {
          unicast;
        }
      }
    }
  }
  ospf {
    area 0.0.0.0 {
      interface t3-0/3/3.0;
      interface lo0.0 {
        passive;
      }
    }
  }
  ldp {
    interface all;
  }
}
```

```
policy-options {  
  policy-statement A-in {  
    from community A;  
    then accept;  
  }  
  policy-statement A-out {  
    then {  
      community add A;  
      accept;  
    }  
  }  
  policy-statement B-in {  
    from community B;  
    then accept;  
  }  
  policy-statement B-out {  
    then {  
      community add B;  
      accept;  
    }  
  }  
  policy-statement AB-in {  
    from community [A B];  
    then accept;  
  }  
  policy-statement AB-out {  
    then {  
      community add A;  
      community add B;  
      accept;  
    }  
  }  
  community A members target:69:1;  
  community B members target:69:2;  
}
```

```

routing-instances {
  apply-groups vrf-export-on;
  A {
    instance-type vrf;
    interface t1-0/0/0.0;
    route-distinguisher 10.255.255.180:69;
    vrf-import A-in;
    vrf-export A-out;
    routing-options {
      static {
        route 1.1.1.1/32 next-hop t1-0/0/0.0;
        route 1.1.1.2/32 next-hop t1-0/0/0.0;
      }
    }
  }
  AB {
    instance-type vrf;
    interface t1-0/0/0.2;
    route-distinguisher 10.255.255.180:69;
    vrf-import AB-in;
    vrf-export AB-out;
    routing-options {
      static {
        route 1.1.3.1/32 next-hop t1-0/0/0.2;
        route 1.1.3.2/32 next-hop t1-0/0/0.2;
      }
    }
  }
  B {
    instance-type vrf;
    interface t1-0/0/0.1;
    route-distinguisher 10.255.255.180:69;
    vrf-import B-in;
    vrf-export B-out;
    routing-options {
      static {
        route 1.1.2.1/32 next-hop t1-0/0/0.1;
        route 1.1.2.2/32 next-hop t1-0/0/0.1;
      }
    }
  }
}

```

As a provider core transit router, Router P0 only needs to provide connectivity to the PE routers. You configure OSPF, MPLS, and LDP on the interfaces pointing to both PE routers.

```

Router P0 [edit]
  protocols {
    mpls {
      interface all;
    }
    ospf {
      area 0.0.0.0 {
        interface t3-0/0/3.0;
        interface t1-0/1/1.0;
      }
    }
    ldp {
      interface all;
    }
  }

```

Like Router P0, Router P1 also needs to provide basic core connectivity for the PE routers. You can configure OSPF, MPLS, and LDP on the interfaces pointing toward routers P0 and PE2.

```

Router P1 [edit]
  protocols {
    mpls {
      interface all;
    }
    ospf {
      area 0.0.0.0 {
        interface t1-0/1/1.0;
        interface t3-0/0/3.0;
      }
    }
    ldp {
      interface all;
    }
  }

```

At Router PE2, complete your IBGP connection to PE1 and finish the VPN connection to CE Router CE2 through VRF instance AB. The VRF import policy named AB-in is the same as the export policy used for the OSPF protocol in the AB VRF instance. The policy statements add communities A and B to all outbound routes and accept any traffic coming from these communities.

```

Router PE2 [edit]
interfaces {
  lo0
    unit 0
      family inet {
        address 10.255.255.182/32;
      }
    }
}
protocols {
  mpls {
    interface all;
  }
  bgp {
    keep all;
    group pepe {
      type internal;
      neighbor 10.255.255.180 {
        family inet-vpn {
          unicast;
        }
      }
    }
  }
  ospf {
    area 0.0.0.0 {
      interface t3-0/0/3.0;
      interface lo0.0 {
        passive;
      }
    }
  }
  ldp {
    interface all;
  }
}

```

```

policy-options {
  policy-statement AB-in {
    from community [A B];
    then accept;
  }
  policy-statement AB-out {
    then {
      community add A;
      community add B;
      accept;
    }
  }
  community A members target:69:1;
  community B members target:69:2;
}
routing-instances {
  AB {
    instance-type vrf;
    interface t3-0/0/0.0;
    route-distinguisher 10.255.255.182:69;
    vrf-import AB-in;
    vrf-export AB-out;
    protocols {
      ospf {
        export AB-in;
        area 0.0.0.0 {
          interface all;
        }
      }
    }
  }
}

```

At Router CE2, advertise the 10.255.255.174 loopback address into the VPN. Look for this route when you check the routing tables for the A, B, and AB instances on Router PE1. If the route appears in these instances, interinstance route sharing is successful.

```

Router CE2 [edit]
interfaces {
  lo0
    unit 0
      family inet {
        address 10.255.255.174/32;
      }
    }
}
protocols {
  ospf {
    area 0.0.0.0 {
      interface t3-0/1/3.0;
      interface lo0.0;
    }
  }
}

```

Checking Your Work

To verify that your overlapping VPN configuration is functioning properly, use the following commands:

```
show route export table table-name (brief | detail)
```

```
show route export instance instance-name (brief | detail)
```

```
show route export vrf-target (community community-regular-expression) (brief | detail)
```

The following section shows the output of these commands as used with the configuration example.

Router PE1 Status

```
user@PE1> show route export
```

Table	Export	Routes
A.inet.0	Y	4
AB.inet.0	Y	4
B.inet.0	Y	4

```
user@PE1> show route export detail
```

```
A.inet.0                               Routes: 4
  Flags: <vrf>
AB.inet.0                               Routes: 4
  Flags: <vrf>
B.inet.0                               Routes: 4
  Flags: <vrf>
```

```
user@PE1> show route export instance detail
```

```
Instance: A                            Type: vrf
  Flags: <config> Options: <unicast multicast>
Instance: AB                           Type: vrf
  Flags: <config> Options: <unicast multicast>
Instance: B                            Type: vrf
  Flags: <config> Options: <unicast multicast>
```

```
user@PE1> show route table A.inet.0
```

```
A.inet.0: 10 destinations, 10 routes (10 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both
```

```
1.1.1.1/32    *[Static/5] 02:08:14
> via t1-0/0/0.0
1.1.1.2/32    *[Static/5] 02:08:14
> via t1-0/0/0.0
1.1.3.1/32    *[Static/5] 02:08:14
> via t1-0/0/0.2
1.1.3.2/32    *[Static/5] 02:08:14
> via t1-0/0/0.2
10.3.0.4/30   *[Direct/0] 02:08:14
> via t1-0/0/0.2
10.3.0.5/32   *[Local/0] 02:08:14
Local via t1-0/0/0.2
10.255.255.174/32 *[BGP/170] 00:18:08, MED 2, localpref 100, from 10.255.255.182
AS path: I
> via t3-0/3/3.0, Push 100004, Push 100017(top)
192.255.197.36/30 *[Direct/0] 02:08:14
> via t1-0/0/0.0
192.255.197.38/32 *[Local/0] 02:08:14
Local via t1-0/0/0.0
192.255.197.248/30 *[BGP/170] 00:18:18, localpref 100, from 10.255.255.182
AS path: I
> via t3-0/3/3.0, Push 100003, Push 100017(top)
```

```
user@PE1> show route table B.inet.0
```

```
B.inet.0: 10 destinations, 10 routes (10 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both
```

```
1.1.2.1/32    *[Static/5] 02:09:28
> via t1-0/0/0.1
1.1.2.2/32    *[Static/5] 02:09:28
> via t1-0/0/0.1
1.1.3.1/32    *[Static/5] 02:09:28
> via t1-0/0/0.2
1.1.3.2/32    *[Static/5] 02:09:28
> via t1-0/0/0.2
10.3.0.0/30   *[Direct/0] 02:09:28
> via t1-0/0/0.1
10.3.0.1/32   *[Local/0] 02:09:28
Local via t1-0/0/0.1
10.3.0.4/30   *[Direct/0] 02:09:28
> via t1-0/0/0.2
10.3.0.5/32   *[Local/0] 02:09:28
Local via t1-0/0/0.2
10.255.255.174/32 *[BGP/170] 00:19:22, MED 2, localpref 100, from 10.255.255.182
AS path: I
> via t3-0/3/3.0, Push 100004, Push 100017(top)
192.255.197.248/30 *[BGP/170] 00:19:32, localpref 100, from 10.255.255.182
AS path: I
> via t3-0/3/3.0, Push 100003, Push 100017(top)
```

```
user@PE1> show route table AB.inet.0
```

```
AB.inet.0: 14 destinations, 14 routes (14 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both
```

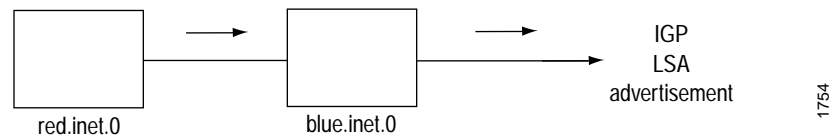
```
1.1.1.1/32    *[Static/5] 02:09:43
> via t1-0/0/0.0
1.1.1.2/32    *[Static/5] 02:09:43
> via t1-0/0/0.0
1.1.2.1/32    *[Static/5] 02:09:43
> via t1-0/0/0.1
1.1.2.2/32    *[Static/5] 02:09:43
> via t1-0/0/0.1
1.1.3.1/32    *[Static/5] 02:09:43
> via t1-0/0/0.2
1.1.3.2/32    *[Static/5] 02:09:43
> via t1-0/0/0.2
10.3.0.0/30   *[Direct/0] 02:09:43
> via t1-0/0/0.1
10.3.0.1/32   *[Local/0] 02:09:43
Local via t1-0/0/0.1
10.3.0.4/30   *[Direct/0] 02:09:43
> via t1-0/0/0.2
10.3.0.5/32   *[Local/0] 02:09:43
Local via t1-0/0/0.2
10.255.255.174/32 *[BGP/170] 00:19:37, MED 2, localpref 100, from 10.255.255.182
AS path: I
> via t3-0/3/3.0, Push 100004, Push 100017(top)
192.255.197.36/30 *[Direct/0] 02:09:43
> via t1-0/0/0.0
192.255.197.38/32 *[Local/0] 02:09:43
Local via t1-0/0/0.0
192.255.197.248/30 *[BGP/170] 00:19:47, localpref 100, from 10.255.255.182
AS path: I
> via t3-0/3/3.0, Push 100003, Push 100017(top)
```

```
user@PE1> show route export vrf-target detail
Target: 69:1          inet unicast
Import table(s): A.inet.0 AB.inet.0
Export table(s): A.inet.0 AB.inet.0
Target: 69:2          inet unicast
Import table(s): AB.inet.0 B.inet.0
Export table(s): AB.inet.0 B.inet.0
```

Configuring Nonforwarding Instances

In nonforwarding instances implemented in JUNOS Release 5.3 and earlier, you could configure interinstance export through use of import routing table groups. A secondary routing instance would import routes from the primary routing instance. Then, IGP would advertise the routes received from the second instance table as shown in the example in Figure 41.

Figure 41: Nonforwarding Instance Concept



In JUNOS Release 5.4 and later, you can use the `instance-import` and `instance-export` policy keywords to perform nonforwarding, interinstance route sharing. The keywords are assigned at the [edit routing-instances *instance-name* routing-options] hierarchy level. These statements are similar to VRF import and VRF export policies used for VRF instances.

The “rt-export” module examines the `from instance` statements present in an instance import policy to construct the list of import tables for a particular exporting instance. The following example illustrates the configuration hierarchy for this feature:

```

[edit]
policy-options {
  policy-statement {
    red-import {
      from instance blue;
      then {
        tag 1;
        accept;
      }
    }
    blue-import {
      from instance red;
      then {
        tag 2;
        accept;
      }
    }
  }
}

```

```

routing-instances {
  red {
    routing-options {
      instance-import red-import;
    }
  }
  blue {
    routing-options {
      instance-import blue-import;
    }
  }
}

```

To advertise instance blue routes through an instance red IGP such as OSPF, you would add an export policy to OSPF to advertise routes from the local table.

```

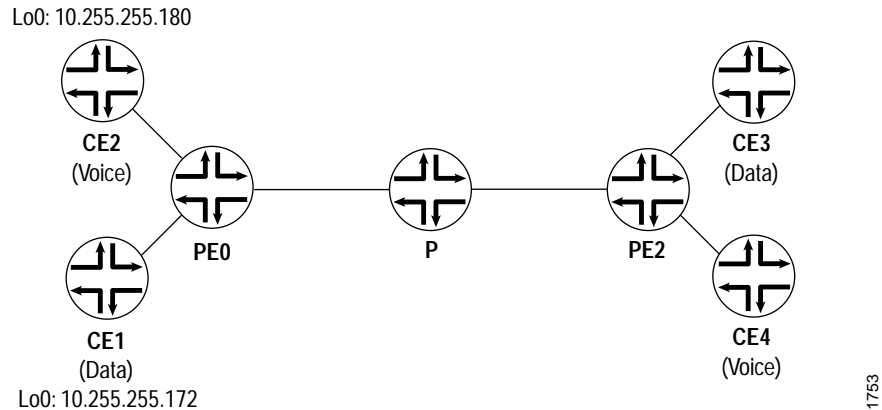
[edit]
policy-options {
  policy-statement ospf-export {
    from /* some criteria */
    then accept;
  }
}
routing-instances {
  red {
    protocols ospf {
      export ospf-export;
    }
  }
}

```

When an instance import policy is configured, the policy is allowed to modify route attributes other than next-hop.

Example: Nonforwarding Instances Configuration

Figure 42: Nonforwarding Instances Topology Diagram



In Figure 42, routers CE1, CE2, CE3, and CE4 are CE routers, PE0 and PE2 are PE routers, and Router P is the provider core transit router. CE1 and CE3 are part of a “community of interest” group called *data*, while CE2 and CE4 belong to a group called *voice*. Your goal is to connect the members of each group to each other by using a nonforwarding instance at the PE routers.

Note that routers PE0, CE1, and CE2 mirror the configurations on PE2, CE3, and CE4, respectively. Therefore, the latter routers are not shown in this example. The loopback addressing scheme for this network is shown in Table 34.

Table 34: Non-Forwarding Instances—Loopback Addresses

Router	Loopback Address
CE1	10.255.255.172
CE2	10.255.255.180
PE0	10.255.255.176
P	10.255.255.178
PE2	10.255.255.174
CE3	10.255.255.182
CE4	10.255.255.181

Routers CE1, CE2, CE3, and CE4 only need basic connectivity to their directly connected PE router. You enable OSPF on the interface that connects the CE routers to the PE routers. Since the configurations for all the CE routers are almost identical, only CE3 and CE4 are shown.

```

Router CE3 [edit]
  protocols {
    ospf {
      area 0.0.0.0 {
        interface t3-0/0/0.0;
      }
    }
  }

```

```

Router CE4 [edit]
  protocols {
    ospf {
      area 0.0.0.0 {
        interface t3-0/0/2.0;
      }
    }
  }

```

PE router configuration is next. Because the configurations for routers PE0 and PE2 mirror each other, only Router PE2 is displayed.

You must enable auto-export at the routing-options level for both the main configuration and the nonforwarding instances, establish policies that set tags on packets arriving from the CE routers, and accept packets into a specific instance that match the corresponding outbound tags. Specifically, you configure the router to attach a *data* tag to all packets coming from Router CE3 and a *voice* tag to all packets arriving from Router CE4. Also, forward any OSPF traffic coming from the core with a *data* tag to Router CE3, while OSPF core traffic with a *voice* tag is sent to Router CE4.

```

Router PE2 [edit]
  routing-options {
    auto-export;
  }
  protocols {
    ospf {
      export [tag-voice tag-data];
      area 0.0.0.0 {
        interface t3-0/1/1.0;
      }
    }
  }
}

```

```
routing-instances {  
  data {  
    instance-type no-forwarding;  
    interface t3-0/1/3.0;  
    routing-options {  
      auto-export;  
    }  
    protocols {  
      ospf {  
        export import-data;  
        area 0.0.0.0 {  
          interface all;  
        }  
      }  
    }  
  }  
  voice {  
    instance-type no-forwarding;  
    interface t3-0/1/0.0;  
    routing-options {  
      auto-export  
    }  
    protocols {  
      ospf {  
        export import-voice;  
        area 0.0.0.0 {  
          interface all;  
        }  
      }  
    }  
  }  
}
```

```

policy-options {
  policy-statement tag-voice {
    from instance voice;
    then {
      tag 11;
      accept;
    }
  }
  policy-statement tag-data {
    from instance data;
    then {
      tag 12;
      accept;
    }
  }
  policy-statement import-voice {
    from {
      instance master;
      protocol ospf;
      tag 11;
    }
    then accept;
  }
  policy-statement import-data {
    from {
      instance master;
      protocol ospf;
      tag 12;
    }
    then accept;
  }
}

```

On Router P, the provider core router configuration is simple. Include the interfaces that connect to the two PE routers (PE0 and PE2) in the OSPF process.

```

Router P [edit]
protocols {
  ospf {
    area 0.0.0.0 {
      interface t1-0/1/1.0;
      interface t3-0/0/1.0;
    }
  }
}

```

If all the configurations are correct, routers CE1 and CE3 (the *data* tagged routers) can send traffic to one another and routers CE2 and CE4 (the *voice* tagged routers) can communicate bidirectionally, but routers with different tag types cannot reach each other.

Checking Your Work

To verify that the nonforwarding instances configuration is functioning properly, you can use the following commands:

```
show ospf database
```

```
show route detail
```

```
ping
```

The following sections show the output of these commands used with the configuration example:

Router PE2 Status on page 443

Router CE3 Status on page 444

Router PE2 Status

```
user@PE2> show ospf database
```

```
OSPF link state database, area 0.0.0.0
Type ID Adv Rtr Seq Age Opt Cksum Len
Router *10.255.255.174 10.255.255.174 0x80000014 180 0x2 0x14b3 60
Router 10.255.255.176 10.255.255.176 0x80000010 592 0x2 0x14c1 60
Router 10.255.255.178 10.255.255.178 0x80000007 1074 0x2 0x9329 84
OSPF AS SCOPE link state database
Type ID Adv Rtr Seq Age Opt Cksum Len
Extern 10.255.255.172 10.255.255.176 0x8000000f 489 0x2 0xd258 36
Extern 10.255.255.180 10.255.255.176 0x8000000f 189 0x2 0x948d 36
Extern *10.255.255.181 10.255.255.174 0x8000000f 780 0x2 0x968c 36
Extern *10.255.255.182 10.255.255.174 0x8000000f 480 0x2 0x7aa8 36
```

```
user@PE2> show ospf database instance voice
```

```
OSPF link state database, area 0.0.0.0
Type ID Adv Rtr Seq Age Opt Cksum Len
Router 10.255.255.181 10.255.255.181 0x80000008 1112 0x2 0x29ac 60
Router *192.255.197.117 192.255.197.117 0x8000000c 2681 0x2 0x5d7a 48
OSPF AS SCOPE link state database
Type ID Adv Rtr Seq Age Opt Cksum Len
Extern *10.255.255.180 192.255.197.117 0x80000001 2681 0x2 0x5cf7 36
```

```
user@PE2> show ospf database instance data
```

```
OSPF link state database, area 0.0.0.0
Type ID Adv Rtr Seq Age Opt Cksum Len
Router 10.255.255.182 10.255.255.182 0x8000000b 1117 0x2 0x53d 60
Router *192.255.197.249 192.255.197.249 0x8000000e 2686 0x2 0xbd05 48
OSPF AS SCOPE link state database
Type ID Adv Rtr Seq Age Opt Cksum Len
Extern *10.255.255.172 192.255.197.249 0x80000002 2686 0x2 0x7d5a 36
```

Router CE3 Status

```
user@CE3> ping 10.255.255.172
PING 10.255.255.172 (10.255.255.172): 56 data bytes
64 bytes from 10.255.255.172: icmp_seq=0 ttl=252 time=2.978 ms
64 bytes from 10.255.255.172: icmp_seq=1 ttl=252 time=2.903 ms
^C
--- 10.255.255.172 ping statistics ---
2 packets transmitted, 2 packets received, 0% packet loss
round-trip min/avg/max/stddev = 2.903/2.941/2.978/0.037 ms
```

```
user@CE3> ping 10.255.255.180
PING 10.255.255.180 (10.255.255.180): 56 data bytes
^C
--- 10.255.255.180 ping statistics ---
2 packets transmitted, 0 packets received, 100% packet loss
```

```
user@CE3> show ospf database
```

```
OSPF link state database, area 0.0.0.0
Type ID Adv Rtr Seq Age Opt Cksum Len
Router *10.255.255.182 10.255.255.182 0x8000000b 1164 0x2 0x53d 60
Router 192.255.197.249 192.255.197.249 0x8000000e 2735 0x2 0xbd05 48
OSPF AS SCOPE link state database
Type ID Adv Rtr Seq Age Opt Cksum Len
Extern 10.255.255.172 192.255.197.249 0x80000002 2735 0x2 0x7d5a 36
```

```
user@CE3> show route 10.255.255.172 detail
```

```
inet.0: 31 destinations, 32 routes (30 active, 0 holddown, 1 hidden)
10.255.255.172/32 (1 entry, 1 announced)
 *OSPF Preference: 150
 Next hop: via t3-0/0/0.0, selected
 State: <Active Int Ext>
 Local AS: 69
 Age: 47:23 Metric: 2 Tag: 12
 Task: OSPF
 Announcement bits (1): 0-KRT
 AS path: I
```

For More Information

For additional information about interinstance route sharing, see the following resources:

JUNOS VPNs Configuration Guide

JUNOS Routing Protocols Configuration Guide

Revision History

2 February 2005—7.1R1 Release. Richard Hendricks.
6 October 2004—7.0R1 Release. Richard Hendricks.
6 July 2004—6.4R1 Release. Richard Hendricks.
5 April 2004—6.3R1 Release. Richard Hendricks.
22 December 2003—6.2R1 Release. Richard Hendricks.
22 September 2003—6.1R1 Release. Richard Hendricks.
30 June 2003—6.0R1 Release. Richard Hendricks.
2 April 2003—5.7R1 Release. Richard Hendricks.
27 December 2002—5.6R1 Release. Richard Hendricks.
30 September 2002—5.5R1 Release. Richard Hendricks.
19 July 2002—5.4R1 Release. Richard Hendricks.
6 May 2002—Initial document written. Richard Hendricks.

