SR Traffic-Engineering

Clarence Filsfils
Kris Michielsen

Acknowledgements

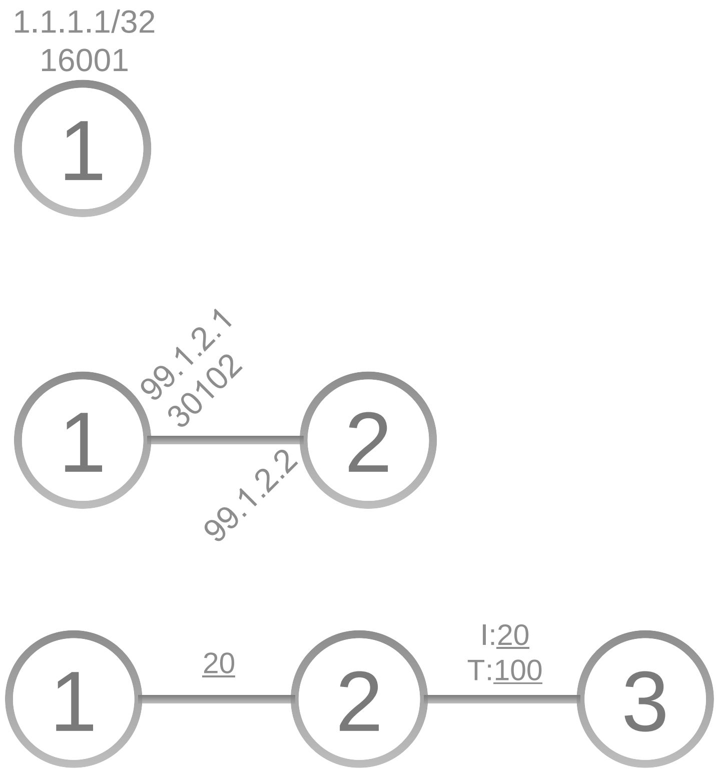
- Bertrand Duvivier
- Jose Liste
- Stefano Previdi

Disclaimer

"Many of the products and features described herein remain in varying stages of development and will be offered on a when-and-if-available basis. This roadmap is subject to change at the sole discretion of Cisco, and Cisco will have no liability for delay in the delivery or failure to deliver any of the products or features set forth in this document."

Illustration Conventions

- For NodeX:
  - Loopback address: 1.1.1.X/32
  - SRGB: [16000 – 23999]
  - Prefix-SID: 16000 + X
- For link NodeX→NodeY:
  - Interface address: 99.X.Y.X/24 (where X<Y)
  - Adjacency-SID: 30X0Y
- Link metric notation
  - IGP & TE metric: <u>xx</u> (default: <u>10</u>)
  - IGP metric: I:<u>xx</u> (default: I:<u>10</u>)
  - TE metric: T:<u>yy</u> (default: T:<u>10</u>)



Key IETF document for SRTE

Internet Engineering Task Force (IETF)
Request for Comments: 9256
Updates: 8402
Category: Standards Track
ISSN: 2070-1721

C. Filsfils
K. Talaulikar, Ed.
Cisco Systems, Inc.
D. Voyer
Bell Canada
A. Bogdanov
British Telecom
P. Mattes
Microsoft
July 2022

Segment Routing Policy Architecture

RSVP-TE

- Little deployment and many issues
- Not scalable
  - Core states in $k \times n^2$
  - No inter-domain
- Complex configuration
  - Tunnel interfaces
- Complex steering
  - PBR, autoroute

n: #head-ends; k: #ECMP

SRTE

- Simple, Automated and Scalable
  - No core state: state in the packet header
  - No tunnel interface: “SR Policy”
  - No head-end a-priori configuration: on-demand policy instantiation
  - No head-end a-priori steering: automated steering
- Multi-Domain
  - SR PCE for compute
  - Binding-SID (BSID) for scale
- Lots of Functionality
  - Designed with lead operators along their use-cases

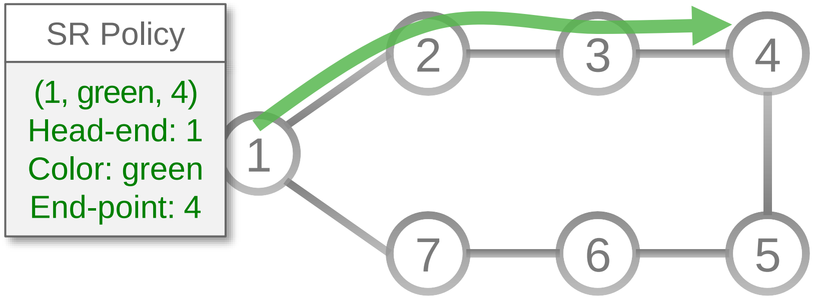
MPLS and SRv6

- The SR and SRTE architecture applies to MPLS and IPv6 data plane implementations
- This document focuses on the MPLS data plane implementation
  - IPv6 data plane implementation (SRv6) will be added in a future revision of this document

SR Policy

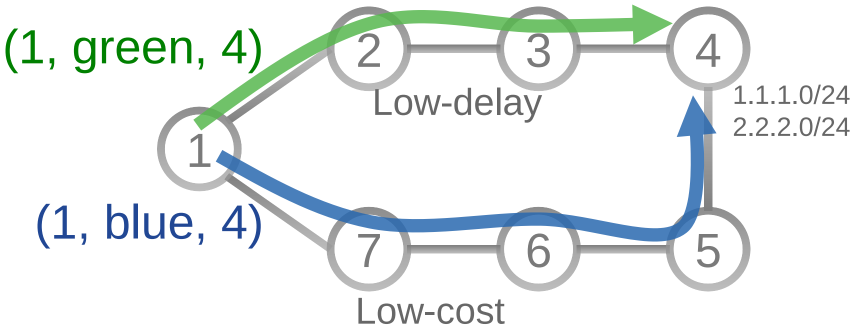
SR Policy Identification

- An SR Policy is uniquely identified by a tuple
  (head-end, color, end-point)
  - Head-end: where the SR Policy is instantiated (*implemented*)
  - Color: a numerical value to differentiate multiple SRTE Policies between the same pair of nodes
  - End-point: the destination of the SR Policy
- At a given head-end, an SR Policy is uniquely identified by a tuple (color, end-point)



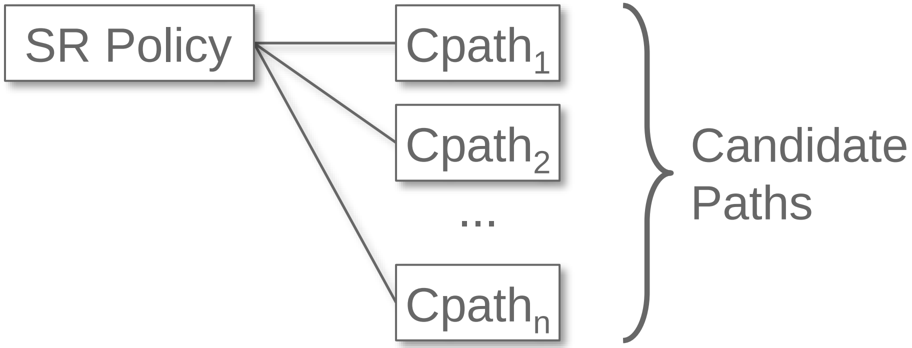
SR Policy Color

- Each SR Policy has a color
  - Color can be used to indicate a certain treatment (SLA, policy) provided by an SR Policy
- Only one SR Policy with a given color C can exist between a given node pair (head-end (H), end-point (E))
  - In other words: each SR Policy triplet (H, C, E) is unique
- Example:
  - Low-cost="blue", Low-delay="green"
  - steer traffic to 1.1.1.0/24 via Node4 into Low-cost SR Policy (1, blue, 4)
  - steer traffic to 2.2.2.0/24 via Node4 into Low-delay SR Policy (1, green, 4)



SR Policy – Candidate Paths

- An SR Policy consists of one or more candidate paths (Cpaths)

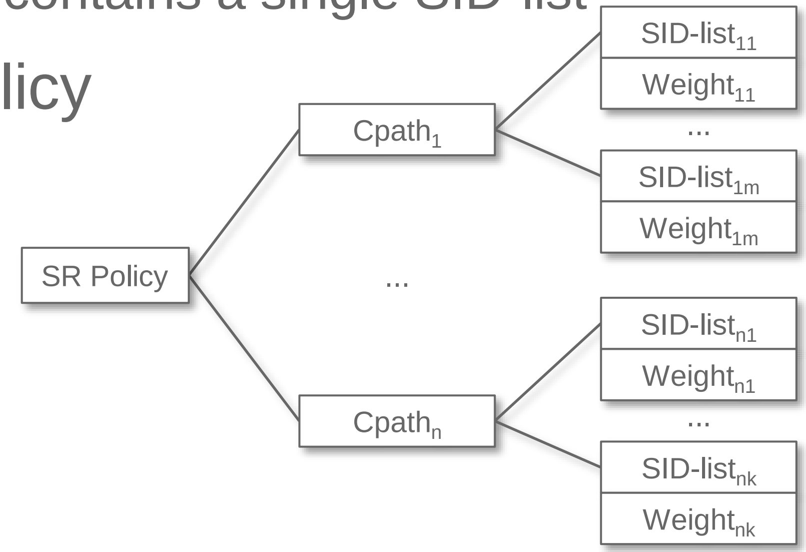


- An SR Policy instantiates one single path in RIB/FIB
  - the selected* path, which is the preferred valid candidate path
- A candidate path is either dynamic or explicit

* See further.

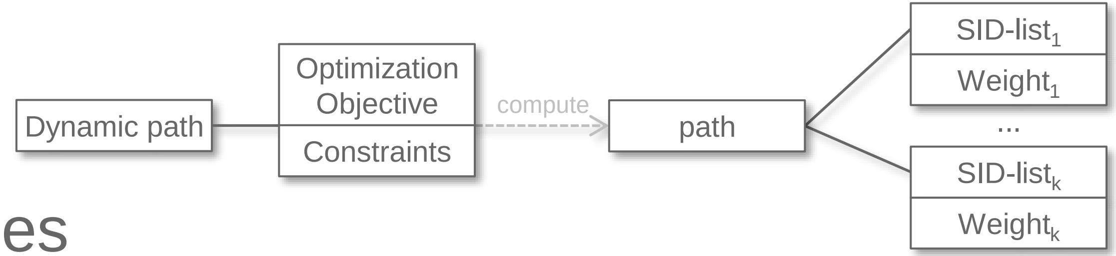
SR Policy – Candidate Path

- A candidate path is a single segment list (SID-list) or a set of weighted* SID-lists
  - Typically, an SR Policy path only contains a single SID-list
- Traffic steered into an SR Policy path is load-shared over all SID-lists of the path



SID = Segment ID
* For Weighted Equal Cost Multi-Path (WECMP) load-sharing. See further.

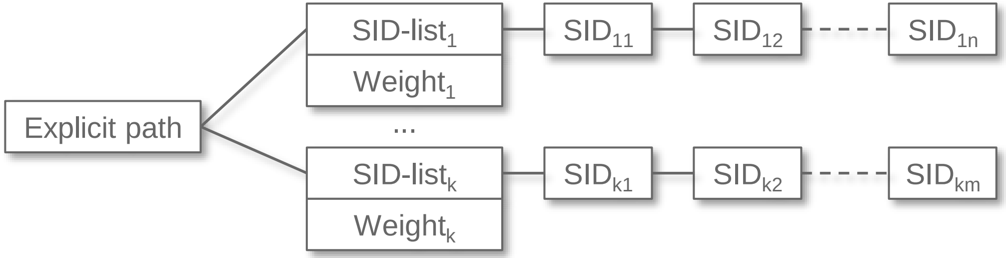
Dynamic Path



- A dynamic path expresses an optimization objective and a set of constraints
- The head-end computes a solution to the optimization problem as a SID-list or a set of SID-lists
- When the head-end does not have enough topological information (e.g. multi-domain problem), the head-end may delegate the computation to a PCE
- Whenever the network situation changes, the path is recomputed

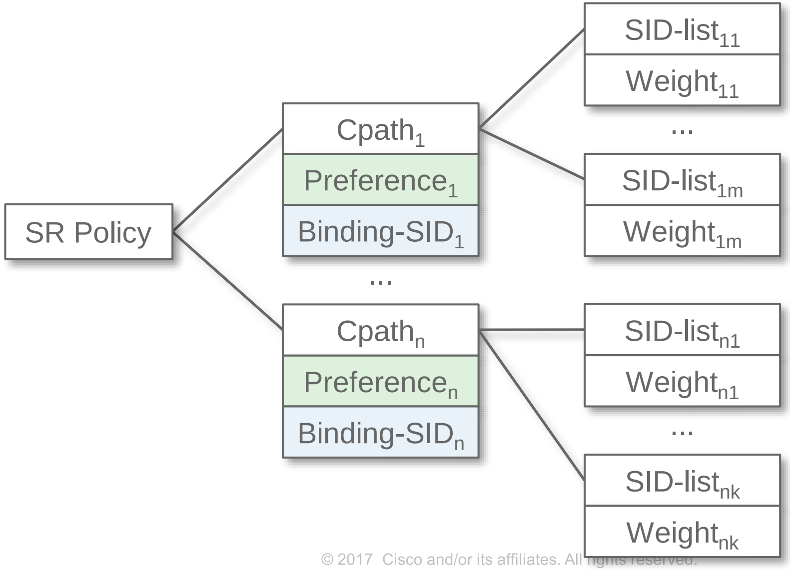
Explicit Path

- An explicit path is an explicitly specified SID-list or set of SID-lists



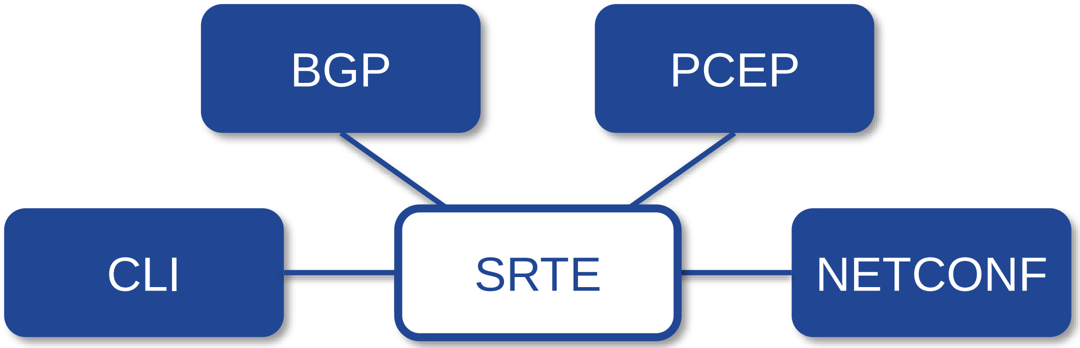
Candidate Paths

- A candidate path has a preference
- A candidate path is associated with a single Binding-SID
- A candidate path is valid if it is usable
  - The validation rules are defined in a later section



Candidate Paths (Cont.)

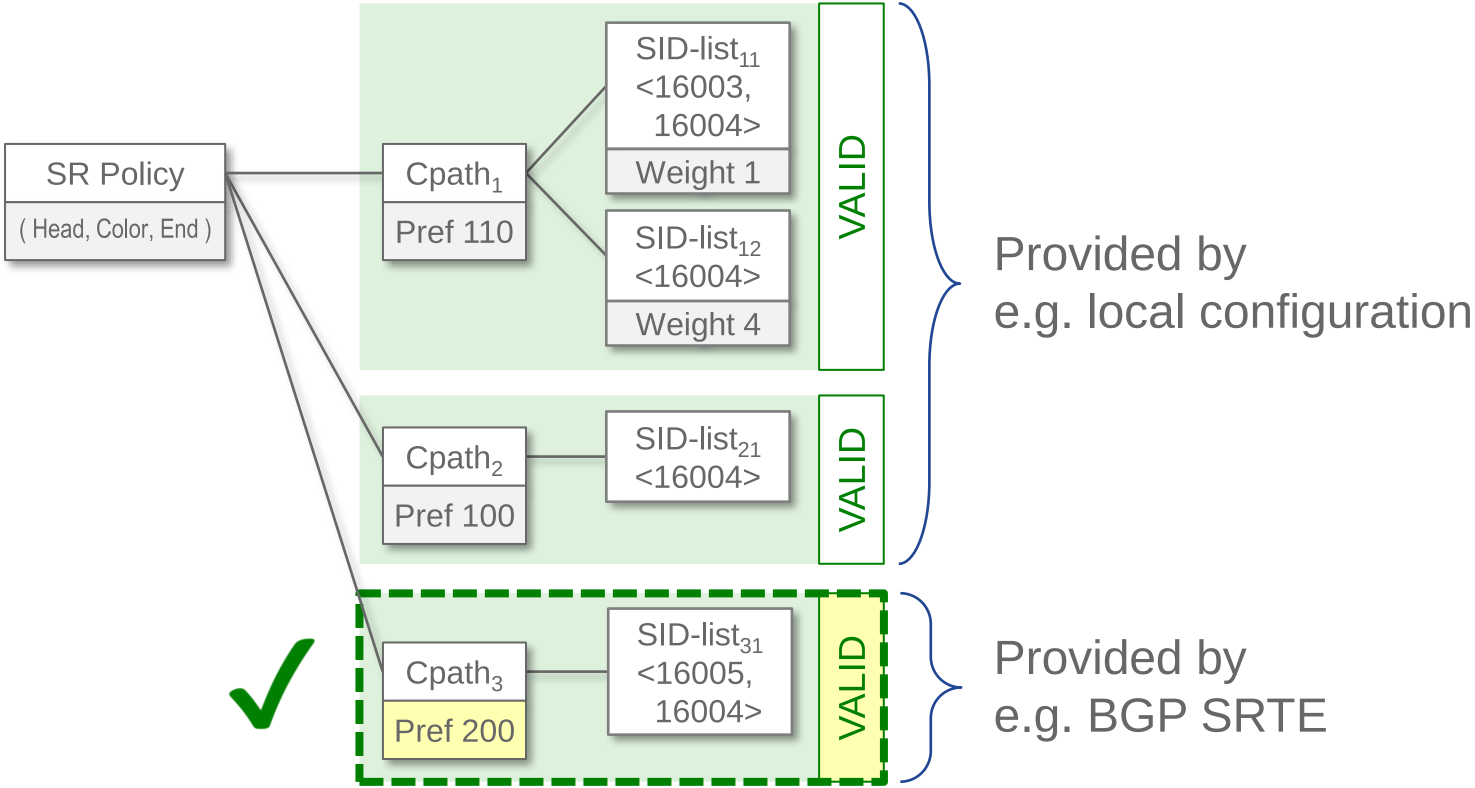
- A head-end may be informed about candidate paths for an SR Policy (color, end-point) by various means including: local configuration (CLI), NETCONF, PCEP, or BGP



Path Selection

- A path is selected for an SR Policy (i.e. it is the preferred path) when the path is valid AND its preference is the best (highest value) among all the candidate paths of the SR Policy
- The protocol source of the path does not matter in the path selection logic

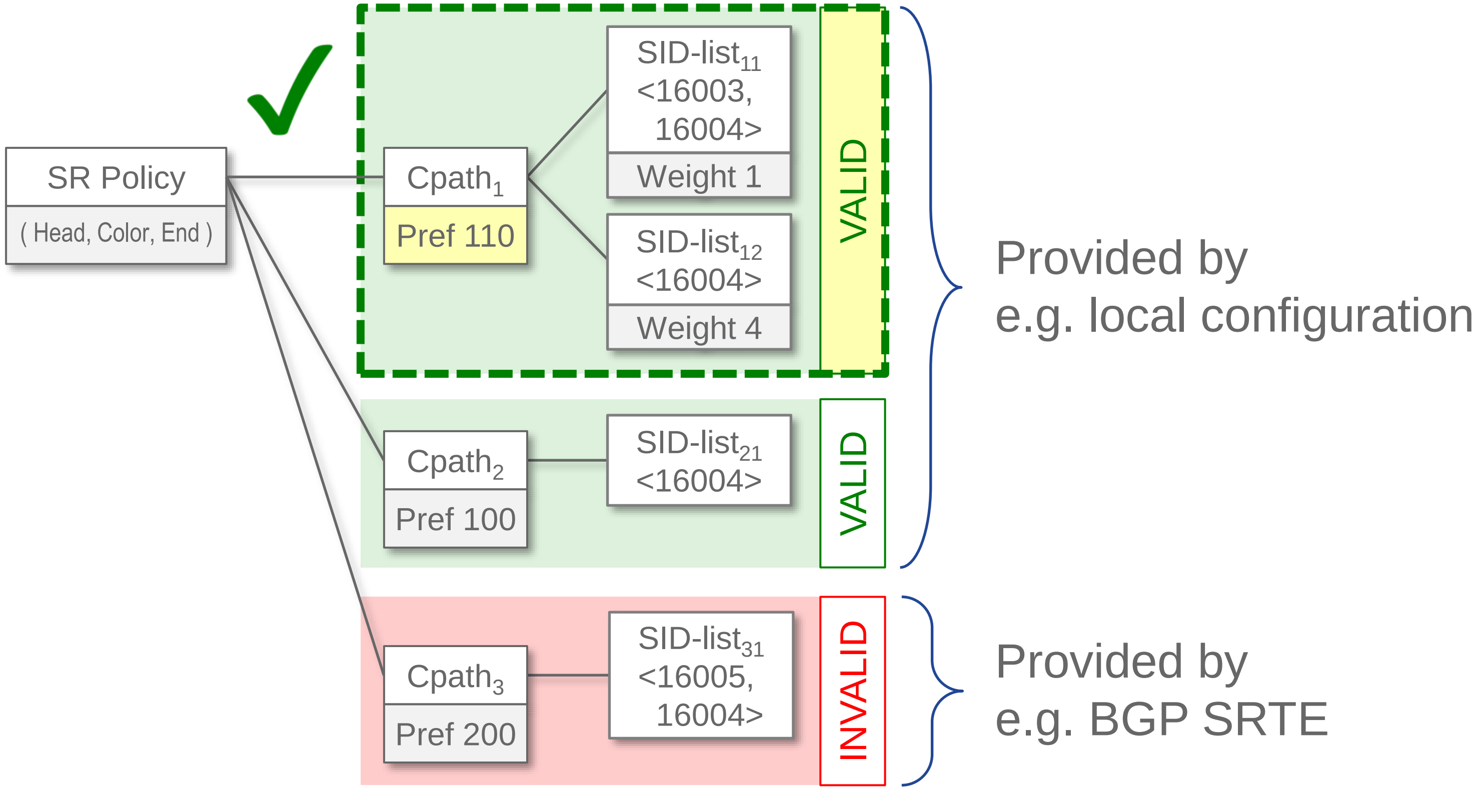
Path's source does not influence selection



Selection of a new preferred path

- Whenever a new candidate path (Cpath) is learned or the validity of an existing Cpath changes or an existing Cpath is changed, the selection process must be re-executed

Selection of a new preferred path



Segment ID (SID)

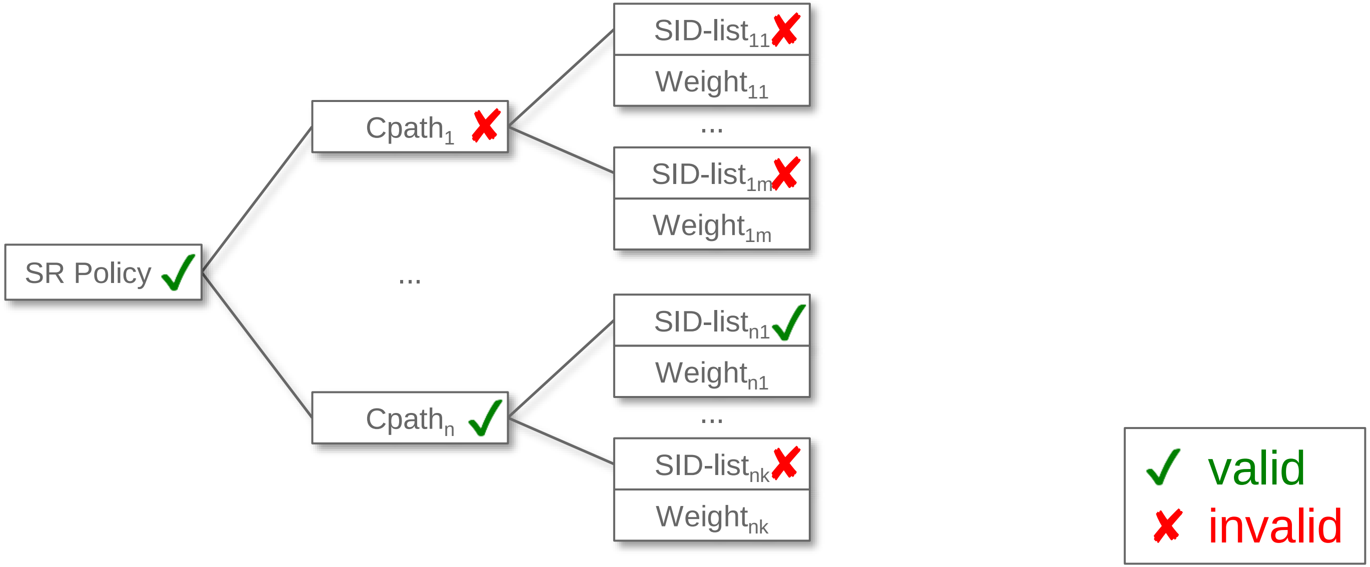
- A SID can either be expressed as
  - A SID (label value)
  - A SID descriptor (used to identify or resolve a SID, e.g. IP address)
- Why?
  - Support inter-domain
    - SID descriptors in remote domains cannot be resolved by the head-end and hence must be expressed as a resolved label
  - Validation control
    - SIDs expressed as label values are not validated (except the first SID in the list)
    - If the designer wants the head-end to validate a SID and that SID is in the SRTE DB of the head-end, then the designer should express it as a SID descriptor

Invalid SID-list

- A SID-list is invalid as soon as:
  - It is empty
  - The head-end is unable to resolve the first SID into one or more outgoing interface(s) and next-hop(s)
  - The head-end is unable to resolve any non-first SID that is expressed as a SID descriptor
- The head-end of an SR Policy updates the validity of a SID-list upon network topological change

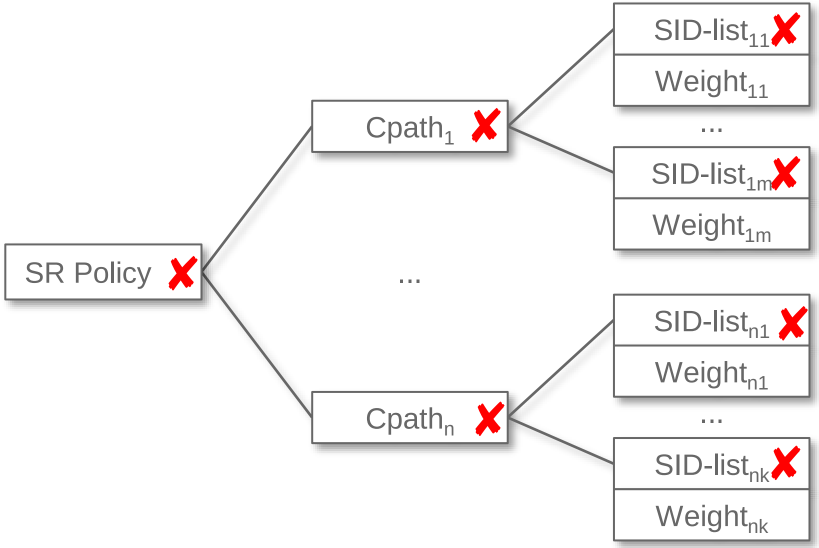
Invalid SR Policy candidate path

- An SR Policy candidate path is invalid as soon as it has no valid SID-list



Invalid SR Policy

- An SR Policy is invalid when
  all its candidate paths are invalid

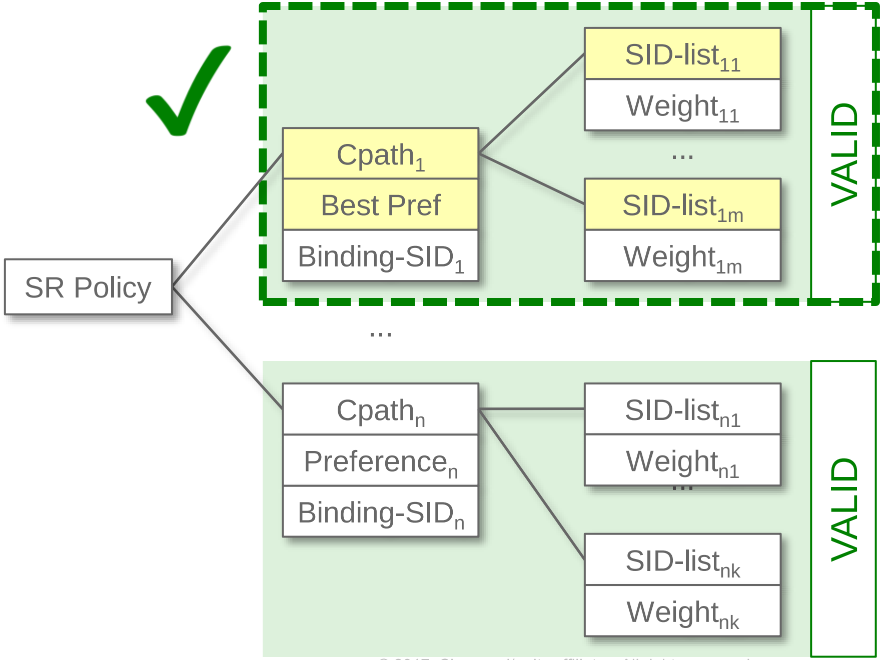


SR Policy invalidation behavior

- If an SR Policy becomes invalid, the invalidation behavior is applied
  - By default: SR Policy forwarding entries are removed and traffic falls back to its default forwarding path (e.g. IGP shortest path)
  - If “invalidation drop” behavior is specified, then the SR Policy forwarding entry (Binding-SID) is kept, but modified to drop all traffic that is steered into the SR Policy

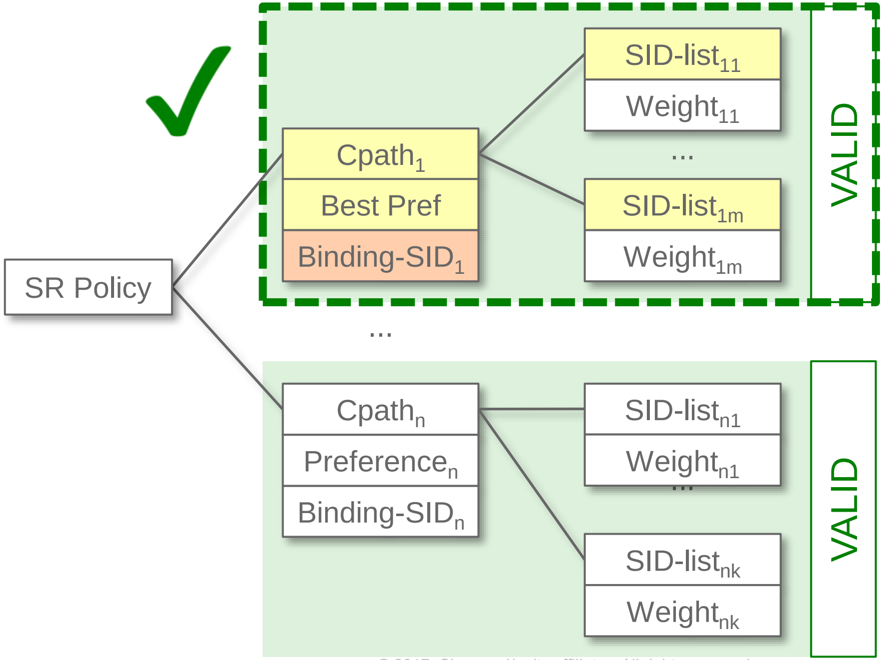
SID-list of an SR Policy

- The SID-list of an SR Policy is the SID-list or set of SID-lists of its selected path
- In practice, most use-cases have a single SID-list per candidate path



Binding-SID (BSID) of an SR Policy

- The BSID of an SR Policy is the BSID of the selected path



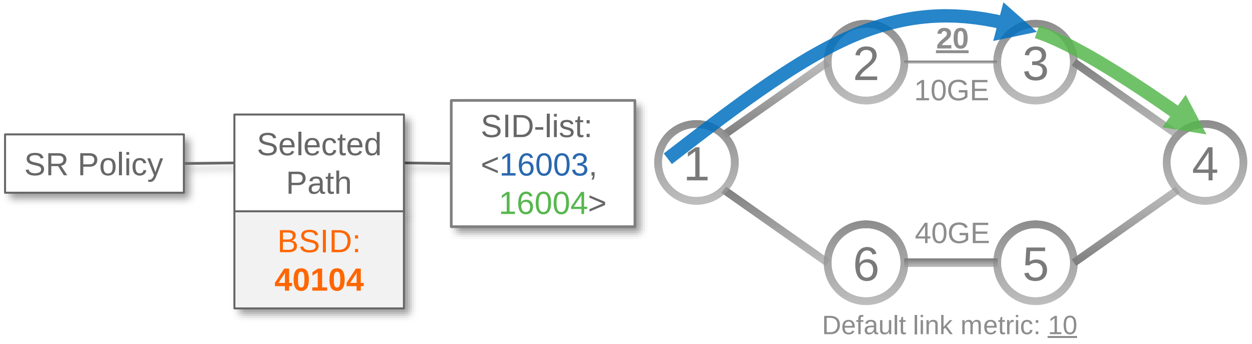
An SR Policy should have a stable BSID

- In all the use-cases known to date, all the candidate paths associated with a given SR Policy have the same BSID
  - Recommendation: design like this!
- One may thus assume that in practice an SR Policy has a stable BSID that is independent of selected-path changes
- One may thus assume that in practice a BSID is an ID of an SR Policy
- However, one should know that a BSID may change over the life of an SR Policy and the true identification of an SR Policy is the tuple (head-end, color, end-point)

Active SR Policy

- An SR Policy (color, end-point) is active at a head-end as soon as this head-end knows about a valid candidate path for this policy
- An active SR Policy installs a BSID-keyed entry in the forwarding table with the action of steering the packets matching this entry to the SID-list(s) of the SR Policy

Active SR Policy – FIB entry

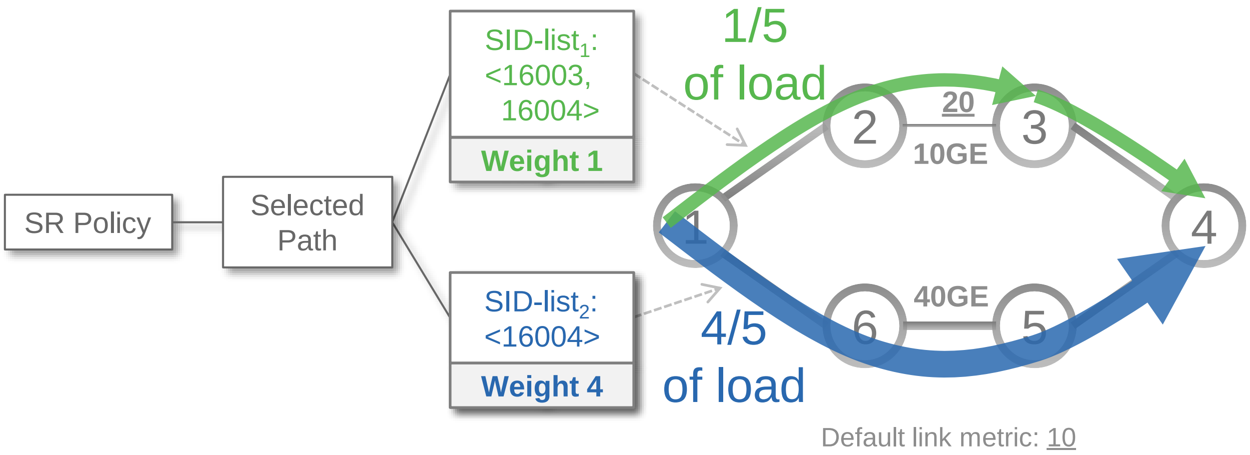


Forwarding table on Node1

In	Out	Out_intf	Fraction
40104	<16003, 16004>	To Node2	**100%**

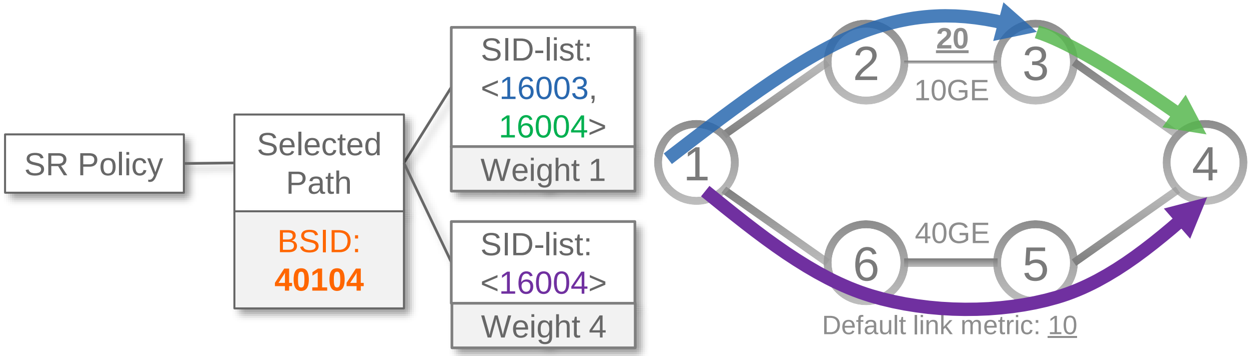
Weighted ECMP (WECMP)

- If a set of SID-lists is associated with the selected path of the SR Policy, then the steering is flow and WECMP-based according to the relative weight of each SID-list



$$\frac{Weight_x}{\sum_{i=1\ldots n} Weight_i}$$

Active SR Policy – FIB entry – WECMP



Forwarding table on Node1

In	Out	Out_intf	Fraction
40104	<16003, 16004>	To Node2	**20%**
	<16004>	To Node6	**80%**

Configuration

Head-end SRTE DB – IGP config

- Enable the following command under ISIS/OSPF to feed the SRTE DB on the head-end:

```
router isis 1
 distribute link-state
```

```
router ospf 1
 distribute link-state
```

- Note: in multi-domain networks, an `instance-id` must be specified with this command
  – See further in this deck for details

Head-end TE Router-ID – IGP config

- Best Practice to configure the TE router-ID in the IGP
  - The implementation assumes this configuration

```
router isis 1
 address-family ipv4 unicast
  router-id Loopback0
```

Head-end SR-TE Policy Source Address

- By default, SR-TE uses the global router-ID as source address
  - This router-ID is automatically determined, and is typically the IPv4 address of the lowest numbered loopback

```
RP/0/RP0/CPU0:R2# show arm router-ids
Router-ID            Interface
1.1.1.2              Loopback0
```

Automatically selected router-id

- Best Practice to configure the SR-TE source address
  - The implementation assumes this configuration

```
segment-routing
 traffic-eng
  candidate-paths
   all
    source-address ipv4 1.1.1.2
```

ARM = Address Repository Manager

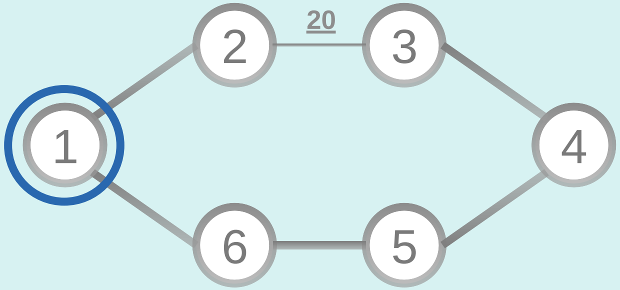
SR Policy – configuration example

On Node1:

```
segment-routing
 traffic-eng
  policy POLICY1
   color 20 end-point ipv4 1.1.1.4
   binding-sid mpls 1000
   candidate-paths
    preference 100
     dynamic
      metric type te
     constraints
      affinity
       exclude-any name red
    !
    preference 200
     explicit segment-list SIDLIST1
  !
  segment-list name SIDLIST1
   index 10 mpls label 16002
   index 20 mpls label 30203
   index 30 mpls label 16004
```

SRTE

SR Policy



Default link metric: 10

```
segment-routing
 traffic-eng
  affinity-map
   name red bit-position 0
```

SR Policy – configuration example

On Node1:

```
segment-routing
 traffic-eng
  policy POLICY1
   color 20 end-point ipv4 1.1.1.4
   binding-sid mpls 1000
   candidate-paths
    preference 100
     dynamic
      metric type te
     constraints
      affinity
       exclude-any name red
    !
    preference 200
     explicit segment-list SIDLIST1
  !
  segment-list name SIDLIST1
   index 10 mpls label 16002
   index 20 mpls label 30203
   index 30 mpls label 16004
```

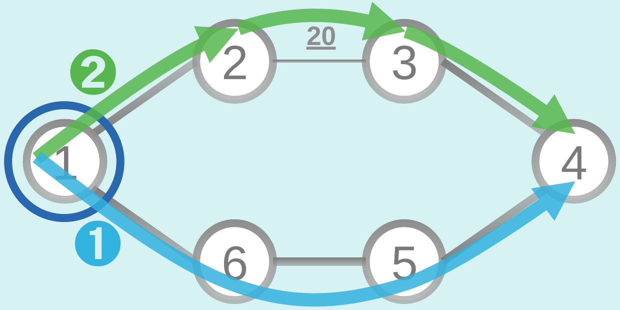
User-defined name

Color and End-point

SR Policy ID: (20,1.1.1.4)

Binding-SID

Local Candidate Paths



Default link metric: 10

```
segment-routing
 traffic-eng
  affinity-map
   name red bit-position 0
```

SR Policy – configuration example

On Node1:

```
segment-routing
 traffic-eng
  policy POLICY1
   color 20 end-point ipv4 1.1.1.4
   binding-sid mpls 1000
   candidate-paths
    preference 100
     dynamic
      metric type te
     constraints
      affinity
       exclude-any name red
    !
    preference 200
     explicit segment-list SIDLIST1
  !
  segment-list name SIDLIST1
   index 10 mpls label 16002
   index 20 mpls label 30203
   index 30 mpls label 16004
```

1 Path preference 100

Dynamic path

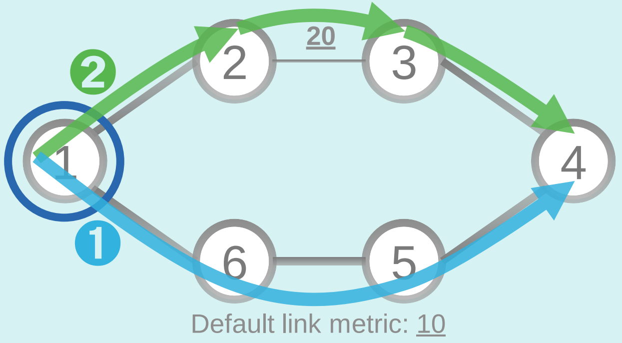
Opt. Obj.: TE metric

Constraint

2 Path preference 200

Explicit SID-list1

SID-list1



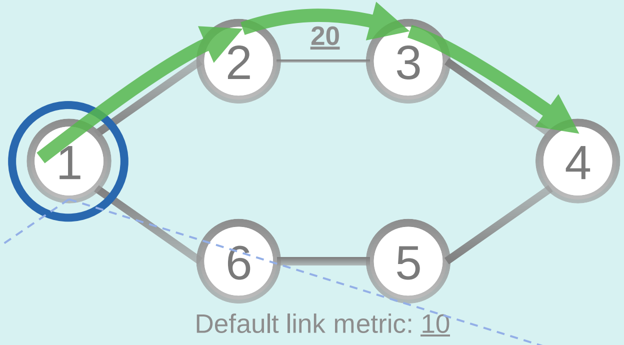

```
segment-routing
 traffic-eng
  affinity-map
   name red bit-position 0
```

SR Policy – configuration example

On Node1:

```
segment-routing
 traffic-eng
  policy POLICY1
   color 20 end-point ipv4 1.1.1.4
   binding-sid mpls 1000
   candidate-paths
    preference 100
     dynamic
      metric type te
     constraints
      affinity
       exclude-any name re
    !
    preference 200
     explicit segment-list SIDLIST1
  !
  segment-list name SIDLIST1
   index 10 mpls label 16002
   index 20 mpls label 30203
   index 30 mpls label 16004
```

Selected Path:
- Valid Path
- Highest Pref value



FIB @ head-end Node1
Incoming label: 1000
Action: pop and push <16002, 30203, 16004>

SR Policy – configuration example

On Node1:

```
segment-routing
 traffic-eng
  policy POLICY1
   color 20 end-point ipv4 1.1.1.4
   binding-sid mpls 1000
   candidate-paths
    preference 100
     dynamic
      metric type te
     constraints
      affinity
       exclude-any name red
    !
    preference 200
     explicit segment-list SIDLIST1
  !
  segment-list name SIDLIST1
   index 10 mpls label 16002
   index 20 mpls label 30203
   index 30 mpls label 16004
```

Node1 may receive other candidate paths for SR Policy (20, 1.1.1.4) from other sources, some examples:

Source of path is not considered for path selection

Selected Path:
- Valid Path
- Highest Pref value

Other candidate paths received for SR Policy (20, 1.1.1.4)

Path received via BGP signaling
preference 150
binding-sid mpls 1000
weight 1, SID-list <16002, 16005>
weight 2, SID-list <16004, 16008>

Path received via PCEP signaling
preference 120
binding-sid mpls 1000
SID-list <16002, 16005>

Path received via NETCONF signaling
preference 50
binding-sid mpls 1000
SID-list <16002, 16005>

WECMP example

On Node1:

```
segment-routing
 traffic-eng
  policy POLICY1
   color 20 end-point ipv4 1.1.1.4
   binding-sid mpls 1000
   candidate-paths
    preference 200
     explicit segment-list SIDLIST1
      weight 1
     !
     explicit segment-list SIDLIST2
      weight 4
     !
  segment-list name SIDLIST1
   index 10 mpls label 16002
   index 20 mpls label 30203
   index 30 mpls label 16004
  !
  segment-list name SIDLIST2
   index 10 address ipv4 1.1.1.4
```

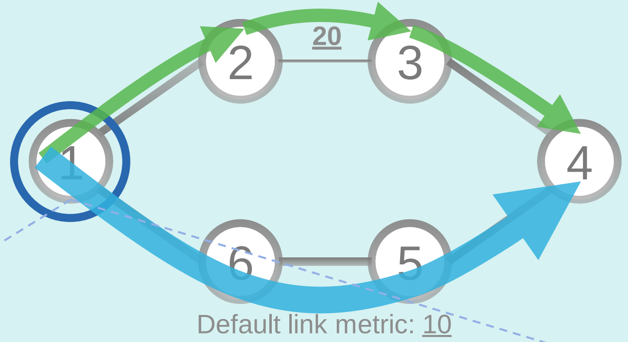
Path preference 200

Explicit SID-list1, Weight 1

Explicit SID-list2, Weight 4

SID-list1

SID-list2



FIB @ head-end Node1
Incoming label: 1000
Action:pop and push <16002, 30203, 16004> (20%)
push <16004> (80%)

Explicit Path

SID-list with addresses – example

On Node1:

```
segment-routing
 traffic-eng
  policy POLICY1
   color 2 end-point ipv4 1.1.1.4
   candidate-paths
    preference 100
     explicit segment-list SIDLIST1
```

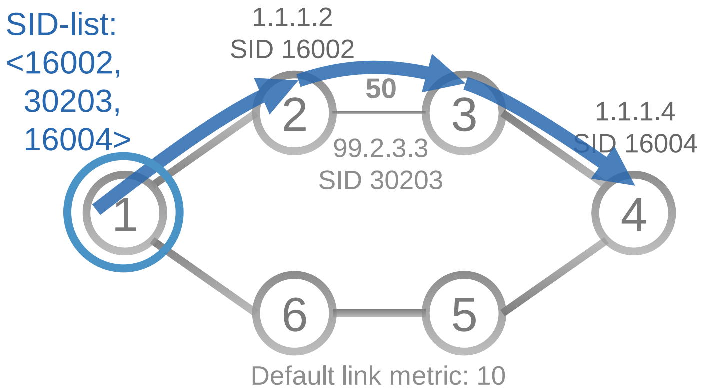
```
segment-list name SIDLIST1
 index 10 address ipv4 1.1.1.2
 index 20 address ipv4 99.2.3.3
 index 30 address ipv4 1.1.1.4
```

→ Prefix-SID 16002

→ Adj-SID 30203

→ Prefix-SID 16004

Outgoing interface from first SID: to Node2



SID-list with labels – example

On Node1:

```
segment-routing
 traffic-eng
  policy POLICY1
   color 2 end-point ipv4 1.1.1.4
   candidate-paths
    preference 100
     explicit segment-list SIDLIST1
```

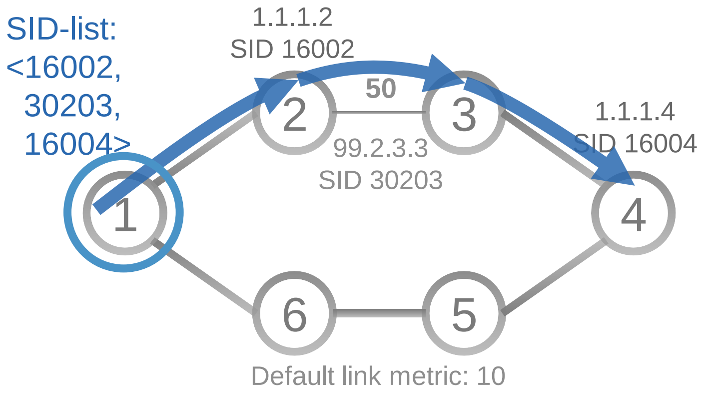
```
segment-list name SIDLIST1
 index 10 mpls label 16002
 index 20 mpls label 30203
 index 30 mpls label 16004
```

→ Prefix-SID Node2

→ Adj-SID Adj2-3

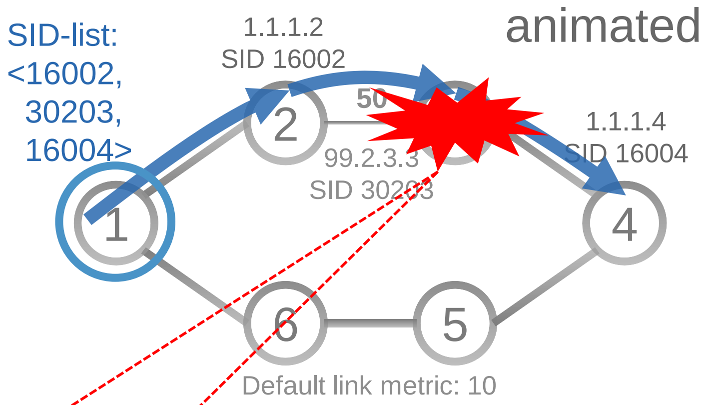
→ Prefix-SID Node4

Outgoing interface from first SID: to Node2



Path Validation

- Validation of:
  - First SID
  - Non-first SID expressed as an IP address




```
segment-list name SIDLIST1
  index 10 mpls label 16002
  index 20 mpls label 30203
  index 30 mpls label 16004
```

Not validated – path remains valid

```
segment-list name SIDLIST2
  index 10 address ipv4 1.1.1.2
  index 20 address ipv4 99.2.3.3
  index 30 address ipv4 1.1.1.4
```

Validated – path is invalid

Set of SID-lists – example

On Node1:

```
segment-routing
 traffic-eng
  policy POLICY1
   color 2 end-point ipv4 1.1.1.4
   candidate-paths
    preference 100
     explicit segment-list SIDLIST1
      weight 1
     !
     explicit segment-list SIDLIST2
      weight 4
     !
    !
   !
  !
```

candidate path

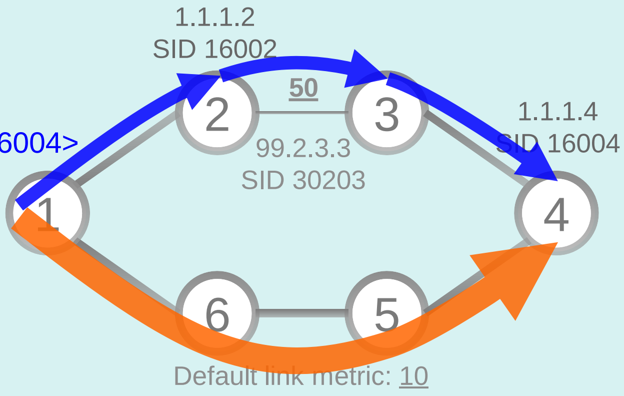
Explicit SID-list1, Weight 1

Explicit SID-list2, Weight 4

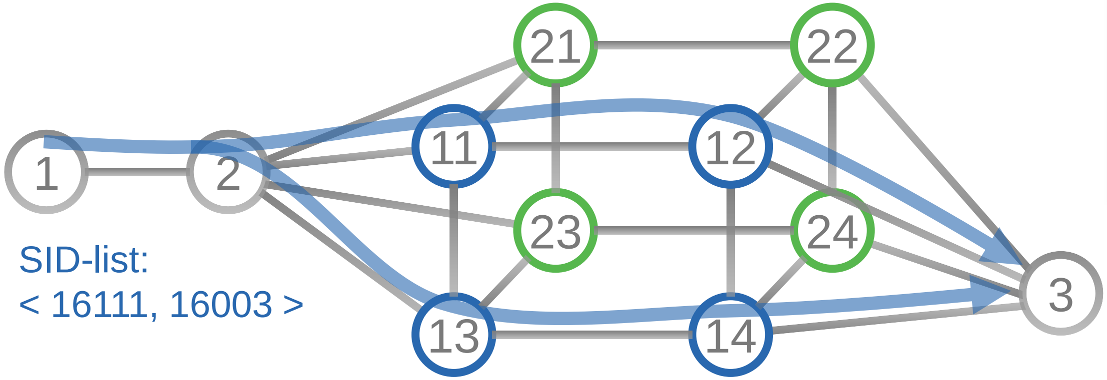
```
segment-list name SIDLIST1
 index 10 address ipv4 1.1.1.2
 index 20 address ipv4 99.2.3.3
 index 30 address ipv4 1.1.1.4
!
segment-list name SIDLIST2
 index 10 address ipv4 1.1.1.4
```

SID-list (weight 1)
<16002, 30203, 16004>

SID-list (weight 4)
<16004>



Use-case
Dual Plane – Anycast-SID

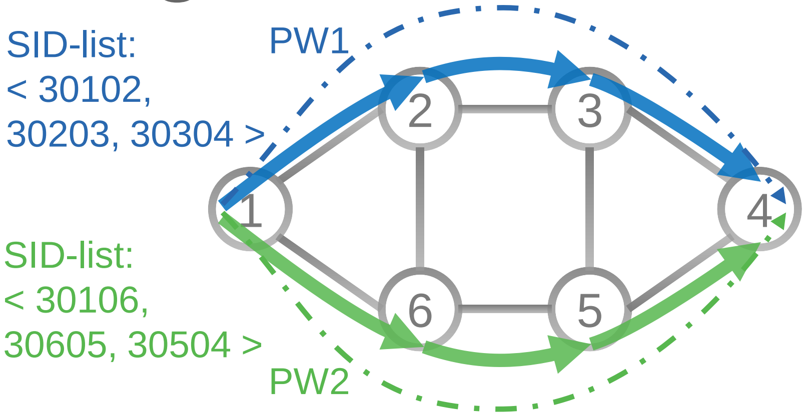


Node1

```
segment-routing
 traffic-eng
  policy POLICY1
   color 2 end-point ipv4 1.1.1.3
   candidate-paths
    preference 100
     explicit segment-list SIDLIST1
  !
  segment-list name SIDLIST1
   index 10 address ipv4 1.1.1.111
   index 20 address ipv4 1.1.1.3
```

- The nodes on Plane1 (blue) advertise Anycast-SID 16111 (1.1.1.111/32)
- The nodes on Plane2 (green) advertise Anycast-SID 16222 (1.1.1.222/32)
- The explicit path on Node1 steers packets via SID-list <16111, 16003>
  - The path stays on Plane1, except if both uplinks to Plane1 fail or Plane1 becomes partitioned

Use-case
TDM migration



- Two disjoint pseudowires from Node1 to Node4
  - PW1 steered into SR Policy BLUE
    PW2 steered into SR Policy GREEN
- PWs are transported via pinned down paths
  - Unprotected: using unprotected Adjacency-SIDs
  - PW traffic dropped when path is invalid (invalidation drop)

Node1

```
segment-routing
 traffic-eng
  policy BLUE
   color 10 end-point ipv4 1.1.1.4
   steering invalidation drop
   candidate-paths
    preference 100
     explicit segment-list SIDLIST1
     constraints segments unprotected
   !
  policy GREEN
   color 20 end-point ipv4 1.1.1.4
   steering invalidation drop
   candidate-paths
    preference 100
     explicit segment-list SIDLIST2
     constraints segments unprotected
   !
  segment-list name SIDLIST1
   index 10 address ipv4 99.1.2.2
   index 20 address ipv4 99.2.3.3
   index 30 address ipv4 99.3.4.4
 !
  segment-list name SIDLIST2
   index 10 address ipv4 99.1.6.6
   index 20 address ipv4 99.5.6.5
   index 30 address ipv4 99.4.5.4
```

Dynamic Path

Optimization Objectives and Constraints

- TE path computation algorithms solve optimization problems with constraints
  - E.g. “find lowest delay path that avoids link RED”, or “find two lowest cost paths that are disjoint”
- New efficient SR-native algorithms have been developed providing solutions that leverage the ECMP-awareness of SR and minimize the size of the resulting SID-list
- Extensive scientific research is backing these new SRTE algorithms: SIGCOMM 2015*

* http://conferences.sigcomm.org/sigcomm/2015/pdf/papers/p15.pdf

SR-optimized algorithms

Circuit optimization vs SR optimization

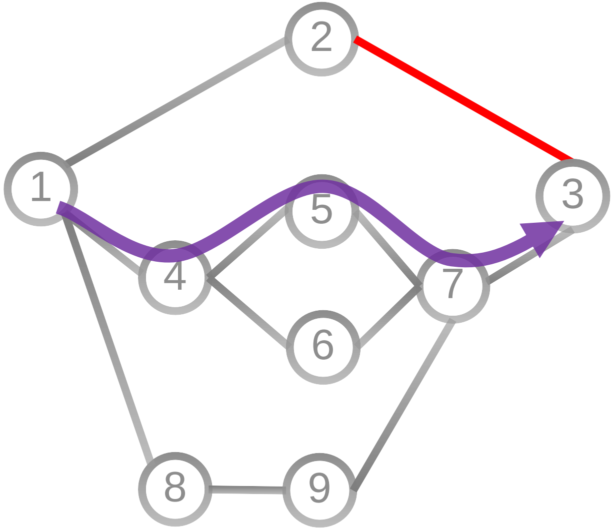
- The introduction of Classic TE (RSVP-TE) made traditional circuit-based L2 (ATM, Frame-relay) functionality available in IP networks
  - Classic TE is circuit-based, including its path computation algorithms
- Though ECMP is omnipresent in IP networks, Classic TE circuit-based paths do not natively leverage ECMP
- SR forwarding and SR-optimized computations preserve ECMP of IP networks and minimize the resulting SID-list size

SR-optimized algorithms

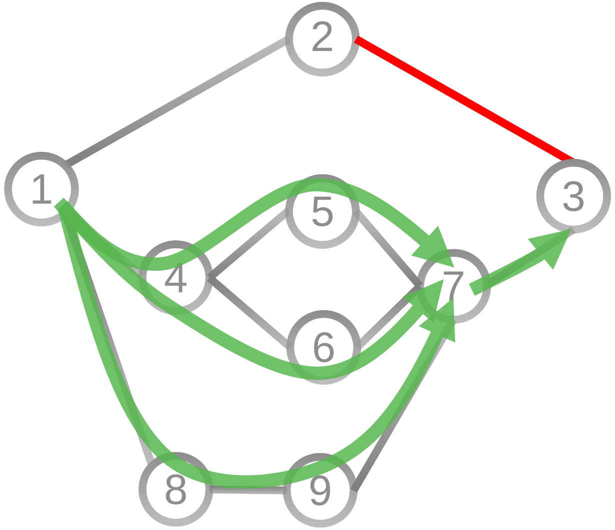
Circuit optimization vs SR optimization

- Using classic TE circuit-based path computation and translating the path in a SID-list does not provide the desired solution
  - Not ECMP-aware, needs multiple circuits for load-sharing
  - Results in a large SID-list to express the path
- A lot of research went into the development of efficient, SR-optimized path computation algorithms
  - Natively ECMP-aware
  - Path expressed in a small SID-list

Circuit Optimization vs SR Optimization



Classic TE is circuit-based
CSPF => non-ECMP path
Classic Algo is no good!!
SID-list: <4, 5, 7, 3>
Poor ECMP, big SID-list, ATM optimized



SR-native TE is needed
!No more circuit!
Recognized Innovation - Sigcomm 2015
SID-list: <7, 3>
ECMP, Small SID-list, IP-optimized

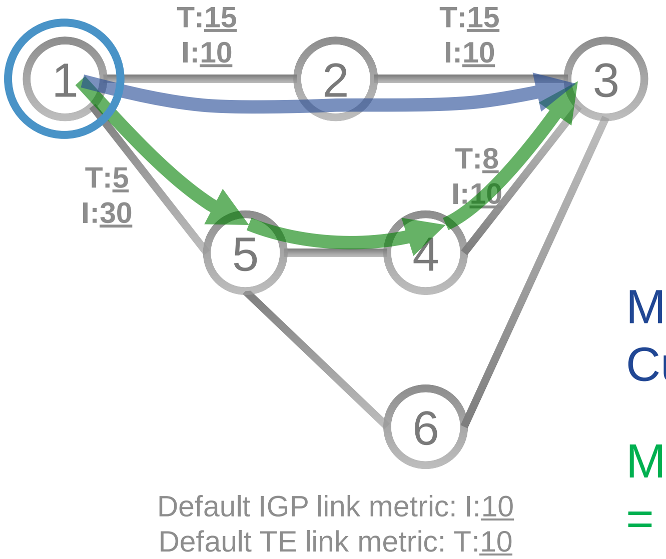
SR-optimized algorithms

Circuit optimization vs SR optimization

- In the vast majority of SR use-cases, native SR-optimized algorithms are preferred
- In some specific use-case (e.g. TDM migration over IP where the circuit notion prevails), one may prefer a classic circuit computation followed by an encoding into SIDs

Optimization Objectives

Min-Metric Optimization



Node1

```
segment-routing
  traffic-eng
    policy POLICY1
      color 20 end-point ipv4 1.1.1.3
      candidate-paths
        preference 100
          dynamic
            metric
              type [igp|te|latency]
```

Min-Metric(1 → 3, IGP) = SID-list <16003>
Cumulated IGP metric: 20

Min-Metric(1 → 3, TE)
= SID-list <16005, 16004, 16003>
Cumulated TE metric: 23

- Head-end computes a SID-list that expresses the shortest-path according to the selected metric

Min-Metric with Margin and max SID-list

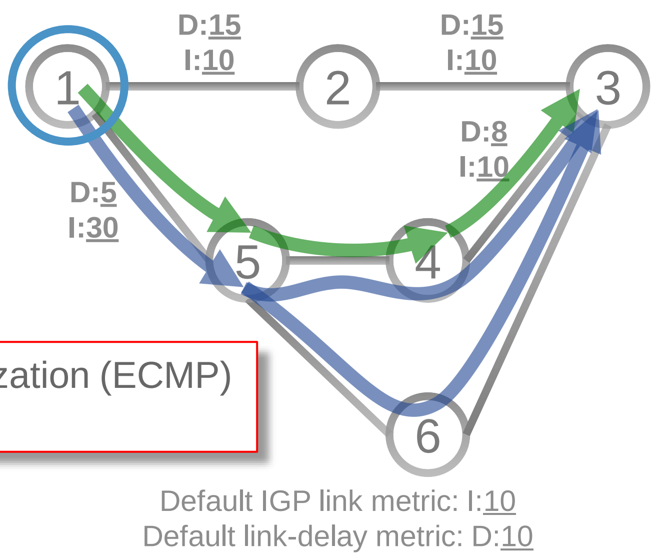
- Head-end computes a SID-list such that packets flowing through it do not use a path whose cumulated optimized metric is larger than the shortest-path for the optimized metric + margin
- Margin can be expressed as an absolute value or as a relative value (percentage) (`margin relative <%>`)

```
segment-routing
  traffic-eng
    policy POLICY1
      color 20 end-point ipv4 1.1.1.3
      candidate-paths
        preference 100
          dynamic mpls
            metric
              type latency
              margin absolute 5
```

Why *Min-metric with margin*?

- In many deployments there are insignificant metric differences between mostly equal paths (e.g. a difference of 100 usec of delay between two paths from NYC to SFO would not matter in most cases)
- The *Min-Metric with margin* relaxes the “absolute” Min-Metric objective to favor more ECMP or shorter SID-list instead of insignificant optimization increment

Min-Metric with Margin and max SID-list



- Optimal link utilization (ECMP)
- Smaller SID-list

Node1

```
segment-routing
  traffic-eng
    policy POLICY1
      color 20 end-point ipv4 1.1.1.3
      candidate-paths
        preference 100
          dynamic
            metric
              type latency
              margin absolute 5
```

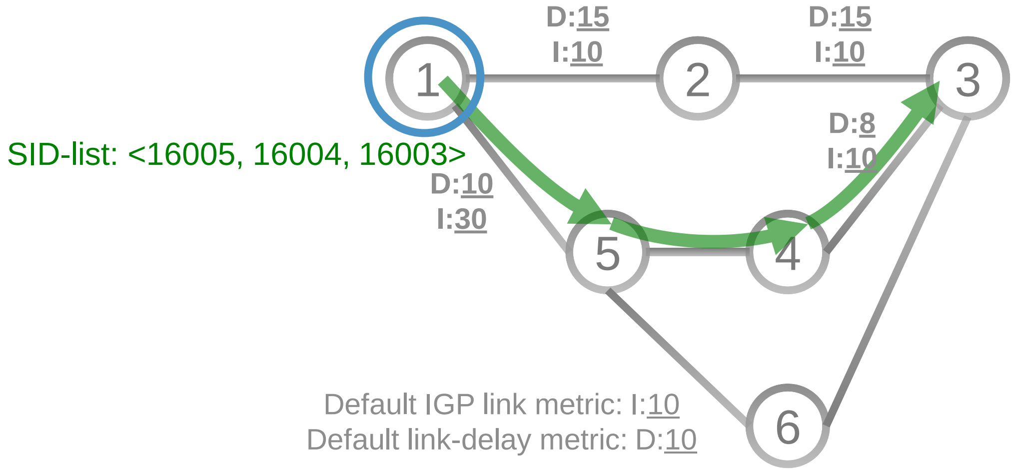
Min-metric

Min-Metric(1 to 3, delay)
= SID-list <16005, 16004, 16003>
Cumulated delay metric = 23

Min-metric with margin

Min-Metric(1 to 3, delay, m=5)
= SID-list <16005, 16003>
Max Cumulated delay metric = 25 < 23 + 5

Use-case
Low-delay

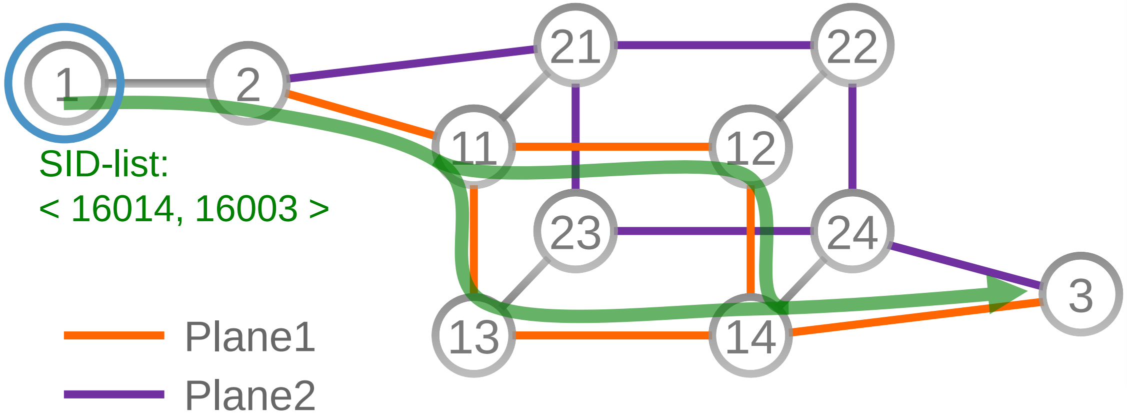


Node1

```
segment-routing
  traffic-eng
    policy POLICY1
      color 20 end-point ipv4 1.1.1.3
      candidate-paths
        preference 100
          dynamic
            metric
              type latency
```

- Min-metric on link-delay metric
  - Same with margin and max-SID
  - Same with link-delay metric automatically measured by a node for its attached links and distributed in the IGP

Use-case
Plane Affinity



Node1

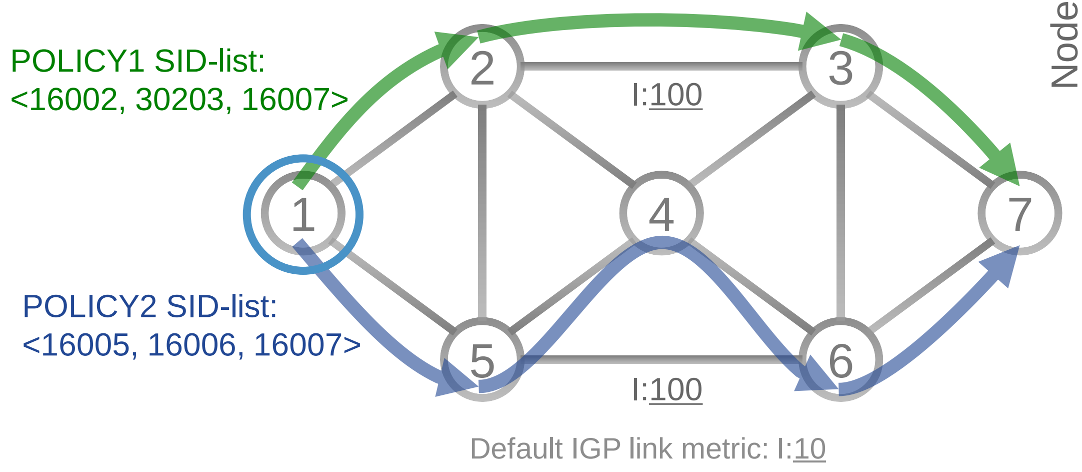
```
segment-routing
  traffic-eng
   affinity-map
     color Plane1 bit-position 0
     color Plane2 bit-position 1
    !
    policy POLICY1
      color 20 end-point ipv4 1.1.1.3
      candidate-paths
        preference 100
         dynamic
           metric type igp
         constraints
          affinity
           exclude-any Plane2
```

- Min-Metric on IGP metric with exclusion of a TE-affinity “Plane2”
  - All the links in Plane2 are set with TE-affinity “Plane2”

More details of affinity configuration in the “Constraints” section

Use-case

Service Disjointness from same head-end



Node1

```
segment-routing
 traffic-eng
   policy POLICY1
    color 20 end-point ipv4 1.1.1.7
    candidate-paths
     preference 100
      dynamic
       metric type igp
      constraints
       disjoint-path group-id 1 type node
   !
   policy POLICY2
    color 30 end-point ipv4 1.1.1.7
    candidate-paths
     preference 100
      dynamic
       metric type igp
      constraints
       disjoint-path group-id 1 type node
```

- The head-end computes two disjoint paths

More details of disjointness configuration in the "Constraints" and "Path Disjointness" sections

Constraints

Constraints

- The following constraints can be specified:
  - Include and/or exclude TE affinity
  - Include and/or exclude IP address
  - Include and/or exclude SRLG
  - Maximum accumulated metric (IGP, TE, and delay)
  - Maximum number of SIDs in the solution SID-list
  - Disjoint from another SR Policy in the same association group

Constraint – TE affinity

- Links in the network can be “colored”
  - E.g. “country X”, “under maintenance”, ...
- SRTE can compute a path that includes or excludes links that have specific (combinations of) colors

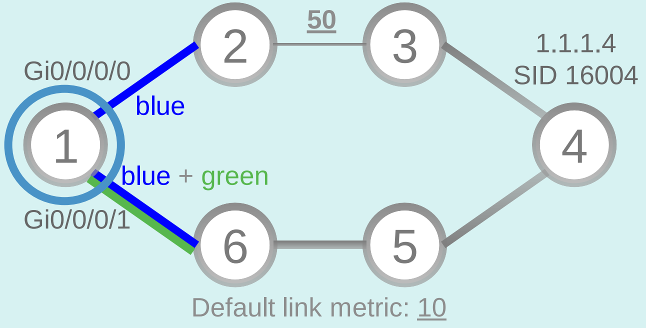
Constraint – Add affinity colors to links

On Node1:

```
segment-routing
 traffic-eng
  affinity-map
   !! 32-bit maps
   color blue bit-position 0
   color red bit-position 1
   color green bit-position 2
  !
  interface Gi0/0/0/0
   affinity color blue
  !
  interface Gi0/0/0/1
   affinity color blue
   affinity color green
```

Define user-friendly names for affinity bit-maps

Assign affinity bit-map to interface



- “Color” links/interfaces by assigning affinity bit-maps to them

Constraint – TE affinity

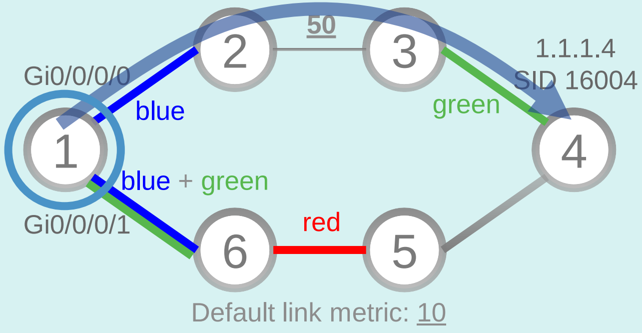
- Specify “affinity” or “relationship” between SR Policy path and link colors
- An SR Policy path can specify:
  - Include-any <color> [<color> ...]: only traverse links that have any of the specified colors
  - Include-all <color> [<color> ...]: only traverse links that have all of the specified colors
  - Exclude-any <color> [<color> ...]: do not traverse links that have any of the specified colors

Constraint – SR Policy Path affinity

On Node1:

```
segment-routing
  traffic-eng
    policy POLICY1
      color 20 end-point ipv4 1.1.1.4
      binding-sid mpls 1000
      candidate-paths
        preference 100
          dynamic
            metric type igp
          constraints
            affinity
              exclude-any name red
```

Don't use links with color red



- Specify the relationship (affinity) of the SR Policy path with the link colors

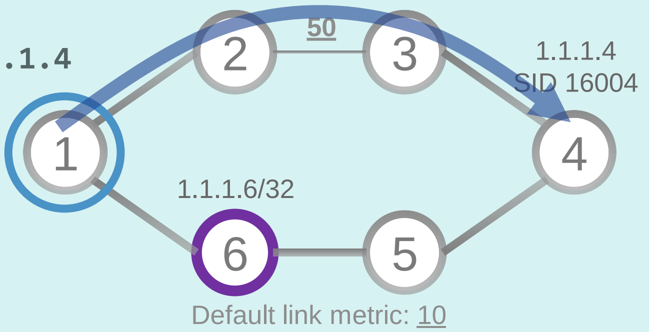
Constraint – IP address

- SRTE can compute paths that avoid specific resources that are identified by their IP address
  - Links
  - Nodes
  - Sets of nodes (anycast set)

On Node1:

```
prefix-set SET1
  1.1.1.6/32
end-set
!
segment-routing
  traffic-eng
    policy POLICY1
      color 20 end-point ipv4 1.1.1.4
      candidate-paths
        preference 100
          dynamic
            metric type igp
          constraints
            address
              exclude SET1
```

Avoid node with address 1.1.1.6/32

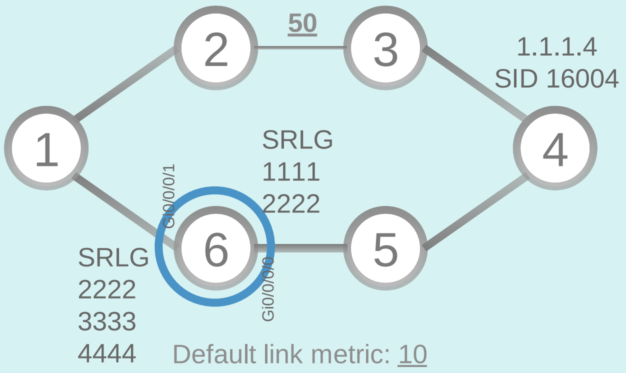


Constraint – SRLG

- Shared Risk Link Groups (SRLGs) are identified by a number
  - Links with the same SRLG id share a common risk (e.g. same fiber conduit)

On Node6:

```
srlg
 interface Gi0/0/0/0
  10 value 1111
  20 value 2222
 !
 interface Gi0/0/0/1
  10 value 2222
  20 value 3333
  30 value 4444
 !
!
```



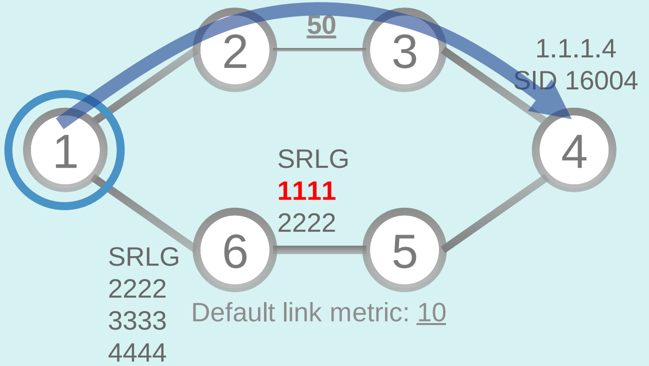
Constraint – SRLG

- SRTE can compute paths that excludes links that have specific SRLGs

On Node1:

```
segment-routing
  traffic-eng
    policy POLICY1
      color 20 end-point ipv4 1.1.1.4
      candidate-paths
        preference 100
          dynamic
            metric type igp
          constraints
           srlg
            exclude 1111
```

Don't use links with SRLG 1111



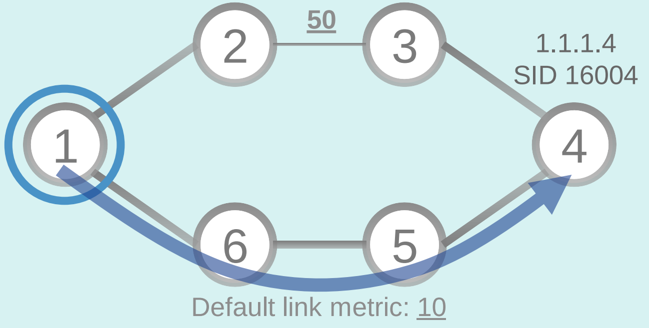
Constraint – maximum metric

- SRTE can put an absolute limit on the cumulative metric of a computed path

On Node1:

```
segment-routing
  traffic-eng
    policy POLICY1
      color 20 end-point ipv4 1.1.1.4
      candidate-paths
        preference 100
          dynamic
            metric type igp
          constraints
            bounds
             cumulative
              type igp 80
```

Cumulative metric must be ≤ 80



Constraint – limit SIDs

- SRTE can put an absolute limit on the number of SIDs in the SID-list of a computed path

```
segment-routing
  traffic-eng
    policy POLICY1
      color 20 end-point ipv4 1.1.1.4
      candidate-paths
        preference 100
          dynamic
            metric type igp
          constraints
            bounds
              cumulative
               type sid 5
```

Maximum 5 SIDs in the solution SID-list

Constraint – disjointness

- SRTE can compute a path that is disjoint from another path in the same disjoint-group
- See Path Disjointness section

```
segment-routing
 traffic-eng
  policy POLICY1
   color 10 end-point 1.1.1.3
   candidate-paths
    preference 100
     dynamic
      pcep
      metric type te
     constraints
      disjoint-path group-id 1 type node
       !! disjoint-path group-id <group ID> type (link | node | srlg | srlg-node )
```

Member of Node-disjoint group 1

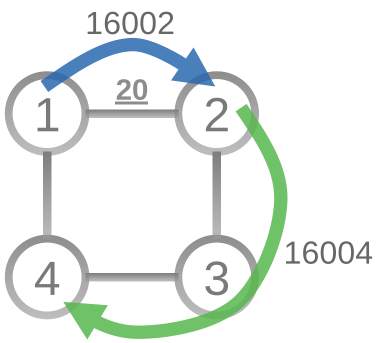
Topological path → SID-list

Topological path → SID-list

- After the path is computed, the SID-list that expresses the desired path is derived
- High-level algorithm:
  1. Node = head-end
  2. Find an IGP prefix-SID that leads as far down the desired path as possible (without using any link not included in the desired path)
  3. If no such prefix-SID exists, use the Adj-SID to the first neighbor along the path
  4. Node = the farthest node that is reached; goto 2.

Topological path → SID-list – Example 1

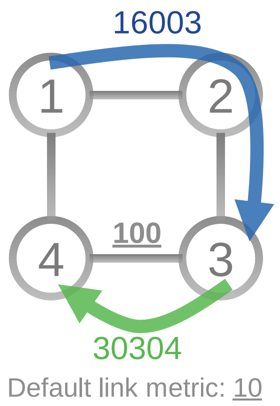
- Desired topological path = 1→2→3→4
- SID-list = <16002, 16004>
  - 16002 brings the packet from 1 to 2 (shortest path from Node1 to Node2)
  - 16004 brings the packet from 2 to 4 via 3 (shortest path from Node2 to Node4)



Default link metric: 10

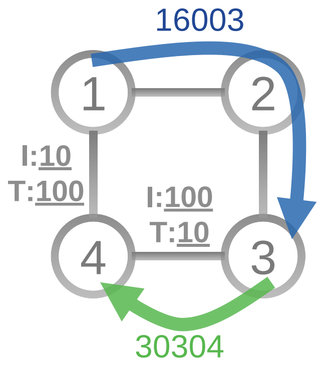
Topological path → SID-list – Example 2

- Desired topological path = 1→2→3→4
- SID-list = <16003, 30304>
  - 16003 brings the packet from 1 to 3 (shortest path from Node1 to Node3)
  - 30304 brings the packet from 3 to 4 using the Adjacency-SID



Topological path to SID-list – TE metric

- Note that the derivation of the SID-list to express a topological path only considers IGP metric, not TE metric
  - Default forwarding uses shortest IGP metric forwarding entries
- Example: shortest TE metric path is 1→2→3→4
  - Cumulative TE metric is 30
  - The IGP metric topology is the same as Example 2 on previous slide
    → resulting SID-list = <16003, 30304>

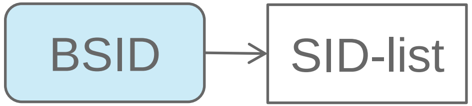


Default IGP link metric: I:10
Default TE link metric: T:10

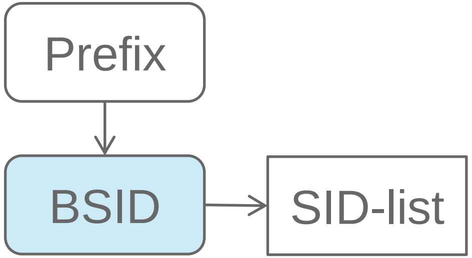
Traffic Steering

Binding-SID (BSID) is fundamental

- The BSID of the SR Policy selected path is installed in the forwarding table
- Remote steering
  - A packet arriving on the SR Policy head-end with the BSID as Active Segment (top of label stack) is steered into the SR Policy associated with the BSID



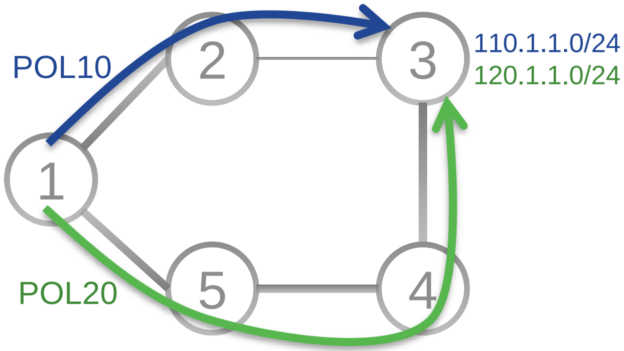
- Local steering
  - A packet that matches a forwarding entry that resolves on the BSID of an SR Policy is steered into that SR Policy



Automated steering

- BGP can automatically steer traffic into an SR Policy based on BGP next-hop and color of a route
  - color of a route is specified by its color extended community attribute
- By default:
  If the BGP next-hop and color of a route match the end-point and color of an SR Policy, then BGP installs the route resolving on the BSID of the SR Policy
  - end-point and color uniquely identify an SR Policy on a given head-end

110.1.1.0/24 (color 10, NH 1.1.1.3)
  via SR Policy POL10 (10, 1.1.1.3)
120.1.1.0/24 (color 20, NH 1.1.1.3)
  via SR Policy POL20 (10, 1.1.1.3)



Color Extended Community attribute

- The Extended Community attribute is specified in RFC 4360
- The color extended community is specified in RFC 5512 and updated in draft-ietf-idr-sr-policy-safi
  - It is a Transitive Opaque Extended community
- CO-bits specify the SR Policy preference (see next slide)
- The color value is a flat 32-bit number

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   type 0x03   | sub-type 0x0b |C O|       Reserved            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                          Color Value                          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Steering – Color-only (CO) bits

- In very specific cases (e.g. SDN-controlled network), an operator may want to steer traffic into an SR Policy based on color only
- This is governed by setting the CO-bits in the color extended community
- By default, CO-bits are 00 and specify the default Automated Steering functionality based on color and nexthop

Steering – Color-only (CO) bits

- Assume route R with next-hop N has a single color C
- The Color-Only (CO) bits in the color extended community attribute flags of R are 00, 01, or 10 (11 is treated as 00)
- BGP steers R according to this preference order:

CO=00 (or CO=11)

Preference:

1. SR Policy(N, C)
2. IGP to N

CO=01

Preference:

1. SR Policy(N, C)
2. SR Policy(null(AF_N), C)
3. SR Policy(null(any), C)
4. IGP to N

CO=10

Preference:

1. SR Policy(N, C)
2. SR Policy(null(AF_N), C)
3. SR Policy(null(any), C)
4. SR Policy(<any(AF_N)>, C)
5. SR Policy(<any(any)>, C)
6. IGP to N

Steering – Color-only (CO) bits – Notes

- Only valid, authorized-to-steer SR Policies are considered for traffic steering
  - Invalid and not authorized-to-steer SR Policies are skipped in the selection
- "IGP to N" is the IGP shortest path to N
- SR Policy(null, C) has a "null end-point"
  - null(AF_N) is the null end-point for the address-family (AF) of N
  - null(any) is the null end-point for any address-family
  - null(IPv4) is 0.0.0.0; null(IPv6) is ::0
- SR Policy(<any>, C) is "any" SR Policy with color C
  - any(AF_N) is any end-point of the address-family of N
  - any(any) is any end-point of any address-family
- Only one SR Policy(N, C) exists on a given node
- Only one SR Policy(null(AF), C) for each AF exists on a given node

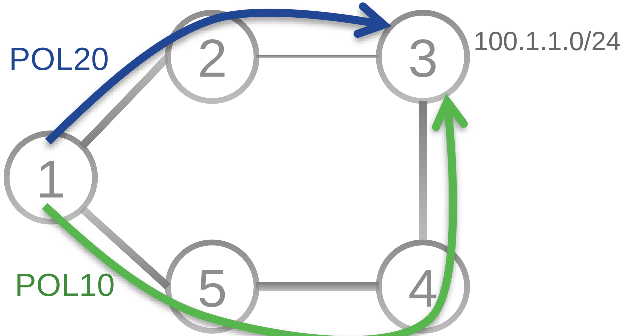
Steering is independent of type of SR Policy

- Steering behavior is absolutely independent of the type/source of the SR Policy
- The SR Policy may have been preconfigured, learned via netconf, PCEP or BGP or on-demand triggered by BGP or another service (LISP)
- Once an SR Policy exists, is valid and authorized to steer, then BGP simply applies the steering preference rules (color value and CO-bits)

Route has multiple colors

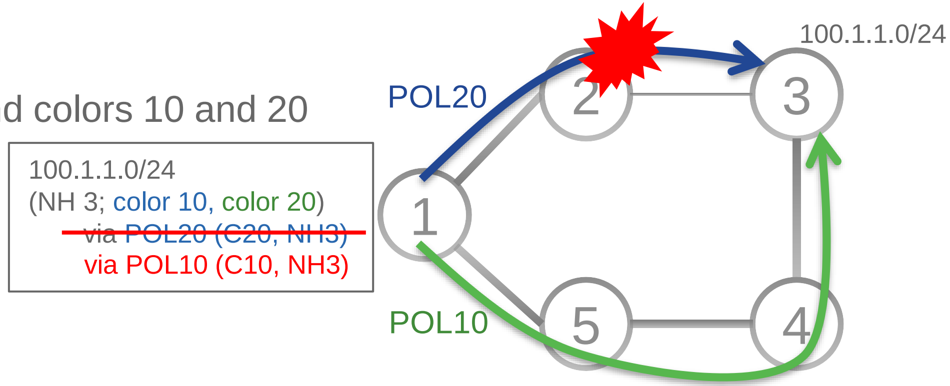
- If a route R with next-hop N has multiple colors C_1 … C_k then BGP steers R into the SR Policy with the numerically highest color
  - Considering only valid and authorized-to-steer SR Policies (C_i,N) with i=1…k
- Example:
  - Node1 receives 100.1.1.0/24 with NH 1.1.1.3 and colors 10 and 20
  - BGP resolves 100.1.1.0/24 on BSID of POL20 (has numerically highest color 20)

100.1.1.0/24 (NH 1.1.1.3; color 20, color 10)
via SR Policy POL20 (20, 1.1.1.3)



Multiple colors for Prim/Secon SR Policies

- Assume route R with next-hop N has colors C_1 , C_2, … ,C_n with $C_i > C_{i+1}$
- SR Policies $(N, C_{1\ldots n})$ are valid and authorized-to-steer
- BGP resolves R on SR Policy (N, C_1) since $C_1 > C_{2\ldots n}$
- If SR Policy (N, C_1) is invalidated, then BGP re-resolves R on SR Policy (N, C_2), with C_2 the next lower numerical color value
- Example:
  - Node1 receives 100.1.1.0/24 with NH 1.1.1.3 and colors 10 and 20
  - BGP resolves 100.1.1.0/24 on BSID of POL20 (color 20 > color 10)
  - After invalidation of POL20, BGP re-resolves 100.1.1.0/24 on BSID of POL10



Disable automated traffic steering

- By default, traffic can be steered on each SR Policy; i.e. each SR Policy is “authorized-to-steer”
- The steering of traffic into a given SR Policy can be disabled by configuration
- Configuration example: disable steering for BGP

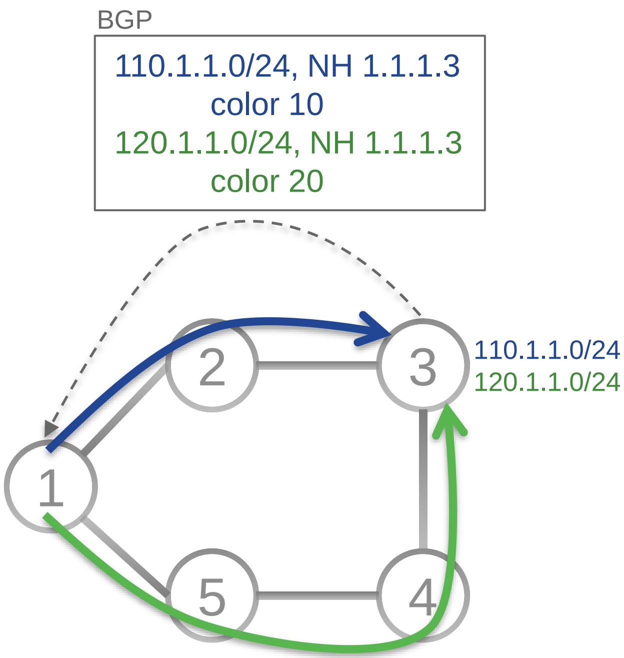
```
segment-routing
 traffic-eng
  policy POLICY1
   color 20 end-point ipv4 1.1.1.4
   steering bgp disable
   candidate-paths
    preference 100
     dynamic
      metric
       type te
```

Setting color of route

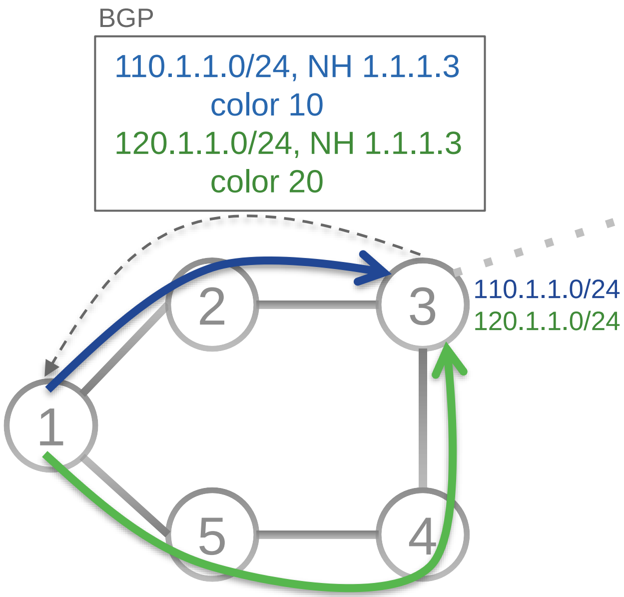
- The color of a BGP route is typically set at the egress PE by adding a color extended community to the route
  - The color extended community is propagated to the ingress PE
  - Traffic steering on the ingress PE is then done automatically based on the color, no route-policy required
- The traffic steering can be influenced on the ingress PE by setting a color extended community for a route using an ingress route-policy

Color assignment on egress PE

- Node1 has two SR Policies with end-point Node3:
  - POL10 with color 10 (blue) via Node2
  - POL20 with color 20 (green) via Node4
- Node3 advertises two prefixes with next-hop 1.1.1.3 in BGP:
  - 110.1.1.0/24 with color 10 (blue)
  - 120.1.1.0/24 with color 20 (green)



Color assignment Egress PE



Node3

```
extcommunity-set opaque BLUE
  10
end-set
!
extcommunity-set opaque GREEN
  20
end-set
!
route-policy SET_COLOR
  if destination in (110.1.1.0/24) then
    set extcommunity color BLUE
  endif
  if destination in (120.1.1.0/24) then
    set extcommunity color GREEN
  endif
end-policy
!
router bgp 1
 neighbor 1.1.1.1
  remote-as 1
  update-source Loopback0
  address-family ipv4 unicast
   route-policy SET_COLOR out
```

RPL attach points to set color ext community

Attach Point	Set
VRF export	✓
VRF import	✗
EVI export	✓
EVI import	✓
Neighbor-in	✓
Neighbor-out	✓
Inter-AFI export	✗
Inter-AFI import	✗
Default-originate	✓

Ingress PE

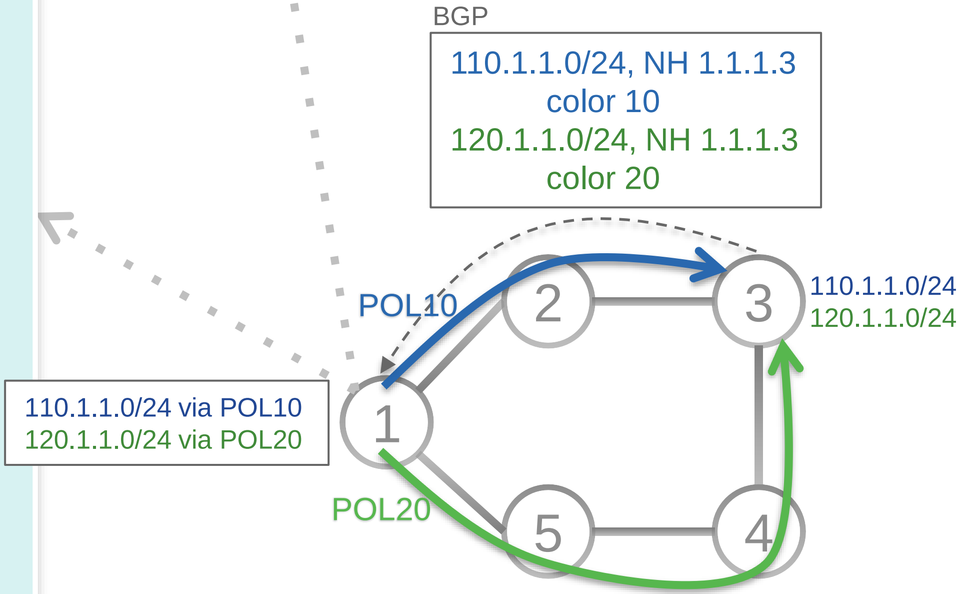
Node1

```
segment-routing
 traffic-eng
  policy POL10
   color 10 end-point ipv4 1.1.1.3
   candidate-paths
    preference 100
     explicit segment-list SIDLIST1
     !
  policy POL20
   color 20 end-point ipv4 1.1.1.3
   candidate-paths
    preference 100
     explicit segment-list SIDLIST2
     !
  segment-list name SIDLIST1
   index 10 address ipv4 1.1.1.3
  !
  segment-list name SIDLIST2
   index 10 address ipv4 1.1.1.4
   index 20 address ipv4 1.1.1.3
```

Node1

```
router bgp 1
 neighbor 1.1.1.3
  remote-as 1
  update-source Loopback0
  address-family ipv4 unicast
```

No route-policy required on Node1!



Color assignment on ingress PE

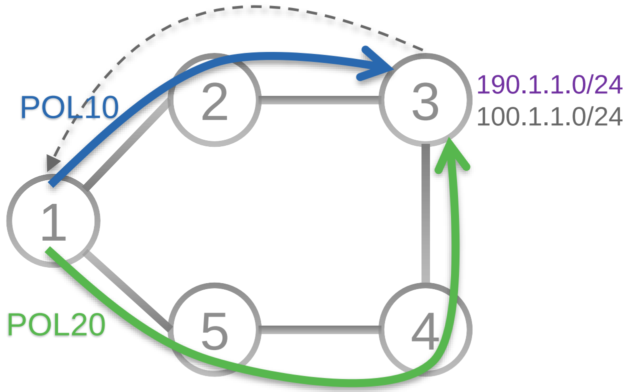
- Node1 has two SR Policies with end-point Node3:
  - POL10 with color 10 (blue) via Node2
  - POL20 with color 20 (green) via Node4
- Node3 advertises two prefixes with next-hop 1.1.1.3 in BGP:
  - 190.1.1.0/24 with color 90 (purple)
  - 100.1.1.0/24 without color
- Node1 sets:
  - color of 190.1.1.0/24 to 10 (blue)
  - color of 100.1.1.0/24 to 20 (green)

BGP

190.1.1.0/24, NH 1.1.1.3
color 90
100.1.1.0/24, NH 1.1.1.3
<no color>

190.1.1.0/24 → color 10
100.1.1.0/24 → color 20

190.1.1.0/24 via POL10
100.1.1.0/24 via POL20



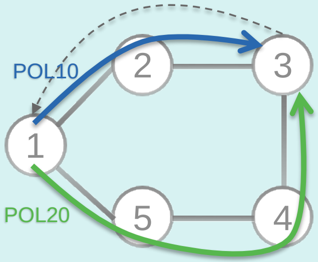
Ingress PE

Node1

```
extcommunity-set opaque BLUE
  10
end-set
!
extcommunity-set opaque GREEN
  20
end-set
!
route-policy SET_COLOR
  if destination in (190.1.1.0/24) then
    set extcommunity color BLUE
  endif
  if destination in (100.1.1.0/24) then
    set extcommunity color GREEN
  endif
end-policy
!
router bgp 1
 neighbor 1.1.1.3
  remote-as 1
  update-source Loopback0
  address-family ipv4 unicast
   route-policy SET_COLOR in
```

BGP

190.1.1.0/24, NH 1.1.1.3
color 90
100.1.1.0/24, NH 1.1.1.3
<no color>



Node1

```
segment-routing
 traffic-eng
  policy POL10
   color 10 end-point ipv4 1.1.1.3
   candidate-paths
    preference 200
     explicit segment-list SIDLIST1
     !
  policy POL20
   color 20 end-point ipv4 1.1.1.3
   candidate-paths
    preference 200
     explicit segment-list SIDLIST2
     !
  segment-list name SIDLIST1
   index 10 address ipv4 1.1.1.3
  !
  segment-list name SIDLIST2
   index 10 address ipv4 1.1.1.4
   index 20 address ipv4 1.1.1.3
```

Pseudowire Preferred path

- The SR Policy used to transport Pseudowire traffic can be specified using the preferred-path configuration
- If using an LDP signaled PW, then the neighbor address must be reachable (via the SR Policy or another path)

```
l2vpn
 pw-class EoMPLS-PWCLASS
  encapsulation mpls
   preferred-path sr-te policy POL1
 !
 xconnect group XCONGRP
  p2p XCON-P2P
   interface TenGigE0/1/0/3
   neighbor ipv4 1.1.1.3 pw-id 1234
    !! below line only if not using LDP
    mpls static label local 2222 remote 3333
    pw-class EoMPLS-PWCLASS
```

On-Demand Nexthop (ODN)

On-Demand Nexthop

- A service head-end automatically instantiates an SR Policy to a BGP next-hop when required (on-demand)
- Color community is used as SLA indicator
- Reminder: an SR Policy is defined (color, end-point)

BGP Color Community

BGP Next-hop

- Automated Steering (AS) automatically steers the BGP traffic into this SR Policy, also based on nexthop and color

On-demand SR Policy

- Configure an SR Policy template for each color for which on-demand SR Policy instantiation is desired
- An example with two color templates configured:
  - color 10 for high bandwidth (optimize IGP metric)
  - color 20 for low-delay (optimize link-delay metric)

```
segment-routing
 traffic-eng
  on-demand color 10
   dynamic
    metric type igp
  !
  on-demand color 20
   dynamic
    metric type latency
```

SR Policy template High-BW (color 10)

SR Policy template low-delay (color 20)

On-demand SR Policy

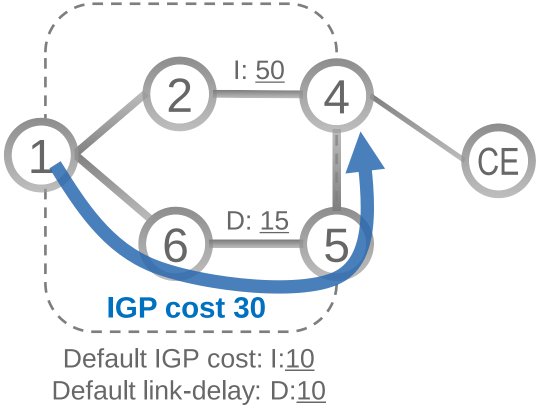
- If an SR Policy template exists for color C, then a route with color C can trigger an on-demand SR Policy candidate path instantiation to its next-hop N, for any N
- The end-points for which an on-demand SR Policy candidate path will be instantiated can be restricted per color
- Example configuration: only instantiate color 10 SR Policies for end-points 1.1.1.10 and 1.1.1.11

```
ipv4 access-list ACL1
 10 permit ipv4 host 1.1.1.10 any
 20 permit ipv4 host 1.1.1.11 any
!
segment-routing
 traffic-eng
  on-demand color 10
   restrict ACL1
   dynamic
    metric type latency
```

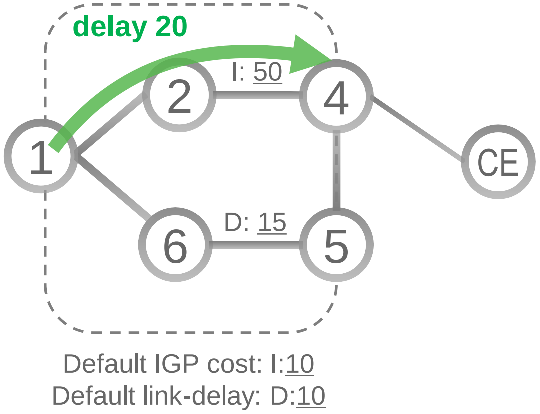
Automated Steering

- Service traffic is automatically steered on the right SLA path
  - Steered into an SR Policy based on color and next-hop of the service route
  - Independent from ODN: SR Policy can already exist or be instantiated on-demand (ODN) when receiving the service route update
- Color community of the service route is used as SLA indicator
- Simple and Performant

Different VPNs need different underlay SLA



Basic VPN should use lowest cost underlay path



Premium VPN should use lowest delay path

Objective: operationalize this service for simplicity, scale and performance

animated

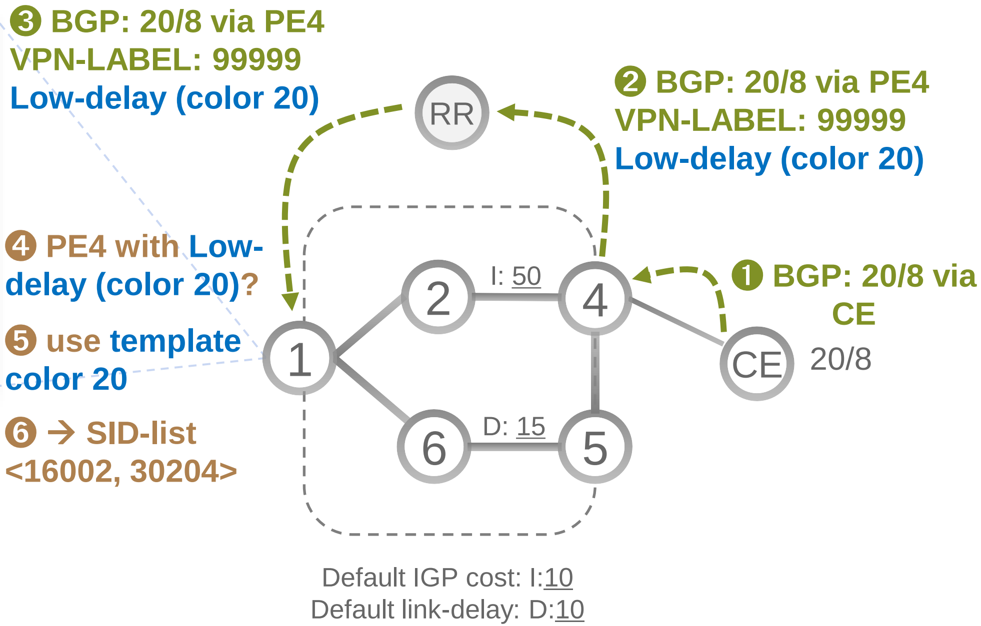
On-demand SR Policy work-flow

❺

```
router bgp 1
 neighbor 1.1.1.10
  address-family vpnv4 unicast
!
segment-routing
 traffic-eng
  on-demand color 20
   dynamic
    metric
     type latency
```

SR Policy template
Low-delay (color 20)

no route-policy required!



Performant Automated Steering

❼❽

FIB table at PE1
BGP: 20/8 via **4001**
SRTE: **4001**: Push <16002, 30204>

Automatically, the service route resolves on the Binding-SID (4001) of the SR Policy it requires

Simplicity and Performance

No complex PBR to configure, no PBR performance tax

❸ **BGP: 20/8 via PE4**
VPN-LABEL: 99999
Low-delay (color 20)

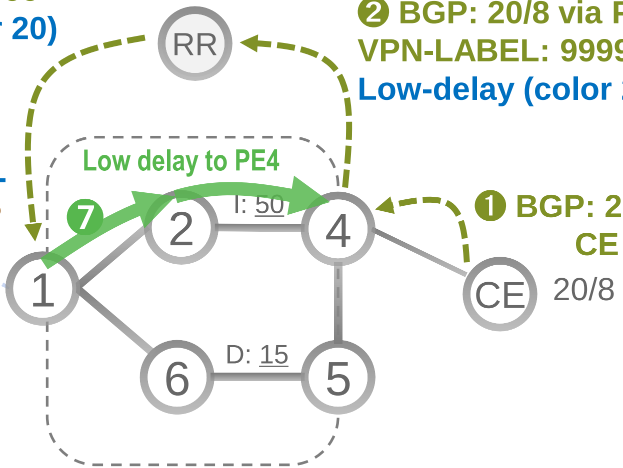
❹ **PE4 with Low-delay (color 20)?**

❺ **use template color 20**

❻ **→ SID-list <16002, 30204>**

❼ **instantiate SR Policy BSID 4001**

❽ **forward 20/8 via BSID 4001**



❷ **BGP: 20/8 via PE4**
VPN-LABEL: 99999
Low-delay (color 20)

❶ **BGP: 20/8 via CE**

Default IGP cost: I:10
Default link-delay: D:10

Benefits

- SLA-aware BGP service
- No a-priori full-mesh of SR Policy configuration
  - 3 to 4 common optimization templates are used throughout the network
    - color → optimization objective
- No complex steering configuration
  - Automated Steering of BGP routes on the right SLA path
  - Data plane performant
  - BGP PIC FRR data plane protection is preserved
  - BGP NHT fast control plane convergence is preserved

PIC FRR = Prefix Independent Convergence Fast ReRoute
NHT = Next-Hop Tracking

Multi-domain On-Demand Nexthop (ODN)

On-Demand Nexthop – multi-domain

- The On-Demand Nexthop and Automated Steering (AS) functionalities also apply to multi-domain networks

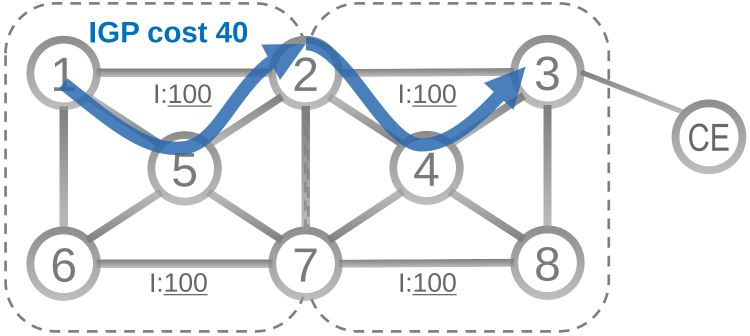
On-Demand Nexthop – multi-domain

- On-demand Nexthop automatically provides inter-domain best-effort reachability and inter-domain reachability with SLA
- Head-end uses SR PCE to automatically provide an SR Policy path to the remote domain destination when needed (On-demand)
- Scaling benefit
  - On-Demand Nexthop: on-demand pull model
  - Classic inter-domain reachability uses a push model
  - Think of a large-scale aggregation with 100k access nodes where each such node only needs to talk to 10’s of other nodes

On-Demand Nexthop – workflow

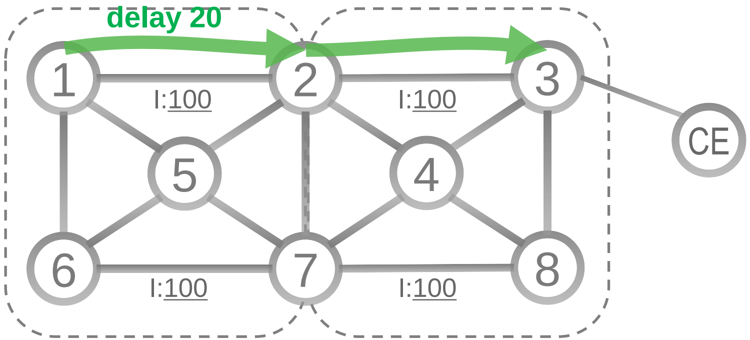
- Service head-end receives an overlay route to a remote service end-point
  - The overlay route can indicate a certain required SLA
- The On-demand Nexthop function automatically sends a stateful PCEP Path Computation request to SR PCE
  - PCEP Request includes the Optimization Objective and Constraints to satisfy the required SLA
- SR PCE computes the inter-domain path to the remote end-point with SLA

On-Demand Nexthop



Default IGP link metric: I:10
Default link-delay metric: D:10

Basic VPN should use best-effort (lowest cost) inter-domain underlay path



Default IGP link metric: I:10
Default link-delay metric: D:10

Premium VPN should use lowest delay inter-domain underlay path

Objective: operationalize this service for simplicity, scale and performance

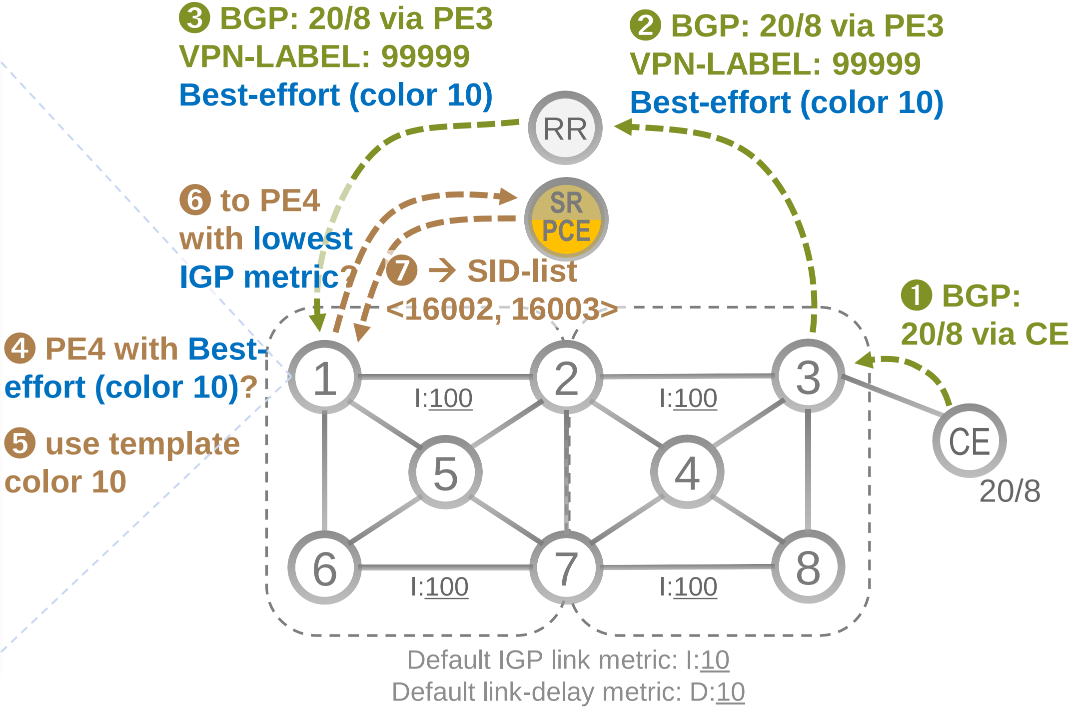
animated

On-Demand Nexthop reachability

5

```
router bgp 1
 neighbor 1.1.1.10
  address-family vpnv4 unicast
!
segment-routing
 traffic-eng
  on-demand color 10
   dynamic
    pcep
    metric
     type igp
  !
  on-demand color 20
   dynamic
    pcep
    metric
     type latency
```

SR Policy template
Best-effort (color 10)



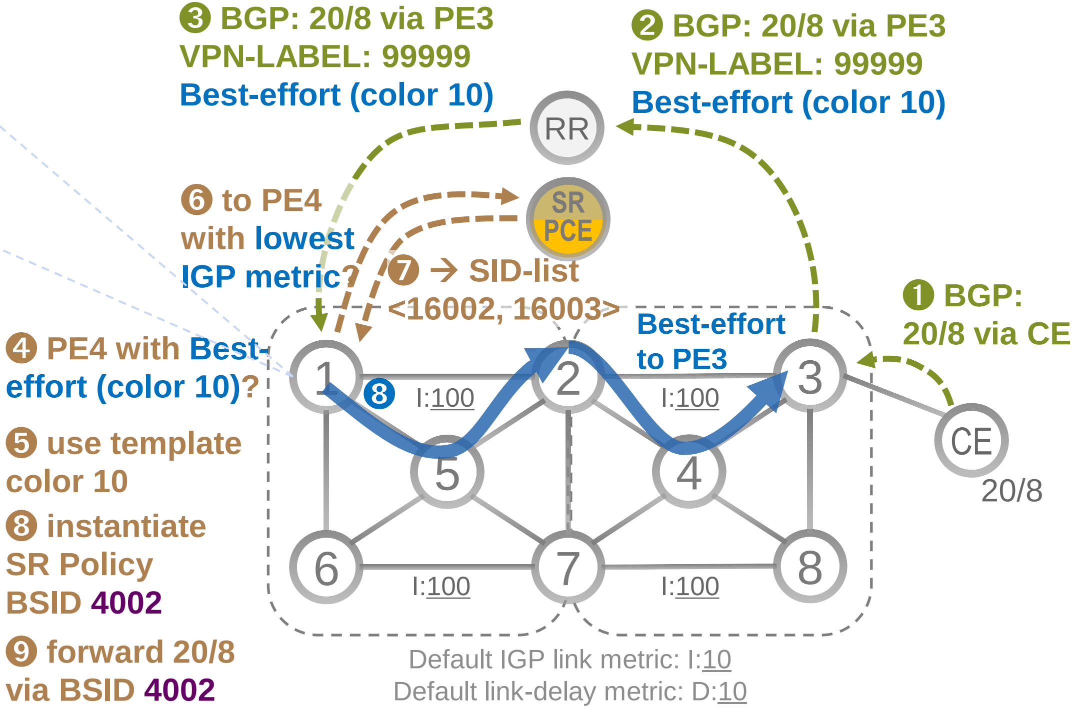
cisco

animated

On-Demand Nexthop reachability

❽❾

FIB table at PE1
BGP: 20/8 via **4002**
SRTE: **4002**: Push <16002, 16003>



Automatically, the service route resolves on the Binding SID (4002) of the SR Policy it requires

Simplicity and Performance

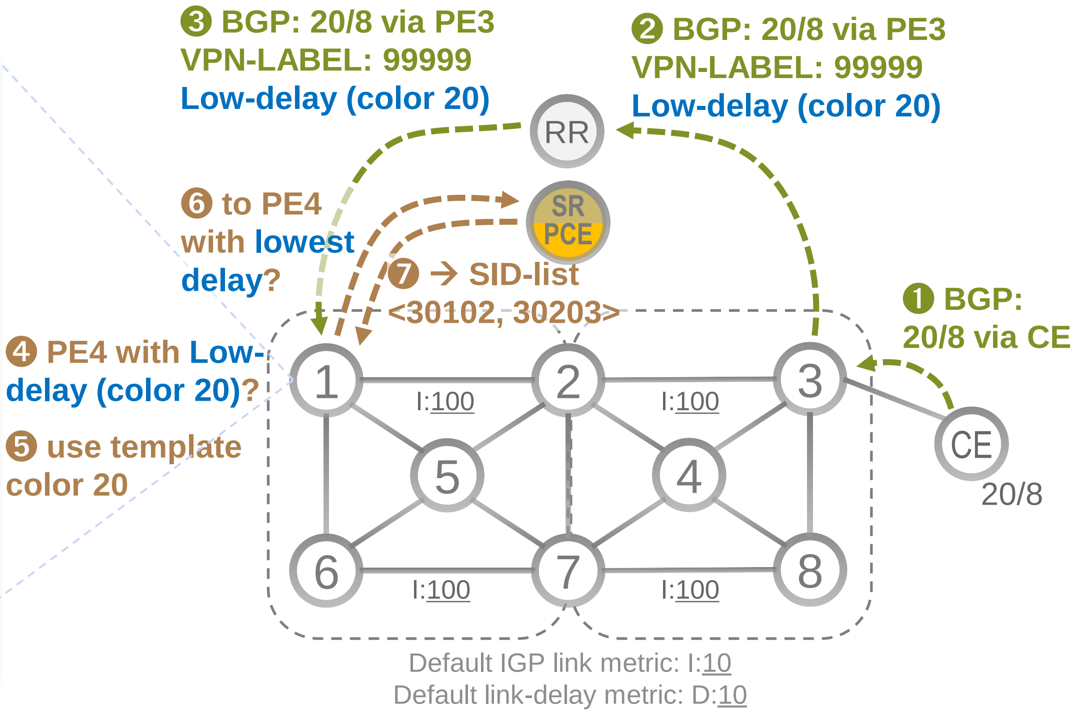
No complex PBR to configure, no PBR performance tax

On-Demand Nexthop with SLA

❺

```
router bgp 1
 neighbor 1.1.1.10
  address-family vpnv4 unicast
!
segment-routing
 traffic-eng
  on-demand color 10
   dynamic
    pcep
    metric
     type igp
   !
  on-demand color 20
   dynamic
    pcep
    metric
     type latency
```

SR Policy template
Low-delay (color 20)



animated

On-Demand Nexthop with SLA

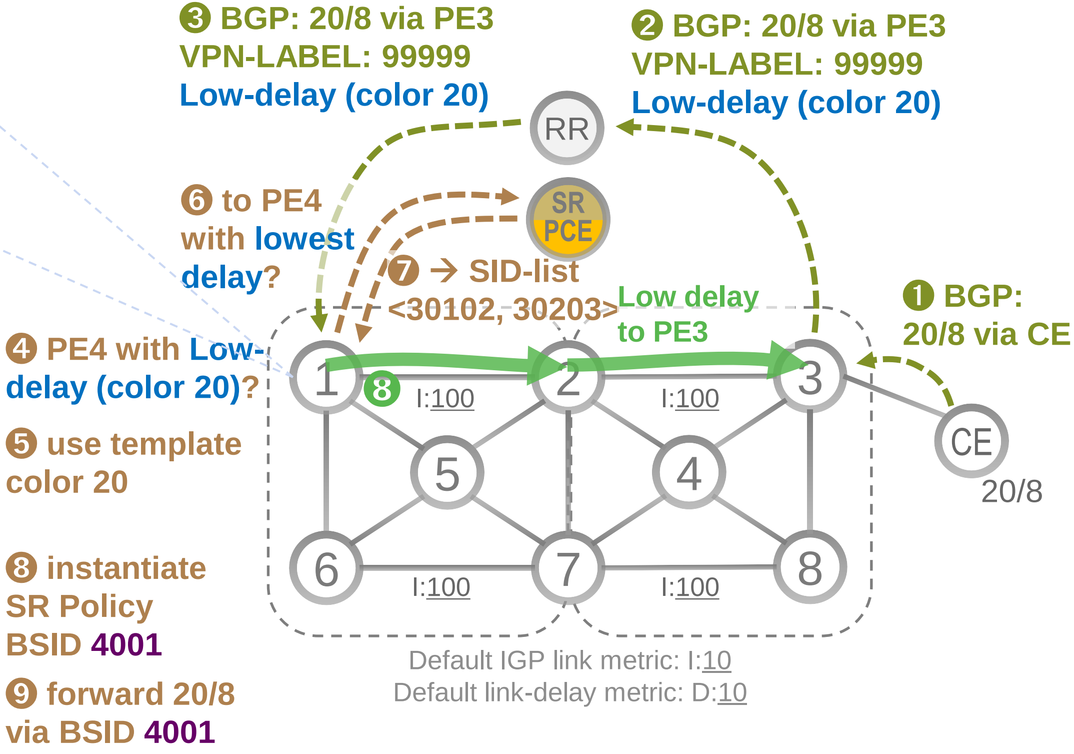
❽❾

FIB table at PE1
BGP: 20/8 via **4001**
SRTE: **4001**: Push <30102, 30203>

Automatically, the service route resolves on the Binding SID (4001) of the SR Policy it requires

Simplicity and Performance

No complex PBR to configure, no PBR performance tax



Benefits

- Scalable – PE1 only gets the inter-domain paths that it needs
- Simple – no BGP3107 pushing all routes everywhere
- No complex steering configuration
  - Automated steering of BGP routes on the right SLA path
  - Data plane performant

* BGP3107 = BGP Labeled Unicast (BGP-LU) according to RFC3107, now obsoleted by RFC8277, often used in “seamless MPLS” deployments (draft-ietf-mpls-seamless-mpls)

Dynamic Path
Distributed or Centralized?

Distributed and Centralized

- There are two possibilities to compute the dynamic path for an SR Policy:
  - Head-end computes the path itself (“distributed”)
  - Head-end requests SR PCE to compute the path (“centralized”*)
- By default, dynamic paths are computed by the head-end
- Head-end uses SR PCE when local computation is not possible
  - SR PCE is required if more information is needed than is available on a head-end; e.g. multi-area/domain paths, or disjoint paths from different head-ends

* “centralized” indicates SR PCE’s capability (having more information), not its position in the network. SR PCE is natively distributed as indicated in the SR PCE section

Head-end and SR PCE: same algorithms

- Head-end and SR PCE use the same SR-optimized computation algorithms

Path Computation

Distributed or Centralized?

- SRTE supports each model where it makes sense

Policy	Single-Domain	Multi-Domain
Reachability	IGPs	**Centralized**
Low delay	Distributed or Centralized	**Centralized**
Disjoint from same node	Distributed or Centralized	**Centralized**
Disjoint from different node	**Centralized**	**Centralized**
Avoiding resources	Distributed or Centralized	**Centralized**
Capacity optimization	Distributed (limited) **Centralized**	**SR PCE**
Maintenance	**Centralized**	**WAE, REX, ODL, Custom app**
Multi-Topology (IP+Optical)	**Centralized**	

SR PCE

SR Path Computation Element (SR PCE)

- SR PCE is an IOS XR multi-domain stateful SR-optimized PCE
  - IOS XR: SR PCE functionality is available on any physical or virtual IOS XR node, activated with a single configuration command
  - Multi-domain: Real-time reactive feed via BGP-LS/ISIS/OSPF from multiple domains; computes inter-area/domain/AS paths
  - Stateful: takes control of SRTE Policies, updates them when required
  - SR-optimized: native SR-optimized computation algorithms
- SR PCE is fundamentally distributed
  - Not a single all-overseeing entity (“god box”), but distributed across the network; RR-alike deployment

SR PCE – IOS XR PCE

- SR PCE functionality is available in IOS XR base image
  - Physical and virtual IOS XR devices
- Enable it by configuring its PCEP* session IP address

On SR PCE:

```
pce
 address ipv4 1.1.1.3
!
```

* Path Computation Element Protocol

SR PCE – Real-time Topology Feed

- SR PCE learns real-time topologies via BGP-LS and/or IGP
- BGP-LS is intended to carry link-state topology information
  - Hence the name “LS” that stands for “Link State”
- BGP-LS has been extended multiple times in order to incorporate other types of topology information:
  - SR Extensions
  - Traffic Engineering Metric Extensions
  - Egress Peer Engineering
  - SR TE Policies

Same multi-domain SRTE DB

- SR PCE uses the same multi-domain SRTE DB as the head-end
  - SR PCE can learn an attached domain topology via its IGP or a BGP-LS session

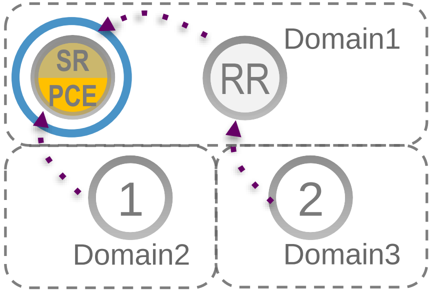
On SR PCE:

```
router isis SR     !! or ospf
 distribute link-state instance-id 32
```

  - SR PCE can learn a non-attached domain topology via a BGP-LS session
    > Direct session or via BGP Route-reflector (RR)

On SR PCE:

```
router bgp 1
 address-family link-state link-state
 !
 neighbor 1.1.1.1
  remote-as 1
  update-source Loopback0
  address-family link-state link-state
```

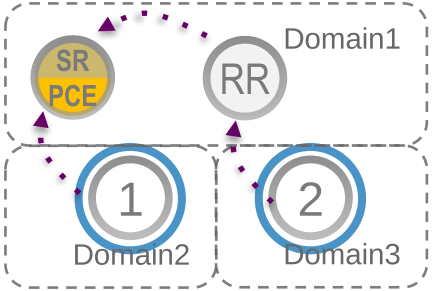


Same multi-domain SRTE DB

- A node that feeds the IGP link-state database in BGP-LS has the following configuration:

On Node1 or Node2:

```
router isis SR    !! or ospf
 distribute link-state instance-id 32
!
router bgp 1
 address-family link-state link-state
 !
 neighbor 1.1.1.10   !! SR PCE or RR
  remote-as 1
  update-source Loopback0
  address-family link-state link-state
```



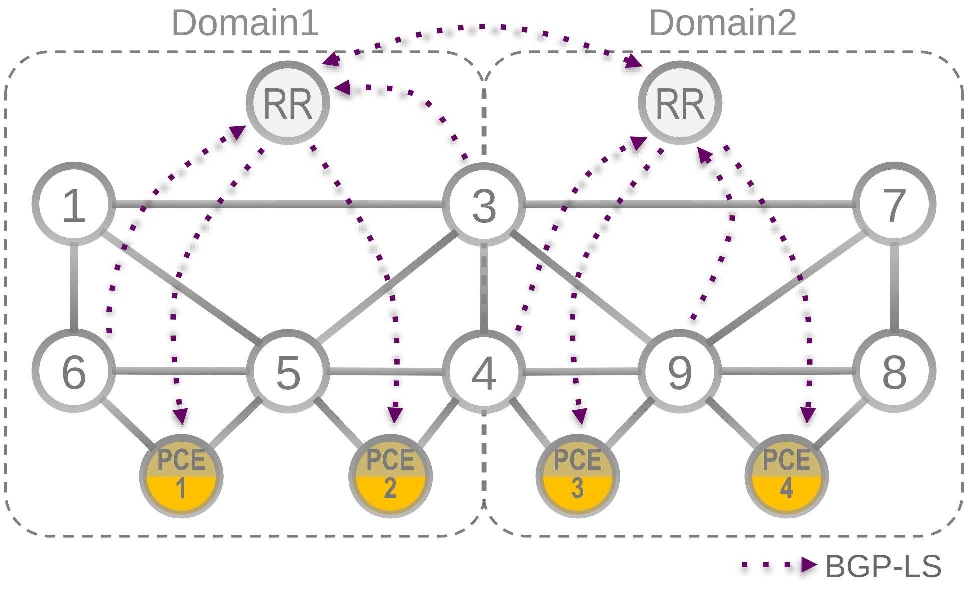
- The illustrations use iBGP BGP-LS sessions, but eBGP is supported as well

SR PCE – Multi-domain real-time topology feed

- SR PCE receives real-time reactive feeds via BGP-LS from multiple domains
  - One or more nodes in each domain feed the topology information via BGP-LS, including IP addresses and SIDs
  - AS peering nodes advertise their peering links information in BGP-LS (Egress Peer Engineering)
  - BGP RRs can be used to scale the BGP-LS feed to the SR PCE nodes (regular BGP functionality)
- SR PCE combines the different information feeds to form a real-time consolidated view of the entire topology
- SR PCE uses this complete topology for path computation

BGP-LS feed to SR PCE

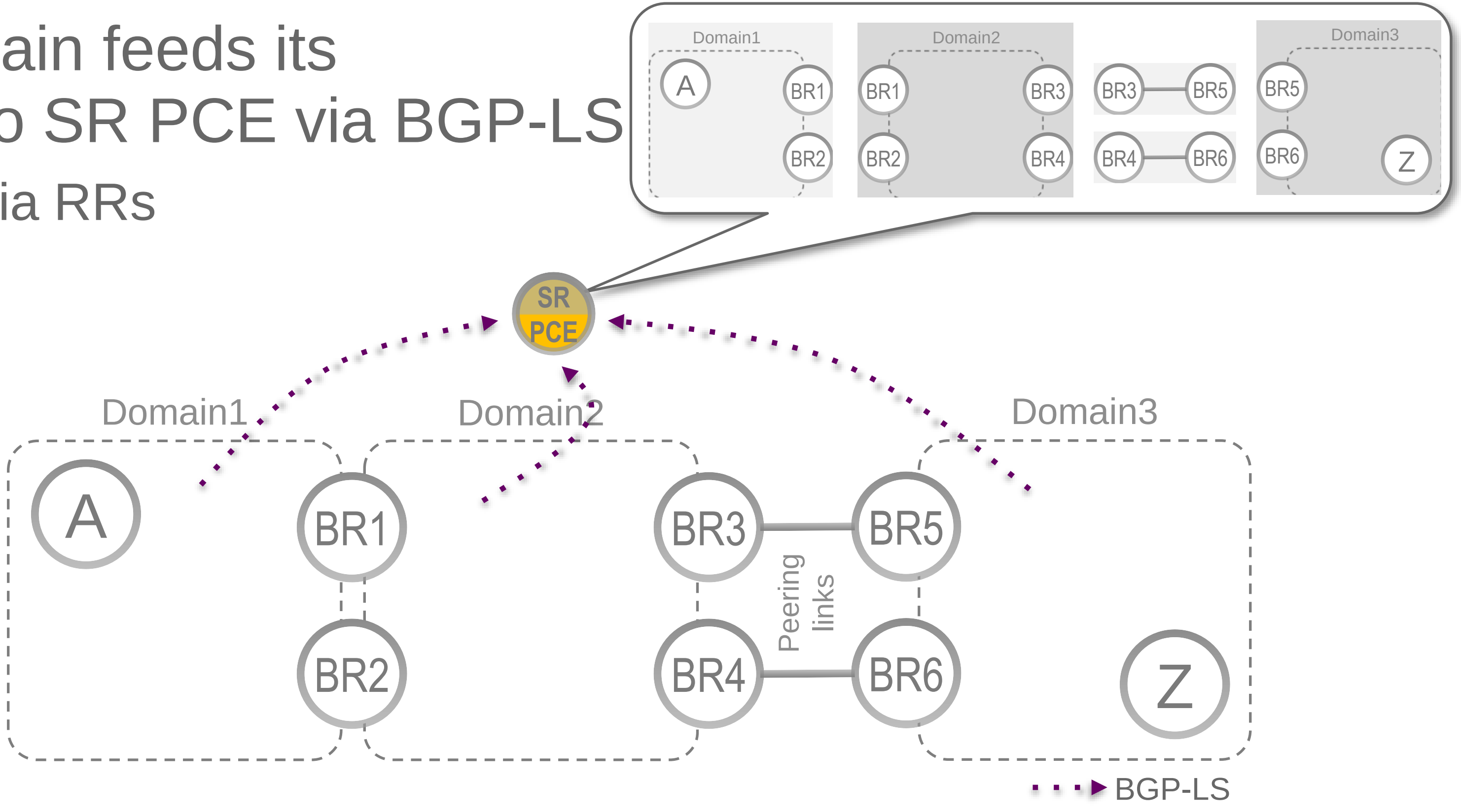
- Typically, BGP RRs are used to scale BGP-LS feeds
- Any node can have a BGP-LS session to the RR
  - Any node can feed its local IGP topology via BGP-LS
  - Peering nodes can feed their EPE information via BGP-LS



In this illustration, Node6 and Node3 distribute Domain1's topology in BGP-LS, Node4 and Node9 distribute Domain2's topology in BGP-LS

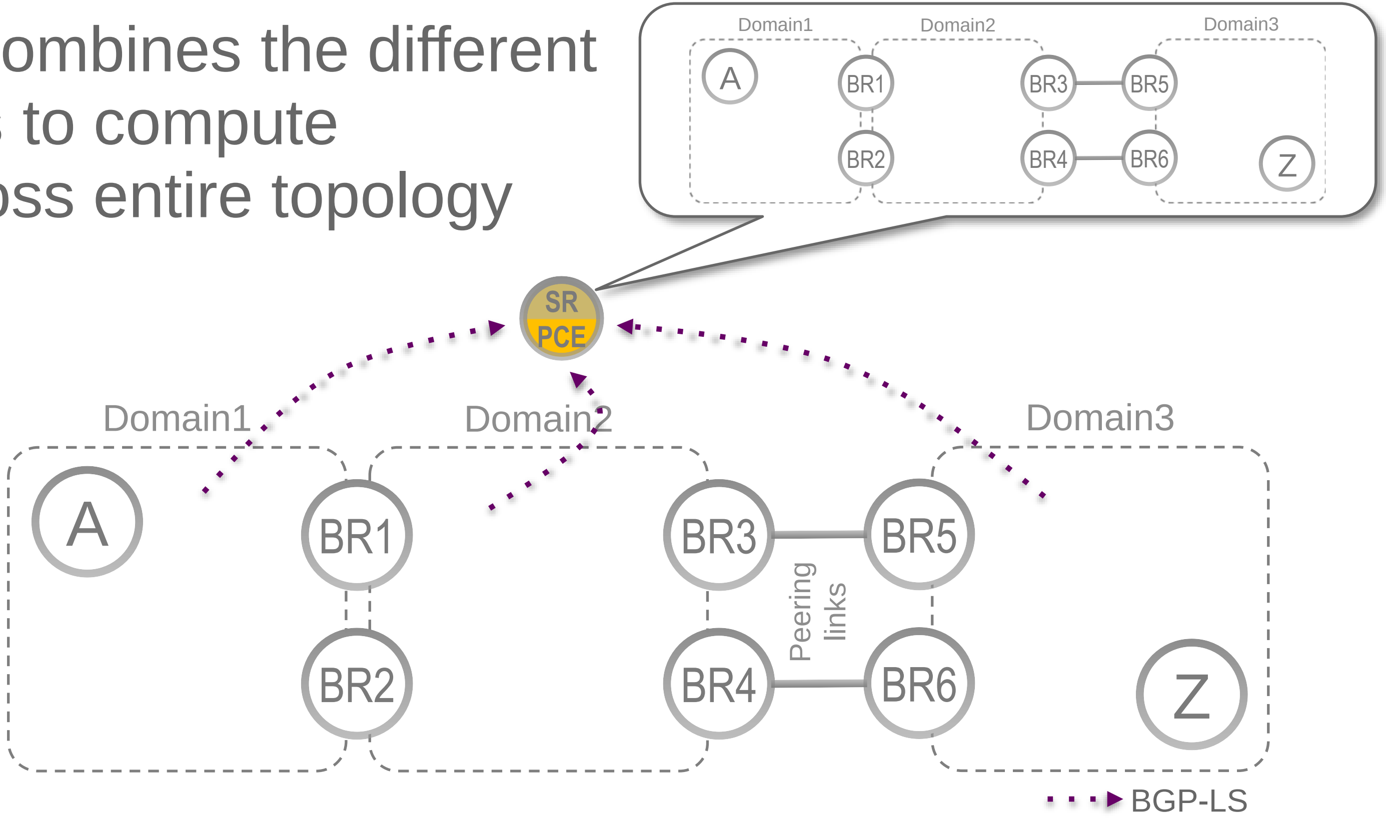
SR PCE receives topology of all domains

- Each domain feeds its topology to SR PCE via BGP-LS
  - Typically via RRs



SR PCE consolidates the topologies

- SR PCE combines the different topologies to compute paths across entire topology



SR PCE and Multi-domain – Notes

- When advertising multiple topologies/domains in BGP-LS, each topology/domain must have a unique instance-id
  - Instance-id identifies a “routing universe”
  - Default: 0 – Value range ISIS: <2-65535>; OSPF: <0-4294967295>
  - Values 1-31 should not be used
    - RFC7752: Values in the range 32 to 2^{64}-1 are for "Private Use"

For example, on the BGP-LS node in Domain1:

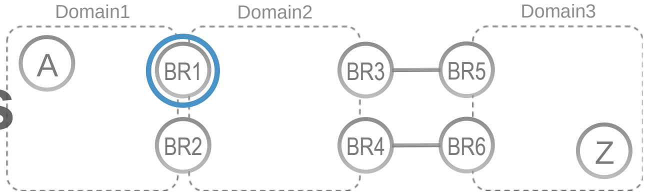
```
router isis Domain1
 distribute link-state instance-id 32
```

Unique instance-id

For example, on the BGP-LS node in Domain2:

```
router isis Domain2
 distribute link-state instance-id 33
```

SR PCE and Multi-domain – Notes



- SR PCE identifies border nodes by a common TE router-id advertised in multiple domains
- Border nodes should advertise the same TE router-id and TE router-id prefix reachability in all its attached domains (i.e. all its IGP instances)

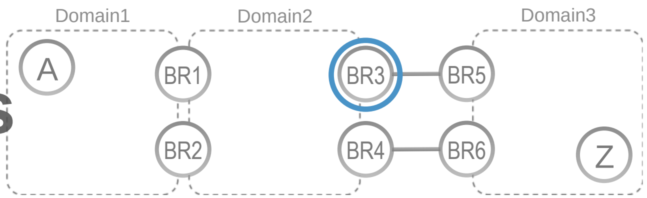
Border node BR1:

```
router isis Domain1
 net 49.0001.1111.0000.0001.00
 address-family ipv4 unicast
  router-id Loopback0
 !
 interface Loopback0
  passive
  address-family ipv4 unicast
   prefix-sid absolute 16001
!
```

```
router isis Domain2
 net 49.0001.2222.0000.0001.00
 address-family ipv4 unicast
  router-id Loopback0
 !
 interface Loopback0
  passive
  address-family ipv4 unicast
   prefix-sid absolute 16001
```

Common TE router-id

SR PCE and Multi-domain – Notes



- SR PCE uses BGP router-id and TE router-id to identify inter-AS border nodes and peering sessions
- Peering nodes should use the same router-id for TE and BGP

Border node BR3:

```
interface Loopback0
 ipv4 address 1.1.1.3/32
!
router isis Domain2
 net 49.0001.3333.0000.0003.00
 address-family ipv4 unicast
  router-id Loopback0
 !
 interface Loopback0
  passive
  address-family ipv4 unicast
   prefix-sid absolute 16003
!
```

```
router bgp 2
 bgp router-id 1.1.1.3
 address-family ipv4 unicast
 !
 neighbor 99.3.5.5
  remote-as 3
  address-family ipv4 unicast
   route-policy bgp_in in
   route-policy bgp_out out
```

IGP TE RID == BGP RID

RID = router-id

Same computation algorithms

- SR PCE uses the same SR-optimized computation algorithms as the head-end

SR PCE computes dynamic path

- A head-end requests SR PCE to compute a dynamic path
- Request/Reply/Report or Report/Update/Report workflow is used
  - IOS XR headend uses the Report/Update/Report workflow
- SR PCE is stateful, it maintains the path, updating the path when required (e.g. after topology change)

SR PCE computes dynamic path

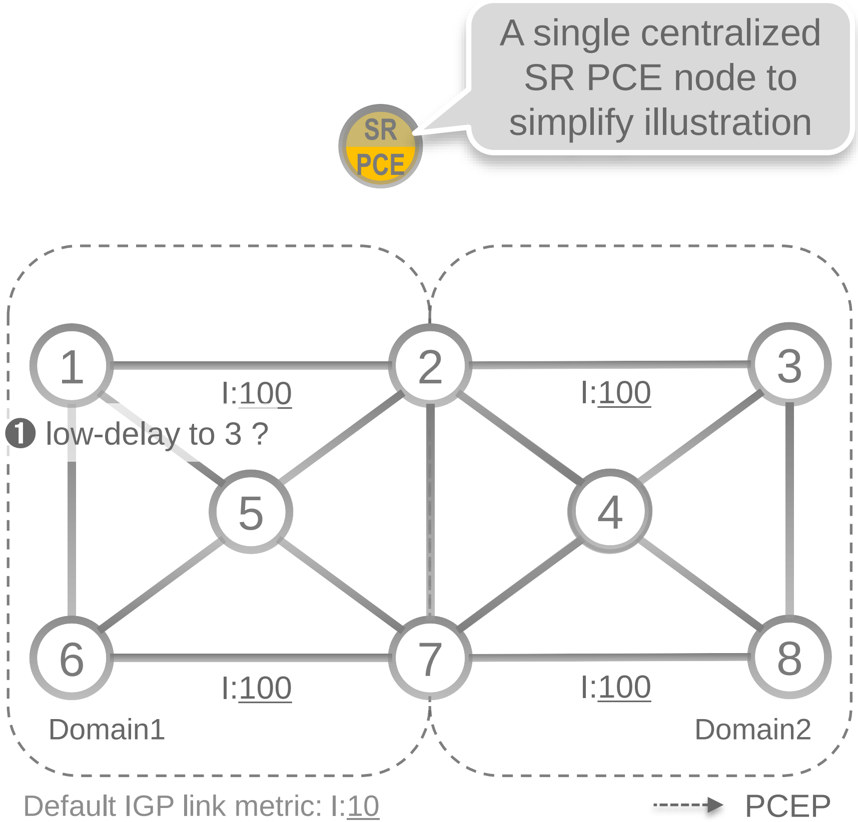
- Request/Reply/Report workflow:
  - head-end requests SR PCE to compute a path
    - Head-end provides optimization objective and constraints to SR PCE
  - SR PCE computes path, derives SID-list and replies to head-end
  - Head-end programs SID-list and reports it to SR PCE
    - Head-end delegates the path to SR PCE

SR PCE computes dynamic path

- Report/Update/Report workflow:
  - head-end reports empty path to SR PCE
    - > Head-end delegates the path to SR PCE
    - > Head-end provides optimization objective and constraints to SR PCE
  - SR PCE computes path, derives SID-list and updates path on head-end
  - Head-end programs SID-list and reports it to SR PCE
    - > Head-end delegates the path to SR PCE
- IOS XR headend uses this Report/Update/Report workflow

Request/Reply/Report workflow

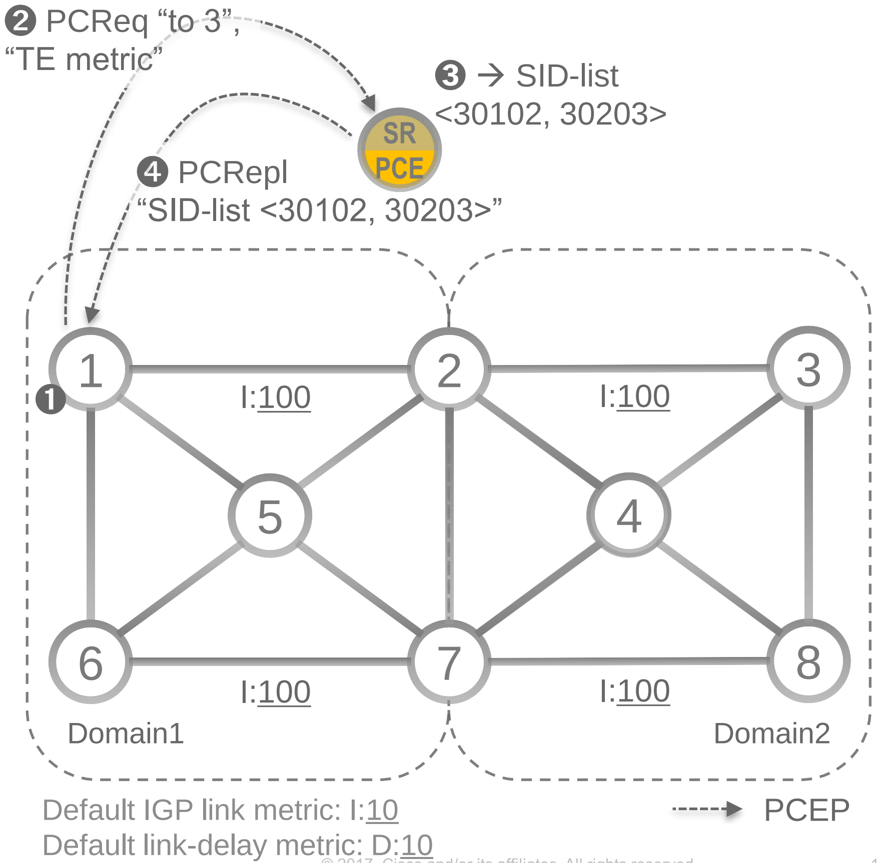
- ➊ Node1 is configured to instantiate a low-delay SR Policy to Node3, e.g. by Network Service Orchestrator (NSO), or a human operator
- Since the end-point Node3 is in a remote domain, Node1 cannot compute the dynamic path locally and must use SR PCE



Default IGP link metric: I:10
Default link-delay metric: D:10

Request/Reply/Report workflow (Cont.)

- ❷ Node1 sends a PCEP Path Computation Request (PCReq) to SR PCE, requesting path “to Node3” with “Optimize TE metric”
- ❸ SR PCE stores the request and computes a TE metric shortest-path from Node1 to Node3, say the resulting SID list is <30102, 30203>
- ❹ PCE sends “SID list <30102, 30203>” to Node1 in PCEP Path Computation Reply (PCRepl)

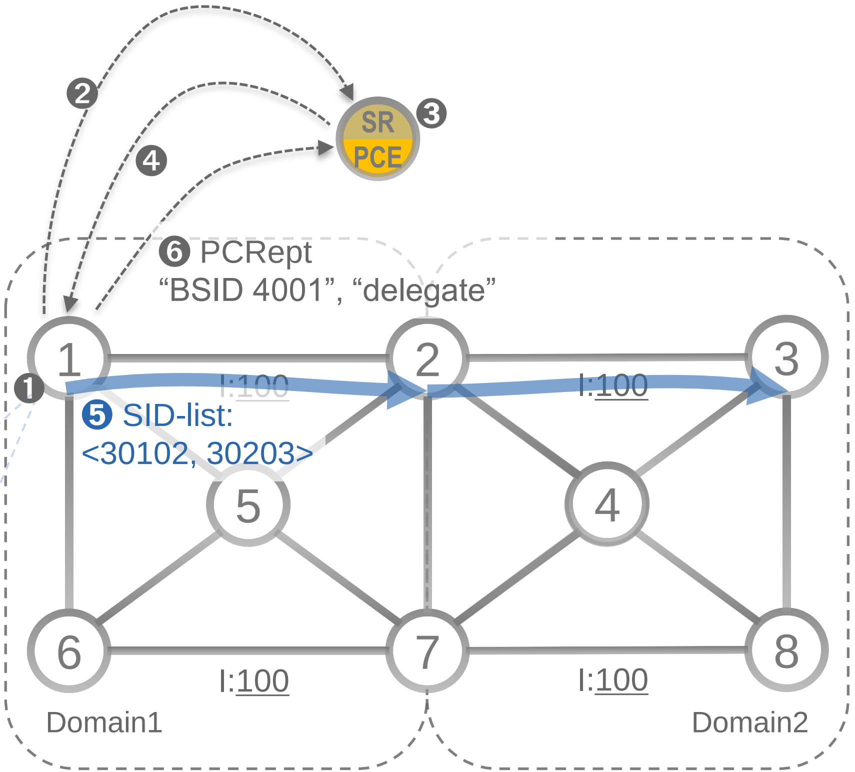


Request/Reply/Report workflow (Cont.)

- ❺ Node1 allocates a BSID 4001 and activates the SR Policy path to Node3 via <30102, 30203>
- and ❻ sends Path Computation Report (PCRpt) to SR PCE, delegating the SR Policy to SR PCE and including BSID

FIB table at Node1
SRTE: 4001: Push <30102, 30203>

BSID



Default IGP link metric: I:10
Default link-delay metric: D:10

PCEP

Decouple overlay/underlay

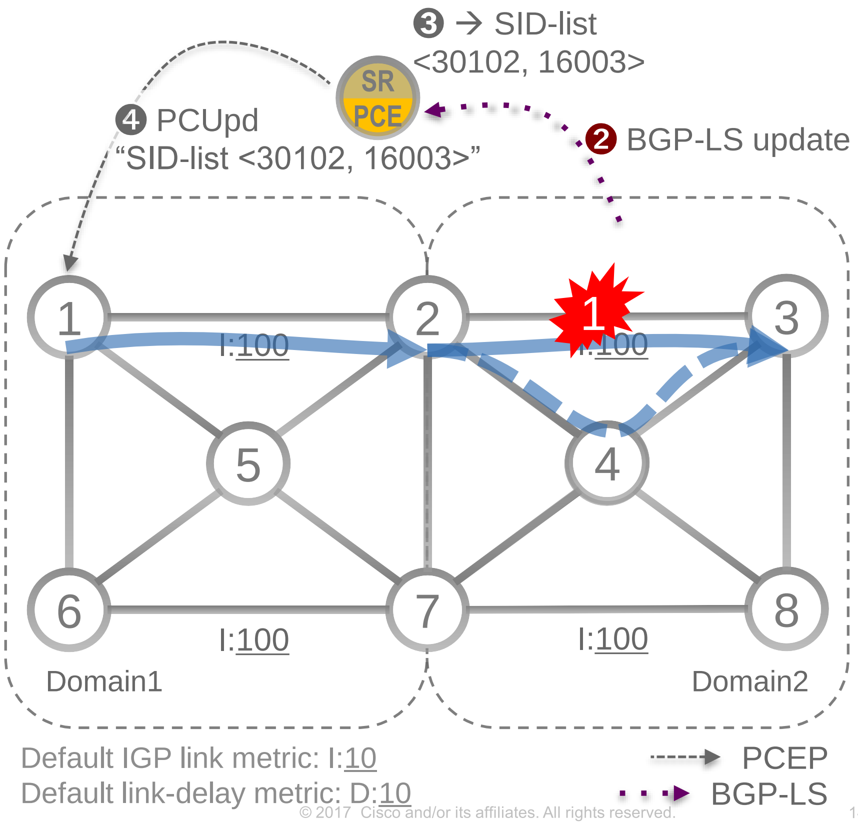
- The Request/Reply model separates the service creation and maintenance (overlay) from the topology and path maintenance (underlay)
  - NSO (“Overlay Controller”) does not need to be aware of the topology
  - SR PCE (“Underlay Controller”) is not aware of the service, SR Policy and traffic steering configuration
  - NSO does not need to interact directly with SR PCE; Overlay Controller is decoupled from Underlay Controller

SR PCE – Stateful

- SR PCE stores path computation requests (stateful)
  - Request includes optimization objective and constraints
- SR PCE has control over the paths delegated to it
- SR PCE updates the paths when required, e.g. following a multi-domain topology change that impacts connectivity
  - Anycast-SIDs and Local FRR (TI-LFA) minimize traffic loss during the stateful re-optimization

Stateful – SR PCE updates path

- ❶ A topology change occurs in Domain2
- TI-LFA protects traffic within 50ms
- ❷ BGP-LS pushes the topology change to SR PCE
- ❸ SR PCE re-computes path; the new SID-list is <30102, 16003>
- ❹ SR PCE sends PCUpd message with "SID list <30102, 16003>" to Node1

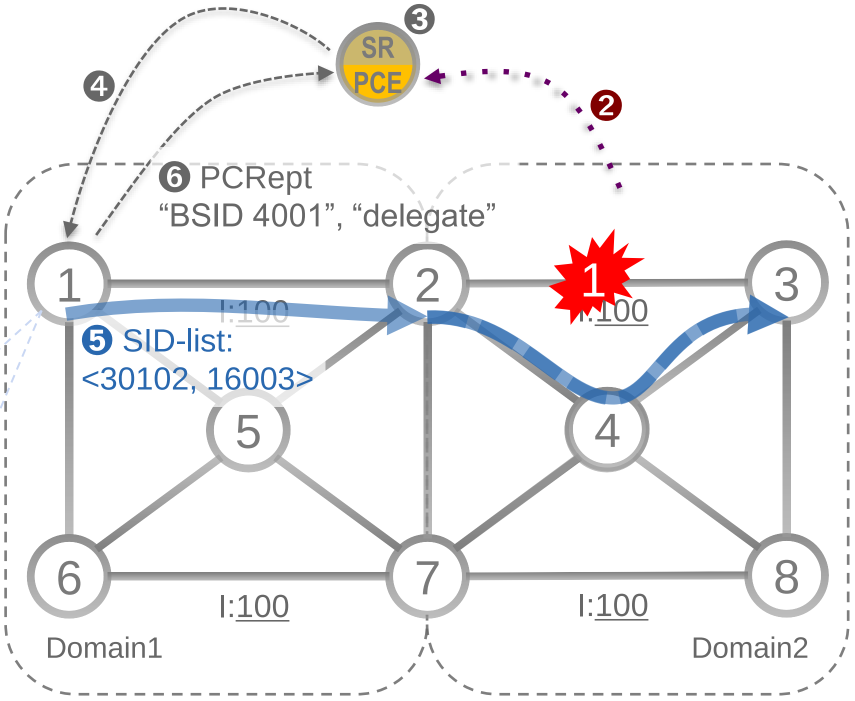


Stateful – SR PCE updates path

- ❺ Node1 updates SR Policy Path via <30102, 16003>, maintaining the BSID 4001
- and ❻ sends Path Computation Report (PCRpt) to SR PCE, delegating the SR Policy to SR PCE and including BSID

FIB table at Node1
SRTE: 4001: Push <30102, 16003>

BSID



Default IGP link metric: I:10
Default link-delay metric: D:10

PCEP
BGP-LS

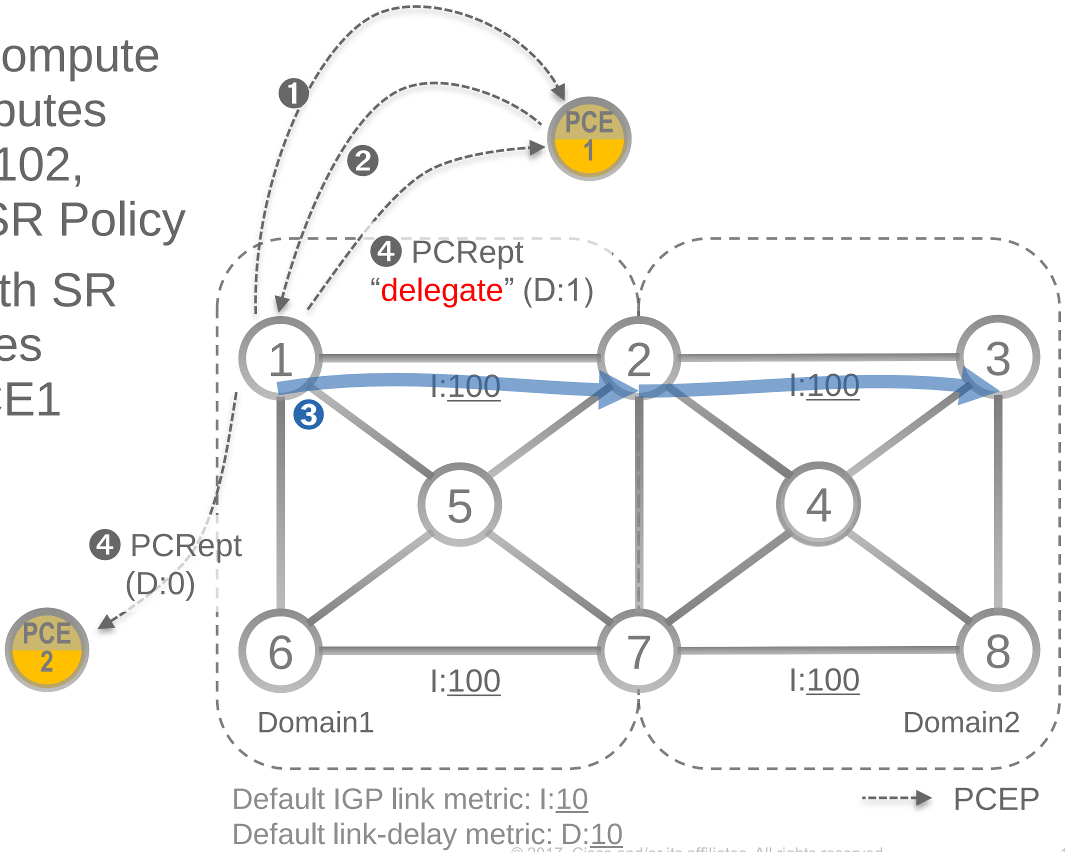
cisco

SR PCE – High Availability (HA)

- SR PCE leverages the well-known standardized PCE HA
- Head-end sends PCEP Report for its SR Policies to all connected SR PCE nodes
- Head-end delegates control to its primary SR PCE
  - Delegate flag (D) is set in PCRept to primary SR PCE
- Upon failure of the primary SR PCE, head-end re-delegates control to another SR PCE

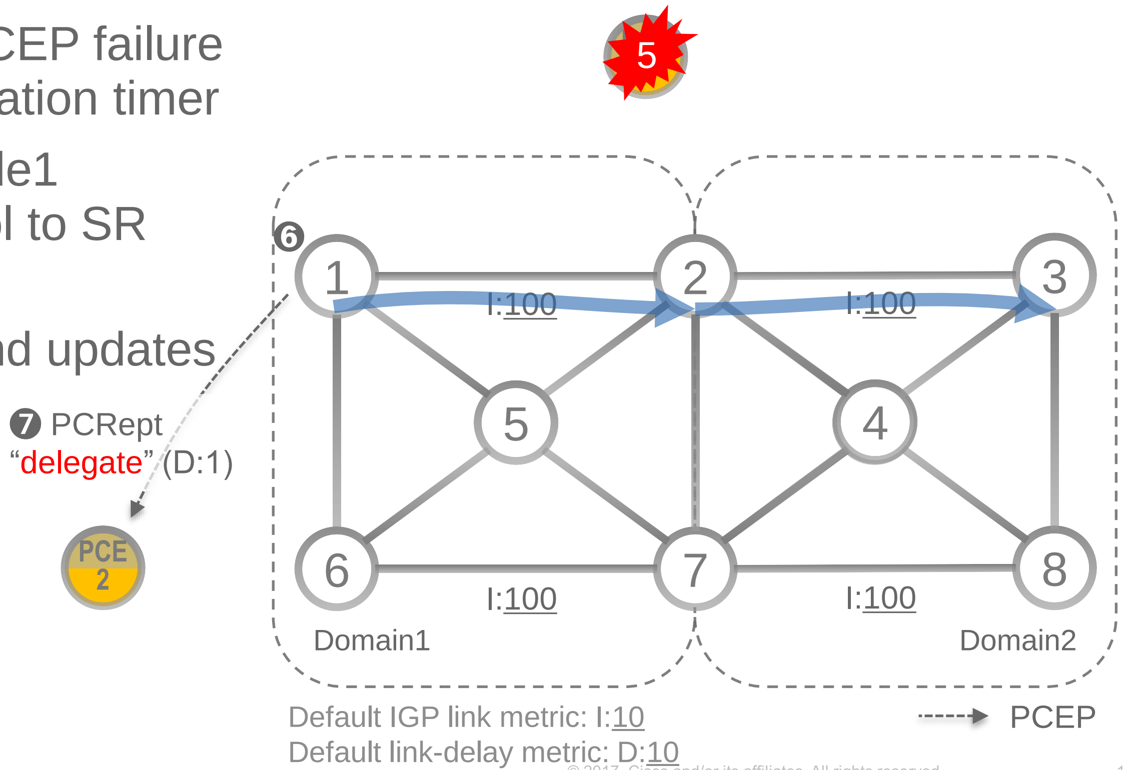
SR PCE HA – workflow

- ❶ Node1 requests SR PCE1 to compute path to Node3, ❷ SR PCE1 computes path and replies with SID list <30102, 30203> and ❸ Node1 activates SR Policy
- ❹ Node1 reports SR Policy to both SR PCE1 and SR PCE2 and delegates control of the SR Policy to SR PCE1 (“delegate” (D:1))



SR PCE HA – workflow

- ❺ SR PCE1 (primary) fails
- ❻ Node1 detects SR PCE1 PCEP failure (keepalive) and starts re-delegation timer
- ❼ when the timer expires, Node1 delegates the SR Policy control to SR PCE2
- SR PCE2 re-computes path and updates path if required

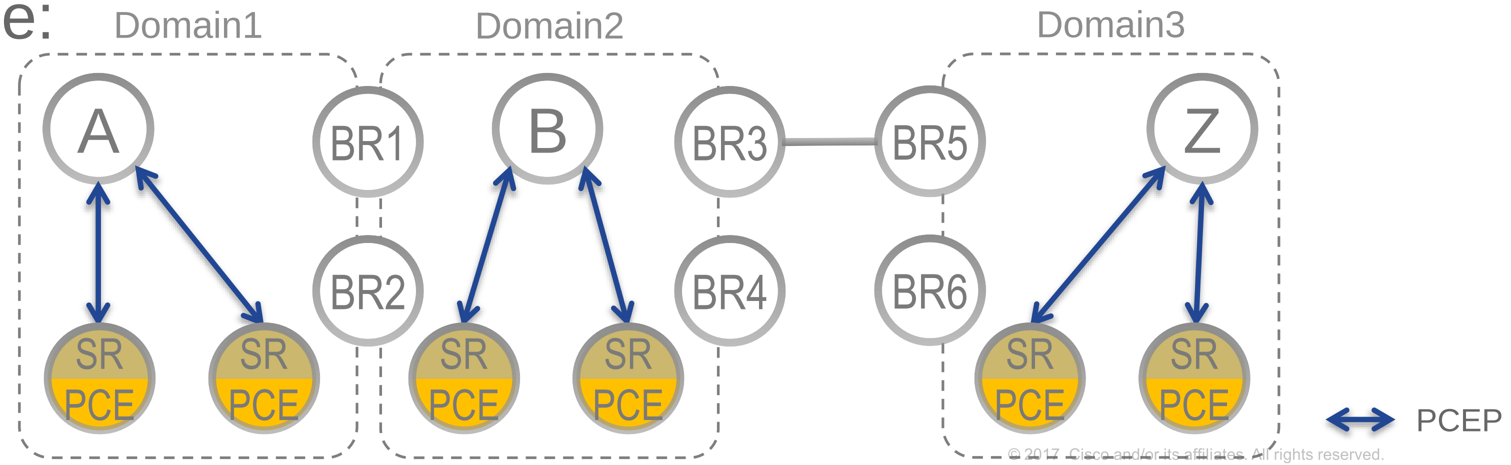


SR PCE – Fundamentally Distributed

- SR PCE not to be considered as a single all-overseeing device
- SR PCE deployment is closer to BGP RR deployment model
- Different service end-points can use different pairs of SR PCE s
- Choice of SR PCE can either be based on proximity or service

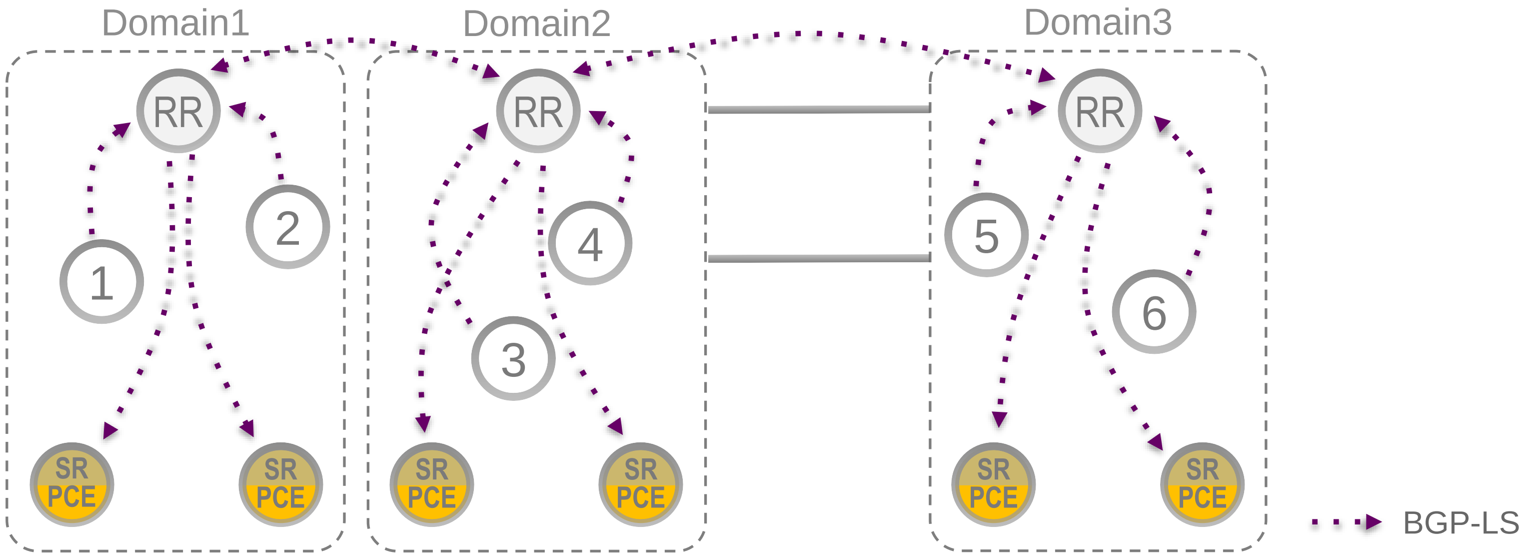
SR PCE – Fundamentally Distributed

- Add SR PCE nodes where needed; per geographic region, per service, ...
  - SR PCE needs to get the required topology information for its task
    - > E.g. to compute inter-domain paths SR PCE needs the topology of all domains
- Example:

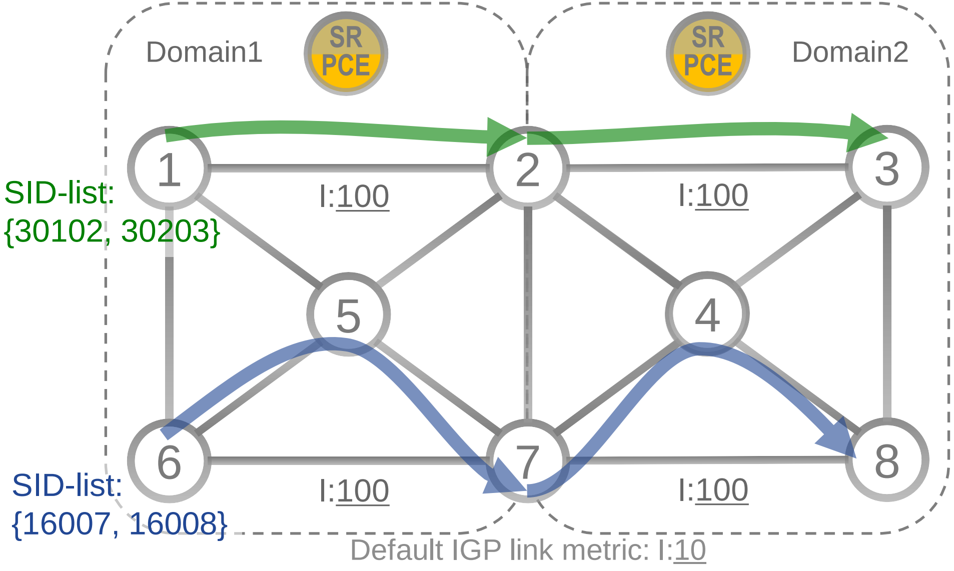


SR PCE – Fundamentally Distributed

- Using RRs to scale the BGP-LS topology distribution
- Any node can have a BGP-LS session to the RR



Use-case
Service Disjointness



Node1

```
segment-routing
 traffic-eng
  policy POLICY1
   color 20 end-point ipv4 1.1.1.3
    candidate-paths
     preference 100
      dynamic
       pcep
       metric type igp
      constraints
       association group 1 type node
```

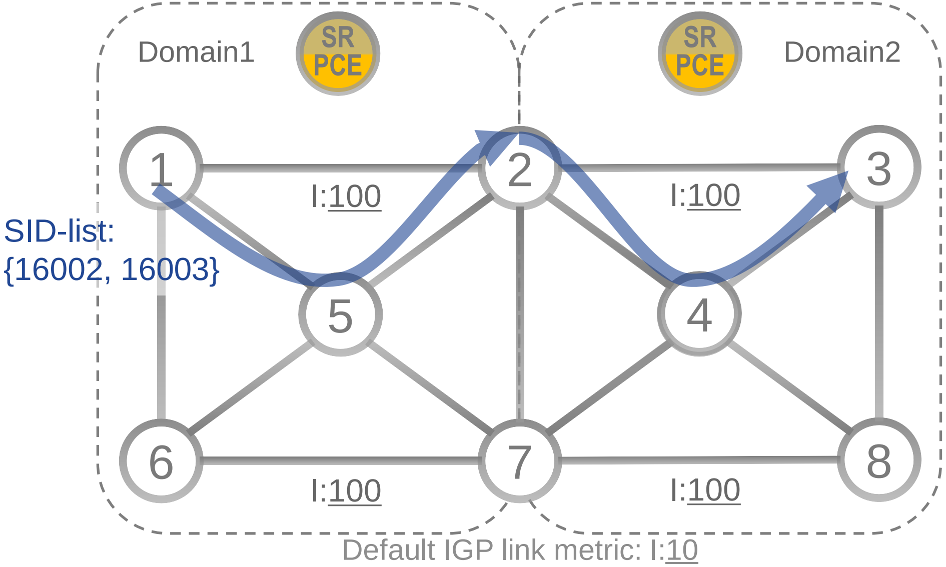
Node6

```
segment-routing
 traffic-eng
  policy POLICY2
   color 20 end-point ipv4 1.1.1.8
    candidate-paths
     preference 100
      dynamic
       pcep
       metric type igp
      constraints
       association group 1 type node
```

- Two dynamic paths between two different pairs of (head-end, end-point) must be disjoint from each other

Use-case

Inter-Domain Path – Best Effort



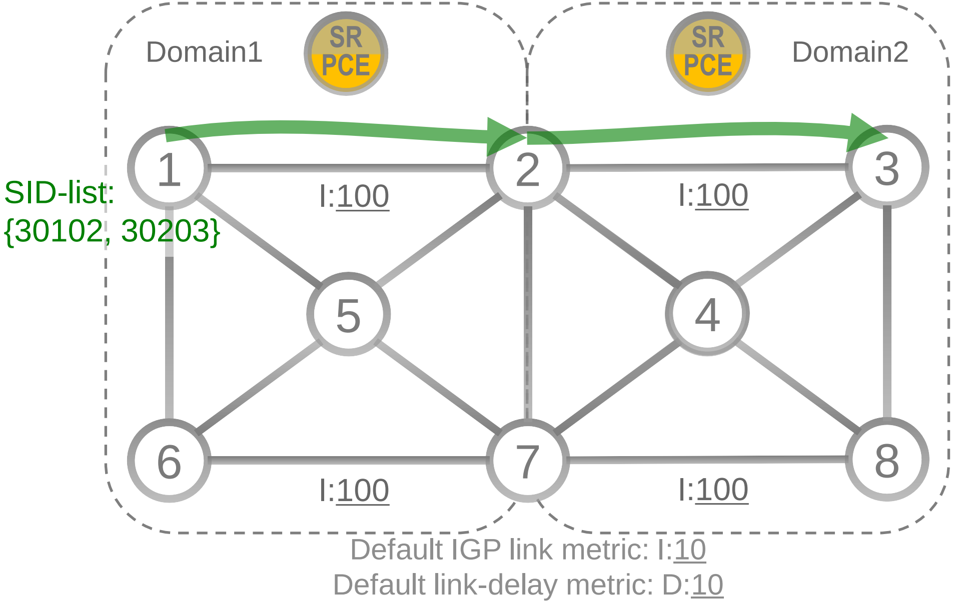
Node1

```
segment-routing
  traffic-eng
    policy POLICY1
      color 20 end-point ipv4 1.1.1.3
      candidate-paths
        preference 100
          dynamic
            pcep
            metric
              type igp
```

- There is no a-priori route distribution between domains

Use-case

Inter-Domain Path – Low-Delay



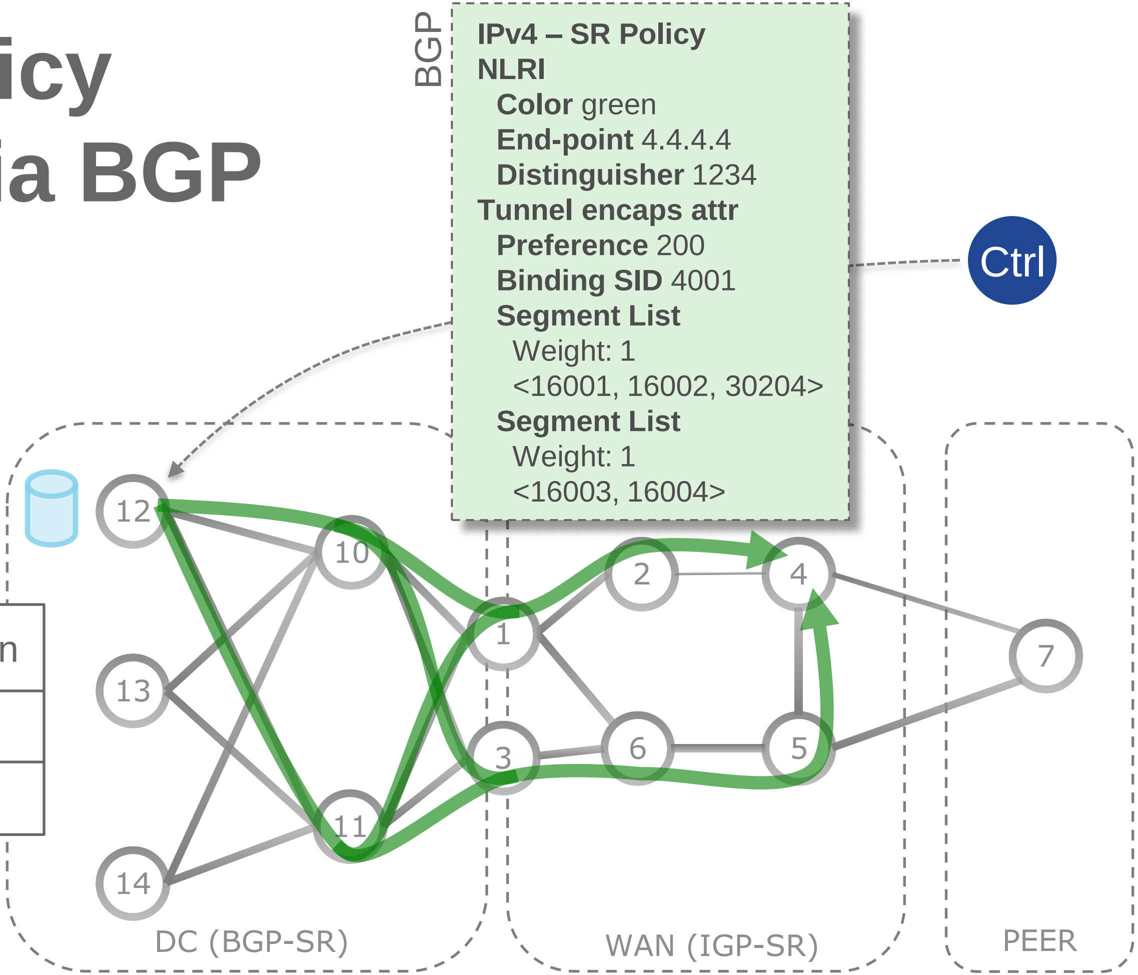
Node1

```
segment-routing
  traffic-eng
    policy POLICY1
      color 20 end-point ipv4 1.1.1.3
      candidate-paths
        preference 100
          dynamic
            pcep
            metric
              type latency
```

- No a-priori route distribution required between domains
- An end-to-end policy is requested

BGP-SRTE
Signaling SR Policy path via BGP

Signaling SR Policy candidate path via BGP



FIB on Node12:

In	Out	Fraction
4001	<16001, 16002, 30204>	50%
	<16003, 16004>	50%

Signaling SR Policy candidate path via BGP

- BGP signals a candidate path of an SR Policy
  - SR Policy is identified by the NLRI
  - If the SR Policy does not yet exist when the candidate path is signaled, then the SR Policy will be automatically instantiated

SAFI and NLRI

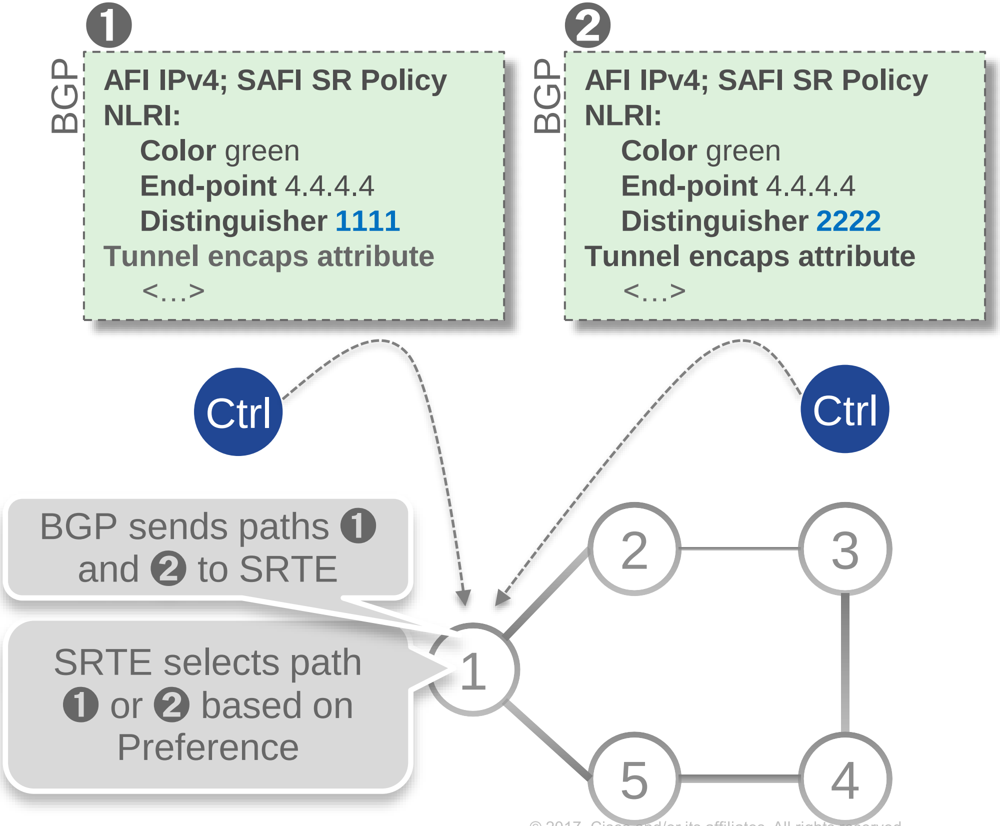
NLRI

```
+------------------------------------------+
|        Distinguisher (4 octets)          |
+------------------------------------------+
|        Policy Color (4 octets)           |
+------------------------------------------+
|        End-point (4 or 16 octets)        |
+------------------------------------------+
```

- A new SAFI is defined: SR Policy SAFI
  - suggested code-point value 73, to be assigned by IANA
- The NLRI identifies the SR Policy
  - Distinguisher: BGP-specific mechanism to allow to distribute multiple paths for the same SR Policy and avoid BGP-based path selection
    - Recommendation: path selection should be done by SRTE as part of the SR Policy behavior
  - Policy Color: identifies the color of the SR Policy
  - End-point: identifies the end-point of the SR Policy

Path selection in SRTE, not in BGP

- Recommendation:
  – Use Distinguishers to avoid BGP path selection
  – Path selection is better done by SRTE process



Path description

- The signaled candidate path for the SR Policy is encoded in a Tunnel Encapsulation Attribute
  - See draft-ietf-idr-tunnel-encaps; new Tunnel Type: “SR Policy”
- One single candidate path is advertised per NLRI

NLRI,
identifies SR Policy

Tunnel Encaps Attribute,
defines a candidate path for the identified SR Policy

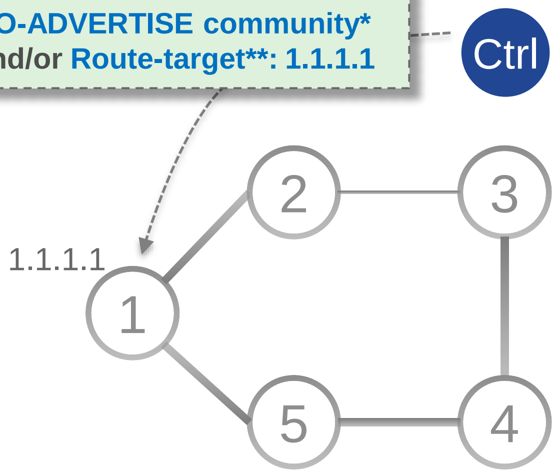
```
SR Policy SAFI NLRI:
  <Distinguisher, Policy-Color, End-point>
Attributes:
  Tunnel Encaps Attribute (23)
     Tunnel Type: SR Policy
       Preference TLV
       Binding SID TLV
       Segment List TLV
         Weight SubTLV
         Segment SubTLV
         Segment SubTLV
         ...
```

Direct session or via RR

Direct session

BGP

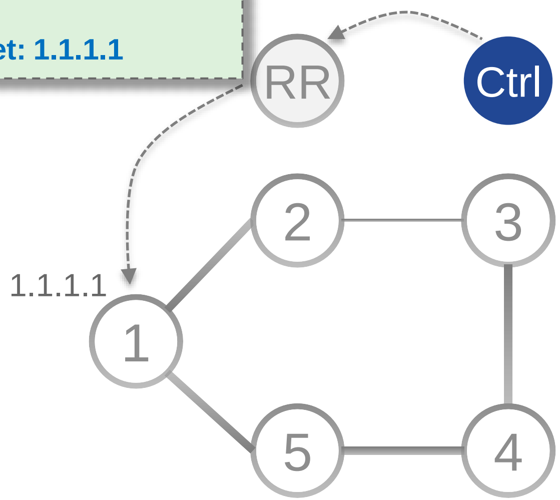
AFI IPv4; SAFI SR Policy
NLRI:
  Color green
  End-point 4.4.4.4
  Distinguisher 1111
Tunnel encaps attribute
  <…>
NO-ADVERTISE community*
and/or Route-target: 1.1.1.1**



Via RR

BGP

AFI IPv4; SAFI SR Policy
NLRI:
  Color green
  End-point 4.4.4.4
  Distinguisher 1111
Tunnel encaps attribute
  <…>
Route-target: 1.1.1.1



* NO-ADVERTISE community: indicates: “do not advertise to any BGP neighbor”
** Route-target extended community (cfr L3VPN)

BGP only a conveyor of information

- BGP does basic sanity checks on the Update message
- If multiple paths have been received for the same NLRI (Distinguisher, Color, End-point), run BGP bestpath
  - Unlikely, see previous recommendation
- Give the path to SR-TE process
  → path is one of the possibly many candidate paths of the SR Policy

Head-end BGP SRTE Configuration

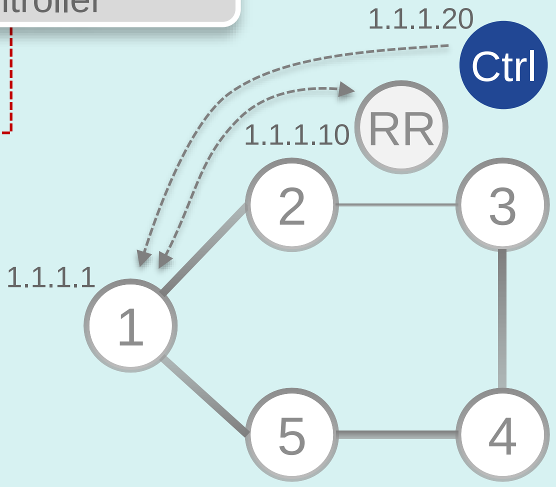
On Node1:

```
router bgp 1
 bgp router-id 1.1.1.1
 address-family ipv4 unicast
 !
 address-family vpnv4 unicast
 !
 address-family ipv4 sr-policy
 !
 neighbor 1.1.1.10
  remote-as 1
  update-source Loopback0
  address-family ipv4 unicast
  !
  address-family vpnv4 unicast
 !
```

To Service RR

```
neighbor 1.1.1.20
 remote-as 1
 update-source Loopback0
 address-family ipv4 sr-policy
```

To BGP SRTE Controller



- 1.1.1.10 is a service RR (IPv4 and VPNv4)
- 1.1.1.20 is a BGP SRTE controller

BGP TE SR Policy – example

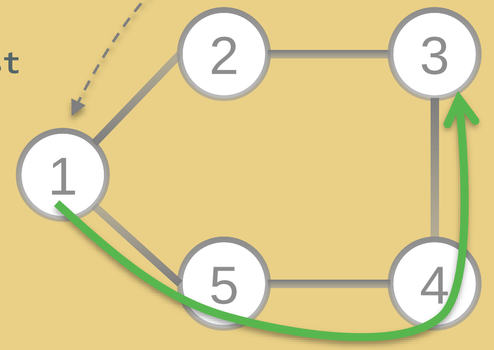
```
RP/0/0/CPU0:XRv-1#show bgp ipv4 sr-policy [2][10][1.1.1.3]/96
BGP routing table entry for [2][10][1.1.1.3]/96
Versions:
  Process           bRIB/RIB  SendTblVer
  Speaker                  4           4
Last Modified: Jun 13 21:18:10.371 for 00:05:50
Paths: (1 available, best #1)
  Not advertised to any peer
  Path #1: Received by speaker 0
  Not advertised to any peer
  Local
    1.1.1.12 (metric 30) from 1.1.1.12 (1.1.1.12)
      Origin IGP, localpref 100, valid, internal, best, group-best
      Received Path ID 0, Local Path ID 0, version 4
      Extended community: RT:1.1.1.1:0
      Tunnel encap attribute type: 15 (SR Policy)
       bsid 900000, preference 100, num of paths 1
       Path 1, weight 0x1
        Sids: {16004} {16003}
       SR TE Policy state is UP, Allocated bsid 900000
```

BGP

IPv4 – SR Policy
NLRI
- **Color** 10
- **End-point** 1.1.1.3
- **Distinguisher** 2

Tunnel encaps attr
- **Preference** 100
- **Binding SID** 900000
- **Segment List**
  - Weight: 1
  - <16004, 16003>

Ctrl

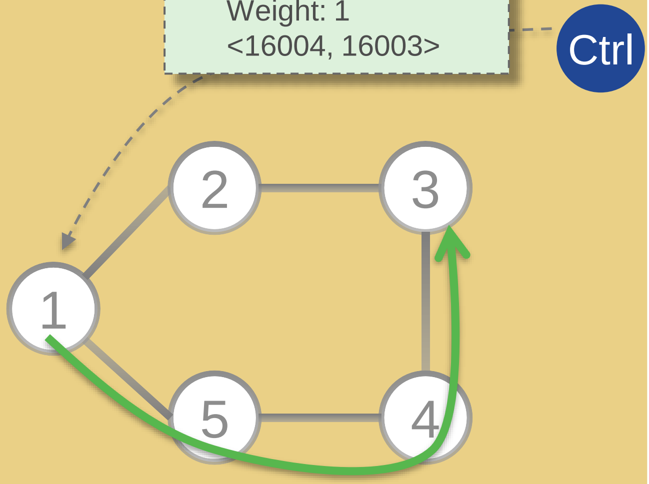


BGP TE SR Policy – example

```
RP/0/0/CPU0:XRv-1#show segment-routing traffic-eng policy
SR-TE policy database
---------------------
Name: bgp_AP_1 (Color: 10, End-point: 1.1.1.3)
    Status:
      Admin: up  Operational: up for 00:08:19 (since Jun 13 21:18:10.469)
    Candidate-paths:
      Preference 100:
        Explicit: segment-list Autopath_1_1* (active)
          Weight: 1
            16004
            16003
    Attributes:
      Binding SID: 900000 (configured)
      Forward Class: 0
      Distinguisher: 2
    Auto-policy info:
      Creator: BGP
      IPv6 caps enable: no
```

BGP

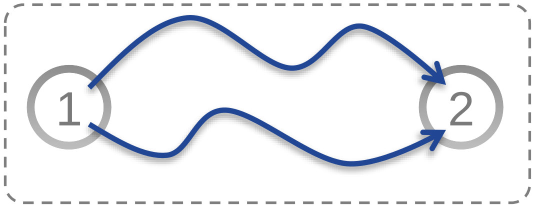
IPv4 – SR Policy
NLRI
Color 10
End-point 1.1.1.3
Distinguisher 2
Tunnel encaps attr
Preference 100
Binding SID 900000
Segment List
Weight: 1
<16004, 16003>

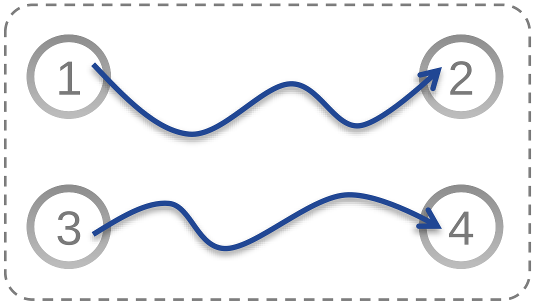


Path disjointness

Path disjointness

- Disjoint paths for a service may be required to guarantee service resiliency
  - Live-live or primary-backup
- Disjoint paths do not share any (or limited) network resources
- Path disjointness may be required for paths between the same pair of nodes, between different pairs of nodes, or a combination (only same head or only same end)



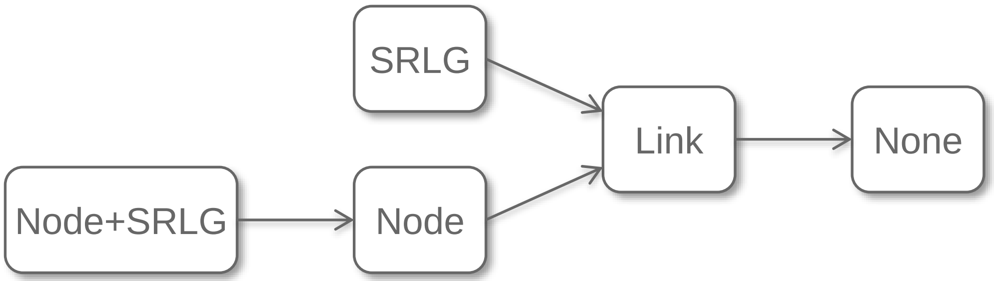


Path disjointness levels

- Different levels of disjointness may be offered:
  - Link disjointness: the paths transit different links (but may not be node or SRLG disjoint)
  - Node disjointness: the paths transit different nodes and different links (but may not be SRLG disjoint)
  - SRLG disjointness: the paths transit different links that do not share SRLG (but may not be node disjoint)
  - Node+SRLG disjointness: the paths transit different links that do not share SRLG and transit different nodes
- Common head-end nodes and end-point nodes are not taken into account for node-disjointness

Path disjointness levels – fallback

- If disjoint paths of a specified level are not available, then a lower level of disjointness will be tried:
  - If no node+SRLG-disjoint paths are available, then compute node-disjoint paths
  - If no SRLG- or node-disjoint paths are available, then compute link-disjoint paths
  - If no link-disjoint paths are available, then compute shortest paths without disjointness constraints
- Operator can disable fallback to another disjointness level



Association Groups

- The PCEP IETF draft-ietf-pce-association-group introduces a generic mechanism to create groups of LSPs
- This grouping mechanism can then be used to define associations between sets of LSPs or between a set of LSPs and a set of attributes (such as configuration parameters or behaviors)
- One application of this mechanism is grouping LSPs that must be mutually disjoint: disjointness association-group or disjoint-group
  - Specified in draft-litkowski-pce-association-diversity

PCEP Association Object

- draft-ietf-pce-association-group specifies the PCEP Association Object
  - This object indicates the association type and the association identifier
  - This object is included in PCReq and PCRept PCEP messages
- An association type is specified for ach disjointness level
  - Link, Node, SRLG, Node+SRLG
- The association identifier consists of a pair: (association-id, association source)

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|         Reserved              |            Flags            |R|
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|      Association type         |      Association ID           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|              IPv4 Association Source                          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
//                   Optional TLVs                             //
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

          Figure 1: The IPv4 ASSOCIATION Object format
```

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|         Reserved              |            Flags            |R|
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|      Association type         |      Association ID           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
|                   IPv6 Association Source                     |
|                                                               |
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
//                   Optional TLVs                             //
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

          Figure 2: The IPv6 ASSOCIATION Object format
```

Disjointness configuration

```
segment-routing
 traffic-eng
  policy POLICY1
   color 20 end-point ipv4 1.1.1.4
   candidate-paths
    preference 100
     dynamic
      pcep
      metric type igp
     constraints
      disjoint-path group-id 1 type node
```

Group-id 1

Node-disjoint

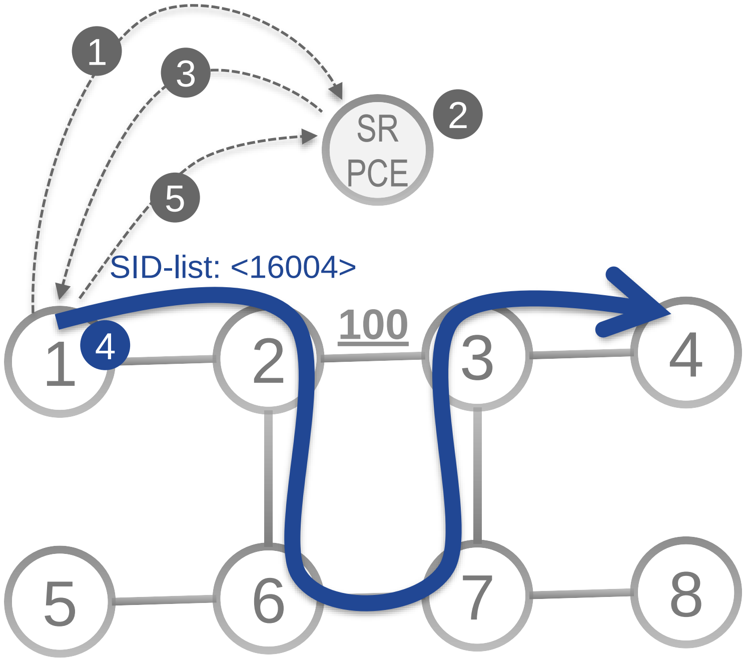
- Policies that must be disjoint must be configured with the same association group id and type

Disjoint paths – workflow

- This is the workflow when requesting disjoint paths:
  - First path of a disjoint-group is requested, it is computed as regular shortest path
  - Second path of a disjoint-group is requested, both paths are computed concurrently to provide the optimum solution and minimizing the combined cumulative metrics of both paths
    - PCE may need to update the first path after this computation
- Following a topology change, SR PCE re-computes both paths and updates them if required

Disjoint paths – workflow

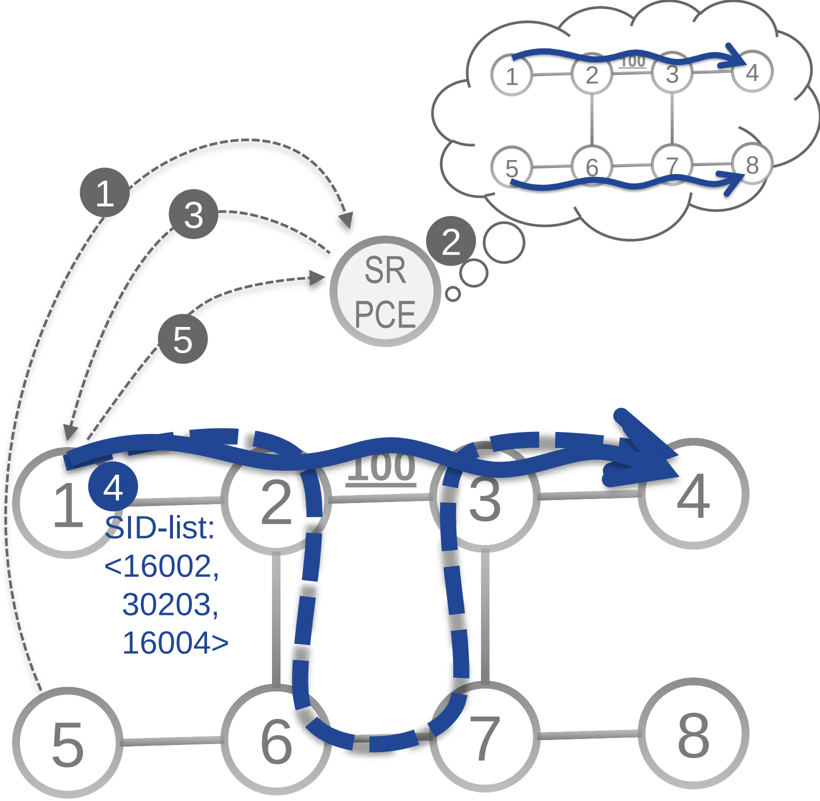
- Two node-disjoint paths are required Node1→Node4 and Node5→Node8
- ❶ Node1 first requests the path to Node4, ❷ SR PCE computes it as a regular shortest path and ❸ replies with SID-list <16004>
- ❹ Node1 installs the path and ❺ reports to SR PCE, delegating control to SR PCE



Default link metric: 10

Disjoint paths – workflow

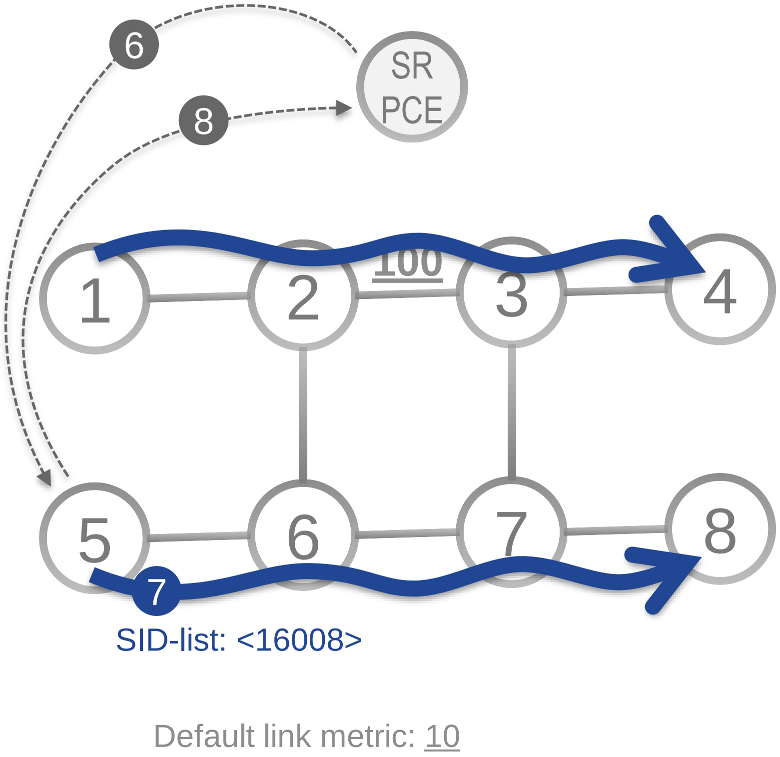
- ❶ Node5 requests path to Node8
- ❷ SR PCE concurrently computes the two paths and finds that the first (existing) path must be updated to accommodate disjointness with the second path
- ❸ SR PCE sends update to Node1 with SID-list <16002, 30203, 16004>
- ❹ Node1 installs the new path and ❺ reports to SR PCE



Default link metric: 10

Disjoint paths – workflow

- ❻ SR PCE sends reply to Node5 with SID-list <16008>
- ❼ Node5 installs path and ❽ sends report to SR PCE



Disjoint paths – workflow

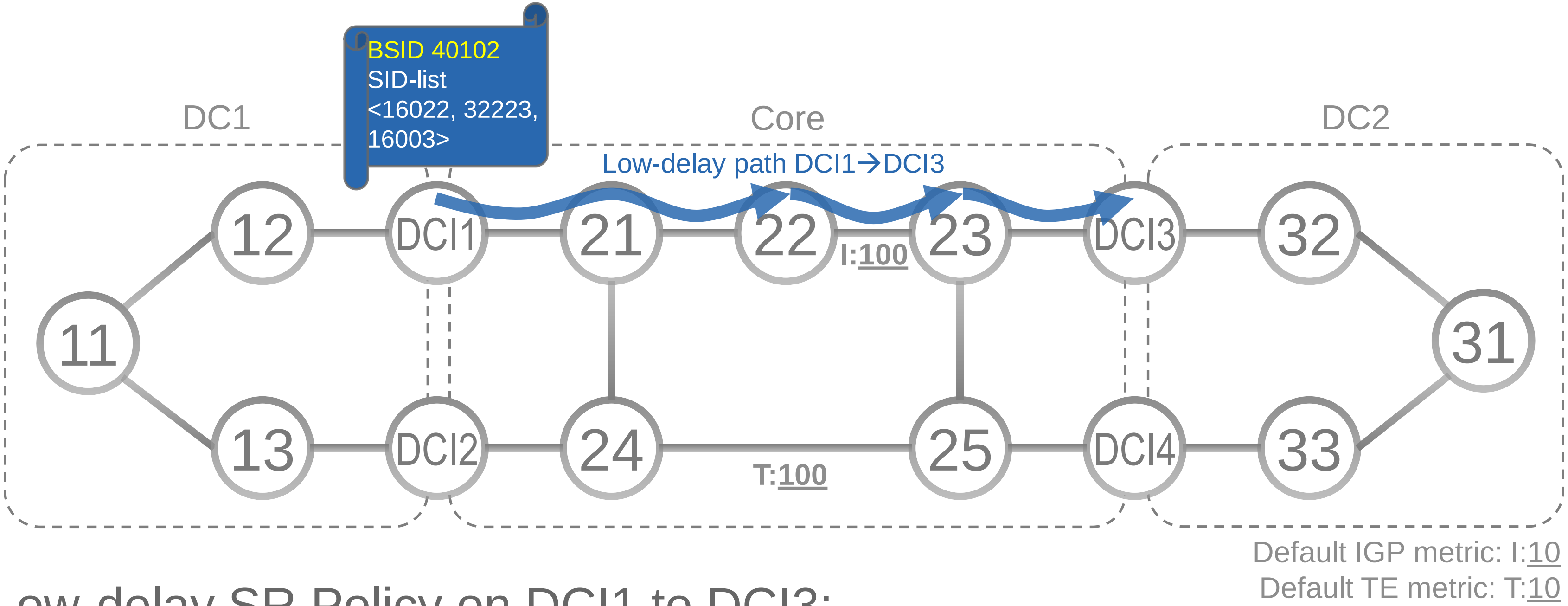
- Following a topology change, SR PCE is notified by IGP/BGP-LS
- SR PCE re-computes both paths and updates them if required

Binding-SID

Binding-SID is fundamental to SR

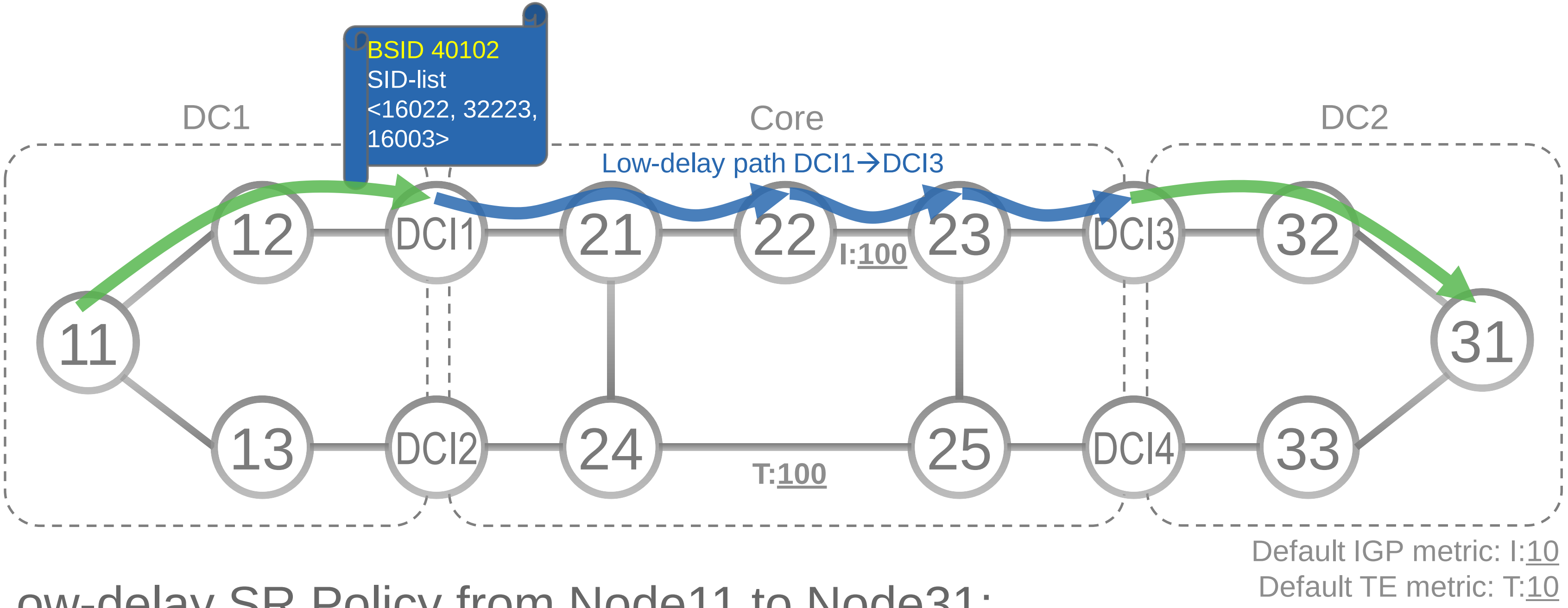
- The Binding-SID is fundamental to SR, it provides scaling, network opacity and service independence
  - Use of BSID decreases the number of segments imposed by the source
  - A BSID acts as a stable anchor point that isolates one domain from the churn of another domain
  - A BSID provides opacity and independence between domains

Binding-SID illustration



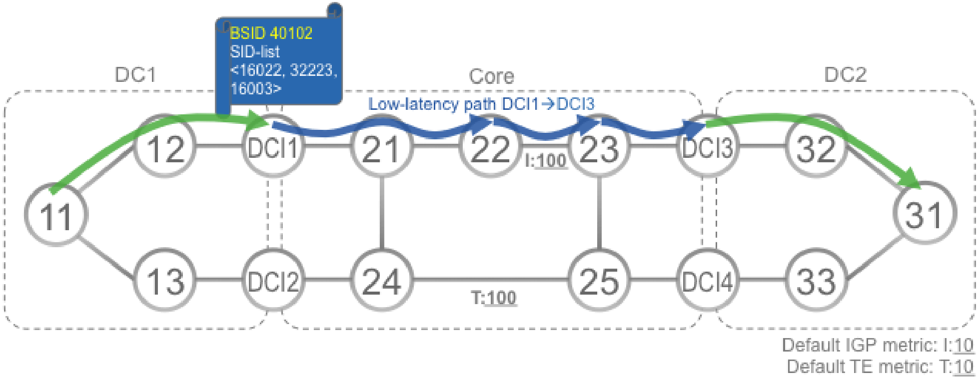
- Low-delay SR Policy on DCI1 to DCI3:
  - BSID: 40102
  - SID-list <16022, 32223, 16003>

Reduced imposition SID-list size



- Low-delay SR Policy from Node11 to Node31:
  - Without intermediate core SR Policy: <16001, 16022, 32223, 16003, 16031>
  - With intermediate core SR Policy: <16001, 40102, 16031>

Stable Anchor Point

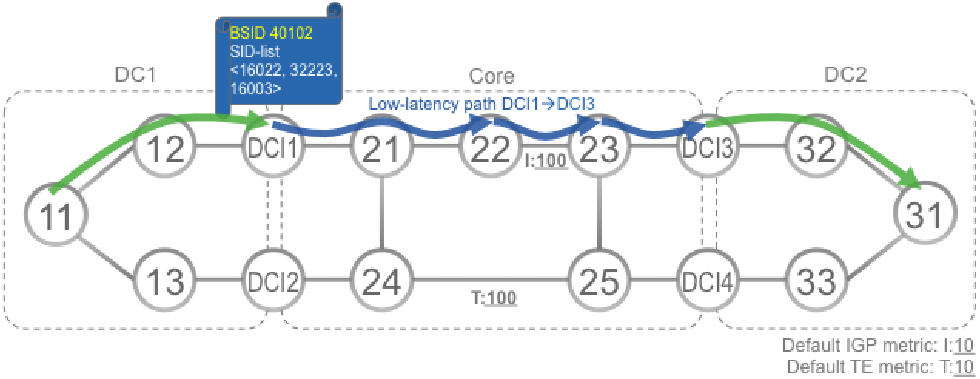


- When the Core domain's topology changes, the BSID of the intermediate SR Policy on DCI1 does not change

→ the SR Policy on Node11 does not change

→ Node11 is shielded from the churn in domain DC1

Opacity and Independence



- The administrative authority of the Core domain does not want to share information about its topology
  → BSID keeps network and service opaque
- Node11 does not know the details of how the Core domain provides the low-delay service

BSID allocation

- By default, BSID is dynamically allocated
- BSID can be explicitly specified
- BSID can be allocated for RSVP-TE tunnel

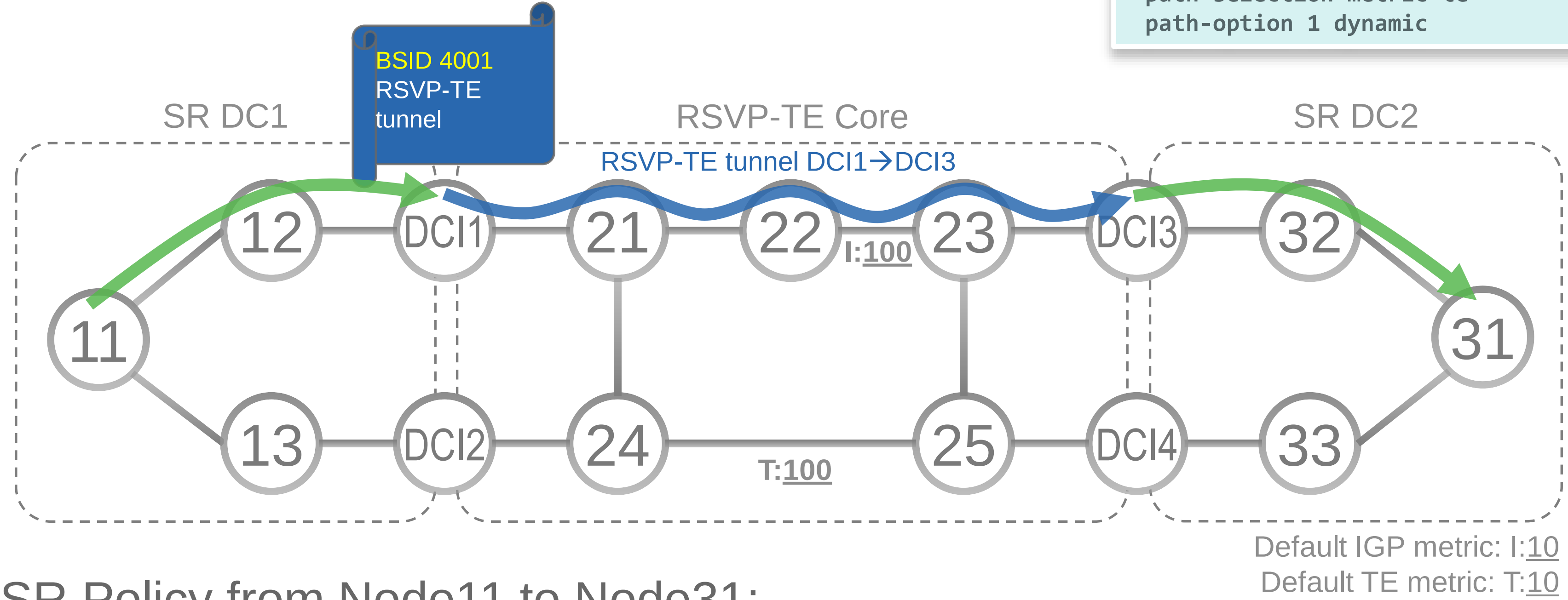
Explicit allocation – Example

```
segment-routing
 traffic-eng
  policy POLICY1
   color 20 end-point ipv4 1.1.1.4
   binding-sid mpls 1000
   candidate-paths
    preference 100
     dynamic
      metric
       type te
```

- Dynamic allocation is the default

SRTE RSVP-TE interworking

```
interface tunnel-te1
 ipv4 unnumbered Loopback0
 destination 1.1.1.3
 binding-sid mpls label 4001
 path-selection metric te
 path-option 1 dynamic
```



- SR Policy from Node11 to Node31:
  - With intermediate RSVP-TE tunnel: <16001, 4001, 16031>

Thank you.