


Virtual Service chaining

Shared with Istio Community

| | |
|---|---|
|  | |
| Owner: hzxuzhonghu, Neeraj Poddar Working Group: Networking | Status: WIP In Review Approved Obsolete Created: 11/29/2019 Approvers: Louis Ryan [], Lin Sun [], Shriram Rajagopalan [], John Howard [], Joshua Blatt [] |

TL;DR

Based on [RFE: Virtual Service Delegation](#). But is slightly different in API, if you have read that proposal, just jump to [Solution](#).

Intro

This document proposes enhancements to the Istio [VirtualService](#) spec which will allow mesh routing configuration to be specified in multiple composable VirtualService resources which can be chained together to create advanced traffic routing functionality in a user friendly way. Composable VirtualService resources allow organizations with different teams to maintain ownership of routing resources for the services they create while allowing operators to manage [Gateway](#) and Ingress level routing to get traffic into the mesh and forward it to the appropriate backend end service routing resources.

Background

The current VirtualService spec requires all the route rules for a host to be specified in a single resource ordered by priority. The [destination hosts](#) for the routes in the virtual services can only be one of the service names discovered from the platform's service registry or hosts declared in the [ServiceEntry](#) resources. The combination of these restrictions lead to the following operational challenges:

- A VirtualService resource can quickly grow in size if an organization has large number of **routes based on paths for a single hostname**. For example if the host "mycompany.com" has path based routing where prefix "/svcA" points to the service "svcA.team-svcA.svc.cluster.local" and "/svcB" points to the service "svcB.team-svcB.svc.cluster.local", currently all the sub-routes for these services

will have to be specified in a single VirtualService as they share the hostname “mycompany.com”. As the size of the VirtualService resource increases it gets harder to manage and reason about routes affecting a particular host and debug configuration related issues.

- VirtualService resources include routing config for services/functionalities owned by different teams. The example mentioned above highlights this issue as route rules for services “svcA” and “svcB” need to be specified in a single VirtualService resource even though these services can be in different namespaces owned by separate application teams. Additionally, the public hostnames (in this example “mycompany.com”) which are owned by operators (SecOps/NetOps) specified in a VirtualService also contain the route rules for backend services (in this example “svcA and svcB”) which are owned by application teams.
- It is difficult to enforce Kubernetes RBAC policies for VirtualService resources. This is a side effect of the point above as you need ownership semantics for resources to enforce RBAC.
- It is difficult for multi services to do blue-green deployment in parallel, especially when they belong to different teams. There is a race, and the later team’s config can revert the previous one. This requires them to do upgrade in order strictly. But it is unacceptable when these services are all part of a big system and it need upgrade ASAP to reduce the influence at most.
- In case a new service deployed, it is hard to add the route for it. The matching conditions is strictly ordered.

This proposal aims to solve the challenges mentioned above while reducing the operating complexity of managing a Istio cluster.

Solution

This proposal suggests a solution which is slightly different with [RFE: Virtual Service Delegation](#).

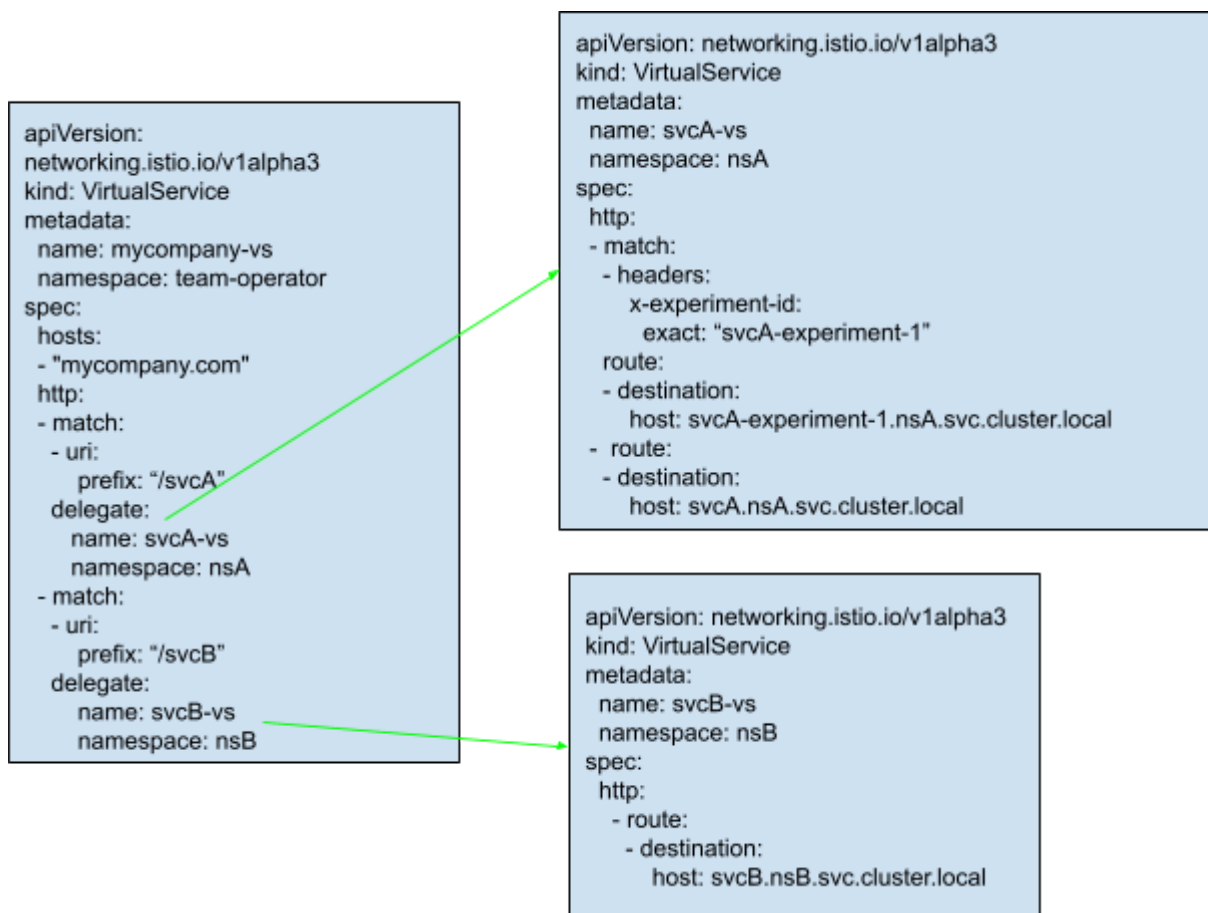
~~It only allows only HTTPRoute be chained, allows HTTP route to be delegated or forward to other VirtualService resources, and I think it makes less sense for TCP or TLS protocol route chaining.~~

The [HTTPRoute](#) specification will be augmented to allow chaining **VirtualService** resources in addition to the **HTTPRouteDestination** defined in it.

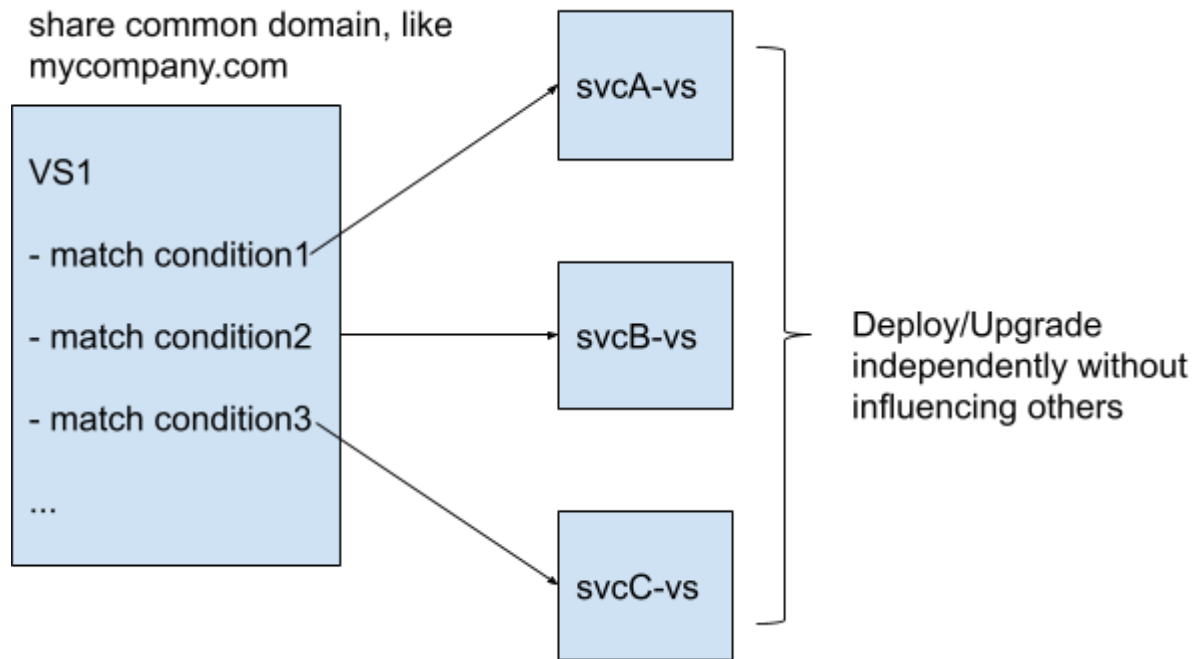
In the new model the configuration flows like this:

- Multiple users (or team's) write the VirtualService(s) for the services they own.
- VirtualService from one team ("mycompany-vs" from the above example) can delegate or forward to route rules in other VirtualService resources.

This is the configuration in the new VirtualService model:



The overall delegate principle is as below:



- Multiple micro services share one common domain
- Ingress gateway routing the traffic/request based on the matching result
- Decouple the routing rules of each service with each other
- Allow upgrading/deployment independently

The controlplane has the responsibility to validate whether there is a conflict. As this procedure can be very complicated since there are many match attributes for HTTP, we can reduce this complex and the possibility of conflicts in two dimensions:

1. Do not allow the chaining routes exceed 2 tiers (rarely users need so many tiers of routing rules)
2. The root HTTPRoute can at most have one **HTTPMatchRequest**, otherwise the matching rules will be multiplexed by the number of the root and the delegator's **HTTPMatchRequest**.

Istio controlplane(pilot) also has the responsibility to merge the chaining VirtualServices into one. The merged VirtualService should look like the one that is written manually. And if there is a conflict, the conflicted route rules will not take any effect.

Using this model allows application teams to write route rules for their services they own independent of other services even if they share the same hostname. Additionally, the VirtualService with public hostname is managed by operators which can delegate to the application team VirtualService resources as needed.

API Changes

HTTP Protocol

A new field “delegate” is added to VirtualService HTTPRoute section. Adding this new field maintains backward API compatibility as only one of “route” “redirect” or “delegate” can be specified for a route destination.

```
type HTTPRoute struct {
    // The name assigned to the route for debugging purposes. The
    // route's name will be concatenated with the match's name and
    will
    // be logged in the access logs for requests matching this
    // route/match.
    Name string `protobuf:"bytes,17,opt,name=name,proto3"
json:"name,omitempty"`
    // Match conditions to be satisfied for the rule to be
    // activated. All conditions inside a single match block have
    AND
    // semantics, while the list of match blocks have OR semantics.
    The rule
    // is matched if any one of the match blocks succeed.
    Match []*HTTPMatchRequest
`protobuf:"bytes,1,rep,name=match,proto3" json:"match,omitempty"`
    // A http rule can either redirect or forward (default)
    traffic. The
    // forwarding target can be one of several versions of a
    service (see
    // glossary in beginning of document). Weights associated with
    the
    // service version determine the proportion of traffic it
    receives.
    Route []*HTTPRouteDestination
`protobuf:"bytes,2,rep,name=route,proto3" json:"route,omitempty"`
    // A http rule can either redirect or forward (default)
    traffic. If
    // traffic passthrough option is specified in the rule,
    // route/redirect will be ignored. The redirect primitive can
    be used to
    // send a HTTP 301 redirect to a different URI or Authority.
    Redirect *HTTPRedirect
`protobuf:"bytes,3,opt,name=redirect,proto3"
json:"redirect,omitempty"`
    // Rewrite HTTP URIs and Authority headers. Rewrite cannot be
    used with
    // Redirect primitive. Rewrite will be performed before
    forwarding.
```

```

Rewrite *HTTPRewrite
`protobuf:"bytes,4,opt,name=rewrite,proto3"
json:"rewrite,omitempty"`
    // Optional: Delegate is used to specify the name of the
    delegated virtualservice.
    // It can be set only when route and redirect are empty. The
    route rules of the cascading virtualservice will be merged with
    // that in the root virtualservice. The merge semantic is
    `AND`.
    // Note:
    //     1. The chaining virtualservice can only specify HTTP
    type route, otherwise these two virtualservices are invalid.
    //     2. The chaining virtualservice's hosts feild must be
    empty, which is used to judge if it is a delegated VS.
    Delegate *Delegate
    ...
}

// The delegated VirtualService
type Delegate struct {
    // The name of the VS
    Name string
    // The namespace of the VS
    Namespace string
}

```

Another change allowed in the current API is making the VirtualService “spec.hosts” field optional, and for the delegate VirtualService the hosts must be empty. So empty hosts indicates it is a delegator and will not take effect standing alone. Otherwise it may introduce unknown behavior when applied alone.

TLS Protocol

Add a delegate to *TLSSRoute*

```

type TLSSRoute struct {
    // Match conditions to be satisfied for the rule to be
    // activated. All conditions inside a single match block have
    AND
    // semantics, while the list of match blocks have OR
    semantics. The rule
    // is matched if any one of the match blocks succeed.
    Match []*TLSSMatchAttributes
    `protobuf:"bytes,1,rep,name=match,proto3" json:"match,omitempty"`
}

```

```

    // The destination to which the connection should be
    forwarded to.
    Route                                []*RouteDestination
`protobuf:"bytes,2,rep,name=route,proto3" json:"route,omitempty"`

    // Only one of `Route` and `Delegate` can be specified
    Delegate *Delegate
}

```

TCP Protocol

```

type TCPRoute struct {
    // Match conditions to be satisfied for the rule to be
    // activated. All conditions inside a single match block have
    AND
    // semantics, while the list of match blocks have OR
    semantics. The rule
    // is matched if any one of the match blocks succeed.
    Match []*L4MatchAttributes
`protobuf:"bytes,1,rep,name=match,proto3" json:"match,omitempty"`
    // The destination to which the connection should be
    forwarded to.
    Route                                []*RouteDestination
`protobuf:"bytes,2,rep,name=route,proto3" json:"route,omitempty"`

    // Only one of `Route` and `Delegate` can be specified
    Delegate *Delegate
}

```

Pilot should walk down the chain of VirtualServices and merge the route rules as described in the next section.

Merging VirtualServices

This is the most hard part, it has to detect conflicts between root and delegate VS, and merge them if no conflicts. The below show the basic model and how we can do that:

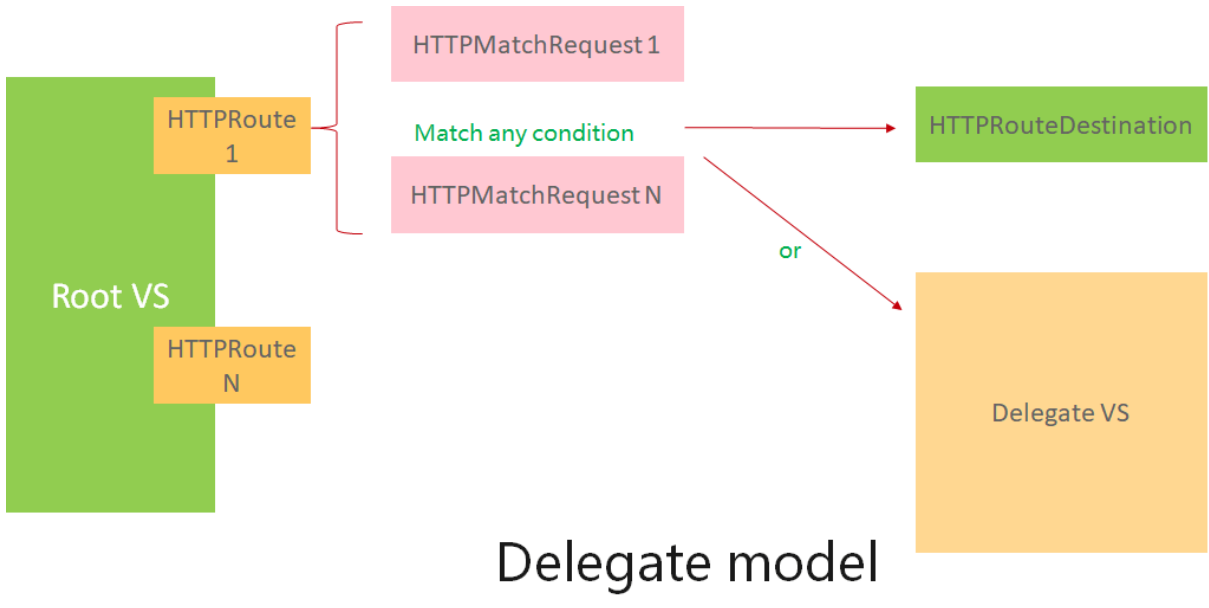


Fig 1. delegate model

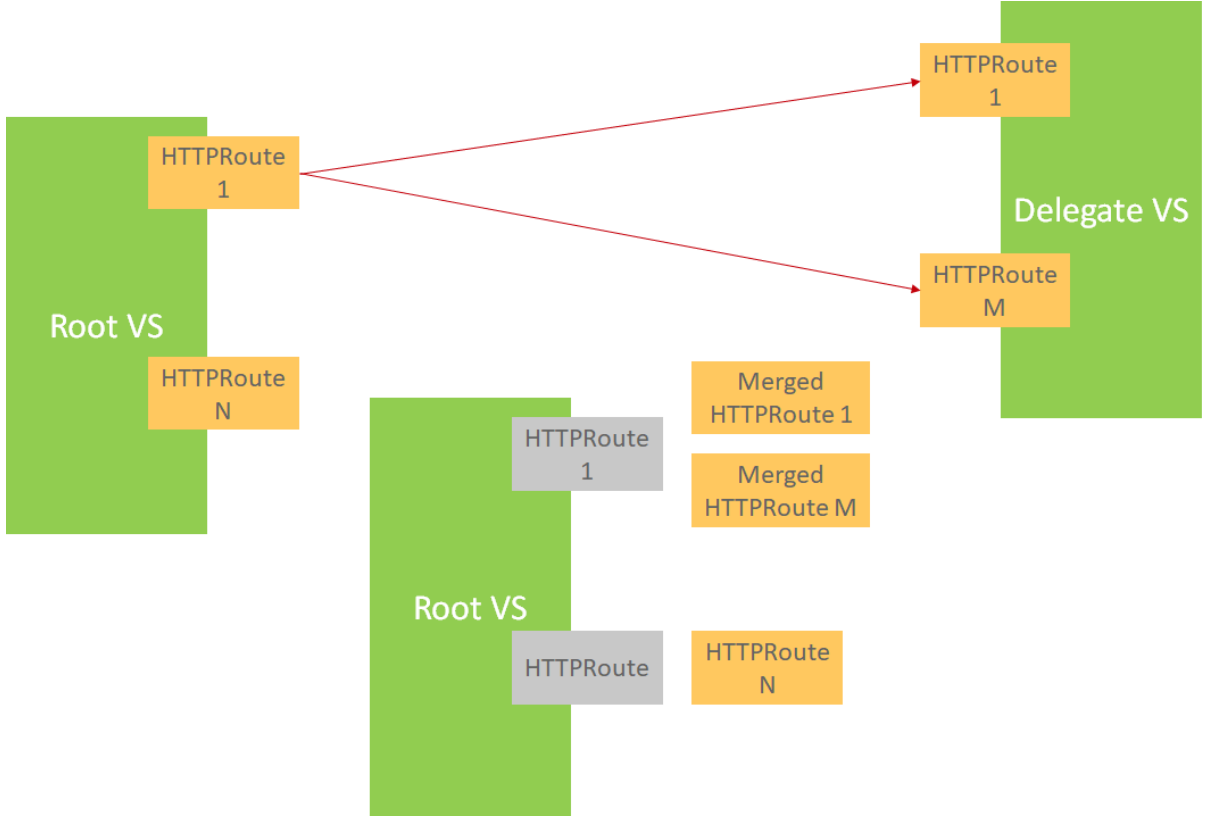


Fig 2. Expected merged VS

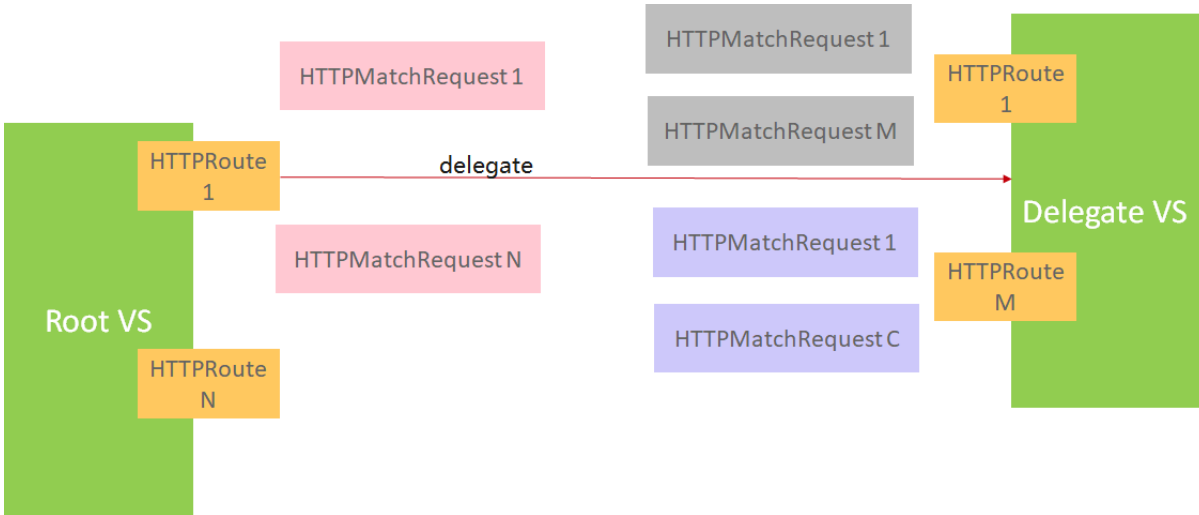
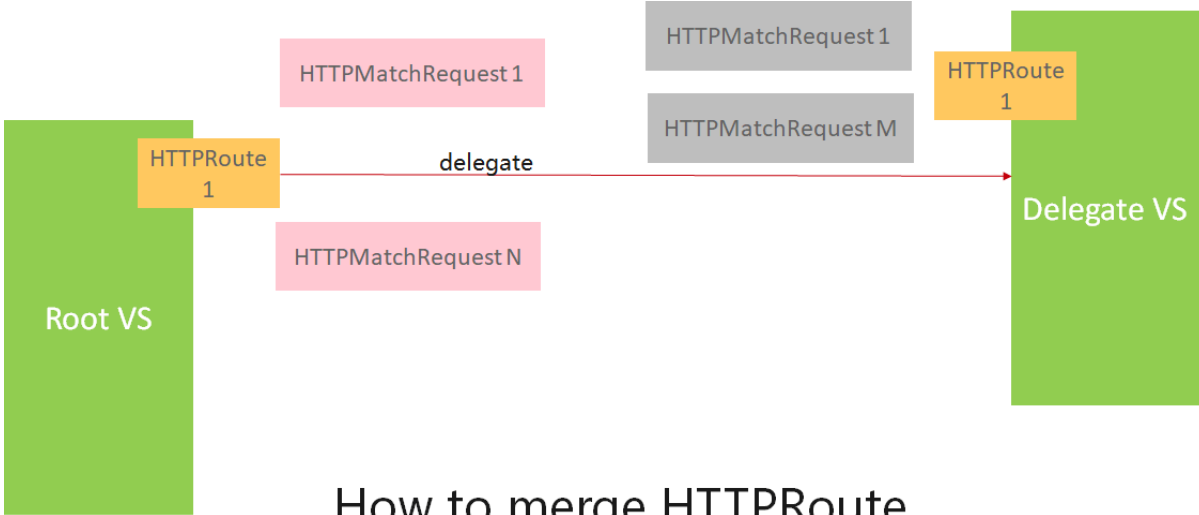
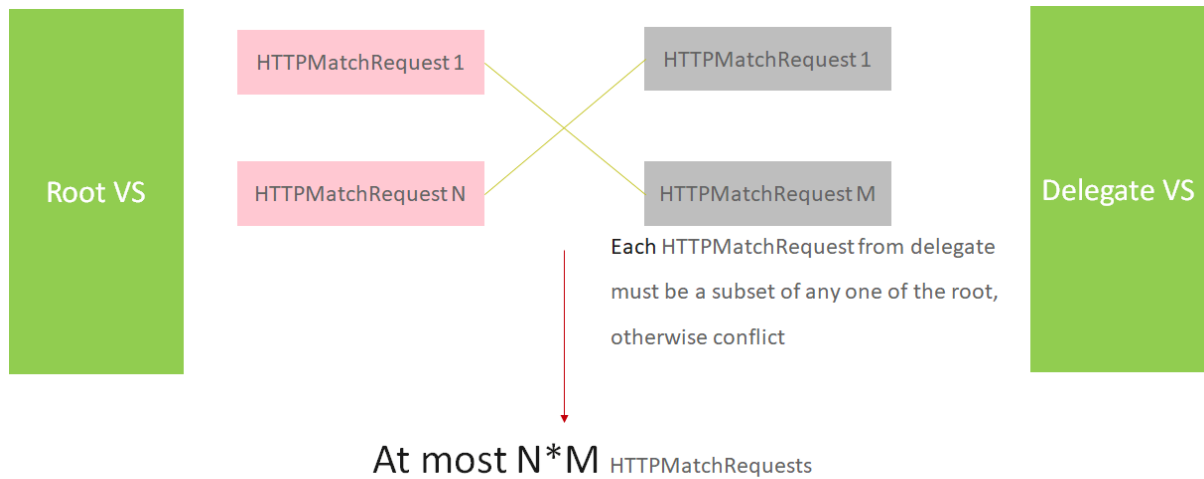


Fig 3. N HTTPMatchRequests per HTTPRoute



How to merge HTTPRoute

Fig 4. HTTPRoute merge



HTTPMatchRequest Merge

Fig 5. HTTPMatchRequestMerge

The route rules from the root VirtualService are merged by the Pilot with the child (delegated) VirtualService, and this is the most complex part. And once there is a conflict, the route rules will not take effect. **So a delegate's match conditions must match a strict subset of the root's match conditions.** In order to make pilot be able to detect conflict and merge route rules, we have to set a lot of restrictions for different fields.

HTTP Protocol

HTTPRoute merging semantics

This table suggests how to detect conflicts and merge route rules based on the restrictions:

| HTTPRoute fields | Restrict | Merge logic | Conflict condition |
|---|--|---------------------------------|--------------------|
| Match []* HTTPMatchRequest | The root VirtualService should at most have one Match, | See below table | |

| | | | |
|---|--|--|---|
| Route | Can not be set with Delegate or Redirect | Take the delegator's | If more than one of these fields set, the route is invalid. |
| Redirect | Can not be set with Route or Delegate | Take the delegator's | |
| Delegate | Can not be set with Route or Redirect | NA | |
| - Rewrite - Timeout - Retries - Fault - Mirror - MirrorPercent - CorsPolicy | NA | The merging semantics should be: use delegator's value if set, otherwise use that of the root. | NA |
| Headers | | See below table | |

[HTTPMatchRequest](#) merging semantics:

The merging requires no conflicts, the delegate's matching conditions must be a subset of the root.

| HTTPMatchRequest fields | Restrict | Merge logic | Conflict condition |
|---|----------|---|--------------------|
| Name | | The delegator's name will override the root's if set. | NA |

| | | | |
|--|---|--|--|
| Uri | No regex match should be specified, for it is difficult to judge whether there is a conflict. | The delegator's Uri will override the root's if set. | delegate Uri is not a subset of the root's. |
| -Scheme -Method -Authority | No regex can be set | Same as Uri | delegate is not a subset of the root's. |
| - Headers - WithoutHeaders - QueryParams | No regex can be set | Union | If the delegate's is not a subset of the root. For example, if a header name specified in both root and delegate, but the delegate's is not a subset of root. e.g. {keyA: {exact: foo} } conflicts with {keyA: {exact: bar} } |
| SourceLabels | | union | Not same value for a single key |
| Gateways | | Override if set | If the delegate is not a subset of root |
| Port | | Set to the non 0 value | Not same value if both are set |
| IgnoreUriCase | | | The values are not equal |

HTTPRoute.Headers merging semantics

| <u>HTTPRoute.Headers</u> fields | Restrict | Merge logic | Conflict condition |
|---------------------------------|----------|---|--------------------|
| - Request - Response | | | |
| Set | | <p>logic Union, merge the map if not intersected; otherwise the delegator's value will override the root's for the same key.</p> <p>set from root VS { a: va, b:vb} merge with {b:vbb, c:vc} from delegate, the output is {a: va,b:vbb, c:vc}</p> | No conflicts |
| Add | | <p>Union: for the intersected key, the value will be joined with a comma, which is the envoy behavior.</p> | No conflicts |
| Remove | | Union slices | No conflicts |

TLS Protocol

TLSMatchAttributes merging semantics

| TLSMatchAttributes | Merge logic | Conflict condition |
|------------------------------------|---------------------------------------|--|
| - SniHosts - DestinationSubnets | The delegated VS overrides the root's | The delegated VS's SniHosts are not subset of the root's |

| | | |
|--------------|--|---------------------------|
| Port | The port in delegate will override the root's if set | If set but not equal |
| SourceSubnet | NA | NA deprecated field |
| SourceLabels | The delegate's override the root's | Not a subset of the root |
| Gateways | Set the non empty value | If both set, but not same |

TCP Protocol

L4MatchAttributes merging semantics

| TLSSMatchAttributes | Merge logic | Conflict condition |
|---------------------|--|--|
| DestinationSubnets | The delegated VS overrides the root's | The delegated VS's SniHosts are not subset of the root's |
| Port | The port in delegate will override the root's if set | If set but not equal |
| SourceSubnet | NA | NA, deprecated field |

| | | |
|--------------|------------------------------------|---------------------------|
| SourceLabels | The delegate's override the root's | Not a subset of the root |
| Gateways | Set the non empty value | If both set, but not same |

Validation

1. Delegated VirtualService **Hosts** must be empty. So in validation, should allow empty Hosts.
2. Regex match is forbidden for fields like Uri, Scheme, Authority, QueryParams, etc.
3. Delegated VirtualService is allowed to contain only **HTTP Route**. **TLS and TCP** not supported, we don't allow route to different protocol at the first draft.
4. Delegated VirtualService can not be delegated, this is mainly for simplicity.
5. VirtualService HTTP Route validate should only allow setting one of delegate, route and redirect.

Alternative Approaches

Merging VR

Merging Istio's [HTTPMatchRequest](#). This has been widely discussed, and the matching conditions are too complex. So it seems not feasible.

Gateways/VirtualService per team

This can not tackle the case when multi micro services from different teams share a common domain.

Put delegate under *Destination*

Basically, *delegation* should be in the same *hierarchy* as destination. Currently, traffic is only routed to a specific destination when matched by a lot of fine grained conditions. Making delegate under destination is confusing and also the merging will have to handle a lot of ops like header mutation other than matching conditions merge.

SHRIRAM: Separate Include Block in VirtualService

Virtual service can have a separate include section that specifies names of other virtual services and conditions under which these virtual services should be included.

Eg

```
host: foo.com
IncludeVirtualServices:
- resource: ns1/vs1
  condition:
  - httpMatch:
    uriMatchHasPrefix: /barbar
    hasHeaderMatchesFor: x-envoy-blah
  - tlsMatch:
    usesSni: foo.example.com
```

This will cause pilot to only include the http rules that have /barbar as a prefix or exact match in the vs1 virtual service in the ns1 namespace.