# Tunneling egress traffic - improvements

Shared with Istio Community

| | |
|---|---|
| **Owner**: Jacek Ewertowski (jewertow@redhat.com)<br>**Working Group:** Networking | **Status**: WIP \| **In Review** \| Approved \| Obsolete<br>**Created**: 2022-11-04<br>**Approvers**: |

## Objective

This document proposes improvements to recently implemented tunneling destination rule.
The main goals are:

1. Support tunneling traffic to HTTP and GRPC services (ideally if possible for all protocols).
2. Simplify semantics of tunnel configuration to remove the need for redirecting traffic with a virtual service.
3. Collect metrics for a target service instead of for a proxy. It would be ideal to collect metrics for the target service as well as for the proxy, but currently Envoy exposes just a few HTTP metrics for the proxy, so it may be difficult to realize.
4. Eliminate limitations of applying other destination rules.
5. Make it possible to automatically deduce SNI and use it as a target host in the CONNECT requests in all TLS modes.

## Background

The limitations listed above come from the decision to start with the simplest implementation which is as close to Envoy API as possible.

1. Traffic cannot be tunneled when a service specifies its protocol as HTTP, GRPC or any other than TCP or TLS (in fact they could be tunneled when routed through a gateway, but with some other problems, so it's not recommended).
   The reason for this limitation is that only TcpProxy can tunnel traffic, so in case of HTTP, where the underlying listener is HttpConnectionManager, it's not possible without redirecting traffic to an intermediary TcpProxy.
2. Tunneling traffic mixes responsibilities of VirtualService (routing) and DestinationRule (post-routing), because to configure **`TcpProxy.tunneling_config`** we must change cluster (routing) and configure a header **`Host: <hostname>:<port>`** for CONNECT

request (post-routing). This is why my initial idea was to use VirtualService with DestinationRule to configure tunneling.

3. As explained in the previous item, applying tunnel settings changes clusters in the underlying listener, so metrics are collected for a proxy, instead of for a target service. This characteristic of current implementation reduces the value of provided metrics, so they should be collected for target service, and in an ideal case, for the target service as well as for the proxy (but I'm not sure if this is possible for now).
4. Because of a semantic that requires redirecting traffic with VirtualService to the proxy and then configuring that proxy with DestinationRule, it's not possible to originate TLS to the target service when tunneling is performed by a sidecar or TLS-terminated gateway. The reason for this limitation is that the DestinationRule specifying tunnel settings must be defined for the proxy hostname instead of for the target host as it might be expected. The only way to enable TLS origination to the target service is to route traffic via a gateway in TLS passthrough mode.
   To avoid confusion, I would like to note that TLS origination to the proxy works perfectly without any limitation.
5. Tunneling TCP over HTTP uses HTTP CONNECT under the hood. A request must include hostname and port to which the connection is tunneled. These values must be explicitly specified in **`tunnel.targetHost`** and **`tunnel.targetPort`** for each target service, but the hostname could be deduced from TLS connections automatically by setting **`%REQUESTED_SERVER_NAME%`** in **`TcpProxy.tunneling_config.hostname`**, but currently it could only work in non-terminated TLS listeners, i.e. when tunneling through sidecar or TLS passthrough gateway.

Let's say we want to tunnel a request to www.google.com by a sidecar proxy via an external proxy. Then the following Istio configuration must be applied:

```
apiVersion: networking.istio.io/v1alpha3
kind: ServiceEntry
metadata:
  name: external-forward-proxy
spec:
  hosts:
  - external-forward-proxy.net
  location: MESH_EXTERNAL
  ports:
  - number: 3128
    name: http
    protocol: HTTP
  resolution: DNS
---
apiVersion: networking.istio.io/v1alpha3
kind: ServiceEntry
metadata:
  name: google
spec:
```

Shared with Istio Community

```
  hosts:
  - www.google.com
  location: MESH_EXTERNAL
  ports:
  - number: 443
    name: tls
    protocol: TLS
  resolution: DNS
---
apiVersion: networking.istio.io/v1alpha3
kind: DestinationRule
metadata:
  name: tunnel-outbound-traffic-to-google-via-external-forward-proxy
spec:
  host: external-forward-proxy.net
  trafficPolicy:
    tunnel:
      targetHost: www.google.com
      targetPort: 443
    tls:
      mode: SIMPLE
      sni: external-forward-proxy.net
---
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: route-outbound-traffic-to-google-via-external-forward-proxy
spec:
  hosts:
  - www.google.com
  gateways:
  - mesh
  tls:
  - match:
    - sniHosts:
      - www.google.com
      port: 443
    route:
    - destination:
        host: external-forward-proxy.net
        port:
          number: 3128
```

That Istio configs apply to the sidecar a configuration similar to the one below:

```
  listeners:
  - name: "0.0.0.0_443"
    address:
```

```
      socket_address:
        address: 0.0.0.0
        port_value: 443
    listener_filters:
    - name: envoy.filters.listener.tls_inspector
      typed_config:
        "@type":
type.googleapis.com/envoy.extensions.filters.listener.tls_inspector.v3.TlsInspe
ctor
    filter_chains:
    - filter_chain_match:
        server_names:
        - "www.google.com"
      filters:
      - name: envoy.filters.network.tcp_proxy
        typed_config:
          "@type":
type.googleapis.com/envoy.extensions.filters.network.tcp_proxy.v3.TcpProxy
          cluster: outbound|3128||external-forward-proxy.net
          stat_prefix: outbound|3128||external-forward-proxy.net
          tunneling_config:
            hostname: "www.google.com:443"
  clusters:
  - name: outbound|3128||external-forward-proxy.net
    ...
    transport_socket:
      name: envoy.transport_sockets.tls
      typed_config:
        "@type":
type.googleapis.com/envoy.extensions.transport_sockets.tls.v3.UpstreamTlsContex
t
        sni: external-forward-proxy.net
```

In the “Background” section I explained the drawbacks of this configuration. A configuration that would solve these problems, is as follows:

```
  listeners:
  - name: "0.0.0.0_80"
    address:
      socket_address:
        address: 0.0.0.0
        port_value: 80
    filter_chains:
    - filters:
      - name: envoy.filters.network.http_connection_manager
        typed_config:
          "@type":
type.googleapis.com/envoy.extensions.filters.network.http_connection_manager.v3
.HttpConnectionManager
```

Shared with Istio Community

```
          http_filters:
          - name: envoy.filters.http.router
            typed_config:
              "@type":
type.googleapis.com/envoy.extensions.filters.http.router.v3.Router
          route_config:
            name: outbound|443||www.google.com
            virtual_hosts:
            - name: www.google.com
              domains:
              - "www.google.com"
              routes:
              - match:
                  prefix: "/"
                route:
                  cluster: outbound|443||www.google.com
          stat_prefix: outbound|443||www.google.com
  - name: "tunnel-www.google.com"
    address:
      pipe:
        path: "@/tunnel-www.google.com"
    listener_filters:
    - name: envoy.filters.listener.tls_inspector
      typed_config:
        "@type":
type.googleapis.com/envoy.extensions.filters.listener.tls_inspector.v3.TlsInspe
ctor
    filter_chains:
    - filter_chain_match:
        server_names:
        - "www.google.com"
      filters:
      - name: envoy.filters.network.tcp_proxy
        typed_config:
          "@type":
type.googleapis.com/envoy.extensions.filters.network.tcp_proxy.v3.TcpProxy
          cluster: outbound|3128||external-forward-proxy.net
          stat_prefix: outbound|3128||external-forward-proxy.net
          tunneling_config:
            hostname: "www.google.com:443"
  clusters:
  - name: outbound|443||www.google.com
    connect_timeout: 0.25s
    type: STATIC
    load_assignment:
      cluster_name: outbound|443||www.google.com
      endpoints:
      - lb_endpoints:
        - endpoint:
```

Shared with Istio Community

```
            address:
              pipe:
                path: "@/tunnel-www.google.com"
    transport_socket:
      name: envoy.transport_sockets.tls
      typed_config:
        "@type":
type.googleapis.com/envoy.extensions.transport_sockets.tls.v3.UpstreamTlsContex
t
        sni: www.google.com
  - name: outbound|3128||external-forward-proxy.net
    ...
    transport_socket:
      name: envoy.transport_sockets.tls
      typed_config:
        "@type":
type.googleapis.com/envoy.extensions.transport_sockets.tls.v3.UpstreamTlsContex
t
```

The first listener in this configuration accepts plain HTTP traffic and routes it to an intermediary TcpProxy listening on UDS socket. This operation makes it possible to count metrics for target service and upgrade connection to TLS independently on whether the first listener terminates TLS or not. If Istio applied this configuration, there wouldn't be a need to specify ports of google service entry as TCP or TLS, and HTTP could be used.
This approach with the second listener solves all limitations listed in the first section.

I implemented an EnvoyFilter as a proof of concept and it works as expected. You can also find more raw Envoy sample configurations here and test them with docker.

## Design Ideas

The first option assumes that we extend the existing API with optional field “proxy”. It could be backward compatible, so that when a proxy is defined, then Istio applies configuration following the new rules, otherwise the current logic is performed.

```
apiVersion: networking.istio.io/v1alpha3
kind: DestinationRule
metadata:
  name: tunnel-outbound-traffic-to-google-via-external-forward-proxy
spec:
  host: "*.google.com"
  trafficPolicy:
    tunnel:
      # Target host may seem to be redundant, but it must be
specified
      # because destination rule's host may start with wildcard,
      # so it can't be deduced from that field.
      targetHost: www.google.com
      targetPort: 443
      # Optional field for being backward compatible.
      # If not specified, current rules would be applied and
virtualoverridden
      # service would be necessary.
      # Otherwise, new logic would be performed.
      # This is required to remove necessity of redirecting traffic
      # with virtual service.
      proxy:
        # It does not create cluster and a related
        # Service or ServiceEntry must still be created
        # to know its protocol and port.
        host: external-forward-proxy.external.svc.cluster.local
        # In the future we could extend proxy capability to add
        # authorization headers, like Proxy-Authorization,
        # but Envoy SDS does not support tokens for now.
        authorization:
          credentialName: secret-with-token
    tls:
      mode: SIMPLE
      sni: www.google.com
---
apiVersion: networking.istio.io/v1alpha3
kind: DestinationRule
metadata:
  name: originate-tls-to-external-forward-proxy
```

Shared with Istio Community

```
spec:
  host: external-forward-proxy.net
  trafficPolicy:
    tls:
      mode: SIMPLE
      sni: external-forward-proxy.net
```

Having such an API, redirection with VirtualService would no longer be needed and all information required to properly create listeners and clusters would be available.

## Alternatives

Another solution is to introduce a completely new API intended only for tunneling. This API could enable tunneling traffic globally (for all mesh-external services) or for selected services.

```
apiVersion: networking.istio.io/v1beta1
kind: Tunnel
metadata:
  name: tunnel-traffic-to-all-mesh-external-services
spec:
  proxy:
    host: external-forward-proxy.net
---
apiVersion: networking.istio.io/v1beta1
kind: ServiceEntry
metadata:
  name: external-proxy
spec:
  hosts:
  - external-forward-proxy.net
  location: MESH_EXTERNAL
  ports:
  - number: 3128
    name: https
    # Other supported protocols: HTTP2-CONNECT, HTTP2-POST
    protocol: HTTP-CONNECT
  resolution: DNS
```

The Tunnel object in the example above enables tunneling traffic to all mesh-external services (`location: MESH_EXTERNAL`) through `external-forward-proxy.net` which is defined by a service entry. That service entry additionally specifies its protocol as **HTTP-CONNECT** to know whether to use CONNECT or POST by TcpProxy. If that external proxy accepts TLS traffic,

a destination rule with simple TLS would be needed.

```
apiVersion: networking.istio.io/v1alpha3
kind: DestinationRule
metadata:
  name: originate-tls-to-external-forward-proxy
spec:
  host: external-forward-proxy.net
  trafficPolicy:
    tls:
      mode: SIMPLE
      sni: external-forward-proxy.net
```

If tunneling would have to be applied only to a subset of services, then we could use selectors to apply the Tunnel object. It's important to note here that selectors specify to which services traffic is tunneled, not to which sidecars/gateways the configuration is applied.

```
apiVersion: networking.istio.io/v1beta1
kind: Tunnel
metadata:
  name: tunnel-traffic-to-google-via-external-forward-proxy
spec:
  selector:
    matchLabels:
      service: google.com
  proxy:
    host: external-forward-proxy.net
---
apiVersion: networking.istio.io/v1beta1
kind: ServiceEntry
metadata:
  name: google
  labels:
    service: google.com
spec:
  hosts:
  # When a service entry specifies multiple hostnames,
  # then the first one is used as target host in the CONNECT request.
  - google.com
  - www.google.com
  location: MESH_EXTERNAL
  ports:
  - number: 443
    name: https
    protocol: TLS
```

```
  resolution: DNS
```

ServiceEntry can specify multiple hostnames, so this solution assumes that the first host is used in a CONNECT request as a target host.
We could also support deduction of the target host from SNI by setting **`targetHost: AUTO_SNI`** or original destination by setting **`targetHost: ORIGINAL_DST`** (but I'm not sure if this is possible for now).

```
apiVersion: networking.istio.io/v1beta1
kind: Tunnel
metadata:
  name: tunnel-traffic-to-all-mesh-external-services-with-auto-sni
spec:
  targetHost: AUTO_SNI
  proxy:
    host: external-forward-proxy.net
```

In the future we could also add an option to enable authentication, e.g. Proxy-Authorization header, but currently SDS does not support this use case. Maybe it could be done in istio-proxy.

```
apiVersion: networking.istio.io/v1beta1
kind: Tunnel
metadata:
  name: tunnel-traffic-to-all-mesh-external-services
spec:
  proxy:
    host: external-forward-proxy.net
    authentication:
      credentialName: secret-name
```

## Questions/Concerns

1. Currently Envoy does not collect metrics of type **envoy_http_*** for requests to tunnel proxy. The only available HTTP-related metrics for tunnel proxy are:
   - envoy_cluster_http1_dropped_headers_with_underscores
   - envoy_cluster_http1_metadata_not_supported_error
   - envoy_cluster_http1_requests_rejected_with_underscores_in_headers
   - envoy_cluster_http1_response_flood
   - envoy_cluster_upstream_cx_http1_total
   - envoy_cluster_upstream_cx_http2_total

Shared with Istio Community

- envoy_cluster_upstream_cx_http3_total
- envoy_cluster_upstream_http3_broken

2. How to pass original destination IP from the first listener to the tunneling TcpProxy? Original destination will change after routing traffic between listeners, so we have to figure out how to configure Envoy for this purpose or maybe it could be implemented somehow in istio-proxy.
3. How does the second solution fit into the plans for “Gateway improvements” (handling HTTP CONNECT by egress and east-west gateways)?
4. How could we version Tunnel resource to start as alpha? Could it be for example networking.istio.io/v1alpha1 to not go to beta?

Shared with Istio Community