Envoy Introduction

Suresh Kumar Ponnusamy

April 25, 2018

Outline

What is Envoy I

- L4/L7 proxy and communication bus
- Goal: "The network should be transparent to applications. When network and application problems do occur it should be easy to determine the source of the problem"
- Data plane: Stateless proxy that routes, load balances etc
  - A standalone/stateless proxy with configuration
  - Example: HAProxy, nginx, Envoy, Linkerd etc
- Control plane: Takes the above stateless proxies and turns them into a distributed system.
  - Distributed, dynamic proxy configuration management
  - Example: Istio, Nelson, SmartStack etc

What is Envoy II

- Typical usage: Envoy runs as sidecar process in each machine, all the application communications (ingress/egress) go through this local sidecar Envoy (== "data plane") and Envoy knows how to reach other services. And Envoy itself is configurable dynamically via "control plane"
- Relationship to Istio: Envoy is the "data plane" and other components in Istio fill the "control plane" role, among others.
- More info here

Features I

- Out of process: Can support any language, not a library
- Proxy: L3/L4, L7
- Routing: L7
- Load balancing: with circuit breaking, global rate limiting, request shadowing, outlier detection (== passive health checking) etc
- Edge proxy
  - TLS(+SNI)
  - HTTP 1.1/2
  - L7 routing
- Configuration
  - Static configuration: Can start standalone, pre-configured
  - Dynamic configuration: Almost all aspects of Envoy are configurable on the fly, without having to restart the proxy
- Observability: statsd, distributed tracing support

Features II

- HTTP/2 support
- gRPC support
  - Load balancing
  - HTTP/1.1 bridge
- Filters: L4, L7
  - MongoDB L7 support: sniffing, stats, logging
  - DynamoDB L7 support: sniffing, stats, logging
  - Other custom filters one can write
- Service discovery
  - DNS
  - REST: gRPC based service discovery
- Health checking
- Hot restart

Components/terminology I

- Downstream: The one connecting to Envoy
- Upstream: The one Envoy connects/forwards to
- Listeners: A named network location that downstream clients connect to
  - HAProxy equivalent is "frontend"
  - Dynamically configurable via Listener Discovery Service (LDS)
- Clusters: A group of logically similar upstream hosts that Envoy connects to
  - HAProxy equivalent is "backend"
  - Cluster membership can be statically configured or discovered via Service Discovery
    - DNS
    - Endpoint Discovery Service (EDS), a REST/gRPC based service
  - Active health checking
  - Requests are load balanced among the cluster members based on load balancer policy

Components/terminology II

- Cluster themselves are discovered via Cluster Discovery Service (CDS)
- Routes: L7 routing
  - HAProxy equivalent is "ACL"
  - Routes can be statically configured or discovered via Route Discovery Service (RDS)

Design/Architecture I

- Envoy is out of process
- Envoy is multi-threaded + evented
  - There are N number of threads (usually equal to number of CPUs, but is configurable).
  - Each thread has its own event loop (using libevent)
  - A connection once accepted stays with the accepting thread, so code doesn't have to be multi-thread aware for most use cases.
- Listeners
  - TCP listeners
  - Listener filters
  - On new connection, filter chain associated with listener is invoked
  - This is where most of the proxy functionality is in: for example, http proxy is a filter chained here
  - Listener Discovery Service (LDS): Service used for adding/removing listeners
- L3/L4 filters

Design/Architecture II

- HTTP connection management filter
  - A filter that implements http protocol: http 1.1, http 2.0, WebSockets
  - access log, requestid generation, distributed tracing, header manipulation, stats etc
  - Route table, configured either statically or dynamically via Route Discovery Service (RDS)
  - HTTP routing
    - Path based (prefix, exact)
    - Header matching
    - WS upgrades
    - Traffic shifting
    - Traffic splitting
    - Retries
  - HTTP filters
- gRPC
  - Bridge filter for languages that has immature gRPC implementation. Client can send requests in http 1.1, Envoy will do the necessary translation.

Design/Architecture III

- gRPC JSON proxy: Client can send requests with JSON payload, Envoy will do the necessary translation and proxy it to gRPC service
- gRPC-Web support: Clients can send a gRPC request to Envoy over HTTP/1.1, Envoy can translate and proxy it to gRPC server
- Envoy uses gRPC in both data plane and control plane

- Cluster manager
  - Cluster Discovery Service (CDS) is used for adding/removing clusters
  - Cluster members discovered via Endpoint Service Discovery (EDS)
- Health check
  - Active and passive
  - Outlier detection (consecutive 5xxs, response time is greater than other machines in the same cluster etc)
- TLS
  - SNI, ALPN, CRL, Session Resumption
- Load balancers
  - Circuit breaking

Design/Architecture IV

- Rate limiting
- Statistics
  - Statsd, DogStatsd, metrics_service (gRPC based one)
- Distributed tracing
  - Request ID generation
  - Integration to external services: LightStep, Zipkin, Jaeger etc
- TCP Proxy
- MongoDB, DynamoDB, Redis filters
- Configuration update
  - Static configuration
    - Hotrestart
    - via DNS (limitations apply)
  - Dynamic configuration
    - For cluster members using EDS
    - For clusters, using CDS
    - For routes, using RDS
    - For listeners, using LDS

Deployment types I

- Sidecar proxy for "service to service"

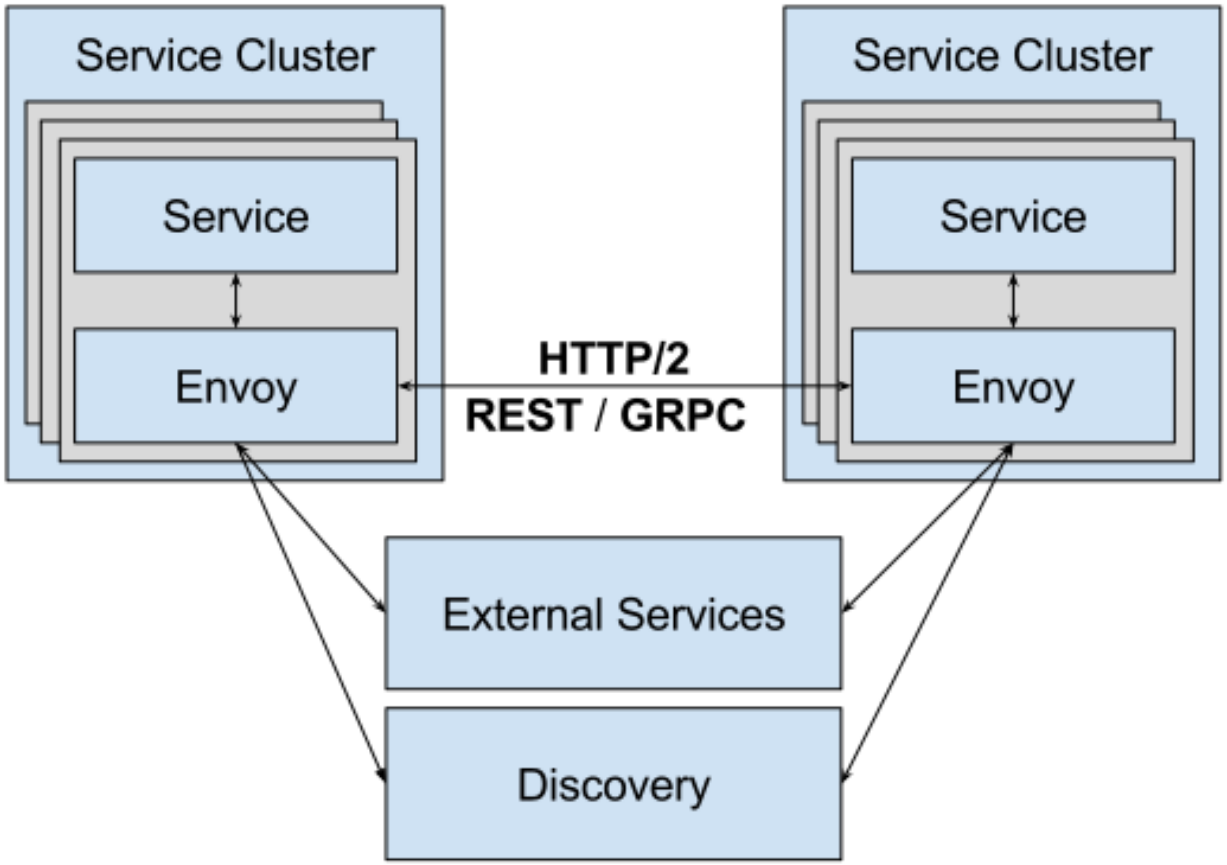

Figure: Service to Service sidecar

Deployment types II

- Edge proxy

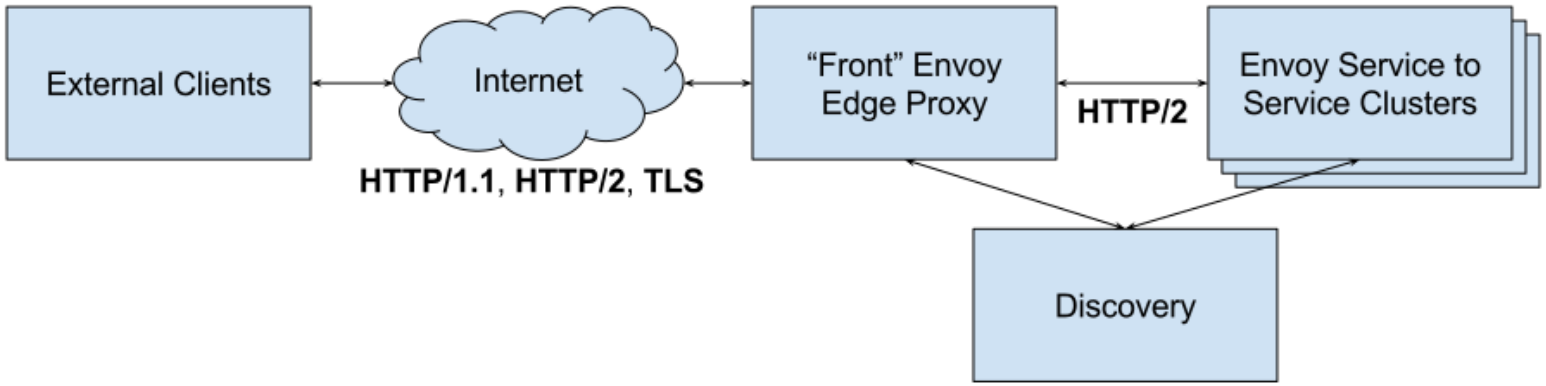

Figure: Edge/Front proxy

- Double edge proxy

Deployment types III

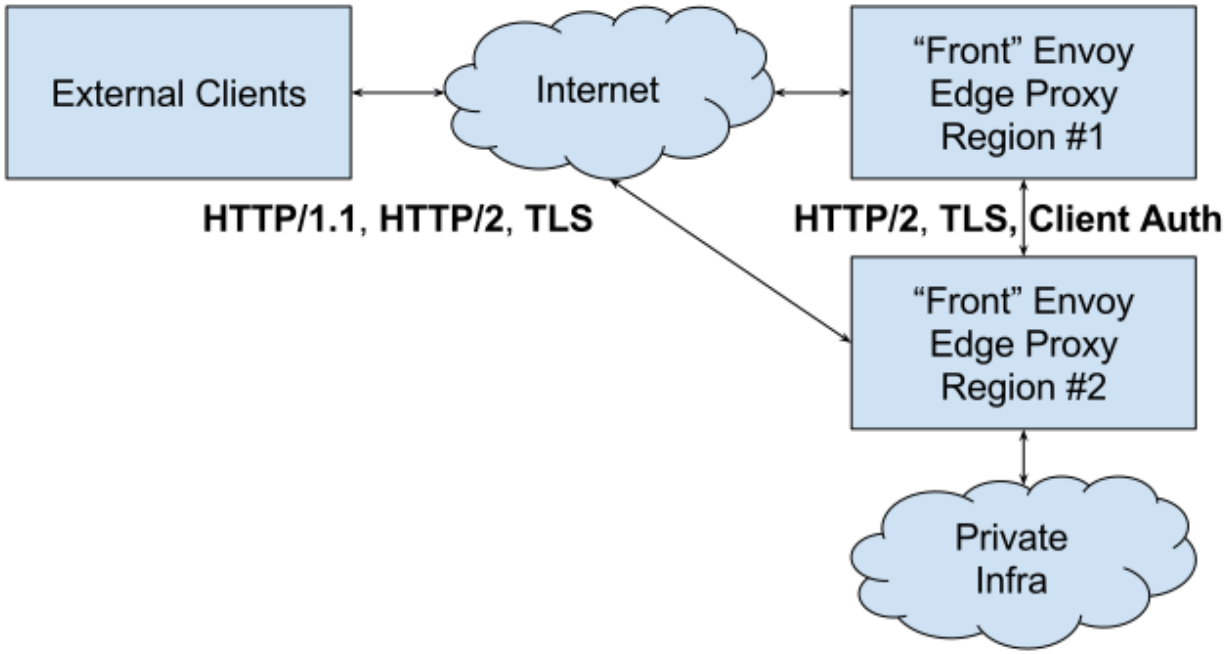

Figure: Double edge proxy

Why Envoy, what value can it add for Freshdesk I

- Better observability within the infra
  - Currently we rely on APM on app side and ELB CloudWatch on edge, everything else in-between is invisible
  - This is going to be a much bigger problem going forward when more microservices are deployed
- Service discovery
  - Currently we rely on OpsWorks to do service discovery
- Clustering
  - Currently we rely on OpsWorks to do the layering, which gets translated into HAProxy "backend" via Chef recipe. Adding/removing machines is clumsy, takes time and error prone.
  - Could potentially help us on minipod effort as well
- Configuration changes
  - Currently we rely on OpsWorks + Stack Settings + Chef recipes to push configuration changes on the fly
  - A system that is designed for dynamic configuration could ease some of these operational challenges

Why Envoy, what value can it add for Freshdesk II

- Customizability
  - We can add custom functionality via filters, both at L7 and L4
  - We can move few things from our Rails app's middleware into Envoy filter, efficient and highly scalable (== apps are not stuck on it, they just do their business logic)
    - Example: when we moved IP blacklisting to HAProxy, we reduced ~10ms on every http request we serve, and improved the overall throughput as well
  - Few things I could think of doing it in Envoy:
    - Ratelimiting
    - WAF features like domain/IP blacklisting etc
    - JWT auth check (already Istio has)?
    - Long shot: MySQL proxying + sharding

Why Envoy, what value can it add for Freshdesk III

- Moving these things out of application, into Envoy, opens up the possibility of choosing appropriate language for each microservice
  - Imagine we want to implement "ticket list" as a microservice in Java
  - If we have most of needed middlewares in Envoy, we can just focus on the business logic (== don't have to worry about how authentication works, how to ratelimit, where/how to find configuration files, how to do sharding etc)
- K8S: Preparing ground for K8S adaption

Possible deployments for Freshdesk I

- As a sidecar in each app machine, infront of nginx/passenger
  - We can configure it to handle all ingress/egress traffic for the application
  - Better observability
  - Other features like circuit breaking, rate limiting etc
  - Distributed tracing?
- As a routing server (replacing HAProxy)
  - Dynamic configuration, service discovery etc
  - We can get out of OpsWorks (w.r.t service discovery and config management)
  - We will still do TLS termination at ELB
- As an edge server
  - Doing TLS termination at our end (for SNI/custom non-wildcard certificates)
  - We will still have NLB/ALB/ELB at the front, just doing TCP termination, firewalling etc.

Q&A