[译文]xDS REST and gRPC protocol

资源类型

xDS API中的每个配置资源都有一个与之相关的类型。资源类型遵循一个版本计划。资源类型是独立于下面描述的运输工具的版本。

以下v3 xDS资源类型被支持。

下面出现了类型URLs的概念，其形式为 `type.googleapis.com/` `<资源类型>` - 例如， `type.googleapis.com/envoy.config.cluster.v3.Cluster` 代表 `Cluster` 资源。在Envoy的各种请求和管理服务器的响应中，都会说明资源类型URL。

文件系统订阅

交付动态配置的最简单方法是将其放置在ConfigSource中指定的一个众所周知的路径。Envoy将使用 `inotify` (macOS上的 `kqueue`)来监控文件的变化，并在更新时解析文件中的DiscoveryResponseproto。二进制protobufs、JSON、YAML和proto文本是DiscoveryResponse的支持格式。

除了统计计数器和日志之外，没有任何机制可用于文件系统订阅ACK/NACK更新。如果发生配置更新拒绝，xDS API的最后有效配置将继续适用。

流式的gRPC订阅

API流程

对于典型的HTTP路由场景，客户端配置的核心资源类型是 `Listener` 、 `RouteConfiguration` 、 `Cluster` 和 `ClusterLoadAssignment` 。每个 `Listener` 资源可以指向一个 `RouteConfiguration` 资源，该资源可以指向一个或多个 `Cluster` 资源，而每个 `Cluster` 资源可以指向一个 `ClusterLoadAssignment` 资源。

Envoy在启动时获取所有 `Listener` 和 `Cluster` 资源。然后，它获取 `Listener` 和 `Cluster` 资源所需要的任何 `RouteConfiguration` 和 `ClusterLoadAssignment` 资源。实际上，每一个 `Listener` 或 `Cluster` 资源都是Envoy配置树的一部分的根。

一个非代理客户端，如gRPC，可能开始时只获取它感兴趣的特定 `Listener` 资源。然后获取这些 `Listener` 资源所需的 `RouteConfiguration` 资源，接着是这些 `RouteConfiguration` 资源所需的任何 `Cluster` 资源，然后是 `Cluster` 资源所需的 `ClusterLoadAssignment` 资源。实际上，原始的 `Listener` 资源是客户端配置树的根。

xDS传输协议的变体

四种变体

通过流式 `gRPC` 使用的 `xDS` 传输协议有四种变体，它们涵盖了两个维度的所有组合。

第一个维度是世界状态（`SotW`）与`增量`。`SotW` 方法是 `xDS` 最初使用的机制，客户必须在每次请求中指定其感兴趣的所有资源名称，对于 `LDS` 和 `CDS` 资源，服务器必须在每次请求中返回客户已订阅的所有资源。这意味着，如果客户已经订阅了99个资源，并想增加一个资源，它必须发送一个包含所有100个资源名称的请求，而不是只发送一个新的资源。而对于 `LDS` 和 `CDS` 资源，服务器必须通过发送所有100个资源来响应，即使已经订阅的99个资源没有变化。这种机制可能限制了可扩展性，这就是为什么后来引入了增量协议变体。增量方法允许客户端和服务器只表示相对于其先前状态的延迟--也就是说，客户端可以说它想增加或删除对某个特定资源名称的订阅，而不重新发送那些没有变化的资源，而服务器可以只为那些已经变化的资源发送更新。增量协议还提供了一种懒惰加载资源的机制。关于增量协议的细节，请看下面的 `增量xDS`。

第二个维度是为每个资源类型使用单独的gRPC流与将所有资源类型聚合到一个gRPC流。前一种方法是xDS使用的原始机制，它提供了一个最终的一致性模型。后一种方法是为需要明确控制顺序的环境而添加的。详情请见下面的最终一致性考虑。

所以，xDS传输协议的四个变体是：

1. 世界的状态（基础xDS）：SotW，每种资源类型都有单独的gRPC流
2. 增量型xDS：增量型，每种资源类型都有单独的gRPC流
3. 聚合发现服务（ADS）：SotW，所有资源类型的聚合流
4. 递增式ADS：递增式，所有资源类型的聚合流

每个变体的RPC服务和方法

对于非聚合协议变体，每种资源类型都有一个单独的RPC服务。这些RPC服务中的每一个都可以为SotW和Incremental协议变体提供一个方法。下面是每种资源类型的RPC服务和方法：

- Listener: Listener Discovery Service (LDS) - SotW: ListenerDiscoveryService.StreamListeners - Incremental: ListenerDiscoveryService.DeltaListeners
- RouteConfiguration: Route Discovery Service (RDS) - SotW: RouteDiscoveryService.StreamRoutes - Incremental: RouteDiscoveryService.DeltaRoutes
- ScopedRouteConfiguration: Scoped Route Discovery Service (SRDS) - SotW: ScopedRouteDiscoveryService.StreamScopedRoutes - Incremental: ScopedRouteDiscoveryService.DeltaScopedRoutes
- VirtualHost: Virtual Host Discovery Service (VHDS) - SotW: N/A - Incremental: VirtualHostDiscoveryService.DeltaVirtualHosts
- Cluster: Cluster Discovery Service (CDS) - SotW: ClusterDiscoveryService.StreamClusters - Incremental: ClusterDiscoveryService.DeltaClusters
- ClusterLoadAssignment: Endpoint Discovery Service (EDS) - SotW: EndpointDiscoveryService.StreamEndpoints - Incremental: EndpointDiscoveryService.DeltaEndpoints

- Secret: Secret Discovery Service (SDS) - SotW: SecretDiscoveryService.StreamSecrets - Incremental: SecretDiscoveryService.DeltaSecrets
- Runtime: Runtime Discovery Service (RTDS) - SotW: RuntimeDiscoveryService.StreamRuntime - Incremental: RuntimeDiscoveryService.DeltaRuntime

在聚合协议变体中，所有资源类型都在单一的gRPC流中被复用，其中每个资源类型都被视为聚合流中的一个单独的逻辑流。实际上，它只是通过将每种资源类型的请求和响应视为单个聚合流上的单独子流，将上述所有单独的API合并为一个流。聚合协议变体的RPC服务和方法是：

- SotW: AggregatedDiscoveryService.StreamAggregatedResources
- Incremental: AggregatedDiscoveryService.DeltaAggregatedResources

对于所有的SotW方法，请求类型是DiscoveryRequest，响应类型是DiscoveryResponse。

对于所有的Incremental方法，请求类型是DeltaDiscoveryRequest，响应类型是DeltaDiscoveryResponse。

配置要使用的变体

在xDS API中，ConfigSource消息指出如何获得特定类型的资源。如果ConfigSource包含一个gRPCApiConfigSource，它指向管理服务器的上游集群；这将为每个xDS资源类型启动一个独立的双向gRPC流，有可能指向不同的管理服务器。如果ConfigSource包含一个AggregatedConfigSource，它告诉客户端使用ADS。

目前，客户端被期望得到一些本地配置，告诉它如何获得Listener和Cluster资源。Listener资源可能包括一个ConfigSource，指示如何获得RouteConfiguration资源，而Cluster资源可能包括一个ConfigSource，指示如何获得ClusterLoadAssignment资源。

客户端配置

在Envoy中，bootstrap文件包含两个ConfigSource消息，一个表示如何获得Listener资源，另一个表示如何获得Cluster资源。它还包含一个单独的ApiConfigSource消息，指示如何联系ADS服务器，只要ConfigSource消息（无论是在引导文件中还是在从管理服务器获得的Listener或Cluster资源中）包含一个AggregatedConfigSource消息，就会使用这个消息。

在使用xDS的gRPC客户端中，只支持ADS，bootstrap文件包含ADS服务器的名称，它将用于所有资源。在Listener和Cluster资源中的ConfigSource消息必须包含AggregatedConfigSource消息。

xDS传输协议

除了上面描述的资源类型版本外，xDS线协议还有一个与之相关的传输版本。这为DiscoveryRequest和DiscoveryResponse等消息提供了类型版本。它也被编码在gRPC方法名称中，所以服务器可以根据客户端调用的方法来确定它的版本。

基本协议概述

每个xDS流以来自客户端的DiscoveryRequest开始，其中指定了要订阅的资源列表、与订阅的资源相对应的类型URL、节点标识符，以及一个可选的资源类型实例版本，表明客户端已经看到的资源类型的最新版本（详见ACK/NACK和资源类型实例版本）。

然后，服务器将发送一个DiscoveryResponse，其中包含客户端订阅的任何资源，这些资源自客户端表示它看到的最后一个资源类型实例版本以来已经改变。当订阅的资源发生变化时，服务器可以在任何时候发送额外的响应。

每当客户端收到一个新的响应，它将发送另一个请求，表明响应中的资源是否有效，详见ACK/NACK和资源类型实例版本。

所有的服务器响应将包含一个nonce，所有来自客户端的后续请求必须将responce字段设置为该流上从服务器收到的最新的nonce。这允许服务器确定一个给定的请求与哪个响应相关，这避免了SotW协议变体中的各种竞争条件。请注意，nonce只在单个xDS流的上下文中有效；它不能在流重启后继续有效。

一个流上只有第一个请求被保证携带节点标识符。同一流上的后续发现请求可能携带一个空的节点标识符。无论同一流上的发现响应是否被接受，这都是真实的。如果在流上出现不止一次，节点标识符应该始终是相同的。因此，只检查第一个消息的节点标识符就足够了。

ACK/NACK和资源类型实例版本

每个xDS资源类型都有一个版本字符串，表示该资源类型的版本。每当该类型的一个资源发生变化，版本就会改变。

在xDS服务器发送的响应中，version_info字段表示该资源类型的当前版本。然后客户向服务器发送另一个请求，version_info字段表示客户看到的最新的有效版本。这为服务器提供了一种方法，以确定它何时发送了一个客户认为无效的版本。

(在增量协议变体中，资源类型实例版本是由服务器在system_version_info字段中发送。然而，这一信息实际上并不被客户端用来沟通哪些资源是有效的，因为增量API变体有一个单独的机制来实现这一目的）。)

资源类型实例版本对每个资源类型都是独立的。当使用聚合协议变体时，每个资源类型都有自己的版本，即使所有资源类型都在同一个流上被发送。

资源类型的实例版本对每个xDS服务器来说也是独立的（其中一个xDS服务器由唯一的ConfigSource标识）。当从多个xDS服务器获得一个给定类型的资源时，每个xDS服务器将有一个不同的版本概念。

注意，资源类型的版本不是单个xDS流的属性，而是资源本身的属性。如果流被破坏，客户创建了一个新的流，客户在新流上的初始请求应该表明客户在前一个流上看到的最新版本。服务器可以决定通过不重新发送客户端在前一个流中已经看到的资源来进行优化，但前提是他们知道客户端没有订阅之前没有订阅过的新资源。例如，当唯一的订阅是通配符订阅时，服务器对LDS和CDS做这种优化通常是安全的，而且在客户端将始终订阅完全相同的资源集的环境中，做这种优化也是安全的

一个EDS请求的例子可能是:

```CSS
version_info:
node: { id: envoy }
resource_names:
- foo
- bar
type_url: type.googleapis.com/envoy.config.endpoint.v3.ClusterLoadAssignment
response_nonce:
```

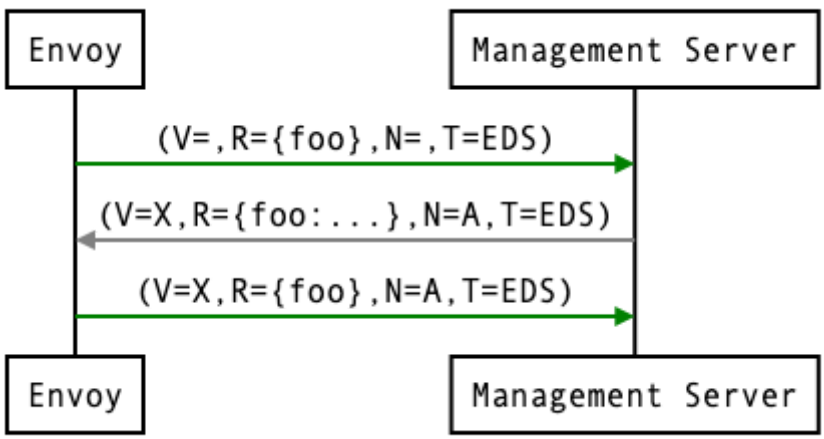
管理服务器可以立即或在所请求的资源可用时以DiscoveryResponse进行答复，例如:

```CSS
version_info: X
resources:
- foo ClusterLoadAssignment proto encoding
- bar ClusterLoadAssignment proto encoding
type_url: type.googleapis.com/envoy.config.endpoint.v3.ClusterLoadAssignment
nonce: A
```

在处理完DiscoveryResponse后，Envoy将在流上发送一个新的请求，指定最后成功应用的版本和管理服务器提供的nonce。版本为Envoy和管理服务器提供了一个关于当前应用的配置的共享概念，以及一个用于ACK/NACK配置更新的机制。

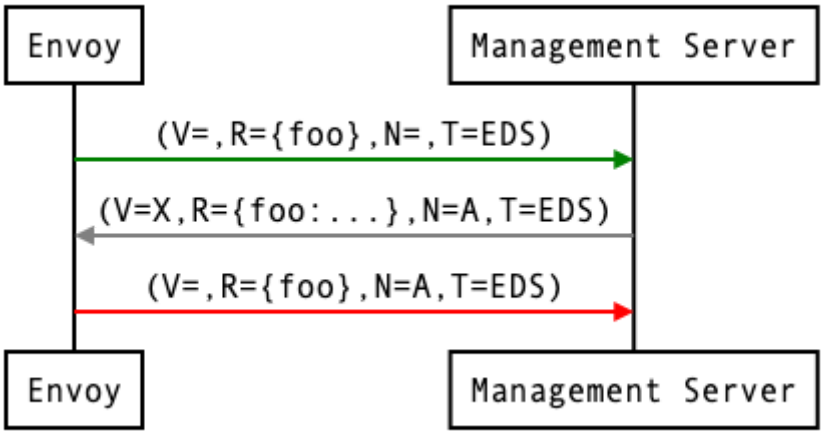
ACK

如果更新被成功应用，version_info将是**X**，如顺序图中所示:



NACK

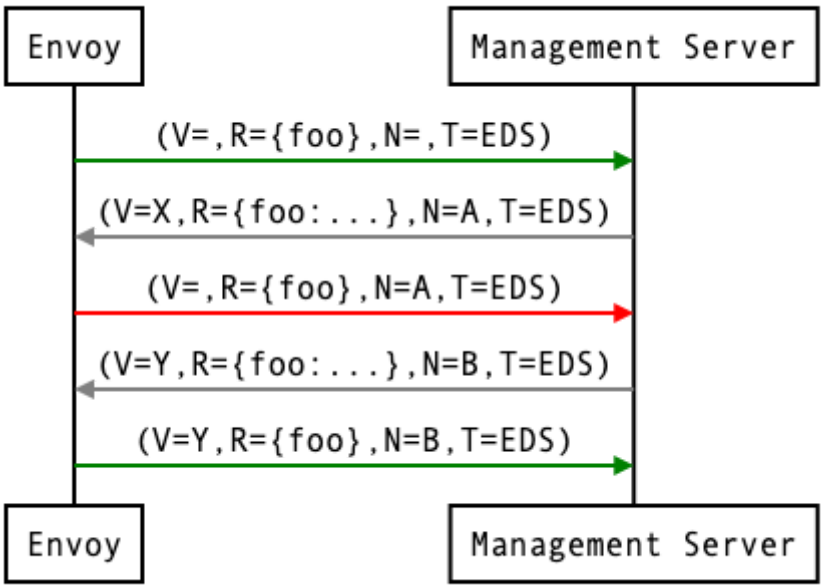
如果Envoy反而拒绝了配置更新**X**，它将回复error_detail弹出的信息和它之前的版本，在这种情况下是空的初始版本。error_detail有更多的细节，围绕着消息字段中填充的确切错误信息:



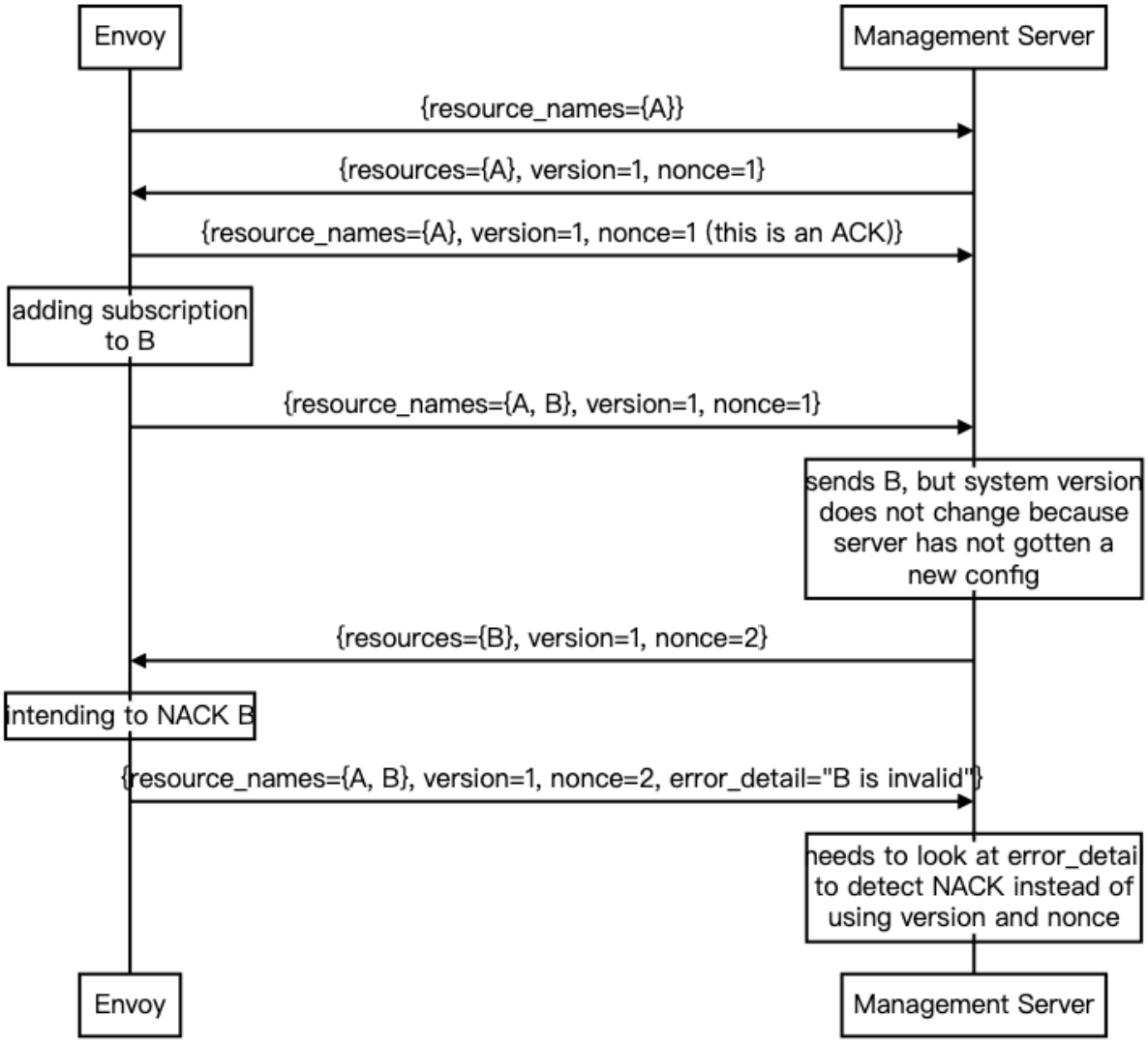
在顺序图中，使用以下格式来缩写信息:

- *DiscoveryRequest*: (V=version_info,R=resource_names,N=response_nonce,T=type_url)
- *DiscoveryResponse*: (V=version_info,R=resources,N=nonce,T=type_url)

在NACK之后，API更新可能在新的版本Y上成功:



服务器检测NACK的首选机制是在客户端发送的请求中寻找error_detail字段的存在。一些老的服务器可能会通过查看请求中的版本和nonce来检测NACK：如果请求中的版本不等于服务器发送的nonce，那么客户就拒绝了最新的版本。然而，这种方法对LDS和CDS以外的API不起作用，因为客户可能会动态地改变他们所订阅的资源集，除非服务器以某种方式安排在任何一个客户订阅新资源时增加资源类型实例的版本。具体来说，考虑下面的例子:



ACK和NACK语义总结

- xDS客户端应该对从管理服务器收到的每个DiscoveryResponse进行ACK或NACK。response_nonce字段告诉服务器，ACK或NACK与它的哪个响应相关。
- ACK表示配置更新成功，并包含来自DiscoveryResponse的[版本信息]。
- NACK表示配置不成功，并通过error_detail字段的出现来表示。version_info表示客户正在使用的最新版本，尽管在客户从现有版本订阅了一个新资源，而该新资源是无效的情况下，这可能不是一个旧版本（见上面的例子）。

何时发送更新

管理服务器应该只在DiscoveryResponse中的资源发生变化时向Envoy客户端发送更新。Envoy在任何DiscoveryResponse被接受或拒绝后，立即用包含ACK/NACK的DiscoveryRequest进行回复。如果管理服务器提供相同的资源集，而不是等待发生变化，就会在客户端和管理服务器上造成不必要的工作，这可能会严重影响性能。

在一个流中，新的DiscoveryRequests会取代之前任何具有相同资源类型的DiscoveryRequests。这意味着管理服务器只需要对每个流中任何给定资源类型的最新DiscoveryRequest作出响应。

客户端如何指定要返回的资源

xDS请求允许客户指定一组资源名称，作为对服务器的提示，说明客户对哪些资源感兴趣。在SotW协议变体中，这是通过DiscoveryRequest中指定的resource_names完成的；在增量协议变体中，这是通过DeltaDiscoveryRequest中的resource_names_subscribe和resource_names_unsubscribe字段完成。

通常情况下（例外情况见下文），请求必须指定客户端感兴趣的资源名称集。管理服务器必须提供所请求的资源，如果它们存在的话。客户端将默默地忽略任何提供的、没有明确请求的资源。当客户端发送一个新的请求，改变了被请求的资源集时，服务器必须重新发送任何新请求的资源，即使它之前发送了这些资源而没有被请求，并且自那时以来资源没有改变。如果资源名称列表变成空的，这意味着客户端对指定类型的任何资源不再感兴趣。

对于Listener和Cluster资源类型，也有一个 "通配符 "订阅，在订阅特殊名称 "*"时触发。在这种情况下，服务器应该使用站点特定的业务逻辑来确定客户感兴趣的全部资源集，通常是基于客户的Node标识。

由于历史原因，如果客户端发送了一个给定资源类型的请求，但从未明确订阅过任何资源名称（即在SotW中，该资源类型流中的所有请求都有一个空的resource_names字段，或者在增量中，从未在该资源类型流中发送过一个非空的resource_names_subscribe字段的请求），服务器应该将其与处理客户端明确订阅了 "*"的情况相同。然而，一旦客户端明确订阅了一个资源名称（无论是 "*"还是其他名称），那么这种传统的语义就不再可用了；此时，清除订阅的资源列表被解释为取消订阅（参见从资源中取消订阅），而不是订阅 "*"。

例如，在SotW中：

- 客户端发送了一个resource_names未设置的请求。服务器将其解释为对 "*"的订阅。
- 客户端发送一个请求，resource_names设置为 "*"和 "A"。服务器将此解释为继续对 "*"的现有订阅，并对 "A "添加新的订阅。
- 客户端发送一个请求，resource_names设置为 "A"。服务器将此解释为取消对 "*"的订阅并继续对 "A "的现有订阅。
- 客户端发送请求时，resource_names没有设置。服务器将此解释为取消对 "A "的订阅（即，客户现在已经取消了对所有资源的订阅）。尽管这个请求与第一个请求相同，但它不会被解释为通配符订阅，因为在这个流上已经有一个为这个资源类型设置resource_names字段的请求。

而在增量中：

- 客户端发送了一个resource_names_subscribe未设置的请求。服务器将其解释为对 "*"的订阅。
- 客户端发送请求，resource_names_subscribe设置为 "A"。服务器将此解释为继续现有的对 "*"的订阅，并添加一个对 "A "的新订阅。- 客户端发送请求，resource_names_unsubscribe设置为 "*"。服务器将此解释为取消对 "*"的订阅并继续对 "A "的现有订阅。

- 客户端发送请求，resource_names_unsubscribe设置为 "A"。服务器将此解释为取消对 "A "的订阅（即，客户现在已经取消了对所有资源的订阅）。虽然现在订阅的资源集是空的，就像初始请求后一样，但它不会被解释为通配符订阅，因为之前在这个流上有一个针对这个资源类型的请求，设置了resource_names_subscribe 字段。

Envoy将始终使用通配符订阅Listener和Cluster资源。然而，其他xDS客户端（如使用xDS的gRPC客户端）可以明确订阅这些资源类型的特定资源名称，例如，如果他们只有一个单子监听器，并且已经从一些带外配置中知道其名称。

将资源分组为响应

在增量协议变体中，服务器在自己的响应中发送每个资源。这意味着，如果服务器之前发送了100个资源，而其中只有一个资源发生了变化，那么它可以发送一个只包含变化的资源的响应；它不需要重新发送没有变化的99个资源，而且客户端不得删除没有变化的资源。

在SotW协议变体中，除Listener和Cluster外，所有资源类型都以与增量协议变体相同的方式被分组为响应。然而，Listener和Cluster资源类型的处理方式不同：服务器必须包括世界的完整状态，这意味着必须包括客户端需要的所有相关类型的资源，即使它们自上次响应以来没有变化。这意味着，如果服务器之前发送了100个资源，而其中只有一个资源发生了变化，那么它必须重新发送所有的100个资源，甚至是没有被修改的99个。

请注意，所有的协议变体都是以整个命名的资源为单位进行操作。没有任何机制可以提供命名资源中重复字段的增量更新。最值得注意的是，目前还没有机制来增量更新EDS响应中的单个端点。

服务器发送一个包含两次相同资源名称的响应是一个错误。客户端应该拒绝包含同一资源名称的多个实例的响应。

删除资源

在增量协议的变体中，服务器通过响应中的removement_resources字段向客户端发出信号，表示应该删除某个资源。这告诉客户端从其本地缓存中删除该资源。

在SotW协议的变体中，删除资源的标准更加复杂。对于Listener和Cluster资源类型，如果以前看到的资源没有出现在新的响应中，这表明该资源已被删除，客户端必须删除它；不包含资源的响应意味着要删除该类型的所有资源。然而，对于其他资源类型，API没有提供任何机制让服务器告诉客户端资源已经被删除；相反，删除是通过父资源被改变为不再引用子资源来隐含地表示的。例如，当客户端收到LDS更新，删除了之前指向RouteConfigurationA的Listener时，如果没有其他Listener指向RouteConfigurationA，那么客户端可能会删除A。对于这些资源类型，从客户端的角度来看，空DiscoveryResponse实际上是一个无用的补偿。

了解所请求的资源不存在的情况

SotW协议的变体没有提供任何明确的机制来确定所请求的资源何时不存在。

对于Listener和Cluster资源类型的响应必须包括客户端请求的所有资源。然而，客户可能无法仅根据响应中不存在的资源而知道该资源不存在，因为更新的交付最终是一致的：如果客户最初发送了对

资源A的请求，然后发送了对资源A和B的请求，然后看到只包含资源A的响应，客户不能得出资源B不存在的结论，因为该响应可能是在服务器看到第二个请求之前，根据第一个请求发送。

对于其它资源类型，因为每个资源都可以在它自己的响应中发送，所以没有办法从下一个响应中知道新请求的资源是否存在，因为下一个响应可能是先前已经订阅的另一个资源的无关的更新。

因此，客户端在发送新资源的请求后，应该使用一个超时（建议持续时间为15秒）时间，超时后如果没有收到资源，他们会认为请求的资源不存在。在Envoy中，对于RouteConfiguration和ClusterLoadAssignment资源，在资源预热过程中会这样做。

请注意，即使所请求的资源在客户端请求时不存在，该资源也可能在任何时候被创建。管理服务器必须记住客户端请求的资源集，如果这些资源中的一个后来突然出现，服务器必须向客户端发送更新，告知其新资源的存在。最初看到一个不存在的资源的客户必须准备好随时创建该资源。

从资源中退订

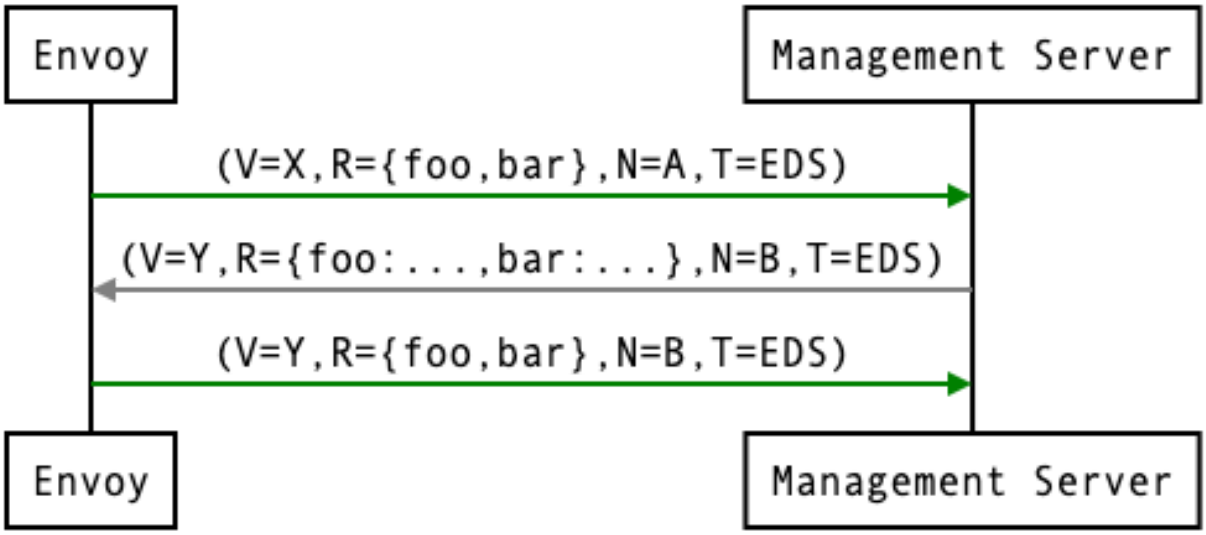
在增量协议变体中，可以通过resource_names_unsubscribe字段取消对资源的订阅。

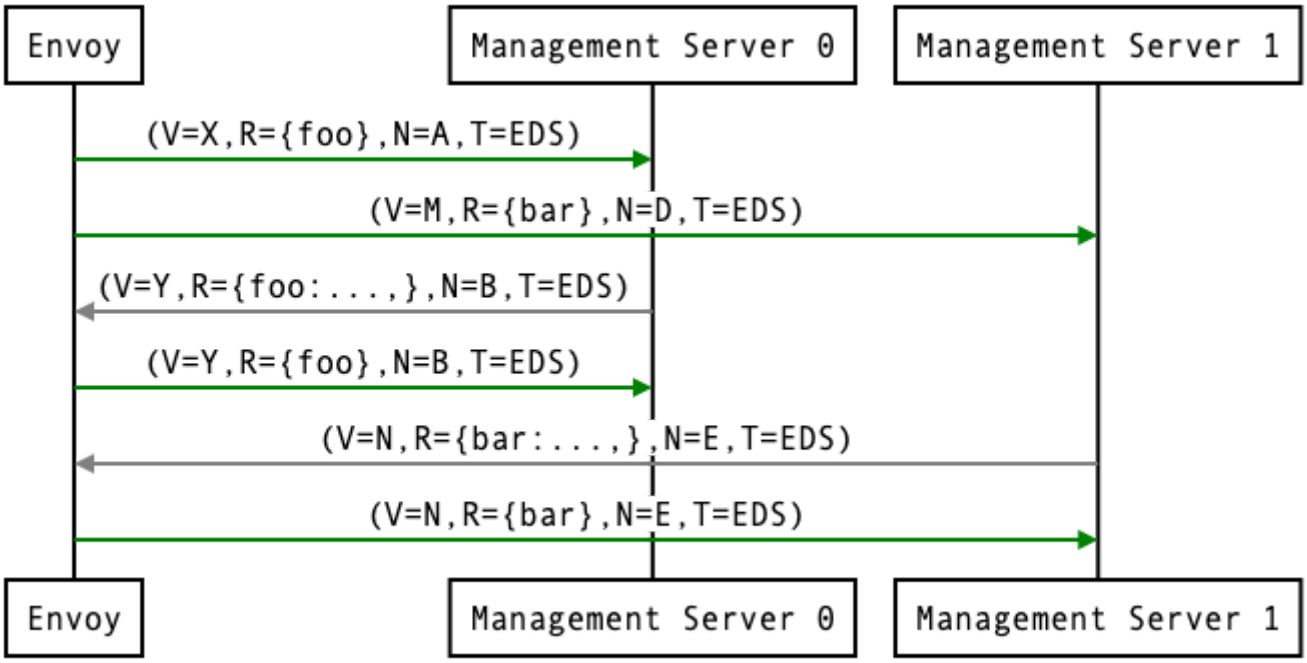
在SotW协议变体中，每个请求必须在resource_names字段中包含被订阅的资源名称的完整列表，因此取消对一组资源的订阅是通过发送一个包含所有仍被订阅的资源名称但不包含被取消订阅的资源名称的新请求完成的。例如，如果客户端之前订阅了资源A和B，但希望取消对B的订阅，它必须发送一个只包含资源A的新请求。

请注意，对于Listener和Cluster资源类型，如果客户端使用 "* "订阅（详见客户端如何指定返回哪些资源），被订阅的资源集由服务器而不是客户端决定，因此客户端不能单独取消订阅这些资源；它只能从通配符中取消订阅，这是整体。

在一个流中请求多个资源

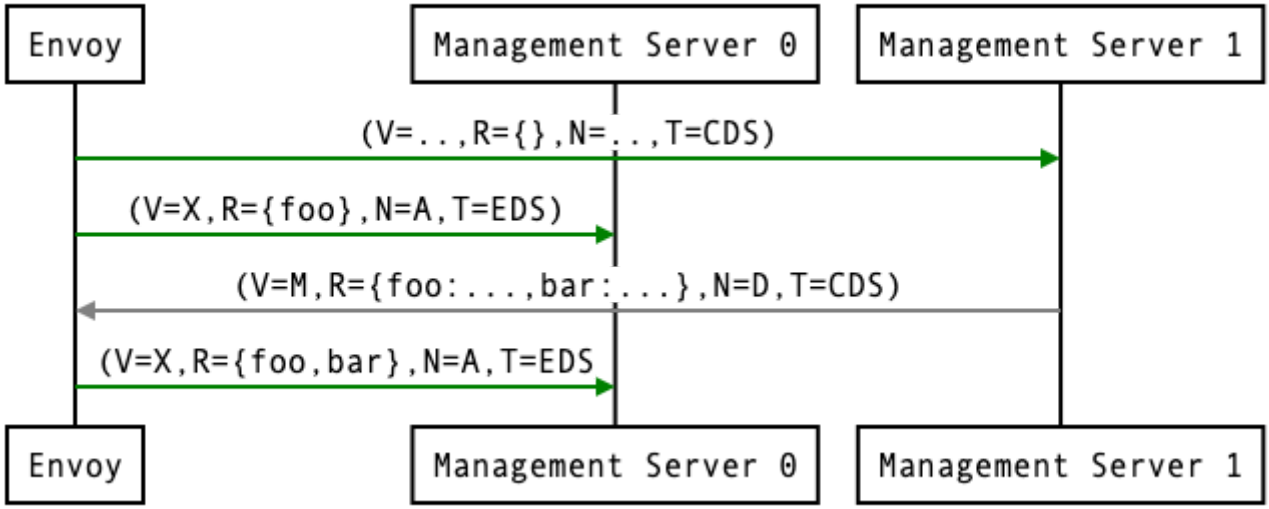
对于EDS/RDS，Envoy可以为每个给定类型的资源生成一个不同的流（例如，如果每个ConfigSource对管理服务器来说有自己不同的上游集群），或者当一个给定资源类型的多个资源请求是针对同一个管理服务器时，可以将它们合并在一起。虽然这有待于具体实施，但管理服务器应该能够在每个请求中处理一个或多个给定资源类型的 resource_names。下面两个序列图对获取两个EDS资源`{foo, bar}`有效：



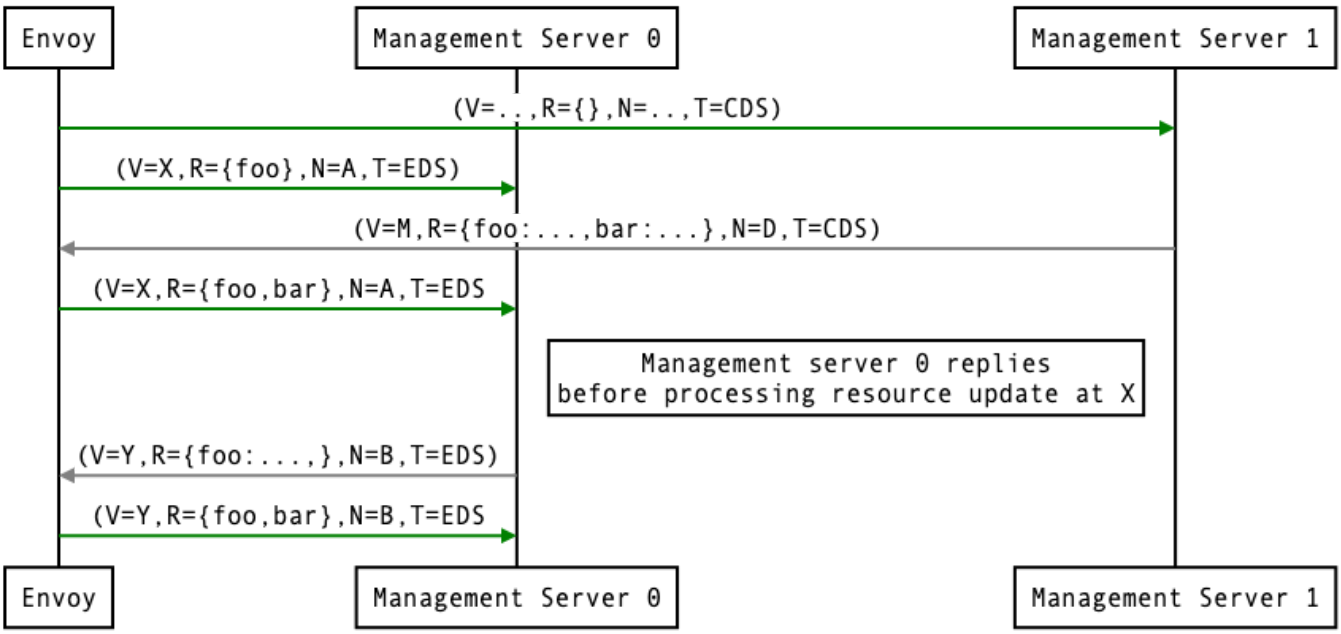


资源更新

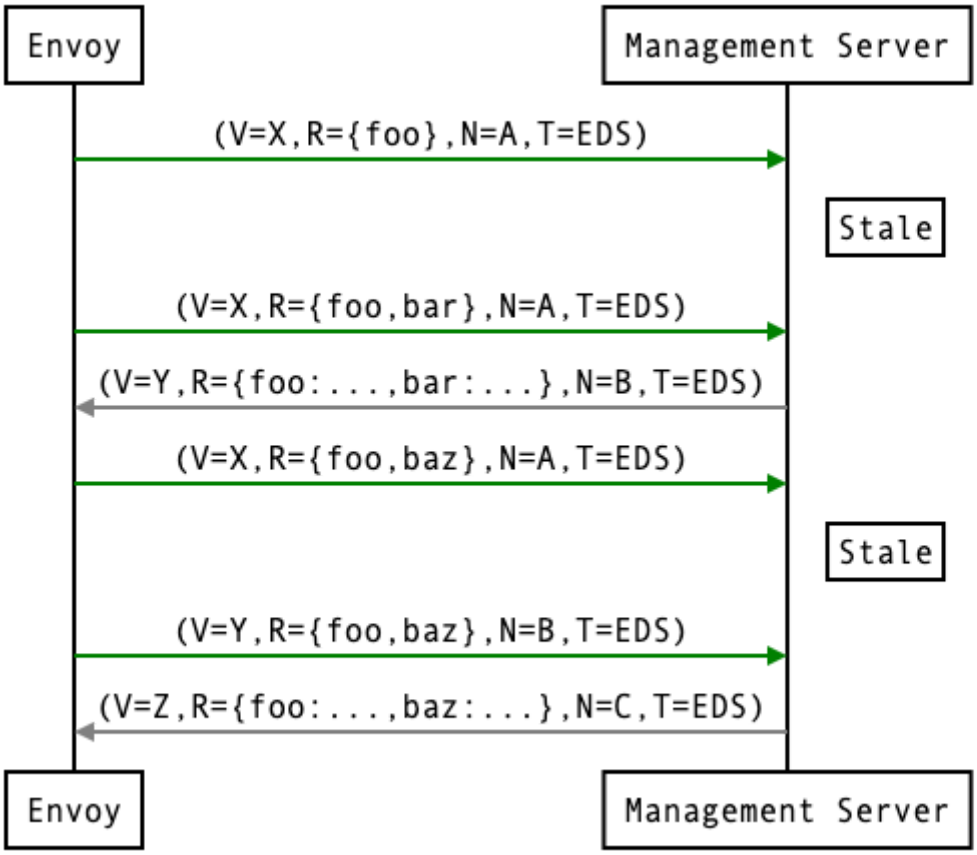
如上所述，Envoy可能会更新它在每个ACK/NACK特定DiscoveryResponse的DiscoveryRequest中提交给管理服务器的资源名称列表。此外，Envoy以后可能会在给定的version_info处发出额外的DiscoveryRequests，以便用新的资源提示更新管理服务器。例如，如果Envoy处于EDS的**X**版本，并且只知道集群`foo`，但是后来收到CDS的更新，并且知道了`bar`，它可以为**X**发出一个额外的DiscoveryRequest，将`{foo,bar}`作为`resource_names'。



这里可能会出现一个竞条件；如果Envoy在**X**发出资源提示更新后，但在管理服务器处理更新之前，它回复了一个新的版本**Y**，那么资源提示更新可能会被解释为拒绝**Y**，提出一个**X**的version_info。为了避免这种情况，管理服务器提供了一个 "nonce"，Envoy用它来指示每个DiscoveryRequest所对应的特定DiscoveryResponse:



管理服务器不应该为任何具有过期nonce的DiscoveryRequest发送DiscoveryResponse。在DiscoveryResponse中向Envoy提交了一个较新的nonce后，nonce就会变得陈旧。管理服务器不需要发送更新，直到它确定有一个新的版本可用。早期的版本请求也会变得陈旧。它可以在一个版本上处理多个DiscoveryRequests，直到一个新的版本准备好。



上述资源更新顺序的一个含义是，Envoy并不期望它发出的每一个DiscoveryRequests都有一个DiscoveryResponse。

资源预热

集群和监听器在为请求提供服务之前要经过预热。这个过程发生在Envoy initialization期间和 `Listener` 或 `Cluster` 被更新时。只有当管理服务器提供 `ClusterLoadAssignment` 响应时， `Cluster` 的预热才会完成。同样，只有当管理服务器提供 `RouteConfiguration` 时，如果监听器指向RDS配置， `Listener` 的预热才会完成。管理服务器应该在预热期间提供EDS/RDS更新。如果管理服务器不提供EDS/RDS响应，Envoy将不会在初始化阶段初始化自己，通过CDS/LDS发送的更新将不会生效，直到EDS/RDS响应被提供。

最终的一致性考虑

由于Envoy的xDS APIs最终是一致的，所以在更新期间流量可能会短暂下降。例如，如果只有集群**X**是通过CDS/EDS知道的， `RouteConfiguration` 引用了集群**X**，然后在CDS/EDS更新提供**Y**之前被调整为集群**Y**，流量将被封锁，直到**Y**被Envoy实例所知道。

对于某些应用来说，暂时性的流量下降是可以接受的，客户端或其他Envoy侧设备的重试将隐藏这种下降。对于其他不能容忍掉线的情况，可以通过提供CDS/EDS更新，同时提供**X**和**Y**，然后RDS更新从**X**重新指向**Y**，然后CDS/EDS更新放弃**X**来避免流量掉线。

一般来说，为了避免流量下降，更新的顺序应该遵循先做后断的模式，其中。

- CDS更新（如果有的话）必须总是先推送。
- EDS更新（如果有的话）必须在各集群的CDS更新之后到达。
- LDS的更新必须在相应的CDS/EDS更新之后到达。
- 与新添加的`Listener`相关的RDS更新必须在CDS/EDS/LDS更新之后到达。
- 与新添加的RouteConfigurations相关的VHDS更新（如果有的话）必须在RDS更新后到达。
- 陈旧的CDS集群和相关的EDS端点（不再被引用的）就可以被删除。

如果没有添加新的`Cluster`/Router/`Listener`，或者在更新期间暂时放弃流量是可以接受的，xDS更新可以独立推送。请注意，在LDS更新的情况下，听众在接收流量之前会被预热，也就是说，如果配置了RDS，依赖路由会被获取。在添加/删除/更新集群的时候，集群会被预热。另一方面，路由不被预热，也就是说，在推送路由的更新之前，管理平面必须确保路由所引用的集群已经到位。

TTL

如果管理服务器无法到达，Envoy收到的最后一个已知配置将持续到连接重新建立。对于某些服务，这可能是不可取的。例如，在故障注入服务的情况下，管理服务器在错误的时间崩溃可能会使Envoy处于一个不理想的状态。TTL设置允许Envoy在与管理服务器失去联系后，在指定的时间段内删除一组资源。例如，这可以用来在管理服务器不能再被联系到时终止故障注入测试。

对于支持*xds.config.supported-resource-ttl*客户端功能的客户端，可以在每个资源上指定一个TTL字段。每个资源将有自己的TTL过期时间，在这个时候，资源将过期。每个xDS类型可能有不同的方式来处理这种过期。

要更新与*资源*相关的TTL，管理服务器用新的TTL重新发送该资源。要删除TTL，管理服务器重新发送资源，但不设置TTL字段。

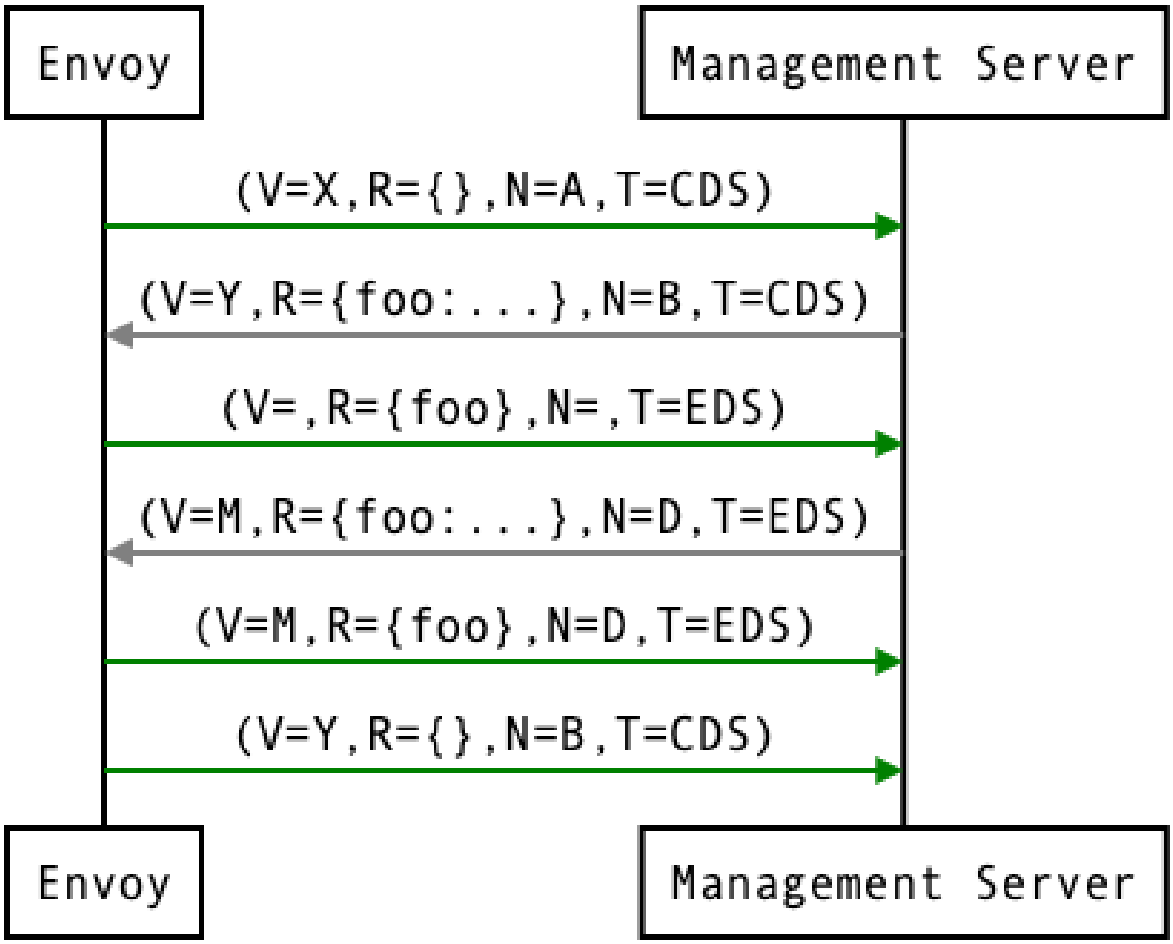
为了允许轻量级的TTL更新（"心跳"），可以发送一个响应，提供一个资源，资源未设置，与最近发送的版本相匹配的版本可以用来更新TTL。这些资源将不会被视为资源更新，而只是作为TTL更新。

为了在SotW xDS中使用TTL，相关的资源必须被包裹在一个资源中。这允许设置与SotW的Delta xDS相同的TTL字段，而无需改变SotW的API。SotW也支持心跳：响应中任何看起来像心跳资源的资源将只用于更新TTL。

这个功能由*xds.config.supported-resource-in-sotw*客户端功能来控制。

聚合发现服务

在管理服务器分布的情况下，要提供上述的顺序保证以避免流量下降是很有挑战性的。ADS允许单个管理服务器通过单个gRPC流来提供所有的API更新。这提供了对更新进行仔细排序以避免流量下降的能力。有了ADS，一个单一的流被用于多个独立的DiscoveryRequest/DiscoveryResponse序列，通过类型URL复用。对于任何给定的类型URL，上述DiscoveryRequest和DiscoveryResponse消息的顺序适用。一个更新序列的例子可能看起来像:



每个Envoy实例可以使用一个ADS流。

一个用于配置ADS的最小`bootstrap.yaml`片段的例子是：

YAML

```yaml
node:
  # set <cluster identifier>
  cluster: envoy_cluster
  # set <node identifier>
  id: envoy_node

dynamic_resources:
  ads_config:
    api_type: GRPC
    transport_api_version: V3
    grpc_services:
    - envoy_grpc:
        cluster_name: ads_cluster
  cds_config:
    resource_api_version: V3
    ads: {}
```

```
  lds_config:
    resource_api_version: V3
    ads: {}

static_resources:
  clusters:
  - name: ads_cluster
    type: STRICT_DNS
    load_assignment:
      cluster_name: ads_cluster
      endpoints:
      - lb_endpoints:
        - endpoint:
            address:
              socket_address:
                # set <ADS management server address>
                address: my-control-plane
                # set <ADS management server port>
                port_value: 777
    # It is recommended to configure either HTTP/2 or TCP keepalives in order to detect
    # connection issues, and allow Envoy to reconnect. TCP keepalive is less expensive, but
    # may be inadequate if there is a TCP proxy between Envoy and the management server.
    # HTTP/2 keepalive is slightly more expensive, but may detect issues through more types
    # of intermediate proxies.
    typed_extension_protocol_options:
      envoy.extensions.upstreams.http.v3.HttpProtocolOptions:
        "@type":
 type.googleapis.com/envoy.extensions.upstreams.http.v3.HttpProtocolOptions
        explicit_http_config:
          http2_protocol_options:
            connection_keepalive:
              interval: 30s
              timeout: 5s
    upstream_connection_options:
      tcp_keepalive: {}
```

增量 xDS

增量xDS是一个单独的xDS端点，它：

- 允许协议以资源/资源名称deltas（"Delta xDS"）的方式进行线上通信。这支持xDS资源的可扩展性目标。当一个集群被修改时，管理服务器不需要交付所有的100个集群，而只需要交付改变了

的单一集群。

- 允许Envoy按需/懒惰地请求额外的资源。例如，只有在对集群的请求到达时才请求该集群。

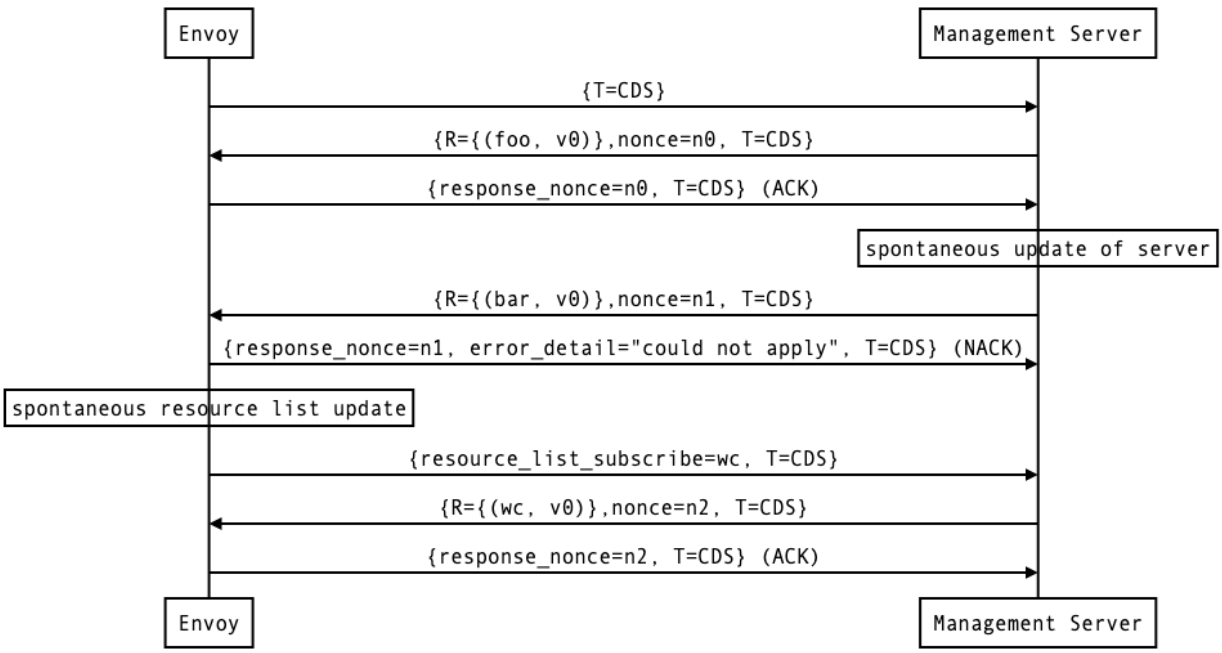
增量xDS会话总是在gRPC双向流的背景下进行。这使得xDS服务器可以跟踪连接到它的xDS客户端的状态。目前还没有增量式xDS的REST版本。

在delta xDS线协议中，nonce字段是必需的，用于将DeltaDiscoveryResponse与DeltaDiscoveryRequest ACK或NACK配对。可选的是，一个响应消息级别的system_version_info只为调试目的出现。

- xDS双向gRPC流中的初始消息。
- 作为对之前DeltaDiscoveryResponse的ACK或NACK响应。在这种情况下，response_nonce被设置为响应中的nonce值。ACK或NACK由error_detail的缺失或存在来决定。
- 来自客户端的自发DeltaDiscoveryRequests。这样做可以从跟踪的resource_names集合中动态地添加或删除元素。在这种情况下，response_nonce必须被省略。

请注意，虽然response_nonce可以在请求中设置，但即使nonce是过时的，服务器也必须尊重对订阅状态的更改。nonce可用于将ack/nack与服务器响应联系起来，但不应*用于拒绝过时的请求。

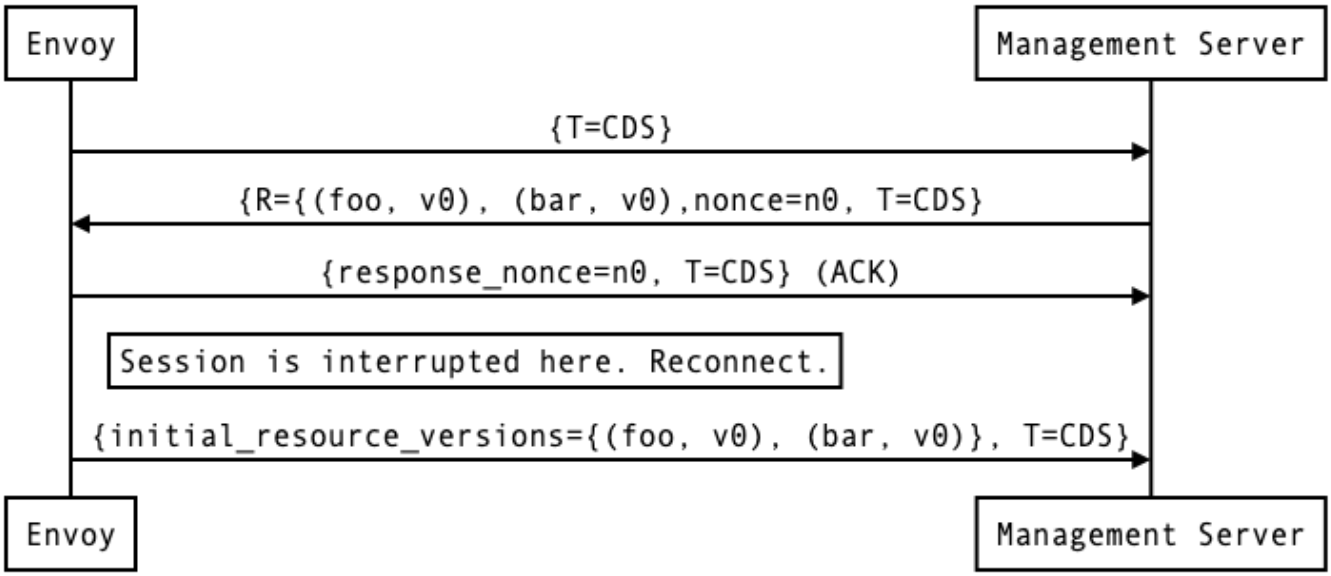
在第一个例子中，客户端连接并收到了第一个更新，它ACK了。第二个更新失败了，客户端NACK了这个更新。后来xDS客户端自发地请求 "wc "资源。



在重新连接时，增量xDS客户端可以告诉服务器它的已知资源，以避免通过在initial_resource_versions中重新发送它们。因为没有假设从以前的流中保留状态，重新连接的客户端必须向服务器提供它感兴趣的所有资源名称。

请注意，对于 "通配符 "订阅（详见客户端如何指定返回哪些资源），请求必须在resource_names_subscribe字段中指定 "*"，或者（传统行为）请求必须在

resource_names_subscribe和resource_names_unsubscribe 中没有资源。



资源名称

资源是由资源名称或别名来识别的。如果存在资源的别名，可以通过DeltaDiscoveryResponse的资源中的别名字段来识别。资源名称将在DeltaDiscoveryResponse的资源中的名称字段中返回。

订阅资源

客户端可以在DeltaDiscoveryRequest的resource_names_subscribe字段中发送一个别名或资源的名称，以便订阅一个资源。资源的名称和别名都应该被检查，以确定有关实体是否被订阅了。

resource_names_subscribe字段可能包含服务器认为客户已经订阅的资源名称，而且还拥有最新的版本。然而，服务器*必须*在响应中提供这些资源；由于对服务器隐藏的实施细节，客户可能已经 "忘记 "了这些资源，尽管表面上仍然订阅了。

退订资源

当客户端对某些资源失去兴趣时，它将通过DeltaDiscoveryRequest的[resource_names_unsubscribe]字段来表示。与resource_names_subscribe一样，这些可以是资源名称或别名。

resource_names_unsubscribe字段可能包含多余的资源名称，服务器认为客户端已经没有订阅。服务器必须干净利落地处理这样的请求；它可以简单地忽略这些幽灵式的取消订阅。

了解所请求的资源不存在的情况

当客户端订阅的资源不存在时，服务器将发送一个资源，其名称字段与客户端订阅的名称相匹配，其资源字段未设置。这允许客户端快速确定资源是否存在，而无需像SotW协议的变体那样等待超时。然而，我们仍然鼓励客户使用超时，以防止管理服务器未能及时发送响应的情况。

REST-JSON轮询订阅

通过REST端点的同步（长）轮询也可用于xDS单体API。上述消息的排序是类似的，只是没有向管理服务器保持持久的流。预计在任何时间点都只有一个未完成的请求，因此在REST-JSON中，响应nonce是可选的。proto3的JSON典型转换被用来编码DiscoveryRequest和DiscoveryResponse消息。ADS不适用于REST-JSON轮询。

当轮询周期被设置为一个较小的值，并打算进行长时间的轮询时，还需要避免发送DiscoveryResponse，除非通过资源更新对基础资源进行了改变。