ProtoBuf optimization and efficiency schema

Tech Con 2025 Abstract 140

HPE Edge;

This abstract did not conform to Tech Con 2025 submission format and may have been redacted.

ProtoBuf optimization and efficiency schema

[redacted]

Abstract

Nowadays, for Aruba Cloud architecture, no matter CNX or classic central, it will meet Proto mismatch issue for "service and service" or "device with cloud". In this paper, we raised a new structure to unify Proto in our cloud architecture to let each service dynamically aware the Proto change and use a strategy to dynamically prune the unchanged Proto or only increment the changed Proto to alleviate the transmitting overload, and this architecture also can benefit for the overload between device and cloud.

Problem statement

Nowadays, for Aruba Cloud architecture, no matter CNX or classic central, it will meet Proto mismatch issue for "service and service" or "device with cloud".

For service and service scenario, if one service interest other service's Proto (Kafka payload), this service need to use its interest service proto, and if interests proto add some new field, it has to upgrade the lib (even upgrade proto library not necessary, but for some service, its necessary, like Kpow), if some service proto library add some new field, maybe cannot notify every team member in a huge team, then will meet mismatch issue. Like Kpow, every service owner needs to submit the code to Kpow repo then Kpow can parse the Proto or Monitor UXI show All related stats (Use Kpow and UXI as example it's because they only need upgrade to latest Proto version without change any logic code).

Device and cloud side also will meet Proto mismatch issue, since now Devie and cloud side use different repo to store the Proto, like ap side use AOS repo and cloud side use ACP-device-proto

For now, **for each Proto we can use optional (ex: optional string VLAN)** to indicate which field was optional, **or GRPC (since GRPC will treated all filed as option)** but **for non-optional field or Kafka cas**e, even though the field value is not changed, we still need to include this un-changed Proto field into Kafka payload.

For AP and cloud scenario, since almost all device side (AP/CX/GW) Proto was non optional, which means each device will periodically (1 mins or 5 mins) report telemetry message to cloud side even some Proto payload is not changed.

So now Proto is used everywhere in our cloud architecture, we do not have a unified approach to management Proto for each scenario. We need a method to unify the Proto and notification when Proto changed, then dynamic do prune or increment the changed Proto field to alleviate the overload of transmitting (like Kafka), then let whole cloud infrastructure more efficient.

Our solution

we raised a new structure to unify Proto in our cloud architecture to let each service dynamically aware the Proto change and use a strategy to dynamically prune the unchanged Proto or only increment the changed Proto to alleviate the transmitting overload, and this architecture also can benefit for the overload between device and cloud.

We define a unify centralized architecture (UCA) to provide registration and subscription function (UCA can be a service): All Proto register to unify architecture and each service to subscription its own interests Proto.

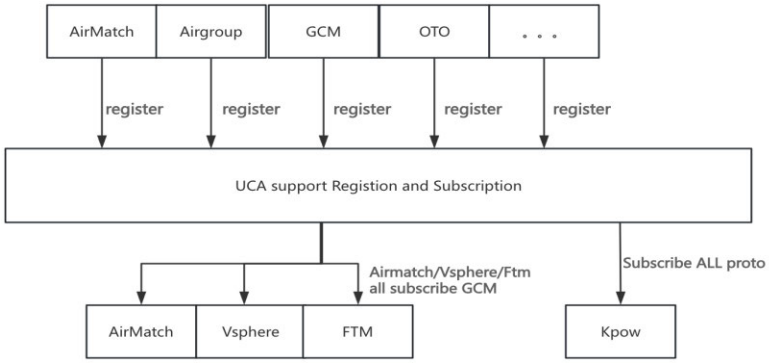
Dynamic notification strategy, which means in each service running stage, if UCA detect one of the register Proto changed than before (assume we only keep latest version), then it will broadcast this changed Proto notification to each subscription service, then each subscription service will dynamically load this Proto change. Kpow or UXI can use this unique architecture to Parse each Proto (subscribe all interest Proto, and only need upgrade to latest Proto version without change any logic code)

Dynamic prune un-changed Proto field or only increment changed Proto field to reduce transmitting (like Kafka) overload: for each transmission, the producer service can prune the un-changed proto field and send to Kafka, then notify UCA, UCA will notify each subscription service and push the prune Proto format to each service accordingly, then the service can parse the prune Proto correctly (each subscription service also need do small change to support this architecture, like dynamic get the new proto structure from unify architecture, but for Kpow or UXI no need changed).

Bi-direction subscription method to reduce the overload between cloud and device, which means cloud side will subscribe device side Proto change (not every device, it can use a dynamic way to subscription device, like some device has huge traffic, how to dynamic subscribe device out of this patent scope), Device side can dynamic prune the un-change proto field, then the cloud side will know (base on subscription), by using this way, it can reduce the overload between device and cloud (since for device side Proto, like state, most of filed was un-changed), same behavior with cloud side.

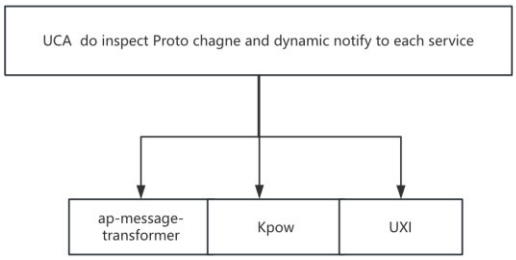
1. Unify centralized architecture (UCA)

The UCA will be a centralized Service which will include register/subscription/notification Proto change and partial Proto prune result. From Figure 1, Each service will register their Proto to the UCA and will subscribe its interests Proto, for example, AIRMATCH/Airgroup/GCM/OTO all register their own specific Proto to UCA , and AIRMATCH/VSPHERE/FTM all interest the GCM Proto (GCM was complicate, may include multiple Proto, Figure 1 just an example, we can also subscribe one of the Proto belong to one Service) and subscribe it, Kpow will subscribe all Proto for debug purpose.



2. Dynamic notification strategy

Based on #1 description, then in each service running stage (dynamic upgrade library without service reload or bootstrap), if UCA detect one of the register Proto changed (version different with before), then it will broadcast this changed Proto notification to subscription service, then each subscription service will dynamically load this Proto's changing, see below picture, KPow/UXI/ap-message-transformer can immediately upgrade the Proto from UCA no need do any logic change, this structure can dramatically benefit to such category service.



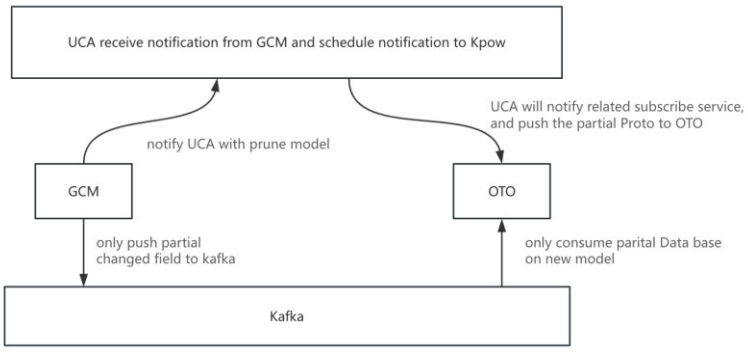
3. Partial Proto with Prune or increment to reduce transmitting overload

Each service can dynamically Prune the un-changed Proto filed or increment the new changed Proto and notify UCA, then UCA will do centralized schedule.

See Figure 3, if GCM wants to send partial (only changed filed) to related service (based on subscription), GCM can do prune and only push partial changed Proto to Kafka, also send a notification to UCA with the new Proto model, then UCA will push this new Proto model (only model, the actually data still comes from Kafka) subscribe service, then each service like OTO and consumer this new Proto, by using this method, it can reduce the Kafka

overload.

4. Bi-direction subscription to reduce overload between Device and Cloud



Bi-direction subscription means device and cloud can subscribe each other:

Cloud subscribe Device (like AP) Proto: which means cloud side will subscribe device side Proto change (not every device, it can use a dynamic way to subscription device, like some device has huge traffic), AP side can dynamic prune the un-change proto field, then cloud side will know (base on subscription), by using this way, it can reduce the overload between device and cloud (since for device side Proto, like state, most of filed was un-changed)

AP subscribe cloud side Proto: AP subscribe and apply interests proto without service interruption, AP can send messages to cloud to subscribe its interests proto, when proto definition is changed, cloud send new proto to AP, device's running process can apply the new definition without building new image or rebooting process, there is no service interruption during this.

AP send changed data incrementally to save bandwidth , Currently, when AP sends telemetry to cloud, it must send all proto fields to cloud even if most part of fields are not changed because some telemetry consumers in cloud hope each telemetry message always has full data. The requirements waste a large amount of uplink bandwidth. After introducing the new architecture in cloud, AP can send changed data incrementally and the new architecture fill missing fields using history data and then publish the message with full fields to consumers.

Evidence the solution works

For now, we are testing in Mesh service, Mesh service will first read Kafka message from ap-message-transformer, now this library does not support python library, so we have to do local compile and load the library, the other was Mesh service between Mesh service also have we own define proto, also Mesh service need listen Kafka topic from GCD, due to all this dependency, we first let the Proto can auto upgrade between Mesh service.

Competitive approaches

ProtoBuf unify should be a high topic to make the system easy to use, and benefit to all server developers. Our proposal will raise a new structure to let Proto change auto notify to each user (service) and release the service developer work, also use dynamic prune and increment to release the Kafka overload.

Current status and Next steps

This proposal was in the initial stage, as mentioned before, we will first make change between Mesh service to auto handle the Proto library change.