Experiences of Implementing ALTO in OpenDaylight

draft-zhang-alto-opendaylight-impl-00

J. Zhang   K. Gao   Y. R. Yang

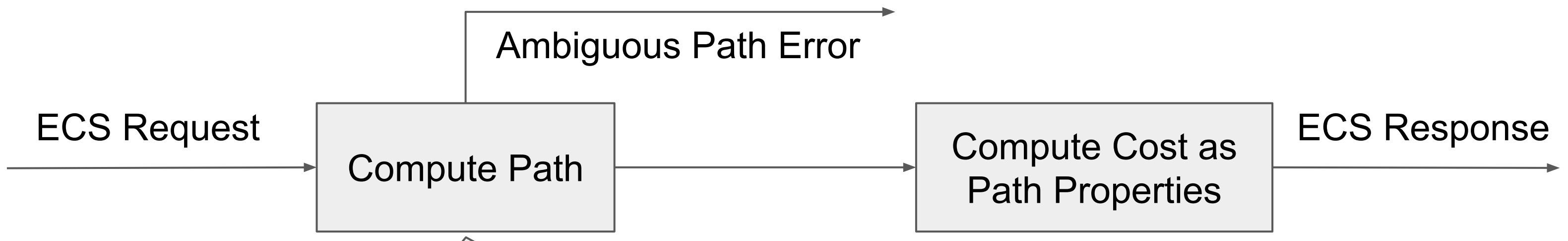
Presenter: Jensen Zhang

October 27, 2015 @ ALTO Interim Meeting

Outline

- Design and Implementation of Endpoint Cost Service (ECS)
- Design of Auto-Map
- YANG Model Issues
- Extensible and Portable Architecture

Implementing ECS: Workflow and Challenges

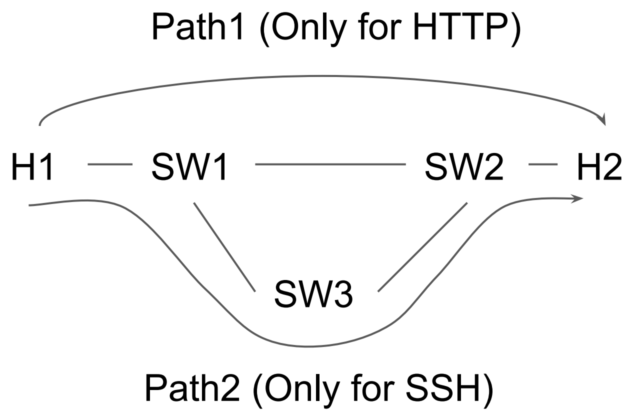


<u>Challenges:</u>

- ODL allows fine-grained paths: multiple paths for the same src-dst pair, distinguished by flow attributes beyond src/dst addresses
- Multiple modules in ODL may get involved in path computation

Path Computing: Fine-Grained Routing

Example:

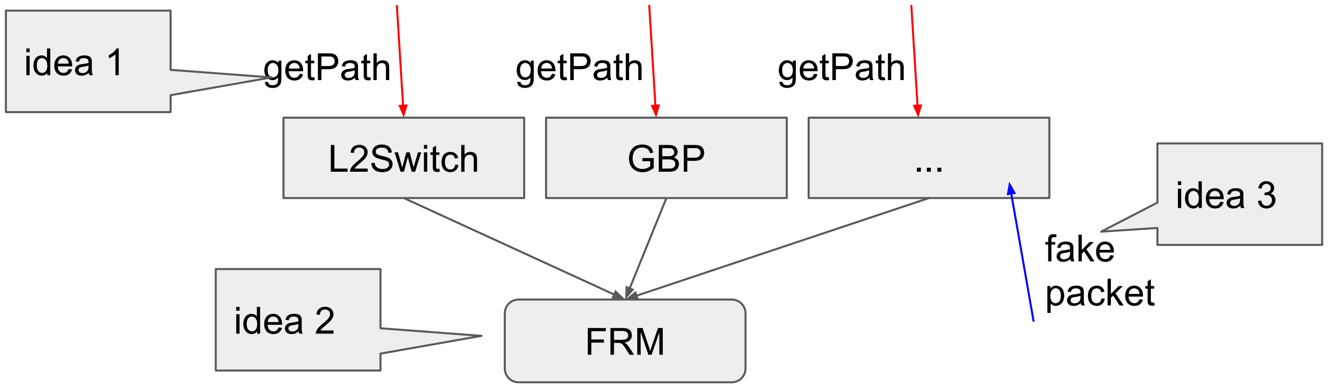


Potential Solution:

- Return E_AMBUGUOUS_RESULT
- Inform users how to refine their request. (Need draft-wang-alto-ecs-flows.)

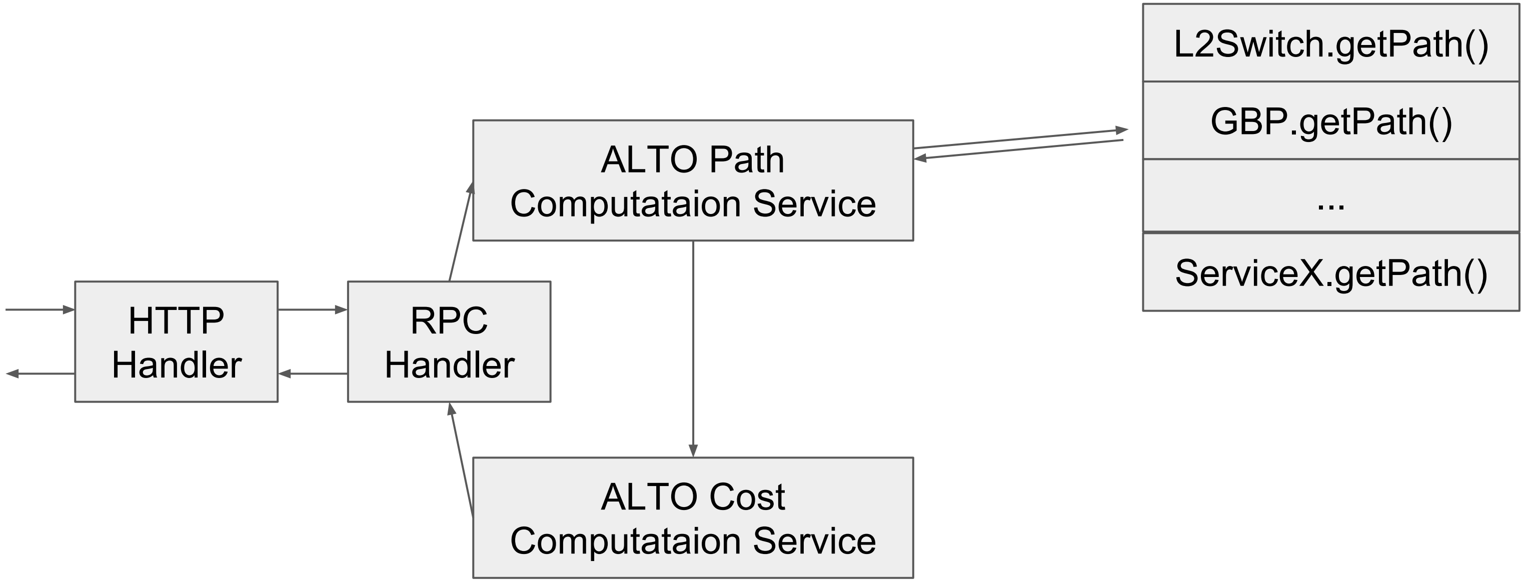
Current Design: Compute Path

- Potential design: using Flow Rule Manager (FRM)
  - FRM provides a unifying data structure to store paths across modules
  - Issue: Some modules may adopt a reactive routing approach (i.e., insert path only upon packet-in) when inserting into FRM
- Our design
  - Idea 1: Introduce a new path computation (PC) interface that routing modules can implement
  - Idea 2: For those modules that have not implemented the PC interface, look up in FRM
  - Idea 3: For those that use reactive routing and no PC interface, use fake packet



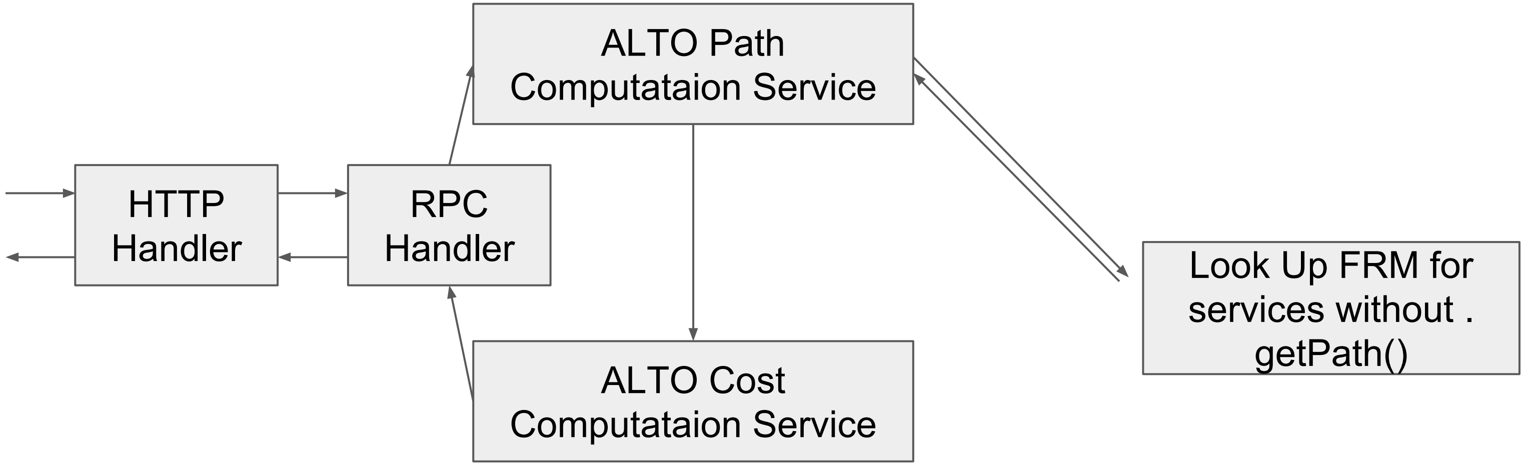
Workflow of Path Computaion

Idea 1: some modules may not implement the PC interface



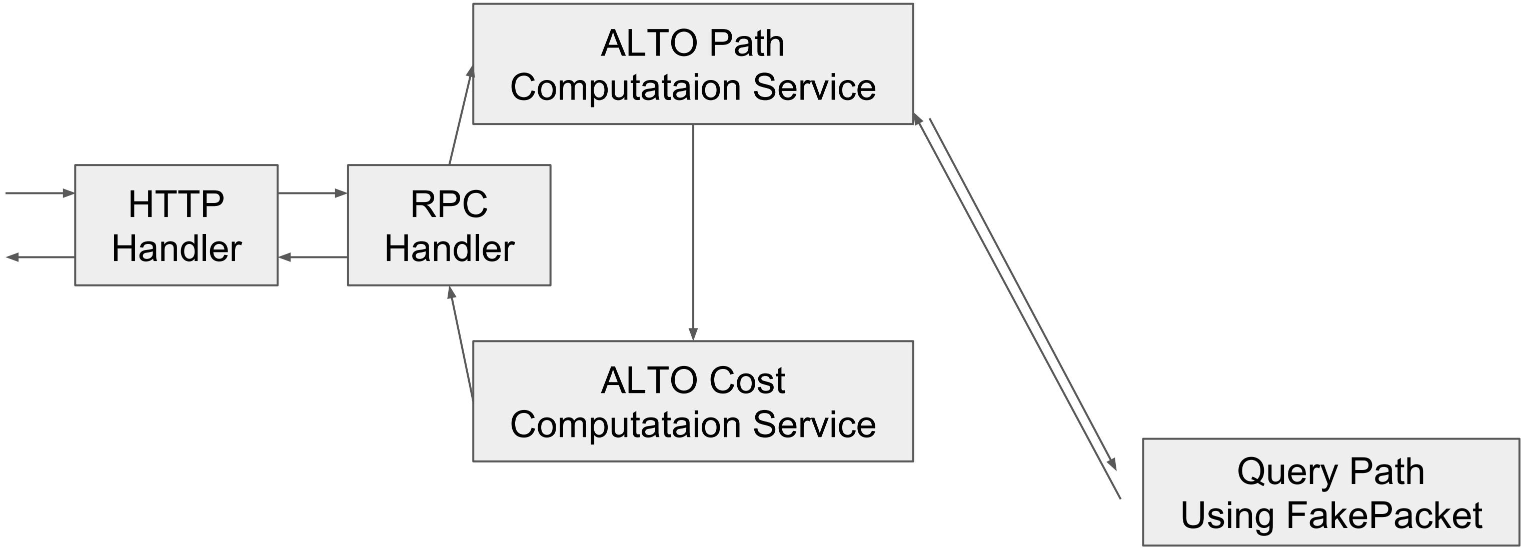
Workflow of Path Computaion

Idea 2: reactive routing approach may not be detected in FRM


ALTO Path Computataion Service
HTTP Handler
RPC Handler
ALTO Cost Computataion Service
Look Up FRM for services without . getPath()

Workflow of Path Computaion

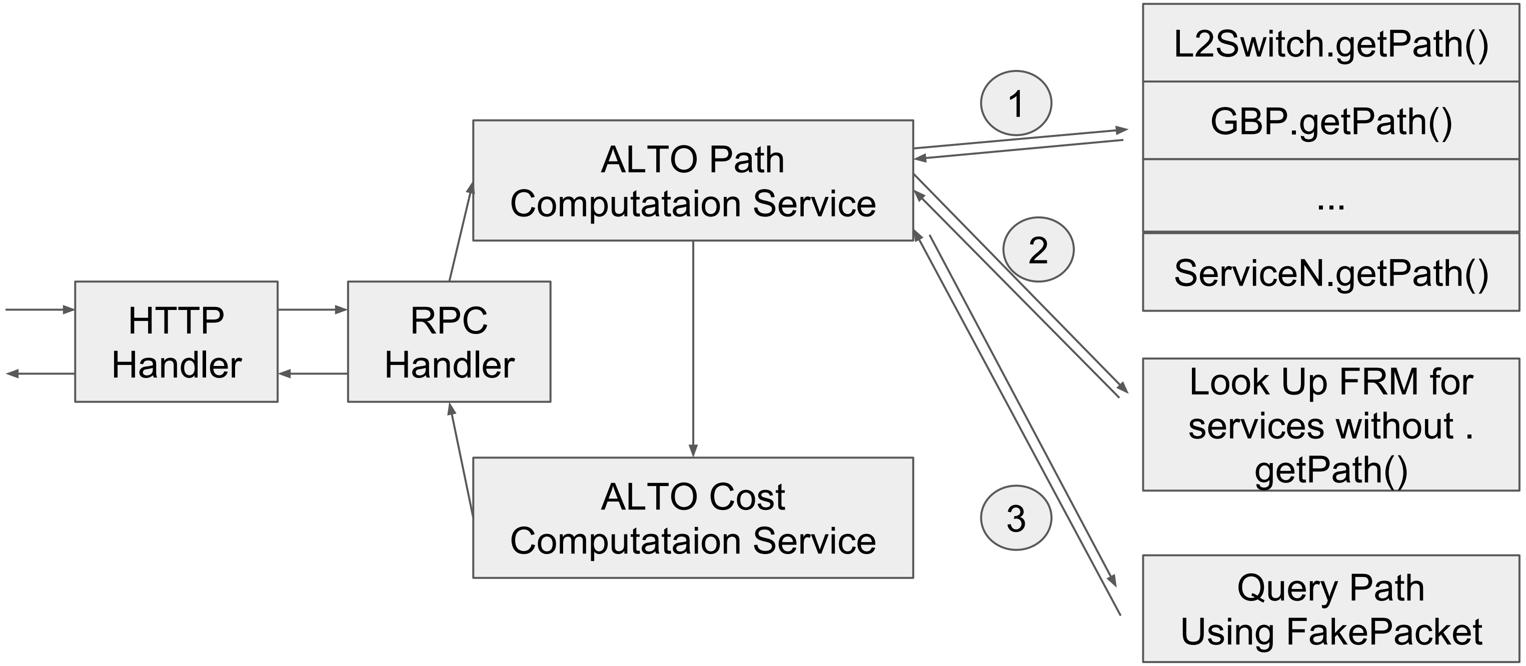
Idea 3: low performance



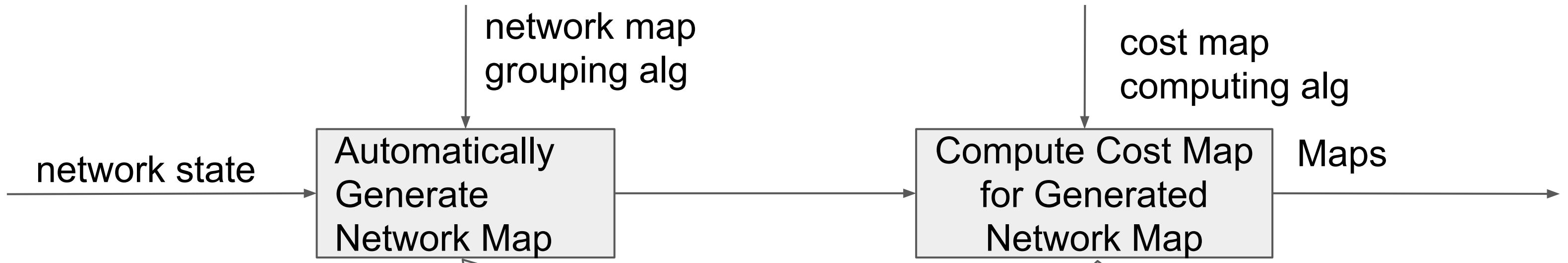
ALTO Path Computataion Service
HTTP Handler
RPC Handler
ALTO Cost Computataion Service
Query Path Using FakePacket

Workflow of Path Computaion

Combine above ideas


HTTP Handler
RPC Handler
ALTO Path Computataion Service
ALTO Cost Computataion Service
1
2
3
L2Switch.getPath()
GBP.getPath()
...
ServiceN.getPath()
Look Up FRM for services without . getPath()
Query Path Using FakePacket

Implementing AutoMap: Workflow and Challenges


network map
grouping alg
cost map
computing alg
network state
Automatically Generate Network Map
Compute Cost Map for Generated Network Map
Maps

<u>Challenge:</u>

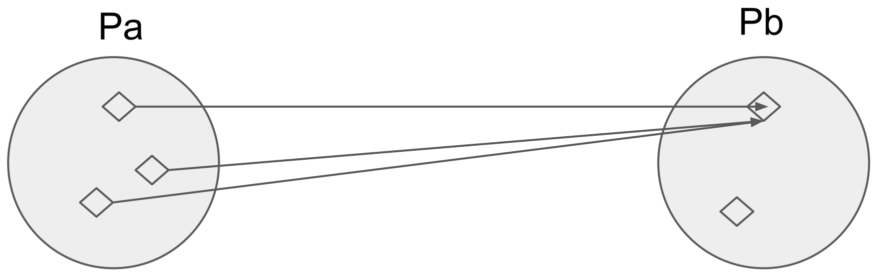
- Provide easy-to-use, yet complete specification and algorithms to allow admin to define grouping of network nodes

<u>Challenges:</u>

- Provide a generic method to define the cost computation between two PIDs.

Current Design (Implementation-in-progress)

- Decouple network-map generation and cost-map computation
  - Define grouping: Provide one automatic network-map generation algorithm: **nearest-neighbor**
  - Compute inter-PID cost from inter-endpoint costs: Given PIDS Pa and Pb, there will be |Pa| x |Pb| inter-endpoint costs. We provide multiple definitions (**median, x-percentile, avg**) as the cost from Pa to Pb, and allow multiple algorithms to do the computation (**total enumeration, random sampling**)



Current Design: Compute Inter-PID Costs

cost-map-config.json

```
{
    "cost-map-id": "cmap1",
    "uses": [ "my-nn-auto-network-map" ],

    "cost-type": {
        "cost-mode": "numerical",
        "cost-metric": "hopcount"
    },
    "cost-map-group-metric": "avg",
    "cost-map-group-alg": {
        "alg": "random-sampling",
        "count" : 10000
    }
}
```

Current Design: Nearest Anchor for Network Map Grouping

`nearest-network-map-config.json`

```
{
    "net-map-id": "nearest-network-map",
    "net-map-grp-alg": "nearest-alg",
    "net-map-grp-para": {
        "metric": "hopcount",
        "anchors": {
            "pid1": ["sw1", "sw2"],
            "pid2": ["sw3"],
            "pid3": ["sw4", "sw5"]
        }
    }
}
```

`Add a new anchor:`

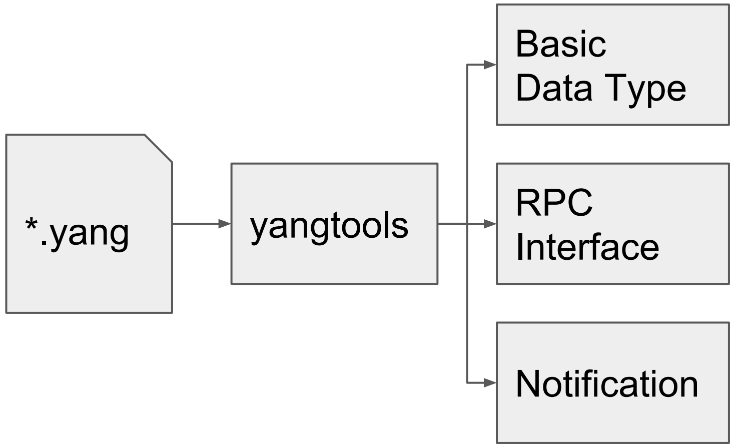
```
> alto-create-pid nearest-network-map pid4

> alto-add-anchor nearest-network-map pid4 sw6

> alto-del-anchor nearest-network-map sw6
```

Implement ALTO using MD-SAL: Background

- ODL is model-driven
- Need to define YANG models for ALTO
- An earlier proposal is in draft-shi-alto-yang-model



Issue of Implementing ALTO using the YANG Model in draft-shi-alto-yang-model

JSON Type

```
object-map {
  TypedEndpointAddr -> JSONValue;
} EndpointDstCosts;

object-map {
  PIDName -> JSONValue;
} DstCosts;
```

YANG Model

```
grouping alto-cost {
  anyxml cost {
  mandatory true;
  description "ALTO cost is a JSONValue, which
  could be an object, array, string, etc. (Ref:
  RFC 7159 Sec.3.)";
  }
}
```

Issue: 'cost' could be differenttypes in different *CostMap*s and *EndpointCostMap*s.

Current Design

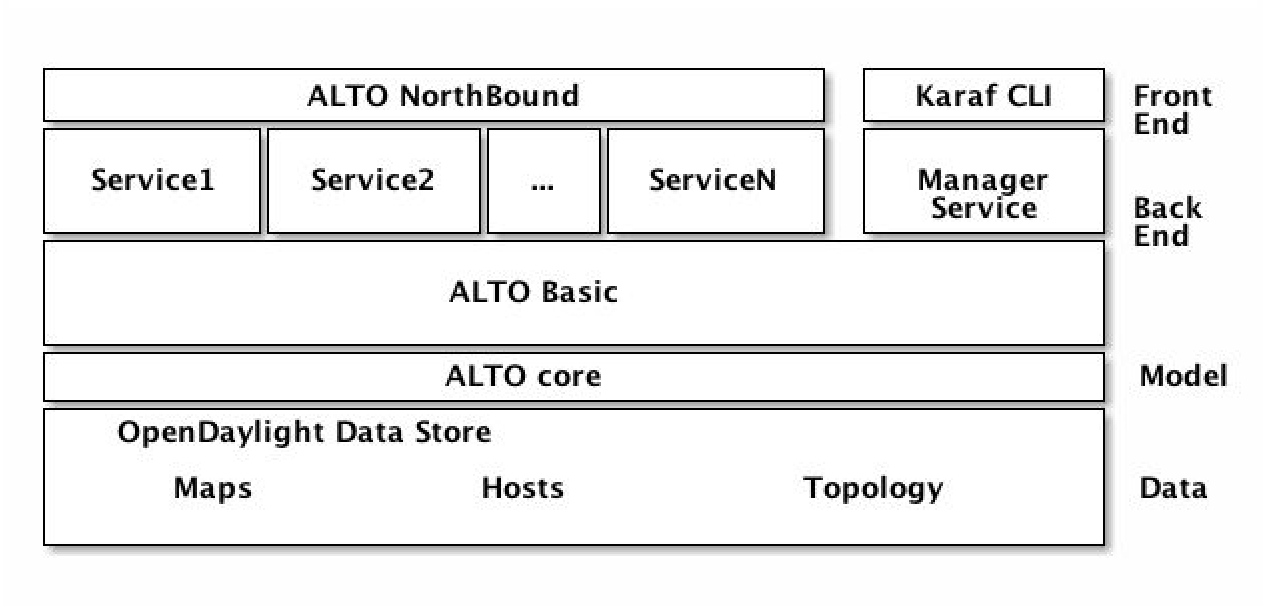
- ‘anyxml’ is not a good solution.
- **‘augment’ is a good one.**
    - extensibility
    - serialization

We can define one ‘cost’ as ‘int’, and another one as ‘decimal’. And it is easy to add more ‘cost’ value type.

```
module alto-cost-default {
  ...
  augment "<node1>" {
    leaf cost-default {
      type int;
    }
  }
  augment "<node2>" {
    leaf cost-default {
      type decimal;
    }
  }
  ...
}
```

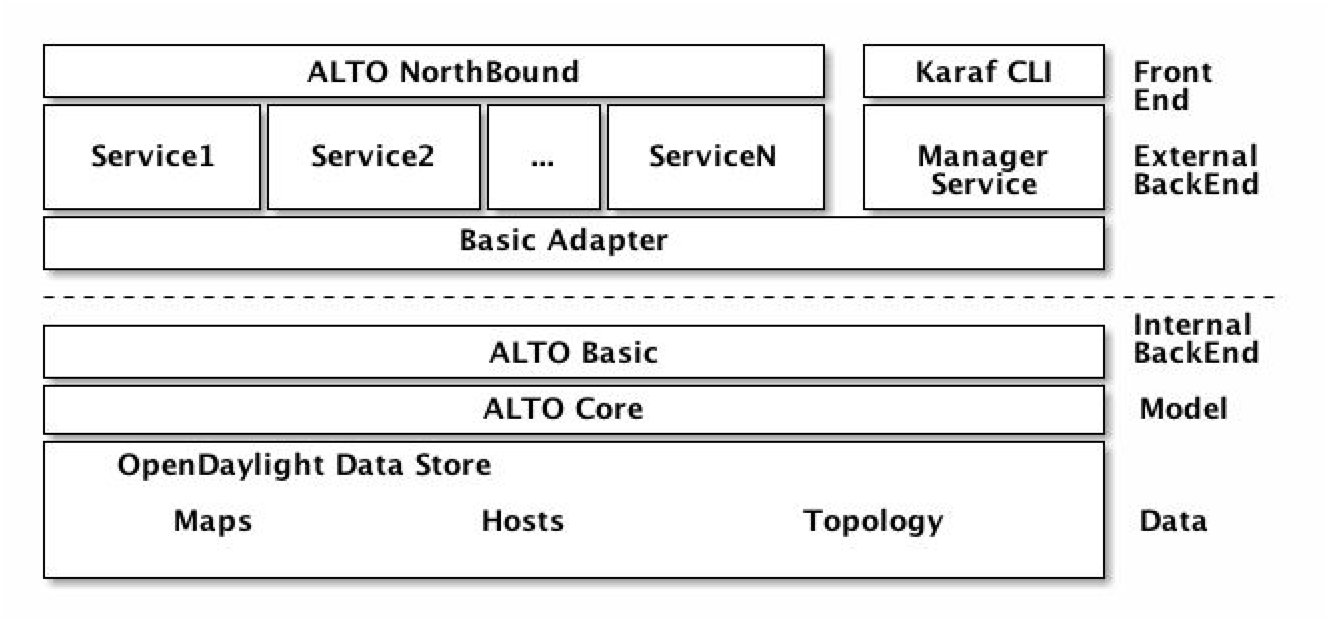
Design for Extensible ALTO Server

Seperate services into different modules.



Design for Cross Platform

Introduce an adapter layer to seperate services from ODL.



Thanks!