ALTO Extension: Path Vector

draft-yang-alto-path-vector-03
draft-gao-alto-routing-state-abstraction-03

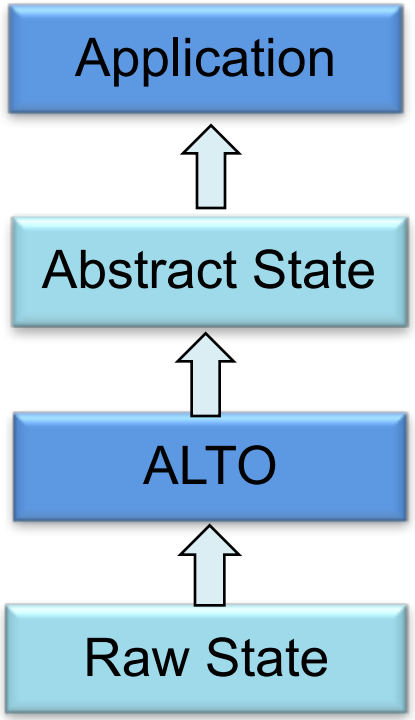
Presenter: Y. Richard Yang

July 21, 2016

IETF96

Overview

- Going beyond “single-switch” topology abstraction is largely agreed upon in the WG with a charter item
- WG has multiple designs related on network graph, e.g.,
  - [NL] node/link graphs
  - [NLP] node/link graph + path vector
  - [AP] Path vector w/ abstract network elements
  - ECMP complexity
- Proposal: First finish the AP design as a delivery

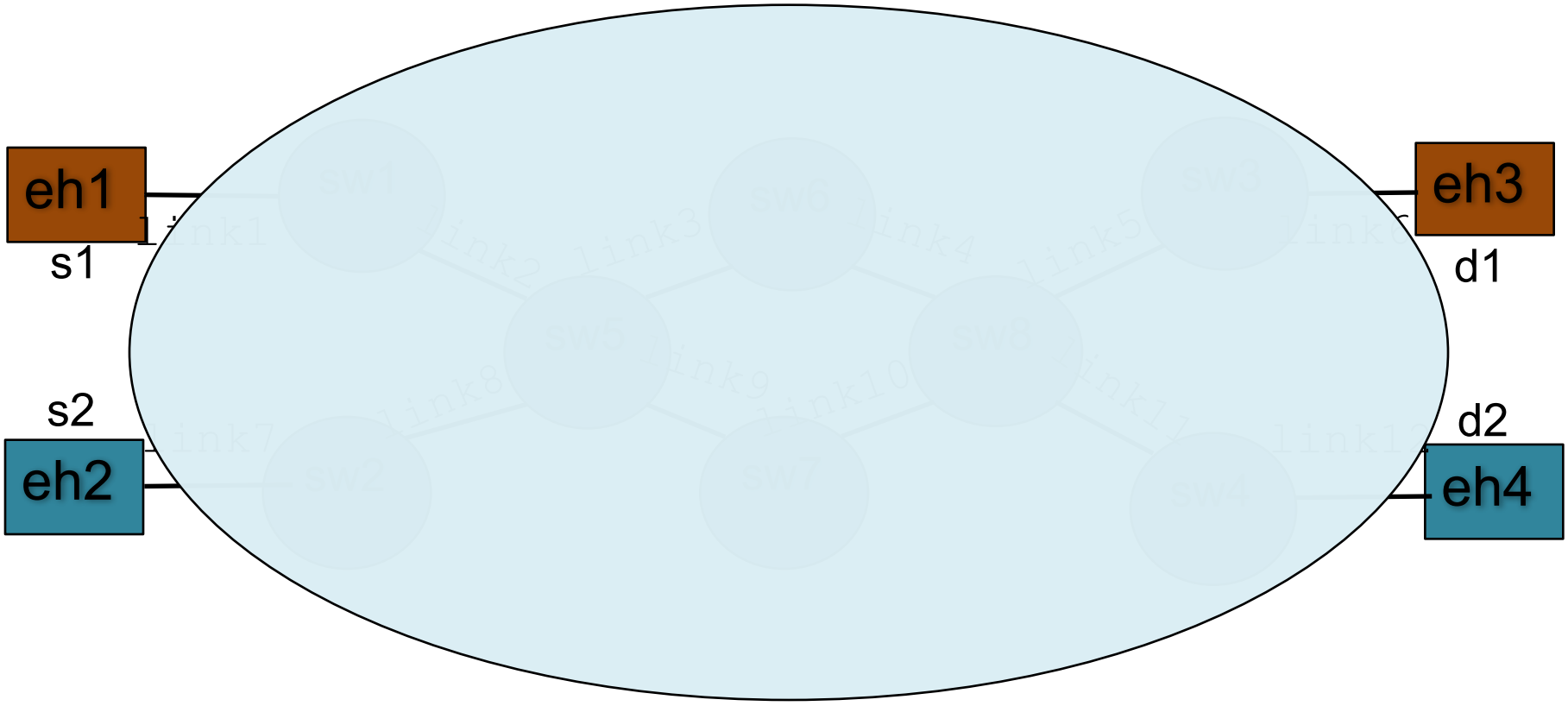


Application-Layer Traffic Optimization (TO) Framework

- App has a set of K flows f[i] = si -> di
- Path f[i].path for flow f[i] is computed by network
- App TO can do is to control traffic volume (f[i].x) for flow f[i]
  - x = [f[1].x, f[2].x, …, f[K].x]
  - App needs path properties (e.g., cost) of f[i] when computing f[i].x

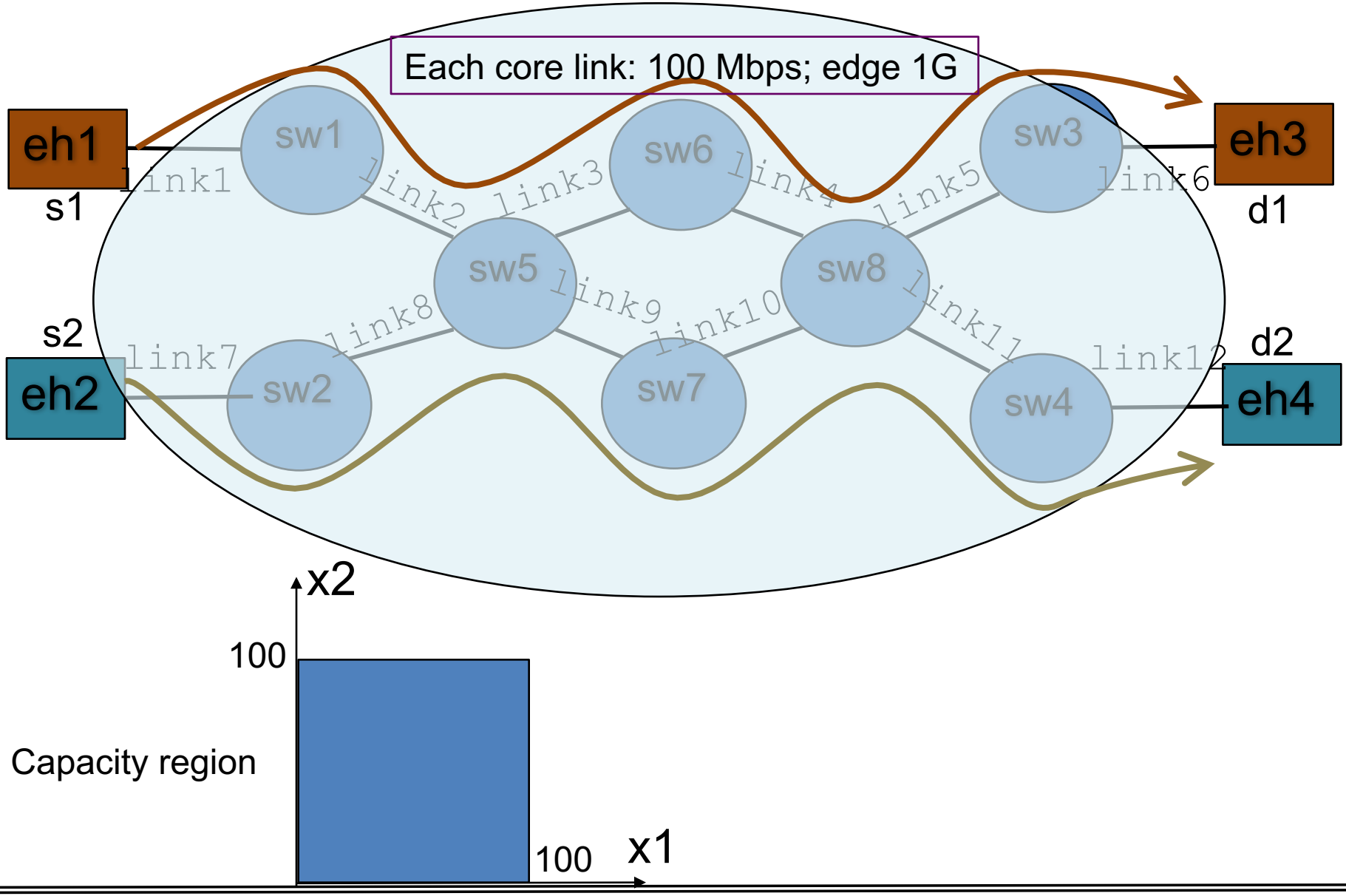
Value of ALTO is to provide hard-to-obtain, easy to use network information to applications.

Main Use Case of Network Graph is Capacity Region

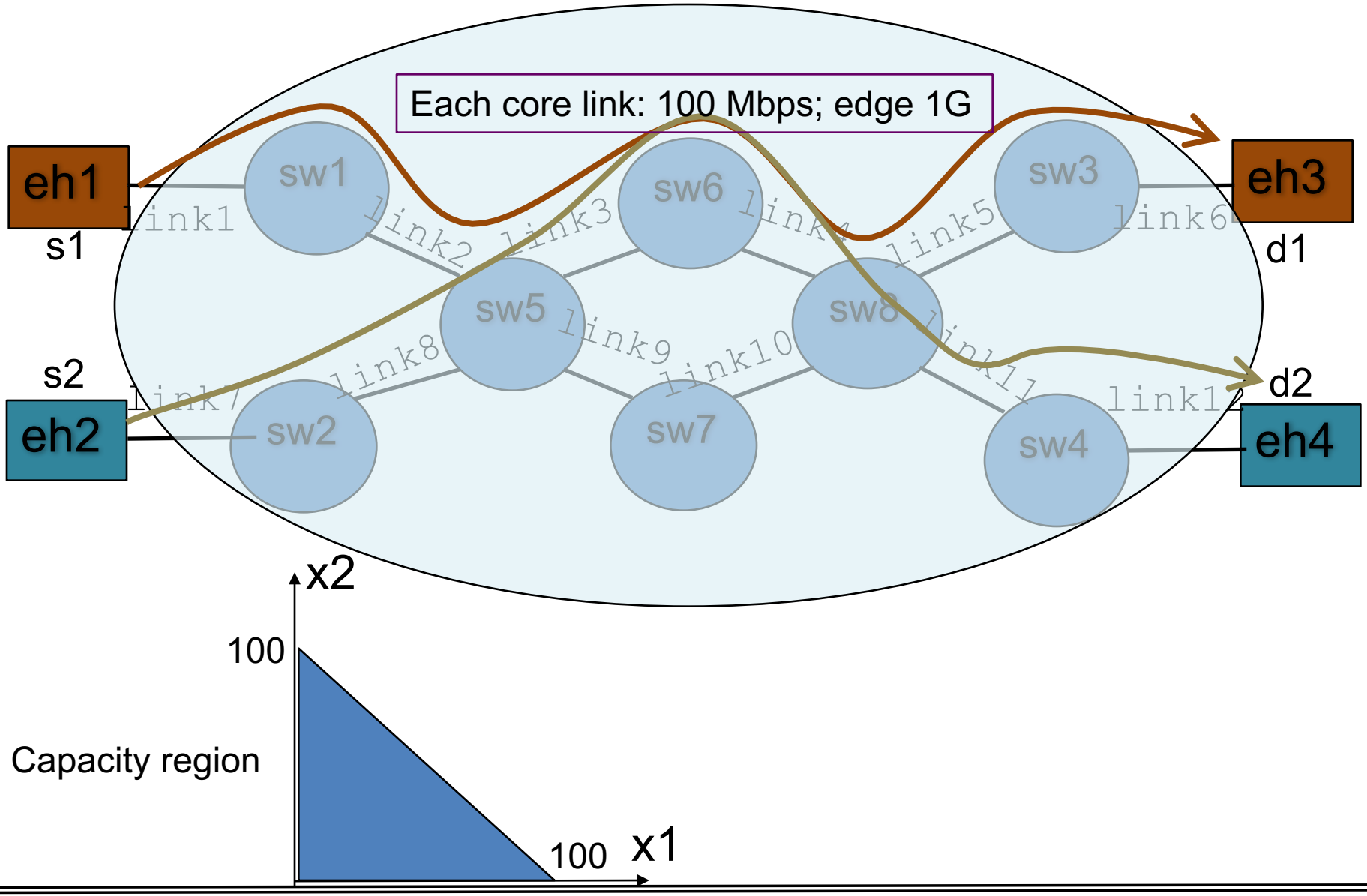


- App requests available bandwidth for two flows (s1->d1; s2->d2)
- Cost map returns f[1].bw = 100 Mbps; f[2].bw = 100 Mbps
- But the result can be ambiguous

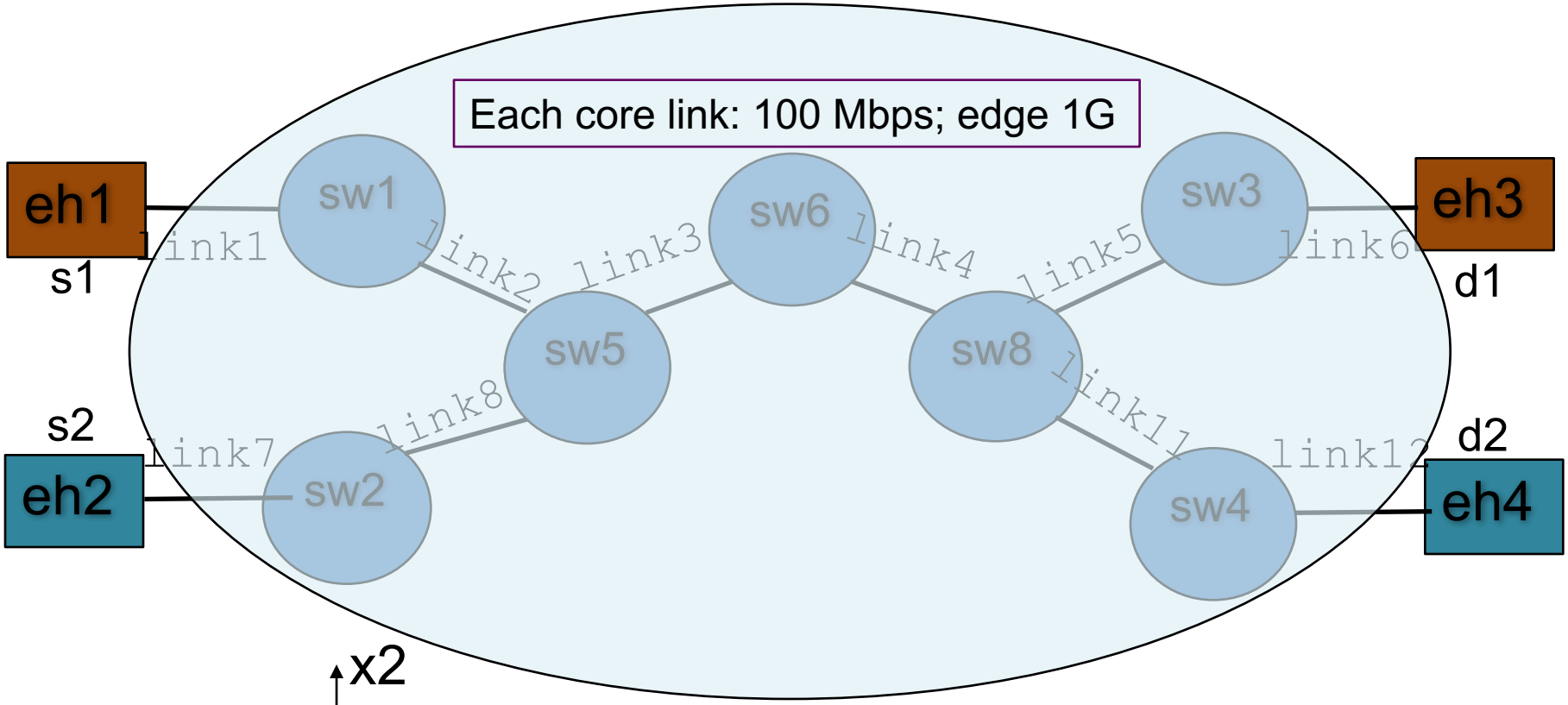
Main Use Case of Network Graph is Capacity Region

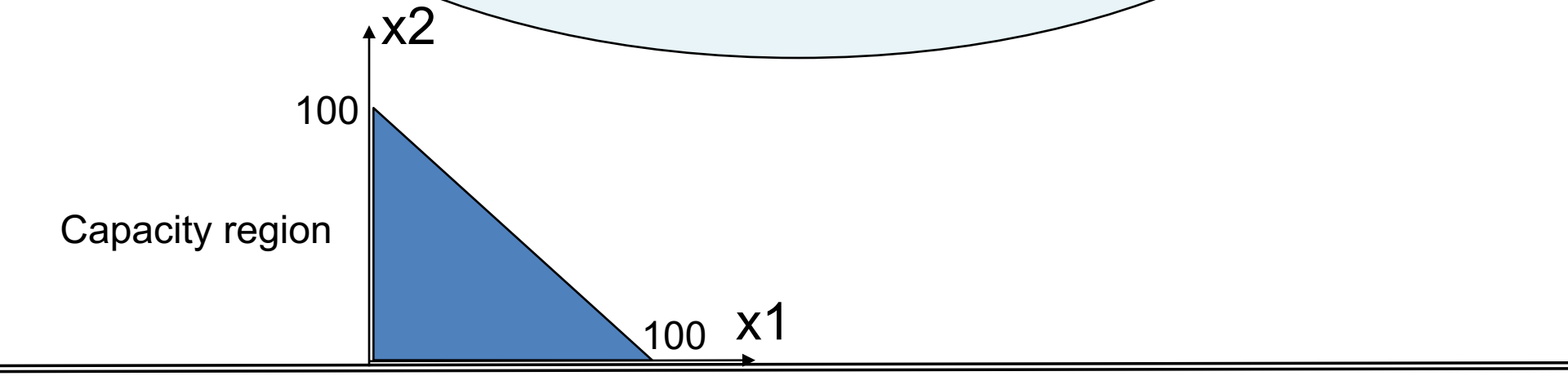


Main Use Case of Network Graph is Capacity Region



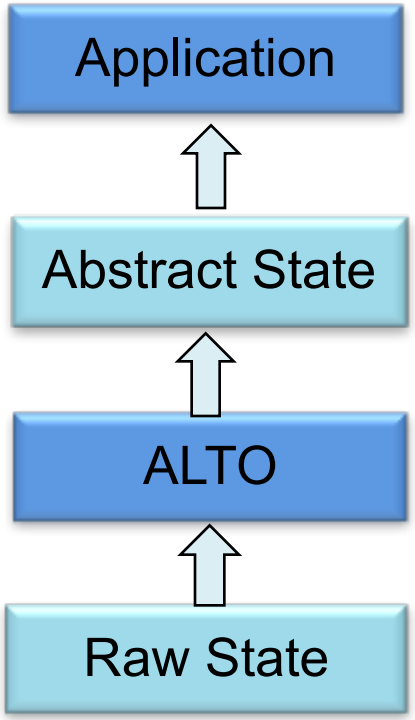
Main Use Case of Network Graph is Capacity Region





Design Choice

- [NL] node/link graphs:
  - Not enough information
- [NLP] node/link graph + path vector
  - No need for all the information
- [AP] Path vector w/ abstract network elements
  - Sweet spot



Path Vector Design Issue 1: Encode PV in Cost Maps

- **Cost Map w/ PV:**

```
HTTP/1.1 200 OK
…
application/alto-costmap+json

{
  "meta" : {
     "dependent-vtags" : […],
     "cost-type" : {“cost-metric”: ?, "cost-mode"  : ? }
  },
  "cost-map" : {
     "PID1": { "PID1":[], "PID2":["ne56", "ne67"], "PID3":[], "PID4":["ne57”]},
     "PID2": { "PID1":["ne75"], "PID2":[], "PID3":["ne75"], "PID4":[]}, …
  }
}
```

- Key Issue: What is the cost-metric and cost mode

AP Design Issue 1: Encode PV in Cost Maps

```
object {
      CostMapData cost-map;
    } InfoResourceCostMap
         : ResponseEntityBase;

    object-map {
      PIDName -> DstCosts;
    } CostMapData;

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

```
object {
      EndpointCostMapData endpoint-cost-map;
    } InfoResourceEndpointCostMap
         : ResponseEntityBase;

    object-map {
      TypedEndpointAddr -> EndpointDstCosts;
    } EndpointCostMapData;

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

- Authors of [RFC7285] anticipated that elements of a CostMap may need to be generic and hence used **JSONValue**

  =>

  CostMap and EndpointCostMap are **polymorphic (generic)** types that need type indicator, to indicate syntax and semantics
  - CostMap<T>
  - EndpointCostMap<T>

Issue 1: Design Choices

- I1-choice 1
  - Introduce specific cost-mode and cost-metric to indicate a PV Cost Map
- I1-Choice 2
  - A unifying cost-type scheme also handling multi-cost, cost calendar

Path Vector Design Issue 2: Query Precision

- Issue:
  - Current CostMap and EndpointCostMap are designed for **cross-product** queries
    {s1, s2, …., sn} -> {d1, d2, ..., dm}
    ===
    s1->d1, s1->d2, ..., s1->dm, s2->d1, ..., sn->dm
  - but such queries can involve a large number of paths, which may not be necessary but limit the opportunity for computing compact, abstract topology

Path Vector Design Issue 2: Query Precision

- Possibility: Make PV query to be non-cross product, e.g., a query enumerates the set of flows

```
POST /capacityregion/lookup HTTP/1.1
Host: alto.example.com
Content-Length: TBD
Content-Type: application/alto-flowparams+json
Accept: application/alto-costmap+json,application/alto-error+json

 {
    "flows" : [
          {"src": "ipv4:192.0.1.1", "dst": "ipv4:192.0.1.2"},
          {"src": "ipv4:192.0.1.3", "dst": "ipv4:192.0.1.4"},
          {"src": "ipv4:192.0.1.1", "dst": "ipv4:192.0.1.4"}
    ]
    … }
```

- Proposal:
  - Coordinate with flow cost service or more generally, do we proceed w/ Routing State Abstraction w/ Declared Equivalence

Issue 2: Design Choices

- I2-choice 1
  - Use native Cost Map for PIDs and Endpoint Cost Map queries
- I2-Choice 2
  - Introduce flows, and more general, introduce routing state abstraction query language as a general query mechanism

Path Vector Design Issue 3: Provide PV Elem. Properties

- Design choices
  1. Inline: Embed in the same cost map
  2. Reference: Use dependent-vtag to refer to a separate map (e.g., unified element properties)

Path Vector Design Issue 3: Inline

```
HTTP/1.1 200 OK
…
application/alto-costmap+json

{
  "meta" : {
     "dependent-vtags" : […],
     "cost-type" : {“cost-metric”: “ane”, "cost-mode"  : ”path-vector”  }
  },
  "cost-map" : {
     "PID1": { "PID1":[], "PID2":["ne56", "ne67"], "PID3":[], "PID4":["ne57”]},
     "PID2": { "PID1":["ne75"], "PID2":[], "PID3":["ne75"], "PID4":[]}, …
  }
  “nep-map” : {
     “ne56”: {“bw”: 10, … }
  }
}
```

- Key remaining issue:
  - How to indicate the properties/values specified in the nep-map?
  - Need to indicate multi-cost map for the nep-map in meta

Path Vector Design Issue 3: Reference

```
HTTP/1.1 200 OK
…
application/alto-costmap+json

{
   "meta" : {
      "dependent-vtags" : […
         refer to an nep map
      ],
      "cost-type" : {“cost-metric”: “ane”, "cost-mode"  : ”path-vector”  }
   },
   "cost-map" : {
      "PID1": { "PID1":[], "PID2":["ne56", "ne67"], "PID3":[], "PID4":["ne57”]},
      "PID2": { "PID1":["ne75"], "PID2":[], "PID3":["ne75"], "PID4":[]}, …
   }
}
```

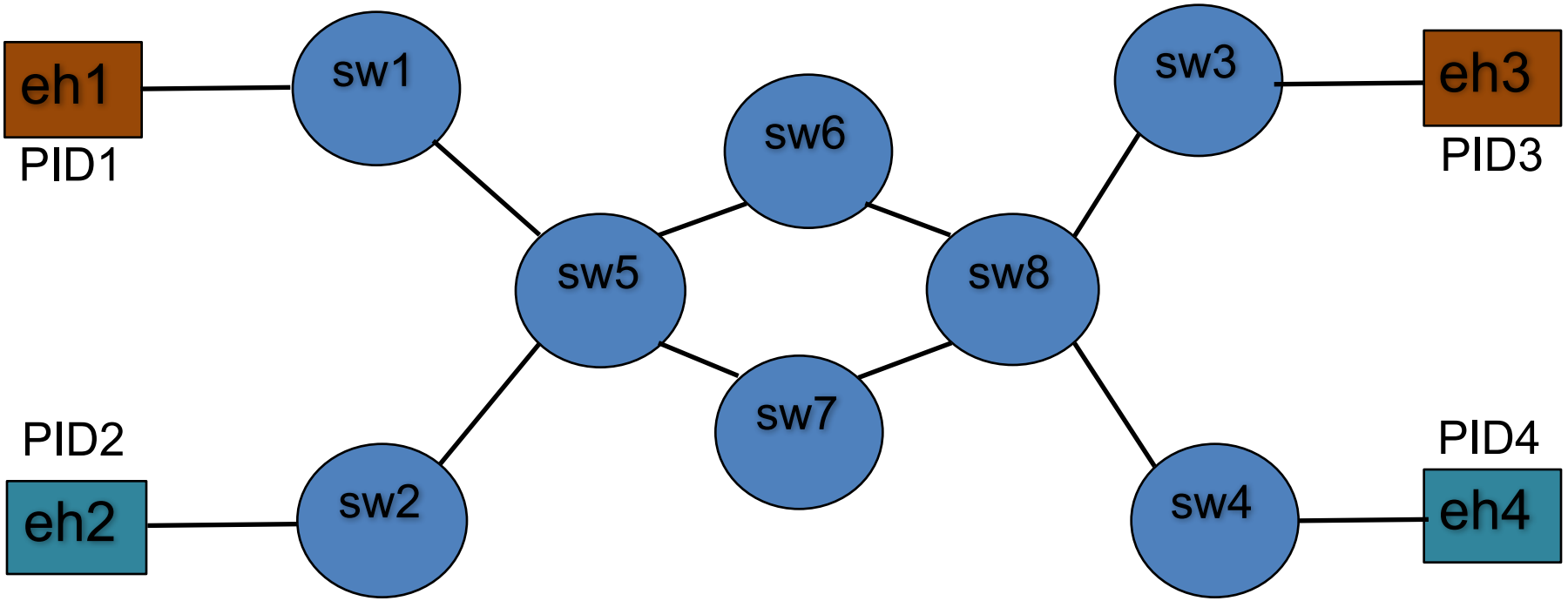
Road Map: Decision Points

NL Graph
NLGraph+PV
APV

Issue 1	Issue 2	Issue 3
Specific Cost Type	Native Cost Map Query	Inline
Scheme together w/ MC, CC	Flow Query+RSA	Reference

Backup Slides

Q: How to Support ECMP Path



- ECMP for eh1 -> eh3, single path through sw6 for eh2 -> eh4
  - PID1 -> PID3: [{“ne”: “sw5-6”, “w”: 0.5},
    {“ne”: “sw6-8”, “w”: 0.5},
    {“ne”: “sw5-7”, “w”: 0.5},
    {“ne”: “sw7-8”, “w”: 0.5}]
  - PID2 -> PID4: : [{“ne”: “sw5-6”, “w”: 1},
    {“ne”: “sw6-8”, “w”: 1}]

```
object {
   JSONNumber w;      // flow weight
   JSONString  ne;    // network element
} FlowElement;

object {
      cost-map.DstCosts.JSONValue
            -> FlowElement<0,*>;
      meta.cost-mode = ”flow”;
} InfoResourcePVCostMap : InfoResourceCostMap;
```

One More Example

- How do we allow statistics on a cost metric (e.g., routingcost)

```
HTTP/1.1 200 OK
…
application/alto-costmap+json

{
   "meta" : {
      "dependent-vtags" : […],
      "cost-type" : {“cost-metric”: “latency-stat”, "cost-mode"  : ”basic-stat-object”  }
   },
   "cost-map" : {
      "PID1": {
            "PID1”:{“min”; 1, “max”: 2, “avg”: 1.5} ,
            "PID2”:{“min”; 2, “max”: 5, “avg”: 2.5} ,
            …
      ...
   }
}
```

Path Vector Design Issue 1: Encode PV in Cost Maps

- The **"cost-mode"** field of the "cost-type" field of "meta" of each CostMap is the **type indicator**

```
object {
      CostMetric  cost-metric;
      CostMode    cost-mode;
      [JSONString description;]
} CostType;
```

  - CostMap<cost-mode>, i.e., CostMap<numerical> and CostMap<ordinal>
    - "numerical" indicates floating point numbers {6.1.2.1}
    - "ordinal" indicates "non-negative" integers {6.1.2.2}
  - "cost-metric": indicates the semantics (routingcost, bw, …)
    - CostMap<numerical> routingcost, bw
    - CostMap<ordinal> routingcost, bw
    - EndpointCostMap<numerical> routingcost, bw
    - EndpointCostMap<ordinal> routingcost, bw