ALTO Extension: Routing State Abstraction using Declarative Equivalence

draft-gao-routing-state-abstraction-00
draft-yang-alto-topology-06

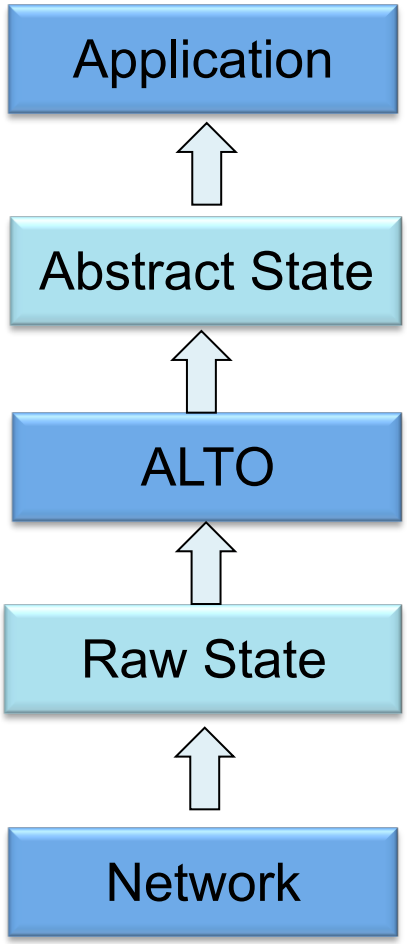
Presenter: Wendy Roome/Young Lee

July 21, 2015 @ IETF 93

Bigger Picture

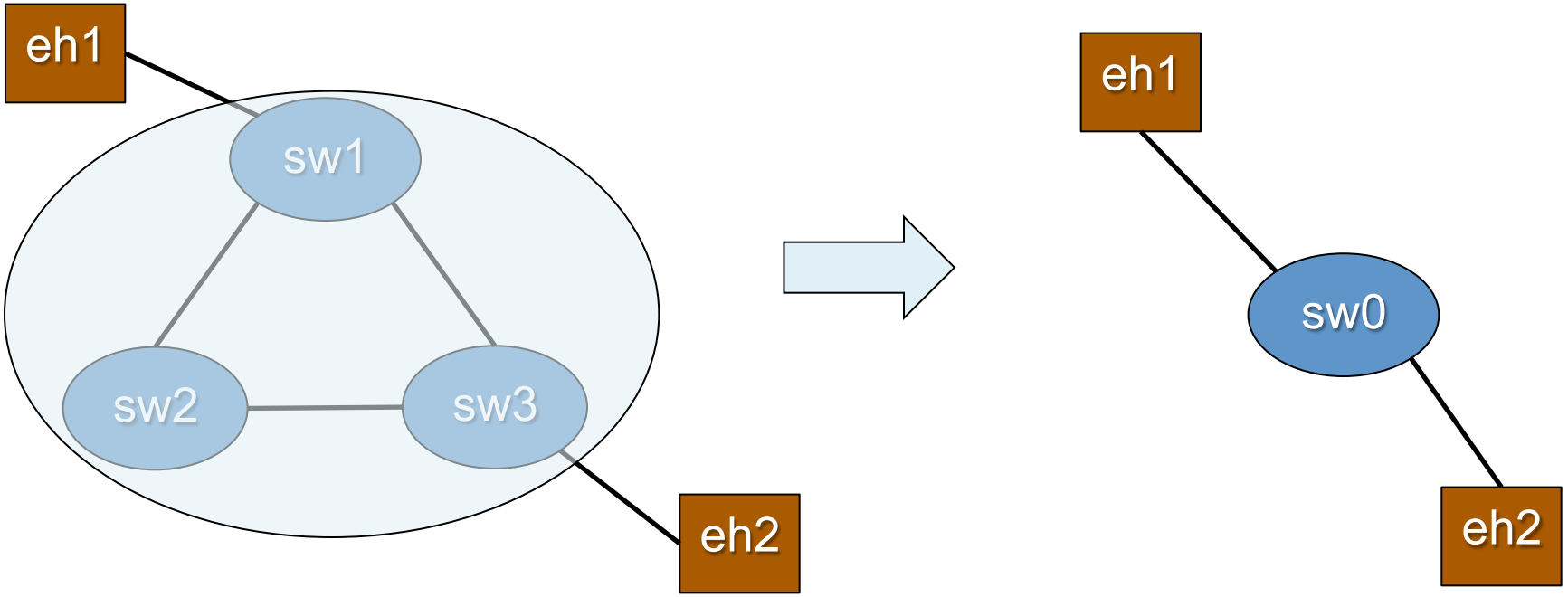
- A general objective of ALTO is to provide network state to applications for better traffic engineering

- It is important that ALTO provide abstract network states, to
  - protect information privacy
  - improve scalability

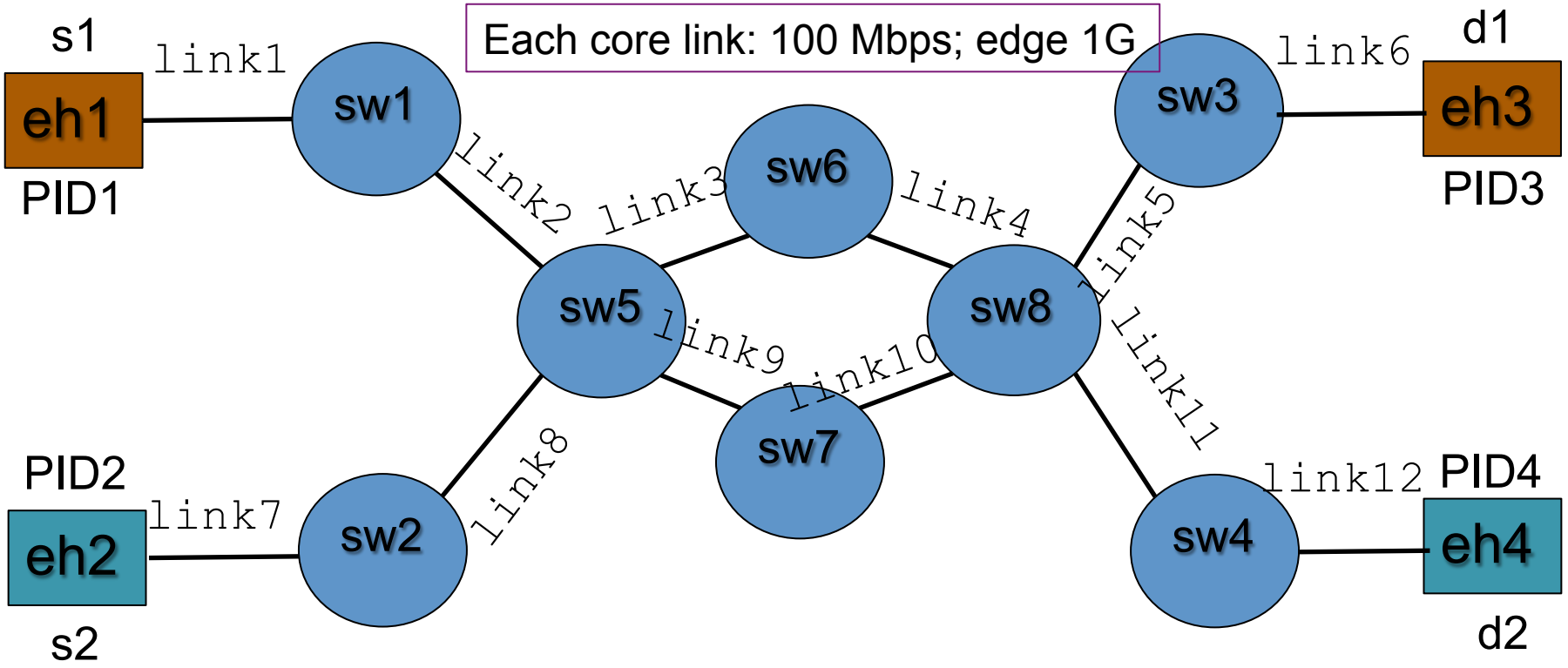


Key Question: How to Compute Abstract State

- Basic approach: static template (e.g., single-node template)



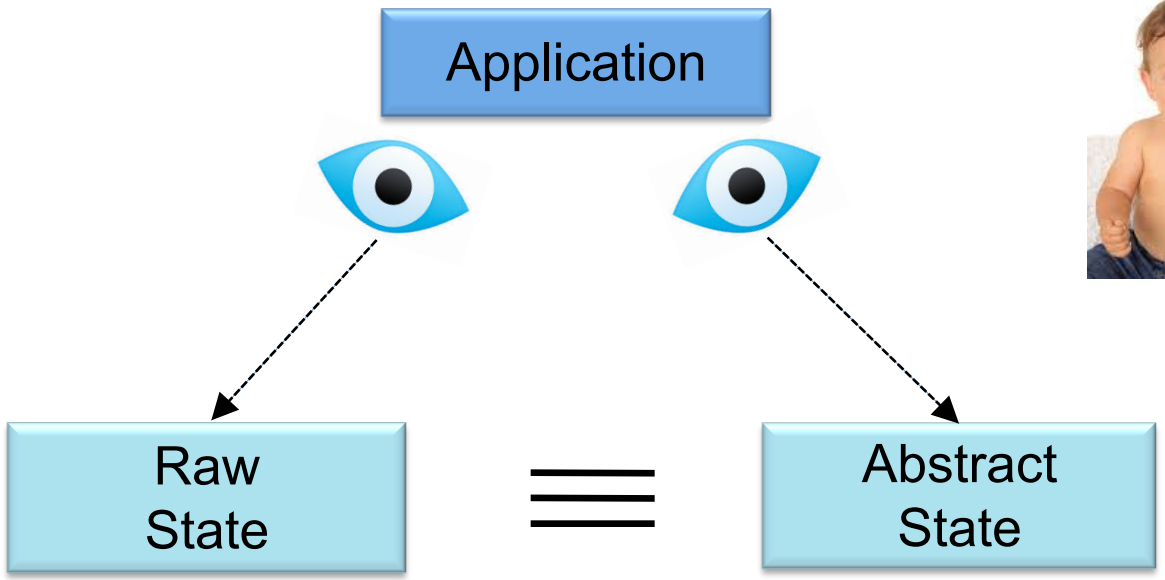
What we learned from the path-vector example



- App requests available bandwidth for two pairs:
  - PID1 (eh1) -> PID3 (eh3); PID2 (eh2) -> PID4 (eh4)
- Ambiguity of cost map based on single-switch abstraction:
  - Two disjoint paths (200 Mbps when concurrent), e.g.,
    - PID1 -> PID3: sw5 -> sw6 -> sw8;
    - PID2 -> PID4: sw5 -> sw7 -> sw8
  - Shared bottleneck (still 100 Mbps when concurrent)

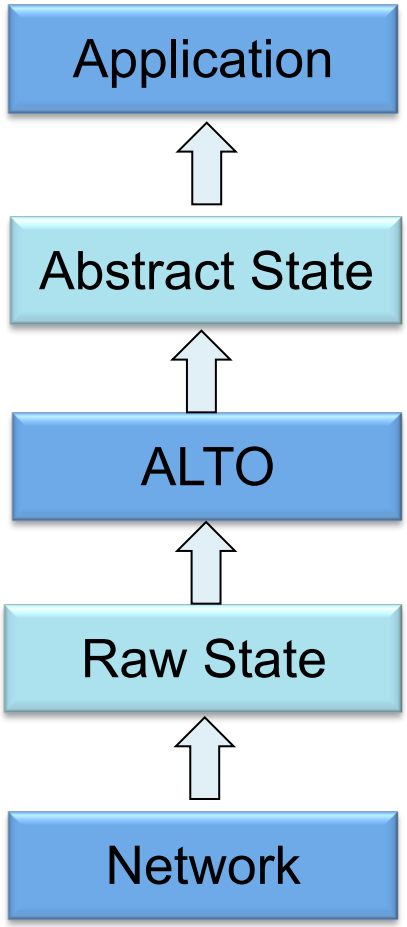
Our General Design Approach

- ALTO server computes dynamic, minimal network state
  - Compute abstract state that is smaller but *equivalent* to application required

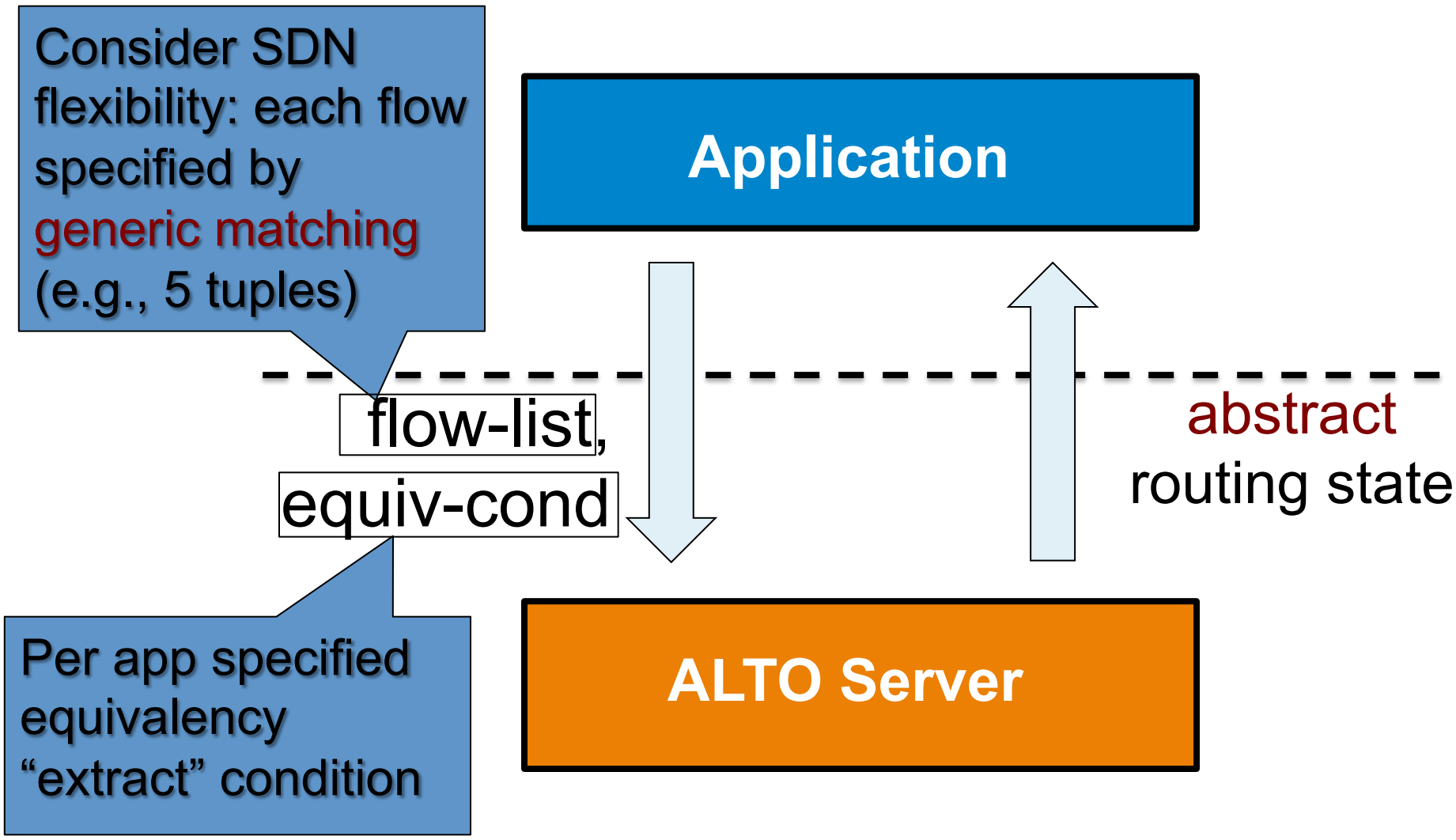


ALTO Extension: Routing State Abstraction Service based on Declarative Equivalence (RSA-DE)

- Why routing state
  - Routing state is basic
  - Current focus of ALTO (e.g., cost map, ECS) is mostly on routing state
- We may consider other types of network state in future extensions



Routing State Abstraction Service



How to Specify `flow-list`

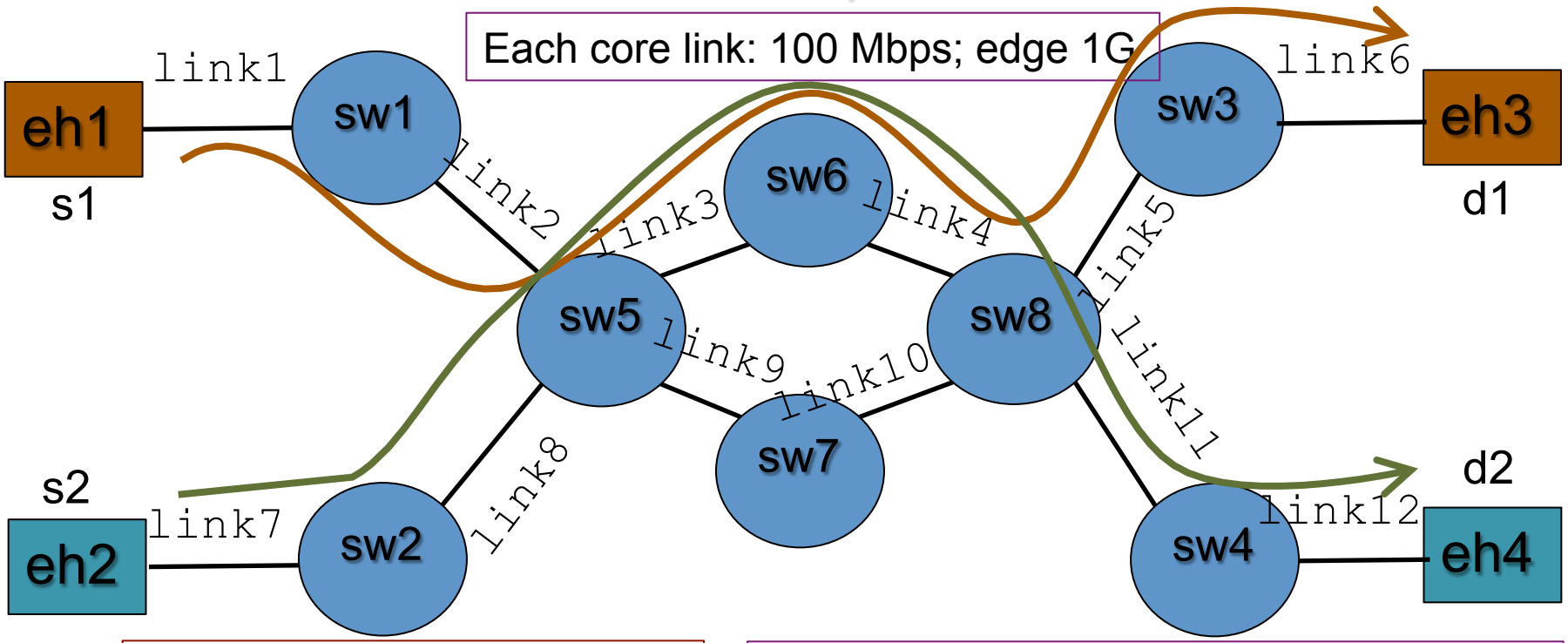
- This is relatively straightforward
  - Server announces matching capabilities in IRD
    - E.g., IP address only, allowing ports, …
    - Incremental deployment: allows IP only
  - A flow-list is a list of flows, where each flow is specified by a matching condition (e.g., OpenFlow like) condition

How to Specify `equiv-cond`: Intuition

- General structure of a network application
  - Has a set of flows
  - Has a set x of variables (e.g., rate of each flow)
  - Has a goal: min/satisfy obj(x)
    s.t.
    x satisfies constraints
  - obj(x) or the constraints may use network information, e.g.,
    - obj = x[1] * routingcost(f1) + x[2] * routingcost(f2)
    - any link e: sum(x[i]: fi uses link e) ≤ bw(e)

- `equiv-cond` is to convey the usage of network information in the application

Example



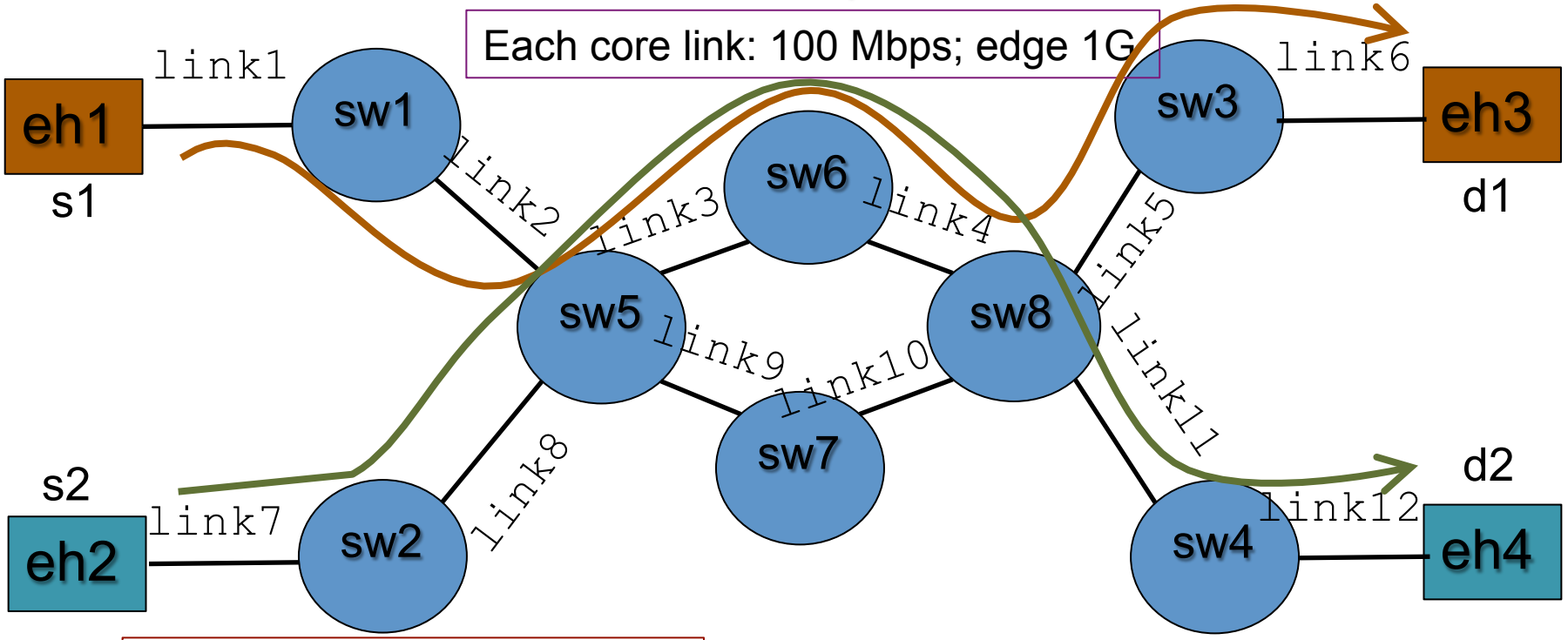
raw state

	r[1]	r[2]	bw
link1	1	0	1G
link2	1	0	100M
link3	1	1	100M
Link4	1	1	100M
link5	1	0	100M
link6	1	0	1G
…			

Given routing for given flows, the network raw state can be considered as a set of vectors, where each vector's dimension is the number of edges (links):

- r[i][e]: a vector representing if flow i uses link e. For example, r[2] shows that the route of flow 2 uses links 3/4, …, not links 1/2/5/6, …
- attr[e]: is the value of attr for link e. For example, bw[e] is the available bandwidth of link e.

Example

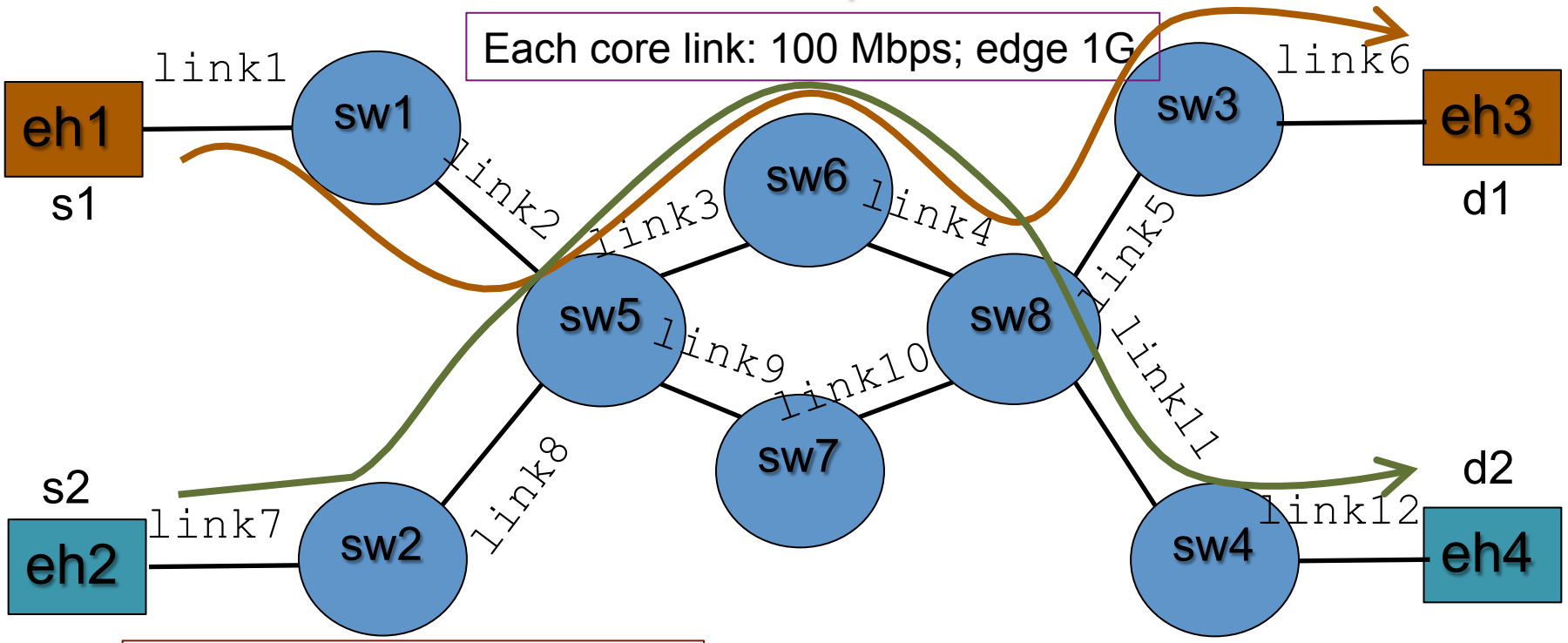


raw state

	r[1]	r[2]	bw
link1	1	0	1G
link2	1	0	100M
link3	1	1	100M
Link4	1	1	100M
link5	1	0	100M
link6	1	0	1G
…			

A naïve approach is to return the whole raw network state

Example



raw state

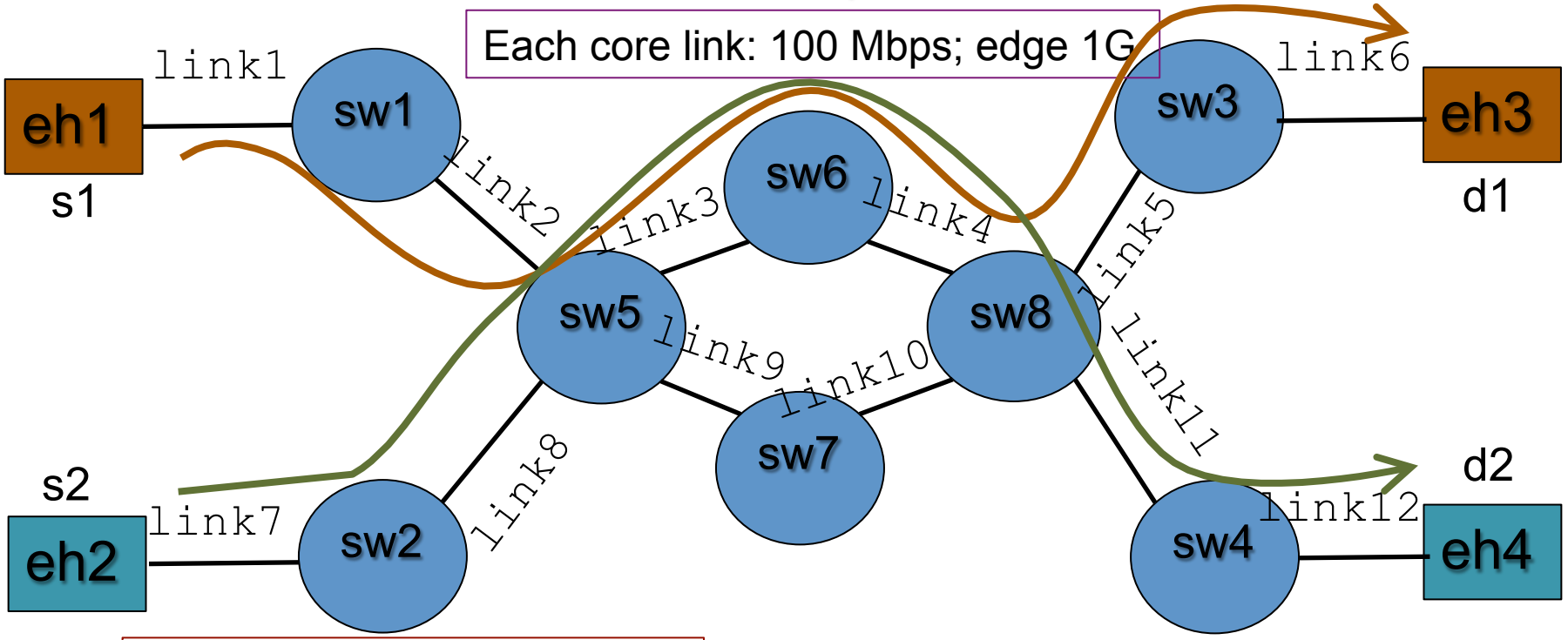
	r[1]	r[2]	bw
link1	1	0	1G
link2	1	0	100M
link3	1	1	100M
Link4	1	1	100M
link5	1	0	100M
link6	1	0	1G
…			

App decision variables: `x[1], x[2].`

`equiv-cond:` Lambda specification of app usage of network state in constraints:

```
any e: r[1][e] * x[1] +
       r[2][e] * x[2]
       <= bw[e]
```

Example



raw state

	r[1]	r[2]	bw
link1	1	0	1G
link2	1	0	100M
link3	1	1	100M
Link4	1	1	100M
link5	1	0	100M
link6	1	0	1G
…			

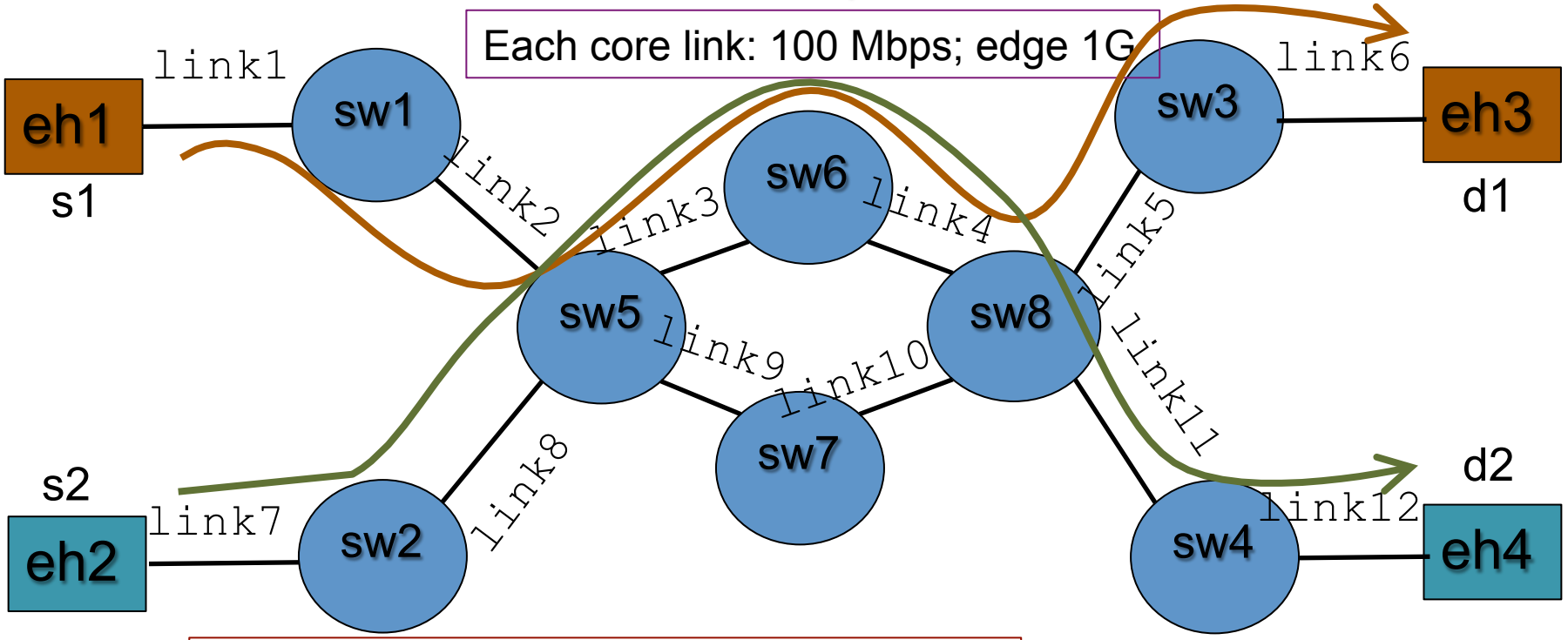
Insight: If the attributes of a link e do not appear in a tight (independent) constraint, the link does not need to be known to the app.

Implementation

- ALTO will not define the implementation, the discussion of implementation is to show feasibility and help with understanding

- Steps to implement RSA-DE
  - ALTO server looks up routing for each flow i to obtain r[i]
    - E.g., looks up in Flow Rule Manager (FRM) in ODL/ONOS
  - ALTO server applies redundancy elimination to find the minimal set of independent links

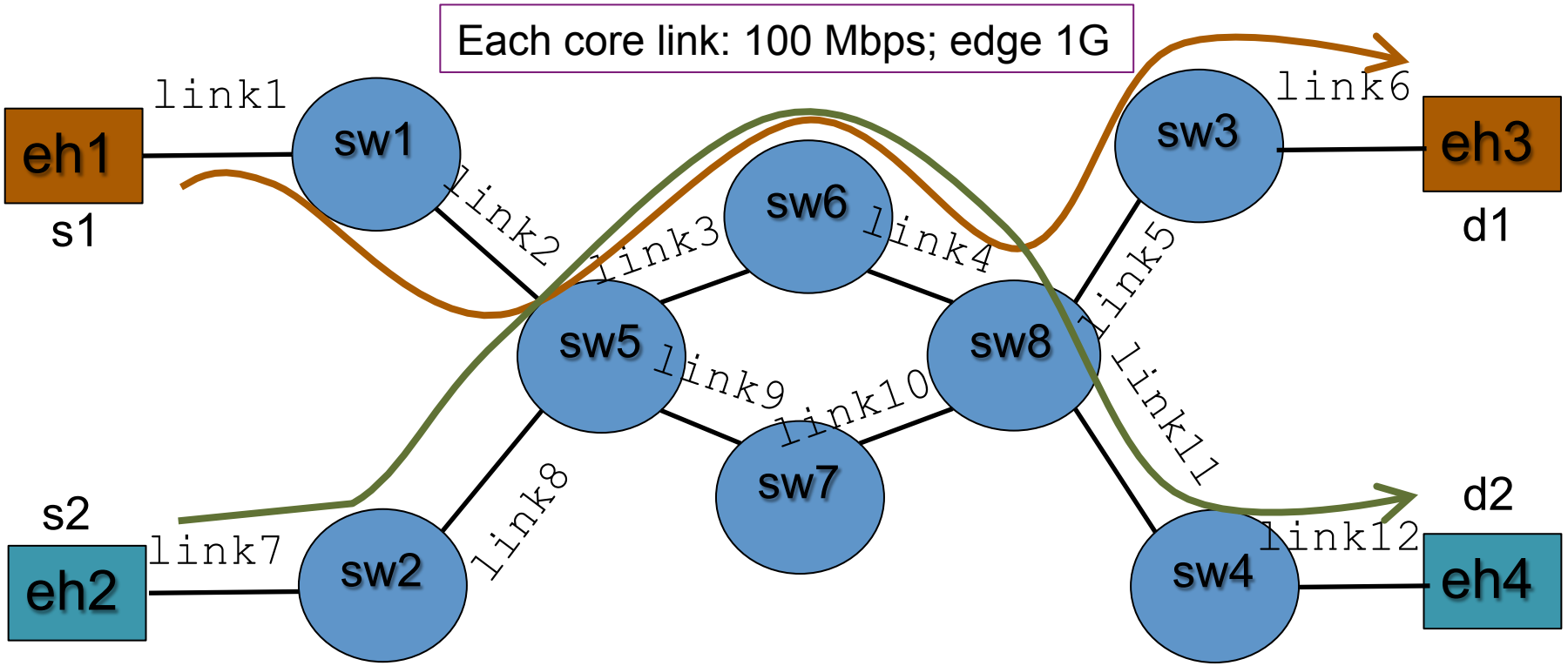
Example



	r[1]	r[2]	var	bw
link1	1	0	x[1]	1G
link2	1	0	x[2]	100M
link3	1	1		100M
Link4	1	1		100M
link5	1	0		100M
link6	1	0		1G
…				

Fast (redundancy elimination) algorithm to compute links providing non-redundant constraints.

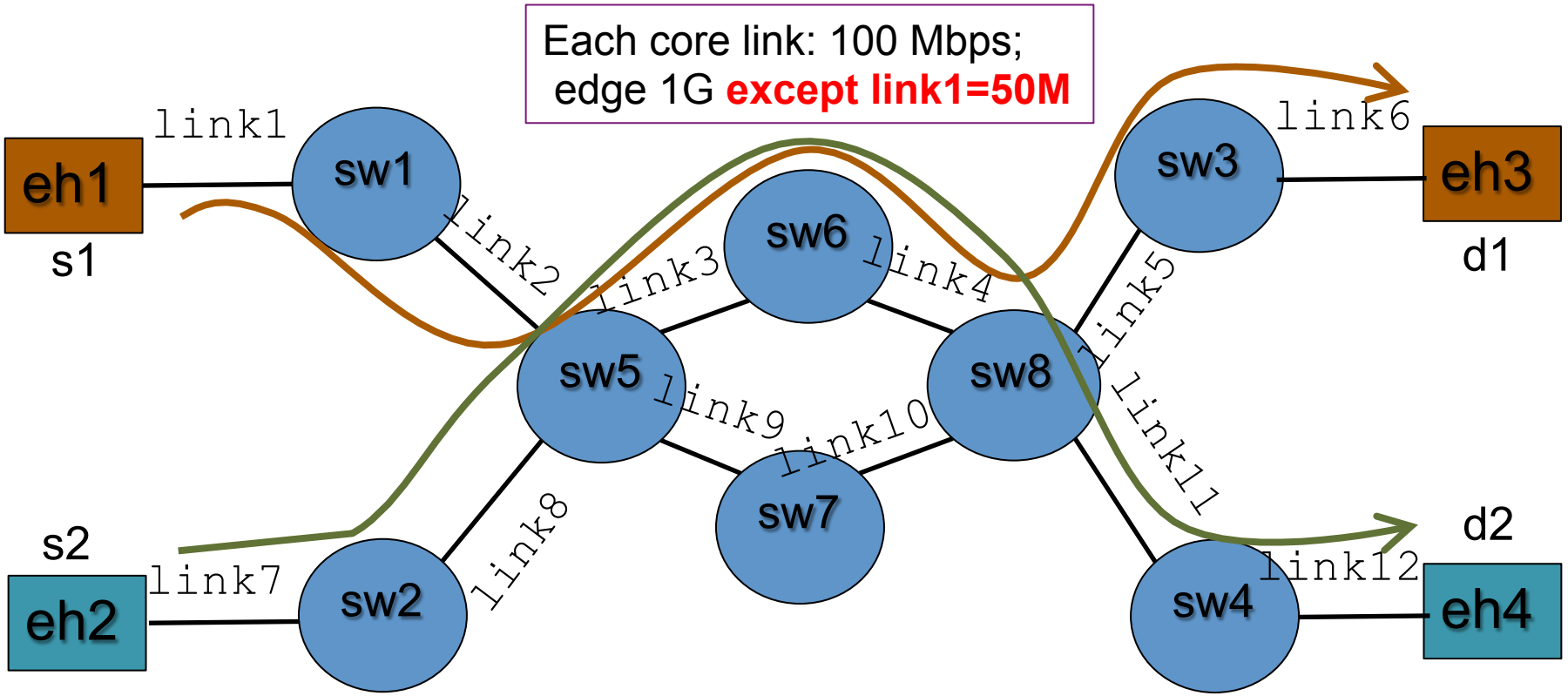
Example: Result



s_1 -> d_1: { ane_1(≤100M) }

s_2 -> d_2: { ane_1(≤100M) }

Example: Result



s_1 -> d_1: { ane_1(≤100M), ane_2(≤50M) }

s_2 -> d_2: { ane_1(≤100M)}

Summary

- RSA-DE provides a powerful, general interface to allow applications to obtain network state
  - It is a generalization of the previous path-vector design
  - It works in the new SDN setting
  - More details on RSA-DE see backup slides

- Interest in the WG to pursue this direction as an ALTO extension?

Backup Slides

How to Specify `equiv-cond`

- Link Properties: announced in IRD capabilities
  - r[i] must be supported
  - bw, delay, loss, routingcost, …
- `equiv-cond`
  - Variables: A list of opaque variables
  - Constraints: A list of lambda inequality expression for a given link, e.g., `any e in E:`

```
r[1][e]*x[1] + r[2][e]*x[2] <= bw[e]
```

  - To simplify the representation:
    - [e] can be ignored since it is implied.
    - Allow meaningful variable names
    - Use underscore to represent array instead of brackets

How to Specify `equiv-cond (cont.)`

```
equiv-cond       :=  constraint | constraint, equiv-cond
constraint       :=  linear-expression OP linear-Expression
OP               :=  < | > | <= | >= | =
linear-expression :=  constant | link-property-name | variable
                    | constant * linear-expression
                    | link-property-name * linear-expression
                    | -linear-expression
                    | linear-expression + linear-expression
```

How to Specify `equiv-cond (cont.)`

- The order to parse a constraint
  - Symbols in the variable-list:
    - As `variables`
  - Symbols announced as capabilities
    - As `link-property-names`
  - Symbols for mathematical constants
    - As `constants`
  - Unknown symbols
    - As undefined if the variable-list is provided
    - As `variables` if intelligent parsing is enabled

How to Specify `equiv-cond (cont.)`

```
variable-list      := variable | variable, variable-list
variable           := [a-zA-Z][0-9a-zA-Z-]*


rs-query           := flow-list constraint-list [variable-list]
flow-list          := flow | flow, flow-list
flow               := generic-match-condition
```

How to Specify `equiv-cond (cont.)`

```
{
    "query": {
        "flows": [
            {
                "dst": "ipv4:192.168.1.22",
                "src": "ipv4:192.168.0.11",
                "tcp-dstport": "80"
            },
            {
                "dst": "ipv4:192.168.3.44",
                "src": "ipv4:192.168.2.33",
                "tcp-dstport": "23"
            }
        ],
        "conditions": [
            "r_0*f0 + r_1*f1 <= bandwidth",
            "r_0*f0 + r_1*f1 <= u*capacity"
        ],
        "variables": [
            "f0",
            "f1",
            "u"
        ]
    }
}
```

An example where the user creates two flows with three variables.

- Variables `f0` and `f1` represent the bandwidth for each flow.
- Variable `u` represents the maximum link utilization.

Routing State Abstraction Service: Query Input

```
rs-query    := flow-list equiv-cond
flow-list   := flow [flow-list]
flow        := generic-match-condition
```

- App provides two input parameters
  - `flow-list`: A set of flows
    - Consider the flexibility of SDN, which allows routing based on generic matching (e.g., 5 tuples) => each flow is specified by generic matching condition
  - `equiv-cond`: app declared equivalency condition

Path Vector: Example

HTTP/1.1 200 OK
Content-Length: TDB
Content-Type:
application/alto-costmap+json

```
{  "meta" : {
     "dependent-vtags" : [
       { "resource-id": "my-default-network-map”,
          "tag": "3ee2cb7e8d63d9fab71b9b34cbf764436315542e” },
       {"resource-id": "my-topology-map",
          "tag": "4xee2cb7e8d63d9fab71b9b34cbf76443631554de"
       }
     ],
     "cost-type" : {“cost-metric”: “bw”, "cost-mode"  : "path-vector”  },

     "cost-map" : {
        "PID1": { "PID1":[], "PID2":["ne56", "ne67"], "PID3":[], "PID4":["ne57"]
         },
         "PID2": { "PID1":["ne75"], "PID2":[], "PID3":["ne75"], "PID4":[]
         }, …
     }
}
```