ALTO Topology Extension

draft-scharf-alto-topology-00
draft-yang-alto-topology-03

M. Scharf (michael.scharf@alcatel-lucent.com)
G. Bernstein (gregb@grotto-networking.com)
Young Lee (leeyoung@huawei.com)
W. Roome (w.roome@alcatel-lucent.com)
Y. Richard Yang (yry@cs.yale.edu)

July 25, 2014 @ IETF 90

Document Status

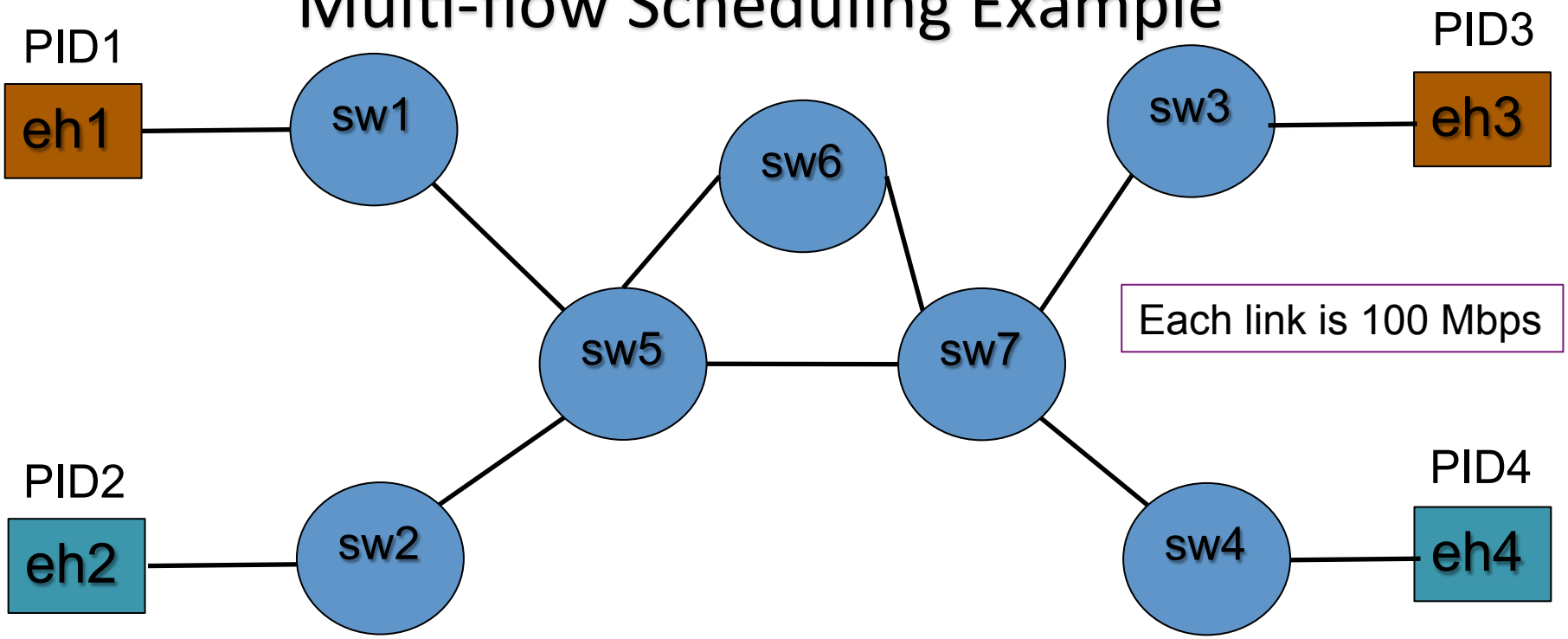
- Both drafts (`draft-scharf-alto-topology-00` and `draft-yang-alto-topology-03`) are based on the re-chartering discussions at IETF89. They will be merged into a single document before the next IETF.

- The goal is to address charter milestone (Jul 2015 - Submit network graph format document)

Changes from –02 to –03

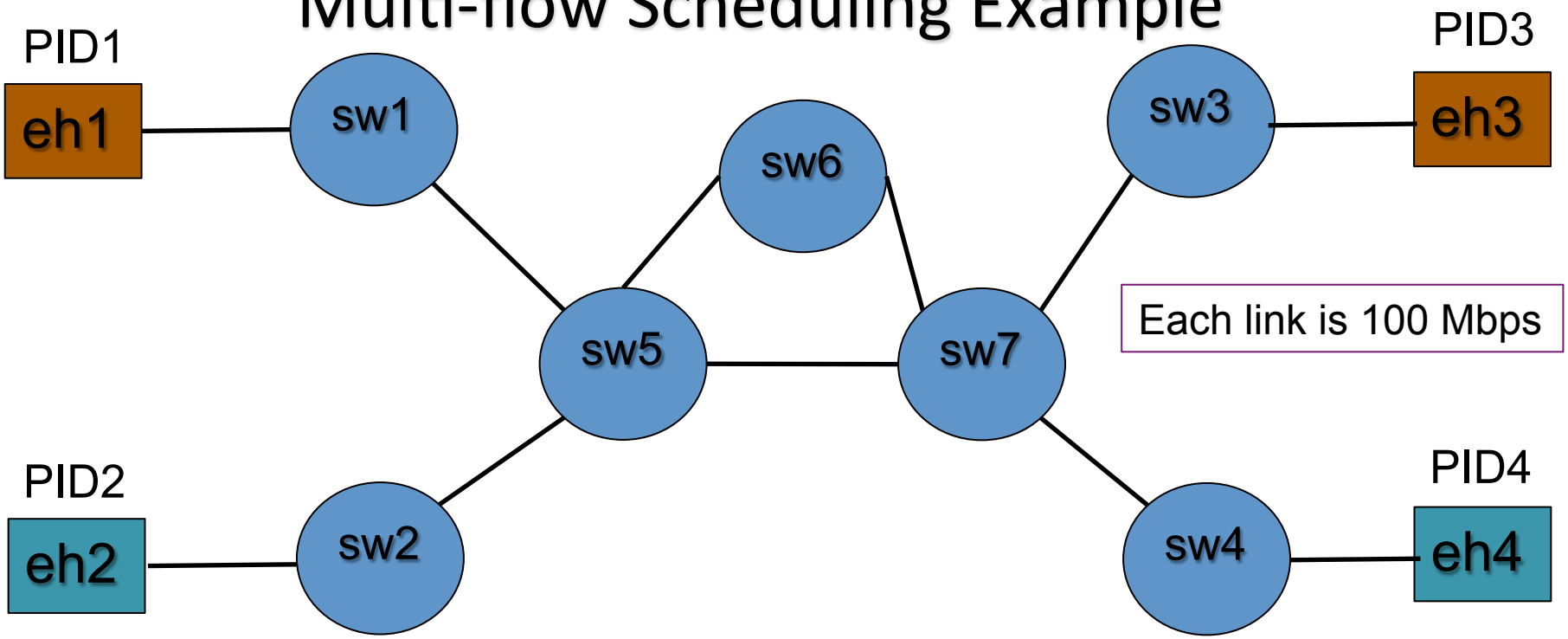
- Add more text to clarify that the goal is to work on the application level, not routing level (i.e., focus on external clients with respect to network providers rather than on internal clients).
- Add a simple, multi-flow use case example (Section 3) to illustrate the need to reveal individual network elements.
- Add a new section (Section 4) to specify the grammar and an example using path vector in cost maps.
- Add a design alternative (opaque network element property map) as Section 5.
- Give one possible node-edge graph grammar in Section 6.

Multi-flow Scheduling Example



- ALTO cost map will indicate
  - PID1 (eh1) -> PID3 (eh3): 100 Mbps; PID2 (eh2) -> PID4 (eh4): 100 Mbps
- If the routing uses different paths (e.g., PID1 -> PID3: sw5 -> sw6 -> sw7; PID2 -> PID4: sw5 -> sw7), the app will get 200 Mbps;
- Otherwise, the app will get only 100 Mbps.

Multi-flow Scheduling Example



- Implications:
  - Only aggregated cost map info cannot distinguish the ambiguity.
  - Need to provide info on on-path individual network elements to reveal shared bottlenecks.

Approach I: Path Vector

- Providing path vector as a new cost mode for ALTO cost map so that application can be informed of individual network elements along network chosen path, through which it can infer shared bottlenecks, shared reliability risk, etc.

```
object {
      cost-map.DstCosts.JSONValue -> JSONString<0,*>;
      meta.cost-mode = "path-vector”;
} InfoResourcePVCostMap : InfoResourceCostMap;
```

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-mode"  : "path-vector” } },

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

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-mode"  : "path-vector” } },

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

Remaining issue: how to specify prop. of elem. of PV?

Design: Opaque Network Element (ONE) Property Map

- Do not reveal whether a network element is a switch/router or a link

```
object-map {
     JSONString -> NetworkElementProperties; // name to properties
} NetworkElementMapData;

object {
    JSONString bw;
    JSONString srlg<0,*>;
} NetworkElementProperties;
```

ONE Property Map: Example

```
HTTP/1.1 200 OK
Content-Length: TBD
Content-Type: application/alto-onepropmap+json

{
  "meta" : {
    "vtag": {
      "resource-id": "my-topology-map",
      "tag": "da65eca2eb7a10ce8b059740b0b2e3f8eb1d4785"
    }
  },
  ”oneprop-map" : {
    "ne57" : {"bw" : 100, "srlg" : [1, 3]}, // link sw5->sw7
    "ne75" : {"bw" : 100, "srlg" : [1, 3]}, // link sw7->sw5
    "ne56" : {"bw" : 100, "srlg" : [1]},    // link sw5->sw6
    "ne65" : {"bw" : 100, "srlg" : [1]},    // link sw6->sw5
    "ne67" : {"bw" : 100, "srlg" : [3]},    // link sw6->sw7
    "ne76" : {"bw" : 100, "srlg" : [3]},    // link sw7->sw6
  }
}
```

Comments on the PV+ONE Property Map Approach

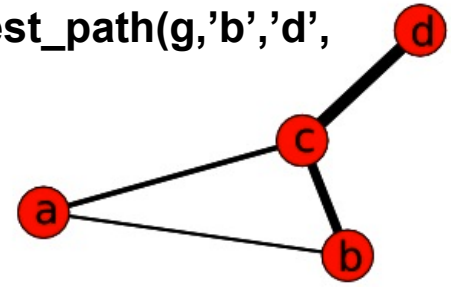
- Advantages
  - It provides high information hiding
    - Better suited for external clients
    - Application may still try to construct a (node-edge) topology, but the revealed path elements in PV may not be complete (ALTO Server provides only enough to allow app detecting shared bottleneck, reliability risks).
  - It can reflect precisely both standard alg. (e.g., shortest path routing) and policies
- Problem
  - PV may not scale: size of $O(N^2 P)$, where N is # PIDs, P is avg # of hop counts.

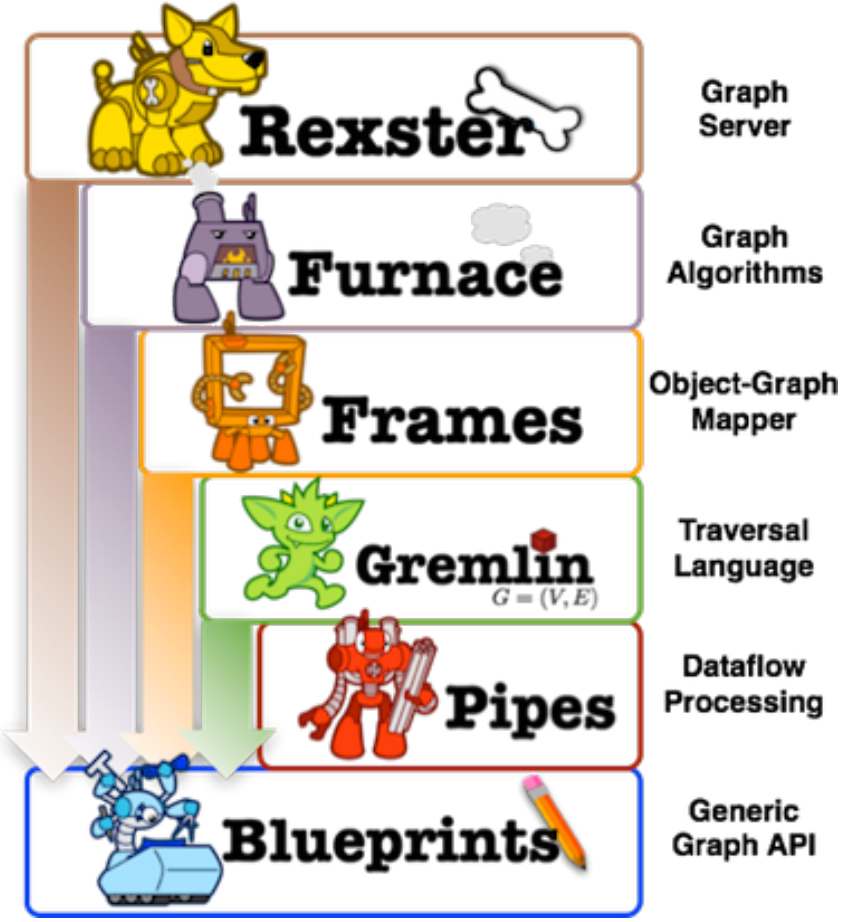
Design II: Node-Edge Graphs

- Benefit: a format that would facilitate easy, advanced graph computation by applications, using existing software tools

Example: using NetworkX

```
>>> import networkx as nx
>>> g=nx.Graph()
>>> g.add_edge('a','b',routingcost=1 )
>>> g.add_edge('b','c', routingcost=15)
>>> print(g.adj)
{'a': {'b': {'routingcost':1}}, 'c': {'b': {...}},
'b': {'a': {...}, 'c': {...}}}
>>> g.add_edge('a','c', routingcost=10)
>>> g.add_edge('c','d', routingcost=22)
>>> print(nx.shortest_path(g,'b','d')
['b', 'c', 'd']
>>> print(nx.shortest_path(g,'b','d',
'routingcost')
['b', 'a', 'c', 'd']
```





Two Usage Models of ALTO Node-Edge Graphs

- A client with only node-edge graph
  - The client may not have access to all network policies
  - Hence, a client may compute only *approximate* paths as guidance in usage

- A client with both node-edge graph and PV
  - The PV size may be substantially reduced if the result is the same as standard alg. on the graph

Initial ALTO Node-Edge Graph Design: Requirements

- Be compatible with and reuse existing ALTO information resources such as Network Maps
- Be simple, yet extensible/general to encode diverse network graphs to applications
- Be modular

Initial ALTO Node-Edge Graph Design: Nodes

- Defined in Network Map; PID properties provide details

```
HTTP/1.1 200 OK
Content-Length: TBD
Content-Type: application/alto-networkmap+json

{
  "meta": {
    ...
  },
  "network-map": {
    "H1": { "ipv4": [ "10.0.1.0/24" ] },
    "H2": { "ipv4": [ "10.0.2.0/24" ] },
    "H3": { "ipv4": [ "10.0.3.0/24" ] },
    "H4": { "ipv4": [ "10.0.4.0/24" ] },
    ”sw1": { }, ”sw2": { }, ”sw3": { }, ”sw4": { }, ”sw5": { }, ”sw6": { }, ”sw7": { },
    "Default": {
      "ipv4": [ "0.0.0.0/0" ], "ipv6": [ "::/0" ]
    }
  }
```

Initial ALTO Node-Edge Graph Design: Edges

- Defined in a new InfoResource named Edge Map

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

{  "meta" : {
     "dependent-vtags" : [
      { "resource-id": "my-default-network-map",
         "tag": "3ee2cb7e8d63d9fab71b9b34cbf764436315542e" },
      {"resource-id": "my-edge-map",
       "tag": "4xee2cb7e8d63d9fab71b9b34cbf76443631554de"
      }],
      "edge-default" : "undirected", "edge-label-default" : "connect",
    }
     "edge-map" : {
        "e1" : {"source" : "PID1", "target": "sw1", "cost": {}}
        …
     }
}
```

single-relational graph as default, allows multi-relational in future

Next Step

- Evaluate current design on extensibility, e.g., supporting additional requirements:
  - Hierarchy, hyperedges, port, dynamics, selector (filtering) API

- Target: a complete version by IETF 91 for WG review.

Backup Slides: Survey of Graph Tools/Formats

GraphViz

- A format used in graph visualization
- Example

```
graph hello2 {

  // Hello World with nice colors and big fonts

  Node1 [label="Hello, World!", color=Blue, fontcolor=Red,
     fontsize=24, shape=box]


  B [label="The boss"]      // node B
  E [label="The employee"]  // node E

  B->E [label="commands", dir=back, fontcolor=red]
  // revert arrow direction
}
```

NetworkX

- NetworkX uses a “dictionary of dictionaries of dictionaries” as the basic network graph data structure
  - The keys are nodes so G[u] returns an adjacency dictionary keyed by neighbor to the edge attribute dictionary.
- **Example: an undirected graph with the edges (‘A’,’B’), (‘B’,’C’)**
  - **>>> G=nx.Graph()**
  - **>>> G.add_edge(’A’,’B’)**
  - **>>> G.add_edge(’B’,’C’)**
  - **>>> print(G.adj)**
  - **{’A’: {’B’: {}}, ’C’: {’B’: {}}, ’B’: {’A’: {}, ’C’: {}}}**

- The format allows fast lookup with reasonable storage for large sparse networks.
- Problem if use in ALTO: unrestricted edge properties; no node properties.

GEXF

- Nodes and edges can have attributes, but must be declared
- Example

```
<gexf xmlns="http://www.gexf.net/1.2draft" xmlns:xsi="http://www.w3.org/2001/
XMLSchema-instance" xsi:schemaLocation="http://www.gexf.net/1.2draft http://
www.gexf.net/1.2draft/gexf.xsd" version="1.2">
  <meta lastmodifieddate="2009-03-20">
    <creator>Gephi.org</creator>
    <description>A Web network</description>
  </meta>
  <graph defaultedgetype="directed">
    <attributes class="node">
      <attribute id="0" title="url" type="string"/>
      <attribute id="1" title="indegree" type="float"/>
      <attribute id="2" title="frog" type="boolean">
        <default>true</default>
      </attribute>
    </attributes>
    SEE NEXT SLIDES
  </graph>
</gexf>
```

GEXF: Example

```
<nodes>
        <node id="0" label="Gephi">
          <attvalues>
            <attvalue for="0" value="http://gephi.org"/> <attvalue for="1" value="1"/>
          </attvalues>
        </node>
        <node id="1" label="Webatlas">
          <attvalues>
            <attvalue for="0" value="http://webatlas.fr"/> <attvalue for="1" value="2"/>
          </attvalues>
        </node>
        <node id="2" label="RTGI">
          <attvalues>
            <attvalue for="0" value="http://rtgi.fr"/> <attvalue for="1" value="1"/>
          </attvalues>
        </node>
        <node id="3" label="BarabasiLab">
          <attvalues>
            <attvalue for="0" value="http://barabasilab.com"/> <attvalue for="1" value="1"/>
            <attvalue for="2" value="false"/>
          </attvalues>
        </node>
</nodes>
```

GEXF: Example

```
<edges>
        <edge id="0" source="0" target="1"/>
        <edge id="1" source="0" target="2"/>
        <edge id="2" source="1" target="0"/>
        <edge id="3" source="2" target="1"/>
        <edge id="4" source="0" target="3"/>
 </edges>
```

GEXF: Advanced Features

- Dynamics
  - Each node/edge/data can have a lifetime, e.g.,
  - <edge id="1" source="0" target="2" start="2009-03-01" end="2009-03-10"/>

- Hierarchy
  - Allow nested definition of nodes
  - Or specify parent id (can have multiple parent ids forming polygeny)

GraphML

```
<?xml version="1.0" encoding="UTF-8"?>
<graphml xmlns="http://graphml.graphdrawing.org/xmlns"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://graphml.graphdrawing.org/xmlns
     http://graphml.graphdrawing.org/xmlns/1.0/graphml.xsd">
 <key id="d0" for="node" attr.name="color" attr.type="string">
  <default>yellow</default>
 </key>
 <key id="d1" for="edge" attr.name="weight”
                                attr.type="double"/>
 <graph id="G" edgedefault="undirected”>
   SEE NEXT SLIDES
 </graph>
</graphml>
```

GraphML: Example

```
<node id="n0">
    <data key="d0">green</data>
</node>
<node id="n1"/>
<node id="n2">
    <data key="d0">blue</data>
</node>
<node id="n3">
    <data key="d0">red</data>
</node>
<node id="n4"/>
<node id="n5">
    <data key="d0">turquoise</data>
</node>
```

GraphML: Example

```
<edge id="e0" source="n0" target="n2">
    <data key="d1">1.0</data>
</edge>
<edge id="e1" source="n0" target="n1">
    <data key="d1">1.0</data>
</edge>
<edge id="e2" source="n1" target="n3">
    <data key="d1">2.0</data>
</edge>
<edge id="e3" source="n3" target="n2"/>
<edge id="e4" source="n2" target="n4"/>
<edge id="e5" source="n3" target="n5"/>
<edge id="e6" source="n5" target="n4">
    <data key="d1">1.1</data>
</edge>
```

GraphML: Advanced features

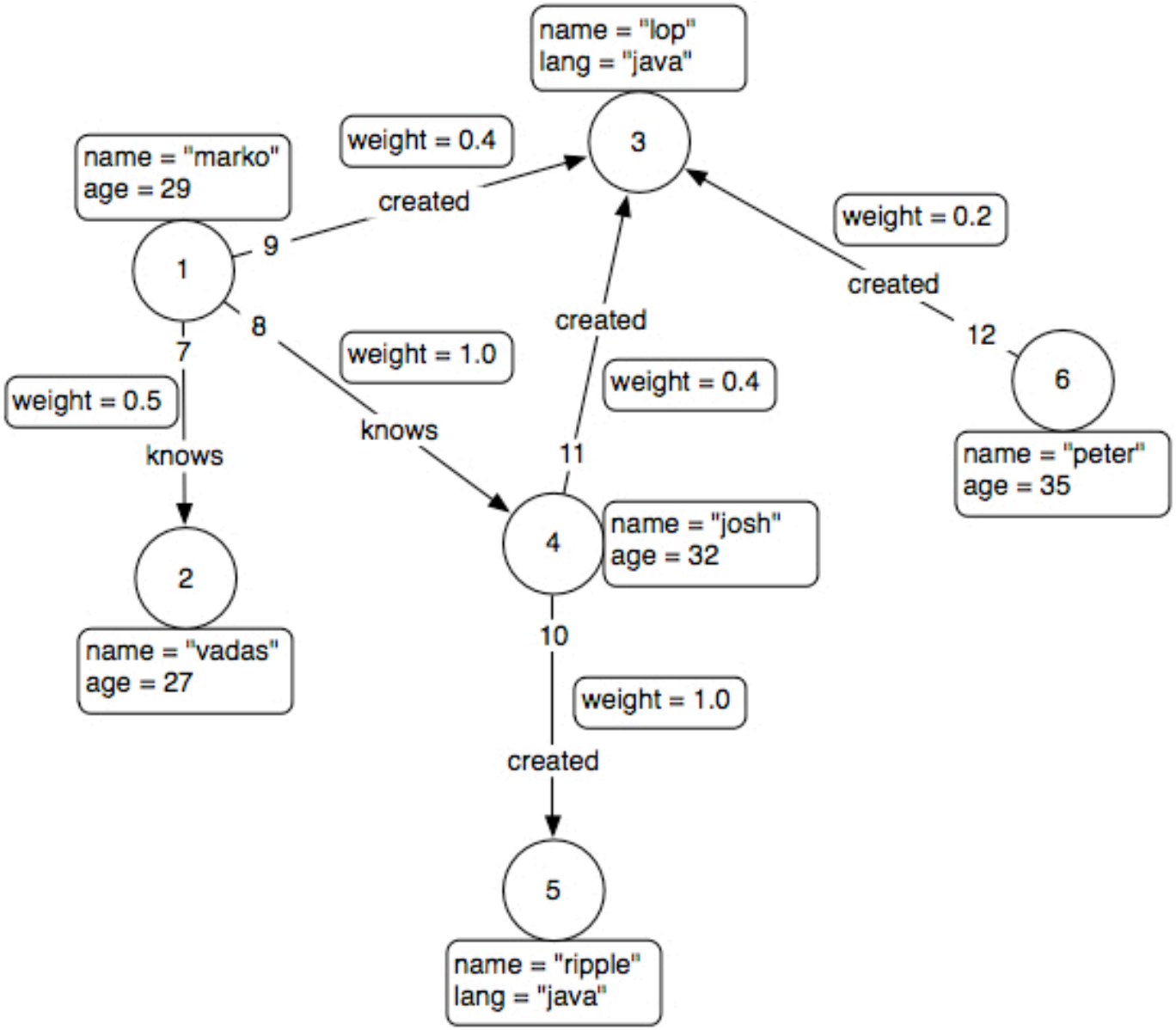
- Nested graphs
- Hyperedges
- Port

Blueprints

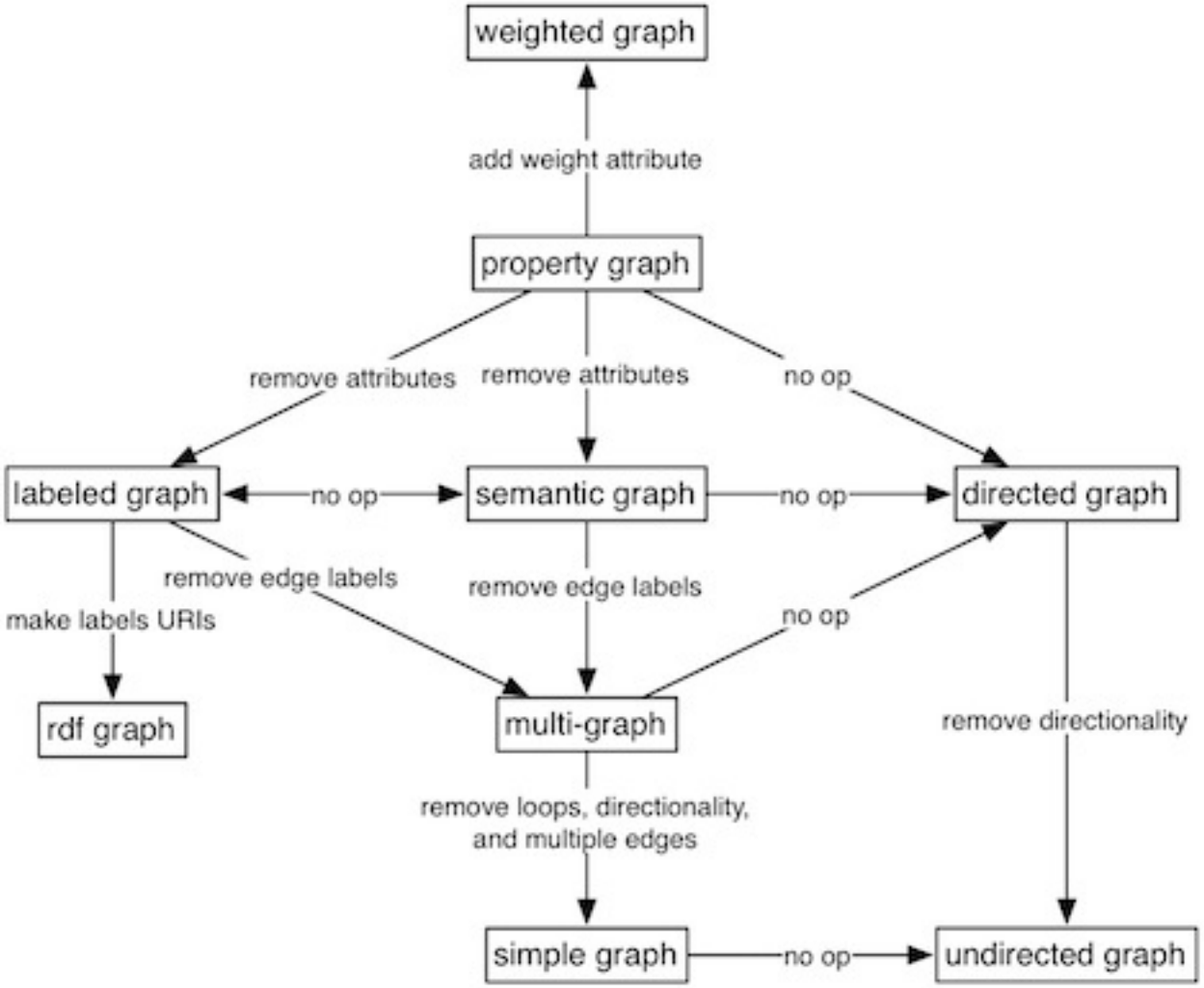
- Excellent review:
  - http://www.slideshare.net/slidarko/gremlin-a-graphbased-programming-language-3876581
- A good classification of graph models
  - https://github.com/tinkerpop/blueprints/wiki/Graph-Morphisms
- Defined for property graphs
- A property graph is an object that contains elements
  - Element: An object that can have any number of key/value pairs associated with it (i.e. properties)
    - Vertex: each vertex has a unique identifier.
      - each vertex has a set of outgoing edges.
      - each vertex has a set of incoming edges.
      - [each vertex has a collection of properties defined by a map from key to value]
    - Edge
      - each edge has a unique identifier.
      - each edge has an outgoing tail vertex.
      - each edge has an incoming head vertex.
      - each edge has a label that denotes the type of relationship between its two vertices.
      - [each edge has a collection of properties defined by a map from key to value.]

Gremlin

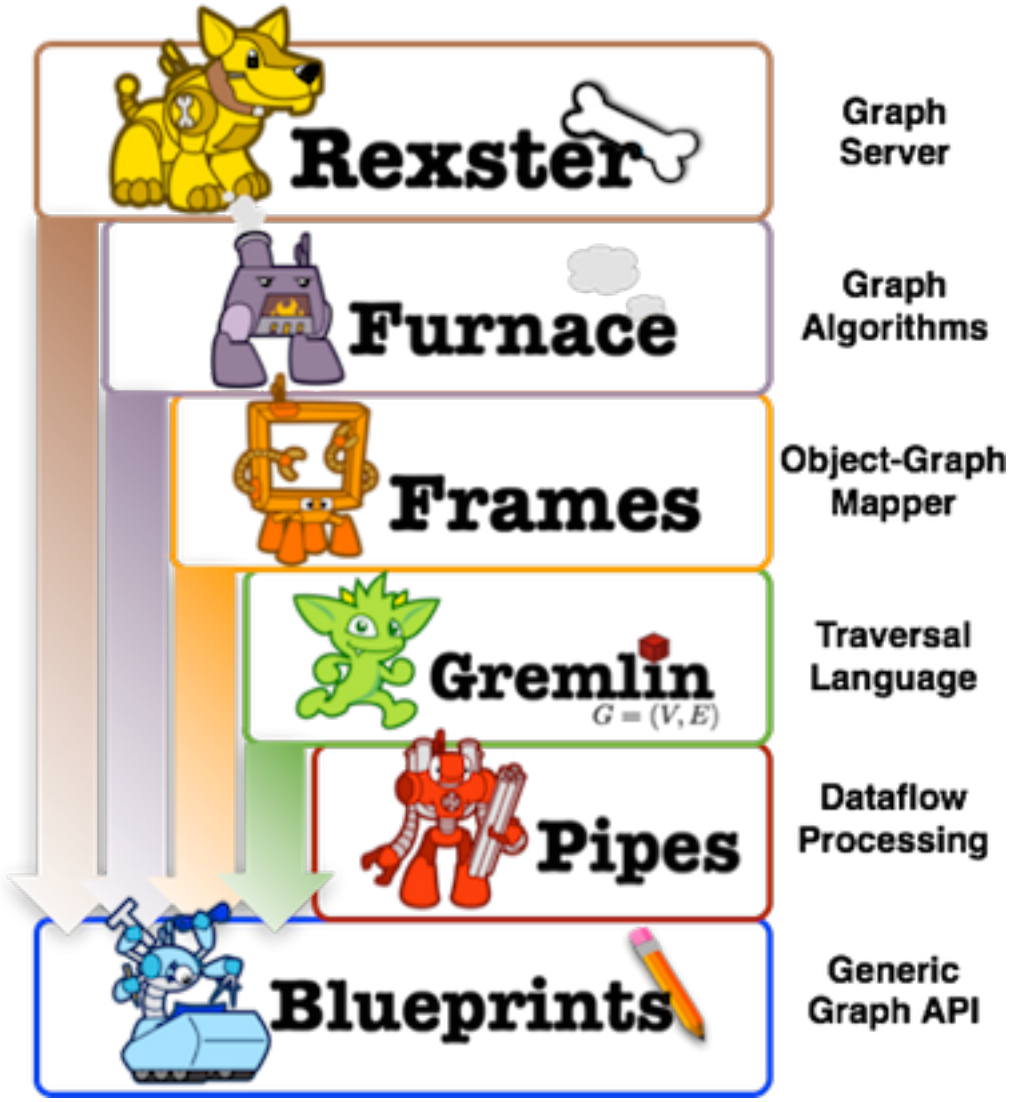
- port



Blueprint

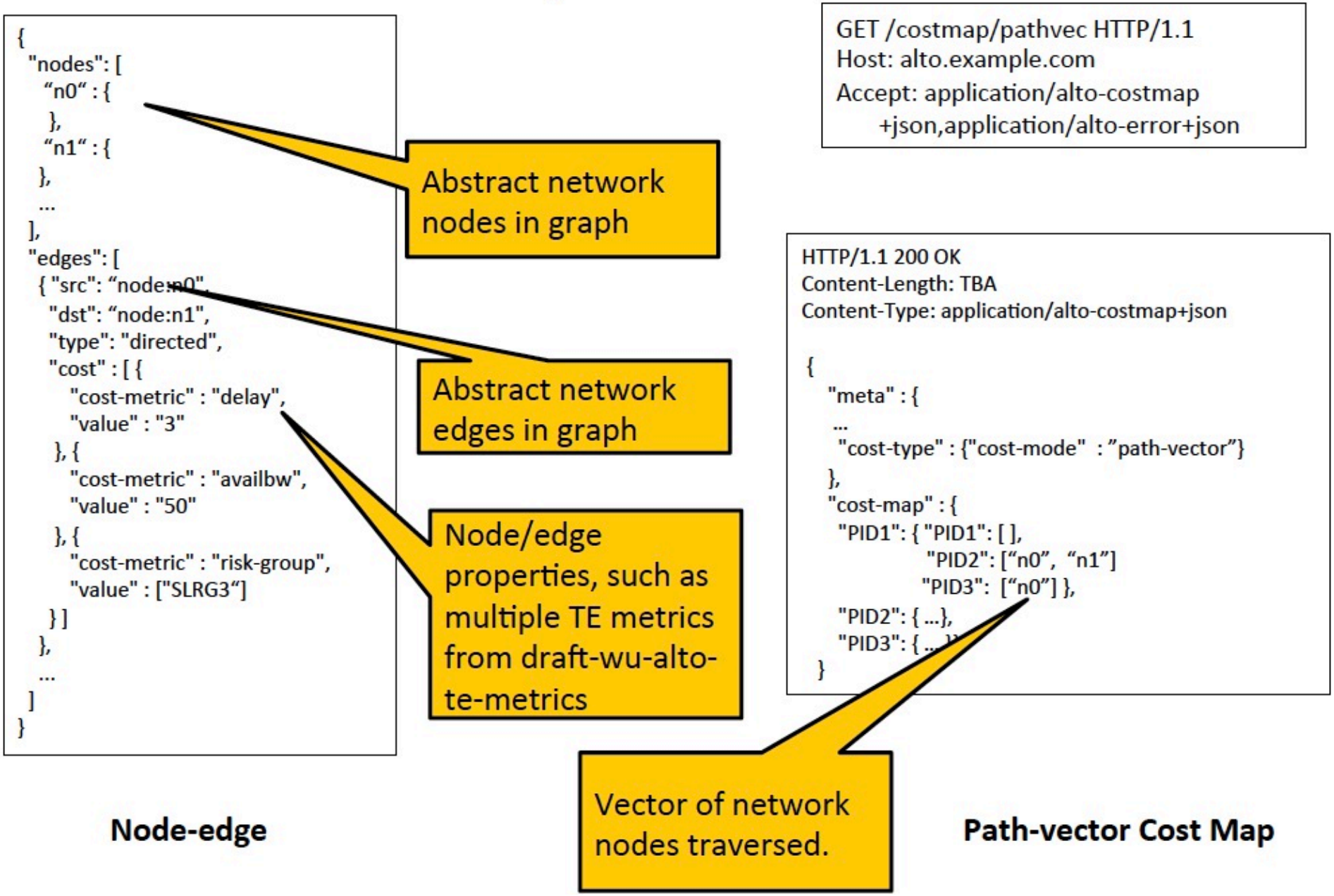

weighted graph
add weight attribute
property graph
remove attributes
remove attributes
no op
labeled graph
no op
semantic graph
no op
directed graph
remove edge labels
remove edge labels
make labels URIs
no op
rdf graph
multi-graph
remove directionality
remove loops, directionality,
and multiple edges
simple graph
no op
undirected graph

Gremlin Framework



Related i2rs Efforts

- http://tools.ietf.org/id/draft-medved-i2rs-topology-requirements-00.txt
  - Abstraction, hierarchy, tracable, filtering, push/subscription
- http://tools.ietf.org/id/draft-medved-i2rs-topology-im-01.txt


{
"nodes": [
"n0" : {
},
"n1" : {
},
...
],
"edges": [
{ "src": "node:n0",
"dst": "node:n1",
"type": "directed",
"cost" : [{
"cost-metric" : "delay",
"value" : "3"
}, {
"cost-metric" : "availbw",
"value" : "50"
}, {
"cost-metric" : "risk-group",
"value" : ["SLRG3"]
}]
},
...
]
}
Abstract network nodes in graph
Abstract network edges in graph
Node/edge properties, such as multiple TE metrics from draft-wu-alto-te-metrics
Node-edge
GET /costmap/pathvec HTTP/1.1
Host: alto.example.com
Accept: application/alto-costmap
+json,application/alto-error+json
HTTP/1.1 200 OK
Content-Length: TBA
Content-Type: application/alto-costmap+json
{
"meta" : {
...
"cost-type" : {"cost-mode" : "path-vector"}
},
"cost-map" : {
"PID1": { "PID1": [ ],
"PID2": ["n0", "n1"]
"PID3": ["n0"] },
"PID2": { ...},
"PID3": { ...}
}
Vector of network nodes traversed.
Path-vector Cost Map