Computer Science is Largely about Abstractions

Instances → Capturing (almost) all instances

- Parsers
- Data Management Systems
- Routing Protocols

- Yacc
- SQL-based systems + application code
- Metarouting (??)
Why do this?

- No one-size-fits-all IGP
  - BGP is now a widely used IGP!
- Hard to define, standardize, and deploy new routing protocols (or minor modifications to existing protocols)
  - Just standardize Metarouting language and leave it up to operator community to standardize protocols using high-level specs...
- It’s fun!
Idea #1

Let's try something radical -- keep these separate!

Protocol = Mechanism + Policy + .... ??? ...

- How are routing messages exchanged and propagated? (example: Link-State, Path Vector)
- How are adjacencies established?
- ...

- How are the attributes of a route described?
- How does configuration attach path characteristics?
- How are best routes selected?
- ...

- ...
Idea #2

- Use “Routing Policy Algebras” as basis for Policy Component
Routing Policy Algebras

How paths are transformed by application of labels

\[ L : \text{link labels} \quad \oplus : L \times \Sigma \rightarrow \Sigma \]

\[ \mathbf{A} = (\Sigma, \leq, L, \oplus, O) \]

How paths are described and compared

\[ \Sigma : \text{path signatures} \]

\( \leq \) is a preference relation over \( \Sigma \):

- (complete) \( \forall x, y \in \Sigma, x \leq y \) or \( y \leq x \) (or both)
- (transitive) \( \forall x, y, z \in \Sigma, \text{if } x \leq y \) and \( y \leq z \) then \( x \leq z \)

A subset of signatures that can be associated with originated routes.
Example --- Addition (ADD)

<table>
<thead>
<tr>
<th>L</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>(\phi)</th>
<th>(\phi)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>(\phi)</td>
<td>(\phi)</td>
<td>(\phi)</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>(\phi)</td>
<td>(\phi)</td>
<td>(\phi)</td>
<td>(\phi)</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>(\phi)</td>
<td>(\phi)</td>
<td>(\phi)</td>
<td>(\phi)</td>
<td>(\phi)</td>
</tr>
</tbody>
</table>

**max label**  ADD(3, 6)  **max signature**

ADD\((n, m)\) is SM if \(0 < n \leq m\)
Guarantees?

We want protocols that are nice!

→ always converge, for every network state

→ unique solution (perhaps modulo some $\equiv$ class)

→ no forwarding loops (after convergence)
Correctness

Monotonicity (M): \( \forall \sigma \in \Sigma/\phi, \lambda \in L \quad \sigma \leq \lambda \oplus \sigma \)

Strict Monotonicity (SM): \( \forall \sigma \in \Sigma/\phi, \lambda \in L \quad \sigma < \lambda \oplus \sigma \)

Isotonicity (I): \( \forall \sigma, \beta \in \Sigma/\phi, \lambda \in L \quad \sigma \leq \beta \rightarrow \lambda \oplus \sigma \leq \lambda \oplus \beta \)
An algebra for OSPF?

(hand-coded from careful reading of RFC 2328)

<table>
<thead>
<tr>
<th></th>
<th>(\varepsilon)</th>
<th>((1, \varepsilon, \sigma))</th>
<th>((1, (1, v), \sigma))</th>
<th>((1, (2, v), \sigma))</th>
<th>((2, \varepsilon, \sigma))</th>
<th>((2, (1, v), \sigma))</th>
<th>((2, (2, v), \sigma))</th>
</tr>
</thead>
<tbody>
<tr>
<td>((1, \lambda))</td>
<td>((1, \varepsilon, \lambda \oplus \varepsilon))</td>
<td>((1, \varepsilon, \lambda \oplus \sigma))</td>
<td>((1, (1, v), \lambda \oplus \sigma))</td>
<td>((1, (2, v), \lambda \oplus \sigma))</td>
<td>((2, \varepsilon, \lambda \oplus \sigma))</td>
<td>((2, (1, v), \lambda \oplus \sigma))</td>
<td>((2, (2, v), \lambda \oplus \sigma))</td>
</tr>
<tr>
<td>((1, (1, v), \lambda))</td>
<td>((1, (1, v), \lambda \oplus \varepsilon))</td>
<td>(\phi)</td>
<td>(\phi)</td>
<td>(\phi)</td>
<td>(\phi)</td>
<td>(\phi)</td>
<td>(\phi)</td>
</tr>
<tr>
<td>((1, (2, v), \lambda))</td>
<td>((1, (2, v), \lambda \oplus \varepsilon))</td>
<td>(\phi)</td>
<td>(\phi)</td>
<td>(\phi)</td>
<td>(\phi)</td>
<td>(\phi)</td>
<td>(\phi)</td>
</tr>
<tr>
<td>((2, \lambda))</td>
<td>((2, \varepsilon, \lambda \oplus \varepsilon))</td>
<td>((2, \varepsilon, \lambda \oplus \sigma))</td>
<td>((2, (1, v), \lambda \oplus \sigma))</td>
<td>((2, (2, v), \lambda \oplus \sigma))</td>
<td>(\phi)</td>
<td>(\phi)</td>
<td>(\phi)</td>
</tr>
<tr>
<td>((2, (1, v), \lambda))</td>
<td>((2, (1, v), \lambda \oplus \varepsilon))</td>
<td>(\phi)</td>
<td>(\phi)</td>
<td>(\phi)</td>
<td>(\phi)</td>
<td>(\phi)</td>
<td>(\phi)</td>
</tr>
<tr>
<td>((2, (2, v), \lambda))</td>
<td>((2, (2, v), \lambda \oplus \varepsilon))</td>
<td>(\phi)</td>
<td>(\phi)</td>
<td>(\phi)</td>
<td>(\phi)</td>
<td>(\phi)</td>
<td>(\phi)</td>
</tr>
</tbody>
</table>

\(<1, \ldots> = \text{intra-area route} \quad <2, \ldots> = \text{inter-area route} \quad <\{1,2\}, \lambda> = \text{“normal” route} \quad <\{1,2\}, <1, v>, \lambda> = \text{type I external} \quad <\{1,2\}, <2, v>, \lambda> = \text{type II external}
Routing Algebras are a good start, but...

- The algebraic framework does not, by itself, provide a way of constructing new and complex algebras.
  - Algebra definition is hard...
  - Proofs are tedious...
  - Modifications to an algebra’s definitions are difficult to manage...
Idea #3
Routing Algebra Meta-Language (RAML)

A ::= B  
| Op(A)  
| A Op A

• “Abstract syntax” for generating new Algebras
• Goals
  – Want to automatically derive properties (M, SM, ...) of the algebra represented by an RAML expression from properties of base algebras and preservation properties of operators
  – Simplicity
  – Expressiveness
**Lexical Product**

\[ \begin{array}{c|c}
A & \otimes & B \\
\hline
\oplus & (\sigma_1, \sigma_2) \\
(\lambda_1, \lambda_2) & (\lambda_1 \oplus \sigma_1, \lambda_2 \oplus \sigma_2) \\
(\phi, _) & (_, \phi) = \phi \\
\end{array} \]

Preference is Lexical order

Preservation properties

\[
\begin{array}{c|c|c|c}
A & B & A \otimes B \\
\hline
M & M & M \\
SM & SM & SM \\
M & SM & SM \\
\end{array}
\]

This suggests a design pattern for SM:

\[
A_1 \otimes A_2 \otimes \ldots \otimes A_i \otimes A(i+1) \otimes \ldots \otimes A_n
\]

all M  SM  don’t care

SM
Point-wise application?

A ★ B

<table>
<thead>
<tr>
<th>0</th>
<th>(σ₁, σ₂)</th>
</tr>
</thead>
<tbody>
<tr>
<td>λ₁</td>
<td>(λ₁ ⊕ σ₁, σ₂)</td>
</tr>
<tr>
<td>λ₂</td>
<td>(σ₁, λ₂ ⊕ σ₂)</td>
</tr>
</tbody>
</table>

Preservation properties

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>A ★ B</th>
</tr>
</thead>
<tbody>
<tr>
<td>SM</td>
<td>SM</td>
<td>M</td>
</tr>
<tr>
<td>SM</td>
<td>M</td>
<td>M</td>
</tr>
<tr>
<td>M</td>
<td>SM</td>
<td>M</td>
</tr>
<tr>
<td>M</td>
<td>M</td>
<td>M</td>
</tr>
</tbody>
</table>

κ(A)

<table>
<thead>
<tr>
<th>0</th>
<th>σ₁</th>
</tr>
</thead>
<tbody>
<tr>
<td>λ</td>
<td>λ₁ ⊕ σ₁</td>
</tr>
<tr>
<td>κ</td>
<td>σ₁</td>
</tr>
</tbody>
</table>

A ★ B = κ(A) ⊗ κ(B)
Scoped Product

\[
\begin{array}{c|c}
\oplus & (\sigma_1, \sigma_2) \\
(\lambda_1, \sigma_3) & (\lambda_1 \oplus \sigma_1, \sigma_3) \\
\lambda_2 & (\sigma_1, \lambda_2 \oplus \sigma_2) \\
\end{array}
\]

Preservation properties

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>A \ominus B</th>
</tr>
</thead>
<tbody>
<tr>
<td>SM</td>
<td>SM</td>
<td>SM</td>
<td>SM</td>
</tr>
<tr>
<td>SM</td>
<td>M</td>
<td>M</td>
<td>M</td>
</tr>
</tbody>
</table>

Can be used to implement IBGP/EBGP-like “information hiding”
Programmatic Labels

\[ A = (\Sigma, 8, L, \oplus, \theta) \]

\[ \text{prog}(A) = (\Sigma, 8, L, \oplus, \theta) \]

\[ \lambda ::= \lambda \mid \lambda_1; \lambda_2 \mid \text{reject} \mid \text{if } \pi \text{ then } \lambda_1 \text{ else } \lambda_2 \]

\[ \lambda \oplus \sigma = \lambda \oplus \sigma \]

\[ (\lambda_1; \lambda_2) \oplus \sigma = \lambda_1 \oplus (\lambda_2 \oplus \sigma) \]

\[ \text{reject } \oplus \sigma = \phi \]

\[ (\text{if } \pi \text{ then } \lambda_1 \text{ else } \lambda_2) \oplus \sigma = \begin{cases} 
\lambda_1 \oplus \sigma & \text{if } \pi(\sigma) \\
\lambda_2 \oplus \sigma & \text{o.w.}
\end{cases} \]
MyFirstIGP

    router-path: SimpleSeq(1000, 30)
    bandwidth: WIDTH(10000)
    tags: TAGS(string[100]))

This is SM
policy my-import {
    metapolicy MyIGP;
    import [set link-bandwidth OC-12; my-import];
    export my-export;
}

policy my-export {
    metapolicy MyIGP;
    if (empty(router-path)) {
        insert tags "sales center NE"
    }
}

BGP-17 {
    router-id 10.20.20.11;
    mechanism link-state hard-state
    neighbor 10.10.10.10 {
        metapolicy MyIGP;
        import [set link-bandwidth OC-12; my-import];
        export my-export;
    }
}
A Metarouting Specification,

\[ \text{RP} = \langle E, M, \text{LM} \rangle \]

An RAML expression

A set of mechanisms that can be bound to adjacencies

Label Modalities
Ongoing, Future Work

• RAML
  – More operators...
    • At the “protocol level”
    • Inter-operation operators

• Implementation
  – Using XORP ([www.xorp.org](http://www.xorp.org))
    • With Mark Handley (UCL) and others
  – Hijack BGP
  – Routing Metaprotocol
## Disjunction ("Injection")

<table>
<thead>
<tr>
<th></th>
<th>A → B</th>
<th>Preservation properties</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>⊕</td>
<td>σ1</td>
<td>σ2</td>
</tr>
<tr>
<td>λ1</td>
<td>λ1 ⊕ σ1</td>
<td>φ</td>
</tr>
<tr>
<td>λ2</td>
<td>φ</td>
<td>λ2 ⊕ σ2</td>
</tr>
<tr>
<td>i</td>
<td>τ(σ)</td>
<td>φ</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>A → B</th>
</tr>
</thead>
<tbody>
<tr>
<td>SM</td>
<td>SM</td>
<td>SM</td>
</tr>
<tr>
<td>SM</td>
<td>M</td>
<td>M</td>
</tr>
<tr>
<td>M</td>
<td>SM</td>
<td>M</td>
</tr>
<tr>
<td>M</td>
<td>M</td>
<td>M</td>
</tr>
</tbody>
</table>

Preservation properties:
- SM: Small Model
- M: Medium
- SM: Small Model
... at the “protocol level”? 

\[ \langle A, M_1, LM_1 \rangle \xrightarrow{\tau} \langle B, M_2, LM_2 \rangle = \langle A \xleftarrow{\tau} B, M_1 + M_2, LM_1 + LM_2 \rangle \]

Not sure what “+” means

Nice way of thinking about “administrative distance” …

Perhaps OSPF is really something like

\[ \langle \text{AREAS, Path-Vector, } LM_1 \rangle \varnothing \langle \text{ADD, Link-State, } LM_2 \rangle \]
Migration operators