Declarative Routing:
Extensible Routing with Declarative Queries

Boon Thau Loo

Joseph M. Hellerstein\textsuperscript{1,2}, Ion Stoica\textsuperscript{1}, Raghu Ramakrishnan\textsuperscript{3},
\textsuperscript{1}University of California at Berkeley, \textsuperscript{2}Intel Research Berkeley,
\textsuperscript{3}University of Wisconsin-Madison
Motivation

- Lack of extensibility and flexibility in today’s Internet routing
- Hard to add/improve/update routing protocols
Two Extremes:

“Hard-coded” protocols:

+ Efficiency, safety
- Flexibility, evolvability

Active Networks

+ Flexibility, evolvability
- Safety, efficiency

Declarative Routing:

+ Flexibility, evolvability, safety

Restricted instantiation of Active Networks for the control plane
Key Idea

Recursive query language for expressing routing protocols:

- Datalog: a declarative recursive query language
- Well-researched in the database community
- Well-suited for querying properties of graphs
Advantages

- Expressiveness: Compact and clean representation of protocols
- Safety: Datalog has desirable safety properties on termination
- Efficiency: No fundamental overhead when executing standard protocols.
Usage Scenarios

**ISP administrators**
- Run different protocols for different nodes
- Modify existing protocols in routers

**End-hosts**
- Set up customized routes for different quality-of-service and policy requirements of applications
Roadmap

- Execution Model
- Introduction to Datalog
- Path-Vector Protocol Example
- Advantages:
  - Expressiveness
  - Safety
  - Efficiency
- Evaluation
Centralized Execution Model

- Store entire network state into a centralized database
- Issue Datalog queries on the centralized database for customized routes
Distributed Execution Model

Routing Protocol

Neighbor Table updates
Forwarding Table updates

Neighbor Table
Forwarding Table
Routing Infrastructure Node

Datalog Queries

Routing Infrastructure

Declarative Routing
Introduction to Datalog

Datalog rule syntax:

\[ \langle \text{head} \rangle \leftarrow \langle \text{precondition}_1 \rangle, \langle \text{precondition}_2 \rangle, \ldots, \langle \text{precondition}_N \rangle. \]
All-Pairs Reachability

R1: reachable(S,D) ← link(S,D)
R2: reachable(S,D) ← link(S,Z), reachable(Z,D)

\textbf{link}(a,b) – “there is a link from node \textit{a} to node \textit{b}”
\textbf{reachable}(a,b) – “node \textit{a} can reach node \textit{b}”

“For all nodes S,D,
If there is a link from S to D, then S can reach D”.

◆ Input: link(source, destination)
◆ Output: reachable(source, destination)
All-Pairs Reachability

R1: reachable(S,D) ← link(S,D)

R2: reachable(S,D) ← link(S,Z), reachable(Z,D)

“For all S, D and Z,
If link(S,Z) exists AND reachable(Z,D) exists, generate reachable(S,D).”

“For all nodes S, D and Z,
If there is a link from S to Z, AND Z can reach D, then S can reach D”.

◆ Input: link(source, destination)
◆ Output: reachable(source, destination)
All-Pairs Reachability

R1: \( \text{reachable}(S,D) \leftarrow \text{link}(S,D) \)

R2: \( \text{reachable}(S,D) \leftarrow \text{link}(S,Z), \text{reachable}(Z,D) \)

Query: \( \text{reachable}(M,N) \)

E.g. R1: \( \text{reachable}(b,c) \leftarrow \text{link}(b,c) \)
All-Pairs Reachability

R1: reachable(S,D) ← link(S,D)

R2: reachable(S,D) ← link(S,Z), reachable(Z,D)

Query: reachable(M,N)

Input table:

<table>
<thead>
<tr>
<th>S</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b</td>
</tr>
<tr>
<td>b</td>
<td>c</td>
</tr>
<tr>
<td>c</td>
<td>d</td>
</tr>
</tbody>
</table>

Output table (Round 2):

<table>
<thead>
<tr>
<th>S</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b</td>
</tr>
<tr>
<td>b</td>
<td>c</td>
</tr>
<tr>
<td>c</td>
<td>d</td>
</tr>
</tbody>
</table>

R2: reachable(b,d) ← link(b,c), reachable(c,d)
All-Pairs Reachability

R1: reachable(S,D) ← link(S,D)

R2: reachable(S,D) ← link(S,Z), reachable(Z,D)

Query: reachable(M,N)

Recursive queries are natural for querying graph topologies
Roadmap

- Execution Model
- Introduction to Datalog
- Path-Vector Protocol Example
  - Distributed Datalog ➔ Execution Plan ➔ Protocol

- Advantages:
  - Expressiveness
  - Safety
  - Efficiency

- Evaluation
Distributed Datalog

R1: \( \text{reachable}(S,D) \leftarrow \text{link}(S,D) \)

R2: \( \text{reachable}(S,D) \leftarrow \text{link}(S,D), \text{reachable}(Z,D) \)

Query: \( \text{reachable}(M,N) \)

Input table:

\[
\begin{array}{cc}
S & D \\
\hline
a & b \\
\end{array}
\]

\[
\begin{array}{cc}
S & D \\
\hline
b & c \\
\end{array}
\]

\[
\begin{array}{cc}
S & D \\
\hline
c & d \\
\end{array}
\]

Output table:

\[
\begin{array}{cc}
S & D \\
\hline
a & b \\
\end{array}
\]

\[
\begin{array}{cc}
S & D \\
\hline
a & d \\
\end{array}
\]

\[
\begin{array}{cc}
S & D \\
\hline
b & d \\
\end{array}
\]

\[
\begin{array}{cc}
S & D \\
\hline
b & d \\
\end{array}
\]

\[
\begin{array}{cc}
S & D \\
\hline
b & d \\
\end{array}
\]
Path Vector Protocol Example

R1: \( \text{path}(S,D,P) \leftarrow \text{link}(S,D), \ P=(S,D). \)

R2: \( \text{path}(S,D,P) \leftarrow \text{link}(Z,S), \ \text{path}(Z,D,P_2), \ P=S+P_2. \)

Query: \( \text{path}(S,D,P) \)

- **Input:** \( \text{link}(\text{source}, \text{destination}) \)
- **Query output:** \( \text{path}(\text{source}, \text{destination}, \ \text{pathVector}) \)
R1: \( \text{path}(S,D,P) \leftarrow \text{link}(S,D), P=(S,D). \)

R2: \( \text{path}(S,D,P) \leftarrow \text{link}(Z,S), \text{path}(Z,D,P_2), P=S+P_2. \)

Matching variable \( Z = \) “Join”

**Recursion**

**Pseudocode at node \( Z \):**

```plaintext
while (receive<path(Z,D,P2)>)) {
  for each neighbor \( S \) {
    newpath = path(S,D,S+P2)
    newpath = path(S,D,S+P2)
  }
  send newpath to neighbor \( S \)
}
```
Query Execution

R1: \( \text{path}(S,D,P) \leftarrow \text{link}(S,D), P=(S,D) \).

R2: \( \text{path}(S,D,P) \leftarrow \text{link}(Z,S), \text{path}(Z,D,P_2), P=S+P_2 \).

Query: \( \text{path}(S,D,P,C) \)

Neighbor table:

<table>
<thead>
<tr>
<th></th>
<th>S</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>c</td>
<td></td>
</tr>
<tr>
<td>c</td>
<td>d</td>
<td></td>
</tr>
</tbody>
</table>

Forwarding table:

<table>
<thead>
<tr>
<th></th>
<th>S</th>
<th>D</th>
<th>P</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b</td>
<td></td>
<td>(a,b)</td>
</tr>
<tr>
<td>b</td>
<td>c</td>
<td></td>
<td>(b,c)</td>
</tr>
<tr>
<td>c</td>
<td>d</td>
<td></td>
<td>(c,d)</td>
</tr>
</tbody>
</table>
Query Execution

R1: \( \text{path}(S,D,P) \leftarrow \text{link}(S,D), \ P=(S,D) \).

R2: \( \text{path}(S,D,P) \leftarrow \text{link}(S,Z), \ \text{path}(Z,D,P_2), \ P=S+P_2 \).

Query: \( \text{path}(S,D,P,C) \)
Query Execution

R1: path(S,D,P) ← link(S,D), P=(S,D).

R2: path(S,D,P) ← link(S,Z), path(Z,D,P_2), P=S+P_2.

Query: path(S,D,P,C)

Neighbor table:

<table>
<thead>
<tr>
<th>Link</th>
<th>Neighbor table</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>a</td>
</tr>
<tr>
<td>D</td>
<td>b</td>
</tr>
<tr>
<td>S</td>
<td>b</td>
</tr>
<tr>
<td>D</td>
<td>c</td>
</tr>
<tr>
<td>S</td>
<td>c</td>
</tr>
<tr>
<td>D</td>
<td>d</td>
</tr>
</tbody>
</table>

Communication patterns are identical to those in the actual path vector protocol.
Roadmap

- Execution Model
- Introduction to Datalog
- Path-Vector Protocol Example
  - Distributed Datalog ➔ Execution Plan ➔ Protocol
- Advantages:
  - Expressiveness
  - Safety
  - Efficiency
- Evaluation
Expressiveness

- Best-Path Routing
- Distance Vector
- Dynamic Source Routing
- Policy Decisions
- QoS-based Routing
- Link-state
- Multicast Overlays (Single-Source & CBT)

Minor variants give many options!
Expressiveness

◆ All-pairs all-paths:

R1: path(S,D,P,C) ← link(S,D,C), P=(S,D).
R2: path(S,D,P,C) ← link(S,Z,C₁), path(Z,D,P₂,C₂), C=C₁+C₂, P=S+P₂.
Query: path(S,D,P,C)
Expressiveness

Best-Path Routing:

R1: \( \text{path}(S, D, P, C) \leftarrow \text{link}(S, D, C), P = (S, D) \).
R2: \( \text{path}(S, D, P, C) \leftarrow \text{link}(S, Z, C_1), \text{path}(Z, D, P_2, C_2), C = C_1 + C_2, P = S + P_2 \).
R3: \( \text{bestPathCost}(S, D, \min <C>) \leftarrow \text{path}(S, D, Z, C) \)
R4: \( \text{bestPath}(S, D, Z, C) \leftarrow \text{bestPathCost}(S, D, C), \text{path}(S, D, P, C) \)

Query: \( \text{bestPath}(S, D, P, C) \)
Expressiveness

Best-Path Routing:

R1: path(S, D, P, C) ← link(S, D, C), P = (S, D).
R2: path(S, D, P, C) ← link(S, Z, C₁), path(Z, D, P₂, C₂), C = FN(C₁, C₂), P = S + P₂.
R3: bestPathCost(S, D, AGG<C>) ← path(S, D, Z, C)
R4: bestPath(S, D, Z, C) ← bestPathCost(S, D, C), path(S, D, P, C)
Query: bestPath(S, D, P, C)

Customizing C, AGG and FN: lowest RTT, lowest loss rate, highest available bandwidth, best-k
Expressiveness

❖ All-pairs all-paths:

R1: \( \text{path}(S,D,P,C) \leftarrow \text{link}(S,D,C), P=(S,D) \).
R2: \( \text{path}(S,D,P,C) \leftarrow \text{link}(S,Z,C_1), \text{path}(Z,D,P_2,C_2), C=C_1+C_2, P=S+P_2 \).

Query: \( \text{path}(S,D,P,C) \)
Expressiveness

Distance Vector:

R1: path(S,D,D,C) ← link(S,D,C)
R2: path(S,D,Z,C) ← link(S,Z,C₁), path(Z,D,W,C₂), C=C₁+C₂
R3: shortestLength(S,D,min<C>) ← path(S,D,Z,C)
R4: nextHop(S,D,Z,C) ← nextHop(S,D,Z,C), shortestLength(S,D,C)
Query: nextHop(S,D,Z,C)

Count to Infinity problem?
Expressiveness

Distance Vector with Split Horizon:

R1: path(S,D,D,C) ← link(S,D,C)
R2: path(S,D,Z,C) ← link(S,Z,C₁), path(Z,D,W,C₂), C=C₁+C₂, W!=S
R3: shortestLength(S,D,min<C>) ← path(S,D,Z,C)
R4: nextHop(S,D,Z,C) ← nextHop(S,D,Z,C), shortestLength(S,D,C)
Query: nextHop(S,D,Z,C)
Expressiveness

Distance Vector with Poisoned Reverse:

R1: \( \text{path}(S,D,D,C) \leftarrow \text{link}(S,D,C) \)
R2: \( \text{path}(S,D,Z,C) \leftarrow \text{link}(S,Z,C_1), \text{path}(Z,D,W,C_2), C=C_1+C_2, W!=S \)
R3: \( \text{path}(S,D,Z,C) \leftarrow \text{link}(S,Z,C_1), \text{path}(Z,D,W,C_2), C=\infty, W=S \)
R4: \( \text{shortestLength}(S,D,\min<C>) \leftarrow \text{path}(S,D,Z,C) \)
R5: \( \text{nextHop}(S,D,Z,C) \leftarrow \text{nextHop}(S,D,Z,C), \text{shortestLength}(S,D,C) \)
Query: \( \text{nextHop}(S,D,Z,C) \)
Expressiveness

All-pairs all-paths:

R1: path(S,D,P,C) ← link(S,D,C), P= (S,D).
R2: path(S,D,P,C) ← link(S,Z,C_1), path(Z,D,P_2,C_2), C=C_1+C_2, P=S+P_2.
Query: path(S,D,P,C)
Expressiveness

Dynamic Source Routing (DSR):

R1: \( \text{path}(S,D,P,C) \leftarrow \text{link}(S,D,C) \), \( P = (S,D) \).
R2: \( \text{path}(S,D,P,C) \leftarrow \text{link}(Z,D,C_2) \), \( \text{path}(S,Z,P_1,C_1) \), \( C = C_1 + C_2 \),
    \( P = P_1 + D \).
Query: \( \text{path}(S,D,P,C) \)

Switching *Right-recursion* to *Left-recursion* execution
\( \rightarrow \) Path vector protocol to DSR.
Expressiveness

- Best-Path routing
- Distance Vector
- Dynamic Source Routing
- Policy-based routing
- QoS-based routing
- Link-state
- Multicast Overlays (Single-Source & CBT)
Safety

- Queries are sand-boxed within query engine
  - Queries use input tables to produce output tables
  - No side-effects on existing input tables
- Pure Datalog guarantees termination:
  - Natural bound on resource consumption of queries
- Static termination checks for our extended Datalog:
  - Identify recursive definitions and check for termination
  - E.g., monotonically increasing/decreasing cost fields whose values are upper/lower bounded
- Orthogonal security issues:
  - Denial-of-service attacks, compromised routers
Efficiency

- Explore well-known database techniques
  - Aggregate selections: avoid sending unnecessary paths to neighbors
  - Limit computation to portion of network
    - Few sources and destinations
    - Magic sets + left-right recursion rewrite
  - Multi-query sharing:
    - Identify “similar” queries, share their computations
    - Reuse previously computed paths
Queries under Churn

- Long-running continuous queries
- Maintain all intermediate derived tuples for query duration
- Incremental updates:
  - Link failures are treated as link updates with cost=\(\text{infinity}\).
  - Paths invalidated (cost=\(\text{infinity}\)), and new paths are incrementally recomputed.
Evaluation Setup

◆ PIER: Distributed relational query processor
  - Each node runs the query engine of PIER
  - Initialized neighbor table directly accessible by PIER.

◆ Simulation:
  - Bandwidth and latency bottlenecks
  - Transit-stub topologies

◆ PlanetLab
  - 72 PIER nodes
  - Random, location-aware topologies
  - Long-running queries
Summary of Results

◆ Simulations:
  ▪ When all nodes issue the same query,
    ◦ Scalability properties show similar trends as traditional DV/PV protocols
  ▪ When few nodes issue the same query,
    ◦ Overhead is reduced using standard query optimizations

◆ PlanetLab experiments:
  ▪ Long-running all-pairs shortest RTT paths query
  ▪ Ability to handle link RTT changes
  ▪ Induced failures (up to 20% of nodes)
    ◦ Recovery time: <1s (median), <3.6s (average)
Conclusion

❖ Declarative routing:
  ■ Express routing protocols using a recursive query language
  ■ Better balance between routing extensibility and safety

❖ Future work:
  ■ Expressing policies as declarative rules
  ■ Run-time query optimizations:
    ◆ Cost-based decisions on query rewrites
    ◆ Multi-query sharing
  ■ Static checker for routing protocols
  ■ Run-time monitoring of routing protocols

❖ Declarative Networks
  ■ Research agenda: Specify and construct networks declaratively
  ■ P2: “Implementing Declarative Overlays” (SOSP 2005)
Thank You