Declarative Routing:
Extensible Routing with Declarative Queries

Boon Thau Loo[1]

Joseph M. Hellerstein[1,2], Ion Stoica[1], Raghu Ramakrishnan[3],

[1]University of California at Berkeley, [2]Intel Research Berkeley,

[3]University of Wisconsin-Madison

Motivation

- Lack of extensibility and flexibility in today's Internet routing
- Hard to add/improve/update routing protocols

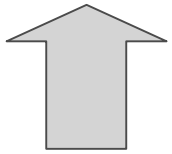
Two "Extremes":

"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

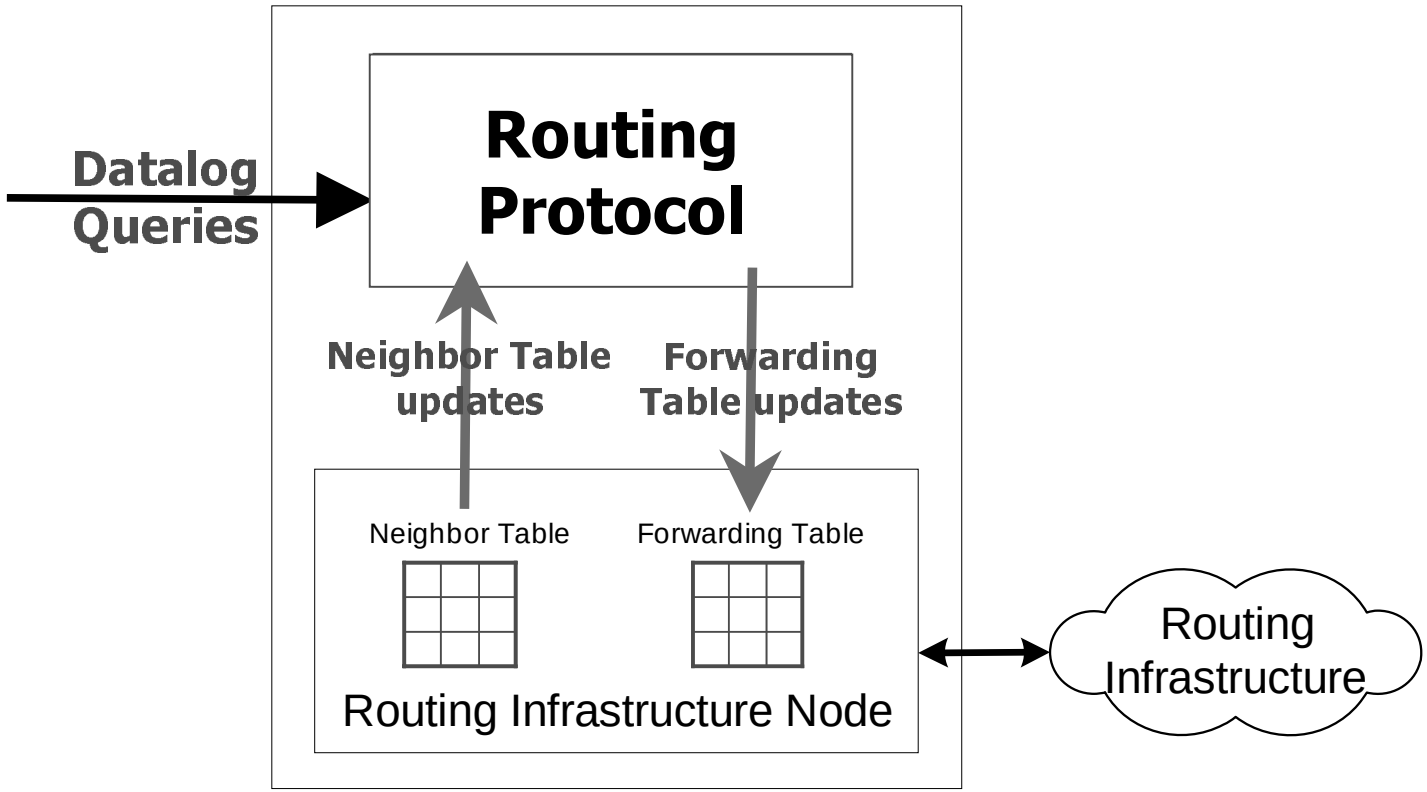
Roadmap

- 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

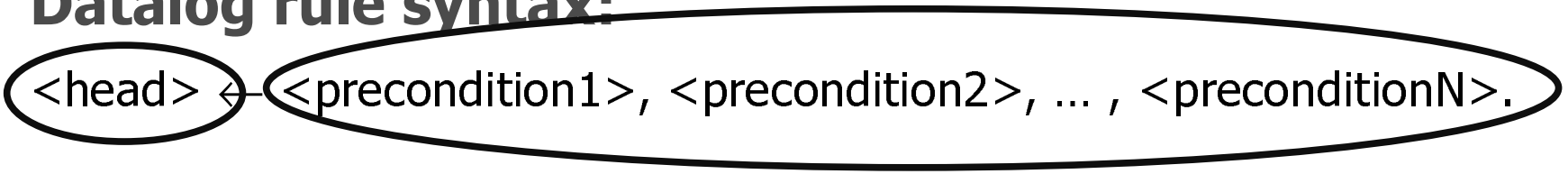
Distributed Execution Model



Declarative Routing

Introduction to Datalog

Datalog rule syntax:



All-Pairs Reachability

R1: reachable(S,D) ← link(S,D)

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

"For all S,D,
If link(S,D) exists, generate reachable(S,D)"

link(a,b) **– "there is a link from node *a* to node *b*"**

reachable(a,b) **– "node *a* can reach node *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".

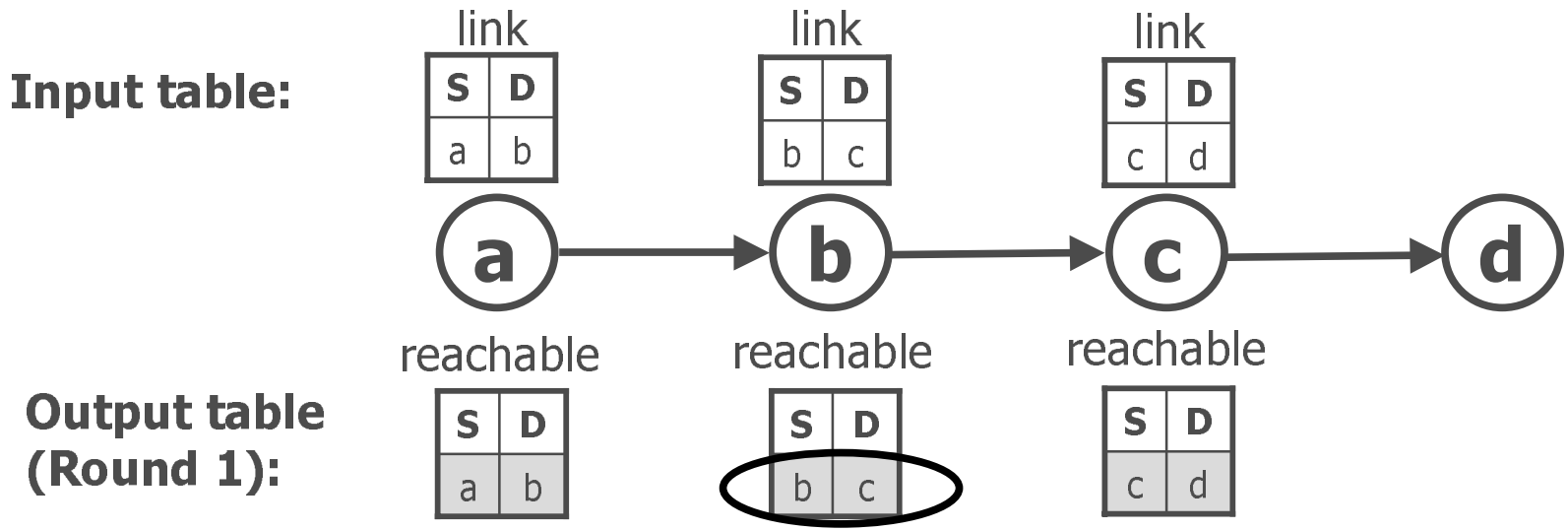
- Input: link(source, destination)
- Output: reachable(source, destination)

All-Pairs Reachability

R1: reachable(S,D) ← link(S,D)

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

Query: reachable(M,N) ← **All-Pairs**



E.g. R1: reachable(b,c) ← 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:

link

S	D
a	b

link

S	D
b	c

link

S	D
c	d

Output table (Round 2):

reachable

S	D
a	b
a	c

reachable

S	D
b	c
b	d

reachable

S	D
c	d

R2: reachable(b,d) ← link(b,c), reachable(c,d)

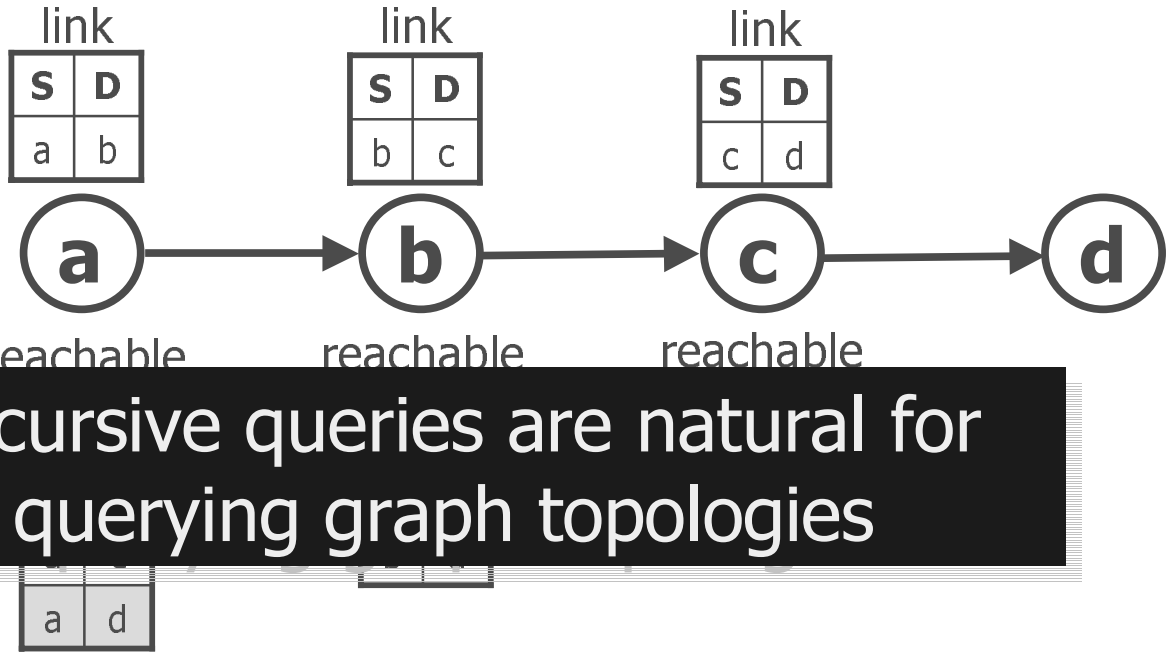
All-Pairs Reachability

R1: reachable(S,D) ← link(S,D)

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

Query: reachable(M,N)

Input table:



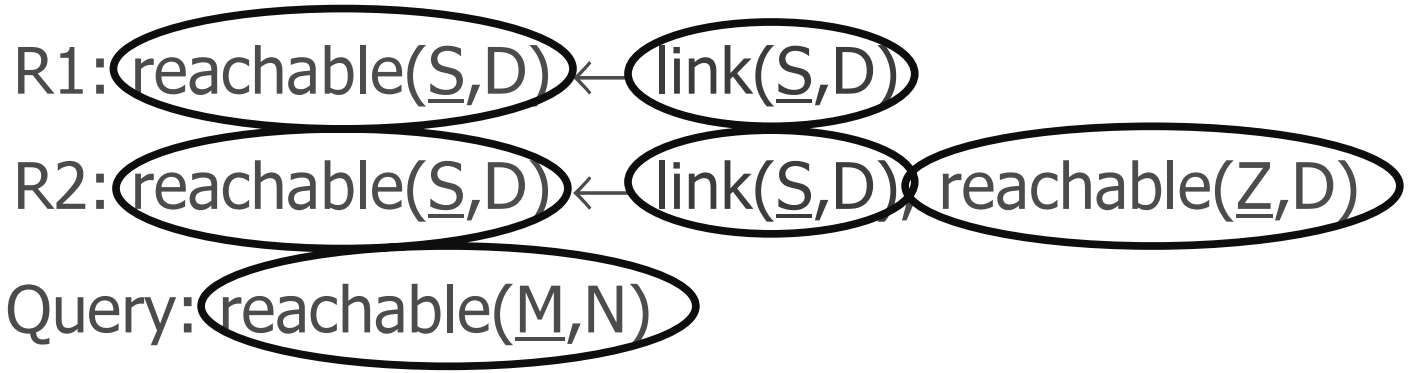
Output ta
(Round 3

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



Input table:

link

S	D
a	b

link

S	D
b	c

link

S	D
c	d

Output table:

reachable

S	D
a	b
a	c
a	d

reachable

S	D
b	c
b	d

reachable

S	D
c	d

Path Vector Protocol Example

R1: path(<u>S</u>,D,P) ← link(<u>S</u>,D), P=(S,D).

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

Query: path(<u>S</u>,D,P)

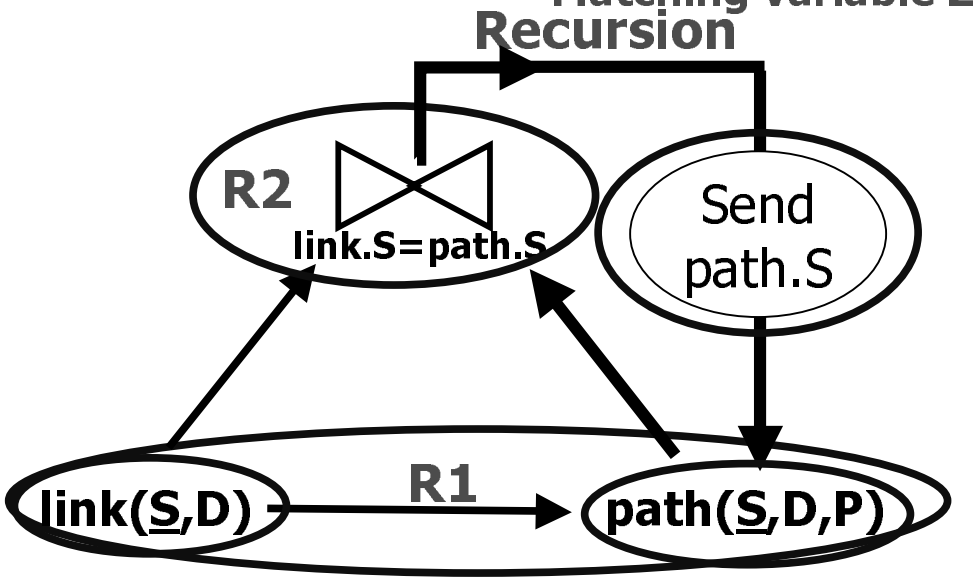
- Input: link(source, destination)
- Query output: path(source, destination, pathVector)

Datalog ➔ Execution Plan

R1: path(S,D,P) ← link(S,D), P=(S,D).

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

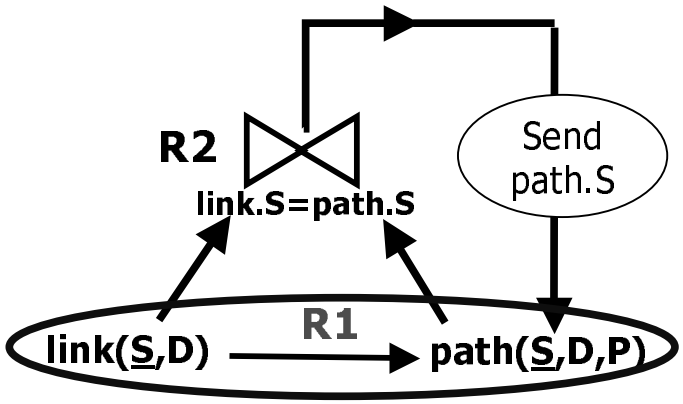
Matching variable Z = "Join" ⋈



Pseudocode at node Z:

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

Query Execution



R1: path(S,D,P) ← link(S,D), P=(S,D).

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

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

Neighbor table:

link

S	D
a	b

link

S	D
b	c
b	a

link

S	D
c	b
c	d

link

S	D
d	c

Forwarding table:

path

S	D	P
a	b	(a,b)

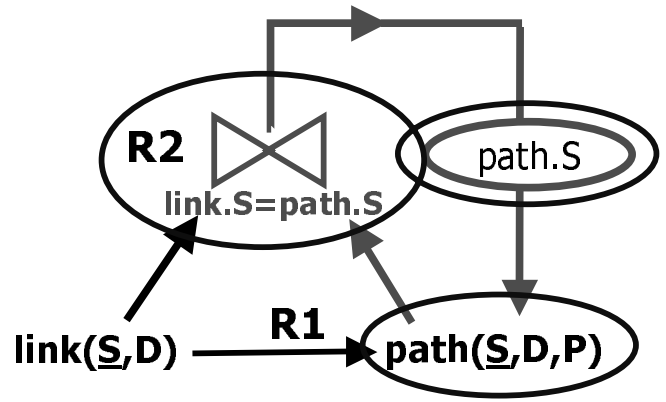
path

S	D	P
b	c	(b,c)

path

S	D	P
c	d	(c,d)

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:

link

S	D
a	b

link

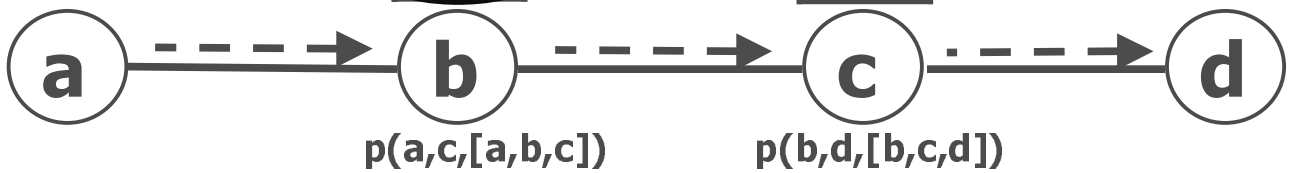
S	D
b	c
b	a

link

S	D
c	b
c	d

link

S	D
d	c



Forwarding table:

path

S	D	P
a	b	(a,b)
a	c	(a,b,c)

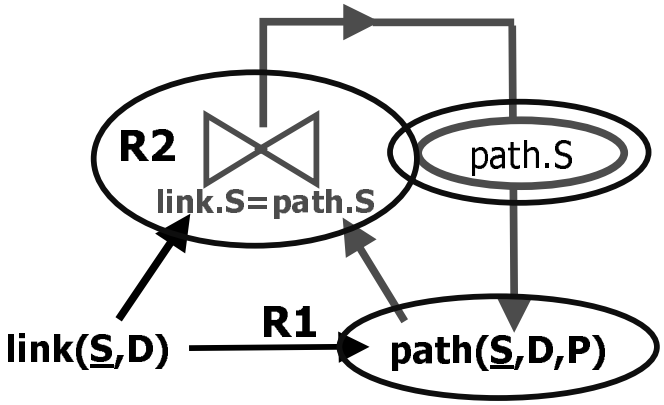
path

S	D	P
b	c	(b,c)
b	d	(b,c,d)

path

S	D	P
c	d	(c,d)

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:

link

S	D
a	b

link

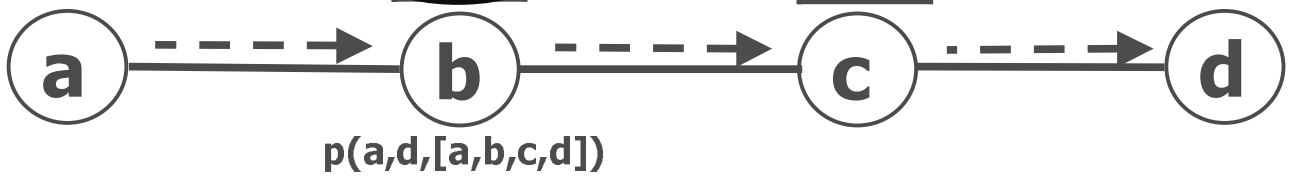
S	D
b	c
b	a

link

S	D
c	b
c	d

link

S	D
d	c



p(a,d,[a,b,c,d])

path

path

path

Forw table

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

a	d	(a,b,c,d)

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

} Minor variants give many options!

- Policy Decisions
- QoS-based Routing
- Link-state
- Multicast Overlays (Single-Source & CBT)

Expressiveness

- All-pairs all-paths:

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

Expressiveness

- Best-Path Routing:

R1: path(<u>S</u>,D,P,C) ← link(<u>S</u>,D,C), P= (<u>S</u>,D).

R2: path(<u>S</u>,D,P,C) ← link(<u>S</u>,Z,C_1), path(<u>Z</u>,D,P_2,C_2), C=C_1+C_2, P= S+P_2.

R3: bestPathCost(<u>S</u>,D,min<C>) ← path(<u>S</u>,D,Z,C)

R4: bestPath(<u>S</u>,D,Z,C) ← bestPathCost(<u>S</u>,D,C), path(<u>S</u>,D,P,C)

Query: bestPath(<u>S</u>,D,P,C)

Expressiveness

- Best-Path Routing:

R1: path(<u>S</u>,D,P,C) ← link(<u>S</u>,D,C), P= (<u>S</u>,D).
R2: path(<u>S</u>,D,P,C) ← link(<u>S</u>,Z,C_1), path(<u>Z</u>,D,P_2,C_2), C=FN(C_1,C_2), P=S+P_2.
R3: bestPathCost(<u>S</u>,D,AGG<C>) ← path(<u>S</u>,D,Z,C)
R4: bestPath(<u>S</u>,D,Z,C) ← bestPathCost(<u>S</u>,D,C), path(<u>S</u>,D,P,C)
Query: bestPath(<u>S</u>,D,P,C)

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

Expressiveness

- All-pairs all-paths:

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

Expressiveness

- Distance Vector:

R1: path(<u>S</u>,D,D,C) ← link(<u>S</u>,D,C)
R2: path(<u>S</u>,D,Z,C) ← link(<u>S</u>,Z,C_1), path(<u>Z</u>,D,W,C_2), C=C_1+C_2
R3: shortestLength(<u>S</u>,D,min<C>) ← path(<u>S</u>,D,Z,C)
R4: nextHop(<u>S</u>,D,Z,C) ← nextHop(<u>S</u>,D,Z,C), shortestLength(<u>S</u>,D,C)
Query: nextHop(<u>S</u>,D,Z,C)

Count to Infinity problem?

Expressiveness

- Distance Vector with Split Horizon:

R1: path(<u>S</u>,D,D,C) $\leftarrow$ link(<u>S</u>,D,C)
R2: path(<u>S</u>,D,Z,C) $\leftarrow$ link(<u>S</u>,Z,C_1), path(<u>Z</u>,D,W,C_2), C=C_1+C_2, W!=S
R3: shortestLength(<u>S</u>,D,min<C>) $\leftarrow$ path(<u>S</u>,D,Z,C)
R4: nextHop(<u>S</u>,D,Z,C) $\leftarrow$ nextHop(<u>S</u>,D,Z,C), shortestLength(<u>S</u>,D,C)
Query: nextHop(<u>S</u>,D,Z,C)

Expressiveness

- Distance Vector with Poisoned Reverse:

R1: path(<u>S</u>,D,D,C) ← link(<u>S</u>,D,C)
R2: path(<u>S</u>,D,Z,C) ← link(<u>S</u>,Z,C_1), path(<u>Z</u>,D,W,C_2), C=C_1+C_2, W!=S
R3: path(<u>S</u>,D,Z,C) ← link(<u>S</u>,Z,C_1), path(<u>Z</u>,D,W,C_2), C=∞, W=S
R4: shortestLength(<u>S</u>,D,min<C>) ← path(<u>S</u>,D,Z,C)
R5: nextHop(<u>S</u>,D,Z,C) ← nextHop(<u>S</u>,D,Z,C), shortestLength(<u>S</u>,D,C)
Query: nextHop(<u>S</u>,D,Z,C)

Expressiveness

- All-pairs all-paths:

R1: path(<u>S</u>,D,P,C) $\leftarrow$ link(<u>S</u>,D,C), P= (S,D).
R2: path(<u>S</u>,D,P,C) $\leftarrow$ link(<u>S</u>,Z,C_1), path(<u>Z</u>,D,P_2,C_2), C=C_1+C_2, P=S+P_2.
Query: path(<u>S</u>,D,P,C)

Expressiveness

- Dynamic Source Routing (DSR):

R1: path(<u>S</u>,D,P,C) ← link(<u>S</u>,D,C), P= (S,D).
R2: path(<u>S</u>,D,P,C) ← link(<u>Z</u>,D,C_2), path(<u>S</u>,Z,P_1,C_1), C=C_1+C_2,
P= P_1+D.
Query: path(<u>S</u>,D,P,C)

Switching *Right-recursion* to *Left-recursion* execution
=> 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=infinity.
  - Paths invalidated (cost=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