Datalog and Its Application to Network Routing Design

Boon Thau Loo
University of Pennsylvania

CS 598D: Formal Methods in Networking

02/22/10 and 02/26/10

Brief Introduction

- Graduated UC Berkeley in 2006.
- Post-doctoral research at MSR (Silicon Valley).
- Joined Penn in 2007 as Assistant Professor

- Research focus:
  - NetDB@Penn (http://netdb.cis.upenn.edu)
  - Distributed data management, Internet-scale query processing, data-centric techniques in networking.
  - Software methodologies and platforms for developing secure and formally verifiable distributed systems

Two-lecture plan

- 1st lecture on fundamentals (22nd Feb)
  - Overview of declarative networking
  - Introduction to Datalog
  - Connections between Datalog and network routing
  - Realizing the connection: distributed recursive query processing, incremental view maintenance, query optimizations

- 2nd lecture on advanced topics (26th Feb)
  - Spillovers from 22nd
  - (If time permits), discuss recent use cases (**overlay networks, security, protocol verification,** wireless networks, network composition, network provenance)

Goals for the two lectures

- Crash course on Datalog and declarative networking
- (More importantly), explore connections/synergies with other topics taught in this seminar. For example:
  - **Programming/specification frameworks:** Datalog vs other programming paradigms (Prolog, constraint logic programming) vs other forms of specifications (Promela)
  - **Verification:** Applying tools presented in class (Yices, Alloy, Isabelle, Kodkod, SAT/SMT solvers) to declarative networks
  - **Practical:** Integration with existing router platforms such as XORP.
- Final projects: *Cross-cutting* themes

“Homework” for you

- Send email to me (or if you prefer, to mailing list) describing
  - Name, year in grad school, advisor, background (in databases, networking, and formal methods) why you are interested in taking the class, how the class can help your research
- Read at least two papers (see next slides)
- RapidNet declarative networking system
  - http://netdb.cis.upenn.edu/rapidnet/
  - Click on “downloads” and download v0.2. Read documentation. Compile, and try one example protocol.
  - http://netdb.cis.upenn.edu/rapidnet/documentation.html has documentations on how to get started.
  - If you are stuck (either compilation or examples), email me for help.
- Older unsupported system
  - P2 (http://p2.cs.berkeley.edu)

Papers of Interest

- Datalog and recursive query processing:
  - ***A Survey of Research on Deductive Database Systems**, Ramakrishnan and Ullman, Journal of Logic Programming, 1993
  - **Database Management Systems,** Ramakrishnan and Gehkre. Chapter on "Deductive Databases".
  - (Courtesy of Simon) **What you always wanted to know about datalog (and never dared to ask)**, by Ceri, Gottlob, and Tanca.
  - **An ~~Amateur's~~ Expert's Guide to Recursive Query Processing**, Bancilhon and Ramakrishnan, SIGMOD Record.
  - **Maintaining Views Incrementally.** Gupta, Mumick, Subrahmanian. SIGMOD 1993.

* Covered in lecture

"Required" reading

Papers of Interest

* Covered in lecture

- Networking use cases:
  - ***Declarative Routing: Extensible Routing with Declarative Queries.** Loo, Hellerstein, Stoica, and Ramakrishnan. SIGCOMM 2005.
  - **Implementing Declarative Overlays.** Loo, Condie, Hellerstein, Maniatis, Roscoe, and Stoica. SOSP 2005.

- Distributed recursive query processing:
  - ***Declarative Networking: Language, Execution and Optimization**. Loo, Condie, Garofalakis, Gay, Hellerstein, Maniatis, Ramakrishnan, Roscoe, and Stoica, SIGMOD 06.
  - **Recursive Computation of Regions and Connectivity in Networks.** Liu, Taylor, Zhou, Ives, and Loo. ICDE 2009.

* Covered in lecture

"Required" reading

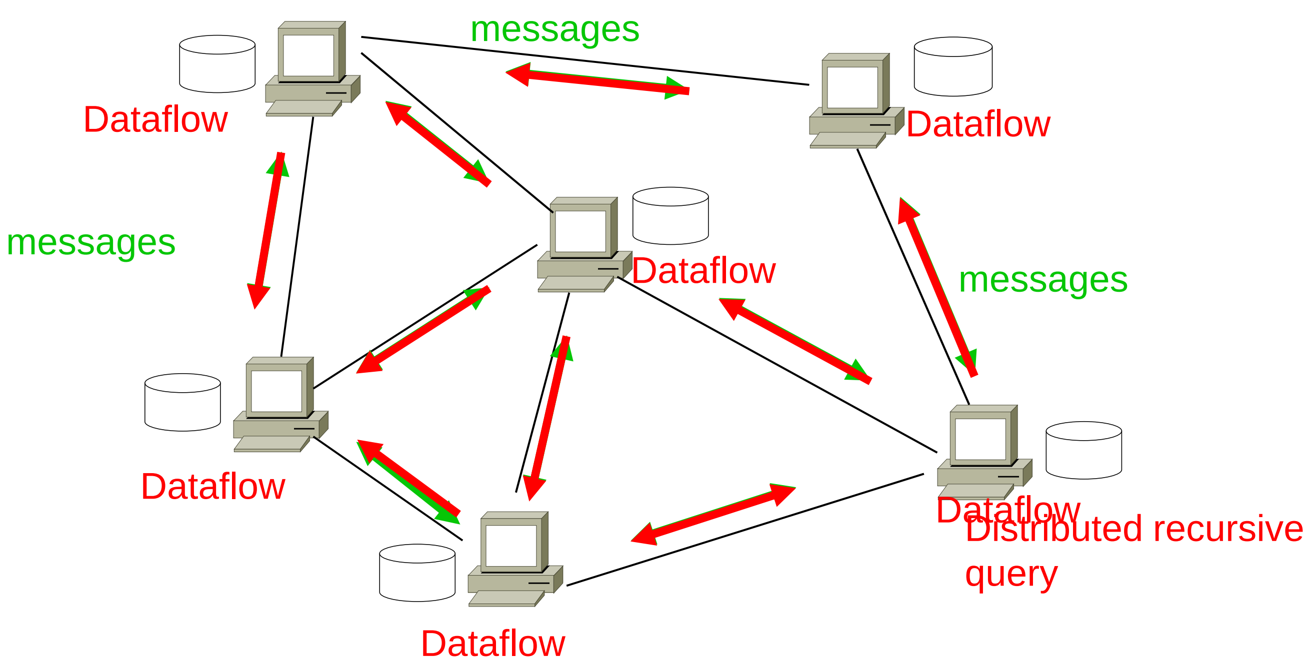
Today's outline

- Overview of declarative networking
- Introduction to Datalog
- Connections between Distributed Datalog and network routing
- Realizing the connection:
  - Distributed recursive query processing
  - Incremental view maintenance
  - Query optimizations

Declarative Networking

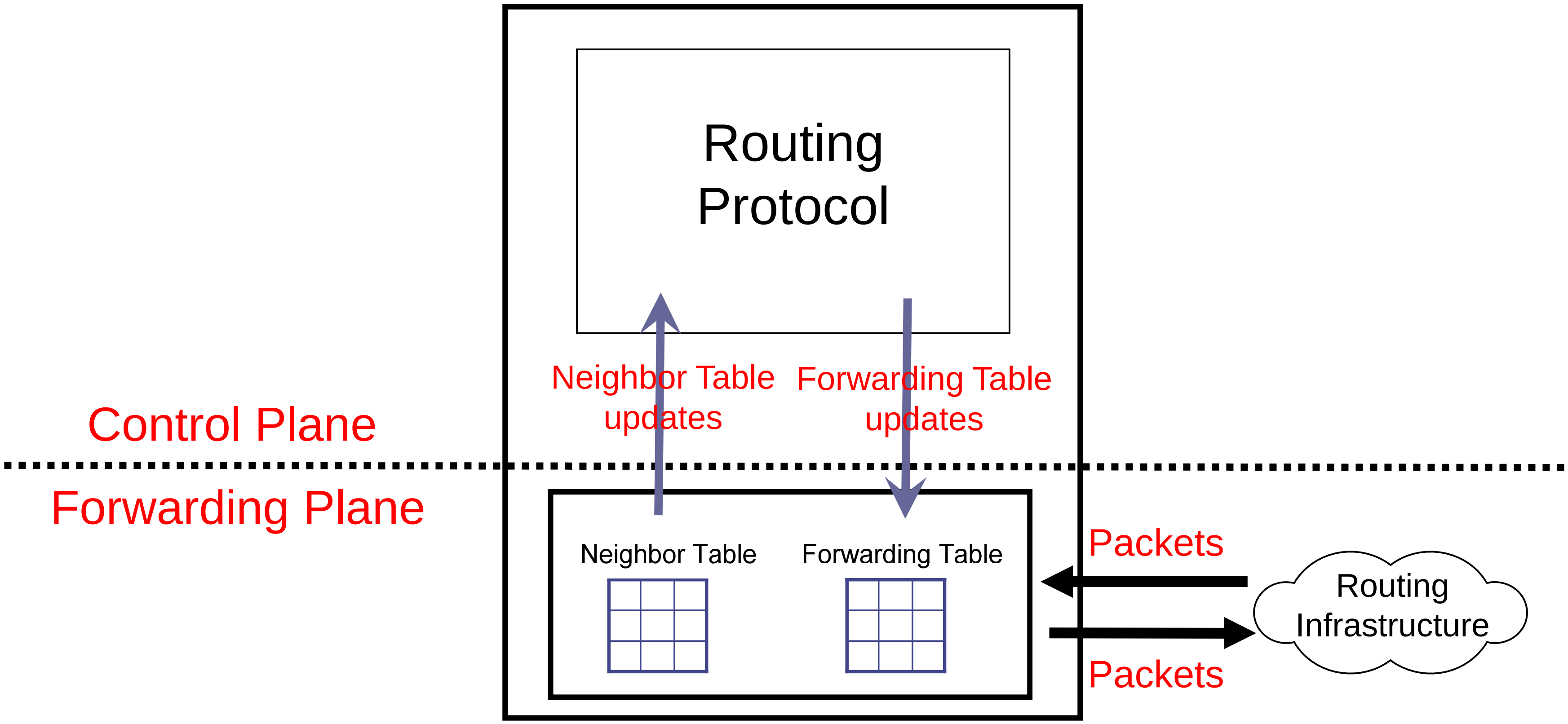
- A declarative framework for networks:
  - Declarative language: *“ask for what you want, not how to implement it”*
  - Declarative specifications of networks, compiled to distributed dataflows
  - Runtime engine to execute distributed dataflows
- Observation: *Recursive queries* are a natural fit for routing
- Recursive queries:
  - Traditionally for querying graph data structures stored in databases
  - Uses the Datalog language. Designed to be processed using database operators with set semantics.
  - Classic examples: Airline flight reservations, “Bill-of-Materials”, transitive closure

A Declarative Network



Traditional Networks	Declarative Networks
Network State	Distributed database
Network protocol	Recursive Query Execution
Network messages	Distributed Dataflow

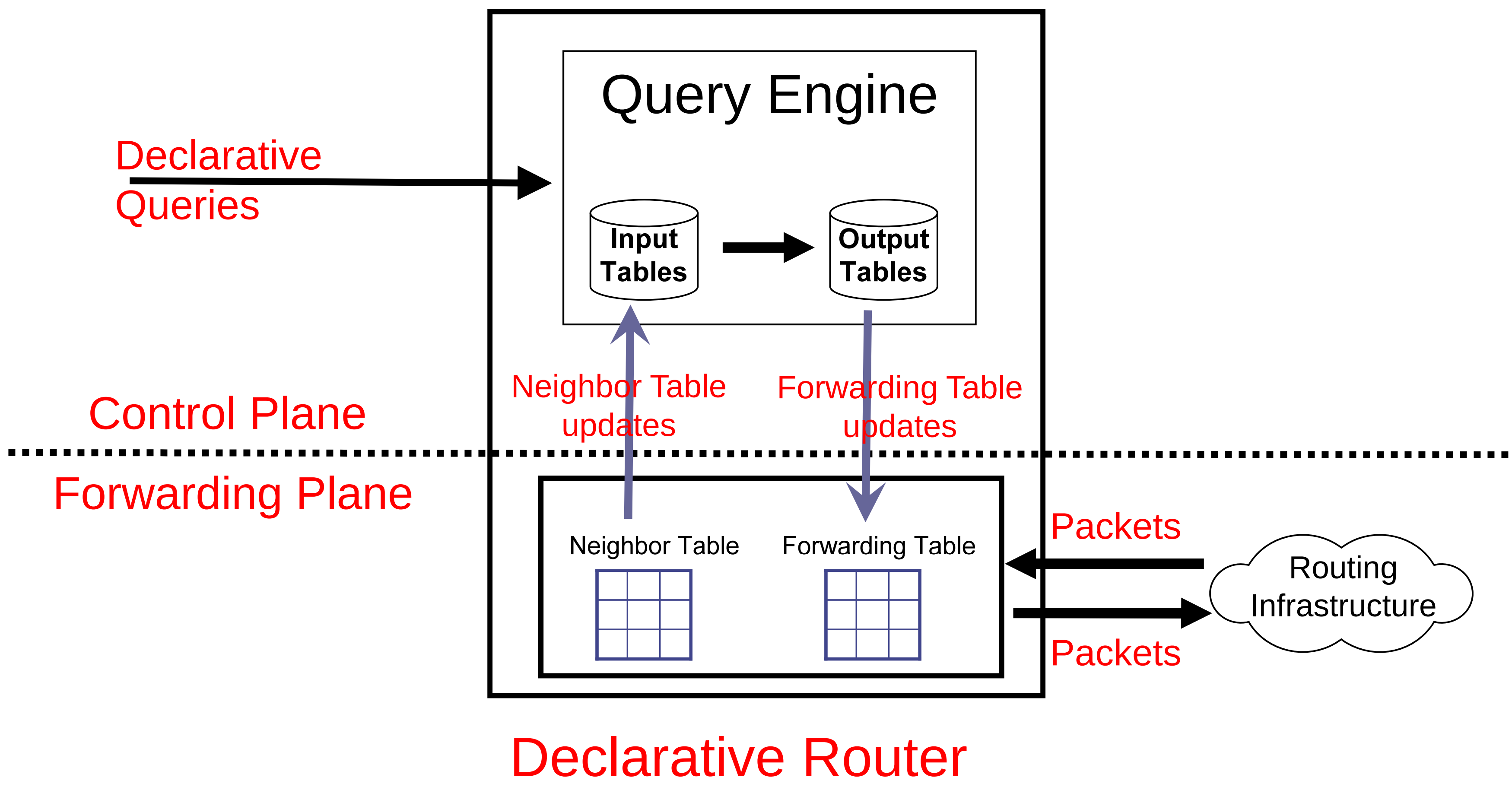
Traditional Router



Traditional Router

Declarative Router

SIGCOMM'05



The Case for Declarative

- Ease of programming:
  - Compact and high-level representation of protocols
  - Orders of magnitude reduction in code size
  - Easy customization and rapid prototyping
- Safety:
  - Queries are “sandboxed” within query processor
  - Potential for static analysis techniques on safety
- What about efficiency?
  - No fundamental overhead when executing standard routing protocols
  - Application of well-studied query optimizations

Large Library of Declarative Protocols

- Example implementations to date:
  - **Routing protocols:** DV, LS, DSR, AODV, OLSR, HSLS, etc.
  - **Overlay networks:** Distributed Hash Tables, Resilient overlay network (RON), Internet Indirection Infrastructure (i3), P2P query processing, multicast trees/meshes, etc.
  - **Network composition:** Chord over RON, i3+RON
  - **Hybrid protocols**: Combining LS and HSLS, epidemic and LS, routing + channel selection
  - **Others:** sensor networking protocols, replication, snapshot, fault tolerance protocols

Today's outline

- Overview of declarative networking
- Introduction to Datalog
- Connections between Distributed Datalog and network routing
- Realizing the connection:
  - Distributed recursive query processing
  - Incremental view maintenance
  - Query optimizations

Introduction to Datalog

Datalog rule syntax:

<result> ← <condition1>, <condition2>, … , <conditionN>.

Head Body

- Types of conditions in body:
  - Input tables: *link(src,dst)* predicate
  - Arithmetic and list operations
- Head is an output table
  - Recursive rules: result of head in rule body

Bottom-up Datalog evaluation

Given some initial input tables (conditions), and a set of rules of the form:
<result> $\leftarrow$ <condition1>, <condition2>, … , <conditionN>.

- Naïve

  Repeat

  Apply all rules (execute rule body to derive rule head)

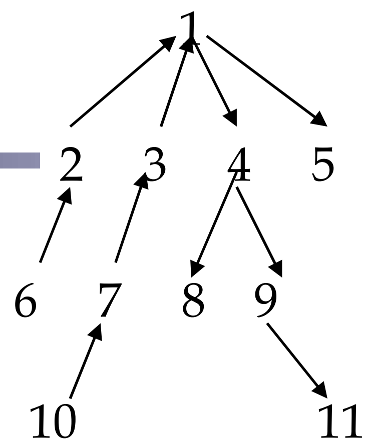
  Until no new tuples generated (*fixpoint* semantics)

- Semi-naïve
  - If a rule is applied in iteration N, at least one body fact must be a fact generated in iteration N-1 (and not before!).
  - No application is repeated.

Next 4 slides are courtesy of Raghu Ramakrishnan's course in Data Models and Languages (UW-Madison cs 784).

Example

up(2,1)	down(1,4)
up(3,1)	down(1,5)
up(6,2)	down(4,8)
up(7,3)	down(4,9)
up(10,7)	down(9,11)



R1: sg(X,Y) :- up(X,Z), down(Z,Y)

R2: sg(X,Y) :- up(X,Z_1), sg(Z_1,Z_2), down(Z_2,Y)

Query sg(6,Y) ?

Naïve Evaluation :

Step(1) (base case)
sg(2,4), sg(2,5), sg(3,4), sg(3,5)

Step(2) *(recursive case)*
Iteration 1
sg(6,8), sg(6,9), sg(7,8), sg(7,9)

Iteration 2
sg(6,8), sg(6,9), sg(7,8), sg(7,9), sg(10,11)

Iteration 3
No new tuples ("fixpoint")

Semi-naïve Evaluation:

Step(1) (base case)
sg(2,4), sg(2,5), sg(3,4), sg(3,5)

Step(2) *(recursive case)*
Iteration 1
sg(6,8), sg(6,9), sg(7,8), sg(7,9)

Iteration 2
sg(10,11)

Iteration 3
No new tuples ("fixpoint")

In-contrast: Top-down Evaluation

- <u>Given:</u>

  Call: $Q(?)$

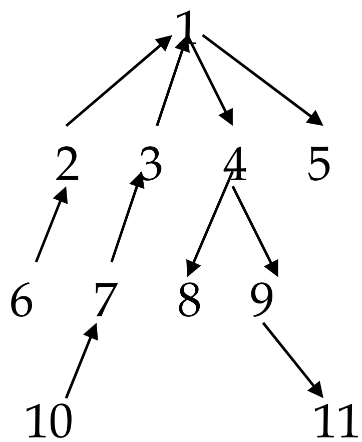
  Rule: Q IF $P_1 \wedge P_2 \ldots \wedge P_n$

  <u>Generate subgoals:</u>

  $P_1(?)\ P_2(?)\ \ldots P_n(?)$
- Advantage:
  - Computation is ‘focused’ in response to a query.
- Prolog is a language implemented in such a fashion.
  - Technique is called *resolution*

Example

up(2,1)	down(1,4)
up(3,1)	down(1,5)
up(6,2)	down(4,8)
up(7,3)	down(4,9)
up(10,7)	down(9,11)



R1: sg(X,Y) :- up(X,Z), down(Z,Y)

R2: sg(X,Y) :- up(X,Z_1), sg(Z_1,Z_2), down(Z_2,Y)

Query: sg(6,Y) ?

Prolog proceed as follows:

sg(6,y)_?

(R1)

up(6,Z)_?(Z=2); down(2,Y)_? fails;

up(6,Z)_? Fails on backtracking; (r1) fails.

(R2)

up(6,Z_1)_?(Z1=2);

sg(2,Z_2)_?

(r1)

up(2,Z')_?(Z'=1); down(1,Y')_?(Y'=Z_2=4);

sg(2,Z_2)_? succeeds with Z_2 =4;

down(4,Y)_?(Y=8);

sg(6,Y)_? succeeds with Y=8

Today's outline

- Overview of declarative networking
- Introduction to Datalog
- Connections between Distributed Datalog and network routing
- Realizing the connection:
  - Distributed recursive query processing
  - Incremental view maintenance
  - Query optimizations
- Use cases:
  - Overlay networks
  - Security
  - Protocol verification

Recap: All-Pairs Reachability

R1: reachable(S,D) ← link(S,D)

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

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

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

reachable(a,b) – “node *a* can reach node *b*”

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

All-Pairs Reachability

R1: reachable(S,D) $\leftarrow$ link(S,D)

➔ R2: reachable(S,D) $\leftarrow$ link(S,Z), reachable(Z,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)

Network Datalog

Location Specifier "@S"

R1: reachable(@S,D) ← link(@S,D)

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

Query: reachable(@M,N) ← All-Pairs Reachability

Input 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

Output table:

reachable

@S	D
@a	b
@a	c
@a	d

Query: reachable(@a,N)

reachable

@S	D
@b	a
@b	c
@b	d

reachable

@S	D
@c	a
@c	b
@c	d

reachable

@S	D
@d	a
@d	b
@d	c

Implicit Communication

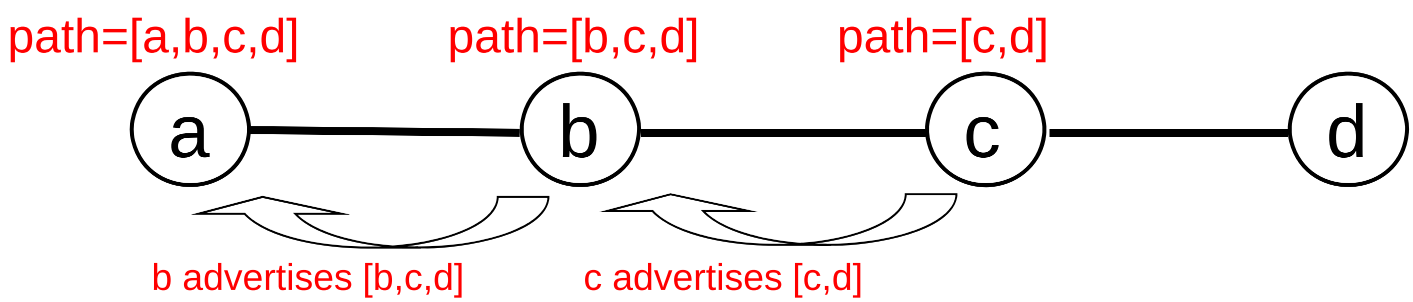
- A networking language with no explicit communication:

R2: reachable(@S,D) $\leftarrow$ link(@S,Z), reachable(@Z,D)

Data placement induces communication

Path Vector Protocol Example

- Advertisement: entire path to a destination
- Each node receives advertisement, add itself to path and forward to neighbors



Path Vector in Network Datalog

R1: $path(@S,D,P) \leftarrow link(@S,D), P=(S,D).$

R2: $path(@S,D,P) \leftarrow link(@Z,S), path(@Z,D,P_2), P=S \bullet P_2.$

Query: path(@S,D,P)

Add S to front of P_2

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

SQL-99 Equivalent

- with recursive path(src, dst, vec, length) as
  (SELECT src,dst, f_initPath(src,dst),1 from link   R1
  UNION
  SELECT link.src,path.dst,link.src ||’.’|| vec, length+1   R2
  FROM link, path where link.dst = path.src)

- create view minHops(src,dst,length) as
  (SELECT src,dst,min(length)
  FROM path group by src,dst)

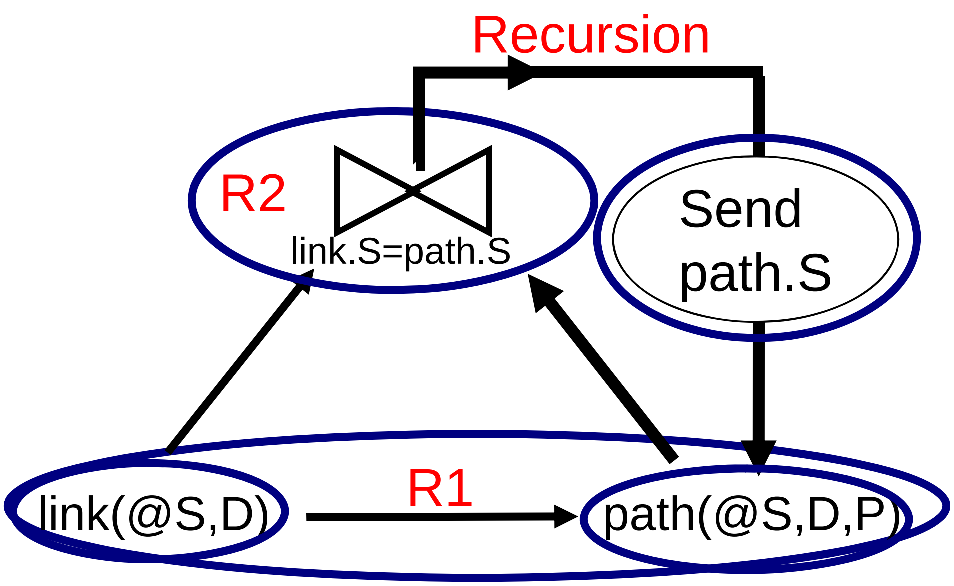
- create view shortestPath(src,dst,vec,length) as
  (SELECT P.src,P.dst,vec,P.length
  FROM path P, minHops H
  WHERE P.src = H.src and P.dst = H.dst and P.length = H.length)

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" ⋈



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(@a,d,P)

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

path

@S	D	P

path

@S	D	P
@c	d	[c,d]

Query Execution

R1: path(@S,D,P) ← link(@S,D), P=(S,D).

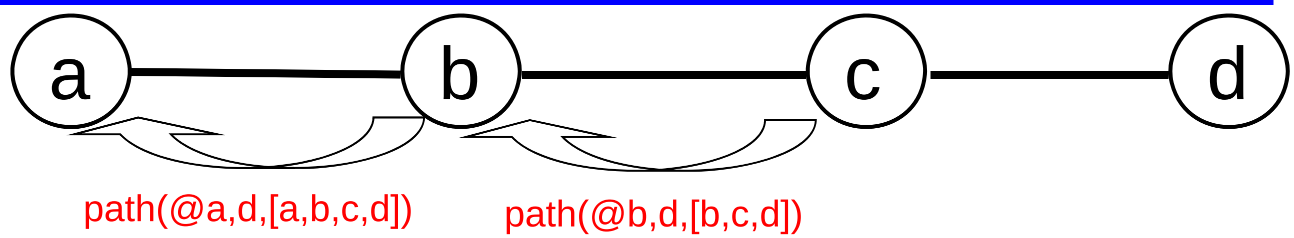
R2: path(@S,D,P) ← link(@Z,S), path(@Z,D,P_2), P=S•P_2.

Query: path(@a,d,P)

Matching variable Z = "Join" ⋈

link link link link

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



Forwarding table:

path

@S	D	P
@a	d	[a,b,c,d]

path

@S	D	P
@b	d	[b,c,d]

path

@S	D	P
@c	d	[c,d]

Example Routing Queries

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

Takeaways:

- Compact, natural representation
- Customization: easy to make modifications to get new protocols
- Connection between query optimization and protocols

All-pairs All-paths

R1: path(@S,D,P,C) $\leftarrow$ link(@S,D,C), P=(S,D).

R2: path(@S,D,P,C) $\leftarrow$ link(@S,Z,C_1), path(@Z,D,P_2,C_2), $C=C_1+C_2$,
P=S$\bullet P_2$.

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

All-pairs *Best*-path

R1: path(@S,D,P,C) $\leftarrow$ link(@S,D,C), P=(S,D).

R2: path(@S,D,P,C) $\leftarrow$ link(@S,Z,C_1), path(@Z,D,P_2,C_2), C=C_1+C_2,
P=S$\bullet P_2$.

R3: bestPathCost(@S,D,min<C>) $\leftarrow$ path(@S,D,P,C).

R4: bestPath(@S,D,P,C) $\leftarrow$ bestPathCost(@S,D,C), path(@S,D,P,C).

Query: bestPath(@S,D,P,C)

Customizable Best-Paths

R1: path(@S,D,P,C) $\leftarrow$ link(@S,D,C), P=(S,D).

R2: path(@S,D,P,C) $\leftarrow$ link(@S,Z,C_1), path(@Z,D,P_2,C_2), C=FN(C_1,C_2), P=S$\bullet$$P_2$.

R3: bestPathCost(@S,D,AGG<C>) $\leftarrow$ path(@S,D,O,C).

R4: bestPath(@S,D,P,C) $\leftarrow$ 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 capacity, *best-k*

All-pairs All-paths

R1: path(@S,D,P,C) $\leftarrow$ link(@S,D,C), P=(S,D).

R2: path(@S,D,P,C) $\leftarrow$ link(@S,Z,C_1), path(@Z,D,P_2,C_2), $C=C_1+C_2$, $P=S\bullet P_2$.

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

Distance Vector

R1: path(@S,D,D,C) ← link(@S,D,C).

R2: path(@S,D,Z,C) ← link(@S,Z,C_1), path(@Z,D,W,C_2), $C=C_1+C_2$

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?

Distance Vector with Split Horizon

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

Distance Vector with Poisoned Reverse

R1: path(@S,D,D,C) $\leftarrow$ link(@S,D,C)
R2: path(@S,D,Z,C) $\leftarrow$ link(@S,Z,C_1), path(@Z,D,W,C_2), C=C_1+C_2, W!=S
R3: path(@S,D,Z,C) $\leftarrow$ link(@S,Z,C_1), path(@Z,D,W,C_2), C=∞, W=S
R4: shortestLength(@S,D,min<C>) $\leftarrow$ path(@S,D,Z,C).
R5: nextHop(@S,D,Z,C) $\leftarrow$ nextHop(@S,D,Z,C), shortestLength(@S,D,C).
Query: nextHop(@S,D,Z,C)

Example Routing Queries

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

Takeaways:

- Compact, natural representation
- Customization: easy to make modifications to get new protocols
- Connection between query optimization and protocols

Today's outline

- Overview of declarative networking
- Introduction to Datalog
- Connections between Distributed Datalog and network routing
- Realizing the connection:
  - Distributed recursive query processing
  - Incremental view maintenance
  - Query optimizations
- Use cases:
  - Overlay networks
  - Security
  - Protocol verification

Recall: Bottom-up Datalog evaluation

- Naïve

  Repeat

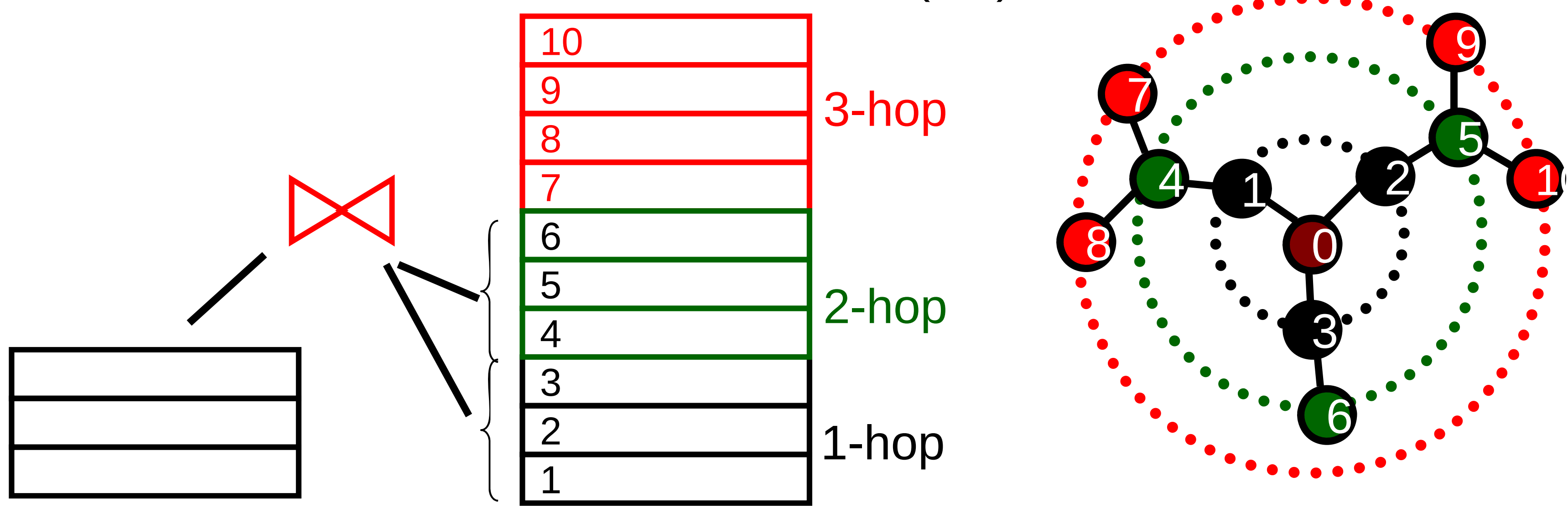
  Apply all rules

  Until no new tuples generated (*fixpoint* semantics)

- Semi-naïve
  - If a rule is applied in iteration N, at least one body fact must be a fact generated in iteration N-1 (and not before!).
  - No application is repeated.

Distributed Semi-naïve Evaluation

- Semi-naïve evaluation:
  - Iterations (rounds) of synchronous computation
  - Results from iteration i^{th} used in $(i+1)^{th}$



Link Table

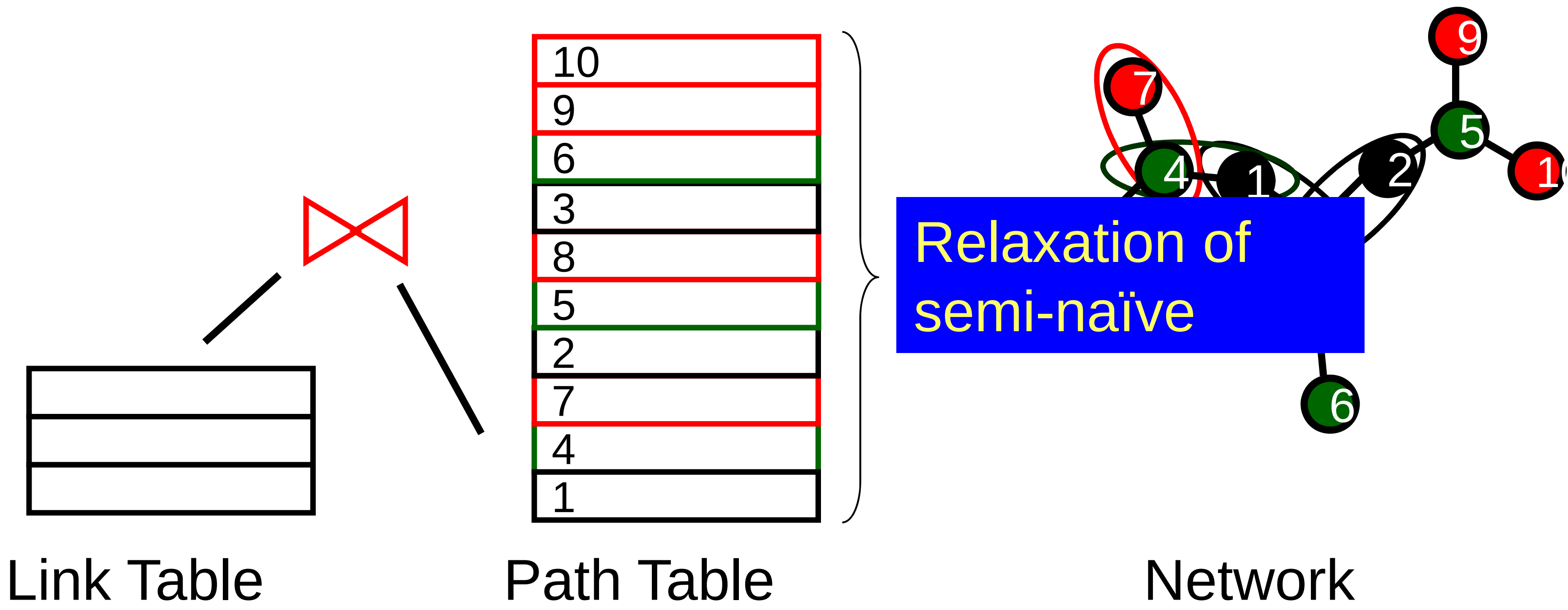
Path Table

Network

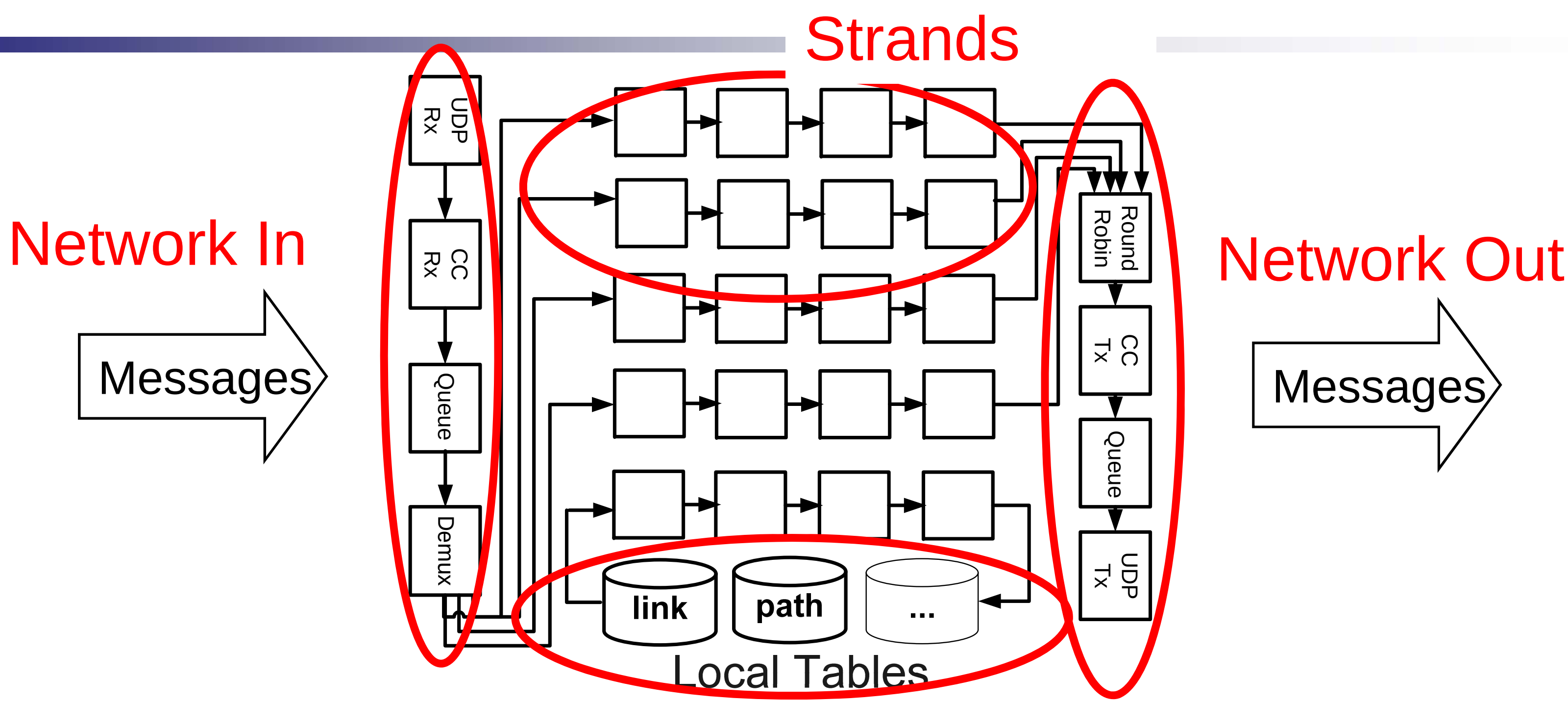
Problem: How do nodes know that an iteration is completed? Unpredictable delays and failures make synchronization difficult/expensive.

Pipelined Semi-naïve (PSN)

- Fully-asynchronous evaluation:
  - Computed tuples in *any* iteration pipelined to next iteration
  - Natural for distributed dataflows



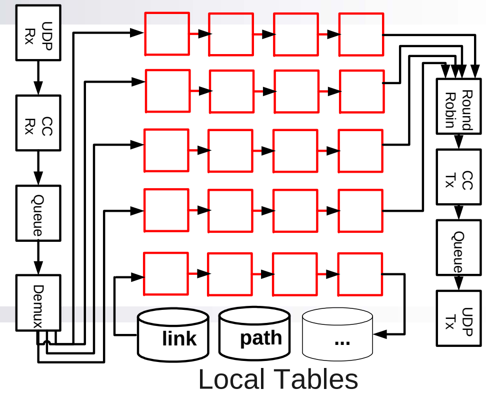
Dataflow Graph

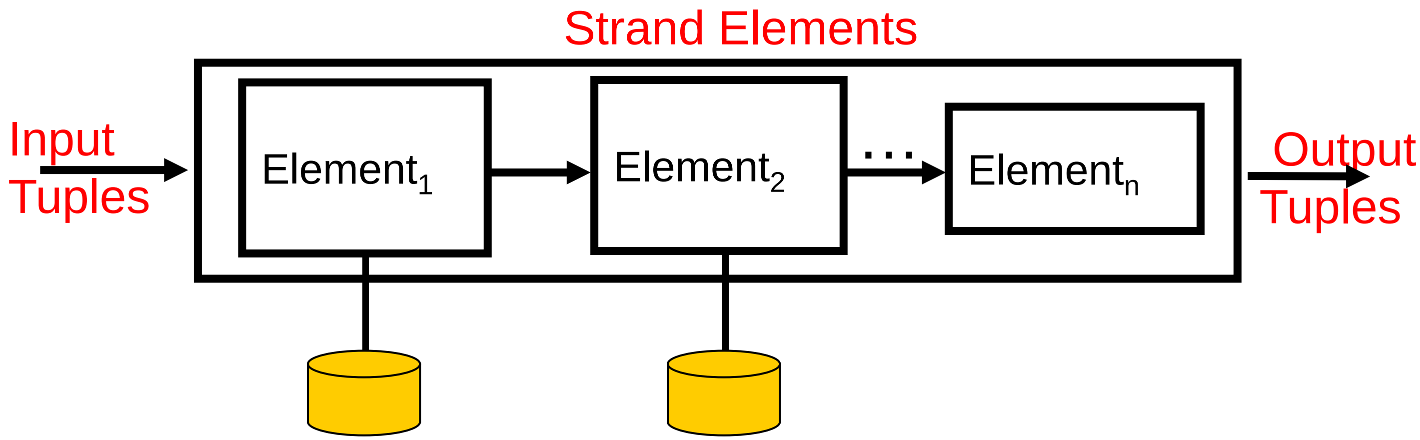


Single Node

- Nodes in dataflow graph ("elements"):
  - Network elements (send/recv, rate limitation, jitter)
  - Flow elements (mux, demux, queues)
  - Relational operators (selects, projects, joins, aggregates)

Dataflow Strand



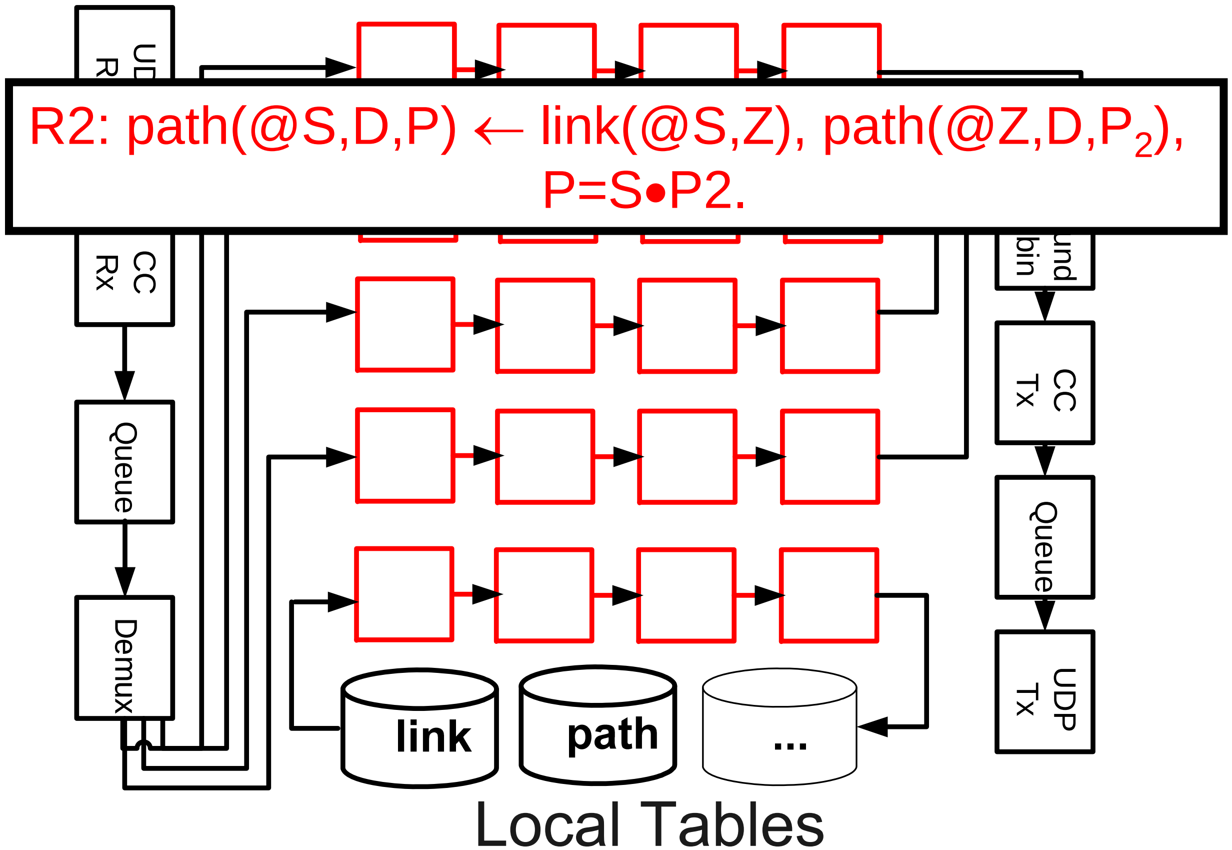


Input: Incoming network messages, local table changes, local timer events

Condition: Process input tuple using strand elements

Output: Outgoing network messages, local table updates

Rule → Dataflow “Strands”



Localization Rewrite

- Rules may have body predicates at different locations:

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

Matching variable Z = "Join" $\bowtie$

Rewritten rules:

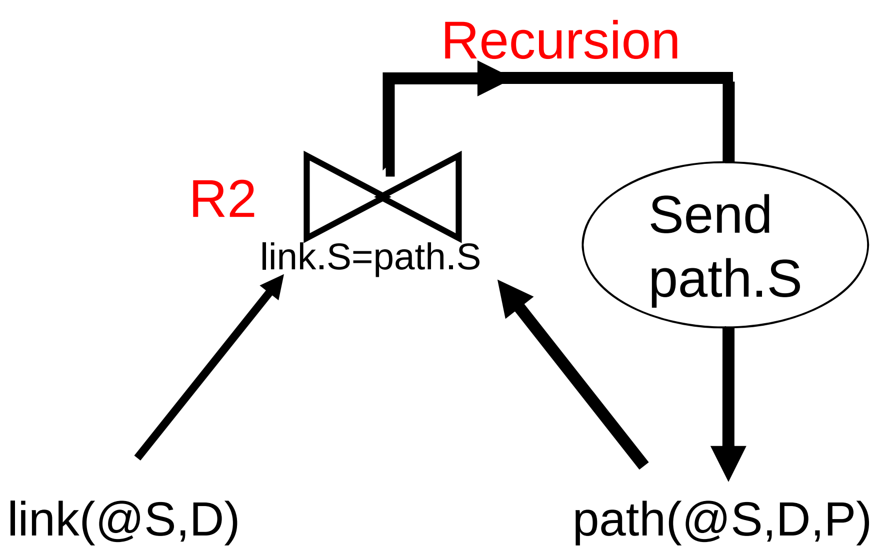
R2a: linkD(S,@D) ← link(@S,D)

R2b: path(@S,D,P) ← linkD(S,@Z), path(@Z,D,P_2), P=S•P_2.

Matching variable Z = "Join" $\bowtie$

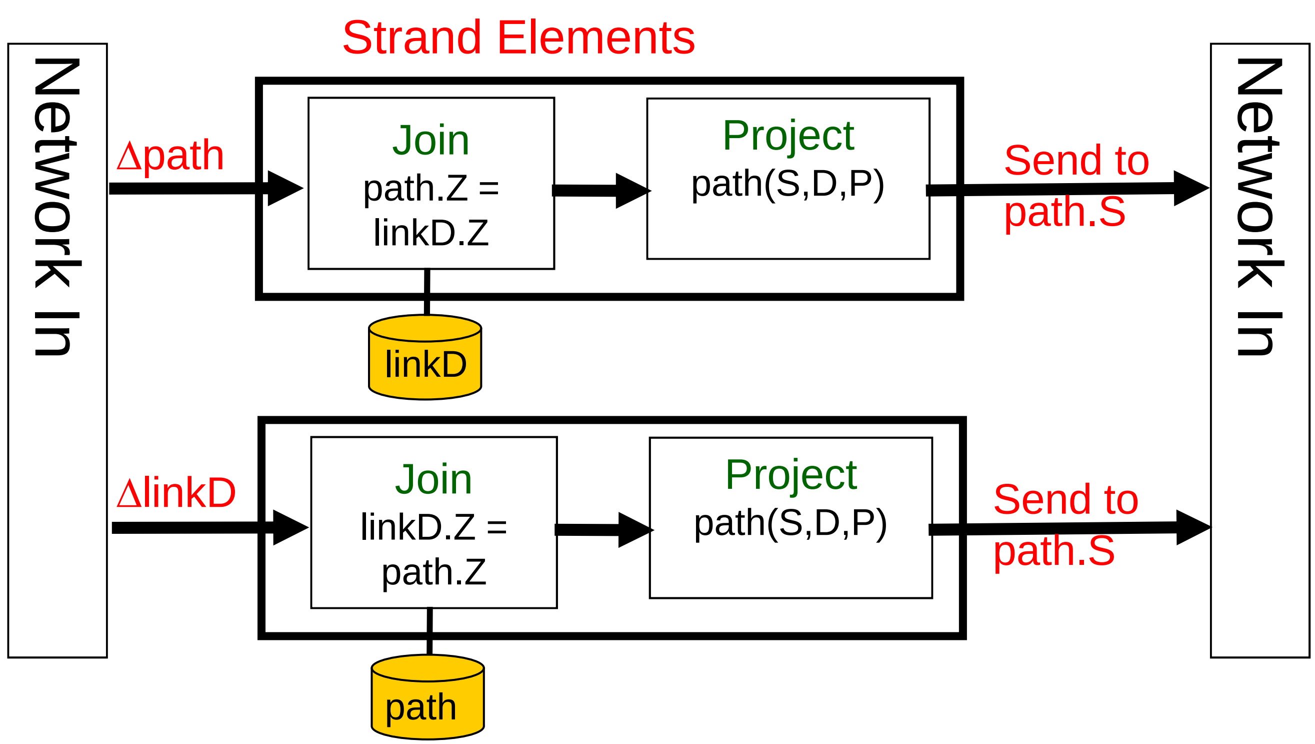
Logical Execution Plan

R2b: path(@S,D,P) $\leftarrow$ link(S,@Z), path(@Z,D,P_2), P=S • P_2.



Physical Execution Plan

R2b: path(@S,D,P) ← linkD(S,@Z), path(@Z,D,P_2), P=S•P_2.



Pipelined semi-naive

- Given a rule, decompose into “event-condition-action” *delta rules*
- *Delta rules translated into rule strands*

Consider the rule path(@S,D,P) ← linkD(S,@Z), path(@Z,D,P_2), P=S$\bullet P_2$.

- Insertion delta rules:

  +path(@S,D,P)> ← +linkD(S,@Z)>, path(@Z,D,P_2), P=S$\bullet P_2$.

  +path(@S,D,P)> ← linkD(S,@Z)>, +path(@Z,D,P_2), P=S$\bullet P_2$.
- Deletion delta rules:

  -path(@S,D,P)> ← -linkD(S,@Z)>, path(@Z,D,P_2), P=S$\bullet P_2$.

  -path(@S,D,P)> ← linkD(S,@Z)>, -path(@Z,D,P_2), P=S$\bullet P_2$.

- What about set semantics?

Pipelined Evaluation

- Challenges:
  - Does PSN produce the correct answer?
  - Is PSN bandwidth efficient?
    - I.e. does it make the minimum number of inferences?
- **Theorems** [SIGMOD'06]:
  - $RS_{SN}(p) = RS_{PSN}(p)$, where RS is results set
  - No repeated inferences in computing $RS_{PSN}(p)$
  - Require per-tuple timestamps in delta rules and FIFO and reliable channels
- Our recent insights: timestamps and FIFO are not strictly required, with some enhancements to PSN (see http://netdb.cis.upenn.edu/fvn for technical report)

Today's outline

- Overview of declarative networking
- Introduction to Datalog
- Overview of declarative networking
- Introduction to Datalog
- Connections between Distributed Datalog and network routing
- Realizing the connection:
  - Distributed recursive query processing
  - Incremental view maintenance
  - Query optimizations

Incremental View Maintenance

- Leverages insertion and deletion delta rules for state modifications
- Complications arise from duplicate evaluations.
- Consider the Reachable query. What if there are many ways to route between two nodes a and b, i.e. many possible derivations for reachable(a,b)?
- Mechanisms: still use delta rules, but additionally, apply
  - Count algorithm (for non-recursive queries)
  - Delete and Rederive (SIGMOD’93). Expensive in distributed settings
  - Per-tuple provenance for derivability test. (ICDE’09)

Maintaining Views Incrementally. Gupta, Mumick, Ramakrishnan, Subrahmanian. SIGMOD 1993.

Recursive Computation of Regions and Connectivity in Networks. Liu, Taylor, Zhou, Ives, and Loo. ICDE 2009.

Today’s outline

- Overview of declarative networking
- Introduction to Datalog
- Overview of declarative networking
- Introduction to Datalog
- Connections between Distributed Datalog and network routing
- Realizing the connection:
  - Distributed recursive query processing
  - Incremental view maintenance
  - Query optimizations

Overview of Optimizations

- Traditional: evaluate in the NW context
  - Aggregate Selections
  - Magic Sets rewrite
  - Predicate Reordering

PV/DV → DSR

- New: motivated by NW context
  - Multi-query optimizations:
    - Query Results caching
    - Opportunistic message sharing
  - Cost-based optimizations
    - Neighborhood density function
    - Hybrid rewrites

Zone Routing Protocol

  - Policy-based adaptation
    - More on Friday (if time permits)

Magic Sets Rewrite

- Unlike Prolog goal-oriented top-down evaluation, Datalog's bottom-up evaluation produces too many unnecessary facts.
- Networking analogy: computing all-pairs shortest paths is an overkill, if we are only interested in specific routes from sources to destinations.
- Solution: *magic sets rewrite*. IBM's DB2 for non-recursive queries.
- Dynamic Source Routing (DSR): PV + magic sets

routeRequest(@D,S,D,P,C) :- *magicSrc(@S), link(@S,@D,C),* P = (S,D).

routeRequest(@D,S,Z,P,C) :- *routeRequest(@Z,S,P1,C1),* link (@Z,D,C2),
C = C1 + C2, P = *P1 • Z.*

spCost(@D,S,min<C>) :- *magicDst(@D),* pathDst(@D,S,P,C).

shortestPath(@D,S,P,C) :- spCost(@D,S,C), pathDst(@D,S,P,C)

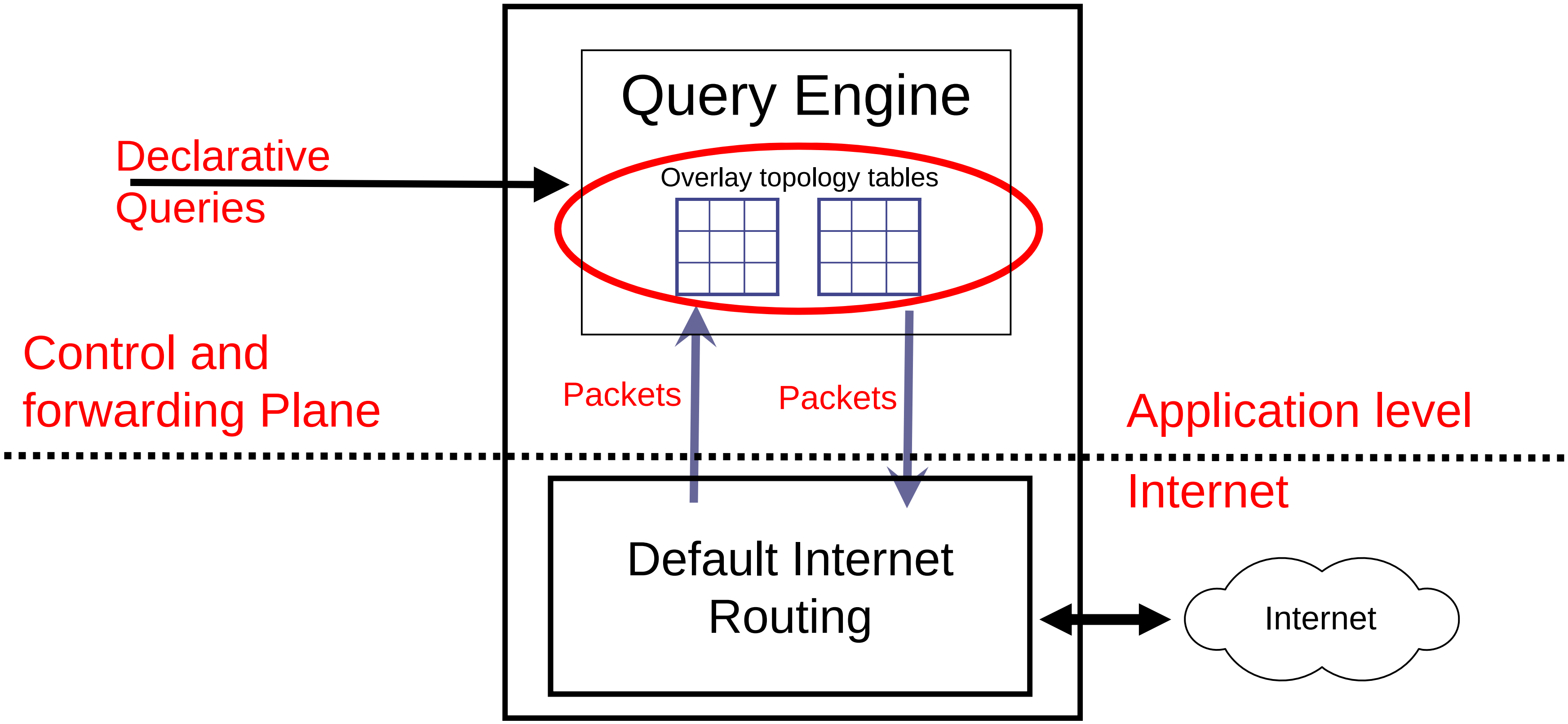
Aggregate Selections

- Prune communication using running state of monotonic aggregate
  - Avoid sending tuples that do not affect value of agg
  - E.g., shortest-paths query
- Challenge in distributed setting:
  - Out-of-order (in terms of monotonic aggregate) arrival of tuples
  - Solution: Periodic aggregate selections
    - Buffer up tuples, periodically send best-agg tuples

Today’s outline

- Overview of declarative networking
- Introduction to Datalog
- Connections between Distributed Datalog and network routing
- Realizing the connection:
  - Distributed recursive query processing
  - Incremental view maintenance
  - Query optimizations
- Use cases:
  - Overlay networks
  - Security
  - Protocol verification

Declarative Overlays



Declarative Overlay Node

Declarative Overlays

- More challenging to specify:
  - Not just querying for routes using input links
  - Rules for generating overlay topology
  - Message delivery, acknowledgements, failure detection, timeouts, periodic probes, etc…
  - Extensive use of timer-based event predicates:

    ping(@D,S) :- periodic(@S,10), link(@S,D)

- Chord DHT in 47 rules [SOSP'05]
- Overlay network composition [CoNEXT'08]

Declarative Secure Distributed Systems

http://netdb.cis.upenn.edu/ds2

- Declarative networking and access control languages are based on logic and Datalog
- Both extend Datalog in surprisingly similar ways
  - Notion of context (location) to identify components (nodes) in a distributed system
  - Suggests possibility to unify both languages
  - Leverage ideas from databases (e.g. efficient query processing and optimizations) to enforce access control policies
- Differences
  - Top-down vs bottom-up evaluation
  - Trust assumptions

Secure Network Datalog (SeNDlog)

- **Rules within a context**
  - Untrusted network
  - Predicates in rule body in local context
- **Authenticated communication**
  - "says" construct
  - *Export predicate:* "X says p@Y"
    - X exports the predicate p to Y.
  - *Import predicate:* "X says p"
    - X asserts the predicate p.

r1: reachable(@S,D) :- link(@S,D).
r2: reachable(@S,D) :- link(@S,Z),
reachable(@Z,D).

⇩ *localization rewrite*

At S:
s1: reachable(@S,D) :- link(S,D).
s2: linkD(D,S)@D :- link(S,D).
s3: reachable(S,D)@Z :- linkD(S,Z),
reachable(S,D).

⇩ *authenticated communication*

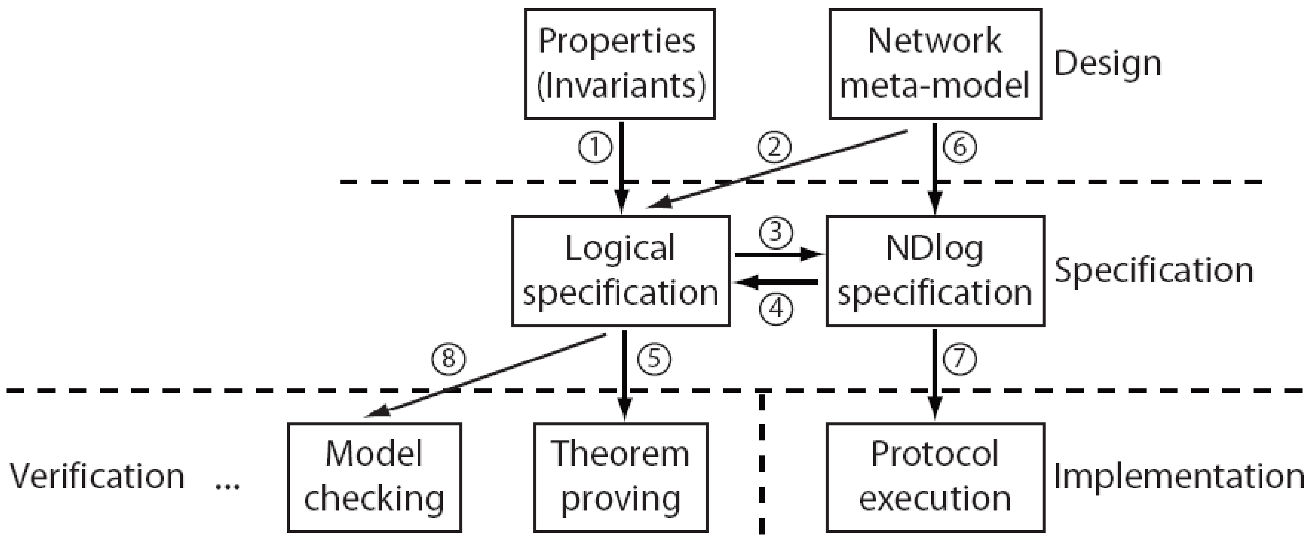
At S:
s1: reachable(S,D) :- link(S,D).
s2: S says linkD(D,S)@D :- link(S,D).
s3: S says reachable(S,D)@Z :-
Z says linkD(S,Z),
W says reachable(S,D).

Example Usage of SeNDlog

- **Secure network routing [ICDE'09]**
  - Nodes import/export signed route advertisements from neighbors
  - Advertisements include signed sub-paths (*authenticated provenance*)
  - Building blocks for secure BGP
  - Secure packet forwarding
- **Customizable anonymous routing [NDSS'10]**
  - Path selection and setting up "onion routes" with layered encryption
  - Application-aware Anonymity (http://a3.cis.upenn.edu)
- ***Network Provenance* [SIGMOD'10]**
  - Distributed proof trees.
  - Protocol analysis and debugging
  - Efficient provenance querying and maintenance

Formally Verifiable Networking (FVN)

http://netdb.cis.upenn.edu/fvn



- Conceptually sound meta-model for program synthesis
  - Formal logical statements specify the behavior and the properties of the protocol
  - Declarative networking bridges logical specifications and actual implementation
- Theorem proving establishes correctness proof
  - System specification => property specification
  - Machine checked proof, proof automation support
- HotNets'09 paper