Declarative Networking

Mothy

Joint work with Boon Thau Loo, Tyson Condie, Joseph M. Hellerstein, Petros Maniatis, Ion Stoica

Intel Research and U.C. Berkeley

June 1, 2005

Intel Research Berkeley

Off the top of my head…

- Running packet networks remains a complex and difficult problem.
- Despite ~25 years of research, no abstractions have emerged to modularize the problem.
- I find this astonishing. Can someone correct me?

It's not all been wasted…

- Lots of measurement
  - ⇒ lots and lots of data now
- Understand "network level" well
  - TCP, BGP, Malware, etc.
- Plenty of control mechanisms
  - DCAN, RCP, 4D, etc., etc.

- Hypothesis: for IP at least, we as researchers already understand this well enough to abstract and uplevel.
- Can we just move on?

Mental exercise

- For a moment, try to forget everything you know about BGP, OSPF, IS-IS, DVMRP, etc., etc.
- Take a deep breath or two.
- Doesn't that feel good?

A different abstraction

- The set of routing tables in a network represents a *distributed data structure*
- The data structure is characterized by a set of ideal *properties* which define the network
  - Think in terms of structure, not protocol
- *Routing* is the process of maintaining these properties in the face of changing ground facts
  - Failures, topology changes, load, policy…

Routing and Query Processing

- In database terms, the routing table is a *view* over changing network conditions and state
- Maintaining it is the domain of distributed continuous query processing

Distributed Continuous Query Processing

- Relatively new and active field
  - SDIMS, Mercury, IrisLog, Sophia, etc., in particular PIER
  - $\Rightarrow$ May not have all the answers yet
- But brings a wealth of experience and knowledge from database systems
  - Relational, deductive, stream processing, etc.

Goal: a constrained declarative language for network specification

- Higher-level view of routing properties
  - More than simply a configuration language
- Modular decomposition of function
- Static analysis for:
  - Optimization techniques
  - Safety checking
- Dynamic optimization
  - C.f. eddies, etc.

Other advantages

- Can incorporate other knowledge into routing policies
  - C.f. Jennifer's examples, and beyond
  - E.g. Physical network knowledge
- Naturally integrates *discovery*
  - If you buy Paul's argument
- Also provides an abstraction point for such information
  - Knowledge itself doesn't need to be exposed.

What are we doing, then?

- Express network properties in *DataLog*
  - Preliminary to better languages
- Execute specifications to maintain routing and discovery
- Two directions / implementations:
  - IP Routing (SIGCOMM 2005)
  - Overlays (under submission)

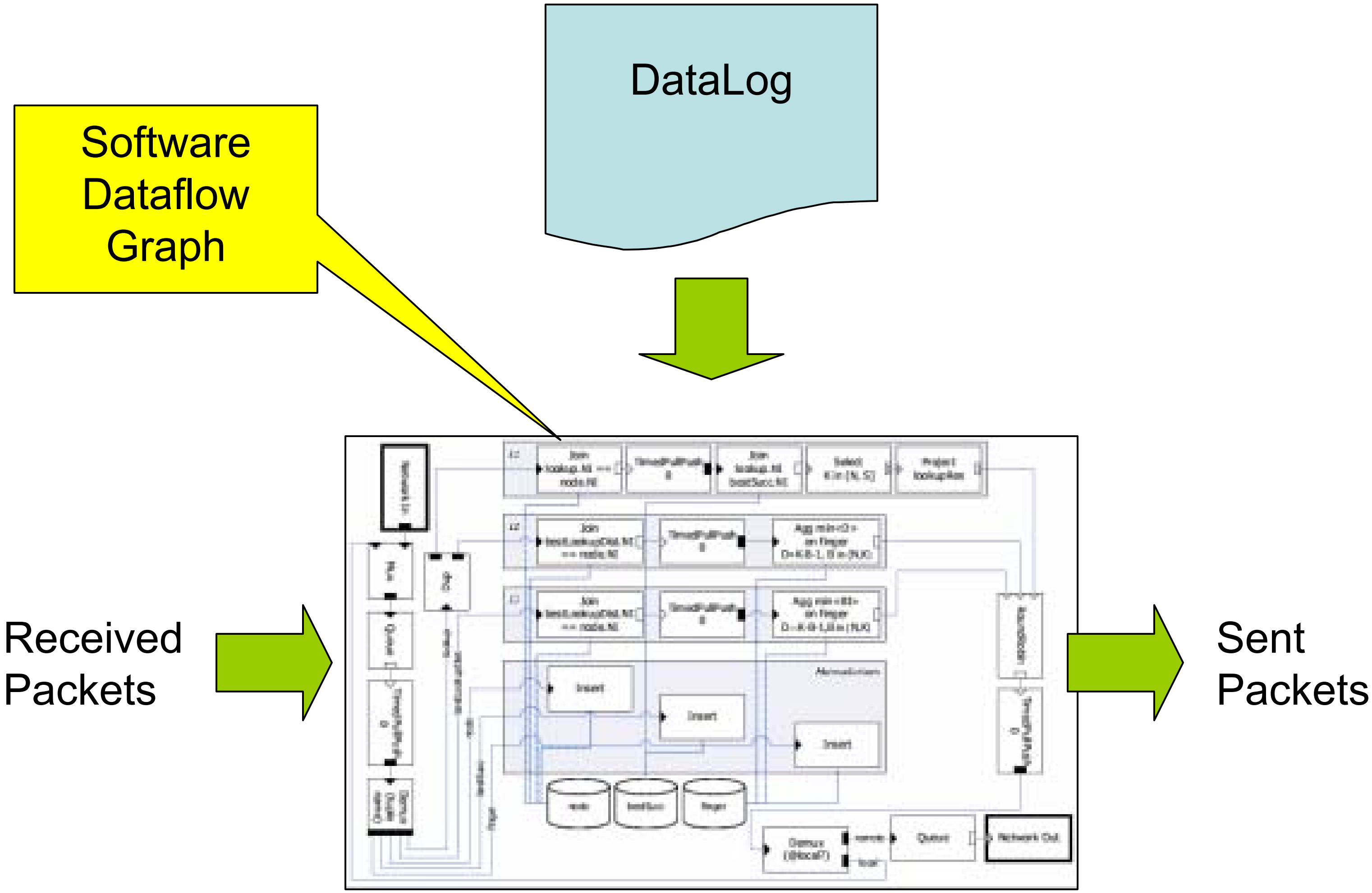
Why overlays?

- Overlays in a very *broad* sense
  - Any application-level routing system
  - Email servers, multicast, CDNs, DHTs, etc.
- Ideal test case
  - Clearly deployable short-term
  - Defers interoperability issues
- The overlay design space is wide
  - $\Rightarrow$ ensures we cover the bases
- Testbed for wider applicability

A Declarative Overlay Engine: "P2"

- Everything is a declarative query
  - Overlay construction, maintenance, routing, monitoring
- Queries compiled to software dataflow graph and directly executed
- System written from scratch (C++)
  - Deployable (PlanetLab, Emulab)
  - Already has reasonable performance for deployed overlays

P2

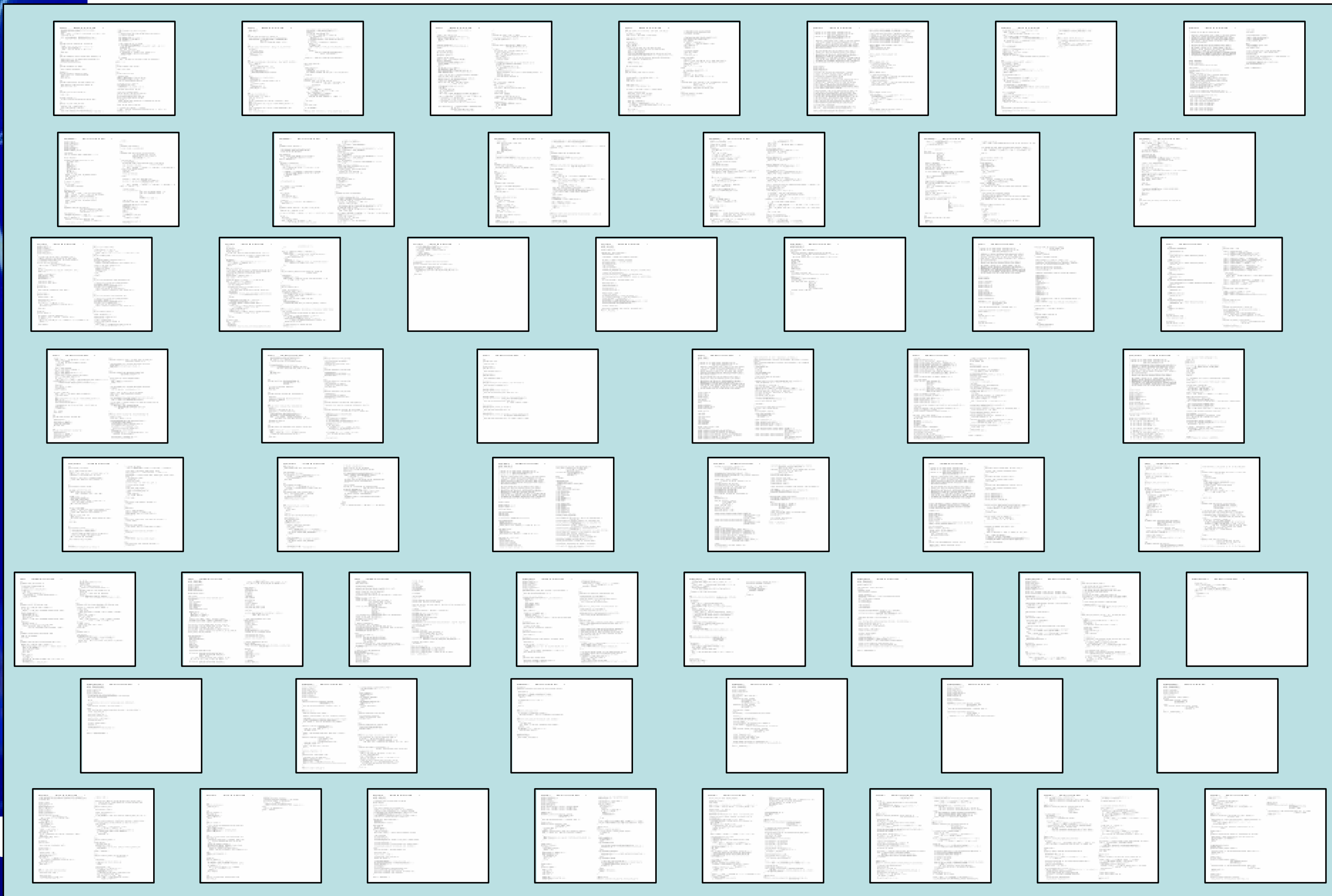


Intel Research Berkeley

Example: Chord in 33 rules

```
chord.plg        Sun Apr 10 23:43:49 2005        1

/*
 * 1.1 Chord
 * ---------
 */

/* The base tuples */
materialise(node, infinity, 1).
materialise(bestSucc, tSucc * 2, 1).
materialise(finger, tFix * 2, 160).
materialise(succ, tSucc * 2, 16).
materialise(pred, infinity, 1).
materialize(join, tJoin, 5).
materialize(landmark, infinity, 5).
materialize(stabilize, tStabilizeWait, 5).
materialize(pingNode, tPingSoftState, infinity).

/** Lookups |||3||| */
rule L1 lookupResults@R(R,K,S,SI,E) :- node@NI(NI,N), lookup@NI(NI,K,R,E),
        bestSucc@NI(NI,S,SI), K in (N,S].
rule L2 bestLookupDist@NI(NI,K,R,E,min<D>) :- lookup@NI(NI,K,R,E),
        finger@NI(NI,I,B,BI), node@NI(NI,N), B in (N,K), D=K-B-1.
rule L3 lookup@BI(min<BI>,K,R,E) :- node@NI(NI, N),
        bestLookupDist@NI(NI,K,R,E,D), B in (N,K).
        finger@NI(NI,I,B,BI), D=K-B-1.

/* Neighbor Selection |||3||| */
rule SU1 bestSuccDist@NI(NI,min<D>) :- node@NI(NI,N), succ@NI(NI,S,SI),
        D=S-N-1.
rule SU2 bestSucc@NI(NI,S,SI) :- succ@NI(NI,S,SI),
        bestSuccDist@NI(NI,D), node@NI(NI,N), D=S-N-1.
rule SU3 finger@NI(NI,0,S,SI) :-
        bestSuccessor@NI(NI,S,SI).

/* Successor eviction |||4||| */
rule SR1 succCount(NI,count<>) :- succ(NI,S,SI).
rule SR2 evictSucc@NI(NI) :- succCount@NI(NI,C), C >
        succSize.
rule SR3 maxSuccDist@NI(NI,max<D>) :- succ@NI(NI,S,SI),
        node@NI(NI,N), D = f_dist(N,S)-1, evictSucc@NI(NI).
rule SR4 delete@NI<succ(NI,S,SI)> :- succ@NI(NI,S,SI),
        maxSuccDist@NI(NI,D), D=f_dist(N,S)-1.

/* Finger fixing |||3||| */
rule F1 fFix@NI(NI,E,I) :- periodic@NI(NI,E,t_Fix), I in [0, fNum),
        f_coinFlip(fFixProb).
rule F2 lookup@NI(NI,K,NI,E) :- fFix@NI(NI,E,I), node(NI,N), K = N + 1
        << I.
rule F3 finger@NI(NI,I,B,BI) :- fFix@NI(NI,E,I),
        lookupRes@NI(NI,K,B,BI,E), K in (N + 1<<I, N), node@NI(NI,N).

/* Churn Handling |||5||| */
rule J1 pred@NI(NI,null,"").
rule J2 joinReq@LI(LI,N,NI,E) :- join@NI(NI,E), node@NI(NI,N),
        landmark@NI(NI,LI), LI != "".
rule J3 succ@NI(NI,N,NI) :- landmark@NI(NI,LI), node@NI(NI,N),
        join@NI(NI,E), LI = "".
rule J4 lookup@LI(LI,N,NI,E) :- joinReq@LI(LI,N,NI,E).
rule J5 succ@NI(NI,S,SI) :- join@NI(NI,E), lookupRes@NI(NI,K,S,SI,E).

/* Stabilization |||8||| */
rule S0 stabilize@NI(NI, E) :- periodic@NI(NI, E, t_stab).
rule S1 stabilizeReq@SI(SI,NI,E) :- stabilize@NI(NI,E),
        bestSucc@NI(NI,S,SI).
rule S2 sendPred@PI1(PI1,P,PI,E) :- stabilizeReq@NI(NI,PI1,E),
        pred@NI(NI,P,PI), PI != "-".
rule S3 succ@NI(NI,P,PI) :- node(NI,N), sendPred@NI(NI,P,PI,E),
        bestSucc@NI(NI,S,SI), P in (N,S), stabilize@NI(NI,E).
rule S4 sendSucc@SI(SI,NI) :- stabilize@NI(NI, E),
        succ@NI(NI,S,SI).
rule S5 succ@PI(PI,S,SI) :- sendSucc@NI(NI,PI), succ@NI(NI,S,SI).
rule S6 notifyPred@SI(SI,N,NI) :- stabilize@NI(NI, E), node@NI(NI,N),
        successor@NI(NI,S,SI).
rule S7 pred@NI(NI,P,PI) :- node@NI(NI,N), notifyPred@NI(NI,P,PI),
        pred@NI(NI,P1,PI1), ((PI1 == "") || (P in (P1, N))).

/** Connectivity Monitoring */
rule C1 pingReq@PI(PI, NI, E, TS) :- periodic@NI(NI, E, tPing),
        pingNode@NI(NI, PI), TS = f_currentTime().
rule C2 pingResp@RI(RI, NI, E, TS) :- pingReq@NI(NI, RI, E, TS).
rule C3 latency@NI(NI, PI, L) :- pingReply@NI(NI, PI, E, TS),
        pingReq@NI(NI, PI, E, TS1), TS2 = f_currentTime(), L = TS2 -
        TS1.

rule CS1 pingNode@NI(NI,SI) :- succ@NI(NI,S,SI).
rule CS2 succ@NI(NI, S, SI) :- succ@NI(NI, S, SI), latency@NI(NI, SI,
        L).

rule CF1 pingNode@NI(NI,PI) :- finger@NI(NI,I,B,BI).
rule CF2 finger@NI(NI, I, B, BI) :- finger@NI(NI, I, B, BI),
        latency@NI(NI, BI, L).
```

Comparison: MIT Chord in C++



Conclusion

- An abstraction and infrastructure for radically rethinking networking
  - One possibility: System R for networks
- Where does the network end and the application begin?
  - E.g. can run queries to monitor the network at the endpoints
  - Integrate resource discovery, management, routing
  - Chance to reshuffle the networking deck

Intel Research Berkeley

Thanks.