Automated Verification of Safety Properties of Declarative Networking Programs

Chen Chen[1], **Lay Kuan Loh[2]**, Limin Jia[2], Wenchao Zhou[3], Boon Thau Loo[1]
[1]University of Pennsylvania, [2]Carnegie Mellon University, [3]Georgetown University

July 15, 2015

Networks are complex and error-prone

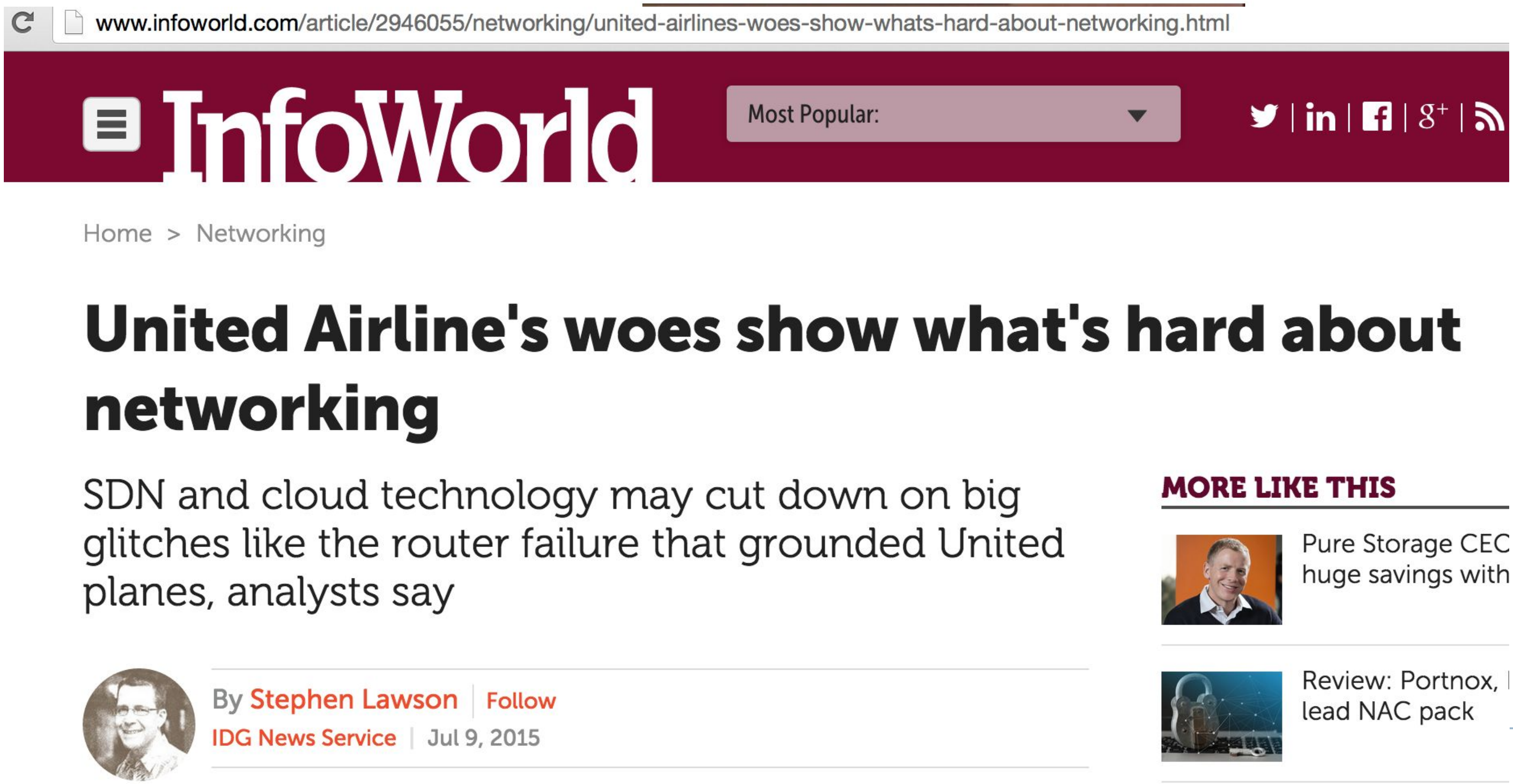
venturebeat.com/2013/08/16/3-minute-outage-costs-google-545000-in-revenue/

VB | NEWS EVENTS

Channels Newsletters Got news? Tell us!

Cloud

5-minute outage costs Google $545,000 in revenue



www.infoworld.com/article/2946055/networking/united-airlines-woes-show-whats-hard-about-networking.html

InfoWorld

Most Popular:

Home > Networking

United Airline's woes show what's hard about networking

SDN and cloud technology may cut down on big glitches like the router failure that grounded United planes, analysts say

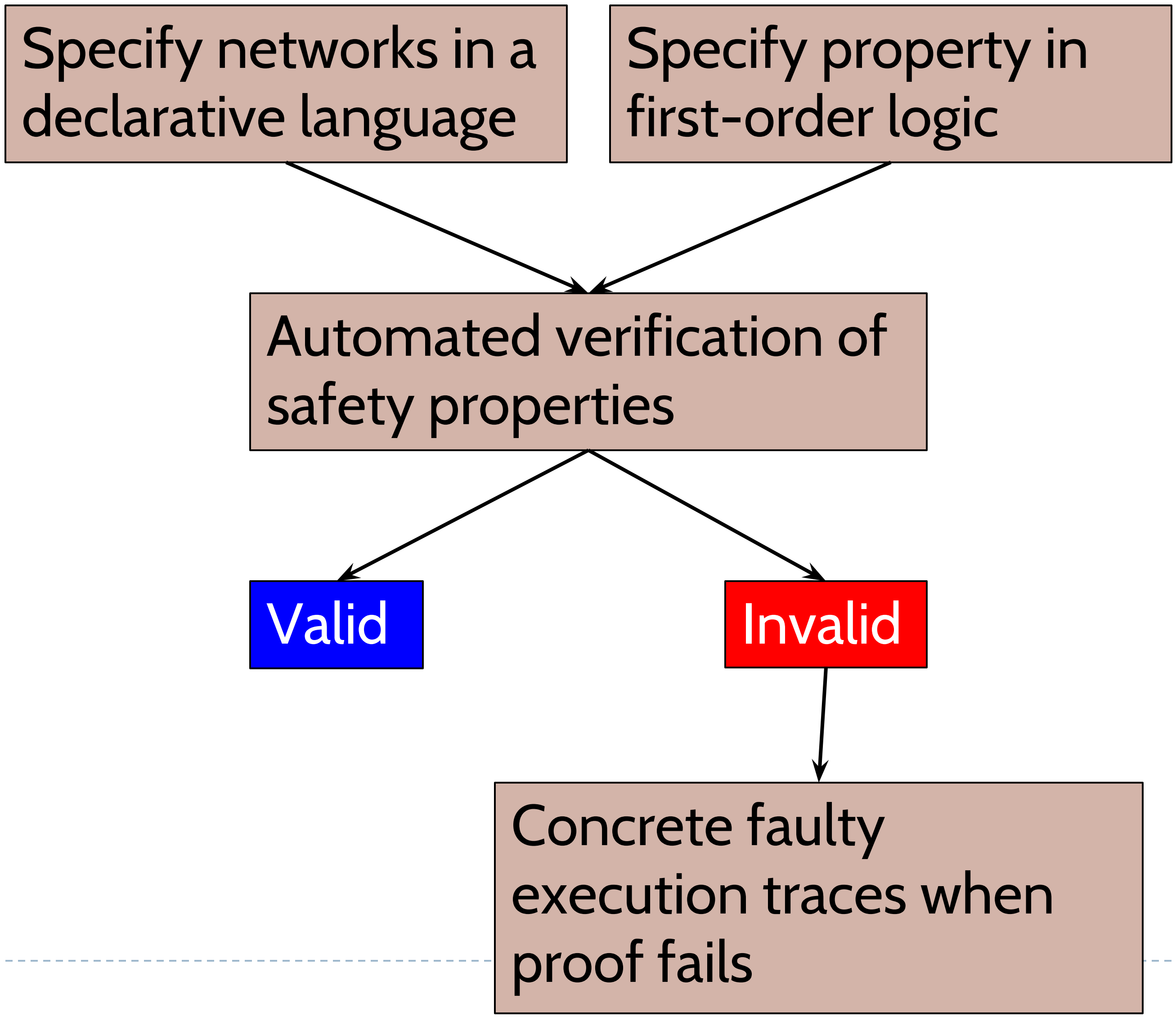
By Stephen Lawson | Follow
IDG News Service | Jul 9, 2015

MORE LIKE THIS

Pure Storage CEC huge savings with

Review: Portnox, lead NAC pack

Our solution

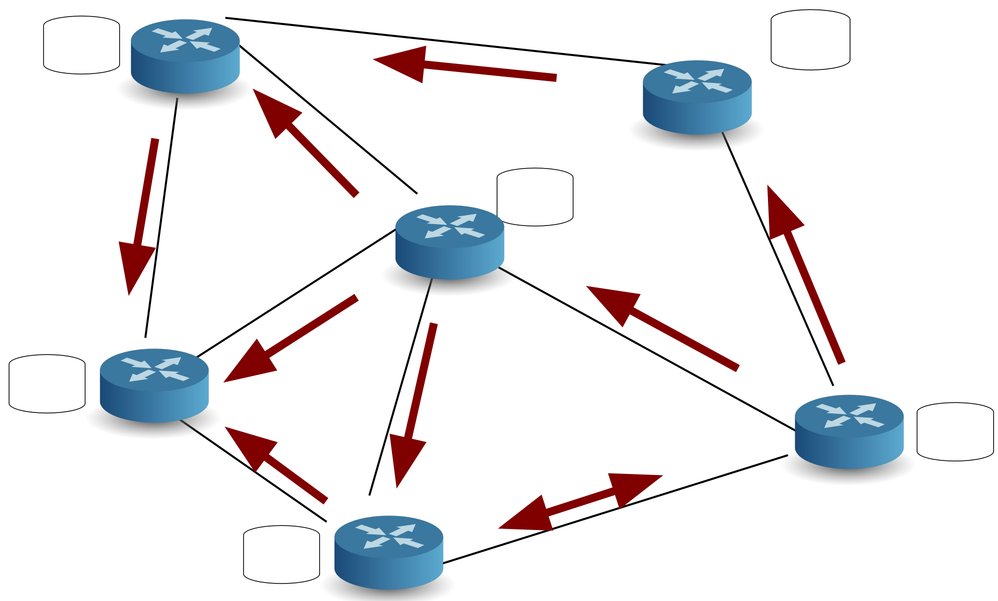


Roadmap

- Introduction of NDLog (Network Datalog)
- Algorithm analysis
  - Derivation pool construction
  - Property query
  - Network constraints
  - Recursive programs
- Case study
- Conclusion

Network Datalog (NDLog) [CACM'09]

- A distributed variant of Datalog
- Recursive query language over network states



Traditional Networks
- Network state
- Network Protocol

Declarative Networks
- Distributed Database
- Datalog Program

Rule format of NDLog

- Rule Head :- $Body_1$, $Body_2$, …, $Body_n$, Constraint
- @: Location specifier

<u>Twohops</u>

R1 onehop(@z,x,c2) :- link(@z,x,c2)

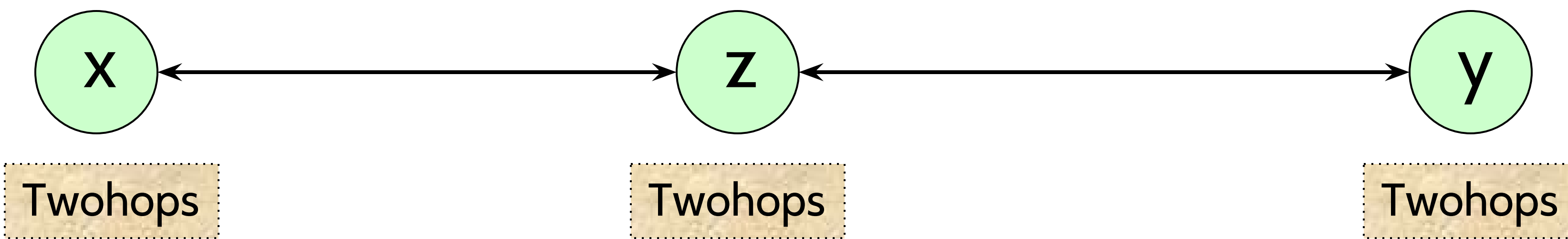
R2 twohops(@x,y,c) :- link(@z,y,c1), onehop(@z,x,c2), c=c1+c2

Running an example NDLog program

Twohops
R1 onehop(@z,x,c2) :- link(@z,x,c2)
R2 twohops(@x,y,c) :- link(@z,y,c1), onehop(@z,x,c2), c=c1+c2

link

@Src	Dst	Cost
@z	y	c1
@z	x	c2



Running an example NDLog program

<u>Twohops</u>

R1 onehop(@z,x,c2) :- link(@z,x,c2)

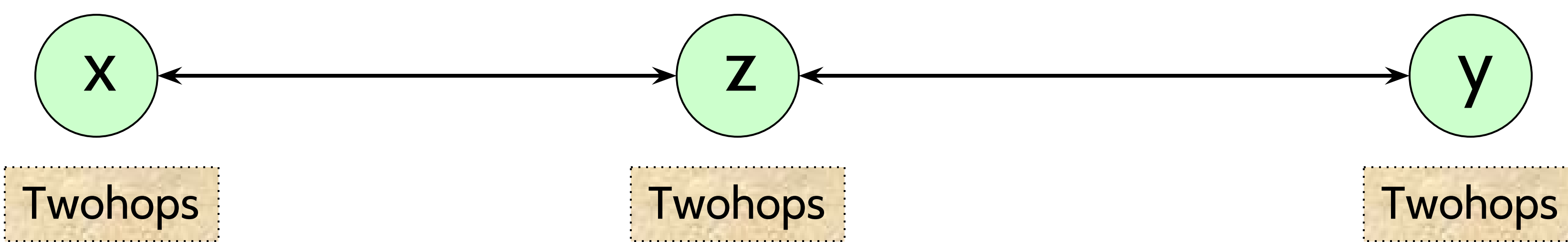
R2 twohops(@x,y,c) :- link(@z,y,c1), onehop(@z,x,c2), c=c1+c2

link

@Src	Dst	Cost
@z	y	c1
@z	x	c2

onehop

@Src	Dst	Cost
@z	x	c2



Running an example NDLog program

<u>Twohops</u>

R1 onehop(@z,x,c2) :- link(@z,x,c2)

R2 twohops(@x,y,c) :- link(@z,y,c1), onehop(@z,x,c2), c=c1+c2

@Src	Dst	Cost
@x	y	c

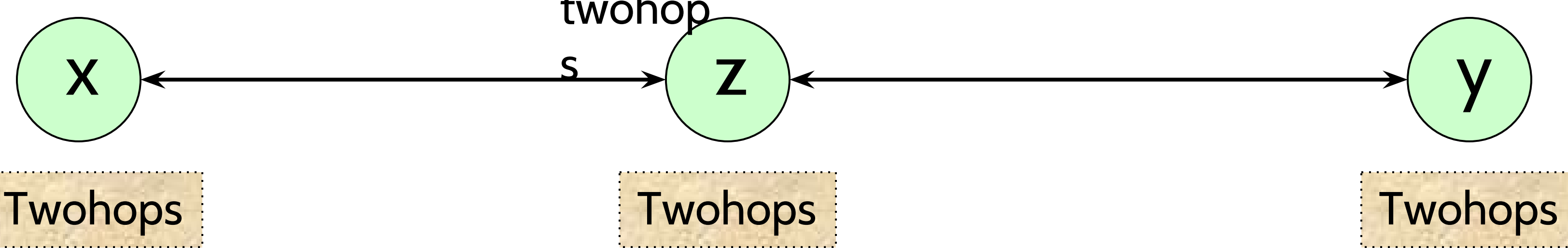
link

@Src	Dst	Cost
@z	y	c1
@z	x	c2

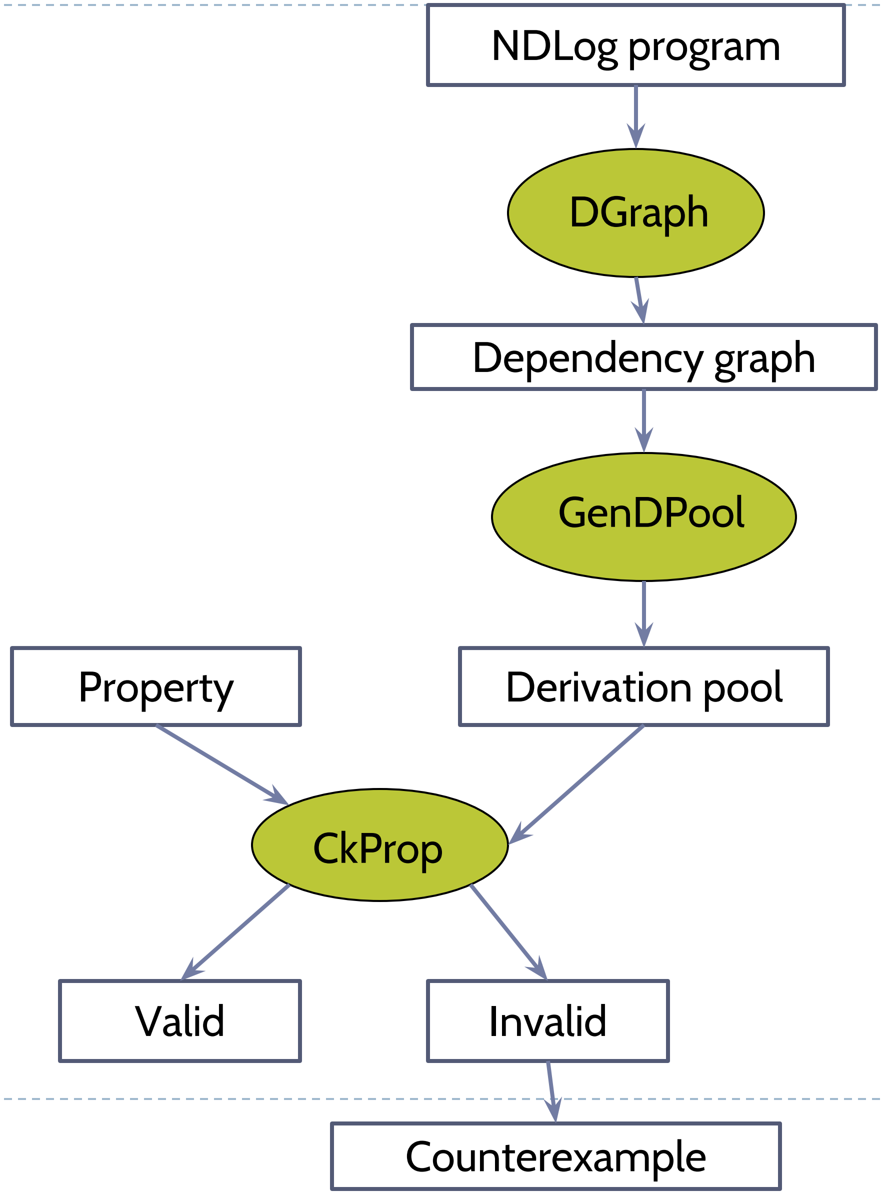
onehop

@Src	Dst	Cost
@z	x	c2

twohops



Overview of framework



Roadmap

- Introduction of NDLog (Network Datalog)
- Algorithm analysis
  - Derivation pool construction
  - Property query
  - Network constrains
  - Recursive Programs
- Case study
- Conclusion

NDLog program ⇒ Dependency graph

- Dependency Graph G:

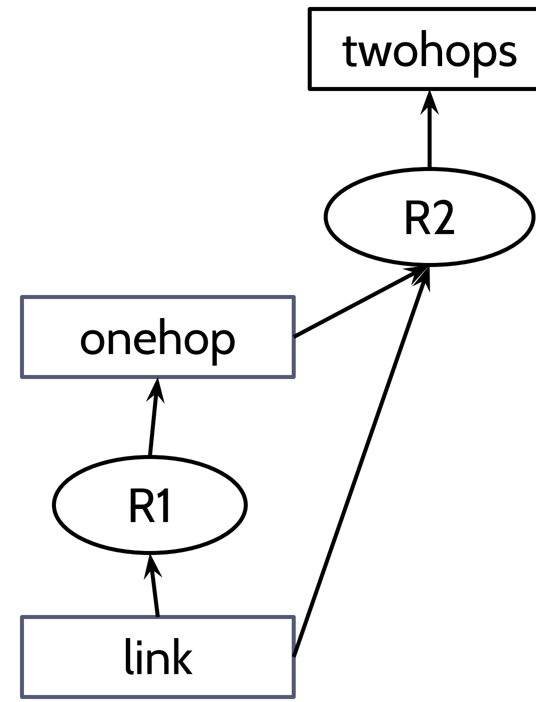
Vertex:

- V1: Tuple nodes
- V2: Rule nodes

Edge:

- (rule node → head node)
- (body node → rule node)

<u>Twohops</u>
R1 onehop(@z,x,c2) :- link(@z,x,c2)
R2 twohops(@x,y,c) :- link(@z,y,c1), onehop(@z,x,c2),
c=c1+c2



Dependency graph ⇒ Derivation pool

R1 onehop(@z,x,c2) :- link(@z,x,c2)
R2 twohops(@x,y,c) :- link(@z,y,c1), onehop(@z,x,c2), c=c1+c2

R1 onehop($@x_1,x_2,x_3$) :- link($@x_4,x_5,x_6$), $x_1=x_4 \wedge x_2=x_5 \wedge x_3=x_6$
R2 twohops($@x_7,x_8,x_9$) :- link($@x_{10},x_{11},x_{12}$), onehop($@x_{13},x_{14},x_{15}$), $x_7=x_{14} \wedge x_8=x_{11} \wedge x_{10}=x_{13} \wedge x_9=x_{12}+x_{15}$

	Link	Onehop
Derivations	(BaseTuple, link $(z_{\lambda1},z_{\lambda2},z_{\lambda3})$)	(R1, onehop(z_{o1},z_{o2},z_{o3}), (BaseTuple, link(z_{o4},z_{o5},z_{o6}))::nil)
Constraints	True	$z_{o1}=z_{o4} \wedge z_{o2}=z_{o5} \wedge z_{o3}=z_{o6}$

Twohops
(R2, twohops(z_{t7},z_{t8},z_{t9}), (BaseTuple,link$(z_{t10},z_{t11},z_{t12})$) ::(R1,onehop$(z_{t13},z_{t14},z_{t15})$, (BT,link$(z_{t3},z_{t4},z_{t5})$):: nil) ::nil)
$(z_{t3}=z_{t13} \wedge z_{t4}=z_{t14} \wedge z_{t5}=z_{t15}) \wedge (x_{t7}=x_{t14} \wedge x_{t8}=x_{t11} \wedge x_{t10}=x_{t13} \wedge x_{t9}=x_{t12}+x_{t15})$

Roadmap

- Introduction of NDLog (Network Datalog)
- Algorithm analysis
  - Derivation pool construction
  - Property query
  - Network constraints
  - Recursive programs
- Case study
- Conclusion

Property specification

- Safety property
  - Something bad never happens
- Restricted property format:
  - ◆ Indicates the temporal operation “past”

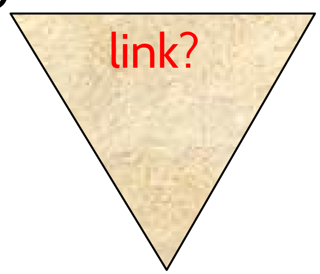
$$\forall \mathbf{x_1}.p_1(\mathbf{x_1}) \wedge \forall \mathbf{x_2}.p_2(\mathbf{x_2}) \wedge \ldots \wedge \forall \mathbf{x_n}.p_n(\mathbf{x_n}) \wedge c_q(\mathbf{x_1}, \ldots, \mathbf{x_n}) \supset \exists \mathbf{y_1}.\blacklozenge q_1(\mathbf{y_1}) \wedge \exists \mathbf{y_2}.\blacklozenge q_2(\mathbf{y_2}) \wedge \ldots \wedge \exists \mathbf{y_m}.\blacklozenge q_m(\mathbf{y_m}) \wedge c_q(\mathbf{x_1}, \ldots, \mathbf{x_n}, \mathbf{y_1}, \ldots, \mathbf{y_m})$$

- Example:
  - $\forall x_1,x_2,x_3.\mathrm{onehop}(x_1,x_2,x_3) \wedge x_3>0 \supset$
    $\exists y_1,y_2,y_3.\blacklozenge \mathrm{link}(y_1,y_2,y_3) \wedge y_3<0$

Property verification algorithm

- Verify the property holds for all possible derivations
  - Enumerate all derivations for the tuples in the antecedent

$$\varphi = \forall x_1, x_2, x_3.\ onehop(x_1,x_2,x_3) \wedge x_3>0 \supset \exists y_1,y_2,y_3.\ \blacklozenge link(y_1,y_2,y_3) \wedge y_3<0$$



onehop

Derivation of onehop *in derivation pool*

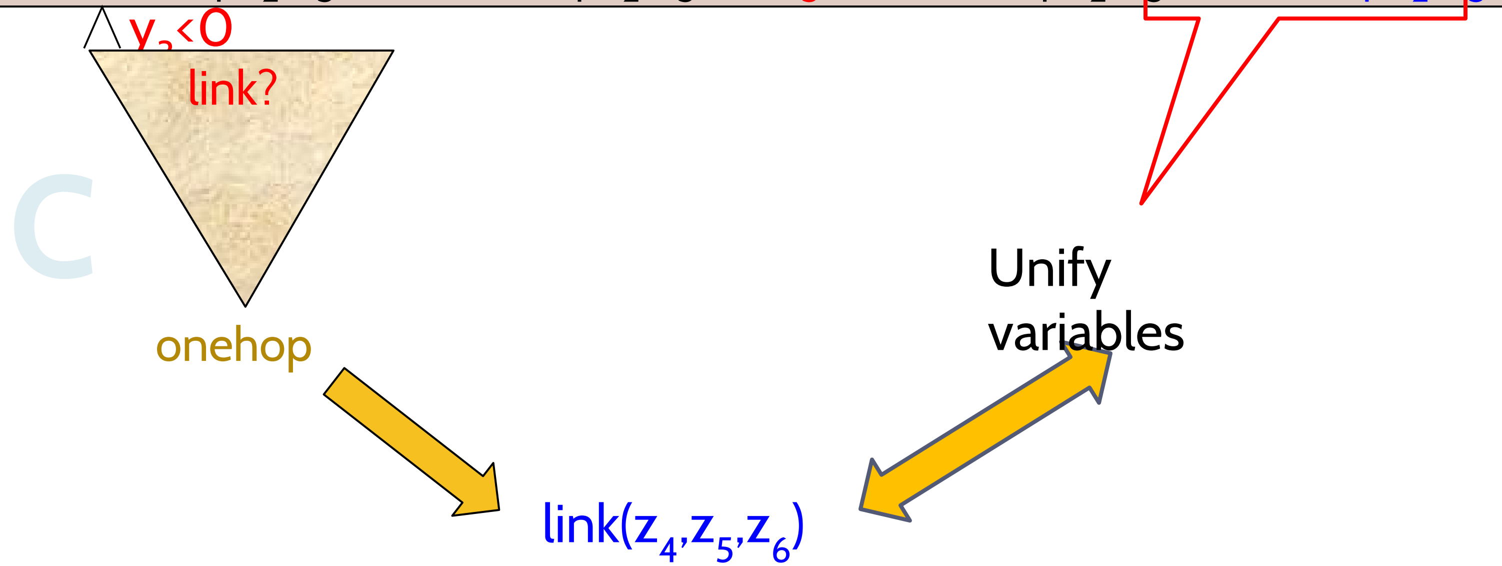
C

Constraint in derivation pool for the derivation on onehop

Property verification algorithm

- Verify existence of tuples in the conclusion
  - Look for instances of tuples in the given derivation

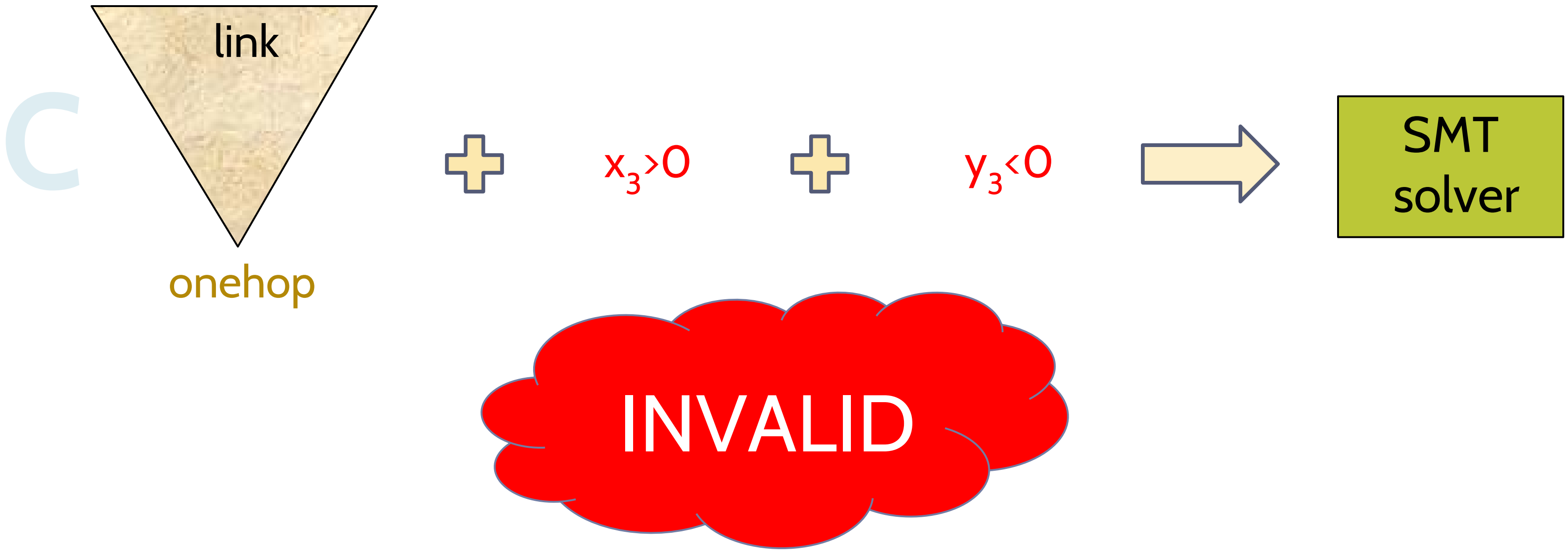
$$\varphi = \forall x_1,x_2,x_3.\ \text{onehop}(x_1,x_2,x_3) \wedge x_3>0 \supset \exists y_1,y_2,y_3.\ \blacklozenge \text{link}(y_1,y_2,y_3)$$



Property verification algorithm

- Verify validity of constraints
  - SMT solver

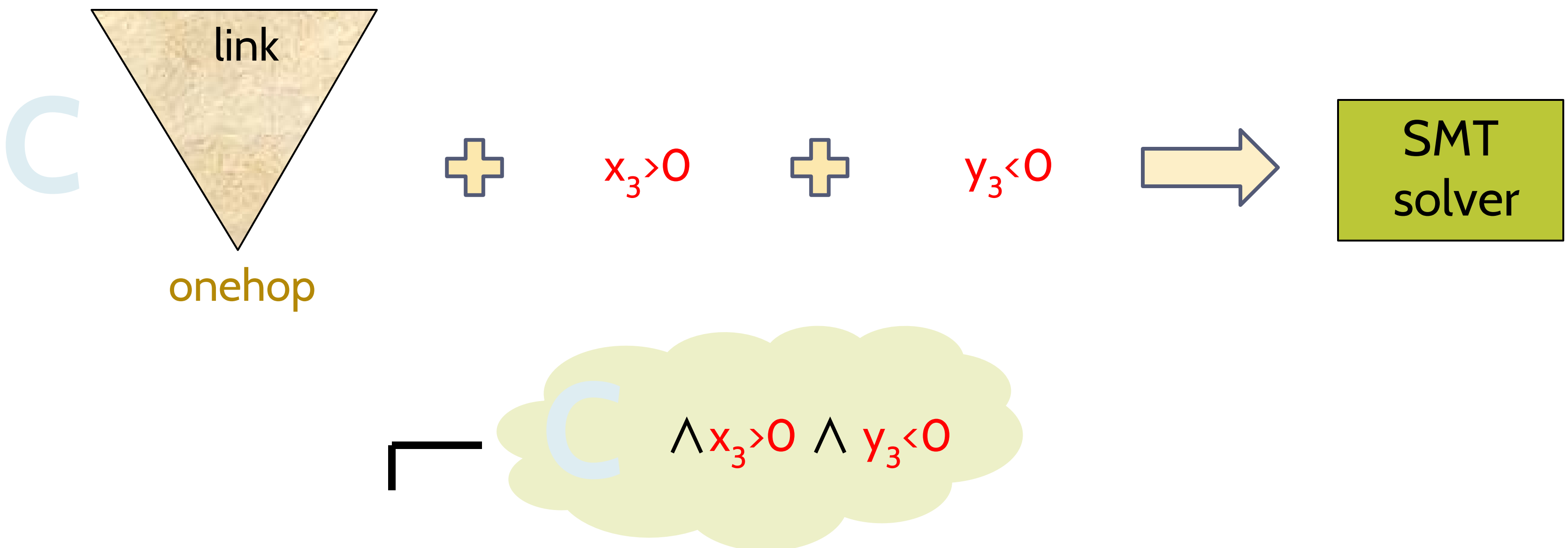
$\varphi = \forall x_1,x_2,x_3.\ \mathrm{onehop}(x_1,x_2,x_3) \wedge x_3>0 \supset \exists y_1,y_2,y_3.\ \blacklozenge \mathrm{link}(y_1,y_2,y_3)$
$\wedge\, y_3<0$



Property verification algorithm

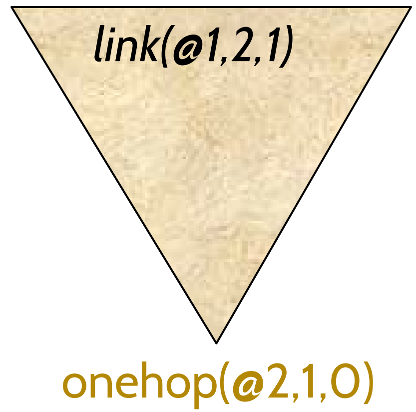
- Find a satisfying substitution for the negation of the constraints to generate a concrete counterexample

$$\varphi = \forall x_1,x_2,x_3.\ \mathrm{onehop}(x_1,x_2,x_3) \wedge x_3>0 \supset \exists y_1,y_2,y_3.\ \blacklozenge \mathrm{link}(y_1,y_2,y_3) \wedge y_3<0$$



Property verification algorithm

$$\varphi = \forall x_1,x_2,x_3.\ onehop(x_1,x_2,x_3) \wedge x_3>0 \supset \exists y_1,y_2,y_3.\ \blacklozenge link(y_1,y_2,y_3) \wedge y_3<0$$

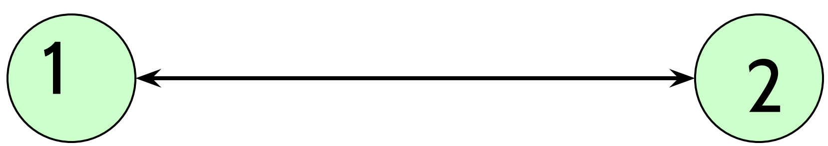


link

@Src	Dst	Cost
@1	2	1

onehop

@Src	Dst	Cost
@2	1	0

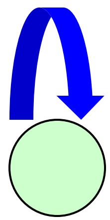


Roadmap

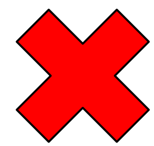
- Introduction of NDLog (Network Datalog)
- Algorithm analysis
  - Derivation pool construction
  - Property query
  - Network constraints
  - Recursive programs
- Case study
- Conclusion

Network constraint examples

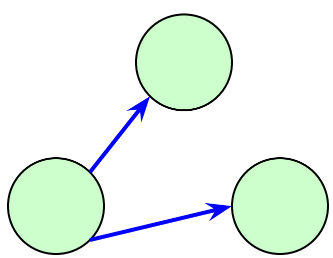
$$\varphi_{net} = \forall u_1,u_2,u_3.\ \mathrm{link}(u_1,u_2,u_3) \supset u_1 \neq u_2$$



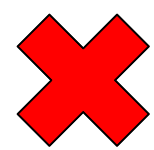
No self-loops



$$\varphi_{net} = \forall u_1,u_2,u_3.\ \mathrm{link}(u_1,u_2,u_3) \land \forall u_4,u_5,u_6.\ \mathrm{link}(u_4,u_5,u_6) \supset (u_1 = u_4 \rightarrow u_2 \neq u_5)$$



Every node in the network has only one outgoing link



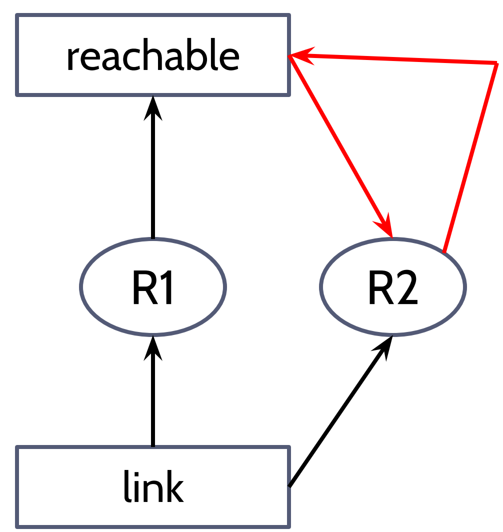
Roadmap

- Introduction of NDLog (Network Datalog)
- Algorithm analysis
  - Derivation pool construction
  - Property query
  - Network constraints
  - Recursive programs
- Case study
- Conclusion

Recursive programs

- Dependency graph has cycles
- Break the cycle using user-provided annotations on tuples on the cycle
  - Equivalent to disjunction of constraints over the list of possible derivations for tuples on the cycle

Reachability
R1 reachable(@x,y) :- link(@x,y)
R2 reachable(@x,y) :- link(@x,z), reachable(@z,y)

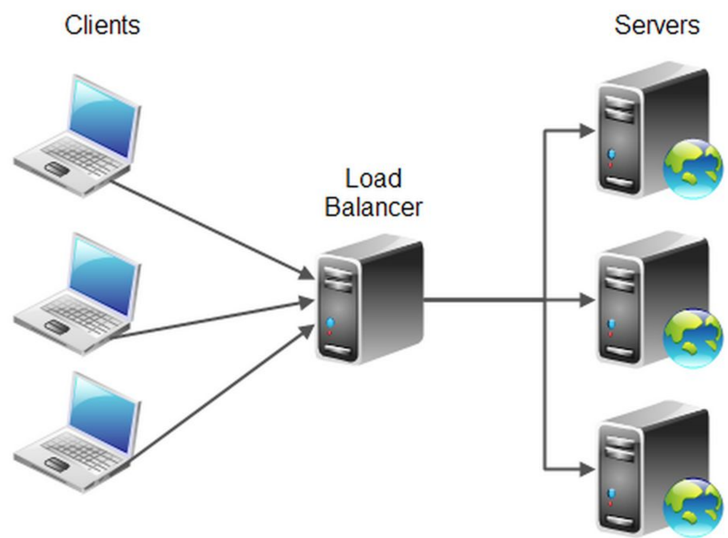


Roadmap

- Introduction of NDLog (Network Datalog)
- Algorithm analysis
  - Derivation pool construction
  - Property query
  - Network constraints
  - Recursive programs
- Case study
- Conclusion

Introduction to load balancers

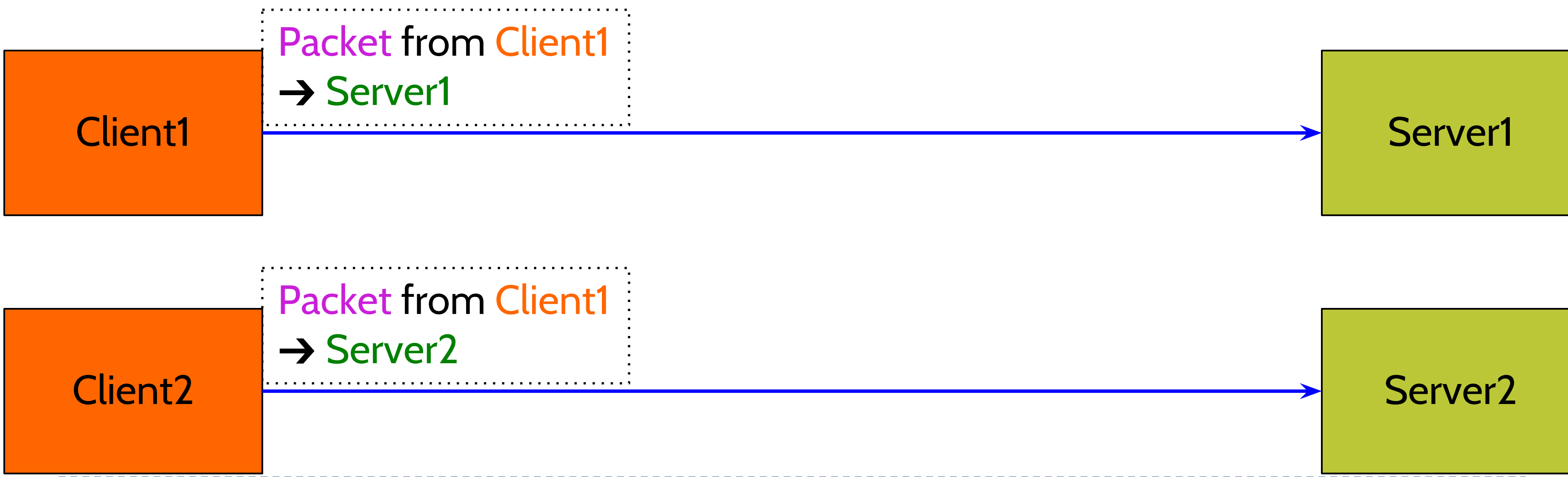
- A way to distribute client requests to an application onto multiple servers
  - Allows application to process a higher work load
  - Provides redundancy in an application
- Flow affinity
  - Packets received on different servers cannot share the same source address



Flow affinity

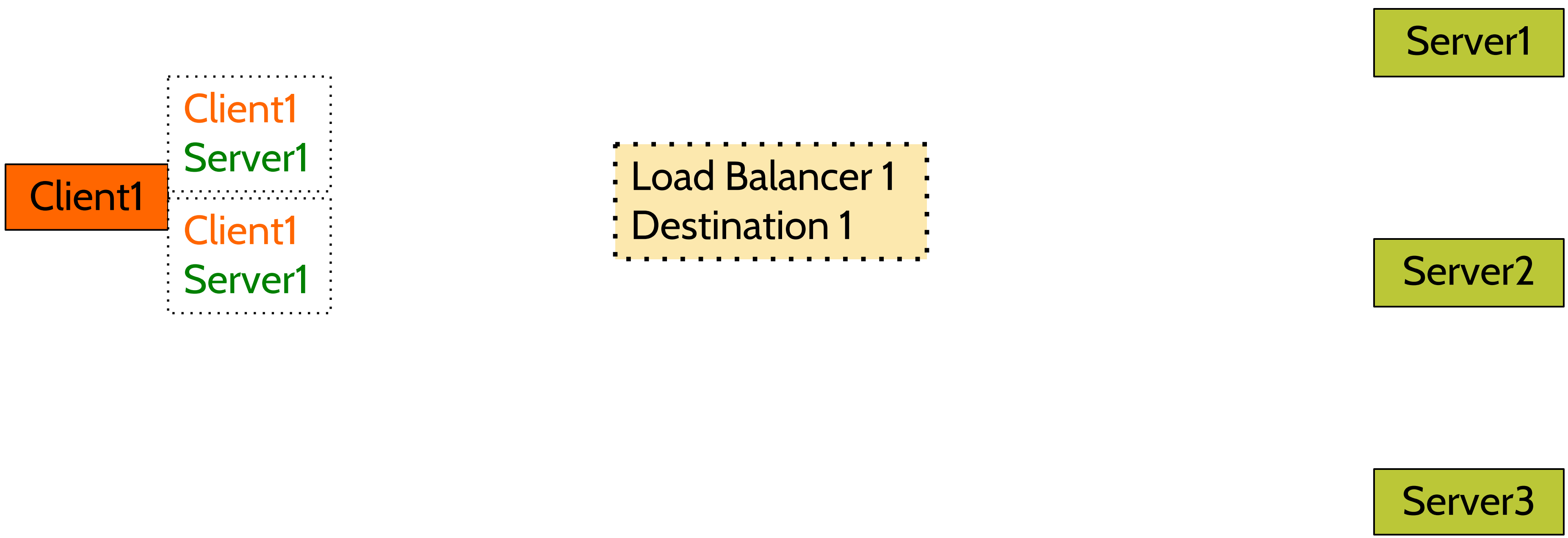
- Packets received on different servers cannot share the same cli

$\forall$ Server1, Client1, LoadBalancer1. $\forall$ Server2, Client2, LoadBalancer2.
recvPacket(Server1, Client1, LoadBalancer1)
$\wedge$ recvPacket(Server2, Client2, LoadBalancer2)
$\wedge$ Server1 $\neq$ Server2 $\supset$
Client1 $\neq$ Client2

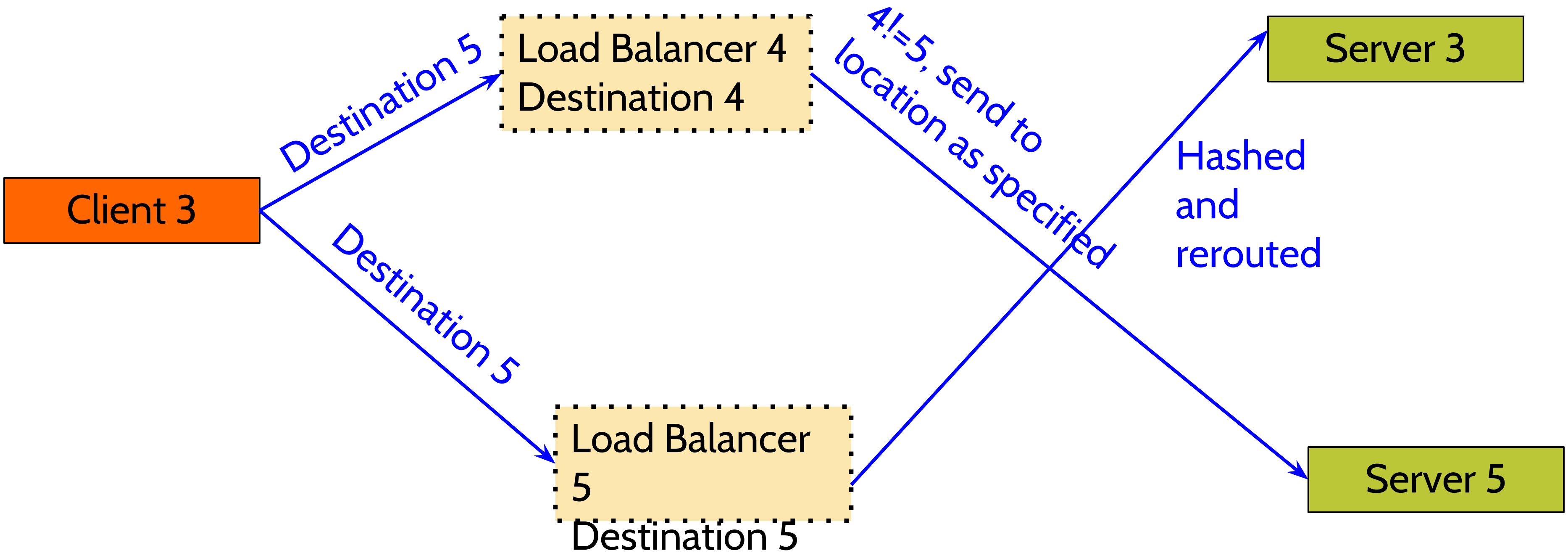


A naïve load balancer

- Our load balancer balances traffic towards a specific destination address
  - Determines the path of a packet based on the hash value of its source address



Counterexample produced by our tool



$\forall$ Server1, Client1, LoadBalancer1. $\forall$ Server2, Client2, LoadBalancer2.

recvPacket(Server1, Client1, LoadBalancer1)

$\wedge$ recvPacket(Server2, Client2, LoadBalancer2)

$\wedge$ Server1 $\neq$ Server2 $\supset$

Client1 $\neq$ Client2

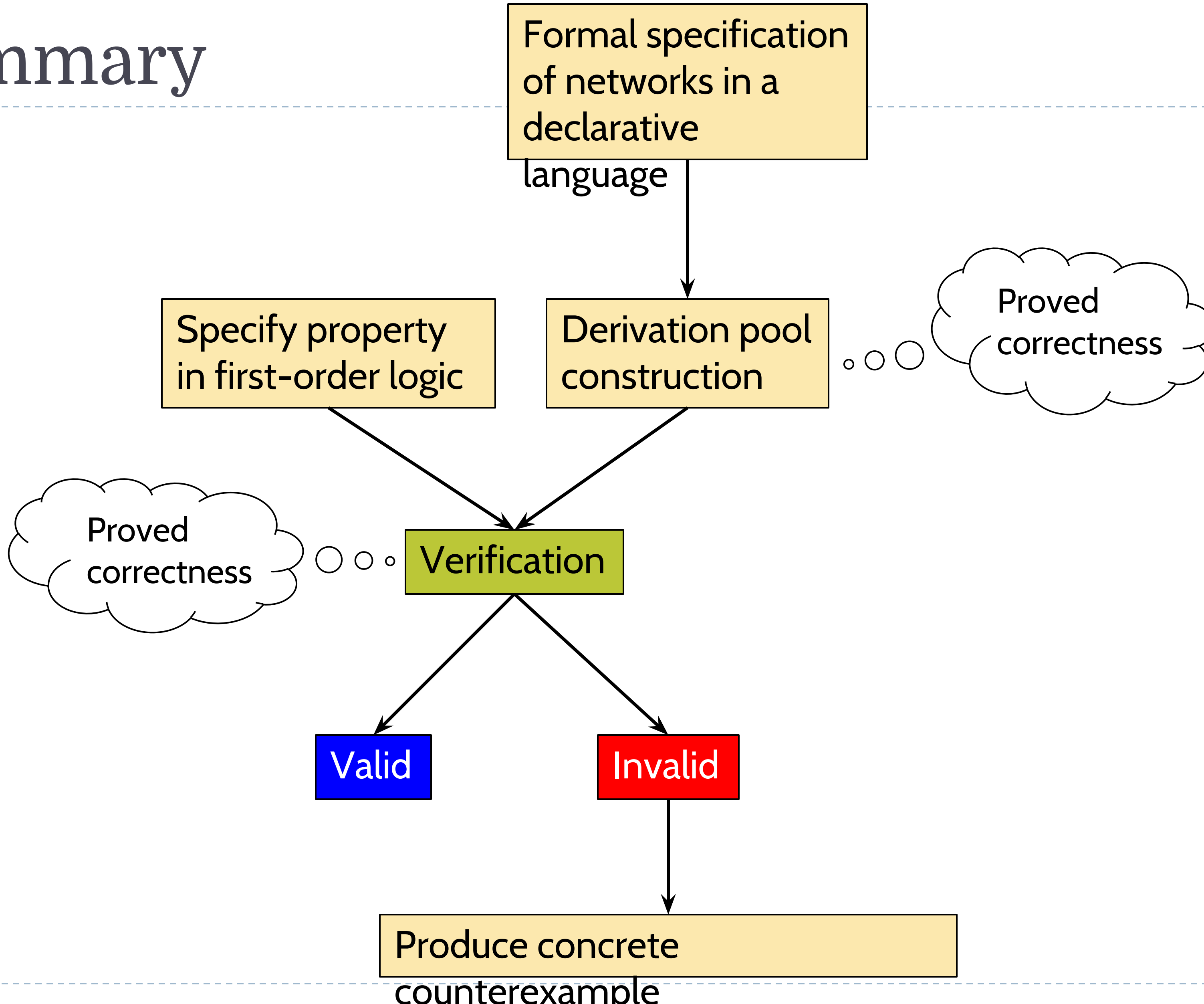
Case studies

Test Case	# rules	# properties	# counterexamples	Max eval time (ms)
Ethernet source learning	11	5	2	60
Firewall	5	3	0	40
Load balancer	4	1	1	80
Address resolution	9	2	0	210

Roadmap

- Introduction of NDLog (Network Datalog)
- Algorithm analysis
  - Derivation pool construction
  - Property query
  - Network constraints
  - Recursive programs
- Case study
- Conclusion

Summary



Formal specification of networks in a declarative language
Specify property in first-order logic
Derivation pool construction
Proved correctness
Proved correctness
Verification
Valid
Invalid
Produce concrete counterexample

Related work

- Network verification
  - Model network behavior using trace semantics. Relies on manual proofs
  - Examples: FORTE'14, TPHOLs'09
  - *Our solution: Enables automated static analysis of safety properties, generates counterexamples for debugging purposes*
- Software defined networking (SDN) verification
  - Specific to analyzing SDN controllers and data places
  - Examples: PLDI'14, SIGCOMM'11
  - *Our solution: Does the above, can also analyze other distributed systems expressible in NDLog*
- Verification of declarative programs
  - Proves correctness properties of networking protocols using theorem provers. User experience with theorem provers required.
  - Example: PADL'09
  - *Our solution: Validate protocol correctness using an SMT solver. User experience with SMT solvers unnecessary.*

Future work

- Analyze liveness properties
  - Something good eventually happens
- Provenance related topics

Questions?

Time complexity (non-recursive)

- Notation
  - $P = p_1,\ldots,p_n$
    - $|P|=n$
  - $Q = q_1,\ldots,q_m$
    - $|Q|=m$
  - Given an NDLog program with R rules
    - Each rule has at most W body tuples

$$\forall \mathbf{x}_1.p_1(\mathbf{x}_1) \wedge \forall \mathbf{x}_2.p_2(\mathbf{x}_2) \wedge \ldots \wedge \forall \mathbf{x}_n.p_n(\mathbf{x}_n) \wedge c_q(\mathbf{x}_1, \ldots, \mathbf{x}_n) \supset \exists \mathbf{y}_1.\blacklozenge q_1(\mathbf{y}_1) \wedge \exists \mathbf{y}_2.\blacklozenge q_2(\mathbf{y}_2) \wedge \ldots \wedge \exists \mathbf{y}_m.\blacklozenge q_m(\mathbf{y}_m) \wedge c_q(\mathbf{x}_1, \ldots, \mathbf{x}_n, \mathbf{y}_1, \ldots, \mathbf{y}_m)$$

- Time complexity: $O((R^{nW^{\wedge}R})n^m W^{Rn})$

- In practice, R and W are small
  - Can be treated as constants

Possible ways to fix the network counterexample

- Add network assumptions
  - Servers are connected to at most one load balancer
- Change property specification
  - Load balanced packets that are forwarded out of different load balancers were not sent out by the same client