Formally Verifiable Networking

Anduo Wang[1] Limin Jia[1] Changbin Liu[1]
Boon Thau Loo[1] Andre Scedrov[1]
Oleg Sokolsky[1] Prithwish Basu[2]

[1]University of Pennsylvania

[2]BBN technologies

Protocol eXchange Oct 7, 2009

Outline

Motivation

Overview

- Background on Declarative Networking

Verification in FVN

- NDLog Program Verification
- Verified Code Generation

Meta-Theoretic Model

- Compositional Routing Algebra

Conclusion

Open issues

Motivation

- Challenges to today's Internet
  - Unwanted and harmful traffic
  - Complexity and fragility in Internet routing

Motivation

- Challenges to today's Internet
  - Unwanted and harmful traffic
  - Complexity and fragility in Internet routing
- New applications demand new capabilities
  - Resiliency (RON, SOSR, Detour...)
  - Scalable Lookup (Chord, Pastry, Tapestry,...)
  - Mobility (i3, DHARMA, HIP)
  - Security (SOS, OverDoSe)
  - Content-distribution (Akamai, CoralCDN)
  - Multicast (Overcast, ESM)

Motivation

- Challenges to today's Internet
  - Unwanted and harmful traffic
  - Complexity and fragility in Internet routing
- New applications demand new capabilities
  - Resiliency (RON, SOSR, Detour...)
  - Scalable Lookup (Chord, Pastry, Tapestry,...)
  - Mobility (i3, DHARMA, HIP)
  - Security (SOS, OverDoSe)
  - Content-distribution (Akamai, CoralCDN)
  - Multicast (Overcast, ESM)
- Clean-slate Internet Design
  - NSF FIND (Future Internet Design)
  - GENI (Global Environment for Network Innovation)

Motivation

- Challenges to today's Internet
  - Unwanted and harmful traffic
  - Complexity and fragility in Internet routing
- New applications demand new capabilities
  - Resiliency (RON, SOSR, Detour...)
  - Scalable Lookup (Chord, Pastry, Tapestry,...)
  - Mobility (i3, DHARMA, HIP)
  - Security (SOS, OverDoSe)
  - Content-distribution (Akamai, CoralCDN)
  - Multicast (Overcast, ESM)
- Clean-slate Internet Design
  - NSF FIND (Future Internet Design)
  - GENI (Global Environment for Network Innovation)

Needed: Better software tools for deploying and analyzing new network protocols and architectures

Recent Efforts in Network Design and Verification

- Correct-by-construction Meta-model
  - Metarouting [SIGCOMM'05]
- Runtime verification
  - Pip [NSDI'06]
  - DS3 [NSDI'08]
- Model checking
  - MaceMC [NSDI'07] best paper
  - CMC [NSDI'04]

Limitations of Current Approaches

- Metarouting
  - Idealized model unlikely to be adapted to actual implementation

Limitations of Current Approaches

- Metarouting
  - Idealized model unlikely to be adapted to actual implementation
- Runtime verification
  - Incur additional runtime overhead
  - Non-exhaustive, limited class of properties
- Model checking network implementation
  - Require model extraction
  - State explosion problem.

Limitations of Current Approaches

- Metarouting
  - Idealized model unlikely to be adapted to actual implementation
- Runtime verification
  - Incur additional runtime overhead
  - Non-exhaustive, limited class of properties
- Model checking network implementation
  - Require model extraction
  - State explosion problem.
- Classical theorem proving
  - High initial investment in formal specification
  - Restricted to design and standard, decoupled from actual implementation

Formally Verifiable Networking

unifying the design, specification, implementation, and verification of networking protocols

- *Formal logical statements* specify the behavior and the properties of the protocol
- *Theorem proving* establishes correctness of formal system specification w.r.t network properties

Formally Verifiable Networking

unifying the design, specification, implementation, and verification of networking protocols

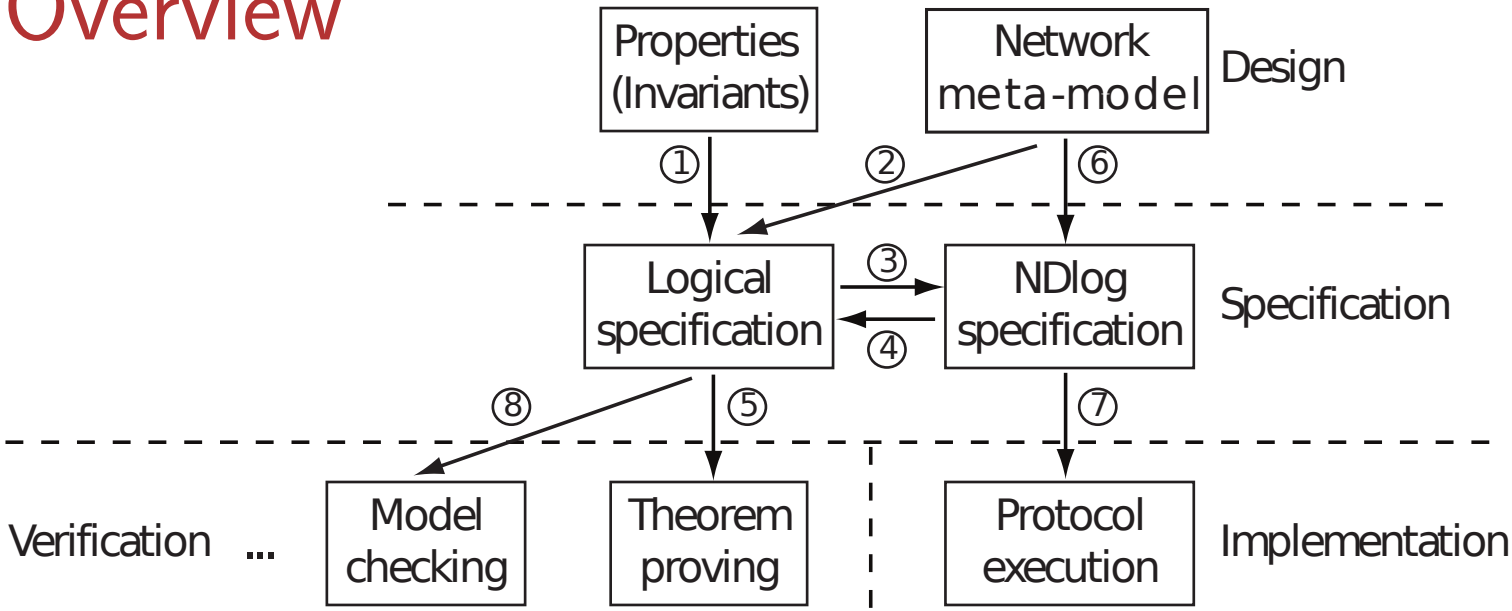
- *Formal logical statements* specify the behavior and the properties of the protocol
- *Theorem proving* establishes correctness of formal system specification w.r.t network properties
- Declarative networking, intermediary layer between logical specification and real implementation
  - Property preserving translations from declarative networking implementation to formal system specifications for verification
  - Code generation from verified formal specification

Formally Verifiable Networking

unifying the design, specification, implementation, and verification of networking protocols

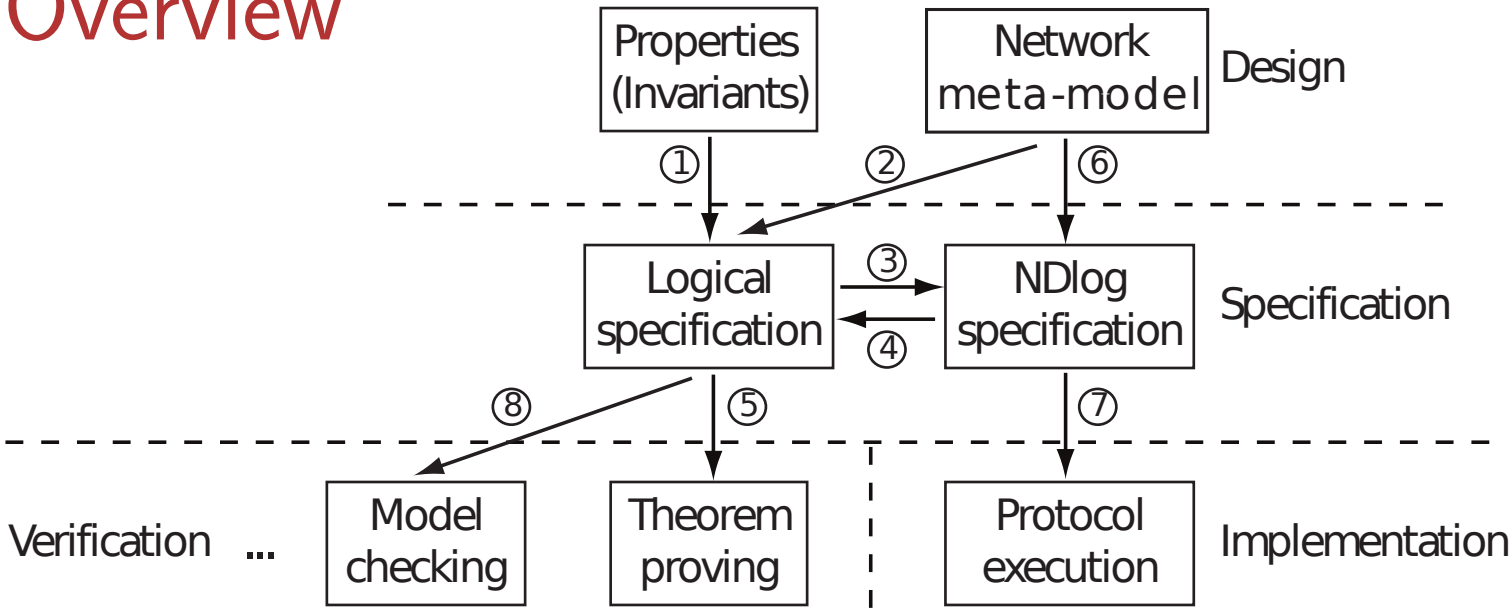
- *Formal logical statements* specify the behavior and the properties of the protocol
- *Theorem proving* establishes correctness of formal system specification w.r.t network properties
- Declarative networking, intermediary layer between logical specification and real implementation
  - Property preserving translations from declarative networking implementation to formal system specifications for verification
  - Code generation from verified formal specification
- Meta-model, correctness-by-construction design

FVN Overview



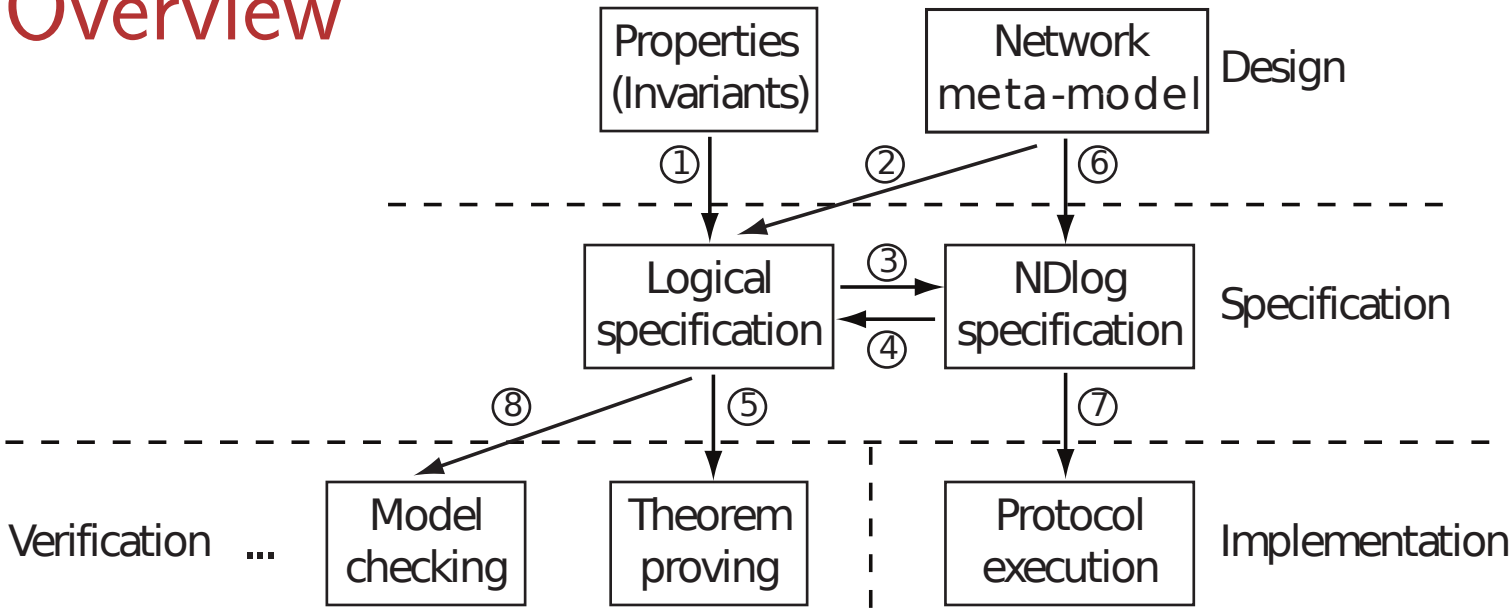
- Design: correctness-by-construction via meta-model

FVN Overview



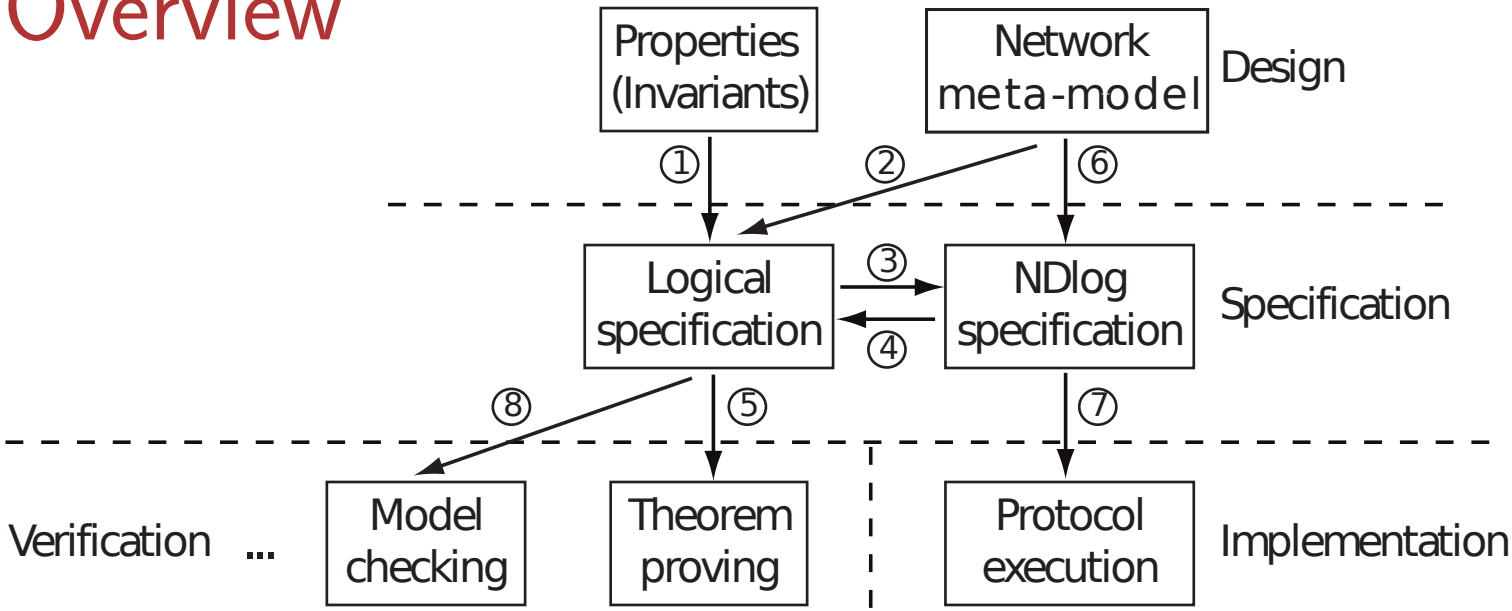
- Design: correctness-by-construction via meta-model
- Specification: two way property preserving translation
  - Formal system specification generated from NDlog program (arc 4)
  - Executable Declarative network synthesized from verified logical specification (arc 3)

FVN Overview



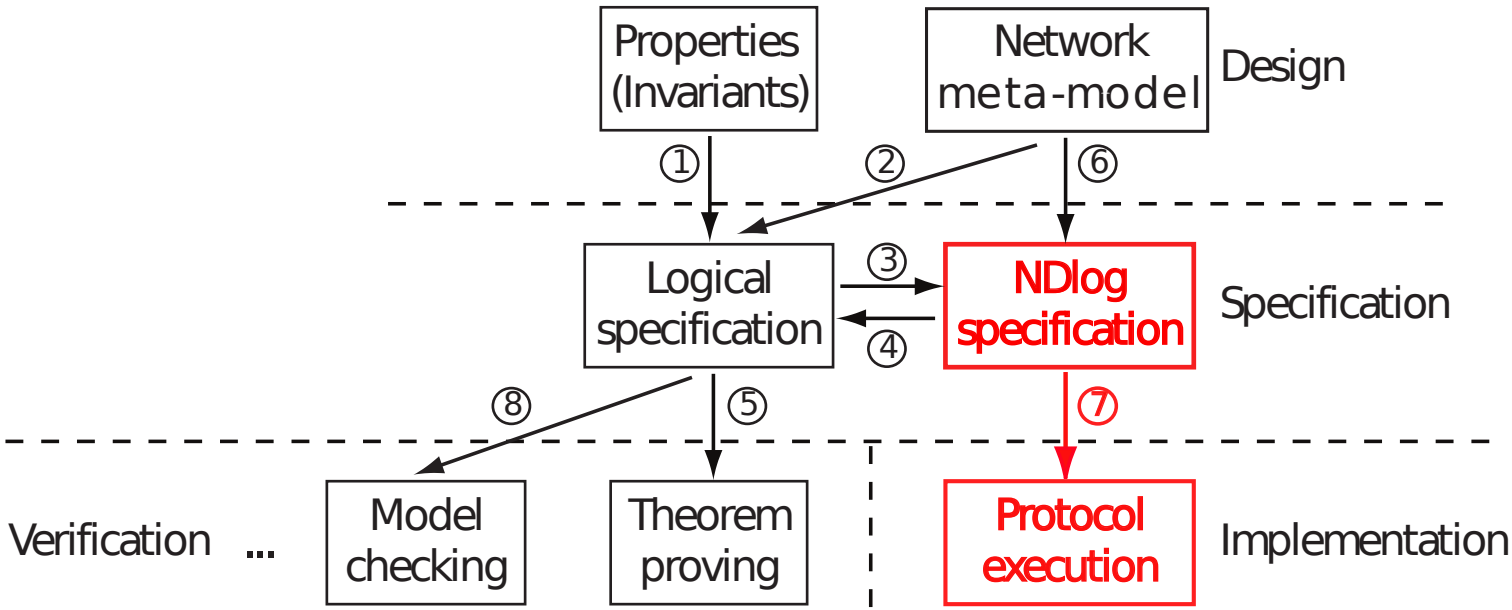
- Design: correctness-by-construction via meta-model
- Specification: two way property preserving translation
  - Formal system specification generated from NDlog program (arc 4)
  - Executable Declarative network synthesized from verified logical specification (arc 3)
- Verification: proving network invariants of system specifications by interacting with theorem prover (arc 5)

FVN Overview



- Design: correctness-by-construction via meta-model
- Specification: two way property preserving translation
  - Formal system specification generated from NDlog program (arc 4)
  - Executable Declarative network synthesized from verified logical specification (arc 3)
- Verification: proving network invariants of system specifications by interacting with theorem prover (arc 5)
- Implementation: distributed query processing (arc 7)

Declarative Networking



- Declarative specifications of networks using *Network Datalog* (NDLog), a distributed variant of Datalog
- NDLog is compiled to distributed dataflows (arc 7)
- Distributed query processor executes the dataflows to implement the network protocols

Declarative Networking, Cntd

- Ease of programming:
  - Compact high-level representation of protocols
  - Orders of magnitude reduction in code size
- Ease of analysis:
  - Amenable to static analysis and theorem proving
- See *Loo et. al* [SOSP '05, SIGMOD '06] for implementation details of declarative networking

Network Datalog (NDlog) by example
All-Pairs Reachability

```
R1:reachable(@S,D)<-link(@S,D)
R2:reachable(@S,D)<-link(@S,Z),reachable(@Z,D)
```

- *input*: `link(@S,D)`, *output*:`reachable(@S,D)`
- `link(@S,D)`:a link from node S to `D`, `reachable(@S,D)`: node `S` can reach `D`
- *Location specifier:* value of attribute prefixed with `@` determines the location of each tuple

Network Datalog (NDlog) by example
All-Pairs Reachability

- R1:reachable(@S,D)<-link(@S,D)
  R2:reachable(@S,D)<-link(@S,Z),reachable(@Z,D)
  - For all nodes S,D: S can reach D if there is a link from S to D
  - *input*: link(@S,D), *output*:reachable(@S,D)
  - link(@S,D):a link from node S to D, reachable(@S,D): node S can reach D
  - *Location specifier:* value of attribute prefixed with @ determines the location of each tuple

Network Datalog (NDlog) by example
All-Pairs Reachability

```
R1:reachable(@S,D)<-link(@S,D)
R2:reachable(@S,D)<-link(@S,Z),reachable(@Z,D)
```

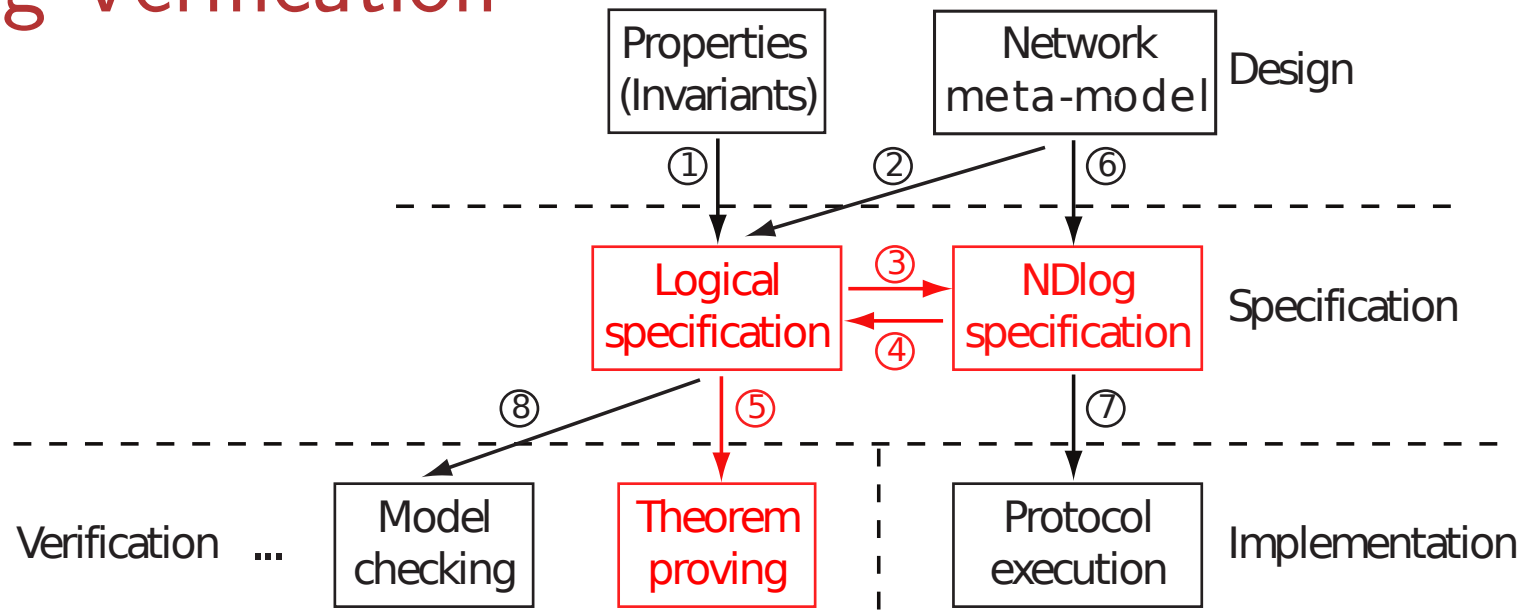
- For all nodes S,D,Z: if there is a link from S to Z, and that Z can reach D, then S can reach D

- *input*: link(@S,D), *output*:reachable(@S,D)
- link(@S,D):a link from node S to D, reachable(@S,D): node S can reach D
- *Location specifier:* value of attribute prefixed with @ determines the location of each tuple

Declarative Networking in Practice

- Example implementations to date:
  - Wired and wireless routing protocols (DV, LS, DSR, AODV, OLSR, etc.) [SIGCOMM'05, ICNP'09]
  - Chord Distributed Hash Table [SOSP'05]
  - Resilient overlay network (RON) [CoNEXT'08]
  - Internet Indirection Infrastructure (i3) [CoNEXT'08]
  - Others: sensor networking protocols [Sensys'07], multicast overlays, replication, snapshot, fault tolerance
- P2 declarative networking system
  - http://p2.cs.berkeley.edu
- A declarative toolkit for rapid network protocol simulation and experimentation [SIGCOMM'09 demo]
  - http://netdb.cis.upenn.edu/rapidnet/

NDlog Verification



- Automatic property preserving translations (arc 4): from declarative networking implementation to formal system specifications recognizable by existing theorem provers
- Verifying network properties (arc 5) by proving invariants against formal system specifications using theorem prover
- Automatic NDlog code generation (arc 3) from verified component specification

Example NDlog Program Verification

- Path-Vector Protocol in NDlog program
  - `p1 path(@S,D,P,C):- link(@S,D,C),P=(S,D).`
  - `p2 path(@S,D,P,C):- link(@S,Z,C1), path(@Z,D,P2,C2),C=C1+C2, P=concatPath(Z,P2).`
- Encoding path-vector in theorem prover
  - p1: $\forall(S, D, P, C).link(S, D, C) \wedge P = f_{init}(S, D) \implies path(S, D, P, C)$
  - p2: $\forall(S, D, P, C).\exists(C_1, C_2, Z, P_2).link(S, Z, C_1) \wedge bestPath(Z, D, P_2, C_2) \wedge C = C_1 + C_2 \wedge P = f_{concatPath}(Z, P_2) \implies path(S, D, P, C)$
- Proving Route optimality property in PVS
  - `FORALL (S,D:Node) (C:Metric) (P:Path): bestPath(S,D,P,C) => NOT (EXISTS (C2:Metric) (P2:Path): path(S,D,P2,C2) AND C2<C)`
  - `("" (skosimp*) (expand bestPath) (prop) (expand bestPathCost) (prop) (skosimp*) (inst -2 C2!1) (grind))`

Handling soft-state in networks

- Soft-state: network state expires after Time-To-Live (TTL) unless refreshed
- Ensures eventual consistency in protocol in the presence of message reordering and/or losses
- Additional rewrite step required for rules that uses soft-state predicates
  - *Declarative Network Verification*., Anduo Wang, Prithwish Basu, Boon Thau Loo, Oleg Sokolsky., University of Pennsylvania, Technical Report No. MS-CIS-08-34, 2008

Distance Vector Routing Protocol with Soft-State

- Distance vector routing:
  - NDLog specifications similar to path-vector routing except only next hop (instead of entire path) is traversed
- An instance of a soft-state NDLog program
  - Nodes periodically advertise to their neighbors their best known distances to other destinations
  - Nodes use these advertisements to select the best neighbor along the shortest path to destination
  - Advertisements timed-out unless refreshed

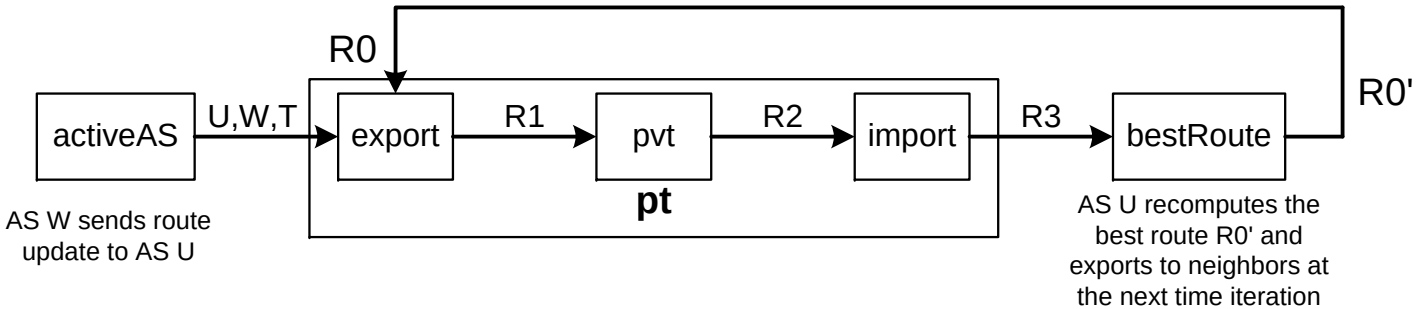
Example Properties Verified using Soft-State
Distance Vector Protocol

- Eventual convergence in stable network

```
bestHopCost_converge: THEOREM EXISTS (j:posnat):
  FORALL (S,D:Node)(C:Metric)(i:posnat): (i>j)
   => bestHopCost(S,D,C,5*i,10)
      = bestHopCost(S,D,C,5*j,10)
```

- Divergence (*count-to-infinity problem*) in dynamic network
- A well known solution: *split-horizon* can avoid count-to-infinity in two-node cycle, but cannot prevent the problem in three-node cycle

Component Based Verification of BGP System



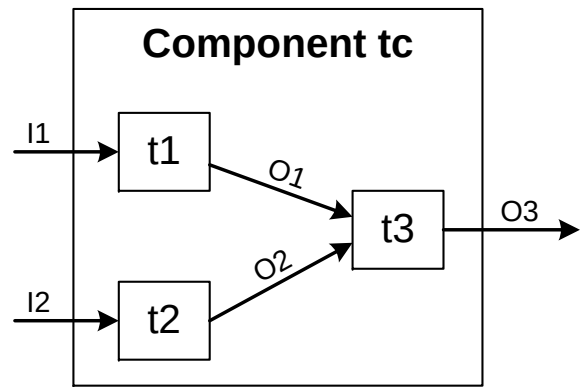
- Specification of BGP components in PVS
  - `bgp(U,W,R0,R3,T): INDUCTIVE bool = EXISTS (R1,R2): activeAS(U,W,T) AND pt(U,W,R0,R3,T) AND bestRoute(W,T,R0)`
  - `pt(U,W,R0,R3,T): INDUCTIVE bool = export(U,W,R0,R1,T) AND pvt(U,W,R1,R2,T) AND import(U,W,R2,R3,T)`
- More details on the PVS specifications and example proofs of various BGP systems
- *Verifiable Policy-based Routing with DRIVER.*, Anduo Wang, Changbin Liu, Boon Thau Loo, Oleg Sokolsky, Prithwash Basu., University of Pennsylvania TR, MS-CIS-09-12, 2009.

Generating Equivalent NDlog implementation

- Specify atomic component t as a rule taking t_in(I) as rule body, deriving t_out(O) as rule head
  - PVS specification `t(I,O):INDUCTIVE bool = CT(I,O)`
  - The equivalent NDlog rule `t_out(O) :- t_in(I), CT(I,O)`
- Specify compositional component as set of rules

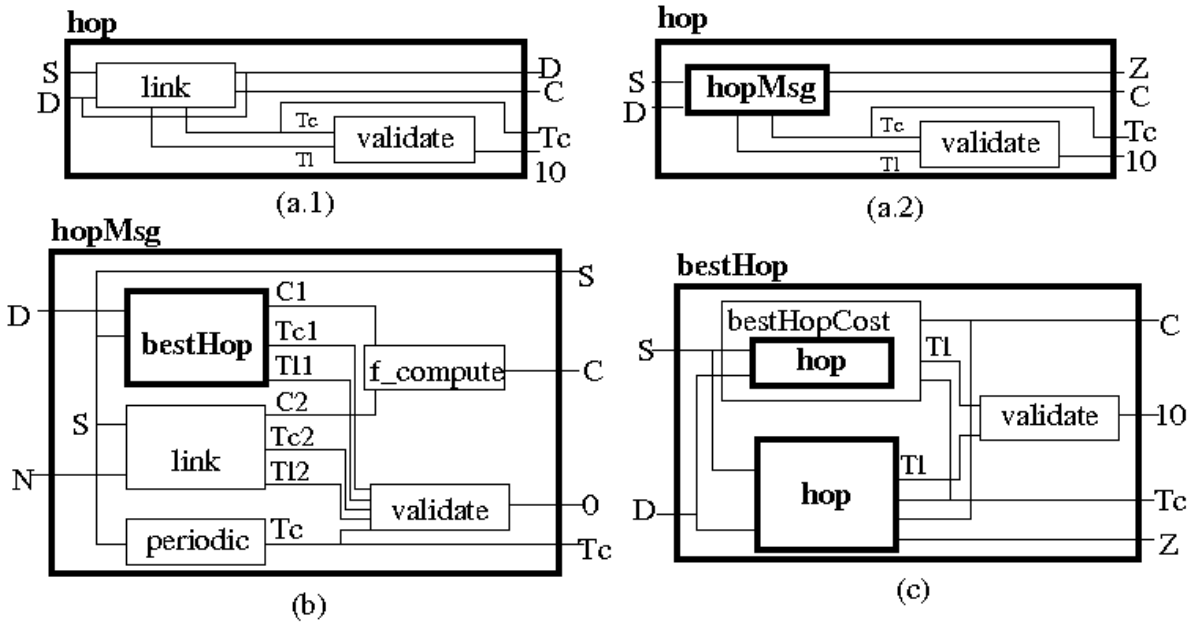
```
tc(I1,I2,O3): INDUCTIVE bool = EXISTS
(O1,O2):t1(I1,O1) AND t2(I2,O2) AND
t3(O1,O2,O3)

t1(I,O): INDUCTIVE bool = C1(I,O)
t2(I,O): INDUCTIVE bool = C2(I,O)
t3(I,O',O): INDUCTIVE bool = C3(I,I',O)
```



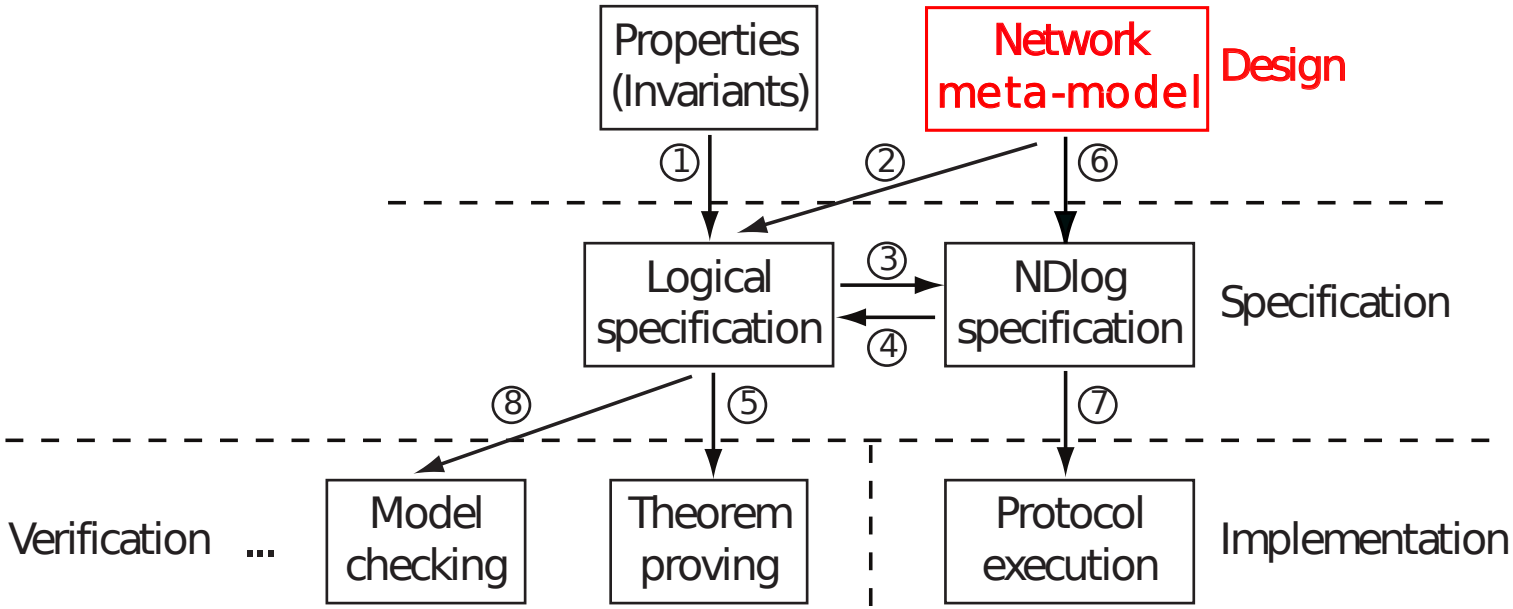
- Equivalent NDlog implementations
  - ```
    t1_out(O1) :- t1_in(I1), C1(I1,O1).
    t2_out(O2) :- t2_in(I2), C2(I2,O2).
    t3_out(O3) :- t1_out(O1), t2_out(O2), C3(O1,O2,O3).
    ```
    ```

Circuit-Like Component Representation of NDlog Program




```
a1: hop(@S,D,D,C,Tc,10):-link(@S,D,C,Tc,10).
a2: hop(@S,D,Z,C,Tc,10):-hopMsg(@S,D,Z,C,Tc2),Tc=Tc2+5
agg bestHopCost(@S,D,min<C>,Tc,10):- hop(@S,D,D,C,Tc,10).
b: hopMsg(@N,D,Z,C,Tc,0):- periodic_dv(@S,5,Tc), link(@S,N,C2,Tc2,10),
bestHop(@S,D,Z,C1,Tc1,10),C=C1+C2,Tc2<Tc<=Tc2+10,Tc1<Tc<=Tc1+10.
c: bestHop(@S,D,Z,C,Tc,10):- bestHopCost(@S,D,C,Tc,10),
hop(@S,D,Z,C,Tc1,10), Tc1<Tc<=Tc1+10.
```

Correctness-By-Construction via Metarouting



- Metarouting, algebraic framework for routing protocol
  - Models BGP systems (today's de facto Internet routing) with convergence guarantee

Correct-by-construction via Metarouting, Cntd

- Our contribution: Formalize fragment of Metarouting theory in FVN using PVS
  - Heavy and interesting use of PVS theory interpretation: mapping and declaration
  - Extend PVS specification logic with metarouting theory
- Our goal: Free network operator from the tedious low-level and trivial theory consistency checking by leveraging PVS specification language and proof environment

Background on BGP system

- *Internet*, network of Autonomous Systems (AS) administrated by Internet Service Provider (ISP)
- *Routing Protocol* computes reachability information
- Internet routing is a combination of Internal Gateway Protocol (IGP) and External Gateway Protocol (EGP)
- Border Gateway Protocol (BGP)
  de facto Internet routing
- BGP is policy based, ISP can influence route decision for economical or performance reasons
  - Import policies select routes to accept
  - Export policies decide routes to be advertised
- BGP is NOT ideal: no convergence guarantee

Metarouting

Algebraic framework for modeling BGP systems

- Abstract routing algebra, mathematical model: $A = \langle \Sigma, \preceq, \mathcal{L}, \oplus, \mathcal{O}, \phi \rangle$

  sorts Σ (paths), $\mathcal{L}$ (links)

  opns $\preceq : \Sigma \times \Sigma \rightarrow bool$ (preference relation)

  $\oplus : \mathcal{L} \times \Sigma \rightarrow \Sigma$ (label application function)

  $\mathcal{O} : subset \quad of \quad \mathcal{L}$ (origination set)

  $\phi : \Sigma$ (prohibited path)

  axioms $\forall_{\alpha \in \Sigma - \{\phi\}} \quad \alpha \preceq \phi \quad (Maximality)$

  $\forall_{l \in \mathcal{L}} \quad l \oplus \phi = \phi \quad (Absorption)$

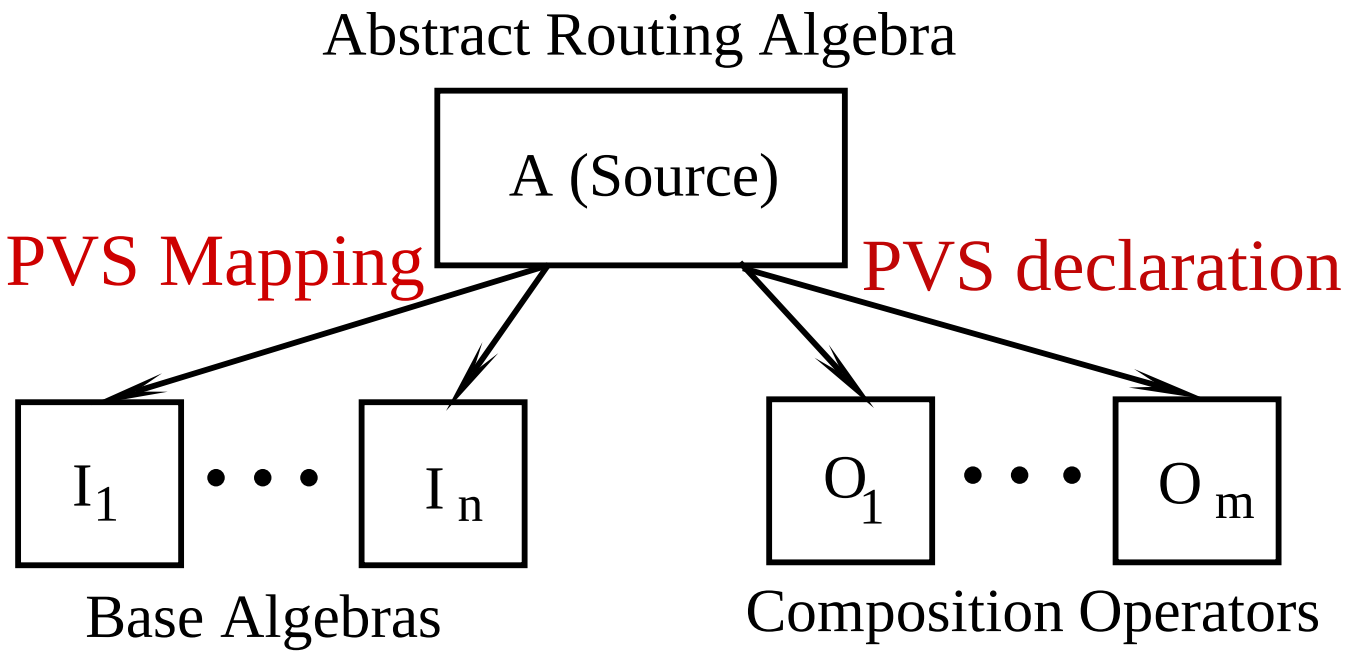
  $\forall_{l \in \mathcal{L}} \forall_{\alpha \in \Sigma} \quad \alpha \preceq l \oplus \alpha \quad (Monotonicity)$

  $\forall_{l \in \mathcal{L}} \forall_{\alpha, \beta \in \Sigma} \quad \alpha \preceq \beta \implies l \oplus \alpha \preceq l \oplus \beta \quad (Isotonicity)$

- Base algebras, building blocks for shortest path routing, etc
- Lexical product for route selection, composition operator

Isotonicity and Monotonicity: sufficient conditions for protocol convergence

Overview of PVS theories



- A: uninterpreted source theory `routeAlgebra`
- I_i: interpreted theory instantiated from A
- O_i: PVS theory taking routing algebra theories as parameters

Abstract Routing Algebra in PVS

```
routeAlgebra: THEORY
 BEGIN
  sig: TYPE+
  label: TYPE+
```

Abstract Routing Algebra in PVS

routeAlgebra: THEORY
BEGIN
sig: TYPE+
label: TYPE+
injected: [label → bool]
org: TYPE = {l: label | injected(l)}
prohibitPath: sig
labelApply: [label, sig → sig]
prefRel: [sig, sig → bool]
eqRel(s_1, s_2: sig): bool = prefRel(s_1, s_2) ∧ prefRel(s_2, s_1)
mono(l: label, s: sig): bool = prefRel(s, labelApply(l, s))

Abstract Routing Algebra in PVS

```
routeAlgebra: THEORY
 BEGIN
  sig: TYPE+
  label: TYPE+
  injected: [label → bool]
  org: TYPE = {l: label | injected(l)}
  prohibitPath: sig
  labelApply: [label, sig → sig]
  prefRel: [sig, sig → bool]
  eqRel(s1, s2: sig): bool = prefRel(s1, s2) ∧ prefRel(s2, s1)
  mono(l: label, s: sig): bool = prefRel(s, labelApply(l, s))
  pref_complete: AXIOM
    ∀ (x, y: sig): prefRel(x, y) ∨ prefRel(y, x)
  absorption: AXIOM
    ∀ (l: label): labelApply(l, prohibitPath) = prohibitPath
  maximality: AXIOM ∀ (s: sig): prefRel(s, prohibitPath)
  monotonicity: AXIOM ∀ (l: label, s: sig): mono(l, s)
  isotonicity: AXIOM
    ∀ (s1, s2: sig)(l: label):
      prefRel(s1, s2) ⇒
       prefRel(labelApply(l, s1), labelApply(l, s2))
 END routeAlgebra
```

Base Algebra for Shortest Path Routing

PVS mapping: Abstract Algebra `routeAlgebra` → Base Algebra `addA`

- PVS mapping makes instantiations of uninterpreted types

$$\begin{aligned} \texttt{sig} &\leftarrow \texttt{upto}(\texttt{m}+1) \\ \texttt{label} &\leftarrow \texttt{upto}(\texttt{n}) \\ \texttt{prohibitPath} &\leftarrow \texttt{m}+1 \\ \texttt{labelApply} &\leftarrow \texttt{APPLY} \\ \texttt{prefRel} &\leftarrow \texttt{PREF} \end{aligned}$$

- PVS mapping generates instances of routeAlgebra axioms as Type Correctness Conditions (*TCCs*)

```
IMP_A_monotonicity_TCC1: OBLIGATION
    FORALL (l: LABEL, s: SIG): mono(l, s)
```

Shortest Path Routing in PVS

Source Theory: Abstract Algebra `routeAlgebra`

Interpreted Theory: Base Algebra `addA`

```
addA: THEORY
 BEGIN
  n: posnat
  m: posnat
  redundant: posnat
  N_M: AXIOM n < m
  LABEL: TYPE = upto(n)
  SIG: TYPE = upto(m + 1)
  PREF(s1, s2: SIG): bool = (s1 ≤ s2)
  APPLY(l: LABEL, s: SIG): SIG =
      IF (l + s < m + 1)
        THEN (l + s)
      ELSE (m + 1)
      ENDIF
  IMPORTING routeAlgebra
                {{sig := SIG, label := LABEL, prohibitPath := m + 1,
                  labelApply(l: LABEL, s: SIG) := APPLY(l, s),
                  prefRel(s1, s2: SIG) := (s1 ≤ s2)}}
 END addA
```

Base Algebra for Provider-Customer, Peer-Peer Guideline

- For economical reasons, ISP reduces use of provider routes, and maximizes availability of customer routes
- $\Sigma(path)$: $C / R / P$ (customer/peer/provider path)
- $\mathcal{L}(link)$: $c / r / p$ (customer/peer/provider link)
- $\oplus$ (label application):

$\oplus$	C	R	P
c	C	C	C
r	R	R	R
p	P	P	P

- $\preceq$ (preference relation): $C \preceq R, \quad R \preceq P, \quad C \preceq P$

Provider-Customer, Peer-Peer Guideline in PVS

For simplicity, rename labels and signatures:
$c \leftarrow 1, r \leftarrow 2, p \leftarrow 3$ and $C \leftarrow 1, R \leftarrow 2, P \leftarrow 3$

```
lpA: THEORY
 BEGIN

  SIG: TYPE = upto(3)

  LABEL: TYPE = upto(3)

  IMPORTING routeAlgebra
                {{sig := SIG, label := LABEL,
                labelApply(l: LABEL, s: SIG) := l,
                prefRel(s1, s2: SIG) := (s1 <= s2),}}

 END lpA
```

Lexical Product $\otimes$ and Route Selection

- Lexicographic comparison models route selection
  - Most important attribute of each route is compared first, if no decision is reached, the next attribute is considered
- Lexical Product $A \otimes B$ built from existing algebras: A, B
  - Models a routing protocol with multiple attributes
  - More important attributes are handled by A, and the less important by B

Lexical Product $A \otimes B$ in PVS

PVS *declaration* and *mapping* ensures resulting algebra $A \otimes B$ is a valid routing algebra, i.e. $\otimes$ is closed under abstract routing algebra

```
lexProduct[A: THEORY routeAlgebra, B: THEORY routeAlgebra]: THEORY
 BEGIN

  SIG: TYPE = [A.sig, B.sig]

  LABEL: TYPE = [A.label, B.label]

  APPLY(l: LABEL, s: SIG): SIG =
      (A.labelApply(l`1, s`1), B.labelApply(l`2, s`2))

  PREF(s1, s2: SIG): bool =
      A.prefRel(s1`1, s2`1) ∨
       (A.eqRel(s1`1, s2`1) ∧ B.prefRel(s1`2, s2`2))

  IMPORTING routeAlgebra
                  {{sig := SIG, label := LABEL,
                    labelApply(l: LABEL, s: SIG) := APPLY(l, s),
                    prefRel(s1, s2: SIG) := PREF(s1, s2)}}

 END lexProduct
```

A Concrete BGP system

- Route paths are measured in terms of customer-provider relationship and distance cost
  - Customer-Provider Peer-Peer guideline must be enforced
  - Once customer-provider policy is satisfied, ISP wants least-cost (shortest) paths
- Decompose this BGP system into two sub-components
  - Sub-component A for customer-provider guideline
  - Sub-component B for shortest-path
  - Check the sub-component A first, and only use B to break tie
- $BGPsystem : \text{THEORY} = \text{lexProduct}\left[A_2, \ B_2\right]$

Concrete BGP system in PVS
Sub-components Instantiated from Base Algebras

```
AlgebraInstance: THEORY
 BEGIN

  IMPORTING addA{{n := 16, m := 16}}

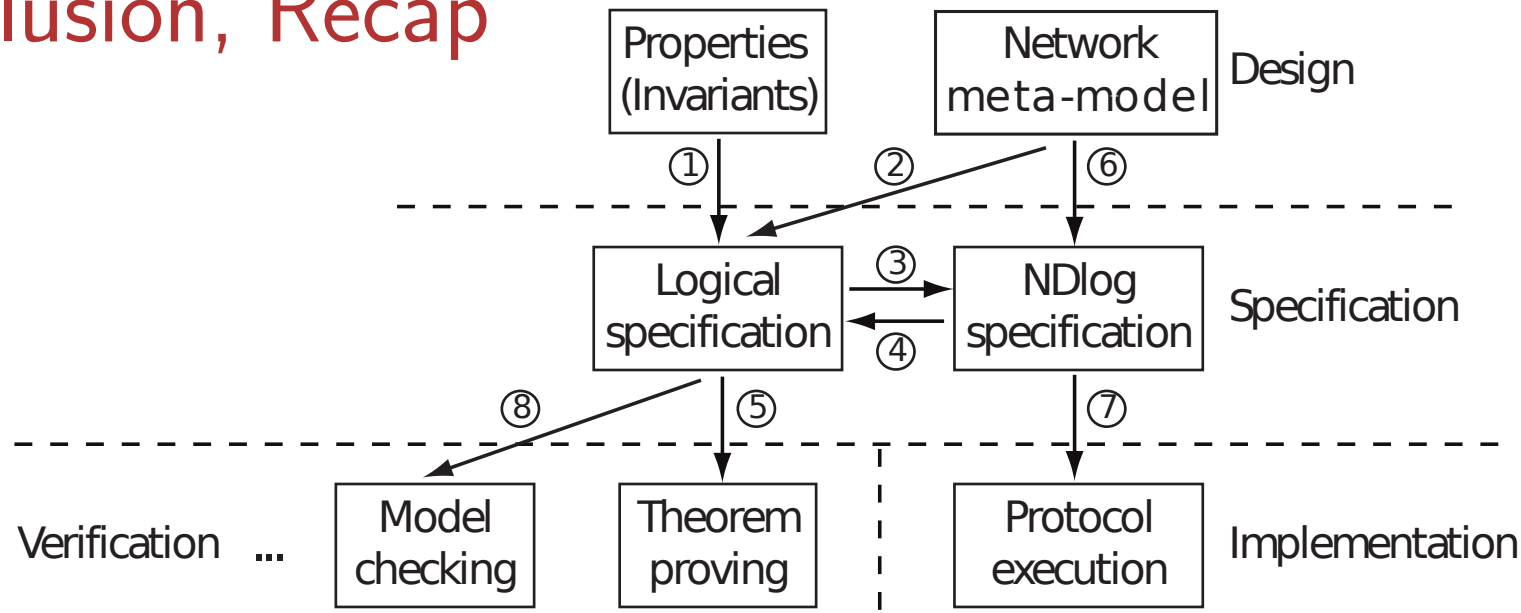
  IMPORTING lpA{{c := 3}}

  A2: THEORY =
        routeAlgebra
             {{sig = lpA.SIG, label = lpA.LABEL,
               labelApply(l: lpA.LABEL, s: lpA.SIG) = l + s, prohibitPath = 4,
               prefRel(s1, s2: int) = (s1 ≤ s2)}}

  B2: THEORY =
        routeAlgebra
             {{sig = addA.SIG, label = addA.LABEL,
               labelApply(l: addA.LABEL, s: addA.SIG) = mod(l + s, 16),
               prohibitPath = 17,
               prefRel(s1, s2: addA.SIG) = (s1 ≤ s2)}}

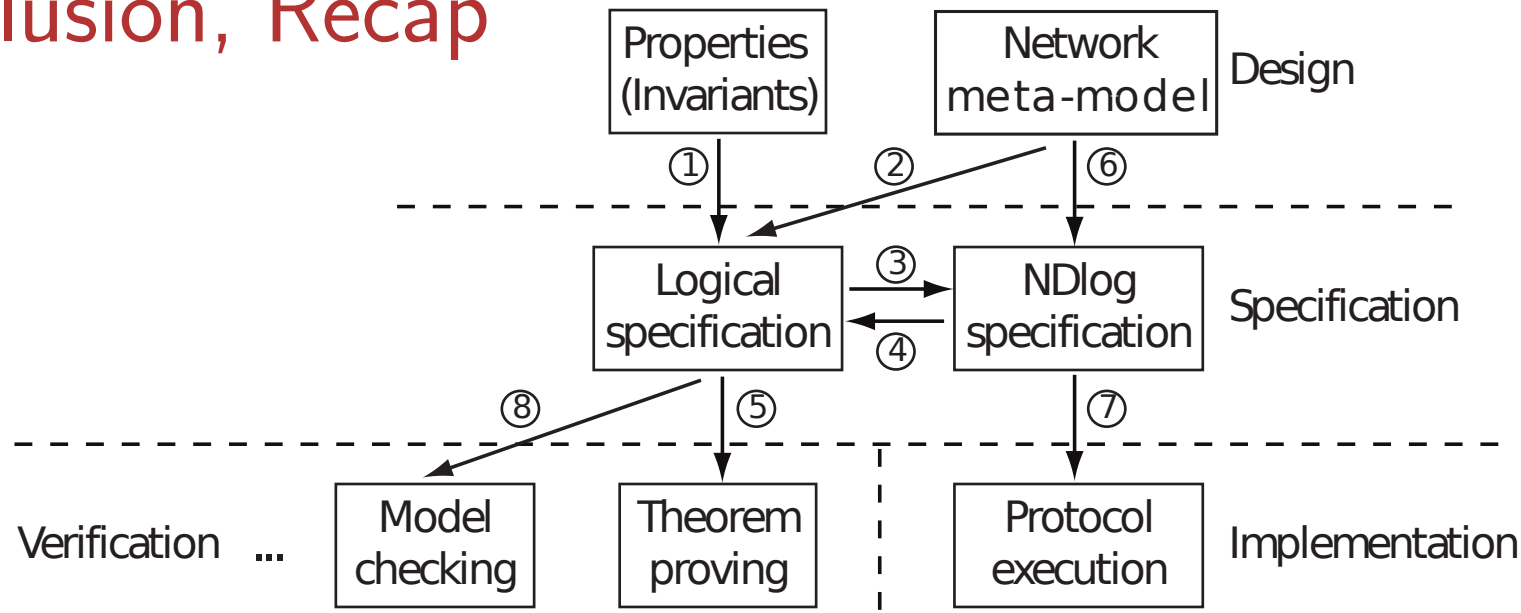

 END AlgebraInstance
```

Conclusion, Recap



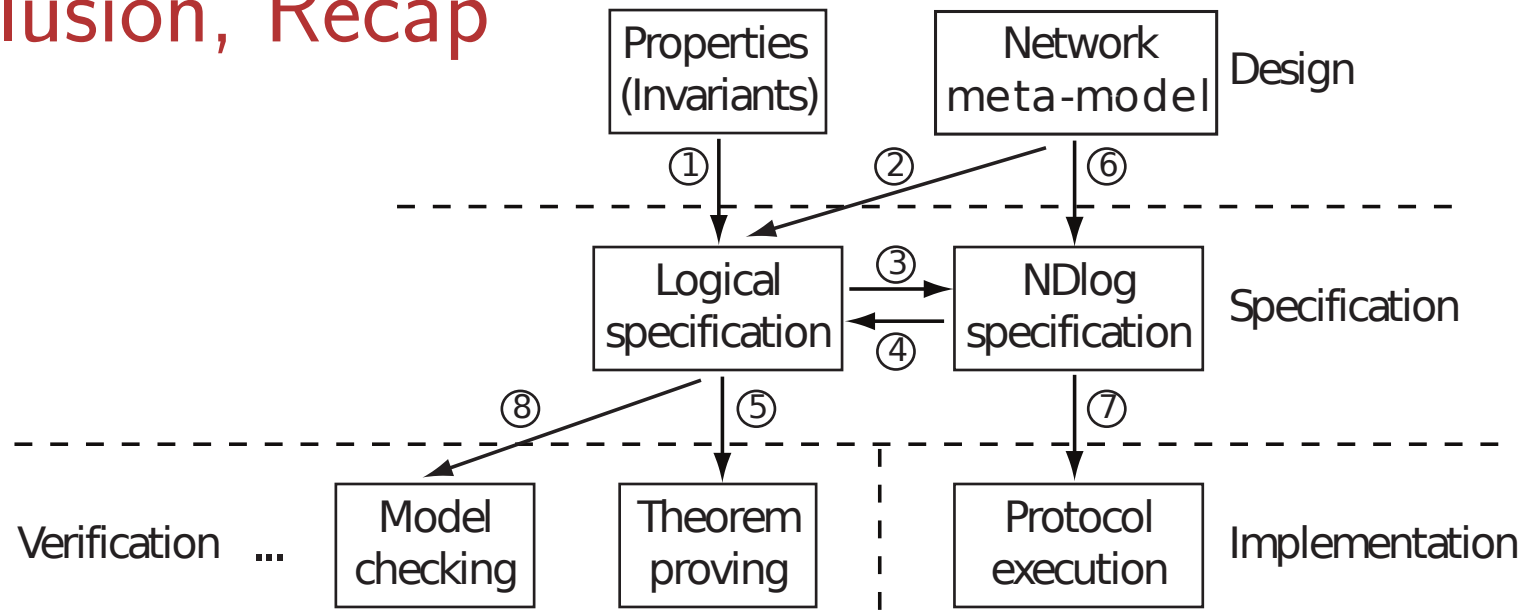
- Design: correctness-by-construction via meta-model

Conclusion, Recap



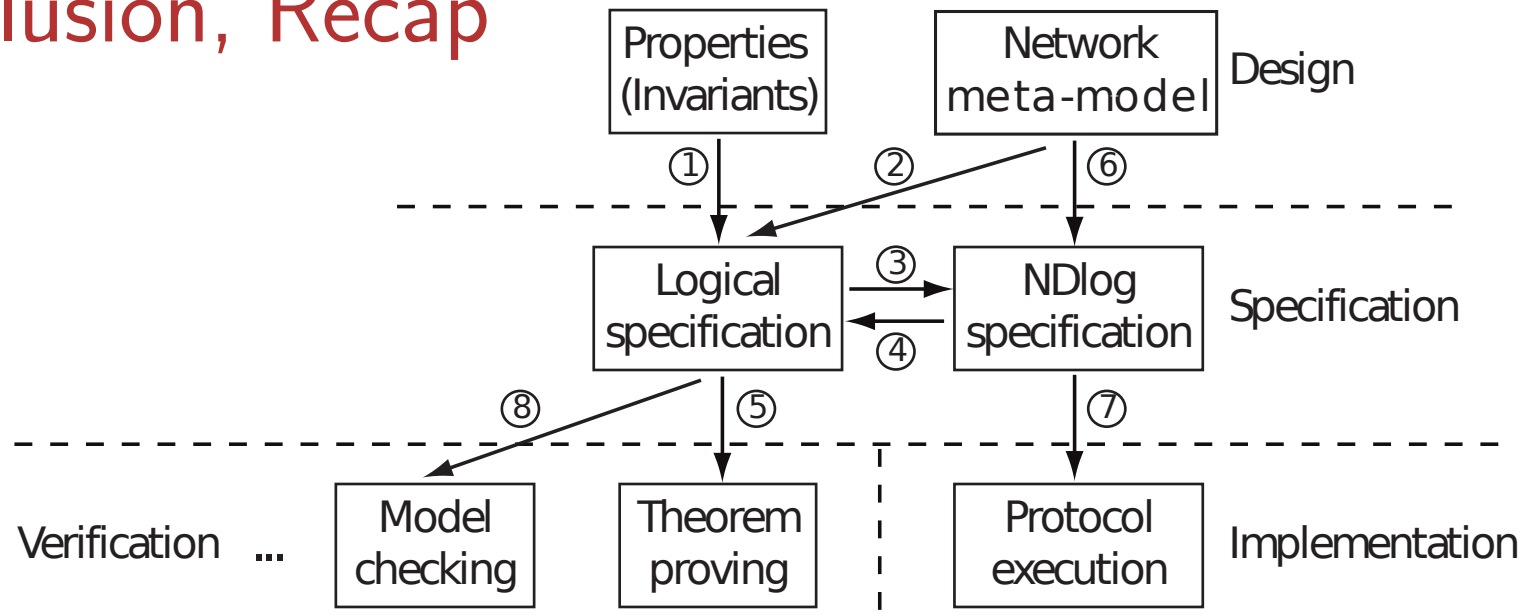
- Design: correctness-by-construction via meta-model
- Specification: two way property preserving translation
  - Formal system specification generated from NDlog program (arc 4)
  - Executable Declarative network synthesized from verified logical specification (arc 3)

Conclusion, Recap



- Design: correctness-by-construction via meta-model
- Specification: two way property preserving translation
  - Formal system specification generated from NDlog program (arc 4)
  - Executable Declarative network synthesized from verified logical specification (arc 3)
- Verification: proving network invariants of system specifications by interacting with theorem prover (arc 5)

Conclusion, Recap



- Design: correctness-by-construction via meta-model
- Specification: two way property preserving translation
  - Formal system specification generated from NDlog program (arc 4)
  - Executable Declarative network synthesized from verified logical specification (arc 3)
- Verification: proving network invariants of system specifications by interacting with theorem prover (arc 5)
- Implementation: distributed query processing (arc 7)

Open Issues
Destructive Updates, Linear Logic, Model Checking

- Non-monotonic updates in Networking and security
  - Link-failure, base tuples removed from routing table
  - Incremental maintenance of derived tuples
- Classical/Intuitive logic is monotonic
  finner control with linear logic?
  - Logical premises consumed (tuples removed) after use in proof/derivation
  - Linear logic as the semantic foundation for Declarative Networking with soft-state?

Open Issues, Cntd
Exploring Various Models, Verification Techniques

- Network Models and Implementation
  - Relaxed Algebraic models for wider range of protocols
    - Metarouting algebra are *monotonic*
    - Non-monotonic attributes provide useful semantic in real networking, e.g. `MED`.
  - Alternative component-based models: Click, Xorp
- Combining Verification Techniques
  - Theorem proving, declarative networking specific decision procedures/proof strategies/tactics
  - Model checking declarative networking, transitions updating routing tables are specified in linear logic
  - Automatically model checking finite small networking case, and using theorem proving to scale

Thank You!

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