Formally Verifiable Networking

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

[1]University of Pennsylvania

[2]BBN technologies

HotNets-VIII Oct 22-23, 2009

Motivation

- Challenges to today's Internet: increasing complexity and fragility in Internet routing
- Proliferation of new network protocols and architectures
- Growing interest in the formal verification of network protocol design and implementation

Motivation

- Challenges to today's Internet: increasing complexity and fragility in Internet routing
- Proliferation of new network protocols and architectures
- Growing interest in the formal verification of network protocol design and implementation
- Challenge: Ensuring that implementation matches design and specification
  - Verification decoupled from actual implementation
  - Actual implementation is not guaranteed to be error-free even when specification/design verified correct

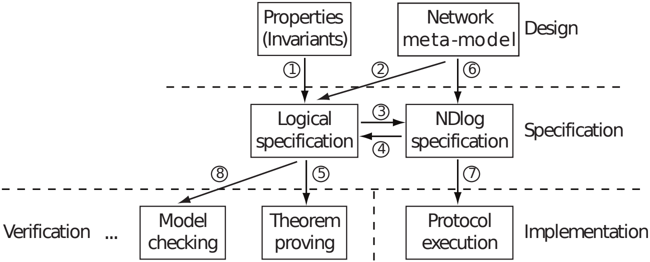
Recent Efforts in Practical Network Verification

- Runtime debugging
  - Pip [NSDI'06], DS3 [NSDI'08]
  - Additional runtime overhead, inconclusive
- Model checking
  - CMC [NSDI'04], MaceMC [NSDI'07]
  - Inconclusive, restricted to temporal property and small network
- Correct-by-construction
  - Metarouting [SIGCOMM'05]
  - Idealized model unlikely to be adapted to actual implementation

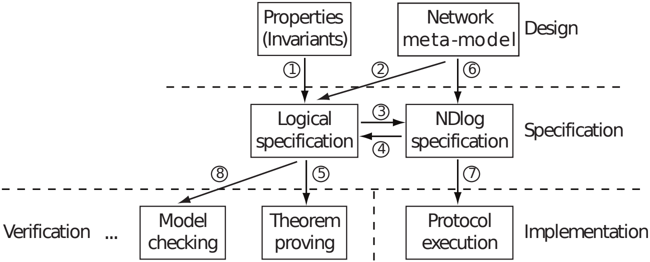
Recent Efforts in Practical Network Verification

- Runtime debugging
  - Pip [NSDI'06], DS3 [NSDI'08]
  - Additional runtime overhead, inconclusive
- Model checking
  - CMC [NSDI'04], MaceMC [NSDI'07]
  - Inconclusive, restricted to temporal property and small network
- Correct-by-construction
  - Metarouting [SIGCOMM'05]
  - Idealized model unlikely to be adapted to actual implementation
- We propose *Formally Verifiable Networking* (FVN)
  - *Unifies the design, specification, implementation, and verification of networking protocols*
  - http://netdb.cis.upenn.edu/fvn/

Formally Verifiable Networking (FVN)

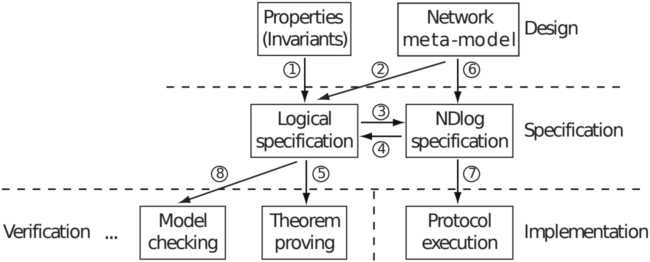


Formally Verifiable Networking (FVN)



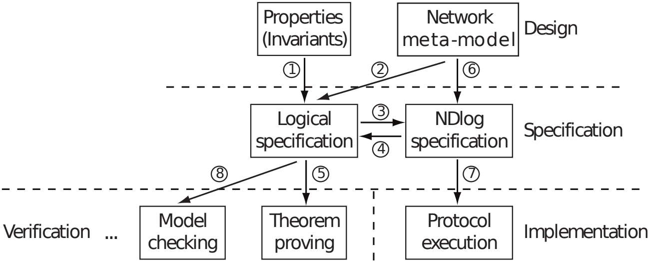
- Conceptually sound *meta-model* for program synthesis

Formally Verifiable Networking (FVN)



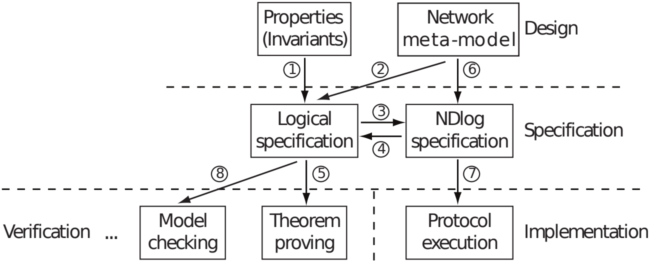
- Conceptually sound *meta-model* for program synthesis
- *Formal logical statements* specify the behavior and the properties of the protocol

Formally Verifiable Networking (FVN)



- Conceptually sound *meta-model* for program synthesis
- *Formal logical statements* specify the behavior and the properties of the protocol
  - Declarative networking [SIGCOMM'05] bridges logical specification and actual implementation

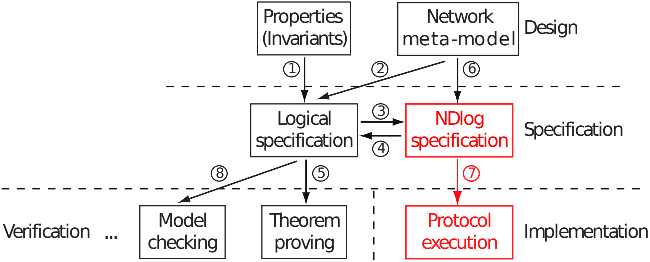
Formally Verifiable Networking (FVN)



- Conceptually sound *meta-model* for program synthesis
- *Formal logical statements* specify the behavior and the properties of the protocol
  - Declarative networking [SIGCOMM'05] bridges logical specification and actual implementation
- *Theorem proving* establishes correctness proof
  - System specification $\Longrightarrow$ property specification
  - Machine checked proof, proof automation support

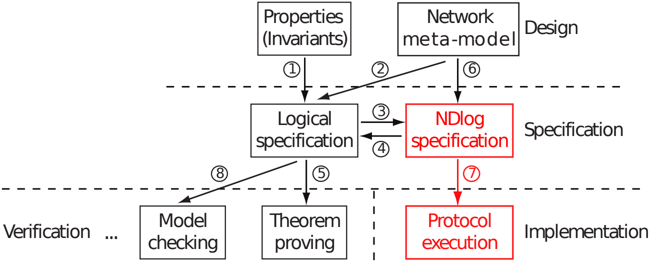
Declarative Networking

Bridges Logical Specification and System Implementation



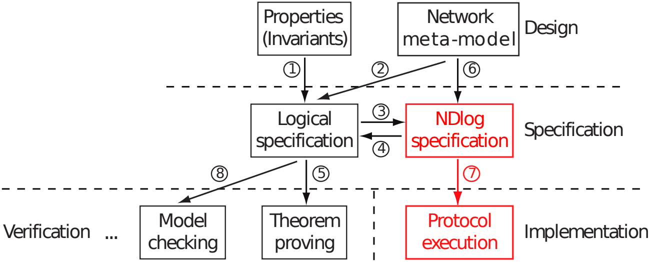
- Network protocols programmed in *Network Datalog* (NDlog), a distributed variant of Datalog

Declarative Networking
Bridges Logical Specification and System Implementation



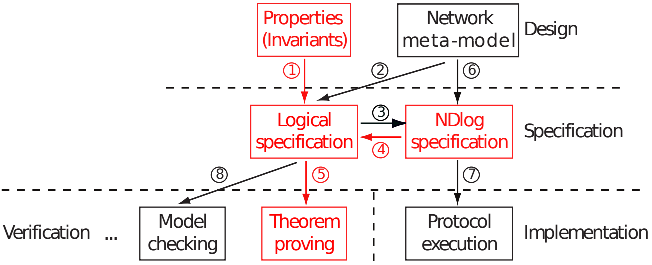
- Network protocols programmed in *Network Datalog* (NDlog), a distributed variant of Datalog
- NDlog is compiled to Click-like dataflows (arrow 7)

Declarative Networking
Bridges Logical Specification and System Implementation



- Network protocols programmed in *Network Datalog* (NDlog), a distributed variant of Datalog
- NDlog is compiled to Click-like dataflows (arrow 7)
- Distributed query processor executes the dataflows to implement the network protocols

Declarative Network Verification



- Given a NDlog program representation, FVN automatically generates formal system specifications recognizable by standard theorem provers (arrow 4)
- Verify network properties (arrow 5) by proving invariants (arrow 1) in theorem prover

Example Declarative Network Verification

Path Vector Protocol

```
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)
```

Example Declarative Network Verification
Path Vector Protocol

- ```
  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)
  ```
  ```

Example Declarative Network Verification
Path Vector Protocol

```
  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)
```

Example Declarative Network Verification
Path Vector Protocol

```
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)
```

- Semantics in first order logic
  - p1: $\forall(S, D, P, C).link(S, D, C) \wedge P = f_{init}(S, D) \implies path(S, D, P, C)$
  - p2: $\forall(S, D, P, C).\exists(C_1, C_2, Z, P_2).\ link(S, Z, C_1) \wedge bestPath(Z, D, P_2, C_2) \wedge C = C_1 + C_2 \wedge P = f_{concatPath}(Z, P_2) \implies path(S, D, P, C)$

Example Declarative Network Verification

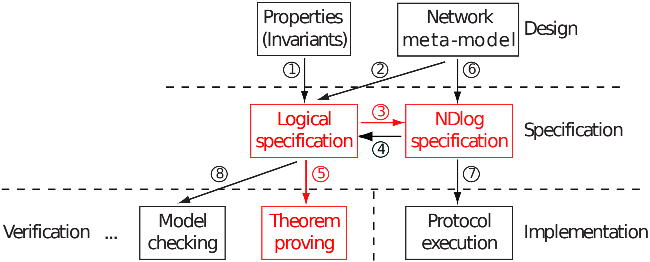
Path Vector Protocol

- Prove route optimality property in example theorem prover: *Prototype Verification System* (PVS, `http://pvs.csl.sri.com/`)
  - **Goal/Theorem**:
    ```
    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)
    ```
  - **Proof script**:
    ```
     ("" (skosimp*) (expand bestPath) (prop) (expand
    bestPathCost) (prop) (skosimp*) (inst -2 C2!1)
    (grind))
    ```

Declarative Network Verification

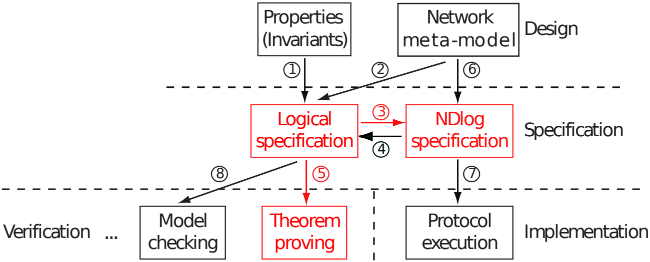
- More details on *Declarative Network Verification*
  Anduo Wang, Prithwish Basu, Boon Thau Loo, Oleg Sokolsky, 11th International Symposium on Practical Aspects of Declarative Languages [PADL'09]
- Representative properties proved for distances-vector protocol
  - Eventual convergence in stable network
  - Divergence (*count-to-infinity*) 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

Verified Code Generation



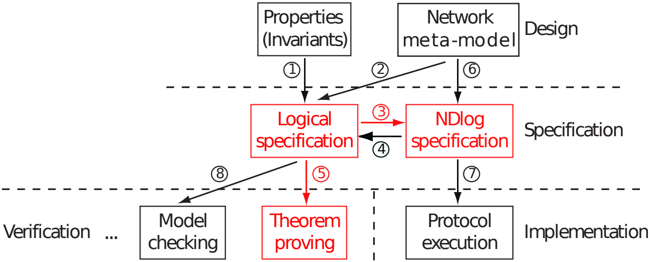
- Start from *component-based* formal system specifications

Verified Code Generation



- Start from *component-based* formal system specifications
- Verifying network properties (arrow 5) by proving invariants within theorem prover

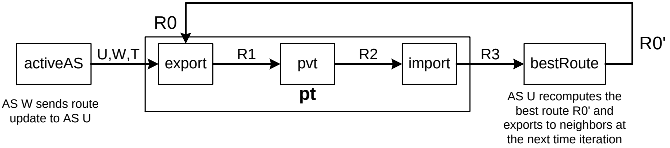
Verified Code Generation



- Start from *component-based* formal system specifications
- Verifying network properties (arrow 5) by proving invariants within theorem prover
- FVN automatically generates equivalent NDlog program (arrow 3) from verified component specification

Component Based Verification of BGP System

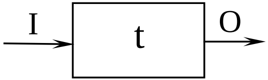
- Component model for BGP system based on route-transformation presented in *An analysis of BGP convergence properties.*, Timothy G.Griffin and Gordon Wilfong [SIGCOMM'99]



- Specification of BGP components in PVS
  - `bgp(U,W,R0,R3,T): INDUCTIVE bool = activeAS(U,W,T) AND pt(U,W,R0,R3,T) AND bestRoute(W,T,R0)`
  - `pt(U,W,R0,R3,T): INDUCTIVE bool = EXISTS (R1,R2): export(U,W,R0,R1,T) AND pvt(U,W,R1,R2,T) AND import(U,W,R2,R3,T)`

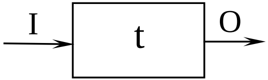
Generating Equivalent NDlog Implementation
Atomic Component Defined by Internal Constraints

- Given verified atomic component t in PVS
  `t(I,O):INDUCTIVE bool = CT(I,O)`



Generating Equivalent NDlog Implementation
Atomic Component Defined by Internal Constraints

- Given verified atomic component `t` in PVS
  `t(I,O):INDUCTIVE bool = CT(I,O)`



- The equivalent NDlog rule
  `t t_out(O) :- t_in(I), CT(I,O)`
  - Deriving output `O` as head
  - Body defined by input `I` and internal constraints `CT(I,O)`

Generating Equivalent NDlog implementation
Compositional Component Implied by Sub-Components

- Given verified compositional component `tc` defined in terms of sub-components `t1,t2,t3`

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

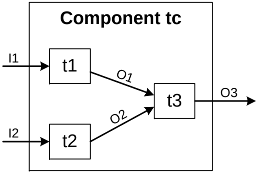
```
t1(I,O): INDUCTIVE bool = C1(I,O)
t2(I,O): INDUCTIVE bool = C2(I,O)
t3(I,O',O): INDUCTIVE bool =
C3(I,I',O)
```

Generating Equivalent NDlog implementation
Compositional Component Implied by Sub-Components

- Given verified compositional component tc defined in terms of sub-components t1,t2,t3

```
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)
```

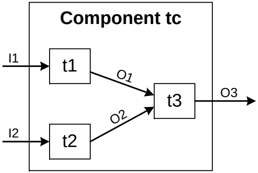


Generating Equivalent NDlog implementation
Compositional Component Implied by Sub-Components

- Given verified compositional component `tc` defined in terms of sub-components `t1,t2,t3`

```
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)
```



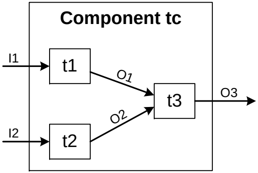
- *Outputs from t1, t2 are inputs for t3*

Generating Equivalent NDlog implementation
Compositional Component Implied by Sub-Components

- Given verified compositional component tc defined in terms of sub-components t1,t2,t3

```
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)
```



- *Outputs from t1, t2 are inputs for t3*
- tc implicitly implied by equivalent NDlog rules for sub-components t1,t2,t3

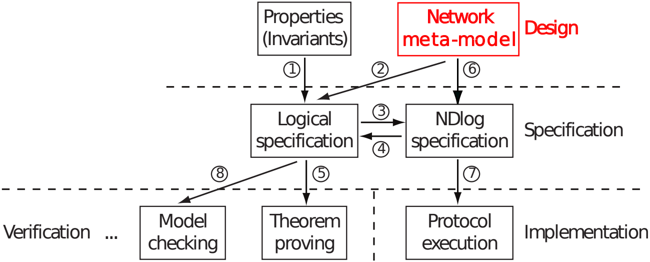
```
t1 t1_out(O1) :- t1_in(I1), C1(I1,O1).
t2 t2_out(O2) :- t2_in(I2), C2(I2,O2).
t3 t3_out(O3) :- t1_out(O1), t2_out(O2), C3(O1,O2,O3).
```

Verified Code Generation of BGP System

- Main take-away: Automatically generate executable NDlog program from BGP component specification
- More details on component-based BGP verification and verified code generation
  - *A Theorem Proving Approach Towards Declarative Networking*., Anduo Wang, Boon Thau Loo, Changbin Liu, Oleg Sokolsky, Prithwish Basu., Theorem Proving in High Order Logics [TPHOLs'09], Emerging Trends, 2009
  - *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.

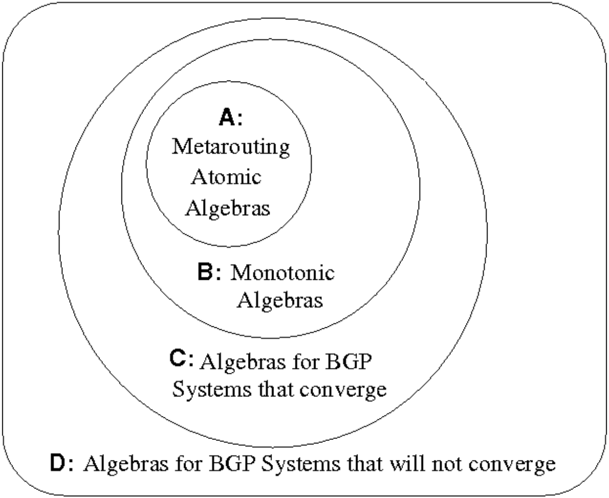
Meta-Theoretic Model

Correctness-By-Construction via Metarouting

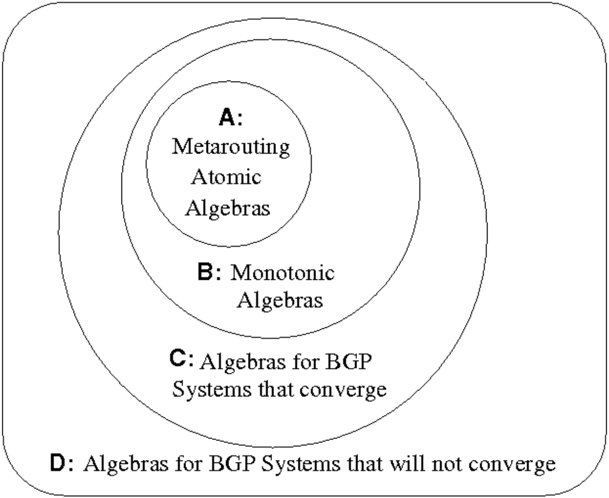


- Meta-model provides sound conceptual model for NDlog program synthesis
- Metarouting, example meta-model
  - Algebraic framework modeling BGP systems with convergence guarantee
  - Impose strong assumptions (e.g. `MED` violates *monotonicity*) unlikely to be adopted in practice

Overview of Metarouting



Overview of Metarouting



- Our goal: Ensure correctness of protocol design by verifying it belongs to **C**, and do synthesis accordingly

Shortest Path Routing in FVN

Source Theory: Abstract Algebra `routeAlgebra`
Interpreted Theory: Base Algebra `addA`

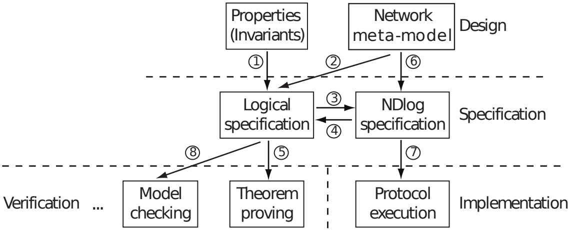
```
addA: THEORY
 BEGIN
  n: posnat
  m: posnat
  redundant: posnat
  N_M: AXIOM n < m
  LABEL: TYPE = upto(n)
  SIG: TYPE = upto(m+1)
  PREF(s1, s2: SIG): bool = (s1 ≤ s2)
  APPLY(l: LABEL, s: SIG): SIG =
      IF (l+s < m+1)
        THEN (l+s)
      ELSE (m+1)
      ENDIF
  IMPORTING routeAlgebra
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

Using Metarouting Algebras in FVN

- More details on the construction of BGP systems
  - *Formalizing Metarouting in PVS.*,
    Anduo Wang and Boon Thau Loo, Automated Formal methods [AFM'09], 2009
- Main take-aways
  - PVS manages proof checking manually required in metarouting
  - Enables user to focus on high-level composition
  - Allows reasoning about well-behaved protocols not captured in metarouting theory

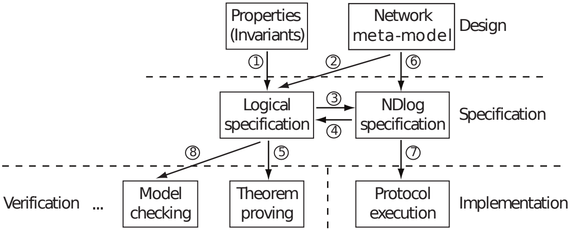
Conclusion, Recap

Formally Verifiable Networking: Unify the Design, Specification, Implementation, and Verification of Networking Protocols



Conclusion, Recap

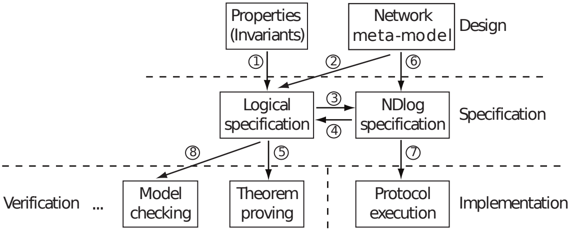
Formally Verifiable Networking: Unify the Design, Specification, Implementation, and Verification of Networking Protocols



- Conceptually sound *meta-model* for program synthesis

Conclusion, Recap

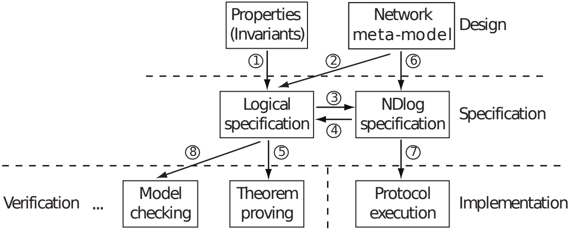
Formally Verifiable Networking: Unify the Design, Specification, Implementation, and Verification of Networking Protocols



- Conceptually sound *meta-model* for program synthesis
- *Formal logical statements* specify the behavior and the properties of the protocol

Conclusion, Recap

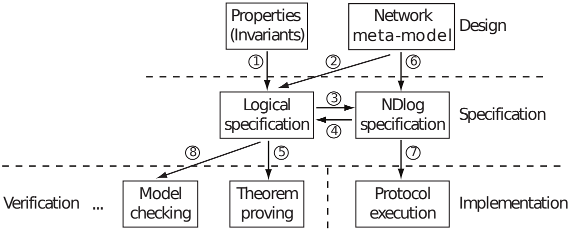
Formally Verifiable Networking: Unify the Design, Specification, Implementation, and Verification of Networking Protocols



- Conceptually sound *meta-model* for program synthesis
- *Formal logical statements* specify the behavior and the properties of the protocol
  - Declarative networking bridges logical specification and actual implementation

Conclusion, Recap

Formally Verifiable Networking: Unify the Design, Specification, Implementation, and Verification of Networking Protocols



- Conceptually sound *meta-model* for program synthesis
- *Formal logical statements* specify the behavior and the properties of the protocol
  - Declarative networking bridges logical specification and actual implementation
- *Theorem proving* establishes correctness proof
  - System specification $\Longrightarrow$ property specification
  - Machine checked proof, proof automation support

Ongoing Work

- Synthesis NDlog program from metarouting specification
- Relaxed network models
  - Non-monotonic algebraic models for wider range of well-behaved protocols
- Automate proof search processes
  - Customize PVS with developing declarative networking specific proof strategies/tactics
  - Model checking declarative networking *protocol state transitions* updating routing tables are specified in *linear logic*
    - *Linear logic* is a novel state-aware logic
- Combining verification techniques
  - Model checking small network instance and use theorem proving to generalize

Thank You!

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