CS 598D Formal Methods in Networking
Princeton University

Lecture 17-18:

Ehab Al-Shaer

Cyber Defense and Network Assurability (CyberDNA) Center
School of Computing & Informatics
University of North Carolina, Charlotte, NC

April 5,9, 2010

Acknowledgment: ***Will Marrero, Hazem Hamed, Adel El-Atawy, and Taghrid Samak***

Lecture 18 (ConfigChecker):

Network Configuration in a Box: BDD-based Model Checker Approach for

Applications:

- Reachability Analysis
- Security Verification
- Routing Protocols Debugging
- QoS Policy Evaluation and Debugging
- Quantifying System Reliability/Resiliency

Limitations and Objectives

- Global & Comprehensive Abstraction:
  - end-to-end verification of network configuration reachability and security requirements,
  - Including all network devices such as routers (unicast and multicast), firewalls, NAT, and IPSec.
- Extensibility
  - Canonical encoding of network access control configuration representation including forwarding/routing, translation, transformation and filtering.
- Scalability
  - Implementing a scalable model checker tool that can handle thousands of devices and millions of configuration rules
- Verifiability
  - Using property-based verification to establish soundness and completeness of network reachability of security requirements

Why Symbolic?

Symbolic Model Checking

Any model checking method that represents state sets *symbolically* as opposed to *explicitly* enumerating states, usually using OBDDs.

Motivation for model checking

- Hardware and software become increasing complicated today. Verification of correctness of them is critical
- Deductive verification is widely used by it is time consuming and can only done by experts
- Model checking can be used to verify finite state concurrent systems. It can be performed *automatically*.
- CTL and OBDD based model checking is very efficient in many cases and it can cope with the state explosion problem.

Modeling systems: *Kripke structure*

Let *AP* be a set of atomic propositions, A *Kripke structure M* over *AP* is a four tuple $M=(S, S_0, R, L)$ where

- S is a finite set of states.
- $S_0 \subseteq S$ is the set of initial states
- $R \subseteq S \times S$ is a transition relation that must be total, that is for every state s in S there is a s' such that $R(s,s')$
- $L : S \rightarrow 2^{AP}$ is a function that labels each state with the set of atomic propositions true in the state

Temporal operators and CTL

- **Temporal logic:** describes sequences of transitions between state but time is not mentioned explicitly, instead, a formula will specify that "eventually" some designated state is reached or error state is "never" entered.
- **Computation Tree Logic (CTL)** is a branching-time logic, meaning that its model of time is a tree-like structure in which the future is not determined; there are different paths in the future, any one of which might be an actual path that is realized.
- CTL is a subset of CTL*
- The operators in CTL* includes:
  - Quantifiers over paths
    - **A (φ)** **A**ll: φ has to hold on all paths starting from the current state.
    - **E (φ)** **E**xist: there exists at least one path starting from the current state where φ holds
  - Temporal specific quantifiers
    - **X φ** Ne**x**t: φ has to hold at the next state.
    - **G φ** **G**lobally: φ has to hold on the entire subsequent path.
    - **F φ** **F**inally: φ eventually has to hold (somewhere on the subsequent path)
    - **Φ U ψ** **U**ntil: φ has to hold until ψ hold. ψ eventually will be verified
    - **Φ R ψ** **R**elease: φ has to hold before ψ ceases to hold.

From [MC]

Model Checking Goal

Given:

- Kripke-Structure K
- CTL Formula φ

Goal:

- Identify the ***set of states*** of K where φ is true.

$$p$$

$$\neg\varphi \quad \varphi \wedge \psi \quad \ldots$$

$$\mathbf{AX}\ \varphi \quad \mathbf{EX}\ \varphi$$

$$\mathbf{AG}\ \varphi \quad \mathbf{EG}\ \varphi$$

$$\mathbf{AF}\ \mathrm{p} \quad \mathbf{EF}\ \varphi$$

$$\mathbf{A}(\varphi\ \mathbf{U}\ \psi)$$

$$\mathbf{E}(\varphi\ \mathbf{U}\ \psi)$$

Calculating State Sets

- State sets of a Kripke-structure can be represented as an OBDD!

$\neg \quad \wedge \quad \vee \quad \rightarrow \quad \leftrightarrow$

- Propositional connectives can be evaluated using OBDD algorithms.
- What about temporal connectives?

Representing Transitions

Primed variables for next state

t_1 (1) $\longrightarrow$ (4) b, t_1

Current State					Next State				
	b	h	t_1	t_2		b'	h'	t'_1	t'_2
0	0	0	0	0	3	1	0	0	0
1	0	0	1	0	0	0	0	0	0
1	0	0	1	0	4	1	0	1	0
2	0	0	1	1	1	0	0	1	0
2	0	0	1	1	5	1	0	1	1
3	1	0	0	0	6	1	1	0	0
4	1	0	1	0	3	1	0	0	0
5	1	0	1	1	4	1	0	1	0
...					...				

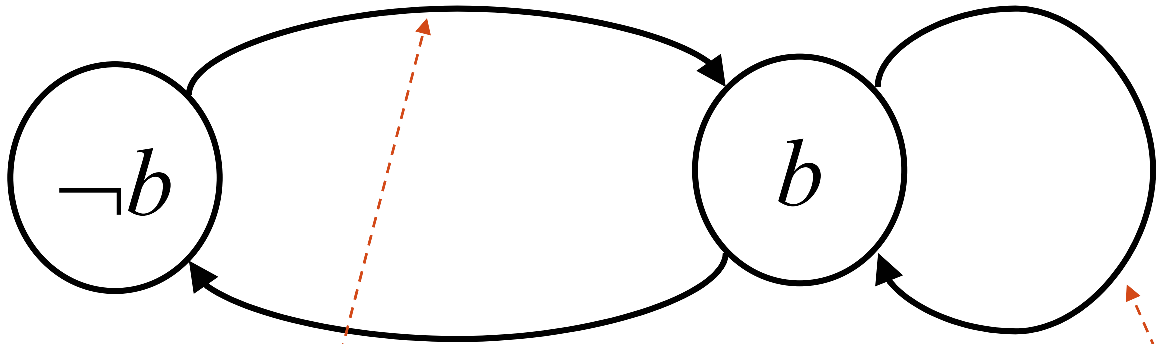
t_1 (1) $\longrightarrow$ (4) b, t_1

$$\neg b \wedge \neg h \wedge t_1 \wedge \neg t_2$$
$$\wedge$$
$$b' \wedge \neg h' \wedge t'_1 \wedge \neg t'_2$$

b (3) $\longrightarrow$ (6) b, h

$$b \wedge \neg h \wedge \neg t_1 \wedge \neg t_2$$
$$\wedge$$
$$b' \wedge h' \wedge \neg t'_1 \wedge \neg t'_2$$

A Simple Example



$$
\begin{aligned}
T &= (\neg b \wedge b') \vee (b \wedge \neg b') \vee (b \wedge b') \\
&= (\neg b \wedge b') \vee b
\end{aligned}
$$

Calculating $\mathbf{EX}$

$$\mathbf{EX}\,\neg b \;=\; \exists b'\,(T \wedge \neg b')$$

Basic CTL operations

- **AX** and **EX**
- **AF** and **EF**
- **AG** and **EG**
- **AU** and **EU**
- **AR** and **ER**

Each of the ten operators can be expressed in terms of three operators **EX**, **EG**, and **EU**(□ means logical and)

- $\mathbf{AX}\, f \equiv \neg\, \mathbf{EX}(\neg f)$
- $\mathbf{EF}\, f \equiv \mathbf{E}\, [\mathit{True}\ \mathbf{U}\, f]$
- $\mathbf{AG}\, f \equiv \neg\, \mathbf{EF}(\neg f)$
- $\mathbf{AF}\, f \equiv \neg\, \mathbf{EG}(\neg f)$
- $\mathbf{A}[f\, \mathbf{U}\, g] \equiv \neg\, \mathbf{E}[\neg g\ \mathbf{U}\ (\neg f \square \neg g)] \square \neg\, \mathbf{EG} \neg g$
- $\mathbf{A}\, [f\, \mathbf{R}\, g] \equiv \neg\, \mathbf{E}[\neg f\, \mathbf{U} \neg g]$
- $\mathbf{E}\, [f\, \mathbf{R}\, g] \equiv \neg\, \mathbf{A}[\neg \mathrm{f}\, \mathbf{U} \neg g]$

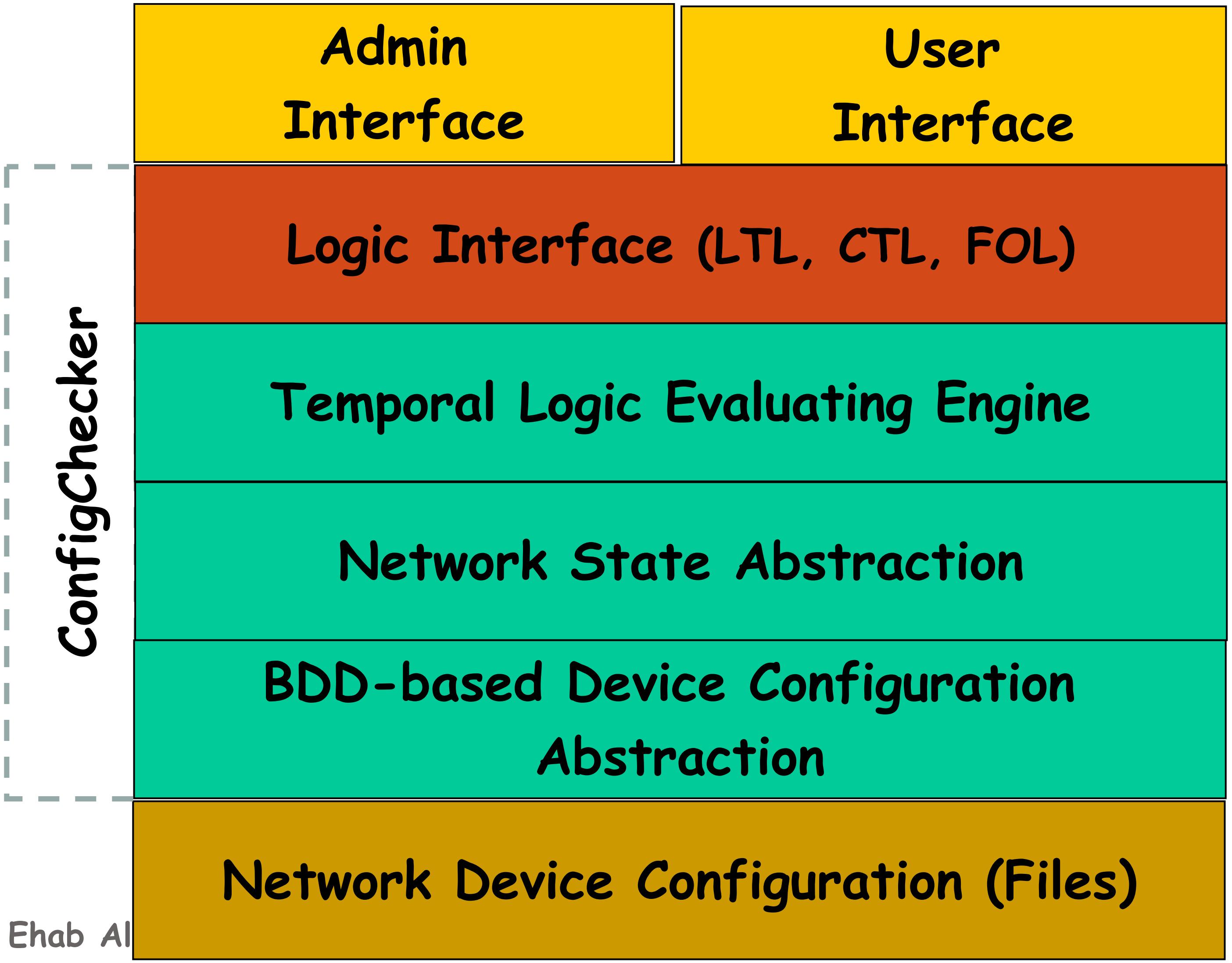
Some examples

- **EF**(*Start* and $\neg$ *Ready*) : It is possible to get a state where Start holds but Ready does not hold
- **AG**(*Req* $\rightarrow$ **AF** *Ack*): If a request occurs, then it will be eventually acknowledged
- **AG**(**AF** *DeviceEnabled*): The proposition DeviceEnabled holds infinitely often on every computation path
- **AG**(**EF** *Restart*): From any state it is possible to get to the Restart state

Examples in Networks

- A state satisfies ***EX***(*loc*=10.10.10.10) if there is a next state in which the packet in location with address 10.10.10.10
- A state satisfies ***AX***(*loc*=10.10.10.10) if in all next states, the packet is at location 10.10.10.10
- A state satisfies ***EF***(*loc*=10.10.10.10) if there is a path from this state along which eventually the location of the packet is 10.10.10.10
- A state satisfies ***AF***(*loc*=10.10.10.10) if along all paths from this state, eventually the packet will be in 10.10.10.10

ConfigChecker



Ehab Al

Formalization – The Basic Model

- The network is modeled as a state machine
  - each state determined by the packet header information and packet location on the network
    - States = Locations X Packets
  - The ***characterization function*** to encode the state of the network in the basic model (abstracting payload)

$$\sigma : \mathrm{IP_s} \times \mathrm{port_s} \times \mathrm{IP_d} \times \mathrm{port_d} \times \mathrm{loc} \rightarrow \{\mathbf{true, false}\}$$

$\mathrm{IP_s}$	the 32-bit source IP address
$\mathrm{port_s}$	the 16-bit source port number
$\mathrm{IP_d}$	the 32-bit destination IP address
$\mathrm{port_d}$	the 16-bit destination port number
loc	the 32-bit IP address of the device currently processing the packet

Formalization – The Basic Model

- Network devices are modeled based on the **packet *matching semantic*** and ***packet transformation***
  - Each rule consists of a condition (Ci) and an action (a): Ci➔a
  - Policy are set of rules matched sequentially with single- or multi-trigger actions
  - Firewall (single trigger) policy encoding using BDD

$$P_a = \bigvee_{i \in index(a)} (\neg C_1 \wedge \neg C_2 \ldots \neg C_{i-1} \wedge C_i)$$

$$= \bigvee_{i \in index(a)} \bigwedge_{j=1}^{i-1} \neg C_j \wedge C_i$$

- **Transformation:**
  - if a pkt *state* matches the rule *condition*, the Action can change the packet <u>location</u> and possibly the <u>headers</u> ➔ means change over the bits of the *state*
- Transition relation is ***characterization function*** as follows:
  - **t**: (Curr_pkt x Curr_loc)x (New_pkt x New_loc) → {true, false}
  - Device Model ϕ = ***loc*** $\wedge$ ***Match_Condition*** $\wedge$ **t** → **{true, false}**

Formalization – The Basic Model

- Global Transitions relation of the entire network:

$$T = \bigvee_{i \in devices} \Phi_{device_i}$$

- Variables
  - Locations is every place that can describe packet position: firewall, router, IPSec device, or application layer service, etc.
  - We allow Location to be different than IPsrc for spoofing
  - There are two versions of each variable: current and new state.
- Each property and field describing the state (i.e., location IP; packet properties: src/dst IP; port, proto, transformation, etc) is represented by bits, according to its size.
- These variables are used via a symbolic representation using Ordered Binary Decision Diagrams.
- Model Checking and CTL are used to answer the queries posed by the administrator.

Formalization – The

FW-IP=1, next-hop-IP=3
IPsrc=2, IPdest=* ➔ allow
IPsrc=*, IPdest=3, Pdest=1 ➔ allow

- Firewall Modeling (Examp

$(s_1 \wedge \overline{s_0}) \vee (d_1 \wedge d_0 \wedge d_p) \wedge$
$\bigwedge_{i\in\{0,1,p\}}(s'_i \Leftrightarrow s_i) \wedge \bigwedge_{i\in\{0,1,p\}}(d'$
$\overline{l_1} \wedge l_0$

Router-IP=2
IPdest=0 ➔ nexthop=0
IPdest=1 ➔ nexthop=0
(default-gateway) ➔ nexthop = 3

- Router Modeling (Example)

$(\overline{d_1} \wedge \overline{l'_1} \wedge \overline{l'_0}) \vee (d_1 \wedge l'_1 \wedge l'_0) \wedge$
$\bigwedge_{i\in\{0,1,p\}}(s'_i \Leftrightarrow s_i) \wedge \bigwedge_{i\in\{0,1,p\}}($

IP(NAT)= 2 connected to IP= 1
IPsrc=3/sport=1, IPdes=1➔
IPsrc=2/sport=0, IPdes=1

- NAT Modeling (Example)

$[s_1 \wedge s_0 \wedge s_p \wedge \overline{l'_1} \wedge l'_0 \wedge s'_1 \wedge \overline{s'_0} \wedge \overline{s'_p} \wedge \bigwedge_{i\in\{0,1,p\}}(d'_i \Leftrightarrow d_i)] \vee$ **outgoing**

$[d_1 \wedge \overline{d_0} \wedge \overline{d_p} \wedge l'_1 \wedge l'_0 \wedge d'_1 \wedge d'_0 \wedge d'_p \wedge \bigwedge_{i\in\{0,1,p\}}(s'_i \Leftrightarrow s_i)] \wedge l_1 \wedge \overline{l_0}$ **incoming**

Formalization – The Basic Model

- Firewall Modeling (Example)

$$(s_1 \wedge \overline{s_0}) \vee (d_1 \wedge d_0 \wedge d_p) \ \bigwedge$$
$$\bigwedge_{i \in \{0,1,p\}} (s'_i \Leftrightarrow s_i) \wedge \bigwedge_{i \in \{0,1,p\}} (d'_i \Leftrightarrow d_i) \wedge l'_1 \wedge l'_0 \ \bigwedge$$
$$\overline{l_1} \wedge l_0$$

- Router Modeling (Example)

$$(\overline{d}_1 \wedge \overline{l'_1} \wedge \overline{l'_0}) \vee (d_1 \wedge l'_1 \wedge l'_0) \quad \bigwedge$$
$$\bigwedge_{i \in \{0,1,p\}} (s'_i \Leftrightarrow s_i) \wedge \bigwedge_{i \in \{0,1,p\}} (d'_i \Leftrightarrow d_i) \ \bigwedge \ l_1 \wedge \overline{l_0}$$

- NAT Modeling (Example)

$$[s_1 \wedge s_0 \wedge s_p \wedge \overline{l'_1} \wedge l'_0 \wedge s'_1 \wedge \overline{s'_0} \wedge \overline{s'_p} \wedge \bigwedge_{i \in \{0,1,p\}} (d'_i \Leftrightarrow d_i)] \bigvee$$

outgoing

$$[d_1 \wedge \overline{d_0} \wedge \overline{d_p} \wedge l'_1 \wedge l'_0 \wedge d'_1 \wedge d'_0 \wedge d'_p \wedge \bigwedge_{i \in \{0,1,p\}} (s'_i \Leftrightarrow s_i)] \bigwedge \ l_1 \wedge \overline{l_0}$$

incoming

Formalization – The Extended Model

- IPSec encapsulation requires new headers and saving the old headers ➔ copier, stack, valid bit
- IPSec Modeling
  - Example: IPsrc=0, IPdest=3 ➔ enc_tunnel

(from Gateway of IP=1 to Gateway of IP=2)

Current location

➡ $l_0 \wedge \overline{l_1}$

Matching Condition

➡ $\wedge \overline{s_0} \wedge \overline{s_1} \wedge d_0 \wedge d_1$

Copying headers

➡ $\wedge \overline{v} \wedge v'$

➡ $\wedge \bigwedge_{i \in \{0,1,p\}} (\widehat{s'_i} \Leftrightarrow s_i) \wedge \bigwedge_{i \in \{0,1,p\}} (\widehat{d'_i} \Leftrightarrow d_i)$

➡ $\wedge \overline{s'_1} \wedge s'_0 \wedge s'_p \leftrightarrow s_p \wedge \overline{d'_0} \wedge d'_1 \wedge d'_p \leftrightarrow d_p$

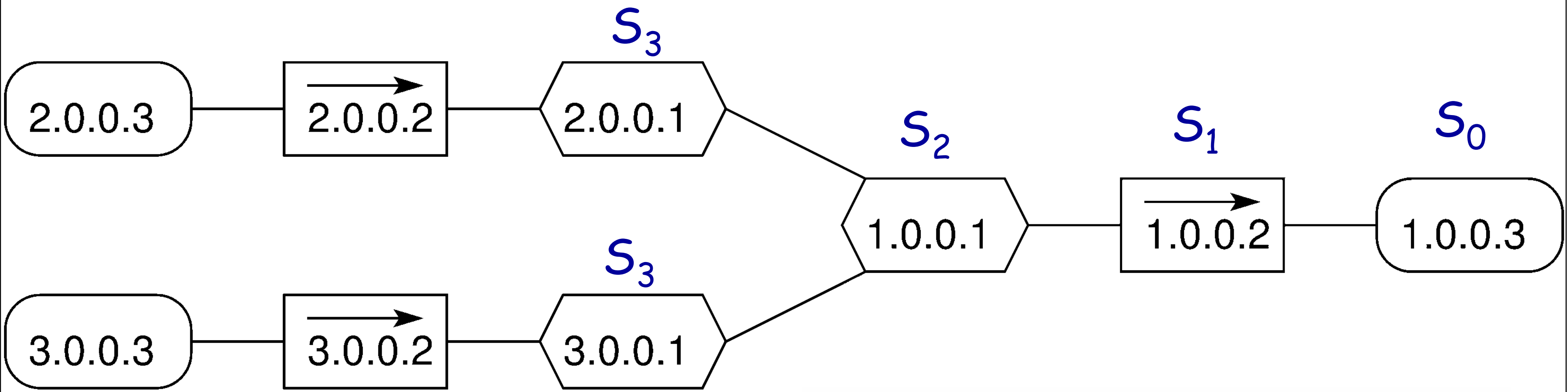
$\wedge \overline{l'_0} \wedge l'_1$

New headers

New location

Example

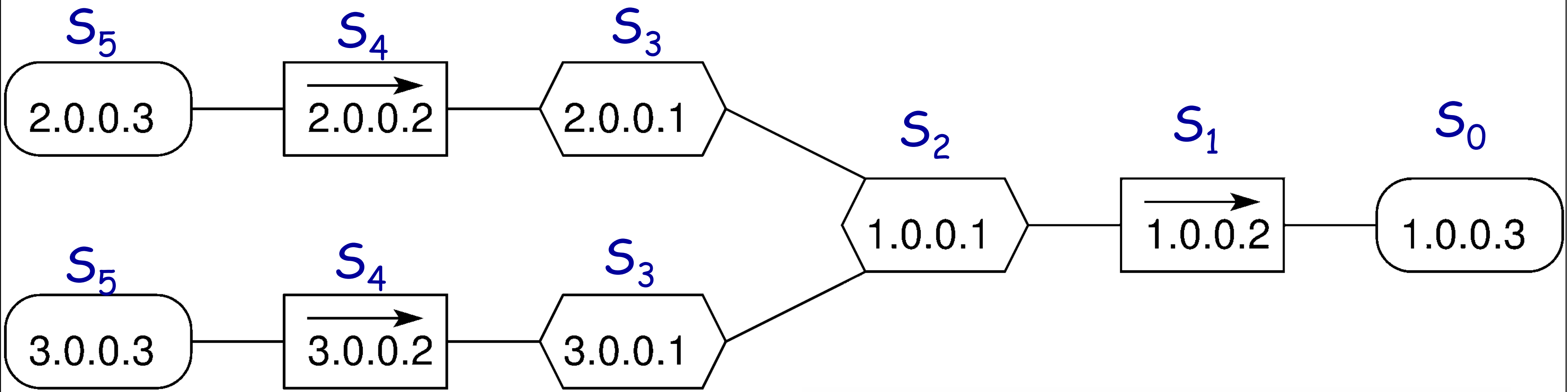
EF(loc=1.0.0.3)



loc	src	dst	loc'	src'	dst'
2.0.0.3	*	*	2.0.0.2	src	dst
2.0.0.2	*	*	2.0.0.1	src	dst
2.0.0.1	*	1.*.*.*	1.0.0.1	src	dst
2.0.0.1	*	3.*.*.*	1.0.0.1	src	dst
3.0.0.3	*	*	3.0.0.2	src	dst
3.0.0.2	*	*	3.0.0.1	src	dst
3.0.0.1	*	1.*.*.*	1.0.0.1	src	dst
3.0.0.1	*	2.*.*.*	1.0.0.1	src	dst
1.0.0.1	*	1.*.*.*	1.0.0.2	src	dst
1.0.0.1	*	2.*.*.*	2.0.0.1	src	dst
1.0.0.1	*	3.*.*.*	3.0.0.1	src	dst
1.0.0.2	2.*.*.*	1.0.0.3	1.0.0.3	src	dst

Example

EF(loc=1.0.0.3)



loc	src	dst	loc'	src'	dst'
2.0.0.3	*	*	2.0.0.2	src	dst
2.0.0.2	*	*	2.0.0.1	src	dst
2.0.0.1	*	1.*.*.*	1.0.0.1	src	dst
2.0.0.1	*	3.*.*.*	1.0.0.1	src	dst
3.0.0.3	*	*	3.0.0.2	src	dst
3.0.0.2	*	*	3.0.0.1	src	dst
3.0.0.1	*	1.*.*.*	1.0.0.1	src	dst
3.0.0.1	*	2.*.*.*	1.0.0.1	src	dst
1.0.0.1	*	1.*.*.*	1.0.0.2	src	dst
1.0.0.1	*	2.*.*.*	2.0.0.1	src	dst
1.0.0.1	*	3.*.*.*	3.0.0.1	src	dst
1.0.0.2	2.*.*.*	1.0.0.3	1.0.0.3	src	dst

S_1 =SAT(T(current_state and Next_state=S_0))
S_2 =SAT(T(current_state and Next_state=S_1))
S_3 =SAT(T(current_state and Next_state=S_2))
= (Loc=**2.0.0.1** ^ src=2.*.*.*. ^ dst=1.0.0.3) v
(Loc=**3.0.0.1** ^ src=2.*.*.*. ^ dst=1.0.0.3)

And so on

Thus the answer will be a set of all states=

(S1 v S2 v S3 v S4 v S5)

ConfigChecker Box-- Querying the Network

- After loading the configuration files and digesting them into the unified model, CTL- (or LTL) based queries can be issued
- Configuration soundness and completeness (e.g., routing, VPN)
- Any general property-based verification
- Satisfying assignments to the CTL-based queries, are the answer to our queries.

Examples of Configuration Analysis using ConfigChecker Query Interface

	Basic reachability
Q1:	$(src = a1 \wedge dest = a2 \wedge loc(a1)) \rightarrow \mathbf{AF}(\ src = a1 \ \wedge \ dest = a2 \ \wedge \ loc(a2))$
	Given a starting location and a flow, do packets of this flow eventually reach the destination?
	Reachability Soundness
Q2:	$[loc(a1) \ \wedge \ src(a1) \ \wedge \ dst(a2) \wedge \mathbf{EF}(loc(a2))] \rightarrow \mathcal{P}connect(a1, a2)$
	If the src can reach the destination in configuration then it must be allowed in CRP.
	Reachability Completeness
Q3:	$\mathcal{P}connect(a1, a2) \rightarrow [loc(a1) \ \wedge \ src(a1) \ \wedge \ dst(a2) \ \rightarrow \ \mathbf{EF}(loc(a2))]$
	if CRP allows a1 to reach a2, then there must a path in the configuration that eventually allows a1 to reach a2.
	Discovering routing loops
Q4:	$loc(a1) \ \wedge \ \mathbf{EX}(\mathbf{EF}(loc(a1))$
	Is there a node that can reach a1 and for the same flow it is the next hop of a1?
	Shadow or Bogus routing entries
Q5:	$\mathbf{EX}(true) \wedge \neg\mathbf{EX}_{\_}(true) \ \wedge \ (loc(router1) \vee loc(router2) \ldots)$
	Given all routers, does any have a decision for traffic will never reach it from its previous hop?
	End-to-end integrity of single/nested or cascaded IPSec encrypted tunnel
Q6:	$(src = a1 \wedge dest = a2 \wedge loc(a1) \wedge IPSec(encT)) \ \rightarrow \ \mathbf{AU}((IPSec(encT) \vee (loc \rightarrow \mathcal{G}), loc(a2))$
	If the traffic is encrypted in a tunnel from the src then it will appear decrypted only at the destination or at intermediate authorized gateways ($\mathcal{G}$) that allow for cascaded tunnels. If $\mathcal{G}$=$false$, then there are no intermediate gateways and the traffic must travel through a single tunnel.
	Comparing configuration for backdoors or broken flows after route changes
Q7a:	$\mathcal{C}_{org} \triangleq [\neg multiroute \wedge src = a1 \wedge dest = a2 \wedge loc(a1) \rightarrow \mathbf{AF}(loc(a2) \wedge src = a1 \wedge dest = a2)]$
Q7b:	$\mathcal{C}_{new} \triangleq [multiroute \wedge src = a1 \wedge dest = a2 \wedge loc(a1) \rightarrow \mathbf{AF}(loc(a2) \wedge src = a1 \wedge dest = a2)]$
Q7:	*Backdoors:* $\neg\mathcal{C}_{org} \wedge \mathcal{C}_{new}$, *Broken flows:* $\neg\mathcal{C}_{new} \wedge \mathcal{C}_{org}$
	what is different in the new configuration as compared with the ordinary original one. Is there any backdoor?

Implementation

- Model and configurations:
  - Device policies and configurations are loaded and compiled into transitions in the global state machine definition.
  - Currently we support a basic text format for devices. Future format-filters can be incorporated for commercialization.
- Model Checker:
  - It was built from scratch over the BuDDy package.
  - We have 1182 variables (104 + tunnel variables)
  - BDD Optimization: Interleaving variable ordering (keep correlated variable close)
- CTL-based queries:
  - Parsed by our framework given the format as specified in our technical report.

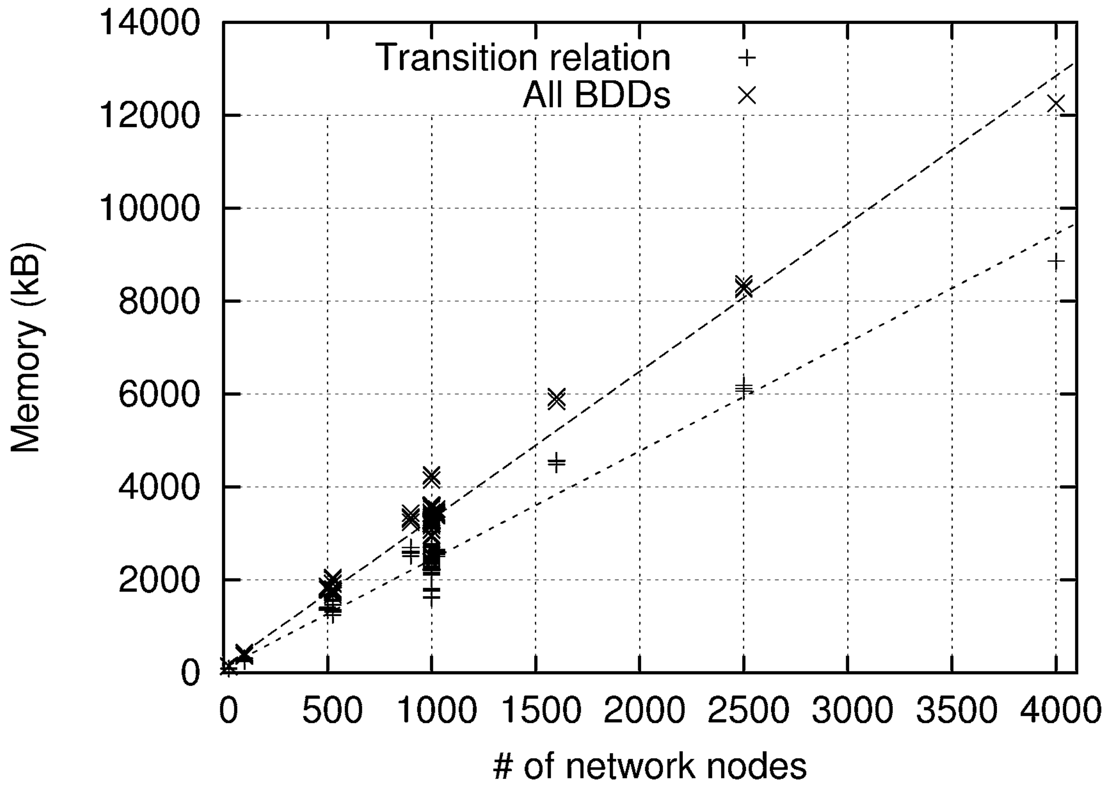
Hints about "Space Explosion" in Model Checking

- Although CTL is linear in size of the model, the model itself might be exponential with the number of variable/components in the system
- Ways to avoid state explosion
  - Using Efficient data structure like OBDD
  - Abstraction: interpret the model abstractly based on specific property
  - Partial order reduction: running several interleaving of components traces (parallelism or multithreaded)
  - Induction: some component traces can be produced by induction
  - Compositionality: breaking the verification problem into several subproblmes that can be logically composed

Evaluation

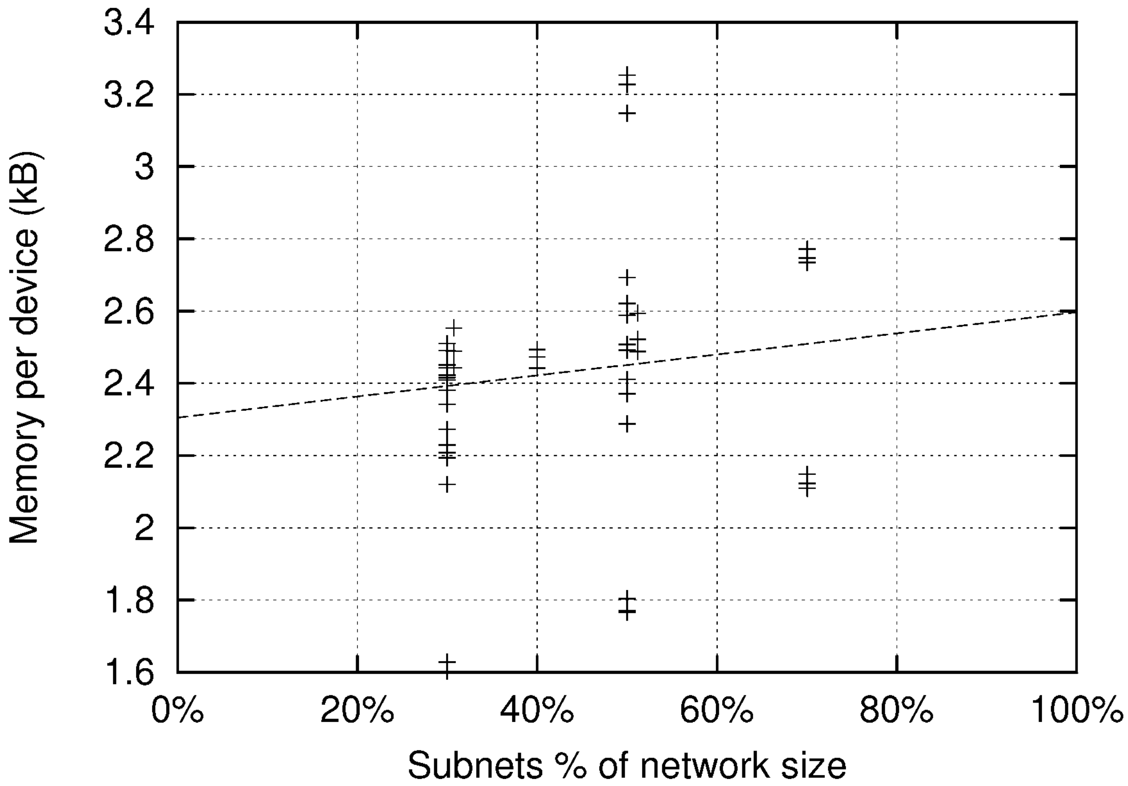
- Using 90 networks with real and random network configuration
- Random (yet reasonable) configuration is important
- Random Policy/Configuration Generation
  - Hierarchical topology network
  - Evaluation parameters: network size, policy size, rule interaction/overlapping, subnet distribution, branching factor or network depth vs. breadth, device type
  - BDD can handle up to 30K rule per device
  - Created 4000 nodes and 6M rules
  - **Details, examples of format, and configurations can be found in http://www.cyberDNA.uncc.edu/projects/ConfigChecker**
- We measure the space requirement and building time
  - Query time is negligible in most of the case

Evaluation



- Memory Required versus Network size
  - The growth is evidently linear in both transition relation size and in overall BDD table entry count.

Evaluation



- Space versus number of rules
  - Increase then almost steady state

Summary

- BDD Pros and Cons

  + powerful canonical representation

  + powerful logical operation: manipulating, testing

    - Each step polynomial complexity

  + Maintain "closure" property

  + Compactness (size usually stay small at least for many applications)

    we used firewall 30K rules for Cisco and 5 millions rules in network testing

  + Efficient Quantification operations

  - Too big for some problems

  - Weak for search problems

  - Must be careful to choose good variable ordering

  - Must have good insights into problem characteristics

Summary

- ConfigCheacker provides a novel approach for end-to-end black box network security configuration verification and analysis
- It provides a flexible, extensible framework rather than addressing specific misconfiguration problems
- Model checker looks scalable for this application domain
  - 4K nodes and 6+ Millions of rules ➔ Max 14M and order of minutes
  - O(V) instead of O(V^3) – ignoring the cost of set/bdd operations
  - Wildcard; common prefixes; overlapping rules, and variable ordering
- Supporting rich and logically expressive interfaces such as CTL is powerful and important, although clumsy for regular users

(Selected) References

[BDD] **An Introduction to Binary Decision Diagrams**, Henrik Reif Anderson, Lecture notes for 49285 advanced algorithms E97, 1997.
[CS] **Logic in Computer Science Modeling and Reasoning about Systems**, Michael Huth and Mark ryan, Cambridge University Press, 2004.
[SS] **System and Software Verification,** B. Berard ate
[CA] **Computer Aided Verification of Coordinating Processes**, Robert Kurshan, 1995
[MC] **Model Checking** (E. Clark ate)
[DP] **Decision Procedure– An Algorithmic Point of View,** D. Kroening and O. Strichman
[ICNP05] Hazem Hamed, Ehab Al-Shaer and Will Marrero, **Modeling and Verification of IPSec and VPN Security Policies**, IEEE ICNP'2005, November 2005
[ICNP09] Ehab Al-Shaer, Will Marrero, Adel El-Atawy and Khalid Elbadawi, **Network Security Configuration in A Box: End-to-End Security Configuration Verification**, IEEE International Conference in Network Protocols (ICNP' 09), October 2009
[JSAC05] Ehab Al-Shaer, Hazem Hamed, Raouf Boutaba and Masum Hasan, **Conflict Classification and Analysis of Distributed Firewall Policies**, IEEE Journal on Selected Areas in Communications, Issue: 10, Volume: 23, Pages: 2069 - 2084, October 2005
[INFOCOM04] Hazem Hamed and Ehab Al-Shaer, **Anomaly Discovery in Distributed Firewalls**, IEEE INFOCOM'04, March 2004
[JSAC08] Ehab Al-Shaer, Adel El-Atawy and Taghrid Samak, **Automated Pseudo-live Testing of Firewall Configuration Enforcement**, IEEE Journal on Selected Areas in Communications, Issue: 3, Volume: 27 , April 2009
[Malik] Robi Malik Slides in Model Checking on SPIN, Waikato Univ, Australia.