DRBD: Dynamic Reliability Block Diagram for System Reliability Modeling

Prof. Haiping Xu
Concurrent Software Systems Laboratory
Computer and Information Science Department
University of Massachusetts Dartmouth

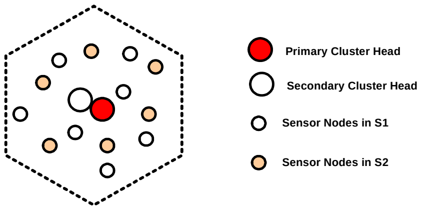
Acknowledgement

- **Dr. Liudong Xing,** Assistant Professor
  Electrical and Computer Engineering Department
  University of Massachusetts Dartmouth
- **Ryan Robidoux,** Graduate Student
  Concurrent Software Systems Laboratory
  Computer and Information Science Department
  University of Massachusetts Dartmouth

Outline

- DRBD controller component blocks
- Development of DRBD models (example)
- Formal specifications of DRBD constructs
- Formal verification of DRBD models
- Conversion of DRBD models into colored Petri nets (*CPN*)
- Case study: modeling, verification
- Conclusions and future work

A Motivating Example



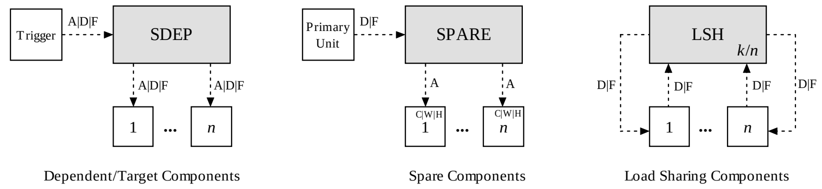
Initially, sensor nodes in *S*1 are operational; sensor nodes in *S*2 are in a sleeping mode

- When the primary cluster head fails, the secondary cluster head will be automatically activated.
- Sensor nodes in *S*1 can be put into a sleeping mode, and sensor nodes in *S*2 will be activated.
- How to model the state dependency between *S*1 and *S*2: Deactivation -> Activation dependency?

The State of the Art

- Most of the existing reliability modeling tools (e.g., RBD) cannot capture the <u>state dependency</u> between components.
- Other tools, such as Dynamic Fault Tree (DFT), may support modeling a functional dependency
  - The failure of a component causes some other dependent components to become inaccessible or unusable
  - However, it still cannot capture the <u>Deactivation -> Activation</u> state dependency between components.
- We propose a set of new ***D***ynamic ***R***eliability ***B***lock ***D***iagram (DRBD) constructs as an extension to the existing RBD modeling tool.

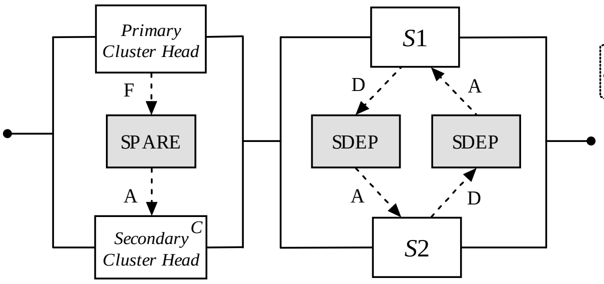
DRBD Controller Component Blocks

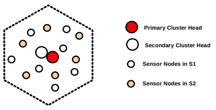


(a) SDEP (state dependency) block (b) SPARE (spare part) block (c) LSH (load sharing) block

- ***A*** stands for an <u>activation</u> event occurred on a component that leads to an *Active* state of that component,
- ***D*** stands for a <u>deactivation</u> event occurred on a component that leads to a *Standby* state of that component, and
- ***F*** stands for a <u>failure</u> event occurred on a component that leads to a *Failed* state of that component.

DRBD Model of the WSN Example



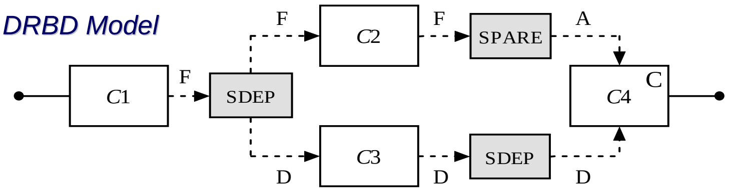


A: Activation
D: Deactivation
F: Failure

- The failure of the primary cluster head will automatically activate the secondary cluster head.
- The components labeled *S*1 and *S*2 represent the two sets of sensor nodes that may work alternatively.
- The deactivation of *S*1 (*S*2) will automatically activate *S*2 (*S*1).

Formal Specifications

DRBD Model



- To support formal verification and validation of our proposed DRBD model, it is necessary to formally define the DRBD modeling constructs.
  - Provide the denotational semantics for the development of DRBD models in a precise manner.
  - Help to eliminate ambiguity in a constructed DRBD model.
- ***Question 1:*** When component *C*1 fails, will *C*4 be in a state of *Active* or *Standby*, or will the result be nondeterministic?

Object-Z Specification

$Event ::= Activation \mid Deactivation \mid Failure$

$$
\begin{array}{l}
\textit{SDEP} \\
\hline
trigger : Component \\
targets : \mathbb{P}\, Component \\
nTargets : \mathbb{N} \\
triggerEvent : Event \\
targetEvents : Component \rightarrow Event \\
sdep : \mathbb{T} \times Component \times Event \rightarrow \mathbb{P}(\mathbb{T} \times Component \times Event) \\
\hline
nTargets = \#targets \land nTargets > 0 \land targets = \operatorname{dom} targetEvents \\
\forall\, c \in targets \bullet c \neq trigger \land probability(c \mid triggerEvent) \neq probability(c) \\
\quad \land\ probability(triggerEvent \mid c) = probability(triggerEvent) \\
\{(t, trigger, triggerEvent) \mid t \in \mathbb{T}\} = \operatorname{dom} sdep \\
\hline
\textit{ActivateTrigger} \\
\hline
\Delta(trigger, targets) \\
t? : \mathbb{T} \\
\hline
(triggerEvent = Active) \land (trigger.state' = Active) \\
\forall\, c \in targets \bullet (t? + \delta_c, c, targetEvents(c)) \in sdep(t?, trigger, triggerEvent) \\
\quad \land\ ((targetEvents(c) = Activation \land c.state' = Active) \\
\quad \lor\ (targetEvents(c) = Deactivation \land c.state' = Standby) \\
\quad \lor\ (targetEvents(c) = Failure \land c.state' = Failed)) \\
\hline
\textit{DeactivateTrigger} \\
\hline
\textit{FailTrigger} \\
\hline
\end{array}
$$

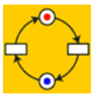
- The target events do not occur simultaneously, but with some random time delay δ_c for target component *c.*
- The failure of *C*2 and deactivation of *C*3 will not happen immediately after the failure of *C*1.
- Which state *C*4 will be in (*Active* or *Standby*) is nondeterministic.
- ***Question 2***: How can we be confident that the model is an accurate representation of the actual system?

Formal Verification Approach

- Testing or simulations are not suitable for verifying DRBD models because it is almost impossible to cover all cases.
- Use formal methods (e.g., model checking techniques) to verify the behavioral properties of a DRBD model before the evaluation process starts.
- Use temporal logic to specify system properties
  - **Property P:** "If component *A* fails, component *B* and *C* will also fail, which leads to the failure of the whole system *S*."
  - The temporal formula in LTL (Linear Temporal Logic) can be written as `[](¬A→(¬B∧¬C)∧<>¬S)`
- When a DRBD model is proved to be incorrect
  - Any quantitative evaluation results might be unusable.
  - The DRBD model needs to be fixed.

Formal Verification Models

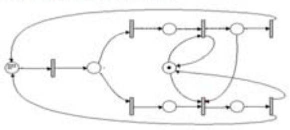
- DRBD models are not formally defined & executable.
- Object-Z specifications of DRBD constructs are formal specifications, however
  - Are not feasible for verification of behavioral properties.
  - Have no effective analysis and verification tool support.
- Convert a DRBD model into a formal executable model such as a state machine or a Petri net model.
- We adopt Colored Petri Net (CPN) model because
  - Is user friendly based on its graphical notations.
  - Has powerful, but intuitive rules for defining structure and dynamic behaviors.
  - Has many existing analysis and verification tools.



CPN

Introduction to Petri Net

- “Three-in-one” capability of Petri net models [Murata 1989]
  - Graphical representation
  - Mathematical description
  - Simulation tool
- Definition:

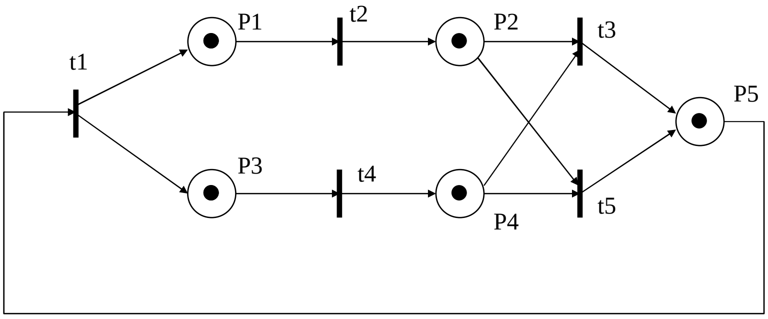


Readers Writers (classical problem)

A Petri net is a 4-tuple, PN = (P, T, F, M_0) where

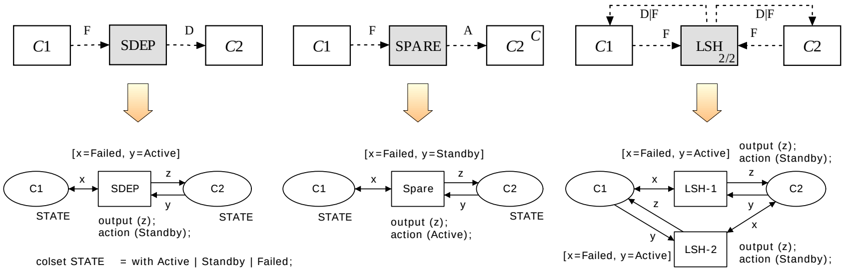
P = {P1, P2, …, Pm} is a finite set of places;
T = {t1, t2, …, tn} is a finite set of transitions;
F $\subseteq$ (P x T) $\cup$ (T x P) is a set of arcs (flow relation);
M_0: P --> {0, 1, 2, 3, …} is the initial marking.

An Ordinary Petri Net



- In an ordinary Petri net, tokens are all of color **black**.
- In a Colored Petri net (*CPN* or *CP-net*),
  - Colors of tokens can represent values.
  - A transition may have a guard and executable code.

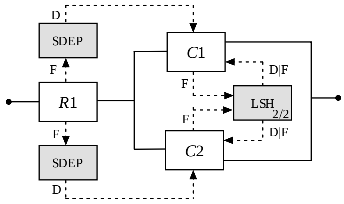
Convert DBBD into CPN Models



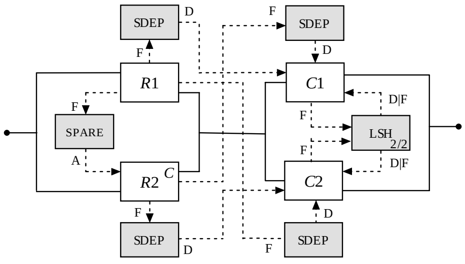
```
colset STATE    = with Active | Standby | Failed;
var x, y, z      : STATE;
```

- Define three different colors/states: *Active*, *Standby* and *Failed*.
- A transition is associated with a guard and executable code
  - Can fire <u>only if</u> the guard `[x=Failed, y=Active]` evaluates to *true*.
  - Code `output(z);action(Standby)`deposits a *Standby* token in *C*2.

A Case Study



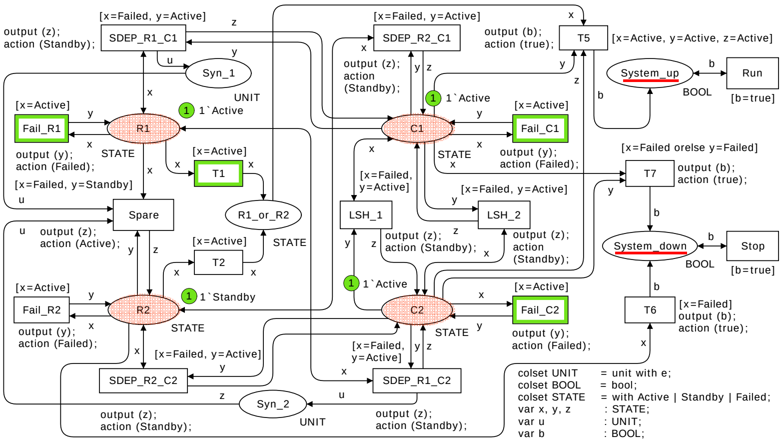
(a) Load sharing servers connected to a router



(b) Load sharing servers connected to a router with a CSP

- Router *R*1 is connected to two server computers *C*1 and *C*2.
  - Server computers *C*1 and *C*2 are load sharing servers.
  - When router *R*1 fails, the computers *C*1 and *C*2 will be deactivated.
- To make the system more reliable, we introduce a cold spare (*CSP*) for router *R*1, which is represented by component *R*2.

Colored Petri Net Model



```
colset UNIT   = unit with e;
colset BOOL   = bool;
colset STATE  = with Active | Standby | Failed;
var x, y, z    : STATE;
var u          : UNIT;
var b          : BOOL;
```

Analysis Results

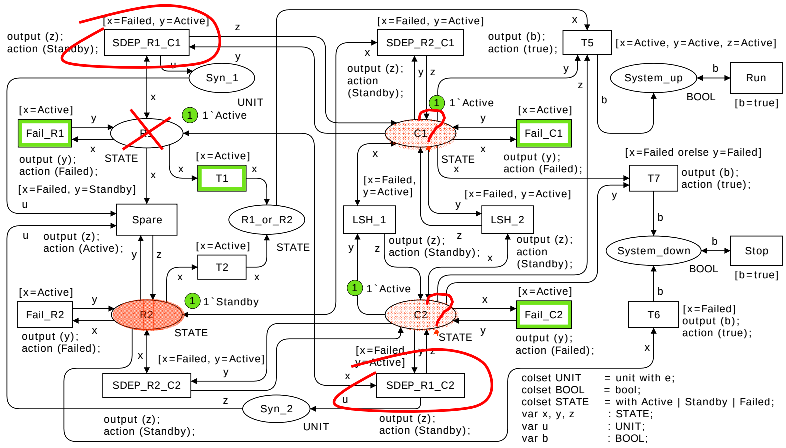
Result-1

```
Statistics
-------------------------
State Space
     Nodes:  33
     Arcs:   69
     Secs:   0
     Status: Full
Scc Graph
     Nodes:  33
     Arcs:   62
     Secs:   0
Liveness Properties
-------------------------
Dead Markings [32]
Dead Transition Instances
     Router'SDEP_R2_C1 1
     Router'SDEP_R2_C2 1
Live Transition Instances
     None
```

Result-2

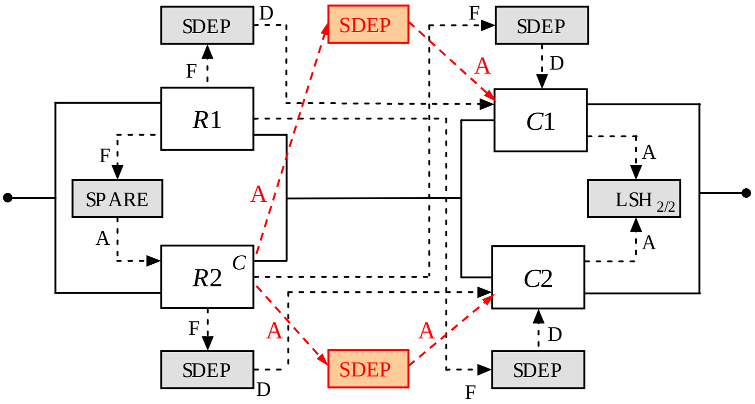
```
DeadMarking(32)
-------------------------
val it = true : bool
print(NodeDescriptor 32)
-------------------------
32:
C1 1: 1`Standby
C2 1: 1`Standby
R1 1: empty
R2 1: empty
R1_or_R2 1: 1`Active
Syn_1 1: empty
Syn_2 1: empty
System_down 1: empty
System_up 1: empty
val it = () : unit
Reachable'(1, 32)
-------------------------
A path from node 1 to 32: [1, 3, 11,
   25, 30, 32]
val it = true : bool
```

Deadlock in CPN



```
colset UNIT   = unit with e;
colset BOOL   = bool;
colset STATE  = with Active | Standby | Failed;
var x, y, z    : STATE;
var u          : UNIT;
var b          : BOOL;
```

Revised DRBD Model



Analysis Results (after revision)

Result-3

```
Statistics
-------------------------
State Space
     Nodes:  67
     Arcs:   162
     Secs:   0
     Status: Full
Scc Graph
     Nodes:  67
     Arcs:   141
     Secs:   0
Liveness Properties
-------------------------
Dead Markings
     None
Dead Transition Instances
     None
Live Transition Instances
     None
```

- Fix the colored Petri net model by adding
  - New transition *SDEP_R2_C12*
  - New synchronization *place Syn_3*
  - And arcs and guards
- The analysis results show no deadlock markings.
- ***Question 3***: How to verify additional properties?

Model Checking Results

Formulas	ASK-CTL in ML	After Rev	Before Rev
Formula_1	val myASKCTLformula = EXIST_UNTIL(TT,NOT(MODAL(TT))); eval_node myASKCTLformula InitNode;	false	true
Functions	fun R1_Failed n = (Mark.R1 1 n = 1`Failed); fun R2_Failed n = (Mark.R2 1 n = 1`Failed); fun SystemFailed n = (Mark.System_down 1 n = 1`true);	-	-

Formula_2	val isFailed = FORALL_UNTIL(TT, NF("",SystemFailed));
	val system = OR(NOT(NF("", R2_Failed)), isFailed);
	val myASKCTLformula = INV(system);
	eval_node myASKCTLformula InitNode
		true	true

Formula_3	val isFailed = FORALL_UNTIL(TT, NF("",SystemFailed));
	val system = OR(NOT(NF("", R1_Failed)), isFailed);
	val myASKCTLformula = INV(system);
	eval_node myASKCTLformula InitNode;
		false	true

Conclusions and Future Work

- Proposed a new modeling approach called Dynamic Reliability Block Diagrams (DRBD)
  - Resolves the shortcomings of the existing work.
  - Provides a powerful but easy-to-use reliability modeling tool for complex and large computer-based systems.
  - Supports automated verification of DRBD models.
- Develop a software tool that can automatically translate DRBD models into colored Petri nets.
- Study efficient evaluation methods for DRBD models.
- Develop a comprehensive system reliability modeling tool that supports editing, formal verification, and evaluation of DRBD models.

Related Publications

- **R. Robidoux, H. Xu, and L. Xing** Towards Automated Verification of Dynamic Reliability Block Diagrams. *To be submitted to journal*, Computer and Information Science Dept., UMass Dartmouth, November 2007.
- **L. Xing, H. Xu, S. V. Amari, and W. Wang** A New Framework for Complex System Reliability Analysis: Modeling, Verification, and Evaluation. *Submitted to Journal of Autonomic and Trusted Computing (JoATC)*, September 2007.
- **H. Xu, L. Xing, and R. Robidoux** DRBD: Dynamic Reliability Block Diagrams for System Reliability Modeling. *Submitted to International Journal of Computers and Applications (IJCA)*, August 2007.
- **H. Xu and L. Xing** Formal Semantics and Verification of Dynamic Reliability Block Diagrams for System Reliability Modeling. In *Proceedings of the 11th International Conference on Software Engineering and Applications (SEA 2007)*, November 19-21, 2007, Cambridge, Massachusetts, USA.

Contact Information

Haiping Xu, Assistant Professor
Computer and Information Science (CIS)
Department, College of Engineering
University of Massachusetts Dartmouth
Phone : (508) 910-6427
Email: hxu@umassd.edu

Liudong Xing, Assistant Professor
Electrical and Computer Engineering (ECE)
Department, College of Engineering
University of Massachusetts Dartmouth
Phone : (508) 999-8883
Email: lxing@umassd.edu

Questions?

The slides for this talk can be downloaded from

http://www.cis.umassd.edu/~hxu