Survivability Applications

- This sequence is based on the paper:
  - Li Tan, and Axel W. Krings, “An Adaptive N-variant Software Architecture for Multi-Core Platforms: Models and Performance Analysis”, The 11th International Conference on Computational Science and Its Applications (ICCSA 2011), in Lecture Notes on Computer Science (LNCS), Springer Verlag, 2011, (16 pages)
  - Other material is from the references of that publication
  - The focus here is on system architectures for survivability and formal analysis tools.

Multi-core Systems

- **They are here and they will grow!**
- Assumptions about the future of multi-core
  - number of cores is increasing
  - most applications still have limited means of using multi-threading
  - degree of parallelism is bound by the largest anti-chain of the execution graph
  - implications on speedup

Reliability and Redundancy

- Redundancy has greatly benefitted reliability
- In the past: homogeneous redundancy
- New focus on heterogeneous redundancy
  - avoidance of common mode faults

Common Mode Faults

- If a SW/HW component fails under a certain input, then it does not matter how many identical components one uses for redundancy => **they all fail**

- Dissimilarity as an approach toward independence of faults

- Two main approaches
  - N-version software
  - N-variant software

N-version Software

- N-version programming (late 70s)
  - software is derived by multiple teams from the same specification in isolation
  - expectation: common mode fault is reduced or eliminated
  - different results by different versions indicate fault
  - limitations
    - how dissimilar are implementations?
    - is there true independence of development?
    - how does one measure the “degree of dissimilarity”?

N-variant Software

- Inspired by N-version software
- Different variants are generated in a more “automated” fashion
- Expectation is that a fault affecting on variant will not affect another in an identical way
- Again, differences detected by different variants indicate fault

Resilient Multi-core systems

- Utilize idle resources to increase resilience
- Specifically

Utilize idle cores for resilience mechanisms

Related work

- Towards Byzantine Fault Tolerance in Many-core Computing Platforms, Casey M. Jeffery and Renato J. O. Figueiredo, 13th IEEE International Symposium on Pacific Rim Dependable Computing, 2007
  - Focus on transient faults

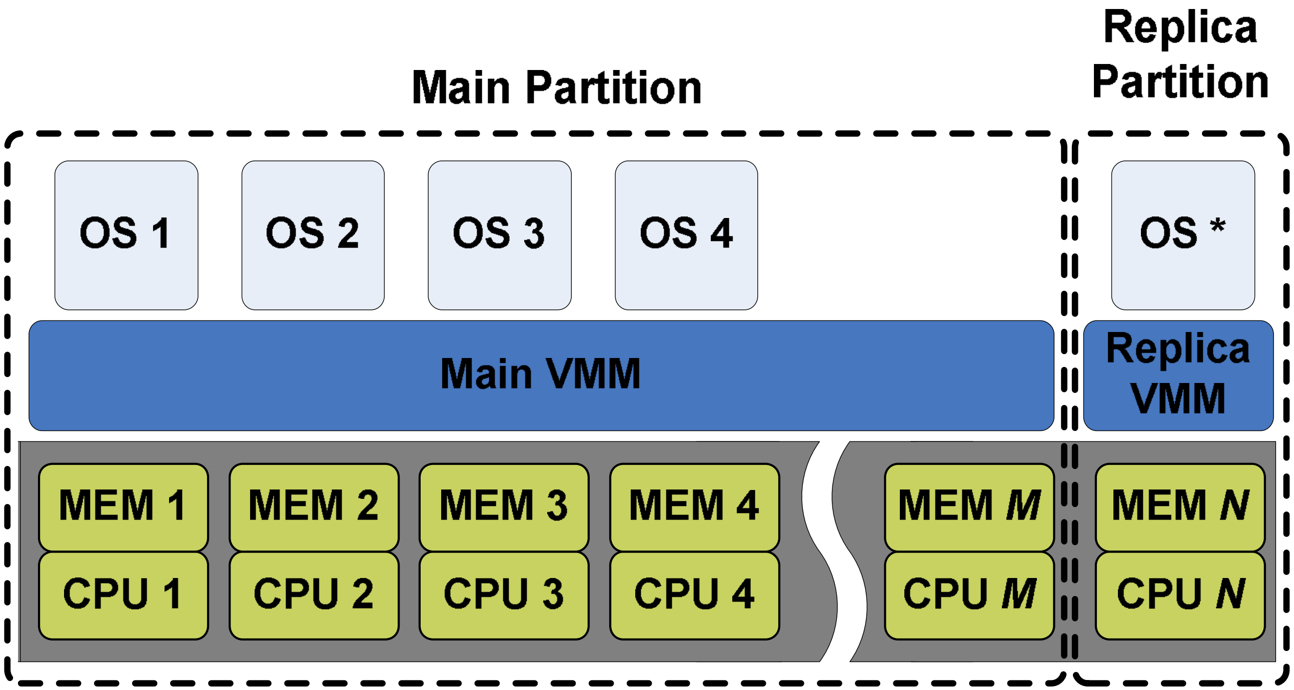


Figure 1. Many-core model with replica partition.

Related work [Cox2006]

- N-Variant Systems A Secretless Framework for Security through Diversity, B. Cox, et. al., USENIX, 2006
    - A set of automatically diversified variants execute on same inputs
    - Difference in referencing memory is observed
    - Identifies execution of injected code
    - Check out section **3. Model** of their paper

Related work [Cox2006]

- Example of two variants using disjoint memory space. Any absolute memory access will be invalid in one the variants

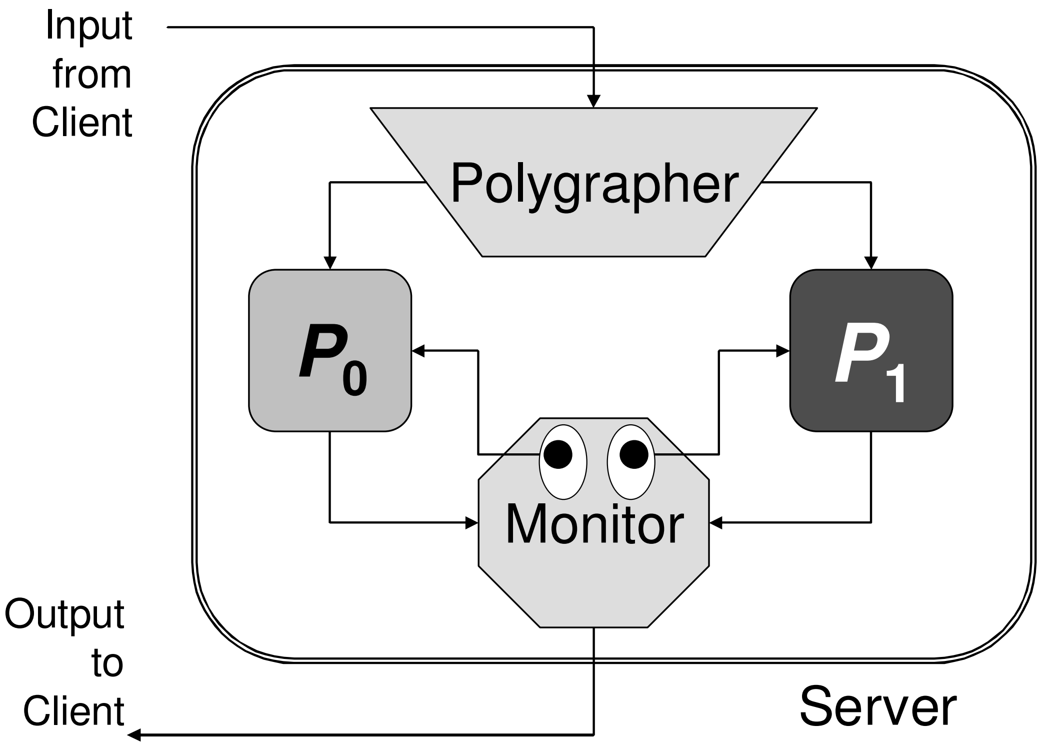


Figure 1. N-Variant System Framework.

[Nguyen-Tuong 2008]

- Security through redundant data diversity
  - Anh Nguyen-Tuong, David Evans, John C. Knight, Benjamin Cox, Jack W. Davidson
  - 38th IEEE/IFPF International Conference on Dependable Systems and Networks, Dependable Computing and Communications Symposium. Anchorage, June 2008.

[Nguyen-Tuong 2008]

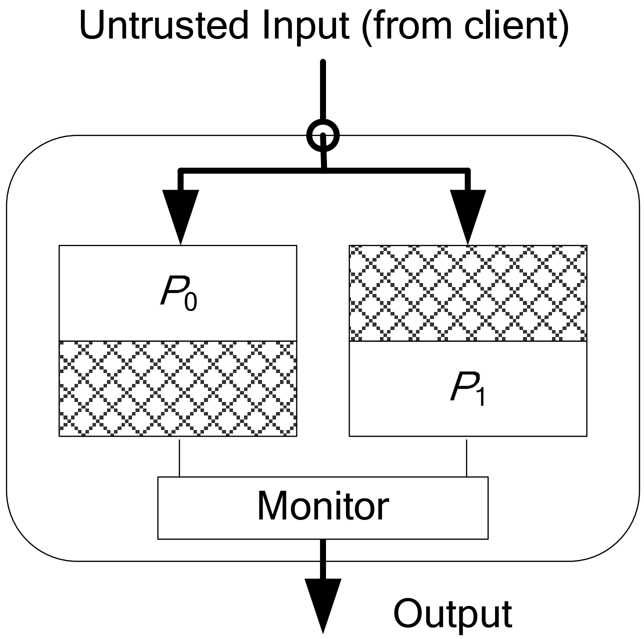


Figure 1. Two-variant address partitioning.

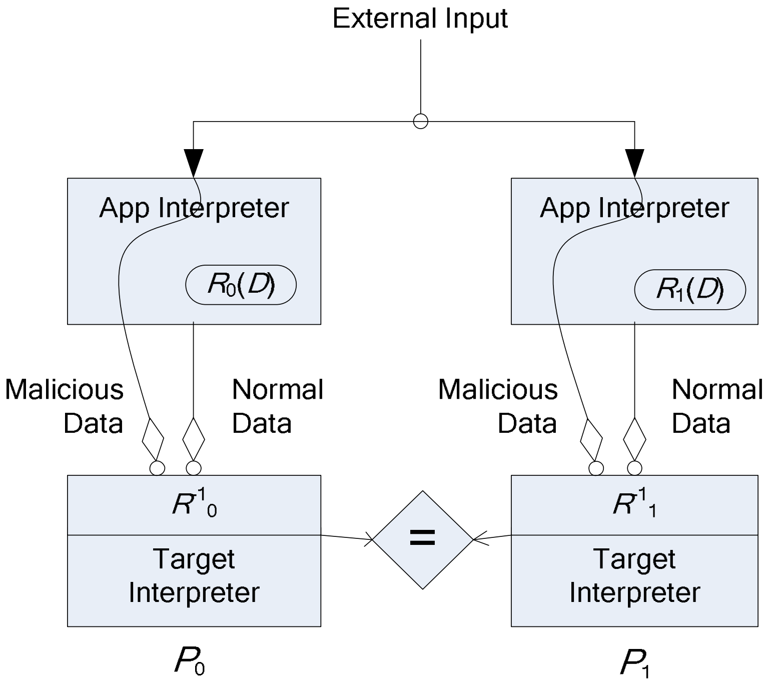


Figure 2. N-Variant Systems with Data Diversity.

Related work [Salamat2008]

- B. Salamat, et. al. 2008
  - Multi-Variant Program Execution: Using Multi-Core Systems to Defuse Buffer-Overflow Vulnerabilities
  - International Conference on Complex, Intelligent and Software Intensive Systems
  - Variants use different direction in memory allocation
  - Buffer overflow “crashes” into different neighboring memory

Related work [Salamat2008]

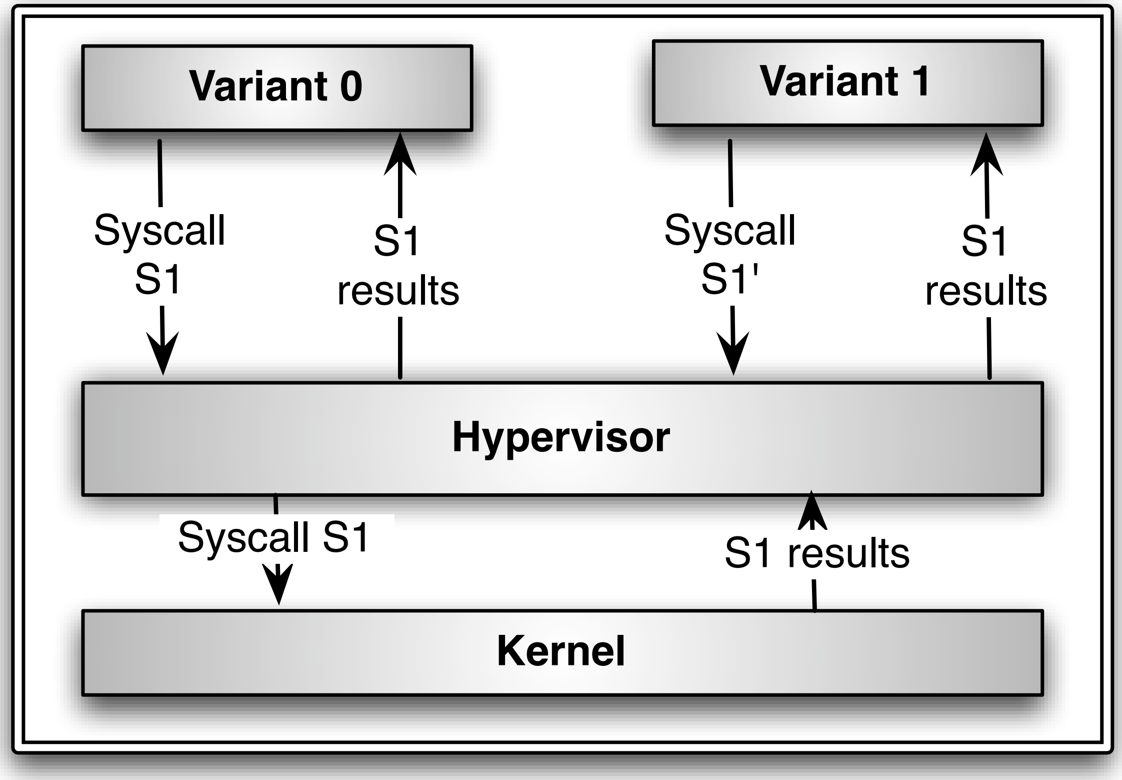


Figure 1. System calls that change the global state are executed by the monitor and the results are communicated to all instances.

General Scheme

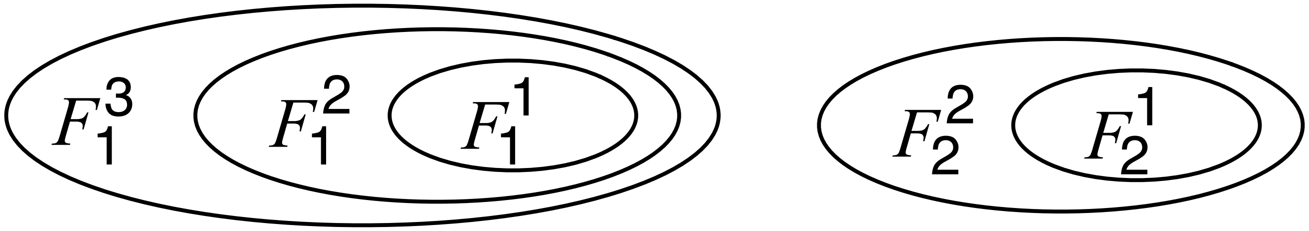
- Execution of multiple versions masks or detects faults
- Overhead
    - N-folding amount of work
    - Redundancy management
    - What can be absorbed?

Two Step Approach

- Specification Model
- Layered adaptive architecture

Specification Model

- Adaptive Functional Capability Model (AFCM)
  - System comprised of functionalities $F_1 \cdots F_m$
    - core operations that are mission critical
    - non-critical, but value-added operations

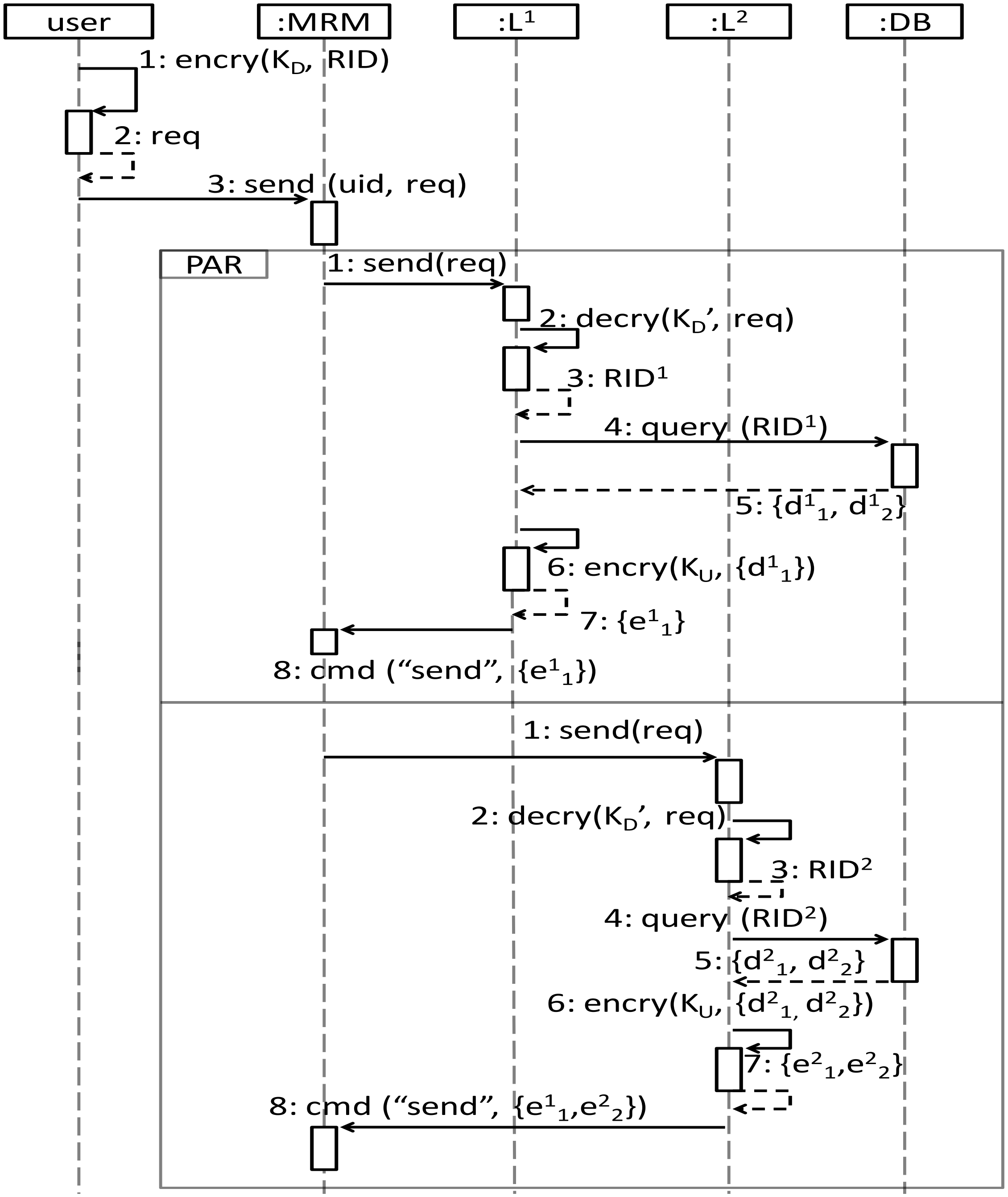


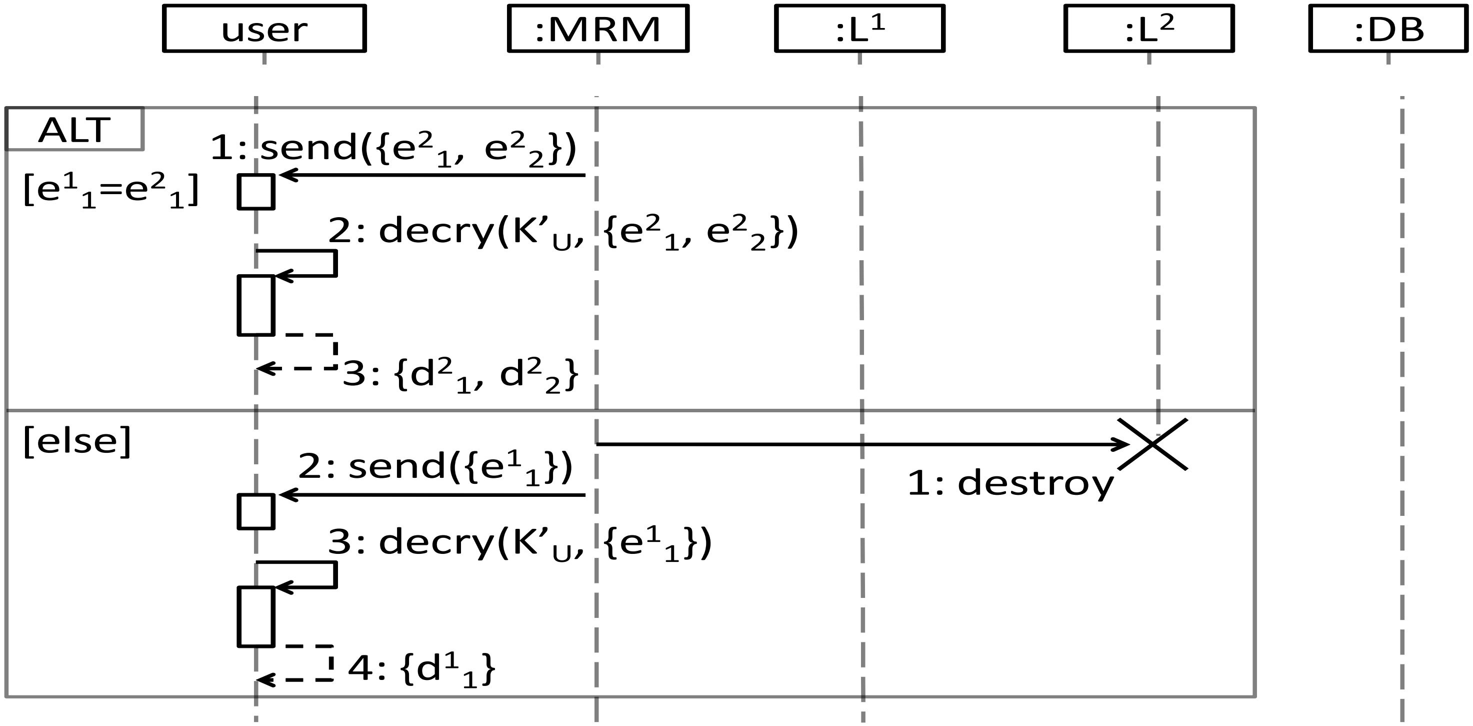
$$F_1^1 \preceq F_1^2 \preceq F_1^3$$

Example: Multi-level Secured Record Keeping

Example

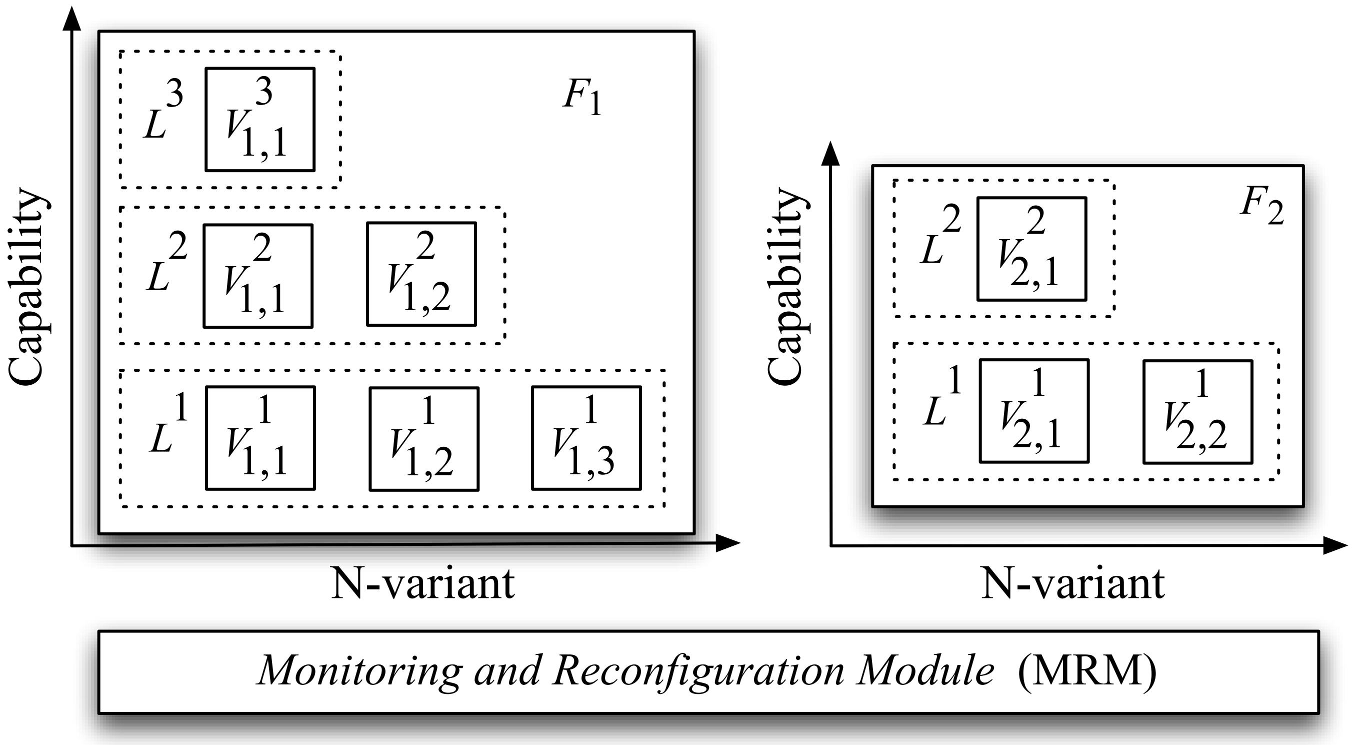
- Secured database system *D*
  - each record in *D* contains two sets of data, i.e., $d = \{d_1, d_2\}$
  - d_1 contains mission critical data
  - d_2 non-mission critical, but value-added data


user
:MRM
:L1
:L2
:DB
1: encry(KD, RID)
2: req
3: send (uid, req)
PAR
1: send(req)
2: decry(KD', req)
3: RID1
4: query (RID1)
5: {d11, d12}
6: encry(KU, {d11})
7: {e11}
8: cmd ("send", {e11})
1: send(req)
2: decry(KD', req)
3: RID2
4: query (RID2)
5: {d21, d22}
6: encry(KU, {d21, d22})
7: {e21,e22}
8: cmd ("send", {e11,e22})


user
:MRM
:L¹
:L²
:DB
ALT
[e¹₁=e²₁]
1: send({e²₁, e²₂})
2: decry(K'ᵤ, {e²₁, e²₂})
3: {d²₁, d²₂}
[else]
1: destroy
2: send({e¹₁})
3: decry(K'ᵤ, {e¹₁})
4: {d¹₁}

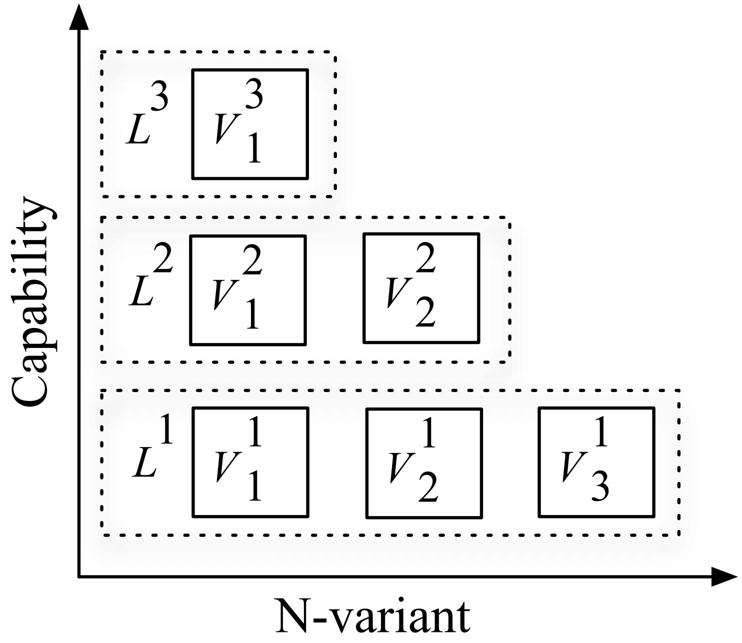
Layered N-variant Architecture

- Multiple functionalities:
  - System is a collection of functionalities



Adaptability and Reconfiguration

- Layers have two purposes
  - lower layer monitors higher layer
  - layers are basis for reconfiguration
  - disagreement results in
    - scaling back to lower layer
    - graceful degradation



Special Cases

- Limitation of current research
  - all functionalities are defined on same layer

- Salamat, et. al. 2008
  - use two variants at the same layer, i.e., layer L_1

$$V_1^1 \text{ and } V_2^1$$

  - the two variants focus on memory referencing

Special Cases

- Cox, et. al. 2006
  - use variants at the same layer, i.e., layer L_1
  - the variants focus on memory referencing

Matching expectations

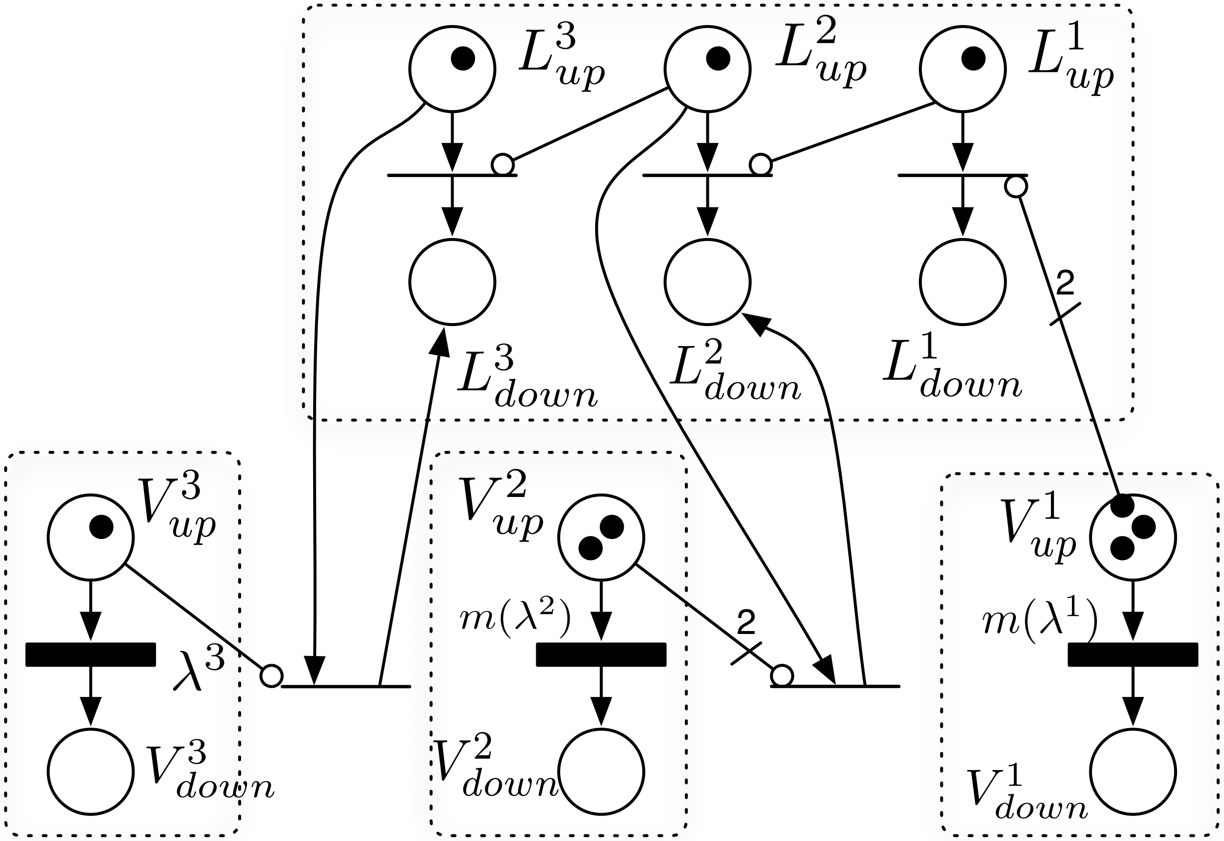
- Specify a suitable system
  - get an idea with GSPN model (Gen. Stochastic Petri Nets)
    - see if/how goal can be met
    - see if the overhead realistic
- Implementation
  - probabilistic automaton-based model
    - closer to real behavior
    - starting point towards implementation

Petri Nets

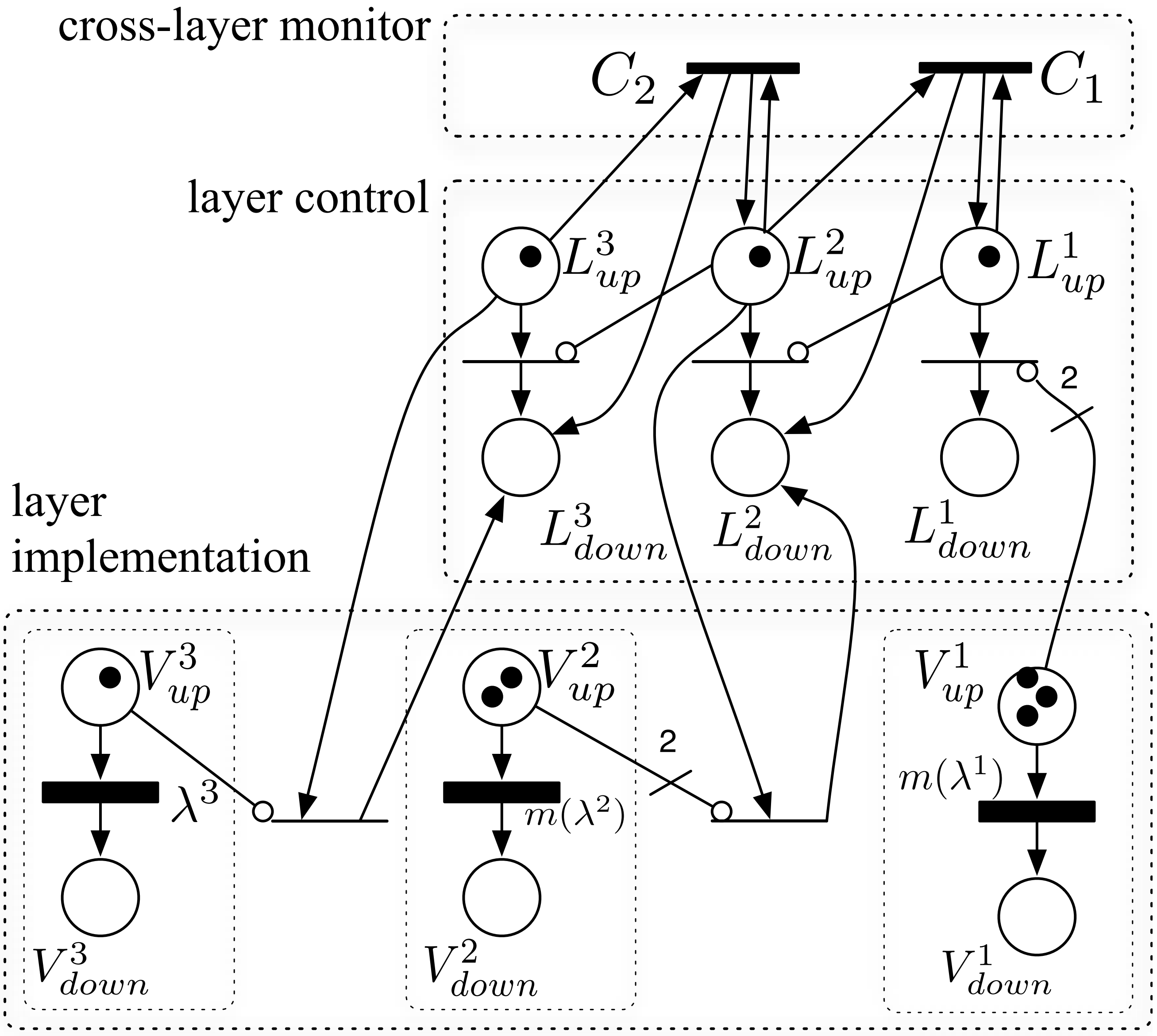
- From Markov Chains to Petri Nets
  - discussion on Markov Chains
  - discussion on Petri Nets
  - you will not be an expert based on this discussion, but you should understand the general ideas, the strength and mathematical/computational limitations.

extra page for notes

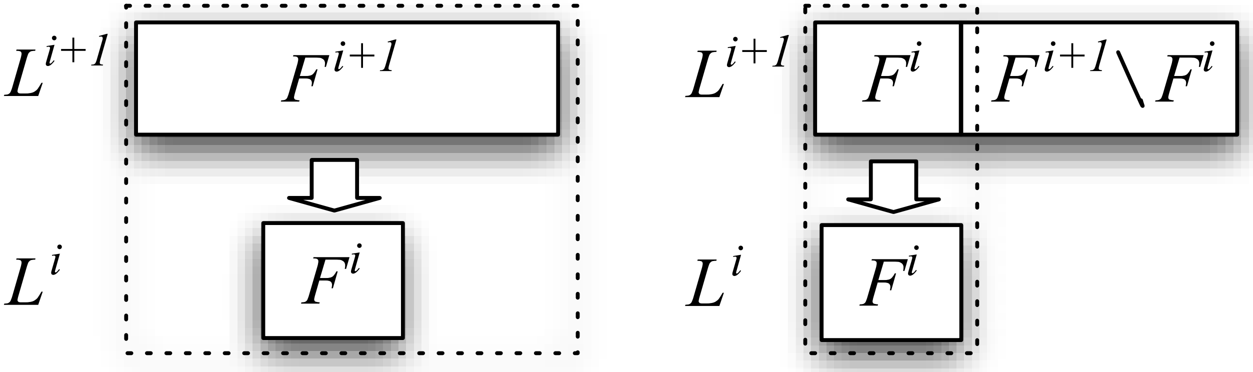
Reliability and Resilience



Reliability and Resilience

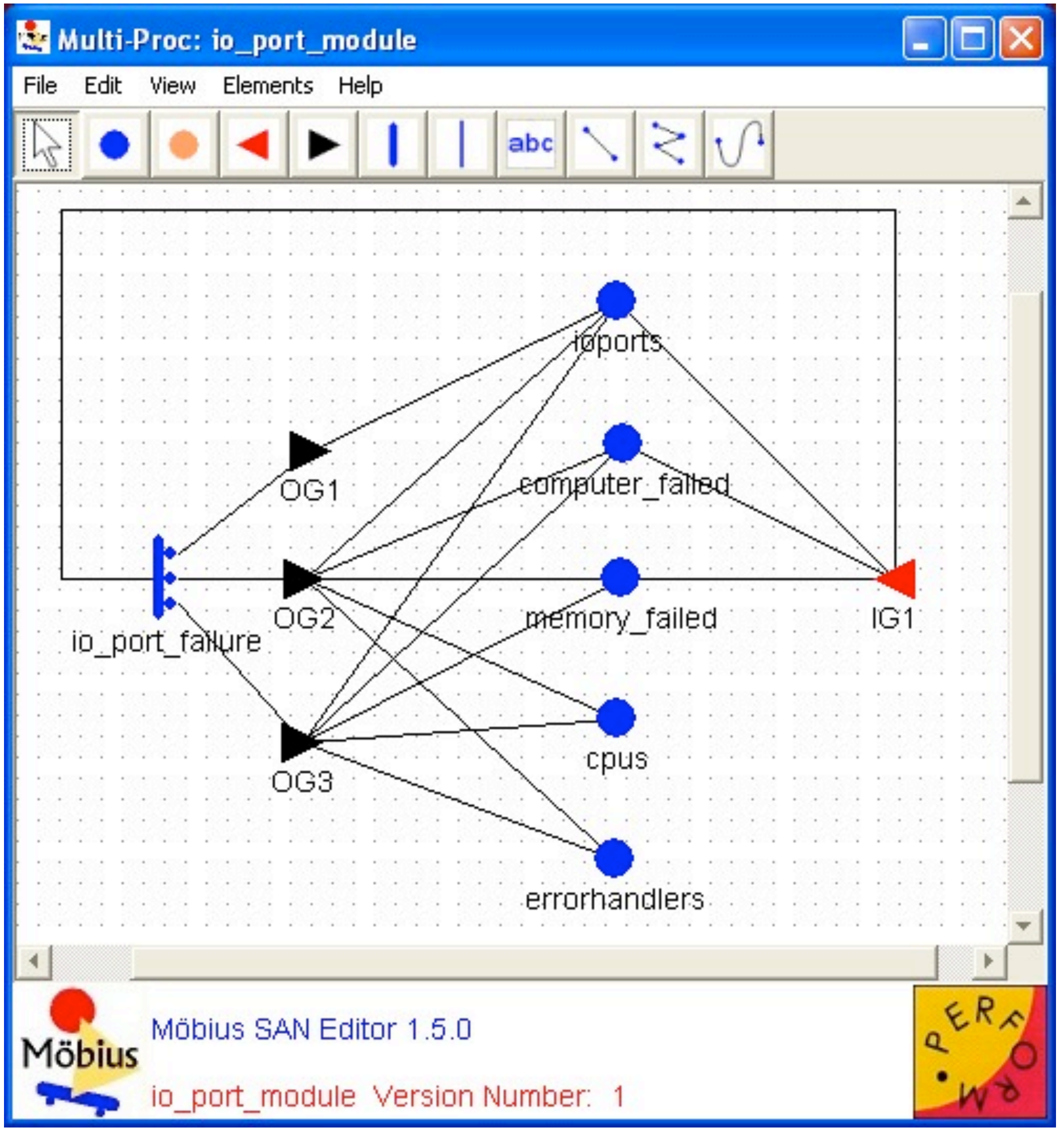


Cross-layer monitoring scope



Stochastic Activity Networks

- Example: Möbius



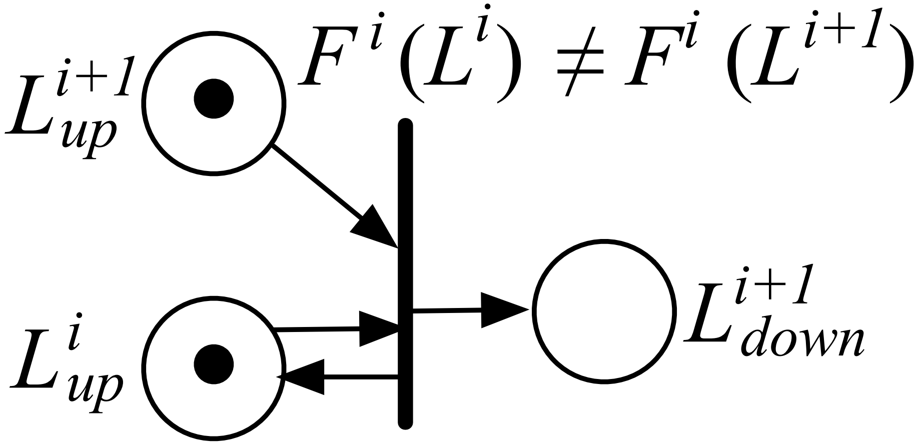
Möbius SAN Editor 1.5.0

io_port_module Version Number: 1

check out
www.mobius.illinois.edu

SAN for cross-layer monitoring

- Note the difference between GSPN and SAN (Stochastic Activity Network)



Stochastic Models

- Evaluation of performance of architecture
  - model stochastic behavior using probabilistic models
  - use probabilistic model checking
- Metrics of interest
  - service availability
  - information security

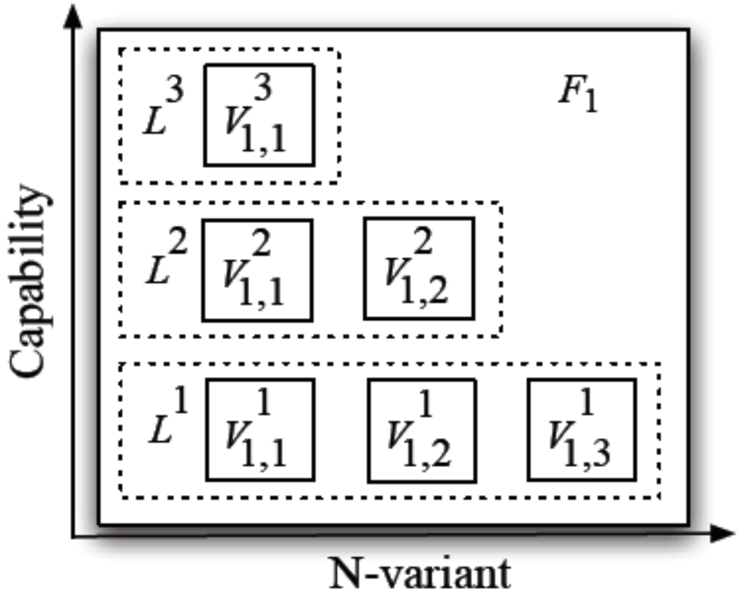
Probabilistic Automata

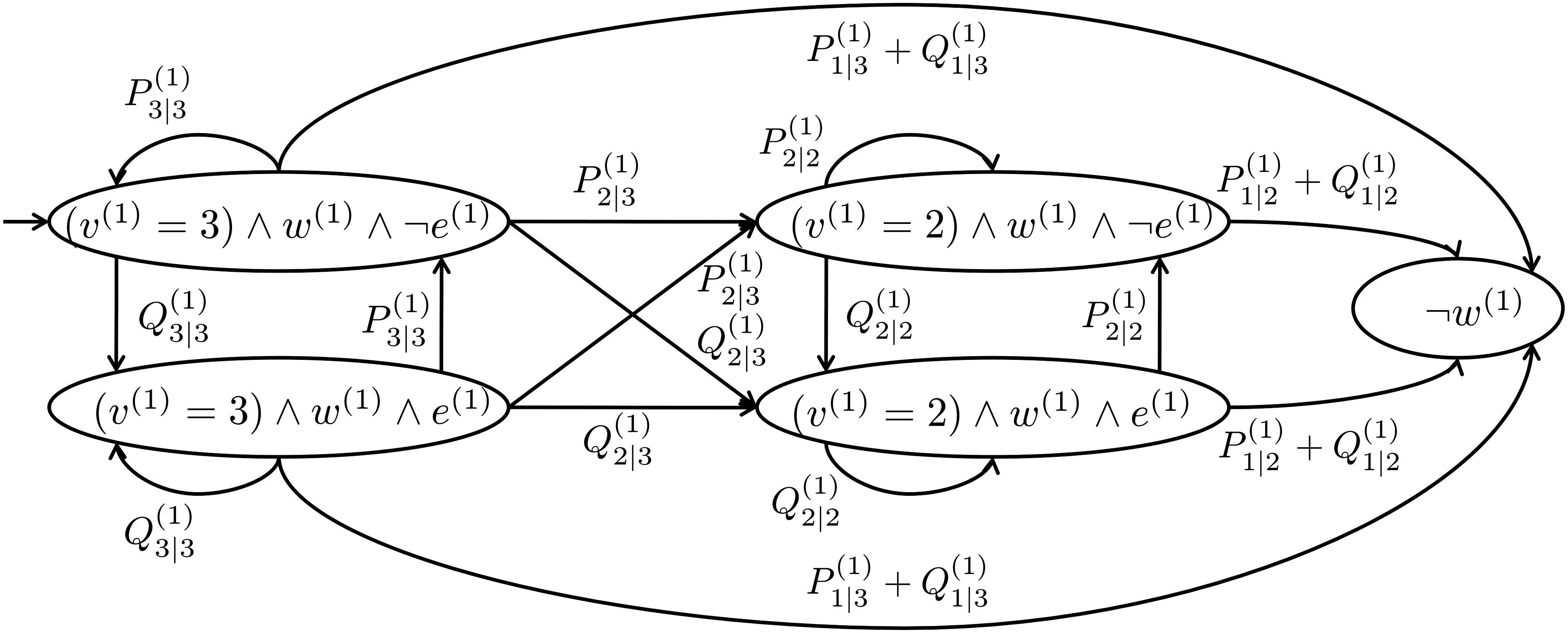
N-tuple $\langle Q, \mathit{\Theta}, \delta, Q_0, F, P_\delta, P_0 \rangle$

1. Q is a set of states,
2. $\mathit{\Theta}$ is a set of input symbols,
3. $\delta \subseteq Q \times \mathit{\Theta} \times Q$ is a set of transitions,
4. $Q_0 \subseteq Q$ is a set of start states,
5. $F \subseteq Q$ is a set of accepting states,
6. $P_\delta : \delta \rightarrow (0, 1]$ assigns each transition a probability, and
7. $P_0 : Q_0 \rightarrow (0, 1]$ assigns each start state a probability.

Probabilistic automaton: Example 1

L^1 **of** F_1



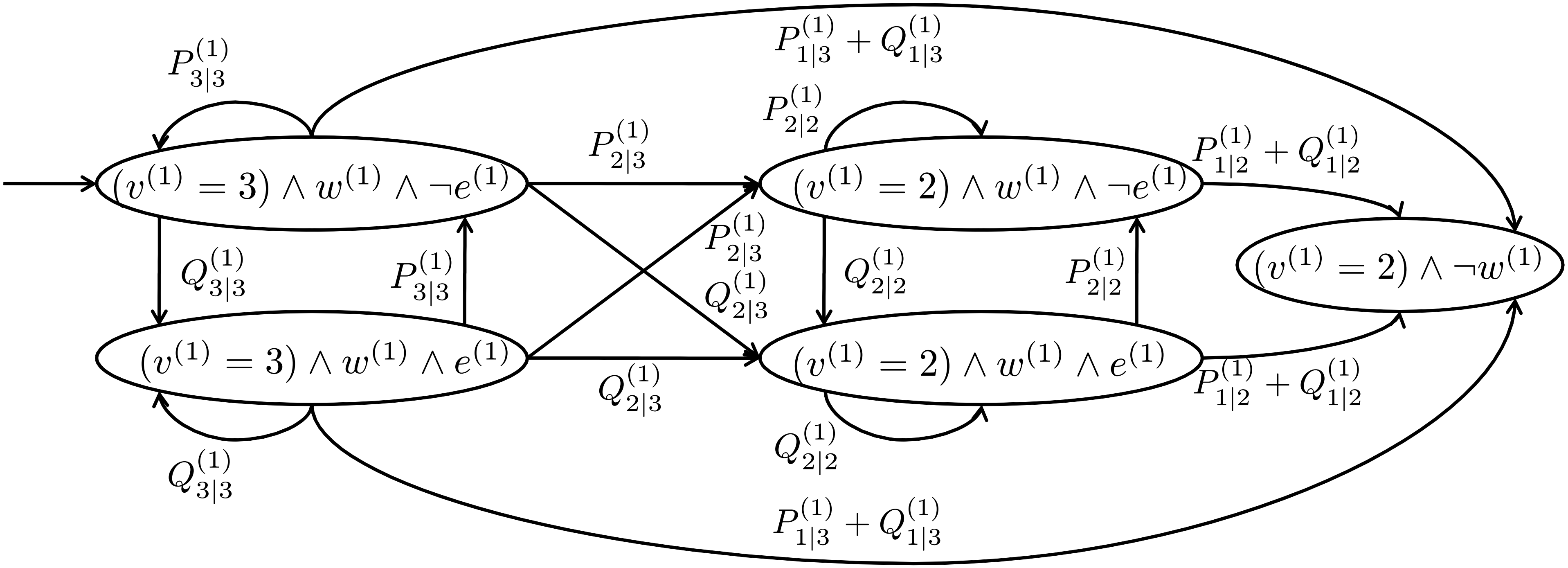

$P^{(1)}_{1|3} + Q^{(1)}_{1|3}$
$P^{(1)}_{3|3}$
$P^{(1)}_{2|2}$
$P^{(1)}_{2|3}$
$P^{(1)}_{1|2} + Q^{(1)}_{1|2}$
$(v^{(1)} = 3) \wedge w^{(1)} \wedge \neg e^{(1)}$
$(v^{(1)} = 2) \wedge w^{(1)} \wedge \neg e^{(1)}$
$P^{(1)}_{2|3}$
$Q^{(1)}_{3|3}$
$P^{(1)}_{3|3}$
$Q^{(1)}_{2|3}$
$Q^{(1)}_{2|2}$
$P^{(1)}_{2|2}$
$\neg w^{(1)}$
$(v^{(1)} = 3) \wedge w^{(1)} \wedge e^{(1)}$
$(v^{(1)} = 2) \wedge w^{(1)} \wedge e^{(1)}$
$Q^{(1)}_{2|3}$
$P^{(1)}_{1|2} + Q^{(1)}_{1|2}$
$Q^{(1)}_{2|2}$
$Q^{(1)}_{3|3}$
$P^{(1)}_{1|3} + Q^{(1)}_{1|3}$

$P_{k|n}$ is the probability that,

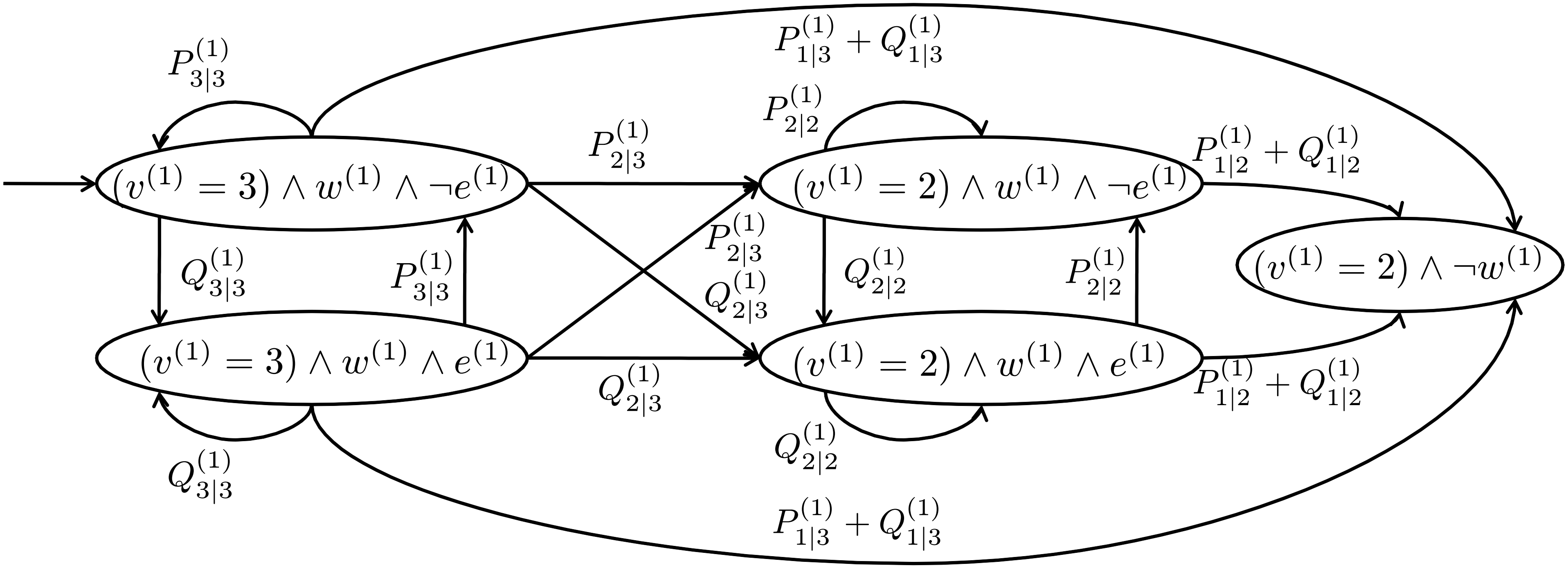
1. The maximal number of n variants producing the same result is k, and;
2. The result is *correct*.

$Q_{k|n}$ is the probability that,

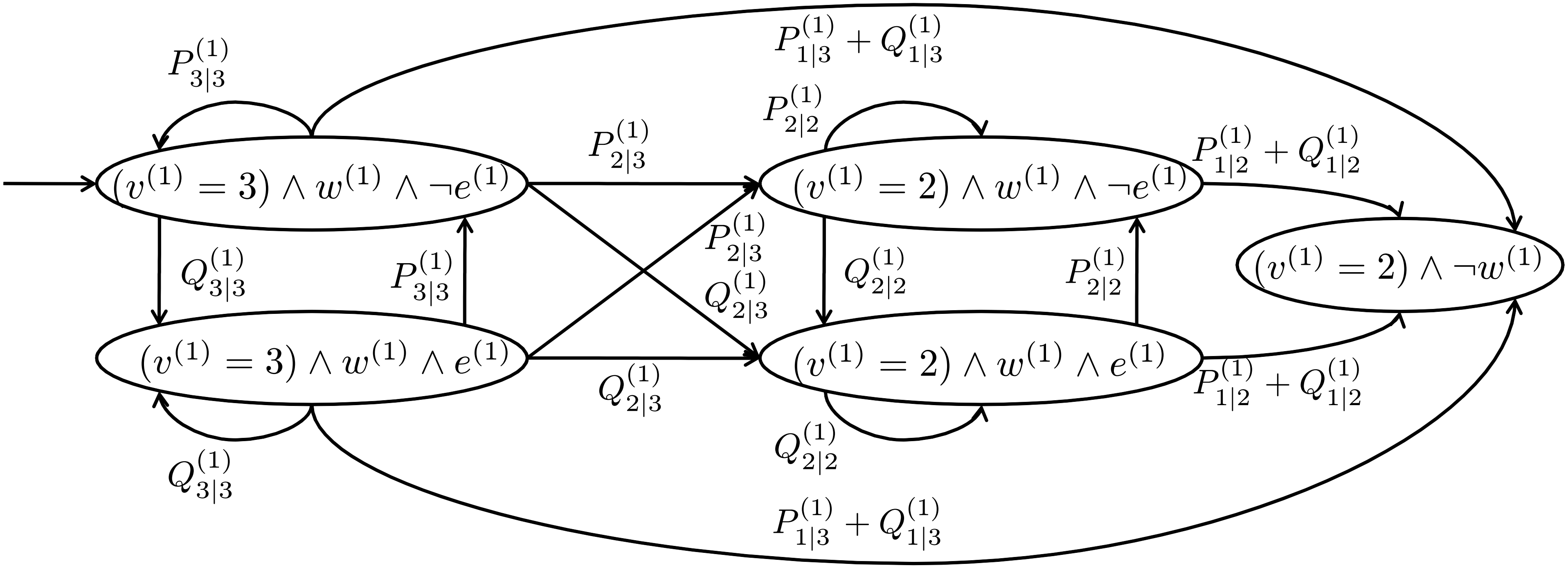
1. The maximal number of n variants producing the same result is k, and;
2. The result is *incorrect*.


$P^{(1)}_{3|3}$
$P^{(1)}_{1|3} + Q^{(1)}_{1|3}$
$P^{(1)}_{2|2}$
$P^{(1)}_{2|3}$
$P^{(1)}_{1|2} + Q^{(1)}_{1|2}$
$(v^{(1)} = 3) \wedge w^{(1)} \wedge \neg e^{(1)}$
$(v^{(1)} = 2) \wedge w^{(1)} \wedge \neg e^{(1)}$
$(v^{(1)} = 2) \wedge \neg w^{(1)}$
$Q^{(1)}_{3|3}$
$P^{(1)}_{3|3}$
$P^{(1)}_{2|3}$
$Q^{(1)}_{2|3}$
$Q^{(1)}_{2|2}$
$P^{(1)}_{2|2}$
$(v^{(1)} = 3) \wedge w^{(1)} \wedge e^{(1)}$
$(v^{(1)} = 2) \wedge w^{(1)} \wedge e^{(1)}$
$Q^{(1)}_{2|3}$
$P^{(1)}_{1|2} + Q^{(1)}_{1|2}$
$Q^{(1)}_{3|3}$
$Q^{(1)}_{2|2}$
$P^{(1)}_{1|3} + Q^{(1)}_{1|3}$

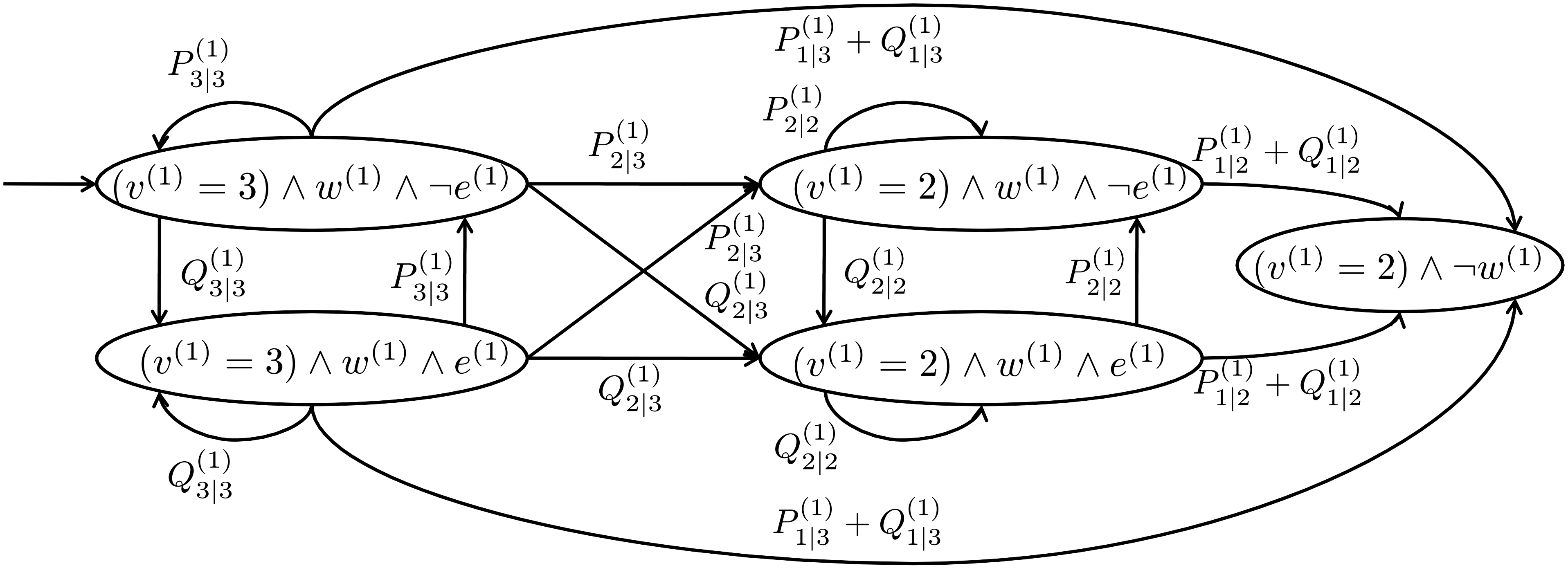
v, the number of working variants. The built-in voting mechanism decides the status of variants by simple majority. For example, if at the start of a clock cycle all 3 variants are working and during the cycle only 2 of 3 variants produce the same result, then the voting mechanism will mark these 2 variants as working, and the other one as *not* working;


$P^{(1)}_{1|3} + Q^{(1)}_{1|3}$
$P^{(1)}_{3|3}$
$P^{(1)}_{2|2}$
$P^{(1)}_{2|3}$
$P^{(1)}_{1|2} + Q^{(1)}_{1|2}$
$(v^{(1)} = 3) \wedge w^{(1)} \wedge \neg e^{(1)}$
$(v^{(1)} = 2) \wedge w^{(1)} \wedge \neg e^{(1)}$
$P^{(1)}_{2|3}$
$(v^{(1)} = 2) \wedge \neg w^{(1)}$
$Q^{(1)}_{3|3}$
$P^{(1)}_{3|3}$
$Q^{(1)}_{2|3}$
$Q^{(1)}_{2|2}$
$P^{(1)}_{2|2}$
$(v^{(1)} = 3) \wedge w^{(1)} \wedge e^{(1)}$
$(v^{(1)} = 2) \wedge w^{(1)} \wedge e^{(1)}$
$Q^{(1)}_{2|3}$
$P^{(1)}_{1|2} + Q^{(1)}_{1|2}$
$Q^{(1)}_{3|3}$
$Q^{(1)}_{2|2}$
$P^{(1)}_{1|3} + Q^{(1)}_{1|3}$

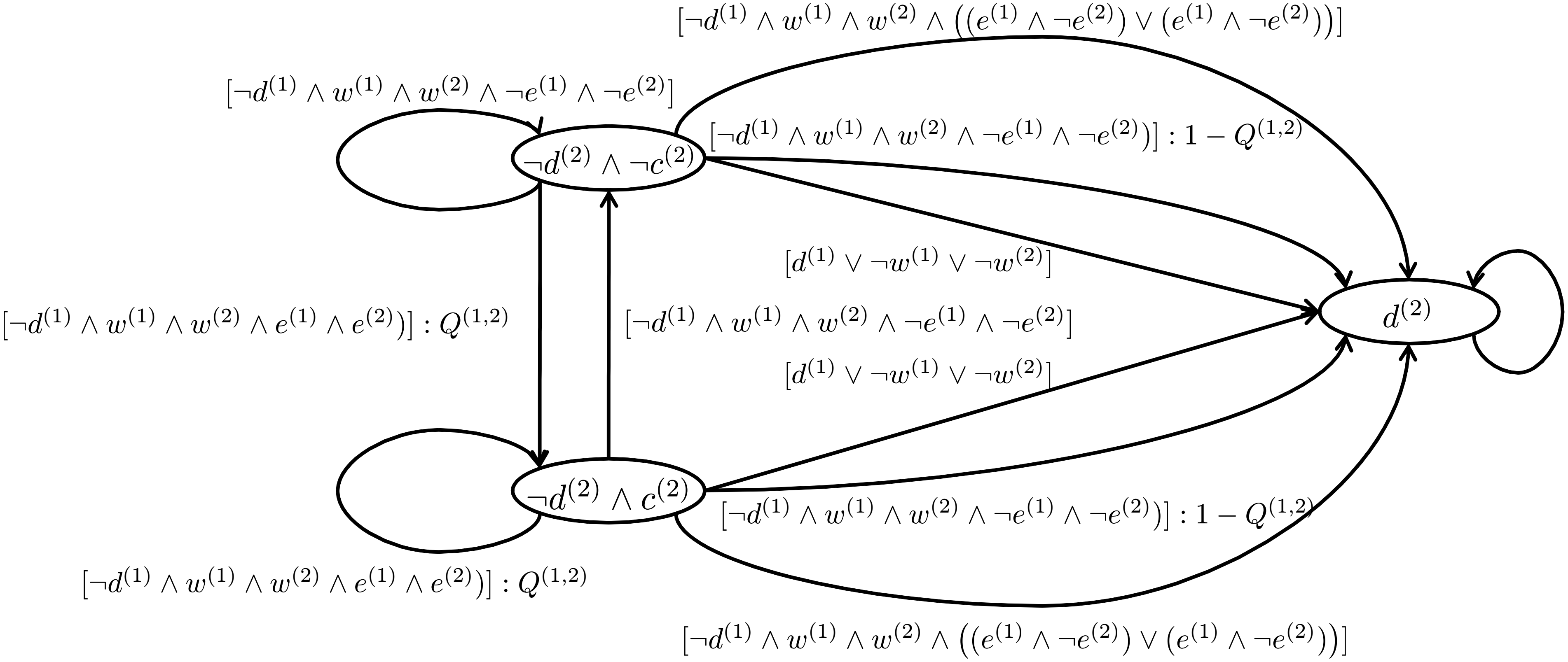
w, the status of a layer. Initially all layers are working. If at one point the voting mechanism cannot decide which variant it can trust, for instance, in case that all 2 working variants report different value, it simply marks the layer as *not* working;


$P^{(1)}_{3|3}$
$P^{(1)}_{1|3} + Q^{(1)}_{1|3}$
$P^{(1)}_{2|2}$
$P^{(1)}_{2|3}$
$P^{(1)}_{1|2} + Q^{(1)}_{1|2}$
$(v^{(1)} = 3) \wedge w^{(1)} \wedge \neg e^{(1)}$
$(v^{(1)} = 2) \wedge w^{(1)} \wedge \neg e^{(1)}$
$P^{(1)}_{2|3}$
$Q^{(1)}_{3|3}$
$P^{(1)}_{3|3}$
$Q^{(1)}_{2|3}$
$Q^{(1)}_{2|2}$
$P^{(1)}_{2|2}$
$(v^{(1)} = 2) \wedge \neg w^{(1)}$
$(v^{(1)} = 3) \wedge w^{(1)} \wedge e^{(1)}$
$(v^{(1)} = 2) \wedge w^{(1)} \wedge e^{(1)}$
$Q^{(1)}_{2|3}$
$P^{(1)}_{1|2} + Q^{(1)}_{1|2}$
$Q^{(1)}_{2|2}$
$Q^{(1)}_{3|3}$
$P^{(1)}_{1|3} + Q^{(1)}_{1|3}$

e, the error flag. $e = true$ indicates that an erroneous output is produced by the layer. This could happen when, for example, all the working variants produce the exactly same erroneous output, although this is a very unlikely scenario especially when we apply N-variant technique. We will discuss this in more details later.


$P_{1|3}^{(1)} + Q_{1|3}^{(1)}$
$P_{3|3}^{(1)}$
$P_{2|2}^{(1)}$
$P_{2|3}^{(1)}$
$P_{1|2}^{(1)} + Q_{1|2}^{(1)}$
$(v^{(1)} = 3) \wedge w^{(1)} \wedge \neg e^{(1)}$
$(v^{(1)} = 2) \wedge w^{(1)} \wedge \neg e^{(1)}$
$P_{2|3}^{(1)}$
$(v^{(1)} = 2) \wedge \neg w^{(1)}$
$Q_{3|3}^{(1)}$
$P_{3|3}^{(1)}$
$Q_{2|3}^{(1)}$
$Q_{2|2}^{(1)}$
$P_{2|2}^{(1)}$
$(v^{(1)} = 3) \wedge w^{(1)} \wedge e^{(1)}$
$(v^{(1)} = 2) \wedge w^{(1)} \wedge e^{(1)}$
$Q_{2|3}^{(1)}$
$P_{1|2}^{(1)} + Q_{1|2}^{(1)}$
$Q_{3|3}^{(1)}$
$Q_{2|2}^{(1)}$
$P_{1|3}^{(1)} + Q_{1|3}^{(1)}$

Monitoring and Reconfiguration Sub-Module in layer 2 (MRSM2)


$[\neg d^{(1)} \wedge w^{(1)} \wedge w^{(2)} \wedge \big((e^{(1)} \wedge \neg e^{(2)}) \vee (e^{(1)} \wedge \neg e^{(2)})\big)]$
$[\neg d^{(1)} \wedge w^{(1)} \wedge w^{(2)} \wedge \neg e^{(1)} \wedge \neg e^{(2)}]$
$[\neg d^{(1)} \wedge w^{(1)} \wedge w^{(2)} \wedge \neg e^{(1)} \wedge \neg e^{(2)})] : 1 - Q^{(1,2)}$
$\neg d^{(2)} \wedge \neg c^{(2)}$
$[d^{(1)} \vee \neg w^{(1)} \vee \neg w^{(2)}]$
$[\neg d^{(1)} \wedge w^{(1)} \wedge w^{(2)} \wedge e^{(1)} \wedge e^{(2)})] : Q^{(1,2)}$
$[\neg d^{(1)} \wedge w^{(1)} \wedge w^{(2)} \wedge \neg e^{(1)} \wedge \neg e^{(2)}]$
$d^{(2)}$
$[d^{(1)} \vee \neg w^{(1)} \vee \neg w^{(2)}]$
$\neg d^{(2)} \wedge c^{(2)}$
$[\neg d^{(1)} \wedge w^{(1)} \wedge w^{(2)} \wedge \neg e^{(1)} \wedge \neg e^{(2)})] : 1 - Q^{(1,2)}$
$[\neg d^{(1)} \wedge w^{(1)} \wedge w^{(2)} \wedge e^{(1)} \wedge e^{(2)})] : Q^{(1,2)}$
$[\neg d^{(1)} \wedge w^{(1)} \wedge w^{(2)} \wedge \big((e^{(1)} \wedge \neg e^{(2)}) \vee (e^{(1)} \wedge \neg e^{(2)})\big)]$

Computational Experiments

- Analysis used:
  - Symbolic Hierarchical Automated Reliability/Performance Evaluator (SHARPE) to analyze GSPNs
  - Probabilistic model checker PRISM to analyze the probabilistic automaton-based model

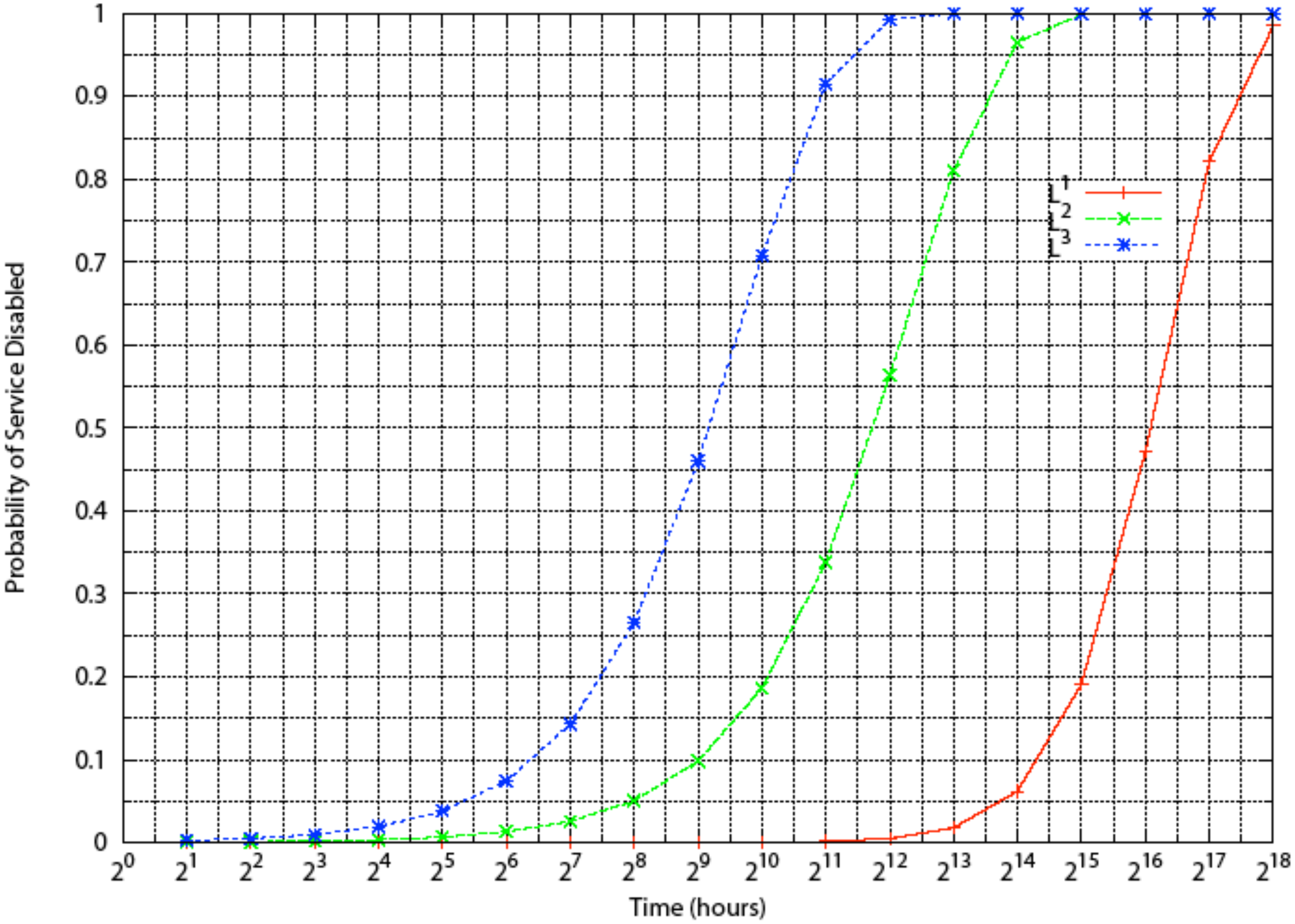


Figure 8. Probability of services being disabled for the GSPN model.

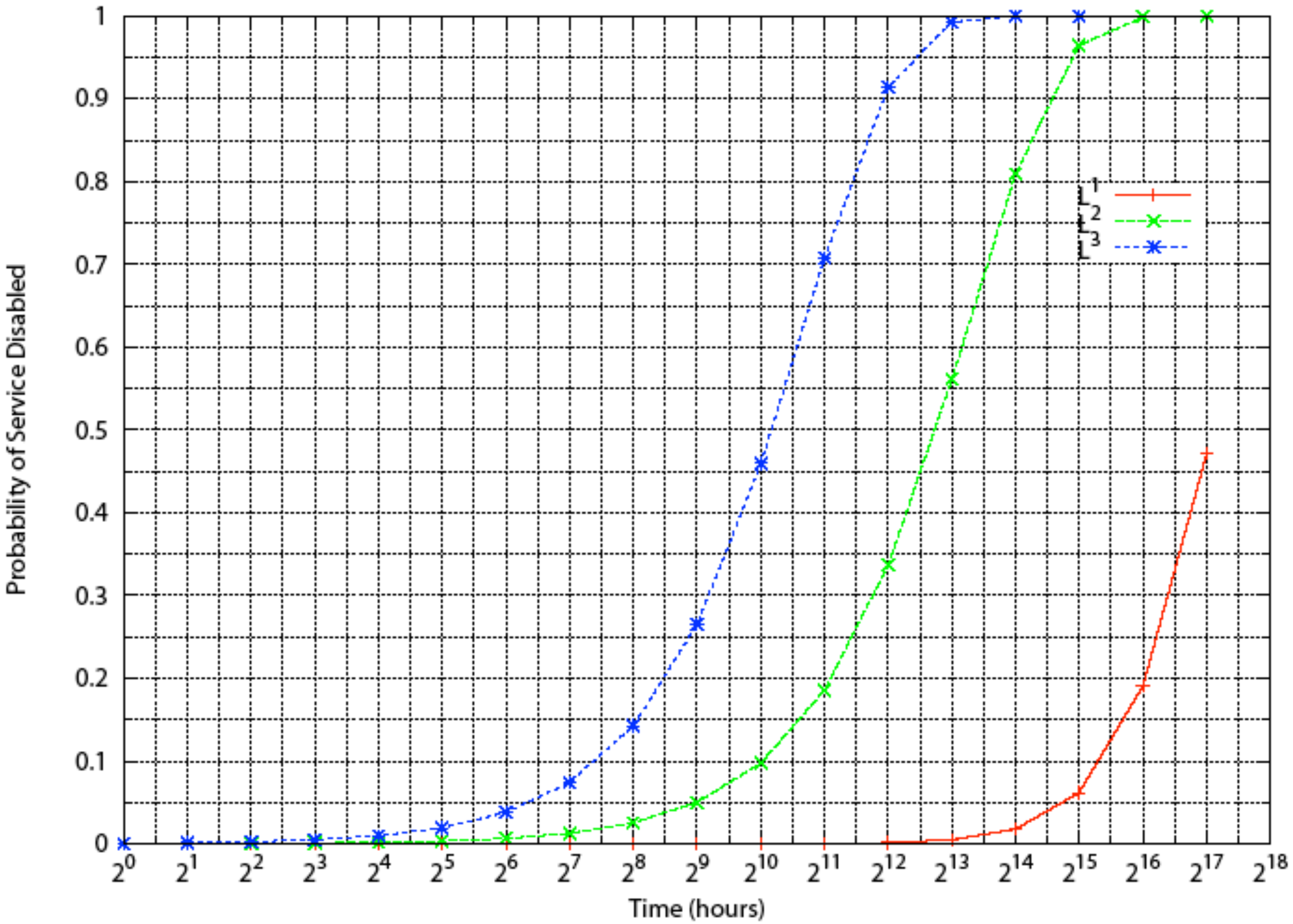
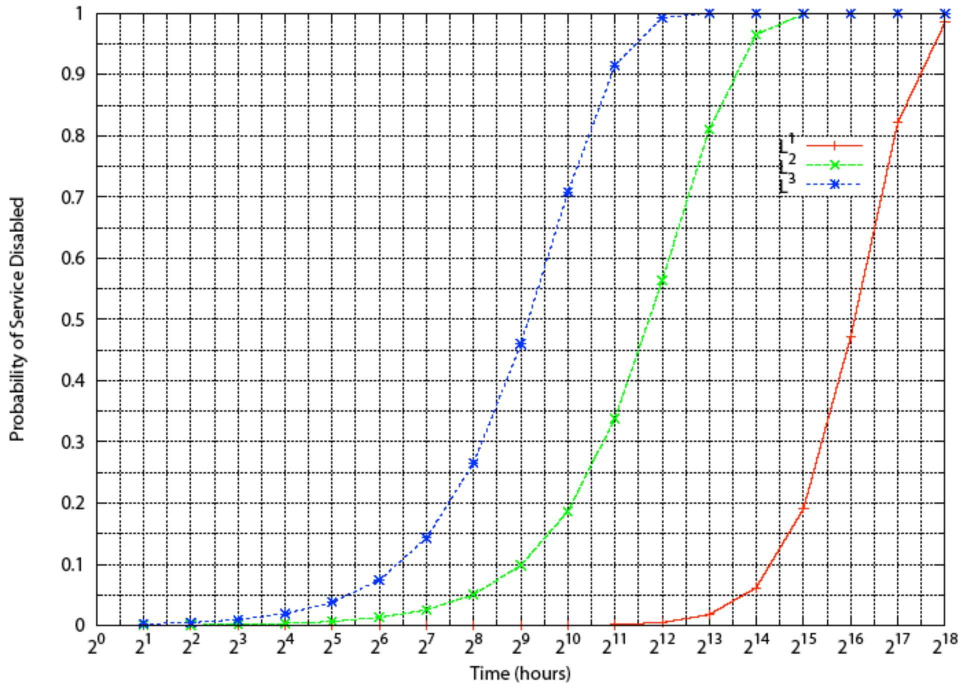
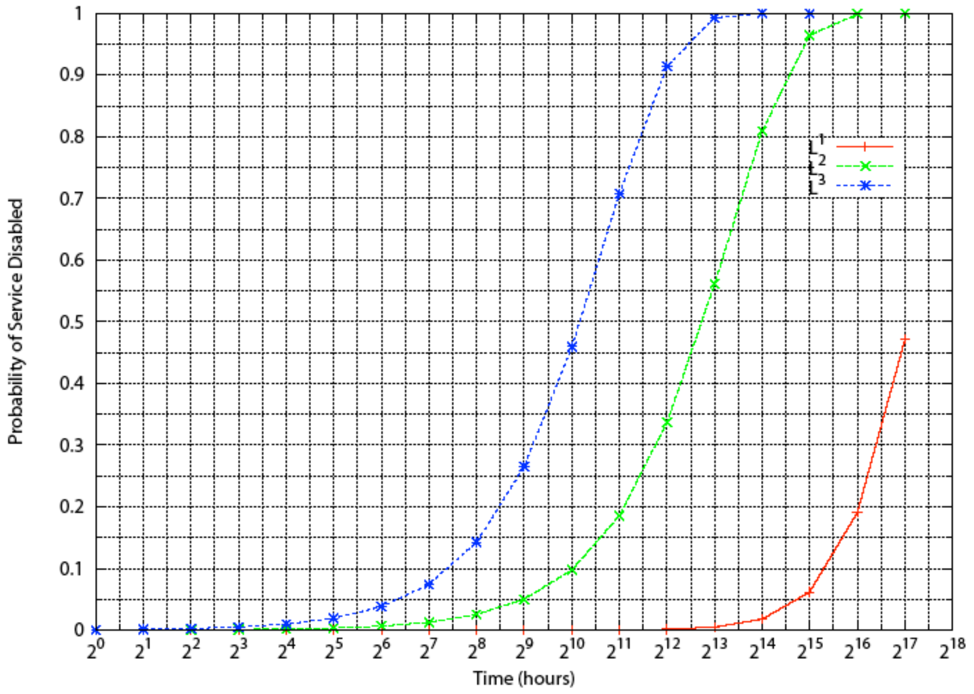


Figure 9. Probability of services being disabled for the probabilistic automaton-based model.



GSPN model



Probability Automaton-based model

Conclusions

- Hierarchical Formal Model was introduced
    - Adaptive Functional Capability Model (AFCM)
    - Multi-layer architecture
    - Adaptation capabilities
    - Reconfiguration capabilities
    - Use Petri Net to deal with design specification experimentation
    - Use model checking to go from design to implementation