Statistical Model Checking of Distributed Adaptive Real-Time Software

David Kyle, Jeffery Hansen, Sagar Chaki

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

Statistical Model Checking (SMC)

Objective:

Estimate the probability that a property holds in a system.

Approach:

Randomized simulation of a model of the system; repeat for many trials, observe if the property holds.

How many trials?

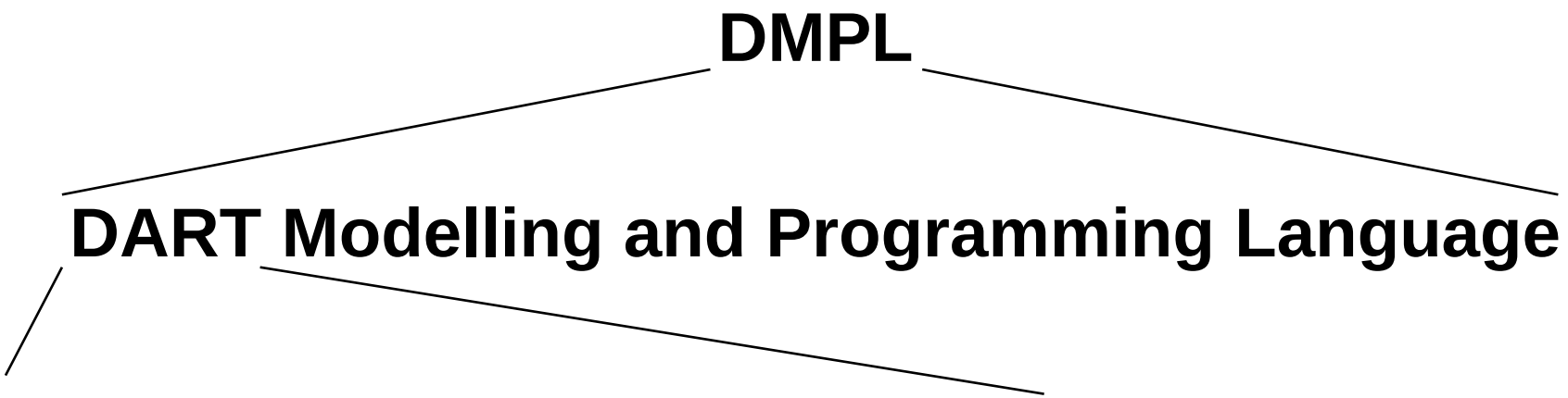
We use a target relative error, calculated as follows:

$$RE(\hat{p}) = \frac{\sqrt{Var(\hat{p})}}{E[\hat{p}]} = \sqrt{\frac{1-p}{pN}} \approx \frac{1}{\sqrt{pN}}$$, for small probabilities

Where p is the true probability that the property holds, $\hat{p}$ is the estimated probability, and N is the number of trials

DMPL

We're developing and using DMPL, a Domain Specific Language:

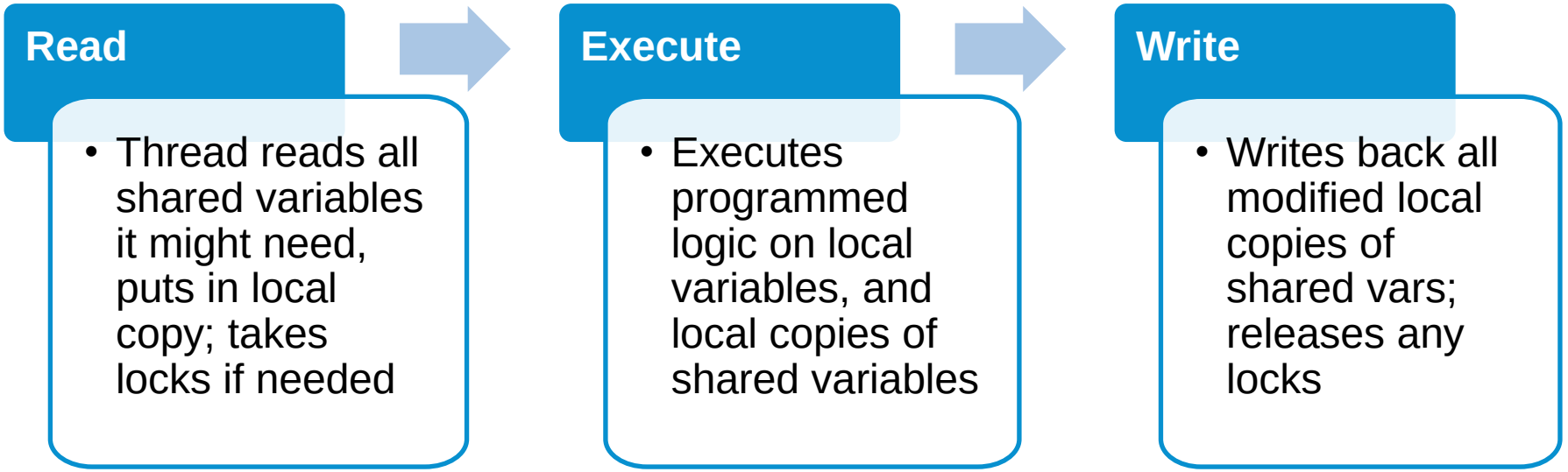


Design Goals:

- Provide features that support creation of DART systems
- Encourage verifiable implementations through language features and restrictions

DMPL Model of Computation

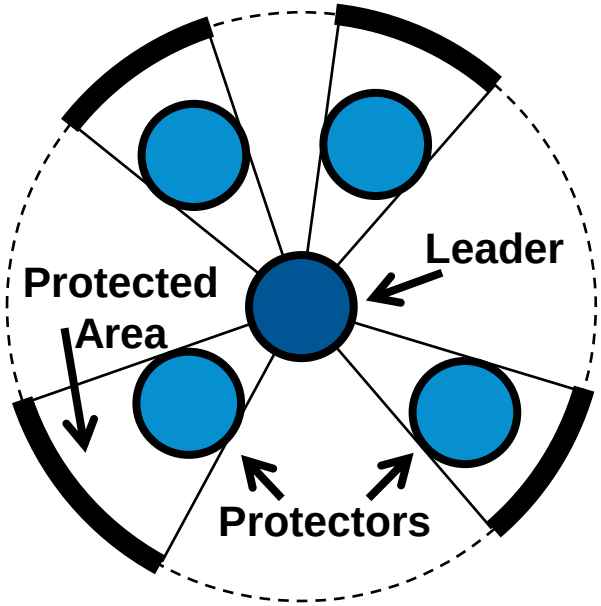
All threads are periodic. Each period, a thread runs as follows:



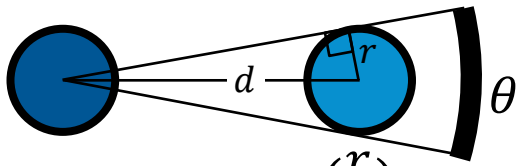
This design aids monitorability. A monitor thread can follow the same pattern, and always get a consistent view of state.

DMPL Example

Total Coverage



Single Protector Coverage

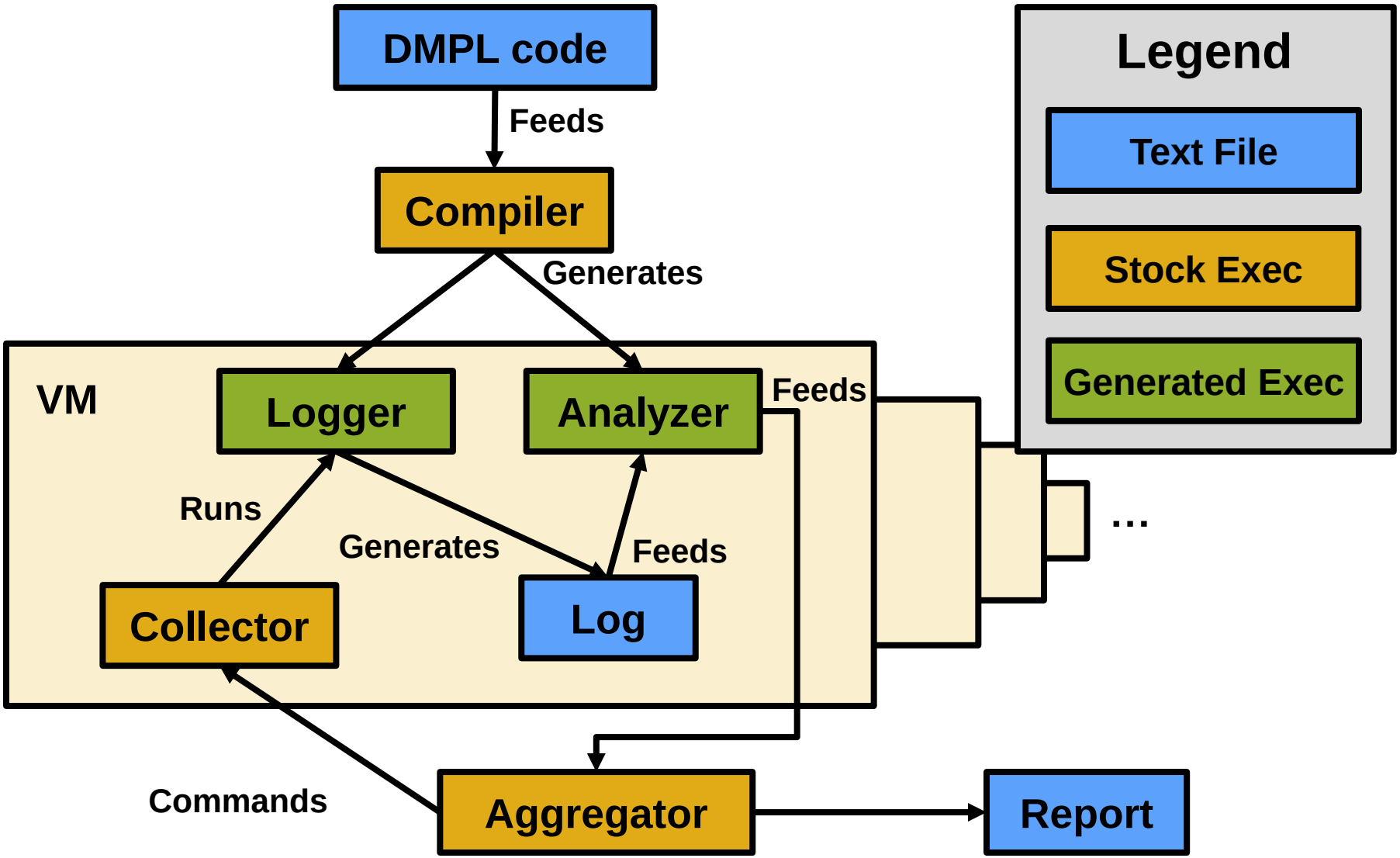


$$\theta = 2\sin^{-1}\left(\frac{r}{d}\right)$$

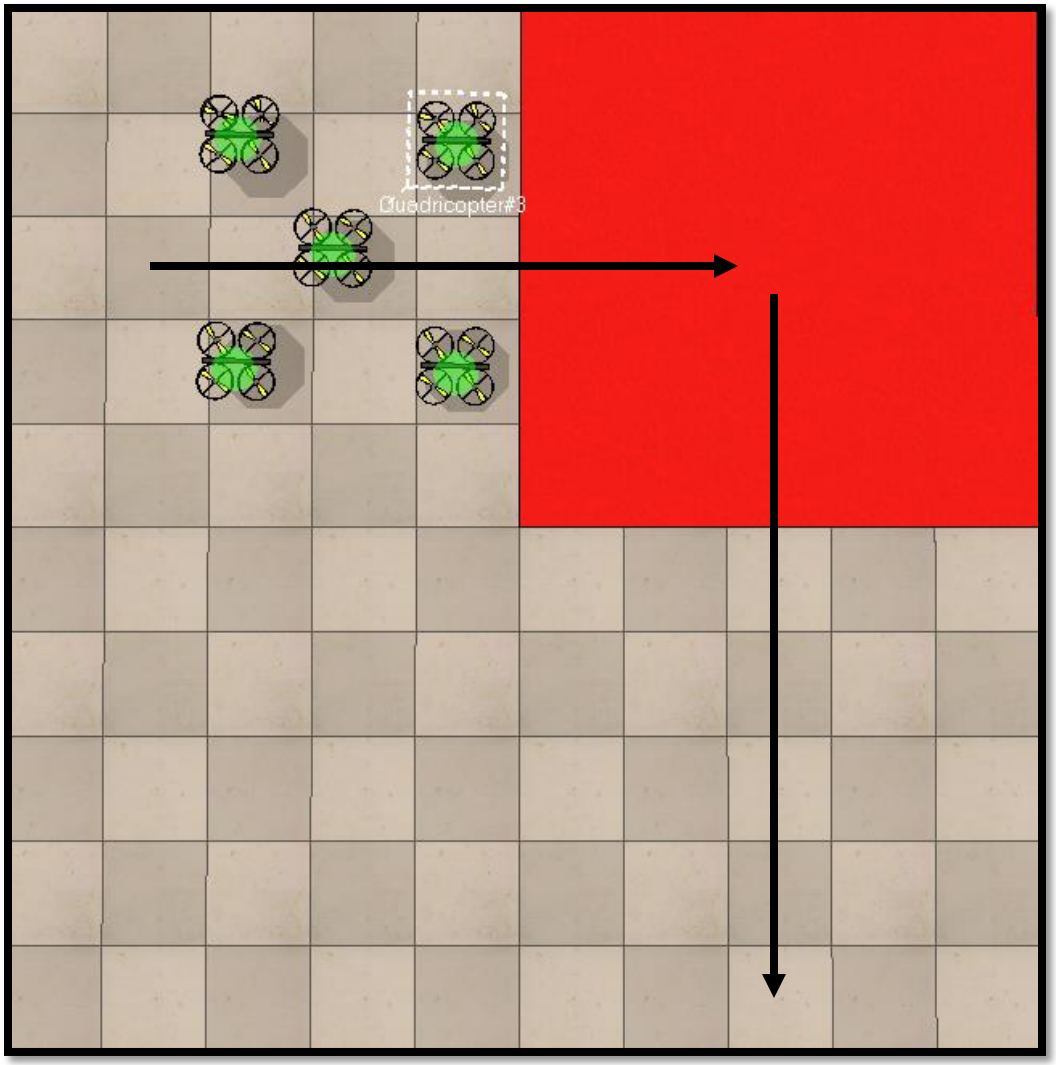
DMPL is generally C-like. One added feature is **expect** clauses, for specification of properties for verification using SMC.

```
pure double coverage()
{
  double cover = 0.0, dist, lat, lng;
  double xd, yd;
  lat = GET_LAT(); lng = GET_LNG();
  forall_other(nid) {
    xd = GET_LAT()@nid – lat;
    yd = GET_LNG()@nid – lng;
    dist = LL2M * sqrt(xd*xd + yd*yd);
    if(dist = 0.0) continue;
    cover = cover + asin(RAD/dist)/M_PI;
  }
  return cover;
}
@AtLeast(0.5) expect(coverage() > 0.9);
@AtEnd expect(reached_home_base());
```

Tool Chain



Exeriment Scenario



Recon mission with 5 drones.

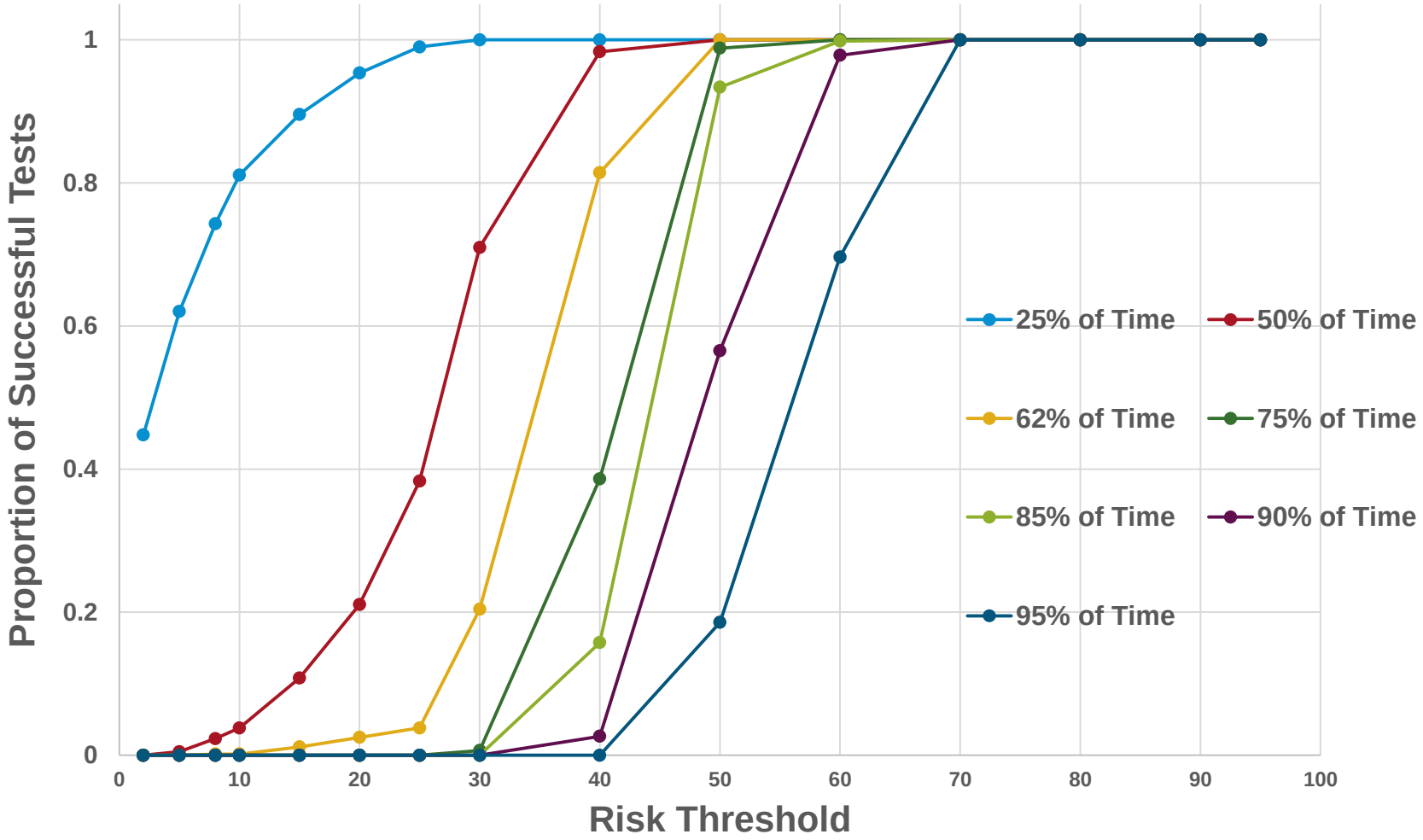
Center drone has mission-critical sensor package. Others defend.

Center has defined path across 2D grid. Others maintain formation.

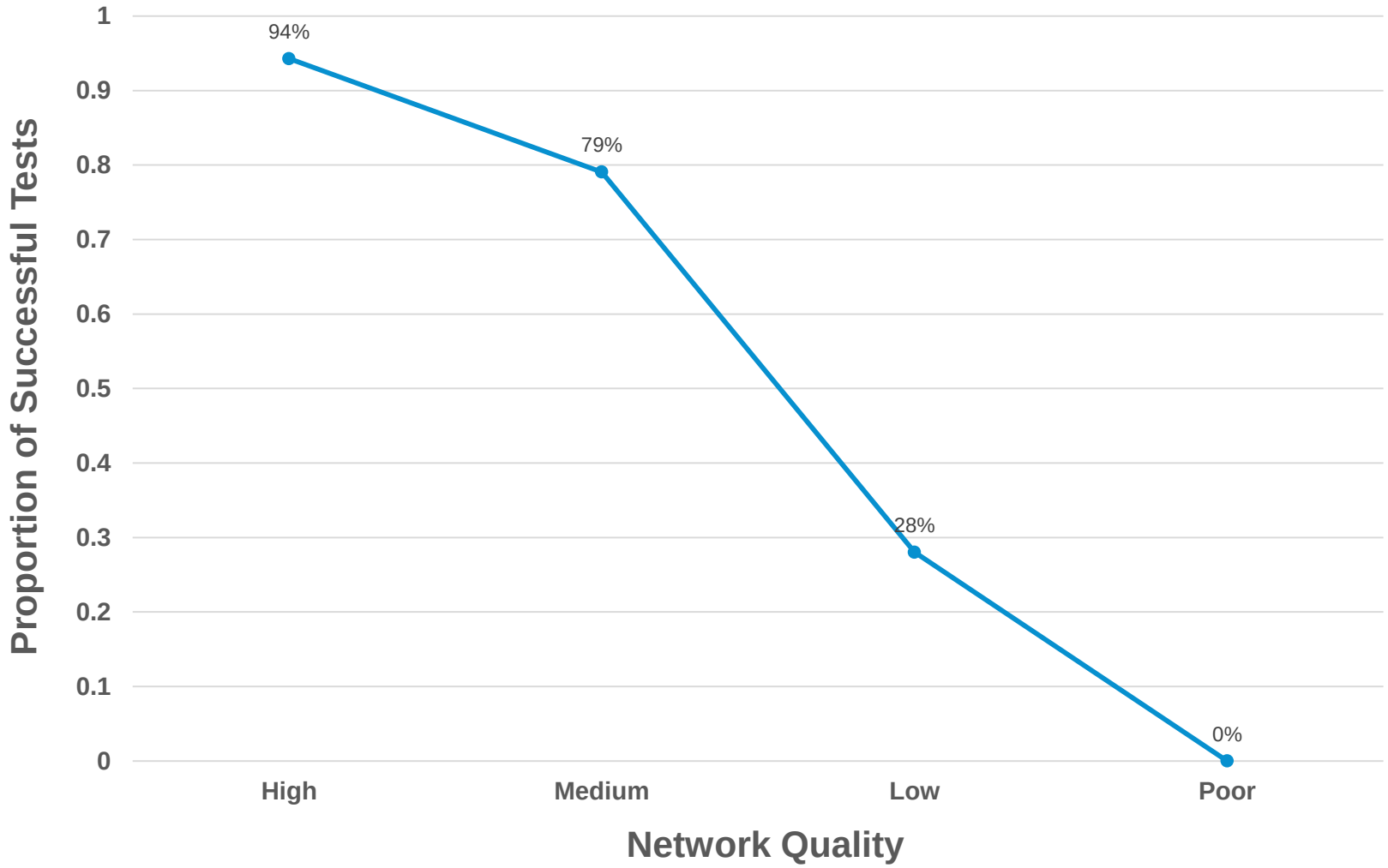
Grid cells have randomly assigned hazards; red cells have higher hazard.

System can opt for close or loose formation. Close is safer from attack, but slower, vs loose.

Experiment 1: Quality of Formation



Experiment 2: Network Disruption



Conclusion

Contributions

- A Domain Specific Language, DMPL, which promotes verification of distributed software written with it
- A toolchain for performing SMC of distributed software, across many compute platforms

Future Work

- Add formal temporal quanitification to DMPL (instead of ad-hoc AtEnd and AtLeast quantifiers)
- Formalize what properties can safely be verified by our approach (due to imperfect synchronization of observations across nodes)

Thank You

Any questions?