Adaptive Middleware for Embedded Systems:

Developing a Formal Model, Language Abstractions and Implementation Techniques

Gul A. Agha

Open Systems Laboratory

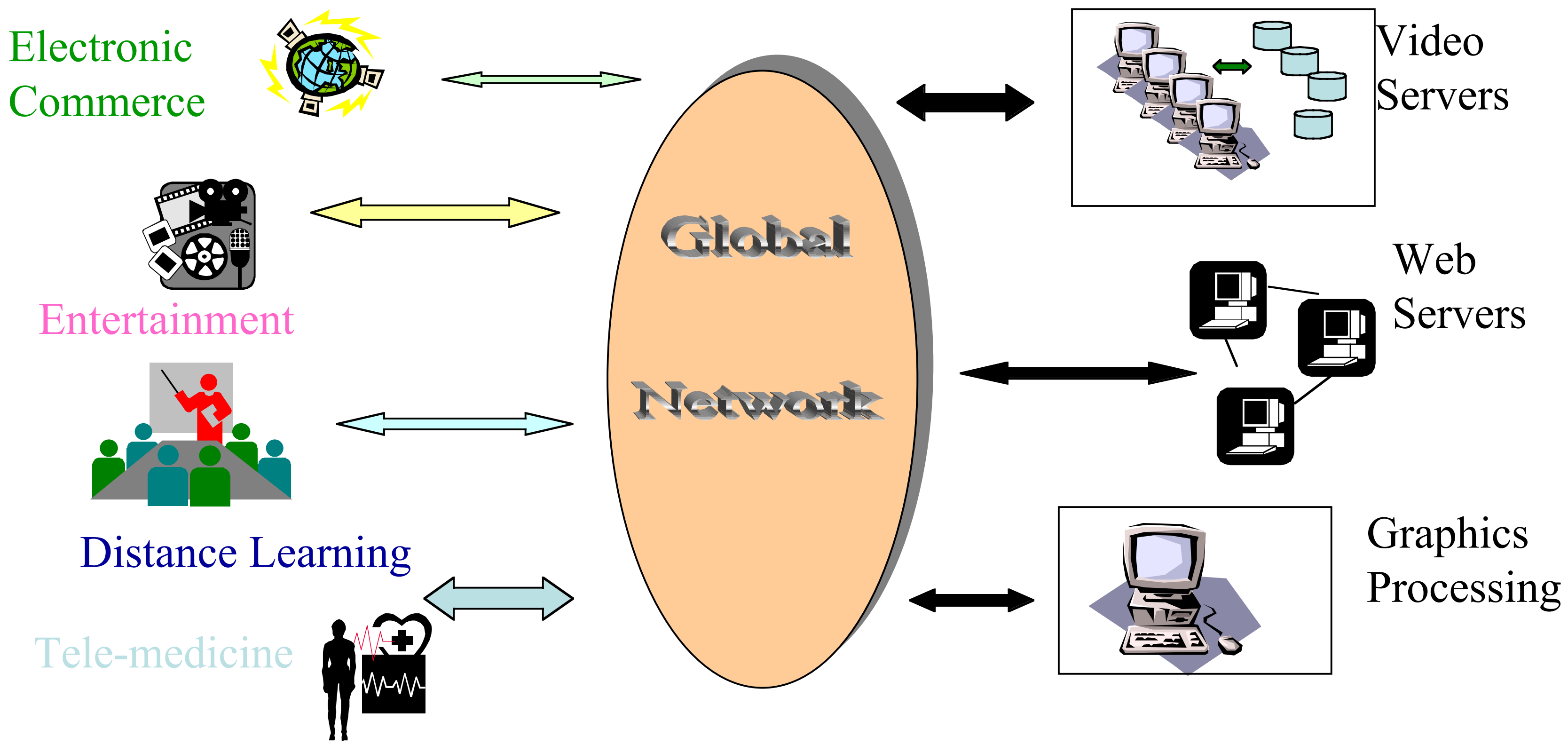
University of Illinois at Urbana-Champaign

Acknowledgements

Joint work with

- Carolyn Talcott, SRI
- Nalini Venkatasubramanian, UC Irvine
- Dan Sturman and Mark Astley, IBM Research
- Svend Frolund, HP Labs
- Other Current and Former Members of OSL
- *Supported by DARPA Nest Program, ONR and National Science Foundation*

Open Distributed Systems (ODS)



Requirements - Availability, Reliability, Quality-of-Service, Security, Adaptability

Outline of Talk

- Context and motivation
- Formal Methods for Distributed Middleware
  - Actor theories and the TLAM
  - Examples
- Network Embedded Systems
  - Modeling Issues
  - Example NEST Middleware Architecture
- Research Directions

From a System Designer or Programmer Point of View

- Would like to design and program at the level of interaction between applications
- Want to specify and program different concerns separately
  - basic functionality
  - security
  - dependability / availability
  - real-time requirements

Problems

- OS provides only low level communication and resource management
- Different languages have different representations and interaction mechanisms
- Coordination of distributed components is complex
- Assuring non-interference -- concurrently executing `independent' services may share
  - resources -- bandwidth, cycles, memory
  - information -- database,   sensors/actuators

Distributed Systems Middleware

- Enables communication across multiple
  - computers
  - programming languages
  - data representations
- Can support QoS requirements
- Provide services for higher-level programming abstractions, e.g.
  - group communication
  - transactions
  - data aggregation

Basic Middleware Services

Middleware services may be built out of basic services:

- Communication:
  - location transparency
  - marshalling/unmarshalling arguments
- Naming / directory
  - locating objects / services
- Life cycle
  - create, activate, stop, delete
  - copy (across machine)
  - persistence (save, restore)
- Scheduling

Middleware needs formal methods support

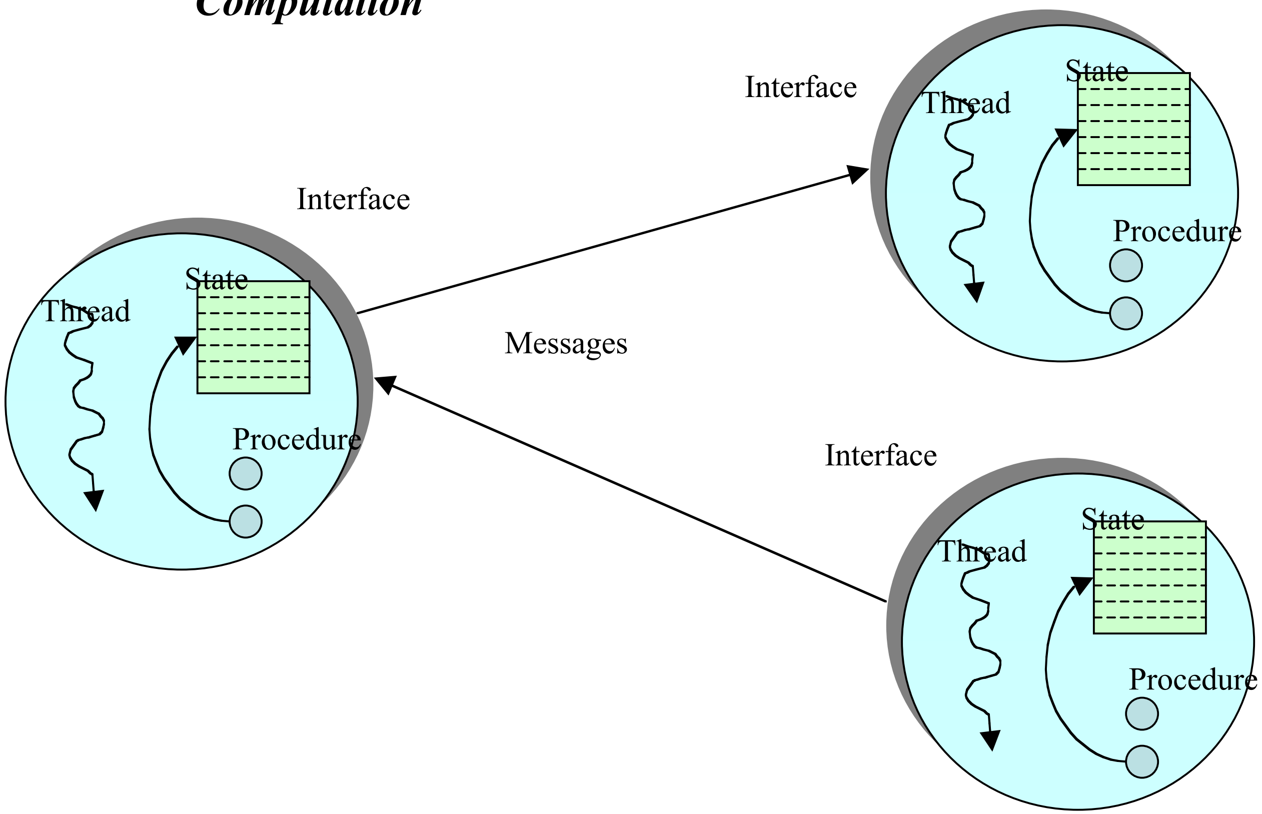
- Agreed upon standards for services and their interfaces (APIs)
- Notion of conformance to standards
- Analysis of standards and service specifications
  - what assumptions do they make for correct operation?
  - what are the potential (positive or negative) interactions?

Two Level Actor Machine (TLAM)

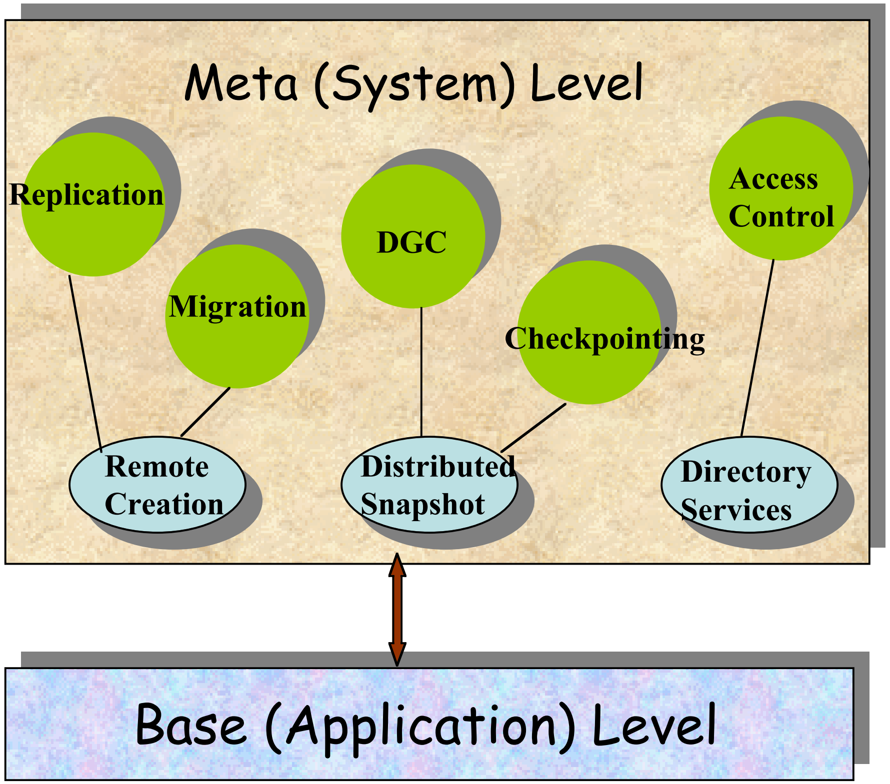
- A semantic framework for specifying and reasoning about middleware services.
- Based on the actor computation model for Open Distributed Systems:
  - base-level actors model application functionality.
  - meta-level actors model middleware services.
- Use of core services to isolate interactions.
- Specification viewpoints

The Actor Model

A Model of Distributed Object Computation

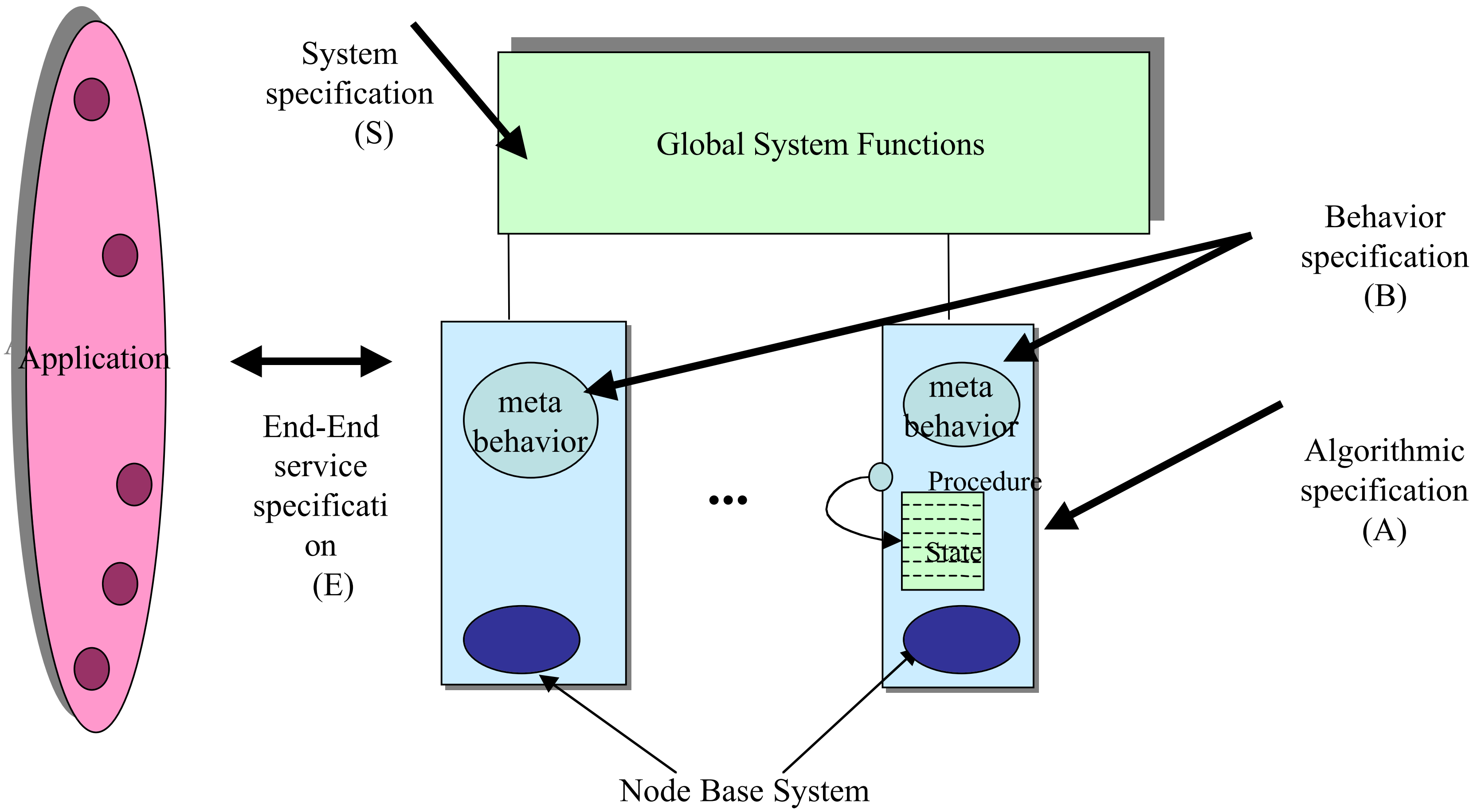


Core Services



Core services allow us to isolate complex interactions
-- *useful for managing composition of services*

Specification Viewpoints



Relating Specification Viewpoints

- (S => E) system spec implies end-to-end service spec
- (B => S if I and NI) behavior spec implies system spec if
  - (I) initial conditions satisfied
  - (NI) non-interference conditions satisfied
- (A=>B) algorithm spec implies behavior spec
- .....

Actor Theories

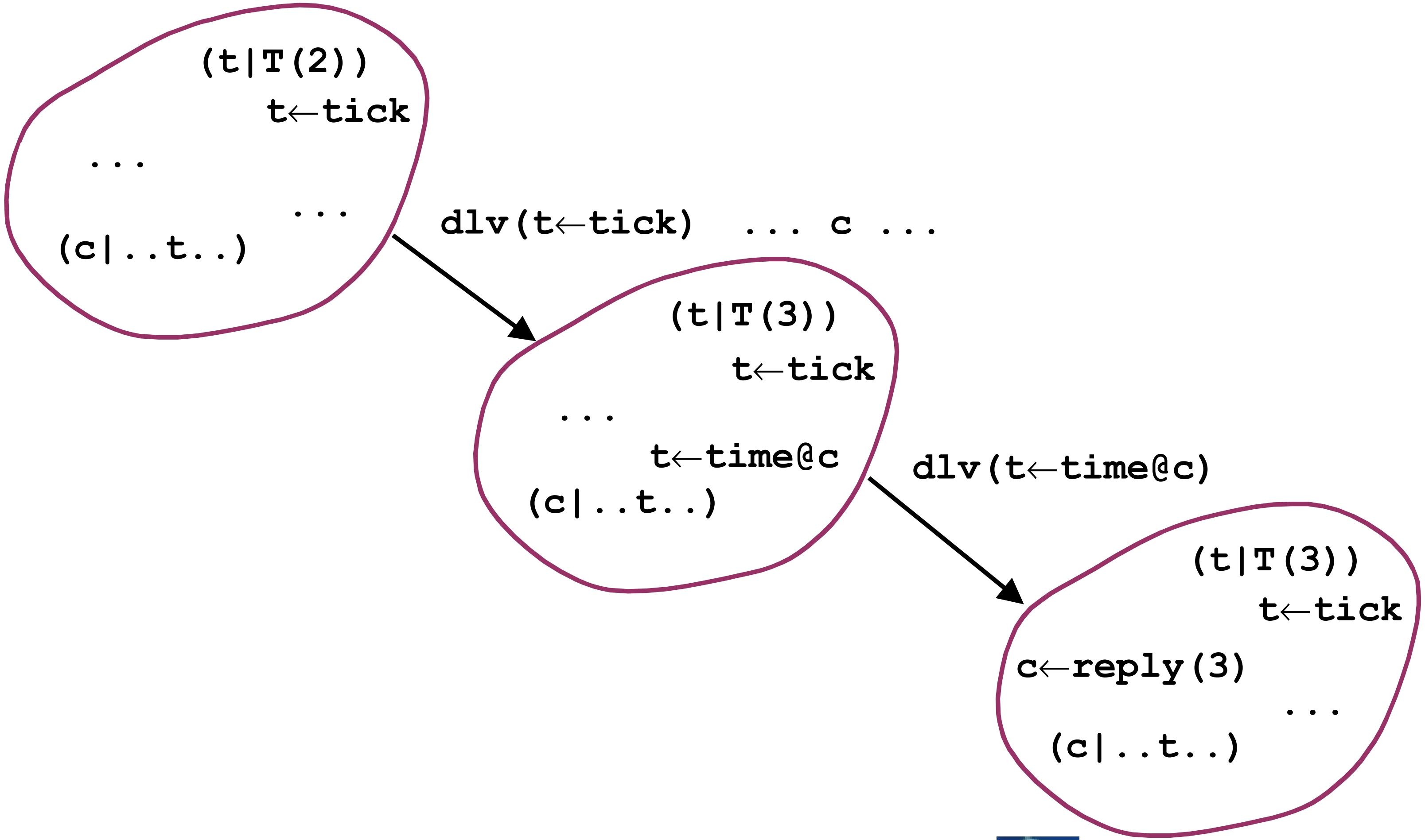
- Actor theories specify:
  - the set of individual actor states
  - the set of messages
  - reaction rules that determine how an actor in a given state may evolve
- An actor system configuration is a `soup' of actors and messages -- a global snapshot from some viewpoint
- An actor system evolves by (concurrent) application of the reaction rules (fairly applied)

Ticker Actor Specification

```
For c,t actor ids, n a number
• States: T(n)
• Messages:  tick, time@c, reply(n)
• Reaction Rules:
    ( t | T(n) ) t ← tick
      ==>
    ( t | T(n+1) ) t ← tick

    ( t | T(n) ) t ← time@c
      ==>
    ( t | T(n) ) c ← reply(n)
```

Ticker Actor Scenario



The Two Level Actor Model (TLAM)

- Stratify actors into
  - Base-level actors (application)
  - Meta-level actors (system level / middleware)
- Base-level actors and messages are augmented with annotations (meta-data)
- Actors and undelivered messages are distributed over a network of nodes and links
- Meta-level actors
  - can examine/modify runtime state and annotations of colocated base-level objects
  - react to local base-level events of interest
  - cooperate with possibly remote meta actors to provide system wide services.

Two Level Actor Theory

- An actor theory extend by
  - annotations for base actor states and messages
  - a set of meta actor states
  - a set of meta-level messages
  - reaction rules for meta-actor
    - parameterized by local base-level configuration
  - event handling rules that determine how a meta-actor reacts to base level events (changes due to base-level reactions or to meta-level modifications)

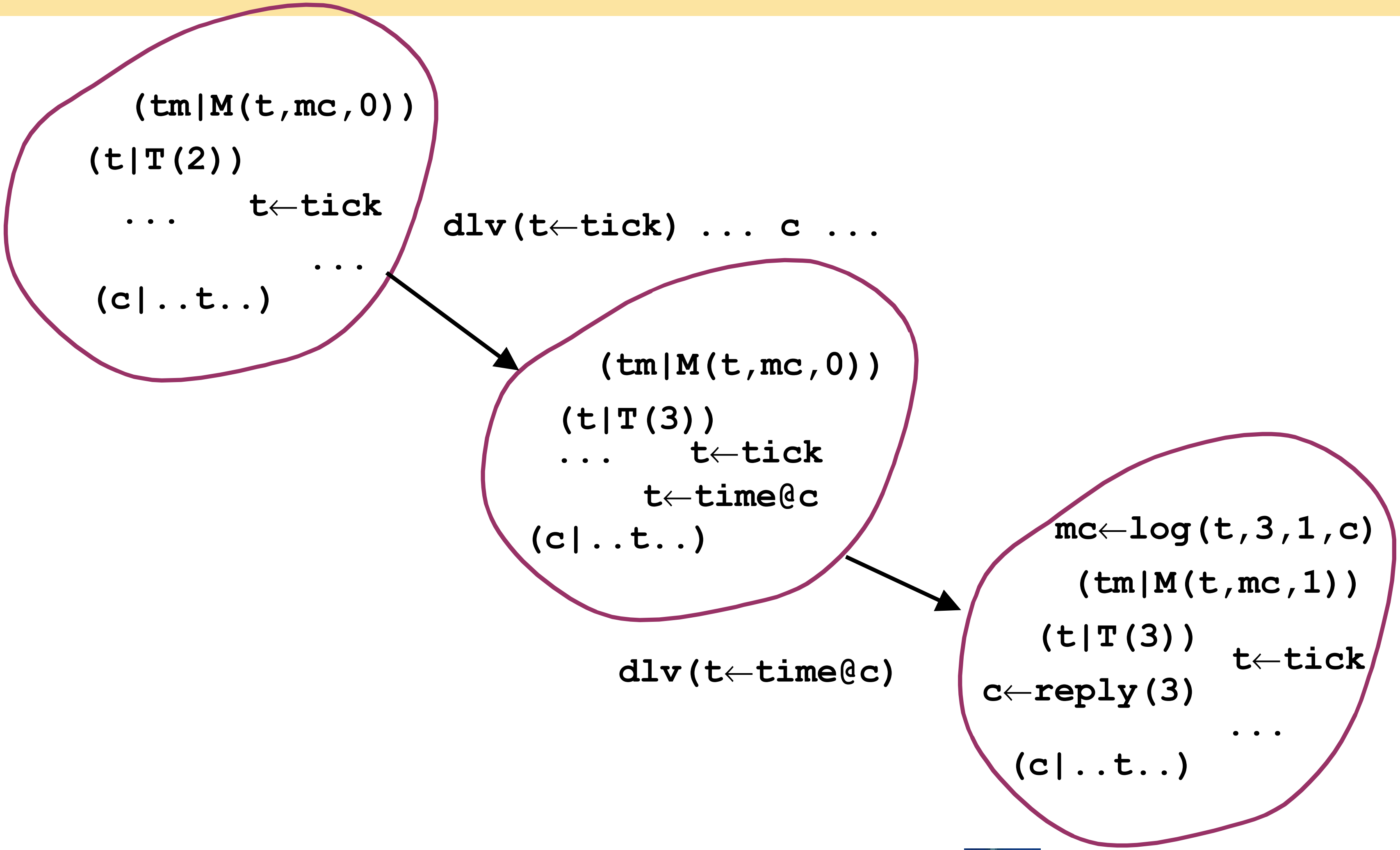
Ticker Monitor Specification

- States: M(t,mc,m)
- Messages: log(t,n,m,c), reset, reset-ack
  - t,mc,c are actor ids, n,m are numbers
- Reaction Rules:

```
(tm | M(t,mc,m))
  ={dlv((t|T(n))t←time@c)/ }=>
(tm | M(t,mc,m+1)) mc ←log(t,n,m+1,c)

(tm | M(t,mc,m)) tm ←reset
  ={ /t:=T(0)}=>
(tm | M(t,mc,0)) mc ← reset-ack
```

Monitored Ticker Scenario



Log Service Example

A logging service

- *Logs* messages delivered to a given set of base actors, and
- When requested *reports* the messages logged since the previous request.

Logging Non-interference Requirement

- A system S satisfies the logging non-interference requirement if:
  - non-logging meta actors do not set Log attributes
  - the only messages sent to logging meta actors by non-logging meta actors are log request messages addressed to the log server

Logging Theorems

- Theorem 1 (*Base-meta noninterference*)
  - If system S has Logging Behavior, then Log meta-actors of S preserve base-level behavior.
- Theorem 2 (*Behavior implies service*)
  - If system S has Logging Behavior and satisfies the logging initial conditions and non-interference requirements, then S provides logging service.

Other Case Studies using TLAM

- QoS based Multimedia (MM) Server
  - Serves requests for presentation of MM object with specified QoS (latency, jitter, frame-rate ...)
  - End-end spec:
    - every request is either served with the required QoS, or
    - explicitly denied if QoS requirements can not be met

Network Embedded Systems

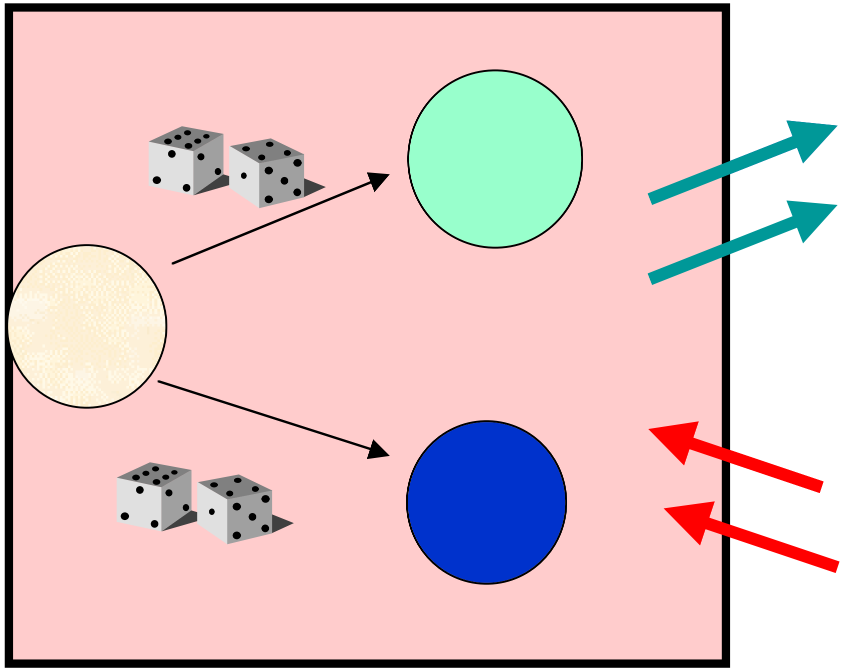
- *From web of computers to web of everything!*
- Paradigm shift from distributed to network embedded systems
  - Large-Scale
  - Real-time sensors and actuators
  - Integration of Discrete and Continuous processes

Modeling Issues for NEST

- Large scale network embedded systems exhibit behaviors that need stochastic analysis.
  - Unpredictable node failures, random communication delay, emerging properties in work load.
  - Incomplete knowledge and uncertainty lead to probabilistic approximation.

Develop a Probabilistic Variant of Real-time Concurrency Semantics

- Probabilities on transitions.
- Summations over execution paths for statistical metrics.
- Quantify approximation and timeliness.

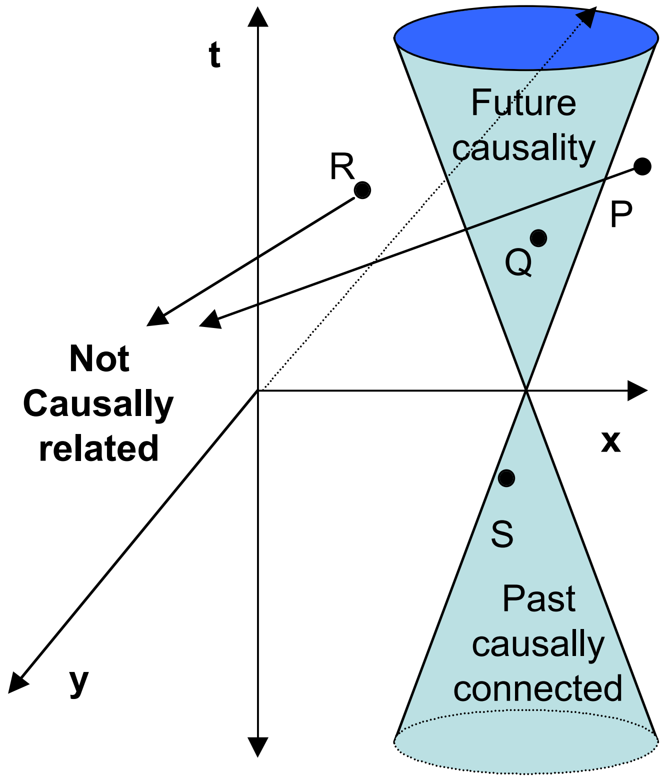


Distributed Model of Time

- Global synchronous wall clock
  - Synchronization is too tight
  - Too detailed an execution model
- Asynchronous, distributed time
  - Vector clocks are too expensive
  - Application behavior is complicated

Need a more expressive model of time:

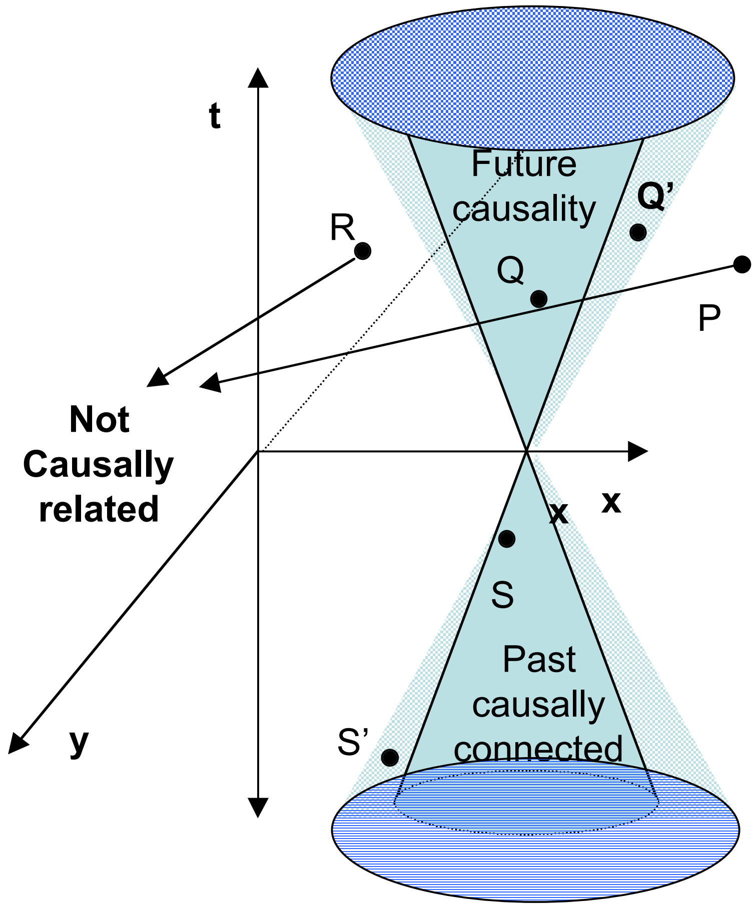
- Notion of distance and distribution.
- Space-Time cone of causal influence.



Light Cones

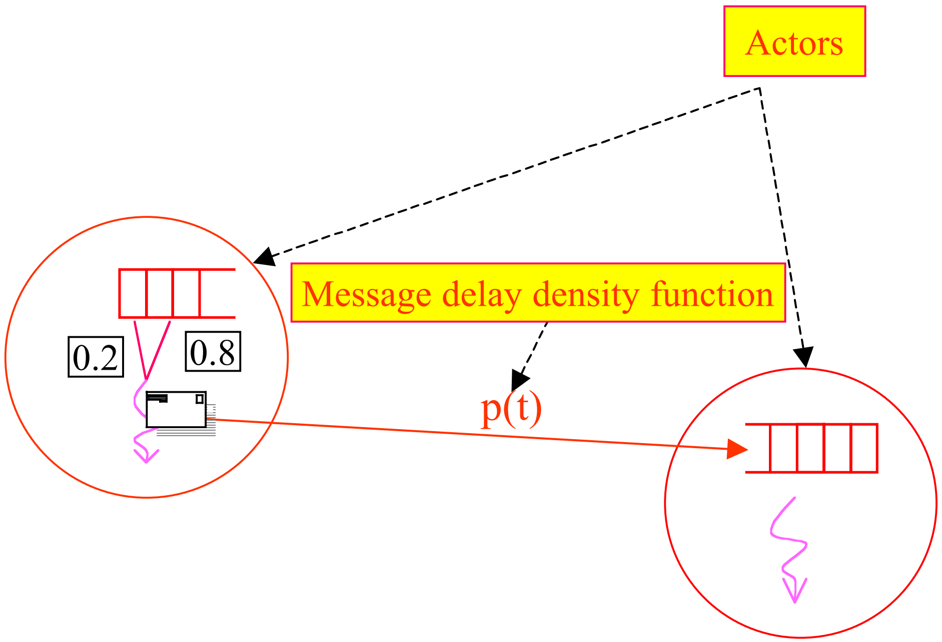
Distributed Time and Probability

- Events separated in space are separated in time:
  - Scheduling delays
  - Latency and communication delays
- Such delays are probabilistic in nature
- Probabilistic cone



Probabilities in Actor Semantics

- Non-determinism in message order replaced by probability distribution
- Total asynchrony replaced by probabilistic delay



Probabilistic Rewrite Theory

- Rewrite theories are abstract (*economic* specifications).
- Rewrite theories can be efficiently implemented (in *Maude*).
- Probabilistic rewrite theory can be used to formally reason about large-scale network embedded systems.
- Time skews subsumed by probabilities.

Probabilistic Rewrite Theory

$$\mathcal{R} = (\Sigma, E, L, R, \rho)$$

- Σ is a signature (sorts and operation declarations)
- E is a set of equations
- L is set of labels (of rewrite rules)
- R is set of rewrite rules
- ρ is a rate function: ρ maps a rule of the form $l : t \rightarrow t'_i$ if C_i to a positive real r

Probabilistic Rewrite Theory (contd.)

For each label $l \in L$ and its associated rule, there are probabilistic rewrite rules:

$$l : t \rightarrow t'_1 \text{ if } C_1 \text{ [rate } r_1(X)]$$

……

$$l : t \rightarrow t'_n \text{ if } C_n \text{ [rate } r_n(X)]$$

where C_i is a conjunction of equation and membership predicates.

Let $T_{\Sigma/E}$ be the ground terms in the initial algebra of a probabilistic rewrite theory. Then

$$\rho : R \rightarrow T_{\Sigma/E}(X)_{\text{PosReal}}$$

where

- ***X is* the set of all free variables in** $t, t'_1, \ldots, t'_n$
- **PosReal is the sort of positive real numbers**

Example

Modeling node failures

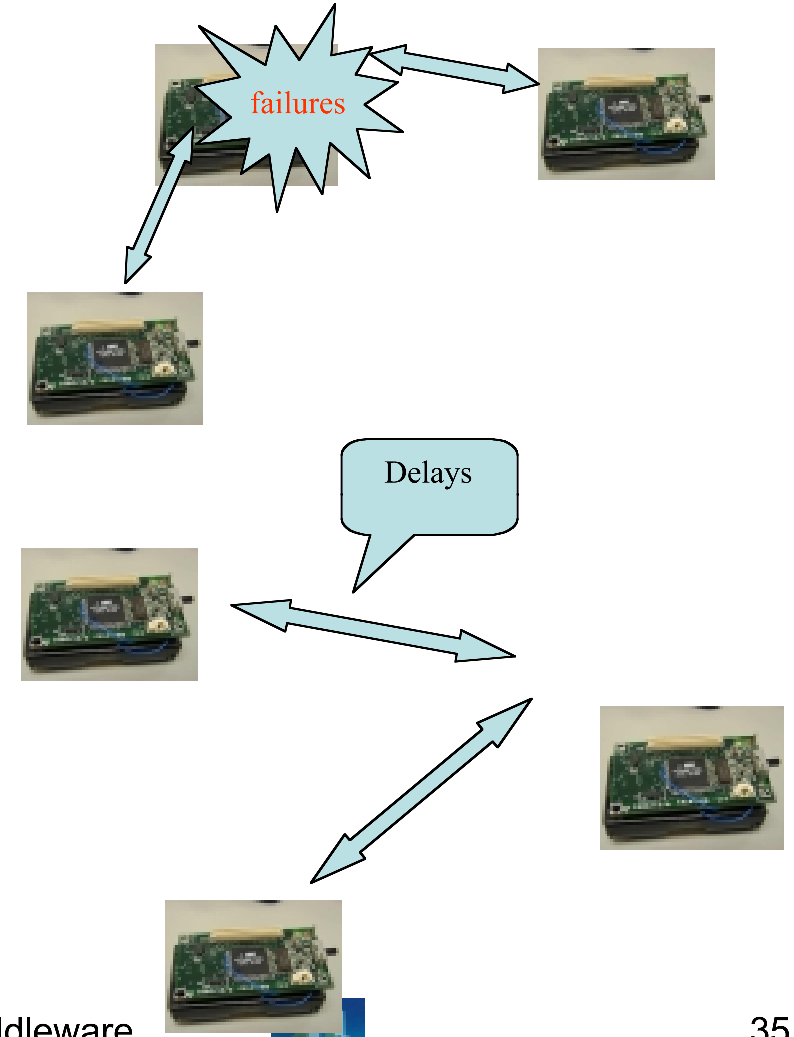
[crl] mote ⇒ **fail** if cond(x,y) [metadata "p(z)"]
[crl] mote ⇒ doAction [metadata "1-p(z)"]

Modeling randomized algorithms

[crl] State ⇒ S(A) if cond (x) [metadata "p(y)"]
[crl] State ⇒ S(B) if cond (z) [metadata "0.2"]

Modeling communication delays

[rl] m<o:recv|time:t> ⇒
<o:recv|time:t+x>[metadata "p(x)"]



Building Network Embedded Systems

- An exact solution is not always necessary
  - Particularly in sensor network applications
- An exact solution is not always of our best interest.
  - Late messages are often useless.
  - They may be even adverse.
- Get a rough estimate first, then refine the answer.
- *The quality of approximation increases with time.*

Global Function Evaluation

*Evaluate a function which is dependent both on the **state** of a node in the network and **time.***

Issues

- Scalability

  e.g., 10^5 nodes ⇒ (at least) ~10^5 messages ⇒ congestion
- Timeliness in response and other real-time constraints
  - unpredictable propagation delays
- Reliability/dependability
  - unreliable communication channels

Approach

- *Use Approximation!*

Observations on Approximate GFE

- Unless an exact answer is required, reliable communication protocols too expensive.
- Early estimates alleviate some real-time concerns.
- Scalability issues:
  - *Prolong data aggregation* phase to alleviate congestion.
  - *Zoom in* on interesting portions of the network.
- Results can be analyzed using a probabilistic model.

Approximate GFE

- $F(S,t)$ – function of interest

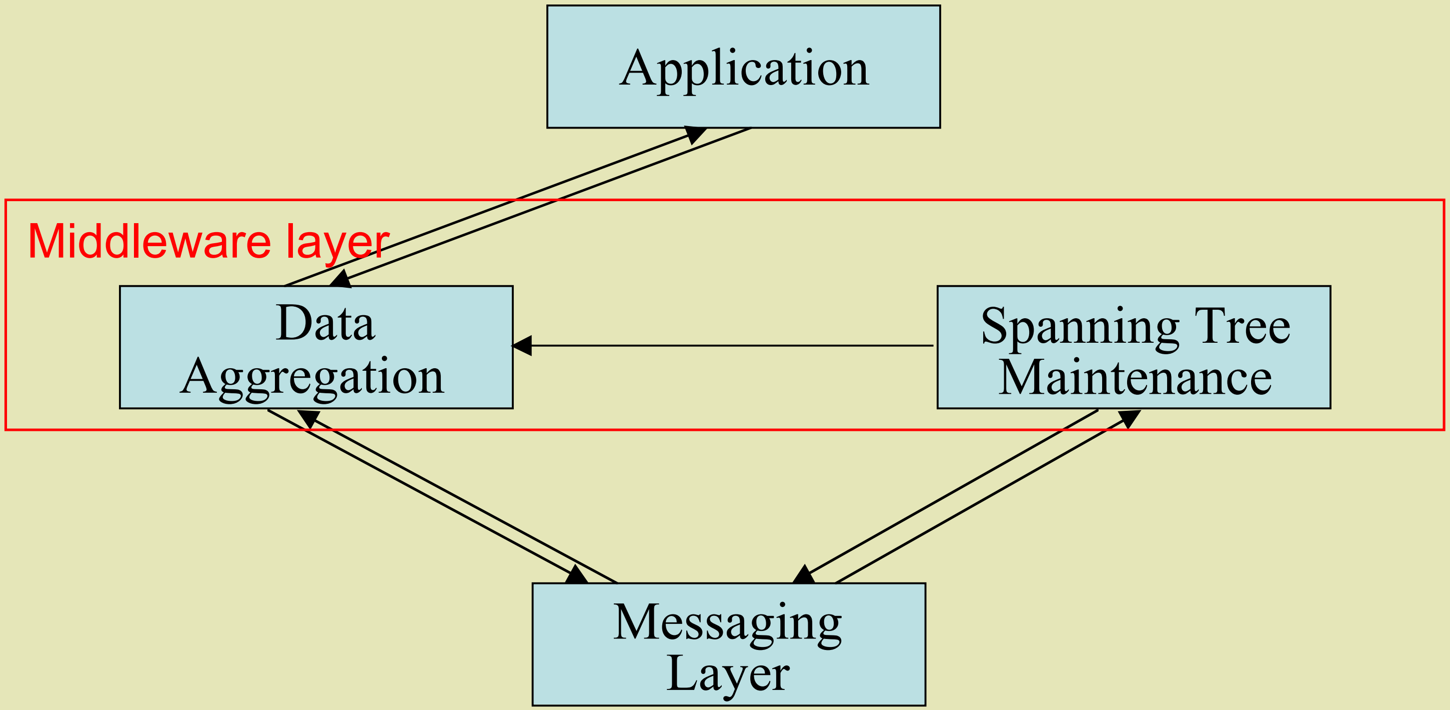
  S = global state t = time
- $A(\overline{x},t)$ – quality of approximation

  $\overline{x}$ = network conditions, # of nodes, etc.
- Some approximation techniques are independent of F.

Example GFE: *Locating a Mobile Target*

- Discover and extrapolate the path of an evader moving (linearly) through a sensor grid.
- *F(S,t): Ax + By = 0* with global state *S* and time *t*.
- The quality of approximation depends on
  - The number of sensor readings
  - The accuracy/consistency of sensor readings

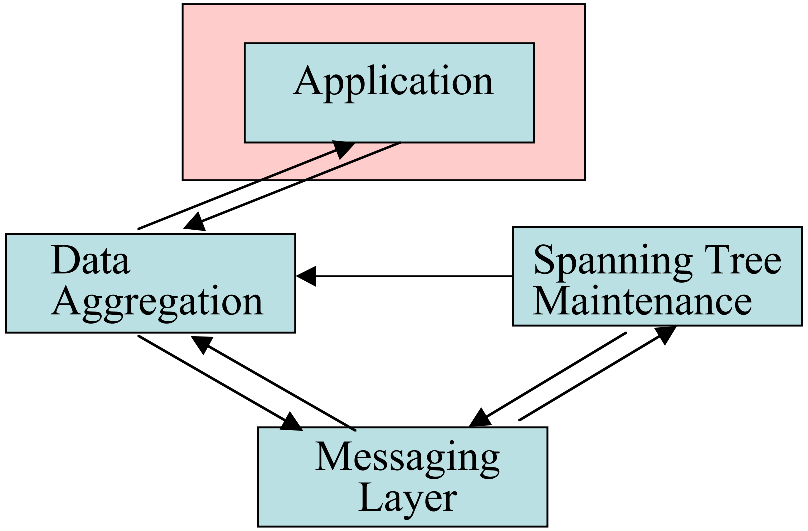
Prototype GFE Node Architecture



Application module

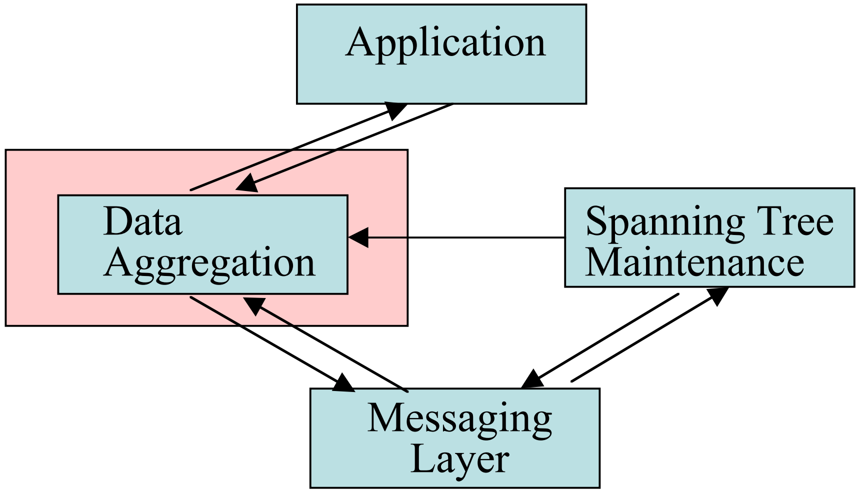
- Implements application dependent functionality:
  - Sensor reading
  - Data processing

- Customizes data aggregation functionality



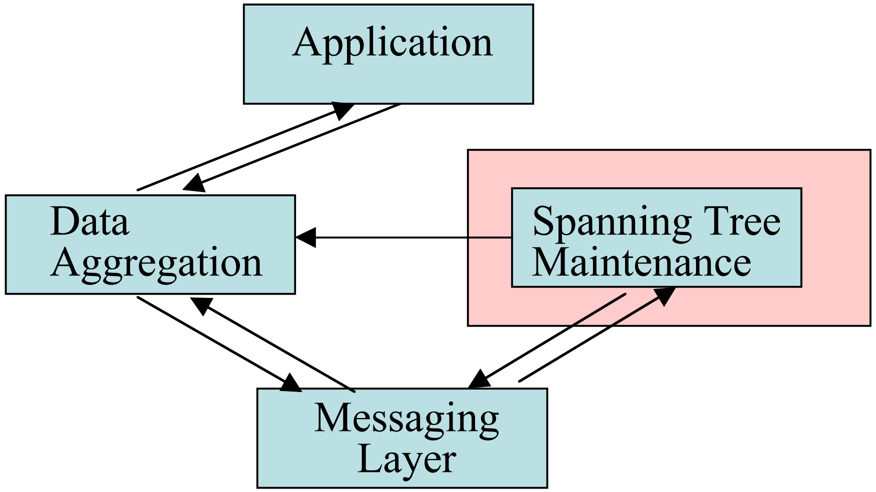
Aggregation Module

- Provides services for:
  - Storing a message
  - Application-assisted message aggregation
  - Rate-controlled message transmission
    - Alleviates congestion
    - Enables reduced power consumption
- Enforces stabilization policies:
  - Control the age of messages accepted
  - Control local congestion

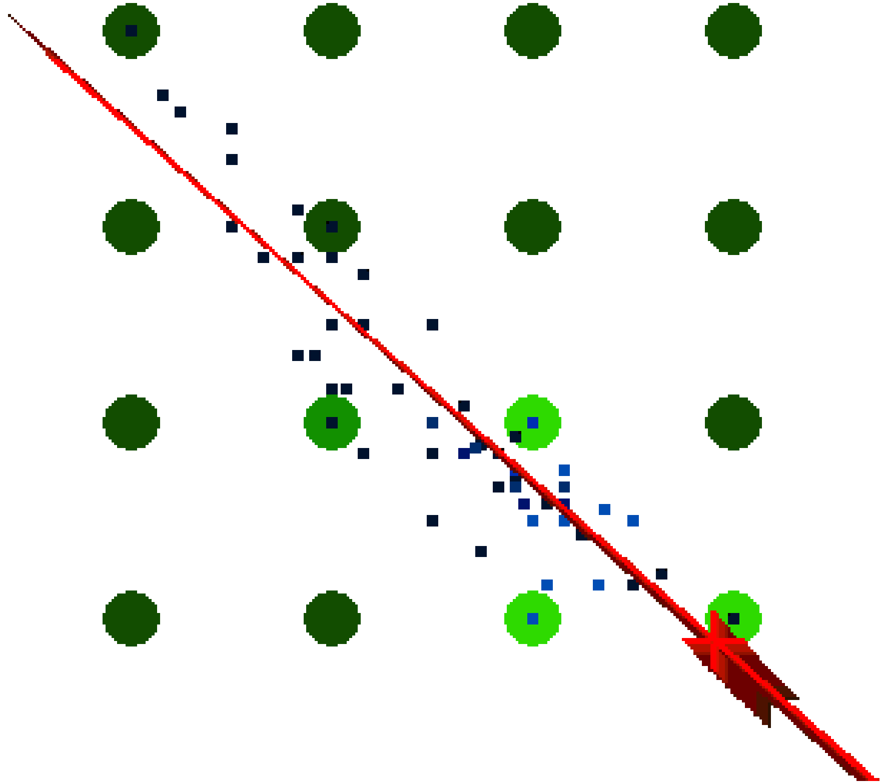


Spanning Tree Module

- Periodically broadcasts heartbeats:
  - construct spanning tree
  - prune dead nodes
  - control topology
- Reduces interference with application messages
  - common messaging layer allows sending tree messages during idle intervals



Approximating Movement



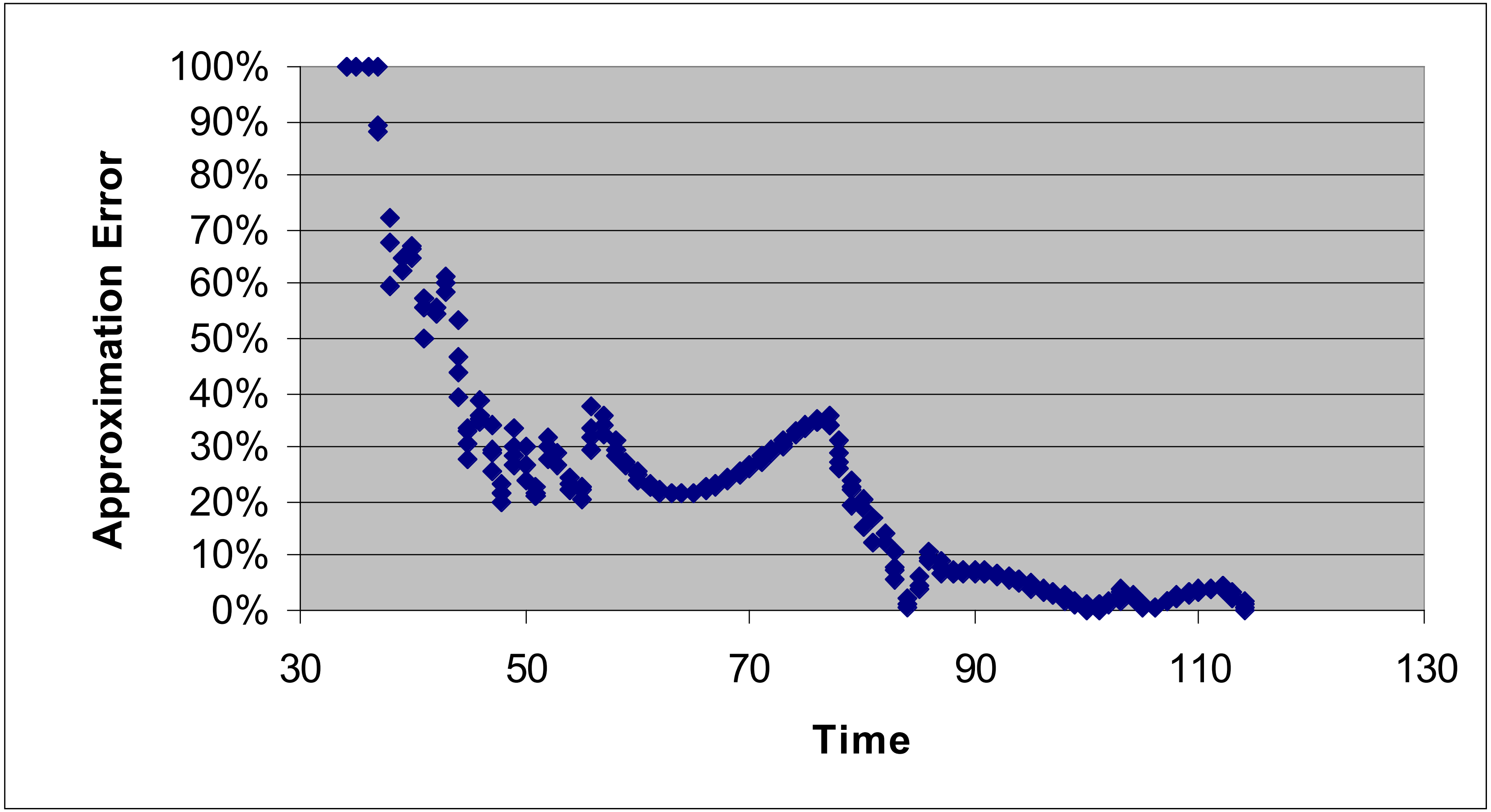
1. Monitoring motes know their location
   - *Currently hard-coded*
   - *Will be computed dynamically*

2. Time synchronization available
- Currently primitive, coarse grain time synchronization.
- Will use an accurate time synchronization which helps synchronize intervals of low power operation.

Measure of Approximation

- Use slope of the line as the measure of correctness of approximation
  - Scalability through piece-wise linear construction of line (introduce memory loss)
- Approximation defined as difference between measured slope and real slope
  - Expressed as a percentage
- Take real slope to be the last estimate of the slope
  - Best solution given all available data

Approximation Error vs. Time



Summary

- Middleware is ripe for formal specification and analysis.
- TLAM is a semantic framework for specifying and reasoning about middleware services
- Probabilistic models are required for network embedded systems
  - Statistical approximations

Future Directions

- Formal Models for Middleware extending Two-Level Actor Semantics:
  - Probabilistic
  - Distributed Time
  - Hybrid
- Provide formal definition for middleware
- Study network embedded systems example applications