**JiST:**
Java in Simulation Time

for

Scalable Simulation of Mobile Ad hoc Networks

Rimon Barr
barr@cs.cornell.edu
Wireless Network Laboratory
Advisor: Prof. Z. J. Haas

MURI Demo
26 August 2003

http://www.cs.cornell.edu/barr/repository/jist/
the world today...

- **Transparent** Parallel and **Optimistic** Execution of Discrete Event Simulations of MANETs in **Java**

- Discrete event simulations are useful and needed
- But, most published ad hoc network simulations
  - Lack network **size** ~250 nodes; or
  - Compromise **detail** packet level; or
  - Curtail **duration** few minutes; or
  - Are of sparse **density** tens of nodes/ km\(^2\); or
  - Etc...

- I.e. limited simulation scalability
the world today... in perspective

• A university **campus**
  - Cornell students ~ 30,000
  - Wireless devices per student average ~ 1
  - Main campus < 4 km².

• The United States **military**
  - Troops deployed in Iraq 100-150,000 (in clusters)
  - Wireless devices per soldier ???
  - Territory 400,000 km²

• And, predictions of
  - smaller devices, better radios and chips
  - smart dust, wearable/disposable/ubiquitous computing

Simulation **scalability** is important.
introduction to jist

- **JiST** - Java in Simulation Time
  - extends object model and execution semantics
    - simulations written in plain Java
  - ... to run discrete event simulations **efficiently**
    - reduces serialization and context-switching overhead
    - allows parallel and speculative simulation execution
  - merges modern language and simulation semantics
jist functionality

- **entities**: extend object model with simulation time components
- **simulation time invocation**: event-based invocation
- **timeless objects**: pass-by-reference to avoid copy
- **proxy entities**: interface-based entity creation
- **continuations**: call and callback, blocking methods
- **concurrency**: channel, threads, monitors, locks...
- **distribution**: separators track entities across machines
- **scripting**: embed engines for Java, Python, Tcl, etc...
a basic example

• the “hello world” of event simulations

```java
class HelloWorld implements JistAPI.Entity {
    public void hello() {
        JistAPI.sleep(1);
        hello();
        System.out.println("hello world, " +
            "time=" + JistAPI.getTime());
    }
}
```

• demo!

<table>
<thead>
<tr>
<th>Java</th>
<th>JiST</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stack overflow @hello</td>
<td>hello world, time=1</td>
</tr>
<tr>
<td></td>
<td>hello world, time=2</td>
</tr>
<tr>
<td></td>
<td>hello world, time=3</td>
</tr>
<tr>
<td></td>
<td>etc.</td>
</tr>
</tbody>
</table>
performance: event throughput

<table>
<thead>
<tr>
<th># events</th>
<th>JiST</th>
<th>GloMoSim</th>
<th>Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>$10^5$</td>
<td>0.044s</td>
<td>0.435s</td>
<td>10%</td>
</tr>
<tr>
<td>$10^6$</td>
<td>0.262s</td>
<td>2.938s</td>
<td>9%</td>
</tr>
<tr>
<td>$10^7$</td>
<td>2.301s</td>
<td>28.04s</td>
<td>8%</td>
</tr>
<tr>
<td>$10^8$</td>
<td>22.48s</td>
<td>278.4s</td>
<td>8%</td>
</tr>
</tbody>
</table>

serial throughput increase of 12x
**performance: memory overhead**

<table>
<thead>
<tr>
<th></th>
<th>memory</th>
<th>entity</th>
<th>event</th>
<th>10K nodes sim.</th>
</tr>
</thead>
<tbody>
<tr>
<td>JiST</td>
<td>36 B</td>
<td>36 B</td>
<td></td>
<td>21 MB</td>
</tr>
<tr>
<td>GloMoSim</td>
<td>36 B</td>
<td>64 B</td>
<td></td>
<td>35 MB</td>
</tr>
<tr>
<td>ns2</td>
<td>544 B</td>
<td>36 B*</td>
<td></td>
<td>72 MB*</td>
</tr>
<tr>
<td>Parsec</td>
<td>28536 B</td>
<td>64 B</td>
<td></td>
<td>2885 MB</td>
</tr>
</tbody>
</table>

**Memory**

- **JiST**: 36 bytes, > 10^6 entities
- **Parsec**: 28536 bytes, ~ 10^4 entities

**JiST scales to more entities per process**
SWANS

- **Scalable Wireless Ad hoc Network Simulator**
  - runs standard Java network applications
  - allows vertical and horizontal aggregation

- **Comparison**
  - Larger than JiST code-base
  - Simpler than GloMoSim and ns2 implementations
  - Less than 3 months

---

Rimon Barr, Wireless Network Lab, Cornell University
performance: SWANS

- simulation configuration
  - field 5x5km²; free-space path loss; no fading
  - radio additive noise; standard power, gain, etc.
  - link 802.11b
  - network IPv4
  - transport UDP
  - mobility random waypoint: v=2-5, p=10
  - application heartbeat neighbor discovery

- ran on:
  - PIII 1.1GHz laptop
  - 384 MB RAM
  - Sun JDK 1.4.2

- memory consumption:
  - 1.2KB per simulated node!
  - demo!

<table>
<thead>
<tr>
<th></th>
<th>nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1,000</td>
</tr>
<tr>
<td>ns2</td>
<td>✓</td>
</tr>
<tr>
<td>GloMoSim</td>
<td>✓</td>
</tr>
<tr>
<td>SWANS</td>
<td>✓</td>
</tr>
</tbody>
</table>
JiST: Java in Simulation Time for Scalable Simulation of Mobile Ad hoc Networks

THANKS!
existing alternatives

ns2 is the gold standard
- C++ with Tcl bindings, $O(n^2)$
- used extensively by community
- written for TCP simulation
- modified for ad hoc networks
- processor and memory intensive
- sequential; max. $\sim 500$ nodes

GloMoSim
- implemented in Parsec, a custom C-like language
- entities are memory intensive
- requires “node aggregation,” which imposes conservative parallelism, loses Parsec benefits
- shown $\sim 10,000$ nodes on NUMA machine (SPARC 1000, est. $\$300k$)

PDNS - parallel distributed ns2
- event loop uses RTI-KIT
- needs fast inter-connect
- distribute memory, $\sim 1000$ nodes

SWAN
- implemented atop the parallel, distributed DaSSF framework
- similar to GloMoSim

OpNet - popular commercial option
- good modeling capabilities
- poor scalability

custom-made simulators
- fast, specialized computation
- lack sophisticated execution and also credibility

Simulation approaches
- languages (e.g. Parsec, Simula)
- libraries (e.g. Yansl, Compose)
- systems (e.g. TWOS, Warped)
simulation time

- program time
  - progress of program independent of time

- real time
  - progress of program is dependent on time

- simulation time
  - progress of time is dependent on program progress
    - instructions take zero (simulation) time
    - time explicitly advanced by the program, sleep
  - simulation event loop embedded in virtual machine
extended object model

- program state contained in **objects**
- **objects** contained in **entities**
  - each entity runs at its own simulation **time**
  - as with objects, entities do not share state
  - think of an entity as a simulation component

![Diagram showing object and entity views of simulation state.](image)
extended execution semantics

- entity references replaced with **separators**
  - event channels; act as **state-time boundary**
- entity methods are an event interface
  - simulation time invocation
  - **non-blocking**; invoked at caller entity time; no continuation
benefits of the jist approach

- more than just scalability
- **application-oriented benefits**
  - type safety: source-target statically checked
  - event types: not required (implicit)
  - event structures: not required (implicit)
  - debugging: dispatch location and state available
- **language-oriented benefits**
  - garbage collection: memory savings, cleaner code
  - reflection: script-based configuration of simulations
  - safety: fine granularity of isolation
  - Java: standard language, compiler, runtime
- **system-oriented benefits**
  - IPC: no context switch; no serialization
  - Java kernel: cross-layer optimization
  - robustness: no memory leaks, no crashes
  - rewriting: no source-code access required
  - concurrency: supports parallel and speculative execution
  - distribution: provides a single system image abstraction
- **hardware-oriented benefits**
  - cost: COTS hardware, clusters (NOW)
  - portability: runs on everything