JiST:
Java in Simulation Time
for
Scalable Simulation of Mobile Ad hoc Networks

Rimon Barr and Zygmunt J. Haas
Wireless Network Laboratory
Cornell University

2nd IRTF Ad hoc Network Scalability Meeting
18 September 2003
Simulation scalability is important

- discrete event simulations are useful and needed
- but, most published ad hoc network simulations
  - lack network size $\sim$250 nodes; or
  - compromise detail packet level; or
  - curtail duration few minutes; or
  - are of sparse density $<10/\text{km}^2$
    i.e. limited simulation scalability

- A university **campus**
  - 30,000 students, $<4\text{ km}^2$, 1 device/student
- The United States **military**
  - 100-150,000 troops, clustered
- **Sensor networks, smart dust, Ubicomp**
  - Many **thousands** of wireless devices in environment

Simulation **scalability** is important
JiST - Java in Simulation Time

- JiST extends Java object model and execution semantics
- ... to run discrete event simulations: transparently
  - simulations written in plain Java
  - compiled classes are modified at load time and efficiently
    - reduces serialization and context-switching overhead
    - allows parallel and speculative simulation execution
- Merges modern language and simulation semantics
  - run Java programs in simulation time
performance: event throughput (and memory)

<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
**SWANS**

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

- **Sim. stack**
  - App
  - SWANS
  - JiST
  - Java

- JiST: shorter and simpler than GloMoSim and ns2
- developed in <3 months

<table>
<thead>
<tr>
<th></th>
<th>files</th>
<th>classes</th>
<th>lines</th>
</tr>
</thead>
<tbody>
<tr>
<td>JiST</td>
<td>26</td>
<td>69</td>
<td>9548</td>
</tr>
<tr>
<td>SWANS</td>
<td>61</td>
<td>127</td>
<td>13999</td>
</tr>
<tr>
<td>Other</td>
<td>18</td>
<td>30</td>
<td>2415</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>105</strong></td>
<td><strong>236</strong></td>
<td><strong>25962</strong></td>
</tr>
</tbody>
</table>
**performance: SWANS**

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

- **ran on:**
  - P III 1.1GHz laptop
  - only **384 MB RAM**
  - Sun JDK 1.4.2

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

<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>Glomo</td>
<td>✓</td>
</tr>
<tr>
<td>SWANS</td>
<td>✓</td>
</tr>
</tbody>
</table>
**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

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

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

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

**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. $\$300k$)

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

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

- **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...
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: pure Java; “runs everywhere”
**performance: memory overhead**

### JiST, Parsec, GloMoSim, ns2

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

<table>
<thead>
<tr>
<th></th>
<th>Memory Limit</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>JiST</strong></td>
<td>36 bytes &gt; 10^6 entities</td>
</tr>
<tr>
<td><strong>Parsec</strong></td>
<td>28536 bytes ~ 10^4 entities</td>
</tr>
</tbody>
</table>

**JiST scales to more entities per process**