JiST:
Java in Simulation Time

for

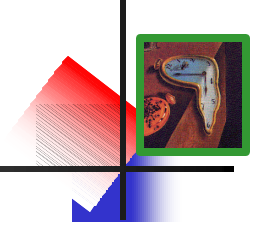
Scalable Simulation of Mobile Ad hoc Networks

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

MURI Poster Session
26 August 2003

http://www.cs.cornell.edu/barr/repository/jist/

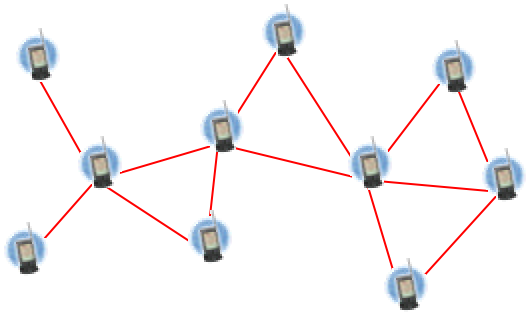
the world today…



- ***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/km²; 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 400,000km²

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

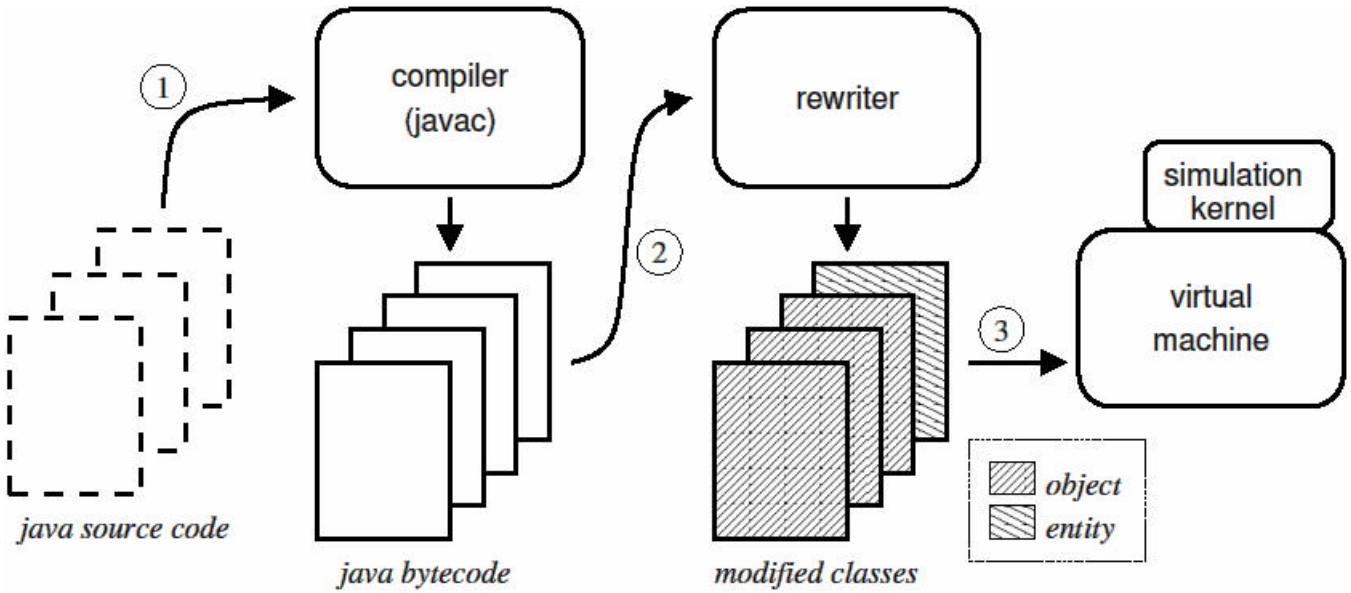
Simulation scalability is important.

introduction to jist

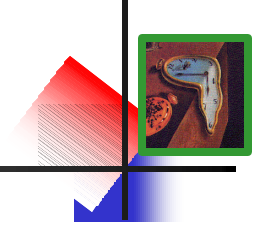
- **Java-based simulation framework**
  - JiST *extends* object model and execution semantics
  - … to run discrete event simulations
  - *transparently*
    - simulations written in plain Java
    - compiled classes are modified at load time
  - and *efficiently*
    - reduces serialization and context-switching overhead
    - allows parallel and speculative simulation execution
  - merges modern language and simulation semantics
    - runs Java programs in simulation time
- **proof of concept**
  - SWANS – Scalable Wireless Ad hoc Network Simulator
  - ideas not specific to Java

system architecture

1. Compile simulation with standard Java compiler.
2. Run simulation within JiST (within Java).
   Simulation classes are dynamically rewritten
   to introduce **simulation time** semantics.
3. Rewritten program interacts with simulation kernel.



a basic example



- **the “hello world” of event simulations**

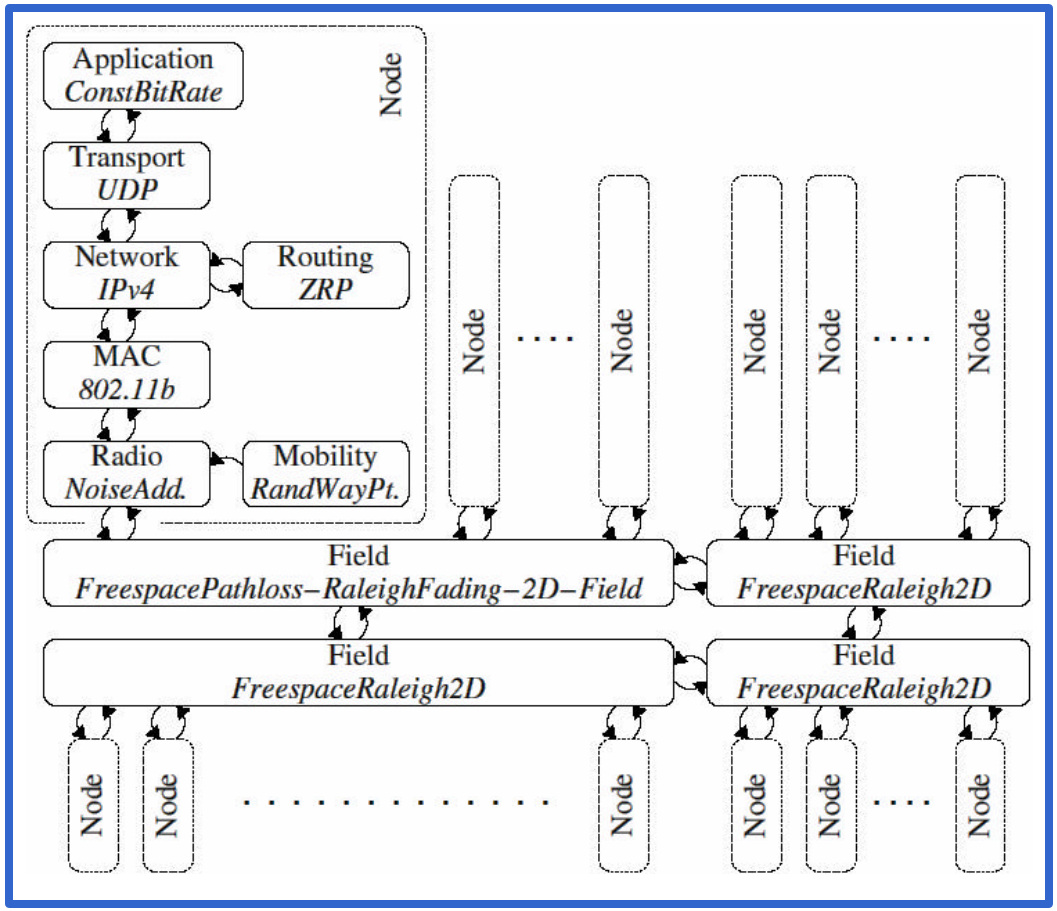
```
class HelloWorld implements JistAPI.Entity
{
  public void hello()
  {
    JistAPI.sleep(1);
    hello();
    System.out.println("hello world, " +
       "time=" + JistAPI.getTime() );
  }
}
```

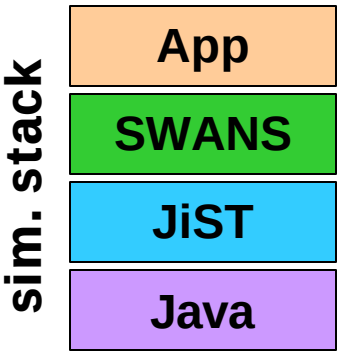
- **demo!**

Java	JiST
Stack overflow @hello	hello world, time=1 hello world, time=2 hello world, time=3 etc.

SWANS

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

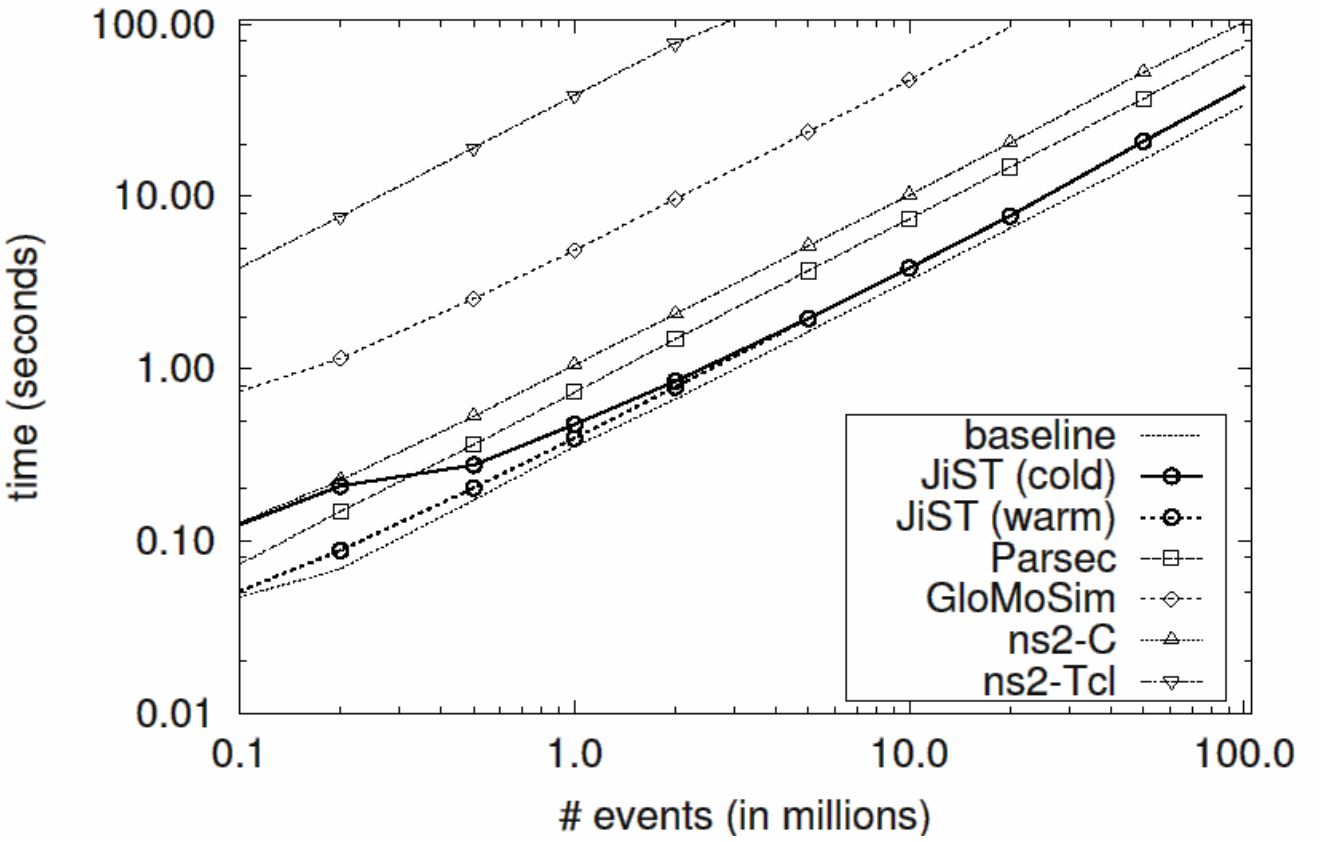




	files	classes	lines
JiST	26	65	9278
SWANS	52	115	12871
Other	16	26	2042
	94	**206**	**24191**

- larger than JiST code-base
- simpler than GloMoSim and ns2 implementations
- developed in <3 months

performance: event throughput

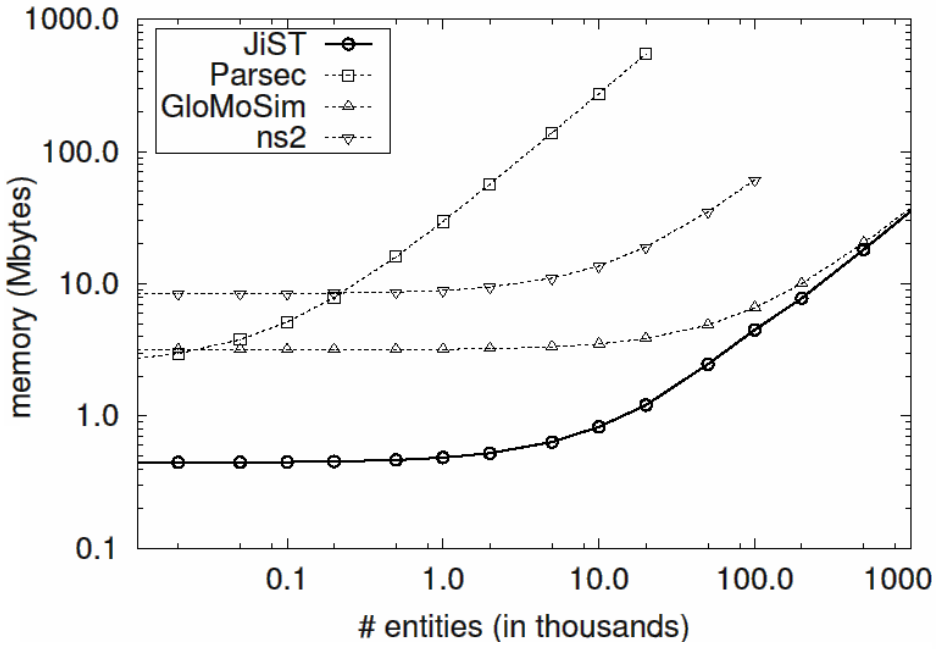


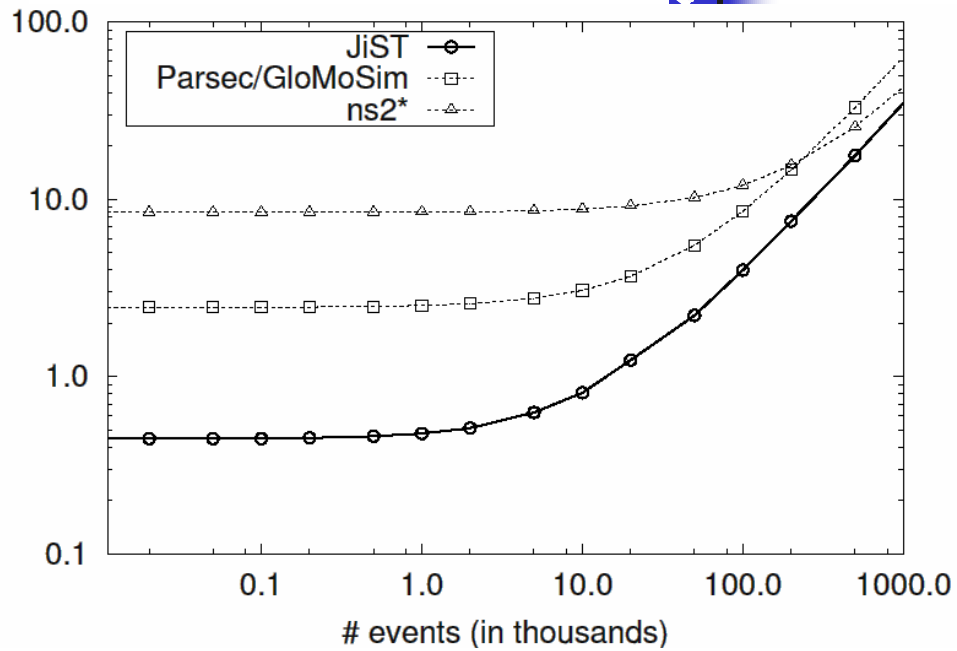
# events	JiST	GloMoSim	Ratio
10^5	0.044s	0.435s	10%
10^6	0.262s	2.938s	9%
10^7	2.301s	28.04s	8%
10^8	22.48s	278.4s	8%

serial throughput increase of 12x

5×10^6 events	time (sec)	vs. baseline	vs. JiST
baseline	1.640	1.0x	0.8x
JiST	**1.957**	**1.2x**	**1.0x**
Parsec	3.705	2.3x	1.9x
ns2-C	5.151	3.1x	2.6x
GloMoSim	23.720	14.5x	12.1x
ns2-Tcl	160.514	97.9x	82.0x

performance: memory overhead



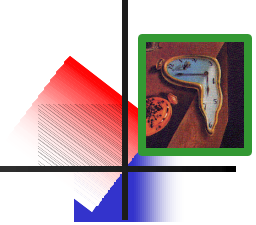


memory	entity	event	10K nodes sim.
JiST	**36 B**	**36 B**	**21 MB**
GloMoSim	36 B	64 B	35 MB
ns2	544 B	36 B*	72 MB*
Parsec	28536 B	64 B	2885 MB

	Memory	Limit
JiST	36 bytes	> 10^6 entities
Parsec	28536 bytes	~ 10^4 entities

JiST scales to more entities per process

performance: SWANS

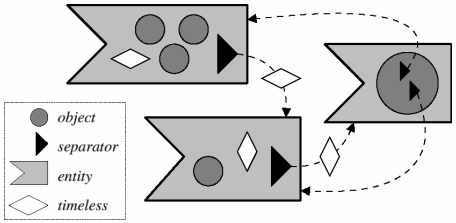


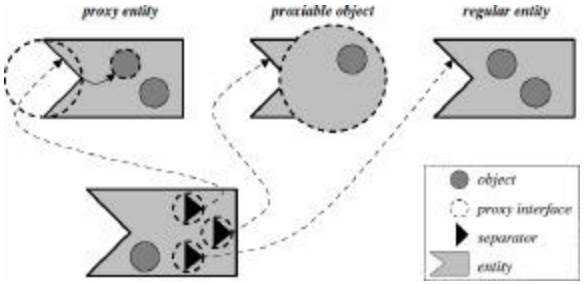
- simulation configuration
  - field — 5x5km²; free-space path loss; no fading
  - mobility — random waypoint: v=2-5m, p=10s
  - radio — additive noise; standard power, gain, etc.
  - link — 802.11b
  - network — IPv4
  - transport — UDP
  - application — heartbeat neighbor discovery
- ran on:
  - PIII 1.1GHz laptop
  - only 384 MB RAM
  - Sun JDK 1.4.2
- memory consumption:
  - 1.2KB per simulated node!

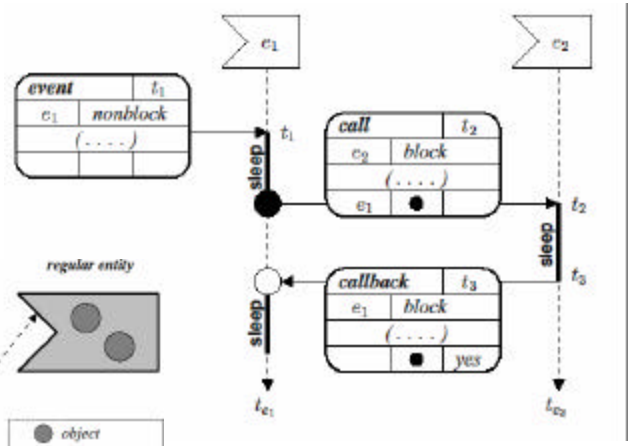
	nodes		
	1,000	10,000	100,000
ns2	☑	☒	☒
Glomo	☑	☑	☒
SWANS	☑	☑	☑

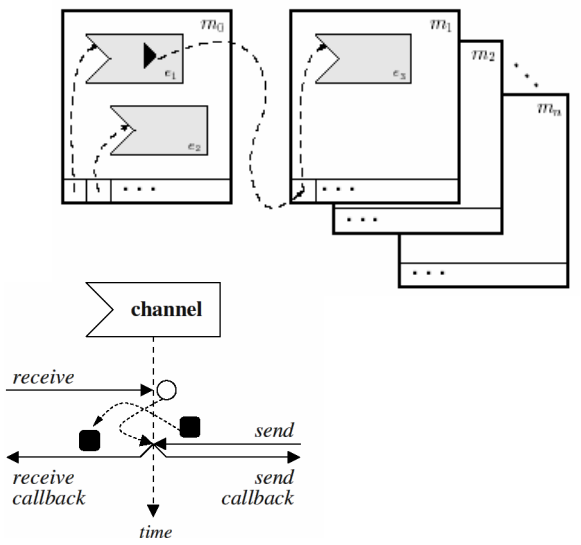
and lots more!

- **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”