JiST:
Java in Simulation Time

for

Scalable Simulation of Mobile Ad hoc Networks

Rimon Barr and Zygmunt J. Haas

Wireless Network Laboratory

Cornell University

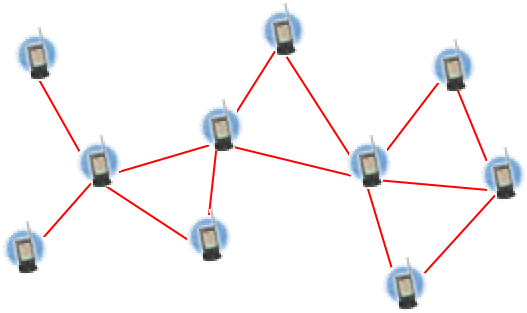
2nd IRTF Ad hoc Network Scalability Meeting

18 September 2003

simulation scalability is important

- discrete event simulations are useful and needed
- but, most published ad hoc network simulations
  - lack network *size* ~250 nodes; or
  - compromise *detail* packet level; or
  - curtail *duration* few minutes; or
  - are of sparse *density* $<10/km^2$

  i.e. limited simulation scalability



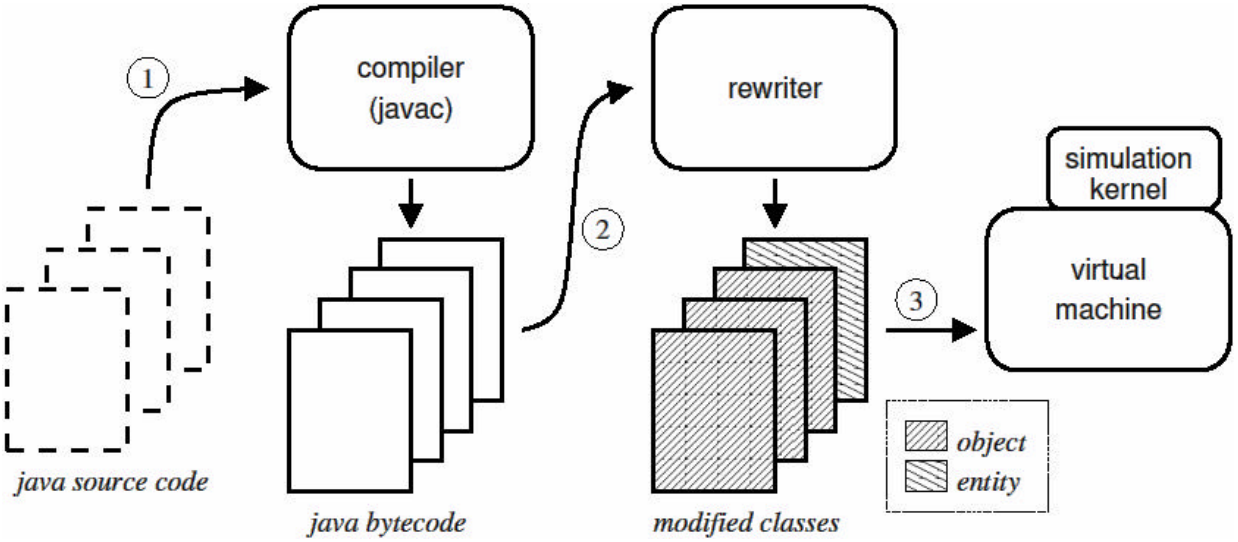
- A university *campus*
  - 30,000 students, < 4 km^2, 1 device/student
- The United States *military*
  - 100-150,000 troops, clustered
- Sensor networks, smart dust, Ubicomp
  - Many thousands of wireless devices in environment

Simulation scalability is important

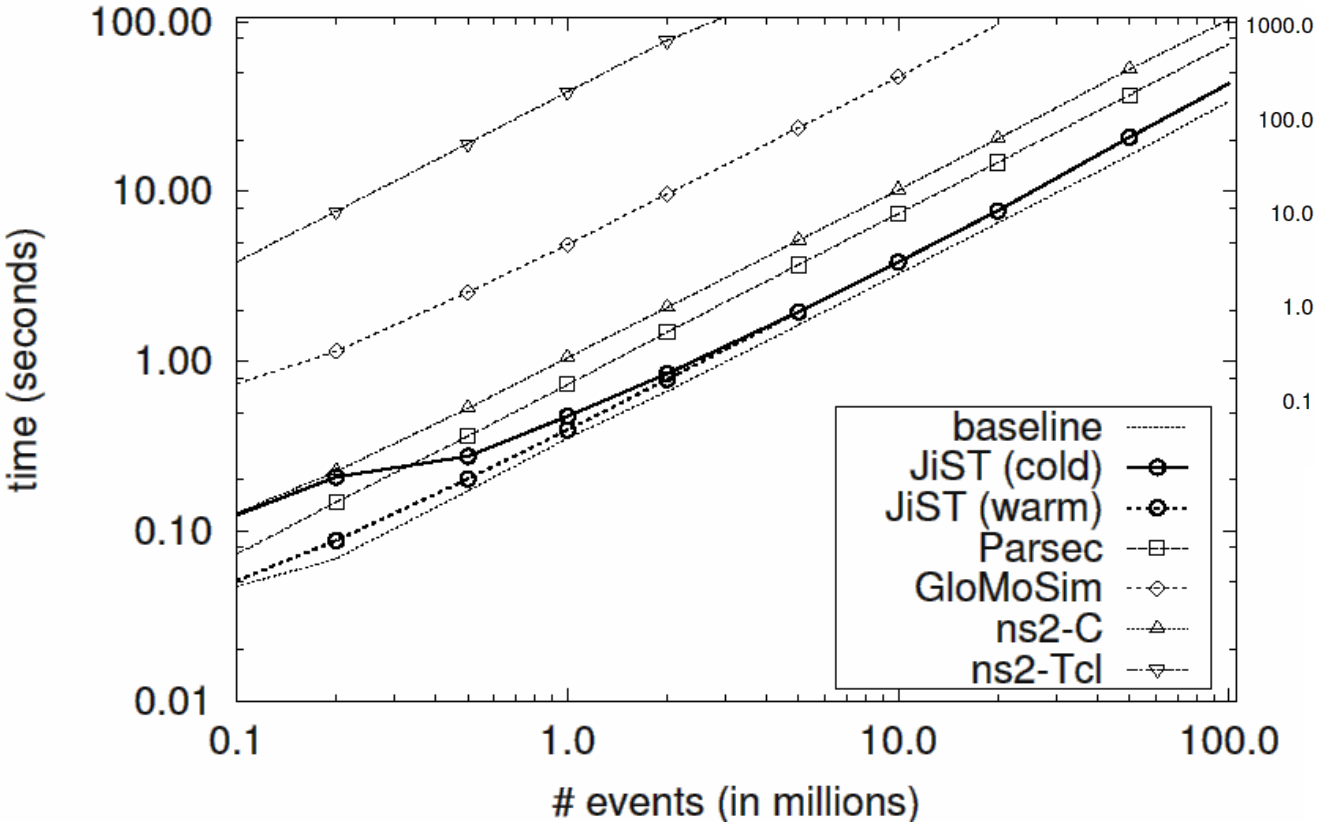
JiST – Java in Simulation Time

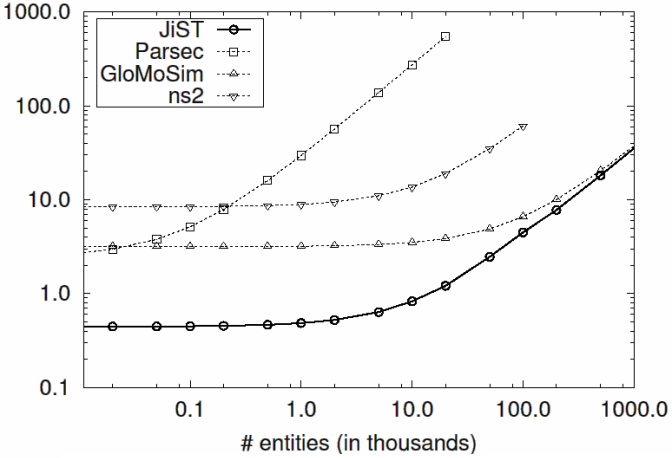
- JiST *extends* Java object model and execution semantics
- … to run discrete event simulations: *transparently*
  - simulations written in plain Java
  - compiled classes are modified at load time

  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)





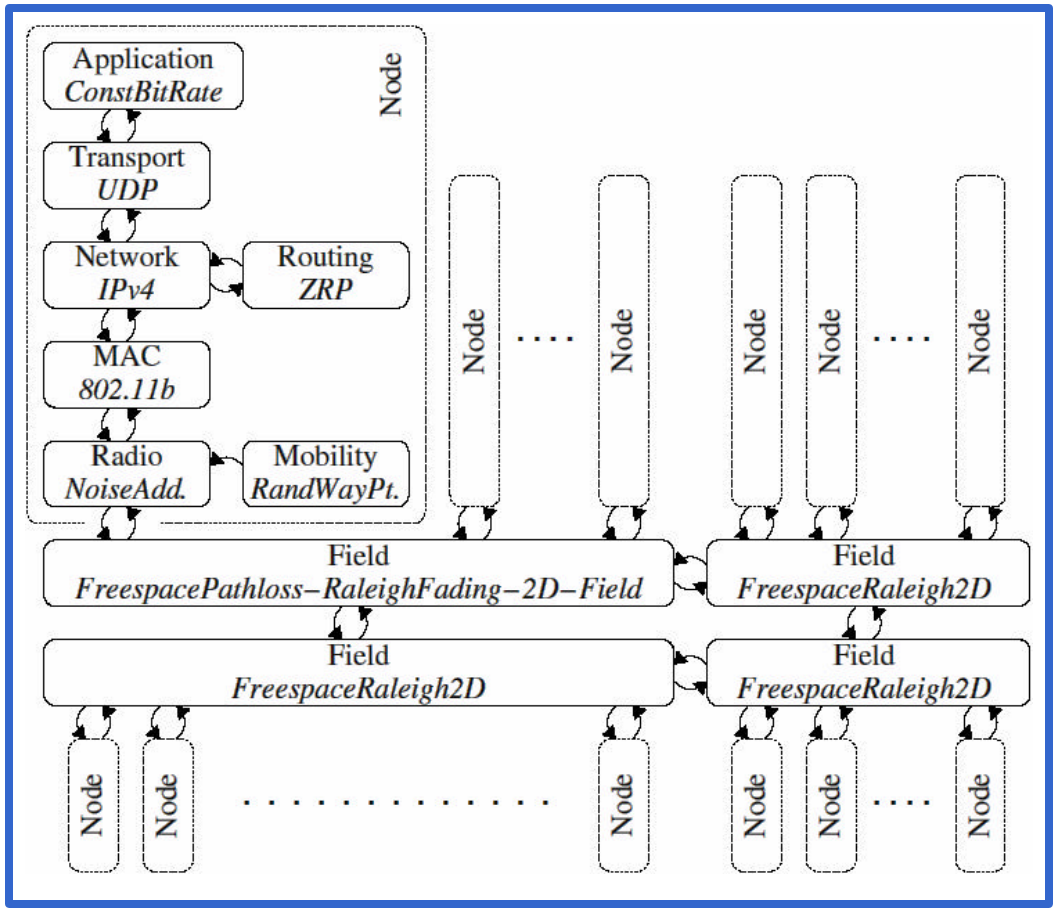
5×10^6 events	time (sec)	vs. baseline	vs. JiST
baseline	1.640	1.0x	0.8x
JiST	**1.957**	**1.2x**	**1.0x**
Parsec	3.705	2.3x	1.9x
ns2-C	5.151	3.1x	2.6x
GloMoSim	23.720	14.5x	12.1x
ns2-Tcl	160.514	97.9x	82.0x

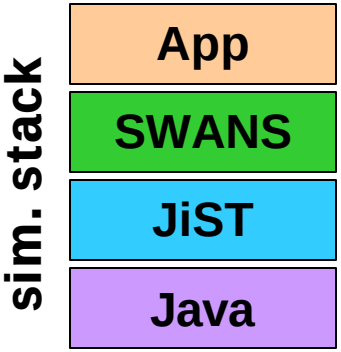
# events	JiST	GloMoSim	Ratio
10^5	0.044s	0.435s	10%
10^6	0.262s	2.938s	9%
10^7	2.301s	28.04s	8%
10^8	22.48s	278.4s	8%

serial throughput increase of 12x

SWANS

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

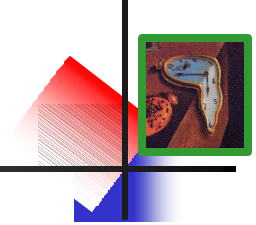




	files	classes	lines
JiST	26	69	9548
SWANS	61	127	13999
Other	18	30	2415
	105	**236**	**25962**

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

performance: SWANS

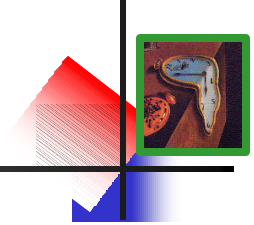


- simulation configuration
  - field — 5x5km²; free-space path loss; no fading
  - mobility — random waypoint: v=2-5m, p=10s
  - radio — additive noise; standard power, gain, etc.
  - link — 802.11b
  - network — IPv4
  - transport — UDP
  - application — heartbeat neighbor discovery
- ran on:
  - PIII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

backup slides

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. ~500 nodes**

PDNS **– parallel distributed ns2**

- **event loop uses RTI-KIT**
- **needs fast inter-connect**
- **distribute memory, ~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 ~10,000 nodes on NUMA machine (SPARC 1000, est. $300k)**

SWAN

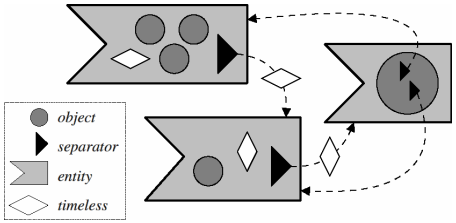
- **implemented atop the parallel, distributed DaSSF framework**
- **similar to GloMoSim**

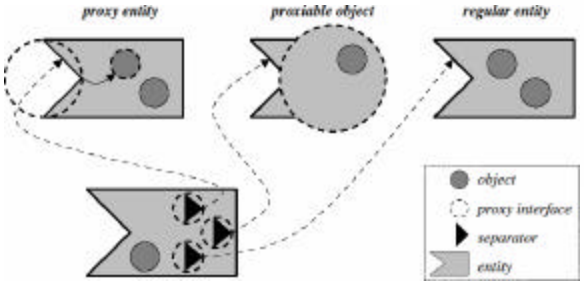
Simulation approaches

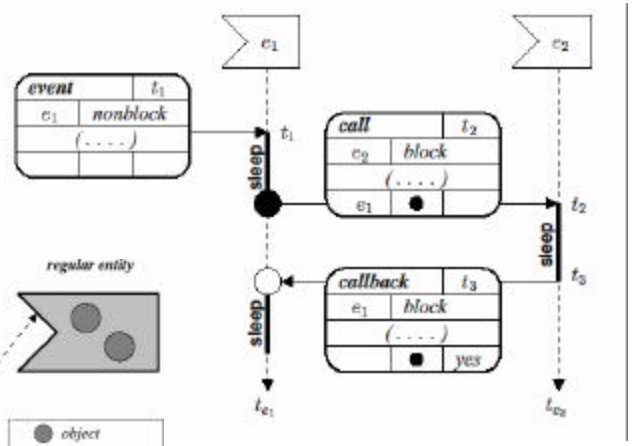
- **languages (e.g. Parsec, Simula)**
- **libraries (e.g. Yansl, Compose)**
- **systems (e.g. TWOS, Warped)**

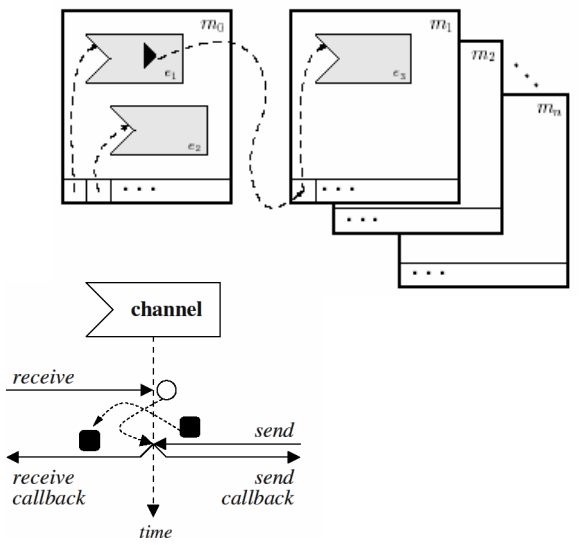
a lot more than simulation time

- **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...





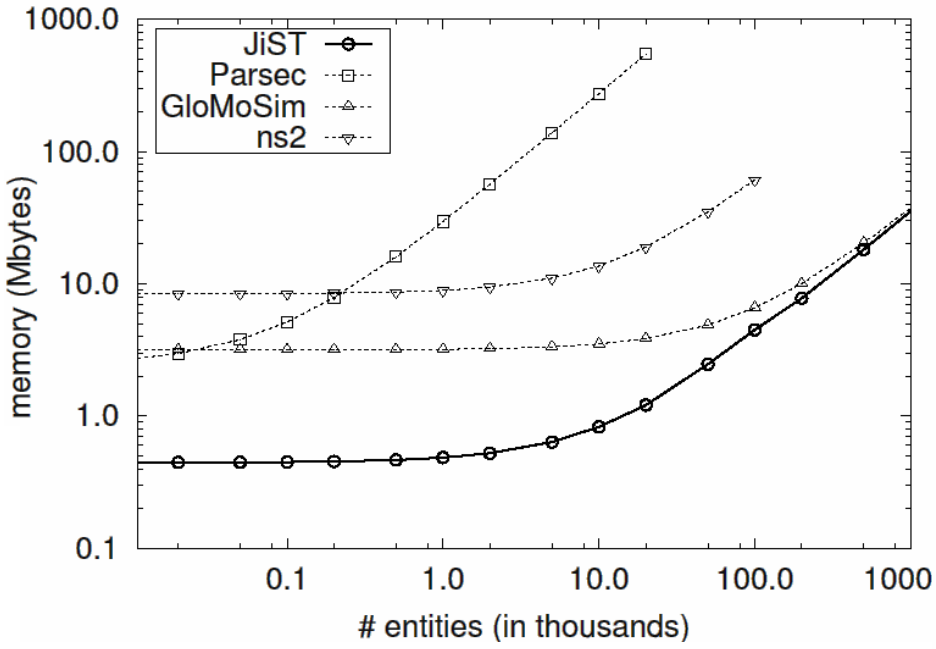


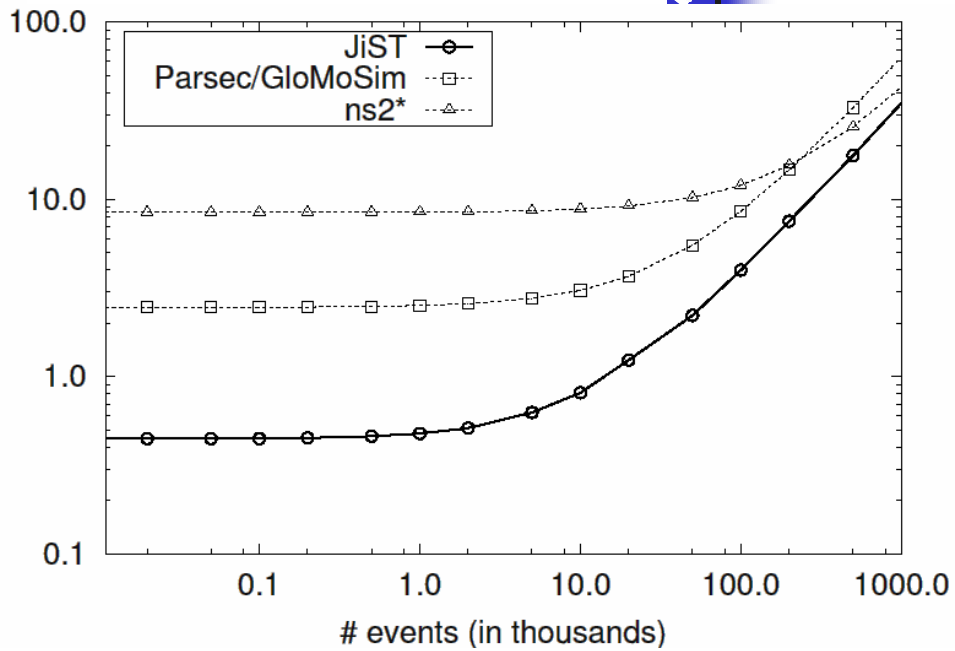


benefits of the jist approach

- 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





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