Seminário

TOSSIM: TinyOS Simulator

Projeto SensorNet

Breno Augusto Dias Vitorino

Luiz Filipe M. Vieira

Sumário

- TinyOS
  - História
  - O que é TinyOS?
  - Projeto, Kernel, Modelo de Execução
- nesC
- TOSSIM
  - Conceitos, O que é?, Limitações
  - Modelo Rádio, Modelo Sensor
  - Exemplo

História

- Inicialmente desenvolvido por Jason Hill, dissertação de Mestrado - UC Berkeley ,2000. Orientado por David Culler (Berkeley/Intel).
- Parte do Projeto Berkeley WEBS.
- Atualmente, o laboratório Intel-Berkeley Research é o maior responsável pelos componentes deste SO.
- – www.intel-research.net/berkeley/

O que é o TinyOS?

- Um Ambiente de Desenvolvimento de Código Aberto
- Um Simples Sistema Operacional
- Um Modelo e uma Linguagem de Programação
- Um Conjunto de Serviços

TinyOS – Características

- Microprocessador ATMEGA 128L
- É um sistema operacional muito simples e compacto, baseado em eventos
- Desenvolvido para apoiar as aplicações de RSSF
- Operações intensamente concorrentes com mínimo de requisitos de hardware e economizando energia.

TinyOS – Um Simples Sistema Operacional

- Escalonador
- Intensamente Concorrente
- Recursos Limitados – componentes de software para modularidade e eficiência.

TinyOS – Um Modelo e uma Linguagem de Programação

- Separação entre construção e composição:
  - programas são construídos a partir de componentes
- Especificação do comportamento dos componentes em termos de um conjunto de interfaces
- Componentes são estaticamente ligados com outros através de interfaces.
  - aumenta eficiência em tempo de execução

TinyOS - Serviços

- Rádio, MAC, Mensagens, Roteamento
- Interface para Sensores
- Gerência de Energia
- Segurança
- Depuração
- Temporização

TinyOS - Objetivos de Projeto

- Apoiar Sistemas Embutidos para Redes
  - dormir mas permanecer vigilante a estímulos
  - rajada de eventos e operações
- Suporte ao Hardware do Mica
  - energia, sensores, computação, comunicação
- Suporte aos Avanços Tecnológicos
  - manter no ritmo de “scale-down”
  - menor, mais barato, baixo consumo de energia (lower power)

TinyOS - Opções de Desenho

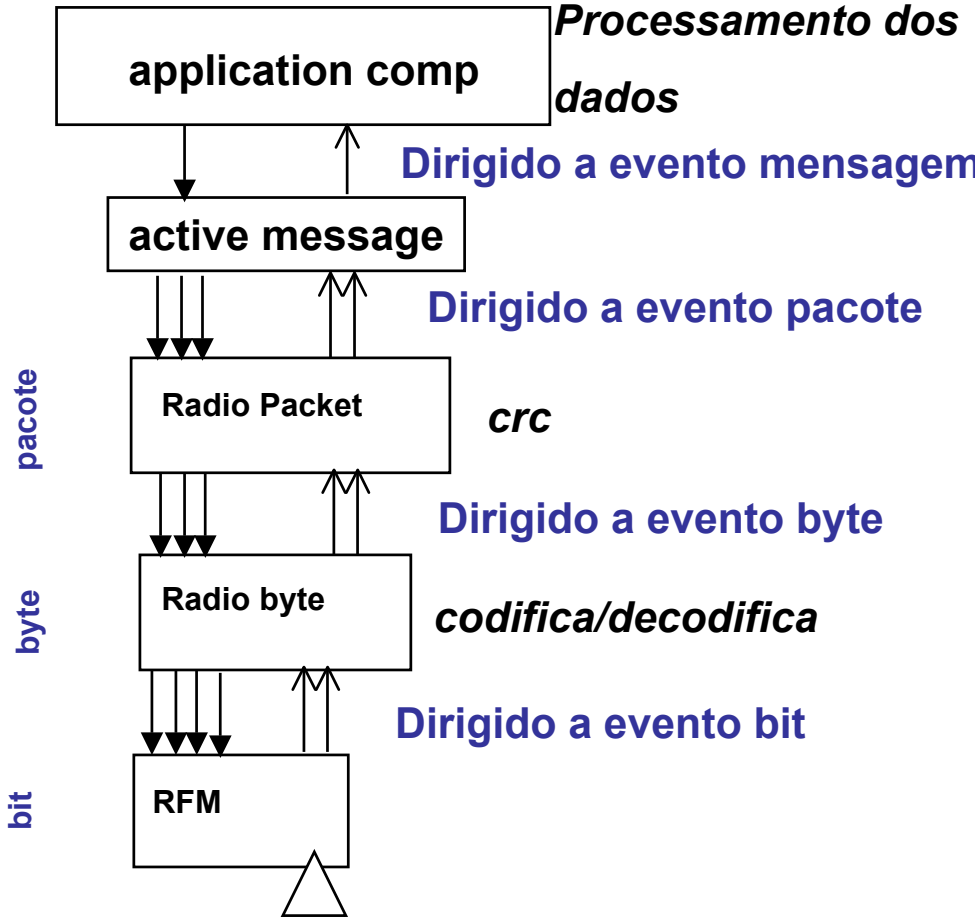
- Não podia usar RTOS existentes
  - Arquitetura de Microkernel
    - VxWorks, QNX, WinCE, PalmOS
  - Execução similar a sistemas Desktop
- PDA's, Telefones Celulares, PC's embutidos
  - Computacionalmente mais caro (mais de uma ordem de magnitude)
  - Consumo de energia

TinyOS - Projeto do Kernel

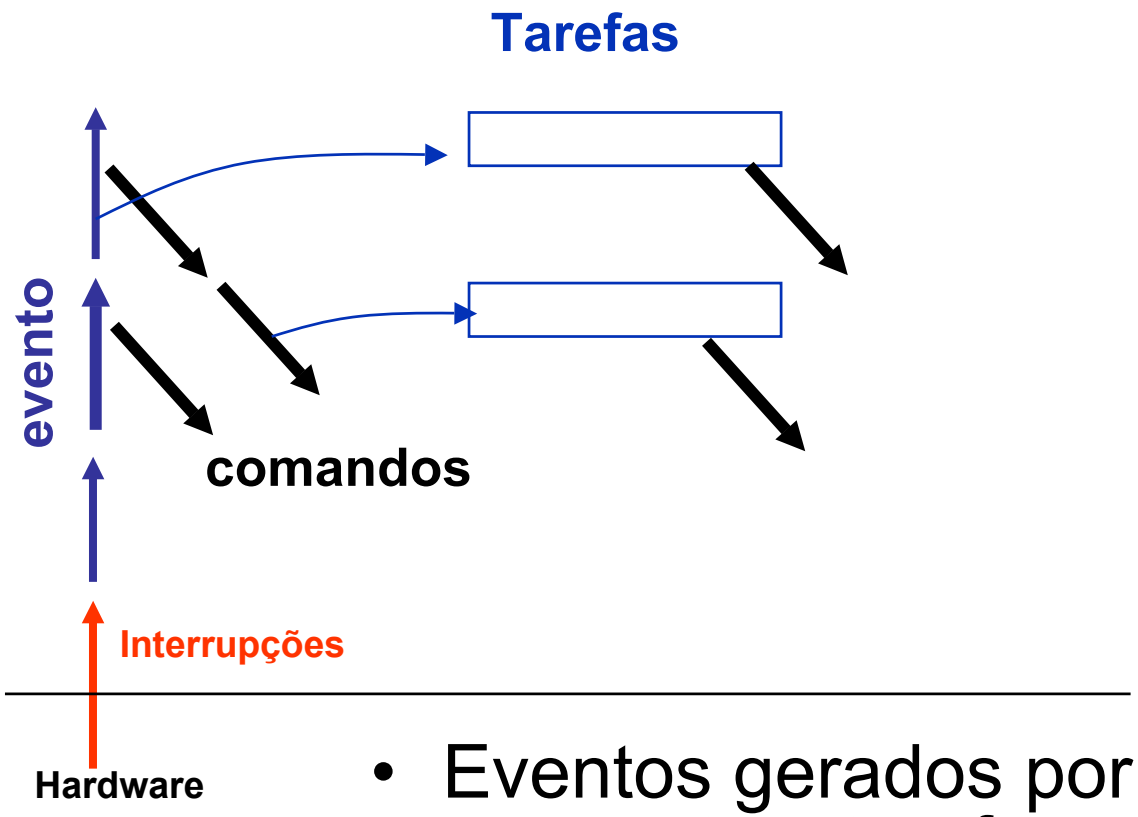
- Kernel do TinyOS: 2 Níveis de Escalonamento
  - Eventos
    - Pequena Quantidade de Processamento
    - Ex. Relógios, Interrupções ADC
    - Podem interromper tarefas
  - Tarefas
    - Não são críticas em relação ao tempo
    - Maior quantidade de processamento
    - Ex. Computar a média de um vetor
    - Executam até completarem

TinyOS - Modelo de Execução

- comandos requisitam ações
  - chamam comandos ou postam tarefas
- eventos notificam ocorrências
  - interrupções de HW
  - podem sinalizar eventos
  - chamam comandos
  - postam tarefas
- Tarefas provêem concorrência lógica
  - preemptados por eventos



TinyOS - Execução dos Contextos



- Eventos gerados por interrupção preemptam tarefas
- Tarefas não preemptam tarefas

Tarefas

- provêem concorrência interna para um componente
  - operações que executam mais tempo
- são preemptadas por eventos
- capazes de realizar operações além do contexto do evento
- podem chamar comandos
- podem sinalizar eventos
- não interrompem tarefas

```
{
...
post TskName();
...
}
```

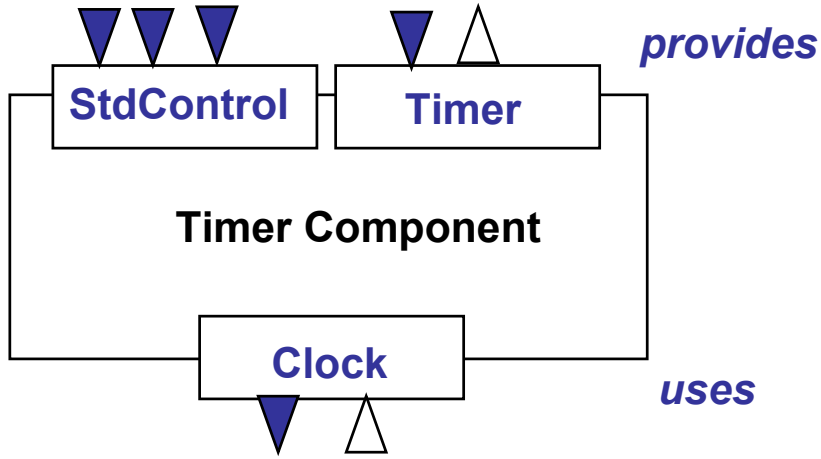
```
task void TskName {
...
}
```

Composição

- Um componente especifica um conjunto de *interfaces* pelo qual ele está conectado a outros componentes
  - provê um conjunto de interfaces para outros componentes
  - usa um conjunto de interfaces provido por outros componentes
- Interfaces são bidirecionais
  - incluem comandos e eventos
- Métodos da interface são os nomes externos dos componentes

Composição

```
provides
      interface StdControl;
    interface Timer:
uses
   interface Clock
```



Componentes

- Módulos
  - prove código que implementa uma ou mais interfaces e o comportamento interno
- Configurações
  - juntam componentes para formar um novo componente
- Interface
  - relacionam um conjunto de comandos e eventos

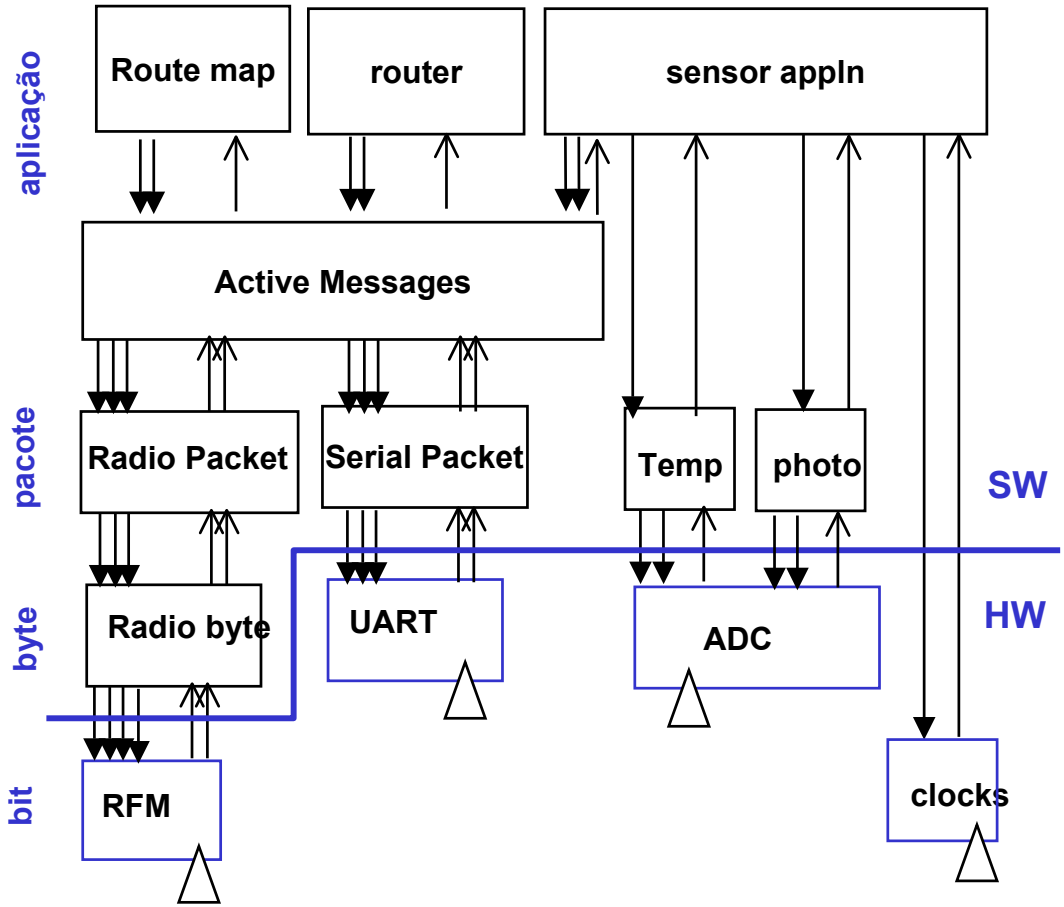
StdControl.nc

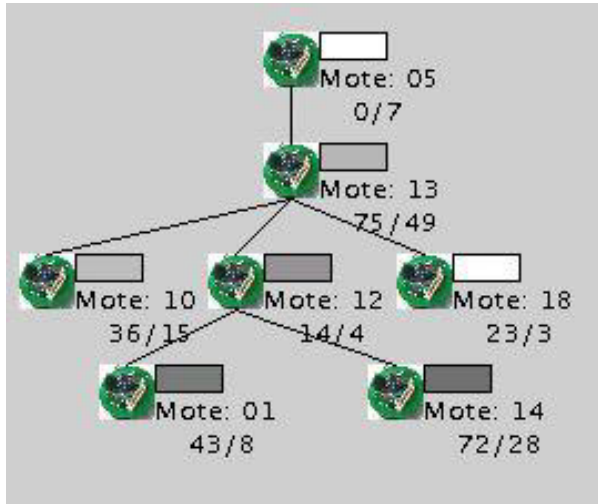
```
interface StdControl {
      command result_t init();
      command result_t start();
      command result_t stop();
}
```

Clock.nc

```
interface Clock {
   command result_t setRate(char interval, char scale);
   event result_t fire();
}
```

Aplicação = Grafo de Componentes





Exemplo: roteamento multi-hop das leituras do sensor de luz

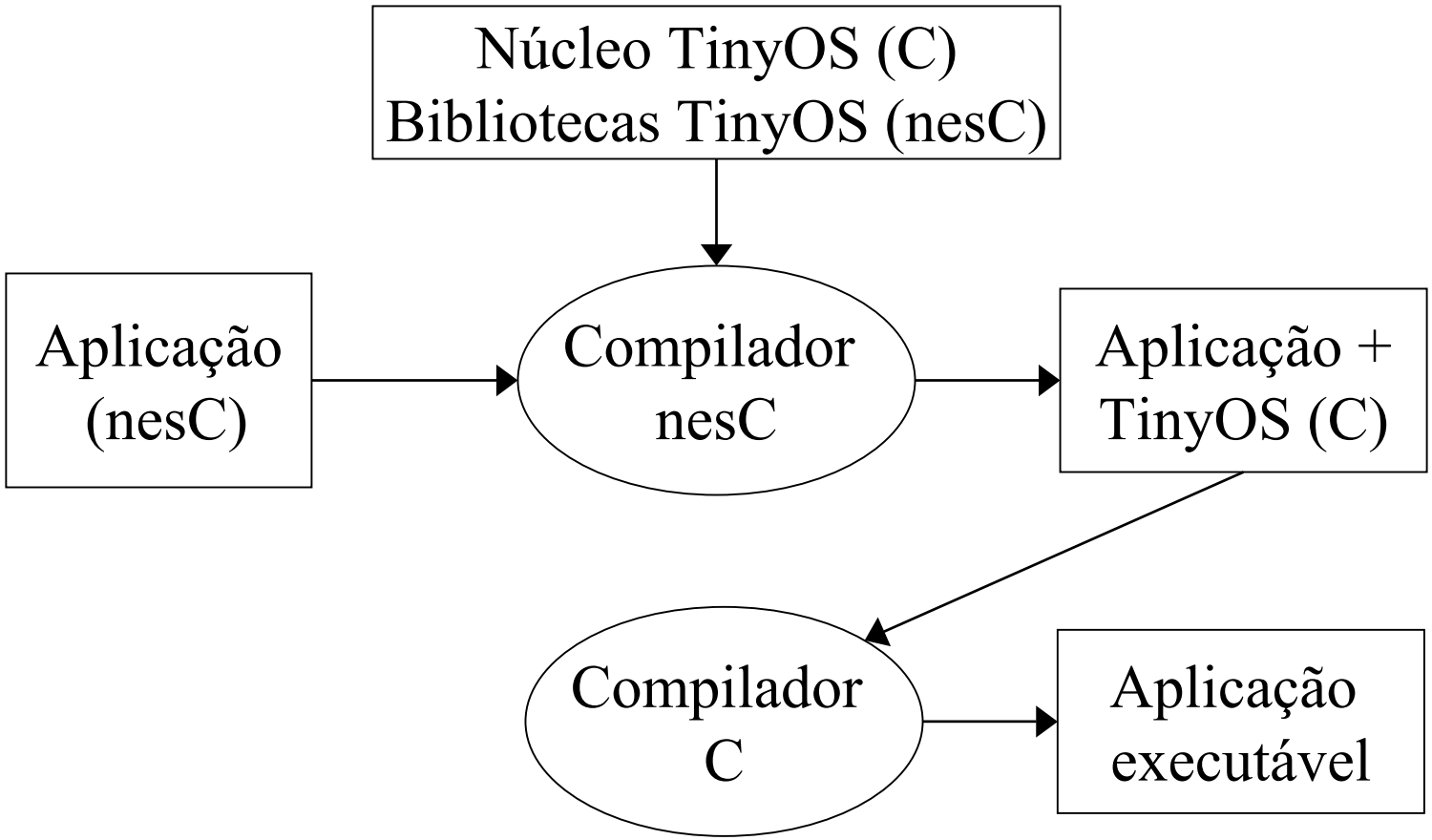
3450 B código
226 B dados

Grafo de cooperação

nesC

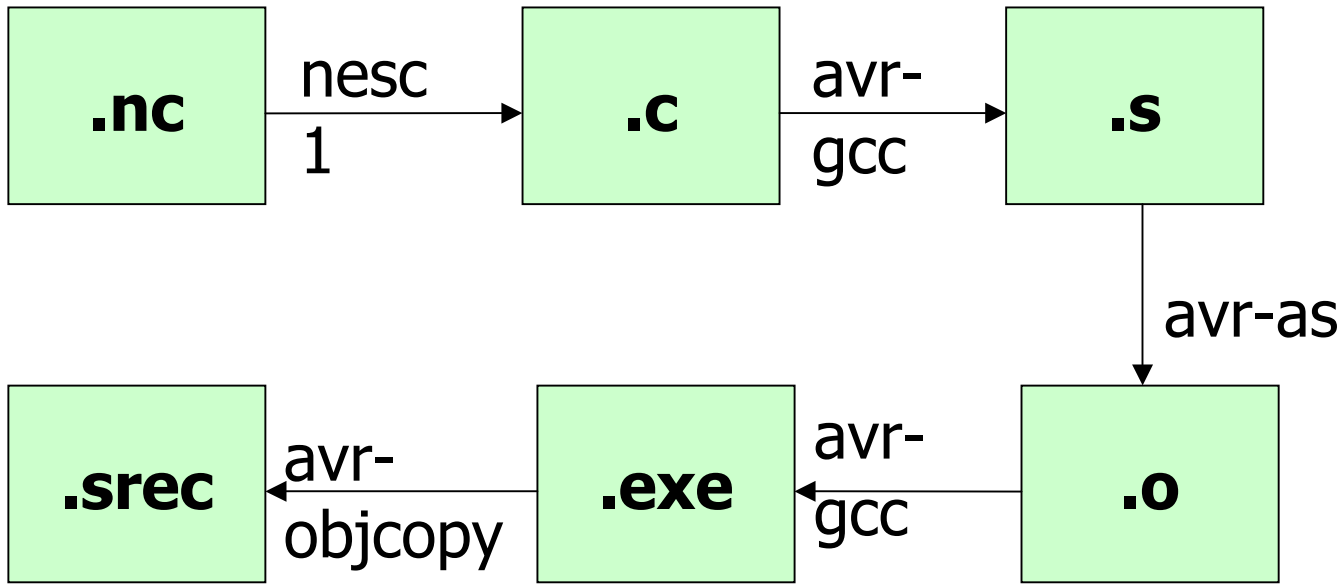
- TinyOS e componentes são escritos em nesC
- NesC (pronunciado "NES-see") é uma extensão da linguagem de programação C projetada para adicionar os conceitos e modelo de execução do TinyOS.
- Mesmos conceitos do TinyOS
  - Especificação do comportamento dos componentes em termos de um conjunto de interfaces
  - Interfaces são bidirecionais
  - Componentes são estaticamente ligados a outros através das interfaces

Desenvolvimento de aplicações no TinyOS



Fluxo de compilação (1/2)

- Tipos de arquivo e aplicações

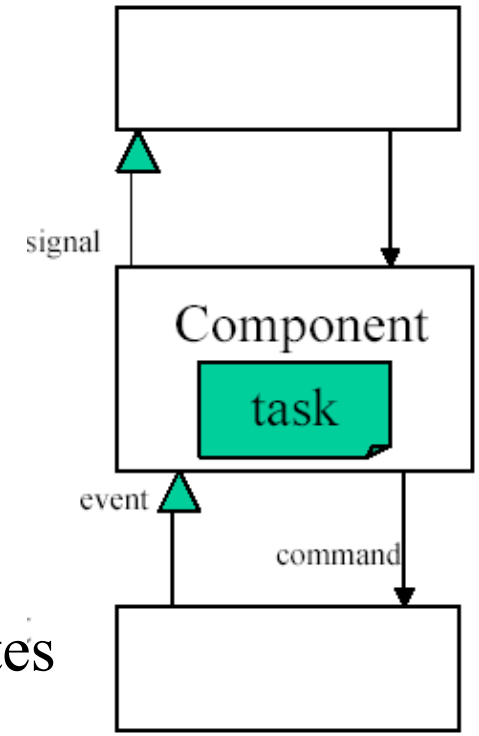


Fluxo de compilação (2/2)

- Tipos de arquivo:
  - .nc: configuração do TinyOS;
  - .c: C p/ AVR;
  - .s: montagem p/ AVR;
  - .o: objeto p/ AVR;
  - .exe: linkeditado p/ AVR;
  - .srec: S-records p/ AVR.

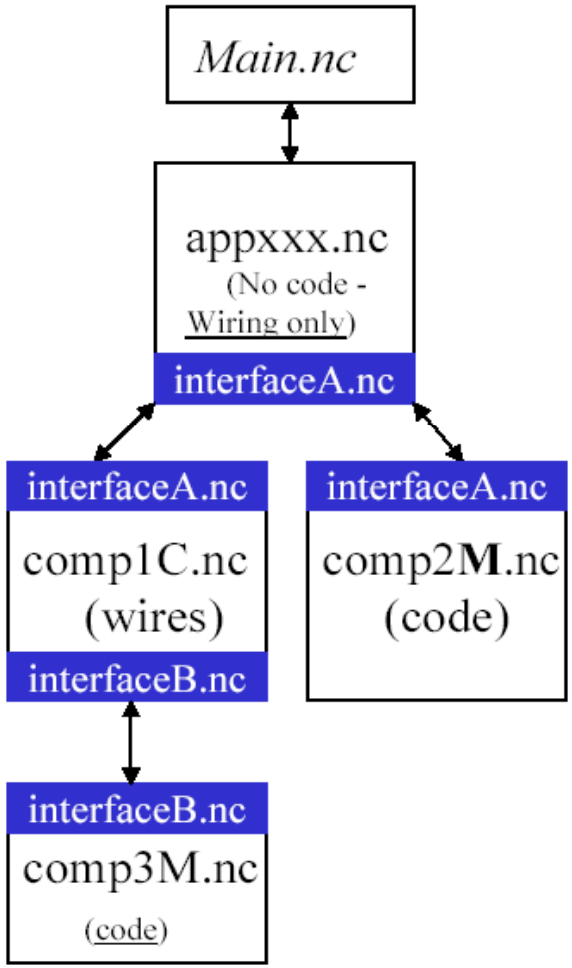
Conceitos do TinyOS embutidos no nesC – Tarefas, Eventos, Comandos

- **Tarefas**
  - Computações em “background”, não são críticas em relação a tempo
- **Eventos**
  - Críticos em relação ao tempo
  - Interrupções externas
  - Originador envia um ‘**Sinal**’
  - Receptor aceita o ‘**Evento**’
- **Comandos**
  - Chamadas de funções para outros componentes
  - Não **sinalizam**



Conceitos dos componentes de SW

- **Interfaces** (xxx.nc)
  - Especificam funcionalidades para o mundo exterior
  - Dizem ao mundo exterior
    - quais comandos podem ser chamados
    - quais eventos precisam ser lidados
- **Componentes de Software**
  - **Module** (xxxM.nc)
    - Arquivo de código com implementação
    - Codifica a **Interface**
  - **Configuration** (xxxC.nc)
    - Ligação de componentes
    - Quando no nível mais alto , não usa a letra C no nome do arquivo xxx.nc



Módulo

- Declaração das interfaces que um componente provê (linhas 2 a 4)
- Interfaces utilizadas (5 a 7)
- No escopo *implementation* (linhas 9 a 16), um módulo implementa os comandos das interfaces providas e os eventos das interfaces das quais se utiliza.

```
1  module M {
2      provides {
3          interface I as P;
4      }
5      uses {
6          interface I as U;
7      }
8  }
9  implementation {
10     command void P.f1(int i) {
11         // Implementacao de f1...
12     }
13     event result_t P.f2(int j) {
14         // Implementacao de f2...
15     }
16 }
```

Configuração

- É o componente que declara como um certo conjunto de módulos deve ser conectado, a fim de oferecer um serviço.
- As interfaces que provê e as interfaces de que se utiliza (linhas 2 a 7).

```
1  configuration C {
2      provides {
3          interface I as X;
4      }
5      uses {
6          interface I as Y;
7      }
8  }
9  implementation {
10     components M;
11
12     X = M.P;
13     M.U -> M.P;
14 }
```

Ligação das Interfaces

- A ligação M.x -> N.y diz que a interface x usada pelo componente M é implementada pela interface y provida pelo componente N.
- Declarações implícitas:
  - M.X -> N é equivalente a M.X -> N.X

Configuração da aplicação principal

- Arquivo <NomeDaAplicacao>.nc.
- não provê nem utiliza nenhuma interface.
- Deve incluir obrigatoriamente o componente *Main*, que possui apenas a interface *StdControl*.
- Essa interface possui os métodos de inicialização de uma aplicação.

```
1
2 configuration App {
3 }
4 implementation {
5     components Main, N;
6
7     Main.StdControl -> N.StdControl;
8     // Mais ligacoes...
9 }
```

Blink.nc

```
configuration Blink {
}
implementation {
  components Main, BlinkM, SingleTimer, LedsC;
  Main.StdControl -> BlinkM.StdControl;
  Main.StdControl -> SingleTimer.StdControl;
  BlinkM.Timer -> SingleTimer.Timer;
  BlinkM.Leds -> LedsC;
}
```

BlinkM.nc

```
module BlinkM {
  provides {
    interface StdControl;
  }
  uses {
    interface Timer;
    interface Leds;
  }
```

BlinkM.nc

```
implementation {
  command result_t StdControl.init() {
    call Leds.init();
    return SUCCESS;
  }
  command result_t StdControl.start() {
    return call Timer.start(TIMER_REPEAT, 1000) ;
  }
  command result_t StdControl.stop() {
    return call Timer.stop();
  }
  event result_t Timer.fired() {
    call Leds.redToggle();
    return SUCCESS;
  }
}
```

nesC Manual

- Para mais detalhes sobre a linguagem, o leitor deve consultar o manual de referência da linguagem nesC.
- GAY, D. et al. nesC v1.1 Language Reference Manual. maio 2003.
- Disponível em:

  http://nescc.sourceforge.net/papers/nesc-ref.pdf.

Pilha do Rádio do MICA2 para o TinyOS

- A família MICA2 dos nós sensores motes usa o modelo transceptor RF ChipCon CC1000 single-chip
  - Frequência 300-1000 MHz
  - Modulação FSK com transmissão de dados de até 76.8 kBaud
  - Codificação Manchester
  - Integrado com sincronizador de bit
  - Sensitividade 110 dBm (@ 2.4 kBaud)
  - Permite selecionar modos de potência de transmissão
  - Interface de controle digital usando registradores de funções especiais

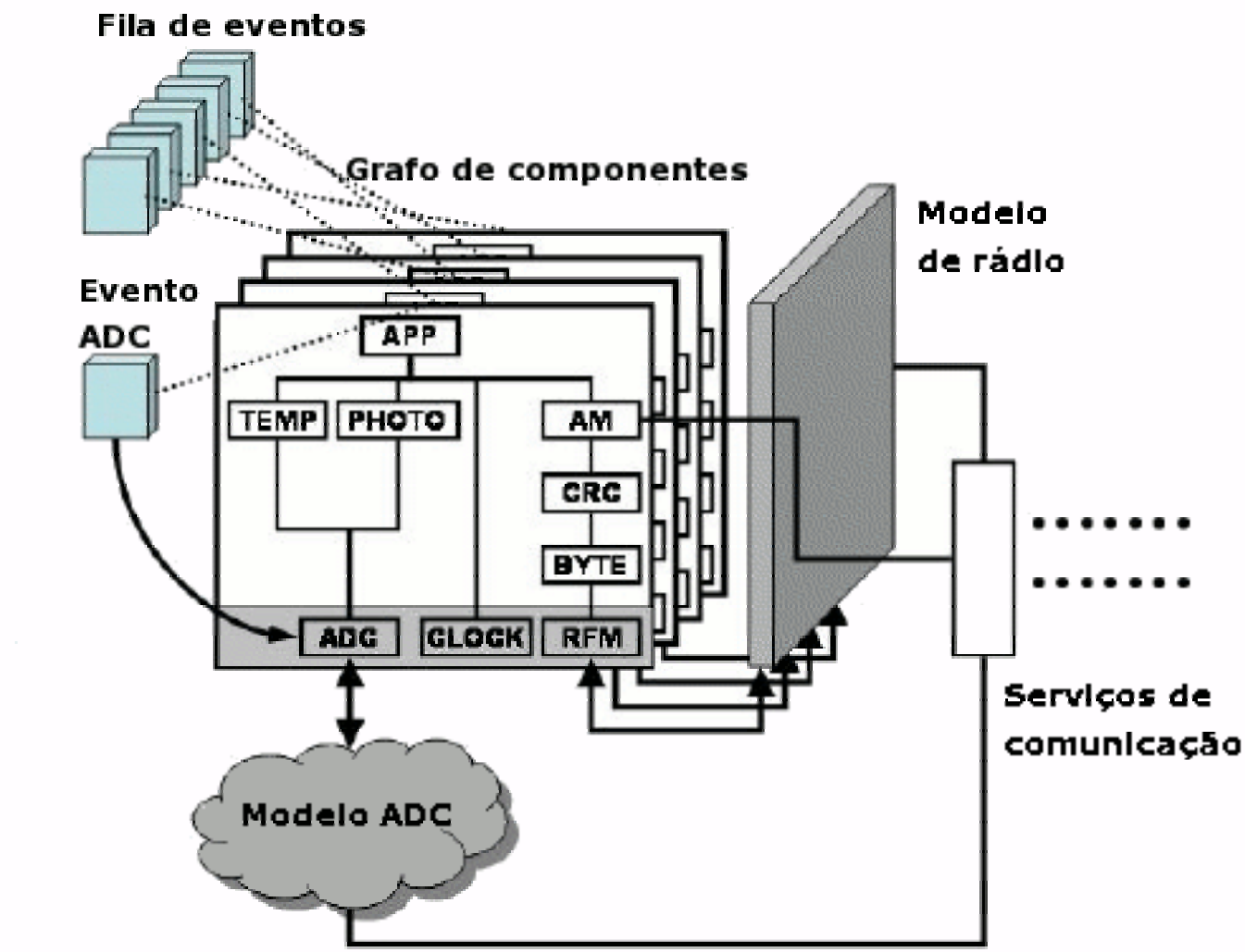
TOSSIM

- É o simulador do TinyOS.
- Compila diretamente código TinyOS.
- A simulação executa nativamente em um PC.
- Pode simular milhares de nós sensores simultaneamente. Cada nó na simulação executa o mesmo programa TinyOS.

Modelo de execução

- Simulador de eventos discretos.
- Eventos de simulação abstraem eventos de hardware.
- Possuem *timestamp* global.
- A cada evento de simulação:
  - Chama eventos ou comandos associados
  - Executa as tarefas na fila enquanto esta não estiver vazia.

Arquitetura TOSSIM



Abstrações do hardware

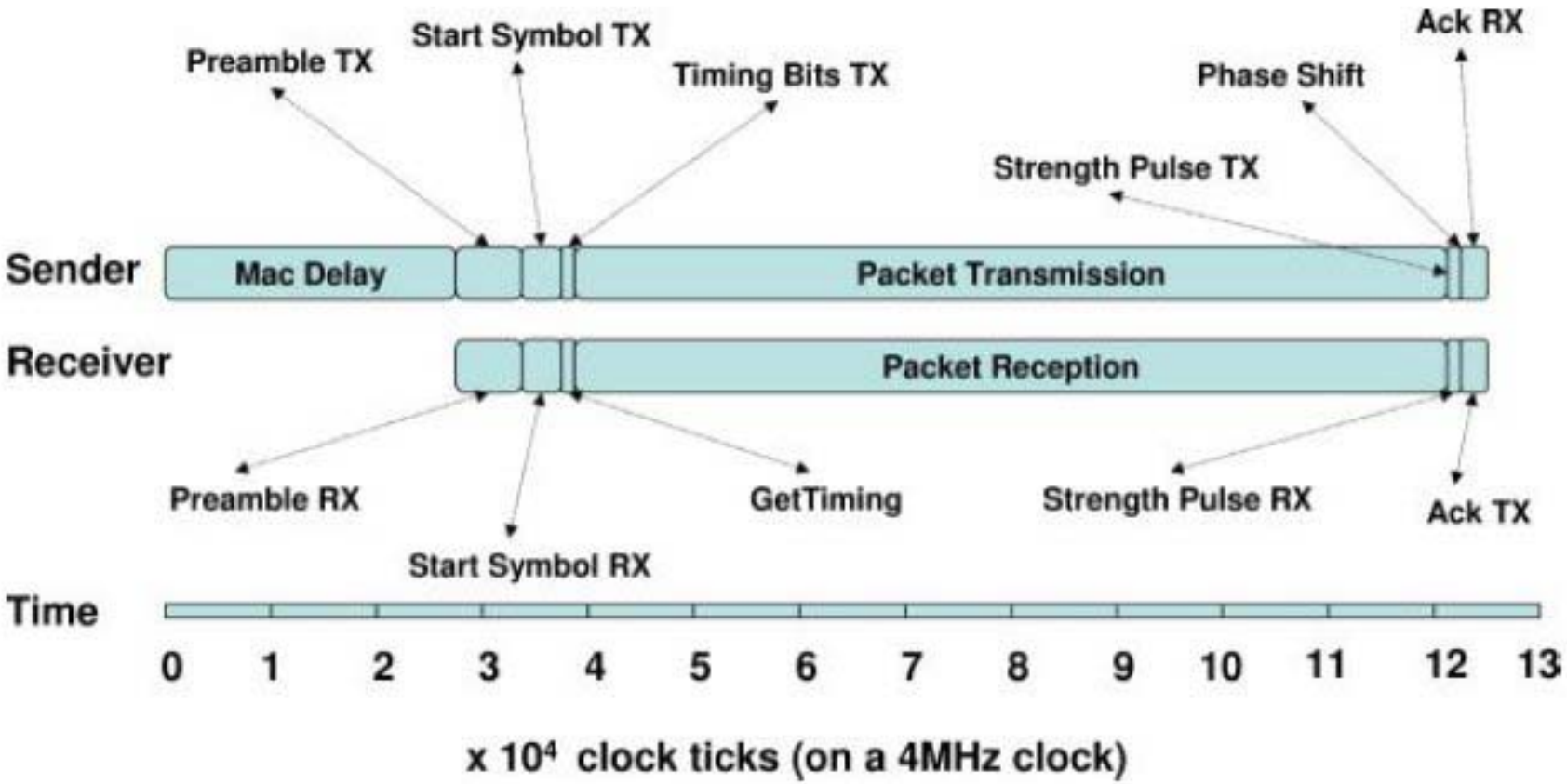
- Simulação em baixo nível:
  - Relógio
    - *Timestamp* corresponde a batidas do relógio do Mica mote (4MHz)
  - EEPROM
    - *Arquivo com conteúdo da memória*
  - Sensores (ADC)
  - Rádio
- Substituição apenas do componentes de baixo nível.

Simulação: ADC

- getData(): requisição ao ADC escalona um evento de simulação a 200 unidades de tempo.
- dataReady(): o evento de simulação dispara evento do TinyOS.
- Modelos de simulação:
  - Random: retorna um número aleatório de 10 bits.
  - Generic: permite atuação sobre o modelo aleatório.

Simulação: Rádio

- Pilha simulada: CSMA com CRC



Simulação: Rádio

- Simulação dos efeitos da comunicação sem fio, em nível de bits:
  - Corrupção de bit
    - Erro no *start symbol* previne recepção.
    - Erro no *acknowledgment* provoca falha na recepção.
  - Terminal escondido
  - Interferência de sinal

Simulação: Rádio

- Grafo de conexões:
  - Vértice: nó sensor
  - Aresta: conexão
  - Peso: probabilidade de erro.
- Simulação do terminal escondido:
  - a:b:0.5
  - b:c:0.4
  - Não há a:c

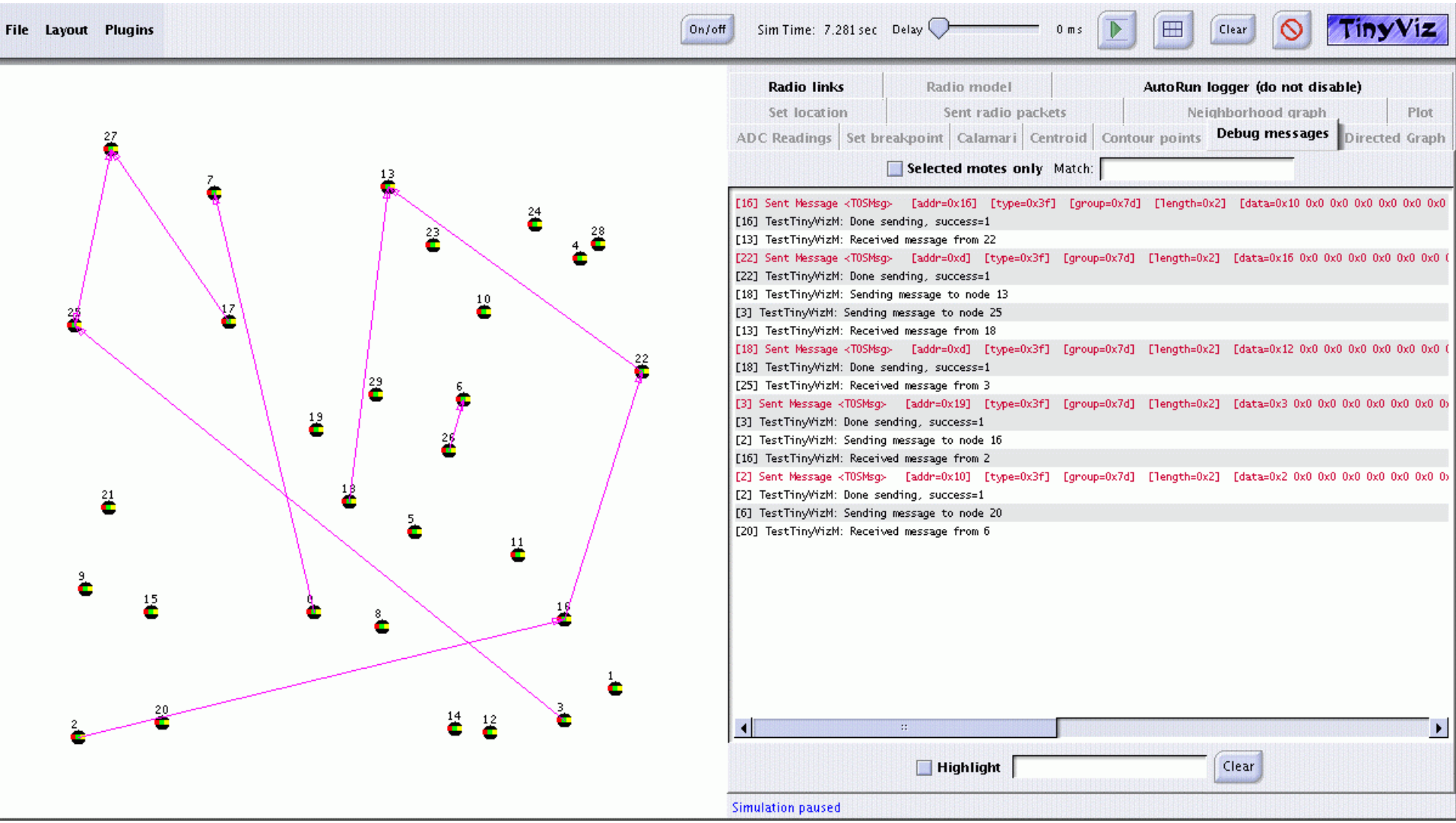
Atuação sobre a simulação

- TOSSIM pode comunicar com outras aplicações através de um soquete TCP.
  - Envia dados da simulação
  - Recebe comandos que atuam sobre a simulação:
    - Injeção de pacotes na rede
    - Alteração da topologia
    - Ligar / desligar nó sensor
    - Modificar valores retornados pelo ADC

TinyViz

- Aplicação em Java com interface gráfica que permite visualizar e controlar a simulação em tempo de execução, inspecionando mensagens de depuração, pacotes de rádio e UART, e outros.
- Provê mecanismos de atuação através de *plugins.*

TinyViz


File
Layout
Plugins
On/off
Sim Time: 7.281 sec
Delay
0 ms
Clear
TinyViz
Radio links
Radio model
AutoRun logger (do not disable)
Set location
Sent radio packets
Neighborhood graph
Plot
ADC Readings
Set breakpoint
Calamari
Centroid
Contour points
Debug messages
Directed Graph
Selected motes only
Match:
[16] Sent Message <TOSMsg> [addr=0x16] [type=0x3f] [group=0x7d] [length=0x2] [data=0x10 0x0 0x0 0x0 0x0 0x0 0x0
[16] TestTinyVizM: Done sending, success=1
[13] TestTinyVizM: Received message from 22
[22] Sent Message <TOSMsg> [addr=0xd] [type=0x3f] [group=0x7d] [length=0x2] [data=0x16 0x0 0x0 0x0 0x0 0x0 0x0
[22] TestTinyVizM: Done sending, success=1
[18] TestTinyVizM: Sending message to node 13
[3] TestTinyVizM: Sending message to node 25
[13] TestTinyVizM: Received message from 18
[18] Sent Message <TOSMsg> [addr=0xd] [type=0x3f] [group=0x7d] [length=0x2] [data=0x12 0x0 0x0 0x0 0x0 0x0 0x0
[18] TestTinyVizM: Done sending, success=1
[25] TestTinyVizM: Received message from 3
[3] Sent Message <TOSMsg> [addr=0x19] [type=0x3f] [group=0x7d] [length=0x2] [data=0x3 0x0 0x0 0x0 0x0 0x0 0x0
[3] TestTinyVizM: Done sending, success=1
[2] TestTinyVizM: Sending message to node 16
[16] TestTinyVizM: Received message from 2
[2] Sent Message <TOSMsg> [addr=0x10] [type=0x3f] [group=0x7d] [length=0x2] [data=0x2 0x0 0x0 0x0 0x0 0x0 0x0
[2] TestTinyVizM: Done sending, success=1
[6] TestTinyVizM: Sending message to node 20
[20] TestTinyVizM: Received message from 6
Highlight
Clear
Simulation paused

Executando o TOSSIM

- Entre no diretório da aplicação
- Execute make pc
- Entre no diretório build/pc
- Execute o programa main.exe
- Exemplo: simulação com três nós:
  - ./main.exe 3

Níveis de depuração

- Controlado pela variável de ambiente DBG
- Exemplo: visualizar pacotes e mudanças nos leds.
  - export DBG=am,led
- A ajuda do TOSSIM contém o conjunto de váriaveis suportadas:
  - ./main.exe -h

Mensagens da aplicação

- Variáveis DBG especiais: usr1,usr2,usr3.
- Comando dbg (similar ao printf de C):
  - dbg(DBG_CRC, "crc check failed: \%x, \%x\n", crc, mcrc);
- Variáveis dbg podem ser combinadas:
  - dbg(DBG_ROUTE|DBG_ERROR, "Received control message: lose our network name!.\n");

Geração de topologias

- LossyBuilder

usage: java net.tinyos.sim.LossyBuilder [options]

options:

-t grid: Topology (grid only and default)

-d <m> <n>: Grid size (m by n) (default: 10 x 10)

-s <scale>: Spacing factor (default: 5.0)

-o <file>: Output file

-i <file>: Input file of positions

-p: Generate positions, not error rates

Referência

- TOSSIM: A Simulator for TinyOS networks.
- www.cs.berkeley.edu/~pal/pubs/nido.pdf