Pontifícia Universidade Católica do Rio de Janeiro – PUC-RIO
Departamento de Informática

INF 2545 - SISTEMAS DISTRIBUÍDOS
Prof. Noemi Rodriguez

Redes de Sensores sem Fio + Terra

Adriano Branco

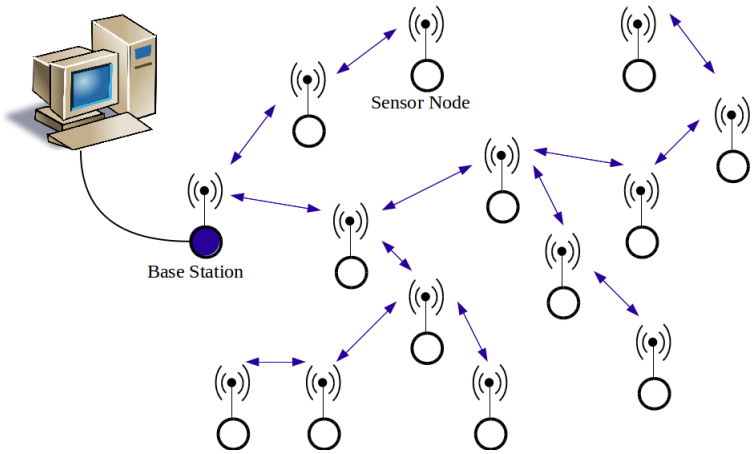
11 de maio de 2017

Motivação

Plataforma alvo: RSSFs – dispositivos limitados em recursos – memória, processamento e energia.

Exemplos:

- Mote MicaZ – Atmel ATmega128L, **4kB RAM** e 128kB ROM
- Mote TelosB – TI MSP430, **10kB RAM** e 48kB ROM
- Rádio IEEE 802.15.4, 2.4MHz, 30m indoor, **2 pilhas AA**.



Desafios

Programação: A programação em RSSF é difícil e propensa a erros.

- escassez de recursos
- orientada a eventos
- coordenação entre uma grande quantidade de nós

Reprogramação remota: Dificuldades para reprogramação dos nós sensores depois de distribuídos fisicamente.

Desafios

Programação: A programação em RSSF é difícil e propensa a erros.

- escassez de recursos
- orientada a eventos
- coordenação entre uma grande quantidade de nós

Reprogramação remota: Dificuldades para reprogramação dos nós sensores depois de distribuídos fisicamente.

Desafios

Programação: A programação em RSSF é difícil e propensa a erros.

- escassez de recursos
- orientada a eventos
- coordenação entre uma grande quantidade de nós

Reprogramação remota: Dificuldades para reprogramação dos nós sensores depois de distribuídos fisicamente.

Aplicações específicas

Normalmente, uma RSSF é projetada para uma única categoria de aplicação (hardware + software). Isso permite definir previamente o conjunto de operações necessárias e os respectivos componentes de software.

Modelos de Programação

Imperativo: Programação tradicional com callbacks – C

- Associa, no escopo global, eventos e procedimentos.
- Tendência a programação sequencial misturada com eventos assíncronos.

Orientado a Eventos: Comando, Eventos e Tarefas – nesC + C

- Extende C com um modelo de comandos e eventos. Abstrai os callbacks assíncronos.
- Posibilita o uso de um "scheduler" para controlar a execução de tarefas.

Reativo: Emite comandos e reage aos eventos – Céu + C

- Estrutura de programação apropriada para reagir aos eventos assíncronos.

Modelos de Programação

Imperativo: Programação tradicional com callbacks – C

- Associa, no escopo global, eventos e procedimentos.
- Tendência a programação sequencial misturada com eventos assíncronos.

Orientado a Eventos: Comando, Eventos e Tarefas – nesC + C

- Extende C com um modelo de comandos e eventos. Abstrai os callbacks assíncronos.
- Posibilita o uso de um "scheduler" para controlar a execução de tarefas.

Reativo: Emite comandos e reage aos eventos – Céu + C

- Estrutura de programação apropriada para reagir aos eventos assíncronos.

Modelos de Programação

Imperativo: Programação tradicional com callbacks – C

- Associa, no escopo global, eventos e procedimentos.
- Tendência a programação sequencial misturada com eventos assíncronos.

Orientado a Eventos: Comando, Eventos e Tarefas – nesC + C

- Extende C com um modelo de comandos e eventos. Abstrai os callbacks assíncronos.
- Posibilita o uso de um "scheduler" para controlar a execução de tarefas.

Reativo: Emite comandos e reage aos eventos – Céu + C

- Estrutura de programação apropriada para reagir aos eventos assíncronos.

Principais problemas na programação de RSSFs

Programação

- Complexidade de programação
  - Orientação a eventos
  - Laços infinitos (*starvation*)
  - Ponteiros inválidos
- Complexidade de rede
  - Acesso direto as funções do rádio.
  - Protocolos de comunicação para rede ad-hoc.

Custos adicionais

- Uso de CPU – Arquitetura de máquina virtual ou middleware (Energia e Tempo de execução)
- Uso do Rádio – Disseminação de código (Energia)

Principais problemas na programação de RSSFs

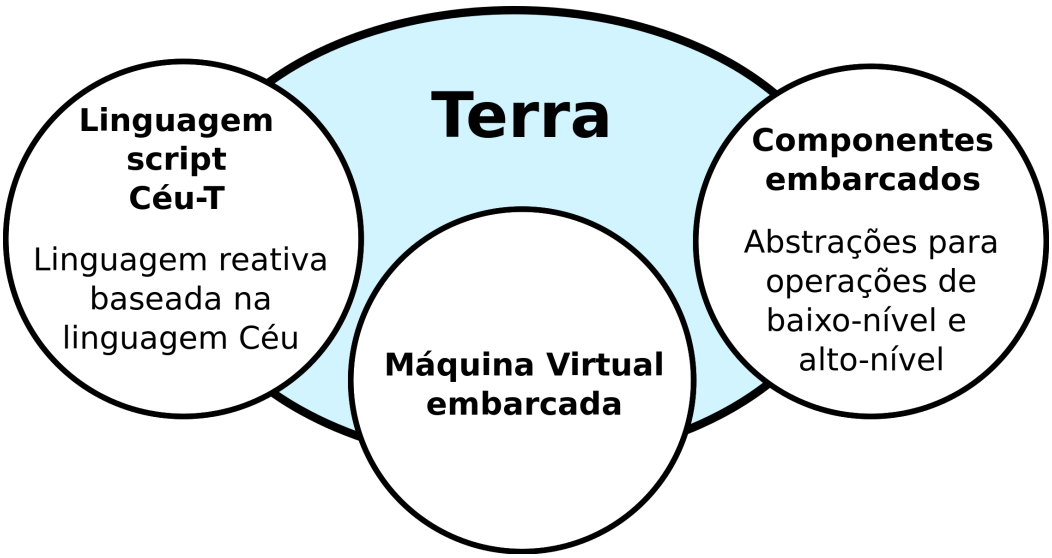
Programação

- Complexidade de programação
  - Orientação a eventos
  - Laços infinitos (*starvation*)
  - Ponteiros inválidos
- Complexidade de rede
  - Acesso direto as funções do rádio.
  - Protocolos de comunicação para rede ad-hoc.

Custos adicionais

- Uso de CPU – Arquitetura de máquina virtual ou middleware (Energia e Tempo de execução)
- Uso do Rádio – Disseminação de código (Energia)

Modelo proposto



Céu-T é baseada na linguagem Céu

- Céu é uma linguagem reativa.
- A análise estática de Céu em tempo de compilação permite garantias de segurança. (Situação de corrida e laços infinitos.)
- Construtores paralelos e comandos bloqueantes para aguardar eventos:

```
// Amostragem
loop do
  par/and do
    <...>
  with
    await 1s;
  end
end
```

```
// Time-out
loop do
  par/or do
    <...>
  with
    await 1s;
  end
end
```

- Céu é compilado para C e permite a inclusão de código C.

Céu-T é baseada na linguagem Céu

- Céu é uma linguagem reativa.
- A análise estática de Céu em tempo de compilação permite garantias de segurança. (Situação de corrida e laços infinitos.)
- Construtores paralelos e comandos bloqueantes para aguardar eventos:

```
// Amostragem
loop do
  par/and do
    <...>
  with
    await 1s;
  end
end
```

```
// Time-out
loop do
  par/or do
    <...>
  with
    await 1s;
  end
end
```

- Céu é compilado para C e permite a inclusão de código C.

Céu-T é baseada na linguagem Céu

- Céu é uma linguagem reativa.
- A análise estática de Céu em tempo de compilação permite garantias de segurança. (Situação de corrida e laços infinitos.)
- Construtores paralelos e comandos bloqueantes para aguardar eventos:

```
// Amostragem
loop do
  par/and do
    <...>
  with
    await 1s;
  end
end
```

```
// Time-out
loop do
  par/or do
    <...>
  with
    await 1s;
  end
end
```

- Céu é compilado para C e permite a inclusão de código C.

Céu-T é baseada na linguagem Céu

- Céu é uma linguagem reativa.
- A análise estática de Céu em tempo de compilação permite garantias de segurança. (Situação de corrida e laços infinitos.)
- Construtores paralelos e comandos bloqueantes para aguardar eventos:

```
// Amostragem
loop do
  par/and do
    <...>
  with
    await 1s;
  end
end
```

```
// Time-out
loop do
  par/or do
    <...>
  with
    await 1s;
  end
end
```

- Céu é compilado para C e permite a inclusão de código C.

Modelo de execução Céu

```
var short val;
par do
  loop do
    emit LED0(TOGGLE);
    await 1s;
  end
with
  loop do
    await 10s;
    emit REQ_TEMP();
    val = await TEMP();
    if var > 200 then
      emit LED1(ON);
    else
      emit LED1(OFF);
    end
  end
end
```

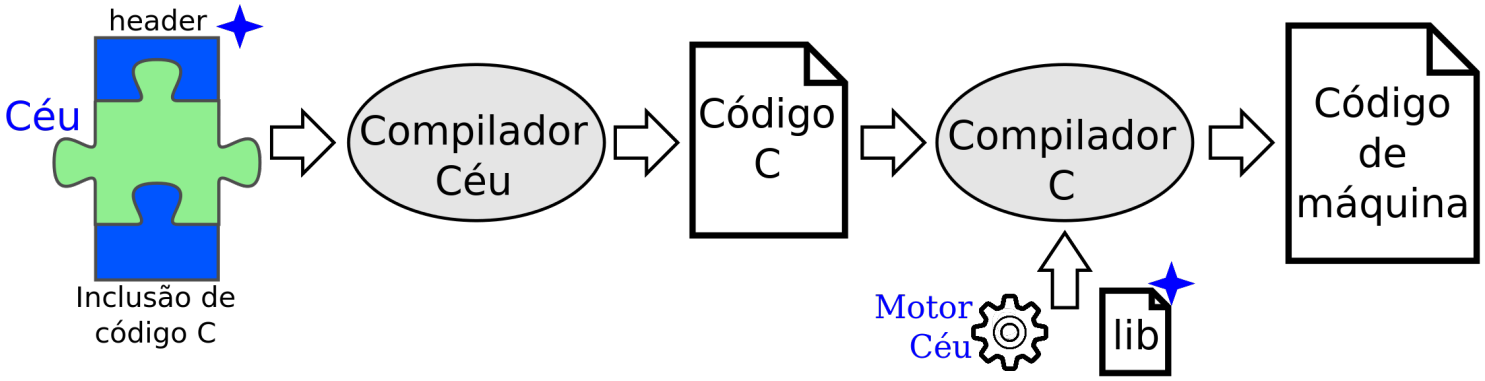
```
_WCLOCK1: Clk Counter; Label
_WCLOCK2: Clk Counter; Label

_TEMP: Evt ID; Label

Trail_1a:
  par do/ loop do/ await 1s
Trail_1b:
  end/ loop do/ await 1s;

Trail_2a:
  with/ loop do/ await 10s
Trail_2b:
  emit REQ_TEMP/ await TEMP
Trail_2c:
  if/ emit LED1/ end/ end/
  loop do/ await 10s
```

Modelo de execução Céu

```
var short val;
par do
  loop do
    emit LED0(TOGGLE);
    await 1s;
  end
with
  loop do
    await 10s;
    emit REQ_TEMP();
    val = await TEMP();
    if var > 200 then
      emit LED1(ON);
    else
      emit LED1(OFF);
    end
  end
end
```

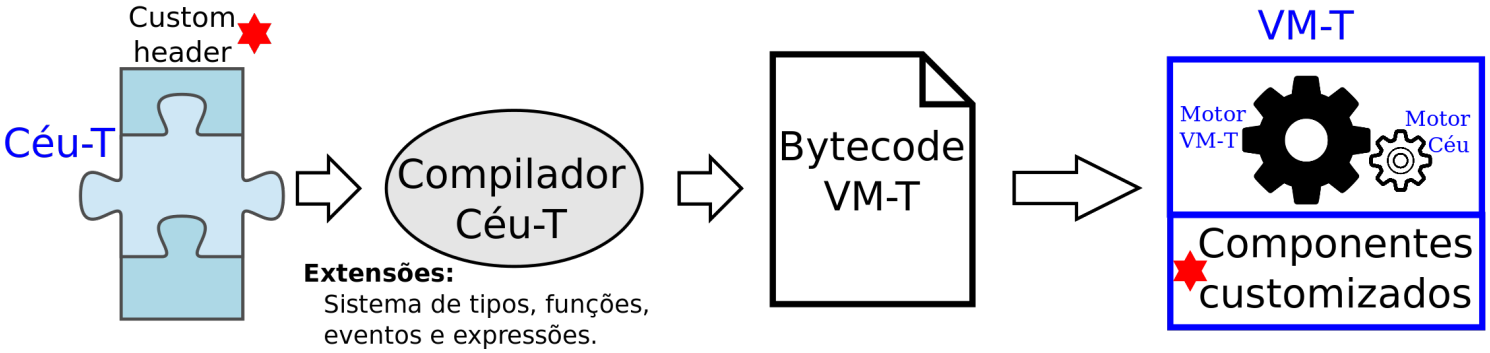
```
_WCLOCK1: Clk Counter; Label
_WCLOCK2: Clk Counter; Label

_TEMP: Evt ID; Label

Trail_1a:
  par do/ loop do/ await 1s
Trail_1b:
  end/ loop do/ await 1s;

Trail_2a:
  with/ loop do/ await 10s
Trail_2b:
  emit REQ_TEMP/ await TEMP
Trail_2c:
  if/ emit LED1/ end/ end/
  loop do/ await 10s
```

Terra *x* Céu



Céu

Terra



Customizações de Terra implementadas atualmente

TerraNet é uma customização usando apenas comunicações simples de rádio e operações de sensores.

TerraGrp usa a biblioteca básica de componentes, que inclui as abstrações de alto nível para controle de grupos de nós.

TerraIno é uma customização para Arduino Mega.

TerraIx é uma customização Linux (RaspberryPI).

Customizações de Terra implementadas atualmente

TerraNet é uma customização usando apenas comunicações simples de rádio e operações de sensores.

TerraGrp usa a biblioteca básica de componentes, que inclui as abstrações de alto nível para controle de grupos de nós.

TerraIno é uma customização para Arduino Mega.

TerraIx é uma customização Linux (RaspberryPI).

Customizações de Terra implementadas atualmente

TerraNet é uma customização usando apenas comunicações simples de rádio e operações de sensores.

TerraGrp usa a biblioteca básica de componentes, que inclui as abstrações de alto nível para controle de grupos de nós.

TerraIno é uma customização para Arduino Mega.

TerraIx é uma customização Linux (RaspberryPI).

Customizações de Terra implementadas atualmente

TerraNet é uma customização usando apenas comunicações simples de rádio e operações de sensores.

TerraGrp usa a biblioteca básica de componentes, que inclui as abstrações de alto nível para controle de grupos de nós.

TerraIno é uma customização para Arduino Mega.

TerraIx é uma customização Linux (RaspberryPI).

App #1a - Monitor e alarme multi-salto

Pseudocódigo em nesC

```
// first part - Build ad-hoc tree
    event booted
        start radio
    event radio started
        if root node then
            broadcast discover message
    event discover message
        if hasParent is false then
            hasParent = true
            broadcast discover message
            start data periodic timer
            start alarm periodic timer
            start monitoring periodic timer
// second part - monitoring functionality
    event alarm timer
        read sensor
    event sensor done
        if is an alarm and alarmMute==false
            alarmMute=true
            start mute timer
            send alarm message to parent node
    event mute timer
        alarmMute = false
    event data timer
        send data message to parent node
    event data message
        send data message to parent node
// empty events must be declared
    event discover message send done
    event data message send done
    event radio stopped
```

Pseudocódigo em Céu-T

```
// first part - Build ad-hoc tree
  if is root node then
    broadcast discover message
  else
    wait to receive a discover message
    broadcast discover message
  end
// second part - monitoring functionality
  do -- in parallel
    loop
      read sensor
  with
    loop
      if is an alarm then
        send alarm to parent node
        wait mute time
  with
    loop
      send sensor to parent node
  with
    loop
      wait sensor or alarm messages
      send message to parent node
  end
```

App #1a - Monitor e alarme multi-salto

Pseudocódigo em nesC

```
// first part - Build ad-hoc tree
    event booted
    event data message
        send data message to parent node
// empty events must be declared
    event discover message send done
    event data message send done
    event radio stopped
```

Pseudocódigo em Céu-T

```
// first part - Build ad-hoc tree
  if is root node then
  with
    loop
      wait sensor or alarm messages
      send message to parent node
  end
```

Operação	nesC	Céu-T
Aguardar evento	**event** void comp.callback(type var){ <*bloco de código*> }	var = **await** EVT_IN();
Chamada externa	var = **call** comp.func();	**emit** EVT_OUT();
		var = func();

Split-phase operation:

emit EVT_OUT();
var = **await** EVT_IN();

App #1a - Monitor e alarme multi-salto

Pseudocódigo em nesC

```
// first part - Build ad-hoc tree
    event booted
        start radio
    event radio started
        if root node then
            broadcast discover message
    event discover message
        if hasParent is false then
            hasParent = true
            broadcast discover message
            start data periodic timer
            start alarm periodic timer
            start
// second part -
    event alarm t
        read sens
    event sensor
        if is an
            alarm
            start
            send
    event mute ti
        alarmMute
    event data ti
        send data message to parent node
    event data message
        send data message to parent node
// empty events must be declared
    event discover message send done
    event data message send done
    event radio stopped
```

```
event booted
    start radio
event radio started
    if root node th
        broadcast c
event discover mess
    if hasParent is
        hasParent =
```

Pseudocódigo em Céu-T

```
// first part - Build ad-hoc tree
  if is root node then
    broadcast discover message
  else
    wait to receive a discover message
    broadcast discover message
  end
// second part - monitoring functionality
  do -- in parallel
    loop
      read sensor
  with
    loop
      if is an alarm then
        send alarm to parent node
        wait mute time
  with
    loop
      send sensor to parent node
  with
    loop
      wait sensor or alarm messages
      send message to parent node
  end
```

App #1a - Monitor e alarme multi-salto

Pseudocódigo em nesC

```
// first part - Build ad-hoc tree
    event booted
        start radio
    event radio started
        if root node then
            broadcast discover message
    event discover message
        if hasParent is false then
            hasParent = true
//
            send alarm message to parent node
    event mute timer
        alarmMute = false
    event data timer
        send data message to parent node
    event data message
        send data message to parent node
// empty events must be declared
    event discover message send done
    event data message send done
    event radio stopped
```

Contando eventos e chamadas

- 16 eventos (nesC events)
- 24 chamadas (nesC calls)
- Relação **complexa** entre eventos e variáveis globais.

Pseudocódigo em Céu-T

```
// first part - Build ad-hoc tree
  if is root node then
    broadcast discover message
  else
    wait to receive a discover message
    broadcast discover message
  end
//
      wait mute time
  with
    loop
      send sensor to parent node
  with
    loop
      wait sensor or alarm messages
      send message to parent node
  end
```

Contando awaits and emits+waits

- 12 awaits
- 5 emits+awaits (split-phase oper.)
- Relação **simplificada** entre eventos e variáveis globais.

App #1b - Monitor e alarme multi-salto

Pseudocódigo em nesC

```
: first part - monitoring functionality
    event booted
        start radio
    event radio started
        if root node
            setRoot
        else
            start data periodic timer
            start alarm periodic timer
        end
    event alarm timers
        read sensor
    event sensor done
        if is an alarm and alarmMute==false
            alarmMute=true
            start mute timer
            send alarm message
    event mute timer
        alarmMute = false
    event data timer
        send data message to parent node
: empty events must be declared
    event data message send done
    event radio stopped
```

Código Céu-T

```
par do
/*****************************************************
 * Monitoring temperature and alarm
 *****************************************************/
    loop do
        emit REQ_TEMP();
        gblTemp = await TEMP;
        await 500ms;
    end
with
    loop do
        if gblTemp > ALARM then
            inc msgAlarm.d8[2];
            msgAlarm.d16[0] = gblTemp;
            emit SEND_BS(msgAlarm);
            await SENDBS_DONE(ID_ALARM);
            await 30s;
        else
            await 500ms;
        end
    end
with
/*****************************************************
 * Send temperature value
 *****************************************************/
    await 500ms; // Waits to read first sensor value
    loop do
        par/or do
            await 60s;
        with
            msgData.d16[0] = gblTemp;
            inc msgData.d8[2];
            await (nodeId*500)ms;
            emit SEND_BS(msgData);
            await SENDBS_DONE(ID_DATA);
            await FOREVER;
        end
    end
end
```

App #1a & #1b - Monitor e alarme multi-salto

Métrica	Terra		TinyOS	
	#1a –	#1b CTP	#1a –	#1b CTP
Linhas de código	199	63	307	224
Bytecode (bytes)	650	189		
Código de máquina (bytes)	40,000	55,300	15,566	21,548
Blocos de disseminação	28	9	649	898

App #1a & #1b - Monitor e alarme multi-salto

Métrica	Terra		TinyOS	
	#1a –	#1b CTP	#1a –	#1b CTP
Linhas de código	199	63	307	224
Bytecode (bytes)	650	189		
Código de máquina (bytes)	40,000	55,300	15,566	21,548
Blocos de disseminação	28	9	649	898

App #1a: TinyOS / Terra

Linhas de código: $1.54x \rightarrow$ algoritmo de rede

Disseminação: $23.18x$

App #1b: TinyOS / Terra

Linhas de código: $3.5x \rightarrow$ componente embarcado

Disseminação: $99.8x$

App #1a & #1b - Monitor e alarme multi-salto

Métrica	Terra		TinyOS	
	#1a –	#1b CTP	#1a –	#1b CTP
Linhas de código	199	63	307	224
Bytecode (bytes)	650	189		
Código de máquina (bytes)	40,000	55,300	15,566	21,548
Blocos de disseminação	28	9	649	898

App Terra: #1a / #1b

Linhas de código: $3.15x \rightarrow$ algoritmo de rede x componente embarcado

Disseminação: $3.1x$

Observações adicionais

Curva de aprendizado

- Experiência em sala de aula.

Starvation

- Em Céu, laços infinitos devem conter pelo menos um await.(quebrado em trilhas)
- A VM de Terra executa cada trilha como uma tarefa do TinyOS.

Ponteiros inválidos

- Em Céu-T todos endereços de memória são estáticos.
- Não existe tipo ponteiro em Céu-T.
- Índices de vetores são verificados na compilação e na execução.

Terra: Programação flexível e segura em RSSF

Adriano Branco