



Universidade de Fortaleza - Unifor

Simulando Algoritmos de Redes com Sinalgo

Marcus Vinícius de Sousa Lemos
<http://www.marvinlemos.net>



O que é o Sinalgo ?

- Framework para simulação e validação de algoritmos de rede
- Diferentemente de outros simuladores, concentra-se na verificação dos algoritmos, abstraindo-se das camadas mais baixas
- Desenvolvido em Java
- <http://dcg.ethz.ch/projects/sinalgo/>



Algumas características

- Prototipagem rápida dos algoritmos de rede em Java
- Alta performance – Executa simulações com 100000 nós em tempo aceitável
- Suporte a 2D e 3D
- Simulações Síncronas e Assíncronas
- Fácil customização dos gráficos de rede.
- Independente de plataforma – escrito em Java
- Sinalgo é livre, publicado sob a licença *BSD*



Instalando o Sinalgo

- A instalação é bastante fácil, consistindo basicamente de um projeto Java que pode ser importado em qualquer IDE Java.
- Nos nossos exemplos, utilizaremos a IDE Eclipse
- Sinalgo pode ser baixado do seguinte endereço:
 - <http://dcg.ethz.ch/projects/sinalgo/download.html>
 - Obs: Deve ser baixado a versão “Regular Release”
- Descompacte o arquivo baixado em qualquer diretório do seu sistema operacional.

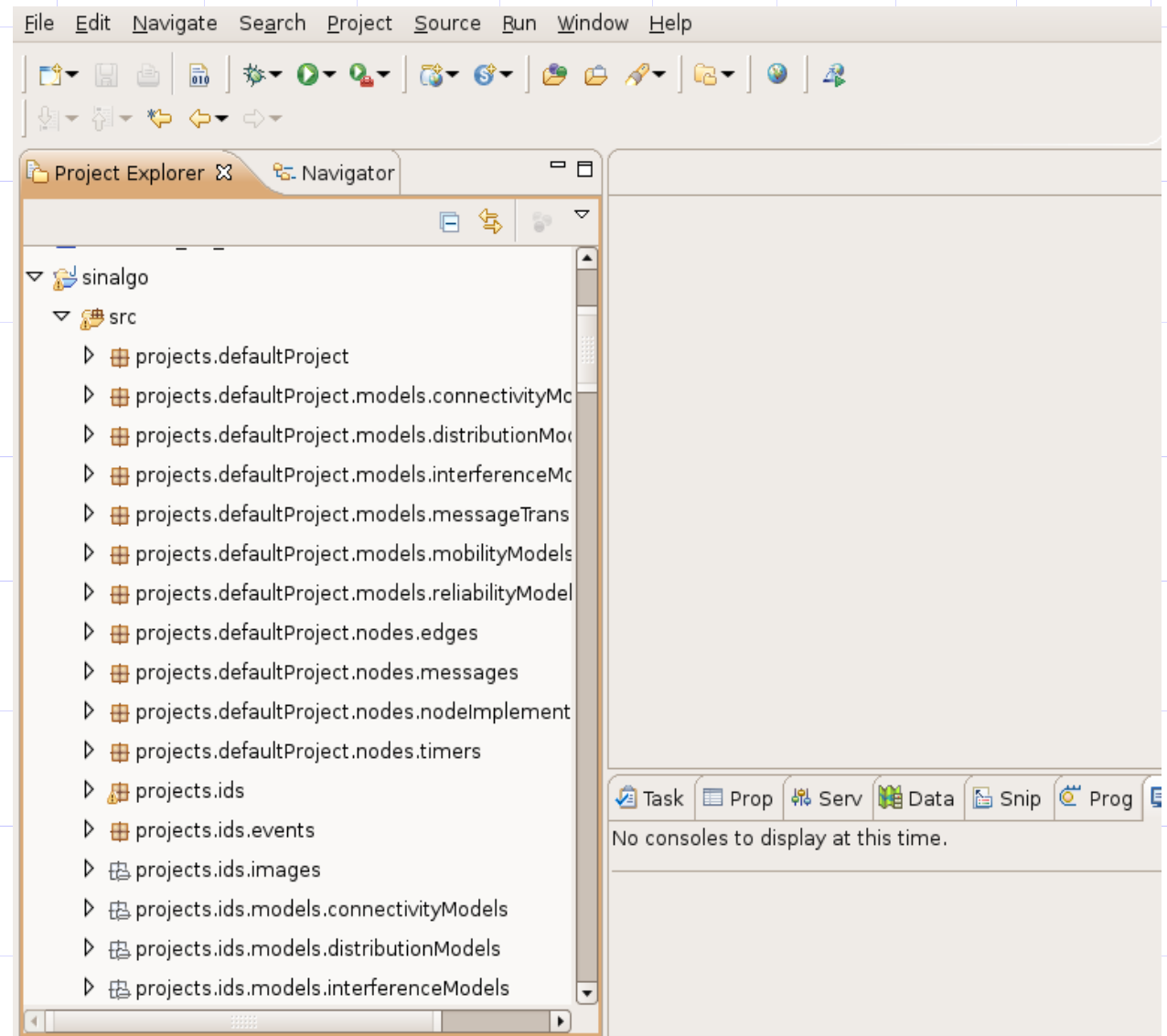


Importando o Sinalgo no Eclipse

- Inicie o Eclipse e crie um novo projeto
 - (File → New → Project)
- Na tela de criação de projeto, selecione a opção “Java Project” e clique para continuar
- Nomeie o projeto de **Sinalgo**
- Na mesma tela, marque a opção “**Create project from existing source**” e selecione o diretório onde o Sinalgo foi descompactado.
- Clique “Finish” para criar o projeto

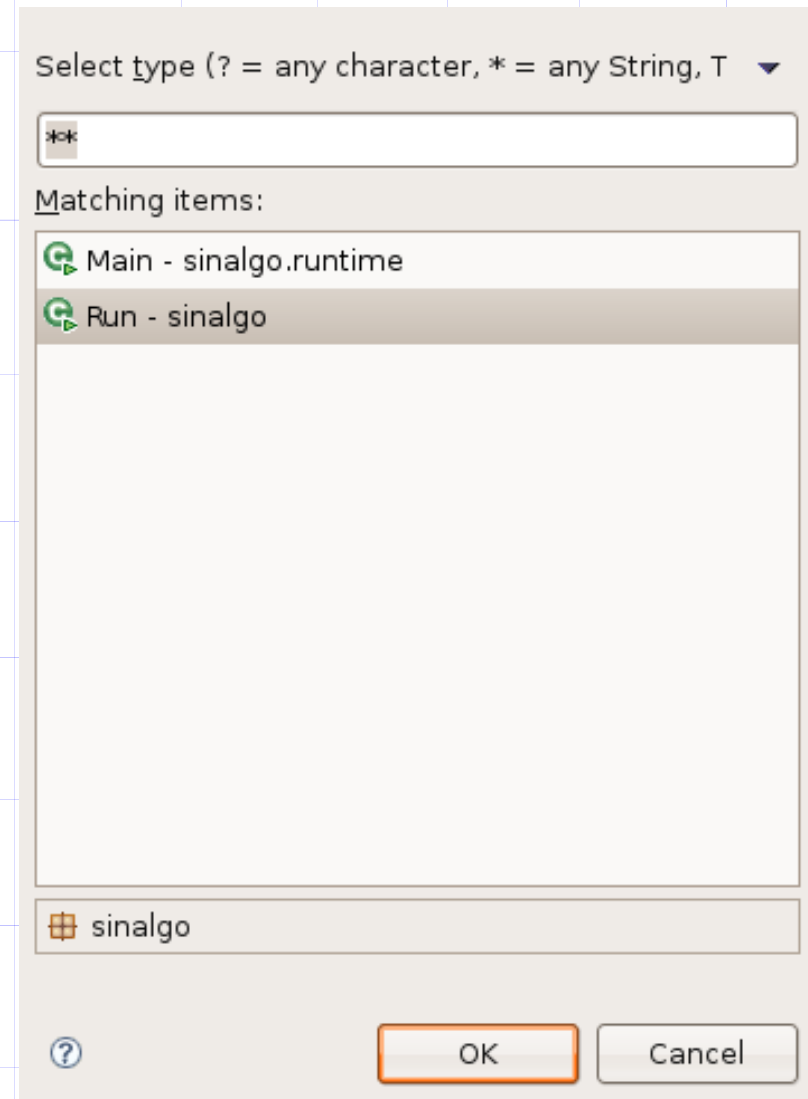
Importando o Sinalgo no Eclipse

Se tudo ocorreu bem, o projeto deve ter sido importado conforme ilustra a figura ao lado:



Executando o Sinalgo

- Para executar o sinalgo, clique com o botão direito na pasta “src” dentro da aba “Project Explorer” ou “Navigator” do Eclipse, e clique na opção “Run As” → “Java Application”
- Na tela “Select Java Application”, selecione a classe “Run”, conforme ilustra a figura ao lado, e clique em “OK”:



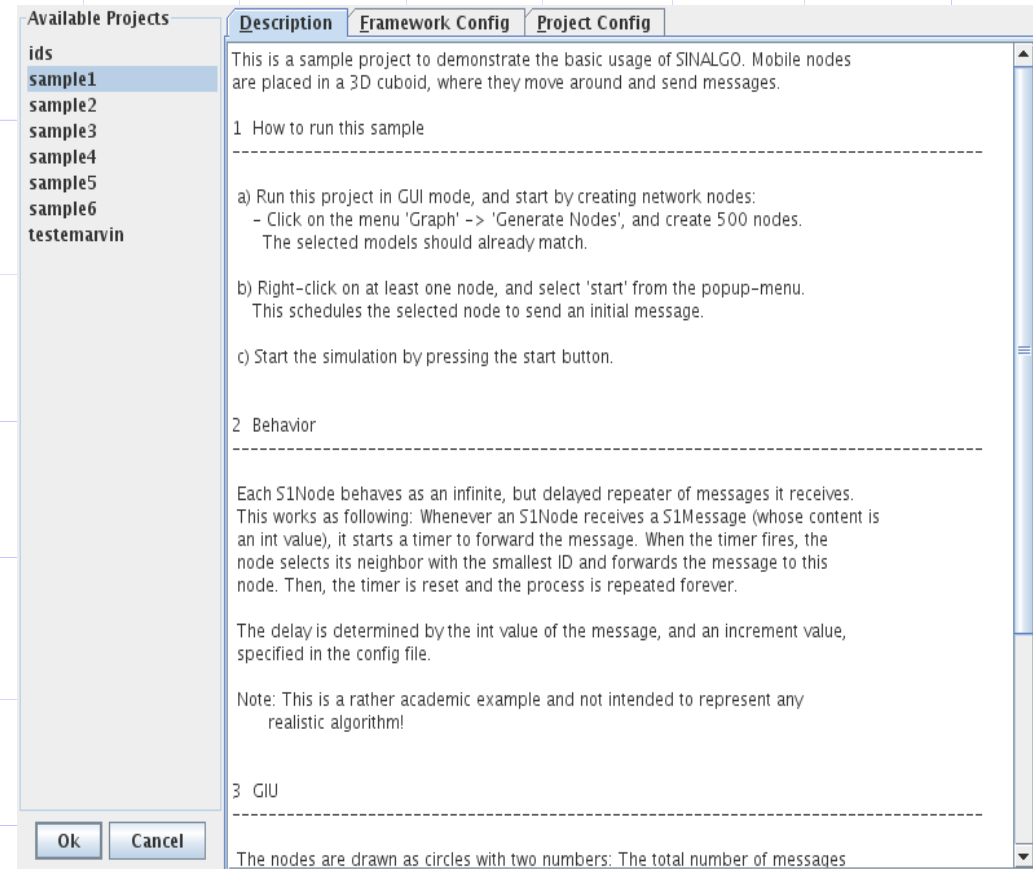


Executando o Sinalgo

- No sinalgo, cada cenário (algoritmo) que você deseja simular corresponde a um sub-pacote do pacote “projects”.
- Sinalgo inclui 6 projetos de exemplos que podemos utilizar para testar suas funcionalidades.
- Além disso, o código-fonte desses projetos são bem documentados, permitindo, assim, utilizarmos como fonte de consulta para a construção dos nossos próprios algoritmos

Executando o Sinalgo

- Ao executarmos o **Sinalgo**, teremos, à nossa disposição, uma tela onde poderemos escolher qual **projeto/cenário/algoritmo** desejamos simular.





Executando o Sinalgo

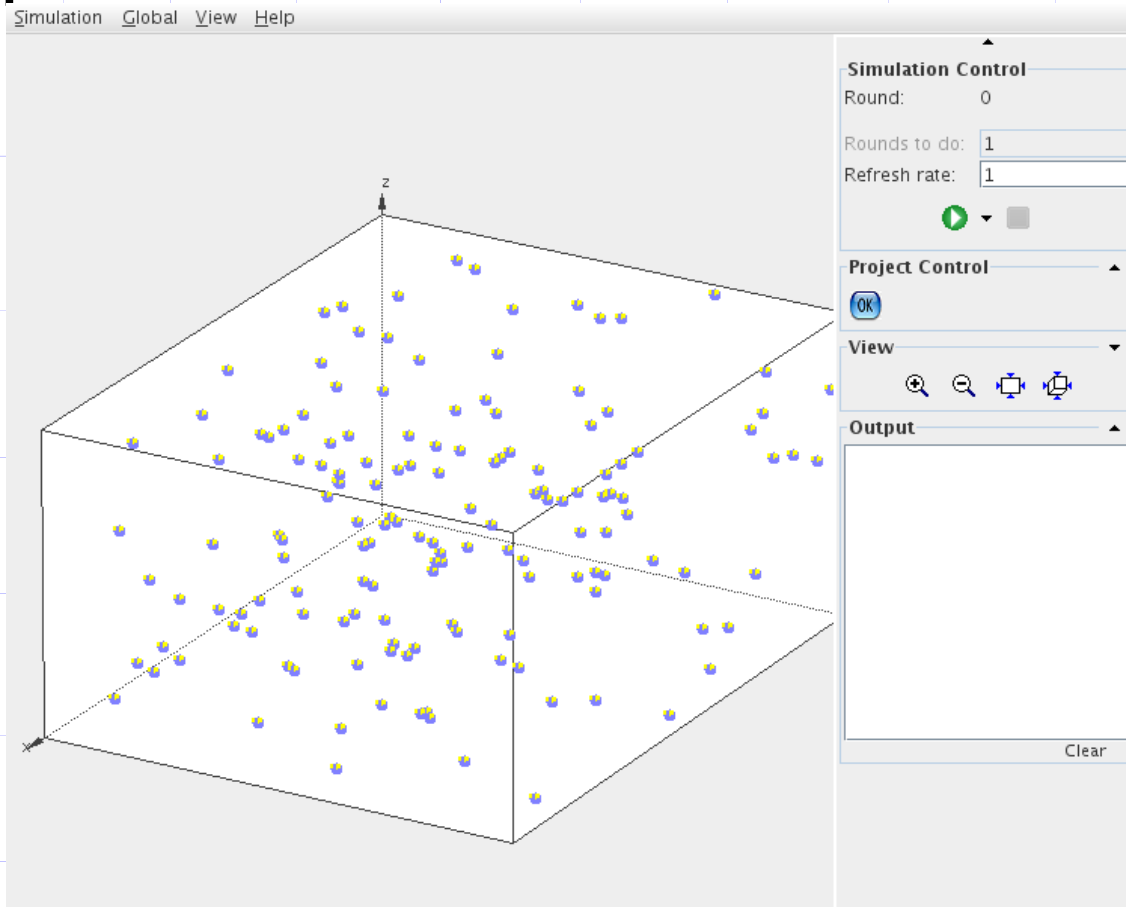
- Vamos simular o primeiro exemplo (sample1):
 - Ao selecionar o projeto, a área “Description” irá mostrar, como o próprio nome sugere, uma descrição do projeto, além de fornecer outras informações úteis, como os procedimentos para realizar a simulação e o comportamento esperado do ambiente simulado.
 - Depois de ter lido a descrição do projeto, clique no botão OK.




Executando o Sinalgo

- A próxima tela corresponde ao ambiente de simulação e é específico do projeto.
- Nosso próximo passo é criar os nós da nossa simulação:
 - Clique no menu “Simulation” → “Generate Node”
 - Na tela seguinte, selecione o número de nós e o modelo de Distribuição. As outras propriedades dos nós pode manter seus valores padrões.
 - Para mais informações sobre cada um desses modelos, refira-se ao Tutorial disponibilizado no site oficial.

Executando o Sinalgo



Ambiente criado com 150 nós

- Para iniciar a simulação, clique no botão “Run”: 
- **Obs:** Antes de começar a implementar seu próprio projeto, sugiro que estude os outros exemplos. Irá ajudar bastante na hora de construir seu cenário.



Criando uma simulação

- Uma vez instalado o Sinalgo, é possível executar várias simulações independentes com a mesma instalação.
 - Para distinguir várias simulações diferentes, os arquivos pertencentes a uma simulação são agrupados em um projeto.
- **Obs:** É altamente recomendado gerar um projeto para cada algoritmo simulado. Contudo, isso frequentemente resulta em código comum. Ao invés de copiar o mesmo código para todos os projetos, é preferível criar um novo projeto que armazenará esse código-comum e do qual todos os outros projetos terão acesso.



Criando uma simulação

- Da perspectiva do desenvolvedor, um projeto é apenas um diretório localizado dentro do diretório “src/projects” do Sinalgo.
- O nome do projeto é dado pelo nome do diretório criado.
- Um projeto do Sinalgo tem uma estrutura básica pré-definida. Assim, ao invés de criarmos essa estrutura manualmente, podemos simplesmente fazer uma cópia de um projeto “esqueleto” localizado dentro de “src/projects”



Criando uma simulação

- Dessa forma, abra o navegador de arquivos do seu sistema operacional e localize o diretório “src/projects” dentro do diretório onde o Sinalgo foi instalado
- Faça uma cópia do diretório “template” para o mesmo diretório corrente (src/projects) mudando apenas o nome do diretório.
- Neste nosso tutorial, iremos definir o nome do diretório para **wsn**



Criando uma simulação

- Iremos simular uma rede de sensores, onde o nó estação-base irá gerar um *flood* na rede para montar uma estrutura de roteamento em árvore.
- Retorne para o eclipse e, na aba “Project Explorer”, clique com o botão direito no diretório “src” e clique na opção “Refresh”.
- O pacote “projects.wsn” estará visível a partir desse momento



Criando uma simulação

- Primeiramente, deve-se corrigir os “imports” das classes “projects.wsn.CustomGlobal.java” e “projects.wsn.LogL.java”
- Abra cada um desses arquivos e altere a linha “**package projects.template;**” para “**package projects.wsn;**”



Criando uma simulação

- Cada nó simulado consiste em uma instância de uma sub-classe da classe “**sinalgo.nodes.Node**”
- Assim como na realidade, cada nó implementa seu próprio comportamento. Entre outros, os nós possuem um método que é chamado quando uma mensagem é recebida, um método usado para enviar mensagens para os nós vizinhos, etc.
- Para implementar o comportamento de um nó, deve-se criar uma classe que herde de “**sinalgo.nodes.Node**” e salvar o arquivo dentro da pasta **node/nodeImplementation** do projeto **wns**
- Crie uma classe chamada **SimpleNode.java**

Criando uma simulação

Java Class
Create a new Java class.

Source folder: sinalgo/src Browse...

Package: projects.wsn.nodes.nodeImplementations Browse...

☐ Enclosing type: Browse...

Name: SimpleNode

Modifiers: ☒ public ☐ default ☐ private ☐ protected
☐ abstract ☐ final ☐ static

Superclass: sinalgo.nodes.Node Browse...

Interfaces: Add... Remove

Which method stubs would you like to create?

☐ public static void main(String[] args)
☐ Constructors from superclass
☒ Inherited abstract methods

Do you want to add comments? (Configure templates and default value [here](#))
☐ Generate comments

Finish Cancel

Criando a classe SimpleNode no Eclipse



Criando uma simulação

- Obrigatoriamente, deve-se implementar o método **handleMessages()** e, opcionalmente, qualquer um dos métodos abstratos da classe **sinalgo.nodes.Node**
 - Refira-se ao tutorial do Sinalgo para mais informações sobre os métodos abstratos da classe **sinalgo.nodes.Node**



Criando uma simulação

- Antes de implementarmos o método “handleMessages”, devemos definir nossa classe de **Mensagem**.
- Nós comunicam-se através de mensagens (messages). Para implementar sua classe de mensagem, deve-se criar uma nova classe que herde de **sinalgo.nodes.messages.Message** e salvar o arquivo dentro do diretório “nodes/messages” do diretório do projeto **wsn**.
- Crie uma classe chamada **WsnMsg.java**”



Criando uma simulação

- A classe abstrata Message requer que seja implementada apenas um único método:
 - **public Message clone();**
- Obs: Quando um nó envia uma mensagem para um nó vizinho, assume-se que o destino recebe o conteúdo da mensagem que foi enviado pelo método **send()**. O *framework*, contudo, não tem como saber se o transmissor ainda tem uma referência para o objeto de mensagem enviado, e conseqüentemente, ele pode alterar a mensagem enviada. Para enviar tais problemas, o framework envia cópias separadas para todos os nós que recebem uma mensagem.

Criando uma simulação

- **Conteúdo da classe WsnMsg.java:**

```
public class WsnMsg extends Message {  
    //Identificador da mensagem  
    public Integer sequenceID;  
  
    //Tempo de vida do Pacote  
    public Integer ttl;  
  
    //Nó de destino  
    public Node destino;  
  
    //Nó de origem  
    public Node origem;  
  
    //No que vai reencaminhar a mensagem  
    public Node forwardingHop;
```

Criando uma simulação

- **Conteúdo da classe WsnMsg.java:**

//Número de saltos até o destino

public Integer saltosAteDestino;

//Tipo do Pacote. 0 para Estabelecimento de Rotas e 1 para pacotes de dados

public Integer tipoMsg = 0;

//Construtor da Classe

```
public WsnMsg(Integer seqID, Node origem, Node destino, Node forwardingHop, Integer tipo){  
    this.sequenceID = seqID;  
    this.origem = origem;  
    this.destino = destino;  
    this.forwardingHop = forwardingHop;  
    this.tipoMsg = tipo;  
}
```




Criando uma simulação

- **Conteúdo da classe WsnMsg.java:**

```
@Override
public Message clone() {
    WsnMsg msg = new WsnMsg(this.sequenceID, this.origem,
this.destino, this.forwardingHop, this.tipoMsg);
    msg.ttl = this.ttl;
    msg.saltosAteDestino = saltosAteDestino;
    return msg;
}
}
```



Criando uma simulação

- A classe é auto-explicativa
- Basicamente definimos os campos da mensagem e a implementação do método **clone()**, que será usado para gerar uma cópia da mensagem.



Criando uma simulação

- Agora, podemos voltar à implementação da classe **SimpleNode**.
- O método mais importante é o `handleMessage()`.
- Esse método é usado para tratar as mensagens que o nó recebeu.
- Cada nó armazena as mensagens que recebe em uma instância da classe **Inbox**.
- **Inbox** provê um iterador para acessar cada mensagem armazenada.
- Para cada mensagem, o iterador armazena meta-informações, como o transmissor (sender) da mensagem.

Criando uma simulação

- **Classe SimpleNode.java:**

```
//Armazenar o nó que sera usado para alcançar a Estacao-Base  
private Node proximoNoAteEstacaoBase;
```

```
//Armazena o número de sequencia da última mensagem recebida  
private Integer sequenceNumber = 0;
```

```
@Override  
public void handleMessages(Inbox inbox) {  
    while (inbox.hasNext()){  
        Message message = inbox.next();  
        if (message instanceof WsnMsg){  
            Boolean encaminhar = Boolean.TRUE;  
            WsnMsg wsnMessage = (WsnMsg) message;  
            if (wsnMessage.forwardingHop.equals(this)) { // A mensagem voltou.  
                O no deve descarta-la  
                encaminhar = Boolean.FALSE;  
            } else if (wsnMessage.tipoMsg == 0) { // A mensagem é um flood.  
                Devemos atualizar a rota
```

Criando uma simulação

- **Classe SimpleNode.java:**

```
if (proximoNoAteEstacaoBase == null){
    proximoNoAteEstacaoBase = inbox.getSender();
    sequenceNumber = wsnMessage.sequenceID;
}else if (sequenceNumber < wsnMessage.sequenceID){
    //Recurso simples para evitar loop.
    //Exemplo: Noh A transmite em broadcast. Noh B recebe a
    //msg e retransmite em broadcast.
    //Consequentemente, noh A irá receber a msg. Sem esse
    //condicional, noh A iria retransmitir novamente, gerando um loop
    sequenceNumber = wsnMessage.sequenceID;
}else{
    encaminhar = Boolean.FALSE;
}
}
```



Criando uma simulação

```
if (encaminhar){  
    //Devemos alterar o campo forwardingHop(da  
    //mensagem) para armazenar o  
    //noh que vai encaminhar a mensagem.  
    wsnMessage.forwardingHop = this;  
    broadcast(wsnMessage);  
}
```

```
}
```

```
}
```

```
}
```



Criando uma simulação

- A classe **SimpleNode** já é capaz de reencaminhar as mensagens (de estabelecimento de rotas) que receber. Perceba que ele retransmite as mensagens para os nós vizinhos através de broadcast:
 - **broadcast(wsnMessage);**
- Contudo, antes da mensagem ser reencaminhado, o nó armazena, no campo **forwardingHop**, uma referência a ele mesmo:
 - **wsnMessage.forwardingHop = this;**
- Dessa forma, o nó vizinho que receber a mensagem irá saber quem a transmitiu



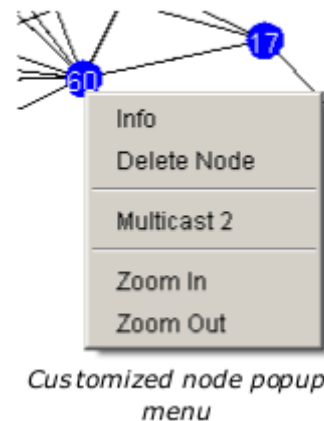
Criando uma simulação

- Agora, iremos implementar o método pelo qual a EB irá construir a árvore de roteamento. Adicione o seguinte método na classe SimpleNode:

```
@NodePopupMethod(menuText="Construir Arvore de Roteamento")
public void construirRoteamento(){
    this.proximoNoAteEstacaoBase = this;
    WsnMsg wsnMessage = new WsnMsg(1, this, null, this, 0);
    WsnMessageTimer timer = new WsnMessageTimer(wsnMessage);
    timer.startRelative(1, this);
}
```


Criando uma simulação

- Algumas considerações sobre o método:
 - A “anotação” **@NodePopupMethod** declara que o método “**construirRoteamento**” será incluído no menu *popup* com o texto “**Construir Arvore de Roteamento**”.
 - Esse *popup* é acessado através de um clique com o botão direito em cima do nó:





Criando uma simulação

- Algumas considerações sobre o método:
 - Nosso método faz referência a um temporizador que ainda não foi criado.
 - Temporizadores servem para especificarmos quando uma determinada ação deverá ser executado.
 - No nosso exemplo, criamos um temporizador com a seguinte declaração:
 - **WsnMessageTimer timer = new WsnMessageTimer(wsnMessage);**
 - Perceba que passamos como parâmetro, para o construtor, a mensagem que queremos transmitir. Mais a frente, explicaremos o porque.



Criando uma simulação

- Algumas considerações sobre o método:
 - Iniciamos o temporizador através do seguinte comando:
 - `timer.startRelative(1, this);`
 - O primeiro parâmetro indica em qual turno o temporizador será disparado. O segundo parâmetro indica qual objeto está disparando o temporizador
 - O tempo 1 indica que o temporizador será disparado no próximo round.

Criando uma simulação

- Construindo o temporizador:
 - Crie uma classe chamada WsnMessageTimer.java dentro de nodes/timers com o seguinte conteúdo:

```
public class WsnMessageTimer extends Timer {  
  
    private WsnMsg message = null;  
  
    public WsnMessageTimer(WsnMsg message){  
        this.message = message;  
    }  
  
    @Override  
    public void fire() {  
        ((SimpleNode)node).broadcast(message);  
    }  
}
```



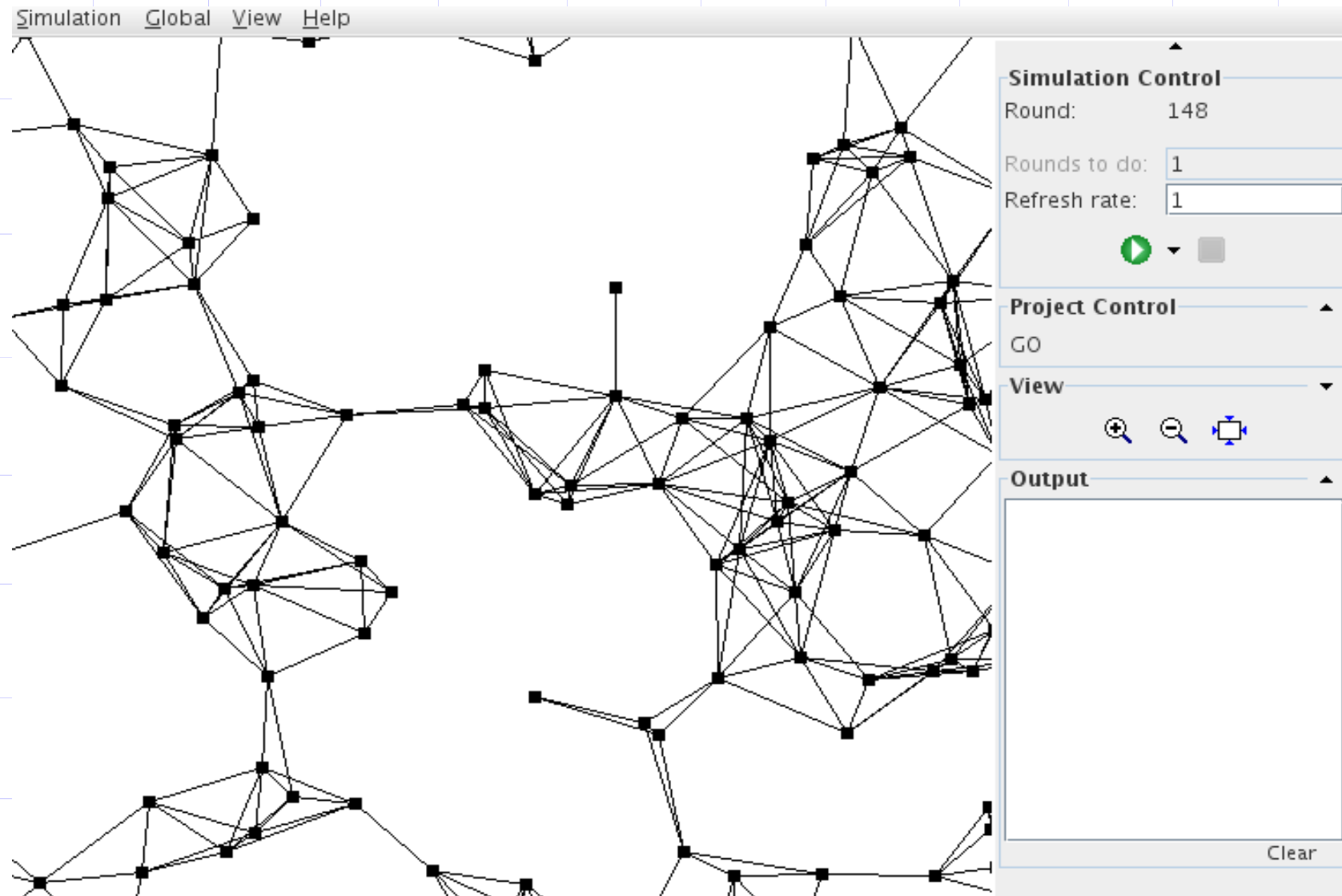
Criando uma simulação

- O temporizador é bastante simples
- Deve-se, basicamente, implementar o método **fire()**, que é a ação a ser executada quando o temporizador for disparado.
- Neste nosso exemplo, toda vez que o temporizador for disparado(ou seja, quando **fire** for executado) o método *broadcast* do nó que iniciou o temporizador será disparado.

Criando uma simulação

- Nossa simulação já está pronta
- Execute o Sinalgo novamente e selecione o projeto wsn.
- Em seguida, crie 200 nós
 - **Detalhe importante:** Na tela de criação de nós, escolha como implementação de nó a classe “**wsn:SimpleNode**”
- Clique no botão Run.
- Perceba que o único evento que ocorreu foi a montagem do grafo de conectividade, ou seja, qual nó está ao alcance do rádio de outro nó

Criando uma simulação



Grafo de conectividade



Criando uma simulação

- Pare a simulação, clicando no botão “Abort Simulation”
- Em seguida, escolha um nó da rede qualquer para ser a estação base.
- Clique, em cima do nó escolhido, com o botão direito. Ao aparecer o menu *popup* clique na opção “Construir árvore de roteamento”
- Esse método será disparado no próximo round, ou seja, quando o botão Run for clicado



Criando uma simulação

- A estação-base irá encaminhar uma mensagem de estabelecimento de rota.
- Cada nó vizinho que receber essa mensagem, irá adicionar o nó transmissor como sua rota até a EB.
- Em seguida, o nó encaminhará a mensagem para seus vizinhos, de forma que, no final, a árvore de roteamento estará montada.



Considerações Finais

- O algoritmo implementado nesse tutorial é bastante simples e serve apenas para efeito didático
- Não levou-se em consideração várias características de uma rede de sensores como energia, congestionamento, etc.
- Contudo, a partir deste tutorial e dos exemplos disponibilizados no próprio Sinalgo, pode-se, praticamente, montar qualquer cenário desejado