SIMULAÇÃO DE REDE DE SENSORES SEM FIO PARA VEÍCULOS AUTÔNOMOS INTELIGENTES UTILIZANDO OMNET++ E MIXIM

THIAGO C. JESUS[1], NATAN H. SANTOS[1].

1. *Departamento de Tecnologia, Universidade Estadual de Feira de Santana*
Av. Transnordestina, s/n, Novo Horizonte. CEP: 44036-900, Feira de Santana-BA
E-mails: natan.ecomp@gmail.com, jesus@ecomp.uefs.br

Abstract— About 5% of the available road space is taken up by cars. This is because humans are so bad at driving. Intelligent vehicles could increase road capacity by up to 273%. Thus, this paper aims to provide an architecture of wireless sensor network to detect, insert and exchange data between vehicles in one of these networks. To this end, a network of wireless sensors is proposed and an algorithm for data communications is presented and simulated.

Keywords— Autonomous vehicles, Intelligent systems, Wireless sensor network, OMNet++/MiXiM.

Resumo— Apenas cerca de 5% do espaço viário disponível é ocupado por carros. Veículos inteligentes poderiam aumentar a capacidade da estrada em até 273%. Assim, este trabalho visa fornecer uma arquitetura de rede de sensores sem fio para detectar, inserir e trocar dados entre veículos em uma dessas redes. Para tanto, uma rede de sensores sem fio é proposta e um algoritmo de comunicação de dados é apresentado e simulado.

Palavras-chave— Veículos autônomos, Sistemas inteligentes, Rede de sensores sem fio, OMNet++/MiXiM.

1 Introdução

Redes de Sensores Sem Fio (RSSF) vem introduzindo uma revolução na forma de monitorar e controlar diversos ambientes. Os avanços nas áreas de redes sem fio, microcontroladores e sensores abriram um leque de possibilidades para desenvolvimento de RSSF, possuindo aplicações nas mais diversas áreas, como: monitoramento de tráfego, diagnósticos em indústria, distribuição de água, automação residencial, saúde, dentre outras. (Zhao e Guibas, 2004; Cunha et al., 2013; Loureiro et al., 2003).

Uma das grandes vantagens da computação móvel sem fio é prover ao usuário acesso contínuo às informações através de uma rede de comunicação sem fio. Este tipo de rede é apropriado para situações nas quais não se pode ter uma instalação com fios e que requer acesso imediato à informação. Esse tipo de ambiente é cada vez mais comum na atualidade, e para tal, as RSSF serão importantes em detectar, coletar e disseminar informações de fenômenos. Aplicações de sensores representam um novo paradigma para operação de rede, que têm objetivos diferentes das redes sem fio tradicionais (Pereira et al., 2004).
Uma das áreas na qual se tem estudado a utilização de RSSF é o monitoramento e controle de tráfego através de veículos autônomos inteligentes. Neste caso é utilizado um conjunto de sensores para obter informações de interesse para o sistema, de forma que sejam tomadas as devidas ações com o intuito de garantir segurança, conforto, estabilidade e rendimento dos veículos automotores (Jung et al., 2005). Essas informações são ainda mais úteis se forem relacionadas com informações de veículos nas proximidades, como aceleração, velocidade e posição desses veículos, que podem ser utilizadas, por exemplo, para reduzir o congestionamento do trânsito.

Em uma estrada ocupada por motoristas humanos (que é de cerca de 2200 veículos por hora por pista), apenas cerca de 5% do espaço viário disponível é ocupado por carros. Isto porque os condutores humanos possuem pouco reflexo, precisando de pistas que são quase o dobro do tamanho dos carros e, em velocidades de estrada, precisam se manter entre 40 e 50 metros longe do carro a sua frente. Veículos inteligentes poderiam aumentar a capacidade da estrada em até 273% (Ackerman, 2012).

Considerando os benefícios da troca de dados entre veículos autônomos, este trabalho propõe uma simulação de uma RSSF entre esses veículos. Cada veículo constitui um nó da rede; os nós trocam dados entre si; a rede não é coordenada por nenhum nó centralizador; qualquer novo nó que esteja no raio de alcance da rede e que implemente os mesmos protocolos e aplicações pode fazer parte da rede. A simulação dessa RSSF foi realizada nos softwares OMNet++ e MiXiM.

1.1 Trabalhos relacionados

Wenjie et al., (2005b) propõe uma arquitetura baseada em RSSF para diminuir o tempo médio de viagem de sistemas de transporte inteligentes, e apresenta simulações feitas apenas para validar o aumento da velocidade média dos veículos nas rodovias, desprezando a troca de mensagens entre nós da rede.

Wenjie et al., (2005a) aborda o problema de detectar a posição de veículos de forma dinâmica e são utilizados os conceitos de RSSF para resolver esse problema de forma a otimizar o fluxo de veículos em um cruzamento, cujos resultados também são apresentados em simulações.

Zou et al., (2012) aplica um novo modelo de simulação de sistemas ciber-físicos a RSSF de veículos não-tripulados, utilizando o Matlab/Simulink.

Xian et al., (2008) apresenta uma comparação entre simuladores de RSSF e mostra que o OMNet++ é, em diversos aspectos melhor e mais escalável do que outros simuladores.

Este trabalho se insere na literatura resolvendo o problema de detecção de veículos móveis em um RSSF e troca de dados entre eles, a fim de otimizar o tráfego de veículos em uma rodovia. Para isso são avaliados dados de simulação com a ferramenta OMNet++.

1.2 Organização do trabalho

A sequência deste trabalho está estruturada da seguinte forma: na seção 2 a nomenclatura e algumas noções preliminares sobre RSSF, o OMNet++ e o MiXiM são apresentadas. Na seção 3 é apresentada a metodologia de desenvolvimento da simulação. Os resultados obtidos são apresentados na seção 4. Finalmente, na seção 5, as conclusões são apresentadas.

2 Noções preliminares

2.1 RSSF

RSSF são redes que normalmente possuem um grande número de nós distribuídos, têm restrições de energia, e devem possuir mecanismos para autoconfiguração e adaptação devido a problemas como falhas de comunicação e perda de nós. Uma RSSF tende a ser autônoma e requer um alto grau de cooperação para executar as tarefas definidas para a rede. Nessas redes, cada nó é equipado com uma variedade de sensores, tais como acústico, sísmico, infravermelho, vídeo-câmera, calor, temperatura, pressão, velocidade, posição, etc. Esses nós podem ser organizados em grupos (*clusters*) onde pelo menos um dos nós deve ser capaz de detectar um evento na região, processá-lo e tomar uma decisão se deve fazer ou não uma difusão (*broadcast*) do resultado para outros nós. A visão é que RSSF se tornem disponíveis em todos os lugares executando as tarefas mais diversas possíveis (Cunha et al., 2013; Loureiro et al., 2003).

2.2 Simulação de RSSF

Existem duas formas de validação de um projeto de RSSF em larga escala. Uma é a realização de experimentação utilizando dispositivos e implantações físicas, e a outra forma é por meio de simulações.

Segundo Colesanti et al., (2007), testes apresentam desafios quanto a reprogramação dos nós, implantações e instrumentação no ambiente físico, que demandam tempo e custos financeiros. Uma RSSF também está fortemente limitada à energia, devido aos nós serem alimentados por baterias. Realizar constantemente a substituição de baterias é economicamente custoso ou inviável na maioria dos casos. Existem problemas também relacionados à fragilidade dos nós e à dificuldade de encontrar falhas em sistemas implantados. Simulações permitem ao pesquisador validar uma RSSF antes de sua implantação e permitem a experimentação em larga escala de protocolos e aplicações em um ambiente flexível. Isso pode ser constatado pelo alto número de trabalhos nos quais a avaliação de desempenho baseia-se principalmente nos resultados da simulação (Wenjie et al., 2005a; Wenjie et al., 2005b; Colesanti et al., 2007; Xian et al., 2008; Zou et al., 2012).

Este trabalho apresenta simulações de RSSF através do pacote OMNet++, que é um simulador de eventos discretos, cujo objetivo é reduzir a complexidade (dependência entre módulos), além de prover ao usuário uma curva de aprendizagem rápida, fazendo com que o desenvolvimento e testes de novos protocolos sejam mais fáceis.

2.2.1 OMNet++

O OMNet++ é um simulador de eventos open-source, orientado a objetos e modular, possuindo um ambiente integrado de desenvolvimento (IDE - *Integrated Development Environment*) próprio, com diversas ferramentas que facilitam a implementação e análise de resultados das simulações. Sua arquitetura genérica permite seu uso para diversos propósitos, sendo um dos simuladores mais comuns e mais utilizados para a simulação de redes. A sua licença é livre para uso acadêmico.

A ideia básica da estrutura do OMNet++ é que um dispositivo possa ser definido por um módulo simples ou composto. O elemento mais básico na estrutura do OMNet++ é o módulo simples (ver Figura 1), que se trata de um elemento indivisível, com sua estrutura, suas *interfaces* de entrada e saída e seus parâmetros descritos em linguagem NED, e seu comportamento em linguagem C++. Os módulos compostos podem ser formados pela união de módulos simples. Módulos se comunicam por troca de mensagens através de suas portas (*gates*).

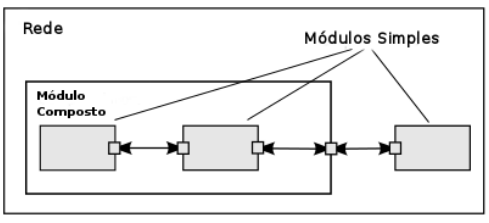


Figura 1. Estrutura do OMNet++.

Uma rede é formada pela união de diversos módulos compostos e simples, aninhados hierarquicamente, que se comunicam passando mensagens uns aos outros. Um módulo pode conter diversos outros submódulos, que por sua vez também pode conter outros submódulos, sendo a profundidade de detalhamento fazendo uso de submódulos ilimitada.

Um projeto no OMNet++ normalmente divide seu fluxo de desenvolvimento como segue:

- O Mapeamento do sistema em uma hierarquia de módulos comunicantes;
- A estrutura modelo é definida em linguagem NED;
- O comportamento dos componentes ativos do modelo (módulos simples) é programado em C++, usando o *kernel* de simulação e bibliotecas de classes;
- A criação de um arquivo de configuração chamado de "omnetpp.ini" para configurar e inicializar parâmetros no modelo OMNet++;
- O projeto é construído e executado, podendo a simulação ser visualizada por meio da ferramenta de linha de comando e pela *interface* gráfica de usuários.;
- Os resultados da simulação são escritos em arquivos de saída vetorial e escalar, podendo esses arquivos ser analisados na ferramenta de análise do próprio IDE. Esses arquivos são arquivos de texto que podem ser lidos com outras ferramentas como R, Scilab, Matlab dentre outras, permitindo alta performance e precisão no tratamento dos dados coletados durante a simulação.

Para auxiliar na visualização da simulação, pode-se utilizar o *framework* MiXiM com o OMNet++.

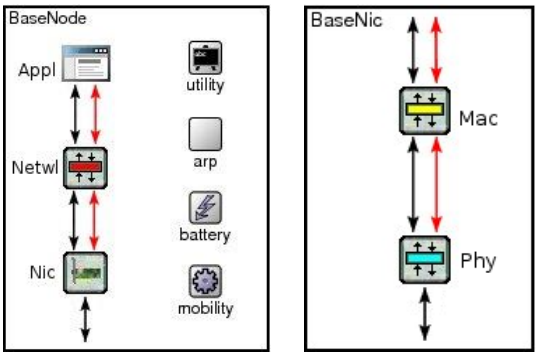
2.2.2 MiXiM

O MiXiM é um *framework* para modelagem de redes sem fio móveis e fixas (como RSSF, WBAN (*Wireless Body Area Network*), *ad-hoc*, veiculares, etc), oferecendo modelos detalhados de propagação de ondas de rádio, estimativa de interferência, consumo de energia do transceptores de rádio e protocolos MAC sem fio, como *Zigbee* (MiXiM Project, 2007).

O projeto do MiXiM oferece recursos para serem utilizados nas camadas inferiores da pilha de protocolo, e é muitas vezes usado em conjunto ao projeto INET *Framework* (INET Framework, 2013), que implementa vários modelos para protocolos de redes cabeadas e wireless, incluindo protocolos UDP, TCP, SCTP, IP, IPv6, Ethernet, PPP, 802.11, MPLS, OSPF, dentre outros.

A Figura 2.a mostra um exemplo típico de módulo presente no MiXiM. Nela é possível ver a divisão em camadas (módulos) que o *framework* faz, tendo o módulo de aplicação ("Appl"), de rede ("Netwl"), e a interface de comunicação ("Nic"), que é dividida em dois módulos (ver Figura 2.b): o módulo que cuida de parte da camada de enlace("Mac") e a camada física ("Phy").

Um módulo interessante e de muita utilidade é o módulo de mobilidade. Com ele é possível simular uma aplicação móvel, determinando a trajetória e velocidade de um nó. Isso se torna útil para simulações de veículos aéreos, carros inteligentes, sensores aquáticos, etc.



(a) (b)

Figura 2. a) Exemplo de um nó no MiXiM. b) Exemplo de um módulo Nic no MiXiM.

Maiores informações sobre o OMNet++ e o MiXiM podem ser encontradas em (MiXiM Project, 2007; Xian et al., 2008; Varga e Hornig, 2008; OMNeT++, 2011).

3 Metodologia e desenvolvimento

Este trabalho destinou-se à utilização do OMNet++ e do MiXiM para a simulação de uma RSSF aplicada ao sensoriamento de veículos autônomos inteligentes. A proposta foi abstrair cada nó da rede como sendo um veículo em movimento, munido de sensores (velocidade, posição) e capaz de se comunicar com outros nós da rede, trocando informações desses sensores. Essas informações serão utilizadas em trabalhos futuros para inferir qual a melhor ação deve ser tomada por cada veículo a fim de otimizar o tráfego.

Alguns problemas da troca de dados em uma rede móvel foram considerados. Primeiro foi preciso pensar no alcance da troca de mensagens. Isso porque dois veículos não estarão trafegando sempre um próximo ao outro, e mais veículos podem se aproximar dos que já estavam se comunicando. Por tanto foi preciso ainda implementar uma solução de identificação e inserção de novos nós na rede. Uma vez que a conexão na rede estivesse estabelecida, restaria apenas a tarefa da troca de dados dos sensores em si.

Essas ideias foram implementadas no OMNet++ e MiXiM. O projeto baseou-se em criar um algoritmo que atue na camada de aplicação de um nó wireless do MiXiM, de forma que os nós sejam móveis, e a aplicação troque dados entre os nós assim que os nós fiquem próximos o bastante para que mensagens possam ser trocadas entre eles. A ideia base é ter um ou mais nós enviando uma mensagem *broadcast* continuamente em intervalos determinados, com intuito de verificar se há algum nó ao redor. Caso algum nó próximo receba essa mensagem ele responde ao *broadcast*. Após a reposta, o nó que inicialmente mandou a mensagem *broadcast* envia seus dados (podendo ser dados sobre sua velocidade, aceleração, posição).

O projeto implementado é constituído de um arquivo "config.xml" que trás definições do modelo analógico e *decider*, utilizados como parâmetros de

perda e atenuação dos sinais para a camada física; um arquivo "Network.ned" que define a topologia da rede; e um arquivo "omnetpp.ini" que define todos os parâmetros do cenário de simulação, como tempo limite de simulação, configurações do canal de comunicação, das camadas física, MAC e de rede, configurações de bateria, mobilidade e aplicação dos nós da rede. Além disso, foi criado um módulo de aplicação que fica dentro do pacote do MiXiM. Para isso foi necessário criar um arquivo com a estrutura da aplicação chamado de "DataExchange.ned", e outros dois arquivos para definir o comportamento do nó: um chamado de "DataExchange.h", que é um arquivo cabeçalho com a declaração de variáveis e métodos da aplicação, e um chamado de "DataExchange.cc" com o código dos métodos e lógica para envio e recebimento de mensagens.

A seguir tem-se a descrição do módulo de aplicação, que é o responsável pelo funcionamento da RSSF implementada.

3.1 Módulo de aplicação

O módulo de aplicação possui como portas lógicas a entrada e saída para dados e para controle (ver Listagem 1). Por meio dessas portas cada módulo se comunica direto com a camada abaixo dele.

```
dataOut = findate("lowerLayerOut");
dataIn = findGate("lowerLayerIn");
ctrlOut = finGate("lowerControlOut");
ctrlIn  = findGate("lowerControlIn");
```

Listagem 1. Configuração de portas.

Existem quatro tipos de mensagens padrões na aplicação: a START_MESSAGE, que é a mensagem inicial que um nó envia quando não há nenhuma requisição a ser atendida. Nesse caso deve ser enviada uma BROADCAST_MESSAGE a fim de descobrir novos nós na rede. Essa é uma mensagem de *broadcast* para a rede, ou seja, todos os nós no alcance do nó remetente receberão essa mensagem. O outro tipo é a mensagem BROADCAST_REPLY _MESSAGE, que é a resposta à mensagem de *broadcast*. Com uma resposta de *broadcast* é possível identificar todos os nós no raio de alcance da rede. Por fim, tem-se a DATA_MESSAGE, que é uma mensagem de dados entre nós.

A troca de mensagens na aplicação é feita com o auxílio dos seguintes métodos: `initialize`, `handleSelfMsg`, `handleLowerMsg`, `sendBroadcast`, `sendReply`, `sendData` e `checkQueue`.

O método `initialize` é responsável por inicializar todas as variáveis utilizadas na aplicação, associando parâmetros e portas da estrutura do módulo. Nesse método é agendada o envio de uma mensagem inicial por meio da função `ScheduleAt` (ver Listagem 2), que é padrão do OMNet++ e realiza o agendamento de uma mensagem para o próprio nó no instante de tempo passado por parâmetro.

```
delayTimer = new cMessage("start",
START_MESSAGE);
scheduleAt(simTime() + dblrand()*10, de-
layTimer);
```

Listagem 2. Agendamento de mensagem de *broadcast*.

O método `handleSelfMsg` (ver Listagem 3) é chamado toda vez que o nó recebe uma mensagem dele mesmo. Normalmente isso ocorre com o agendamento de mensagens e, quando isso ocorre na camada de aplicação para o tipo de mensagem START_MESSAGE, é verificada a fila de requisições (objeto `requests` da Listagem 3). Se ela estiver vazia, é chamado o método `sendBroadcast` que envia efetivamente uma mensagem de *broadcast* (ver Listagem 4). Caso contrário as requisições serão tratadas no método `checkQueue` (ver listagem 5).

```
void DataExchange::handleSelfMsg
                          (cMessage *msg){
  if(requests.empty()){
    if( msg->getKind()== START_MESSAGE){
       sendBroadcast();
       delayTimer = new cMessage("start",
                          START_MESSAGE);
        scheduleAt(simTime() +  5,
                          delayTimer);}}
    else checkQueue();}
```

Listagem 3. Método *handleSelfMsg*.

```
void DataExchange::sendBroadcast(){
    ApplPkt *pkt = new ApplPkt
("BROADCAST_MESSAGE", BROADCAST_MESSAGE);
    pkt->setDestAddr
(LAddress::L3BROADCAST);
    pkt->setSrcAddr( getNode()->getId());
    pkt->setBitLength(headerLength);
    /* set the control info to tell the
     network layer the layer 3 address;*/
    NetwControlInfo::setControlInfo
        (pkt, LAddress::L3BROADCAST );
    sendDown( pkt ); }
```

Listagem 4. Método *sendBroadcast*.

```
void DataExchange::checkQueue(){
  RequestRSSF req = requests.front();
  if(getNode()->getId()!=req.getAddr()){
    requests.pop();
    switch(req.getType()){
      case SEND_REPLY:
        sendReply(req.getAddr()); break;
      case SEND_DATA:
        sendData(req.getAddr()); break;}}
  delayTimer = new cMessage("start",
                          START_MESSAGE);
  scheduleAt(simTime()+5, delayTimer);}
```

Listagem 5. Método *checkQueue*.

O método `checkQueue` apenas verifica o tipo de requisição na fila e atende essa requisição, podendo ser uma resposta de *broadcast* (invocação do método `sendReply`. Ver listagem 6) ou o envio de dados para outro módulo (invocação do método `sendData`. Ver listagem 7).

Uma vez enviada uma mensagem de *broadcast*, cabe ao nó receptor responder essa mensagem. Para tanto, o método `handleLowerMsg` é invocado (ver Listagem 8). Isso ocorre toda vez que a camada de aplicação receber uma mensagem de uma

camada abaixo dela, ou seja, toda vez que houver uma mensagem vinda da camada de rede. Na aplicação esse método verifica qual tipo de mensagem foi recebida e, caso tenha sido uma BROADCAST_MESSAGE, uma requisição do tipo SEND_REPLY é adicionada à fila. Caso tenha sido uma mensagem do tipo BROADCAST_REPLY_MESSAGE, uma requisição do tipo SEND_DATA é adicionada à fila.

```
void DataExchange::sendReply
        (LAddress::L3Type dstAddr){
   ApplPkt *pkt = new ApplPkt
("REPLY_MESSAGE",BROADAST_REPLY_MESSAGE);
   pkt->setDestAddr(dstAddr);
   pkt->setSrcAddr (getNode()->getId());
   pkt->setKind(BROADCAST_REPLY_MESSAGE);
   pkt->setBitLength(headerLength);
   sendDown(pkt);}
```

Listagem 6. Método *sendReply*.

```
void DataExchange::sendData
        ( LAddress::L3Type dstAddr){
   float speed=((float)rand()/
               (float)RAND_MAX)*100;
   char cSpeed[32];
   sprintf(cSpeed,"%.2f",speed);
   ApplPkt *pkt = new ApplPkt
               (cSpeed, DATA_MESSAGE);
   pkt->setDestAddr(dstAddr);
   pkt->setSrcAddr (getNode()->getId());
   pkt->setKind(DATA_MESSAGE);
   pkt->setBitLength(headerLength);
   sendDown(pkt); }
```

Listagem 7. Método *sendData*.

```
void DataExchange::handleLowerMsg
                        (cMessage *msg){
   ApplPkt *m; RequestRSSF r;
   switch( msg->getKind() ) {
     case BROADCAST_MESSAGE:
       m = static_cast<ApplPkt *>(msg);
       dstAddr = m->getSrcAddr();
       r.setAddr(dstAddr);
       r.setType(SEND_REPLY);
       requests.push(r); break;
     case BROADCAST_REPLY_MESSAGE:
       m = static_cast<ApplPkt *>(msg);
       dstAddr = m->getSrcAddr();
       r.setAddr(dstAddr);
       r.setType(SEND_DATA);
       requests.push(r); break;
     case DATA_MESSAGE:
       m = static_cast<ApplPkt *>(msg);
       EV << "\n data arrive, speed:" <<
           m->getName() <<"mps \n"<<endl;
       delete msg; break;  }}
```

Listagem 8. Método *handleLowerMsg*.

No método `sendData` (Listagem 7) é importante notar a criação de uma variável chamada `speed`. Ela é apenas uma abstração da leitura de um sensor. Espera-se, em trabalhos futuros, que esse valor enviado seja o valor real da leitura de um sensor associado ao movimento do veículo.

É importante ressaltar ainda a utilização da linha de código "`getNode()->getId()`" nas Listagens 4, 5, 6 e 7. Isso se deve ao fato de se estar implementando a camada de aplicação da rede, portanto apenas endereços lógicos, dados em forma de identificadores, são utilizados. Endereços físicos e de rede são encapsulados pelo simulador, sendo transparentes ao usuário.

4 Resultados

Após descrever todo o comportamento da aplicação e especificar as características físicas da rede, passou-se para a etapa de simulação. A visualização gráfica de uma RSSF fornecida pelo MiXiM por ser vista na Figura 3. Nela é possível ver quatro nós, que estão representando quatro veículos, com uma seta dupla entre eles, representando que os nós estão dentro de um raio de alcance. O texto na seta dupla indica que o nó *1* enviou uma mensagem de *broadcast* para a rede, que nesse caso é recebida pelos nós *0*, *2* e *3*.

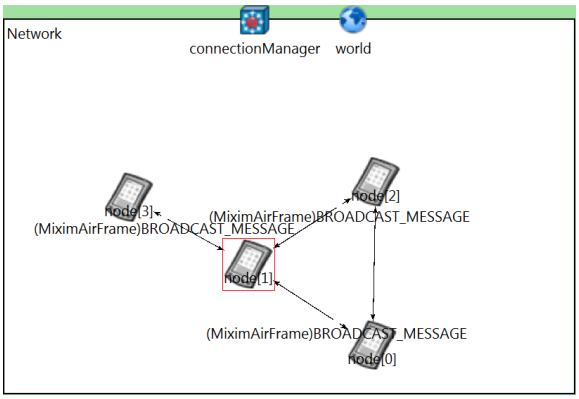


Figura 3. Interface gráfica da simulação. Mensagem de *broadcast*.

Após receber uma mensagem de *broadcast* os nós da rede respondem a essa mensagem, como pode ser visto na Figura 4. Isso indica a aplicação que um nó foi encontrado na rede.

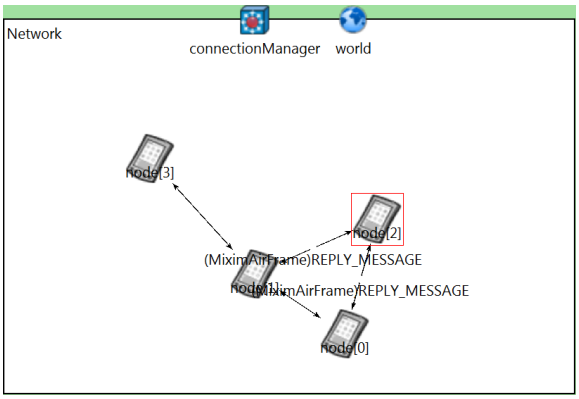


Figura 4. Mensagem de resposta de broadcast.

O próximo passo da simulação é obter o endereço do remetente da resposta de *broadcast* e enviar para esse nó um dado de sensor. Nesse caso, como mostra a Figura 5, o nó *1* está enviando o dado *93.12* para os nós *0*, *2* e *3*. Toda essa sequência de eventos é repetida enquanto a simulação estiver ativa. Eventualmente, devido a mobilidade dos nós, esses se afastam e ficam fora do alcance um do outro, implicando na quebra da comunicação. Nesse caso,

as mensagens de *broadcast* não têm resposta, não havendo então a troca de dados entre nós.

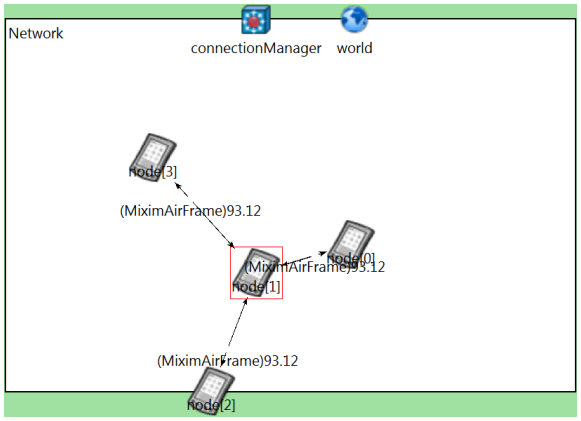


Figura 5. Mensagem de dados.

5 Conclusão

Neste trabalho foi apresentado um projeto de simulação de RSSF para o monitoramento e controle de tráfego através de veículos autônomos inteligentes. A proposta foi simular a identificação, inserção e troca de dados entre nós na rede. Esse problema foi resolvido através de uma estratégia simples de confirmações de mensagens na rede.

As ferramentas utilizadas na simulação foram o OMNet++ e o MiXiM. Elas foram escolhidas por serem simples de usar, eficientes, oferecem *interface* amigável, dar suporte a uma vasta de protocolos de rede, além de ter fácil integração com outros *softwares* de simulação, como o R, Scilab e o Matlab.

Os resultados obtidos neste trabalho compõe o passo inicial de um projeto maior. Uma vez que se é possível identificar, inserir e trocar mensagens entre nós em uma rede móvel e altamente dinâmica, deve-se agora utilizar os dados trafegados na rede para tomar a melhor decisão no trânsito (acelerar, reduzir, mudar de faixa, encostar, parar) a fim de otimizar a utilização das rodovias. Pode-se pensar ainda em fazer um mapa geoposicionado do tráfego ao seu redor para dar maior segurança aos carros não-autônomos, gerando assim maior precisão aos sistemas GPS.

Agradecimentos

Este trabalho foi parcialmente financiado pela FAPESB.

Referências Bibliográficas

Ackerman, E. (2012). Study: Intelligent Cars Could Boost Highway Capacity by 273%. IEEE Spectrum. http://spectrum.ieee.org/automaton/robotics/artificial-intelligence/intelligent-cars-could-boost-highway-capacity-by-273 (Acesso em 03/06/2013).

Colesanti, U. M.; Crociani, C. and Vitaletti, A. (2007). On the Accuracy of OMNeT++ in the Wireless Sensor Networks Domain: Simulation vs. Testbed. Proc. of the 4th ACM workshop on Performance evaluation of wireless ad hoc, sensor and ubiquitous networks.

Cunha, F. D., Cunha, I., , Loureiro, A. A. F. and Oliveira, L. B. (2013) . Uma Nova Abordagem para Acesso ao Meio em Redes de Sensores Sem Fio.31º Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos.

INET Framework (2013). http://inet.omnetpp.org/. (Acesso em 03/06/2013).

Jung, C. R.; Osório, F. S.; Kelber, C. and Heinen, F. (2005) "Computação embarcada: Projeto e implementação de veículos autônomos inteligentes", In: Anais do CSBC'05 XXIV Jornada de Atualização em Informática. São Leopoldo, RS, v.1, pp. 1358–1406.

Loureiro, A. A. F., Nogueira, J. M. S., Ruiz, L. B., Mini, R. A., Nakamura, E. F., and Figueiredo, C. M. S. (2003). Redes de sensores sem fio. In Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos, pp 179–226, Natal,RN, Brazil.

MiXiM Project (2007). http://mixim.sourceforge.net/index.html. (Acesso em 03/06/2013).

OMNeT++. User Manual - Version 4.3. (2011). http://www.omnetpp.org/doc/omnetpp/Manual.pdf. (Acesso em 03/06/2013).

Pereira, M. R., Amorim, C. L. and Castro, M. C. S. (2004). Tutorial sobre redes de sensores. In Simpósio Brasileiro de Redes de computadores.

Varga, A. and Hornig, R. (2008). An overview of the OMNeT++ simulation environment. In Proceedings of the 1st International Conference on Simulation tools and techniques for communications, networks and systems & workshops, Brussels, Belgium.

Wenjie, C., Lifeng, C., Zhanglong, C., Shiliang, T., (2005a). A Realtime Dynamic Traffic Control System Based on Wireless Sensor Network. Proceedings of the International Conference on Parallel Processing Workshops. pp.258-264.

Wenjie, C., Liqiang, G., Zhilei, C., Zhanglong, C., and Shiliang, T. (2005b). An Intelligent Guiding and Controlling System for Transportation Network Based on Wireless Sensor Network Technology. Proc. of the Fifth International Conference on Computer and Information Technology.

Xian, X., Shi, W., and Huang, H. (2008). Comparison of OMNET++ and Other Simulator for WSN Simulation. Proc. 3rd IEEE Conference on Industrial Electronics and Applications ICIEA, pp. 1439-1443.

Zhao, F. and Guibas, L. (2004). Wireless Sensor Networks: An Information Processing Approach. 1ª edição. Morgan Kaufmann. 2004. 376 p.

Zou, C., Wan, J., Chen, M. and Li, D. (2012). "Simulation Modeling of Cyber-Physical Systems Exemplified by Unmanned Vehicles with WSNs Navigation". Embedded and Multimedia Computing Technology and Service. Lecture Notes in Electrical Engineering. Vol. 181, pp 269-275.