UNIVERSIDADE FEDERAL DO CEARÁ

CAMPUS QUIXADÁ

TECNÓLOGO EM REDES DE COMPUTADORES

FRANCISCO SAMUEL DE SOUSA SILVA

AVALIAÇÃO DE SIMULADORES PARA O ENSINO DE IOT EM REDES DE COMPUTADORES

QUIXADÁ – CEARÁ

2017

FRANCISCO SAMUEL DE SOUSA SILVA

AVALIAÇÃO DE SIMULADORES PARA O ENSINO DE IOT EM REDES DE COMPUTADORES

Monografia apresentada no curso de Redes de Computadores da Universidade Federal do Ceará, como requisito parcial à obtenção do título de tecnólogo em Redes de Computadores. Área de concentração: Computação.

Orientador: Prof. Me. Michel Sales Bonfim

QUIXADÁ – CEARÁ

2017

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Biblioteca Universitária
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

S58a Silva, Francisco Samuel de Sousa.
Avaliação de simuladores para o ensino de IOT em redes de computadores / Francisco Samuel de Sousa Silva. – 2017.
55 f. : il. color.

Trabalho de Conclusão de Curso (graduação) – Universidade Federal do Ceará, Campus de Quixadá, Curso de Redes de Computadores, Quixadá, 2017.
Orientação: Prof. Me. Michel Sales Bonfim.

1. Simulador. 2. Internet das coisas. 3. Redes de computadores - Ensino-Brasil. I. Título.
CDD 004.6

FRANCISCO SAMUEL DE SOUSA SILVA

AVALIAÇÃO DE SIMULADORES PARA O ENSINO DE IOT EM REDES DE COMPUTADORES

Monografia apresentada no curso de Redes de Computadores da Universidade Federal do Ceará, como requisito parcial à obtenção do título de tecnólogo em Redes de Computadores. Área de concentração: Computação.

Aprovada em: ____/____/____

BANCA EXAMINADORA

Prof. Me. Michel Sales Bonfim (Orientador)
Universidade Federal do Ceará – UFC

Prof. Me. Jeandro de Mesquita Bezerra
Universidade Federal do Ceará - UFC

Prof. Dr. Paulo Antonio Leal Rego
Universidade Federal do Ceará - UFC

A Deus.

Aos meus pais, Joana Dar'c de Sousa Silva e Francisco Amancio da Silva.

AGRADECIMENTOS

À Deus, por ser sempre um pilar de segurança e força em minha vida.

Ao Prof. Ms. Michel Sales Bonfim, pela excelente orientação e ensinamentos em sala de aula.

Aos professores participantes da banca examinadora Prof. Me. Jeandro de Mesquita Bezerra e Prof. Dr. Paulo Antonio Leal Rego, pelo tempo, pelas valiosas colaborações e sugestões.

À todos os professores, que contribuíram para a minha formação acadêmica.

Aos colegas de turma, pelas reflexões, críticas e sugestões recebidas.

Aos Alisson Lima e Roseli Rocha, pelas correções no texto e ajuda na formatação.

À minha irmã Círia de Sousa e minha amiga Paula Holanda, pela cobrança, força e ânimo, que contribuíram muito para a conclusão deste trabalho.

“Nunca nada lhe foi prometido, você não assinou nenhum contrato.”

(Charles Bukowski)

RESUMO

Com o crescimento da Internet das Coisas, torna-se necessária a simulação de redes, que é uma técnica essencial para o estudo e avaliação de desempenho de aplicações. O uso de simuladores de redes é de fundamental importância no processo de novas pesquisas, desenvolvimento de novos protocolos de redes e análises de desempenho de uma rede, como também no desenvolvimento de novas tecnologias. Em relação ao ensino, os simuladores são ferramentas que podem desempenhar um papel primordial, contribuindo em peso para um melhor entendimento do conteúdo pelos alunos. Este trabalho tem como finalidade geral demonstrar os benefícios do uso de simuladores no ensino na área de IoT. Para isso, avaliamos três simuladores disponíveis: Cooja, TOSSIM e ns-3, considerando suas principais características, recursos e protocolos disponíveis.

Palavras-chave: Simulador. Internet das coisas. Redes de computadores - Ensino-Brasil.

ABSTRACT

With the growth of the internet of things, it becomes necessary to simulate networks, which is an essential technique for the study and evaluation of application performance. The use of network simulators is of fundamental importance in the process of new research, development of new network protocols and analysis of network performance, as well as in the development of new technologies. In relation to teaching, simulators are tools that can play an important role, contributing in a significant way to a better understanding of the content by the students. This work has the general purpose of demonstrating the benefits of using simulators in teaching in the area of IoT. For this, we evaluated three available simulators: Cooja, TOSSIM and ns-3, considering their main characteristics, resources and protocols available.

Keywords: Simulator. Internet of things. Computer Networks - Teaching-Brazil.

LISTA DE FIGURAS

Figura 1 – Arquitetura de cinco camadas 19
Figura 2 – Aplicações *IoT* 20
Figura 3 – Protocolos *IoT* 21
Figura 4 – Janelas para configuração de experimentos 24
Figura 5 – Editor de Script de Simulação 26
Figura 6 – Informação de Todos os Motes Simulados 27
Figura 7 – Histórico de Consumo de Energia 27
Figura 8 – NetAnim 29
Figura 9 – PyViz 30
Figura 10 – Resultados do Testes realizados pelo Waf 31
Figura 11 – Log de Resultados no NS-3 33
Figura 12 – Saída da compilação do simulador TOSSIM 35
Figura 13 – Assinatura do objeto 36
Figura 14 – Resultado do script completo para simular a transmissão de pacotes entre motes 38
Figura 15 – TinyViz 39
Figura 16 – Serial Forwarder 40
Figura 17 – Topologia Básica 42
Figura 18 – Criação de uma nova simulação 43
Figura 19 – Criação de um mote 43
Figura 20 – Simulação em Andamento 44
Figura 21 – Leitura do arquivo .pcap através da aplicação tcpdump 47

LISTA DE TABELAS

Tabela 1 – Relação entre temas propostos e ferramentas 16
Tabela 2 – Métricas disponíveis em gráficos no simulador Cooja 28
Tabela 3 – Comparação técnica entre os simuladores 49
Tabela 4 – Suporte para protocolos de *IoT*. 49

LISTA DE ABREVIATURAS E SIGLAS

6LoWPAN	*IPv6 over Low power Wireless Personal Area Networks*
API	*Application Programming Interface*
COAP	*Constrained Application Protocol*
GUI	*Graphical User Interface*
IOT	*Internet of Things*
ITU	*International Telecommunication Union*
JNI	*Java Native Interface*
MQTT	*Message Queue Telemetry Transport*
PCAP	*Packet capture*
IOT	*Personal Digital Assistants*
RFID	*Radio-Frequency IDentification*
RPL	*Routing Protocol for Low power and Lossy Networks*

SUMÁRIO

1	**INTRODUÇÃO**	12
1.1	**Objetivos**	14
1.2	**Estrutura do Trabalho**	15
2	**TRABALHOS RELACIONADOS**	16
3	**FUNDAMENTAÇÃO TEÓRICA**	18
3.1	**Internet das Coisas (*IoT*)**	18
3.2	**Simulação**	22
4	**SIMULAÇÃO EM *IOT***	23
4.1	***Cooja***	23
4.1.1	***Script Editor***	25
4.1.2	***Sensor Collect***	26
4.2	***NS-3***	28
4.3	**TOSSIM**	34
5	**AVALIAÇÃO DAS FERRAMENTAS DE SIMULAÇÃO**	41
5.1	**Executando um Cenário Simples**	41
5.2	**Simulação Cooja**	42
5.3	**Simulação NS-3**	44
5.4	**Simulação TOSSIM**	47
5.5	**Comparação das Ferramentas de Simulação**	48
6	**CONSIDERAÇÕES FINAIS**	51
	REFERÊNCIAS	52

1 INTRODUÇÃO

A taxa de crescimento do desenvolvimento humano vem aumentando rapidamente nos últimos dois séculos, devido o uso de diferentes tecnologias. Com o aumento em relação a quantidade dos dispositivos, a redução de custo e de tamanho físico diminui, enquanto o desempenho e o número de usuários continuam em uma crescente. Pesquisas mostram que nos próximos anos haverá um aumento no número de conexões de redes através das quais quase todos os usuários estarão conectados de diferentes dispositivos, como desktops, laptops, smartphones, PDA, etc. As razões proeminentes são o tamanho, o custo e a nova tecnologia IPv6, que permite bilhões de endereços, o que é suficiente para fornecer um endereço IP para cada objeto conectado (DEOGIRIKAR; VIDHATE, 2017). A Internet das Coisas (do inglês, *Internet of Things, IoT*) conecta as coisas com aplicações comerciais e operacionais - redes de energia, pisos de fabricação, dispositivos de saúde, carros. Ao vincular coisas inteligentes entre si e com pessoas, lugares e dados, podemos trabalhar de maneiras mais simples, mais automatizadas e mais inteligentes (CISCO, 2017).

Em seu relatório de mobilidade anual, a Ericsson, empresa líder mundial no setor de telecomunicações, prevê um número de aproximadamente 29 bilhões de dispositivos conectados à internet já em 2022.

O termo *Internet of Things* foi introduzido pela primeira vez como uma ideia em 1999 por Kevin Ashton , que agora evoluiu para uma realidade que interliga sensores, dispositivos eletrônicos e sistemas do mundo real para a Internet (ASHTON, 2011). No ano de 2005, a União Internacional de Telecomunicações (ITU) divulgou um relatório anual sobre "Internet das coisas", no relatório, a UIT apontou que tecnologias como RFID abriram uma era que possibilita interligar dispositivos em um âmbito global (ITU, 2005).

A *IoT* pode ser aplicada em várias áreas, como transporte, agricultura, saúde, etc. As vantagens da *IoT* são quase ilimitadas e suas aplicações estão mudando a maneira como trabalhamos e vivemos, economizando tempo e recursos, além de abrir novas oportunidades de crescimento, inovação e intercâmbio de conhecimentos entre entidades (DEOGIRIKAR; VIDHATE, 2017).

Com toda essa pesquisa no cenário da *IoT*, torna-se necessário usar um método específico para construir, testar e analisar novas soluções para estes cenários, de maneira rápida e eficiente. Os três métodos mais conhecidos são: *Testbeds*(medições em ambientes reais), simulação e emulação. Cada método possui alguns pontos fortes e fracos que se complementam.

A implementação e implantação desses métodos dependem de recursos, suporte de infraestrutura e requisitos (SARI; WIRYA, 2007).

Testbeds são baseadas em um conjunto de execuções de teste de um protocolo sobre uma rede real, e portanto é capaz de produzir resultados mais confiáveis. Não obstante, por causa da grande influência da topologia, dos sistemas e das configurações envolvidas (rede não controlada sujeita à tráfego de fundo, etc.), os resultados coletados são específicos de uma configuração e são difíceis de se reproduzir por outros pesquisadores. *Testbeds* naturalmente, exige a implementação do protocolo, ou no mínimo de um protótipo do mesmo (MUHAMMAD et al., 2004).

Por outro lado, um emulador é uma ferramenta que reproduz uma plataforma virtualizada permitindo que uma dada arquitetura de computador consiga executar sistemas que foram desenvolvidos para outra arquitetura específica. Desse modo, um emulador permite que o usuário faça com que o seu computador pessoal aparente ser outra plataforma (como um *switch* ou roteador) para rodar outro sistema operacional (SOUSA et al., 2016). A limitação fica por conta do desempenho do elemento emulado, notadamente inferior ao de um elemento físico. Por este motivo, é desaconselhável a aplicação de elementos emulados em testes de desempenho (FILIPPETTI, 2010).

Já o uso de simulações, é baseada na utilização de algoritmos matemáticos que representam um determinado objeto a ser simulado. Nesse contexto, os pacotes e os eventos que ocorrem em uma simulação são apenas fictícios, com as trocas de dados e os resultados originados de cálculos representativos. O comportamento da rede portanto, não deve alterar durante suas execuções, tendo como vantagem a obtenção resultados iguais ou similares a cada simulação. Durante cada execução, é importante lembrar que não existem dados reais ou mesmo recursos que necessitem da máquina hospedeira, todo conhecimento é representativo e não uma ação real dentro de um sistema. No entanto, os algoritmos matemáticos devem representar de forma fidedigna o mundo que pretende simular. Outra vantagem que é importante destacar, é a possibilidade de utilizar recursos que não estão disponíveis, como realizar uma simulação do comportamento de uma rede com equipamentos operando a 100 Gigabits por segundo, em um período de 10 anos (MENOSSI et al., 2017).

A partir dos conceitos expostos é importante mencionar que a inserção de novas tecnologias é cada vez mais frequentes, muitas vezes dificultando o aprendizado e o acompanhamento de cada evolução tecnológica. Baseado nesse contexto, o papel da educação é

fundamental na formação de profissionais mais qualificados, cujo objetivo é os adequar as novas competências que atendam às necessidades do mercado.

No âmbito de ensino na área de redes de computadores não é diferente, os conceitos são difíceis de serem entendidos na forma pedagógica tradicional. Entretanto, utilizando tecnologias digitais é possível facilitar o aprendizado do aluno, através da utilização de atividades, simulações e exercícios que complementam a fixação dos assuntos abordados (VOSS et al., 2012). Porém, para arcar com um ambiente de experimentação que tenha suporte a diversos tipos de experimentos em redes de computadores, as instituições teriam que realizar um investimento de alto custo em equipamentos de redes (e.g. Concentradores, Roteadores, Firewall, etc.), e até mesmo em relação a espaço físico, tornando inviável manter os laboratórios compatíveis com a realidade do mercado. Visto isso, optar pelo uso de simuladores e emuladores seria a opção mais viável.

Com base no que foi descrito, este trabalho se justifica, entre outros motivos, pela necessidade da criação de um novo cenário educacional no qual é fundamental conhecer novas formas de experimentação de redes voltadas à *IoT*. Para este estudo, foram utilizados três simuladores (Cooja, NS-3 e TOSSIM), com o intuito de qualificar qual a ferramenta mais adequada para o ensino de *IoT* na graduação em Redes de Computadores, demonstrando suas funcionalidades, tipos de *hardware*, protocolos suportados, vantagens e desvantagens.

A escolha das ferramentas foi baseada em um levantamento bibliográfico exploratório, com o objetivo de identificar quais simuladores seriam mais adequados para o ensino de *IoT* para alunos de graduação. Optamos também pela escolha de *softwares* livres, onde tornaria mais viável a sua implantação em uma instituição de pesquisa.

1.1 Objetivos

Este trabalho tem como o objetivo, avaliar ferramentas de simulação que provêm suporte a *IoT*, para o auxílio no ensino de Redes de Computadores, contribuindo para um melhor entendimento dos assuntos abordados nas disciplinas.

Os seguintes objetivos específicos foram levados em conta para alcançar a meta do objetivo geral.

- Identificar simuladores para análise.
- Avaliar a usabilidade de diferentes simuladores;
- Comparar e recomendar os simuladores de redes, de acordo com a necessidade

de uso;

1.2 Estrutura do Trabalho

Esta monografia está organizada da seguinte forma: o capítulo 1 contextualiza o trabalho e apresenta os objetivos propostos. O capítulo 2 apresenta uma breve discussão sobre os estudos relacionados a este trabalho. Nos capítulos 3 e 4, são apresentados os fundamentos e definições referentes à Internet das Coisas e simulação, bem como as ferramentas utilizadas. A validação da proposta é definida na seção 5, esclarecendo o planejamento para a criação do cenário, as métricas utilizadas e os resultados obtidos. Por fim, o capítulo 6 aponta as considerações finais encontradas com a realização desta pesquisa.

2 TRABALHOS RELACIONADOS

Em Voss et al. (2012), é apresentada uma proposta de ensino baseada na utilização de vários *softwares* simuladores. A pesquisa foi desenvolvida a partir de um estudo sobre os temas abordados na disciplina de Redes de Computadores em nível de graduação. Na primeira etapa do trabalho, foram analisadas as ementas dessa disciplina de três instituições de ensino. Na segunda etapa, foi feito um comparativo entre os conteúdos, considerando a aplicação de laboratórios virtuais baseados nos simuladores de redes utilizados (*Cisco Packet Tracer, NCTUns, JimSim, GNS3, NetSimK, NS-2, OPNET, Cnet e SiReViW*). Na avaliação, foram considerados os objetivos, carga horária e conteúdos apresentados. Considerando a descrição dos objetivos, associados aos conteúdos em comum nas ementas, foram escolhidos os temas que poderiam ser trabalhados com as ferramentas. Com base na análise e comparação entre os temas apresentados na disciplina e as ferramentas utilizadas na abordagem, os autores demonstraram que é viável a utilização das ferramentas, de forma articulada, contribuindo assim para uma melhor reflexão e compressão dos temas envolvidos no ensino. A Tabela 1 apresenta resumidamente o resultado da análise realizada para estabelecer uma correspondência entre os conteúdos abordados na disciplina e quais ferramentas podem ser utilizadas para auxiliar na compreensão e reflexão sobre um determinado assunto.

Tabela 1 – Relação entre temas propostos e ferramentas

Temas Propostos	Packet Tracer	Jim SIm	Net Simk	OPNET IT Guru	SireViW
Topologias de Rede	X		X	X	X
Modelo TCP/IP	X	X	X	X	X
Projeto e Gerência de Rede	X		X	X	X

Fonte – Voss et al. (2012)

Já em Santos (2016), os autores buscam identificar as contribuições do uso de *softwares* simuladores de redes, como estratégias no processo de ensino-aprendizagem do conteúdo de Redes de Computadores, identificando entre as ferramentas, a mais indicada. Além disso, buscam validar a ferramenta escolhida e identificar a percepção de professores e alunos em relação ao uso da ferramenta no ensino de Redes de Computadores. Na avaliação, foram levantadas problemáticas sobre desafios no ensino de Redes de Computadores, realidade virtual na educação e um novo modelo de ensino para aulas mais dinâmicas, conceitos esses que

acabaram por contribuir para o desenvolver desse documento. Por fim, listar as contribuições da ferramenta no ensino de Redes de Computadores e fatores que contribuem na aprendizagem sobre Redes de Computadores.

Porém, os autores não realizam nenhum experimento, bem como, não demonstram nenhum resultado, ficando como gancho para o próximo artigo.

Siraj, Gupta e Badgujar (2012), apresentaram uma proposta baseada em testes com diferentes simuladores de redes, destacando suas principais características, vantagens e desvantagens. O objetivo do artigo era servir de fonte de referência para ajudar na escolha do simulador mais apropriado para uma determinada aplicação. Os autores fizeram uma abordagem da importância do uso de simuladores em diversos setores da sociedade e destacaram suas contribuições relacionadas à redução de custos e tempo de implementações. A comparação entre os simuladores foram baseadas no intervalo (do simples para muito complexo), especificação de *hosts*, das ligações e o tráfego entre eles, na especificação dos protocolos existentes, aplicações gráficas (interface com usuário), modos de operação (normal 23 e avançado) e nas ferramentas para criar as simulações. Dentre os diferentes tipos de simuladores existentes, os autores utilizaram, como objeto de estudo, os simuladores *OPNET, NS-2, NS-3, NetSim, OMNeT ++, REAL, J-Sim e QualNet*O interessante do artigo é que esses simuladores utilizados possuem características bem diferenciadas, contribuindo com várias opções de escolha. Na conclusão do artigo, os autores apresentaram uma simulação específica para cada um dos simuladores, sem definir qual é o melhor, o mais utilizado, entre outros. A ideia foi mostrar as características distintas de cada um deles, com base nas simulações, sendo possível identificar a aplicação de cada um deles no ensino-aprendizagem de Redes de Computadores.

É nítida a relação dos trabalhos citados acima com o nosso, eles descrevem modelos de avaliação, tipos de simuladores, levantamento de problemáticas e definições de tecnologias que foram de suma importância para o levantamento bibliográfico. No entanto, o nosso trabalho se diferencia dos demais pelo fato que o foco de estudo é voltado para o ensino de Internet das Coisas (*IoT*) no âmbito acadêmico de Redes de Computadores. Por isso, a escolha das ferramentas, métricas e funcionalidades também foram voltadas para essa área.

3 FUNDAMENTAÇÃO TEÓRICA

Este capítulo destina-se à introdução dos fundamentos conceituais sobre os temas IoT, simulação e emulação, que estão diretamente relacionados a este trabalho.

3.1 Internet das Coisas (*IoT*)

Em nossa vida cotidiana, incluindo nossas casas e locais de trabalho, os aparelhos tornaram-se uma parte imperativa da nossa vida. Estamos nos movendo para a conexão "a qualquer hora em qualquer lugar conectado a qualquer coisa". Com o surgimento de tecnologias avançadas como a *IoT*, estamos em um modo "conectado"com as coisas que nos rodeiam. Tais avanços se desenvolveram em nossas vidas diárias de forma muito robusta. Através da *IoT*, as coisas do mundo real são uma parte da Internet, combinando perfeitamente o mundo físico e digital (DATTA; SHARMA, 2017).

No início dos anos 2000, Kevin Ashton estava lançando as bases para o que se tornaria a *Internet of Things* (*IoT*) no laboratório do *MIT AutoID*. Ashton foi um dos pioneiros que concebeu esta noção, ele procurava maneiras de como a Proctor & Gamble (empresa estadunidense fundada em 1836) poderia melhorar seus negócios, ligando informações *RFID* para a Internet. O conceito era simples mas poderoso. Se todos os objetos na vida diária fossem equipados com identificadores e conectividade sem fio, esses objetos poderiam ser comunicados uns com os outros e serem gerenciados por computadores (CISCO, 2001).

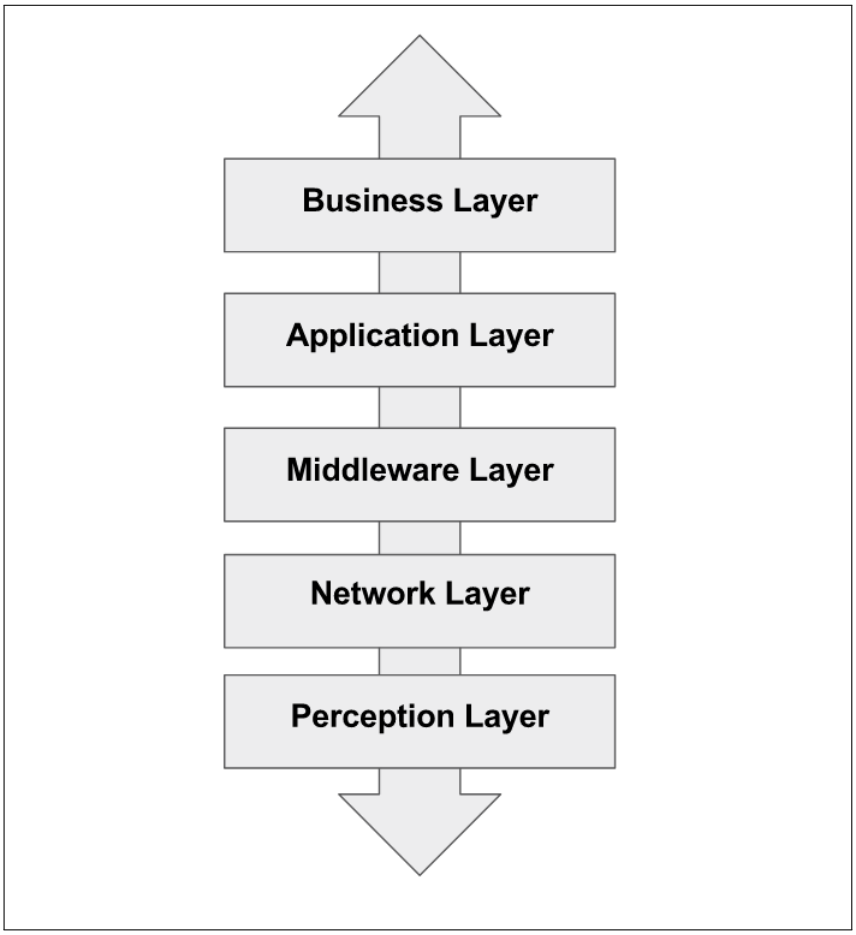
A *IoT* foi definida na recomendação UIT-T Y.2060 (06/2012) como uma infraestrutura global para a sociedade da informação, permitindo serviços avançados através da interconexão de elementos (físicos e virtuais) com base na informação interoperável existente e em evolução nas tecnologias de comunicação (ASHTON, 2011) . No âmbito de arquitetura, as duas formas mais comuns e amplamente consideradas em negócios, pesquisas industriais e aplicações, são as conhecidas como arquiteturas de três camadas e cinco camadas, respectivamente (AL-QASEEMI et al., 2016).

A arquitetura de três camadas é considerada uma arquitetura *IoT* essencial. De acordo com o próprio nome, basicamente consiste em três camadas: camada de percepção, camada de rede e camada de aplicação. A camada de percepção é conhecida como uma camada física porque contém os dispositivos físicos. Nesta camada, os sensores incorporados coletam os dados dos dispositivos e enviam esses dados para a camada de rede. A camada de rede é

responsável pela conexão da camada de percepção com a camada física. Esta camada deve ter tecnologias de conexão, como conexão segura com fio ou sem fio. Os dados recebidos por camada de aplicação serão analisados e processados para fornecer serviços e tomar decisões, enviar os resultados de volta da camada de aplicação para a camada de percepção através da camada de rede (AL-QASEEMI et al., 2016).

A arquitetura de cinco camadas tem as mesmas três camadas, mas com duas camadas adicionais que são a camada de acesso e a camada de *middleware*, como mostrada na Figura 1. As camadas de percepção, rede e aplicação têm a mesma funcionalidade que possuem na arquitetura de três camadas. A camada de acesso é responsável por gerenciar a comunicação *IoT* no ambiente particular e trocar mensagens entre objetos e sistemas. A camada *Middleware* proporciona uma associação mais flexível entre dispositivos de *hardware* e aplicativos (AL-QASEEMI et al., 2016).

Figura 1 – Arquitetura de cinco camadas

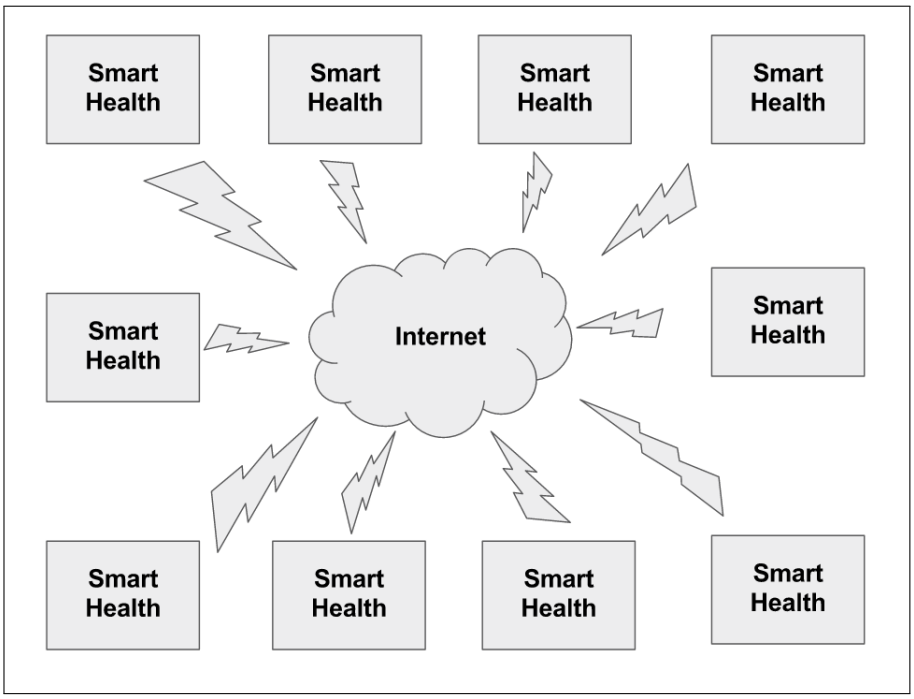


Fonte – Produzida pelo Autor

Ao longo do tempo, espera-se que a *IoT* tenha aplicações significativas, contribuindo

para a qualidade de vida e para o crescimento da economia mundial. Por exemplo, as casas inteligentes permitirão que seus moradores abram automaticamente sua garagem ao chegar em casa, monitorem sistemas de controle climático, *TVs* e outros aparelhos, como mostrado na Figura 2, e exemplificado logo em seguida.

Figura 2 – Aplicações *IoT*



Fonte – Produzida pelo Autor

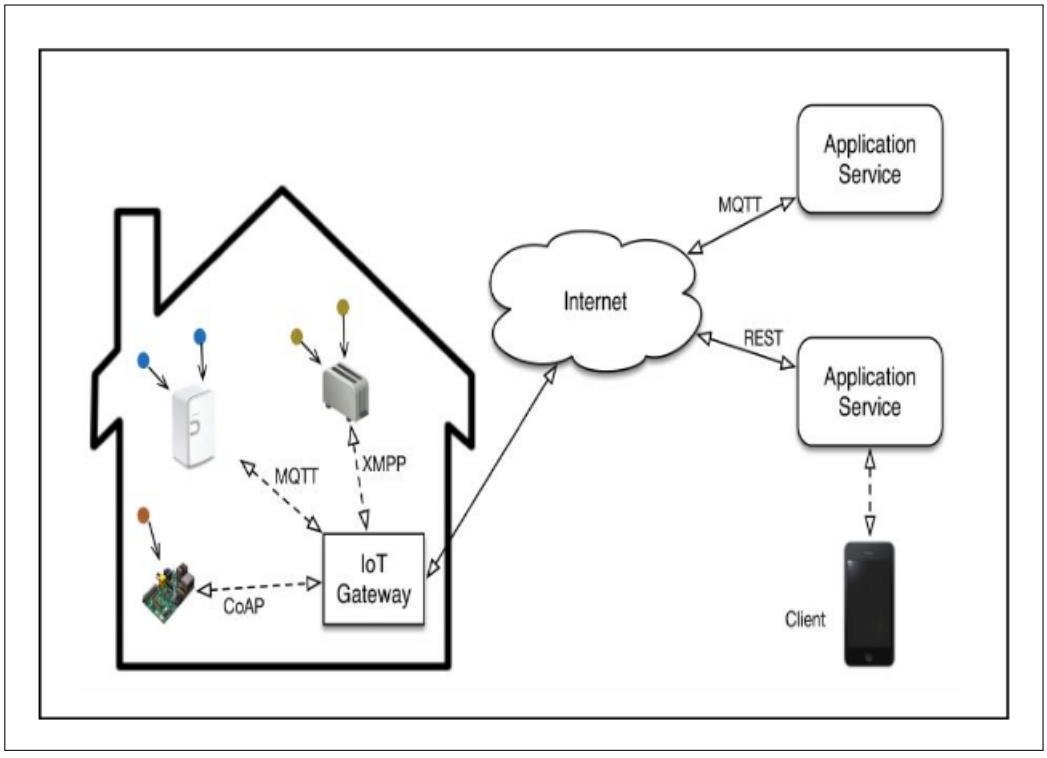
- Transporte inteligente: De carros conectados ou autônomos a transporte inteligente e sistemas de logística, a *IoT* pode salvar vidas, reduzir o tráfego e minimizar o impacto dos veículos no meio ambiente.
- Energia: Por meio da *IoT*, os vários dispositivos da rede de energia podem compartilhar informações em tempo real para distribuir energia de maneira mais eficiente.
- Assistência médica: Desde vestíveis clínicos até tablets para socorristas e equipamentos de conjunto cirúrgico sofisticado, a *IoT* está transformando os cuidados de saúde.
- Manufatura inteligente: A tecnologia de *IoT* permite que as fábricas atuais desbloqueiem a eficiência operacional, otimizem a produção e aumentem a segurança dos funcionários.
- Varejo: Para os varejistas, a *IoT* oferece oportunidades ilimitadas para aumentar

as eficiências da cadeia de suprimentos, desenvolver novos serviços e reformular a experiência dos clientes.

- Edifícios inteligentes: A *IoT* está abordando os crescentes custos de energia, a sustentabilidade e o cumprimento de códigos conectando, gerenciando e protegendo os dispositivos que coletam dados dos sistemas essenciais.
- Casas inteligentes: Desde o reconhecimento da sua voz até saber quem está batendo à porta, a tecnologia de *IoT* está tornando realidade o sonho de uma casa segura e inteligente.

Mas para isso, novos protocolos são necessários para compatibilidade de comunicação entre coisas heterogêneas, dentre os que já vigoram, estão o protocolo *MQTT*, que é responsável pelo envio de mensagens com sobrecarga mínima de transporte, o protocolo *CoAP*, que é usado para comunicação web, também os protocolos *6LoWPAN, RPL*, etc. A Figura 3 mostra um cenário *IoT*, exemplificando a comunicação de *motes* (*hardware* que você pode customizar e embutir sua aplicação, conectando assim qualquer coisa à Internet e ainda agregando o mínimo de custo possível.) com serviços de aplicações, através da utilização de alguns protocolos.

Figura 3 – Protocolos *IoT*



Fonte – Datta e Sharma (2017)

A simulação é a um dos conceitos centrais neste trabalho. Toda a pesquisa e desenvolvimento dos experimentos é voltado para essa área, com o objetivo de facilitar a assimilação dos conteúdos abordados na sala de aula nas disciplinas de Redes de Computadores.

3.2 Simulação

Os estudos de simulação auxiliam os projetistas na concepção de algoritmos de controle de tráfego a Qualidade de Serviço e também aperfeiçoam a relação custo/desempenho da rede (WIRTH, 1997). No desenvolvimento de um programa de simulação para rede de comunicações, há uma necessidade de modelar as demandas dos usuários aos recursos de rede, caracterizar os recursos exigidos para atender a essas demandas e estimar o desempenho baseado em dados de saída gerados pela simulação (FROST; MELAMED, 1994). Um simulador é um programa de computador que tenta reproduzir com uma maior fidelidade possível alguma experiência ou comportamento real (MINAKOV et al., 2016). Em outras palavras, simuladores envolvem a criação de uma história artificial do sistema, em que, por meio de observações, é possível retirar inferências sobre as características do sistema real. Esses modelos são elaborados tomando como base um conjunto de suposições em relação a operação do sistema. Tais suposições são representadas utilizando expressões matemáticas, lógicas e um relacionamento simbólico entre as entidades ou objetos de interesse compreendidos no sistema. Uma vez desenvolvido e validado, um modelo pode ser utilizado para investigar uma vasta diversidade de questionamentos sobre o sistema no mundo real (MARTINS, 2016). Exemplos de simuladores de redes são: *Cisco Packet Tracer; NS-2; NS-3; Cooja; OMNET++; TOSSIM*.

Apesar das muitas características positivas na simulação, ela fornece apenas uma modelagem matemática. Pontos de consumo de memória, utilização de processamento compartilhado, gargalos de troca de mensagens ou mesmo problemas ocultos no sistema operacional não podem ser simulados, simplesmente por não serem previsíveis. A emulação permite o uso de *hardware* simples, com poucos recursos para representar uma grande rede. Neste contexto da emulação, os *links*, processos e aplicações consomem de acordo com a necessidade e podem entregar de diversas formas possíveis resultados, diferenciados conforme sua execução (SILVA; MARQUES; LIMA, 2017).

4 SIMULAÇÃO EM *IOT*

Este capítulo compreende as tecnologias envolvidas e os conceitos empregados para o entendimento da proposta e seu desenvolvimento. Ele prevê cobrir todo conhecimento necessário para melhor compreensão de toda interpretação dos dados presentes na implementação e resultados.

A escolha dos simuladores foi baseada em uma pesquisa exploratória, através da leitura de artigos e surveys, onde pudemos identificar quais ferramentas mais se adequariam na aplicação do ensino de *IoT*, contribuindo ainda mais para um melhor entendimento do conteúdo abordado.

4.1 *Cooja*

Cooja é o simulador de rede do sistema operacional Contiki. O *Cooja* permite simular redes grandes e pequenas de motes Contiki. Os Motes podem ser emulados no nível de *hardware*, que é mais lento, mas permite uma inspeção precisa do comportamento do sistema, ou em um nível menos detalhado, que é mais rápido e permite a simulação de redes maiores (CONTIKIOS, 2017).

A *Cooja* é uma ferramenta útil para o desenvolvimento Contiki, pois permite aos desenvolvedores testar seus códigos e sistemas muito antes de executá-lo no *hardware* do destino. Os desenvolvedores estabelecem regularmente novas simulações tanto para depurar o *software* quanto verificar o comportamento de seus sistemas (CONTIKIOS, 2017).

O *ContikiOS* possui um sistema de compilação destinado a facilitar a execução do *Contiki* diretamente em *hardware*. O sistema de compilação foi projetado para ser o mesmo em diferentes plataformas de *hardware*, de modo que os comandos de compilação sejam familiares ao mudar de *hardware*.

Um Mote Contiki simulado no *Cooja* é um sistema Contiki compilado e executado. O sistema é controlado e analisado pelo *Cooja*. Isso é realizado compilando o sistema Contiki para a plataforma nativa como uma biblioteca compartilhada e carregando a biblioteca em Java usando *Java Native Interfaces (JNI)*. Várias bibliotecas diferentes do Contiki podem ser compiladas e carregadas na mesma simulação *Cooja*, representando diferentes tipos de nós de sensores (redes heterogêneas). O *Cooja* controla e analisa um sistema *Contiki* através de algumas funções. Por exemplo, o simulador informa o sistema Contiki para lidar com um evento ou busca

toda a memória do sistema *Contiki* para análise. Esta abordagem dá ao simulador o controle total de sistemas simulados. O mais significativo é a dependência de ferramentas externas, como compiladores e vinculadores, e seus argumentos de tempo de execução (CONTIKIOS, 2017).

Existem basicamente duas maneiras de se obter o simulador *Cooja*, instalando os pacotes necessários e usando o *ContikiOS* do repositório *Github*[1] ou através do download do Instant Contiki[2], que é uma máquina virtual criada com todas as cadeias de ferramentas e *software* necessários para o desenvolvimento do *ContikiOS*. Neste trabalho, optamos pela segunda opção.

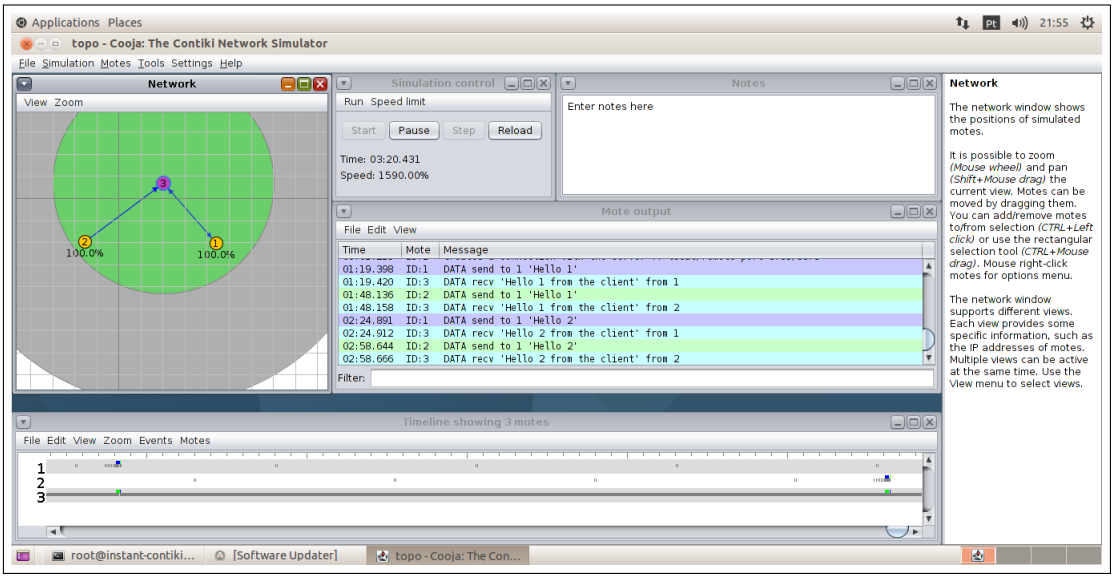
Após o download do Instant *Contiki*, você também precisa baixar o *VMWare Player*[3], que é um *software*/máquina virtual que permite a instalação e utilização de um sistema operacional dentro de outro dando suporte real a *software* de outros sistemas operacionais.

A forma simples de executar o *Cooja* é executá-lo dentro de seu próprio diretório:

```
1 cd contiki/tools/cooja
2 ant run
```

Depois de criar uma nova simulação, a janela do *Cooja* é preenchida com as principais ferramentas de simulação, conforme mostrado na Figura 4.

Figura 4 – Janelas para configuração de experimentos



Fonte – Elaborada pelo Autor

As principais funcionalidades de cada ferramenta:

1 <https://github.com/contiki-os/contiki>.
2 <https://sourceforge.net/projects/contiki/files/Instant%20Contiki/>.
3 <https://my.vmware.com/en/web/vmware/free#desktop_end_user_computing/vmware_workstation_player/14_0>.

- Rede - Mostra a localização de cada nó na rede. Pode ser usado para visualizar o status de cada nó, incluindo *LEDs, mote IDs*, endereços, saídas de lof, etc. Inicialmente, esta janela está vazia e precisa ser preenchida com sensores;
- Controle de simulação - Este painel é usado para iniciar, pausar, recarregar ou executar etapas da simulação. Mostra o tempo de execução e a velocidade da simulação. Isso significa que se pode executar os eventos várias vezes mais rápido do que seria necessário na execução em tempo real;
- Notas - Este é um bloco de notas simples para tomar notas sobre a simulação.;
- Saída de Mote - Mostra toda a saída de interface serial dos nós. É possível habilitar uma janela de saída de Mote para cada nó na simulação;
- Linha do tempo - Linha do tempo de simulação onde são exibidas mensagens e eventos como mudança de canal, LEDs, saídas de registro, etc.

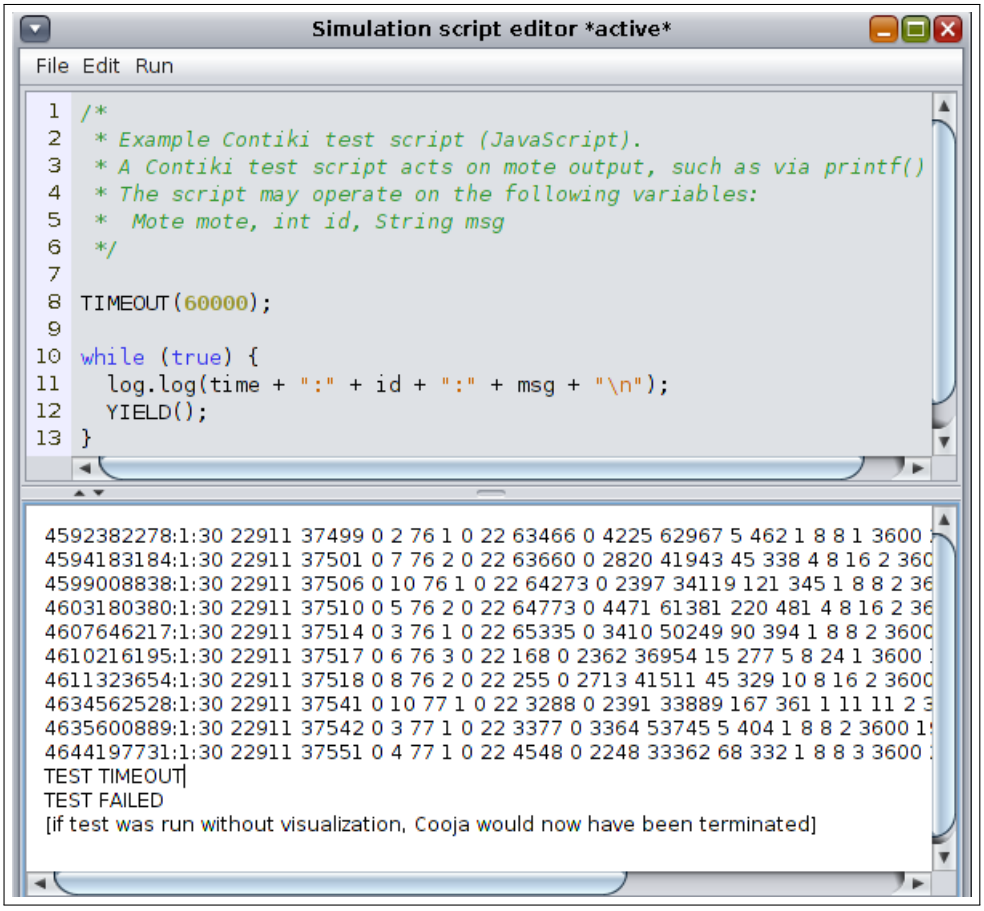
Além das ferramentas padrão, é possível exibir outras ferramentas, como pontos de interrupção, mensagens de rádio, editor de *scripts*, visualização de buffer e ciclo de uso de Mote, que podem ser ativadas no menu Ferramentas.

Os resultados obtidos nas simulações podem ser colhidos de duas maneiras essenciais, pela ferramenta *Script Editor* ou pela ferramenta *Sensor Collect*, que serão descritos a seguir.

4.1.1 Script Editor

O *Script Editor* pode ser usado apenas para exibir mensagens e para definir um temporizador na simulação. Para fazer isso, dentro do menu Editor de *scripts*, selecione o menu suspenso 'Arquivo' do qual 'Carregar script de exemplo' é a única opção. Em seguida, selecione 'Basta registrar todos os printf()' e timeout. Isso resulta em um *script* simples como na Figura 5, sendo o maior ponto de interesse o *TIMEOUT*. Este número é exibido em milissegundos, portanto o 60000 mostrado é de 60 segundos, ou 1 minuto. Isso pode ser ajustado em conformidade.

Figura 5 – Editor de Script de Simulação



Fonte – Elaborada pelo Autor

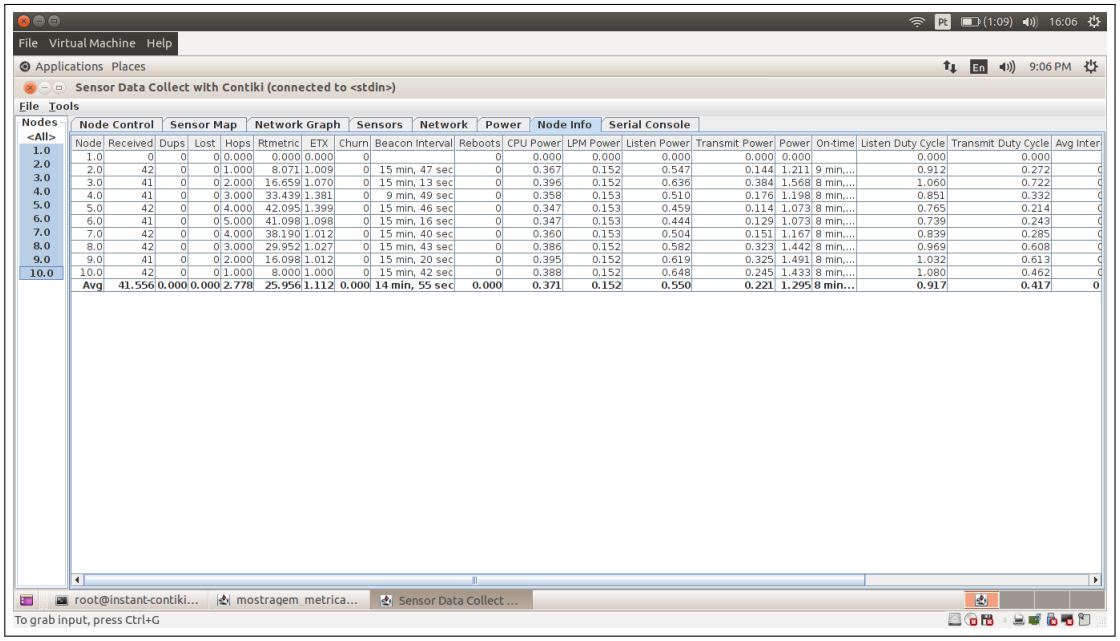
4.1.2 Sensor Collect

A outra forma essencial de se obter os resultados dos experimentos no simulador *Cooja* é através do *Sensor Collect*, Na janela 'Create Mote Type', escolha um nome adequado para o tipo de mote. Na opção 'Contiki process/Firmware', siga o caminho examples/collect e selecione o programa 'collect-view-shell.c'. Compile e clique em Criar.

Após iniciar a simulação, clique com o botão direito do mouse em qualquer mote na janela 'Network', clique em 'Mote tools <número do mote criado>' e selecione 'Collect View'. A aplicação Collect View será iniciada.

A Figura 6 mostra a janela que reúne um compilado de diversas métricas, para todos os motes simulados, como a quantidade de pacotes recebidos, perdidos, *Beacon Interval*, etc .

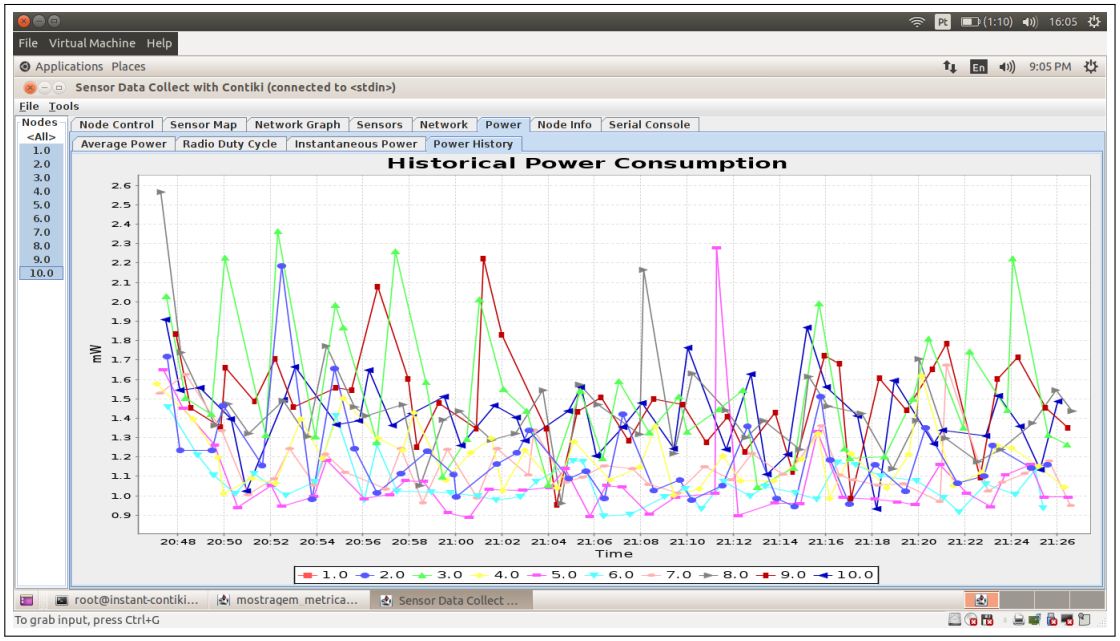
Figura 6 – Informação de Todos os Motes Simulados



Fonte – Elaborada pelo Autor

Na Figura 7, é mostrado o gráfico de histórico de consumo de energia disponível na GUI do Cooja.

Figura 7 – Histórico de Consumo de Energia



Fonte – Elaborada pelo Autor

A Tabela 2 mostra as métricas fornecidas pela GUI do simulador Cooja, divididos em Gráficos Topológicos, Gráficos sujeitos à disponibilidade de um determinado tipo de sensor

nos motes, Gráficos relacionados com métricas de rede, Métricas de Energia e Outros dados.

Tabela 2 – Métricas disponíveis em gráficos no simulador Cooja

Gráficos Topológicos	Disponibilidade À um determinado tipo de sensor	Métricas de rede	Métricas de Energia	Outros dados
- Mapa de Sensores	- Sensor de Temperatura	- Vizinhos	- Consumo médio	- Node Info
- Gráfico de Rede	- Sensor de umidade	- Intervalo	- Consumo Instantâneo	- Serial Console
- Sensores Presentes nos Motes	- Sensor de bateria	- Beacon	- Histórico de consumo	
	- Sensor de luz	- Saltos de Rede	- Ciclo de funcionamento por nó	
		- Métricas do Roteador - ETX - Próximo salto - Latência - Pacotes Perdidos - Pacotes Recebidos		

Fonte – Elaborada pelo Autor

Embora o *Cooja* mostre-se uma ótima opção para simulação em *IoT*, há desafios envolvidos em seu uso, particularmente em relação à falta de documentação. A documentação oficial não abrange mais do que uma configuração básica de rede e uma explicação superficial sobre a ferramenta. Para um entendimento mais profundo, o usuário terá que ficar a mercê de fóruns na internet e da lista de e-mails dos desenvolvedores *Contiki*.

4.2 *NS-3*

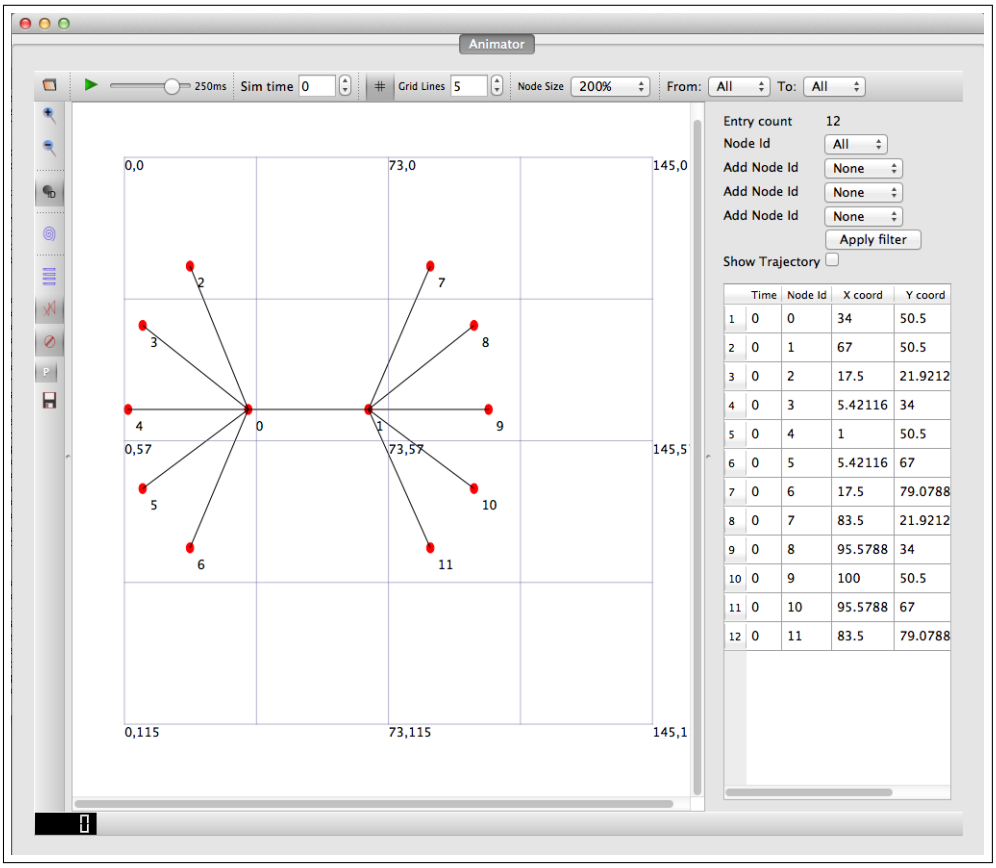
O *NS-3* foi desenvolvido para fornecer uma plataforma de simulação de rede aberta e extensível, para pesquisa em rede e educação. Em resumo, *NS-3* fornece modelos de como as redes de dados de pacotes funcionam, executam e fornecem um mecanismo de simulação para que os usuários conduzam experimentos de simulação. Algumas das razões para usar *NS-3* incluem realizar estudos que são mais difíceis ou impossíveis de realizar com sistemas reais, estudar o comportamento do sistema em um ambiente controlado e reprodutível e aprender sobre como as redes funcionam (NS3, 2017b).

O *NS-3* foi criado como um sistema de bibliotecas de *software* que trabalham em conjunto. Os programas são escritos (ou importados) nessas bibliotecas. O *NS-3* é distribuído

como código-fonte, o que significa que o sistema de destino precisa ter um ambiente de desenvolvimento de *software* para construir as bibliotecas primeiro e, em seguida, criar o programa do usuário (NS3, 2017a).

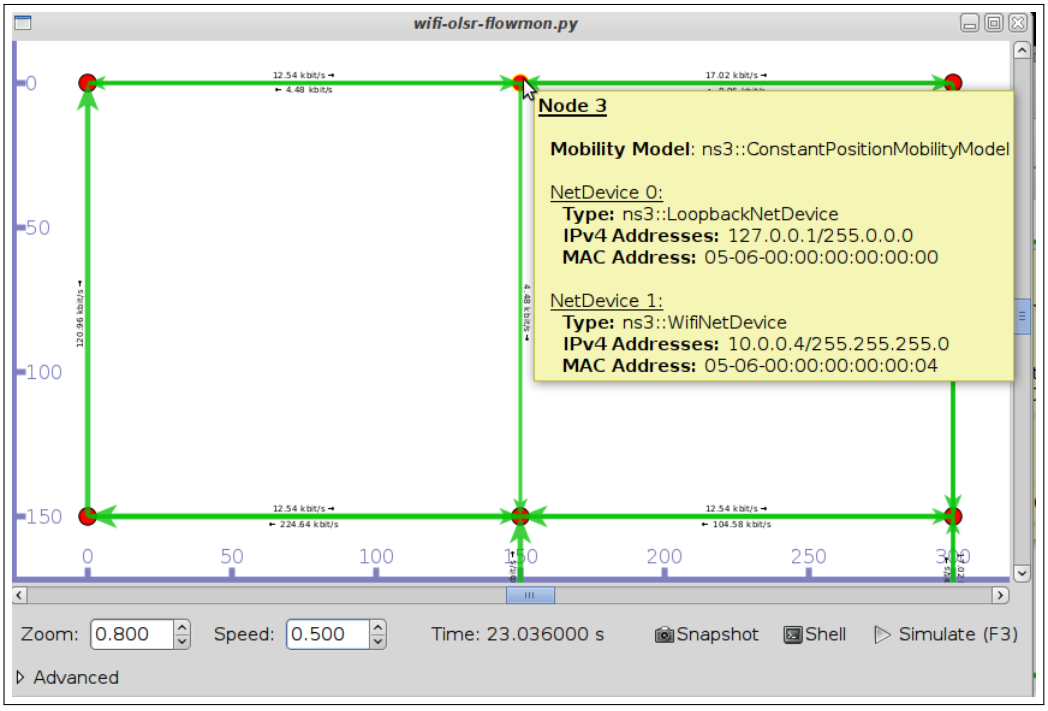
Diferente do simulador *Cooja*, citado anteriormente, o *NS-3* não contém uma ferramenta de animação gráfica padrão, mesmo assim, existem duas maneiras de prover animação gráfica para as simulações, usando o método PyViz ou o método NetAnim. O NetAnim é um animador *offline* baseado no kit de ferramentas Qt. Ele atualmente anima a simulação usando um arquivo de rastreamento XML coletado durante a simulação. Já o PyViz, é um visualizador de simulação em tempo real, o que significa que ele não usa arquivos de rastreamento. Pode ser mais útil para fins de depuração, ou seja, descobrir se os modelos de mobilidade são o que você espera, onde os pacotes estão sendo descartados, etc. Há também um console python interativo que pode ser usado para depurar o estado dos objetos em execução. Embora seja principalmente escrito em Python, ele funciona tanto com Python quanto com simulações em C++. As Figuras 8 e 9 mostram a GUI NetAnim e PyViz, respectivamente.

Figura 8 – NetAnim



Fonte – NETANIM (2017)

Figura 9 – PyViz



Fonte – Elaborada pelo Autor

A programação no NS-3 é feita em C++ ou Python, para esse trabalho, optamos por usar a linguagem C++ para a configuração dos experimentos. A partir do NS-3.2, a maioria das APIs já estão disponíveis em Python, mas os modelos continuam sendo escritos em C++. O NS-3 utiliza vários componentes do conjunto de ferramentas GNU — "GNU toolchain" — para o desenvolvimento. Um *software* toolchain provêm de uma ampla coleção de ferramentas de programação produzidas pelo Projeto GNU. O NS-3 usa o gcc, GNU binutils e gdb. Porém, não usa as ferramentas GNU para compilar o sistema, nem o Make e nem o Autotools. Para estas funções é utilizado o Waf, que é um framework baseado em Python para configurar, compilar e instalar aplicativos..

Os códigos fonte do NS-3 estão disponíveis através dos repositórios do Mercurial no servidor http://code.nsnam.org. Os fontes compactados podem ser obtidos em http://www.nsnam.org/releases/ (NS3, 2017b).

O código NS-3 está disponível nos repositórios Mercurial no servidor http://code.nsnam.org . O download também pode ser realizado em uma versão de tarball em http://www.nsnam.org/release/ (maneira escolhida para a realização deste trabalho), ou você pode trabalhar com repositórios usando o Mercurial.

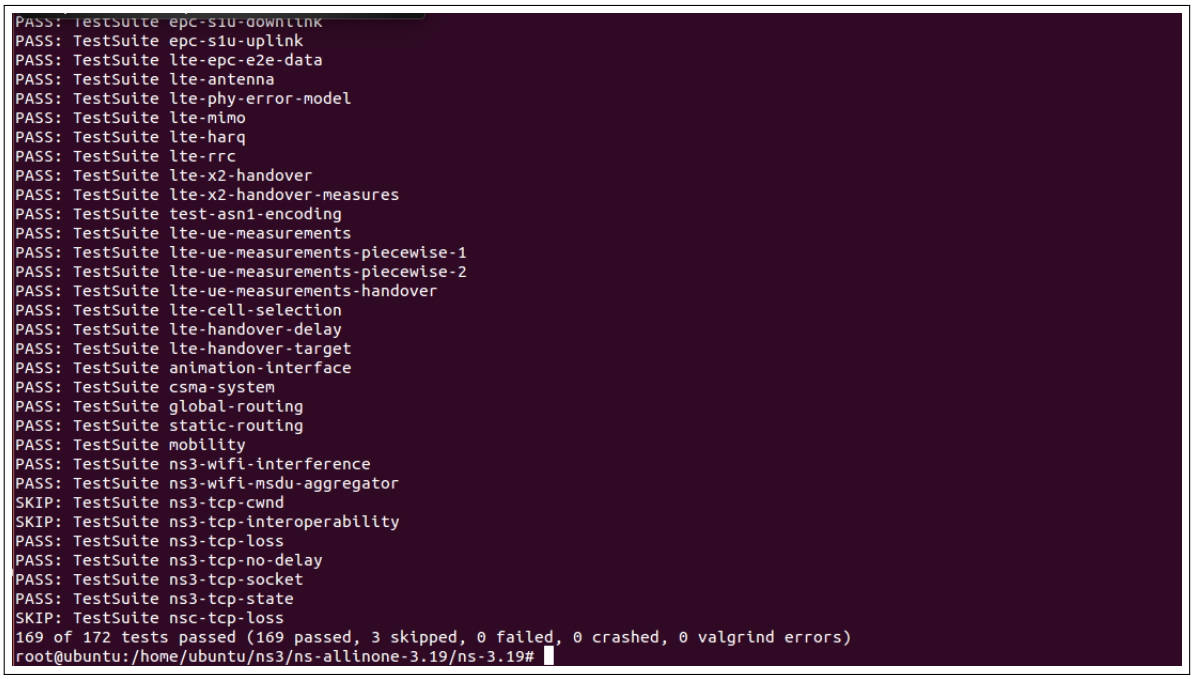
Após a instalação correta do NS-3, utilizamos o seguinte comando para verificar as

várias dependências necessárias para o funcionamento do simulador:

```
./test.py -c core
```

Estes testes são executados em paralelo pelo Waf. No final, você deverá receber uma mensagem como mostrada na Figura 10.

Figura 10 – Resultados do Testes realizados pelo Waf



Fonte – Elaborada pelo Autor

A documentação NS-3 é mantida usando o Doxygen[4]. O Doxygen é normalmente usado para documentar API e organiza tal documentação em diferentes módulos. O NS-3 usa o Doxygen para construir a documentação definitiva da API mantida. A documentação adicional do projeto NS-3 pode ser encontrada no site[5] do projeto. O script a seguir mostra a configuração, em C++, de uma simulação básica no NS-3.

```
#include "ns3/core-module.h"
NS_LOG_COMPONENT_DEFINE ("HelloSimulator");
using namespace ns3;
int main (int argc, char *argv[])
{
  NS_LOG_UNCOND ("Hello Simulator");
}
```

4 <http://www.stack.nl/~dimitri/doxygen/>.
5 <https://www.nsnam.org/ns-3-27/documentation/>.

Os resultados das simulações feitas no NS-3 são obtidos através de mensagens de registro, ou seja, o NS-3 permite que o usuário tenha visões de todos os níveis do sistema através dessas mensagens. O sistema de registro deve ser usado para depurar informações, alertas, mensagens de erros ou para mostrar qualquer informação dos *scripts* ou modelos.

Atualmente existem sete níveis de mensagens de registro definidas no sistema, são eles:

- NS_LOG_ERROR — Registra mensagens de erro;
- NS_LOG_WARN — Registra mensagens de alertas;
- NS_LOG_DEBUG — Registra mensagens mais raras, mensagens de depuração ad-hoc;
- NS_LOG_INFO — Registra mensagens informativas sobre o progresso do programa;
- NS_LOG_FUNCTION — Registra mensagens descrevendo cada função chamada;
- NS_LOG_LOGIC — Registra mensagens que descrevem o fluxo lógico dentro de uma função;
- NS_LOG_ALL — Registra tudo.

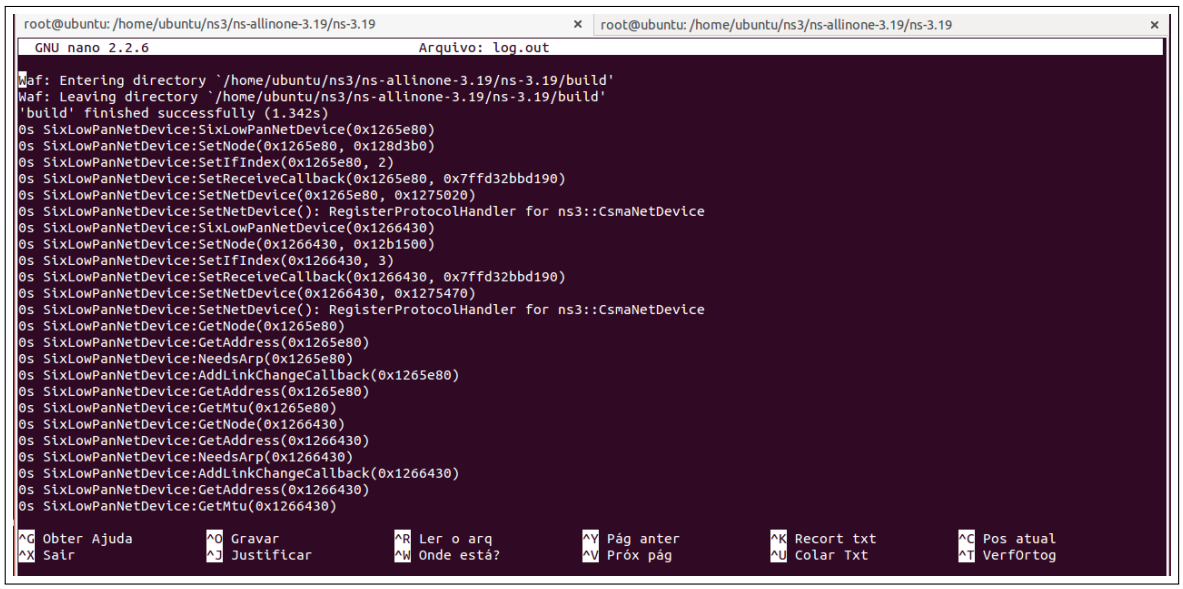
Também é fornecido um nível de registro incondicional, que sempre é exibido independente do nível de registro ou do componente selecionado:

- NS_LOG_UNCOND — Registra mensagens incondicionalmente.

Cada nível pode ser requerido individualmente ou de forma cumulativa. O registro pode ser configurado usando uma variável de ambiente (NS_LOG) ou através de uma chamada ao sistema de registro. Os resultados podem ser expostos na tela ao fim da simulação ou serem salvos em um arquivo de log.

A Figura 11 mostra um exemplo de arquivo de log no NS-3. Nele, podemos observar a saída da depuração da simulação.

Figura 11 – Log de Resultados no NS-3



Fonte – Elaborada pelo Autor

Outra forma de se obter os resultados das simulações é através do sistema de rastreamento disponível no NS-3.

O sistema de rastreamento do NS-3 é feito através de conceitos independentes de rastreamento na origem e no destino, e um mecanismo uniforme para conectar a origem ao destino. O rastreador na origem são entidades que podem demonstrar eventos que ocorrem na simulação e fornece acesso aos dados importantes. Por exemplo, um rastreador de origem podem indicar quando um pacote é recebido por um dispositivo de rede e provê acesso aos comentários de pacote para os interessados no rastreador do destino.

Além de não possuir interface gráfica por padrão, o NS-3 também não disponibiliza as métricas previamente calculadas em forma de gráfico, como no simulador Cooja. Existem dois tipos de de sistemas de rastreamento disponíveis no NS-3, o Rastreamento ASCII e o Rastreamento PCAP, o primeiro habilita as saídas em arquivos ASCII (texto puro), o segundo permite a captura de pacotes e é uma API que inclui a descrição de um arquivo no formato .pcap. Com isso, torna-se necessário a formulação de *scripts* (pelos usuários) para processar os traces e gerar as métricas (largura de banda, latência, etc.), a partir desses arquivos.

Visto que a configuração das simulações devem ser feitas através da programação de exemplos, utilizando as linguagens C++ ou Python, o uso desta ferramenta poderá ser um empecilho para alguns usuários, pois a documentação oficial considera que o leitor possua conhecimento básico nessas linguagens, além do domínio de conceitos de orientação a objetos.

Porém, a documentação do NS-3 torna-se abrangente e detalhada, pois além de apresentar trechos do código fonte para os interessados em aprofundar-se no funcionamento do sistema, a documentação ainda consta com mais de 100 páginas, que demonstra desde o necessário para realizar o download da ferramenta até conceitos mais avançados, como classes de rastreamento de objetos. Além disso, o NS-3 consta com uma comunidade de desenvolvedores participativos, onde se pode encontrar tutoriais com várias resoluções para um problema específico.

Então, com uma leitura completa da documentação, um usuário que tenha o conhecimento básico de programação deve conseguir realizar a configuração de um experimento de acordo com suas necessidades

É importante salientar nesta pesquisa que o simulador NS-3 não é voltado estritamente para *IoT*, como os outros dois simuladores estudados, ele apenas provê suporte para algumas tecnologias na área.

4.3 TOSSIM

O TOSSIM é um simulador de eventos discreto para redes de sensores TinyOS. Em vez de compilar um aplicativo TinyOS para um mote, os usuários podem compilá-lo no framework TOSSIM, que é executado em um PC. Isso permite aos usuários depurar, testar e analisar algoritmos em um ambiente controlado e repetitivo. À medida que o TOSSIM é executado em um PC, os usuários podem examinar seu código TinyOS usando depuradores e outras ferramentas de desenvolvimento.(LEVIS; LEE, 2003).

TOSSIM é um simulador TinyOS, ele é obtido juntamente com a instação[6] do seu SO hospedeiro. Seu código principal fica em tos/lib/tossim. Todo diretório de origem TinyOS possui um subdiretório opcional denominado sim/, que pode conter implementações de simulação desse pacote. Por exemplo, *tos/chips/atm128/timer/sim* contém implementações TOSSIM de algumas das abstrações do temporizador Atmega128[7], você pode utilizar essas implementações para simular uma rede que utilize aquela determinada tecnologia.

Após a instalação do TinyOS, você terá acesso a pasta de aplicações do SO. O TOSSIM é construído automaticamente quando você compila um aplicativo. Os aplicativos são compilados entrando em um diretório de aplicativo (por exemplo /apps/Blink) e digitando:

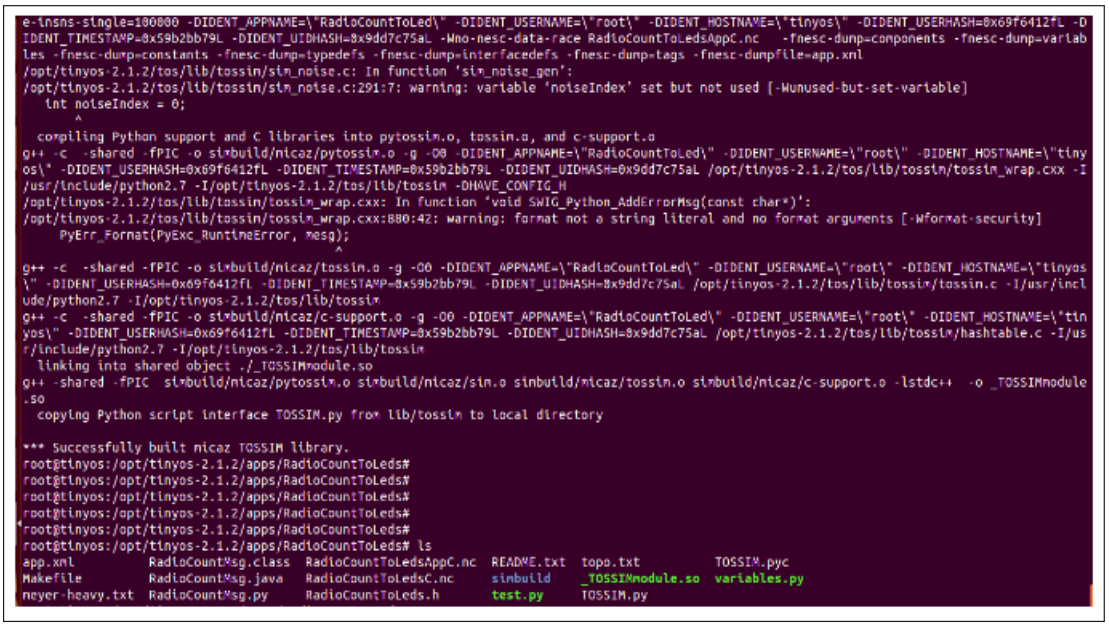
6 <http://tinyos.stanford.edu/tinyos-wiki/index.php/Automatic_installation>.
7 <http://www.atmel.com/images/doc2467.pdf>.

```
make micaz sim
```

MICAz é um mote da tecnologia Crossbow[8]. É composto do microcontrolador ATmega128L e do chip de rádio CC2420[9]. Atualmente, o micaz é a única plataforma suportada pelo TOSSIM (TOSSIM, 2013).

Após compilar o aplicativo definido, o sistema deverá retornar a seguinte saída, de acordo com a Figura 12.

Figura 12 – Saída da compilação do simulador TOSSIM



Fonte – Elaborada pelo Autor

Podemos observar a criação do diretório simbuild/ após a compilação do aplicativo, neste diretório encontra-se todos os arquivos necessários para a simulação. O TOSSIM suporta duas interfaces de programação: Python e C ++. Neste trabalho, optamos por usar Python.

Como falado anteriormente, TOSSIM é uma biblioteca: você deve escrever um programa que configure uma simulação e a execute. Ex:

```
>>> from TOSSIM import *
>>> t = Tossim([])
>>> t.runNextEvent()
0
```

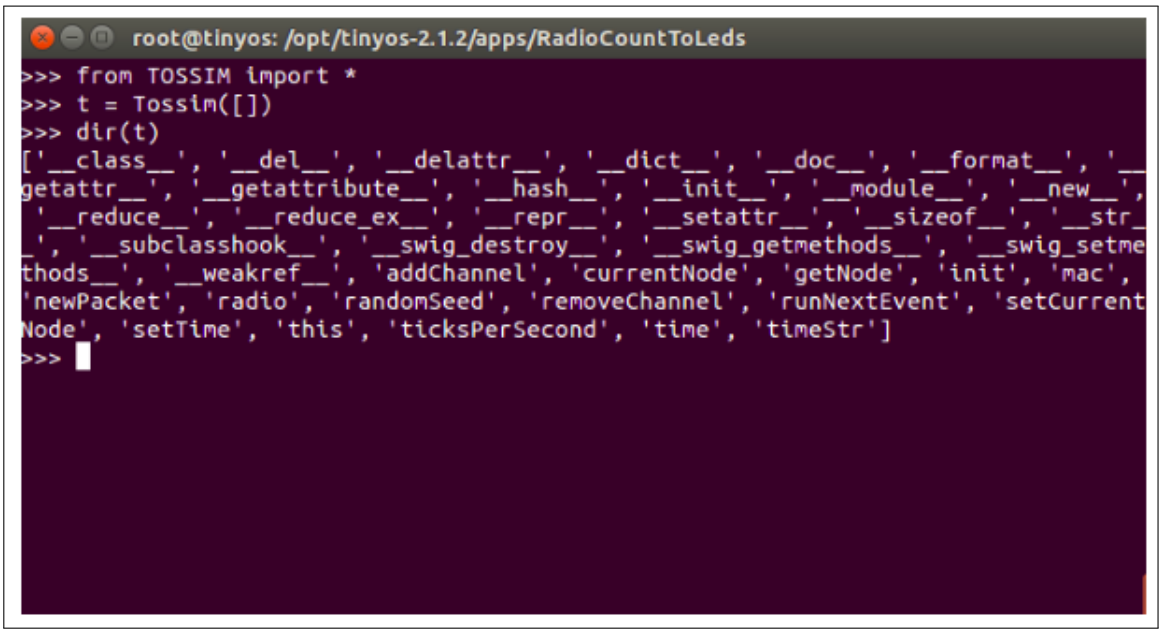
8 <http://www.openautomation.net/uploadsproductos/micaz_datasheet.pdf>.
9 <http://www.ti.com/product/CC2420>.

No código acima, é importada a biblioteca, criado um objeto TOSSIM e executa uma simulação TOSSIM com a função runNextEvent. Quando você informa ao TOSSIM para executar o próximo evento, ele retorna 0. Isso significa que não houve nenhum próximo evento a ser executado. No nosso caso, não há um próximo evento, porque não configuramos a inicialização de nenhum nó.

```
1 >>> m = t.getNode(32)
2 >>> m.bootAtTime(45654)
3 >>> t.runNextEvent()
4 1
```

Agora, runNextEvent retorna 1, porque houve um evento para executar. Um objeto Tossim possui várias funções úteis. Em Python, geralmente você pode ver a assinatura de um objeto com a função dir, demonstrado na Figura 13.

Figura 13 – Assinatura do objeto



Fonte – Elaborada pelo Autor

Métodos com duplo sublinhado, "_ _", na frente e na parte de trás, são, geralmente, funções internas. Por exemplo, __init__ é chamado internamente durante a criação de um objeto.

As funções mais comuns são:

- currentNode(): Retorna a ID do nó atual.
- getNode(id): Retorna um objeto que representa um nó específico
- runNextEvent(): Execute um evento de simulação

- time(): Retorna a hora atual em ticks de simulação como um inteiro
- timeStr(): Retorna uma representação de string do tempo atual
- init(): inicializa o TOSSIM
- mac(): Retorna o objeto que representa a camada de acesso à mídia
- radio(): Retorna o objeto que representa o modelo de rádio
- ticksPerSecond():Retorna quantos ticks de simulação existem em um segundo simulado.

Os resultados das simulações podem ser obtidos através de Debugging Statements, chamado dbg. Existem quatro chamadas dbg:

- dbg: Imprime uma declaração de depuração precedida do ID do nó.
- dbg_clear: Imprime uma declaração de depuração que não é precedida pelo ID do nó. Isso permite que você imprima facilmente tipos de dados complexos, como pacotes, sem intercalar IDs de nós através da saída.
- dbgerror: Imprime uma declaração de erro precedida pelo ID do nó
- dbgerror_clear: Imprime uma declaração de erro não precedida pela ID do nó

Neste script, é simulada a transmissão de pacotes entre motes em uma rede de sensores. Primeiramente, esse script irá carregar um arquivo (topo.txt) que especifica cada link de transmissão como uma linha com três valores, a origem, o destino e dBm, por exemplo. Isso será usado para definir qual nó se comunicará com o outro nó, em um determinado momento da simulação:

```
1  #! /usr/bin/python
2  from TOSSIM import *
3  import sys
4
5  t = Tossim([])
6  r = t.radio()
7  f = open("topo.txt", "r")
8
9  for line in f:
10   s = line.split()
11   if s:
12     print " ", s[0], " ", s[1], " ", s[2];
13     r.add(int(s[0]), int(s[1]), float(s[2]))
14
15 t.addChannel("RadioCountToLedsC", sys.stdout)
16 t.addChannel("Boot", sys.stdout)
17
18 noise = open("meyer-heavy.txt", "r")
19 for line in noise:
```

```
20    str1 = line.strip()
21    if str1:
22      val = int(str1)
23      for i in range(1, 4):
24        t.getNode(i).addNoiseTraceReading(val)
25
26  for i in range(1, 4):
27    print "Creating noise model for ",i;
28    t.getNode(i).createNoiseModel()
29
30  t.getNode(1).bootAtTime(100001);
31  t.getNode(2).bootAtTime(800008);
32  t.getNode(3).bootAtTime(1800009);
33
34  for i in range(100):
35    t.runNextEvent()
```

A Figura 14 mostra o resultado do script descrito acima.

Figura 14 – Resultado do script completo para simular a transmissão de pacotes entre motes

```
root@tinyos:/opt/tinyos-2.1.2/apps/RadioCountToLeds# ./test.py
  1   2   -54.0
  2   1   -55.0
  1   3   -60.0
  3   1   -60.0
  2   3   -64.0
  3   2   -64.0
Creating noise model for  1
Creating noise model for  2
Creating noise model for  3
DEBUG (1): Application booted.
DEBUG (1): Application booted again.
DEBUG (1): Application booted a third time.
DEBUG (2): Application booted.
DEBUG (2): Application booted again.
DEBUG (2): Application booted a third time.
DEBUG (3): Application booted.
DEBUG (3): Application booted again.
DEBUG (3): Application booted a third time.
DEBUG (1): RadioCountToLedsC: timer fired, counter is 1.
DEBUG (1): RadioCountToLedsC: packet sent.
DEBUG (2): RadioCountToLedsC: timer fired, counter is 1.
DEBUG (2): RadioCountToLedsC: packet sent.
DEBUG (3): RadioCountToLedsC: timer fired, counter is 1.
DEBUG (3): RadioCountToLedsC: packet sent.
DEBUG (2): Received packet of length 2.
DEBUG (1): Received packet of length 2.
DEBUG (3): Received packet of length 2.
DEBUG (2): Received packet of length 2.
DEBUG (3): Received packet of length 2.
DEBUG (1): Received packet of length 2.
DEBUG (1): RadioCountToLedsC: timer fired, counter is 2.
DEBUG (1): RadioCountToLedsC: packet sent.
DEBUG (2): RadioCountToLedsC: timer fired, counter is 2.
DEBUG (2): RadioCountToLedsC: packet sent.
DEBUG (3): RadioCountToLedsC: timer fired, counter is 2.
DEBUG (3): RadioCountToLedsC: packet sent.
```

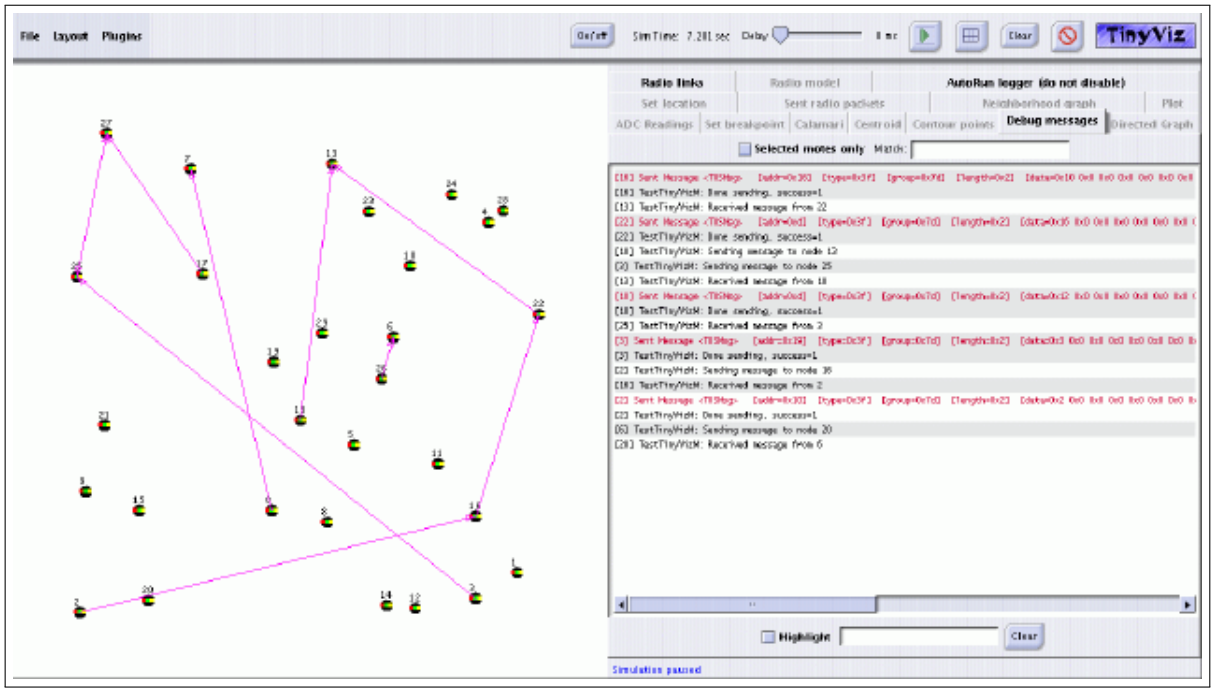
Fonte – Elaborada pelo Autor

Semelhante ao NS-3, o TOSSIM também não provê uma GUI por padrão, isso pode ser obtido através de um arquivo em Java. TinyViz é um ambiente de visualização e atuação de

Java para o TOSSIM. A principal classe TinyViz é um arquivo jar, que pode ser encontrado em tools/java/net/tinyos/sim/tinyviz.jar. TinyViz pode ser anexado a uma simulação em execução. Além disso, pode-se configurar o TOSSIM para esperar que TinyViz se conecte, antes de iniciar a simulação. Isso permite aos usuários ter certeza de que o TinyViz captura todos os eventos em uma determinada simulação.

TinyViz não é realmente um visualizador, em vez disso, é uma estrutura na qual os plugins podem fornecer a funcionalidade desejada. Por si só, TinyViz faz pouco além de desenhar motes e seus LEDs. No entanto, ele vem com alguns exemplos de plugins, como um que visualiza o tráfego de rede. a Figura 15 mostra a GUI TinyViz em execução.

Figura 15 – TinyViz



Fonte – Scott (2014)

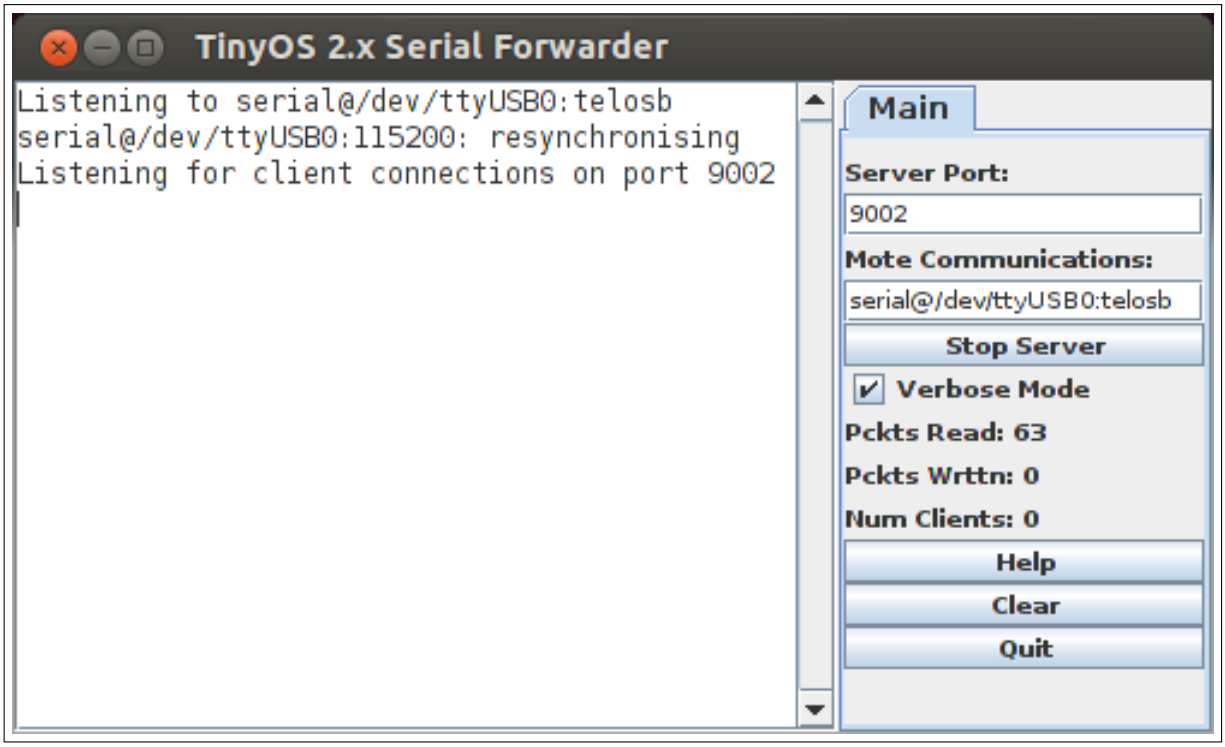
Para interagir com uma rede simulada e obter algumas métricas da simulação, deve-se usar o SerialForwarder, a ferramenta de interface TinyOS padrão. Para trabalhar com o TOSSIM, a fonte de entrada do SerialForwarder deve ser configurada adequadamente. O TOSSIM fornece dois modos: comunicação através de uma porta serial para o mote 0 e snooping da rede.

No modo de porta serial ("tossim-serial"no campo "Mote Communications"ou -comm tossim-serial"na linha de comando) a aplicação interage com o mote 0 em sua porta serial. Os programas que se conectam ao SerialForwarder podem ler as mensagens enviadas para a sua porta serial e enviar mensagens para o mote 0 em sua porta serial.

O modo Snooping ("tossim-radio"no campo "Mote Communications"ou -comm tossim-radio"na linha de comando) fica no topo do modelo de rede TOSSIM. Os programas que se conectam ao SerialForwarder ouvem todas as mensagens de rádio enviadas na rede e podem injetar mensagens de rádio para chegar (sem erro) em qualquer mote. Como este modo emite cada mensagem enviada, não considera perda; A conexão de programas irá ouvir pacotes que podem não chegar com sucesso em qualquer momento.

A Figura 16 mostra a aplicação SerialForwarder em uso.

Figura 16 – Serial Forwarder



Fonte – Scott (2014)

Outra forma de se obter as métricas no simulador TOSSIM, é através da criação de *scripts*, bem semelhante ao simulador NS-3. Esses *scripts* podem ser configurados para ler instruções *dbg* salvas em arquivos e formular métricas para análise do experimento, como perda de pacotes, latência, etc.

Semelhante ao NS-3, o TOSSIM também provê uma documentação que abrange desde a criação de um mote até os métodos de coleta de resultados. Porém, não chega a detalhar cada classe e métodos como no Doxygen do NS-3.

5 AVALIAÇÃO DAS FERRAMENTAS DE SIMULAÇÃO

Neste capítulo é apresentada uma avaliação qualitativa das ferramentas, demonstrando os pontos positivos e negativos, bem como a configuração de um mesmo cenário básico em cada um dos simuladores.

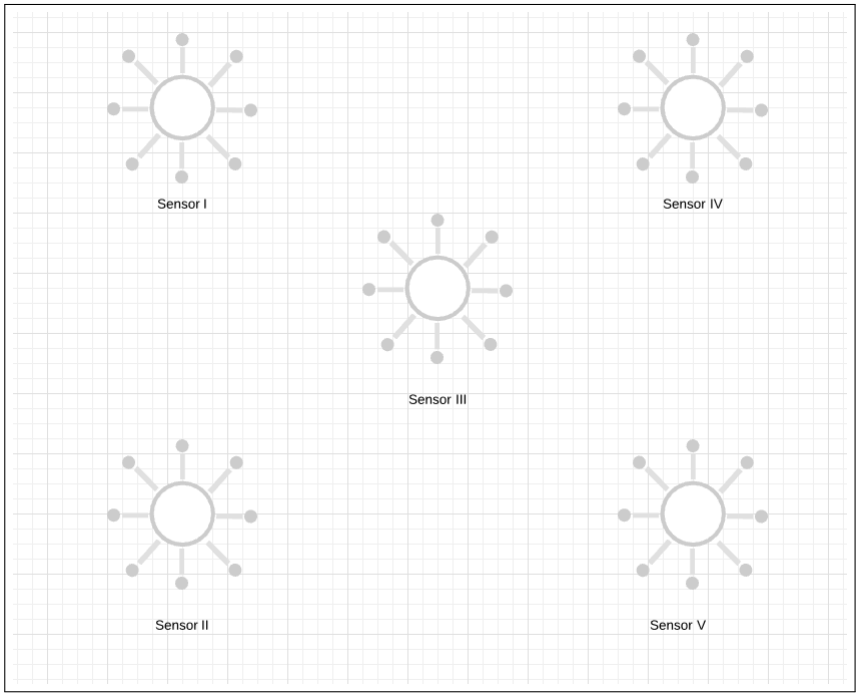
5.1 Executando um Cenário Simples

Nesta seção, relatamos a experiência da criação de uma topologia básica em cada um dos simuladores, enfatizando a heterogeneidade no aprendizado em cada uma das ferramentas.

Na Figura 17 é mostrada a topologia básica escolhida para a criação das simulações, nela é proposta uma rede de sensores com 5 dispositivos interligados, que se comunicam com protocolos IoT. Esta topologia será usada como base para a criação das simulações.

Uma grande diferença nos simuladores, a respeito da configuração dessa topologia, é a maior complexidade de se configurar a comunicação entre os motes. Enquanto no Cooja, a comunicação é feita de modo automático, no TOSSIM e no NS-3, toda a comunicação deve ser criada via código, informando desde o protocolo que será utilizado para envio dos pacotes até mesmo qual nó será o emissor e qual será o receptor. Visto isso, a escolha do simulador que será utilizado na disciplina, fica a cargo do professor, observando qual mais se adéqua a ementa da disciplina e ensino na sala de aula.

Figura 17 – Topologia Básica

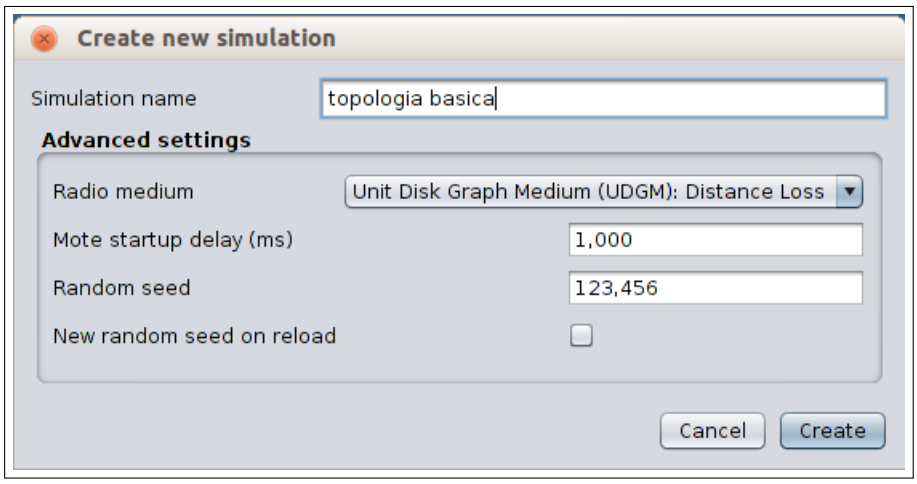


Fonte – Elaborada pelo Autor

5.2 Simulação Cooja

Com o auxílio da GUI do simulador Cooja, a configuração de uma experimento torna-se bem simples e intuitiva, já que o controle de toda a simulação pode ser feita através da interface gráfica, como dito na sessão de apresentação das ferramentas. Para reproduzir a topologia abordada o iníçio desta seção basta criar uma nova simulação, como descrito na Figura 18.

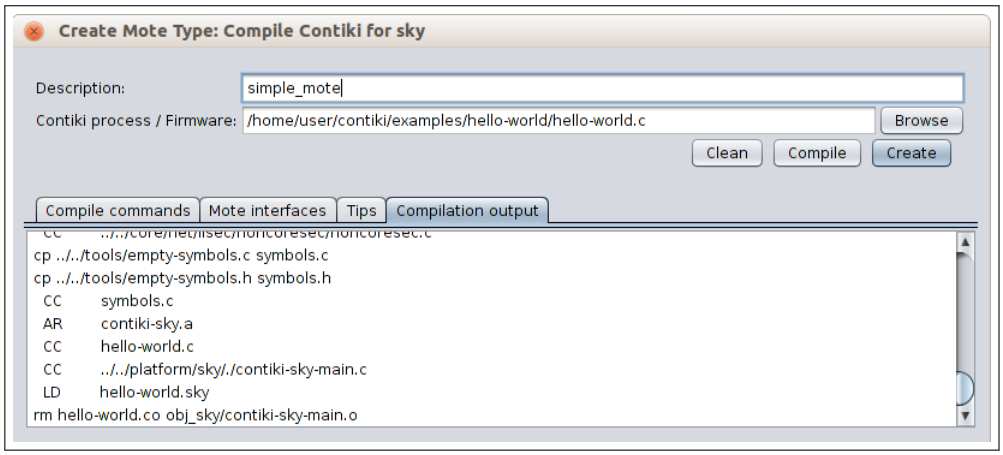
Figura 18 – Criação de uma nova simulação



Fonte – Elaborada pelo Autor

Após a criação da simulação, partimos para a criação dos motes, onde podemos escolher entre os diversos motes disponíveis no simulador Cooja e compilar a aplicação necessária para a simulação. Isso é demonstrado na Figura 19.

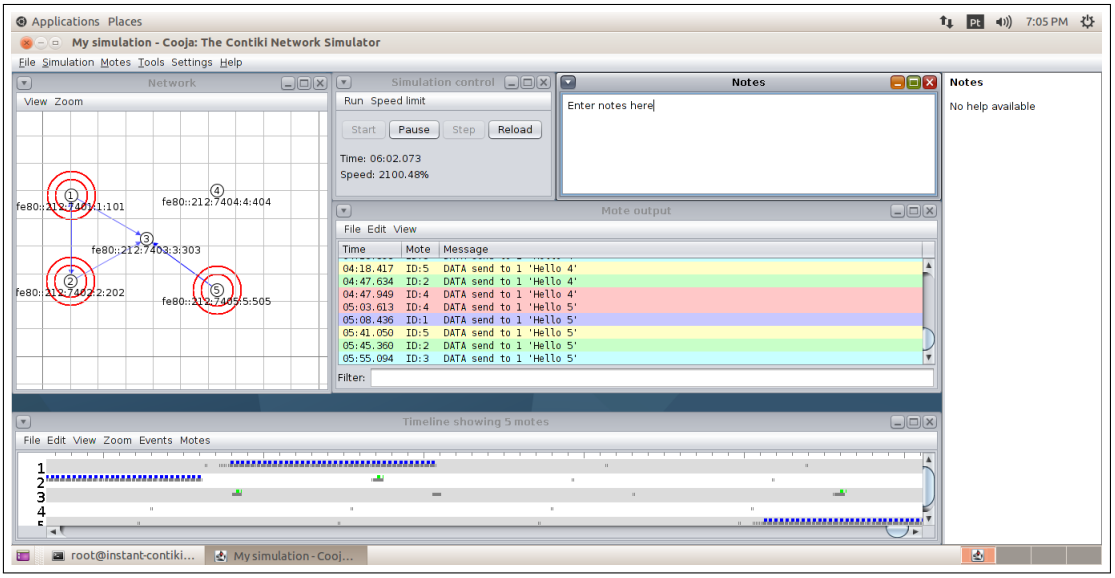
Figura 19 – Criação de um mote



Fonte – Elaborada pelo Autor

Já com a simulação criada, podemos iniciar o experimento, onde é possível observar os resultados, tráfego de pacotes, pausar, exportar os resultados, configurar os motes durante a simulação, etc. A figura 20 mostra uma simulação em andamento.

Figura 20 – Simulação em Andamento



Fonte – Elaborada pelo Autor

Como visto, a interface gráfica do simulador Cooja é robusta, contendo funcionalidades suficientes para que não seja necessária praticamente nenhuma configuração no código fonte das aplicações para a realização de uma simulação.

5.3 Simulação NS-3

Neste trabalho, optamos por criar a simulação a partir de um *script* de exemplo já existente no simulador.Nele, podemos observar a comunicação entre os motes utilizando alguns protocolos *IoT*, como 6LoWPAN, ICMPv6, etc. Por fim, exporta-se os resultados para arquivos ASCII e PCAP, que serão usados para obter as métricas para a formulação dos resultados.

```
1  #include <fstream>
2  #include "ns3/core-module.h"
3  #include "ns3/internet-module.h"
4  #include "ns3/sixlowpan-module.h"
5  #include "ns3/lr-wpan-module.h"
6  #include "ns3/internet-apps-module.h"
7  #include "ns3/mobility-module.h"
8
9  using namespace ns3;
10
11 NS_LOG_COMPONENT_DEFINE ("Ping6WsnExample");
12
13 int main (int argc, char **argv)
14 {
15   bool verbose = false;
```

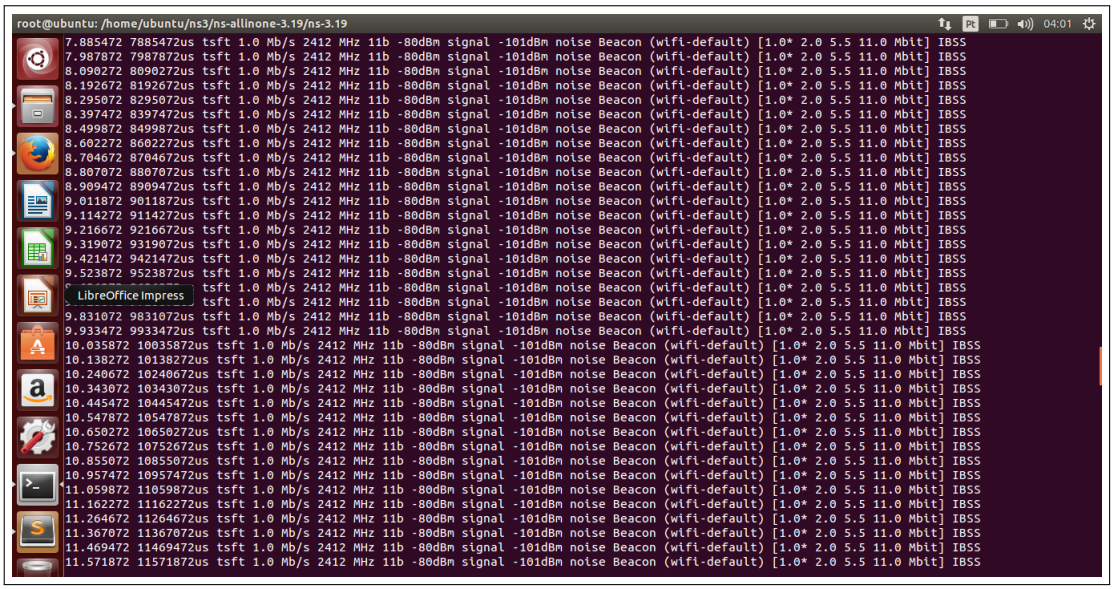
```
16
17      CommandLine cmd;
18      cmd.AddValue ("verbose", "turn on log components", verbose);
19      cmd.Parse (argc, argv);
20
21      if (verbose)
22        {
23          LogComponentEnable ("Ping6WsnExample", LOG_LEVEL_INFO);
24          LogComponentEnable ("Ipv6EndPointDemux", LOG_LEVEL_ALL);
25          LogComponentEnable ("Ipv6L3Protocol", LOG_LEVEL_ALL);
26          LogComponentEnable ("Ipv6StaticRouting", LOG_LEVEL_ALL);
27          LogComponentEnable ("Ipv6ListRouting", LOG_LEVEL_ALL);
28          LogComponentEnable ("Ipv6Interface", LOG_LEVEL_ALL);
29          LogComponentEnable ("Icmpv6L4Protocol", LOG_LEVEL_ALL);
30          LogComponentEnable ("Ping6Application", LOG_LEVEL_ALL);
31          LogComponentEnable ("NdiscCache", LOG_LEVEL_ALL);
32          LogComponentEnable ("SixLowPanNetDevice", LOG_LEVEL_ALL);
33        }
34
35      NS_LOG_INFO ("Create nodes.");
36      NodeContainer nodes;
37      nodes.Create (2);
38
39      // Set seed for random numbers
40      SeedManager::SetSeed (167);
41
42      // Install mobility
43      MobilityHelper mobility;
44      mobility.SetMobilityModel ("ns3::ConstantPositionMobilityModel");
45
46      Ptr<ListPositionAllocator> nodesPositionAlloc = CreateObject<ListPositionAllocator> ()
            ;
47      nodesPositionAlloc->Add (Vector (0.0, 0.0, 0.0));
48      nodesPositionAlloc->Add (Vector (50.0, 0.0, 0.0));
49      mobility.SetPositionAllocator (nodesPositionAlloc);
50      mobility.Install (nodes);
51
52      NS_LOG_INFO ("Create channels.");
53      LrWpanHelper lrWpanHelper;
54      // Add and install the LrWpanNetDevice for each node
55      // lrWpanHelper.EnableLogComponents();
56      NetDeviceContainer devContainer = lrWpanHelper.Install(nodes);
57      lrWpanHelper.AssociateToPan (devContainer, 10);
58
59      std::cout << "Created " << devContainer.GetN() << " devices" << std::endl;
60      std::cout << "There are " << nodes.GetN() << " nodes" << std::endl;
61
62      /* Install IPv4/IPv6 stack */
63      NS_LOG_INFO ("Install Internet stack.");
```

```
64      InternetStackHelper internetv6;
65      internetv6.SetIpv4StackInstall (false);
66      internetv6.Install (nodes);
67
68      // Install 6LowPan layer
69      NS_LOG_INFO ("Install 6LoWPAN.");
70      SixLowPanHelper sixlowpan;
71      NetDeviceContainer six1 = sixlowpan.Install (devContainer);
72
73      NS_LOG_INFO ("Assign addresses.");
74      Ipv6AddressHelper ipv6;
75      ipv6.SetBase (Ipv6Address ("2001:1::"), Ipv6Prefix (64));
76      Ipv6InterfaceContainer i = ipv6.Assign (six1);
77
78      NS_LOG_INFO ("Create Applications.");
79
80      /* Create a Ping6 application to send ICMPv6 echo request from node zero to
81       * all-nodes (ff02::1).
82       */
83      uint32_t packetSize = 10;
84      uint32_t maxPacketCount = 5;
85      Time interPacketInterval = Seconds (1.);
86      Ping6Helper ping6;
87
88      ping6.SetLocal (i.GetAddress (0, 1));
89      ping6.SetRemote (i.GetAddress (1, 1));
90      // ping6.SetRemote (Ipv6Address::GetAllNodesMulticast ());
91
92      ping6.SetAttribute ("MaxPackets", UintegerValue (maxPacketCount));
93      ping6.SetAttribute ("Interval", TimeValue (interPacketInterval));
94      ping6.SetAttribute ("PacketSize", UintegerValue (packetSize));
95      ApplicationContainer apps = ping6.Install (nodes.Get (0));
96      apps.Start (Seconds (2.0));
97      apps.Stop (Seconds (10.0));
98
99      AsciiTraceHelper ascii;
100     lrWpanHelper.EnableAsciiAll (ascii.CreateFileStream ("ping6wsn.tr"));
101     lrWpanHelper.EnablePcapAll (std::string ("ping6wsn"), true);
102
103     NS_LOG_INFO ("Run Simulation.");
104     Simulator::Run ();
105     Simulator::Destroy ();
106     NS_LOG_INFO ("Done.");
107   }
108 }
```

Após a execução da simulação, é criado os arquivos .pcap, donde pode-se se obrtido as métricas necessárias para a fabricação dos resultados. A Figura 21 mostra uma leitura desses

arquivos através da ferramenta tcpdump.

Figura 21 – Leitura do arquivo .pcap através da aplicação tcpdump



Fonte – Elaborada pelo Autor

5.4 Simulação TOSSIM

Como falado anteriormente, o simulador TOSSIM, na verdade, é uma biblioteca TinyOS, ele funciona substituindo os componentes por implementações de simulações.

Sua funcionalidade é bem parecida com a do NS3, onde o usuário terá que realizar desde a criação dos motes até a configuração da aplicação. Por esse fato, também é requerido que o usuário tenha um conhecimento básico em programação, para a configuração dos experimentos.

Dentre as restrições e limites do simulador está no fato de que ele apenas simula a plataforma de *hardware* MicaZ, restringindo o seu uso para outros tipos de sensores.

Conforme o feito no simulador NS-3, também optamos por realizar a experimentação no TOSSIM à partir de um exemplo previamente estabelecido na ferramenta, alterando apenas o necessário para a criação da topologia proposta. Esse script é o mesmo usado sa sessão de experimentos em *IoT*, neste, modificamos apenas o número de motes simulados. O *script* é descrito abaixo.

```
1 from TOSSIM import *
2 import sys
3
4 t = Tossim([])
5 r = t.radio()
6 f = open("topo.txt", "r")
```

```
7
8  for line in f:
9    s = line.split()
10   if s:
11     print " ", s[0], " ", s[1], " ", s[2];
12     r.add(int(s[0]), int(s[1]), float(s[2]))
13
14 t.addChannel("RadioCountToLedsC", sys.stdout)
15 t.addChannel("Boot", sys.stdout)
16
17 noise = open("meyer-heavy.txt", "r")
18 for line in noise:
19   str1 = line.strip()
20   if str1:
21     val = int(str1)
22     for i in range(1, 6):
23       t.getNode(i).addNoiseTraceReading(val)
24
25 for i in range(1, 6):
26   print "Creating noise model for ",i;
27   t.getNode(i).createNoiseModel()
28
29 t.getNode(1).bootAtTime(100001);
30 t.getNode(2).bootAtTime(400008);
31 t.getNode(3).bootAtTime(500008);
32 t.getNode(4).bootAtTime(800008);
33 t.getNode(5).bootAtTime(1800009);
34
35 for i in range(100):
36   t.runNextEvent()
```

5.5 Comparação das Ferramentas de Simulação

De acordo com a experiência de uso descritas nas seções 6.2, 6.3 e 6.4, avaliamos que, em critério de complexidade, durante a criação da topologia, o simulador Cooja mostrou-se bem mais intuitivo, diferentemente do TOSSIM e do NS-3. Pelo fato do Cooja conter uma GUI poderosa, não há a necessidade de configuração manual de protocolos e comunicação, tornando a configuração dos experimentos simples e rápida.

Através de uma pesquisa explorativa em artigos e surveys que realizavam algum tipo de comparação entre simuladores, definimos algumas métricas qualitativas para pudéssemos comparar os simuladores abordados. A A Tabela 3 faz um levantamento expositivo sobre algumas características das ferramentas abordadas, demonstrando quais simuladores se sobrepõe sobre os

outros quanto as métricas empregadas.

Tabela 3 – Comparação técnica entre os simuladores

	Cooja	NS-3	TOSSIM
Suporte GUI	Por Padrão	Instalação a Parte da Padrão/NetAnim ou PyViz	Instalação a Parte da Padrão/TyniViz
Documentação Oficial	Disponibilizada pela Organização	Disponibilizada pela Organização	Disponibilizada pela Organização
Plataforma Operacional	Linux	Linux, FreeBSD, Mac OS X, Windows	Cygwin/Windows, Linux
Resultados	Geradas Automaticamente	Via *Script*	Via *Script*

Fonte – Elaborada pelo Autor

Em se tratando de tipos de dispositivos (*hardwares*) simulados, temos uma grande diferença entre os simuladores. Enquanto o *Cooja* simula diversos tipos de motes, como o *MicaZ, Sky, EXP430F5438, Wismote*, etc., o *TOSSIM* provê suporte apenas ao *MicaZ*. Já no *ns-3*, a diferença se dá pelo fato de que o nó simulado se caracteriza pela classe *NodeContainer*, que contém uma lista de todos os nós criados. Somente após a criação dos nós no *ns-3* é que serão setados os atributos, como meio de comunicação, protocolo utilizado, etc.

Ainda no no paralelo avaliativo, levantamos os principais protocolos relacionados a IoT e criamos uma tabela de comparação em relação aos servidores. A Tabela 4 mostra quais simuladores provêm suporte aos principais protocolos na área de *IoT*.

Tabela 4 – Suporte para protocolos de *IoT*.

	Cooja	NS-3	TOSSIM
MQTT	Sim	Sim	Não
Coap	Sim	Sim	Sim
6LoWPAN	Sim	Sim	Sim
RPL	Sim	Sim	Sim
UDP	Sim	Sim	Sim
IEEE 802.15.4	Sim	Não	Não

Fonte – Elaborada pelo Autor

Com a avaliação dos simuladores, pode-se perceber que estes suportam diversos tipos de simulações *IoT*, fornecendo a coleta das métricas necessárias para a análise dos resultados. O Cooja se diferencia dos outros dois pela vasta usabilidade da interface gráfica, podendo realizar

quase que todas as configurações de uma simulação sem a necessidade de configuração manual dos experimentos, mostrando ser um simulador bem mais intuitivo e de fácil compreensão. Além da funcionalidade de apresentação das métricas de forma automática, em gráficos, durante e depois da simulação. Porém, como toda a configuração da simulação, desde a criação dos motes, a configuração da aplicação, protocolos, meios de transportes e até mesmo o uso de banda, que é feita nos simuladores NS-3 e TOSSIM, os usuários têm um maior poder de controle sobre o experimento criado, além de extrair somente as métricas necessárias, através de *scripts*.

Mesmo não sendo voltado diretamente para *IoT*, o NS-3 suporta as principais simulações nesta área. Além de, juntamente com o TOSSIM, incluir um ambiente completo para permitir usuários de diversos níveis personalizem os tipos de informação para serem extraídas de suas simulações. O NS-3, diferente dos outros dois simuladores, dispõe de de uma documentação oficial bem abrangente, minimizando assim a curva de aprendizagem com a ferramenta. Já o Cooja e o NS-3, dispõem apenas de uma documentação introdutória sobre as ferramentas, deixando o usuário a mercê da comunidade de desenvolvedores, que no caso do TOSSIM, não é tão grande como a do Cooja.

Por fim, a escolha da ferramenta vai de acordo com as necessidades da disciplina, do professor e dos alunos. Caso o intúito seja apenas expor os conceitos de básicos da internet das coisas, sem entrar em uma configuração mais profunda, o Cooja seria a melhor opção, pelo fato simplicidade de configuração e da sua interface intuitiva. Caso o programa busque, além da exposição dos conceitos, uma configuração mais detalhada sobre protocolos e funcionalidades, partiríamos para o uso do TOSSIM, por ser apenas uma biblioteca do TinyOS e compilar para simulação as aplicações usadas no próprio sistema. Por fim, caso a finalidade das aulas sejam uma discussão mais profunda sobre *IoT*, onde será necessária uma experimentação mais precisa e com um poder de configuração mais abrangente, a melhor solução seria o uso do simulador NS-3, que provê uma composição de simulação que pode englobar toda a necessidade dos usuários, além de fornecer uma documentação e suporte bem amplos, o que contribui para a solução das problemáticas encontradas no decorrer do uso.

6 CONSIDERAÇÕES FINAIS

Neste trabalho, discutimos a importância da simulação no âmbito da internet das coisas, além da sua aplicação no ensino de redes de computadores. Apresentamos alguns dos principais simuladores que possuem suporte para *IoT* e realizamos uma breve comparação entre eles, mostrando suas principais características e funcionalidades.

Com isso, disponibilizamos os resultados necessários para que os professores de redes possam selecionar qual simulador mais se adequa ao tipo de ensino necessário para o ensino de *IoT* nos cursos de TI. Dessa forma, pode-se utilizar da simulação como viés para fixação prática dos conceitos abordados teoricamente, o que contribuirá ainda mais para a compreensão dos alunos do curso nas disciplinas de redes de computadores.

Outro resultado obtido, além da experiência inicial com pesquisa acadêmica, foi o conhecimento adquirido nos diversos tipos de simuladores estudados, o que auxiliará em futuras pesquisas na área de IoT.

REFERÊNCIAS

AL-QASEEMI, S. A.; ALMULHIM, H. A.; ALMULHIM, M. F.; CHAUDHRY, S. R. Iot architecture challenges and issues: Lack of standardization. In: IEEE. **Future Technologies Conference (FTC)**. [S.l.], 2016. p. 731–738.

ASHTON, K. That 'internet of things' thing. **RFiD Journal**, v. 22, n. 7, 2011.

CISCO. **An Introduction to the Internet of Things(IoT)**. 2001. Disponível em: <http://www.cisco.com/c/dam/en_us/solutions/trends/iot/introduction_to_IoT_november.pdf>. Acesso em: 11 out 2017.

CISCO. **Cisco IoT Networking - Deploy. Accelerate. Innovate.** 2017.

CONTIKIOS. **Get Started With Contiki**. 2017. Disponível em: <http://www.contiki-os.org/start.html>. Acesso em: 11 out 2017.

DATTA, P.; SHARMA, B. A survey on iot architectures, protocols, security and smart city based applications. In: **2017 8th International Conference on Computing, Communication and Networking Technologies (ICCCNT)**. [S.l.: s.n.], 2017. p. 1–5.

DEOGIRIKAR, J.; VIDHATE, A. Security attacks in iot: A survey. In: IEEE. **I-SMAC (IoT in Social, Mobile, Analytics and Cloud)(I-SMAC), 2017 International Conference on**. [S.l.], 2017. p. 32–37.

FILIPPETTI, M. A. **Redes de Computadores i:** arquitetura para laboratórios híbridos com acesso remoto. 2010.

FROST, V. S.; MELAMED, B. Traffic modeling for telecommunications networks. **IEEE Communications Magazine**, IEEE, v. 32, n. 3, p. 70–81, 1994.

ITU. **ITU Internet Reports 2005**. 2005. Disponível em: <https://www.itu.int/net/wsis/tunis/newsroom/stats/The-Internet-of-Things-2005.pdf>. Acesso em: 10 out 2017.

LEVIS, P.; LEE, N. Tossim: A simulator for tinyos networks. **UC Berkeley, September**, v. 24, 2003.

MARTINS, G. N. Avaliação da confiabilidade de simulador em redes de sensores sem fio com base em plataforma real de sensoriamento. 2016.

MENOSSI, J. et al. Análise de desempenho comparativo entre emulação, simulação e métodos híbridos para sdn. Universidade Federal de São Carlos, 2017.

MINAKOV, I.; PASSERONE, R.; RIZZARDI, A.; SICARI, S. A comparative study of recent wireless sensor network simulators. **ACM Transactions on Sensor Networks (TOSN)**, ACM, v. 12, n. 3, p. 20, 2016.

MUHAMMAD, H. H.; BEDIN, G. B.; FACCHINI, G.; BARCELLOS, M. P. Quebrando a barreira entre simulação e experimentação prática em redes de computadores. **XXII Simpósio Brasileiro de Redes de Computadores, SBRC 2004**, v. 1, p. 87–100, 2004.

NETANIM. **NetAnim**. 2017. Disponível em: <https://www.nsnam.org/wiki/NetAnim>. Acesso em: 18 out 2017.

NS3. **Getting Started**. 2017. Disponível em: <https://www.nsnam.org/docs/tutorial/html/getting-started.html>. Acesso em: 15 out 2017.

NS3. **ns-3 Networking Simulator**. 2017. Disponível em: <https://www.nsnam.org/docs/release/3.26/tutorial/ns-3-tutorial.pdf>. Acesso em: 16 out 2017.

SANTOS, W. Uso de simuladores como ferramenta no ensino-aprendizagem de redes de computadores. **Projetos e Dissertações em Sistemas de Informação e Gestão do Conhecimento**, v. 4, n. 2, 2016.

SARI, R. F.; WIRYA, P. L. P. Performance analysis of session initiation protocol on emulation network using nist net. In: IEEE. **Advanced Communication Technology, The 9th International Conference on**. [S.l.], 2007. v. 1, p. 506–510.

SCOTT, G. **Notas de estudo Tinyos**. 2014. Disponível em: <http://www.cnblogs.com/gaohongchen01/p/3728981.html>. Acesso em: 18 out 2017.

SILVA, I.; MARQUES, C.; LIMA, R. Integrando o emulador gns3 como suporte de ensino na disciplina de redes de computadores no ambiente ava. In: **Brazilian Symposium on Computers in Education (Simpósio Brasileiro de Informática na Educação-SBIE)**. [S.l.: s.n.], 2017. v. 28, n. 1, p. 1727.

SIRAJ, S.; GUPTA, A.; BADGUJAR, R. Network simulation tools survey. **International Journal of Advanced Research in Computer and Communication Engineering**, v. 1, n. 4, p. 199–206, 2012.

SOUSA, M.; SILVA, J.; SOUZA, W.; CABRAL, Y.; ROCHA, U.; OLIMPIO, T. Simuladores e emuladores de rede para o projeto e solução de problemas em ambientes de produção. **Revista de Tecnologia da Informação e Comunicação**, v. 6, n. 2, p. 16–25, 2016.

TOSSIM. **TOSSIM Compilation Steps**. 2013. Disponível em: <http://tinyos.stanford.edu/tinyos-wiki/index.php/TOSSIM#Append_B:_TOSSIM_Compilation_Steps>. Acesso em: 17 out 2017.

VOSS, G. B.; MEDINA, R. D.; AMARAL, É. M.; ARAÚJO, F. V. do; NUNES, F. B.; OLIVEIRA, T. B. de. Proposta de utilização de laboratórios virtuais para o ensino de redes de computadores: Articulando ferramentas, conteúdos e possibilidades.(fase i). **RENOTE**, v. 10, n. 2, 2012.

WIRTH, P. E. The role of teletraffic modeling in the new communications paradigms. **IEEE Communications Magazine**, IEEE, v. 35, n. 8, p. 86–92, 1997.