GRCM – Grupo de Redes de Computadores e Multimídia
Programa de Pós-Graduação em Informática
Universidade Federal do Amazonas

ICOMP
Instituto de Computação

Trabalho Prático 2 de Comunicação Sem Fio

Manaus - Amazonas
November – 2018 – Novembro

Testes de Desempenho usando DTN2 com *Raspberry*

Diogo Soares[1], Helmer Mourão[1], Waldomiro Seabra[1]
{x, x, x}@gmail.com

[1] Universidade Federal do Amazonas (UFAM)
Manaus – AM – Brasil

Resumo: Este relatório técnico apresenta um experimento de medição de desempenho de comunicação entre *raspberries* usando o protocolo de comunicação DTN2, uma biblioteca que emula o comportamento do protocolo DTN dentro de um dispositivo.

Palavras-Chave: Raspberry Pi, Avaliação de desempenho, DTN, DTN2.

---

1. Introdução

O trabalho desenvolvido neste relatório técnico foi desenvolvido no segundo semestre de 2018 como parte da disciplina de Comunicação sem Fio (CsF) sob supervisão do professor Dr. Eng. Edjair de Souza Mota.

Dados dos Equipamentos utilizados neste laboratório:

- Placa Raspberry Pi – modelo B+ V1.2
- Placa Raspberry Pi 2 – modelo B
- 2x Memória SSD para Raspberry – 8GB
- Bateria raspberry –
- 2x Fonte raspberry –
- PC - (para acesso remoto) – Ubuntu 18.04

Dados dos Softwares:

- Sistema Operacional Raspberry – Raspbian Lite
- Sistema Operacional Ubuntu (PC) – 16.04 LTS
- *DTN2* – 2.9
- *Oasys* – 1.6.0
- *TCL* – 8.5.18
- *DB Berkeley* – 5.3.28 disponível em github.com/MatheusOliveira2015/db

2. Relato de Atividades

2.1. Atividades do TP2

Pode-se dividir a montagem e operação do laboratório nas seguintes atividades macros:

1. Configuração dos módulos necessários para instalação do DTN2.

2. Configuração da rede entre os *raspberries*.
3. Realização das medições.
4. Análise dos dados.
5. Finalização do relatório técnico.

Os códigos e scripts utilizados neste trabalho foram disponibilizados no endereço https://github.com/diogosm/tp2CSF. Além disso, os logs gerados durante o exercício também estão incluídos no repositório *git*.

2.2. Instalação de Módulos

Primeiramente fizemos a instalação dos itens básicos em nossa imagem *raspbian* construída no TP1 de CsF. Assim instalamos os seguintes componentes, já incluídos na descrição do trabalho. Contudo, no primeiro momento, excluímos os *downgrades* realizados no compilador do C/C++, desse modo utilizando a versão já instalada nos raspberries, a gcc/g++ 6.3. Além disso, a versão de *kernel* utilizada foi a 4.14.50+.

```
$ sudo apt-get update && sudo apt-get -f install expat xsdcxx hgsvn ecl bison cvs flex gperf texinfo automake libtool help2man gawk autoconf libexpat1-dev libexpat1 libxerces-c2-dev libssl-dev libncurses5-dev
```

Fonte: Os Autores

A instalação do componente TCL e DB *Berkeley* foram feitas de acordo com a descrição do trabalho (exceto pela versão 6.3 do gcc/g++). Além disso, foi preciso alterar a versão do TCL para 8.5.18 pois a versão 8.5.19 estava indisponível. Elas finalizaram sem erros.

Durante a instalação do componente oasys, acabamos tendo problemas com a versão 6.3 do compilador gcc. Assim, foi necessário fazer uma instalação do gcc/g++ 4.8 (de acordo com a descrição do trabalho) e usar o *exports* iguais ao descrito na seção 1 do enunciado do trabalho. Após isso, o oasys finalizou sem erros.

Finalmente, durante a instalação do DTN2 tivemos alguns problemas na compilação. Na primeira tentativa escolhemos utilizar o gcc/g++ 4.8 pelos mesmos motivos do oasys (falta de suporte da versão 6 do gcc/g++). Contudo, obtivemos o erro apresentado abaixo, que segundo alguns fóruns estava relacionado a mudanças no *build* do gcc 4.8.

```
$ g++-4.8: error: unrecognized command line option '-fstack-protector-strong'
```

Fonte: DTN2

Na segunda tentativa utilizamos o gcc/g++ 4.9, novamente baseado em algumas respostas dos fóruns de C++. Assim, passamos a ter problemas no *make* devido a libssl instalada ser a 1.1. Não

compreendemos bem exatamente o problema, mas o erro apresentado tinha relação a um import do componente ans1, incluso na biblioteca openssl. O erro que obtivemos é apresentado na figura a seguir. Assim, fizemos uma tentativa de downgrade utilizando apt-get para libssl1.0-dev. Após isso, a compilação finalizou com sucesso.

```
$ security/cms_structs.h:538:24: error: expected constructor, destructor, or type conversion before 'OUR_ContentInfo'
```

Fonte: DTN2

2.3. Configuração da rede *Ad-Hoc*

Para realização de configuração da rede *ad-hoc*, foi necessário apenas a modificação do arquivo /etc/network/interfaces de cada raspberry para manter a configuração estática e sempre inicializada com os *raspberries*. O resultado é demonstrado abaixo.

```
## add essas linhas para modo ad-hoc
auto wlan0
iface wlan0 inet static
        address 192.168.12.3
        netmask 255.255.255.0
        wireless-channel 9
        wireless-essid icomp-ctic
        wireless-mode ad-hoc
```

Fonte: *interfaces* de um dos *raspberries*

2.4. Configuração do DTN2

Para configuração do realizamos os seguintes procedimentos:

```
$ sudo mkdir -p ~/dtn && sudo mkdir -p ~/dtn/bundles && sudo mkdir -p ~/dtn/db && sudo cp /etc/dtn.conf ~/dtn
```

Fonte: Os autores

Após isso, realizamos as configurações necessárias no arquivo dtn.conf. Essas instruções foram realizadas de acordo com as instruções descritas no enunciado e compiladas abaixo como linhas a serem adicionadas nos *raspberries* 1 e 2.

```
$ sudo mkdir -p ~/dtn && sudo mkdir -p ~/dtn/bundles && sudo mkdir -p ~/dtn/db && sudo cp /etc/dtn.conf ~/dtn
```

Fonte: Os autores

```
## add essas linhas no ~/dtn/dtn.conf dos raspberries
storage set payloaddir /home/pi/dtn/bundles
storage set dbdir          /home/pi/dtn/db
link add link_tcp 192.168.12.4 ALWAYSON tcp
## alteramos o /etc/hostname e /etc/hosts
route local_eid "dtn://[info hostname].dtn"
route add dtn://[info hostname]/* link_tcp
discovery add lawliet ip local_addr=192.168.12.3 port=9556 unicast=false
continue_on_error=true
discovery announce link_tcp lawliet tcp cl_addr=192.168.12.3 cl_port=9557 interval=1
```

Fonte: dtn.conf de um dos *raspberries*

Após isso fizemos os ajustes do servidor ntp. Para isso ajustamos primeiramente a zona para Manaus como mostrado abaixo.

```
$ sudo timedatectl set-timezone America/Manaus
```

Fonte: Os autores

Após isso, atualizamos o servidor ntp, conforme demonstrado abaixo.

```
$ sudo ntpdate pool.ntp.org
```

Fonte: Os autores

O resultado pode ser confirmado depois usando o comando *date*. Após isso, começamos a realizar os testes de comunicação DTN2 usando modo *ad-hoc* de rede.

No primeiro momento notamos que a execução de *dtnping* entre os nós não funcionava. Após a análise da documentação oficial [1], notamos algumas diferenças da documentação descrita no enunciado. Após algumas modificações, nossa configuração de roteamento (configurações de *storage* permaneceram iguais a descrição dita anteriormente) do dtn.conf de cada raspberry ficou como mostrado abaixo. Para todos efeitos assumimos que a comunicação ocorre entre o nó chamado Lawliet e um nó chamado Kira, como demonstrado na Figura 1.

Figura 1: Cenário de comunicação entre os nós cliente/servidor.

```
## add essas pro Lawliet
discovery add lawliet ip local_addr=192.168.12.3 port=9557 unicast=false
continue_on_error=true
discovery announce link2 lawliet tcp cl_addr=192.168.12.3 cl_port=9556 interval=1
```

Fonte: dtn.conf do nó Lawliet

```
## add essas pro Kira
link add link2 192.168.12.3 ALWAYSON tcp
route add dtn://kira.dtn/* link2
discovery add kira ip local_addr=192.168.12.4 port=9556 unicast=false
continue_on_error=true
discovery announce link2 kira tcp cl_addr=192.168.12.4 cl_port=9557 interval=1
```

Fonte: dtn.conf do nó Kira

Após essas modificações nossa comunicação foi estabilizada entre os dois nós. Além disso, nós alteramos o comando de execução do *dtn daemon* para nos dá mais informações e *logs*. Para executar em cada nó, fizemos:

```
$ sudo dtnd -c ~/dtn/dtn.conf -t -l info
## -t: reseta o banco
## -l info: log com mais informações
```

Fonte: Os autores

2.5. Experimento

A realização do experimento consistia na execução do *dtnperf*, ferramenta de medição de desempenho do DTN2 e da verificação da potência de sinal entre os nós. Para isso, realizamos medições em 5, 10 e 15 metros de distância entre os nós com ambos a uma altura de 0,7m do solo. Os experimentos foram realizados no corredor do bloco do Centro de Processamento de Dados. Além disso foi realizado taxas de envio de 50K, 100K, 250K, 500K, 1M, 2,5M e 10M e cada experimento foi executado 5 vezes. A Figura 2 demonstra o cenário onde o experimento foi realizado.

Figura 2: Cenário de experimentação.

2.6. Resultados

A primeira parte de nossos resultados consistiu na análise da qualidade de sinal entre os *raspberries*. Para isso utilizamos o *iwconfig* para extrair o valor de qualidade de sinal (RSSI). Os resultados podem ser vistos na Figura 3, lembrando que quanto mais próximo de 30 (-30 dBm), melhor o sinal. Percebemos pelo resultado que havia uma boa qualidade de sinal (quando a força do sinal está acima de -70 dBm, a comunicação é considerada estável) entre os raspberries durante o horário de execução do experimento, embora a interferência co-canal seja um problema conhecido naquela localidade como demonstramos no TP1 da disciplina.

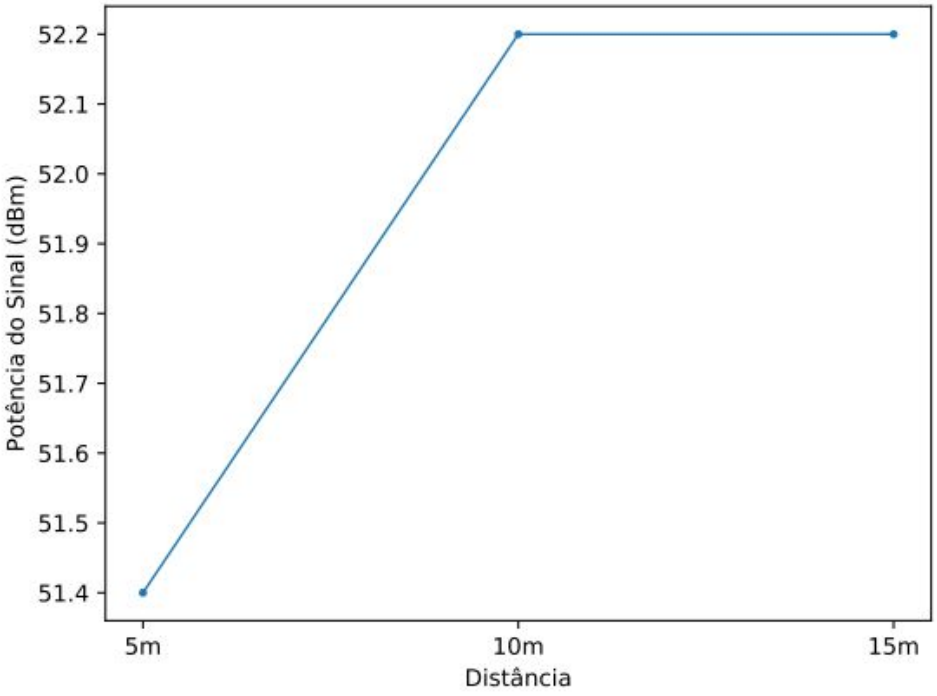

Figura 3: Resultado da potência do sinal adquirido (em dBm) pela distância entre os *raspberries*.

O segundo resultado que extraímos foi a vazão média (*goodput*) extraídos da ferramenta *dtnperf*. Esse resultado representa a quantidade de dados úteis e não perdidos transmitidos no canal durante uma determinada quantidade de tempo. Os resultados podem ser vistos na Figura 4.

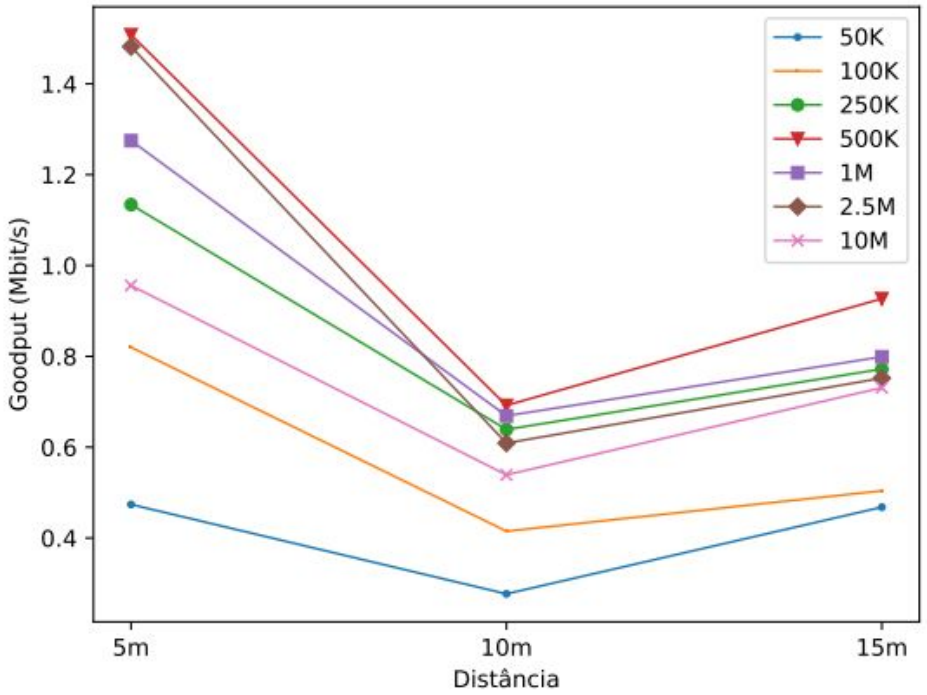

Figura 4: Resultado do *goodput* (em Mbits/segundo) pela distância.

Por fim, analisamos o *goodput* obtido para cada taxa de envio utilizada no *dtnperf*. Para questões de esclarecimento, foi utilizado o modo de dados no *dtnperf* [5]. No modo de dados, o *dtnperf* fica responsável por enviar N *bytes* para o servidor que retorna o tempo utilizado para esta transmissão. O resultado é apresentado na Figura 5.

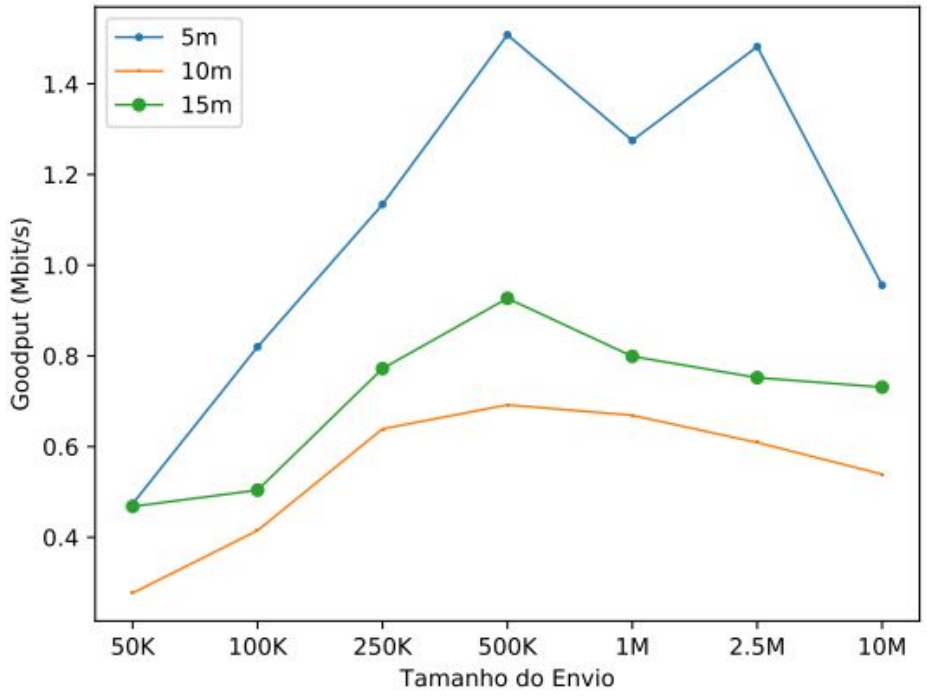

Figura 5: Resultado do *goodput* (em Mbits/segundo) pela taxa de envio.

Pelo resultado apresentado percebemos que quando o aumentamos a distância, a perda de desempenho do *goodput* foi de até 57% quando usamos como comparação o maior *goodput* encontrado na distância de 5 metros (aproximadamente 1,4 Mbits/seg). Embora a distância parece ser determinante para o *goodput*, percebemos que quando a distância foi aumentada de 10 metros para 15 metros, chegamos a ter ganhos próximos a 50% em algumas das taxas de envio nessas duas distâncias. Talvez um teste comparando outras condições de cenário como menor interferência co-canal ou variando o tamanho de *payload* poderia nos mostrar outras considerações relevantes para esta análise de desempenho.

3. Referências

[1] Documentação oficial do DTN2. Acessado em novembro de 2018. Disponível em: http://dtn.sourceforge.net/DTN2/doc/manual/.

[2] Analisis Transfer Data Pada Bundle Protocol Wireless Router Bergerak Di Jaringan Delay Tolerant Network (DTN). *Akhir, Tugas*, acessado em novembro de

2018. Disponível em: : http://www.digilib.unsri.ac.id/index.php?p=fstream-pdf&fid=8590&bid=8559.

[3] g++ 4.8 error: -fstack-protector-strong. Acessado em novembro de 2018. Disponível em: https://github.com/pytorch/pytorch/issues/1098.

[4] Aplicações para Redes Veiculares Tolerantes a Atrasos. Nunes, José J. S., acessado em novembro de 2018. Disponível em: http://mariel.inesc-id.pt/prbp/diversos/DissertacaoJorgeNunes.pdf.

[5] DTNPERF help file. Lannacone, L., acessado em novembro de 2018. Disponível em: https://github.com/Leolannacone/dtn/blob/master/apps/dtnperf/dtnperf-instructions.txt.