# IEC684 - Simulação de Sistemas

**Professor:** Edjair Mota

Relatório de Simulação – Lab. 02

**Alunos:**

| | |
|---|---|
| Bruno Machado | 20901630 |
| Cristiano Medeiros | 20810162 |
| Daniele Seixas | 20810175 |
| Manoel Mota | 20810481 |


Manaus, 2012

## 1. INTRODUÇÃO

Os protocolos TCP/IP são amplamente conhecidos e utilizados por serem robustos e eficientes em várias aplicações, inclusive a rede mundial de computadores, a Internet. Porém, em redes onde são freqüentes os atrasos e desconexões, tais protocolos tornam-se ineficazes para os propósitos de transmissão dos dados a princípio não críticos e tolerantes a atrasos. Para tal situação pode-se utilizar uma arquitetura de redes mais apropriada chamada de DTN (Delay-tolerant networking), um acrônimo para Redes Tolerantes a Atrasos em inglês.

Utilizaremos nesse trabalho técnicas de simulação de sistemas para modelar um problema real, descrito abaixo, a fim de possibilitar a verificação da viabilidade da aplicação, possíveis problemas, pontos de melhorias dentre outros resultados, dispensando a necessidade de uma implementação real para realização de testes.

O problema real abordado será a troca de mensagens (entenda mensagem como um arquivo digital contendo som, áudio, vídeo, imagem, etc.) entre dispositivos móveis (*hosts*) que se conectam formando uma rede DTN.

Computadores, celulares, tablets e outros dispositivos eletrônicos tem recursos para armazenamento limitados. Em virtude dessa limitação não podemos implementar uma aplicação que envie um número ilimitado de mensagens esperando que o *host* de destino consiga armazenar tudo. Para solucionar esse problema a aplicação deve implementar políticas de descarte a fim de gerenciar o espaço de armazenamento e as mensagens mais importantes. Abordaremos duas políticas de descarte: Drop Last recently received e Drop Social.

## 2. IMPLEMENTAÇÃO

### 2.1. MODELOS DE MOBILIDADE

Para possibilitar a simulação do problema precisaremos utilizar um modelo de mobilidade. Foram verificados os modelos Random way point, Random Walk e Modelo de mobilidade da Universidade Federal do Amazonas (UFAM). Adotamos o último por se tratar de um modelo que reflete melhor o modelo de mobilidade dos frequentadores da universidade.

#### Random Way point Model

O modelo Random Way consiste em pegar cada nó da área simulada e inseri-los numa determinada posição e fazer-los pausar por um determinado tempo e após a esse tempo de pausa é escolhido aleatoriamente um novo destino (sempre respeitando os limites da área simulada) com uma velocidade ß uniformemente distribuída no intervalo[ßmin, ßmax]. Após chegar ao novo destino uma nova pausa ocorre e se repete o processo descrito anteriormente.

#### Random Walk Model

O Modelo random walk consite em termos um determinado nó numa Mn posição, aí será escolhido uma direção D de forma aleatória com uma velocidade de deslocamento ßn uniformemente distribuída no intervalo [0,2π] e [ßmin,ßmax] respectivamente. Dessa forma ocorrerá um deslocamento em linha reta do nó(µ) até a nova posição Mn+1 com a velocidade definida. Este nó irá respeitar as limitações da área que será simulada, de forma a altera a sua trajetória fazendo uma reflexão na borda da área com o ângulo de saída similar ao ângulo de incidência na borda. Assim serão determinadas novas direções e velocidades para cada nó, de acordo com o intervalo ø ou a distância percorrida ¶.

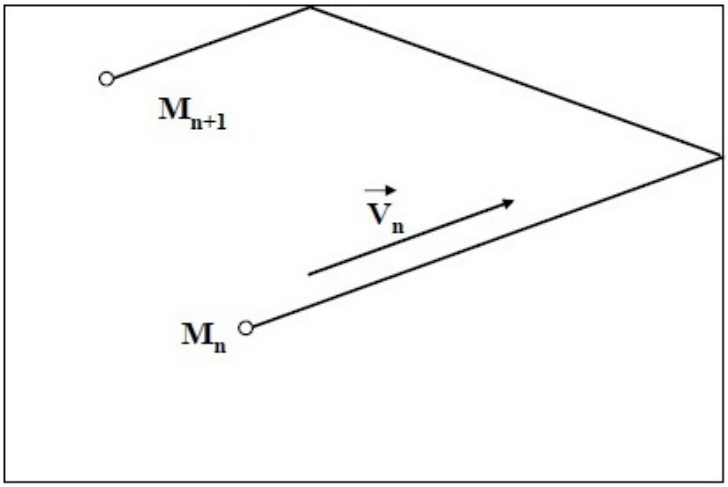



**Figura 1 - Random Walk com reflexão**

Nosso grupo não ultilizou os modelos descritos acima pelo fato do deslocamento realizado pelas coletas ter apresentado alguns obstáculos como prédios, mata fechada entre outros, o que não ocorreria com estes modelos aleatórios, já que usamos dados coletados por um aplicativo usando GPS. O que nós proporciona a criar um modelo de mobilidade da área usada neste caso uma área previamente delimitada na UFAM para este caso.

## 2.2. POLÍTICAS DE DESCARTE

Como os *hosts* são equipamentos compactos e de recursos limitados, é muito provável que em um certo momento o espaço dedicado ao armazenamento das mensagens se esgote de forma que a próxima mensagem não caiba mais no *buffer,* sendo necessário então adotar uma das políticas de descarte.

### 2.2.1. Política Drop Last Recently Received

De acordo com essa política devem ser descartadas as mensagens que estiverem a mais tempo armazenadas no *host* em caso de *overflow* no *buffer*.

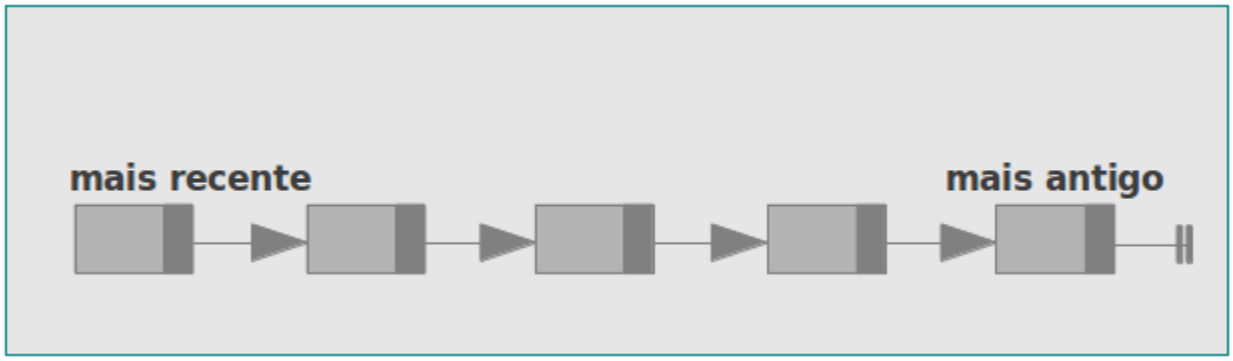



**Figura 01: Remover a mensagem que está a mais tempo no buffer.**

Para utilização dessa política no One criamos o método **getMaisAntiga()** na classe **ActiveRouter.java.** Esse método deve retornar a mensagem que está a mais tempo no buffer. Segue abaixo a implementação:

```
protected  Message getMaisAntiga(boolean maisvelha)
{
        Collection<Message> hist = this.getMessageCollection(); (1)
              Message antiga = null; (2)
                 for (Message mensagem : hist) (3)
                {
                          if (maisvelha && isSending(mensagem.getId()))
                          {
                                  continue; (4)
                          }
                          if (antiga == null )
                          {
                                  antiga = mensagem; (5)
                          }
                         else if (antiga.getReceiveTime() > mensagem.getReceiveTime())
                         {
                                  antiga = mensagem; (6)
                          }
                }
                return antiga; (7)
}
```

Comentários:

(1) A coleção *hist* contém as mensagens armazenadas no *buffer.*

(2) O objeto “antiga” da classe Message é inicializado com null pois ainda não se sabe qual é a mensagem mais antiga.

(3) O *buffer* começa a ser varrido.

(4) Se a mensagem, que está prestes a ser excluída por ser a mais antiga, estiver sendo enviada, a exclusão é abortada e o *buffer* continua sendo varrido em busca da próxima mensagem.

(5) A primeira mensagem encontrada no *buffer* será cosiderada a mais antiga para comparações posteriores.

(6) Sempre que for encontrada uma mensagem mais antiga que a atual, essa deve assumir o posto de mais antiga.

(7) O método retorna a mensagem mais antiga.

Por se tratar de uma política de descarte comum e de fácil implementação a equipe não encontrou dificuldades e nem precisou discutir muito sobre a mesma.

### 2.2.2. Política Drop Social

De acordo com essa política deve-se implementar uma maneira de mensurar o grau de afinidade entre o *localhost* e os dispositivos presentes em sua lista de *hosts* conhecidos. Dessa maneira, em caso de *overflow* no *buffer* podemos eliminar as mensagens dos *hosts* que possuem os menores graus de afinidade.

Caso ocorra uma situação em que os *hosts* possuem o mesmo grau de afinidade o outro parâmetro que deve ser levado em consideração é o tamanho das mensagens. Em suma, as mensagens com maior tamanho devem ser eliminadas primeiro.

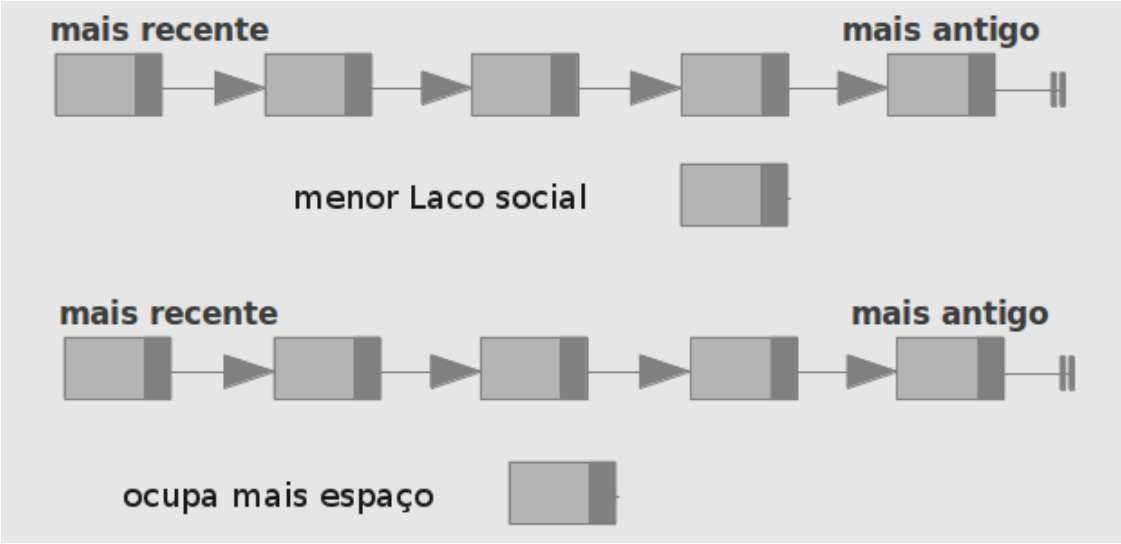



**Figura 02: As mensagens com menor grau de afinidade e maior tamanho devem ser removidas.**

A política Drop Social se mostrou mais interessante que a anterior porque despertou a

discussão na equipe acerca de decisões que deveríamos tomar para definir o modelo da nossa simulação.

A primeira conversa foi para determinarmos como conseguiríamos mensurar o grau de afinidade de um *host* com seus vizinhos.

Decidimos que na classe DTNHost.java, de onde são instanciados os objetos que representam os dispositivos móveis do mundo real, precisávamos incluir um atributo “grauAfinidade” para armazenar essa informação. Isso gerou outras questões: de que tipo será o atributo? Quais valores ele deve assumir? Quando ele deve ser atribuído ou alterado?

A equipe decidiu que tal atributo deve ser do tipo inteiro para facilitar os cálculos e ganhar um pouco mais no desempenho já que estamos trabalhando com um sistema embarcado. Os possíveis valores devem variar de 0 a 100. Quando for zero significa que o *host* vizinho não possui afinidade alguma com o *host* que está descartando a mensagem e 100 quando tiver um nível excelente de afinidade com o mesmo.

Sempre que um novo vizinho é descoberto inicializa-se o atributo com zero. A cada nova mensagem recebida ou enviada do/para o *host,* o mesmo tem seu atributo grauAfinidade incrementado de 1.

Foi uma decisão unanime da equipe que, a cada novo dia o atributo grauAfinidade dos *hosts* conhecidos deveriam ser  zerados. Dessa forma evitaremos tal situação:

O *host* A tem um grau de afinidade de 98 com o *host* B em um determinado momento da segunda-feira, por exemplo. Uma explicação para esse caso é que o *host* A transmitiu e recebeu muitas mensagem para/do *host* B nesse dia. No entanto, no dia seguinte, pode ser que os usuários desses dois dispositivos não se cruzem na universidade e por tanto deveriam ter cada um o grau de afinidade bem menor que 98. Com isso mensagens podem ser descartadas equivocadamente.

Definimos o intervalo de 0 a 100 para o atributo grauAfinidade porque julgamos que em um dia se dois dispositivos fazem 100 trocas de mensagens eles se conhecem bastante, tem bastante contato um com o outro. Além de ficar prontinho pra mostrarmos um status de afinidade em porcentagem para o usuário, por exemplo, definimos que um dispositivo que tem grau de afinidade 95 com outro *host* possui na nossa aplicação 95% de afinidade com o mesmo.

Durante a implementação da política DropSocial, foram modificados apenas os arquivos ActiveRouter.java, localizado no diretório /routinge e o arquivo Message.java localizado no

diretório /core. Ambos os diretório estão dentro do ONE. Para a implementação da política de descarte foram feitas as seguintes alterações:

**Message.java**

1 - criamos uma variável de nome laço social:

```
private int lacoSocial; // indica o laco social do host na rede DTN
```

2 - no metodo construtor Message inicializamos a variável setando zero;

```
this.lacoSocial = 0; // indica que o host ainda não possui laco social
```

3 - definimos os métodos, **getLacoSocial** e **setLacoSocial**, o primeiro devolve o valor de laço social de um host e o segundo método abribui um valor de laço social;

```
public int getLacoSocial() {
        return this.lacoSocial;
}

public void setLacoSocial(int Social) {
        this.lacoSocial = Social;
}
```

**ActiveRouter.java**

Para a implementação dessa política adicionamos desenvolvemos o método, getDropSocial. Definimos uma variável para pegar o menor laço social no buffer, em seguida uma outra variável que irá guardar a mensagem que ocupa maior espaço no buffer dentre as que tiverem o mesmo laço social. segue a implementação:

```
protected  Message getDropSocial(boolean eleita){
        Collection<Message> hist = this.getMessageCollection(); //mensagens no buffer
        Message menorLacoSocial = null;
        Message mensagemMaior = null;
 for (Message mensagem : hist) { // percorre o buffer
        if (isSending(mensagem.getId())) { //pega o id da mensagem
                continue;
         }
        if (menorLacoSocial == null) {
                menorLacoSocial = mensagem;
        }else if (menorLacoSocial.getLacoSocial() > mensagem.getLacoSocial()) {  //encontra a
mensagem com menor laco social
        menorLacoSocial = mensagem;
       }
 }
 for (Message mensagem : hist) { // percorre o buffer
```

```
          if (isSending(mensagem.getId())) {
             continue;
          }
if(mensagem.getLacoSocial() == menorLacoSocial.getLacoSocial()){ //verifica se a mensagem tem o mesmo laco social encontrado anteriormente

          if(mensagemMaior == null){
                    mensagemMaior = mensagem;
          }else{
                    if(mensagemMaior.getSize() < mensagem.getSize()){ // encontra a mensagem que ocupa maior espaco no buffer
                    mensagemMaior = mensagem;
                    }
          }
 }
}

 if (mensagemMaior != null){
     return mensagemMaior; // retorna a mensagem que ocupa maior tamanho no buffer
  }else  return menorLacoSocial; // retorna mensagem com menor laco social

}
```

Com essa implementação obdecemos as regras estabelecidas para a política: se todos as mensagens do buffer tiverem o mesmo laço social é necessário descartar a mensagem que ocupar mais espaço no buffer.

### 2.3. MODELO DE MOBILIDADE DA UFAM

Para gerar o modelo de mobilidade da UFAM realizamos coletas dos registros de movimento (*traces*) de alguns alunos da universidade durante os meses de abril e maio de 2012.

Para a captura e geração dos logs dos *traces* utilizamos o aplicativo **MyTracks** instalado em celulares Android. Temos duas configurações distintas a saber:

Configuração 1: Celular Samsung Galaxy 5 modelo GT-I5500. Sistema operacional Android v2.3.7. Aplicativo My Tracks versão 1.1.11.

Configuração 2: Celular Samsung Galaxy 5 modelo GT-I5500. Sistema operacional Android v2.3.7. Aplicativo My Tracks versão 1.1.16.

Alguns celulares, com a configuração 1, foram ceditos pelo IComp por meio do professor Edjair Mota para utilização neste trabalho de simulação.

Também fizemos coletas com um celular de configuração 2 pertencente a aluna Daniele Seixas, uma das integrantes dessa equipe.

Utilizamos um notebook Acer modelo Extensa 5230E, 3GB de memória e processador Celeron de 2.8 GHz de clock 32 bits com sistema operacional Linux Ubuntu 12.04 para realizar todos os processamentos.

Os logs dos *traces* foram exportados dos dispositivos no formato .gpx para serem processados utilizando o aplicativo ***bonnmotion*** versão 2.0 a fim de gerar os arquivos contendo os parâmetros e movimentos dos *traces.* Os passos são descritos abaixo:

1. Primeiramente deve ser feito o download do programa bonnmotion. Este pode ser encontrado no site http://net.cs.uni-bonn.de/wg/cs/applications/bonnmotion/.
2. O arquivo bonnmotion-2.0.zip deve ser descompactado em um diretório de fácil acesso. Utilizamos o diretório **/home/usuario/bonnmotion-2.0** para descompactar o arquivo, com o comando **usuario$: sudo unzip bonnmotion-2.0.zip /home/usuario**.
3. Instalamos o aplicativo com o comando **usuario$: ./install** no diretório */home/usuario/ bonnmotion-2.0/*:
4. Os arquivos .gpx precisam conter uma tag chamada bounds onde devem ser informadas as coordenadas de máximas e mínimas longitudes e latitudes.

Como nossa simulação deve abranger toda a área construída da UFAM capturamos dois pontos no mapa utilizando o programa Google Earth para servir de ponto máximo e ponto mínimo (ver imagem 03).

**<bounds minlat="-3.102631" minlon="-59.974447" maxlat="-3.086878" maxlon="-59.964511"/>**


**Figura 03: Definição dos pontos de máximo e mímino utilizando Google Earth**

5. Os arquivos .gpx gerados no dispositivo foram importados para o diretório /home/usuario/ bonnmotion-2.0/bin.

6. Executamos o comando GPXImport para cada arquivo .gpx importado. Dessa maneira são gerados os arquivos .gpx.params e .gpx.movements utilizados posteriormente na geração do arquivo .one que deve ser utilizado no simulador.

Exemplo:

No terminal digitamos **cd /home/usuario/bonnmotion-2.0/bin**

**./bm GPXImport -f c1.gpx -c true**

7. Após esses comandos podemos utilizar um outro comando o **TheONEFile** do bonnmotion para converter o arquivo de mobilidade para o formato one para todos os arquivos .gpx importados.

Exemplo:

No terminal digitamos **./bm TheONEFile -f c1.gpx**

8. Os arquivos gerados após esse comando são algo como .gpx.one. A partir desses arquivos, devemos juntar os dados de cada arquivo e colocar em um só, ordenando-os na ordem crescente de acordo com o tempo. Para esse ordenação, usamos o programa Microsoft Excel.
9. Nesse momento temos um arquivo chamado ordenados.gpx.one contendo as coordenadas de todos os pontos capturados nos *traces*.

Esse arquivo é utilizado para executar a simulação no One. Basta alterar algumas linhas no arquivo default_settings.txt do simulador One para definir nosso modelo como default, veja:

Group.movementModel = **ExternalMovement**
ExternalMovement.file = **/home/manoel/Download/manoel_gpx/ordenados.gpx.one**
Group.nrofHosts = **11**

10. Para executar a simulação no One é preciso estar no diretório do simulador e digitar **./ One .sh**.

### 3.2. Verificação e Validação

A verificação e validação deste trabalho ocorreu com o tomador de decisões no caso o monitor da disciplina, Camilo, onde conforme estamos avançando nas etapas do trabalho, estavamos procurando o Camilo, para o feedback se estamos realizando corretamente a etapa corrente, até chegamos ao modelo de mobilidade da UFAM essas verificações foram feitas por meio de emails e bate-papos. Além de recorrer ao Camilo também buscamos informações com integrantes do laboratório do GRCM como o Diogo e a Poliana, que nos orientaram em relação a dúvidas que surgiram a cada avanço das etapas.

Na primeira etapa validamos nosso dados a partir do nosso problema que era a mobilidade de pessoas numa determinada área da UFAM. Enviamos as políticas que nos foram designada, já implementadas para o monitor Camilo, porem até o fechamento deste relatório não ocorreu o feedback.

Na fase seguinte geramos o modelo de mobilidade da UFAM, para chegamos ao resultado do mesmo recorremos algumas vezes ao Camilo, com o propósito de validar as decisões tomadas durante o trabalho.

## 4. Resultados e Dificuldades

Os logs usados neste trabalho foram previamente selecionados, uma vez que alguns

logs coletados não apresentavam condições para serem usados por seus baixos tamanho de conteúdo.

Na analise dos dados dos logs e geraçao dos arquivos One, percebemos que os logs coletados de um dos celulares não estavam gerando arquivos no One, pois este celular(da integrante Daniele) em particular tinha uma versão mais recente do aplicativo *My Tracks* em relação ao celular fornecido pelo professor. No primeiro celular o tempo dos milésimos estava sendo fornecido no formato ponto flutuante quanto que no segundo estava no formato inteiro, resolvemos padronizar no formato inteiro, o que solucionou o problema.

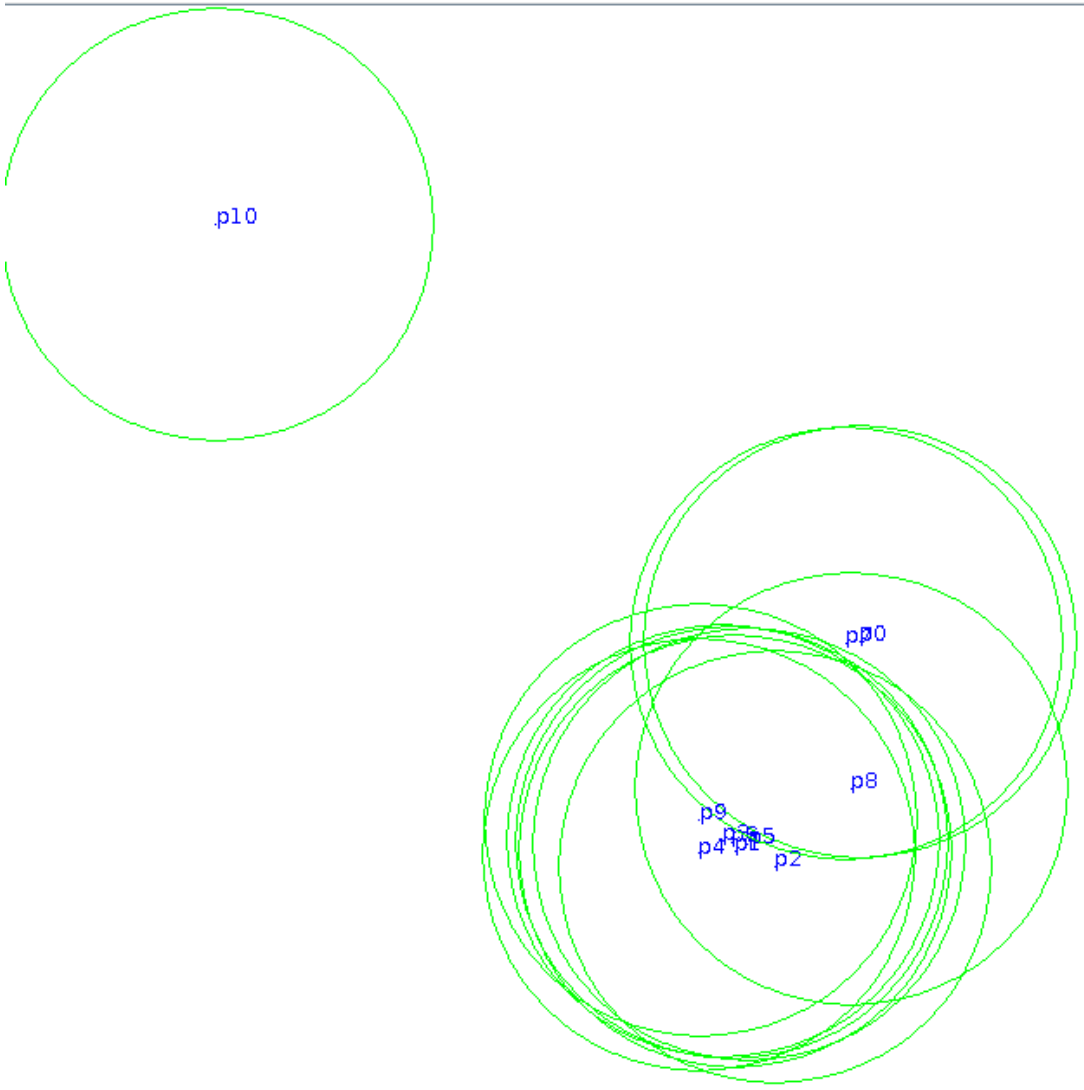
Os arquivos coletados geraram ao final um arquivo ordenados.gpx.one que foi utilizado para rodar a simulação.

Encontramos alguns problemas em um dos arquivos .gpx onde algumas coordenadas apareceram com erro causados porque o trace coletado estava fora dos limites definidos no modelo.

Eliminamos esses registros para que fosse possível abrir o modelo no One.

Ocorreram outros erros ao executar a simulação que, orientados pelo monitor Diogo Soares, corrigimos ajustando o arquivo default_settings.txt do One pois no arquivo não tínhamos configurado ainda todas as variáveis necessárias para rodar a simulação, como tamanho da área do modelo.

Ao executar a simulação no One notamos que os hosts ficavam sempre muito próximos uns dos outros porque os dispositivos foram distrubuídos entre os alunos do mesmo curso, para uma simulação melhor a aplicação MyTracks deve estar rodando em dispositivos de pessoas de diversas áreas da universidade.




## 5. Referências Bibliográficas

[ 1 ] BIGWOOD, Greg; HENDERSON, Tristan - Social DNT Routing. School of Computer Science University of ST. Andrews, Fife, UK, 2009.

[ 2 ] OLIVEIRA, C. T., MOREIRA, M. D. D., RUBINSTEIN, M. G., COSTA, L. H. M. K., E DUARTE, O. C. M. B. - Redes Tolerantes a Atrasos e Desconexões. Em 78 Minicursos do Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos (SBRC'07) (maio de 2007), pág. 203–256.

[ 3 ] J. Y. Le Boudec. Understanding the Simulation of. Mobility Models with Palm Calculus. In Proc. of IEEE Infocom ' 23, June, 2005.

[ 4 ] MEDEIROS, Marcus Vinícius; SALLES, Ronaldo - Emprego de Redes Tolerantes a Atrasos e Desconexões em Sistemas de Comunicações Militares. Em Instituto de Engenharia Militar, Rio de Janeiro (Março de 2009)

[5] C. A. V. Campos, R. M. S. Fernandes e L. F. M. Moraes,"Uma avaliação das rede tolerantes a atrasos e desconexões através de traces reais de mobilidade humana" Simpósio Brasileiro de Redes de Computadores e Sistemas, Distribuídos, 27, 2009, Recife, PE, Brasil.

[6] RASHID, Sulma; ABDULLAH, A.H; ZAHID, M. S.; AVUB, Oaisar; 3 ed. (2012), pp. 396-407, disponível em www.europeanjournalofscientificresearch.com/ISSUES/EJSR_70_3_07.pdf

[7]ANDRADE, Esdras; Instalação: Geoprocessamento para Linux, disponível em http://geoparalinux.wordpress.com/category/instalacao/page/3/

[8] http://transition.fcc.gov/mb/audio/bickel/DDDMMSS-decimal.html

# Extras

**1. Reuniões da Equipe**

**Data:** 24.04
**Horário:** 15:00 às 16:00 horas

**Abordagens:** Conhecimento do problema, Abordagem das politicas e como funcionam, definição de ideias para implementação.

**Data:** 28.04
**Horário:** 17:00 às 19 horas

**Abordagens:** Planejamento do Modelo, Implementação de Políticas de Descarte.

**Data:** 05.05
**Horário:** 10:00 às 13 horas

**Abordagens:** Definição do Modelo, Verificação do log do programa My Tracks, Desenvolvimento do Relatório.

## DIFICULDADES

- Sábado, cinco de maio, enfrentamos a dificuldade na elaboração do trabalho devido à falta de internet na UFAM.

- O Aplicativo ***My Tracks*** não guarda log de comunicação e troca de mensagens entre os aparelhos. Se guardam não conseguimos localizar essas informações no aparelho o que nos levou a suposição que o aplicativo é impróprio / inviável para o experimento o qual o professor deseja realizar.
- Na verdade havíamos entendido que esses logs já nos retornariam diretamente os dados referente ao possível encontro de dois ou mais nós, que nos forneceria o tempo de contato e possíveis trocas de informações entre os mesmos. Após conversar com o Diogo sobre os dados colhidos pelos logs, ficamos cientes de que o log só

nos forneceria os caminhos percorridos por cada nó. E que já estamos pensando na simulação em si.

- E seguida apareceu uma outra dificuldade: como iríamos usar estes logs no One. E após conversar com o Camilo é que ficou claro como iríamos dar procedimento ao trabalho da forma correta que se estava sendo solicitado, pois até então estamos fazendo o trabalho sem muita segurança quantos aos passos finais.
- Quando tentamos gerar os arquivos gpx.one no One, não estamos conseguindo por incompatibilidade de dado. E após comparamos o nosso arquivo ao arquivo descrito no manual (para gerar um modelo de mobilidade real para One utilizando GPS fornecido pelo Diogo, através do Camilo) percebemos que estava faltando algumas variáveis e seus valores, estes se tratavam .