UnB

CIC0203 - Computação Experimental - TA - 2022.2 - Tarefa T6 - Aprimoramento de uma Simulação

URL Read-only Overleaf: https://www.overleaf.com/read/rbvrrxytdqwm

Paulo Mauricio Costa Lopes (RequiemDosVivos)

Brasília, 2023-02-01 02:54:28Z

Lista de tarefas pendentes

Sumário

Lista de Figuras

Lista de Tabelas

Resumo

Este documento contém o produto da tarefa especificada no título deste documento, conforme as orientações em https://www.overleaf.com/read/cytswcjsxxqh.

Parte I

Preparação

Capítulo 1

Orientações Iniciais (Este capítulo não deve estar presente no documento de entrega da tarefa). Comente a correspondente linha de input

Leia atentamente as orientações a seguir, tendo em vista que lhe auxiliarão no melhor desempenho neste curso-disciplina.

1.1 Importância deste documento

Faça um clone deste documento na sua conta overleaf (comando *copy*), e ele doravante será chamado **seu documento Overleaf**, e conterá o registro de todas as suas evidências de aprendizagem na disciplina-turma de Computação Experimental, a partir da tarefa T4.

Trate o seu **seu documento Overleaf** como um ambiente experimental, de laboratório, e considere que a organização regular do laboratório será essencial para o seu bom desempenho.

Desse modo, quando for editar **seu documento Overleaf** tome cuidado para sempre deixá-lo em plena condição compilável, sem Erros, e com um mínimo de *warnings*, os quais podem prejudicar a avaliação das suas atividades.

O documento a ser clonado para a criação do seu **seu documento Overleaf** atualmente contém apenas um *warnings*, e nenhum erro.

Antes de encerrar seu trabalho na realização das tarefas da disciplina resolva qualquer Erro ou *warning* causados pela sua edição, com especial atenção para duplicidade de rótulos. A introdução de qualquer erro ou *warning* que prejudicar o seu trabalho poderá implicar em penalização das tarefas.

O Professor deve ter acesso à URL que permite a edição do **seu documento Overleaf**. Todos os demais colegas da disciplina terão acesso de leitura a este documento, mas não de gravação. Dessa forma. sempre observe a **Lista de Tarefas Pendentes**, no início do

documento, para verificar se há alguma pendência associada ao seu usuário, que pode ter sido inserida pelo professor.

1.2 Conteúdo do seu documento Overleaf

Toda e qualquer inserção de texto, programa de computador, dados, enfim, qualquer documento, feito por estudante no **seu documento Overleaf**, deve ocorrer em um dos seguintes pontos:

1. Dentro de um subdiretório **estudantes** correspondente à tarefa em execução;

2. Dentro do diretório *exploratory-data-analysis* do estudante, onde o nome do diretório é o seu github username (veja, por exemplo, o professor, que tem como github username: jhcf);

3. No arquivo "packages-estudantes.tex", onde eventualmente podem ser inseridos novos pacotes para apoiar o uso de algum recurso específico; e

4. Na substituição do arquivo RESIC.bib por outro mais recente, obtido pela exportação completa da biblioteca RESIC que se encontra na plataforma Zotero, na url https://www.zotero.org/groups/2465026/resic.

Cada um dos arquivos correspondentes às tarefas que você realizará neste "laboratório" deverá ser acessível por um arquivo LaTeX no formato:

`main-<NumeroDaTarefa>-<NomeDaTarefa>.tex`

Este arquivo deve estar no diretório raiz do **seu documento Overleaf**, onde `<NumeroDaTarefa>` e `<NomeDaTarefa>` são, respectivamente, o número e o nome da tarefa, conforme informado no título do capítulo que descreve a tarefa, por exemplo **T4** e **Análise-Bibliométrica**, resultando no nome do arquivo como a seguir:

`main-T4-Análise-Bibliométrica.tex`

1.3 Geração dos documentos correspondentes às tarefas

Uma vez que tenha desenvolvido e compilado a tarefa no **seu documento Overleaf**, baixe o arquivo do documento correspondente à tarefa, em formato PDF, contendo o texto do capítulo e a sua correspondente bibliografia, e renomeie o arquivo para a forma:

`main-<NumeroDaTarefa>-<NomeDaTarefa>-<githubusername>.pdf`

, onde `<githubusername>` é o seu username no github.

Por exemplo, se **T4** e **Análise-Bibliométrica** representam a tarefa a ser entregue, e `jhcf` é o seu username, o nome do arquivo com o PDF da tarefa deve ser, como a seguir.

`T4-Análise-Bibliométrica-jhcf.pdf`

1.4 Uso sincronizado de Repositório no Github e Overleaf: Repositório origin

O repositório origin da disciplina está na na url:
https://github.com/jhcf/Comput-Experim-20222.

Estando de posse do arquivo PDF correspondente à tarefa a ser entregue, faça o que se pede:

1. Clone, se ainda não fez, o Repositório git **origin** em seu computador, em um diretório apropriado;

```
git clone https://github.com/jhcf/Comput-Experim-20222.git
```

2. Sempre atualize o repositório local (seu clone), com o comando `git pull`, antes de editar o arquivo main.tex;

3. Mova o arquivo PDF correspondente à tarefa para a área de depósito no diretório **estudantes** correspondente à tarefa;

4. Atualize o arquivo main.tex no mesmo diretório de entrega da tarefa, para que o seu documento seja inserido na compilação, usando o comando a seguir, onde `<diretório>` é o diretório da especificação da tarefa, e o número e nome da tarefa, juntamente com o githubusername são conforme indicados anteriormente:

```
\includepdf[pages=-]{<diretório>/estudantes/main-<NumeroDaTarefa>-<NomeDaTarefa>-<githubusername>.pdf}
```

5. Adicione ao repositorio local o arquivo correspondente à execução da tarefa (comando `git add`) e faça o commit (comando `git commit`), registrando a mensagem no *commit* conforme as instruções contidas no capítulo de especificação da tarefa;

6. Envie as atualizações para o repositório origin (comando `git push`), e resolva qualquer problema de merge que possa a vir ocorrer; e

7. Uma vez resolvido qualquer problema de merge, informe na tarefa no aprender.unb.br, o número do commit do repositório origin.

Parte II

Simulação Computacional

Capítulo 2

T6 - Aprimoramento de uma Simulação: Laboratório e Experimento virus_on_network, por Paulo Mauricio Costa Lopes (RequiemDosVivos)

2.1 Introdução

O modelo que será objeto de estudo neste trabalho é o de **virus_on_network** "A virus model with some number of agents entering in contact through a network of relations", um modelo multi-agente de conexão e infecção para COVID-19. Esse trabalho foi adaptado para ser usado em infecção de conectividade de computadores.

O estudo a seguir é composto pelas seguintes seções:

1. Descrição do fenômeno real;
2. Apresentação do laboratório de simulações;
3. Apresentação de análises exploratórias dos dados de experimentos realizados com o uso do laboratório;
4. Discussão sobre *insights* obtidos após os experimentos; e
5. Conclusões.

2.2 O Fenômeno do Mundo Real

O fenômeno trabalhado no estudo a seguir trata-se da infecção por vírus em uma rede de computadores e as subsequentes interações que podem ser observadas na interação entre os

dispositivos dentro dela. Diversas redes podem estar suscetíveis a este cenário diariamente caso haja a possibilidade de um componente dela ser infectado.

Este tipo de contágio por malwares pode acontecer em dispositivos na mesma rede de forma acidental (no caso de um download acidental do malware), ou de forma proposital (em caso de sabotagem). De qualquer forma, faz-se necessário que haja um monitoramento das comunicações dos dispositivos da rede para que possam ser detectadas irregularidades e faz-se necessário estudar como que uma rede de computadores pode ser afetada nesse caso.

2.3 O Laboratório virus_on_network

O laboratório virus_on_network é um modelo genérico que simula a transmissão de um vírus em uma rede e pode ser utilizado tanto para simular uma infecção viral como uma infecção por um vírus em uma rede de computadores.

Assim, fez-se a oportunidade de realizar um estudo semelhante utilizando esta ferramenta para simular uma "epidemia"em uma rede de computadores.

2.3.1 O Conceito da Simulação

O código base utilizado está presente no repositório do *GitHub* denominado *projectmesa/mesa-examples/Virus_on_network* link para o repositório.

O modelo foi composto de forma a representar cada sistema de uma rede de conexões como um agente, tendo assim uma rede de agentes interconectados. Dessa forma, desenha-se um grafo onde cada vértice é um dispositivo e as arestas correspondem às conexões entre eles.

Esse modelo proposto foi então utilizado como base para elaborar uma simulação que retrata a dispersão de um vírus na rede proposta. Para definir o comportamento desta simulação, o problema foi dividido um conjunto de variáveis, selecionadas de forma a isolar os fatores centrais que definem o comportamento do modelo para o problema escolhido.

Assim foram selecionadas 5 variáveis dependentes e 8 variáveis independentes.

1. nº de agentes
2. media de nodos vizinhos
3. numero inicial de infectados
4. chance de contágio
5. frequência de checagem por vírus
6. chance de recuperação
7. chance de resistencia
8. numero inicial de *watchdogs*

1. Infectado
2. Susceptible
3. Resistant
4. Watchdog
5. Dorment

2.3.2 O Simulador

No código estão presentes as seguintes variáveis:

num_nodes contém o número de agentes presentes no modelo.

avg_node_degree representa o número médio de arestas entre os agentes, ou seja, o número médio de conexões entre os computadores.

initial_outbreak_size define a quantidade inicial de aparelhos infectados.

virus_spread_chance estipula a chance de contágio do vírus.

virus_check_frequency define a frequência com que um watchdog procura por virus em computadores proximos.

recovery_chance representa a chance de um computador ser recuperado da infecção.

gain_resistance_chance contém a chance de um agente mudar seu estado para

resistant, representando a chance de o usuário do computador ter detectado a infecção e, além de ter mitigado o ataque, ter criado uma proteção.

dorment, é um computador infectado porém ele esta dormente (não conseguiu estabelecer conexão com outro computador).

watchdog é o computador responsavel por monitorar os nós proximos.

A seguinte imagem **??** mostra interface gráfica do modelo:

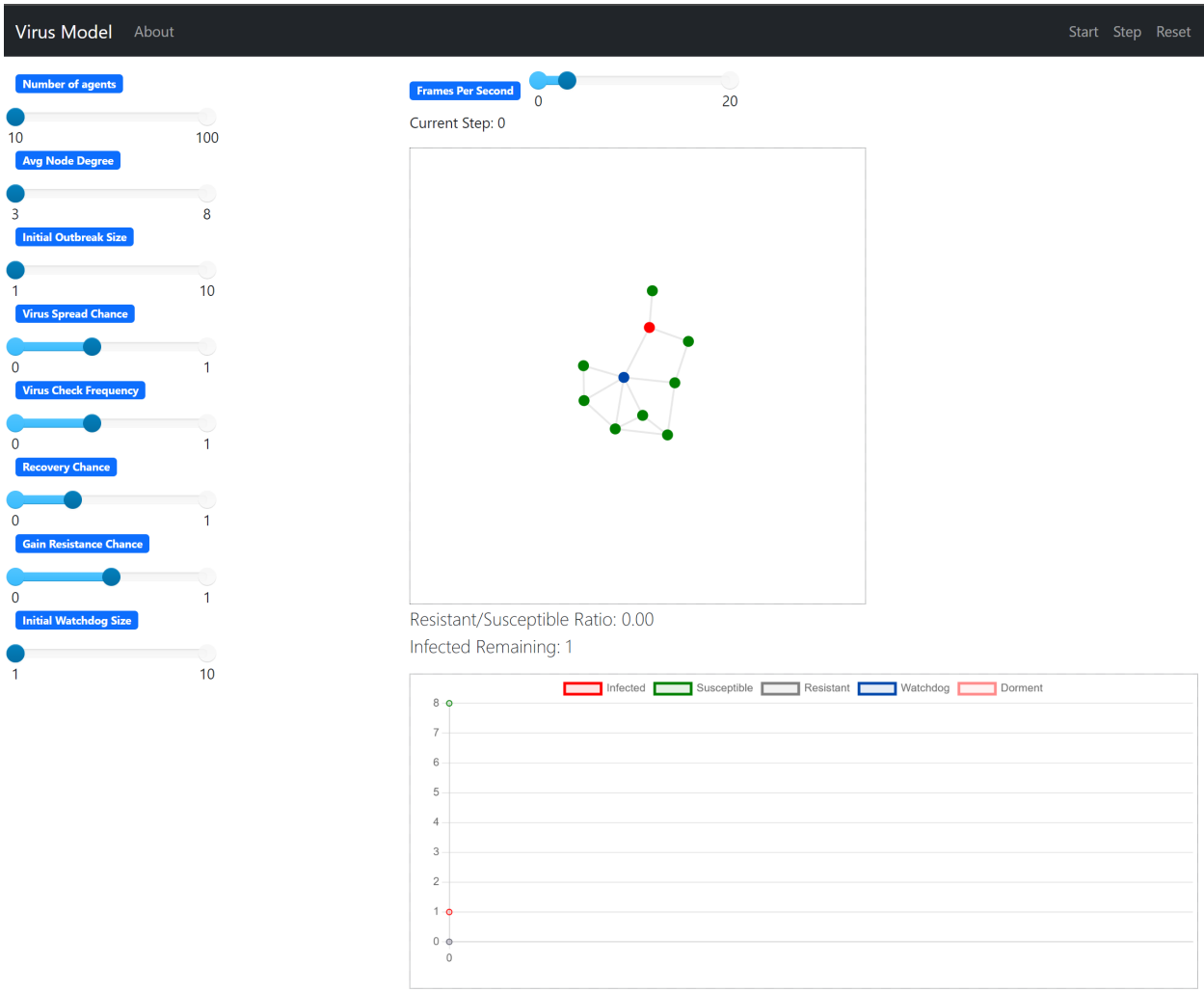

Figura 2.1: Interface do modelo

2.3.2.1 Variáveis Independentes ou de Controle

São as seguintes as variáveis Independentes ou de Controle, manipuláveis na interface gráfica do simulador:

Number of agents define a quantidade de agentes na simulação, ou seja, o quantidade de vértices no grafo, em um intervalo de 10 a 100.

Avg Node Degree é a quantidade média de arestas que cada vértice possui, em um intervalo de 3 a 8.

Recovery Chance, define a probabilidade de um agente se recuperar (cor verde) (*suscetible*) dado que está com a cor vermelha (*infected*), em um intervalo de 0 a 1.

Gain Resistance Chance, controla a probabilidade de um agente tornar-se resistente (cor cinza) (*resistant*), em um intervalo de 0 a 1.

Initial watchdog size, controla a quantidade inicial de watchdogs (cor azul).

Initial Outbreak Size controla a quantidade de agentes na cor vermelha (*infected* no primeiro passo da simulação, em um intervalo de 1 a 10.

Virus Spread Chance define a probabilidade de um agente infectado infectar seus vizinhos (representado por vermelho, em um intervalo de 0 a 1.

Virus Check Frequency controla a probabilidade da simulação mudar o estado de um agente para *infected*, em um intervalo de 0 a 1.

Virus spread chance, é usado para determinar se o vírus se espalhou ou não pelo sistema.

2.3.2.2 Variáveis Dependentes

São as seguintes as variáveis Dependentes, cujos valores são coletados e apresentados na interface gráfica do simulador:

Infected representa a quantidade de agentes infectados.

Dorment represente infectados que estão dormentes no sistema.

Suscetible representa a quantidade de agentes suscetíveis a serem infectados.

Resistant representa a quantidade de agentes resistentes ao vírus, ou seja, mesma que tenham contato com um agente infectado, esses agentes resistentes não serão infectados.

Watchdog representa a quantidade de agentes responsáveis por monitorar as atividades da rede.

2.3.3 A Hipótese Causal

Quanto ao comportamento da rede no cenário proposto, houveram as seguintes suposições:

Um malware infectando uma rede altamente conexa consegue infectar mais elementos do que uma rede com baixa conectividade (em questão de topologia)?

Em uma rede que faz uso de dispositivos **"watchdogs"**, é possível que determinados dispositivos fiquem no "ponto cego"do dispositivo e o comportamento anormal jamais seja mitigado?

Culminando na seguinte hipótese causal : “a topologia de uma rede pode afetar o processo de mitigação de malwares."

2.3.4 O Código do Simulador

Com o intuito de comprovar a veracidade da hipótese causal, adicionei uma variável independente *Virus Letal Chance* e uma variável dependente Dead para aprimorar o modelo e assim simular a infecção por vírus com chances de letalidade diferentes.

As classes presentes no simulador são:

State, classe responsável por controlar o estado de cada agente e possui os atributos *SUSCETIBLE = 0*, *INFECTED = 1*, *RESISTANT = 2*, *WATCHDOG = 3* e *DORMENT = 4*.

VirusAgent, classe resposável por declarar os agentes.

VirusOnNetwork, classe resposável pelo modelo em si, bem como a coleta de dados. Essa classe instancia os agentes.

```
class State(Enum):
    SUSCEPTIBLE = 0
    INFECTED = 1
    RESISTANT = 2
    WATCHDOG = 3
    DORMENT = 4
```

VirusOnNetwork, trecho do código:

```
class VirusOnNetwork(mesa.Model):
    """A virus model with some number of agents"""

    def __init__(
        self,
        num_nodes=10,
        avg_node_degree=3,
        initial_outbreak_size=1,
        virus_spread_chance=0.4,
        virus_check_frequency=0.4,
        recovery_chance=0.3,
        gain_resistance_chance=0.5,
  initial_watchdog_size=1,
    ):

        self.num_nodes = num_nodes
        prob = avg_node_degree / self.num_nodes
        self.G = nx.erdos_renyi_graph(n=self.num_nodes, p=prob)
```

```
        self.grid = mesa.space.NetworkGrid(self.G)
        self.schedule = mesa.time.RandomActivation(self)
        self.initial_outbreak_size = (
            initial_outbreak_size if initial_outbreak_size <= num_nodes else num_nodes
        )
        self.virus_spread_chance = virus_spread_chance
        self.virus_check_frequency = virus_check_frequency
        self.recovery_chance = recovery_chance
        self.gain_resistance_chance = gain_resistance_chance
        self.initial_watchdog_size = (
            initial_watchdog_size if initial_watchdog_size <= num_nodes else num_nodes
        )

        self.datacollector = mesa.DataCollector(
            {
                "Infected": number_infected,
                "Susceptible": number_susceptible,
                "Resistant": number_resistant,
            }
        )

        # Create agents
        for i, node in enumerate(self.G.nodes()):
            a = VirusAgent(
                i,
                self,
                State.SUSCEPTIBLE,
                self.virus_spread_chance,
                self.virus_check_frequency,
                self.recovery_chance,
                self.gain_resistance_chance,
#                self.watchdog,
            )
            self.schedule.add(a)
            # Add the agent to the node
            self.grid.place_agent(a, node)

        # Create some watchdogs
        watchdog_nodes = self.random.sample(list(self.G), self.initial_watchdog_size)
        for a in self.grid.get_cell_list_contents(watchdog_nodes):
            a.state = State.WATCHDOG
```

```
        # Infect some nodes
        infected_nodes = self.random.sample(list(self.G), self.initial_outbreak_size)
        for a in self.grid.get_cell_list_contents(infected_nodes):
            a.state = State.INFECTED


        self.running = True
        self.datacollector.collect(self)

    def resistant_susceptible_ratio(self):
        try:
            return number_state(self, State.RESISTANT) / number_state(
                self, State.SUSCEPTIBLE
            )
        except ZeroDivisionError:
            return math.inf

    def step(self):
        self.schedule.step()
        # collect data
        self.datacollector.collect(self)

    def run_model(self, n):
        for i in range(n):
            self.step()
```

VirusAgent, alguns comportamentos são descritos aqui, como dorment() e *try_ remove_ infection()* que foi alterado para realizar o comportamento desejado.

```
class VirusAgent(mesa.Agent):
    def __init__(
        self,
        unique_id,
        model,
        initial_state,
        virus_spread_chance,
        virus_check_frequency,
        recovery_chance,
        gain_resistance_chance,
    ):
        super().__init__(unique_id, model)
```

```
        self.state = initial_state

        self.virus_spread_chance = virus_spread_chance
        self.virus_check_frequency = virus_check_frequency
        self.recovery_chance = recovery_chance
        self.gain_resistance_chance = gain_resistance_chance

    def try_to_infect_neighbors(self):
        neighbors_nodes = self.model.grid.get_neighbors(self.pos, include_center=False)
        susceptible_neighbors = [
            agent
            for agent in self.model.grid.get_cell_list_contents(neighbors_nodes)
            if agent.state is State.SUSCEPTIBLE or agent.state is State.DORMENT
        ]
        for a in susceptible_neighbors:
            if a.state is State.SUSCEPTIBLE:
                if self.random.random() < self.virus_spread_chance:
                    a.state = State.INFECTED
            if a.state is State.DORMENT:
                a.state = State.INFECTED

    def try_gain_resistance(self):
        if self.random.random() < self.gain_resistance_chance:
            self.state = State.RESISTANT

    def try_remove_infection(self):
        # Try to remove
        neighbors_nodes = self.model.grid.get_neighbors(self.pos, include_center=False)
        infected_neighbors = [
            agent
            for agent in self.model.grid.get_cell_list_contents(neighbors_nodes)
            if agent.state is State.INFECTED or agent.state is State.DORMENT
        ]

        for neighbor in infected_neighbors:
             if self.random.random() < self.recovery_chance:
                 neighbor.state = State.SUSCEPTIBLE
                 neighbor.try_gain_resistance()

    def try_check_situation(self):
```

```
        if self.random.random() < self.virus_check_frequency:
            # Checking...
            if self.state is State.WATCHDOG:
                self.try_remove_infection()



    def dorment(self):
        neighbors_nodes = self.model.grid.get_neighbors(self.pos, include_center=Fals
        state_neighbors = [
            agent
            for agent in self.model.grid.get_cell_list_contents(neighbors_nodes)
            if agent.state is State.INFECTED or agent.state is State.SUSCEPTIBLE
        ]
        if len(state_neighbors)==0:
            self.state = State.DORMENT



    def step(self):
        if self.state is State.INFECTED:
            self.try_to_infect_neighbors()
            self.dorment()
        self.try_check_situation()
```

2.4 Os Experimentos Realizados

Foram feitos 2 experimentos principais para verificar a veracidade da hipótese causal:

Experimento I: A variável *Avg Node Degree* com o valor máximo e as outras variáveis com um valor aleatórios (diversas amostras) e depois com o menor valor possível. Objetivo, ver o impacto da variável no modelo de simulação.

Experimento II: A variável *Initial Watchdog Size* com o valor máximo, mínimo e depois cerca de 10% do valor de *Number of agents*, com *Virus Spread Chance* mínimo, médio e máximo. Esse experimento teve intuito de verificar o impacto que os watchdogs tem em relação a infecção (os demais atributos foram de valores aleatórios).

Além desses experimentos foram feitos outros experimentos para tentar encontrar pontos chaves em que o sistema conseguisse maior numero de infectados ou menor numero de infectados.

A baixo temos dois gráficos, um realizando o experimento com baixa conectividade e outro realizando com o máximo de conectividade

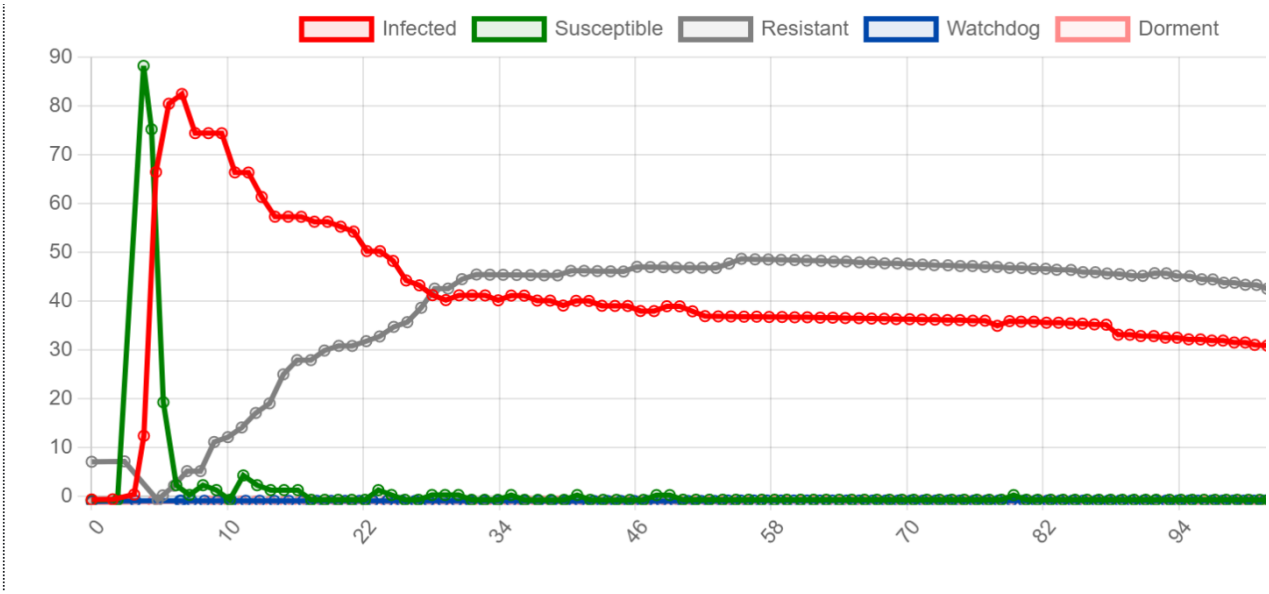

Figura 2.2: Gráfico Alta Conectividade

Figura 2.3: Gráfico Baixa Conectividade

2.5 Discussão e *insights* preliminares sobre as hipóteses

Levando em conta os resultados obtidos com o sistema simplificado proposto para a simulação, vê-se uma série de comportamentos interessantes.

Primeiro vê-se que como proposto na hipótese causal, a topologia afetou significativamente o numero de maquinas que se infectaram e a velocidade com a qual a mitigação ocorria.

Verificou-se que ao contrário do que se esperava, redes de menor porte com menor interconectividade tiveram maior numero de maquinas que permaneceram infectadas após atuação dos watchdogs.

Também se constatou que ainda houveram máquinas infectadas que continuaram se comunicando após a tentativa de mitigação em determinados casos, e que isso está realmente mais relacionado com o a presença de menores números de dispositivos watchdogs.

Por fim, faz-se válido comentar que houveram dispositivos que nunca chegaram a ser infectados pelo vírus, sendo por que apenas estavam conectados com as watchdogs ou porque estes tornaram o único caminho até uma maquina isolada resistente.

2.6 Conclusão

Conseguimos confirmar algumas hipóteses enquanto outras foram negadas por nossos experimentos. É interessante ressaltar que apesar do fato de que as redes com alta conectividade (topologicamente) foram as redes com menor numero de infectados finais elas também foram as redes com maior pico de infectados, talvez isso implique que elas são redes com melhor recuperação dos sistemas mas são redes mais suscetíveis a ataques quando comparadas a redes com baixa conectividade.