Universidade Federal Fluminense
Escola de Engenharia
Curso de Graduação em Engenharia de Telecomunicações

Daniel Rocha de Castro

Redes Neurais Artificiais do Tipo Hopfield-Tank e o Balanceamento de Carga em Roteadores

Niterói – RJ

Dezembro de 2017

Daniel Rocha de Castro

Redes Neurais Artificiais do Tipo Hopfield-Tank e o Balanceamento de Carga em Roteadores

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Engenharia de Telecomunicações da Universidade Federal Fluminense, como requisito parcial para obtenção do Grau de Engenheiro de Telecomunicações.

Orientador: Prof. Dr. João Marcos Meirelles da Silva

Niterói – RJ

Dezembro de 2017

.

A figura referente ao arquivo

FichaCatalografica.jpg

fornecido pela Biblioteca

deverá aparecer aqui.

ATENÇÃO: Na versão impressa, essa página deverá ficar no verso da página anterior.

.

Daniel Rocha de Castro

Redes Neurais Artificiais do Tipo Hopfield-Tank e o Balanceamento de Carga em Roteadores

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Engenharia de Telecomunicações da Universidade Federal Fluminense, como requisito parcial para obtenção do Grau de Engenheiro de Telecomunicações.

Aprovada em 15 de Dezembro de 2017.

BANCA EXAMINADORA

Prof. Dr. João Marcos Meirelles da Silva - Orientador
Universidade Federal Fluminese - UFF

Prof. Dra. Natália Castro Fernandes
Universidade Federal Fluminese - UFF

Prof. Dr. Ricardo Campanha Carrano
Universidade Federal Fluminese - UFF

Niterói – RJ
Dezembro de 2017

Resumo

Neste trabalho, o problema do balanceamento de carga é modelado de forma a ser representado por uma rede neural artificial conhecida como Modelo de Hopfield-Tank com atrasos. Um modelo experimental capaz de simular os aspectos relevantes de interfaces de rede é proposto para que o algoritmo de balanceamento possa ser testado. Diversos cenários são simulados para verificar se o algoritmo é eficaz no balanceamento de carga entre interfaces de rede. Os resultados são examinados e demonstram que o modelo é válido para todos os casos testados.

Palavras-chave: Redes neurais artificiais. Balanceamento de carga. Hopfield-Tank.

Abstract

In this article, the load balancing problem is modeled by an artificial neural network known as Delayed Hopfield-Tank network. An experimental model is devised in order to simulate the relevant aspects of network interfaces, so that the load balancing algorith can be tested. Multiple scenarios are simulated to verify if the algorithm is efficient when balancing loads between network interfaces. The results are then examined to show that the model is valid on all the testes cases.

Keywords: Artificial neural networks. Load balancing. Hopfield-Tank.

Lista de Figuras

Lista de Tabelas

Sumário

Capítulo 1

Introdução

1.1 Introdução

A necessidade de nos comunicarmos à distância existe desde a formação das primeiras civilizações humanas. Durante milhares de anos as únicas alternativas disponíveis envolviam o deslocamento de pessoas de uma ponta à outra da conversação, processo que poderia levar dias ou até meses para trajetos de longa distância.

Somente nos últimos séculos, com a invenção do telégrafo, fomos capazes de interligar cidades distantes com canais de comunicação instantâneos, finalmente encurtando as distâncias entre países e permitindo uma propagação mais rápida de informações importantes.

Desde então, os meios de comunicação tem evoluído de forma tão acelerada que a maioria das tecnologias se torna obsoleta dentro de poucos anos.

A demanda por redes de comunicação robustas e de alta velocidade é enorme e todos os setores da economia global que conhecemos hoje são dependentes do funcionamento destas redes.

Diversos desafios são enfrentados diariamente pelos milhares de engenheiros responsáveis pela criação e manutenção dos sistemas de telecomunicações, que precisam constantemente desenvolver novas técnicas para lidar com quantidades cada vez maiores de dados.

Dentre estes desafios, o balanceamento de carga se destaca por não possuir uma solução única estabelecida, abrindo espaço para o desenvolvimento de novas pesquisas.

1.2 O problema de balanceamento de carga em roteadores

O balanceamento de carga em roteadores se refere à capacidade de distribuir tráfego de rede entre dois ou mais enlaces sem o auxílio de protocolos complexos de roteamento como o BGP.

Especialmente em empresas e órgãos governamentais, é prática cada vez mais comum a contratação de links de Internet redundantes, ou seja, aqueles onde existem pelo menos duas conexões físicas diferentes entre o cliente e o prestador de serviço.

Muitas vezes, as redes são configuradas para operarem em modo *Hot Standby*, onde somente um dos enlaces é realmente utilizado para comunicação, mas todo o tráfego é replicado entre as outras. No caso do enlace principal falhar, outra interface pode assumir o tráfego instantaneamente. Este método garante que a conexão somente será interrompida se todos os enlaces falharem simultaneamente, mas desperdiça parte da banda total disponível tendo em vista que ao menos um enlace ficará ocioso.

Idealmente, todos os enlaces disponíveis poderiam trabalhar paralelamente, de forma a criar um único enlace lógico com taxa de transmissão que se aproxima das taxas somadas dos seus componentes. Esta abordagem pode preservar as características de redundância do modelo *Hot Standby*, desde que o tráfego seja desviado do enlace defeituoso em caso de queda.

Entretanto, dividir o trabalho de forma eficiente entre os recursos disponíveis não é algo trivial. Um algoritmo de balanceamento de carga deve ser capaz distribuir o tráfego de forma que nenhum enlace fique ocioso enquanto houver informação a ser transmitida.

1.3 Motivação

A crescente demanda por redes de comunicação mais velozes e robustas impulsiona pesquisas em diversas áreas da engenharia e ciências da computação.

Sistemas que possuem mais de um enlace de rede conectando dois pontos são candidatos a necessitar de alguma forma de balanceamento de carga. Um sistema bem balanceado irá apresentar maiores taxas médias de transmissão, menores latências de comunicação e será invisível ao usuário final.

Os problemas de balanceamento de carga são classificados como NP-HARD, o que significa que em muitos casos práticos não será possível obter uma solução ótima apenas com modelos formais. Por este motivo, problemas de balanceamento de carga comumente são resolvidos utilizando heurísticas que tentam obter soluções quase ótimas.

1.4 Objetivo

Este trabalho possui três objetivos:

1. Demonstrar que o problema de balanceamento de carga entre as filas de interfaces de roteadores pode ser visto como uma rede neural artificial do tipo Hopfield-Tank;

2. Desenvolver um mecanismo capaz de simular o balanceamento de carga entre as interfaces de rede;

3. Encontrar os valores de pesos da rede neural artificial de forma a promover um balanceamento quase ótimo.

A vantagem de reconhecer problemas atuais que possam ser modelados como redes neurais artificiais decorre da grande variedade de teorias, ferramentas matemáticas e computacionais eficientes que refletem a maturidade do estudo nessa área.

1.5 Revisão Bibliográfica

Dentre os diversos modelos disponíveis na literatura para tratar o problema do balanceamento de carga, destacam-se os desenvolvidos por *Ghanem et al.* [1] e *Hui et al.* [3] por incluírem em seus modelos sistemas heterogêneos.

O modelo de *Ghanem et al* considera também os chamados atrasos de comunicação e atraso de transferência. O atraso de comunicação é o tempo transcorrido para um nó enviar uma informação, geralmente com poucos bytes, para outro nó. Já o atraso de transferência é o tempo transcorrido para um nó transferir uma determinada tarefa para outro nó. O atraso de transferência é, normalmente, muito maior que o atraso de comunicação.

O modelo de Hopfield-Tank, descrito em [4], para redes neurais pode ser utilizado para modelar diversos problemas, dentre eles o problema do balanceamento de carga como definido por Ghanem, desde que algumas considerações sejam feitas.

A relação entre os modelos de Ghanem e Hopfield-Tank é estabelecida por da Silva em [5] e [6], que se aproveita desta correspondência para desenvolver um método para calcular valores para os ganhos utilizados no modelo de Ghanem.

Capítulo 2

O modelo matemático

O problema de balanceamento de carga é inerentemente associado à filas ou distribuição de grandezas contínuas ou discretas entre estas. Isto ocorre naturalmente em diversas áreas como filas de supermercado, chamadas telefônicas em espera, evacuação de veículos em estacionamento e pessoas em recintos fechados, etc.

Embora diversos modelos matemáticos para o problema de balanceamento de carga entre filas existam, a maioria aborda a questão através do número de entidades (no caso desse trabalho: quadros) existentes em cada fila, o tempo de processamento de cada quadro, que pode ser definido pelo tempo que um processador retira o pacote da fila e o envia por uma interface, e o tempo necessário para retirar um pacote de uma fila para inserir em outra fila, etc.

2.1 O modelo de Ghanem

Embora diversos modelos matemáticos existam para o problema de balanceamento de carga, um de maior sucesso é o sugerido por *Ghanem et al.* [1], que aborda o problema pelo tempo de espera no lugar do número de quadros em cada fila:

$$\frac{dx_i(t)}{dt} = -\mu(x_i(t)) + v_i(y_i(t)) - \sum_{j=1}^{n} p_{ij}\frac{t_{p_i}}{t_{p_j}} v_j(y_j(t - h_{ij})) + \lambda_i \tag{2.1}$$

$$v_i(y_i(t)) = -K_i\phi(y_i(t)) \tag{2.2}$$

$$\phi(y_i(t)) = uhsat(y_i(t)) \tag{2.3}$$

$$y_i(t) = x_i(t) - \frac{\sum_{j=1}^{n} x_j(t - \tau_{ij})}{n} \tag{2.4}$$

onde

$$p_{ij} \equiv \frac{uhsat(x_j^{avg} - x_i(t - \tau_{ji}))}{\sum uhsat(x_j^{avg} - x_i(t - \tau_{ji})}, \quad i, j = 1, 2, \ldots, n \tag{2.5}$$

$$p_{ij} \geq 0, p_{jj} = 0, \sum_{i=1}^{n} p_{ij} = 1 \tag{2.6}$$

$$uhsat(y_i(t)) = \begin{cases} y_{max}, & \text{if} \quad y_i(t) > y_{max} \\ y_i(t), & \text{if} \quad 0 \leq y_i(t) \leq y_{max} \\ 0, & \text{if} \quad y_i(t) < 0 \end{cases} \tag{2.7}$$

Como definido em [1]:

- $x_i(t)$ é o tempo de espera de um quadro que é inserido na i-ésima fila.
- t_{p_i} é o tempo médio necessário para a i-ésima interface processar um quadro correspondente à sua respectiva fila. O tempo previsto de espera é dado por $x_i(t) = q_i(t)t_{p_i}$, e $q_j(t) = x_j(t)/t_{p_j}$ é o número de quadros na fila da interface j no tempo t.
- $\lambda_i \geq 0$ é a taxa de geração de tempo de espera na i-ésimo interface causada pela adição de quadros em sua fila.
- $\mu(x_i(t)) \geq 0$ é a taxa de redução de tempo de espera causado pela transmissão de quadros na i-ésima interface, e é definido como $\mu(x_i(t)) = 1$ para todo i se $x_i(t) > 0$ e $\mu(x_i(t)) = 0$ se $x_i(t) = 0$.
- $v_i(t)$ é taxa de remoção (transferência) das quadros da interface i no tempo t pelo algoritmo de balanceamento de carga na interface i.
- p_{ij} é a fração de quadros da interface j que devem ser transferidas para a interface i.
- A razão t_{p_i}/t_{p_j} converte tempo de espera de quadros na interface j para tempo de espera na interface i.

Neste modelo, todas as taxas estão em unidades de taxa de alteração de *tempo previsto de espera*, ou *tempo/tempo*, que é adimensional.

A Equação 2.2 pode ser vista como um controle não linear do problema de balanceamento de carga com matriz de ganhos diagonal $K_d = diag(k_i)$. Esta lei de controle afirma que se o tempo de espera $x_t(t)$ da i-ésima interface estiver acima da estimativa de média do sistema $x_i^{avg} = (\sum_{j=1}^{n} x_j(t - \tau_{ij}))/n$, então quadros são enviados para as outras interfaces, limitados pela capacidade do canal que corresponde a y_{max}, como definido na Equação 2.7. Caso contrário, nada é enviado.

A j-ésima interface recebe a fração $\int_{t_1}^{t_2} p_{ji}(t_{p_i}/t_{p_j})u_i(t)dt$ do tempo de espera transferido $\int_{t_1}^{t_2} u_i(t)dt$ atrasado pelo tempo h_{ij}, que é o atraso de transferência de quadros.

É importante frisar que, como $v_i(t) < 0$, a interface i só pode enviar quadros para outras interfaces, não podendo iniciar uma transferência de outras interface para ela. A lei de controle $v_i(t) = -K_i\phi(y_i(t))$ garante que se o tempo de espera da i-ésima interface $x_i(t)$ estiver acima da média local, então ela enviará quadros para as outras interfaces.

Os autores do modelo assumem as seguintes hipóteses:

- O consumo de quadros pelas interfaces acontece simultaneamente ao balanceamento delas;
- O número de quadros é suficientemente alto para que o tamanho das filas possa ser aproximado por uma variável contínua;
- O modelo considera que todas as interfaces conseguem trocar informações diretamente, como em uma topologia full-mesh ou barramento;
- As funções não lienares $\phi(y_i(t)) = uhsat(y_i(t))$ são as mesmas

Apesar de o modelo 2.1 descrever uma lei de controle passível de implementação em controladores de balanceamento de carga, os autores não apresentam uma forma de determinar os valores de ganho K baseado nesta lei de controle, tornando a escolha destes valores subjetiva.

2.2 O modelo de Hopfield-Tank

Através de seus trabalhos publicados, J. J. Hopfield e D. W. Tank apresentaram um modelo dinâmico para representar redes neurais artificiais, hoje conhecido como modelo

de Hopfield-Tank.

Este modelo possui uma interpretação física como um circuito elétrico contendo amplificadores não-lineares interligados por uma rede de ganhos lineares. Tal circuito pode ser comparado a uma rede neural artificial, pois cada amplificador não-linear pode ser visto como um modelo de uma função de ativação neural, e a rede de interconexão, como as sinapses.

A forma geral com que este circuito pode ser representado é dado por [4]:

$$C_i \frac{dx_i(t)}{dt} = -\frac{x_i(t)}{R_i} + \sum_{j=1}^{n} t_{ij} v_j(t) + I_i \tag{2.8}$$

$$v_j(t) = \phi(x_j(t)) \tag{2.9}$$

e, na forma matricial

$$C\dot{x}(t) = -Gx(t) + Tv(t) + I \tag{2.10}$$

Onde $C > 0$ é uma matriz diagonal de capacitâncias de entrada do aplificador neural; $G > 0$ é a matriz de admitâncias dos amplificadores; $T = t_{ij}$ é a matriz n x n real e constante de interconexão da rede; I é o vetor de correntes externas e constantes de entrada; $x = [x_1(t), ..., x_n(t)]^T$ é o vetor das tensões neurais e o vetor $v(t) = [v_1(t), ..., v_n(t)]^T$ são as tensões de saída dos neurônios.

O modelo apresentado por Hopfield na equação 2.8, na prática, apresentava comportamento instável em muitas ocasiões devido ao atraso de tempo enfrentado pelo sinal na realimentação.

Por este motivo, o modelo matemático foi revisado de forma a contemplar os efeitos do atraso, produzindo o modelo *Hopfield Delayed Neural Network*, apresentado em sua forma matricial na equação 2.11.

$$\dot{x}(t) = -x(t) + W\phi(x(t)) + W_h\phi(x(t-h)) + u \tag{2.11}$$

onde $x(t) = [x_1(t), ..., x_n(t)]^T$ é o vetor de estados, $\phi(x(t)) = [\phi_1(x_1(t), ..., \phi_n(x_n(t))]^T$ é o vetor de saída, $W = \{w_{ij}\}$ é a matriz de realimentação, $W_h = \{w_{h_{ij}}\}$ é a matriz de realimentação atrasada, $u = [u_1, ..., u_n]^T$ é o vetor constante e h é o atraso.

2.3 O modelo base

Os modelos matemáticos de Ghanem para o balanceamento de carga e o de Hopfield-Tank para redes neurais apresentam grandes similaridades. A Tabela 2.1 exposta em [5] compara os modelos termo a termo.

Modelo	1º termo	2º termo	3º termo	4º termo
Ghanem	$-\mu(x(t))$	$-K_d\phi(y(x(t)))$	$TK_d\phi(y(x(t-h)))$	λ
Hopfield-Tank	$-x(t)$	$W\phi(x(t))$	$W^h\phi(x(t-h))$	u

Tabela 2.1: Comparação entre os modelos matemáticos de Ghanem e Hopfield-Tank.

O primeiro termo do modelo 2.1 é uma função não linear e descontínua de $x(t)$, e não a própria variável de estado $x(t)$ como no modelo de Hopfield.

Além disso, o segundo e terceiro termos do modelo 2.1 são funções compostas do tipo $\phi \circ y \circ x$, enquanto no modelo de Hopfield a função composta é do tipo $y \circ x$.

Essas diferenças são estudadas em [5], que demonstra a equivalência dos dois modelos para o balanceamento de carga em clusters de processadores dado que as seguintes hipóteses sejam admitidas com relação ao modelo 2.1:

h1 O atraso de comunicação nos links, por ser geralmente muito menor que o atraso de transferência em um cluster, será desconsiderado;

h2 O atraso de transferência será considerado único, ou homogêneo;

h3 O termo correspondente ao consumo de tarefas $\mu(x)$ não será considerado porque não há sentido em falarmos de consumo de tarefas simultaneamente ao balanceamento de carga pois, ou o processador está utilizando o seu esforço para consumir tarefas ou para processar o balanceamento de carga;

h4 O termo p_{ij} será assumido como constante e definido por $p_{ij} = p = 1/(n-1)$;

h5 Será considerado que não há falha ou desconexão de processadores, consequentemente, o número n de processadores do sistema será constante;

h6 Apesar dos sistemas operacionais modernos atuarem com diversas filas (cada qual com uma prioridade diferente), trabalharemos com todas as filas como sendo uma

única pois estaremos considerando o tempo de espera estimado médio $x_i(t)$ para uma tarefa ser executada no i-ésimo processador.

As hipóteses 1, 2, 4, 5 e 6 tratam de proposições diretamente correlatas ao problema de balanceamento carga em interfaces de rede e podem ser aplicadas diretamente.

A hipótese 3 não foi levada em consideração no modelo experimental desenvolvido para o balanceamento de carga em interfaces rede, já que não é necessário interromper a transmissão de um quadro pelo meio físico para que a fila da interface seja manipulada.

O Capítulo 4 irá demonstrar que esta concessão não prejudicou os resultados obtidos pelo algoritmo.

O modelo de Ghanem apresentado na Seção 2.1, quando unido às considerações propostas acima para representa-lo de forma equivalente utilizando o modelo de Hopfield-Tank, produz o modelo que será empregado até o final deste trabalho, de agora em diante denominado Modelo Base:

$$\dot{x}(t) = v_i(y_i(t)) - \sum_{j=1}^{n} v_j(y_j(t-h)) + \lambda_i \tag{2.12}$$

onde

$$v_i(y(t)) = -K_i \phi(y_i(t)) \tag{2.13}$$

$$\phi(y_i(t)) = uhsat(y_i(t)) \tag{2.14}$$

$$y_i(t) = x_i(t) - \frac{\sum_{j=1}^{n} x_j(t)}{n} \tag{2.15}$$

$$p = \frac{1}{n-1} \tag{2.16}$$

Uma implementação do Modelo Base foi realizada empregando a ferramenta SimEvents, de forma a transformar a unidade básica de processamento em um quadro a ser transmitido por uma interface de rede, em vez de uma tarefa a ser processada por um processador. Esta implementação será descrita no Capítulo 3.

As equações a seguir relacionam as grandezas presentes somente no modelo discreto com as grandezas aplicadas no modelo contínuo:

$$t_{p_i} = \frac{t_q}{r_i} \tag{2.17}$$

$$x_i(t) = t_{p_i} q_i(t) \tag{2.18}$$

Onde t_q é o tamanho do quadro a ser transmitido dado em bits e r_i é a taxa de transmissão da i-ésima interface de rede dada em bits/s.

Sabemos, pela Equação 2.2, que $v_i(t)$ é a taxa de remoção de quadros da i-ésima interface no tempo t, mas o modelo experimental requer que o número de quadros a ser removidos de cada interface no tempo t seja calculado. Este valor é dado pela Equação 2.19.

$$\Delta_q(t) = \frac{v_i(t)}{t_{pi}} p \tag{2.19}$$

A matriz de ganhos K é o único parâmetro cujo cálculo dos valores ainda não foi definido. Em [5] e [6], da Silva propõe uma metodologia para o cálculo desta matriz para o caso do balanceamento de carga em clusters de processadores.

Este mesmo método, cujo desenvolvimento matemático está fora do escopo deste trabalho, foi utilizado para encontrar os valores de K empregados nas simulações do Capítulo 4.

De posse dos recursos matemáticos necessários para a modelagem do problema de balanceamento de carga entre interfaces de rede, foi desenvolvido o modelo apresentado no Capítulo 3.

Capítulo 3

O modelo experimental

Um modelo experimental foi desenvolvido para que o modelo matemático desenvolvido no Capítulo 2 pudesse ser testado.

Este Capítulo irá fazer uma breve introdução sobre o SimEvents, que foi a ferramenta utilizada, mostrando seu funcionamento e blocos básicos. Em seguida, serão apresentados os blocos que foram criados especificamente para este trabalho assim como testes de seu funcionamento. Finalmente, o modelo completo será apresentado e cada uma de suas partes será explorada individualmente.

3.1 SimEvents

O pacote SimEvents torna o toolbox Simulink da ferramenta MATLAB capaz de realizar simulações de eventos discretos.

Simulações de eventos discretos tratam a simulação como uma sequência de eventos que ocorrem em instantes definidos de tempo. Cada evento marca uma possível alteração no estado do sistema, e assume-se que entre dois eventos subsequentes o estado do sistema se mantém estático.

Ao contrário das simulações contínuas, onde as variações do sistema são calculadas para pequenos intervalos de tempo, simulações de eventos discretos podem saltar diretamente para o próximo instante de tempo onde algum evento ocorre, potencialmente reduzindo seu custo computacional.

3.1.1 Entidades

Simulações de eventos discretos são especialmente úteis para representar problemas envolvendo o tráfego de unidades indivisíveis por um sistema, como é o caso deste trabalho que trata de quadros transitando por uma rede de computadores.

Diferentemente do Simulink, que trabalha com sinais contínuos no tempo, o SimEvents tem como unidade básica a entidade (do inglês *"Entity"*). Entidades trafegam por uma rede de blocos que representam filas, servidores, comutadores, etc durante o curso da simulação, e podem ser geradas ou terminadas por blocos específicos.

Cada entidade pode conter um conjunto de informações chamados atributos, que podem ser utilizados pelo modelo para decidir o caminho que a entidade irá seguir em bifurcações na rede, qual será sua prioridade de serviço ou qualquer outra característica de interesse ao problema que se deseja modelar.

Dentre outra infinidade de possibilidades, entidades podem representar pessoas aguardando em uma fila de banco, containers sendo carregados em navios, aviões esperando acesso à uma pista de pouso ou mensagens em uma rede de comunicações.

Os blocos por onde as entidades trafegam pode ler e escrever seus atributos, e cada tipo de bloco fornece saídas com estatísticas sobre seu funcionamento como número de entidades atendidas, número de entidades na fila ou tempo médio de espera, por exemplo. Entretanto, ao contrários dos blocos não existe representação gráfica para as entidades no SimEvents.

Entidades no SimEvents podem ser de três tipos:

Anonymous: tipo mais simples de entidade, que armazena apenas um valor.

Structured: entidades estruturadas definidas diretamente na tela de configuração do bloco gerador. Podem armazenar mais de um atributo e seus atributos são nomeados. Entidades estruturadas são úteis para representar objetos complexos que serão gerados em um único ponto da simulação, já sua estrutura precisa ser replicada em cada bloco gerador.

Bus Objects: entidades estruturadas definidas no painel Bus Editor, cuja definição pode ser reutilizada por toda a simulação. Cada Bus Object cadastrado define um novo tipo de entidade, com nome único e atributos especificados. Atributos podem ser de vários tipos, como números inteiros e de ponto flutuante, valores binários ou

até mesmo outros Bus Objects, o que possibilita a criação de entidades complexas. Como a definição deste tipo de entidade é centralizada em um único ponto do modelo, pode-se garantir que entidades com mesmo nome terão a mesma estrutura. Além disso, uma vez que alguns blocos permitem que um tipo específico de Bus Object seja selecionado para trafegar em suas entradas ou saídas, podemos ter certeza que as entidades que interagem com este bloco terão formato compatível com as operações realizadas.

3.1.2 Eventos

Em simulações de eventos discretos um evento representa a observação de um incidente que pode alterar o estado de uma variável, de uma saída, ou iniciar a ocorrência de outros eventos.

No SimEvents, cada entidade está associada a um valor de prioridade. A prioridade de uma entidade é um número inteiro que será usado como critério de desempate caso duas entidades disparem eventos em um mesmo instante de tempo, sendo valores menores indicativos de uma prioridade maior para a entidade.

A ordem de execução dos eventos segue a seguinte regra:

1. O evento que acontecer mais cedo será executado primeiro.
2. Se duas entidade tem eventos ocorrendo ao mesmo tempo, o evento da entidade com maior prioridade será executado primeiro.
3. Se ambas entidade tiverem a mesma prioridade, a ordem de execução dos eventos não é determinística.

Alguns blocos permitem que seus eventos desencadeiem a execução de códigos definidos pelo usuário, permitindo que atributos das entidades e parâmetros do bloco sejam lidos e modificados, por exemplo.

Esta característica será aproveitada na Seção 3.5.1, onde um bloco Entity Generator será adaptado para gerar múltiplas entidades num mesmo instante.

3.1.3 Blocos

Blocos são usados no SimEvents, assim como no Simulink, para definir o comportamento e efetivamente modelar o problema que se deseja simular.

Um bloco é composto por um conjunto de entradas e saídas. Entradas e saídas podem transmitir entidades ou sinais contínuos, mas nunca os dois simultaneamente. Da mesma forma, saídas de um tipo somente podem ser conectadas à entradas do mesmo tipo, ou seja, entidades e sinais contínuos não são intercambiáveis.

A conexão entre duas portas é identificada por uma seta partindo de uma das saídas de um bloco para uma das entradas de outro. O tipo de sinal que trafega pela conexão é facilmente identificado, já que sinais contínuos são representados por uma linha simples e entidades por uma linha dupla.

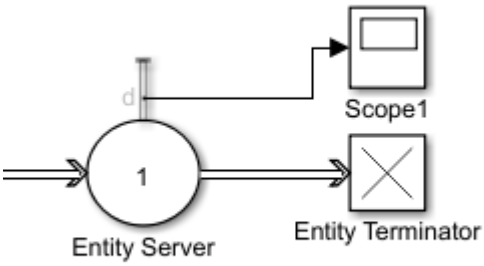

Figura 3.1: Conexões entre blocos no SimEvents.

Apesar das simulações de eventos discretos terem como foco o fluxo de entidades através do sistema, sinais contínuos permanecem sendo úteis como forma de extrair estatísticas sobre o comportamento dos blocos e das próprias entidades que trafegam por eles.

A maioria dos blocos que compõem o SimEvents disponibilizam de forma opcional saídas contínuas com informações como número de entidades servidas até o momento, número de entidades na fila neste instante ou tempo médio de espera. Essas informações podem ser exibidas em gráficos, armazenadas em blocos de armazenamento ou exportadas para uso em outras ferramentas.

É imprescindível entender o funcionamento básico dos principais blocos do SimEvents para compreender o modelo que será apresentado na Seção 3.5.

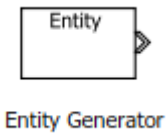

Figura 3.2: Bloco Entity Generator.

O bloco Entity Generator, exibido na Figura 3.2, permite que entidades sejam geradas segundo uma de duas regras possíveis:

1. Sempre que uma entidade for recebida em sua porta de entrada;
2. Periodicamente com período definido manualmente, através de uma porta de entrada ou através de uma função do MATLAB.

O tipo de entidade gerada pode ser selecionado no painel de configuração, e seus atributos podem ser modificados através dos eventos Generate e Exit.

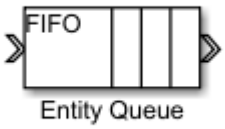

Figura 3.3: Bloco Entity Queue.

O Bloco Entity Queue, exibido na Figura 3.3, recebe Entidades na sua porta de entrada e as repassa para a porta de saída seguindo uma configuração de tipo de serviço (FIFO, LIFO ou prioridade). A saída das entidades é imediata sempre que a porta de saída não estiver bloqueada.

Figura 3.4: Bloco Entity Server.

O Bloco Entity Server, exibido na Figura 3.4, recebe Entidades em sua porta de entrada e as repassa para a porta de saída após um tempo definido em seu painel de

configuração. O tempo de serviço de cada Entidade pode ser fixo ou baseado em algum atributo da Entidade que está sendo servida.

Entity Servers podem ser configurados para servir múltiplas entidades simultaneamente, entretanto esta opção não é utilizada na modelagem de interfaces de rede já que estas transmitem apenas um quadro por vez. Durante o período em que o número máximo de entidades permitida pelo bloco está sendo servida sua porta de entrada fica bloqueada, impossibilitando que qualquer outra entidade seja absorvida pelo bloco.

Figura 3.5: Bloco Entity Terminator.

O bloco Entity Terminator, exibido na Figura 3.5, recebe entidades de qualquer tipo e as elimina instantaneamente. Como sua entrada nunca está bloqueada, ele não interfere na taxa de saída de entidades do bloco anterior. Este bloco provê apenas uma saída contínua que informa a quantidade de entidades absorvidas até o momento, informação que será de grande utilidade na Seção 3.3, onde será caracterizado o comportamento do bloco Queue.

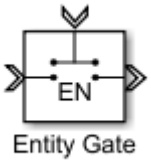

Figura 3.6: Bloco Entity Gate.

O bloco Entity Gate, exibido na Figura 3.6, permite a passagem de entidades entre as portas de entrada e saída de acordo com o tipo de entidades recebidas na sua porta de controle. O bloco opera segundo um de seus três modos:

1. No modo *enabled gate* a passagem de entidades ficará aberta quando uma entidade com valor positivo for recebida na porta de controle.

2. No modo *release gate*, cada entidade recebida na porta de controle liberará apenas uma entidade para atravessar o bloco.

3. No modo *selection gate* somente as entidades com valor igual ao último valor recebido na porta de controle conseguem passar.

Figura 3.7: Bloco Entity Input Switch.

O bloco Entity Input Switch, exibido na Figura 3.7, transforma a saída de dois ou mais blocos em uma única saída. O regime de seleção das portas de entrada pode ser configurado de forma que todas as portas fiquem abertas o tempo todo ou somente uma de cada vez, respeitando alguma regra.

Sua função é fundamental para os casos onde múltiplos blocos produzem entidades que devem ser combinadas e entregues em uma única entrada, sem que as entidades sejam modificadas.

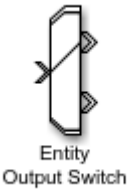

Figura 3.8: Bloco Entity Output Switch.

O bloco Entity Output Switch, exibido na Figura 3.8, atua como inverso do bloco Entity Input Switch, transformando uma única entrada de entidades em duas ou mais saídas.

O regime de saída pode ser configurado para utilizar sempre a primeira porta disponível, seguir o algoritmo Round Robin, usar o valor de entrada de uma porta de

controle, usar algum atributo da entidade como identificador da porta escolhida ou para distribuir as entidades de forma equiprovável entre as portas.

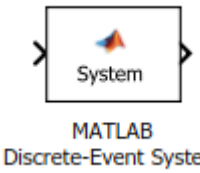

Figura 3.9: Bloco MATLAB Discreve-Event System.

O bloco Matlab discrete event system exige como parâmetro apenas o nome da classe que será usada para definir seu comportamento, sendo que esta classe deve obrigatoriamente implementar a interface matlab.DiscreteEventSystem.

A interface DiscreteEventSystem define todos os métodos que são chamados internamento pelo SimEvents durante a simulação para determinar a entrada e saída de entidades do bloco, o valor de suas saídas contínuas, o estado de suas filas internas e até mesmo o ícone que será exibido no circuito.

Apesar de sua configuração ser trabalhosa e não muito documentada nos manuais do MATLAB, o bloco Matlab discrete event system será utilizado nas seções 3.3 e 3.4 para criar dois blocos fundamentais na a modelagem do problema descrito neste trabalho.

A possibilidade de controlar a fundo componentes da simulação tornou viável a modelagem de partes do sistema que não eram possíveis sem recorrer a soluções paliativas que poderiam prejudicar o resultado final.

3.2 Interface de rede

Uma interface de rede simplex, quando vista de dentro do equipamento, pode ser modelada no SimEvents por um bloco Entity Queue conectado a um bloco Entity Server. O Entity Queue funciona como o buffer da interface de rede, recebendo e armazenando os quadros que aguardam para serem enviados. Assim como no caso real, a fila tem uma capacidade máxima que, quando excedida, impossibilita que novos quadros sejam enfileirados.

O Entity Server, configurado com capacidade igual a um, atua como hardware transmissor, recebendo um quadro por vez e o transmitindo durante o tempo $t_p = t_q/r$.

A saída do Entity Server pode ser conectada a um circuito similar invertido, simbolizando uma interface receptora, ou a um bloco Entity Terminator caso o destino dos quadros não seja relevante.

Considerando, por simplicidade, que todos os quadros tenham um mesmo tamanho t_q, o tempo de espera previsto para um quadro inserido na fila é dado por $x = q \cdot t_p$. Como o bloco Entity Queue informa a quantidade de entidades na fila a cada instante pela sua porta de saída n, o cálculo do valor de x é trivial.

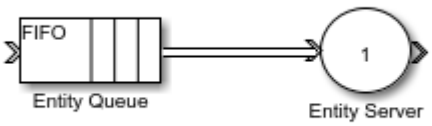

Figura 3.10: Modelo simplificado de fila no SimEvents.

O modelo reproduzido na Figura 3.10 caracteriza de forma simples o comportamento de uma interface de rede unidirecional, porém carece de uma característica essencial para a modelagem do sistema proposto neste trabalho.

Para que o balanceamento de carga seja realizado, é necessário que quantidades definidas de quadros sejam removidas de cada interface de rede para que sejam redistribuídos entre as demais, sem que o funcionamento da interface em questão seja afetado.

O bloco Entity Queue não permite que sua fila interna seja manipulada livremente, ou seja, quadros só podem deixar a fila por sua saída normal, em quantidades definidas pela disponibilidade do Entity Server, que não é suficientemente configurável.

Como nenhum dos blocos disponíveis na biblioteca padrão do SimEvents viabiliza a manipulação de uma fila com o nível de controle exigido, foi necessário desenvolver um bloco diferenciado utilizando o Matlab discrete event system, descrito na Seção 3.3.

3.3 Bloco Queue

Para que o bloco Queue atenda às necessidades do projeto, foi definido que suas atribuições seriam:

1. Receber entidades do tipo *Frame* em sua porta Input;

2. Armazenar as entidades recebidas na porta Input em sua fila interna;

3. Direcionar entidades da fila interna para a porta Output, sempre que esta não estiver bloqueada;

4. Receber entidades do tipo FlushCommand em sua porta Flush Command;

5. Encaminhar a quantidade de entidades definidas no atributo da entidade FlushCommand para a porta Flushed;

6. Informar, a todo instante, a quantidade de entidades presentes na fila interna pela porta de saída *c*;

Enquanto a porta Flush Command está desconectada, o bloco Queue atua de forma idêntica ao bloco Entity Queue, encaminhando as entidades recebidas na porta de entrada para a porta de saída sempre que o bloco seguinte não tiver sua entrada bloqueada. Caso a taxa de entrada seja maior que a taxa de saída, as entidades são armazenadas em uma fila interna e servidas segundo um regime FIFO.

A porta Flush Command recebe entidades do tipo FlushCommand, que contém apenas um atributo numérico inteiro chamado *count*. O valor contido no atributo *count* será usado pelo bloco Queue para identificar a quantidade de entidades que devem ser removidas da fila interna e encaminhadas para a saída Flushed.

Esta diferença, apesar de pequena, faz com o que um sistema com múltiplos blocos Queue possam ser manipulados por um único bloco responsável pelos cálculos de balanceamento, que calcula os valores de *count* e encaminha as entidades FlushCommand para cada fila. As entidades liberadas na porta Flushed podem então ser direcionadas pra outras filas sem que o fluxo normal entre entrada e saída seja comprometido.

Como o bloco Queue não faz parte da biblioteca padrão do SimEvents, uma bateria de testes foi elaborada para garantir que seu funcionamento quando incluído no modelo estará de acordo com as características especificadas, e que sua introdução não enviesará os resultados.

Todos os cenários de teste utilizarão o circuito mostrado na Figura 3.12, que foi desenvolvido para testar o bloco utilizando a menor quantidade de partes móveis possível, a fim de garantir seu isolamento. Os blocos do tipo Scope não interferem de qualquer forma na simulação, e sua presença apenas simplifica a leitura dos dados.

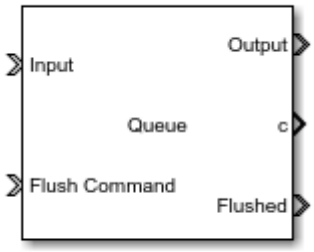

Figura 3.11: Bloco Queue.

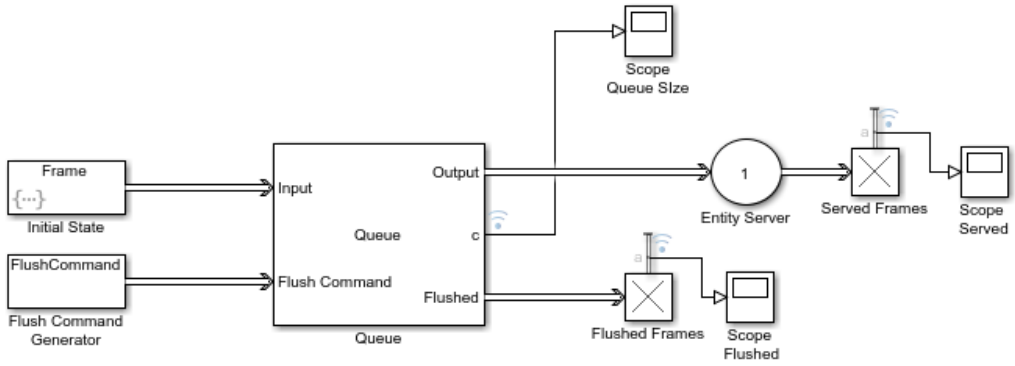

Figura 3.12: Circuito de teste do bloco Queue.

O seguintes parâmetros serão dados como entrada para o sistema de teste, variando apenas seus valores:

- q_0 é o número de entidades geradas no instante inicial da simulação;
- t_p é o tempo total de serviço de cada entidade;
- T_b é o período de geração das entidades do tipo FlushCommand;
- Δq é a quantidade de quadros a ser removido da fila ao receber um FlushCommand.

No instante inicial da simulação, q_o entidades do tipo Frame são geradas pelo bloco Initial State e enviados para a porta Input do bloco Queue.

Todas as entidades que chegam na porta Output do bloco Queue são encaminhadas para o bloco Entity Server, que tem tempo de serviço t_p e capacidade igual a um, ou seja, somente uma entidade pode ser servida a cada instante.

A saída do Entity Server é conectada a um bloco Entity Terminator, que elimina as entidades servidas e fornece como saída o número total de entidades eliminadas. Esta saída é conectada ao bloco Scope Served para ser monitorada.

Entidades do tipo FlushCommand são geradas a cada T_b segundos pelo bloco Flush Command Generator, sempre com atributo *count* igual à Δq. Estas entidades são em seguida encaminhadas para a porta correspondente do bloco Queue, que deve iniciar o processo de Flush.

Por fim, a saída Flushed é conectada ao segundo bloco Entity Terminator, cuja saída será utilizada para monitorar a quantidade de entidades que foram removidas da fila pelo recebimento de entidades FlushCommand.

Os resultados serão exibidos na forma de gráficos com 3 curvas, onde o eixo vertical representa número de quadros e o eixo horizontal representa tempo em segundos.

- A cuva de cor roxa acompanha a quantidade de quadros que saem do bloco Queue pela porta Flushed;
- A curva amarela simboliza o número de entidades existentes dentro da fila;
- A curva azul reflete a quantidade de quadros servidos pelo Entity Server.

Dois cenários de teste foram elaborados para validar o bloco.

3.3.1 Primeiro cenário de teste

q_o	T_b	Δq	t_p
10 quadros	Infinito	Não se aplica	1 segundo

Tabela 3.1: Parâmetros do primeiro cenário de teste.

O primeiro cenário demonstra o comportamento do bloco Queue como fila simples. Como T_b é infinito, nenhuma entidade do tipo FlushCommand será gerada durante o período da simulação e, por isso, o parâmetro Δq é irrelevante.

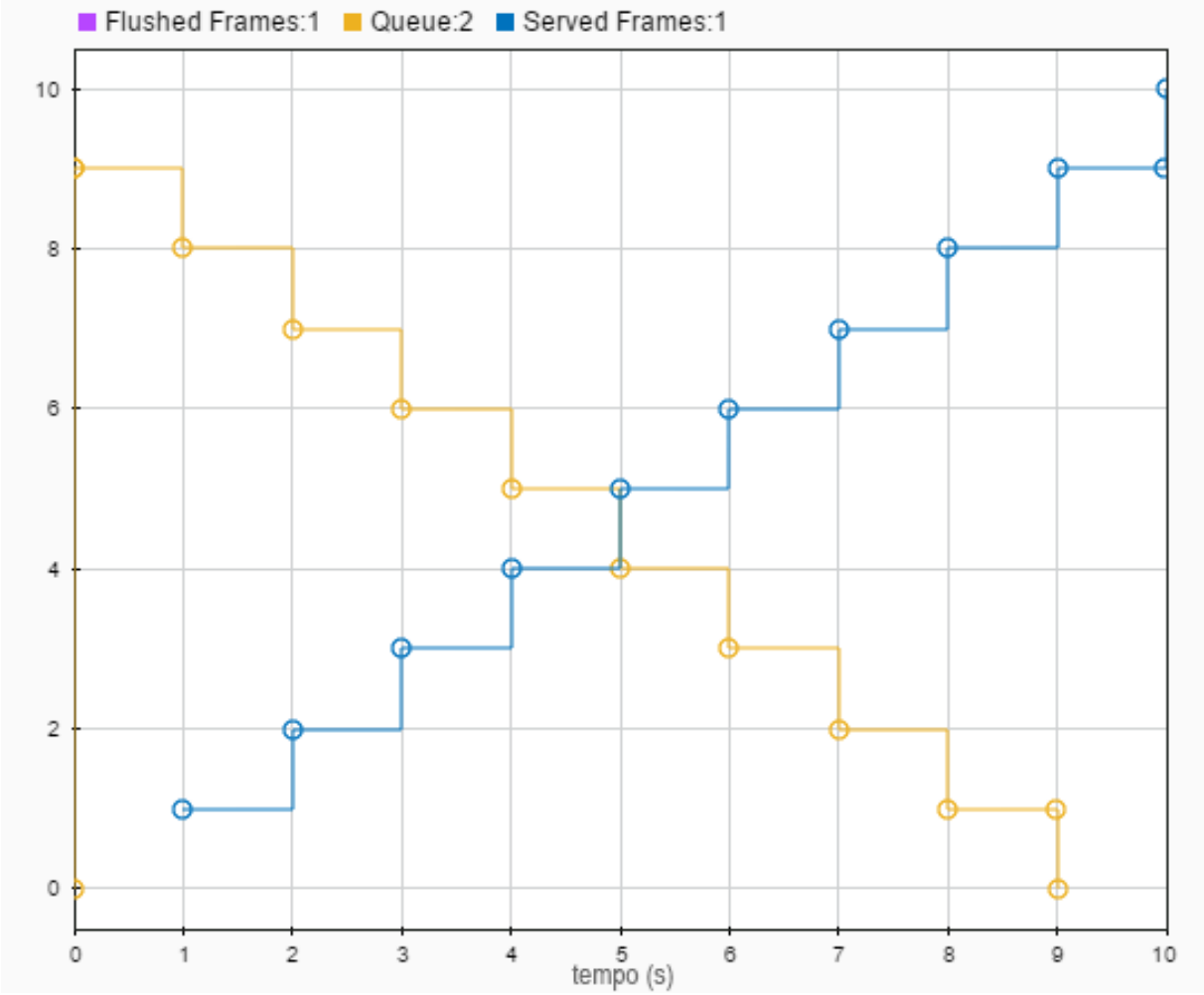

Figura 3.13: Resultados do primeiro cenário de teste do bloco Queue.

No gráfico da Figura 3.13, vemos que no instante inicial 10 entidades são adicionados à fila, mas instantaneamente a primeira é encaminhada para o Entity Server, que termina de servi-la 1 segundo depois.

Assim que a primeira entidade é servida, mais uma é removida da fila e encaminhada para o Entity Server. Esse processo se repete até que a fila se esgota após 9 segundos, e a última entidade termina de ser servida após 10 segundos.

3.3.2 Segundo cenário de teste

q_o	T_b	Δq	t_p
10 quadros	4 segundos	2 quadros	Infinito

Tabela 3.2: Parâmetros do primeiro cenário.

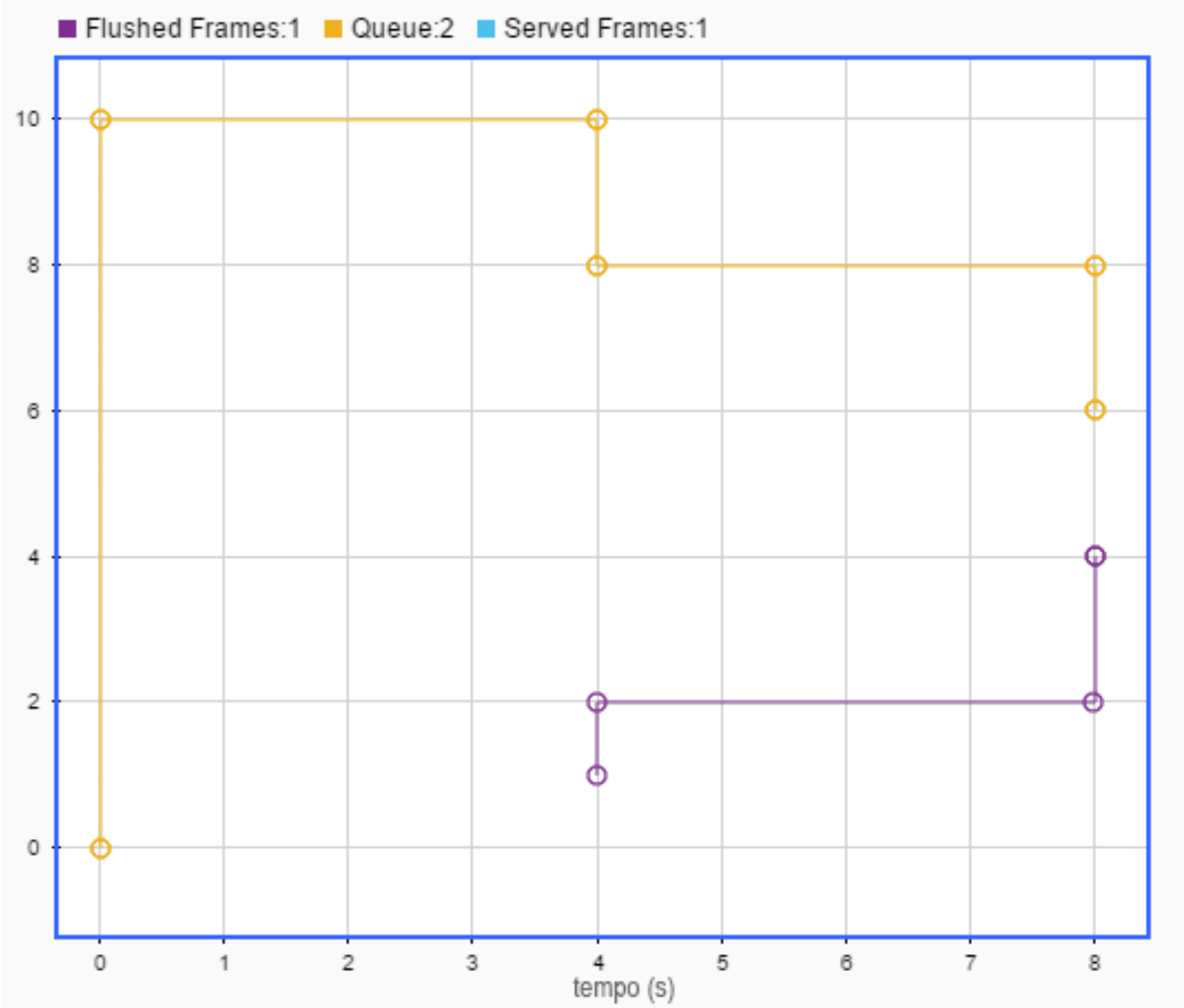

Figura 3.14: Resultados do segundo cenário de teste do bloco Queue.

No segundo cenário de teste o tempo de serviço do Entity Server é infinito, o que faz com que sua porta de entrada fique sempre bloqueada. Desta forma, nenhuma entidade deixa o bloco Queue pela porta Output, e o teste é focado somente na saída de entidades pela porta Flushed.

No gráfico da Figura 3.14, vemos que a cada T_b segundos, Δq entidades são removidas da fila pela saída Flushed. Este teste demonstra a capacidade do bloco Queue de remover uma determinada quantidade de entidades de sua fila interna ao receber entidades do tipo FlushCommand.

Como nenhuma entidade é servida pelo Entity Server, vemos que o número de entidades na fila se mantém constante até o instante em que um FlushCommand é recebido, momento em que seu valor é reduzido de Δq.

3.4 Bloco Load Balancer

O bloco Load Balancer, exibido na Figura 3.15 foi desenvolvido utilizando o bloco Matlab Discrete Event System com o objetivo de ser o controlador principal de balanceamento de carga nas simulações.

O bloco recebe constantemente como entrada os tamanhos das filas das interfaces presentes na simulação pelas entradas q1, q2 e q3. Os valores de q, juntos com os valores dos ganhos K e dos tempos de transmissão t_p são suficientes para que os valores de Δq (quantidade de quadros que deve ser removida da fila de cada interface) sejam calculados.

A entrada BalanceCommand aguarda o recebimento de uma entidade de mesmo nome, que indica o instante onde o balanceamento deve ser executado. Sempre que uma entidade BalanceCommand é recebida, os valores de Δq são calculados e carregados em entidades FlushCommand que são enviadas para as saídas FlushCommand, que são numeradas de acordo com a interface a qual comandam.

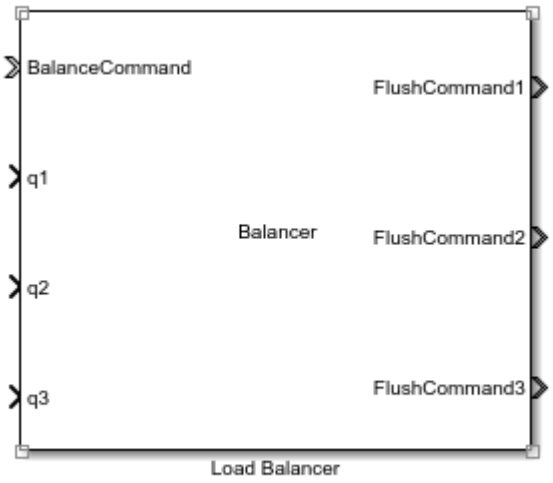

Figura 3.15: Bloco Load Balancer.

Assim como para o bloco Queue, como o bloco Load Balancer não faz parte da biblioteca padrão do SimEvents, foram realizados testes para garantir que seu funcionamento está de acordo com o esperado.

Os testes utilizaram o circuito mostrado na Figura 3.16, e tem como objetivo avaliar apenas o comportamento do bloco quanto a geração de entidades do tipo FlushCommand nos tempos corretos.

Os valores carregados nas entidades FlushCommand, ou seja, a quantidade de quadros Δq que devem ser removidos de cada interface de rede não serão objeto dos testes do bloco Load Balancer, já que a exatidão destes valores será efetivamente colocada à prova nas simulações realizadas no Capítulo 4.

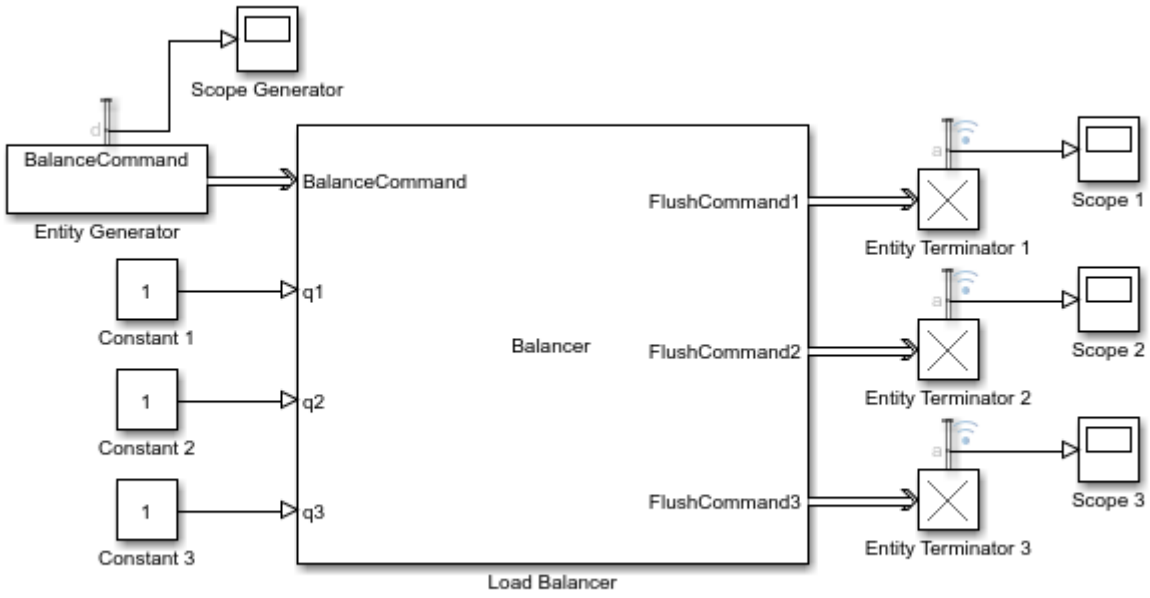

Figura 3.16: Circuito de teste do bloco Load Balancer.

No circuito de teste, o bloco Entity Generator cria uma entidade do tipo BalanceCommand a cada 2 segundos. Os blocos Entity Terminator recebem as entidades do tipo FlushCommand geradas pelo LoadBalancer e tem como saída a quantidade de entidades absorvidas.

Para que o bloco funcione corretamente, os valores lidos em todos os Entity Terminators devem ter um acréscimo de uma unidade a cada 2 segundos. As entradas q1, q2 e q3 recebem valores constantes durante toda a simulação pois são utilizadas apenas para o cálculo dos valores de Δq, que não são avaliados neste teste.

A Figura 3.16 mostra o resultado do teste realizado. Como as 3 curvas tem exatamente o mesmo formato, vemos somente a curva amarela que foi a última desenhada pelo SimEvents. A cada 2 segundos, vemos que a quantidade de entidades geradas aumenta em uma unidade, que é o comportamento esperado.

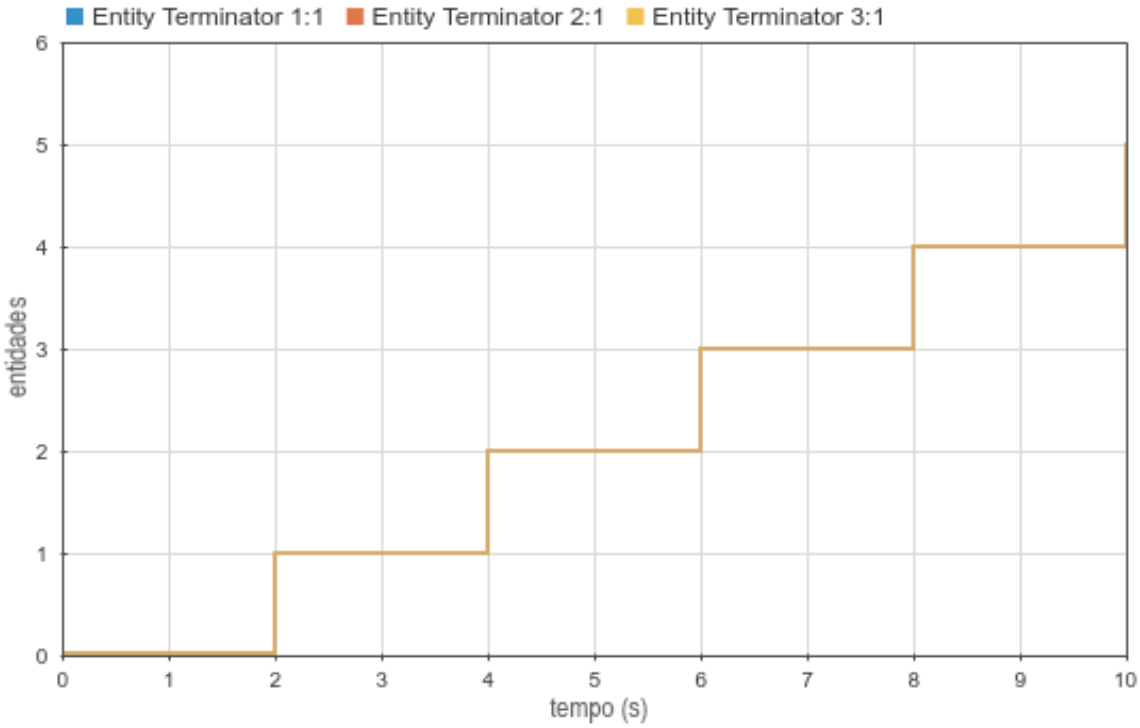

Figura 3.17: Resultado do teste do bloco Load Balancer.

3.5 Circuito completo

A Figura 3.18 mostra o modelo utilizado para simular todas as situações listadas no Capítulo 4, agrupando cada parte conforme sua função em blocos de mesma cor.

Este modelo é capaz de representar três interfaces de rede com suas filas e taxas de transmissão distintas. Quadros podem ser adicionados a cada interface no instante inicial da simulação ou periodicamente após seu início. O modelo também permite que seja configurado se haverá ou não transmissão de quadros pelas interfaces.

A frequência de execução do balanceamento pode ser configurada e os valores de $q(t)$ (quantidade de quadros na fila da interface) são enviados constantemente para o bloco balanceador. Além disso, cada interface tem seu tempo de atraso de transferência de quadros para outras filas configurável.

Cada componente será apresentado com maiores detalhes nas seções a seguir.

3.5.1 Geradores de quadros

Os componentes geradores de quadros são responsáveis por criar o estado inicial da simulação, gerando as quantidades configuradas de quadros e transferindo-os para as filas das

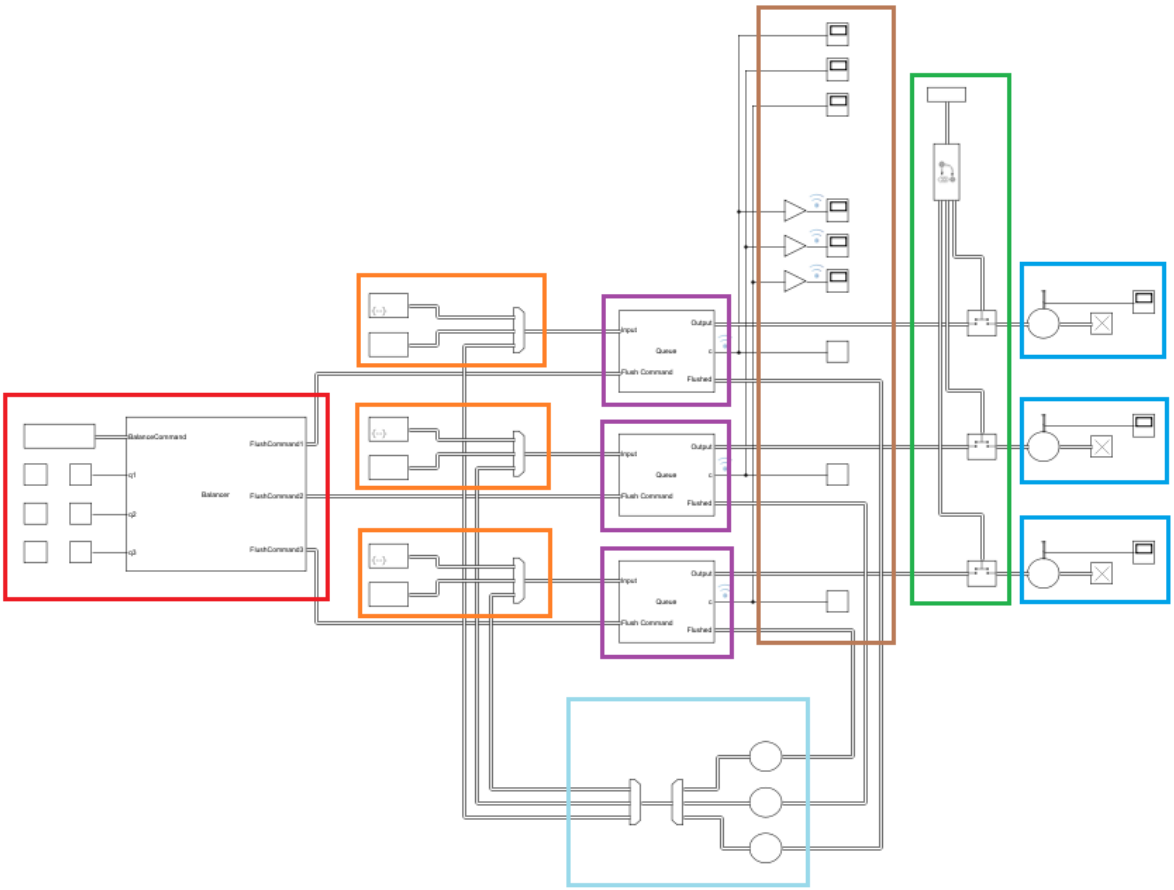

Figura 3.18: Circuito completo utilizado nas simulações.

interfaces. Além disso, caso seja de interesse da cenário a ser simulado, este componente pode inserir periodicamente novas entidades no sistema após o início da simulação. Um exemplo deste componente pode ser visto na Figura 3.19.

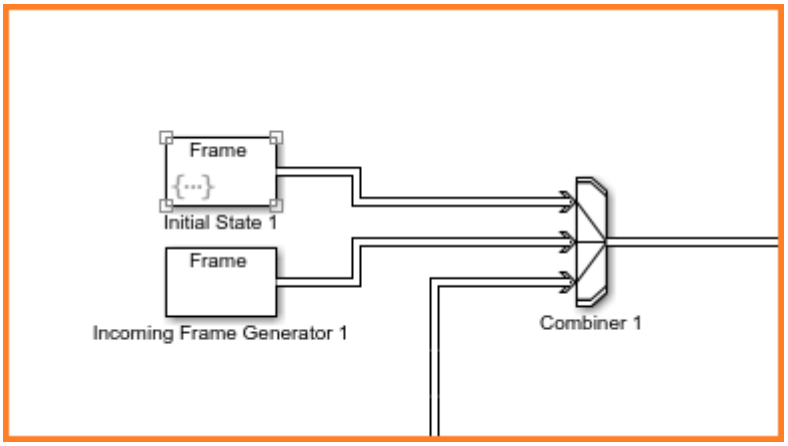

Figura 3.19: Componente gerador de quadros.

O bloco *Incoming Frame Generator* é responsável por gerar quadros periodicamente após o início da simulação. O período de geração dos quadros é determinado pela variável T_g, que pode ser definida como infinito caso a função do bloco não seja desejada.

O bloco *Initial State* gera uma quantidade definida de quadros no instante inicial da simulação, e fica inativo até que ela seja concluída.

Como este comportamento não se enquadra em nenhum dos comportamentos padronizados do bloco, foi necessário adapta-lo usando uma função do MATLAB como origem dos tempos de geração das entidades. A função utilizada pode ser vista na Figura 3.20.

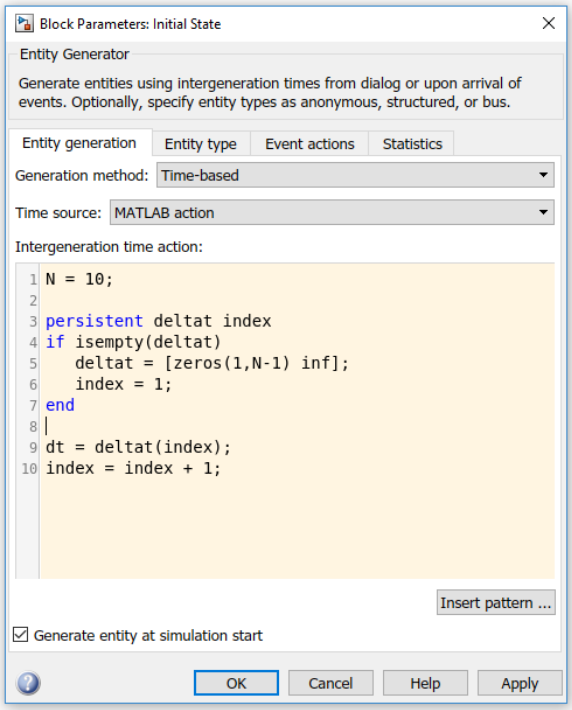

Figura 3.20: Função utilizada para definir o comportamento do bloco *Initial State*.

O parâmetro *Intergeneration time action* recebe uma função que retorna o tempo que deve ser aguardado entre a criação da entidade atual e a criação da próxima entidade.

A variável N representa a quantidade total de entidades que devem ser geradas. O tempo retornado será igual a 0 para as primeiras N chamadas da função, indicando que a próxima entidade deve ser gerada imediatamente nestes casos. A chamada $N + 1$ retorna um tempo infinito, orientando o bloco a não gerar mais nenhuma entidade até o fim da simulação.

O bloco *Combiner* é um Input Switch responsável por agregar as entidades vindas

dos dois blocos geradores, bem como aquelas vindas da saída Flushed do bloco Queue. Estas entidades em seguida são encaminhadas para a entrada do bloco Queue, apresentado na próxima seção.

Sem o auxílio do bloco Combiner, o bloco Queue precisaria ser modelado para suportar três entradas distintas, o que aumentaria sua complexidade.

3.5.2 Filas

Cada bloco Queue representa a fila de uma das três interfaces de rede do modelo. A porta Input recebe os quadros enviados pelo componente gerador de quadros exposto na Seção 3.5.1, e os armazena em sua fila interna.

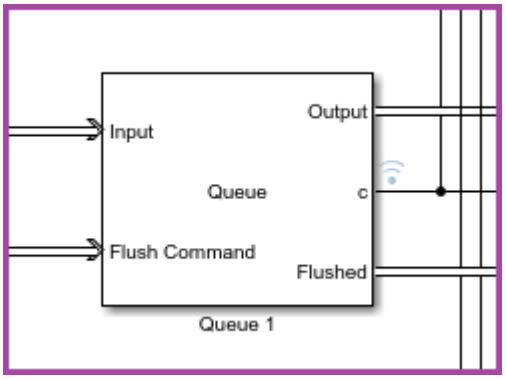

Figura 3.21: Fila de uma das interfaces de rede.

Um quadro é encaminhado para a saída Output sempre que esta saída estiver disponível e a fila não estiver vazia. Ademais, quando a porta FlushCommand recebe uma entidade, o número especificado de quadros é encaminhado para a porta Flushed, conforme comportamento descrito na Seção 3.3.

Este componente fornece como saída contínua a quantidade de quadros presentes em sua fila interna a cada instante. Os valores de $q(t)$ são então encaminhados para as entradas do bloco balanceador, que depende destes dados para realizar os cálculos de balanceamento.

3.5.3 Componente controlador de saída

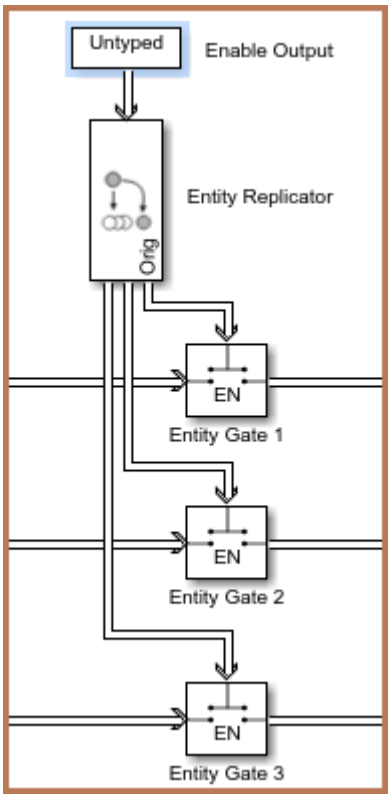

Figura 3.22: Componente controlador de saída.

O componente controlador de saída controla a passagem de entidades entre a saída Output dos blocos Queue e a entrada dos componentes transmissores, que serão apresentados na Seção 3.5.4. A passagem de entidades por este componente simboliza que as interfaces estão transmitindo quadros, característica que será fundamental para as simulações do Capítulo 4.

O bloco *Enable Output* gera uma entidade anônima com valor baseado em uma variável definida no ambiente do MATLAB. Esta entidade é replicada pelo bloco Entity Replicator e encaminhada para cada bloco Entity Gate, que irá liberar a passagem de entidades caso o valor recebido seja positivo, ou bloquear a passagem caso contrário.

3.5.4 Componente transmissor

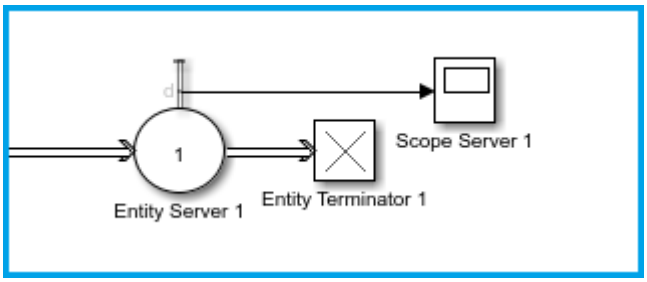

Figura 3.23: Componente transmissor.

O componente transmissor simula a transmissão de quadros pela interface de rede.

Todo quadro recebido pelo bloco Entity Server fica retido nele por um tempo igual ao tempo de transmissão t_p da sua interface correspondente, e nenhum outro quadro pode ser recebido até que este tempo seja esgotado.

Como não há interesse em reutilizar os quadros já transmitidos, depois de passar pelo bloco Entity Server todas as entidades são encaminhadas para o bloco Entity Terminator, que as destrói.

3.5.5 Componente realimentador

A função do componente realimentador é receber os quadros que saem das filas devido ao balanceamento e redistribui-los entre as interfaces de rede com um atraso igual ao valor da variável h.

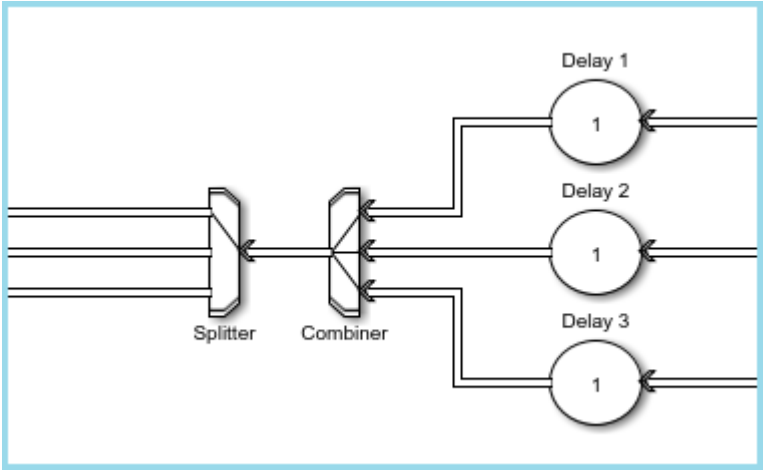

Figura 3.24: Componente realimentador.

Todos os quadros que chegam no componente passam pelos blocos *Delay*, que são Entity Servers configurados para servir apenas uma entidade de cada vez por um tempo h. Desta forma, mesmo que os blocos Queue disponibilizem vários quadros em um mesmo instante de tempo, eles precisarão passar um a um pelo atrasador, simulando o atraso de transferência entre as filas.

Em seguida, as entidades são agrupadas pelo bloco *Combiner* e distribuídas pelo bloco *Splitter* entre as suas três saídas de forma equiprovável. Cada uma destas saídas é então encaminhada para uma das filas do modelo.

3.5.6 Coletores de dados

O componente coletor de dados é responsável pela leitura e distribuição dos valores de $q(t)$ disponibilizados pelas filas.

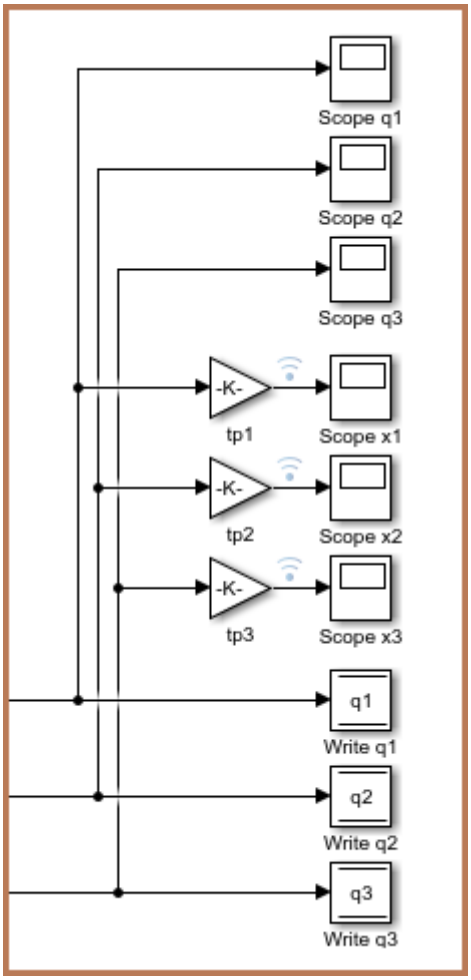

Figura 3.25: Coletores de dados.

Os blocos *Write* escrevem os valores recebidos em memórias internas que podem ser lidas em qualquer ponto do modelo. Como consequência, não é necessário que exista uma linha conectando diretamente as saídas dos blocos Queue com as entradas correspondentes do bloco balanceador, o que torna o circuito mais limpo.

Os bloco *Scope q1*, *Scope q2* e *Scope q3* leem diretamente os valores de $q(t)$ e os utilizam na geração de gráficos.

Já os blocos *Scope x1*, *Scope x2* e *Scope x3* tem suas entradas multiplicadas pelos valores de t_{p_i}, transformando-os nos valores de $x_i(t)$ conforme equação 2.18. A leitura destes dados permite que gráficos de tempo de espera sejam gerados.

3.5.7 Balanceador

O componente balanceador atua como controlador principal da simulação, calculando os valores de Δq baseado nos tamanhos das filas das interfaces.

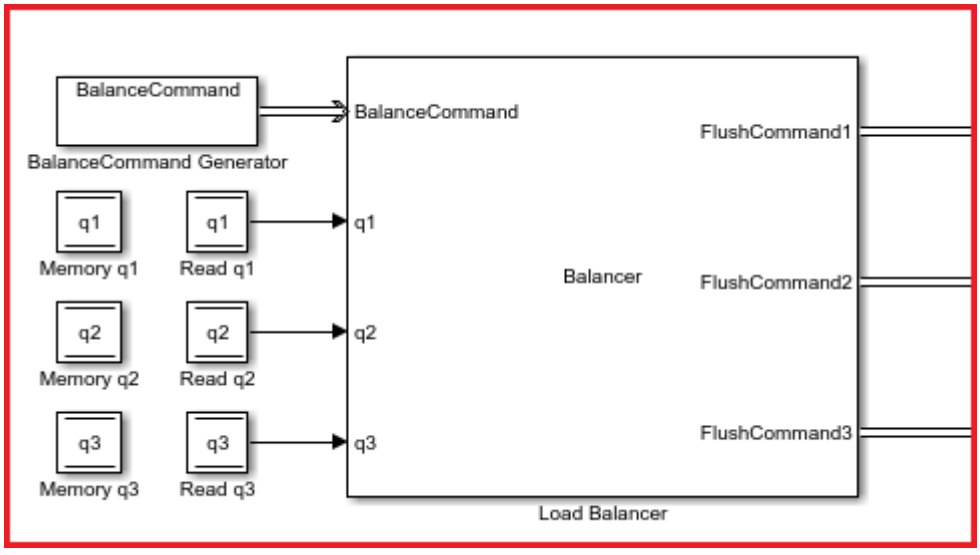

Figura 3.26: Balanceador.

O bloco *BalanceCommand Generator* gera uma entidade do tipo BalanceCommand a cada T_b segundos, que indica para o Load Balancer que o balanceamento deve ser iniciado.

O bloco Load Balancer lê os valores de $q(t)$ gravados nas memórias pelo componente coletor de dados e os utiliza para descobrir os valores apropriados de Δq. Os valores calculados para Δq são então carregados em entidades do tipo FlushCommand, que em sequência são encaminhados para os blocos Queue, iniciando o processo de balanceamento.

Caso não seja desejado que o balanceador atue na simulação, o valor do período T_b pode ser configurado como infinito para que nenhuma entidade do tipo BalanceCommand seja gerada, efetivamente desligando o balanceamento de carga.

Capítulo 4

Balanceamento & Simulações

Primeiramente neste capítulo será estabelecida a metodologia utilizada nas simulações. A Seção 4.1 descreve detalhadamente todo o processo seguido na realização dos diversos cenários de teste. Em seguida, algumas limitações encontradas ao aplicar o modelo experimental desenvolvido no Capítulo 3 serão exploradas. Finalmente, os resultados das simulações serão exibidos e analisados.

4.1 Metodologia

Primeiramente, foi definido que todos os quadros utilizados na simulação teriam o mesmo tamanho t_q. Esta simplificação reduz um pouco a complexidade do modelo, já que o tempo de transmissão dos quadros dado pela equação $t_{p_i} = t_q/r_i$ pode ser calculado uma única vez para cada interface.

O tamanho escolhido para os quadros foi de 1500 bytes, que é um valor realista para redes de computadores já que este é o MTU do padrão Ethernet.

O estado inicial do modelo é definido em sua totalidade pela quantidade de quadros armazenados na fila de cada interface. Para todas as simulações será utilizado o mesmo vetor de tempos de espera inicial $x_0 = [0,2; 0,4; 0,2]$, que pode ser traduzido no vetor quantidade de quadros no instante inicial q_0 pela equação 2.18.

A Tabela 4.1 resume as configurações de interfaces que serão trabalhadas.

Configuração	Interface 1	Interface 2	Interface 3
1	10 Mbps	10 Mbps	10 Mbps
2	32 Mbps	16 Mbps	10 Mbps
3	8 Mbps	16 Mbps	24 Mbps

Tabela 4.1: Resumo das simulações realizadas.

Três configurações de interfaces de rede foram selecionadas, sendo uma com interfaces homogêneas e duas com interfaces heterogêneas. Para cada configuração serão desenvolvidos três cenários diferentes, que adicionam progressivamente variáveis ao modelo.

No primeiro cenário as filas das interfaces serão preenchidas com sua quantidade inicial de quadros q_0, e não haverá entrada ou saída de quadros do sistema. Desta forma, a simulação tratará da situação mais simples, onde não há chegada de novos quadros para serem transmitidos, e também não é feita a transmissão de qualquer quadro.

O balanceamento será executado periodicamente com perído $T_b = 0,05s$, o que fará com o que os quadros sejam redistribuídos entre as filas das interfaces. Nesta situação espera-se encontrar, após algumas iterações do balanceamento, um tempo de espera $x(t)$ muito próximo entre as três interfaces.

Como neste cenário nenhum quadro deixa o sistema, as filas das interfaces estarão cheias no término da simulação.

No segundo cenário as filas também serão preenchidas com sua quantidade inicial de quadros e a entrada de novos quadros estará desativada, porém neste cenário a interface estará transmitindo quadros durante a simulação. Como a quantidade de quadros no sistema estará constantemente sendo reduzida, eventualmente todas as filas se esvaziarão.

O balanceamento será executando periodicamente como no primeiro cenário, mas a quantidade de quadros nas filas irá variar entre execuções subsequentes. É esperado que, após algumas iterações do balanceamento, os quadros estejam distribuídos entre as filas de forma que elas se esgotarão simultaneamente, mesmo que suas taxas de transmissão sejam diferentes.

O terceiro cenário é muito similar ao segundo, mas acrescenta a entrada de quadros no sistema. Cada interface receberá quadros novos com um período constante, e transmitirá os quadros segundo sua taxa de transmissão r_i.

O período de geração de novos quadros $T_{in} = 0,003s$ foi escolhido de forma que a taxa de saída total do sistema seja sempre maior que a taxa total de entrada. Caso contrário, o sistema teria um crescimento constante no número de quadros e as filas nunca se esgotariam.

Além disso, o crescimento desenfreado no número de entidades no sistema aumentaria o risco de travamento da simulação por exceder o número máximo de eventos simultâneos permitidos pelo SimEvents, problema descrito na Seção 4.2.

4.2 Problemas e limitações

A ferramenta SimEvents, apesar de muito robusta, impõe algumas restrições quanto aos modelos que podem ser produzidos.

Caso a simulação produza eventos muito rapidamente, é possível que eventos que ocorreram em instantes diferentes sejam processados como se tivessem acontecido ao mesmo tempo, ou até mesmo que parte dos eventos seja totalmente ignorada.

Este problema se verificou ao tentar representar interfaces com taxas de transmissão muito mais altas do que as exploradas neste trabalho, que ocasionava comportamentos imprevisíveis e até mesmo travamento da ferramenta.

Outra limitação do SimEvents se dá quanto ao número de eventos permitidos em um mesmo instante de tempo. Como pode ser visto em [10], o limite de 5.000 eventos simultâneos por bloco ou 100.000 eventos simultâneos para toda a simulação não pode ser excedido.

Esta restrição, criada para evitar travamento da ferramenta no caso da criação de um *loop* acidental pelo usuário, impede que o modelo proposto no Capítulo 3 seja utilizado para simular situações envolvendo grandes quantidades de quadros.

Cada componente gerador de quadros (seção 3.5.1) dispara, já no primeiro instante da simulação, um número de eventos igual ao número total de quadros que serão usados como estado inicial das interfaces. O componente fila (descrito na Seção 3.5.2) também gera uma grande quantidade de eventos simultâneos quando transfere entidades devido ao balanceamento.

Por causa deste limite, todos os cenários propostos nas seções subsequentes foram elaborados de forma que a quantidade total de entidades no sistema nunca excedesse

5.000.

Pelas equações 2.18 e 2.17, temos que $q_0 = x_0 r / t_q$. Mantendo o tamanho dos quadros t_q constante, é preciso equilibrar os valores de x_0 e r para que o modelo funcione. Para simular interfaces mais rápidas, o tempo de espera inicial deve ser reduzido de forma a não produzir um valor muito alto de q_0. Da mesma forma, caso o tempo de espera inicial x_0 seja muito alto, a taxa de transferência da interface deve ser reduzida para que o número de entidades não exceda os limites do sistema.

Outra limitação do modelo proposto é que ele foi desenvolvido desde o início para suportar exatamente três interfaces de rede.

Adaptar o modelo para trabalhar com somente duas interfaces seria bastante simples, porém estende-lo para um número maior de interfaces seria um processo trabalhoso e propenso a erros.

O Simulink permite que blocos sejam agrupados em módulos reutilizáveis, melhorando a organização de circuitos complexos e facilitando a replicação de componentes. No entanto, esta segmentação não foi realizada no modelo apresentado neste trabalho por uma questão de simplicidade, já que foi previsto desde o início que somente três interfaces de rede seriam simuladas.

4.3 Interfaces Homogêneas

A primeira configuração irá testar a situação onde todas as interfaces tem a mesma taxa de transmissão r_i. A taxa de 10 Megabits por segundo foi escolhida de forma a respeitar os limites expostos na Seção 4.2, e tem a vantagem de ser uma velocidade amplamente disponível para enlaces comerciais de internet.

Sabemos pela equação 2.18 que t_p depende somente de t_q e de r. Como t_q foi assumido constante e r é igual para as três interfaces, temos que os três valores de t_p também serão iguais.

Entretanto, como explicado na Seção 4.1, valores distintos de x_0 serão atribuídos à cada interface. A equação 2.18 deixa claro que esta diferença irá produzir, no caso homogêneo, valores de q_0 necessariamente diferentes.

A Tabela 4.2 traz um resumo dos parâmetros usados como entrada para a simulação.

	Interface 1	Interface 2	Interface 2
r	$10 \cdot 10^6$ Mb/s	$10 \cdot 10^6$ Mb/s	$10 \cdot 10^6$ Mb/s
t_p	$12 \cdot 10^{-4}$ s	$12 \cdot 10^{-4}$ s	$12 \cdot 10^{-4}$ s
x_0	0,6 s	0,4 s	0,2 s
q_0	500 quadros	333 quadros	167 quadros
K	2.587	2.584	2.262

Tabela 4.2: Parâmetros da primeira configuração.

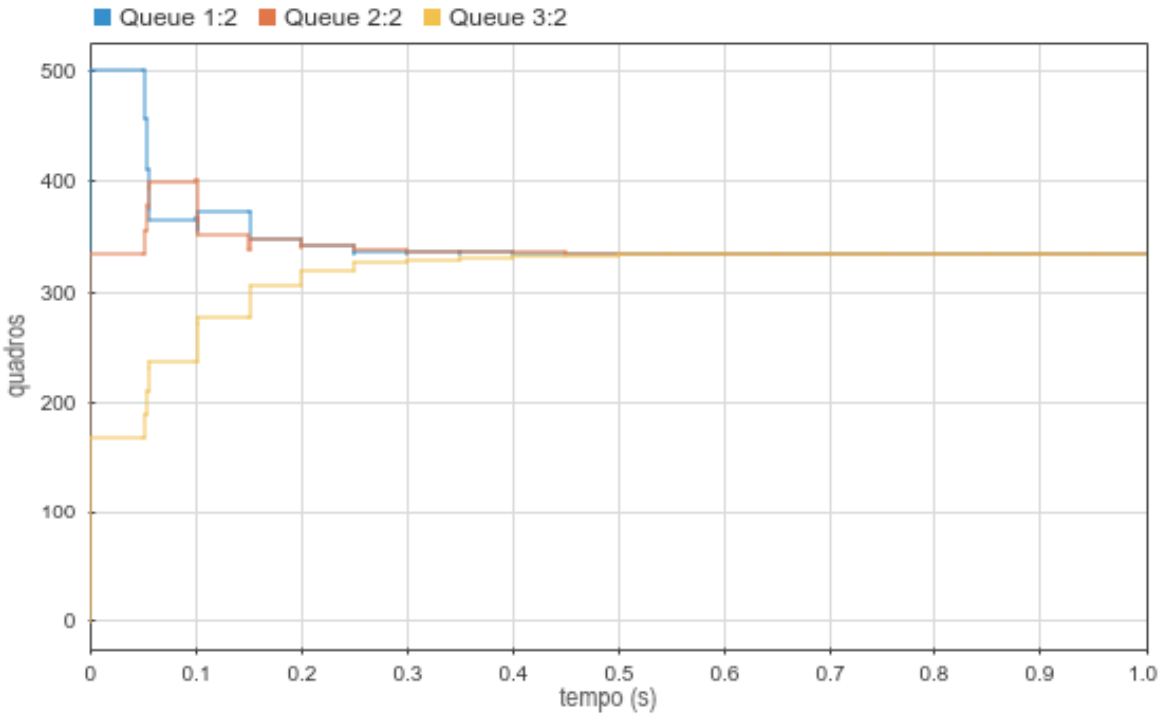

Figura 4.1: Gráfico $q_i(t)$ para o 1º cenário da 1ª configuração.

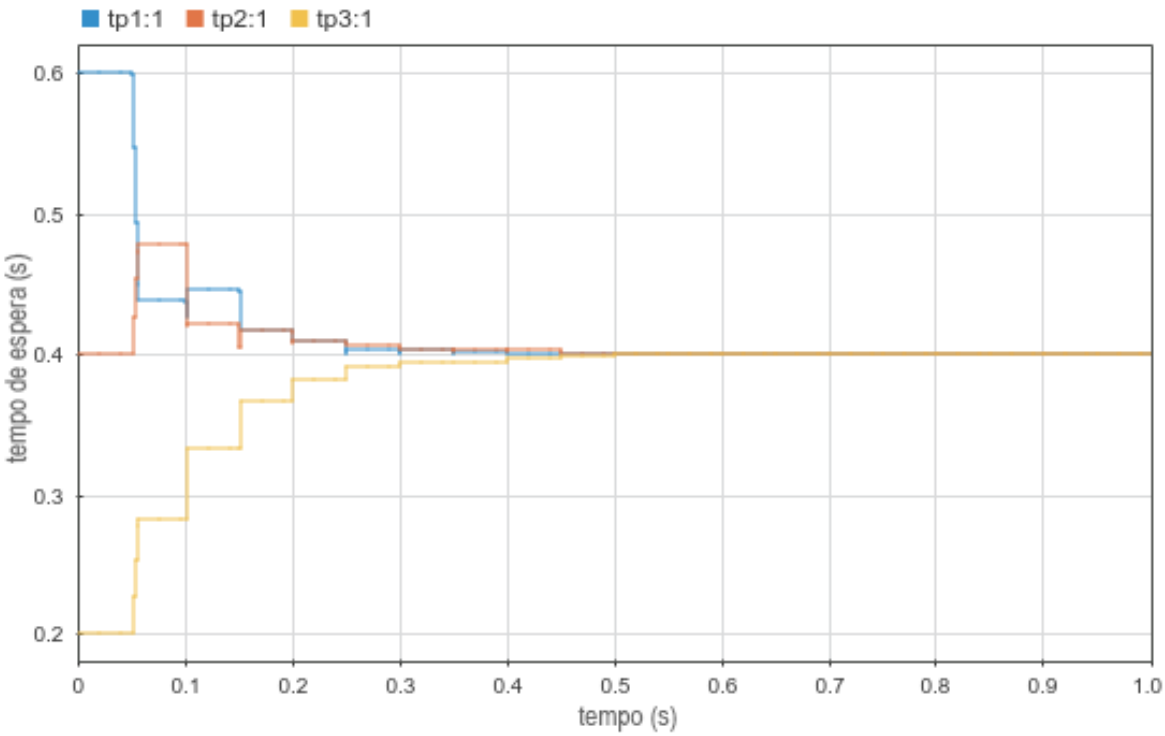

Figura 4.2: Gráfico $x_i(t)$ para o 1º cenário da 1ª configuração.

A Figura 4.1 mostra a variação da quantidade de quadros nas filas de cada interface pelo tempo, que são os valores brutos da saída c do bloco Queue. Já a Figura 4.2 demonstra a variação do tempo de espera de cada interface pelo tempo, que é igual à quantidade de quadros existentes na fila multiplicado pelo tempo de transmissão de um quadro naquela interface.

Como nesta configuração as três interfaces tem mesma taxa de transmissão r e todos os quadros tem o mesmo tamanho $t_q = 1500$ bytes, é evidente que o sistema somente estará balanceado caso todas as interfaces tenham suas filas com aproximadamente o mesmo tamanho.

Para os gráficos das Figuras 4.1 e 4.2, assim como para todos os outros presentes até o final deste capítulo, as interfaces de rede podem ser identificadas pela cor do traço. A linha azul irá sempre representar a interface de número um, a linha laranja a interface dois e a linha amarela e terceira interface.

O primeiro instante dos dois gráficos é conveniente para verificar que os parâmetros de estado inicial da simulação estão sendo respeitados.

Na Figura 4.1 vemos que as filas iniciam com quantidades de quadros iguais aos valores do vetor q_0, enquanto a Figura 4.2 demonstra o mesmo para o vetor de tempos de

espera x_0.

Sabemos que todas as simulações obedecem um mesmo período de balanceamento $T_b = 0,05s$. Como neste cenário a única fonte de movimentação de quadros no sistema é a transferência de quadros entre interfaces devido ao balanceamento de carga, fica evidente em ambos os gráficos que T_b está sendo respeitado já que todas as transições de estado se iniciam em tempos múltiplos de $0,05$.

Sempre que o balanceamento é executado o bloco balanceador calcula a quantidade de quadros que deve ser removida de cada interface, e encaminha para as filas uma entidade do tipo FlushCommand contendo este valor.

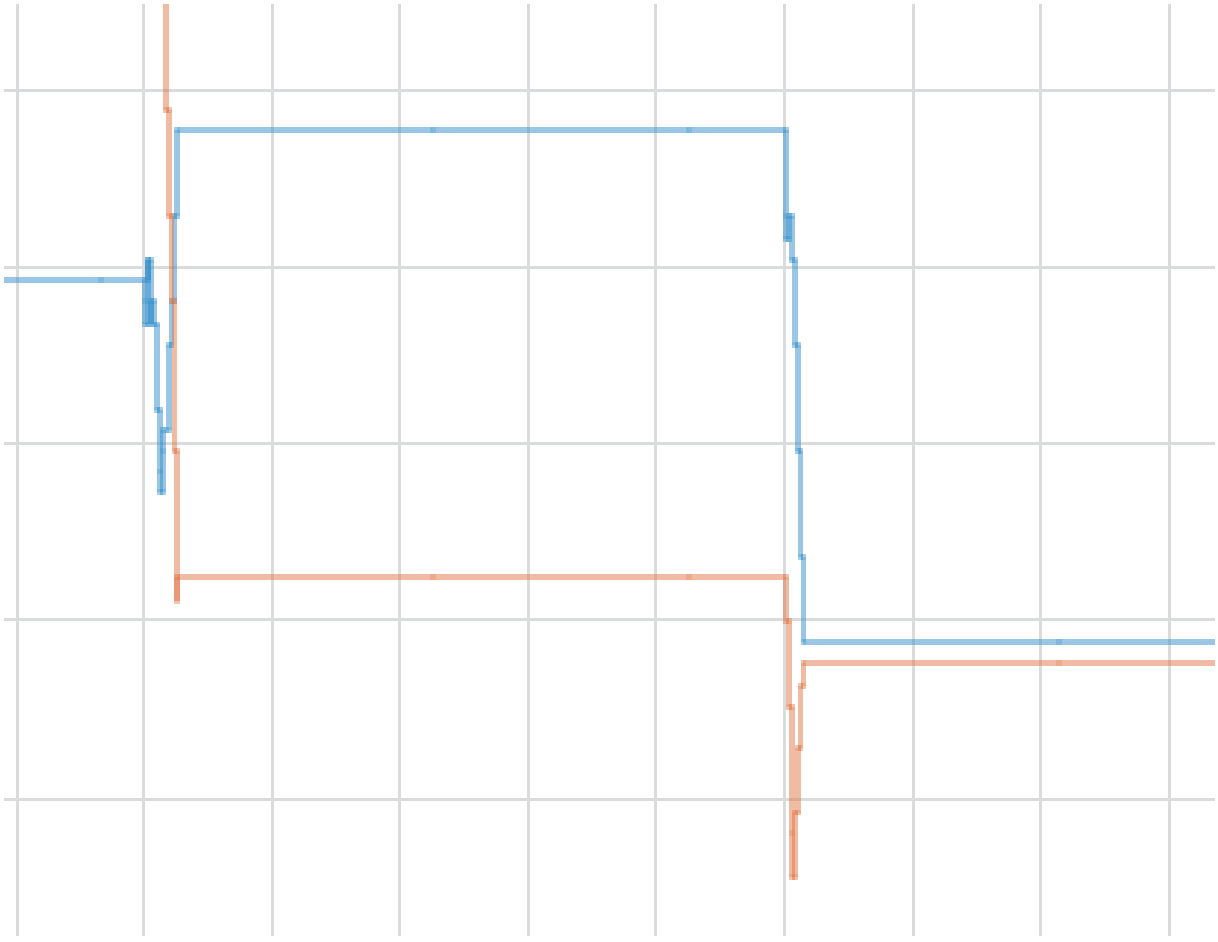

Figura 4.3: Ampliação de parte do gráfico $x_i(t)$ para o 1º cenário da 1ª configuração.

Como cada quadro leva um tempo $h = 3*10^{-5}$ para ser transferido de uma fila para a outra, vemos em todos os gráficos que existe um pequeno período após cada iteração do balanceamento até que as filas se estabilizem. Este fenômeno pode ser observado com mais detalhes na Figura 4.3, que é uma ampliação de parte da Figura 4.1.

O tempo total que o sistema levou para ser balanceado pode ser estimado com facilidade observando as curvas de tempo de espera, neste caso a Figura 4.2. Quanto mais próximas estiverem as curvas de tempo de espera, mais próximo o sistema se encontrará do estado perfeitamente balanceado.

Entretanto, é razoável intuir que, como as taxas de transmissão são iguais e o tamanho dos quadros é fixo, as filas precisam ter a mesma quantidade de quadros para que estejam balanceadas, o que é confirmado pela Figura 4.1. Desta forma, excepcionalmente para o caso homogêneo, ambos os gráficos são igualmente úteis para estimar o tempo de balanceamento.

Os gráficos 4.1 e 4.2 mostram que após 0,3 segundos o sistema já está bem próximo do ponto ótimo, aquele onde os tempos de espera são exatamente iguais. Após 0,45 este ponto é atingido, e as interfaces estão perfeitamente balanceadas. Isto significa que, caso as interfaces passassem a transmitir os quadros de suas filas a partir do 0,45 segundos, todas as filas se esgotariam simultaneamente.

4.3.1 Segundo cenário

Neste cenário as três interfaces estarão transmitindo um quadro a cada t_p segundos. A transmissão de um quadro significa que ele será removido da fila e permanentemente eliminado do sistema, contrariamente à remoção de quadros das filas para balanceamento, já que estes são realimentados no sistema.

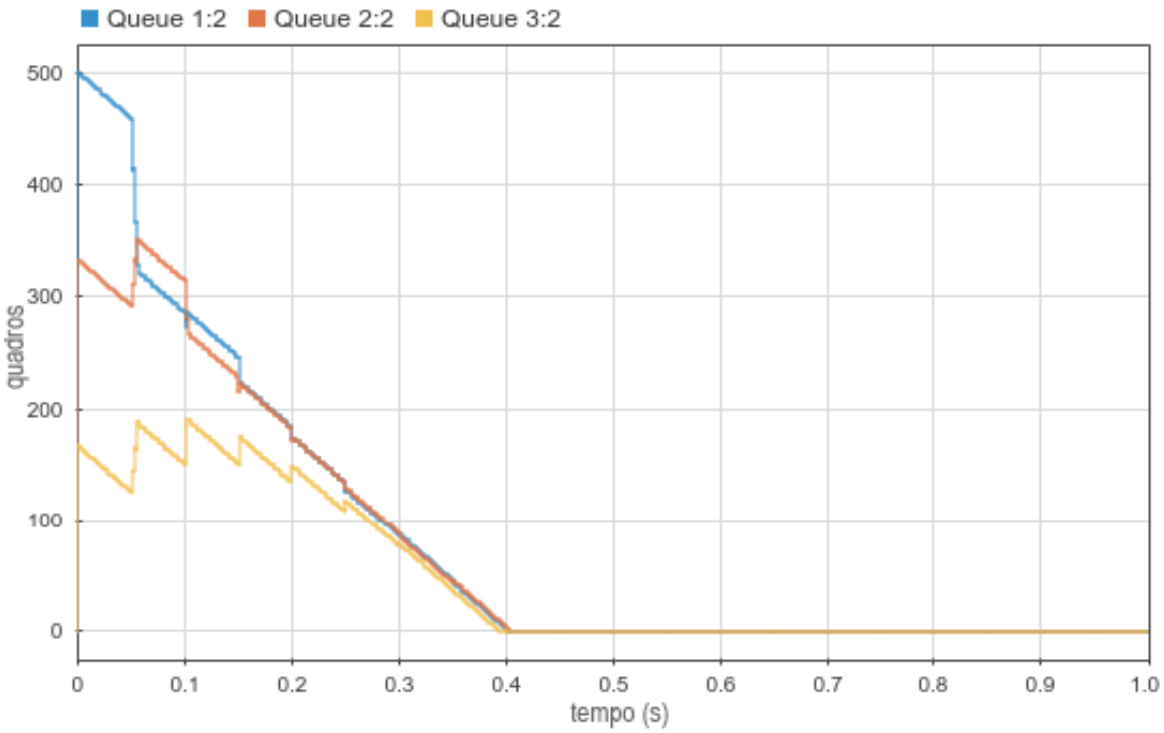

Figura 4.4: Gráfico $q_i(t)$ para o 2º cenário da 1ª configuração.

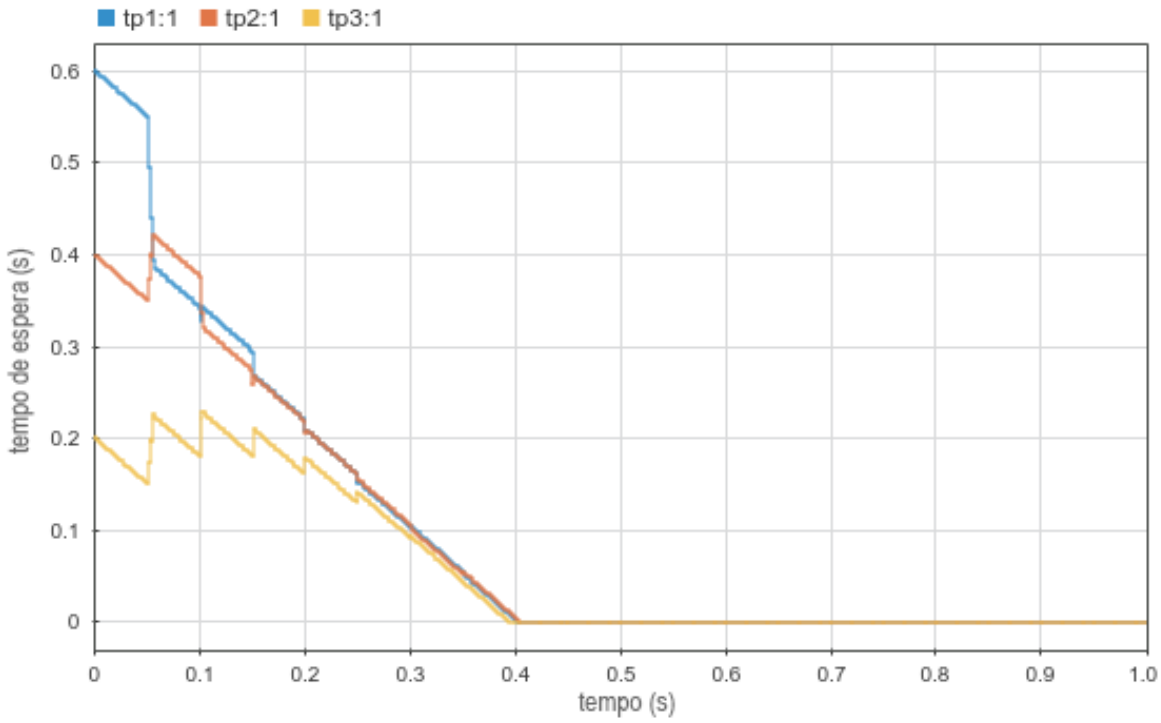

Figura 4.5: Gráfico $x_i(t)$ para o 2 cenário da 1ª configuração.

A primeira diferença que podemos observar entre as Figuras 4.4 e 4.5 quando

comparadas com as figuras do primeiro cenário é que, neste caso, a quantidade de quadros nas filas varia mesmo quando não há transferência de quadros devido ao balanceamento.

Como na configuração atual as três interfaces tem taxa de transmissão de quadros r iguais, as inclinações das três curvas nas Figuras 4.4 e 4.5 também são iguais, exceto quando o processo de balanceamento está em curso e os quadros estão sendo transferidos de uma fila para outra. Também devido ao fato das interfaces serem homogêneas, as curvas de $q(t)$ e $x(t)$ tem o mesmo formato.

Assim como no primeiro cenário, ambos os gráficos podem ser utilizados para medir o tempo total de balanceamento. Na Figura 4.5, vemos que a partir de 0,3 segundos as interfaces ficam balanceadas, fazendo com que as três filas se esgotem com menos de 0,01 segundos de diferença.

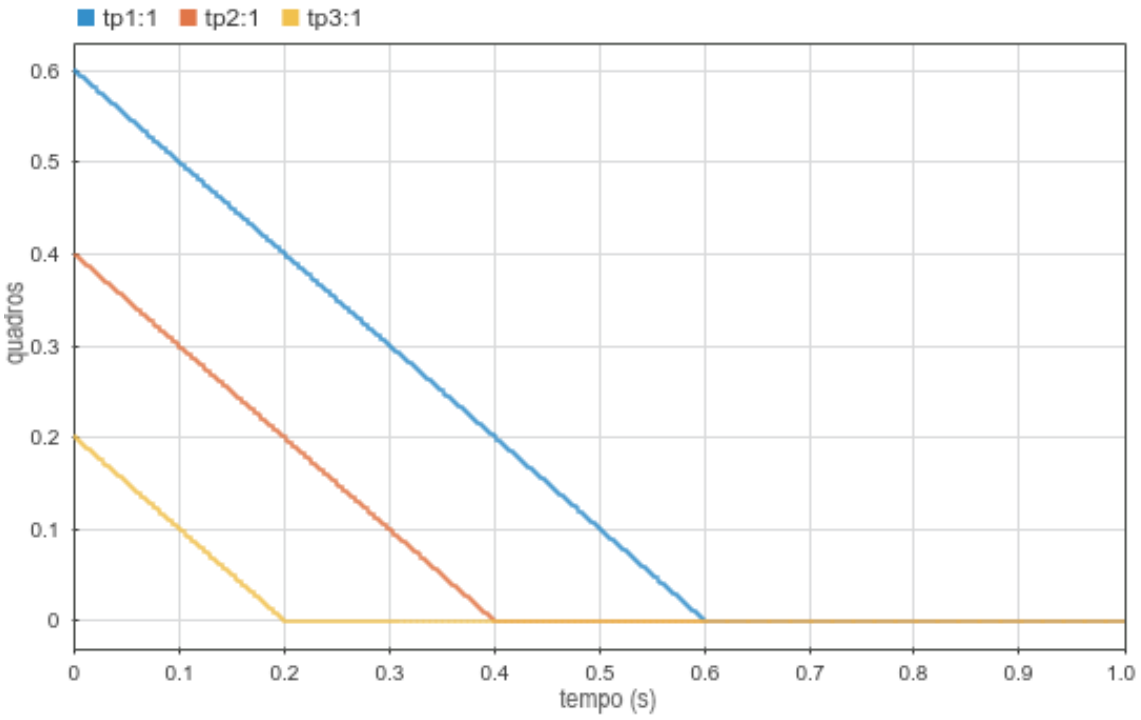

Figura 4.6: Gráfico $q_i(t)$ para o 2º cenário da 1ª configuração com o balanceamento desligado.

A Figura 4.6 demonstra o comportamento do tamanho das filas deste mesmo cenário caso o balanceamento estivesse desligado. A terceira interface, por iniciar com número de quadros menor, esgota sua fila muito rapidamente e fica ociosa enquanto as outras duas interfaces continuam transmitindo. Logo em seguida o mesmo acontece com a segunda interface, que também fica ociosa até o final da simulação.

A importância do balanceamento de carga é evidenciada quando comparamos os gráficos das Figuras 4.4 e 4.6. O sistema desbalanceado desperdiça grande parte dos seus recursos mantendo duas interfaces inoperantes, e leva cerca de 0,6 segundos para finalizar a transmissão de seus quadros. Em contrapartida o sistema balanceado concluiu sua transmissão em aproximadamente 0,4 segundos, equivalente a uma redução de 33%.

Uma carga bem distribuída corresponde a um menor tempo médio de espera, que no contexto de telecomunicações é traduzido diretamente em uma menor latência média de comunicação, característica fundamental para aplicações de tempo real como chamadas telefônicas e video-conferências.

4.3.2 Terceiro cenário

Este cenário adiciona como último obstáculo ao balanceamento a entrada de novos quadros no sistema. As condições iniciais das interfaces são mantidas, porém um novo quadro é acrescentado à cada fila com uma frequência de $1/T_{in}$.

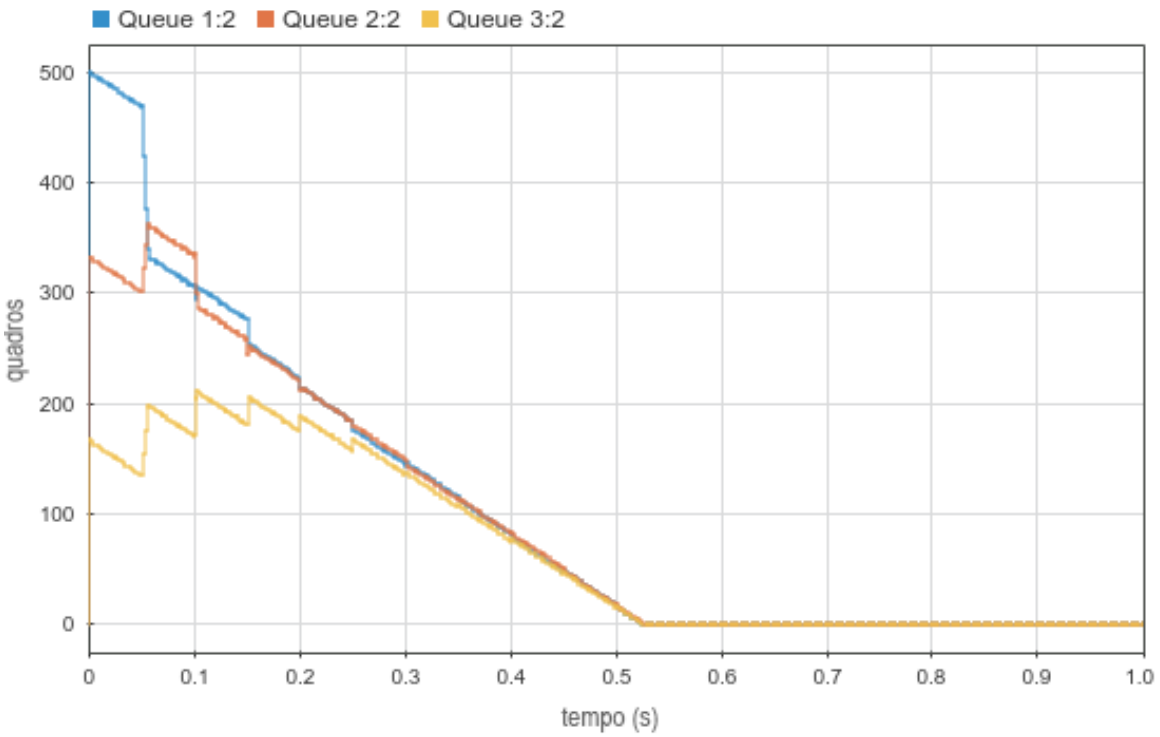

Figura 4.7: Gráfico $q_i(t)$ para o 3º cenário da 1ª configuração.

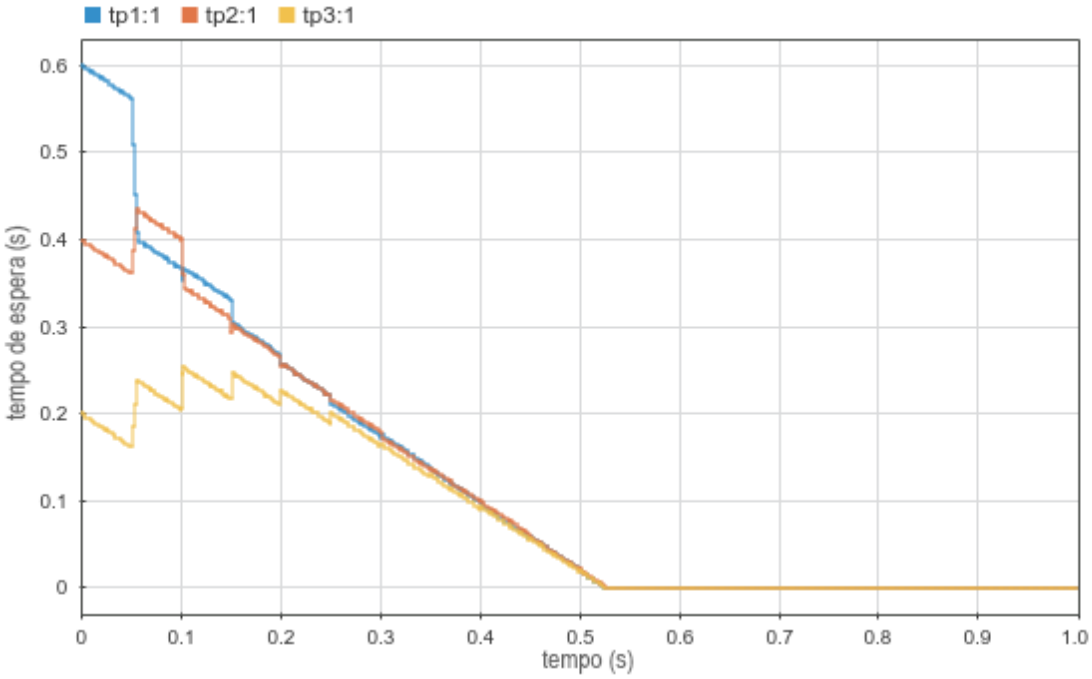

Figura 4.8: Gráfico $x_i(t)$ para o 3º cenário da 1ª configuração.

Sabendo que as três interfaces tem mesma taxa de transmissão r e recebem quantidades iguais de novos quadros por unidade de tempo, uma vez que o balanceamento é atingido não há nenhuma força que atue em favor de um novo desbalanceamento. Esta característica é uma exclusividade das interfaces homogêneas, e o caso oposto será explorado nas seções seguintes.

Os gráficos 4.7 e 4.8, quando comparados aos gráficos do cenário anterior, mostram que o efeito da entrada de novos quadros no caso homogêneo é basicamente a diminuição da taxa líquida de quadros consumidos pelo sistema, ou seja, os segmentos de reta entre dois balanceamentos subsequentes (após período de estabilização das filas) tem inclinação menor.

Como consequência direta disto temos o aumento do tempo necessário para que as filas sejam esvaziadas. Ainda assim, vemos pela Figura 4.8 que o balanceamento foi atingido após aproximadamente 0,3 segundos, e que as filas se esgotaram simultaneamente como esperado.

4.4 Interfaces heterogêneas - caso favorável

Diferentemente da primeira configuração testada, esta configuração contará com três interfaces de rede com taxas de transmissão diversas. As taxas escolhidas são suficientemente diferentes para que novos comportamentos sejam observados nos resultados das simulações, mas suficientemente baixas para que os limites estabelecidos na Seção 4.2 sejam respeitados.

Assim como na configuração homogênea, o mesmo vetor de tempos de espera iniciais x_0 será usado para calcular o estado inicial q_0 da simulação que será repetido em todos os cenários.

A Tabela 4.3 traz um resumo dos parâmetros usados como entrada para a simulação.

	Interface 1	Interface 2	Interface 2
r	$32 \cdot 10^6$ Mb/s	$16 \cdot 10^6$ Mb/s	$10 \cdot 10^6$ Mb/s
t_p	$4 \cdot 10^{-4}$ s	$8 \cdot 10^{-4}$ s	$12 \cdot 10^{-4}$ s
x_0	0,6 s	0,4 s	0,2 s
q_0	1600 quadros	533 quadros	167 quadros
K	2.457	4.002	5.241

Tabela 4.3: Parâmetros da segunda configuração.

4.4.1 Primeiro cenário

De forma análoga ao primeiro cenário da configuração anterior, o número total de quadros no sistema será constante para este cenário já que tanto a entrada de novos quadros como a transmissão de quadros pelas interfaces estarão desativadas.

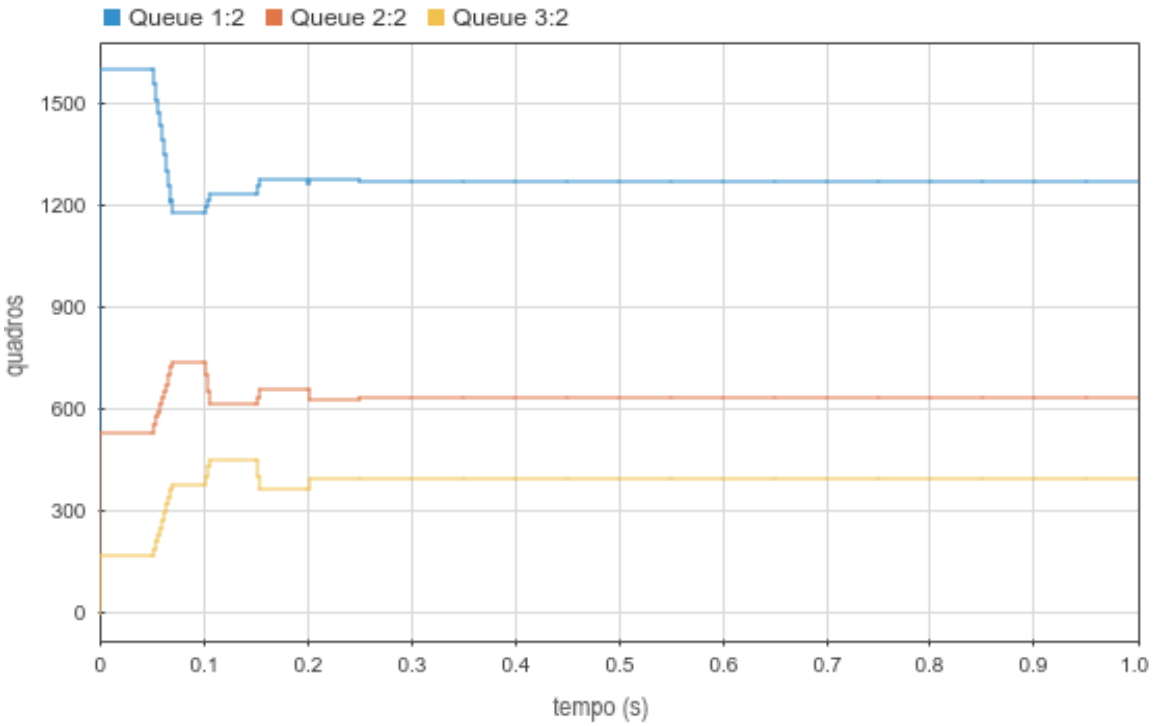

Figura 4.9: Gráfico $q_i(t)$ para o 1º cenário da 2ª configuração.

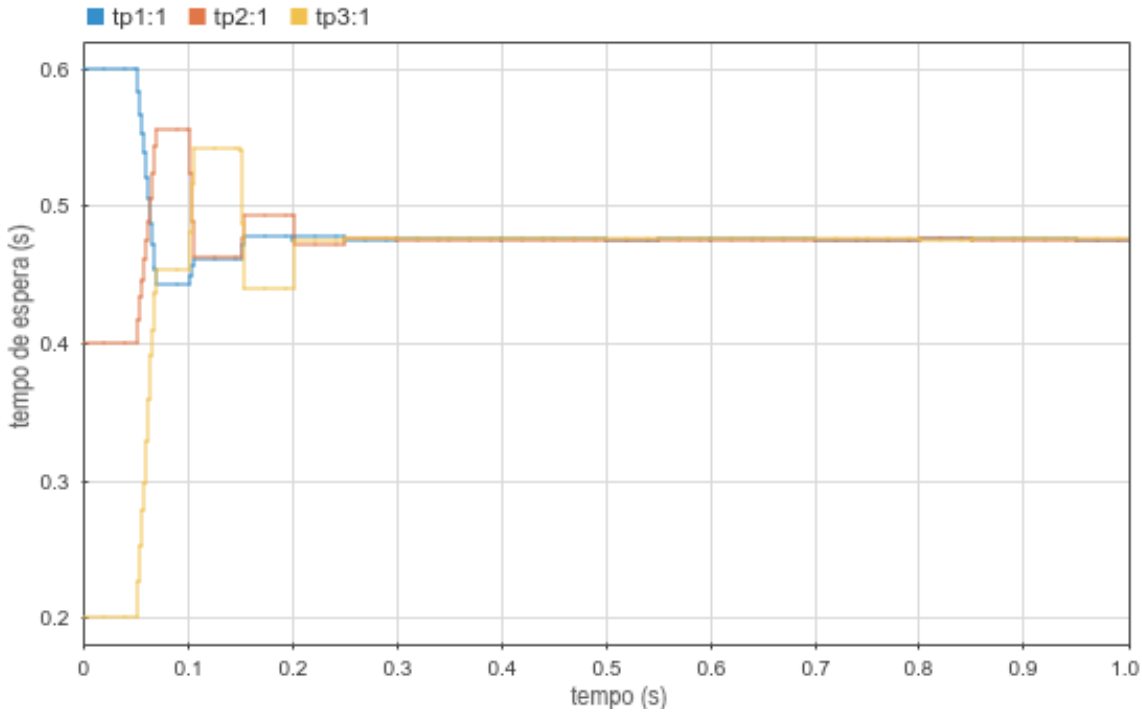

Figura 4.10: Gráfico $x_i(t)$ para o 1º cenário da 2ª configuração.

Dado que o tempo total de consumo de uma fila, desconsiderando efeitos do ba-

lanceamento, é diretamente proporcional ao tempo de transmissão de um quadro t_p e ao número total de quadros na fila q, é esperado que interfaces com taxas de transmissão r diferentes tenham filas com tamanhos diferentes após o balanceamento ser concluído.

Além disso, já que neste primeiro cenário o número total de quadros no sistema se mantém constante, não podemos avaliar a eficácia do balanceamento analisando se as filas se esgotam simultaneamente. Mais uma vez, as curvas de tempo de espera (figura 4.10) podem ser usadas já que o sistema estará balanceado a partir do momento em que as três interfaces tiverem tempos de espera muito próximos.

A Figura 4.9 mostra que apesar de boa parte dos quadros serem redistribuídos durante a simulação, as filas não trocam de posição quando ordenadas por suas quantidades de quadros.

Este fenômeno é observado quando as quantidades de quadros em cada interface no instante inicial são proporcionais às taxas de transferência das interfaces, ou seja, a interface mais rápida inicia com sua fila mais carregada.

Essa situação é mais favorável para o balanceamento e os tempos de espera tendem a convergir mais rapidamente do que no caso contrário, dado que uma porcentagem menor de quadros precisa ser movida. A configuração tratada na Seção 4.5 explora o caso contrário, onde o sistema inicia de forma mais desbalanceada.

Vemos pela Figura 4.10 que os tempos de espera seguem os especificados na Tabela 4.3, e que eles se aproximam bem rapidamente, chegando a um estado bem balanceado após 0,2 segundos e ficando indistinguíveis a partir de 0,25 segundos.

4.4.2 Segundo cenário

O segundo cenário utiliza as mesmas condições iniciais do cenário anterior, porém libera a transmissão dos quadros pelas interfaces de rede. Uma vez que neste cenário o número de quadros no sistema decresce, o resultado desejado é que todas as filas sejam esvaziadas ao mesmo tempo, sem que nenhuma interface fique ociosa.

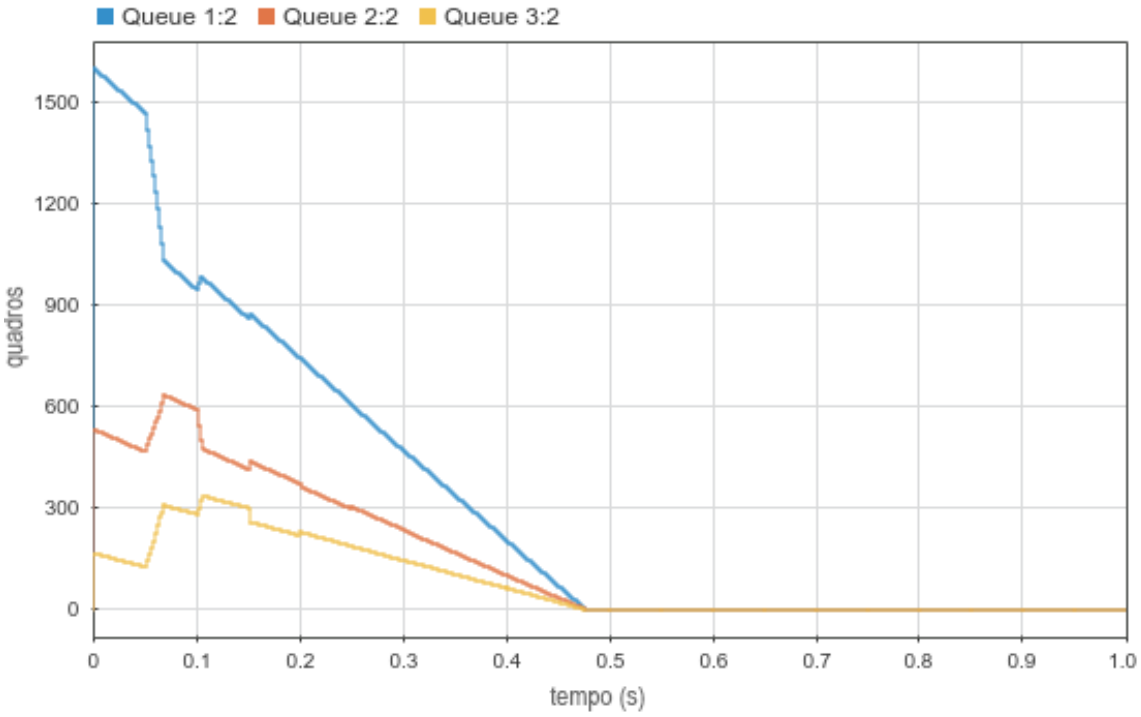

Figura 4.11: Gráfico $q_i(t)$ para o 2º cenário da 2ª configuração.

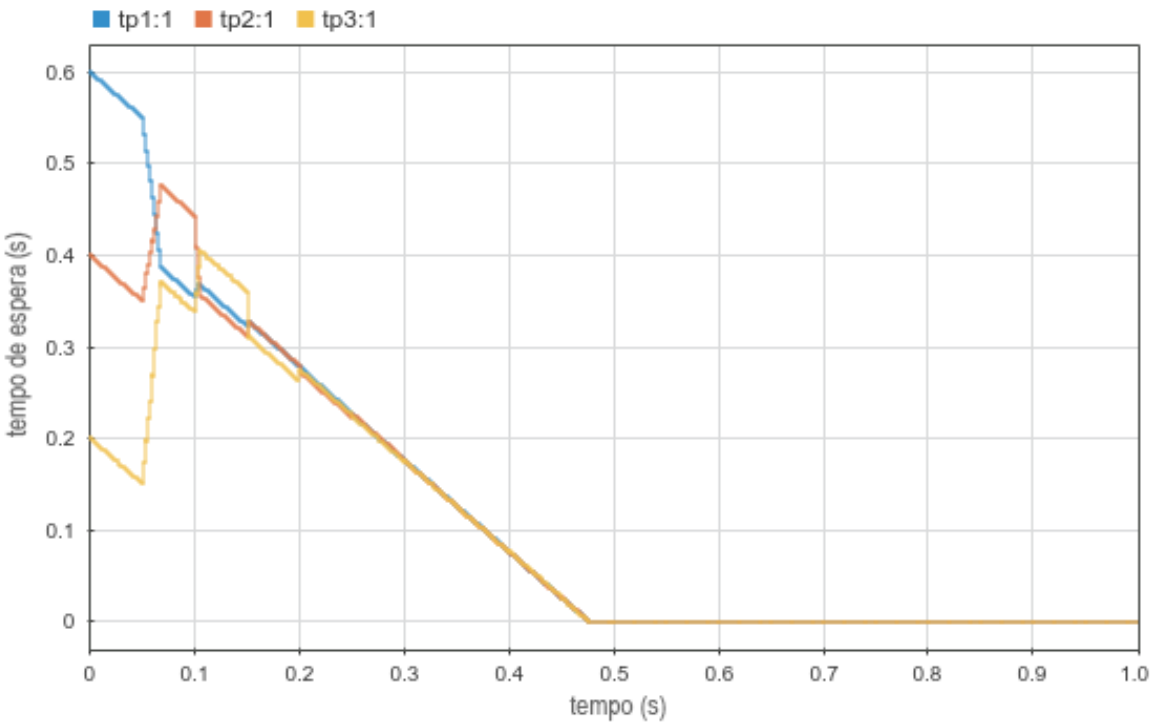

Figura 4.12: Gráfico $x_i(t)$ para o 2º cenário da 2ª configuração.

A primeira diferença que observamos ao comparar o gráfico 4.11 com o gráfico

4.4 da configuração anterior (interfaces homogêneas) é que, na configuração atual, as inclinações dos segmentos de reta entre cada execução do balanceamento são diferentes.

O gráfico 4.13 ilustra o comportamento dos tamanhos das filas no cenário atual caso o balanceamento fosse desligado, e facilita a visualização desta diferença, que é causada pela diversidade de valores de taxa de transmissão r e, consequentemente, pelos diferentes tempos de transmissão de quadros t_p.

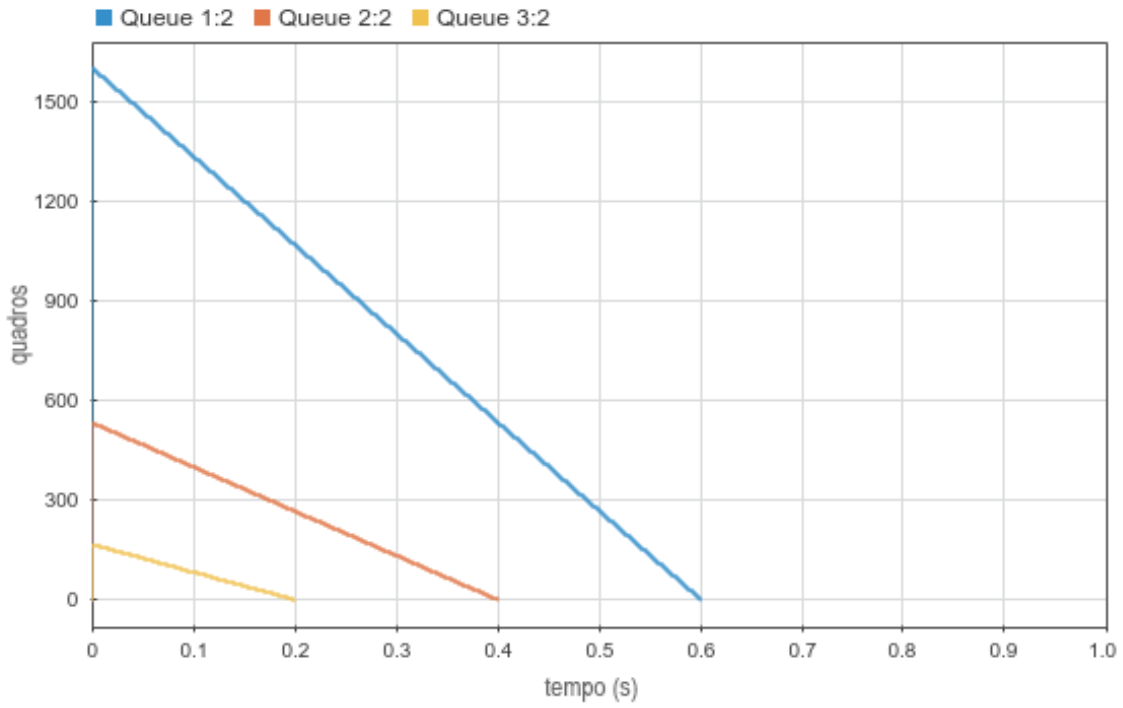

Figura 4.13: Gráfico $x_i(t)$ para o 2º cenário da 2ª configuração com o balanceamento desligado.

O mesmo não acontece com o gráfico de 4.12 dos tempos de espera, que terá sempre uma inclinação de 45 graus nas situações onde os quadros somente saem do sistema, sem que ocorra nenhuma entrada. Isso acontece porque, a cada t_p segundos, um quadro é transmitido pela interface de rede e, consequentemente, os mesmos t_p segundos são reduzidos do tempo total de espera daquela interface.

A Figura 4.13 também contribui para demonstrar que os recursos são subutilizados quando o balanceador não atua, tendo em vista que duas interfaces ficam ociosas enquanto uma interface com excesso de carga continua transmitindo seus quadros.

Vemos no gráfico 4.12 que o balanceamento é atingido aproximadamente aos 0,2 segundos, que é o primeiro ponto onde temos tempos de espera muito próximos. Como

esperado, ambos os gráficos deixam claro que as filas se esgotam ao mesmo tempo, demonstrando que o sistema foi balanceado com sucesso.

4.4.3 Terceiro cenário

O terceiro cenário mantém as premissas do cenário anterior mas adiciona a entrada periódica de quadros nas filas como forma de resistência ao balanceamento.

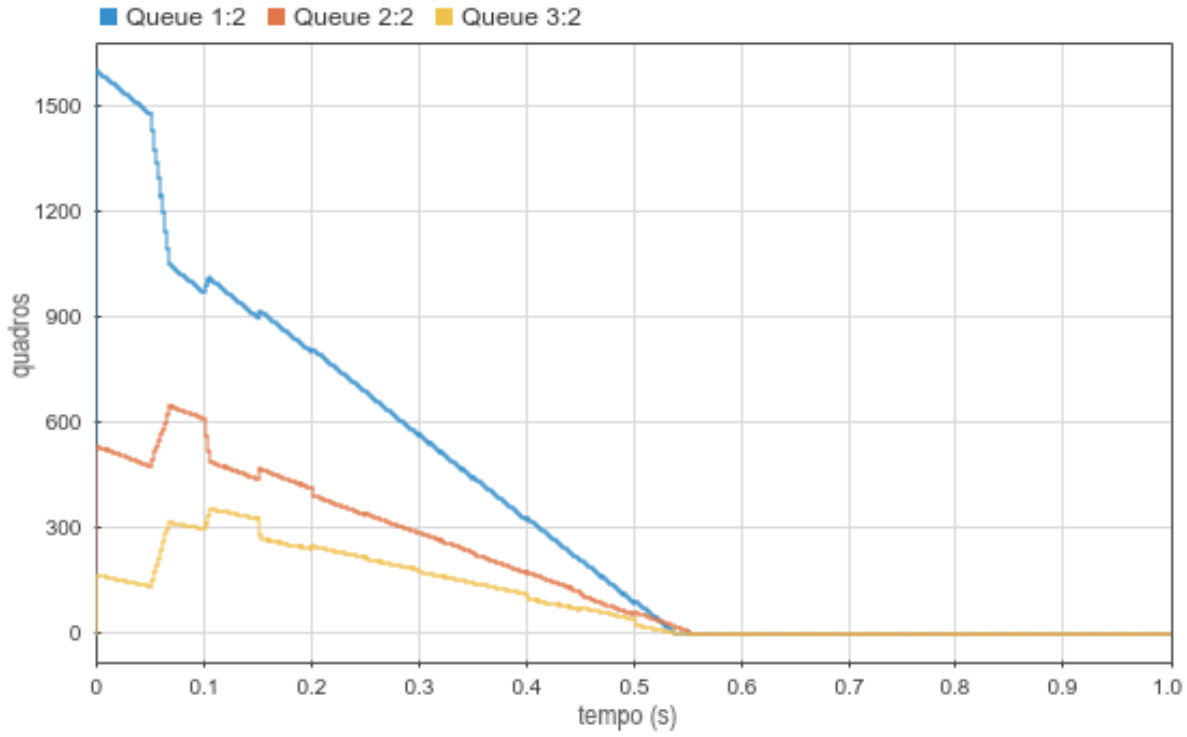

Figura 4.14: Gráfico $q_i(t)$ para o 3º cenário da 2ª configuração.

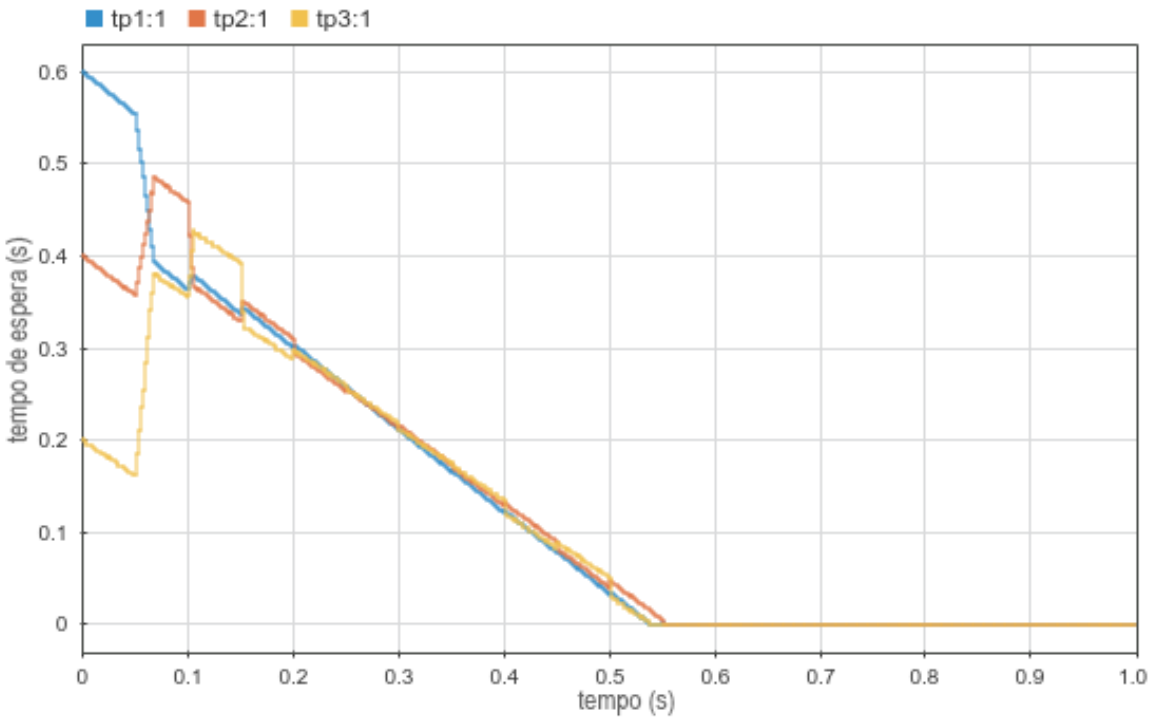

Figura 4.15: Gráfico $x_i(t)$ para o 3º cenário da 2ª configuração.

O gráfico 4.14 da quantidade de quadros nas filas é bastante similar ao gráfico 4.7 do cenário anterior, porém as inclinações dos segmentos de reta entre os balanceamentos são um pouco menores.

Isso acontece porque cada interface recebe um novo quadro a cada T_{in} segundos, efetivamente reduzindo a taxa de redução do tamanho da fila, já que as taxas de transmissão t_p de cada fila são mantidas.

Podemos observar no gráfico 4.15 que o balanceamento é concluído após aproximadamente 0,2 segundos. Entretanto, como a taxa de entrada de quadros é igual para todas as interfaces e a taxa de saída é diferente, o sistema tende a se desbalancear com o tempo.

A partir de 0,2 segundos vemos que as curvas de tempo de espera se afastam gradativamente conforme novos quadros entram no sistema.

Como a performance computacional do algoritmo não é alvo de estudo deste trabalho, não foi adotado um critério de parada para o balanceamento nas simulações. Em uma possível implementação real do algoritmo, um critério deve ser escolhido para evitar que o balanceamento seja executado mesmo que as interfaces estejam suficientemente balanceadas, evitando o desperdício de recursos da máquina.

Por causa disso, vemos no gráfico 4.15 que o balanceamento continua tentando corrigir qualquer pequena diferença no tempo de espera das interfaces causada pela entrada de novos quadros, mesmo que essa diferença ainda seja pequena demais para de fato comprometer o resultado final.

4.5 Interfaces Heterogêneas - caso desfavorável

Esta configuração apresenta três interfaces de rede com taxas de transmissão distintas, assim como a configuração anterior. Entretanto, desta vez as interfaces estão ordenadas de forma crescente quanto às suas velocidades.

Sabendo que todas as simulações utilizam o mesmo vetor de tempos de espera no estado inicial $x_0 = [0,6; 0,4; 0,2]$ segundos, temos que a interface mais lenta irá iniciar com o maior tempo de espera, ou seja, o sistema parte de um estado mais desbalanceado se comparado com a configuração da Seção 4.4.

A Tabela 4.4 resume os valores usados como entrada da simulação.

	Interface 1	Interface 2	Interface 2
r	$8 \cdot 10^6$ Mb/s	$16 \cdot 10^6$ Mb/s	$24 \cdot 10^6$ Mb/s
t_p	$15 \cdot 10^{-4}$ s	$7,5 \cdot 10^{-4}$ s	$5 \cdot 10^{-4}$ s
x_0	0,6 s	0,4 s	0,2 s
q_0	400 quadros	533 quadros	400 quadros
K	5.703	3.547	2.501

Tabela 4.4: Parâmetros da terceira configuração.

4.5.1 Primeiro cenário

No primeiro cenário tanto a entrada como a saída de quadros do sistema não acontece. Assim como no mesmo cenário da segunda configuração (seção 4.4), é esperado que as filas tenham tamanhos diferentes após a conclusão do balanceamento, mas os tempos de espera devem ser muito próximos, o que pode ser observado no gráfico 4.17.

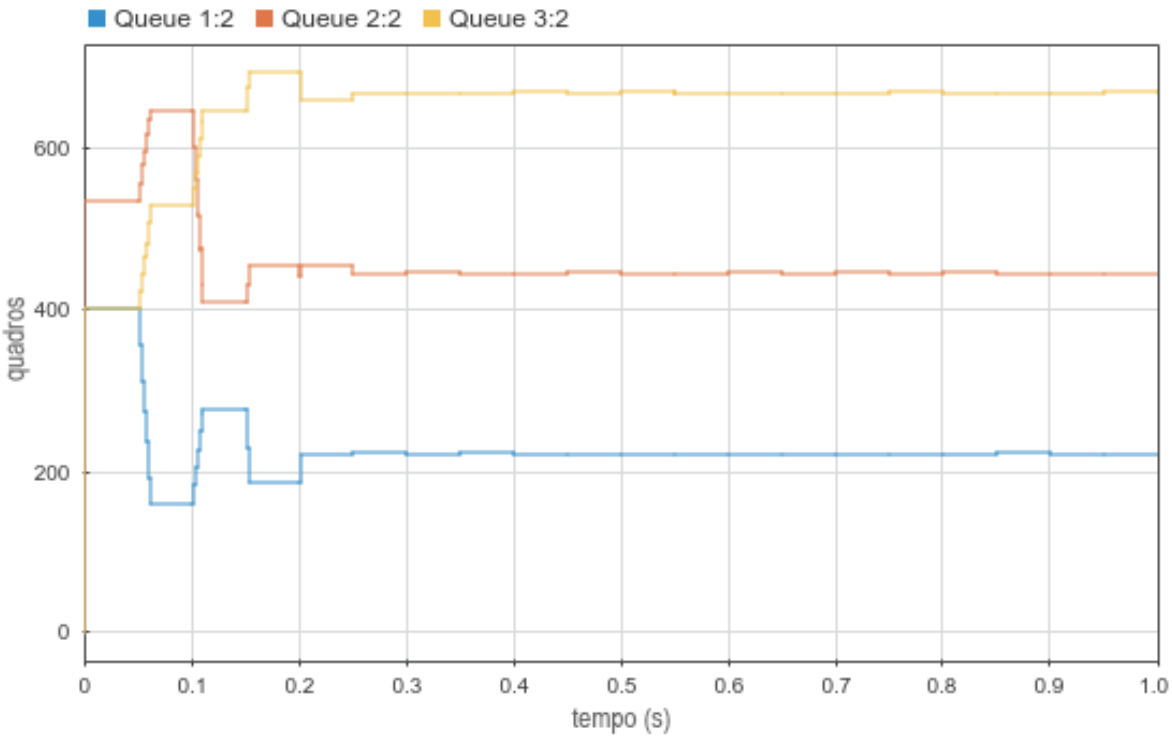

Figura 4.16: Gráfico $q_i(t)$ para o 1º cenário da 3ª configuração.

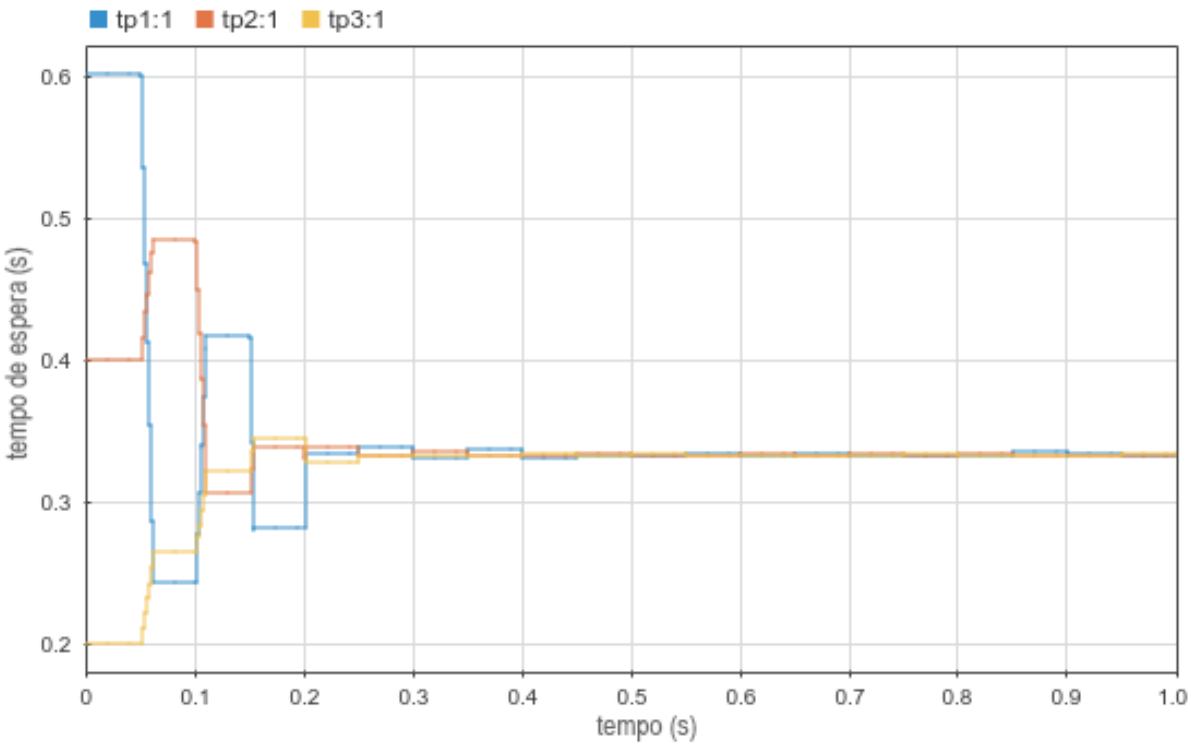

Figura 4.17: Gráfico $x_i(t)$ para o 1º cenário da 3ª configuração.

Após aproximadamente 0,2 segundos vemos que as curvas de tempo de espera já

se encontram muito próximas, indicando que o sistema já está próximo do ponto ótimo de balanceamento. Ainda assim, pequenas variações continuam acontecendo até o final da simulação devido à falta de um critério de parada no balanceador.

O gráfico 4.16 também deixa claro que foi necessário mover uma parcela grande dos quadros presentes nas filas para que o balanceamento fosse concluído, fato que era esperado devido à natureza desfavorável do estado inicial da simulação.

4.5.2 Segundo cenário

O segundo cenário libera a transmissão de quadros pelas interfaces, reduzindo progressivamente a quantidade total de entidades no sistema.

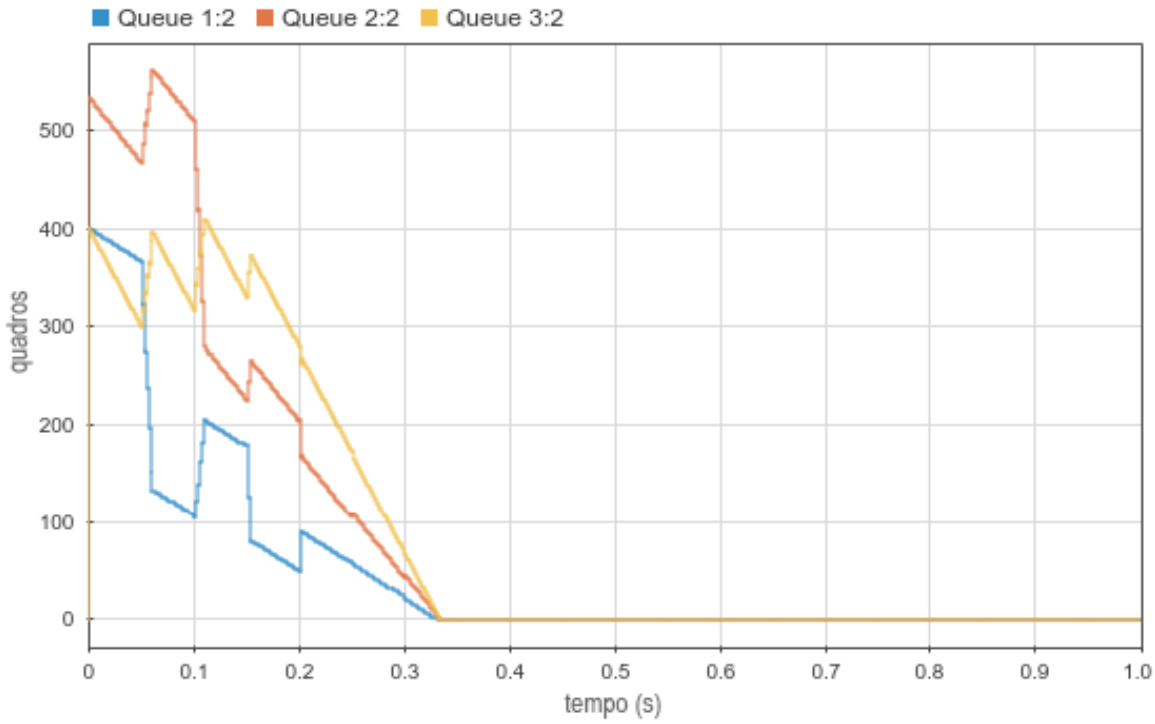

Figura 4.18: Gráfico $q_i(t)$ para o 2º cenário da 3ª configuração.

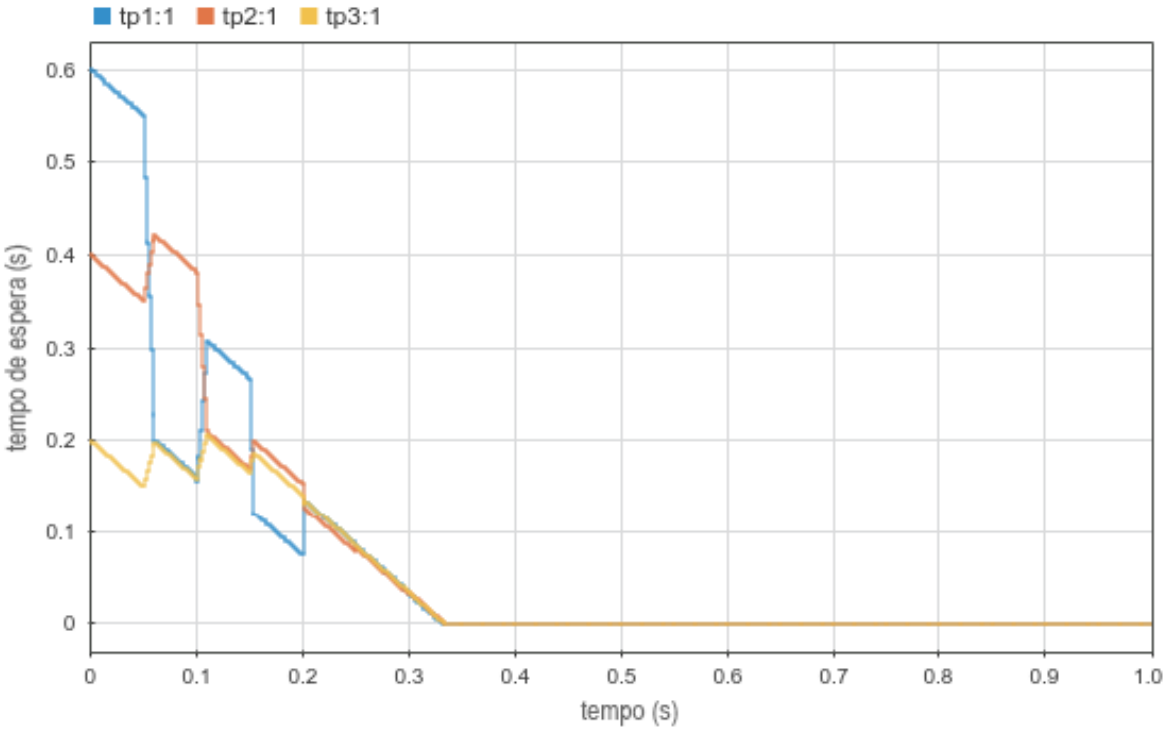

Figura 4.19: Gráfico $x_i(t)$ para o 2º cenário da 3ª configuração.

Novamente vemos pelo gráfico da quantidade de quadros em cada fila (figura 4.18) que cada curva apresenta uma inclinação diferente nos segmentos de reta compreendidos entre as iterações do balanceamento.

Isso se deve aos diferentes tempos de transmissão t_p das interfaces, resultado dos valores diferentes de taxas de transmissão r. Quanto mais alta for a taxa de transmissão de uma interface, mais rapidamente ela consegue eliminar quadros de sua fila e mais vertical será seu gráfico de $q(t)$.

De forma análoga ao mesmo cenário da configuração anterior, devemos usar o gráfico de tempos de espera para avaliar o tempo total de balanceamento quando as interfaces não são homogêneas. O gráfico da Figura 4.19 mostra que o balanceamento foi atingido por volta de 0,2 segundos, e que as filas permaneceram balanceadas até o fim de simulação.

Entretanto, se compararmos as curvas do gráfico 4.19 com as curvas do gráfico equivalente da configuração anterior (figura 4.12), percebemos que as filas oscilam muito mais no caso atual, onde as condições iniciais são mais desfavoráveis.

Estas oscilações são indesejadas já que, em uma implementação real do algoritmo, a transferência desnecessária de quadros entre interfaces é traduzida diretamente em des-

perdício de recursos computacionais.

Ainda assim, podemos observar tanto pelo gráfico 4.18 como pelo gráfico 4.19 que as filas se esgotam simultaneamente, que é o resultado desejado.

4.5.3 Terceiro cenário

O terceiro e último cenário introduz a entrada periódica de novos quadros nas filas das interfaces de rede, mantendo as saídas por transmissão abertas como no cenário anterior.

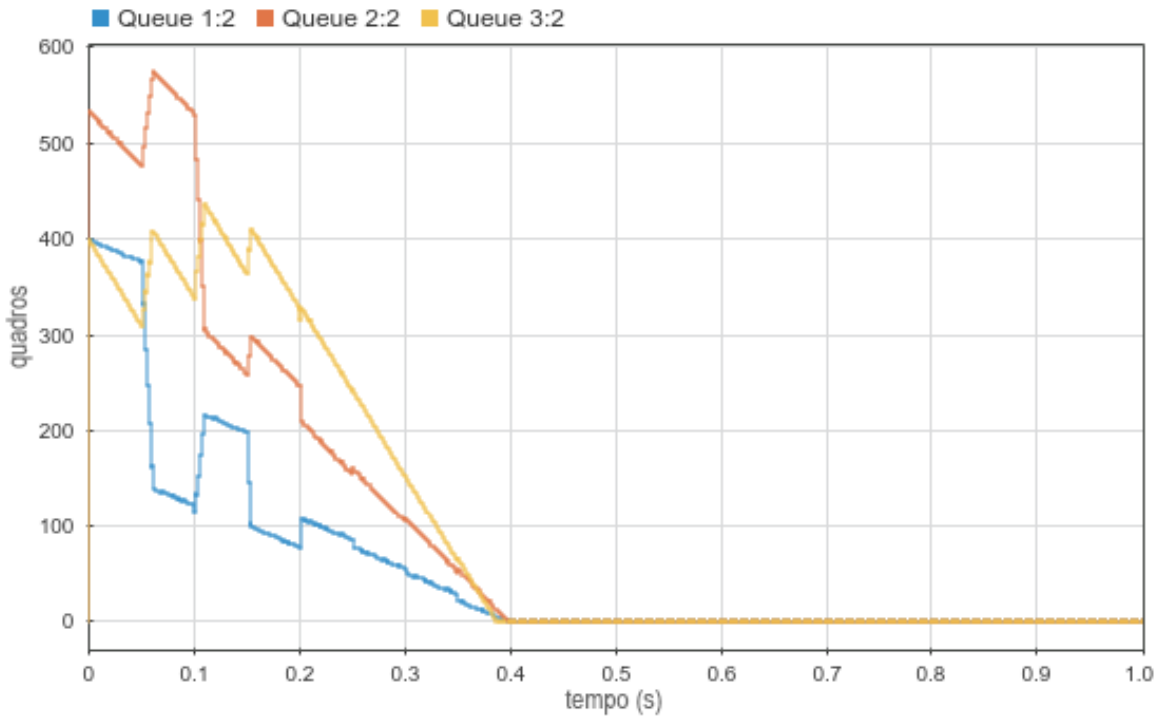

Figura 4.20: Gráfico $q_i(t)$ para o 3º cenário da 3ª configuração.

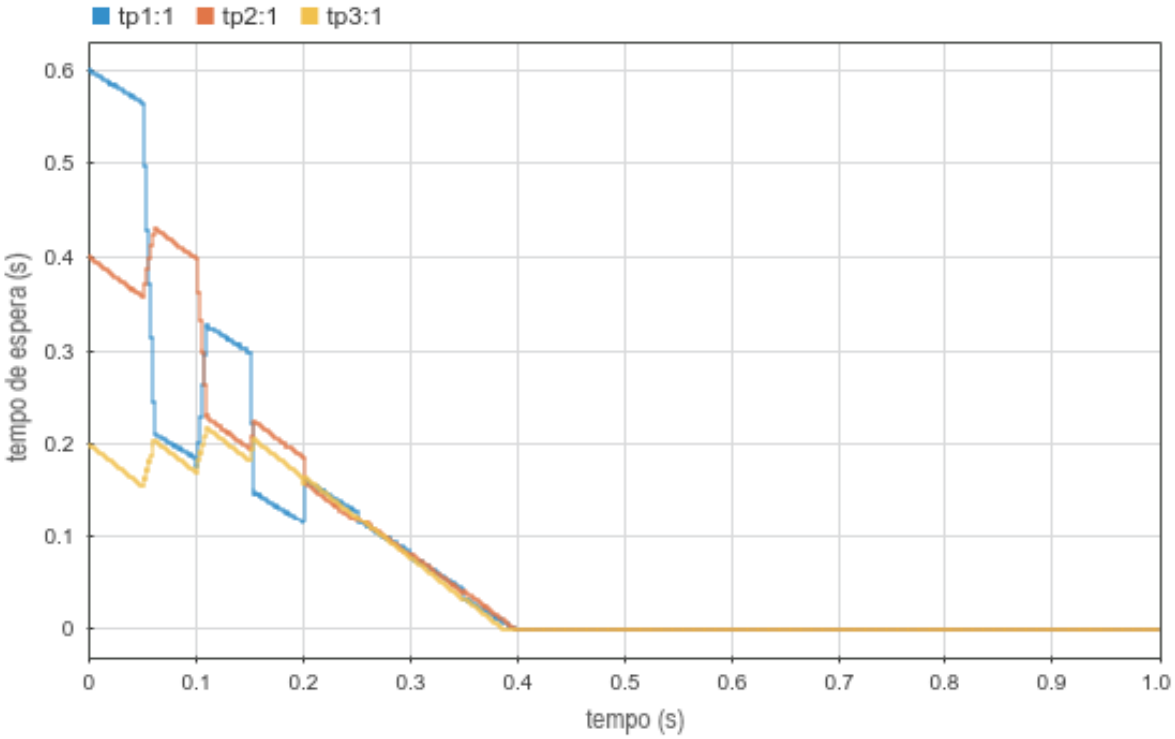

Figura 4.21: Gráfico $x_i(t)$ para o 3º cenário da 3ª configuração.

Vemos pelas curvas da Figura 4.21 que as três interfaces iniciam com os tempos de espera especificados no vetor x_0, mas se aproximam a cada iteração do balanceamento.

Após aproximadamente 0,2 segundos as curvas de tempo de espera já se encontram umas sobre as outras, indicando que o balanceamento foi atingido. Ainda assim, como todas as interfaces recebem quadros com uma mesma frequência mas transmitem segundo taxas diferentes, as curvas de tempo de espera tendem a se afastar novamente.

Já que não foi adotado critério de parada para o balanceamento, a cada T_b segundos o balanceador tenta corrigir a diferença desenvolvida no tempo de espera, por menor que esta seja.

Tanto o gráfico 4.21 como o 4.20 mostram que todas as filas se esgotam aproximadamente após 0,4 segundos, que é o resultado esperado para um sistema balanceado.

Capítulo 5

Conclusões

Primeiramente foi introduzido o modelo contínuo de Ghanem-Chiasson, que aborda o problema do balanceamento de cargas através do tempo de espera nas filas. Foi visto que a lei de controle descrita por este modelo depende de valores de ganho que precisam ser arbitrados, já que não é descrita forma de calcula-los.

Em seguida, o modelo de Hopfield-Tank para redes neurais foi apresentado, assim como sua variação, o modelo de Hopfield-Tank com atrasos, ou HDNN.

Algumas simplificações foram feitas no modelo de Ghanem de forma a demonstrar que este pode ser visto como um caso particular de uma rede neural de Hopfield-Tank com atraso. Isso nos permitiu usar o modelo proposto por *da Silva, J.M.M.* em [5] para calcular valores apropriados para os ganhos.

Em seguida, foi apresentada a ferramenta SimEvents, que proporciona uma forma simples e visual de modelar e simular problemas envolvendo unidade indivisíveis, chamadas entidades.

Um modelo de interface de rede simplex foi idealizado no SimEvents, e foi demonstrado que este modelo não seria capaz de atender a todas as necessidades do projeto.

Por este motivo, foram desenvolvidos novos blocos para a ferramenta, o que possibilitou a criação de um modelo experimental completo capaz de simular situações de balanceamento de carga entre interfaces de rede.

Foram estabelecidos diversos cenários para colocar a prova a eficácia do algoritmo de balanceamento, e estes cenários foram simulados por intermédio do modelo experimental desenvolvido.

Os resultados das simulações foram analisados, e mostraram resultados bastante

positivos. Salvo pequenas diferenças no tempo necessário para balancear os sistemas, em todos os casos testados o balanceamento foi bem sucedido, fazendo com que as filas das interfaces se esgotassem simultaneamente.

Capítulo 6

Sugestões para trabalhos futuros

O modelo descrito no Capítulo 3 pode ser modularizado, de forma a facilitar o desenvolvimento de um modelo com número maior de interfaces de rede, tornando-o mais genérico.

Este mesmo modelo pode ser implementado utilizando uma ferramenta que não sofra com as limitações expostas na Seção 4.2, de forma a permitir que casos mais extremos sejam simulados, como interfaces de rede com altas velocidades ou filas excessivamente carregadas.

Ademais, o modelo de balanceamento de carga exposto neste trabalho pode ser implementado via software em roteadores reais utilizando sistemas operacionais especializados como o OpenWrt. Desta forma, podem ser desenvolvidos novos casos de teste a fim de demonstrar a eficácia do algoritmo de balanceamento em situações mais realistas.

Referências Bibliográficas

[1] Ghanem, J., Abdallah, C.T., Hayat, M., Dhakal, S., Birdwell, J.D., Chiasson, J. and Tang, Z. *Implementation of the load balancing algorithm over a local area network and the internet* 43rd IEEE Conference on Decision and Control, Vol. 1, IEEE, Piscataway, NJ, pp. 4199-204, 2004.

[2] CHIASSON, J., TANG, Z., GHANEM, J., ET AL., *The effect of time delays on the stability of load balancing algorithms for parallel computations* IEEE Transactionson Control Systems Technology v. 13, n. 6, pp. 932–942, November 2005.

[3] HUI, C.-C., CHANSON, S. T., *Hydrodynamic load balancing.* IEEE Transactions on Parallel and Distributed Systems v. 10, n. 11, pp. 1118–1137, November 1999.

[4] Tank, D. W., Hopfield, J. J *Simple neural optimization networks: An a/d converter, signal decision circuit, and a linear programming network* IEEE Transactions on Circuits and Systems v. 33, pp. 533–541, 1986.

[5] da Silva, J.M.M. *Controller synthesis for the load balancing problem in heterogeneous clusters* PhD thesis, Rio de Janeiro's Federal University (COPPE/UFRJ), Rio de Janeiro, 2008.

[6] João Marcos Meirelles da Silva, Eugenius Kaszkurewicz, *A proposed solution for the load balancing problem on heterogeneous clusters based on a delayed neural network* International Journal of Intelligent Computing and Cybernetics, Vol. 3 Iss: 1 pp. 73 - 93, 2010.

[7] **Mathworks** (Entity Types), *https://www.mathworks.com/help/simevents/ug/entity-types.html*

[8] **Mathworks** (What is an Entity), *https://www.mathworks.com/help/simevents/gs/what-is-an-entity.html*

[9] **Mathworks** (Events and event actions), *https://www.mathworks.com/help/simevents/ug/events-and-event-actions.html*

[10] **Mathworks** (Livelock Prevention), *https://www.mathworks.com/help/simevents/ug/livelock-prevention.html*