



TESE DE DOUTORADO EM ENGENHARIA DE SISTEMAS
ELETRÔNICOS E DE AUTOMAÇÃO

**Novas Metodologias AQM e TCP Visando a Eficiência de Fluxos de
Controle UDP e Dados TCP/IP Compartilhados**

Luciano Mauro Arley Sup

Brasília, Julho de 2017

UNIVERSIDADE DE BRASÍLIA

FACULDADE DE TECNOLOGIA

UNIVERSIDADE DE BRASÍLIA
FACULDADE DE TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

Novas Metodologias AQM e TCP Visando a Eficiência de Fluxos de Controle

UDP e Dados TCP/IP Compartilhados

Luciano Mauro Arley Sup

Orientador: Dr. Renato Mariz de Moraes

Co-Orientador: Dr. Adolfo Bauchspiess

TESE DE DOUTORADO EM ENGENHARIA DE SISTEMAS ELETRÔNICOS E DE
AUTOMAÇÃO

Brasília/DF: Julho de 2017

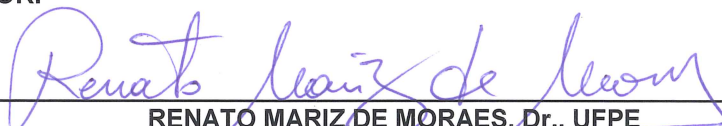
**UNIVERSIDADE DE BRASÍLIA
FACULDADE DE TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA**

**NOVAS METODOLOGIAS AQM E TCP VISANDO A EFICIÊNCIA DE
FLUXOS DE CONTROLE UDP E DADOS TCP/IP
COMPARTILHADOS**

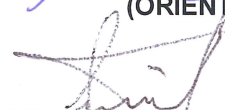
LUCIANO MAURO ARLEY SUP

TESE DE DOUTORADO SUBMETIDA AO DEPARTAMENTO DE ENGENHARIA ELÉTRICA DA
FACULDADE DE TECNOLOGIA DA UNIVERSIDADE DE BRASÍLIA, COMO PARTE DOS
REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE DOUTOR.


APROVADA POR:



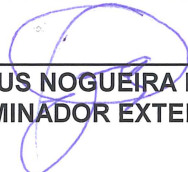
RENATO MARIZ DE MORAES, Dr., UFPE
(ORIENTADOR)



JACIR LUIZ BONDIM, Dr., CIC/UNB
(EXAMINADOR INTERNO)



MARCELO MENEZES DE CARVALHO, Dr., ENE/UNB
(EXAMINADOR INTERNO)



ANTÔNIO MARCUS NOGUEIRA LIMA, Dr., UFCG
(EXAMINADOR EXTERNO)

Brasília, 21 de julho de 2017.

FICHA CATALOGRÁFICA

LUCIANO MAURO ARLEY, SUP	
Novas Metodologias AQM e TCP Visando a Eficiência de Fluxos de Controle UDP e Dados TCP/IP Compartilhados	
[Distrito Federal] 2017.	
x, 101p., 297 mm (FT/UnB, Doutorado, Engenharia de Sistemas Eletrônicos e de Automação, 2017). Tese de doutorado – Universidade de Brasília.Faculdade de Tecnologia.	
I. Eléctrica/FT/UnB	II. Título (Série)

REFERÊNCIA BIBLIOGRÁFICA

SUP, LUCIANO MAURO ARLEY, (2017). Novas Metodologias AQM e TCP Visando a Eficiência de Fluxos de Controle UDP e Dados TCP/IP Compartilhados. Tese de Doutorado em Engenharia de Sistemas Eletrônicos e Automação, Publicação FT.TG-n°119/17, Faculdade de Tecnologia, Universidade de Brasília, Brasília, DF.

CESSÃO DE DIREITOS


AUTOR: Luciano Mauro Arley Sup

TÍTULO DA TESE: Novas Metodologias AQM e TCP Visando a Eficiência de Fluxos de Controle UDP e Dados TCP/IP Compartilhados.

GRAU: Doutor

ANO: 2017

É concedida à Universidade de Brasília permissão para reproduzir cópias deste Trabalho de Doutorado e para emprestar ou vender tais cópias somente para propósitos acadêmicos e científicos. O autor reserva outros direitos de publicação e nenhuma parte dessa Tese pode ser reproduzida sem autorização por escrito do autor.



Luciano Mauro Arley Sup

Brasília – DF – Brasil.

Dedicatória

Dedicado a todos os camponeses da província de Misiones, em especial às crianças que trabalham nas plantações de fumo.

Luciano Mauro Arley Sup

Agradecimentos

Agradeço primeiramente a Deus, que de forma singular me conduziu até aqui.

Agradeço aos meus pais, Pedro Sup e Tereza Etefvina Marostega que apesar de serem camponeses quase analfabetos são pessoa muito sabias e possuidoras de grandes conhecimentos empíricos. Foram eles meus primeiros e grandes mestres, deles tenho aprendido princípios e valores que fazem parte fundamental da minha formação.

Agradeço muito aos meus orientadores Dr. Renato Mariz de Moraes e Dr. Adolfo Bauchspiess pela amizade e grande empenho que tenham colocado na orientação desse trabalho.

Agradeço à CAPES pela bolsa de estudos.

E agradeço a todos os professores e alunos do PGEA e todas as pessoas que tenham contribuído de forma direta ou indireta para a realização deste trabalho.

A todos, Muito Obrigado!

Luciano Mauro Arley Sup

RESUMO

Resumo: Na atualidade a Internet se tornou uma rede capaz de interconectar diversos tipos de usuários, dispositivos, casas, edifícios, plantas industriais, automóveis, hospitais, escolas, paradigma conhecido como Internet das Coisas (IoC). Convertendo-se assim em um meio que abriga fluxos de dados TCP/IP e fluxos UDP de sistemas de controle em rede denominados NCS (*Networked Control Systems*) utilizados em diversas aplicações, tais como, automação industrial, automação predial, telecirurgias, *smart grid* e *smart city*. Para estudar ambos os tipos de fluxos em uma abordagem concomitante, neste trabalho, primeiramente foi desenvolvida uma metodologia que admite modelar e simular topologias de redes de comunicação junto com seus protocolos e algoritmos usando o UPPAAL, uma ferramenta de software que admite modelar sistemas físicos e lógicos através de autômatos temporizados e também fazer simulações e verificações determinísticas e estatísticas do modelo. Então, foram modelados sistemas de controle através da Internet junto com outros fluxos TCP/IP genéricos, e foi observado e estudado como as características do sistema de comunicação afetam o desempenho do sistema de controle em diferentes cenários de simulações e com diferentes técnicas AQM (*Active Queue Management*) implementadas nos roteadores. Neste contexto, com o objetivo de corrigir esses efeitos, e visando à eficiência de ambos os tipos de fluxos (maior vazão para fluxos TCP/IP e menor ITAE (*Integral Time-weighted Absolute Error*) para a NCS) foi desenvolvida uma nova metodologia que combina ações AQM com ações TCP/IP, chamada ENCN (*Explicit Non-Congestion Notification*) cujo desempenho foi comparado com as técnicas AQM Drop Tail, RED, CoDel e PIE trabalhando em conjunto com TCP-Reno, e com os protocolos TCP-Jersey, DCTCP e E-DCTCP (todos modelados e simulados no UPPAAL). Resultados de simulações e verificações estatísticas mostraram que comparado com essas metodologias o ENCN melhorou tanto a vazão e a justiça dos fluxos TCP/IP quanto o ITAE da NCS (fluxo UDP). Porém, o ENCN utiliza alguns recursos tecnológicos (bits para notificar não congestionamento) que nem sempre estão disponíveis na prática. Para superar essa limitação do ENCN, foi desenvolvida uma metodologia chamada ANCE (*Acknowledge-based Non-Congestion Estimation*) a qual trabalha de forma análoga a ENCN, porém, utiliza o valor estimado do tamanho da fila ao invés de utilizar uma notificação explícita de não congestionamento advinda dos roteadores, o que torna essa técnica mais viável para implementação. Finalmente, para superar algumas limitações de ANCE, foi desenvolvido um protocolo de transporte de dados chamado TCP-Puerto-Londero que a diferença de ANCE faz um ajuste adaptativo da janela de transmissão de dados em função de um atraso relativo no caminho de ida, ao invés de utilizar o RTT. Resultados de simulações mostraram que apesar de não utilizar recursos para fazer notificações explícitas de não congestionamento, o TCP-Puerto-Londero fornece um desempenho comparável ao ENCN, superando o TCP-Jersey em 12,03 % e o E-DCTCP em 4,21 % em termos de vazão para os fluxos TCP/IP, e reduzindo o ITAE da NCS em 36,84 %, 36,68 % e 4,16 % em relação ao TCP-Jersey, o E-DCTCP e o ENCN, respectivamente. Esse bom desempenho do TCP-Puerto-Londero também foi observado nas verificações estatísticas do modelo.

Palavras Chave: Sistemas de Controle em Rede, Autômatos Temporizados, AQM, TCP.

ABSTRACT

Abstract: Actually the Internet has become a network capable of interconnecting several types of users, objects, houses, buildings, industrial plants, automobiles, hospitals, schools, this paradigm is known as Internet of Things (IoC). Thus, the Internet has becoming a communication medium that hosts TCP/IP data flows and networked control systems (NCS) UDP flows used in several applications, such as, industrial automation, building automation, remote surgery , smart grid and Smart city. To study both types of flows in a concomitant approach, in this work, a modeling methodology was developed that supports modeling and simulation of communication network topologies along with their protocols and others algorithms using the UPPAAL, a software tool that admit modeling of physical and logical systems through timed automata and make simulations and statistical verifications. Thus, control systems over the Internet were modeled along with other generic flows on which we can observe and study how the communication network features affect control performance behavior for different simulation scenarios and with different AQM (Active Queue Management) techniques implemented in the routers. Then, in order to improve the tradeoff between TCP-throughput and ITAE (Integral Time-weighted Absolute Error) for the NCS that share de same network topology, we've developed a new AQM technique called ENCN (Explicit Non-Congestion Notification) whose performance was compared with TCP-Jersey, DCTCP and, E-DCTCP ECN based transport layer protocols, and with Drop Tail, RED, CoDel, and PIE schemes implemented joint with TCP-Reno protocol (all them modeled in UPPAAL). Simulations and statistical verifications show that the ENCN provides better throughput and fairness for TCP/IP flows, and ITAE for NCS UDP flow compared to other methodologies. However, the ENCN uses some technological resources (bits to notify non-congestion) that are not always available. In order to overcome this ENCN limitation we developed a methodology called ANCE (Acknowledge-Based Non-Congestion Estimation), which work like ENCN but making inferences about queue length instead to using explicit non-congestion notification coming from the routers, which makes this technique more feasible for implementation. Finally, in order to overcome some ANCE limitations we developed a new transport layer protocol called TCP-Puerto-Londero, which rather than using the RTT, makes an adaptive adjustment of the congestion window as a function of a relative delay in the forward path. Simulation results showed that TCP-Puerto-Londero provides performance comparable to ENCN, overcoming TCP-Jersey by 12.03 % and the E-DCTCP in 4.21 % in terms of throughput for TCP/IP flows and reducing the ITAE of the NCS UDP flow by 36.84 %, 36.68 % and 4.16 % in relation to the TCP-Jersey, E-DCTCP, and ENCN, respectively. This good performance of TCP-Puerto-Londero was also observed in the statistical verifications.

Keywords: Networked Control Systems, Timed Automata, AQM, TCP.

SUMÁRIO

1	INTRODUÇÃO	1
1.1	INTERDISCIPLINARIEDADE EM UMA NCS	3
1.2	SISTEMAS DE CONTROLE VIA INTERNET	4
1.3	ANÁLISES DO ATRASO DE RESPOSTA MEDIANTE <i>Model Checking</i>	5
1.4	OBJETIVOS DO TRABALHO	7
1.5	METODOLOGIA	7
1.6	CONTRIBUIÇÕES	8
1.7	RESULTADOS OBTIDOS	9
1.8	APRESENTAÇÃO DO MANUSCRITO	10
2	FUNDAMENTOS - REDES COM FLUXOS TCP E NCS COMPARTILHADOS	11
2.1	INTRODUÇÃO	11
2.2	TROCA DE INFORMAÇÃO EM UMA NCS	13
2.3	PARÂMETROS DE DESEMPENHO EM SISTEMAS DE CONTROLE	13
2.4	TRANSPORTE DE DADOS NA INTERNET	16
2.4.1	TCP-RENO	17
2.5	JUSTIÇA PARA FLUXOS QUE COMPARTILHAM UMA MESMA TOPOLOGIA DE REDE DE COMUNICAÇÃO	18
2.6	TÉCNICAS AQM	18
2.6.1	RED (<i>Random Early Detection</i>)	19
2.6.2	CoDEL (<i>Control Delay</i>)	20
2.6.3	PIE (<i>Proportional Integral controller Enhanced</i>)	20
2.6.4	NOTIFICAÇÃO EXPLÍCITA DE CONGESTIONAMENTO (ECN)	23
2.7	PROTOCOLOS DE TRANSPORTE DE DADOS BASEADOS EM ECN	26
2.7.1	TCP-JERSEY	27
2.7.2	DATA CENTER TCP (DCTCP)	28
2.7.3	E-DCTCP	28
2.8	CONCLUSÕES	29
3	MODELAGEM, SIMULAÇÃO E VERIFICAÇÃO EM UPPAAL DE FLUXOS TCP E NCS QUE COMPARTILHAM A MESMA TOPOLOGIA DE REDE DE COMUNICAÇÃO	31
3.1	INTRODUÇÃO	31
3.2	UPPAAL	32

3.2.1	MODELAGEM MEDIANTE AUTÔMATOS TEMPORIZADOS	33
3.2.2	EXEMPLO BÁSICO DE MODELAGEM, SIMULAÇÃO E VERIFICAÇÃO EM UPPAAL	33
3.3	MODELO DE UMA TOPOLOGIA DE REDE DE COMUNICAÇÃO <i>daisy-chain</i> COM UM FLUXO TCP-RENO EM UPPAAL	35
3.3.1	MODELO DO TRANSMISSOR	36
3.3.2	MODELO DO CANAL 1 DE IDA	38
3.3.3	MODELO DA FILA NO CAMINHO DE IDA NO ROTEADOR 1	40
3.3.4	MODELO DO RECEPTOR	41
3.4	SIMULAÇÃO DO MODELO TCP-RENO EM UPPAAL	42
3.5	MODELOS DE TÉCNICAS AQM EM UPPAAL	44
3.5.1	VAZÃO	47
3.5.2	FENÔMENO DE <i>bufferempty</i>	48
3.6	MODELAGEM DE PROTOCOLOS DE TRANSPORTE DE DADOS BASEADOS EM ECN EM UPPAAL.....	50
3.7	MODELAGEM DE NCSs EM UPPAAL	52
3.7.1	INFLUÊNCIA DO SISTEMA DE COMUNICAÇÃO NO DESEMPENHO DO SISTEMA DE CONTROLE	54
3.7.2	INFLUÊNCIA DE OUTROS FLUXOS QUE COMPARTILHAM A MESMA TOPOLOGIA DE REDE DE COMUNICAÇÃO COM UMA NCS	57
3.7.3	INFLUÊNCIA DE TÉCNICAS AQM NOS FLUXOS QUE COMPARTILHAM A MESMA TOPOLOGIA DE REDE DE COMUNICAÇÃO.....	60
3.7.4	ANÁLISE DO DESEMPENHO DE NCSs ATRAVÉS DA INTERNET MEDIANTE A FERRAMENTA <i>Statistical Model Checking</i> (SMC) DO UPPAAL	65
3.8	CONCLUSÕES	68
4	NOVAS TÉCNICAS AQM ENCN E ANCE	70
4.1	INTRODUÇÃO	70
4.2	ENCN (EXPLICIT NON-CONGESTION NOTIFICATION)	71
4.2.1	MARCAÇÃO ENCN	72
4.2.2	REAÇÃO ENCN.....	73
4.2.3	MODELO DO ALGORITMO ENCN NO UPPAAL.....	75
4.2.4	AVALIAÇÃO DO DESEMPENHO DE ENCN EM UMA TOPOLOGIA DAISY-CHAIN ...	78
4.2.5	AVALIAÇÃO DO DESEMPENHO DO ENCN EM UMA TOPOLOGIA <i>Dumbbell</i>	81
4.2.6	AVALIAÇÃO DO DESEMPENHO DO ENCN EM UMA NCS	89
4.2.7	COMPARAÇÃO DO DESEMPENHO DO ENCN COM PROTOCOLOS BASEADOS EM ECN EM UMA NCS	100
4.3	ANCE (<i>Acknowledge-based Non-Congestion Estimation</i>)	104
4.3.1	ESTIMAÇÃO DO TAMANHO FILA	105
4.3.2	MODELO DO ALGORITMO ANCE NO UPPAAL.....	108
4.3.3	AVALIAÇÃO DO DESEMPENHO DE ANCE EM UMA TOPOLOGIA DE DAISY-CHAIN	108
4.3.4	AVALIAÇÃO DO DESEMPENHO DE ANCE EM UMA NCS.....	110
4.3.5	APLICAÇÃO DE ANCE EM TOPOLOGIAS COM FLUXOS NO CAMINHO INVERSO	113

4.4	CONCLUSÕES	115
5	TCP-PUERTO-LONDERO	117
5.1	INTRODUÇÃO	117
5.2	ATRASO RELATIVO NO CAMINHO DE IDA.....	118
5.2.1	USO DO ATRASO RELATIVO NO CAMINHO DE IDA NO TCP-PUERTO-LONDERO	119
5.3	TCP-PUERTO-LONDERO.....	120
5.3.1	IDENTIFICAÇÃO DA FUNÇÃO DE TRANSFERÊNCIA NOMINAL	120
5.3.2	PROJETO DO CONTROLADOR NOMINAL	122
5.3.3	DESEMPENHO DO CONTROLADOR NOMINAL	122
5.3.4	PROJETO DO CONTROLADOR ADAPTATIVO	124
5.3.5	DESCRIÇÃO DO TCP-PUERTO-LONDERO	126
5.4	DESEMPENHO DO TCP-PUERTO-LONDERO NA TOPOLOGIA DE REDE DAISY- CHAIN	128
5.4.1	DESEMPENHO DO TCP-PUERTO-LONDERO NA TOPOLOGIA DE REDE DAISY- CHAIN MODIFICADA.....	130
5.5	DESEMPENHO DO TCP-PUERTO-LONDERO COM CONGESTIONAMENTO NO CAMINHO DE RETORNO	131
5.6	DESEMPENHO DO TCP-PUERTO-LONDERO DIANTE DE FLUXOS QUE INICIAM MAIS TARDE.....	132
5.7	DESEMPENHO DO TCP-PUERTO-LONDERO EM UMA TOPOLOGIA <i>Dumbell</i> COM UMA NCS	133
5.7.1	PROJETO DE UM CONTROLADOR PI BASEADO EM PACOTES.....	134
5.7.2	RESULTADOS DAS SIMULAÇÕES	136
5.7.3	AVALIAÇÃO DO DESEMPENHO DO TCP-PUERTO-LONDERO MEDIANTE A FER- RAMENTA SMC	142
5.8	CONCLUSÃO.....	143
6	CONCLUSÕES	145
6.1	TRABALHOS FUTUROS	148
	REFERÊNCIAS BIBLIOGRÁFICAS	150
	REFERÊNCIAS BIBLIOGRÁFICAS	150
	ANEXOS.....	163
I	MODELO DA TOPOLOGIA DAISY CHAIN NO UPPAAL	164
I.0.1	MODELO DO TRANSMISSOR	165
I.0.2	MODELO DO CANAL 1 DE IDA	167
I.0.3	MODELO DA FILA NO CAMINHO DE IDA NO ROTEADOR 1.....	169
I.0.4	MODELO DO RECEPTOR	170
I.0.5	MODELO DO CANAL 2 DE IDA	171
I.0.6	MODELO DA FILA NO CAMINHO DE IDA NO ROTEADOR 2.....	173

I.0.7	MODELO DO CANAL 3 DE IDA	174
I.0.8	MODELO DO RECEPTOR	176
I.0.9	MODELO DO CANAL 3 DE RETORNO.....	177
I.0.10	MODELO DA FILA NO CAMINHO DE RETORNO NO ROTEADOR 2	178
I.0.11	MODELO DO CANAL 2 DE RETORNO.....	179
I.0.12	MODELO DA FILA NO CAMINHO DE RETORNO NO ROTEADOR 1	181
I.0.13	MODELO DO CANAL 1 DE RETORNO.....	182
II	ANEXO: MODELOS DE TÉCNICAS AQM EM UPPAAL	184
II.1	MODELO DO ALGORITMO RED NO UPPAAL	184
II.2	MODELO DO ALGORITMO CoDEL NO UPPAAL	187
II.3	MODELO DO ALGORITMO PIE NO UPPAAL	189
II.3.1	MODIFICAÇÕES NO MODELO DOS CANAIS PARA MODELAR A TÉCNICA AQM PIE	190
II.3.2	MODIFICAÇÕES NO MODELO DAS FILAS PARA MODELAR A TÉCNICA AQM PIE	193
III	ANEXO: MODELOS DE NCS EM UPPAAL.....	196
III.1	MODELO DA PLANTA	196
III.2	MODELO DO CONTROLADOR	197

LISTA DE FIGURAS

2.1	Networked Control System (parâmetros e variáveis).	12
2.2	Diagrama temporal de atrasos na rede.	13
2.3	Parâmetros da resposta de sistema de controle genérico ao degrau unitário.	14
3.1	Simple autômato de dois estados em UPPAAL.	34
3.2	Exemplo de verificação em UPPAAL.	34
3.3	Topologia de rede de comunicação daisy-chain com um fluxo TCP.	35
3.4	Conjunto de autômatos que modelam a topologia de rede de comunicação daisy-Chain da Figura 3.3.	36
3.5	Modelo do transmissor TCP-Reno.	37
3.6	Modelo do canal 1 de ida.	39
3.7	Modelo da fila de dados no caminho de ida no roteador 1.	41
3.8	Modelo do receptor.	42
3.9	Janela do Transmissor TCP-Reno para Drop Tail implementada na topologia de rede de comunicação daisy-chain.	43
3.10	Fila no caminho de ida no roteador 1 para Drop Tail implementada na topologia de rede de comunicação daisy-chain.	44
3.11	Janela do Transmissor TCP-Reno para RED implementado na Topologia de rede de comunicação daisy-chain.	45
3.12	Fila no caminho de ida do roteador 1 para RED implementado na Topologia de rede de comunicação daisy-chain.	45
3.13	Janela do Transmissor TCP-Reno para CoDel implementado na topologia de rede de comunicação daisy-chain.	46
3.14	Fila no caminho de ida do roteador 1 para CoDel implementado na topologia de rede de comunicação daisy-chain.	46
3.15	Janela do Transmissor TCP-Reno para PIE implementada na topologia de rede de comunicação daisy-chain.	47
3.16	Fila no caminho de ida do roteador 1 para PIE implementada na topologia de rede de comunicação daisy-chain.	47
3.17	Vazão para diferentes técnicas AQM com TCP Reno, implementadas na topologia de rede de comunicação daisy-chain	49
3.18	Períodos de <i>bufferempty</i> causado pela implementação de CoDel na topologia de rede de comunicação daisy-chain.	49
3.19	Janela do Transmissor TCP-Jersey na topologia de rede de comunicação daisy-chain.	51

3.20	Fila no caminho de ida do roteador 1 para TCP-Jersey implementado na topologia de rede de comunicação daisy-Chain.....	51
3.21	Janela do Transmissor E-DCTCP implementado na topologia de rede de comunicação daisy-chain.	51
3.22	Fila no caminho de ida do roteador 1 para E-DCTCP implementado na topologia de rede de comunicação daisy-chain.....	52
3.23	Vazão para TCP-Jersey e E-DCTCP implementados na Topologia de rede de comunicação daisy-chain	52
3.24	Motor CC Maxon F2140.....	53
3.25	NCS através da Internet.	54
3.26	Posição do motor e referência para o Cenário 1.	55
3.27	Posição do motor e referência para o Cenário 2.	56
3.28	Posição do motor e referência para o Cenário 3.	56
3.29	ITAE para os Cenários 1, 2 e 3.....	56
3.30	NCS compartilhando a mesma topologia de rede com um fluxo TCP genérico.....	57
3.31	Posição do motor e referência para o Cenário A.....	58
3.32	Posição do motor e referência para o Cenário B.....	58
3.33	ITAE para os Cenários A e B.	59
3.34	Posição do motor para o Cenário C.....	60
3.35	Posição do motor para a técnica AQM Drop Tail.	61
3.36	Posição do motor para a técnica AQM RED	61
3.37	Posição do motor para a técnica AQM CoDel	61
3.38	Posição do motor para a técnica AQM PIE.	62
3.39	ITAE para diferentes técnicas AQM.	62
3.40	Vazão total dos Fluxos TCP-Reno para diferentes técnicas AQM.	62
3.41	Fila no caminho de ida do roteador 1 para a técnica AQM Drop Tail.....	63
3.42	Fila no caminho de ida do roteador 1 para a técnica AQM RED.....	64
3.43	Fila no caminho de ida do roteador 1 para a técnica AQM CoDel.....	64
3.44	Fila no caminho de ida do roteador 1 para a técnica AQM PIE.....	64
3.45	NCS compartilhando a mesma topologia de rede com um fluxo TCP genérico e um fluxo Poisson aleatório.....	65
3.46	Posição do motor segundo resultado de uma única rodada de simulação.	66
3.47	ITAE segundo resultado de uma única rodada de simulação.	66
3.48	Probabilidade do ITAE ser maior que 1179 rad.s em um tempo menor ou igual a 10 s.	67
3.49	Probabilidade acumulada do ITAE ser maior que 1179 rad.s em menos de 10 s.	67
3.50	Histograma e Polígonos de frequências da probabilidade do ITAE ser maior que 1179 rad.s em menos de 10 s.....	68
3.51	Probabilidade da posição do motor ultrapassar 2 rad antes do primeiro segundo de simulação.....	68
4.1	Funcionamento de ENCN em um roteador.....	73
4.2	Modelo da fila de dados no caminho de retorno, no roteador 1.....	77

4.3	Janela do Transmissor TCP para ENCN implementada na topologia de rede de comunicação ilustrada na Figura 3.3.....	79
4.4	Fila no sentido transmissor/receptor do roteador 1 para ENCN implementada na topologia de rede de comunicação ilustrada na Figura 3.3.	79
4.5	Vazão de ENCN comparada com outras técnicas AQM, implementadas na topologia de rede de comunicação dada na Figura 3.3.	80
4.6	Vazão de ENCN comparada com protocolos TCP baseados em ECN 3.3.	81
4.7	Topologia de rede de comunicação tipo <i>Dumbbell</i> com três fluxos TCP.	82
4.8	Autômato utilizado no UPPAAL para calcular o índice de Jain.	82
4.9	Vazão para a implementação RED na topologia de rede de comunicação ilustrada na Figura 4.7.....	83
4.10	Vazão para a implementação CoDel na topologia de rede de comunicação ilustrada na Figura 4.7.....	83
4.11	Vazão para a implementação PIE na topologia de rede de comunicação ilustrada na Figura 4.7.....	84
4.12	Vazão para a implementação ENCN na topologia de rede de comunicação ilustrada na Figura 4.7.....	84
4.13	Índice de Jain para a topologia de rede dada na Figura 4.7 para diferentes técnicas AQM.	85
4.14	Fila para RED implementado na topologia de rede de comunicação ilustrada na Figura 4.7.....	86
4.15	Fila para CoDel implementado na topologia de rede de comunicação ilustrada na Figura 4.7.....	86
4.16	Fila para PIE implementado na topologia de rede de comunicação ilustrada na Figura 4.7.....	86
4.17	Fila para ENCN implementado na topologia de rede de comunicação ilustrada na Figura 4.7.....	87
4.18	Vazão para TCP-Jersey implementado na topologia de rede de comunicação ilustrada na Figura 4.7.....	87
4.19	Vazão para E-DCTCP implementado na topologia de rede de comunicação ilustrada na Figura 4.7.....	88
4.20	Fila no caminho de ida do roteador 1 para TCP-Jersey implementado na topologia de rede de comunicação ilustrada na Figura 4.7.	88
4.21	Fila no caminho de ida do roteador 1 para E-DCTCP implementado na topologia de rede de comunicação ilustrada na Figura 4.7.	89
4.22	Índice de Jain para a topologia de rede dada na Figura 4.7 para diferentes protocolos.	89
4.23	Topologia de rede de comunicação tipo <i>Dumbbell</i> compartilhada por três fluxos TCP genéricos e um fluxo UDP de uma NCS.	91
4.24	Posição do motor obtida com a implementação de RED.	92
4.25	Posição do motor obtida com a implementação de CoDel.....	92
4.26	Posição do motor obtida com a implementação de PIE.	92
4.27	Posição do motor obtida com a implementação de ENCN.	93

4.28	ITAE para ENCN comparada com outras técnicas AQM.	93
4.29	Vazão para ENCN comparada com outras técnicas AQM.	94
4.30	Índice de Jain para ENCN comparada com outras técnicas AQM.	95
4.31	Fila para RED implementado na topologia de rede de comunicação ilustrada na Figura 4.23.	95
4.32	Fila para CoDel implementado na topologia de rede de comunicação ilustrada na Figura 4.23.	95
4.33	Fila para PIE implementado na topologia de rede de comunicação ilustrada na Figura 4.23.	96
4.34	Fila para ENCN implementado na topologia de rede de comunicação ilustrada na Figura 4.23.	96
4.35	Probabilidade acumulada do ITAE ser maior que 8727 rad.s em até 30 s.	97
4.36	Histograma e Polígonos de frequências da probabilidade do ITAE gerado por RED ser maior que 8727 rad.s em até 30 s.	98
4.37	Histograma e Polígonos de frequências da probabilidade do ITAE gerado por CoDel ser maior que 8727 rad.s em até 30 s.	98
4.38	Histograma e Polígonos de frequências da probabilidade do ITAE gerado por PIE ser maior que 8727 rad.s em até 30 s.	98
4.39	Probabilidade acumulada da vazão total ser maior que 1,45 Mbps em até 30 s.	99
4.40	Histograma e Polígonos de frequências da probabilidade da vazão total gerada por ENCN ser maior que 1,45 Mbps em até 30 s.	100
4.41	Histograma e Polígonos de frequências da probabilidade da vazão total gerada por CoDel ser maior que 1,45 Mbps em até 30 s.	100
4.42	Posição do motor obtida com a implementação de TCP-Jersey.	101
4.43	Posição do motor obtida com a implementação de E-DCTCP.	101
4.44	ITAE para ENCN comparado com protocolos baseados em ECN.	101
4.45	Vazão para ENCN comparado com protocolos baseados em ECN.	102
4.46	Fila para TCP-Jersey implementado na topologia de rede de comunicação ilustrada na Figura 4.23.	103
4.47	Fila para E-DCTCP implementado na topologia de rede de comunicação ilustrada na Figura 4.23.	103
4.48	Índice de Jain para ENCN comparado com protocolos baseados em ECN.	104
4.49	Banda Estimada utilizando a topologia de rede de comunicação da Figura 3.3.	107
4.50	Fila e Fila Estimada utilizando a topologia de rede de comunicação da Figura 3.3.	108
4.51	W para ANCE, implementada na topologia de rede de comunicação ilustrada na Figura 3.3.	109
4.52	Fila e fila estimada para ANCE, implementada na topologia de rede de comunicação ilustrada na Figura 3.3.	110
4.53	Vazão para ANCE comparada com outras técnicas AQM, implementadas na topo- logia de rede de comunicação daisy-chain da Figura 3.3.	110
4.54	Posição do motor para ANCE implementada na topologia de rede de comunicação ilustrada na Figura 4.23.	112

4.55	ITAE obtido com a implementação de diferentes técnicas AQM, implementadas na topologia de rede de comunicação ilustrada na Figura 4.23.....	112
4.56	Vazão para ANCE comparada com técnicas AQM, implementadas na topologia de rede de comunicação ilustrada na Figura 4.23.....	112
4.57	Índice de Jain para ANCE comparada com técnicas AQM, implementadas na topologia de rede de comunicação ilustrada na Figura 4.23.	113
4.58	Fila para ANCE implementada na topologia de rede de comunicação ilustrada na Figura 4.23	113
4.59	Topologia de rede de comunicação usada para avaliar o ANCE com fluxo Poisson no caminho inverso.	114
4.60	Fila no roteador 1 no sentido do fluxo ANCE e fila estimada pelo transmissor ANCE na presença de congestionamento no caminho inverso.	114
4.61	Fila no roteador 2 no sentido contrário ao fluxo ANCE.	115
4.62	Vazão com e sem congestionamento no sentido contrário ao fluxo ANCE.	115
5.1	Parâmetro da função de transferência da topologia de rede de comunicação da Figura 3.3.	121
5.2	Parâmetro da função de transferência da topologia de rede de comunicação da Figura 3.3.	121
5.3	Atraso relativo no caminho de ida fornecido pelo controlador nominal.	123
5.4	Janela de transmissão W obtida mediante o uso do controlador nominal.	123
5.5	Fila no caminho de ida gerada pelo controlador nominal.	123
5.6	Atraso relativo no caminho de ida.	124
5.7	Fila no caminho de ida.	124
5.8	Diagrama em blocos do controlador adaptativo.	126
5.9	Atraso relativo no caminho de ida.	128
5.10	W	129
5.11	Fila no caminho de ida.	129
5.12	Fila no caminho de ida.	129
5.13	Atraso relativo no caminho de ida.	130
5.14	Janela de transmissão.	130
5.15	Fila no caminho de ida.	131
5.16	Fila no roteador 1 no sentido do fluxo TCP-Puerto-Londero na presença de congestionamento no caminho de retorno.	131
5.17	Fila no roteador 2 no sentido contrário ao fluxo TCP-Puerto-Londero.	132
5.18	Readaptação do parâmetro θ_1 em cada fluxo TCP-Puerto-Londero.	133
5.19	Readaptação do parâmetro θ_2 em cada fluxo TCP-Puerto-Londero.	133
5.20	Fila no roteador 1	133
5.21	Malha de controle.	134
5.22	Ruído de medição colocado na saída da planta.	134
5.23	Possíveis casos para o controlador PI baseado em pacotes.	136

5.24	Posição do motor para o caso no qual o controlador está junto com a planta com perturbação de 3 V aplicado aos 7 s.	137
5.25	Posição do motor com perturbação de 3 V aplicado aos 7 s, compartilhando a mesma topologia de rede com fluxos TCP-Jersey	137
5.26	Posição do motor com perturbação de 3 V aplicado aos 7 s, compartilhando a mesma topologia de rede com fluxos E-DCTCP.	137
5.27	Posição do motor com perturbação de 3 V aplicado aos 7 s, compartilhando a mesma topologia de rede com fluxos ENCN.	138
5.28	Posição do motor com perturbação de 3 V aplicado aos 7 s, compartilhando a mesma topologia de rede com fluxos TCP-Puerto-Londero.	138
5.29	ITAE para diferentes protocolos, implementados na topologia de rede de comunicação ilustrada na Figura 4.23.	138
5.30	Vazão para diferentes protocolos, implementados na topologia de rede de comunicação ilustrada na Figura 4.23.	139
5.31	Índice de Jain para diferentes protocolos, implementados na topologia de rede de comunicação ilustrada na Figura 4.23.	140
5.32	Fila para TCP-Jersey.	140
5.33	Fila para E-DCTCP.	141
5.34	Fila para ENCN.	141
5.35	Fila para TCP-Puerto-Londero.	141
5.36	Topologia de rede de comunicação tipo <i>Dumbbell</i> com um tráfego de fundo compartilhada por três fluxos TCP e um fluxo UDP de uma NCS.	142
5.37	Probabilidade acumulada do ITAE ser maior que 3500,00 rad.s em até 20 s.	143
I.1	Topologia de rede de comunicação daisy-chain com um fluxo TCP.	164
I.2	Conjunto de autômatos que modelam a topologia de rede de comunicação daisy-chain da Figura I.1.	165
I.3	Modelo do transmissor TCP-Reno.	166
I.4	Modelo do canal 1 de ida.	168
I.5	Modelo da fila de dados no caminho de ida, no roteador 1.	170
I.6	Modelo do receptor.	171
I.7	Modelo do canal 2 de ida.	172
I.8	Modelo da fila de dados no caminho de ida, no roteador 2.	174
I.9	Modelo do canal 3 de ida.	175
I.10	Modelo do receptor.	176
I.11	Modelo do canal 3 de retorno.	178
I.12	Modelo da fila no caminho de retorno, no roteador 2.	179
I.13	Modelo do canal 2 de retorno.	180
I.14	Modelo da fila de dados no caminho de retorno, no roteador 1.	182
I.15	Modelo do canal 1 de retorno.	183
II.1	Modelo autômato temporizado para o canal de ida situado entre o transmissor e o roteador 1 utilizando RED.	185

II.2	Modelo autômato temporizado para a fila canal de ida no roteador 1 utilizando CoDel.	188
II.3	Modelo autômato temporizado para realizar a primeira parte do algoritmo utilizando PIE	190
II.4	Modelo autômato temporizado para o canal de ida situado entre o transmissor e o roteador 1 utilizando PIE.....	191
II.5	Modelo autômato temporizado para a Fila canal de ida no roteador 1 utilizando PIE	194
III.1	Automato que modela a planta.	197
III.2	Modelo do controlador	198

LISTA DE TABELAS

2.1	Bits ECN	25
3.1	Descrição das variáveis, funções e constantes utilizadas no modelo do transmissor.....	37
3.2	Descrição das variáveis, funções e constantes utilizadas no modelo do canal 1 de ida.	39
3.3	Descrição das variáveis, funções e constantes utilizadas no modelo da fila de ida no roteador 1.....	40
3.4	Descrição das variáveis, funções e constantes utilizadas no modelo do receptor.....	42
3.5	Valor final do ITAE para os primeiros três cenários de simulação.	56
3.6	Valor final do ITAE para os primeiros os Cenários A e B.....	59
3.7	Valor final do ITAE e da Vazão para diferentes Técnicas AQM.....	64
4.1	Dinâmica de BF para o ENCN.	75
4.2	Descrição das variáveis, funções e constantes utilizadas no modelo da fila de retorno no roteador 1 com a técnica AQM ENCN.....	77
4.3	Vazão obtida na topologia de rede daisy-chain para diferentes técnicas AQM.....	80
4.4	Vazão obtida na topologia de rede daisy-chain para diferentes protocolos aos 10 s de simulação.....	81
4.5	Vazão total e Índice de Jain ao final da simulação.	85
4.6	Vazão total e Índice de Jain ao final da simulação.	89
4.7	Vazão total, índice de Jain e ITAE para diferentes técnicas AQM.	94
4.8	Intervalo de probabilidade do ITAE ser maior que 8727 rad.s em até 30 s.	97
4.9	Intervalo de probabilidade da vazão total ser maior que 1,45 Mbps em até 30 s.	99
4.10	Vazão total, índice de Jain e ITAE para diferentes técnicas AQM atingidos ao final da simulação.	104
4.11	Tabela Vazão obtida aos 10 s de simulação para ANCE comparada com outras técnicas AQM, implementadas na topologia de rede de comunicação ilustrada na Figura 3.3.....	110
4.12	Vazão, índice de Jain e ITAE ao final da simulação para diferentes técnicas AQM implementadas na topologia de rede de comunicação ilustrada na Figura 4.23.	113
5.1	Vazão, índice de Jain e ITAE obtidos ao final da simulação para diferentes protocolos, implementados na topologia de rede de comunicação ilustrada na Figura 4.23.	142
5.2	Intervalo de probabilidade do ITAE ser maior que 3500,00 rad.s em até 20 s.	143

I.1	Descrição das variáveis, funções e constantes utilizadas no modelo do transmissor.....	166
I.2	Descrição das variáveis, funções e constantes utilizadas no modelo do canal 1 de ida.	168
I.3	Descrição das variáveis, funções e constantes utilizadas no modelo da fila de ida no roteador 1.....	169
I.4	Descrição das variáveis, funções e constantes utilizadas no modelo do receptor.....	171
I.5	Descrição das variáveis, funções e constantes utilizadas no modelo do canal 2 de ida.	172
I.6	Descrição das variáveis, funções e constantes utilizadas no modelo da fila de ida no roteador 2.....	173
I.7	Descrição das variáveis, funções e constantes utilizadas no modelo do canal 3 de ida.	175
I.8	Descrição das variáveis, funções e constantes utilizadas no modelo do receptor.....	176
I.9	Descrição das variáveis, funções e constantes utilizadas no modelo do canal 3 de retorno.....	177
I.10	Descrição das variáveis, funções e constantes utilizadas no modelo da fila de retorno no roteador 2.....	179
I.11	Descrição das variáveis, funções e constantes utilizadas no modelo do canal 2 de retorno.....	180
I.12	Descrição das variáveis, funções e constantes utilizadas no modelo da fila de retorno no roteador 1.....	181
I.13	Descrição das variáveis, funções e constantes utilizadas no modelo do canal 1 de retorno.	183
II.1	Descrição das variáveis, funções e constantes utilizadas no modelo do canal 1 de ida com a técnica AQM RED.....	185
II.2	Descrição das variáveis, funções e constantes utilizadas no modelo da fila de ida no roteador 1 com a técnica AQM CoDel.	187
II.3	Descrição das variáveis, funções e constantes utilizadas no modelo da primeira parte do algoritmo PIE.	190
II.4	Descrição das variáveis, funções e constantes utilizadas no modelo do canal 1 de ida com a técnica AQM PIE.....	191
II.5	Descrição das variáveis, funções e constantes utilizadas no modelo da fila de ida no roteador 1 com a técnica AQM PIE.	193
III.1	Descrição das variáveis, funções e constantes utilizadas no modelo do canal 1 de ida com a técnica AQM RED.....	196
III.2	Descrição das variáveis, funções e constantes utilizadas no modelo do canal 1 de ida com a técnica AQM RED.....	198

LISTA DE SÍMBOLOS

Símbolos Latinos

R referência de posição angular (motor) [rad]

Símbolos Gregos

τ Atraso de propagação em sistemas de controle em rede [s]

Grupos Adimensionais

i Contador de autômatos de uma rede de autômatos.

j Contador de autômatos de uma rede de autômatos.

k Contador de sequência de pacotes e amostrar de tempo discreto

Siglas

ACK	Acknowledgement
ANCE	Acknowledge-based Non-Congestion Estimation
AQM	Active Queue Management
CE	<i>Congestion Experienced</i>
CoDel	Control Delay
cwnd	congestion windows
<i>dfp</i>	delay in forward path
ECN	Explicit Congestion Notification
ENCN	Explicit Non Congestion Notification
ECT	<i>ECN-Capable Transport</i>
IAE	Integral do erro absoluto - Integral of Absolute Error
IP	Internet Protocol
ISE	Integral do erro quadrático - Integral of Squared Error
ITAE	Integral do erro absoluto ponderado - <i>Integral Time-weighted Absolute Error</i>
NCS	Networked Control Systems
NS	<i>Nonce Sum</i>
NS-2	Network Simulator 2
NS-3	Network Simulator 3
PI	Proporcional Integral
PIE	Proportional Integral controller-Enhanced
<i>rdfp</i>	relative delay in forward path
<i>rdfpTarget</i>	relative delay in forward path target
RED	Random Early Detection
RTT	Round-Trip delay Time
rwnd	receiver windows
SMC	<i>Statistical Model Checking</i>
Ssthr	Slow Start Threshold
TCP	Transmission Control Protocol
TOS	<i>Type of Service</i>
UDP	User Datagram Protocol
UPPAAL	Ferramenta de software desenvolvida de forma conjunta pela universidade de <i>Uppsala</i> da Suécia e a universidade de <i>Aalborg</i> da Dinamarca

Capítulo 1

Introdução

O desenvolvimento da automação na indústria, nos automóveis e, mais recentemente, na automação predial, e outras aplicações tais como *smart grids* e *smart city*, vem demandando cada vez mais sistemas de controle complexos, capazes de atender com eficiência, precisão e confiabilidade processos distribuídos. Neste sentido, a tendência é a implementação de sistemas de controle em rede, denominados NCS (*Networked Control Systems*) [1, 2, 3, 4, 5, 6, 7, 8].

Ainda, segundo as mesmas referências supracitadas, uma questão chave relativa à implementação NCS é o fato do controle geralmente ser realizado por meio de uma topologia hierárquica, onde os controladores distribuídos executam algumas tarefas de forma individual, baseados em medições locais, e outras a partir de informações e instruções advindas de um controlador central, o qual envia comandos para os diversos controladores distribuídos.

Assim, junto com sua grande utilidade na área de controle, as NCSs trazem consigo novos problemas a serem abordados, o que tem gerado um grande número de pesquisas. Um desses primeiros problemas foi encontrado já nos anos 80 por um grupo de pesquisa da *Halevi Pennsylvania State University* que estudou a implementação de NCSs em aeronaves (nesses trabalhos NCS foram chamadas de sistemas integrados de comunicação e controle). A necessidade de implementar diversos sistemas de controle em um mesmo sistema físico aumentou a quantidade de cabeamento até superar as limitações de peso e tamanho para essas aplicações, surgindo assim a necessidade de utilizar malhas de controle compartilhadas [9, 10, 11, 12, 13, 14, 15]. Também foi abordada a necessidade de desenvolver modelos matemáticos que representassem uma NCS permitindo estabelecer condições para garantir a estabilidade do sistema de controle.

Outros problemas das NCS estão ligados ao fato de que neste tipo de topologia o controlador e a planta podem ficar fisicamente alocados em locais separados e trocarem dados através de uma rede de comunicação formando uma malha de controle em que controlador, sensor e processo estão fisicamente separados. Assim sendo, parâmetros de desempenho de controle, tais como sobrepasso, tempo de acomodação, erro em regime permanente e até mesmo a estabilidade passam a depender de parâmetros de desempenho do sistema de comunicação que faz a conexão entre a planta e o controlador, tais como atrasos, perda e descarte de pacotes, queda de conexão e interferências sofridas no canal. Logo, surge assim a necessidade de se considerar as influências do sistema de

comunicação no projeto de sistemas de controle críticos.

Neste sentido, nos anos 90, foram realizados trabalhos com o objetivo de compensar problemas de atrasos introduzidos pelo sistema de comunicação no desempenho do controlador. A maior parte desses trabalhos abordaram o problema desde um ponto de vista estocástico, realizando uma análise estatística dos atrasos introduzidos pelo sistema de comunicação para logo projetar um controlador adaptado ao problema [16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27].

Nas referências [16, 17, 18, 19, 20] a metodologia utilizada está baseada em teoria de filas, e são utilizadas informações probabilísticas no projeto do controlador mediante o uso de um estimador para estimar o estado futuro da planta e um preditor para calcular o sinal de controle. Enquanto que nos trabalhos liderados por Björn Wittenmark e Johan Nilsson [21, 22, 23, 24, 25, 26, 27, 28] os problemas temporais (atrasos aleatórios e falta de sincronização que aparecem nas NCSs) foram descritos e analisados desde um ponto de vista estocástico. Foram ainda propostas soluções ao problema mediante controladores ótimos estocásticos. Assumindo que o algoritmo de controle dispõe de informações referentes ao instante de tempo em que se gerou a informação a ser utilizada no cálculo da lei de controle. O objetivo foi minimizar uma função de custo assumindo que se dispõe de informações completas acerca do estado da planta e dos atrasos introduzidos pelo sistema de comunicação. Esse grupo de pesquisa também trabalhou com modelagem no espaço de estados para sistema de controle em malha fechada, afetados pela presença de atrasos maiores e menores a um período de amostragem. Os atrasos considerados no modelo foram obtidos como a soma dos atrasos existentes entre sensor e controlador, controlador e atuador e o atraso do controlador no cálculo da lei de controle. Uma descrição detalhada desse modelo encontra-se em [28].

No começo da década de 2000 um grupo de pesquisa liderado por Michael S. Branicky utilizaram esse modelo para desenvolver estudos acerca das regiões de estabilidade para sistemas de controle em malha fechada, considerando a influência da frequência de amostragem e de atrasos maiores e menores a um período de amostragem. Também foram realizados gráficos da região de estabilidade em função desses dois fatores. Dessa região de estabilidade se derivam condições suficientes que, se impostas sobre o período de transmissão, podem garantir a estabilidade para uma faixa de atrasos introduzidos pelo sistema de comunicação. Porém, nessa abordagem, os sensores são orientados a tempo, enquanto que atuadores e o controlador são orientados a eventos. Assim, a taxa de amostragem da lei de controle não pode ser considerada constante [28, 29, 30, 31, 32, 33].

Por outro lado, nessa mesma época, trabalhos realizados na universidade de *Miami* [34, 35, 36, 37, 38, 39] se centraram no estudo de NCSs com atrasos variáveis, mas conhecidos. Uma curiosidade mostrada nesses trabalhos é que grandes atrasos introduzidos pelo sistema de comunicação não são necessariamente nocivos para a estabilidade do sistema de controle. Nesses trabalhos também foram encontradas condições suficientes de estabilidade e foram propostas estruturas de controle baseadas em reguladores distribuídos na rede, compensando atrasos de acesso ao meio. E ainda foram feitos profundos estudos da natureza de atrasos variáveis introduzidos pelo sistema de comunicação e foram analisadas suas propriedades.

Também, com esse mesmo objetivo de estudar e compensar as influências do sistema de comunicação no desempenho do sistema de controle, um grupo de pesquisa da *University of Michigan*

[40, 41, 42, 43, 44, 45, 46, 47, 48, 49], avaliou algumas redes de comunicação específicas utilizadas na automação industrial tais como *Ethernet*, *ControlNet* e *DeviceNet*, abordando a problemática de atrasos e desenvolvendo critérios matemáticos que permitam ao projetista do sistema de controle analisar a influência de atrasos na qualidade do controle. Também, foram propostos critérios para a seleção do período de amostragem. Uma interessante curiosidade mostrada nesses trabalhos é o fato de que, diferentemente das malhas de controle convencional, em uma NCS a redução do período de amostragem poderá degradar o desempenho do sistema de controle já que aumentará o tráfego na rede de comunicação e, conseqüentemente, poderá aumentar atrasos e perdas de pacotes. Em [48, 49] também foram desenvolvidos estimadores para determinar a informação perdida caso ocorram perdas de pacotes na rede de comunicação.

Outras propostas para compensar os efeitos introduzidos pelo sistema de comunicação são encontradas em trabalhos realizados pelo grupo de pesquisa liderado por Mo-Yuen Chow. Esses trabalhos abordaram essa problemática tanto do ponto de vista do projeto de controladores, como do ponto de vista da estratégia de acesso ao meio em sistemas de comunicações compartilhados. Referente a projetos de controladores foram propostas estruturas de controle baseadas em lógica nebulosa capazes de compensar as influências de atrasos introduzidos pelo sistema de comunicação. Para essa finalidade foi empregado um regulador *Proporcional-Integral* (PI) cujos parâmetros são ajustados externamente a partir de funções de pertinência baseadas no erro percebido pelo controlador [50, 51, 52, 53, 54, 55, 56, 57].

Referente às estratégias de acesso ao meio no sistema de comunicação compartilhado, foi utilizado o conceito de *middleware* no qual todas as aplicações que compartilham o mesmo sistema de comunicação colaboram no esforço de controle. A metodologia proposta consiste em modificar a ação de controle gerada pelo controlador em função das condições de tráfego observadas, isso se traduz em adaptar a lei de controle às condições de utilização do sistema de comunicação. Ou seja, foi proposta uma metodologia de adaptação do controlador baseado em métricas de qualidade de serviço (QoS) do sistema de comunicação. Nessa abordagem, se assume que o controlador é capaz de medir as condições de tráfego no sistema de comunicação, e se determinada QoS não está sendo atendida o controlador adaptará seus parâmetro com o objetivo de conseguir melhores métricas [58, 56, 59, 60, 61, 62].

Alguns estudos mais recentes abordaram a problemática de compensar as influências do sistema de comunicação no desempenho do sistema de controle mediante o projeto de controladores quadráticos ótimos dependentes de atrasos [63, 64, 65, 66, 67]. Para NCSs com atrasos estocásticos o projeto de controladores quadráticos ótimo tem sido desenvolvido tanto para sistemas nos quais o controlador consegue calcular o atraso na rede [68, 67], como para sistemas nos quais o atraso é desconhecido pelo controlador [69].

1.1 Interdisciplinarietà em uma NCS

Em uma NCS, o desempenho do sistema de controle depende do sistema de comunicação, ou seja, há uma confluência entre controle e tecnologias de informação e comunicação. Essa confluência

entre domínios torna o projeto de ambos complexos. Conforme as referências bibliográficas citadas na seção anterior, o sistema de controle pode ser projetado para funcionar com as limitações do sistema de comunicação. Mas, por outro lado, também o sistema de comunicação pode ser projetado para atender os requisitos do sistema de controle.

Nessa abordagem, é possível definir compensações entre parâmetros de desempenho e projetar diferentes soluções. Por exemplo, estudar os limites de estabilidade de um sistema de controle em função de diferentes protocolos de comunicação. Ou então, estimar o pior caso para atrasos e perdas de pacotes e projetar um controle robusto a essas condições. Porém, essa tática pode ser pessimista e trabalhar com um uso ineficiente dos recursos. Assim, visando abordar essa problemática de uma maneira mais eficiente, alguns trabalhos mais recentes propuseram uma metodologia de *co-design* na qual os requisitos do projeto de controle reúnem restrições dos diferentes domínios.

Assim, em [70] é proposto um controlador quadrático ótimo para um problema de *co-design* entre canal e controlador. Em, [71] foi proposto um *co-design* para redes de automação com o objetivo de ter um controle confiável na presença de imperfeições temporais. Em [72], a metodologia de *co-design* utilizada consiste em projetar um controle chaveado que faz um chaveamento entre diferentes estratégias de controle em função do atraso medido na rede. Esse chaveamento ainda é feito com o objetivo de reduzir o tráfego no sistema de comunicação e assim reduzir atrasos. Visando, por um lado, conseguir que o sistema de controle atinja um desempenho desejado e, por outro lado, fazer um uso eficiente dos recursos do sistema de comunicação.

Em [73], uma metodologia de *co-design* é projetada para fazer um melhor aproveitamento da banda disponível no sistema de comunicação. Assim, um controlador orientado a evento foi modelado junto com um gerador de eventos como um sistema chaveado de tempo discreto com parâmetros incertos, de maneira tal que as transmissões de dados entre planta e controlador só ocorrem quando for realmente necessário. O *co-design* foi projetado de tal forma que garante que o sistema de malha fechada seja globalmente assintoticamente estável, já que o sistema é baseado em um agendamento de eventos formulados como um problema de Desigualdades Matriciais Lineares (LMI) resolvido para garantir um eficiente uso dos recursos de comunicação.

Uma metodologia de *co-design* baseada em um projeto de controlador com transmissões agendadas mediante um período de amostragem variante também foi proposta em [74]. Onde a NCS trabalha com um sistema de detecção de falhas do sistema de comunicação mediante o qual faz ajustes no período de amostragem. Metodologia parecida a essa também foi utilizada em [75] onde foi proposto utilizar um filtro baseado em lógica nebulosa para detetar falhas no sistema de comunicação, no qual o ganho do filtro e o ganho do sistema de controle são projetados simultaneamente.

1.2 Sistemas de Controle via Internet

Embora em sua origem a Internet foi criada como uma rede de computadores, na atualidade ela se tornou uma rede capaz de interconectar diversos tipos de usuários, dispositivos, casas, edifícios, plantas industriais, automóveis, hospitais, escolas, evoluindo assim para um novo paradigma co-

nhecido como Internet das Coisas (IoC), a qual é literalmente uma rede com diversos objetos físicos ou coisas embutidas mediante dispositivos eletrônicos, *softwares*, sensores, atuadores e conexões de redes que permitem que esses objetos colem e troquem dados entre si.

Este paradigma da Internet das Coisas gerou um crescente número de pesquisas em diversas áreas, que estudam, também, o uso de NCS através da Internet, ou seja, o controle remoto de objetos através de uma infraestrutura de rede já existente. Este tipo de malha de controle vem sendo utilizada em diversas aplicações, entre elas, controle de veículos robóticos [3], controle de braços robóticos em cirurgias remotas [76, 5], e outras aplicações da robótica na medicina [4], controle de cargas, geradores distribuídos e gerenciamento geral de sub-redes em aplicações de *smart grid* [2, 6, 7] e automação predial [8].

Uma das características destas NCS é lidar com longos atrasos ponta a ponta [77, 78] dado que controlador e planta podem estar separados por grandes distâncias, e os parâmetros do sistema de comunicação, tais como atrasos, erros de transmissão, perdas e descarte de pacotes passam a depender também das características da Internet, tais como, protocolos de transporte de dados, algoritmos de gestão de filas, algoritmos de roteamento e interferências de diversos outros fluxos que compartilham o mesmo meio de comunicação.

Logo, os parâmetros de qualidade do sistema de controle também dependerão dessas características, gerando assim novos desafios de pesquisas que podem estar voltadas tanto para o lado do projeto de controladores quanto para o lado da rede de comunicação.

O presente trabalho segue essa segunda linha de pesquisa, ou seja, ao invés de atuar no projeto de controladores, se atuará no sistema de comunicação, mais especificamente nos protocolos de transporte de dados e nos algoritmos AQM (*Active Queue Management*) visando reduzir os efeitos da Internet sobre o desempenho de sistemas de controle que a utiliza como meio de comunicação entre controlador e planta, tudo sem prejudicar outros fluxos genéricos que possam eventualmente estar compartilhando a mesma topologia de rede de comunicação.

Assim sendo, ao invés de seguir abordagens prévias e inserir entre controlador e planta taxas aleatórias ou constantes de atrasos e perdas de pacotes, no presente trabalho serão realizadas simulações para determinadas topologias de rede de comunicação e determinadas condições de tráfego nas quais essas taxas serão medidas, e seus efeitos poderão ser observados e estudados.

1.3 Análises do atraso de resposta mediante *Model Checking*

Para estudar a influência do sistema de comunicação no sistema de controle em uma NCS, alguns trabalhos [79, 80, 81, 82, 71, 83, 84, 85] usam o conceito de *atraso de resposta*, que é o tempo que ocorre entre a ocorrência de um evento no processo e a ocorrência da correspondente resposta do sistema a tal evento. Note que com essa definição, o atraso de resposta não é afetado apenas por atrasos de propagações de sinais. Mas, perdas ou descartes de pacotes, perdas de conexões, interferências externas, tempos de espera nas filas, e vários tipos de problemas que ocorrem no sistema de comunicação se verão refletidos na resposta temporal.

E, além de capturar a influência do sistema de comunicação, o *atraso de resposta* também captura a influência do sistema computacional, demora do controlador em capturar informações e calcular a lei de controle. Assim, o *atraso de resposta* geralmente é de natureza estocástica, contendo em geral combinações de sinais determinísticos e estocásticos.

Tradicionalmente, o estudo do *atraso de resposta* é realizado mediante métodos analíticos [79, 82] ou simulação [81]. Essas abordagens, em geral, procuram obter os limites das flutuações temporais ou então estimar comportamentos para casos específicos e particulares cobrindo apenas um limitado número de comportamentos temporais possíveis [86]. Porém, esses métodos podem ser insuficientes para explorar todo o espaço de estados possíveis que um sistema real, medianamente complexo possa apresentar.

Assim sendo, na última década, alguns trabalhos propuseram analisar o desempenho de NCSs mediante a ferramenta *model checking* [87, 86, 88], na qual é possível quantificar a probabilidade que o sistema em estudo atinja determinados estados [89]. Em uma NCS, *model checking* pode ser usado tanto para checar estados do sistema de comunicação, por exemplo, a probabilidade que o atraso possa ser maior a um determinado limiar, quanto do sistema de controle, por exemplo, a probabilidade do erro ou da integral do erro ser maior a um determinado valor admissível, quanto do próprio processo a ser controlado, por exemplo, a probabilidade ou até a possibilidade de ocorrer derramamento em um sistema de controle de nível de líquidos.

De forma simplificada, o *model checking* consiste na exploração exaustiva de todo o espaço de estados gerado pelo sistema, procurando comportamentos que não satisfazem alguma propriedade previamente especificada. A aplicação clássica da técnica, consiste em modelar o sistema (utilizando alguma linguagem de modelagem), e especificando no modelo alguma lógica do comportamento temporal, e logo sobre o modelo, aplicar o algoritmo *model checking* usando alguma ferramenta que, de forma automática comprove se o sistema satisfaz determinadas propriedades.

Para aplicar a ferramenta *model checking*, por um lado o sistema a ser analisado deve ser simplificado para reduzir seu tamanho, de tal forma que possa ser representado na memória da máquina. Para atingir esse objetivo, ou seja, para otimizar a representação do espaço de estados, diferentes técnicas foram desenvolvidas [90, 91, 92, 93, 94, 95].

Essas técnicas possibilitaram o uso de *model checking* no análise de sistemas cada vez maiores e mais complexos. Por outro lado, para construir o espaço de estados é necessário que o sistema esteja completo, ou seja, o modelo deve englobar todas as possibilidades, tarefa que pode ser complicada, pois geralmente o sistema a ser modelado pode estar inserido em um entorno muito complexo que geralmente se comporta de forma imprevisível. Porém, para modelar seu comportamento podem ser utilizados modelos capazes de descrever seu comportamento real com o máximo de confiabilidade.

Uma das ferramentas de verificação que foi recentemente utilizada com êxito na análise de processos industriais é a ferramenta UPPAAL que foi desenvolvida de forma conjunta pela universidade de *Uppsala* da Suécia e a universidade de *Aalborg* da Dinamarca. Assim, em [96], o UPPAAL foi utilizado junto com a ferramenta *simulink* para sintetizar o controlador de um termostato. Em [88], o UPPAAL foi utilizado para modelar, simular e verificar a resposta temporal

de uma rede de automação.

Em [97], a ferramenta *Estatistical Model Checking* (SMC) do UPPAAL, foi utilizada para verificar comportamentos do protocolo de roteamento *ZigBee* aplicado em uma sub-rede de uma *smart grid*, checando prevenção de colisões e detecção de possível *deadlock* no protocolo previamente modelado mediante autômatos temporizados. Em [98], o padrão de comunicação *Spacewire*, foi modelado no UPPAAL mediante autômatos temporizados e logo foram verificadas diversas propriedades temporais do modelo mediante a ferramenta SMC. Em, [99] o UPPAAL foi utilizado para testar as propriedades temporais e segurança de funções em diagramas de blocos escritas como códigos para controladores lógicos programáveis aplicados em controle ferroviário.

1.4 Objetivos do trabalho

Os objetivos desse trabalho são:

1. Analisar como o desempenho de um sistema de controle através da Internet depende do comportamento e da natureza de outros fluxos presentes na rede de comunicação.
2. Estudar e comparar o desempenho de diferentes metodologias AQM e TCP (*Transmission Control Protocol*) aplicadas em topologias de rede que suportem fluxos de controle UDP e Dados TCP/IP compartilhados.
3. Propor novas metodologias AQM e TCP capazes de fornecer um bom desempenho tanto para NCSs via Internet, quanto para outros fluxos genéricos que compartilham a mesma topologia de rede de comunicação.
4. Desenvolver um modelo de redes de comunicação junto com seus fluxos, protocolos e algoritmos que possibilite fazer verificações estatísticas do desempenho de NCSs via Internet.

1.5 Metodologia

Para atingir esses objetivos, foi utilizada a ferramenta de *software* UPPAAL, na qual primeiramente foi desenvolvida uma metodologia para modelar, simular e verificar topologias de redes de comunicação junto com seus fluxos, protocolos e algoritmos. O modelo do sistema de comunicação foi realizado mediante autômatos temporizados baseados em uma linguagem TCTL (*Timed Computation Tree Logic*), e facilitou fazer verificações estatísticas do desempenho de diferentes protocolos e algoritmos AQM por meio do módulo SMC do UPPAAL.

Nesta metodologia de modelagem desenvolvida, ainda foi possível, simular de forma conjunta fluxos de controle UDP (*User Datagram Protocol*) de uma NCS com outros fluxos TCP/IP genéricos que compartilham a mesma topologia de rede de comunicação.

O desempenho dos fluxos de controle UDP foram avaliados mediante o ITAE (*Integral Time-weighted Absolute Error*) enquanto que o desempenho dos fluxos de dados TCP/IP foram avaliados mediante a vazão média e a justiça.

1.6 Contribuições

As contribuições desse trabalho foram:

1. Foi desenvolvida uma metodologia para modelar, simular e verificar redes de comunicação junto com seus protocolos e algoritmos, na qual:
 - (a) Foi possível analisar mediante simulações e verificações estatísticas o desempenho de sistemas de controle através da Internet.
 - (b) Encontrou-se a existência de influências mútuas entre fluxos de controle UDP e Dados TCP/IP que compartilham a mesma topologia de redes de comunicação.
 - (c) Observou-se que protocolos de transporte de dados e técnicas AQM implementadas nos roteadores da Internet afetam ambos tipos de fluxos, ocorrendo uma compensação entre o ITAE de sistemas de controle em rede (fluxos UDP) e a vazão dos fluxos TCP/IP genéricos.
 - (d) Identificou-se o indesejado fenômeno de *bufferempty*, produzido por metodologias que visam corrigir o indesejado fenômeno de *bufferbloat*.
2. Visando solucionar o fenômeno de *bufferempty*, foi proposta uma metodologia denominada ENCN (*Explicit Non-Congestion Notification*), que combina técnicas AQM com TCP usando notificação explícita de não congestionamento. Que permitiu melhorar tanto a vazão e a justiça para os fluxos TCP/IP quanto o ITAE para a NCS (fluxo UDP).
3. Com o objetivo de fazer gestão de filas ponta a ponta (sem utilizar recursos para fazer notificações explícitas de congestionamento e/ou não congestionamento) foi desenvolvida uma metodologia denominada ANCE (*Acknowledge-based Non-Congestion Estimation*), que basicamente atua de forma análoga ao ENCN. Porém, sua ação é feita em função de uma estimativa do tamanho da fila ao invés de basear-se em notificações explícitas de não congestionamento. ANCE conseguiu fornecer um desempenho próximo ao ENCN, porém, por estar baseada no RTT apresentou pobre desempenho na presença de congestionamento no caminho de retorno.
4. Visando superar as limitações de ANCE foi proposto o protocolo TCP-Puerto-Londero que, diferentemente de ANCE, atua na janela de transmissão em função do atraso relativo no caminho de ida ao invés de utilizar o RTT. Apesar de não necessitar recursos para fazer notificações explícitas de congestionamento e/ou não congestionamento, o TCP-Puerto-Londero, superou as limitações de ANCE e forneceu um desempenho comparável ao ENCN, superando o TCP-Jersey e o E-DCTCP em termos de vazão e justiça para os fluxos TCP/IP, e ITAE para a NCS (fluxo UDP).

1.7 Resultados obtidos

O desempenho das metodologias desenvolvidas neste trabalho (ENCN, ANCE e TCP-Puerto-Londero) foram comparados com as técnicas AQM Drop Tail, RED, CoDel e PIE implementadas junto com o TCP-Reno, e com os protocolos de transporte de dados baseados em ECN TCP-Jersey, DCTCP e E-DCTCP (todos modelados no UPPAAL). Resultados de simulações verificações estatísticas mostram que ENCN, ANCE e TCP-Puerto-Londero da mesma forma que CoDel e PIE evitam o indesejado fenômeno de *bufferbloat* (crescimento excessivo do tamanho da fila) [100, 101], assim, essas técnicas fornecem um bom desempenho para sistemas de controle em rede (NCS) no que se refere ao ITAE, já que ao manter a fila pequena reduzem o atraso ponta a ponta, o que é um aspecto crítico para as NCSs.

Neste sentido, ENCN apresentou o melhor desempenho, reduzindo o ITAE do sistema de controle (fluxo UDP) em 31,6 %, 31,54 %, 10,78 %, 0,52 % e 16,68 % comparado com RED, CoDel, PIE, TCP-Jersey e E-DCTCP, respectivamente. Por outro lado, ANCE, apresentou pior desempenho que ENCN mas melhor desempenho que RED, CoDel e PIE, apesar de usar menos recursos que essas metodologias (devido a que ANCE não utiliza bits para fazer notificação explícita de congestionamento). Porém, ANCE apresentou pobre desempenho na presença de congestionamento no caminho de retorno. O TCP-Puerto-Londero superou essas limitações do ANCE e apresentou um desempenho ligeiramente melhor que o ENCN em termos do ITAE, apesar de tampouco necessitar de bits para fazer a notificação explícita de congestionamento.

Por outro lado, ENCN, apresentou melhor estabilidade do tamanho da fila que os algoritmos AQM RED, CoDel e PIE, e que os protocolos de transporte de dados TCP-Jersey, DCTCP e EDCTCP. Assim, além de manter a fila em torno de valores pequenos, o ENCN, também conseguiu evitar o indesejado fenômeno de *bufferempty* (um fenômeno identificado neste trabalho e que ocorre quando as técnicas AQM freiam em demasiado os transmissores, ou então quando os transmissores reagem de forma exagerada diante de uma percepção de congestionamento, e conseqüentemente, durante alguns períodos de tempo as filas em todos os roteadores ficam vazias). Ao evitar esse fenômeno, o ENCN melhorou a vazão dos fluxos TCP/IP em 12 %, 2,32 %, 14,7 %, 32 % e 2,9 % comparado com RED, CoDel, PIE, TCP-Jersey e E-DCTCP, respectivamente.

Neste último aspecto, vale a pena destacar que Drop Tail também possui boa capacidade para evitar o indesejado fenômeno de *bufferempty* e assim fornece melhor vazão que RED, CoDel e PIE. Porém, Drop Tail produz altas taxas de perdas de pacotes e conseqüentemente fornece pior vazão que ENCN.

Além disso, Drop Tail somente atua quando o *buffer* estiver cheio, assim sendo, constantemente provoca o indesejado fenômeno de *bufferbloat*, e conseqüentemente fornece maiores atrasos ponta a ponta, e por conseguinte, pior desempenho para sistemas de controle em rede que as outras técnicas simuladas.

Os resultados obtidos por simulações também foram refletidos nas verificações estatísticas do modelo, feitas mediante o módulo SMC do UPPAAL.

Resultados atuais geraram até o momento cinco artigos, entre eles [2, 102], um terceiro já

submetido e outros dois a serem submetidos em breve para periódicos internacionais.

1.8 Apresentação do manuscrito

O restante deste trabalho está estruturado da seguinte forma. No Capítulo 2 é realizada uma descrição das NCSs seus parâmetros e variáveis, assim como uma revisão acerca de algoritmos TCP e técnicas AQM encontradas na bibliografia. No terceiro Capítulo é abordada a modelagem de topologias de redes de comunicação, algoritmos TCP, técnicas AQM e NCS no UPPAAL. No Capítulo 4 é apresentada a técnica AQM proposta, chamada ENCN, a qual ainda é comparada em diferentes cenários de simulação com outras técnicas AQM encontradas na literatura. Neste Capítulo também são apresentadas abordagens de gestão de fila ponta a ponta e a técnica ANCE desenvolvida. No Capítulo 5 é apresentado o TCP-Puerto-Londero e simulações e verificações usando a ferramenta SMC do UPPAAL. Finalmente no Capítulo 6 são apresentadas as conclusões, e propostas para trabalhos futuros.

Capítulo 2

Fundamentos - Redes com Fluxos TCP e NCS Compartilhados

2.1 Introdução

No presente capítulo se estudarão questões acerca de NCSs, sendo o enfoque principal a implementação de sistemas de controle através da Internet. Assim, serão abordadas questões relativas ao tráfego de dados através da Internet e seus efeitos no desempenho de sistemas de controle realizados através desse meio. Primeiramente, serão descritos os parâmetros gerais de uma NCS, e como esses parâmetros dependem de questões próprias da Internet, como presença de diversos fluxos, técnicas AQM e tempos de propagação e bandas dos canais.

Uma NCS é um sistema de controle no qual a malha de controle é fechada através de uma rede de comunicação, ou seja, controlador e planta (processo a ser controlado) estão alocados em locais separados e trocam informações mediante transferências de pacotes através de uma rede de comunicação [10, 12, 28, 51, 31, 32, 103].

A Figura 2.1 ilustra uma configuração básica de uma NCS, na qual uma planta com função de transferência G_p e um controlador com função de transferência G_c trocam dados através de um sistema de comunicação. Um conjunto de sensores medem o sinal de saída da planta $y_p(t)$ e o transmitem para o controlador. Logo o controlador recebe essas medições $y_p(t)$ e as usa como seu sinal de entrada $u_c(t)$ para logo calcular a lei de controle $y_c(t)$, a qual é enviada para a planta através de um canal de comunicação de retorno. Então, o módulo de atuadores presente na planta recebe este sinal $y_c(t)$ e o utiliza como seu sinal de entrada $u_p(t)$ a fim de manter o estado da planta em um determinado valor desejado.

Essa transmissão pode ser de natureza periódica ou eventual [30, 74, 73], ou seja, se o sensor é orientado a tempo fará uma medição e transmissão dos valores de $y_p(t)$ a cada h segundos, onde h é o período de amostragem (veja Figura 2.2).

No caso em que o sensor seja orientado a eventos, $y_p(t)$ é medida e enviada ao controlador após a ocorrência de um determinado evento físico, normalmente quando alguma variável de estado

atinge determinado limiar, neste caso geralmente é necessário um observador local.

Também o controlador pode ser orientado a tempo ou a eventos. No caso de ser orientado a tempo, calcula a lei de controle a cada h segundos, e no caso de ser orientado a eventos geralmente o evento é a chegada de um sinal de medição do estado da planta, assim a lei de controle é calculada imediatamente após a recepção de $y_p(t)$.

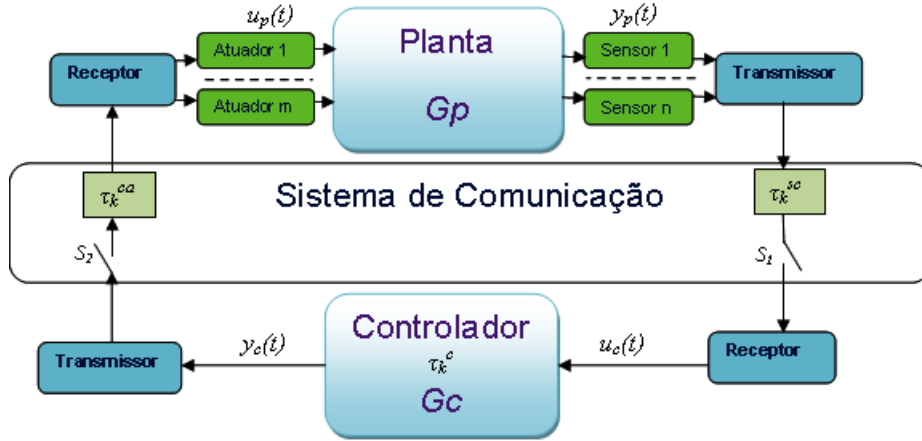


Figura 2.1: Networked Control System (parâmetros e variáveis).

A decisão de empregar um sistema de controle orientado a tempo ou a eventos depende da estratégia de controle a ser implementada, e dos dispositivos que são utilizados. Por exemplo, sensores orientados a eventos possuem a vantagem de reduzir o fluxo na rede, já que permitem evitar a transmissão de pacotes nos instantes em que os estados da planta se encontrem dentro de uma taxa de valores desejados, e enviá-los somente quando estejam fora dessa taxa. Essa metodologia pode ser aplicada quando se deseja reduzir o fluxo de dados no sistema de comunicação [74, 73, 75].

Conforme ilustra a Figura 2.2, os canais de comunicação de ida e de retorno que conectam planta com controlador e controlador com a planta, respectivamente, podem inserir atrasos e sofrer perdas de pacotes. Assim, em uma NCS podemos classificar as seguintes componentes de atrasos:

- Atraso entre sensor e controlador (τ_k^{sc});
- Atraso entre controlador e atuador (τ_k^{ca});
- Atraso do controlador no cálculo da lei de controle (τ_k^c).

Por outro lado, no sistema de comunicação, normalmente podem haver perdas de pacotes, que podem ocorrer por interferências nos canais, queda de conexão, saturação das filas nos *buffers* dos roteadores. E no caso de sistemas de controle através da Internet pacotes ainda podem ser descartados por algoritmos de gestão de filas [104, 105, 106, 100, 101]. Assim, as chaves S_1 e S_2 representam a possibilidade de perdas de pacotes enviados por sensor e controlador, respectivamente. Deste modo, quando estão fechados significa que os pacotes atingem seu destino, caso contrário significa que os pacotes são perdidos. Embora existam aplicações nas quais o controlador

é sobredimensionado e esses atrasos e perdas de pacotes não afetam o desempenho do sistema de controle em rede, também existem várias outras aplicações onde essas questões são muito relevantes, principalmente em controle de processos críticos, como questões relativas a segurança em uma *smart grid*, por exemplo [103, 97, 6, 7].

2.2 Troca de informação em uma NCS

Considere o esquema de controle com taxa de amostragem constante ilustrado na Figura 2.2. O controlador recebe a medição do sensor no instante $l_k^c h + \tau_k^{sc}$, onde o índice $k \in N$ representa o tempo de amostragem do k -ésimo pacote medido. Então, o controlador calcula a lei de controle $y_c(t)$, e o envia ao atuador no instante $l_k^c h + \tau_k^{sc} + \tau_k^c$. Finalmente o atuador recebe o k -ésimo sinal de controle no instante $l_k^a h + \tau_k$, sendo, τ_k a soma do atraso entre sensor e controlador (τ_k^{sc}) mais o atraso entre controlador e atuador (τ_k^{ca}), mais o atraso do controlador no cálculo da lei de controle (τ_k^c). Em algumas redes de comunicação, pacotes podem ser roteados por diferentes caminhos e podem chegar ao controlador fora de ordem. Neste caso, somente a informação mais recente é interessante e o pacote mais antigo, que contém informação passada pode ser descartado.

Por outro lado, no caso de ocorrer perdas ou descarte de pacotes (perdas acontecem devido a interferências externas e descartes são causados por algoritmos tais como técnicas AQM implementadas nos roteadores) normalmente é mantido o último valor de $u_p(t)$ no atuador ou o último valor de $u_c(t)$ no controlador [9, 51, 107].

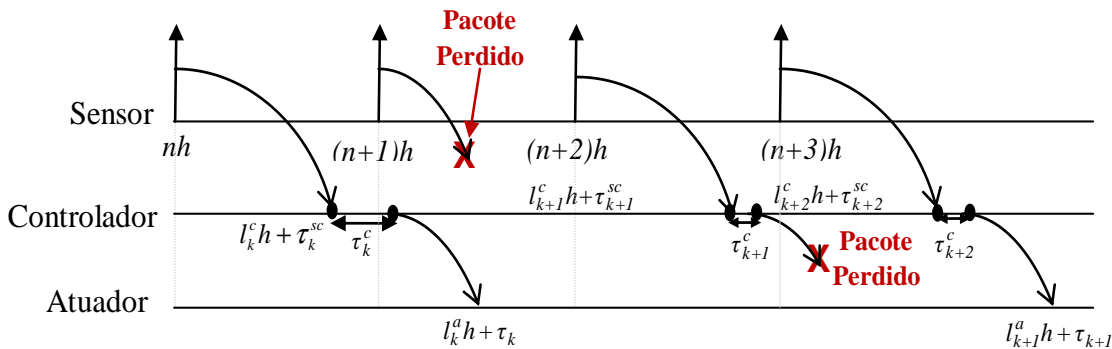


Figura 2.2: Diagrama temporal de atrasos na rede.

2.3 Parâmetros de desempenho em sistemas de controle

Conforme visto no Capítulo anterior, o sistema de comunicação utilizado para conectar a planta ao controlador pode apresentar diferentes características e tecnologias dependendo da natureza da planta, dos processos a serem controlados e da distância que separa o controlador da planta.

No presente trabalho será focado o estudo de processos a serem controlados através da Internet, onde o principal objeto de estudo consiste em analisar como o desempenho do sistema de controle depende do comportamento e natureza de outros fluxos presentes na rede. Essa depen-

dência é função dos algoritmos de gestão de filas implementados nos roteadores da Internet e dos protocolos de transporte de dados a serem utilizados, que será o tema a ser abordado em seções seguintes.

Um controle ideal seria aquele capaz de fazer com que os estados da planta sempre fossem iguais aos valores das referências (valor desejado). Porém, em sistemas de controle reais, normalmente o controlador, no esforço para levar os estados da planta até os valores desejados, apresenta uma dinâmica a determinadas referências. Essa resposta normalmente não é exatamente igual ao valor desejado, e sua qualidade pode ser caracterizada por uma série de parâmetros.

A título de exemplo, na Figura 2.3, é apresentada uma resposta genérica a uma referência, ou seja, o estado da planta parte de zero e se pretende levá-lo a um valor R (referência) constante, na qual são ilustradas os principais parâmetros utilizados para avaliar a qualidade de um controlador. Conforme descritos de forma sintetizada a seguir:

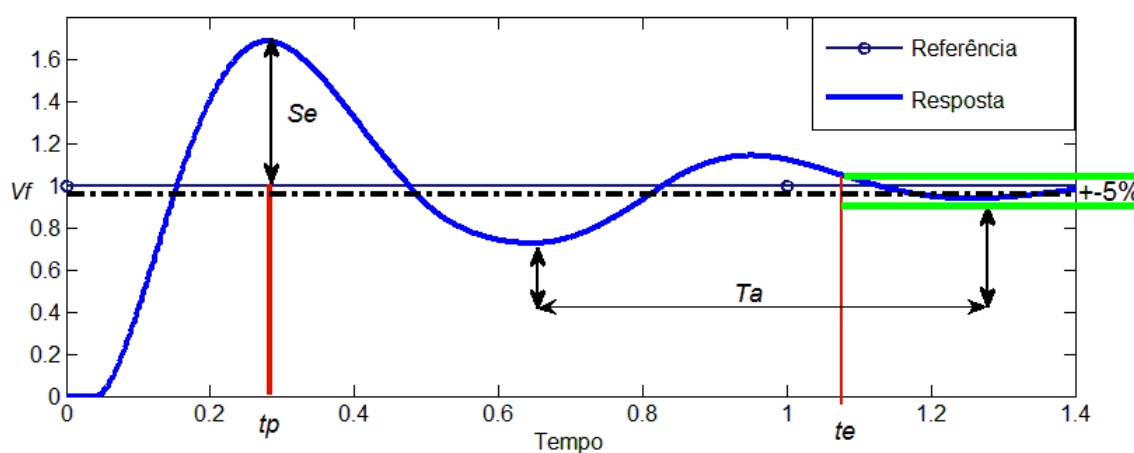


Figura 2.3: Parâmetros da resposta de sistema de controle genérico ao degrau unitário.

- V_f , corresponde ao valor final ao que tende a resposta do controlador;
- Se é a sobre-elevação máxima (ou *overshoot*), e corresponde à diferença entre o valor máximo e o valor final da resposta, geralmente medida como porcentagem ou fração do valor final.
- tp é o tempo de pico, e corresponde ao tempo que a resposta do controlador leva para atingir seu valor máximo;
- Ta é o período das oscilações amortecidas que a resposta apresenta em torno do valor final. Esse parâmetro somente se define se existem oscilações e essas forem periódicas;
- te é o tempo de estabelecimento, e corresponde ao tempo em que a resposta do controlador leva para entrar em uma determinada margem em torno do valor final. Geralmente, essa margem é definida em porcentagens do valor final, e geralmente existe uma compensação entre te e Se , ou seja, quando se deseja projetar um controlador que forneça pequena Se , geralmente se terá um maior te como consequência e vice-versa.

Topologias de redes de comunicação utilizadas em NCSs inserem atrasos e há perdas de pacotes e conseqüentemente provocam distorções na resposta do sistema de controle [30, 108, 1, 107, 109]. Para comparar a qualidade de controle conseguida em diferentes topologias de redes de comunicação deve ser utilizado algum índice de desempenho, que quantifique o erro, ou seja, que permita comparar diferentes projetos de NCS.

Para essa finalidade, geralmente se utilizam critérios baseados na integral do erro. Porém para analisar controle no espaço discreto, as integrais são expressas na forma de somatório, e o erro na forma de erro discreto, $e(k)$, sendo k o tempo discreto. Alguns desses índices de desempenho são descritos de forma sintetizada a seguir.

- Integral do erro quadrático (ISE *Integral Square Error*): Descrito em tempo discreto pela Equação (2.1), e se utiliza para penalizar respostas com erros grandes devido ao termo quadrático, ou seja, quantifica com maior peso erros iniciais. Portanto, não é indicado para sistemas oscilatórios. Assim,

$$ISE = \sum_{k=1}^n e^2(k). \quad (2.1)$$

- Integral do erro absoluto (IAE *Integral Absolute Error*): Descrito em tempo discreto pela Equação (2.2), e se utiliza quando se deseja penalizar erros grandes ou pequenos com o mesmo peso, mas sua desvantagem é que esses erros devem ser medidos com grande precisão. Logo,

$$IAE = \sum_{k=1}^n |e(k)|. \quad (2.2)$$

- Integral do erro absoluto ponderado (ITAE *Integral Time-weighted Absolute Error*): Descrito em tempo discreto pela Equação (2.3), e se utiliza quando se deseja penalizar erros que persistem por um longo tempo, já que não apenas o erro instantâneo é levado em consideração. Portanto,

$$ITAE = \sum_{k=0}^n k h |e(k)|. \quad (2.3)$$

Esses indicadores de desempenho dependem primeiramente do projeto do controlador e da adequada sintonia de seus parâmetros (por exemplo, a correta escolha das constantes de proporcionalidade e de integração para um controlador proporcional integral), mas também dependem de atrasos e perdas de pacotes introduzidos pela rede de comunicação sobre a qual controlador e planta trocam informações.

Então, quando essa troca de dados for feita através da Internet, esses indicadores de desempenho passarão a depender de características próprias da Internet. Uma característica fundamental da Internet está no fato de que diversos tipos de fluxo compartilham o mesmo meio de comunicação, e esses fluxos são controlados por protocolos de transporte de dados tais como UDP e diversas variantes do TCP. Portanto, a seguir, será feita uma breve descrição de como é feito o transporte de dados na Internet.

2.4 Transporte de dados na Internet

Um dos principais protocolos utilizados para o transporte de dados na Internet é o protocolo TCP. Essencialmente, o TCP procura estabelecer uma conexão ponto a ponto entre transmissor e receptor, e controla para que todos os pacotes cheguem ao destino corretamente, fazendo retransmissões caso seja necessário.

Para atender essa finalidade, basicamente é associado um número de sequência a cada pacote e o receptor confirma a chegada de pacotes enviando um pacote de confirmação ACK (*acknowledgement*) para o remetente, indicando o número de sequência do próximo pacote a ser recebido. O ACK pode ser enviado imediatamente após a recepção de um pacote ou após um determinado tempo de espera confirmando a chegada de uma sequência de pacotes. Então, o remetente pode saber que pacotes foram entregues com sucesso e quais pacotes o receptor está esperando receber.

A fim de controlar congestionamento na rede, o TCP utiliza uma janela de transmissão W , que consiste no número máximo de pacotes que podem ser transmitidos antes de receber um ACK, ou seja, o número máximo de pacotes pendentes de confirmação de recepção que podem estar na rede. Para o cálculo dessa variável, inicialmente, remetente e receptor concordam um tamanho de janela que será utilizado, a fim de evitar que o remetente sature o receptor. Essa janela imposta pelo receptor é denominada *rwnd* (*receiver window*). Porém, as condições de tráfego na rede de comunicação nem sempre permitem o uso total dessa janela.

Então, além da janela imposta pelo remetente, existe outra janela denominada janela de congestionamento *cwnd* (*congestion window*); e a efetiva janela W que será utilizada pelo remetente será calculada como o mínimo dessas janelas, isto é

$$W \leftarrow \min(rwnd, cwnd). \quad (2.4)$$

Portanto, uma das formas mais básicas de controlar o congestionamento na rede consiste em atuar sobre *cwnd*. Esse controle é realizado de ponta a ponta mediante cálculo da *cwnd* adequada para as condições de tráfego estimadas. Muitos algoritmos ponta a ponta foram propostos para atender essa finalidade em diferentes topologias de redes de comunicação.

Assim, TCP Hybla, por exemplo, foi projetado para eliminar penalizações de congestionamento para transmissões com grandes atrasos como as que ocorrem em comunicações por satélites, enquanto que TCP-BIC e TCP-CUBIC foram projetados para redes de alta velocidade e grandes atrasos de propagações. TCP-Veno (uma combinação de TCP-Reno e TCP-Vegas) [110], TCP Vegas-W [111] e TCP-NCE [112] foram propostos para redes sem fio, nas quais é frequente a ocorrência de perdas de pacotes por interferências ou queda de algum canal entre transmissor e receptor, e assim sendo, perdas de pacotes nem sempre implicam em congestionamento.

Em geral todas essas versões TCP estão baseadas no TCP-Reno [113] que é um dos mais clássicos e ainda um dos mais utilizados na prática [114] e descrito brevemente a seguir.

2.4.1 TCP-Reno

No TCP Reno [113, 115, 116], ao iniciar a transmissão o remetente utiliza um mecanismo denominado partida lenta, que consiste em, inicialmente, enviar um único pacote ao receptor, ou seja, começa com $cwnd \leftarrow 1$. Uma vez recebido esse pacote, o receptor envia um ACK confirmando a recepção e solicitando o próximo segmento (uma quantidade em bytes correspondente a um pacote de dados). Por cada ACK recebido, o transmissor incrementa sua janela em um segmento, isto é, faz $cwnd \leftarrow cwnd + 1$. Note que, na fase de partida lenta $cwnd$ duplica seu valor por cada janela de ACKs recebidos, resultando um crescimento exponencial da janela.

Além disso, se uma vez transmitido um segmento o transmissor não receber ACKs dentro de um determinado tempo de espera, (evento que é conhecido como *timeout*) entenderá que esse segmento foi perdido e torna a transmiti-lo. Também entenderá que essa perda de pacote é um sinal de congestionamento e, portanto, utiliza um limiar denominado *Ssthr* (*Slow Start Threshold*) o qual é ajustado como sendo a metade da janela de congestionamento atual $Ssthr = cwnd/2$ e reinicia a fase de partida lenta fazendo novamente $cwnd = 1$, e incrementando-a em uma unidade por cada ACK recebido.

Quando $cwnd$ atingir *Ssthr*, o transmissor sai da fase de partida lenta e entra em uma fase denominada prevenção de congestionamento, na qual por cada janela de ACK recebida incrementa-se $cwnd$ em uma unidade, ou seja, em prevenção de congestionamento $cwnd$ é incrementada em um pacote por cada W ACKs recebidos, resultando em um crescimento linear da janela.

Se o receptor receber o pacote com número de sequência $k-1$, enviará um ACK solicitando o pacote k , logo, se chegar o pacote $k+1$ antes do pacote com sequência k , o receptor enviará novamente um ACK solicitando o pacote k , assim, o transmissor poderá receber ACKs duplicados para a sequência k .

Se, estando na fase de partida lenta, ou na fase de prevenção de congestionamento, o transmissor receber três ACKs duplicados para o pacote com sequência k , entenderá que a rede está congestionada e entrará em uma fase denominada retransmissão rápida, na qual, assumirá que o pacote k foi perdido e irá retransmiti-lo, antes de ocorrer um *timeout*, e logo entrará em uma fase denominada recuperação rápida, na qual fará $Ssthr \leftarrow W/2$. Logo esse valor será truncado, ou seja, a parte fracionária de *Ssthr* será desprezada, e a janela de congestionamento será atualizada como $cwnd \leftarrow Ssthr + 3$ segmentos.

Estando em recuperação rápida, por cada ACK recebido, o transmissor aumenta $cwnd$ em um segmento e retransmite um novo pacote se $cwnd$ permitir (ou seja, se $cwnd$ for maior do que o valor que tinha no instante em que foi detectada a perda do pacote k). Quando um novo ACK é recebido, isto é, ocorrer uma confirmação de recepção de dados novos, o transmissor sai da fase de recuperação rápida fazendo $cwnd \leftarrow Ssthr$, sendo *Ssthr* o valor ajustado no passo anterior, e logo continua em prevenção de congestionamento.

2.5 Justiça para fluxos que compartilham uma mesma topologia de rede de comunicação

Quando mais de um fluxo de dados compartilham um mesmo canal de uma rede de comunicação pode ser desejável conseguir um justo compartilhamento da banda disponível, ou seja, em uma transmissão justa cada fluxo ocuparia uma mesma porção da largura de banda do canal.

Neste trabalho, para quantificar justiça, foi adotada a métrica proposta por Raj Jain em [117]. Se um conjunto de F fluxos com vazões ($V_1, V_2, V_3, \dots, V_F$) (medidas em unidades correntes, como Mbps) compartilham um mesmo canal, o índice de justiça de Jain (JJ) entre esses fluxos é calculado como

$$JJ = \frac{\left(\sum_{i=1}^F V_i\right)^2}{F \sum_{i=1}^F V_i^2}. \quad (2.5)$$

O índice de justiça de Jain sempre resulta em um número entre 0 e 1, e se supõe que justo significa igual. Assim, o melhor caso acontece quando cada fluxo recebe uma fatia igual da largura da banda, neste caso o índice de justiça será 1.

2.6 Técnicas AQM

Além dos protocolos de transporte de dados na Internet, tais como as diferentes versões do TCP que fazem um controle de congestionamento ponta a ponta, modernos dispositivos eletrônicos facilitaram o desenvolvimento de técnicas de gestão de filas que fazem controle de congestionamento no nível de rede. Essas técnicas são conhecidas como AQM e são implementadas nos roteadores da rede de comunicação, onde atuam descartando pacotes em função do tamanho e/ou da dinâmica da fila [104].

Também existe o controle de congestionamento entre camadas, onde os algoritmos AQM implementados nos roteadores, ao invés de descartarem pacotes, atuam marcando pacotes, ou seja, fazendo uma notificação explícita de congestionamento ECN (*Explicit Congestion Notification*). Quando um pacote marcado chega ao receptor, um ACK marcado é enviado para o transmissor informando-o que existe congestionamento no caminho. Assim, o transmissor TCP atua na sua taxa de transmissão reduzindo sua janela de congestionamento ($cwnd$) após a recepção de um ACK marcado [118].

O algoritmo AQM mais básico é o Drop Tail [104] onde pacotes que chegam na fila são descartados com probabilidade um quando, e somente quando, a fila do roteador está cheia, ou seja já não existe espaço para enfileirar mais pacotes. A principal desvantagem de Drop Tail é que pode causar que todos os fluxos TCP que atravessam a fila congestionada reduzam suas taxas de transmissão ao mesmo tempo, já que em cada episódio de congestionamento existe uma alta chance do Drop Tail descartar pacotes de cada fluxo ativo, e cada fluxo ativo reduz sua janela de congestionamento cada vez que detecta perdas de pacotes. Esse fenômeno indesejado é conhecido

como sincronização global e faz com que a ocupação do enlace seja sub-ótima.

Para superar tais inconvenientes, diversas outras técnicas AQM foram propostas, as quais, ao invés de esperar a fila encher, atuam com antecedência, respondendo a eventos de congestionamento, marcando ou descartando pacotes antes da fila encher. Alguns destes algoritmos serão brevemente descritos a seguir.

2.6.1 RED (*Random Early Detection*)

RED foi proposto por Floyd e Jacobson [104] e, basicamente, detecta congestionamento por estimação do tamanho médio da fila. Assim, em RED dois limiares min_{th} e max_{th} são definidos. max_{th} geralmente é ajustado como $max_{th} = 3min_{th}$ e, cada vez que um pacote chega à fila esses limiares são comparados com a estimação do tamanho médio da fila (Q_{AVG}), o qual é atualizado cada vez que um pacote chega na fila do roteador da seguinte forma

$$Q_{AVG} = (1 - w_q) Q + w_q qlen, \quad (2.6)$$

sendo w_q um peso normalmente ajustado para 0,002 e $qlen$ é o tamanho atual da fila. Uma vez estimado o tamanho médio da fila, a probabilidade P de marcar ou descartar um pacote é calculada como

$$P = \begin{cases} 0, & \text{para } Q_{AVG} < min_{th} \\ P_{max} \cdot \frac{Q_{AVG} - min_{th}}{max_{th} - min_{th}}, & \text{para } min_{th} < Q_{AVG} < max_{th} \\ 1, & \text{para } Q_{AVG} > max_{th}, \end{cases} \quad (2.7)$$

onde P_{max} normalmente ajustado em 0,1, é o máximo valor que pode assumir a probabilidade (P) de descartar ou marcar pacotes na fila.

RED é simples, e apesar de ser efetivo para superar problemas de sincronização global e controlar o tamanho da fila, seu desempenho depende da configuração de seus parâmetros e pode apresentar pobre desempenho em alguns casos. Assim, algumas modificações de RED foram propostas tais como ARED (RED adaptativo) [105] que consegue um menor tempo de convergência ajustando dinamicamente o valor de max_{th} e RARED [106] que considera a taxa de entrada de pacotes na fila como critério para calcular a probabilidade de descartar pacotes e, caso o crescimento dessa taxa de entrada seja maior que um determinado limiar δ , uma agressiva probabilidade de descarte ou marcação de pacotes é aplicada e, caso contrário RARED, funciona como RED.

Por outro lado, com a introdução de roteadores com grandes capacidades de armazenamento, capazes de enfileirar enorme quantidade de pacotes, os problemas de congestionamento na Internet foram paradoxalmente piorando ainda mais, já que esses roteadores permitem enfileirar um grande número de pacotes e longas filas de pacotes implicam em grandes atrasos ponta a ponta. Esse fenômeno foi recentemente exposto como problema de *bufferbloat* [100, 101]. Novas técnicas AQM, tais como CoDel (*Control Delay*) [100] e PIE (*Proportional Integral controller Enhanced*) [101] foram recentemente propostas para atacar esses problemas de *bufferbloat*.

2.6.2 CoDel (*Control Delay*)

CoDel foi proposto por Nichols e Jacobson [100] e a grande diferença de CoDel com respeito a RED está no fato de que CoDel evita a utilização de parâmetros que devem ser configurados pelo usuário e que possam conduzir a pobre desempenho quando a configuração escolhida não for adequada para as condições de tráfego e não for apropriada para a topologia de rede de comunicação utilizada.

Por outro lado, além da similaridade com o RED de marcar ou descartar pacotes antes que eventos de congestionamento ocorram, e assim evitar o indesejado fenômeno de sincronização global, também visa evitar filas longas nos roteadores, para assim conseguir menores atrasos ponta a ponta. Para atingir seus objetivos, CoDel trabalha com parâmetros totalmente independentes de condições de tráfego ou de topologias de redes de comunicação, ou seja, totalmente independente de atrasos de ponta a ponta, tamanho da banda e/ou tempo de propagação nos canais, quantidade de transmissores, carga do tráfego ou qualquer outro fator ou condição que não possa ser prevista.

CoDel utiliza um período de observação T_{obs} (ajustado inicialmente como 100 ms) no qual mede o tempo que cada pacote permanece na fila. Ao finalizar o período de observação, o CoDel compara o valor mínimo de atraso na fila T_{min} encontrado neste período (T_{obs}) com um valor alvo (normalmente ajustado para 5 ms). Se o T_{min} observado for maior que o valor alvo, o pacote que está na cabeça da fila é descartado ou marcado, e o intervalo de observação é recalculado como uma função do número de pacotes consecutivos que foram descartados ou marcados, de acordo com

$$T_{obs} = \frac{100}{\sqrt{nDrop}}, \quad (2.8)$$

onde $nDrop$ é a quantidade de pacotes consecutivos que foram descartados ou marcados.

Se o valor mínimo de atraso na fila observado for menor ou igual que o valor alvo, o pacote que está na cabeça da fila é enviado para seu destino (não é marcado nem descartado) e o período de observação é ajustado novamente para 100 ms e $nDrop$ é ajustado para 0.

As desvantagens do CoDel estão no fato que descartes de pacotes são realizados na cabeça da fila; assim, pacotes necessitam ser enfileirados antes de serem descartados. Além disso, CoDel necessita de um registro de tempo no cabeçalho TCP para poder medir quando o pacote entrou na fila e quando o pacote saiu da fila para assim poder calcular o tempo de espera para cada pacote. De acordo com [101], esse requisito implica em alta necessidade de infraestrutura e recursos de processamento.

2.6.3 PIE (*Proportional Integral controller Enhanced*)

A fim de superar os inconvenientes do CoDel, em [101] foi proposta uma técnica AQM denominada PIE (*Proportional Integral controller Enhanced*), a qual dinamicamente calcula uma probabilidade de descartar ou marcar pacotes com base em uma estimativa (ao invés de uma medição como em CoDel) do tempo de espera na fila, o qual é função da estimativa de uma taxa de

partida que é avaliada após a saída de cada pacote. Além disso, PIE incorpora em seu algoritmo uma previsão para chegadas em rajadas, que em caso de estar habilitada os pacotes passam pela fila sem aplicar a probabilidade de descarte. Assim, o algoritmo PIE possui três partes descritas a seguir.

Primeira Parte: Cálculo da probabilidade de marcar ou descartar pacotes:

Nesta primeira parte, PIE utiliza um tempo constante de atualização do algoritmo chamado T_{Update} cujo valor recomendado é de 30 ms. Assim, após cada T_{Update} são executados os seguintes passos:

- É feita uma estimativa do atual atraso na fila como

$$CurDelay = \frac{qlen}{AvgDrate}; \quad (2.9)$$

sendo $qlen$ o tamanho da fila, $AvgDrate$ é a taxa de drenagem da fila (obtida na segunda parte do algoritmo PIE).

- Com base na atual probabilidade de descartar ou marcar pacotes (P) são determinados os seguintes parâmetros:

$$\begin{cases} \alpha = \hat{\alpha}/8; & \beta = \hat{\beta}/8 \text{ para } P < 1\%, \\ \alpha = \hat{\alpha}/2; & \beta = \hat{\beta}/2 \text{ para } P < 10\%, \\ \alpha = \hat{\alpha}; & \beta = \hat{\beta} \text{ para } P \geq 10\%; \end{cases} \quad (2.10)$$

sendo α e β fatores de escalonamento, e $\hat{\alpha}$ e $\hat{\beta}$ constantes cujos valores recomendados são 0,125 Hz e 1,25 Hz, respectivamente.

- Calcula-se a probabilidade de marcar ou descartar pacotes como

$$P = P + \alpha (CurDel - RefDel) + \beta (CurDel - OldDel); \quad (2.11)$$

- Atualiza-se a estimativa anterior do tempo de espera na fila como

$$OldDel = CurDel, \quad (2.12)$$

onde, $CurDel$ e $OldDel$ representam o valor atual e passado da estimativa do tempo de espera na fila, e $RefDel$ é um tempo de espera na fila utilizado como referência que é constante e o valor recomendado é de 20 ms.

Segunda Parte: Cálculo da taxa de partida de pacotes

Essa segunda parte do PIE é executada cada vez que um pacote sai da fila, ou seja, é atendido. Os seguintes passos são executados.

- Se $qlen > DqThreshold$, o algoritmo entra em um ciclo de medição e:
- Atualiza-se o contador de saída de pacotes $DqCount$ como:

$$DqCount = DqCount + DqPacketSize;$$

- Se $DqCount > DqThreshold$, a taxa de partida e os contadores são atualizados como:

$$DqInt = Tnow - Tstart, \quad (2.13)$$

$$DqRate = \frac{DqCount}{DqInt}, \quad (2.14)$$

$$AvgdDrate = (1 - \epsilon).AvgdDrate + \epsilon DqRate, \quad (2.15)$$

$$Tstart = Tnow, \quad (2.16)$$

$$DqCount = 0, \quad (2.17)$$

onde $DqPacketSize$ é o tamanho do pacote que acaba de sair da fila, $DqThreshold$ é um limiar do tamanho da fila cujo valor recomendado é de 10 kbytes, $DqInt$ é a diferença entre o tempo atual ($Tnow$) e do tempo da última atualização da taxa de saída de pacote ($Tstart$), e ϵ é um peso do modelo ajustado entre 0 e 1.

Terceira Parte: Cálculo da tolerância às rajadas

Essa terceira parte de PIE é executada cada vez que um pacote entra na fila. E os seguintes passos são executados:

- Se $BurstAllow > 0$ o pacote é enfileirado sem a aplicação da probabilidade P de descartar ou marcar o pacote;
- Após uma atualização da taxa de partida (realizada na segunda parte do algoritmo PIE) atualiza-se $BursAllow$ de acordo com

$$BursAllow = BursAllow - DqInt; \quad (2.18)$$

- Se, após uma atualização da taxa de partida (realizada na segunda parte do algoritmo PIE) se cumprir que $P = 0$ e ambos $CurDel$ e $OldDel$ forem menores que $RefDel/2$, então, se reinicia $BursAllow$ como

$$BursAllow = MaxBurst, \quad (2.19)$$

onde $BurstAllow$ é uma variável que controla a tolerância de PIE às rajadas e $MaxBurst$ uma constante igual ao máximo valor que pode assumir $BursAllow$. O valor recomendado para $MaxBurst$ é de 100 ms.

2.6.4 Notificação Explícita de Congestionamento (ECN)

As técnicas AQM descritas acima podem atuar tanto descartando pacotes nas filas dos roteadores como marcando pacotes com a finalidade de notificar o transmissor acerca da existência de congestionamento, ou seja, em uma comunicação TCP/IP (*Transmission Control Protocol / Internet Protocol*) que suporta ECN (*Explicit Congestion Notification*), um roteador pode colocar uma marca no cabeçalho IP em vez de descartar o pacote, marcando um bit de notificação de congestionamento, e assim o transmissor poderá reduzir sua taxa de transmissão ao receber essa notificação de congestionamento [118].

ECN é descrita na RFC 3168 (2001) [119] e pode ser usada na comunicação entre dois dispositivos que suportem a utilização de ECN quando a infraestrutura de rede de comunicação usada para trocar dados também suporta o uso ECN, ou seja, para trabalhar com ECN, necessita-se de recursos tecnológicos implementados tanto nos roteadores como no transmissor e no receptor.

Ainda, pode ocorrer que a opção ECN seja dependente do caminho de roteamento de pacotes entre transmissor e receptor, ou seja, roteadores existentes em um determinado caminho podem estar habilitados para trabalhar com ECN, porém, quando o pacote é roteado por outro caminho pode encontrar roteadores que não suportem essa opção [120]. Um protocolo TCP baseado em ECN (como os descritos na seção seguinte) poderia funcionar de forma inadequada quando isso ocorre, o que pode desincentivar a implementação desse tipo de protocolo.

Assim sendo, uma desvantagem de utilizar notificação explícita de congestionamento, ao invés de detectar congestionamento de forma implícita (por detecção de perdas de pacotes mediante recepção de ACKs duplicados ou timeout) é a necessidade de maiores recursos tecnológicos que garantam uma cooperação entre todos os dispositivos que participam na comunicação.

Quando transmissor e receptor estão habilitados para trabalhar com ECN, mas a infraestrutura da rede não suporta seu uso, normalmente esse bit é ignorado pelos roteadores e a comunicação é feita como se essa opção não estivesse disponível. Porém, alguns trabalhos mostraram que, nestes casos, podem ocorrer problemas na comunicação, já que em algumas ocasiões roteadores não compatíveis com ECN podem descartar ou danificar os pacotes que possuam esse bit disponível no cabeçalho IP [121, 122, 123]. Esse problema obviamente compromete a segurança da comunicação e pode até torná-la impossível.

Outro problema pode ocorrer quando o receptor não é compatível para trabalhar com ECN, e ainda erroneamente concordar com o transmissor em usar ECN. Neste caso, pode ocorrer que todos os pacotes de ACKs enviados para o transmissor sejam marcados pelo receptor, logo o transmissor interpretará isso como sucessivas notificações de congestionamento e nunca aumentará o tamanho de sua janela de transmissão, fazendo assim com que a vazão seja muito pequena [119].

Essa carência de recursos tecnológicos para implementar ECN em parte se deve a que a falta de compatibilidade dos transmissores para trabalhar com ECN desincentiva a implementação de roteadores que tenham essa opção disponível junto com uma técnica AQM e vice-versa. Porém, esses problemas ocasionados por incompatibilidade tecnológica foram mais frequentes logo após a padronização de ECN (dada no ano 2001 [119]), e estão diminuindo com o passar dos anos.

A quantidade de servidores web capazes de trabalhar com ECN cresceu exponencialmente desde então. Assim, experimentos feitos no ano 2001 sobre 84.000 servidores web, mostram que somente 1,1 % eram capazes de trabalhar com ECN [124]. Porém, para o ano 2004 essa fração aumentou para 2,1% [125]. E para o ano 2011 subiu para 11,2 % [122], atingindo 29,48 % no ano 2012 [123]. Já experimentos feitos no ano 2015 sobre 600 mil servidores web [120] mostraram que 65,41% dos servidores IPv6 e 56,17 % dos servidores IPv4 já eram capazes de trabalhar com ECN.

Já os problemas de segurança gerado por incompatibilidade de ECN, tais como as distorções ou descarte de pacotes feitos por roteadores que não entendem esses bits, reduziram para menos de 1 % em 2015 [120].

Porém, a disponibilidade de trabalhar com ECN por parte dos servidores web é apenas uma parte do problema. Para que a técnica ECN possa ser implementada na prática, além de transmissor e receptor também todos os roteadores existentes no caminho devem ser capazes de trabalhar com ECN.

Um ponto a favor do uso de ECN neste sentido, ocorreu em junho de 2015 quando a *Apple* (empresa multinacional norte-americana que tem o objetivo de projetar e comercializar produtos eletrônicos de consumo, software de computador e computadores pessoais) anunciou que a opção ECN estaria disponível em todos seus produtos para ajudar a impulsionar a adoção da sinalização ECN em toda a indústria [126]. Como é possível observar, o panorama está cada vez mais favorável para a implementação de ECN na prática.

Funcionamento de ECN

Na RFC 3168 [119] foram definidos dois bits do cabeçalho IPv4 ou IPv6 para ECN, que juntos, codificam 4 possibilidades: O primeiro foi denominado ECT (*ECN-Capable Transport*), e é ativado pelo transmissor indicando se as estações finais estão habilitadas ou não para trabalhar com ECN. O segundo foi denominado CE (*Congestion Experienced*) e é ativado pelo roteador para indicar congestionamento.

Quando ambos dispositivos estão habilitados para trabalhar com o protocolo eles marcam seus pacotes com ECT (0) ou ECT (1). Se o pacote passa por uma fila congestionada de um roteador que possui uma técnica AQM, e que está habilitado para trabalhar com ECN, então o roteador muda o estado do bit CE, ao invés de descartar o pacote.

Para essa finalidade é sugerido utilizar os bits 6 e 7 do campo TOS (*Type of Service*) do IPv4 RFC 791 [127] para trabalhar com ECN, sendo sugerido o bit 6 para ECT e o bit 7 para o CE. No caso do IPv6 (Ref 2460 [128]) podem ser utilizados os bits 6 e 7 do campo *Traffic Class* que é equivalente ao campo TOS do IPv4.

Funcionamento de ECN com TCP

TCP suporta o uso de ECN utilizando 3 bits do cabeçalho do pacote TCP. O primeiro é o bit NS (*Nonce Sum*) que é usado para prevenir erros acidentais ou malicioso de cancelamento do uso

Tabela 2.1: Bits ECN

CE/ECT	Descrição
00	Comunicação com ECN não habilitada.
10	Comunicação com ECN habilitada, ECT (0).
01	Comunicação com ECN habilitada, ECT (1).
11	Congestionamento encontrado.

de ECN pelo transmissor TCP, que poderia ser feito com a finalidade de ganhar uma parcela maior da banda do canal e assim gerar injustiça na transmissão. Segundo a Tabela 2.1 o transmissor pode usar os dois códigos ECT (ECT (0) ou ECT (1)) no cabeçalho IP para acordar o uso de ECN na comunicação, e assim um ECT (0) no cabeçalho IP deve ir acompanhado de um NS (0) no cabeçalho TCP e um ECT (1) no cabeçalho IP deve ir acompanhado de um NS (1) no cabeçalho TCP (RFC 3540 [129]).

Quanto aos outros dois bits utilizados, uma vez recebida uma notificação explícita de congestionamento vinda do roteador, o receptor usa o bit ECE (*ECN-Echo*) do cabeçalho TCP para transmitir essa notificação para o transmissor em cada ACK a ser enviado. Quando o transmissor TCP recebe essa notificação do receptor, ele usa o bit CWR (*Window Reduced*) para informar o receptor de que a notificação explícita de congestionamento foi recebida com sucesso. Uma vez que o receptor recebe essa notificação do transmissor ele deixa de marcar o bit ECE de seus ACKs até receber uma nova notificação de congestionamento vinda do roteador.

O uso de ECN em conexões TCP é opcional. Portanto, antes de usá-lo, transmissor e receptor negociam o estabelecimento de uma conexão ECN mediante a inclusão da opção no segmento SYN e no SYN-ACK, respectivamente. Uma vez negociado o uso de ECN, o transmissor marca o pacote IP que carrega o segmento TCP com o código ECT permitindo assim, que os roteadores intermediários entendam que esse pacote pode ser marcado ao invés de ser descartado pela técnica AQM neles implementada, estando o bit CE do cabeçalho IP disponível para essa opção.

Benefícios do uso de ECN

Dado que ECN é efetivo somente quando trabalha junto com uma técnica AQM implementada nos roteadores, é de se esperar que seus benefícios dependam da lógica utilizada pela técnica AQM para marcar pacotes, e também de como o transmissor TCP reagirá diante da recepção de uma notificação explícita de congestionamento.

ECN reduz a quantidade de pacotes descartados em uma comunicação TCP/IP, e consequentemente, pode evitar *timeouts* e necessidades de fazer retransmissões de pacotes [130] reduzindo assim os atrasos. Porém, efeitos de ECN na vazão são menos claros [131]. Nos Capítulos 3 e 4 deste trabalho se mostrará que, em algumas situações, o transmissor pode reagir de forma exagerada a uma notificação de congestionamento dada pela técnica AQM e, assim, as filas nos roteadores podem permanecer completamente vazias durante alguns períodos de tempo. Consequentemente a vazão acaba sendo prejudicada. Esse fenômeno indesejado foi identificado neste trabalho e chamado de fenômeno de *bufferempty*. Também será mostrado que esse fenômeno indesejado pode ser

solucionado melhorando o trabalho em conjunto da técnica AQM e da ação do transmissor TCP diante da recepção de uma notificação de congestionamento.

Outras metodologias utilizadas para notificação explícita de congestionamento

Com o objetivo de conseguir fazer notificações explícitas de congestionamento no menor tempo possível, ou seja, notificar rapidamente o transmissor quando ocorre um congestionamento na rede, diferentes técnicas para marcar pacotes foram propostos na literatura. Em [132], por exemplo, foi proposta uma técnica denominada DM (*Differentiated Marking*), a qual quando uma fila está congestionada além de marcar um pacote de dados que viaja no sentido transmissor/receptor, também é marcado um ACK que viaja no sentido receptor/transmissor, para assim reduzir o tempo de notificação de congestionamento, porém essa técnica somente poderá reduzir o tempo de notificação de forma eficaz quando o ACK é enviado no sentido contrário mas pelo mesmo caminho do pacote de dado cuja recepção está sendo confirmada.

Com esse mesmo objetivo, em [133] foi proposta uma estratégia chamada *Mark Front*, na qual, quando um pacote chega em uma fila congestionada a técnica AQM não marca esse pacote mas sim o pacote que está na ponta da fila, ou seja, o primeiro pacote a ser emitido pra frente, e assim faz uma notificação de congestionamento mais rápida.

Também na literatura foram propostas outras técnicas que além de fazer uma notificação explícita de congestionamento, pretendem dar também ao transmissor maiores informações de quão congestionada está a fila no roteador. Assim, em [134, 135], foi proposta uma técnica denominada EWA (*Explicit Window Adaptation*), a qual basicamente consiste de modificar o cabeçalho do ACK que viaja no sentido receptor/transmissor alterando nele a janela *rwnd* imposta pelo receptor para fazer controle de fluxo. Assim, o algoritmo EWA implementado no roteador modifica *rwnd* em função do espaço livre existente em seu *buffer* e, ao receber essa notificação, o transmissor calculará sua janela de acordo com a Equação (2.4). Porém, na prática, a implementação dessa técnica só é eficaz quando o ACK viaja no sentido contrário mas pelo mesmo caminho que o pacote de dados cuja recepção está sendo confirmada. Seguindo essa mesma metodologia, Talau [136] propôs uma técnica denominada *Early Congestion Control* (ECC) a qual funciona de forma semelhante a EWA.

2.7 Protocolos de transporte de dados baseados em ECN

Devido ao crescente uso de notificações de congestionamento em transmissões TCP, nos últimos anos foram propostos diversos algoritmos TCP baseados em ECN, com o objetivo de aproveitar melhor a informação contida em ECN, e assim casar técnicas AQM implementadas em roteadores com ajustes na janela TCP feita nos transmissores. Alguns desses protocolos são TCP-Jersey [137], Data Center TCP (DCTCP) [138] e suas variantes Enhanced DCTCP (E-DCTCP) [139], DCTCP⁺ [140], Double-Threshold DCTCP [141] e Omniscient TCP (OTCP) [142]. Com a finalidade de fazer comparações com as metodologias propostas neste trabalho serão usados o TCP-Jersey (um dos protocolos baseado em ECN mais clássicos e consolidados na literatura), o DCTCP

(um dos protocolos baseado em ECN mais citado na literatura recente [138, 140, 141, 139, 142]) e o E-DCTCP (uma das modificações mais recentes do DCTCP), que serão descritos de forma sintetizada a seguir.

2.7.1 TCP-Jersey

TCP-Jersey [137] é um protocolo que utiliza a janela de congestionamento $cwnd$ em função de recepções de notificações explícitas de congestionamento feitas por técnicas AQM implementadas nos roteadores. Para essa finalidade, nos roteadores é implementado um algoritmo AQM denominado advertência de congestionamento CW (*Congestion Warning*), que trabalha com ECN, marcando pacotes com probabilidade 1 quando o tamanho da fila no roteador for maior que um determinado limiar e com probabilidade zero quando for menor que esse limiar.

CW auxilia o transmissor TCP-Jersey tanto para diferenciar perdas de pacotes por congestionamento das que ocorrem por erros em conexões sem fio, quanto para fazer um ajuste dinâmico de sua janela de congestionamento.

Assim, no transmissor TCP-Jersey, é implementado um algoritmo de estimação da banda disponível (ABE) (*Available Bandwidth Estimation*) (baseado no algoritmo de estimação de banda disponível do TCP-Westwood). ABE basicamente monitora a taxa de recepção de ACKs para estimar a banda disponível e então calcula uma $cwnd$ ótima ($ownd$) em função da banda disponível estimada.

$ownd$ é calculada uma vez por RTT, e quando o transmissor TCP-Jersey recebe uma notificação de advertência de congestionamento (CW) $cwnd$ e $ssth$ são ajustados para $ownd$, ou seja, $cwnd \leftarrow ownd$ e $ssth \leftarrow ownd$. Se chegar 3 ACK repetidos não marcados, TCP-Jersey faz a retransmissão do pacote considerado perdido e vai para a fase de recuperação rápida, mas se o terceiro ACK repetido estiver marcado, então o transmissor primeiro ajusta sua janela de congestionamento e seu $ssth$ para $ownd$, ou seja, faz $cwnd \leftarrow ownd$ e $ssth \leftarrow ownd$ e logo faz a retransmissão do pacote considerado perdido e finalmente entra na fase de recuperação rápida.

O calculo de $ownd$ é feito pelo algoritmo ABE da seguinte forma:

Cada vez que um ACK chega no transmissor a banda disponível é calculada como:

$$R_n = \frac{RTT R_{n-1} + L_n}{(t_n - t_{n-1}) + RTT}, \quad (2.20)$$

onde, R_n é a banda estimada na chegada do ACK n , t_n é o instante de chegada do ACK n , t_{n-1} é o instante de chegada do ACK $n - 1$, L_n é o tamanho do pacote de dado cuja recepção foi confirmada pelo ACK n e RTT é *Round Trip Time*.

Logo, a janela ótima de transmissão ($ownd$) é calculada em unidades de segmentos como

$$ownd_n = \frac{RTT R_n}{SegSize}, \quad (2.21)$$

onde $SegSize$ é o tamanho do segmento.

2.7.2 Data Center TCP (DCTCP)

Data Center TCP (DCTCP), análogo ao TCP-Jersey, é uma protocolo TCP baseado em AQM e ECN. A técnica AQM utilizada é baseada em RED com $min_{th} = max_{th} = K$ e $w_q = 1$. Assim, diferentemente de RED, o único parâmetro que deve ser ajustado pelo projetista é o parâmetro K , e note que, como w_q é igual a um, o tamanho médio da fila é exatamente igual ao tamanho da fila (veja a Equação (2.6)).

Do lado do transmissor DCTCP, as notificações de congestionamento recebidas não são vistas apenas como dois estados lógicos possíveis (congestionado ou não congestionado), mas sim como uma proporção de quão congestionado possa estar a fila no gargalo. Assim, a janela de transmissão $cwnd$ é atualizada em função da proporção de notificações de congestionamento recebidas durante um RTT como

$$cwnd \leftarrow (1 - \frac{\alpha}{2}) cwnd, \quad (2.22)$$

sendo α um parâmetro calculado em cada RTT como

$$\alpha \leftarrow (1 - g) \alpha + g F, \quad (2.23)$$

onde F é a fração de pacotes marcados durante o RTT e g é um fator de peso cujo valor deve estar entre 0 e 1, sendo o valor recomendado 1/16. O parâmetro α é atualizado uma vez por RTT.

2.7.3 E-DCTCP

E-DCTCP [139] é uma modificação do protocolo DCTCP também baseado em ECN, que além de usar notificação explícita de congestionamento, utiliza uma notificação de perdas de pacotes para evitar *timeout*, e um algoritmo que calcula um tempo aleatório para fazer retransmissões dos pacotes perdidos e assim reagir de forma mais rápida que DCTCP diante de perdas de pacotes. Em DCTCP, a partir do momento em que a fila em um roteador ultrapassar um determinado limiar K , todos os pacotes serão marcados. Porém, quando o *buffer* estiver completamente cheio, os novos pacotes que chegam na fila serão descartados e poderá ocorrer um *timeout* no transmissor, e somente após o *timeout* o transmissor retransmitirá os pacotes perdidos. E-DCTCP, além de ECN, usa um método de notificação de perdas de pacotes PL (*Packet Loss*).

Assim, quando um pacote chega a um roteador cuja fila está superando a capacidade do *buffer*, ele terá um enfileiramento privilegiado, e o roteador modificará o cabeçalho IP do pacote trocando remetente e destinatário, colocando como destino o próprio transmissor e como remetente o receptor. Logo, moverá o pacote do módulo de entrada para o módulo de saída dentro da mesma porta por onde o pacote entrou, colocando ainda o pacote na cabeça da fila de saída, adicionando também uma marca PL no cabeçalho IP do pacote.

Para garantir o enfileiramento privilegiado em um *buffer* cheio, outro pacote que está no final da fila também será removido e retornado para seu correspondente transmissor com a marca PL,

a fim de gerar um espaço para o pacote que está chegando. Todo esse processo interno é possível de ser implementado nos roteadores comerciais existentes atualmente.

A marca PL é feita utilizando os mesmos bits de ECN. Observe que a RFC 3168 não faz diferença entre ECT (0) e ECT (1) (veja a Tabela 2.1), eles estão abertos para serem padronizados pelos protocolos TCP. Assim, no funcionamento padrão de ECN, se um pacote entra em uma fila congestionada com a combinação ECN 01 ou 10, ou seja, ECT(0) ou ECT(1), a técnica AQM implementada no roteador marcará o pacote com 11. Ainda, segundo a RFC 3168 [119], ECT (0) deve ser preservado se somente um código ECT for necessário. Assim, E-DCTCP usa ECT (1) como indicador de PL.

A segunda modificação de E-DCTCP com relação a DCTCP é um algoritmo de retransmissão de pacotes. Quando um pacote marcado com PL chega no transmissor, ele extrairá do cabeçalho IP os endereços de remetente e destinatário para identificar o fluxo. E logo, extrairá também o número de sequência e o tamanho do pacote para decidir que faixa de dados devem ser retransmitidos.

Depois, ao invés de ir diretamente para a fase de retransmissão, o transmissor E-DCTCP usa um algoritmo que gera um tempo de espera selecionado aleatoriamente e uniformemente distribuído entre 0 e $2^n \cdot rtt$ antes de ir para a fase de retransmissão. Isso é coerente, pois fazer retransmissões imediatas de pacotes tendo a informação de que uma fila está completamente congestionada poderia gerar novas perdas. rtt é uma estimativa do RTT para o correspondente fluxo TCP. n é calculada como $n = \min(k, b)$, onde k é um contador de retransmissões e b uma constante.

2.8 Conclusões

O desempenho de alguns sistemas de controle em rede pode depender das características da rede, principalmente de atrasos e perdas de pacotes. Na bibliografia, geralmente o estudo das NCS é realizado inserindo atrasos e taxas de perdas de pacotes contantes ou aleatórias entre controlador e planta. Porém, em sistemas de controle através da Internet, geralmente controlador e planta trocarão dados em uma topologia de rede de comunicação compartilhada com diversos outros fluxos de dados simultâneos. Logo, provavelmente, haverá influências mútuas entre os diferentes fluxos que compartilham a mesma topologia de rede de comunicação.

Assim, atrasos e perdas de pacotes além de dependerem das características físicas do meio de comunicação que faz a conexão entre planta e controlador, tais como tempo de propagação, banda dos canais, confiabilidade do meio, interferências externas tais como ruídos que provocam distorções de sinais ou obstáculos físicos que atenuem os sinais em canais sem fios, também dependerão das características dos outros fluxos que compartilham o mesmo canal (como tamanho dos pacotes, tipos de algoritmos utilizados na camada de transporte de dados e velocidades de transmissões utilizadas).

Por outro lado, em fluxos de dados através da Internet, atrasos, descarte e perdas de pacotes também dependem das características das técnicas AQM utilizadas, as quais podem atuar marcando ou descartando pacotes com o objetivo de gerenciar o tamanho das filas nos roteadores.

Para que as técnicas AQM consigam fazer notificações explícitas de congestionamento transmissor, receptor e todos os roteadores existentes nos caminhos entre transmissor e receptor devem ser capazes de trabalhar com ECN. Isso pode dificultar a implementação prática de ECN, tornando assim ainda interessante desenvolver protocolos que sejam capazes de fazer uma adequada notificação implícita de congestionamento ao invés de utilizar ECN, e procurar soluções que superem os problemas de limitações tecnológicas.

Assim sendo, para simular sistemas de controle através da Internet, com um melhor modelo para atrasos e perdas de pacotes, é necessário uma ferramenta que permita modelar as características da rede de comunicação, as características de outros fluxos que compartilham a mesma topologia de rede de comunicação, e os algoritmos AQM. E ainda, dado que a resposta temporal para sistemas de controle através da Internet possui tanto componentes determinísticas quanto aleatórias seu estudo mediante simulações requer de ferramentas que possibilitem fazer análises probabilístico dos resultados.

Considerando essas características dos sistemas de controle através da Internet, no próximo capítulo será abordado modelagens de redes de comunicação na ferramenta de software UPPAAL, na qual serão simulados, de forma conjunta, fluxos TCP, fluxos UDP de uma NCS e técnicas AQM implementadas nos roteadores presentes na rede de comunicação.

Capítulo 3

Modelagem, Simulação e Verificação em UPPAAL de Fluxos TCP e NCS que Compartilham a mesma Topologia de rede de comunicação

3.1 Introdução

Neste capítulo serão descritas as características gerais de uma NCSs, tendo como foco o caso particular de NCSs que usam a Internet como meio de comunicação entre controlador e planta. Será estudado como os diversos fluxos de dados existentes na Internet afetam a resposta temporal desse tipo de NCSs, e como essas influências dependem dos protocolos de transporte de dados e das técnicas de gestão de filas implementadas nos roteadores da Internet.

Estudos acerca desse tema serão realizados mediante uma metodologia de modelagem de NCS desenvolvida neste trabalho, que admite modelar, simular e fazer a verificação de cunho probabilístico de topologias de redes de comunicação e seus protocolos e algoritmos na ferramenta de software UPPAAL. No UPPAAL, foi possível modelar e simular tanto um sistema de controle via Internet, quanto a topologia de rede de comunicação com seus diferentes fluxos, protocolos e algoritmos. Assim, a ferramenta UPPAAL foi escolhida por admitir vários níveis de modelagem e simulação e por ainda possibilitar fazer verificações probabilísticas, testando a probabilidade de determinadas variáveis, parâmetros e/ou indicadores de desempenho atingir certos valores em um determinado tempo.

Assim, neste capítulo, serão modelados sistemas de controle através da Internet, medindo tanto o desempenho do sistema de controle mediante o ITAE, quanto o desempenho dos outros fluxos genéricos que compartilham a mesma topologia de rede de comunicação em termos de vazão. Esses indicadores de desempenho serão avaliados mediante simulações e verificações estatísticas para diferentes protocolos de transporte de dados e diferentes técnicas AQM implementadas nos

roteadores.

3.2 UPPAAL

O UPPAAL [143, 144] é uma ferramenta para modelagem, simulação e verificação de sistemas de tempo real modelados como um conjunto de autômatos dependentes de condições temporais. Essa ferramenta foi desenvolvida de forma conjunta pela universidade de *Uppsala* da Suécia e a universidade de *Aalborg* da Dinamarca, sendo lançada a primeira versão no ano 1995. A ferramenta é livre e gratuita para fins acadêmicos e pode ser obtida da página *www.uppaal.org* onde também é possível encontrar diversos tutoriais e exemplos de aplicação.

Mediante a ferramenta de software UPPAAL, sistemas físicos e lógicos, cuja dinâmica muda por eventos discretos de natureza aleatória e/ou determinística, podem ser modelados em conjuntos de autômatos temporizados através de uma linguagem baseada em TCTL (*Timed Computation Tree Logic*), facilitando assim realizar uma verificação formal de todo o sistema modelado [144].

A lógica TCTL [145] é uma extensão para tempo real da lógica CTL [146] sendo assim adequada para expressar propriedades temporais, como por exemplo, “o motor deve alcançar uma velocidade de mil revoluções por minuto em até 5 s após acionado o botão de arranque”. Esse tipo de formalismo é muito utilizado em sistemas de controle industrial [147], em transmissões de dados com perdas de pacotes [148] e em análises de protocolos utilizados na automação industrial [149].

UPPAAL ainda conta com uma ferramenta denominada *Statistical Model Checking* (SMC) que permite fazer verificações e simulações estatísticas do sistema modelado, ou seja, pode-se verificar se um sistema em estudo pode chegar a um determinado estado, e ainda pode-se também quantificar a probabilidade com que isso poderá acontecer [144, 97, 143, 99, 88, 2].

Quanto à modelagem, UPPAAL admite o uso de variáveis do tipo relógio, variáveis inteiras, flutuantes e funções definidas pelo usuário.

Relógios são a forma de lidar com o tempo em UPPAAL, e o tempo corre em números inteiros e sua evolução é global e muda ao mesmo passo em todo o sistema. As variáveis tipo relógio medem o progresso do tempo, e UPPAAL permite comparar valores das variáveis de relógio, ajustá-las para determinados valores ou reiniciá-las. Os sistemas físicos podem se modelados mediante autômatos temporizados, cada autômato é um grafo, com estados e transições entre estados. Transições acontecem quando determinadas condições de guarda se satisfazem. Guarda é uma condição colocada em relógios e/ou em outros tipos de variáveis que administram as transições habilitando-as ou não.

Além de condições de guarda, o UPPAAL também permite utilizar condições invariantes. Uma invariante indica quanto tempo o sistema pode permanecer em um determinado estado. Além disso, durante uma transição algumas ações são possíveis, tais como, atualização de variáveis, chamada de funções, reinicialização de relógios ou combinações delas.

Guardas são condições colocadas em transições enquanto que invariantes são condições colocadas nos estados. Ambas são escritas utilizando operadores lógicos tais como $>$, $<$, $>=$, $<=$ ou $==$.

Enquanto que as atualizações de variáveis e reinício de relógios são escritas utilizando o operador lógico = ou :=.

3.2.1 Modelagem mediante Autômatos Temporizados

Os autômatos temporizados, introduzidos por Alur e Dill [145], especificam um formalismo para modelar sistemas cujo comportamento depende do tempo. Ou seja, a dinâmica do sistema é composto por uma série de eventos que acontecem após expirações de cronômetros. Esses cronômetros são definidos como variáveis do tipo relógio (*clock*) e, na modelagem global do sistema, todas as variáveis de relógios definidas devem ter a mesma base de tempo (exemplo: milissegundos (ms), segundos (s), minutos, horas, dias e outros). Em [150] um autômato temporizado TA é definido como uma tupla $\langle S, S_0, Act, R, E, Inv \rangle$, na qual:

- S É um conjunto de estados;
- $S_0 \in S$ é o estado inicial;
- Act é um conjunto de ações;
- R um conjunto finito de relógios;
- $E \subseteq S \times B(R) \times Act \times 2^R \times S$ é um conjunto de transições entre os estados;
- $Inv : S \rightarrow B(R)$ é uma função que designa invariantes (restrições de relógios) a cada estado do conjunto S .

Um conjunto A de autômatos temporizados é definido como uma sequência $A = (A_1, \dots, A_n)$ de n TA na qual se $A_i = \langle S^i, S_0^i, Act^i, R^i, E^i, Inv^i \rangle$ então para todo i e j com $1 \leq i \leq n$ e $1 \leq j \leq n$ e $i \neq j$ satisfaz $S^i \cap S^j = \emptyset$ e $R^i \cap R^j = \emptyset$; ou seja, dois autômatos diferentes presentes em um mesmo conjunto de autômatos nunca compartilharão um mesmo estado. Ou, em outras palavras, o conjunto de estados ocupado por um autômato de um conjunto de autômatos nunca terá um elemento pertencente ao conjunto de estados ocupados por outro autômato.

3.2.2 Exemplo básico de modelagem, simulação e verificação em UPPAAL

Para ilustrar os principais elementos do UPPAAL utilizados neste trabalho, considere o sistema de um único autômato e dois estados A e B ilustrado na Figura 3.1, no qual x é uma variável inteira e ta e tb são variáveis de relógio. O sistema inicia no estado A (o estado inicial é marcado com um círculo duplo).

O estado A está condicionado pela invariante $ta \leq 10$, o que significa que o sistema não poderá permanecer no estado A por mais de 10 unidades de tempo. Porém, de acordo com a condição de guarda na seta da esquerda ($ta \geq 10 \ \&\& \ x < 5$), após 10 unidades de tempo o sistema pode transitar para esse mesmo estado enquanto a variável x seja menor que 5. Nesta transição o relógio ta é reinicializado e x é incrementada em uma unidade.

Quando ta for maior ou igual a 10 e x for maior ou igual a 5 o sistema passa do estado A para o estado B, e nesta transição o relógio tb é reinicializado. O estado B contém a invariante $tb \leq 10$, isso significa que o sistema não pode permanecer por mais de 10 unidades de tempo neste estado. Dado que não há nenhuma condição de guarda para sair do estado B o sistema permanecerá neste estado por um tempo aleatório uniformemente distribuído entre 0 e 10 unidades de tempo. Note que se na seta de retorno de B para A fosse adicionada a condição de guarda $tb \geq 10$ o processo se transformaria em determinístico e sairia do estado B exatamente após 10 unidades de tempo.

Conforme pode ser visto na Figura 3.2, UPPAAL ainda facilita fazer algumas verificações acerca do sistema e de suas variáveis, por exemplo, checar se não existe nenhuma possibilidade de ocorrer *deadlock* com a sintaxe $A [] \text{not deadlok}$ (um ponto verde indica que a sintaxe se satisfaz, ou seja, não existe *deadlock*). Ou checar se alguma variável pode assumir determinados valores, como por exemplo, checar se existe pelo menos uma possibilidade da variável x ser maior que 5, com a sintaxe $E \langle \rangle x > 5$ (um ponto vermelho indica que a sintaxe não se satisfaz, ou seja, não existe nenhum valor de x maior que 5).

Também pode-se checar a probabilidade de alguma variável atingir determinados valores em um determinado período de tempo, exemplo, a probabilidade da variável x ser maior que 4 em até 100 unidades de tempo, com a sintaxe $\text{Pr} [\leq 100] \{ \langle \rangle x > 4 \}$. Para efetuar essa tarefa o UPPAAL toma como amostra populacional uma determinada quantidade de simulações feitas com sementes diferentes até poder fazer uma estimativa de um intervalo de probabilidade populacional com uma confiança de 95 %. Para isso, foram feitas 36 simulações neste caso, que foram suficientes para estimar com 95 % de confiança que a probabilidade de x ser maior que quatro em até 100 unidades de tempo está no intervalo de 0,902606 a 1 (veja a Figura 3.2).

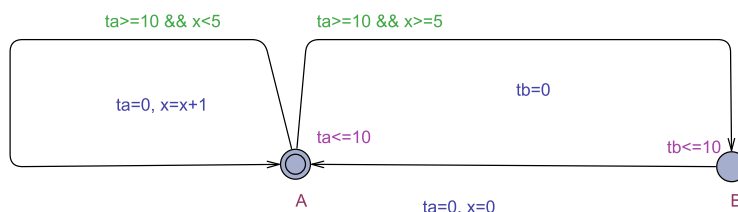


Figura 3.1: Simples autômato de dois estados em UPPAAL.

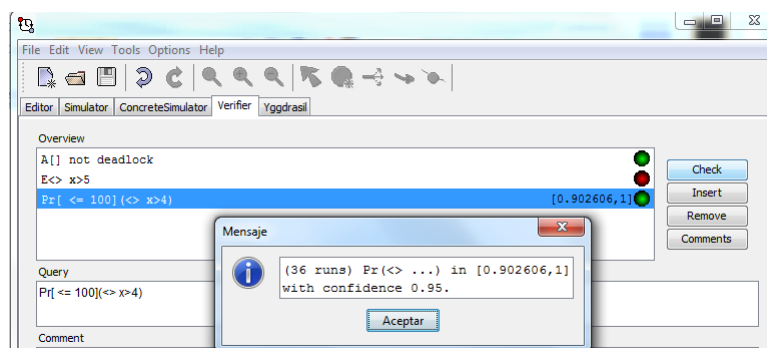


Figura 3.2: Exemplo de verificação em UPPAAL.

3.3 Modelo de uma Topologia de rede de comunicação *daisy-chain* com um fluxo TCP-Reno em UPPAAL

Neste trabalho foi desenvolvida uma metodologia de modelagem de sistemas de comunicação mediante um sistema autômato finito realizado no UPPAAL, que admite modelar, simular e verificar topologias de redes de comunicação com seus respectivos fluxos, algoritmos e protocolos. Ou seja, possibilita realizar vários níveis de modelagem, simulações e verificações em conjunto.

Para ilustrar essa metodologia de modelagem, e com a finalidade de comparar resultados obtidos em UPPAAL, primeiramente foi modelada e simulada a mesma topologia de rede de comunicação tipo *daisy-chain* com um fluxo TCP-Reno, encontrada na Seção 4.1 de [116] e ilustrada na Figura 3.3. Na qual, um transmissor transmite pacotes de 1 kbyte para um receptor através de uma rede composta por dois roteadores (chamados roteador 1 e roteador 2, respectivamente) cujas filas (ou *buffer*) possuem capacidades para enfileirar até 17 pacotes de 1 kbyte cada, e a partir do momento que a fila estiver cheia, os roteadores começam a descartar os novos pacotes que chagam na fila.

Os valores iniciais das variáveis do TCP-Reno $Ssth$ e $rwnd$, foram ajustados em 30 e 64 pacotes, respectivamente (igual a [116]). Os roteadores são conectados por meio de um canal de 1,5 Mbps de capacidade e 5 ms de atraso de propagação, chamado canal 2. Por outro lado, o transmissor é conectado com o roteador 1 por meio de um canal com capacidade de 10 Mbps e 2 ms de atraso de propagação chamado canal 1, finalmente o receptor é conectado com o roteador 2, por meio de um canal com capacidade de 10 Mbps e 33 ms de atraso de propagação chamado canal 3.

No modelo feito em UPPAAL dessa topologia de rede de comunicação, transmissor, receptor, roteadores e cada canal de comunicação foram modelados mediante autômatos temporizados, ou seja, mediante máquina de estados finitos estendidas com relógios e variáveis inteiras e flutuantes, e a topologia de rede de comunicação completa junto com seu fluxo, algoritmos e protocolos funciona como um sistema de tais autômatos temporizados em paralelo.

A Figura 3.4 ilustra o conjunto de autômatos completo que modelam a topologia de rede de comunicação da Figura 3.3. Esse modelo foi simulado por 10 s nos quais foi considerado que o transmissor TCP-Reno sempre tinha pacotes para transmitir.

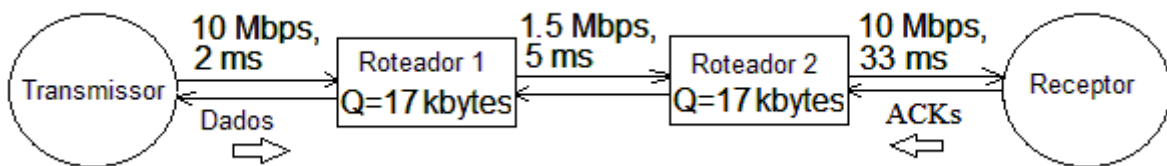


Figura 3.3: Topologia de rede de comunicação *daisy-chain* com um fluxo TCP.

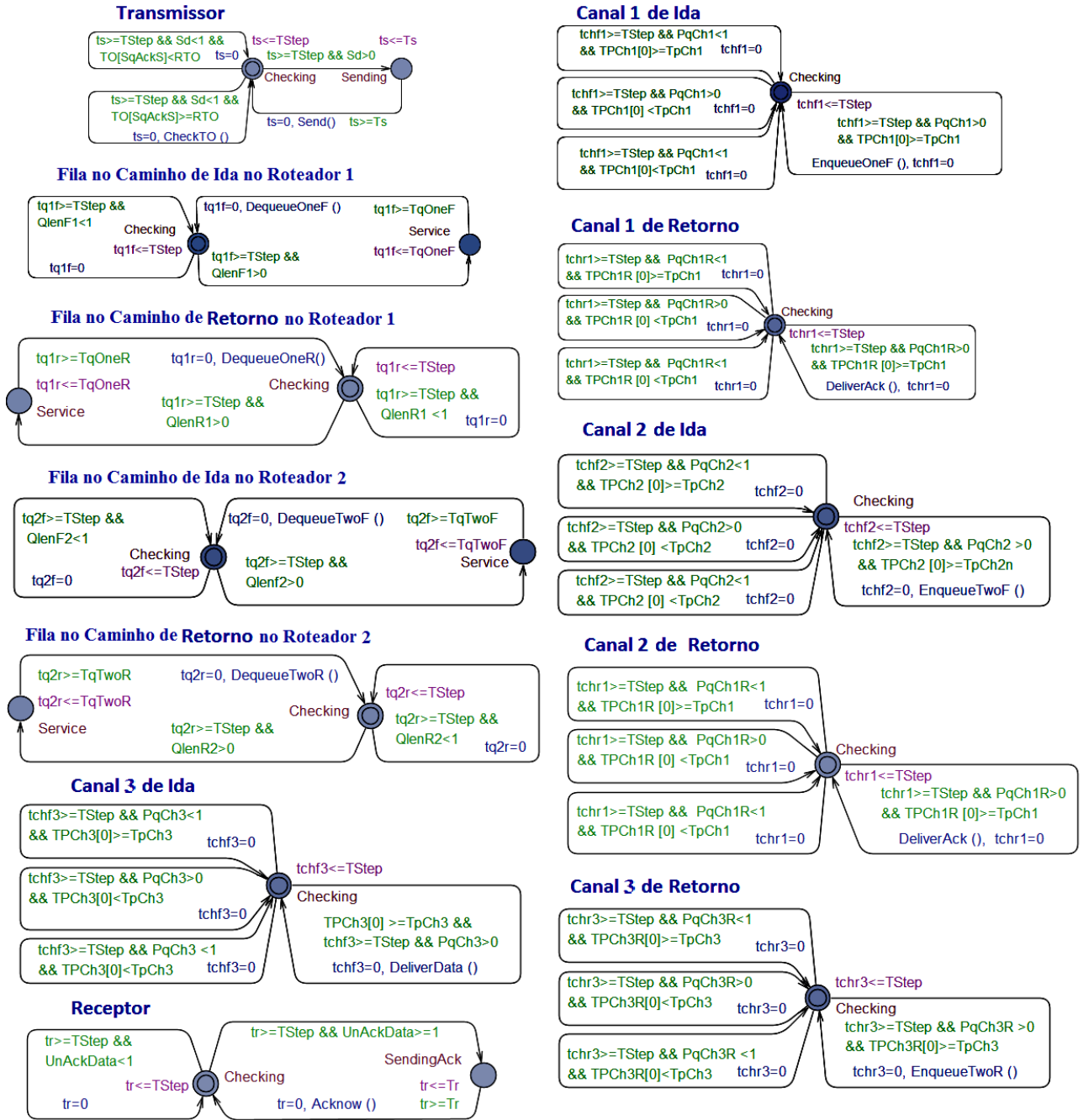


Figura 3.4: Conjunto de autômatos que modelam a topologia de rede de comunicação daisy-Chain da Figura 3.3.

A seguir serão descritos os autômatos ilustrados na Figura 3.4.

3.3.1 Modelo do Transmissor

A Figura 3.5 mostra o modelo do transmissor o qual consiste de dois estados denominados **Checking** e **Sending**, e o pseudocódigo para este autômato é dado no Algoritmo 3.1, enquanto que as variáveis utilizadas no modelo autômato temporizado são descritas na Tabela 3.1.

Conforme observa-se na Figura 3.5, o autômato inicia no estado **Checking** e permanece nele durante T_{Step} unidades de tempo. Logo se tiver pacotes para enviar, ou seja, se após um tempo T_{Step} a variável S_d for maior que zero, o autômato migra para o estado **Sending**, onde permanece por T_s unidades de tempo. Depois, transita novamente para o estado **Checking** e nessa transição chama a função **Send** e reinicia seu relógio ts .

Tabela 3.1: Descrição das variáveis, funções e constantes utilizadas no modelo do transmissor.

Nome	Descrição
T_{Step}	Passo de tempo (resolução temporal) - valor adotado $=1\mu s$.
T_s	Constante igual ao tempo em μs requerido para colocar um pacote de 1 kbyte em um canal com capacidade 10 Mbps (canal 1).
ts	Relógio que coordena as transições dos estados Checking e Sending .
S_d	Variável que contém a quantidade de pacotes que o transmissor TCP pode enviar, incrementada de acordo com o algoritmo TCP-Reno que faz a atualização cada vez que recebe um ACK, e decrementada cada vez que o transmissor envia um pacote.
T_O	Arranjo de Relógios que contenham o tempo de espera para cada pacote enviado pelo transmissor e cuja recepção ainda não foi confirmada por um Ack.
$SqAckS$	Cadeia de T_O .
CheckTO	Função chamada cada vez que ocorrer um <i>timeout</i> . Basicamente reinicia a fase de partida lenta e ajusta a sequência de pacotes para retransmitir o pacote considerado perdido.
Send	Função chamada cada vez que o transmissor envia um pacote. Basicamente decrementa a variável S_d em um pacote e incrementa a variável $PqCh1$ em um pacote
$PqCh1$	Variável que contém a quantidade de pacotes dentro do canal 1 de ida.

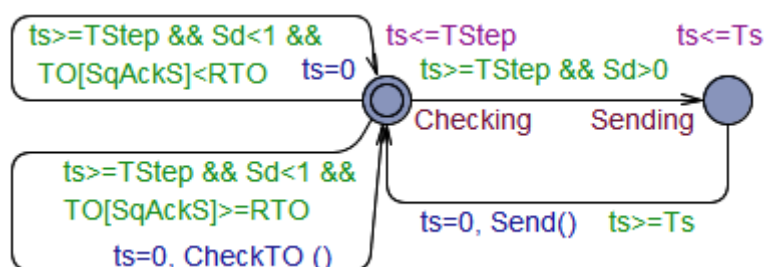


Figura 3.5: Modelo do transmissor TCP-Reno.

Algoritmo 3.1 Pseudocódigo para o autômato da Figura 3.5.

	Observações
Inicializa no estado <code>Checking</code>	
<code>if (ts >= TStep) {</code>	
<code>if (Sd > 0) { vai para o estado <code>Sending</code> espera <code>Ts</code> unidades de tempo e então retorna para o estado <code>Checking</code>, reinicia <code>ts</code> e chama a função <code>Send</code></code>	Transição do lado direito na Figura 3.5
<code>}</code>	
<code>if (Sd = 0 and TO [SqAckS] < RTO) { Faz uma transição para o mesmo estado e reinicia <code>ts</code> }</code>	Transição do lado superior esquerdo na Figura 3.5
<code>if (Sd = 0 and TO [SqAckS] >= RTO) { Faz uma transição para o mesmo estado reinicia <code>ts</code> e chama a função <code>CheckTO</code> }</code>	Transição do lado inferior esquerdo na Figura 3.5
<code>}</code>	
<code>else {espera até <code>ts >= TStep</code> ser verdadeiro}</code>	Fim do bloco (<code>ts >= TStep</code>)

Se, estando no estado `Checking`, ocorrer um *timeout*, o autômato transita para esse mesmo estado conforme se observa e na seta inferior esquerda da Figura 3.5, e nessa transição chama a função `CheckTO`, que reinicia a fase de partida lenta do TCP e retransmite o pacote considerado perdido. Se estando no estado `Checking` passar `TStep` unidades de tempo e não houver pacotes para transmitir e nem ocorrer *timeout*, então o autômato faz uma transição para esse mesmo estado e reinicia o relógio `ts` conforme ilustra a seta superior esquerda da Figura 3.5.

A variável `Sd`, além de ser atualizada pela função `Send`, também é atualizada cada vez que o transmissor receber um ACK do receptor, de acordo com o valor da janela de transmissão que evolui seguindo o algoritmo TCP-Reno. Essa dinâmica é calculada pela função `DeliverAck` chamada pelo autômato que modela o canal 1 de retorno.

3.3.2 Modelo do canal 1 de ida

A rede mostrada na Figura 3.3 contém três canais, o primeiro localizado entre o transmissor e o roteador 1 (chamado canal 1), o segundo entre o roteador 1 e o roteador 2 (chamado canal 2), e o terceiro localizado entre o roteador 2 e o receptor (chamado canal 3). Cada canal foi modelado com dois autômatos um para a ida, por onde circulam dados que vão do remetente ao receptor, e outro de retorno, por onde circulam ACKs do receptor para o transmissor.

A Figura 3.6 ilustra o autômato que modela o canal 1 de ida (localizado entre o transmissor 1 e o roteador 1), o pseudocódigo desse modelo é dado no Algoritmo 3.2, enquanto que as constantes, variáveis e funções utilizadas no modelo autômato temporizado são descritas na Tabela 3.2.

Conforme pode ser observado na Figura 3.6, esse autômato contém transições para um único estado que acontecem a cada `TStep` unidades de tempo. A transição da direita acontece cada vez que existe pelo menos um pacote no canal, ou seja, `PqCh1 > 0` for verdadeiro, e um pacote já chegou no roteador 1, isto é, `TPCh1[0] >= TpCh1` também for verdadeiro. Onde `TPCh1 [0]` é um cronômetro disparado no momento que o pacote que está no extremo do canal (chegando na fila do roteador 1) saiu do transmissor. Nessa transição é reiniciado o relógio `tchf1` e é chamada a função `EnqueueOneF` a qual basicamente modela a saída de um pacote do respectivo canal e sua entrada

na fila de ida do roteador 1.

As transições da esquerda ocorrem quando após um tempo $TStep$ não existir pacotes no canal ou se existir ainda não se propagaram até o roteador 1.

Os outros canais foram modelados de forma análoga.

Tabela 3.2: Descrição das variáveis, funções e constantes utilizadas no modelo do canal 1 de ida.

Nome	Descrição
$TStep$	Passo de tempo (resolução temporal) - valor adotado $=1\mu s$
$tchf1$	Relógio que coordena as transições do estado Checking.
$TpCh1$	Constante igual ao tempo de propagação no canal 1 (2 ms).
$TPCh1$	Arranjo de Relógios que contenham o tempo de propagação de cada um dos pacotes que já saíram do transmissor mas ainda não chegaram no roteador 1, ou seja, o tempo transcorrido desde que cada pacote saiu do transmissor até o instante atual.
$EnqueueOneF$	Função chamada cada vez que um pacote chega à fila de ida do roteador 1. Basicamente incrementa a variável $QlenF1$ em um pacote e decrementa a variável $PqCh1$ em um pacote.
$PqCh1$	Variável que contém a quantidade de pacotes dentro do canal 1 de ida.
$QlenF1$	Variável que contém a quantidade de pacotes dentro da fila de ida do roteador 1.

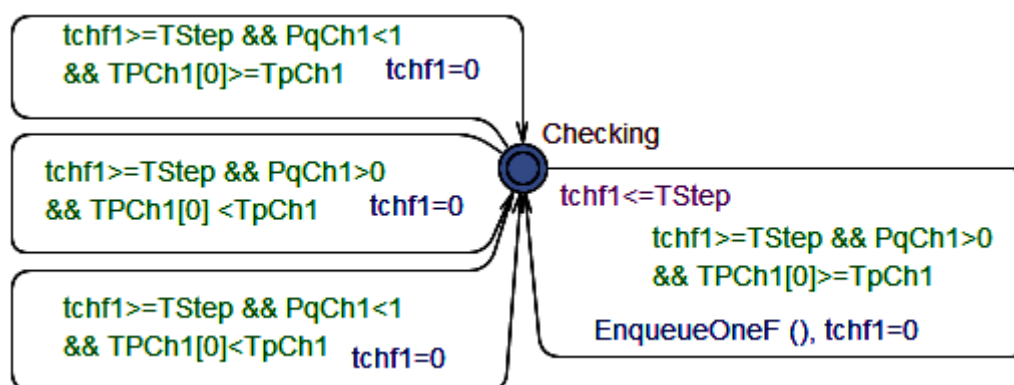


Figura 3.6: Modelo do canal 1 de ida.

Algoritmo 3.2 Pseudocódigo para o autômato da Figura 3.6.

```

Inicializa no estado Checking
if (tchf1 >= TStep) {
  if (PqCh1 > 0 and TPCh1 [0] >= TpCh1)
  { Faz uma transição para o mesmo estado reinicia tchf1 e chama a
  função EnqueueOneF }
  if (PqCh1 < 1 or TPCh1 [0] < TpCh1)
  { Faz uma transição para o mesmo estado e reinicia tchf1 }
}

else {espera até tchf1 >= TStep ser verdadeiro }

```

Observações

Transição do lado direito na Figura 3.6

Transições do lado esquerdo na Figura 3.6

Fim do bloco ($tchf1 \geq TStep$)

3.3.3 Modelo da fila no caminho de ida no roteador 1

A topologia de rede de comunicação mostrada na Figura 3.3 contém dois roteadores. Cada roteador foi modelado com dois autômatos, um autômato para modelar a fila no caminho de ida, onde são enfileirados os pacotes de dados que viajam do transmissor para o receptor, e outro para modelar a fila no caminho de retorno, onde são enfileirados os pacotes de ACKs que viajam do receptor para o transmissor.

A Figura 3.7 ilustra o autômato que modela a fila do caminho de ida no roteador 1, o qual consiste de dois estados, denominados **Checking** e **Service**. O pseudocódigo para este autômato é dado no Algoritmo 3.3, enquanto que as constantes, variáveis e funções utilizadas no modelo autômato temporizado são descritas na Tabela 3.3.

Conforme pode ser observado na Figura 3.7, o autômato inicia no estado **Checking** e permanece nele durante $TStep$ unidades de tempo. Logo, se tiver pacotes na fila, ou seja, se $QlenF1 \geq 1$ for verdadeiro, o autômato migra para o estado **Service**, conforme ilustra a seta inferior direita na Figura 3.7. No estado **Service** o autômato permanece por $TqOneF$ unidades de tempo (igual ao tempo necessário para colocar um pacote de 1 kbyte em um canal de 1,5 Mbps). Depois, o autômato retorna para o estado **Checking** e nesta transição de retorno chama a função `DequeueOneF`, e reinicia seu relógio `tq1f` conforme ilustra a seta superior para a esquerda na Figura 3.7.

Se estando no estado **Checking** passar $TStep$ unidades de tempo, ou seja, $tq1f \geq TStep$ for verdadeiro, e não houver pacotes na fila, ou seja, $QlenF1 < 1$ também for verdadeiro, então o autômato faz uma transição para esse mesmo estado e reinicia o relógio `tq1f` conforme ilustra a seta esquerda da Figura 3.7.

As outras filas foram modeladas de forma análoga.

Tabela 3.3: Descrição das variáveis, funções e constantes utilizadas no modelo da fila de ida no roteador 1.

Nome	Descrição
<code>TStep</code>	Passo de tempo (resolução temporal) - valor adotado $=1\mu s$.
<code>tq1f</code>	Relógio que coordena as transições dos estados Checking e Service .
<code>TqOneF</code>	Constante igual ao tempo em μs requerido para colocar um pacote de 1 kbyte em um canal com capacidade 1,5 Mbps (canal 2).
<code>DequeueOneF</code>	Função chamada cada vez que um pacote sai da fila de ida do roteador 1. Basicamente decrementa <code>QlenF1</code> em um pacote e incrementa a variável <code>PqCh2</code> em um pacote.
<code>QlenF1</code>	Variável que contém a quantidade de pacotes dentro da fila de ida do roteador 1.
<code>PqCh2</code>	Variável que conta a quantidade de pacotes dentro do canal 2 de ida.

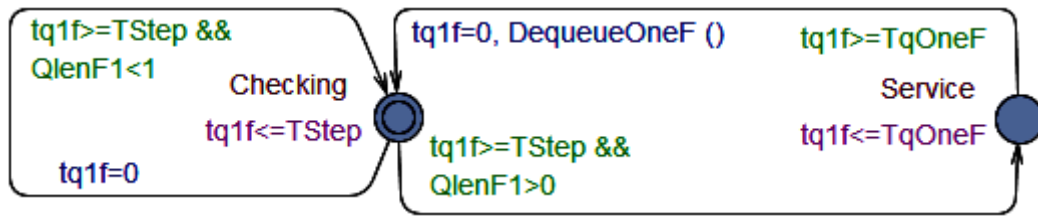


Figura 3.7: Modelo da fila de dados no caminho de ida no roteador 1.

Algoritmo 3.3 Pseudocódigo para o autômato da Figura 3.7.

```

Inicializa no estado Checking
if (tq1f >= TStep) {
  if (QlenF1 > 0) { vai para o estado Service espera TqOneF unidades de
  tempo e então retorna para o estado Checking, reinicia tq1f e chama a
  função DequeueOneF }
  if (QlenF1 < 1) { Faz uma transição para o mesmo estado e reinicia
  tq1f }
}
else { espera até tq1f >= TStep ser verdadeiro }

```

Observações

Transição do lado direito na Figura 3.7

Transição do lado esquerdo na Figura 3.7

Fim do bloco (tq1f >= TStep)

3.3.4 Modelo do Receptor

Figura 3.8 ilustra o autômato que modela o receptor, o pseudocódigo desse modelo é dado no Algoritmo 3.4, enquanto que as constantes, variáveis e funções utilizadas no modelo autômato temporizado são descritas na Tabela 3.4.

Conforme pode ser observado na Figura 3.8 esse autômato contém dois estados denominados **Checking** e **SendingAck**. O autômato começa no estado **Checking** e cada vez que chegar um pacote, isto é, quando $UnAckP > 0$ for verdadeiro o autômato migra do estado **Checking** para o estado **SendingAck** onde permanece por T_s unidades de tempo. Depois, o autômato retorna para o estado **Checking**, e nessa transição de retorno chama a função **AckNow**.

Se o autômato estiver no estado **Checking** e após um tempo T_{Step} não haver novos pacotes de dados cujos ACKs ainda não foram enviados, ou seja, $UnAckP < 1$ for verdadeiro, então o autômato faz uma transição para esse mesmo estado e reinicia seu relógio tr , conforme ilustra a seta da esquerda na Figura 3.8.

Tabela 3.4: Descrição das variáveis, funções e constantes utilizadas no modelo do receptor.

Nome	Descrição
TStep	Passo de tempo (resolução temporal) - valor adotado $=1\mu s$.
tr	Relógio que coordena as transições dos estados Checking e SendingAck .
Tr	Limiar de tempo requerido para colocar um ACK em um canal com capacidade 10 Mbps (canal 3).
UnAckP	Variável que contém a quantidade de pacotes que chegaram ao receptor e cujos ACKs ainda não foram enviados.
AckNow	Função chamada cada vez que o receptor envia um ACK. Basicamente calcula a sequência do ACK a ser enviado e decrementa a variável UnAckP em um pacote e incrementa a variável PqCh3R em um pacote.
UnAckP	Variável que contém a quantidade de pacotes que já chegaram no receptor e cujo ACK ainda não tem sido enviado.
PqCh3R	Variável que contém a quantidade de ACKs dentro do canal 3 de retorno.

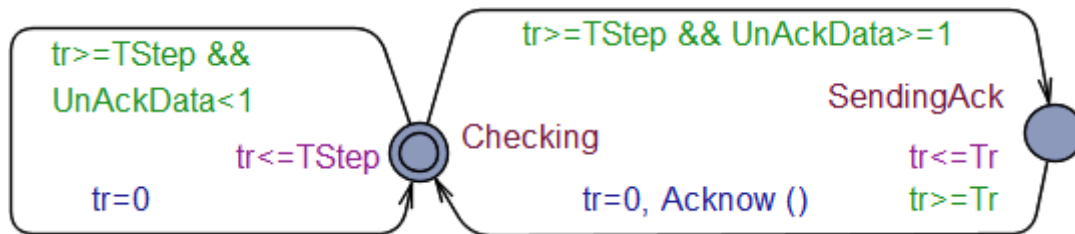


Figura 3.8: Modelo do receptor.

Algoritmo 3.4 Pseudocódigo para o autômato da Figura 3.8.

```

Inicializa no estado Checking
if (tr >= TStep) {
if (UnAckP > 0) { Vai para o estado SendingAck espera Tr unidades de
tempo e então retorna para o estado Checking, reinicia tr e chama a
função Acknow }
if (UnAckP < 1) { Faz uma transição para o mesmo estado para
reiniciar tr}
}
else { espera até tr >= TStep ser verdadeiro }

```

Observações

Transição do lado direito na Figura 3.8

Transição do lado esquerdo na Figura 3.8

Fim do bloco if (tr >= TStep)

3.4 Simulação do modelo TCP-Reno em UPPAAL

Os resultados das simulações em UPPAAL podem ser observados na Figura 3.9 (janela de congestionamento TCP-Reno) e na Figura 3.10 (tamanho da fila no roteador 1). Uma vez que a fila está cheia (17 pacotes), o roteador começa a descartar os novos pacotes que chegam na fila. Logo, o receptor recebe pacotes fora de sequência e começa a enviar ACKs repetidos solicitando o primeiro pacote descartado.

Ao receber três ACKs repetidos, o transmissor retransmite o primeiro pacote perdido e reduz sua janela de transmissão pela metade. Esse procedimento produz oscilações tanto no tamanho da janela de transmissão (Figura 3.9) quanto no tamanho da fila no roteador 1 (Figura 3.10). Note que enfileiramentos somente ocorrem no roteador 1 já que na sua saída se encontra o canal com menor capacidade (1,5 Mbps) que é o gargalo nesta topologia (Figura 3.3).

Os resultados apresentados nas Figuras 3.9 e 3.10 obtidos em UPPAAL, conferem com os resultados apresentados na Seção IV de [116], já clássicos, obtidas no simulador NS-2. Verificando-se assim que uma topologia de rede de comunicação, junto com seus protocolos, pode ser modelada como um conjunto de autômatos temporizados. E, como se verá no próximo capítulo, essa modelagem por autômatos temporizados facilita o estudo de sistemas de controle que usam a Internet.

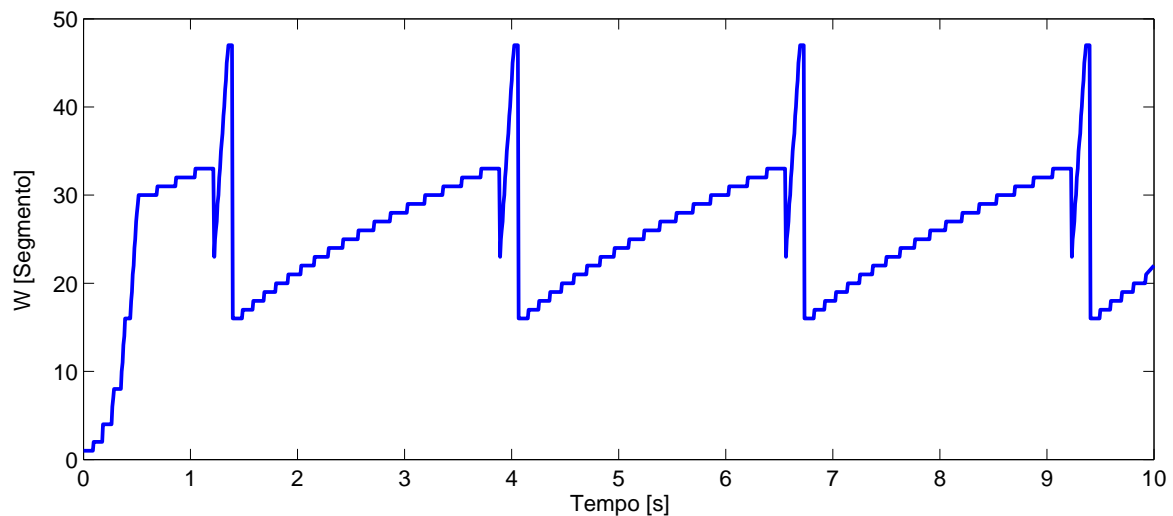


Figura 3.9: Janela do Transmissor TCP-Reno para Drop Tail implementada na topologia de rede de comunicação daisy-chain.

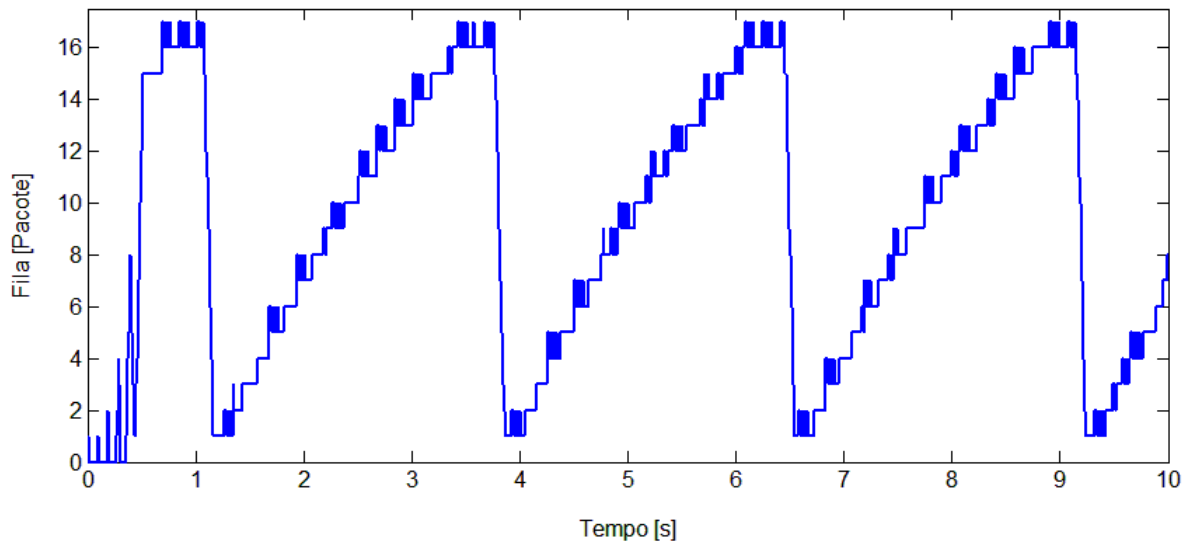


Figura 3.10: Fila no caminho de ida no roteador 1 para Drop Tail implementada na topologia de rede de comunicação daisy-chain.

3.5 Modelos de Técnicas AQM em UPPAAL

Além de Drop Tail, outras técnicas AQM também foram modeladas e simuladas no UPPAAL, mediante a metodologia de modelagem desenvolvida neste trabalho, entre elas RED, CoDel e PIE. Todas essas simulações foram feitas considerando que todos os pontos do sistema de comunicação (transmissor, receptor e roteadores) são capazes de trabalhar com ECN. Assim, quando uma técnica AQM julgar que a fila está congestionada, mudará o estado do bit CE do cabeçalho IP ao invés de descartar o pacote. Logo, quando o receptor receber esse pacote de dados marcado, enviará um ACK marcado com o bit ECE do cabeçalho TCP para transmitir essa notificação para o transmissor informando-o acerca do congestionamento no caminho. Quando, finalmente, o transmissor receber esse ACK marcado reduzirá sua janela de congestionamento (*cwnd*) antes da fila no roteador encher. Porém, no TCP-Reno, mesmo que o transmissor receba pacotes marcados consecutivos somente reduzirá sua janela uma vez por RTT.

Nesta seção, serão apresentados os resultados das simulações das técnicas AQM RED, CoDel e PIE, realizadas no UPPAAL utilizando a topologia de rede de comunicação daisy-chain ilustrada na Figura 3.3. Detalhes acerca da modelagem dessas técnicas no UPPAAL estão descritas no Anexo I.

As Figuras 3.11 e 3.12 apresentam a dinâmica da Janela do TCP-Reno e do tamanho da fila no roteador 1, respectivamente, para uma implementação RED com os seguintes parâmetros $W_q = 0,002$, $P_{max} = 0,1$, $min_{th} = 4$ pacotes e $max_{th} = 12$ pacotes [104].

Como pode ser observado, uma única vez (logo após o primeiro segundo de simulação) a fila encheu e um pacote foi descartado. Porém, quando RED entrou em regime, pacotes foram marcados antes da fila encher e consequentemente o transmissor atuou com antecipação em sua janela de

transmissão e não ocorreram novas perdas de pacotes.

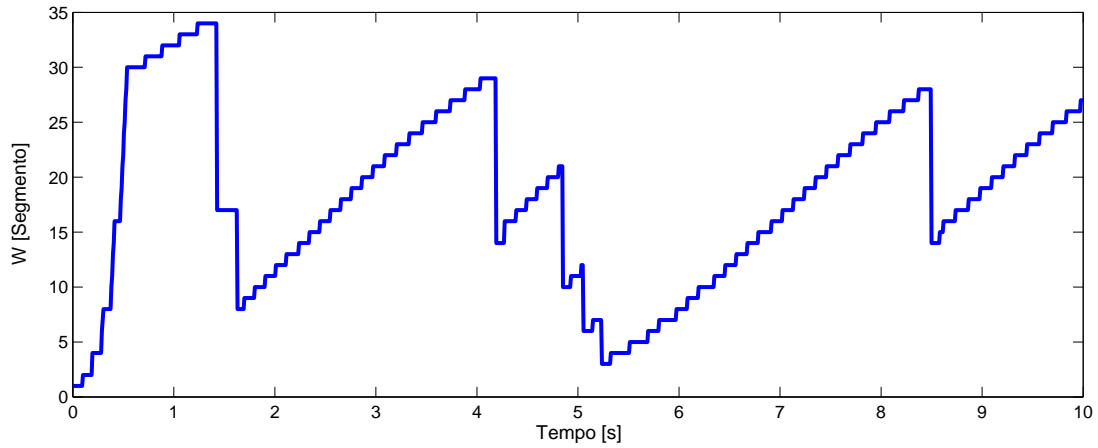


Figura 3.11: Janela do Transmissor TCP-Reno para RED implementado na Topologia de rede de comunicação daisy-chain.

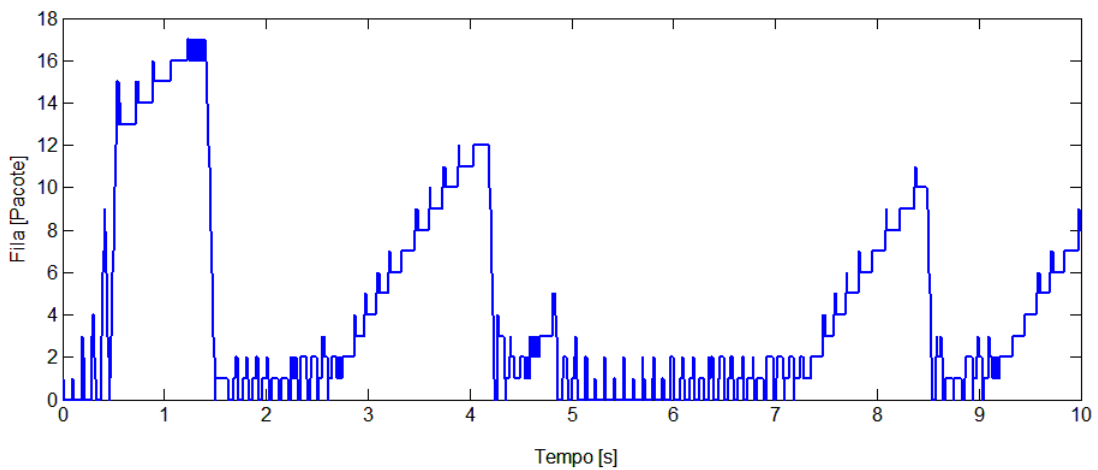


Figura 3.12: Fila no caminho de ida do roteador 1 para RED implementado na Topologia de rede de comunicação daisy-chain.

As Figuras 3.13 e 3.14 apresentam a dinâmica da Janela do TCP-Reno e do tamanho da fila no roteador 1, respectivamente, para a técnica AQM CoDel. Finalmente, as Figuras 3.15 e 3.16 elucidam essas mesmas dinâmicas para a técnica AQM PIE.

Conforme pode ser observado, RED apresenta menos oscilações na fila quando comparado a Drop Tail (Figuras 3.12 e 3.10, respectivamente), entretanto, CoDel e PIE conseguem reduzir ainda mais essas oscilações, fornecendo assim melhor desempenho em termos de estabilidade do tamanho da fila.

Por outro lado, diferentemente de Drop Tail e RED, CoDel e PIE mantêm a fila em torno a valores pequenos, ou seja, evitam o indesejado fenômeno de *bufferbloat*.

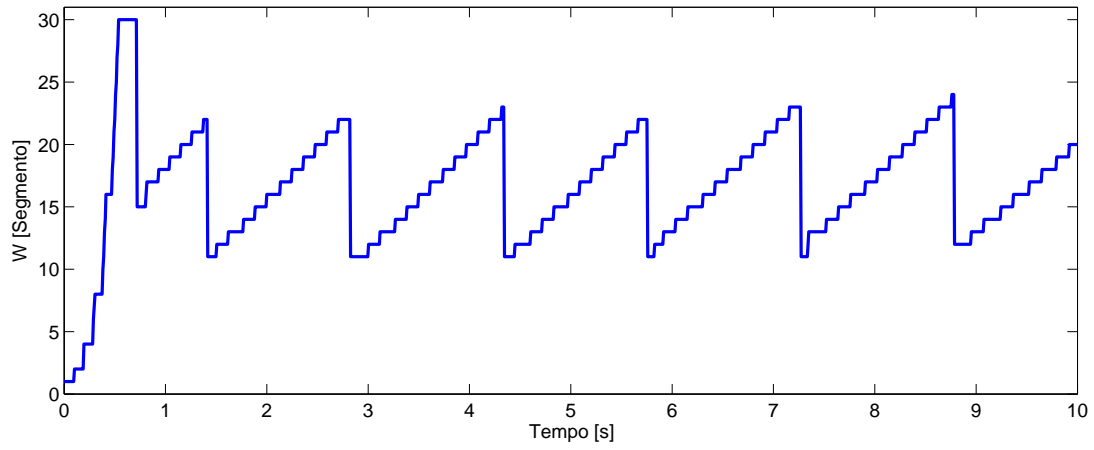


Figura 3.13: Janela do Transmissor TCP-Reno para CoDel implementado na topologia de rede de comunicação daisy-chain.

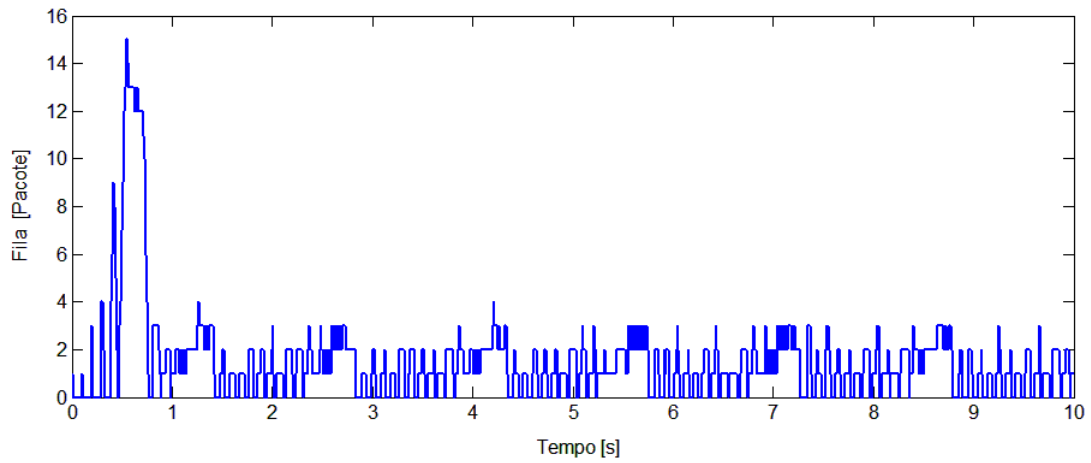


Figura 3.14: Fila no caminho de ida do roteador 1 para CoDel implementado na topologia de rede de comunicação daisy-chain.

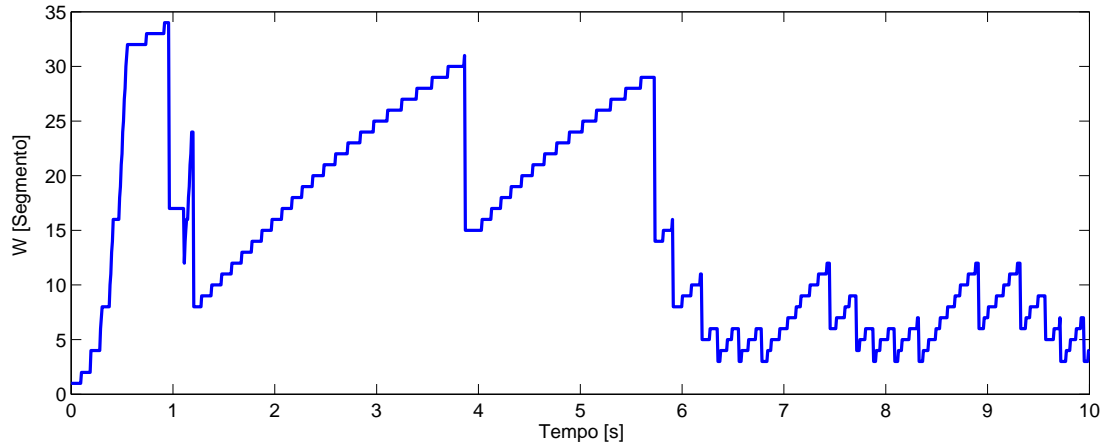


Figura 3.15: Janela do Transmissor TCP-Reno para PIE implementada na topologia de rede de comunicação daisy-chain.

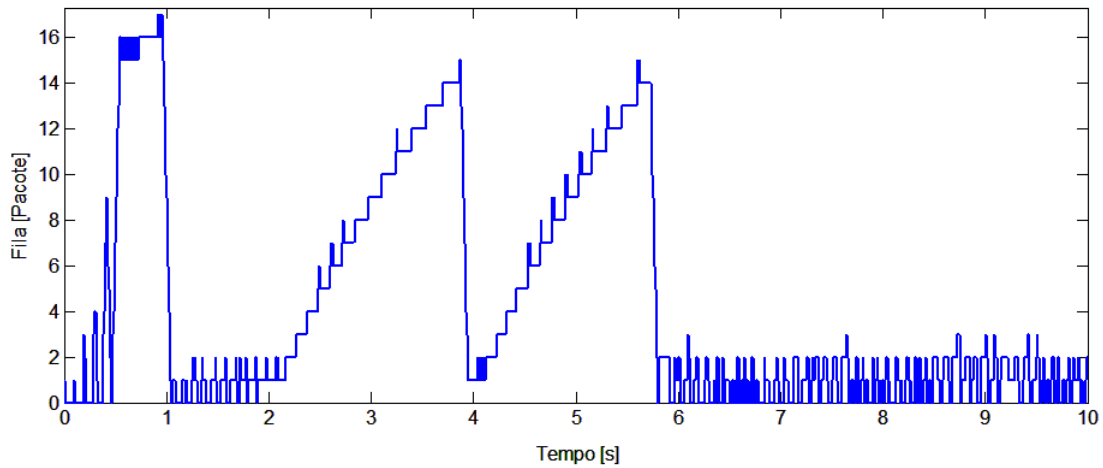


Figura 3.16: Fila no caminho de ida do roteador 1 para PIE implementada na topologia de rede de comunicação daisy-chain.

3.5.1 Vazão

No presente trabalho, vazão foi definida como:

$$vazão = pqSize \frac{N_p}{t}, \quad (3.1)$$

onde N_p é a quantidade de pacotes que foram transmitidos e cuja recepção foi confirmada até o instante t , e $pqSize$ é o tamanho em bits do pacote cuja recepção está sendo confirmada no instante t .

Ou seja, vazão será a quantidade bits corretamente transmitidos cuja recepção tenha sido confirmada por unidade de tempo. E essa será uma das métricas a ser utilizadas neste trabalho

para comparar o desempenho de diferentes protocolos de transporte de dados e técnicas AQM.

3.5.2 Fenômeno de *bufferempty*

Além da estabilidade na fila e do indesejado fenômeno de *bufferbloat*, outro fenômeno interessante pode ser observado nas Figuras 3.12, 3.14 e 3.16. Notas-se que tanto RED como CoDel e PIE apresentam diversos períodos de tempo nos quais a fila no roteador permanece completamente vazia. Esse evento, foi observado neste trabalho e denominado fenômeno de “*bufferempty*” e é um fenômeno contrário ao fenômeno de *bufferbloat*.

As técnicas AQM CoDel e PIE, por exemplo, foram projetadas para evitar *bufferbloat*, ou seja, possuem o objetivo de evitar demasiado crescimento nas filas dos roteadores afim de reduzir atrasos ponta a ponta e, conseqüentemente, em algumas aplicações elas acabam reduzindo tanto o tamanho da fila que resultam períodos nos quais a fila fica totalmente vazia, não aproveitando assim o uso da banda do canal.

Para ilustrar melhor esse fenômeno de *bufferempty*, na Figura 3.18 é apresentado um zoom capturando os primeiros 0,5 s da fila no gargalo quando implementado a técnica CoDel. Observam-se vários períodos ociosos, ou seja, períodos nos quais não existe nenhum pacote na fila do roteador. O tamanho da fila inclui o pacote que está no atendimento, assim, se a fila estiver vazia significa que o módulo de saída desta porta do roteador está ocioso.

Assim como o fenômeno de *bufferbloat* é indesejado por aumentar os atrasos (aumentando o tempo de espera nas filas), o fenômeno de *bufferempty* também é indesejado por reduzir a vazão, conforme pode ser observado na Figura 3.17. Isso acontece porque, nestes períodos de *bufferempty*, mesmo que os transmissores possuam pacotes de dados para transmitir e todo o meio de comunicação existente entre transmissor e receptor esteja livre para efetuar essa transmissão, o trabalho em conjunto do protocolo de transporte de dados e da técnica AQM utilizada impede o transmissor de enviar pacotes.

Então, o fenômeno de *bufferempty* acaba sendo mais comum nas técnicas AQM projetadas para evitar *bufferbloat*. Conforme pode ser observado na Figura 3.10, quando foi utilizada a técnica Drop Tail, ocorreram várias perdas de pacotes na fila por transbordamento do *buffer*. Já quando foi utilizada a técnica RED, perdas de pacotes por transbordamento do *buffer* ocorreram uma única vez (entre o primeiro e o segundo segundo de simulação (Figura 3.12)). Já quando utilizadas as técnicas CoDel e PIE, não houve nenhuma perda de pacote por transbordamento do *buffer* (veja as Figuras 3.14 e 3.16).

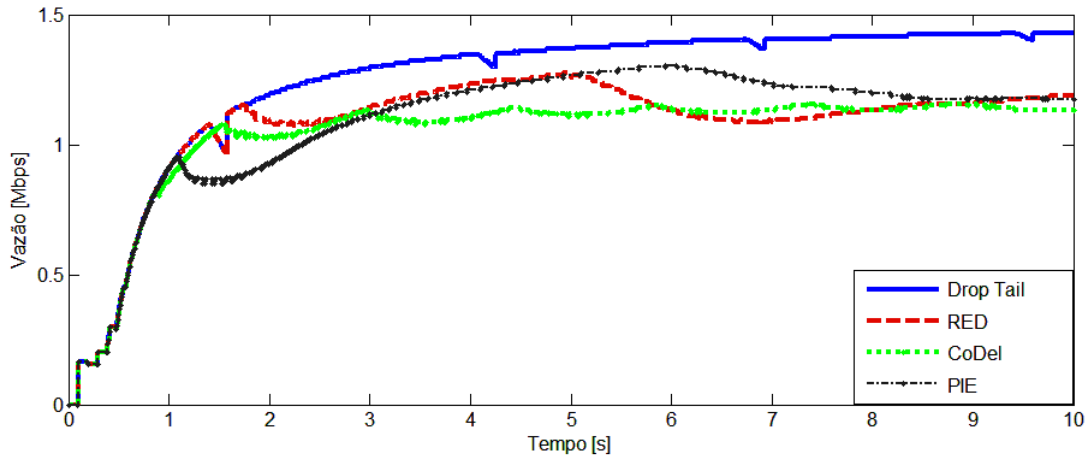


Figura 3.17: Vazão para diferentes técnicas AQM com TCP Reno, implementadas na topologia de rede de comunicação daisy-chain .

Porém, mesmo assim, conforme pode ser visto na Figura 3.17, as técnicas CoDel e PIE apresentam a pior vazão entre as técnicas comparadas. Esse pobre desempenho em termos de vazão apresentado por CoDel e PIE não foi consequência de perdas e nem de descarte de pacotes (já que esses eventos não ocorreram) mas foi consequência do fenômeno de *bufferempty* causado por um mal uso dos recursos de comunicação disponíveis.

Na Figura 3.17 pode ser observado, como a técnica AQM Drop Tail, apesar de ter maior número de perdas de pacotes apresenta melhor vazão que as outras técnicas analisadas. Isso se deve a que Drop Tail não apresenta o indesejado fenômeno de *bufferempty* (veja a Figura 3.10) , mas obviamente, por sua vez, Drop Tail apresenta outros fenômenos indesejados entre eles *bufferbloat* conforme observado na Figura 3.10, e sincronização global, descrito em [104].

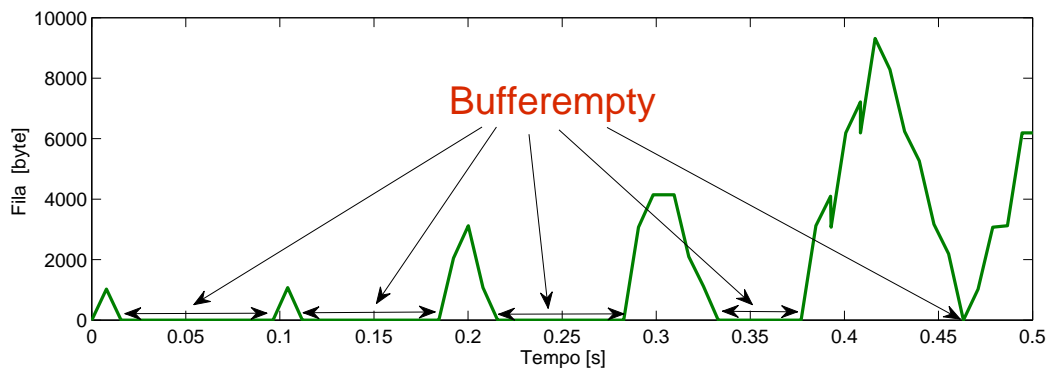


Figura 3.18: Períodos de *bufferempty* causado pela implementação de CoDel na topologia de rede de comunicação daisy-chain.

3.6 Modelagem de protocolos de transporte de dados baseados em ECN em UPPAAL

Baseado em um modelo análogo ao apresentado na seção 3.3 (e modificando as funções chamadas pelos autômatos) também foram modelados no UPPAAL os protocolos de transporte de dados TCP-Jersey [137], DCTCP [138] e E-DCTCP [139]. Sendo o TCP-Jersey um dos primeiros protocolos baseados em ECN, o DCTCP uma proposta recente que foi muito referenciada nos últimos anos [138, 140, 141, 139, 142] e o E-DCTCP uma das variantes mais recentes do DCTCP. Esses protocolos foram escolhidos com o propósito de fazer comparações com o desempenho de ENCN e outras metodologias propostas neste trabalho que serão apresentadas nos próximos capítulos.

Nesta seção, serão apresentados os resultados das simulações do modelo utilizando a topologia de rede de comunicação daisy-chain ilustrada na Figura 3.3.

As Figuras 3.19 e 3.20 apresentam a dinâmica da Janela do TCP-Jersey e do tamanho da fila no roteador 1, respectivamente. Observa-se que durante aproximadamente os primeiros sete segundos de simulação, ocorrem tanto o fenômeno de *bufferempty* (que desaparece aproximadamente aos quatro segundos) quanto o fenômeno de *bufferbloat* devido a grandes oscilações no tamanho da fila. Logo em seguida, TCP-Jersey entra em regime e as oscilações no tamanho da Janela e no tamanho da fila no roteador diminuem e a fila se mantém com pequenas oscilações em torno de 10 pacotes (de 1 kbyte cada).

Os resultados para E-DCTCP são apresentados nas Figuras 3.21 (dinâmica da Janela W) e 3.22 (tamanho da fila no roteador 1). E-DCTCP praticamente não apresenta o fenômeno de *bufferbloat* (o tamanho máximo da fila no roteador foi de apenas sete pacotes de 1 kbyte cada). Comparado com TCP-Jersey, o E-DCTCP ainda conseguiu ser mais rápido para entrar em regime, isso ocorreu logo após os três segundos de simulação.

Durante o período transitório, aproximadamente os três primeiros segundos, o E-DCTCP apresentou vários longos períodos de *bufferempty* consequentemente ao longo dos primeiros dez segundos de simulação o E-DCTCP transmitiu menos pacotes que o TCP-Jersey, ou seja, como consequência do fenômeno de *bufferempty* o E-DCTCP apresentou pior vazão que o TCP-Jersey conforme pode ser observado na Figura 3.23.

Esses resultados são indícios de que provavelmente existe uma compensação entre os fenômenos de *bufferempty* e *bufferbloat*, ou seja, algoritmos que causam mais *bufferbloat* tendem a causar menos *bufferempty* e vice-versa.

Devido a não ocorrência de perdas de pacotes por transbordamento do *buffer*, nesta topologia de rede obteve-se exatamente os mesmos resultados para DCTCP e E-DCTCP.

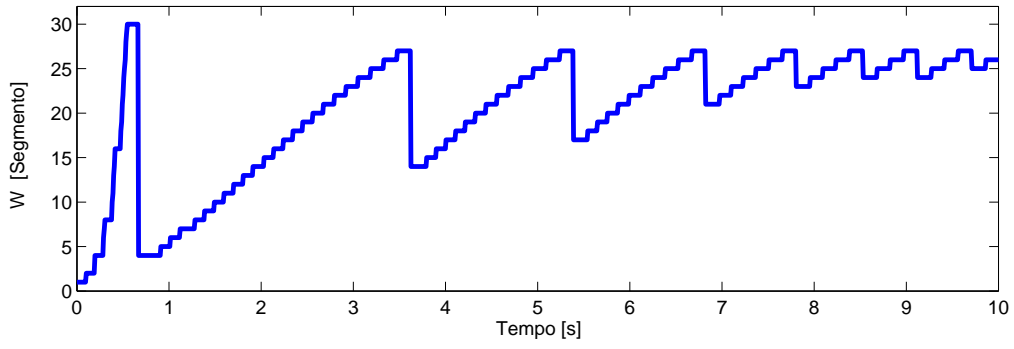


Figura 3.19: Janela do Transmissor TCP-Jersey na topologia de rede de comunicação daisy-chain.

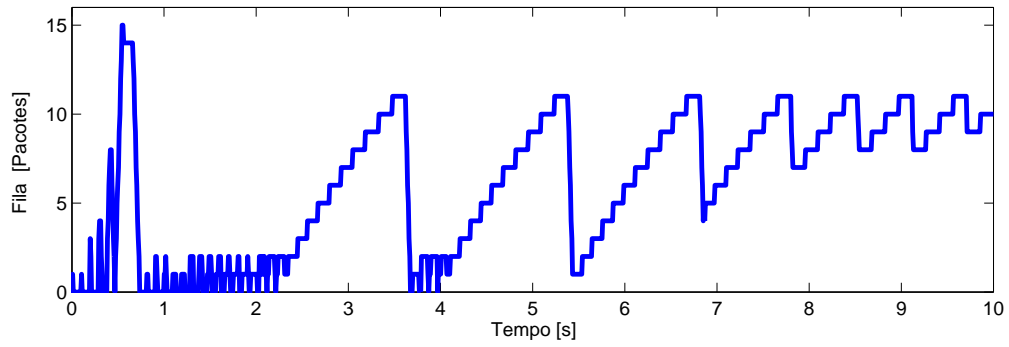


Figura 3.20: Fila no caminho de ida do roteador 1 para TCP-Jersey implementado na topologia de rede de comunicação daisy-Chain.

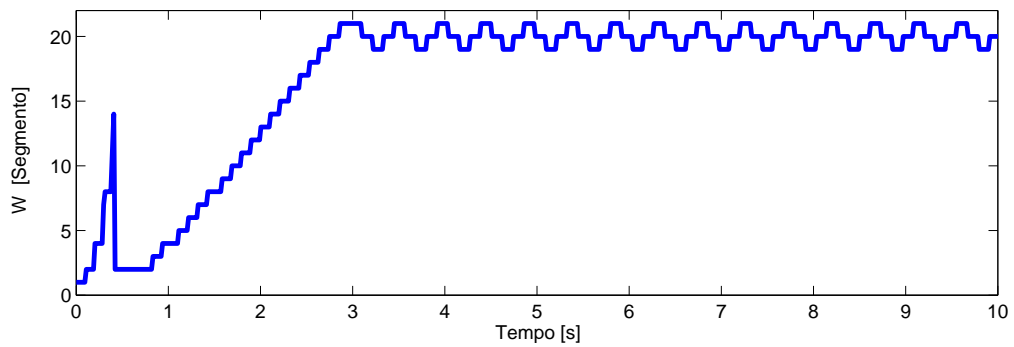


Figura 3.21: Janela do Transmissor E-DCTCP implementado na topologia de rede de comunicação daisy-chain.

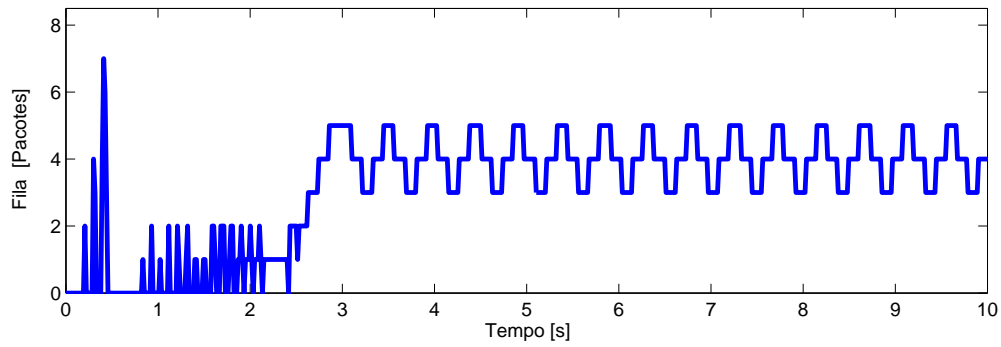


Figura 3.22: Fila no caminho de ida do roteador 1 para E-DCTCP implementado na topologia de rede de comunicação daisy-chain.

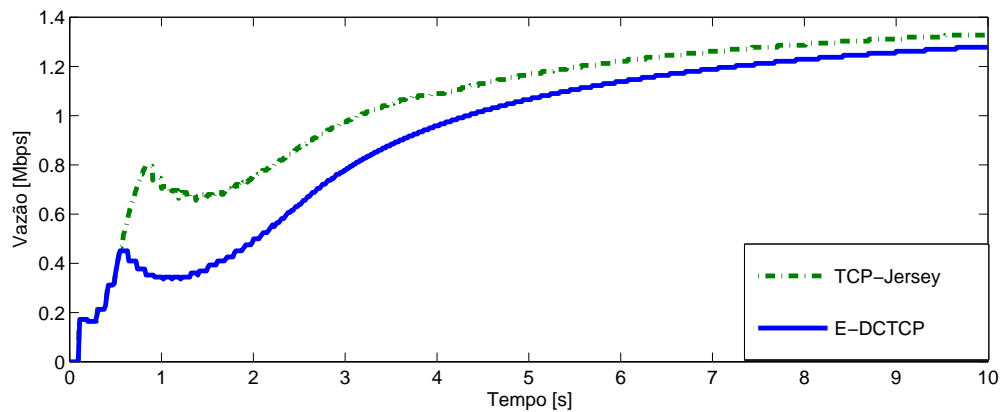


Figura 3.23: Vazão para TCP-Jersey e E-DCTCP implementados na Topologia de rede de comunicação daisy-chain .

3.7 Modelagem de NCSs em UPPAAL

A metodologia de modelagem de redes de comunicações implementada neste trabalho possibilita fazer simulações e verificações conjuntas de diferentes tipos de fluxos, assim como fazer análises estatísticas do desempenho de sistemas de controle através da Internet.

Para ilustrar essa utilidade, nesta seção serão apresentados estudos feitos mediante modelagem, simulações e verificações em UPPAAL de uma NCS, na qual um controlador controla uma planta remota através da Internet mediante um fluxo UDP. Logo, na mesma topologia de rede de comunicação serão adicionados outros fluxos TCP-Reno, e se efetuará o estudo das influências mútuas entre ambos tipos de fluxos com diferentes técnicas AQM implementadas nos roteadores.

Para essa finalidade, foi utilizado o exemplo 2 dado em [107], no qual um controlador proporcional integral (PI) com $Kp = 11,86$ e $Ki = 47,45$ é utilizado para controlar a posição de um motor CC Maxon F2140 (ilustrado na Figura 3.24) através da Internet. A função de transferência do motor, identificada experimentalmente, é dada por

$$H(s) = \frac{36,3}{s^2 + 36,17s}. \quad (3.2)$$

Observe que a função de transferência tem um pólo na origem do plano s . Logo, o erro em malha fechada para degraus de referência seria nulo, e não seria necessário utilizar a parte integral do controlador. Porém, para rejeitar perturbações constantes por partes do ambiente ou introduzidas pelo sistema de comunicação, tais como atrasos e perdas de pacotes, faz-se necessário o controle PI.



Figura 3.24: Motor CC Maxon F2140.

O modelo discreto do motor foi obtido para um período de amostragem $h = 0,014$ s. Motor e controlador PI, assim como o sistema de comunicação utilizado para a troca de dados, foram modelados mediante autômatos temporizados em UPPAAL, resolvendo-se a equação a diferenças correspondente a cada h segundos. Note que o passo de cálculo do UPPAAL adotado é de $1 \mu s$. Detalhes do modelo podem ser vistos no Anexo II.

Uma vez desenvolvido o modelo UPPAAL, foram feitas simulações e verificações estatísticas em diferentes cenários, cujos resultados destacam:

- A influência do sistema de comunicação no desempenho do sistema de controle (causado principalmente por atrasos e perdas de pacotes);
- A influência de outros fluxos que compartilham a mesma topologia de rede de comunicação com uma NCS;
- A influência de diferentes técnicas AQM em parâmetros de desempenho dos fluxos de dados que compartilham a mesma topologia de rede de comunicação.

3.7.1 Influência do sistema de comunicação no desempenho do sistema de controle

Para estudar a influência do sistema de comunicação no sistema de controle foram simulados 3 cenários diferentes, descritos a seguir:

- Cenário 1: O controlador foi colocado junto a uma planta, ou seja, não existe sistema de comunicação e conseqüentemente tampouco existem atrasos e/ou perdas de pacotes (exceto um pequeno atraso que ocorre no controlador no cálculo da lei de controle);
- Cenário 2: Planta e controlador trocam dados através da Internet por meio da topologia de rede de comunicação (NCS) ilustrada na Figura 3.25;
- Cenário 3: Uma taxa de perdas de pacotes aleatórias de 33 % é adicionada na NCS da Figura 3.25;

Nos três cenários, o motor segue uma referência R constante de 1 rad.

A transmissão de dados entre controlador e planta é realizado mediante o uso do protocolo UDP. A vantagem de utilizar UDP em NCS ao invés de utilizar TCP, está no fato de que UDP é mais simples e não faz retransmissões de pacotes perdidos, o que de fato não é necessário na maioria das aplicações NCSs, já que conforme foi visto na Seção 2.2, quando o controlador recebe uma informação nova do estado da planta, a informação passada pode ser descartada. De forma análoga, quando a planta receber uma informação nova da lei de controle a informação passada pode ser descartada. A técnica AQM utilizada nos Cenários 2 e 3 é Drop Tail, ou seja, os roteadores somente descartam pacotes uma vez que a fila estiver cheia.

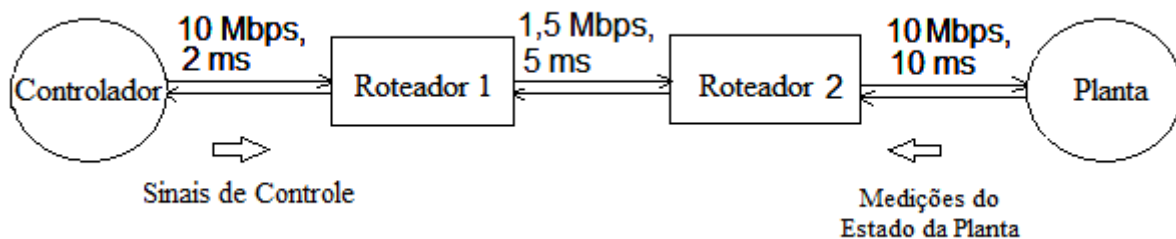


Figura 3.25: NCS através da Internet.

Análise dos resultados

As Figuras 3.26, 3.27 e 3.28 ilustram a posição do motor em função do tempo para os Cenários 1, 2 e 3, respectivamente. Conforme pode ser observado, em todos os cenários ocorre um erro, ou seja, a posição do motor difere em menor ou maior medida da referência R que deveria ser seguida. Graficamente é possível observar que o erro é muito maior para o Cenário 2 do que para o Cenário 1, ou seja, a presença de um sistema de comunicação entre controlador e planta introduziu atrasos e esses atrasos afetaram bastante o desempenho do sistema de controle.

Comparando os Cenários 2 e 3 (Figuras 3.27 e 3.28) observamos que o erro é ligeiramente maior no Cenário 3 do que no Cenário 2, ou seja, as perdas de pacotes afetam negativamente o desempenho do sistema de controle. Porém, a diferença é muito maior do Cenário 1 para o Cenário 2 (Figuras 3.26 e 3.27) do que do Cenário 2 para o Cenário 3 (Figuras 3.27 e 3.28). Assim, atrasos geraram maior degradação no desempenho do sistema de controle do que as perdas de pacotes.

As observações gráficas elucidam bastante bem as diferenças entre os diferentes Cenários, porém, com o objetivo de fazer observações mais precisas foi feito também uma análise numérica dessas diferenças, para essa finalidade o erro do sistema de controle foi quantificado utilizando a integral do erro absoluto ponderado (ITAE) (veja a Equação (2.3)). Assim, o ITAE foi avaliado por 2 s. A Figura 3.29 ilustra a evolução temporal do ITAE para os três cenários de simulação e o resultado final é apresentado na Tabela 3.5. O melhor desempenho corresponde ao cenário com menor valor de ITAE.

Da Tabela 3.5 observa-se que com a introdução de um sistema de comunicação entre planta e controlador (do Cenário 1 para o Cenário 2) o ITAE aumentou em 148,98 %, já com a introdução das perdas de pacotes (do Cenário 2 para o Cenário 3) o ITAE aumentou em 62 %, reforçando-se assim que atrasos foram mais prejudiciais para o desempenho do sistema de controle do que as perdas de pacotes.

Essa conclusão pode ser muito relevante no momento de projetar um algoritmo AQM a ser implementado em roteadores de uma NCS que utiliza a Internet, já que são essas técnicas que podem reduzir atrasos de tempo de espera na fila mediante descarte ou marcação de pacotes.

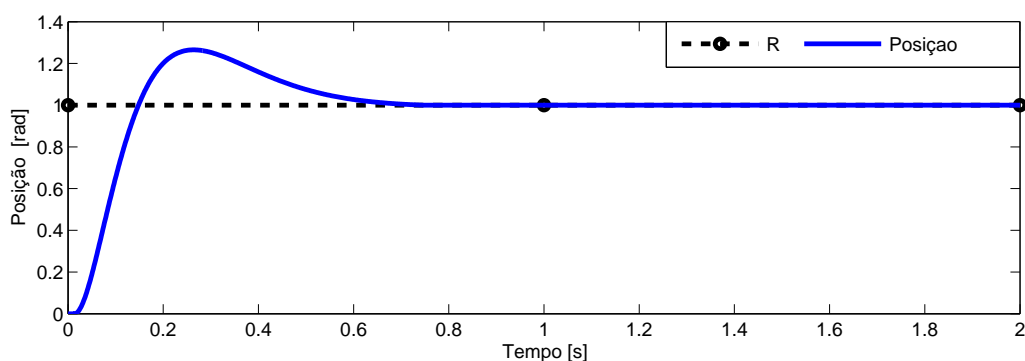


Figura 3.26: Posição do motor e referência para o Cenário 1.

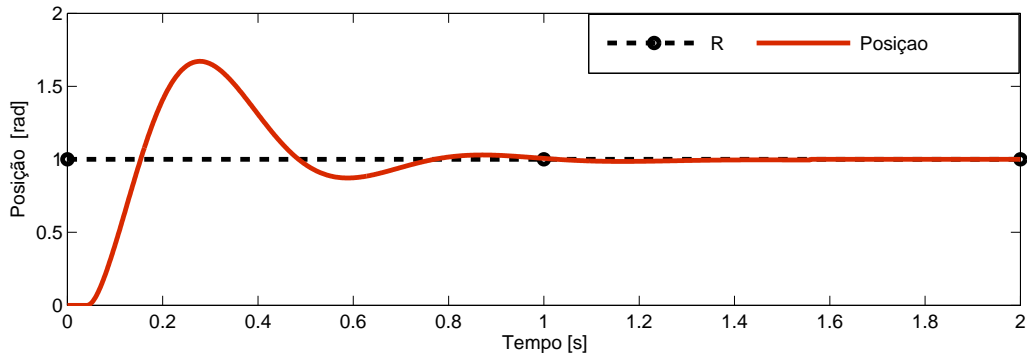


Figura 3.27: Posição do motor e referência para o Cenário 2.

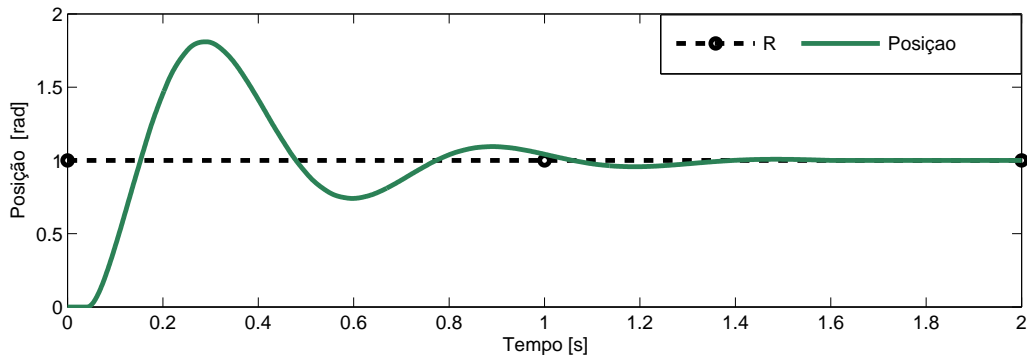


Figura 3.28: Posição do motor e referência para o Cenário 3.

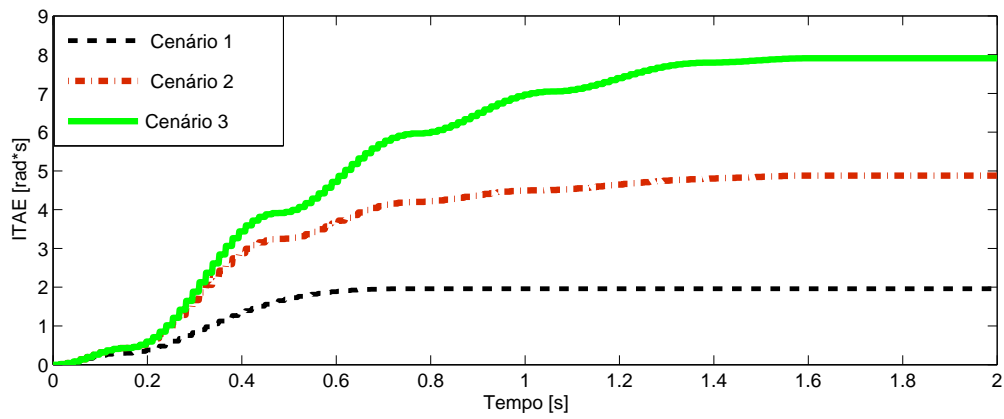


Figura 3.29: ITAE para os Cenários 1, 2 e 3.

Tabela 3.5: Valor final do ITAE para os primeiros três cenários de simulação.

Cenário	1	2	3
ITAE [rad.s]	1,960	4,880	7,906

3.7.2 Influência de outros fluxos que compartilham a mesma topologia de rede de comunicação com uma NCS

Para estudar a influência de outros fluxos que compartilham a mesma topologia de rede de comunicação com uma NCS foram simulados 3 cenários (A, B e C) diferentes descritos a seguir:

- Cenário A: Planta e controlador trocam dados através da Internet por meio da topologia de rede de comunicação (NCS) ilustrada na Figura 3.25 (o mesmo que o segundo cenário da seção anterior);
- Cenário B: À topologia da Figura 3.25 foi adicionado um fluxo TCP-Reno formando assim a NCS ilustrada na Figura 3.30;
- Cenário C: Partindo da topologia de rede de comunicação ilustrada na Figura 3.25, porém, colocando um atraso de 10 ms para o canal situado entre o controlador e o roteador 1, de tal maneira que o atraso de propagação total (de ida e volta) existente na rede é de 50 ms, o qual de acordo com [107] é ainda menor que o máximo atraso requerido para uma operação estável. Após 4 segundos de simulação foi adicionado um fluxo TCP-Reno em paralelo que causará um aumento no tempo de espera na fila e conseqüentemente poderá aumentar o atraso na NCS até valores acima do máximo permitido para uma operação estável, obtendo assim uma topologia de rede de comunicação semelhante à ilustrada na Figura 3.30.

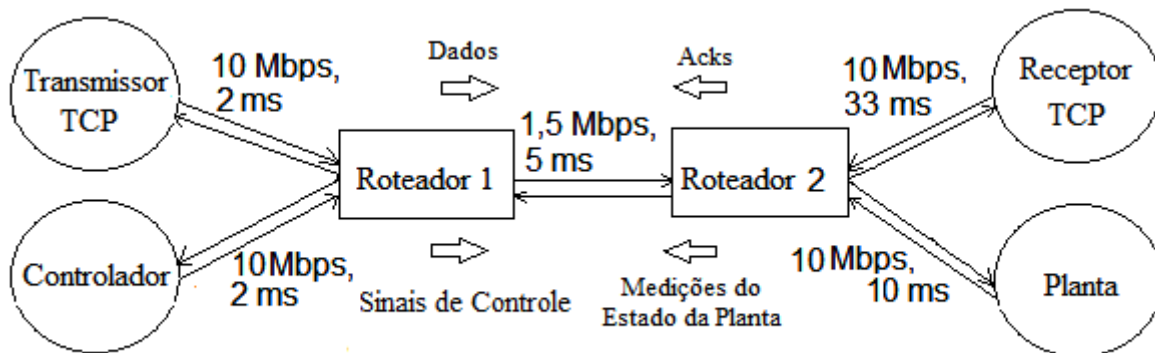


Figura 3.30: NCS compartilhando a mesma topologia de rede com um fluxo TCP genérico.

A técnica AQM utilizada nos Cenário A, B e C é Drop Tail e, o protocolo de transporte de dados utilizado para comunicar controlador e planta é o protocolo UDP. Nos cenários A e B, o motor segue uma referência R constante de 1 rad, enquanto que no Cenário C o motor segue uma referência de onda quadrada que varia entre 1 e 2 rad com período de 2 s.

Análise dos resultados

As Figuras 3.31 e 3.32 ilustram a posição do motor em função do tempo para os cenários A e B, respectivamente. Graficamente, é possível observar que o erro é maior para o cenário B do que

para o cenário A, ou seja, a presença de outros fluxos na mesma topologia de rede de comunicação aumenta os atrasos e conseqüentemente afetam o desempenho do sistema de controle.

Para quantificar esse fenômeno numericamente foi utilizado o ITAE, cuja evolução temporal é dada na Figura 3.33, e cujos valores finais para 2 segundos de simulação são mostrados na Tabela 3.6. Observa-se que com a introdução de um único fluxo TCP-Reno adicional na mesma topologia de rede de comunicação utilizada pela NCS, o ITAE piorou em 119,2 %.

Esse resultado pode ser relevante no momento de projetar protocolos de transporte de dados a serem utilizados na Internet, já que atrasos introduzidos pela presença de outros fluxos depende de quão agressivos eles são. Esse fenômeno ficou ainda mais claro no Cenário C.

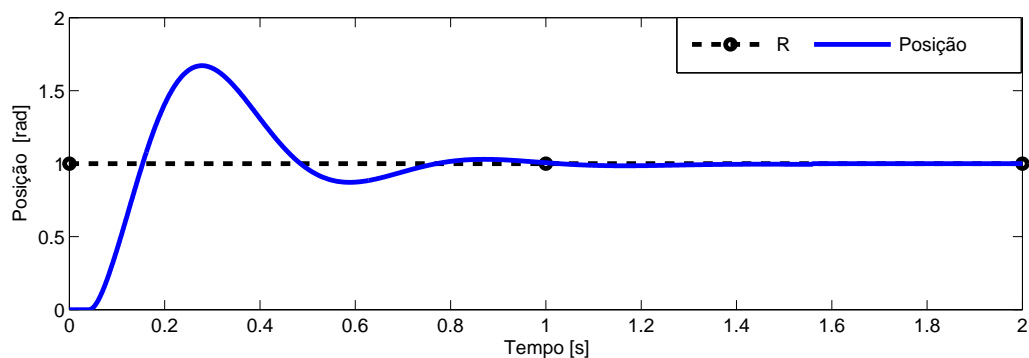


Figura 3.31: Posição do motor e referência para o Cenário A.

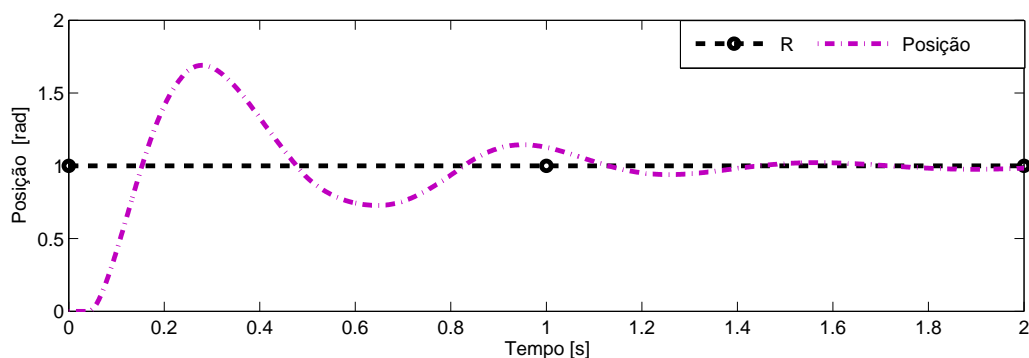


Figura 3.32: Posição do motor e referência para o Cenário B.

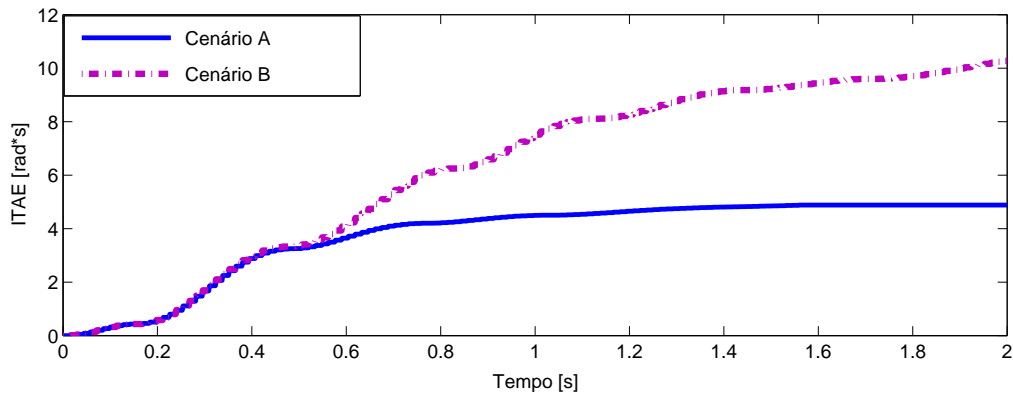


Figura 3.33: ITAE para os Cenários A e B.

Tabela 3.6: Valor final do ITAE para os primeiros os Cenários A e B.

Cenário	A	B
ITAE [rad.s]	4,88	10,27

A Figura 3.34 mostra a posição do motor para o cenário C. Note que nos primeiros 4 segundos, enquanto que somente o fluxo UDP da NCS está ligado, esse pequeno fluxo não produz enfileiramentos de pacotes, e assim sendo, não existe tempo de espera na fila e o atraso total introduzido pela rede é apenas a soma do tempo de propagação nos canais e o tempo de atendimento dos pacotes, ou seja, menor que 78 ms (máximo atraso para uma operação estável do processo que está sendo controlado segundo [107]), assim, o sistema permanece estável.

Porém, a partir dos 4 s, o fluxo TCP é ligado e, conseqüentemente, a fila no caminho de ida do roteador 1 e a fila no caminho de retorno do roteador 2 começam a aumentar e o atraso ponta a ponta também aumenta, até que a partir dos 6 segundos a estabilidade já não pode ser mais garantida. Observe que como o canal é de 1,5 Mbps, o tempo necessário para atender um pacote de 1 kbyte é de aproximadamente 5,46 ms. Portanto, o tempo de espera na fila será de 5,46 ms vezes a quantidade de pacotes do fluxo TCP enfileirados.

Esse resultado indica que o desempenho de um sistema de controle através da Internet depende fortemente da técnica AQM que está sendo utilizada nos roteadores, já que são essas técnicas que controlam os tamanhos das filas nos roteadores, e evitam problemas de *bufferbloat* atuando assim indiretamente no atraso ponta a ponta. Assim, no presente trabalho foi proposto pesquisar o desempenho global do sistema para diferentes técnicas AQM, conforme será visto a seguir.

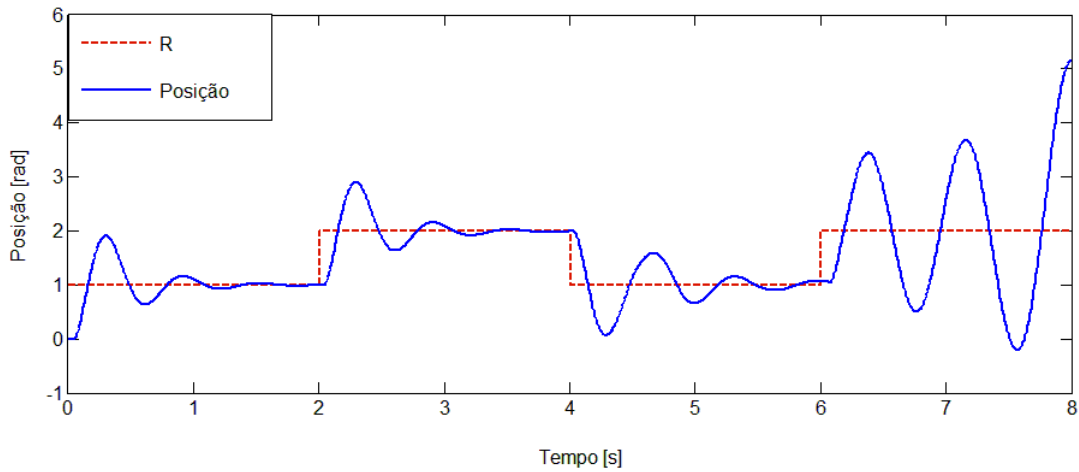


Figura 3.34: Posição do motor para o Cenário C.

3.7.3 Influência de técnicas AQM nos fluxos que compartilham a mesma topologia de rede de comunicação.

Para estudar a influência de diferentes técnicas AQM em parâmetros de desempenho dos fluxos de dados que compartilham a mesma topologia de rede de comunicação, foi utilizada a mesma NCS ilustrada na Figura 3.30, na qual planta e controlador trocam dados mediante um fluxo UDP na mesma topologia de rede de comunicação utilizada por um fluxo TCP-Reno genérico. Neste mesmo contexto, foi avaliado tanto o desempenho do sistema de controle (fluxo UDP) em termos de ITAE, quanto o desempenho do fluxo TCP-Reno em termos de vazão para as técnicas AQM Drop Tail, RED, CoDel e PIE implementadas nos roteadores.

Análise dos resultados

As Figuras 3.35, 3.36, 3.37 e 3.38 ilustram a posição do motor em função do tempo para as técnicas AQM DT, RED, CoDel e PIE, respectivamente. Observa-se que, aproximadamente nos primeiros três segundos de simulação, Drop Tail e RED resultam praticamente o mesmo comportamento. Porém, após três segundos acontece uma compensação entre vazão do fluxo TCP (ilustrado na Figura 3.40) e o desempenho do sistema de controle em termos de ITAE (ilustrado na Figura 3.39).

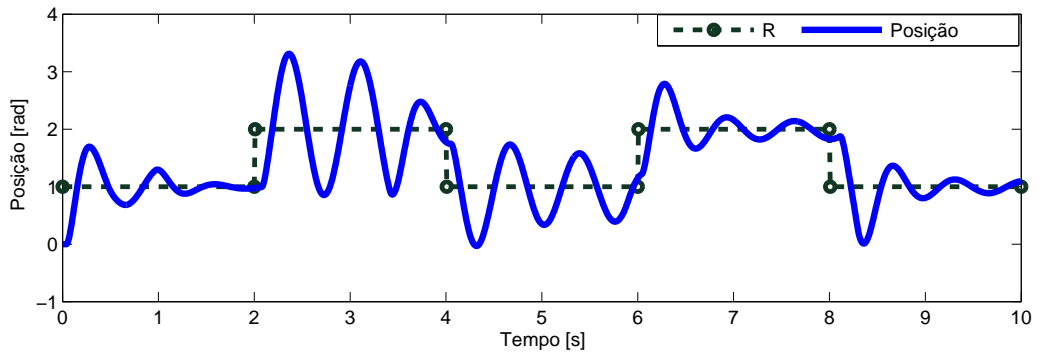


Figura 3.35: Posição do motor para a técnica AQM Drop Tail.

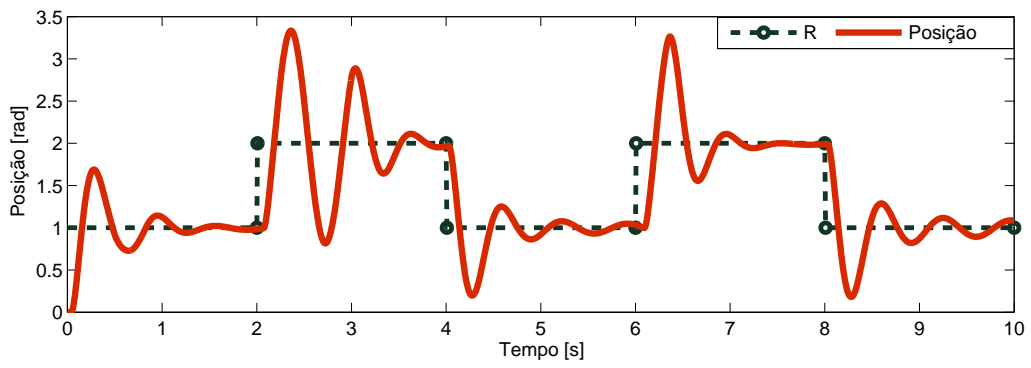


Figura 3.36: Posição do motor para a técnica AQM RED

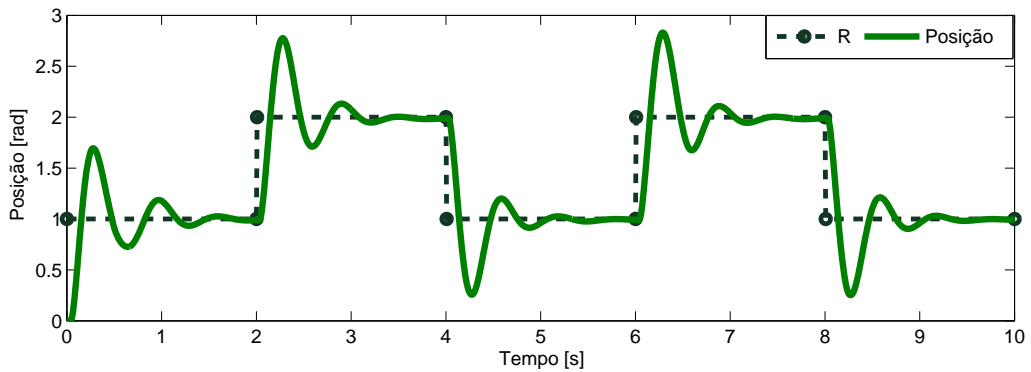


Figura 3.37: Posição do motor para a técnica AQM CoDel

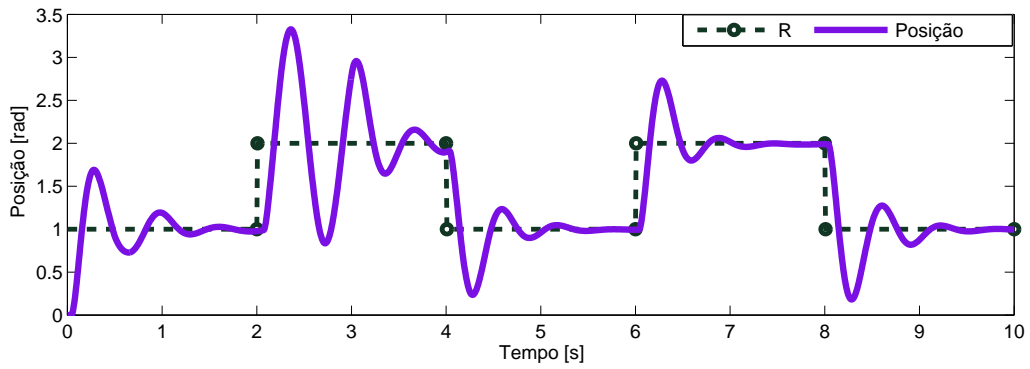


Figura 3.38: Posição do motor para a técnica AQM PIE.

Essa compensação acontece devido a que o algoritmo RED entra em regime e começa a marcar pacotes na fila com o objetivo de controlar o congestionamento, logo, como pode ser observado nas Figuras 3.41 e 3.42, RED mantém a fila em menor tamanho, o que implica em menores atrasos ponta a ponta e, conseqüentemente, melhor desempenho do sistema de controle comparado com Drop Tail. Porém, esse benefício tem um preço que se vê refletido na vazão do fluxo TCP que corre em paralelo, já que quando RED começa a marcar pacotes o transmissor TCP reduz sua taxa de transmissão e conseqüentemente piora a vazão como pode ser observado na Figura 3.40.

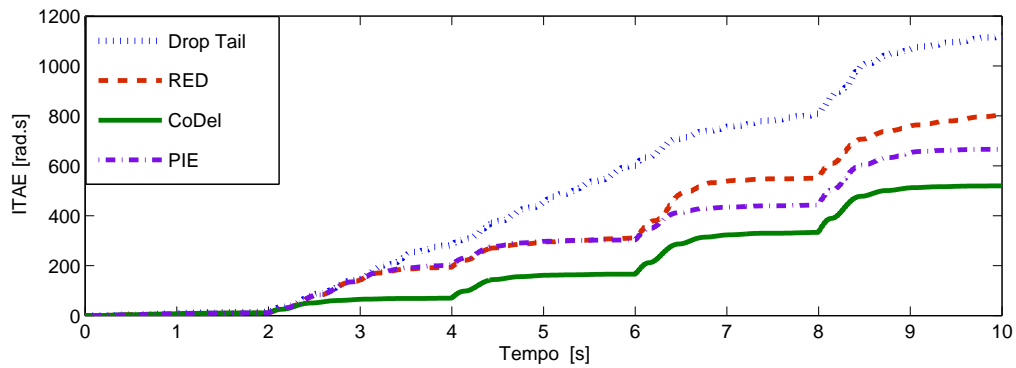


Figura 3.39: ITAE para diferentes técnicas AQM.

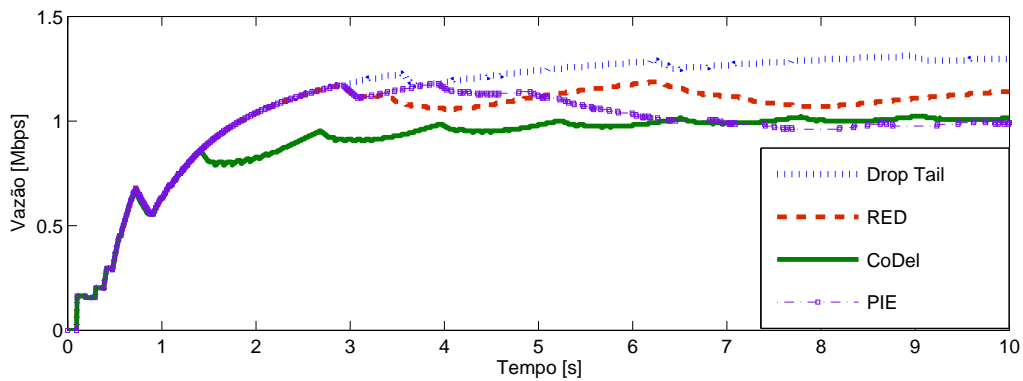


Figura 3.40: Vazão total dos Fluxos TCP-Reno para diferentes técnicas AQM.

Por outro lado, CoDel começa sua ação antes que o RED. Assim, CoDel começa marcar pacotes logo após o primeiro segundo de simulação (veja a Figura 3.43). Comparado com Drop Tail e RED, CoDel mantém o tamanho da fila menor com vários períodos de *bufferempty* (conforme pode ser observado nas Figuras 3.42 e 3.43 respectivamente) o que implica em menor atraso ponta a ponta e, conseqüentemente, melhor desempenho do sistema de controle em termos de ITAE, porém resulta pior vazão para o fluxo TCP, como pode ser observado nas Figuras 3.39 e 3.40, respectivamente.

Enquanto a PIE, essa técnica tardou um pouco mais que CoDel e pouco menos que RED para entrar em regime, conseguindo assim praticamente o mesmo desempenho que CoDel quanto à vazão do fluxo TCP-Reno (Figura 3.40), mas um desempenho ligeiramente inferior a CoDel em termos de ITAE para o sistema de controle em rede (veja a Figura 3.39).

Na Tabela 3.7 são indicados os valores finais (obtidos após 10 s de simulação) para ITAE da NCS e vazão do fluxo TCP para cada técnica AQM.

Interessante observar que as técnicas AQM que evitam o fenômeno de *bufferbloat* (CoDel e PIE) melhoram o desempenho do sistema de controle em termos de ITAE, devido a que reduzem o tempo de espera na fila e conseqüentemente o atraso ponta a ponta, o que neste caso foi bom para o sistema de controle. Mas, em contrapartida, afetam a vazão do fluxo TCP devido a que apresentam vários períodos de *bufferempty*. Por outro lado, a técnica que evita períodos de *bufferempty* (Drop Tail) apresenta melhor vazão para o fluxo TCP, mas piora o desempenho da NCS em termos de ITAE devido a que apresentam vários períodos de *bufferbloat*.

Em síntese, observa-se que as técnicas AQM que fornecem boa vazão para o fluxo TCP produzem maiores atrasos e pioram o ITAE para a NCS, e as que fornecem um bom ITAE para a NCS pioram a vazão para o fluxo TCP.

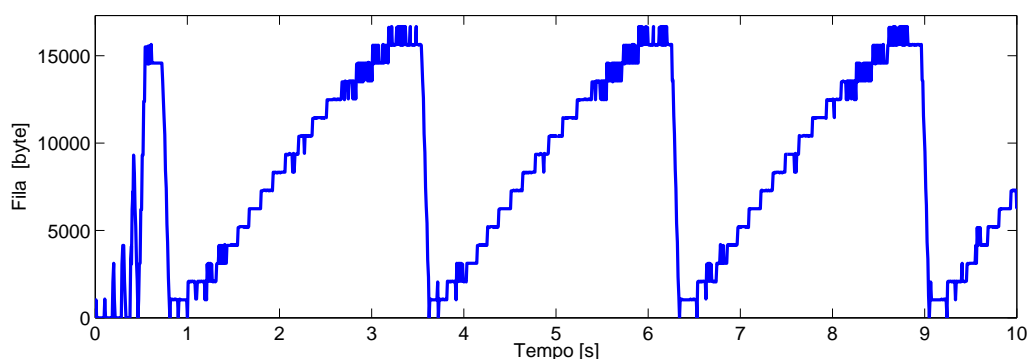


Figura 3.41: Fila no caminho de ida do roteador 1 para a técnica AQM Drop Tail.

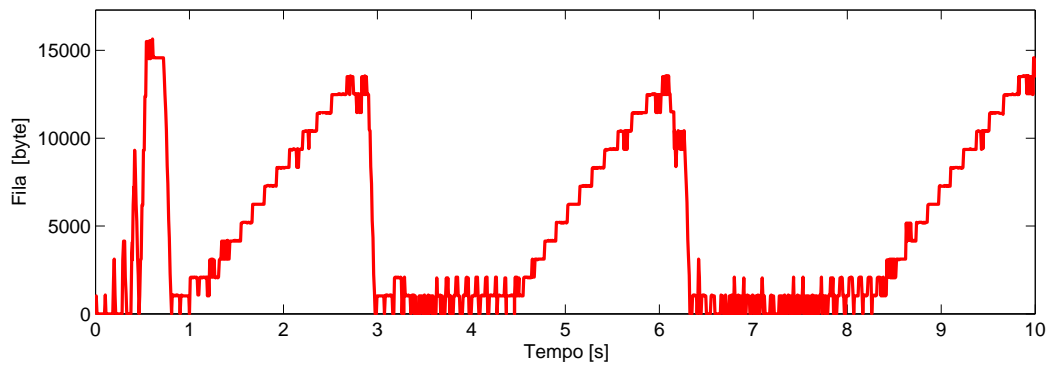


Figura 3.42: Fila no caminho de ida do roteador 1 para a técnica AQM RED.

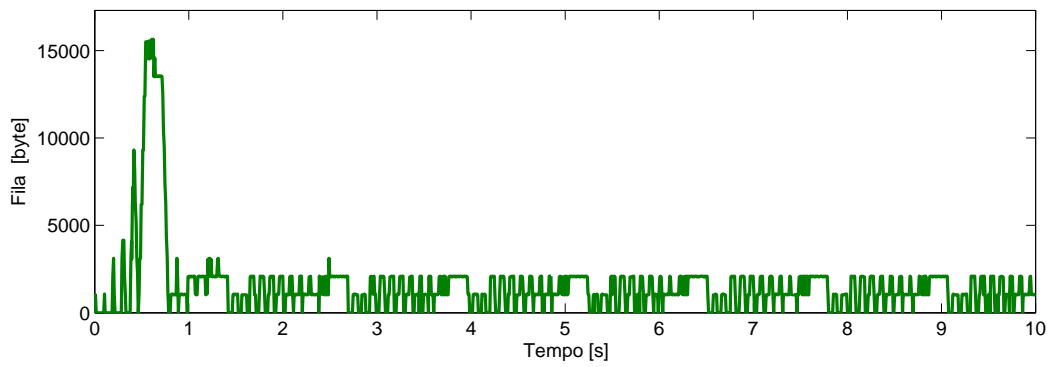


Figura 3.43: Fila no caminho de ida do roteador 1 para a técnica AQM CoDel.

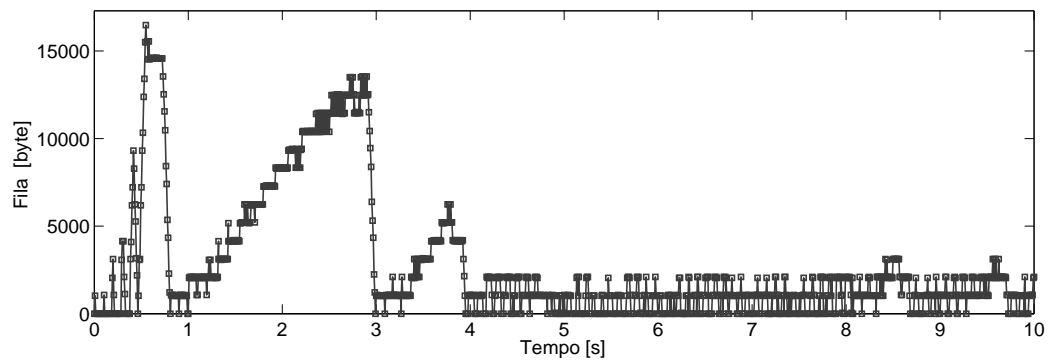


Figura 3.44: Fila no caminho de ida do roteador 1 para a técnica AQM PIE.

Tabela 3.7: Valor final do ITAE e da Vazão para diferentes Técnicas AQM.

	Drop Tail	RED	PIE	CoDel
ITAE [Rad.S] Para a NCS	1120,4	806	666,2	519,9
Vazão [Mbps] Para o Fluxo TCP	1,29	1,14	1	1,01

3.7.4 Análise do desempenho de NCSs através da Internet mediante a ferramenta *Statistical Model Checking* (SMC) do UPPAAL

Conforme visto nos capítulos anteriores, em uma NCS ocorrem tanto eventos determinísticos quanto eventos aleatórios. Assim, por exemplo, técnicas AQM geralmente marcam ou descartam pacotes segundo algoritmos probabilísticos. Perdas de pacotes, ruídos nos canais e perdas de conexões também são eventos probabilísticos, assim sendo, em determinadas ocasiões é importante fazer um análise probabilística dos resultados, ou seja, verificar a possibilidade e/ou a probabilidade com que determinados eventos possam acontecer.

A ferramenta UPPAAL facilita fazer esse tipo de análises mediante o módulo SMC. Para ilustrar a utilidade dessa ferramenta, foi utilizada a topologia de rede de comunicação ilustrada na Figura 3.45, na qual existe um fluxo UDP de uma NCS, um fluxo TCP-Reno genérico e um tráfego de fundo, dado por um fluxo Poisson aleatório que transmite pacotes de 1kbyte com uma taxa $\lambda = 15$ pacotes/s no mesmo sentido que o fluxo TCP, sendo implementados nos roteadores a técnica AQM Drop Tail. Nesta topologia ainda foi adicionada uma taxa aleatória de perdas de pacotes de 10 % nos canais existentes entre o controlador e o roteador 1 e entre a planta e o roteador 2.

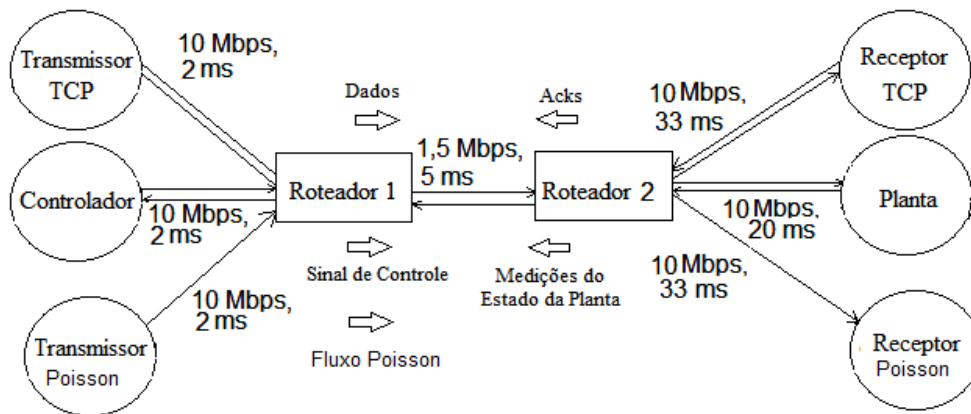


Figura 3.45: NCS compartilhando a mesma topologia de rede com um fluxo TCP genérico e um fluxo Poisson aleatório.

Primeiramente, foi obtido o desempenho do sistema de controle para uma rodada de simulação de 10 s. As Figuras 3.46 e 3.47 ilustram a posição do motor e a evolução temporal do ITAE, respectivamente, nessa rodada de simulação o ITAE atingiu um valor final de 1179 rad.s.

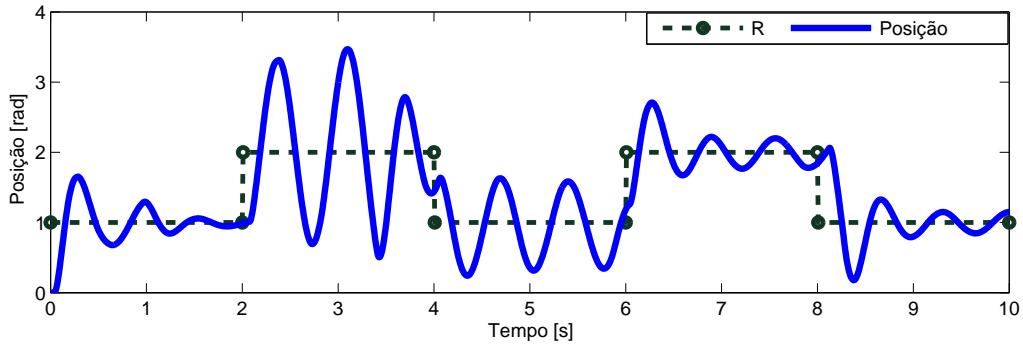


Figura 3.46: Posição do motor segundo resultado de uma única rodada de simulação.

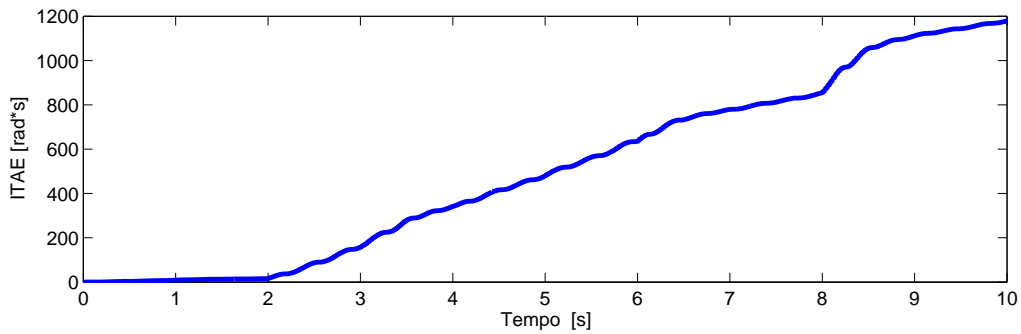


Figura 3.47: ITAE segundo resultado de uma única rodada de simulação.

A simulação é útil para obter resultados de uma forma mais célere, mas nela são analisados somente algumas possibilidades da evolução da dinâmica do sistema. Para obter resultados mais completos é necessário fazer uma verificação formal.

Assim, foi realizado uma análise probabilística mediante a ferramenta SMC do UPPAAL para checar a probabilidade do ITAE atingir valores maiores ao valor obtido na simulação, ou seja, foi checada a probabilidade do ITAE ser maior que 1179 rad.s em até 10 s usando a sintaxe $\text{Pr } [\leq 10000000] \{ \langle \rangle \text{ITAE} > 1179 \}$ onde ITAE é a variável que representa o ITAE. No UPPAAL está sendo utilizado microsegundos como unidade de tempo, assim 10000000 correspondem a 10 s.

Para efetuar essa tarefa o UPPAAL utilizou como amostra o resultado de 357 simulações feitas com sementes diferentes. Logo, estimou com uma confiança de 95 % que o ITAE pode ultrapassar o valor de 1179 rad.s em até 10 s com uma probabilidade dada no intervalo de 0,620899 a 0,720753 (veja a Figura 3.48).

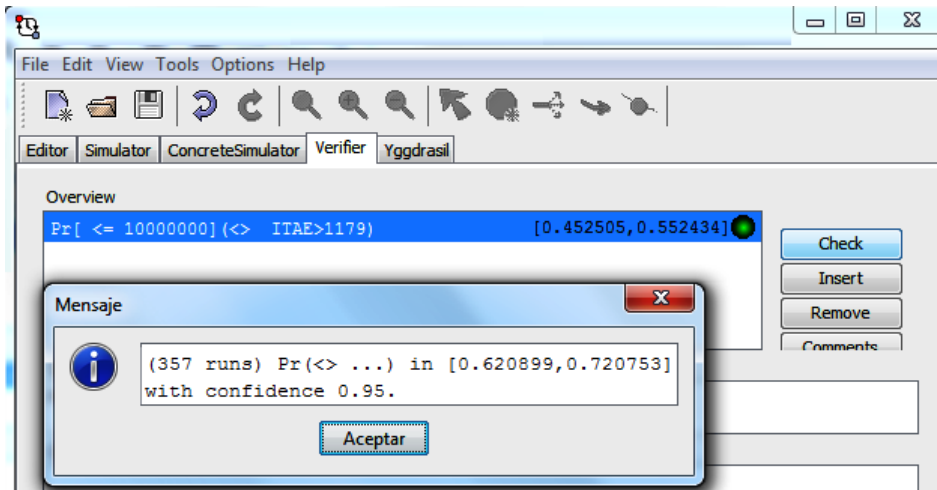


Figura 3.48: Probabilidade do ITAE ser maior que 1179 rad.s em um tempo menor ou igual a 10 s.

UPPAAL ainda fornece gráficos da frequência acumulada, no qual é possível observar a partir de que instante de tempo existe uma probabilidade diferente de zero do evento acontecer. A Figura 3.49 ilustra a probabilidade acumulada do ITAE ultrapassar 1179 rad.s. Conforme pode ser observado essa probabilidade só é diferente de zero após os 8,4 s.

Por outro lado, UPPAAL ainda oferece histogramas e polígonos de frequências, nos quais é possível observar quais são os instantes de tempo mais prováveis no qual o evento estudado pode ocorrer. Na Figura 3.50 observa-se o histograma e o polígono de frequências da probabilidade do ITAE ultrapassar 1179 rad.s em até 10 s. De acordo com esse gráfico os intervalos de tempo mais prováveis ocorrem nas classes com ponto médio em 9,6 e 9,9 s.

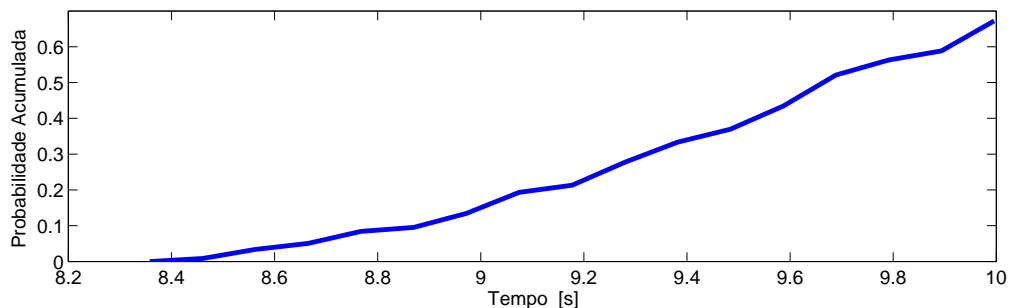


Figura 3.49: Probabilidade acumulada do ITAE ser maior que 1179 rad.s em menos de 10 s.

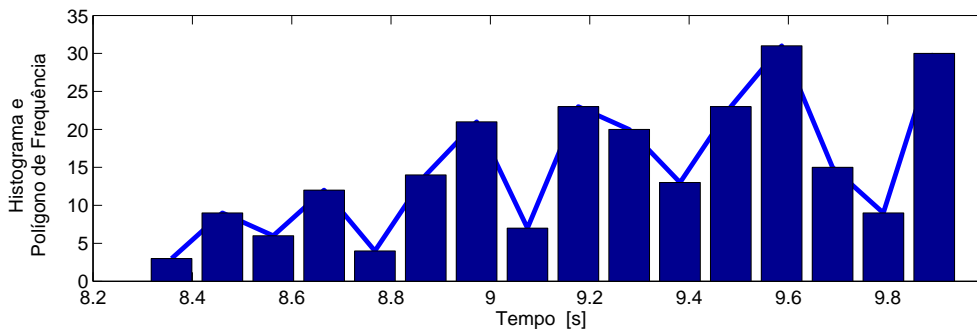


Figura 3.50: Histograma e Polígonos de frequências da probabilidade do ITAE ser maior que 1179 rad.s em menos de 10 s.

Também foi checada a probabilidade da posição do motor ultrapassar 2 rad antes do primeiro segundo de simulação, usando a sintaxe $\text{Pr} [\leq 1000000] \{ \langle \rangle \text{Position} > 2 \}$, onde Position é a variável que representa a posição do motor. Para efetuar essa tarefa o UPPAAL pegou como amostra o resultado de 400 simulações feitas com sementes diferentes. Logo, estimou com uma confiança de 95 % que a probabilidade da posição do motor ultrapassar 2 rad antes do primeiro segundo de simulação está no intervalo de 0 a 0,0973938 (veja a Figura 3.51).

Neste caso, o UPPAAL não forneceu gráficos de frequência acumulada e nem polígonos de frequência, pois em nenhuma das 400 simulações a posição do motor ultrapassou 2 rad antes do primeiro segundo de simulação.

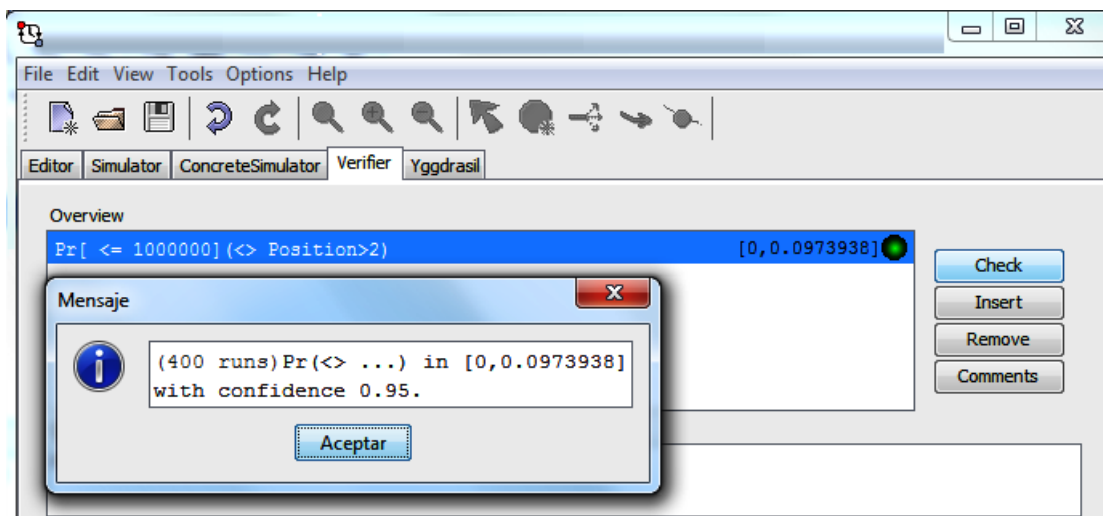


Figura 3.51: Probabilidade da posição do motor ultrapassar 2 rad antes do primeiro segundo de simulação.

3.8 Conclusões

Neste capítulo foi mostrado que topologias de redes de comunicação com fluxos TCP e UDP e com técnicas AQM implementadas em roteadores, assim como sistemas de controle através da

Internet, podem ser modelados mediante conjuntos de autômatos temporizados através de uma linguagem baseada em lógica TCTL utilizando a ferramenta de software UPPAAL.

A simulação do modelo de topologias de redes de comunicação obtidas em UPPAAL são compatíveis com resultados encontrados na bibliografia obtidos com simulador NS-2. Além disso, UPPAAL facilita o estudo de sistemas de controle através da Internet e permite realizar uma modelagem de rede de comunicação nas quais o desempenho de diferentes fluxos podem ser analisados de forma conjunta.

Assim, mediante a metodologia de modelagem de rede de comunicação implementada, foi possível identificar e estudar as influências mútuas que ocorrem entre o fluxo UDP de uma NCS e fluxos TCP que correm em paralelo compartilhando a mesma topologia de rede de comunicação. E ainda foi possível analisar o desempenho desses fluxos para diferentes técnicas AQM implementadas nos roteadores.

Neste sentido, foi identificado que técnicas AQM geralmente apresentam um fenômeno de *bufferempty* que acontece quando o algoritmo AQM freia de forma excessiva os transmissores TCPs ou quando o transmissor TCP reage de forma exagerada diante de uma notificação explícita ou implícita de congestionamento, de maneira que a fila no roteador permanece completamente desocupada por alguns períodos de tempo. Fazendo assim com que a correspondente porta de saída do roteador permaneça desocupada, apesar de que o transmissor tenha pacotes para transmitir. Mostrou-se ainda que esse fenômeno é indesejado e pode comprometer a vazão. Observou-se também que técnicas AQM e protocolos de transporte de dados que apresentam menos e/ou menores períodos de *bufferempty* em geral tendem a apresentar o fenômeno de *bufferbloat*.

Por outro lado, na abordagem de simulações conjuntas de diferentes fluxos, foi possível observar que os atrasos na rede assim como perdas de pacotes por descarte nas filas dos roteadores estão ambos intimamente vinculados com a técnica AQM implementada nos roteadores. Observou-se ainda que técnicas AQM que não apresentam o fenômeno de *bufferbloat* apresentam melhor desempenho para aplicações NCS através da Internet em termos de ITAE já que essas técnicas reduzem o tempo de espera na fila e conseqüentemente o atraso ponta a ponta, mas, em contrapartida, essas técnicas degradam o desempenho dos fluxos TCP em termos de vazão. Por outro lado, técnicas AQM que não apresentam o fenômeno de *bufferempty* melhoram a vazão para os fluxos TCP, mas aumentam os atrasos na rede degradando assim o desempenho de NCS através da Internet em termos de ITAE.

Essas observações podem ser úteis no momento de escolher ou elaborar um algoritmo AQM a ser implementado em uma NCS na qual a topologia de rede de comunicação que comunica o controlador com a planta é compartilhada com outros fluxos.

Nos capítulos seguintes serão propostas técnicas de gestão de filas baseadas em explícita e implícitas notificações de congestionamento que consigam evitar o fenômeno de *bufferbloat* sem causar demasiados períodos de *bufferempty*, para que assim consigam alcançar bom desempenho tanto para fluxos TCP quanto para fluxos UDP de uma NCS. O desempenho será comparado mediante simulações e verificações em UPPAAL com as técnicas AQM vistas neste capítulo.

Capítulo 4

Novas Técnicas AQM ENCN e ANCE

4.1 Introdução

No capítulo anterior foi identificado que algumas técnicas AQM encontradas na literatura podem apresentar um fenômeno indesejado chamado neste trabalho de fenômeno de *bufferempty*, que ocorre quando a técnica AQM freia em excesso os transmissores, ou então, quando os transmissores reagem demasiado a uma notificação explícita ou implícita de congestionamento. Assim, por alguns instantes de tempo, nenhum pacote chega no roteador conectado ao canal gargalo, e o roteador fica ocioso e em consequência a vazão resulta prejudicada.

Foi visto também que esse fenômeno de *bufferempty* é mais comum em técnicas AQM projetadas para evitar o fenômeno de *bufferbloat*. Por outro lado, no capítulo anterior, também foi identificado que quando um processo é controlado através da Internet, no qual geralmente a NCS compartilha uma mesma topologia de rede de comunicação com diversos outros fluxos genéricos, pode ocorrer uma compensação entre desempenho do sistema de controle e vazão para os outros fluxos genéricos, e essa compensação depende fortemente da técnica AQM implementada nos roteadores.

Assim, visando por um lado reduzir o fenômeno de *bufferempty* e por outro lado conseguir uma melhor compensação entre sistemas de controle através da Internet e outros fluxos genéricos que compartilham a mesma topologia de rede de comunicação, neste capítulo é apresentada uma nova técnica AQM chamada Notificação Explícita de Não Congestionamento (ENCN, do inglês *Explicit Non-Congestion Notification*) a qual, basicamente, ao invés de notificar congestionamento como faz ECN, utiliza esses mesmos bits para fazer uma notificação explícita de não congestionamento.

Além disso, igual aos protocolos TCP citados e descritos na Seção 2.7 do Capítulo 2, ENCN “casa” ações na fila do roteador com ações na janela de transmissão TCP, ou seja, as ações em *cwnd* são realizadas em função da recepção de ACKs marcados, visando assim melhor aproveitar essa informação de não congestionamento proveniente dos roteadores, com a diferença de que os protocolos TCP descritos na Seção 2.7 são baseados em notificações explícita de congestionamento, enquanto que ENCN no intuito de evitar o fenômeno de *bufferempty* é baseado em notificações explícitas de não congestionamento.

Assim, ENCN procura melhorar o trabalho em conjunto de algoritmos de gestão de fila implementado nos roteadores com algoritmos implementados no transmissor, com o objetivo de evitar tanto o fenômeno de *bufferbloat* quanto o fenômeno de *bufferempty*.

Porém, conforme estudado na Subseção 2.6.4 do Capítulo 2, para trabalhar com notificações explícitas é necessário contar com diversos recursos tecnológicos, ou seja, tanto transmissor e receptor quanto todos os roteadores existentes no caminho de comunicação entre transmissor e receptor devem ser capazes de trabalhar com ECN. Uma vez que ENCN utiliza os mesmos recursos que ECN fica também dependente desses recursos tecnológicos que nem sempre estarão disponíveis. Logo, sua aplicação prática fica limitada.

Para solucionar esse problema, neste trabalho também foi proposta uma metodologia de gestão de fila ponta a ponta baseada em notificação implícita de não congestionamento, a qual possui unicamente algoritmos implementados no transmissor, sem necessidade de implementar algoritmos AQM nos roteadores, sendo portanto, independente dos recursos tecnológicos usados por ECN e ENCN. Essa técnica foi denominada ANCE (do inglês *Acknowledge-based Non-Congestion Estimation*).

ANCE estima períodos de não congestionamento e, assim, atua na janela de transmissão *cwnd* no intuito de evitar tanto o fenômeno de *bufferbloat* quanto o fenômeno de *bufferempty*. Para essa finalidade, ANCE utiliza uma técnica de estimação do tamanho da fila no gargalo. Essa técnica de estimação do tamanho da fila está baseada em uma outra técnica de estimação da banda do canal congestionado. Esses estimadores são implementados nos transmissores TCP.

Assim, ANCE funciona de forma análoga a ENCN com a diferença de que os períodos de não congestionamento são observados de forma implícita em função do tamanho da fila estimado pelo transmissor ao invés de usar notificação implícita de não congestionamento como faz ENCN.

Dessa forma, ANCE consegue fazer uma gestão de filas ponta a ponta sem necessidade de utilizar os recursos tecnológicos requeridos por protocolos de transporte de dados baseados em ECN ou ENCN.

A técnica de estimação de fila utilizada por ANCE está baseada na técnica de estimação de fila implementada no TCP-NCE [112]. Com a diferença de que, no TCP-NCE, a estimação do tamanho da fila é realizado supondo conhecido a banda no canal gargalo. No presente trabalho foi adicionado uma técnica para estimar essa banda em função do mínimo tempo entre chegadas dos ACKs, para assim poder implementar essa técnica de estimação de fila em cenários mais reais, nos quais tanto banda do canal gargalo quanto a banda de qualquer outro canal no caminho entre transmissor e receptor são normalmente desconhecidas.

4.2 ENCN (Explicit Non-Congestion Notification)

Nessa seção será apresentado um novo protocolo desenvolvido neste trabalho chamado ENCN que é baseado em notificações explícitas de não congestionamento em um caminho entre transmissor e receptor, e combina transporte de dados na Internet com técnicas AQM.

Assim, diferentemente de outras técnicas AQM que trabalham em conjunto com ECN fazendo notificações explícitas de congestionamento em uma fila, ENCN notifica não congestionamento em um caminho existente entre transmissor e receptor. Assumindo que o caminho do fluxo de ACKs é também um caminho possível para o fluxo de dados no sentido oposto.

Para essa finalidade, ENCN atua tanto nos roteadores como nos transmissores e conta com dois algoritmos, um denominado “Marcação ENCN” implementado nos roteadores e descrito no Algoritmo 4.1, e outro denominado “Reação ENCN” implementado nos transmissores e descrito no Algoritmo 4.2.

4.2.1 Marcação ENCN

Para marcar pacotes, o ENCN utiliza os mesmos recursos que ECN (descrito na Subseção 2.6.4). Assim, primeiramente transmissor e receptor concordam em trabalhar com ENCN, usando o bit ECT do cabeçalho IP.

Logo, cada vez que o receptor emitir um ACK, colocará o bit CE do cabeçalho IP no valor 1, ou seja, o ACK sairá do receptor com uma marca que em princípio indica não congestionamento. Ao longo de seu caminho, esse ACK passará por vários roteadores até chegar no transmissor. Quando esse ACK chegar no módulo de entrada (sentido receptor/transmissor) da porta de um roteador cujo tamanho da fila no módulo de saída (sentido transmissor/receptor) Q_{inst} for maior que um limiar Q_{min} (conforme ilustra a Figura 4.1), esse bit CE do ACK será configurado em zero, ou seja, será removida a marca de não congestionamento. Caso contrário, ou seja, quando a fila no sentido transmissor/ receptor for menor que Q_{min} , a marca CE do ACK será deixada em 1 indicando assim ao transmissor que esse caminho no sentido transmissor/receptor não está congestionado.

Algoritmo 4.1 Marcação ENCN

Marcação ENCN	Observações
Cada vez que um ACK sai do Receptor {CE=1;} Cada vez que um ACK chega no roteador { if ($Q_{inst} > Q_{min}$) {CE=0;} } else {Espera até um ACK chegar}	Cada vez que um ACK sai do receptor o bit CE é configurado em 1. Se o tamanho da fila no módulo de saída (sentido transmissor/receptor) Q_{inst} for maior que um limiar Q_{min} . O bit CE do ACK é configurado em zero.

Quando o transmissor receber um ACK com CE em 1, entenderá que existe pelo menos um caminho entre ele e o correspondente receptor que não está congestionado, e então, atuará de imediato em sua janela de transmissão $cwnd$ de acordo com o algoritmo “Reação ENCN” a fim de evitar o fenômeno de *bufferempty*. Já que essa informação de não congestionamento é um indicador de que todas as possíveis filas existentes neste caminho estão abaixo do limiar Q_{min} , ou seja, existe um risco potencial de ocorrer o fenômeno de *bufferempty*.

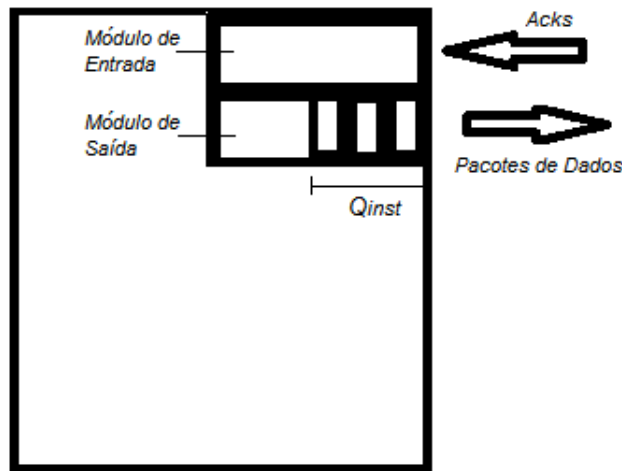


Figura 4.1: Funcionamento de ENCN em um roteador.

Observe que, quando um ACK marcado chega a um roteador, ele traz consigo a informação de que todas as filas existentes no sentido oposto deste caminho estão não congestionadas. Essa informação pode ser útil também para auxiliar algoritmos de roteamento na escolha de caminhos menos congestionados. Porém, esse tema está fora do escopo desse trabalho.

A banda do canal ligado ao módulo de saída é a que limita a velocidade com que os pacotes de dados irão sair da fila. Assim, o tempo de espera na fila será igual ao produto do tamanho da fila pela banda do canal. Logo, o valor de Q_{min} pode ser calculado em função do tempo de espera na fila considerado admissível e da banda do canal ligado à porta cuja fila do módulo de saída está sendo comparada com Q_{min} de acordo com

$$Q_{min} = \alpha.B, \quad (4.1)$$

onde α é uma constante dependente do tempo de espera na fila considerado admissível; assim, se for designado a α o valor de 1/100 se terá um tempo de espera na fila de 10 ms, quando Q_{inst} for igual a Q_{min} . E B é a banda do canal. Se B for colocada em bits/s o valor de Q_{min} ficará expresso em bits.

Observe, que de forma semelhante aos protocolos TCP baseados em ECN, tais como TCP-Jersey, DCTCP e E-DCTCP, o algoritmo de ENCN implementado nos roteadores é simples, apenas marca pacotes de forma determinística com base em um limiar, deixando a tarefa maior para os transmissores que reagirão diante de uma notificação explícita de não congestionamento em um caminho de acordo com o algoritmo “Reação ENCN” a ser descrito a seguir.

4.2.2 Reação ENCN

Com a finalidade de casar ações TCP com ações AQM, quando o transmissor ENCN receber uma notificação explícita de não congestionamento em um caminho, ou seja, quando receber um ACK marcado com CE em 1, modificará sua janela de transmissão seguindo um algoritmo

semelhante ao TCP-Reno; porém, com as seguintes modificações:

- No caso em que o transmissor ENCN esteja na fase de partida lenta e receber um ACK marcado, o transmissor atualizará sua janela de transmissão (W) para $W = \min(rwnd, Ssth)$ e logo continuará em prevenção de congestionamento ($Ssth$ deve ter um valor inicial finito previamente definido);
- Na ausência de recepção de pacotes de ACKs marcados, a fase de partida lenta funcionará como em TCP-Reno;
- Na fase de prevenção de congestionamento, o ENCN implementa um mecanismo de balanço de justiça no qual o transmissor incrementará sua janela de congestionamento $cwnd$ em um pacote por RTT se, e somente se, receber pelo menos um ACK marcado durante esse RTT. Na ausência de recepção de ACKs marcados o transmissor reduzirá sua janela de congestionamento em um pacote após a recepção de BF (*Balance Fairness*) ACKs com o objetivo de manter a fila em torno de Q_{min} visando assim evitar tanto o fenômeno de *bufferempty* quanto o fenômeno de *bufferbloat*.

BF é uma variável ajustada em $Ssth$ pacotes cada vez que o transmissor entra na fase de prevenção de congestionamento, e durante essa fase segue a dinâmica dada na Tabela 4.1. BF determina assim um mecanismo de justiça para o ENCN.

Por conseguinte, quando um transmissor aumenta sua janela de transmissão (W), reduzirá sua variável BF e conseqüentemente irá reduzir sua janela com maior frequência que outro transmissor com menor valor de W . Por outro lado, devido ao fato de W não incrementar mais de uma vez por RTT, ou seja, não mais de uma vez por recepção de W ACKs, transmissores com maiores W incrementarão suas janelas com menor frequência do que aqueles com menor W .

Este mecanismo de justiça forçará cada transmissor que compartilha uma mesma topologia de rede de comunicação a fazer um justo compartilhamento da banda disponível.

As fases de retransmissão rápida e recuperação rápida funcionam como no TCP-Reno.

Observe que diferentemente de ECN, ENCN não necessita dos bits ECE e CWR do cabeçalho TCP (descritos na Subseção 2.6.4).

O autômato inicia no estado **Checking** e permanece nele durante $TStep$ unidades de tempo. Logo, se tiver pacotes na fila, isto é, se $QlenR1 \geq 1$ for verdadeiro, o autômato migra para o estado **Service** conforme ilustra a seta inferior para a esquerda na Figura 4.2. No estado **Service** o autômato permanece por $TqOneR$ unidades de tempo, equivalente ao tempo necessário para colocar um pacote de ACK em um canal de 10 Mbps (canal 1 sentido receptor/transmissor). Em seguida o autômato retorna para o estado **Checking**.

Na transição do estado **Service** para o estado **Checking**, existem dois caminhos possíveis. A transição pelo caminho inferior da Figura 4.2 acontece quando o tamanho da fila no caminho de ida no roteador 1 (fila de pacotes de dados que viajam do transmissor para o receptor) for menor ou igual que o limiar $Qmin$, ou seja, quando a condição $QlenF1 \leq Qmin$ for verdadeira, nessa transição o automato reinicia seu relógio $tq1r$ e chama a função **DequeueOneR**. A função **DequeueOneR** basicamente modela a saída do ACK dessa fila e sua entrada no canal 1 (no sentido receptor/transmissor). Assim, o pacote de ACK é enviado para o transmissor sem alterar o bit CE do seu cabeçalho IP, notificando deste modo que o caminho no sentido transmissor/receptor no roteador 1 não está congestionado.

A transição do estado **Service** para o estado **Checking** pelo caminho superior da Figura 4.2 acontece quando o tamanho da fila no caminho no sentido transmissor/receptor no roteador 1 for maior que o limiar $Qmin$, isto é, quando a condição $QlenF1 > Qmin$ for verdadeira. Nesta transição o autômato reinicia seu relógio $tq1r$ e chama as funções **RemoveENCNQOne** e **DequeueOneR**. A função **RemoveENCNQOne** basicamente coloca o bit CE do cabeçalho IP do ACK em 0 (caso esse bit ainda não tenha sido colocado em zero por outros roteadores). Assim, o bit CE é removido antes do pacote de ACK ser enviado para o transmissor.

Se estando no estado **Checking** passar $TStep$ unidades de tempo, ou seja, $tq1r \geq TStep$ for verdadeiro, e não houver pacotes na fila, ou seja, $QlenR1 < 1$ também for verdadeiro, então o autômato faz uma transição para esse mesmo estado e reinicia o relógio $tq1r$ conforme ilustra a seta da direita na Figura 4.2.

Tabela 4.2: Descrição das variáveis, funções e constantes utilizadas no modelo da fila de retorno no roteador 1 com a técnica AQM ENCN.

Nome	Descrição
TStep	Passo de tempo (resolução temporal) - valor adotado = $1\mu s$.
tq1r	Relógio que coordena as transições dos estados Checking e Service.
TqOneR	Constante igual ao tempo em μs requerido para colocar um ACK em um canal com capacidade 10 Mbps (canal 1).
QlenF1	Variável que contém a quantidade de pacotes de dados dentro da fila de ida do roteador 1.
DequeueOneR	Função chamada cada vez que um pacote sai da fila de retorno do roteador 1. Basicamente decrementa a variável QlenR1 em um pacote e incrementa a variável PqCh1R em um pacote.
QlenR1	Variável que contém a quantidade de pacotes de ACKs dentro da fila de retorno do roteador 1.
PqCh1R	Variável que contém a quantidade de pacotes de ACKs dentro do canal 1 de retorno.
Qmin	Limiar igual ao valor em bytes ou pacotes em que se deseja manter o tamanho da fila no gargalo.
RemoveENCNQOne	Função que basicamente remove a marca coloca em zero o bit CE da cabeça de um ACK quando a fila for menor que Qmin.

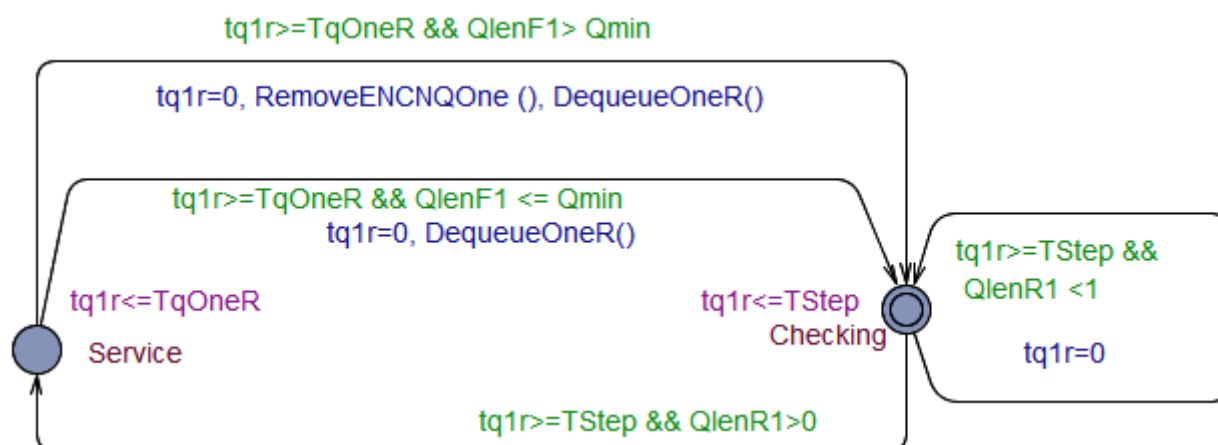


Figura 4.2: Modelo da fila de dados no caminho de retorno, no roteador 1.

Algoritmo 4.3 Pseudocódigo para o autômato da Figura 4.2.

<pre>Inicializa no estado Checking if (tq1r >= TStep) { if (QlenR1 > 0) { vai para o estado Service espera TqOneR unidades de tempo e então: if (QlenF1 <= Qmin) { retorna para o estado Checking, reinicia tq1r e chama a função DequeueOneR } else {retorna para o estado Checking, reinicia tq1r e chama as funções RemoveENCNQOne e DequeueOneR} } if (QlenR1 < 1) { Faz uma transição para o mesmo estado para reiniciar tq1r } } else { espera até tq1r >= TStep ser verdadeiro }</pre>	Observações Transição do lado esquerdo na Figura 4.2 Fim do bloco if (QlenF1 <= Qmin) Transição do lado direito na Figura 4.2 Fim do bloco (tq1r >= TStep)
--	--

Além dessas modificações, também foi modificado a função `DeliverAck` do autômato da Figura I.15 descrito no Anexo I. Assim, essa função não atualizará o valor de `cwnd` de acordo com o protocolo TCP-Reno conforme descrito na Tabela I.13, e sim de acordo com o protocolo ENCN descrito acima. Nesta função (`DeliverAck`) também foi implementado o mecanismo de balanço de justiça descrito na Tabela 4.1.

4.2.4 Avaliação do desempenho de ENCN em uma topologia daisy-chain

A técnica ENCN foi simulada no UPPAAL utilizando primeiramente a mesma topologia de rede de comunicação daisy-chain dada na Figura 3.3 com $Q_{min} = 4$ pacotes e $Ssthr$ inicial de 30 pacotes. A Figura 4.3 ilustra a dinâmica da Janela do transmissor ENCN enquanto que a dinâmica da fila do caminho de ida no roteador 1 é elucidada na Figura 4.4. Conforme pode ser observado o transmissor ENCN começa a transmitir pacotes na fase de partida lenta com $W=1$ segmento, onde um segmento corresponde a um pacote de 1 kbyte, como só existe um único fluxo na rede a fila no roteador começa vazia. Logo, ao começar o tráfego o transmissor recebe um ACK marcado indicando que existe pelo menos um caminho não congestionado entre ele e o receptor; então, o transmissor ajusta sua janela de transmissão para $W = \min(Ssthr, rwnd) = \min(30, 64) = 30$ segmentos. E sai da fase de partida lenta para entrar na fase de prevenção de congestionamento.

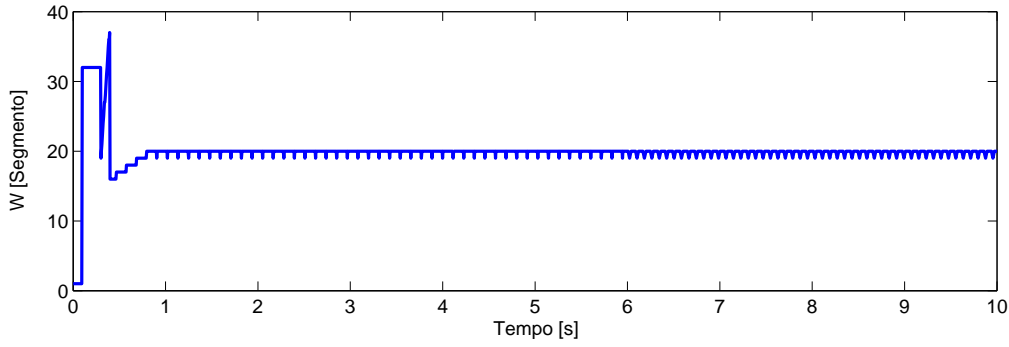


Figura 4.3: Janela do Transmissor TCP para ENCN implementada na topologia de rede de comunicação ilustrada na Figura 3.3.

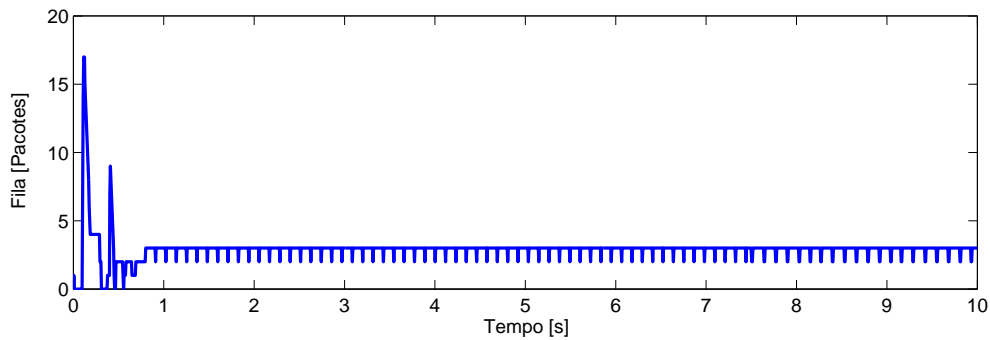


Figura 4.4: Fila no sentido transmissor/receptor do roteador 1 para ENCN implementada na topologia de rede de comunicação ilustrada na Figura 3.3.

O ajuste de $W=30$ segmentos faz com que o transmissor emita vários pacotes em rajada, que acabam saturando a fila no roteador e, conseqüentemente, ocorre perda de pacotes, que logo é detectada pelo transmissor mediante a recepção de ACKs duplicados. Ao receber 3 ACKs duplicados o transmissor retransmite o pacote considerado perdido e ajusta sua janela para $W = W/2 = 15$ segmentos, e então entra na fase de recuperação rápida onde permanecerá até receber ACKs para novos pacotes.

Na fase de recuperação rápida o transmissor não transmite novos pacotes antes de receber meia janela de ACKs, ou seja, deve esperar até que a diferença entre a seqüência do pacote a ser enviado e a seqüência do pacote cuja recepção foi confirmada por ACKs não seja maior que W , ou seja, 15 segmentos.

Essa espera por ACKs faz com que a fila no roteador reduza ficando menor que Q_{min} . Ao receber ACKs confirmando a recepção de novos pacotes, o transmissor entra em prevenção de congestionamento e devido ao fato de que todas as filas estão abaixo de Q_{min} os ACKs novamente chegarão marcados. Conseqüentemente, a janela de transmissão será aumentada em um segmento por RTT até o momento em que o tamanho da fila no roteador alcança Q_{min} . A partir desse momento o roteador remove a marca do cabeçalho IP dos ACKs e o transmissor mantém sua janela constante até receber *BF* ACKs não marcados. Em seguida, o transmissor reduz sua janela

em um segmento a fim de forçar uma redução no tamanho da fila, o que provoca que o roteador deixe de desmarcar ACKs, e ao receber um novo ACK marcado o transmissor aumenta novamente sua janela em um segmento.

Essa dinâmica é mantida até o final do período de simulação, provocando oscilações de pequena amplitude tanto na janela do transmissor, quanto na fila do roteador. Evitando assim tanto o fenômeno de *bufferbloat* quanto o fenômeno de *bufferempty*.

Comparando as Figuras 4.3 e 4.4 com as Figuras 3.10, 3.12, 3.14 e 3.16 observa-se que, em uma topologia de rede de comunicação daisy-chain, ENCN apresenta menos oscilações no tamanho da fila quando comparado com Drop Tail, RED, CoDel e PIE. Visto que após o início do transiente, a janela do transmissor e o tamanho da fila no roteador apresentam pequenas flutuações.

Por outro lado, a Figura 4.5 ilustra a vazão em função do tempo para as técnicas Drop Tail, RED, CoDel, PIE e ENCN implementadas na topologia de rede de comunicação daisy-chain ilustrada na Figura 3.3 e a Tabela 4.3 resume os valores finais dessas curvas. Conforme pode ser observado, entre as técnicas simuladas ENCN apresentou a melhor vazão.

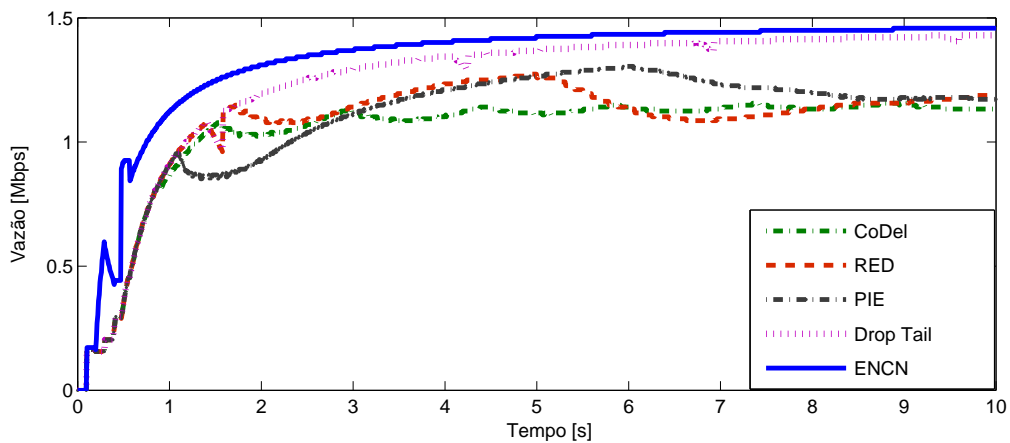


Figura 4.5: Vazão de ENCN comparada com outras técnicas AQM, implementadas na topologia de rede de comunicação dada na Figura 3.3.

Tabela 4.3: Vazão obtida na topologia de rede daisy-chain para diferentes técnicas AQM

Técnica AQM	Drop Tail	RED	CoDel	PIE	ENCN
Vazão [Mbps]	1,430	1,187	1,133	1,172	1,460

Esse bom desempenho do ENCN e de Drop tail em termos de vazão se deve ao fato de que, quando entram em regime estacionário, essas técnicas não apresentam o indesejado fenômeno de *bufferempty*, que no capítulo anterior foi identificado para as técnicas RED, CoDel e PIE. Assim, ENCN e Drop Tail apresentam melhor vazão quando comparadas com essas técnicas.

ENCN apresenta ainda melhor vazão que Drop Tail por fornecer menos perdas de pacotes. Note que com Drop Tail o transmissor somente reage a eventos de congestionamento quando detecta perdas de pacotes, e essas perdas afetam a vazão. Outra questão interessante a ser destacada é o fato de que como visto no capítulo anterior, a técnica Drop Tail fornece uma boa vazão a custo de

indesejados crescimentos na fila, que aumentam o tempo de espera na fila e consequentemente o RTT, fenômeno conhecido como *bufferbloat* [100, 101].

As técnicas CoDel e PIE evitam esse indesejado fenômeno de *bufferbloat*; porém, provocam o fenômeno contrário (*bufferempty*) e consequentemente prejudicam a vazão. Já a técnica ENCN fornece uma vazão melhor que Drop Tail e ainda, quando entra em regime, evita tanto o indesejado fenômeno de *bufferbloat* quanto o fenômeno de *bufferempty*. Já que a fila não cresce de forma ilimitada, e nem reduz a zero, mas é mantida ao redor de Q_{min} , fornecendo assim um RTT estável e em valores próximos ao mínimo, ou seja, evita a componente de atraso provocada por espera na fila. Isso resulta num bom desempenho para sistemas de controle em rede que possam compartilhar essa mesma topologia de rede de comunicação como será visto na Subsecção 4.2.6.

Conforme visto nas Figuras 4.4, 3.20 e 3.22 ENCN apresenta menos períodos de *bufferempty* do que TCP-Jersey e E-DCTCP, consequentemente (conforme pode ser observado na Figura 4.6) ENCN também fornece melhor vazão do que esses protocolos baseados em ECN. A Tabela 4.4 esboça os valores da vazão média aos 10 s de simulação para os protocolos TCP-Jersey, E-DCTCP e ENCN.

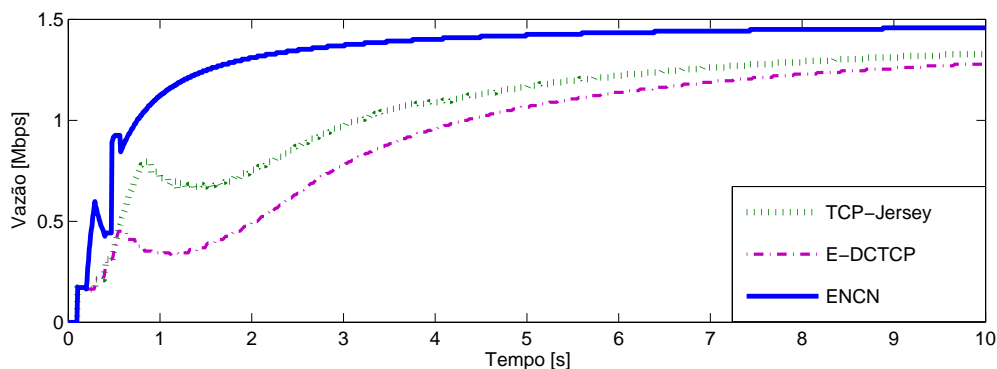


Figura 4.6: Vazão de ENCN comparada com protocolos TCP baseados em ECN 3.3.

Tabela 4.4: Vazão obtida na topologia de rede daisy-chain para diferentes protocolos aos 10 s de simulação.

Protocolo	TCP-Jersey	DCTCP	E-DCTCP	ENCN
Vazão [Mbps]	1,335	1,393	1,393	1,460

Na subsecção seguinte será avaliado o desempenho do ENCN comparando com outras técnicas em uma topologia de rede tipo *Dumbbell*, na qual também será possível avaliar o desempenho em termos de justiça, testando assim o algoritmo *Balance Fairness* do ENCN.

4.2.5 Avaliação do desempenho do ENCN em uma Topologia *Dumbbell*

Além da topologia de rede de comunicação daisy-chain ilustrada na Figura 3.3, o desempenho do ENCN foi também comparado com o desempenho de RED, CoDel e PIE em uma topologia de

rede de comunicação tipo *Dumbbell*, como a ilustrada na Figura 4.7, na qual três fluxos genéricos A, B e C compartilham a mesma rede de dois roteadores. As filas (ou *buffer*) possuem capacidades para enfileirar até 17 pacotes de 1 Kbyte cada um. Os roteadores são conectados por meio de um canal de 1,5 Mbps de capacidade e 5 ms de atraso de propagação.

Por outro lado, cada transmissor é conectado com o roteador 1 por meio de canais com capacidades de 10 Mbps e 2 ms de atrasos de propagação, enquanto que os receptores são conectados com o roteador 2 por meio de canais com capacidades de 10 Mbps e 33 ms de atraso de propagação cada um.

Essa topologia de rede de comunicação foi modelada no UPPAAL, onde além de comparar o desempenho de diferentes técnicas AQM em termos de vazão, estabilidade e tamanho da fila, também foi comparado o desempenho dessas técnicas quanto a justiça, utilizando para essa finalidade o índice de justiça de Jain (ver Equação (2.5)).

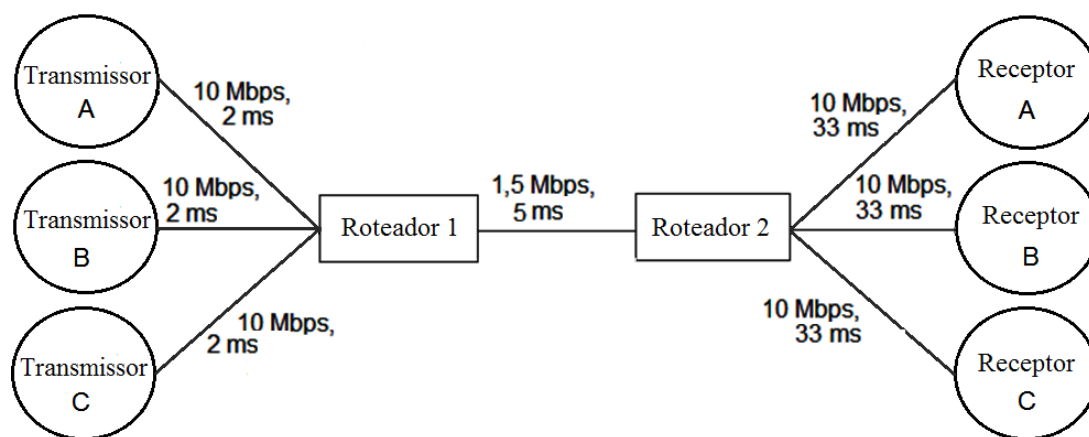


Figura 4.7: Topologia de rede de comunicação tipo *Dumbbell* com três fluxos TCP.

Para calcular o índice de Jain no UPPAAL, foi utilizado o autômato ilustrado na Figura 4.8. O qual é regido pela variável de relógio t_{jain} , e faz transições para um único estado a cada T_{Jain} unidades de tempo, sendo T_{Jain} uma constante igual a 1 ms. Em cada transição é chamada a função `ComputeJain ()`, a qual, basicamente calcula o índice de Jain seguindo a Equação (2.5).

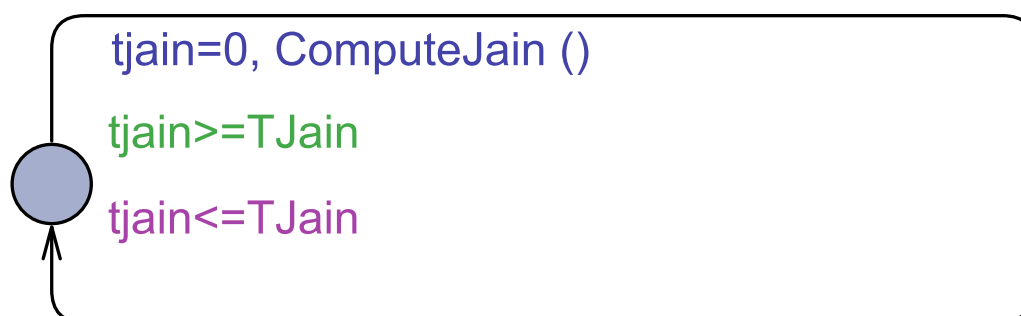


Figura 4.8: Autômato utilizado no UPPAAL para calcular o índice de Jain.

As Figuras 4.9, 4.10, 4.11 e 4.12 apresentam a vazão utilizando as técnicas AQM RED, CoDel,

PIE e ENCN, respectivamente, para cada um dos três fluxos (A, B e C). Conforme pode ser observado nessas figuras, em ENCN os três fluxos A, B e C, possuem suas respectivas curvas de vazão mais próximas uma da outra do que as outras técnicas AQM, ou seja, com a implementação de ENCN cada fluxo usa aproximadamente a mesma porção da banda disponível.

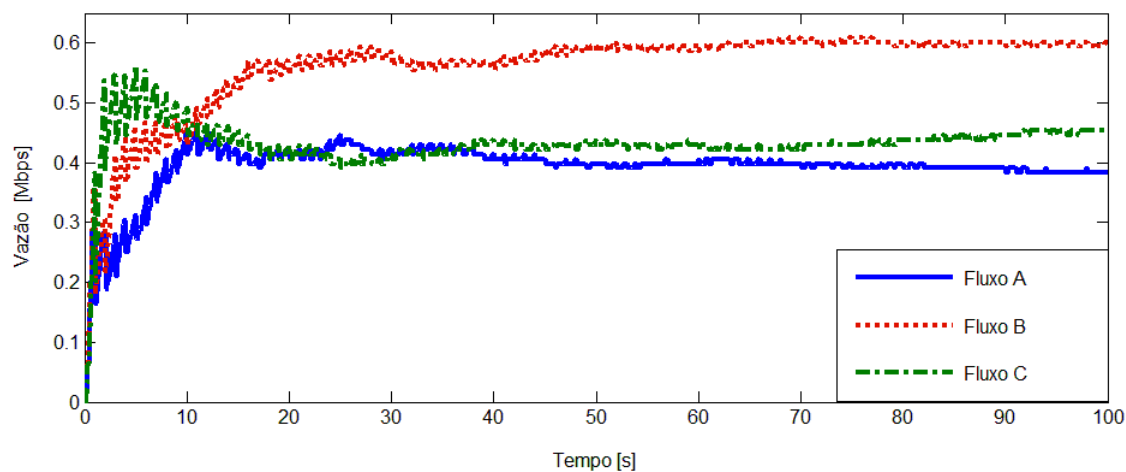


Figura 4.9: Vazão para a implementação RED na topologia de rede de comunicação ilustrada na Figura 4.7.

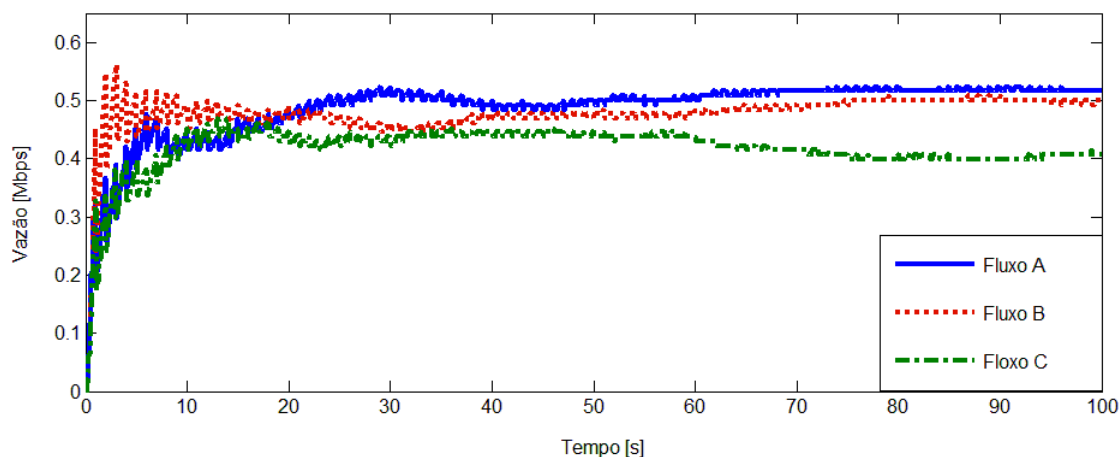


Figura 4.10: Vazão para a implementação CoDel na topologia de rede de comunicação ilustrada na Figura 4.7.

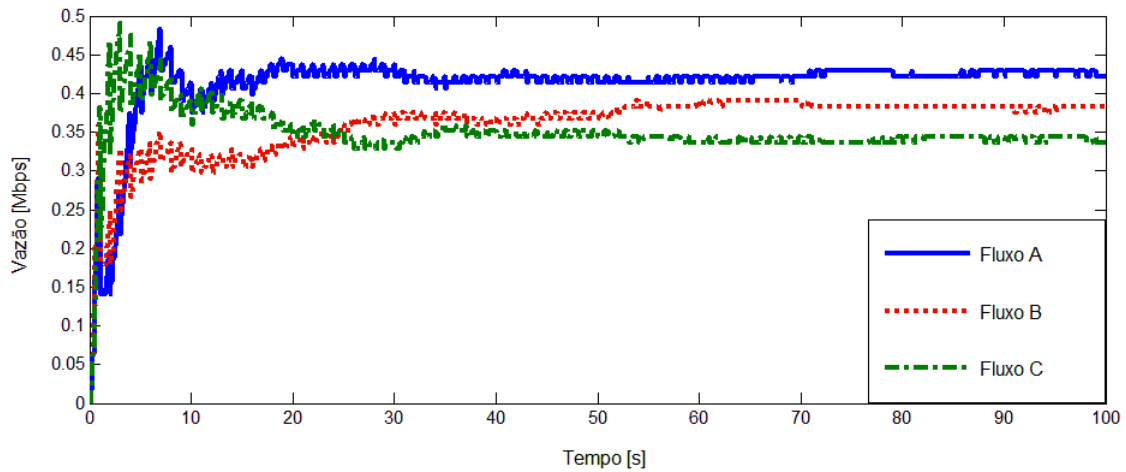


Figura 4.11: Vazão para a implementação PIE na topologia de rede de comunicação ilustrada na Figura 4.7.

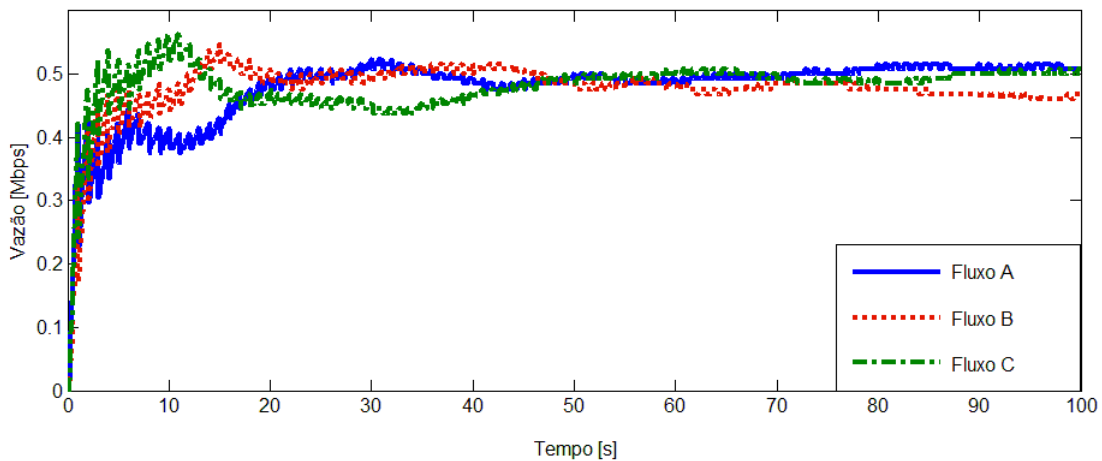


Figura 4.12: Vazão para a implementação ENCN na topologia de rede de comunicação ilustrada na Figura 4.7.

Esse justo compartilhamento da banda pode ser observado de forma mais nítida na Figura 4.13 que mostra o gráfico temporal do índice de Jain entre os três fluxos A, B e C para cada técnica AQM simulada. Pode ser observado que a curva do índice de Jain para ENCN permanece mais tempo próximo a 1 (máximo valor possível) do que as outras técnicas simuladas, sendo o pior resultado para RED.

Finalmente a Tabela 4.5 resume os valores finais da vazão para cada um dos fluxos, a vazão total e o valor final do índice de Jain, fornecidos por cada uma das técnicas AQM implementadas nos roteadores. A vazão total foi calculada como a soma das vazões de cada um dos fluxos (A, B e C), onde idealmente (para uma utilidade de 100 %) a vazão total seria de 1,5 Mbps (igual ao valor da banda no canal de gargalo).

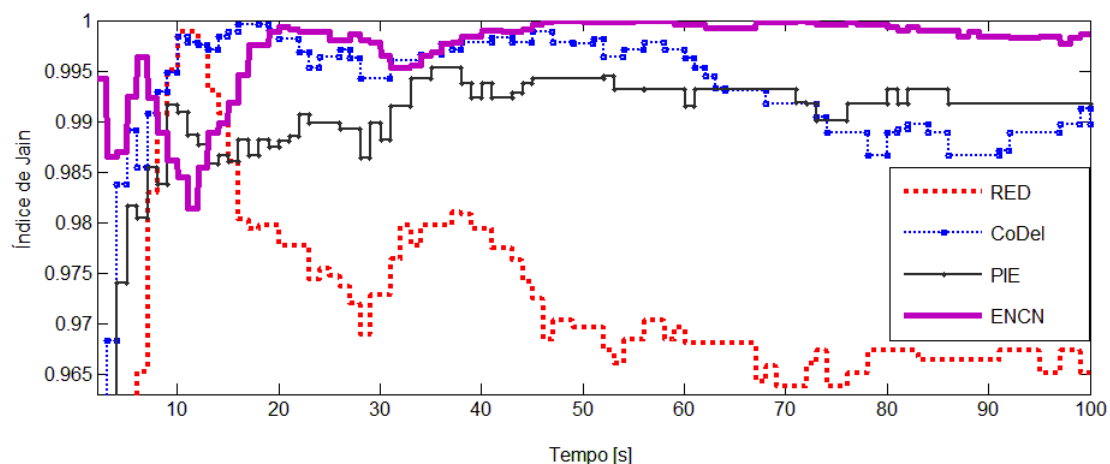


Figura 4.13: Índice de Jain para a topologia de rede dada na Figura 4.7 para diferentes técnicas AQM.

Tabela 4.5: Vazão total e Índice de Jain ao final da simulação.

	Vazão do Fluxo A [Mbps]	Vazão do Fluxo B [Mbps]	Vazão do Fluxo C [Mbps]	Vazão total [Mbps]	Índice de Jain
RED	0,383	0,601	0,453	1,437	0,965
CoDel	0,515	0,500	0,462	1,477	0,989
PIE	0,422	0,382	0,336	1,140	0,991
ENCN	0,5079	0,4688	0,5079	1,484	0,998

Entre as técnicas AQM simuladas, ENCN apresentou melhor desempenho em termos de vazão e justiça. Para poder entender a razão dessa melhor vazão fornecida pelo ENCN, nas Figuras 4.14, 4.15, 4.16 e 4.17 são ilustradas as dinâmicas da fila no gargalo fornecida por RED, CoDel, PIE e ENCN, respectivamente. Conforme pode ser observado, análogo ao caso da topologia de rede de comunicação daisy-chain (abordada na seção anterior) nesta topologia de rede tipo *Dumbbell*, ENCN também evita o indesejado fenômeno de *bufferempty*, conseqüentemente, o roteador foi mantido ocupado, e logo obteve-se uma boa vazão. Já as outras técnicas AQM simuladas (RED, CoDel e PIE) apresentam vários períodos nos quais a fila no roteador fica completamente vazia, assim o roteador fica desocupado e a banda disponível não é aproveitada.

Por outro lado, o melhor desempenho quanto à justiça fornecido pelo ENCN, se deve ao mecanismo de justiça nele implementado (veja a Tabela 4.1).

Outra questão interessante que pode ser observado nas Figuras 4.14, 4.15, 4.16 e 4.17 é o fato de que o ENCN e o PIE apresentam o melhor desempenho quanto à estabilidade da fila, e evitam o problema de *bufferbloat* (já que o tamanho da fila se mantém em valores baixos (em torno de Q_{min} para o caso do ENCN)). Neste sentido, RED apresentou o pior desempenho (já que a fila frequentemente cresce até valores próximos ao valor máximo), provocando assim longos atrasos ponta a ponta.

Conforme visto em seções anteriores esse problema de *bufferbloat* é altamente maléfico para sistemas de controle que eventualmente utilizem essa topologia de rede. Assim sendo, na Subseção 4.2.6 na topologia de rede de comunicação da Figura 4.7 será adicionado um quarto fluxo, correspondente ao exemplo de um sistema de controle em rede abordado na Seção 3.7 do Capítulo 3, no qual controlador e planta trocam dados mediante um fluxo UDP compartilhando a rede de comunicação.

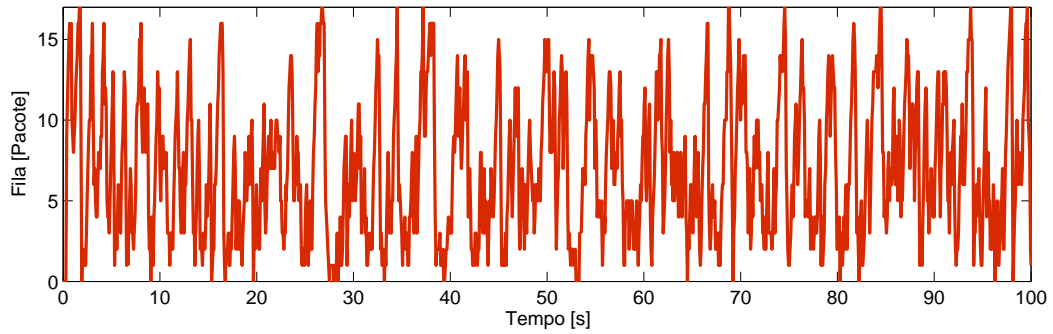


Figura 4.14: Fila para RED implementado na topologia de rede de comunicação ilustrada na Figura 4.7.

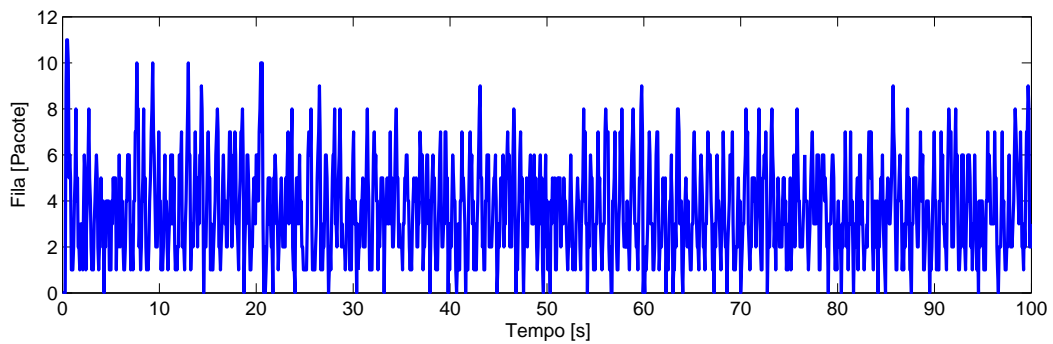


Figura 4.15: Fila para CoDel implementado na topologia de rede de comunicação ilustrada na Figura 4.7.

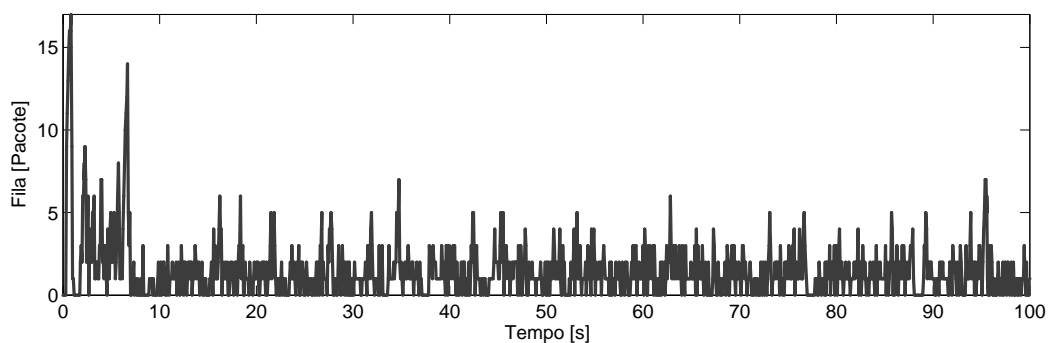


Figura 4.16: Fila para PIE implementado na topologia de rede de comunicação ilustrada na Figura 4.7.

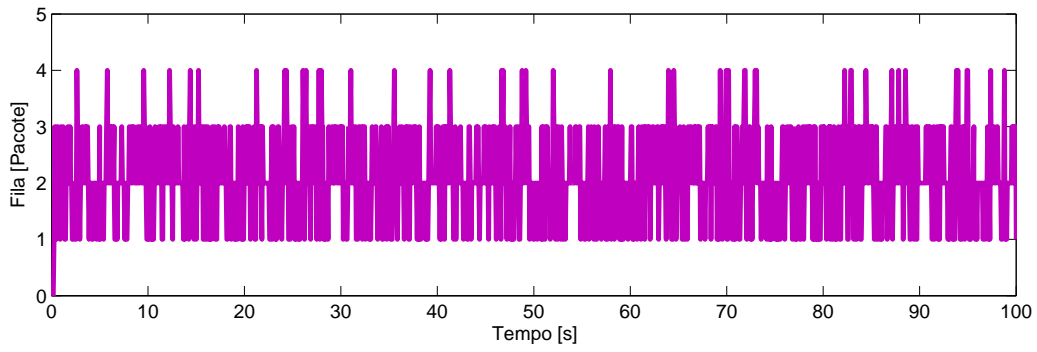


Figura 4.17: Fila para ENCN implementado na topologia de rede de comunicação ilustrada na Figura 4.7.

Comparação do desempenho de ENCN com protocolos baseados em ECN

Nesta mesma topologia o desempenho de ENCN também foi comparado com os protocolos TCP-Jersey e E-DCTCP que são baseados em ECN. As Figuras 4.18 e 4.19 esboçam a vazão para os fluxos A, B e C obtidos com a implementação de TCP-Jersey e o E-DCTCP respectivamente. E a Tabela 4.6 resume os valores finais da vazão para cada um dos fluxos, a vazão total e o valor final do índice de Jain, fornecidos por cada um desses protocolos e por ENCN.

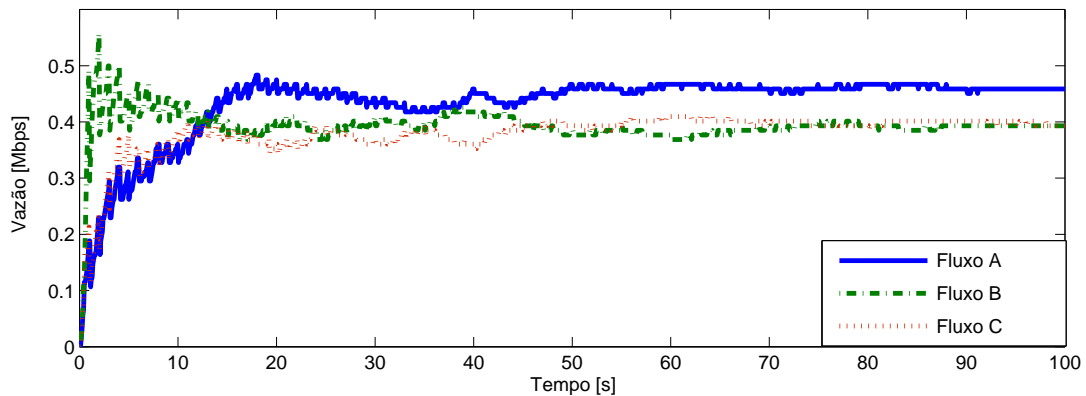


Figura 4.18: Vazão para TCP-Jersey implementado na topologia de rede de comunicação ilustrada na Figura 4.7.

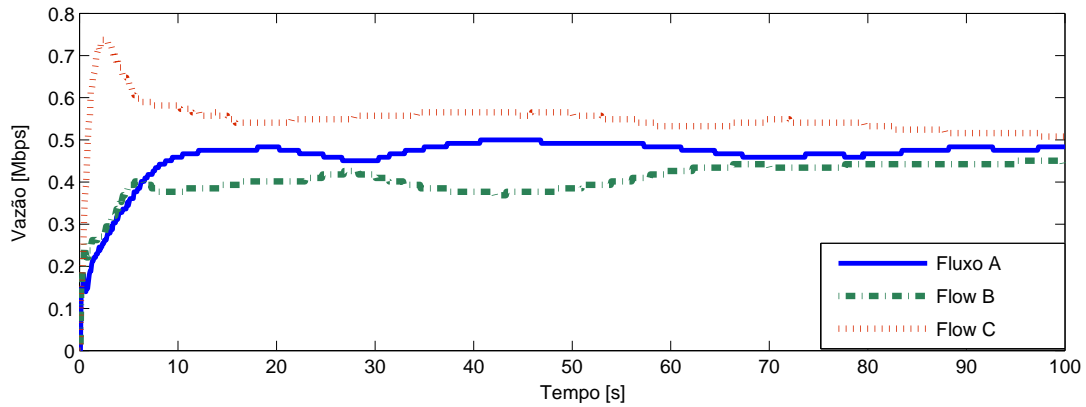


Figura 4.19: Vazão para E-DCTCP implementado na topologia de rede de comunicação ilustrada na Figura 4.7.

Conforme pode ser observado ENCN apresentou melhor vazão do que esses protocolos baseados em ECN. A razão dessa melhor vazão se deve ao fato de que o TCP-Jersey e o E-DCTCP apresentaram vários períodos de *bufferempty*.

As Figuras 4.20 e 4.21 ilustram as dinâmicas das filas no roteador 1 obtidas com a implementação de TCP-Jersey e E-DCTCP, respectivamente. Conforme pode ser observado, TCP-Jersey apresenta mais períodos de *bufferempty* do que o E-DCTCP e conseqüentemente apresenta também o pior desempenho em termos de vazão. Por outro lado, os valores máximos dos tamanhos das filas foram de 12 pacotes para TCP-Jersey e de 7 pacotes para E-DCTCP, respectivamente, ou seja, não ocorreram perdas de pacotes por transbordamento do *buffer* (já que o *buffer* tem capacidade para enfileirar até 17 pacotes de 1 kbyte cada) sendo assim o fenômeno de *bufferempty* o único fator que afetou negativamente a vazão.

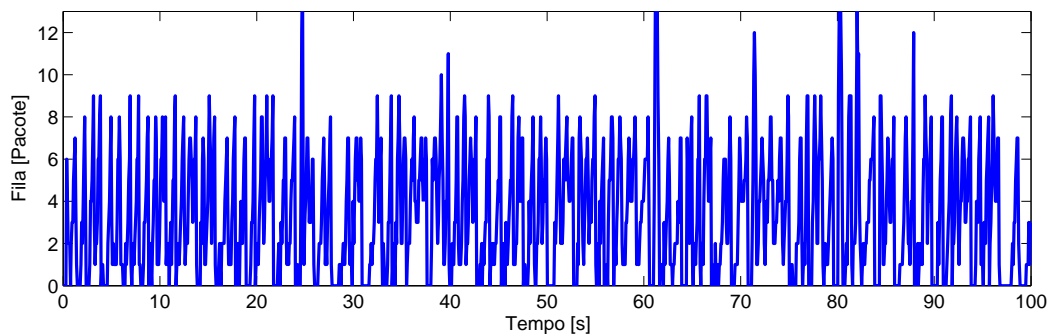


Figura 4.20: Fila no caminho de ida do roteador 1 para TCP-Jersey implementado na topologia de rede de comunicação ilustrada na Figura 4.7.

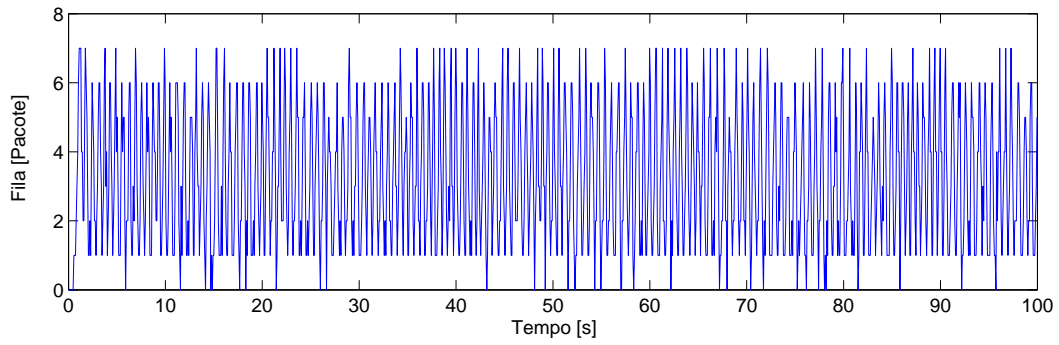


Figura 4.21: Fila no caminho de ida do roteador 1 para E-DCTCP implementado na topologia de rede de comunicação ilustrada na Figura 4.7.

Conforme pode ser observado na Figura 4.22 e na Tabela 4.6, os três protocolos comparados forneceram um justo compartilhamento da banda do canal com um índice de Jain bem próximo da unidade. Porém, o ENCN forneceu um valor do índice de Jain ligeiramente melhor do que o TCP-Jersey e o E-DCTCP.

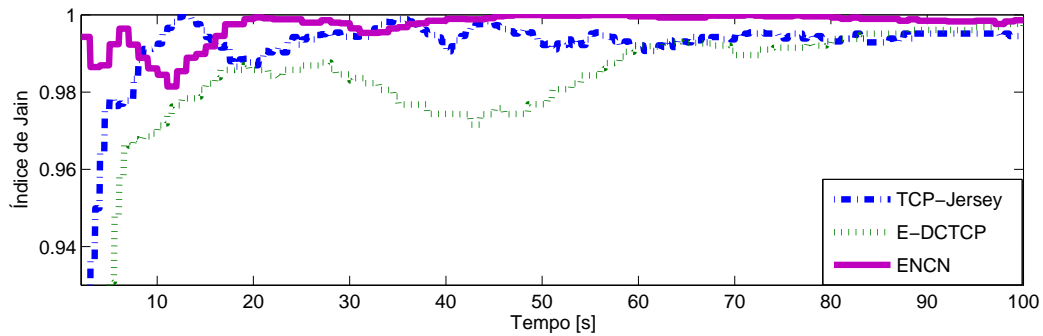


Figura 4.22: Índice de Jain para a topologia de rede dada na Figura 4.7 para diferentes protocolos.

Tabela 4.6: Vazão total e Índice de Jain ao final da simulação.

	Vazão do Fluxo A [Mbps]	Vazão do Fluxo B [Mbps]	Vazão do Fluxo C [Mbps]	Vazão total [Mbps]	Índice de Jain
TCP-Jersey	0,459	0,393	0,393	1,245	0,994
E-DCTCP	0,483	0,442	0,508	1,433	0,996
ENCN	0,508	0,468	0,5079	1,484	0,998

4.2.6 Avaliação do desempenho do ENCN em uma NCS

Na Seção 3.7 foi identificada a existência de uma compensação entre vazão de fluxos genéricos e desempenho de uma NCS que compartilham uma mesma topologia de rede de comunicação. Observou-se que as técnicas AQM que fornecem boa vazão para os fluxos TCP (como no caso de Drop Tail e RED) geram filas de maiores tamanho (fenômeno de *bufferbloat*) e consequentemente

maiores atrasos ponta a ponta resultando em um pior desempenho para sistemas de controle que compartilham a mesma topologia de rede de comunicação.

Por outro lado, técnicas AQM que evitam o fenômeno de *bufferbloat* (como no caso de CoDel e PIE), não permitem que a fila cresça demasiado; e assim, fornecem menores atrasos ponta a ponta e consequentemente bom desempenho para a NCS. Porém, provocam o fenômeno de *bufferempty* e consequentemente degradam a vazão para fluxos genéricos que compartilham a mesma topologia de rede de comunicação.

Com o objetivo de avaliar essa compensação entre vazão para fluxos genéricos e desempenho de uma NCS que compartilha a mesma topologia de rede de comunicação, seguindo a mesma metodologia descrita na Seção 3.3 foi modelada no UPPAAL a topologia de rede de comunicação apresentada na Figura 4.23, na qual, 3 fluxos TCP genéricos compartilham uma mesma topologia de rede de comunicação com um fluxo UDP de uma NCS.

Foram simulados os primeiros 30 s e foi avaliado o desempenho do sistema de controle em termos de ITAE, e o desempenho dos fluxos genéricos em termos de vazão e justiça. Para avaliar a justiça foi utilizado o índice de justiça de Jain. Porém, o critério de justiça foi avaliado unicamente para os três fluxos genéricos, já que o fluxo UDP da NCS é pequeno em relação aos fluxos TCP genéricos e, além disso, a vazão não é um parâmetro de desempenho para uma NCS.

Outra consideração é que a rede está habilitada para trabalhar com ECN ou ENCN e consequentemente as técnicas AQM atuam marcando pacotes ao invés de descartá-los (somente descartam pacotes se o *buffer* estiver completamente cheio) e somente os transmissores reagem a notificações de congestionamento. O fluxo UDP da NCS naturalmente não faz controle de congestionamento e não reage a pacotes marcados.

As técnicas AQM RED, CoDel e PIE foram avaliadas trabalhando em conjunto com TCP-Reno, também foi avaliado o desempenho de TCP-Jersey e E-DCTCP. Todos foram comparados com o protocolo ENCN (que atua tanto nos roteadores (como uma técnica AQM) quanto no transmissor (de forma análoga a um protocolo TCP)).

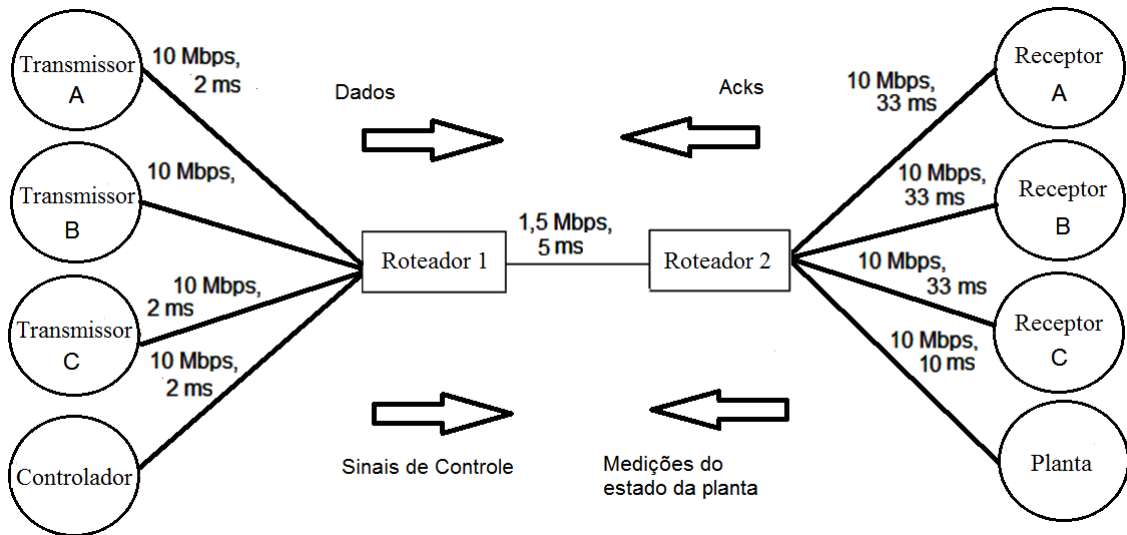


Figura 4.23: Topologia de rede de comunicação tipo *Dumbbell* compartilhada por três fluxos TCP genéricos e um fluxo UDP de uma NCS.

As Figuras 4.24, 4.25, 4.26 e 4.27 ilustram a posição do motor em função do tempo, para cada uma das técnicas AQM simuladas. Graficamente é possível observar que o erro na posição do motor obtido quando o ENCN for implementado é menor do que o erro obtido quando implementado as técnicas RED, CoDel e PIE.

Porém, com o objetivo de fazer comparações numéricas desse erro na Figura 4.28 é esboçada a avaliação temporal do desempenho do sistema de controle fornecido por cada uma dessas técnicas em termos do ITAE, cujo valor final (após 30 s) é dado na Tabela 4.7, na qual é possível observar que o ENCN reduziu o ITAE do sistema de controle (fluxo UDP) em 31,6 %, 31,54 % e 10,78 % comparado com RED, CoDel e PIE, respectivamente.

A fim de ter uma referência para fazer comparações também foi obtido o ITAE para o caso no qual o controlador está alocado junto com a planta. Observa-se na Tabela 4.7 que com a implementação de ENCN nos três fluxos TCP genéricos e nos roteadores, o ITAE da NCS piorou em 185% comparado ao valor obtido com o controlador junto com a planta. Já quando foram utilizadas as técnica AQM PIE, CoDel e RED o ITAE piorou em 220,3 %, 317,42 e 317,8%, respectivamente.

Esse bom desempenho do ENCN em relação as outras técnicas AQM se deve ao fato de que o ENCN conseguiu manter o tamanho da fila em valores baixos (próximos a zero), conforme pode ser observado na Figura 4.34, evitando assim o indesejado fenômeno de *bufferbloat* e, conseqüentemente proporcionou menores atrasos ponta a ponta que as técnicas RED, CoDel e PIE, fornecendo assim, bom desempenho para a NCS que depende fortemente de atrasos na rede. Neste sentido, RED forneceu o pior desempenho para o sistema de controle, isso foi conseqüência de que a dinâmica da fila obtida com RED (Figura 4.31) contém oscilações de grande amplitude e, portanto, períodos de grandes atrasos ponta a ponta.

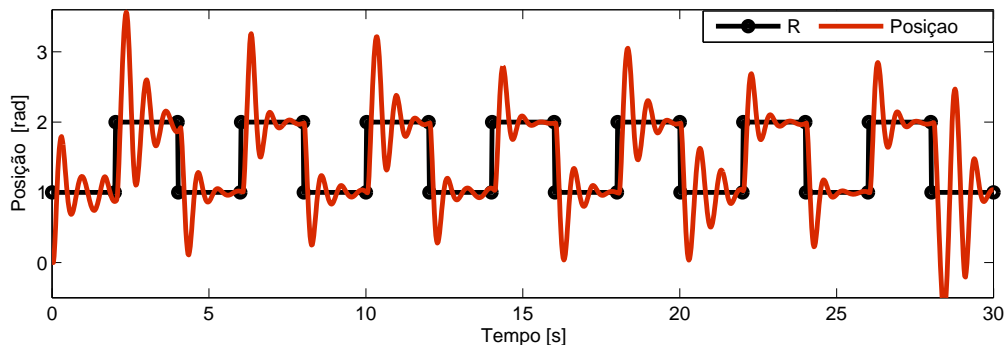


Figura 4.24: Posição do motor obtida com a implementação de RED.

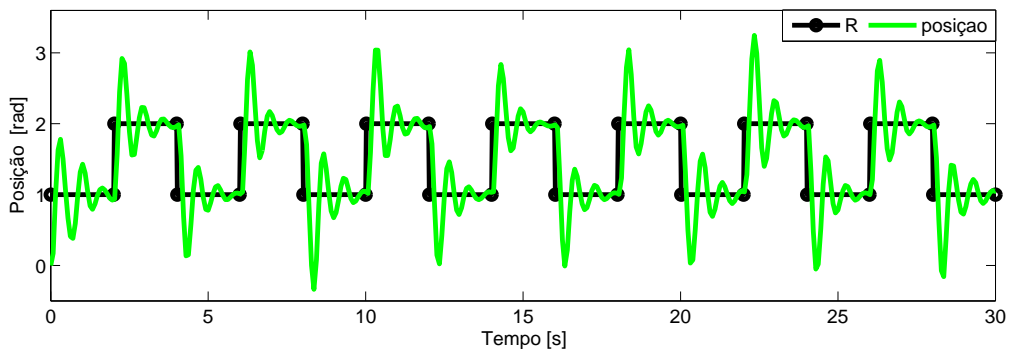


Figura 4.25: Posição do motor obtida com a implementação de CoDel.

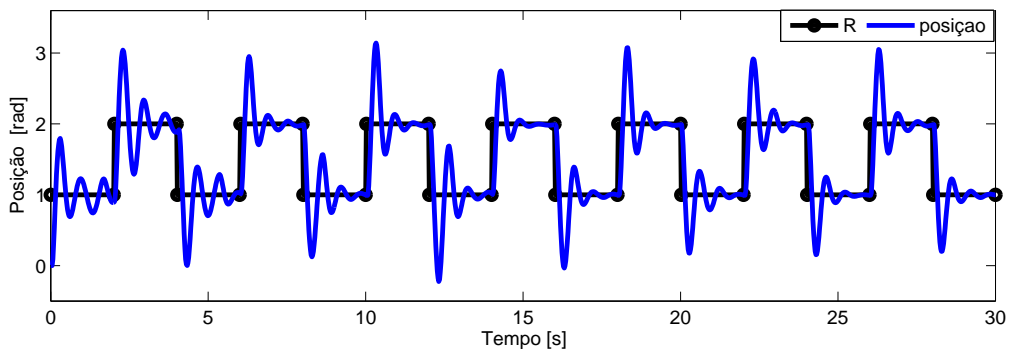


Figura 4.26: Posição do motor obtida com a implementação de PIE.

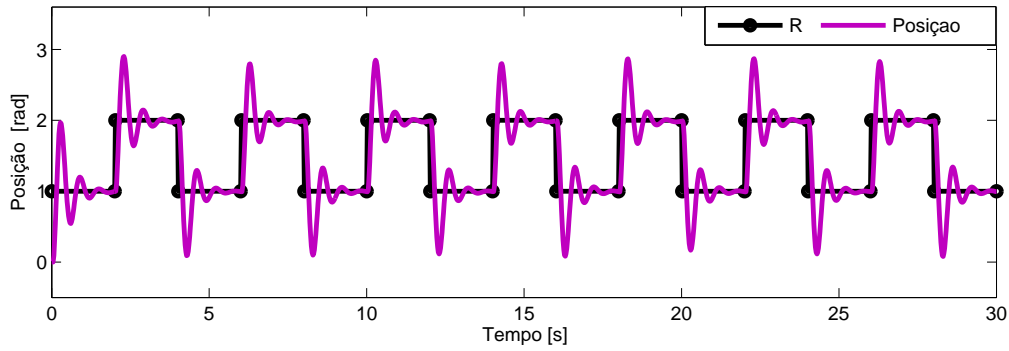


Figura 4.27: Posição do motor obtida com a implementação de ENCN.

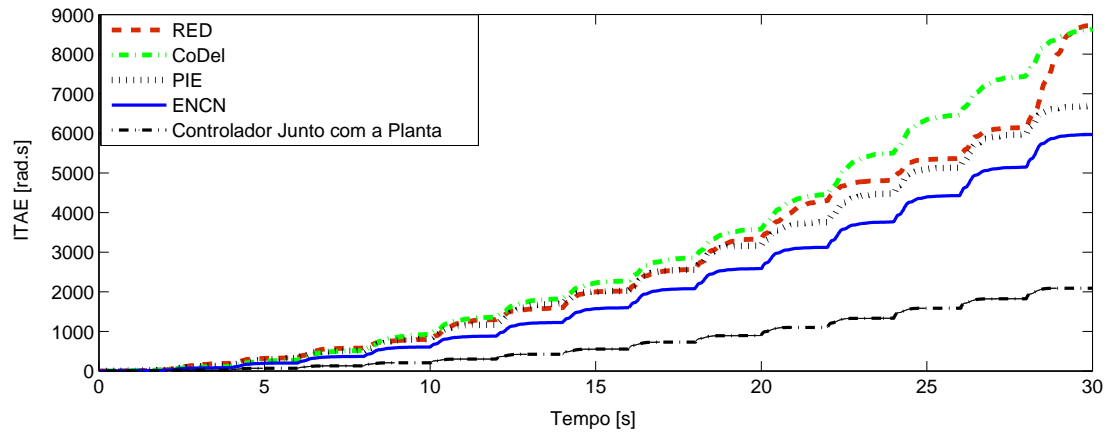


Figura 4.28: ITAE para ENCN comparada com outras técnicas AQM.

A avaliação dos fluxos genéricos quanto à vazão total é ilustrada na Figura 4.29, na qual a vazão total foi obtida como a somatória das vazões individuais para cada um dos três fluxos genéricos (A, B e C) que compartilham a mesma topologia de rede de comunicação com a NCS. Os valores da vazão obtida ao final dos 30 s de simulação são esboçados na Tabela 4.7. Conforme pode ser observado, o ENCN forneceu uma vazão 12 % melhor que RED, e 2,32 %, 14, 7 % melhor que CoDel e PIE, respectivamente.

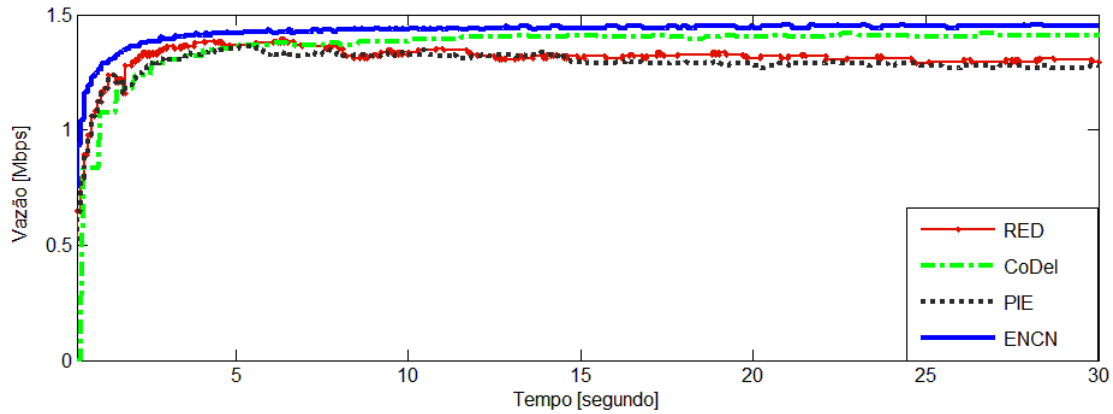


Figura 4.29: Vazão para ENCN comparada com outras técnicas AQM.

Tabela 4.7: Vazão total, índice de Jain e ITAE para diferentes técnicas AQM.

	Vazão total [Mbps]	Índice de Jain	ITAE [rad.s]
RED	1,294	0,981	8727
CoDel	1,417	0,978	8720
PIE	1,270	0,988	6691
ENCN	1,450	1	5970
Controlador Junto com a Planta			2089

Esse bom desempenho de ENCN em relação as outras técnicas AQM em termos de vazão se deve ao fato de que o ENCN conseguiu evitar períodos de *bufferempty* conforme pode ser observado na Figura 4.34, e conseqüentemente utilizou melhor a banda do canal gargalo. A técnica PIE forneceu o pior desempenho quanto a vazão por apresentar vários períodos de *bufferempty* como pode ser observado na Figura 4.33.

Finalmente, conforme pode ser observado na Figura 4.30, o ENCN apresentou uma boa justiça entre os três fluxos genéricos, ou seja, houve um justo compartilhamento da banda do canal, chegando a atingir o valor 1 (máximo valor possível) para o índice de Jain nos últimos segundos de simulação (veja a Tabela 4.7). Esse bom desempenho é causado pelo mecanismo de justiça implementado no ENCN (veja a Tabela 4.1).

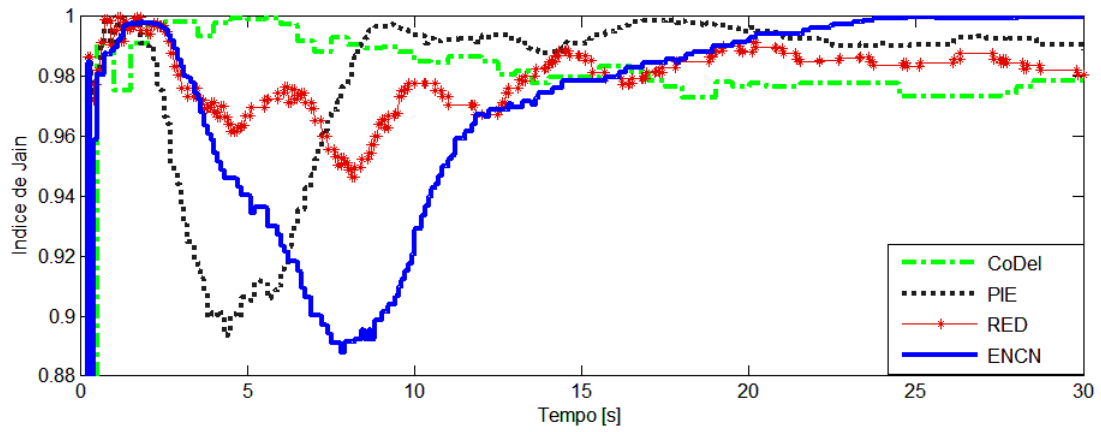


Figura 4.30: Índice de Jain para ENCN comparada com outras técnicas AQM.

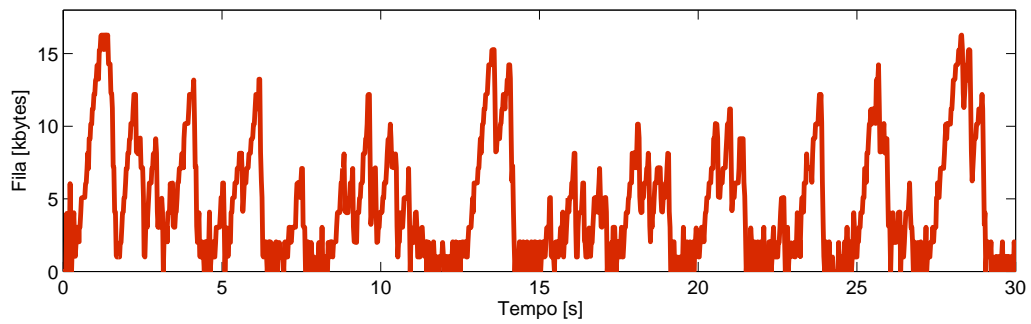


Figura 4.31: Fila para RED implementado na topologia de rede de comunicação ilustrada na Figura 4.23.

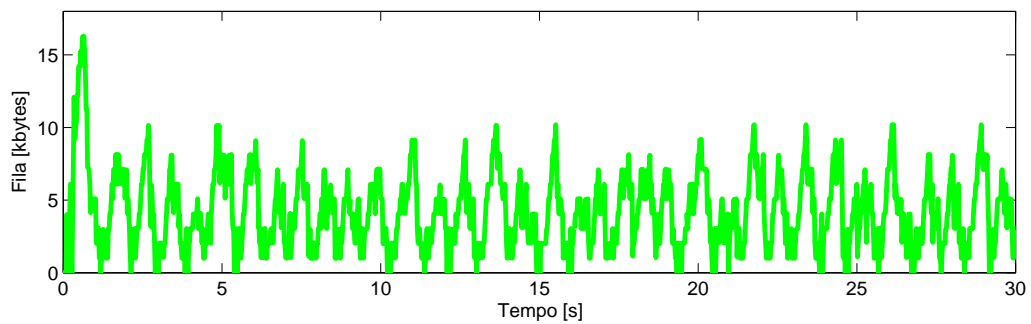


Figura 4.32: Fila para CoDel implementado na topologia de rede de comunicação ilustrada na Figura 4.23..

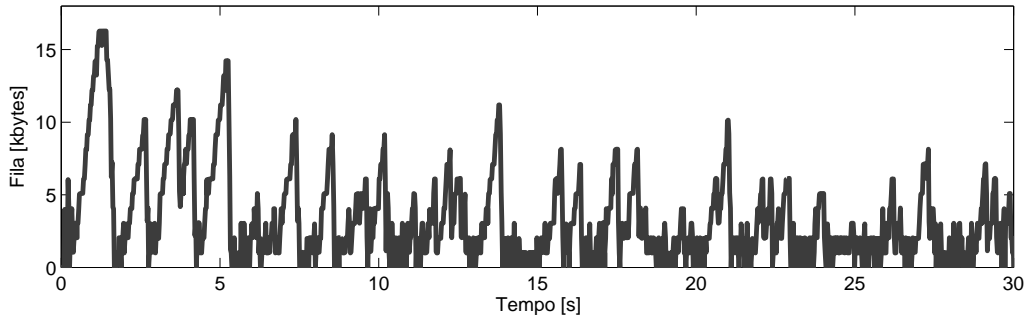


Figura 4.33: Fila para PIE implementado na topologia de rede de comunicação ilustrada na Figura 4.23..

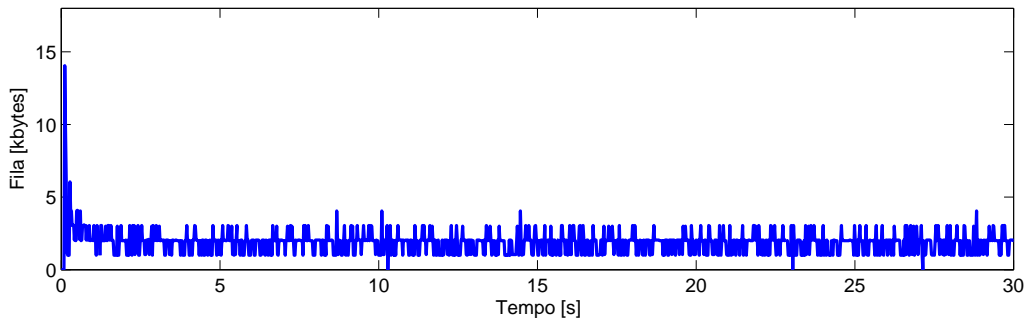


Figura 4.34: Fila para ENCN implementado na topologia de rede de comunicação ilustrada na Figura 4.23.

Comparação de técnicas AQM mediante SMC

Os resultados apresentados acima são os resultados de uma única simulação. Porém, na Internet, ocorrem eventos determinísticos e aleatórios, por exemplo, os tempos de propagações nos canais são determinísticos, mas algumas técnicas AQM agem de forma aleatória consequentemente os atrasos são aleatórios, assim simulações feitas com “sementes” diferentes podem gerar resultados diferentes.

Então, para fazer uma comparação mais adequada do desempenho fornecido pelas diferentes técnicas AQM comparadas, foi utilizado a ferramenta SMC do UPPAAL a qual calcula probabilidades com base nos resultados de várias simulações feitas com “sementes” diferentes.

Por conseguinte, para comparar o desempenho do ENCN com RED, CoDel e PIE em termos do ITAE para a NCS e da vazão para os fluxos genéricos, foram utilizados os maiores valores para ITAE e vazão obtidos na simulação anterior e esboçados na Tabela 4.7. E em seguida foi calculada a probabilidade de superar esses valores para cada uma das técnicas AQM comparadas.

Assim, para calcular a probabilidade do ITAE ser maior que 8727 rad.s (maior valor obtido por simulação) em até 30 s, foi usada a sintaxe $\text{Pr } [\leq 30000000] \{ \langle \rangle \text{ITAE} > 8727 \}$ onde ITAE é a variável que representa o ITAE. No UPPAAL está sendo utilizado μs (microsegundos) como

unidade de tempo; assim, 30000000 correspondem que 30 s.

A Tabela 4.8 apresenta o intervalo de probabilidade estimado com uma confiança de 95 % para RED, CoDel, PIE e ENCN. Conforme pode ser observado ENCN possui a menor probabilidade de gerar um ITAE maior que 8727 rad.s em até 30 s.

A Figura 4.35 ilustra a probabilidade acumulada para RED, CoDel e PIE. Para ENCN o SMC não gerou gráfico de probabilidade acumulada, já que em nenhuma das simulações usadas para estimar a probabilidade o valor do ITAE ultrapassou 8727 rad.s em um tempo menor ou igual que 30 s. Também pode ser observado que o ITAE gerado com a implementação da técnica RED possui a maior probabilidade de ultrapassar esse valor. Isso ocorre porque RED apresenta maiores fenômenos de *bufferbloat* que as outras técnicas AQM verificadas.

Tabela 4.8: Intervalo de probabilidade do ITAE ser maior que 8727 rad.s em até 30 s.

Técnica AQM	RED	CoDel	PIE	ENCN
Intervalo de Probabilidade	0,649 a 0,749	0,457 a 0,557	0,068 a 0,168	0 a 0,097

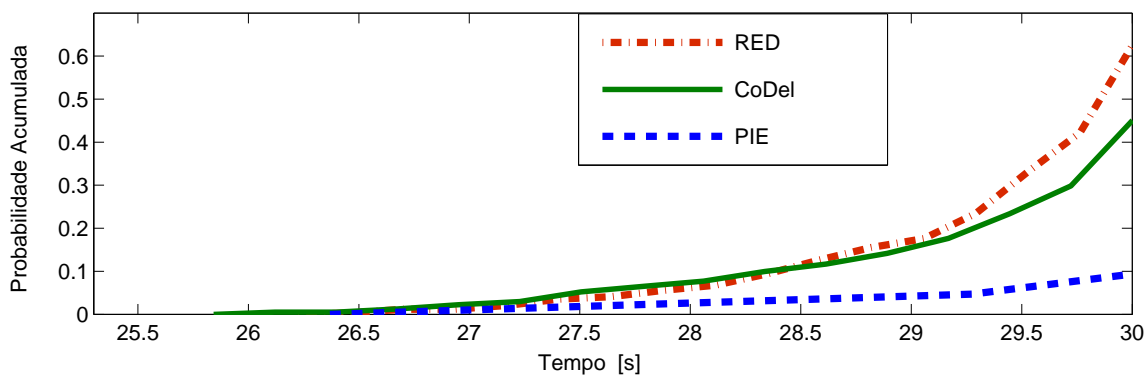


Figura 4.35: Probabilidade acumulada do ITAE ser maior que 8727 rad.s em até 30 s.

As Figuras 4.36, 4.37 e 4.38 ilustram os histogramas de frequência para RED, CoDel e PIE respectivamente. Conforme pode ser observado os intervalos de tempo mais prováveis do ITAE ultrapassar 8727 rad.s ocorrem nas classes com ponto médio entre 29 e 30 s, isso é esperado, devido a que a curva do ITAE é crescente em função do tempo.

Na Figura 4.38 observa-se que em apenas 18 simulações a técnica AQM PIE gerou um ITAE maior que 8727 rad.s em até 30 s. Já com a implementação de RED (Figura 4.36) e CoDel (Figura 4.37) isso ocorreu mais de 100 vezes, e em ENCN nenhuma vez (por isso não gerou gráficos). Constata-se assim, que ENCN fornece melhor desempenho para a NCS do que as outras técnicas AQM verificadas com SMC.

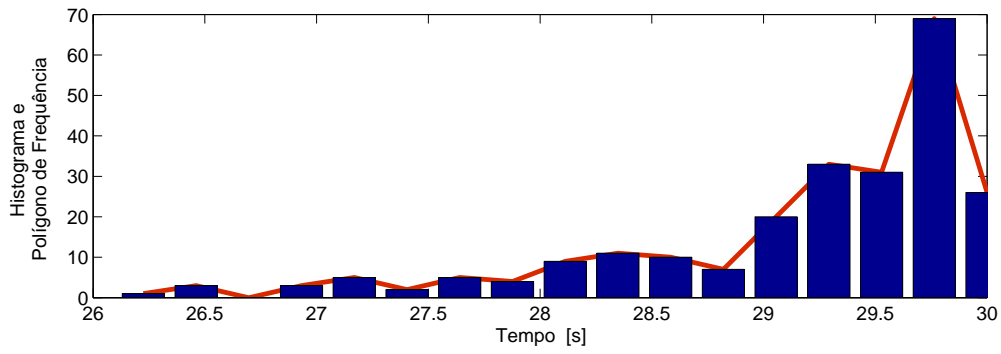


Figura 4.36: Histograma e Polígonos de frequências da probabilidade do ITAE gerado por RED ser maior que 8727 rad.s em até 30 s.

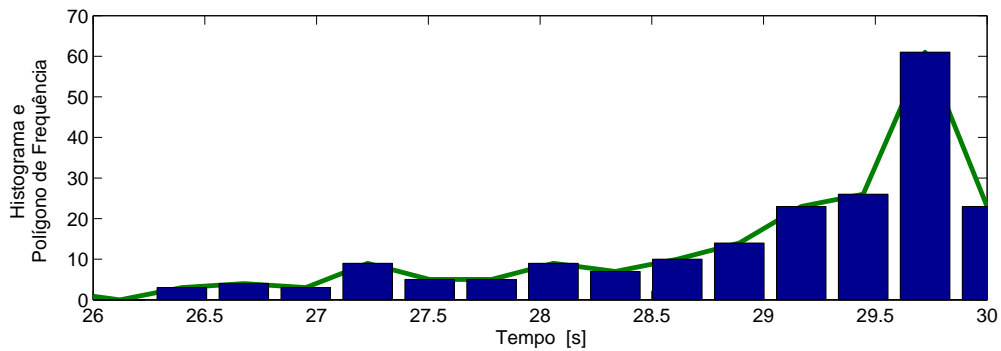


Figura 4.37: Histograma e Polígonos de frequências da probabilidade do ITAE gerado por CoDel ser maior que 8727 rad.s em até 30 s.

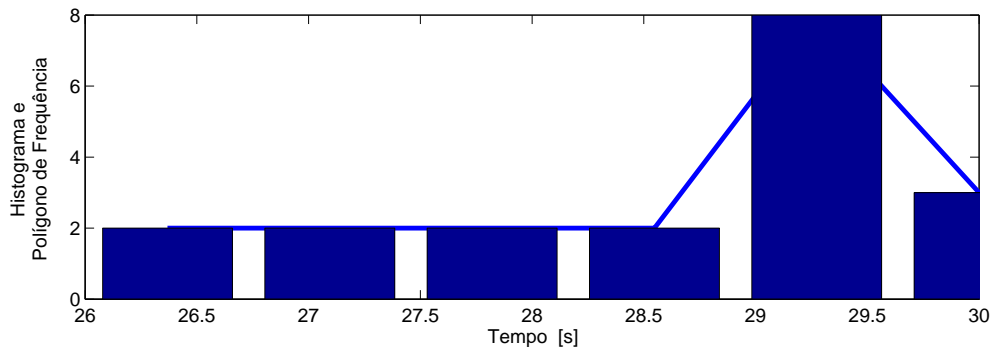


Figura 4.38: Histograma e Polígonos de frequências da probabilidade do ITAE gerado por PIE ser maior que 8727 rad.s em até 30 s.

Também foi calculada a probabilidade da vazão total (soma da vazão dos três fluxos genéricos A, B e C) ser maior que 1,45 Mbps (que conforme pode ser visto na Tabela 4.7, foi a máxima vazão obtida nas simulações) em um tempo menor ou igual que 30 s. Na Tabela 4.9 são mostrados os intervalos de probabilidade estimados com uma confiança de 95 % para RED, CoDel, PIE e ENCN. Conforme pode ser observado entre as técnicas AQM comparadas, ENCN possui a maior

probabilidade de gerar uma vazão total maior que 1,45 Mbps em um tempo menor ou igual que 30 s.

Na Figura 4.39 é apresentada a probabilidade acumulada para ENCN e CoDel como pode ser observado a probabilidade assume valores diferentes de zero a partir dos 5 s (aproximadamente). Isso acontece porque as curvas de vazão ficam mais estáveis a partir desse instante (veja a Figura 4.29); assim, em algumas simulações a vazão consegue atingir um valor maior do que 1,45 Mbps já em torno aos primeiros 5 s.

Isso fica ainda mais nítido nas Figuras 4.40 e 4.41 que ilustram o histograma e o polígono de frequência para ENCN e CoDel, respectivamente. Observa-se que embora os intervalos de classes mais repetidos ocorrem nas classes com ponto médio maior que 20 s, classes com ponto médio em torno de 5 s também possuem uma frequência diferente de zero.

Note que o fato da vazão atingir 1,45 Mbps aos 5 s não significa que aos 30 s ela terá um valor maior ou igual que 1,45 Mbps, pois diferentemente da curva do ITAE a curva da vazão pode ficar constante, crescer ou decrescer em função do tempo (veja a Equação (3.1)).

Tabela 4.9: Intervalo de probabilidade da vazão total ser maior que 1,45 Mbps em até 30 s.

Técnica AQM	RED	CoDel	PIE	ENCN
Intervalo de Probabilidade	0 a 0,097	0,074 a 0,174	0 a 0,097	0,637 a 0,737

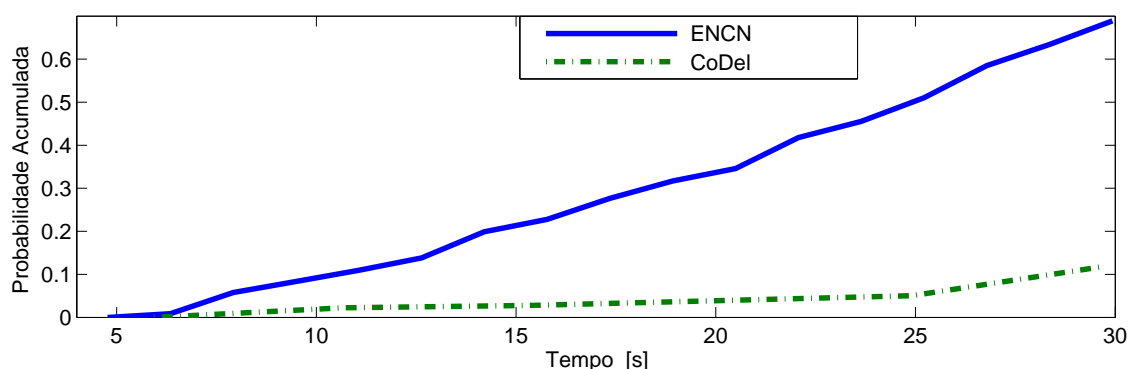


Figura 4.39: Probabilidade acumulada da vazão total ser maior que 1,45 Mbps em até 30 s.

O SMC não gerou gráficos de histogramas e polígonos de frequências para RED e PIE já que em nenhuma simulação essas técnicas forneceram uma vazão total maior que 1,45 Mbps em um tempo menor ou igual que 30 s. Logo, a frequência foi zero para todas as classes.

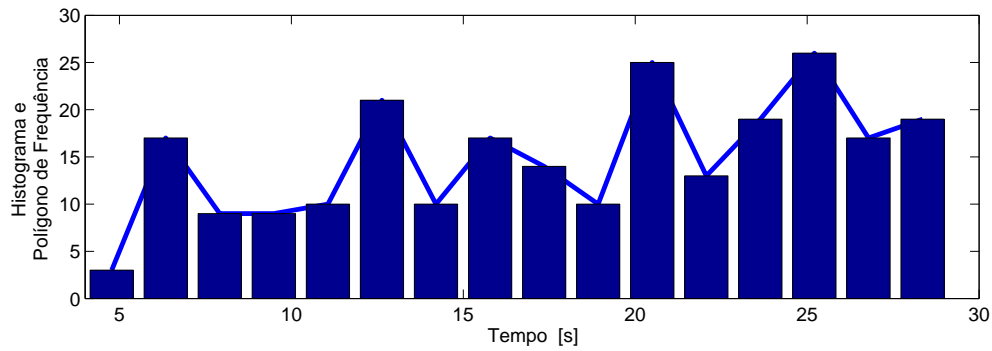


Figura 4.40: Histograma e Polígonos de frequências da probabilidade da vazão total gerada por ENCN ser maior que 1,45 Mbps em até 30 s.

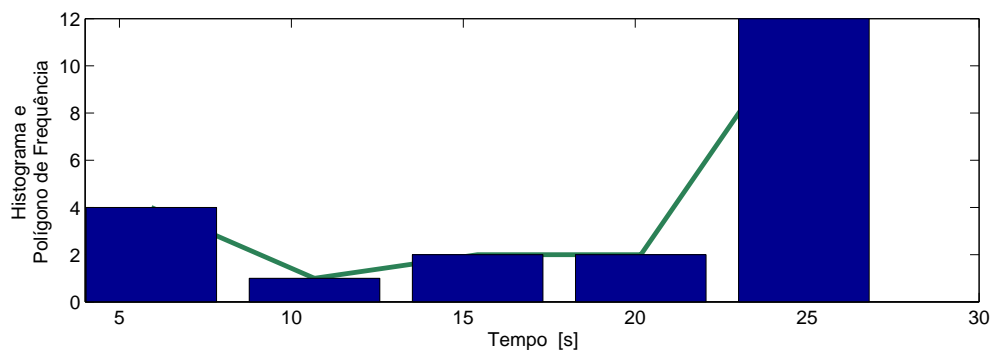


Figura 4.41: Histograma e Polígonos de frequências da probabilidade da vazão total gerada por CoDel ser maior que 1,45 Mbps em até 30 s.

Conclui-se assim mediante verificações feitas com a ferramenta SMC do UPPAAL que ENCN tem uma maior probabilidade de fornecer uma vazão melhor para os fluxos genéricos e um melhor ITAE para a NCS do que as técnicas RED, CoDel e PIE.

4.2.7 Comparação do desempenho do ENCN com protocolos baseados em ECN em uma NCS

O desempenho do ENCN em topologias de rede de comunicação compartilhadas por fluxos TCP genéricos e fluxos UDP de uma NCS também foi comparado com o desempenho obtido com a implementação de protocolos baseados em ECN nos fluxos genéricos na Figura 4.23.

Assim, as Figuras 4.42 e 4.43 ilustram a posição do motor em função do tempo quando nos fluxos genéricos foram implementados os protocolos TCP-Jersey e E-DCTCP, respectivamente. E a Figura 4.44 ilustra a evolução temporal do ITAE obtido com a implementação desses dois protocolos e o obtido com a implementação de ENCN.

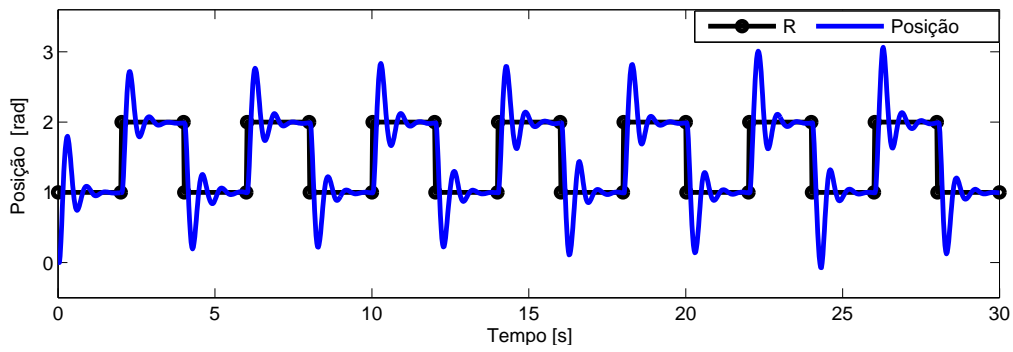


Figura 4.42: Posição do motor obtida com a implementação de TCP-Jersey.

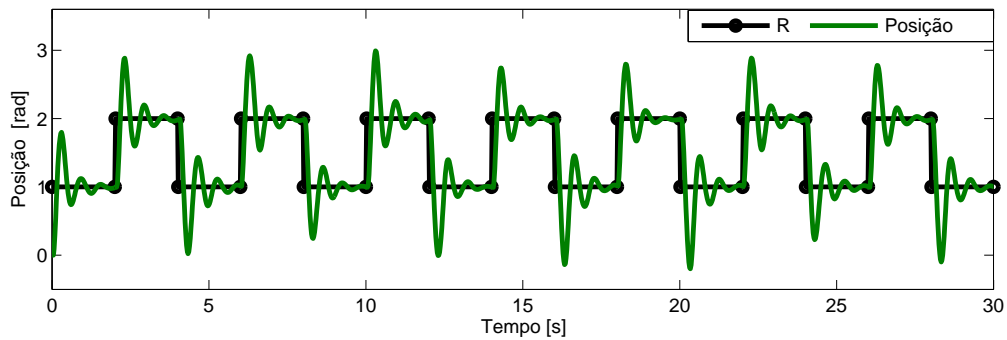


Figura 4.43: Posição do motor obtida com a implementação de E-DCTCP.

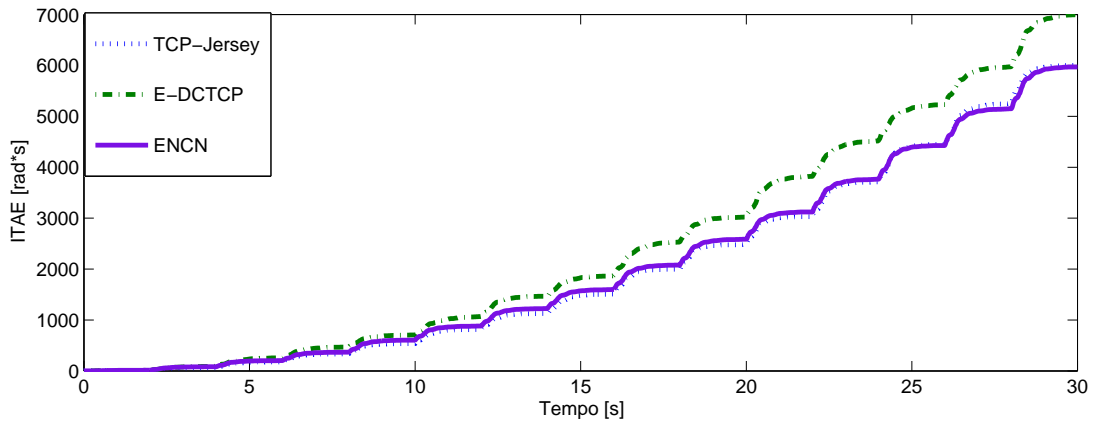


Figura 4.44: ITAE para ENCN comparado com protocolos baseados em ECN.

Conforme pode ser observado a implementação de E-DCTCP forneceu o pior desempenho para a NCS em termos de ITAE. Já com a implementação do TCP-Jersey se obteve um ITAE muito próximo ao obtido com a implementação de ENCN. Sendo o fornecido por ENCN ligeiramente melhor conforme pode ser observado mais nitidamente na Tabela 4.10.

Esse bom desempenho em termos de ITAE obtido com a implementação de TCP-Jersey se deve ao fato de que na rede ocorreram muitos períodos de *bufferempty* conforme pode ser observado na

Figura 4.46 (que ilustra a dinâmica da fila no gargalo com a implementação do TCP-Jersey). Isso foi bom para a NCS já que *bufferempty* reduz atrasos na rede de comunicação e menores atrasos geralmente melhoram a qualidade do sistema de controle. Porém, isso foi ruim para a vazão dos fluxos TCP-Jersey. Assim, TCP-Jersey forneceu a pior vazão entre os protocolos comparados conforme pode ser observado na Figura 4.45 (que ilustra a vazão total, soma da vazão dos fluxos A, B e C, com a implementação dos diferentes protocolos).

Portanto, diferentemente do ENCN o bom ITAE fornecido por TCP-Jersey teve um custo que se viu refletido na vazão dos fluxos genéricos.

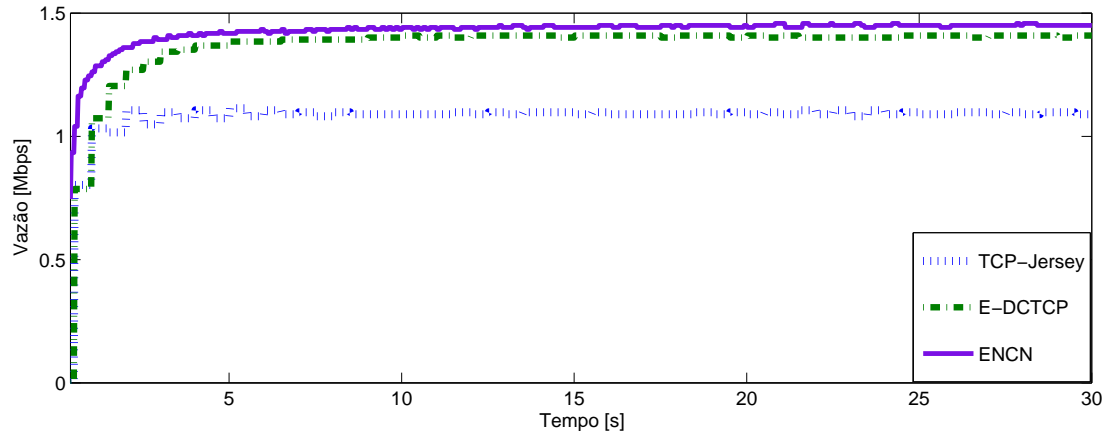


Figura 4.45: Vazão para ENCN comparado com protocolos baseados em ECN.

Já E-DCTCP forneceu uma vazão próxima à vazão obtida com a implementação de ENCN. Isso ocorreu porque conforme pode ser observado na Figura 4.47 (que ilustra a dinâmica da fila no gargalo com a implementação do E-DCTCP) com a implementação de E-DCTCP ocorreram menos períodos de *bufferempty* do que com a implementação de TCP-Jersey.

Porém, conforme pode ser visto na Figura 4.47, E-DCTCP apresentou maiores períodos de *bufferbloat* que ENCN, ou seja, maiores filas e maiores atrasos ponta a ponta. Assim, diferentemente do ENCN a boa vazão fornecida pelo E-DCTCP teve um custo que se viu refletido no ITAE da NCS que compartilha a mesma topologia de rede de comunicação.

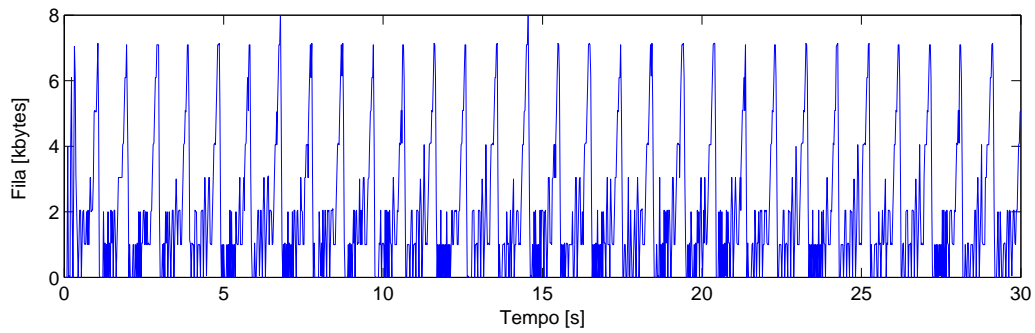


Figura 4.46: Fila para TCP-Jersey implementado na topologia de rede de comunicação ilustrada na Figura 4.23.

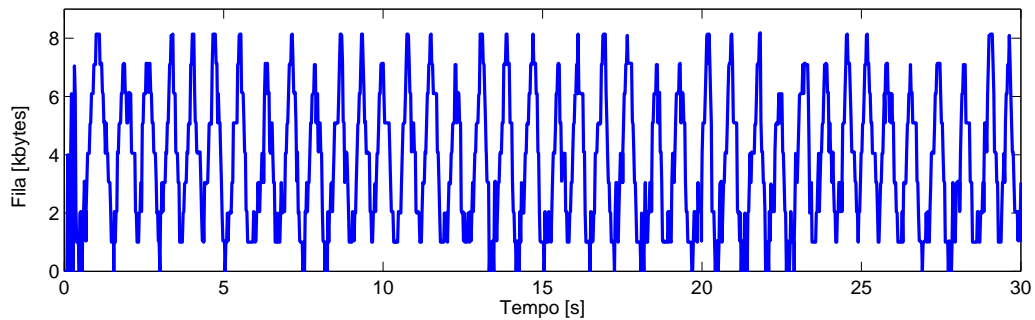


Figura 4.47: Fila para E-DCTCP implementado na topologia de rede de comunicação ilustrada na Figura 4.23.

O ENCN também forneceu melhor desempenho em termos de justiça para os três fluxos genéricos A, B e C conforme pode ser observado na Figura 4.48

Os valores finais, obtidos após 30 s de simulação para ITAE, vazão total, e Índice de Jain são sintetizados na Tabela 4.10. Na qual pode ser observado que o ENCN forneceu um ITAE para a NCS 0,52 % e 16, 68 % melhor que TCP-Jersey e E-DCTCP, respectivamente, e uma vazão para os fluxos TCP/IP genéricos 32 % e 2,9 % melhor que TCP-Jersey e E-DCTCP, respectivamente.

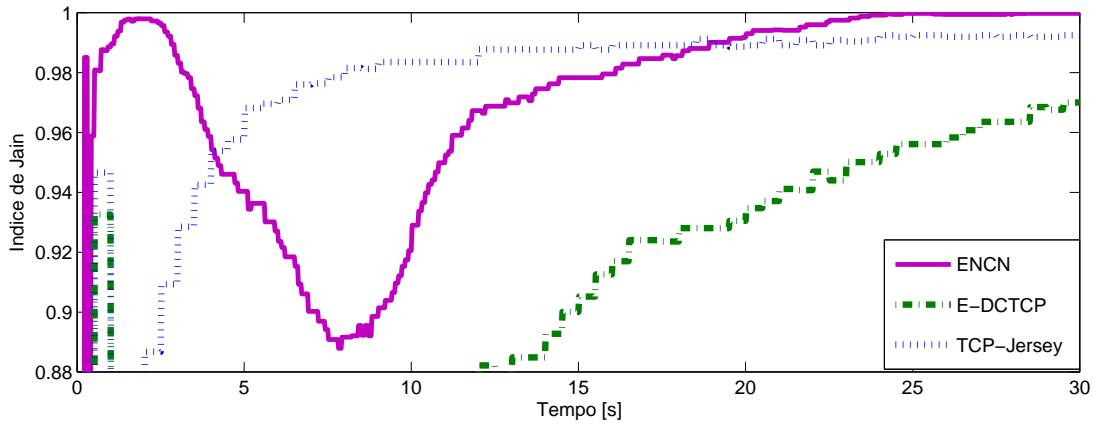


Figura 4.48: Índice de Jain para ENCN comparado com protocolos baseados em ECN.

Tabela 4.10: Vazão total, índice de Jain e ITAE para diferentes técnicas AQM atingidos ao final da simulação.

	Vazão total [Mbps]	Índice de Jain	ITAE [rad.s]
TCP-Jersey	1,097	0,991	6001
E-DCTCP	1,409	0,970	6997
ENCN	1,450	1	5970

4.3 ANCE (*Acknowledge-based Non-Congestion Estimation*)

Nessa seção será apresentado um novo protocolo de transporte de dados na Internet que foi desenvolvido neste trabalho denominado ANCE.

ANCE é um protocolo baseado em notificação implícita de não congestionamento. E visa fazer uma gestão de filas ponta a ponta, assim ANCE pode ser visto tanto como um protocolo de transporte de dados quanto como uma técnica AQM que trabalha nos transmissores ao invés de trabalhar nos roteadores.

Diferentemente de ENCN, o transmissor ANCE faz uma estimativa de não congestionamento ao invés de receber uma notificação explícita de não congestionamento dos roteadores presentes no caminho entre transmissor e receptor. Para essa finalidade ANCE estima o tamanho instantâneo da fila no gargalo ($qlen_E$) e faz uma gestão da fila ponta a ponta.

Uma vez estimado o tamanho na fila, ANCE funciona de forma análoga ao ENCN, e visa manter o tamanho da fila no roteador conectado ao canal de gargalo próximo a um limiar Q_{min} .

Assim, análogo ao ENCN, o transmissor ANCE modificará sua janela de transmissão seguindo um algoritmo semelhante ao TCP-Reno; porém, com as seguintes modificações:

- No caso em que o transmissor ANCE esteja na fase de partida lenta, e o tamanho de $qlen_E$ for maior que um limiar Q_{min} , o transmissor atualizará sua janela de transmissão (W) para

$W = \min(rwnd, Ssthr)$ e em seguida continuará em prevenção de congestionamento ($Ssthr$ deve ter um valor inicial finito previamente definido);

- Se $qlen_E$ for menor que o limiar Q_{min} , então a fase de partida lenta de ANCE funcionará como em TCP-Reno;
- Na fase de prevenção de congestionamento, o ANCE também implementa o mecanismo de balanço de justiça usado pelo ENCN. Assim, o transmissor ANCE incrementará sua janela de congestionamento $cwnd$ em um segmento por RTT se e somente $qlen_E$ for menor que Q_{min} . Se $qlen_E$ permanecer maior que Q_{min} o transmissor ANCE reduzirá sua janela de congestionamento em um pacote após a recepção de BF ACKs com o objetivo de manter a fila em torno de Q_{min} , visando assim evitar tanto o fenômeno de *bufferempty* quanto o fenômeno de *bufferbloat*.

BF é uma variável ajustada em $Ssthr$ pacotes cada vez que o transmissor ANCE entra na fase de prevenção de congestionamento, e durante essa fase segue a dinâmica dada na Tabela 4.1.

4.3.1 Estimação do tamanho fila

O método de estimação do tamanho da fila utilizado pelo protocolo ANCE é baseado no método de estimação do tamanho da fila utilizado pelo protocolo TCP-NCE [112].

Assim, cada vez que um ACK chega no transmissor ANCE o tamanho da fila ($qlen_E$) é estimado em função do RTT como sendo:

$$qlen_E = B.(RTT_{atual} - RTT_{min}), \quad (4.2)$$

onde B é a largura da banda no canal de gargalo, RTT_{atual} é o RTT e RTT_{min} é o mínimo RTT observado até o momento. A Equação (4.2) está baseada no fato de que o RTT_{min} ocorrerá quando não existir enfileiramento e a diferença entre o RTT_{atual} e o RTT_{min} corresponderá ao tempo de espera na fila que multiplicado pela banda do canal será equivalente ao tamanho da fila.

Porém, o atraso mínimo entre transmissor e receptor poderia aumentar por eventos não relacionados a filas nos roteadores, tais como, eventuais mudanças no caminho de roteamento ou incrementos na distância entre o transmissor e o receptor (o que é possível no caso de dispositivos móveis). Esses eventos, gerariam um incremento no RTT e o transmissor poderia interpretar que esse incremento fosse causado por filas nos roteadores, e logo, reduziria o valor de $cwnd$ de forma desnecessária. Gerando assim, o indesejado fenômeno de *bufferempty*.

Para evitar esse problema, sempre e quando W atinge seu valor mínimo de 1 segmento o valor do RTT_{min} é ajustado para o valor do RTT_{atual} . Além disso, o RTT não será atualizado na chegada de ACKs repetidos.

No TCP-NCE [112] a banda B do canal gargalo é considerada conhecida. Porém, na prática dificilmente essa banda poderá ser conhecida pelo transmissor. Assim, ANCE implementa também um algoritmo para estimar esse parâmetro.

Estimação da Banda no canal

Em ANCE não se considera conhecida a banda no canal de gargalo (B), sendo estimada ponta a ponta a partir da medição do tempo entre chegadas dos ACKs. Para fazer essa estimação, primeiramente no início da comunicação o transmissor assume que a banda no canal de gargalo é igual à banda no seu canal de saída. Depois, uma vez que os ACKs começam a chegar, o transmissor começa estimar a banda no canal de gargalo como sendo

$$B_E = Ack_l \cdot (tch_{min}), \quad (4.3)$$

onde Ack_l é o tamanho do ACK em bits, tch_{min} é o mínimo tempo entre chegadas em segundos (s) e, B_E é a banda estimada do canal de gargalo em bits/s. tch_{min} é atualizado toda vez que um ACK chega no transmissor como

$$tch_{min} \leftarrow \min(tch_{min}, (t_{atual} - t_{ant})), \quad (4.4)$$

onde t_{atual} é o instante de tempo em que o último ACK chegou no transmissor e t_{ant} é o instante de tempo em que chegou o ACK anterior.

Observe que o método de estimação da banda do canal utilizado por ANCE não deve ser confundido com os métodos de estimação de banda utilizados pelos protocolos TCP-Westwood e TCP-Jersey. Já que esses protocolos estimam uma banda disponível (que é variante no tempo e depende das condições de tráfego) para assim ajustar a janela de transmissão, enquanto que ANCE estima a banda do canal de gargalo (que normalmente é constante a menos que se mude o caminho) para assim poder estimar e controlar o tamanho da fila no roteador. Em outras palavras, TCP-Westwood e TCP-Jersey estimam uma fração da banda enquanto que ANCE estima a banda total.

Cada vez que um ACK chega no transmissor, se estima também o tamanho da fila $qlen_E$ em função do atraso ponta a ponta conforme a Equação (4.2), mas utilizando B_E ao invés de B (já que B normalmente é desconhecido pelo transmissor). Ou seja, o método de estimação do tamanho da fila usado por ANCE usa informações tanto do RTT quanto do tempo entre chegadas dos ACKs.

As Equações (4.3) e (4.4) se baseiam no fato que o tempo mínimo entre chegadas entre ACKs ocorrerá quando no canal de gargalo dois ACKs compartilhem posições subseqüentes na fila. Assim, imediatamente após o primeiro ACK ser atendido e enviado para frente, outro ACK com o mesmo destino vai para o atendimento, conseqüentemente, esses dois ACKs chegarão ao destino com um tempo entre chegadas igual que B/Ack_l s. Não poderá existir nenhum tempo entre chegadas menor que esse, já que esse é o mínimo tempo em que o roteador conectado ao canal de gargalo poderá inserir um ACK neste canal, e quando ocorrer esse evento o transmissor poderá conhecer a banda no canal de gargalo a partir da Equação (4.3).

Para ilustrar a técnica proposta foi estimada a fila e banda no canal de gargalo no cenário básico encontrado na Seção 4.1 de [116] e ilustrado na Figura 3.3. A dinâmica da fila no gargalo é a mesma que a ilustrada na Figura 3.10; porém, para capturar melhor o regime transitório as

simulações foram feitas por apenas 2 s.

A Figura 4.49 ilustra a estimação da banda no canal. Conforme pode ser observado inicialmente o transmissor assume que o gargalo é o canal de saída, ou seja 10 Mbps; em seguida, o primeiro pacote é enviado e logo chega o primeiro ACK, onde o tempo entre chegadas de ACKs é exageradamente grande, então o transmissor estima uma banda bem pequena. Depois, dois novos pacotes são enviados, quando um novo ACK chega o tempo entre chegadas é menor ao anterior, e uma banda maior é estimada (pouco antes de 0,2 s de simulação). Finalmente, próximo a 0,3 s de simulação é obtido o mínimo tempo possível entre chegadas de dois ACKs, e a banda é estimada como sendo 1,5 Mbps. A partir desse momento não ocorre nenhum tempo entre chegadas menor a esse, e então a estimação da banda é mantida em 1,5 Mbps (exatamente igual que a banda do gargalo) até o final da simulação.

O tamanho da fila no roteador e seu valor estimado é ilustrado na Figura 4.50 conforme pode ser observado até aproximadamente 0,3 s de simulação (antes de conseguir uma boa estimação da banda no canal de gargalo) o tamanho da fila estimado é muito diferente ao tamanho real. Já a partir do momento em que uma banda de 1,5 Mbps é estimada a estimação do tamanho da fila melhora significativamente. Algumas vezes o tamanho real da fila difere em um pacote do tamanho estimado, isso ocorre devido ao fato de que no modelo realizado a fila está sendo medida em pacotes (não em bits) e a fila somente reduz em um pacote quando todos os bits do pacote que está no atendimento são inseridos no canal. Assim sendo, no caso em que maior parte dos bits desse pacote já estejam no canal o atraso ponta a ponta será menor que no caso em que a maior parte dos bits estiverem ainda no roteador, logo essa diferença no RTT poderá afetar a estimação do tamanho da fila em um pacote.

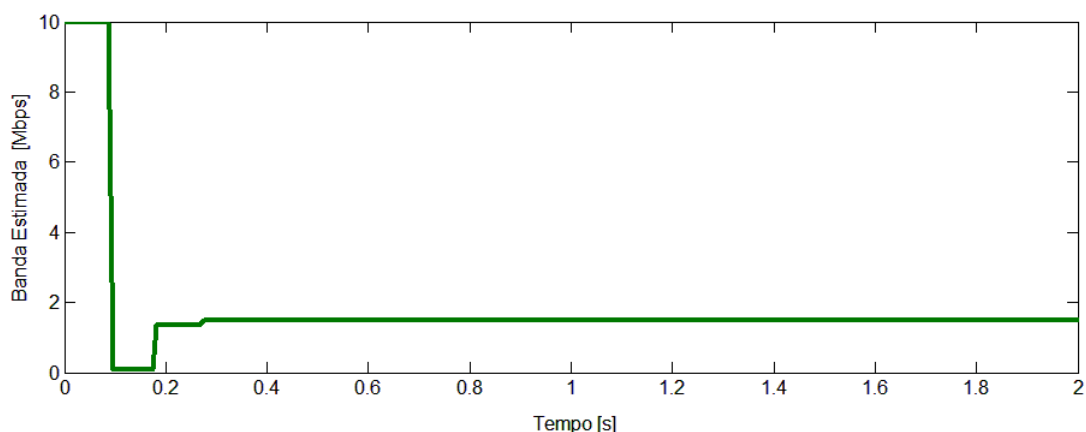


Figura 4.49: Banda Estimada utilizando a topologia de rede de comunicação da Figura 3.3.

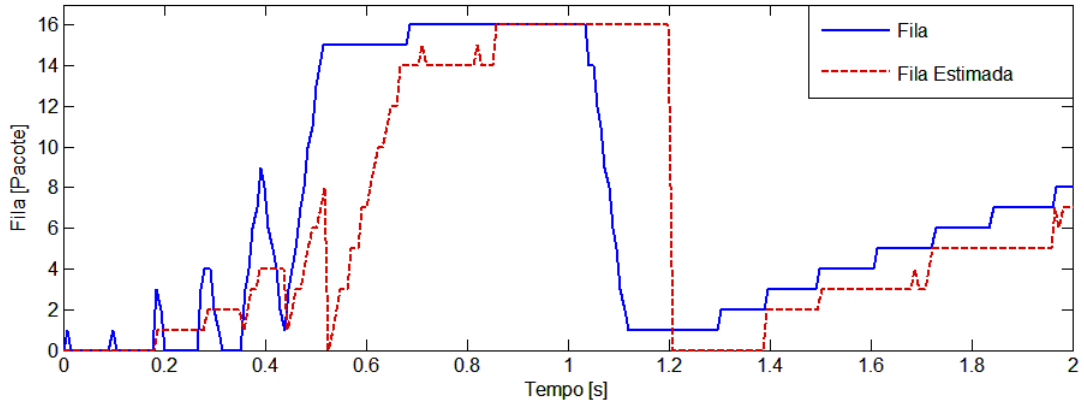


Figura 4.50: Fila e Fila Estimada utilizando a topologia de rede de comunicação da Figura 3.3.

4.3.2 Modelo do algoritmo ANCE no UPPAAL

Para modelar no UPPAAL a implementação da técnica AQM ANCE na topologia de rede de comunicação daisy-chain da Figura 3.3, ao modelo de autômatos temporizados apresentado no Anexo I foram feitas modificações na função `DeliverAck` do autômato da Figura I.15. Assim, essa função não atualizará o valor de W de acordo com o protocolo TCP-Reno conforme descrito na Tabela I.13, e sim de acordo com o protocolo ANCE descrito acima.

Para essa finalidade, na função `DeliverAck` também foram implementados os algoritmos de estimação de banda e fila no gargalo descritos na Subseção 4.3.1 e o mecanismo de balanço de justiça descrito na Tabela 4.1. E incrementos em $cwnd$ são realizados de acordo com o algoritmo ANCE e de acordo com esses incrementos é atualizada a variável Sd .

4.3.3 Avaliação do desempenho de ANCE em uma topologia de daisy-chain

O desempenho de ANCE foi comparado com o desempenho de outras técnicas AQM em termos de vazão, para a topologia de rede de comunicação daisy-chain dada na Figura 3.3. A Figura 4.51 ilustra a dinâmica da Janela do transmissor ANCE enquanto que a dinâmica da fila do caminho de ida no roteador 1 é ilustrada na Figura 4.52. Conforme pode ser observado, o transmissor ANCE começa a transmitir pacotes na fase de partida lenta com $W=1$ segmento, onde um segmento corresponde a um pacote de 1 kbyte, como só existe um único fluxo na rede a fila no roteador começa vazia. Logo, ao começar o tráfego, o transmissor estima que o tamanho da fila é menor que o limiar Q_{min} e entende que não há congestionamento entre ele e o receptor; então, o transmissor ajusta sua janela de transmissão para $W = \min(Ssthr, rwnd) = \min(30, 64) = 30$ segmentos. E sai da fase de partida lenta para entrar na fase de prevenção de congestionamento.

A partir desse momento, o transmissor ANCE faz ajustes em sua janela de transmissão visando manter o tamanho da fila em torno a Q_{min} . Reduzindo W em um segmento por cada BF ACKs recebido quando a fila estimada ($qlen_E$) for maior que Q_{min} e incrementando W em um segmento

por RTT quando $qlen_E$ for maior que Q_{min} . Essa dinâmica é mantida até o final do período de simulação, provocando oscilações de pequena amplitude tanto na janela do transmissor, quanto na fila do roteador.

A vazão em Mbps em função do tempo é ilustrada na Figura 4.53 e os valores finais para cada técnica AQM simulada são mostrados na Tabela 4.11.

Conforme pode ser observado, ANCE fornece uma vazão bem próxima a ENCN e ainda melhor que Drop Tail, RED, CoDel e PIE. Essa boa vazão se deve ao fato que como pode ser observado na Figura 4.52, ANCE apresenta menos períodos de *bufferempty* que RED, CoDel e PIE (Figuras 3.12, 3.14 e 3.16). Porém, ANCE apresentou maiores períodos de *bufferempty* que ENCN (Figura 4.52) e Drop Tail (Figura 3.10), e conseqüentemente uma vazão ligeiramente menor que ENCN; porém, maior que Drop Tail, já que Drop Tail apesar de não apresentar fenômeno de *bufferempty*, apresenta vários períodos nos quais ocorrem perdas de pacotes por transbordamento do *buffer* e conseqüentemente fornece uma vazão inferior a fornecida por ANCE.

Esse bom desempenho do ANCE é conseguido com menos recursos tecnológicos que nos casos de RED, CoDel, PIE e ENCN já que ANCE não atua descartando pacotes no roteador, e nem requer de uma notificação explícita de congestionamento ou não congestionamento advinda do roteador; e assim, pode funcionar em aplicações nas quais transmissor e/ou receptor e/ou roteadores não sejam capazes de trabalhar com ECN ou ENCN.

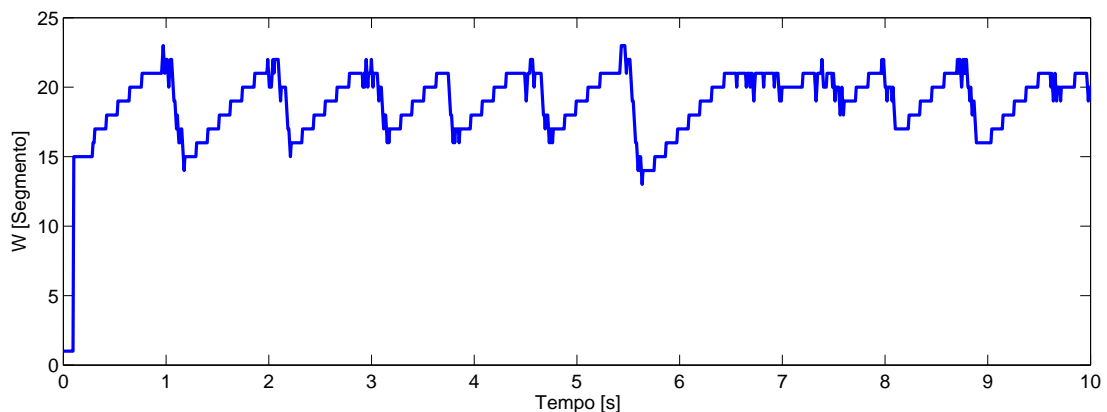


Figura 4.51: W para ANCE, implementada na topologia de rede de comunicação ilustrada na Figura 3.3.

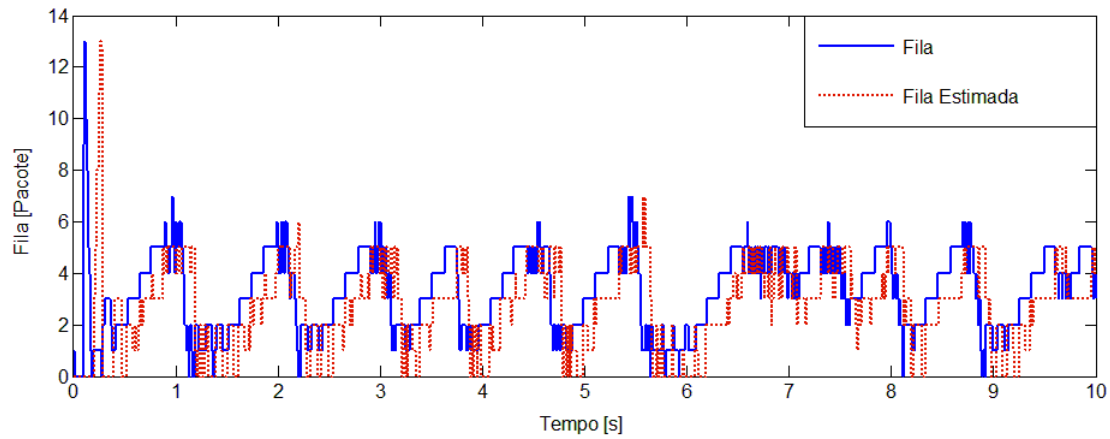


Figura 4.52: Fila e fila estimada para ANCE, implementada na topologia de rede de comunicação ilustrada na Figura 3.3.

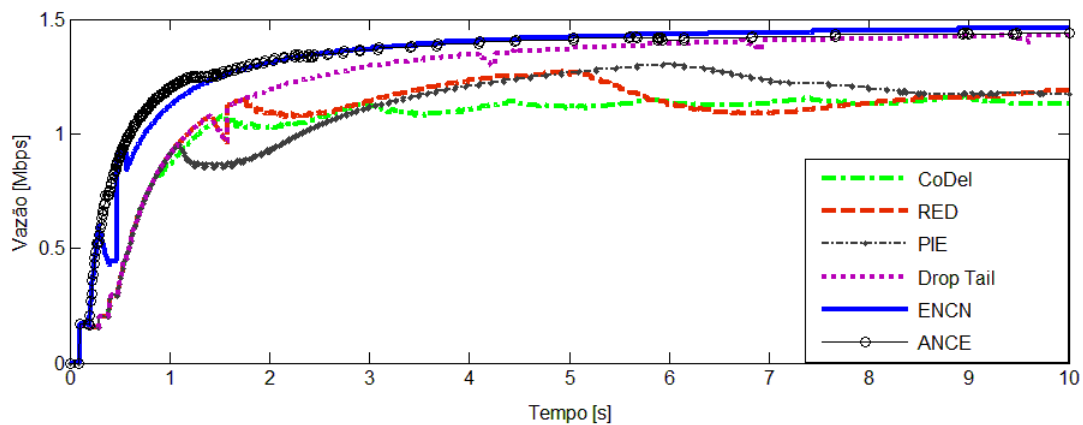


Figura 4.53: Vazão para ANCE comparada com outras técnicas AQM, implementadas na topologia de rede de comunicação daisy-chain da Figura 3.3.

Tabela 4.11: Tabela Vazão obtida aos 10 s de simulação para ANCE comparada com outras técnicas AQM, implementadas na topologia de rede de comunicação ilustrada na Figura 3.3.

Técnica AQM	Drop Tail	RED	CoDel	PIE	ENCN	ANCE
Vazão [Mbps]	1,430	1,187	1,133	1,172	1,460	1,442

4.3.4 Avaliação do desempenho de ANCE em uma NCS

Na presente seção será investigado o desempenho da técnica ANCE em uma NCS. Para essa finalidade foi utilizado a topologia de rede de comunicação ilustrada na Figura 4.23, sobre a qual foi reproduzida a mesma simulação feita na Subseção 4.2.6. Porém, com a implementação de ANCE, para assim comparar seu desempenho com as técnicas AQM (RED, CoDel, PIE e ENCN), em

termos de ITAE (para o fluxo UDP do sistema de controle em rede) e vazão e justiça para os três fluxos genéricos que compartilham a mesma topologia de rede de comunicação.

A posição do motor em função do tempo obtido com a implementação de ANCE nos fluxos genéricos é ilustrada na Figura 4.54, enquanto que a Figura 4.55 mostra a evolução temporal do ITAE obtido com a implementação de ANCE, ENCN, RED, CoDel e PIE.

Conforme pode ser observado, o melhor desempenho para o sistema de controle em rede foi fornecido pela técnica ENCN. Porém, ANCE ainda fornece um melhor desempenho para o sistema de controle que RED, CoDel e PIE; mesmo sem utilizar os recursos de ECN usados por ENCN, RED, CoDel e PIE. Esse bom desempenho de ANCE está no fato de que conforme pode ser observado na Figura 4.58 (dinâmica da fila no roteador 1 gerada pela implementação de ANCE), ANCE apresenta menos problemas de *bufferbloat* que RED, CoDel e PIE (cuja dinâmica da fila é ilustrada nas Figuras 4.31, 4.32, 4.33, respectivamente). Ou seja, ANCE mantém o tamanho da fila em valores menores que essas técnicas, conseqüentemente, fornece menores atrasos ponta a ponta.

A avaliação do desempenho dos fluxos genéricos quanto à vazão total é ilustrado na Figura 4.56, na qual a vazão total foi obtida como a somatória das vazões individuais de cada um dos três fluxos (A, B e C) que compartilham a mesma topologia de rede de comunicação. Conforme pode ser observado ANCE apresentou uma vazão total para os fluxos genéricos ligeiramente pior que ENCN e CoDel; porém, ligeiramente melhor que PIE e RED. Conforme abordado em seções anteriores, esse desempenho depende da capacidade da técnica de evitar o indesejado fenômeno de *bufferempty*. Na Figura 4.58 é fácil observar que ANCE contém maiores períodos de *bufferempty* que ENCN (cuja dinâmica da fila é ilustrada na Figuras 4.34), conseqüentemente fornece pior vazão.

Finalmente, a dinâmica temporal do índice de Jain em função do tempo para as diferentes técnicas AQM é mostrada na Figura 4.57. Conforme pode ser observado ANCE apresentou uma boa justiça entre os três fluxos genéricos, chegando a atingir o valor 1 (máximo valor possível) em vários períodos. Esse bom desempenho é causado pelo mecanismo de justiça implementado em ANCE e ENCN (veja a Tabela 4.1).

A Tabela 4.12 sintetiza os valores finais (valores atingidos aos 30 s de simulação) para vazão total, índice de Jain e ITAE para cada uma das técnicas AQM simuladas.

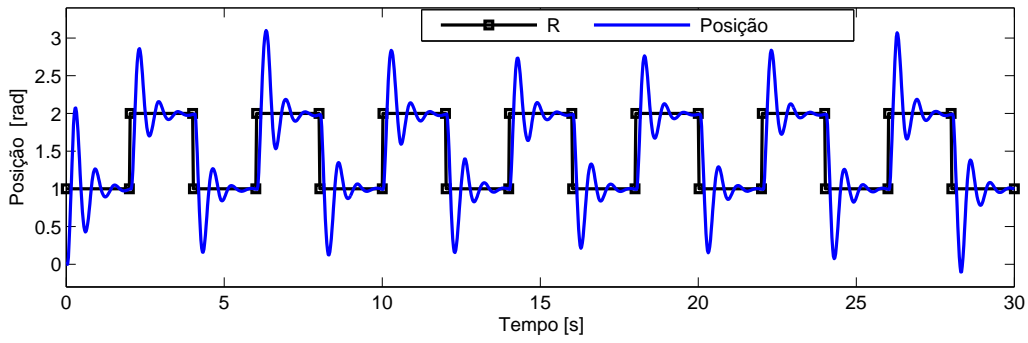


Figura 4.54: Posição do motor para ANCE implementada na topologia de rede de comunicação ilustrada na Figura 4.23.

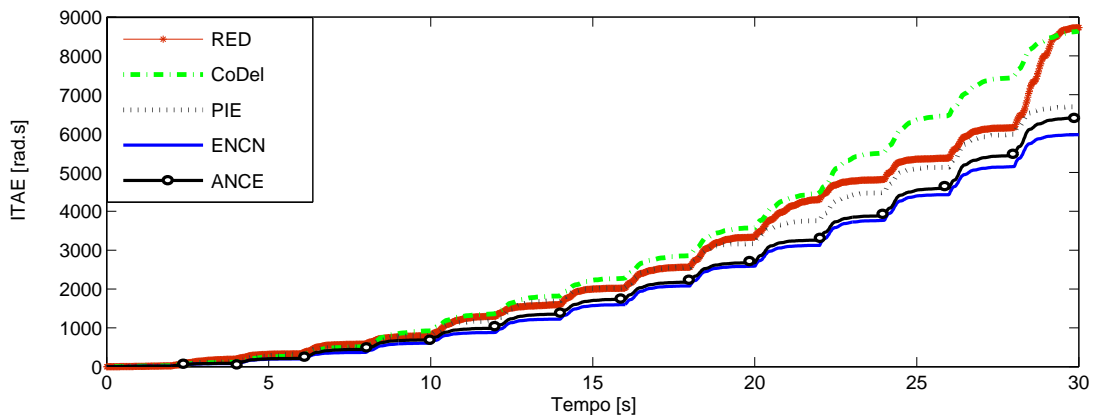


Figura 4.55: ITAE obtido com a implementação de diferentes técnicas AQM, implementadas na topologia de rede de comunicação ilustrada na Figura 4.23.

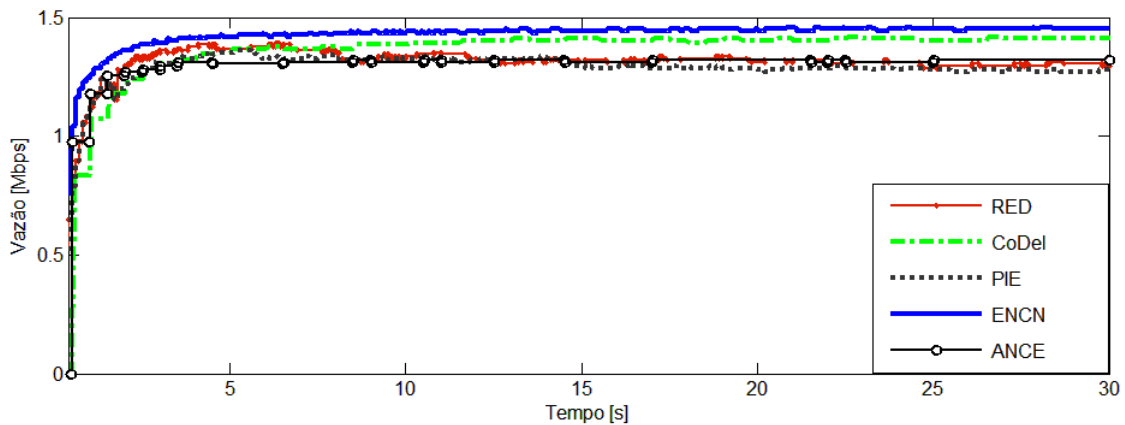


Figura 4.56: Vazão para ANCE comparada com técnicas AQM, implementadas na topologia de rede de comunicação ilustrada na Figura 4.23.

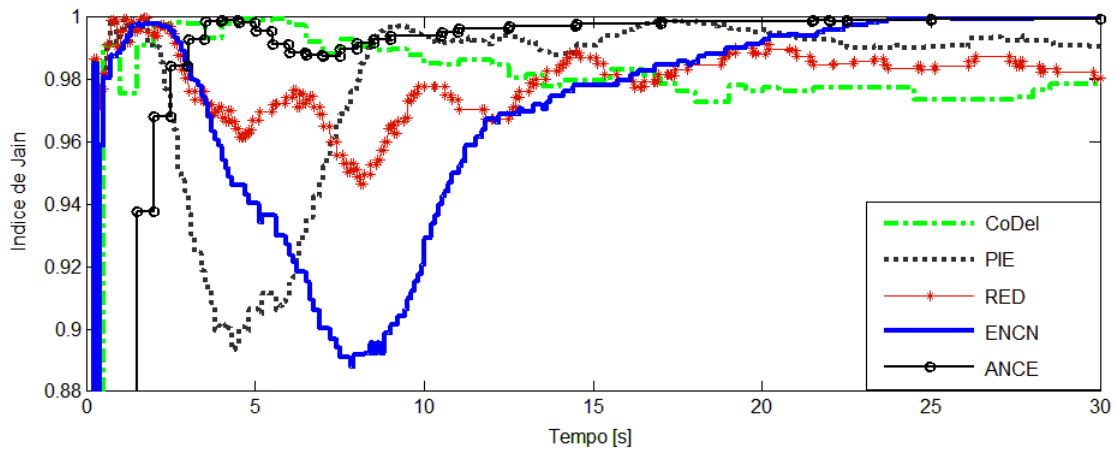


Figura 4.57: Índice de Jain para ANCE comparada com técnicas AQM, implementadas na topologia de rede de comunicação ilustrada na Figura 4.23.

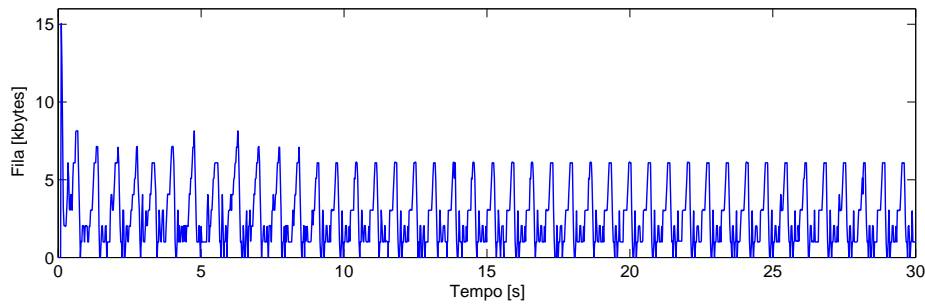


Figura 4.58: Fila para ANCE implementada na topologia de rede de comunicação ilustrada na Figura 4.23 .

Tabela 4.12: Vazão, índice de Jain e ITAE ao final da simulação para diferentes técnicas AQM implementadas na topologia de rede de comunicação ilustrada na Figura 4.23.

	Vazão total [Mbps]	Índice de Jain	ITAE [rad.s]
RED	1,294	0,981	8727
CoDel	1,417	0,978	8720
PIE	1,270	0,988	6691
ENCN	1,450	1	5970
ANCE	1,32	1	6401

4.3.5 Aplicação de ANCE em topologias com fluxos no caminho inverso

Com a finalidade de testar o desempenho de ANCE em topologias de rede de comunicação com congestionamento no caminho inverso foi utilizado a topologia de rede de comunicação ilustrada na Figura 4.59, na qual um transmissor ANCE transmite pacotes de 1 kbyte cada para um receptor ANCE. E no sentido contrário a esse fluxo ANCE um transmissor Poisson transmite aleatoriamente pacotes de 1kbyte com uma taxa $\lambda = 40$ pacotes/s, congestionando assim a fila no roteador 2 ligada

ao canal de gargalo (canal de 1,5 Mbps) (congestionamento que ocorre no sentido contrário ao fluxo ANCE).

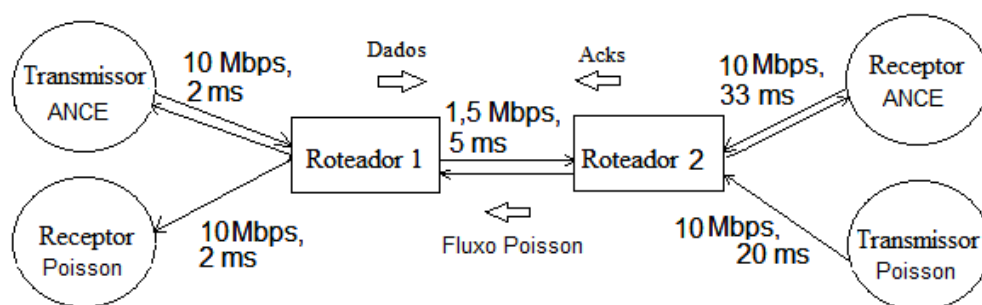


Figura 4.59: Topologia de rede de comunicação usada para avaliar o ANCE com fluxo Poisson no caminho inverso.

A Figura 4.60 ilustra a fila no roteador 1 (sentido do fluxo ANCE) e a correspondente fila estimada pelo transmissor ANCE. Como pode ser observado a fila estimada difere muito do tamanho real da fila, isso acontece porque ANCE estima o tamanho da fila em função do RTT (atraso de ida e volta); logo, a fila estimada pelo transmissor ANCE é a soma da fila no caminho de ida e a fila no caminho de retorno (sentido inverso ao fluxo ANCE) que ocorre no roteador 2 é ilustrada na Figura 4.61, onde é possível observar de forma mais nítida a razão desse erro de estimação do tamanho da fila cometido pelo transmissor ANCE.

Como consequência da diferença entre a fila estimada pelo transmissor ANCE e a fila real no sentido do fluxo ANCE, o transmissor ANCE reage a congestionamento no caminho inverso (congestionamento que não está sendo causado pelo fluxo ANCE); logo, a fila no sentido do fluxo ANCE permanece pequena ocorrendo vários períodos de *bufferempty*. Assim, o transmissor faz um uso ineficiente da banda do canal e conseqüentemente isso acaba afetando a vazão conforme pode ser observado na Figura 4.62 que ilustra a vazão com e sem um fluxo Poisson no sentido contrário ao fluxo ANCE.

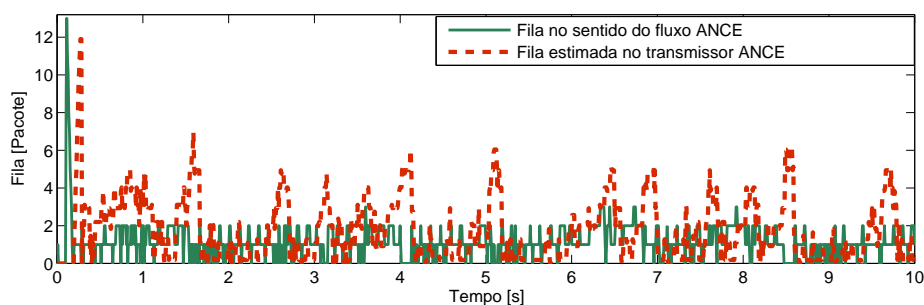


Figura 4.60: Fila no roteador 1 no sentido do fluxo ANCE e fila estimada pelo transmissor ANCE na presença de congestionamento no caminho inverso.

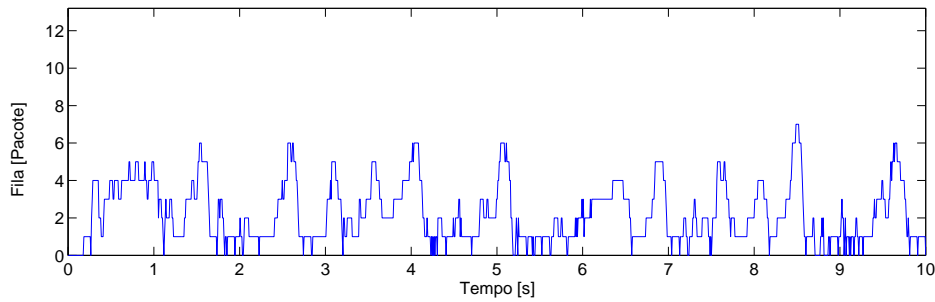


Figura 4.61: Fila no roteador 2 no sentido contrário ao fluxo ANCE.

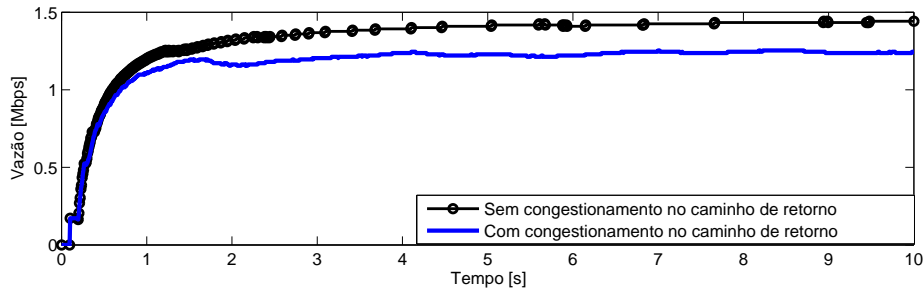


Figura 4.62: Vazão com e sem congestionamento no sentido contrário ao fluxo ANCE.

Para solucionar esse problema produzido pelo ANCE, no próximo capítulo será apresentado o TCP-Puerto-Londero um protocolo baseado em um atraso relativo no caminho de ida entre transmissor e receptor ao invés de usar o RTT (atraso de ida e volta).

4.4 Conclusões

Neste capítulo foram apresentados dois novos protocolos desenvolvidos neste trabalho que combinam transporte de dados na Internet com gestão ativa de filas nos roteadores.

O primeiro protocolo apresentado foi o ENCN (*Explicit Non-Congestion Notification*) o qual, com o objetivo de evitar o indesejado fenômeno de *bufferempty* e assim melhorar a vazão, utiliza alguns recursos de ECN para notificar não congestionamento no caminho entre transmissor e receptor ao invés de notificar congestionamento nos roteadores (como fazem outras técnicas AQM encontradas na bibliografia). Além disso, ENCN além de atuar marcando pacotes na fila dos roteadores também atua nas janelas dos transmissores de acordo com as recepções de ACKs marcados, e ainda, implementa um mecanismo para melhorar a justiça quando mais de um fluxo compartilham uma mesma topologia de rede de comunicação.

Por outro lado, análogo ao CoDel e PIE, ENCN também evita o indesejado fenômeno de *bufferbloat*, e assim resulta em menores atrasos ponta a ponta e conseqüentemente fornece bom desempenho para sistemas de controle através da Internet.

Resultados de simulações e verificações em UPPAAL mostraram que em topologias de rede de

comunicação tipo *Dumbbell* compartilhadas por fluxos genéricos e um fluxo UDP de uma NCS, ENCN fornece melhor vazão e justiça para fluxos TCP genéricos, e melhor ITAE para a NCS, comparado com o protocolo TCP-Reno trabalhando em conjunto com as técnicas AQM Drop Tail, RED, CoDel e PIE e com protocolos baseados em ECN (TCP-Jersey e E-DCTCP).

Por outro lado, neste capítulo foi proposto uma inovadora forma de fazer gestão ativa de filas, mediante estimação do tamanho da fila e banda no canal de gargalo, para assim, implementar técnicas AQM nos transmissores ao invés de implementá-las nos roteadores.

Utilizando essa metodologia de implementar algoritmos AQM nos transmissores, foi proposta uma técnica denominada ANCE (*Acknowledge-based Non-Congestion Estimation*) a qual basicamente consiste em fazer uma estimação da fila no roteador e reproduzir os efeitos da técnica ENCN nos transmissores TCP, sem necessitar utilizar um bit de notificação de congestionamento.

Resultados de simulações mostraram que, apesar de utilizar menos recursos, ANCE consegue fazer uma boa gestão de filas e atinge um desempenho próximo às técnicas que utilizam recursos de ECN para notificar o estado da fila no roteador, e ainda, alguns parâmetros de desempenho de ANCE resultaram até melhores que os obtidos com algumas das outras técnicas AQM simuladas. Porém ANCE apresentou pobre desempenho em termos de vazão quando ocorre congestionamento no sentido contrário a seu fluxo. Esse problema ocorre porque ANCE estima o tamanho da fila com base no RTT e assim a fila estimada é a soma da fila no caminho de ida e a fila no caminho de retorno.

Visando solucionar essas limitações do ANCE, no próximo capítulo será apresentado o TCP-Puerto-Londero um protocolo que ajusta a janela de transmissão mediante um sistema de controle adaptativo que calcula de forma dinâmica a função de transferência da rede de comunicação, tendo como entrada um atraso relativo no caminho de ida entre transmissor e receptor, e como saída o valor da janela de transmissão *cwnd*. Assim, diferentemente de ANCE, TCP-Puerto-Londero ajusta sua janela de transmissão em função de um atraso relativo no caminho de ida ao invés de usar o RTT (atraso de ida e volta).

Capítulo 6

Conclusões

Com o objetivo de estudar sistemas de controle através da Internet junto com outros fluxos genéricos mediante simulações e verificações probabilísticas que considerem melhores modelos para atrasos e perdas de pacotes do que aqueles encontrados na bibliografia, assim como, poder estudar as causas que provocam esses inconvenientes, o presente trabalho apresenta uma inovadora metodologia de modelagem de redes de comunicação mediante um conjunto de autômatos temporizados através de uma linguagem baseada em TCTL (*Timed Computation Tree Logic*), facilitando assim realizar uma verificação formal de todo o sistema em estudo.

Essa modelagem foi feita na ferramenta de software UPPAAL, no qual cada dispositivo da rede de comunicação (canais, roteadores, transmissores, receptores) foi modelado com um ou mais autômatos temporizados, ou seja, mediante máquinas de estados finitos estendidas com relógios e variáveis inteiras e flutuantes, e o sistema completo (toda a topologia da rede de comunicação) funciona como um conjunto de tais autômatos temporizados em paralelo.

O modelo desenvolvido ainda possibilita fazer simulações para comparar o desempenho de diferentes protocolos e técnicas de gestão de filas utilizadas em redes TCP/IP, assim como fazer verificações estatísticas mediante a ferramenta SMC (*Statistical Model Checking*) do UPPAAL que possibilita checar se determinadas variáveis podem atingir determinados valores e no caso afirmativo com que probabilidade isso pode acontecer.

O modelo resultante, facilita a observação dos efeitos das características da rede de comunicação (tais como tempo de propagação, banda dos canais, confiabilidade do meio, interferências externas como ruídos que provocam distorções de sinais ou obstáculos físicos que atenuem os sinais em canais sem fios provocando perdas de pacotes) no desempenho dos fluxos. Também facilita estudar as interferências mútuas causadas entre os diferentes tipos de fluxos que compartilham a mesma topologia de rede, por exemplo, as interferências causadas por um fluxo TCP em um fluxo UDP de uma NCS (*Networked Control Systems*) que compartilha a mesma rede de comunicação.

Esse trabalho ainda apresenta uma aplicação na qual um motor DC é controlado através de uma rede IP por um controlador PI (*proportional integral*). A troca de dados entre planta e controlador foi estudada em diferentes topologias de rede, com diferentes condições de tráfego, e com a aplicação de diferentes técnicas AQM nos roteadores. Nestes cenários o desempenho do

sistema de controle foi medido empregando o ITAE.

O sistema a ser controlado foi escolhido intencionalmente simples, já que o objetivo do trabalho não foi desenvolver um novo algoritmo de controle, mas sim estudar as interferências dos parâmetros do sistema de comunicação e de outros fluxos presentes na rede no desempenho de algoritmos de controle tradicionalmente utilizados, para assim propor novas soluções mediante o emprego de algoritmos de gestão de filas e protocolos de transporte de dados capazes de reduzir atrasos e perdas de pacotes e, conseqüentemente fornecer um bom desempenho para sistemas de controle através da Internet sem sacrificar a vazão de outros fluxos genéricos que compartilham o mesmo meio de comunicação.

Por outro lado, também foram modelados no UPPAAL alguns protocolos de transporte de dados baseados em ECN (*Explicit Congestion Notification*) (TCP-Jersey, DCTCP e E-DCTCP) e as técnicas AQM Drop Tail, RED, CoDel e PIE, e foram estudados os desempenhos de cada uma dessas metodologias em fluxos TCP (considerando a vazão e justiça) e em um fluxo UDP de uma NCS que compartilha a mesma topologia de rede de comunicação considerando o ITAE.

Esses estudos foram feitos mediante simulações e verificações probabilísticas feitas no UPPAAL, em cujos resultados foi observado a existência de uma compensação entre o desempenho dos fluxos genéricos e o desempenho de uma NCS que utiliza a Internet. Assim, técnicas AQM e protocolos que fornecem boa vazão para os fluxos genéricos, geralmente geram filas maiores nos roteadores e conseqüentemente maiores atrasos ponta a ponta e assim um pobre desempenho para o sistema de controle que compartilha a mesma topologia de rede de comunicação. Enquanto que técnicas AQM e protocolos que proporcionam bom desempenho para o sistema de controle geralmente fornecem pobre vazão para os fluxos genéricos.

Nestas simulações e verificações também foi possível observar que o bom o desempenho para a NCS geralmente é alcançado por protocolos e/ou técnicas AQM que visam manter a fila nos roteadores em tamanhos pequenos a fim de evitar o indesejado fenômeno de *bufferbloat* e conseqüentemente reduzem atrasos. Porém, no intuito de evitar *bufferbloat* a fila no gargalo pode permanecer completamente vazia por alguns períodos de tempo e esse é o efeito que prejudica a vazão para os fluxos genéricos. Esse fenômeno foi identificado neste trabalho e chamado de *bufferempty* e é o fenômeno oposto ao fenômeno de *bufferbloat*.

Logo, visando obter uma técnica de gestão de fila que consiga evitar o indesejado fenômeno de *bufferbloat* sem produzir *bufferempty*, e conseqüentemente permita obter uma melhor compensação entre o desempenho de uma NCS que utiliza a Internet e fluxos TCP genéricos que compartilham a mesma topologia de rede de comunicação, foi proposta uma metodologia denominada ENCN (*Explicit Non-Congestion Notification*), a qual, diferentemente de outras técnicas AQM encontradas na bibliografia, utiliza alguns recursos de ECN para notificar não congestionamento no caminho entre transmissor e receptor ao invés de notificar congestionamento nos roteadores. Além de atuar nos roteadores ENCN também atua nas janelas dos transmissores de acordo com as recepções de Acks marcados, e ainda, implementa um mecanismo para melhorar a justiça quando mais de um fluxo compartilham uma mesma topologia de rede.

Resultados de simulações e verificações estatísticas mostraram que nas topologias de rede de

comunicação simuladas, ENCN evita tanto o problema de *bufferempty* quanto o problema de *bufferbloat* e conseqüentemente comparado com as técnicas AQM Drop Tail, RED, CoDel e PIE, e com alguns protocolos baseados em ECN, (TCP-Jersey, DCTCP e E-DCTCP), ENCN melhora a compensação entre o desempenho de fluxos genéricos e o fluxo de uma NCS que utiliza a Internet, fornecendo assim melhor vazão e justiça para os fluxos genéricos e melhor ITAE para a NCS.

Porém, ENCN utiliza alguns recursos de ECN para fazer explícita notificação de não congestionamento, esses recursos nem sempre estão disponíveis na prática, logo, o uso prático de ENCN ao igual que o uso de protocolos baseados em ECN fica limitado. Para superar essas limitações tecnológicas neste trabalho também foi proposta uma inovadora metodologia de gestão de filas mediante implícita notificação de congestionamento realizada ponta a ponta com base em uma estimativa do tamanho da fila e da banda no canal de gargalo, para assim, atuar somente nos transmissores ao invés de atuar também nos roteadores.

Utilizando essa metodologia, foi proposta uma técnica denominada ANCE (*Acknowledge-based Non-Congestion Estimation*) a qual basicamente consiste de um protocolo em que o transmissor faz uma estimativa da fila no gargalo e logo tenta controlar o tamanho da fila mediante atuações em sua janela de transmissão, sem necessitar utilizar bits de notificação de congestionamento ou de não congestionamento.

Resultados de simulações mostraram que, apesar de utilizar menos recursos, ANCE consegue fazer uma boa gestão de filas e consegue um desempenho próximo ao desempenho fornecido por metodologias que utilizam ECN para notificar o estado da fila no roteador, e ainda, alguns parâmetros de desempenho de ANCE resultaram até melhores que os obtidos com algumas das outras técnicas AQM simuladas.

Porém, ANCE estima o tamanho da fila no gargalo com base no RTT. Mas, o RTT varia tanto em função do congestionamento no caminho de ida (sentido transmissor/receptor) quando em função do congestionamento no caminho de retorno (sentido receptor/transmissor), logo, a fila estimada por ANCE resulta ser a soma das filas em ambos sentidos. Conseqüentemente ANCE pode reagir a um congestionamento que ocorre no sentido contrário a seu fluxo e cujo controle está fora de seu alcance. Isso pode causar o indesejado fenômeno de *bufferempty* no sentido de seu fluxo e assim prejudicar a vazão.

Para superar essas limitações do ANCE foi proposto o TCP-Puerto-Londero, o qual análogo ao ANCE visa controlar o tamanho da fila mediante uma abordagem ponta a ponta, porém, com base em um atraso relativo existente apenas no caminho de ida (sentido transmissor/receptor). Assim, TCP-Puerto-Londero não reage a congestionamentos no sentido inverso a seu fluxo e conseqüentemente comparado com ANCE o TCP-Puerto-Londero consegue fazer um melhor aproveitamento da banda disponível, evitando *bufferempty* no sentido transmissor/receptor e conseguindo assim melhor desempenho em termos de vazão nas situações de congestionamento no sentido oposto.

Com a finalidade de controlar o tamanho da fila no sentido de seu fluxo evitando tanto o fenômeno de *bufferbloat* quanto o fenômeno de *bufferempty* cada transmissor TCP-Puerto-Londero utiliza um controlador adaptativo com banda morta, cuja lei de controle é ajustada dinamicamente com base na função de transferência da rede de comunicação existente entre transmissor e receptor

que, por sua vez, é estimada mediante um algoritmo de mínimos quadrados recursivos com fator de esquecimento.

No TCP-Puerto-Londero a função de transferência da rede de comunicação existente entre transmissor e receptor tem como entrada o atraso relativo no caminho de ida (sentido transmissor/receptor) e como saída um valor para a janela de congestionamento. E a lei de controle visa manter o atraso no caminho de ida próximo a um atraso de referência que é também dinamicamente ajustado.

Diferentemente dos protocolos baseados em ECN ou em ENCN, tais como TCP-Jersey, DCTCP e E-DCTCP e ENCN, o TCP-Puerto-Londero não necessita de recursos tecnológicos para fazer explícita notificação de congestionamento ou não congestionamento, já que esses eventos são detectados de forma implícita com base em um atraso relativo existente entre transmissor e receptor.

O TCP-Puerto-Londero foi modelado no UPPAAL e seu desempenho foi comparado mediante simulações e verificações com o desempenho dos protocolos TCP-Jersey, E-DCTCP e ENCN. Resultados mostram que o TCP-Puerto-Londero fornece melhor desempenho que o TCP-Jersey e que o E-DCTCP para a vazão e índice de Jain. E dado que os transmissores TCP-Puerto-Londero evitam longas filas nos *buffers* dos roteadores, e conseqüentemente, introduzem pouco atraso no sistema de comunicação, acabam colaborando indiretamente para um bom desempenho de uma NCS com fluxos UDP que compartilham a mesma topologia de rede de comunicação. Fornecendo também melhor ITAE que o TCP-Jersey e que o E-DCTCP.

TCP-Puerto-Londero ainda fornece um desempenho muito próximo ao desempenho fornecido por ENCN, com a vantagem de não necessitar recursos tecnológicos para fazer explícita notificação de não congestionamento.

6.1 Trabalhos futuros

Futuramente os conceitos de ENCN poderão ser aplicados também em fluxos UDP de NCSs que usam a Internet. A informação de não congestionamento poderá ser utilizada por um controlador para fazer eventuais chaveamentos entre diferentes leis de controle utilizando controladores mais rápidos quando uma explícita notificação de não congestionamento for recebida e controladores mais lentos na ausência dessas notificações.

Uma explícita notificação de não congestionamento também poderá ser útil para utilizar metodologias de *co-design*, projetando controladores baseados em agendamento de eventos que priorizem fazer transmissões quando for recebida uma explícita notificação de não congestionamento já que nesses instantes o tráfego na rede é reduzido.

Também poderão ser projetados controladores com período de amostragem variantes utilizando menores períodos de amostragem quando um não congestionamento for detetado.

Por outro lado, ENCN também poderá ser utilizado para auxiliar algoritmos de roteamento de pacotes implementado nos roteadores, já que ENCN permite identificar caminhos não congestionados.

Finalmente, também poderá ser estudado o desempenho de ENCN, ANCE e TCP-Puerto-Londero em redes sem fio e fazer eventuais adaptações dessas metodologias para atender características típicas desses meios, como por exemplo, diante de perdas de pacotes, estimar se a mesma foi provocada por congestionamento nos roteadores ou por perdas de conexões.

REFERÊNCIAS BIBLIOGRÁFICAS

- [1] Z. Taferra. Process control over wireless sensor networks. Master's thesis, Kungliga Tekniska Hogskolan (KTH), Stockholm, Sweden, 2013.
- [2] L. M. A. Sup, R. M. Moraes, and A. Bauchspiess. Simultaneous TCP and NCS flows in a UPPAAL framework with a new AQM technique. In *Proc. of IEEE International Conference on Industrial Informatics (INDIN)*, Poitiers, France, July 2016.
- [3] S. M. T. Ahsan, Z. Hassan, T. Imam, K. Faysal, A. Sadat, R. Hazari, and L. N. Manzoor. Design and implementation of a robotic vehicle with real-time video feedback control via internet. In *Proc. of IEEE International Conference on Electrical Engineering and Information and Communication Technology ICEEICT*, Dhaka Bangladesh, April 2014.
- [4] H. Ohta and M. Kawashima. Technical feasibility of patient-friendly screening and treatment of digestive disease by remote control robotic capsule endoscopes via the internet. In *Proc. of the 36th IEEE Annual International Conference of the Engineering in Medicine and Biology Society (EMBC)*, Chicago, IL, USA, Aug. 2014.
- [5] G. M. Bilgerig, J. R. Martínez, S. A. Salinas, V. Z. Pérez, J. J. Padilla, and M. J. Betancur. Teleoperated robotic system with application in laparoscopic training: Peg transfer test. *IEEE Latin American Transactions*, 14(7):3178 – 3184, 2016.
- [6] H. Mortaji, O. S. Hock, M. Moghavvemi, and A. F. Haider. Smart grid demand response management using internet of things for load shedding and smart-direct load control. In *Proc. of the IEEE International Conference on Industry Applications Society Annual Meeting*, Portland, OR USA, Oct 2016.
- [7] M. RANA. Architecture of the internet of energy network: An application to smart grid communications. *IEEE ACCESS Journal Special Section: The Internet of Energy: Architectures, Cyber Security, and Applications*, 5(1):4704 – 4710, 2017.
- [8] YW. Bai, HW. Su, and WC Hsu. Indoor and remote controls and management of home appliances by a smartphone with a four-quadrant user interface. In *Proc. of the IEEE International Conference on Consumer Electronics (ICCE)*, Las Vegas, NV, USA., Jan 2017.
- [9] Y. Halevi and A. Ray. Integrated communication and control systems: Part i - analysis. *ASME journal of dynamic systems, measurement and control*, 110(1):367 – 373, 1988.

- [10] A. Ray and Y. Halevi. Integrated communication and control systems: Part ii - design considerations. *ASME journal of dynamic systems, measurement and control*, 110(1):374 – 381, 1988.
- [11] A. Ray. Distributed data communication networks for real-time process control. *Chemical engineering communication*, 65(1):139 – 154, 1988.
- [12] A. Ray. Introduction to networking for integrated control systems. *IEEE control system magazine*, 9(1):76 – 79, 1989.
- [13] Y. Halevi and A. Ray. Performance analysis of integrated communication and control systems networks. *ASME journal of dynamic systems, measurement and control*, 112(1):365 – 370, 1990.
- [14] L. W. Liou and A. Ray. Integrated communication and control systems : Part iii - nonidentical sensor and controller sampling. *ASME journal of dynamic systems, measurement and control*, 112(1):357 – 364, 1990.
- [15] L. W. Liou and A. Ray. On modelling of integrated communication and control systems. *ASME journal of dynamic systems, measurement and control*, 112(1):790 – 794, 1990.
- [16] R. Luck and A. Ray. An observer-based compensator for distributed delays. *Automatica*, 26(5):903 –908, 1990.
- [17] L. W. Liou and A. Ray. A stochastic regulator for integrated communication and control systems : Part i -"formulation of control law. *ASME journal of dynamic systems*, 113(1):604 –611, 1991.
- [18] L. W. Liou and A. Ray. A stochastic regulator for integrated communication and control systems: Part ii -"numerical analysis and simulation. *ASME journal of dynamic systems, measurement and control*, 113(1):612 –619, 1991.
- [19] R. Luck and A. Ray. Experimental verification of a delay compensation algorithm for integrated communication and control systems. *International journal of control*, 59(6):1357–1372, 1994.
- [20] A. Ray. Optput feedback control under randomly varying distributed delays. *AIAA journal of guidance, control and dynamics*, 17(4):701 –711, 1994.
- [21] J. Nilsson, B. Wittenmark, M. Tornngren, and M. Sanfridson. Timing problems in real-time control systems. In *Proc. of 14th american control conference*, Seattle USA, Jun 1995.
- [22] J. Nilsson and B. Bernhardsson. LQG control over a markov communication network. In *Proc. of 36th IEEE conference on decision and control*, San Diego USA, Jun 1997.
- [23] J. Nilsson, B. Bernhardsson, and B. Wittenmark. Some topics in real-time control. In *Proc. of the 17th IEEE american control conference*, Philadelphia USA, Jun 1998.

- [24] J. Nilsson, B. Bernhardsson, and B. Wittenmark. Stochastic analysis and control of real-time systems with random time delays. *Automatica*, 34(1):57–64, 1998.
- [25] J. Nilsson. Real-time control systems with delays, 1998.
- [26] M. Torngren. Fundamentals of implementing real-time control applications in distributed computer systems. *Real-time systems*, 14(1):219–250, 1998.
- [27] B. Wittenmark, Bastian B., and J. Nilsson. Analysis of time delays in synchronous and asynchronous control loops. In *Proc. of the 37th IEEE conference on decision and control*, Tampa, Florida USA, Dec 1998.
- [28] K. J. Astrom and B. Wittenmark. *Computer Controlled Systems: Theory and Design*. Prentice-Hall, Englewood Cliffs, NJ, 3 edition, 1997.
- [29] M. S. Branicky, S. M. Phillips, and W. Zhang. Stability of networked control systems: Explicit analysis of delay. In *Proc. of the 19th IEEE american control conference*, Chicago, IL, USA, June 2000.
- [30] W. Zhang and M. S. Branicky. Stability of networked control systems with time-varying transmission period. In *Proc. of the 39th Allerton conference on communication, control, and computing*, Urbana, IL, USA, Oct. 2001.
- [31] W. Zhang, M. S. Brannicky, and S. M. Philips. Stability of networked control systems. *IEEE Control Systems Magazine*, 21(1):84–99, 2001.
- [32] M. S. Branicky, S. M. Phillips, and W. Zhang. Scheduling and feedback co-design for networked control systems. In *Proc. of the 41th IEEE conference on decision and control*, Las Vegas, NV, USA, Dec. 2002.
- [33] M. S. Branicky, V. Liberatore, and S. M. Phillips. Networked control system co-simulation for co-design. In *Proc. of the 22nd IEEE american control conference*, Denver, CO, USA, June 2003.
- [34] P. H. Bauer, M. Sicitui, and K. Premaratne. Closing the loop through communication networks: The case of an integrator plant and multiple controllers. In *Proc. of the 38th IEEE conference on decision and control*, Phoenix, AZ, USA, Dec. 1999.
- [35] P. H. Bauer, M. Sicitui, and K. Premaratne. Controlling an integrator through data networks: Stability in the presence of unknown time-variant delays. In *Proc. of the IEEE International symposium on circuits and systems*, Orlando, FL, USA, June 1999.
- [36] P. H. Bauer, M. Sicitui, C. Lorand, and K. Premaratne. Total delay compensation in lan control systems and implications for scheduling. In *Proc. of the 20th IEEE american control conference*, Arlington, VA, USA, June 2001.
- [37] P. H. Bauer, M. Sicitui, and K. Premaratne. Stability of 2-d distributed processes with time-variant communication delays. In *Proc. of the IEEE International symposium on circuits and systems*, Sydney, NSW, Australia, May 2001.

- [38] C. Lorand, M. Sichitiu, Bauer P. H., and G. Schmidt. Stability of first order discrete time systems with time-variant communication delays in the feedback path. In *Proc. of the IEEE Asia Pacific conference on circuits and systems*, Tianjin, China, Dec. 2000.
- [39] M. Sichitiu, P. H. Bauer, and K. Premaratne. The effect of uncertain time variant delays in atm networks with explicit rate feedback. In *Proc. of the 20th IEEE american control conference*, Arlington, VA, USA, June 2001.
- [40] F-L. Lian, J. R. Moyne, and D. M. Tilbury. Performance evaluation of control networks for manufacturing systems. *ASME dynamic systems and control division*, 67(1):853–860, 2000.
- [41] F-L. Lian, J. R. Moyne, and D. M. Tilbury. Control performance study of a networked machining cell. In *Proc. of the*, Chicago, IL, USA, June 2000.
- [42] F-L. Lian, J. R. Moyne, and D. M. Tilbury. Performance evaluation of control networks: Ethernet, controlnet, and devicenet. *IEEE control systems magazine*, 21(1):66–83, 2001.
- [43] F-L. Lian, J. R. Moyne, and D. M. Tilbury. Analysis and modelling of networked control systems: MIMO case with multiple time delays. In *Proc. of the 20th IEEE american control conference*, Arlington, VA, USA, June 2001.
- [44] F-L. Lian, J. R. Moyne, and D. M. Tilbury. Time delay modelling and sample time selection for networked control systems. *ASME dynamic systems and control division*, 69(1):1–8, 2001.
- [45] F-L. Lian, J. R. Moyne, and D. M. Tilbury. Network design consideration for distributed control systems. *IEEE transactions on control systems technology*, 10(2):297–307, 2002.
- [46] F-L. Lian, J. R. Moyne, and D. M. Tilbury. Optimal controller design and evaluation for a class of networked control systems with distributed constant delays. In *Proc. of the 21th IEEE american control conference*, Anchorage, AK, USA, May 2000.
- [47] P. G. Otanez, J. R. Moyne, and D. M. Tilbury. Using deadbands to reduce communication in networked control systems. In *Proc. of the 21th IEEE american control conference*, Anchorage, AK, USA, May 2000.
- [48] J. K. Yook, D. M. Tilbury, and N. R. Sopakar. Trading computation for bandwidth: reducing communication in distributed control systems using state estimators. *IEEE transactions on control systems technology*, 10(4):503–518, 2002.
- [49] F-L. Lian, J. K. Yook, P. G. Otanez, D. M. Tilbury, and J. R. Moyne. Design of sampling and transmission rates for achieving control and communication performance in networked agent systems. In *Proc. of the 22 th IEEE american control conference*, Denver, CO, USA, June 2003.
- [50] N. B. Almutairi, M-Y. Chow, and Y. Tipsuwan. Network-based controlled DC motor with fuzzy compensation. In *Proc. of the 27 th IEEE annual conference on industrial electronics society*, Denver, CO, USA, Nov. 2001.

- [51] M-Y. Chow and Y. Tipsuwan. Network-based control systems: A tutorial. In *Proc. of the 27th IEEE annual conference on industrial electronics society*, Denver, CO, USA, Nov. 2001.
- [52] M-Y. Chow and Y. Tipsuwan. Network-based control adaptation for network qos variation. In *Proc. of the 27th IEEE military communications conference*, McLean, VA, USA, Oct. 2001.
- [53] Y. Tipsuwan and M-Y. Chow. Network-based controller adaptation based on qos negotiation and deterioration. In *Proc. of the 27th IEEE annual conference on industrial electronics society*, Denver, CO, USA, Nov. 2001.
- [54] N. B. Almutairi and M-Y. Chow. PI parameterization using adaptive fuzzy modulation (AFM) for IP networked PI control systems - part i: Partial adaptation. In *Proc. of the 28th IEEE annual conference on industrial electronics society*, Sevilla, Spain, Nov. 2002.
- [55] N. B. Almutairi and M-Y. Chow. PI parameterization using adaptive fuzzy modulation (AFM) for IP networked PI control systems - part ii: Full adaptation. In *Proc. of the 28th IEEE annual conference on industrial electronics society*, Sevilla, Spain, Nov. 2002.
- [56] Y. Tipsuwan and M-Y. Chow. Control methodologies in networked control systems. *ScienceDirect Control engineering practice*, 11(10):1099–1111, 2003.
- [57] N. B. Almutairi and M-Y. Chow. Stabilization of networked PI control system using fuzzy logic modulation. In *Proc. of the 22th IEEE american control conference*, Denver, CO, USA, June 2003.
- [58] M-Y. Chow and Y. Tipsuwan. Gain adaptation of networked DC motor controllers on QoS variations. *IEEE transactions on industrial electronics*, 50(5):936–943, 2003.
- [59] Y. Tipsuwan and M-Y. Chow. On the gain scheduling for networked PI controller over IP network. *IEEE transactions on mechatronics*, 9(3):491–498, 2004.
- [60] Y. Tipsuwan and M-Y. Chow. Model predictive path tracking via middleware for networked mobile robot over IP network. In *Proc. of the 23th IEEE american control conference*, Boston, MA, USA, July 2004.
- [61] Y. Tipsuwan and M-Y. Chow. Gain scheduler middleware: A methodology to enable existing controllers for networked control and teleoperation - part i: Networked control. *IEEE transactions on industrial electronics*, 51(6):1218–1227, 2004.
- [62] Y. Tipsuwan and M-Y. Chow. Gain scheduler middleware: A methodology to enable existing controllers for networked control and teleoperation - part ii: Teleoperations. *IEEE transactions on industrial electronics*, 51(6):1228–1237, 2004.
- [63] L. Hetel, J. Daafouz, J. P. Richard, and M. Jungers. Delay dependent sampled-data control based on delay estimates. *Elsevier, Systems Control Letter*, 60(2):146–150, 2011.

- [64] A. Kruszewski, W. J. Jiang, E. Fridman, A. Richard, and J. P. Toguyeni. A switched system approach to exponential stabilization through communication network. *IEEE Transactions on Control Systems*, 20(4):887–900, 2012.
- [65] B. Demirel, C. Briat, and K. Johansson. Deterministic and stochastic approaches to supervisory control design for networked systems with time-varying communication delays. *Elsevier, Nonlinear Analysis: Hybrid Systems*, 10(1):94–110, 2013.
- [66] F. R. P. Safaei, S. G. Ghiocel, J. P. Hespanha, and Chow J. H. Stability of an adaptive switched controller for power system oscillation damping using remote synchrophasor signals. In *Proc. of the 53rd IEEE Annual Conference on Decision and Control*, Los Angeles, CA, USA, Dec. 2014.
- [67] F. R. P. Safaei, S. G. Ghiocel, J. P. Hespanha, and J. H. Chow. Stability of an adaptive switched controller for power system oscillation damping using remote synchrophasor signals. In *Proc. of the 53rd IEEE Annual Conference on Decision and Control*, Los Angeles, CA, USA, Dec. 2014.
- [68] I. Kordonis and G. P. Papavassilopoulos. On stability and LQ control of MJLS with a markov chain with general state space. *IEEE Transactions on Automatic Control*, 59(2):535–540, 2014.
- [69] S. Xu, S. Jagannathanm, and F. L. Lewis. Stochastic optimal control of unknown linear networked control system in the presence of random delays and packet losses. *Elsevier, Automatica*, 48(6):1017–1030, 2012.
- [70] W. Chen and L. Qiu. Linear quadratic optimal control of continuous-time LTI systems with random input gains. *IEEE Transactions on Automatic Control*, 61(7):2008–2013, 2013.
- [71] S. Balasubramaniyan, S. Srinivasan, B. Subathra, and S. Ramaswamy. Design contracts for networked automation systems co-design. *CoRR*, abs/1507.05991, 2015.
- [72] D. Soudbakhsh, L. T. X. Phan, A. M. Annaswamy, and O. Sokolsky. Co-design of arbitrated network control systems with overrun strategies. *IEEE Transactions on Control of Network Systems*, PP(99):1–14, 2016.
- [73] Y. Niu, Y. Liang, and H. Yang. Event-triggered robust control and dynamic scheduling co-design for networked control system. In *Proc. of the 14th IEEE International Bhurban Conference on Applied Sciences and Technology (IBCAST)*, Islamabad, Pakistan, Jan. 2017.
- [74] Z. Min, W. Jin, and M. Aojia. Co-design of optimal feedback scheduling strategy and fault detection for networked control systems. In *Proc. of the IEEE Chinese Guidance, Navigation and Control Conference*, Nanjing, China, Aug. 2016.
- [75] S. Wang, Y. Jiang, Y. Li, and D. Liu. Fault detection and control co-design for discrete-time delayed fuzzy networked control systems subject to quantization and multiple packet dropouts. *Elsevier, Fuzzy Sets and Systems*, 306:1–25, 2017.

- [76] C. Meng, T. Wang, W. Chou, S. Luan, Y. Zhang, and Z. Tian. Remote surgery case: robot-assisted teleneurosurgery. In *Proc. of the IEEE International Conference on Robotics and Automation*, New Orleans, LA, USA, April 2004.
- [77] W. Hu, H. Zhou, and Q. Deng. Wireless event-driven networked predictive control over internet. In *Proc. of the IEEE International Conference on Control*, Cardiff, UK, Sep. 2012.
- [78] M. S. Ananyevskiy. Control over internet with timecheck denial gain. *IFAC-PapersOnLine*, 48(11):691–693, 2015.
- [79] M. H. Hung, J. Tsai, F. T. Cheng, and H. C. Yang. Development of an ethernet-based equipment integration framework for factory automation. *Elsevier, Robotics and Computer-Integrated Manufacturing*, 20(5):369–383, 2004.
- [80] M. Gaele, B. Denis, J. M. Faure, and G. Frey. Evaluation of response time in ethernet-based automation systems. In *Proc. of the IEEE International Conference on Emerging Technologies and Factory Automation*, Prague, Czech Republic, Sep. 2006.
- [81] L. Liu and G. Frey. Simulation approach for evaluating response times in networked automation systems. In *Proc. of the IEEE Conference on Emerging Technologies and Factory Automation*, Patras, Greece, Sep. 2007.
- [82] B. Addad, S. Amari, and J. J. Lesage. Analytic calculus of response time in networked automation systems. *IEEE Transactions on Automation Science and Engineering*, 7(4):858–869, 2010.
- [83] S. Srinivasan, F. Buonopane, J. Vain, and S. Ramaswamy. Model checking response times in networked automation systems using jitter bounds. *Elsevier, Computers in Industry*, 74:186–200, 2015.
- [84] W. Geelen, D. Antunes, J. P. M. Voeten, R. R. H. Schiffelers, and W. P. M. H. Heemels. The impact of deadline misses on the control performance of high-end motion control systems. *IEEE Transactions on Industrial Electronics*, 63(2):1218–1229, 2016.
- [85] P. Lindgren, J. Eriksson, M. Lindner, A. Lindner, D. Pereira, and L. M. Pinho. The impact of deadline misses on the control performance of high-end motion control systems. *IEEE Transactions on industrial informatics*, 13(1):287–296, 2017.
- [86] B. Vogel-Heuser, j. Folmer, G. Frey, L. Liu, H. Hermanns, and A. Hartmanns. Modeling of networked automation systems for simulation and model checking of time behavior. In *Proc. of the 9th IEEE Conference on Emerging Technologies and Factory Automation*, Chemnitz, Germany, Mar. 2012.
- [87] J. Greifeneder and G. Frey. Optimizing quality of control in networked automation systems using probabilistic models. In *Proc. of the IEEE Conference on Emerging Technologies and Factory Automation*, Prague, Czech Republic, Sep. 2006.

- [88] J. Greifeneder and G. Frey. Model checking response times in networked automation systems using jitter bounds. In *Proc. of the IEEE International Symposium on Software Reliability Engineering Workshops*, Naples, Italy, Nov. 2014.
- [89] M. Strelec, K. Macek, and A. Abate. Modeling and simulation of a microgrid as a stochastic hybrid system. In *Proc. of the 3rd IEEE PES Innovative Smart Grid Technologies Europe (ISGT Europe)*, Berlin, Germany, Oct. 2012.
- [90] E. M. Clarke, O. Grumberg, and D. E. Long. Model checking and abstraction. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 16(5):1512–1542, 1994.
- [91] D. Dams, R. Gerth, and O. Grumberg. Abstract interpretation of reactive systems. *ACM Transactions on Programming Languages and Systems*, 19(2):253–291, 1994.
- [92] P. de la Cámara, M. M. Gallardo, P. Merino, and D. Sanán. Model checking software with well-defined apis: the socket case. In *Proc. of the 10th international Workshop on Formal Methods For industrial Critical Systems*, Lisbon, Portugal, Sept. 2005.
- [93] M. Gallardo, P. Merino, and E. Pimentel. Refinement of LTL formulas for abstract model checking. In *Proc. of the 9th international Symposium on Static Analysis (LNCS)*, Madrid, Spain, Sept. 2002.
- [94] J. Hatcliff, M. B. Dwyer, and H. Zheng. Slicing software for model construction. *Higher-Order and Symbolic Computation*, 13(4):315–353, 2000.
- [95] G. J. Holzmann and R. Joshi. Model-driven software verification. In *Proc. of the International SPIN Workshop on Model Checking of Software*, Barcelona, Spain, Apr. 2004.
- [96] J. I. Rasmussen, K. G. Larsen, and A. David. Guided controller synthesis for climate controller using UPPAAL. In *Proc. of the International Conference on Formal Modeling and Analysis of Timed Systems*, Salzburg, Austria, Oct. 2007.
- [97] A. Rashid, O. Hasan, and K. Saghar. Formal analysis of a zigbee-based routing protocol for smart grids using UPPAAL. In *Proc. of the IEEE International Conference on High-Capacity Optical Networks and Enabling/Emerging Technologies (HONET)*, Islamabad, Pakistan, Dec. 2007.
- [98] P. Luo, R. Wang, X. Li, Y. Guan, and J. Zhang. Model checking for spacewire link interface design using UPPAAL. In *Proc. of the 37th IEEE Annual Computer Software and Applications Conference Workshops*, Japan, Japan, July 2013.
- [99] E. P. Enoiu, D. Sundmark, and P. Pettersson. Model-based test suite generation for function block diagrams using the UPPAAL model checker. In *Proc. of the 6th IEEE International Conference on Software Testing, Verification and Validation Workshops*, Luxembourg, Luxembourg, Mar. 2013.
- [100] K. Nichols and V. Jacobson. Controlling queue delay: a modern AQM is just one piece of the solution to bufferbloat. *ACM Queue - Networks*, 10(5):1–15, 2012.

- [101] R. Pan, P. Natarajan, C. Piglione, M. S. Prabhu, V. Subramanian, F. Baker, and B. VerSteeg. PIE: A lightweight control scheme to address the bufferbloat problem. In *Proc. of IEEE International Conference on High Performance Switching and Routing (HPSR)*, Taipei-Taiwan, July 2013.
- [102] L. M. A. Sup, R. M. Moraes, and A. Bauchspiess. Explicit non-congestion notification: A new AQM approach for TCP networks. In *Proc. of 13th IEEE International Wireless Communications and Mobile Computing Conference*, Valencia, Spain, Jun 2017.
- [103] L. M. A. Sup. Arquitetura física e lógica de uma smart microgrid para a gestão integrada da energia: Um modelo para o parque tecnológico itaipú. Master's thesis, Unioeste, Universidade do Oeste do Paraná, 2012.
- [104] S. Floyd and V. Jacobson. Random early detection gateways for congestion avoidance. *IEEE/ACM Transactions Networking*, 1(4):397–413, 1993.
- [105] WC. Feng, D. D. Kandlur, D. Saha, and K. G. Shin. A self-configuring red gateway. In *in Proc. of 18th IEEE Annual Joint Conference on Computer and Communications Societies INFOCOM*, New York, NY, USA, 1999.
- [106] CF. Ku, S. J. Chen, J. M. Ho, and R. I. Chang. Improving end-to-end performance by active queue management. In *in Proc. of 19th International Conference on Advanced Information Networking and Applications, AINA'05*, Taipei, Taiwan, March 2005.
- [107] L. F. Figueredo, E. D. S. Alves, J. Y. Ishihara, G. A. Borges, and A. Bauchspiess. Stability of networked control systems with dynamic controllers in the feedback loop. In *Proc. of IEEE Mediterranean Conference on Control and Automation (MED)*, Marrakech, Morocco, June 2010.
- [108] W. Zhang. Stability analysis of networked control systems, 2001.
- [109] S. Savazzi, S. Guardiano, and U. Spagnolini. Wireless critical process control in oil and gas refinery plants. In *Proc. of IEEE International Conference on Industrial Technology (ICIT)*, Athens, Greece, 2012.
- [110] C. P. Fu and S. C. Liew. TCP VenO: TCP enhancement for transmission over wireless access networks. *IEEE Journal on Selected Areas in Communications*, 21(2):216–228, 2003.
- [111] L. Ding, X. Wang, Y. Xu, W. Zhang, and W. Chen. Vegas-W: An enhanced TCP-Vegas for wireless ad hoc networks. In *Proc. of the IEEE International Conference on Communications*, Beijing, China, May 2008.
- [112] P. Sreekumari and SH. Chung. TCP NCE: A unified solution for non-congestion events to improve the performance of TCP over wireless networks. *EURASIP Journal on Wireless Communications and Networking*, 23(1):1–20, 2011.
- [113] V. Jacobson. Modified TCP congestion avoidance algorithm. *Technical Report 30*, April 1990.

- [114] A. S. Peng, F. Cardona, K. Shafiee, and V. C. M. Leung. TCP performance evaluation over GEO and LEO satellite links between performance enhancement proxies. In *Proc. of the IEEE Vehicular Technology Conference (VTC Fall)*, Quebec City, QC, Canada, Sept. 2012.
- [115] K. Fall and S. Floyd. Simulation-based comparisons of tahoe, reno and SACK TCP. *SIGCOMM Comput. Commun. Rev.*, 26(3):5–21, 1996.
- [116] C. Parsa and J. J. Garcia-Luna-Aceves. Improving TCP congestion control over internets with heterogeneous transmission media. In *in Proc. of IEEE International Conference on Network Protocols (ICNP)*, California, USA, Nov 1999.
- [117] R. Jain, D. Chiu, and Hawe W. A quantitative measure of fairness and discrimination for resource allocation in shared computer systems. *CoRR*, cs.NI/9809099, 1998.
- [118] S. Floyd. TCP and explicit congestion notification. *ACM Computer Communications Review*, 24(5):10–23, 1994.
- [119] S. Ramakrishnan, K. anf Floyd and D. Black. The addition of explicit congestion notification ECN to IP. In *RFC 3168*, Hong Kong, China, Sept. 2001.
- [120] B. Trammell, M. Kuhlewind, D. Boppart, I. Learmonth, G. Fairhurst, and R. Scheffenegger. Enabling internet-wide deployment of explicit congestion notation. In *Proc. of the Passive and Active Measurement Conference*, New York, USA, Mar. 2015.
- [121] A. Medina, M. Allman, and S. Floyd. Measuring interactions between transport protocols and middleboxes. In *Proc. of 4th ACM SIGCOMM Conference on Internet Measurement (IMC)*, Taormina, Sicily, Italy, Nov. 2004.
- [122] S. Bauer, R. Beverly, and A. Berger. Measuring the state of ECN readiness in servers, clients, and routers. In *Proc. of 11th ACM ACM SIGCOMM Conference on Internet Measurement (IMC)*, Berlin, Germany, Nov. 2011.
- [123] M. Kühlewind, S. Neuner, and B. Trammell. On the state of ECN and TCP options on the Internet. In *Proc. of the 14th Conference on Passive and Active Measurement (PAM)*, Hong Kong, China, Mar. 2013.
- [124] J. Padhye and S. Floyd. Identifying the TCP behavior of web servers. In *Proc. of the ACM SIGCOMM conference*, San Diego, CA, USA, Aug. 2001.
- [125] A. Medina, M. Allman, and S. Floyd. Measuring the evolution of transport protocols in the internet. *SIGCOMM Computer Communications Review*, 35(2):37–52, 2005.
- [126] Your app and next generation networks. In *Apple Inc.*, June 2015.
- [127] J. Postel. Internet procol. In *RFC 791*, California, IEFT, 1981.
- [128] J. Postel. Internet procol, version 6 IPv6 specification. In *RFC 2460*, California, IEFT, 1998.
- [129] N. Spring, D. Wetherall, and D. Ely. Robust explicit congestion notification ECN signaling with nonces. In *RFC 3540*, University of Washington, 2003.

- [130] H. Jamal and A. Uvaiz. Performance evaluation of explicit congestion notification ECN in IP networks. In *RFC 2884*, 2000.
- [131] M. Malowidzki. Simulation-based study of ECN performance in RED networks, 2003.
- [132] Z. Li and G. Zhang. A novel differentiated marking strategy based on explicit congestion notification for wireless TCP. In *Proc. of the IEEE TENCN conference*, Hong Kong, China, Nov. 2006.
- [133] C. Liu and R. Jain. Delivering faster congestion feedback with the mark-front strategy. In *Proc. of the IEEE International Conference on Communication Technology Proceedings WCC-ICCT*, Beijing, China, Aug. 2000.
- [134] L. Kalampoukas, A. Varma, and K. K. Ramakrishnan. Explicit window adaptation: a method to enhance TCP performance. In *Proc. of the 17th Annual Joint Conference of the IEEE Computer and Communications Societies*, San Francisco, USA, Apr. 1998.
- [135] L. Kalampoukas, A. Varma, and K. K. Ramakrishnan. Explicit window adaptation: A method to enhance TCP performance. *IEEE/ACM Transactions on Networking*, 10(3):338 – 350, 2002.
- [136] M. Talau, M. Fonseca, and Wille C. G. TCP em redes ad hoc: A influência do tamanho de pacotes e das filas. In *Proc. of the XXXIV Simpósio Brasileiro de telecomunicações SBrT*, Santarmém, PA, Brasil, Aug. 2016.
- [137] K. Xu, Y. Tian, and N. Ansari. TCP-Jersey for wireless IP communications. *IEEE Journal on selected areas in communications*, 22(4):747–756, 2004.
- [138] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan. Data center TCP DCTCP. In *Proc. of ACM SIGCOMM International Conference*, New York, NY, USA, Oct. 2010.
- [139] Y. Huang and B. Hu. Enhanced DCTCP to explicitly inform of packet loss. In *Proc. of IEEE International Conference on Communications (ICC)*, London, UK, June 2015.
- [140] M. Miao, P. Cheng, F. Ren, and R. Shu. Slowing little quickens more: Improving DCTCP for massive concurrent flows. In *Proc. of 14 th IEEE International Conference on Parallel Processing*, Beijing, China, Sept. 2015.
- [141] W. Chen, P. Cheng, F. Ren, R. Shu, and C. Lin. Ease the queue oscillation: Analysis and enhancement of DCTCP. In *Proc. of the 33rd IEEE International Conference on Distributed Computing Systems*, Philadelphia, PA, USA, July 2013.
- [142] S. Jouet, C. Perkins, and D. Pezaros. Slowing little quickens more: Improving DCTCP for massive concurrent flows. In *Proc. of the IEEE/IFIP NOMS Network Operations and Management Symposium*, Istanbul, Turkey, Apr. 2016.

- [143] H. B. Mokadem, B. Bérard, O. D. Smet, and J. M. Roussel. Verification of a timed multitask system with UPPAAL. *IEEE Transactions on Automation Science and Engineering*, 7(4):921–932, 2010.
- [144] G. Behrmann, A. David, K. G. Larsen, J. Hakansson, P. Pettersson, W. Yi, and M. Hendriks. UPPAAL 4.0. *in Proc. of QEST, Riverside*, Sept 2006.
- [145] R. Alur and D. L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2), 1994.
- [146] E. A. Emerson and E. M. Clarke. Using branching-time temporal logic to synthesize synchronization skeletons. *Science of Computer Programming*, 2:241–266, 1982.
- [147] T. Hune, K. G. Larsen, and P. Pettersson. Guided synthesis of control programs using UPPAAL. *In Proc. of IEEE ICDCS International Workshop on Distributed Systems Verification and Validation*, Apr. 2000.
- [148] P. R. D’Argenio, J. P. Katoen, T. C. Ruys, and G. J. Tretmans. The bounded retransmission protocol must be on time. *In Proc. of 3rd International Workshop on tools and algorithms for the construction and analysis of systems*, Enschede, Netherlands, Apr. 1997.
- [149] A. David and W. Yi. Modelling and analysis of a commercial field bus protocol. *In Proc. of 12th IEEE Euromicro Conference on Real-Time Systems Euromicro RTS*, Stockholm, Sweden, June 2000.
- [150] M. G. SOrensen. Controller synthesis for home automation. Master’s thesis, Computer Science Department, Aalborg University, Denmark, June 2014.
- [151] K. J. Astrom and B. Wittenmark. *Adaptive Control*. Addison-Wesley, Englewood Cliffs, NJ, 2 edition, 1995.
- [152] M. C. C. Melo, A. A. S. Heyder, and A. Bauchspiess. Energy saving of adaptive thermal control for pwm driven air-conditioners. *LARA-Laboratório de Automação e Robótica, UnB*, 2014.
- [153] M. Guinaldo, J. Sánchez, and S. Dormido. Diseño de un sistema de control anticipativo basado en paquetes para control en red. *In Proc. of 9th Conferencia Iberoamericana en Sistemas, Cibernética e Informática*, Orlando, FL, USA, July 2010.
- [154] K. J. Astrom, C. C. Hang, and B. C. Lim. A new smith predictor for controlling a process with an integrator and long dead-time. *IEEE transactions on automatic control*, 39(2):343 – 345, 1994.
- [155] M. Wakaiki, M. Ogura, and J. P. Hespanha. Linear quadratic control for sampled-data systems with stochastic delays. *In Proc. of American Control Conference*, WA, USA, May. 2017.
- [156] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow. TCP selective acknowledgment options. *RFC 2018*, Oct 1996.

- [157] R. de Oliveira and T. Braun. A smart TCP acknowledgment approach for multihop wireless networks. *IEEE Transactions on Mobile Computing*, 6(2):192–205, 2007.
- [158] C. F. Ku, S. J. Chen, and R. Chang. Improving end-to-end performance by active queue management. In *International Conference on Advanced Information Networking and Applications (AINA)*, Taipei, Taiwan, March 2005.
- [159] D. Amol and P. Rajesh. A review on active queue management techniques of congestion control. In *Proc. of IEEE International Conference on Electronic Systems, Signal Processing and Computing Technologies (ICESC)*, Nagpur, India, Jan 2014. IEEE.
- [160] R. Adams. Active queue management: a survey. *IEEE Communication Surveys & Tutorials*, 15(3):1425–1476, 2013.
- [161] F. Jäger, T. C. Schmidt, and M. Wählisch. How dia-shows turn into video flows: adapting scalable video communication to heterogeneous network conditions in real-time. In *Proc. of IEEE Conference on Local Computer Networks (LCN)*, Edmonton Canada, Sept 2014.
- [162] S. Floyd. TCP and explicit congestion notification. *ACM Computer Communications Review*, 24(5):8–23, 1994.
- [163] P. Varutti and R. Findeisen. Event-based NMPC for networked control systems over UDP-like communication channels. in *Proc. of IEEE American Control Conference (ACC)*, Jun 2011.
- [164] J. M. Llopis, J. Pieczerek, and T. Janaszka. Minimizing latency of critical traffic through SDN. In *Proc. of IEEE International Conference on Networking, Architecture, and Storage (NAS)*, CA USA, Jun 2016.
- [165] H. Lin, H. Su, and Z. Shu. Optimal estimation in UDP-Like networked control systems with intermittent inputs: Stability analysis and suboptimal filter design. *IEEE Trans. on Automatic Control*, 61(7):1794–1809, 2016.
- [166] H. S. Khan and M. B. Kadri. Fuzzy scheduled gain middleware for networked control systems. in *Proc. of IEEE International Conference on Robotics and Emerging Allied Technologies in Engineering (iCREATE)*, Apr 2014.
- [167] J. S. Farrokh and H. Iraj. Experimental analysis of mobile-robot teleoperation via shared impedance control. *IEEE Transaction on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 41(2):591 – 606, 2011.

ANEXOS

I. MODELO DA TOPOLOGIA DAISY CHAIN NO UPPAAL

Neste Anexo será apresentada uma modelagem de um fluxo TCP-Reno cuja topologia de comunicação foi modelada como um sistema autômato finito realizado no UPPAAL. Na qual, transmissor, receptor, roteadores e cada canal foram modelados como autômatos temporizados e o sistema inteiro funciona como um conjunto de tais autômatos em paralelo, ou seja, a topologia da rede de comunicação foi modelada como um conjunto de autômatos temporizados.

Com a finalidade de comparar resultados obtidos em UPPAAL, primeiramente foi modelado o mesmo cenário básico encontrado na Seção 4.1 de [116] e ilustrado na Figura I.1. No qual um transmissor transmite pacotes de 1 kbyte para um receptor através de uma rede composta por dois roteadores (chamados roteador 1 e roteador 2 respectivamente) cujas filas (ou *buffer*) possuem capacidades para enfileirar até 17 pacotes de 1 kbyte cada, e a partir do momento que a fila estiver cheia, os roteadores começam a descartar pacotes.

Os valores iniciais das variáveis do TCP-Reno $Ssth$ e $rwnd$, foram ajustados em 30 e 64 pacotes, respectivamente (igual a [116]). Os roteadores são conectados por meio de um canal de 1,5 Mbps de capacidade e 5 ms de atraso de propagação, chamado canal 2. Por outro lado, o transmissor é conectado com o roteador 1 por meio de um canal com capacidade de 10 Mbps e 2 ms de atraso de propagação chamado canal 1, finalmente o receptor é conectado com o roteador 2, por meio de um canal com capacidade de 10 Mbps e 33 ms de atraso de propagação chamado canal 3.

No modelo feito em UPPAAL dessa topologia de rede de comunicação, transmissor, receptor, roteadores e cada canal de comunicação foram modelados mediante autômatos temporizados, ou seja, mediante máquina de estados finitos estendidas com relógios e variáveis inteiras e flutuantes, e a topologia de rede de comunicação completa funciona como um sistema de tais autômatos temporizados em paralelo.

A Figura I.2 ilustra o conjunto de autômatos completo que modelam a topologia de rede de comunicação da Figura I.1. Esse modelo foi simulado por 10 s nos quais foi considerado que o transmissor TCP-Reno sempre tinha pacotes para transmitir.

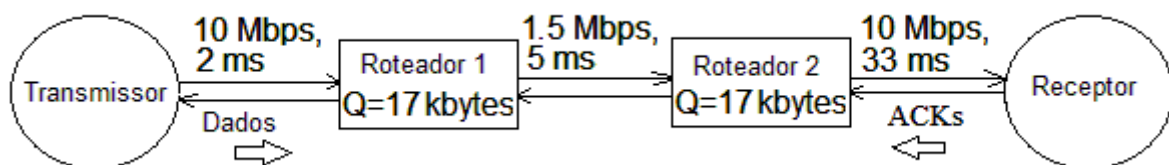


Figura I.1: Topologia de rede de comunicação daisy-chain com um fluxo TCP.

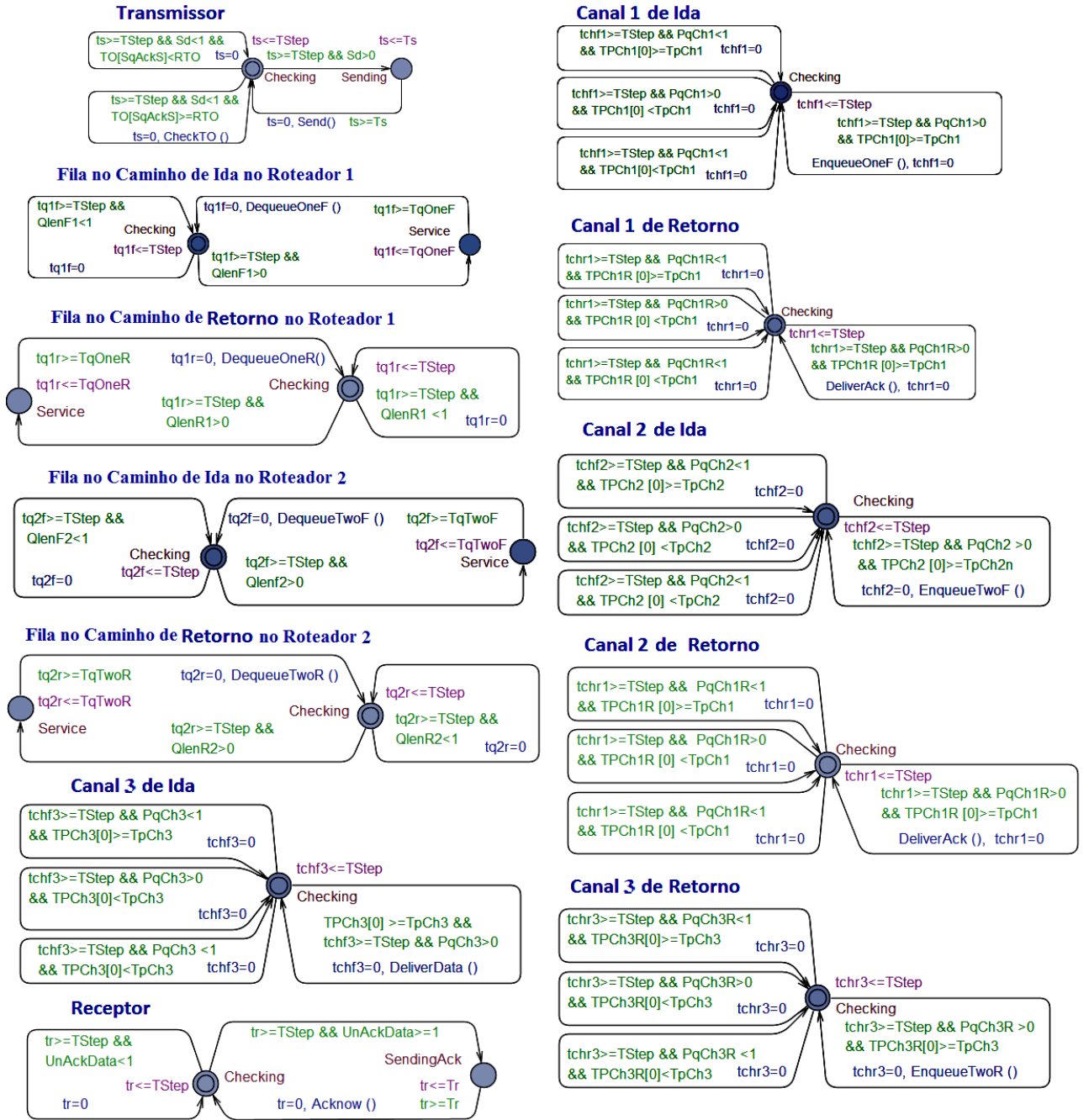


Figura I.2: Conjunto de autômatos que modelam a topologia de rede de comunicação daisy-Chain da Figura I.1.

A seguir serão descritos os autômatos ilustrados na Figura I.2.

I.0.1 Modelo do Transmissor

A Figura I.3 mostra o modelo do transmissor o qual consiste de dois estados denominados *Checking* e *Sending*, e o pseudocódigo para este autômato é dado no Algoritmo I.1, enquanto que as variáveis utilizadas no modelo autômato temporizado são descritas na Tabela I.0.1.

Conforme observasse na Figura I.3 o autômato inicia no estado **Checking** e permanece nele durante T_{Step} unidades de tempo. Logo se tiver pacotes para enviar, ou seja, se após um tempo T_{Step} a variável S_d for maior que zero, o autômato migra para o estado **Sending**, onde permanece por T_s unidades de tempo. Depois, transita novamente para o estado **Checking** e nessa transição chama a função **Send** e reinicia seu relógio ts .

Tabela I.1: Descrição das variáveis, funções e constantes utilizadas no modelo do transmissor.

Nome	Descrição
T_{Step}	Passo de tempo (resolução temporal) - valor adotado $=1\mu s$.
T_s	Constante igual ao tempo em μs requerido para colocar um pacote de 1 kbyte em um canal com capacidade 10 Mbps (canal 1).
ts	Relógio que coordena as transições dos estados Checking e Sending .
S_d	Variável que contém a quantidade de pacotes que o transmissor TCP pode enviar, incrementada de acordo com o algoritmo TCP-Reno que faz a atualização cada vez que recebe um ACK, e decrementada cada vez que o transmissor envia um pacote.
T_O	Arranjo de Relógios que contenham o tempo de espera para cada pacote enviado pelo transmissor e cuja recepção ainda não foi confirmada por um ACK.
S_qAckS	Cadeia de T_O .
CheckTO	Função chamada cada vez que ocorrer um <i>timeout</i> . Basicamente reinicia a fase de partida lenta e ajusta a sequência de pacotes para retransmitir o pacote considerado perdido.
Send	Função chamada cada vez que o transmissor envia um pacote. Basicamente decrementa a variável S_d em um pacote e incrementa a variável P_qCh1 em um pacote
P_qCh1	Variável que contém a quantidade de pacotes dentro do canal 1 de ida.

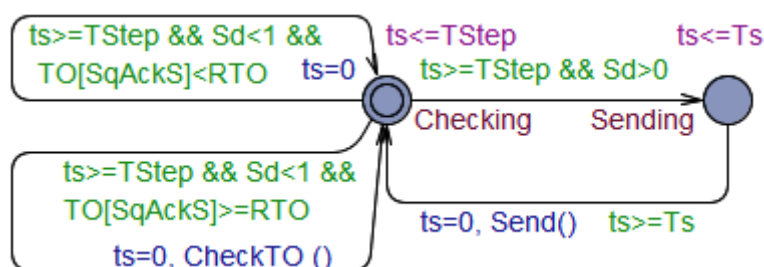


Figura I.3: Modelo do transmissor TCP-Reno.

Algoritmo I.1 Pseudocódigo para o autômato da Figura I.3.

Pseudocódigo para o autômato do transmissor	Observações
Inicializa no estado <code>Checking</code>	
if ($ts \geq TStep$) {	
if ($Sd > 0$) { vai para o estado <code>Sending</code> espera Ts unidades de tempo e então retorna para o estado <code>Checking</code> , reinicia ts e chama a função <code>Send</code> }	Transição do lado direito na Figura I.3
if ($Sd = 0$ and $TO [SqAckS] < RTO$) { Faz uma transição para o mesmo estado e reinicia ts }	Transição do lado superior esquerdo na Figura I.3
if ($Sd = 0$ and $TO [SqAckS] \geq RTO$) { Faz uma transição para o mesmo estado reinicia ts e chama a função <code>CheckTO</code> }	Transição do lado inferior esquerdo na Figura I.3
}	Fim do bloco ($ts \geq TStep$)
else {espera até $ts \geq TStep$ ser verdadeiro}	

Se, estando no estado `Checking`, ocorrer um *timeout* o autômato transita para esse mesmo estado conforme se observa e na seta inferior esquerda da Figura I.3, e nessa transição chama a função `CheckTO`, que reinicia a fase de partida lenta do TCP e retransmite o pacote considerado perdido. Se estando no estado `Checking` passar $TStep$ unidades de tempo e não houver pacotes para transmitir e nem ocorrer *timeout*, então o autômato faz uma transição para esse mesmo estado e reinicia o relógio ts conforme ilustra a seta superior esquerda da Figura I.3.

A variável Sd além de ser atualizada pela função `Send`, também é atualizada cada vez que o transmissor receber um ACK do receptor, de acordo com o valor da janela de transmissão que evolui seguindo o algoritmo TCP-Reno. Essa dinâmica é calculada pela função `DeliverAck` chamada pelo autômato que modela o canal 1 de retorno.

I.0.2 Modelo do canal 1 de ida

A rede mostrada na Figura I.1 contém três canais, o primeiro localizado entre o transmissor e o roteador 1 (chamado canal 1), o segundo entre o roteador 1 e o roteador 2 (chamado canal 2), e o terceiro localizado entre o roteador 2 e o receptor (chamado canal 3). Cada canal foi modelado com dois autômatos um para a ida, por onde circulam dados que vão do remetente ao receptor, e outro de retorno, por onde circulam ACKs do receptor para o transmissor.

A Figura I.4 ilustra o autômato que modela o canal 1 de ida (localizado entre o transmissor 1 e o roteador 1), o pseudocódigo desse modelo é dado no Algoritmo I.2, enquanto que as constantes, variáveis e funções utilizadas no modelo autômato temporizado são descritas na Tabela I.2.

Conforme pode ser observado na Figura I.4, esse autômato contém transições para um único estado que acontecem a cada $TStep$ unidades de tempo. A transição da direita acontece cada vez que existe pelo menos um pacote no canal, ou seja, $PqCh1 > 0$ for verdadeiro, e um pacote já chegou no roteador 1, isto é, $TPCh1[0] \geq TpCh1$ também for verdadeiro. Onde $TPCh1[0]$ é um cronômetro disparado no momento que o pacote que está no extremo do canal (chegando na fila do roteador 1) saiu do transmissor. Nessa transição é reiniciado o relógio $tchf1$ e é chamada a função `EnqueueOneF` a qual basicamente modela a saída de um pacote do respectivo canal e sua entrada

na fila de ida do roteador 1.

As transições da esquerda ocorrem quando após um tempo $TStep$ não existir pacotes no canal ou se existir ainda não se propagaram até o roteador 1.

Os outros canais foram modelados de forma análoga.

Tabela I.2: Descrição das variáveis, funções e constantes utilizadas no modelo do canal 1 de ida.

Nome	Descrição
$TStep$	Passo de tempo (resolução temporal) - valor adotado $=1\mu s$
$tchf1$	Relógio que coordena as transições do estado Checking.
$TpCh1$	Constante igual ao tempo de propagação no canal 1 (2 ms).
$TPCh1$	Arranjo de Relógios que contenham o tempo de de propagação de cada um dos pacotes que já saíram do transmissor mas ainda não chegaram no roteador 1, ou seja, o tempo transcorrido desde que cada pacote saiu do transmissor até o instante atual.
$EnqueueOneF$	Função chamada cada vez que um pacote chega à fila de ida do roteador 1. Basicamente incrementa a variável $QlenF1$ em um pacote e decrementa a variável $PqCh1$ em um pacote.
$PqCh1$	Variável que contém a quantidade de pacotes dentro do canal 1 de ida.
$QlenF1$	Variável que contém a quantidade de pacotes dentro da fila de ida do roteador 1.

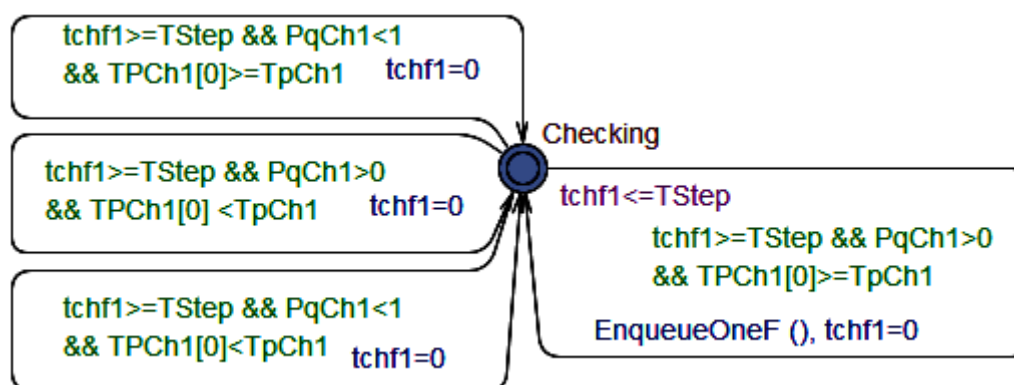


Figura I.4: Modelo do canal 1 de ida.

Algoritmo I.2 Pseudocódigo para o autômato da Figura I.4.

Pseudocódigo para o autômato do canal 1 de ida

Inicializa no estado Checking

```

if (tchf1 >= TStep) {
  if (PqCh1 > 0 and TPCh1 [0] >= TpCh1)
  { Faz uma transição para o mesmo estado reinicia tchf1 e chama
  a função EnqueueOneF }
  if (PqCh1 < 1 or TPCh1 [0] < TpCh1)
  { Faz uma transição para o mesmo estado e reinicia tchf1 }
}
else {espera até tchf1 >= TStep ser verdadeiro }

```

Observações

Transição do lado direito na Figura I.4

Transições do lado esquerdo na Figura I.4

Fim do bloco (tchf1 >= TStep)

I.0.3 Modelo da fila no caminho de ida no roteador 1

A topologia de rede de comunicação mostrada na Figura I.1 contém dois roteadores. Cada roteador foi modelado com dois autômatos, um autômato para modelar a fila no caminho de ida, onde são enfileirados os pacotes de dados que viajam do transmissor para o receptor, e outro para modelar a fila no caminho de retorno, onde são enfileirados os pacotes de ACKs que viajam do receptor para o transmissor.

A Figura I.5 ilustra o autômato que modela a fila do caminho de ida no roteador 1, o qual consiste de dois estados, denominados **Checking** e **Service**. O pseudocódigo para este autômato é dado no Algoritmo I.3, enquanto que as constantes, variáveis e funções utilizadas no modelo autômato temporizado são descritas na Tabela I.3.

Conforme pode ser observado na Figura I.5, o autômato inicia no estado **Checking** e permanece nele durante **TStep** unidades de tempo. Logo, se tiver pacotes na fila, ou seja, se $QlenF1 \geq 1$ for verdadeiro, o autômato migra para o estado **Service**, conforme ilustra a seta inferior direita na Figura I.5. No estado **Service** o autômato permanece por **TqOneF** unidades de tempo (igual ao tempo necessário para colocar um pacote de 1 kbyte em um canal de 1,5 Mbps). Depois, o autômato retorna para o estado **Checking** e nesta transição de retorno chama a função **DequeueOneF**, e reinicia seu relógio **tq1f** conforme ilustra a seta superior para a esquerda na Figura I.5.

Se estando no estado **Checking** passar **TStep** unidades de tempo, ou seja, $tq1f \geq TStep$ for verdadeiro, e não houver pacotes na fila, ou seja, $QlenF1 < 1$ também for verdadeiro, então o autômato faz uma transição para esse mesmo estado e reinicia o relógio **tq1f** conforme ilustra a seta esquerda da Figura I.5.

As outras filas foram modeladas de forma análoga.

Tabela I.3: Descrição das variáveis, funções e constantes utilizadas no modelo da fila de ida no roteador 1.

Nome	Descrição
TStep	Passo de tempo (resolução temporal) - valor adotado $=1\mu s$.
tq1f	Relógio que coordena as transições dos estados Checking e Service .
TqOneF	Constante igual ao tempo em μs requerido para colocar um pacote de 1 kbyte em um canal com capacidade 1,5 Mbps (canal 2).
DequeueOneF	Função chamada cada vez que um pacote sai da fila de ida do roteador 1. Basicamente decrementa QlenF1 em um pacote e incrementa a variável PqCh2 em um pacote.
QlenF1	Variável que contém a quantidade de pacotes dentro da fila de ida do roteador 1.
PqCh2	Variável que conta a quantidade de pacotes dentro do canal 2 de ida.

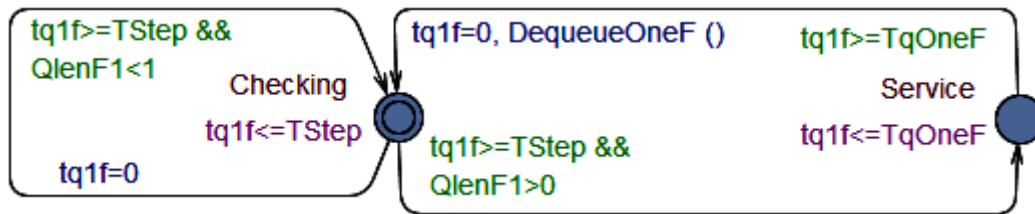


Figura I.5: Modelo da fila de dados no caminho de ida, no roteador 1.

Algoritmo I.3 Pseudocódigo para o autômato da Figura I.5.

Pseudocódigo do autômato que modela a fila no caminho de ida no roteador 1

```

Inicializa no estado Checking
if (tq1f >= TStep) {
  if (QlenF1 > 0) { vai para o estado Service espera TqOneF
  unidades de tempo e então retorna para o estado Checking,
  reinicia tq1f e chama a função DequeueOneF }
  if (QlenF1 < 1) { Faz uma transição para o mesmo estado e
  reinicia tq1f }
}
else { espera até tq1f >= TStep ser verdadeiro }

```

Observações

Transição do lado direito na
Figura I.5

Transição do lado esquerdo na
Figura I.5

Fim do bloco (tq1f >= TStep)

I.0.4 Modelo do Receptor

Figura I.6 ilustra o autômato que modela o receptor, o pseudocódigo desse modelo é dado no Algoritmo I.4, enquanto que as constantes, variáveis e funções utilizadas no modelo autômato temporizado são descritas na Tabela I.4.

Conforme pode ser observado na Figura I.6 esse autômato contém dois estados denominados **Checking** e **SendingAck**. O autômato começa no estado **Checking** e cada vez que chegar um pacote, isto é, quando $UnAckP > 0$ for verdadeiro o autômato migra do estado **Checking** para o estado **SendingAck** onde permanece por T_s unidades de tempo. Depois, o autômato retorna para o estado **Checking**, e nessa transição de retorno chama a função **AckNow**.

Se o automato estiver no estado **Checking** e após um tempo T_{Step} não haver novos pacotes de dados cujos ACKs ainda não foram enviados, ou seja, $UnAckP < 1$ for verdadeiro, então o autômato faz uma transição para esse mesmo estado e reinicia seu relógio tr , conforme ilustra a seta da esquerda na Figura I.6.

Tabela I.4: Descrição das variáveis, funções e constantes utilizadas no modelo do receptor.

Nome	Descrição
TStep	Passo de tempo (resolução temporal) - valor adotado $=1\mu s$.
tr	Relógio que coordena as transições dos estados Checking e SendingAck.
Tr	Limiar de tempo requerido para colocar um ACK em um canal com capacidade 10 Mbps (canal 3).
UnAckP	Variável que contém a quantidade de pacotes que chegaram ao receptor e cujos ACKs ainda não foram enviados.
AckNow	Função chamada cada vez que o receptor envia um ACK. Basicamente calcula a sequência do ACK a ser enviado e decrementa a variável UnAckP em um pacote e incrementa a variável PqCh3R em um pacote.
UnAckP	Variável que contém a quantidade de pacotes que já chegaram no receptor e cujo ACK ainda não tem sido enviado
PqCh3R	Variável que contém a quantidade de ACKs dentro do canal 3 de retorno.

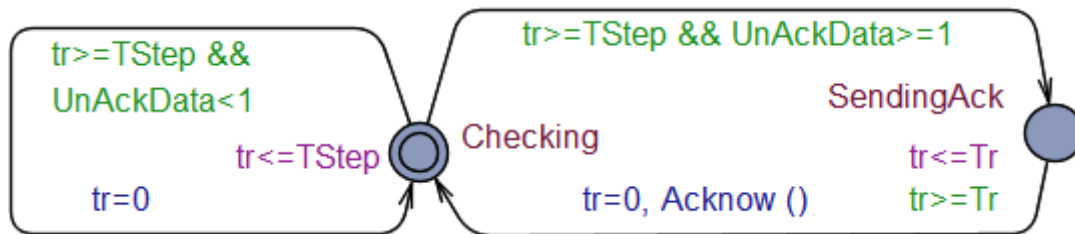


Figura I.6: Modelo do receptor.

Algoritmo I.4 Pseudocódigo para o autômato da Figura I.6.

Pseudocódigo para o autômato do receptor	Observações
<pre> Inicializa no estado Checking if (tr >= TStep) { if (UnAckP > 0) { Vai para o estado SendingAck espera Tr unidades de tempo e então retorna para o estado Checking, reinicia tr e chama a função Acknow } if (UnAckP < 1) { Faz uma transição para o mesmo estado para reiniciar tr} } else { espera até tr >= TStep ser verdadeiro } </pre>	<p>Transição do lado direito na Figura I.6</p> <p>Transição do lado esquerdo na Figura I.6</p> <p>Fim do bloco if (tr >= TStep)</p>

I.0.5 Modelo do canal 2 de ida

A Figura I.7 ilustra o autômato que modela o canal 2 de ida (localizado entre o roteador 1 e o roteador 2), o pseudocódigo desse modelo é dado no Algoritmo I.5, enquanto que as constantes, variáveis e funções utilizadas no modelo autômato temporizado são descritas na Tabela I.5.

Conforme pode ser observado na Figura I.7, esse autômato contém transições para um único estado que acontecem a cada TStep unidades de tempo. A transição da direita acontece cada vez que existe pelo menos um pacote no canal, ou seja, $PqCh2 > 0$ for verdadeiro e um pacote já

chegou no roteador 2, isto é, $TPCh2[0] \geq TpCh2$ também for verdadeiro. Onde $TPCh2 [0]$ é um cronômetro disparado no momento que o pacote que está no extremo do canal (chegando na fila do roteador 2) saiu do roteador 1. Nessa transição é reiniciado o relógio $tchf2$ e é chamada a função `EnqueueTwoF` a qual basicamente modela a saída de um pacote do respectivo canal e sua entrada na fila de ida do roteador 2.

As transições da esquerda ocorrem quando após um tempo $TStep$ não existir pacotes no canal ou se existir ainda não se propagaram até o roteador 2.

Tabela I.5: Descrição das variáveis, funções e constantes utilizadas no modelo do canal 2 de ida.

Nome	Descrição
$TStep$	Passo de tempo (resolução temporal) - valor adotado $=1\mu s$.
$tchf2$	Relógio que coordena as transições do estados Checking.
$TpCh2$	Limiar de tempo igual ao atraso de propagação no canal 2 (5 ms).
$TPCh2$	Arranjo de Relógios que contenham o tempo de de propagação de cada um dos pacotes que já saíram do roteador 1 mas ainda não chegaram no roteador 2, ou seja, o tempo transcorrido desde que cada pacote saiu do roteador 1 até o instante atual.
<code>EnqueueTwoF</code>	Função chamada cada vez que um pacote chega na fila de ida do roteador 2. Basicamente incrementa a variável $QlenF2$ em um pacote e decrementa a variável $PqCh2$ em um pacote.
$PqCh2$	Variável que conta a quantidade de pacotes dentro do canal 2 de ida.
$QlenF2$	Variável que contém a quantidade de pacotes dentro da fila de ida no roteador 2.

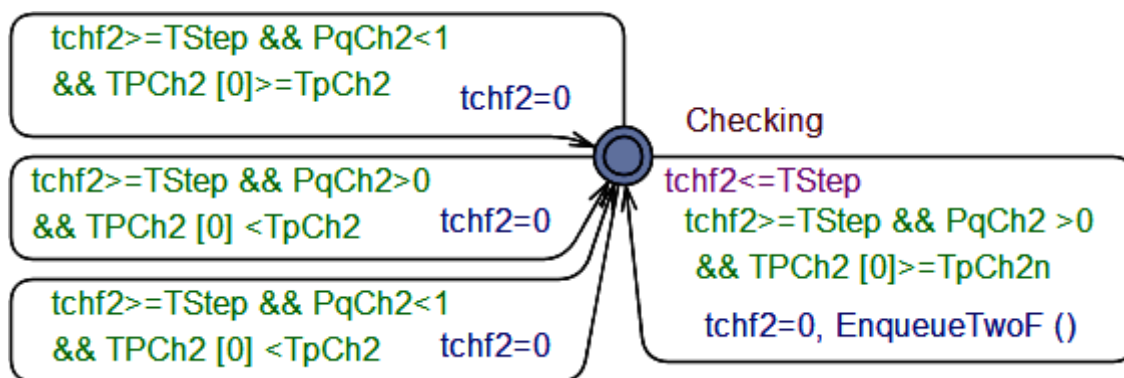


Figura I.7: Modelo do canal 2 de ida.

Algoritmo I.5 Pseudocódigo para o autômato da Figura I.7.

Pseudocódigo para o autômato do canal 2 de ida	Observações
Inicializa no estado Checking	
if ($tchf2 \geq TStep$) {	
if ($PqCh2 > 0$ and $TPCh2[0] \geq TpCh2$)	
{ Faz uma transição para o mesmo estado reinicia $tchf2$ e chama a função EnqueueTwoF }	Transição do lado direito na Figura I.7
if ($PqCh2 < 1$ or $TPCh2[0] < TpCh2$)	
{ Faz uma transição para o mesmo estado para reiniciar $tchf2$ }	Transições do lado esquerdo na Figura I.7
}	Fim do bloco ($tchf2 \geq TStep$)
else {espera até $tchf2 \geq TStep$ ser verdadeiro }	

I.0.6 Modelo da fila no caminho de ida no roteador 2

A Figura I.8 ilustra o autômato que modela a fila do caminho de ida no roteador 2, o qual consiste de dois estados, denominados **Checking** e **Service**, o pseudocódigo para este autômato é dado no Algoritmo I.6, enquanto que as constantes, variáveis e funções utilizadas no modelo autômato temporizado são descritas na Tabela I.6.

O autômato inicia no estado **Checking** e permanece nele durante $TStep$ unidades de tempo. Logo, se tiver pacotes na fila, ou seja, se $QlenF2 \geq 1$ for verdadeiro o autômato migra para o estado **Service**, conforme ilustra a seta inferior direita na Figura I.8. No estado **Service** o autômato permanece por $TqTwoF$ unidades de tempo, equivalente ao tempo necessário para colocar um pacote de 1 kbyte em um canal de 10 Mbps (canal 3). Depois, o autômato retorna para o estado **Checking** e nesta transição de retorno chama a função **DequeueTwoF**, e reinicia seu relógio $tq2f$ conforme ilustra a seta superior para a esquerda na Figura I.8.

Se estando no estado **Checking** passar $TStep$ unidades de tempo, ou seja, $tq2f \geq TStep$ for verdadeiro, e não houver pacotes na fila, isto é, $QlenF2 < 1$ também for verdadeiro, então o autômato faz uma transição para esse mesmo estado e reinicia o relógio $tq2f$ conforme ilustra a seta esquerda da Figura I.8.

Tabela I.6: Descrição das variáveis, funções e constantes utilizadas no modelo da fila de ida no roteador 2.

Nome	Descrição
$TStep$	Passo de tempo (resolução temporal) - valor adotado $=1\mu s$.
$tq2f$	Relógio que coordena as transições dos estados Checking e Service .
$TqTwoF$	Constante igual ao tempo em μs requerido para colocar um pacote de 1 kbyte em um canal com capacidade 10 Mbps (canal 3).
DequeueTwoF	Função chamada cada vez que um pacote sai da fila de ida do roteador 2. Basicamente decrementa a variável $QlenF2$ em um pacote e incrementa a variável $PqCh3$ em um pacote.
$QlenF2$	Variável que contém a quantidade de pacotes dentro da fila de ida no roteador 2.
$PqCh3$	Variável que contém a quantidade de pacotes dentro do canal 3 de ida.

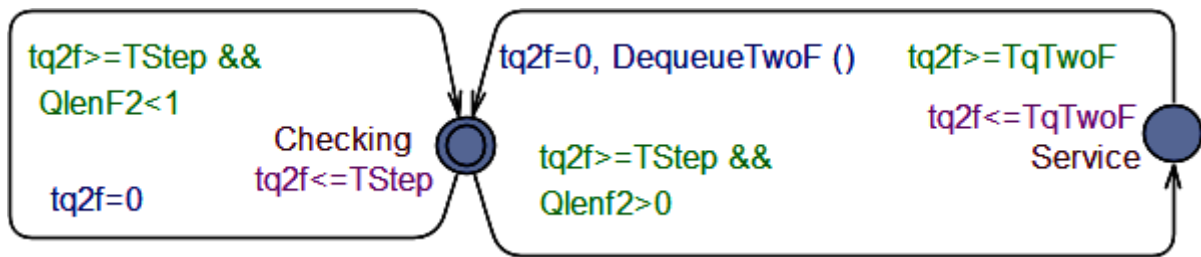


Figura I.8: Modelo da fila de dados no caminho de ida, no roteador 2.

Algoritmo I.6 Pseudocódigo para o autômato da Figura I.8.

Pseudocódigo do autômato que modela a fila no caminho de ida no roteador 2	Observações
Inicializa no estado Checking if (tq2f >= TStep) { if (QlenF2 > 0) { vai para o estado Service espera TqTwoF unidades de tempo e então retorna para o estado Checking, reinicia tq2f e chama a função DequeueTwoF } if (QlenF2 < 1) { Faz uma transição para o mesmo estado para reiniciar tq2f } } else { espera até tq2f >= TStep ser verdadeiro }	Transição do lado direito na Figura I.8 Transição do lado esquerdo na Figura I.8 Fim do bloco (tq2f >= TStep)

I.0.7 Modelo do canal 3 de ida

A Figura I.9 ilustra o autômato que modela o canal 3 de ida (localizado entre o roteador 2 e o receptor), o pseudocódigo desse modelo é dado no Algoritmo I.7, enquanto que as constantes, variáveis e funções utilizadas no modelo autômato temporizado são descritas na Tabela I.7.

Conforme pode ser observado na Figura I.9, esse autômato contém transições para um único estado que acontecem a cada TStep unidades de tempo, a transição da direita acontece cada vez que existe pelo menos um pacote no canal, ou seja, PqCh3 > 0 for verdadeiro e um pacote já chegou no receptor, isto é, TPCh3[0] >= TpCh3 também for verdadeiro. Onde TPCh3 [0] é um cronômetro disparado no momento que o pacote que está no extremo do canal (chegando no receptor) saiu do roteador 2. Nessa transição é reiniciado o relógio tchf3 e é chamada a função DeliverData, a qual basicamente modela a saída de um pacote do respectivo canal e sua chegada no receptor.

Tabela I.7: Descrição das variáveis, funções e constantes utilizadas no modelo do canal 3 de ida.

Nome	Descrição
TStep	Passo de tempo (resolução temporal) - valor adotado = $1\mu s$.
tchf3	Relógio que coordena as transições do estado Checking.
TpCh3	Limiar de tempo igual ao atraso de propagação no canal 3 (33 ms).
TPCh3	Arranjo de Relógios que contenham o tempo de de propagação de cada um dos pacotes que já saíram do roteador 2 mas ainda não chegaram no receptor, ou seja, o tempo transcorrido desde que cada pacote saiu do roteador 2 até o instante atual.
DeliverData	Função chamada cada vez que um pacote chega no receptor. Basicamente incrementa a variável UnAckP em um pacote e decrementa a variável PqCh3 em um pacote.
PqCh3	Variável que contém a quantidade de pacotes dentro do canal 3 de ida.
UnAckP	Variável que contém a quantidade de pacotes que já chegaram no receptor e cujo ACK ainda não tem sido enviado

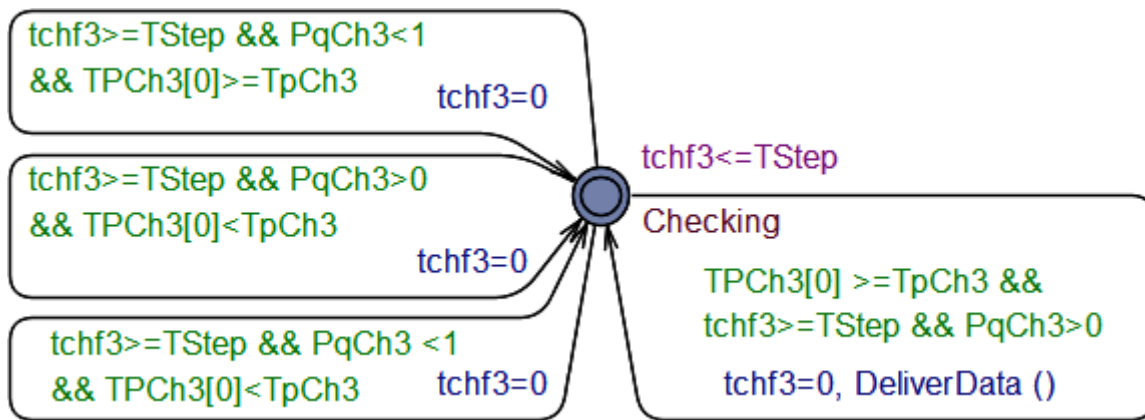


Figura I.9: Modelo do canal 3 de ida.

Algoritmo I.7 Pseudocódigo para o autômato da Figura I.9.

Pseudocódigo para o autômato do canal 3 de ida

```

Inicializa no estado Checking
if (tchf3 >= TStep) {
  if (PqCh3 > 0 and TPCh3[0] >= TpCh3)
  { Faz uma transição para o mesmo estado para reiniciar tchf3 e
  chama a função DeliverData}
  if (PqCh3 < 1 or TPCh3 [0] < TpCh3)
  { Faz uma transição para o mesmo estado para reiniciar tchf3 }
}
else {espera até tchf3 >= TStep ser verdadeiro }
    
```

Observações

Transição do lado direito na Figura I.9

Transições do lado esquerdo na Figura I.9

Fim do bloco ($tchf3 \geq TStep$)

As transições da esquerda ocorrem quando após um tempo TStep não existir pacotes no canal ou se existir ainda não se propagaram até o receptor.

I.0.8 Modelo do Receptor

Figura I.10 ilustra o autômato que modela o receptor, o pseudocódigo desse modelo é dado no Algoritmo I.8, enquanto que as constantes, variáveis e funções utilizadas no modelo autômato temporizado são descritas na Tabela I.8.

Conforme pode ser observado na Figura I.10 esse autômato contém dois estados denominados **Checking** e **SendingAck**. O autômato começa no estado **Checking** e cada vez que chegar um pacote, isto é, quando $UnAckP > 0$ for verdadeiro o autômato migra do estado **Checking** para o estado **SendingAck** onde permanece por T_s unidades de tempo. Depois, o autômato retorna para o estado **Checking**, e nessa transição de retorno chama a função **AckNow**.

Se o automato estiver no estado **Checking** e após um tempo T_{Step} não houver novos pacotes de dados cujos ACKs ainda não foram enviados, ou seja, $UnAckP < 1$ for verdadeiro, então o autômato faz uma transição para esse mesmo estado e reinicia seu relógio tr , conforme ilustra a seta da esquerda na Figura I.10.

Tabela I.8: Descrição das variáveis, funções e constantes utilizadas no modelo do receptor.

Nome	Descrição
T_{Step}	Passo de tempo (resolução temporal) - valor adotado $= 1\mu s$.
tr	Relógio que coordena as transições dos estados Checking e SendingAck .
T_r	Limiar de tempo requerido para colocar um ACK de 1 kbyte em um canal com capacidade 10 Mbps (canal 3).
$UnAckP$	Variável que contém a quantidade de pacotes que chegaram ao receptor e cujos ACKs ainda não foram enviados.
AckNow	Função chamada cada vez que o receptor envia um ACK. Basicamente calcula a sequência do ACK a ser enviado e decrementa a variável $UnAckP$ em um pacote e incrementa a variável $PqCh3R$ em um pacote.
$UnAckP$	Variável que contém a quantidade de pacotes que já chegaram no receptor e cujo ACK ainda não tem sido enviado
$PqCh3R$	Variável que contém a quantidade de ACKs dentro do canal 3 de retorno.

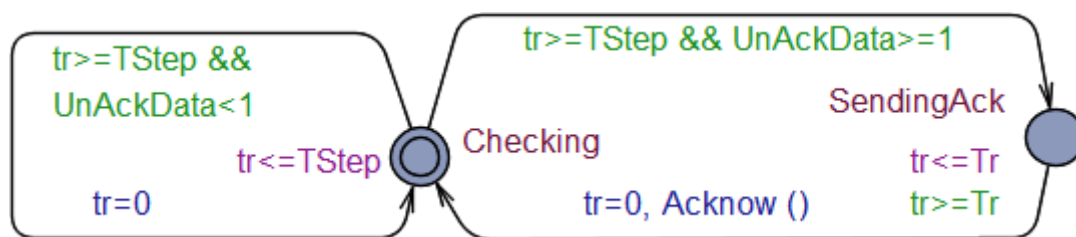


Figura I.10: Modelo do receptor.

Algoritmo I.8 Pseudocódigo para o autômato da Figura I.10.

Pseudocódigo para o autômato do receptor	Observações
Inicializa no estado <code>Checking</code>	
if (<code>tr</code> \geq <code>TStep</code>) {	
if (<code>UnAckP</code> $>$ 0) { Vai para o estado <code>SendingAck</code> espera <code>Tr</code> unidades de tempo e então retorna para o estado <code>Checking</code> , reinicia <code>tr</code> e chama a função <code>Acknow</code> }	Transição do lado direito na Figura I.10
if (<code>UnAckP</code> $<$ 1) { Faz uma transição para o mesmo estado para reiniciar <code>tr</code> }	Transição do lado esquerdo na Figura I.10
}	Fim do bloco if (<code>tr</code> \geq <code>TStep</code>)
else { espera até <code>tr</code> \geq <code>TStep</code> ser verdadeiro }	

I.0.9 Modelo do canal 3 de retorno

A Figura I.11 ilustra o autômato que modela o canal 3 de retorno (localizado entre o receptor e o roteador 2), o pseudocódigo desse modelo é dado no Algoritmo I.9, enquanto que as constantes, variáveis e funções utilizadas no modelo autômato temporizado são descritas na Tabela I.9.

Conforme pode ser observado na Figura I.11, esse autômato contém transições para um único estado que acontecem a cada `TStep` unidades de tempo. A transição da direita acontece cada vez que existe pelo menos um pacote de ACK no canal, ou seja, `PqCh3R` $>$ 0 for verdadeiro e um pacote de ACK já chegou no roteador 2, ou seja, `TPCh3R [0]` \geq `TpCh3` também for verdadeiro. Onde `TPCh3R [0]` é um cronômetro disparado no momento em que o pacote de ACK que está no extremo do canal (chegando no roteador 2) saiu do receptor. Nessa transição é reiniciado o relógio `tchr3` e é chamada a função `EnqueueTwoR` a qual basicamente modela a saída de um pacote do respectivo canal e sua chegada na fila de retorno no roteador 2.

As transições da esquerda ocorrem quando após um tempo `TStep` não existir pacotes de ACKs no canal ou se existir ainda não se propagaram até o roteador 2.

Tabela I.9: Descrição das variáveis, funções e constantes utilizadas no modelo do canal 3 de retorno.

Nome	Descrição
<code>TStep</code>	Passo de tempo (resolução temporal) - valor adotado $=1\mu s$.
<code>tchr3</code>	Relógio que coordena as transições do estado <code>Checking</code> .
<code>TpCh3</code>	Limiar de tempo igual ao atraso de propagação no canal 3 (33 ms).
<code>TPCh3R</code>	Arranjo de Relógios que contenham o tempo de de propagação de cada um dos ACK que já saíram do receptor mas ainda não chegaram no roteador 2, ou seja, o tempo transcorrido desde que cada ACK saiu do receptor até o instante atual.
<code>EnqueueTwoR</code>	Função chamada cada vez que um pacote de ACK chega no roteador 2. Basicamente incrementa a variável <code>QlenR2</code> em um pacote e decrementa a variável <code>PqCh3R</code> em um pacote.
<code>PqCh3R</code>	Variável que contém a quantidade de ACKs dentro do canal 3 de retorno.
<code>QlenR2</code>	Variável que contém a quantidade de pacotes de ACKs dentro da fila de retorno no roteador 2.

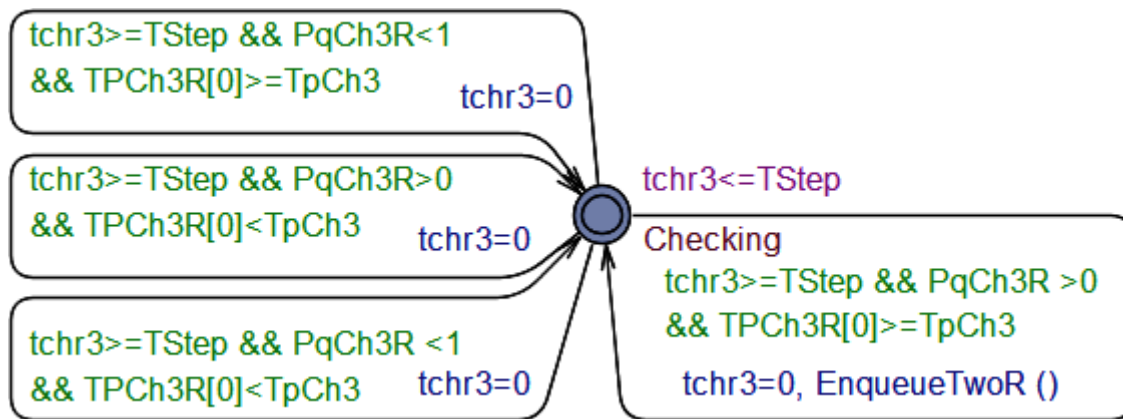


Figura I.11: Modelo do canal 3 de retorno.

Algoritmo I.9 Pseudocódigo para o autômato da Figura I.11.

Pseudocódigo para o autômato do canal 3 de retorno	Observações
<pre> Inicializa no estado Checking if (tchr3 >= TStep) { if (PqCh3R > 0 and TPCh3R [0] >= TpCh3) { Faz uma transição para o mesmo estado para reiniciar tchr3 e chama a função EnqueueTwoR} if (PqCh3R < 1 or TPCh3R [0] < TpCh3) { Faz uma transição para o mesmo estado para reiniciar tchr3 } } else {espera até tchr3 >= TStep ser verdadeiro } </pre>	<p>Transição do lado direito na Figura I.11</p> <p>Transições do lado esquerdo na Figura I.11</p> <p>Fim do bloco (tchr3 >= TStep)</p>

I.0.10 Modelo da fila no caminho de retorno no roteador 2

A Figura I.12 ilustra o autômato que modela a fila do caminho de retorno no roteador 2, o qual consiste de dois estados, denominados *Checking* e *Service*. O pseudocódigo para este autômato é dado no Algoritmo I.10, enquanto que as constantes, variáveis e funções utilizadas no modelo autômato temporizado são descritas na Tabela I.10.

O autômato inicia no estado *Checking* e permanece nele durante $TStep$ unidades de tempo, logo, se tiver pacotes de ACKs na fila, ou seja, se $QlenR2 \geq 1$ for verdadeiro, o autômato migra para o estado *Service*, conforme ilustra a seta inferior esquerda na Figura I.12. No estado *Service* o autômato permanece por $TqTwoR$ unidades de tempo, equivalente ao tempo necessário para colocar um pacote de 1 kbyte em um canal de 1,5 Mbps (canal 2). Depois, o autômato retorna para o estado *Checking* e nesta transição de retorno chama a função *DequeueTwoR* e reinicia seu relógio $tq2r$ conforme ilustra a seta superior para a direita na Figura I.12.

Se estando no estado *Checking* passar $TStep$ unidades de tempo, ou seja, $tq2r \geq TStep$ for verdadeiro, e não houver pacotes na fila, ou seja, $QlenR2 < 1$ também for verdadeiro, então o autômato faz uma transição para esse mesmo estado e reinicia o relógio $tq2r$ conforme ilustra a seta da direita na Figura I.12.

Tabela I.10: Descrição das variáveis, funções e constantes utilizadas no modelo da fila de retorno no roteador 2.

Nome	Descrição
TStep	Passo de tempo (resolução temporal) - valor adotado = $1\mu s$.
tq2r	Relógio que coordena as transições dos estados Checking e Service.
TqTwoR	Constante igual ao tempo em μs requerido para colocar um pacote de 1 kbyte em um canal com capacidade 1, 5 Mbps (canal 2).
DequeueTwoR	Função chamada cada vez que um pacote sai da fila de retorno do roteador 2. Basicamente decrementa a variável QlenR2 em um pacote e incrementa a variável PqCh2R em um pacote.
QlenR2	Variável que contém a quantidade de pacotes de ACKs dentro da fila de retorno no roteador 2.
PqCh2R	Variável que contém a quantidade de pacotes dentro do canal 2 de retorno.

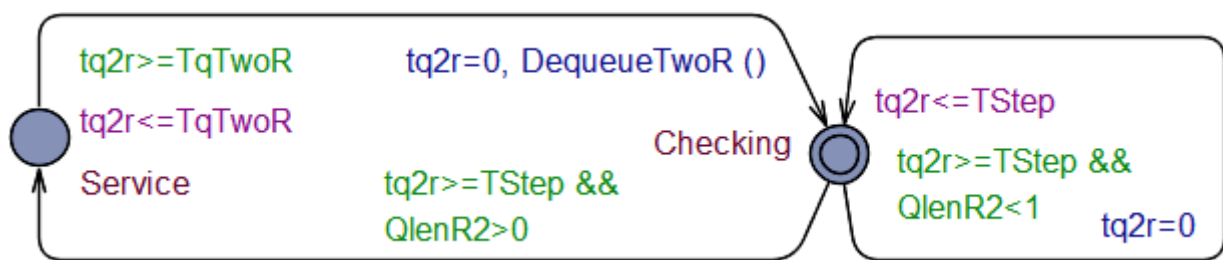


Figura I.12: Modelo da fila no caminho de retorno, no roteador 2.

Algoritmo I.10 Pseudocódigo para o autômato da Figura I.12.

Pseudocódigo do autômato que modela a fila no caminho de retorno no roteador 2	Observações
Inicializa no estado Checking if (tq2r >= TStep) { if (QlenR2 > 0) { vai para o estado Service espera TqTwoR unidades de tempo e então retorna para o estado Checking, reinicia tq2r e chama a função DequeueTwoR } if (QlenR2 < 1) { Faz uma transição para o mesmo estado para reiniciar tq2r } } else { espera até tq2r >= TStep ser verdadeiro }	Transição do lado esquerdo na Figura I.12 Transição do lado direito na Figura I.12 Fim do bloco (tq2r >= TStep)

I.0.11 Modelo do canal 2 de retorno

A Figura I.13 ilustra o autômato que modela o canal 2 de retorno (localizado entre o roteador 2 e o roteador 1). O pseudocódigo desse modelo é dado no Algoritmo I.11, enquanto que as constantes, variáveis e funções utilizadas no modelo autômato temporizado são descritas na Tabela I.11.

Conforme pode ser observado na Figura I.13, esse autômato contém transições para um único estado que acontecem a cada TStep unidades de tempo. A transição da direita acontece cada vez que existe pelo menos um pacote de ACK no canal, ou seja, PqCh2R > 0 for verdadeiro e um pacote

já chegou no roteador 1, isto é, $TPCh2R [0] \geq TpCh2$ também for verdadeiro. Onde $TPCh2R [0]$ é um cronômetro disparado no momento em que o pacote de ACK que está no extremo do canal (chegando no roteador 1) saiu do roteador 2. Nessa última transição é reiniciado o relógio $tchr2$ e é chamada a função `EnqueueOneR` a qual basicamente modela a saída de um pacote do respectivo canal e sua chegada na fila de retorno no roteador 1.

As transições da esquerda ocorrem quando após um tempo $TStep$ não existir pacotes de ACKs no canal ou se existir ainda não se propagaram até o roteador 1.

Tabela I.11: Descrição das variáveis, funções e constantes utilizadas no modelo do canal 2 de retorno.

Nome	Descrição
$TStep$	Passo de tempo (resolução temporal) - valor adotado $=1\mu s$.
$tchr2$	Relógio que coordena as transições do estado <code>Checking</code> .
$TpCh2$	Limiar de tempo igual ao atraso de propagação no canal 2 (5 ms).
$TPCh2R$	Arranjo de Relógios que contenham o tempo de de propagação de cada um dos ACK que já saíram do roteador 2 mas ainda não chegaram no roteador 1, ou seja, o tempo transcorrido desde que cada ACK saiu do roteador 2 até o instante atual.
<code>EnqueueOneR</code>	Função chamada cada vez que um pacote de ACK chega no roteador 1. Basicamente incrementa a variável $QlenR1$ em um pacote e decrementa a variável $PqCh2R$ em um pacote.
$QlenR1$	Variável que contém a quantidade de pacotes de ACKs dentro da fila de retorno no roteador 1.

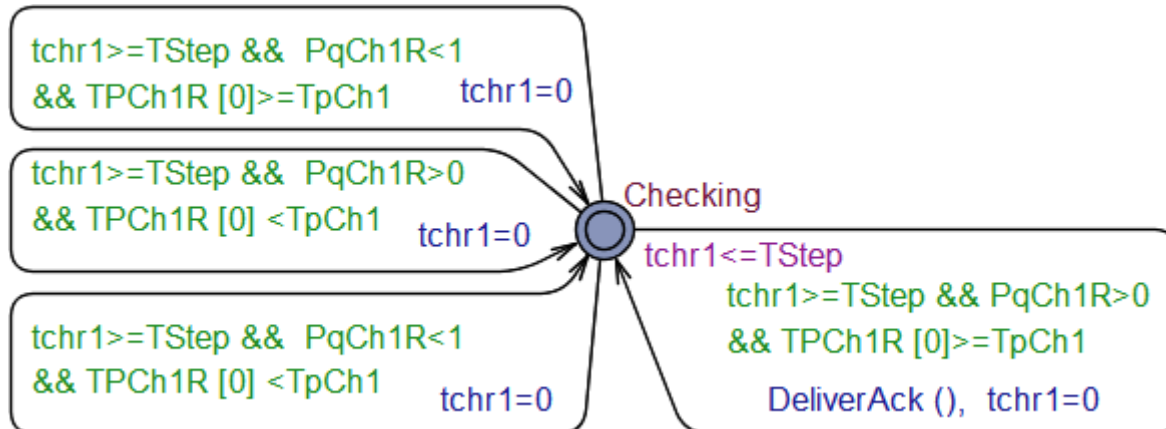


Figura I.13: Modelo do canal 2 de retorno.

Algoritmo I.11 Pseudocódigo para o autômato da Figura I.13.

Pseudocódigo para o autômato do canal 2 de retorno	Observações
Inicializa no estado Checking	
if ($tchr2 \geq TStep$) {	
if ($PqCh2R > 0$ and $TPCh2R [0] \geq TpCh2$)	Transição do lado direito na Figura I.13
{ Faz uma transição para o mesmo estado para reiniciar $tchr2$ e chama a função EnqueueOneR }	
if ($PqCh2R < 1$ or $TPCh2R [0] < TpCh2$)	
{ Faz uma transição para o mesmo estado para reiniciar $tchr2$ }	Transições do lado esquerdo na Figura I.13
}	
}	Fim do bloco ($tchr2 \geq TStep$)
else {espera até $tchr2 \geq TStep$ ser verdadeiro }	

I.0.12 Modelo da fila no caminho de retorno no roteador 1

A Figura I.14 ilustra o autômato que modela a fila do caminho de retorno no roteador 2, o qual consiste de dois estados, denominados **Checking** e **Service**, o pseudocódigo para este autômato é dado no Algoritmo I.12, enquanto que as constantes, variáveis e funções utilizadas no modelo autômato temporizado são descritas na Tabela I.12.

O autômato inicia no estado **Checking** e permanece nele durante $TStep$ unidades de tempo. Logo, se tiver pacotes na fila, ou seja, se $QlenR1 \geq 1$ for verdadeiro, o autômato migra para o estado **Service**, conforme ilustra a seta inferior para a esquerda na Figura I.14. No estado **Service** o autômato permanece por $TqOneR$ unidades de tempo, equivalente ao tempo necessário para colocar um pacote de 1 kbyte em um canal de 10 Mbps (canal 1). Depois, o autômato retorna para o estado **Checking** e nesta transição de retorno chama a função **DequeueOneR**, e reinicia seu relógio $tq1r$ conforme ilustra a seta superior para a esquerda na Figura I.14.

Se estando no estado **Checking** passar $TStep$ unidades de tempo, ou seja, $tq1r \geq TStep$ for verdadeiro, e não houver pacotes na fila, ou seja, $QlenR1 < 1$ também for verdadeiro, então o autômato faz uma transição para esse mesmo estado e reinicia o relógio $tq1r$ conforme ilustra a seta da direita na Figura I.14.

Tabela I.12: Descrição das variáveis, funções e constantes utilizadas no modelo da fila de retorno no roteador 1.

Nome	Descrição
$TStep$	Passo de tempo (resolução temporal) - valor adotado $=1\mu s$.
$tq1r$	Relógio que coordena as transições dos estados Checking e Service .
$TqOneR$	Constante igual ao tempo em μs requerido para colocar um pacote de 1 kbyte em um canal com capacidade 10 Mbps (canal 1).
DequeueOneR	Função chamada cada vez que um pacote sai da fila de retorno do roteador 1. Basicamente decrementa a variável $QlenR1$ em um pacote e incrementa a variável $PqCh1R$ em um pacote.
$QlenR1$	Variável que contém a quantidade de pacotes de ACKs dentro da fila de retorno no roteador 1.
$PqCh1R$	Variável que contém a quantidade de pacotes dentro do canal 1 de retorno.

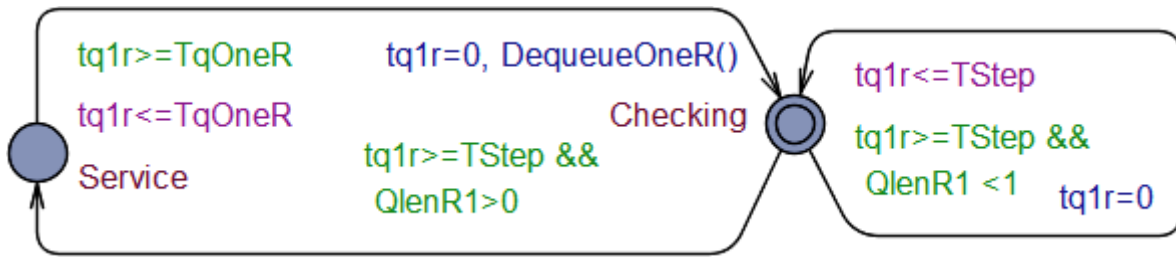


Figura I.14: Modelo da fila de dados no caminho de retorno, no roteador 1.

Algoritmo I.12 Pseudocódigo para o autômato da Figura I.14.

Pseudocódigo do autômato que modela a fila no caminho de retorno no roteador 1	Observações
<pre> Inicializa no estado Checking if (tq1r >= TStep) { if (QlenR1 > 0) { vai para o estado Service espera TqOneR unidades de tempo e então retorna para o estado Checking, reinicia tq1r e chama a função DequeueOneR } if (QlenR1 < 1) { Faz uma transição para o mesmo estado para reiniciar tq1r } } else { espera até tq1r >= TStep ser verdadeiro } </pre>	<p>Transição do lado esquerdo na Figura I.14</p> <p>Transição do lado direito na Figura I.14</p> <p>Fim do bloco (tq1r >= TStep)</p>

I.0.13 Modelo do canal 1 de retorno

A Figura I.15 ilustra o autômato que modela o canal 1 de retorno (localizado entre o roteador 1 e o transmissor). O pseudocódigo desse modelo é dado no Algoritmo I.13, enquanto que as constantes, variáveis e funções utilizadas no modelo autômato temporizado são descritas na Tabela I.13.

Conforme pode ser observado na Figura I.15, esse autômato contém transições para um único estado que acontecem a cada $TStep$ unidades de tempo. A transição da direita acontece cada vez que existe pelo menos um pacote de ACK no canal, ou seja, $PqCh1R > 0$ for verdadeiro, e um pacote de ACK já chegou no roteador 1, isto é, $TPCh1R [0] \geq TpCh1$ também for verdadeiro. Onde $TPCh1R [0]$ é um cronômetro disparado no momento em que o pacote que está no extremo do canal (chegando no transmissor) saiu do roteador 1.

Nessa última transição é reiniciado o relógio $tchr1$ e é chamada a função `DeliverAck` a qual basicamente modela a saída de um pacote do respectivo canal e sua chegada na fila de retorno no transmissor, e também é responsável por executar o algoritmo TCP-Reno (descrito brevemente na subseção 2.4.1). Assim, a função `DeliverAck` calcula e muda as fases do algoritmo TCP-Reno de acordo com a recepção dos ACKs e suas respectivas sequências, e ainda atualiza o valor de W , e de acordo com o incremento em W realiza um incremento na variável Sd .

As transições da esquerda ocorrem quando após um tempo $TStep$ não existir pacotes no canal ou se existir ainda não se propagaram até o transmissor.

Tabela I.13: Descrição das variáveis, funções e constantes utilizadas no modelo do canal 1 de retorno.

Nome	Descrição
TStep	Passo de tempo (resolução temporal) - valor adotado = $1 \mu s$.
tchr1	Relógio que coordena as transições do estado Checking.
TpCh1	Limiar de tempo igual ao atraso de propagação no canal 1 (2 ms).
TPCh1R	Arranjo de Relógios que contenham o tempo de de propagação de cada um dos ACK que já saíram do roteador 1 mas ainda não chegaram no transmissor, ou seja, o tempo transcorrido desde que cada ACK saiu do roteador 1 até o instante atual.
DeliverAck	Função chamada cada vez que um pacote de ACK chega no transmissor. Basicamente executa o protocolo TCP-Reno atualiza W e calcula um incremento na variável Sd e ainda decrementa a variável $PqCh1R$ em um pacote.
Sd	Variável que contém a quantidade de pacotes que o transmissor TCP pode enviar, incrementada de acordo com o algoritmo TCP-Reno que faz a atualização cada vez que recebe um ACK, e decrementada cada vez que o transmissor envia um pacote.
PqCh1R	Variável que contém a quantidade de pacotes dentro do canal 1 de retorno.

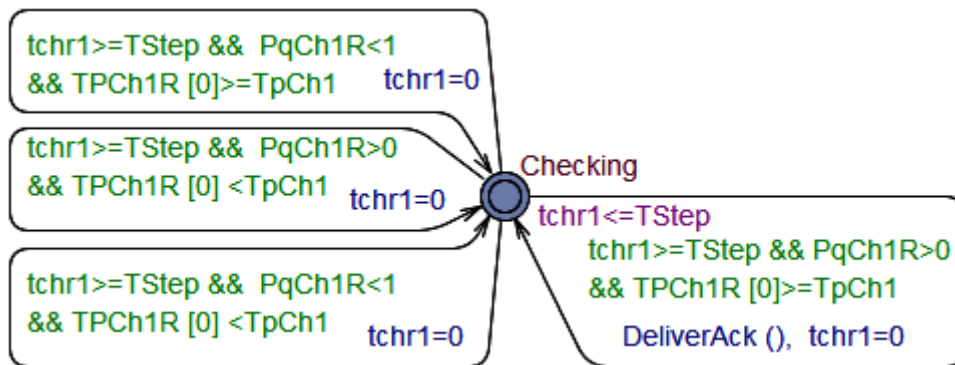


Figura I.15: Modelo do canal 1 de retorno.

Algoritmo I.13 Pseudocódigo para o autômato da Figura I.15.

Pseudocódigo para o autômato do canal 1 de retorno	Observações
<pre> Inicializa no estado Checking if (tchr1 >= TStep) { if (PqCh1R > 0 and TPCh1R [0] >= TpCh1) { Faz uma transição para o mesmo estado para reiniciar tchr1 e chama a função DeliverAck} if (PqCh1R < 1 or TPCh1R [0] < TpCh1) { Faz uma transição para o mesmo estado para reiniciar tchr1 } } else {espera até tchr2 >= TStep ser verdadeiro } </pre>	<p>Transição do lado direito na Figura I.15</p> <p>Transições do lado esquerdo na Figura I.15</p> <p>Fim do bloco (tchr1 >=TStep)</p>

II. ANEXO: MODELOS DE TÉCNICAS AQM EM UPPAAL

No Anexo I, foi apresentado a modelagem de uma topologia de rede daisy-chain mediante um sistema de autômatos temporizados realizado no UPPAAL, no qual um transmissor TCP transmite dados a um receptor mediante uma topologia de rede composta por dois roteadores nos quais funcionam com a técnica AQM básica conhecida como Drop Tail (na qual pacotes são descartados se e somente se a fila do *buffer* estiver cheia). No modelo dessa topologia de rede, transmissor, receptor, canais e roteadores foram modelados mediante autômatos temporizados e toda a topologia de rede de comunicação funciona como um sistema de tais autômatos em paralelo.

Neste anexo, será apresentado as modificações que foram feitas no modelo dado na Seção 3.3 para modelar as técnicas AQM RED, CoDel, PIE, ENCN e ANCE nesta mesma topologia de rede de comunicação básica. Em todas essas modelagens aqui apresentadas foi considerado que o Bit ECN está habilitado, assim, o roteador coloca uma marca no cabeçalho IP ao invés de descartar o pacote. Logo, quando o receptor receber esse pacote de dados marcado, enviará um ACK especial para o transmissor informando-o acerca do congestionamento no caminho. Quando finalmente o transmissor receber esse ACK especial reduzirá sua janela de congestionamento (*cwnd*) pela metade antes da fila no roteador encher. Porém, mesmo que receba pacotes marcados consecutivos o transmissor somente reduzirá sua janela uma vez por RTT. Nesta seção serão apresentados maiores detalhes acerca da modelagem das técnicas AQM RED, CoDel e PIE, realizadas no UPPAAL utilizando a topologia de rede daisy-chain ilustrada na Figura 3.3.

II.1 Modelo do algoritmo RED no UPPAAL

Para modelar no UPPAAL a implementação da técnica AQM RED na topologia de rede de comunicação daisy-chain da Figura 3.3, foi utilizado o mesmo modelo de autômatos temporizados apresentado na Seção 3.3 mas foram modificados os modelos de alguns canais de transmissão. Assim sendo, a Figura II.1 ilustra o modelo autômato que modela o canal 1 de ida (situado entre o transmissor e o roteador 1). Conforme pode ser observado, esse autômato está composto por um estado e um ponto de ramificação (marcado com um círculo menor). Pontos de ramificação são utilizados para criar ramificações probabilísticas. A cada ramificação é atribuído um peso dado por um número inteiro constante e positivo. A probabilidade do autômato seguir um determinado caminho da ramificação é dada pelo quociente entre a taxa de seu peso e a soma de todos os pesos designado a cada caminho possível.

O pseudocódigo para este autômato é dado no Algoritmo II.1, enquanto que as constantes, variáveis e funções utilizadas no modelo autômato temporizado são descritas na Tabela II.1.

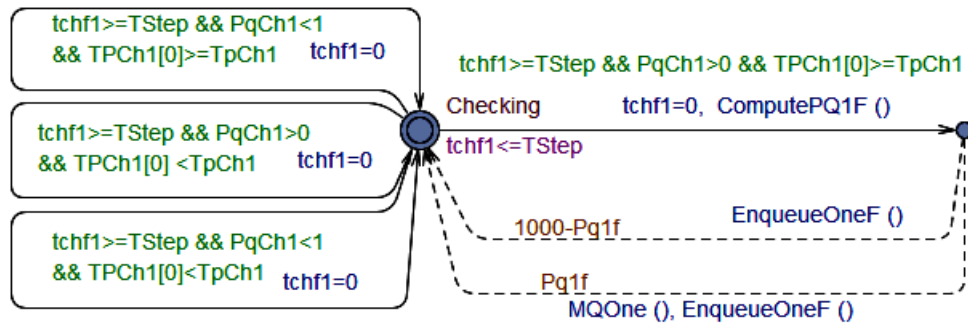


Figura II.1: Modelo autômato temporizado para o canal de ida situado entre o transmissor e o roteador 1 utilizando RED.

Tabela II.1: Descrição das variáveis, funções e constantes utilizadas no modelo do canal 1 de ida com a técnica AQM RED.

Nome	Descrição
TStep	Passo de tempo (resolução temporal) - valor adotado $= 1\mu s$.
tchf1	Relógio que coordena as transições do estado Checking.
TpCh1	Constante igual ao tempo de propagação no canal 1 (2 ms).
TPCh1	Arranjo de Relógios que contenham o tempo de de propagação de cada um dos pacotes que já saíram do transmissor mas ainda não chegaram no roteador 1, ou seja, o tempo transcorrido desde que cada pacote saiu do transmissor até o instante atual.
ComputePQ1F	Função chamada cada vez que um pacote chega à fila de ida do roteador 1. Basicamente calcula o tamanho médio da fila utilizando a Equação 2.6 e a probabilidade $Pq1f/1000$ de marcar o pacote que entra na fila do roteador utilizando a Equação (2.7).
Pq1f	Variável que contém a probabilidade de marcar um pacotes multiplicada por um fator de 1000.
MQOne	Função chamada com probabilidade $Pq1f/1000$ cada vez que um pacote chega à fila de ida do roteador 1. Basicamente coloca uma marca no bit ECN no cabeçalho do pacote.
EnqueueOneF	Função chamada cada vez que um pacote chega à fila de ida do roteador 1. Basicamente incrementa a variável $QlenF1$ em um pacote e decrementa a variável $PqCh1$ em um pacote.
PqCh1	Variável que contém a quantidade de pacotes dentro do canal 1 de ida.
QlenF1	Variável que contém a quantidade de pacotes dentro da fila de ida do roteador 1.

Algoritmo II.1 Pseudocódigo para o autômato da Figura ??.

Pseudocódigo para o autômato do canal 1 de ida com a técnica AQM RED.	Observações
<pre>Inicializa no estado Checking if (tchf1 >= TStep) { if (PqCh1 > 0 and TPCh1 [0] >= TpCh1) {Vai para o ponto de ramificação, e nessa transição reinicia tchf1 e chama a função ComputePQ1F. Com probabilidade (1000-Pq1f)/1000 retorna para o estado Checking pelo caminho superior, e nessa transição chama a função EnqueueOneF. Com probabilidade Pq1f/1000 retorna para o estado Checking pelo caminho inferior, e nessa transição chama as funções MQOne e EnqueueOneF. } if (PqCh1 < 1 or TPCh1 [0] < TpCh1) { Faz uma transição para o mesmo estado para reiniciar tchf1 } } else {espera até tchf1 >= TStep ser verdadeiro }</pre>	<p>Transição do lado direito na Figura II.1.</p> <p>fim do bloco if (PqCh1 > 0 and TPCh1 [0] >= TpCh1)</p> <p>Transições do lado esquerdo na Figura II.1.</p> <p>Fim do bloco (tchf1 >=TStep)</p>

Assim na Figura II.1 o autômato inicia no estado **Checking**, onde permanece por **TStep** unidades de tempo. Passado esse limiar de tempo, se existir pelo menos um pacote no canal, ou seja, $PqCh1 > 0$ for verdadeiro e um pacote já chegou no roteador 1, ou seja, $TProp[0] \geq TpCh1$ também for verdadeiro, o autômato migra para o ponto de ramificação situado à direita da Figura II.1, e nessa transição chama a função **ComputePQ1** a qual basicamente calcula o tamanho médio da fila utilizando a Equação (2.6). Em seguida calcula a probabilidade de marcar o pacote que entra na fila do roteador utilizando a Equação (2.7). Essa probabilidade é logo multiplicada por um fator de 1000 (essa multiplicação é feita para convertê-la em um número inteiro já que o UPPAAL não aceita números flutuantes como peso para suas ramificações) e representada pela variável **Pq1f**, a qual é utilizada para atribuir um peso aos dois caminhos possíveis que partem desde o ponto de ramificação até o retorno para o estado **Checking**.

Assim, no caminho inferior é designado o peso **Pq1f** e ao superior o peso **1000-Pq1f**. Se o autômato seguir o caminho inferior a função **MQOne** é chamada, a qual basicamente coloca uma marca no pacote. Logo é chamada a função **EnqueueOneF**, que basicamente modela a saída de um pacote deste canal e sua entrada na fila de dados do roteador 1. Caso o autômato siga o caminho superior da ramificação, então o pacote não será marcado e somente a função **EnqueueOneF** é chamada.

As transições da esquerda ocorrem quando após um tempo **TStep** ainda não existir pacotes no canal, ou de existir ainda não se propagaram até o roteador 1.

Os outros canais que chegam nos roteadores, ou seja, que chamam funções de enfileiramento de pacotes foram modelados de forma semelhante. E os resultados das simulações são apresentados na Seção 3.5.

II.2 Modelo do algoritmo CoDel no UPPAAL

Para modelar no UPPAAL a implementação da técnica AQM CoDel na topologia de rede de comunicação daisy-chain da Figura 3.3, ao modelo de autômatos temporizados apresentado na Seção 3.3 foram modificados os modelos das filas nos roteadores.

Assim sendo, a Figura II.2 ilustra o novo modelo autômato temporizado da fila no caminho de ida no roteador 1, no qual são enfileirados pacotes de dados que viajam do transmissor para o receptor. Conforme pode ser observado, esse modelo está composto por quatro estados denominados *Checking*, *Obs*, *TargetTime* e *Service*. Dois deles (*Obs* e *TargetTime*) se encontram marcados com a letra *U*, isso significa que são estados urgentes, ou seja, o sistema não gasta tempo nestes estados (entra e sai em zero unidades de tempo).

O pseudocódigo para este autômato é dado no Algoritmo II.2, enquanto que as constantes, variáveis e funções utilizadas no modelo autômato temporizado são descritas na Tabela II.2.

Tabela II.2: Descrição das variáveis, funções e constantes utilizadas no modelo da fila de ida no roteador 1 com a técnica AQM CoDel.

Nome	Descrição
TStep	Passo de tempo (resolução temporal) - valor adotado $=1\mu s$.
tq1f	Relógio que coordena as transições dos estados <i>Checking</i> e <i>Service</i> .
TqOneF	Constante igual ao tempo em μs requerido para colocar um pacote de 1 kbyte em um canal com capacidade 1,5 Mbps (canal 2).
tobsQ1f	Relógio que mede cada período de observação.
TObsQ1f	Variável que contém o valor do período de observação .
TminQ1f	Variável igual ao mínimo tempo de espera na fila observado.
CompTminQ1f	Função chamada cada vez que um pacote sai da fila de ida do roteador 1. Basicamente calcula o mínimo tempo de espera (TminQ1f) para cada período de observação (TObsQ1f).
Target	Constante, igual ao valor alvo do mínimo tempo de espera para cada período de observação. Ou seja, 5 ms.
DropQ1f	Variável que contém a quantidade intervalos de observação consecutivos nos quais pacotes foram marcados. É reinicializada cada vez que ao final de um intervalo de observação o último pacote não for marcado. E é incrementada em uma unidade cada vez que ao final de um intervalo de observação o último pacote for marcado.
MQ1f	Função que basicamente coloca uma marca no bit ECN no cabeçalho do pacote.
DequeueOneF	Função chamada cada vez que um pacote sai da fila de ida do roteador 1. Basicamente decrementa QlenF1 em um pacote e incrementa a variável PqCh2 em um pacote.
QlenF1	Variável que contém a quantidade de pacotes dentro da fila de ida do roteador 1.
PqCh2	Variável que conta a quantidade de pacotes dentro do canal 2 de ida.

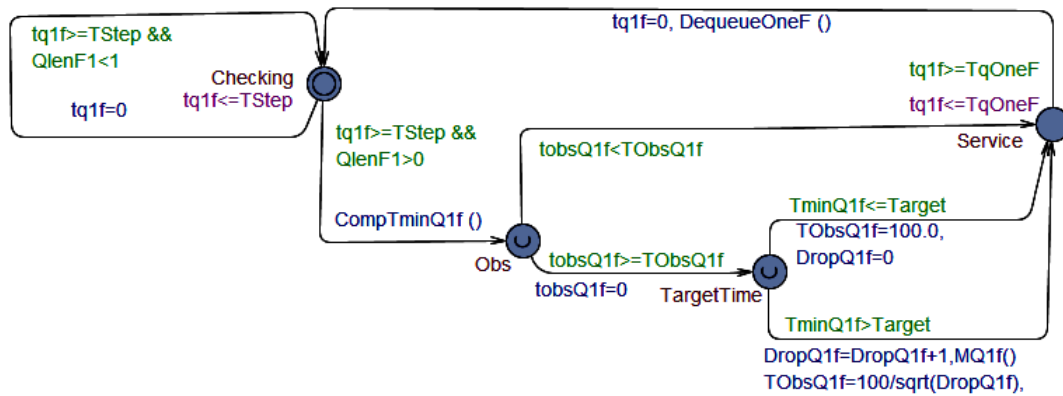


Figura II.2: Modelo autômato temporizado para a fila canal de ida no roteador 1 utilizando CoDel.

Algoritmo II.2 Pseudocódigo para o autômato da Figura II.2.

Pseudocódigo do autômato que modela a fila no caminho de ida no roteador 1 com a técnica AQM CoDel.

Inicializa no estado Checking

if ($tq1f \geq TStep$) {

if ($QlenF1 > 0$) { vai para o estado Obs.

if ($tobsQ1f \geq TObsQ1f$) { vai para o estado TargetTime.

if ($TminQ1f \leq Target$) { vai para o estado Service, e nessa transição faz $TObsQ1f=100$ e $DropQ1f=0$. Em Service espera $TqOneF$ unidades de tempo e então retorna para o estado Checking, reinicia $tq1f$ e chama a função $DequeueOneF$ }
 else { vai para o estado Service, e nessa transição incrementa $DropQ1f$ em um pacote, chama a função $MQ1f$ e faz $TObsQ1f=100/sqrt(DropQ1f)$. Em Service espera $TqOneF$ unidades de tempo e então retorna para o estado Checking, reinicia $tq1f$ e chama a função $DequeueOneF$ }

}

else { vai para o estado Service, espera $TqOneF$ unidades de tempo e então retorna para o estado Checking, reinicia $tq1f$ e chama a função $DequeueOneF$ }

}

else { Faz uma transição para o mesmo estado para reiniciar $tq1f$ }

}

else { espera até $tq1f \geq TStep$ ser verdadeiro }

Observações

Transição do lado direito na Figura II.2

fim if ($tobsQ1f \geq TObsQ1f$)

fim if ($QlenF1 > 0$)

Transição do lado esquerdo na Figura II.2

Fim do bloco ($tq1f \geq TStep$)

O autômato inicia no estado Checking e após $TStep$ unidades de tempo, isto é, quando $tq1f \geq TStep$ for verdadeiros e não há pacotes na fila, ou seja, $QlenF1 < 1$ também for verdadeiros, o autômato faz uma transição para esse mesmo estado e reinicializa o relógio $tq1f$ conforme ilustra a seta da esquerda na Figura II.2. Porém, quando for verdadeiro que $tq1f \geq TStep$ e $QlenF1 \geq 1$ o autômato transita do estado Checking para o estado Obs conforme ilustra a seta inferior direita na Figura II.2. Nesta transição o autômato chama a função $CompTminQ1f$ a qual basicamente compara o

tempo que o pacote permaneceu na fila com o mínimo tempo de permanência na fila neste período de observação. Se o tempo de espera do pacote atual foi menor que o mínimo calculado até esse instante, o mínimo (T_{minQ1f}) é atualizado, caso contrario nenhuma operação é feita.

O estado **Obs** tem dois caminhos de saída, ou seja, duas situações são possíveis. O caminho ilustrado na seta superior será habilitado para aqueles pacotes que cheguem antes de terminar um período de observação, ou seja, quando $tobsQ1f < T_{ObsQ1f}$ for verdadeiro. $tobsQ1f$ é um relógio iniciado no momento do início do atual período de observação, e T_{ObsQ1f} é uma variável cujo valor inicia em 100 ms e representa o tempo que deve durar cada período de observação (T_{ObsQ1f} é a variável T_{obs} dada na Equação (2.8) para o caso particular da fila aqui modelada). Neste caso o autômato migra diretamente para o estado **Service**.

Caso contrário, uma vez finalizado um período de observação, ou seja, quando $tobsQ1f \geq T_{ObsQ1f}$ for verdadeiro de acordo com o algoritmo CoDel é necessário verificar se o mínimo tempo de espera observado neste período é menor que o valor objetivo de 5 ms. Então, o autômato migra para o estado **TargetTime**. Onde novamente existem dois caminhos de saída, o caminho ilustrado na seta superior será habilitado quando o mínimo tempo de espera na fila observado neste período seja menor ao valor alvo ou seja $T_{minQ1f} \leq Target$. Onde **Target** é uma constante igual ao valor alvo do mínimo tempo de espera para cada período de observação (5 ms). Neste caso o autômato migra para o estado **Service** e o novo intervalo de observação é ajustado para 100 ms de acordo com as especificações do algoritmo CoDel descritas na subseção 2.6.2.

Caso contrário, ou seja, quando $T_{minQ1f} \geq Target$ for verdadeiro, isto é, o mínimo tempo de espera for maior ao mínimo desejado, então o algoritmo CoDel interpreta como sinal de congestionamento, e o modelo autômato também migra para o estado **Service**. Porém, nessa transição é chamada a função **MQ1f** a qual basicamente coloca uma marca no pacote, que irá para o atendimento, e também é atualizado o tempo que durará o próximo período de observação ajustando a variável T_{ObsQ1f} de acordo com a Equação (2.8).

No estado **Service** o autômato permanece por T_{qOneF} unidades de tempo, equivalente ao tempo necessário para colocar um pacote de 1 kbyte em um canal de 1,5 Mbps. Logo passado esse tempo, ou seja, quando a condição $tq1f \geq T_{qOneF}$ for verdadeira o autômato retorna para o estado **Checking** e nesta transição chama a função **DequeueOneF**, conforme ilustra a seta superior para a esquerda na Figura II.2. A função **DequeueOneF** basicamente modela a saída de um pacote de essa fila e sua entrada dentro do canal de ida existente entre roteador 1 e o roteador 2. As outras filas foram modeladas de forma parecida. E os resultados das simulações são apresentados na seção 3.5.

II.3 Modelo do algoritmo PIE no UPPAAL

Para modelar no UPPAAL a implementação da técnica AQM PIE na topologia de rede de comunicação daisy-chain da Figura 3.3, ao modelo de autômatos temporizados apresentado na Seção 3.3 foram modificados os autômatos correspondentes ao modelo de alguns canais e das filas nos roteadores, e ainda, foi adicionado um novo autômato mostrado na Figura II.3, e modela a

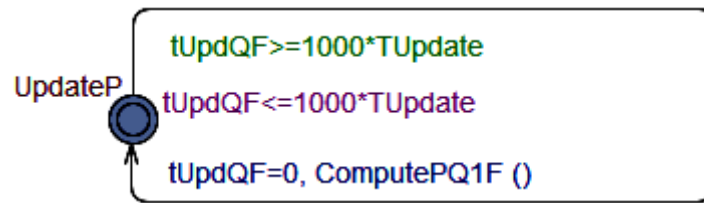


Figura II.3: Modelo autômato temporizado para realizar a primeira parte do algoritmo utilizando PIE

primeira parte do algoritmo PIE descrito na subseção 2.6.3. O mesmo consiste de um único estado denominado UpdateP e coordenado pelo relógio tUpdQF.

O pseudocódigo para este autômato é dado no Algoritmo II.3, enquanto que as constantes, variáveis e funções utilizadas no modelo autômato temporizado são descritas na Tabela II.3.

Tabela II.3: Descrição das variáveis, funções e constantes utilizadas no modelo da primeira parte do algoritmo PIE.

Nome	Descrição
TUpdate	Constante igual a 30 ms. Igual ao tempo requerido para a atualização da primeira parte do algoritmo PIE.
tUpdQF	Relógio que coordena o autômato .
ComputePQ1F	Função chamada a cada 1000.TUpdate μs . Basicamente executa todos os cálculos correspondente a primeira parte do algoritmo PIE descrito na subseção 2.6.3.

Algoritmo II.3 Pseudocódigo para o autômato da Figura II.3.

Pseudocódigo para o autômato que executa a primeira parte do algoritmo PIE.	Observações
Inicializa no estado UpdateP if (tUpdQF >= 1000.TUpdate) {Faz uma transição para o mesmo estado reinicia tUpdQF e chama a função ComputePQ1F }	Transição do lado direito na Figura II.3

A cada mil TUpdate unidades de tempo, ou seja, quando a condição $tUpdQF \geq 1000 \cdot TUpdate$ for verdadeira, na qual TUpdate é uma constante igual a 30 ms multiplicada por um fator de 1000 para convertê-la a microssegundos (unidade utilizada em todos os relógio dos autômatos) o autômato faz uma transição para esse mesmo estado na qual reinicia seu relógio tUpdQF e chama a função ComputePQ1F a qual executa todos os cálculos correspondente a primeira parte do algoritmo PIE descrito na subseção 2.6.3.

II.3.1 Modificações no modelo dos canais para modelar a técnica AQM PIE

Para implementar a técnica AQM PIE alguns canais foram modificados com respeito ao modelo da topologia de rede apresentado na Seção 3.3. Assim a Figura II.4 ilustra o novo modelo autômato temporizado para o canal no caminho de ida localizado entre o transmissor e o roteador 1. Esse

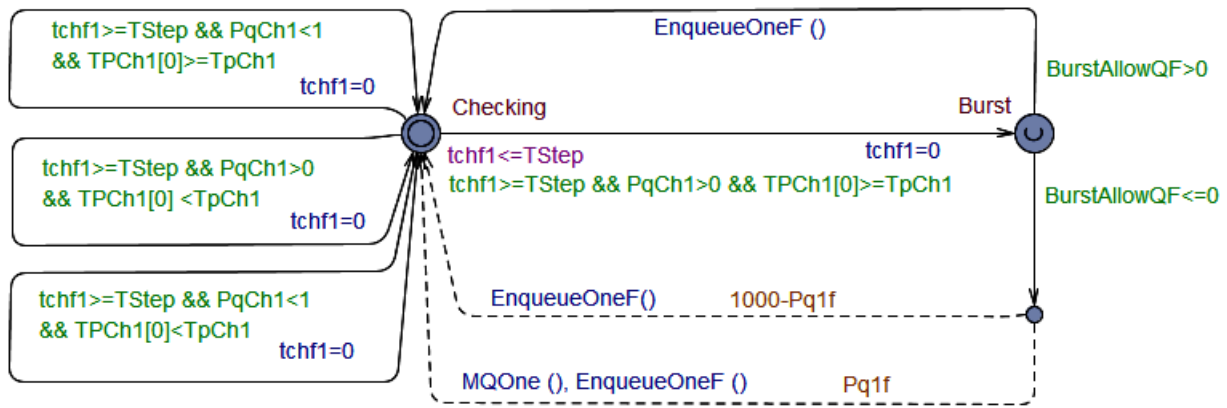


Figura II.4: Modelo autômato temporizado para o canal de ida situado entre o transmissor e o roteador 1 utilizando PIE.

autômato temporizado está formado por dois estados denominados Checking e Burst e um ponto de ramificação (marcado com um círculo menor).

O pseudocódigo para este autômato é dado no Algoritmo II.4, enquanto que as constantes, variáveis e funções utilizadas no modelo autômato temporizado são descritas na Tabela II.4.

Tabela II.4: Descrição das variáveis, funções e constantes utilizadas no modelo do canal 1 de ida com a técnica AQM PIE.

Nome	Descrição
TStep	Passo de tempo (resolução temporal) - valor adotado $=1\mu s$.
tchf1	Relógio que coordena as transições do estado Checking.
TpCh1	Constante igual ao tempo de propagação no canal 1 (2 ms).
TPCh1	Arranjo de Relógios que contenham o tempo de de propagação de cada um dos pacotes que já saíram do transmissor mas ainda não chegaram no roteador 1, ou seja, o tempo transcorrido desde que cada pacote saiu do transmissor até o instante atual.
Pq1f	Variável que contém a probabilidade de marcar um pacotes multiplicada por um fator de 1000.
MQOne	Função chamada com probabilidade $Pq1f/1000$ cada vez que um pacote chega à fila de ida do roteador 1. Basicamente coloca uma marca no bit ECN no cabeçalho do pacote.
EnqueueOneF	Função chamada cada vez que um pacote chega à fila de ida do roteador 1. Basicamente incrementa a variável $QlenF1$ em um pacote e decrementa a variável $PqCh1$ em um pacote.
PqCh1	Variável que contém a quantidade de pacotes dentro do canal 1 de ida.
QlenF1	Variável que contém a quantidade de pacotes dentro da fila de ida do roteador 1.
BurstAllowQF	Variável utilizada para controlar a tolerância a rajadas descrita na terceira parte do algoritmo PIE (veja a subseção 2.6.3).

Algoritmo II.4 Pseudocódigo para o autômato da Figura II.4.

Pseudocódigo para o autômato do canal 1 de ida com a técnica AQM PIE.

Inicializa no estado Checking

if ($tchf1 \geq TStep$) {if ($PqCh1 > 0$ and $TPCh1 [0] \geq TpCh1$) {Vai para o estado Burst e reinicia o relógio $tq1f$.if ($BurstAllowQF > 0$) {retorna para o estado Checking e nessa transição chama a função `EnqueueOneF`}else {Vai para o ponto de ramificação. Com probabilidade $(1000-Pq1f)/1000$ retorna para o estado Checking pelo caminho superior, e nessa transição chama a função `EnqueueOneF`.Com probabilidade $Pq1f/1000$ retorna para o estado Checking pelo caminho inferior, e nessa transição chama as funções `MQOne` e `EnqueueOneF`}

}

else { if ($PqCh1 < 1$ or $TPCh1 [0] < TpCh1$){ Faz uma transição para o mesmo estado para reiniciar $tchf1$ }}

}

else {espera até $tchf1 \geq TStep$ ser verdadeiro }**Observações**

Transição do lado direito na Figura II.4.

fim do bloco if ($PqCh1 > 0$ and $TPCh1 [0] \geq TpCh1$)

Transições do lado esquerdo na Figura II.4.

Fim do bloco ($tchf1 \geq TStep$)

O autômato inicia no estado `Checking`, no qual permanece por $TStep$ unidades de tempo. Passado esse tempo, ou seja, quando a condição $tch1f \geq TStep$ for verdadeira, e se existir pelo menos um pacote no canal, ou seja, se a condição $PqCh1 > 0$ for verdadeira e um pacote já chegou no roteador 1, isto é, se a condição $TProp[0] \geq TpCh1$ também for verdadeira o autômato transita do estado `Checking` para o estado `Burst` e reinicia seu relógio $tch1f$.

O estado `Burst` é um estado urgente e admite duas possíveis transições, cujo objetivo é modelar a tolerância a rajadas descrita na terceira parte do algoritmo `PIE` (veja a subseção 2.6.3). Assim conforme o algoritmo `PIE` se $BurstAllowQF$ for maior que zero o pacote é enfileirado sem aplicar a probabilidade de marcar pacotes ($BurstAllowQF$ é a variável *BurstAllow* descrita na subseção 2.6.3 para o caso particular da fila aqui modelada). Então, o autômato retorna para o estado `Checking` e nessa transição chama a função `EnqueueOneF` que modela o enfileiramento do pacote (veja a seta superior que parte desde o estado `Burst` até o estado `Checking` na Figura II.4).

Desde o estado `Burst`, a segunda transição é habilitada quando a condição $BurstAllowQF \leq 0$ for verdadeira. Neste caso o autômato migra para o ponto de ramificação ilustrado na parte inferior direita da Figura II.4, no qual novamente existem dois caminhos possíveis um com peso $Pq1f$ e outro com peso $1000-Pq1f$. $Pq1f$ é a probabilidade de marcar pacotes na fila no caminho de ida do roteador 1 multiplicada por um fator de 1000 (essa multiplicação é feita para convertê-la em um número inteiro já que o `UPPAAL` não aceita números flutuantes como peso para suas ramificações), calculada pela função `ComputePQ1`. Assim, se o autômato migra pelo caminho com peso $Pq1f$ é chamada a função `MQOne`, a qual coloca uma marca no cabeçalho do pacote, e logo é chamada a função `EnqueueOneF` que modela o enfileiramento do pacote. Caso o autômato migre

pelo caminho com peso 1000-Pq1f o pacote não será marcado e somente será enfileirado pela função EnqueueOneF.

Estando no estado **Checking**, as transições da esquerda na Figura II.4 ocorrem quando após um tempo TStep não existir pacotes no canal, ou se existir, ainda não se propagaram até o roteador 1.

II.3.2 Modificações no modelo das filas para modelar a técnica AQM PIE

Para implementar a técnica AQM PIE, também foram modificados os autômatos temporizados que modelam as filas nos roteadores respeito ao modelo da topologia de rede apresentado na Seção 3.3, assim a Figura II.5 ilustra o novo modelo autômato temporizado para a fila no caminho de ida no roteador 1, onde são enfileirados pacotes de dados que viajam do transmissor para o receptor. Conforme pode ser observado, esse autômato temporizado está formado por quatro estados, chamados **Checking**, **Service**, **Measurement** e **UpdateRate**.

O pseudocódigo para este autômato é dado no Algoritmo II.5, enquanto que as constantes, variáveis e funções utilizadas no modelo autômato temporizado são descritas na Tabela II.5.

Tabela II.5: Descrição das variáveis, funções e constantes utilizadas no modelo da fila de ida no roteador 1 com a técnica AQM PIE.

Nome	Descrição
TStep	Passo de tempo (resolução temporal) - valor adotado $=1\mu s$.
tq1f	Relógio que coordena as transições dos estados Checking e Service .
TqOneF	Constante igual ao tempo em μs requerido para colocar um pacote de 1 kbyte em um canal com capacidade 1,5 Mbps (canal 2).
QlenF1	Variável que contém a quantidade de pacotes dentro da fila de ida do roteador 1.
DequeueOneF	Função chamada cada vez que um pacote sai da fila de ida do roteador 1. Basicamente decrementa QlenF1 em um pacote e incrementa a variável PqCh2 em um pacote.
PqCh2	Variável que conta a quantidade de pacotes dentro do canal 2 de ida.
DqThr	Constante igual a 10 Kbyte (equivalente à variável <i>DqThreshol</i> do algoritmo PIE subseção 2.6.3).
DqCountQF	Variável que conta a quantidade de Bytes que saíram da fila em um determinado período. É equivalente à variável <i>DqCount</i> descrita na subseção 2.6.3 para o caso particular da fila aqui modelada.
UpDateQF	Função que basicamente realiza as atualizações descritas pelas Equações (2.13), (2.14), (2.15), (2.16) e (2.17) da segunda parte e as Equações (2.18) e (2.19) da terceira parte do algoritmo PIE descrito na subseção 2.6.3.

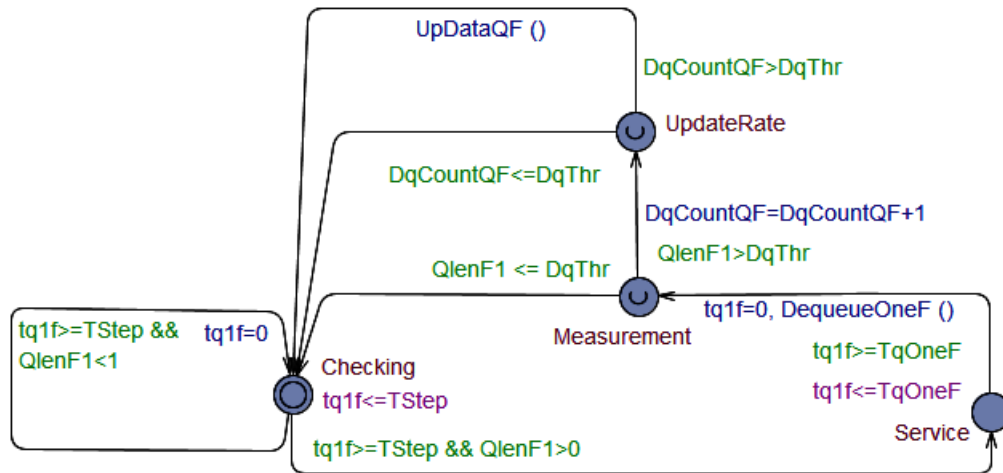


Figura II.5: Modelo autômato temporizado para a Fila canal de ida no roteador 1 utilizando PIE

Algoritmo II.5 Pseudocódigo para o autômato da Figura II.5.

<p>Pseudocódigo do autômato que modela a fila no caminho de ida no roteador 1 com a técnica AQM PIE.</p> <p>Inicializa no estado Checking</p> <pre> if (tq1f >= TStep) { if (QlenF1 > 0) { vai para o estado Service espera TqOneF unidades de tempo. Logo, vai para o estado Measurement, reinicia tq1f e chama a função DequeueOneF. if (QlenF1 > DqThr) { vai para o estado UpdateRate e nesta transição incrementa a variável DqCountQF em um pacote. if (DqCountQF > DqThr) {vai para o estado Checking, e nesta transição chama a função UpDateQF} else {vai para o estado Checking} } else {vai para o estado Checking} } else { Faz uma transição para o mesmo estado para reiniciar tq1f } } else { espera até tq1f >= TStep ser verdadeiro } </pre>	<p>Observações</p> <p>Transição do lado direito na Figura II.5</p> <p>fim do bloco if (QlenF1 > DqThr)</p> <p>fim do bloco if (QlenF1 > 0)</p> <p>Transição do lado esquerdo na Figura II.5</p> <p>Fim do bloco (tq1f >= TStep)</p>
---	---

O autômato começa no estado Checking e após TStep unidades de tempo, ou seja, quando a condição $tq1f \geq TStep$ for verdadeira, e não há pacotes na fila, isto é, a condição $QlenF1 < 1$, também for verdadeira o autômato faz uma transição para esse mesmo estado e reinicializa seu relógio $tq1f$ conforme ilustra a seta da esquerda na Figura II.5.

Após TStep unidades de tempo quando houver pacotes na fila, ou seja, quando as condições $tq1f \geq TStep$ e a $QlenF1 \geq 1$ forem ambas verdadeiras o autômato transita do estado Checking para o estado Service conforme ilustra a seta inferior direita na Figura II.5.

No estado Service o autômato permanece por TqOneF unidades de tempo, equivalente ao tempo necessário para colocar um pacote de 1 kbyte em um canal de 1,5 Mbps. Logo, passado esse tempo

o autômato migra para o estado **Measurement** e nesta transição chama a função **DequeueOneF**. O estado **Measurement** é um estado urgente, utilizado para modelar a segunda parte do algoritmo PIE descrito na subseção 2.6.3, e contém duas transições possíveis. A primeira acontece quando **QlenF1** for menor que **DqThr**. **DqThr** é uma constante igual a 10 Kbyte (equivalente à variável *DqThreshol* do algoritmo PIE subseção 2.6.3). Neste primeiro caso o autômato retorna para o estado **Checking** e nenhuma operação é realizada.

A segunda possível transição desde o estado **Measurement** acontece quando **QlenF1** for maior que **DqThr**. Neste caso o autômato migra para o estado **UpdateRate** e nesta transição incrementa em 1 Kbyte o contador **DqCountQF** (**DqCountQF** é a variável *DqCount* descrita na subseção 2.6.3 para o caso particular da fila aqui modelada).

O estado **UpdateRate** também é um estado urgente, e também contém duas possíveis transições, a primeira acontece quando **DqCount** for menor ou igual a **DqThr**. Neste caso o autômato retorna para o estado **Checking** e nenhuma operação é realizada, e a segunda acontece se **DqCount** for maior que **DqThr**, neste caso, o autômato também migra para o estado **Checking**, mas nesta transição é chamada a função **UpDataQF** que basicamente realiza as atualizações descritas pelas Equações (2.13), (2.14), (2.15), (2.16) e (2.17) da segunda parte e as Equações (2.18) e (2.19) da terceira parte do algoritmo PIE descrito na subseção 2.6.3.

III. ANEXO: MODELOS DE NCS EM UPPAAL

Neste Anexo, será apresentado o modelo autômato da planta (motor cc Maxon F2140 ilustrado na Figura 3.24) e do controlador utilizado na NCS descrita na seção 3.7 e ilustrada na Figura 3.25.

III.1 Modelo da planta

A Figura III.1 mostra o modelo autômato da planta o qual consiste de dois estados denominados `SamplingP` e `SendingP`, e o pseudocódigo para este autômato é dado no Algoritmo III.1, enquanto que as variáveis utilizadas no modelo autômato temporizado são descritas na Tabela III.1.

Conforme observasse na Figura III.1 o autômato inicia no estado `SamplingP` e permanece nele durante h unidades de tempo, sendo h o período de amostragem de 0,014 s (equivalente ao h utilizado na seção 3.7).

Logo, o autômato migra para o estado `SendingP`, e nessa transição chama as funções `ComputePosition` e `ComputeITAE` as quais basicamente calculam a saída $y(k)$ da planta e o valor do ITAE respectivamente.

No estado `SendingP` o autômato permanece por T_{psm} unidades de tempo, equivalente ao tempo necessário para inserir um pacote de 48 bytes (contendo o valor medido de $y(k)$) em um canal de 10 Mbps. Depois, transita novamente para o estado `SamplingP` e nessa transição chama a função `SendMeasured` a qual basicamente incrementa o número de pacotes dentro do canal localizado entre o roteador 1 e a planta (veja a Figura 3.25).

Tabela III.1: Descrição das variáveis, funções e constantes utilizadas no modelo do canal 1 de ida com a técnica AQM RED.

Nome	Descrição
h	período de amostragem (0,014 s)
tp	Relógio que coordena as transições entre estados.
T_{psm}	Constante igual ao tempo necessário para inserir um pacote de 48 bytes em um canal de 10 Mbps.
<code>ComputePosition</code>	Função chamada a cada h unidades de tempo. Basicamente calcula a saída $y(k)$ da planta.
<code>ComputeITAE</code>	Função chamada a cada h unidades de tempo. Basicamente calcula o valor atual do ITAE.
<code>SendMesasured</code>	Função chamada cada vez que a planta envia um pacote contendo o valor da saída $y(k)$. Basicamente incrementa o número de pacotes dentro do canal localizado entre o roteador 1 e a planta

Algoritmo III.1 Pseudocódigo para o autômato da Figura III.1.

Pseudocódigo para o autômato que modela a dinâmica da planta.

Inicializa no estado **SamplingP**

```
if (tp >= h) {Vai para o estado SendingP e nesta transição  
reinicia o relógio tp e chama as funções ComputePosition e  
ComputeITAE. Espera  $T_{psm}$  unidades de tempo e então retorna  
para o estado SamplingP e chama a função SendMeasured}  
else {espera até  $tp \geq h$  ser verdadeiro }
```

Observações

Transição do lado direito na
Figura III.1.

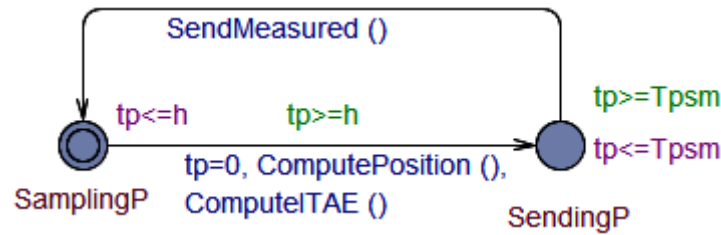


Figura III.1: Automato que modela a planta.

III.2 Modelo do controlador

A Figura III.2 mostra o modelo autômato do controlador o qual consiste de dois estados denominados **SamplingCS** e **SendingCS**, e o pseudocódigo para este autômato é dado no Algoritmo III.2, enquanto que as variáveis utilizadas no modelo autômato temporizado são descritas na Tabela III.2.

Conforme observasse na Figura III.2 o autômato inicia no estado **SamplingCS** e permanece nele durante h unidades de tempo, sendo h o período de amostragem de 0,014 s (equivalente ao h utilizado na seção 3.7).

Logo, o autômato migra para o estado **SendingPCS**, e nessa transição chama a função **ComputeControlSignal** a qual basicamente calcula a lei de controle $u(k)$.

No estado **SendingCS** o autômato permanece por T_{csc} unidades de tempo, equivalente ao tempo necessário para inserir um pacote de 48 bytes (contendo o valor calculado de $u(k)$) em um canal de 10 Mbps. Depois, transita novamente para o estado **SendingCS** e nessa transição chama a função **SendControlSignal** a qual basicamente incrementa o número de pacotes dentro do canal localizado entre o controlador e o roteador 2 (veja a Figura 3.25).

Tabela III.2: Descrição das variáveis, funções e constantes utilizadas no modelo do canal 1 de ida com a técnica AQM RED.

Nome	Descrição
h	período de amostragem (0,014 s)
tc	Relógio que coordena as transições entre estados.
Tcsc	Constante igual ao tempo necessário para inserir um pacote de 48 bytes em um canal de 10 Mbps.
ComputeControlSignal	Função chamada a cada h unidades de tempo. Basicamente calcula a lei de controle $u(k)$.
SendControlSignal	Função chamada cada vez que o controlador envia um pacote contendo o valor $u(k)$ da lei de controle. Basicamente incrementa o número de pacotes dentro do canal localizado entre o controlador e o roteador 2.

Algoritmo III.2 Pseudocódigo para o autômato da Figura III.2.

Pseudocódigo para o autômato que modela o controlador.	Observações
Inicializa no estado SamplingP	
if (tc >= h) {Vai para o estado SendingCS e nesta transição reinicia o relógio tc e chama a função ComputeControlSignal.	Transição do lado direito na Figura III.2.
Espera Tcsc unidades de tempo e então retorna para o estado SamplingCS e chama a função SendControlSignal}	
else {espera até tc >=h ser verdadeiro }	

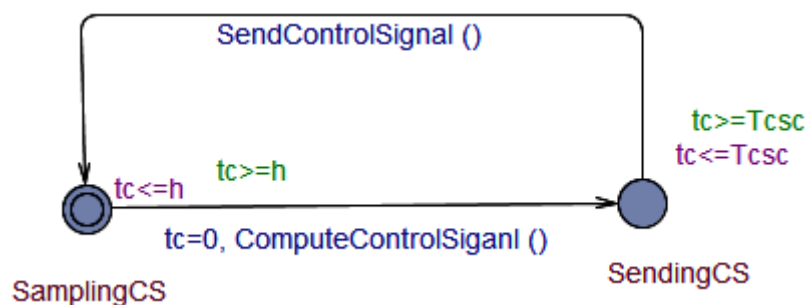


Figura III.2: Modelo do controlador