Dissertação de Mestrado

Inatel
Instituto Nacional de Telecomunicações

ANÁLISE DE DESEMPENHO DE REDES TCP SEM FIO

BRUNO COUTO COSTA PINTO

AGOSTO / 2006

Análise de Desempenho de Redes TCP Sem Fio

BRUNO COUTO COSTA PINTO

Dissertação apresentada ao Instituto Nacional de Telecomunicações, como parte dos requisitos para obtenção do Título de Mestre em Telecomunicações.

ORIENTADOR: Prof. Dr. José Marcos Câmara Brito

Santa Rita do Sapucaí
2006

FOLHA DE APROVAÇÃO

Dissertação defendida e aprovada em 14/08/2006,
pela comissão julgadora:

_____

Prof. Dr. José Marcos Câmara Brito – INATEL

Orientador

_____

Prof. Dr. Antônio Marcos Alberti – INATEL

_____

Prof. Dr. Anilton Salles Garcia – UFES, ES.

_____

Prof. Dr. Adonias C. da Silveira
Coordenador do Curso de Mestrado
em Telecomunicações

Agradecimentos

A Deus, por ter me dado a força necessária para ultrapassar mais um obstáculo.

Ao Inatel, por ter sido o agente facilitador para que este mestrado se concretizasse.

Ao meu orientador José Marcos Câmara Brito, pelo grande envolvimento neste trabalho.

Aos meus pais Marise e Marco Aurelio por todo o incentivo que sempre me deram.

À minha namorada Anna Paula, por toda ajuda e compreensão.

Resumo

As redes TCP/IP foram concebidas considerando a utilização de meios de transmissão com fio. No entanto, com o grande crescimento da Internet e das transmissões sem fio, inserir o TCP neste novo cenário se tornou uma tarefa necessária. Como algumas características das redes sem fio são prejudiciais ao TCP padrão, entre elas as maiores taxas de erro de bit, as perdas intermitentes de sinal entre extremidades do enlace sem fio, assimetria de largura de banda entre os sentidos *downlink* e *uplink* do enlace sem fio e os longos atrasos de propagação no caso dos enlaces satélite, muito esforço foi feito no sentido de adaptá-lo a esse novo ambiente. O objetivo desta dissertação é apresentar algumas soluções encontradas na literatura desenvolvidas para amenizar o impacto dessas características inerentes às redes sem fio e dessa forma melhorar o desempenho do TCP. Além disso, especificamente para o problema das altas taxas de erro de bit, desenvolver um modelo analítico capaz de estimar o desempenho do TCP quando a solução para melhoria do seu desempenho é a utilização de mecanismos de controle de erro *Stop-and-Wait ARQ* (*Automatic Repeat reQuest*), *Go-Back-N ARQ* e *FEC* (*Forward Error Correction*) no enlace sem fio. Neste modelo serão considerados sistemas adaptativos, nos quais uma estimativa da qualidade do canal está disponível, e sistemas não adaptativos. Fazendo uso deste modelo, analisar redes com enlace característico de uma rede celular e ainda redes com enlace característico de uma rede satélite.

Abstract

The TCP/IP networks were designed considering the use of wired media. Although, the huge growth of the Internet together with the growth of the wireless transmissions caused the need of insertion of TCP in this new scenario. As some of the wireless networks characteristics are harmful to the standard TCP, as the higher bit error rates, the intermittent signal loss between the wireless link ends, the bandwidth asymmetry between wireless uplink and downlink and the long propagation delays in the case of the satellite links, much effort was done to adapt TCP to this new environment. The objective of this dissertation is to present some of the solutions found in the literature developed to decrease the impact of those characteristics inherent to wireless networks in the TCP performance. Besides, concerning the higher bit error rates problem, to develop an analytical model able to estimate the TCP performance when the solution to improve its performance is the use of Stop-and-Wait *ARQ* (Automatic Repeat Request), Go-Back-N *ARQ* and *FEC* (Forward Error Correction) error control mechanisms. In this model, adaptive systems, in what signal strength estimative is available, and non-adaptive systems are considered. With this model, we are going to analyze networks with cellular links and satellite links characteristics.

Lista de Figuras

Figura 1.1 – Número de aparelhos celulares móveis no Brasil. Fonte: [1]. 16
Figura 1.1 – Número de *hosts* conectados à Internet no Brasil. Fonte: [3]. 16
Figura 2.1 – O cabeçalho TCP. Fonte: [6]..... 21
Figura 2.2 – ACKs Cumulativos. 23
Figura 2.3 – *Fast Retransmission* do segmento 2..... 26
Figura 2.4 – Janela Deslizante. Fonte: [6]. 27
Figura 2.5 – Movimentos das laterais da janela. Fonte: [6]. 27
Figura 2.6 - Evolução dinâmica da janela de congestionamento do TCP. Fonte: [11]. 30
Figura 3.1 – Evolução seqüencial do TCP. 32
Figura 3.2 – Rede com enlace sem fio..... 33
Figura 3.3 – Diagrama de estados do sistema híbrido de múltiplos estados *FEC*. Fonte: [15]. 38
Figura 3.4 – Topologia do sistema abordado pelo Snoop. Fonte: [11]. 40
Figura 3.5 – Protocolo Snoop com transmissão no sentido HF para HM. Fonte: [11]. 42
Figura 3.6 – Evolução seqüencial do TCP quando das perdas intermitentes de sinal..... 44
Figura 3.7 – *Goodput* normalizado para uma rede assimétrica com *buffer* infinito no caminho reverso. 50
Figura 3.8 – Evolução seqüencial do TCP da para uma rede assimétrica com *buffer* finito no caminho reverso..... 51
Figura 3.9 – Rede com enlace satélite. 54
Figura 3.10 – *Split*-TCP em uma conexão com enlace satélite. 56
Figura 3.11 – TCP fim a fim *versus* TCP *Spoofing*..... 58
Figura 4.1 – Modelo original. Fonte: [28]. 62
Figura 4.2 – Estado da rede no estado inicial. 62
Figura 4.3 – Estado da rede após a transmissão do segmento 1 no enlace sem fio. 63
Figura 4.4 – Estado da rede após a transmissão do segmento 2 no enlace sem fio. 63
Figura 4.5 – Estado da rede na transmissão dos ACKs dos segmentos 1 e 2 no enlace sem fio. 63
Figura 4.6 – Estado da rede que ilustrando o espaçamento temporal t_{esp} entre os ACKs dos.. 64
Figura 4.7 – Estado da rede após a transmissão do segmento 5 no enlace com fio. 64
Figura 4.8 – Estado da rede na transmissão do segmento 4 no enlace sem fio. 64
Figura 4.9 – Estado da rede na transmissão do segmento 5 no enlace sem fio. 64
Figura 4.10 – Rede totalmente preenchida com segmentos e ACKs..... 65
Figura 4.11 – Estado da rede com *bit pipe* com um segmento a mais que o máximo suportado. 65
Figura 4.12 – Comportamento cíclico da janela de congestionamento para transmissores 66
(a) TCP-Reno e (b) TCP-Tahoe. 66
Figura 4.13 – *Throughput* Normalizado do TCP-Reno quando (a) τ = 1s e (b) B = 20 pacotes. 69
Figura 4.14 – *Throughput* normalizado do TCP-Tahoe quando (a) τ = 1s e (b) B = 20 pacotes. 69
Figura 4.15 – Modelo modificado 71
Figura 4.16 – *Throughput* normalizado do TCP para a Rede C no cenário não adaptativo com k = 4320, τ = 1s e B = 20 para os sistemas (a) *GBN* e Híbrido, e (b) *SW* e *FEC*. 73
Figura 4.17 – *Throughput* normalizado do TCP para a Rede C com *FEC* no cenário não adaptativo quando os parâmetros k, B e τ são alterados. 74

Figura 4.18 – *Throughput* normalizado do TCP para Rede C com *FEC* nos cenários adaptativo simples e adaptativo ideal. 75
Figura 4.19 – *Throughput* normalizado do TCP para a Rede S com *FEC* nos cenários não adaptativo, adaptativo simples e adaptativo ideal. 76
Figura 4.20 – Estado da rede no instante t = 1. 77
Figura 4.21 – Estado da rede nos instantes t = 9,6, t = 11,7 e t = 12,5. 77
Figura 4.22 – Estado da rede no instante t = 15,8. 78
Figura 4.23 – Estado da rede nos instantes t = 23,6. 78
Figura 4.24 – Estado da rede nos instantes t = 30,5. 78
Figura 4.25 – Estado da rede nos instantes t = 38,4. 79
Figura 4.26 – Estado da rede nos instantes t = 0 e t = 3,0. 79
Figura 4.27 – Estado da rede nos instantes t = 3,8 e t = 6,8. 80
Figura 4.28 – Estado da rede no instante t = 7,6. 80
Figura 4.29 – Estado da rede no instante t = 18,2. 80
Figura 4.30 – Estado da rede no instante t = 25,8. 81
Figura 4.31 – Estado da rede no instante em que o *buffer* volta a ter 2 segmentos. 81
Figura 4.32 – Rede com parâmetros alterados. 82
Figura 4.33 – Evolução da Janela de Congestionamento. 83
Figura 4.34 – Evolução da Janela de Congestionamento: ocorrência de *timeout* e duplo *FR*/*FR*. 83
Figura 4.35 – Evolução da janela de congestionamento para o evento de 1 *FR/FR* quando: (a) $\beta' < 1$ e (b) $\beta' > 1$. 85
Figura 4.36 – Evolução da janela de congestionamento para o evento de 2 *FR/FR* quando: (a) $\beta' < 1$, (b) $1 < \beta' < 3$ e (c) $\beta' > 3$. 86
Figura 4.37 – Evolução da janela de congestionamento para o evento de 2 *FR/FR* quando: (a) $\beta' < 1$, (b) $1 < \beta' < 7$ e (c) $\beta' > 7$. 88
Figura 4.38 – *Throughputs* normalizados do TCP obtidos analiticamente e através de simulação para (a) código (856,820) e (b) código (4408,4320). 90
Figura 4.39 – Curvas das probabilidades associadas à ocorrência de cada evento quando o tamanho do bloco é (a) 856 bits e (b) 4408 bits. 97
Figura 4.40 – *Throughput* médio gerado na ocorrência de cada evento quando o tamanho do bloco é (a) 856 bits e (b) 4408 bits. 97
Figura 4.41 – *Throughput* normalizado do TCP com *SW* para o cenário não adaptativo quando (a) $B = 20$ e $\tau = 1$ e (b) B e τ são alterados. 98
Figura 4.42 – *Throughput* normalizado do TCP com *SW* para a Rede C nos cenários adaptativo simples e adaptativo ideal. 100
Figura 4.43 – *Throughput* normalizado do TCP com *SW* para a Rede S no cenário não adaptativo. 100
Figura 4.44 – Estado inicial da rede. 101
Figura 4.45 – Estados da rede após as retransmissões dos segmentos 3, 4 e 5 no enlace sem fio. 101
Figura 4.46 – Estados da rede que ilustram o espaçamento temporal $6 \cdot t_{GBN}$ criado entre o segmento 15 e o ACK referente ao segmento 4. 102
Figura 4.47 – Estado da rede que ilustra a redução da ocupação do *buffer*, após a ocorrência de uma retransmissão. 102
Figura 4.48– *Throughput* normalizado do TCP com *GBN* para o cenário não adaptativo. 108
Figura 4.49 – *Throughput* normalizado do TCP com *SW* para os cenários adaptativo simples e adaptativo ideal. 109
Figura 4.50 – *Throughput* normalizado do TCP com *GBN* nos cenários não adaptativo, adaptativo simples e adaptativo ideal para a Rede S. 111

Figura 4.51 – *Throughputs* normalizados do TCP com os MCEs no cenário não adaptativo para a Rede C. 111
Figura 4.52 - *Throughputs* normalizados do TCP com os MCEs nos cenários (a) adaptativo simples e (b) adaptativo ideal para a Rede C. 112
Figura 4.53 – *Throughputs* normalizados do TCP com os MCEs no cenário não adaptativo para a Rede S. 113
Figura 4.54 – *Throughputs* normalizados do TCP com os MCEs no cenário adaptativo para a Rede S. 113
Figura A.1 – Esquema da simulação para a Rede C. 124

Lista de Tabelas

Tabela 3.1 – Tempo de *Slow Start* para redes com satélites LEO, MEO e GEO. Fonte: [23]. 55
Tabela 4.1 – Número de bits de paridade para o *FEC* nos cenários adaptativos para a Rede C. 75
Tabela 4.2 – Número de bits de informação e número de bits de paridade para o *SW* nos cenários adaptativos para a Rede C. 99
Tabela 4.3 – Número de bits de informação e número de bits de paridade para o *GBN* nos cenários adaptativos para a Rede C. 109
Tabela 4.4 – Número de bits de informação e número de bits de paridade para o *GBN* nos cenários adaptativos para a Rede S. 110

Lista de Acrônimos

ACK – *Acknowledgment*.

ARQ – Automatic Repeat Request.

ASCII – *American Standard Code for Information Interchange*.

AT – Adaptação do Transmissor.

B – Tamanho do buffer.

BER – *Bit Error Rate*.

BSC – *Binary Symmetric Channel*.

c – Fator de largura de banda normalizado.

cab – Número de bits do cabeçalho TCP/IP.

cwnd – *Congestion Window*.

d_{min} – Distância mínima de um código.

E – Número esperado de transmissões no enlace sem fio para os segmentos de uma dada rodada de janela.

EB – Estação Base.

ELN – *Explicit Loss Notification*.

FA – Filtragem de ACKs.

FEC – Forward Error Correction.

FR/FR – Fast Retransmission/Fast Recovery.

FWA – *Full Window Advertisement*.

GBN – Go-Back-N.

GEO – *Geosynchronous Earth Orbit*.

HF – *Host* Fixo.

HM – *Host* Móvel.

HTTP – *Hypertext Transfer Protocol*.

IP – *Internet Protocol*.

I-TCP – *Indirect*-TCP.

k – Número de bits de informação em um bloco.

k' – Número de bits de dados do TCP presente em cada bloco.

L_D – Tamanho dos pacotes no caminho direto.

LEO – *Low Earth Orbit*.

L_R – Tamanho dos pacotes enviados no caminho reverso.

MCE – Mecanismos de controle de erro.

MEO – *Medium Earth Orbit*.

MSS – *Maximum Segment Size*.

n – Número total de bits em um bloco.

n_A – Número de pacotes enviados na sub-fase linear.

NACK – *Negative Acknowledgment*.

n_B – Número de pacotes enviados na sub-fase sub-linear.

N_c – Número de pacotes enviados em um ciclo.

ns-2 – *network simulator* 2.

n_{SS} – Número de pacotes enviados na fase de *Slow Start*.

n_T – Número de pacotes transmitidos no intervalo t_T.

n_1 – Número de pacotes transmitidos no intervalo t_1.

n_2 – Número de pacotes transmitidos no intervalo t_2.

p – Probabilidade de erro de bit no canal.

P_{arq} – Probabilidade do mecanismo híbrido *ARQ-FEC* com múltiplos estados *FEC* estar no estado *ARQ*.

P_c – Probabilidade de um bloco recebido não conter erros.

P_e – Probabilidade de um bloco conter um erro detectável.

P_{cca} – Probabilidade de o mecanismo *ARQ* entregar um bloco sem erros para as camadas superiores.

P_{ccf} – Probabilidade de o mecanismo *FEC* entregar um bloco sem erros para as camadas superiores.

P_{cch} – Probabilidade de o mecanismo híbrido *ARQ-FEC* com múltiplos estados *FEC* entregar um bloco sem erros para as camadas superiores.

P_i - Probabilidade de que o número de transmissões no enlace sem fio para uma determinada rodada de janela seja igual a um determinado limiar.

P_{nu} - Probabilidade de que o número de transmissões no enlace sem fio para uma determinada rodada de janela não ultrapasse um determinado limiar.

P_{NW} – Probabilidade do evento de nenhum descarte de pacote no *buffer*, dada uma rodada de janela de tamanho W.

P_T – Probabilidade da ocorrência do evento de um *timeout*.

P_{TW} – Probabilidade do evento de timeout, dada uma rodada de janela de tamanho W.

P_{ue} – Probabilidade de um bloco conter um erro não detectável.

P_1 – Probabilidade da ocorrência do evento de um *FR/FR*.

P_{1W} – Probabilidade do evento de 1 *FR/FR*, dada uma rodada de janela de tamanho W.

P_2 – Probabilidade da ocorrência do evento de dois *FR/FR*.

P_{1W} – Probabilidade do evento de 2 *FR/FR*, dada uma rodada de janela de tamanho W.

RA – Reconstrução de ACKs.

R_D – Largura de banda no sentido direto.

R_R – Largura de banda no sentido reverso.

RTO – *Retransmission Timeout*.

RTT – *Round-Trip Time*.

RTTVAR – *Round-Trip Time Variation*.

SR – Selective-Repeat.

SRTT – *Smoothed Round-Trip Time*.

ssthresh - *slow start threshold*.

SW – Stop-and-Wait.

SYN – Segmento especial para sincronismo do TCP.

t – Capacidade (de correção de erro) do código.

T – Atraso global somado ao tempo de transmissão no enlace gargalo.

t_A – Duração da sub-fase linear.

t_{aw} – Tempo de transmissão do ACK no enlace sem fio.

t_B – Duração da sub-fase sub-linear.

T_c – Duração de um ciclo.

TCP – Transmission Control Protocol.

TN – *Throughput* Normalizado do TCP.

t_{pw} – Tempo de propagação no enlace sem fio.

t_{SS} – Duração da fase de Slow Start.

T/TCP – TCP for Transactions.

t_T – Intervalo de tempo entre a diminuição da janela de congestionamento e o início de seu crescimento, no caso de um timeout.

t_1 – Intervalo de tempo entre a diminuição da janela de congestionamento e o início de seu crescimento, no caso de um *FR/FR*.

t_2 – Intervalo de tempo entre a diminuição da janela de congestionamento e o início de seu crescimento, no caso de dois *FR/FR*.

y – Número de estados *FEC* no mecanismo híbrido *ARQ-FEC* com múltiplos estados *FEC*.

W – Tamanho da janela de congestionamento.

W_{fSS} – Tamanho da janela de congestionamento ao final da fase de *Slow Start*.

W_{fA} – Tamanho final da janela na sub-fase linear.

W_{fB} – Tamanho final da janela na sub-fase sub-linear.

W_{iA} – Tamanho inicial da janela na sub-fase linear.

W_{iB} – Tamanho final da janela na sub-fase sub-linear.

win – window, janela utilizada no controle de fluxo do TCP.

W_{pipe} – Máxima janela de congestionamento que pode ser acomodada no "bit pipe".

ZWA – Zero Window Advertisement.

ZWP – Zero Window Probe.

β – Tamanho de *buffer* normalizado.

$\overline{\lambda}$ – *Throughput* médio do TCP em pacotes/segundo.

$\overline{\lambda}_R$ – *Throughput* médio do TCP-Reno em pacotes/segundo.

$\overline{\lambda}_T$ – *Throughput* médio do TCP-Tahoe em pacotes/segundo.

λ_T – *Throughput* associado à ocorrência do evento de um *timeout*.

λ_{TW} – *Throughput* associado à ocorrência do evento de um *timeout*, dada uma rodada de janela *W*.

λ_1 – *Throughput* associado à ocorrência do evento de um *FR/FR*.

λ_{1W} – *Throughput* associado à ocorrência do evento de um *FR/FR*, dada uma rodada de janela *W*.

λ_2 – *Throughput* associado à ocorrência do evento de dois *FR/FR*.

λ_{2W} – *Throughput* associado à ocorrência do evento de dois *FR/FR*, dada uma rodada de janela *W*.

μ – Taxa de transmissão do enlace gargalo.

η_{FEC} – Eficiência do mecanismo *FEC*.

η_{GBN} – Eficiência do mecanismo *GBN*.

η_h – Eficiência do mecanismo híbrido *ARQ-FEC* com múltiplos estados *FEC*.

η_{SR} – Eficiência do mecanismo *SR*.

η_{SW} – Eficiência do mecanismo *SW*.

τ – Atraso global.

τ_b – Atrasos inerentes aos protocolos baseados em retransmissão *ARQ*.

σ – Número de pacotes acrescidos ao *buffer* em cada ocorrência de retransmissão do *GBN*.

Sumário

Lista de Figuras vi
Lista de Tabelas ix
Lista de Acrônimos x
1. Introdução 16
2. *Transmission Control Protocol* 19
2.1. Características gerais 19
2.2. O cabeçalho TCP 21
2.3. MSS e janela de recepção 22
2.4. ACKs cumulativos 23
2.5. Retransmissão por temporização 23
2.6. *Fast Retransmission* (Retransmissão Rápida) 25
2.7. Controle de fluxo 26
2.8. Controle de congestionamento 28
3. O TCP e as redes sem fio 31
3.1. Redes com altas BERs 32
3.1.1. Protocolos de camada de enlace 33
3.1.1.1. Códigos de controle de erro 34
3.1.1.2. Capacidade de detecção e correção de erros 34
3.1.1.3. Limitantes para o cálculo da capacidade de correção e detecção de erros 35
3.1.1.4. Técnica *FEC* 36
3.1.1.5. Protocolos *ARQ* 36
3.1.1.6. Esquemas híbridos 38
3.1.1.7. Interações adversas entre os protocolos de camada de enlace e de transporte 39
3.1.2. O protocolo Snoop 40
3.1.2.1. Transferência de dados no sentido HF para HM 41
3.1.2.2. Transferência de dados no sentido HM para HF 42
3.2. Redes com perdas intermitentes de sinal 43
3.2.1. O *Freeze*–TCP 44
3.2.2. O ATCP 46
3.2.2.1 Transferência de dados no sentido HM para HF 46
3.2.2.2 Transferência de dados no sentido HF para HM 47
3.3. Redes com assimetria de largura de banda 48
3.3.1 – Filtragem de ACKs (FA) 52
3.3.2 – Adaptação do Transmissor (AT) 52
3.3.3 – Reconstrução de ACKs (RA) 53
3.4. Redes com longos atrasos de propagação 54
3.4.1 – Aumento da janela inicial 55
3.4.2 – O *Split*-TCP 56
3.4.3 – TCP *Spoofing* 57
3.4.4 – O TCP-*Peach* 58
4. *Throughput* do TCP com mecanismos de controle de erro *FEC*, *GBN* ou *SW* no enlace sem fio. 61
4.1 O modelo original 61
4.1.1 Validação do modelo e resultados obtidos 68
4.2 A primeira modificação 70
4.2.1 Validação do modelo e resultados obtidos 71
4.2.1.1. Resultados para a Rede C 73
4.2.1.2. Resultados para a Rede S 75

4.3 O modelo final 76
4.3.1 – Modelamento do *SW-ARQ* 76
4.3.1.1. Cálculo dos *throughputs* 84
4.3.1.1.1. Cálculo do *throughput* associado ao evento de 1 *FR/FR* 84
4.3.1.1.2. Cálculo do *throughput* associado ao evento de 2 *FR/FR* 86
4.3.1.1.3. Cálculo do *throughput* associado ao evento de um *timeout* 87
4.3.1.1.4. Validação do cálculo dos *throughputs* associados a cada evento utilizando probabilidades de eventos extraída da simulação 89
4.3.1.2. Cálculo das probabilidades dos eventos 90
4.3.1.2.1. Cálculo da probabilidade do evento de nenhum descarte no *buffer* dada a rodada de janela $W \leq W_{pipe}$' 91
4.3.1.2.2. Cálculo da probabilidade do evento de 1 *FR/FR* dada a rodada de janela $W \leq W_{pipe}$' 93
4.3.1.2.3. Cálculo da probabilidade do evento de 2 *FR/FR* dada a rodada de janela $W \leq W_{pipe}$' 94
4.3.1.2.4. Cálculo da probabilidade do evento de *timeout* dada a rodada de janela $W \leq W_{pipe}$' 94
4.3.1.2.5. Cálculo da probabilidade geral do evento de 1 *FR*/*FR* 95
4.3.1.2.6. Cálculo da probabilidade geral do evento de 2 *FR/FR* 95
4.3.1.2.7. Cálculo da probabilidade geral do evento de *timeout* 96
4.3.1.2.8. Validação do cálculo das probabilidades dos eventos 96
4.3.1.3. Validação do modelo e resultados obtidos 98
4.3.1.3.1. Resultados obtidos para a Rede C 98
4.3.1.3.2. Resultados obtidos para a Rede S 100
4.3.2 – Modelamento do *GBN-ARQ* 101
4.3.2.1. Cálculo dos *throughputs* associados aos eventos 103
4.3.2.1.1. Cálculo do *throughput* associado ao evento de 1 *FR/FR* dada uma rodada de janela $W \leq W_{pipe}$' 103
4.3.2.1.2. Cálculo do *throughput* associado ao evento de 2 *FR/FR* dada uma rodada de janela $W \leq W_{pipe}$' 104
4.3.2.1.3. Cálculo do *throughput* associado ao evento de *timeout* dada uma rodada de janela $W \leq W_{pipe}$' 105
4.3.2.2. Cálculo das probabilidades dos eventos 106
4.3.2.2.1. Cálculo da probabilidade do evento de nenhum descarte no *buffer* dada a rodada de janela $W \leq W_{pipe}$' 106
4.3.2.2.2. Cálculo da probabilidade do evento de 1 *FR/FR* dada a rodada de janela $W \leq W_{pipe}$' 106
4.3.2.2.3. Cálculo da probabilidade do evento de 2 *FR/FR* dada a rodada de janela $W \leq W_{pipe}$' 106
4.3.2.2.4. Cálculo da probabilidade do evento de *timeout* dada a rodada de janela $W \leq W_{pipe}$' 107
4.3.2.3. Cálculo dos produtos $P_1 \cdot \lambda_1$, $P_2 \cdot \lambda_2$ e $P_T \cdot \lambda_T$ 107
4.3.2.4. Validação do modelo e resultados obtidos 108
4.3.2.4.1. Resultados obtidos para a Rede C 108
4.3.2.4.2. Resultados obtidos para a Rede S 109
4.4. Conclusões sobre os resultados obtidos 111
5. Conclusões 114
Referência Bibliográficas 118
Apêndice I – O Modelo de Simulação no ns-2 121

1. Introdução

A década de 90 foi marcante no âmbito das telecomunicações no Brasil e no mundo. O Brasil, no final daquela década, assistiu ao início do crescimento exponencial do número de aparelhos celulares móveis, que em agosto de 2003 atingiu a impressionante marca de 40,09 milhões, ultrapassando os 39,10 milhões de telefones fixos [1]. No final do ano de 2005, o número de aparelhos celulares móveis já atingia o dobro do número de fixos [1]. No mundo, o número de telefones celulares móveis já ultrapassa a marca de dois bilhões, ou seja, um aparelho celular para cada três habitantes do planeta [2]. A Figura 1.1 mostra o crescimento do número de telefones celulares móveis no Brasil [1].

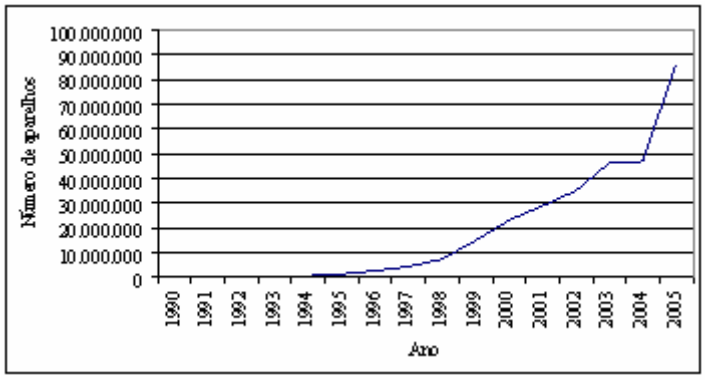


Figura 1.1 – Número de aparelhos celulares móveis no Brasil. Fonte: [1].

Na mesma época, consolidava-se no Brasil o crescimento no acesso à maior rede de compartilhamento de informações do mundo, a Internet. Estima-se que no final de 2005 cerca de 25 milhões de brasileiros tinham acesso à rede mundial [3]. No mundo, estudos mostram que mais de 1 bilhão de pessoas acessam a Internet com alguma regularidade, o que representa cerca de 16% da população mundial [3]. A Figura 1.2 ilustra o crescimento da Internet no Brasil, através da evolução do número de *hosts* conectados à Internet neste país [3].

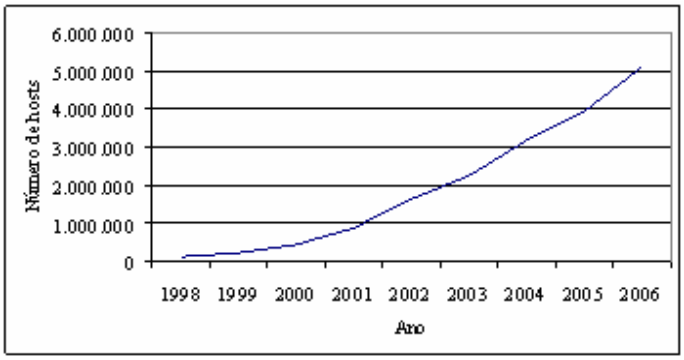


Figura 1.1 – Número de *hosts* conectados à Internet no Brasil. Fonte: [3].

Analisando os dados estatísticos do uso da telefonia móvel e do crescimento da Internet no Brasil e no mundo, imagina-se como natural a fusão destas duas tecnologias, com a utilização de tecnologia móvel celular (e de outras tecnologias de redes sem fio, como as redes satélite, por exemplo) servindo como meio de acesso à rede mundial. No entanto, este processo foi de certa forma lento. Um dos principais motivos dessa lentidão foi a dificuldade encontrada na adequação do principal protocolo de transporte da arquitetura TCP/IP (*Transmission Control Protocol/Internet Protocol*), o TCP, a algumas das características encontradas na rede celular e nas redes sem fio de forma geral, como, por exemplo, as altas taxas de erro de bit.

Desde o final da década de 90 vários estudos já buscavam qualificar e quantificar o impacto desse novo cenário do desempenho do protocolo TCP. A partir destes estudos, várias propostas de melhorias, tanto no sentido de modificações do TCP quanto no sentido da adequação dos protocolos utilizados nas redes sem fio, foram desenvolvidas como o objetivo de facilitar a introdução da transferência de dados neste novo ambiente.

O objetivo desta dissertação é descrever algumas das características adversas ao funcionamento do TCP presentes nas redes sem fio, assim como apresentar algumas das propostas de melhorias encontradas na literatura, desenvolvidas para solucionar cada um dos problemas abordados. Além disso, especificamente para a principal característica adversa ao TCP encontrada nos enlaces sem fio, as altas taxas de erros de bit, desenvolveu-se um modelo analítico para computar o desempenho do TCP quando protocolos de controle de erro baseados em retransmissão, como *SW-ARQ* (*Stop-and-Wait Automatic Repeat reQuest*) e *GBN-ARQ* (*Go-Back-N*), ou correção direta no receptor, o *FEC* (*Forward Error Correction*), são utilizados no enlace sem fio a fim de minimizar o impacto dessas altas taxas de erros de bit no desempenho final do TCP. Utilizando este modelo, são apresentados resultados analíticos do desempenho do TCP para dois tipos de rede, a primeira com características comuns a uma rede celular, e a segunda com características comuns a uma rede com enlace satélite tipo *GEO* (*Geosynchronous Earth Orbit*). Para cada rede e cada protocolo de controle de erro verificado, o desempenho do TCP é calculado levando em consideração a possibilidade ou não da estimativa da taxa de erro de bit no enlace sem fio. Nos casos onde esta estimativa é viável, os protocolos de controle de erro utilizam esta informação para melhorar seu desempenho e, por conseguinte o desempenho do TCP.

Para facilitar o entendimento dessa dissertação, ela será estruturada da seguinte forma: no Capítulo 2, serão mostradas as características principais do protocolo TCP; no Capítulo 3, algumas das características inerentes às redes sem fio, prejudiciais ao TCP, serão analisadas, assim como as principais soluções propostas para a adaptação do TCP a cada uma delas; no Capítulo 4 será descrito um modelo analítico capaz de computar o desempenho do TCP quando protocolos de camada de controle de erro são utilizados no enlace sem fio a fim de reduzir o impacto das altas taxas de erros de bit desses enlaces no desempenho da conexão TCP e, finalmente no Capítulo 5 serão mostradas as conclusões desta dissertação.

2. *Transmission Control Protocol*

O TCP (*Transmission Control Protocol*), inicialmente definido em [4], é hoje o protocolo de fato da camada de transporte do modelo de camadas TCP/IP (*Internet Protocol*) da Internet para transferência de dados orientados à conexão. Segundo as estimativas mais recentes [5], ele é responsável pelo controle de 80% dos fluxos fim-a-fim, pelo transporte de 90% dos pacotes e 95% dos bytes das redes de computadores atuais. Neste capítulo, apresenta-se um resumo da operação do protocolo TCP.

2.1. Características gerais

As principais características do TCP, de acordo com [6], são prover um serviço confiável, orientado à conexão e baseado em cadeias de bytes, às aplicações dos usuários da rede.

Ser orientado à conexão significa que duas aplicações que precisam transferir dados devem, anteriormente ao início dessa troca de informações, estabelecer uma conexão TCP. Esse estabelecimento é efetuado através de duas trocas de mensagens de sincronismo (SYN) e uma de reconhecimento, e é por esse motivo também conhecido como *three way handshaking*. O processo de abertura de conexão do TCP envolve a definição do valor de várias variáveis, e seu detalhamento não se faz necessário neste trabalho. Mais detalhes sobre abertura e também o fechamento de uma conexão podem ser obtidos em [4] e em [6].

Os dados provenientes das aplicações, a serem transmitidos pela rede, são divididos em unidades de informação, chamadas segmento. O TCP provê a confiabilidade na transmissão dos segmentos através da utilização de:

- Reconhecimentos: toda vez que o TCP recebe dados corretamente da outra parte envolvida na conexão, ele envia de volta uma mensagem chamada ACK (*Acknowledgment*).

- Temporizadores: toda vez que o TCP envia dados para a outra parte, ele inicia um temporizador, e aguarda o reconhecimento dos dados transmitidos. Caso o reconhecimento não chegue dentro de um tempo pré-determinado, um *timeout* ocorre e o segmento com os dados é então retransmitido.

- *Checksum*: O cabeçalho dos segmentos TCP é responsável por todo o controle na transmissão dos dados. Um dos campos do cabeçalho é o *checksum*, que permite que o receptor verifique a integridade dos dados recebidos. Caso um segmento chegue com *checksum* inválido, ele é descartado pelo TCP e o ACK não é gerado. Este segmento será retransmitido, por exemplo, através de um *timeout* no transmissor dos dados.

- Reordenação: Como os segmentos normalmente são transmitidos utilizando o protocolo IP que é não orientado à conexão, estes não necessariamente chegam na mesma ordem em que foram transmitidos. O receptor TCP é responsável por reordenar segmentos, caso isso seja necessário, entregando-os em ordem à aplicação.

- Descarte de segmentos duplicados: Toda vez que um segmento é recebido de forma duplicada, este é descartado.

- Controle de fluxo: O TCP receptor possui um *buffer* limitado para a recepção dos segmentos. É importante, portanto, prevenir que o transmissor continue enviando dados quando o *buffer* de recepção estiver cheio. Esta função é denominada controle de fluxo e é implementada utilizando o campo *Window Size* (*win*) do cabeçalho do TCP (descrito na seção a seguir).

- Controle de Congestionamento: O controle de congestionamento tem por função evitar que a rede fique sobrecarregada, o que resultaria no descarte de pacotes e conseqüente retransmissão dos mesmos. Os mecanismos de controle de fluxo e de congestionamento são particularmente importantes para o objetivo desta dissertação e são analisados em detalhe nas Seções 2.7 e 2.8, respectivamente.

O TCP considera que os dados são compostos de cadeias de bytes. Isto significa que um transmissor TCP envia segmentos com quantidades variáveis de bytes (com uma limitação máxima), mas que são transformadas na recepção em uma cadeia única, sem que a aplicação receptora consiga distinguir em que segmento cada byte foi transmitido. Além disso, o TCP não faz distinção do tipo de dado que está sendo transportado. Cabe às aplicações interpretar a cadeia de bytes recebida entre dados binários, ASCII ou qualquer que seja.

2.2. O cabeçalho TCP

A Figura 2.1 mostra o formato do cabeçalho TCP. Seu tamanho usual é de 20 bytes, a menos que o campo *Options* seja utilizado.

Cada segmento TCP contém dois campos *Port Number* (16 bits), um de origem e outro de destino, para que as aplicações que utilizam o TCP possam ser identificadas.

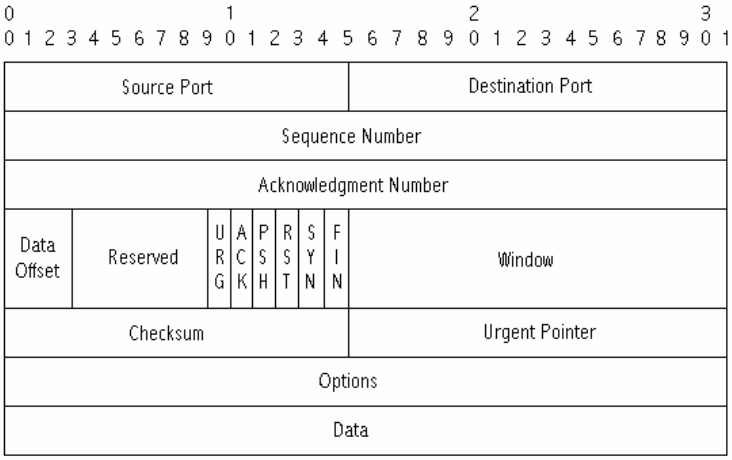


Figura 2.1 – O cabeçalho TCP. Fonte: [6].

O campo *Sequence Number* (32 bits) identifica o número de seqüência do primeiro byte, entre todos os bytes da cadeia enviada pelo transmissor, presente no referido segmento TCP. Como todos os bytes enviados são numerados, o campo *Acknowledgment Number* (32 bits) identifica o valor do próximo *sequence number* que o receptor espera receber. O valor do *Acknowledgment Number* deve ser, portanto, igual ao valor do último byte recebido com sucesso, acrescido de 1.

O campo *Data Offset* (4 bits), identifica o tamanho do cabeçalho TCP, ou seja, em qual ponto a partir do início do cabeçalho se encontram os dados da aplicação contidos no segmento. Este campo é necessário pois o campo *Options* tem tamanho variável (como descrito a seguir) tornando variável o tamanho do cabeçalho.

Após um campo reservado (6 bits) para uso futuro, aparecem as seis *flags* (1 bit cada) do TCP, sendo elas:

- URG: indica que o modo de urgência está ativado.
- ACK: indica que o valor do *acknowledgment number* deve ser considerado.

- PSH: indica que o receptor deve passar o conteúdo do segmento o mais rápido possível.
- RST: utilizado para reiniciar uma conexão.
- SYN: utilizado para indicar a sincronização de numeração no início da conexão.
- FIN: transmissor indica fim da transmissão de dados

O campo *Window* ou *Window Size* (16 bits) indica a quantidade de bytes que o receptor pode receber em um determinado momento. Este valor é utilizado pelo transmissor no controle de fluxo do TCP (Seção 2.7).

O campo *Checksum* (16 bits) é utilizado pelo receptor para verificar a integridade de todo o segmento (não só o cabeçalho) TCP. Como dito anteriormente, segmentos com *Checksum* inválido são descartados.

O campo *Urgent Pointer* (16 bits), válido somente com o modo de urgência ativado (*flag* URG), aponta o último byte dos dados urgentes dentro do segmento. O modo de urgência é o modo com que o transmissor envia dados urgentes para o receptor.

A utilização mais comum para o campo *Options* (tamanho variável) é para a definição do *Maximum Segment Size* (MSS) que é mostrado na seção seguinte.

Mais detalhes sobre o cabeçalho TCP e a funcionalidade de cada campo podem ser obtidos em [6].

2.3. MSS e janela de recepção

Como dito anteriormente, antes de qualquer transmissão de dados ser iniciada, o transmissor TCP necessita abrir uma conexão com a outra parte, o receptor. Durante essa abertura algumas variáveis são definidas, dentre as quais se destacam o MSS (*Maximun Segment Size*), que define o tamanho máximo de segmento (em bytes) que o receptor pode receber, e o tamanho inicial da janela de recepção, ou seja, o tamanho inicial (em bytes) do *buffer* de recepção. Convenciona-se neste trabalho, a partir desse ponto, que todo segmento transmitido tem tamanho igual ao MSS, salvo quando especificado o contrário. Da mesma forma, o *buffer* de recepção, cujo tamanho corrente é chamado de janela de recepção, é dado em número de

segmentos, ao invés de bytes, como é de fato feito pelo TCP. Essas simplificações facilitam o entendimento do funcionamento do fluxo de dados do TCP.

2.4. ACKs cumulativos

No TCP, mais de um segmento pode ser enviado em seqüência (até um limite imposto pelo tamanho da janela de recepção ou de congestionamento, como é visto adiante), sem a necessidade da espera da chegada dos ACKs dos segmentos anteriormente enviados. Isto possibilita a técnica de ACKs cumulativos. Com esta técnica, a chegada ao transmissor de um reconhecimento *K* implica que todos os segmentos anteriores a *K* foram recebidos com sucesso. Com isto, a perda de um ACK pode não resultar em retransmissão, caso um ACK de algum segmento posterior chegue corretamente ao transmissor antes de ocorrer o *timeout* (Seção 2.5) do segmento associado ao ACK perdido. A Figura 2.2 mostra o envio de 3 segmentos em seqüência, com a perda do ACK 3 (referente ao segundo segmento), sendo este reconhecido posteriormente através do terceiro e último ACK (ACK 4), que chega corretamente ao TCP transmissor.

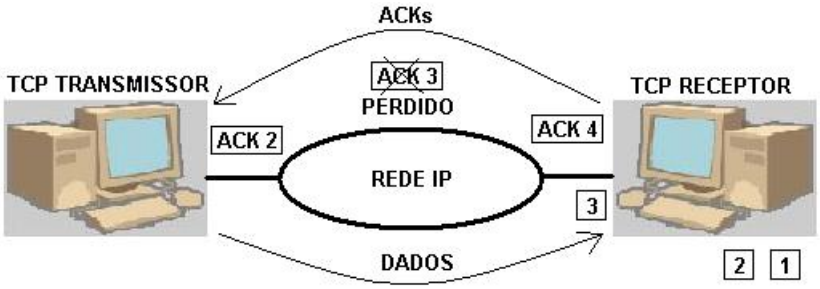


Figura 2.2 – ACKs Cumulativos.

2.5. Retransmissão por temporização

A retransmissão baseada em temporização é um mecanismo utilizado pelo TCP desde sua primeira versão, definido em [4]. Contudo, ao longo dos anos várias sugestões de melhorias foram feitas até que a proposta definida em [7] foi aceita, gerando as modificações definidas em [8]. Segundo esta última recomendação, o cálculo do temporizador deve ser feito baseado nos tempos de resposta (RTT – *Round-Trip Time*), que são os tempos medidos entre o envio dos segmentos e a chegada dos ACKs correspondentes, considerando ainda duas variáveis de estado: SRTT (*Smoothed Round-Trip Time*) e RTTVAR (*Round-Trip Time VARiation*). Quando uma primeira medida R de RTT é efetuada, O TCP atualiza os valores das variáveis de estado da seguinte forma:

$$RTTVAR \leftarrow R/2 \quad (2.1)$$

$$SRTT \leftarrow R \quad (2.2)$$

O valor do temporizador de retransmissão RTO (*Retransmission TimeOut*) é então calculado como:

$$RTO \leftarrow SRTT + 4 * RTTVAR \quad (2.3)$$

A partir da segunda medição R' de RTT, o cálculo das variáveis de estado se altera, passando a ser feito como mostrado abaixo:

$$RTTVAR \leftarrow (1 - \beta) * RTTVAR + \beta * |SRTT - R'| \quad (2.4)$$

$$SRTT \leftarrow (1 - \alpha) * SRTT + \alpha * R' \quad (2.5)$$

onde α e β são coeficientes de ponderação e segundo [8] devem possuir valores iguais a 0,125 e 0,25 respectivamente. Contudo, o cálculo de RTO continua sendo efetuado como anteriormente, mostrado na expressão (2.3).

De acordo com [7], efetuando o cálculo desta forma, o TCP é capaz de perceber pequenas e grandes variações de RTT, visto que o cálculo de RTO não só leva em consideração a média dos RTTs medidos (como era feito inicialmente em [4]), mas também sua variação ao longo do tempo.

Outra característica do cálculo do temporizador de retransmissão, mostrada em [8], é a presença do algoritmo de *backoff* exponencial. Toda vez que um *timeout* ocorre, o valor de RTO para o segmento retransmitido é dobrado, com o objetivo de fazer com que o segmento retransmitido consiga atravessar a rede, possivelmente congestionada, e chegar ao destino. Caso um novo *timeout* ocorra, o valor do temporizador é dobrado novamente e assim sucessivamente até um limite máximo, definido em 64 segundos em grande parte das implementações do TCP.

Um problema, denominado Problema de Ambigüidade de Retransmissão, pode ocorrer após um *timeout*, como descrito adiante. Como um segmento retransmitido é cópia fiel do segmento originalmente transmitido, o mesmo ocorre para os ACKs gerados para esses dois segmentos no receptor. Portanto, a chegada de um ACK, após uma retransmissão ter sido efetuada, pode representar um reconhecimento tardio do segmento enviado originalmente (devido a atrasos ocorridos na rede IP) ou ser apenas o reconhecimento do próprio segmento

retransmitido, pois, como dito anteriormente, não há distinção entre esses ACKs. Como resultado, o TCP poderia tomar a chegada do ACK do segmento original como base para calcular o RTT do segmento retransmitido, o que acarretaria em um erro no valor de RTT. De acordo com [9], a solução deste problema consiste em não se levar em consideração o RTT após a ocorrência de uma retransmissão, mantendo-se o valor do RTO para o segmento seguinte. Com a transmissão de um novo segmento e uma nova medida de RTT efetuada, o cálculo dos valores de SRTT, RTTVAR e RTO devem ser efetuados respectivamente de acordo com as expressões (2.1), (2.2) e (2.3), como recomendado em [8].

2.6. *Fast Retransmission* (Retransmissão Rápida)

Até este ponto, a única forma apresentada para o TCP se recuperar de um segmento perdido na rede IP se dá através de *timeouts*. Em algumas implementações do TCP, como o TCP Tahoe [10], o estouro do temporizador é realmente a única forma do TCP iniciar a retransmissão de um segmento. Contudo, várias implementações do TCP, dentre elas a mais usada, o TCP Reno [10], possuiu um segundo mecanismo para identificar que um segmento foi perdido na rede e, como conseqüência, retransmitir o mesmo. Este mecanismo de retransmissão se baseia em ACKs duplicados, e sua operação só é possível devido a uma característica do TCP: toda vez que K segmentos são enviados, dentre os quais um segmento N ($N < K$) é perdido pela rede, todos os ACKs referentes aos segmentos M ($K \geq M > N$) reconhecem apenas até o segmento N, visto que o receptor não pode reconhecer os segmentos M, dada a característica de ACKs cumulativos do TCP. Dessa forma, a chegada desses múltiplos reconhecimentos do segmento N é utilizada para indicar ao transmissor um lacuna na recepção dos segmentos enviados, o que pode ser ocasionado por uma simples desordenação dos pacotes na camada de rede, ou seja, um segmento J ($J > N$) chega ao seu destino anteriormente à chegada do segmento N, ou pela falha na chegada do segmento N em questão. O TCP então, após a chegada de 4 ACKs iguais (número suficiente para desconsiderar a hipótese de desordenação na camada de rede), sendo 1 originalmente gerado pelo segmento $N - 1$ e os 3 seguintes sendo gerados pelos segmentos $N + 1$, $N + 2$ e $N + 3$, identifica a falha na transmissão do segmento N e inicia sua retransmissão. A retransmissão baseada em ACKs duplicados foi proposta em [7] com o nome de *Fast Retransmission*. A Figura 2.3 mostra um diagrama temporal para uma ocorrência de *Fast Retransmission*.

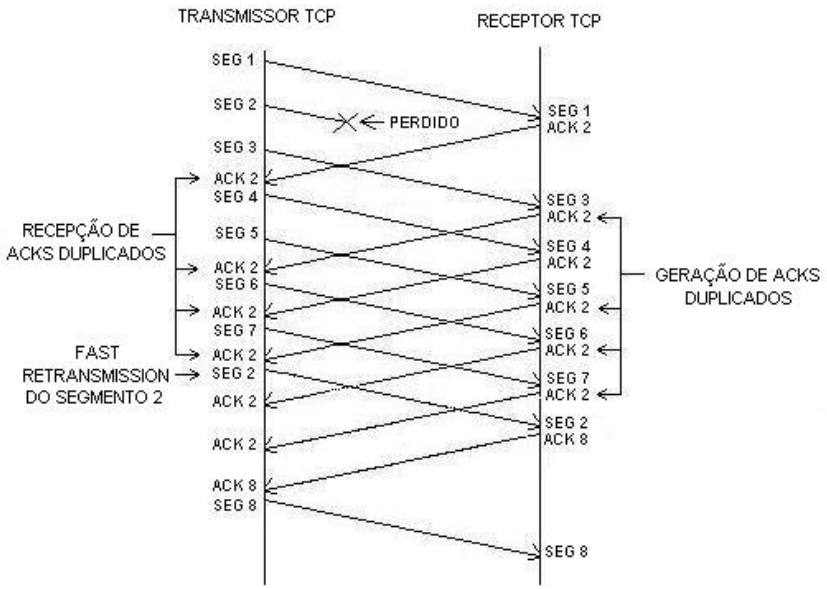


Figura 2.3 – *Fast Retransmission* do segmento 2.

2.7. Controle de fluxo

Como dito anteriormente, durante a abertura de uma conexão TCP, algumas variáveis são definidas entre transmissor e receptor, dentre as quais está o tamanho inicial da janela de recepção no receptor TCP. Cabe ressaltar que o tamanho dessa janela deve ser informado constantemente ao transmissor, e não somente no momento da abertura da conexão. Isto se dá devido ao fato do *buffer* de recepção não ser constantemente lido – e, por conseguinte esvaziado – pela camada de aplicação do receptor. Na verdade, esta leitura é feita de tempos em tempos, levando a uma variação no tamanho na janela de recepção e, consequentemente, à necessidade do transmissor ser sempre informado do tamanho corrente da janela de recepção, de modo a não transmitir mais dados do que o receptor possa receber. Este mecanismo de controle de fluxo é denominado de protocolo de janela deslizante e está ilustrado na Figura 2.4. O retângulo indica a janela oferecida pelo receptor (*offered window*), que engloba os segmentos de 4 a 9 (tamanho 6). Isto significa que os segmentos de 1 a 3 já foram enviados e seus respectivos reconhecimentos já foram recebidos pelo transmissor. Dentro desta janela, os segmentos de 4 a 6 já foram transmitidos mas seus respectivos ACKs ainda não chegaram ao transmissor. Os segmentos de 7 a 9 ainda não foram transmitidos, mas estão “liberados” para tal, constituindo a janela utilizável.

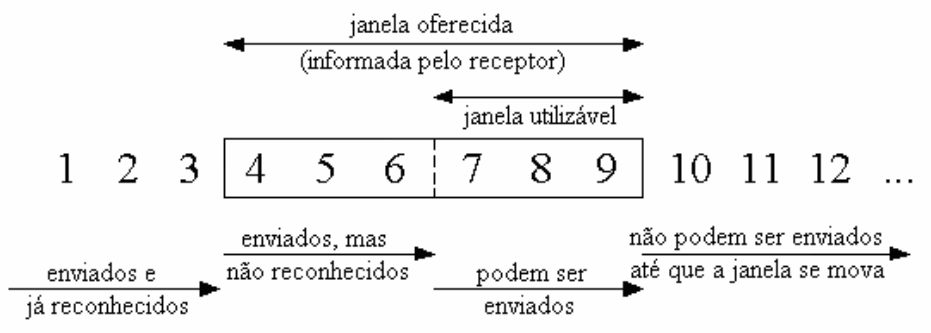


Figura 2.4 – Janela Deslizante. Fonte: [6].

Com a chegada dos ACKs, a janela deslizante move-se para a direita, fazendo assim com que novos segmentos possam ser enviados. O movimento relativo das laterais da janela faz com que esta aumente ou diminua de tamanho. Basicamente, três termos são definidos para descrever os movimentos da janela, mostrados na Figura 2.5.

- A janela fecha quando a lateral esquerda move-se para a direita. Isto ocorre quando segmentos são enviados e reconhecidos;
- A janela abre quando a lateral direita move-se para a direita, permitindo que mais dados possam ser transmitidos. Isto ocorre quando a aplicação no receptor lê os dados recebidos, tirando-os do *buffer* de recepção e, por conseguinte, liberando mais espaço;
- A janela encolhe quando a lateral direita move-se para a esquerda. Apesar de [34] desencorajar fortemente este tipo de movimento, o TCP deve saber lidar com ele, caso ocorra. Detalhes sobre o movimento de encolhimento da janela podem ser vistos em [6].

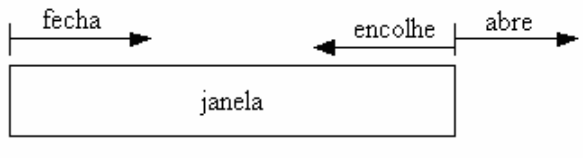


Figura 2.5 – Movimentos das laterais da janela. Fonte: [6].

A forma com que o receptor informa ao transmissor o tamanho da janela de recepção é enviando este valor no campo *Window Size* (Seção 2.2) do cabeçalho dos segmentos, ACKs ou outros segmentos de dados, destinados ao transmissor. Desta forma, o transmissor atualiza o valor da janela de recepção (*win*) a cada segmento recebido.

Durante uma conexão TCP, em um determinado momento, o receptor pode indicar uma janela de recepção de tamanho 0 ao transmissor (quando a lateral esquerda alcança a lateral direita na Figura 2.5, por exemplo), fazendo com que esse interrompa a transferência de dados. Neste caso, o transmissor retém o valor da janela de congestionamento (*cwnd*) assim como de RTO e das variáveis de estado SRTT e RTTVAR, esperando que o receptor aumente o valor de *win* assim que seu *buffer* puder receber mais dados. Essa atualização do valor da janela de recepção é feita através do re-envio de um ACK do último segmento recebido, agora com o valor atual de *win.* Contudo, assim como qualquer segmento TCP enviado numa rede IP, este pode ser perdido, devido a congestionamento, por exemplo. Caso isso aconteça, o transmissor TCP continua esperando pela atualização da janela de recepção, que já foi efetuada, e o receptor TCP, apesar de pronto para receber mais dados, jamais os recebe. Para prevenir a ocorrência deste *deadlock* o transmissor utiliza um *persist timer* que faz com que ele examine a janela de recepção periodicamente através do envio de um segmento chamado *window probe*. O transmissor responde a esse *probe* da mesma forma com que faz a atualização da janela de retransmissão: reenvia um ACK do último segmento recebido com o valor atual da janela de recepção. Caso o transmissor não receba resposta do receptor, ou ainda o valor da janela de recepção continue sendo 0, o transmissor aumenta o tempo de espera até o envio do próximo *window probe* através de *backoff* exponencial, exatamente como visto no cálculo de RTO. Vale ressaltar que *probes* perdidos na rede não fazem com que o transmissor invoque seu protocolo de congestionamento, ou seja, o TCP transmissor fica realmente retido em seu estado anterior à chegada do ACK com janela de recepção de tamanho 0.

2.8. Controle de congestionamento

O controle de congestionamento, que é sem dúvida o mais complexo dos algoritmos do TCP, tem como preocupação evitar a sobrecarga de dados na rede que transporta os segmentos. Seu funcionamento baseia-se no envio dos dados de acordo com uma janela de congestionamento mantida pelo transmissor que varia de tamanho de acordo com a recepção dos ACKs durante a conexão.

Quando uma nova conexão é iniciada e o transmissor possui dados a serem enviados para o receptor, o controle de congestionamento entra na sua fase inicial, a chamada *Slow Start*. Nesta fase, a janela de congestionamento (*cwnd*) começa com valor unitário, e o TCP então envia um segmento para o receptor. Com a chegada do ACK deste segmento antes do tempo

de RTO, o TCP percebe que a rede está funcionando corretamente e aumenta o valor da janela para 2 segmentos. O TCP agora envia estes dois segmentos e espera pelos ACKs. Caso os ACKs cheguem corretamente, o TCP aumenta a janela para 4, em seguida 8, 16 e assim sucessivamente caso todas as transmissões sejam efetuadas com sucesso. Portanto, nesta fase cada ACK recebido faz com que *cwnd* aumente seu tamanho em 1 segmento, caracterizando um crescimento exponencial, que apesar do nome *Slow Start* (partida lenta), é bem rápido. A janela então cresce até um limiar, chamado *slow start threshold* (*ssthresh*), que possui um valor inicial que varia com cada implementação do TCP. Ao atingir esse limiar, o TCP entende que a continuação do crescimento exponencial levaria provavelmente a rede a uma situação de congestionamento. A transmissão então entra na segunda fase, a chamada *Congestion Avoidance*, na qual *cwnd* não mais cresce de 1 segmento a cada ACK recebido e sim a cada conjunto de ACKs que reconhece a janela de congestionamento por inteiro. Isto quer dizer, por exemplo, que se em algum momento a janela de retransmissão tem valor 8, são necessários os 8 ACKs destes segmentos para que a janela de congestionamento passe então ao tamanho de 9 segmentos.

A ocorrência de um *timeout,* em qualquer fase, é interpretada pelo TCP como descarte de segmento devido à ocorrência de congestionamento na rede, indicando que a taxa de transmissão de segmentos deve ser reduzida abruptamente. Quando isto ocorre, o TCP vai para a fase de *Slow Start* inicial, ou seja, o valor da janela *cwnd* volta a ser 1. O novo valor de *ssthresh* é ajustado então para a metade do valor de *cwnd* no momento da ocorrência do *timeout*. O TCP novamente cresce exponencialmente a cada ACK recebido, até a janela atingir o novo valor de *ssthresh* quando entra novamente na fase de *Congestion Avoidance*, de crescimento linear. Quando há a detecção de perda de segmento por meio de ACKs duplicados (*Fast Retransmission*), a janela de transmissão não mais volta a fase de *Slow Start*. Neste caso, a janela de congestionamento é reduzida à metade, e o TCP permanece na fase de crescimento linear (*Congestion Avoidance*), caracterizando o algoritmo de *Fast Recovery*. O crescimento linear segue novamente até a ocorrência de outro *Fast Retransmission/Fast Recovery*, permanecendo na fase de *Congestion Avoidance*, ou até a ocorrência de um *timeout,* que leva o TCP novamente para a fase de *Slow Start*.

É importante dizer que os algoritmos de controle de fluxo (descritos na Seção 2.7) e controle de congestionamento funcionam paralelamente. O número de segmentos a serem transmitidos (janela de transmissão) é de fato o valor mínimo entre *win* e *cwnd*. Dessa forma o TCP

respeita tanto a capacidade do receptor de receber os dados quanto a capacidade da rede IP de encaminhar os segmentos.

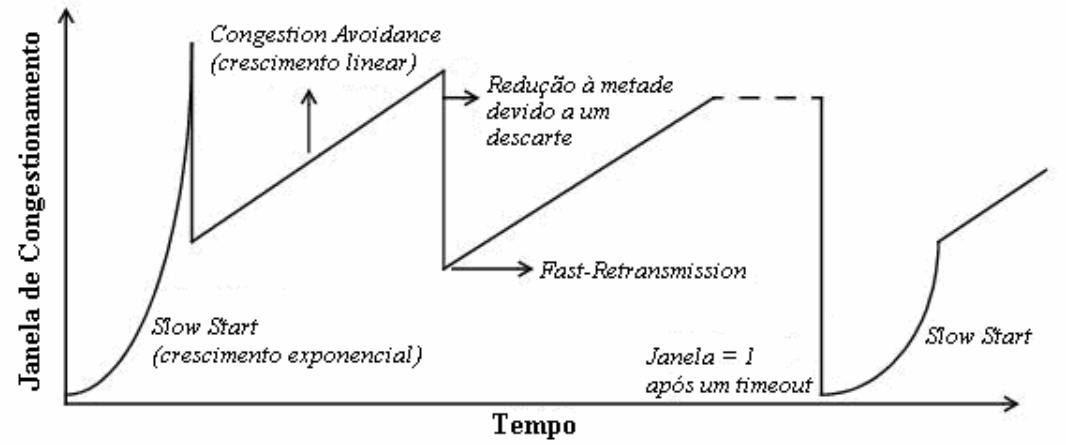


Figura 2.6 - Evolução dinâmica da janela de congestionamento do TCP. Fonte: [11].

A Figura 2.6 mostra a evolução dinâmica da janela de congestionamento do TCP para uma conexão já estabelecida. Num primeiro instante a janela tem tamanho unitário e o seu crescimento é exponencial. A janela cresce até a ocorrência de um evento de *Fast Retransmission*, quando seu tamanho cai pela metade e o seu crescimento passa a ser linear, caracterizando a fase de *Congestion Avoidance*. Em seguida, uma nova diminuição do tamanho de *cwnd* para a metade, mas ainda com crescimento linear, caracterizando novamente o *Fast Retransmission/Fast Recovery*. A janela continua crescendo, dessa vez até a ocorrência de um *timeout*, que faz o TCP passar a fase de *Slow Start* com janela de tamanho unitário. A janela cresce então exponencialmente até que o limiar *ssthresh* – que possui valor igual à metade do valor de *cwnd* no momento da ocorrência do *timeout* – seja atingido, quando passa a ter novamente crescimento linear, na fase de *Congestion Avoidance*.

Vistos todos os mecanismos de controle do TCP, pode-se afirmar que o funcionamento do controle de congestionamento é especialmente dependente do tráfego encontrado na rede IP. Se o tráfego é excessivo, implicando em descarte de pacotes na rede, o TCP tende a frear o envio de segmentos para a rede, de forma a aliviá-la. Caso o TCP identifique, através da chegada sucessiva dos ACKs, que a rede não mais está sobrecarregada, sua janela novamente cresce, fazendo com que mais dados sejam injetados na rede, até que novamente ela entre em colapso, quando o TCP irá novamente reduzir a taxa de envio dos segmentos na rede.

3. O TCP e as redes sem fio

A principal característica adversa ao desempenho do TCP, como visto no capítulo anterior, é a presença de congestionamento. Nas redes com fio isso se torna particularmente verdade, visto que não há nenhum outro tipo de impedimento para o crescimento da janela de congestionamento e, por conseqüência, para o crescimento da taxa de transmissão de segmentos (desconsiderando o limite imposto pelo receptor através do controle de fluxo).

Com o crescimento da utilização de redes com enlaces sem fio, cujas características são, em muitos casos, prejudiciais ao desempenho do TCP, faz-se necessária a implementação de novos mecanismos para que o desempenho do protocolo não degrade significativamente. Algumas das características deste novo contexto são estudadas neste capítulo. A primeira delas, abordada na Seção 3.1, é a presença de altas taxas de erros de bit (BER – *Bit Error Rate*) nos enlaces sem fio, que levam o TCP transmissor a uma redução na taxa de envio de segmentos e, por conseguinte, na diminuição de seu desempenho. Isto se deve ao fato de o TCP considerar toda perda de segmento como sendo devido a um congestionamento na rede, não considerando a possibilidade de descarte de segmentos por erros de bit. A segunda característica, abordada na Seção 3.2, é a ocorrência de perdas intermitentes de sinal entre as extremidades do enlace sem fio, gerando perdas de segmento. Essas perdas novamente são erroneamente interpretadas como perdas por congestionamento, degradando o desempenho do TCP. A terceira característica, mostrada na Seção 3.3, é a presença de assimetria de largura de banda entre as transferências no sentido transmissor-receptor (sentido direto) e no sentido receptor-transmissor (sentido reverso). Em redes com alta assimetria de largura de banda o desempenho do TCP é degradado devido a uma imperfeição na realimentação do transmissor em relação aos ACKs do receptor. A quarta e última característica adversa estudada, abordada na Seção 3.4, é a presença de longos atrasos de propagação em alguns enlaces sem fio (particularmente nas redes de comunicação por satélite). Essa característica faz com que o TCP, ao iniciar uma transmissão, demore um longo período até que a taxa de transmissão alcance a vazão oferecida pela rede, fazendo principalmente com que conexões com transferência de pequenas quantidades de dados tenham desempenho ruim.

Para cada característica adversa detalhada nas seções subseqüentes, algumas soluções propostas na literatura são apresentadas.

3.1. Redes com altas BERs

O TCP, ao receber um segmento, verifica, através do campo *checksum* (Seção 2.2), a existência de erros de bit. Caso o segmento de dados seja recebido com erro, este é descartado e o ACK correspondente não é gerado. Este fato pode resultar em dois fenômenos: na ocorrência de um *timeout* no transmissor ou, caso segmentos subseqüentes cheguem corretamente ao receptor, na geração de ACKs duplicados. Em ambos os casos isto resulta no acionamento dos mecanismos de controle de congestionamento no TCP transmissor, pois o mesmo interpreta toda perda de segmento (sinalizada pela ocorrência de *timeout* ou pelo recebimento de ACKs duplicados) como sendo devido a congestionamento. No caso da ocorrência de um *timeout*, o TCP reduz sua janela de congestionamento para o tamanho unitário e entra na fase de *Slow Start*; no caso do recebimento de ACKs duplicados, a janela é reduzida à metade e o TCP entra na fase de *Congestion Avoidance*. Nos dois casos, a diminuição da janela de congestionamento leva a uma diminuição da quantidade de dados injetada na rede. Logo, verifica-se que o descarte de um segmento com erro resulta na piora do desempenho da conexão TCP.

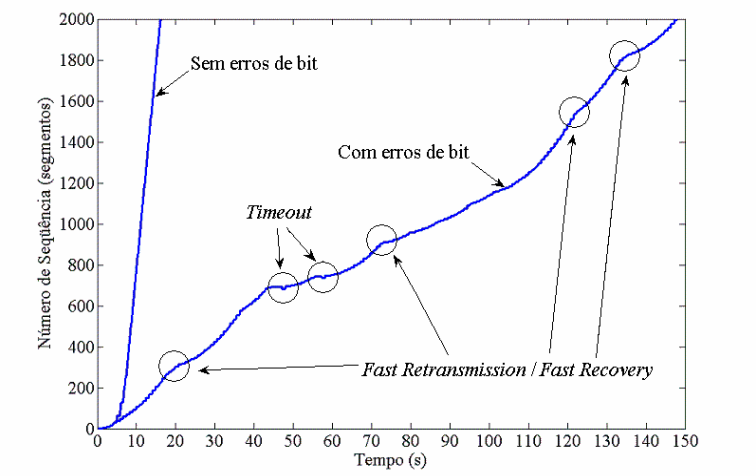


Figura 3.1 – Evolução seqüencial do TCP.

A Figura 3.1 compara o desempenho do TCP-Reno na rede sem fio da Figura 3.2 (na qual não há congestionamento) quando da ocorrência de erros de bit (curva à direita) com a mesma situação quando da não ocorrência dos erros de bit (curva à esquerda). O parâmetro considerado é o número de segmentos transmitidos (representado pelo número de seqüência dos segmentos TCP) por unidade de tempo.

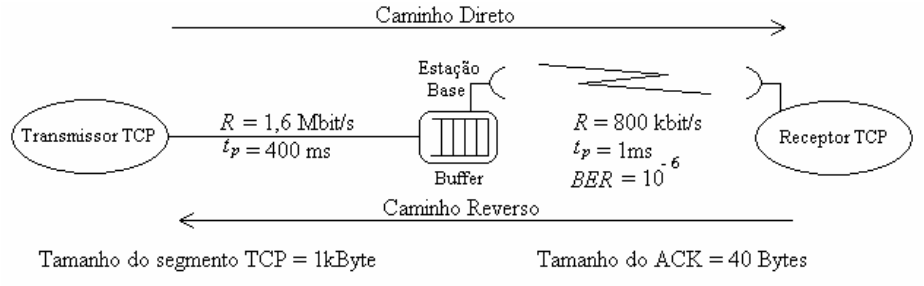


Figura 3.2 – Rede com enlace sem fio.

O desempenho da curva à esquerda mostra-se superior, pois neste caso não há a ocorrência de *timeouts*, representados pelas lacunas encontradas na curva à direita, nem tampouco a ocorrência de *Fast Retransmission/Fast Recovery*, representado pelas reduções na inclinação da curva. Em função do grande tamanho do *buffer* (escolhido de forma que não haja descarte durante os experimentos) não há congestionamento e a ocorrência de *Fast Retransmission/Fast Recovery* se dá apenas devido ao descarte de segmentos com erros de bit. Neste experimento, por exemplo, na situação em que há erros de bit, a transferência de dados é aproximadamente 8 vezes mais lenta do que quando estes erros não estão presentes.

A seguir apresentam-se algumas soluções propostas para contornar o problema de taxa de erro de bit elevada no enlace sem fio.

3.1.1. Protocolos de camada de enlace

De acordo com [11], os protocolos de camada de enlace são convencionalmente utilizados para lidar com os erros de bit decorrentes da utilização de enlaces sem fio. Estes protocolos protegem o TCP dos erros de bit encontrados nesses enlaces através da utilização de recuperações locais, fazendo com que a taxa de erro de bit vista pelo TCP seja efetivamente mais baixa.

O grande atrativo para a utilização dos protocolos de camada de enlace é o seu encaixe natural dentro da estrutura de camadas dos protocolos de rede. O controle de erro no canal pode ser feito por meio de detecção de erro e retransmissão, denominado *ARQ* (*Automatic Repeat reQuest*), por meio de correção automática no receptor, denominado *FEC* (*Forward Error Correction*) ou por meio de protocolos híbridos que combinam estes dois mecanismos.

3.1.1.1. Códigos de controle de erro

Em qualquer dos mecanismos de controle de erro, é necessária a utilização de um código de controle de erro (codificação de canal), onde bits de redundância são acrescentados aos bits de informação para permitir a detecção (*ARQ*) ou a correção (*FEC*) dos erros. O codificador de canal transforma a seqüência de bits de informação em uma seqüência de bits codificada, chamada palavra-código.

Dentre os códigos de controle de erro destacam-se os códigos de bloco lineares, representados por (n,k). Nestes, o codificador transforma, segundo regras específicas para cada tipo de código, um bloco contendo k bits de informação (vetor-mensagem) em uma palavra código (ou vetor-código) contendo n bits, adicionando, portanto, $n - k$ bits de redundância (também chamados bits de paridade). Um código de bloco só é dito linear se seu conjunto de vetores-código contém o vetor nulo e se e somente se a soma em módulo-2 de duas palavras-código quaisquer resultar em uma outra palavra-código.

3.1.1.2. Capacidade de detecção e correção de erros

O peso de Hamming de um vetor código é definido como sendo o número de bits iguais a 1 no vetor e a distância de Hamming entre dois vetores é definida como o número de posições nas quais os vetores diferem. Por exemplo, a distância de Hamming entre os vetores $\mathbf{U}$ = [100101101] e $\mathbf{V}$ = [011110100] é d($\mathbf{U}$,$\mathbf{V}$) = 6.

A distância mínima de um código é a menor distância de Hamming no conjunto de distâncias calculadas tomando-se todos os pares de palavras-código. Ou seja, se a distância mínima de um código é d_{min}, a distância de Hamming entre quaisquer duas palavras-código é pelo menos igual a d_{min}. Observa-se que, pela propriedade da linearidade, a distância mínima do código é igual ao mínimo peso de Hamming das palavras código. A distância mínima é uma característica de cada código e define sua capacidade de detecção e correção de erros [12].

Um código com distância mínima d_{min} é capaz de corrigir todos os padrões de erro que afetem até t bits no bloco de n bits, com $t = \lfloor (d_{min} - 1) / 2 \rfloor$. O parâmetro t é denominado capacidade (de correção de erro) do código. Deve-se observar que, usualmente, o código é também capaz de corrigir alguns padrões de erro que afetem $t + 1$ (ou mais) erros [12].

Assim, para um código de bloco com capacidade *t*, desprezando os eventuais padrões de erro corrigíveis que afetam mais de *t* bits, a probabilidade de um bloco de *n* bits conter erros que podem ser corrigidos automaticamente é dada pela distribuição binomial

$$P_{ccf} = \sum_{i=0}^{t} \binom{n}{i} p^i (1-p)^{n-i} \ , \tag{3.1}$$

onde *p* é a probabilidade de erro de bit no canal, suposto sem memória e simétrico.

No decodificador, há a possibilidade de se receber 2^n vetores diferentes, uma vez que o vetor-código transmitido pode ser alterado no canal. O vetor recebido, **U'**, pode ser visto como a soma do vetor código transmitido, **U**, com um vetor que represente o erro do canal, **E** (se **E** = 0 então não houve erro no canal). O vetor recebido é detectado como contendo erro se ele difere de um dos 2^k vetores-código possíveis. Devido à propriedade da linearidade do código conclui-se que o código não é capaz de detectar o erro apenas se o padrão de erro coincidir com uma das $2^k - 1$ palavras-código não nulas. Ainda, pode-se afirmar que o código será capaz de detectar todos os padrões de erro que afetem $d_{min} - 1$ ou menos bits.

A probabilidade de um erro não ser detectado no decodificador é igual a probabilidade de ocorrer um padrão de erro igual a uma palavra-código. Se A_j é o número de palavras-código com peso de Hamming *j* no código, os números A_0, A_1, A_2, ..., A_n são chamados de distribuição de peso do código, e a probabilidade de ocorrer um erro não detectável, admitindo-se um canal BSC (*Binary Symmetric Channel*), é dada por [12]

$$P_{ue} = \sum_{j=1}^{n} A_j p^j (1-p)^{n-j} \ . \tag{3.2}$$

3.1.1.3. Limitantes para o cálculo da capacidade de correção e detecção de erros

Em muitas situações deseja-se estimar o número de bits de paridade necessário no código para se obter uma determinada probabilidade de detecção ou correção de erro. Para encontrar este número é necessário definir um tipo de código e, a partir do conhecimento da distribuição de peso (caso o parâmetro fixado for a probabilidade de detecção de erro) ou da distância mínima (caso o parâmetro fixado for a probabilidade de correção de erro), investigar os diversos códigos de uma família de códigos conhecida, em busca daquele que satisfaz o parâmetro fixado com menor número de bits de redundância. Essa tarefa muitas vezes não é simples, uma vez que as informações de distribuição de peso ou mesmo de distância mínima nem

sempre estão disponíveis para uma determinada família de códigos. Para facilitar esta tarefa, vários limitantes foram desenvolvidos, dentre os quais se destacam dois: o limitante de Korzhik e o limitante de Plotkin.

O limitante de Korzhik é utilizado para estimar o número de bits de paridade necessário para se obter uma dada probabilidade de não se detectar um erro. Ele estabelece que existe um código de bloco (n,k) cuja probabilidade de erro não detectável satisfaz a seguinte inequação [13]

$$P_{ue} \leq 2^{-(n-k)}(1-(1-p)^n)) \quad . \tag{3.3}$$

Se o parâmetro de desempenho fixado é a probabilidade de correção de erro de bit, que é refletida na distância mínima necessária ao código, o limitante de Plotkin pode ser usado. Ele permite calcular, dado o valor da distância mínima d_{min} desejada, qual o mínimo valor de $n-k$ através da seguinte inequação [14]

$$n-k \geq 2(d_{min}-1)-\log_2 d_{min} \quad . \tag{3.4}$$

3.1.1.4. Técnica *FEC*

Na técnica *FEC*, o transmissor adiciona à mensagem bits de paridade suficientes para um nível predeterminado de capacidade de correção de erros. No receptor, o *FEC* encaminha a mensagem para as camadas superiores em duas situações: caso o bloco tenha sido recebido corretamente ou caso ele tenha sido recebido com erro, mas se a mensagem foi devidamente corrigida. A eficiência do protocolo *FEC*, de acordo com [12], é calculada por

$$\eta_{fec} = \frac{k}{n_f} \ , \tag{3.5}$$

onde n_f é o tamanho do bloco quando o mecanismo *FEC* é utilizado.

A probabilidade de o protocolo *FEC* entregar um bloco sem erros para as camadas superiores equivale à probabilidade do *FEC* conseguir corrigir um bloco, dado por (3.1).

3.1.1.5. Protocolos *ARQ*

Os esquemas *ARQ* usualmente utilizam retransmissões baseadas em três protocolos: *Stop-and-Wait* (*SW*), *Go-Back-N* (*GBN*) e *Selective Repeat* (*SR*).

No *SW*, o transmissor envia um bloco e interrompe a transmissão, à espera do reconhecimento do bloco transmitido. Se o reconhecimento é positivo, indicando que não houve erro detectável na transmissão, um novo bloco é enviado; se o reconhecimento é negativo, o bloco é retransmitido. Sua eficiência, de acordo com [12], pode ser calculada como

$$\eta_{sw} = \frac{k \cdot P_c}{n_a + R \cdot \tau_b} \quad , \tag{3.6}$$

onde n_a representa o tamanho do bloco quando o mecanismo *ARQ* é utilizado, R representa a taxa do enlace dada em bits por unidade de tempo, τ_b representa os atrasos do enlace e P_c é a probabilidade do bloco recebido não conter erros, dado por

$$P_c = (1 - BER)^{n_a} \quad . \tag{3.7}$$

No *GBN*, o transmissor envia seus blocos de dados continuamente (usualmente há um limite, imposto pelo protocolo da camada de controle do enlace, no número de blocos que podem ser enviados sem que um reconhecimento seja recebido). Quando o receptor recebe um bloco errado, este envia uma mensagem de reconhecimento negativo indicando o número do bloco que continha erro (usualmente o receptor é obrigado a enviar um reconhecimento, mesmo que nenhum bloco tenha sido recebido com erro, para não ultrapassar os limites do protocolo da camada de enlace). Ao receber o reconhecimento negativo, o transmissor volta ao bloco que foi recebido com erro e retransmite todos os blocos a partir daquele. Sua eficiência, de acordo com [12], pode ser escrita como

$$\eta_{gbn} = \frac{k \cdot P_c}{n_a + R \cdot \tau_b \cdot P_c} \quad . \tag{3.8}$$

O protocolo *SR* assemelha-se ao *GBN*, consistindo a diferença no fato do transmissor reenviar apenas os blocos detectados como errados (e não todos os blocos a partir deste). Claramente, o protocolo *SR* apresenta maior eficiência do que o protocolo *GBN,* que por sua vez tem eficiência maior do que o *SW*. Sua eficiência, de acordo com [12], é dada por

$$\eta_{sr} = \frac{k \cdot P_c}{n_a} \quad . \tag{3.9}$$

Uma das características do *SR* é a necessidade de reordenação dos segmentos por parte do receptor, característica essa que faz com que a utilização dos protocolos *GBN* e *SW* seja mais atraente nas redes TCP/IP. Isto ocorre, pois a chegada de segmentos de forma não seqüencial

pode levar o TCP a executar retransmissões desnecessárias, como visto no Capítulo anterior, degradando seu desempenho.

A probabilidade de o protocolo *ARQ* entregar um bloco sem erros para as camadas superiores é dada por [15]

$$P_{cca} = P_c + P_c \cdot \sum_{k=1}^{\infty} P_e^k = \frac{P_c}{1-P_e} = \frac{P_c}{P_c + P_{ue}} \tag{3.10}$$

onde P_c é definido em (3.7), P_e é a probabilidade do bloco conter um erro detectável e P_{ue} é a probabilidade do bloco conter um erro não detectável. Obviamente, essas probabilidades estão relacionadas através da expressão

$$P_c + P_e + P_{ue} = 1. \tag{3.11}$$

3.1.1.6. Esquemas híbridos

Os protocolos híbridos são aqueles que combinam as técnicas *ARQ* e *FEC*. Por exemplo, no sistema híbrido tipo 1 [16], dois códigos são utilizados. O quadro de k bits é codificado, utilizando-se um código (n_a, k) para detectar erros no bloco, gerando uma palavra código de n_a bits. Esta palavra código é, por sua vez, codificada utilizando-se um código (n_f, n_a) para corrigir erros. Quando o bloco resultante chega ao receptor, ele primeiro é decodificado utilizando-se o decodificador *FEC*. A mensagem de n_a bits é então enviada ao decodificador *ARQ* para verificar se há erro residual. Se um erro é detectado, o bloco é descartado e uma retransmissão é solicitada.

Outro exemplo é o esquema híbrido *ARQ-FEC* com múltiplos estados *FEC* [15]. Nesse esquema, o sistema comuta do estado *ARQ* para o estado *FEC* quando um bloco é recebido com erro (e a retransmissão solicitada) e só comuta do estado *FEC* de volta para o estado *ARQ* depois de um determinado número y de blocos recebidos com sucesso. O diagrama de estados desse sistema é mostrado na Figura 3.3.

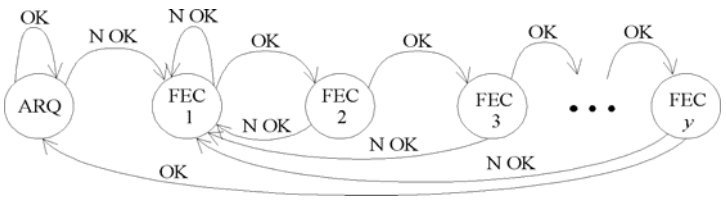


Figura 3.3 – Diagrama de estados do sistema híbrido de múltiplos estados *FEC*. Fonte: [15].

De acordo com [15], a probabilidade do sistema se encontrar no estado *ARQ* é dada por

$$P_{arq} = \frac{P_{cf}^{y}}{P_{cf}^{y} + P_e \cdot \sum_{j=0}^{y-1} P_{cf}^{j}} \quad . \tag{3.12}$$

A eficiência do protocolo híbrido *ARQ-FEC* com múltiplos estados *FEC*, derivada da expressão mostrada em [15], pode ser escrita como

$$\eta_h = \frac{k}{\left[n_a(P_c) + \left(n_a + R \cdot \tau + n_f\right)\left(1 - P_c\right)\right] P_{arq} + n_f\left(1 - P_{arq}\right)} \quad . \tag{3.13}$$

A probabilidade do protocolo híbrido *ARQ-FEC* com múltiplos estados *FEC* entregar um bloco sem erros para as camadas superiores é dado por [15]

$$P_{cch} = \left(P_c + P_e \cdot P_{ccf}\right) \cdot P_{arq} + P_{ccf} \cdot \left(1 - P_{arq}\right) \tag{3.14}$$

onde P_c é dado por (3.7) e representa a probabilidade de um bloco não conter erros no estado *ARQ*, P_{ccf} é dado por (3.1) e representa a probabilidade de um bloco conter um erro detectável no estado *FEC*, P_e é dado através de (3.11) e representa a probabilidade de um bloco conter um erro detectável no estado *ARQ* e P_{arq} é dado por (3.12).

3.1.1.7. Interações adversas entre os protocolos de camada de enlace e de transporte

Existem ocasiões em que a transmissão fim a fim não é melhorada quando da utilização de protocolos de camada de enlace convencionais. Existem basicamente três formas de interações adversas entre as camadas de transporte e enlace:

- Interações temporais: temporizadores independentes nas duas camadas podem resultar em transmissões redundantes nas duas camadas [17]; dessa forma, a camada de transporte não é protegida pela camada de enlace. Especificamente, o TCP não sofre com este problema devido à forma conservativa com que o valor do temporizador de retransmissão é calculado (Seção 2.5);
- Interações de *Fast Retransmission*: este tipo de interação ocorre quando o protocolo de camada de enlace provê confiabilidade através de retransmissões, contudo não preserva a ordem seqüencial da entrega dos segmentos TCP. Dessa forma a chegada de segmentos posteriores ao esperado causam a geração de ACKs duplicados, que farão com que a janela seja reduzida, reduzindo portanto o *throughput*. Protocolos de enlace deste tipo não protegem, de fato, o TCP dos erros de bit [11];

- Grandes variações de tempo de resposta: caso o protocolo de enlace seja projetado de forma muito robusta, como é, por exemplo, o protocolo de transporte TCP, grandes latências no envio dos segmentos, assim como uma grande variância neste atraso, serão vistos pelo TCP transmissor. Dessa forma, RTOs extremamente conservadores serão gerados o que acarretará em grandes esperas quando da ocorrência de perdas na rede.

Pode-se notar, portanto, que apesar de os protocolos de enlace proverem a recuperação local dos erros de bit, eles nem sempre protegem efetivamente o TCP destes erros. Quando esta proteção não ocorre, o TCP tem seu desempenho reduzido.

3.1.2. O protocolo Snoop

O protocolo Snoop, solução proposta em [11] para os erros de bit encontrados nos enlaces sem fio, consiste em fazer com que o TCP não invoque o controle de congestionamento para segmentos descartados devido a este tipo de erro. Dessa forma, o TCP não diminui o tamanho da janela, e, portanto, não reduz de forma incorreta a taxa de transmissão de segmentos.

Para verificar o funcionamento do protocolo Snoop, faz-se necessário entender a estrutura topológica sobre a qual ele foi projetado, típica nas redes atuais com enlaces sem fio. A Figura 3.4 mostra esta estrutura e seus principais componentes: um Host Fixo (HF) e um Host Móvel (HM) – as estações que transferem dados através da conexão TCP; uma rede com fio que transporta os dados do HF até a Estação Base (EB), estação essa que faz a interface entre a rede com fio e o enlace sem fio, onde predominantemente ocorrem os erros de bit [11].

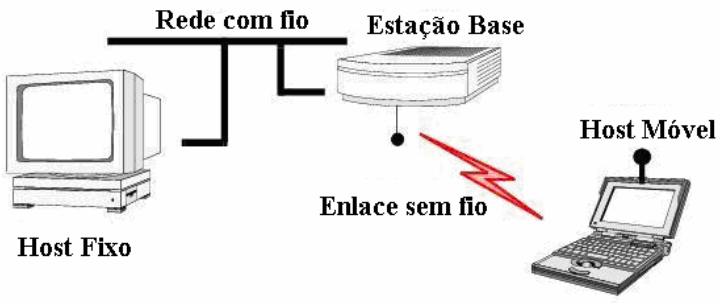


Figura 3.4 – Topologia do sistema abordado pelo Snoop. Fonte: [11].

O protocolo Snoop baseia-se na idéia de que os erros de bit causados no enlace sem fio consistem num problema local e devem ser resolvidos como tal, preservando a largura de

banda da rede com fio dos problemas causados pelo enlace sem fio. Uma solução baseada em retransmissões locais pode vir a sofrer dos problemas descritos na Seção 3.1.1.6, portanto o protocolo Snoop busca evitar as interações adversas já citadas.

Apesar de o protocolo Snoop ter sido desenvolvido com o objetivo de melhorar o desempenho fim-a-fim das transferências de dados tanto no sentido do HF para o HM, quanto do HM para o HF, os mecanismos usados em cada um dos casos são distintos. Portanto, a descrição deste protocolo é subdividida nos dois possíveis sentidos da transferência de dados.

3.1.2.1. Transferência de dados no sentido HF para HM

Na transmissão do HF para o HM o protocolo Snoop melhora o desempenho do TCP através da utilização de um agente na EB, chamado agente Snoop. Este agente monitora todos os segmentos que atravessam a conexão em ambos os sentidos e é responsável pela retransmissão rápida e local efetuada no enlace sem fio, baseada nas informações dos ACKs provenientes do receptor (HM). O princípio de funcionamento do protocolo Snoop pode ser resumido da seguinte forma: todos os pacotes que chegam à EB necessariamente atravessam o agente, visto que este se situa entre as camadas de rede (IP) e de enlace (embora ele possa visualizar o conteúdo da camada de transporte, o que o autor chama de "*TCP aware link layer protocol*"). Quando um segmento de dados chega à EB, ela o adiciona a um *cache* e em seguida o passa normalmente para a camada de enlace, responsável pela transmissão para o HM. Paralelamente, o agente monitora os ACKs gerados pelo HM, utilizando-os para esvaziar o *cache*, mantendo guardados, desta forma, somente os segmentos ainda não reconhecidos. Quando ACKs duplicados chegam à EB ou ainda quando há estouro de um temporizador local, o agente retransmite – com alta prioridade – os segmentos perdidos, caso estes estejam no *cache*. Adicionalmente, ele elimina os ACKs duplicados correspondentes às perdas por erros de bit, impedindo que o TCP transmissor invoque, de forma equivocada, o controle de congestionamento.

A geração de ACKs duplicados referentes a um segmento ausente no *cache* da EB, implica em uma provável perda na rede com fio, possivelmente devido a congestionamento na mesma (contudo existe a possibilidade, ainda que remota, de um descarte por erro de bit na rede com fio). Nesse caso, o agente Snoop encaminha normalmente os ACKs duplicados para o HF, visto que seus algoritmos de controle de congestionamento devem ser executados.

Com a utilização do agente Snoop, o protocolo "esconde" do TCP transmissor as perdas de pacotes por erros de bit, mas ao mesmo tempo garante que o HF continue reagindo adequadamente a um possível congestionamento na rede com fio.

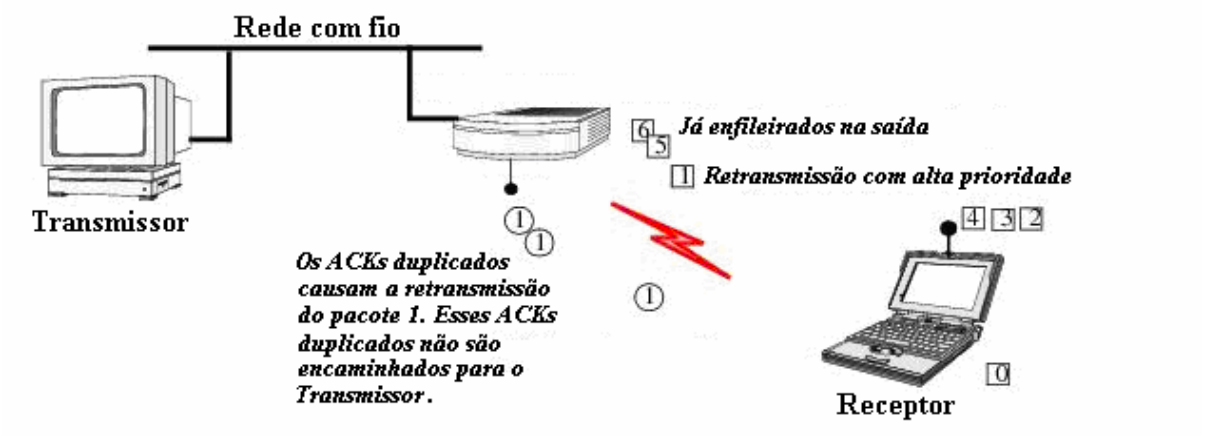


Figura 3.5 – Protocolo Snoop com transmissão no sentido HF para HM. Fonte: [11].

A Figura 3.5 ilustra a atuação do protocolo Snoop na ocorrência de descarte de um segmento por erro de bit. O HF envia sete segmentos que atravessam a rede com fio e chegam com sucesso à EB. Na estação base, estes são adicionados ao *cache* e em seguida encaminhados para o HM. O segmento 0 é transmitido corretamente no enlace sem fio, enquanto o segmento 1 é corrompido. Como os segmentos 2, 3 e 4 chegam ao HM corretamente, ACKs duplicados referentes ao segmento 1 são gerados. Quando a EB recebe estes ACKs duplicados, esta confere se o segmento indicado está presente no *cache.* Como o segmento 1 está armazenado na EB, os ACKs duplicados não são encaminhados para o HF e o segmento 1 é retransmitido com prioridade sobre a transmissão dos segmentos 5 e 6.

3.1.2.2. Transferência de dados no sentido HM para HF

Prover um bom desempenho neste caso, utilizando somente as modificações na EB mostradas na seção anterior para transmissões no sentido HF para HM, segundo [11], é muito difícil, se não impossível. Guardar os segmentos em *cache* na EB, por exemplo, não faz sentido dado que os erros de bit ocorrem na transmissão entre o HM e a EB.

A idéia então é fazer com que o transmissor saiba "distinguir" segmentos descartados por erros de bit de segmentos perdidos por congestionamento, invocando esquemas de

recuperação adequados a cada um dos casos. No protocolo Snoop, um mecanismo chamado *Explicit Loss Notification* (ELN) é utilizado para informar ao transmissor a causa da perda do segmento.

A EB, ao invés de manter um *cache* dos segmentos recebidos e encaminhados, mantém uma lista de "buracos" referente às chegadas de segmentos não seqüenciais. Esses buracos devem corresponder aos segmentos por ela descartados devido a erros de bit. Entretanto, é importante discernir entre os segmentos realmente descartados por erros de bit daqueles perdidos por congestionamento na entrada da EB. Para tanto, um "buraco" só é adicionado à lista caso a fila de chegada da EB não estiver perto de seu tamanho máximo.

Quando ACKs duplicados provenientes do receptor, o HF, chegam à EB, esta verifica sua lista de "buracos". Caso o ACK corresponda a um segmento presente na sua lista, a EB marca um bit de ELN no cabeçalho TCP deste ACK (usado o campo reservado para uso futuro, visto na Seção 2.2, por exemplo) antes de repassá-lo ao HM. A EB aproveita também para limpar todos os "buracos" associados a segmentos com número de seqüência inferior ao confirmado pelo ACK, dado que tais segmentos já foram recebidos com sucesso. Quando o transmissor recebe os ACKs duplicados com a informação de ELN marcada, ele retransmite o segmento, mas não invoca o controle de congestionamento, dessa forma melhorando o desempenho do TCP.

O transmissor toma também o cuidado de, após a retransmissão de um segmento marcado com ELN, esperar um intervalo de tempo igual a um tempo de resposta antes de transmitir novamente este segmento. Isto é necessário dado que a EB marca e repassa todos os ACKs duplicados para o transmissor fazendo com que vários ACKs duplicados referentes ao mesmo segmento perdido possam chegar ao transmissor em seqüência. Aguardando este intervalo, o transmissor evita retransmissões repetidas e desnecessárias referentes a essa rajada de ACKs duplicados.

3.2. Redes com perdas intermitentes de sinal

Outro problema das redes sem fio são as perdas intermitentes de sinal entre o HM e a EB, particularmente quando os usuários são dotados de mobilidade. Novamente o desempenho do TCP é prejudicado, pois essas perdas de sinal geram perdas de segmentos, que serão

interpretadas como congestionamento pelo TCP padrão, fazendo com que este invoque os protocolos de controle de forma incorreta. A Figura 3.6 mostra o resultado do crescimento seqüencial dos segmentos enviados quando há perdas intermitentes comparado com a situação onde não há essas perdas intermitentes de sinal. A rede utilizada para a obtenção das curvas é a mesma da Figura 3.2, com as perdas intermitentes inseridas no enlace sem fio. O processo para a obtenção das curvas foi o de simulação no ns-2.

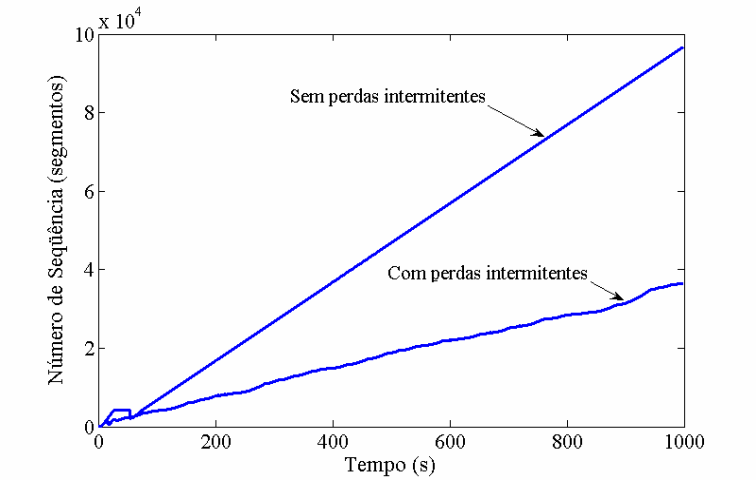


Figura 3.6 – Evolução seqüencial do TCP quando das perdas intermitentes de sinal.

3.2.1. O *Freeze*–TCP

O *Freeze*–TCP, proposto em [18], é um protocolo desenvolvido de forma a minimizar os efeitos das constantes perdas de sinal verificadas em algumas redes sem fio. A estrutura topológica de seu funcionamento é a mesma mostrada na Figura 3.4, contudo, o *Freeze*–TCP só aborda transmissão de dados no sentido HF para HM.

No *Freeze*-TCP todo o ônus de sinalizações relativas às perdas de sinal é passado ao HM e, devido a este fato, somente o protocolo TCP receptor é alterado. Seu funcionamento baseia-se na idéia de que todo HM monitora (ou pode monitorar) constantemente a qualidade do sinal proveniente da EB e, portanto, pode definir padrões para saber quando uma perda de sinal está iminente. Desta forma, antes que uma perda de sinal efetivamente ocorra, o HM solicita ao transmissor que "trave" sua janela de transmissão e teste a conectividade entre ambos até que esta novamente esteja estabelecida. Isto é feito utilizando o envio de um ou mais ACKs indicando janela de recepção de tamanho 0 – um ZWA, *Zero Window Advertisement* – por parte do receptor (como visto na Seção 2.7). Com a chegada do ZWA ao transmissor, este "congela" seu estado, e começa a enviar *probes* – ZWPs, *Zero Window Probes* – para

verificar se o tamanho da janela foi atualizado pelo receptor. Caso o receptor responda positivamente a um ZWP indicando que o sinal entre a EB e o HM já está estabelecido, a transmissão dos dados é reiniciada, sem que a janela de congestionamento tenha seu tamanho reduzido. Vale ressaltar que o envio de ZWPs por parte do transmissor segue o algoritmo de *backoff* exponencial, visto na Seção 2.5.

A questão chave no *Freeze*–TCP é definir quanto tempo antes da ocorrência de uma perda de sinal o receptor deve enviar um ZWA para o transmissor. Idealmente, este período, chamado *warning period*, deve ser grande apenas o suficiente para garantir que um, e somente um, ZWA chegue até o transmissor. Períodos maiores fazem com que o transmissor "congele" sua janela prematuramente, deixando de enviar segmentos de dados e, por conseguinte, diminuindo o desempenho da conexão. Em contrapartida, se o *warning period* for muito pequeno, o ZWA pode não alcançar o transmissor. Neste caso a transmissão não é congelada, e todos os problemas referentes à perda de segmentos durante a perda de sinal irão ocorrer, diminuindo drasticamente o desempenho do TCP. Dado estes fatos, segundo [18], um valor razoável para o *warning period* é o valor de RTT. Dessa forma, há tempo suficiente para o transmissor receber os ACKs referentes aos segmentos transmitidos exatamente antes do início do *warning period.* Um *warning period* com valor superior ou inferior ao RTT, segundo resultados experimentais, leva o *Freeze*–TCP a desempenhos inferiores.

Um problema que ainda pode acometer o *Freeze*–TCP se dá quando a conexão do HM à EB é restabelecida imediatamente após o envio de um ZWP por parte do transmissor. Devido à característica de *backoff* exponencial entre envios de ZWPs, o tempo até que o transmissor envie outro ZWP pode ser de até 64 segundos (limite máximo do *backoff* exponencial na maioria das implementações do TCP), fazendo com que a conexão fique desnecessariamente inoperante durante todo esse intervalo. Para aliviar este problema, toda vez que o HM percebe o restabelecimento do sinal com a EB, este envia três cópias do ACK referente ao último segmento recebido com sucesso, fazendo com que o TCP transmissor, ao recebê-los, entre imediatamente em *Fast Retransmission*/*Fast Recovery*, diminuindo consideravelmente o tempo inoperante pós restabelecimento de sinal.

3.2.2. O ATCP

Como visto na seção anterior, o *Freeze*–TCP é restrito a melhorias na transmissão no sentido HF para HM. Diferentemente, a proposta apresentada em [19], o ATCP, envolve melhoria da transmissão em ambos sentidos. Assim como no *Freeze*-TCP, o ATCP também passa o ônus destas melhorias para o HM, pois considera que este sempre possui (ou pode possuir) informações sobre o sinal de conexão com a EB. Desta forma, também no ATCP, somente o TCP do HM é modificado.

Como as modificações propostas pelo ATCP dependem do sentido da transmissão, esta seção é subdividida nos dois possíveis casos.

3.2.2.1 Transferência de dados no sentido HM para HF

O HM pode medir o nível de sinal proveniente da EB e, através desta medida, definir dois eventos: evento de perda de sinal, quando o HM deixa de receber o sinal proveniente da EB e o evento de recuperação de sinal, quando o HM volta a receber o sinal da EB. Além de agir explicitamente na ocorrência desses dois eventos, o ATCP também trata o evento de *timeout* de uma forma diferente do TCP. De fato, o ATCP provê a melhora no desempenho da transmissão alterando a forma com que o TCP padrão trata o RTO e as variáveis de janela (*cwnd e ssthresh*) de acordo com a ocorrência dos eventos de perda de sinal, recuperação de sinal e *timeout*.

Na ocorrência do evento de perda de sinal, caso a janela de transmissão esteja aberta, ou seja, existam segmentos ainda a serem enviados, o TCP transmissor interrompe a transmissão e deixa de esperar pelos ACKs enviados antes da perda de sinal, cancelando o tempo de RTO; caso a janela esteja fechada, ou seja, caso não haja segmentos a serem enviados e o TCP transmissor esteja apenas esperando pelos ACKs referentes aos segmentos já enviados, nenhuma ação é tomada e um evento de *timeout* pode vir a ocorrer durante o período em que o HM não recebe sinal da EB.

Quando há um evento de recuperação de sinal, caso a janela esteja aberta, o ATCP transmite os segmentos ainda não enviados, definindo novos RTOs para os mesmos. Já que os ACKs do TCP são cumulativos (Seção 2.4), todos os segmentos enviados anteriormente à perda de sinal, caso tenham chegado com sucesso ao receptor, são confirmados pelos ACKs dos novos

segmentos enviados. Se a janela estiver fechada e um evento de *timeout* não tiver ocorrido, o ATCP espera até que ele porventura aconteça.

Na ocorrência de um evento de *timeout*, o ATCP verifica se houve alguma perda de sinal durante o intervalo de RTO. Caso tenha ocorrido, ao invés de reduzir o valor de *ssthresh* pela metade (como faria o TCP), o ATCP o ajusta para o valor de *cwnd* no instante da perda de sinal e ajusta *cwnd* para o valor unitário. Dessa forma, o ATCP faz com que o valor de *cwnd* cresça rapidamente (exponencialmente) até seu valor no momento anterior à perda de sinal, diminuindo assim os efeitos das perdas intermitentes de sinal. Caso ocorra um *timeout*, mas nenhum evento de perda de sinal tenha ocorrido, o TCP age normalmente, pois segmentos podem ter sido perdidos na rede com fio e, nesse caso, os algoritmos de controle de congestionamento do TCP devem ser invocados.

3.2.2.2 Transferência de dados no sentido HF para HM

No ATCP, toda vez que o HM recebe segmentos sem erros de bit provenientes do HF ele imediatamente gera os reconhecimentos para os mesmos, excetuando-se pelos dois últimos bytes do último segmento recebido[1]. De fato, as confirmações para esse bytes são atrasadas em um curto período de tempo, de forma que não haja a ocorrência de *timeout* no TCP transmissor[2]. Com este atraso de confirmações, a probabilidade do ATCP possuir pelo menos dois bytes ainda não reconhecidos no caso da ocorrência de uma perda de sinal é aumentada.

Quando do evento de recuperação de sinal (após a ocorrência de uma perda de sinal), o ATCP receptor utiliza esses dois últimos bytes para enviar em seqüência dois ACKs diferenciados: um ZWA, que "congela" o estado do transmissor (como visto na Seção 3.2.1), e um FWA (*Full Window Advertisement*) que retoma a transmissão de dados com o valor da janela de recepção indicando seu valor máximo.

Segundo [20], quando um transmissor recebe um ZWA, este deve desconsiderar todos os valores de RTO para os segmentos transmitidos antes da chegada do ZWA. O ATCP se beneficia dessa característica, fazendo com que após a chegada dos ACKs ZWA e FWA,

[1] Isto só é possível, pois, na verdade, os ACKs do TCP confirmam a cadeia de bytes presente em cada segmento, e não o próprio segmento, permitindo, dessa forma, a exclusão de alguns bytes. Mais informações podem ser encontradas em [6].

[2] Para mais informações, consultar *Delayed Acknowledgements* em [6]

novos RTOs sejam calculados para os segmentos possivelmente perdidos durante o período de perda de sinal. Dessa forma, essas possíveis perdas não acarretam na diminuição da janela de congestionamento, não degradando, portanto, o desempenho do TCP.

3.3. Redes com assimetria de largura de banda

Uma das características de algumas redes atuais é a presença de assimetria. De acordo com [11], uma rede exibe assimetria com respeito ao desempenho do TCP se o *throughput* alcançado não é somente função do enlace e das características de tráfego no sentido direto (transmissor → receptor), mas também depende significativamente desses fatores no sentido reverso (receptor → transmissor).

As redes assimétricas de interesse neste estudo são aquelas que possuem assimetria de largura de banda nas transmissões entre HF e HM. Na prática, a transmissão no sentido HF para HM tende a ter taxas mais altas do que no sentido contrário e isto ocorre basicamente devido a dois fatores: a característica do tráfego *Web* atual, com predominância do tráfego no sentido de *downlink* (HF para HM) sobre o tráfego no sentido de *uplink* (HM para HF); a restrição de potência do sinal enviado pelos HMs – normalmente alimentados por baterias – que deve ser compensada com menores taxas de transmissão para que níveis menores de taxa de erros de bit sejam atingidos [21].

Os efeitos de assimetria afetam o TCP devido a sua característica de funcionamento baseado na recepção dos ACKs, como visto na Seção 2.8. Dessa forma, qualquer degradação no processo de realimentação do transmissor pelos ACKs do receptor limita o desempenho da transmissão dos dados.

Em redes com largura de banda limitada no sentido reverso, duas situações podem vir a ocorrer. Na primeira, a fila presente no enlace de saída do HM é grande o suficiente para que ACKs não sejam descartados. Neste caso, o desempenho é função do chamado *fator de largura de banda normalizado* [22], c, definido como a razão das larguras de bandas no sentido direto e reverso, dividida pela razão dos respectivos tamanhos de pacote. Se $c > 1$, e os ACKs não são perdidos, o funcionamento do TCP baseado em ACKs, mais especificamente sua temporização (ACK *clocking*), não funciona corretamente. Para ilustrar este fato, pode-se considerar dois pacotes sendo seguidamente transmitidos para o receptor.

No caminho direto, estes pacotes são espaçados de acordo com o "gargalo" encontrado no sentido direto. O princípio do ACK *clocking* é que os ACKs gerados como resposta ao envio dos pacotes preservam o espaçamento temporal durante todo o caminho no sentido reverso, para que o transmissor possa transmitir novos segmentos com o mesmo espaçamento. Para ilustrar a quebra da temporização, pode-se considerar dois ACKs sendo seguidamente gerados pelo HM e encaminhados para o transmissor. Como $c > 1$, os ACKs são gerados numa taxa superior àquela que enlace de saída do HM pode suportar, fazendo com que estes sejam enfileirados em seqüência. Como o tempo de transmissão nesse ponto é superior a todos os tempos de transmissão encontrados no sentido direto (visto que esse é o enlace gargalo), o espaçamento temporal quando eles atravessam o enlace HM → EB é aumentado, sendo este aumento função da largura de banda desse enlace. Nesse caso, o transmissor envia novos segmentos com uma taxa inferior à que seria utilizada caso não tivesse ocorrido o aumento no espaçamento temporal entre ACKs. Desta forma, o desempenho do TCP não mais depende apenas das características de tráfego no sentido direto, sendo regulado também pela taxa de chegada dos ACKs, que é função de um enlace do sentido reverso.

A Figura 3.7 ilustra o comportamento do *goodput* normalizado (normalizado com relação ao caso em que não há assimetria, por exemplo, quando $c = 1$) da rede da Figura 3.2, obtido através de simulação no ns-2, quando se altera o valor da taxa de transmissão do enlace sem fio no caminho reverso para $R_R = (R_D \cdot L_R)/(L_D \cdot c)$, onde R_D = 800kbps representa a largura de banda no sentido direto (dado pela menor taxa de transmissão encontrada no sentido direto, a do enlace sem fio), $L_R = 40$ bytes representa o tamanho dos pacotes enviados no caminho reverso (no caso os ACKs), $L_D = 1000$ bytes representa o tamanho dos pacotes no caminho direto (no caso os segmentos TCP) e o valor de c será variado. O parâmetro R_R na verdade representa a largura de banda no sentido reverso, mas esta será dada pela taxa do enlace sem fio no sentido reverso, visto que a taxa do enlace com fio no sentido reverso tem valor superior. Observa-se que quanto maior o valor de c, menor o desempenho da rede.

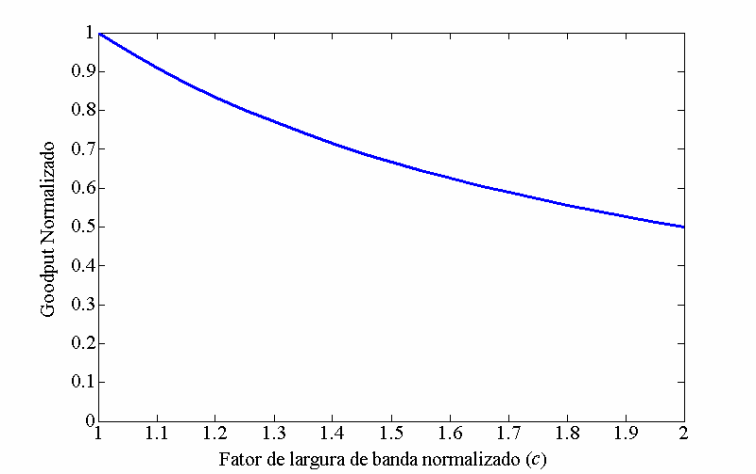


Figura 3.7 – *Goodput* normalizado para uma rede assimétrica com *buffer* infinito no caminho reverso.

A segunda situação ocorre quando a fila de saída do HM não é suficiente para acomodar todos os ACKs por ele gerados. Quando a janela de transmissão cresce, esta fila enche e a partir daí ACKs são descartados. Considerando que todos os segmentos geram reconhecimentos por parte do receptor, apenas um a cada c ACKs alcança o transmissor, enquanto os $(c - 1)$ restantes são descartados devido a estouro de *buffer* na fila de saída do HM. Neste caso, de acordo com [11], a menor largura de banda no sentido reverso e a demora na chegada dos ACKs não são diretamente responsáveis pela diminuição do desempenho do TCP. Contudo, essas características implicam em três fatores que levam à diminuição do desempenho, ocasionados pela freqüência alterada dos ACKs:

- Quando o transmissor recebe apenas um segmento a cada c, ele transmite dados em rajadas de c (ou mais) segmentos devido ao fato de a janela deslizante se abrir de pelo menos os c segmentos reconhecidos. Esta transferência em rajadas aumenta a probabilidade de segmentos serem perdidos no sentido direto devido ao fato de os roteadores da rede não lidarem satisfatoriamente com este tipo de tráfego.
- O TCP aumenta a janela de congestionamento baseado no número de ACKs recebidos, e não na quantidade de dados que foi realmente reconhecida pelo ACK. Desta forma, uma quantidade reduzida de ACKs faz com que a janela de congestionamento cresça com taxas inferiores, degradando o desempenho do TCP.
- Os algoritmos de *Fast Retransmission/Fast Recovery* são menos efetivos quando o número de ACKs é reduzido. Isto ocorre pois o transmissor pode não receber o número de ACKs duplicados necessários para ativar estes algoritmos, mesmo quando o receptor envia a quantidade necessária para tal.

Como os efeitos da assimetria nesse caso são “indiretos”, para visualizá-los faz-se necessário promover algumas modificações na rede da Figura 3.2. Considere que na EB existe um *buffer* limitado a 10 pacotes no sentido da transmissão do enlace sem fio, criando um gargalo com *buffer* limitado no sentido direto. Na saída do receptor TCP, é colocado um *buffer* com capacidade para 3 ACKs, limitando o *buffer* do sentido reverso. Quando não se reduz o valor da taxa de transmissão do enlace sem fio, tal que $c > 1$ e, portanto, não havendo assimetria, obtém-se a curva de crescimento seqüencial indicada na Figura 3.8. Quando se altera o valor da taxa de transmissão do enlace sem fio tal que $c = 2$ e, portanto, tem-se assimetria com *buffer* finito no caminho reverso, obtém-se o resultado também mostrado na Figura 3.8. Claramente, a presença de assimetria, mesmo quando o *buffer* é limitado, prejudica o desempenho do TCP.

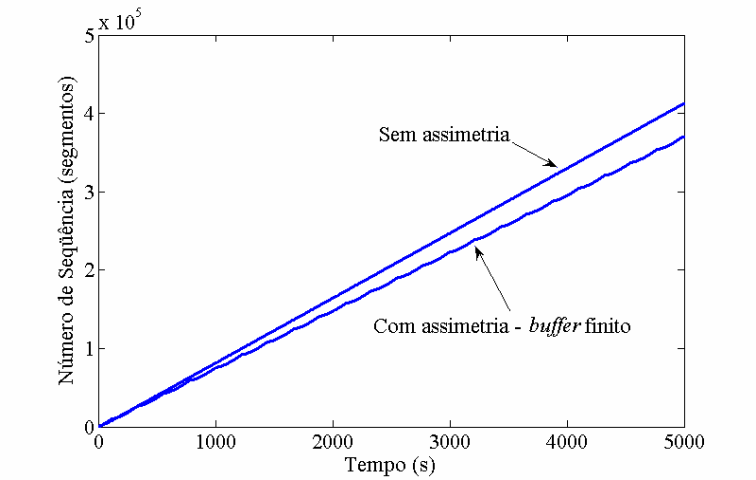


Figura 3.8 – Evolução seqüencial do TCP da para uma rede assimétrica com *buffer* finito no caminho reverso.

Em resumo, redes com assimetria de largura de banda sofrem com a degradação de desempenho devido a uma realimentação lenta e/ou imperfeita dos ACKs. Isto sugere uma solução em que a freqüência dos ACKs enviados pelo receptor seja reduzida, visto que essas imperfeições devem-se à grande quantidade de ACKs na rede. Entretanto, ainda é necessário alguma solução que lide com a redução da freqüência dos ACKs, dados os problemas que podem decorrer dessa redução. Juntas, soluções deste tipo aumentam o desempenho em redes com assimetria de largura de banda.

Em [11], é proposta a técnica de Filtragem de ACKs (*ACK Filtering*) para a redução da freqüência dos ACKs. Para lidar com a infreqüência dos ACKs duas soluções são

apresentadas: a Adaptação do Transmissor (*Sender Adaptation*) e a Reconstrução de ACKs (*ACK Reconstruction*).

3.3.1 – Filtragem de ACKs (FA)

A FA é considerada por seu proponente [11] como uma técnica de camada de enlace que é capaz de ler e interpretar o cabeçalho TCP (denominada "*TCP aware link layer technique*"), e que, através disto, reduz o número de ACKs enviados no caminho reverso. O desafio da FA é fazer com que o transmissor não fique paralisado, esperando por ACKs, o que poderia acontecer caso os ACKs fossem indiscriminadamente removidos. A idéia da FA é remover alguns ACKs sem prejudicar o TCP transmissor, levando em conta a característica de ACKs cumulativos (Seção 2.4).

Toda vez que um novo ACK chega à fila de saída do HM, este verifica a presença de ACKs referentes a segmentos anteriores ainda enfileirados. Se estes de fato existirem, ele os remove da fila, diminuindo a quantidade de ACKs redundantes enviados ao transmissor e esvaziando a fila para que esta possa receber o novo ACK. Dessa forma a FA faz com que ACKs referentes a segmentos posteriores jamais sejam enfileirados atrás de ACKs de segmentos anteriores.

3.3.2 – Adaptação do Transmissor (AT)

Como visto anteriormente, a FA alivia o problema de congestionamento no caminho reverso através da redução da freqüência dos ACKs. Contudo, esta é apenas a primeira parte de uma solução global para a melhoria do desempenho em uma rede com assimetria de largura de banda, visto que a simples redução da freqüência dos ACKs também causa problemas ao TCP, vistos na Seção 3.3. A AT lida com estes problemas alterando os algoritmos do TCP, adaptando-os à baixa freqüência dos ACKs.

Para resolver o problema da transmissão em rajadas, a AT estabelece um limite máximo de segmentos que podem ser transmitidos seguidamente, mesmo que a janela de transmissão permita a transmissão de mais dados. Desta forma, grandes rajadas são divididas em rajadas menores, aliviando a rede e resolvendo o primeiro dos problemas decorrentes da baixa freqüência dos ACKs.

A AT leva em consideração a quantidade de dados reconhecida por cada ACK, e não a quantidade de ACKs para efetuar o crescimento da janela de congestionamento. Dessa forma, se um ACK reconhece *u* segmentos, a janela de congestionamento cresce como se *u* segmentos tivessem chegado ao transmissor. Assim, o problema do crescimento da janela de congestionamento, segundo problema decorrente da infreqüência dos ACKs, é resolvido.

A terceira modificação proposta pela AT é a de não levar em conta somente o número de ACKs duplicados para efetuar os algoritmos de *Fast Retransmission/Fast Recovery*. Como a FA retira ACKs da fila de saída do HM, alguns destes podem ser ACKs duplicados, o que dificultaria a identificação de perda de segmentos no transmissor. Para sanar este problema, com a chegada de um novo ACK duplicado à fila e a retirada de um ACK duplicado anterior, a informação do número total de ACKs duplicados já retirados da fila é inserida no cabeçalho deste novo ACK duplicado (fazendo novamente dos bits reservados para uso futuro, vistos na Seção 2.2, por exemplo). A AT retira então esta informação dos ACKs que chegam ao TCP transmissor para executar os algoritmos de *Fast Retransmission/Fast Recovery* de forma apropriada.

3.3.3 – Reconstrução de ACKs (RA)

Assim como a FA, a RA é uma técnica de camada de enlace capaz de ler e interpretar o cabeçalho TCP, que reconstrói a cadeia de ACKs modificada pela FA, prevenindo a degradação do TCP padrão pela diminuição da freqüência dos ACKs. Com a utilização da RA, os algoritmos do TCP do transmissor não precisam ser alterados, o que não ocorre quando a técnica de AT é utilizada.

A RA utiliza um agente chamado reconstrutor de ACKs situado na EB. Este agente cuidadosamente preenche os "buracos" da seqüência proveniente da FA com novos ACKs, suavizando a seqüência de ACKs vista pelo transmissor.

Sejam dois ACKs *a1* e *a2*, por exemplo, que chegam ao reconstrutor após atravessar a FA e o enlace HM → EB. Seja ainda $a2 - a1 = \Delta a > 1$. Caso *a2* chegue ao transmissor após *a1* sem ACKs intermediários entre ambos, pelo menos Δa segmentos serão transmitidos em rajada pelo transmissor e a janela de congestionamento irá crescer de no máximo um segmento, independentemente do valor de Δa. A RA resolve este problema adicionando ACKs

intermediários entre *a1* e *a2*; dessa forma a RA reduz as rajadas no transmissor e permite que a janela cresça mais rapidamente.

3.4. Redes com longos atrasos de propagação

Além dos já citados erros de bit, uma outra característica de algumas redes, como as redes com enlace via satélite – mostrada na Figura 3.9, pode vir a degradar o desempenho do TCP: os longos atrasos de propagação. Isto ocorre devido ao fato da duração da fase de *Slow Start*, quando ainda é pequena a taxa de transmissão no TCP transmissor, poder se tornar muito longa.

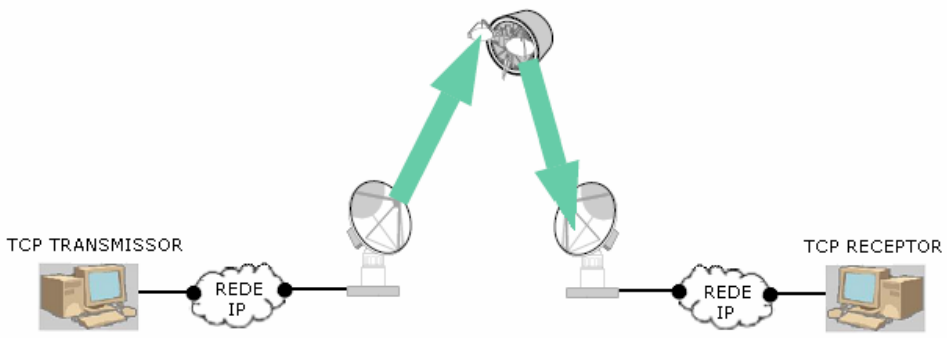


Figura 3.9 – Rede com enlace satélite.

De acordo com [23], o tempo necessário para o TCP transmissor atingir uma taxa de bits R durante a fase de *Slow Start* é dado por

$$t_{Slow\ Start} = RTT \,.\, (1 + log_2\,(R \cdot RTT/L)) \quad , \tag{3.15}$$

onde L é o tamanho médio, em bits, dos segmentos. Ressalta-se que a expressão só é válida caso o mecanismo de *Delayed Acknowledgements* [6] não esteja sendo utilizado, ou seja, um ACK é imediatamente gerado para cada segmento que chega ao receptor. Caso contrário, segundo [24], o tempo em *Slow Start* necessário para que uma taxa R seja alcançada é ainda superior ao dado por (3.15).

A Tabela 3.1 ilustra o tempo necessário para o TCP atingir taxas de bits B durante a fase de *Slow Start*, dado o tamanho médio de segmento L de 1 kbyte (valor típico), para redes de comunicação por satélites com diferentes tipos de órbitas, como *LEO* (*Low Earth Orbit*), *MEO* (*Medium Earth Orbit*) e *GEO* (*Geosynchronous Earth Orbit*).

Tabela 3.1 – Tempo de *Slow Start* para redes com satélites LEO, MEO e GEO. Fonte: [23].

Tipo de Satélite	*RTT*	$t_{Slow\ Start}$ (R = 1Mb/s)	$t_{Slow\ Start}$ (R = 10Mb/s)	$t_{Slow\ Start}$ (R = 155Mb/s)
LEO	50 ms	0,18s	0,35s	0,55s
GEO	250 ms	1,49s	2,32s	3,31s
MEO	550 ms	3,91s	5,73s	7,91s

Muitas das aplicações atuais, como as HTTP da Web, são baseadas em transferências de pequenos arquivos. Dessa forma, a probabilidade de toda a transferência ser executada na fase de *Slow Start* é grande, fazendo com que nesses casos o TCP não consiga usufruir de todos os recursos da rede.

3.4.1 – Aumento da janela inicial

Definido em [35], o aumento do tamanho da janela inicial (janela de congestionamento no início da transmissão TCP, de tamanho unitário) é a técnica mais simples para melhorar o desempenho do *Slow Start*, quando redes com longos atrasos de propagação são utilizadas.

Caso o TCP transmissor implemente o aumento do tamanho de janela inicial, esta deve obedecer um tamanho máximo definido pela equação

$$min\ (4 \cdot MSS,\ max\ (2 \cdot MSS,\ 4380\ \text{bytes}))\ . \qquad (3.16)$$

De forma geral, a Equação (3.16) faz com que a janela inicial salte do valor de $1 \cdot MSS$ (tipicamente 1 kbyte) para aproximadamente $4 \cdot MSS$ (ou aproximadamente 4 kbytes). Isto, de acordo com [35], leva a algumas vantagens, das quais destacam-se duas:

- A redução no tempo de transmissão para conexões que transmitem apenas uma pequena quantidade de dados, como as utilizadas em aplicações HTTP, vistas anteriormente. Qualquer transferência de uma quantidade de dados inferior ao novo tamanho da janela inicial é reduzida a um RTT;
- Para conexões com maiores transferências de dados, a utilização de uma janela inicial maior elimina até 3 RTTs no crescimento da janela de congestionamento na fase de *Slow Start*, o que é particularmente interessante em transferências em redes com longos atrasos de propagação, como as redes com enlace via satélite.

A recomendação [35] reitera ainda que as modificações do tamanho inicial da janela só devem ser aplicadas quando do início da transmissão de dados em uma conexão, não devendo ser utilizadas, por exemplo, quando da ocorrência de retransmissão por *timeout*, quando a janela deve realmente retornar ao valor unitário.

3.4.2 – O *Split*-TCP

A idéia básica dos protocolos *Split*-TCP, como o I-TCP definido em [25], é segmentar uma conexão TCP em múltiplas seções, de forma que existam várias conexões menores interligando transmissor e receptor. No caso de uma conexão com enlace satélite, por exemplo, o TCP pode ser dividido em três seções como mostrado na Figura 3.10, onde os terminais intermediários podem ser roteadores com funcionalidades do TCP.

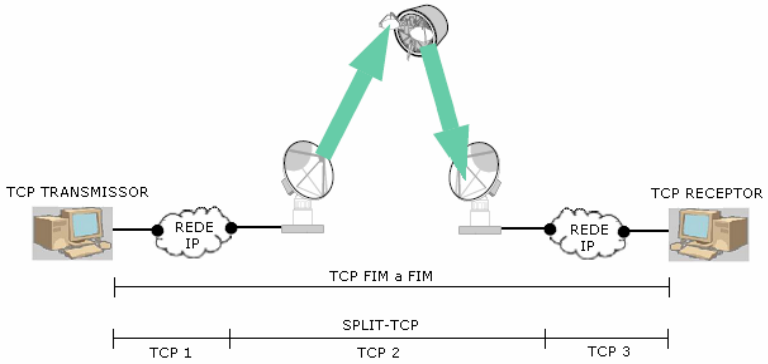


Figura 3.10 – *Split*-TCP em uma conexão com enlace satélite.

Com a utilização de protocolos *Split*-TCP, os efeitos dos longos atrasos de propagação não têm impacto no transmissor, já que sua conexão (TCP 1 na Figura 3.10) não engloba o enlace satélite. Dessa forma, a fase de *Slow Start* no transmissor ocorre rapidamente, levando o TCP transmissor a taxas maiores em menores períodos de tempo. Isto faz com que a primeira conexão, vista isoladamente, apresente um desempenho satisfatório. Contudo, apenas a divisão da conexão fim a fim em conexões menores não garante uma melhoria na transmissão de dados entre transmissor e receptor. Isto ocorre porque a conexão intermediária (TCP 2 na Figura 3.10) ainda sofre com as características adversas decorrentes do enlace satélite, apresentadas na Seção 3.4. Dessa forma, esta atua como "gargalo" na transmissão de dados fim a fim, degradando seu desempenho.

Para solucionar este problema, é necessário que na conexão intermediária seja utilizado um protocolo TCP modificado, que se ajuste às adversidades do enlace satélite. De acordo com [26], uma das possibilidades é utilizar o T/TCP [6], visto que entre suas características está evitar a utilização do *Slow Start*, prejudicial ao desempenho nas redes com enlace satélite, como visto anteriormente.

Em resumo, segundo [25], as vantagens de se utilizar protocolos *Split*-TCP como o I-TCP são:

- Separar as funcionalidades de controle de congestionamento e controle de fluxo da rede sem fio do restante da rede cabeada. Isto é desejável devido às enormes diferenças de características entre ambas, como taxas de transmissão, tempos de propagação e taxas de erros de bit;
- Um protocolo de transporte distinto pode lidar apropriadamente com eventos inerentes à rede sem fio, como os de perda de sinal e mobilidade;
- No caso de redes como a da Figura 3.4, a divisão da conexão permite que a estação base gerencie grande parte do *overhead* de comunicação do HM, fazendo com que serviços como Web, por exemplo, possam ser acessados sem que o HM possua toda a pilha de protocolos TCP/IP implementada.

3.4.3 – TCP *Spoofing*

Diferentemente dos protocolos *Split*-TCP, a técnica de TCP *Spoofing* [27] não divide a conexão fim a fim em conexões menores. A semelhança entre ambas se dá no fato de um roteador anterior ao enlace satélite, ao receber um segmento e encaminhá-lo ao próximo elemento da rede, gerar imediatamente um falso ACK (*Spoofing* ACK) de volta ao transmissor, como mostra a Figura 3.11. Desta forma, os atrasos de enlace ocorridos no enlace satélite são escondidos, fazendo com que a janela do transmissor cresça mais rapidamente durante a fase de *Slow Start*, assim como ocorre com a utilização de protocolos *Split*-TCP, vistos na seção anterior.

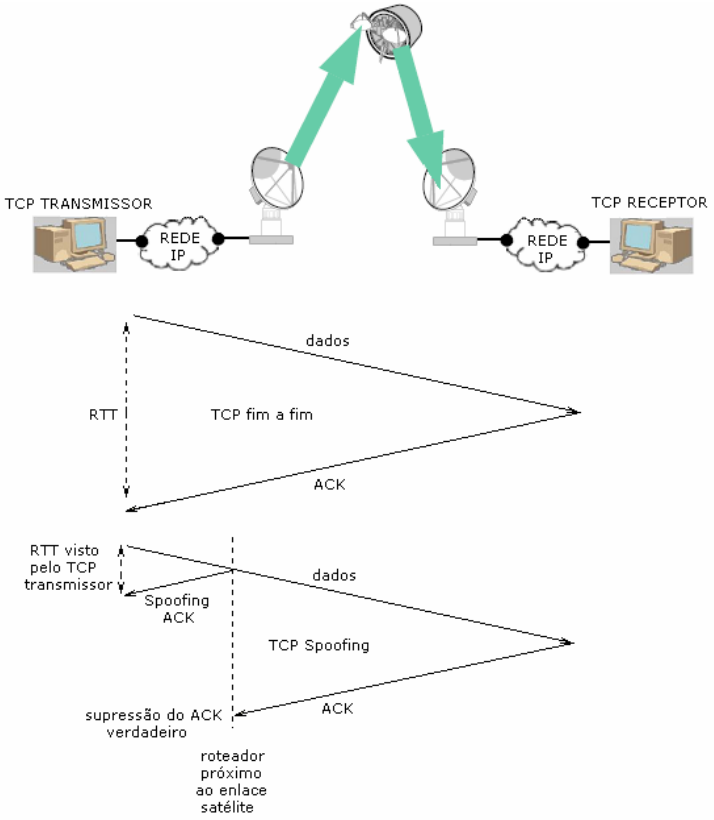


Figura 3.11 – TCP fim a fim *versus* TCP *Spoofing*.

Ao receber o verdadeiro ACK proveniente do receptor, o roteador responsável pelo *Spoofing* o retira da rede, para que retransmissões indesejadas através de ACKs duplicados não ocorram no transmissor. Ele ainda retira os segmentos reconhecidos pelo ACK do seu *buffer*, liberando espaço para a chegada de novos dados. Caso este roteador não receba ACKs referentes a um determinado segmento dentro de um tempo pré-determinado ou ainda receba alguns ACKs duplicados (como visto na Seção 2.6), ele retransmite o segmento perdido.

3.4.4 – O TCP-*Peach*

De acordo com [23], as técnicas de aumento da janela inicial e de TCP *Spoofing*, assim como os protocolos *Split*-TCP possuem algumas restrições, o que impedem em alguns casos suas utilizações. Para o autor, as principais restrições de cada método são:

- Aumento da janela inicial: como o ganho máximo de tempo durante a fase de *Slow Start* é de 3RTTs, em certos casos, como alguns mostrados na Tabela 3.1, o tempo de *Slow Start* pode ainda ser muito alto;
- TCP *Spoofing*: o roteador responsável pelo *Spoofing* fica sobrecarregado, pois este é responsável pela entrega correta dos segmentos, visto que ele gera um falso ACK

(*Spoofing* ACK) para o transmissor; caso o roteador responsável pelo *Spoofing* entre em falha, dados já confirmados ao transmissor na verdade podem nunca alcançar o receptor e finalmente, caso haja a possibilidade dos ACKs verdadeiros atravessarem um caminho distinto dos segmentos de dados (o que é muito comum na Internet) a técnica não pode ser utilizada;

- *Split*-TCP: sofre com os mesmos problemas do TCP *Spoofing*, com exceção do último.

Impulsionado pelas restrições encontradas nos métodos anteriores, o TCP-*Peach*, definido em [24], é um protocolo que objetiva melhorar o desempenho do TCP em enlaces satélite utilizando um controle de congestionamento diferenciado, no qual dois novos algoritmos são utilizados: o *Sudden Start*, em substituição ao *Slow Start*, e o *Rapid Recovery*, que substitui o algoritmo de *Fast Recovery*.

O funcionamento do TCP-*Peach* baseia-se na utilização de segmentos *dummy*, cópias do último segmento original, transmitidas com baixa prioridade. Segmentos *dummy*, portanto, não carregam novas informações para o receptor. A idéia é fazer com que o transmissor envie esses segmentos para testar a disponibilidade da rede. Caso a rede esteja efetivamente congestionada, os *dummy* são descartados primeiramente devido à sua baixa prioridade, fazendo com que sua transmissão não reduza o desempenho das transmissões de segmentos "verdadeiros". Caso a rede não esteja congestionada, ACKs para os segmentos *dummy* são gerados no receptor (ACKs *dummy*), que quando chegam de volta ao transmissor fazem com que este último incremente o valor da sua janela de congestionamento, aumentando o desempenho da transmissão.

Quando a transmissão de dados é iniciada, o TCP-*Peach* entra na fase de *Sudden Start*, na qual o tamanho da janela de congestionamento é unitário, como no *Slow Start* original. Após o envio do primeiro segmento (no instante t = 0, por exemplo), o *Sudden Start* envia (*rwnd* – 1) segmentos *dummy* dentro do intervalo de RTT do primeiro segmento, onde *rwnd* é o tamanho da janela de recepção inicial indicada pelo receptor durante a abertura da conexão. Caso não haja congestionamento na rede, os ACKs *dummy* chegam ao transmissor no intervalo $RTT \leq t < 2RTT$. Como cada ACK *dummy* faz com que a janela de congestionamento do transmissor seja incrementada de um, neste intervalo a janela é repentinamente "inflada" do tamanho unitário para o tamanho máximo permitido no receptor (*rwnd*), o que explica o nome do algoritmo *Sudden Start* (Partida Súbita). A partir desse

instante, o TCP-*Peach* executa o algoritmo de *Congestion Avoidance*, exatamente como mostrado na Seção 2.8.

Com o algoritmo de *Sudden Start* o TCP-*Peach* resolve o problema da interação adversa entre o TCP-Reno e as redes com longos atrasos de propagação, visto que, com 2 RTTs (pouco mais de 1s em redes com enlace *GEO*) o TCP-*Peach* alcança sua taxa máxima de transmissão.

Quando há a ocorrência de certo número de ACKs duplicados, o TCP-*Peach* inicia a retransmissão do segmento aparentemente perdido – assim como no TCP-Reno – caracterizando o algoritmo de *Fast Retransmission*. Em seguida, o algoritmo de *Rapid Recovery* é acionado, tendo sua duração limitada ao RTT do segmento retransmitido. Nesta fase a janela de congestionamento é reduzida à metade ($cwnd = cwnd_0/2$), assim como no TCP-Reno. Em seguida, $cwnd_0$ segmentos *dummy* são enviados dentro do intervalo de RTT do segmento retransmitido. Quando o ACK referente ao segmento retransmitido chega ao transmissor, este entra na fase de Congestion Avoidance, como no TCP-Reno. A partir deste momento, os ACKs referentes aos segmentos *dummy* começam a chegar ao transmissor. Os primeiros $cwnd_0/2$ ainda não têm como objetivo "inflar" a janela de congestionamento. Na verdade estes são utilizados para verificar a verdadeira condição da rede. Caso o ACK do segmento retransmitido assim como os $cwnd_0/2$ ACKs *dummy* não cheguem dentro do período de RTO do segmento retransmitido, o TCP-*Peach* entende que a rede apresenta um severo congestionamento e então retorna a fase de *Sudden Start*. Caso estes ACKs cheguem corretamente ao transmissor, cada um dos $cwnd_0/2$ ACKs *dummy* restantes faz com que a janela seja incrementada de um segmento. Dessa forma, a janela cresce rapidamente de $cwnd_0/2$ para $cwnd_0$, ou seja, a janela retorna ao tamanho anterior à perda do segmento em aproximadamente 2RTT.

Com a utilização do algoritmo de *Rapid Recovery*, o TCP-*Peach* apresenta um melhor desempenho na ocorrência de descarte de segmentos por erros de bit se comparado ao TCP padrão. Isto porque em apenas 2 RTTs a janela de congestionamento volta a ter o mesmo tamanho quando da ocorrência do erro de bit.

4. *Throughput* do TCP com mecanismos de controle de erro *FEC*, *GBN* ou *SW* no enlace sem fio.

Neste capítulo são mostrados os modelos analíticos criados para computar o desempenho do TCP quando mecanismos de controle de erro (MCEs) *FEC*, *GBN* ou *SW* são utilizados no enlace sem fio. Na Seção 4.1 é mostrado o modelo analítico base de todos os outros, o modelo proposto em [28] para redes TCP sem erros. A partir deste modelo, uma primeira gama de modificações foi feita, e o resultado, visto na Seção 4.2, é o modelo proposto em [29], capaz de computar o desempenho do TCP quando o mecanismo *FEC* é utilizado no enlace sem fio, mas ainda deficiente para o cálculo do desempenho do TCP quando os mecanismos *SW* ou *GBN* são utilizados. Novas alterações são propostas, e o resultado é o modelo apresentado em [30] para redes com enlace celular, visto na Seção 4.3, que computa o desempenho do TCP quando estes dois últimos mecanismos de controle de erro são utilizados no enlace sem fio. Para este último modelo, os resultados mostrados em [31] abordam a utilização de enlaces satélite.

É importante dizer que alguns trabalhos anteriores contemplam este assunto, como em [37], onde o desempenho do TCP com mecanismos *ARQ*/*FEC* é analisado, via simulação, para redes com canais sem fio com características de *fading*; em [38], onde o desempenho do TCP com vários mecanismos de controle de erro como *ARQ*, o protocolo Snoop e RLP (*Radio Link Protocols*) é comparado através de simulação; em [39], onde o desempenho do TCP com um protocolo híbrido *ARQ*/*FEC* é analisado, através de simulação, para canais com erros que seguem a distribuição de Bernoulli; ou em [15], onde o desempenho dos mecanismos *ARQ*, *FEC* e híbridos *ARQ*-*FEC* são analisados analiticamente para redes ATM (*Asynchronous Transfer Mode*). Vale ressaltar que, apesar da grande quantidade de material relacionado encontrado, nenhum destes contempla o cálculo analítico do desempenho do TCP quando mecanismos de controle de erro *SW*-*ARQ*, *GBN*-*ARQ* ou *FEC* são utilizados no enlace sem fio, como é proposto nesta dissertação.

4.1 O modelo original

O modelo analítico base deste estudo é o modelo proposto em [28]. Esse modelo consiste de: um transmissor TCP com uma fonte infinita de dados a serem transmitidos; um enlace com a menor taxa de transmissão da rede (μ pacotes/s), chamado enlace gargalo, um *buffer* tipo

FIFO com *B* posições de espera e um receptor TCP que gera reconhecimentos (ACKs) para pacotes recebidos com sucesso. Neste modelo, todos os atrasos do sistema, excetuando-se o tempo de transmissão no enlace gargalo e o tempo de espera no *buffer* são agrupados em um único "atraso global" τ que inclui os tempos de transmissão e propagação dos segmentos TCP até chegarem ao *buffer*, o tempo de propagação dos segmentos TCP no enlace gargalo e os tempos de transmissão e propagação dos ACKs. Neste sistema todos os segmentos TCP são transmitidos com tamanho fixo, igual ao MSS (*Maximum Segment Size*), embora, na prática, o tamanho dos segmentos TCP possa variar. A Figura 4.1 ilustra o modelo utilizado [28].

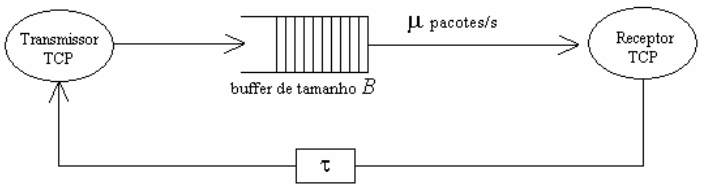


Figura 4.1 – Modelo original. Fonte: [28].

De acordo com [28], a janela de congestionamento do TCP, em estado de regime permanente, apresenta comportamento cíclico. Para ilustrar esse comportamento é utilizada uma rede para a qual o modelo apresentado na Figura 4.1 é adequado. Esta rede possui somente dois enlaces, sendo um deles com fio e o outro sem fio. O enlace sem fio é considerado o gargalo do sistema. A interface entre os dois enlaces é feita por uma estação base (EB) com *buffer* para três segmentos. A Figura 4.2 ilustra a rede utilizada, na qual se admite que o transmissor TCP esteja inicialmente na fase de *Congestion Avoidance*, com janela de congestionamento igual a dois segmentos.

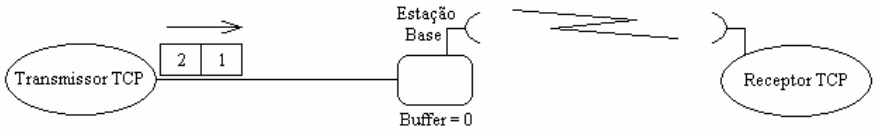


Figura 4.2 – Estado da rede no estado inicial.

Num primeiro instante, os dois segmentos TCP (1 e 2) são enviados no enlace com fio. Quando o segmento 1 acaba de ser propagado no primeiro enlace, esse é transmitido pela EB no enlace sem fio. Como nesse enlace a transmissão dos segmentos é mais demorada visto que ele é o gargalo do sistema, o segmento 2 chega à EB antes do término da transmissão do

segmento 1 no enlace sem fio ser finalizada e por isso este segmento aguarda no *buffer* o momento de ser transmitido, como ilustra a Figura 4.3.

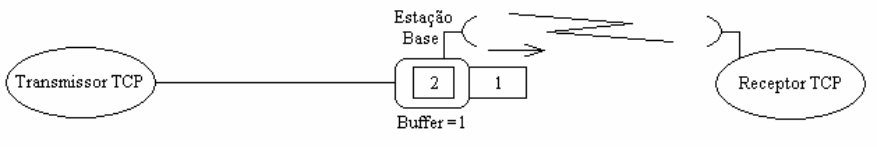


Figura 4.3 – Estado da rede após a transmissão do segmento 1 no enlace sem fio.

Ao final da transmissão do segmento 1, a transmissão do segmento 2 se inicia imediatamente, como ilustra a Figura 4.4.

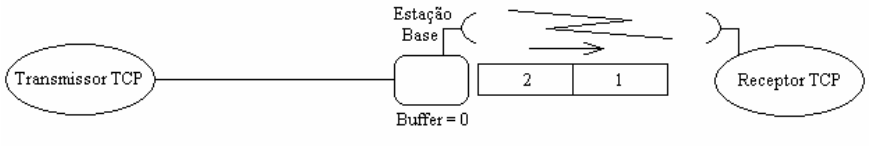


Figura 4.4 – Estado da rede após a transmissão do segmento 2 no enlace sem fio.

Desconsiderando o tempo de processamento nos nós e também qualquer tipo de retardo na transmissão dos ACKs (como o mecanismo de *delayed ACKs* [6], por exemplo), quando o primeiro segmento chega ao receptor TCP, a transmissão de seu ACK é *imediatamente* iniciada. Como neste momento o segundo segmento ainda não foi recebido pelo receptor TCP, cria-se uma lacuna até que esse seja recebido e seu correspondente ACK seja transmitido, como ilustrado na Figura 4.5. Essa lacuna temporal t_{esp} tem exatamente um tempo de transmissão do enlace gargalo, ou seja, $t_{esp} = 1/\mu$.

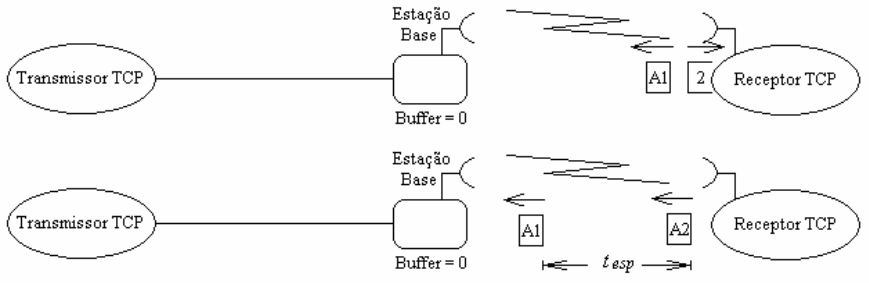


Figura 4.5 – Estado da rede na transmissão dos ACKs dos segmentos 1 e 2 no enlace sem fio.

Quando o ACK do segmento 1 chega ao transmissor TCP, a janela de congestionamento desliza e então um novo segmento pode ser inserido na rede (o segmento 3). A chegada do ACK do segmento 2 ao transmissor TCP faz com que a janela de congestionamento deslize e cresça de um segmento, passando a ter tamanho 3, liberando os segmentos 4 e 5 para

transmissão. A Figura 4.6 ilustra a chegada dos ACKs referentes aos segmentos 1 e 2 ao transmissor TCP, e a Figura 4.7 ilustra a inserção dos novos segmentos.

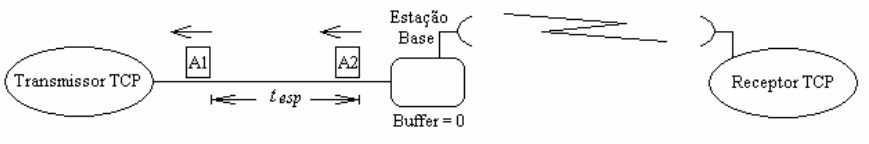


Figura 4.6 – Estado da rede que ilustrando o espaçamento temporal t_{esp} entre os ACKs dos segmentos 1 e 2.

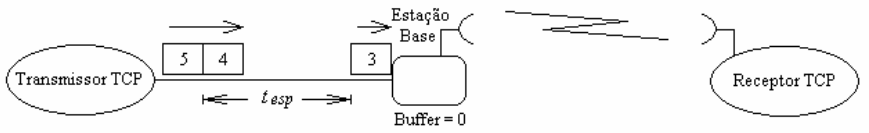


Figura 4.7 – Estado da rede após a transmissão do segmento 5 no enlace com fio.

Quando o segmento 3 chega à EB, sua transmissão é iniciada no enlace sem fio. No exato instante em que a transmissão do segmento 3 é finalizada, o segmento 4 chega à EB e sua transmissão no enlace sem fio é iniciada, agrupando de volta os segmentos, como mostra a Figura 4.8.

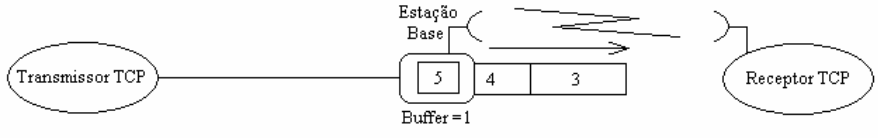


Figura 4.8 – Estado da rede na transmissão do segmento 4 no enlace sem fio.

Quando a transmissão do segmento 4 cessa, o segmento 5, que já aguarda no *buffer*, é transmitido pela EB, como mostra a Figura 4.9.

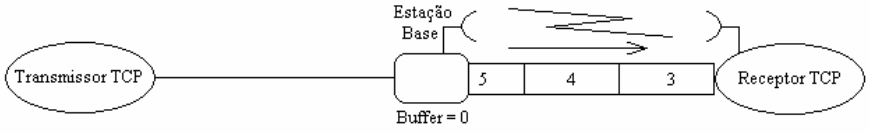


Figura 4.9 – Estado da rede na transmissão do segmento 5 no enlace sem fio.

Todo processo a partir de então se repete: um espaçamento temporal t_{esp} é inserido entre os segmentos 3 e 4, e também entre os segmentos 4 e 5. A chegada dos ACKs dos segmento 3 e 4 no transmissor implica na transmissão dos segmentos 6 e 7 (com o espaçamento t_{esp} entre ambos), assim como a chegada do ACK do segmento 5 implica no aumento do tamanho da

janela para 4 segmentos e, portanto, na transmissão dos segmentos 8 (espaçado de t_{esp} com relação ao segmento 7) e 9 (sem espaçamento para o segmento 8).

Num determinado instante, depois da inserção de vários segmentos na rede, há segmentos e ACKs ocupando toda a rede, como ilustra a Figura 4.10.

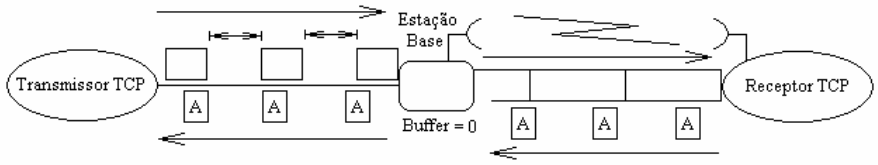


Figura 4.10 – Rede totalmente preenchida com segmentos e ACKs.

Nesse instante, o crescimento da janela faz com que o *buffer* seja ocupado por um segmento permanentemente. A transmissão continua e a janela cresce novamente, fazendo com que em seguida o *buffer* passe a ter dois segmentos permanentemente. O processo continua até que o *buffer* tenha 3 segmentos permanentemente. Nesse instante, o crescimento da janela e a conseqüente inserção de um novo segmento na rede fazem com que sejam necessárias quatro posições no *buffer* para armazenar todos os segmentos. Como o *buffer* tem somente três posições de espera, esse novo segmento é descartado, como ilustra a Figura 4.11.

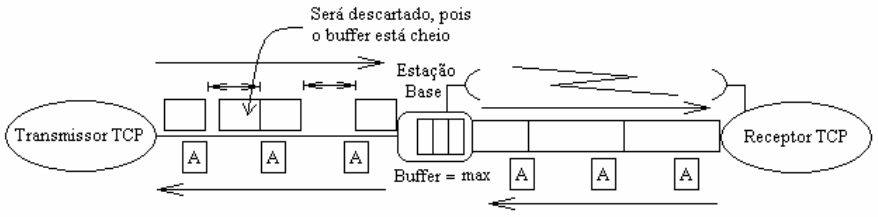
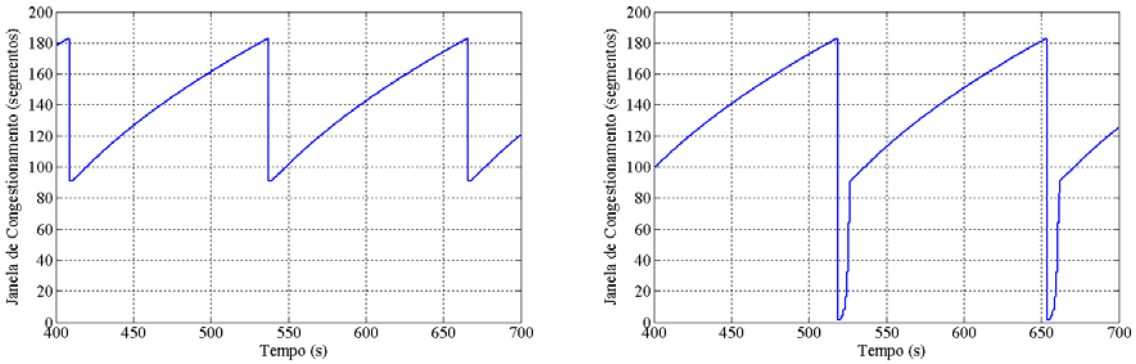


Figura 4.11 – Estado da rede com *bit pipe* com um segmento a mais que o máximo suportado.

O processo de retransmissão do segmento descartado depende da versão do TCP em uso. No caso do TCP-Reno, o transmissor percebe a perda do segmento através da chegada de ACKs duplicados referentes aos segmentos subseqüentes ao segmento descartado. Desse modo, após o descarte, uma quantidade aproximadamente igual à janela de congestionamento ainda será transmitida pelo transmissor TCP até que a retransmissão do segmento descartado seja efetuada e a janela de congestionamento seja reduzida à metade, caracterizando os mecanismos de *Fast Retransmission/Fast Recovery* (*FR/FR*). O TCP-Reno volta então para a fase de *Congestion Avoidance* e como a janela é diminuída, reduzindo a taxa de transmissão

dos segmentos, o *buffer* se esvazia. Todo o processo descrito anteriormente se repete, com a janela de congestionamento crescendo até que o *buffer* chegue novamente ao limite e outro segmento seja descartado. No caso do TCP-Tahoe, a retransmissão do segmento descartado é baseada no *timeout* do mesmo. O transmissor TCP-Tahoe vai então para a fase de *Slow Start*, com sua janela reduzida à unidade e crescendo rapidamente até atingir metade do seu tamanho no instante do *timeout*. A partir desse instante, o TCP-Tahoe entra na fase de *Congestion Avoidance* e todo o processo novamente se repete até um novo segmento ser descartado. A Figura 4.12 mostra a característica cíclica da janela de congestionamento nos casos do TCP-Reno e do TCP-Tahoe para redes cujo modelamento pode ser representado pela Figura 4.1.



(a) (b)

Figura 4.12 – Comportamento cíclico da janela de congestionamento para transmissores (a) TCP-Reno e (b) TCP-Tahoe.

Uma conseqüência direta da característica cíclica na evolução da janela de congestionamento é que, dado o número de pacotes N_c enviados nos ciclos cuja duração é T_c, o *throughput* médio da transmissão do TCP, dado em segmentos por unidade de tempo, pode ser aproximado pela razão $\bar{\lambda} = N_c/T_c$. O modelamento apresentado em [28] visa justamente calcular os valores de N_c e T_c para chegar ao *throughput* médio da transmissão. Para isso, são definidos: a variável T, atraso global somado ao tempo de transmissão no enlace gargalo, dada por

$$T = \tau + 1/\mu, \tag{4.1}$$

e a máxima janela de congestionamento que pode ser acomodada no *bit pipe* da rede,

$$W_{pipe} = \mu \cdot T + B, \tag{4.2}$$

onde B computa os segmentos armazenados no *buffer* e o produto $\mu \cdot T$ representa os pacotes em trânsito na rede.

No caso do TCP-Tahoe pode-se considerar que cada ciclo começa com a fase de *Slow Start*. De acordo com [28], essa fase dura aproximadamente

$$t_{SS} = T \cdot log_2(W_{fSS}) \tag{4.3}$$

onde W_{fSS} é o tamanho da janela de congestionamento ao final da fase de *Slow Start*. Como o fim da fase de *Slow Start* é também o início da fase de *Congestion Avoidance*, $W_{fSS} = W_{pipe}/2$. Durante essa fase, são enviados aproximadamente

$$n_{SS} = W_{fSS} \tag{4.4}$$

pacotes.

A caracterização da fase de *Congestion Avoidance* depende do "tamanho de *buffer* normalizado", definido em [28] por

$$\beta = B / (\mu \cdot T). \tag{4.5}$$

Caso $\beta < 1$, no inicio da fase de *Congestion Avoidance*, o *buffer* está totalmente vazio, pois $W_{pipe}/2 < \mu.T$, e esta fase é dividida em duas sub-fases: a sub-fase (*A*), na qual a janela de congestionamento cresce de $W_{pipe}/2$ até o tamanho igual ao produto $\mu \cdot T$, tendo este crescimento a característica linear. Esta sub-fase dura aproximadamente

$$t_A = T \cdot (W_{fA} - W_{iA}) \tag{4.6}$$

onde $W_{iA} = W_{pipe}/2$ é o tamanho inicial da janela na sub-fase *A* e $W_{fA} = \mu \cdot T$ define o tamanho final da janela nesta sub-fase. Neste período são transmitidos aproximadamente

$$n_A = [W_{iA} \cdot t_A + t_A^{\,2}/(2 \cdot T)]/T \tag{4.7}$$

pacotes. A sub-fase seguinte é caracterizada pelo crescimento da janela a partir do produto $\mu \cdot T$ até se atingir o valor de W_{pipe}. Nesta sub-fase (*B*) o crescimento da janela é sub-linear, e sua duração é de aproximadamente

$$t_B = [W_{fB}^{\,2} - W_{iB}^{\,2}]/(2 \cdot \mu), \tag{4.8}$$

sendo $W_{iB} = \mu \cdot T$ e $W_{fB} = W_{pipe}$, respectivamente, o tamanho da janela no início e no fim desta sub-fase. Durante a sub-fase *B*, são transmitidos aproximadamente

$$n_B = \mu \cdot t_B \tag{4.9}$$

pacotes. Ao final da sub-fase *B*, a janela atinge seu valor máximo, um segmento é descartado e o ciclo recomeça.

Caso $\beta > 1$, o TCP-Tahoe passa diretamente da fase de *Slow Start* para a fase de *Congestion Avoidance* na sub-fase de crescimento sub-linear *B*. Isto ocorre, pois já no início da *Congestion Avoidance* o *buffer* está parcialmente ocupado, dado que nesse caso $W_{pipe}/2 > \mu \cdot T$. A janela crescerá de $W_{pipe}/2$ até W_{pipe}, e o tempo de crescimento, assim como o número de pacotes transmitidos serão dados, respectivamente, pelas equações (4.8) e (4.9), fazendo $W_{iB} = W_{pipe}/2$ e $W_{fB} = W_{pipe}$.

Assim como no TCP-Tahoe, no TCP-Reno a existência das sub-fases *A* e *B* ou a existência somente da sub-fase *B* durante a *Congestion Avoidance* também depende do parâmetro β. No entanto, ao final da sub-fase *B*, o TCP-Reno passa diretamente para o início da sub-fase *A* (caso $\beta < 1$) ou retorna ao início da sub-fase *B* (caso $\beta > 1$), não entrando na fase de *Slow Start*, dada a característica da recuperação do segmento descartado através dos mecanismos de *FR/FR*.

Lembrando que o *throughput* médio em regime pode ser aproximado pelo *throughput* de um ciclo, os *throughputs* do TCP-Tahoe e do TCP-Reno podem ser calculados, respectivamente, por

$$\bar{\lambda}_T = \begin{cases} \dfrac{n_{SS1}+n_{A1}+n_{B1}}{t_{SS1}+t_{A1}+t_{B1}}, & \text{se } \beta < 1 \\ \dfrac{n_{SS1}+n_{B2}}{t_{SS1}+t_{B2}}, & \text{se } \beta > 1 \end{cases} \tag{4.10}$$

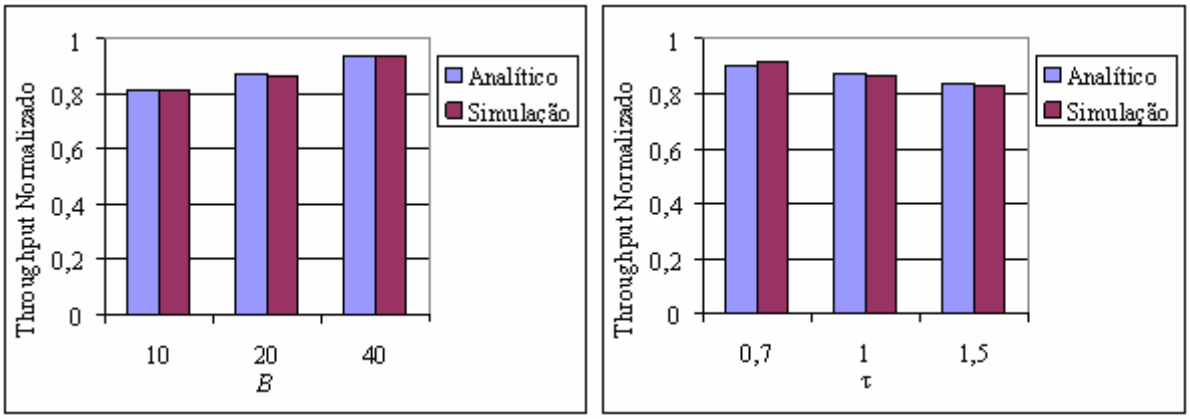
$$\bar{\lambda}_R = \begin{cases} \dfrac{n_{A1}+n_{B1}}{t_{A1}+t_{B1}}, & \text{se } \beta < 1 \\ \dfrac{n_{B2}}{t_{B2}}, & \text{se } \beta > 1 \end{cases} \tag{4.11}$$

onde t_{SS1} e n_{SS1} são definidos pelas equações (4.3) e (4.4) com $W_{fSS} = W_{pipe}/2$; t_{A1} e n_{A1} são definidos por pelas equações (4.6) e (4.7) com $W_{iA} = W_{pipe}/2$ e $W_{fA} = \mu \cdot T$; t_{B1} e n_{B1} são definidos por pelas equações (4.8) e (4.9) com $W_{iB} = \mu \cdot T$ e $W_{fB} = W_{pipe}$ e, por fim, t_{B2} e n_{B2} são definidos por pelas equações (4.8) e (4.9) com $W_{iB} = W_{pipe}/2$ e $W_{fB} = W_{pipe}$

4.1.1 Validação do modelo e resultados obtidos

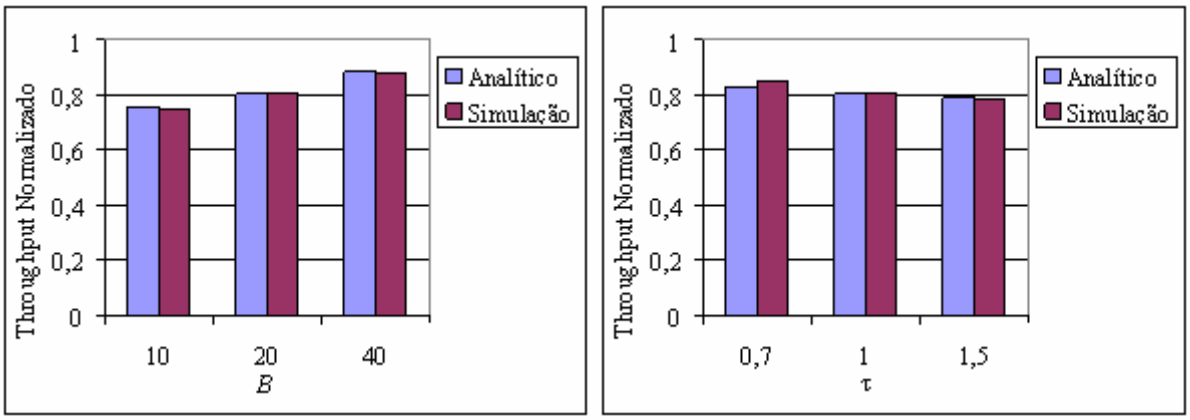
A validação do modelo proposto em [28] é feita através de uma rede cujo modelamento da Figura 4.1 é apropriado. Nesta rede, o valor de μ foi fixado em 100 pacotes/s e os parâmetros

B e τ foram variados. O resultado é dado em termos do *throughput* normalizado, ou seja, é dado por $TN = \bar{\lambda}/\mu$, onde $\bar{\lambda}$ representa o *throughput* do TCP também em pacotes/s, dado por (4.10) ou (4.11), dependendo se a versão do TCP em uso é o Tahoe ou o Reno, respectivamente. Os resultados obtidos através de simulação e do modelamento analítico proposto por [28] para esta rede estão apresentados nas Figuras 4.13 e 4.14, respectivamente quando os protocolos TCP-Reno e TCP-Tahoe são utilizados.



(a) (b)

Figura 4.13 – *Throughput* Normalizado do TCP-Reno quando (a) τ = 1s e (b) B = 20 pacotes.



(a) (b)

Figura 4.14 – *Throughput* normalizado do TCP-Tahoe quando (a) τ = 1s e (b) B = 20 pacotes.

Nota-se, pelas Figuras 4.13 (a) e 4.14 (a) que quanto maior o valor do *buffer* melhor o desempenho do TCP. Em contrapartida, um aumento no atraso global τ implica na degradação de seu desempenho, como pode ser visto nas Figuras 4.13 (b) e 4.14 (b). Outra conclusão que pode ser tirada observando as Figuras 4.13 e 4.14 é que o desempenho do TCP-Reno, para este tipo de rede, é sempre superior ao do TCP-Tahoe.

4.2 A primeira modificação

Nesta seção, o modelo proposto anteriormente é adaptado de modo que o gargalo do novo sistema represente um enlace sem fio com erros de bit, onde é considerado um dos mecanismos de controle de erro (MCE), *ARQ*, *FEC* ou híbrido *ARQ-FEC* com múltiplos estados *FEC*, analisados na Seção 3.1.1. Neste novo modelo, algumas alterações são efetuadas com relação ao modelo original para que possam ser captados os efeitos da inserção destes protocolos. Nos casos do *ARQ* e do protocolo híbrido *ARQ-FEC* com múltiplos estados *FEC*, um novo parâmetro τ_b é inserido no enlace sem fio e representa os atrasos inerentes ao protocolo baseado em retransmissão, tais como o tempo de propagação do quadro e os tempos de transmissão e propagação no envio do ACK da camada de enlace. Como no protocolo *FEC* não há retransmissão de segmentos, não há a inserção desse atraso τ_b no enlace sem fio, ou seja, $\tau_b = 0$. A taxa de transmissão no enlace sem fio também é alterada para levar em conta o *overhead* causado pelo protocolo de camada de enlace. Seu novo valor passa a ser dado por

$$\mu' = \eta \cdot \mu, \tag{4.12}$$

onde η representa a eficiência do protocolo em uso (mostrada na Seção 3.1.1). Como nos casos dos protocolos *ARQ* e híbrido *ARQ-FEC* com múltiplos estados *FEC* o cálculo da eficiência envolve, entre outros, os atrasos de propagação no enlace sem fio, é necessário modificar o valor de τ para um novo valor τ', de forma a evitar uma duplicidade na computação desse atraso (visto que este já é computado por τ_b). No caso dos protocolos *SW-ARQ*, *GBN-ARQ* e híbrido *ARQ-FEC*, os valores de τ' serão dados respectivamente por

$$\tau' = \tau - t_{pw}, \tag{4.13}$$

$$\tau' = \tau - t_{pw} \cdot (1 - P_c), \tag{4.14}$$

$$\tau' = \tau - t_{pw} \cdot (1 - P_c) \cdot P_{arq}, \tag{4.15}$$

onde o parâmetro t_{pw} representa o atraso de propagação no enlace sem fio, P_c e P_{arq} são definidos em (3.7) e (3.12) respectivamente. Os valores de T, β e W_{pipe} também são atualizados para T', β' e W_{pipe}' definidos pelas equações

$$T' = \tau' + 1 / \mu', \tag{4.16}$$

$$\beta' = B / (\mu' \cdot T'), \tag{4.17}$$

$$W_{pipe}' = \mu' \cdot T' + B. \tag{4.18}$$

É importante notar que neste caso o valor de β', T' e W_{pipe}' variam não só com as características de atrasos da rede, mas também com qualquer parâmetro que influencie na

eficiência do protocolo em uso, por exemplo, a BER, no caso dos protocolos *ARQ*. A Figura 4.15 ilustra as modificações feitas no modelo original.

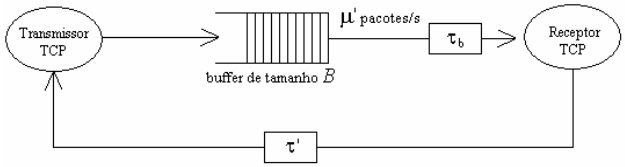


Figura 4.15 – Modelo modificado

Apesar do modelo desenvolvido em [28] analisar o desempenho do TCP-Tahoe assim como do TCP-Reno, a partir deste ponto a análise ficará restrita ao último, dado que esta é a versão do TCP mais utilizada atualmente [5]. Portanto, toda vez que o TCP for mencionado, entenda-se TCP-Reno.

4.2.1 Validação do modelo e resultados obtidos

Independentemente do sistema modelado, todos os códigos de bloco foram definidos de forma que o enlace sem fio opere, com relação às características de erros de bit, como um enlace óptico, ou seja, após a ação do controle de erro de um dos protocolos utilizados, a probabilidade de um pacote conter erros deve ser igual à probabilidade obtida em um enlace óptico quando nenhum protocolo de controle de erro é utilizado. Este critério está de acordo com a conclusão apresentada em [32].

Desta forma, para definir os códigos utilizados, foi considerado que a probabilidade do protocolo de camada de enlace do HM entregar um pacote sem erros para as camadas superiores deve ser superior a 0,9999999 e, portanto, os valores de P_{cca} (quando o protocolo *ARQ* é utilizado) dado em (3.10), P_{ccf} (quando o esquema *FEC* é utilizado) dado em (3.1) e P_{cch} (quando o esquema híbrido *ARQ-FEC* é utilizado) dado em (3.14), devem ser superiores a este valor. A partir desse critério, os limitantes mostrados na Seção 3.1.1.2 são utilizados para computar o número de bits de paridade necessário no caso da utilização do *ARQ* ou do *FEC*. No caso do esquema híbrido *ARQ-FEC* com múltiplos estados *FEC*, a definição do número de bits de paridade para o *ARQ* e para o *FEC* é um pouco mais complexa, visto que, de acordo com [15], existem inúmeras combinações de paridade para o *ARQ* e o *FEC* que satisfazem um determinado valor de P_{cch}.
O procedimento para obtenção da combinação mais adequada utilizado é proposto em [15].

Para validar o modelo analítico da seção anterior, duas redes serão propostas, ambas como aquela mostrada na Figura 4.2. Na primeira, a Rede C, é considerado que o atraso de propagação no enlace sem fio é de t_{pw} = 50μs, valor comumente encontrado em um enlace celular. Na segunda, a Rede S, este valor é aumentado para t_{pw} = 250ms, valor característico de um enlace satélite tipo GEO. Ambas as redes tem modelo inicial como mostrado na Figura 4.1, cujos parâmetros são, a não ser que explicitado o contrário, para a Rede C, μ = 100 pacotes/s, τ = 1s e B = 20 pacotes e para a Rede S, μ = 100 pacotes/s, τ = 1.5s e B = 100 pacotes. A este modelo serão inseridos os protocolos de camada de enlace no enlace sem fio, resultando no modelo mostrado na Figura 4.15. Na Rede C, é fixado o valor de τ_b = 1ms para os protocolos *ARQ* e híbrido *ARQ-FEC* com múltiplos estados *FEC*, resultado de um tempo de transmissão do ACK da camada de enlace t_{aw} = 0,9ms ($\tau_b = 2 \cdot t_{pw} + t_{aw}$). No caso da Rede S, este valor passará para τ_b = 500,9ms.

Tanto para a Rede C, quanto para a Rede S, são considerados três possíveis cenários com relação ao impacto da taxa de erro de bit na definição dos números de bits de informação e de paridade do bloco: no primeiro, chamado não adaptativo, o número de bits de informação é fixo e o número de bits de redundância necessário para prover a capacidade de detecção de erros (no caso do *ARQ*) ou a capacidade de correção de erros (no caso do *FEC*) é calculado baseado na maior BER do canal, escolhida como 10^{-2}; no segundo cenário, chamado adaptativo simples, o número de bits de informação também é fixo, mas o número de bits de redundância varia com a taxa de erros de bit do canal, melhorando o desempenho com relação ao primeiro cenário; no terceiro e último cenário, chamado adaptativo ideal, o número de bits de informação, assim como o número de bits de redundância, são otimizados, maximizando a eficiência do sistema proposto. Em ambos os cenários adaptativos, foi desconsiderado qualquer *overhead* necessário à estimativa da taxa de erro utilizada e à mudança nos tamanhos dos blocos.

Os resultados de cada sistema são mostrados em termos do *throughput* normalizado de dados do TCP, dado através da expressão

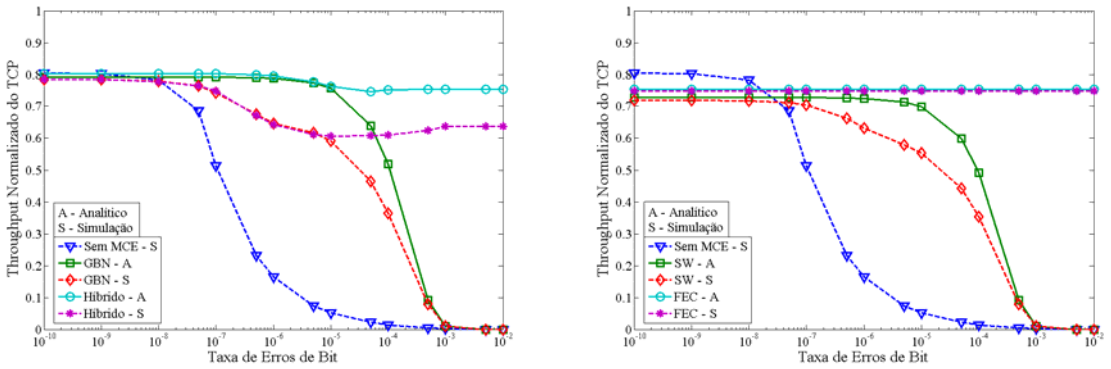
$$TN = \frac{\bar{\lambda}_R \cdot k'}{R} \ , \tag{4.19}$$

onde $\bar{\lambda}_R$ é dado por (4.11), k' representa o número de bits de dados do TCP presente em cada bloco transmitido e R é a taxa de transmissão do enlace sem fio, fixada em 100 pacotes por

segundo quando nenhum mecanismo de controle de erro é utilizado, ou seja, $R = \mu \cdot k = 100 \cdot k$, onde k representa o número de bits de informação de cada bloco (tamanho do pacote IP), visto pelo mecanismo de controle de erro utilizado. Os valores de k e k' estão associados pela expressão $k = k' + cab$, onde cab = 320 bits representa os cabeçalhos dos protocolos TCP e IP.

4.2.1.1. Resultados para a Rede C

Inicialmente, para os cenários não adaptativo e adaptativo simples, o valor de k' é fixado em k' = 4000 bits (500 bytes, valor comumente encontrado) e, portanto, k = 4320 bits. No cenário não adaptativo, o número de bits de redundância necessário para o *ARQ* e para o *FEC*, calculados através dos limitantes mostrados na Seção 3.1.1.2, são respectivamente $n - k$ = 88 e $n - k$ = 337 bits. Já no caso do esquema híbrido *ARQ-FEC* com múltilpos estados *FEC*, os valores de y = 11 estados *FEC*, do número de bits de paridade no estado *ARQ* igual a $n - k$ = 18 e do número de bits de paridade no estado *FEC* igual a $n - k$ = 337 foram calculados de acordo com [15]. Com esses valores de bits de paridade, o tamanho do bloco será para o *ARQ*, n = 4408 bits, para o *FEC*, n = 4657 bits e para o híbrido *ARQ-FEC* com múltiplos estados *FEC* será, no estado *ARQ*, n = 4338 e no estado *FEC*, n = 4657. Os resultados analíticos para esse cenário são mostrados na Figura 4.16, juntamente com os resultados obtidos nas simulações feitas no ns-2.



(a) (b)

Figura 4.16 – *Throughput* normalizado do TCP para a Rede C no cenário não adaptativo com k = 4320, τ = 1s e B = 20 para os sistemas (a) *GBN* e Híbrido, e (b) *SW* e *FEC*.

Os resultados de simulação mostram que para os sistemas que utilizam *GBN*, *SW* ou híbrido *ARQ-FEC* com múltilpos estados *FEC* o modelamento utilizado não é adequado. Em contrapartida, o resultado de simulação também mostra que o modelo analítico para sistemas com *FEC* é suficientemente preciso. Outra conclusão que se pode tirar a partir do resultado de

simulação mostrado na Figura 4.16 (a) é o baixo desempenho dos sistemas que utilizam mecanismo híbrido *ARQ-FEC* com múltiplos estados *FEC*, que apesar de pouco superior ao do *FEC* para BERs inferiores à 10^{-7} (especificamente neste exemplo), é muito inferior ao do próprio *FEC* quando este limite é ultrapassado, fazendo com que este último protocolo seja uma escolha muita mais interessante entre os dois mecanismos. Devido a esse baixo desempenho, o mecanismo híbrido *ARQ-FEC* com múltiplos estados *FEC* não será mais abordado.

Como apenas os resultados do *FEC* se mostram coerentes no primeiro cenário analisado, apenas seus resultados são mostrados para os cenários subseqüentes. Análises mais detalhadas dos comportamentos dos sistemas com *SW* e com *GBN* são efetuadas nas seções seguintes.

Ainda no cenário não adaptativo, a Figura 4.17 mostra o desempenho do sistema com *FEC* quando os parâmetros τ, B e k são alterados. Primeiramente, pode-se concluir que a diminuição do atraso global da rede τ ou o aumento no tamanho do *buffer* B, levam a uma melhora no desempenho do TCP. Qualitativamente, esses dois resultados são sempre válidos, independentemente do cenário, ou do mecanismo de controle de erro em uso. A outra conclusão é que quando o *FEC* é utilizado, a diminuição do tamanho do pacote k leva a diminuição do desempenho do TCP. Vale dizer que no caso em que k = 820 bits, mostrado na Figura 4.17, o número de bits de paridade necessários para o sistema *FEC* foi de $n - k$ = 111 bits.

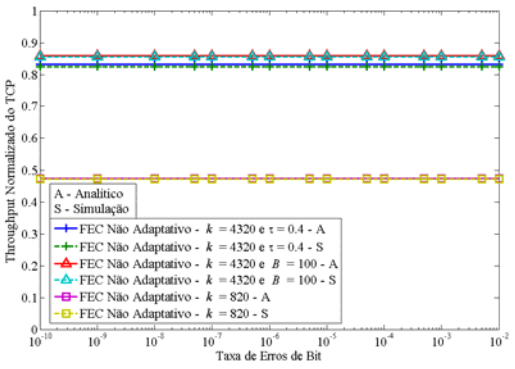


Figura 4.17 – *Throughput* normalizado do TCP para a Rede C com *FEC* no cenário não adaptativo quando os parâmetros k, B e τ são alterados.

No cenário adaptativo simples, o número de bits de informação do TCP continua fixo e igual a k = 4320 bits, no entanto o número de bits de redundância para o *FEC* varia de acordo com

a BER. No cenário adaptativo ideal, o número de bits de informação do segmento TCP é otimizado. No entanto, para o sistema *FEC*, o tamanho que otimiza o desempenho é sempre o máximo possível (dado que o número de bits de paridade cresce sublinearmente com o número de bits de informação, fazendo com que para blocos maiores tenham menor *overhead*), ou seja, k' = 523960 bits, limitado pelo tamanho máximo do datagrama IP, k = 524280 bits (ou 65535 bytes [6]). A Tabela 4.1 mostra o número de bits de paridade necessário em cada cenário.

Tabela 4.1 – Número de bits de paridade para o *FEC* nos cenários adaptativos para a Rede C.

			BER								
			1.10^{-2}	1.10^{-3}	1.10^{-4}	1.10^{-5}	1.10^{-6}	1.10^{-7}	1.10^{-8}	1.10^{-9}	1.10^{-10}
FEC	$k = 4320$	$(n-k)$	3	3	3	3	6	13	25	71	337
	$k = 524280$	$(n-k)$	3	6	6	13	25	79	369	2590	23443

A Figura 4.18 mostra os resultados para os cenários adaptativo simples e adaptativo ideal. Como dito anteriormente, o cenário adaptativo simples melhora o desempenho com relação ao não adaptativo, pois regula o número de bits de paridade de acordo com a BER corrente. Quando o tamanho do segmento também é otimizado, como ocorre no cenário adaptativo ideal, o TCP apresenta o melhor desempenho.

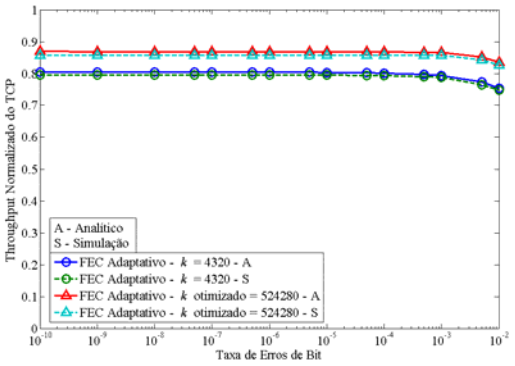


Figura 4.18 – *Throughput* normalizado do TCP para Rede C com *FEC* nos cenários adaptativo simples e adaptativo ideal.

4.2.1.2. Resultados para a Rede S

Para a Rede S, os cenários verificados assim como os respectivos números de bits de informação e de paridade utilizados em cada caso são exatamente os mesmos da seção anterior. Na Figura 4.19 são mostrados os resultados de *throughput* normalizado para os três cenários propostos.

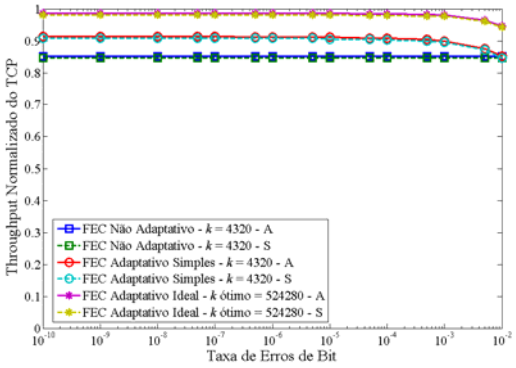


Figura 4.19 – *Throughput* normalizado do TCP para a Rede S com *FEC* nos cenários não adaptativo, adaptativo simples e adaptativo ideal.

Os resultados apresentados para a Rede S confirmam o melhor desempenho do sistema adaptativo ideal com relação ao sistema adaptativo simples e ao não adaptativo. Pode-se verificar também uma pequena redução no desempenho do TCP no cenário não adaptativo, quando comparado ao mesmo cenário da Rede C quando B = 100. Apesar de não mostrada, esta pequena redução também se estende aos outros cenários. Ela ocorre, pois, apesar do valor do tempo de propagação no enlace sem fio não ter impacto direto no desempenho do *FEC* (vide (3.5)), o aumento no valor deste tempo de propagação gera um aumento no atraso global da rede τ, o que degrada (neste caso, apenas ligeiramente) o desempenho do TCP.

4.3 O modelo final

Apesar dos bons resultados obtidos analiticamente para o esquema *FEC*, se comparados com a simulação, o mesmo não se aplica aos modelos *SW-ARQ* e *GBN-ARQ*. Faz-se necessário, portanto, um estudo mais criterioso do impacto da inserção de cada um desses protocolos confiáveis de camada de enlace no modelo analisado.

4.3.1 – Modelamento do *SW-ARQ*

Para que possam ser visualizados os efeitos da inserção do protocolo *SW-ARQ* na camada de enlace do enlace sem fio, é utilizada a mesma rede exemplo da Figura 4.2, com os seguintes parâmetros de tempo adimensionais, escolhidos de forma a simplificar o entendimento da temporização dos eventos: tempo de transmissão de um segmento no enlace com fio t_{seg} = 0,5, tempo de transmissão de um ACK no enlace com fio t_a = 0,1, tempo de propagação no enlace com fio t_p = 7,6, tempo de transmissão de um segmento no enlace sem fio t_{segw} = 0,6, tempo

de transmissão de um ACK no enlace sem fio t_{aw} = 0,2 e tempo de propagação no enlace sem fio t_{pw} = 1,5. A Figura 4.20 ilustra a rede exemplo com o TCP na fase de *Congestion Avoidance*, inicialmente com janela de congestionamento de tamanho dois, no instante t = 1 (t_{seg1} + t_{seg2}). Num primeiro momento, é considerado que a taxa de erros de bit no enlace sem fio é nula. No início, os dois segmentos são transmitidos sequencialmente.

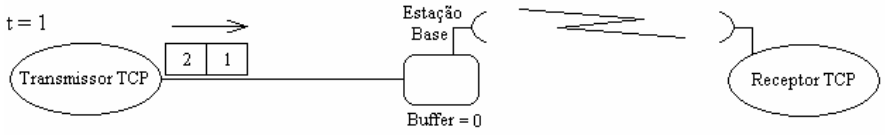


Figura 4.20 – Estado da rede no instante t = 1.

Assim que chega à EB no instante t = 8,1 (0 + t_{seg1} + t_{p1}), ao segmento 1 são adicionados os bits de paridade, e ele é transmitido no enlace sem fio. Uma cópia sua é mantida no *cache* de transmissão (fora do *buffer* principal *B*), e o segmento 2 – ao qual também são anexados bits de paridade – aguarda, no *buffer*, a chegada do ACK correspondente ao segmento 1, quando a partir de então pode ser transmitido. Na Figura 4.21, no instante t = 9,6 (8,1 + t_{segw1} + (t_{pw1} – 0,6)) o segmento 1 começa a ser recebido pelo receptor TCP. No instante t = 11,7 (9,6 + 0,6 + t_{aw1} + (t_{pw1} – 0,2)) o A1 começa a ser recebido pela EB. No instante t = 12,5 (11,7 + 0,2 + t_{segw2}) o segmento 2 está sendo propagado para o receptor, enquanto o A1 é propagado para o transmissor TCP.

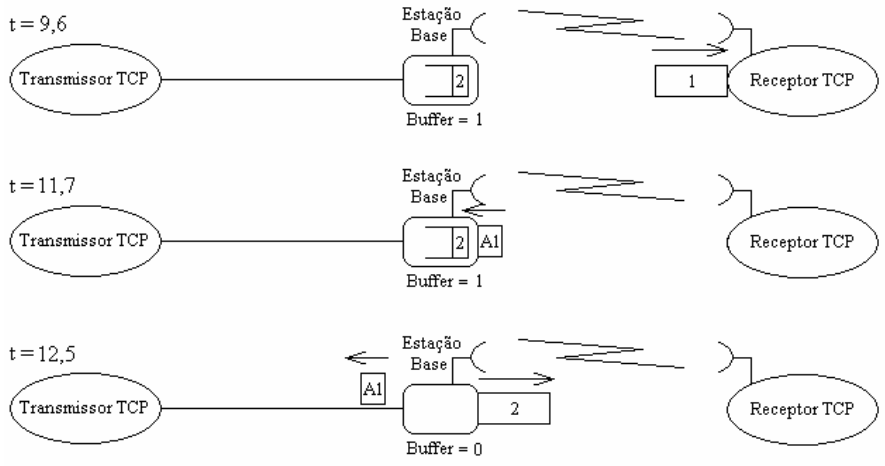


Figura 4.21 – Estado da rede nos instantes t = 9,6, t = 11,7 e t = 12,5.

Considerando sempre que o receptor gera os ACKs TCP e de enlace imediatamente após a chegada correta dos segmentos (lembrando que, para efeito de simplificação, nenhum atraso de processamento para a geração dos ACKs TCP é considerado), o valor do espaçamento

temporal criado entre os ACKs referentes aos segmentos 1 e 2 é de exatamente $t_{SW} = t_{segw} + t_{pw} + t_{aw} + t_{pw}$ = 3,8, como ilustra a Figura 4.22, no instante t = 15,8 (12,5 + $t_{pw2} + t_{aw2} + t_{pw2} + t_{a2}$). Note que A1 (A2) representa os ACKs das camadas de enlace e TCP, referentes ao segmento 1 (segmento 2), agrupados no enlace sem fio. No enlace com fio, A1 (A2) passa a representar somente o ACK do TCP.

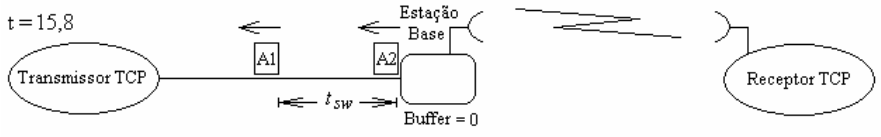


Figura 4.22 – Estado da rede no instante t = 15,8.

Quando o ACK do segmento 1 chega ao transmissor TCP, o segmento 3 é imediatamente transmitido. Em seguida, o ACK do segmento 2 chega ao transmissor, a janela de congestionamento cresce e os segmentos 4 e 5 são transmitidos, como mostra a Figura 4.23, no instante t = 23,6 (15,8 + $t_{p2} + t_{seg4} + t_{seg5}$).

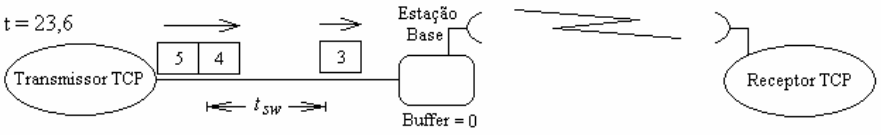


Figura 4.23 – Estado da rede nos instantes t = 23,6.

O segmento 3 é transmitido no enlace sem fio assim que chega à EB. O ACK do segmento 3 e o segmento 4 chegam à EB no mesmo instante, dado o valor de t_{SW}, como mostra a Figura 4.24, pouco antes da chegada de ambos, no instante t = 30,5 (23,6 + 3,3 (tempo ainda restante para a chegada do segmento 3 à EB) + $t_{segw3} + t_{pw3} + t_{aw3} + (t_{pw3} - 0,2)$).

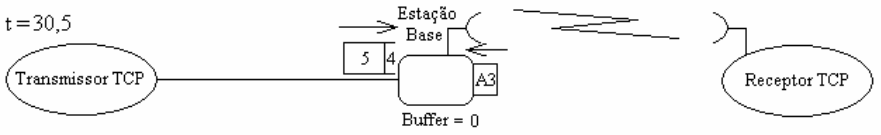


Figura 4.24 – Estado da rede nos instantes t = 30,5.

O segmento 5 aguarda no *buffer* até que o ACK do segmento 4 chegue à EB, quando então é transmitido no enlace sem fio. Entre os ACKs de ambos é criado um espaçamento temporal t_{SW}, exatamente como visto anteriormente e ilustrado na Figura 4.25, no instante t = 38,4 (30,5 + 0,2 + $t_{segw4} + t_{pw4} + t_{aw4} + t_{pw4} + t_{segw5} + t_{pw5} + t_{aw5} + t_{pw5} + t_{a5}$)

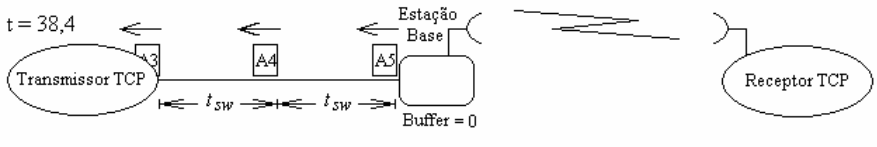


Figura 4.25 – Estado da rede nos instantes t = 38,4.

Este processo se repete diversas vezes até que o *buffer* encha e tenha seu limite atingido, sendo descartado um segmento. O TCP o retransmite e o ciclo recomeça como mostra a Seção 4.1.

Para verificar o impacto da ocorrência dos erros de bit no enlace sem fio, admitindo que os erros de bit ocorram somente na transmissão dos segmentos e nunca nos ACKs, considere que a rede em questão está inicialmente com o *buffer* vazio, no instante t = 0. Admitindo que um erro de bit ocorra no segmento 4 durante a sua transmissão no enlace sem fio, e considerando que o tempo de transmissão de um NACK é equivalente ao de um ACK, o instante em que o NACK do segmento 4 gerado pela camada de enlace do receptor TCP chega à EB é exatamente o mesmo no qual o segmento 5 (cujo espaçamento era de t_{SW} para o segmento 4) é recebido pela EB, como ilustrado na Figura 4.26 quando t = 3,0 ($0 + t_{pw4} + t_{aw4} + (t_{pw4} - 0,2)$).

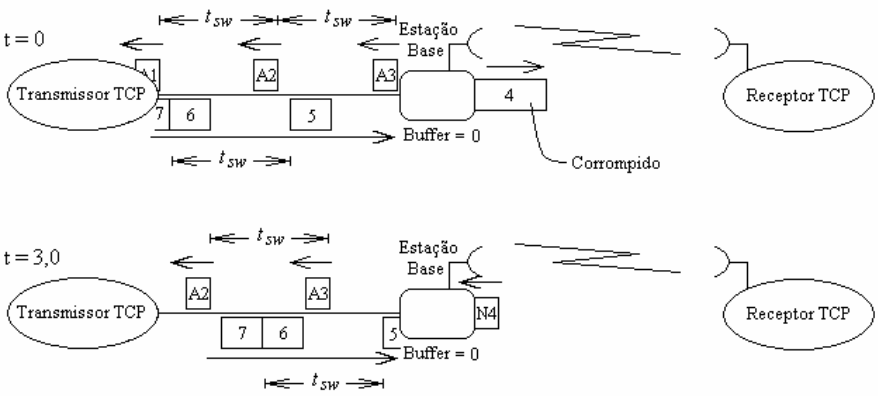


Figura 4.26 – Estado da rede nos instantes t = 0 e t = 3,0.

Como há a necessidade da retransmissão do segmento 4 no enlace sem fio, o segmento 5 não pode ser transmitido e é colocado no *buffer*, que passa nesse momento a ter um segmento. A retransmissão do segmento 4, guardado no *cache* de transmissão, é efetuada com sucesso no instante t = 3,8 ($3,0 + 0,2 + t_{segw4}$) e seu ACK chega à EB no mesmo instante em que o segmento 6 chega à EB para ser transmitido no enlace sem fio, como mostra a Figura 4.27, no instante t = 6,8 ($3,8 + t_{pw4} + t_{aw4} + (t_{pw4} - 0,2)$).

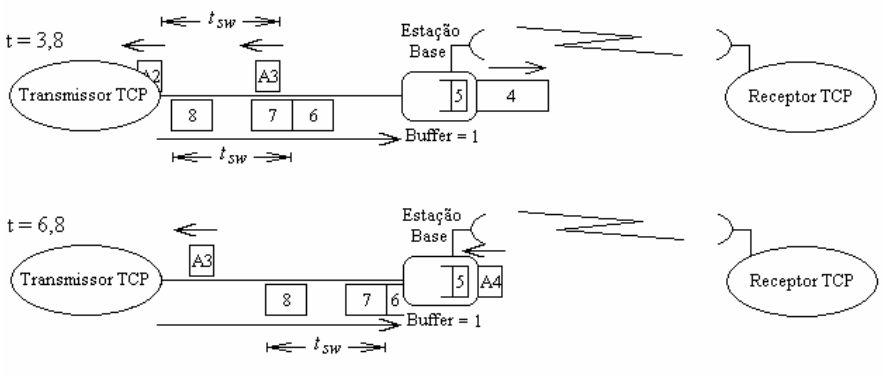


Figura 4.27 – Estado da rede nos instantes t = 3,8 e t = 6,8.

O espaçamento gerado entre os ACKs dos segmentos 3 e 4 é de exatos $2 \cdot t_{SW}$ devido à necessidade de retransmissão desse último segmento. Quando a transmissão do segmento 5 é finalizada pela EB, os segmentos 6 e 7 já estão no *buffer*, que passa a ter 2 segmentos, como mostra a Figura 4.28, no instante t = 7,6 (6,8 + 0,2 + t_{segw5}).

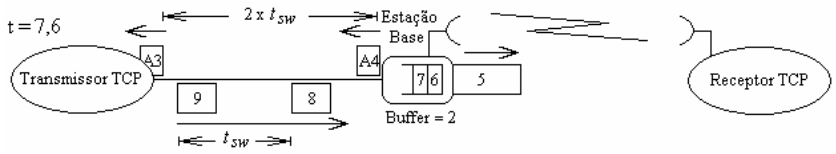


Figura 4.28 – Estado da rede no instante t = 7,6.

Os segmentos 5, 6 e 7 são transmitidos corretamente, e os ACKs gerados com espaçamento t_{SW} entre eles. Entre os segmentos 10 e 11 (transmitidos com a chegada dos ACKs referentes aos segmentos 3 e 4, respectivamente) existirá um espaçamento temporal $2 \cdot t_{SW}$, como mostra a Figura 4.29, no instante t = 18,2 (7,6 + t_{pw5} + t_{aw5} + t_{pw5} + t_{segw6} + t_{pw6} + t_{aw6} + t_{pw6} + t_{segw7} + t_{pw7} + t_{aw7} + (t_{pw7} – 0,2)).

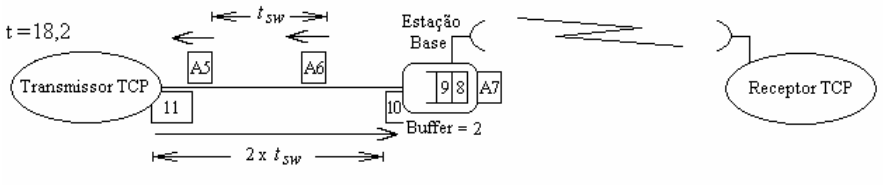


Figura 4.29 – Estado da rede no instante t = 18,2.

O *buffer* continua então com dois segmentos, mas após a chegada do segmento 10 à EB, o espaçamento temporal criado até o segmento 11, que vale $2 \cdot t_{SW}$, corresponde ao tempo necessário para que dois segmentos sejam transmitidos no enlace sem fio (os segmentos 8 e 9, considerando que não ocorrem novas retransmissões). Portanto, no instante em que o

segmento 11 finalmente chega à EB apenas o segmento 10 aguarda sua transmissão e o *buffer* volta a ter apenas um segmento, como mostra a Figura 4.30 no instante t = 25,8 (18,2 + 0,2 + $t_{segw8} + t_{pw8} + t_{aw8} + t_{pw8} + t_{segw9} + t_{pw9} + t_{aw9} + (t_{pw9} - 0{,}2)$).

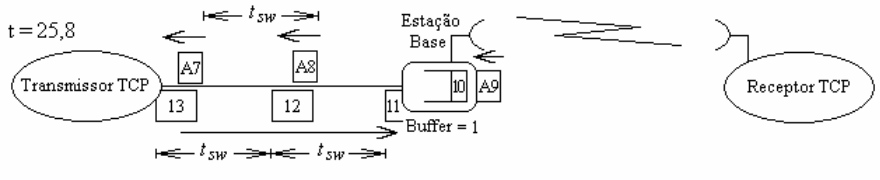


Figura 4.30 – Estado da rede no instante t = 25,8.

O processo continua até que um novo crescimento da janela, caracterizado pelo não espaçamento entre os segmentos 14 e 15, faz com que o *buffer* cresça, passando a ter dois segmentos, como ilustrado na Figura 4.31.

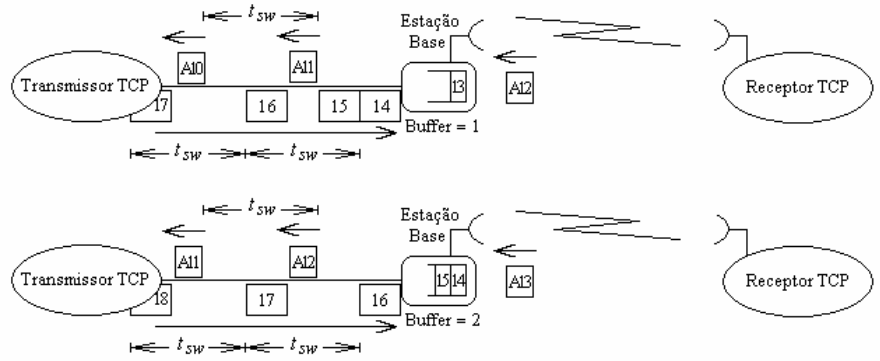


Figura 4.31 – Estado da rede no instante em que o *buffer* volta a ter 2 segmentos.

Em síntese, considerando que a rede já se encontra completamente ocupada de segmentos e ACKs, mas que *buffer* ainda com algum espaço vazio, pode-se afirmar que cada retransmissão no enlace sem fio faz com que:

1. O espaçamento temporal entre o ACK referente ao segmento anterior ao retransmitido e o ACK do próprio segmento retransmitido aumente em 1 t_{SW};
2. A ocupação do *buffer* cresça temporariamente de um segmento;
3. A ocupação do *buffer* decresça de um segmento assim que a lacuna temporal dilatada gerada pela retransmissão, refletida na geração dos segmentos referentes aos ACKs anterior e posterior à retransmissão, "retorne" à EB.

Pode-se então dizer que, quando o protocolo *SW-ARQ* é usado na camada de enlace do enlace sem fio, o crescimento do *buffer*, novamente considerando que a rede já se encontra completamente ocupada de segmentos e ACKs, se dá de duas formas:

- Crescimento efetivo, através do aumento da janela de congestionamento e conseqüente inserção de um novo segmento na rede;
- Crescimento devido à ocorrência de uma retransmissão.

Considere então a rede da Figura 4.2, modelada como a Figura 4.15, cujos parâmetros originais são: $B = 20$, $t_{pw} = 50\mu s$, $t_{aw} = 0{,}9$ ms, $\tau = 500$ ms, $\mu = 100$ pacotes/s, o tamanho do pacote IP é $k = 4320$ bits e a BER = 0. Considerando que são utilizados 88 bits de paridade para o *SW* no enlace sem fio (independente da BER), tem-se um bloco (4320, 4408). Nesse caso $\tau_b = t_{pw} + t_{aw} + t_{pw} = 1$ms e, portanto, através de (4.13), $\tau' = 499{,}95$ ms. Através de (3.6) conclui-se que $\eta_{SW} = 0{,}893$, e portanto, $\mu' = 89{,}3$ pacotes/s. Através das equações (4.16), (4.17) e (4.18) respectivamente, conclui-se que $T' = 0{,}511$ s, $\beta' = 0{,}439$ e $W_{pipe}' \approx 65$ pacotes. O valor de $\mu' \cdot T'$ é, portanto, aproximadamente igual a 45 pacotes. A Figura 4.32 representa a rede com os novos parâmetros.

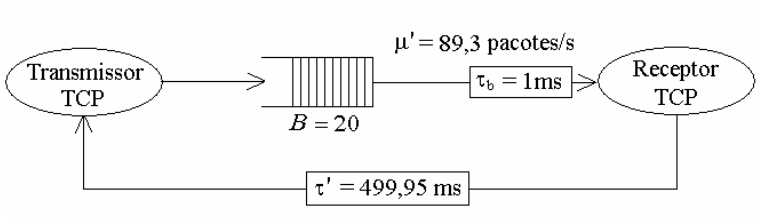


Figura 4.32 – Rede com parâmetros alterados.

Nesta situação sem erros de bit e, portanto, sem retransmissões, a janela de congestionamento cresce até 65 segmentos, quando um segmento é descartado e o TCP executa os mecanismos de *FR*/*FR*, sendo a janela reduzida a exatos $\lfloor 65/2 \rfloor = 32$ segmentos, voltando a crescer a partir de então. Contudo, quando erros de bit e, por conseguinte, retransmissões ocorrem no enlace sem fio, a janela não necessariamente cresce até o seu valor máximo. Para essa rede, por exemplo, se a taxa de erros de bit for elevada para BER = 10^{-8}, todos os valores calculados anteriormente ficam praticamente inalterados. No entanto, a Figura 4.33 mostra que o crescimento da janela de congestionamento pode não necessariamente atingir o valor de W_{pipe}', visto que o segundo crescimento ilustrado se dá apenas até 64 segmentos. Certamente, no instante em que a janela tinha exatos 63 segmentos, uma retransmissão fez com que o *buffer*, que possuía 19 segmentos nesse instante, passasse a ter 20 segmentos chegando ao seu limite. Antes que o tamanho do *buffer* fosse reduzido devido ao retorno da lacuna temporal duplicada gerada pela retransmissão, o crescimento da janela para 64 segmentos fez com que um novo segmento fosse inserido na rede e descartado no *buffer*, ainda com a janela de

congestionamento sem atingir seu limite máximo. Os mecanismos de *FR*/*FR* são executados e em seguida a janela é reduzida aos mesmos $\lfloor 64/2 \rfloor = 32$ segmentos.

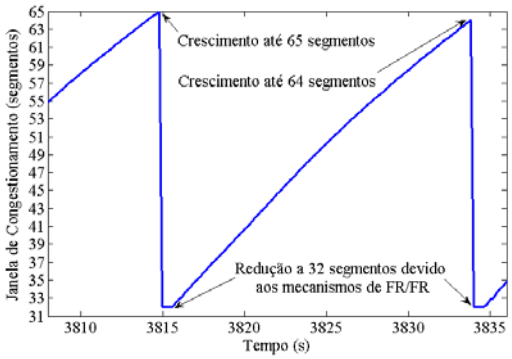


Figura 4.33 – Evolução da Janela de Congestionamento.

Uma outra característica da rede com enlace *SW-ARQ* é a possibilidade da ocorrência de mais de um descarte durante o crescimento da janela, o que acarreta em maiores reduções da mesma. As simulações efetuadas no ns-2 mostram que, além dos casos em que há a ocorrência de um *FR*/*FR*, em algumas situações os mecanismos de *FR*/*FR* são acionados duas vezes devido a dois descartes praticamente em seqüência e, portanto, se o tamanho da janela é W no instante do descarte do primeiro segmento, a janela é reduzida à $\lfloor W/4 \rfloor$. Existem ainda casos em que ocorrem três ou mais descartes em seqüência, sendo que nestes não há ACKs duplicados suficientes para a recuperação através de *FR*/*FR* e um *timeout* ocorre. Esta conclusão está qualitativamente de acordo com [33]. A Figura 4.34 ilustra a evolução da janela de congestionamento da rede anterior quando há a ocorrência de um *timeout* e de um duplo *FR/FR*, respectivamente.

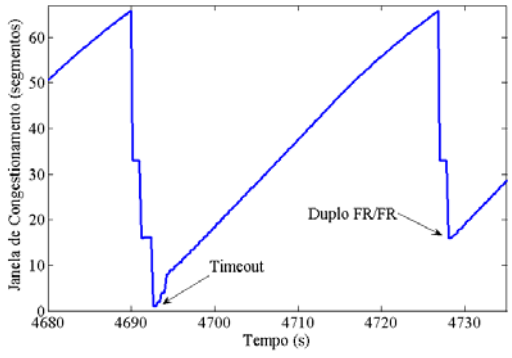


Figura 4.34 – Evolução da Janela de Congestionamento: ocorrência de *timeout* e duplo *FR*/*FR*.

Existem, desta forma, três eventos possíveis ao final do crescimento da janela do TCP: a ocorrência de um *FR/FR*, a ocorrência de dois *FRs/FRs* em seqüência ou ainda a ocorrência de um *timeout*.

Portanto, se for possível calcular o *throughput* associado a cada um desses eventos, assim como suas probabilidades de ocorrência, é possível calcular o *throughput* médio de toda a transmissão TCP. Neste caso, o *throughput* médio será dado por:

$$\bar{\lambda} = P_1.\lambda_1 + P_2.\lambda_2 + P_T.\lambda_T\,, \tag{4.20}$$

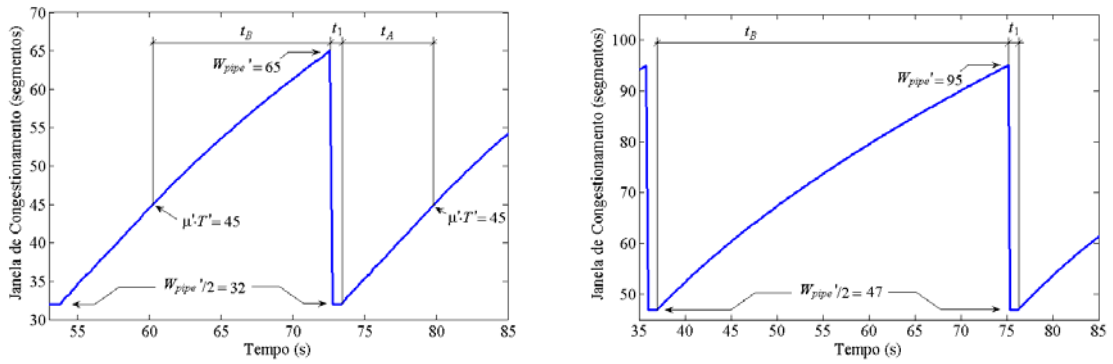
onde P_1 é a probabilidade da ocorrência de somente um *FR/FR* na mesma janela e λ_1 é o *throughput* associado à ocorrência deste evento; P_2 é a probabilidade da ocorrência de dois *FR/FR* e λ_2 é o *throughput* associado à ocorrência deste evento; P_T é a probabilidade da ocorrência de um *timeout* e λ_T é o *throughput* associado à ocorrência deste evento.

4.3.1.1. Cálculo dos *throughputs*

Inicialmente, é feita a seguinte consideração simplificadora: toda janela cresce até o valor exato do tamanho máximo médio W_{pipe}' dado por (4.18). Para que possa ser verificado o *throughput* de cada evento, é considerado ainda que cada ciclo (apesar dos eventos não mais serem cíclicos, esta nomenclatura será mantida) tem início na sub-fase *B*, mais precisamente no ponto $\mu' \cdot T'$ (quando $\beta' < 1$) ou no ponto $W_{pipe}'/2$ (quando $\beta' > 1$), ocorrendo em seguida a redução da janela (através de um ou dois *FR/FR* ou ainda através de um *timeout*). A seguir, a fase de *Slow Start* (somente no caso da ocorrência de um *timeout*) e por fim a ocorrência da sub-fase *A* (caso $\beta' < 1$) ou o retorno a sub-fase *B* (caso $\beta' > 1$), quando finalmente o ciclo se encerra. Apesar de temporalmente não ser a maneira mais intuitiva de avaliar um ciclo, esta metodologia possibilita que os efeitos do tipo de redução efetuada sejam computados no próprio ciclo, não tendo influência alguma no ciclo subseqüente.

4.3.1.1.1. Cálculo do *throughput* associado ao evento de 1 *FR/FR*

A Figura 4.35 mostra detalhadamente a evolução da janela de congestionamento quando da ocorrência de um *FR/FR*, para $\beta' < 1$ (rede da Figura 4.32) e $\beta' > 1$ (rede da Figura 4.32, alterando o *buffer* para $B = 50$), respectivamente.



(a) (b)

Figura 4.35 – Evolução da janela de congestionamento para o evento de 1 *FR/FR* quando: (a) $\beta' < 1$ e (b) $\beta' > 1$.

No caso de $\beta' < 1$, a janela cresce a partir de $\mu' \cdot T'$ na sub-fase *B*, um segmento é descartado e o TCP se recupera da perda através de ACKs duplicados. A janela é reduzida à metade e em seguida volta a crescer, passando pela sub-fase *A* e finalizando no início da sub-fase *B* quando a janela novamente tem $\mu' \cdot T'$ segmentos. Caso $\beta' > 1$, a janela cresce a partir de $W_{pipe}'/2$ na sub-fase *B*, um segmento é descartado e o TCP se recupera da perda através de ACKs duplicados. A janela é reduzida à metade e o ciclo se encerra no início da próxima sub-fase *B*, quando novamente a janela tem $W_{pipe}'/2$ segmentos. Considerando a existência de um intervalo t_1, entre o fim da sub-fase *B* e a retomada do crescimento da janela, o *throughput* médio gerado na ocorrência do evento de somente um *FR/FR* pode ser calculado como:

$$\lambda_1 = \begin{cases} \dfrac{n_{B3} + n_{A2}}{t_{B3} + t_1 + t_{A2}}, & \text{caso } \beta' < 1 \\ \dfrac{n_{B4}}{t_{B4} + t_1}, & \text{caso } \beta' > 1 \end{cases} \tag{4.21}$$

onde t_{B3} e n_{B3} são definidos pelas equações (4.8) e (4.9) com $W_{iB} = \mu' \cdot T'$ e $W_{fB} = W_{pipe}'$; t_{A2} e n_{A2} são definidos por pelas equações (4.6) e (4.7) com $W_{iA} = W_{pipe}'/2$ e $W_{fA} = \mu' \cdot T'$ e t_{B4} e n_{B4} são definidos por pelas equações (4.8) e (4.9) com $W_{iB} = W_{pipe}'/2$ e $W_{fB} = W_{pipe}'$. O parâmetro t_1 foi definido através de observações das simulações efetuadas. Apesar de seu valor apresentar variações dependendo do valor de β', uma boa aproximação para o seu valor é o do seu próprio valor mínimo, dado com $\beta' < 1$:

$$t_1 \approx t_{1\,\min} = (W_{pipe}'/2) \cdot (1/\mu') \tag{4.22}$$

4.3.1.1.2. Cálculo do *throughput* associado ao evento de 2 *FR/FR*

A Figura 4.36 mostra detalhadamente a evolução da janela de congestionamento quando da ocorrência de dois descartes de segmentos e, por conseguinte, dois *FR/FR*, nos casos $\beta' < 1$ (rede da Figura 4.32), $1 < \beta' < 3$ (rede da Figura 4.32, alterando o *buffer* para $B = 50$) e $\beta' > 3$ (rede da Figura 4.32, alterando o *buffer* para $B = 150$).

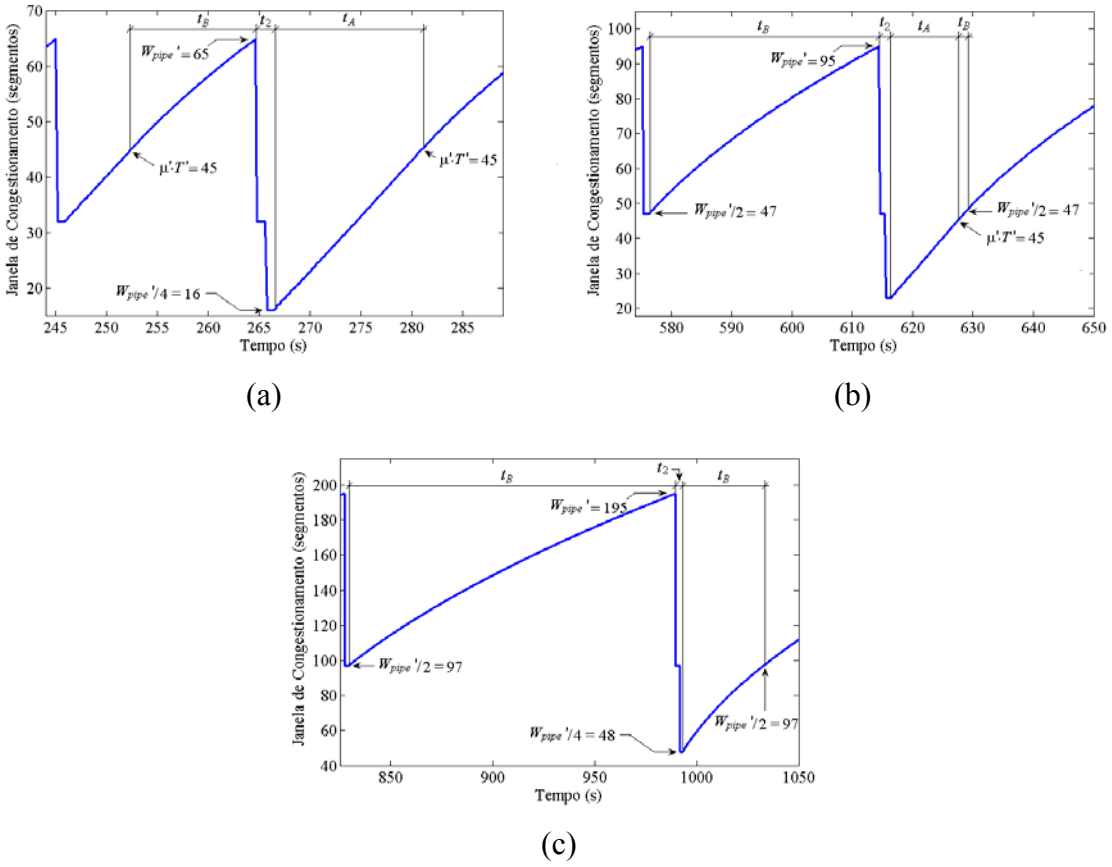


Figura 4.36 – Evolução da janela de congestionamento para o evento de 2 *FR/FR* quando: (a) $\beta' < 1$, (b) $1 < \beta' < 3$ e (c) $\beta' > 3$.

No caso de $\beta' < 1$, a janela cresce a partir de $\mu' \cdot T'$ na sub-fase *B*, dois segmentos são descartados ao seu final e o TCP se recupera das perdas através de ACKs duplicados. Em seguida a janela volta a crescer na sub-fase *A* até atingir novamente $\mu' \cdot T'$, quando o ciclo se encerra. Caso $1 < \beta' < 3$ ($W_{pipe}'/4 < \mu' \cdot T' < W_{pipe}'/2$) o início do ciclo se dá em $W_{pipe}'/2$, na sub-fase *B*. Após as duas perdas, a janela cresce de $W_{pipe}'/4$ até $\mu' \cdot T'$ na sub-fase *A*. Em seguida, a sub-fase *B* leva a janela de $\mu' \cdot T'$ até $W_{pipe}'/2$, quando o ciclo se encerra. Caso $\beta' > 3$

($\mu' \cdot T' > W_{pipe}'/4$), extingue-se a sub-fase *A* após a redução da janela, quando o crescimento entra diretamente na sub-fase *B*, indo de $W_{pipe}'/4$ até $W_{pipe}'/2$, quando o ciclo se encerra.

Novamente, considerando a existência de um intervalo t_2, entre o fim da sub-fase *B* e o início do crescimento da janela, o *throughput* médio gerado na ocorrência do evento de dois *FR/FR* pode ser calculado como:

$$\lambda_2 = \begin{cases} \dfrac{n_{B3} + n_2 + n_{A3}}{t_{B3} + t_2 + t_{A3}}, \text{ caso } \beta < 1 \\ \dfrac{n_{B4} + n_2 + n_{A3} + n_{B5}}{t_{B4} + t_2 + t_{A3} + t_{B5}}, \text{ caso } 1 < \beta < 3 \\ \dfrac{n_{B4} + n_2 + n_{B6}}{t_{B4} + t_2 + t_{B6}}, \text{ caso } \beta > 3 \end{cases} \quad (4.23)$$

onde t_{A3} e n_{A3} são definidos pelas equações (4.6) e (4.7) com $W_{iA} = W_{pipe}'/4$ e $W_{fA} = \mu' \cdot T'$; t_{B5} e n_{B5} são definidos pelas equações (4.8) e (4.9) com $W_{iB} = \mu' \cdot T'$ e $W_{fB} = W_{pipe}'/2$ e t_{B6} e n_{B6} são definidos pelas equações (4.8) e (4.9) com $W_{iB} = W_{pipe}'/4$ e $W_{fB} = W_{pipe}'/2$. Assim como o parâmetro t_1, os parâmetros t_2 e n_2 foram definidos através de observações das simulações efetuadas. O valor de t_2 para todos os casos é aproximado para seu valor mínimo, dado quando $\beta' < 1$, que é de aproximadamente:

$$t_2 \approx t_{2\,\min} \approx (W_{pipe}'/2) \cdot (1/\mu') + 2 \cdot T' \,. \quad (4.24)$$

Por sua vez, o valor de n_2 será de aproximadamente metade da janela máxima de congestionamento, ou seja:

$$n_2 \approx (W_{pipe}'/2) \,. \quad (4.25)$$

4.3.1.1.3. Cálculo do *throughput* associado ao evento de um *timeout*

A Figura 4.37 mostra detalhadamente a evolução da janela de congestionamento quando da ocorrência da perda de três ou mais segmentos e, por conseguinte, na ocorrência de um *timeout*. São ilustrados os casos onde $\beta' < 1$ (rede da Figura 4.32), $1 < \beta' < 7$ (rede da Figura 4.32, alterando o *buffer* para $B = 150$) e $\beta' > 7$ (rede da Figura 4.32, alterando o *buffer* para $B = 350$).

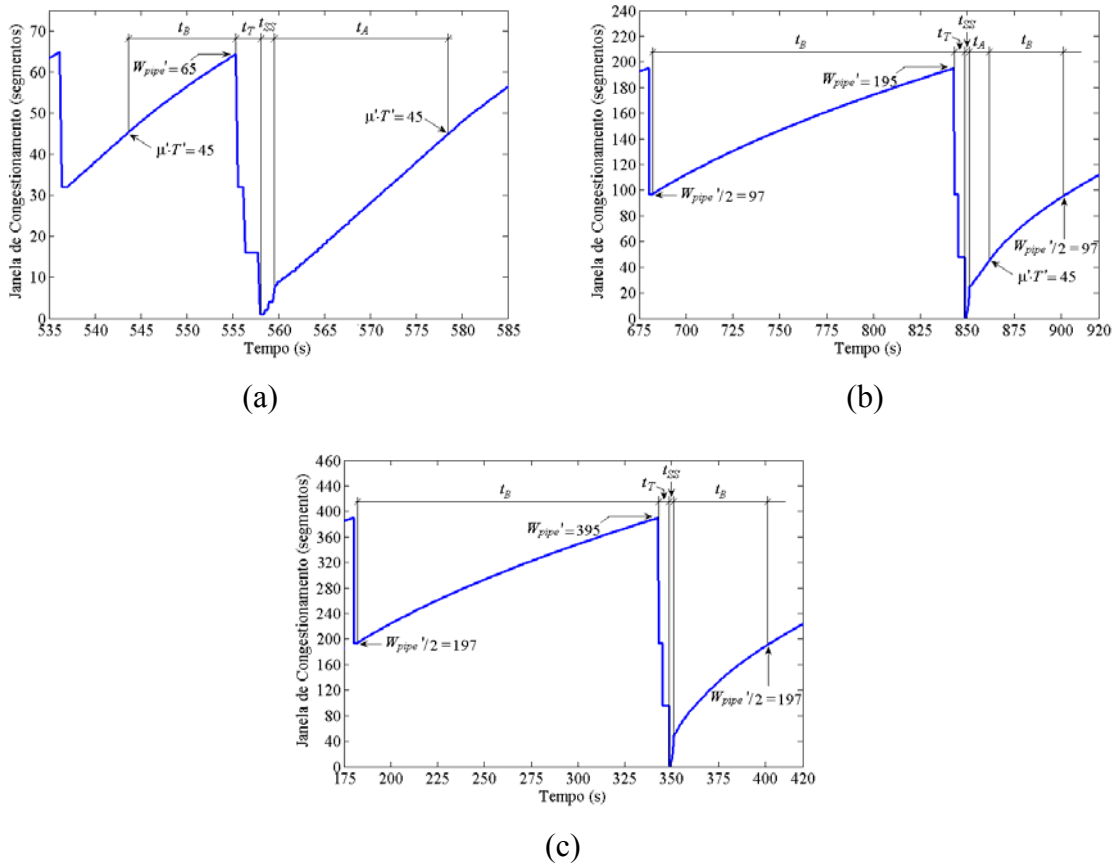


Figura 4.37 – Evolução da janela de congestionamento para o evento de 2 *FR/FR* quando: (a) $\beta' < 1$, (b) $1 < \beta' < 7$ e (c) $\beta' > 7$.

Caso $\beta' < 1$, a janela cresce a partir de $\mu' \cdot T'$ passando pela sub-fase *B*, três ou mais segmentos são descartados e o TCP se recupera através de ACKs duplicados nas duas primeiras perdas, mas um *timeout* ocorre para o terceiro segmento perdido. A janela é reduzida à unidade, cresce exponencialmente até $W_{pipe}'/8$, caracterizando a fase de *Slow Start*, quando finalmente entra na sub-fase *A* e volta até $\mu'.T'$. Caso $1 < \beta' < 7$, a janela se inicia em $W_{pipe}'/2$ e cresce na sub-fase *B* até o valor máximo W_{pipe}', quando as três ou mais perdas levam ao *timeout*. Após a fase de *Slow Start*, o crescimento da janela entra na sub-fase *A* até atingir $\mu'.T'$, quando finalmente entra na sub-fase *B* até atingir $W_{pipe}'/2$, quando o ciclo se encerra. Caso $\beta' > 7$, a sub-fase *A* se extingue e após o *Slow Start*, o crescimento entra diretamente na sub-fase *B*, indo de $W_{pipe}'/8$ até $W_{pipe}'/2$.

Novamente, considerando a existência de um intervalo t_T, entre o fim da sub-fase *B* e o início do *Slow Start*, o *throughput* médio gerado na ocorrência do evento de um *timeout* pode ser calculado como:

$$\lambda_T = \begin{cases} \dfrac{n_{B3} + n_T + n_{SS2} + n_{A4}}{t_{B3} + t_T + t_{SS2} + t_{A4}}, & \text{caso } \beta < 1 \\ \dfrac{n_{B4} + n_T + n_{SS2} + n_{A4} + n_{B5}}{t_{B4} + t_T + t_{SS2} + t_{A4} + t_{B5}}, & \text{caso } 1 < \beta < 7 \\ \dfrac{n_{B4} + n_T + n_{SS2} + n_{B7}}{t_{B4} + t_T + t_{SS2} + t_{B7}}, & \text{caso } \beta > 7 \end{cases} \quad (4.26)$$

onde t_{SS2} e n_{SS2} são definidos pelas equações (4.3) e (4.4) com e $W_{fSS} = W_{pipe}'/8$; t_{A4} e n_{A4} são definidos por pelas equações (4.6) e (4.7) com $W_{iA} = W_{pipe}'/8$ e $W_{fA} = \mu' \cdot T'$ e t_{B7} e n_{B7} são definidos por pelas equações (4.8) e (4.9) com $W_{iB} = W_{pipe}'/8$ e $W_{fB} = W_{pipe}'/2$. Assim com os parâmetros t_1, t_2 e n_2 os parâmetros t_T e n_T foram definidos através de observações das simulações efetuadas. O valor de t_T para todos os casos será aproximado para seu valor mínimo, dado quando $\beta' < 1$, que é de aproximadamente:

$$t_T \approx t_{T\min} \approx (W_{pipe}'/2) \cdot (1/\mu') + 2 \cdot T' + \max(1; 1{,}5 \cdot T') \,. \quad (4.27)$$

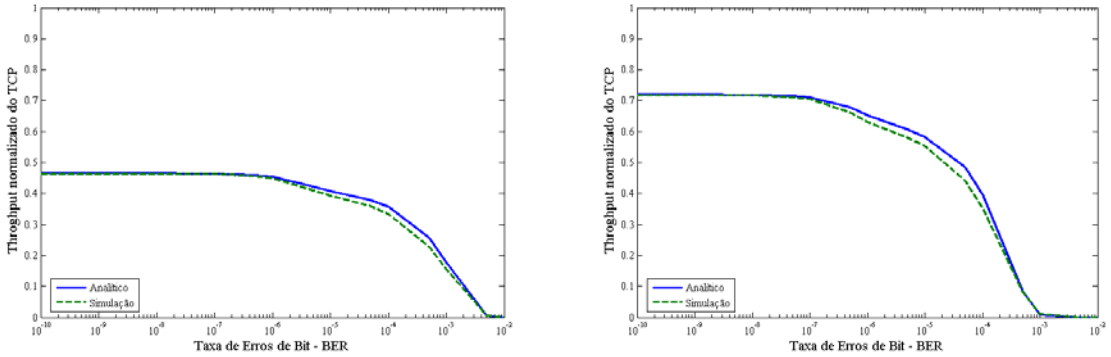
As observações da simulação mostram também que o valor de n_2 é de aproximadamente metade da janela máxima de congestionamento, ou seja:

$$n_T \approx (W_{pipe}'/2) \,. \quad (4.28)$$

4.3.1.1.4. Validação do cálculo dos *throughputs* associados a cada evento utilizando probabilidades de eventos extraída da simulação

Para validar os *throughputs* médios de cada evento, é comparado o valor do *throughput* normalizado obtido por simulação com o *throughput* normalizado calculado utilizando os *throughputs* médios de cada evento. Os valores das probabilidades dos eventos (PEs) são extraídos da própria simulação, através do processo de contagem.

A rede utilizada para validação dos *throughputs* médios dos eventos é a mesma ilustrada na Figura 4.32. Para a validação, é utilizado o *SW-ARQ* não adaptativo, com códigos (856,820) e (4408,4320). A taxa de erros de bit no enlace sem fio é variada entre 10^{-10} e 10^{-2}. A Figura 4.38 compara o *throughput* calculado analiticamente (através dos *throughputs* médios) com o obtido através da simulação no NS-2, para os dois tamanhos de blocos utilizados.



(a) (b)

Figura 4.38 – *Throughputs* normalizados do TCP obtidos analiticamente e através de simulação para (a) código (856,820) e (b) código (4408,4320).

As diferenças entre as curvas geradas analiticamente e através de simulação ilustradas na Figura 4.38 podem ser explicadas pela consideração feita de que em todos os ciclos a janela cresce até seu tamanho máximo médio dado por (4.18), o que de fato não é verdade para taxas de erros de bit intermediárias e altas ($> \sim 10^{-7}$), fazendo com que principalmente nesta faixa o modelo analítico seja ligeiramente sobreestimado. Contudo, essa comparação permite afirmar que o modelo dos *throughputs* médios gera boa aproximação para o *throughput* real do TCP. O próximo passo passa a ser a obtenção do equacionamento analítico das probabilidades associadas a cada evento, fechando, desta forma, o modelamento do *SW-ARQ*.

4.3.1.2. Cálculo das probabilidades dos eventos

Para que se possa gerar a probabilidade dos eventos similar à extraída na simulação, é importante lembrar que durante o ciclo, a janela de congestionamento não necessariamente chega até W_{pipe}', como ilustrado na Seção 4.3.1.1. Desta forma, para que o cálculo das probabilidades dos eventos seja mais próximo da realidade encontrada na simulação, não é feita a consideração que a janela necessariamente cresce até W_{pipe}', como feito até aqui.

A partir desse ponto também é definida a entidade "rodada de janela". Uma rodada de janela se inicia no primeiro segmento transmitido após um crescimento da janela e se encerra no segmento que gera o próximo crescimento da janela. Portanto, se dois segmentos pertencem à mesma rodada de janela, no instante em que eles foram transmitidos a janela de congestionamento possuía exatamente o mesmo tamanho. Se uma rodada de janela X é subseqüente a uma rodada de janela Y, certamente o tamanho da janela durante a rodada de

janela *X* é diferente do tamanho da janela durante a rodada de janela *Y*. Comumente, será utilizado o tamanho da janela para identificar a rodada de janela.

Antes de passar ao cálculo dos eventos propriamente ditos, faz-se necessário entender a seguinte associação utilizada: nos casos em que o início do ciclo é iniciado com a rodada de janela $\mu' \cdot T'$ ($\beta < 1$) considerar-se-á que nesse instante o *buffer* estará vazio, sendo preenchido a cada nova rodada de janela até que atinja seu limite e um ou mais segmentos sejam descartados. Nos casos em que o início do ciclo é dado com a rodada de janela $W_{pipe}'/2$, será considerado que o *buffer* nesse instante tem exatos $\lfloor W_{pipe}'/2 - \mu' \cdot T' \rfloor$ segmentos, sendo preenchido com o passar das rodadas de janela, até que tenha seu limite atingido e um ou mais segmentos sejam descartados.

No caso de rodadas de janela inferiores à W_{pipe}', quatro eventos podem ocorrer: uma nova rodada de janela pode ser alcançada (crescimento efetivo da janela sem que haja descartes), 1 *FR/FR*, 2 *FRs/FRs* ou ainda um *timeout*. Se a rodada de janela W_{pipe}' é atingida, ou seja, caso não haja descartes em rodadas de janela anteriores à rodada de janela W_{pipe}', apenas os três últimos eventos podem ocorrer, não sendo possível, portanto, uma nova rodada de janela sem que haja ao menos uma perda.

Uma característica importante do modelo proposto é a ausência de memória. De fato, é feita a consideração de que na rodada de janela anterior à rodada de janela corrente, o número de transmissões de blocos efetuadas no enlace sem fio é exatamente o número médio (ou número esperado, que será mostrado mais adiante) levando em consideração a BER. Obviamente o número exato de transmissões efetuadas na rodada de janela anterior pode ainda ser menor ou maior do que número esperado calculado. No entanto, considerar que o número de transmissões efetuadas na rodada de janela anterior é exatamente o número esperado facilita o modelamento do sistema e se mostra uma aproximação válida, como mostrarão os resultados mais adiante.

4.3.1.2.1. Cálculo da probabilidade do evento de nenhum descarte no *buffer* dada a rodada de janela $W \leq W_{pipe}'$

Uma boa aproximação para a ocorrência deste evento é a seguinte: não haverá descarte de segmentos no *buffer* caso o número de transmissões dos segmentos na rodada de janela *W* no

enlace sem fio não ultrapasse o número esperado de transmissões destes segmentos somado ao espaço disponível no *buffer* no início da rodada de janela menos um. Caso o número de transmissões ultrapasse esse valor, pelo menos um descarte ocorrerá no *buffer*.

A probabilidade de que o número de transmissões no enlace sem fio para uma rodada de janela de tamanho J qualquer não ultrapasse um determinado limiar LT ($LT \geq J$), dada a probabilidade P_c de um segmento ser transmitido com sucesso no enlace sem fio (definida em (3.7)) pode ser escrita como

$$P_{nu} = P_c^J + \sum_{i=J+1}^{LT} \left(\binom{i}{J} - \binom{i-1}{J} \right) \cdot (1-P_c)^{i-J} \cdot P_c^J \ . \tag{4.29}$$

A probabilidade do evento de nenhum descarte de pacote no *buffer*, dada a rodada de janela de tamanho W, é, portanto, dada por

$$P_{NW} = \begin{cases} P_{nu1}, \text{se} \ \ W < W_{pipe}' \\ 0, \text{caso contrário} \end{cases} \ , \tag{4.30}$$

onde P_{nu1} é definido por (4.29), com $J = W$ e $LT = E + b - 1$, onde b é o espaço disponível no *buffer* no início da rodada de janela e E é o número esperado de transmissões no enlace sem fio para os segmentos da rodada de janela de tamanho J, dado aproximadamente por

$$E = \left\lfloor \frac{1}{P_c} \cdot J \right\rfloor \ . \tag{4.31}$$

Como dito anteriormente, quando $W = W_{pipe}'$ há certamente um descarte final da rodada de janela e, portanto, a probabilidade de nenhum descarte é nula.

Para exemplificar, seja uma rede na qual $W_{pipe}' = 20$ segmentos e $B = 10$. Estando esta rede na rodada de janela $W = 15$, o espaço disponível no *buffer* no início desta rodada de janela era de 3 segmentos. Considerando $P_c = 0{,}833$, tem-se $E = 18$ transmissões no enlace sem fio. Caso o número efetivo de transmissões seja menor ou igual à $LT = 18 + 3 - 1 = 20$ transmissões, não haverá descarte no *buffer*. Isto ocorre, pois na rodada de janela anterior, quando $W = 14$ segmentos, foi considerado que foram efetuadas exatamente 17 transmissões (dado por $\lceil (1/P_c) \cdot W \rceil = 17$) e, portanto, o *buffer* que poderia terminar a rodada de janela com espaço disponível de até 6 segmentos, termina com espaço disponível igual a $20 - 17 = 3$ segmentos. No entanto, na rodada de janela seguinte ($W = 15$ segmentos) essa ocupação adicional de três

segmentos será desfeita com a chegada das lacunas temporais geradas na rodada de janela anterior (W = 14 segmentos), como visto na Seção 4.3.1. Com o incremento do tamanho da janela, de 14 para 15 segmentos, a ocupação do *buffer* cresce de um segmento, ou seja, o espaço livre passa a 5 segmentos. Com isso, caso o número de transmissões não ultrapasse o número de 20, ou seja, caso ocorram somente 5 retransmissões durante a rodada de janela W = 15, não haverá descarte no *buffer* e a rodada de janela W = 16 segmentos será alcançada.

4.3.1.2.2. Cálculo da probabilidade do evento de 1 *FR/FR* dada a rodada de janela $W \leq W_{pipe}$'

Para calcular a probabilidade deste evento é necessário entender que no caso de um descarte, uma nova rodada de janela inteira ainda é transmitida até que o TCP perceba este descarte através da chegada de ACKs duplicados e o segmento descartado seja retransmitido. Portanto, será considerado que o evento de 1 *FR/FR* ocorrerá quando o número de transmissões dos segmentos da rodada de janela W no enlace sem fio for exatamente igual ao número esperado de transmissões somado ao espaço livre no *buffer* e que o número de transmissões ocorridas na rodada de janela seguinte ao descarte não ultrapasse o número esperado de transmissões desta rodada de janela mais o espaço disponível no *buffer* no início da rodada de janela anterior. Isso significa que se considerará que na rodada de janela de tamanho W nenhum descarte devido a uma retransmissão pode ocorrer, sendo descartado um segmento somente ao final da rodada de janela, após o crescimento efetivo. Na rodada de janela seguinte não pode haver descarte algum; caso contrário cair-se-ia na condição de 2 *FR/FR* ou ainda de um *timeout*.

A probabilidade de que o número de transmissões no enlace sem fio para os segmentos de uma rodada de janela J qualquer seja igual a NT ($NT \geq J$), dada a probabilidade P_c de um segmento ser transmitido com sucesso nesse enlace, pode ser escrita como

$$P_i = \begin{cases} P_c^{NT}, \text{se } NT = J \\ \left(\binom{NT}{J} - \binom{NT-1}{J} \right) \cdot (1-P_c)^{NT-J} \cdot P_c^{\,J}, \text{se } NT > J \end{cases}. \tag{4.32}$$

A probabilidade do evento de 1 *FR/FR* dada uma rodada de janela de tamanho W pode então ser dada por

$$P_{1W} = P_{i1} \cdot P_{nu2} \tag{4.33}$$

onde P_{i1} e P_{nu2} são definidos, respectivamente, pelas equações (4.32) e (4.29) com $J = W$ e $NT = E + b$.

4.3.1.2.3. Cálculo da probabilidade do evento de 2 *FR/FR* dada a rodada de janela $W \leq W_{pipe}$'

Para efetuar esse cálculo devem ser levadas em conta as duas possibilidades para este evento. No primeiro caso ocorrerá o evento de 2 *FR/FR* se o número de transmissões dos segmentos da rodada de janela W no enlace sem fio for exatamente igual ao número esperado de transmissões somado ao espaço livre no *buffer e* se o número de transmissões ocorridas na rodada de janela seguinte for igual ao número esperado de transmissões desta rodada de janela adicionado ao espaço anteriormente disponível no *buffer* mais um. O segundo caso ocorrerá se o número de transmissões dos segmentos da rodada de janela W no enlace sem fio for exatamente igual ao número esperado de transmissões somado ao espaço livre no *buffer* mais um *e* se o número de transmissões ocorridas na "meia rodada de janela" seguinte não ultrapasse o número esperado de transmissões desta "meia rodada de janela" mais metade do espaço anteriormente disponível no *buffer*. Ou seja, no primeiro caso uma quantidade de segmentos igual à W (praticamente uma nova rodada de janela) é transmitida após o primeiro descarte, visto que este só ocorre ao final da rodada de janela corrente. No segundo caso é feita uma aproximação de que o primeiro dos dois descartes na janela W ocorre para um segmento aproximadamente no meio da rodada de janela, e como uma quantidade de segmentos aproximadamente igual a W será transmitida após esse descarte, apenas $W/2$ segmentos serão transmitidos após o término da rodada de janela W (aproximadamente metade de uma nova rodada de janela).

Portanto, calcular o evento de 2 *FR/FR* dada uma janela de tamanho W consiste em somar as expressões dadas pelo primeiro caso, ou seja

$$P_{2W} = P_{i1} \cdot P_{i2} + P_{i2} \cdot P_{nu3} \tag{4.34}$$

onde P_{i2} é definido por (4.32) com $J = W$ e $NT = E + b + 1$ e P_{nu3} é definido por (4.29) com $J = W/2$ e $NT = E + \lfloor b/2 \rfloor$.

4.3.1.2.4. Cálculo da probabilidade do evento de *timeout* dada a rodada de janela $W \leq W_{pipe}$'

A probabilidade do evento de *timeout* dada a rodada de janela $W \leq W_{pipe}$' será dada por

$$P_{T\,W} = 1 - P_{NW} - P_{1W} - P_{2W} - P_{NW}\,. \tag{4.35}$$

4.3.1.2.5. Cálculo da probabilidade geral do evento de 1 *FR*/*FR*

Lembrando que quando $\beta' < 1$ o ciclo se inicia na rodada de janela $W = \mu' \cdot T'$ com o *buffer* vazio e ainda que, quando $\beta' > 1$, o ciclo se inicia em $W = W_{pipe}'/2$ com o *buffer* com aproximadamente $\theta = \lfloor W_{pipe}'/2 - \mu' \cdot T' \rfloor$ pacotes armazenados, a probabilidade geral do evento de 1 *FR*/*FR* pode ser escrita como

$$P_1 = \begin{cases} P_{1W1} + \sum_{i=1}^{B} \left[\left(\prod_{j=0}^{i-1} P_{NW1} \right) \cdot P_{1W2} \right], \text{se } \beta' < 1 \\ P_{1W3} + \sum_{i=1}^{B-\theta} \left[\left(\prod_{j=0}^{i-1} P_{NW2} \right) \cdot P_{1W4} \right], \text{se } \beta' > 1 \end{cases} \tag{4.36}$$

onde P_{1W1} e P_{1W3} representam as probabilidades da ocorrência de 1 *FR/FR* na rodada de janela inicial do ciclo, e são definidas por (4.33) com, respectivamente, $W = (\mu' \cdot T')$ e $b = B$, $W = (W_{pipe}'/2)$ e $b = (B - \theta)$. Por sua vez, P_{1W2} e P_{1W4} representam as probabilidades da ocorrência de 1 *FR/FR* nas rodadas de janela subseqüentes à inicial, e são definidas por (4.33) com, respectivamente, $W = (\mu' \cdot T' + i)$ e $b = (B - i)$, $W = (W_{pipe}'/2 + i)$ e $b = (B - \theta - i)$. Os produtórios $\left(\prod_{j=0}^{i-1} P_{NW1} \right)$ e $\left(\prod_{j=0}^{i-1} P_{NW2} \right)$ representam as probabilidades das rodadas de janela $W = (\mu' \cdot T' + i)$ e $W = (W_{pipe}'/2 + i)$, respectivamente, serem alcançadas durante o ciclo. P_{NW1} e P_{NW2} são definidos por (4.29), respectivamente com $W = (\mu' \cdot T' + j)$ e $b = (B - j)$, e com $W = (W_{pipe}'/2 + j)$ e $b = (B - \theta - j)$.

4.3.1.2.6. Cálculo da probabilidade geral do evento de 2 *FR/FR*

Similarmente ao caso anterior, a probabilidade geral do evento de 2 *FR*/*FR* pode ser calculada como

$$P_2 = \begin{cases} P_{2W1} + \sum_{i=1}^{B} \left[\left(\prod_{j=0}^{i-1} P_{NW1} \right) \cdot P_{2W2} \right], \text{se } \beta' < 1 \\ P_{2W3} + \sum_{i=1}^{B-\theta} \left[\left(\prod_{j=0}^{i-1} P_{NW2} \right) \cdot P_{2W4} \right], \text{se } \beta' > 1 \end{cases} \tag{4.37}$$

onde P_{2W1} e P_{2W3} representam as probabilidades da ocorrência de 2 *FR/FR* na rodada de janela inicial do ciclo, e são definidas por (4.34) com, respectivamente, $W = (\mu' \cdot T')$ e $b = B$, $W = (W_{pipe}'/2)$ e $b = (B - \theta)$. Por sua vez, P_{2W2} e P_{2W4} representam as probabilidades da ocorrência de 2 *FR/FR* nas rodadas de janela subseqüentes à inicial, e são definidas por (4.34) com, respectivamente, $W = (\mu' \cdot T' + i)$ e $b = (B - i)$, $W = (W_{pipe}'/2 + i)$ e $b = (B - \theta - i)$.

4.3.1.2.7. Cálculo da probabilidade geral do evento de *timeout*

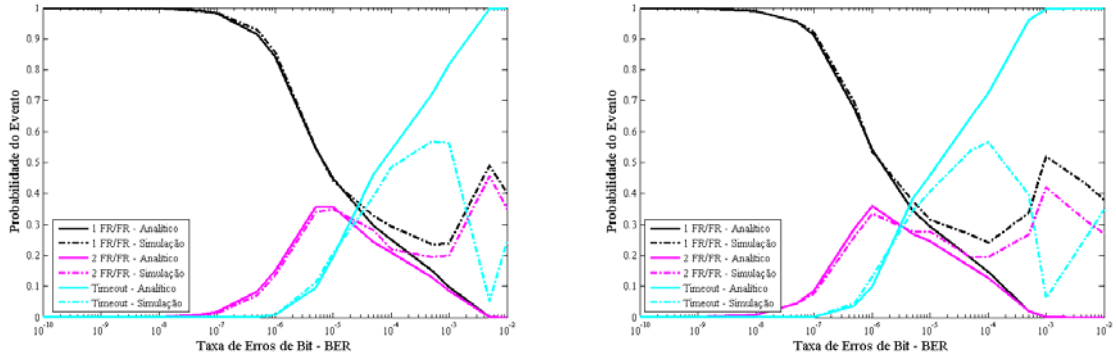
Analogamente, a probabilidade geral do evento de *timeout* pode ser calculada como

$$P_T = \begin{cases} P_{TW1} + \sum_{i=1}^{B} \left[\left(\prod_{j=0}^{i-1} P_{NW1} \right) \cdot P_{TW2} \right], \text{se } \beta' < 1 \\ P_{TW3} + \sum_{i=1}^{B-\theta} \left[\left(\prod_{j=0}^{i-1} P_{NW2} \right) \cdot P_{TW4} \right], \text{se } \beta' > 1 \end{cases} \quad (4.38)$$

onde P_{TW1} e P_{TW3} representam as probabilidades da ocorrência de *timeout* na rodada de janela inicial do ciclo, e são definidas por (4.35) com, respectivamente, $W = (\mu' \cdot T')$ e $b = B$, e $W = (W_{pipe}'/2)$ e $b = (B - \theta)$. Por sua vez, P_{TW2} e P_{TW4} representam as probabilidades da ocorrência de 2 *FR/FR* nas rodadas de janela subseqüentes à inicial, e são definidas por (4.35) com, respectivamente, $W = (\mu' \cdot T' + i)$ e $b = (B - i)$, $W = (W_{pipe}'/2 + i)$ e $b = (B - \theta - i)$.

4.3.1.2.8. Validação do cálculo das probabilidades dos eventos

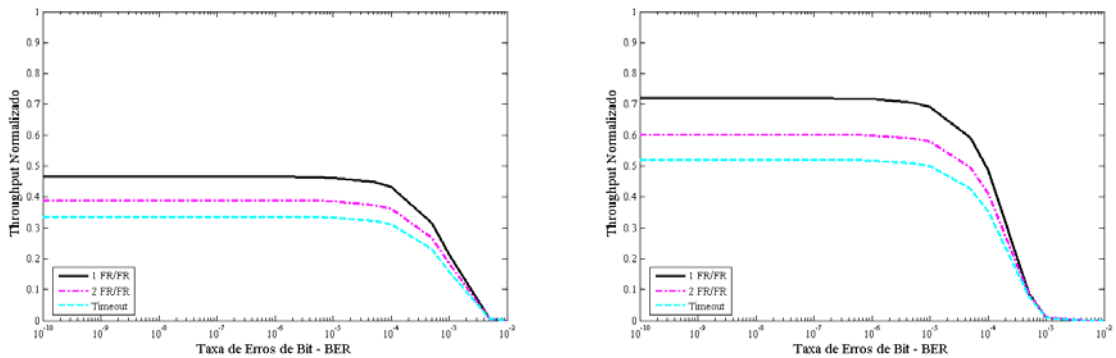
A Figura 4.39 ilustra as Probabilidades dos Eventos de 1 *FR*/*FR*, 2 *FR*/*FR* e *timeout* obtidas nas simulações mostradas na Seção 4.3.1.1.4, assim como as mesmas probabilidades quando analiticamente computadas através das equações (4.36), (4.37) e (4.38). Em ambos os casos, as PEs calculadas analiticamente se mostram boas aproximações das obtidas na simulação para a maior parte da faixa de BER considerada. No entanto, a partir de um determinado limite, aproximadamente $5 \cdot 10^{-4}$ para o código de bloco (856,820) e $1 \cdot 10^{-4}$ para o código de bloco (4408,4320), a diferença entre o modelo analítico e a simulação não é mais aceitável. Esse comportamento se deve ao fato de que para BERs muito altas a probabilidade de que cada ciclo realmente consiga atingir $\mu' \cdot T'$ (caso $\beta' < 1$) ou $W_{pipe}'/2$ (caso $\beta' > 1$) seja muito pequena. Desta forma o modelamento, que é baseado nessas suposições, fica comprometido.



(a) (b)

Figura 4.39 – Curvas das probabilidades associadas à ocorrência de cada evento quando o tamanho do bloco é (a) 856 bits e (b) 4408 bits.

Apesar desse problema, a Figura 4.38, que compara o desempenho do TCP obtido por simulação com o desempenho obtido analiticamente através do cálculo das PEs e do *throughput* médio de cada evento, mostra uma boa aproximação, tal qual a obtida na Seção 4.3.1.1.4, quando as PEs foram extraídas da simulação. Surge então o questionamento: como os resultados finais obtidos analiticamente podem ainda assim estar tão próximos dos obtidos por simulação? A resposta dessa questão pode ser obtida analisando a Figura 4.40, que mostra o *throughput* gerado na ocorrência de cada evento, obtido analiticamente em ambos os casos da Seção 4.3.1.1.4. É possível verificar que para os dois casos, na faixa de BER onde as PEs obtidas analiticamente não são boas aproximações das obtidas por simulação o *throughput* gerado por cada evento é aproximadamente o mesmo, fazendo com que os valores das probabilidades praticamente não tenham influência no resultado final do *throughput* do TCP.



(a) (b)

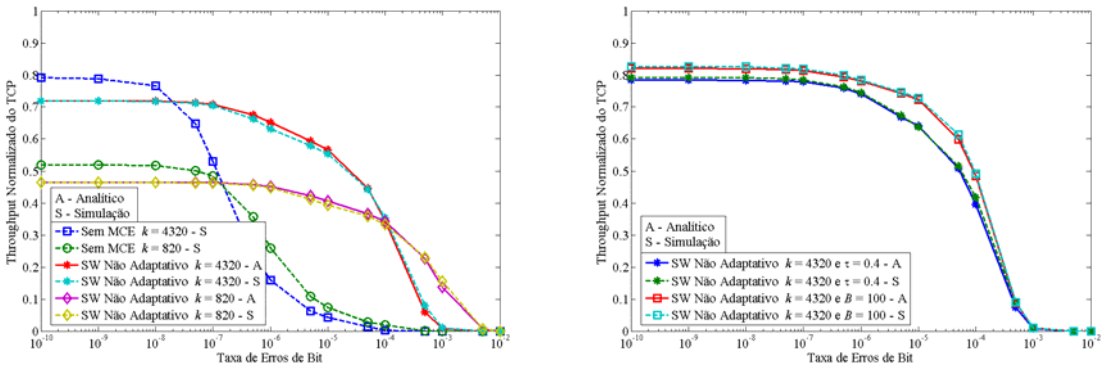
Figura 4.40 – *Throughput* médio gerado na ocorrência de cada evento quando o tamanho do bloco é (a) 856 bits e (b) 4408 bits.

4.3.1.3. Validação do modelo e resultados obtidos

Os resultados para o *SW* são mostrados utilizando-se a Rede C e a Rede S, ambas descritas na Seção 4.2.1. Novamente, para cada uma delas três cenários serão abordados: não adaptativo, adaptativo simples e adaptativo ideal.

4.3.1.3.1. Resultados obtidos para a Rede C

Para a Rede C, cenário não adaptativo são verificados os valores de k = 4320 e k = 820, cujas paridades necessárias são, respectivamente, 88 e 36 bits, calculadas novamente através dos limitantes apresentados na Seção 3.1.1.2. A Figura 4.41 mostra o resultado em ambos os casos, quando comparados ao caso onde nenhum MCE é utilizado, e mostra também o impacto da modificação dos valores de τ e B, quando k = 4320 bits.



(a) (b)

Figura 4.41 – *Throughput* normalizado do TCP com *SW* para o cenário não adaptativo quando (a) B = 20 e τ = 1 e (b) B e τ são alterados.

Diferentemente do resultado obtido quando da utilização do mecanismo *FEC*, com o mecanismo *SW* vê-se que a redução do tamanho do pacote gera redução no desempenho para a faixa de BERs entre 10^{-10} e 10^{-4}. Para BERs superiores a 10^{-4}, a redução no tamanho do pacote é benéfica ao desempenho do TCP. Este resultado já era esperado, visto que para pacotes maiores a probabilidade de um pacote chegar sem erro de bit ao transmissor é menor do que para pacotes de pequeno tamanho. Portanto, o maior *overhead* de cabeçalho no caso dos pacotes menores (4,39% no caso (856,820) contra 2,04% no caso (4408,4320)) é compensado pela menor probabilidade de retransmissão, fazendo com que para altas BERs pacotes de tamanhos reduzidos gerem melhores desempenhos do TCP. Com relação à variação dos parâmetros τ e B, o resultado qualitativo é o mesmo que o encontrado quando o

FEC é utilizado, ou seja, diminuir o atraso global τ, assim como aumentar o tamanho do *buffer B*, implica no aumento da eficiência do TCP.

A Tabela 4.2 mostra os resultados dos valores de k e $(n - k)$ para os cenários adaptativos. No caso do sistema adaptativo simples o número de bits de informação em cada pacote foi fixado em $k = 4320$ bits e a paridade calculada de acordo com a BER corrente. No caso do sistema adaptativo ideal, o modelo analítico foi utilizado para a execução do método de varredura em força bruta, onde todas as combinações possíveis de k e $(n - k)$ foram verificadas, e a combinação escolhida foi aquela que otimizou o desempenho do TCP para cada valor de BER. Vale lembrar que o valor mínimo do tamanho do pacote é $k = 321$ bits, dado que existem no mínimo $cab = 320$ bits referentes aos cabeçalhos TCP/IP, e ainda que seu tamanho máximo é dado por 524280 bits, valor máximo do datagrama IP.

Tabela 4.2 – Número de bits de informação e número de bits de paridade para o *SW* nos cenários adaptativos para a Rede C.

		BER								
		1.10^{-10}	1.10^{-9}	1.10^{-8}	1.10^{-7}	1.10^{-6}	1.10^{-5}	1.10^{-4}	1.10^{-3}	1.10^{-2}
SW Adaptativo Simples $k = 4320$	$(n - k)$	3	6	9	13	16	19	23	30	88
SW Adaptativo Ideal	k	294480	94112	30776	10936	6408	4672	1768	680	404
	$(n - k)$	9	10	12	14	16	19	21	24	30

O resultado para o *SW* nos cenários adaptativos da Figura 4.42, mostram uma ligeira melhora no caso adaptativo simples, com relação ao cenário não adaptativo. Já no cenário adaptativo ideal, uma melhora significativa pode ser vista principalmente nas BERs das extremidades. Isto ocorre devido à grande diferença no valor do tamanho do pacote quando comparado ao cenário referência não adaptativo para as BERs elevadas e para as BERs reduzidas. Nas BERs intermediárias, o valor do tamanho ideal do pacote se aproxima mais dos $k = 4320$ bits do cenário não adaptativo e por isso o ganho de eficiência do TCP não é muito significativo.

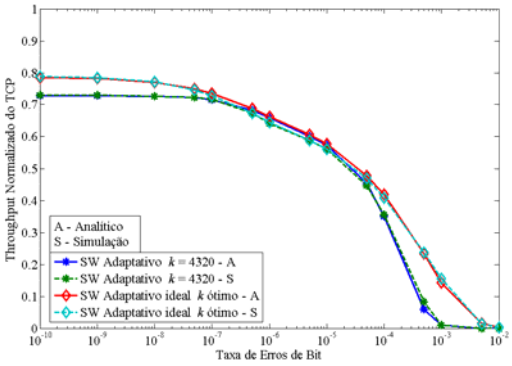


Figura 4.42 – *Throughput* normalizado do TCP com *SW* para a Rede C nos cenários adaptativo simples e adaptativo ideal.

4.3.1.3.2. Resultados obtidos para a Rede S

Para a Rede S, no cenário não adaptativo é usado o valor de k = 4320 bits e a paridade é novamente $(n - k)$ = 88 bits. A Figura 4.43 mostra o resultado para este cenário.

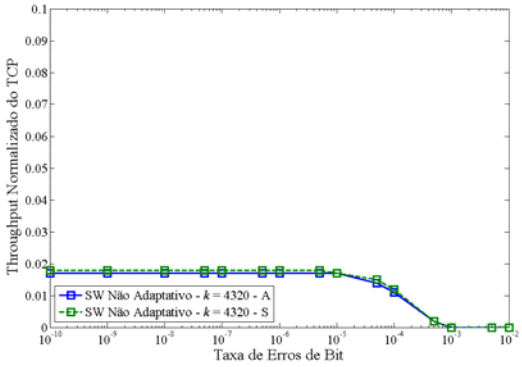


Figura 4.43 – *Throughput* normalizado do TCP com *SW* para a Rede S no cenário não adaptativo.

Como pode ser visto na Figura 4.43, quando o *SW* é utilizado no enlace satélite o desempenho do TCP é bastante reduzido, devido ao baixo aproveitamento do próprio *SW* nesse caso. Isto pode ser explicado pelo funcionamento do *SW*: um novo bloco só é transmitido quando a confirmação do bloco anterior é recebida. Como no enlace satélite o valor do tempo de propagação é muito elevado, o *SW* fica um longo tempo ocioso entre as transmissões de cada bloco, baixando significativamente seu desempenho. Os demais cenários não são analisados já que claramente o *SW* não é um MCE adequado para enlaces satélite.

4.3.2 – Modelamento do *GBN-ARQ*

Para visualizar o efeito da inserção do protocolo *GBN-ARQ*, a rede exemplo é novamente ilustrada, agora na Figura 4.44. Inicialmente tem-se a janela de congestionamento com tamanho de doze segmentos, no instante a EB recebe um NACK referente ao segmento 4, que foi corrompido durante a transmissão no enlace sem fio. Os segmentos 5, 6, 7 e 8 são transmitidos antes mesmo da chegada do NACK do segmento 4 à EB, que é uma característica do protocolo *GBN-ARQ*.

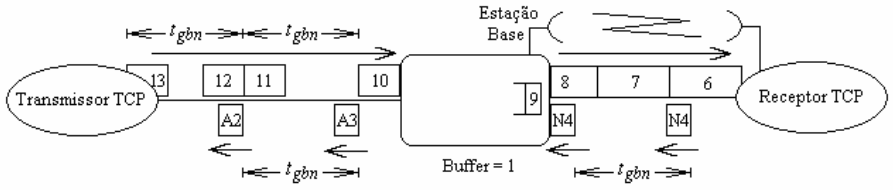


Figura 4.44 – Estado inicial da rede.

Quando o NACK do segmento 4 chega à EB, ela inicia a retransmissão deste segmento e de todos os segmentos posteriores, como mostra a Figura 4.45. O segmento 4 e todos os segmentos posteriores já transmitidos mas ainda não confirmados aguardam no *cache* de transmissão[3] e por esse motivo não são vistos no *buffer* principal.

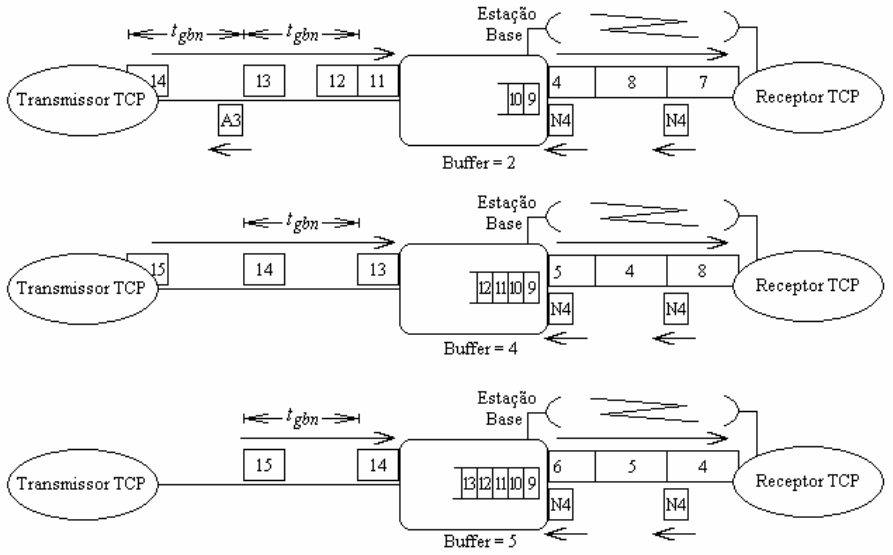


Figura 4.45 – Estados da rede após as retransmissões dos segmentos 3, 4 e 5 no enlace sem fio.

[3] Vale esclarecer que o *cache* de transmissão é necessário (principalmente no *GBN-ARQ*) para que o tamanho total dos *buffers* da camada de enlace (*buffer* principal + *cache* de transmissão) tenha o valor mínimo necessário para que o *GBN* não tenha sua eficiência comprometida, podendo ser corretamente calculada por (3.8).

A retransmissão do segmento 4 e de seus subseqüentes faz com que o espaçamento temporal entre os ACKs referentes aos segmentos 3 e 4 (ou o espaçamento temporal entre os segmentos 15 e 16, transmitidos com a chegada dos ACKs referentes aos segmentos 3 e 4) cresça de $1 \cdot t_{GBN}$ para $6 \cdot t_{GBN}$ e que a ocupação do *buffer* seja incrementada de 5 segmentos (na verdade o *buffer* cresce de 6 segmentos, no entanto um segmento se deve ao crescimento da janela com a chegada do segmento 12, e não às retransmissões), como mostra a Figura 4.46.

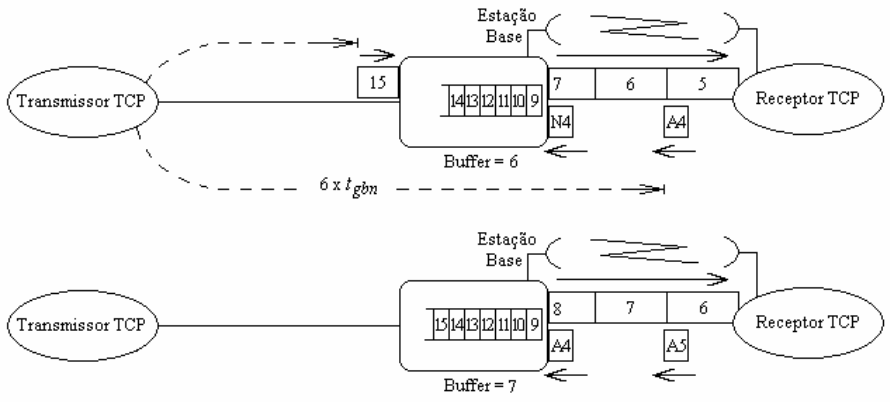


Figura 4.46 – Estados da rede que ilustram o espaçamento temporal $6 \cdot t_{GBN}$ criado entre o segmento 15 e o ACK referente ao segmento 4.

Após a chegada do segmento 15 à EB, esta fica um longo tempo sem receber novos segmentos, devido ao espaçamento temporal gerado pelas retransmissões. Neste intervalo são transmitidos cinco segmentos armazenados no *buffer* (os segmentos 9, 10, 11, 12 e 13), fazendo com que sua ocupação retome o tamanho de dois segmentos (um segmento já existente no instante da retransmissão e o outro devido à chegada do segmento 12, quando do crescimento da janela) no instante em que o segmento 16 chega à EB, como ilustrado na Figura 4.47.

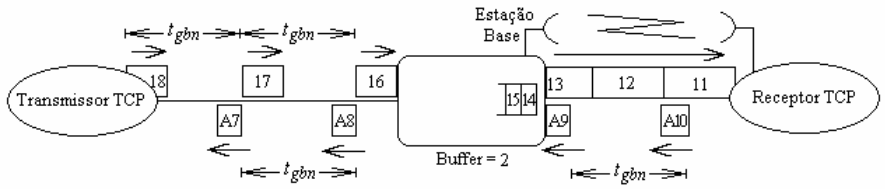


Figura 4.47 – Estado da rede que ilustra a redução da ocupação do *buffer*, após a ocorrência de uma retransmissão.

Como cada retransmissão gera um crescimento do *buffer* que é sempre maior do que a unidade, o cálculo das PEs precisa ser modificado. Além disso, a consideração de que a janela sempre alcança W_{pipe}', feita para o *SW-ARQ*, também se mostra inadequada, dado a maior

probabilidade de descartes no *buffer*. Faz-se necessário, portanto, uma análise um pouco mais complexa do que a vista para o *SW-ARQ*. De fato, os tamanhos iniciais da janela em cada ciclo, que como visto anteriormente são dependentes de β', são mantidos. No entanto, é considerado que o ciclo pode ser finalizado em uma rodada de janela anterior à W_{pipe}', o que de fato ocorre na prática.

4.3.2.1. Cálculo dos *throughputs* associados aos eventos

Diferentemente do modelo para o *SW-ARQ*, no modelo do *GBN-ARQ* é necessário que os *throughputs* associados a cada evento sejam função da rodada de janela $W \leq W_{pipe}$' alcançada em cada ciclo.

4.3.2.1.1. Cálculo do *throughput* associado ao evento de 1 *FR/FR* dada uma rodada de janela $W \leq W_{pipe}$'

No caso de $\beta' < 1$, a janela cresce de $\mu' \cdot T'$ até W na sub-fase *B*, um segmento é descartado e o TCP se recupera da perda através de ACKs duplicados. A janela é reduzida à $W/2$ e em seguida volta a crescer, passando pela sub-fase *A* e finalizando no início da sub-fase *B* quando a janela novamente tem $\mu' \cdot T'$ segmentos. Caso $\beta' > 1$, a evolução da janela de congestionamento dependerá do valor de $W/2$ em relação à $\mu' \cdot T'$. Caso $W/2 < \mu' \cdot T'$ a janela se inicia na sub-fase *B*, saindo de $W_{pipe}'/2$ e chegando a W, quando um segmento é descartado e o TCP se recupera da perda através de ACKs duplicados. A janela é reduzida à $W/2$ e o crescimento da janela entra na sub-fase *A*, indo de $W/2$ até $\mu' \cdot T'$. A janela entra novamente na sub-fase B, indo de $\mu' \cdot T'$ à $W_{pipe}'/2$, quando o ciclo finalmente se encerra. Caso $W/2 > \mu' \cdot T'$, extingue-se a sub-fase *A*, e depois da perda do pacote o crescimento entra diretamente na sub-fase *B*, indo de $W/2$ à $W_{pipe}'/2$ quando o ciclo se encerra.

Considerando a existência de um intervalo t_1, entre o fim da sub-fase *B* e a retomada do crescimento da janela, o *throughput* médio para o evento de somente um *FR/FR* pode ser calculado como:

$$\lambda_{1W} = \begin{cases} \dfrac{n_{B8} + n_{A5}}{t_{B8} + t_1 + t_{A5}}, \text{ caso } \beta' < 1 \\ \dfrac{n_{B9} + n_{A5} + n_{B10}}{t_{B9} + t_1 + t_{A5} + t_{B10}}, \text{ caso } \beta' > 1 \text{ e } W/2 < \mu'.T' \\ \dfrac{n_{B8} + n_{B11}}{t_{B8} + t_1 + t_{B11}}, \text{ caso contrário} \end{cases} \quad (4.39)$$

onde t_{B8} e n_{B8} são definidos pelas equações (4.8) e (4.9) com $W_{iB} = \mu' \cdot T'$ e $W_{fB} = W$; t_{A5} e n_{A5} são definidos por pelas equações (4.6) e (4.7) com $W_{iA} = W/2$ e $W_{fA} = \mu' \cdot T'$; t_{B9} e n_{B9} são definidos pelas equações (4.8) e (4.9) com $W_{iB} = W_{pipe}'/2$ e $W_{fB} = W_{pipe}'$; t_{B10} e n_{B10} são definidos pelas equações (4.8) e (4.9) com $W_{iB} = \mu' \cdot T'$ e $W_{fB} = W_{pipe}'/2$ e, finalmente, t_{B11} e n_{B11} são definidos pelas equações (4.8) e (4.9) com $W_{iB} = W/2$ e $W_{fB} = W_{pipe}'/2$. O valor de t_1 é definido em (4.22).

4.3.2.1.2. Cálculo do *throughput* associado ao evento de 2 *FR/FR* dada uma rodada de janela $W \leq W_{pipe}'$

No caso de $\beta' < 1$, a janela cresce de $\mu' \cdot T'$ até W na sub-fase *B*, dois segmentos são descartados e o TCP se recupera da perda através de ACKs duplicados. A janela é reduzida à $W/4$ e em seguida volta a crescer, passando pela sub-fase *A* e finalizando no início da sub-fase *B* quando a janela novamente tem $\mu' \cdot T'$ segmentos. Caso $\beta' > 1$, a evolução da janela de congestionamento dependerá do valor de $W/4$ em relação à $\mu' \cdot T'$. Caso $W/4 < \mu' \cdot T'$ a janela se inicia na sub-fase *B*, saindo de $W_{pipe}'/2$ e chegando a W, quando um segmento é descartado e o TCP se recupera da perda através de ACKs duplicados. A janela é reduzida à $W/4$ e o crescimento da janela entra na sub-fase *A*, indo de $W/4$ até $\mu' \cdot T'$. A janela entra novamente na sub-fase B, indo de $\mu' \cdot T'$ à $W_{pipe}'/2$, quando o ciclo finalmente se encerra. Caso $W/4 > \mu' \cdot T'$, extingue-se a sub-fase *A*, e depois da perda do pacote o crescimento entra diretamente na sub-fase *B*, indo de $W/4$ à $W_{pipe}'/2$ quando o ciclo se encerra.

Considerando a existência de um intervalo t_2, entre o fim da sub-fase *B* e a retomada do crescimento da janela, o *throughput* médio para o evento de somente um *FR/FR* pode ser calculado como:

$$\lambda_{2W} = \begin{cases} \dfrac{n_{B8} + n_2 + n_{A6}}{t_{B8} + t_2 + t_{A6}}, \text{ caso } \beta' < 1 \\ \dfrac{n_{B9} + n_2 + n_{A6} + n_{B10}}{t_{B9} + t_2 + t_{A6} + t_{B10}}, \text{ caso } \beta' > 1 \text{ e } W/4 < \mu'.T' \\ \dfrac{n_{B9} + n_2 + n_{B12}}{t_{B9} + t_2 + t_{B12}}, \text{ caso contrário} \end{cases} \quad (4.40)$$

onde t_{A6} e n_{A6} são definidos pelas equações (4.6) e (4.7) com $W_{iA} = W/4$ e $W_{fA} = \mu' \cdot T'$; t_{B10} e n_{B9} são definidos pelas equações (4.8) e (4.9) com $W_{iB} = W_{pipe}'/2$ e $W_{fB} = W_{pipe}'$ e, finalmente, t_{B12} e n_{B12} são definidos pelas equações (4.8) e (4.9) com $W_{iB} = W/4$ e $W_{fB} = W_{pipe}'/2$. Os valores de t_2 e n_2 são definidos em (4.24) e (4.25), respectivamente.

4.3.2.1.3. Cálculo do *throughput* associado ao evento de *timeout* dada uma rodada de janela $W \leq W_{pipe}'$

Caso $\beta' < 1$, a janela cresce a partir de $\mu' \cdot T'$ passando pela sub-fase *B*, três ou mais segmentos são descartados e o TCP se recupera através de ACKs duplicados nas duas primeiras perdas, mas um *timeout* ocorre para o terceiro segmento perdido. A janela é reduzida à unidade, cresce exponencialmente até $W_{pipe}'/8$, caracterizando a fase de *Slow Start*, quando finalmente entra na sub-fase *A*, crescendo até $\mu'.T'$. Caso $\beta' > 1$, a evolução da janela de congestionamento dependerá do valor de $W/8$ em relação à $\mu' \cdot T'$. Caso $W/8 < \mu' \cdot T'$, a janela se inicia em $W_{pipe}'/2$ e cresce na sub-fase *B* até o valor máximo *W*, quando as três ou mais perdas levam ao *timeout*. Após a fase de *Slow Start*, que leva a janela da unidade até $W/8$, o crescimento da janela entra na sub-fase *A* até atingir $\mu'.T'$. Em seguida, o crescimento da janela entra na sub-fase *B*, indo até $W_{pipe}'/2$, quando o ciclo se encerra. Caso $W/8 > \mu' \cdot T'$, a sub-fase *A* se extingue e após o *Slow Start*, o crescimento entra diretamente na sub-fase *B*, indo de $W/8$ até $W_{pipe}'/2$.

$$\lambda_{TW} = \begin{cases} \dfrac{n_{B8} + n_T + n_{SS3} + n_{A7}}{t_{B8} + t_T + t_{SS3} + t_{A7}}, \text{ caso } \beta' < 1 \\ \dfrac{n_{B9} + n_T + n_{SS3} + n_{A7} + n_{B10}}{t_{B9} + t_T + t_{SS3} + t_{A7} + t_{B10}}, \text{ caso } \beta' < 1 \text{ e } W/8 < \mu'.T' \\ \dfrac{n_{B9} + n_T + n_{SS3} + n_{B13}}{t_{B9} + t_T + t_{SS3} + t_{B13}}, \text{ caso contrário} \end{cases} \quad (4.41)$$

onde t_{A7} e n_{A7} são definidos por pelas equações (4.6) e (4.7) com $W_{iA} = W/8$ e $W_{fA} = \mu' \cdot T'$; t_{SS3} e n_{SS3} são definidos pelas equações (4.3) e (4.4) com $W_{fSS} = W_{pipe}'/8$ e,

finalmente, t_{B13} e n_{B13} são definidos pelas equações (4.8) e (4.9) com $W_{iB} = W/8$ e $W_{fB} = W_{pipe}'/2$. Os valores de t_T e n_T são definidos em (4.27) e (4.28), respectivamente.

4.3.2.2. Cálculo das probabilidades dos eventos

Nesta seção são calculadas as probabilidades dos eventos de 1 *FR/FR*, 2 *FR/FR* e *timeout*. Para isso é necessário entender que no caso do *GBN-ARQ*, quando o *bit pipe* está cheio, a ocorrência de cada retransmissão faz com que o *buffer* cresça de aproximadamente $\sigma = [(t_{segw} + \tau_b)/t_{segw}]$ segmentos, onde $t_{segw} = n/(\mu \cdot k)$ é o tempo de transmissão do bloco no enlace sem fio. Claramente, $\sigma > 1$.

4.3.2.2.1. Cálculo da probabilidade do evento de nenhum descarte no *buffer* dada a rodada de janela $W \leq W_{pipe}'$

O cálculo da probabilidade do evento de nenhum descarte segue as mesmas premissas do cálculo no *SW-ARQ*. No entanto, o fato de cada retransmissão gerar um crescimento da ocupação do *buffer* maior do que a unidade deve ser levado em conta através do parâmetro σ. A probabilidade de nenhum descarte é alterada, portanto para

$$P_{NW} = \begin{cases} P_{nu4}, \text{ se } (\beta' < 1 \text{ e } \mu'.T' \leq W < W_{pipe}') \text{ ou } (\beta' > 1 \text{ e } W_{pipe}'/2 \leq W < W_{pipe}') \\ 0, \text{caso contrário} \end{cases} \quad (4.42)$$

onde P_{nu4} é definido por (4.29) com $J = W$ e $LT = E + \lfloor (b-1)/\sigma \rfloor$.

4.3.2.2.2. Cálculo da probabilidade do evento de 1 *FR/FR* dada a rodada de janela $W \leq W_{pipe}'$

O cálculo da probabilidade do evento de 1 *FR/FR* também segue as premissas do *SW-ARQ*. Seu valor pode ser escrito como

$$P_{1W} = \begin{cases} P_{i3} \cdot P_{nu5}, \text{ se } S_1 > S_2 \\ 0, \text{ caso contrário} \end{cases} \quad (4.43)$$

onde P_{i3} é definido por (4.32) com $J = W$ e $NT = S_1 = E + \lfloor b/\sigma \rfloor$. O valor de S_2 é igual à $E + \lfloor (b-1)/\sigma \rfloor$. Note que a diferença para o cálculo no *SW-ARQ*, além do parâmetro σ, se dá com $S_1 = S_2$, pois nesse caso P_{i3} é um subcaso de P_{nu4} e, portanto não haverá descarte na primeira rodada de janela, o que leva ao evento da seção anterior (já estando computado em (4.42)).

4.3.2.2.3. Cálculo da probabilidade do evento de 2 *FR/FR* dada a rodada de janela $W \leq W_{pipe}'$

Novamente, as premissas do cálculo no *SW-ARQ* são seguidas, no entanto algumas restrições devem ser observadas. O cálculo da probabilidade do evento de 2 *FR/FR* numa rodada de janela *W* pode ser escrita como

$$P_{2W} = \begin{cases} P_{i3} \cdot P_{i4} + P_{i4} \cdot P_{nu6}, \text{se} \left(S_1 > S_2\right) \text{e} \left(S_3 > S_1\right) \\ P_{i3} \cdot P_{i4}, \text{ se} \left(S_1 > S_2\right) \text{e} \left(S_3 = S_1\right) \\ P_{e4} \cdot P_{nu6}, \text{ se} \left(S_1 = S_2\right) \text{e} \left(S_3 > S_1\right) \end{cases} \tag{4.44}$$

onde P_{i4} é definido por (4.32) com $J = W$ e $NT = S_3 = E + \lfloor (b + 1)/\sigma \rfloor$. Da mesma forma, P_{nu6} é definido por (4.29) com $J = W/2$ e $LT = \text{E} + \lfloor (b + 1)/(2\sigma) \rfloor$.

Aqui existem duas restrições: a primeira é exatamente à do caso anterior, quando $S_1 = S_2$, e, portanto, P_{i3} é um subcaso de P_{nu4}, não havendo descarte na primeira rodada de janela; a segunda restrição se dá quando $S_3 = S_1$, e, portanto, $P_{i4} = P_{i3}$, sendo que neste caso, ao invés de dois descartes na primeira rodada de janela, apenas um ocorrerá (a probabilidade deste evento já é levada em conta em (4.43)).

4.3.2.2.4. Cálculo da probabilidade do evento de *timeout* dada a rodada de janela $W \leq W_{pipe}$'

A probabilidade de *timeout*, dada a rodada de janela *W* é computada através de

$$\text{P}_{T\,W} = 1 - P_{1\,W} - P_{2\,W} - P_{N\,W}. \tag{4.45}$$

4.3.2.3. Cálculo dos produtos $P_1 \cdot \lambda_1$, $P_2 \cdot \lambda_2$ e $P_T \cdot \lambda_T$

Os valores desses produtos $P_1 \cdot \lambda_1$, $P_2 \cdot \lambda_2$ e $P_T \cdot \lambda_T$ são dados, respectivamente, por

$$P_1 \cdot \lambda_1 = \begin{cases} P_{1W1} \cdot \lambda_{1W1} + \sum_{i=1}^{B} \left[\left(\prod_{j=0}^{i-1} P_{NW1} \right) \cdot P_{1W2} \cdot \lambda_{1W2} \right], \text{se } \beta' < 1 \\ P_{1W3} \cdot \lambda_{1W3} + \sum_{i=1}^{B-\theta} \left[\left(\prod_{j=0}^{i-1} P_{NW2} \right) \cdot P_{1W4} \cdot \lambda_{1W4} \right], \text{se } \beta' > 1 \end{cases} \tag{4.46}$$

$$P_2 \cdot \lambda_2 = \begin{cases} P_{2W1} \cdot \lambda_{2W1} + \sum_{i=1}^{B} \left[\left(\prod_{j=0}^{i-1} P_{NW1} \right) \cdot P_{2W2} \cdot \lambda_{2W2} \right], \text{se } \beta' < 1 \\ P_{2W3} \cdot \lambda_{2W3} + \sum_{i=1}^{B-\theta} \left[\left(\prod_{j=0}^{i-1} P_{NW2} \right) \cdot P_{2W4} \cdot \lambda_{2W4} \right], \text{se } \beta' > 1 \end{cases} \tag{4.47}$$

$$P_T \cdot \lambda_T = \begin{cases} P_{TW1} \cdot \lambda_{TW1} + \sum_{i=1}^{B} \left[\left(\prod_{j=0}^{i-1} P_{NW1} \right) \cdot P_{TW2} \cdot \lambda_{TW2} \right], \text{se } \beta' < 1 \\ P_{TW3} \cdot \lambda_{TW3} + \sum_{i=1}^{B-\theta} \left[\left(\prod_{j=0}^{i-1} P_{NW2} \right) \cdot P_{TW4} \cdot \lambda_{TW4} \right], \text{se } \beta' > 1 \end{cases} \tag{4.48}$$

onde P_{1Wi}, λ_{1Wi}, P_{2Wi}, λ_{2Wi}, P_{TWi}, λ_{TWi} e P_{NWi} (para todos os i) são dados, respectivamente, por (4.43), (4.39), (4.44), (4.40), (4.45), (4.41) e (4.42) com: $W = \mu' T'$ e $b = B$, para P_{1W1}, λ_{1W1}, P_{2W1}, λ_{2W1}, P_{TW1} e λ_{TW1}; $W = \mu' T' + i$ e $b = B - i$, para P_{1W2}, λ_{1W2}, P_{2W2}, λ_{2W2}, P_{TW2} e λ_{TW2}; $W = \mu' T' + j$ e $b = B - j$, para P_{NW1}; $W = W_{pipe}'/2$ e $b = B - \theta$, para P_{1W3}, λ_{1W3}, P_{2W3}, λ_{2W3}, P_{TW3} e λ_{TW3}; $W = W_{pipe}'/2 + i$ e $b = B - \theta - i$, para P_{1W4}, λ_{1W4}, P_{2W4}, λ_{2W4}, P_{TW4} e λ_{TW4} e finalmente, $W = W_{pipe}'/2 + j$ e $b = B - \theta - j$, para P_{NW2}. O parâmetro $\theta = \lfloor W_{pipe}'/2 - \mu' T' \rfloor$ representa a ocupação inicial do *buffer* no caso de $\beta' > 1$.

4.3.2.4. Validação do modelo e resultados obtidos

Os resultados para o *GBN* são mostrados utilizando-se a Rede C e a Rede S, ambas descritas na Seção 4.2.1. Novamente, para cada uma delas três cenários são abordados: não adaptativo, adaptativo simples e adaptativo ideal.

4.3.2.4.1. Resultados obtidos para a Rede C

Os resultados para o sistema com *GBN* são qualitativamente os mesmos encontrados na Seção 4.3.1.3.1. No cenário não adaptativo, mostrado pela Figura 4.48, é possível constatar que o desempenho do TCP apresenta melhor desempenho para BERs reduzidas quando o tamanho do bloco é elevado. Já para BERs superiores, o TCP tem melhor desempenho quando o tamanho dos blocos é reduzido. Com relação aos valores de τ e B, o resultado é o mesmo visto no *SW*, ou seja, a redução no atraso global τ assim como o aumento do tamanho do *buffer* B leva o TCP a um desempenho superior.

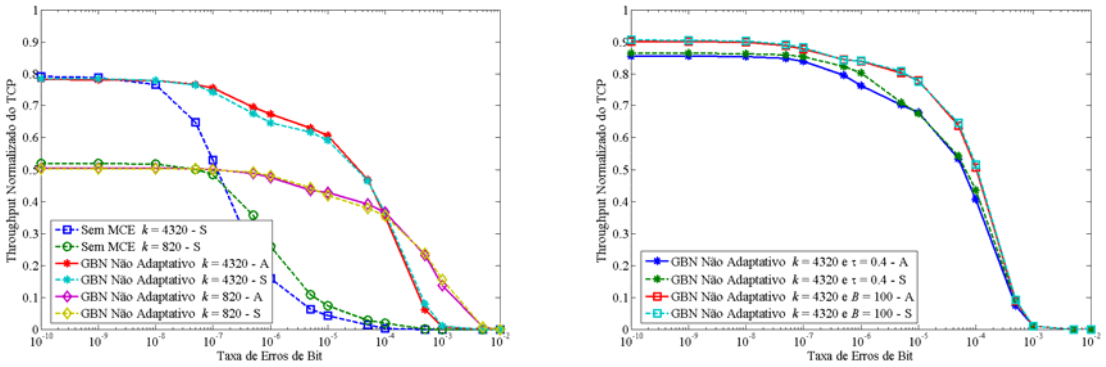


Figura 4.48– *Throughput* normalizado do TCP com *GBN* para o cenário não adaptativo.

A Tabela 4.3 mostra os resultados dos valores de k e $(n - k)$ para os cenários adaptativos. No caso do sistema adaptativo simples, assim como no *SW*, o número de bits de informação no

bloco foi fixado em k = 4320 bits e a redundância calculada de acordo com a BER corrente. No caso do sistema adaptativo ideal, assim como no *SW*, o modelo analítico foi utilizado para a execução do método de varredura em força bruta, onde todas as combinações possíveis de k e $(n - k)$ foram verificadas, e a combinação escolhida foi aquela que otimizou o desempenho do TCP para cada valor de BER.

Tabela 4.3 – Número de bits de informação e número de bits de paridade para o *GBN* nos cenários adaptativos para a Rede C.

		BER								
		1.10^{-10}	1.10^{-9}	1.10^{-8}	1.10^{-7}	1.10^{-6}	1.10^{-5}	1.10^{-4}	1.10^{-3}	1.10^{-2}
GBN Adaptativo Simples $k = 4320$	$(n - k)$	3	6	9	13	16	19	23	30	88
GBN Adaptativo Ideal	k	191955	61536	20246	7449	6460	4386	1735	691	404
	$(n - k)$	8	10	11	13	16	19	21	23	27

A Figura 4.49 mostra o resultado para os cenários adaptativos. Novamente o resultado para o cenário adaptativo simples mostra-se apenas levemente superior ao resultado obtido com o cenário não adaptativo. No entanto, assim como no *SW*, uma melhora significativa pode ser vista no caso do sistema adaptativo ideal, principalmente nas faixas extremas de BERs.

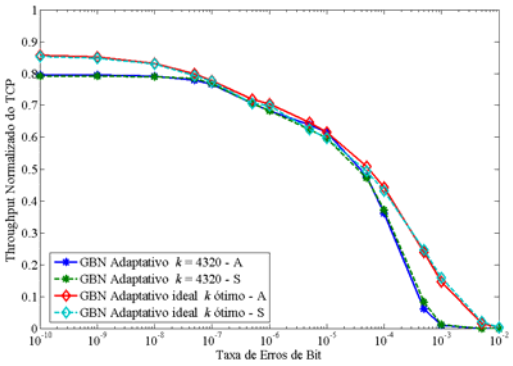


Figura 4.49 – *Throughput* normalizado do TCP com *SW* para os cenários adaptativo simples e adaptativo ideal.

4.3.2.4.2. Resultados obtidos para a Rede S

Novamente os três cenários são analisados. No cenário não adaptativo, o número de bits de informação em cada bloco é k = 4320 e o número de bits de paridade é $n - k$ = 88. Para o cenário adaptativo simples, o valor do número de bits de informação em cada bloco também é de k = 4320, no entanto, o número de bits de paridade é mostrado na Tabela 4.4. No caso do

sistema adaptativo ideal, assim como para a Rede C, o modelo analítico foi utilizado para a execução do método de varredura em força bruta, onde todas as combinações possíveis de k e $(n - k)$ foram verificadas, e a combinação escolhida foi aquela que otimizou o desempenho do TCP para cada valor de BER. Os valores de k e $n - k$ para este cenário também estão mostrados na Tabela 4.4.

Tabela 4.4 – Número de bits de informação e número de bits de paridade para o *GBN* nos cenários adaptativos para a Rede S.

		BER								
		1.10^{-10}	1.10^{-9}	1.10^{-8}	1.10^{-7}	1.10^{-6}	1.10^{-5}	1.10^{-4}	1.10^{-3}	1.10^{-2}
GBN Adaptativo Simples $k = 4320$	$(n - k)$	3	6	9	13	16	19	23	30	88
GBN Adaptativo Ideal	k	33821	11173	4186	2733	1623	1333	807	580	401
	$(n - k)$	6	7	9	13	15	18	20	24	31

A Figura 4.50 mostra os resultados de *throughput* normalizado do TCP para os três cenários propostos, quando a Rede S é utilizada. No cenário não adaptativo, percebe-se que o desempenho do TCP na Rede S para a BER mais baixa da faixa verificada (10^{-10}) é apenas um pouco inferior quando comparado ao da Rede C com $B = 100$ no mesmo cenário. Como para essa BER o número de retransmissões é desprezível, a diferença de desempenho é associada à diferença no atraso global τ da Rede C ($\tau = 1$s) e da Rede S ($\tau = 1.5$s). No entanto, com o aumento da BER e o conseqüente aumento no número de retransmissões, a diferença de desempenho do TCP nas Redes C e S aumenta significamente. Isto se deve ao elevado valor do atraso de propagação no enlace satélite, que torna o processo de retransmissão muito oneroso ao desempenho do TCP.

Com relação ao cenário adaptativo simples, este apresenta apenas uma pequena melhora no desempenho do TCP com relação ao cenário não adaptativo. Já no cenário adaptativo ideal, apresenta uma melhora um pouco mais significativa, principalmente nas BERs extremas.

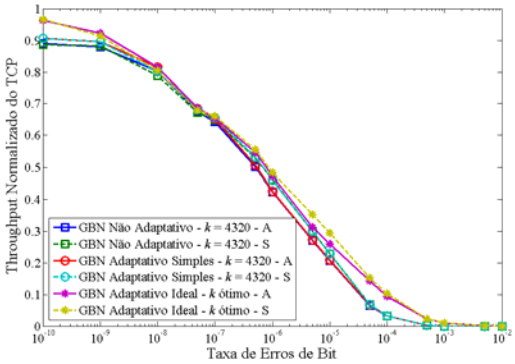


Figura 4.50 – *Throughput* normalizado do TCP com *GBN* nos cenários não adaptativo, adaptativo simples e adaptativo ideal para a Rede S.

4.4. Conclusões sobre os resultados obtidos

A Figura 4.51 mostra um resumo dos resultados obtidos para a Rede C no cenário não adaptativo. A primeira conclusão que se pode tirar desta figura é a confirmação da não adequação do TCP *standalone* nas redes sem fio com taxas elevadas de BER, visto seu desempenho. Este baixo rendimento ocorre devido aos fatores mostrados no Capítulo 3.

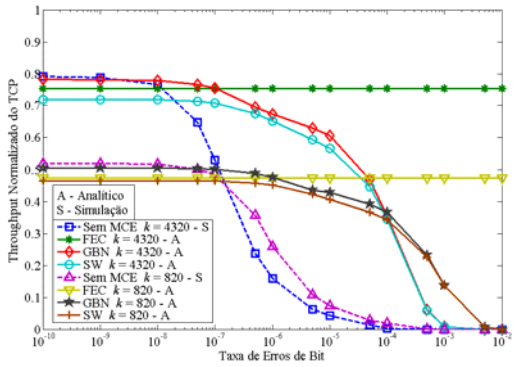
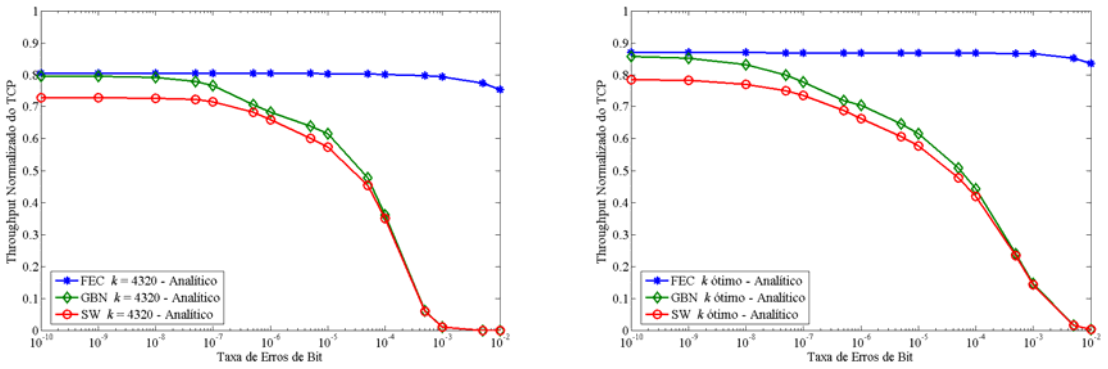


Figura 4.51 – *Throughputs* normalizados do TCP com os MCEs no cenário não adaptativo para a Rede C.

Outra conclusão que pode ser tirada é que o desempenho do TCP com a utilização dos MCEs no cenário não adaptativo é consideravelmente melhorado com relação ao caso onde o TCP opera sozinho. Neste caso, o TCP com *GBN* apresenta o melhor desempenho para taxas de erro de bit reduzidas, independentemente do tamanho do segmento TCP. No entanto, para taxas de erros de bit superiores a valores próximos a 10^{-7} ou 10^{-6} (dependendo do valor de k) o TCP com *FEC* passa a apresentar o melhor desempenho. Dentre as MCEs propostas o TCP com *SW* apresenta o pior desempenho em toda a faixa de taxa de erros de bit considerada, no

entanto, pode ser uma escolha interessante se levado em conta sua menor complexidade de implementação prática. Todos estes resultados estão qualitativamente de acordo com [15].

A Figura 4.52 mostra um resumo dos resultados obtidos quando são utilizadas MCEs adaptativas no enlace sem fio da Rede C. Uma conclusão que pode ser tirada desta Figura é que o TCP com *FEC* apresenta o melhor desempenho em toda faixa de BERs consideradas, seja quando o cenário é o adaptativo simples ou quando o cenário é o adaptativo ideal. Esta conclusão também está de acordo com [15]. Vale enfatizar que foi considerado que a estimativa de erros de bit é sempre possível e que não há *overhead* algum neste processo. Na prática, a estimativa da taxa de erros de bit pode ser de difícil implementação.



(a) (b)

Figura 4.52 - *Throughputs* normalizados do TCP com os MCEs nos cenários (a) adaptativo simples e (b) adaptativo ideal para a Rede C.

A Figura 4.53 resume os resultados obtidos para a Rede S no cenário não adaptativo quando k = 4320 bits. Como dito anteriormente, para enlaces satélite o *SW* claramente não é uma escolha adequada. Outra conclusão que pode ser tirada é que neste caso o cruzamento das curvas de desempenho para o TCP com *FEC* e para o TCP com *GBN* ocorre para uma BER inferior se comparado com a Rede C. Portanto, o *FEC* passa a ser uma escolha mais interessante entre ambos, visto que a faixa em que o desempenho do TCP com *GBN* é superior ao desempenho do TCP com *FEC* é muito reduzida.

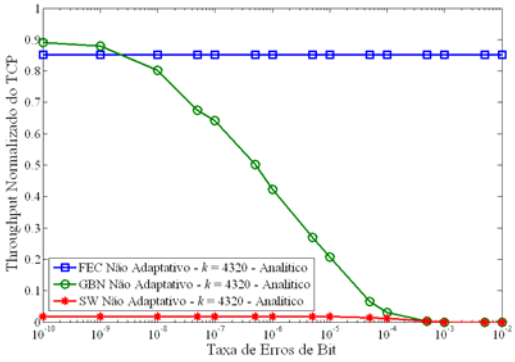


Figura 4.53 – *Throughputs* normalizados do TCP com os MCEs no cenário não adaptativo para a Rede S.

A Figura 4.54 mostra o resumo dos resultados obtidos para a Rede S nos cenários adaptativos. Assim como para a Rede C, o desempenho do TCP com *FEC* é sempre superior ao desempenho do TCP com *GBN*, independente do cenário adaptativo simples ou adaptativo ideal. Novamente, o *FEC* se mostra a alternativa mais interessante.

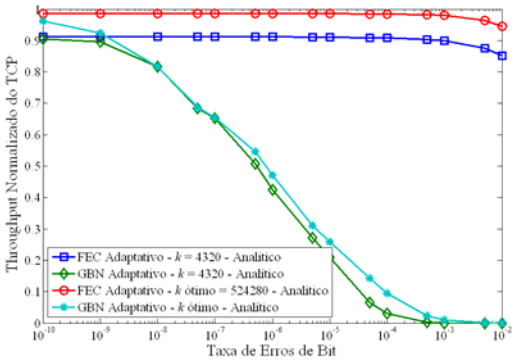


Figura 4.54 – *Throughputs* normalizados do TCP com os MCEs no cenário adaptativo para a Rede S.

Finalmente, com relação à validade dos modelos, pode-se concluir que o modelo com a primeira modificação proposta na Seção 4.2 se adequa bem quando o *FEC* é utilizado na camada de enlace sem fio, gerando resultados analíticos bem próximos aos obtidos através da simulação, validando-o, por conseguinte. No entanto, quando os protocolos *SW* ou *GBN* são utilizados, essa primeira modificação é inadequada, sendo necessárias as modificações propostas na Seção 4.3. Com estas modificações, o modelo analítico proposto também gera resultados bem próximos aos obtidos por simulação, validando-o.

5. Conclusões

Alguns dos principais problemas associados à utilização do TCP nas redes sem fio são as altas taxas de erros de bit, as perdas intermitentes de sinal entre as extremidades do enlace sem fio, a assimetria das taxas de transmissão dos sentidos *downlink* e *uplink* no enlace sem fio e, finalmente, os longos atrasos de propagação vistos em alguns enlaces sem fio, como os enlaces satélite.

No caso das altas taxas de erros de bit, foram apresentadas duas soluções encontradas na literatura para melhorar o desempenho do TCP. A primeira, chamada protocolo *Snoop*, melhora o desempenho do TCP, no sentido da transmissão *Host* Fixo para *Host* Móvel, através de retransmissões locais no enlace sem fio. No caso da transmissão no sentido reverso, o agente *Snoop* faz uso do ELN para indicar ao transmissor TCP que não invoque os mecanismos de controle de congestionamento quando segmentos tiverem sido descartados devido a erros de bit. A segunda proposta para melhora do TCP é a utilização de protocolos de controle de erro no enlace sem fio, seja através de retransmissões (*ARQ*), através da correção automática no receptor (*FEC*), ou ainda através da união desses dois mecanismos (Híbridos).

No caso das perdas intermitentes de sinal entre as extremidades do enlace sem fio, novamente duas soluções foram apresentadas. A primeira, chamada *Freeze*-TCP e aplicada somente quando a transmissão de dados se dá no sentido *Host* Fixo para *Host* Móvel, busca antecipar-se à ocorrência de perda de sinal solicitando ao transmissor que interrompa a transmissão de dados para que segmentos não sejam perdidos. Diferentemente do *Freeze*-TCP, a segunda solução apresentada, o ATCP, busca melhorar o funcionamento do TCP na transmissão de dados em ambos os sentidos. No caso da transmissão no sentido *Host* Móvel para *Host* Fixo, o ATCP altera o funcionamento dos mecanismos de controle de congestionamento do TCP de forma que após a ocorrência de uma perda de sinal entre *Host* Móvel e Estação Base, a taxa de transmissão do TCP retorne rapidamente àquela anterior à ocorrência da perda de sinal. No sentido de transmissão do *Host* Fixo para *Host* Móvel, o receptor TCP busca retardar o envio de alguns ACKs ao transmissor, de forma que após a ocorrência de uma perda de sinal, o receptor possa imediatamente enviar estes ACKs e, por conseguinte, o transmissor TCP possa retomar a transmissão dos segmentos.

A assimetria de largura de banda entre os sentidos *downlink* e *uplink* nos enlaces sem fio foi mais uma característica prejudicial ao desempenho do TCP apresentada. Esta característica de algumas redes sem fio atrapalha a realimentação do transmissor com os ACKs do receptor, degradando a temporização do envio de novos segmentos. O mecanismo de Filtragem de ACKs é utilizado neste caso para reduzir a freqüência dos ACKs enviados pelo receptor. Em conjunto com a técnica de Reconstrução de ACKs, que reconstrói da melhor forma possível a cadeia de ACKs originalmente enviada pelo receptor TCP, ou ainda em conjunto com uma pequena modificação no transmissor TCP, a chamada Adaptação do Transmissor, que tem como objetivo mudar a forma com que o TCP trata a chegada intencionalmente reduzida (pela Filtragem de ACKs) dos ACKs, o mecanismo de Filtragem de ACKs melhora o desempenho do TCP.

A última característica presente em algumas redes sem fio, mais precisamente nas redes com enlace satélite, são os longos atrasos de propagação destes enlaces. Esta característica é prejudicial ao TCP principalmente nas transmissões de pequenas quantidades de dados, visto que a taxa de transmissão inicial do TCP é sempre baixa (é limitada pelo reduzido tamanho inicial da janela de congestionamento). Para amenizar esse problema, quatro soluções são apresentadas. A primeira e mais simples solução mostrada é o aumento do tamanho da janela de congestionamento inicial do TCP, que acaba aumentando a taxa de envio de segmentos inicial. A segunda solução mostrada consiste na utilização de protocolos *Split*-TCP, que dividem a conexão TCP fim-a-fim em conexões menores, fazendo com que o impacto dos atrasos de propagação do enlace satélite seja "escondido" do transmissor TCP. A terceira solução é a utilização da técnica de TCP *Spoofing*, que apesar de não dividir a conexão TCP também busca "esconder" o enlace de maior atraso de propagação do transmissor TCP, através da utilização de *Spoofing* ACKs. A última solução apresentada, o TCP-*Peach*, consiste em modificações nos mecanismos de controle de congestionamento que buscam "inflar" de forma rápida a janela de congestionamento do transmissor TCP, aumentando sua taxa de transmissão. O TCP-*Peach* usa segmentos de dados de baixa prioridade (segmentos *dummy*) e ACKs de baixa prioridade (ACKs *dummy*) para verificar a disponibilidade da rede para poder efetuar o aumento na taxa de transmissão.

Com relação aos resultados obtidos através dos modelos analíticos desenvolvidos para o cálculo do desempenho do TCP quando os protocolos de controle de erro *FEC*, *SW-ARQ* e *GBN-ARQ* são utilizados no enlace sem fio, algumas conclusões podem ser tiradas. A

primeira delas é, na verdade, a confirmação de que o desempenho do TCP é muito reduzido quando nenhuma modificação é efetuada na sua estrutura de transmissão, ou seja, quando o TCP opera *standalone*. Os gráficos de resultados do Capítulo 4 mostram que para taxas de erros de bit ainda relativamente baixas (10^{-8} ou 10^{-7}) o TCP já opera de forma muito deficiente. De forma geral, no primeiro cenário proposto, o cenário não-adaptativo, onde nenhuma entidade da rede é capaz de estimar a taxa de erro de bit do enlace sem fio, conclui-se que o TCP apresenta o melhor desempenho para taxas de erros baixas quando o *GBN-ARQ* é utilizado no enlace sem fio. A partir de um determinado limite (imposto pelas características da rede e da transmissão TCP tais como tamanho dos segmentos TCP, atrasos de propagação no enlace sem fio, tamanho dos *buffers*, entre outros) o TCP passa a apresentar melhor desempenho operando com *FEC* no enlace sem fio. Independentemente da rede em uso, quando o *SW-ARQ* é utilizado no enlace sem fio, o TCP sempre apresenta o pior desempenho. Quando o cenário é adaptativo, onde é possível estimar a taxa de erros de bit do canal sem fio, seja ele o cenário adaptativo simples, onde essa estimativa é utilizada apenas para alterar o número de bits de paridade dos blocos transmitidos no enlace sem fio, ou ainda o cenário adaptativo ideal, onde o número de bits de informação dos blocos também é otimizado, o TCP com *FEC* no enlace sem fio sempre apresenta melhor desempenho quando comparado ao TCP com *GBN-ARQ* ou com *SW-ARQ*. Novamente, em ambos os casos, o TCP com *SW-ARQ* apresenta o pior desempenho. Todas essas conclusões estão qualitativamente de acordo com aquelas encontradas nos estudos anteriores, apresentados em [15]. Especificamente nas redes com enlace satélite, é notório que a escolha do *FEC* é a mais interessante mesmo no cenário não-adaptativo, visto que o desempenho do TCP quando o *GBN-ARQ* é utilizado só é maior do que aquele quando o *FEC* é usado para taxas de erro de bit muito reduzidas, ou seja, para quase toda a faixa de erros de bit verificada, o TCP com *FEC* apresenta melhor desempenho. Essa conclusão também está de acordo com a recomendação indicada em [36].

Quanto ao modelo analítico proposto em si, é possível verificar uma boa proximidade dos seus resultados com os resultados obtidos através das simulações geradas no ns-2 em todas os sistemas computados. Desta forma, conclui-se sobre a validade do mesmo.

Para a continuidade deste trabalho, algumas sugestões podem ser colocadas, como a aprimoração do cálculo das probabilidades dos eventos, de forma que este se aproxime mais dos resultados obtidos por simulação; a adaptação do modelo para que se possa analisar o desempenho do TCP com os mesmos mecanismos de controle de erro quando o canal possuir

características de *fading*; a adaptação do modelo ao uso do mecanismo de *Selective Repeat ARQ* no enlace sem fio e, finalmente, a adaptação do modelo para analisar o desempenho do TCP quando implementações *cross-layer* são efetuadas, como por exemplo, o tamanho do segmento TCP sendo função da BER vista pelo mecanismo de camada de enlace.

Referência Bibliográficas

[1] Agência Nacional de Telecomunicações – ANATEL, www.anatel.gov.br.

[2] Wireless Intelligence, www.wirelessintelligence.com.

[3] Internet World Stats, www.internetworldstats.com.

[4] POSTEL, J.**Transmission Control Protocol**, RFC 793, Setembro 1981.

[5] CLAFFY, K.; MILLER, G.; THOMPSON, K. **The Nature of the Beast: Recent Traffic Measurements from an Internet Backbone**, 1998. Trabalho apresentado ao INET, Genebra, Suiça, Julho 1998.

[6] STEVENS, William Richard. **TCP/IP Illustrated, Vol. 1: The Protocols.** New Jersey: Addison-Wesley, 1997.

[7] JACOBSON, V., **Modified TCP congestion Avoidance Algorithm**, Lista de E-mail, Abril 1990.

[8] PAXSON, V.; ALLMAN, M. **Computing TCP's Retransmission Timer**, RFC 2988, Novembro 2000.

[9] KARN, P.; PARTRIDGE, C. **Improving round-trip time estimates in reliable transport protocols**. Trabalho apresentado ao SIGCOMM '87, Vermont, Canadá, Agosto 1987.

[10] FALL, Kevin.; FLOYD, Sally. Simulation-based comparisons of Tahoe, Reno, and SACK TCP. **Computer Communication Review**, v. 26, n. 3, Julho 1996.

[11] BALAKRISHNAN, Hari. **Challenges to Reliable Data Transport Over Heterogeneous Wireless Networks.** Berkeley: University of California, 1998. p. 203, Tese (Doutorado) – University of California, Berkeley, 1998.

[12] LIN, Shu; COSTELLO, Daniel J. **Error Control Coding: Fundamentals and Applications**. Prentice-Hall, 1983.

[13] REDD, Irvin. S.; CHEN, Xuemin. **Error Control Coding for Data Networks**, Kluer Academic Publishers, 1999.

[14] PROAKIS, John G. **Digital Communications, second edition**, McGraw-Hill, 1989.

[15] BRITO, José M.C. **Controle de erro em redes ATM sem fio**. Campinas: Universidade Estadual de Campinas, 2003. p. 142, Tese (Doutorado) – Universidade Estadual de Campinas, Campinas, Campinas, 2003

[16] WICKER, Stephen. B. **Error Control Systems for Digital Communication and Storage**. Prentice Hall, 1995.

[17] DESIMONE A.; CHUAH, M. C.; YUE O. C. **Throughput Performance of Transport-Layer Protocols over Wireless LANs**. Trabalho apresentado ao GLOBECOM'93, Houston, EUA, Dezembro 1993.

[18] GOFF, T.; MORONSKI J., PHATAK, D. S.; GUPTA, V. **Freeze-TCP: A true end-to-end TCP enhancement mechanism for mobile environments**, Trabalho apresentado ao IEEE Infocom, Março, 2000.

[19] SINGH, Ajay K.; IYER, Sridhar. **ATCP: Improving TCP Performance over Mobile Wireless Environments.** Trabalho apresentado ao IEEE Conference on Mobile and Wireless Communications Networks, Estocolmo, Suécia, Setembro, 2002.

[20] BRADEN, R. **Requirements for Internet Hosts - Communication Layers**, RFC 1122, Outubro, 1989.

[21] HAYKIN, Simon. **Digital Communication**. John Wiley & Sons, 1988.

[22] LAKSHMAN, T. V.; MADHOW U.; SUTTER, B. **Window-based Error Recovery and Flow Control with a Slow Acknowledgement Channel: a Study of TCP/IP Performance**, Trabalho apresentado ao INFOCOM, Kobe, Japão, Abril 1997.

[23] PARTRIDGE, C.; SHEPARD, T. TCP Performance over Satellite Links. **IEEE Network**, v. 11, n. 5, p. 44-49, Setembro 1997.

[25] AKYILDIZ, I. F.; MORABITO, G.; PALAZZO, S. TCP-Peach: A New Congestion Control Scheme for Satellite IP Networks. **IEEE/ACM Transactions on Networking**, v. 9, n. 3, p. 307-321, Fevereiro 2001.

[25] BAKRE, A.; BADRINATH, B. R. **I-TCP: Indirect TCP for Mobile Hosts**. Trabalho apresentado ao 15th International Conference on Distributed Computing Systems – ICDCS, Sendai, Japão, Maio 1995.

[26] ZHANG, Y.; DELUCIA, D.; RYU, B.; DAO, S. **Satellite Communications in the Global Internet: Issues, Pitfalls, and Potential**. Trabalho apresentado ao INET'97, Kuala Lumpur, Malásia, Junho 1997.

[27] ISHAC, J.; ALLMAN, M. **On the Performance of TCP Spoofing in Satellite Networks.** Trabalho apresentado ao IEEE MILCOM, Washington, EUA, Outubro 2001.

[28] LAKSHMAN, T.; MADHOW, U. The Performance of TCP/IP for Networks with High Bandwidth-Delay Products and Random Loss. **IEEE/ACM Transactions on Networking**, v. 5, n. 3, p. 336-350, Junho 1997.

[29] COSTA PINTO, Bruno C.; BRITO, José M. C. **Throughput of Wireless TCP Networks using Reliable Radio Links**. Trabalho apresentado ao 63° IEEE Vehicular Technology Conference VTC06, Melbourne, Austrália, Maio 2006.

[30] COSTA PINTO, Bruno C.; BRITO, José M. C. **An Analytical Model to Compute the Throughput of Wireless TCP Networks**. Trabalho aprovado a ser apresentado ao International Telecommunications Symposium 2006 – ITS2006, Fortaleza, Ceará, Setembro 2006.

[31] COSTA PINTO, Bruno C.; BRITO, José M. C. **Throughput of Satellite TCP Networks**. Trabalho aprovado a ser apresentado ao 3° IASTED International Conference on Communications and Computer Networks – CCN 2006, Lima, Peru, Outubro 2006.

[32] CAIN, J. B.; MCGREGOR, D. N. A Recommended Error Control Architecture for ATM Networks with Wireless Links. **IEEE Journal on Selected Areas in Communications**, v. 15, n. 1, p. 16-28, Janeiro 1997.

[33] GOYAL, Rohit; JAIN, Raj; KALYANARAMAN, Shiv; FAHMY, Sonia; VANDALORE, Bobby. Improving the Performance of TCP over the ATM-UBR service. **Computer Communications**, v. 21, n. 10, p. 898-911, Julho 1998.

[34] BRANDEN, R. **A Perspective on the Host Requirements RFCs.** RFC 1127, Outubro 1989.

[35] ALLMAN, M.; FLOYD, S.; PARTRIDGE C. **Increasing TCP's Initial Window.** RFC 2414, Setembro 1998.

[36] ALLMAN, M.; GLOVER, D.; SANCHEZ, L. **Enhancing TCP Over Satellite Channels using Standard Mechanisms**. RFC 2488, Janeiro 1999.

[37] CHOKALINGAM, A.; ZORZI, M. **Wireless TCP Performance with Link Layer *FEC/ARQ***, Trabalho apresentado ao IEEE International Conference on Communications, Vancouver, Canada, Junho 1999.

[38] XYLOMENOS, G.; POLYZOS, G. C. **Wireless Link Layer Enhancements for TCP and UDP Applications**, Trabalho apresentado ao 17° International Symposium on Parallel and Distributed Processing, 2003.

[39] BARAKAT, C.; AL FAWAL, A. Analysis of Link-Level Hybrid *FEC*/*ARQ*-*SR* for Wireless Links and Long-Lived TCP Traffic. **INRIA Research Report RR-4752**, Fevereiro 2003.

Apêndice I – O Modelo de Simulação no ns-2.

Neste apêndice será mostrado, de forma geral, o modelo utilizado para obtenção dos resultados de simulação utilizados para validação dos modelos analíticos propostos. Todas os modelos de simulação foram criados utilizando o software ns-2 (network simulator 2) na versão 2.28.

Para facilitar o entendimento do processo de criação do modelo de simulação, será mostrado o caso do modelamento da Rede C, definida na Seção 4.2.1, cujos parâmetros são: μ = 100 pacotes/s, B = 20 pacotes, t_{pw} = 50µs, t_{aw} = 0,9ms e τ = 1s ($\tau_b = 2 \cdot t_{pw} + t_{aw}$). Para construir o modelo desta rede é necessário criar um arquivo de script TCL, que fica da seguinte forma:

Arquivo rede.tcl:

```
### Cria um novo objeto de simulação ###
set ns [new Simulator]

### Procedimento fim {} ####
proc fim {} {
  global ns tcp TCP_size rate dur cwn sst nf tr TCP_size
  $ns flush-trace
  #np recebe o total de segmentos TCP enviados
  set np [$tcp set ndatapack_]
  #gn recebe o valor do goodput normalizado obtido pela simulação
  set gn [expr ((1.0*$np/($dur*100.0))*((8.0*$TCP_size-320.0)/(8.0*$TCP_size)))]
  #Imprime o resultado na tela
  puts "Goodput Normalizado: $gn"
  #Termina a simulação
  exit 0
}

### Parâmetros da simulação ###
#Tamanho do segmento TCP (em bytes)
set TCP_size 540
#Tamanho do ACK TCP (em bytes)
set ack_size 1
```

```
#Tempo de simulação (em segundos)
set dur 15000

### Criação da Rede ###
#Cria 3 nós
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
#Define uma taxa de transmissão enlace n0->n1 tal que o atraso de transmissão seja 5ms
set rate01 [expr 8*$TCP_size/0.005]
#Define uma taxa de transmissão enlace n1->n2 (enlace sem fio - gargalo do sistema) igual a
100 pacotes/segundo
set rate12 [expr 8*$TCP_size*100]
#Define uma taxa de transmissão enlace n2->n1 tal que o atraso de transmissão seja 0.9ms
set rate21 [expr 8*$ack_size/0.0009]
#Define a taxa de transmissão a ser usada no enlace n1->n0
set rate10 [expr 8*$ack_size/0.005]
#Cria os enlaces com as taxas de transmissão já definidas e os atrasos em milisegundos.
#Define também o tipo de fila nas saídas dos nós.
#(Droptail = FIFO com descarte dos segmentos que chegam quando o buffer já está cheio)
$ns simplex-link $n0 $n1 $rate01 494.5ms DropTail
$ns simplex-link $n1 $n2 $rate12 0.05ms DropTail
$ns simplex-link $n2 $n1 $rate21 0.05ms DropTail
$ns simplex-link $n1 $n0 $rate10 494.5ms DropTail
#Define o tamanho do buffer do enlace n1->n2 (enlace sem fio) como 20
#(na verdade, 20 na fila + 1 sendo transmitido = 21)
$ns queue-limit $n1 $n2 21

### Cria e configura conexão TCP ###
#Cria agente TCP-Reno
set tcp [new Agent/TCP/Reno]
#Configura janela máxima de congestionamento para 10000 (infinito)
$tcp set window_ 10000
#Configura tamanho dos segmentos TCP
```

```
$tcp set packetSize_ $TCP_size
#Configura não utilização de cabeçalhos TCP/IP
#(Os bytes relativos a esses cabeçalhos já estão computados no tamanho do TCP)
$tcp set useHeaders_ false
#Configura tamanho inicial de janela para 1
$tcp set windowInit_ 1
#Associa agente TCP-Reno ao nó n0
$ns attach-agent $n0 $tcp
#Cria um coletor para o TCP-Reno
#(o coletor é responsável pela retirada dos segmentos TCP da simulação e pela geração dos
#ACKs TCP)
set sink [new Agent/TCPSink]
#Configura o tamanho dos ACKs TCP
$sink set packetSize_ $ack_size
#Associa o coletor ao nó n2
$ns attach-agent $n2 $sink
#Conecta o agente TCP-Reno do nó n0 ao coletor do nó n2 (cria a conexão TCP)
$ns connect $tcp $sink

### Cria aplicação FTP (esta aplicação gera dados infinitamente) ###
set ftp [new Application/FTP]
#Associa o agente FTP ao agente TCP-Reno (desta forma o TCP sempre tem dados a serem
transmitidos)
$ftp attach-agent $tcp

### Agendamento da simulação ###
#Inicia a aplicação FTP no instante 0
$ns at 0 "$ftp start"
#Finaliza a aplicação FTP no instante $dur (que define a duração da simulação)
$ns at $dur "$ftp stop"
#Chama o procedimento fim{} no instante $dur (que define a duração da simulação)
$ns at $dur "fim"

### Roda a simulação ###
$ns run
```

A rede criada por este script será a dada na Figura A.1. O atraso global será dado por τ = 5ms (tempo de transmissão n0→n1) + 494,5ms (atraso de propagação n0→n1) + 50μs (atraso de propagação n1→n2) + 0,9ms (tempo de transmissão n2→n1) + 50μs (atraso de propagação n2→n1) + 5ms (tempo de transmissão n1→n0) + 494,5ms (atraso de propagação n1→n0) = 1 s, como era desejado.

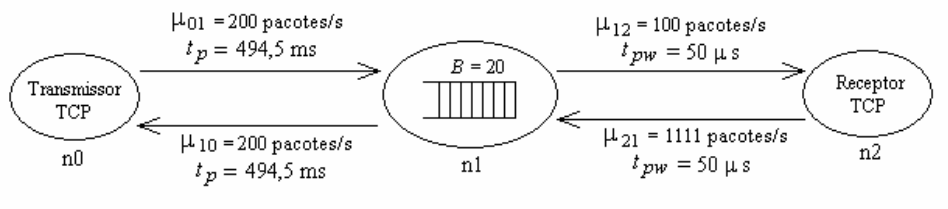


Figura A.1 – Esquema da simulação para a Rede C.

Para simular a inserção dos protocolos de camada de enlace no enlace sem fio serão efetuadas alterações no método void LinkDelay::recv(Packet* p, Handler* h) da classe LinkDelay, presente no arquivo delay.cc. Este procedimento é responsável por agendar na simulação dois eventos referentes às transmissões dos pacotes nos enlaces: a retirada do pacote do *buffer* de saída do nó de origem e chegada do pacote no nó de destino. Alterando a temporização desses agendamentos, pode-se simular a inserção dos mecanismos de camada de enlace. Cada alteração é dependente do mecanismo inserido no enlace sem fio. Quando nenhum protocolo é utilizado, o procedimento recv utilizado é o originalmente escrito no ns-2, dado por:

```
void LinkDelay::recv(Packet* p, Handler* h) {
  //Obtém o tempo de transmissão no enlace a ser transmitido
  double txt = txtime(p);
  //Cria uma instância de agendamento
  Scheduler& s = Scheduler::instance();
  //Agenda o recebimento do pacote no nó de destino.
  s.schedule(target_, p, txt + delay_);
  //Agenda a retirada no pacote do buffer do nó de origem.
  s.schedule(h, &intr_, txt);
}
```

Quando os mecanismos de controle de erro são utilizados, o procedimento recv é alterado. O código abaixo ilustra as modificações em recv, que neste caso específico está configurado para a utilização do *FEC*, com BER = 10^{-2} e código (4657,4320).

```
void LinkDelay::recv(Packet* p, Handler* h) {
  double txt = txtime(p);
  Scheduler& s = Scheduler::instance();
  //Variáveis adicionadas.
  double tr;    //Define instante de tempo em que o bloco será recebido no nó de destino.
  double  tl;   //Define instante de tempo em que o bloco será liberado do buffer de saída do
                //nó de origem.
  double k;     //Define o número de bits de informação do bloco.
  double red;   //Define o número de bits de paridade do bloco.
  double n;     //Define o tamanho total do bloco.
  int mec;      //Define o mecanismo de controle de erro em uso. FEC = 0, SW-ARQ = 1 e
                //GBN-ARQ = 2.
 //Inicialização das variáveis adicionadas.
  tack = 0.9e-3;
  BER = 1e-2;
  k = 4320.0;
  red = 337.0;
  n = k+red;
 mec = 0;
 //Só entra neste "if" caso o enlace em questão seja o enlace sem fio,
 //pois é o único em que o tempo de transmissão é igual a 0.01s.
  if (txt == 0.01)) {
        //txt é atualizado para um tempo de transmissão "virtual", dependente dos valores de n
        //e k.
        txt = n/(100.0*k);
        //switch dependente do mecanismo de controle de erro utilizado.
        switch (mec) {
              case 0: //FEC
                    //O instante de liberação é atualizado para após a transmissão do bloco,
                    //considerando o novo tempo de transmissão.
```

```
            tl = txt;
            //O instante de recebimento é atualizado para o novo tempo de
            //transmissão mais o tempo de propagação no enlace, dado por delay_.
            tr = tl + delay_;
break;
case 1: //SW-ARQ
            //O instante de recebimento é atualizado.
            tr = txt + delay_;
            //O instante de liberação é atualizado, levando-se em conta o SW-ARQ.
            tl = tr + tack + delay_;
            //Sorteia um número aleatório entre 0 e 1.
            r = ((double)rand())/RAND_MAX;
            //Calcula a probabilidade de o bloco chegar sem erro no nó de destino.
            pc = pow((1.0-BER),n);
            //Caso tenha sido transmitido corretamente na primeira transmissão o
            //loop "while" não é acessado. Caso entre no loop, os valores das
            //variáveis de tempo são atualizadas levando em conta o SW-ARQ, e
            //um novo teste para verificação da correção da transmissão do bloco é
            //efetuada. O programa sai do loop quando o bloco é finalmente
            //transmitido com sucesso.
            while (r>pc) {
                        tr = tr + tack + delay_ + txt + delay_;
                        tl = tr + tack + delay_;
                        r = ((double)rand())/RAND_MAX;
            }
break;
case 2: //GBN-ARQ
            //O instante de liberação é atualizado, levando-se em conta o
            //GBN-ARQ.
            tl = txt;
            //O instante de recebimento é atualizado.
            tr = tl + delay_;
            //Sorteia um número aleatório entre 0 e 1.
            r = ((double)rand())/RAND_MAX;
```

```
                    //Calcula a probabilidade de o bloco chegar sem erro no nó de destino.
                    pc = pow((1.0-BER),n);
                    //Caso tenha sido transmitido corretamente na primeira transmissão o
                    //loop "while" não é acessado. Caso entre no loop, os valores das
                    //variáveis de tempo são atualizadas levando em conta o GBN-ARQ, e
                    //um novo teste para verificação da correção da transmissão do bloco é
                    //efetuada. O programa sai do loop quando o bloco é finalmente
                    //transmitido com sucesso.
                        while (r>pc) {
                        tl = tr + tack + delay_ + txt;
                        tr = tl + delay_;
                        r = ((double)rand())/RAND_MAX;
                    }
                break;
        }
        //Agendamento do recebimento do bloco no nó seguinte.
        s.schedule(target_, p, tr);
        // Agendamento da liberação bob loco do buffer de saída do nó de origem.
        s.schedule(h, &intr_, tl);
  } else {
        //Agendamentos normais para os demais enlaces.
        s.schedule(target_, p, txt + delay_);
        s.schedule(h, &intr_, txt);
  }
}
```

Nota-se que no caso do *FEC*, a simulação da inserção deste mecanismo se dá através de uma alteração no algoritmo que faz com que o envio do bloco seja mais lento do que ocorreria quando o *FEC* não fosse utilizado e, portanto, quando bits de paridade não fossem anexados aos bits de informação. No caso dos mecanismos *SW-ARQ* e *GBN-ARQ*, a simulação da inserção destes mecanismos se dá através do atraso do instante de retirada do bloco do *buffer* de saída do nó de origem e do instante de recebimento do bloco no nó de destino. Dependendo do número de transmissões necessárias para que o bloco chegue corretamente ao

destino, serão calculados novos instantes para a retirada do bloco do *buffer* de saída do nó de origem e para o recebimento deste bloco no nó de destino.

É importante destacar que após toda alteração feita no método recv, é necessário uma recompilação do ns-2, visto que essas alterações são feitas no *core* do simulador.

Finalmente, é importante ressaltar que o resultado final do throughput normalizado obtido através de simulação para cada cenário (mecanismo de controle de erro utilizado, tamanho do buffer, atrasos de transmissão e propagação e BER) foi calculado a partir a média dos resultados de dez simulações. Desta forma, fica diminuída a probabilidade de sobreestimar ou subestimar os resultados obtidos na simulação.