# UMA ANÁLISE DOS PROBLEMAS DO PROTOCOLO TCP EM REDES ASSIMÉTRICAS E SOLUÇÕES EXISTENTES

ERIC FERNANDES DE MELLO ARAÚJO

# UMA ANÁLISE DOS PROBLEMAS DO PROTOCOLO TCP EM REDES ASSIMÉTRICAS E SOLUÇÕES EXISTENTES

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação do Instituto de Ciências Exatas da Universidade Federal de Minas Gerais como requisito parcial para a obtenção do grau de Mestre em Ciência da Computação.

ORIENTADOR: DORGIVAL OLAVO GUEDES NETO

Belo Horizonte

Setembro de 2009

UNIVERSIDADE FEDERAL DE MINAS GERAIS
INSTITUTO DE CIÊNCIAS EXATAS
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

# FOLHA DE APROVAÇÃO

Uma análise dos Problemas do protocolo TCP em redes assimétricas e soluções existentes

**ERIC FERNANDES DE MELLO ARAÚJO**

Dissertação defendida e aprovada pela banca examinadora constituída pelos Senhores:

PROF. DORGIVAL OLAVO GUEDES NETO - Orientador
Departamento de Ciência da Computação - UFMG

PROF. CARLOS DE CASTRO GOULART
Departamento de Informática - UFV

PROF. JOSÉ MARCOS SILVA NOGUEIRA
Departamento de Ciência da Computação - UFMG

Belo Horizonte, 30 de setembro de 2009.

*Dedico essa dissertação ao meu avô Jandir, exemplo de vida e que deixou muitas saudades.*

# Agradecimentos

Agradeço aos professores que ao longo da minha carreira têm me feito crer que o ensino é papel fundamental para a melhoria de nossa sociedade. Em especial, quero agradecer ao Professor Dorgival, que me orientou durante esse tempo com paciência e compreensão, sempre em busca de um trabalho coerente e carregado de excelência.

Agradeço à minha família, por ser meu ponto de apoio em momentos de dificuldade. Agradeço aos amigos pelas risadas e descontrações necessárias em momentos mais estressantes.

Agradeço à Natália, pelo carinho, apoio e admiração, que cada dia me fazem querer ser uma pessoa e um profissional melhor.

Agradeço ao Roberto e família pelo companheirismo, cumplicidade e carinho em todos os momentos.

Agradeço aos meus pais, pela vida. À minha mãe pelos livros. Ao meu pai pela integridade e força de vontade, exemplo maior na minha caminhada.

Agradeço a Deus, por me dar oportunidades e experiências tão importantes e invejáveis. Que eu possa ser um farol que brilha durante uma noite de tempestade, um abrigo no deserto, uma flecha que acerta o alvo, de forma a refletir a sabedoria do Criador maior.

Agradeço ao Coldplay, Jars of Clay, Bob Marley, Teatro Mágico, Vanessa da Mata, Third Day, Érica Machado, Michael Jackson, entre tantos, por embalarem minhas noites sem sono enquanto esse texto estava sendo redigido.

*"Quem não serve, não serve."*
(Autor desconhecido)

# Resumo

As redes assimétricas ganharam grande aceitação no mercado de acesso à Internet devido a facilidade de sua implementação, que necessita apenas de uma linha telefônica para tal. A arquitetura dessas redes foi feita considerando que o usuário é mais passivo do que ativo quando interage com a Web, ou seja, faz mais *download* do que *upload* através do seu computador. Com o surgimento da Web 2.0 e de redes P2P, novos paradigmas surgiram alterando significativamente o comportamento do usuário, que agora é tão ativo quanto passivo no trânsito de informações para a Internet. O protocolo TCP (*Transmission Control Protocol*) é um protocolo da camada de transporte da pilha TCP/IP e hoje é o mais utilizado nas aplicações da Internet. O TCP apresenta como características importantes a confiabilidade na entrega dos segmentos enviados, bem como baseia a taxa que transmite na chega de pacotes de confirmação (os ACKs), que são usados para perceber se a rede está congestionada ou não, de forma a regular a quantidade de dados que serão enviados pelo emissor da transmissão. Quando transmitido em redes assimétricas, com segmentos de dados sendo transmitidos nos dois sentidos, temos que o canal de retorno para o ACKs (*uplink*) não será suficiente para enviar todos os dados e os ACKs simultaneamente, gerando atrasos para os ACKs. Isso irá gerar atrasos ou até mesmo perdas em algumas situações, e consequentemente irá reduzir o desempenho do TCP. Esse trabalho se propõe a estudar esse problema, levantando as causas, modelando-o matematicamente para melhor entendimento e estudando outros problemas que buscam solucioná-lo, categorizando-os de acordo com sua abordagem, podendo esta ser alterando o protocolo TCP ou incluindo outros dispositivos nas entradas dos canais para efetuar as tarefas.

**Palavras-chave:** TCP, Redes Assimétricas, ACKs.

# Abstract

The asymmetric networks have gained wide acceptance in the market for Internet access because of its ease of implementation, which requires only one phone line for this. The architecture of these networks was made considering that the User is more passive than active when interacting with the Web, that is, it download more than upload from his computer. With the emergence of Web 2.0 and P2P networks, new paradigms have emerged significantly changing the behavior of User, which is now so active as passive in traffic information for the Internet. The TCP (Transmission Control Protocol) is a protocol of transport layer of the TCP / IP and is now the most widely used in Internet applications. TCP has some important characteristics as the reliability in the delivery of segments sent, using confirmations to the packet that arrives correctly (the ACKs). This packets are used to see if the network is congested or not, to regulate amount of data that will be sent by the sender of the transmission. When TCP is placed in asymmetric networks, with data segments being transmitted in both directions, we have the return channel for ACKs (uplink) will not be enough to send all the data and ACKs at the same time, creating delays for ACKs. This will cause delays or even losses in some situations, and therefore will reduce the performance of TCP. This paper aims to study this problem, raising the causes, modeling it mathematically to better understanding and studying other problems that seek to solve it, categorizing them according to their approach, which may be changing the TCP or other devices including the entrances of the channels to perform the tasks.

**Keywords:** TCP, Asymmetric Links, Acknowledgments.

# Lista de Figuras

# Lista de Tabelas

# Sumário

# Capítulo 1

# Introdução

A Internet é hoje mundialmente divulgada e fundamental para o funcionamento da economia e das grandes empresas e governos no mundo, além de fazer parte do cotidiano das pessoas, fornecendo lazer, informação e ferramentas para trabalho. A Internet inicialmente recebeu o nome de ArpaNet. Através da antiga ArpaNet, instituições de ensino se comunicavam dentro dos Estados Unidos apenas. A sua popularização se deu em função da facilidade de uso e rapidez no envio de informações entre pontos distintos no mundo. Com duas máquinas ligadas pela Internet é possível enviar e receber dados de maneira confiável. Com o surgimento do WWW (*World Wide Web*), foi possível integrar imagens e sons às comunicações, o que tornou a Internet mais atraente e extremamente útil para inúmeras finalidades.

Várias tecnologias surgiram para o uso da Internet à medida que a mesma se popularizava. Entre elas, as redes domésticas ganharam um grande espaço. As principais redes domésticas utilizam os sistemas telefônicos para transmissão dos dados, ou seja, permitem a conexão na grande maioria das casas, que já contam com telefonia. Por isso, o custo dessas redes foi reduzido e, atualmente, são as redes mais utilizadas para conexão das máquinas a Internet. As redes domésticas levavam em conta que o usuário era mais passivo do que ativo, ou seja, ele recebia mais informações do que enviava para a rede. Dessa forma, temos que as redes domésticas apresentam uma capacidade de receber dados por parte do cliente centenas a milhares de vezes maior do que a capacidade de enviar dados, sendo dessa forma assimétricas.

Com o avanço das aplicações na Internet, como a Web 2.0 e as redes par-a-par (P2P), mudaram-se as características que antes eram predominantes e que foram bases para a implementação assimétrica das redes domésticas. Agora, o usuário interage de tal forma com a rede que a capacidade de envio de dados deve ser tão grande quanto a de receber, o que não é possível se usada uma rede doméstica.

Aplicações P2P, por exemplo, são aplicações que comunicam um nó da rede (*peer*) a vários outros nós e transmite e recebe dados nos dois sentidos, a fim de compartilhar arquivos. Dessa forma, várias conexões são abertas simultaneamente, sendo que um *peer* pode apenas enviar ou receber informações de um outro *peer* da rede, ou pode enviar e receber simultaneamente de um mesmo *peer*. No primeiro caso, estão ocorrendo várias conexões unidirecionais, e no segundo são transmissões bidirecionais.

Em aplicações da Web 2.0, é possível ao usuário utilizar serviços da Internet como se fossem aplicações no seu próprio computador. Dessa forma, o usuário interage de maneira mais constante com o servidor que provê o site, produzindo assim tráfego nos dois sentidos e bidirecional, uma vez que envia informações e recebe respostas praticamente ao mesmo tempo.

Algumas outras aplicações unidirecionais incluem o correio eletrônico, vídeos por demanda, entre outros. Atualmente, aplicações unidirecionais são predominantes nos dois sentidos de transmissão, porém para redes P2P, o uso de tráfego bidirecional se torna mais presente.

Por trás de toda essa estrutura, é usada uma arquitetura que chamamos de TCP/IP. Essa arquitetura se baseia em camadas que se intercomunicam de forma a garantir algumas características para a transmissão de informações. O TCP (*Transmission Control Protocol*) é um protocolo usado na camada de transporte e tem algumas características importantes como confiabilidade, transmissão em fluxos, entre outras que veremos no capítulo 2. O princípio que informa a chegada de pacotes corretamente para essa arquitetura é o envio de pacotes de confirmação para os dados que chegam corretamente ao seu destino. A esses pacotes é dado o nome de *Acknowledgments*, ou ACKs. Para que a eficiência do TCP não seja comprometida, é importante que os ACKs cheguem sem bloqueios ou atrasos ao longo do caminho reverso.

Outro protocolo de transporte comumente utilizado é o UDP (*User Datagram Protocol*). Este protocolo não utiliza ACKs para confirmação, e portanto não é confiável como o TCP. É utilizado em transmissões de áudio e vídeo em tempo real, entre outras aplicações que não necessitam que os pacotes sejam confirmados. O IP (*Internet Protocol*) é o protocolo usado na camada de rede, e seu cabeçalho fornece informações para que os roteadores da rede enviem os dados presentes no pacote para o local correto.

Dessa forma, como a maioria das aplicações utiliza o TCP e acessa a Internet através de redes domésticas assimétricas, temos problemas resultantes da pouca banda disponível no canal reverso, como veremos nesse trabalho.

## 1.1 Descrição do Problema

Em redes onde há assimetria entre as bandas de *download* (envio de dados da Internet para o cliente) e *upload* (envio de dados do cliente para a Internet) temos problemas para protocolos de transporte que necessitam de confirmação, como o TCP. Mesmo que o canal de *download* esteja desimpedido, temos que um congestionamento no canal reverso (*upload*) pode levar a transmissão pelo canal de *download* a sofrer perdas de desempenho.

Quando temos tráfego de dados ocupando o canal reverso, é possível que um pacote de confirmação da outra transmissão fique aguardando na fila de saída do receptor (cliente) para ser enviado. Esse atraso pode ser suficiente para que outros ACKs se enfileirem atrás do pacote de dados, fazendo com que vários ACKs sejam emitidos um após o outro para o transmissor, o que leva o TCP do emissor a enviar pacotes de dados um após o outro, consequentemente. Esse envio de pacotes de dados em rajadas pode ser prejudicial para a transmissão, e não é aconselhável, como veremos mais a frente. Além disso, as esperas podem ser grandes o suficiente para que o emissor fique um tempo considerável aguardando para transmitir novos dados. O fenômeno causado aqui é chamado de Compressão de ACKs, como veremos nos capítulos seguintes.

Esta dissertação tem como motivação entender melhor o problema presente em redes assimétricas quando temos o canal reverso saturado com dados e ACKs. Buscamos mapear o estado da arte nesse campo de pesquisa onde ainda não foi apresentada uma solução eficaz, causando uma degradação considerável quando há imperfeição e variabilidade no retorno dos ACKs (Balakrishnan et al., 2002). Ao longo desse trabalho, verificamos que trabalhos mais atuais não apresentam grandes avanços na resolução do problema aqui apresentado. Dessa forma, as principais referências aqui consideradas são as mais relevantes no que diz respeito a resolução do problema. Outros trabalhos mais recentes (Balakrishnan & Padmanabhan, 2001; Louati et al., 2003; Al-Khatib & Gunavathi, 2008) não apresentaram melhorias ao problema, mas apenas utilizaram técnicas já implementadas para verificar o comportamento das alterações em diferentes ambientes.

Balakrishnan et al. (2002) definem a assimetria de rede como sendo a diferença entre os canais de recepção e envio de dados em um dado ponto da mesma. Essas redes podem ser cabeadas, como as redes ADSL (*Assymmetric Digital Subscriber Line*), redes via satélite, redes via rádio, que apresentam diferenças entre os canais que guiam os fluxos de entrada e saída. Essas diferenças podem dizer respeito a capacidade de transmissão do canal, diferenças nas taxas de perdas ou diferenças nos caminhos de entrada e saída.

Alguns trabalhos mostram resultados em ambientes assimétricos diferentes do que vemos hoje (Wu et al., 2005; Dawkins et al., 2000; Balakrishnan et al., 1997). Um exemplo são as redes via satélite, que há um tempo atrás só eram utilizadas para o canal de *download*. Os pacotes de confirmação geralmente retornavam ao destino por rede discada. Dessa forma, a assimetria, além de ser agravada pela diferença entre as bandas, ainda sofria com a diferença nos tempos de transmissão nos dois sentidos. Atualmente, mesmo em redes de satélites, tanto *download* quanto *upload* são feitos utilizando o mesmo canal, ou seja, não há assimetria de atrasos no envio de dados. O tamanho dos pacotes também sofreram alterações com o aumento do MTU (*Maximum Transmission Unit*) atual na Internet. Além disso, alguns protocolos de compressão de cabeçalhos caíram em desuso, como é o caso do CSLIP (*Compressed Serial Line Interface Protocol*). Hoje, com aumento nas velocidades das bandas e com uma diferença maior entre pacotes de dados e ACKs (cerca de 1 para 25) não é tão significativo o uso de compressão nos pacotes.

A assimetria devido à diferença de capacidade de transmissão entre os dois canais da rede pode ser devido a implementação da rede, que busca oferecer ao cliente uma banda mais rápida de recepção de dados em detrimento da capacidade de envio, geralmente por motivos relacionados ao custo da implantação — redes ADSL (Knight, 1999), por exemplo.

Essa assimetria é agravada quando o canal de menor capacidade é utilizado por fluxos que vão do cliente da transmissão para o restante da rede, como visto na grande maioria dos trabalhos apresentados. Isso faz com que a assimetria se agrave e prejudique todos os fluxos naquele ponto específico da rede. Isso se deve à necessidade do protocolo TCP de que os pacotes de confirmação não encontrem barreiras no caminho de retorno ao emissor, o que garante que os mecanismos que se baseiam nessas chegadas não seja prejudicado, alterando consequentemente o fluxo maior de dados. Além disso, o fluxo do canal reverso é prejudicado na medida que os pacotes de confirmação da transmissão mais rápida ocupam espaço no canal reverso que poderia ser utilizado para envio de dados do cliente para o restante da rede (*upload*).

Buscando resolver os problemas da assimetria no desempenho do TCP, vários autores apresentaram propostas das mais diversas, sugerindo alterações no TCP ou ao longo do canal de transmissão. Neste trabalho fazemos um levantamento desses trabalhos. Identificamos as razões para os problemas entre TCP e redes assimétricas e mostramos as técnicas utilizadas para tentar solucionar a perda de desempenho nestes casos.

A maioria dos trabalhos se preocupa inicialmente em garantir uma transmissão ótima do *download*, garantindo aos ACKs condições especiais no canal reverso de forma

que cheguem ao emissor sem maiores problemas. O problema se agrava quando o desempenho no envio de dados pelo canal reverso também é levado em consideração. Aplicações que demandam envio e recepção de dados no cliente, como P2P, acabam sofrendo consideráveis degradações quando apenas o canal de *download* é otimizado. Na maioria dos casos o canal de *upload* é suficiente para transmitir todos os ACKs sem problemas, mas quando temos dados trafegando é preciso definir melhor como será o uso do canal.

Segundo Balakrishnan et al. (2002) existem duas necessidades importantes quando enfrentamos os problemas descritos aqui neste trabalho. A primeira delas é a importância do gerenciamento da capacidade do canal reverso, onde dados e ACKs competem pelo espaço. Neste caso, se dermos prioridades máximas a um dos dois, afetaremos fatalmente a outra transmissão. A segunda necessidade é evitar o impacto causado quando temos ACKs menos frequentes na rede. Reduzir a quantidade de pacotes de confirmação enviadas de volta afeta diretamente o TCP emissor, que depende dos ACKs para manter a sua taxa de envio.

Através de um modelo matemático conseguimos entender como é a interferência que os dados transmitidos no canal reverso têm sobre a chegada dos ACKs. Estudamos e expusemos os trabalhos que, por vários meios, buscam eliminar o fenômeno de compressão de ACKs e melhorar a eficiência nas taxas de transmissão. Devido ao uso em massa de redes com *cable modem*, consideramos valores de bandas referentes às bandas atualmente fornecidas pelas empresas de Internet no Brasil em nossa modelagem.

Separamos as estratégias em categorias diferentes para melhor entender as abordagens de solução, bem como a viabilidade de sua implementação no contexto atual. Essa categorização nos ajuda a entender melhor os caminhos que podemos seguir quando buscamos uma solução para a perda de desempenho em redes assimétricas.

Outros tipos de redes assimétricas também são contempladas tanto pelo modelo matemático quanto pelas soluções apresentadas, uma vez que apresentarão comportamento parecido ou idêntico ao que iremos expor em redes ADSL. O cenário básico, tanto para os experimentos dos autores quanto para a nossa modelagem se encontra na figura 1.1

## 1.2 Objetivos

Nosso objetivo principal é fazer um levantamento do estado da arte para o problema de assimetria em redes que utilizam o TCP para transmissão de dados. Dessa forma, buscamos conhecer melhor o problema e as soluções já propostas.

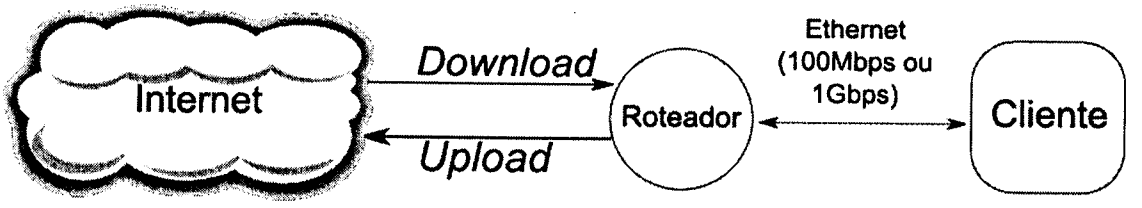


**Figura 1.1.** Modelo de transmissão usado nas simulações para estudar a assimetria de banda em uma transmissão TCP

Para isso, fizemos um levantamento bibliográfico dos trabalhos mais relevantes na área, com um estudo aprofundado das abordagens e soluções propostas. Ao longo desse trabalho, observamos que grande parte das referências mais relevantes no contexto de redes assimétricas estudava o fenômeno em satélites, onde a assimetria era agravada com a diferença entre os atrasos no envio e recepção de dados. Aqui levamos em consideração apenas a assimetria de banda, devido ao novo contexto em que vivemos.

Através de uma modelagem matemática conhecemos o quanto as transmissões são degradadas quando lidamos com tráfego bidirecional. Ao fim, buscamos uma categorização para entender as diversas formas de abordagens e soluções para o problema até o presente momento.

## 1.3 Metodologia

A dissertação tem um foco estritamente teórico, onde queremos esgotar o assunto por meio do levantamento dos trabalhos na área. Os desafios aqui presentes dizem respeito ao método de pesquisa, onde buscou-se melhor entender o problema e expor as soluções até o momento apresentadas de maneira a deixar o leitor contextualizado e ciente das dificuldades encontradas nesse campo de pesquisa.

Como veremos, nenhuma solução é suficientemente boa e todas apresentam suas dificuldades ou incapacidades, o que deixa o problema do TCP em redes assimétricas ainda em aberto. Bons resultados são encontrados, mas dificuldades na implantação dos métodos muitas das vezes impedem sua utilização em larga escala.

## 1.4 Abordagem desta dissertação

Esta dissertação apresenta a seguinte organização:

- No capítulo 2, temos um estudo mais aprofundado do protocolo TCP para melhor entender seu funcionamento e os mecanismos que ele apresenta.

- No capítulo 3 apresentamos nosso modelo matemático, que busca levantar o problema e embasar de uma maneira analítica o estudo aqui presente.
- No capítulo 4, apresentamos nosso estudo com as técnicas propostas. Categorizamos os trabalhos estudados com a finalidade de mapear os esforços para resolver os problemas aqui já expostos.
- No capítulo 5 fazemos nossas conclusões e considerações finais, bem como expomos trabalhos futuros que possam vir a surgir partindo do material aqui preparado.

Esperamos com esse material contextualizar qualquer leitor que não tenha tanto conhecimento na área ainda, expondo de maneira clara os conceitos básicos que definem o funcionamento do TCP e os trabalhos em redes assimétricas que expõem e solucionam os problemas enfrentados quando colocamos o TCP nesse tipo de ambiente.

## 1.5 Contribuições

Nessa dissertação, temos como contribuições principais:

- Um estudo detalhado do TCP, bem como dos problemas enfrentados em redes assimétricas. Através desse estudo, foi possível entender os mecanismos que o TCP utiliza para enviar dados de maneira ótima e confiável, o que contribui para o entendimento do problema causado quando a transmissão se dá via redes assimétricas.
- Uma modelagem matemática para o problema em questão. Nessa modelagem pudemos averiguar e quantificar o problema do TCP de maneira mais concreta.
- Levantamento do estado da arte para os trabalhos que estudam o TCP em redes assimétricas e suas soluções. Dentre esses trabalhos, destacamos os que apresentam idéias inovadoras para a solução do problema.

Esperamos, com isso, suscitar no leitor o interesse pelo estudo e pela busca de melhorias no que tange o uso do TCP nas redes com assimetria de bandas.

# Capítulo 2

# Protocolo TCP

O TCP (*Transmission Control Protocol*) é o protocolo que provê confiabilidade na entrega dos dados dentro da arquitetura TCP/IP (Tanenbaum, 2002). A arquitetura TCP/IP apresenta como protocolos principais o TCP e o UDP (*User Datagram Protocol*) na camada de transporte e o IP na camada de rede. A utilização do TCP em conjunto com o IP (*Internet Protocol*) corresponde à maioria do tráfego existente hoje na Internet. Este protocolo tem como características principais a garantia na entrega dos dados, o que não ocorre com o UDP, mais usado em aplicações com transações de requisição/resposta e em aplicações de tempo real, como transmissões de áudio e vídeo. O TCP, por sua vez, dá suporte a aplicações como transferência de dados onde a confiabilidade na entrega dos mesmos seja importante para o sucesso da operação. Assim, clientes de correio eletrônico, páginas web, *peer-to-peer*, entre outros, necessitam de um protocolo confiável, e portanto utilizam o TCP.

Neste capítulo, dedicado ao TCP, iremos tratar do seu funcionamento e das ferramentas utilizadas por ele para melhorar o desempenho na transmissão de dados em uma rede heterogênea. Analisaremos os algoritmos de crescimento da janela de transmissão, bem como o controle de congestionamento e o impacto gerado quando ocorrem perdas ao longo da transmissão.

## 2.1 Uma visão geral do TCP

O TCP, abreviação de *Transmission Control Protocol*, ou Protocolo de Controle de Transmissão, em uma tradução livre, se apresenta como a melhor alternativa para transmitir dados de maneira confiável fim-a-fim na arquitetura apresentada hoje pela Internet. Para isso conta com mecanismos de garantia de entrega dos dados em sequência, bem como controle de fluxo e congestionamento para uma boa utilização do

canal de transmissão.

Para garantir a entrega em ordem, cada segmento TCP contém um número de sequência e um contador com o número de bytes de dados enviados. Através dessas informações, é possível garantir a ordenação dos pacotes na recepção e com uma velocidade que não sobrecarregue o emissor ou o receptor da transmissão. Além disso, através dos números de sequência é possível saber quais pacotes já foram recebidos com sucesso. Em caso de perdas na transmissão, pode-se detectar e reenviar os dados que não chegaram com sucesso ao seu destino.

O TCP provê um canal de comunicação entre os processos nas pontas de transmissão com confiabilidade, transmissão nas duas direções (*full-duplex*) e com continuidade (*streaming*). Os pacotes recebidos da camada de sessão são quebrados em pequenos segmentos e sobre eles são colocados os cabeçalhos TCP, e então enviados para a rede para serem entregues. Os campos do cabeçalho TCP têm informações relevantes para garantir as características colocadas aqui.

Com a finalidade de melhorar o desempenho e atender aplicações específicas, várias versões do protocolo TCP foram desenvolvidas após a sua criação. Entre elas podemos citar o TCP Tahoe (Fall & Floyd, 1996), o mais antigo, TCP Reno (Mo et al., 1999; Fall & Floyd, 1996), o mais atual, TCP Vegas (Brakmo & Peterson, 1995; Mo et al., 1999), que utiliza um algoritmo de congestionamento diferente do padrão adotado atualmente, TCP SACK (Floyd et al., 2000), que tem como característica a confirmação de pacotes específicos, entre outros. O TCP tratado nesse trabalho é o TCP Reno, que é o padrão adotado mundialmente até o presente momento.

Analisando de forma geral, podemos associar ao TCP algumas características importantes que trazem um melhor entendimento do seu funcionamento e de suas limitações.

O TCP é um protocolo que permite apenas comunicação entre duas partes, ou seja, não apresenta suporte para *broadcasting* ou *multicasting* em suas transmissões. Além disso, o TCP é orientado a conexões. Seu algoritmo apresenta estados que são sincronizados entre as duas partes em comunicação. Dessa forma, é garantido que as duas partes irão se comunicar de forma confiável. A confiabilidade do protocolo garante que haverá detecção em caso de perdas de pacotes ao longo da transmissão, bem como da chegada fora de ordem, duplicação ou corrupção dos mesmos.

Os dados transmitidos usando o TCP apresentam um padrão de *streaming*, não obedecendo à uma política rígida de delimitadores para emissão e recepção de segmentos. Dessa forma, um emissor pode lançar três segmentos e o receptor poderá receber todos os dados em apenas uma leitura. O tamanho dos blocos que circulam entre as terminações da conexão TCP, denominados segmentos, será definido pelo tamanho

máximo de segmento que o receptor consegue capturar de uma vez.

Outra característica muito importante para este trabalho é que o TCP é adaptável no que diz respeito à taxa de transmissão de dados. Dessa forma, o TCP se adapta ao que encontrar pela frente, seja uma rede congestionada ou um receptor com um canal de menor capacidade. Não há um valor pré-determinado para a taxa de transmissão, sendo que esta segue um algoritmo que explicaremos mais à frente e que controla a taxa de transmissão se baseando na capacidade do receptor e no caminho por onde os dados irão trafegar.

O serviço TCP é provido, tanto no emissor quanto no receptor, pela definição de um par (IP, porto). O IP é o endereço correspondente a aplicação do outro lado da transmissão, enquanto o porto é o local por onde a comunicação irá transmitir e receber dados.

## 2.2 Controle de Fluxo e Janela Deslizante

O TCP busca otimizar o uso do canal de transmissão de forma a tornar a eficiência o melhor possível. Se o TCP envia uma grande quantidade de dados inicialmente na rede, pode ser que cause um congestionamento levando a perdas. Caso o TCP seja conservador a ponto de não causar perda nenhuma, teremos uma subutilização do canal, onde haverá mais espaço ocioso para envio de dados. Para encontrar o ponto de equilíbrio na transmissão deve-se levar em conta as capacidades do transmissor, do receptor e da rede. Para isso, alguns mecanismos de controle de fluxo são utilizados.

O protocolo de janela deslizante é usado para auxiliar na busca pelo ponto ótimo da transmissão. O receptor limita a quantidade de dados que pode receber informando ao transmissor quanto de espaço livre tem em seu *buffer*. O emissor não poderá ter mais do que a quantidade informada pelo emissor em trânsito até que essa informação seja atualizada. Todo pacote de confirmação traz consigo o tamanho da janela anunciada do receptor atualizada.

A rede também apresenta uma quantidade de dados suportada devido a sua limitação física na capacidade de transmissão. Dessa forma, teremos uma janela de congestionamento associada a rede que será a quantidade de dados que é possível enviar sem que congestionamentos sejam provocados. O transmissor adaptará sua taxa de transmissão baseando-se nos dois limites colocados para o receptor e para a rede, sendo a taxa de transferência máxima possível o valor mínimo entre a janela anunciada pelo receptor e a janela de congestionamento.

A janela de congestionamento é o controle de fluxo imposto pelo emissor, en-

quanto a janela anunciada é controle de fluxo imposto pelo receptor. O menor valor entre eles prevalecerá de forma a não causar danos à transmissão.

O envio de dados pelo TCP é baseado na quantidade de bytes que são transmitidos/confirmados. O algoritmo de janela deslizante apresenta o comportamento a seguir (Peterson & Davie, 2007). Primeiramente o transmissor atribui um número de sequência (NumSeq), a cada segmento. O mesmo transmissor mantém o valor de três variáveis para o estado da transmissão. São elas:

- O Tamanho da Janela de Envio, ou TJE, que indica o limite superior para o número de bytes sem confirmação que podem ser transmitidos;
- Última Confirmação Recebida, ou UCR e;
- Último Segmento Enviado, ou USE.

O Tamanho da Janela de Envio, ou TJE, deve ser maior ou igual a diferença entre o último quadro enviado e a última confirmação recebida,

$$USE - UCR \leq TJE$$

A figura 2.1 mostra como a janela se move ao longo do tempo. Quando um pacote de confirmação é recebido, temos que o UCR é movido para a direita, permitindo o envio de um novo segmento. O transmissor deve manter até TJE segmentos em seu *buffer* para os casos de retransmissão por perdas. A variável USE vai definir qual o último segmento transmitido na rede.

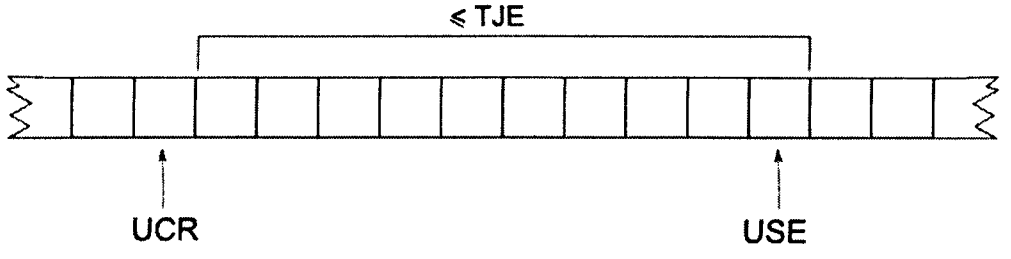


**Figura 2.1.** Janela de transmissão do TCP

O receptor apresenta também sua janela de recepção. Três variáveis são mantidas para manter o funcionamento correto desse mecanismo:

- O Tamanho da Janela de Recepção, ou TJR, com o limite superior para o número de bytes que podem ser recebidos fora de ordem;
- O número de sequência do Maior Segmento Aceitável, ou MSA e;

- O número de sequência do Último Segmento Recebido, ou USR.

O Tamanho da Janela de Recepção, ou TJR, deve ser maior que a diferença entre o MSA e o USR, ou seja,

$$MSA - USR \leq TJR$$

A figura 2.2 nos mostra como as variáveis são usadas dentro da janela. Quando um segmento chega, o receptor verifica se o NumSeq está entre USR e MSA para aceitá-lo. O valor das variáveis USR e MSA são atualizados de forma a definir o envio do pacote de confirmação.

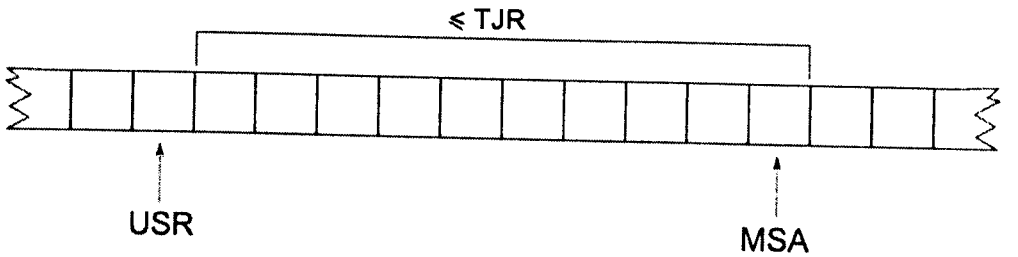


**Figura 2.2.** Janela de transmissão do TCP

Para manter essas variáveis, o TCP conta com informações transmitidas em seu cabeçalho, como veremos na seção 2.3.

Como vimos, a principal função do mecanismo de janela deslizante diz respeito ao controle de fluxo. O TCP, dessa forma, baseia o deslocamento da janela de transmissão e consequentemente o controle de fluxo na chegada de ACKs. A medida que os mesmos chegam, tanto a janela é deslocada permitindo que novos segmentos sejam enviados. A essa dinâmica dá-se o nome de *ACK-clocking*.

## 2.3 O cabeçalho do TCP

As informações contidas nos cabeçalhos de pacotes TCP são de fundamental importância para seu bom funcionamento. Como mostrado na figura 2.3, dentro do cabeçalho estão informações que são importantes para os mecanismos de janela deslizante e controle de congestionamento.

As primeiras informações contêm os portos do emissor e do receptor. Toda conexão TCP, quando aberta, causa a alocação dos portos, um no lado transmissor e outro no lado receptor da transmissão. Esses portos são a origem e destino final das comunicações por TCP. Dessa forma, para manter o caráter orientado a conexão do

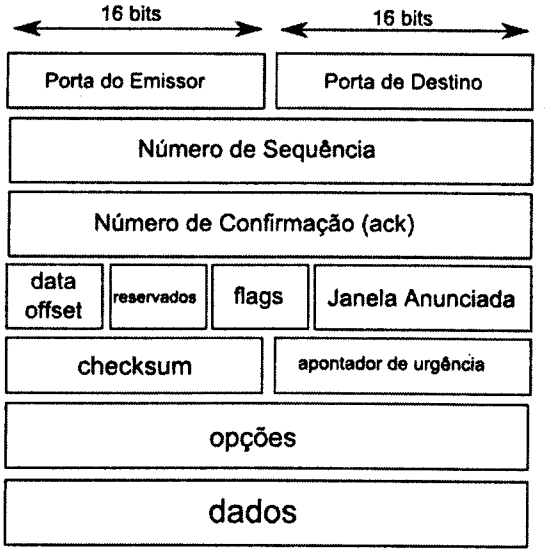


**Figura 2.3.** Cabeçalho de um pacote TCP

TCP, é importante que os portos estejam corretos no cabeçalho TCP, de forma que o dado chegue corretamente à sua aplicação destino.

O número de sequência irá informar qual a posição que o segmento ocupa na sequência de dados que foi enviada. Mais precisamente, esse é o número do primeiro *byte* do segmento.

O número de confirmação (*Acknowledgment* ou ACK) é a informação sobre o último segmento recebido na sequência correta pelo lado receptor. Essa informação é enviada na direção reversa da transmissão, ou seja, do receptor para o emissor, preenchendo-se o campo de ACK naquele sentido. O número de reconhecimento que o receptor atribui a seu segmento é o número de sequência do próximo *byte* que ele aguarda do transmissor.

Outro campo importante é o campo Janela Anunciada, que representa a janela de recepção do TCP. Neste campo será informado o espaço disponível na janela de transmissão permitida pelo controle de fluxo. Uma vez que temos um valor k para a janela, podemos enviar, sem confirmação, simultaneamente, k bytes pela rede. Se já existirem k bytes em tráfego sem confirmação, deve-se esperar a confirmação de um ou mais bytes para que outros sejam liberados para transmissão.

O campo *checksum* serve para garantir a integridade dos pacotes, de forma que se tenha certeza que nenhum bit foi corrompido ao longo do caminho.

O campo de *flags* contém vários bits que são configurados quando se tem um propósito específico para os dados, como mostra a figura 2.4. O campo URG é usado em conjunto com o apontador de urgência (figura 2.3) para indicar um conjunto de

dados que tenha urgência na entrega. Esse recurso é de interesse apenas para algumas aplicações específicas, como programas de terminal virtual, como o TELNET, por exemplo.

Em nosso trabalho, é importante ressaltar o *flag* ACK. Ele indica a existência de informações de entrega de dados no segmento em questão.

Os campos RST, SYN e FIN são usados nas etapas de estabelecimento e finalização de conexão para controle de estado no TCP. O flag PSH é usado pela aplicação transmissora para indicar que deseja que os dados enviados sejam entregues o mais rápido possivel ao receptor.

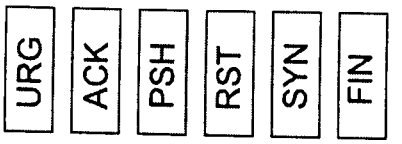


**Figura 2.4.** Flags do Cabeçalho de um pacote TCP

## 2.4 Controle da Conexão

Apesar de não ser foco deste trabalho, iremos tratar de maneira resumida do mecanismo de estabelecimento e finalização de conexão do TCP.

A fase inicial de conexão do TCP é chamado também de *three-way handshake*, algo como "negociação de três vias". Nesta etapa a operação de conexão segue o seguinte processo:

- O sistema que inicia a conexão envia à outra extremidade da comunicação um número a ser usado como número de sequência inicial usando um segmento vazio (sem dados) com o flag SYN ativado
- O sistema remoto responde com um segmento vazio com o ACK para o número inicial e com um flag SYN, com o número de sequência inicial que será usado por aquela extremidade para enviar seus dados e informa o tamanho do *buffer* do receptor
- A ponta local responde com o ACK para o novo número de sequência da outra extremidade

A partir desse processo, a conexão está aberta e os dados podem ser enviados. A figura 2.5 ilustra o processo.

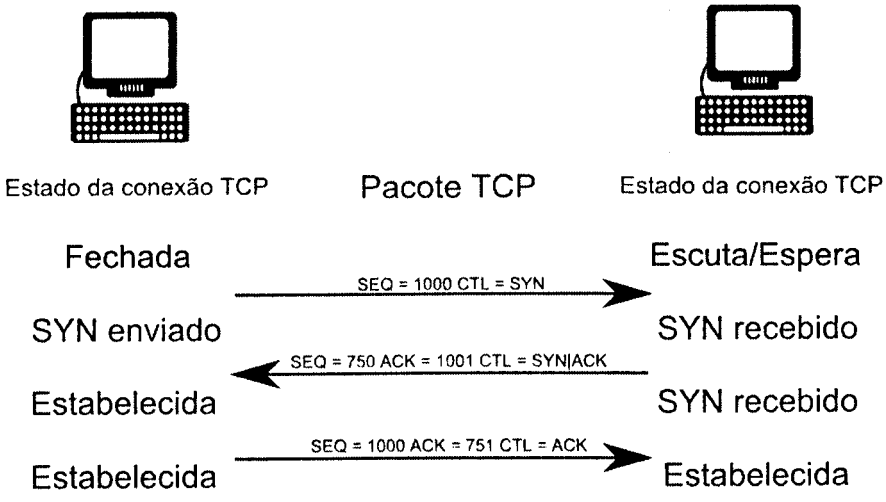


**Figura 2.5.** Handshake de três vias

Para finalizar uma conexão, as duas extremidades precisam estar cientes de que não há mais dados a transmitir. Dessa forma, não basta que um lado da conexão finalize a mesma, enquanto o outro lado continua aguardando ou enviando dados. Para isso o TCP executa as seguintes etapas:

- O lado cliente pede o fechamento da conexão enviando um segmento com o *flag* FIN para o servidor;
- O lado servidor ao receber o pedido de fechamento, envia um ACK em resposta ao FIN do cliente, e logo após um tempo, o servidor envia o seu FIN dizendo que a conexão foi fechada do seu lado;
- O lado do cliente ao receber o ACK, fecha sua conexão e envia um ACK de volta para o servidor
- Conexão encerrada quando o servidor recebe o ACK.

Dessa forma, fica garantido para os dois lados da conexão o fechamento correto das transmissões.

## 2.5 *Piggybacking*

Esse mecanismo é muito útil e utilizado em casos onde temos transmissões bidirecionais. As transmissões bidirecionais dizem respeito a duas transmissões TCP entre duas extremidades no sentido transmissor-receptor e receptor-transmissor. Dessa forma, pa-

cotes de dados são enviados nos dois sentidos, bem como pacotes de confirmação, ou ACKs. Neste caso, a técnica denominada *piggybacking* utiliza os pacotes de dados para confirmar a chegada dos segmentos recebidos da transmissão no outro sentido.

Através desse método, evita-se que pacotes sem dados sejam enviados apenas para confirmação da chegada correta de segmentos, e passa-se a aproveitar o cabeçalho dos segmentos com dados para tal função. Um exemplo de aplicação para esse mecanismo são os *peers* de uma transmissão P2P que enviam e recebem dados entre o mesmo par. Dessa forma, ao fazer *upload* de dados, uma das pontas pode confirmar dados recebidos via *download*.

## 2.6 Desempenho do TCP

O TCP procura maximizar a eficiência na transmissão de dados, buscando um ponto de equilíbrio para fazer uso da rede sem perdas de segmentos. Por isso, é importante ter conhecimento dos meios utilizados pelo TCP para encontrar o ponto ideal de operações.

### 2.6.1 Perda de Pacotes

As perdas de pacotes ocorrem em redes TCP/IP, em sua grande maioria, por causa de congestionamentos ao longo do canal de transmissão. Os roteadores são responsáveis por definir a trajetória dos pacotes que chegam a eles. Quando há uma grande quantidade de pacotes chegando, o mesmo usa uma fila para armazenar os pacotes até que seja possível transmiti-los. Quando um pacote é enviado e encontra um roteador com sobrecarga de dados para redirecionar, o mesmo pode ser descartado por falta de espaço na fila do mesmo. Outro motivo menos comum de perdas são erros nos *bits* recebidos.

A perda de um segmento TCP pode ser detectada pelo transmissor de duas formas: através do mecanismo de ACKs duplicados, ou através da estimativa de RTT da rede, que vai criar um tempo esperado para a chegada do segmento ao seu destino e da chegada do seu ACK correspondente de volta.

Quando apenas um pacote em meio a vários é perdido, temos que os demais chegarão ao destino e farão com que o receptor gere ACKs duplicados para cada pacote que tem uma sequência maior que o pacote perdido. Isso se deve ao fato de que o receptor da transmissão sempre enviará um ACK com o número do último byte recebido na ordem. Quando os segmentos fora de ordem chegam, o receptor gera ACKs para o próximo byte na sequência, que acaba sendo duplicado. A chegada dos ACKs duplicados é usada como sinal para o emissor de que ocorreu uma perda. Para

reduzir as chances de falsos positivos nesse processo, por segmentos que possam apenas ter se atrasado ou chegado fora de ordem, o TCP exige que haja três ACKs duplicados antes de considerar que o pacote se perdeu.

A segunda forma de detectar a perda de um pacote é aguardando o fim do tempo de espera para o ACK (*timeout*) do dado enviado. Quando esse tempo se esgotar, automaticamente o emissor reenvia o pacote que não foi confirmado até o momento esperado. Esse tempo é estimado com base no tempo de ida e volta da comunicação (*round-trip time* - RTT) calculado ao longo da conexão a cada momento. Essa variável é calculada levando-se em consideração o tempo que um pacote leva para ir ao seu destino e ter a sua confirmação recebida de volta pelo emissor.

Para prever um tempo de vida do pacote adequado com a situação atual da rede, é importante estimar bem o RTT, de forma que não seja curto demais, o que pode gerar retransmissões desnecessárias, nem longo demais, o que geraria esperas muito longas até que um pacote perdido seja recolocado no canal de transmissão. Na versão atual mais comum de implementação, usa-se uma média amortecida das medidas de RTT acrescida de um fator para se considerar a variância dessas medidas.

### 2.6.2 Controle de Taxa de Transmissão (*Throughput*)

Como discutido na seção 2.2, o TCP deve ajustar sua taxa de transmissão seguindo os limites impostos pela rede de forma a equilibrar a taxa de perdas e otimizar o envio de dados, aproveitando melhor o canal de transmissão. Aumentar a taxa de transmissão ilimitadamente geraria um congestionamento na rede e rapidamente causaria perdas de pacotes ao longo do canal. A retransmissão de pacotes, por sua vez, forçada pelas perdas, causa queda na eficiência de transferência dos dados. Por outro lado, garantir que não há perdas no canal obrigaria o TCP a reduzir a taxa de transmissão à um valor tão baixo que poderia tornar o canal mal aproveitado, com uma taxa de transmissão que não o utiliza por completo.

Para encontrar um meio termo nesse processo, o TCP tenta garantir que todo segmento sai do emissor ao mesmo tempo em que outro é absorvido pelo receptor. Dessa forma, o canal não estará inutilizado nem sobrecarregado. O desafio no desenvolvimento desse mecanismo está na dificuldade em se encontrar a taxa correta para as duas extremidades da transmissão.

A taxa de transmissão do TCP será definida pelo protocolo de janela deslizante, onde é levado em conta a capacidade do receptor da transmissão, e pela janela de congestionamento definida pelo emissor se baseando no estado da rede.

Encontrar essa taxa é um outro problema que promoverá um controle do tráfego

adequado ao canal de transmissão, bem como uma justiça entre várias conexões, onde cada uma irá ocupar apenas uma parte da banda sem "roubar" banda de outras conexões.

Dois modos para transmissão são usados pelo TCP para encontrar essa taxa sem causar perdas excessivas. Estes modos são a Partida Lenta e o *Congestion Avoidance*[1], métodos que apresentaremos a seguir.

#### 2.6.2.1 Partida Lenta

Imaginemos se o TCP, ao abrir uma conexão, enviasse o máximo de bytes que sua janela suporte (ou seja, o tamanho do seu *buffer* em dados) de uma única só vez. Essa estratégia não levaria em consideração o estado da rede, bem como descarta os limites que o receptor venha a ter para coletar todos os bytes sem perdas.

Para evitar esse problema, o TCP adota uma estratégia mais conservadora, tentando evitar congestionamentos e perdas ao longo da rede já no início da conexão. Quando começa sua transmissão, o emissor envia apenas o mínimo de bytes possível. À medida que vai percebendo a resposta da rede, aumenta a quantidade de bytes em trânsito na rede.

No início, como são poucos bytes, a chance de sucesso é grande. Com o aumento da quantidade de bytes enviados, chega um momento onde poderá haver perdas. Nesse instante a taxa de emissão é reduzida, bem como o valor da janela de transmissão.

Uma unidade importante utilizada pelo TCP para definir o tamanho dos segmentos é o MSS (*Maximum Segment Size*). O MSS é o tamanho em bytes do maior segmento que o transmissor pode enviar.

Quando uma conexão é estabelecida, durante a fase de *three-way handshake* é definido o tamanho da janela para o receptor. Alguns estudos mostram que um valor inicial de dois segmentos é mais confiável para a média dos casos (Allman et al., 2002). A partir daí, o TCP emissor começa a enviar segmentos e fica aguardando a confirmação dos mesmos. Quando um segmento é confirmado, o TCP aumenta a sua janela de transmissão em 1 segmento e envia dois segmentos. Agora, o TCP terá que aguardar a confirmação destes novos segmentos transmitidos.Quando cada um destes dois segmentos for confirmado, a janela de congestionamento é aumentada para quatro. Isto provê uma aumento exponencial. Essa política recebe o nome de Partida Lenta.

Um parâmetro usado para controlar a fase de Partida Lenta é o *sstresh*. Esse parâmetro recebe, inicialmente, o valor de 65536 bytes, e serve para definir se o TCP

[1] O termo *Congestion Avoidance* não apresenta uma tradução padrão para o português, logo, preferimos manter o nome em inglês. Poderíamos ter usado como tradução algo como Prevenção de Congestionamento.

está na fase de Partida Lenta ou de *Congestion Avoidance*. Quando esse valor é ultrapassado, o TCP emissor passa para a fase de *Congestion Avoidance*. Quando ocorre alguma perda por *timeout*, o valor de *sstresh* é alterado para a metade do valor da janela de transmissão atual. A perda também faz com que a janela de congestionamento seja reduzida.

#### 2.6.2.2 Congestion Avoidance

A taxa ideal de transmissão pode ser maior ou menor do que a atual. Para encontrar essa taxa ideal, é preciso um mecanismo que se adapte e busque a otimização no envio de dados. Dessa forma, se a taxa ideal é maior do que a praticada, ou seja, não temos perdas com a taxa atual, devemos aumentar a mesma (aditivamente). Caso a taxa ideal seja menor, ou seja, a rede tem congestionamento, é importante reduzir rapidamente a mesma a fim de evitar um colapso na rede.

Para controlar esse mecanismo no emissor, foi criada a janela de congestionamento para o TCP, comumente conhecida como *cwnd*. A janela de congestionamento, também conhecida como cwnd, inicia com poucos segmentos, sendo incrementada à medida que as confirmações são recebidas sem perdas ao longo do canal (a perda indica que o canal foi saturado).

Quando ocorre perda, a janela de congestionamento é alterada para a metade do valor atual, e novamente é acrescida à medida que novas confirmações são recebidas. Dessa forma, é garantida uma taxa ótima (Jacobson, 1988) à medida que a janela de transmissão oscila entre o valor ótimo da rede, além de perceber alterações na capacidade dos canais para baixo ou para cima da banda disponível no momento.

Assim, o *Congestion Avoidance* tenta prever o ponto de estrangulamento na rede com um crescimento linear da taxa de envio, ao contrário da fase de Partida Lenta, onde há um crescimento exponencial. Assim como a Partida Lenta, essa fase tenta encontrar o ponto onde ocorre congestionamento na rede. Para dividir os dois estados, usa-se um parâmetro definido como *sstresh*. Esse parâmetro irá ser o ponto onde finaliza-se a fase de Partida Lenta e começa a fase de Congestion Avoidance.

Nesse ponto, a janela de congestionamento *cwnd* é incrementado em

$$MSS \times \frac{MSS}{cwnd} \tag{2.1}$$

em resposta a cada ACK não duplicado recebido. Assim, a cada janela completa enviada com sucesso teremos a janela com um crescimento de um MSS. Esse crescimento irá ocorrer até a perda de um pacote.

### 2.6.3 Outras técnicas para detecção e controle de congestionamento

Além dos mecanismos próprios do TCP para controle de congestionamento, algumas técnicas foram criadas com a intenção de melhorar a detecção de congestionamentos ao longo da rede em transmissões TCP de forma antecipada. Esses mecanismos exigem a participação dos roteadores ao longo da conexão, que devem simplesmente implementar as técnicas a seguir:

#### 2.6.3.1 RED

O RED (Floyd & Jacobson, 1993; Lin & Morris, 1997)— *Random Early Detection* — permite à rede detectar o início dos congestionamentos antes que eles piorem através dos roteadores (Floyd & Fall, 1997). Um roteador usando o RED pode descartar um pacote mesmo que a rede ainda não esteja congestionada, de forma a prever um possível congestionamento caso as conexões TCP continuem a manter ou aumentar o fluxo de transmissão.

Uma linha de probabilidade baseada no espaço ocupado no *buffer* do roteador define quais as chances de um pacote ser descartado. À medida que o *buffer* do roteador vai sendo ocupado, aumentam as chances de um pacote ser descartado baseado em uma probabilidade que segue a função ilustrada na figura 2.6.

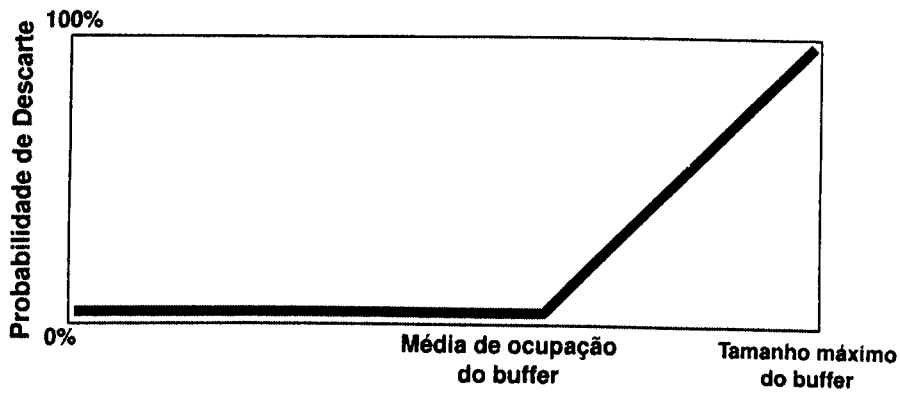


**Figura 2.6.** Probabilidade de um pacote ser descartado no RED

Observe que a partir de um limiar a curva de probabilidade começa a crescer até onde o *buffer* se torna completamente cheio, e aí há 100% de chances de descarte. O efeito colateral desejado nesse ambiente é que essas perdas forcem a diminuição da janela de transmissão de conexões TCP antes que a rede se sobrecarregue, o que levaria todos os fluxos a perderem pacotes ao mesmo tempo. Isso levaria à uma diminuição

em massa de todas as janelas de transmissão de todos os fluxos, fazendo com que todas as conexões TCP perdessem desempenho.

Com a técnica RED, os fluxos que estiverem enviando mais dados terão uma chance maior de perder um pacote antecipadamente, enquanto os fluxos menores poderão continuar normalmente sem perdas (Floyd & Jacobson, 1993).

O descarte aleatório do RED também tem como objetivo reduzir a ocorrência de temporizações em TCP. Como a perda aleatória ocorre antes do congestionamento, há mais chances dela ser detectada com a retransmissão rápida através de ACKs duplicados, sem exigir que a conexão pare até uma temporização.

Essa técnica exige alterações no meio da rede, nos roteadores, sendo que os clientes não sofrem alterações.

#### 2.6.3.2 ECN

O algoritmo ECN (*Explicit Congestion Notification*), desenvolvido por Floyd (1994), é uma alternativa ao RED que não descarta pacotes aleatoriamente (Karandikar et al., 2000). Ao invés disso, ele simplesmente "comunica" ao emissor que existe um potencial de congestionamento na fila do roteador alterando algum bit em um dos cabeçalhos dos segmentos presentes no fluxo. Esses segmentos chegam ao destino, que pode então repassar essa notificação ao transmissor. Quando o transmissor recebe a notificação através dessa alteração, ele responde como se tivesse perdido um segmento.

Para isso, são usados dois bits extras (até então desocupados) do cabeçalho TCP. O primeiro deles indica se o emissor da transmissão tem capacidade de reagir a um sinal de ECN. Esse é o bit ECT. O outro bit é o CE (*Congestion Experienced*), que é alterado pelo roteador caso o mesmo perceba que há possibilidade de congestionamento. Ele irá perceber isso da mesma forma que o RED faria, colocando um limiar no seu tamanho da fila do *buffer* que delimite o risco de congestionar.

Somente segmentos com o bit ECT configurado poderão ser usados pelo ECN. Para isso, na fase de abertura de conexão, o segmento inicial TCP SYN inclui os bits de ECN-echo e CWR (*Congestin Window Reduced*) — que é usado para dizer que houve uma resposta por parte do transmissor em relação a um indício de congestionamento.

Esse procedimento faz com que emissor e receptor negociem como será tratado o bit CE, caso este surja ao longo da transmissão. Daí em diante todos os segmentos enviados serão marcados no bit ECT. Quando o receptor recebe um segmento com o ECN alterado pelo roteador, envia a informação de que o canal está na iminência de congestionar através do ACK, ou seja, ele repassa a informação ao transmissor da conexão. Ao pacote ACK com o ECN de resposta por parte do receptor dá-se o nome

de ENC-*echo*. Caso o emissor receba um ECN-*echo*, ou seja, um pacote de confirmação com a informação do bit CE habilitado, este irá reduzir sua janela de congestionamento como se houvesse perdido um pacote.

O bit CWR é enviado no próximo segmento para que o receptor e a rede saibam que o emissor reagiu perante o possível congestionamento. Isso irá garantir que vários segmentos com ECN-*echo* não possam atuar sobre o emissor em um mesmo RTT, o que reduziria além do necessário a janela de transmissão. Dessa forma, o receptor irá configurar os bits de ECN-*echo* para todos os segmentos até receber o CWR do emissor.

Esse mecanismo, além de exigir alterações para os roteadores da rede, exige que os pares que estão se comunicando também atuem no processo de detectar com antecedência um possível congestionamento. Alguns sistemas operacionais, como Linux, Mac OS X e FreeBSD já têm a tecnologia do ECN habilitada no TCP. Os sistemas Windows Vista e Windows Server 2008 também apresentam o ECN desabilitado por padrão.

# Capítulo 3

# Modelo Analítico

Neste capítulo analisaremos o comportamento de uma rede assimétrica quanto ao desempenho do canal de *download* em relação ao uso do canal reverso. Para isso, modelamos matematicamente uma rede com bandas assimétricas transmitindo dados nos dois sentidos. Não levamos em consideração o atraso nas transmissões, bem como desconsideramos os efeitos do crescimento das janelas de transmissão, de modo a simplificar o modelo. Dessa forma, transmissões com curta duração, como requisições via Web, não podem ser atendidos por esse modelo, bem como topologias que apresentem outros tipos de assimetria que não de banda. Estas suposições são embasadas nas redes utilizadas atualmente pela grande maioria dos usuários da Internet.

Quando dados são enviados nos dois sentidos da transmissão, temos pacotes com dados e pacotes de confirmação disputando o mesmo canal e, consequentemente, consumindo parte da banda. Quando lidamos com o canal reverso (*uplink*), consideramos que os dados tem preferência ao serem enviados em detrimento dos ACKs. Assim, temos que a ocupação do canal é dada pela quantidade de dados que está sendo transmitida em relação à capacidade total do mesmo.

Nesse modelo, levamos em consideração que cada ACK irá gerar um pacote de dados em resposta. Consideramos também que podemos controlar a fração do canal reverso utilizada para envio dos dados. Essa suposição se baseia no fato de que as aplicações atuais que demandam tráfego nos dois sentidos deixam a cargo do cliente configurar o valor da banda de *upload* que será dedicada ao envio de dados. Um exemplo claro desse fato são os sistemas P2P de *bittorrent*.

O primeiro caso para o qual apresentamos o comportamento tem duas transmissões unidirecionais em sentidos opostos, enquanto o segundo caso trata de uma transmissão bidirecional. Exemplos desses casos foram mostrados e discutidos no capítulo 1. Quando o tráfego é bidirecional, consideramos que o mecanismo de *piggybacking*

está ativo e interfere no resultado final.

Os valores utilizados para as bandas e tamanhos dos pacotes nesse trabalho atendem aos parâmetros mais atuais. Como dito anteriormente, alguns trabalhos mais antigos mostram resultados em situações diferentes do que vemos hoje. Um exemplo são as redes via satélite, onde os pacotes de confirmação geralmente eram retornados ao destino por uma rede telefônica, e não pelo mesmo canal de *download*. A assimetria era então mais agravada pela diferença nos atrasos de transmissão nos dois sentidos. Atualmente não há assimetria de atrasos no envio de dados, uma vez que os dados e os ACKs são transmitidos através do mesmo canal.

Buscamos também valores dentro do que é fornecido atualmente pelas empresas de Internet, como demonstrado na tabela 3.1 (Speedy[1], Net Virtua[2], GVT, Telefônica e Oi Velox[3], ). Quanto ao tamanho dos pacotes desconsideramos também o uso de compressões (SLIP, por exemplo). Com o aumento nas velocidades das bandas e com uma diferença maior entre pacotes de dados e ACKs (cerca de 1 para 25) não é justificável o uso de compressão nos pacotes.

**Tabela 3.1.** Alguns valores de banda fornecidos por empresas de Internet no Brasil

| *Plano/Empresa* | *Download* | *Upload* |
|---|---|---|
| 8 MB - Telefônica e TVA | 8Mbps | 1Mbps |
| 1 MB - Speedy | 1Mbps | 300Kbps |
| 3MB - GVT | 3Mbps | 750Kbps |
| 2MB - Net Virtua | 2Mbps | 300Kbps |
| 4MB - Oi Velox | 4Mbps | 500Kbps |
| 1MB - Oi Velox | 1Mbps | 300Kbps |

A tabela 3.2 mostra as variáveis e notações utilizadas para os modelos.

Já é possível limitar a banda de *upload*, bem como escalonar pacotes com prioridades através de várias ferramentas criadas em alguns sistemas operacionais e softwares. O Netem (Hemminger, 2005) é um exemplo no sistema operacional Linux, e as ferramentas P2P geralmente vem com módulos para regular as bandas de *download* e *upload*. Utilizando essas ferramentas, podemos definir nosso modelo em função da variável $\beta$, que nada mais é do que a fração do canal de *upload* que será dedicada ao envio de dados. Partindo daí, pode-se afirmar que $1 - \beta$ será a fração ociosa do canal

[1]http://www.insite.com.br/speedy visitado em 08 de setembro de 2009 às 20hs

[2]http://www.thecnica.com/dicas/texto.asp?src=a&Cod=131 acessado em 08 de setembro de 2009 às 20hs

[3]Informações recebida via atendimento telefônico direto com a operadora

**Tabela 3.2.** Notações e variáveis do modelo.

| *Notações* | *Definiçoes* |
|---|---|
| D | Banda de *Download* do canal, em bps |
| U | Banda de *Upload* do canal, em bps |
| $\alpha$ | Razão entre *upload* e *download*: $\frac{U}{D}$ |
| $\beta$ | Fração do canal de *upload* reservado para dados |
| $b_d$ | Taxa de *download* utilizada, em bps |
| $b_u$ | Taxa de *upload* utilizada, em bps |
| $p_A$ | Tamanho dos pacotes de confirmação, ou ACKs |
| $p_d$ | Tamanho dos pacotes de dados no *downlink* |
| $p_u$ | Tamanho dos pacotes de dados no *uplink* |

reverso que poderá ser utilizado para transmissão de ACKs. O valor de $\beta$ pode ser controlado pelo usuário da aplicação que acessa a rede.

O valor de $\alpha$ é a fração entre as bandas de *upload* e *download*, ou seja, o quanto o *uplink* é menor do que o *downlink*.

D e U definem a capacidade máxima de transmissão nos canais de *download* e *upload*, respectivamente. Na tabela 3.1 podemos visualizar algumas taxas comumente usadas por empresas de Internet atualmente.

$b_d$ e $b_u$ são as bandas de *download* e *upload* resultantes das conexões nos dois sentidos, respectivamente. É trivial de se perceber que $b_d \leq D$ e $b_u \leq U$. Os tamanhos dos pacotes de dados e de ACKs também seguem os valores mais utilizados atualmente pelo TCP, e são definidos por $p_d$ e $p_u$, como sendo o tamanho dos pacotes de dados no canal de *download* e no canal reverso, respectivamente. $p_A$ é o tamanho do ACK da transmissão.

A partir dessas variáveis, partiremos para a modelagem, que será dividida em dois modelos distintos, com duas transmissões unidirecionais e com uma transmissão bidirecional.

## 3.1 Duas transmissões unidirecionais

Passemos ao primeiro modelo, que é ilustrado pela figura 3.1. Para calcular o grau de assimetria entre os canais, utilizaremos a razão entre a capacidade de *upload* e *download* como sendo

$$\alpha = U/D \text{ ou } U = \alpha D. \tag{3.1}$$

O tamanho dos pacotes são $p_A$ (ACKs), $p_d$ (pacotes de dados no canal principal)

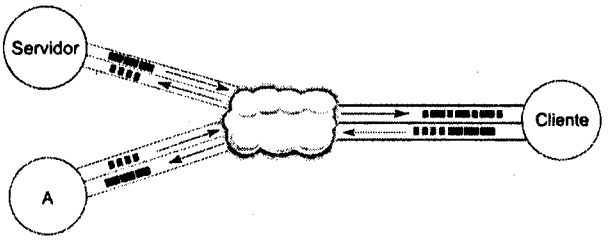


**Figura 3.1.** Transmissão do cliente com dois pontos diferentes da rede

e $p_u$ (pacotes de dados no canal reverso), conforme já anunciado anteriormente, onde $0 \leq p_u \leq p_d$. Atualmente, o TCP utiliza $p_u = p_d$, mas consideramos importante desmembrar esses valores na expectativa que a mudança no tamanho do pacote de dados no canal reverso possa representar alguma melhora para o desempenho da conexão como um todo.

Como dito anteriormente, os pacotes de dados terão prioridade sobre os ACKs no escalonamento de pacotes na saída do canal reverso. Denotamos como $\beta$ a ocupação desejada pelo usuário da rede no canal reverso. Para esse controle podemos utilizar ferramentas complementares ao sistema operacional ou mesmo aplicações próprias que façam o controle do valor de $\beta$. A taxa de utilização do canal reverso, dessa forma, não depende de outros fatores. Assim, a banda de *upload* será dada por:

$$b_u = \beta \cdot U = \alpha \cdot \beta \cdot D. \tag{3.2}$$

Para obter a taxa de transmissão no canal reverso em termos de pacotes/segundo, teremos que:

$$T_u = \frac{b_u}{p_u} = \frac{\alpha \cdot \beta \cdot D}{p_u}. \tag{3.3}$$

Para calcularmos a taxa de *download* possível ($b_d$) alcançada, temos que levar em consideração a banda consumida pelos ACKs da transmissão reversa. Quando o canal reverso faz *upload* de dados para a outra ponta, temos que os ACKs irão consumir uma parte da banda de *download*. Através de análises considerando os valores para as tecnologias atuais verificamos que esse valor pode ser desprezado para os propósitos deste trabalho, uma vez que não altera significativamente a capacidade do canal de *download*. Se considerarmos uma conexão Oi - Velox de 1MB, como demonstrado na tabela 3.1, com pacotes de dados de 1500 bytes, e ACKs de 60 bytes, teríamos que uma utilização máxima do canal de *upload* levaria a transmissão de 25 pacotes de dados por segundo. Em contrapartida, isso faria com que 12Kbps do canal de *download* fossem utilizados para o envio de ACKs, ou seja, apenas 1,2% do canal.

A taxa de pacotes no canal de *download* pode, então, ser calculada por:

$$T_d = min\left(\frac{D}{p_d}, \frac{U - b_u}{p_A}\right) = min\left(\frac{D}{p_d}, \frac{\alpha D(1-\beta)}{p_A}\right). \tag{3.4}$$

Dessa forma, a taxa de pacotes no canal principal será dado pelo valor mínimo entre a máxima taxa possível de *download* e a taxa permitida pela banda restante no *uplink* para o tráfego de ACKs ($U - b_u$).

A banda disponível será dada pela taxa de pacotes que são permitidos no canal de *download* pelo tamanho dos pacotes de dados:

$$b_d = T_d \cdot p_d \Rightarrow b_d = f(D, \alpha, \beta). \tag{3.5}$$

Analisando a equação 3.4, vemos que o valor de $\frac{D}{p_d}$ será menor do que o valor de $\frac{\alpha D(1-\beta)}{p_A}$ até que o canal de *upload* seja utilizado de tal forma que não reste mais banda para envio de todos as confirmações para os pacotes que chegam pelo canal de *download*. Logo, temos que o ponto onde se inicia a queda no desempenho do *download* é dado por:

$$\frac{D}{p_d} = \frac{U - b_u}{p_A} \Rightarrow U - b_u = D \cdot \frac{p_A}{p_d} \Rightarrow b_u = U - D \cdot \frac{p_A}{p_d}. \tag{3.6}$$

Usamos no nosso modelo uma variável criada por Balakrishnan et al. (1997), chamada de Constante de Assimetria em uma rede assimétrica. Esse valor é definido como $k$, e é dado por

$$k = \frac{D/U}{p_d/p_A}. \tag{3.7}$$

Fizemos com que o valor de $k$ seja calculado em função da razão definida por $\alpha$. Assim,

$$k(\alpha) = \frac{p_A}{\alpha p_d} \tag{3.8}$$

Dessa forma, podemos calcular o valor de $\beta$ onde a taxa de *download* começará a ser afetada em função de $k$, partindo da equação 3.6:

$$\frac{b_u}{U} = 1 - \frac{D}{U} \times \frac{p_A}{p_d} \Rightarrow \beta = 1 - k. \tag{3.9}$$

O gráfico da figura 3.2 mostra a variação da taxa de download para algumas bandas mais comuns na atualidade. Com a equação 3.9 podemos constatar que o ponto de quebra na curva para o modelo apresentado varia de acordo com o grau de

assimetria entre as bandas do canal para tamanhos fixos de pacotes de dados e ACKs.

Para selecionar as bandas a serem utilizadas na figura 3.2, levamos em conta informações da ABUSAR[4] e as demais informações contidas na tabela 3.1. Os valores compreendem os limites de 256kbps a 8Mbps para o canal de *download* e de até 1Mbps no canal de *upload*.

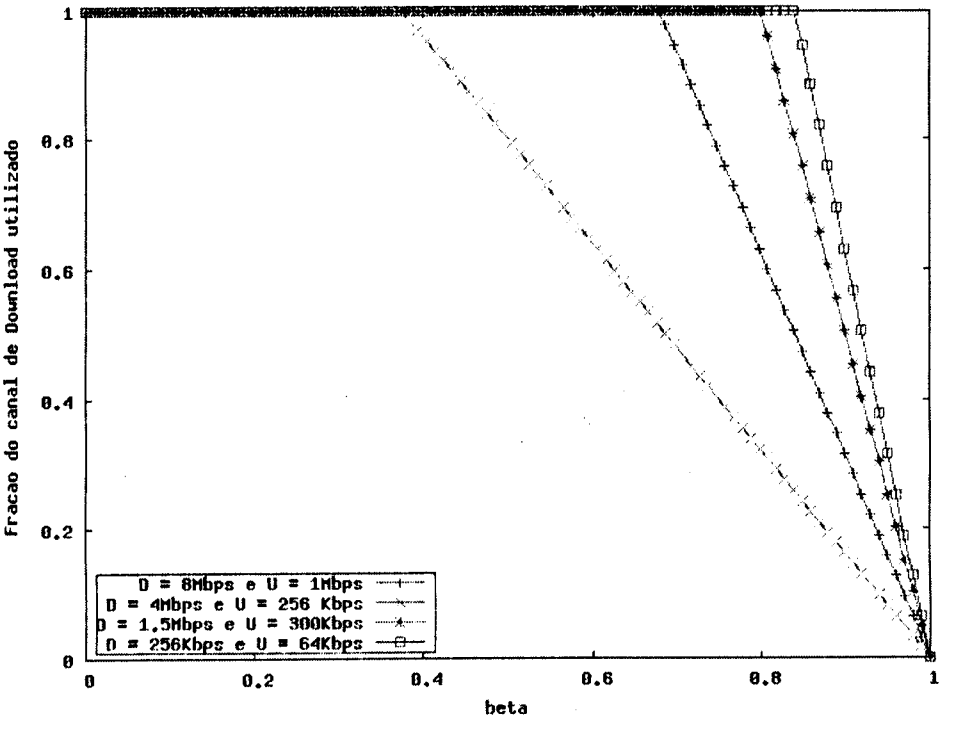


**Figura 3.2.** Transmissão do cliente com dois pontos da rede

Como exemplo, tomemos uma rede de 1,5Mbps por 300Kbps, com pacotes de dados com tamanho de 1500 bytes e pacotes de confirmação de 60 bytes. Nesse caso, temos que $k = \frac{1500}{300}/\frac{1500}{60} = 0,2$, ou seja, quando tivermos $1-k = 0,8$ (ou 80%) do canal reverso utilizado, começaremos a observar perdas na taxa de *download*. Isso pode ser confirmado pelo gráfico da figura 3.2, onde $p_A$ e $p_d$ apresentam estes valores.

A figura do gráfico 3.3 nos mostra a influência da assimetria entre as bandas de *download* e *upload* em relação ao valor de $\beta$. Consideramos os pacotes de dados com 1500 bytes, 1000 bytes e 5000 bytes, e os pacotes de confirmação, ou ACKs, com 60 bytes.

É possível visualizar que a redução na assimetria entre as bandas de *download* e *upload* ($\alpha$ maior) faz com que o canal reverso possa ter uma maior utilização antes de

[4]http://www.abusar.org/adsl.html acessado em 24 de julho de 2009

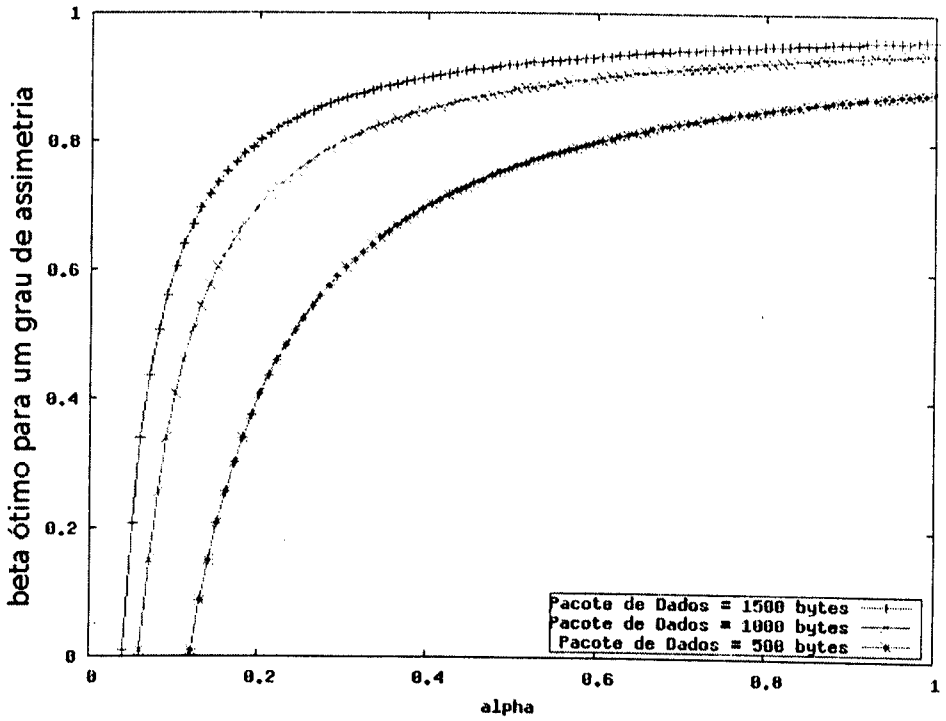


**Figura 3.3.** Utilização máxima do canal reverso ($\beta$) que pode ser utilizado sem prejudicar o *download* para um dado $\alpha$

saturar e interferir na taxa de *download* da transmissão. Também é possível observar, através do gráfico 3.3, que a redução no tamanho do pacote de dados reduz a tolerância no uso do canal reverso. Dessa forma, para pacotes menores, temos um limite de $\beta$ reduzido, logo, a fração do canal reverso que pode ser utilizada para dados irá interferir na taxa de *download* mais cedo do que quando são utilizados pacotes maiores.

## 3.2 Uma transmissão bidirecional

Agora consideramos duas transmissões entre os mesmos nós em uma mesma conexão TCP, do emissor para o receptor e vice-versa. A transmissão é bidirecional, ou seja, temos dois pontos na rede enviando e recebendo dados pelo mesmo canal. É possível utilizar o mecanismo de *piggybacking* para confirmar os pacotes de dados do canal de *download*, bem como de *upload*. Também consideramos que um ACK é gerado por pacote. A figura 3.4 ilustra uma transmissão bidirecional.

Como usamos o *piggybacking*, temos que os segmentos de dados enviados pelo canal reverso serão utilizados para confirmar os segmentos que chegam pelo canal de *download*. Dessa forma, a taxa máxima de transmissão de pacotes pelo canal mais


**Figura 3.4.** Transmissão bidirecional entre cliente e servidor

rápido será definido pelo canal que saturar primeiro:

$$T_d = min\left(\frac{D}{p_d}, \frac{U}{p_u}\right) = min\left(\frac{D}{p_d}, \frac{\alpha D}{p_u}\right). \tag{3.10}$$

As taxas alcançadas no canal de *upload* e *download* são, respectivamente,

$$\begin{aligned} b_u &= T_u \times p_u; \\ b_d &= T_d \times p_d \end{aligned} \tag{3.11}$$

sendo a taxa de *download* uma função de D, $\alpha$ e $p_u$.

Usualmente, $p_d = p_u$, mas para determinados fins é plausível alterar o valor de $p_u$ de forma a melhorar o desempenho da conexão, como veremos na seção 4.2.4.

Observe que neste caso consideramos que todo o canal de *upload* é utilizado. Assim, através do gráfico da figura 3.5 é possível observar como a redução nos tamanhos dos pacotes de dados no canal reverso podem alterar a taxa de *download* para diferentes valores de $\alpha$. Para um $\alpha$ menor, temos uma maior diferença entre a banda de *upload* e de *download*, o que resulta em uma vazão menor para o *download*. É possível também observar que a redução do tamanho dos pacotes no canal reverso aumenta consideravelmente a taxa de *download* devido à maior quantidade de ACKs que são suportados no canal reverso.

Agora consideremos que apenas uma fração do canal reverso seja utilizado para os pacotes de dados. Digamos que o canal de *upload* esteja com seu uso limitado em $\beta$, que é a fração do canal a ser dedicada para envio de pacotes de dados. Dessa forma, teremos que a taxa de pacotes de dados no canal reverso será dada por

$$T_{up} = min\left(\frac{D}{p_d}, \frac{\beta \cdot U}{p_u}\right) = min\left(\frac{D}{p_d}, \frac{\alpha\beta D}{p_u}\right). \tag{3.12}$$

Com essa informação é possível definir a taxa de *download* alcançada levando-se em conta que o restante da banda de *upload* será utilizada para os ACKs serem enviados, ou seja,

$$T_d = min\left(\frac{D}{p_d}, T_{up} + \frac{(1-\beta)\alpha D}{p_A}\right). \tag{3.13}$$

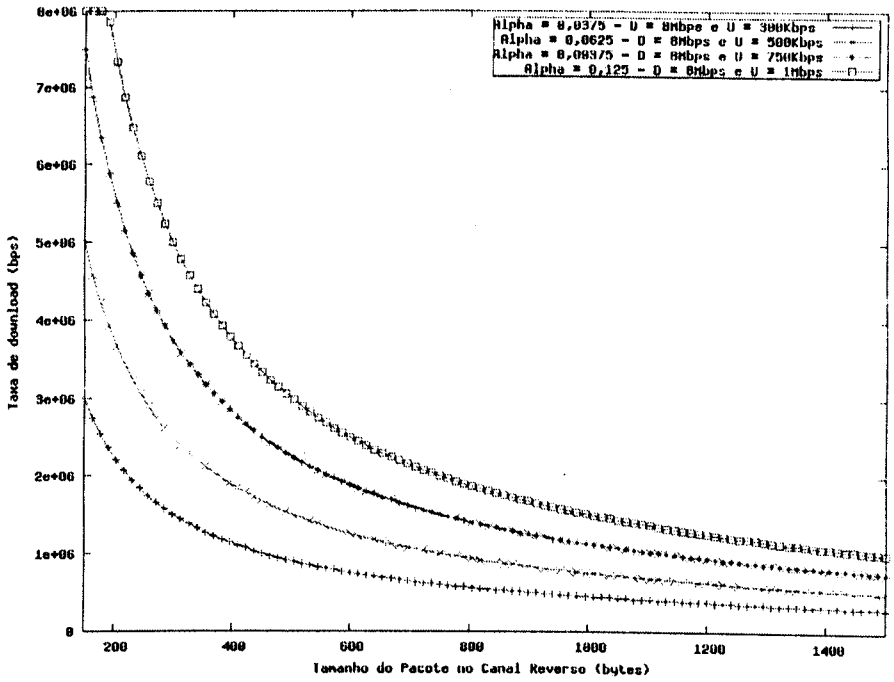


**Figura 3.5.** Piggybacking sendo utilizado para enviar confirmações no canal reverso - Pacotes de dados no canal principal com 1500 bytes de tamanho, pacotes de dados no canal reverso variando entre 150 bytes e 1500 bytes, banda de download de 8Mbps

Temos que a taxa de *download* máxima será dada por:

$$b_d = T_d \times p_d. \tag{3.14}$$

O gráfico da figura 3.6 nos mostra como se comportam algumas redes assimétricas quando levada em consideração uma transmissão bidirecional com o uso de *piggybacking*.

É importante observar o ponto de quebra da curva onde o desempenho do *download* começa a ser deteriorado. Para isso, avaliamos o caso onde a taxa de pacotes que saem do canal de *upload* acrescidos da quantidade de ACKs possíveis no restante de banda se torna menor do que a capacidade máxima do canal de *download*, ou seja,

$$T_{up} + \frac{(1-\beta)\alpha D}{p_A} < \frac{D}{p_d}. \tag{3.15}$$

Partindo daí, temos que a fração da banda de *upload* que irá saturar o canal de *upload* e reduzir o desempenho do *download* é dado por:

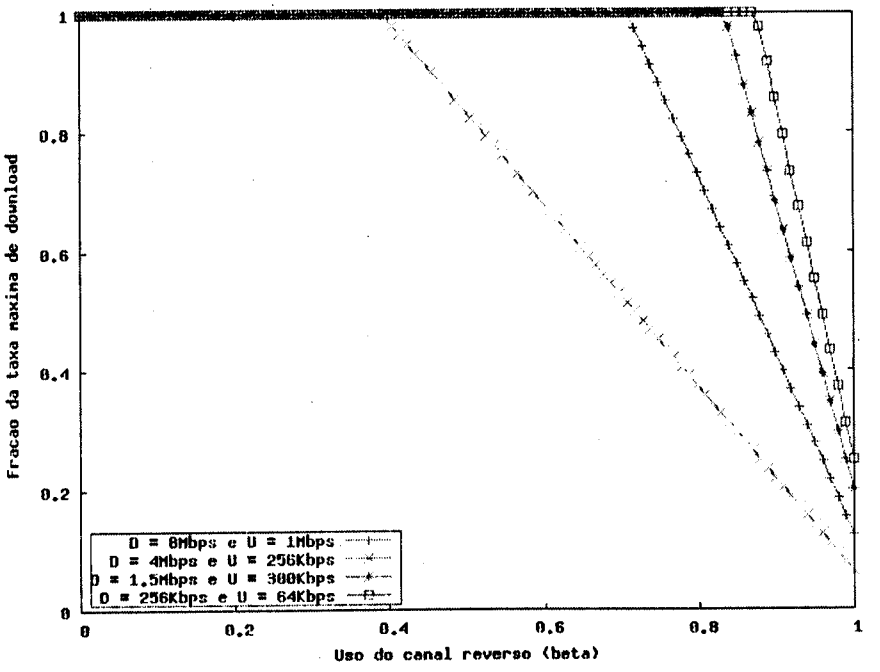


**Figura 3.6.** *Piggybacking* sendo utilizado para enviar confirmações no canal reverso

$$\beta = \frac{1}{\alpha}\left(\frac{p_A - \alpha p_d}{p_A - p_d}\right). \tag{3.16}$$

Como observado no gráfico da figura 3.6, com valores de $p_A$ e $p_d$ de 60 e 1500 bytes (os pacotes de dados nos dois canais são iguais para este caso) e as bandas ali demonstradas, é possível chegar aos valores de quebra na curva. Para a transmissão de 8Mbps/1Mbps, por exemplo, temos que o canal de upload será saturado quando $\beta = 0,407$, e para a transmissão de 4Mbps/256Kbps, quando $\beta = 0,708$, o que é claramente observado no gráfico em questão.

Assim, é possível prever quando o *uplink* será saturado. Caso tenhamos alterações nos tamanhos dos pacotes do canal reverso, teremos que o valor de saturação é

$$\beta = \frac{p_u p_A - \alpha p_u p_d}{\alpha p_d (p_A - p_u)}. \tag{3.17}$$

Variando-se o tamanho dos pacotes do canal reverso, podemos observar as variações ocorridas em relação ao ponto de saturação do canal de *upload* através do gráfico da figura 3.7. É possível observar que a redução dos pacotes faz com que o valor de $\beta$ seja melhor para ambientes com maior assimetria entre as bandas, hipótese levantada em outros estudos no mesmo campo (?).

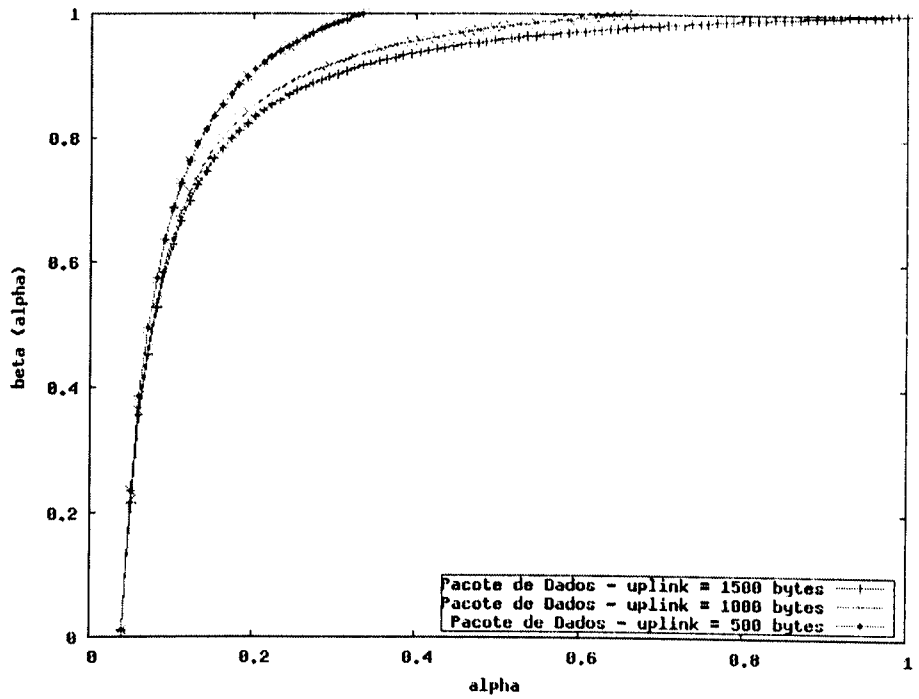


**Figura 3.7.** Variação de $\beta$ em função de $\alpha$ para pacotes com tamanhos diferentes no canal reverso

## 3.3 Considerações

Mostramos, então, que a taxa de *upload* interfere diretamente no desempenho do canal de *download* em transmissões bidirecionais ou não, sendo fator limitador para a taxa de transmissão do canal mais rápido. Isso se dá principalmente porque o canal reverso precisa de enviar pacotes de confirmação, que ocupam parte da banda, e que podem ser restritos caso o canal de upload esteja sendo utilizado para envio de dados.

O mecanismo de *piggybacking* agrava a perda de desempenho na medida que os seus pacotes de dados, quando preenchem todo o canal, não são capazes de confirmar todos os pacotes de *download* se considerarmos um pacote de confirmação para cada segmento de dados recebido. Todos os valores testados foram baseados em bandas encontradas atualmente disponíveis pelas principais fornecedoras de Internet no país.

# Capítulo 4

# Problemas do TCP em redes assimétricas

Neste capítulo analisaremos o problema da perda de desempenho do TCP em redes assimétricas. No capítulo 3 propusemos um modelo matemático e agora estudaremos os métodos já existentes que buscam amortizar ou reduzir os efeitos causados ao canal de *download* em transmissões bidirecionais.

## 4.1 Técnicas para melhoria de desempenho

Partindo do modelo exposto, temos uma melhor visualização dos problemas enfrentados quando temos transmissões em uma rede com diferenças grandes entre as bandas de *download* e *upload*. Nesta seção mostraremos as técnicas já criadas que buscam de alguma forma alterar o TCP ou incluir mecanismos ao longo das transmissões de forma a não permitir que os efeitos de estrangulamento na banda do canal reverso interfiram no desempenho em ambos os canais.

### 4.1.1 Redução de ACKs

A primeira técnica que iremos abordar é a Redução de ACKs. Como visto anteriormente, os ACKs são pacotes de confirmação para os dados entregues corretamente ao receptor de uma transmissão. ACKs são cumulativos, ou seja, com um pacote de confirmação pode-se informar a entrega de um ou mais pacotes de dados anteriores ao número de sequência confirmado. Assim, caso um ACK seja perdido ao longo do caminho até o originador dos dados, é confirmada a entrega do pacote caso o próximo ACK chegue corretamente.

O TCP inicialmente foi projetado para enviar um ACK para cada pacote de dados recebido (Information Sciences, 1981), mas logo foi notado que essa relação poderia ser aumentada sem grandes danos ao protocolo. Dessa forma, se confirmamos 4 pacotes por ACK, por exemplo, temos uma redução de 75% na quantidade de ACKs emitidos. Voltando ao modelo matemático exposto, vemos que a redução na quantidade de ACKs pode significar uma melhor utilização do canal de *upload* para confirmação dos pacotes, principalmente quando esses recursos são escassos.

Outro fator que pode ser considerado quando reduzimos a quantidade de ACKs no canal reverso é um melhor aproveitamento do canal para envio de dados para *upload*. Se utilizamos uma banda menor para confirmações, reduzimos a perda para os dados enviados no sentido contrário.

Os problemas que são causados quando utilizamos essa técnica dizem respeito, principalmente, à contagem de ACKs no controle de fluxo do TCP emissor. O emissor se baseia na chegada dos ACKs tanto para estimar o RTT da transmissão quanto para aumentar a sua taxa de transmissão (na versão usual do TCP, cada ACK aumenta a janela de transmissão em MSS/cwnd no estado de *Congestion Avoidance*).

Para o problema de estimativa de RTT, temos que o valor para definir o *timeout* da transmissão pode ser modificado, uma vez que somente após a recepção do k-ésimo pacote, será enviado o ACK, aumentando consideravelmente o RTT.

Para o crescimento da janela de transmissão, teremos perdas de desempenho à medida em que o crescimento é afetado diretamente com a redução de ACKs. Para resolver esse problema, alguns trabalhos modificam o TCP do emissor, fazendo com que o mesmo conte a quantidade de *bytes* confirmados, ao invés da quantidade de ACKs, como veremos mais à frente.

Além disso, ainda temos outro problema associado a essa solução: o envio de rajadas de dados. Uma vez que o emissor receberá uma só confirmação para vários segmentos, a janela se abre abruptamente e causa o envio de vários segmentos, um após o outro, o que aumenta as chances de perdas dos dados ao longo do caminho. Para esse problema, alguns trabalhos limitam os tamanhos das rajadas emitidas ou incluem um mecanismo de controle de taxa de transmissão, como por exemplo o *token bucket*.

### 4.1.2 Escalonamento de pacotes no canal reverso

Outra técnica que pode ser utilizada é o escalonamento no canal reverso dos segmentos emitidos pelo *uplink*. A forma como o canal é escalonado pode variar de acordo com a abordagem do problema, podendo esta usar uma modelagem matemática mais simples

ou mais refinada. Busca-se melhorar o desempenho definindo a ordem na qual pacotes de dados e ACKs serão enviados de forma a não gerar esperas muito longas para os pacotes de confirmação e ao mesmo tempo não permitir que os dados enviados pelo canal reverso sejam descartados na fila de saída.

Uma das técnicas propostas é a priorização dos ACKs em relação aos dados. Se priorizarmos os ACKs, certamente teremos um quadro diferente do que propusemos no modelo apresentado, onde os pacotes de dados tinham prioridade no canal. Além disso, devido ao tamanho inferior dos ACKs, temos que o tempo para transmiti-los é bem menor do que para os dados. Em contrapartida, a priorização dos ACKs pode gerar esperas muito longas por parte da transmissão reversa, enfileirando os dados de maneira que eles nunca sejam enviados, dependendo da banda de *download* da transmissão. A priorização dos pacotes de dados não é uma boa solução, uma vez que o tamanho dos pacotes de dados é suficientemente grande para ocupar o canal de *upload* por um tempo considerável, atrasando o envio dos ACKs e reduzindo o aumento na janela de transmissão no canal de *download*.

Uma solução interessante é escalonar, usando algum modelo de justiça, de modo a tornar o envio de ACKs e dados proporcional o suficiente para não deteriorar nenhum dos dois fluxos de transmissão. O escalonamento pode ser feito por um componente externo às extremidades da transmissão, sendo este um roteador com o algoritmo de escalonamentono desejado. À este elemento também damos o nome de *gateway*, por ser o responsável pelos pacotes que saem do *uplink*.

### 4.1.3 Alteração no tamanho dos pacotes

Como dissemos na modelagem matemática, o tamanho dos pacotes de dados no canal reverso pode influenciar a taxa de *download* de uma transmissão. Quando utilizado o mecanismo de *piggybacking* em uma transmissão bidirecional, podemos tirar proveito da redução no tamanho dos pacotes no canal reverso para confirmar mais pacotes do fluxo de *download*.

Já aumentando o tamanho dos pacotes no canal de *download*, temos indiretamente uma redução na quantidade de ACKs a serem enviados de volta, aliviando o canal reverso. De fato, alterações no tamanho dos pacotes estão presentes na grande maioria, se não em todos os trabalhos que envolvem propostas de melhoria. No passado, compressão de cabeçalhos foi proposto inicialmente para os ACKs de forma a consumir uma banda menor de *upload*. Hoje essa técnica já não se justifica devido às mudanças nas taxas de *download* e *upload* presentes.

Outra proposta pode ser o uso de um roteador intermediário, chamado de *proxy*,

que altere o tamanho dos pacotes de forma a melhorar o desempenho da rede de acordo com a estratégia definida. Alguns tipos específicos de *proxy* dividem a conexão TCP em duas de forma a configurar características distintas para as duas extremidades. Dessa forma é possível se aumentar ou reduzir o tamanho dos pacotes enviados em uma ponta sem interferir na conexão da outra extremidade. A esses tipos de mecanismos damos o nome de PEP — *Performance Enhancement Proxy* (Border et al., 2001). Outras configurações também podem ser feitas usando essa ferramenta, que através de um elemento intermediário (o próprio PEP) permite alterações das transmissões no meio do caminho.

Outros exemplos de mecanismos para melhorar a transmissão TCP que utilizam PEP's (Border et al., 2001) são o AF (*ACK Filtering*) e AR (*ACK Reconstruction*) — seção 4.2.1 —, para fazer a filtragem e reconstrução dos ACKs, o mecanismo de Espaçamento de ACKs, ou *Pacing*, para espaçar os ACKs e reduzir a quantidade de rajadas de dados em resposta a chegada dos ACKs um após o outro, como veremos mais à frente.

### 4.1.4 Alterações derivadas

O uso de técnicas para melhoria de desempenho em redes que utilizam o TCP para comunicar geralmente gera efeitos colaterais. Como vimos, a redução de ACKs torna a estimativa de RTT inválida, além de interferir no crescimento da janela de transmissão do emissor. Dessa forma, como veremos a seguir, os trabalhos que utilizam essas técnicas necessitam, em sua grande maioria, de técnicas secundárias para tratar os efeitos colaterais gerados.

Essas técnicas podem ser aplicadas no TCP em ambas as pontas ou não, de acordo com a abordagem desejada. O importante é não permitir que a resolução do problema causado pelo estreitamento do canal reverso leve a outros, não sendo eficaz. Na próxima seção analisaremos trabalhos que se utilizam das técnicas aqui colocadas.

## 4.2 Soluções Propostas por Outros Autores

Vários trabalhos já foram desenvolvidos com o intuito de melhorar o desempenho do TCP em redes assimétricas, como veremos a seguir. Separamos os trabalhos relacionados mais relevantes até o momento e que tratam do problema em questão de várias formas distintas, mas sempre buscando impedir que a assimetria nos canais de transmissão seja fator degenerativo para a vazão da rede.

Faremos uma comparação entre as técnicas estudadas que buscam reduzir o problema estudado nessa dissertação.

### 4.2.1 Redução de ACKs

Nesta seção estudaremos os trabalhos que levam em consideração a técnica de redução de ACKs para melhorar o desempenho do TCP em redes assimétricas. Veremos algumas técnicas para reduzir os ACKs, bem como avaliações de resultados e alterações derivadas dos trabalhos aqui discutidos.

Balakrishnan et al. (1997) apresentam duas possíveis soluções para problemas relacionados ao TCP em redes assimétricas através de alterações nas pontas da transmissão de forma a reduzir a quantidade de ACKs enviados pelo canal reverso (cliente → servidor).

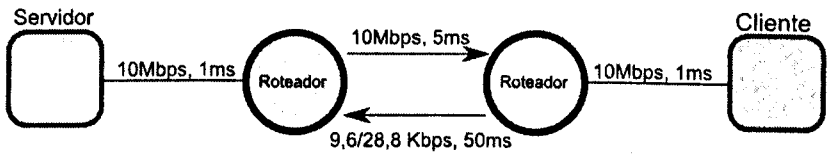


**Figura 4.1.** Topologia usada para estudos retirado de Balakrishnan et al. (1997)

A figura 4.1 ilustra a transmissão de dados entre um cliente e servidor, onde as bandas do servidor para o cliente (*download*) e do cliente para o servidor (*upload*) apresentam assimetria elevada (10Mbps de *download* e 28,8Kbps de *upload*). O trabalho apresentado por Balakrishnan et al. (1997) utiliza essa topologia, que também é utilizada na maioria dos trabalhos levantados, como veremos mais à frente. É importante ressaltar aqui que esse grau de assimetria não é mais presente nas redes fornecidas atualmente. Além disso, os autores desse trabalho utilizam uma topologia que usa transmissão via satélite para o *download* e por linha discada para *upload*, agravando a assimetria que, além de ser presente nas bandas, está presente também na taxa de perdas e no atraso dos pacotes. Tecnologias de satélites atualmente não apresentam mais essa assimetria, uma vez que são utilizados tanto para *download* quanto para o *upload* e com taxas menos discrepantes[1].

Essa topologia utilizada permite que sejam simuladas transmissões nas duas direções entre cliente e servidor, de forma a tornar claro os efeitos gerados pela limitação

[1]O GESAC (Governo Eletrônico — Serviço de Atendimento ao Cidadão), programa do governo para prover Internet para toda a população, mostra que as bandas de transmissão via satélite fornecidos variam entre 256Kbps/64Kbps a 8Mbps/1Mbps de downlink/uplink - http://www.direitoacomunicacao.org.br/novo/content.php?option=com_content&task=view&id=475 acessado em 29 de julho de 2009

no canal reverso no crescimento da janela de transmissão do servidor, bem como nos demais mecanismos do TCP para utilizar o canal com a máxima eficiência.

Uma medida importante nos experimentos é o grau de assimetria do canal. Os autores criaram um parâmetro avaliador que leva em conta as diferenças entre as bandas de transmissão e entre os pacotes que são transmitidos, k, já mencionado no capítulo 3, que representa a razão entre a taxa de pacotes nos canais de *upload* e *download* (equação 3.7). Esse fator define o grau de assimetria da conexão. O valor de k é também a quantidade de pacotes de dados que precisam ser confirmados cumulativamente por apenas um ACK para que não se tenha congestionamento no canal reverso (*uplink*) apenas com ACKs (Aggarwal et al., 2000; Kalampoukas et al., 1998; Brouer & Hansen, 2004).

Os autores argumentam que, quando o canal reverso é congestionado, os pacotes de confirmação serão enfileirados ou perdidos, gerando grandes atrasos para o servidor, que por sua vez depende da chegada dos ACKs para manter o crescimento da sua janela de transmissão (Kleinrock, 1975; Balakrishnan et al., 2002; Purdue & Blanton, 2005; Allman & Blanton, 2005).

Nesse caso, uma possível perda dos ACKs ao longo do caminho irá afetar diretamente o envio de dados do servidor. Com a redução dos ACKs, a perda de um deles irá gerar uma demora demasiada até que o próximo seja enviado informando a chegada correta dos pacotes anteriores. Como já foi dito, os ACKs no TCP são cumulativos, então a perda de um deles pode não gerar retransmissões de dados, mas ainda assim afeta o desempenho do *download*, que depende da chegada dos ACKs para aumentar a taxa de transmissão. Esses problemas, quando colocados em transmissões bidirecionais, onde tanto cliente quanto servidor enviam pacotes de dados, geram um agravamento da assimetria e consequentemente maiores perdas de desempenho.

Os mecanismos apresentados para tentar solucionar os problemas são chamados de Controle de Congestionamento de ACKs (*ACK Congestion Control* — ACC) e Filtragem de ACKs (*ACK Filtering* — AF). A seguir veremos os princípios de funcionamento e os trabalhos relacionados a eles que utilizam das técnicas para melhorar o desempenho de transmissões TCP.

**Controle de Congestionamento de ACKs – ACC**

O mecanismo ACC propõe que o receptor (cliente) também tenha um mecanismo de controle similar ao Controle de Congestionamento clássico do TCP (Allman et al., 2002) para os ACKs que envia. O mecanismo controla a quantidade de pacotes de dados que receberão confimação com um único ACK (atrasos na confirmação). Um parâmetro $d$ é criado no receptor e define a quantidade de dados que o mesmo irá esperar receber para enviar um ACK. O ACC se baseia no fato de que os ACKs são

cumulativos.

Para que esse mecanismo funcione, os autores propõem que sejam feitas duas alterações. A primeira delas é no *gateway* do canal reverso. Neste caso, é usado o mecanismo ECN junto ao *gateway*, que irá marcar o bit ECT no cabeçalho TCP quando houver indícios de congestionamento na rede. O algoritmo usado por Balakrishnan et al. (1997) marca tanto pacotes de dados quanto pacotes de confirmações e detecta o congestionamento através de análise no uso do *buffer*. Quando o emissor recebe um pacote de confirmação com o bit CE marcado, ele reduz sua taxa de transmissão. O mesmo vale para o receptor com relação aos ACKs ao receber um pacote de dados marcado.

A segunda alteração é no receptor, onde o TCP sofrerá alterações no mecanismo de envio de confirmações. Quando o receptor recebe um pacote de dados com o *flag* CE do ECN marcado, ele aumenta o valor de $d$ multiplicativamente. Dessa forma, o atraso para o envio de ACKs será maior e menos pacotes de confirmação serão transmitidos pelo canal reverso. A cada RTT onde o *gateway* não receba um pacote de confirmação com o bit CE configurado faz com que o valor de $d$ seja reduzido linearmente. Quando $d$ é reduzido, maior será a quantidade de ACKs lançados na rede. Dessa forma, busca-se reduzir a quantidade de ACKs no canal reverso de acordo com o estado da rede.

Este mecanismo interfere diretamente na taxa de emissão de dados, uma vez que o TCP depende da quantidade de ACKs para aumentar sua janela de transmissão(Allman, 1998). Logo, para que esse mecanismo funcione sem maiores problemas, seria imprescindível a alteração do TCP no emissor, onde o mesmo basearia o crescimento no tamanho de sua janela de transmissão na quantidade de *bytes* confirmados pelo ACK, e não mais pela quantidade de ACKs recebidos, como é o padrão. Os trabalhos de Allman (1998, 2003, 1999) fazem uso ou mostram os benefícios desta ferramenta. Balakrishnan et al. (1997) também propõem soluções, como veremos a seguir.

**Filtragem de ACKs – AF**

O mecanismo AF também é usado para reduzir a quantidade de confirmações no canal reverso (Barakat & Altman, 2000; Balakrishnan et al., 1997; Wu et al., 1999). Para que o mesmo funcione, bastam alterações no roteador do canal reverso ou no próprio receptor, caso utilize um sistema operacional que suporte o mecanismo nativamente. Dessa forma, quando um pacote de confirmação chega ao roteador de saída, o mesmo verifica se já existe outro pacote de confirmação para o mesmo fluxo, e se aproveita do fato de que os ACKs são cumulativos para eliminar as confirmações mais antigas do *buffer*. É claro que para esta técnica não são necessárias mais informações além das que vêm no cabeçalho do ACK.

Em Wu et al. (1999) os autores apresentam três possíveis políticas de descarte dos

pacotes, bem como uma análise do efeito que as mesmas têm em relação ao desempenho final de transmissão:

1. Eliminação de todos os ACKs anteriores, deterministicamente;
2. Eliminar aleatoriamente, de acordo com o estado da fila;
3. Substituir os ACKs antigos pelo novo, ou seja, colocar o ACK mais novo na posição do ACK mais antigo da fila.

Foi mostrado que o mecanismo AF melhora consideravelmente o desempenho, bem como reduz o RTT das transmissões. O problema levantado quanto ao AF são as rajadas de dados que são enviados quando um ACK confirma vários pacotes simultaneamente. Como mencionado anteriormente, esse não é um efeito desejável, e poderia ser contornado se os autores tivessem criado mecanismos no emissor para que o mesmo reduzisse as rajadas limitando a quantidade de pacotes que poderiam ser enviados um após o outro. O problema dessa solução é que seriam necessárias alterações em outros pontos da conexão, reduzindo a simplicidade que o AF apresenta. Balakrishnan et al. (1997) implementa uma técnica complementar para redução das rajadas.

Quanto ao crescimento da janela de congestionamento, Balakrishnan et al. (1997) altera o TCP emissor para considerar a quantidade de dados confirmados ao invés do número de ACKs que chegam ao emissor.

Os resultados para transmissões unidirecionais apresentam melhoras significativas. Para as técnicas propostas por Balakrishnan et al. (1997), há ganhos ainda maiores quando usada a compressão dos cabeçalhos do protocolo SLIP, chegando a quase 100% de utilização com uma banda no canal reverso de 28,8Kbps e compressão. O uso do protocolo caiu em desuso atualmente, junto com a técnica de compressão, que não faz mais tanto sentido devido aos avanços nas bandas de *download* e *upload*.

Ainda para transmissões unidirecionais, experimentos com vários fluxos no mesmo sentido foram feitos para averiguar a justiça no compartilhamento do canal. Nos testes, os autores utilizam algumas versões diferentes do TCP e as adaptações propostas. O TCP Reno obteve maior vazão, porém não obteve um índice de justiça [2] superior aos testes feitos usando os mecanismos ACC e AF. Os resultados apresentados por Wu et al. (1999) também mostram grandes avanços na melhoria de desempenho em transmissões unidirecionais, bem como uma maior justiça quando o AF é utilizado para os fluxos

[2] O valor do índice de justiça é dado por $f$ e definido como $f = \left(\sum_{i=1}^{n} x_i\right)^2 / n \cdot \sum_{i=1}^{n} x_i^2$, onde $n$ é o número de conexões concorrentes na rede e a banda utilizada pela conexão $i$ é dada por $x_i$, $1 <= i <= n$.

em transmissão com destaques para a política de descarte de todos os ACKs antigos na fila deterministicamente.

Para as transmissões bidirecionais, Balakrishnan et al. (1997) apresenta resultados distintos de acordo com a ordem em que as transmissões se iniciam nos dois sentidos. Quando a conexão no sentido servidor → cliente se inicia primeiro, os ACKs ficarão enfileirados primeiro no *buffer* do canal reverso, impedindo a transmissão dos dados no outro sentido. Quando o fluxo cliente → servidor se inicia primeiro, os pacotes de dados preenchem o buffer e impedem que os ACKs da outra transmissão cheguem ao seu destino. Os resultados apresentados mostraram que é possível conseguir uma taxa ótima de utilização do *downlink*, mas comprometendo a vazão no canal reverso. No trabalho de Wu et al. (1999) o comportamento é idêntico. Com o início da transmissão servidor → cliente antes da transmissão reversa, o *buffer* no canal reverso ficará cheio de ACKs, gerando perdas de dados e levando a taxa de transmissão no canal reverso a zero para o TCP normal e grandes avanços quando utilizado o AF. Porém, as transmissões utilizando o AF sofrem de congelamento (*starvation*) quando a transmissão reversa é iniciada.

Dessa forma, já é possível observar as dificuldades em se conseguir mecanismos que consigam avançar na melhoria de desempenho em várias transmissões nos dois sentidos. Ainda com as melhoras apresentadas, os resultados se mostram insuficientes para promover uma utilização ótima nos dois canais de transmissão, sendo que um fluxo sempre ficará prejudicado para que o outro avance no crescimento de sua taxa de transmissão.

Para os fins dessa dissertação, podemos dizer que as técnicas de outros autores aqui estudadas ainda apresentam dificuldades de serem implementadas, uma vez que demandam modificações no TCP do emissor e no roteador de saída do mesmo. Além disso, é importante notar que a maioria das soluções levam a outros problemas, que demandam mais esforços e alterações ao longo da rede, dificultando a difusão das mesmas no contexto atual em que vivemos.

**O Modelo AMP**

O trabalho de Hua et al. (2002) apresenta um modelo matemático denominado AMP e mostra resultados quando utilizada uma técnica similar ao Filtro de ACKs. Os autores partem do pré-suposto que uma fração do canal reverso é utilizado para envio exclusivo de ACKs, sendo o restante dedicado aos dados.

Foi usado um modelo de topologia para os experimentos parecido com os do trabalho de Balakrishnan et al. (1997), utilizando canais diferentes com redes cabeadas (*modem*) e sem fio (rádio) para *upload* e *download*, respectivamente. Hua et al. (2002) caracterizam o impacto da assimetria utilizando a razão normalizada de assimetria, o

parâmetro $k$, definido por Lakshman et al. (1997) e também utilizado por Balakrishnan et al. (1997). Os autores consideraram os tráfegos bidirecionais e *overheads* de protocolos utilizados como fatores que podem alterar o valor de $k$.

O modelo AMP recebe esse nome porque utiliza as variáveis que definem a capacidade que o canal principal obterá sobre determinadas condições do canal reverso. Consideremos, em um dado instante, que a fração alocada no canal reverso para os ACKs satura quando chega a **A** ACKs/segundo. Cada ACK irá gerar **M** pacotes em média devido aos efeitos do TCP (*ACK-clocking*), sendo o tamanho médio dos pacotes definido por **P** bits/pacote. Sendo a capacidade do canal mais rápido **F bits/s**, os autores chegaram ao resultado de que a vazão máxima do canal principal é dada por

$$min(F, A \cdot M \cdot P) bits/segundo \tag{4.1}$$

É possível, através dessa expressão, explorar algumas formas para melhorar a vazão do canal mais rápido (do servidor para o cliente). Se obtivermos um fator multiplicativo maior (M), uma maior fração do canal alocado para os ACKs, ou pacotes maiores, é possível tornar a vazão maior.

Para testar o modelo em transmissões bidirecionais, os autores criaram filas distintas para dados e ACKs, com a finalidade de controlar a variável A. Considerou-se que pacotes de dados no canal de *download* ocuparão uma fração $f$ da capacidade total do canal ($C_f$) e no canal reverso uma fração $g$ ocupará parte da capacidade do canal reverso ($C_g$). Assim, as frações dos canais de *download* e *upload* dedicadas aos ACKs (A) serão dadas por $1 - f$ e $1 - g$, respectivamente.

Suponhamos um canal de 1Mbps/300Kbps de *download/upload*. Sendo f a fração do *download* utilizada, ou seja, $0 \leq f \leq 1$, e $f \times 1Mbps$ é a parte do canal de *download* que será dedicado aos dados, será atingida uma banda de $(1 - f) \times 1Mbps$ para envio de ACKs no canal principal. O mesmo vale para o canal reverso, onde a variável $g$ é utilizada para definir a fração do canal de *upload* que será usado para os dados.

É fácil notar que o fluxo de dados do canal reverso depende do valor de $g$, que define a fração do canal que será usada pelos segmentos de dados. Já para o canal de *download*, é preciso que se leve em consideração a fração deixada para os ACKs retornarem pelo canal reverso $(1-g)$, o fator M e o tamanho dos pacotes de confirmação e de dados[3].

Este modelo se relaciona com o nosso modelo ao considerar a fração dos canais utilizados pelos dados como uma variável controlada. Dessa forma, os autores consideram que há um meio para utilizar o canal da forma desejada, dedicando parte dele

[3] A taxa máxima de dados em pacotes/segundo é dado por $Min\left[\frac{(1-g)C_r M}{P_{ack}}, \frac{f*C_f}{P}\right]$

aos dados. Isso facilita a modelagem, uma vez que permite definir o ponto onde haverá saturação do canal de *upload* e ao mesmo tempo pode ser usado para impedir essa saturação.

Partindo desse modelo, Hua et al. (2002) propuseram duas alternativas para melhorar o desempenho de redes assimétricas, o SAD (Eliminação Inteligente de ACKs, em tradução livre) e a Política de Regeneração de ACKs.

**Eliminação Inteligente de ACKs - SAD**

O SAD é uma adaptação do AF de Balakrishnan et al. (1997). É criado um mecanismo no *gateway* de saída do receptor com uma fila para os ACKs de tamanho N, onde N é o número de fluxos sendo transmitidos. Uma tabela *hash* é utilizada para armazenar a informação da existência de um ACK referente a um determinado fluxo na fila, bem como o número do último ACK a entrar na fila. A política usada para escalonar ACKs e dados é o CBQ (Irwin et al., 1998) — *Class Based Queueing* —, onde os fluxos são divididos em classes hierarquicas separadas e os pacotes escalonados de maneira a manter a taxa para cada fluxo dentro do determinado.

Quando chega um ACK de um dos fluxos, é verificado se não há outro na fila. Caso não exista, o bit referente à presença de um pacote de confirmação do fluxo é mudado para 1, e o valor do ACK é armazenado junto à tabela. Quando já existe outro ACK na fila, é atualizada a informação do valor do ACK mais atual e descartado o pacote que chegou. Quando o ACK for enviado pelo canal, o último valor na tabela é copiado para seu cabeçalho e o bit é reconfigurado para informar que não há ACKs desse fluxo em fila.

Assim como no AF, essa técnica gera rajadas de dados no emissor. Para contornar esse problema, a Reconstrução/Regeneração de ACKs (AR) — esquema também sugerido por Balakrishnan et al. (1997) — é aplicada na chegada do transmissor, de forma a recompor os ACKs que foram eliminados antes de sua chegada. O AR reconstrói os ACKs suprimidos pelo SAD. É definido um fator de regeneração R por Hua et al. (2002) sobre quantos ACKs serão reconstruídos por MSS, o que altera o desempenho do emissor e o crescimento da janela de transmissão.

Os resultados apresentados por Hua et al. (2002) usando o simulador ns2 (Mccanne et al., 2009) mostram que as técnicas atuantes podem trazer ganhos significativos para o desempenho do TCP. Nas transmissões unidirecionais, o SAD apresentou uma vazão no canal de *download* muito boa, com destaques para o cenário onde o valor de R para o AR foi definido como $R = 4$, fazendo com que o aproveitamento do canal chegasse a mais de 93%.

Em tráfegos bidirecionais é possível de se observar os mesmos problemas relacionados ao canal mais lento de transmissão. Priorizando ACKs, prejudicamos o envio

de dados pelo canal reverso, e priorizando dados alteramos o funcionamento da janela de transmissão do emissor do canal mais rápido. Esse cenário foi estudado colocando-se uma fila FIFO no canal reverso. Nesse caso tivemos uma queda de desempenho muito grande no canal de *download*, enquanto o canal reverso apresentava taxas maiores e bem próximas de sua capacidade.

Com $f = 99,7\%$, ou seja, 99,7% do canal de *download* reservado para os dados, e variando-se o valor de $g$, que define a porcentagem do canal de *upload* que irá ser usado para transmissão de dados, foram obtidos bons resultados utilizando o SAD, o AR e a política CBQ nas filas, na ordem de 10 a 20 vezes quando compara-se com o uso de filas FIFO simplesmente para gerenciamento do canal reverso.

O trabalho de Hua et al. (2002) não apresenta alterações no TCP das pontas da transmissão, apenas o uso de roteadores ligados as pontas para fazer o controle do uso das bandas. Como dito anteriormente, os roteadores podem ser alterados com a implementação de filas separadas por hierarquias de forma a serem escalonadas e consequentemente delimitando a capacidade de cada classe representada por cada fila, além de priorização de algum tipo de informação a ser transmitida. Esse mecanismo certamente é mais simples de ser implementado em relação ao ACC, por exemplo, que exige alteração no TCP do receptor.

Apesar do trabalho desenvolvido por Allman (1998) (seção 4.2.2) não levar em conta ambientes assimétricos, é fácil notar que os problemas levantados em um ambiente simétrico se agravam mais quando colocados em assimetria de bandas. Logo, os resultados e melhoras em relação ao atraso dos ACKs podem ser considerados em ambientes assimétricos, como pode ser visto no trabalho desenvolvido por Landström & Larzon (2007).

Este trabalho (Landström & Larzon, 2007) faz um estudo quantitativo da redução de ACKs em comparação com a estratégia de atraso dos ACKs usadas por Allman (1998). Reduzir a quantidade de ACKs, como já visto, tem como objetivo melhorar o desempenho da rede em transmissões assimétricas, e tem se tornado vantajoso em ambientes onde há escassez de recursos no canal onde os ACKs são transmitidos.

Landström & Larzon (2007) fizeram modificações nas pontas da rede para reduzir os ACKs por considerarem as informações presentes no emissor e no receptor mais ricas para a escolha de quais ACKs deveriam ser enviados.

O mecanismo ABC (*Appropriate Byte Counting*), de Allman (2003), limita a quantidade de dados que podem ser confirmados por um ACK usando um parâmetro $L$ que é a quantidade máxima de segmentos aceita para confirmação. Esse mecanismo também faz a contagem de bytes recebidos por um pacote de confirmação ao invés da quantidade de ACKs para regular o crescimento da janela de transmissão. Isso faz com

que os danos causados pela redução dos ACKs para o emissor sejam diluídos.

Landström & Larzon (2007) partiram do trabalho desenvolvido por Allman (1998) criando as mesmas condições de simulação mas verificando o efeito que o atraso dos ACKs pode ter em ambientes assimétricos. Além disso, foi alterado o mecanismo ABC para que fosse possível confirmar mais do que dois pacotes com um ACK. Para garantir que os pacotes não seriam reenviados por causa do maior atraso causado, foram feitas também alterações na temporização da transmissão, de forma a garantir que os atrasos não seriam interpretados como perdas.

Utilizando diferentes estratégias, Landström & Larzon (2007) buscaram encontrar um valor ótimo de pacotes de dados que deveriam ser esperados para o envio de uma confirmação. Isso consequentemente reduz a quantidade de ACKs enviados pelo emissor e reduz o tráfego no canal reverso. Através de simulações e estudos, Landström & Larzon (2007) chegaram a conclusão de que dois ACKs por janela de transmissão são suficientes para utilizar totalmente a banda do canal, sendo que a i-ésima rajada de dados deve chegar ao receptor tão logo a confirmação da (i-1)-ésima rajada seja enviada pelo receptor.

As estratégias usadas para coletar dados foram:

- confirmar cada segmento (AE)
- atrasos para envio de confirmação com ($L = 2$) (DAL2)
- confirmar o quarto pacote com $L = 4$ (A4L4)
- confirmar o oitavo pacote com $L = 8$ (A8L8)
- confirmar dois segmentos para cada janela de dados enviada (2W)
- confirmar quatro segmentos para cada janela de dados enviada (4W)

Landström & Larzon (2007) verificaram que a confirmação de segmentos por janela enviada (2W e 4W) gera uma menor quantidade de ACKs em relação aos demais. Isso é fácil de ser explicado uma vez que independente do tamanho da janela de transmissão, sempre haverá um valor fixo de ACKs enviados por janela. Logo, quanto maior a janela, melhor vai ser o efeito de redução de ACKs das estratégias 2W e 4W em comparação com os demais. É válido comentar que isso também irá gerar rajadas de dados, um efeito nada desejado para as transmissões TCP, como já comentamos anteriormente.

Para os casos onde o fluxo de dados é muito baixo, os autores mostraram que as estratégias que confirma o i-ésimo pacotes (A4L4 e A8L8) serão prejudicados. Dessa

forma, é colocado em um limite inferior que faz com que o segundo pacote seja confirmado caso o fluxo de dados esteja muito baixo.

Essas estratégias levam a vários problemas que foram estudados e solucionados em separado por Landström & Larzon (2007). Estudos em fluxos unidirecionais são feitos usando as mesmas simulações de (Allman, 1998) para mostrar as soluções apresentadas.

A primeira alteração diz relação ao mecanismo ABC. A contagem de bytes apresenta um limite de apenas 2 pacotes de dados por confirmação em sua versão original. Após um *timeout*, o ABC configura $L = 1$ limitando o crescimento da janela de transmissão na fase de partida lenta. Com as alterações, quando ocorre um *timeout*, o receptor reduz a frequência dos ACKs aumentando o limite $L$ do mecanismo quando a taxa de transmissão do emissor é alta e não há espaços entre os segmentos recebidos. Isso evita que o emissor continue dobrando sua taxa da transmissão durante o restante do período de partida lenta.

Landström & Larzon (2007) verificaram que as estratégias 4W e A4L4 apresentam melhor desempenho quando variados tanto os tamanhos dos arquivos enviados quanto o tamanho dos *buffers* para esse ambiente.

Para analisar as taxas de perdas e a recuperação de perdas, Landström & Larzon (2007) fizeram simulações usando fluxos multiplexados e variações na banda de transmissão e nos RTTs. Foram feitas simulações usando bandas de 1Mbps e 100Mbps. Para a banda de 1Mbps, o mecanismo 2W não obteve nenhuma vazão para um fluxo apenas, enquanto o 4W tem uma utilização maior para a banda de 1Mbps. Para o 2W a perda por *timeout* é comum. Um mecanismo chamado de AF (*Acknoledgment Fixes*) é criado para resolver o problema quando uma retransmissão rápida ocorre, gerando aumento na taxa de envio do emissor e mais perdas. Esse mecanismo aumentou de 73% para 86% a utilização do canal pelo 2W em um canal de 1Mbps e de 88% para 94% no caso do 4W com uma banda de 100Mbps. Resultados para outras bandas não apresentaram o mesmo desempenho.

Adaptações no tamanho do *buffer* foram feitas levando-se em conta as necessidades de cada estratégia. A estratégia 4W mais uma vez apresentou bons resultados exigindo poucos recursos de *buffer*. A vazão dos ACKs também atende ao modelo dos autores, sendo que o 4W apresenta o maior limite inferior dentre todos.

Landström & Larzon (2007) mostraram então que a estratégia de redução de ACKs combinada com a contagem de bytes (ABC) é viável e apresenta bons resultados quando são enviados quatro ACKs por janela de transmissão.

O trabalho desenvolvido por Landström & Larzon (2007) tem uma abordagem concreta e avalia de maneira a complementar o trabalho de Allman (1998), que não leva em conta ambientes assimétricos. Um valor ótimo de redução foi encontrado, mas

a abordagem de modificação do TCP nas pontas de transmissão gera problemas para uma implementação desta solução em larga escala. Apesar disso, os resultados obtidos por Landström & Larzon (2007) podem ser úteis para estudos futuros que busquem reduzir os ACKs em redes assimétricas.

### 4.2.2 Atrasos maiores para os ACKs

Nesta seção discutimos trabalhos que propõem a técnica de aumento nos atrasos para o envio dos ACKs baseando-se no fato de que os ACKs são cumulativos. Esta técnica poderia estar entre as consideradas na seção 4.2.1, uma vez que o atraso dos pacotes de confirmação acaba reduzindo a quantidade dos mesmos no canal reverso, mas decidimos separar os trabalhos, uma vez que essa abordagem apresenta trabalhos mais específicos e que buscam estudar o fenômeno do atraso dos ACKs com maior profundidade (Allman, 1998; Afifi et al., 1998), enquanto as demais técnicas não levam os atrasos em consideração nos experimentos.

No trabalho desenvolvido por Afifi et al. (1998), a solução desenvolvida abrange as redes assimétricas, em especial as domésticas (ADSL). A idéia principal é que o esforço para alterar a rede não necessite de muitas alterações no protocolo TCP. A viabilidade é colocada como fator importante no processo.

Três propostas são feitas por Afifi et al. (1998) para melhorar o desempenho de redes assimétricas, sendo que uma delas propõe alterações no mecanismo de controle de fluxo do TCP, enquanto as outras não exigem modificações diretas no TCP.

A primeiro proposta de Afifi et al. (1998) é atrasar o envio de ACKs combinado com compressão de cabeçalho, técnica essa que não é mais justificável, como discutido anteriormente. Levando-se em conta que os ACKs são cumulativos, não é preciso que a confirmação seja enviada tão logo o pacote de dados tenha chegado ao destino. Para verificar um valor razoável para o atraso máximo, Afifi et al. (1998) fizeram simulações com ns2 (Mccanne et al., 2009). Após testar vários atrasos, verificou-se que o atraso de $20ms$ apresenta uma vazão ótima para a conexão na fase de *Congestion Avoidance*, enquanto o atraso de $0ms$ se apresentou melhor na fase de Partida Lenta. A compressão de cabeçalhos é outra alternativa apresentada, mas que pode encontrar problemas caso a rede seja híbrida, além de ter caído em desuso atualmente, como já expusemos anteriormente na seção 4.2.1.

A segunda proposta de Afifi et al. (1998) modifica a política de descartes da fila no canal reverso. Mais uma vez considerando que os ACKs são cumulativos, pode-se alterar a política de descarte das filas, que por padrão descartam os últimos pacotes a chegar a no *buffer* (*drop-tail*). Com a alteração, se torna mais interessante descartar

os primeiros pacotes da fila, mais antigos e que chegaram primeiro (*drop-front*). Dessa forma, pode-se descartar um ACK mais antigo contando que o novo ACK do mesmo fluxo que entrar na fila de roteamento por último irá confirmar todos os dados que iriam ser confirmados anteriormente pelo ACK descartado. Usando essa estratégia os autores conseguiram um ganho de 20% no desempenho do canal mais rápido. O TCP por padrão conta a quantidade de ACKs recebidos para aumentar sua janela de transmissão, ou seja, a quantidade de dados confirmados não interfere no crescimento da janela. Se tornamos os ACKs mais escassos, prejudicamos o crescimento da janela e consequentemente reduzimos o desempenho do TCP emissor. Para contornar esse problema, Afifi et al. (1998) aumentam sua janela de transmissão se baseando na quantidade de dados confirmados por cada ACK. Esta técnica, quando utilizada, exige alterações para que o TCP do emissor faça a contagem dos *bytes* recebidos ao invés do número de ACKs. Um ponto ruim deste trabalho é que os autores simplesmente mudaram a política de descarte de *drop-tail* para *drop-front*, e não mostraram de onde vieram as melhoras, apesar dos dados comprovarem que essa técnica melhora, de fato, o desempenho.

A última alteração no trabalho de Afifi et al. (1998) consiste na modificação do receptor para prever congestionamentos no canal reverso. É criado um algoritmo baseado na separação do RTT em FTT (*forward-trip time*) e BTT (*backward-trip time*), que são os tempos gastos pelo pacote de dados para chegar ao receptor e do ACK para chegar ao emissor, respectivamente. Esses valores são calculados usando-se o *timestamp* do cabeçalho TCP. O receptor mede a variação de tempo reportada pelos *timestamps* do emissor e comparando-os com os instantes em que ele recebeu os ACKs. Essas informações, em conjunto com o tamanho da fila e a vazão da rede são usados para aumentar ou reduzir a quantidade de ACKs enviados no canal. O valor estimado da vazão da rede ($\mu$) e o tamanho da fila, denotado por $q_r$ entram em uma função que permite ao receptor encontrar a quantidade de segmentos de dados que devem esperar para o envio de um ACK. Essa alteração no fluxo de ACKs é dinâmico, e faz com que o atraso aumente ou reduza de acordo com o estado do canal reverso. Os resultados apresentados mostraram que essa técnica gera ganhos em relação ao uso do TCP Reno com e sem atrasos estáticos de ACKs.

É possível notar, como dissemos ao início dessa seção, que a tarefa de atrasar a emissão dos ACKs leva a uma redução da quantidade de confirmações lançados no canal reverso. Logo, o atraso no envio dos ACKs pode ser considerado um método para reduzir o tráfego no canal de *upload*, e consequentemente para melhorar o desempenho do *download*, que é afetado nos casos onde ACKs são perdidos ou sofrem atrasos muito grandes ao longo do canal de transmissão, interferindo diretamente no crescimento da

janela de transmissão e na sua taxa de envio.

Afifi et al. (1998), apesar de obterem bons resultados, explicitam de maneira muito pobre a relação entre o algoritmo e as melhoras em um tráfego dinâmico de rajadas, sendo que essa informação fica sem uma consistente justificativa no trabalho. Apesar de não entrar em detalhes sobre a última ferramenta além da exposição do algoritmo que é usado, é claro que para sua implementação, devemos fazer alterações no TCP do cliente ou receptor, que irá regular os atrasos para o envio das confirmações segundo o algoritmo proposto.

### 4.2.3 Impacto da técnica de contagem de bytes

O trabalho desenvolvido por Allman (1998) também busca estudar os efeitos do atraso no envio dos pacotes de confirmação em transmissões TCP. Este trabalho se encontra mais completo que o proposto por Afifi et al. (1998) no que diz respeito à análise das possibilidades de alterações no TCP e na rede para que o atraso nos ACKs não tenha efeitos colaterais. Isso se deve principalmente ao fato de Allman (1998) considerar o uso da Contagem de Bytes no emissor ao invés de simplesmente atrasar os ACKs e esperar que a taxa de envio de dados continue crescendo normalmente. Os autores não levam em conta a assimetria dos canais, mas ainda assim é possível entender que a técnica aqui proposta exige que outras técnicas sejam implementadas para resolver os efeitos colaterais das alterações.

Allman (1998) propõe três alternativas para entender o uso dos ACKs. O primeiro deles é a técnica padrão de enviar um ACK por pacote recebido durante a fase de partida lenta, com atrasos após essa fase. A segunda é o uso da contagem de bytes de maneira ilimitada, onde o TCP irá aumentar sua taxa de transmissão de acordo com a quantidade de dados confirmados, e não mais pela quantidade de ACKs recebidos. A terceira é uma versão limitada da contagem de bytes, onde a quantidade máxima de pacotes confirmados é dois. Isso limita o tamanho e a quantidade de rajadas de dados, que faz também com que ACKs sejam enviados em conjunto um após o outro.

Allman (1998) avalia uma transmissão unidirecional e algumas transmissões bidirecionais dinâmicas, levando em conta tanto transmissões longas quanto curtas. Ambientes que utilizam o TCP SACK também foram simulados para obter resultados. Enviando um ACK por segmento trouxe os melhores resultados em todos os experimentos, apesar de apresentar um custo maior para o canal de confirmação. Esse mecanismo também apresentou mais perdas, uma vez que gera maior crescimento na taxa de transmissão do que com os atrasos nos ACKs. O uso da técnica denominada DAASS (atraso somente depois da Partida Lenta) também apresentou resultados posi-

tivos. Para não afetar a fase inicial de transmissão, que é onde a janela de transmissão cresce acentuadamente, foi proposto que os atrasos nos ACKs só fossem implementados após a fase de Partida Lenta. Essa técnica necessita de um mecanismo extra para que o receptor detecte quando o emissor está na fase de Partida Lenta ou não, que é um trabalho deixado em aberto por Allman (1998). A técnica de contagem ilimitada de bytes (UBC) foi desaconselhada devido ao aumento exacerbado na quantidade de rajadas emitidas. Já a técnica de contagem de bytes com limites possui um melhor desempenho, com um crescimento não muito significativo nas rajadas e perdas.

### 4.2.4 Alteração no Tamanho dos Pacotes

Nesta seção veremos trabalhos que utilizam da técnica de alteração nos tamanhos dos pacotes para melhorar o desempenho de redes assimétricas. Assim como na seção 4.2.2, vale notar que esta, como a maioria das técnicas buscam a redução na quantidade de ACKs, como veremos a seguir. Separamos esta seção no intuito de mostrar uma abordagem diferente para a solução de redução de ACKs, onde não há necessidade que as pontas (emissor e receptor) sofram alterações.

Como dito anteriormente, o PEP (Border et al., 2001) geralmente é usado para melhorar o desempenho de protocolos da Internet, quebrando a conexão ao longo do caminho em várias conexões, usando parâmetros diferentes nas duas pontas de transmissão. Hasegawa et al. (2003) fazem uma proposta de melhoria para o TCP em redes assimétricas usando o PEP para reduzir a quantidade de ACKs no canal reverso, sem a necessidade de alterações no TCP emissor ou receptor.

Esta proposta (Hasegawa et al., 2003) divide o fluxo TCP em dois, um do emissor até o *proxy*, e outro do *proxy* até o receptor. O *proxy* (PEP) acumula os dados enviados pelo emissor, e aumenta o tamanho dos pacotes de dados antes que os mesmos sejam enviados. Com o aumento dos tamanhos dos pacotes de dados, menos pacotes são enviados para o receptor, resultando em uma menor quantidade de confirmações no canal reverso.

Quando as confirmações chegam ao *proxy*, são retransmitidas para o emissor como se não houvesse alterações no fluxo, ou seja, o *proxy* envia a quantidade de ACKs para o emissor de acordo com a quantidade de pacotes que o mesmo enviou. Dessa forma, a relação entre o crescimento da janela de transmissão e a chegada de ACKs não é afetada.

Durante a fase de conexão (*three-way handshake*), o *proxy* configura o MSS (tamanho máximo do segmento) para as duas pontas. Para o servidor é configurado o MSS padrão (1460 bytes), enquanto o cliente recebe um MSS de tamanho bem maior

que o padrão (4420 bytes, por exemplo). Como o pacote terá um tamanho maior do que o MTU da rede, o mesmo será quebrado em vários na camada IP pelo próprio *proxy* usando uma função de fragmentação dos datagramas, sem alterar a dinâmica do TCP. Dessa forma, quando os pacotes chegarem ao cliente, serão agrupados pela camada IP e repassados para o TCP cliente.

Hasegawa et al. (2003) usaram máquinas reais com sistemas operacionais distintos para obter as informações sobre o ganho ao utilizar o PEP. Foi verificado que o processo de rearranjo na camada IP provocado pelo aumento do MSS entre o *proxy* e o cliente não sobrecarrega o processamento da máquina cliente. Além disso, os ganhos chegam a mais de 100% em alguns sistemas, bem como há uma redução na quantidade de ACKs para um terço da quantidade anterior, valor referente a quantidade de pacotes IP's que são fragmentados com um MSS de 4420 bytes.

Essa é uma técnica elegante e promissora devido aos poucos pontos da rede onde são necessárias alterações para o funcionamento. O *proxy* pode, por exemplo, ser instalado pelas empresas provedoras de Internet via ADSL sem custos adicionais para o cliente, o que facilita sua implementação. Não se vê, neste trabalhos, efeitos colaterais aparentes ou outras ferramentas complementares necessárias, sendo considerado um trabalho muito bem elaborado. A apresentação feita por Hasegawa et al. (2003) é clara e objetiva.

### 4.2.5 Escalonamento de Pacotes

Nesta seção veremos alguns trabalhos que utilizam mecanismos para escalonar os pacotes que saem ou entram pelos canais de *downlink* e *uplink*, com a intenção de melhor utilizar os canais, melhorando o desempenho de redes assimétricas.

No trabalho desenvolvido por Kalampoukas et al. (1998) busca-se resolver o problema de compressão de ACKs em transmissões bidirecionais em canais assimétricos através de um mecanismo PEP que gerencie a fila do canal reverso, de maneira a tornar o fluxo de *download* e *upload* o mais otimizado possível. Esse trabalho (Kalampoukas et al., 1998) tem como característica predominante a apresentação de modelos matemáticos das soluções propostas.

Três formas de gerenciar a fila são implementadas, modeladas e simuladas. A topologia da rede usada é similar à apresentada na figura 4.1 com variações no valor da banda nos dois sentidos e no atraso de cada canal.

A primeira tentativa de melhoria foi usando a priorização dos ACKs na fila do canal reverso. É fácil de ver as consequências desse mecanismo, conforme discussões anteriores. Primeiramente, há um ganho de vazão considerável para o canal de *down-*

*load*, uma vez que os ACKs chegarão ao servidor sem problemas de atrasos. A janela de congestionamento do emissor irá crescer de maneira natural, sem prejudicar o TCP do emissor. Em contrapartida, haverá problemas no canal reverso, que ao enviar dados deverá aguardar a transmissão dos ACKs, utilizando apenas o que restar da banda do canal. Caso se tenha uma taxa de transmissão alta no canal mais rápido, mais ACKs serão enviados pelo receptor, podendo levar a transmissão reversa a uma taxa zero de transferência.

Os experimentos mostraram esse fenômeno, levando a taxa de transmissão de dados no canal reverso de 0 a 10% de sua capacidade, ou seja, uma melhora na eficiência da transmissão pelo canal mais rápido, mas deteriorando claramente o canal reverso.

A segunda tentativa de melhoria foi limitando o tamanho do *buffer* na camada IP para os pacotes de dados. Essa técnica é aplicada nos dois fluxos de dados. Para que não sejam perdidos pacotes, um mecanismo de *backpressure* é usado na camada TCP, que envia os pacotes para a camada IP. Dessa forma o TCP só redireciona pacotes para a camada IP dentro do limite colocado pelo tamanho do *buffer* do IP.

Os ACKs serão enfileirados após os mesmos até que sejam enviados a medida que os segmentos TCP são enviados à camada IP. Quanto maior o tamanho dos segmentos de dados, maior a quantidade de ACKs enfileirados, uma vez que o tempo gasto na transmissão dos segmentos é grande. Isso irá gerar rajadas de ACKs ao serem enviados um após o outro, que consequentemente gerarão rajadas de dados no lado transmissor.

Esta técnica se mostrou bastante sensível ao tamanho dos pacotes de dados, que altera a quantidade de ACKs seguidos enviados nas duas extremidades. Essa imprevisibilidade aliada à heterogeneidade das redes atuais, não garante o bom funcionamento dessa técnica em larga escala.

A terceira proposta feita é também a mais interessante. Baseando-se nas modelagens matemáticas para calcular a eficiência dos canais nos dois sentidos, Kalampoukas et al. (1998) propõem um algoritmo que regula a fila do canal reverso. Esse controle busca uma maximização da vazão do canal mais rápido e uma eficiência mínima para o fluxo de dados do canal reverso controlando a quantidade de ACKs que serão enviados para cada pacote de dados. Duas filas são usadas para separar os ACKs dos dados em conjunto com um escalonador que se baseia na eficiência desejada para definir quais pacotes serão enviados.

O escalonador implementado por Kalampoukas et al. (1998) conta o tamanho dos ACKs em *bytes* transmitidos e envia um pacote de dados quando este valor ultrapassa um limite, que define a fração da banda usada pelos pacotes de confirmação. Caso não existam dados para transmissão, há a priorização dos ACKs. A título de exemplo, o algoritmo mostrado na figura 4.2 garante 50% do uso do canal reverso para dados e 50%

para ACKs. Esse algoritmo é implementado na fila do canal reverso. Para alterar essa razão basta modificar a forma como *ack_bytes_tx* é comparado com *data_bytes_tx*, gerando a relação desejada entre ACKs e dados no canal reverso.

```
if sistema ocioso then
| aguarda o próximo pacote;
end
if (data_present) and (! ack_present) then
| ack_bytes_tx ← 0;
| data_bytes_tx ← data_packet_size;
| send(data packet);
end
if (ack_present) and (! data_present) then
| data_bytes_tx ← 0;
| ack_bytes_tx ← ack_size;
| send(ack);
end
if (data_present) and (ack_present) then
| if ack_bytes_tx ≥ data_bytes_tx then
| | send(data packet);
| | data_bytes_tx+ = data_pkt_size ;
| | if data_bytes_tx > ack_bytes_tx then
| | | data_bytes_tx− = ack_bytes_tx ;
| | | ack_bytes_tx = 0 ;
| | end
| else                                   /* mais dados do que ACKs */
| | send(ack);
| | ack_bytes_tx+ = ack_size ;
| | if ack_bytes_tx > data_bytes_tx then
| | | ack_bytes_tx− = data_bytes_tx ;
| | | data_bytes_tx = 0 ;
| | end
| end
end
```

**Figura 4.2.** Pseudo-código do escalonador das filas de envio da conexão reversa — retirado de (Kalampoukas et al., 1998)

O algoritmo garante a eficiência desejada balanceando a quantidade de bytes enviados para confirmações e dados, o que o torna estável em relação a alterações nas variáveis dos fluxos nos dois sentidos.

Quando o tamanho dos segmentos da transmissão mais rápida são reduzidos, apenas o fluxo mais rápido é afetado com uma queda significativa (caindo para 7% de eficiência). A conexão reversa não sofre com alterações dos pacotes do outro canal.

Essa queda ocorre devido à maior quantidade de ACKs necessários para confirmar a maior quantidade de pacotes de dados, o que torna mais difícil de se manter a janela de transmissão crescendo na mesma taxa anterior com as limitações impostas pela fila do canal reverso. Isso é um bom resultado, uma vez que a vazão do canal reverso só depende do escalonador de fila e dos parâmetros do seu fluxo.

Um trabalho que segue a mesma linha de raciocínio desenvolvida por Kalampoukas et al. (1998) foi proposto por Brouer & Hansen (2004). Esse é mais um trabalho que busca não alterar o TCP em nenhuma das pontas, colocando um PEP intermediário na transmissão dos dados que priorize ACKs e escalone os pacotes que saiam do emissor. O que diferencia esse trabalho é o uso de redes reais ADSL com 200 usuários para coletar os dados, bem como a consideração da existência de tráfego *peer-to-peer* na rede.

Brouer & Hansen (2004) montaram um experimento em uma rede e modificaram o roteador de saída da mesma com o objetivo de verificar a eficácia de alguns métodos. Primeiramente fizeram duas simulações com um *download* de 200MB de dados com tráfego real e artificial no canal reverso. Partindo daí, aumentaram a janela de transmissão consideravelmente, gerando perdas maiores, apesar da melhora no desempenho do *download*.

Após isso, testaram a técnica de priorização dos ACKs, e detectaram, assim como Kalampoukas et al. (1998), que esse método leva a uma ótima utilização do canal de *download* em detrimento do canal reverso, que sofre com grandes atrasos causados pela ocupação pelos ACKs do canal de *download*.

Partindo daí, Brouer & Hansen (2004) passam a dividir o fluxo de *upload* entre cinco categorias definidas de acordo com o tipo de tráfego presente no canal. Essas categorias são definidas por:

**Iterativa:** tráfego de *shell* seguro e de protocolos de *chat*;

**Confimações:** ACKs de todas as classes de tráfego;

**Tráfego bom:** serviços bem conhecidos incluindo protocolos como http, ftp, dns, etc.;

**Tráfego ruim:** *peer-to-peer*, protocolos como eDonkey, Kazaa, BitTorrent, etc.;

**Padrão:** tráfego que não se enquadra nas demais categorias.

Para criar o ambiente de coleta de dados, Brouer & Hansen (2004) utilizaram o HTB (*Hierarquical Token Bucket*), ferramenta implementada em sistemas Linux para controle de banda. O HTB fornece meios de separar os tráfegos em categorias distintas

de acordo com o seu comportamento, como por exemplo, o porto utilizado, e além disso permite que os tráfegos de categorias distintas tenham limites de bandas separados.

Usando o HTB (Benita, 2005), para definir as classes, as prioridades e as frações de uso do canal, Brouer & Hansen (2004) detectaram que essa estratégia pode melhorar em muito o desempenho do canal reverso, bem como garantir que todos os fluxos consigam ser transmitidos.

Como resultados, os autores mostraram que separando 90% do canal reverso para os ACKs com priorização é possível atingir 60% de eficiência no *download*. Separando 45% do canal reverso para os ACKs, obtem-se uma vazão do *download* em torno de 50% devido ao aumento no atraso para a chegada dos ACKs.

Brouer & Hansen (2004) fazem uma simulação do AF (Balakrishnan et al., 1997) reduzindo o tamanho da fila de ACKs no roteador de saída para cinco ACKs. Essa estratégia melhora significativamente os resultados, e se baseia na cumulatividade dos ACKs. Se fosse utilizada uma estratégia de descarte diferente, como feito em (Afifi et al., 1998), onde os pacotes são descartados da frente, talvez melhores resultados fossem encontrados do que os apresentados, uma vez que descartaríamos os ACKs mais velhos.

Para que o PEP criado utilize todo o canal de *upload*, é preciso que a taxa de transmissão seja a mais próxima possível da capacidade do canal. Para isso, é importante que o *overhead* da rede utilizada seja calculado. Como naquele trabalho é utilizada uma rede ATM (Kouvatsos, 2000; Tanenbaum, 2002), há um pequeno estudo com a finalidade de considerar este *overhead* no cálculo da quantidade de dados emitidos no canal.

O trabalho desenvolvido por Park & Hong (2007) também se enquadra dentro das técnicas de escalonamento. Nesse trabalho são estudados os efeitos da assimetria em redes domésticas com uma análise voltada para o *gateway* da rede. Ao analisar o desempenho nos canais através de tráfegos P2P, foi percebido que a taxa de *upload* atingida não ultrapassava um terço da capacidade total, enquanto a taxa de *download* se encontrava aquém de sua máxima taxa de envio.

Segundo Park & Hong (2007), a intenção é que apenas o *gateway* sofra alterações, de forma a viabilizar a solução. Nesse caso, foi proposta uma separação entre os ACKs e os dados, com uma prioridade maior dada aos ACKs. Através de experimentos, foram detectados os problemas e comprovados pelos resultados a fim de sanar os problemas de queda no desempenho das taxas nos dois canais quando há tráfego nos dois sentidos.

Foi proposto um ambiente no *gateway* onde há duas filas na saída do canal reverso e há priorização dos ACKs. Alegando que a demanda pelos dados no canal reverso é bem maior do que a demanda dos ACKs, Park & Hong (2007) garantem

que não ocorrerá o caso de apenas ACKs serem transmitidos. A banda consumida por ACKs é de aproximadamente 11,5% da banda total. Também usando o HTB (Benita, 2005) são implementadas as filas e os resultados apontam uma melhora significativa principalmente para a vazão do canal reverso.

## 4.3 Classificação dos Trabalhos

Nesta seção faremos uma análise dos artigos aqui estudados tentando extrair e mapear as técnicas utilizadas para solucionar os problemas enfrentados pelo TCP em redes assimétricas. Nos baseamos em parte na categorização de Balakrishnan et al. (2002), um trabalho extremamente relevante e que busca mapear, da mesma forma como queremos, as soluções apresentadas para o problema aqui estudado.

Como dito anteriormente, Balakrishnan et al. (2002) definem a assimetria de rede como sendo a diferença entre os canais de recepção e envio de dados em um dado ponto da mesma.

Baseado nesse problema, técnicas distintas foram utilizadas até o presente momento buscando amenizar os efeitos da assimetria em todos os tipos de redes citados aqui. O foco do nosso trabalho se encontra na assimetria de banda e nas consequências para o TCP, que sofre perdas quando tem seus pacotes de confirmação atrasados ou perdidos devido ao tráfego no sentido contrário da transmissão. Essas abordagens estão dispostas na tabela 4.1.

Como vimos anteriormente, existem duas necessidades importantes quando enfrentamos os problemas descritos até aqui neste trabalho, segundo Balakrishnan et al. (2002) . A primeira delas é a importância do gerenciando da capacidade do canal reverso, onde dados e ACKs competem pelo espaço. Se for dada prioridade máxima a um dos dois, afetaremos fatalmente a outra transmissão. A segunda necessidade é evitar o impacto causado quando temos ACKs menos frequentes na rede. Reduzir a quantidade de pacotes de confirmação enviadas de volta afeta diretamente o TCP emissor, que depende dos ACKs para manter a sua taxa de envio.

Partindo daí, uma possível divisão entre as técnicas poderia ser em técnicas que demandam alterações diretas no TCP ou que não exigem uma modificação direta do protocolo. Essa divisão é bem clara nos trabalhos apresentados e pode definir a viabilidade de algumas técnicas no contexto atual. As modificações no TCP são mais difíceis de serem aceitas, uma vez que demandariam alterações em todos os sistemas que utilizam o TCP, ou seja, mais de 90% da Internet.

As técnicas que não exigem essas alterações apresentam vantagens na medida em

**Tabela 4.1.** Classificação dos trabalhos estudados

| *Técnica* | *Trabalhos que utilizam* | *Modificações* |
|---|---|---|
| Aumento no MSS | (Hasegawa et al., 2003) | Uso de PEP |
| Reconstrução de ACKs | (Balakrishnan et al., 1997; Hua et al., 2002) | Uso de PEP no canal de download |
| Compressão de Cabeçalhos (ACKs) | (Balakrishnan et al., 1997; Afifi et al., 1998) | Canal de upload/TCP Receptor |
| Filtro de ACKs | (Balakrishnan et al., 1997; Hua et al., 2002) | PEP no canal de upload |
| Escalonador por fluxo | (Kalampoukas et al., 1998; Brouer & Hansen, 2004; Afifi et al., 1998) | PEP no canal de upload |
| Priorização de ACKs | (Kalampoukas et al., 1998; Brouer & Hansen, 2004) | PEP no canal de upload |
| Controle de Congestionamento de ACKs | (Balakrishnan et al., 1997) | TCP do Emissor e Receptor |
| Espaçamento (Pacing) no emissor | (Balakrishnan et al., 1997; Landström & Larzon, 2007) | TCP do Emissor |
| Contagem de bytes | (Balakrishnan et al., 1997; Landström & Larzon, 2007) | TCP do Emissor |
| Mudanças no atraso dos ACKs | (Afifi et al., 1998; Landström & Larzon, 2007) | TCP do Receptor |
| Backpressure | (Kalampoukas et al., 1998) | TCP do Receptor |

que alterações ao longo do canal podem ser feitas uma única vez e afetar a todo um conjunto de usuários da rede, por exemplo, em casos onde PEPs são instalados para redução de ACKs, atrasos, escalonamento de pacotes, etc. Essas técnicas podem estar na saída do canal reverso ou na chegada do canal principal. Geralmente as que ficam na saída do canal reverso são utilizadas para melhor utilizar o *uplink*, escalonando ou reduzindo a quantidade de ACKs presentes no mesmo. Como essas técnicas geram efeitos colaterais para o TCP emissor, em grande parte dos trabalhos há a necessidade de se instalar um PEP no canal de *download*. Nesse caso, esses *proxies* são utilizados para reconstrução de ACKs filtrados, gerar espaçamento entre os pacotes de dados (*pacing*), etc.

As técnicas que demandam alterações diretas no TCP podem afetar o emissor, o receptor ou ambos. Técnicas que afetam o emissor de dados podem ser consideradas para espaçar os dados em micro-rajadas (*pacing*), mecanismos para contagem de bytes, ao invés da quantidade de ACKs recebidos (UBC, LBC, ABC), etc. Já no caso de técnicas que demandam alterações no receptor temos os mecanismos de atrasos ou

redução no envio de ACKs, a ferramenta de *backpressure*, etc. O ACC exige alterações tanto no emissor quanto no receptor da transmissão, bem como no canal reverso. Para o seu bom funcionamento, é preciso que o *gateway* de saída acione o bit CE do ECN dos segmentos para indicar um possível congestionamento. O TCP receptor irá variar a taxa de envio de ACKs de acordo com o estado da rede, enquanto o emissor de dados deverá usar um mecanismo de contagem de bytes para que o crescimento na taxa de transmissão não sofra quedas por causa da redução de pacotes de confirmação. A figura 4.3 mostra as separações entre os artigos segundo as categorias que definimos.

As soluções mais encontradas utilizam o canal reverso sem alterações diretas no protocolo TCP. A redução na quantidade de ACKs enviados pelo canal reverso é uma excelente solução e apresenta melhoras significativas em todos os trabalhos que utilizaram técnicas neste ponto da transmissão. O uso de estratégias transparentes para melhorar o desempenho pode ter um incentivo a mais quando implementadas pela provedora do serviço (em redes ADSL) sem a necessidade de gastos com tempo e dinheiro por usuários domésticos. Além disso, temos a vantagem de que tanto emissor quanto receptor não precisarão se preocupar com nenhuma alteração visível do seu funcionamento, obtendo ganhos em cima do que a conexão irá fornecer.

Podemos também dividir os trabalhos segundo a abordagem utilizada na seção 4.1:

- Técnicas de Redução de ACKs, que incluem atrasos nos ACKs, ACC, AF, etc.;
- Técnicas de Escalonamento no Canal Reverso, que incluem o uso de PEPs com filas separadas e prioridades especificadas para cada tipo de pacote que trafega no canal reverso, escalonadores por fluxo, etc.;
- Técnicas de Alterações no Tamanho dos Pacotes, podendo ser estas PEPs que alterem o tamanho de ACKs, com compressão dos mesmos, ou aumento no tamanho dos pacotes de dados do canal principal;
- Alterações Derivadas, que nada mais são do que técnicas complementares para sanar os efeitos colaterais causados pelas técnicas anteriores, como a Reconstrução de ACKs (AR), Contagem de Bytes, etc.

Nesse tipo de categorização, podemos enfatizar a predominância de técnicas que visam reduzir a quantidade de ACKs no canal reverso, consumindo menor quantidade de recurso neste canal e permitindo que a taxa de upload não seja afetada diretamente pela presença de ACKs no canal.

Dessa forma, buscamos, como Balakrishnan et al. (2002), classificar os trabalhos desenvolvidos em redes assimétricas que buscam melhorias para o protocolo TCP. Não existem apenas essas formas de categorização, mas cremos que dessa forma já é possível mapear o estado da arte do problema e entender melhor quais são as abordagens do problema mais aconselháveis atualmente.

## 4.4 Considerações finais

Baseando no que foi exposto, não consideramos até o presente momento nenhum trabalho favorável à uma solução ideal do problema em estudo. A assimetria causa problemas na transmissão de dados nos dois sentidos de transmissão de um determinado ponto da Internet, e o fato do TCP ser baseado em confirmações agrava o problema. Através do modelo pudemos observar que o comportamento do canal reverso influencia diretamente a taxa de *download*, e que a busca por uma taxa ótima de transmissão pelo canal mais rápido requer uma deterioração do fluxo de *upload*.

Trabalhos que buscam reduzir a quantidade de ACKs estão mais presentes na literatura, mas trabalhos com propostas que reduzam os impactos causados pela redução de ACKs têm mais chances de serem candidatos a solução.

Outros protocolos que utilizam confirmações para garantir a entrega dos dados também seriam prejudicados pela assimetria na banda, como é o caso do SCTP (*Stream Control Transmission Protocol*), que também é baseado em janela de transmissão. Protocolos que exigem confirmações mas que são baseados em fluxo, como os casos do RTP (*Real-time Transport Protocol*), do UDT (*UDP based Data Transfer*) e do TCP Vegas demandariam mais estudos para averiguar os reais impactos causados pela assimetria.

Diante de tudo que foi exposto aqui, temos em mente uma possível solução que traria grandes benefícios e um trabalho ainda não abordado ate o presente momento. Propomos controlar o valor de $\beta$ (utilização do canal reverso para dados) utilizando *pacing*, de forma a encontrar um ponto ótimo entre as taxas de *download* e *upload* de maneira autônoma. Dessa forma, teríamos um sistema que se adaptaria ao ambiente em que fosse colocado, levando em conta os problemas enfrentados em redes assimétricas, regulando nesses casos o uso do canal reverso de maneira a não prejudicar as transmissões nos dois sentidos. Para categorizar essa possível solução, teríamos a necessidade de alterações diretas no TCP receptor, ou mesmo a implementação desse mecanismo em um PEP ou *gateway*. Apesar da segunda solução parecer mais simples de implementar, vemos na primeira alternativa um maior potencial de melhorias e contribuição para o

problema. Alterando diretamente no TCP temos que o sistema operacional poderá se encarregar de lidar com os problemas na rede, tirando do usuário o trabalho de dizer qual a melhor utilização do *uplink* por meio de ferramentas secundárias. Isso tornaria o TCP mais dinâmico e adaptável através da percepção da rede em que a máquina está conectada.

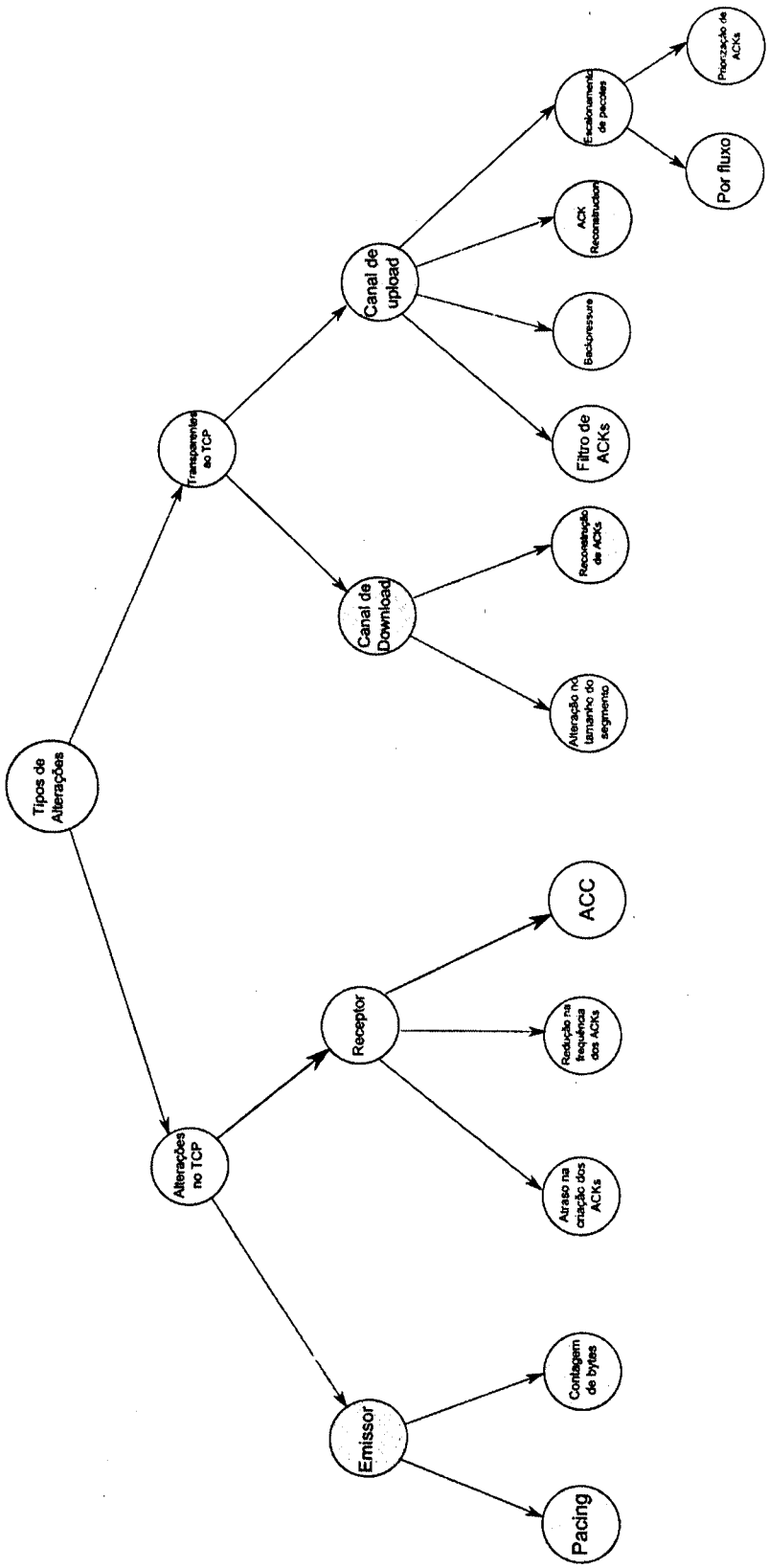


**Figura 4.3.** Diagrama de soluções

# Capítulo 5

# Conclusão

Os trabalhos apresentados e estudados não apresentam uma palavra final sobre o problema estudado. Dessa forma, todos os métodos e soluções apresentam ainda dificuldades, seja para divulgação e implementação em larga escala, ou alta complexidade na abordagem dos problemas, levando a resultados nem tão consistentes.

A concorrência pelo canal reverso entre dados e ACKs gera perdas significativas para os tráfegos nos dois sentidos da transmissão, e uma abordagem para melhor utilizar o canal reverso através de controle de fluxos é bem visto pela maioria dos autores aqui pesquisados. Os efeitos causados pela assimetria podem ser vistos em nosso modelo matemático, onde levamos em consideração que há priorização para os dados no canal reverso.

Consideramos que trabalhos que não necessitem de alterações no TCP são mais facilmente implementáveis, enquanto apresentam menor quantidade de informações sobre o estado das conexões para tomarem decisões. Os trabalhos que alteram o TCP diretamente apresentam resultados mais consistentes, como o caso do ACC e do mecanismo de Contagem de Bytes, mas ainda assim demandam um esforço muito grande para se tornarem realidade atualmente.

Trabalhos que priorizem os ACKs não apresentam resultados satisfatórios, e podem levar a transmissão do canal reverso a uma taxa zero de envio. Trabalhos que escalonam dados e ACKs são melhor estruturados e certamente melhores soluções para o problema. Além disso, a implementação dessas técnicas é facilitada quando feita nos roteadores de saída do canal reverso, que podem ser alterados e garantir a melhoria para centenas de usuários que estejam utilizando a mesma rede sem que as máquinas sofram alterações no TCP.

A grande maioria dos trabalhos que buscam reduzir a quantidade de ACKs para poupar o canal reverso sofrem dos efeitos colaterais de compressão de ACKs e con-

sequentes envios de rajadas de dados pelo emissor da transmissão, tornando mais complexa a implementação das soluções propostas, como o caso do ACC e do AF, que necessitam do AR para reconstrução dos pacotes de confirmação para manter o crescimento da janela de transmissão do TCP, que é baseado na chegada dos ACKs.

## 5.1 Trabalhos Futuros

Este material pode ser utilizado para nortear futuras melhorias no TCP com informações relevantes para que a abordagem do problema seja melhor informada e contextualizada. Dessa forma, contribuiremos com todo embasamento teórico necessário para que a abordagem seja mais profícua e traga resultados significativos.

Comprovações através de experimentos reais para o modelo matemático aqui proposto também estão entre os trabalhos futuros. Dessa forma buscaremos confirmar o comportamento do TCP em redes reais assimétricas segundo o nosso modelo.

É possível também que seja proposta, partindo do que foi exposto aqui, uma solução adaptativa onde o TCP teria um mecanismo capaz de se adaptar em função das características da rede, controlando as bandas de *download* e *upload* de forma a otimizar a eficiência no uso dos dois canais sem a intervenção do cliente do sistema, o que traria ganhos consideráveis para o problema de assimetria nas redes domésticas. Esse mecanismo atuaria de maneira autônoma e seria capaz de perceber o canal e tomar decisões baseando-se no meio em que está inserido.

Esse mecanismo seria implementado no TCP das pontas de transmissão, e poderia derivar técnicas vistas aqui como o AF e o ACC junto ao seu sistema. Consideramos essa abordagem mais coerente, uma vez que alterando o TCP diretamente não teríamos necessidade de nenhuma adaptação ou complemento ao usuário da rede, bastando ao mesmo ter o TCP modificado implementado em seu sistema operacional.

Futuras abordagens que tenham diferenças significativas em relação às aqui já mapeadas e categorizadas também podem ser incluídas em conjunto com este trabalho para manter o estado da arte.

# Referências Bibliográficas


Afifi, H.; Elloumi, O. & Rubino, G. (1998). A Dynamic Delayed Acknowledgment Mechanism to Improve TCP Performance for Asymmetric Links. *IEEE Symposium on Computers and Communications*, 0:188.

Aggarwal, A.; Savage, S. & Anderson, T. (2000). Understanding the Performance of TCP Pacing. In *Proceedings of the IEEE INFOCOM*, pp. 1157–1165. IEEE.

Al-Khatib, W. & Gunavathi, K. (2008). A Novel Mechanism to Improve Performance of TCP Protocol over Asymmetric Networks. *International Arab Journal of Information Technology (IAJIT)*, 5(1):66–74.

Allman, M. (1998). On the generation and use of TCP acknowledgements. *ACM Computer Communications Review*, 28(5):4–21.

Allman, M. (1999). TCP byte counting refinements. *SIGCOMM Computer Communications Review*, 29(3):14–22.

Allman, M. (2003). TCP Congestion Control with Appropriate Byte Counting (ABC). RFC 3465 (Experimental).

Allman, M. & Blanton, E. (2005). Notes on burst mitigation for transport protocols. *SIGCOMM Computer Communications Review*, 35(2):53–60.

Allman, M.; Floyd, S. & Partridge, C. (2002). Increasing TCP's initial window. RFC 3390 (Proposed Standard).

Balakrishnan, H.; Padmanabhan, V.; Fairhurst, G. & Sooriyabandara, M. (2002). TCP Performance Implications of Network Path Asymmetry. RFC 3449 (Best Current Practice).

Balakrishnan, H. & Padmanabhan, V. N. (2001). How network asymmetry affects TCP. *IEEE Communications Magazine*, 39(4):60–67.

Balakrishnan, H.; Padmanabhan, V. N. & Katz, R. H. (1997). The effects of asymmetry on TCP performance. In *MobiCom '97: Proceedings of the 3rd annual ACM/IEEE International Conference on Mobile Computing and Networking*, pp. 77–89, New York, NY, USA. ACM Press.

Barakat, C. & Altman, E. (2000). On ACK Filtering on a Slow Reverse Channel. In *Quality of Future Internet Services (QofIS)*, pp. 80–92.

Benita, Y. (2005). Kernel korner: analysis of the HTB queuing discipline. *Linux Journal*, 2005(131):13.

Border, J.; Kojo, M.; Griner, J.; Montenegro, G. & Shelby, Z. (2001). Performance Enhancing Proxies Intended to Mitigate Link-Related Degradations. RFC 3135 (Informational).

Brakmo, L. S. & Peterson, L. L. (1995). TCP Vegas: end to end congestion avoidance on a global Internet. *IEEE Journal on Selected Areas in Communications*, 13(8):1465–1480.

Brouer, J. D. & Hansen, J. S. (2004). Experiences with reducing TCP performance Problems on ADSL. Relatório Técnico 04/07, Datalogisk Institut Københavns Universitet (DIKU).

Dawkins, S.; Glover, D.; Griner, J.; Tran, D.; Henderson, T.; Heidemann, J.; Touch, J.; Kruse, H.; Ostermann, S.; Scott, K. & Semke, J. (2000). Ongoing TCP Research Related to Satellites. RFC 2760.

Fall, K. & Floyd, S. (1996). Simulation-based comparisons of Tahoe, Reno and SACK TCP. *SIGCOMM Computer Communications Review*, 26(3):5–21.

Floyd, S. (1994). TCP and Explicit Congestion Notification. *ACM Computer Communication Review*, 24:10–23.

Floyd, S. & Fall, K. (1997). NS Simulator Tests for Random Early Detection (RED) Queue Management. Relatório Técnico, CiteSeerX - Scientific Literature Digital Library and Search Engine [http://citeseerx.ist.psu.edu/oai2] (EUA). Visitado em novembro de 2009.

Floyd, S. & Jacobson, V. (1993). Random Early Detection Gateways for Congestion Avoidance. *IEEE/ACM Transactions on Networking*, 1:397–413.

Floyd, S.; Mahdavi, J.; Mathis, M. & Podolsky, M. (2000). An Extension to the Selective Acknowledgement (SACK) Option for TCP. RFC 2883 (Informational).

Hasegawa, T.; Hasegawa, T. & Lagreze, M. (2003). A Mechanism for TCP Performance Enhancement over Asymmetrical Environment. In *IEEE Symposium on Computers and Communications (ISCC)*, pp. 1135–1140. IEEE Computer Society.

Hemminger, S. (2005). Network Emulation with NetEm. In *Linux.conf.au. http://developer.osdl.org/shemminger/netem/LCA2005_paper.pdf. Obtido em fevereiro de 2009.*

Hua, D. S.; Qin, H.; Kalyanaraman, S. & Kidambi, K. (2002). Performance Optimization of TCP/IP over Asymmetric Wired and Wireless Links. Department of Eletrical, Computer and Systems Engineering, Rensselaer Polytechnic Institute. Disponível em `http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.22.5293&rep=rep1&type=url&i=0`. Visitado em 26/11/2009.

Information Sciences, I. (1981). Transmission Control Protocol. RFC 793. Editado por Jon Postel. Disponível em http://rfc.sunsite.dk/rfc/rfc793.html.

Irwin, K. L.; Gui, I. & Chua, K. (1998). Performance of a Linux Implementation of Class Based Queueing. In *IC3N '98: Proceedings of the International Conference on Computer Communications and Networks*, pp. 370–378, Washington, DC, EUA. IEEE Computer Society.

Jacobson, V. (1988). Congestion Avoidance and Control. In *Proceedings of the ACM SIGCOMM '88*, pp. 314–329, Stanford, CA.

Kalampoukas, L.; Varma, A. & Ramakrishnan, K. K. (1998). Improving TCP Throughput over Two-Way Asymmetric Links: Analysis and Solutions. In *Proceedings of the SIGMETRICS 98.*

Karandikar, S.; Kalyanaraman, S.; Bagal, P. & Packer, B. (2000). TCP rate control. *SIGCOMM Computer Communication Review*, 30(1):45–58.

Kleinrock, L. (1975). *Queueing Systems, Volume 1: Theory.* Wiley.

Knight, S. (1999). The Role of ADSL in Internet Access. *Computer*, 32(7):103–106.

Kouvatsos, D. D., editor (2000). *Performance Analysis of ATM Networks, IFIP TC6 WG6.3/WG6.4 Fifth International Workshop on Performance Modelling and Evaluation of ATM Networks, July 21-23, 1997, Ilkley, UK*, volume 157 of *IFIP Conference Proceedings.* Kluwer.

Lakshman, T. V.; Suter, B. & Madhow, U. (1997). Window-Based Error Recovery and Flow Control with a Slow Acknowledgement Channel: A Study of TCP/IP Performance. In *INFOCOM '97: Proceedings of the INFOCOM '97. Sixteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Driving the Information Revolution*, Washington, DC, USA. IEEE Computer Society.

Landström, S. & Larzon, L.-A. (2007). Reducing the TCP acknowledgment frequency. *SIGCOMM Compututer Communication Review*, 37(3):5–16.

Lin, D. & Morris, R. (1997). Dynamics of Random Early Detection. *SIGCOMM Compututer Communication Review*, 27(4):127–137.

Louati, F.; Barakat, C. & Dabbous, W. (2003). Handling Two-Way TCP Traffic in Bandwidth Asymmetric Networks. Relatório Técnico RR-4950, INRIA, França.

Mccanne, S.; Floyd, S. & Fall, K. (2009). The Network Simulator NS-2. `http://www.isi.edu/nsnam/ns/`. Visitado em 26/11/2009.

Mo, J.; La, R. J.; Anantharam, V. & Walrand, J. (1999). Analysis and comparison of TCP Reno and Vegas. In *INFOCOM '99. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 3, pp. 1556–1563.

Park, J. & Hong, S. (2007). Preventing network performance interference with ack-separation queuing mechanism in a home network gateway using an asymmetric link. In *RTCSA '07: Proceedings of the 13th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, pp. 550–555, Washington, DC, USA. IEEE Computer Society.

Peterson, L. L. & Davie, B. S. (2007). *Computer Networks: A Systems Approach, Fourth Edition (The Morgan Kaufmann Series in Networking)*. Morgan Kaufmann.

Purdue, E. B. & Blanton, E. (2005). On the Impact of Bursting on TCP Performance. In *Proceedings of the Passive and Active Measurement Workshop (PAM)*, pp. 1–12, Boston, MA, EUA.

Tanenbaum, A. S. (2002). *Computer Networks*. Prentice Hall PTR.

Wu, H.; Wu, J.; Cheng, S. & Ma, J. (1999). ACK filtering on bandwidth asymmetry networks. In *Fifth Asia-Pacific Conference on Communications, 1999 (APCC/OECC '99) and Fourth Optoelectronics and Communications Conference*, volume 1, pp. 175–178.

Wu, Y.; Niu, Z. & Zheng, J. (2005). Study of the TCP upstream/downstream unfairness issue with per-flow queuing over infrastructure-mode WLANs. *Wireless Communications and Mobile Computing*, 5(4):459–471.