UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA E INFORMÁTICA INDUSTRIAL

MARCELO RICARDO LEITNER

UMA REVISÃO DO CONTROLE DE CONGESTIONAMENTO DO SCTP

DISSERTAÇÃO

CURITIBA

2019

MARCELO RICARDO LEITNER

UMA REVISÃO DO CONTROLE DE CONGESTIONAMENTO DO SCTP

Dissertação apresentada ao Programa de Pós-graduação em Engenharia Elétrica e Informática Industrial da Universidade Tecnológica Federal do Paraná como requisito parcial para obtenção do grau de "Mestre em Ciências" – Área de Concentração: Telecomunicações e Redes.

Orientador: Mauro S. P. Fonseca

Co-orientadora: Anelise Munaretto Fonseca

CURITIBA

2019

Dados Internacionais de Catalogação na Publicação

Leitner, Marcelo Ricardo
Uma revisão do controle de congestionamento do SCTP [recurso eletrônico] / Marcelo Ricardo Leitner.-- 2019.
1 arquivo texto (52 f.): PDF; 608 KB.

Modo de acesso: World Wide Web
Título extraído da tela de título (visualizado em 11 set. 2019)
Texto em português com resumo em inglês
Dissertação (Mestrado) - Universidade Tecnológica Federal do Paraná. Programa de Pós-graduação em Engenharia Elétrica e Informática Industrial, Curitiba, 2019
Bibliografia: f. 50-52

1. Engenharia elétrica - Dissertações. 2. Protocolos de comunicação de dados. 3. Protocolos de roteamento (Protocolos de redes de computação). 4. TCP/IP (Protocolo de rede de computação). 5. Algoritmos computacionais. I. Fonseca, Mauro Sergio Pereira. II. Fonseca, Anelise Munaretto. III. Universidade Tecnológica Federal do Paraná. Programa de Pós-graduação em Engenharia Elétrica e Informática Industrial. IV. Título.

CDD: ed. 23 – 621.3

Biblioteca Central da UTFPR, Câmpus Curitiba
Bibliotecário: Adriano Lopes CRB-9/1429

Ministério da Educação
Universidade Tecnológica Federal do Paraná
Diretoria de Pesquisa e Pós-Graduação

TERMO DE APROVAÇÃO DE DISSERTAÇÃO Nº 833

A Dissertação de Mestrado intitulada **"Uma revisão do controle de congestionamento do SCTP"** defendida em sessão pública pelo(a) candidato(a) **Marcelo Ricardo Leitner**, no dia **12 de agosto de 2019**, foi julgada para a obtenção do título de Mestre em Ciências, área de concentração **Telecomunicações e redes**, e aprovada em sua forma final, pelo Programa de Pós-Graduação em Engenharia Elétrica e Informática Industrial.

BANCA EXAMINADORA:

Prof(a). Dr(a). Mauro Sergio Pereira Fonseca - Presidente – (UTFPR)
Prof(a). Dr(a). Carlos Marcelo Pedroso - (UFPR)
Prof(a). Dr(a). Keiko Verônica Ono Fonseca - (UTFPR)

A via original deste documento encontra-se arquivada na Secretaria do Programa, contendo a assinatura da Coordenação após a entrega da versão corrigida do trabalho.

Curitiba, 12 de agosto de 2019.

Dedico essa dissertação a meus avós.

AGRADECIMENTOS

Agradeço aos professores, por me mostrarem vez após vez que a educação é o único caminho para melhorarmos nossa sociedade. Que mesmo muitas vezes sendo um processo longo, trabalhoso, seus resultados transcendem nossos horizontes.

Agradeço aos meus pais, pela minha vida. Agradeço a meus amigos, por todo o apoio, de entender o frequente ”hoje não consigo” até me aguentarem falando sobre minhas pesquisas repetidamente.

Agradeço a Priscilla, pelas ideias, pela inspiração.

Você nunca sabe que resultados virão da sua ação. Mas se você não fizer nada, não existirão resultados. (Mahatma Gandhi)

RESUMO

Leitner, Marcelo Ricardo. UMA REVISÃO DO CONTROLE DE CONGESTIONAMENTO DO SCTP. 52 f. Dissertação – Programa de Pós-graduação em Engenharia Elétrica e Informática Industrial, Universidade Tecnológica Federal do Paraná. Curitiba, 2019.

O protocolo de transporte *Concurrent Multipath Transfer for Stream Control Transmission Protocol draft RFC* (CMT-SCTP) (AMER et al., 2019) especifica que o transmissor deve implementar um novo algoritmo, *Cwnd Update for CMT* (CUC), para lidar com o aumento de saltos na sequência de *chunks* (blocos) de dados presentes nos *selective acknowledgment*s (SACKs) do SCTP, que são causados naturalmente pelo uso de múltiplos caminhos. O algoritmo foi proposto de maneira a lidar com a visão tradicional de que saltos na sequência de blocos de dados indicam perdas de pacote e mitigar os efeitos que a reordenação natural do CMT tem neles. O SCTP agora tem uma errata (STEWART et al., 2019b) que trabalha melhor com tais *selective acknowledgment*s (SACKs) na fase de prevenção de congestionamento, pois permite aumentar a janela de congestionamento mesmo sem um novo ACK acumulativo. Todavia a errata não alterou a fase de partida lenta, que será examinada nesse trabalho. Nossos resultados mostram um desempenho similar para baixos tempos de ida e volta (*round trip time*, RTTs) e um ganho para RTTs elevados, enquanto a quantidade total de retransmissões é reduzida. Portanto, nossa proposta é tornar o indicador de recuperação rápida uma variável por transporte, permitir o aumento da janela de congestionamento sem um novo ACK acumulativo mesmo na fase de partida lenta e remover o algoritmo CUC da especificação do CMT-SCTP.

Palavras-chave: SCTP, SFR, CUC, múltiplos caminhos, camada de transporte

ABSTRACT

Leitner, Marcelo Ricardo. A REVIEW OF SCTP'S CONGESTION CONTROL. 52 f. Dissertação – Programa de Pós-graduação em Engenharia Elétrica e Informática Industrial, Universidade Tecnológica Federal do Paraná. Curitiba, 2019.

CMT-SCTP specifies that the sender should implement a new algorithm, CUC, to cope with the increased amount of *gap blocks* present in SCTP *selective acknowledgment*s (SACKs), naturally caused by multipath usage. It was proposed in order to cope with the traditional view that gap blocks indicate packet loss and to mitigate the effects that natural reordering on CMT had on it. SCTP now has an errata (STEWART et al., 2019b) that better deals with such *selective acknowledgment*s (SACKs) in congestion avoidance phase, as it allows increasing congestion window even if the cumulative ACK did not advance. It did not change, though, the slow start phase, and that is what we examine in this paper. Our results indicate the performance is similar for low RTTs and improved for higher ones, while reducing the total amount of retransmissions. Therefore, our proposal is to make the Fast Recovery flag to be per transport, to allow increasing the congestion window without new cumulative ACKs also in slow start phase and to remove CUC indication from CMT-SCTP specification.

Keywords: SCTP, SFR, CUC, multipath, transport layer

LISTA DE FIGURAS

LISTA DE SIGLAS

BDP *bandwidth-delay product*

CMT-SCTP *Concurrent Multipath Transfer for Stream Control Transmission Protocol*

CUC *Cwnd Update for CMT*

CUCv2 *Cwnd Update for CMT version 2*

DAC *Delayed ACK for CMT*

CMT-SCTP *Concurrent Multipath Transfer for Stream Control Transmission Protocol draft RFC*

HTNA TSN mais alto recém confirmado (*Highest TSN Newly Acknowledged*)

IETF *Internet Engineering Task Force*

ISDN *Integrated Services Digital Network*

ISUP *ISDN (Integrated Services Digital Network) User Part*

M2PA *Signaling System 7 (SS7) Message Transfer Part 2 (MTP2) - User Peer-to-Peer Adaptation Layer*

M3UA *Signaling System 7 (SS7) Message Transfer Part 3 (MTP3) - User Adaptation Layer*

MTU unidade máxima de transmissão (*Maximum Transmission Unit*)

NR-SACK *non-renegable selective acknowledgment*

PDU pacote de dados do protocolo (*Protocol Data Unit*)

PMTU unidade máxima de transmissão do caminho (*Path Maximum Transmission Unit*)

PSTN rede pública telefonia comutada (*Public Switched Telephone Network*)

RSerPool *Reliable Server Pooling*

RTO tempo limite de retransmissão (*retransmit time out*)

RTT tempo de ida e volta (*round trip time*)

SACK *selective acknowledgment*

SCTP *Stream Control Transmission Protocol*

SFR *Split Fast Retransmit*

SFR-CACC *Split Fast Retransmit Changeover Aware Congestion Control*

TSN *transmission sequence number*

UA camadas de adaptação do usuário de sinalização (*Signalling User Adaptation Layers*)

SUMÁRIO

1 INTRODUÇÃO

1.1 MOTIVAÇÕES

O SCTP foi inicialmente projetado para o transporte de sinalização quando houve a necessidade de transportar os sinais das redes SS7 em redes IP. Naquela ocasião foi constatado que nem o TCP e nem o UDP atendiam as necessidades do SS7 e foi decidido por projetar um novo protocolo para tal. Hoje o SCTP é uma parte fundamental das camadas de adaptação do usuário de sinalizações (*Signalling User Adaptation Layers*, UAs) do grupo de trabalho SIGTRAN (derivado de *Signalling Transport*), da *Internet Engineering Task Force* (IETF). É ele que serve de transporte para implementações de *SIGTRAN UA* como *Signaling System 7 (SS7) Message Transfer Part 2 (MTP2) - User Peer-to-Peer Adaptation Layer* (M2PA) (DANTU et al., 2005) e o *Signaling System 7 (SS7) Message Transfer Part 3 (MTP3) - User Adaptation Layer* (M3UA) (PASTOR; MORNEAULT, 2006), sendo que esse último é o responsável por permitir que protocolos como *ISDN (Integrated Services Digital Network) User Part* (ISUP) funcionem em redes IP ao invés de em redes *Integrated Services Digital Network* (ISDN) ou rede pública telefonia comutada (*Public Switched Telephone Network*, PSTN). O ISUP é utilizado para estabelecer ligações telefônicas.

Com o tempo o SCTP ganhou alguns novos usuários, já não mais interessados em sinalização propriamente dita. Um exemplo é o *Reliable Server Pooling* (RSerPool) (TÜXEN et al., 2008), que é um conjunto de protocolos para gerenciamento e acesso a vários servidores coordenados (agrupados).

Devido ao seu rico conjunto de funcionalidades, ele tende a atrair atenção para outros casos de usos também. A funcionalidade de *multihoming* provê uma confiabilidade maior, pois a associação entre dois *endpoints* pode sobreviver a falhas nos caminhos (considerando que pelo menos um caminho esteja disponível no momento) e ele vai inclusive retransmitir as mensagens que foram perdidas durante a falha do caminho se necessário for. De acordo com a RFC4960 (STEWART, 2007), somente um transporte (vide seção 2.1.3 para definição) pode ser utilizado para transmitir dados em um dado momento. Uma extensão natural

para a funcionalidade de *multihoming* é transferir dados utilizando mais de um transporte simultaneamente. Algumas propostas foram feitas (AL et al., 2004; LIAO et al., 2008; PEROTTO et al., 2007; SHAILENDRA et al., 2011) e a mais proeminente é o *Concurrent Multipath Transfer for Stream Control Transmission Protocol* (CMT-SCTP) (IYENGAR et al., 2006), que não requer novos tipos de pacotes.

Esse último identificou algumas motivações para tal desenvolvimento, dentre as quais que, com o acesso a Internet cada vez mais e mais acessível, é possível que isso motive um usuário a ter mais de uma conexão simultânea com a Internet através de várias operadoras. Ainda, é cada vez mais comum dispositivos com mais de uma interface ativa, como máquinas com uma conexão cabeada e outra sem fio. O uso de *multihoming* melhora a tolerância a falhas, porém múltiplas interfaces ativas também sugere o uso simultâneo delas para melhorar a largura de banda disponível.

O CMT-SCTP serviu, então, como base para o rascunho de RFC que herdou o seu nome (AMER et al., 2019). O trabalho de Iyengar ressaltou vários aspectos que necessitavam de alterações de maneira a lidar com a violação da suposição inicial de que somente um transporte estaria ativo por vez. Desses, focamos no gerenciamento da janela de congestionamento.

A janela de congestionamento e a janela de recepção limitam o quão rápido podemos transferir dados em um dado transporte. O primeiro é realizado no transmissor e dita quantos bytes podem estar em voo, isto é, bytes que foram enviados e que o receptor ainda não confirmou recebimento. É importante que a janela de congestionamento seja bem tratada para que, considerando que o receptor tenha o *buffer* de recepção necessário disponível, o transporte não vá funcionar aquém do esperado, ser injusto com outros usuários da rede ou desperdiçar recursos da rede por exceder a capacidade dela e causar perdas de pacote.

A RFC4960 especifica na seção 7 que apenas blocos de dados (*data chunks*) confirmados primeiramente por um ACK acumulativo podem ser considerados para aumentar a janela de congestionamento. Isso funciona bem para o caso de somente um transporte ativo em um dado momento, mas se mais transportes estiverem ativos, saltos de *selective acknowledgment* (SACK) tornam-se muito comuns. Existe a necessidade então de entender se essas lacunas são causadas por reordenação, como a reordenação em um mesmo caminho, ou se são apenas um efeito de caminhos com diferentes tempos de ida e volta (*round trip time*, RTTs) terem sido utilizados, por exemplo.

Essas lacunas foram estudadas por (IYENGAR et al., 2006) e também (PEROTTO et al., 2007) e as suas soluções propostas são bastante similares. O CMT-SCTP propôs o *Cwnd Update for CMT* (CUC). Ele funciona rastreando em qual transporte cada bloco de dados foi

enviado e, com isso, ele consegue saber se a entrega ocorreu em ordem para um dado transporte mesmo que no pacote de SACK haja saltos. Mais detalhes em como o CUC funciona podem ser encontrados na seção 2.5 na página 26.

Aprendendo com problemas que surgiram em implantações, a RFC4960 tem uma errata (STEWART et al., 2019b) e ela alterou o comportamento mencionado anteriormente. Ela passou a permitir então a consideração de blocos de dados confirmados primeiramente nas lacunas do SACK para o aumento da janela de congestionamento quando na fase de prevenção de congestionamento. Com essa alteração, parte da condição inicial que justificou o desenvolvimento do CUC, de que *Stream Control Transmission Protocol* (SCTP) (STEWART, 2007) só permitia o aumento da janela de congestionamento mediante avanço do ACK acumulativo (em qualquer fase), não se aplica mais e portanto é necessário reavaliar como o uso de múltiplos caminhos pode afetar a janela de congestionamento.

Considerando os três fatores a seguir, podemos simplificar o tratamento dos pacotes de SACK no transmissor. Primeiro, o *Split Fast Retransmit* (SFR), também proposto em (IYENGAR et al., 2006), já rastreia por transporte o maior valor de *transmission sequence number* (TSN) recebido por transporte e é capaz de perceber quando uma retransmissão rápida é necessária mesmo quando com lacunas no SACK causadas pelo uso de vários transportes simultaneamente. Segundo, que quando retransmissões rápidas ocorrem o **transporte** deve entrar em modo de recuperação rápida (o que impede o crescimento da janela). Terceiro, com esse novo entendimento da errata e utilizando esse mesmo conceito como condição para o aumento da janela de congestionamento para quando também na fase de partida lenta, podemos então simplificar o tratamento dos pacotes de SACK no transmissor e evitar o CUC por completo.

1.2 OBJETIVOS

Nesse trabalho analisamos o desempenho do *Cwnd Update for CMT version 2* (CUCv2) em nossa nova implementação do *Concurrent Multipath Transfer for Stream Control Transmission Protocol draft RFC* (CMT-SCTP) (AMER et al., 2019) na pilha do Linux, que já tinha a errata aplicada. Então removemos o CUCv2 e aplicamos o restante de nossa proposta. As propostas nesse trabalho são:

- considerar os blocos de dados confirmados em lacunas do SACK na fase de partida lenta para o aumento da janela de congestionamento;
- considerar o sinalizador de recuperação rápida por transporte,

- remover o CUC e utilizar apenas a condição de não aumentar a janela de congestionamento durante a fase de recuperação rápida para limitar seu crescimento.

1.3 ESTRUTURA DA DISSERTAÇÃO

Primeiro revisamos a história do SCTP no capítulo 2, para melhor entender a sua proposta inicial e as suas funcionalidades. Em seguida, apresentamos a terminologia SCTP e o desenvolvimento do CMT-SCTP, para então fazer uma revisão da literatura e apresentar a nossa proposta. O capítulo 3 apresenta de forma detalhada os experimentos que foram feitos para verificar a proposta. O capítulo 4 apresenta os resultados e discussões oriundas dos experimentos realizados, e o capítulo 5 trata da conclusão desses resultados.

2 FUNDAMENTAÇÃO TEÓRICA

2.1 O SCTP

2.1.1 HISTÓRIA

O SCTP é um protocolo de transporte e é um dos resultados do trabalho do grupo de trabalho SIGTRAN (derivado de *Signalling Transport*), da IETF. O SIGTRAN produziu uma série de especificações para uma família de protocolos e são uma extensão a família de protocolos SS7. Eles suportam as mesmas funcionalidades que o SS7, como gerenciamento de ligações, mas utilizam redes IP através do SCTP para transmitir a sinalização de PSTN.

SS7 é um conjunto de protocolos desenvolvido em 1975 e é responsável por estabelecer e encerrar ligações telefônicas na maioria das redes PSTN do mundo, mesmo apesar de várias adaptações locais.

O SIGTRAN foi então fortemente influenciado por engenheiros da área de telecomunicações, que queriam utilizar as novas redes IP em seus projetos. Apesar disso, cada vez mais novos usos são propostos para o SCTP e, com isso, ele está ganhando um caráter mais genérico. Inicialmente publicado em Outubro de 2000, já somam-se 23 RFCs publicadas sobre o SCTP.

Atualmente o grupo de trabalho Transport Area do IETF (TSVWG) é quem mantém protocolo SCTP.

2.1.2 FUNCIONALIDADES

O SCTP, definido pela RFC4960, é um protocolo de transporte que une funcionalidades do TCP e do UDP. Ele é orientado a conexões, similarmente ao TCP, enquanto também é orientado a mensagens, assim como o UDP. Suporta transmissão ordenada e não ordenada e provém a fragmentação das mensagens do usuário, de maneira a evitar a fragmentação a nível de protocolo de rede.

Originalmente o SCTP provém apenas transmissão de dados confiável. Ou seja, ele garante a entrega dos dados. A introdução da RFC3758 (RAMALHO et al., 2004) permite prover então também a confiança parcial. Com ela, o SCTP pode expirar mensagens que ainda estejam na fila de envio, mesmo que já tenham sido enviadas, e descartá-las. A RFC7496 (TÜXEN et al., 2015) adiciona duas políticas extras de expiração para a confiança parcial. Com elas, por exemplo, é possível instruir o SCTP para não retransmitir uma mensagem caso ela não tenha sido entregue em até 2s ou em 3 retransmissões. Nesse caso, o lado transmissor gera automaticamente um pacote especial (com um bloco *FWDTSN*) para o receptor instruindo que tais *transmission sequence number*s (TSNs) foram abortados e que não devem mais ser considerados. O receptor, nesse momento, passa a considerá-los como recebidos.

Além das funcionalidades encontradas no TCP e no UDP, ele também possui algumas funcionalidades próprias, como *multistreaming* e *multihoming*. O *multihoming* pode inclusive fazer uso de endereços IPv4 e IPv6 concomitantemente. A Fig. 1 ajuda a ilustrar a arquitetura do protocolo.

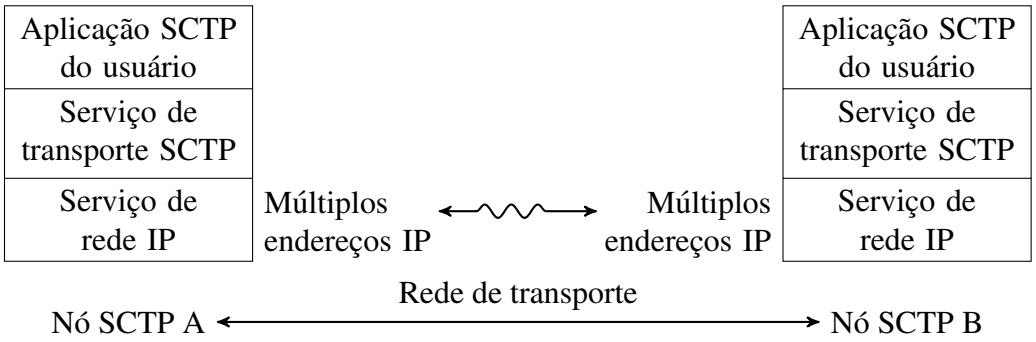

Figura 1: Uma associação SCTP

Outra diferença do SCTP para o TCP e o UDP é que ele provém mais informações para a aplicação em relação ao que se passa com a associação (vide seção 2.1.3 para definição). É possível a aplicação receber eventos informando que um dado transporte teve problemas e está temporariamente desligado, que ele retornou, que o *buffer* de envio está vazio, alterações nos endereços IP de uma associação, falhas no envio (referente a confiança parcial), entre outras.

2.1.3 TERMINOLOGIA

A terminologia do protocolo difere da do TCP principalmente de maneira a acomodar a funcionalidade de *multihoming*. Alguns novos termos são necessários, uma vez que os conceitos diferem mesmo que levemente.

Um endereço de transporte é tradicionalmente definido por um endereço de rede

(*network layer*), um protocolo de transporte (*transport layer*) e uma porta (*transport layer*). No caso do SCTP sobre IP, um endereço de transporte é definido pela informação de um endereço IP e uma porta SCTP, sendo que SCTP é o protocolo de transporte.

Um *endpoint* é um transmissor/receptor lógico de pacotes SCTP. Em um dispositivo com *multihoming*, um *endpoint* SCTP é representado aos seus pares pela combinação de um conjunto de possíveis endereços de transporte de destino, aos quais pacotes SCTP podem ser enviados, e de um conjunto de possíveis endereços de transporte de origem dos quais pacotes podem ser recebidos. Todos os endereços de transporte utilizados por uma associação SCTP devem utilizar a mesma porta, mas podem utilizar múltiplos endereços IP.

Uma associação SCTP é a relação entre *endpoints* SCTP, composta de dois *endpoints* SCTP e informação de estado do protocolo, incluindo os TSNs. Um *socket* SCTP pode conter uma (estilo 1-para-1, similar ao TCP) ou mais associações (1-para-vários, similar ao UDP).

O TSN é um número de 32 bits utilizado internamente pelo SCTP, presente nos blocos de dados enviados e que a aplicação não tem maior controle sobre. Um TSN é atribuído sequencialmente a cada mensagem do usuário (ou a cada parte dela, caso venha a ser fragmentada) para permitir que o receptor SCTP possa confirmar tal recepção e entregá-las em ordem para a aplicação, além de também ser utilizado para detectar entregas duplicadas. Há uma nova extensão que permite uma mínima manipulação do TSN por parte da aplicação. É a RFC6525 e ela permite reiniciar o TSN.

Um transporte, ou um caminho, é a rota realizada por pacotes SCTP enviados por um *endpoint* SCTP ao seu par. Cada transporte tem o seu próprio gerenciamento de janela de congestionamento, sem que haja interação entre eles. Em outras palavras, o SCTP assume que os caminhos são independentes. Já a garantia de entrega, é realizada a nível de associação. Se uma mensagem foi transmitida num transporte que teve problemas e ela não chegou a ser entregue, ela poderá ser retransmitida através de um outro transporte ativo, caso haja um.

2.2 JANELA DE CONGESTIONAMENTO

O controle de congestionamento baseado em janelas deslizantes é o mecanismo mais comum em uso hoje, de acordo com (VARMA, 2015), graças ao TCP. As janelas deslizantes são utilizadas para controlar quantos dados já foram transmitidos e que ainda não foram confirmados pelo receptor. Em outras palavras, quantos dados podem estar em voo num dado momento. As janelas podem ser definidas em bytes ou em múltiplos de pacotes, sendo que para o SCTP ela é obrigatoriamente definida em bytes.

Como veremos, em termos computacionais, o funcionamento das janelas deslizantes não é complexo e a Fig. 2 ajuda o entendimento. Nesse exemplo optamos por uma janela baseada em múltiplos de pacotes e de tamanho fixo, de 4 pacotes. Num primeiro momento, tem-se 16 pacotes a serem transmitidos, mas apenas os 4 primeiros podem ser enviados pois é o que a janela atual comporta. Somente com a chegada da confirmação de recebimento dos 2 primeiros, os próximos 2 pacotes, 5 e 6 no caso, podem ser transmitidos, e assim sucessivamente.

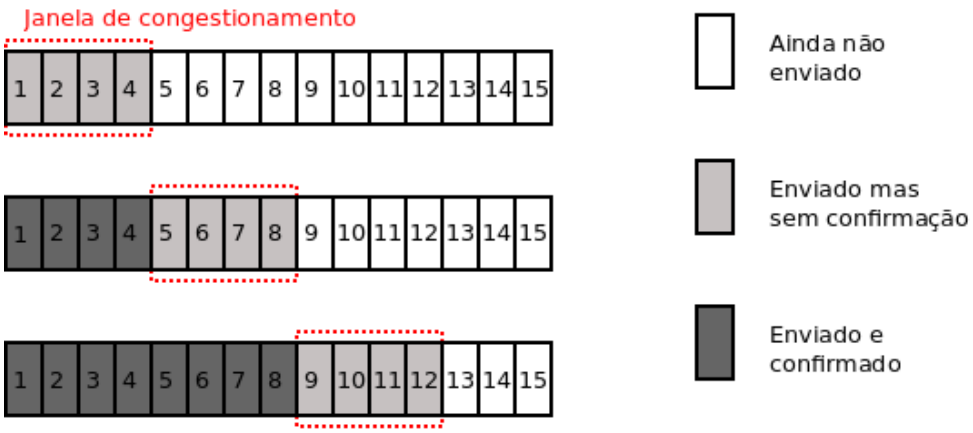

Figura 2: Operação da janela deslizante.

Podemos notar então duas variáveis que tem um grande impacto na taxa de transmissão: a latência total da conexão (também chamado de RTT) e o tamanho da janela de congestionamento. Uma janela de congestionamento muito pequena irá resultar em uma transmissão em rajadas, pois enquanto o transmissor não receber as confirmações, ele não pode enviar novos pacotes e enquanto isso os links potencialmente ficarão subutilizados. Já uma janela muito grande pode fazer com que o transmissor envie mais dados do que a rede comporta num dado momento, acarretando no acúmulo em buffers e potencialmente em perdas de pacote.

Como visto na Fig. 2, a janela de congestionamento está diretamente relacionada com o *buffer* de transmissão da aplicação e pode ser entendida como um subconjunto temporal do *buffer*. A partir daí podemos relacionar essas duas variáveis diretamente, no que chamamos de *bandwidth-delay product*, conforme a fórmula 1. Nela, `B` é a largura de banda em bits por segundo, `D` é a latência de RTT em segundos e `b` é o tamanho do *buffer*, em bits.

$$B * D = b \tag{1}$$

Esse produto calcula o mínimo de *buffer* que uma aplicação precisa para que atinja uma determinada taxa de transmissão, e é exatamente o tamanho mínimo da janela de congestionamento para que isso ocorra. Vale notar que ter mais *buffer* do que o especificado pelo produto resulta na possibilidade de obter mais largura de banda mas que se ela não for

obtida, resulta apenas em latência adicional para a aplicação.

O TCP tem uma regra básica para as confirmações de recebimento que é enviar uma confirmação pelo menos a cada dois pacotes de dados recebidos (BRADEN, 1989). Isso faz com que haja um fluxo de pacotes de confirmação no sentido contrário da transmissão e, juntamente com as próprias transmissões, eles geram o que chamamos de auto-temporização. Essa propriedade faz com que a janela deslizante seja capaz de reagir a congestionamentos na rede. Nessas situações, os pacotes demoram mais para serem transmitidos pela rede, e isso faz com que o transmissor não possa enviar o próximo pacote enquanto a confirmação (atrasada) não chegar, diminuindo assim então a taxa de transmissão.

Algoritmos utilizados atualmente, como o Cubic (RHEE et al., 2018), trabalham com janelas de congestionamento de tamanho dinâmico. Isso faz com que eles possam melhor se adequar as condições atuais, para ajudar tanto a rede quanto a aplicação em si. Há algumas regras gerais para o como e quando o tamanho da janela de congestionamento deve ser alterada, sendo que uma parte considerável depende do algoritmo em questão. Os gatilhos para as alterações são genéricos e ocorrem basicamente na confirmação de recepção de pacotes e na confirmação de perdas de pacote. Alguns algoritmos, como o Cubic, estão interessados apenas no sucesso/falha de transmissão nesses eventos, mas também há algoritmos como o Vegas (BRAKMO; PETERSON, 1995) que se utilizam desses eventos para analisar a latência de transmissão para tais pacotes. Duas operações básicas são então necessárias: aumentar e reduzir o tamanho da janela de congestionamento. Como descrito em (VARMA, 2015), existem quatro principais maneiras para isso:

- AIAD - *additive increase additive decrease* (aumento aditivo redução aditiva);
- AIMD - *additive increase multiplicative decrease* (aumento aditivo redução multiplicativa);
- MIAD - *multiplicative increase additive decrease* (aumento multiplicativo redução aditiva),
- MIMD - *multiplicative increase multiplicative decrease* (aumento multiplicativo redução multiplicativa).

Dessas, apenas a AIMD é estável e é a única utilizada, enquanto o quanto é ajustado é definido pelo algoritmo utilizado. Uma análise detalhada desses métodos foge do escopo desse trabalho e pode ser encontrada em (VARMA, 2015).

2.2.1 TRATAMENTO DA JANELA DE CONGESTIONAMENTO NA RFC4960

O tratamento da janela de congestionamento no SCTP é descrito na RFC4960 (seção 7) e é baseado no TCP.

Assim como o TCP, o aumento da janela de congestionamento não é permitido se um ACK acumulativo não avançar em um dado SACK. Isso é devido à visão convencional de que a presença de lacunas no SACK é um indicador de possíveis perdas de pacote. Também não é permitido aumentar a janela de congestionamento se ela não estiver sendo completamente utilizada ou se estiver em estado de Recuperação Rápida. Caso ela não esteja sendo completamente utilizada, significa que a janela atual é suficiente para suprir a demanda da aplicação e portanto não há a necessidade de aumentá-la. Já quando em Recuperação Rápida, o transporte está se recuperando de uma possível perda de pacote e detalharemos na seção 2.4.

Um transmissor SCTP deve manipular a janela de congestionamento por transporte. Isto é, ele precisa ter um conjunto de variáveis para cada transporte. Porém, de acordo com a RFC4960 seção 7.2.4 item 6, o sinalizador de recuperação rápida e seu ponto de saída são globais para a associação.

2.3 PARTIDA LENTA E PREVENÇÃO DE CONGESTIONAMENTO

O aumento da janela no SCTP é dividido nas mesmas duas fases que no TCP, partida lenta e prevenção de congestionamento. A primeira é responsável pela busca rápida da largura de banda disponível e a segunda por manter um ponto mais estável, ainda buscando por mais largura de banda mas cautelosamente.

Apesar do nome, a partida lenta tem um comportamento bastante agressivo. O crescimento da janela nessa fase é de maneira exponencial: para cada confirmação recebida que ocupou um segmento inteiro, a janela cresce mais um segmento. Nessa fase, ela é atualizada de acordo com a Eq. 2 a seguir:

$$W = W + 1 \tag{2}$$

Em que W é em unidades de pacotes. Por exemplo, num dado momento, 4 pacotes estão em voo. Quando esses 4 tiverem sido confirmados, o transmissor poderá enviar até 8 pacotes, pois para cada pacote confirmado, além dele um pacote adicional poderá ser enviado, e assim sucessivamente. O nome se dá pois essa fase foi proposta em (JACOBSON, 1988) para corrigir um problema de perdas de pacote que havia sem essa fase. Sem ela, ao iniciar,

o transmissor despejava o equivalente a toda janela de congestionamento logo no início da conexão, possivelmente saturando buffers ao longo da rota. Então comparado com esse método, a partida lenta é, de fato, mais lenta.

Como ela depende muito do retorno de pacotes, o desempenho da partida lenta é inversamente proporcional ao RTT da conexão. Conexões com maiores latências demoram mais para sair da fase de partida lenta pois cada iteração é mais alongada.

O fim da fase de partida lenta se dá quando a janela de congestionamento atinge um tamanho igual ou maior ao limiar de partida lenta, conhecido como *ssthresh* (*Slow Start Threshold*). Ao atingir esse valor, o aumento da janela passa a ser feito através da fase de prevenção de congestionamento. Quando perdas de pacote ocorrem e são recuperadas através do esgotamento de tempo, a metade do valor atual da janela de congestionamento é atribuído a *ssthresh* e a janela é configurada a 1 unidade máxima de transmissão (*Maximum Transmission Unit*, MTU). Dessa maneira, quando a janela de congestionamento atingir esse valor novamente, ela será atualizada linearmente. O *sshthresh* normalmente é inicializado a 64 KBytes no início da conexão, e é o caso da implementação do SCTP no Linux. A janela de congestionamento pode nunca alcançar esse valor e isso é comum principalmente quando o fator limitante é a aplicação em si. Afinal, nem todas as aplicações necessitam utilizar toda a largura de banda que a rede tem disponível.

Durante a fase de prevenção de congestionamento o incremento da janela de congestionamento é feito através da Eq. 3 a seguir:

$$W = W + 1/W \tag{3}$$

Sendo que W é em unidades de pacotes. Portanto ela irá crescer apenas 1 pacote quando toda a janela de congestionamento tiver sido transmitida, o que pode levar um ou mais RTTs.

Na Fig. 3 na seção a seguir é possível observar o desenvolvimento dessas duas fases no tempo.

2.4 RETRANSMISSÃO RÁPIDA E RECUPERAÇÃO RÁPIDA

O TCP tem dois métodos básicos para recuperação de perdas de pacote: por esgotamento de tempo e por retransmissão rápida, que foi proposta mais tarde. Um temporizador é armado para o pacote mais antigo em voo e ele expira em um tempo

proporcional ao RTT. Inicialmente ele expirava em um múltiplo direto do RTT, mas depois de (JACOBSON, 1988) é utilizado uma média amortecida dele e o desvio médio amortecido dessa.

A recuperação por esgotamento de tempo é bastante impactante pois ela demora para ser acionada e, quando é, o tamanho da janela de congestionamento é forçado para o equivalente a 1 pacote. Ainda, caso essa retransmissão não consiga corrigir o problema, o tempo de retransmissão é dobrado. A transmissão, portanto, sofre uma grande penalidade.

A retransmissão rápida foi então proposta e ela é acionada muito mais rapidamente. Quando o transmissor recebe 3 ACKs duplicados, indicando assim uma possível perda de pacote, ele entra em modo de recuperação rápida e já retransmite o pacote seguinte ao indicado.

No TCP Reno (VARMA, 2015), ao entrar no modo de recuperação rápida, o transmissor reduz o tamanho da janela de congestionamento para a metade da atual, guarda esse valor em *ssthresh* também e, após a retransmissão, ajusta novamente a janela de congestionamento para o novo valor atual mais o equivalente a 3 pacotes. Para cada novo ACK duplicado recebido, ele aumenta a janela de congestionamento de acordo com o tamanho do pacote e, se a nova janela permitir, ele transmite um novo pacote. Já no SCTP, essa compensação dos 3 ACKs duplicados não ocorre e de acordo com a seção 7.2.3 da RFC4960 (STEWART, 2007) o limiar de partida lenta é atualizado com a Eq. 4. Dessa maneira, é garantido um valor mínimo para *ssthresh* de $4 * MTU$.

$$ssthresh \leftarrow max(cwnd/2, 4 * MTU) \tag{4}$$

Quando o receptor confirmar o recebimento do pacote retransmitido e então o ponto de ACK acumulativo alcançar o ponto de saída da recuperação rápida, o transmissor sai do modo de recuperação rápida. Note como o tamanho resultante da janela de congestionamento é consideravelmente maior de que quando a recuperação é feita através de esgotamento de tempo.

A RFC4960, na seção 7.2.4, especifica que quando um ACK parcial é recebido durante a recuperação rápida, a indicação de perda (contador de ACKs duplicados) para todos os TSNs não confirmados é incrementada e que, quando a atingirem a contagem de 3, são marcados para retransmissão rápida. Essa descrição é similar ao funcionamento do TCP Reno e é como o Linux implementa a pilha SCTP.

Vale salientar aqui que se a retransmissão rápida também for perdida apenas o esgotamento de tempo será capaz de efetuar a recuperação. A RFC4960 sugere um valor mínimo de tempo limite de retransmissão (*retransmit time out*, RTO) de 1 segundo,

independentemente do RTT.

O SCTP conta com os dois métodos de recuperação desde a sua primeira especificação.

A Fig. 3 ajuda a ilustrar as fases do controle de congestionamento e a recuperação rápida.

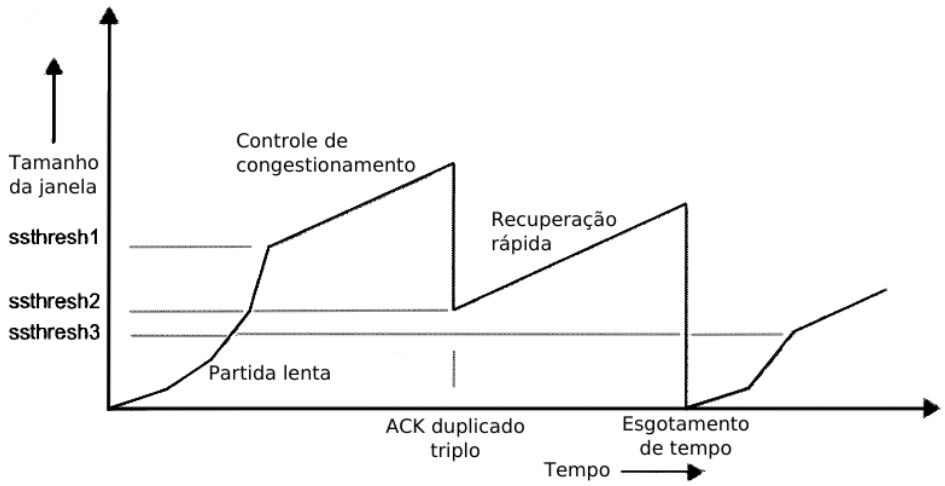

Figura 3: Fases do controle de congestionamento.

2.5 CMT-SCTP E O ALGORITMO CUC

Como os SACKs são efetivos para a associação toda, quando utilizando vários transportes ao mesmo tempo torna-se ambíguo do ponto de vista da RFC4960 se uma lacuna é uma indicação de perda de pacote ou se foi uma consequência natural. Isto é, se é apenas uma consequência da utilização simultânea de múltiplos caminhos.

O CMT-SCTP reconhece o problema e propõe um algoritmo que é capaz de isolar a reordenação induzida pelo transmissor e, com isso, permite ao transmissor aumentar a janela de congestionamento como a RFC4960 especifica. O algoritmo é chamado de CUC e a sua segunda versão consegue rastrear inclusive retransmissões feitas em um transporte diferente do original. Nota que (IYENGAR et al., 2006) especifica 5 políticas de retransmissão e que elas podem ocorrer em diferentes transportes mesmo que não seja durante uma falha de um transporte.

O algoritmo se baseia no rastreamento do menor TSN ainda não confirmado pelo receptor por transporte e atualiza a janela de congestionamento, mesmo na ausência de um novo ACK acumulativo no pacote SACK recebido. Esse TSN, então, é utilizado como o pseudo_cumack (ou rtx_pseudo_cumack no caso de retransmissões por diferentes caminhos, no

CUCv2) para o destino, e a decisão sobre o aumento da janela de congestionamento é realizada baseada nele.

Em outras palavras, ele calcula um novo ACK acumulativo para cada destino, que então é utilizado somente para identificar se é permitido aumentar a janela de congestionamento, e mantém todo o restante do tratamento da janela como anteriormente. É uma alteração específica do CMT-SCTP de maneira a acomodar um requisito originário da RFC4960. Nota que esses pseudo *cumacks* não podem ser utilizados para limpar a fila de retransmissão.

Para melhor evidenciarmos a complexidade dos algoritmos CUC e CUCv2, eles estão representados a seguir na Fig. 4 e Fig. 5, respectivamente. Ressaltamos que ele é uma nova forma de ver a mesma condição para o aumento da janela de congestionamento. Ele apenas contextualiza a restrição inicial para o CMT-SCTP.

At beginning of an association [Sender side behavior]:
∀ destinations d, reset
d.find_pseudo_cumack = TRUE;
On receipt of a SACK [Sender side behavior]:
1) ∀ destinations d, reset *d.new_pseudo_cumack* = FALSE;
2) **if** the SACK carries a new cum ack **then**
for each TSN t_c being cum acked for the first time, that was not acked through prior gap reports **do**
(i) let d_c be the destination to which t_c was sent;
(ii) set $d_c.find\_pseudo\_cumack$ = TRUE;
(iii) set $d_c.new\_pseudo\_cumack$ = TRUE;
3) **if** gap reports are present in the SACK **then**
for each TSN t_p being processed from the retransmission queue **do**
(i) let d_p be the destination to which t_p was sent;
(ii) **if** ($d_p.find\_pseudo\_cumack$ = TRUE) **and** t_p was not acked in the past **then**
$d_p.pseudo\_cumack = t_p$;
$d_p.find\_pseudo\_cumack$ = FALSE;
(iii) **if** t_p is acked via gap reports for first time **and** ($d_p.pseudo\_cumack = t_p$) **then**
$d_p.new\_pseudo\_cumack$ = TRUE;
$d_p.find\_pseudo\_cumack$ = TRUE;
4) **for** each destination d **do**
if (*d.new_pseudo_cumack* = TRUE) **then** update cwnd [1], [2];

Figura 4: Algoritmo CUC, como proposto por (IYENGAR et al., 2006). Imagem de (IYENGAR et al., 2006)

2.6 DESENVOLVIMENTOS RECENTES DA RFC4960

A errata da RFC4960 (seção 3.22) reconhece que blocos de dados confirmados primeiramente em uma lacuna devem sim ser considerados para o aumento da janela de congestionamento mesmo quando o ACK acumulativo não tenha avançado, pois eles indicam a recepção bem sucedida de tais blocos. Ela faz isso porém somente para quando na fase de

At beginning of an association [Sender side behavior]:
∀ destinations d, reset
d.find_pseudo_cumack = *d.find_rtx_pseudo_cumack* = TRUE;
On receipt of a SACK [Sender side behavior]:
1) ∀ destinations d, reset
d.new_pseudo_cumack = *d.new_rtx_pseudo_cumack* = FALSE;
2) **if** the ack carries a new cum ack **then**
for each TSN t_c being cum acked for the first time, that was not acked through prior gap reports **do**
(i) let d_c be the destination to which t_c was sent;
(ii) set $d_c.find\_pseudo\_cumack$ = $d_c.find\_rtx\_pseudo\_cumack$ = TRUE;
(iii) set $d_c.new\_pseudo\_cumack$ = $d_c.new\_rtx\_pseudo\_cumack$ = TRUE;
3) **if** gap reports are present in the ack **then**
for each TSN t_p being processed from the retransmission queue **do**
(i) let d_p be the destination to which t_p was sent;
(ii) **if** ($d_p.find\_pseudo\_cumack$ = TRUE) **and** t_p was not acked in the past **and** t_p was not retransmitted **then**
$d_p.pseudo\_cumack$ = t_p;
$d_p.find\_pseudo\_cumack$ = FALSE;
(iii) **if** t_p is acked via gap reports for first time **and** ($d_p.pseudo\_cumack$ = t_p) **then**
$d_p.new\_pseudo\_cumack$ = TRUE;
$d_p.find\_pseudo\_cumack$ = TRUE;
(iv) **if** ($d_p.find\_rtx\_pseudo\_cumack$ = TRUE) **and** t_p was not acked in the past **and** t_p was retransmitted **then**
$d_p.rtx\_pseudo\_cumack$ = t_p;
$d_p.find\_rtx\_pseudo\_cumack$ = FALSE;
(v) **if** t_p is acked via gap reports for first time **and** ($d_p.rtx\_pseudo\_cumack$ = t_p) **then**
$d_p.new\_rtx\_pseudo\_cumack$ = TRUE;
$d_p.find\_rtx\_pseudo\_cumack$ = TRUE;
4) **for** each destination d **do**
if (*d.new_pseudo_cumack* = TRUE) **or** (*d.new_rtx_pseudo_cumack* = TRUE) **then**
Update cwnd [1], [2];

Figura 5: Algoritmo CUCv2, como proposto por (IYENGAR et al., 2006). A versão 2 permite o crescimento da janela de congestionamento quando retransmissões são feitas em outros transportes além do transporte originalmente utilizado. Imagem de (IYENGAR et al., 2006)

prevenção de congestionamento e mantém a fase de partida lenta inalterada.

Ainda, isso significa que o algoritmo CUC agora só é necessário na fase de partida lenta, pois não há mais a necessidade de rastrear o progresso do ACK acumulativo por transporte quando na fase de prevenção de congestionamento.

Convém lembrarmos que a RFC4960bis (STEWART et al., 2019a) em desenvolvimento é baseada na RFC4960 e deverá contar com a aplicação de suas erratas até então, caso um dia aprovada.

2.7 REVISÃO DA LITERATURA

O LS-SCTP (AL et al., 2004), o cmpSCTP (LIAO et al., 2008) e o MPSCTP (SHAILENDRA et al., 2011) mitigam o problema causado pelo reordenamento com o uso de vários caminhos através da utilização de dois números de sequência, sendo um por associação e outro por caminho. Já (PEROTTO et al., 2007) utilizou um algoritmo similar ao CUC.

Todas as propostas de múltiplos caminhos então citadas foram baseadas na restrição de que a janela de congestionamento não pode ser aumentada se não houver o avanço do ACK acumulativo. Considerando que essa restrição está presente na RFC4960 desde a sua especificação, é esperado que muitos artigos tenham dependido dela, incluindo o próprio CMT-SCTP.

A errata data de Março de 2016 e não pudemos encontrar algum artigo já considerando a mesma. Artigos mais antigos também não consideraram a ideia, como em (WALLACE; SHAMI, 2014). Artigos que trabalham especificamente com controle de congestionamento, como (LAN; SHA, 2011) e o já mencionado (PEROTTO et al., 2007), também utilizaram essa restrição do ACK acumulativo.

Nem (IYENGAR et al., 2006) ou (DREIBHOLZ et al., 2010a) indicaram ter estudado a possibilidade de tratar a sinalização de recuperação rápida por transporte.

Publicamos um artigo (LEITNER et al., 2019) no ACM/SIGAPP SAC 2019 com nossa proposta. No melhor de nosso conhecimento, somos os primeiros a verificar essa combinação.

3 PROPOSTA

O raciocínio utilizado pela errata na fase de prevenção de congestionamento também é válido para a fase de partida lenta: blocos de dados confirmados em lacunas foram bem recebidos. A diferença entre as duas fases é que o aumento na de partida lenta é muito mais agressivo e que a presença das lacunas tem maior probabilidade de indicar reais perdas de pacote.

Se considerarmos que a retransmissão rápida já faz um bom trabalho na identificação de tais perdas, que a sua atualização SFR é capaz de detectar melhor tais perdas no cenário CMT, e que eles rastreiam TSNs primeiramente confirmados em lacunas, nós podemos aproveitar a retransmissão rápida como um fator condicionador para o aumento da janela de congestionamento.

RFC4960 diz que sempre que a retransmissão rápida for ativada, a **associação** deve entrar em modo de recuperação rápida. Ao invés de depender de um ACK acumulativo, o que requer um outro mecanismo de rastreio de SACK (como o CUC), podemos utilizar o estado de recuperação rápida como um fator de bloqueio para o aumento da janela de congestionamento: sempre que um TSN for recentemente confirmado, a janela de congestionamento pode aumentar somente se o **transporte** não estiver no estado de recuperação rápida.

Um indicador de recuperação rápida para toda a associação não é ótimo. Não apenas para durante a recuperação em si mas também para a reação a congestionamentos. O item 2 da seção 7.2.4 da RFC4960 impede a redução da janela de congestionamento em outro transporte se a associação já estiver no estado de recuperação rápida, possivelmente causando perdas evitáveis de pacotes.

Apenas essa alteração, independente de considerar os TSNs confirmados nas lacunas, já torna o aumento mais agressivo pois ele vai reagir mais tardiamente que o comportamento original. Por também considerar os TSNs primeiramente confirmados em lacunas, ele vai ser ainda mais agressivo mas sem prejudicar a rede, pois perdas de pacote ainda vão acionar os freios no aumento da janela de congestionamento.

Com essas duas simples alterações o CMT-SCTP é capaz de trabalhar com ou sem a reordenação induzida pelo transmissor e sem a ajuda do CUC.

Vale salientarmos que essa agressividade é modesta e que o resultado esperado é que os desempenhos dos dois métodos sejam próximos. No caso de não haver perdas de pacote, o desempenho esperado é o mesmo que com a utilização do CUC. Já no caso com perdas recuperadas com a retransmissão rápida durante a fase de partida lenta, a janela de congestionamento será aumentada em 3 pacotes a mais (somente 3 pois a partir de então o transporte entra em modo de recuperação rápida e o crescimento da janela é proibido). A Fig. 6 exemplifica tal situação. Caso em fase de prevenção de congestionamento, o comportamento também deve ser o mesmo.

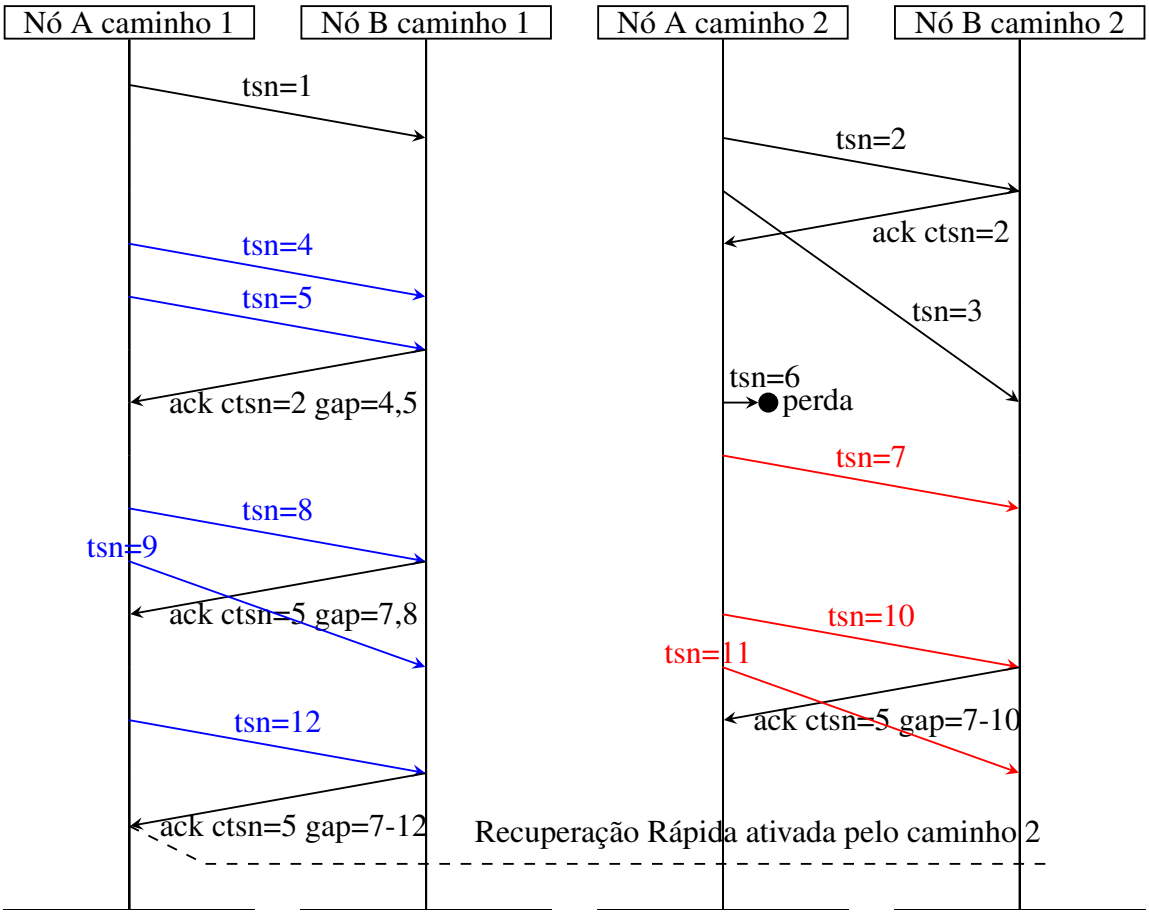

Figura 6: Exemplo evidenciando as diferenças da visão tradicional, do CUC e da nossa proposta em relação a quais TSNs podem ser utilizados para o crescimento da janela de congestionamento. Em preto os TSNs aceitos pela visão tradicional, em azul os aceitos pelo CUC e em vermelho os aceitos pela nossa proposta. ”tsn” é o número sequencial dos pacotes, ”ctsn” se refere ao valor de ACK acumulativo no pacote, enquanto ”gap” se refere aos TSNs confirmados em lacunas.

Na Fig. 6 podemos observar em preto os TSNs que são aceitos para o crescimento

da janela na visão tradicional. Isto é, apenas os TSNs 1, 2 e 3 são considerados em seus respectivos caminhos até que a perda seja recuperada. Isso porque o TSN 3 demorou para chegar no receptor, o que fez com que os demais TSNs do caminho 1 fossem confirmados em lacunas e no caminho 2 houve uma perda logo após o TSN 3.

Com o CUC, os TSNs em azul (4, 5, 8, 9 e 12) também passam a ser considerados, uma vez que o *pseudo_cumack* do caminho 1 é avançado com todos eles inclusos. O caminho 1, então, até o ponto em que a Recuperação Rápida é ativada, funciona como se fosse uma transmissão em um único caminho. Já no caminho 2, o *pseudo_cumack* dele não avança a partir do TSN 3 devido a perda do TSN 6, o que faz com que os TSNs 7, 10 e 11 não sejam considerados para o crescimento da janela. Porém ao atingir tal ponto, alterações na janela de congestionamento do caminho 1 também é proibido, uma vez que a Recuperação Rápida é por associação.

Com a nossa proposta, além dos TSNs aceitos pelo CUC, os TSNs em vermelho (7, 10 e 11) também passam a ser considerados, mesmo que o TSN 6 tenha sido perdido e independentemente de em qual fase o controle de congestionamento do transporte 2 se encontra. Já no ponto em que a Recuperação Rápida é ativada, apenas o transporte 2 entra nesse modo e então alterações na janela de congestionamento dele são proibidos. No transporte 1, elas seguem inalteradas.

3.1 ALGORITMOS ALTERADOS

A RFC4960 (STEWART, 2007) define o crescimento da janela de congestionamento da seguinte maneira:

> 7.2.1. Partida lenta
>
> Iniciar a transmissão de dados em uma rede com condições desconhecidas ou após um período de silêncio suficientemente longo requer que o SCTP identifique as condições da rede para determinar a sua capacidade disponível. O algoritmo de partida lenta é utilizado para esse propósito no início de uma transferência, ou após correções de perdas detectadas por esgotamento de tempo.
>
> - A janela de congestionamento inicial antes da transmissão de dados ou após um período de silêncio suficientemente grande deve ser de `mínimo(4*MTU, máximo(2*MTU, 4380 bytes))`.
> - A janela de congestionamento inicial após uma retransmissão por esgotamento de tempo deve ser de não mais do que `1*MTU`.
> - O valor inicial de `ssthresh` pode ser arbitrariamente alto (por exemplo, as implementações podem utilizar o tamanho da janela de recepção anunciada pelo par).

> - Sempre que a janela de congestionamento for maior que zero, é permitido ao *endpoint* ter `cwnd` bytes de dados pendentes naquele endereço de transporte.
> - Quando a janela de congestionamento for menor ou igual que o `ssthresh`, um *endpoint* SCTP deve usar o algoritmo de partida lenta para aumentar a janela de congestionamento se a janela atual está sendo completamente utilizada, **um SACK recebido avança o ponto de ACK acumulativo**, e o transmissor de dados não está em modo de Recuperação Rápida. Somente quando essas três condições são atendidas é que a janela de congestionamento pode ser aumentada; caso contrário, a janela não deve ser aumentada. Se essas condições forem atendidas, então a janela de congestionamento deve ser aumentada por, no máximo, o menor de 1) o tamanho total dos dados previamente pendentes então confirmados, e 2) o unidade máxima de transmissão do caminho (*Path Maximum Transmission Unit*, PMTU) do destino. Esse limite superior protege contra ataques do tipo *ACK-Splitting*.
>
> Em instâncias nas quais o par usa *multihoming*, se um *endpoint* receber um SACK que avance o seu ponto de ACK acumulativo, então ele deve atualizar a sua janela de congestionamento (ou janelas, caso mais de uma) proporcionalmente aos endereços de destino aos quais transmitiu os dados confirmados. No entanto, se o SACK recebido não avançar o ponto de ACK acumulativo, o `endpoint` não deve ajustar a janela de congestionamento de qualquer endereço de destino.
>
> Porque a janela de congestionamento de um *endpoint* não é atrelada a seu ponto de ACK acumulativo, conforme a recepção de SACKs, mesmo apesar que eles não avancem o ponto de ACK acumulativo um *endpoint* ainda pode usá-los como base de tempo para enviar novos dados. Isto é, os dados recém confirmados pelo SACK diminui a quantidade de dados que estão em voo para menos do que a janela de congestionamento, e então a atual, não modificada janela de congestionamento agora permite que novos dados sejam enviados. Por outro lado, o aumento da janela de congestionamento precisa estar atrelado ao avanço do ponto de ACK acumulativo como especificado acima. Caso contrário, os SACKs duplicados não somente vão permitir o envio de novos dados, como vão também permitir o envio de mais dados novos em maior quantidade do que saíram da rede, durante um período de possível congestionamento.
>
> - Quando um *endpoint* não transmitir dados em um dado endereço de transporte, a janela de congestionamento do endereço de transporte deve ser ajustada para `máximo(cwnd/2, 4*MTU)` por RTO.
>
> (Tradução livre da seção 7.2.1 da RFC4960 (STEWART, 2007))

A condição destacada em negrito é a que propomos remover, assim como a errata já fez com a fase de prevenção de congestionamento.

Em relação ao acionamento da Recuperação Rápida, a RFC4960 especifica:

> 7.2.4. Retransmissão rápida em relatórios de lacunas
>
> Na ausência de perda de dados, um *endpoint* usa confirmações atrasadas. Porém, quando um *endpoint* percebe um buraco na sequência de TSNs recebida, ele deve iniciar o retorno de um SACK toda vez que um pacote chegar com dados até que o buraco seja preenchido.
>
> Sempre que um *endpoint* receber um SACK que indique que alguns TSNs estão faltando, ele deve esperar duas outras indicações de ausências (através de SACKs

subsequentes para um total de três indicações de ausência) nos mesmos TSNs antes de tomar ação com relação a Retransmissão Rápida.

Indicações de ausência devem seguir o TSN mais alto recém confirmado (*Highest TSN Newly Acknowledged*, HTNA). Para cada SACK entrante, indicações de perdas são incrementadas somente para os TSNs ausentes anteriores ao TSN mais alto recém confirmado no SACK. Um bloco de dados recém confirmado é um que não havia sido confirmado num SACK até então. Se um *endpoint* está em modo de Recuperação Rápida e um SACK recebido tal que avança o ponto de ACK acumulativo, as indicações de perdas são incrementadas para todos os TSNs relatados ausentes no SACK.

Quando a terceira indicação consecutiva de ausência é recebida para dados TSNs, o transmissor dos dados deve proceder da seguinte maneira:

1. Marcar os blocos de dados com três indicações de perda para retransmissão.
2. Se não estiver em modo de Recuperação Rápida, ajustar o `ssthresh` e a janela de congestionamento dos endereço(s) de destino ao(s) qual(ais) os dados faltantes foram transmitidos por último, de acordo com a fórmula descrita na seção 7.2.3.
3. Determinar quantos dos dados mais antigos (ou seja, menor TSN) marcos para retransmissão caberão em um único pacote, sujeito a restrição do PMTU do transporte de destino ao qual o pacote setá sendo enviado. Chamar esse valor de `K`. Retransmitir esses `K` blocos de dados em um único pacote. Quando uma Retransmissão Rápida está sendo executada, o transmissor deve ignorar o valor da janela de congestionamento e não deve atrasar a retransmissão para este único pacote.
4. Reiniciar o temporizador `T3-rtx` somente se o último SACK recebido confirmou a recepção do TSN mais baixo ausente enviado para aquele endereço, ou o *endpoint* está retransmitindo o primeiro bloco de dados enviado para aquele endereço.
5. Marcar os blocos de dados como sendo retransmitidos rapidamente e portanto inelegíveis para novas retransmissões rápidas. Aqueles TSNs marcados para retransmissão pelo algoritmo de Retransmissão Rápida que não couberam no datagrama enviado carregando `K` outros TSNs também são marcados como inelegíveis para uma retransmissão rápida sucessiva. Contudo, como eles estão marcados para retransmissão, eles serão retransmitidos assim que a janela de congestionamento permitir.
6. Se não estiver em estado de Recuperação Rápida, entrar nele e registrar o TSN mais alto ausente como o ponto de saída da Recuperação Rápida. Quando um SACK confirmar todos os TSNs até e incluindo esse ponto de saída, o modo de Recuperação Rápida é desligado. Quando em Recuperação Rápida, o `ssthresh` e a janela de congestionamento não devem ser alteradas para **quaisquer** destinos devido a um evento de Recuperação Rápida subsequente (isto é, o transmissor não deve reduzir mais a janela de congestionamento devido uma retransmissão rápida subsequente).

Nota: antes dos ajustes acima, se o SACK recebido também confirmar a recepção de novos blocos de dados e avançar o ponto de ACK acumulativo, as regras para aumento da janela de congestionamento definidas na seção 7.2.1 e seção 7.2.2 devem ser aplicadas antes.

Uma implementação direta dos passos acima é manter um contador para cada lacuna de TSN relatada por um SACK. O contador incrementa para cada SACK consecutivo relatando essa lacuna de TSN. Após alcançar 3 e iniciar o procedimento de Recuperação Rápida, o contador é ajustado para 0.

Devido a janela de congestionamento no SCTP controlar indiretamente o número de TSNs em trânsito, o efeito da Recuperação Rápida do TCP é alcançado

> automaticamente sem sejam necessários ajustes ao tamanho da janela de controle de congestionamento.
>
> (Tradução livre da seção 7.2.4 da RFC4960 (STEWART, 2007))

A palavra ”quaisquer” destacada em negrito no item 6 faz com que tal evento em um transporte afete os demais, o que não é desejado em um ambiente de CMT. Mesmo acompanhada do termo ”não devem”, que torna esse comportamento opcional por implementação, nossa proposta é para que essa recomendação seja apenas para o transporte em questão. Assim fica evidenciado que um acionamento de Recuperação Rápida não deve afetar demais transportes de uma associação. Ainda, a nota dessa seção da RFC4960 também necessita ser alterada, mas devido a alteração anterior.

A terceira alteração é a não implementação por completo do algoritmo CUC, que está disponível para referência na Fig. 5, na pág. 28.

3.2 IMPLEMENTAÇÃO

A implementação foi baseada no kernel Linux v4.18, o qual não conta com nenhum suporte ao CMT-SCTP. Para tal, algumas alterações preliminares precisaram ser feitas e vamos descrevê-las nessa seção.

Para não violar as camadas da implementação do SCTP, foi preciso mover o controle de quando um pacote é tido como cheio da camada de envio de pacote (função `sctp_packet_transmit_chunk` no arquivo `output.c`) para a camada de fila de saída (`outqueue.c`). Nisso, aproveitamos para renomeá-la para `sctp_outq_transmit_chunk` e passar a trabalhar com a estrutura `sctp_flush_ctx` diretamente.

O kernel Linux v4.18 conta com uma implementação de *Split Fast Retransmit Changeover Aware Congestion Control* (SFR-CACC) (IYENGAR, 2005), a qual substituímos integralmente pelo SFR proposto em (IYENGAR et al., 2006). Para isso, removemos os campos referentes ao SFR-CACC da estrutura `sctp_transport.cacc` e adicionamos os campos do SFR no lugar. Toda a parte referente ao SFR-CACC foi removida da função `sctp_assoc_set_primary`. As funções `sctp_cacc_skip_3_1_d`, `sctp_cacc_skip_3_1_f`, `sctp_cacc_skip_3_1`, `sctp_cacc_skip_3_2` e `sctp_cacc_skip` foram removidas. Já as funções `sctp_outq_sack`, `sctp_check_transmitted`, `sctp_mark_missing` e `sctp_transport_reset` precisaram ser adaptadas para implementar o novo algoritmo.

A implementação do *Delayed ACK for CMT* (DAC) (IYENGAR et al., 2006) foi feita em duas etapas. Primeiro, implementamos o lado do receptor. Na estrutura `sctp_association` incluímos um campo para a contagem de quantos pacotes de dados do protocolo (*Protocol Data Unit*, PDUs) foram recebidos no intervalo entre-SACKs. Alteramos a função `sctp_make_sack` para que ela incluísse esse contador no campo de sinalizações do bloco. Desabilitamos o envio de SACK imediato na função `sctp_gen_sack`, apenas comentando o código. Enfim, alteramos a função `sctp_eat_data` para incrementar tal contador ao recebimento de cada PDU.

Para o lado transmissor do DAC, alteramos a função `sctp_check_transmitted` para retornar verdadeiro caso o transporte tenha tido algum bloco de dados confirmado nesse SACK. Com isso, a função `sctp_outq_sack` pode contar quantos transportes tiveram novas confirmações no dado SACK e então implementar as condições do DAC na função `sctp_mark_missing`.

O CMT-SCTP não trata de como escolher por qual caminho enviar os pacotes. Esse é um assunto complexo por natureza e ele tem agravantes para o SCTP, como por exemplo o fato de ter *multistreaming* e de ter um único controle de entrega para a associação toda. Como demonstrado em (DREIBHOLZ et al., 2010b) a utilização do *multistreaming* juntamente com uma política avançada de escalonamento de *streams* pode melhorar o desempenho do CMT-SCTP percebido pela aplicação. Nossa implementação, portanto, seguiu um modelo básico e que acreditamos suficiente para verificarmos a nossa proposta. Optamos por fazer o SCTP escolher, a cada pacote enviado, o transporte com a maior janela de congestionamento disponível. Para isso, criamos a função `sctp_outq_select_next_transport`, que localiza tal transporte e prepara a associação para utlizá-lo no próximo pacote a ser enviado. Ela é chamada, portanto, ao final da função `sctp_outq_transmit_packet`.

Optamos por trabalhar com o CUCv2, mesmo que nossa implementação tenha utilizado apenas a política de retransmissão RTX-SAME. Para tal, adicionamos seis novos campos na estrutura `sctp_transport.sfr`: `pseudo_cumack`, `rtx_pseudo_cumack`, `new_pseudo_cumack`, `new_rtx_pseudo_cumack`, `find_pseudo_cumack` e `find_rtx_pseudo_cumack`, todos conforme previsto no CUCv2. Alteramos a função `sctp_acked` para que ela indique se o TSN em questão foi confirmado através de uma lacuna. O passo 1 do algoritmo foi implementado na função `sctp_outq_sack`. Os passos 2 e 3, na função `sctp_check_transmitted`. Já o passo 4, foi implementado na função `sctp_transport_raise_cwnd`. Nota que o kernel Linux v4.18 já conta com a implementação de nossa proposta em relação a permitir o aumento da

janela de congestionamento na ausência de um novo ACK acumulativo mesmo na fase de partida lenta. A integração do CUCv2, portanto, efetivamente reverte essa característica.

Por fim, tornamos a Recuperação Rápida por transporte. Movemos os campos `fast_recovery` e `fast_recovery_exit` da estrutura `sctp_assiciation` para a estrutura `sctp_transport` e atualizamos os lugares que lidam com eles, como as funções `sctp_transport_raise_cwnd` e `sctp_transport_lower_cwnd`.

O código foi validado através da análise manual de captura de tráfego feita em testes menores em conjunto com registro de ativação de funções importantes, como `sctp_transport_raise_cwnd` e `sctp_transport_lower_cwnd`. Com isso, pudemos acompanhar a distribuição dos pacotes pelos dois caminhos e como eles influenciavam as janelas de congestionamento. Num primeiro momento verificamos sem perdas em ambos os caminhos. Então introduzimos perdas de 1% em um dos caminhos e, em seguida, em ambos.

4 AVALIAÇÃO DE DESEMPENHO

Nesta seção apresentamos a avaliação de desempenho da solução proposta. Descreveremos o ambiente de testes, testes realizados e os seus resultados.

4.1 AMBIENTE DE TESTES

Os testes foram conduzidos em um kernel Linux v4.18 alterado com a nossa implementação da extensão CMT. Utilizamos *network namespaces* para isolar o transmissor e o receptor e *NetEm* para emular as características de uma ligação, como vazão máxima, latência e perdas de pacote. Dos modelos de perdas suportados pelo *NetEm* (LUDOVICI; PFEIFER, 2011), utilizamos o modelo de Bernoulli (não testamos outros modelos). O modelo de Bernoulli é um caso particular do modelo de Gilbert-Elliot e seu único parâmetro é a probabilidade de perda, cujos valores utilizados estão representados tanto na Fig. 7 quanto nos resultados na seção 4.2.

A topologia utilizada pode ser vista na Fig. 7. Os túneis *veth* que conectam os *namespaces* já proveem isolamento suficiente e não havia a necessidade de adicionar saltos intermediários entre eles.

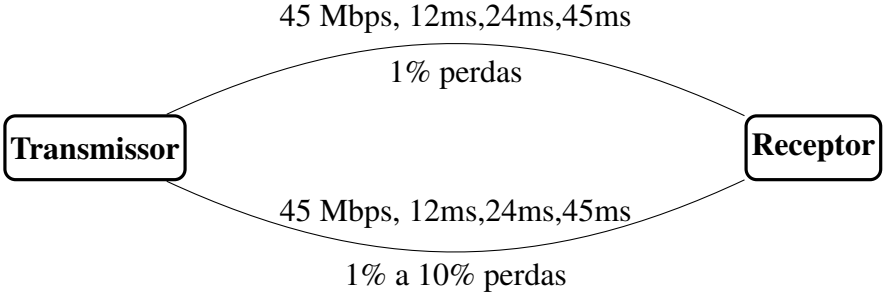

Figura 7: Topologia de simulação utilizada para a avaliação. Cada ligação é um túnel *veth* através de *network namespaces* com *NetEm* em ambos os lados.

Contenção de CPU não foi um problema durante os testes. Limitando a vazão a algumas dezenas de megabits por segundo e não executando muitos testes paralelamente

mitigou essa possibilidade.

Cada *network namespace* foi criado com o comando:

```
ip netns add <nome>
```

Cada conexão entre eles foi feita com:

```
ip netns exec <netnsA> ip link add <ifA> type veth \
        peer name <ifB>
ip netns exec <netnsA> ip link set <ifB> netns <netnsB>
```

Cada endereço IP foi configurado com:

```
ip netns exec <netns> ip addr add <ip>/<mask> \
        dev <interface>
```

O *NetEm* foi configurado no lado transmissor de cada interface com:

```
ip netns exec <netns> tc qdisc replace dev <interface> \
        root netem rate 45mbit limit 1000 \
        delay <latência> \
        loss gemodel <taxa de perda>
```

O *netperf* foi executado com:

```
ip netns exec <netns> netperf -v2 -l -8388608 \
        -t SCTP_STREAM -H 10.0.0.1 -- \
        -H 10.0.0.1 -L 10.0.0.2 \
        -H 10.0.1.2 -L 10.0.1.2
        -m 1452 -S 4194304 -s 4194304
```

O parâmetro `-m 1452` alinha o tamanho da mensagem com o MTU utilizado. Assim, cada pacote transporta 1 única mensagem e essa mensagem precisa de apenas 1 pacote para ser transportada. Nota que o *netperf* multiplica o tamanho dos buffers por 2 automaticamente.

Como não implementamos o *non-renegable selective acknowledgment* (NR-SACK) (NATARAJAN et al., 2008) (cujo uso é facultativo de acordo com (AMER et al., 2019; YILMAZ et al., 2010)), para não incorrer nos problemas de travamento de *buffer* descritos em

(IYENGAR et al., 2006, 2007), simplesmente utilizamos *buffers* de *socket* grandes, de 8MB. Nota que o Linux utiliza esse espaço tanto para dados quanto para metadados. Isso significa que poderia ocorrer alguma contenção na janela de recebimento, mas esse tamanho de *buffer* é consideravelmente maior que o *bandwidth-delay product* (BDP) e enquanto verificamos os resultados os bytes em voo nunca alcançaram a janela de recebimento anunciada. Da política de retransmissão, utilizamos a RTX-SAME (IYENGAR et al., 2006), que já estava disponível no Linux.

4.2 RESULTADOS

Tratar o sinalizador de recuperação rápida por transporte permite que a janela de congestionamento se adeque melhor às condições atuais do transporte em questão, tanto para reduzi-la quanto para aumentá-la. Isso trouxe ganhos tanto em termos de melhoria de desempenho quanto em redução de retransmissões, pois um transporte pôde reagir melhor a sua situação atual, independentemente da situação do outro.

Como esgotamentos de tempo ocorreram, a fase de partida lenta também foi utilizada ao longo dos testes. A recuperação mais ágil nessas situações, mesmo que pequena conforme visto no capítulo 3, também ajudou a reduzir o tempo de transferência.

Finalmente, nossos resultados são consistentes e corroboram com a proposta de que com as demais alterações, o algoritmo CUC não é mais necessário. No geral os resultados estão de acordo com o esperado, com exceção apenas do aumento de retransmissões nos testes com mesmas taxas de perdas nos dois caminhos.

Nos testes com a taxa de perda constante no caminho 1, com resultados em Fig. 8, Fig. 9, Fig. 10, Fig. 11, Fig. 12, Fig. 13 e em Fig. 14, podemos observar na Fig. 8 que o tempo necessário para transferir o arquivo foi reduzido em até 20,8% e até 26,4% menos retransmissões em Fig. 9 e em Fig. 10.

Já nos casos com mesmas taxas de perdas, o aumento da quantidade de retransmissões que pode ser observado em Fig. 16, Fig. 17, Fig. 18, Fig. 19, Fig. 20 e em Fig. 21 para taxas de perdas mais elevadas foi devido a uma maior troca de retransmissões por esgotamento de tempo por retransmissões rápidas. Essa troca pode ser observada em Fig. 22, Fig. 23, Fig. 24, Fig. 25, Fig. 26 e em Fig. 27. Isso ocasionou então um maior número total de retransmissões, mas elas sendo menos severas para a aplicação. Assim, foi possível melhorar o tempo de transmissão em até 23,7% quando com perdas de 10% nos dois caminhos, como visto na Fig. 15.

Os gráficos de retransmissões, exceto quando dito o contrário, incluem tanto as

retransmissões rápidas quanto as causadas por esgotamento de tempo e de ambos os transportes. Os marcadores de erro estão indicando um intervalo de confiança de 95% para as amostras, com 300 testes efetuados para cada ponto.

Com RTTs maiores é exatamente onde esperamos que a alteração faça maior diferença em termos de tempo de transferência, pois os esgotamentos de tempo fazem com que o transporte retorne à fase de partida lenta. Nossa alteração torna essa fase mais resiliente a perdas de pacote e, portanto, ela consegue recuperar mais rapidamente desses eventos. Em RTTs menores, a fase de partida lenta é mais curta e a diferença é menos perceptível.

A incidência de esgotamentos de tempo tem dois efeitos expressivos nos resultados. Primeiro, os marcadores de erro maiores nas maiores taxas de perdas de pacote são devido a maior incidência de esgotamentos de tempo para retransmissão. Como mais perdas acontecem, é provável que mais retransmissões também sejam perdidas e o SCTP depende apenas do esgotamento de tempo para recuperar de tais situações. Segundo, o tempo total de transferência é bastante aumentado, uma vez que tempo mínimo de esgotamento de tempo do SCTP é de 1 segundo.

O uso da política de retransmissão RTX-SAME, de acordo com (IYENGAR et al., 2006, 2007), tem resultados marginalmente piores mas consistentes com as nossas condições de teste, uma vez que ela não leva em consideração os efeitos das perdas de pacote. Mais importantemente, as políticas de retransmissão ajudam a evitar o problema de travamento do *buffer* de recepção. Como no teste os dados não podem ser entregues fora de ordem para a aplicação, pode ser que perdas de pacote em um caminho faça com que os dados transmitidos com sucesso através do outro caminho não possam ser entregues a aplicação até que tais perdas sejam recuperadas. Nos nossos testes tal travamento não ocorreu pois utilizamos de uma simplificação também utilizada nos trabalhos então mencionados, conforme descrito na seção 4.1.

De maneira similar, (NATARAJAN et al., 2008) descreve o travamento do *buffer* de transmissão. Como o SCTP só permite remover dados transmitidos da fila de transmissão quando eles forem confirmados em um ACK acumulativo, perdas de pacote em um caminho podem fazer com que o *buffer* de transmissão não possa avançar até que elas sejam recuperadas. Pois com a recuperação os dados até então confirmados em lacunas são confirmados com um ACK acumulativo e, então, não poderão mais ser renegados. Esse problema é mitigado com a utilização de um novo bloco de ACK, o NR-SACK. Como dito na seção 4.1, não o implementamos e por isso utilizamos da mesma simplificação para não incorrer nesse problema.

Para fins de nossos testes essas simplificações são válidas pois o teste consiste em uma

transferência de dados em massa (*bulk transfer*), na qual o receptor está sempre consumindo os dados e, neste trabalho, não estamos preocupados com a latência percebida pela aplicação. Ainda, a janela de congestionamento continua atrelada as condições do caminho e não pode crescer demasiadamente só por ter *buffers* disponíveis, o que, então, não prejudica a rede. Finalmente, nossas alterações não estão diretamente relacionadas a como os *buffers* são utilizados e ressaltamos que nossa proposta não tem por fim evitar os travamentos de *buffers*. Em (YILMAZ et al., 2010) o uso do NR-SACK o nunca causou um desempenho pior, e é o que esperamos em uma eventual integração das propostas. De qualquer maneira, são simplificações que não são realísticas e que precisam ser reavaliadas em um trabalho futuro.

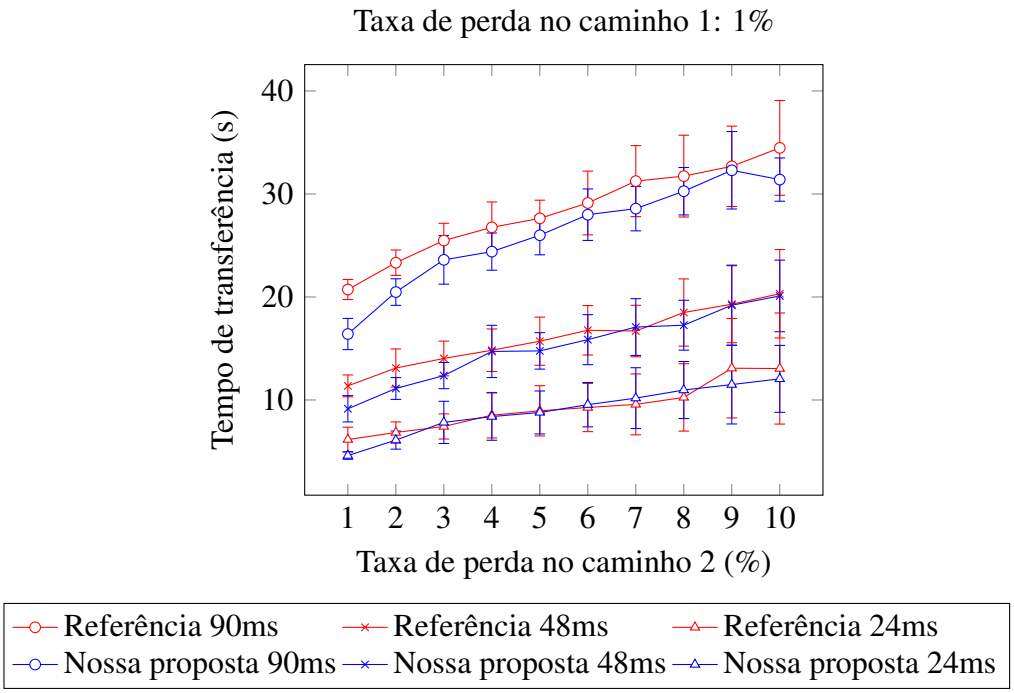

Figura 8: Tempo de transferência de um arquivo de 8MB com diferentes taxas de perda no caminho 2 e diferentes RTTs.

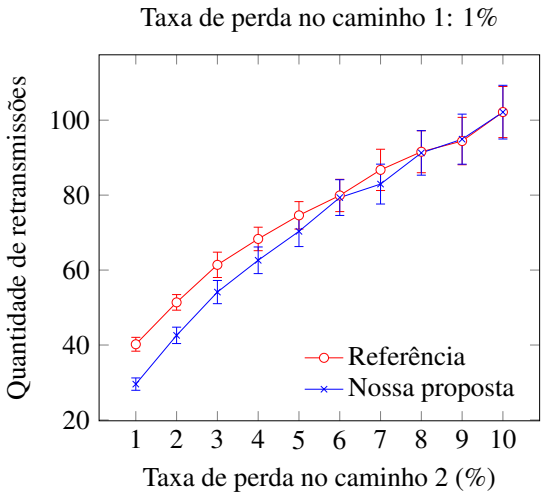

Figura 9: Total de retransmissões com várias taxa de perda no caminho 2 e RTTs de 24ms.

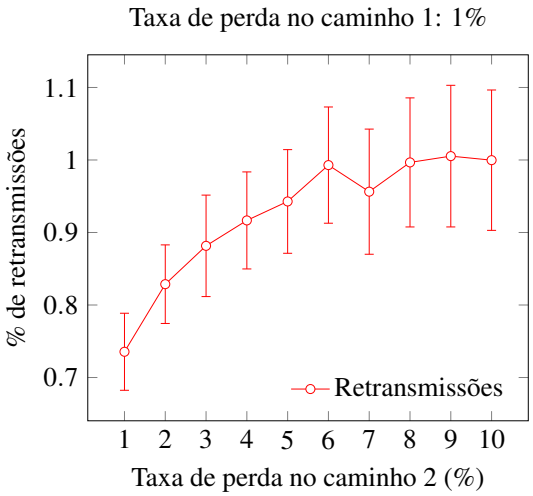

Figura 10: Total de retransmissões de nossa proposta proporcional a referência com várias taxas de perdas no caminho 2 e RTTs de 24ms.

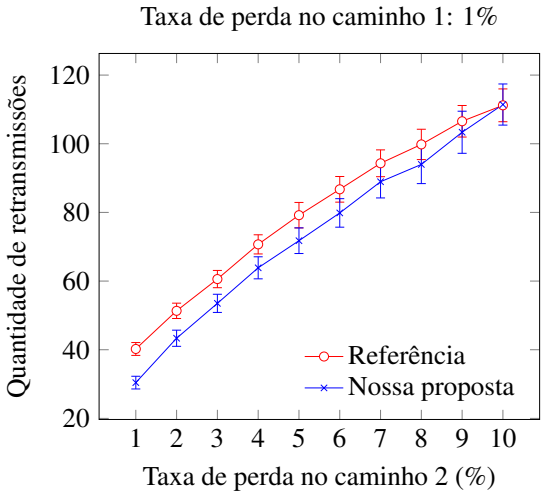

Figura 11: Total de retransmissões com várias taxa de perda no caminho 2 e RTTs de 48ms.

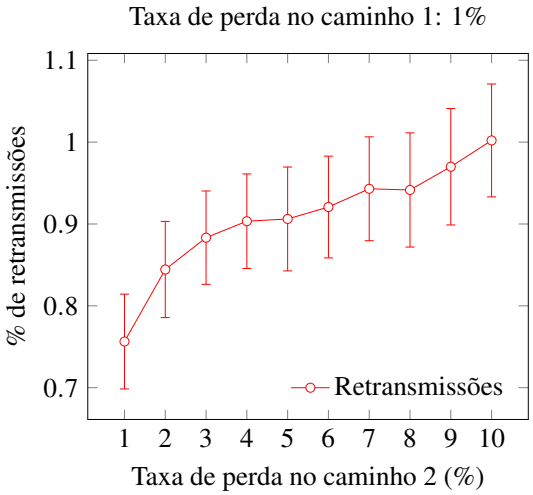

Figura 12: Total de retransmissões de nossa proposta proporcional a referência com várias taxas de perdas no caminho 2 e RTTs de 48ms.

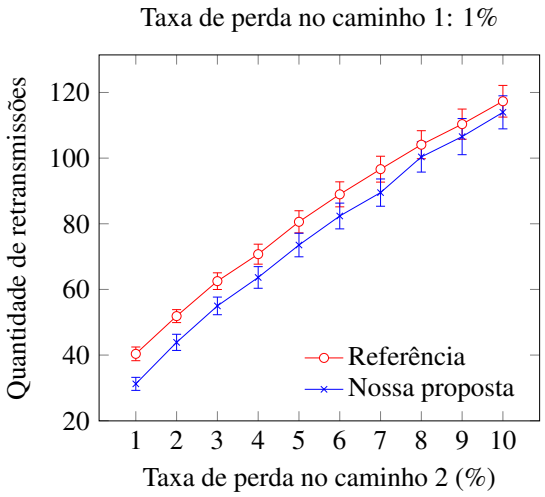

Figura 13: Total de retransmissões com várias taxa de perda no caminho 2 e RTTs de 90ms.

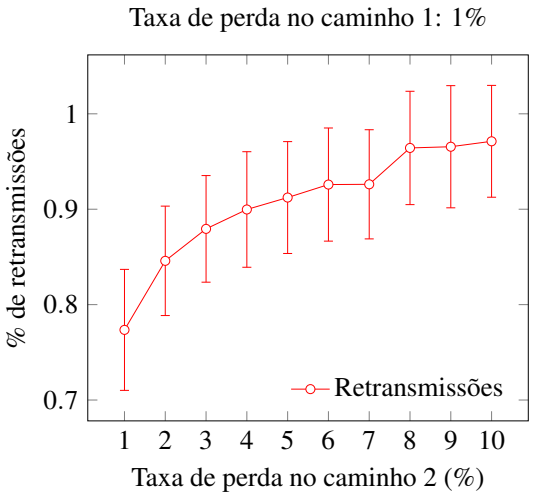

Figura 14: Total de retransmissões de nossa proposta proporcional a referência com várias taxas de perdas no caminho 2 e RTTs de 90ms.

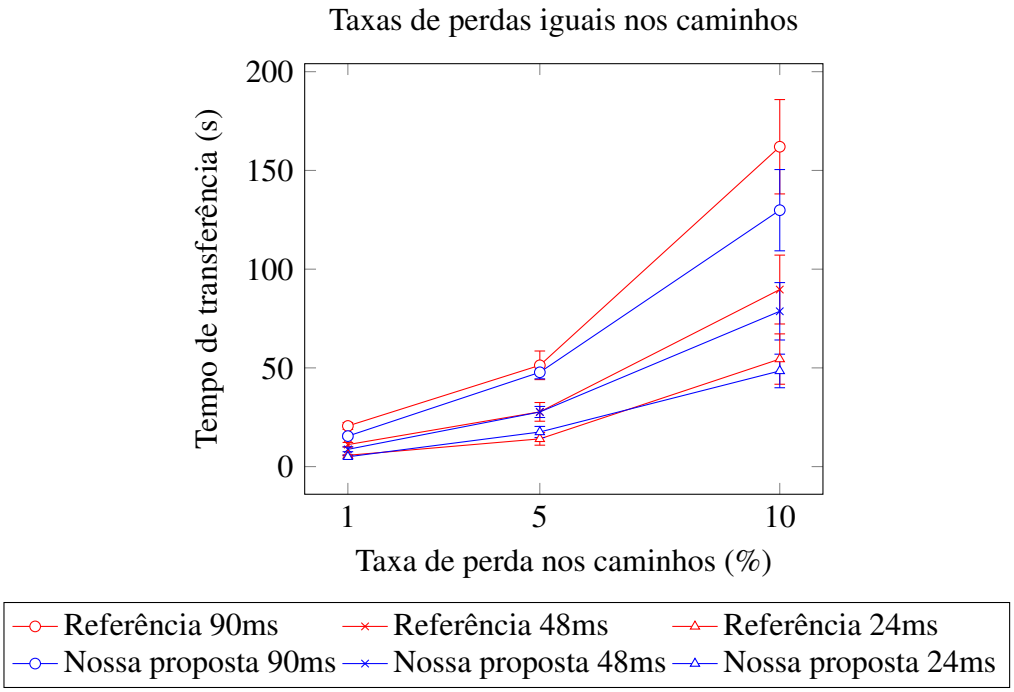

Figura 15: Tempo de transferência de um arquivo de 8MB com taxas de perdas iguais nos caminhos e diferentes RTTs.

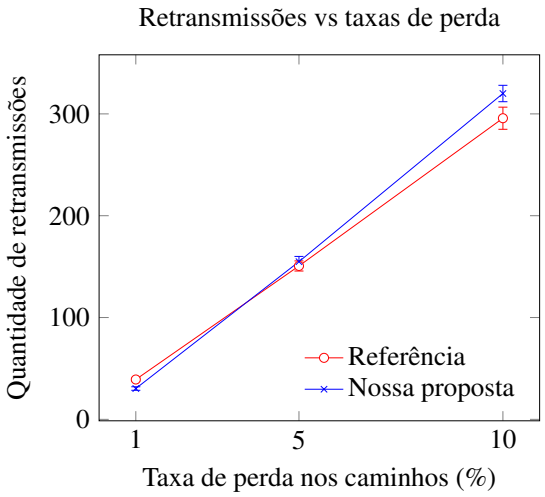

Figura 16: Total de retransmissões com várias taxas de perdas iguais nos caminhos e RTTs de 24ms.

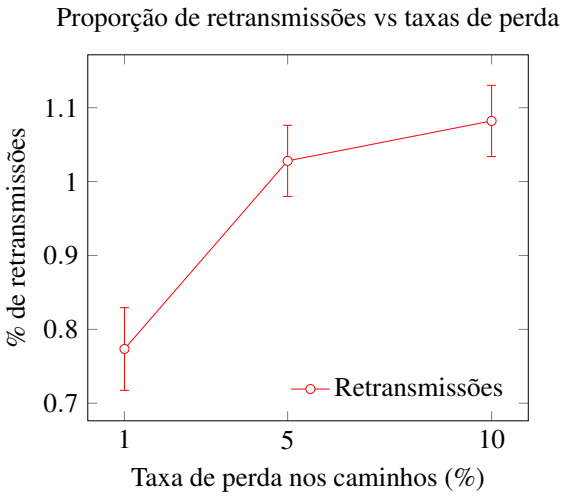

Figura 17: Total de retransmissões de nossa proposta proporcional a referência com várias taxas de perdas iguais nos caminhos e RTTs de 24ms.

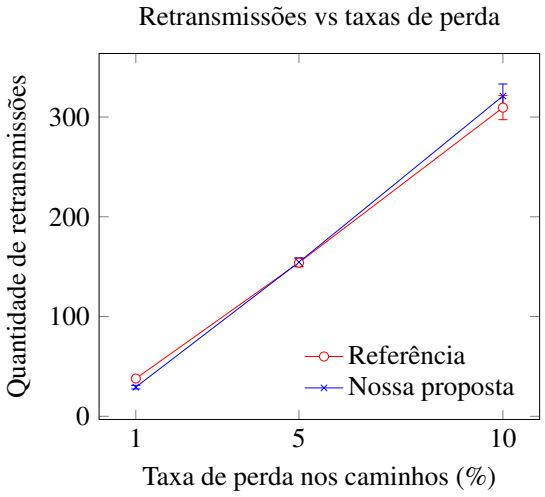

Figura 18: Total de retransmissões com várias taxas de perdas iguais nos caminhos e RTTs de 48ms.

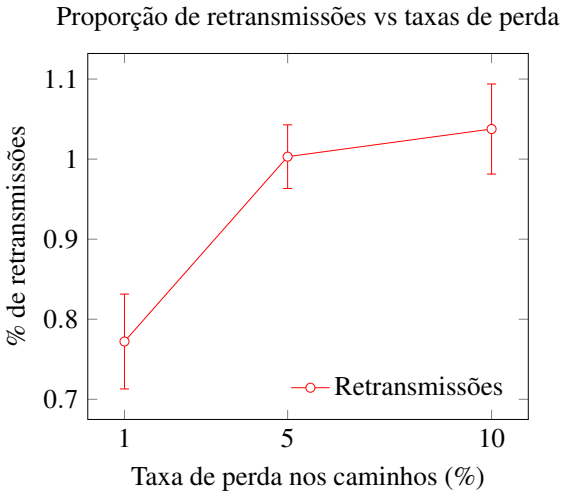

Figura 19: Total de retransmissões de nossa proposta proporcional a referência com várias taxas de perdas iguais nos caminhos e RTTs de 48ms.

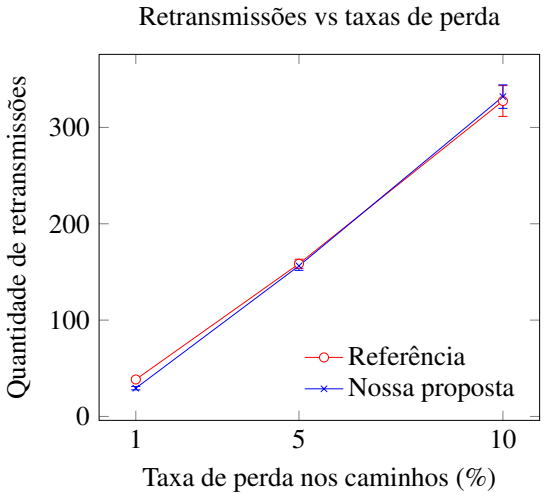

Figura 20: Total de retransmissões com várias taxas de perdas iguais nos caminhos e RTTs de 90ms.

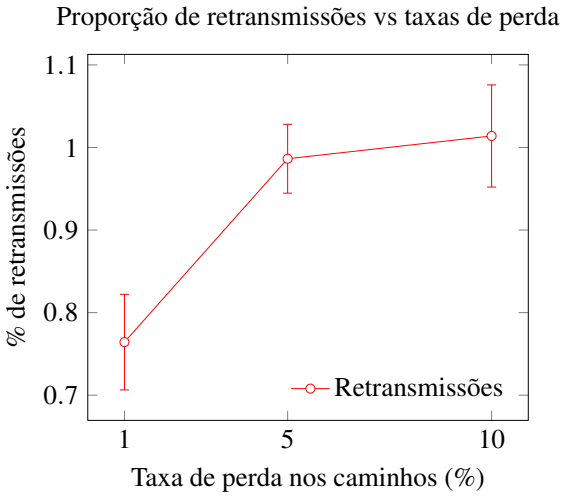

Figura 21: Total de retransmissões de nossa proposta proporcional a referência com várias taxas de perdas iguais nos caminhos e RTTs de 90ms.

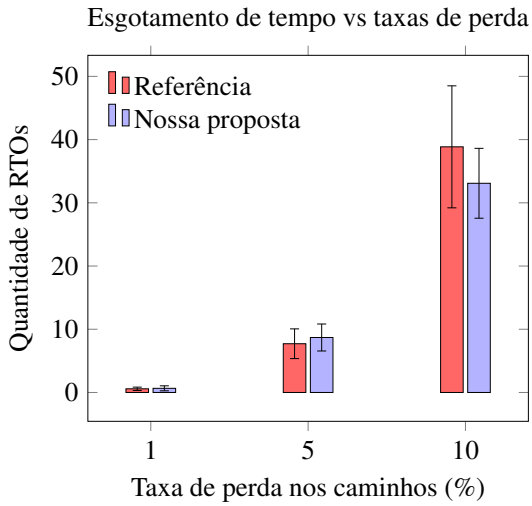

Figura 22: Total de retransmissões por esgotamento de tempo com várias taxas de perdas iguais nos caminhos e RTTs de 24ms.

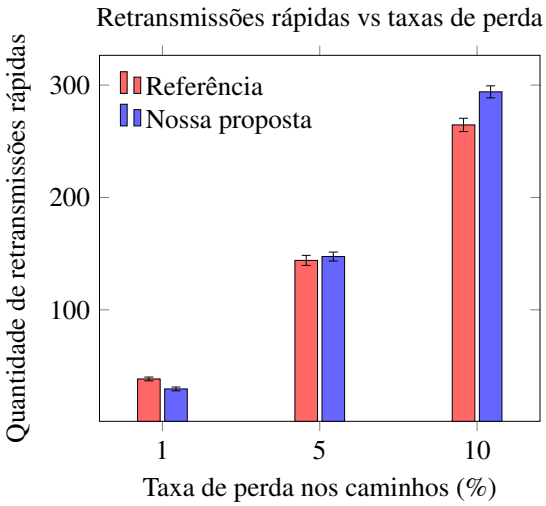

Figura 23: Total de retransmissões de nossa proposta proporcional a referência com várias taxas de perdas iguais nos caminhos e RTTs de 24ms.

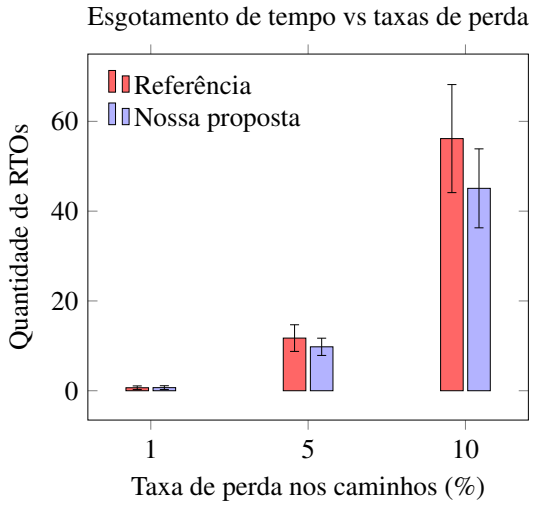

Figura 24: Total de retransmissões por esgotamento de tempo com várias taxas de perdas iguais nos caminhos e RTTs de 48ms.

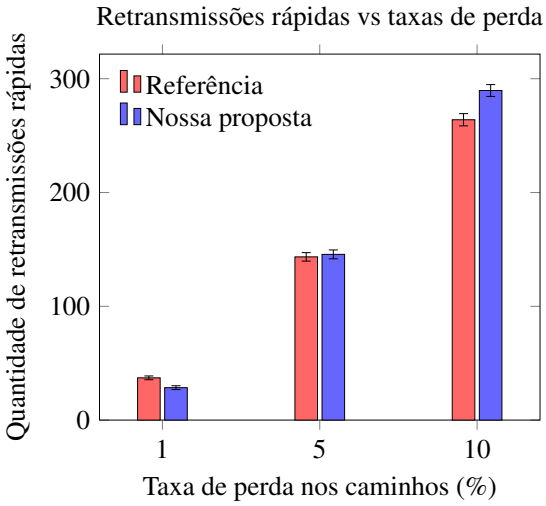

Figura 25: Total de retransmissões de nossa proposta proporcional a referência com várias taxas de perdas iguais nos caminhos e RTTs de 48ms.

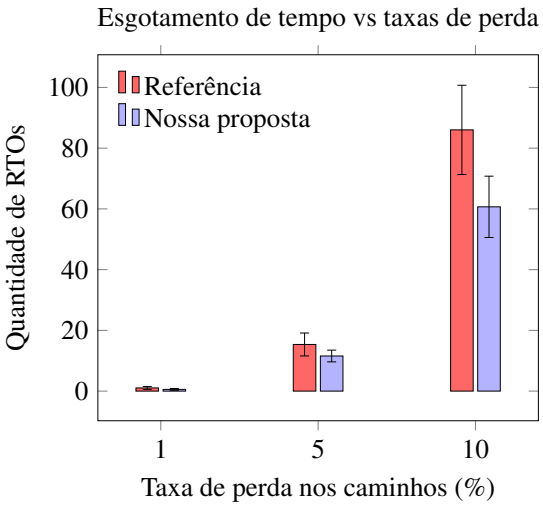

Figura 26: Total de retransmissões por esgotamento de tempo com várias taxas de perdas iguais nos caminhos e RTTs de 90ms.

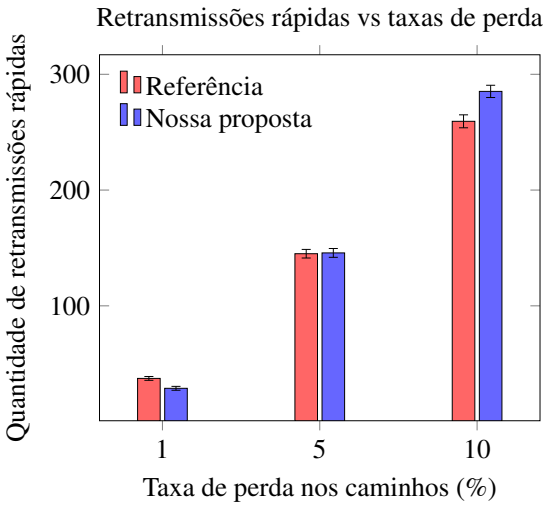

Figura 27: Total de retransmissões de nossa proposta proporcional a referência com várias taxas de perdas iguais nos caminhos e RTTs de 90ms.

5 CONCLUSÕES

Nesse trabalho revisamos o tratamento atual da fase de partida lenta no SCTP tradicional e no CMT-SCTP. Notamos o novo entendimento sobre as lacunas durante a fase de prevenção de congestionamento pela errata do SCTP e, com isso, estendemos a ideia para a fase de partida lenta. Os resultados indicam que com a nossa proposta o desempenho é similar ao que é alcançado pelo CMT-SCTP em baixos RTTs e é consistentemente melhor em RTTs maiores. As retransmissões foram reduzidas em quase todos os testes, um resultado direto de considerar a recuperação rápida por transporte. Quando não foram reduzidas de maneira absoluta, menos retransmissões por esgotamento de tempo foram necessárias, também indicando que o transporte foi capaz de se adequar melhor as condições atuais.

Mesmo considerando que testamos apenas com o RTX-SAME, que é a implementação mais simples, ela é também a que fornece os piores resultados no CMT-SCTP de acordo com (IYENGAR et al., 2006). Assim acreditamos que outras políticas de retransmissão só forneceriam melhores resultados.

Situações de janela fechada também podem ser afetadas por essa alteração pois o transmissor pode precisar voltar à fase de partida lenta se ficar ocioso. Lembramos que um transporte é tido como ocioso quando ele não consegue mensurar o RTT em um intervalo de *heartbeat* (30 s por padrão) e, nesse caso, as alterações propostas devem resultar apenas em uma recuperação mais rápida. Caso contrário, como a janela de congestionamento não estará sendo completamente utilizada, as alterações não devem ter efeitos.

Considerando que o CMT-SCTP é um protocolo complexo que requer implementações não triviais, esses resultados são importantes pois mesmo que os números possam indicar ganhos marginais em alguns casos, eles sustentam a simplificação no protocolo. Essa simplificação é uma grande contribuição da nossa proposta.

Portanto propomos três alterações. Primeiro, a indicação de estado de recuperação rápida deve ser por transporte, assim como as demais variáveis relacionadas a janela de congestionamento. Segundo, que blocos de dados confirmados em lacunas também sejam

considerados para o aumento da janela de congestionamento durante a fase de partida lenta, tanto na versão tradicional do SCTP quanto no CMT-SCTP. Terceiro, utilizando as anteriores, que o algoritmo CUC não precisa mais ser incluído na RFC do CMT-SCTP.

5.1 TRABALHOS FUTUROS

Os efeitos de tráfego de fundo e a interação entre os dois modelos ainda não foram testados. Como o aumento da janela de congestionamento ainda é atrelada ao RTT, não esperamos resultados negativos nesses testes.

Como testamos apenas a política de transmissão RTX-SAME, as outras políticas ainda precisam ser estudadas.

Apesar de nossa proposta não ter relação direta com os travamentos de *buffers* previstos por (IYENGAR et al., 2006, 2007; NATARAJAN et al., 2008), é interessante validá-la sem a simplificação (utilização de *buffers* grandes) e com a implementação do NR-SACK. Portanto, da mesma maneira, também é interessante testes com cenários heterogêneos (com RTT diferentes).

REFERÊNCIAS

AL, A. A. E.; SAADAWI, T.; LEE, M. LS-SCTP: a bandwidth aggregation technique for stream control transmission protocol. **Computer Communications**, v. 27, n. 10, p. 1012 – 1024, 2004. ISSN 0140-3664. Protocol Engineering for Wired and Wireless Networks.

AMER, P. P. D. et al. Internet-Draft, **Load Sharing for the Stream Control Transmission Protocol (SCTP)**. Internet Engineering Task Force, jan. 2019. Work in Progress. Disponível em: <https://datatracker.ietf.org/doc/html/draft-tuexen-tsvwg-sctp-multipath-17>.

BRADEN, R. T. **Requirements for Internet Hosts - Communication Layers**. RFC Editor, out. 1989. RFC 1122. (Request for Comments, 1122). Disponível em: <https://rfc-editor.org/rfc/rfc1122.txt>.

BRAKMO, L.; PETERSON, L. TCP Vegas: end to end congestion avoidance on a global internet. **IEEE Journal on Selected Areas in Communications**, Institute of Electrical and Electronics Engineers Inc., v. 13, n. 8, p. 1465–1480, 10 1995. ISSN 0733-8716.

DANTU, R. et al. **Signaling System 7 (SS7) Message Transfer Part 2 (MTP2) - User Peer-to-Peer Adaptation Layer (M2PA)**. RFC Editor, set. 2005. RFC 4165. (Request for Comments, 4165). Disponível em: <https://rfc-editor.org/rfc/rfc4165.txt>.

DREIBHOLZ, T. et al. **Implementation and Evaluation of Concurrent Multipath Transfer for SCTP in the INET Framework**. 03 2010.

DREIBHOLZ, T. et al. Transmission Scheduling Optimizations for Concurrent Multipath Transfer. In: **Proceedings of the 8th International Workshop on Protocols for Future, Large-Scale and Diverse Network Transports (PFLDNeT)**. Lancaster, Pennsylvania/U.S.A.: [s.n.], 2010. v. 8. ISSN 2074-5168. Disponível em: <https://www.wiwi.uni-due.de/fileadmin/fileupload/I-TDR/SCTP/Paper/PFLDNeT2010.pdf>.

IYENGAR, J. Internet-Draft, **Preventing SCTP Congestion Window Overgrowth During Changeover**. Internet Engineering Task Force, dez. 2005. Work in Progress. Disponível em: <https://datatracker.ietf.org/doc/html/draft-iyengar-sctp-cacc-03>.

IYENGAR, J. R.; AMER, P. D.; STEWART, R. Concurrent Multipath Transfer Using SCTP Multihoming over Independent End-to-end Paths. **IEEE/ACM Trans. Netw.**, IEEE Press, Piscataway, NJ, USA, v. 14, n. 5, p. 951–964, out. 2006. ISSN 1063-6692. Disponível em: <http://dx.doi.org/10.1109/TNET.2006.882843>.

IYENGAR, J. R.; AMER, P. D.; STEWART, R. Performance implications of a bounded receive buffer in concurrent multipath transfer. **Computer Communications**, v. 30, n. 4, p. 818 – 829, 2007. ISSN 0140-3664. Nature-Ispired Distributed Computing.

JACOBSON, V. Congestion Avoidance and Control. **SIGCOMM Comput. Commun. Rev.**, ACM, New York, NY, USA, v. 18, n. 4, p. 314–329, ago. 1988. ISSN 0146-4833. Disponível em: <http://doi.acm.org/10.1145/52325.52356>.

LAN, K.; SHA, N. A CMT congestion window updates mechanism based on TCP Westwood. **Proceedings 2011 International Conference on Mechatronic Science, Electric Engineering and Computer, MEC 2011**, 08 2011.

LEITNER, M. R.; FONSECA, M. S. P.; MUNARETTO, A. On Simplifying Congestion Window Handling for CMT-SCTP. In: **Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing**. New York, NY, USA: ACM, 2019. (SAC '19), p. 1373–1377. ISBN 978-1-4503-5933-7. Disponível em: <http://doi.acm.org/10.1145/3297280.3300180>.

LIAO, J.; WANG, J.; ZHU, X. cmpSCTP: An Extension of SCTP to Support Concurrent Multi-Path Transfer. In: **2008 IEEE International Conference on Communications**. Piscataway, NJ, USA: IEEE Press, 2008. p. 5762–5766. ISSN 1550-3607.

LUDOVICI, F.; PFEIFER, H. P. **NetEm manual**. 2011. `http://man7.org/linux/man-pages/man8/tc-netem.8.html`. [Online; acessado 31-Maio-2019].

NATARAJAN, P. et al. Non-Renegable Selective Acknowledgments (NR-SACKs) for SCTP. In: **2008 IEEE International Conference on Network Protocols**. Piscataway, NJ, USA: IEEE Press, 2008. p. 187–196. ISSN 1092-1648.

PASTOR, J.; MORNEAULT, K. **Signaling System 7 (SS7) Message Transfer Part 3 (MTP3) - User Adaptation Layer (M3UA)**. RFC Editor, set. 2006. RFC 4666. (Request for Comments, 4666). Disponível em: <https://rfc-editor.org/rfc/rfc4666.txt>.

PEROTTO, F.; CASETTI, C.; GALANTE, G. SCTP-based Transport Protocols for Concurrent Multipath Transfer. In: **2007 IEEE Wireless Communications and Networking Conference**. Piscataway, NJ, USA: IEEE Press, 2007. p. 2969–2974. ISSN 1525-3511.

RAMALHO, D. M. A. et al. **Stream Control Transmission Protocol (SCTP) Partial Reliability Extension**. RFC Editor, maio 2004. RFC 3758. (Request for Comments, 3758). Disponível em: <https://rfc-editor.org/rfc/rfc3758.txt>.

RHEE, I. et al. **CUBIC for Fast Long-Distance Networks**. RFC Editor, fev. 2018. RFC 8312. (Request for Comments, 8312). Disponível em: <https://rfc-editor.org/rfc/rfc8312.txt>.

SHAILENDRA, S.; BHATTACHARJEE, R.; BOSE, S. K. MPSCTP: A Simple and Efficient Multipath Algorithm for SCTP. **IEEE Communications Letters**, IEEE Press, Piscataway, NJ, USA, v. 15, n. 10, p. 1139–1141, October 2011. ISSN 1089-7798.

STEWART, R. R. **Stream Control Transmission Protocol**. RFC Editor, set. 2007. RFC 4960. (Request for Comments, 4960). Disponível em: <https://rfc-editor.org/rfc/rfc4960.txt>.

STEWART, R. R.; TÜXEN, M.; NIELSEN, K. Internet-Draft, **Stream Control Transmission Protocol**. Internet Engineering Task Force, mar. 2019. Work in Progress. Disponível em: <https://datatracker.ietf.org/doc/html/draft-ietf-tsvwg-rfc4960-bis-01>.

STEWART, R. R.; TÜXEN, M.; PROSHIN, M. **Stream Control Transmission Protocol: Errata and Issues in RFC 4960**. RFC Editor, fev. 2019. RFC 8540. (Request for Comments, 8540). Disponível em: <https://rfc-editor.org/rfc/rfc8540.txt>.

TÜXEN, M. et al. **An Overview of Reliable Server Pooling Protocols**. RFC Editor, set. 2008. RFC 5351. (Request for Comments, 5351). Disponível em: <https://rfc-editor.org/rfc/rfc5351.txt>.

TÜXEN, M. et al. **Additional Policies for the Partially Reliable Stream Control Transmission Protocol Extension**. RFC Editor, abr. 2015. RFC 7496. (Request for Comments, 7496). Disponível em: <https://rfc-editor.org/rfc/rfc7496.txt>.

VARMA, S. **Internet Congestion Control**. 1. ed. Waltham, Massachusetts, USA: Morgan Kaufmann, 2015. ISBN 978-0-12-803583-2.

WALLACE, T. D.; SHAMI, A. Concurrent Multipath Transfer Using SCTP: Modelling and Congestion Window Management. **Mobile Computing, IEEE Transactions on**, IEEE Press, Piscataway, NJ, USA, v. 13, p. 2510–2523, 11 2014.

YILMAZ, E. et al. Throughput analysis of non-renegable selective acknowledgments (NR-SACKs) for SCTP. **Computer Communications**, v. 33, p. 1982–1991, 10 2010.