UNIVERSIDADE FEDERAL FLUMINENSE
MESTRADO EM ENGENHARIA DE TELECOMUNICACOES

NÉSTOR FELIPE MAYA QUINTERO

MECANISMO END TO END TRAFFIC SHAPING E2E

NITERÓI
2006

NÉSTOR FELIPE MAYA QUINTERO

MECANISMO FIM A FIM DE CONTROLE E MODULAÇÃO DE TRÁFEGO E2E

Dissertação apresentada ao Curso de Pós-Graduação em Engenharia de Telecomunicações da Universidade Federal Fluminense, como requisito parcial para obtenção do Grau de Mestre.

Orientador: LUIZ CLAUDIO SCHARA MAGALHÃES, PHD

Niterói
2006

NÉSTOR FELIPE MAYA QUINTERO

MECANISMO FIM A FIM DE CONTROLE E MODULAÇÃO DE TRÁFEGO E2E

Dissertação apresentada ao Curso de Pós-Graduação em Engenharia de Telecomunicações da Universidade Federal Fluminense, como requisito parcial para obtenção do Grau de Mestre.

Aprovada em 18 de janeiro de 2007

BANCA EXAMINADORA

Prof Luiz Cláudio Schara Magalhaes – Orientador
Universidade Federal Fluminense

Prof[a] Débora Christina Muchaluat Saade
PHD - UFF

Prof[a]. Noemi de la Roque Rodriguez
PHD - PUC

Niterói
2007

RESUMO

A maior parte dos mecanismos fim a fim desenvolvidos para dar forma ao tráfego nas comunicações de dados permite configurações estáticas, que não podem ser variadas dinamicamente de acordo com as características mutáveis da rede. Dar forma ao tráfego fim a fim de maneira dinâmica é um dos focos de estudo deste trabalho, que busca criar um controle de fluxo baseado na largura de banda disponível em um caminho, espaçando os pacotes regularmente, de acordo com a largura de banda medida. Como resultado deste trabalho, foi desenvolvido o mecanismo denominado e2e. Este permite a qualquer protocolo de transporte, como o UDP ou o TCP, fazer uso da ferramenta de controle de fluxo, com a finalidade de apoiar o controle de congestionamento. O uso desta ferramenta permite avaliar qual seria o comportamento dos protocolos de transporte se estes fossem baseados em taxa.

ABSTRACT

The majority of end-to-end traffic shaping mechanisms developed for data communication allows only static configurations, and cannot be used to dynamically follow network conditions. The focus of this work is to dynamically shape end-to-end traffic, to create flows that stay within the available path bandwidth, and have regular spacing between packets, through end-to-end bandwidth measurements. The result of this work is the e2e mechanism. It allows any flow, be it UDP or TCP, to be controlled, helping network congestion control. It also helps investigating what would be the transport protocol behavior if it were rate-based.

INDICE

LISTA DE FIGURAS

LISTA DE TABELAS

1. INTRODUÇÃO

Este trabalho descreve um mecanismo fim a fim de controle e modulação de tráfego, chamado e2e. Ele foi desenvolvido a partir das técnicas desenvolvidas para o protocolo MMTP [46] (Multimedia Multiplexing Transport Protocol), que busca diminuir as perdas de pacotes por mecanismos diferentes dos estabelecidos no controle de congestionamento do TCP, através de estimativas de banda disponível em um caminho por meio de técnicas de provas ativas, para calcular a taxa de transmissão usada num caminho, e por ser *multihomed*[1], usar esta taxa para o balanceamento de carga, distribuindo o tráfego enviando pacotes por todas as interfaces de rede simultaneamente [21].

O MMTP foi desenvolvido inicialmente entre a camada de aplicação e a camada de transporte, usando o UDP como protocolo base. O projeto e2e resultou da necessidade de adquirir resultados destas técnicas ao nível de kernel, utilizando uma melhor resolução de timers disponíveis no kernel em contraste com aqueles disponíveis à nível de usuário, usando as camadas mais baixas do modelo de referência OSI e a estrutura algorítmica do protocolo MMTP.

O projeto e2e funciona como controle de tráfego para aplicações UDP e TCP. Devido as características intrínsecas do UDP e do TCP, com UDP o e2e permite um controle de congestionamento dependendo do tráfego na rota, enquanto com o TCP, o controle de congestionamento é acompanhado por um controle de fluxo baseado em taxa, para diminuir o número de perdas causadas nos roteadores por saturação das filas de pacotes. Estas funções provem uma alternativa para regular a taxa de transmissão de forma dinâmica, permitindo o controle do envio de pacotes.

Outros resultados obtidos são as transmissões de pacotes através de múltiplas interfaces de rede, ainda que este não seja o tópico central desta pesquisa. Foi estudado um método para determinar o dispositivo de rede cujo tráfego do enlace seja menor, de forma que este pode ser escolhido para realizar a transmissão.

---

[1] *Multihomed* significa ter múltiplas interfaces ativas simultaneamente, cada uma com seu próprio endereço IP por estar potencialmente conectada à redes diferentes

O mecanismo e2e permite a qualquer aplicação ativar o controle de fluxo utilizando um soquete IP, um protocolo de transporte e porta de enlace como configuração base para realizar medidas e analisar de forma experimental o *throughput* de um protocolo de transporte baseado em taxa.

2. OBJETIVOS

2.1. Objetivos gerais

Desenvolver um mecanismo de controle de fluxo que permita variar a taxa de transmissão de um protocolo de transporte, dependendo do tráfego do enlace, com a finalidade minimizar o número de pacotes perdidos e aumentar o desempenho nas comunicações de dados.

2.2. Objetivos específicos

- Pesquisar quais são as ferramentas desenvolvidas atualmente para realizar o controle de fluxo baseado em taxa.
- Identificar problemas nos protocolos de transporte UDP e TCP nas comunicações de dados.
- Detalhar soluções para possíveis problemas, utilizando um mecanismo de controle de fluxo baseado em taxa.
- Analisar, modelar e desenvolver um mecanismo, especificando as características mais importantes para a realização do controle de fluxo baseado em taxa.
- Realizar medidas comparativas entre os protocolos de transporte UDP e TCP em diferentes cenários utilizando o mecanismo de controle de fluxo baseado em taxa.

3. HIPÓTESES

3.1. Protocolos de transporte

O problema inicial da Internet, na infância dos protocolos de transporte, foi o colapso ocasionado pelas transmissões de pacotes que eram enviados sem nenhum tipo de controle de congestionamento [11]. O congestionamento ocorria nos roteadores, fazendo com que pacotes fossem descartados. No caso do TCP quando um pacote era descartado um *timeout* seguido de uma retransmissão acontecia, causando um congestionamento ainda maior. Ao longo do tempo, houve incidentes onde problemas sérios incapacitavam grandes segmentos da rede. Alguns destes incidentes foram resultados dos algoritmos utilizados, ou da falta de algoritmos de controle de congestionamento, no TCP [12]. No caso do UDP, não precisando ter confiabilidade de transporte, os pacotes sempre eram descartados, mas a falta de controle do congestionamento deste protocolo também gerava problemas para a rede.

É importante compreender que a estratégia do TCP é controlar o congestionamento quando este ocorre, em vez de evitar o a ocorrência do mesmo [12]. Em outras palavras, o TCP precisa criar perdas de pacotes para encontrar a largura de banda disponível na conexão.

3.2. Mecanismo de controle de congestionamento do TCP

O controle de congestionamento do TCP padronizado no RFC 2582 [14] define os mecanismos usados no algoritmo tal como: slow start, congestion avoidance, fast retransmit e fast recovery.

Os termos comuns do controle de congestionamento são:

- ***Segment***: qualquer pacote de dados TCP/IP ou acknowledgement (ack).

- ***Sender Maximum Segment Size*** (SMSS): é o maior tamanho do segmento que pode ser transmitido. Este valor pode se basear no Maximum Transmission Unit (MTU) da rede. O tamanho do segmento não inclui os cabeçalhos.
- ***Receiver Maximum Segment Size*** (RMSS): é o tamanho do maior segmento que o receptor pode aceitar. Este valor é especificado na opção MSS enviada pelo receptor no inicio da conexão. Se o MSS não é usado, o valor por default será de 536 bytes. O tamanho do segmento não inclui os cabeçalhos.
- ***Full-Sized Segment*** (FSS): máximo número de bytes permitidos no segmento.
- ***Receiver Window*** (rwnd): é o mais recente *advertised received window [34]*.
- ***Congestion Window*** (cwnd): variável de estado que limita a quantidade de dados que o TCP pode enviar.
- ***Initial Window*** (IW): tamanho inicial do cwnd depois que *three-way handshake* é completado.
- ***Loss Window*** (LW): tamanho do cwnd depois que o TCP detecta uma perda, causado pelo mecanismo de timeout.
- ***Restart Window*** (RW): tamanho do cwnd depois que o TCP reinicia a retransmissão causada por um potencial burst inapropriado [26].
- ***Flight Size***, quantidade de dados que foram enviados, mas que não receberam o ack.

3.2.1. Slow start

Os algoritmos de *slow start* e *congestion avoidance* devem ser utilizados para controlar a quantidade de dados transmitidos na rede, aumentando o fluxo de pacotes de uma conexão até encontrar a capacidade disponível do canal. Na implementação do algoritmo duas variáveis de estado são amplamente usadas: *cwnd,* que limita a quantidade de bytes que podem ser transmitidos e o *rwnd* no lado receptor, que limita a quantidade de dados que podem ser recebidos. O menor valor entre cwnd e rwnd é usado na transmissão de dados [14].

Começar a transmissão na rede com condições desconhecidas, requer que o TCP prove lentamente a rota para determinar a capacidade disponível e prevenir congestionamento.

O algoritmo de *slow start* cumpre com este propósito e é usado no inicio da transferência e depois de reparar uma perda detectada pelo *timeout*.

Outra variável de estado é *slow start threshold* (ssthresh) o qual determina se o algoritmo de *slow start* ou de *congestion avoidance* vai ser usado para controlar a transmissão de dados. O valor inicial de *ssthresh* pode ser arbitrariamente alto como, por exemplo, o valor de *advertised window,* mas este valor deve ser reduzido em resposta à congestão. O *slow start* é usado quando a variável cwnd é menor que *ssthresh*, entretanto o *congestion avoidance* é usado quando a variável *cwnd* é maior que *ssthresh* [14].

Durante o *slow start*, TCP incrementa cwnd pelo menos em um SMSS para cada ack recebido. O *slow start* termina quando cwnd excede o máximo valor permitido para cwnd ou quando o congestionamento é observado.

Durante o *congestion avoidance,* o cwnd é incrementado em um FSS por cada round trip time (RTT). O *congestion avoidance* continua até que o congestionamento seja detectado.

Quando o emissor detecta que um segmento foi perdido devido a um *timeout*, o *ssthresh* deve adquirir um valor não maior ao dado na seguinte equação:

$$ssthresh = \max(FlightSize / 2, 2*SMSS) \qquad (1)$$

3.2.2. Fast Retransmit / Fast Recovery

Um receptor TCP deve enviar um duplicate ack quando um segmento fora de ordem chegar. O propósito deste ack é informar ao emissor o número da seqüência esperada quando o segmento foi perdido ou foi recebido fora de ordem.

Na perspectiva do emissor, o duplicate ack pode ser causado por um grande número de problemas na rede alguns dos quais são: segmentos perdidos, reordenação dos segmentos e replicação do ack ou do segmento [14].

O emissor TCP deve usar o algoritmo fast retransmit para detectar e reparar perdas baseado no duplicate ack. O fast retransmit usa a chegada de três duplicate ack como condição que o segmento foi perdido, retransmitindo o segmento sem esperar que um timeout aconteça.

Depois que o algoritmo fast retransmit informa a falta de um segmento, o algoritmo fast recovery é executado até um non-duplicate ack chegar. A razão do TCP não entrar no estado de slow start é devido ao duplicate ack indicar que o segmento foi perdido, mas que foi retransmitido.

O algoritmo *fast retransmit* e *fast recovery* está implementado da seguinte forma:

1. Quando o terceiro *duplicate ack* é recebido, *ssthresh* adquire um valor não maior que o resultado na equação (1).
2. Quando retransmite o segmento perdido e *cwnd* adquire o valor de ***ssthresh + 3*SMSS***
3. Cada *duplicate ack* recebido, incrementa o *cwnd* em um SMSS.
4. Transmitir um segmento se estiver permitido pelo novo valor do *cwnd* e pelo *advertised window* do receptor.
5. Quando o próximo *ack* de um novo segmento chegar, o cwnd adquire o valor igual ao do *ssthresh*.

3.3. Análise de protocolos de transporte

Os algoritmos do TCP que o tornam um protocolo de transporte confiável são complexos, e precisam gerar congestionamento e criar perdas para conhecer a largura de banda alcançável. Ainda que o TCP seja bastante utilizado como um protocolo da camada de transporte em Internet para serviços de rede que suportam tanto melhor esforço como multimídia, este protocolo não é eficiente para prover serviços para este último caso [20].

Outros serviços podem usar o UDP como protocolo da camada de transporte, mas como o UDP não tem um mecanismo de controle de congestionamento, o aumento do tráfego do UDP para serviços novos pode causar a instabilidade da rede [20].

A transmissão de dados em redes onde a largura de banda é alocada dinamicamente tal como Internet, pode dar um comportamento agressivo quando existe um tráfego significativo levando as filas dos roteadores a transbordar e subseqüentemente perda dos pacotes. No entanto, se for possível obter informações como largura de banda, taxa do enlace, carga de rede, etc., o conhecimento destas medidas permitiria uma maior eficiência e confiabilidade do uso de Internet para:

1. Usuários:
    - Aplicações flexíveis e protocolos que poderiam se adaptar às variações da largura de banda. Isto pode ser decisivo para a transferência de dados em tempo real.
    - O custo de acesso à Internet geralmente depende da largura de banda da conexão.

2. Provedores:
    - Uma largura de banda dinâmica e uma estimativa da carga do enlace poderiam ser usadas para desenhar protocolos de roteamento mais eficientes e sólidos.
    - Utilizando estatísticas de tráfego, os provedores poderiam localizar fontes de ineficiência e planejar melhores capacidades.

4. TRABALHOS RELACIONADOS

4.1. Trajeto dos pacotes no Kernel de Linux

O trajeto de um pacote entre as primeiras camadas do modelo de referência OSI, pode ser explicado com alguns conceitos da API (Application Program Interface) do Kernel de Linux [29], usada no desenvolvimento de módulos que permitem o envio e recepção de dados na rede.

4.1.1. Da camada um a camada dois

O Linux suporta um vasto número de tipos de interfaces, tal como Ethernet (10/100baseT ou Gigabit Ethernet), atm, isdn, fddi, wireless lan e outros. Cada interface tem sua maneira de receber os pacotes, como por exemplo, o adaptador Ethernet normalmente está dividido em duas regiões usadas para receber e enviar pacotes, tal como, o modelo 3c509 com 4KB que usa 2KB para recepção (Rx), 2KB para transmissão (Tx), ou o modelo 3c509B com 8KB pode usar 4/4, 5/3 ou 6/2 para Rx/Tx [18].

A interface no Linux é representada pela estrutura de dados chamada net_device [31]. Esta estrutura captura as informações vindas do meio físico ou da camada três, relativa ao nome da interface, memória I/O, interrupção, informação de queue de Tx/Rx, alguns ponteiros de funções usadas para controlar o dispositivo, tal como, transmitir pacotes, manipular cabeçalhos de hardware etc.

O pacote é armazenado em uma região de memória com uma estrutura FIFO (First Input First Output) chamada *rx-ring [33].* Depois da recepção do pacote, o adaptador usará uma interrupção para informar à *cpu* deste evento. Um novo buffer de rede *sk_buff*[2] (socket buffer) é alocado e o pacote é copiado neste [17].

---

[2] Um buffer de rede Linux é uma estrutura de dados sk_buff definida no *include/linux/skbuff.h*

O tempo gasto para o processo de interrupção é crítico e deve ser mantido ao mínimo. Os pacotes podem se acumular no *rx-ring* e alguns pacotes podem ser descartados se não houver mais *tokens* disponíveis. Uma vez que o pacote é transferido ao sk_buff, este é passado às camadas mais acima do modelo OSI.

4.1.2. O buffer de rede sk_buff

Um *sk_buff* é uma estrutura de controle adicionada a um bloco de memória. Existem dois tipos primários de funções providas na biblioteca *sk_buff*. Primeiramente, rotinas para manipular listas duplamente enlaçadas de sk_buff, segundo funções para o controle da memória. Os buffers enlaçados nas listas para operações de rede, são adicionados ao final das listas e removidos no inicio. A maioria das operações de rede acontece durante as interrupções. Estas operações são usadas para manipular grupos de pacotes. A memória é usada para manipular as rotinas de obtenção do conteúdo do pacote de forma padronizada e eficiente.

4.1.3. Netfilter – camada três

Netfilter [28] ou filtro de rede recebe os pacotes que entram na camada três, processados pela função *ip_rcv*. Neste ponto o *hook* do netfilter pode dar um tratamento particularizado tal como packet filtering, packet mangling, NAT, Forwarding, etc. Qualquer módulo do kernel pode usá-lo tanto para pacotes que entram ou que saem segundo a configuração de parâmetros *hook*. Em um hook pode ser decidido se o pacote é apagado ou pode continuar, às vezes, depois de uma modificação (mangling) [18].

O Netfilter está atualmente implementado para ipv4, ipv6 e DECnet. Basicamente existem cinco tipos de hooks: NF_IP_PRE_ROUTING, NF_IP_LOCAL_IN, NF_IP_FORWARD, NF_IP_POST_ROUTING, NF_IP_LOCAL_OUT.

4.2. Técnicas de Estimativas para Provas Ativas

Nas comunicações da camada física, o termo largura de banda é relacionado com a largura espectral de sinais eletromagnéticos ou com as características da propagação dos sistemas de comunicações. No contexto de redes de dados, o termo largura de banda, quantifica a taxa de dados que um enlace de rede ou um trajeto da rede pode transferir. A análise de provas ativas se focaliza na estimativa de medidas da largura de banda neste último contexto.

Os desenvolvimentos para calcular a largura de banda disponível, são baseados em uma serie de técnicas que medem de forma ativa ou passiva a quantidade de bits que podem trafegar em uma conexão para um determinado período. Muitos deles podem fornecer estimativas exatas sob determinadas circunstâncias [1]. Algumas iniciativas para realizar testes para estimar a largura de banda de forma ativa, passiva, medidas e estudos de avaliações parciais das ferramentas já foram publicadas, embora nenhum resultado substancial tenha sido relatado para a estimativa ativa [2] [3].

4.3. Análise de técnicas

Nas análises de cálculos para realizar estimativas de largura de banda, foram avaliadas simultaneamente técnicas e ferramentas para determinar a capacidade do canal, a largura de banda disponível e a estimativa da capacidade de transferência, o qual permitirá abstrair um modelo comum. Nesta subseção, é apresentada uma estrutura genérica para representar uma estimativa ativa da largura de banda.

Uma revisão de artigos e de códigos fontes de algumas ferramentas ativas para calcular a largura de banda foi realizada. Foram verificados os códigos de **pathChirp** [4] que foi desenvolvido usando o código de **NetDyn** [5] como ponto de início, **IGI** [8] que introduz o conceito de *packet transmission rate* (PTR), **pathLoad** e **pathRate** [9], **cprobe** [6] e **pipechar** [7] os quais têm em comum a realização de medidas baseadas em técnicas de envio de pacotes de prova tais como *packet pair* e *packet train* [8].

Duas etapas são identificadas nestas ferramentas avaliadas: Medida e Estimativa. A medida envolve a realização de um teste com um pacote de prova padrão, sua transmissão através da rede e a medida da recepção. A estimativa compreende o processo estatístico e heurístico das medidas de acordo com algum modelo da rede.

Três medidas estão constantemente associadas para estimar a largura de banda: *capacidade*, *largura de banda disponível, Bulk-Transport-Capacity (BTC).* A *capacidade* é o *throughput* máximo que o trajeto pode fornecer a uma aplicação quando não há nenhuma carga de tráfego. A *largura de banda disponível* é o *throughput* máximo fornecido por uma aplicação dada à carga transversal atual do tráfego do trajeto. O *BTC* [10] define a capacidade de transferir quantidades significantes de dados durante uma conexão.

Existem diferentes técnicas utilizadas para encontrar estas medidas, tais como *Variable Packet Size* (**VPS**), que estima a capacidade de cada hop ao longo do trajeto, *Packet Pair/Train Dispersion* (**PPTD**), que estima a capacidade end-to-end [42], *Self-Loading Periodic Streams* (**SLoPS**) e *Trains of Packet Pairs* (**TOPP**) que são utilizados para estimar a largura de banda disponível.

O *VPS* mede o RTT desde a fonte até cada hop do trajeto. O VPS usa o campo do cabeçalho IP Time-To-Live (TTL) para forçar os pacotes a finalizar o trajeto em um hop determinado. O router neste hop descartará o pacote retornando uma mensagem de erro ICMP "Time-exceeded". A fonte usa o pacote ICMP recebido para medir o RTT até este hop. Desafortunadamente o VPS pode produzir erros significativos se a medida do trajeto incluir switches store-and-forward.

O *Packet Pair* estima a capacidade end-to-end do trajeto, quando a fonte envia múltiplos packet pair ao receptor [36]. Cada packet pair consiste em dois pacotes do mesmo tamanho enviados back-to-back. A dispersão do packet pair em um enlace específico do trajeto é a diferença de tempo entre a chegada de cada pacote ao receptor.

O *Packet Train* estende a técnica *packet pair* usando múltiplos pacotes back-to-back. A dispersão de um packet train no enlace é a quantidade de tempo entre o último bit do primeiro pacote e o último bit do último pacote.

O *SLoPS* estima a largura de banda end-to-end enviando um número definido de pacotes de igual tamanho ao receptor, com uma determinada taxa de transmissão. O método consiste em monitorar as variações no *delay* dos pacotes de prova [43]. Se a taxa do fluxo for maior que a largura de banda disponível do trajeto, o fluxo causará sobrecarga no enfileiramento da menor largura de banda disponível do enlace.

O *TOPP* estende a técnica *packet train, este* estima a largura de banda enviando vários *packet pair* incrementando gradativamente a taxa de transmissão entre cada par.

4.4. TOPP

O *Train Of Packet Pair* (TOPP) é uma medida usada para estimar a largura de banda baseada em provas ativas e inclui análises por regressão segmentada [45]. Este método pode estimar a medida da largura de banda disponível em um enlace congestionado. Ao contrário das estimativas tradicionais do packet pair, esta estimativa não está limitada pela taxa. Este método detecta engarrafamentos que são invisíveis nos outros métodos, tenta solucionar alguns problemas que outros métodos não fazem. Estes problemas incluem a inabilidade de identificar situações de diferentes tipos de engarrafamento que ocorrem sobre enlaces separados.

4.4.1. Compartilhamento proporcional

Quando um enlace sobre uma rota tem uma largura de banda "*l*", um tráfego com uma taxa "*mi*", e o emissor introduz um tráfego com uma taxa "*o*" tal que "$m < l < m + o$", existirá então uma situação de sobrecarga. Nesta situação, a política comum de um router é a de *First Come First Serve* (FCFS), que assume um escalonamento proporcional e a

política de descartar pacotes em caso de sobrecarga do buffer, a fim de prover uma largura de banda segundo a taxa oferecida dada na equação (2).

$$f = o*l / (m + o) \qquad (2)$$

Neste esquema são assumidos enlaces compartilhados proporcionais na rota. Sobre esta concepção existiriam larguras de bandas proporcionais para cada enlace o que daria problemas de engarrafamento escondido, que tenta solucionar o método TOPP.

4.4.2. Surplus Bandwidth

Surplus bottleneck bandwidth [45], significa que um novo emissor pode transmitir a maior taxa possível sem afetar o tráfego sobre a rota, diminuir a taxa total ou começar a perder pacotes.

Assume-se que a soma do tráfego no enlace "*li*" tem uma taxa substancial de "*mi*", então o *surplus bandwidth* no enlace será "*si = li – mi*" quando *li* ≥ *si* para todo *i*. O *surplus botleneck bandwidth* será então "*sb = min{s1, s2, ..., si}*".

4.4.3. Método de medida do TOPP

O método de medida do TOPP é formado por duas fases separadas. A primeira consiste na fase de provas ativas, na qual, os pacotes de provas são transmitidos através da rede. A segunda fase consiste de análises na qual a largura de banda é estimada baseada nos tempos de recepção dos pacotes de prova.

Na ***fase de provas*** do TOOP, é gerado um tráfego de pacotes de prova começando com uma taxa mínima "o_{min}" de "*n*" pares de pacotes de igual tamanho. Após o envio destes "*n*" pacotes, a taxa oferecida é incrementada em "Δ*o*" e deste modo, outra série de "*n*" pacotes é enviada. Este procedimento é realizado até que "*o*" alcance "o_{max}" que indicará o final da fase de provas.

O tempo entre cada *packet pair* "ΔT" é selecionado de tal maneira que o enfileiramento entre dois pacotes de prova seja pequeno. Isto beneficia fase de provas, pois os nós ao longo do trajeto não experimentarão *bursts* longos [35].

No lado do receptor, os pacotes de prova recebem a medida do tempo da recepção (timestamp) e são reenviados à origem, isto é realizado em um único sentido para prevenir o problema de trajetos assimétricos.

A ***fase de análise*** é baseada no princípio do efeito do espaçamento no engarrafamento. Tendo-se "*b*" como tamanho dos pacotes e um tempo de separação "ΔR", a largura de banda experimentada através do enlace pode ser estimada com a seguinte equação:

$$f = b / \Delta R \qquad (3)$$

4.5. Queue Discipline

Queue discipline (Qdisc) [27] é uma série de técnicas de enfileiramento que permitem mudar a maneira em que os dados são enviados, podendo ser utilizado para dar forma ao tráfego, possibilitando assim o controle do engarrafamento do enlace pela interface de rede.

Normalmente, os pacotes são enfileirados através da técnica FIFO (Fisrt In, First Out), enviando aproximadamente um pacote a cada interrupção, conforme mostra a figura 1.
O mecanismo "*queue discipline*" usado por default pelos dispositivos de rede é chamado de *pfifo_fast*[3].

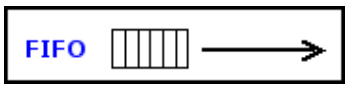

Figura 1. FIFO

O *pfifo_fast* significa que nenhum pacote recebe um tratamento especial [15]. Este utiliza três bandas onde é aplicada uma regra FIFO. Quando uma das bandas está sendo

[3] pfifo_fast definido em net/sch/sch_generic.c

utilizada, as outras duas ficam em espera, não permitindo o uso das bandas simultaneamente. As regras FIFO estão diretamente relacionadas com o campo do cabeçalho IP, Type of Service (TOS) e cuidam para que a banda zero tenha o mínimo delay.

O TOS determina como as prioridades atribuídas ao pacote são mapeadas nas bandas. A maioria das técnicas FIFO pode enviar pacotes **P** alcançando uma taxa de transmissão máxima de pacote multiplicado por HZ[4] conforme mostra a equação 4:

$$Rate = P * HZ \qquad (4)$$

Este processo pode funcionar de maneira assíncrona devido à entrada de pacotes no sistema. Com este tipo de técnica, o envio de pacotes na rede a uma taxa fixa poderia ocasionar congestionamentos e consequentemente perdas [15].

Outros tipos de "*queue discipline"* como *Token Buket Filter (TBF)* ou *Stochastic Fairness Queueing (SFQ)* entre outros, dão um tratamento especial ao fluxo de pacotes, como por exemplo, limitar a taxa de transmissão, latência, burst, além de permitir a aplicação de classes de serviço comumente utilizados em Quality of Service (QoS).

4.6. TCP Pacing

Este projeto é chamado de tcp_pacing [39] porque cria espaços de tempo entre os pacotes, com a finalidade de variar a taxa de transmissão, tentar ajustar o mecanismo *congestion avoidance* dando um melhor desempenho ao protocolo TCP [19]. No tcp_pacing todos os segmentos são enviados dentro de certo espaço de tempo. Este projeto requer as seguintes mudanças no TCP:

1. Detecção do tempo de inatividade, o que indica que o TCP necessita ser começado forçando um slow start evitando um slow-start-restart.
2. Estimativa da largura de banda, método dado por TCP-Vegas.

[4] HZ, constante de freqüência das interrupções do kernel.

3. Cálculo da janela esperada e o sincronismo entre segmentos nessa janela.
4. Um mecanismo de relógio para emitir os segmentos.

4.7. Datagram Congestion Control Protocol - DCCP

O DCCP é um protocolo de transporte que provê controle de congestionamento e fluxo controlado de datagramas sem confiabilidade para aplicações de delay sensitiva, tal como, mídia streaming, telefonia e games, onde se prefere timeliness à confiabilidade, ajustando a transmissão baseada em taxa [41] (rate based) sobre um congestionamento de retorno [20].

4.8. MMTP – Multimedia Multiplexing Transport Protocol

O MMTP é um protocolo que atua na camada de aplicação em conjunto com o UDP da camada de transporte. Este protocolo provê mobilidade utilizando múltiplas interfaces de rede, enviando dados de maneira multiplexada balanceando as cargas de forma confiável, de tal modo que, se qualquer uma das interfaces estiver desabilitada para o envio de dados, as outras continuam realizando esta atividade, aumentando a carga em cada dispositivo de rede disponível. Este também suporta o controle de congestionamento, capaz de estimar a largura de banda utilizando a técnica packet pair, permitindo controlar a taxa de transmissão dos pacotes [21].

Este protocolo combina o controle da taxa de transmissão com o controle de congestionamento. O algoritmo está dividido em três fases que são:

1. Exponential increase: experimenta a rede com a finalidade de determinar a largura de banda disponível com pacotes de prova, ajustando o tempo de envio dos pacotes segundo os resultados das operações com as medidas capturadas.
2. Congestion avoidance: interpreta a seqüência de jitters positivos que mostram que a rede está carregada e que, os queues estão sendo incrementados anunciando que a taxa deve ser regulada.

3. Congestion Control: Notifica que a rede está congestionada devido às perdas relatadas, de tal maneira que adapta o tamanho da janela com um determinado número de segmentos.

4.9. Traffic Shaping

Traffic shaping é o termo geral dado a uma série de técnicas projetadas para priorizar políticas de transmissão de dados sobre um enlace da rede. Alguns efeitos de latência e enfileiramento podem ser experimentados com um retardo dos pacotes nas comunicações de rede, quando há tráfego significativo. Este problema ocorre porque não são dadas condições quando o pacote é enfileirado para a transmissão utilizando a técnica qdisc baseada em FIFO e a transmissão é escalonada com "First Come, First Serve" (FCFS).

4.9.1. Shaper Device

Shaper Device é um mecanismo simples suportado pelo kernel de Linux, projetado para limitar a quantidade de largura de banda que cada trajeto pode utilizar. Shaper Device é um dispositivo de rede virtual configurado com dois importantes parâmetros: o limite da largura de banda e a interface física de rede ao qual é aderido. A soma total do tráfego da rota via "shaper device" será limitado pela capacidade da largura de banda, descartando qualquer pacote que a exceda.

4.9.2. QoS Packet Scheduling

Enquanto o "shaper device" é muito utilizado em várias circunstâncias, muitas aplicações demandam mais flexíveis e sofisticadas técnicas shaping, tal como, padrões de "Quality of Service" (QoS). As técnicas mais conhecidas são: "Resource ReSerVation Protocol" (RSVP) descrito no RFC-2205, "Integrated Service" descrito no RFC-1633 e "Differentiated Service" descrito no RFC-2475.

Os mecanismos QoS mais importantes de classificação de pacotes são RSVP, firewall marking, route-based e Ugly 32-bit key; scheduling disciplines, tal como, Class-Based Queues (CBQ), Token Bucket Filter (TBF) [40], 3-band priority queues, Random Early Drop (RED), Stochastic Fairness Queueing (SFQ). Cada mecanismo de classificação e de escalonamento pode ser combinado em múltiplas formas conhecendo-se a variedade do QoS e os requerimentos de tráfego IP.

5. DESENVOLVIMENTO

5.1. Modelagem do mecanismo e2e

O mecanismo e2e fornece uma série de funções para distinguir a melhor interface de rede e permitir o envio de pacotes a uma taxa variável, dependendo do congestionamento do enlace.

Este mecanismo experimenta a rota enviando pacotes de prova por todas as interfaces de rede ativas e determina qual pode fornecer a melhor taxa de transmissão informando o enlace menos congestionado, o que seria mais eficiente para enviar os pacotes. Este modelo do mecanismo permite que um protocolo de transporte envie os dados, diminuindo a probabilidade de perda de pacotes ajustando-se à capacidade do canal, minimizando o congestionamento.

5.1.1. Requisitos gerais

O mecanismo e2e cumpre com os requisitos que orientam o envio de pacotes aplicando uma taxa de transmissão variável com o propósito de minimizar as perdas de pacotes causadas pelo congestionamento em um determinado enlace. Para satisfazer esta necessidade são requeridas as seguintes características:

- *Medidas* – Envio de pacotes sintéticos que não contêm dados da aplicação e são enviados para capturar medidas de tempo.
- *Estimativas* – Cada medida realizada com os pacotes de prova deve prover informações suficientes para a realização de cálculos estatísticos e determinação da taxa de transmissão.
- *Controle da taxa* – As medidas de tempo capturadas deverão prover estimativas para variar a taxa de transmissão.
- *Seleção da interface de rede* – Segundo as estimativas calculadas, deverá ser possível determinar qual das interfaces ativas pode fornecer uma maior taxa de transmissão reduzindo a probabilidade de perda de pacotes.

- Configuração – O mecanismo deve permitir flexibilidade de configuração desde a camada de aplicação.

5.1.2. Atores e processos relevantes

Atores	**Processos relevantes**
Mecanismo e2e	Guardar configuração do enlace realizada pela aplicação. Enviar pacotes de prova por todas as interfaces ativas de rede. Receber os pacotes de prova e capturar as medidas. Realizar as estimativas com as medidas capturadas. Identificar qual é a melhor interface para transmitir os pacotes. Determinar a taxa de transmissão com que os pacotes devem trafegar na rede. Detectar os pacotes da aplicação a serem controlados e transmiti-los dependendo da taxa calculada. Detectar a inatividade da aplicação e parar de realizar o controle. Detectar atividade da aplicação e recomeçar o controle.
Aplicação	Informar ao mecanismo e2e a configuração do enlace que vai ser controlado. Ativar e desativar o controle realizado pelo mecanismo e2e. Transmitir pacotes.
Receptor	Receber os pacotes de prova e responder com as medidas de tempo necessárias, para que o mecanismo e2e realize as estimativas.

5.1.3. Casos de uso

5.1.3.1. A aplicação ativa o controle de fluxo

Atores: Aplicação e o mecanismo e2e.
Objetivo: A aplicação informa ao mecanismo e2e as configurações do enlace para ativar o controle de fluxo.
Visão geral: A aplicação envia o comando de ativação do controle de fluxo, o mecanismo e2e recebe as configurações e as guarda para iniciar a variação da taxa.

Aplicação	**Mecanismo e2e**
1. Abre a conexão	2. Aceita a conexão
3. Cria comando com as configurações do enlace	
4. Envia comando de ativação	5. Recebe comando de ativação
	6. Guarda configuração do enlace
	7. Ativa o controle de fluxo
8. Inicia o tráfego dos pacotes.	

Curso alternativo:

- *Caso 6*: Se o controle de fluxo havia sido ativado e estava em estado desativado o controle é reativado e as configurações não são guardadas.

5.1.3.2. A aplicação desativa controle do fluxo

Atores: Aplicação e o mecanismo e2e.
Objetivo: A aplicação termina as atividades na rede e informa ao mecanismo e2e que não requer mais controlar o fluxo.
Visão geral: A aplicação manda ao mecanismo e2e o comando de desativação, o mecanismo e2e desativa as funções de controle de fluxo, apaga configurações e fecha a conexão da aplicação.

Aplicação	Mecanismo e2e
1. Termina as atividades na rede.	
2. Envia o comando de desativação	3. Recebe o comando de desativação
	4. Desativa o controle
	5. Apaga as configurações do controle
6. Fecha a conexão	7. Fecha a conexão

5.1.3.3. O mecanismo e2e envia pacotes de prova para estimar a taxa

Atores: Mecanismo e2e, receptor.
Objetivo: O mecanismo e2e envia pacotes de prova para cada destino através de todos os enlaces configurados na aplicação, recebe as medidas de tempo e estima a taxa.
Visão geral: O mecanismo envia pacotes de prova para medir o atraso que existe no trajeto, com base nestas medidas realiza as estimativas e determina qual é a taxa com que os pacotes devem trafegar no enlace.

Mecanismo e2e	Receptor
1. Cria um par de pacotes de prova	
2. Envia cada pacote Informando o tempo de saída	3. Recebe cada pacote informando o tempo de chegada
5. Captura os pacotes de prova	4. Devolve cada pacote informado o tempo de saída do receptor
6. Recebe as medidas de tempo dos pacotes de prova.	
7. Calcula o RTT de cada pacote de prova	
8. Calcula o delay produzido por cada par	
9. Realiza as estimativas	
10. Informa a nova taxa no enlace	
11. Controla o fluxo com os dados das estimativas calculadas	

Curso alternativo:

Caso 5: Se o número de seqüência das provas ativas não coincidirem com o valor esperado, a seqüência deverá ser descartada.

5.1.3.4. O mecanismo e2e controla a taxa

Atores: Mecanismo e2e, aplicação.
Objetivo: O mecanismo e2e varia as configurações da taxa para cada enlace e controla o fluxo dos pacotes.
Visão geral: O Mecanismo e2e recebe as estimativas feitas com as medidas dos pacotes de prova, varia a taxa do enlace configurado pela aplicação e efetua o controle de fluxo.

Mecanismo e2e	**Aplicação**
	1. Envia um fluxo constante de pacotes
2. Recebe as estimativas	
3. Identifica as configurações do enlace da aplicação	
4. Varia a configuração da taxa de transmissão	
5. Limita o envio de pacotes na rede	

5.1.3.5. O mecanismo e2e detecta inatividade da aplicação no enlace

Atores: Mecanismo e2e, aplicação.
Objetivo: O mecanismo e2e determina um período de inatividade para desativação do controle de fluxo.
Visão geral: A aplicação interrompe a transmissão de dados, o enlace detecta a inatividade da aplicação através de um temporizador e desativa o controle de fluxo.

Mecanismo e2e	**Aplicação**
	1.Interrompe a transmissão.
2. Recebe um timeout relacionado com a inatividade da aplicação.	
3. Para de medir e estimar as diferentes variáveis do enlace.	
4. O enlace entra no estado inativo	
5. Ativa o temporizador para fechar o controle de fluxo.	
6. Entra em estado de espera para reativação.	

Curso alternativo:

Caso 3: O mecanismo e2e espera por um novo fluxo de pacotes enviados pela aplicação para reativar o controle de fluxo.

Caso 4: Se o tempo de reativação do temporizador for esgotado, o mecanismo e2e fechará a conexão e apagará as configurações do enlace.

5.1.4. Diagrama de casos de uso

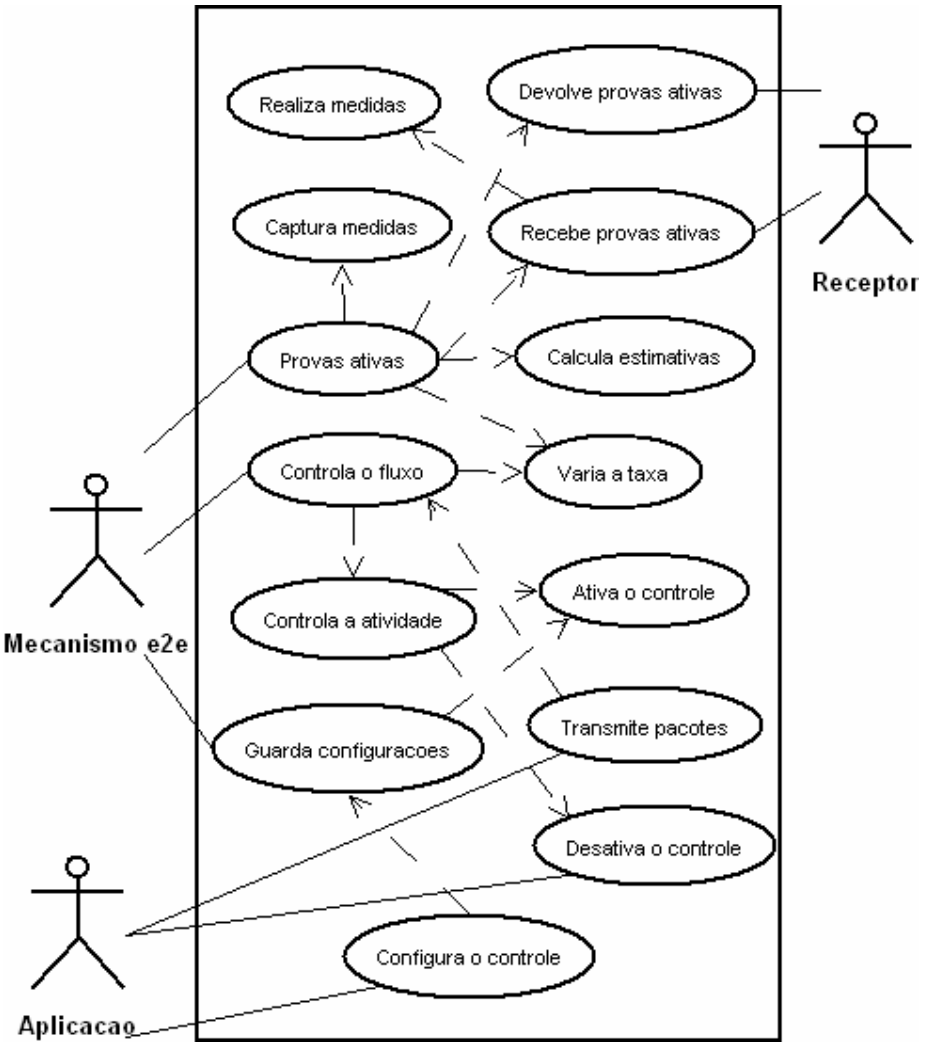

Figura 2. Diagrama de casos de uso.

5.2. Diagrama de classes

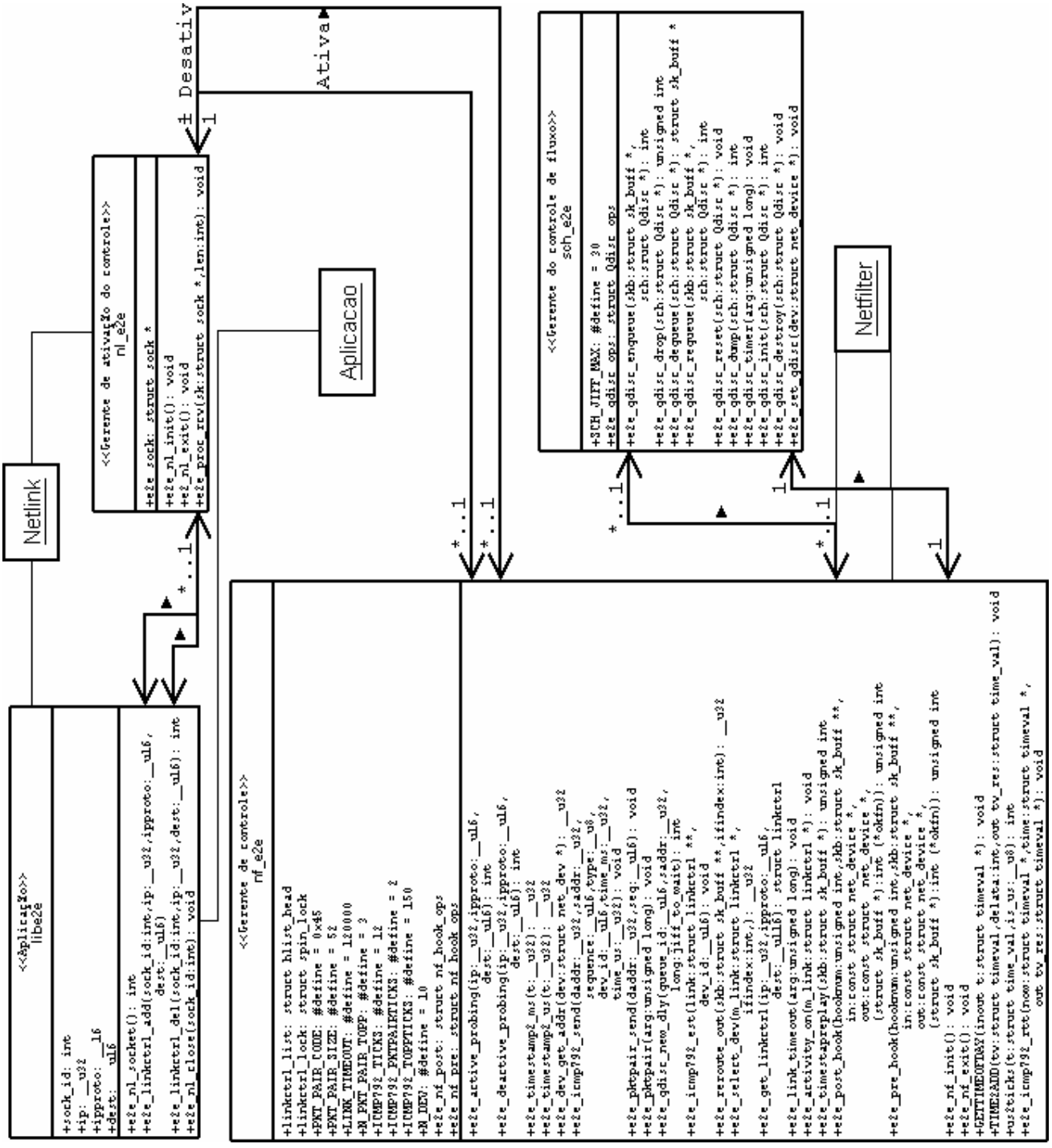

Figura 3. Diagrama de classes.

5.3. Diagramas de interação

O diagrama de interação dará a visão de como cada componente do projeto é relacionado às atividades e funções aplicadas para realização do controle de fluxo. Em cada diagrama apresentado é usada a conotação de gerente, para relacionar cada módulo do projeto da seguinte forma:

- Gerente de ativação ou nl_e2e (netlink_e2e)
- Gerente de controle ou nf_e2e (netfilter_e2e)
- Gerente de controle de fluxo ou sch_e2e (scheduler_e2e)

5.3.1. Ativação do controle de fluxo

O *gerente de ativação* é o meio de comunicação entre a aplicação e o *gerente de controle* como é mostrado na figura 4.

A aplicação deve abrir um socket de netlink com e2e_nl_socket e fornecer as variáveis de ativação com a função e2e_linkctrl_add. O gerente de ativação captura a configuração dada pela aplicação e as entrega ao *gerente de controle*. Conseqüentemente a configuração é guardada em uma lista com a função hlist_add_head e finalmente, o mecanismo é aplicado com a função e2e_activity_on.

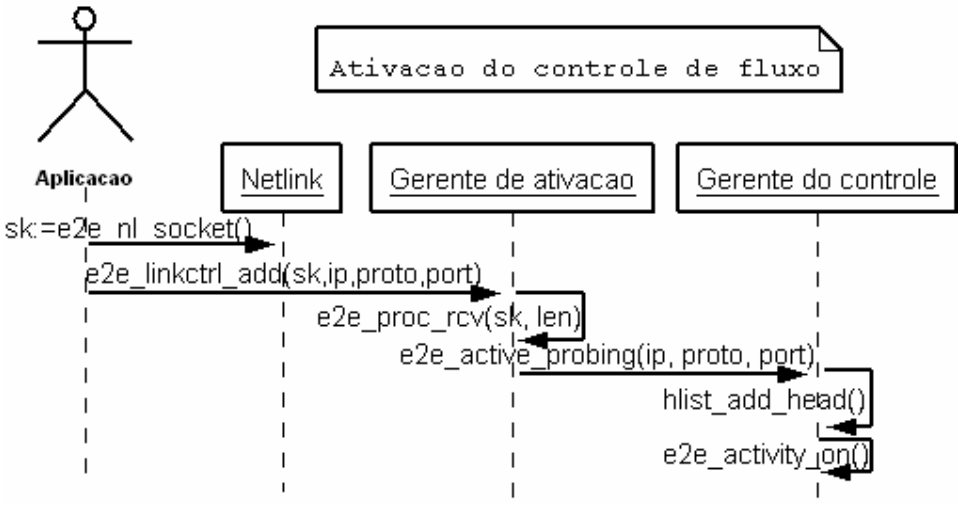

Figura 4. Diagrama de interação para ativação do controle de fluxo.

5.3.2. Cálculo de estimativas e variação da taxa

O *gerente de controle* envia pares de pacotes ao *receptor* com a função e2e_pktpair_send para realizar as medidas de "*interarrival time*", como é mostrado na figura 5. O *receptor* adiciona aos pacotes o tempo de chegada e os devolve. O *Netfilter* filtra os pacotes de prova entrantes e os passa para o *gerente de controle* com a função e2e_pre_hook, a qual extrai as medidas de tempo. Identifica a estrutura de configuração na lista de ativação com a função e2e_get_linkctrl, converte para microsegundos se as medidas estiverem em milisegundos utilizando a função timems2us e realiza os cálculos de estimativas com a função e2e_icmp792_est. O *gerente de controle* utiliza um sk_buff para enviar os resultados das estimativas ao gerente de controle de fluxo, utilizando a função e2e_qdisc_new_dly. O sk_buff é capturado pelo *gerente de controle de fluxo* com a função e2e_qdisc_enqueue identificando as novas configurações e variando a taxa no respectivo queue. Finalmente o sk_buff de configuração é descartado com kfree_skb.

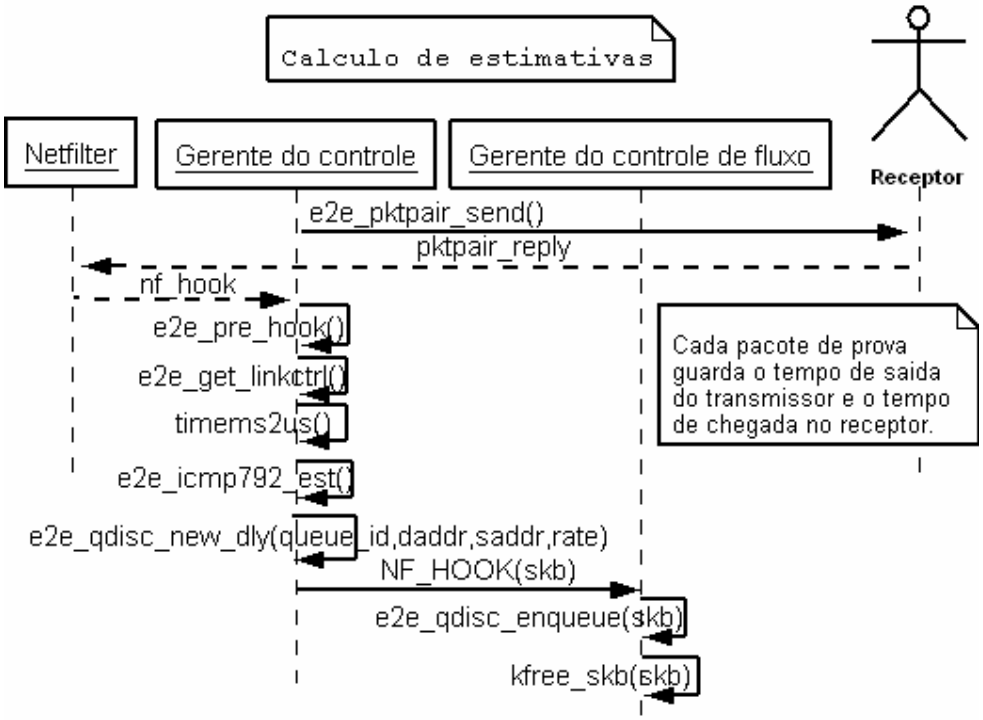

Figura 5. Diagrama de interação para cálculo de estimativas.

5.3.3. Controle do queue

A *aplicação* transmite os pacotes utilizando normalmente a função sendto do C. Os pacotes são filtrados com *Netfilter* e entregues ao *gerente de controle* com a função e2e_post_hook. Cada IP, porta e protocolo do pacote são comparados com a lista de configuração do gerente de controle com a função e2e_get_link_ctrl, cujo resultado, é um identificador de queue (qid). Cada pacote é marcado com este identificador, continuando o trajeto para a camada dois (o queue deve ser identificado porque as variações de taxa são diferentes em cada um). O gerente de controle de fluxo captura o pacote (sk_buff, skb) com a função e2e_qdisc_enqueue, identifica o queue com a função get_qdisc e enfileira o pacote neste (enqueue). A seqüência de funções para este processo pode ser vista na figura 6. Cada sk_buff armazenará o identificador do queue para ser enviado ao *gerente de controle de fluxo* e garante que seja enfileirado no respectivo queue. Se os pacotes não são identificados pelo gerente de controle, o sk_buff não recebe nenhum tratamento especial e é enfileirado no primeiro queue onde não haverá controle de taxa.

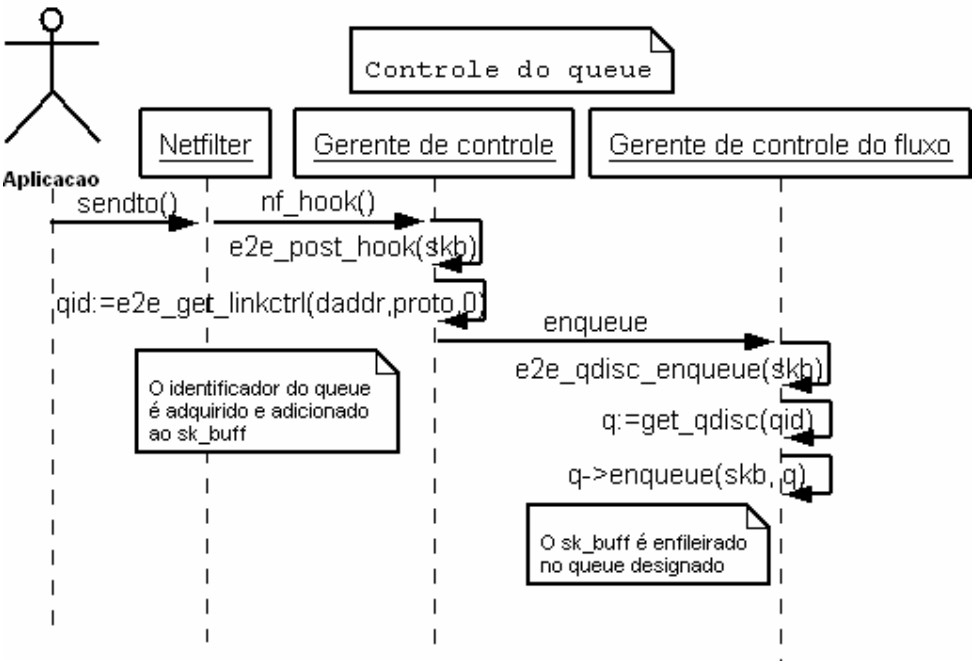

Figura 6. Diagrama de interação para o controle do queue

5.3.4. Seleção da interface de rede

As provas ativas constantemente ofereceram medidas de atraso todas às interfaces de rede disponíveis, estimando a taxa de transmissão. As taxas calculadas de cada interface de rede são comparadas com a função e2e_select_dev() retornando um identificador de interface de rede (dev_id), como se mostra na figura 7. A função e2e_reroute() altera a rota do sk_buff (skb) para alcançar o dispositivo de rede identificado. A função e2e_get_linkctrl identifica o queue desse dispositivo de rede, como se apresento no diagrama de interação do controle do queue, anteriormente descrito.

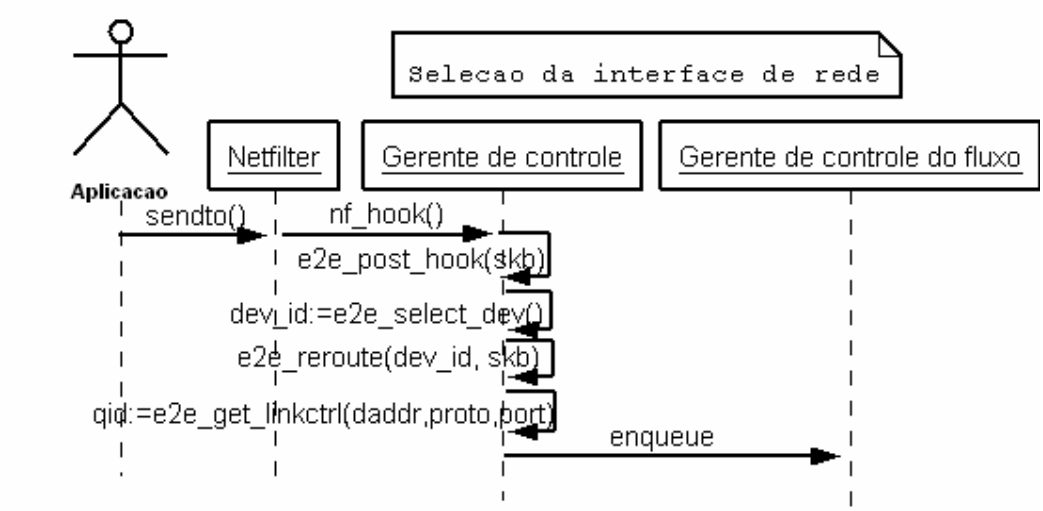

Figura 7. Diagrama de interação para seleção da interface de rede

5.4. Diagrama de atividades para o controle do dequeue

O controle do dequeue permitirá a transmissão dos pacotes que estão em espera no queue segundo a taxa estimada. A figura 8 apresenta as atividades do dequeue sobre vários queues. O evento time_to_send compara o tempo de envio com o tempo nesse instante. Se for o instante de ser enviado, o sk_buff é transmitido e o time_to_send adquire o valor estimado da taxa para esse queue que determinará o instante de saída do próximo pacote. Se não for o instante de saída, o sk_buff volta para o queue na atividade requeue e avança para o próximo queue continuando as atividades do dequeue.

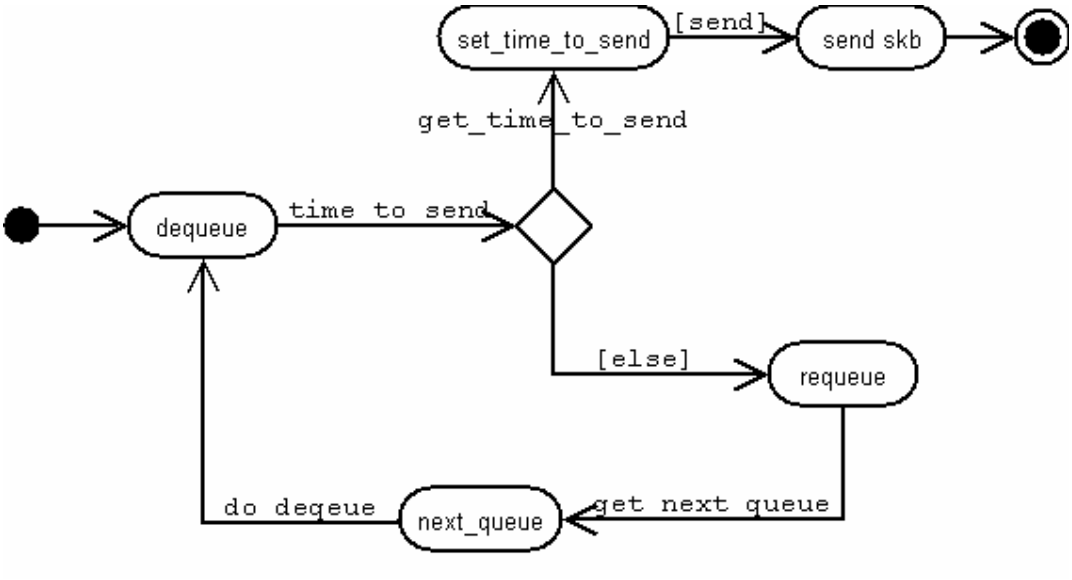

Figura 8. Diagrama de atividades para o controle do dequeue

6. MECANISMO E2E

O mecanismo e2e é uma iniciativa para adquirir resultados de controle de congestionamento ortogonal aos protocolos de transporte utilizados, como TCP e UDP. Foi elaborado nas camadas mais baixas do modelo de referência OSI, com um desenvolvimento ao nível de kernel [30]. Este projeto implementa uma série de técnicas como Packet Pair, Train Of Packet Pair, seleção da melhor interface de rede para o envio de pacotes e queue disciplines. O mecanismo e2e é formado por diferentes módulos de kernel próprios do desenvolvimento que podem ser facilmente classificados dentro do modelo OSI, como está representado na figura 9.

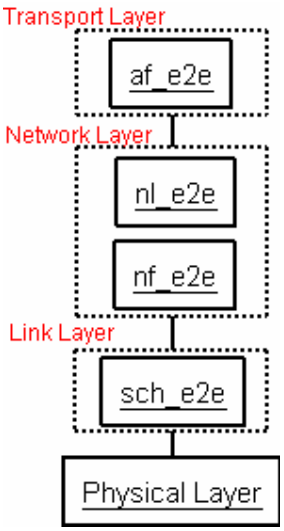

Figura 9. Representação dos módulos nas camadas do modelo OSI

O af_e2e, é o módulo que representa o protocolo de transporte, define o nome do protocolo e registra no kernel as funções de socket necessárias para estabelecer as comunicações entre a camada de aplicação e a camada de transporte. Pelo fato deste módulo levar mais tempo do que o estabelecido para a conclusão de uma tese de mestrado, este não foi totalmente desenvolvido e foi deixado o código como modelo para futuros desenvolvimentos.

Para substituir esta parte do protocolo, foram utilizadas funções de netfilter através do módulo nf_e2e que realiza a captura dos pacotes relacionados com o controle de fluxo.

Funções de netlink através do módulo nl_e2e para comunicar o módulo de nf_e2e com a aplicação e estabelecer as configurações do enlace.

A aplicação envia uma chamada de netlink utilizando como meio o módulo nl_e2e para informar ao módulo nf_e2e o registro do controle de fluxo do enlace, começar as provas ativas, realizar as medidas e estimar as diferentes variáveis. Estas provas ativas permitirão determinar de maneira constante a quantidade de pacotes por segundo que podem ser enviados na rede quando há um tráfego significativo, utilizando a técnica de "queue discipline" determinada no módulo sch_e2e agindo na camada de enlace [32]. O módulo sch_e2e recebe as estimativas calculadas pelo módulo nf_e2e e identifica o queue que deverá receber a variação da taxa de transmissão.

6.1. ICMP Active Probing

O modulo nf_e2e, envia um par de pacotes ICMP segundo o RFC 792 do tipo *Timestamp* e *Timestamp Reply Message (13 e 14)* [15], [24], [5] cujas informações do cabeçalho são especificadas na Figura 10.

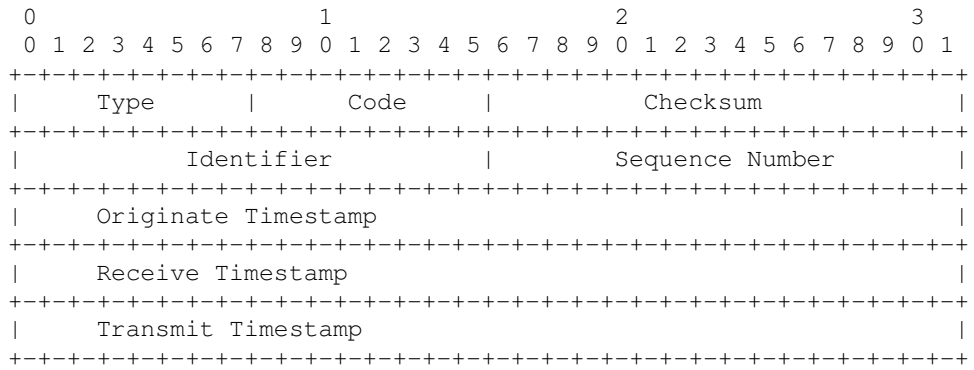

Figura 10. Cabeçalho ICMP tipo 13, 14.

Segundo as especificações do protocolo ICMP para TIMESTAMP, o tempo originado (originate timestamp) corresponde ao tempo de saída do pacote do emissor, o tempo recebido (receive timestamp), é o tempo de chegada do pacote ao receptor e o tempo transmitido, corresponde ao tempo de saída do pacote do receptor. Cada *timestamp* corresponde ao tempo em milisegundos desde a meia noite.

Os componentes principais dos cálculos de estimativas com provas ativas no *mecanismo e2e são*:

1. O *Active Probing:* envia de forma periódica um par de pacotes ICMP tipo 13 (ICMP_TIMESTAMP).
2. *O receptor*: recebe os pacotes ICMP tipo 13 e responde com o mesmo pacote mudando para o tipo 14 (ICMP_TIMESTAMPREPLY).
3. O *gerente de controle:* captura os pacotes de resposta das provas ativas.

O mecanismo pode ser interpretado nas quatro primeiras camadas do modelo OSI como é ilustrado na figura 11. O módulo nf_e2e envia provas ativas, que são pacotes ICMP tipo 13 com informação do tempo de saída usando todas as interfaces ativas de rede. Quando os pacotes são recebidos no receptor, uma resposta com um pacote ICMP tipo 14 é produzida com informação do tempo de chegada e tempo de saída. O gerente de controle usa o netfilter para identificar as respostas das provas ativas detectando a chegada dos pacotes, verificando as informações de tempo capturadas e realiza as estimativas. Cada resultado das provas ativas é associado a uma a interface de rede, isto permitirá a comparação de todos os resultados e obtendo-se critérios de seleção da interface de rede mais adequada para o envio de pacotes.

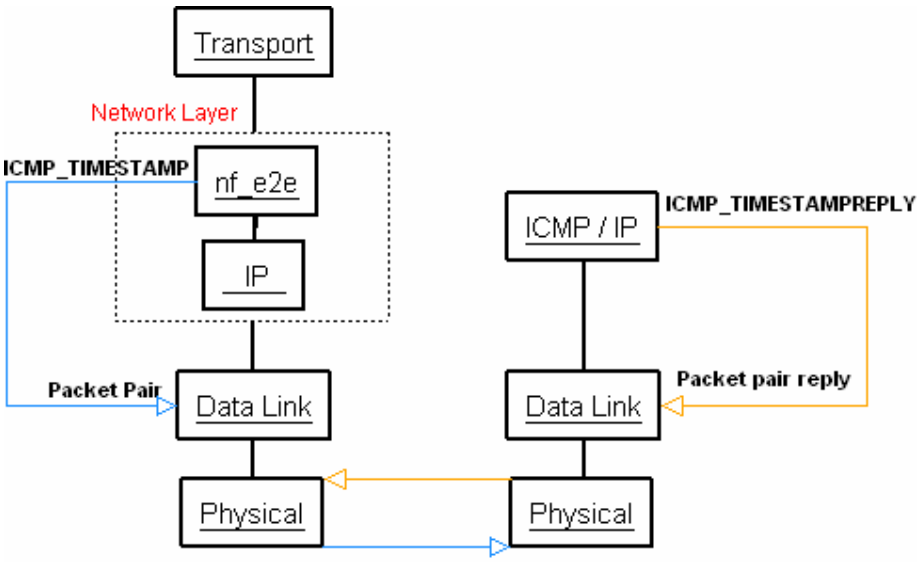

Figura 11. Provas ativas do mecanismo e2e.

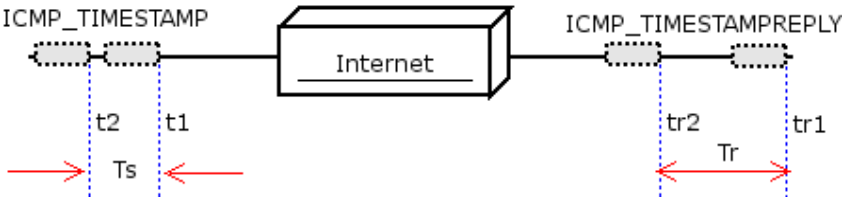

Figura 12. ICMP active probing

Como se ilustra na figura 12, o mecanismo de provas ativas permite conhecer a diferença entre o tempo de chegada dos pacotes ao receptor (Tr) como mostra a equação 5:

$$Tr = tr2 - tr1 \qquad (5)$$

Conhecendo-se o tempo de saída de cada pacote de prova, o período Ts pode ser calculado com a seguinte fórmula:

$$Ts_i = t2 - t1 \qquad (6)$$

A partir do resultado das equações anteriores, pode ser calculado o delay do enlace com a seguinte equação:

$$Delay = Tr - Ts \qquad (7)$$

Ou integrando a equação (5) e (6), o delay pode-se calcular de forma direta, como se mostra na seguinte equação:

$$Delay = (tr2 - tr1) - (t2 - t1) \qquad (8)$$

6.2. e2e - Active Probing

O RFC 792 especifica uma precisão de tempo em milisegundos [15], resultado adequado para redes cuja largura de banda é definida em Kbps ou Mbps, mas para redes de alta velocidade cuja largura de banda é definida em Gbps, seria necessária uma precisão de tempo em microsegundos para atingir a taxa estimada [23]. Com este propósito, foi

adicionado três campos de 32 bits ao cabeçalho ICMP informando o valor complementar em microsegundos, dos três primeiros campos como é mostrado na Figura 13.

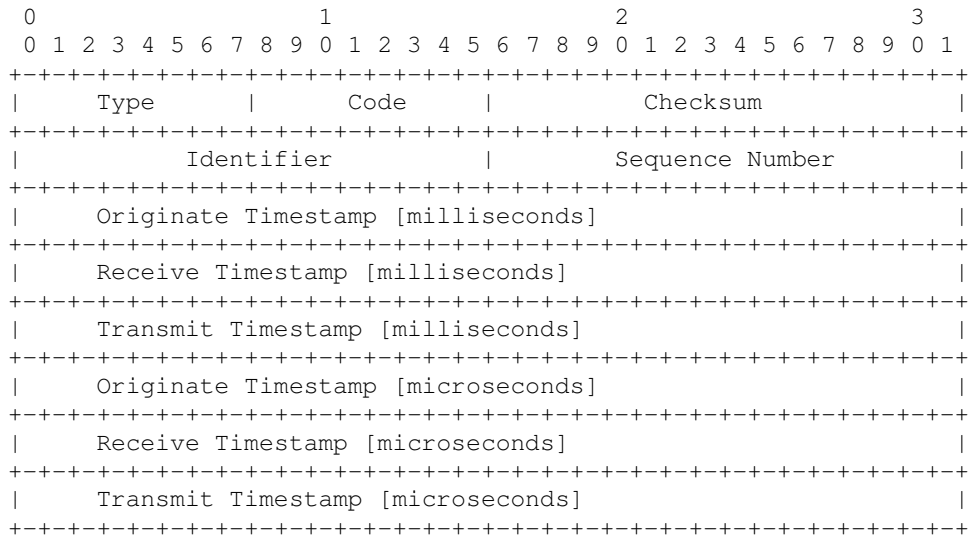

Figura 13. ICMP 792 com alterações do cabeçalho.

Esta modificação do cabeçalho permite adquirir medidas de timestamp completas (fornecidas pela função do_gettimeofday) e realizar estimativas para uma variação da taxa de transmissão com uma maior precisão. Para que este esquema funcione, é necessária a presença do modulo nf_e2e no receptor como é mostrado na Figura 14.

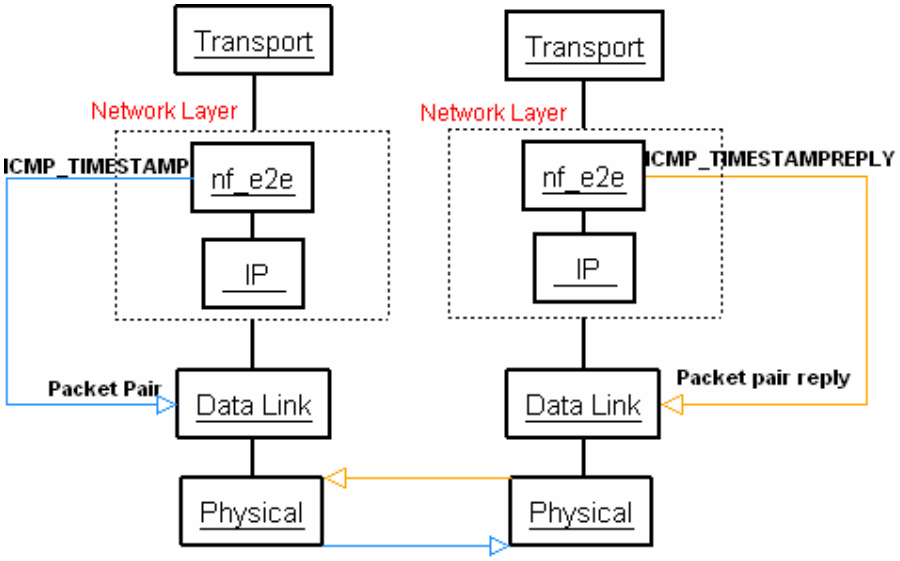

Figura 14. ICMP-e2e timestamp

Ainda que os resultados das estimativas sejam calculados com uma precisão em microsegundos, a variação da taxa de transmissão dependerá sempre da freqüência das interrupções do kernel (versão 2.6), o significa que, se a taxa é variada em microsegundos e cada interrupção acontece a cada milisegundo não seria obtido o efeito esperado. Para atingir a taxa alcançável com uma variação do delay em microsegundos, é necessário que o período entre cada interrupção ocorra a cada microsegundo.

6.3. TOPP do e2e

A técnica de provas ativas do mecanismo e2e utiliza o método TOPP para realizar as estimativas enviando um trem de ***N*** pares de pacotes a uma taxa ***Tc***, incrementa gradativamente o espaço entre os pacotes de cada par no instante de saída ***O*** com um período para cada trem ***Ts,*** como é mostrado na Figura 15**.**

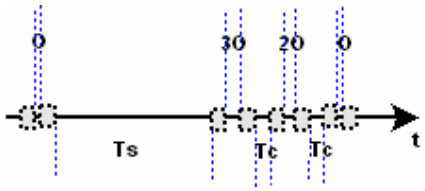

Figura 15. TOPP-e2e

A diferença da técnica TOPP que incrementa a taxa **Oi** em cada trem até **On** para finalmente realizar as estimativas, o mecanismo e2e incrementa **On** em cada par, é obtido como resultado uma estimativa para um único trem. O resultado final do delay será a média dos delays capturados no trem, como é mostrado na equação (9):

$$\text{Média\_delay} = (\text{delay}(\mathbf{0})+\text{delay}(\mathbf{1})+...+\text{delay}(\mathbf{n-1}))/\mathbf{n} \qquad (9)$$

As estimativas calculadas para cada seqüência TOPP no mecanismo e2e são: máximo e mínimo RTT, máximo e mínimo delay, média do delay e o moving average da média do delay. A variação da taxa de transmissão é realizada com o moving average da média do delay (Ma_delay) com é mostrado na equação (10):

$$\text{Ma\_delay} = (1-1/\alpha)\ \text{Ma\_delay} + \text{Média\_delay}/\ \alpha \qquad (10)$$

O número de interrupções (Ticks) que fornece a taxa estimada pode ser calculado com a seguinte equação:

$$Ticks = Ma\_delay * HZ \qquad (11)$$

6.4. Scheduler e2e

Para produzir o controle de fluxo baseado em taxa, foi desenvolvida uma nova técnica de "queue *discipline*" no módulo sch_e2e, baseada em enfileiramentos paralelos e controle da taxa tal como se mostra na figura 16.

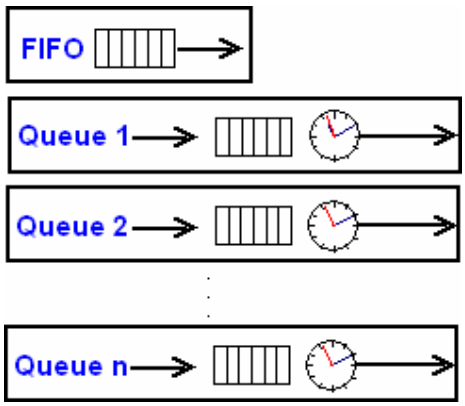

Figura 16. Esquema do scheduler sch_e2e

Este mecanismo deixa a primeira fila para utilizar a técnica convencional FIFO onde os pacotes são enfileirados sem precisar de um tratamento especial, tal como as provas ativas ou pacotes que não são configurados para o controle de fluxo.

As outras filas representam cada enlace configurado pela aplicação, o que quer dizer que cada fluxo é identificado e enviado para a respectiva fila, garantindo desta maneira que a taxa seja variável de forma independente em cada queue.

6.5. Mecanismo de controle de fluxo

O mecanismo de controle de fluxo depende da realização constante das provas ativas para adquirir constantemente as estimativas do delay. Quando os números de seqüências do

TOPP são completados no gerente de controle (nf_e2e), as estimativas são realizadas e entregues em um socket buffer (sk_buff) para o gerente de controle de fluxo (nl_e2e) guardando as informações dos resultados. Este sk_buff tem como finalidade configurar a nova taxa de transmissão que será aplicada nos pacotes que ainda estão no queue. O valor da taxa é aplicado depois de realizadas as estimativas, o que garante variabilidade.

O controle de fluxo depende da função dequeue do módulo sch_e2e o qual tem um token que percorre todos os queues verificando se os pacotes podem ser enviados, dependendo do tempo de saída, como é mostrado na figura 17.

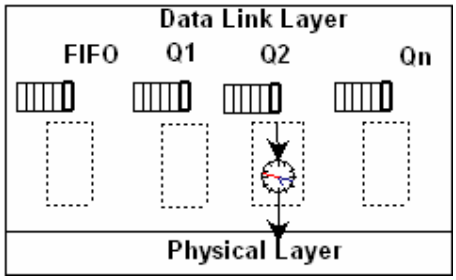

Figura 17. Mecanismo de dequeue

O primeiro queue efetuará o dequeue sempre que houver um pacote. O token continua com o próximo queue na seguinte interrupção. Se nesse próximo queue não for o instante do tempo para o envio do pacote, o token continua o percurso nos outros queues até encontrar o pacote que deve ser enviado. Se o percurso chegar até o último queue e verificar que nenhum pacote foi transmitido, o percurso do token continua na posição FIFO e transmite o pacote que estiver no queue.

Para cada queue, a taxa de transmissão pode variar dinamicamente dependendo das estimativas como mostra a equação (12). A variável **Ticks** representa o número de interrupções para obter a taxa estimada a cada pacote **P**. A medida desta taxa está representada em pacotes por segundo (pps).

$$Rate = P * Ticks \qquad (12)$$

A quantidade de pacotes em cada queue dependerá do número de Ticks estimado, isto oferece um controle do burst tal como se representa na equação (13), onde queue_tx_len é a quantidade de pacotes que a interface permite.

Burst=queue_tx_len / Ticks (13)

Este limite de pacotes permitirá que os queues garantam espaço de memória na interface de rede para todos os fluxos, sem armazenar pacotes de forma desnecessária, prevenindo-se a obstrução dos outros queues.

6.6. API de desenvolvimento

Com o propósito de estabelecer as comunicações entre a aplicação e os módulos do kernel, foi desenvolvida uma biblioteca com funções netlink com linguagem de programação C, denominada **libe2e**[5], cujos principais propósitos são: configurar, ativar e desativar o controle de fluxo. O método de compilação usa gcc com o parâmetro LIB "**–le2e"** referente à biblioteca de funções. O método de compilação pode ser visto a seguir:

gcc –o app-bin app-src.c **–le2e**

Um código fonte de um cliente UDP para ativar e desativar o controle de fluxo pode ser visto como exemplo no anexo-1. O código inicia com a configuração da estrutura sockaddr, abre um socket de netlink para estabelecer as comunicações com o módulo através da função e2e_nl_socket(), continua com a ativação do controle de fluxo com a função e2e_linkctrl_add(**sock_id, IP, IPPROTO, PORT**) passando os valores do identificador do socket, endereço IP, protocolo e porta. A função de desativação e2e_linkctrl_del(**sock_id, IP, IPPROTO, PORT**) pega os mesmos valores de ativação e apaga as informações do controle. Por último o socket de netlink é fechado com a função e2e_nl_close(**sock_id**). Em cada configuração de protocolo, podem ser aceitas definições do tipo IPPROTO_UDP, IPPROTO_TCP[6], etc.

---

[5] Localizada em **/usr/lib/libe2e.so**

[6] IPPROTO_ definidos em netinet/in.h e /etc/protocols

6.7. Comando e2e

Para aplicar o mecanismo e2e sem alterar um código fonte com a adição de funções, foi desenvolvido um comando **e2e** para facilitar o processo de ativação e desativação do controle de fluxo, que se aplica da seguinte forma:

e2e –t *ip port*

O comando anterior indica que será realizada a configuração do controle do fluxo para todo enlace que use o protocolo de transporte TCP indicado com –t (-u para UDP) com um endereço ip e a porta especifica. Para desativação do controle de fluxo se executa o mesmo comando com os mesmos parâmetros de configuração especificando --d:

e2e –t *ip port* **--d**

7. TESTES

Para obter resultados com o mecanismo e2e, foram realizados testes usando aplicações cliente/servidor IPERF[7] [22], TFTP para TCP e UDP respectivamente. A configuração de rede do teste consiste de duas redes ethernet-100Mbps enlaçadas com uma largura de banda de 28kbps, como mostra a figura 18. Isto foi feito para gerar um gargalo que tanto produziria perdas de pacotes como geraria um fluxo passível de ser controlado usando a granularidade de interrupções disponíveis no kernel do Linux [44]. Na configuração da largura de banda, foi executado o comando de *traffic control* (tc) [37] restringindo o burst com 15 kbytes e a latência com 5 mili-segundos usando o scheduler *Token Bucket Filter (tbf) [40]*. O comando foi aplicado da seguinte maneira: *"tc qdisc add dev eth0 tbf burst 15000 rate 28kbit latency 5ms"*.

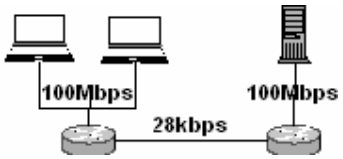

Figura 18. Diagrama de rede para testes

No servidor roda linux com o kernel 2.4.26, em cada sistema cliente roda Linux com o kernel 2.6.17 compilado com a variável de freqüência HZ_100[8] equivalente a um período de 1ms (mínimo período entre cada interrupção). Esta configuração geralmente é importante para adquirir resultados com mecanismos rate-based, que permitirá minimizar o intervalo na transmissão de pacotes dando granularidade à taxa atingível pelo protocolo. Isto permitirá ao mecanismo e2e temporizar granularmente as interrupções para o envio de cada pacote.

Os testes compreendem duas máquinas em um lado da rede que enviam um fluxo constante ao mesmo tempo de forma unidirecional através de uma largura de banda limitada durante uma hora. Uma das máquinas será necessária para gerar um tráfego TCP

[7] Iperf , programa usado para medir a largura de banda

[8] HZ definido no kernel/kconfig.hz

e o outro será considerado para adquirir as medidas. Inicialmente serão analisados fluxos TCP cujo resultado será comparado com os testes quando é aplicado o mecanismo e2e.

Esta configuração dará condições de tráfego estáticas que permitirão avaliar o desempenho que o mecanismo e2e pode oferecer. Os testes devem determinar o comportamento do protocolo de transporte TCP e UDP para um link congestionado.

Para adquirir as medidas foi alterado o código do Token Bucket Filter (TBF) para imprimir cada cinco segundos um contador de pacotes vindos do endereço IP que utiliza o mecanismo traffic shaping, dado indispensável para calcular o número de pacotes descartados. Além disso, o código FIFO foi alterado para imprimir igualmente o contador de pacotes na máquina que gera tráfego para comparar o desempenho do TCP acompanhado de um tráfego baseado em taxa.

7.1. Testes com TCP

Este teste permitirá observar o comportamento típico do TCP quando não utiliza o mecanismo e2e, produzido por dois fluxos IPERF em diferentes máquinas, dará uma visão comparativa com os testes posteriores onde será usado o mecanismo e2e. Um gráfico típico de um fluxo TCP pode ser visto na figura 19.

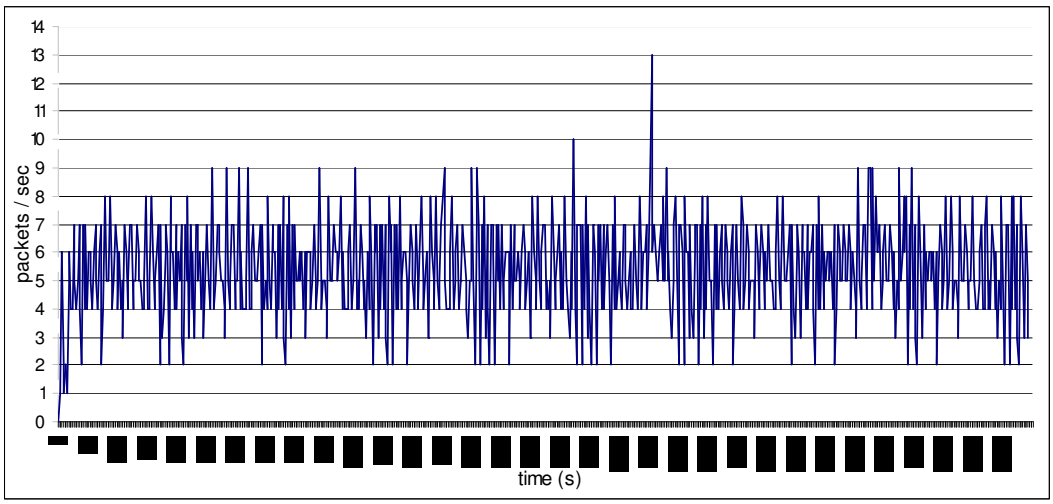

Figura 19. TCP sem o mecanismo e2e

A tabela 1 apresenta os resultados de cinco testes obtendo-se como resultado médio uma porcentagem de perda de 9.1%, um aproveitamento da largura de banda de um 11,48 kbps, 4,938MB capturados com aproximadamente 3946 pacotes recebidos.

	Time (s)	Packets	Size (MBytes)	BW (kbps)	Received (pkt)	% lost
Test 1	3606,5	3845	4,8	11,2	3519	8,4785436
Test 2	3615,4	4085	5,01	11,6	3719	8,9596083
Test 3	3609,2	3820	5,11	11,9	3475	9,0314136
Test 4	3615,4	4033	4,98	11,6	3634	9,8933796
Test 5	3605,5	3877	4,79	11,1	3494	9,8787722
Media	*3610,4*	*3945,75*	*4,938*	*11,48*	*3586,75*	*9,09074*

Tabela 1. Testes com TCP

A figura 20 mostra o cúmulo de pacotes desde o inicio da transmissão até o fim da transmissão. O gráfico representa o comportamento linear, incrementando o número de pacotes transmitidos a cada segundo. Este gráfico servirá como ponto de comparação com os testes seguintes usando o mecanismo e2e e permitirá determinar a estabilidade da transmissão de pacotes que este pode oferecer.

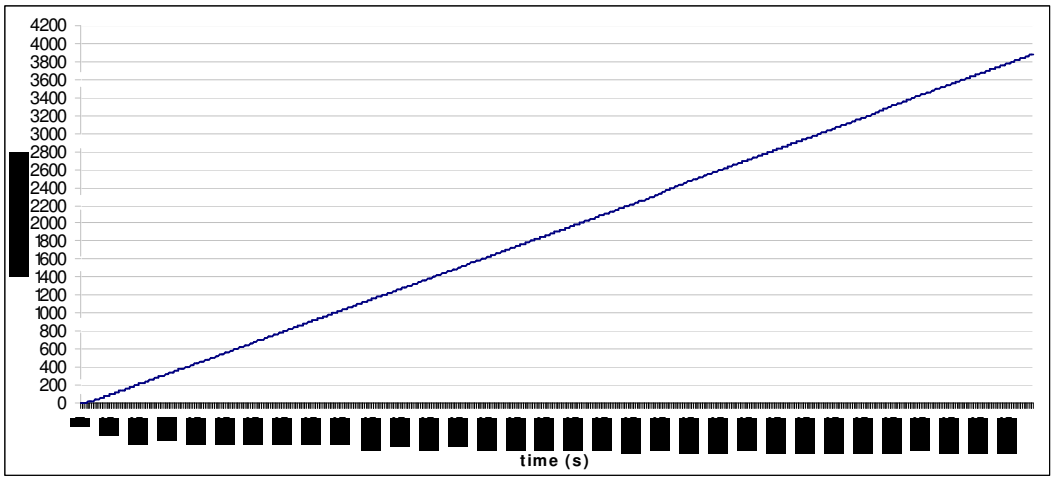

Figura 20. Gráfico cumulativo de pacotes do TCP

7.2. Teste e2e com UDP

Nos resultados adquiridos com o UDP utilizando o mecanismo e2e, foi necessário modificar o código fonte do TFTP para enviar pacotes independentemente da chegada do ack. Esta modificação no código fonte foi feita porque os pacotes eram enviados de forma

análoga ao mecanismo *stop and wait* do TCP o qual não gerava congestionamento e não atingia a largura de banda disponível.

Neste teste será aplicado o mecanismo e2e para realizar o controle de congestionamento ao protocolo UDP, este deverá minimizar as perdas de pacotes que normalmente UDP apresenta sem nenhum controle. A medição será feita em cada fim para capturar informações de pacotes enviados pelo cliente TFTP, pacotes recebidos pelo servidor e assim determinar o número de pacotes perdidos.

7.2.1. Resultados

As medidas capturadas foram feitas cinco vezes dando como resultado um valor médio de 4544 pacotes recebidos pelo servidor TFTP de 5349 pacotes enviados cada vez. Como resultado, o mecanismo garantiu que 75% dos pacotes UDP chegassem ao servidor como se mostra na tabela 2. O tempo médio consumido foi 1459 segundos, alcançando 15 kbps de largura de banda, transmitindo 63.75% mais pacotes do que o cross traffic TCP enviu no mesmo intervalo de tempo. O UDP é um protocolo não confiável significando que o arquivo nunca chegou completo ao servidor, porém, o mecanismo poderia dar certa efetividade às aplicações UDP que não precisem de confiabilidade.

	Time (s)	**TCP (pkt)**	**UDP (pkt)**	**Size (MB)**	**BW (kbps)**	**Rcv (pkt)**	**% Lost**
Test 1	1480,8	1930	5348	2736340	14,783	4564	14,66
Test 2	1444,8	1920	5348	2736340	15,152	4518	15,52
Test 3	1465,6	1974	5348	2736340	14,936	4491	16,02
Test 4	1442,6	1909	5348	2736340	15,174	4571	14,53
Test 5	1463,2	1959	5348	2736340	14,961	4577	14,42
Media	*1459,4*	*1938,4*	*5348*	*2736340*	*15,0012*	*4544,2*	*15,03*

Tabela 2. Resultados com UDP

A figura 21 mostra o gráfico típico de um fluxo UDP utilizando o mecanismo e2e. Este gráfico apresenta a variação de pacotes transmitidos dependendo do delay medido, o qual varia de acordo com o congestionamento gerado pelo fluxo cross traffic TCP. A variação de pacotes transmitidos depende das medidas do delay. Um aumento do delay é gerado quando ocorre uma perda pelo cross traffic TCP, eventualmente a taxa de transmissão é diminuída. Quando o delay diminui pode ser interpretado o inicio de um slow start do

cross traffic TCP, consequentemente a variação da taxa é incrementada. Para poucas variações do delay pode ser interpretado como um congestion avoidance freqüente.

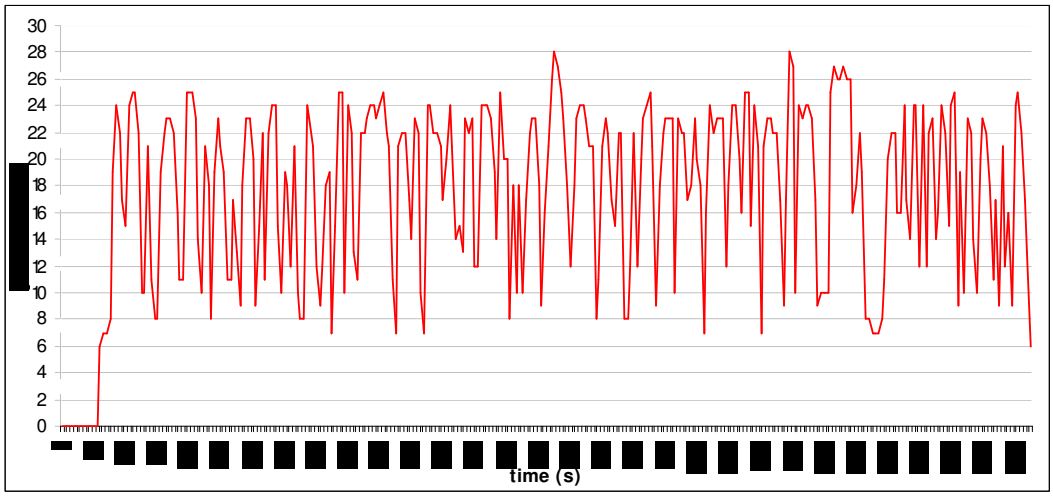

Figura 21. Fluxo UDP-e2e comparado com os pacotes descartados

A Figura 22 mostra o fluxo UDP (linha vermelha) utilizando o mecanismo e2e junto com o fluxo cross traffic TCP (linha preta).

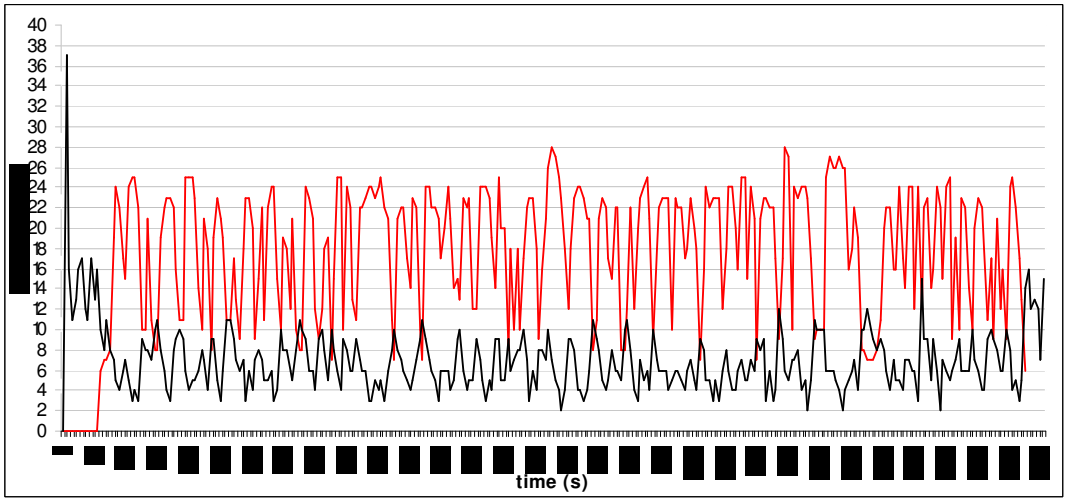

Figura 22. Fluxo UDP-e2e comparado com um fluxo cross traffic TCP

A figura 23 mostra o gráfico dos pacotes por segundo, recebidos pelo servidor. O gráfico dos pacotes recebidos pode ser comparado com o gráfico de pacotes transmitidos os quais tem muita semelhança, indicando que a variação da taxa estimada permitiu atingir a largura de banda disponível.

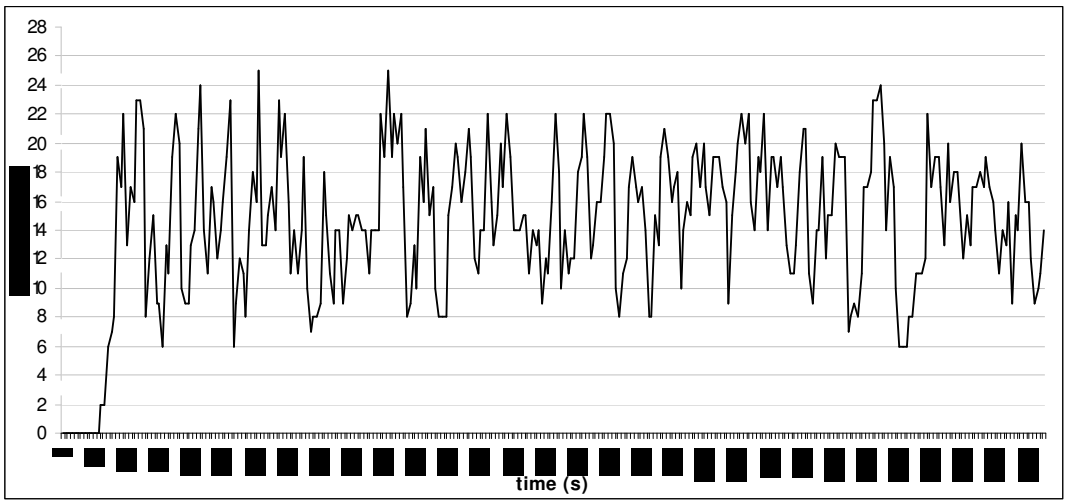

Figura 23. Pacotes recebidos do fluxo UDP utilizando o mecanismo e2e

A figura 24 mostra o cúmulo de pacotes do fluxo UDP (linha vermelha), fluxo cross traffic TCP (linha preta) e o fluxo de referência comparativa TCP da seção 7.1 (linha azul). No gráfico se pode observar que o UDP envia maior número de pacotes do que o cross traffic TCP. A performance do cross traffic TCP não resultou influenciando pelo fluxo UDP, que em comparação com o fluxo de referência TCP (seção 7.1), transmitiu um maior número de pacotes nesse intervalo de tempo.

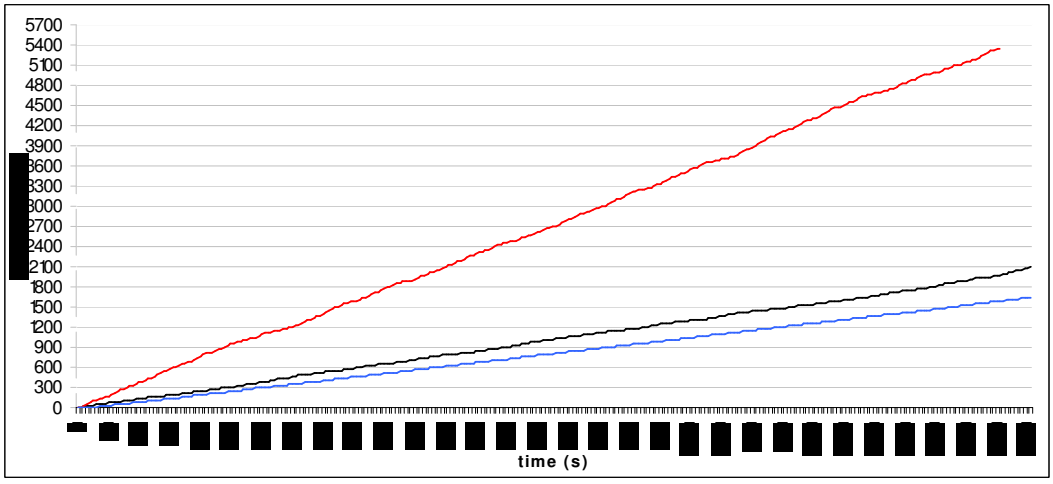

Figura 24. Gráfico cumulativo de pacotes UDP, cross traffic TCP.

Um fluxo UDP controlado pelo mecanismo e2e necessita de um procedimento que aumente gradativamente o número de pacotes transmitidos tipo *slow start* do TCP para

diminuir o número de perdas e garantir que o fluxo UDP não congestione ainda mais a rede.

7.3. Testes e2e com TCP

Nesta experiência se tentará estudar o comportamento do protocolo TCP quando estiver utilizando o mecanismo e2e. Esta mistura deve permitir que o TCP diminua o número de perdas aumentando o desempenho. Para observar o comportamento do controle de congestionamento do TCP foi alterado o código fonte do TCP-BIC[9] [23] (Congestion Agent default no kernel 2.6.17[10]) para imprimir os eventos ocasionados ao utilizar o mecanismo e2e.

7.3.1. Resultados

As medidas foram capturadas cinco vezes como se mostra na tabela 2, obtendo-se uma porcentagem media de 5.02% na perda de pacotes, incrementando o número de bytes recebidos em 4.17% e aproveitando a largura de banda em 4.36%. Um gráfico típico pode ser visto na figura 25.

	Time (s)	Packets	Size (MBytes)	BW (kbps)	Recived (pkt)	% Lost
Test 1	3605	3874	5,07	11,8	3677	5,09
Test 2	3609,1	4024	5,27	12,3	3825	4,95
Test 3	3608,5	3463	4,51	10,5	3270	5,57
Test 4	3603,4	4227	5,55	12,9	4028	4,71
Test 5	3610,1	3885	5,07	11,8	3676	5,38
Test 6	3603,1	3656	4,76	11,1	3458	5,46
Media	*3606,533333*	*3894,6*	*5,038333333*	*11,73333333*	*3695,2*	*5,12*

Tabela 3. Testes com TCP utilizando o mecanismo e2e

[9] Código fonte localizado no net/ipv4/tcp_bic.c

[10] /proc/sys/net/tcp_congestion_control

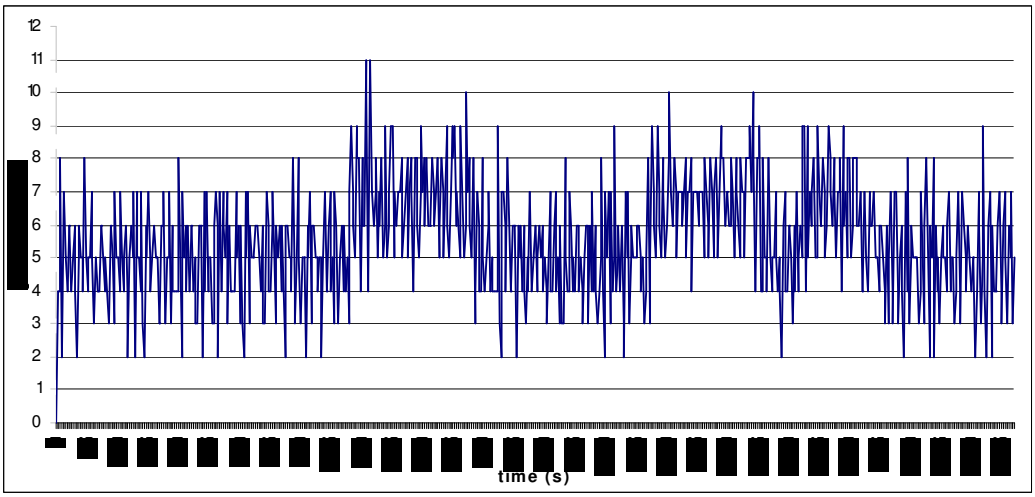

Figura 25. Fluxo TCP utilizando o mecanismo e2e

Nos testes feitos com o mecanismo e2e foi necessário limitar o valor de latência máxima para coexistir com o tráfego TCP quando as variações do delay eram muito altas e freqüentes, configurado para 70 ms. Se este valor não fosse limitado, começaria a entrar em conflito com o controle de congestionamento do TCP gerando uma série eventos de *congestion avoidance, slow start, slow start-restart* e por último um evento de TCP_CA_Loss (TCP Congestion Agent Loss) que indica que o *congestion agent* não foi suficientemente capaz para subsistir no enlace congestionado. Finalmente a transmissão é totalmente perdida e a aplicação deve ser reiniciada para uma nova tentativa.

A figura 26 mostra o cúmulo de pacotes do Fluxo TCP usando o mecanismo e2e (linha vermelha) e o cúmulo de pacotes típico do TCP de referência comparativa da seção 7.1. Observa-se no gráfico que o cúmulo de pacotes transmitidos pelo TCP usando o mecanismo e2e tem um incremento menor de pacotes transmitidos, mas ao final do intervalo de tempo, atinge o mesmo número de pacotes que comparado com o cúmulo de referência TCP (seção 7.1), indica menor congestionamento e um desempenho similar.

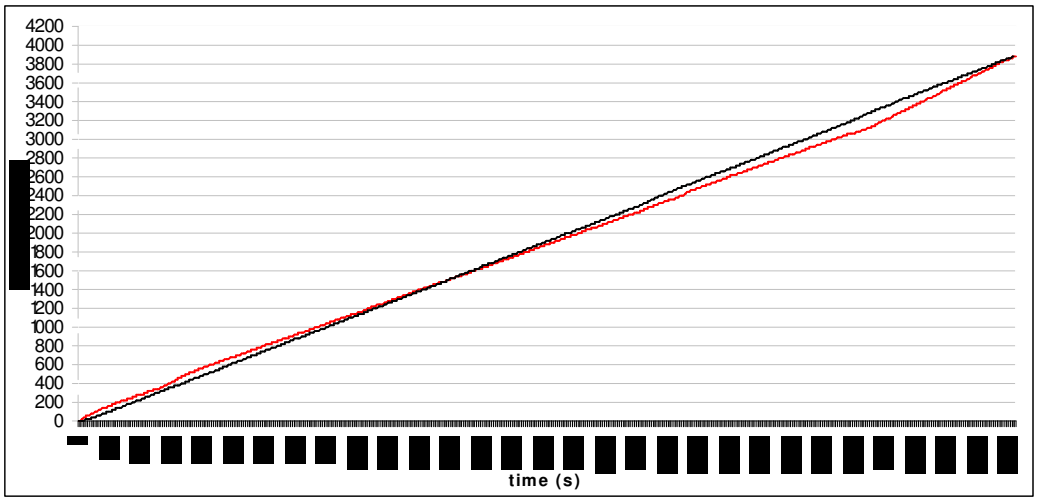

Figura 26. Gráfico cumulativo de pacotes do TCP usando o mecanismo e2e

7.4. Teste de um enlace congestionado com fluxos TCP usando e2e

Neste teste se tentará demonstrar que uma rede congestionada com fluxos TCP utilizando o mecanismo e2e deve dar como resultado um maior aproveitamento da largura de banda e menor número de pacotes perdidos, aumentando o número de pacotes recebidos. A idéia central deste teste é emitir dois fluxos TCP em duas máquinas configuradas para usar o mecanismo e2e.

7.4.1. Resultados

As medidas foram adquiridas seis vezes como é mostrado na tabela 4 dando como resultado um valor de 4,61% de perdas aumentando em 9,61% o número de bytes recebidos e incrementando o uso da largura de banda em 9,54%. O gráfico típico pode ser visto na figura 27.

	Time (s)	Packets	Size (MBytes)	BW (kbps)	Recived (pkt)	% Lost
Test 1	3606,8	4213	5,53	12,9	4016	4,6760028
Test 2	3609,4	4007	5,25	12,2	3827	4,4921388
Test 3	3611,8	4171	5,48	12,7	3975	4,6991129
Test 4	3608	4096	5,39	12,5	3909	4,5654297
Media	*3609*	*4121,75*	*5,4125*	*12,575*	*3931,75*	*4,608171*

Tabela 4. Testes com dois fluxos tcp usando e2e

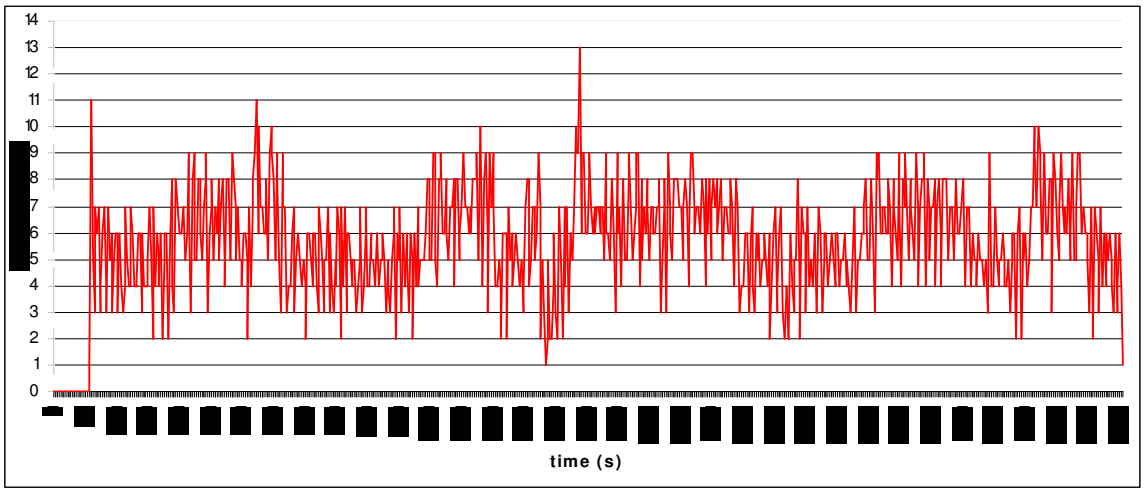

Figura 27. TCP-e2e em uma rede congestionado com fluxos e2e

É possível observar que neste período de tempo foi enviado um número de pacotes maior. A figura 28 apresenta o fluxo TCP usando o mecanismo e2e da outra máquina implicada neste teste. Neste gráfico, inicia-se a transmissão com um número alto de pacotes quando o enlace não esta congestionado, continuando com uma redução do número de pacotes quando entra outro fluxo e finalmente se incrementa quando o enlace novamente esta livre de congestionamento.

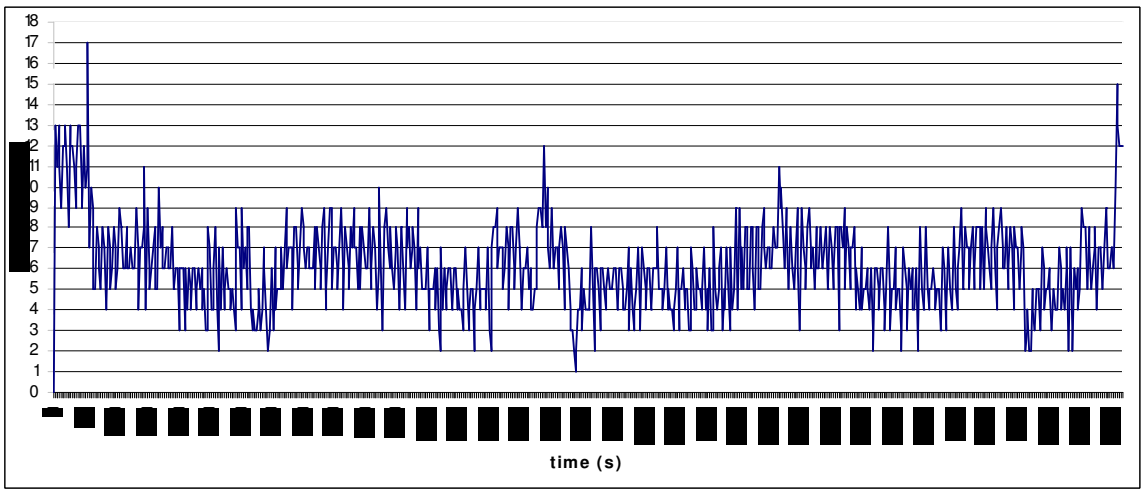

Figura 28. Fluxo e2e no gerador de tráfego

A figura 29 apresenta os dois fluxos e2e cross traffic, onde o fluxo representado com linha vermelha ilustra a transmissão de pacotes para as análises e comparações.

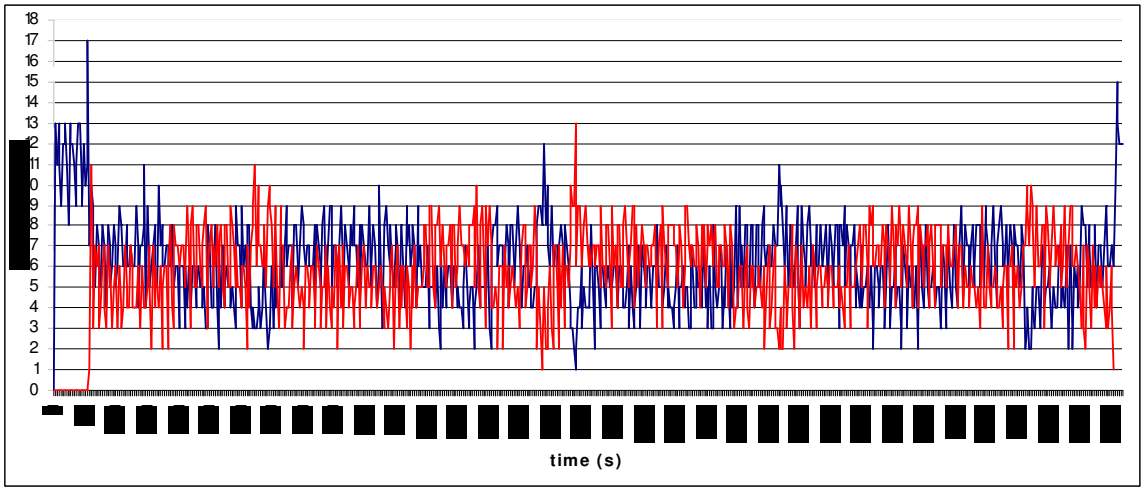

Figura 29. Dois fluxos e2e compartilhando banda

Na figura 30 mostra o cúmulo de pacotes dos fluxos TCP (linha vermelha, cross traffic em linnha preta), junto com o cúmulo de pacotes típico do TCP de referência (linha azul) da seção 7.1. É possível verificar que o número de pacotes transmitidos é maior que em uma rede congestionada com fluxos TCP, isto acontece porque o mecanismo identifica a largura de banda disponível sem necessidade de gerar muitas perdas para este propósito, transmitindo os pacotes de forma espaçada de acordo com o número de interrupções que assegura a taxa estimada.

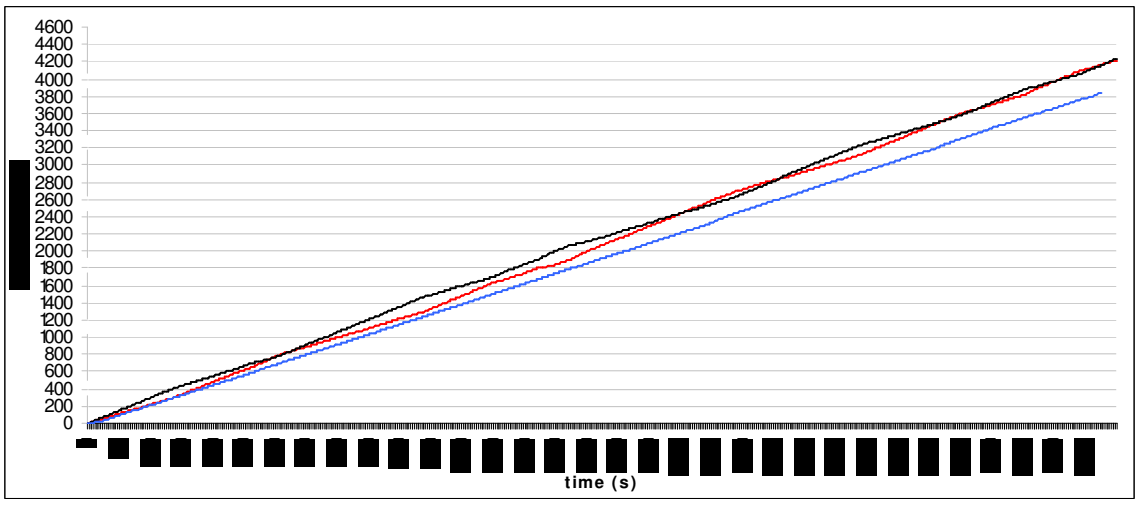

Figura 30. Gráfico cumulativo do TCP usando o mecanismo e2e

7.5. Comparação dos resultados com TCP

A tabela 4 apresenta os valores médios de todos os testes realizados anteriormente com TCP. O primeiro teste tcp-tcp consiste em dois fluxos TCP. O teste tcp-e2e significa um fluxo TCP cross traffic TCP utilizando o mecanismo e2e. O e2e-e2e significa dois fluxos TCP usando o mecanismo e2e.

	Time (s)	Packets	Size (MBytes)	BW (kbps)	Received (pkt)	% Lost
tcp-tcp	3610,4	3945,75	4,938	11,48	3586,75	9,09
tcp-e2e	3606,14	4002,5	5,144	11,98	3801,5	5,02
e2e-e2e	3609	4121,75	5,4125	12,575	3931,75	4,61

Tabela 5. Resultados comparativos com TCP

Em comparação com dois fluxos TCP sem utilizar o mecanismo e2e, obteve-se como resultado:

1. **tcp-e2e** teve **4,06%** menos perdas de pacotes aumentando em **4,17%** o número de bytes recebidos e incrementando o uso da largura de banda em **4,36%**.
2. **Dois fluxos tcp usando e2e** tiveram **4,48%** menos perdas de pacotes, aumentando em **9,61%** o número de bytes recebidos e incrementando o uso da largura de banda em **9,54%.**

8. TRABALHOS FUTUROS

Para trabalhos futuros o mecanismo e2e pode ser instalado na camada de transporte, programando todas as técnicas com funções de sockets, descartando o uso de netfilter e netlink. Isto poderia dar origem a novos protocolos de transporte para fins específicos. Por outro lado, as funções poderiam ser adicionadas ao protocolo UDP para dar controle de congestionamento.

O mecanismo e2e poderá ser usado para realizar tarefas de traffic shapping de forma dinâmica entre roteadores, podendo variar a taxa de transmissão impedindo que os pacotes continuem o percurso, pacotes sejam descartados em saltos mais distantes e impedindo eventualmente o congestionamento desnecessário da rede.

O mecanismo e2e poderá ser evoluído para um meio de criar QoS, dando controle aos parâmetros envolvidos tal como burst e latência, de forma dinâmica dependendo da largura de banda alcançável.

Como mecanismo de envio de pacotes por múltiplas interfaces de rede, é necessário desenvolver um algoritmo que reordene os pacotes quando são recebidos no receptor para passá-los posteriormente à camada de transporte.

Os cálculos do mecanismo e2e foram baseados em medidas do delay, não sendo o único método para estimar a taxa, um trabalho futuro seria estimar a taxa baseado em cálculos do erro do delay, analisando os jitters positivos e negativos.

9. CONCLUSÕES

O mecanismo e2e é uma ferramenta desenvolvida de forma experimental, projetada para variar a taxa de transmissão de um protocolo de transporte em uma determinada configuração do enlace dada pela aplicação, enviar constantemente pacotes de prova e medir a largura de banda disponível.

O mecanismo e2e oferece uma alternativa de controle de fluxo com uma técnica de "queue discipline" utilizando trens de pares de pacotes para calcular o delay e estimar a taxa que é variada em cada queue associado com um endereço IP destino.

O objetivo principal desta ferramenta foi obter resultados de variação da taxa de transmissão ao nível de kernel nas camadas mais baixas do modelo de referência OSI utilizando o UDP e o TCP como protocolos de transporte. No caso do TCP misturado com o controle de fluxo, experimentou-se a reação de um controle de congestionamento convencional quando se variava à taxa e assim determinar se era possível melhorar o throughput com um agente externo sem alterações do código fonte do TCP. No caso do UDP, oferecer uma alternativa de controle de fluxo minimizando as perdas de pacotes que são produzidas normalmente.

Os resultados adquiridos nos testes demonstram que enquanto o TCP aumenta a quantidade de pacotes no enlace, o mecanismo e2e captura valores do delay altos, minimizando a taxa de transmissão. Se o delay for muito alto e este valor for muito freqüente, significa que o TCP congestionou a rede atingindo a máxima largura de banda fazendo com que o mecanismo e2e transmita cada vez menos, mas quando o TCP está em fase de slow start o delay pode ser muito baixo e fazer com que o mecanismo transmita mais. Enquanto à fase de Congestion Avoidance, a transmissão de pacotes pode ser mais constante quando o delay varia com menor freqüência.

O mecanismo e2e pode ser usado para realizar um controle de congestionamento para o protocolo UDP necessitando de um mecanismo de slow start para minimizar o número de

pacotes perdidos e não desestabilizar a rede com um tráfego muito alto adequado às especificações de TCP-Friendly [38].

O mecanismo convencional do TCP é congestionar a rede para assegurar uma largura de banda proporcional incrementando gradativamente o número de pacotes e gerando perdas. Porém um mecanismo baseado em taxa em um enlace congestionado por TCP, deve congestionar igualmente o enlace para adquirir uma largura de banda equivalente. Para que o mecanismo e2e coexista com uma rede congestionada com TCP, o valor de taxa deve ser variado para que atue de um jeito que possa persistir e oferecer vantagens quando comparado com o TCP.

Uma rede congestionada com fluxos TCP utilizando o mecanismo e2e poderia aperfeiçoar o desempenho da rede e diminuir o número de perdas de pacotes. Mas esta seria uma rede utópica porque ao entrar um fluxo TCP que não use o mecanismo e2e, o esquema se desestabilizaria.

Ao enviar os pacotes por múltiplas interfaces ativas se observou que os pacotes sempre chegavam desordenadamente, o que causava instabilidade nas comunicações de dados quando as seqüências não eram consecutivas.

A principal contribuição deste trabalho foi o desenvolvimento de um mecanismo fim a fim de controle e modulação de tráfego com a finalidade de controlar o burst e a taxa de transmissão aplicada a um protocolo de transporte que pode ser tanto TCP como UDP, capturando informações de largura de banda alcançável em um determinado trajeto. Este estudo dará origem a outros métodos de transmissão de pacotes como um protocolo de transporte ou uma nova forma de QoS.

10. REFERÊNCIAS BIBLIOGRÁFICAS

[1] Prasad, R.S., Murray M., "Bandwidth estimation: metrics, measurement techniques, and tools". IEEE Network Magazine. 2003.

[2] Hu, N., Steenkiste, "Evaluation and Characterization of Available Bandwidth Probing Techniques." **IEEE JSAC** *Special Issue in Internet and WWW Measurement, Mapping, and Modeling*, Vol. 21(6), Aug. 2003.

[3] Strauss, J., Katabi, D., Kaashoek, F., "A Measurement Study of Available Bandwidth Estimation Tools". ACM SIGCOMM Internet Measurement Workshop, 2003.

[4] Vinay J. Ribeiro, Rudolf H. Riedi, "pathChirp: Efficient Available Bandwidth Estimation for Network Paths", Passive and Active Measurement Workshop, 2003.

[5] Banerjee, S., Agrawala, A.K., "Estimating Available Capacity of a Network Connection". In: IEEE International Conference on Networks, Singapore, 2000.

[6] Carter, R.L., Crovella, M.E, "Dynamic Server Selection Using Bandwidth Probing in Wide-Area Networks", Technical Report TR-96-007, Boston University, Computer Science Department, 1996.

[7] Jin, G., Yang, G., "Network Characterization Service (NCS)", In: 10th IEEE Symposium on High Performance Distributed Computing, 2001.

[8] Ningning Hu, Peter Steenkiste., "Evaluation and Characterization of Available Bandwidth Probing Techniques", IEEE JSAC Special Issue in Internet and WWW Measurement, Mapping, and Modeling, 2003.

[9] Constantinos Dovrolis, Paramkeswaran Ramanathan, "Packet Dispersion Techniques and a Capacity Estimation Methodology", Part of this paper has previously appeared in an Infocom, 2001.

[10] M. Mathis, “A Framework for Defining Empirical Bulk Transfer Capacity Metrics”, IETF RFC-3148, Julho 2001.

[11] Soares, Lemos e Colcher. “Redes de Computadores das LANs, MANs e WANs às Redes ATM”,Editora Campus, 2a. Edição, 1997.

[12] Jon Postel, “Transmission Control Protocol”, IETF, RFC 793, Setempbro de 1981.

[13] Bob Sullivan, “Remembering the Net Crash of ’88”, MSNBC, 1998.

[14] M. Allman, “TCP Congestion Control”, IETF, RFC-2581, 1999.

[15] Thomas Graf, Greg Maxwell, “Linux Advanced Routing & Traffic Control”, HOWTO, 2006.

[16] Netfilter: http://www.netfilter.org/, 2006.

[17] Vincent Guffens. “Path of a packet in the Linux kernel”, Kansas University, Course Summer 2005.

[18] P. Gortmaker. “Linux ethernet-howto”, HOWTO, 2000.

[19] Vikram Visweswaraiah, John Heidemann, “Rate Based Pacing for TCP”, preliminary DRAFT, 1997.

[20] Eddie Kohler, Mark Handley, “Designing DCCP: Congestion Control Without Reliability”, *ACM SIGCOMM, 2006*.

[21] Luiz C. Schara Magalhães. “A Transport Layer Approach to Host Mobility”, Tese de doutorado. 2005

[22] A Tirumala, F Qin, J Dugan, “Measuring end-to-end bandwidth with Iperf using Web100”, PAM, 2005.

[23] Yee-Ting Li, Douglas Leith, Robert N. Shorten, “Experimental Evaluation of TCP Protocols for High-Speed Networks”, Terena Conference, 2005.

[24] Fabien Viger, “Active Probing with ICMP Packets”, Internship in the Department of Electrical and Electronic Engineering, University of Melbourne, 2003.

[25] J. Postel, “Internet Control Message Protocol”, IETF RFC-792, Setembro de 1981.

[26] Ryousei Takano, “Realtime Burstiness Measurement”, http://www.hpcc.jp/pfldnet2006/paper/s5_03.pdf, 2006.

[27] Saravanan Radhakrishnan, “Linux - Advanced Networking Overview, Version 1”, http://qos.ittc.ku.edu/howto/index.html, 1999.

[28] Lu-chuan Kung, “A Walk-Through of Netfilter”, Lectures, http://lion.cs.uiuc.edu/courses/cs397hou/lectures/Inside_Netfilter.pdf, 2004.

[29] Jonathan Corbet, “Porting drivers to the 2.5 kernel”, Linux Symposium, 2003

[30] Free Software Foundation, Inc., “The Linux Kernel API”, EBook, 2006

[31] Jennifer Hou, “Network Devices”, Lectures, Departament of the computer science, University o Illinois, 2005.

[32] Jennifer Hou, “Data Link Layer”, Lectures, Departament of the computer science, University o Illinois, 2005.

[33] Experimental Computer Systems Lab, “Linux Networking Kernel”, Technical Documentation, University of New York, 2003

[34] Michael Smith, Steve Bishop, “Flow Control in the Linux Network Stack”, TACS, ESOP, 2002.

[35] Constantinos Dovrolis, Hao Jiang, “Source-Level IP Packet Bursts: Causes and Effects”, Internet Measurement Conference, 2003

[36] Ningning Hu, Peter Steenkiste, “Estimating Available Bandwidth Using Packet Pair Probing”, IEEE JSAC Special Issue in Internet and WWW Measurement, Mapping, and Modeling, 2003.

[37] Bert Hubert, “Linux Advanced Routing and Traffic Control”, HOWTO, 2006.

[38] Dorgham Sisalem, Henning Schulzrinne. “The Loss-Delay Based Adjustment Algorithm: A TCP-Friendly Adaptation Scheme”, NOSSDAV, 1998.

[39] Amit Aggarwal, Stefan Savage, Thomas Anderson, “Understanding the Performance of TCP Pacing”, *IEEE INFOCOM 2000 Conference on Computer Communications*, 2000.

[40] P.F. Chimento, “Standard Token Bucket Terminology”, Technical documentation, 2000.

[41] Jitendra Padhye, Jim Kurose, “A Model Based TCP-Friendly Rate Control Protocol”, NOSSDAV, 1999.

[42] Sally Floyd, Kevin Fall, “Promoting the Use End-to-End Congestion Control in the Internet”, IEEE ACM Transactions on Networking,1999.

[43] Kevin Lai, Mary Baker, “Measuring Link Bandwidths Using Deterministic Model of Packet Delay”, SIGCOMM, 2000.

[44] Vern Paxson, “End-to-End Internet Packet Dynamics”, IEEE/ACM Transactions on Networking, 1997.

[45] Bob Melander, Mats Bjokman, “Regression-Based Available Bandwidth Measurements”, IEEE Network, 2003.

[46] MAGALHAES, LUIZ; KRAVETS, Robin. MMTP: Multimedia Multiplexing Transport Protocol. In: The First Workshop on Data Communications in Latin America and the Caribbean (SIGCOMM-LA 2001), 2001, San Jose, Costa Rica. Supplement to Computer Communication Review. New York: The Association for Computer Machinery, 2001. v. 31. p. 220-243.

11. ANEXOS

Anexo 1. Código de mostra para ativação do controle do fluxo.

```
#include <asm/types.h>
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <string.h>

#define BUFF_SIZE 512
#define PORT 5432

int main(int argc, char *argv[]){
        struct hostent *hp;
        struct sockaddr_in sin;
        int e2e_sk, ret, i=0;
        char host[17];
        char buffer[BUFF_SIZE];
        int socket_id, len;
        if(argc == 2) strncpy(host, argv[1], 15);
        else{
                fprintf(stderr, "usage: simplex-talk host\n");
                return -1;
        }
        hp = gethostbyname( (char *) host);

        if(!hp){
                fprintf(stderr, "simplex-talk: unknown host %s\n", host);
                return -1;
        }

        bzero((char *)&sin, sizeof(sin));
        sin.sin_family = AF_INET;
        bcopy(hp->h_addr, (char *)&sin.sin_addr, hp->h_length);
        sin.sin_port = htons(PORT);

        e2e_sk = e2e_nl_socket();
        if(rio_sock_id == -1)
                fprintf(stderr, "can't link to %s\n", host);
        ret = e2e_linkctrl_add(e2e_sk, sin.sin_addr,
                                IPPROTO_UDP, sin.sin_port);
        if(ret == -1) fprintf(stderr, "RIO: add to %s failed\n", host);
        while(i < 255){
                if((socket_id = socket(PF_INET, SOCK_DGRAM, 0)) < 0){
                        perror("simplex-talk: socket");
```

```
                return -1;
        }
        if( ( connect(socket_id, (struct sockaddr *)&sin, sizeof(sin)) ) < 0 ){
                perror("simplex-talk: socket");
                return -1;
        }
        buffer[BUFF_SIZE - 1] = '\0';
        sprintf(buffer, "%d", i);

        send(socket_id, buffer, len, 0);
        close(socket_id);
        i++;
    }

    ret = e2e_linkctrl_del(e2e_sk, sin.sin_addr,
                                IPPROTO_UDP, sin.sin_port);
    e2e_nl_close(e2e_sk);
}
```