UNIVERSIDADE FEDERAL FLUMINENSE
MESTRADO EM ENGENHARIA DE TELECOMUNICAÇÕES

NÉSTOR FELIPE MAYA QUINTERO

MECANISMO DO KERNEL PARA O CONTROLE DE TRÁFEGO E VARIAÇÃO DA TAXA FIM A FIM - E2E

NITERÓI
2007

NÉSTOR FELIPE MAYA QUINTERO

MECANISMO DO KERNEL PARA O CONTROLE DE TRÁFEGO E VARIAÇÃO DA TAXA FIM A FIM - 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, PH D

Niterói

2007

NÉSTOR FELIPE MAYA QUINTERO

MECANISMO DO KERNEL PARA O CONTROLE DE TRÁFEGO E VARIAÇÃO DA TAXA FIM A FIM - 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^{o} Luiz Cláudio Schara Magalhães – Ph D, Orientador
Universidade Federal Fluminense

Prof^{o} Débora Christina Muchaluat Saade, Ph D.
Universidade Federal Fluminense

Prof^{o}. Noemi de la Roque Rodriguez, Ph D
Pontifícia Universidade Católica do Rio de Janeiro

Niterói
2007

AGRADECIMENTOS

Em especial ao Programa de Pós-graduação do Departamento de Engenharia de Telecomunicações da Universidade Federal Fluminense pela ajuda, orientação e o ensino recebido.

Ao professor Luiz Cláudio Schara Magalhães, pela orientação, interesse e ajuda dispensada para a elaboração desta dissertação.

Aos professores e funcionários do Programa de Mestrado de Engenharia Telecomunicações pelo suporte fornecido.

RESUMO

A maior parte dos mecanismos fim a fim, desenvolvidos para dar forma ao tráfego nas comunicações de dados permite apenas 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, tal como o UDP ou o TCP, fazer uso da ferramenta de controle do fluxo, com a finalidade de variar a taxa de transmissão atingindo a largura de banda disponível estimada. O uso desta ferramenta permite avaliar qual seria o comportamento dos protocolos de transporte se estes fossem baseados em taxa.

Palavras chave: Telecomunicações; Mecanismo e2e; Network Kernel Programming; Controle de fluxo; Protocolos de transporte; Taxa de transmissão; Rate based.

ABSTRACT

The majority of end-to-end traffic shaping mechanism developed for data communications 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 e2e mechanism. It allows any flow, such as UDP or TCP, to be rate based controlled depending of bandwidth estimated. It also helps investigating what would be the transport protocol behavior if it were rate-based.

Keywords: Telecommunications; Mechanism e2e; Flow Control; Transport protocols; Sending rates, Network Kernel Programming; rate based.

ÍNDICE

LISTA DE FIGURAS

LISTA DE TABELAS

1. INTRODUÇÃO

Nos primórdios da Internet, notou-se um problema que era a transmissão excessiva de pacotes causada pela não existência de controle de congestionamento. Quando um caminho ficava sobrecarregado, e pacotes eram perdidos por falta de espaço de armazenamento temporário nos roteadores, o comportamento padrão era reenviar o pacote perdido imediatamente, agravando o congestionamento da rede [41]. Ao longo do tempo, houve incidentes onde grandes segmentos da rede eram incapacitados por excesso de tráfego. Apesar de o TCP ser o maior culpado [34] nestes incidentes por causa das retransmissões, o UDP também pode causar problemas, mesmo não oferecendo confiabilidade de transporte (isto é, retransmissões de pacotes perdidos). O problema é que aplicações que usam UDP podem enviar um grande número de pacotes em pouco tempo e não diminuem sua taxa de transmissão quando a rede não consegue suportar a taxa sendo usada (indicado pela perda de pacotes do fluxo).

É importante compreender que a estratégia do TCP é controlar o congestionamento quando este ocorre, em vez de evitar a ocorrência do mesmo [34]. Para fixar o seu ponto de operação, o TCP aumenta a taxa de transmissão até haver perda de pacotes para, desta forma, encontrar a largura de banda disponível. Também, os algoritmos do TCP que o tornam um protocolo de transporte confiável são complexos e tem interações que são difíceis de serem analisadas. Da mesma forma, apesar do UDP não ter que causar congestionamento (já que ele não mede a banda disponível e não implementa um mecanismo de controle de congestionamento), o aumento do tráfego do UDP para serviços novos pode causar a instabilidade da rede [21].

A transmissão de dados em redes onde a largura de banda é alocada dinamicamente, tal como Internet, pode resultar em um comportamento instável quando existe um tráfego significativo (isto é, próximo da capacidade dos enlaces). A prevalência do tráfego em rajadas causa o transbordamento das filas dos roteadores e subseqüentemente a perda de pacotes. No entanto, se fosse 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 projetar protocolos de roteamento mais eficientes e sólidos. Utilizando estatísticas de tráfego, os provedores poderiam localizar fontes de ineficiência e planejar melhores capacidades.

Uma alternativa para obter um melhor desempenho nas comunicações de dados é aplicar técnicas de controle de tráfego, mas as técnicas atuais realizam configurações estáticas, podendo apenas ser modificadas manualmente. Se as configurações de controle do tráfego fossem variadas dinamicamente, possivelmente o desempenho de um protocolo de transporte poderia aumentar. Isto significaria diminuir o número de perdas causadas nos roteadores por saturação das filas de pacotes, evitando transmissões desnecessárias na origem.

Outra técnica para aperfeiçoar as transmissões de pacotes é a utilização de múltiplas interfaces de rede. Com o aumento de dispositivos móveis, que potencialmente podem ter várias opções de conexão à rede, esta técnica é cada vez mais importante. Identificar o dispositivo de rede cujo tráfego do trajeto seja menor pode permitir balancear o tráfego da rede e maximizar o uso de dispositivos de rede existentes em um equipamento de computação.

1.1. Objetivos

Inicialmente, o objetivo desta dissertação era implementar no *kernel* do Linux os protocolos de transporte baseados em taxa MMTP e R-MTP relatados em [24] e [25] respectivamente. Estes protocolos tinham sido implementados como aplicações baseadas em UDP, usando os mecanismos de controle de congestionamento relatados nas referências citadas, ou como um agente para simulações em NS-2 [46], cujos resultados estão relatados em [44]. No entanto, a implementação dos protocolos como uma aplicação

não permite sua ampla utilização, pois seu uso requer modificações extensas nas aplicações alvo. Se os protocolos fossem módulos do kernel, qualquer aplicação poderia ser compilada usando o novo protocolo, bastando alterar o tipo de *socket* a ser criado, minimizando o número de alterações requeridas para o uso do novo protocolo.

O desafio e contribuição originais seriam, então, a implementação dos XMTPs (nome genérico dado para os protocolos R-MTP e MMTP). No entanto, apesar de interessante, esta empreitada se mostrou muito ambiciosa para uma dissertação de mestrado. Como resultado das pesquisas para criar os protocolos XMTP como módulos do *kernel*, foi desenvolvido um resultado intermediário, que utiliza as principais características dos XMTPs, que são as medições de banda disponível fim a fim e a adequação da taxa de envio a esta banda disponível sem, no entanto, desenvolver um protocolo completo. Foi criado um mecanismo que faz a moldagem de tráfego de qualquer fluxo, como por exemplo, fluxos TCP, os adequa à banda disponível fim a fim, e elimina as rajadas, enviando os pacotes periodicamente à taxa máxima medida. Este mecanismo de controle de tráfego e variação da taxa fim a fim é denominado e2e, e sua criação e validação são as contribuições deste trabalho.

O objetivo principal dos protocolos XMTP é a diminuição das perdas de pacotes, por meio de técnicas de provas ativas e variação da taxa de transmissão. Para dar suporte à mobilidade, estes protocolos são *multihomed*[1], e o cálculo da banda disponível também é utilizado para realizar o balanceamento de carga, distribuindo o tráfego por todas as interfaces de rede disponíveis simultaneamente [24]. Esta característica foi investigada, mas não está presente no e2e, que é apenas capaz de chavear entre interfaces, e não distribuir tráfego. Assim, é possível utilizar a melhor interface (definida como a que tenha maior banda disponível), mas não todas simultaneamente para um único fluxo. Isto é importante porque de forma geral o TCP não aceita o reordenamento de pacotes, que é muito freqüente quando se utilizam múltiplas interfaces, com diferentes tempos de trânsito (RTT – *round trip time*), simultaneamente.

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

Como foi mencionando, os protocolos XMTP foram desenvolvidos entre a camada de aplicação e a camada de transporte, usando o UDP (*User Datagram Protocol)* como protocolo base. O resultado deste trabalho, o módulo e2e, faz aplicação dos algoritmos desenvolvidos para os protocolos XMTP no nível de "*kernel*", sendo capaz de utilizar a melhor resolução de "*timers*" disponíveis neste nível, e as facilidades das camadas mais baixas da arquitetura de rede.

A contribuição deste trabalho é, com foi dito, a criação do módulo e2e, que permite dar forma ao tráfego de qualquer fluxo, de forma dinâmica, através da medida das características mutáveis da rede. Este módulo possibilita que qualquer protocolo tenha os benefícios das técnicas de controle de congestionamento desenvolvidas para os protocolos XMTP, sem que seja necessária a alteração das aplicações já existentes.

Finalmente, o módulo e2e também possibilita a análise do efeito nos protocolos de transporte quando estes são configurados para o funcionamento baseado em taxa.

1.2. Estrutura da dissertação

Nos próximos capítulos, faremos uma análise dos trabalhos relacionados, sobre as técnicas para medição de largura de banda, os mecanismos de controle de congestionamento, as técnicas de controle de tráfego e os protocolos de transporte baseados em taxa. Após os trabalhos relacionados, definiremos o mecanismo e2e. Ele pode ser utilizado para realizar o controle do fluxo fim a fim para qualquer protocolo de transporte. Uma vez definido o mecanismo e2e, serão especificadas as técnicas de desenvolvimento baseadas em UML (Unified Model Language), programação no nível "kernel" de Linux, mencionando o trajeto dos pacotes, modelagem do sistema, os diagramas de interação e diagramas de atividades.

Para comprovar a eficiência do mecanismo e2e, foram realizados vários testes, usando aplicações cliente-servidor. Estes testes foram realizados aplicando o TCP e o UDP como protocolos de transporte em um trajeto congestionado. São analisados os resultados dos testes nos quais é minimizado o número de pacotes descartados, aumentando a eficiência nos protocolos de transporte quando configurados para o funcionamento baseado em taxa. Como resultado final, são comparados os resultados dos testes com o funcionamento

normal do protocolo TCP, sendo apresentada a porcentagem de otimização do fluxo quando é utilizado o mecanismo e2e. Finaliza-se a dissertação com as conclusões e possíveis trabalhos futuros.

2. TRABALHOS RELACIONADOS

Neste capítulo, serão apresentados os métodos mais relevantes para medir a largura de banda, entre os quais estão as técnicas de provas passivas e ativas, o controle de congestionamento do TCP, alguns protocolos de transporte baseados em taxa e por último as técnicas de controle de tráfego.

2.1. Técnicas para medir a largura de banda

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 podem ser transmitidos na rede. A largura de banda pode ser medida de forma passiva ou ativa. A técnica passiva se restringe a coletar informações relacionadas à largura de banda, apenas dos pacotes que passam na rede, injetados por outras aplicações. Em contraste, a técnica de provas ativas (*active probing*) se refere à injeção de pacotes na rede para a medição de suas características.

Os métodos analisados para calcular a largura de banda disponível são baseados em uma série de técnicas que medem de forma ativa ou passiva a quantidade de bits que podem trafegar na rede em um determinado período. Muitos deles podem fornecer estimativas exatas sob determinadas circunstâncias [35]. 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 o método de provas ativas [18] [42].

2.1.1. Provas Ativas

Nas provas ativas é guardado em cada pacote o tempo de saída registrado no transmissor, como é mostrado na figura 1. A diferença de tempo entre cada pacote no instante da saída

é chamado de "*Interdeparture-Time*". Quando os pacotes chegam ao roteador, estes são recebidos aplicando a política FCFS (*First Come First Serve*), onde o primeiro pacote recebido será o primeiro a ser servido. Durante este período o roteador poderá receber pacotes que não pertencem às provas ativas, e que também serão enfileirados na estrutura FIFO (*First In, First Out*), gerando um tempo de atraso ou "*delay*". Isto resulta em um espaçamento final maior do que o tempo "*Interdeparture*". Quando os pacotes chegam ao receptor, as medidas de tempo são guardadas e, esta diferença é chamada de "*Interarrival time*". Estes pacotes deverão retornar ao transmissor para o cálculo das estimativas da largura de banda disponível a partir dos tempos capturados. Na Figura 1 o tempo entre pacotes subseqüentes na origem (lado esquerdo) é o *interdepature time* e o tempo entre pacotes subseqüentes na chegada (lado direito) é o *interarrival time*.

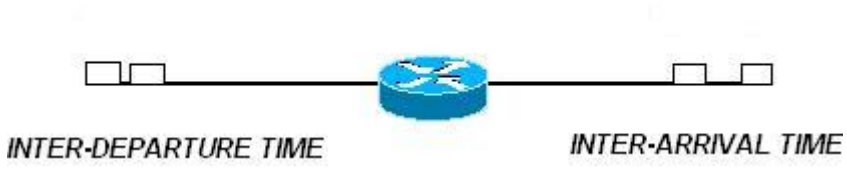

Figura 1. Provas ativas

2.1.2. Análise de técnicas

Nas análises das técnicas de estimativa de largura de banda, foram avaliadas também técnicas e ferramentas para determinar a capacidade do canal, a largura de banda disponível e a estimativa da capacidade de transferência, para 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 para calcular a largura de banda foi realizada, sendo analisados os códigos do **pathchirp** [37] que foi desenvolvido usando o código de **Netdyn** [3] como ponto de partida, do **IGI** [18] que introduz o conceito de "*packet transmission rate*" (PTR – Taxa de Transmissão de Pacotes) e também dos aplicativos **pathload** e **pathrate** [8], **cprobe** [4] e **pipechar** [20] os quais têm em comum a realização de medidas baseadas em técnicas de envio de

pacotes de prova tais como "*packet pair"* (par de pacotes) e "*packet train"* (trem ou série de pacotes) [18].

Duas etapas são identificadas nas técnicas de provas ativas: 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 do tempo de recepção. A estimativa compreende o processo estatístico e heurístico de análise das medidas e da geração de uma expectativa de acordo com algum modelo da rede.

Três medidas estão constantemente associadas para estimar a largura de banda: capacidade, largura de banda disponível e *"Bulk-Transport-Capacity"* (BTC – Capacidade de Transporte em Larga Escala). A *capacidade* é a vazão (*throughput)* máxima fornecida a uma aplicação quando não há nenhuma outra carga ou tráfego. A *largura de banda disponível* é a vazão máxima fornecida para uma aplicação dada a carga existente. O *BTC* [26] define a capacidade de transferir grande quantidade de dados durante uma conexão.

Existem diferentes técnicas utilizadas para encontrar as medidas anteriormente mencionadas, tais como "*Variable Packet Size"* (**VPS – Tamanho de Pacote Variável**), que estima a capacidade de cada enlace ao longo do caminho, "*Packet Pair/Train Dispersion"* (**PPTD – Dispersão de Par de Pacotes/Trem de Pacotes**), que estima a capacidade fim a fim [10], "*Self-Loading Periodic Streams"* (**SLoPS – Fluxos Periódicos com Carga Auto-Ajustável**) e "*Trains of Packet Pairs"* (**TOPP – Trens de Pares de Pacotes**) que são utilizados para estimar a largura de banda disponível. Estas técnicas serão brevemente descritas a seguir.

O *VPS* mede o RTT (*Round Trip Time*) desde a fonte até cada enlace do trajeto. O VPS usa o campo do cabeçalho IP "*Time-To-Live*" (TTL – no IP, o número máximo de roteadores intermediários que um pacote pode passar) para forçar os pacotes a finalizar o trajeto em um enlace determinado. O roteador neste enlace descartará o pacote retornando uma mensagem de erro ICMP "*Time-exceeded*" (tempo excedido). A fonte usa o pacote ICMP recebido para medir o RTT até este enlace.

O "*Packet Pair"* estima a capacidade fim a fim, quando a fonte envia múltiplos pares de pacotes ao receptor [17]. Cada par de pacotes é composto por pacotes do mesmo tamanho enviados "*back-to-back*" (um imediatamente após o outro). A dispersão do par de pacotes é igual a diferença de tempo de chegada no receptor de cada pacote.

O "*Packet Train"* estende a técnica de par de pacotes usando múltiplos pacotes *back-to-back*. A dispersão de um trem de pacotes 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 fim a fim 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 atraso dos pacotes de prova [23]. Se a taxa do fluxo for maior que a largura de banda disponível do trajeto, o fluxo causará sobrecarga no enfileiramento do enlace de menor largura de banda do trajeto.

O *TOPP* estende a técnica "*packet train",* estimando a largura de banda com o envio de vários pares de pacotes, incrementando gradativamente a taxa de transmissão, diminuindo o intervalo entre cada pacote do par a cada seqüência do trem.

2.1.3. TOPP

O "*Train Of Packet Pairs"* (TOPP) é uma medida usada para estimar a largura de banda baseada em provas ativas e inclui análises por regressão segmentada [28], que significa que as variações da largura de banda serão analisadas em cada estimativa com as medições anteriores ajudando a detectar engarrafamentos que seriam invisíveis aos outros métodos. Este método pode estimar a largura de banda disponível em um caminho congestionado. Ao contrário das estimativas tradicionais do par de pacotes, esta estimativa não está limitada pela taxa. Este método tenta solucionar alguns problemas que outros métodos não consideram, que incluem a inabilidade de identificar diversas situações de congestionamento que ocorrem em enlaces distintos.

2.1.3.1. 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 TOPP, 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 "Δo" 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 par de pacotes, "ΔT", é selecionado de tal maneira que o enfileiramento entre dois pacotes de prova seja pequeno. Isto beneficia a fase de provas, pois os nós ao longo do trajeto não experimentarão rajadas longas [7].

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. Tendo-se "*b*" como tamanho dos pacotes "ΔR" como tempo de separação, a largura de banda experimentada através do enlace pode ser estimada com a seguinte equação:

$$f = b / \Delta R \qquad (1)$$

2.2. Mecanismo de controle de congestionamento na camada de transporte

2.2.1. Controle de congestionamento do TCP

O controle de congestionamento do TCP padronizado na RFC 2582 [2] define os mecanismos usados no algoritmo tal como: "*slow start*", "*congestion avoidance*", "*fast retransmit*" e "*fast recovery*". Serão relatados abaixo estes algoritmos como implementados no TCP New Reno. Outros tipos de TCP podem ter comportamentos diferentes, já que as implementações do controle de congestionamento são o que diferem um TCP de outro (por exemplo, TCP New Reno de TCP Vegas).

Os termos comuns do controle de congestionamento são:

Segmento (*"Segment"*): qualquer pacote de dados TCP ou *acknowledgement* (ack).
Tamanho máximo de segmento no emissor (*"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.
Tamanho máximo de segmento no receptor (*"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 início da conexão. Se o MSS não é usado, o valor por default será de 536 bytes. O tamanho do segmento não inclui os cabeçalhos.
Segmento Cheio (*"Full-Sized Segment"* - FSS): segmento com o máximo número de bytes permitido.
Janela do receptor (*"Receiver Window"* - rwnd): é a mais recente *advertised received window* [34].
Janela de Congestionamento (*"Congestion Window"* - cwnd): variável de estado que limita a quantidade de dados que o TCP pode enviar.
Janela Inicial (*"Initial Window"* - IW): tamanho inicial do cwnd depois que a fase de conexão (*three-way handshake*) é concluída.
Janela de Perda (*"Loss Window"* - LW): tamanho do cwnd depois que o TCP detecta uma perda, causado pelo mecanismo de *timeout*.

Janela de Reinício (*"Restart Window"* - RW): tamanho do cwnd depois que o TCP reinicia a retransmissão causada por uma rajada potencialmente inapropriada [33].
Pacotes em trânsito (*"Flight Size"*): quantidade de dados que foram enviados, mas que não receberam o ack.

2.2.1.1. Slow start

Os algoritmos de partida lenta ("*slow start"*) e para evitar congestionamento ("*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 [2].

Começar a transmissão na rede com condições desconhecidas obriga o TCP a testar a rota para determinar a capacidade de banda disponível e prevenir congestionamento. O algoritmo de "*slow start"* cumpre com este propósito e é usado no início da transferência (e também depois de reparar uma perda detectada por "*timeout"*).

Outra variável de estado é o Limiar de Partida Lenta ("*slow start threshold"* - ssthresh) o qual determina qual algoritmo, "*slow start"* ou "*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 ao congestionamento. O "*slow start"* é usado quando a variável cwnd é menor que *ssthresh*, e o "*congestion avoidance"* é usado quando a variável *cwnd* é maior que *ssthresh* [2].

Durante o "*slow start"*, o TCP incrementa cwnd pelo menos em um MSS para cada "ack" recebido. O "*slow start"* termina quando cwnd excede seu valor máximo permitido 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 seja detectado congestionamento.

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

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

2.2.1.2. Fast Retransmit / Fast Recovery

Um receptor TCP deve enviar um *ack* duplicado ("*duplicate ack*") a cada vez que recebe um segmento fora de ordem. O propósito deste *ack* é informar ao emissor o número de seqüência esperado, quando um segmento foi perdido ou foi recebido fora de ordem.

Na perspectiva do emissor, o ack duplicado 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 [2].

O emissor TCP deve usar o algoritmo de retransmissão rápida ("*fast retransmit*") para detectar e reparar perdas baseadas no *ack* duplicado. A retransmissão rápida usa a chegada de três acks duplicados como indicativo de 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 de recuperação rápida ("*fast recovery*") é executado até um ack não duplicado (na seqüência correta) chegar. A razão do TCP não entrar no estado de partida lenta é devida ao ack duplicado indicar que apesar do segmento ter sido perdido, outros segmentos continuaram chegando, então a condição que causou a perda não é tão grave, ou foi transitória.

Os algoritmos de retransmissão rápida e recuperação rápida ("*fast retransmit"* e "*fast recovery"*) são implementados da seguinte forma:

1. Quando o terceiro *ack* duplicado é recebido, *ssthresh* adquire um valor não maior que o resultado na equação (2).
2. Quando o segmento perdido é retransmitido, *cwnd* adquire o valor de ***ssthresh* + *3*SMSS***
3. A cada "*duplicate ack"* recebido o *cwnd* é incrementado em um MSS.
4. Um segmento novo é transmitido 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 recebe o valor de *ssthresh*.

2.2.2. TCP Pacing

O projeto "*tcp_pacing*" [1] insere espaços de tempo entre os pacotes, com a finalidade de variar a taxa de transmissão e tentar ajustar o mecanismo "*congestion avoidance"* dando um melhor desempenho ao protocolo TCP [48]. 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 fornecido por *TCP-Vegas*.
3. Cálculo da janela esperada e o sincronismo entre segmentos nessa janela.
4. Um mecanismo de relógio para emitir os segmentos.

2.2.3. 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 sensíveis ao atraso ("*delay*"), tais como, "*mídia streaming*", telefonia e jogos, onde se prefere que os dados cheguem a tempo de serem exibidos do que todos os dados cheguem (confiabilidade). O

DCCP emprega transmissão baseada em taxa [30] (*rate based*) espaçando os pacotes [21] e realizando controle das rajadas.

O DCCP utiliza um mecanismo de identificadores do controle de congestionamento (*Congestion Control Identifiers* ou CCID). O transmissor envia os pacotes de dados DCCP e o receptor responde com pacotes DCCP-Ack.

2.2.4. 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 [24].

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, as filas estão sendo incrementadas 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.

2.3. Técnicas de controle de tráfego

2.3.1. Queue Discipline

"Queue discipline" (Qdisc) [36] é uma série de técnicas de enfileiramento que permitem mudar a maneira em que os dados são enviados, podendo ser utilizada para dar forma ao tráfego, possibilitando assim o controle do tráfego pela interface de rede.

Normalmente, os pacotes são enfileirados através da técnica FIFO, enviando aproximadamente um pacote a cada interrupção, conforme mostra a figura 2.
O mecanismo "*queue discipline*" usado por default pelos dispositivos de rede no Linux é chamado de *pfifo_fast*[2].

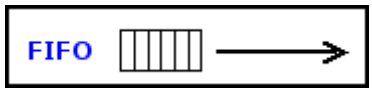

Figura 2. FIFO

O "*pfifo_fast"* significa que nenhum pacote recebe um tratamento especial [13]. Este utiliza três filas onde é aplicada uma regra FIFO. Estas regras FIFO são configurações definidas pelo usuário para controlar a latência, "burst" ou largura de banda. Quando uma das filas está sendo utilizada, as outras duas ficam em espera, não permitindo o uso das filas 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 filas, isto depende de cada regra. A maioria das técnicas FIFO pode enviar pacotes **P** alcançando uma taxa de transmissão máxima de pacote multiplicado por HZ[3] conforme mostra a equação 3:

$$Rate = P * HZ \qquad (3)$$

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

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

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 conseqüentemente perdas [13].

Outros tipos de "*queue discipline"* como "*Token Bucket 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).

2.3.2. Traffic Shaping

"*Traffic shaping*" é o termo geral dado a uma série de técnicas projetadas para priorizar políticas de transmissão de dados sobre pequenos trajetos da rede, isto permite que nestes trajetos não seja excedida a máxima largura de banda que este permite. Alguns efeitos de latência nos processos do enfileiramento podem ser experimentados com um retardo dos pacotes nas comunicações de rede quando há tráfego significativo. Outro caso seria que os pacotes podem ser descartados por excesso na fila. Estes problemas ocorrem porque não são dadas condições quando o pacote é enfileirado para a transmissão utilizando a técnica "*queue discipline*" baseada em FIFO e a transmissão é escalonada com FCFS.

2.3.2.1. Shaper Device

"*Shaper Device*" é um mecanismo simples, projetado para limitar a quantidade de largura de banda que cada trajeto pode utilizar. O "*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.

2.3.3. QoS Packet Scheduling

Enquanto o "*shaper device*" é muito utilizado em várias circunstâncias, muitas aplicações demandam mais flexíveis e sofisticadas técnicas de "*shaping*", tal como, padrões de QoS. As técnicas mais conhecidas são: "*Resource ReSerVation Protocol*" (RSVP) descrito na RFC-2205, "*Integrated Service*" descrito na RFC-1633 e "*Differentiated Service*" descrito na RFC-2475.

Os mecanismos de 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) [5], "*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 da QoS e os requisitos de tráfego IP.

3. MECANISMO E2E

O mecanismo e2e pode ser usado para o controle e moldagem de fluxos de qualquer protocolo de transporte. Ele reside em várias camadas da arquitetura de rede, pois necessita realizar várias funções (como identificar fluxos e medir banda fim a fim para determinados destinos) que são específicas para uma camada. Os módulos foram criados usando desenvolvimento ao nível de "kernel" [11]. Para medidas de largura de banda disponível, foram implementadas técnicas como pares de pacotes ("*Packet Pair*") e trens de pares de pacotes ("*Train Of Packet Pairs*"). No caso de existirem várias interfaces de rede, a seleção da melhor interface de rede para o envio de pacotes de um determinado fluxo também é feita. Para gerenciar os pacotes, foram implementadas também disciplinas de gerenciamento de filas. Na figura 3 pode-se ver os diferentes módulos de "*kernel*" que compõem a implementação do mecanismo e2e.

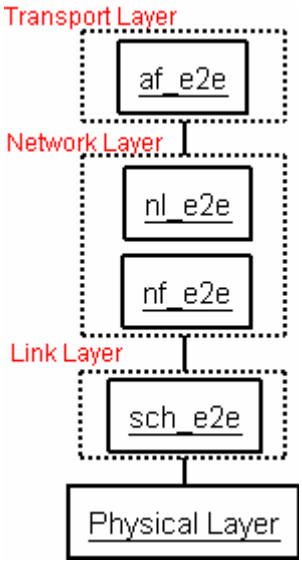

Figura 3. Representação dos módulos na arquitetura de rede

O "*af_e2e*" é o módulo que foi implementado originalmente para criar os protocolos de transporte XMTP no kernel. Ele define o nome do protocolo usado 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 não ter sido completamente acabado ele não será analisado neste trabalho. No entanto, o código pode ser usado como modelo para futuros desenvolvimentos que completem este trabalho.

Para substituir o módulo “af_e2e”, o projeto foi subdividido em diferentes módulos posicionados em diferentes camadas da arquitetura de rede: na camada de rede o “netlink_e2e” e o “netfilter_e2e” e na camada de enlace o “scheduler_e2e”.

O gerente de ativação (nl_e2e) se encarrega de receber as configurações que a camada de aplicação solicita para realizar o controle de tráfego.

O gerente de controle (nf_e2e) se encarrega de aplicar a configuração e iniciar as provas ativas para calcular as estimativas relacionadas com a largura de banda.

O gerente de controle do fluxo (sch_e2e) recebe os cálculos de estimativas e aplica a taxa necessária para atingir a largura de banda disponível.

Foram utilizadas funções de “netfilter” [22] ou filtro de rede, através do módulo “nf_e2e” que recebe os pacotes que entram na camada três. Neste ponto o “*netfilter*” pode fornecer um tratamento particularizado tal como “*packet filtering*”, “*packet mangling*” [12], NAT, “*Forwarding*”, etc. Com ajuda de “*netfilter*”, o módulo “nf_e2e” realiza a captura dos pacotes relacionados com o mecanismo e2e.

Foram utilizadas funções de “netlink” para implementar a comunicação entre a camada de aplicação com os módulos de “kernel”. As funções de “netlink” são usadas pelo módulo nl_e2e para estabelecer ou apagar configurações do mecanismo e2e. O procedimento de ativação consiste em uma configuração inicial que a aplicação envia em uma chamada de “netlink” utilizando como meio o módulo “nl_e2e”. Esta configuração é fornecida ao módulo “nf_e2e” para ativar o mecanismo de controle e começar as provas ativas e estimar a taxa de transmissão segundo a largura de banda disponível. A taxa de transmissão permitirá regular constantemente a quantidade de pacotes que podem ser enviados na rede.

O escalonador do módulo sch_e2e utiliza uma técnica própria de gerência de filas na camada de enlace [15]. O módulo “sch_e2e” recebe as estimativas calculadas pelo módulo “nf_e2e” e identifica a fila que deverá alterar sua taxa de transmissão.

3.1. ICMP Active Probing

O modulo “nf_e2e”, envia um par de pacotes ICMP segundo a RFC 792 do tipo “*Timestamp”* e “*Timestamp Reply Message” (13 e 14)* [13], [47], [3] cujas informações do cabeçalho são especificadas na Figura 4.

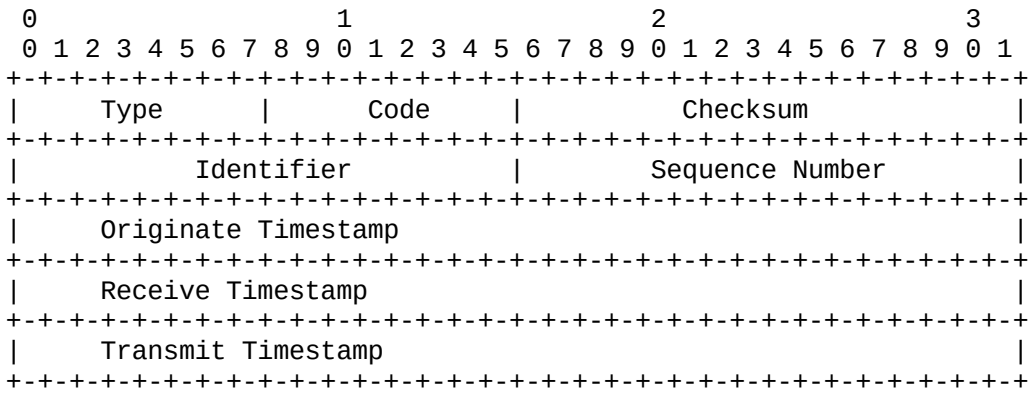

Figura 4. Cabeçalho ICMP tipo 13, 14.

Segundo as especificações do protocolo ICMP para TIMESTAMP, o tempo de origem (*originate timestamp*) corresponde ao tempo de saída do transmissor, 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 milissegundos desde a meia noite.

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

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

O mecanismo pode ser interpretado nas quatro primeiras camadas da arquitetura de rede como é ilustrado na figura 5. 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 interface de rede, isto permitirá a comparação de todos os resultados obtendo-se critérios de seleção da interface de rede mais adequada para o envio de pacotes.

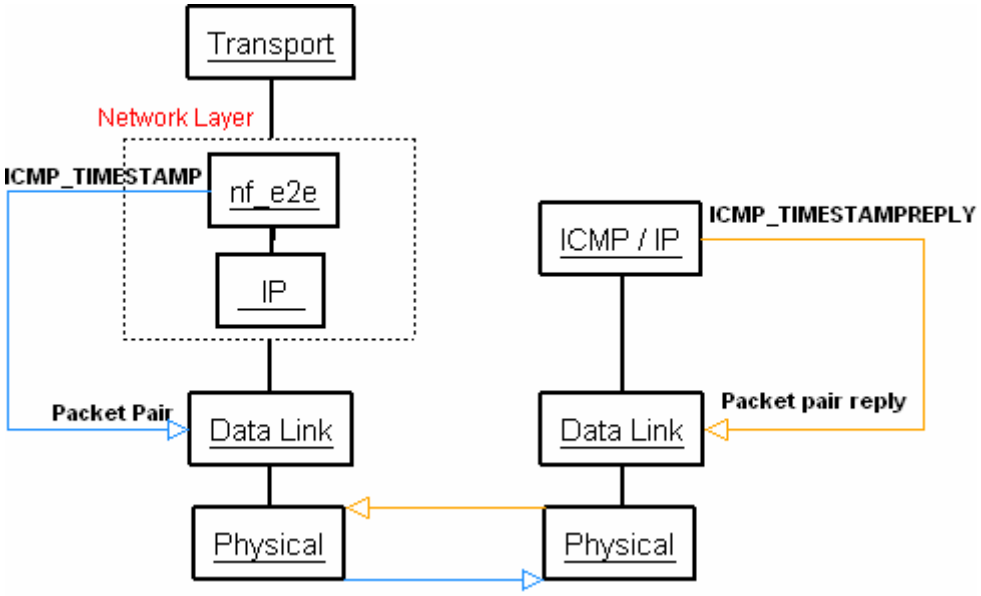

Figura 5. Provas ativas do mecanismo e2e.

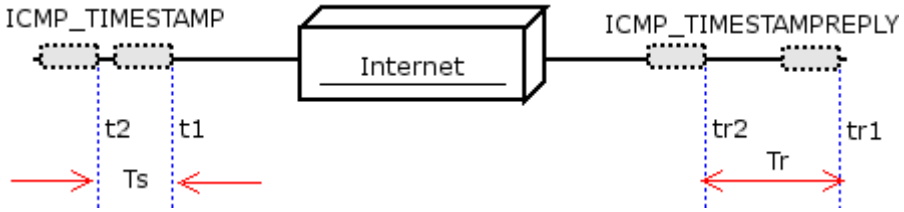

Figura 6. ICMP active probing

Como se ilustra na figura 6, o mecanismo de provas ativas permite conhecer a diferença entre o tempo de chegada dos pacotes ao receptor (Tr) como mostra a equação 4:

$$Tr = tr2 - tr1 \qquad (4)$$

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

$$Ts_i = t2 - t1 \qquad (5)$$

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

$$Delay = Tr - Ts \qquad (6)$$

Ou integrando as equações (4) e (5), pode-se calcular o delay de forma direta, como se mostra na seguinte equação:

$$Delay = (tr2 - tr1) - (t2 - t1) \qquad (7)$$

3.2. e2e - Active Probing

A RFC 792 especifica uma precisão de tempo em milissegundos [13], 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 microssegundos para atingir a taxa estimada [48]. Com este propósito, foram adicionados três campos de 32 bits ao cabeçalho ICMP informando o valor complementar em microssegundos, dos três primeiros campos como é mostrado na Figura 7.

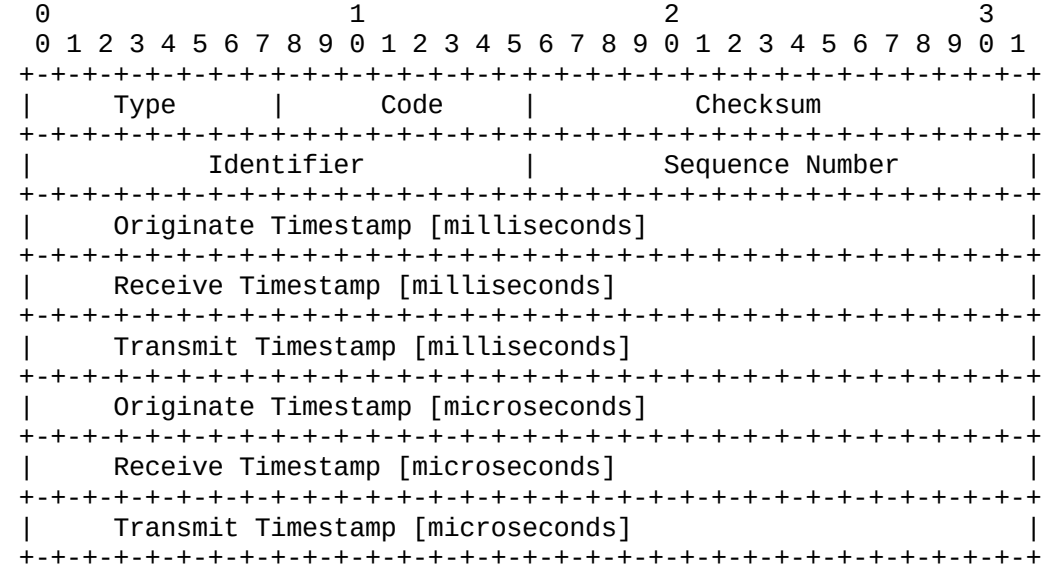

Figura 7. ICMP 792 com alterações do cabeçalho.

Esta modificação do cabeçalho permite obter 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 8.

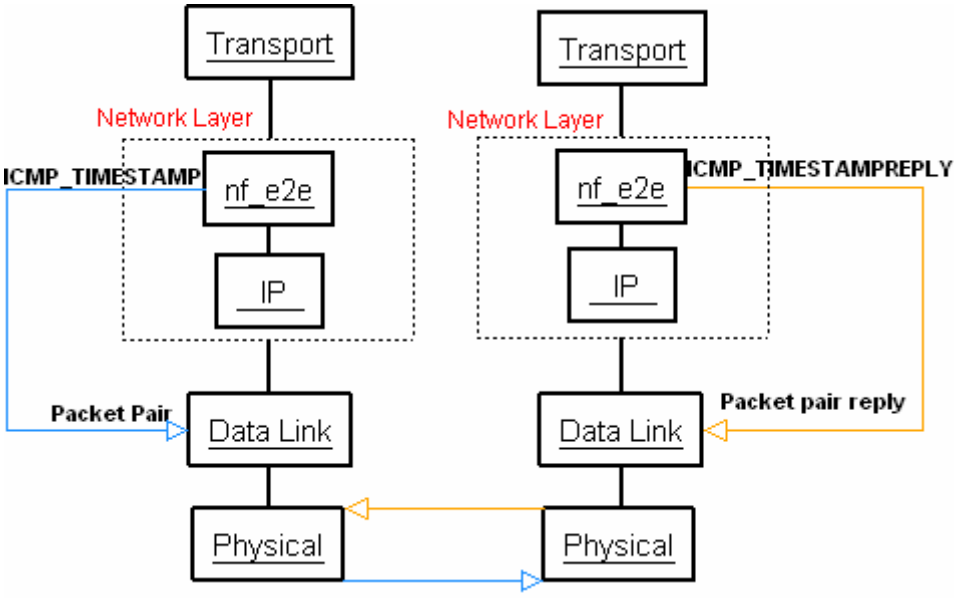

Figura 8. ICMP-e2e timestamp

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

3.3. 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.*** O intervalo entre cada trem é representado por ***Ts***, como é mostrado na Figura 9**.**

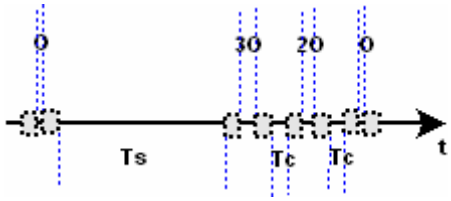

Figura 9. TOPP-e2e

Diferente 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. Assim, 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 (8):

$$\text{Média\_delay} = (\ \text{delay}(\mathbf{0})+\text{delay}(\mathbf{1})+...+\text{delay}(\mathbf{n-1})\ )/\mathbf{n} \qquad (8)$$

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 (9):

$$\text{Ma\_delay} = (1-1/\alpha)\ \text{Ma\_delay} + \text{Média\_delay}/\ \alpha \qquad (9)$$

O número de interrupções (Ticks) que fornece a taxa estimada pode ser encontrado com a equação (10):

$$\text{Ticks} = \text{Ma\_delay} * \text{HZ} \qquad (10)$$

3.4. Scheduler e2e

Para produzir o controle de fluxo baseado em taxa, foi desenvolvida uma nova técnica de gerência de fila no módulo "sch_e2e", baseada em enfileiramentos paralelos e controle da taxa, tal como se mostra na figura 10.

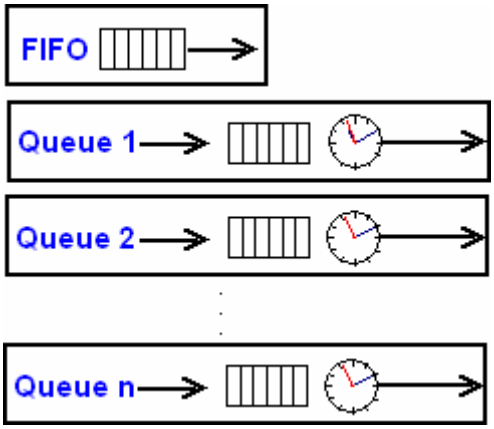

Figura 10. Esquema do scheduler sch_e2e

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

As outras filas implementam o controle do fluxo que foi configurado pela aplicação com o fornecimento de parâmetros de IP, protocolo e porta, associado a um identificador de fila.

Cada fluxo é identificado e enviado para a respectiva fila, garantindo desta maneira que a taxa seja variável de forma independente para cada fila.

Cada IP, protocolo e porta do pacote são comparados com a lista de configuração do gerente de controle, cujo resultado, é um identificador de fila. Cada pacote é marcado com este identificador, continuando o trajeto para a camada dois (a fila deve ser identificada porque as variações de taxa são diferentes em cada uma). O gerente de controle de fluxo captura o pacote, identifica a fila e enfileira o pacote nesta (em termos de programação de kernel a função é chamada de enqueue()).

3.5. Mecanismo de controle de fluxo

O mecanismo de controle de fluxo depende da realização freqüente das provas ativas para obter constantemente as estimativas do atraso. Quando os pacotes com números de seqüência do TOPP correspondente a um determinado trem são recebidos completamente

pelo 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). Este por sua vez guarda as informações dos resultados. O "sk_buff" criado tem como finalidade configurar a nova taxa de transmissão que será aplicada nos pacotes que ainda estão na fila. O valor da taxa é aplicado depois de realizadas as estimativas, o que garante variabilidade.

As provas ativas são realizadas através de cada interface de rede, obtendo-se cálculos de taxas que comparadas identificam o dispositivo de rede mais adequado para transmitir o fluxo de pacotes. Para este propósito a rota padrão do pacote deve ser alterada para alcançar este dispositivo de rede.

O controle de fluxo depende da função desenfileiramento (em termos de programação de kernel igualmente denominada como dequeue()) do módulo "sch_e2e", o qual tem um "token" que percorre todas as filas verificando se os pacotes podem ser enviados, dependendo do tempo de saída, como é mostrado na figura 11.

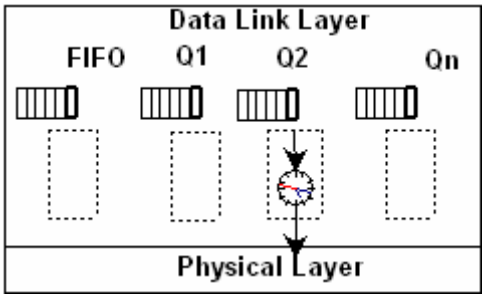

Figura 11. Mecanismo de dequeue

A primeira fila efetuará o desenfileiramento sempre que houver um pacote. O "token" continua com a próxima fila "queue" na interrupção seguinte. Se nessa próxima fila não for o instante de tempo para o envio do pacote, o "token" continua o percurso nas outras filas até encontrar o pacote que deve ser enviado. Se o percurso chegar até a última fila e verificar que nenhum pacote foi transmitido, o percurso do "token" continua na posição FIFO e transmite o pacote que estiver na fila.

Para cada fila, a taxa de transmissão pode variar dinamicamente dependendo das estimativas como mostra a equação 11. 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).

$$Rate = P * Ticks \qquad (11)$$

A quantidade de pacotes em cada fila dependerá do número de "ticks" estimado, isto oferece um controle do tamanho da rajada ("burst") tal como se representa na equação (12), onde "queue_tx_len" é a quantidade de pacotes que a interface permite.

$$Burst = queue\_tx\_len / Ticks \qquad (12)$$

Este limite de pacotes permitirá que em cada fila seja garantido um espaço de memória na interface de rede para todos os fluxos, sem armazenar pacotes de forma desnecessária, prevenindo-se a obstrução das outras filas.

4. Implementação do mecanismo E2E no Kernel de Linux

Neste capítulo serão apresentados os tópicos relacionados com a implementação do mecanismo E2E, tais como o trajeto dos pacotes no Kernel de Linux e a modelagem do mecanismo E2E, destacando-se as partes dos módulos, especificação dos componentes, diagramas conceituais e a aplicação do mesmo.

4.1. Trajeto dos pacotes no Kernel de Linux

O trajeto de um pacote, entre as primeiras camadas da arquitetura de rede, pode ser explicado com alguns conceitos da API (Application Program Interface) do "kernel" de Linux [6], 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. Um adaptador Ethernet, por exemplo, normalmente está dividido em duas regiões usadas para receber e enviar pacotes, tal como, no modelo 3c509 com 4KB que usa 2KB para recepção (Rx), 2KB para transmissão (Tx), ou no modelo 3c509B com 8KB pode usar 4/4, 5/3 ou 6/2 para Rx/Tx respectivamente [12].

A interface no Linux é representada pela estrutura de dados chamada "net_device" [16]. Esta estrutura captura as informações vindas do meio físico ou da camada três, relativas ao nome da interface, memória I/O, interrupção, informação da fila 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*" [9]. Depois da recepção do pacote, o adaptador usará

uma interrupção para informar a *cpu* deste evento. Um novo "buffer" de rede "*sk_buff*"[4] (socket buffer) é alocado e o pacote é copiado neste [14].

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 na arquitetura de rede.

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 encadeadas de "sk_buff", segundo funções para o controle da memória. Os "buffers" encadeados nas listas para operações de rede são adicionados ao final das listas e removidos no início. A maioria das operações de rede acontece durante 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.

4.2. Modelagem do mecanismo e2e

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

Este mecanismo deve medir a largura de banda fim a fim, enviando pacotes de prova por todas as interfaces ativas de rede. Isto permite determinar qual interface pode fornecer a melhor taxa de transmissão informando o enlace menos congestionado, que seria mais eficiente para enviar os pacotes. O mecanismo deve permitir que um protocolo de transporte envie os dados, diminuindo a probabilidade de perda de pacotes ajustando-se à capacidade do canal e minimizando o congestionamento.

---

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

4.2.1. Requisitos gerais

O mecanismo e2e deve permitir o envio de pacotes usando 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.

4.2.2. 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.

4.2.3. Casos de uso

4.2.3.1. 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.

4.2.3.2. 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 envia 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

4.2.3.3. 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 informando 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	
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.

4.2.3.4. 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	

4.2.3.5. 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.

4.2.4. Diagrama de casos de uso

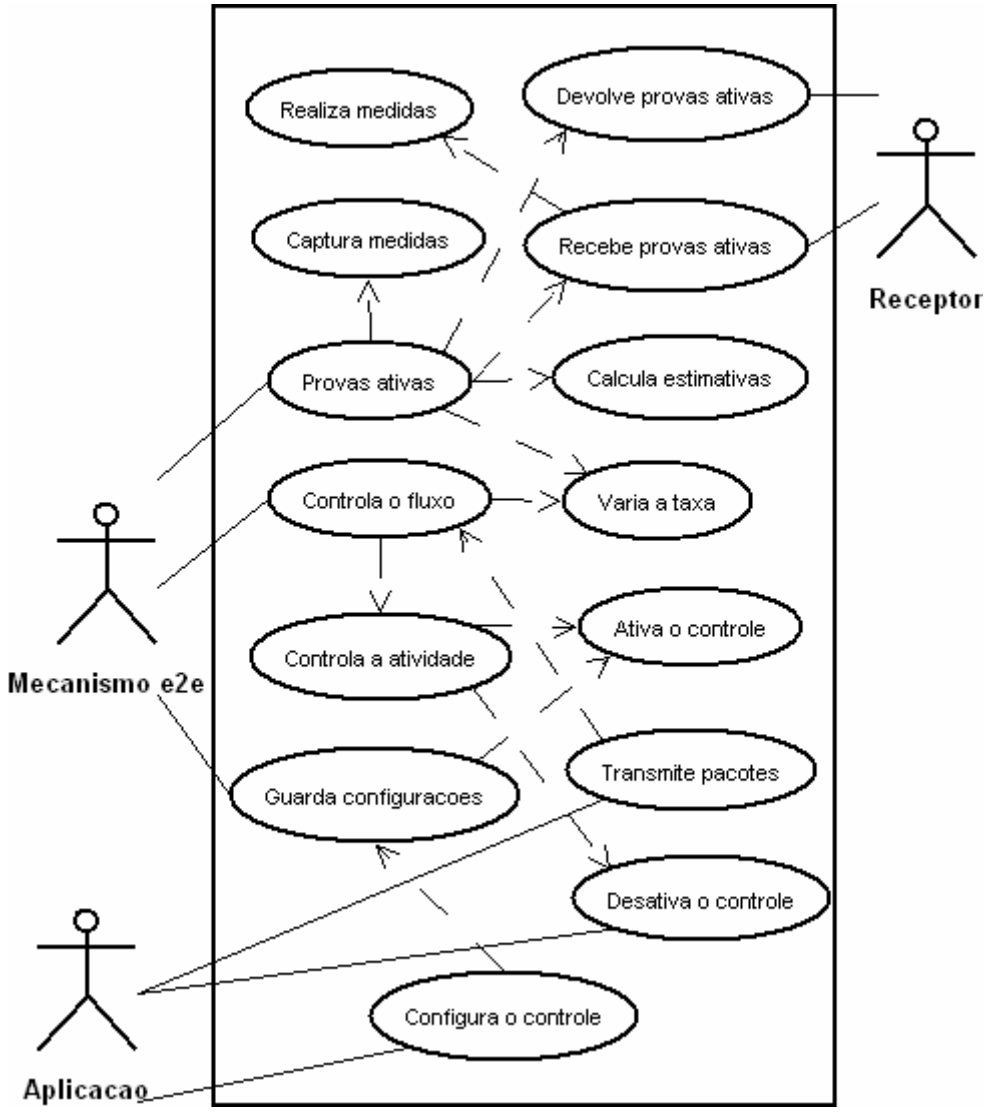

Figura 12. Diagrama de casos de uso.

4.3. 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)

4.3.1. 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 13.

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 fornecida 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”.

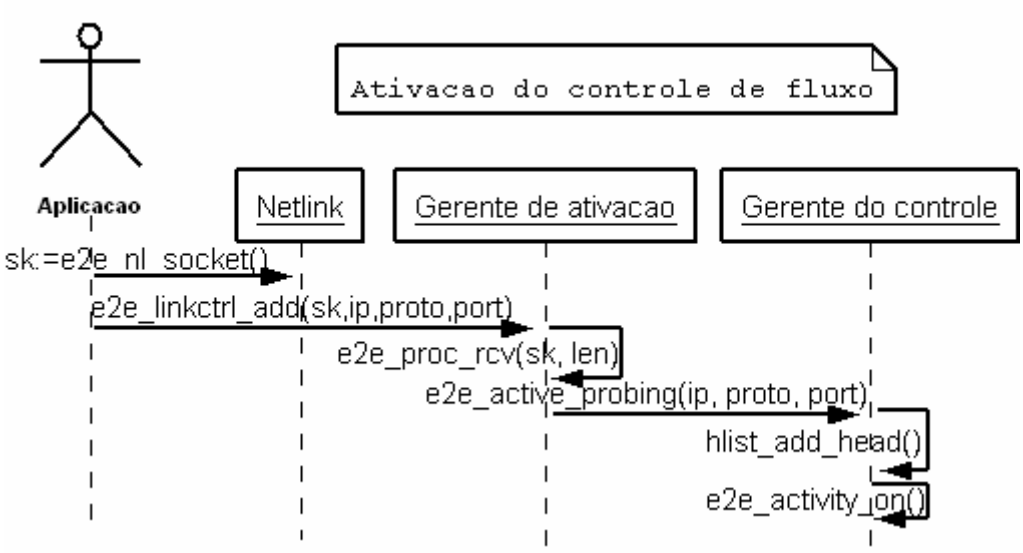

Figura 13. Diagrama de interação para ativação do controle de fluxo.

4.3.2. 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 14. O *receptor* adiciona aos pacotes o tempo de chegada e os devolve. O “n*etfilter”*

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 microssegundos se as medidas estiverem em milissegundos 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".

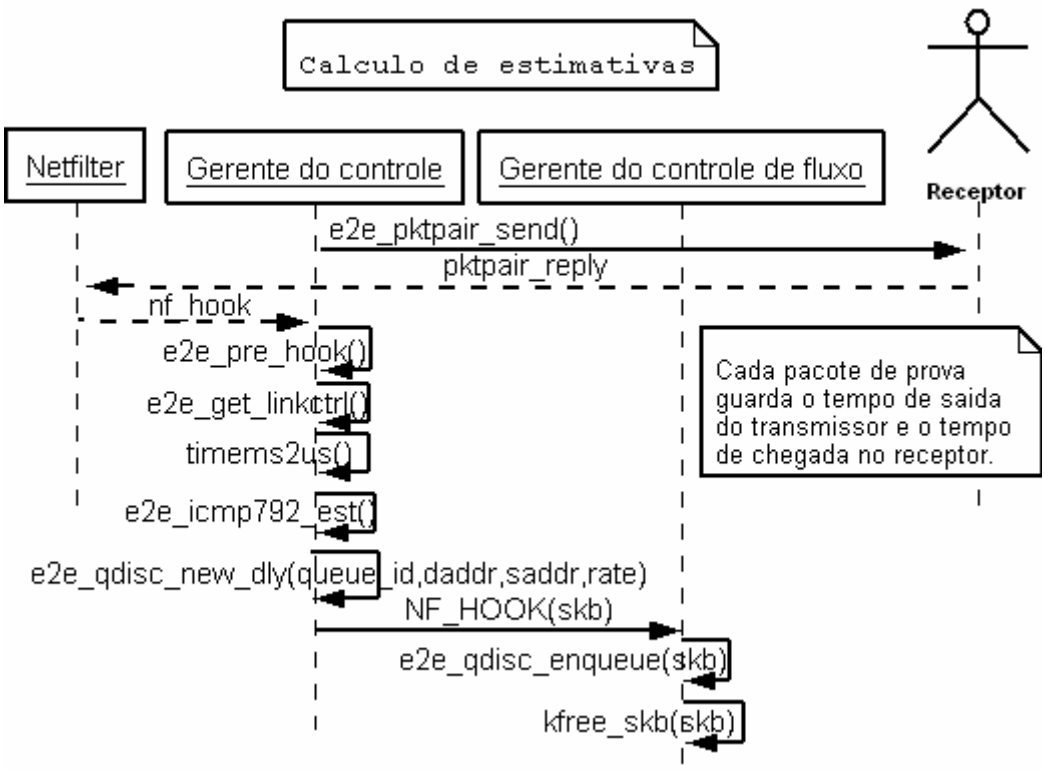

Figura 14. Diagrama de interação para cálculo de estimativas.

4.3.3. Controle de filas

A *aplicação* transmite os pacotes utilizando normalmente a função "sendto" do C. Os pacotes são filtrados com "n*etfilter"* e entregues ao *gerente de controle* com a função "e2e_post_hook". Cada IP, protocolo e porta 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 fila (qid). Cada pacote é marcado com este identificador, continuando o trajeto para a camada dois (a fila deve ser identificada porque as variações de taxa são diferentes em cada uma). 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.

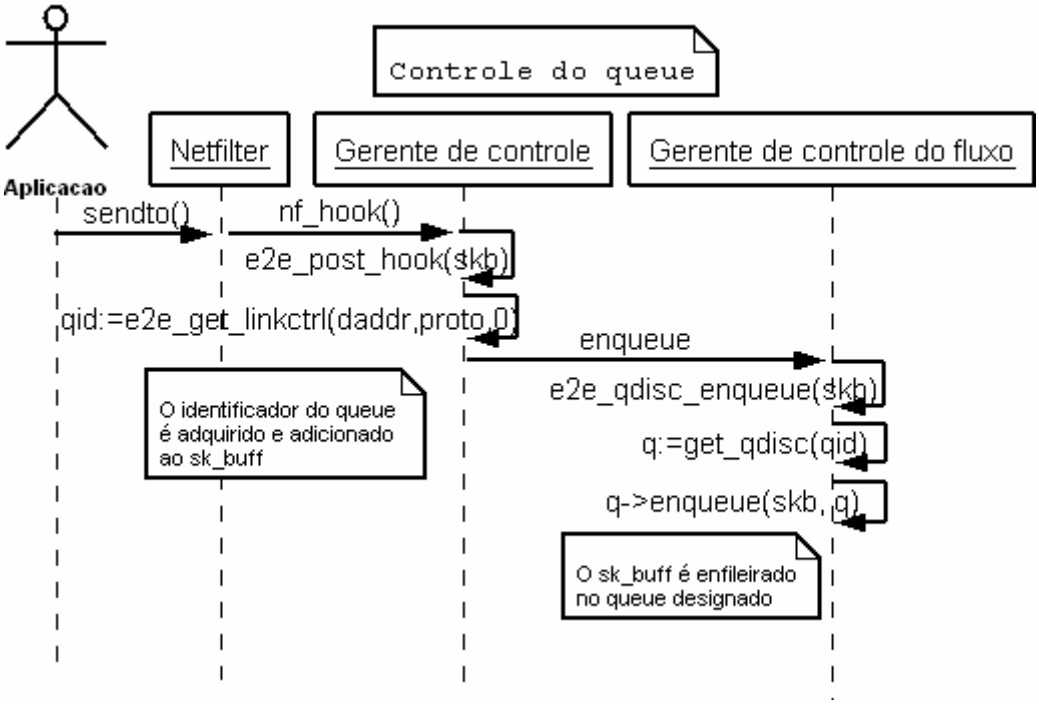

Figura 15. Diagrama de interação para o controle de filas

4.3.4. Seleção da interface de rede

As provas ativas constantemente ofereceram medidas de atraso para todas as 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 a fila desse dispositivo de rede, como se apresenta no diagrama de interação do controle de filas, anteriormente descrito.

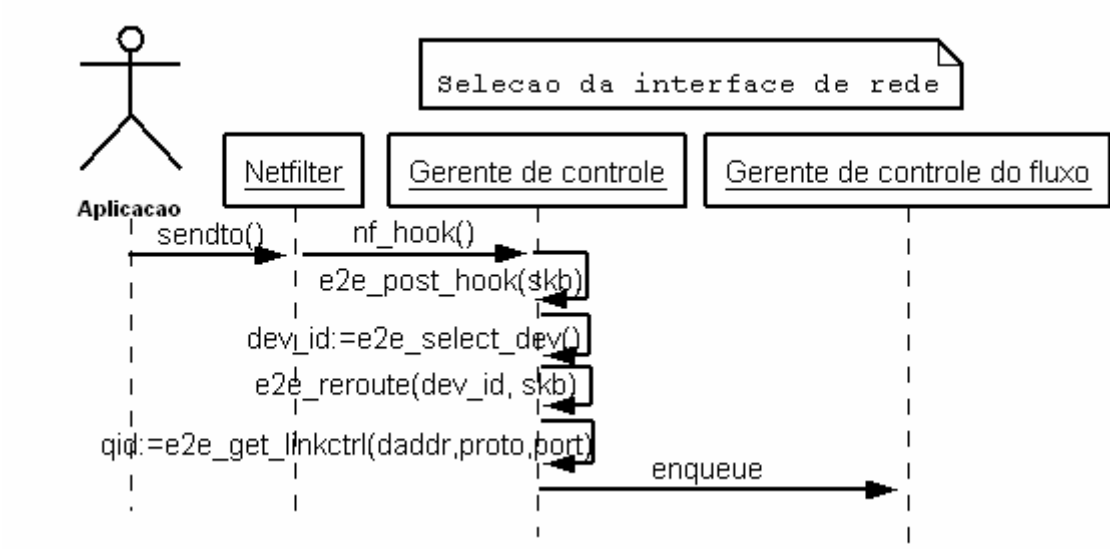

Figura 16. Diagrama de interação para seleção da interface de rede

4.4. Diagrama de atividades para o controle do desenfileiramento

O controle do desenfileiramento permitirá a transmissão dos pacotes que estão em espera na fila segundo a taxa estimada. A figura 17 apresenta as atividades do desenfileiramento sobre várias filas. Um relógio verifica freqüentemente o tempo de envio (time_to_send) comparando o número de interrupção em que o pacote deve ser transmitido com o atual. Este número de interrupção em que o pacote deve ser transmitido deve ser igual ao número de "ticks" estimado com as provas ativas, mais o número de interrupção no instante de configuração da nova taxa. Se não for o número de interrupção para o envio, o pacote volta para a fila na atividade "requeue" e o controle avança para a próxima fila continuando as atividades do desenfileiramento.

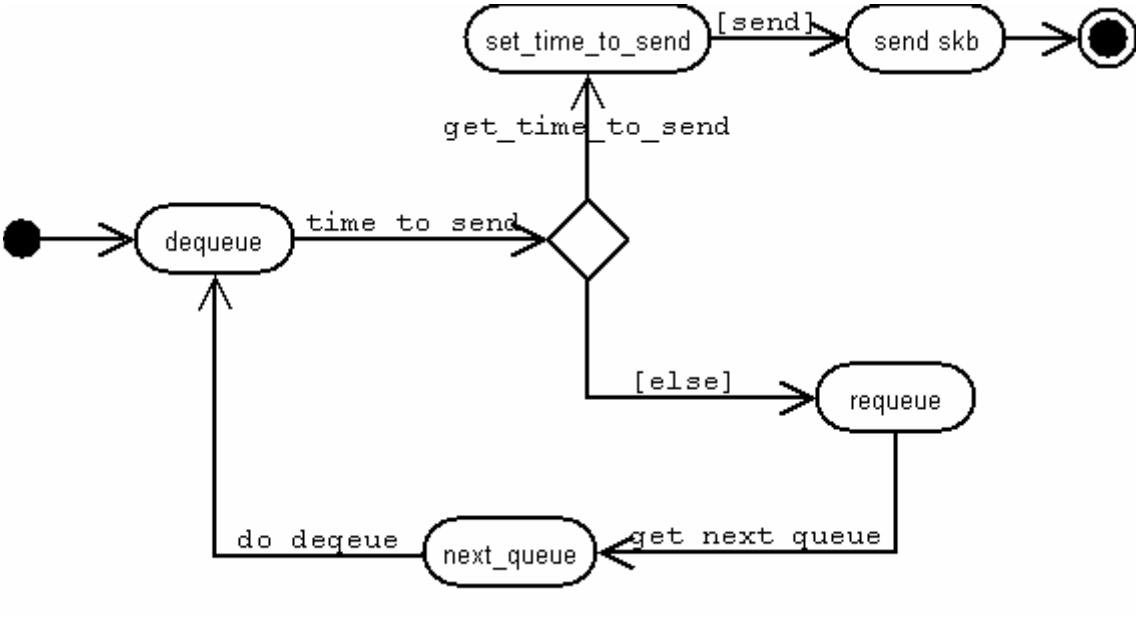

Figura 17. Diagrama de atividades para o controle do desenfileiramento

4.5. 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

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

função e2e_nl_close(**sock_id**). Em cada configuração de protocolo, podem ser aceitas definições do tipo IPPROTO_UDP, IPPROTO_TCP[6], etc.

4.6. 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**

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

5. TESTES

Neste capítulo são apresentados os testes realizados com o mecanismo e2e, sendo detalhados os componentes, comandos, configuração de rede dos elementos que compreendem os testes, realização dos experimentos e os resultados.

Para obter resultados com o mecanismo e2e, foram realizados testes usando aplicações cliente/servidor IPERF[7] [45], TFTP para TCP e UDP respectivamente. A configuração de rede do teste consiste de duas redes "ethernet-100Mbps" conectadas 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 [31]. Na configuração da largura de banda, foi executado o comando de "*traffic control"* (roteador esquerdo da figura) [19] restringindo o tamanho das rajadas ("burst") em 15 Kbytes e a latência em 5 milissegundos usando o escalonador "*Token Bucket Filter" (tbf)* [5]. O comando foi aplicado da seguinte maneira: *"tc qdisc add dev eth0 tbf burst 15000 rate 28kbit latency 5ms"*.

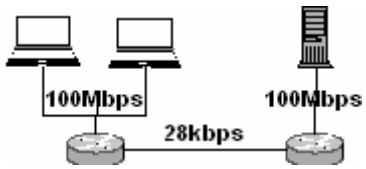

Figura 18. Diagrama de rede para testes

O servidor (lado direito da figura) roda Linux com o "kernel 2.4.26", e os sistemas clientes rodam Linux com o "kernel 2.6.17" compilado com a variável de freqüência HZ_100[8] equivalente a um período de 1ms (período mínimo entre cada interrupção). Esta configuração geralmente é importante para obter 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 corretamente as interrupções para o envio de cada pacote.

---

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

[8] HZ definido no kernel/kconfig.hz

Os testes usam duas máquinas em um lado da rede (lado esquerdo da figura 18) que enviam um fluxo constante ao mesmo tempo, de forma unidirecional, através de um enlace com largura de banda limitada durante uma hora. 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 enlace congestionado.

Para obter as medidas foi alterado o código do "*Token Bucket Filter*" (TBF) para imprimir a cada cinco segundos um contador de pacotes, dado indispensável para calcular o número de pacotes descartados. Além disso, o código FIFO foi alterado para imprimir igualmente o contador de pacotes no computador que gera um fluxo "*cross-traffic*" para comparar o desempenho do TCP acompanhado de um tráfego baseado em taxa.

5.1. Testes com TCP

Este teste permite observar o comportamento típico do TCP quando não é utilizado o mecanismo e2e, produzido por dois fluxos IPERF em diferentes máquinas, e dá 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.

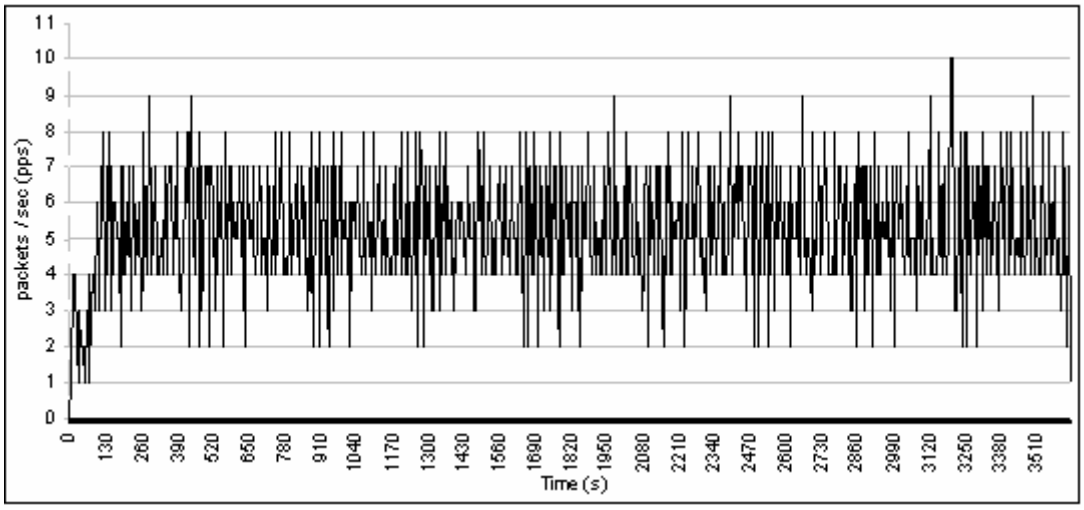

Figura 19. TCP sem o mecanismo e2e

A tabela 1 apresenta os resultados de cinco testes obtendo-se como resultado médio uma porcentagem de perda de pacotes de 9.1%, um aproveitamento da largura de banda de 11,48 kbps e 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	8,48
Test 2	3615,4	4085	5,01	11,6	3719	8,96
Test 3	3609,2	3820	5,11	11,9	3475	9,03
Test 4	3615,4	4033	4,98	11,6	3634	9,89
Test 5	3605,5	3877	4,79	11,1	3494	9,88
Media	*3610,4*	*3945,75*	*4,938*	*11,48*	*3586,75*	*9,09*

Tabela 1 Testes com TCP

A figura 20 mostra o acúmulo de pacotes desde o início 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.

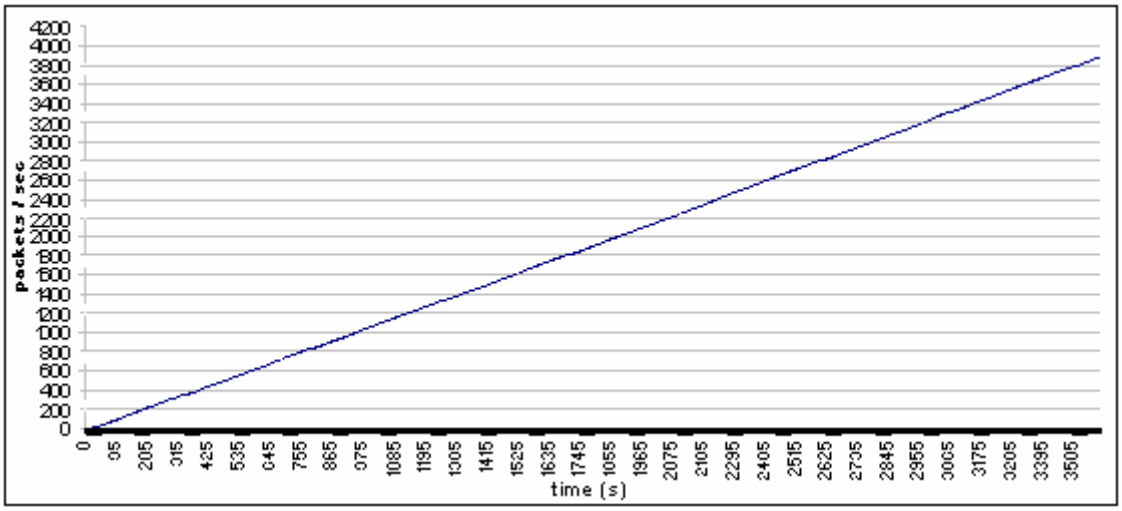

Figura 20. Gráfico cumulativo de pacotes do TCP

5.2. Teste e2e com UDP

Uma opção para realizar a medição com UDP era usar IPERF, mas nas experiências realizadas, foi determinado que o IPERF realiza controle de congestionamento diminuindo a taxa de transmissão atingindo a largura de banda disponível impedindo que

o mecanismo e2e efetuasse esta tarefa e obter medições certas com o UDP. Para os testes realizados com UDP, foi utilizado o esquema cliente servidor TFTP.

Nos resultados obtidos 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"* 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.

5.2.1. 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 85% 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 enviado 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 (B)	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 atraso medido, o qual varia de acordo com o congestionamento gerado pelo fluxo "cross traffic" TCP. A variação de pacotes transmitidos depende das medidas do atraso. Um aumento do atraso é gerado quando ocorre uma perda pelo "cross traffic" TCP, eventualmente a taxa de transmissão é diminuída. Quando o atraso diminui pode ser interpretado o início de um "slow start" do "cross traffic" TCP, conseqüentemente a variação da taxa é incrementada. Para poucas variações do atraso pode ser interpretado como um "congestion avoidance" freqüente do controle de congestionamento do TCP.

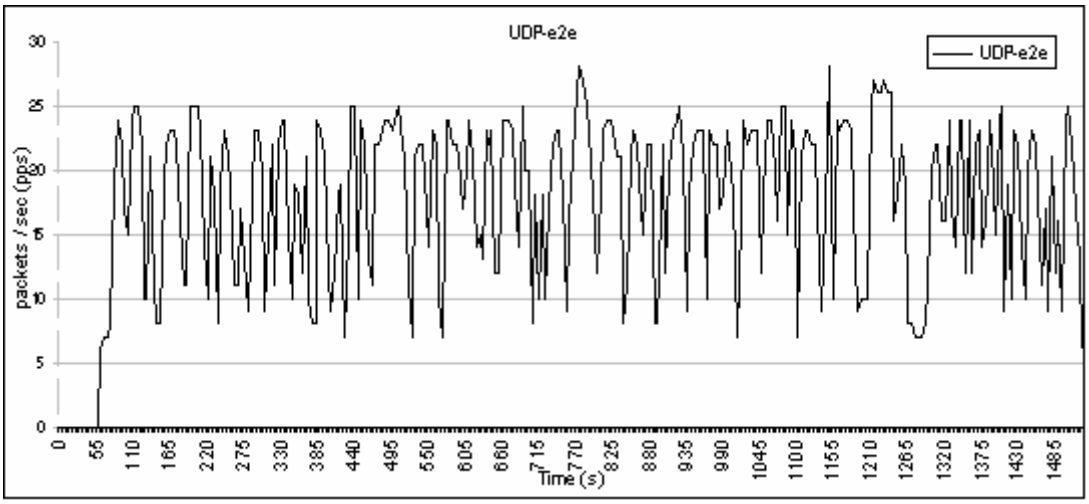

Figura 21. Fluxo UDP-e2e

A Figura 22 mostra o fluxo UDP (linha fina) utilizando o mecanismo e2e junto com o fluxo "cross traffic" TCP (linha grossa).

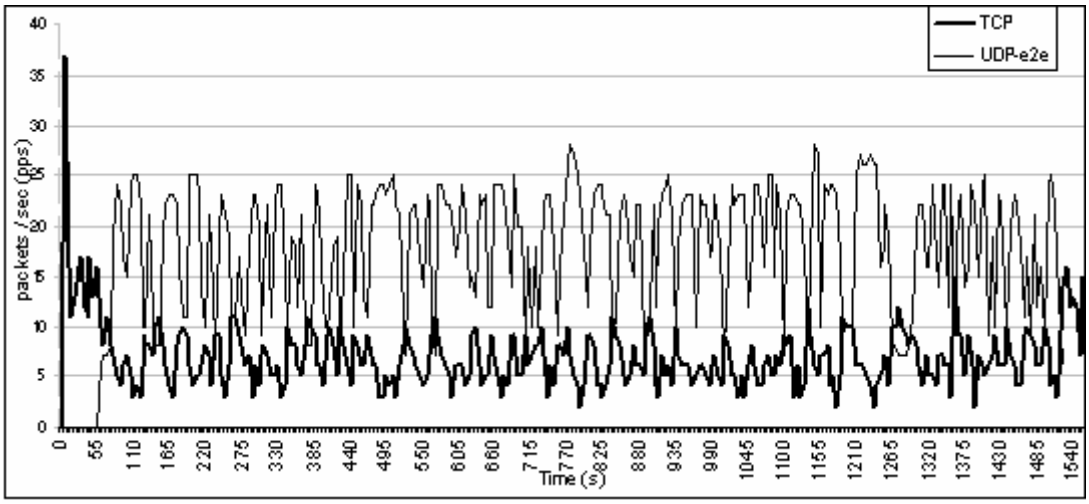

Figura 22. Fluxo UDP-e2e comparado com um fluxo cross traffic TCP

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.

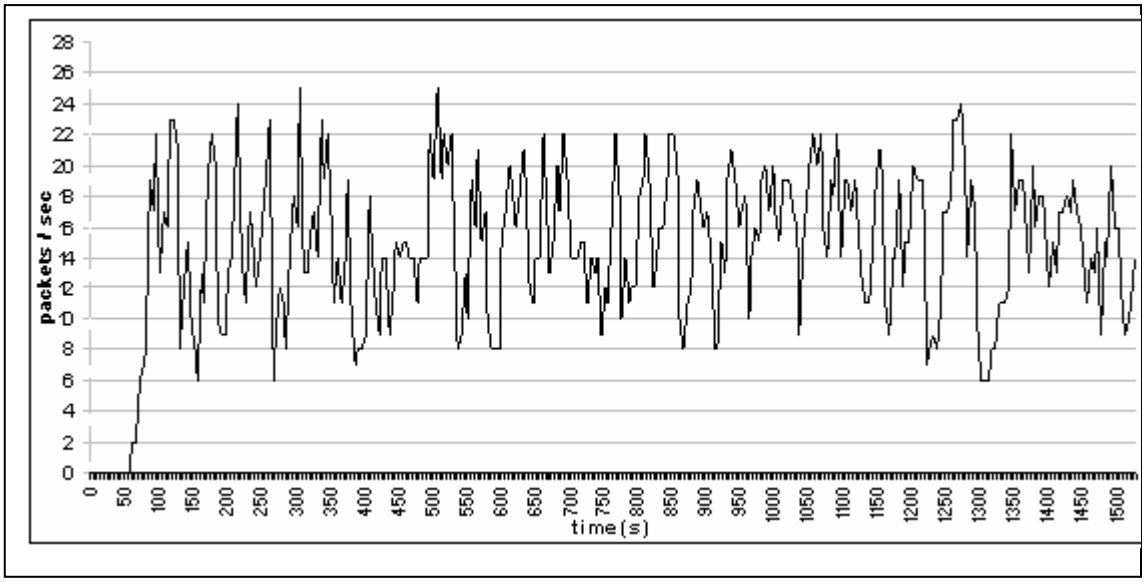

Figura 23. Pacotes recebidos do fluxo UDP utilizando o mecanismo e2e

A figura 24 mostra o acúmulo de pacotes do fluxo UDP (linha fina), fluxo "cross traffic" TCP (linha pontilhada) e o fluxo de referência comparativa TCP da seção 5.1 (linha grossa). No gráfico se pode observar que o UDP envia maior número de pacotes do que o "cross traffic" TCP. O desempenho do "cross traffic" TCP não resultou prejudicado pelo fluxo UDP, o que normalmente desestabiliza o fluxo TCP quando não existe nenhum tipo de controle. Em comparação com o fluxo de referência TCP (seção 5.1), o fluxo TCP deste experimento transmitiu um maior número de pacotes nesse intervalo de tempo. Assim, podemos notar que o mecanismo e2e resultou em um melhor desempenho com um fluxo "cross traffic" UDP.

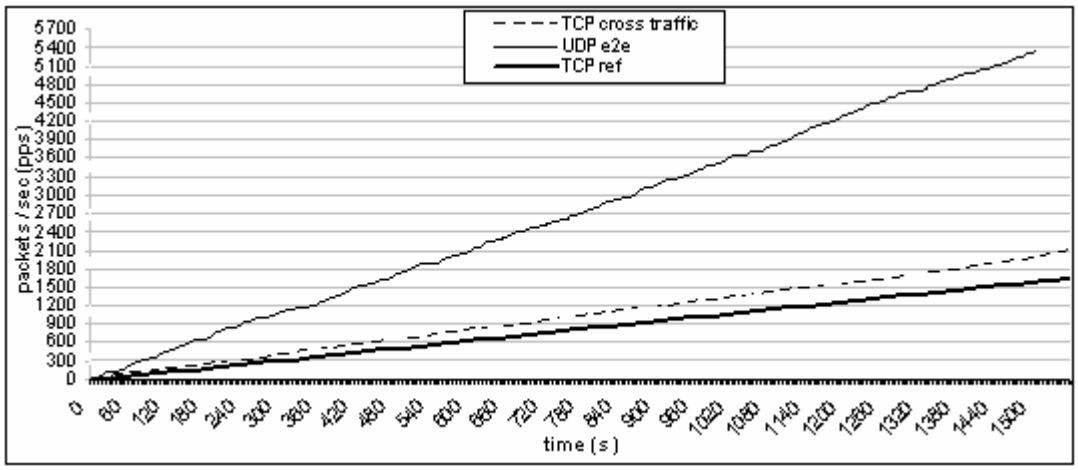

Figura 24. Gráfico cumulativo de pacotes UDP, cross traffic TCP.

Um fluxo UDP controlado pelo mecanismo e2e necessita de um procedimento que aumente gradativamente o número de pacotes transmitidos para diminuir o número de perdas e garantir que o fluxo UDP não congestione ainda mais a rede. Isto pode ser brevemente explicado. Considere a mesma configuração de rede dos testes, mas sem nenhum tipo de congestionamento, as provas ativas determinarão que o número de interrupções (Ticks) para atingir a taxa estimada é igual zero, transmitindo pacotes em cada interrupção. Só depois de um grande número de perdas, a próxima prova ativa fornecerá uma nova taxa segundo o tráfego gerado pelo próprio fluxo.

5.3. Testes e2e com TCP

Nesta experiência se estudará o comportamento do protocolo TCP quando estiver utilizando o mecanismo e2e. Esta combinação 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] [49] (Congestion Agent default no kernel 2.6.17[10]) para imprimir os eventos ocasionados ao utilizar o mecanismo e2e.

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

[10] /proc/sys/net/tcp_congestion_control

5.3.1. Resultados

As medidas foram capturadas cinco vezes como se mostra na tabela 3, obtendo-se uma porcentagem média de 5.12% 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	*3606,53*	*3894,6*	*5,04*	*11,73*	*3695,2*	*5,12*

Tabela 3 Testes com TCP usando o mecanismo e2e

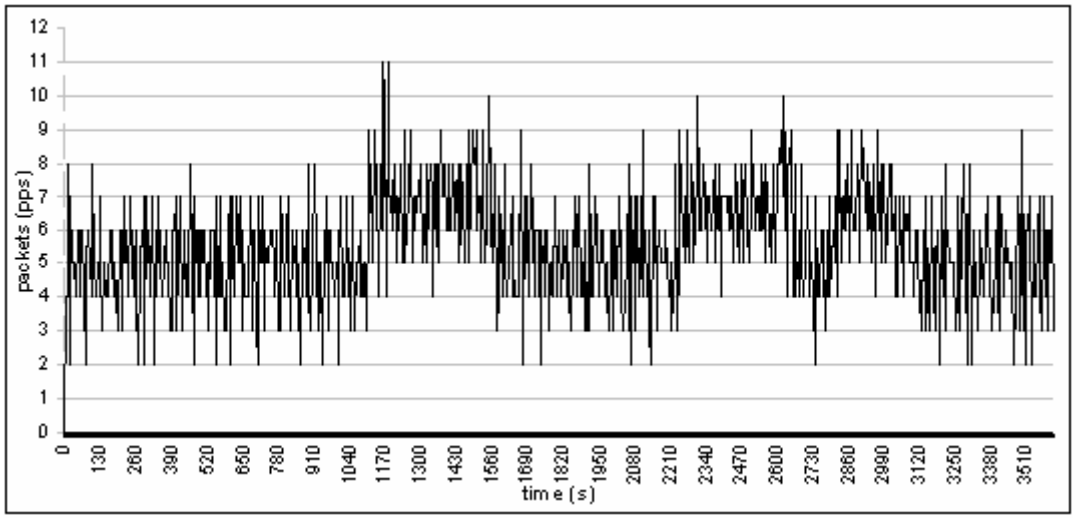

Figura 25. Fluxo TCP utilizando o mecanismo e2e

Nos testes realizados com o mecanismo e2e foi necessário limitar o valor de número de "ticks" máximo para coexistir com o tráfego TCP quando as variações do atraso eram muito altas e freqüentes, configurado para 70ms. 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 acúmulo de pacotes do Fluxo TCP usando o mecanismo e2e (linha fina) e o acúmulo de pacotes típico do TCP de referência comparativa da seção 5.1. Observa-se no gráfico que o acúmulo 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 acumulados que o fluxo de referência TCP (seção 5.1), indicando menor congestionamento e um desempenho similar.

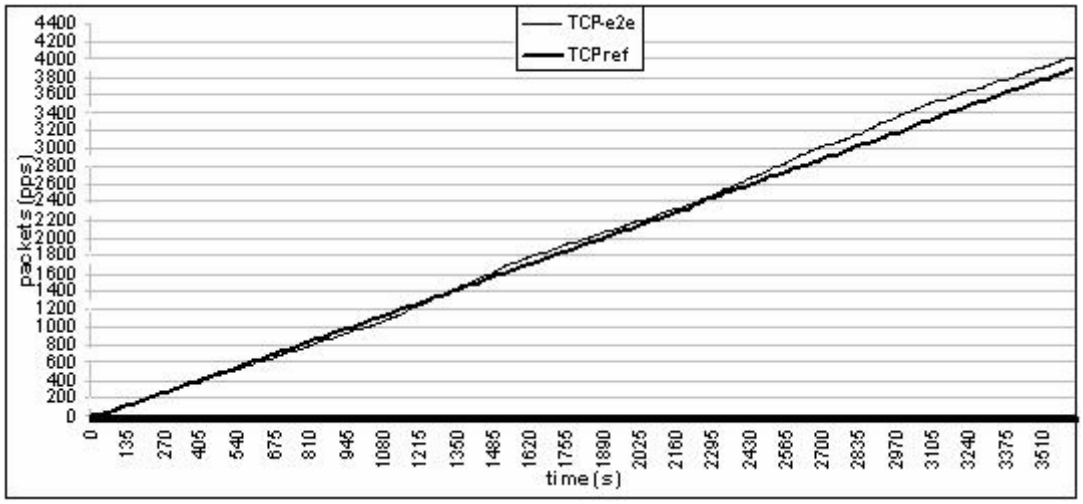

Figura 26. Gráfico acumulativo de pacotes do TCP usando o mecanismo e2e

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

Neste teste será demonstrado que uma rede congestionada com fluxos TCP utilizando o mecanismo e2e deve oferecer 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.

5.4.1. Resultados

As medidas foram obtidas 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% em relação á porcentagem do teste 1, teste de comparação (sem usar o mecanismo e2e). 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*	*5,41*	*12,58*	*3931,75*	*4,61*

Tabela 4 Testes com dois fluxos TCP usando e2e

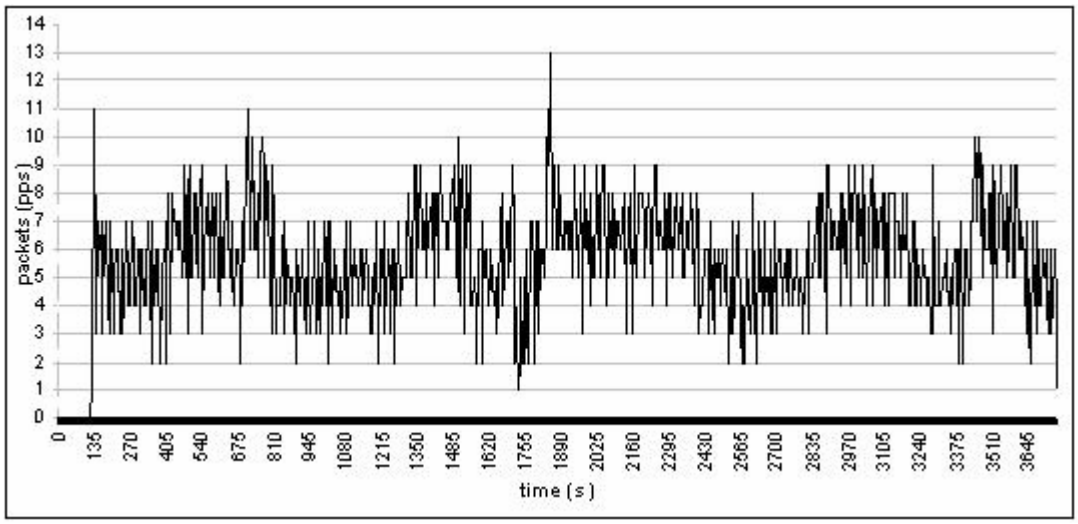

Figura 27. TCP-e2e em uma rede congestionada com dois fluxos e2e

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 usada neste teste. Neste gráfico, inicia-se a transmissão com um número alto de pacotes quando a rede não esta congestionada, continuando com uma redução do número de pacotes quando entra outro fluxo e finalmente incrementando quando o enlace está novamente livre de congestionamento.

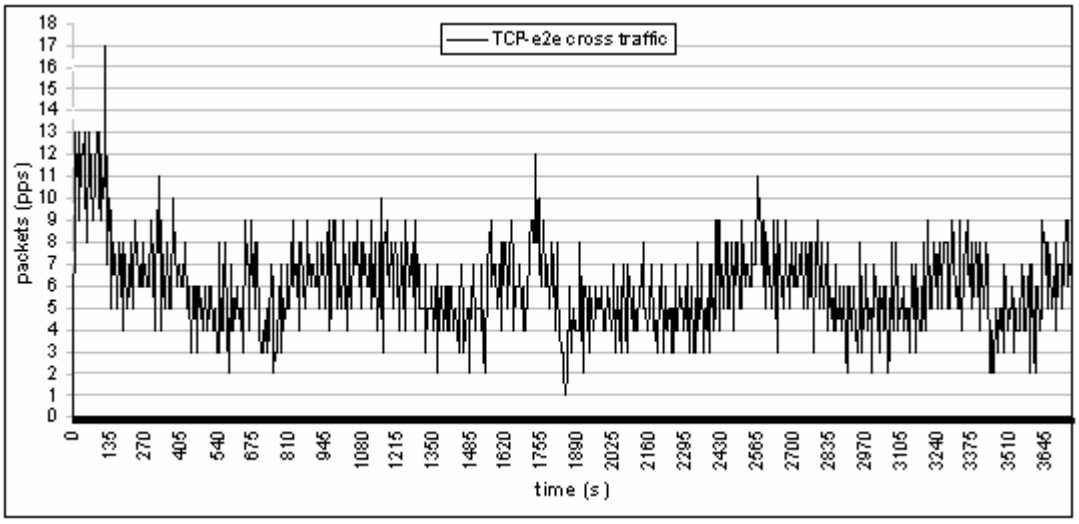

Figura 28. Fluxo e2e cross traffic

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

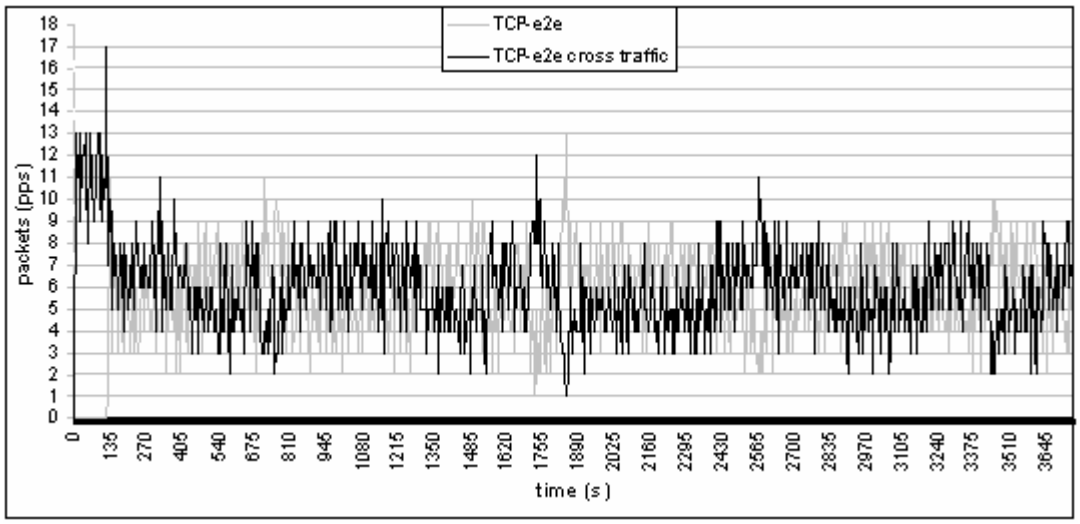

Figura 29. Dois fluxos tcp-e2e compartilhando banda

A figura 30 mostra o acúmulo de pacotes dos fluxos TCP representado com linha fina, o "cross traffic" em linha pontilhada, junto com o acúmulo de pacotes típico do TCP de referência da seção 5.1, representado com linha grossa. 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.

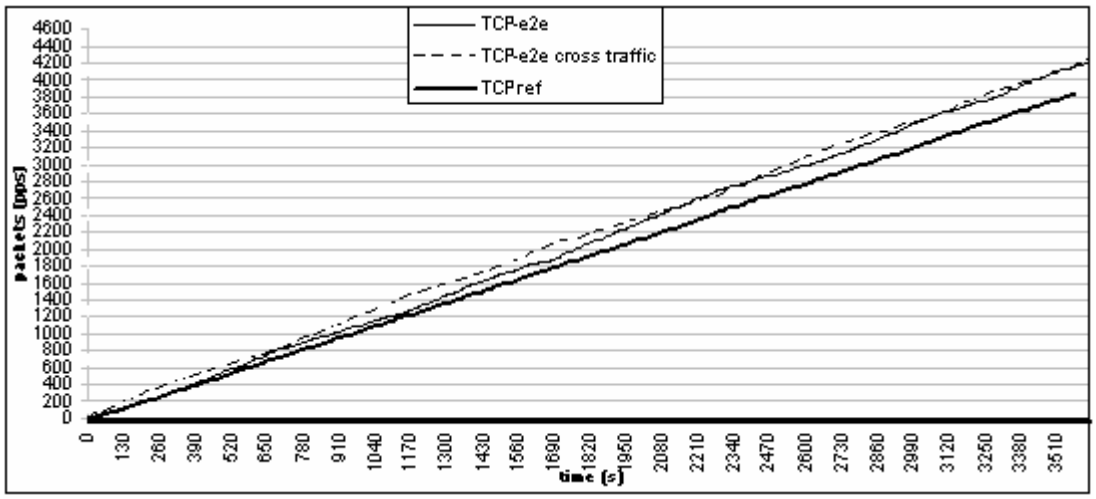

Figura 30. Gráfico cumulativo do TCP usando o mecanismo e2e

5.5. 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" e um TCP utilizando o mecanismo e2e. O e2e-e2e significa dois fluxos TCP usando o mecanismo e2e.

	Packets	Size (MBytes)	BW (kbps)	Received (pkt)	% Lost
tcp-tcp	3945,75	4,938	11,48	3586,75	9,09
tcp-e2e	4002,5	5,144	11,98	3801,5	5,02
e2e-e2e	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:

- **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%**.
- **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%.**

Optimizations	**TCP-e2e**	**E2E-E2E**
Packet Lost	4,06%	4,48%
Rate	4,36%	9,54%
Bytes received	4,17%	9,61%

Tabela 6 Testes com TCP

6. CONCLUSÕES

Este capítulo concluirá o trabalho de dissertação fornecendo as contribuições do mecanismo e2e, sendo finalmente descritos os possíveis trabalhos futuros como continuação deste desenvolvimento.

6.1. Contribuições

Preliminarmente, o mecanismo e2e pode ser definido como uma ferramenta desenvolvida de forma experimental, projetada para variar a taxa de transmissão de protocolos de transporte, em uma determinada configuração da conexão fornecida pela aplicação aos módulos de "kernel" que conformam o mecanismo, através do envio constante de pacotes de prova para medir a largura de banda disponível e desta forma controlar o fluxo de pacotes de acordo com as estimativas realizadas.

Particularmente, o mecanismo e2e oferece uma alternativa de controle de fluxo com uma técnica de gerência de filas, utilizando trens de pares de pacotes para calcular o atraso e estimar a taxa que é variada em cada fila associado com um endereço IP destino, porta e protocolo.

Mediante esta ferramenta foi possível obter resultados de variação da taxa de transmissão ao nível de "kernel", nas camadas mais baixas da arquitetura de rede, mesmo 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. Desta forma, foi observada uma forma de melhorar o "throughput" com um agente externo sem alterações do código fonte do TCP. No caso do UDP, se oferece uma alternativa de controle do fluxo, minimizando as perdas de pacotes que são produzidas quando nenhum tipo de controle é utilizado.

Os resultados obtidos nos testes demonstram que, enquanto o TCP aumenta a quantidade de pacotes no enlace, o mecanismo e2e captura valores de atraso altos, minimizando a taxa de transmissão. Se o atraso for muito alto e este valor for muito freqüente, significa que o TCP pode congestionar 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 atraso pode ser muito baixo e fazer com que o mecanismo transmita mais pacotes. Com relação à fase "Congestion Avoidance", a transmissão de pacotes pode ser mais constante quando o atraso varia com menor freqüência.

O mecanismo e2e pode ser utilizado para realizar o controle do fluxo do protocolo UDP, necessitando de um mecanismo de transmissão gradativa de pacotes, para atingir a largura de banda alcançável e minimizar o número de pacotes descartados, causados pela saturação do próprio fluxo. Isto permite não desestabilizar a rede com um tráfego muito alto, tornando o UDP TCP-Friendly [39].

Observa-se que o comportamento convencional do TCP é congestionar a rede para obter medidas e determinar a largura de banda que deve ser aplicada, num mecanismo indireto. Esta técnica incrementa gradativamente o número de pacotes, gerando perdas. Porém, um mecanismo baseado em taxa em um enlace sendo usado por fluxos TCP, deve ser igualmente agressivo, também tentando obter a taxa máxima do enlace de forma a obter uma largura de banda equivalente. Para que o mecanismo e2e coexista com uma rede com TCP, o valor de taxa deve ser variado para que aja de um jeito que possa coexistir e oferecer vantagens quando comparado com o TCP.

Uma rede com fluxos TCP, utilizando o mecanismo e2e, pode ter seu desempenho melhorado e apresentar diminuição na perda de pacotes. Mas esta seria uma rede utópica porque ao entrar um fluxo TCP que não use o mecanismo e2e, o esquema se desestabilizaria.

Na presença de múltiplas interfaces em uma única máquina, ao enviar os pacotes por múltiplas interfaces ativas, se observou que os pacotes sempre chegavam fora de ordem, o que causava instabilidade no TCP. A diferença do mecanismo e2e quando comparado com os protocolos de transporte baseados em taxa como o "tcp-pacing" ou o "DCCP", é a realização de cálculos de estimativas da largura de banda, baseado em provas ativas,

variando a taxa de transmissão aplicada na camada de enlace e realizando o controle na camada de rede. Esta forma de variação permite controlar o tamanho das rajadas nas filas da placa de rede permitindo que qualquer protocolo de transporte o use de forma transparente, por estar implementado nas camadas mais baixas da arquitetura de rede.

A principal contribuição deste trabalho foi o desenvolvimento de um mecanismo de controle do tráfego que permite variar a taxa de transmissão fim a fim. Este mecanismo espaça regularmente os pacotes de cada fluxo, realizando a moldagem do fluxo para não exceder a capacidade medida da rede. O mecanismo ainda varia dinamicamente o tamanho das rajadas e pode ser aplicado a qualquer protocolo de transporte. A captura de informações de largura de banda alcançável é feita por meio de técnicas de provas ativas.

6.2. Trabalhos Futuros

Podem ser sugeridos trabalhos futuros como a instalação do mecanismo e2e na camada de transporte, mediante a programação de funções de "sockets", descartando o uso de "netfilter" e "netlink". Isto poderia dar início a novos protocolos de transporte para fins específicos. Por outro lado, as funções do mecanismo e2e poderiam ser adaptadas ao protocolo UDP para fornecer uma nova forma de controle de congestionamento.

Também, a implementação de outros algoritmos para testar provas ativas ("path chirp", "packet train", etc.) pode servir para comparar qual fornece uma melhor precisão nos cálculos das estimativas.

Uma futura evolução do mecanismo e2e poderia ser uma nova técnica de "QoS", dando controle aos parâmetros envolvidos tal como "burst" e latência, de forma dinâmica dependendo da largura de banda alcançável.

Como continuação do mecanismo de envio de pacotes por múltiplas interfaces de rede do mecanismo e2e, seria necessário desenvolver um algoritmo que permita reordenar os pacotes quando estes são recebidos no receptor, antes de sua entrega à camada de transporte.

Com relação aos cálculos de estimativas do mecanismo e2e, o método utilizado para medir a largura de banda disponível, foi baseado em medidas do atraso, sem considerar o efeito de valores do mesmo quando são negativos. Para um trabalho futuro a taxa de transmissão pode ser enfocada nos cálculos do "erro do atraso", analisando as variações positivas e negativas, comparando a eficiência alcançada com a técnica do atraso utilizada neste trabalho.

7. REFERÊNCIAS BIBLIOGRÁFICAS

1. AGGARWAL, A.; SAVAGE, S.; ANDERSON, T. **Understanding the performance of TCP pacing.** IEEE INFOCOM 2000 Conference on Computer Communications, 2000.

2. ALLMAN, M. **TCP congestion control.** IETF, RFC-2581, 1999.

3. BANERJEE, S.; AGRAWALA, A.K. **Estimating available capacity of a network connection**. In: IEEE International Conference on Networks. Singapore, 2000.

4. 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.

5. CHIMENTO, P. F. **Standard Token Bucket Terminology**, Technical documentation, 2000.

6. CORBET, J. **Porting drivers to the 2.5 kernel.** Linux Symposium, 2003

7. DOVROLIS, C.; HAO, J. **Source level IP packet bursts: causes and effects**, Internet Measurement Conference, 2003

8. DOVROLIS, C.; RAMANATHAN, P. **Packet dispersion techniques and a capacity estimation methodology.** Part of this paper has previously appeared in an Infocom, 2001.

9. EXPERIMENTAL COMPUTER SYSTEMS, Lab. **Linux Networking Kernel.** University of New York , 2003. (Technical Documentation)

10. FLOYD, S.; FALL, K. **Promoting the use end-to-end congestion control in the internet.** IEEE ACM Transactions on Networking,1999.

11. FREE SOFTWARE FOUNDATION, Inc. **The Linux Kernel API.** EBook, 2006.

12. GORTMAKER, P. **Linux ethernet-howto**, HOWTO, 2000.

13. GRAF, T.; MAXWELL, G. **Linux advanced routing & traffic control**, HOWTO, 2006.

14. GUFFENS, V. **Path of a packet in the Linux kernel.** Kansas University, Course Summer 2005.

15. HOU, J., **Data link layer.** University of Illinois. Departament of the Computer Science, 2005. (Lectures) .

16. HOU, J. **Network Devices**. University of Illinois. Departament of the Computer Science, 2005. (Lectures)

17. HU, N.; STEENKISTE P. **Estimating available bandwidth using packet pair probing**, IEEE JSAC Special Issue in Internet and WWW Measurement, Mapping, and Modeling, 2003. [36]

18. HU, N.; STEENKISTE, P. **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.

19. HUBERT, B. **Linux advanced routing and traffic control**, HOWTO, 2006.

20. JIN, G.; YANG, G. **Network characterization service (NCS).** In: 10th IEEE Symposium on High Performance Distributed Computing, 2001.

21. KOHLER, E.; HANDLEY, M. **Designing DCCP: congestion control without reliability.** *ACM SIGCOMM, 2006*.

22. KUNG, L., **A Walk-Through of Netfilter**", Lectures, 2004. http://lion.cs.uiuc.edu/courses /cs397hou/lectures/Inside_Netfilter.pdf.

23. LAI, K.; BAKER, M. **Measuring link bandwidths using deterministic model of packet delay.** SIGCOMM, 2000. [43]

24. MAGALHAES, L. C. S. **A transport layer approach to host mobility.** University of Illinois. Faculty of the Department of Computer Science, 2005. (Tese Ph. D. in Computer Science).

25. MAGALHAES, L C. S.; KRAVETS, R. **MMTP: multimedia multiplexed transport protocol**. The First Workshop on Data Communications in Latin America and Caribbean (SIGCOMM-LA 2001), San José, Costa Rica. Suplemment to Computer Communication Review. New York: The Association for Computer Machinary, 2001. v. 31 p. 220-243.

26. MATHIS, M.. **A framework for defining empirical bulk transfer capacity metrics.** IETF RFC-3148, julho 2001.

27. MAYA, F.; MAGALHAES, L. **Kernel mechanism for end-to-end traffic shaping**", 5th International Information and Telecommunication Technologies Symposium, 2006.

28. MELANDER, B.; BJOKMAN, M. **Regression based available bandwidth measurements.** IEEE Network, 2003.

29. **Netfilter**: http://www.netfilter.org/, 2006.

30. PADHYE, J.; KUROSE, J. **A model based TCP-friendly rate control protocol.** NOSSDAV, 1999.

31. PAXSON, V. **End-to-end internet packet dynamics**, IEEE/ACM Transactions on Networking, 1997.

32. POSTEL J. **Internet Control Message Protocol.** IETF RFC-792, Setembro de 1981.

33. TAKANO, R. **Realtime burstiness measurement**", 2006. http://www.hpcc.jp /pfldnet2006 /paper /s5_03. [26]

34. POSTEL, J. **Transmission control protocol.** IETF, RFC 793, Setempbro de 1981.

35. PRASAD, R.S.; MURRAY M., **Bandwidth estimation: metrics, measurement techniques, and tools**. IEEE Network Magazine. 2003.

36. RADHAKRIHNAMN, R. **Linux - advanced networking overview, Ver. 1**", http://qos.ittc.ku.edu/howto/index.html, 1999.

37. RIBEIRO, V. J.; RIEDI, R. H. **Pathchirp: efficient available bandwidth estimation for network paths.** Passive and Active Measurement Workshop, 2003. [4]

38. RADHAKRIHNAMN, R. **Linux - advanced networking overview, Ver. 1**, http://qos.ittc.ku.edu/howto/index.html, 1999. [27]

39. SISALEM, D.; SCHULZRINNE, H. **The loss-delay based adjustment algorithm: a TCP-friendly adaptation scheme.** NOSSDAV, 1998. [39]

40. SMITH, M.; BISHOP, S. **Flow control in the Linux network stack**, TACS, ESOP, 2002.

41. SOARES, L. C. **Redes de computadores das LANs, MANs e WANs às redes ATM.** São Paulo, Editora Campus, 2 ed. 1997.

42. STRAUSS, J.; KATABI, D., KAASHOEK, F. **A measurement study of available bandwidth estimation tools**. ACM SIGCOMM Internet Measurement Workshop, 2003.

43. SULLIVAN, B. **Remembering the net crash of "88"**, MSNBC, 1998.

44. TEIXEIRA GUERRA, FABIO. **Protocolos de Transporte Para Redes de Alta Velocidade: Um Estudo Comparativo,** Trabalho de dissertação de mestrado, UFF, 2006.

45. TIRUMALA, A.; OIN, F.; DUNGAN, J. **Measuring end-to-end bandwidth with iperf using web100**, PAM, 2005.

46. UC BERKELEY. **The ns Manual (formerly ns Notes and Documentation),** 2007.

47. VIGER, F. **Active Probing with ICMP Packets.** University of Melbourne. Internship in the Department of Electrical and Electronic Engineering, 2003. [24]

48. VISWESWARAIAH, V.; HEIDEMANN, J. **Rate Based Pacing for TCP**, preliminary DRAFT, 1997.

49. YEE-TING, L.; LEITH, D.; SHORTEN, R. N. **Experimental evaluation of TCP protocols for high-speed networks**", Terena Conference, 2005.

8. ANEXOS

8.1. 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);
}
```

8.2. Siglas e Abreviaturas

Abreviaturas	Descrição
API	*Application Program Interface*
BTC	*Bulk-Transport-Capacity*
CWnd	*Congestion Window*
DCCP	*Datagram Congestion Control Protocol*
CA	*Congestion Avoidance*
CBQ	*Class-Based Queues*
CC	*Congestion Control*
CCID	*Congestion Control Identifiers*
EI	*Exponential Increase*
FCFS	*First Come First Serve*
FIFO	*Fisrt In, First Out*
Fs	*Flight Size*
FSS	*Full-Sized Segment*
IW	*Initial Window*
LW	*Loss Window*
MMTP	*Multimidia Multiplexed Transport Protocol*
MSS	*Maximum Segment Size*
MTU	*Maximum Transmission Unit*
NF_e2e	*Netfilter_e2e*
NL_e2e	Netlink_e2e
PPTD	*Packet Pair/Train Dispersion*
QoS	*Quality of Service*
QDisc	*Queue discipline*
PTR	*Packet Transmission Rate*
RED	*Random Early Drop*
RTT	*Round Trip Time*
RMSS	*Receiver Maximum Segment Size*
RSVP	*Resource ReSerVation Protocol*
RW	*Restart Window*

RWnd	*Receiver Window*
SFQ	*Stochastic Fairness Queueing*
SCH_e2e	Nscheduler_e2e
SK_BUFF	*Socket Buffer*
SLoPS	*Self-Loading Periodic Streams*
SMSS	*Sender Maximum Segment Size*
TBF	*Token Bucket Filter*
TCP	*Transport Control Protocol*
TFTP	*Trivial File Transfer Protocol*
ToPP	*Train Of Packet Pair*
ToS	*Type of Service*
TTL	*Time-To-Live*
UDP	*User Datagram Protocol*
VPS .	*Variable Packet Size*