Controlo de elementos de rede para suporte de Qualidade de Serviço: reconfiguração

Luis Sousa, Nelson Santiago, Rui L. Aguiar

Resumo – O problema de fornecimento de qualidade de serviço em redes de telecomunicações cobre múltiplos aspectos, um dos quais o suporte específico fornecido pelos equipamentos. Este artigo analisa o comportamento de equipamentos CISCO e Linux em ambientes de grande mobilidade. São identificados aspectos a ter em conta aquando da reconfiguração de uma estrutura DiffServ, de forma a minimizar o tempo que lhe está associado nessas situações. Verifica-se que a escolha apropriada da árvore de QoS tem forte impacto no desempenho final dos dispositivos.

Abstract – Providing Quality of Service in telecommunication networks covers multiple aspects, including the specific support provided by the equipments. This paper analyses the behavior of CISCO equipments and Linux-based routers in high mobility environments. We identify aspects relevant to the reconfiguration of differentiated services structures, in order to minimize the associated reconfiguration time in these high mobility environments. We verified that the choice of QoS tree has a major impact in the final performance of the devices.

I. INTRODUÇÃO

Dado o desenvolvimento da Internet, existe uma necessidade de compreensão das tecnologias que a suportam. Neste sentido, os aspectos de Qualidade de Serviço (QoS) na Internet aparecem como uma das áreas mais relevantes por dois aspectos: i) QoS é uma área de impacto crescente, existindo já algumas ofertas comerciais que consideram estes aspectos; ii) o fornecimento de QoS apresenta uma complexidade elevada, cobrindo múltiplos aspectos, desde planeamento, técnicas de escalonamento, políticas de tráfego, etc..

Independentemente de todos os problemas arquitecturais possíveis de estudar neste tipo de redes, um problema fundamental centra-se no controlo dos dispositivos de rede em termos de fornecimento de QoS. Com efeito, o ambiente de referência de serviços diferenciados [1], um dos ambientes centrais de fornecimento de QoS, encontra-se ainda longe de uma implementação clara e universal nos *routers* [2].

Este artigo apresenta secção II de introdução sobre serviços diferenciados. Os serviços diferenciados são um modelo escalável de QoS desenvolvido dentro do IETF. Na secção III são discutidas as técnicas de suporte do ponto de vista do suporte de equipamento, nomeadamente equipamento CISCO e máquinas Linux. Na secção IV são apresentados alguns resultados experimentais, e a secção V discute as conclusões principais.

II. ARQUITECTURA DE SERVIÇOS DIFERENCIADOS

No modelo de QoS desenvolvido no âmbito do grupo DiffServ, o tráfego é classificado ao entrar num domínio, e separado em agregados *(behaviour aggregates*). Na entrada, o tráfego pode ser adicionalmente condicionado.

Cada agregado é diferenciado através de um campo predefinido, o DSCP [3], que é marcado nos primeiros 6 bits do octeto TOS (IPv4) ou Class of Traffic (IPv6). Um agregado não é mais do que uma colecção de pacotes com um mesmo DSCP que atravessam uma ligação numa determinada direcção. Ao percorrer uma rede DiffServ, os pacotes estão sujeitos, em cada um dos nós, a um tratamento de acordo com o *Per Host Behaviour* (PHB) [4,5] associado ao DSCP com que cada pacote foi marcado.

Neste modelo, nem todos os elementos de um domínio têm a obrigatoriedade de suportar todos os blocos que caracterizam o modelo DiffServ (e representado Fig.1). De facto, existe uma clara distinção entre dispositivos situados na fronteira e os que estão no interior de um domínio DiffServ, sendo estes últimos objecto de uma análise mais aprofundada.

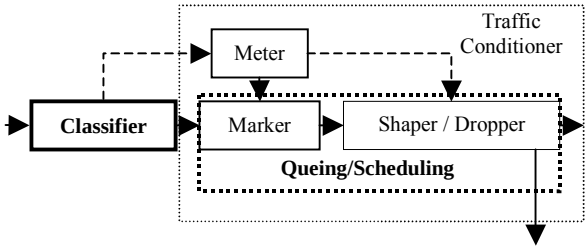

Fig.1 - Blocos DiffServ

Do ponto de vista de desempenho, foram fundamentalmente analisados os blocos *Classifier* e *Queue/Scheduler*, uma vez que nos centramos no comportamento dos dispositivos de rede do ponto de vista de rede de *core*.

III. SUPORTE EM EQUIPAMENTOS

O suporte deste modelo de QoS, de serviços diferenciados, foi analisado em duas plataformas distintas: uma desenvolvida pelo ICA/EPFL baseada num dos blocos do Linux, o *Network Linux Traffic Control* (TC), e outra baseada na plataforma Cisco, correndo a versão do IOS *12.2(13)T*. Em ambos os casos, foi analisada a facilidade de reconfiguração dos mecanismos de classificação, escalonamento de pacotes e de gestão das filas de espera, i.e. quanto tempo era necessário para alterar uma árvore de QoS (criar nós, remover nós, alterar nós, etc..) Como ambiente de referência foi utilizado o conjunto de serviços desenvolvido pelo projecto Moby Dick [6].

A. Cisco

Existe um conjunto vasto de componentes específicas a suportar o modelo DiffServ nas plataformas CISCO [7]: *Modular Quality of Service CLI, Class-Based Packet Marking, Traffic Policing, Class-Based Shaping, Class Based Weighted Fair Queuing* (*CBWFQ), DiffServ Compliant Weighted Random Early Detection, Enhanced "show policy-map interface" Command Enhancements for Class-based Accounting, etc...*

No entanto, a interface com o utilizador é uniforme, e da responsabilidade do módulo de *Command Line Interface* (CLI) [8]. Este módulo implementa uma abordagem de três níveis no controlo de QoS:

1. definição de **classes de tráfego** (*traffic classes)*, agrupamentos de pacotes, o que incluiu a configuração dos elementos classificadores.
2. criação de uma ou mais **políticas de serviço** (*service policies)* associadas às classes de tráfego, i.e. definir o processamento a atribuir à classe;
3. associação das políticas de serviço à respectiva interface.

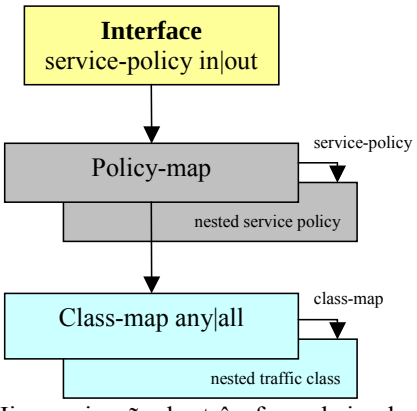

Fig.2 - Hierarquização das três fases de implementação DiffServ em Cisco

Com base nesta estrutura podem ser implementadas múltiplas árvores de QoS. No entanto, existem restrições quanto ao tipo de processamento que pode ser implementado. As diferentes classes de serviço são geridas por disciplinas de escalonamento bem definidas, e dentro de cada classe existem mecanismos de gestão de filas de espera fundamentalmente ortogonais a estas disciplinas de escalonamento. Um exemplo típico de árvores de QoS dentro deste ambiente encontra-se na Fig. 3.

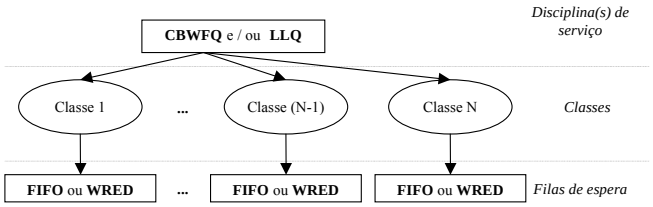

Fig.3 - Estrutura de árvore implementável com *Modular Quality of Service CLI*

De notar que para um mesmo objectivo de fornecimento de serviços (a alto nível) diferentes configurações desta árvore podem apresentar desempenhos semelhantes. A sua reconfiguração (passar de uma árvore para outra) de acordo com mudanças nos utilizadores (mobilidade) pode apresentar desempenhos muito diferentes, no entanto.

B. Linux

O suporte de QoS em Linux pode ser efectuado por diferentes módulos, mas tradicionalmente o bloco responsável pela implementação de DiffServ no Linux é o *Linux Network Traffic Control* (TC) [9].

A estrutura geral de controlo de tráfego em máquinas Linux encontra-se representada na Fig. 4. O TC está direccionado essencialmente para a parte *Egress* (saída) dos dispositivos de rede. Esta particularidade é contrastante com a definição da arquitectura DiffServ, que visa sobretudo tratar o tráfego através da interface *Ingress* (entrada). Foram acrescentadas algumas funcionalidades ao ambiente TC, para se aproximar mais do modelo conceptual DiffServ, funcionalidades estas que permitem um policiamento (básico) no *Ingress*. O código TC é assim constituído essencialmente por quatro componentes:

1. disciplinas de serviço;
2. classes;
3. filtros;
4. policiamento.

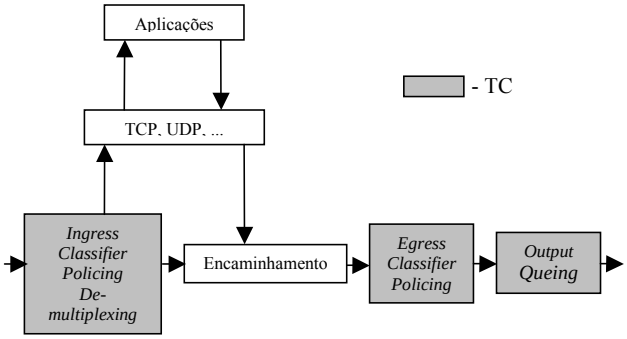

Fig.4 - Controlo de tráfego em Linux

Uma **disciplina de serviço** define a forma como os pacotes contidos numa ou várias filas de espera associadas a uma placa de rede são tratados. Cada disciplina de serviço tem uma ou mais **classes** associadas, com o intuito de poder tratar os agregados de tráfego da forma mais adequada. Ao definir uma classe, está-se a garantir uma série de recursos (mínimos) e formas de tratamento. Quanto à forma de direccionar os pacotes para uma classe, os **filtros** são responsáveis pela diferenciação do tráfego. Consoante a estrutura da disciplina de serviço, existem listas dos filtros que podem ser mantidas pela própria disciplina ou pelas classes nela definidas, em que cada filtro é ordenado pela respectiva prioridade. Quanto à componente de **policiamento**, esta serve essencialmente para garantir que o tráfego não exceda determinados limites associados a cada agregado.

Este último não foi alvo de estudo, tendo em conta que actua, por norma, nos limites de um domínio DiffServ. Os módulos analisados neste trabalho encontram-se conceptualmente representados na Fig. 5. De notar que o seu mapeamento na estrutura o TC apresenta um organização substancialmente diferente (Fig. 6). Este desalinhamento entre os conceitos básicos do TC e o modelo conceptual DiffServ é um dos problemas principais do suporte de QoS em equipamentos Linux.

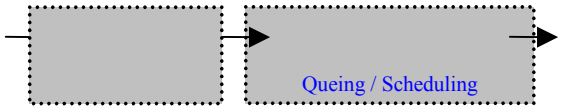

Fig.5 - Modelo Conceptual (blocos estudados)

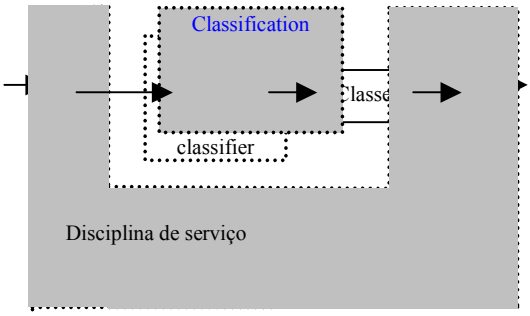

Fig.6 - Elementos TC estudados

Em termos de interface, existem duas interfaces principais para o controlo da TC: tc-interface, e a TC-API [10]. Uma outra possibilidade, SNMP, também foi analisada, mas não fornece correntemente suporte adequado para uma utilização efectiva da TC (Fig. 7).

A primeira interface disponibilizada para configurar o *Linux Network Traffic Control* (TC) foi a tc-interface, disponibilizada pela tc. Este é o programa de configuração de componentes QoS mais utilizado, fazendo parte do pacote *iproute2* [11]. No entanto, apesar da ferramenta 'tc' fornecer um conjunto diversificado de comandos para implementações QoS, não fornece uma interface de programação, pelo que não pode ser directamente utilizada dentro de um programa que necessite estabelecer ou modificar parâmetros QoS. A livraria TC-API providencia uma livraria que permite a uma aplicação *User Space* configurar directamente o módulo TC. As funções contidas nesta livraria geram todas as mensagens *netlink* necessárias, redirecionando-as para o kernel através de *netlink sockets*. Após o envio das configurações para TC, a livraria envia uma mensagem *netlink* para verificar se os valores foram correctamente passados. A forma como a TC-API acede ao bloco TC é em tudo semelhante à forma como a ferramenta tc-interface o faz.

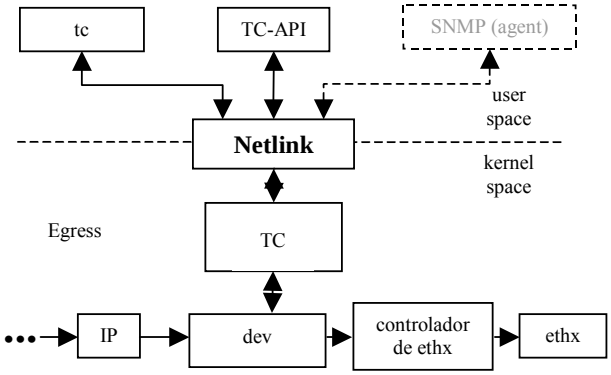

Fig.7 - Comunicação entre o bloco TC e programas no *User Space*

Um exemplo de árvores de QoS implementáveis em máquinas Linux encontra-se na Fig. 8, na próxima secção.

IV. RESULTADOS

Foram feitos testes de configuração e reconfiguração de árvores de QoS tendo como base o conjunto de serviços definidos no projecto Moby Dick. Como indicação, os serviços utilizados encontram-se sumariados na Tabela 1. Durante os testes criaram-se e modificaram-se árvores de QoS dentro dos equipamentos capazes de implementar alguns destes serviços, simulando as condições existentes numa rede com mobilidade de utilizadores, suportando QoS. Os testes mediram fundamentalmente o tempo de resposta dos equipamentos, utilizando uma estratégia indirecta: a análise do comportamento do tráfego á saída do equipamento quando na sua entrada era injectado tráfego de uma forma contínua.

Serviço		Prioridade Relativa	Parâmetros de Serviço	Descrição de utilização típica
Nome	**Classe**			
SIG	AF41	2a	Não-especificado Sinalização	(controlo da rede)
S1	EF	1	Largura de Banda (pico): 32 kbit/s	Serviços tempo-real
S2	AF21	2b	CIR: 256 kbit/s	Transferência de dados prioritária
S3	AF1*	2c	Três precedências de perdas (kbps): AF11 – 64 AF12 – 128 AF13 – 256	Serviços "olímpicos" (melhor-que-BE: streaming, ftp, etc)
S4	BE	3	Largura de Banda (pico): 32 kbit/s	Best Effort (BE)
S5	BE	3	Largura de Banda (pico): 64 kbit/s	Best effort
S6	BE	3	Largura de Banda (pico): 256 kbit/s	Best effort

Tabela 1 – Serviços de referência para os testes utilizados

A. Cisco

A disciplina de serviço utilizada nos equipamentos CISCO foi a CBWFQ. Associada a esta disciplina de serviço foram implementadas classes de configuração que poderão conter dois tipos de fila de espera WRED (com Drop) ou FIFO (sem Drop). Os tempos obtidos na implementação dos mecanismos DiffServ sobre Cisco foram os seguintes:

- Criar uma classe: cerca de 105 ms
- Configurar uma classe (numa *service-policy*): cerca de 220 ms
- Associar uma *service-policy* a uma interface: cerca de 170 ms

Isto significa que a reestruturação de uma árvore de QoS, por questões de suporte a um novo utilizador (num ambiente móvel, p.ex.), pode atingir valores no âmbito de 0.5 seg. Este valor é muito elevado para muitos serviços.

B. Linux

A diferenciação de serviços feita através do bloco TC define os componentes essenciais, tais como disciplinas de serviços (*pfifo_fast*, *FIFO*, *prio*, *TBF*, *DSMark*, *RED*, *GRED*, *CBQ* e *HTB*), classes e filtros (*tcindex* e *u32*). Devido a esta diversidade de componentes apresentados, o TC é muito flexível.

O tempo de configuração e/ou reconfiguração é influenciado pelas diversas combinações de parâmetros. Nestes sistemas, o processamento da árvore de QoS é crítico, e diferentes árvores podem ter desempenhos de processamento muito diferentes, dependendo da forma como a árvore é construída. Um exemplo de tal facto é o parâmetro *maxburst* da disciplina de serviço CBQ, podendo o seu valor representar um incremento temporal para a efectivação da implementação da referida disciplina por um factor múltiplo de aproximadamente 10.

No caso dos filtros, também se fazem sentir diferenças na configuração ao nível do tempo, devido ao tipo de filtro utilizado (tcindex ou u32) e à organização dos mesmos. Os filtros tcindex são mais pesados, em relação aos u32, por causa do maior número de decisões e operações que estes implementam para mapear um pacote. Este pode ser directo, indirecto e algorítmico. Caso os filtros e as classes estejam associados a um mesmo nó, o mapeamento terá que ser obrigatoriamente o directo. Este implica a existência de um filtro por cada classe, aumentando o peso da estrutura e o tempo de reconfiguração associado a cada uma destas, pois sempre que se quiser alterar os seus parâmetros, terão de ser apagados não só os nós "filhos" da classe, como também o filtro que está a ela associada. Apresenta a vantagem de ter um processamento pouco complexo, pois age directamente sobre valores de campos configuráveis encaminhando directamente os pacotes para as classes desejadas. Para o caso em que as classes e os filtros não partilham o mesmo nó, o mapeamento utilizado é então o indirecto. Sempre que um pacote contemplar todas as condições de um filtro, este coloca o valor de retorno *classid* no *skb->tc_index*, de forma a que este seja reconhecido nos nós filhos. Para direccionar os pacotes para as respectivas classes, é então necessário utilizar um dos outros tipos de mapeamento (directo e algorítmico). A escolha natural recai no algorítmico de forma a evitar as limitações associadas ao mapeamento directo. O mapeamento é feito através de um algoritmo "matemático", que atribuiu os pacotes em função do resultado de:

parent : (skb->tc_index & 0xF0) >> 4,

evitando assim a configuração de um filtro por cada classe. Em contrapartida, a complexidade de processamento é aumentada. Como não existe nenhuma ligação directa com as classes, a reconfiguração destas torna-se mais rápida, pois não há necessidade de apagar o filtro. É de salientar que este tipo de mapeamento só é possível com filtros tcindex.

Em relação às filas de espera, o tempo de reconfiguração associado a cada uma varia proporcionalmente com o grau de complexidade dos algoritmos que cada uma implementa.

Conforme já referido, existem duas interfaces de configuração: a tc-interface e a TC-API. Os seus desempenhos e limitações diferem para uma mesma

estrutura. De facto, apesar dos scripts tc e programas TC-API implementarem uma mesma árvore, a utilização de scripts apresenta tempos de implementação superiores.

Com todos estes factores foi elaborada uma estrutura idealizada, representada pela figura 8. Através desta estrutura, foram recolhidos valores dos tempos de reconfiguração desta estrutura numa plataforma Linux, através da interface TC-API (Tabela 2). Verifica-se que os tempos de reconfiguração destas estruturas, muito mais complexas e completas, é muito inferior ao das estruturas implementadas na plataforma Cisco. O tempo de criação de toda a árvore corresponde a alguns milisegundos.

De notar que esta diferença de comportamento reflecte apenas o *overhead* colocado pelos diferentes sistemas de suporte a QoS internos a cada um dos equipamentos. Com tráfego elevado, as máquinas Linux apresentam problemas de desempenho muito maiores do que os valores indicados.

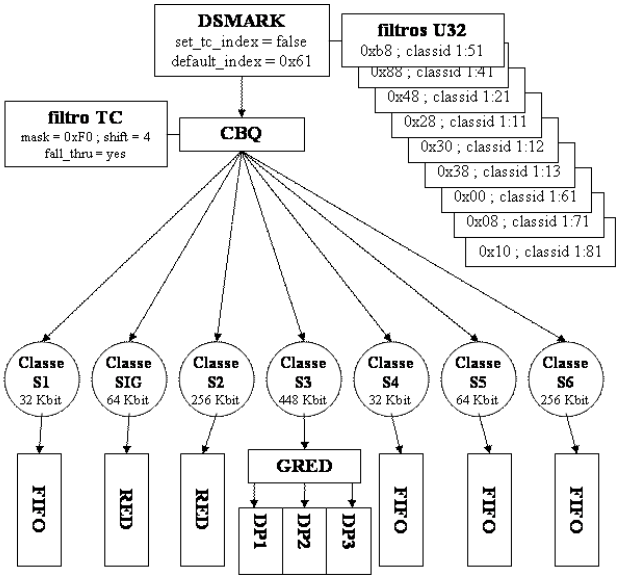

Fig.8 – Estrutura implementada com filtros u32

Estrutura: reconfigura	*Figura 8* tempo
toda a árvore	3,017 ms
só classe S1 (+ bFifo)	185 µs
só classe SIG (+ RED)	676 µs
só classe S3 (+ GRED)	283 µs
só filtro u32	107 µs
só fila de espera bFIFO	106 µs
só fila de espera RED	211 µs
só fila de espera GRED	545 µs

Tabela 2 - Tempos de reconfiguração com TC-API

V. CONCLUSÕES

Este artigo aborda a questão da reconfiguração de equipamentos DiffServ em duas plataformas distintas, do ponto de vista da prestação de serviços a utilizadores móveis.

Na implementação DiffServ sobre plataformas Cisco, verifica-se que apesar das configurações através do CLI demonstrarem que é possível diferenciar o tráfego que passa por dispositivos Cisco, a variedade de mecanismos disponíveis é algo limitada. Além disso os tempos de reconfiguração associados aos comandos CLI poderão ser demasiado elevados num contexto real de mobilidade.

Quanto à implementação da arquitectura DiffServ na plataforma Linux, foi possível verificar que apesar das duas interfaces (tc-interface e TC-API) servirem para implementarem literalmente a mesma árvore no bloco TC, os valores temporais obtidos através da tc-interface são superiores aos da TC-API. Este facto não é de estranhar e vai ao encontro de alguns dos motivos que levaram à criação da TC-API. Para cada linha de comando tc-interface, existe uma inicialização de um novo processo, onde é necessário abrir e fechar uma ligação *Netlink*, que conduz a um aumento do *overhead* associado à criação da árvore. Como as funções TC-API são executadas num mesmo processo e estas utilizam uma mesma ligação *Netlink*, é de esperar que o tempo de processamento desta seja sempre inferior ao de um script tc. É também de notar que a plataforma Linux é mais complexa,mas esta complexidade torna esta plataforma muito flexível na configuração de uma árvore de QoS.

Confrontando as duas plataformas, verifica-se que a plataforma Linux apresenta franca vantagem tanto em termos temporais de reconfiguração, como em termos de implementação de todos os mecanismos disponíveis na arquitectura DiffServ. No entanto as máquinas Linux não foram projectadas para interfaces com tráfego muito elevado.

VI. REFERÊNCIAS

[1] D. Black, Blake, S., Carlson, M., Davies, E., Wang, Z. e W. Weiss, "An Architecture for Differentiated Services", RFC 2475, December 1998.

[2] Luis Sousa e Nelson Santiago, "Controlo de elementos de rede para suporte de Serviços Diferenciados", Relatório final de projecto, 2003.

[3] Nichols, K., Blake, S., Baker, F. e D. Black, "Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers", RFC 2474, December 1998.

[4] F. Baker, W. Weiss, J. Wroclawski. "Assured Forwarding PHB Group", RFC 2597, J. Heinanen, June 1999.

[5] V. Jacobson, K. Nichols, K. Poduri. "An Expedited Forwarding PHB", RFC 2598, June 1999.

[6] Victor Marques, Rui L. Aguiar, Carlos Garcia, Jose Ignacio Moreno, Christophe Beaujean, Eric Melin, Marco Liebsch, "An IP-based QoS architecture for 4G operator scenarios", IEEE Wireless Magazine, June 2003, pp. 54-62.

[7] "Implementing DiffServ for End-to-End Quality of Service", Cisco Technical Manual.

[8] "Modular Quality of Service Command-Line Interface", Cisco Technical Manual.
[9] Werner Almesberger, Jamal Salim, Alexey Kuznetsov, "Differentiated Services on Linux", Internet-Draft draft-almesberger-wajhak-diffserv-linux-01.txt, June 1999.
[10] Gopi Vaddi & Pramodh Malipatna "An API for Linux QoS Support", http://www.ittc.ukans.edu/~pramodh/courses/linux_qos/mainpage.html
[11] Burt Hubert, "Linux Advanced Routing & Traffic Control HOW TO", http://lartc.org/howto/