# Comparação de plataformas para Suporte de Serviços Diferenciados em redes IP

Carlos Parada, Francisco Fontes, Jorge Carapinha
*{est-c-parada, fontes, jorgec}@ptinovacao.pt*
Portugal Telecom INovação, S.A., 3810-106 Aveiro

Solange Lima, Paulo Carvalho
*{solange, pmmc}@di.uminho.pt*
Departamento de Informática, Universidade do Minho, 4710-059 Gualtar, Braga

**Palavras Chave:** *Diffserv*, Qualidade de Serviço (QoS), Condicionamento de Tráfego (TC), *Per Hop Behaviour* (PHB), *Best-Effort* (BE), *Expedited Forwarding* (EF), *Assured Forwarding* (AF), Testes de Desempenho.

***Resumo***
*O principal objectivo deste trabalho é fazer a comparação, de um ponto de vista prático, de duas plataformas para o suporte de Serviços Diferenciados em redes IP: uma desenvolvida pelo ICA/EPFL para plataformas Linux, e a outra baseada em routers Cisco. Para além de comparar e estabelecer as estratégias de configuração em cada plataforma, funcionalidades Diffserv como a classificação, condicionamento de tráfego ou queuing, são testadas, avaliadas e discutidas. Particularmente, são testadas possíveis implementações dos PHBs EF, AFs e BE. As capacidades de oferecer largura de banda garantida, prioritização de tráfego, ou mecanismos adequados de condicionamento de tráfego são avaliadas, assim como o impacto que o tráfego de background tem no atraso dos pacotes de prioridade mais alta. A utilização extra do CPU provocada pelas funcionalidades Diffserv nos routers é também avaliada.*

## 1 Introdução

A arquitectura dos Serviços Diferenciados (*Diffserv*), definida pelo *Internet Engineering Task Force* (IETF), é considerada como uma solução promissora para implementar Qualidade de Serviço (QoS) na Internet de uma forma escalável [1]. Ao colocar a complexidade da rede nos *routers* de fronteira (*edge*), esta arquitectura tem como objectivo simplificar o interior (*core*) da rede. Contrariamente à arquitectura de Serviços Integrados (*IntServ*) [2], o *Diffserv* não precisa de manter informação de estado nem sinalização em cada *router* [3]. Em vez de efectuar reserva de recursos com base em fluxos individuais, o tráfego é agregado num número limitado de Classes de Serviço (CoS) de acordo com os seus requisitos de QoS. O tráfego pertencente a cada classe tem um tratamento adequado e é encaminhado, em cada nó, de acordo com o *Per Hop Behaviour* (PHB) definido. Dois PHBs foram já definidos pelo IETF: o *Expedited Forwarding* (EF) tem como objectivo oferecer atrasos, *jitters* e perdas reduzidas, e um

serviço de largura de banda garantida [4, 5]; o PHB *Assured Forwarding* (AF) oferece um conjunto de quatro classes de serviço com um determinado nível de recursos assegurados (*buffers* e largura de banda) e três *Drop Precedences* (DP) por classe, assegurando a cada classe uma largura de banda mínima garantida [6]. Numa rede *Diffserv* os serviços são fornecidos utilizando mecanismos de classificação e condicionamento de tráfego (TC) na fronteira da rede, e implementando PHBs ao longo do caminho (*path*) [1]. O comportamento que um conjunto de pacotes (tráfego agregado) recebe quando atravessa um domínio DS, foi recentemente definido pelo IETF como *Per-Domain Behaviour* (PDB) [7].

Os *routers Diffserv* devem suportar diferentes funcionalidades (não mutuamente exclusivas), genericamente classificadas como componentes *edge* e *core*, dependendo da sua localização no domínio. Os componentes *edge* localizam-se nas extremidades dos domínios *Diffserv*, sendo as suas principais funções a classificação e o condicionamento do tráfego (TC) de acordo com um conjunto de regras preestabelecias. O TC é fundamental para que os *Service Level Agreements* (SLAs) possam ser estabelecidos, quer entre clientes e domínios DS, quer entre domínios DS. Os componentes *core* localizam-se dentro dos domínios DS e as suas principais funções são a classificação do tráfego previamente agregado e o *queuing* (também presente nos nós *edge*).

O principal objectivo deste trabalho é fazer uma comparação, do ponto de vista prático, entre duas das plataformas *Diffserv* mais utilizadas actualmente: uma desenvolvida pelo ICA/EPFL (*the Institute for Computer Communications and Applications / École Polytechnique Fédérale de Lausanne*) para sistemas *Linux* [8] e a outra baseada na plataforma *Cisco*, correndo as versões do IOS *12.1(1)T* e *12.1(3)T*. As duas plataformas são de âmbito diferente, sendo a implementação ICA/EPFL de domínio público e a implementação da *Cisco* proprietária. Embora existam outras implementações *Diffserv* [9, 10, 11], a escolha recaiu nestas pelo facto de serem plataformas bastante utilizadas, e com as quais o grupo PT também trabalha. A utilização de IP sobre ATM (IP/ATM) tem a ver com o facto de grande parte do *backbone* PT actual assentar em redes ATM.

Através de um vasto conjunto de cenários de teste, este estudo analisa a forma como os Serviços Diferenciados podem ser configurados em cada uma das plataformas, discutindo as suas funcionalidades Os componentes *edge* e *core* são avaliados tendo em conta o comportamento esperado. O seu impacto no desempenho dos *routers* é também medido em termos de utilização do CPU. A implementação dos PHBs é avaliada tendo em conta a alocação de largura de banda e o débito obtido. No *testbed Cisco*, o impacto de diferente tráfego de *background* em classes prioritárias é também medido em termos de atraso (latência). O *testbed Linux*, envolve *routers edge* e *core* que interligam duas redes locais geograficamente localizadas na Portugal Telecom INovação, S.A. (PT IN) e no Instituto de Telecomunicações (IT), em Aveiro. O *testbed Cisco* localiza-se totalmente na PT IN.

Recentes estudos [12, 13, 14, 15] focam aspectos particulares deste trabalho. No entanto, a ampla variedade de testes realizados identificando vantagens, desvantagens e problemas das duas plataformas são uma contribuição adicional nesta área.

Depois de uma breve introdução, as funcionalidades *Diffserv* a avaliar são resumidas na secção 2. A forma como essas funcionalidades são suportadas em *Linux* e *Cisco* descrevem-se na secção 3. O conjunto completo dos testes em *Linux* e *Cisco*, e

respectivos resultados obtidos, são apresentados na secção 4. Também nesta secção são apresentados os aspectos positivos e negativos de cada implementação. As principais conclusões do nosso trabalho encontram-se na secção 5.

# 2 Funcionalidades Diffserv

As funcionalidades *Diffserv* podem ser divididas em componentes *edge* e *core*. Segue-se uma descrição dos seus objectivos antes da discussão sobre a sua implementação nas plataformas ICA/EPFL *Linux* e *Cisco IOS*.

## 2.1 Componente Edge

Um componente *edge* deverá incluir as seguintes funcionalidades: Classificação, Marcação, Medição, Policiamento e *Shaping*. Estas podem ser aplicadas a fluxos individuais de tráfego (geralmente nos *ingress nodes*) ou a tráfego agregado (geralmente nos *ingress* e *egress nodes*). Embora estas funcionalidades sejam conceptualmente distintas, desde o ponto de vista de uma implementação, podem estar agrupadas.

**Classificação –** A classificação ao nível do *edge* consiste na selecção de pacotes baseada, ou em vários campos contidos num pacote IP (*Multi-Field Classification - MF*), ou no campo DSCP (*DS CodePoint*) (*Behaviour Aggregate Classification - BA*). Enquanto a primeira é habitualmente aplicada a fluxos individuais, a segunda aplica-se a tráfego agregado previamente marcado.

**Medição –** Esta funcionalidade mede as propriedades temporais do tráfego previamente classificado. Esta informação, confrontada com os perfis de tráfego especificados no *Traffic Conditioning Agreement* (TCA), é utilizada por outras funções de TC para regular o tráfego conforme este esteja *in-profile* ou *out-of-profile*.

**Marcação –** Esta funcionalidade marca o campo *DS-field* com um valor específico (DSCP) [3]. Com base nas regras do TCA, os pacotes IP são marcados/remarcados de modo a especificar os PHBs a aplicar. Por esse motivo, a arquitectura DS baseia-se fortemente na marcação.

**Policiamento –** O policiamento garante que o tráfego está em conformidade com a um perfil específico, eliminando ou remarcando para prioridades mais baixas o tráfego não conforme. Por exemplo, enquanto os pacotes AF *out-of-profile* podem ser remarcados para aumentar a sua probabilidade de descarte, os pacotes EF excedentes podem ser imediatamente eliminados. Mecanismos comuns de policiamento são definidas em [16, 17, 18].

**Shaping –** Esta funcionalidade regula o tráfego que é submetido a um domínio DS. O *Shaping* pode atrasar os pacotes de forma a ajustar as suas características ao perfil definido[19]. O *Token Bucket* (TB) e o *Leaky Bucket* (LB) são algoritmos habitualmente utilizados para implementar *Shaping* [20].

## 2.2 Componente Core

O componente *core* é mais simples do que o *edge* já que requer um número reduzido de funcionalidades - Classificação BA e *Queuing*. Este aspecto favorece fortemente a escalabilidade da arquitectura *Diffserv*.

**Classificação *Behaviour-Aggregate* -** Este tipo de classificação é substancialmente mais simples do que a classificação MF já que trabalha com tráfego agregado e utiliza apenas um único campo IP – o *DS-Field*. Isto simplifica o processamento no *core*.

**Queuing –** Genericamente, o *queuing* conjuga tarefas de armazenamento, descarte e escalonamento de pacotes. Assim, mecanismos de alocação e gestão de *buffers*, e mecanismos de escalonamento são requisitos obrigatórios para que os PHBs possam ser implementados. Tendo em conta a implementação de várias CoS, o tráfego é normalmente dividido em filas de espera separadas e tratado por disciplinas que seleccionam os pacotes a serem enviados de acordo com a QoS pretendida.

Para além do FIFO, que por si só não permite a diferenciação do tráfego, o *Priority Queuing* (PQ) [21] e o *Fair Queuing* (FQ) [22], são exemplos de abordagens possíveis na implementação de disciplinas de *queuing* para fornecer QoS. A PQ implementa prioridade estrita entre as filas. Isto quer dizer que enquanto as filas de maior prioridade tiverem tráfego para enviar as de mais baixa prioridade não serão servidas. Embora este mecanismo seja particularmente interessante para implementar o PHB EF, pode causar problemas de *starvation* e/ou *burstiness* ao longo do *path* [12, 23]. O FQs tenta colmatar as limitações do PQs, controlando a largura de banda atribuída a cada classe. Esta atribuição pode seguir critérios baseados em modelos proporcionais ou probabilísticos. Variantes do FQ foram utilizadas para implementar o grupo de PHBs AF e em alguns casos EF e AFs [12]. Mais particularmente o CB-WFQ está especialmente orientado para lidar com diferentes CoS [24].

**Controlo de congestão** – O principal objectivo do controlo de congestão é o de prevenir que a congestão aconteça, eliminando pacotes de acordo com um determinado critério. Inicialmente proposto por Jacobson e Floyd [25], o algoritmo RED (*Random Early Detection*) é uma referência nesta funcionalidade. Várias variantes do RED foram já propostas e adoptadas pela comunidade IP. Um estudo comparativo sobre algumas variantes do RED, incluindo WRED, GRED, RIO-C e RIO-DC, é apresentada em [26]. As extensões do RED com suporte multi-nível (vários DPs) são adequadas à implementação dos DPs das classes do PHB AF.

# 3 Implementações Diffserv

## 3.1 Implementação Linux

A implementação *Linux* utilizada neste estudo foi desenvolvida pelo ICA/IPFL (*Institute for Computer Communications and Applications / École Polytechnique Fédérale de Lausanne*) [8]. Esta plataforma engloba as componentes *edge* e *core*, incluindo diferentes tipo de algoritmos de *queuing* e controlo de congestão.

Para estabelecer uma rede *Diffserv* a estratégia de configuração utilizada baseia-se no aninhamento de funções. Este tipo de configuração é efectuada utilizando principalmente o utilitário *tc* e o *ip, route,* ou *ipchains* em alguns casos. Três entidades podem ser configuradas com o *tc*: nós (tendo disciplinas de *queuing* - *qdisc*s - associadas), classificadores (*filters*), e classes, como se exemplifica em [27, 28].

### 3.1.1 Componente Edge

Ao nível do *edge* as principais funcionalidades implementadas são as seguintes:

**Classificação Multi-Field (MF)** e **Marking** são implementadas recorrendo à *qdisc* DSMARK, sendo criadas tipicamente tantas classes quantas as CoS existentes.

**Policiamento** é também implementado utilizando à *qdisc* DSMARK. É possível descartar (*Dropping*) ou remarcar os pacotes que excedam um dado perfil, ou aplicar outras políticas como, por exemplo, remarcar pacotes entre um débito X e Y e eliminar pacotes que excedam Y.

**Shaping** é implementado utilizando a *qdisc* TBF (*Token Bucket Filter*), que segue o algoritmo TB (*Token Bucket*) [20] para limitar os débitos.

### 3.1.2 Componente Core

Ao nível do *core*, as principais funcionalidades implementadas são as seguintes:

**Classificação Behaviour-Aggregate (BA)** é configurada ao nível dos nós, utilizando os filtros sobre o campo *DS-Field* (*filters*).

**Queuing** é configurado principalmente utilizando a *qdisc* CBQ (*Class-Based Queuing*). No entanto, existem outras *qdiscs* disponíveis (para além da FIFO) que implementam diversas políticas e que podem coexistir na mesma CBQ (por exemplo, prioridade estrita ou partilha de largura de banda proporcional). Nesta implementação, a reserva de recursos, a partilha de recursos, a limitação e o isolamento de tráfego podem também ser configurados. Na configuração de uma CBQ é atribuída uma prioridade a cada classe. As classes com maior prioridade serão sempre servidas antes das menos prioritárias. No entanto, as classes não têm que ter necessariamente diferentes prioridades. Nesse caso o algoritmo WRR (*Weighted Round Robin*) é utilizado (mais precisamente o DRR - *Deficit Round Robin*). Por esta razão a designação de PWRR (*Priority Weighted Round Robin*) é por vezes utilizada.

**Controlo de Congestão** é implementado recorrendo às *qdisc*s RED (*Random Early Detection*) e GRED (*Generalized RED*). A primeira é uma implementação baseada no algoritmo RED tradicional, enquanto que a segunda permite a definição de *n* REDs simples, com diferentes *Drop Precedences*. No RED são definidos três parâmetros: Mínimo (*Min*), Máximo (*Max*) e a probabilidade de descarte (DP) no ponto *Max* (Figura 3). No GRED são definidos parâmetros similares, dependendo do grau de diferenciação de tráfego pretendido (e.g. ideal para implementar os 3 DPs das classes AF).

## 3.2 Implementação Cisco

A plataforma *Cisco* inclui como principal equipamento dois *routers* com versões de IOS *12.1(1)T* e *12.1(3)T*, que suportam diferenciação de serviços. Os serviços são

configurados utilizando o sistema tradicional CLI (*Command Line Interface*). A configuração de *Diffserv* em *Cisco* é bastante mais simples que em *Linux*. No entanto, é menos flexível já que não permite o aninhamento de funcionalidades. Apesar disso, os mecanismos oferecidos parecem ser suficientes quer para os componentes *edge* ou *core.* A falta de suporte na utilização do campo DSCP nas versões testadas veio a ser colmatada a partir da versão *12.1(5)T*.

### 3.2.1 Componente Edge

Ao nível do *edge* as principais funcionalidades implementadas são as seguintes:

**Classificação Multi-Field (MF)** e **Marcação** que tal como em *Linux* não são configuradas separadamente. A implementação *Cisco* utiliza o *Policy-Based Routing* (PBR), que permite a classificação de pacotes baseada em múltiplos campos.

**Policiamento** utiliza o CAR (*Committed Access Rate*) [24], podendo o tráfego excedente ser eliminado ou remarcado.

**Shaping** é a funcionalidade implementada pela *Cisco* através do *Generic Traffic Shaping* (GTS). Este mecanismo segue o algoritmo TB (*Token Bucket*).

### 3.2.2 Componente Core

Ao nível do *core* as principais funcionalidades implementadas são as seguintes:

**Classificação Behaviour-Aggregate (BA)** esta funcionalidade encontra-se embebida no *queuing*.

**Queuing** inclui três algoritmos diferentes para além do FIFO. Um é o PQ que implementa um esquema de prioridade estrita. A maior limitação deste algoritmo é que apenas quatro classes podem ser definidas. O segundo algoritmo é o CQ (*Custom Queuing*) em que uma percentagem da largura de banda total é alocada à classe. Finalmente a WFQ (*Weighted Fair Queuing*) divide-se em duas variantes: a FB-WFQ (*Flow-Based Weighted Fair Queuing*) que implementa o escalonamento dinâmico utilizando heurísticas próprias (sem necessidade de configuração). Como este mecanismo não é adequado à implementação de *Diffserv*, o mais interessante é o CB-WFQ (*Class-Based Weighted Fair Queuing*), pois permite a implementação de todos os PHBs que foram definidos até agora. Para cada classe é possível garantir um mínimo de largura de banda (útil para implementar classes AF), sendo o resto da largura de banda distribuída proporcionalmente às reservas efectuadas. É ainda possível definir uma classe que tem prioridade absoluta sobre as restantes. A definição desta classe pode ser crítica já que pode provocar *starvation*. No entanto, com algum cuidado, o CB-WFQ é uma boa solução para implementar EF, AFs e BE simultaneamente em *Cisco*.

**Controlo de Congestão** é um mecanismo que em *Cisco* é implementado utilizando WRED (*Weighted RED*). Esta implementação tal como a GRED em *Linux* baseia-se no algoritmo RED e permite a definição de múltiplos *drop precedences*. O WRED é parametrizado da mesmo forma que o *Linux*, definindo um *Min*, *Max* e a correspondente *drop precedence* no ponto *Max* (equivalente ao mostrado na Figura 3).

# 4 Testes Diffserv

## 4.1 Implementação Linux

Nesta secção a implementação *Diffserv Linux* é configurada, testada e avaliada [29].

### 4.1.1 Equipamentos e ferramentas de teste

Duas ferramentas de geração de tráfego foram utilizadas nos testes: o *mgen* (NRL) para tráfego UDP e o *netperf* para tráfego TCP. Enquanto que com o *mgen* o volume de tráfego gerado precisa de ser especificado, com o *netperf* o débito é regulado pelo TCP, adaptando-se às condições da rede (algoritmo *slow-start*). A análise do tráfego é feita utilizando a ferramenta *tcpdump* (LBNL) e o *PrismLite/RADCOM* (*RAD Group*). Enquanto que *tcpdump* foi utilizado para medições de largura de banda e *debug*, o *PrismLite/RADCOM* permitiu análises estatísticas mais elaboradas, com suporte de uma folha de cálculo.

O equipamento principal utilizado no *testbed* consistiu em *routers* e sistemas terminais *Linux* com *RedHat 6.X* e *Mandrake 7.X*. A versão do *tc* utilizada nestes testes é a versão incluída no pacote *iproute2-000503*, com o *patch ds-8* para o *iproute* e usando o *kernel* 2.3.40.

### 4.1.2 Testbed Linux

A plataforma de testes *Diffserv* em *Linux* ilustra-se na Figura 1. As nuvens representam as redes *Diffserv* localizadas na PT IN e no IT. A rede inclui dois sistemas terminais *Linux*, um emissor e um receptor, e três *routers Linux*. Estes *routers* (dois de *edge* e um de *core*) pertencem a um mesmo domínio *Diffserv*.

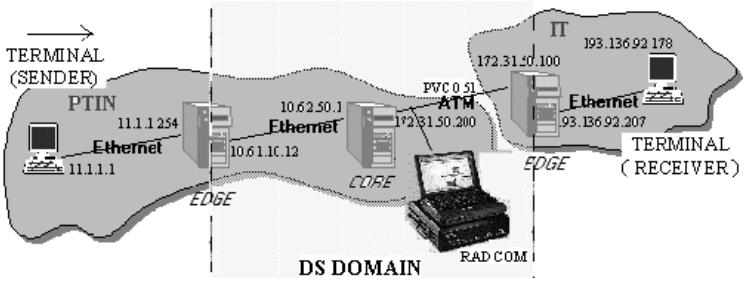



Figura 1 – Testbed utilizado nos testes *Linux*.

Todos os nós encontram-se ligados via *Ethernet* (10Mbps), excepto o *router* de *core* na PT IN e o de *edge* no IT, os quais usam uma conexão ATM (CBR) com *Classical IP over ATM* [30]. Esta ligação ATM permite limitar o débito máximo (*peak rate*) a um valor pré-definido. Esta opção foi utilizada em todos os testes de forma a provocar congestão intencionalmente. Assim, o nível de congestão pode ser facilmente gerido.

As medidas usando o *PrismLite*/*RADCOM* foram efectuadas neste troço ATM, como mostra a Figura 1.

#### 4.1.2.1 Funcionalidades Edge

O objectivo deste conjunto de testes é avaliar a eficácia das acções de condicionamento de tráfego (TC). As ferramentas *mgen* e *tcpdump* foram utilizadas como gerador e analisador de tráfego, respectivamente. Para cada tipo de teste foram considerados diferentes cenários, fazendo variar débitos, tamanho de pacotes, endereços origem/destino e portas. A Tabela 1 mostra de forma sumária os resultados e conclusões obtidos.

| TESTE | OBJECTIVO | RESULTADOS |
|---|---|---|
| ***Marcação (DSMARK)*** | Testar se todos os pacotes são correctamente marcados com base nas características do tráfego | A marcação é correcta para todos os cenários testados |
| ***Policiamento com Dropping (DSMARK drop)*** | Testar se os pacotes são policiados e eliminados (se necessário) correctamente | Resultados não conclusivos (discutidos a seguir) |
| ***Policiamento com Remarcação (DSMARK remark)*** | Testar se os pacotes são policiados e remarcados (se necessário) correctamente | Resultados não conclusivos (discutidos a seguir) |
| ***Shaping (TBF)*** | Testar o *Shaping* do tráfego | Comportamento correcto na generalidade. O *Shaping* do débito é feito correctamente mas o espaçamento entre pacotes (*inter-packet time*) varia entre 1 e 15ms (discutido a seguir) |

Tabela 1 - Sumário dos resultados obtidos nos testes *Linux* (*edge*).

No policiamento com *dropping*, depois de muitas medidas realizadas, não se registou a eliminação de nenhum pacote. A razão para este comportamento pode ser explicada por problemas de implementação na versão utilizada.

No policiamento com remarcação, após testes exaustivos, verificou-se que apenas funcionava correctamente com policiamentos até 800Kbps. As alíneas seguintes resumem o funcionamento correcto e incorrecto da remarcação:

a) Débito de envio a 800Kbps e policiamento feito a 800Kbps: nenhum pacote remarcado: *correcto*
b) Débito de envio a 1000Kbps e policiamento feito a 800Kbps: ~200Kb foram remarcados: *correcto*
c) Débito de envio a 1000Kbps e policiamento feito a 1000Kbps: ~200Kb foram remarcados: *incorrecto*
d) Débito de envio a 1500Kbps e policiamento feito a 2000Kbps: ~700Kb foram remarcados: *incorrecto*

De facto, o *router edge Linux* (*Pentium* II a 400MHz, 64MB RAM) não policiou o tráfego correctamente para débitos superiores a 800Kbps, limitando-o a 800Kbps sempre que se definia um valor superior. Repetindo a mesma experiência usando um outro *router Linux* (*Pentium* II a 266MHz, 64MB RAM), obteve-se um comportamento semelhante, agora limitado a 1200Kbps. Uma razão possível para este comportamento é a falta de precisão do relógio do *Linux*, que não é capaz de lidar com pequenos intervalos de tempo. Este facto também explica a variação do espaçamento dos pacotes (*inter-packet time*) verificado no *Shaping*. Este comportamento já foi antes verificado e explicado em [31]. A variação de tempo introduzida pelo nível ATM, é insignificante

quando comparada com os valores obtidos. Almesberger [32] está a trabalhar numa nova arquitectura de controlo de tráfego para *Linux*.

#### 4.1.2.2 PHBs

Esta secção engloba um conjunto de testes com o objectivo de avaliar o comportamento do tráfego nos *routers* de *core*, verificando se este é diferenciado ou não. Os testes incidem na análise dos PHBs EF, AF e BE, em termos de largura de banda alocada e utilizada, na presença de tráfego UDP e TCP. A *qdisc* CBQ foi utilizada nestes testes. A largura de banda ao nível do ATM foi limitada a 4 e a 6Mbps dependendo dos testes (aproximadamente 3.6 e 5.5 ao nível do IP). Outros parâmetros utilizados foram: *peak rate* EF a 1Mbps; débito mínimo garantido do AF1x e AF4x a 1Mbps e 1.5Mbps, respectivamente.

Os resultados obtidos são muito semelhantes para o tráfego UDP (Figura 2-a) e 2-b)) e TCP (2-c) e d)). As Figuras 2-a) e c) representam o débito obtido quando é usado apenas tráfego EF e BE, enquanto que em b) e d) o tráfego utilizado é EF, AF1x, AF4x e BE.

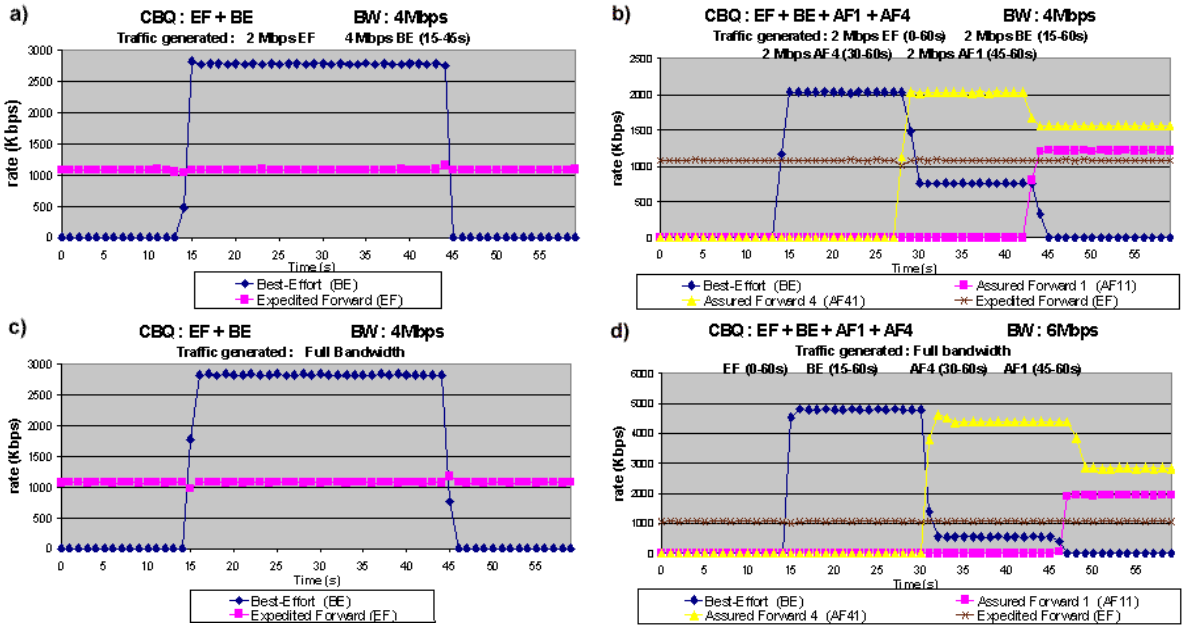



Figura 2 – Débito por PHB para tráfego UDP e TCP (UDP – a) e b); TCP – c) e d)).

A Figura 2 mostra que o tráfego EF é claramente isolado do restante; i.e. o tráfego BE e AF não interferem no débito do EF (embora se note uma pequena variação quando outro tráfego é injectado na rede). Assim, o tráfego EF é independente do restante tráfego e o débito alocado é garantido para o tráfego *in-profile*, i.e. acima de um 1Mbps é eliminado. Para o tráfego AF a largura de banda mínima também foi respeitada. A restante largura de banda (para além de 1 e 1.5Mbps) foi partilhada proporcionalmente à largura de banda garantida. Tal como seria de esperar o BE apenas teve algum débito quando não existia tráfego de maior prioridade para saturar o *link*.

#### 4.1.2.3 Drop Precedence

Esta secção pretende avaliar como é que uma classe com diferentes DPx é afectada pela congestão e por um mecanismo de congestão específico - GRED. Nestes testes, é

analisada uma única classe AF com os seus três DPx, usando tráfego UDP e TCP. Os parâmetros GRED utilizados estão ilustrados na Figura 3.

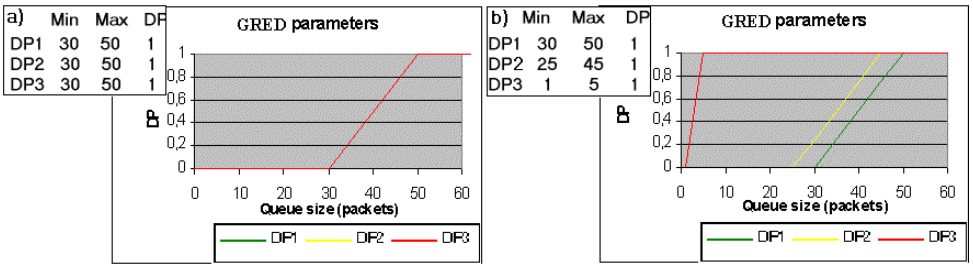



Figura 3 - Parâmetros GRED.

A largura de banda da ligação ATM foi limitada a 2Mbps (aproximadamente 1.8Mbps ao nível do IP). Os resultados obtidos para o tráfego TCP e UDP são apresentados na Figura 4, para os parâmetros GRED definidos.

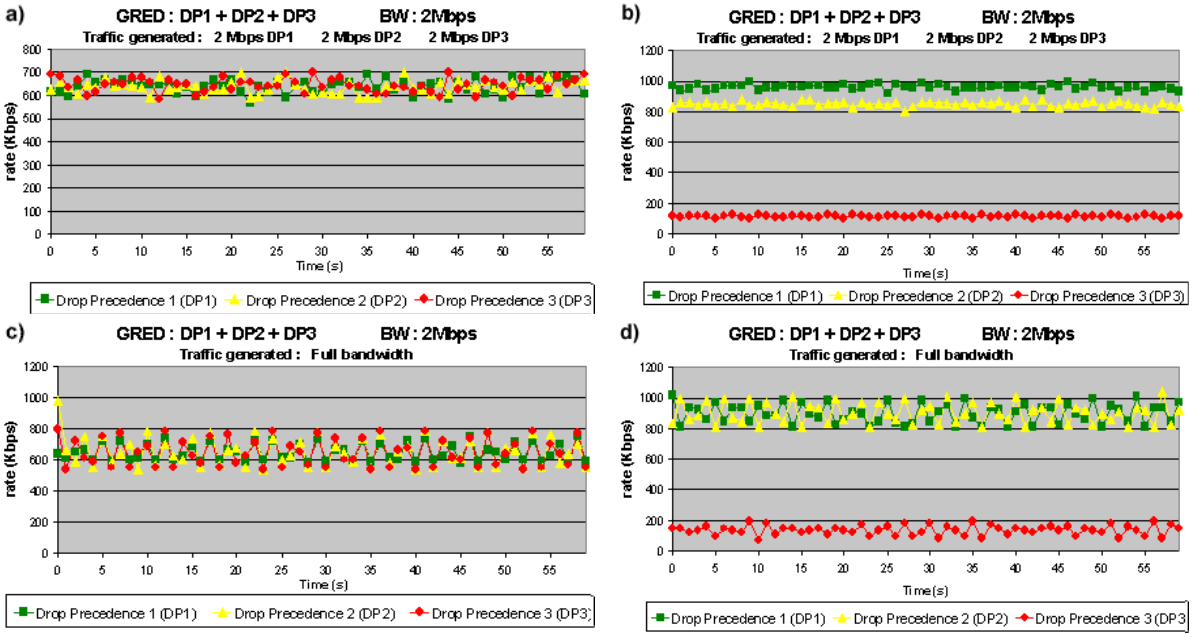



Figura 4 – Débito da classe AF1 (UDP – a) e b); TCP – c) e d)).

A Figura 4 mostra que, para parâmetros GRED iguais, cada AF1x recebe um tratamento idêntico em relação ao débito e à perda de pacotes ocorrida. Para diferentes parâmetros GRED (*Min*, *Max*) o comportamento da classe AF1x é também consistente. Por exemplo, tráfego com DP elevado é drasticamente eliminado, obtendo o AF13 um débito muito reduzido. Finalmente, salienta-se que o débito registado em TCP mostrou uma maior variabilidade que em UDP, consequência do mecanismo da adaptação do TCP.

### 4.1.3 Testes Linux - Sumário

Aspectos positivos:

? flexibilidade na configuração dos mecanismos *Diffserv*, graças à capacidade de aninhamento de funcionalidades;
? número de funcionalidades *Diffserv* implementadas;

? funcionamento correcto da marcação e *Shaping*;
? facilidade de implementação dos PHBs normalizados com os mecanismos de *queuing* e controlo de congestão existentes;
? bom desempenho para tráfego UDP e TCP.

Aspectos negativos:

? complexidade e tamanho das *scripts* de configuração;
? falta de documentação;
? problemas com o policiamento (*Dropping*/Remarcação) e *Shaping* (variabilidade do *jitter*).

## 4.2 Implementação Cisco

Nesta secção a implementação *Diffserv Cisco* é configurada, testada e avaliada [29].

### 4.2.1 Equipamentos e ferramentas de teste

Tal como anteriormente, as ferramentas de geração de tráfego foram: o *mgen* para o tráfego UDP e o *netperf* para o tráfego TCP. Para a análise de tráfego foi utilizado o *tcpdump* e o *PrismLite/RADCOM.* O equipamento *Smartbits* (*Netcom Systems*) foi também utilizado, de forma a resolver os problemas de sincronização. O principal equipamento de testes inclui um *Cisco 7200* com *IOS 12.1(3)T*, um *Cisco 7500* com *IOS 12.1(1)T*, *routers* e sistemas terminais com *Linux RedHat 6.X* e *Mandrake 7.X*.

### 4.2.2 Testbed Cisco

A plataforma de testes utilizada encontra-se ilustrada na Figura 5. A rede inclui dois sistemas terminais *Linux* (emissor e receptor) e três *routers*. O primeiro *router* (*ingress router*) é um *Cisco7500* com *IOS 12.1(1)T* e o *core router* um *Cisco7200* com *IOS 12.1(3)T*. O terceiro *router* (*egress router*) é um *Linux RedHat* 6.X que actua apenas como *router* normal, sem implementar funcionalidades *Diffserv*.

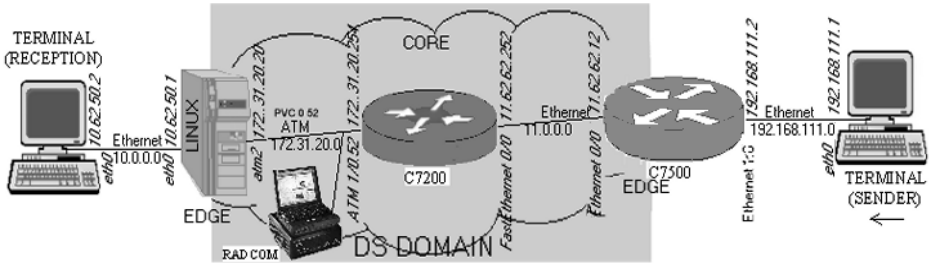



Figura 5 - Testbed utilizado nos testes *Cisco*.

Todos os nós, localizados nas instalações da PT IN, estão ligados usando *Ethernet* a 10Mbps, excepto o *router* de *core* e o *router Linux* que usam uma conexão ATM nrt-VBR com *Classical IP over ATM* [30]. Tal como em *Linux*, esta ligação permite a configuração do seu débito máximo e portanto controlar o grau de congestão na rede.

A medição com o equipamento *PrismLite/RADCOM* é feita nessa ligação ATM como mostra a Figura 5.

#### 4.2.2.1 Funcionalidades Edge

Seguidamente é apresentado um conjunto de testes que permitem avaliar as funcionalidades de condicionamento de tráfego (TC). Tal como em *Linux*, o *mgen* e o *tcpdump* são utilizados para geração e análise de tráfego, respectivamente. A tabela 2 sumaria os resultados obtidos.

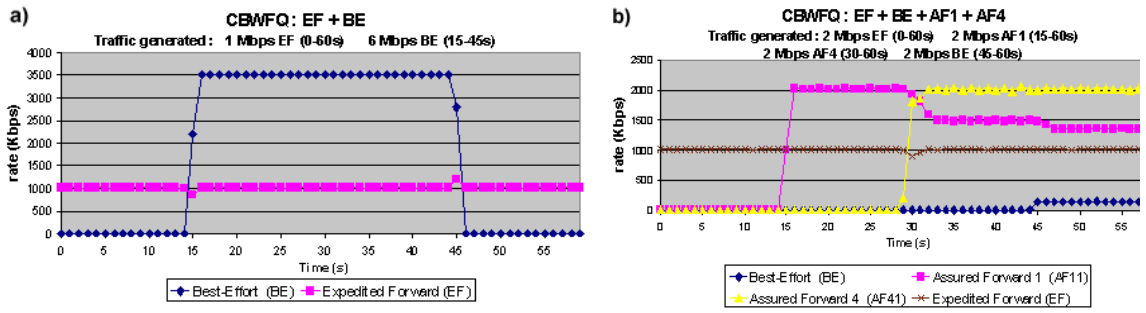
| TESTE | OBJECTIVO | RESULTADOS |
|---|---|---|
| ***Marcação (PBR)*** | Testar se todos os pacotes são correctamente marcados com base nas características do tráfego | Marcação correcta para todos os cenários testados |
| ***Policiamento com Dropping (CAR drop)*** | Testar se os pacotes são policiados e eliminados (se necessário) correctamente | Policiamento com *Dropping* correcto para todos os cenários testados |
| ***Policiamento com remarcação (CAR remark)*** | Testar se os pacotes são policiados e remarcados (se necessário) correctamente | Policiamento com Remarcação correcto para todos os cenários testados |
| ***Shaping (GTS)*** | Testar o *Shaping* do tráfego | Shaping correcto para todos os cenários testados. Quer o débito quer a variação entre pacotes (*inter-packet time*) é correcta. |

Tabela 2 - Sumário dos resultados obtidos nos testes *Cisco* (*edge*).

#### 4.2.2.2 PHBs

Tal como em *Linux*, o objectivo destes testes é avaliar a implementação dos PHBs EF, AFs e BE, em termos de largura de banda alocada e utilizada, na presença de tráfego UDP e TCP, separadamente. Nestes testes a ligação ATM foi limitada a 5Mbps ao nível do ATM (removendo o *overhead* ATM fica aproximadamente em 4.5Mbps ao nível do IP). Os parâmetros de tráfego definidos são: um débito máximo de 1Mbps para o EF; 1Mbps e 1.5Mbps para o AF1x e AF4x, respectivamente. O mecanismo de *queuing* utilizado foi o CB-WFQ.

Os principais resultados destes testes mostram-se na Figura 6. A Figura 6-d) é a única em que os fluxos de tráfego são todos iniciados em t = 0. Nos testes TCP apenas uma conexão é aberta para cada tipo de tráfego.



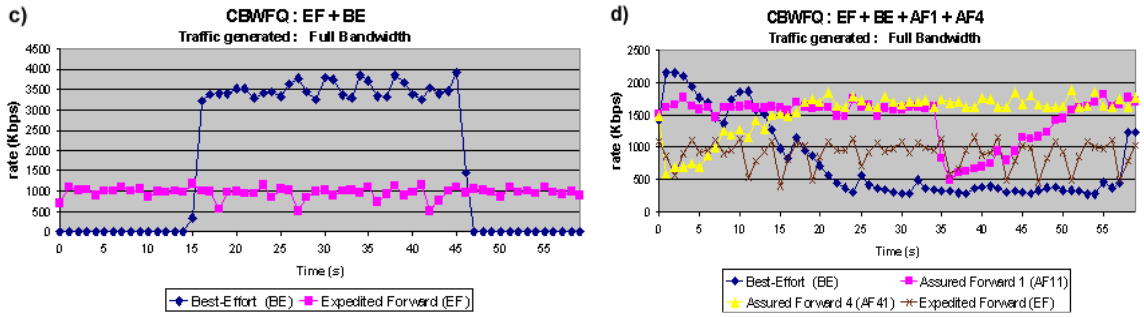



Figura 6 – Débito por PHB para tráfego UDP e TCP (UDP – a) e b); TCP – c) e d)).

Os resultados obtidos com tráfego UDP e TCP diferem consideravelmente. Os testes realizados com tráfego UDP seguem as nossas expectativas. O débito obtido pelo tráfego EF é constante e limitado pelo valor máximo (*peak rate*). As classes AF recebem a sua largura de banda mínima garantida, sendo a restante dividida proporcionalmente à largura de banda reservada. Tendo em conta que em [1, 3] não se define como é que a restante largura de banda deve ser dividida, esta configuração faz com que o BE partilhe em igualdade de circunstâncias essa largura de banda (por opção nossa e ao contrário do que se fez em *Linux* (ver Figura 6-b)). Para o tráfego TCP a instabilidade nos débitos obtidos é notória. Para além do período inicial de convergência, a Figura 6-d) mostra que, por exemplo, a classe AF1x tem débitos bastante inferiores à largura de banda mínima por mais de 10s. Na nossa opinião este comportamento é o resultado da eliminação de pacotes no *router* e consequente activação do mecanismo de *slow-start* do TCP.

#### 4.2.2.3 Drop Precedence

Nos testes seguintes o comportamento do tráfego AF1 é analisado utilizando o mecanismo de controlo de congestão WRED. O tráfego é gerado como AF11, AF12 e AF13, numa ligação ATM limitada a 2Mbps (1.8Mbps ao nível do IP). Os testes foram efectuados utilizando tráfego UDP e TCP e os resultados obtidos são apresentados na Figura 7-a) e 7-b) respectivamente.

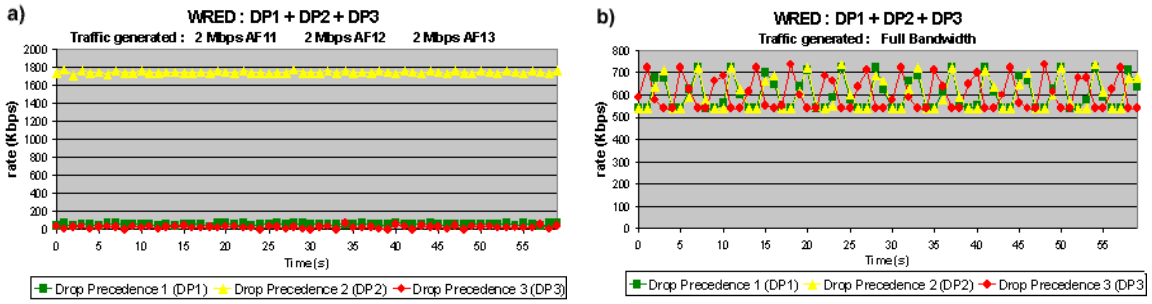



Figura 7 - Débito da classe AF1 (UDP – a); TCP – b)).

Ao contrário do que seria de esperar as medidas obtidas mostram uma invariância dos resultados face aos parâmetros apresentados na Figura 3. Os resultados esperados deveriam ser semelhantes aos obtidos para *Linux*, apresentados na Figura 4. No entanto, para o tráfego UDP, de acordo com as medições efectuadas, um dos AF1x, aleatoriamente, obtinha quase a totalidade da largura de banda, sendo a restante dividida entre os outros dois (Figura 7-a)). Já para o tráfego TCP a capacidade do *link* é sempre

distribuída pelos três AF1x de forma equilibrada (Figura 7-b)). Ainda que este comportamento seja de esperar no teste em que os parâmetros WRED são idênticos (3-a)), para os parâmetros em 3-b) é inesperado. Este comportamento pode ser devido ao uso das interfaces ATM (PVCs) onde não está disponível a configuração do WRED ao nível da interface. Com o objectivo de resolver esta limitação foi criado um mecanismo CB-WFQ e, dentro duma classe deste, foi definido o WRED. Esta solução, embora permitisse prosseguir com os testes, não foi eficaz.

#### 4.2.2.4 Utilização do CPU

Nesta secção é avaliado o impacto que as funcionalidades *Diffserv* têm nos *router*s em termos de utilização do CPU. Neste teste foi enviado para a rede tráfego num total de 8Mbps, equitativamente distribuído pelos CoS EF, AF1x, AF4x e BE (2Mbps cada). O policiamento e o *Shaping* foram efectuados a 1Mbps para cada classe.

As funcionalidades de *edge* em análise são: *Sem Diffserv; Classificação MF (PBR); Classificação e Policiamento com dropping (CAR); Classificação e Policiamento com Remarcação (CAR); Shaping (GTS)*. As funcionalidade de *core* analisadas foram: *Sem Diffserv; Priority Queuing; Custom Queuing; CBWFQ com Tail Drop* e *CBWFQ com WRED*. Os testes foram efectuados utilizando diferentes tamanhos de pacotes, mantendo constante o débito, i.e. aumentando o número de pacotes.

Os gráficos a) e b) da Figura 8 ilustram a utilização do CPU num *edge router* para pacotes de 500 e 50 *bytes*, respectivamente. Os gráficos c) e d) correspondem a medições num *router core*, também com pacotes de tamanho idêntico.

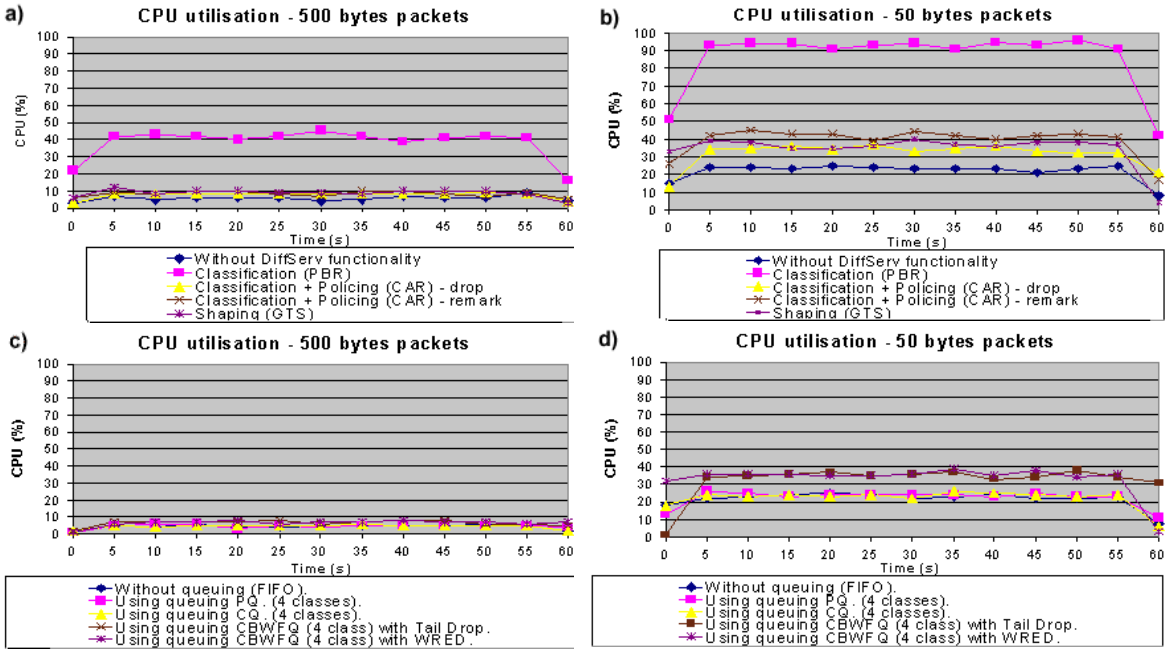



Figura 8 – Utilização do CPU.

Os resultados apresentados mostram que as funcionalidades de *edge* requerem um esforço computacional considerável, quando comparado com a situação *Sem Diffserv*. Isto acontece, especialmente, para pacotes de tamanho reduzido (corresponde a um maior número de pacotes). A funcionalidade de Classificação MF (usando PBR) é de longe a que mais CPU consome. Este comportamento é inesperado já que a

Classificação e Policiamento (usando CAR) faz exactamente o mesmo e ainda remarcação de pacotes, sendo bastante menos pesada. Estes resultados mostram que a implementação do mecanismo PBR deve ser mais ineficiente que o CAR. A Figura 8-a) mostra também um incremento de 10% a 40% na utilização do CPU, enquanto que para pacotes mais pequenos (50 *bytes*) esse valor sobe para 90% (Figura 8-b)). A utilização do CPU quase duplica quando se utiliza a Classificação e Policiamento com remarcação (CAR); este incremento é inferior quando se faz *dropping* em vez de remarcação.

A maior parte das funcionalidade de *core* testadas provocam um incremento na utilização do CPU que pode ser considerado negligenciável, quer para pacotes de tamanho de 500 ou 50 *bytes*. Só a implementação do mecanismo CB-WFQ causa uma utilização adicional de cerca de 10%, para pacotes de pequeno tamanho (50 *bytes* por exemplo).

Na generalidade, os resultados mostram que a redução do tamanho dos pacotes (mantendo o débito constante), requer um extra-processamento nos *routers* DS, especialmente nas funcionalidades de *edge*. Note-se ainda que o débito total ronda os 8Mbps; para débitos mais elevados o incremento da percentagem de utilização do CPU pode ser substancial e limitativo.

#### 4.2.2.5 Atrasos

O principal objectivo destes testes é avaliar a capacidade do *Diffserv* oferecer reduzido atraso a tráfego de alta prioridade. A configuração de rede (*testbed*) inicialmente definido na Figura 5 foi ligeiramente modificada de forma a que uma mesma máquina (*SmartBits*) seja simultaneamente emissora e receptora (evitando problemas de sincronização). O esquema mostra-se na Figura 9. Tendo em conta a sensibilidade desta medição e para aumentar o grau de confiança, os valores aqui apresentados resultam da média de 100 medições.

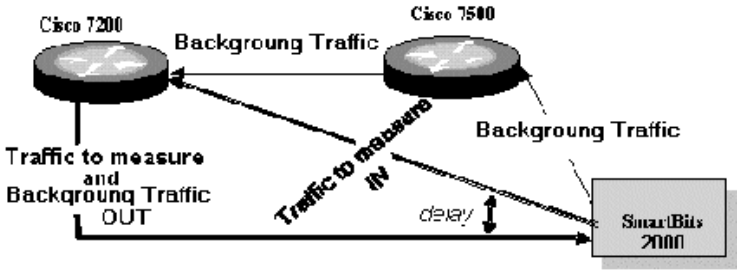



Figura 9 – testbed utilizado nos testes de atraso.

O primeiro teste tem como objectivo mostrar qual é o impacto que a presença funcionalidades *Diffserv* têm no atraso. As métricas usadas, *cut-through latency*[1] e *store-and-forward latency*[2] [33], são apresentadas na Figura[3] 10 a) e b), mostrando os valores obtidos para a Classificação e Marcação de pacotes (PBR) e para o *Queuing* (CB-WFQ). Os atrasos são medidos para diferentes tamanhos de pacotes de tráfego de *background* (128-1152*bytes*), e para diferentes débitos (1.5 e 1.9Mbps).

[1] *Cut-through latency* - mede o atraso desde que o primeiro bit do pacote é enviado pela interface emissora, até que o primeiro bit desse mesmo pacote é recebido pela interface receptora.

[2] *Store and Forward latency* - mede o atraso desde que o último bit do pacote é enviado pela interface emissora, até que o primeiro bit desse mesmo pacote é recebido pela interface receptora.

[3] De facto, as Figuras 10 e 11 representam dados discretos; a utilização de linhas unindo os pontos apenas tem como objectivo melhorar a leitura dos dados.

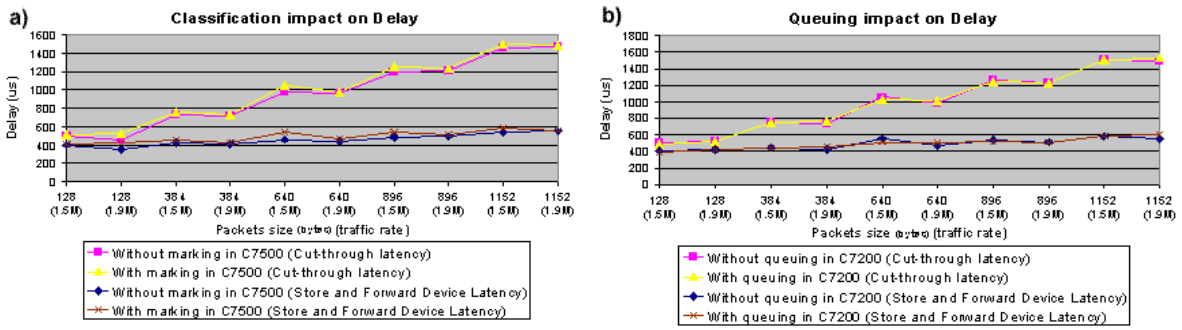



Figura 10 – Medição de atrasos.

Os resultados mostram que o principal incremento no atraso ocorre quando os tamanhos dos pacotes são maiores e quando se usa a métrica *cut-through latency*. Isto acontece porque a métrica *cut-through latency* inclui no atraso o tempo de transmissão do pacote. A Figura 10 mostra também que, no geral, o impacto que as funcionalidades *Diffserv* têm no atraso é desprezável, pelo menos para os débitos considerados.

O segundo conjunto de testes tem como principal objectivo avaliar se podem ser garantidos atrasos reduzidos a PHBs como o EF, perante tráfegos de *background* de prioridade inferior. A ligação ATM foi limitada a 5Mbps (~4.5Mbps em IP). A geração do tráfego é tal que a rede opera sem perdas. O cenário de testes inclui diferentes tipos de tráfego de *background* competindo com um fluxo de tráfego EF. O tráfego de *background* pode ser BE (CBR (*Constant Bit Rate*) ou *on-off* (*bursty*)) ou AF (CBR), para diferentes tamanhos de pacotes e débitos de 1.5 e 1.9Mbps.

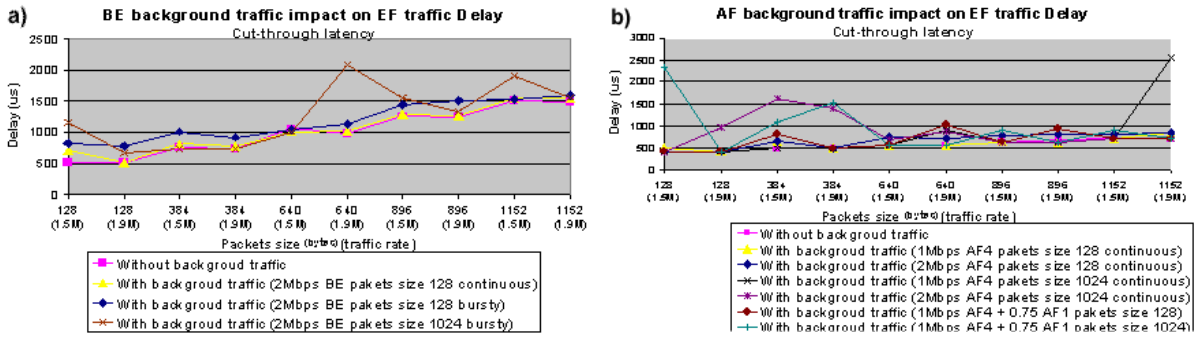



Figura 11- Impacto do tráfego de background no PHB EF.

A Figura 11 mostra que o tráfego EF é ligeiramente atrasado pelo tráfego de *background* BE, principalmente na presença de *bursty traffic* (ON-OFF). O atraso no EF é ainda influenciado pelo tamanho dos pacotes, tal como a Figura 11-a) mostra. Quando o tráfego de *background* está a ser transmitido, mesmo que chegue um pacote EF terá que esperar até que o pacote BE acabe de ser transmitido completamente (relembra-se que o CB-WFQ implementa uma classe LLQ- *low latency queing*). Assim, o aumento do tamanho dos pacotes faz com que a média destes tempos de espera também aumente. Na presença de tráfego *bursty* este efeito acentua-se já que as filas BE têm momentos de grande actividade. A diferença entre ter tráfego de *background* BE ou AF não parece ser significativa.

### 4.2.3 Testes Cisco - Sumário

Aspectos positivos:

? configuração dos mecanismos simples e sucinta, através de CLI;
? número de funcionalidades *Diffserv*;
? mecanismos de TC *(Marking, Policing* e *Shaping)* funcionam correctamente;
? mecanismos de *queuing* funcionam correctamente;
? documentação disponível suficiente;
? plataforma muito utilizada.

Aspectos negativos:

? falta de suporte do campo DSCP, ainda que este problema já esteja resolvido a partir da versão do *Cisco IOS 12.1(5)T*;
? mapeamento do *Diffserv* para camadas inferiores com limitações (e.g. ATM, ISDN);
? problemas com o WRED em interfaces ATM (nas versões usadas);
? mecanismo de *queuing* CB-WFQ tem alguns problemas com tráfego TCP (pode estar relacionado com o tipo de interface em uso);
? menor flexibilidade na implementação de funcionalidade (não tem aninhamentos).

# 5 Conclusões

Neste trabalho foi feita uma comparação entre duas conhecidas plataformas de Serviços Diferenciados: *Linux* e *Cisco IOS*. As duas implementações foram configuradas, testadas e avaliadas, tendo em conta as suas funcionalidades *Diffserv* e os algoritmos suportados, tendo sido criado um *testbed* para cada implementação. Este trabalho também avalia o impacto que as funcionalidades de *edge* e de *core* têm nos *routers Cisco*, bem como o impacto que o tráfego de *background* tem no tráfego EF.

A implementação *Diffserv* desenvolvida pelo ICA/EPFL para *Linux* permite uma grande flexibilidade na configuração de serviços através do aninhamento de funções. Esta flexibilidade pode, no entanto, aumentar a complexidade e dimensão das configurações. O *testbed Cisco* apesar de incluir um menor número de funcionalidades orientadas à diferenciação do tráfego, fornece serviços *Diffserv* mantendo a configuração mais simples.

No que respeita às funcionalidades *Diffserv*, a implementação ICA/EPFL permite uma correcta definição de regras e mecanismos capazes de oferecer uma diferenciação eficiente. Por exemplo, o tráfego EF não é afectado pelo restante tráfego e o AF tem garantidos níveis de largura de banda mínimos. As *drop precedence* para a CoS AF, funcionam como seria de esperar, para UDP e TCP, usando GRED. O Policiamento mostra alguns problemas em lidar com taxas de pico acima de alguns valores (dependentes da máquina). Nestes casos, a remarcação não funciona correctamente para o tráfego excedente, como consequência da granularidade do relógio em sistemas *Linux*.

A plataforma *Cisco* revelou-se muito eficaz na implementação da componente *edge*, sendo o Policiamento e o *Shaping* correctamente implementados. Quanto à componente *core*, menos eficaz, obteve-se um comportamento inesperado para tráfego TCP. O estudo sobre as *drop precedence* não produziu resultados correctos, devido às limitações do mecanismo WRED em interfaces ATM PVCs.

Quanto às medidas de utilização de CPU mostra-se que as funcionalidades de *core* não têm um peso significativo. No entanto, as funcionalidades de *edge* podem ser pesadas dependendo dos mecanismos de classificação e policiamento com remarcação utilizados. O incremento da utilização de CPU é notório quando o número de pacotes aumenta para o mesmo débito (pacotes mais pequenos). O impacto que o tráfego de *background* tem no tráfego EF traduz-se num ligeiro incremento no atraso deste último, principalmente quando o primeiro é *bursty* e tem pacotes mais longos.

Como resultado final, não se pode dizer que uma das plataformas tenha sido melhor ou pior do que a outra, já que ambas têm pontos positivos e negativos. Enquanto a plataforma Linux mostrou melhores desempenhos ao nível do *core*, a Cisco mostrou melhor funcionamento ao nível do *edge*. O resultado desta confrontação pode dizer-se que fica igualado.

No futuro, o trabalho será estendido por forma a estudar o comportamento resultante da concatenação de vários PHBs, ou mais genericamente, dos PDBs.

## Agradecimentos

Este trabalho foi realizado no âmbito do projecto europeu EURESCOM P1006 / DISCMAN (*DIfferentiated Services - network Configuration and MANagement*).

## Referências

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

[2] R. Braden, D.Clark, S. Shenker,, *RFC 1633 – Integrated Services in the Internet Architecture: an Overview*, Junho 1994.

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

[4] V. Jacobson, K. Nichols, K. Poduri, *RFC 2598 - An Expedited Forwarding PHB*, Junho 1999.

[5] B.Davie, A. Charny, F. Baker, J. Boudec, W. Courtney, V. Firoiu, K. Ramakrishnam, Internet Draft - *An Expedited Forwarding PHB*, Abril 2001.

[6] J. Heinanen, F. Baker, W. Weiss, J. Wroclawski, *RFC 2597 - Assured Forwarding PHB Group*, Junho 1999.

[7] K. Nichols, B. Carpenter, *RFC 3086 – Definition of the Differentiated Services Per Domain Behaviors and Rules for their Specification*, Abril 2001.

[8] W. Almesberger, *Differentiated services on Linux*, http://icawww.epfl.ch/linux-Diffserv/, Novembro 1999.

[9] NEC, http://www.nec-cx.com/products/, Setembro 2001.

[10] IBM, http://www.networking.ibm.com/netprod.html, Setembro 2001.

[11] KIDS, http://www.telematik.informatik.uni-karlsruhe.de/forschung/diffserv/KIDS/, Setembro 2001.

[12] T. Ferrari, P. Chimento, *A Measurement-based Analysis of Expedited Forwarding PHB Mechanisms*, TERENA 2000.

[13] S. Radhakrishan, *Internet Protocol QoS Page, Implementations and Evaluation-Performance Measurements, http://qos.ittc.ukans.edu.*

[14] The Joint DANTE/TERENA Task Force TF-TANT, http://www.dante.net/tf-tant/.

[15] Quantum Project (Quality Network Technology for User-Oriented Multi-Media), http://www.dante.net/quantum/.

[16] J. Heinanen, R. Guerin, *RFC 2697 – A Single Rate Three Color Marker*, Setembro 1999.

[17] J. Heinanen, R. Guerin, *RFC 2698 – A Two Rate Three Color Marker*, Setembro 1999.

[18] W. Fang, N. Seddigh, B. Nandy, *RFC 2859 – A Time Sliding Window Three Colour Marker (TSWTCM)*, Junho 2000.

[19] J.S. Turner, *New Directions in Communications (or Which Way to the Information Age)*, IEEE Communications, V. 24, N.10, Outubro 1986, pp 8-15.

[20] C. Partridge, *Gigabit Networking*, Addison-Wesley Professional Computing Series, 1994.

[21] K.L.E, *The Bandwidth Guaranteed Prioritized Queuing and its Implementation Law*, GLOBCOM´97, IEEE V. 3 1997, pp 1445-1449.

[22] A.Demers, S. Keshav, S. Shenkar, *Analysis and Simulation of a Fair Queuing Algorithm, Internet Research and Experience*, V. 1, 1990.

[23] T. Ferrari, G. Pau, C. Raffaelli, *PriorityQueuing Applied to Expedited Forwarding: a Measurement-based Analysis*, 2000.

[24] Cisco Systems (Documentation), *Cisco IOS (TM) Software - QoS Solutions*, 2000.

[25] S. Floyd, V. Jacobson, *Random Early Detection gateways for Congestion Avoidance*, IEEE/ACM Transactions on Networking, V.1 N.4, Agosto 1993, pp 397-413.

[26] R. Makkar, I. Lambadaris, J. Salim, N. Seddigh, B. Nandy, J. Babiarz, *Empirical Study of Buffer Management Scheme for Diffserv Assured Forwarding PHB*, ICCCN2000.

[27] W. Almesberger - ICA/EPFL, *Internet draft: Linux Network Traffic Control - Implementation Overview*, Abril 1999.

[28] IETF, Diferentiated Services on Linux, *Internet draft: draft-almesberger-wajhak-diffserv-linux-01.txt*, Junho, 1999.

[29] C. Parada, S. Lima, J. Carapinha, *Qualidade de Serviço em Redes IP (Serviços Diferenciados)*, Technical Report, Dezembro 2000.

[30] M. Lauback, J. Halpern, *RFC 2225 – Classical IP and ARP over ATM*, Abril 1998.

[31] K.Wehrle, H. Ritter, *Problems of Traffic Shaping in ATM and IP Networks Using Standard End Systems*, ATM2000 Conference, http://atm2000.ccrle.nec.de/.

[32] Werner Almesberger, http://icawww1.epfl.ch/~almesber/.

[33] S. Bradner, *RFC 1242 – Benchmarking Terminology for Network Interconnection Devices*, Junho 1991.