Série VoIP (3) – Roteadores e QoS

J. R. Smolka

Estamos indo bem. Este é o terceiro artigo da série VoIP, onde nosso objetivo é examinar os roteadores, que são máquinas com hardware e software especialmente projetados para máxima eficiência no papel de *gateways*, como sistemas de computação.

A discussão é genérica, e não pretende fazer nenhuma consideração ou comparação entre estratégias de implementação adotadas por este ou aquele fabricante de roteadores.

Embora os roteadores possam implementar uma enorme variedade de funções acessórias (ex.: agentes de gerência, servidores de *proxy* DHCP, etc.) vamos nos limitar a discutir os aspectos arquiteturais que influenciam o desempenho da sua função primordial: receber pacotes em uma interface, verificar por qual interface eles devem ser enviados e remeter os pacotes adiante pelas interfaces selecionadas.

Depois disso, vamos examinar os mecanismos de QoS disponíveis em redes TCP/IP, e entender qual é o papel desempenhado pelos roteadores em cada um deles.

Arquitetura lógica de um roteador:

Como qualquer sistema e computação, um roteador pode ser entendido como uma coleção de processos que trabalham sob o controle de um sistema operacional. Reduzindo ao básico, a estrutura de processos do roteador pode ser descrita como na figura 1.

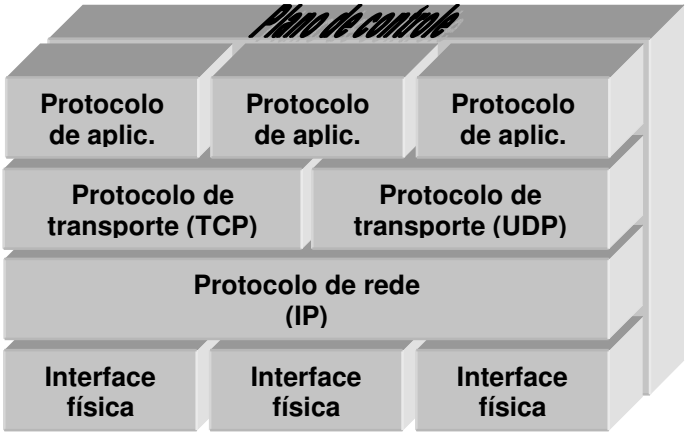

Figura 1 – Arquitetura lógica de um roteador

Os processos responsáveis pelos protocolos de aplicação suportam as funções do plano de controle que necessitem de recursos específicos de comunicação neste nível (ex.: agentes de gerência precisam do protocolo SNMP, *dynamic routing* necessita de algum dos protocolos desta categoria – RIP, OSPF, etc.). Como são protocolos de aplicação, suas mensagens precisam ser mediadas pelos protocolos de transporte (TCP ou UDP).

O plano de controle executa as tarefas de "arrumação da casa", tais como: coleta de estatísticas de utilização de recursos, autenticação de usuários, implementação da interface de comandos (*user shell*) para usuários locais ou remotos, etc.

Embora todas as situações onde ocorram transferências de dados entre os processos operacionais e do plano de controle do roteador afetem, de uma forma ou de outra, a *performance* geral da máquina, a parte realmente crítica para um desempenho satisfatório, é a comunicação entre o processo responsável pelos algoritmos de roteamento (o protocolo IP propriamente dito) e os processos que administram as interfaces físicas do roteador, para encaminhar pacotes em direção aos *links* de transmissão. O modelo de trabalho entre estes processos pode ser entendido, esquematicamente, como na figura 2.

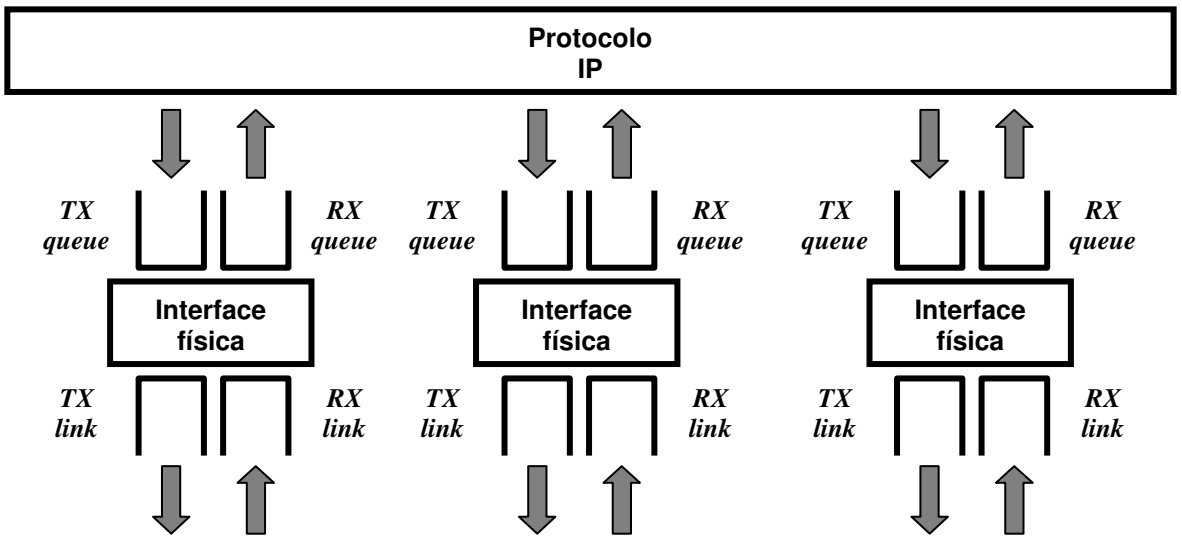

Figura 2 – Tráfego de pacotes entre IP e interfaces físicas

Os processos que controlam as interfaces físicas mantém filas para os pacotes que estão aguardando para ser transmitidos (*TX queue*) e que estão aguardando para ser encaminhados para tratamento pelo processo IP (*RX queue*). Pacotes que estão sendo transmitidos ocupam o enlace de transmissão (*TX link*), e pacotes que estão sendo recebidos ocupam o enlace de recepção (*RX link*)

O desempenho da comunicação entre os processos operacionais (e do plano de controle) do roteador é comandado, primeiro pelos mecanismos de *inter-process communication* que o seu sistema operacional implemente, e segundo pela forma de distribuição destes processos que a arquitetura de hardware permita (multiprocessamento, *switching fabrics*, etc.). Mas, onde provavelmente ocorrerá um desequilíbrio sensível na *performance* é no processo de remover pacotes da *TX queue* e colocá-los no *TX link*.

Pode parecer, à primeira vista, que quanto maior for a capacidade da interface menor será o desequilíbrio de desempenho. Entretanto é melhor lembrar que, se um roteador possui apenas interfaces de alta capacidade, é porque, provavelmente, ele tem um papel

de entroncamento de alto volume de tráfego, e o problema pode acontecer sempre que ocorrer um pico de tráfego destinado a uma única interface. E se o roteador possui apenas uma interface de alta capacidade, o mais provável é que ela seja usada como *uplink* de concentração do tráfego das outras interfaces, e também está sujeita à ocorrência de sobrecarga.

De maneira geral, a capacidade de escoamento de tráfego nas interfaces é o fator individual mais importante para determinar o tempo que um pacote leva para entrar e sair de um roteador, chamado de latência (*latency*). Portanto, vamos olhar com atenção quais são as estratégias de administração das filas de transmissão.

Estratégias para as filas:

As filas são áreas de armazenamento em memória (*buffers*) com tamanho finito, embora configurável – dentro dos limites de disponibilidade física da máquina. Eventualmente a quantidade de pacotes aguardando processamento em uma fila esgota a capacidade de memória alocada (*buffer overflow*), e quaisquer novos pacotes que tentem usar aquela fila serão descartados (*dropped*). Quando isto acontece, o roteador está congestionado (*congested*), e o descarte indiscriminado de novos pacotes na fila congestionada é chamado *tail drop*.

Fora de congestionamento, os pacotes são alocados às filas e aguardam processamento. O tempo de espera de um pacote na fila depende de quantos pacotes já estão na fila, e da capacidade do elemento que retira pacotes da fila (o *TX link*, para a fila de transmissão, e o protocolo IP para a fila de recepção). Como não há como prever quantos pacotes estarão ocupando a fila em um determinado instante, o tempo de espera é variável, e, consequentemente, a latência também será.

Como, mesmo em um ambiente com *headiness index*[1] +5, existem pacotes para os quais queremos minimizar a latência e o risco de descarte em caso de congestionamento (ex.: pacotes dos protocolos de *dynamic routing*), portanto precisamos de algumas estratégias para tratar de forma diferente os desiguais.

Os objetivos são dois: diminuir a probabilidade de ocorrência de congestionamento (*congestion avoidance*), e, caso ele ocorra, minimizar as conseqüências do congestionamento (*congestion control*).

Com a implementação de algoritmos para *congestion avoidance*, queremos evitar a ocorrência de *tail drop*. O algoritmo básico é chamado RED (*random early detection*), e baseia-se no seguinte procedimento:

a) Estabelece-se um limite máximo (*threshold*) para utilização do *buffer*. Enquanto a utilização estiver abaixo disto, nenhuma ação é tomada;

b) Quando a utilização excede o *threshold*, uma proporção definida dos pacotes que entram na fila são descartados, aleatoriamente. A proporção de tráfego descartado

---

[1] Ver definição do *headiness index* no final do primeiro artigo desta série.

pode tornar-se mais agressiva, à medida que a utilização da fila se aproxima dos 100%. Se chegar ao ponto de haver *buffer overflow*, ocorrerá *tail drop*;

c) O processo de descarte continua até que a utilização do *buffer* caia abaixo do *threshold*.

A idéia do RED é "suavizar" picos de tráfego de saída na interface, e dar uma certa previsibilidade da proporção do tráfego que será perdida em caso de congestionamento. Entretanto o descarte atinge, indiscriminadamente, tráfego prioritário ou não. Além disso, considerando que o tráfego usa principalmente TCP como protocolo de transporte, a forma de descarte adotada no RED faz com que todas as janelas, de todas as sessões TCP, sejam percebidas como perdidas e retransmitidas quase simultaneamente. O resultado final é que o roteador fica oscilando entre estados de baixo tráfego (após o descarte) e congestionamento (quando todas as sessões tentam retransmissão quase simultânea).

A alternativa é diferenciar a maneira como será feito o descarte se o *threshold* for atingido. Mantendo a proporção do tráfego total que é descartado, vamos fazer com que o descarte ocorra de forma desigual, afetando menos os fluxos (*flows*) de maior prioridade, e mais os *flows* de menor prioridade.

Mas, então, é necessário alguma forma de designação explícita do grau de importância de um pacote. Existe um byte no *header* IP para designação do TOS (*type of service*), cuja padronização original na RFC 791 previa que os três bits mais significativos servissem para indicar a prioridade do pacote (IP *precedence bits*), os próximos três bit fossem utilizados para definir a classe de serviço desejada (*delay*, *throughput* ou *reliability*), e os últimos dois bits ficassem reservados (*currently unused*). Mais tarde esta padronização foi alterada pela proposta *DiffServ*, que veremos depois.

Quando o descarte de pacotes tiver de ser iniciado, porque o *threshold* de utilização do *buffer* foi atingido, a proporção de pacotes descartados em cada configuração de TOS pode ser diferente, mantendo igual o percentual do tráfego total que será descartado. Em outras palavras, o administrador da rede pode alterar a probabilidade de descarte de tráfegos prioritários em caso de congestionamento. Esta variante do RED é chamada de WRED (*weighted* RED).

Apenas a capacidade de suavizar picos de tráfego e diminuir a probabilidade de ocorrência de *tail drop* não é o suficiente, entretanto. Dado que a ordem de entrada de pacotes com configurações TOS diferentes na fila não pode ser prevista, permanece o problema de garantir que cada *flow class* (identificada por uma configuração TOS) receba uma fração adequada da banda disponível no *TX link*.

O caso mais simples é conhecido como prioridade absoluta de transmissão (*strict priority*). Neste caso, os pacotes pertencentes à *flow class* especificada como *strict priority* "furam" a fila, e são transmitidos antes. Os pacotes das demais *flow classes* são atendidos, desde que não existam pacotes *strict priority* aguardando, em modo *first came, first served*[2] (FCFS).

[2] Também conhecido como *first in, first out* (FIFO).

A próxima alternativa é denominada *fair queueing* (FQ), que tenta garantir que cada *flow class* receba a mesma quantidade de banda do *TX link*. Para isso é utilizado um algoritmo conhecido como "balde de fichas" (*token bucket*), que funciona assim:

a) Uma parte do algoritmo faz a geração periódica de *tokens*. Cada *token* representa o direito de transmitir uma certa quantidade de bits;

b) Cada *flow class* tem o direito de utilizar uma fração definida dos *tokens* gerados. Para que um pacote pertencente uma *flow class* seja transmitido tem de existir *tokens* suficientes disponíveis para esta *flow class*;

c) Se existem *tokens* suficientes, o pacote é transmitido e os *tokens* são removidos do "balde";

d) Se não existirem *tokens* suficientes, então o pacote tem de aguardar na fila até que existam *tokens* para sua *flow class* em quantidade suficiente para que ele seja transmitido.

Também é fácil, desta forma, implementar uma variação do algoritmo, onde a quantidade total de *tokens* gerados não é distribuído de forma igual entre as *flow classes*. Assim, a "fatia" de banda do *TX link* que cada *flow class* pode utilizar pode ser desigual. Esta variante é chamada *weighted fair queueing* (WFQ).

A depender da realidade de tráfego em cada rede, *strict priority*, FQ ou WFQ podem ser suficientes para resolver o problema de repartição de banda do *TX link* entre as *flow classes*. No *core* de grandes redes (corporativas ou a Internet), geralmente a situação é mais complexa, e exige um pouco de tudo isto.

Os roteadores que suportam mecanismos de garantia de *quality of service* (QoS) implementam, além do WRED, uma mescla de *strict priority* com WFQ, chamada *class-based weighted fair queueing* (CBWFQ). Neste modo de operação, existe uma *flow class* com *strict priority*, e as demais *flow classes* são servidas via WFQ.

Um ponto importante a observar é que, embora seja possível implementar diferenças de tratamento entre *flow classes*, dentro da mesma *flow class* o tratamento sempre será FCFS. Assim, o grande trabalho de *traffic engineering* em redes TCP/IP está em adequar a banda total dos *links* físicos ao comportamento observado do tráfego agregado de cada *flow class*.

IntServ e *DiffServ*:

Quando começou a discussão sobre a viabilidade de implementação de mecanismos globais de garantia de QoS em redes TCP/IP, as primeiras propostas convergiram para uma tentativa de fazer *call admission* (*headiness index* –5), chamada *integrated services* (*IntServ*).

Na proposta *IntServ*, para que um novo *flow* pudesse ser inserido na rede, deveria haver previamente uma negociação para reserva explícita de recursos (na prática, banda de

passagem - *bandwidth*) em quantidade adequada à categoria de serviço exigida para aquele *flow*.

Para viabilizar esta negociação, foi proposto um novo protocolo de aplicação, chamado *resource reservation protocol* (RSVP). Através das primitivas RSVP, um *host* pode informar as características do novo *flow* (*traffic specifications* – TSPEC) e solicitar as características de serviço necessárias (*service request specifications* – RSPEC). Conforme o tipo de serviço solicitado, todos os *gateways* na rota fazem a reserva de banda necessária (via *token buckets*). Embora este tipo de mecanismo soe como música para os *bellheads*, ele apresenta alguns problemas, que limitam a sua aplicabilidade em grandes redes:

a) Como todos os hosts na rota tem de confirmar explicitamente a reserva de banda para aplicações críticas, a negociação para admissão do novo *flow* fica complexa e demorada;

b) À medida que o número de *flows* que passam por um *gateway* cresce, o esforço computacional para manter todos eles dentro das características de serviço negociadas cresce de forma não-linear, o que é indesejável;

c) Se o tráfego tem de cruzar a fronteira administrativa entre AS diferentes, é muito difícil garantir que os critérios de reserva de recursos serão aplicados de forma homogênea fim a fim.

Assim, a não ser em pequena escala, o modelo *IntServ* não é muito popular. A alternativa que foi adotada em grande escala na Internet é chamada *DiffServ* (*differentiated services*), descrita nas RFCs 2474 e 2475.

No *DiffServ*, o btye TOS tem a sua função detalhada, com a preocupação de manter compatibilidade com as definições anteriores (afinal, ainda existem por aí equipamentos que implementam apenas a definição original do TOS), e passa a ser denominado *DiffServ field* (DS). Um domínio *DiffServ* (*DiffServ domain*) é um conjunto de roteadores que seguem as mesmas definições de encaminhamento de tráfego, com base no conteúdo do DS dos pacotes.

Os pacotes tem o conteúdo do DS marcado no seu ponto de ingresso no *DiffServ domain*. Logicamente, a definição de onde fica a fronteira do domínio é de responsabilidade dos administradores da rede. A depender da situação, esta fronteira pode estar no próprio *host* do usuário, no equipamento de acesso à rede (tipicamente LAN *switches*), ou no roteador de acesso à rede do provedor. A figura 3 mostra a configuração dos bits do DS.

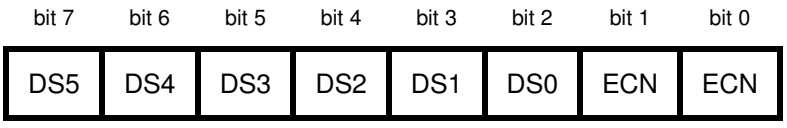

Figura 3 – bits do *DiffServ field* (DS)

Os bits ECN (*explicit congestion notification*) não fazem parte da definição do *DiffServ*, e são reservados. Os bits DS5 a DS0 definem o *DiffServ code point* (DSCP). Cada roteador do *DiffServ domain* adota um comportamento local, chamado PHB (*per-hop behavior*) associado aos DSCPs.

As RFCs 2597 e 2598 definem três tipos de PHB: *expedite forwarding* (EF), *assured forwarding* (AF) e *best-effort* (BE). Só existe um PHB EF, que define um serviço de emulação de circuito (*virtual leased line* – VLL) fim a fim, com baixa perda, baixo *delay*, *jitter* controlado e garantia de banda (cacilda!! Meu *headiness index meter* bateu fora de escala no lado negativo!). Os PHBs AF (são doze no total, combinando características de prioridade de transmissão e probabilidade de perda) servem para criar serviços diferenciados em relação ao serviço provido pelo PHB *default* BE (que significa nenhuma diferenciação do tráfego).

Se vocês estão percebendo alguma ligação entre o que falamos antes (WRED e CBWFQ) e os PHBs do *DiffServ*, isto não é mera coincidência. Aqueles algoritmos são a forma de implementação dos PHBs em cada roteador. A figura 4 mostra as diversas configurações dos DSCPs com os tipos de PHB definidos.

PHB		**DSCP**	
Tipo	**Classe**	**Decimal**	**Binário**
EF	–	46	101110
AF	AF11	10	001010
	AF21	18	010010
	AF31	26	011010
	AF41	34	100010
	AF12	12	001100
	AF22	20	010100
	AF32	28	011100
	AF42	36	100100
	AF13	14	001110
	AF23	22	010110
	AF33	30	011110
	AF43	38	100110
BE	–	0	000000

Figura 4 – PHBs e DSCPs

As classes do PHB AF são designadas na forma AF*xy*, onde *x* indica a prioridade de transmissão (4 = alta, 1 = baixa) e *y* indica a probabilidade de descarte em caso de congestionamento (1 = baixa, 2 = média, 3 = alta).

O comportamento típico de um roteador que implementa *DiffServ* é: fora de congestionamento os pacotes marcados como EF tem serviço VLL garantido, podendo exceder a banda mínima garantida, se necessário, e os demais pacotes são encaminhados em modo FCFS; em congestionamento, entram em vigor os limites de banda mínima definidos para cada classe (EF e AF), sem que a classe EF possa exceder o limite estabelecido, e os pacotes marcados como BE terão a "sobra" da capacidade de transmissão (se houver).

Voltando ao que falamos antes na descrição do algoritmo CBWFQ, o fato de definir uma classe de prioridade absoluta (que normalmente estará associada ao PHB EF) garante apenas o menor *delay* possível. À medida que o tráfego EF que tem de ser encaminhado através de uma interface cresce, a disputa pela banda reservada aumenta, e isto vai refletir no aumento do *jitter*. Para controlar isto, a única maneira é ajustar a banda reservada para o PHB EF naquela interface de acordo com o volume esperado de tráfego nesta classe de serviço. E, se o congestionamento for severo, mesmo com WRED ocorre *tail drop*, e, neste caso, o tráfego EF também vai sofrer.

Multi-Protocol Label Switching (MPLS):

O Segundo maior fator que afeta a latência de um roteador é o tempo necessário para que o protocolo IP decida para qual interface física o pacote deve ser encaminhado, através da inspeção da sua tabela de rotas (buscando o *largest prefix match* do CIDR).

É evidente que, quanto maior for a tabela de rotas, maior será o tempo necessário para a tomada desta decisão, e, consequentemente, maior será a latência.

Os protocolos de *dynamic routing*, tanto para *exterior routing* quanto para *interior routing*, preocupam-se com isto, tentando minimizar o número de entradas na tabela de rotas pelo uso de rotas-sumário (*summary routes*). Mesmo assim, na Internet é comum encontrarmos roteadores cujas tabelas de rotas tem alguns milhares de entradas.

É desejável, então, algum mecanismo que permita simplificar o processo de roteamento de pacotes através de grandes redes, evitando o custo computacional de pesquisa em grandes tabelas de rotas. O método predileto, hoje em dia, para conseguir isto é o MPLS (*multi-protocol label switching*).

O MPLS surgiu como uma proposta proprietária da Cisco Systems, chamada *tag switching*. Posteriormente foi padronizada pelo IETF na RFC 3031. A idéia básica é acelerar o encaminhamento (forwarding) dos pacotes, através do uso de rotas pré-definidas (chamadas *label paths*).

Ao ingressar na rede MPLS, o pacote recebe um *header* MPLS, que pode conter um ou mais *labels*. A identificação no label permite que os roteadores MPLS associem o pacote a uma *forwarding equivalence class* (FEC), e cada FEC está associada a um *label path* na rede. Os protocolos para troca de informações de associação de FECs aos

label paths, e configuração (o termo *bellhead* para isto é provisionamento) dos *label paths* entre os roteadores MPLS são o LDP (*label distribution protocol*) e o RSVP-TE (*resource reservation protocol – traffic engineering extension*).

Hum... meu *headiness index meter* está descendo... descendo... estabilizou no –4. Um fato sobre o MPLS precisa ser bem compreendido: ele não foi desenvolvido com o objetivo principal de acelerar o roteamento em redes IP. O objetivo real era posicionar uma rede de roteadores como alternativa viável ao *frame-relay* e ao ATM para a construção de redes multi-serviço (leia-se: capazes de transportar voz e dados como entidades separadas). E, especialmente com os acréscimos do PWE3[3] (*pseudo-wire emulation edge-to-edge*, também conhecido como *martini drafts*), esta vocação fica mais que evidente.

Tenho minhas dúvidas se o desempenho de roteamento no *core* de uma rede TCP/IP de grande porte será significativamente melhor com o uso do MPLS do que, por exemplo, uma política agressiva de sumarização de rotas via *dynamic routing* associada ao uso de roteadores que armazenem a routing table em CAM (*content-addressable memory*). Querendo ou não, o processo de adicionar e remover *labels* aos pacotes IP vai exigir algum aumento de latência nos pontos onde isto ocorra. Se o aumento da latência for maior que a diminuição no *delay* para atravessar a rede MPLS (comparado com o *delay* do roteamento IP convencional) então não há ganho.

O que o MPLS tem de realmente bom, especialmente para os provedores de serviço, são duas coisas: a possibilidade de configurar VPNs (*virtual private networks*), que neste caso atendem pelo acrônimo VRF (*virtual routing/forwarding facility*); e mecanismos de reconvergência em caso de falha tão rápidos quanto nas redes SDH/*Sonet* – FRR (*fast reroute*).

Enfim... Uma vez que o MPLS está aí para ficar, vamos ver como ele funciona. Os roteadores na borda da rede MPLS, chamados LER (*label edge router*) ou PE (*provider edge*), são os responsáveis pela inserção e remoção[4] dos *labels* associados aos pacotes IP. Os roteadores PE são os que "carregam o piano" em uma rede MPLS, porque precisam inspecionar a tabela de rotas local e definir a FEC que será inserida no *label*.

O core de uma rede MPLS é formada por roteadores LSR (*label switching routers*), também conhecidos como roteadores P (*provider*), cuja única função é fazer o *forwarding* dos pacotes, com base na FEC informada no *label* e em uma tabela simples de associação entre FECs e *label paths*.

Outro conceito errado sobre MPLS é que ele é de grande ajuda (ou até indispensável) para a garantia de QoS em redes TCP/IP. Então vamos à próxima seção, para vermos como o QoS é tratado em redes MPLS.

---

[3] www.ietf.org/html.charters/pwe3-charter.html

[4] Por razões de performance, a remoção do *label* ocorre, na verdade, no penúltimo roteador do *label path*, em um processo denominado PHP (*penultimate hop popping*).

MPLS e QoS:

No label MPLS existem três bits EXP (*experimental*), que são utilizados para associar informação de prioridade de *forwarding* aos pacotes. O fato de serem chamados "*experimental*" dá uma boa idéia do que a proposta original do MPLS dizia sobre QoS.

A forma mais comum para trasladar a informação dos DSCPs *DiffServ* dos pacotes para o MPLS é repetir, nos bits EXP do *label*, os bits DS5, DS4 e DS3 do byte DS no *header* IP.

Não tenho nenhuma informação objetiva sobre isso, mas não vejo porque um algoritmo como o WFQ não pudesse ser utilizado no tratamento da fila de pacotes associada a cada *label path*. De qualquer forma, assim podem ser criadas até oito classes de prioridade para *forwarding* MPLS, compatíveis com as classes de prioridade do *DiffServ*.

Uma confusão comum é achar que, configurando VRFs MPLS, é possível fazer redes virtuais "mais prioritárias" do que outras (isto é especialmente comum em operadoras de telecom). Só que não é verdade. O conceito de VRF serve apenas para que as VPNs configuradas compartilhem a mesma infra-estrutura de transmissão de forma logicamente independente, mas todos os pacotes, de todas as VRFs, que tenham a mesma configuração nos bits EXP serão encaminhados com a mesma prioridade.

Vamos a um exemplo prático (e comum na praça). A operadora XYZ telecom quer implementar um *core* multi-serviço MPLS para poder implementar duas VRFs TCP/IP separadas: uma para o tráfego de serviços aos assinantes, e outra para a sua rede corporativa (tráfego administrativo interno). Suponhamos que na VRF de serviços existam *hosts* que implementam serviços VoIP para os assinantes, e que na VRF corporativa existam *hosts* que implementam um PABX (*private automatic branch exchange*) VoIP.

O mais provável é que ocorra o seguinte: tanto na VRF de serviços quanto na VRF corporativa, os pacotes de voz e de sinalização recebam os mesmos DSCPs (EF e AF43, respectivamente). Enquanto os tráfegos fiquem segregados, tudo bem. Mas, no momento que eles passam a trafegar pelo *core* MPLS, eles receberão as mesmas configurações de bits EXP, portanto o tráfego VoIP dos assinantes irá competir diretamente com o tráfego VoIP corporativo.

Conclusões:

Do que vimos nesta parte da série, chegamos às seguintes conclusões:

a) É possível garantir QoS em redes TCP/IP, mas o *traffic engineering* é mais complicado, e as características específicas de como cada fabricante de roteadores implementa os PHBs *DiffServ* é um dos fatores que deve ser levado em conta (em redes *multi-vendor*, isto pode originar problemas muito interessantes);

b) MPLS não é panacéia. Existem casos onde ele será útil, e outros onde ele só vai atrapalhar.

c) Uma rede de roteadores IP (com ou sem um *core* MPLS) não é, nem vai ser, uma rede de comutação de circuitos. Podemos configurar a rede para minimizar a probabilidade de ocorrência de perdas de pacotes, manter o menor *delay* possível, e controlar o *jitter*, mas o comportamento destas variáveis é estatístico, e não determinístico.

A seguir:

Estamos quase no ponto onde podemos mergulhar mais fundo nas características das aplicações VoIP. Só falta um ponto de base a cobrir, que é fundamental para o dimensionamento da rede TCP/IP que irá transportar tráfego VoIP: a teoria do tráfego telefônico, que será o assunto do próximo artigo.

Inté...