Laborator 6

QoS – Calitatea Serviciilor

Partea 1: Tehnologii Cisco QoS

1.1 Generalităţi

Termenul QoS, folosit extensiv în ultimii 10+ ani, are un sens foarte larg şi se poate referi la orice modalitate prin care se poate asigura o îmbunătăţire/prioritizare faţă de serviciul standard oferit de Internet, şi anume *„best effort*". Implicit, termenul „*best effort*" poate fi considerat sinonim cu *lipsa* oricărui mecanism QoS.

În continuare, ne vom concentra pe o parte din acele tehnologii/mecanisme QoS configurabile pe ruterele Cisco (deci QoS la nivelul 3), folosind protocolul IP. De notat că alte tehnologii, cum ar fi ATM, asigură, conform unora, „abilităţi" QoS mai ridicate, sau chiar singurele „abilităţi" QoS „adevărate".

O clasificare largă a mecanismelor QoS identifică următoarele categorii:

- mecanisme şi politici de control al cozilor şi de control/evitare a congestiilor
- mecanisme de *traffic policing* şi *traffic shaping*
- mecanisme de marcare/clasificare a traficului (ToS, Diffserv, NBAR)
- mecanisme de rezervare (RSVP)

Figura 1 ilustrează, pe modelul unui ruter, locul de acţiune a diferitelor mecanisme QoS.

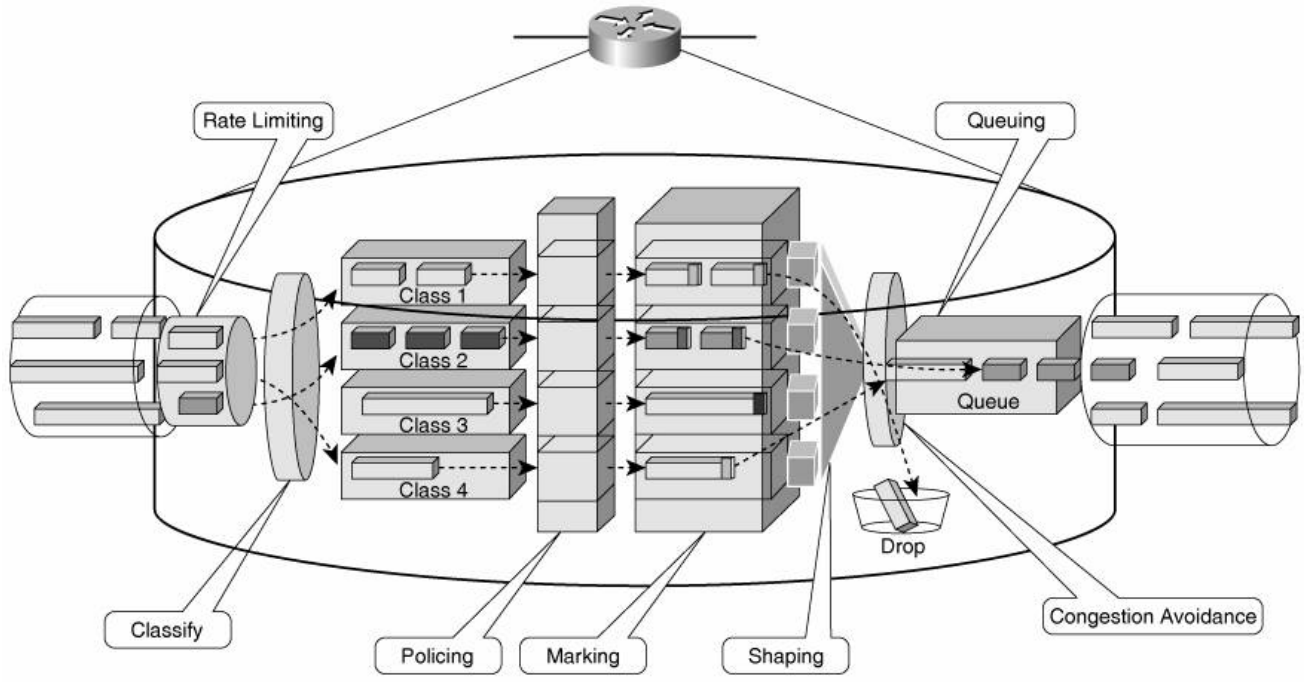

Fig. 1 Acţiunea mecanismelor QoS într-un ruter Cisco; *sursa: Cisco*

1.2 Congestie; cozi

Atunci cînd cererea de bandă depăşeste disponibilitatea apare fenomenul de *congestie*. De exemplu, există mai multe fluxuri de intrare într-un nod, iar suma benzilor (maxime) ale acestor fluxuri depăşeşte banda disponibilă pe interfaţa de ieşire. În acest caz, soluţia este împiedicarea (sau prevenirea) congestiei, dar atunci cînd acest lucru nu este posibil, implementarea unor

mecanisme de *control al cozilor* face ca măcar o parte din trafic (în funcţie de priorităţi) să poată folosi interfaţa de ieşire în condiţii *controlate*.

Două mecanisme sînt în general folosite ca referinţă în procesele de control al cozilor:

1. *Leaky Bucket* (găleata găurită). Se referă la o situaţie similară cu a unei găleţi cu o gaură la partea inferioară şi cu umplerea pe la partea superioară. Gaura "de ieşire" face ca debitul de scurgere a "apei" să fie constant, în timp ce diametrul mult mai mare al "gurii" găleţii permite acumularea cu debit variabil a apei în găleată. Atîta vreme cît găleata nu se umple complet, este posibil să se succeadă perioade în care găleata să se umple cu viteză mult mai mare decît se poate goli, şi perioade în care se umple încet sau de loc. La ieşire, aceste evenimente care au loc la intrare nu se văd – găleata se goleşte cu rată constantă.

2. *Token Bucket* (găleata cu jetoane). Se referă la o găleată care se umple şi se goleşte pe aceeaşi parte (pe sus). Umplerea găleţii se face ca mai înainte; în schimb, golirea găleţii se face pe baza unor jetoane: o "aplicaţie" care doreşte să extragă o anumită cantitate de "apă" dintr-o găleată o poate face doar dacă are un jeton. Jetoanele sînt introduse în găleată de către "controlorul de ieşire", în speţă de către reţea, în funcţie de diferite criterii. Aplicaţiile nu pot scoate o cantitate mai mare de apă decît cantitatea de jetoane disponibile. Pe de altă parte, dacă jetoanele se adună şi nu sînt folosite, peste o limită ele se pierd şi nu mai pot fi utilizate – acest proces se poate modela ca o găleată în care se scurg jetoane, şi care atunci cînd "dă pe dinafară", jetoanele respective se pierd. Aceasta înseamnă că, spre deosebire de *leaky bucket*, un mecanism de tip *token bucket* permite ca la ieşire să se formeze *burst*-uri de trafic, adică momente în care viteza traficului de ieşire este mai mare decît media, dar mărimea acestor *burst*-uri este controlată de numărul maxim de jetoane care se pot acumula.

Tipuri de cozi

FIFO (*First In First Out*) este cel mai simplu tip de coadă; se bazează pe *leaky bucket*, nu se asigură nici o altă prioritate decît prioritatea "primului venit".

FQ (*Fair Queueing*) reprezintă un mecanism în care traficul de intrare este împărţit în fluxuri sau "conversaţii". Împărţirea se face automat, pe baza adresei S/D, portului şi protocolului; Nici un flux nu poate "monopoliza" interfaţa de ieşire; fiecare flux va fi alocat unei cozi şi cozile vor fi deservite pe rînd spre a avea acces la această interfaţă. Mai mult, dacă anumite fluxuri de trafic au bandă ocupată mică, dar au constrîngeri de întîrziere (de exemplu, traficul Telnet), pachetele relativ rare ale acestor fluxuri vor avea prioritate faţă de pachetele acelor fluxuri de bandă mare şi fără constrîngeri de întîrziere (de exemplu FTP). Numărul de cozi este, tipic, mare (256).

WFQ (*Weighted Fair Queueing*) este similar cu FQ, dar se ţine seama şi de prioritatea pachetului în funcţie de un cîmp special – tipic, cîmpul ToS (*Type of Service*) din header-ul IP. WFQ este varianta de FQ implicită pe ruterele Cisco. Pachetele cu prioritate mare primesc o pondere (*weight*) mai mare şi au acces preferenţial la ieşire. Aceasta se face păstrînd ordinea prin care sînt deservite cozile, dar extrăgînd mai multe pachete dintr-o coadă atunci cînd vine rîndul cozii şi pachetele au pondere mai mare.

La FQ şi WFQ traficul este împărţit automat în "conversaţii" separate, fără ca administratorul să poată interveni. Dacă se doreşte configurarea manuală a unor clase pe baza unor ACL-uri, protocoale sau interfeţe, se foloseşte CBWFQ (*Class-Based WFQ*), care permite definirea a 64 de clase. O anumită clasă poate primi un procent semnificativ mai mare din bandă decît altă clasă; de exemplu, dacă se doreşte transferul unui flux video prin FQ/WFQ, în prezenţa multor altor fluxuri, datorită conceptului de "*fairness*", banda alocată fluxului video poate scădea sub minimul necesar unei transmisii inteligibile; în cazul CBWFQ, se poate aloca un minim pentru traficul video, care să fie "*unfair*" pentru celelate forme de trafic, dar care va garanta o anumită calitate pentru acest trafic.

PQ (*Priority Queueing*) este un mecanism care îi dă administratorului un control foarte mare asupra traficului, prin punerea la dispoziţie a 4 cozi în care să împartă tipurile de trafic dorite. Cele 4 cozi au priorităţi în ordine descrescătoare (*high, medium, normal* şi *low*). PQ funcţionează pe principiul că întotdeauna pachetele dintr-o coadă de prioritate mai mare va trece înaintea traficului de prioritate mai mică, în mod integral (nu ponderat). Deci, după ce un pachet iese prin interfaţa de ieşire, se examinează pe rînd, în ordine, cozile, şi următorul pachet este luat din coada de prioritatea cea mai mare. Aceasta înseamnă că eventuala existenţă continuă a traficul prioritar va bloca complet traficul de prioritate mai mică. Acesta este un avantaj dacă există cu adevărat trafic al cărui prioritate justifică blocarea altor tipuri de trafic (de exemplu, Telnet faţă de FTP), dar prezintă dezavantajul că, în cazul unei configurări incorecte (de exemplu, alocarea unei priorităţi mari traficului FTP sau HTTP) se poate bloca complet, pe perioade lungi, interfaţa de ieşire.

CQ *(Custom Queuing*) este o variantă mai "soft" de PQ, care permite stabilirea unor ordini stricte a priorităţii fără a bloca complet traficul mai puţin prioritar. Se alocă maxim16 cozi; pentru fiecare coadă se defineşte numărul de octeţi care vor fi extraşi atunci cînd vine rîndul cozii respective la ieşire; cozile sînt servite pe rînd (*round-robin*). Prin urmare, cozilor de prioritate mare li se vor configura un număr mai mare de octeţi decît celor de prioritate mai mică.

Un amănunt specific CQ este că el nu fragmentează pachetele. Dacă unei cozi i se alocă 1600 de octeţi şi un pachet Ethernet are 1500 octeţi (MTU standard pe Ethernet), cînd vine rîndul cozii respective se procedează astfel: se transmite primul pachet, se constată că mai rămîn 1600-1500=100 de octeţi, deci se începe transmiterea următorului pachet, şi *nu se întrerupe* ceea ce ar însemna fragmentarea pachetului; prin urmare, se vor transmite 1500+1500=3000 de octeţi, deşi administratorul a configurat 1600 ! Trebuie ţinut seama de acest aspect la configurare.

Cozi software şi hardware

Toate mecanismele precedente sînt implementate în software-ul IOS şi folosesc buffere de memorie pentru memorarea pachetelor în cozi. Astfel, mecanismul şi mărimea cozii sînt configurabile. Interfaţa propriu-zisă are, în plus, şi o coadă hardware de tip FIFO, de dimensiuni reduse. Mecanismul de *queueing*, atunci cînd decide ce pachet să fie scos din coadă şi trimis la ieşire, de fapt scrie acest pachet în coada hardware.

Acest lucru este util, în principal pentru a reduce numărul de întreruperi pe care CPU-ul îl primeşte de la interfeţe, dar şi pentru a asigura o viteză cît mai constantă de ieşire.

Eliminarea congestiei

În toate cazurile precedente, coada (sau cozile) de prioritate mai mică pot ajunge să fie deservite prea rar şi ca urmare, prin analogie cu modelul „găleţii" (găurite sau cu jetoane), găleata se umple şi „dă pe dinafară". În termeni de reţea, la depăşirea spaţiului de memorie alocat cozii, pachetele ulterioare vor fi eliminate (*dropped*). Acest lucru este detectat de către entitatea de protocol care emite pachetele respective; dacă aceasta foloseşte un mecanism de control al fluxului (tipic, în cazul TCP, mecanismul *sliding window*), acest mecanism va determina scăderea fluxului emis.

1.3 *Traffic Policing* şi *Traffic Shaping*

Suplimentar faţă de mecanismele prezentate, se poate dori ca diferite tipuri de trafic să fie configurate astfel încît banda fiecăruia atunci cînd părăseşte ruterul să fie diferită şi/sau banda totală să nu fie egală cu banda maximă disponibilă la ieşirea din ruter. Motivele pentru care putem dori acest comportament ar fi:

- ştim că interfaţa de ieşire va fi conectată la o interfaţă de viteză mai mică; de exemplu, avem o interfaţă Fast Ethernet de 100Mbps dar ştim că ne vom conecta la un echipament Ethernet de 10Mbps.
- dorim să alocăm anumite valori maxime de trafic din motive administrative, de exemplu traficul de tip *bit torrent* să nu ocupe toată banda, în detrimentul traficului http, ci doar o fracţiune predeterminată din banda disponibilă.

Pentru a implementa aceste cerinţe se folosesc mecanisme numite *traffic policing* şi *shaping*.

Traffic policing

Se foloseşte un mecanism de tip *token bucket* pentru a limita în mod strict banda. Mecanismul Cisco CAR (*Commited Acces Rate*) se bazează pe acest principiu. Pachetele în exces de rata maximă sînt fie eliminate, fie cîmpul ToS (vezi mai jos, paragraful 1.4) le este rescris astfel încît să fie „candidate" la eliminare mai tîrziu.

Traffic shaping

Este o extensie faţă de cazul precedent; se poate lua în considerare valoarea *medie* a benzii de ieşire în locul celei maxime; astfel, este permisă depăşirea valorii medii de către *burst*-uri, cu condiţia ca acestea să fie de lungime şi bandă maximă controlată, şi să existe şi perioade de trafic zero sau sub medie. Practic, se face *policing* pe valoarea medie a benzii, ceea ce este benefic pentru anumite tipuri de aplicaţii care nu sînt de bandă constantă.

Implementarea *shaping* se face pe baza unui buffer (ca în *leaky bucket*). Atunci cînd banda medie este depăşită, pachetele sînt stocate în buffer. Dezavantajul este consumul de memorie pe ruter, precum şi faptul că *jitter*-ul pachetelor devine variabil.

Mecanismul Cisco corespunzător se numeşte GTS (*Generic Traffic Shaping*). Se folosesc liste de acces pentru clasificarea traficului şi fiecare listă primeşte un tratament diferenţiat. Singura opţiune pentru pachetele care depăşesc rata maximă şi nu pot fi stocate în coadă este *drop* (nu şi rescrierea ToS);

1.4 Clasificarea pachetelor – ToS şi DSCP

Încă din 1981, în header-ul pachetelor IP s-a definit cîmpul de 8 biţi numit ToS (*Type of Service*), care permite diferenţierea pachetelor. Acest cîmp a fost modificat de către RFC1394. Tipurile de pachete se împart după 2 criterii:

- criteriul *Precedence* (primii 3 biţi), cu valori de la 000 (normal) pînă la 111 (*Network Control*). În principiu, o valoare mai mare înseamnă o prioritate mai mare şi un tratament preferenţial (mai puţin susceptibile la eliminare în caz de congestie).
- citeriul *Service Profile* (următorii 4 biţi), cu următoarele valori: 0000=normal, 0001=*minimize monetary cost*, 0010=*maximize reliability*, 0100=*maximize throughput*, 1000=*minimize delay*. În principiu, nu se pot pune 2 biţi simultan pe „1";

Cele 2 criterii s-au folosit, istoric vorbind, din diferite motive, foarte puţin, deşi există de la începuturile Internetului, ceea ce a făcut ca acesta din urmă să fie considerat în general „*best effort*"; majoritatea ruterelor au tratat traficul ca fiind de tipurile „000" respectiv „0000".

În 1998, grupul de lucru *Diffserv* (de la *Differentiated Services*, definite de RFC2474, RFC2475) a propus înlocuirea cîmpului ToS cu un cîmp în mare parte incompatibil, numit DSCP (*Diffserv Code Point*), cu diferenţa de principiu că acest cîmp nu va fi standard şi constant pe întreaga durată de viaţă a pachetului (cît timp acesta străbate Internetul), ci este interpretat (şi poate fi modificat) în interiorul unui *domeniu Diffserv*. Într-un astfel de domeniu, se defineşte conceptul de PHB (*Per-*

Hop Behaviour) care se referă la modul în care este tratat un pachet într-un nod (*hop*) în funcţie de valoarea DSCP.

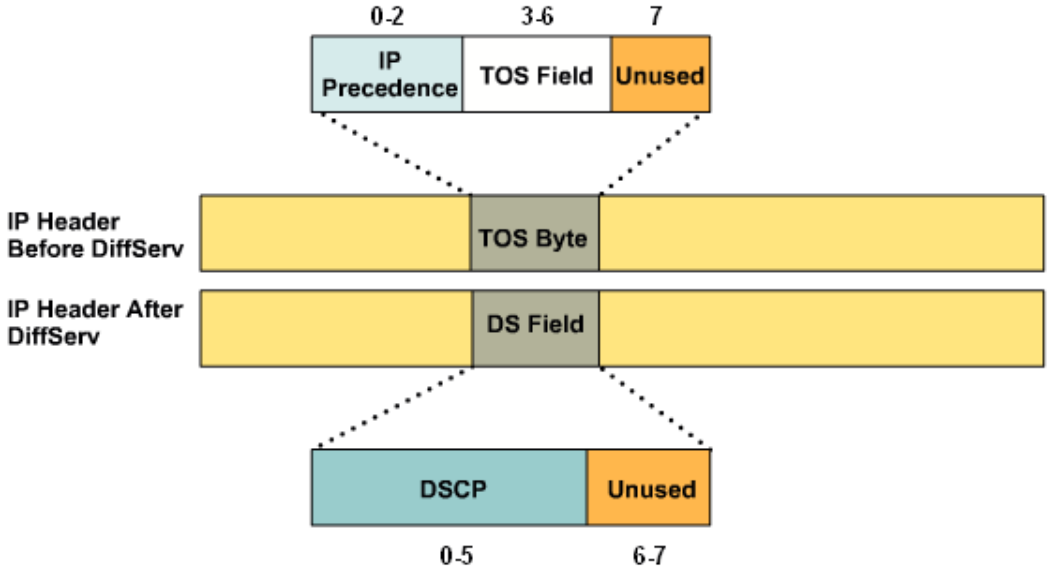

Fig. 4 Cîmpurile ToS şi DSCP; sursa: [7]

Practic, într-un domeniu Diffserv se procedează astfel: se definesc un număr (redus) de clase de serviciu dorite; fiecărei clase i se alocă o valoare DSCP; se face *clasificarea* pachetelor în ruterele de frontieră ale domeniului, în funcţie de criteriile dorite, şi li se alocă valoarea DSCP respectivă; ruterele din interiorul domeniului tratează toate pachetele cu un anumit DSCP într-un anumit mod, adică un anumit PHB; aceasta poate însemna o anumită prioritate într-o coadă, o anumită bandă alocată prin *shaping*, etc.

Structura DSCP presupune folosirea primilor 6 biţi din octet; valoarea 000000 este rezervată pentru „*best effort*". În afara acesteia, sînt 5 clase relativ standardizate:

- clasa EF (*Expedited Forwarding*) este clasa de prioritate maximă; pachetele marcate EF ar trebui să beneficieze de un comportament similar unui circuit dedicat, adică delay cît mai mic şi fără pierderi
- 4 clase AF (*Assured Forwarding*) notate AF1...AF4, cu AF4 avînd prioritatea cea mai mare. În cadrul fiecărei clase se definesc 3 sub-clase numite *Gold, Silver* şi *Bronze* – în total 12 categorii care diferă prin parametrii cu care se configurează mecanismele de *queueing*, *policing* şi/sau *shaping*.

Aşadar trebuie reţinut că Diffserv nu reprezintă un set de valori standard, ci posibilitatea administratorilor dintr-un anumit domeniu (de exemplu, reţeaua unui ISP) de a împărţi traficul în clase DSCP după criterii proprii (adresă S/D, protocol, port, etc) şi de a-i asigura un tratament diferenţiat în noduri (PHB). DSCP este suportat atît pe IPv4 cît şi IPv6.

1.5 Clasificarea automată a pachetelor - NBAR

O variantă alternativă clasificării manuale este clasificarea automată, pe baza recunoaşterii tipului pachetului prin examinarea sa de către ruter, numită NBAR (*Network Based Application Recognition*). NBAR este o funcţie avansată, destinată doar pachetelor IP (nu şi IPX, etc), care nu examinează doar numărul de port, ci şi o parte din conţinutul pachetului – în cazul pachetelor HTTP, se examinează primii 400 de octeţi pentru a determina *MIME-TYPE*, adică tipul pachetelor multimedia cum ar fi .mp3, etc. Pe baza examinării pachetului, se determină aplicaţia şi protocolul care îl foloseşte, inclusiv (în cazul anumitor protocoale) situaţiile în care numărul portului este

alocat dinamic. Fiecare versiune mai nouă de IOS a adus recunoaşterea unor aplicaţii suplimentare cum ar fi SSH, Napster, eDonkey, Gnutella, Kazaa, Bit Torrent, Skype, etc.

De notat că, dacă o versiune IOS mai veche nu recunoaşte anumite protocoale mai noi, se poate folosi un fişier PLDM (*Packet Description Language Module*) care specifică stringuri de text din interiorul pachetelor, ce sînt asociate cu protocoalele. Fişierul se încarcă în *flash*-ul ruterului, fără a fi necesară schimbarea IOS-ului.

NBAR nu poate funcţiona în cazul tunelelor sau a traficului criptat. În plus, folosirea NBAR poate consuma resurse semnificative de memorie şi de procesor.

1.6 Suportul Cisco IOS pentru QoS

În general, strategiile de *queuing*, *policing* şi *shaping* sînt disponibile pe majoritatea imaginilor IOS pentru ruterele Cisco, inclusiv pe cele de tip „*low-end*" cum ar fi 800, 1700, 1800, 2500, 2600 etc, deoarece se consideră că aceste strategii trebuie implementate la periferia reţelei, şi nu în „*core*" unde se folosesc, tipic, rutere de tip *„high-end"* cum ar fi seriile 7000 etc, dar al căror rol este în principal să ruteze pachetele pe linkuri de mare viteză (*backbone*).

Feature-set-ul numit generic IP BASIC suportă o parte din strategii, cele considerate de bază. Un set complet, cuprinzînd şi cele mai avansate sînt suportate în variantele IP PLUS. *Feature-set*-ul specific unei imagini IOS se poate determina folosind [6].

Pe linkurile de viteză mai mică sau egală cu E1 (2.048Mbps) politica de *queueing* implicită este (începînd cu IOS 11.0) WFQ, în timp ce pe linkurile mai rapide este FIFO. Aceasta pentru că, deşi mai ineficient, FIFO este mai rapid, şi pe interfeţele de viteză mare se presupune că se doreşte (implicit) să se consume cît mai puţin timp de procesare.

NBAR şi CAR sînt suportate numai pe IOS care includ CEF – *Cisco Express Forwarding*, un tip de *forwarding* „rapid" al pachetelor IP între interfeţe, implementat din punct de vedere istoric în versiuni mai noi de IOS decît cel „clasic". Practic, se crează o tabelă de „forwarding rapid" numită FIB – *Forwarding Information Base* pe baza tabelei de rutare, care nu mai este consultată la fiecare pachet. Folosirea CEF este activată prin comanda

```
router(config)# ip cef
```

care apare implicit în fişierul de configurare la IOS mai noi (şi se poate da manual la IOS mai vechi care îl suportă). Pe seria 7000, CEF este activat implicit. De notat că `ip cef` este o comandă care, spre deosebire de comenzile globale folosite pînă acum (gen `ip classless`, `ip subnet-zero`, `ip routing` etc.) modifică modul intern de funcţionare al ruterului, nu activează/configurează un serviciu.

Pe seriile 2600, 3600, 7100, 7200 NBAR este disponibil de la IOS 12.1(5)T şi pe 1700 de la 12.2(2)T. În general T semnifică *Technology Release* şi marchează apariţia unei funcţionalităţi noi.

Partea 2. Desfăşurare; exerciţii

Faza 1. Generarea traficului de test între PC-uri

Se realizează următoarea topologie:

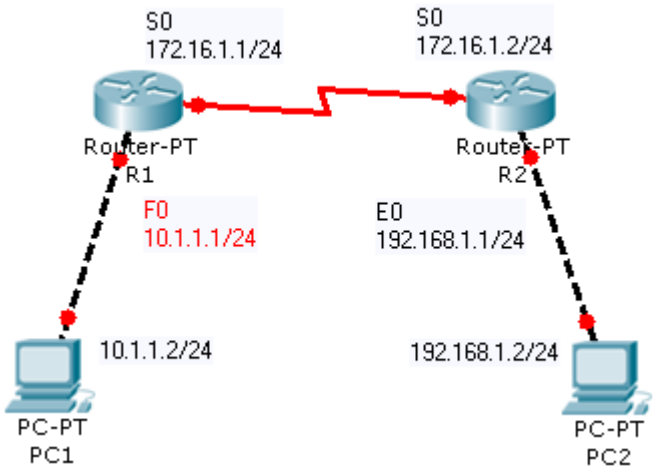

Fig. 5 Topologia folosită

Q1. Desenaţi topologia corespunzătoare echipei, notînd numele ruterelor, interfeţelor şi PC-urilor folosite.

Pe rutere:
- seriala DCE va fi configurată cu `clock rate 500000` (nu altă valoare!) ; ***Atenţie !*** în funcţie de tipul ruterului folosit, este posibil să primiţi o eroare de tipul « unsupported clock rate » sau « clock rate exceeds .... » ; în acest caz, configuraţi celălalt capăt ca DCE (schimbînd corespunzător cablul), căci o interfaţă care nu acceptă să genereze această valoare, o va accepta dacă este generată de celălalt capăt.
- se configurează RIP v2 pentru a avea comunicaţie capăt-la-capăt.
- pe R1 se porneşte `ip cef`

Dacă nu e deja, pe fiecare dintre cele 2 PC-uri trebuie configurată cîte o rută statică către net-ul celuilalt, de exemplu:

```
PC1# ifconfig eth1 10.1.1.2 netmask 255.255.255.0
PC1# route add -net 192.168.1.0/24 gw 10.1.1.1
```

Rutele pe PC se văd cu comanda:

```
PC1$ route -n
```

Se verifică abilitatea de a da ping de pe un PC pe celălalt (trebuie să funcţioneze).
Verificaţi de pe R2 posibilitatea de a da ping pe PC1.

Q2. De ce nu funcţionează, deşi funcţionează ping între PC-uri ? ce trebuie făcut pentru ca să funcţioneze (fără a adăuga rute suplimentare) ?

Pe PC-uri, traficul de test se poate genera/măsura folosind utilitarul *iperf*. Acesta se apelează în modul „client” pentru generarea traficului, care se conectează la un „server” unde tot *iperf*

recepţionează traficul şi afişează rezultatele (viteză, jitter, pachete pierdute). Cîteva comenzi (în linia de comandă Linux):

```
PC1$ iperf –c 192.168.1.2 –l 100 –b 100k –p 1234 –t 10
```

Trimite (în mod client, către serverul 192.168.1.1) pachete de lungime 100 biţi, **cu rata de 100kbiţi/s**, UDP, portul 1234, *timp de funcţionare 10s*;

Specificarea –b înseamnă automat UDP.

```
PC2$ iperf –s –u  –p 1234 –i 0.5
```

Recepţionează în mod server, UDP, portul 1234, interval de afisare 0.5 secunde

Faza 2. Strategii de *queueing*

1. Se examinează cele 2 interfeţe active de pe R1 şi R2 folosind `show interface nume`

Q3. Ce tip de *queueing strategy* este folosită pe fiecare interfaţă? corespunde aceasta cu regula prezentată în 1.7, privind legătura dintre bandă şi strategia implicită?

Se verifică banda afişată cu comanda show de mai sus, pe R1/S0. Se observă că nu este egală cu *clock rate*-ul setat. La ruterele Cisco, banda afişată (si setată) cu comanda `bandwidth` este doar o convenţie, nu este automat egală cu viteza de la nivelul 2; este responsabilitatea userului să o seteze cu comanda `bandwidth`, pentru ca protocoalele de rutare şi alte funcţii care folosesc banda, să folosească valoarea corectă (vezi OSPF).

Întrucît viteza liniei a fost setată la 500kbps, se setează pe ambele rutere:

```
R1(config)# int S0
R1(config–if)# bandwidth 500
```

2. Se configurează pe seriala R1 strategia **FIFO** folosind**:**

```
R1(config)# interface S0
R1(config–if)# no fair–queue
```

Se testează capacitatea liniei; se pornesc 2 „conversaţii” simultane între PC-uri (lansaţi atîtea terminale cît este nevoie)
- porniţi ping de pe PC2 pe PC1
- folosind *iperf*, generaţi trafic de pe PC1 pe PC2: banda 100kbps, lungime 100b, timp de lucru 10s; portul poate fi lăsat *default* (se va alege automat 5001); pe PC1 lansaţi un client şi pe PC2 un server, folosind comenzile ilustrate în introducere.

Q4. Cît este banda raportată de server (PC2), precum şi *packet loss* ? Mai merge pingul în celălalt terminal ?

Se repetă testul *iperf*, incrementînd succesiv banda la emisie, la valorile: 300kbps, 500kbps, 1Mbps (prescurtări folosite: k=kbps, M=Mbps).

Q5. Cît este banda raportată de server (PC2), precum şi *packet loss* ? Mai merge pingul în celălalt terminal ? calculaţi banda obţinută ca procentaj din viteza liniei.

Observaţi că banda pe care o puteţi obţine practic este mai mică decît viteza liniei. La ruterele Cisco, se consideră că banda disponibilă (şi care poate fi rezervată, cînd faceţi calcule de QoS) este de max. 75% din viteza liniei, restul fiind folosit de traficul *overhead* de nivel 2, traficul de control, şi altele.

Q6. Examinaţi seriala pe R1 cu comanda show. Observaţi vreun *drop* ?

Se creşte bufferul alocat cozii FIFO de ieşire, pentru a vedea efectul acestuia:

```
R1(config-if)# hold-queue 4096 out
```

Se şterg statisticile pe interfeţe cu comanda:

```
R1# clear counters
```

Se porneşte din nou traficul *iperf* de pe PC1 pe PC2 cu o bandă de 1Mbps, pentru 15 secunde. Ce observaţi ?

Q7. Cum este procentajul de pachete pierdute la începutul, respectiv la sfîrsitul intervalului de 15s ? Argumentaţi, ţinînd seama de comanda care tocmai a fost dată pe ruter.

Se verifică nr. de pachete pierdute pe interfaţă:

```
R1# sh int S0 summary
```

Concluzia este că, în prezenţa congestiei, introducerea unei cozi de tip FIFO nu are efect pe termen lung. Mecanismele de *queueing* nu pot compensa decît în mică măsură situaţiile în care banda cerută este semnificativ mai mare decît banda disponibilă.

3. Se configurează strategia **WFQ**

Comanda este:

```
R1(config)# interface S0
R1(config-if)# fair-queue [A [B C]]
```

Unde A, B, C sînt (momentan nu se introduc, astfel că se vor folosi valorile *default*):
A = *congestive discard treshold* (putere a lui 2, 16-4096) = pragul unde începe eliminarea preventivă a pachetelor dintr-un flux în caz de congestie. WFQ începe eliminarea pachetelor din fluxul cel mai „agresiv”.
B = numărul de cozi dinamice pentru conversaţii (16-4096, implicit 256); trebuie să fie o putere a lui 2
C = numărul de cozi rezervabile prin RSVP (0-100)

Se verifică parametrii setaţi cu `show interface` (ei sînt listaţi imediat după ce este precizată *queueing strategy*) şi cu comanda `show queueing fair`

Cu ping-ul pornit în continuare, se generează din nou trafic *iperf* de pe R1 pe R2, cu banda de 1Mbps.

Q8. Cît este banda obţinută la recepţie în *iperf* ? diferă faţă de cazul FIFO ? Ce se întîmplă cu pachetele ICMP ping (observaţi dacă sînt pierderi sau nu urmărind nr. de secvenţă)? Explicaţi diferenţa de comportament şi implicit avantajul WFQ.

Q9. Cît este numărul de fluxuri (conversaţii) indicat pe R1 cu `sh queue s0` ?

Observaţie: WFQ funcţionează în mod automat şi împarte singur traficul în fluxuri (conversaţii). WFQ alocă o coadă (din cele maxim 4096, parametrul B) atunci cînd primeşte un pachet pe care nu-l poate clasifica într-o conversaţie în derulare, adică pentru care nu există o coadă. Odată ce ultimul pachet dintr-o conversaţie este trimis şi coada se goleşte, conversaţia respectivă este eliminată din lista de conversaţii. O nouă serie de pachete trimise mai tîrziu cu aceiaşi parametri vor forma o coadă, si implicit o conversaţie, separată.

Pentru a genera mai multe conversaţii simultane, se pornesc (în terminale separate, alături de ping care se lasă pornit) 2 *iperf*-uri: 2 clienţi pe PC1 şi 2 servere pe PC2, cu parametrii similari cu cei precedenţi, cu diferenţa:

- un flux de 1Mbps pe portul 2000
- un flux de 1Mbps pe portul 3000

Q10. Cît este banda raportată de *iperf* la recepţie ?

Q11. Cît este numărul de conversaţii indicat pe R1? vizualizaţi parametrii fiecăreia (nr. protocolului şi portul) folosind comanda `sh queue s0` în timp ce traficul e în desfăşurare.

Q12. Pe baza observaţiei de mai sus, explicaţi de ce traficul ICMP (1 pachet ping pe secundă) apare/nu apare ca o conversaţie separată în listă.

4. Se configurează strategia **PQ**

Priority Qeueing foloseşte un set de 4 cozi cu priorităţi diferite; reamintim că se preia *un singur pachet* dintr-o coadă la un moment dat. Congestia într-o coadă de prioritate *N* înseamnă automat congestia în toate cozile de prioritate < *N*.

Configurarea se face astfel:

- se împart tipurile de pachete dorite în 4 cozi, cu numele *high, medium, normal, low*, pe diverse criterii: protocol, interfaţă, port, mărimea pachetului sau ACL
- se asignează un tip de pachete o coadă, folosind cuvîntul-cheie **`priority-list`**. Ca şi la definirea unui ACL, un priority-list poate avea mai multe definiţii. În fiecare definiţie, cuvintele-cheie `high, medium, normal, low` specifică cărei cozi i se aplică definiţia. Comanda este:

`priority-list` *`număr`* **`protocol`** *`nume`* **`high|medium|normal|low`** *`criteriu valoare`*

Criteriul *criteriu* şi valoarea poate fi unul din următoarele:

- `fragment`
- `gt/lt` *`dimensiune`* (*greater than/less than*)

```
-list număr_acl
-tcp/udp număr_port
```

Mai există şi variantele:

priority-list *număr* **interface** *nume* **high|medium|normal|low**
(clasifică în funcţie de interfaţa pe care vine pachetul)

priority-list *număr* **default high|medium|normal|low**
(defineşte valoarea implicită a pachetelor care nu sînt clasificate altfel)

- se setează dimensiunea celor 4 cozi (dacă nu convine valoarea implicită) folosind **priority-list** *număr* **queue-limit *A B C D.*** Valorile implicite sînt 20,40,60,80 pachete (în ordine de la *high* la *low*)
- se aplică listele de priorităţi pe interfeţe, folosind cuvîntul-cheie **priority-group** *număr*, folosind acelaşi număr de listă care a fost folosit în **priority-list** (similar cu comenzile cunoscute `access-list` şi `access-group`)

Se configurează 4 liste, toate în grupul 1. Prima listă plasează pachetele ping în coada de prioritate *high.* Diferenţierea se face printr-un ACL.

Q13. Scrieţi şi configuraţi ACL-ul cu numărul 101 pentru selectarea pachetelor ping

Următoarele 2 liste configurează în cozile *medium/normal* traficul UDP pe porturile 2000/3000. Ultima configurează „tot restul" ca prioritate *low*.

```
R1(config)# priority-list 1 protocol ip high list 101
R1(config)# priority-list 1 protocol ip medium UDP 2000
R1(config)# priority-list 1 protocol ip normal UDP 3000
R1(config)# priority-list 1 default low

R1(config)# int s0
R1(config-if)# priority-group 1
```

Se testează generînd trafic ca pînă acum (ping R2->R1, 2 *iperf* R1->R2 pe UDP/2000 şi 3000, ambii *iperf* cu banda de 300k, timp de lucru 10s)

Se urmăresc rezultatele pe PC2, precum şi pe R1 folosind:

```
show int S0
show queueing priority
show queue S0
```

Q14. Cît este banda şi *packet loss* obţinute pentru cele 2 porturi UDP? Explicaţi !

Q15. Există pierderi pentru pachetele ping ?

Pentru a vedea „în timp real" ce se întîmplă, daţi comanda:

```
R1# debug priority
```

şi reporniţi cele 2 *iperf* pentru 2-3 secunde. Observaţi că activitatea intensă de debug încarcă procesorul lui R1 şi afectează timpul RTT la ping.

Opriţi debugging-ul cu `undebug all`.
4. Reamintim că, în afara cozilor software, interfaţa are o mică coadă hardware de tip FIFO. Pentru seriale/ethernet, aceasta poate fi vizualizată cu comanda `show controllers serial` *`număr`*, respectiv `show controllers fastethernet` *`număr`*, urmărind în rezultat variabila *tx_limited* (cifra din paranteză) sau *tx_ring_limit* sau *tx_ring*.

Q16. Cît este mărimea cozii hardware pe interfeţele serială/ FastEthernet ale R1? explicaţi mărimile diferite.

Faza 3. Configurarea CAR şi GTS

CAR şi GTS sînt mecanismele Cisco pentru *traffic policing* şi *traffic shaping*; diferenţa dintre ele este că *shaping*-ul poate lucra cu valori medii, ceea ce înseamnă că foloseşte buffere pentru a „acumula" pachetele care vin în burst-uri şi le trimite la ieşire cu o rată constantă. În acest fel, atîta vreme cît rata medie nu este depăşită, este posibilă acceptarea unor rate de vîrf mari. Prin contrast, *policing* nu asigură buffering pentru a permite depăşirea unei rate maxime, dar poate accepta, în anumite limite, depăşirea acestei rate de către bursturi (*policing* fără nici un fel de depăşire se numeşte *true policing*).

1. Configurarea CAR. Sintaxa CAR este versatilă, suportînd un mare număr de opţiuni:

```
Router(config-if)# rate-limit {input / output}
[access-group [rate-limit] număr_acl / qos-group număr | dscp
dscp] mean-rate Bc Be
conform-action { drop | transmit | continue |
set-prec-transmit value | set-prec-continue value /
set-qos-transmit value / set-qos-continue value
set-dscp-transmit value | set-dscp-continue value}
exceed-action { drop | transmit | continue |
set-prec-transmit value | set-prec-continue value /
set-qos-transmit value / set-qos-continue value
set-dscp-transmit value | set-dscp-continue value}
```

Comanda se poate aplica la *intrarea* sau la *ieşirea* dintr-o interfaţă şi are 4 părţi:

- partea de clasificare; se poate face după un acces list, un grup QoS (setat de altă comandă rate-limit), o valoare DSCP
- valoarea de comparat; în cazul *true policing* se defineşte o singură valoare; în cazul în care se acceptă depăşiri, *Bc* este valoarea burstului normal, adică numărul de jetoane care se adaugă periodic în mecanismul *token-bucket* şi *Be* este burstul în exces, adică burstul iniţial permis cînd *bucket*-ul este plin cu numărul maxim de jetoane
- acţiunea în cazul în care rata corespunde ratei maxime: *drop*, *transmit*, *continue* (evaluează următoarea comandă `rate-limit`) sau setează o valoare (*prec* adică precedenţă ToS, *dscp* adică valoare DSCP, *qos* pentru un grup QoS); setarea valorii se poate face urmată de transmiterea pachetului (variantele cu *transmit* la coadă) sau cu evaluarea următoarei comenzi `rate-limit` (variantele cu *continue* la coadă);
- similar, acţiunea în cazul în care rata nu corespunde.

Verificarea funcţionării se face cu comanda `show interface` *`interfaţa`* `rate-limit`

Se configurează R1 pentru a face *policing* asupra traficului UDP/2000 dinspre R1 spre R2 la valoarea de 200kbps, respectiv 100kbps pentru 3000, cu posibilităţile următoare de depăşire: Bc=4000 octeţi şi Be=96000 octeţi (de notat că rata este în bps şi următorii 2 parametri sînt în octeţi)

Se definesc listele 110, 120 care permit traficul UDP pe porturile, respectiv, 2000 şi 3000

Q17. Scrieţi aceste ACL.

Se aplică comanda pe interfaţa S0:

```
R1(config)# int S0
R1(config-if)# rate-limit output access-group 110 200000 4000
96000 conform-action transmit exceed-action drop
```

Şi similar pentru lista 120, banda *100000*

Q18. Ce observaţi cu comanda `sh int S0 rate-limit` ?

Testaţi folosind *iperf* cu pachete de lungime 1000.

Q19. Ce bandă (în medie) aţi obţinut la recepţie pentru cele 2 porturi ? traficul ping a fost afectat ?

2. Configurarea GTS; la nivel global, se face folosind comanda:

```
Router(config-if)# traffic-shape rate rata [Bc [Be]]
```

iar la nivel de flux de trafic, identificat printr-un ACL, se face folosind comanda:

```
Router(config-if)# traffic-shape group număr_ACL rata [Bc [Be]]
```

Observaţii:
1) pe aceeaşi interfaţă nu se pot combina cele 2 comenzi
2) în cazul folosirii access-grupurilor, traficul care nu corespunde nici unui ACL nu este *shaped*

Se elimină cu **no** comenzile rate-limit de pe interfaţa S0;

Se configurează R0 pentru a face *shaping* asupra traficului de tip web către R2; acesta nu va depăşi 64kbps; restul traficului nu va fi afectat:

```
R1(config)# int S1
R1(config-if)# traffic-shape group 130 64000
```

Q20. Scrieţi şi configuraţi ACL 130 care trebuie folosit în conjuncţie cu această comandă.

Pentru vizualizare se folosesc comenzile:

```
show traffic-shape
show traffic-shape statistic
show traffic-shape queue
```

(testarea este opţională)

Se elimină cu **no** comenzile `traffic-shape` de pe interfaţa S0.

Faza 4. Configurarea NBAR

1. Vizualizarea asocierilor cunoscute protocol-port se face cu comanda:

```
show ip nbar port-map
```

Q21. Pe ce protocoale/porturi funcţionează următoarele: netbios, dns, ssh, tftp, pop3, rip, bgp ?

Pentru a configura NBAR pe o interfaţă mai întîi se activează funcţia de *protocol discovery*.

```
R1(config)# int s0
R1(config-if)# ip nbar protocol-discovery
```

Vizualizarea protocoalelor descoperite se face cu comanda

```
show ip nbar protocol-discovery
```

Q22. Executaţi acţiuni la alegerea voastră pe PC-uri, pentru a face să apară pachete în lista NBAR (mai mult de 0 pachete) pe R1 la cel puţin 3 protocoale cunoscute, şi documentaţi alegerile făcute.

2. Folosim NBAR ca să clasifice pachetele, şi configurăm clase de QoS pentru pachetele clasificate. Vom defini 3 clase, de prioritate mare, mică şi *default* folosind comanda `class-map`; apoi vom asocia ce procentaj din banda disponibilă să fie alocată fiecărei clase, folosind comanda `policy-map`.

Protocoalele configurate vor fi dintre cele recunoscute de NBAR (se poate da ? în locul numelui protocolului pentru a vedea opţiunile):

```
R1(config)# class-map match-any mare
R1(config-cmap)# match protocol rip
R1(config-cmap)# match protocol icmp
R1(config-cmap)# class-map match-any mica
R1(config-cmap)# match protocol http
```

Se vizualizează apariţia clasei class-default:

```
R1# show class-map
```

Se alocă procentaje din bandă celor 3 clase:

```
R1(config)# policy-map laborator
R1(config-pmap)# class mare
R1(config-pmap-c)# bandwidth percent 50
R1(config-pmap)# class mica
R1(config-pmap-c)# bandwidth percent 20
```

```
R1(config-pmap)# class class-default
R1(config-pmap-c)# bandwidth percent 5
```

Se vizualizează rezultatul:

```
R1# show policy-map
```

Se aplică politica pe interfaţa S0, sensul de ieşire:

```
R1(config-if)# service-policy out laborator
```

Q23. Ce benzi sînt afişate cu comanda `show policy-map int s0` ?
Q24. Propuneţi o metodă de testare a politicilor definite.

Bibliografie

[1] *Administering Cisco QoS in IP Networks*, Syngress, 2001
[2] *Introduction to IP QoS*, Cisco Systems, 2001
[3] RFC791, *Internet Protocol*
[4] RFC2474, *Definition of the Differentiated Services Field*
[5] RFC2475, *An architecture for Differentiated Services*
[6] Cisco Feature Navigator, *http://tools.cisco.com/ITDIT/CFN/jsp/index.jsp*
[7] CCNP4, v5.0