큐잉 매커니즘의 이해와 활용

큐잉은 QoS에서 중요한 4가지 측정요소인 대역폭, 지연, 지터, 패킷 손실을 직접적으로 컨트롤할 수 있는 기술이다. 특히 중요한 트래픽에 대한 패킷 손실을 예방하며 허용된 대역폭 내에서 지연을 최소화할 수 있는 막강한 기능을 제공한다. 많은 이들이 QoS라는 말을 들을 때 제일 먼저 큐잉을 떠올릴 정도로 애용되고 있으며, QoS를 제공하기 위한 DiffServ의 여러 모듈들 중에서 유일하게 매번 사용되는 모듈이다.

김화식
zion@onsetel.co.kr 온세통신 시설운영팀 CCIE
s12840@freechal.com NRC Network+ 팀 마스터

난 이 도 上 中 下

지난호에서 우리는 트래픽 쉐이핑과 폴리싱에 대해 알아봤다. 임계값(threshold)을 초과한 트래픽에 대해 버퍼를 이용해 지연시켰다가 서비스하는 쉐이핑과 드롭을 시키는 폴리싱에 대해 설명을 했으며, 쉐이핑을 이용하면 네트워크 상의 이그레스 블록킹(egress bloking) 문제와 대부분의 QoS 관련 이슈(패킷 손실에 의한 성능저하 문제)를 해결할 수 있다는 점을 설명했다. 또한 폴리싱을 이용하면 네트워크의 용량 부족 문제를 해결할 수 있으며, 대역폭을 쉽게 관리할 수 있고, 바이러스나 P2P 패킷과 같은 일부 서비스가 전체 네트워크 서비스에 영향을 미치는 것을 예방할 수 있다는 것도 알아봤다.

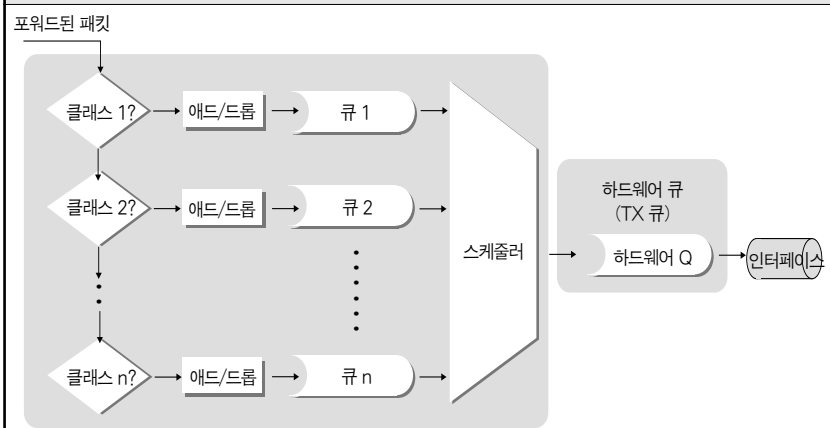
이번호에는 큐잉(queuing)에 대해 알아볼 것이다. 큐잉의 종류와 장단점, 그리고 실무에서의 적용 방법 등을 중심으로 이야기를 풀어나갈 것이다. 우선 큐(queue)의 기본에 대해 알아보자. 큐를 이해하는 가장 간단한 방법은 버스를 탈 때 줄서는 모습을 생각하면 된다. 보통 차례대로 버스를 타게 되면 제일 앞에 줄서있는 사람이 제일 먼저 버스에 올라탈 수 있다. 이런 원리가 큐의 가장 기본이 되는 FIFO(First-In First-Out) 큐인 것이다.

네트워크에서 큐잉의 의미

네트워크에서 큐잉은 어떤 네트워크 장비가 처리(서비스)할 수 있는 것 이상으로 패킷이 도착하거나, 동시에 동일한 목적지로 향하는 패킷들이 존재할 때 발생한다. 즉, 한꺼번에 처리할 수 없는 패킷들을 잠시 동안 버퍼(메모리)에 저장해 두었다가 나중에 서비스를 하는 것이 큐잉이다. 이때 사용되는 버퍼의 종류는 하드웨어 큐와 소프트웨어 큐로 나눌 수 있다.

좀더 쉽게 풀어서 설명하면 라우터로 들어온 인바운드 트래픽은 라우팅 프로세스에 의해 라우팅 경로가 결정되며, 결정된 출력 인터페이스를 통해 서비스된다. 이때 라우터의 인터페이스에서 연속 지연(serialization delay) 시간 동안 패킷을 저장해야 하는 상황이 생기며, 이를 처리하기 위해 각 인터페이스마다 일정량의 하드웨어 큐를 갖는다. 이

그림 1 | 큐잉의 구성 요소



· 큐잉은 소프트웨어 큐, 하드웨어 큐, 인터페이스 등의 세가지로 구성된다.

때 인터페이스에 할당되는 하드웨어 큐를 보통 TX 큐(Transmit Queue) 또는 TX 링(Transmit Ring)이라고 말한다. 이 하드웨어 큐의 크기는 라우터의 성능에 따라서 달라진다. TX 큐의 특징은 다음과 같다.

- FIFO 스케줄링(scheduling)만 지원하며 변경할 수 없다.
- 인터페이스별로 1개만 사용할 수 있다.
- 다른 큐잉 스케줄링을 사용하면 IOS는 TX 큐의 길이를 자동으로 줄여준다.
- 관리자가 임의의 설정을 통해 TX 큐의 길이를 조절할 수 있다.

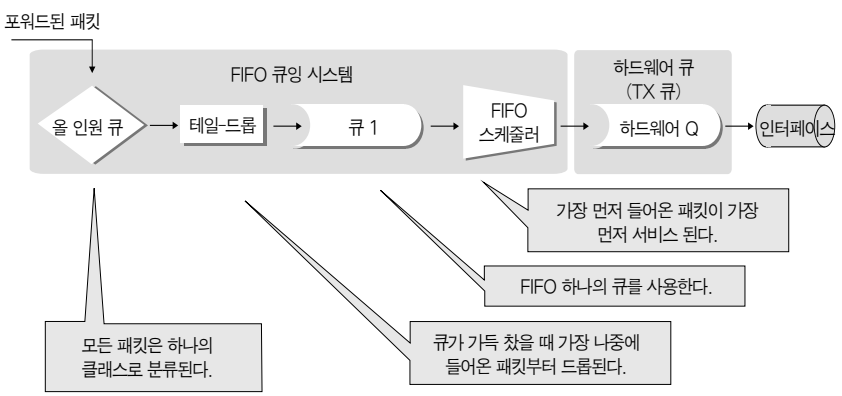
TX 큐는 FIFO 스케줄링만을 지원하기 때문에 FIFO 큐잉의 가장 큰 단점인, 모든 트래픽에 대해 단일 트래픽으로 인식하고 단지 큐에 들어온 순서에 의해서만 서비스를 해주는(베스트 에포트 서비스) 문제를 갖고 있다. 이 문제를 해결하기 위해 TX 큐 앞에 소프트웨어 큐를 두어 스케줄링을 제공한다. 우리가 일반적으로 알고 있는 큐잉은 이 소프트웨어 큐를 지칭하는 말이다.

FIFO 큐잉의 이해

FIFO 큐잉은 단일 FIFO 큐를 사용하는 것을 말한다. 가장 기본적인 큐잉의 구조로 하드웨어 큐의 연장선상에서 생각하면 된다. 하드웨어 큐(TX 큐) 앞에 또 하나의 하드웨어 큐를 제공하는 것과 같은 동작을 한다. 일반적으로 FIFO 큐잉은 하나의 큐에 모든 클래스의 트래픽을 저장하게 된다. 그 이유는 큐가 하나밖에 없기 때문이다.

FIFO 큐잉에 사용되는 스케줄링 방식은 'First Come First Serve'다. 즉, 패킷의 클래스나 우선순위에 상관없이 먼저 입력된 패킷을 먼저 서비스하게 된다. 베스트 에포트 서비스 모델만을 갖고 있는 전통적인 인터넷에서 사용되는 큐잉과 스케줄링 구조에 해당한다. 트래픽의 구분이 존재하지 않기 때문에, FIFO 큐잉을 사용하는 장비에서는 패킷 분류(classification)나 마킹(marking)처럼 클래스와 관련된 기능은 필요 없다.

그림 2 | FIFO 큐잉

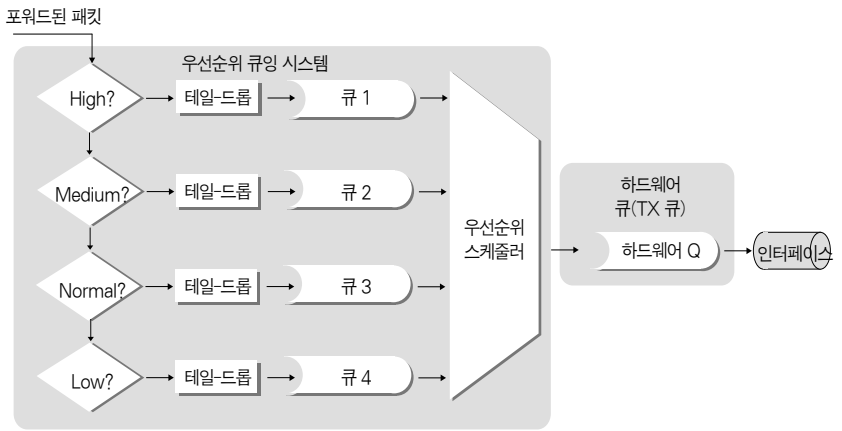


· FIFO 큐는 기본적으로 하드웨어 큐의 연장선상에서 동작한다.

FIFO 큐잉의 장점은 구현이 간단하며 FIFO 큐의 동작이 예측 가능하다는 것이다. 즉, 패킷들의 순서가 유지되며, 패킷의 최대 지연은 큐의 최대 크기에 의해 결정된다. FIFO 큐잉의 단점으로는 클래스의 구분이 없기 때문에 차등화된 서비스를 제공하는 것이 불가능하다는 점이다. 또한, 테일(tail) 드롭이 발생하기 때문에 버스트 트래픽 서비스에 부적합하며, 혼잡이 발생하는 경우 TCP보다 UDP 트래픽에 유리하다.

즉, TCP와 UDP 트래픽이 혼재하는 경우 혼잡이 발생하면, TCP 센더는 흐름 제어 알고리즘에 의해 전송 속도를 줄이게 되지만, UDP 센더는 계속해서 트래픽을 보내게 되며 TCP의 흐름 제어에 의해 발생한 대역폭을 차지하게 돼 결국에는 TCP 트래픽의 서비스가 어렵게 된다. TCP 흐름제어에 대한 자세한 설명은 혼잡회피부분에서 보다 자세하게 설명할 것이다.

그림 3 | 우선권 큐잉



· 우선순위 큐잉은 4개의 FIFO 큐를 사용한다.

FIFO 큐잉을 개선한 '우선순위 큐잉'

엄격한 우선순위(strict priority) 큐잉이라고도 불리는 우선순위(priority) 큐잉은 FIFO 큐잉의 단점인 클래스의 구분이 없기 때문에 차등화된 서비스를 제공하지 못하는 문제를 해결하기 위해 만들어 졌다.

(그림 3)을 보면 우선순위 큐는 기존의 FIFO 큐잉이 단일 FIFO 큐를 사용하는 것에 비해 'High' 'Medium' 'Normal' 'Low'의 4가지

FIFO 큐를 사용하는 것을 알 수 있다. 4개의 FIFO 큐를 사용하므로, 각각의 큐가 서로 다른 트래픽 클래스에 매핑이 된다.

우선순위 큐잉을 사용하는 경우 스케줄링 방식은 아주 단순하다. (그림 4)를 보면 낮은 우선순위 큐에 저장돼 있는 패킷들은 높은 우선순위 큐에 저장돼 있는 패킷들이 모두 서비스 된 이후에 서비스가 되는 것을 알 수 있다. 만약, 낮은 우선순위 큐에 저장돼 있는 패킷이 서비스되는 도중이라도 높은 우선순위 큐에 패킷이 입력되면, 낮은 우선순위 큐는 서비스를 잠시 멈추고 높은 우선순위 큐에 새로 도착한 패킷을 먼저 서비스 해주게 된다.

우선순위 큐잉 방식의 장점은 간단한 방법(우선순위가 높은 패킷을 우선 서비스)으로 차등화된 서비스를 제공할 수 있다는 것이다. 이 때문에 실시간 애플리케이션의 지원이 가능하다. 그러나 높은 우선순위 큐에 패킷이 계속해서 입력될 때, 낮은 우선순위 큐에 저장된 패킷이 서비스가 되지 못하는 아사(starvation) 현상이 발생되는 문제가 발생한다. 또한, 동일한 우선순위의 패킷만 많이 들어오는 경우에는 FIFO 큐처럼 동작이 되는 문제도 생길 수 있다.

커스텀 큐잉의 구현 원리

페어(fair) 큐잉이라고도 부르는 커스텀(custom) 큐잉은 앞서 설명한 우선순위 큐잉의 단점인 우선순위가 낮은 클래스의 패킷들이 우선순위가 높은 트래픽에 의해 아사되는 문제를 해결하기 위해 만들어졌다.

(그림 5)를 보면 커스텀 큐잉은 최대 16개의 FIFO 큐를 사용하는 것을 알 수 있다. 관리자가 각 큐에 대해 클래스를 구분해서 각각의 클래스마다 각각의 큐를 지정해 사용할 수 있다. 우선순위 큐에서 클래스를 4개로 나눠 분리하는 것보다 더 세분화해 분리할 수 있다는 것이 장점이다.

이렇게 세분화된 트래픽은 각각의 해당 큐로 보내져 라운드-로빈 스케줄링에 의해 하드웨어 큐(TX 큐)로 서비스된다. 즉, 모든 큐의 패킷이 공평하게 서비스되는 것이다. 우선순위 큐잉에서는 높은 우선순위 큐가 패킷을 갖고 있는 한,

그림 4 | 우선권 큐잉 스케줄링

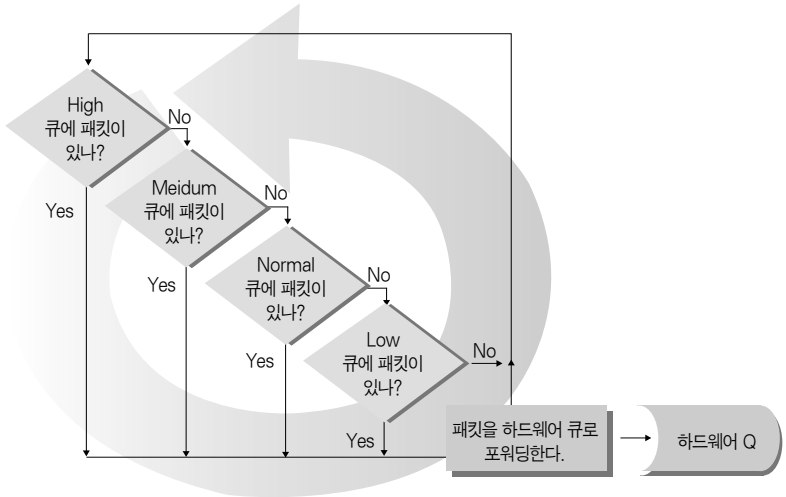
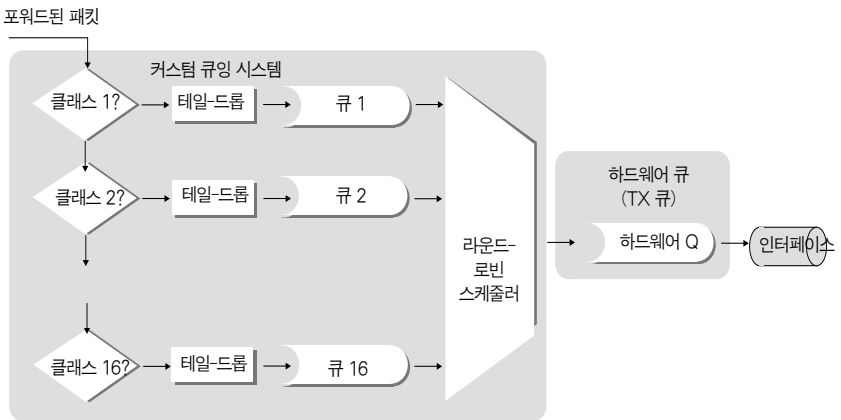
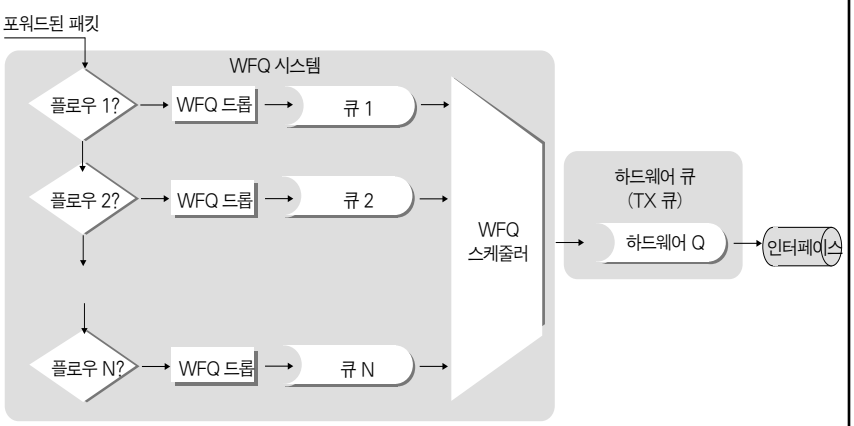


그림 5 | 커스텀 큐잉의 구현 원리



· 커스텀 큐잉은 16개의 FIFO 큐를 관리자가 정의해 사용할 수 있다.

그림 6 | WFQ의 구현 원리



· WFQ는 4096개의 FIFO 큐를 관리자가 정의해 사용할 수 있다.

낮은 우선순위 큐에 비해 절대적인 서비스 우선순위를 유지한다. 때문에 낮은 우선순위 큐가 아사 현상을 경험하게 되지만, 커스텀 큐잉에서는 모든 큐가 동일한 우선순위를 갖기 때문에 아사 현상은 발생하지 않게 된다. 그러나, 이 방식은 트래픽의 특성을 고려하지 않고 서비스 차원에서의 공정성만을 감안하고 있기 때문에, 스케줄링 차원에서 차등화된 서비스를 제공하는 것은 불가능하며, 관리자의 잘못된 설정에 의해 대역폭의 비효율적인 할당과 지연시간의 증가 등의 문제가 발생할 수 있다.

우선순위 큐잉과 커스텀 큐잉 방식의 장점만 결합한 WFQ

WFQ는 우선순위 큐잉의 단점인 우선순위가 낮은 클래스의 패킷들이 우선순위가 높은 트래픽에 의해 아사되는 문제를 해결하는 동시에 커스텀 큐잉 방식에서 차등화된 서비스를 제공하지 못하는 현상을 해소하기 위해 개발된 것이다.

(그림 6)을 보면 WFQ는 최대 4096개의 큐를 사용하는 것을 알 수 있다. 굉장히 많은 큐를 제공하기 때문에 클래스를 플로우 단위로 나눠 사용한다. 각각의 큐는 IP 우선권(precedence)을 기준으로 가중치(weight)를 받게 된다. 더 정확하게 말하면 IP 우선권의 크기에 반비례해 부여한다. 실제 패킷의 크기와 수식으로 계산을 해서 나온 가상 패킷의 크기를 기준으로 서비스되도록 스케줄링이 이뤄진다. 이 같은 수식을 적용하는 이유는 우선순위가 높거나 크기가 작은 패킷들에 대해 우선적으로 서비스하기 위함이다.

(그림 7)은 WFQ에서 어떤 방식으로 스케줄링이 이뤄지는지를 보여주는 그림이다. 큐 1에는 우선권 값이 5인 트래픽이 들어 있고, 큐 2에는 우선권 값이 0인 트래픽이 들어 있다. 실제 패킷의 크기는 큐 1에 있는 패킷이 더 큰 것을 알 수 있다. 이 같은 환경에서 큐의 실행은 다음과 같다.

- FIFO 큐잉으로 서비스를 한다고 하면 서비스되는 순서는 큐1_1 → 큐2_1 → 큐2_2 → 큐2_3 → 큐2_4 → 큐1_2가 된다.
- 우선순위 큐잉으로 서비스를 한다고 하면 우선순위가 낮은 큐 2에 있는 패킷은 큐 1에 계속 패킷이 들어오기 때문에 서비스되지 않는다. 이것이 앞서 말한 아사(starvation) 현상이다.
- 커스텀 큐잉으로 서비스를 한다고 하면 관리자의 설정(바이트 카운트, MTU)에 따라서 결과가 너무 차이가 나게 된다.
- WFQ를 이용해 서비스를 한다고 하면 '실제 패킷의 크기 / 우선권(precedence)에 의한 웨이트 = 가상 패킷의 크기' 의 공식을 적용해 계산된 가상 패킷의 크기를 구한 다음, 끝나는 시간이 적은 패킷을 우선해 서비스한다. (그림 7)에서의 결과를 보면 큐1_1 → 큐2_1 → 큐1_2 → 큐2_2 → 큐1_3 → 큐2_3 등의 순서로 서비스 됨을 알 수 있다. 즉, WFQ를 이용해 서비스를 하면 우선순위가 높은 패킷을 먼저 서비스하면서 우선순위가 낮은 패킷에 대해서도 서비스를 제공함을 알 수 있다.

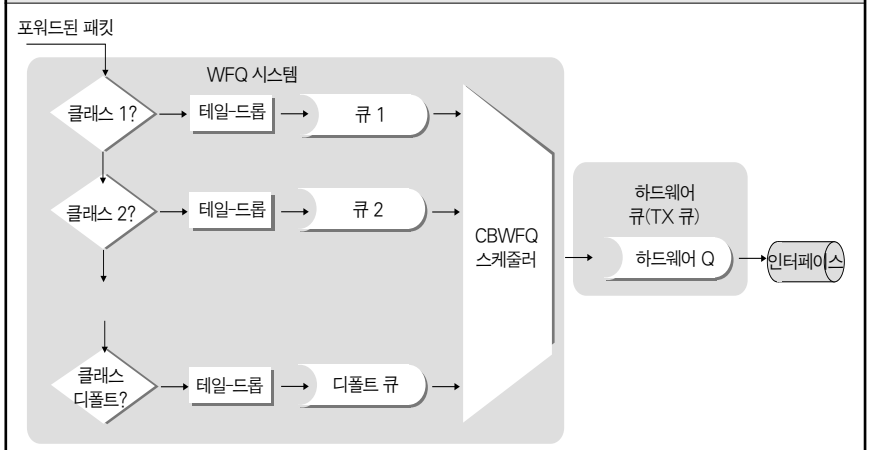
WFQ 개선시킨 3단계 큐잉 기법 'CBWFQ'

CBWFQ는 WFQ를 한층 발전시킨 모습으로, 관리자들이 설정하기 쉽게 3단계를 이용해 구현한다. 1단계에서는 클래스를 구분하고, 2단계에서는 정책을 적용하고, 3단계에서는 인터페이스에 설정을 한다.

CBWFQ는 커스텀 큐잉의 장점인 각각의 큐마다 최저 대역폭을 바이트 단위로 할당하는 기능을 발전시켜서 전체 대역폭의 %로도 최저 대역폭을 보장할 수 있게 했다. 하지만 기존의 WFQ가 플로우 기반으로 구현돼 큐가 많이 필요했던 것에 비해(4098개) CBWFQ는 클래스를 기준으로 큐를 구분하기 때문에 64개로도 설정이 충분하다. 드롭 정책에도 WFQ가 테일 드롭(tail drop) 밖에 사용하지 못했던 것에 비해 CBWFQ에서는 테일 드롭과 병행해 WRED(Weighted random

그림 7 | WFQ 스케줄링 상에서의 가중치

그림 8 | CB-WFQ의 구현원리



· CBWFQ는 최대 63 FIFO 큐(클래스)와 하나의 디폴트 클래스 큐를 사용한다.

rarly detection)을 사용해 글로벌 싱크로나이제이션(global synchronization) 문제를 최소화했다.

CBWFQ의 실제 적용

자, 이론 설명은 여기까지 마치고 지금부터는 연습문제를 통해 CBWFQ의 초급 설정에 대해 알아보자.

☞ 연습문제 1

(그림 9)에서 라우터 3는 프레임 릴레이 128kbps로 연결돼 있다. 다음의 조건을 만족하도록 라우터 3에 설정하라.

1. 모든 VoIP 트래픽은 VoIP 전용 큐를 사용해 서비스하라.
2. VoIP를 제외한 다른 트래픽은 일반 큐를 사용해 서비스하라.
3. 전체 대역폭의 50%를 VoIP 전용 큐에 할당하라.
4. 일반 큐는 WFQ를 사용하라.
5. CBWFQ(Class-Based WFQ)를 이용해 설정하라.

☞ 연습문제 1의 해결 방법

CBWFQ의 설정 순서는 다음과 같다.

1. Class-map match-all/any 명령어를 이용해 클래스를 설정한다.
2. policy-map 명령어를 이용해 적용할 정책을 설정한다.
3. 2번에서 만든 policy-map 이름을 적용하고자 하는 인터페이스에서 service-policy 명령어를 이용해 적용한다.

CBWFQ의 설정(초급)은 다음과 같다.

```
ip cef
!
class-map match-all voip-rtp        → Voip-rtp라는 이름의 Class-map을 설정(①)
  match ip rtp 16384 16383          → VoIP의 범위를 정함(클래스를 나누는 모습)
!
policy-map queue-voip               → Queue-voip라는 이름의 policy-map을 설정(②)
  class voip-rtp                    → ①에서 만든 Class-map을 불러옴
    bandwidth percent 50            → ①에서 만든 VoIP 클래스에 대역폭의 50%를 할당
  class class-default               → VoIP를 제외한 트래픽은 class-default로 정의
    fair-queue                      → 일반큐를 WFQ로 설정하는 모습
!
interface Serial0/0
```

그림 9 | CBWFQ 적용을 위한 실전문제(초급)

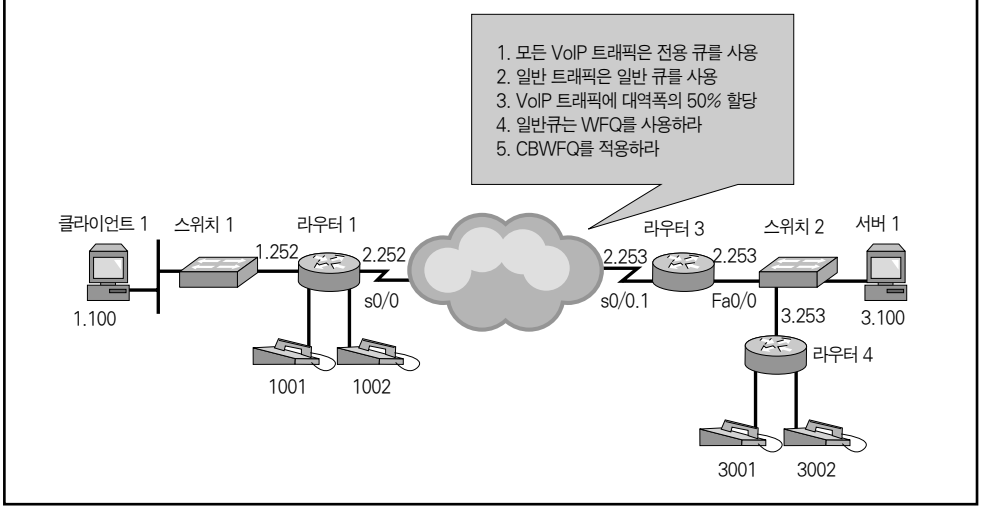
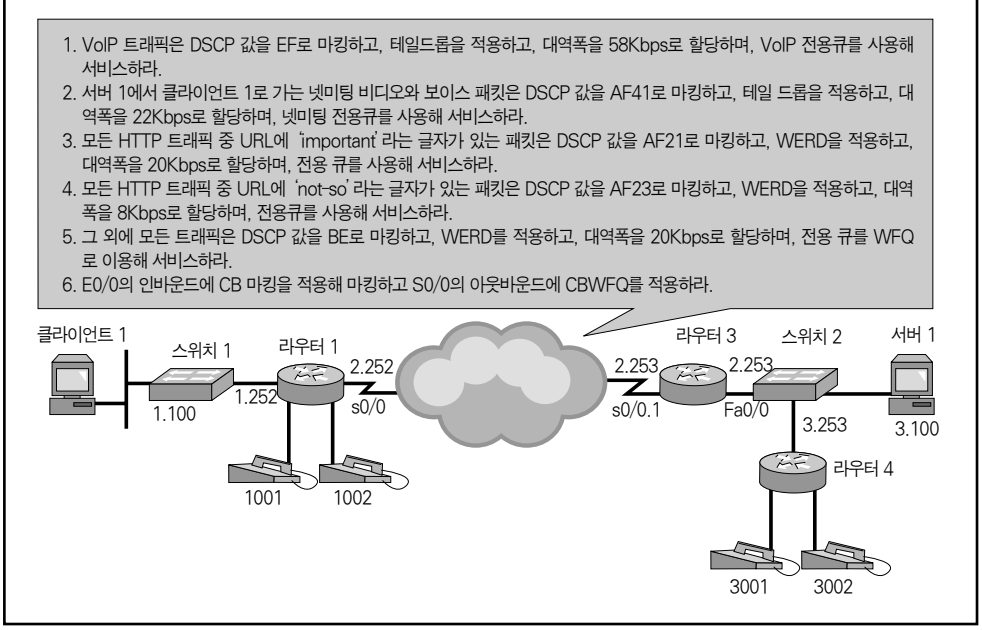


그림 10 | CBWFQ 적용을 위한 실전문제(고급)



```
 no ip address
 encapsulation frame-relay
 load-interval 30
 bandwidth 128
 service-policy output queue-voip → ②에서 만든 queue-voip를 인터페이스에 적용
(③)
 clockrate 128000
!
interface Serial0/0.1 point-to-point
 ip address 192.168.2.253 255.255.255.0
 frame-relay interface-dlci 101
```

CBWFQ 설정한 것을 확인하면 다음과 같다(초급).

```
R3# show policy-map int s0/0

Serial0/0

service-policy output: queue-voip

  Class-map:  voip-rtp  (match-all)
    136435 packets, 8731840 bytes
    30 second offered rate 51000 bps, drop rate 0 bps
    Match: ip rtp 16384 16383
    Weighted Fair Queueing
      Output queue: Conversation 256
      Bandwidth 50 (%) Max Threshold 64 (packets)
      (pkts matched/bytes matched) 48550/3107200
      (depth/total drops/no-buffer drops) 14/0/0
  Class-map:  class-default  (match-any)
    1958 packets, 1122560 byte
    30 second offered rate 59000 bps, drop rate 0 bps
    Match: any
    Weighted Fair Queueing
      Flow Based Fair Queueing
      Maximum Number of Hashed Queues 256
      (total queued/total drops/no-buffer drops) 15/0/0
```

연습문제 1은 초급단계라서 생각보다 쉽다고 느낄지도 모르겠다. 이번에는 조금 복잡한 문제를 풀어보자.

☞ 연습문제 2

(그림 10)에서 라우터 3는 프레임 릴레이 128kbps로 연결돼 있다. 다음의 조건을 만족하도록 라우터 3에 설정하라.

1. VoIP 트래픽은 DSCP→ EF 마킹, 테일 드롭 적용, 대역폭 58Kbps 할당, 전용 큐를 사용
2. 서버 1→ 클라이언트 1 넷미팅 트래픽은 DSCP → AF41, 테일 드롭 적용, 대역폭 22Kbps 할당, 전용 큐를 사용
3. URL에 "important" 트래픽은 DSCP → AF21, WRED 적용, 대역폭 20Kbps 할당, 전용 큐를 사용
4. URL에 "not-so" 트래픽은 DSCP → AF23, WRED 적용, 대역폭 8Kbps 할당, 전용 큐를 WFQ로 사용
5. 그 외 모든 트래픽은 DSCP → BE, WRED 적용, 대역폭 20Kbps 할당, 전용 큐를 WFQ로 사용
6. E0/0의 인바운드에 CB 마킹을 적용해 마킹하고 S0/0의 아웃바운드에 CBWFQ를 적용

☞ 연습문제 2의 해결 방법

연습문제 2의 설정 방법은 다음과 같다.

```
ip cef
!
class-map match-all dscp-ef        → Dscp-ef라는 이름의 Class-map을 설정(①)
 match ip dscp ef                  → 범위를 정함(클래스를 나누는 모습)
class-map match-all dscp-af41
 match ip dscp af41
class-map match-all dscp-af21
 match ip dscp af21
class-map match-all dscp-af23
```

용▸어▸설▸명▸

VoIP
Voice over IP. 패킷전송망인 인터넷을 통해 음성을 주고받는 기술의 통칭이다. 기존의 음성 서비스가 PSTN을 통해 전송되는 것과 달리 VoIP는 게이트웨이에서 음성 신호를 표준 규격(G.711, G.729A, G.723.1)에 맞게 압축해 상대 게이트웨이로 전송함으로써 음성 통화를 구현하게 된다. VoIP를 이용한 가장 대표적인 솔루션은 인터넷 전화로, 장거리 전화나 국제전화 등 통화 요금을 절감할 수 있는 기술로 각광받고 있다.

FIFO
First-In First-Out. 리스트나 테이블, 큐 등에서 어떤 요소를 저장하거나 꺼낼 때, 먼저 저장된 것을 먼저 꺼내는 방식을 가리키는 말. 큐(Queue)와 같은 말이며, 상대되는 용어로 LIFO(Last-In First-Out)이 있다. LIFO는 스택(stack)과 같은 말로 마지막에 저장된 것을 먼저 꺼내는 방식을 가리킨다.

```
  match ip dscp af23
class-map match-all http-impo      → Http-impo라는 이름의 Class-map을 설정(①)
  match protocol http url "*important*" → nbar를 이용해 url을 확인
class-map match-all http-not
  match protocol http url "*not-so*"
class-map match-all voip-rtp       → Voip-rtp라는 이름의 Class-map을 설정(①)
  match ip rtp 16384 16383         → 범위를 정함(클래스를 나누는 모습)
class-map match-all NetMeet        → NetMeet라는 이름의 Class-map을 설정(①)
  match access-group 101           → access-list를 이용해 클래스를 구분함
!
!
policy-map laundry-list            → Laundry-list라는 이름의 policy-map을 설정
(②)
  class voip-rtp                   → ①에서 만든 Class-map을 불러옴
    set ip dscp ef                 → dscp 값은 ef로 설정
  class NetMeet
    set ip dscp af41
  class http-impo
    set ip dscp af21
  class http-not
    set ip dscp af23
  class class-default
    set ip dscp default
policy-map queue-on-dscp           → Queue-on-dscp라는 이름의 policy-map을 설정
(②)
  class dscp-ef                    → ①에서 만든 Class-map을 불러옴
    bandwidth 58                   → 대역폭을 58Kbps로 할당
  class dscp-af41
    bandwidth 22
  class dscp-af21
    bandwidth 20
    random-detect dscp-based       → WRED를 설정함
  class dscp-af23
    bandwidth 8
    random-detect dscp-based
  class class-default
    fair-queue
    random-detect dscp-based
!
interface Ethernet0/0
  ip address 192.168.3.253 255.255.255.0
  ip nbar protocol-discovery
  half-duplex
  service-policy input laundry-list
   → ②에서 만든 laundry-list를 인터페이스에 적용(③)
!
interface Serial0/0
  no ip address
  encapsulation frame-relay
  load-interval 30
  bandwidth 128
  max-reserved-bandwidth 85
  service-policy output queue-on-dscp
   → ②에서 만든 queue-on-dscp를 인터페이스에 적용(③)
  clockrate 128000
!
interface Serial0/0.1 point-to-point
  ip address 192.168.2.253 255.255.255.0
  frame-relay interface-dlci 101
!
access-list 101 permit u에 host 192.168.3.100 range 16384 32767 192.168.1.0
0.0.0.255 range 16384 32767
```

조금 복잡하기는 하지만 천천히 확인해보면 그 동안 배웠던 설정으로 돼있음을 알 수 있다. 이번 시간에 배운 설정 내용을 완벽하게 이해할 수 있다면 QoS에 대한 내공이 상당하다고 자신해도 된다. 지금까지 4회에 걸친 강좌를 통해 QoS의 큰 골격에 대해서는 머리부터 발끝까지 살펴봤다. 다음 강좌부터는 세부적인 튜닝 부분과 왜 테일 드롭이 좋지 않은 것인지, 그리고 글로벌 싱크로나이제이션(global synchronization) 문제가 네트워크에 어떤 영향을 끼치는지에 대해 살펴보자. NET

용▸어▸설▸명▸

WFQ
Weighted Fair Queuing. 라우터에서 QoS를 보장하기 위해 사용하는 방식으로, 중요도가 높은 트래픽을 미리 설정해 두고, 그 중요도에 따라 서버가 받아들이는 요청의 수를 달리하는 패킷 스케줄링 기술이다. 때문에 혼잡 상태가 발생하더라도 기본적으로 예약된 플로우에게 충분한 대역폭을 제공해, 최소한의 지연을 넘지 않도록 한다. 표준 WFQ는 플로우에 기반한 큐를 단행한다.

DSCP
Differentiated Service Code Point. 이름에서도 알 수 있듯이 DSCP(Differentiated Service Code Point) 필드는 DiffServ 모델의 핵심이라고 할 수 있다. DSCP는 DiffServ 모델이 제안되기 이전에 사용됐던 우선권을 포함하면서 확장한 개념으로, 이는 기존 IP 체계에 큰 변화없이 헤더값에서 우선권과 ToS 필드가 사용하던 비트를 대체하는 방법이다. DSCP는 ▲디폴트 PHB ▲CS PHB ▲AF PHB ▲EF PHB의 4가지로 구분된다.