Understanding of QoS Technologies

김 학 용

코어세스 연구소

http://hakyongkim.net

1. 머리말

군사와 연구 목적으로 1970년대 중반부터 개발되기 시작한 인터넷은, 1990년대 초반부터 상업화의 길로 접어들면서 일반인들에게도 그 문호를 개방하기에 이르렀다. 그로부터 10년 정도가 지난 지금, 대부분의 국가에서는 인터넷이 유무선 전화와 더불어 보편적이며 없어서는 안 되는 통신 수단이 되어 있으며, 저개발 국가에서도 급속도로 확산 보급되고 있다.

초기 베스트 에포트(best-effort)라는 단일 서비스 모델을 사용하는 인터넷이 상업화와 더불어 일반인들 사이에서 사용되기 시작하면서 나타난 가장 두드러진 특징은 인터넷 트래픽의 폭발적인 증가와 그로 인한 네트웍 인프라의 진화로 요약될 수 있다. 인터넷 트래픽의 증가 원인은 단순히 인터넷 사용자의 증가라는 측면을 생각할 수 있으나, 더 나아가 다양한 인터넷 어플리케이션의 개발 및 다량의 데이터 전송을 요하는 어플리케이션의 출현 등에서 그 원인을 찾아볼 수 있다.

양적인 측면에 있어서의 인터넷 트래픽의 증가는 회선 및 스위칭/라우팅 장비의 확충을 통한 인프라의 고속·고성능화를 통해서 어느 정도 해결이 되었으며, 현재도 다양한 기술을 통한 고속화 및 고성능화가 진행중이다. 반면에, 질적인 측면에 있어서는 여전히 답보 상태에 있다. 즉 다양한 어플리케이션들이 생성해 내는 다양한 유형의 트래픽들은 아직도 베스트 에포트의 단일 서비스 모델에 의해 서비스 되고 있다. 다양한 유형의 트래픽들을 그 특성에 맞게 처리함으로써 질적인 향상을 꾀하기 위해서는 서비스 모델이 다양화 되고 다양한 서비스 모델을 통해 각 트래픽 특성을 보장해 줄 수 있는 기법들이 전제가 되어야 한다. 그러나, 이러한 기술들은 아직까지 개발 및 실제 적용을 위한 테스트 단계에 있다. 일반적으로 어플리케이션의 특성에 따라 트래픽을 차등화 해서 처리함으로써 차별화된 서비스 특성을 제공하는 것을 인터넷 상에서 QoS (Quality of Service)를 제공한다고 하거나 혹은 QoS를 보장한다고 한다.

이 글에서는 요즘 많은 관심을 끌고 있는 인터넷 QoS 기술에 대해서 살펴보고자 한다. 먼저, 개념적인 설명을 통해 QoS와 CoS 그리고 Traffic Engineering을 구분하고, 실제로 QoS가 적용되는 구조에 대해서 살펴볼 것이다. 그리고, QoS를 보장하기 위해 사용되는 다양한 QoS 기술들을 하나씩 살펴볼 것이다.

2. QoS vs. CoS vs. Traffic Engineering

인터넷 QoS가 많은 관심을 끌면서, QoS와 더불어 CoS 및 Traffic Engineering (TE)라는 용어도 자주 언급되고 있다. QoS도 최근에 새로 만들어진 용어가 아니지만, 이 두 용어는 QoS만큼 새로운 용어는 아니며 이미 일반적으로 사용되던 개념이다. 먼저, 이들 각각의 정의를 살펴보기로 하자.

QoS는 선택된 네트웍 트래픽 혹은 어플리케이션에 대해 더 나은 혹은 차별화 되는 서비스를 제공하는 네트웍의 능력을 말한다. QoS는 각 플로우 단위(per-flow)로 end-to-end로 보장되어야 한

다. 플로우라는 것은 사용자와 서버 혹은 사용자와 사용자 사이에 존재하는 어플리케이션의 세션을 말한다. 예를 들면, FTP를 사용해서 파일을 전송하는 경우, 파일이 전송되는 동안 하나의 세션이 존재하게 되고 이러한 세션을 통해 전달되는 데이터를 트래픽 플로우라고 한다. 각각의 플로우에 대해 QoS를 보장한다는 것은 각 플로우가 요구하는 트래픽 특성, 즉 대역폭이나 딜레이, 지터, 패킷 손실과 같은 성능 기준을 만족시켜 준다는 것을 말한다.[1] 이를 위해서는 서로 다른 플로우에 속한 트래픽들이 다른 취급을 받아야 한다. 여기에는 전달 우선순위(delivery priority)도 포함한다. QoS가 end-to-end로 보장된다는 것은 사용자와 서버 사이에 여러 개의 스위칭/라우팅 노드가 존재할지라도 네트웍 환경의 변화에 상관없이 특정 플로우에 대해 일관된 성능을 유지해 준다는 것을 말한다.

QoS가 end-to-end로 보장되어야 하는 것임에 반해, CoS (Class of Service)는 hop 단위로, 즉 노드와 노드 사이에서의 차별화를 말한다. 즉, 노드와 노드 사이에서의 트래픽의 전달 우선순위에만 관련된 것이다. 즉, 네트웍의 안쪽에 존재하는 코어 (core) 노드들은 에지 (edge) 노드에서 처리된 결과를 가지고 서로 다른 우선순위를 가지고 고속으로 패킷을 전달(forwarding)하는 역할을 하게 된다. CoS가 노드와 노드 사이에서 정의되는 개념이기 때문에, L2의 Data Link 계층에서 고려되는 것이라 생각할 수 있다. 그림 1은 이러한 QoS와 CoS의 개념상의 차이를 보여주고 있다.

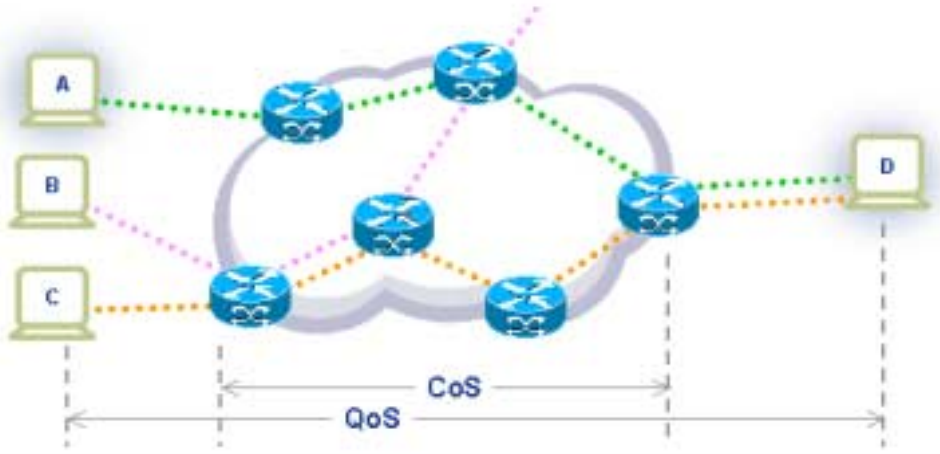


그림 1. QoS와 CoS의 구분: QoS는 플로우 단위로 end-to-end로 보장되어야 하는 트래픽 특성과 전달 우선순위를 말하며, CoS는 노드와 노드 사이(per-hop)에서 보장되는 전달 우선순위를 말한다.

QoS와 CoS가 트래픽의 특성 혹은 전달의 우선순위와 관련된 것이라면, TE는 트래픽 혹은 네트웍 대역폭의 균등 분배와 관련이 있다. 즉, 가용한 자원을 가진 링크로 트래픽을 보내어 네트웍에 균등하게 트래픽이 부가되도록 함으로써 네트웍의 효율성을 높이는 것을 TE라고 한다.

3. QoS의 필요성과 방법들

3.1 QoS의 필요성

앞에서도 언급되었지만, 인터넷이 상업화의 과정을 거치면서 다양한 어플리케이션이 출현하게 되었다. 이러한 어플리케이션들은 대부분이 여러 트래픽 유형을 한꺼번에 사용하는 멀티미디어 서비스나 실시간 처리를 필요로 하는 인터액티브 서비스가 대부분이다. 이들은 고속 처리는 물론, 많은 대역폭을 필요로 한다. 그러나, 네트웍 자원은 항상 여유롭지 못하며 충분한 네트웍 자원을 제공하는 것

[1] 대역폭, 딜레이, 지터, 그리고 패킷 손실과 같은 성능 측정 기준(performance metrics)에 대해서는 3장에서 자세히 다룰 것이다.

은 많은 비용을 필요로 한다. 이러한 어려움을 네트웍 자원을 효율적으로 사용함으로써 어느 정도 해결이 될 수 있으며, 이를 위해서는 다양한 QoS를 보장해 줄 수 있는 기술이 필요하게 된다. 트래픽 중에는 기존의 베스트 에포트 방식으로 서비스 되어도 상관없는 트래픽이 존재하는 반면, 실시간으로 처리해 주어야 하는 트래픽도 존재하며, 패킷 손실이 있어서는 안 되는 그런 트래픽도 존재한다. 이처럼 다양한 유형의 트래픽 특성을 만족시켜 주기 위해서는 트래픽을 구분하고 트래픽 유형에 다른 방식으로 처리를 할 수 있어야 한다.

3.2 QoS 접근 방법

기존의 베스트 에포트 네트웍에 QoS를 제공하는 방식은 네트웍의 대역폭을 증가시키는 방법과 복잡한 QoS 기술들을 사용하는 방식으로 나누어 생각해 볼 수 있다. 전자를 Big Pipe 접근 방법 혹은 Big Bandwidth 접근 방법이라 하며, 후자를 Controlled Bandwidth 접근 방법이라고 한다.

Big Bandwidth 방법은 네트웍의 대역폭을 증가시킴으로써 QoS의 문제를 해결하는 방법이다. 링크의 수를 늘리고 저속 장비를 고속 장비로 대체함으로써 간단히 해결할 수 있으나, 비용이 많이 드는 방법이다. 또한, 무한하게 대역폭을 증가시키지 않는 한 혼잡(congestion)을 비롯한 QoS 문제는 언제라도 발생할 수 있다. 따라서, QoS 문제를 해결하기 위해서는 반드시 필요하지만 궁극적인 해결 방법이라고는 할 수 없다.

Controlled Bandwidth 방법은 다양한 QoS 기술들을 사용해서 합리적인 수준의 대역폭을 제공하는 것과 동시에 트래픽을 적절하게 조절하는 방법이다. 이를 위해서는 DiffServ, IntServ, MPLS, IEEE 802.1p/Q와 같은 다양한 QoS 관련 프로토콜이나 표준 및 이들을 구체화 해주는 QoS 구현 기술들을 사용할 수 있다. QoS 구현 기술(enabling technology)로는 packet classification, mapping, metering, marking & remarking, rate limiting, shaping, flow 혹은 congestion control 등과 같은 것들이 있다. 그러나, 이 접근 방법 역시 효용성의 관점에서는 한계가 있다. 따라서, 현실적인 수준에서의 대역폭 증가와 효율적이고 효과적인 QoS 구현 기술들을 동시에 고려하는 것이 필요하다.

3.3 QoS 적용 구조

QoS의 적용은 베스트 에포트의 단일 서비스 모델을 멀티 서비스 모델로 전환하는 것을 전제로 한다. 이렇게 구분된 여러 서비스 모델은 각각 서로 다른 트래픽 클래스와 연결되며, 이들은 서로 다른 QoS 프로파일을 가지고 처리 된다. 일반적으로, 베스트 에포트의 단일 서비스 모델은 에지(edge) 혹은 경계(boundary) 노드에서 여러 개의 서비스 클래스로 구분이 되며, 에지 노드 및 코어(core) 노드에서 각각의 클래스에 상응하는 QoS 프로파일을 가지고 처리된다. QoS 구현 기술들이 에지 노드 및 코어 노드에서 적용되는 구조는 5장에서 설명된다.

베스트 에포트 네트웍에서 QoS를 지원하기 위해서는 그 네트웍을 구성하는 모든 노드가 QoS 기능을 가지고 있어야 한다. 만약, 중간의 어느 노드가 QoS를 지원하지 않고 오직 베스트 에포트 서비스만 한다면, 그 노드를 통과하는 트래픽들에 대해서는 어떠한 QoS 특성도 보장해 줄 수 없게 된다. 이런 관점에서, 사용자의 장비들도 QoS를 인식할 수 있는 것이어야 한다. 그러나, 현재 대부분의 사용자 장비는 베스트 에포트 방식으로 트래픽을 전송하고 에지 노드에서 이들 트래픽을 구분해 주고 있어서, 엄밀한 의미에서 end-to-end로 QoS를 보장하는 것은 불가능하다고 할 수 있다. 또한, 모든 장비들은 동일한 QoS 적용 구조를 가지고 있어야 일관된 QoS를 제공할 수 있다. 만약, 서로 다른 QoS 적용 구조를 가지고 있다면, 일관된 QoS 보장이 불가능해 진다. 이런 경우, 서로 다

른 QoS 적용 구조를 가진 네트웍 장비 혹은 서비스 도메인 사이에서 서로 다르게 정의되는 트래픽 클래스를 유기적으로 연결시켜주는 것이 필수적이다. 이러한 기능은 뒤에서 설명될 mapping 기능을 통해 이루어진다.

4. QoS 측정 기준 (QoS Metrics)

어떤 트래픽의 QoS 특성(QoS profile)은 여러 종류의 트래픽 파라미터에 의해 나타내어진다. 이처럼 트래픽의 QoS 특성을 나타내는데 사용되는 트래픽 파라미터를 QoS 파라미터라 하며, 대표적인 QoS 파라미터로는 대역폭(bandwidth), 딜레이(delay), 지터(jitter), 그리고 패킷 손실(packet loss)이 사용된다.

4.1 대역폭 (Bandwidth)

대역폭은 간혹 쓰루풋(throughput)과 혼용되어 사용되기도 하지만, 이들 사이에는 분명한 차이가 있다. 쓰루풋이 실제 입력된 트래픽에 대해 출력으로 얻어진 트래픽의 양으로 나타내어지는 것으로 단위가 존재하지 않지만, 대역폭은 특정한 어플리케이션 플로우에 할당된 네트웍 자원의 양을 말하는 것으로 Mbps나 Gbps와 같은 단위가 사용될 수 있다. 따라서, 대역폭을 보장해 준다고 하는 것은 약속된 속도(rate)를 보장해 주는 것을 의미한다.

대역폭과 관련된 QoS 파라미터에는 Sustained Data Rate, Peak Data Rate, Minimum/Maximum Data Rate이 있다. Sustained Data Rate은 Average data rate과 비슷한 개념으로 이해될 수도 있으나, 그보다는 Guaranteed Data Rate이나 Committed Data Rate처럼 항상 보장되어야 하는 개념으로 이해해야 한다. Minimum Data Rate은 최소한으로 보장되어야 하는 속도를 말한다. 그리고 Maximum Data Rate은 어떤 커넥션에 대해 보장해 줄 수 있는 최고 속도를 말한다. Peak Data Rate은 최대 허용 가능한 속도를 나타내며, Maximum Data Rate과는 달리 트래픽에 대한 보장의 개념을 포함하고 있지는 않다.

4.2 딜레이 (Delay/Latency)

딜레이는 레이턴시(latency)와 혼용되어 사용된다. 이들 사이에는 미묘한 차이는 있으나 같은 개념으로 이해해도 무리는 없다.[2] QoS 측면에서 말해지는 딜레이는 end-to-end 딜레이를 말하며, 흐름 제어와 관련된 경우는 Round-trip delay를 지칭하기도 한다.

End-to-end 딜레이는 serialization delay와 propagation delay, 그리고 switching delay의 합으로 나타내어진다. Serialization delay는 패킷을 링크 상에서 시리얼화 하는데 걸리는 시간이다. 1000 byte로 구성된 패킷을 100 Mbps 링크에서 시리얼화 하는 데는 8000 bits/(100×10^6 bits/sec) = 80 μsec가 걸린다. 즉, Serialization delay는 패킷의 길이에 비례하며 링크의 속도(대역폭)에 반비례한다. Propagation delay는 특정한 매체로 된 링크를 통해 한 비트를 전달하는데 걸리는 시간으로, 매체의 특성에 영향을 받으며 링크의 길이에 비례한다. Propagation delay는 매체의 길이를 매체를 통해 신호가 전송되는 속도로 나눔으로써 구해진다. 예를 들어, 광섬유를 통해 서울과 대전 사이를

[2] 딜레이는 어떠한 처리 혹은 서비스를 위해 기다림으로 발생하는 지연을 말하며, 레이턴시는 기다림으로 인해 발생하는 지연과 실제로 서비스를 받는 과정에서 발생하는 지연을 모두 포함한다. 큐잉 이론의 용어를 빌려 표현하면, 우리가 흔히 말하는 딜레이는 waiting delay에 해당하며, 레이턴시는 system delay (=waiting delay + service delay)를 말한다.

연결하는 링크의 경우, 거리는 160 Km, 광섬유를 통해 신호가 전달되는 속도는 2×10^5 Km/s이므로, Propagation delay는 $160/2\times10^5 = 800$ μsec가 된다. 마지막으로, Switching delay는 패킷이 스위치나 라우터 같은 장비에 입력되었다가 출력되는데 걸리는 시간으로 패킷 처리에 걸리는 시간 및 버퍼에서 기다리는 시간을 포함한다. 입력되는 트래픽의 양(부하), 시스템의 처리 능력, 그리고 큐잉 및 스케줄링 방식이 Switching delay에 가장 커다란 영향을 미치는 요소가 된다.

이곳에서 예로 든 것은 특정한 경우이기 때문에, Serialization delay가 Propagation delay보다 반드시 작다고 말할 수는 없다. 따라서, 과도한 딜레이가 발생하는 경우는 어떤 딜레이 요인에 의한 것인지 분석하는 것이 선행되어야 하며, 해당 부분을 향상시키려는 노력을 해야 한다. 그러나, Propagation delay는 물리적인 특성에 의해 결정되는 요인이기 때문에 줄일 수 없다.

4.3 지터 (Jitter)

일반적으로 지터는 어떤 신호가 네트웍을 통해 전달되면서 원래의 신호로부터 왜곡되는 정도를 나타내는데 사용되는 값이다. 그러나, 패킷 교환 네트웍에서는 전송될 때의 패킷들 사이의 시간에 대한 수신되는 패킷들 사이의 시간의 왜곡되는 정도를 나타내는 것으로 흔히 delay variation으로 말하여지기도 한다.

이런 왜곡 현상은 일반 데이터의 전송에는 아무런 영향을 미치지 않지만, 멀티미디어 트래픽에 치명적인 영향을 미친다. 지터를 줄이는 방법으로는 모든 패킷들이 일정한 딜레이를 갖도록 큐잉을 하게 되는데, 이는 지터는 줄이지만 추가적인 딜레이를 발생시키는 원인이 된다. 지터를 일으키는 원인에는 다음과 같은 것들이 있다.

- 큐 길이에 있어서의 변동
- 순서대로 도착하지 않은 패킷들을 재정렬 하는데 필요한 처리 시간에 있어서의 변동
- 소스에 의해 조각 내어진 패킷들을 재결합 하는데 필요한 처리 시간에 있어서의 변동

4.4 패킷 손실 (Packet Loss)

패킷 손실은 패킷을 전달하는 과정에서 발생하는 패킷 손실의 정도를 나타내는 것이다. 패킷 손실은 링크나 장비의 물리적인 결함에 의해 발생하는 손실도 생각할 수 있으나, 일반적으로 이러한 원인에 의해 발생하는 패킷 손실은 없다고 가정을 한다. 패킷 손실을 가장 많이 유발하는 원인은 혼잡에 의해 버퍼 오버플로우가 발생하거나, 흐름 제어 알고리즘이 임의적으로 패킷을 버리는 현상에 의해 발생할 수 있다. 일반적으로 버려지는 패킷들은 재전송되어야 하기 때문에, 네트웍의 혼잡을 가중 시키는 원인이 되기도 하며, 전체 전송 시간(딜레이)을 늘리는 주된 원인이 되기도 한다. 잘 관리되는 네트웍에서의 패킷 손실은 평균적으로 1% 미만이라고 말하여지지만, 실제로는 이보다 훨씬 낮은 값을 갖는다.

4.5 다른 QoS 측정 기준

앞에서 살펴본 네 가지 QoS 성능 측정 기준 이외에 패킷의 순서(packet order)나 네트웍 가용성(Availability) 등도 QoS 측정 기준에서 생각해 볼 수 있다. 이 중에서 패킷 순서는 딜레이 및 지터와 관련이 깊으며, 가용성은 네트웍의 신뢰성 측면에서 생각해 볼 수 있다. 가용성은 MTBF(mean time between failure)를 MTBF와 MTTR (Mean time to repair)을 합한 것으로 나눈 것이며, 한달 혹은 일년을 단위로 측정된다. 일반적으로 99.999% 이상의 가용성을 보장해 주어야 한다.

5. QoS 구현 기술 (QoS Enabling Technologies)

QoS를 구현하는 방법은 다양한 QoS 프로토콜이나 표준을 사용함으로써 이루어진다. 대표적인 QoS 관련 프로토콜 및 표준에는 ATM, DiffServ, IntServ, RSVP, MPLS, IEEE 802.1p/Q 등이 있다. 이 중에서 ATM을 제외한 프로토콜/표준이 IP 네트웍에서 QoS를 제공하는데 사용되고 있다. 이러한 프로토콜/표준들은 그 자체로써 QoS 제공 기능을 갖는 것이 아니며, 다양한 QoS 구현 기술과 이들을 적용하는 방식 혹은 정책들로 이루어져 있다.

지금까지 많은 문서에서 QoS와 관련된 프로토콜들이 논의되었으므로, 이 문서에서는 그러한 프로토콜이 실제로 구현 가능한 것이 되도록 만드는 QoS 구현 기술에 초점을 맞추려 한다. 이 절에서는 이러한 다양한 QoS 관련 프로토콜/표준을 구현하는데 사용되는 QoS 구현 기술들을 살펴볼 것이다. 먼저, 에지 노드 및 코어 노드에서 QoS 구현 기술들이 적용되는 구조를 설명한 후 각각의 기술들을 설명하도록 할 것이다.

5.1 QoS 구현 기술 적용 구조

3절에서도 언급된 것처럼, 에지 노드와 코어 노드에서 수행되는 QoS 기능은 약간의 차이가 있다. 에지 노드에서는 네트웍으로 입력되는 트래픽을 다양한 기준에 따라 여러 개의 트래픽 클래스로 구분을 하며, 구분된 트래픽 클래스에 대해 각기 다른 QoS 프로파일을 가지고 트래픽을 처리하게 된다. 코어 노드에서는 에지 노드에서 처리된 결과를 가지고 서로 다른 우선순위를 가지고 고속으로 패킷을 전달(forwarding)하는 역할을 주로 한다. 그림 2와 그림 3은 에지 노드와 코어 노드에서 QoS 구현 기술이 적용되는 구조를 보여주고 있다. 그림대로라면 미터링, 마킹, 그리고 쉐이핑 혹은 폴리싱 기능(이들 모두를 트래픽 컨디셔닝이라고 한다)의 수행 여부가 눈에 보이는 차이이겠지만, 에지 노드와 코어 노드에서의 QoS 적용 구조의 가장 큰 차이는 코어 노드에서는 패킷 클래시피케이션(packet classification)과 패킷 클래시피케이션 이후에 오는 트래픽 컨디셔닝을 하지 않는다는 것이다. 이는 코어 노드가 에지 노드에서의 패킷 클래시피케이션의 결과를 그대로 사용함을 의미하는 것이다. 코어 노드는 패킷/프레임에 대해 많은 처리를 수행해야 하는 패킷 클래시피케이션과 트래픽 컨디셔닝을 수행하지 않음으로써 고속으로 패킷을 전달(forwarding)하는 것이 가능하게 된다.

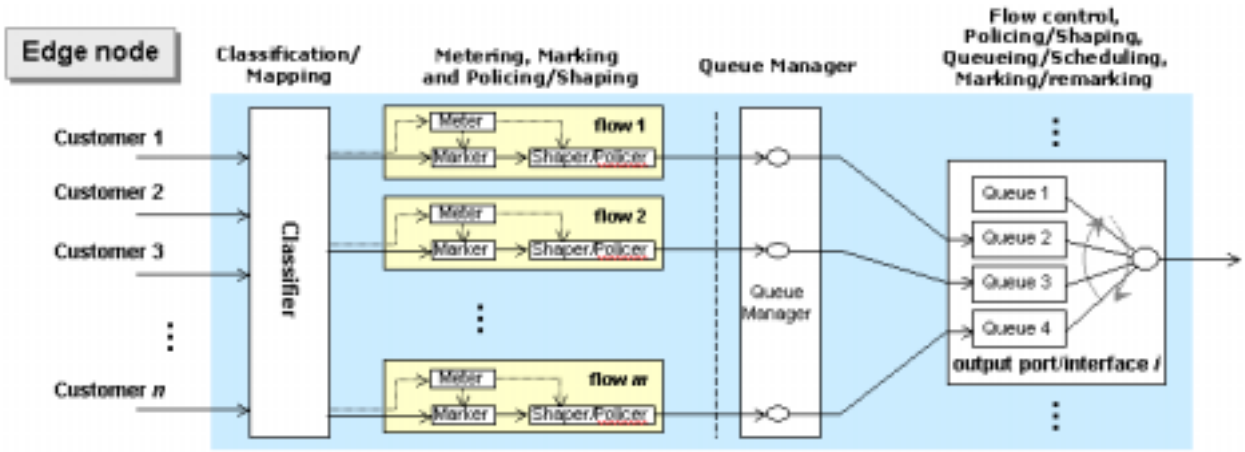


그림 2. 에지 노드에서의 QoS 구현 기술의 적용 구조.

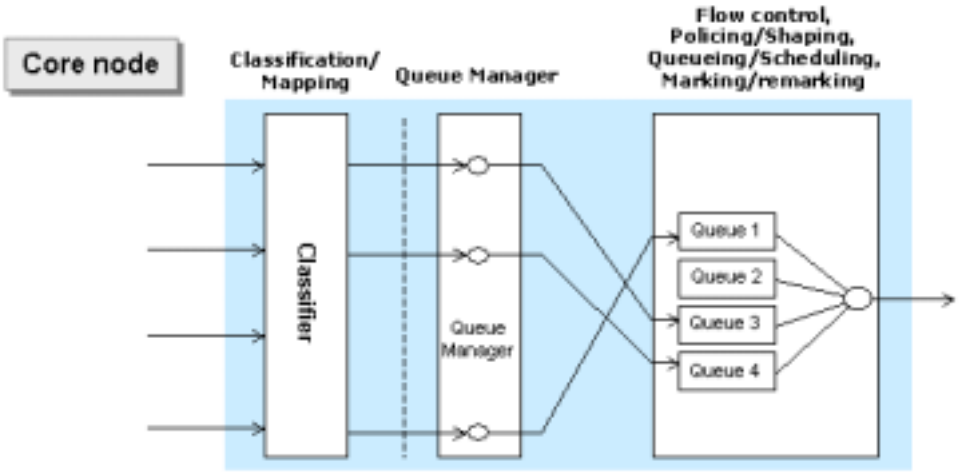


그림 3. 코어 노드에서의 QoS 구현 기술의 적용 구조.

각각의 QoS 구현 기술을 설명하기에 앞서, 에지 노드 및 코어 노드에서 QoS 기술이 적용되는 구조를 간단히 살펴보자. 먼저 에지 노드에서는 들어온 사용자 패킷들(트래픽 플로우)이 클래시파이어(classifier)를 통과함으로써 여러 개의 클래스로 분류된다. 즉, 도착한 패킷이 어떤 클래스 큐에 저장이 될지를 결정하게 된다. 좀 더 구체적으로 이야기를 하면, 입력된 트래픽들은 액세스 컨트롤 리스트(access control list)에서 정의된 플로우 단위로 구분이 되며, 이와 동시에 플로우가 속하게 될 트래픽 클래스가 결정이 된다. 트래픽 클래스가 결정된다는 것은, 입력된 트래픽이 실제로 저장될 큐와 서비스(스케줄링)되는 방식이 결정됨을 의미한다. 클래스가 결정된 패킷들은 각 트래픽 플로우에 할당된 미터(meter)를 통해 도착한 트래픽 플로우의 특성을 측정하게 된다. 측정된 결과는 사전에 약속된 QoS 트래픽 특성과 비교 되며, 그 결과에 따라 몇 가지 우선순위로 마킹(marking)된다. 마킹된 패킷들은 쉐이퍼(shaper)나 드라퍼(dropper)를 거치면서 사전에 약속된 트래픽의 대역폭 특성에 맞도록 조절된다. 일반적으로 쉐이핑 기능보다는 드라퍼를 사용한 폴리싱(policing) 혹은 레잇 리미팅(rate limiting) 기능이 수행된다. 미터, 마커, 그리고 쉐이퍼/드라퍼로 구성된 모듈을 트래픽 컨디셔너(traffic conditioner)라 하며, 이들 모두가 하나의 기능 모듈로 구현된다. 트래픽 컨디셔너에서는 경우에 따라 흐름 제어(flow control)도 수행할 수도 있다. 트래픽 컨디셔너를 통과한 패킷들은 큐 매니저(queue manager)를 거치며 클래시파이어에서 결정된 자신의 클래스에 맞는 큐에 저장이 된다. 큐에 저장된 패킷들은 스케줄링 과정을 통해 출력 링크로 내 보내어진다. 흐름 제어 및 쉐이핑 기능은 이곳에서 수행된다.

에지 노드의 기능이 비교적 복잡한데 반해, 코어 노드의 기능은 단순하다. 앞에서도 설명된 것처럼 코어 노드는 에지 노드의 클래시피케이션 결과를 이용해 고속으로 패킷을 전달하는 것이 주된 임무이며, 혼잡 문제와 관련된 기능들만이 수행되기 때문이다. 따라서, 그림에 보이는 것처럼 코어 노드에는 트래픽 컨디셔너 부분이 존재하지 않는다. 또한, 패킷 클래시파이어의 기능도 현저하게 단순화된다. 먼저 코어 노드로 들어온 패킷은 클래시파이어를 통과하게 된다. 에지 노드의 클래시파이어는 패킷의 다양한 필드 값 및 다양한 조건들을 바탕으로 패킷의 클래스를 결정하지만, 코어 노드의 클래시파이어는 L2 프레임의 CoS 필드나 L3의 ToS 필드의 IP Precedence 값 혹은 DS 필드의 DSCP 값 중 하나만을 참조해서 패킷의 클래스를 결정하게 된다. 에지 노드에서 복잡한 과정을 거쳐 패킷 클래시피케이션을 수행한 결과가 상기와 같은 필드에 저장이 된다. 이렇게 분류된 패킷들은 트래픽 컨디셔닝 과정을 거치지 않고 큐 매니저에 의해 해당 클래스 큐에 저장이 되고, 흐름 제어 및 스케줄링 과정을 거쳐 출력 링크로 보내어진다.

5.2 패킷 클래시피케이션 (Packet Classification)

패킷 클래시피케이션은 말 그대로 입력된 패킷을 어떤 기준에 따라 여러 개의 클래스로 구분을 하는 기능을 가리킨다. 용어에 분류(classify)라는 말이 들어가기 때문에 패킷을 여러 종류로 나누어 복잡하게 처리하는 것으로 잘못 이해하는 경우가 많은데, 사실은 여러 종류의 패킷을 제한된 몇 개의 클래스로 분류하는 개념으로 이해해야 한다. 이것은 패킷 클래시피케이션의 목적이 동일하거나 유사한 특성을 갖는 패킷들을 함께 처리함으로써 QoS 구조를 단순하게 하자는 데에 있다. 즉, 다양한 QoS 특성을 갖는 트래픽들을 각각 처리해 주게 된다면 수없이 많이 종류의 패킷 처리 기준이 존재해야 한다. 이것은 현실적으로 불가능하며, 가능하다 할지라도 노드에 커다란 부하를 가하게 된다. 따라서, 제한된 클래스로 패킷을 나누어 처리하는 것이다.

5.1절에서도 간단히 살펴 보았지만, 에지 노드와 코어 노드에서 수행되는 패킷 클래시피케이션의 기능에는 다소 차이가 있다. 에지 노드에서는 여러 기준에 의해 패킷들을 제한된 수의 클래스로 분류를 하게 된다. 제한된 수의 클래스로 분류한다는 것은 입력된 트래픽이 저장될 큐를 결정한다는 것을 의미한다. 그러나, 실제로 큐에 저장되기 전까지는 액세스 컨트롤 리스트에서 정의된 플로우 단위(per-flow)로 모든 처리가 이루어진다. 반면에 코어 노드에서는 에지 노드에서 분류된 결과를 이용하거나, 다른 QoS 처리 과정에서 패킷의 클래스가 변경된 것을 반영하게 된다. 즉, 에지 노드에서는 다양한 종류의 플로우를 몇 개의 트래픽 클래스로 분류를 하게 되며, (만약 에지 노드와 동일한 QoS 적용 구조를 가지고 있다면) 코어 노드에서는 에지 노드에서 분류된 적은 수의 클래스를 그대로 유지하게 되는 것이다. 따라서, 에지 노드에서 수행되는 패킷 클래시피케이션이 코어 노드에서 수행되는 것보다 훨씬 복잡하고 노드에 많은 부하를 주게 된다. 에지 노드에서 사용되는 패킷 클래시피케이션 기준에 대해서는 5.2.2절에서 자세히 알아볼 것이다.

5.2.1 클래스의 종류

패킷 클래시피케이션을 수행하기 위해서는 클래스의 종류와 클래스를 결정하기 위한 기준이 있어야 한다. 절대적인 것은 아니지만, 패킷의 클래스는 패킷의 우선순위와 비슷한 것으로 이해할 수 있다. 따라서, 클래스의 개수는 사용되는 프로토콜의 종류에 따라 달라질 수 있다. 또한, 5.2.2절에서 설명될 여러 가지 값들을 조합해서 클래스를 결정할 수 있기 때문에, 클래스의 개수는 무한하게 많아질 수도 있다. 그러나, 클래시피케이션이 QoS 처리를 단순화 하려는데 있기 때문에, 일반적으로 8 이하의 적은 수의 클래스가 사용된다.

잘 알려진 프로토콜 중의 하나인 DiffServ는 6 비트의 DSCP 필드 값을 사용해서 패킷들을 최대 64개의 클래스로 구분할 수 있다. 그러나, 실제로는 32개의 표준 PHB가 클래스로 사용될 수 있으며, 이중에서 EF (Expedited Forwarding), AFx, 그리고 BE 등 6 개의 클래스가 많이 사용된다.[3] IP 프로토콜의 경우는 ToS 필드의 3 비트 IP Precedence 값을 기본적인 클래스 구분의 기준으로 사용하며, IEEE 802.1p/Q 표준을 사용하는 L2 MAC 프레임의 경우는 3 비트의 Priority 필드를, MPLS 프로토콜의 경우는 역시 3 비트의 Exp 필드를 사용한다. 이것은 최대 8개의 클래스를 사용할 수 있음을 의미한다. 그러나, 실제의 경우에 있어서는 적게는 2개에서 많게는 8개 까지의 클래스

[3] AF (Assured Forwarding)에는 4 개의 클래스가 존재하며, 각 클래스에 대해 3개의 서로 다른 drop precedence를 가지게 된다. 각 클래스 별로 서로 다른 drop precedence를 갖는 패킷들을 별도의 클래스로 구분할 수도 있다. 이런 경우, DiffServ는 14 개의 클래스를 사용하게 된다.

를 사용하고 있으며, 가장 일반적인 경우가 4개의 클래스를 사용하는 것이다. 큐잉과 관련된 여러 연구들을 살펴보면, 큐나 클래스의 개수가 1개에서 2개로 증가할 때 복잡해 지는 것에 대한 성능 향상의 효과가 가장 크며, 2개에서 4개로 증가할 때도 받아들일 수 있는 수준이 된다. 그러나, 4개에서 8개 혹은 그 이상으로 증가할 때는 복잡도는 배로 증가하지만, 성능 향상은 수 % 이내로 제한되어 향상되기 때문에 많은 수의 클래스를 두는 것이 항상 좋다고 할 수는 없다. 따라서, 이곳에서는 3개 혹은 4개의 클래스를 자주 사용할 것이다.

이번에는 실제로 사용되는 클래스의 종류를 살펴보기로 하자. 가장 많이 알려진 클래스의 종류는 DiffServ 관련 RFC에 나오는 올림픽 모델이다. 즉, 클래스를 Gold, Silver, 그리고 Bronze로 나누는 것이다. 이와 비슷하게 많이 사용되는 것은 클래스를 High, Medium, Low로 구분하는 것이다. 일반적으로, 이렇게 세 종류로 구분을 하기도 하며, 경우에 따라서는 BE (Best Effort) 클래스를 포함시키기도 한다. BE 클래스가 포함되지 않는 경우는 Bronze나 Low 클래스가 BE 클래스에 해당하게 된다. 또한, Committed, Premium, Business, Standard와 같이 구분할 수도 있다. 그리고, 잘 알고 있는 것처럼 ATM의 경우는 CBR, VBR, VBR+ , ABR, UBR의 5 가지 클래스로 구분하고 있다.

일반적으로 이들 클래스 사이에는 어떤 우선순위를 두어 구분해서 사용하고는 있으나, 반드시 절대적인 우선순위에 의해 구분될 필요는 없다. 이것은 클래스라는 것은 특정한 QoS 특성을 가진 패킷들의 집합일 뿐이며, 그러한 QoS 특성이 다른 클래스의 QoS 특성에 대해 절대적인 우선순위를 갖는 것이 아니라는 사실에서 확인된다. 그러나, 서비스(스케줄링) 과정에서는 서비스의 순서를 결정할 필요가 있기 때문에, 클래스에 대한 우선순위를 고려하게 된다. 그림 4는 사용자 트래픽을 3 가지 클래스로 분류하는 예를 보여주고 있다. 음성 트래픽을 High 클래스에, 비디오 트래픽을 Medium 클래스에, 그리고 일반 데이터 트래픽을 Low 클래스로 분류하고 있다.

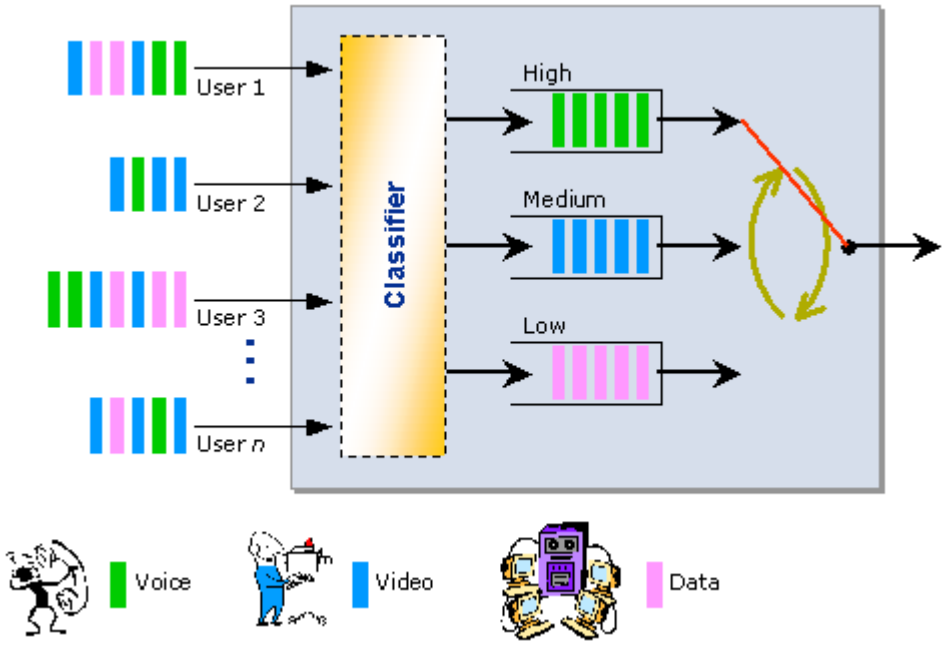


그림 4. 패킷 클래시피케이션의 예.

5.2.2 패킷 클래시피케이션 기준

패킷의 구분은 여러 가지 기준을 사용해서 수행할 수 있다. 패킷이 입력된 장비의 포트/인터페이스를 기준으로 패킷을 구분할 수도 있으며, MAC이나 IP 같은 주소 정보를 사용할 수도 있으며, 프로

토콜에서 정의된 패킷 헤더의 필드 값들을 사용할 수도 있다. 이들 중 어느 하나만을 이용해서 패킷을 분류할 수도 있으며, 여러 필드들의 조합을 통해 패킷을 분류할 수도 있다. 후자의 경우처럼 패킷 헤더의 여러 필드 정보를 이용해서 패킷을 분류하는 것을 Multi-Field Packet Classification (MFPC or MFC) 이라고 한다. 또한, 사용자가 가입한 서비스 종류나 사용자의 소속 등과 같은 사용자 정보를 바탕으로 패킷을 분류할 수도 있다. 일반적으로, 사용자 정보를 사용해서 패킷을 분류하는 경우는 사용자 인증 절차를 필요로 한다. 좀 더 현실적인 예를 들자면, www.corecess.com과 같은 인터넷 접속 주소나 응용 프로그램(어플리케이션)의 종류에 따라 패킷을 분류할 수도 있다. 이와 같은 것은 어플리케이션 레이어(application layer)에서 이루어지는 패킷 클래시피케이션의 예가 된다. 이 절에서는 패킷 클래시피케이션의 기준에 대해 살펴볼 것이다. 그러나, 사용자 정보를 이용하거나 어플리케이션 레이어에서 수행되는 것은 다루지 않고, OSI 7 계층 중에서 L2에서 L4 사이의 정보를 사용해서 패킷을 분류하는 것에 대해서 살펴보기로 한다. L2에서 L4 사이의 정보를 이용하기 위해서는 일반적으로 많이 사용되는 L2~L4의 프레임 혹은 패킷들의 헤더 구조를 알아야 한다.

그림 5는 L2에서 사용되는 대표적인 프레임들의 구조를 보여주고 있다. 가장 위에 보이는 프레임은 일반적으로 사용되는 Ethernet 프레임의 구조이며, 중간에 보이는 것은 IEEE 802.1p/Q 프레임 구조에 해당한다. 그리고, 가장 아래에 보이는 프레임 구조는 MPLS over Ethernet의 프레임 구조를 보여주고 있다.

먼저 Ethernet 프레임 구조를 보자. 맨 위에 있는 그림에서 패킷을 구분할 수 있는 정보를 담고 있는 필드는 처음 세 개의 필드, 즉 Destination MAC Address (DA MAC) 필드와 Source MAC Address (SA MAC) 필드, 그리고 Type 필드다. DA MAC 필드와 SA MAC 필드는 장비의 고유한 값에 해당하며, Type 필드는 그림에서 IP Datagram이라 표시된 페이로드(Payload) 필드에 포함되는 패킷의 종류(프로토콜)를 나타낸다. 이들 정보를 사용해서 패킷들을 분류하는 예를 생각해 보자. 먼저 생각할 수 있는 것은 특정한 호스트에서 들어오는 프레임들을 따로 분류하기 위해서 SA MAC 정보를 사용할 수 있다. 마찬가지로 특정한 호스트로 향하는 프레임들을 따로 분류하기 위해 DA MAC 정보를 사용할 수 있다. 특정한 프로토콜을 사용하는 패킷들을 따로 분류하기 위해서 Type 필드의 정보를 사용할 수도 있다. 특정한 SA MAC에서 특정한 DA MAC으로 향하는 특정한 프로토콜에 속한 패킷을 따로 분류하기 위해서 세 개의 필드를 조합해서 한꺼번에 사용할 수도 있다.

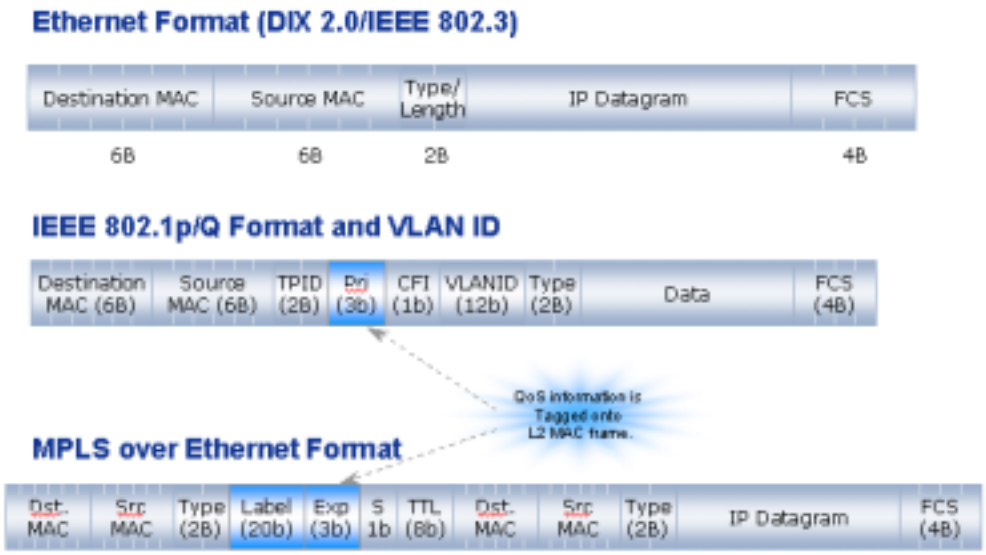


그림 5. L2 프레임 구조.

그림 5의 중간에 보이는 IEEE 802.1p/Q 프레임의 경우, DA MAC, SA MAC, Type 필드 외에 TPID나 PRI, CFI, VLAN ID 필드가 존재함을 알 수 있다. 이 중에서 TPID 필드나 CFI 필드는 각각의 목적을 가진 필드이므로 더 이상 언급하지 않기로 하며, Priority 필드와 VLAN ID 필드를 보자. Priority 필드는 3비트로 패킷 우선순위를 8 개의 레벨로 분류할 수 있도록 해 준다. 즉, 이 필드의 값에 따라 L2 프레임을 차별화 해서 처리해 줄 수 있음을 말한다. VLAN ID 필드는 이름 그대로 VLAN ID 값이다. 즉, 프레임이 어떤 VLAN에 속해 있는지를 구분 짓는데 사용된다. 이는 VLAN의 종류에 따라 패킷들을 달리 처리할 수 있음을 말한다. Priority 필드와 VLAN ID 필드의 값들은 DA MAC, SA MAC, 그리고 Type 필드의 값들과 조합해서 프레임들을 상세하게 분류할 수도 있다.

그림 5의 아래에 보이는 것은 MPLS over Ethernet 프레임 구조를 보여준다. 이 프레임 구조 역시 기본적으로 Ethernet 프레임에 사용되는 필드와 더불어, 20 비트의 Label 필드 그리고 3 비트의 Experimental 필드를 가지고 있다. IEEE 802.1p/Q 프레임 구조와 비교할 때, Label 필드는 VLAN ID 필드에 상응하며 Experimental 필드는 Priority 필드에 상응한다. 즉, Label은 MPLS에서 사용되는 커넥션의 패쓰(path)를 구분 지어주며 Experimental 필드는 프레임의 우선순위를 나타내는데 사용된다. 이들 역시, 다른 Ethernet에 사용되는 필드 값들과 조합해서 사용될 수 있다.

그림 5의 첫번째 프레임 구조와 두 번째 그리고 세 번째 프레임 구조를 비교했을 때 가장 커다란 차이는 두 번째 및 세 번째 프레임 구조에서는 프레임의 우선순위를 나타낼 수 있는 필드를 가지고 있다는 것이다. 실제적으로 이 필드의 값들로 인해 순수한 Ethernet 프레임에 대해서도 차등화된 서비스를 제공할 수 있게 되는 것이다. 이 값들은 L3 IP 패킷과의 우선순위 매핑(mapping)에도 사용된다.

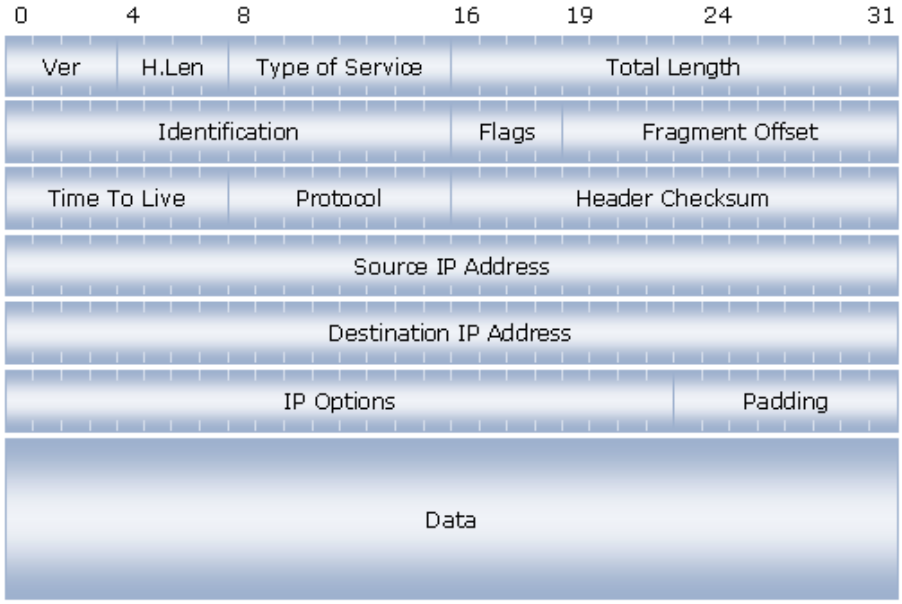


그림 6. IP 패킷 구조.

그림 6은 L3의 IP 패킷 구조를 보여주고 있다. L2 MAC 프레임처럼 Source IP Address와 Destination IP Address가 패킷을 구분하는데 사용될 수 있다. 또한 Type of Service (ToS) 필드와 Protocol 필드가 패킷을 구분하는데 사용된다. ToS 필드는 8비트로 이중 처음의 3 비트에 해당하는 IP Precedence 필드는 패킷의 우선순위를 나타내는데 사용된다. 따라서, L2 프레임의 Priority 필드나 Experimental 필드와 1대 1로 매핑이 해서 사용할 수 있다. Protocol 필드는 Data 필드에 사용되는 L4 프로토콜의 종류를 나타내는데 사용된다.

그림 7은 IP 패킷 헤더의 ToS 필드의 구조와 DiffServ의 DS 필드를 보여준다. 그림에 보이는 것처럼 8비트의 ToS 필드는 3 비트의 IP Precedence 필드와 DTRC의 라우팅에 사용되는 정보 필드를 가지고 있다. 이 필드는 DiffServ에서 DS 필드로 다시 정의 되었는데, DS 필드는 6비트의 DSCP (DiffServ Code Point) 필드와 사용되지 않는 CU (Currently Unused) 필드로 되어 있다. DSCP 필드의 처음 3 비트는 클래스를 나타내는데 사용되며 다음의 두 비트는 Drop Precedence를 나타내는데 사용된다. 처음 3비트는 IP ToS 필드의 IP Precedence와 호환된다. 따라서, L2 프레임과 L3 패킷들 사이에는 우선순위를 일관되게 유지할 수 있다.

그림 7. IP ToS 필드 및 DiffServ의 DS 필드

L4에 사용되는 패킷 중에는 TCP와 UDP 패킷들이 대부분이다. 이들은 패킷 헤더에 Source Port Number와 Destination Port Number 필드를 가지고 있어 패킷들을 분류하는데 사용될 수 있다. 다른 L4 프로토콜 패킷들도 각각 독특한 패킷 구조를 취하고 있으며, 특정한 필드 값을 사용해서 동일한 프로토콜에 속한 서로 다른 패킷들을 구분하게 된다. 일반적으로, L4 정보는 L3의 Protocol 정보와 함께 사용된다.

6. Mapping

매핑(mapping)이란 서로 다른 계층(layer), 서로 다른 프로토콜, 서로 다른 서비스 도메인들 사이에서 QoS 특성을 서로 연결해 주는 기능을 말한다. QoS 특성이라는 것은 좁은 범위에서는 QoS 성능 측정 기준(QoS metrics)이 될 수도 있으나, 이 보다는 서로 다른 영역에서 패킷/프레임의 우선순위나 클래스 혹은 서비스들의 특성을 나타낸다.

QoS 구조에서 매핑이 필요한 이유는 노드의 동작이라는 측면과 네트웍 전체 QoS의 제공이라는 측면에서 생각해 볼 수 있다. 앞에서도 언급된 것처럼 QoS는 end-to-end로 보장되어야 하며, 이를 위해서는 두 종단 시스템(end systems) 사이에 존재하는 모든 네트웍 엘리먼트(network elements)들이 상호 유기적인 구조로 연결되어서 네트웍 엘리먼트들 사이에 일관된 QoS 특성을 유지할 수 있도록 해야 한다. 이것이 네트웍 전체 QoS의 제공이라는 측면에서의 매핑의 필요성에 설명한다. 노드의 동작 측면에서 매핑이 필요한 이유는, end-to-end로 QoS 특성을 유지하기 위해서는 각 네트웍 엘리먼트 내부에서도 QoS 특성이 일관되게 유지되도록 해 주어야 하기 때문이다. 즉, 하위 계층에서 제공되는 QoS 특성을 상위 계층의 성능 인자(performance indicators)에 영향을 미치도록 하는 구조가 필요한 것이다. 이와 반대로 상위 계층에서 요구되는 QoS 특성이 하위 계층의 성능 인자에 영향을 미치도록 하는 구조도 필요하다.

QoS의 특성은 어플리케이션 계층에서 구체화가 되지만, 이러한 어플리케이션 트래픽의 QoS 특성에 영향을 미치는 동작은 주로 L2와 L3에서 일어난다. 따라서, 매핑 동작은 L2에서 L3 방향으로 혹은 L3에서 L2 방향으로 이루어질 수 있다. 경우에 따라서는 L3와 L3처럼 같은 계층에서도 매핑 관계가 정의될 수 있다. 아래 그림 8에 보이는 예는 L2-L3, L3-L3, 그리고 L3-L2 사이의 매핑 관계를 보여주고 있다. L2와 L3 사이에서는 L2의 CoS(IEEE 802.1p의 Priority 혹은 MPLS의 Experimental 필드 값) 값과 L3의 DSCP값 사이의 매핑 관계를 보여주고 있다. IP Precedence와 DSCP 사이의 매핑 관계는 동일한 L3에서 이루어지는 것임을 알 수 있다. 이러한 매핑 관계는 동일

한 서비스 도메인이나 프로토콜 네트웍의 내부에서는 발생하지 않으며, 서비스 도메인이나 네트웍의 경계 노드에서 발생한다.

L2 – L3 Mapping		L3 – L3 Mapping		L3 – L2 Mapping	
CoS → DSCP		IP Prec → DSCP		DSCP → CoS	
CoS	DSCP	IP Prec	DSCP	DSCP	CoS
0	0	0	0	0~7	0
1	8	1	8	8~15	1
2	16	2	16	16~23	2
3	24	3	24	24~31	3
4	32	4	32	32~39	4
5	40	5	40	40~47	5
6	48	6	48	48~55	6
7	56	7	56	56~63	7

그림 8. L2와 L3 사이에서의 매핑 관계.

7. Traffic Conditioning

7.1 트래픽 컨디셔너

트래픽 컨디셔닝(traffic conditioning)은 에지 라우터에서 입력되는 트래픽이 사전에 정의된(약속된) 사용자의 트래픽 프로파일을 따르는지의 여부를 확인하고, 순응 여부에 따라 기본적인 QoS 처리를 해 주거나 다른 QoS 처리를 해 주는 근거를 마련해 주는 프로세스를 말한다. 여기에서 말하는 기본적인 QoS 처리는 패킷의 우선순위를 변경하는 마킹(marking)이나 레잇 리미팅(rate limiting)을 통한 패킷 드랍(packet drop)과 같은 것이 될 수 있다. 다른 QoS 처리는 트래픽 컨디셔닝의 결과가 흐름 제어나 패킷 스케줄링 과정에서 영향을 받을 수 있음을 말한다. 이들에 대해서는 뒤에서 자세히 설명된다.

트래픽 컨디셔닝을 구현하는 기능 모듈을 트래픽 컨디셔너(traffic conditioner)라 하며, 이는 트래픽 컨디셔닝을 어떤 범위에서 정의하느냐에 따라 그 구성이 달라진다. 트래픽 컨디셔닝을 광범위한 측면에서 정의하면, 트래픽 컨디셔너는 패킷 클래시파이어(packet classifier), 미터(meter), 마커(marker), 그리고 드라퍼(dropper)나 쉐이퍼(shaper)로 구성된다. 그러나, 좁은 의미에서 트래픽 컨디셔너는 미터와 마커로만 구성되며, 트래픽을 측정하고 그 결과에 따라 순응 여부를 패킷에 표시하는 기능만을 포함하게 된다. 그림 9는 미터와 마커로만 구성된 트래픽 컨디셔너를 보여주고 있다. 이는 그림 2의 에지 노드의 노란색으로 표시된 부분의 일부에 해당한다.

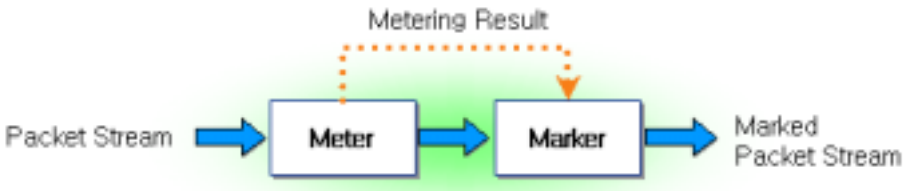


그림 9. 미터와 마커로 구성된 트래픽 컨디셔너.

7.2 미터 (Meter)와 마커(Marker)

미터와 마커는 그림 9에 보이는 것처럼 기능적인 측면에서는 구분되지만, 일반적으로 함께 존재한다. 이러한 이유 때문에, 좁은 의미에서 트래픽 컨디셔너를 구성하게 된다. 미터는 장비로 입력되어 패킷 클래시피케이션 된 트래픽 플로우를 측정한다. 일반적으로 트래픽 플로우의 입력 속도를 통해 대역폭을 측정하거나 버스트 정도를 측정하게 된다. 미터는 미리 약속된/정의된 트래픽 프로파일과 입력된 플로우의 트래픽 프로파일을 비교함으로써 순응 여부를 결정하게 되며, 그 결과에 따라 마커는 필요한 마킹 처리를 하게 된다. 일반적으로, 미리 약속된 트래픽 프로파일을 만족하는 경우, 일정한 범위 내에서 초과하는 경우, 일정한 범위를 넘어서는 경우의 세 가지로 구분을 하며, DiffServ에서는 각 경우에 대해 패킷을 Green, Yellow, 그리고 Red로 표시를 한다. Cisco 장비의 경우는 Committed, Excess, 그리고 Violated 라는 용어를 사용해서 각 경우를 나타내기도 한다.

DiffServ의 경우 single-rate three-color marker (sr-TCM)과 two-rate three-color marker (tr-TCM)이 대표적인 마커 혹은 트래픽 컨디셔너로 사용되고 있다. RFC 2697 및 2698에 각각 기술되어 있는 sr-TCM과 tr-TCM은 기본적으로 이중 토큰 버킷 구조를 사용하고 있다. sr-TCM이 토큰의 업데이트 속도로 CIR (committed information rate)만 사용하는 반면, tr-TCM은 토큰 업데이트 속도로 CIR과 PIR (peak information rate) 두 가지를 사용한다. 그림 10은 sr-TCM의 동작 방식을 보여준다. 입력된 패킷의 크기 B가 토큰 카운터 Tc보다 작으면 Green 패킷으로 마크되며, Tc보다 크지만 Te보다 적은 경우 Yellow 패킷으로 마크되고 Te보다도 큰 경우에는 Red 패킷으로 마크되는 과정을 보여주고 있다.

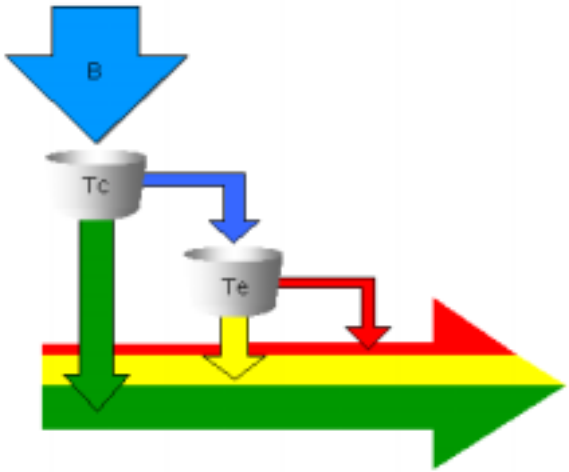


그림 10. sr-TCM의 동작.

8. Rate Limiting & Shaping

레잇 리미팅과 쉐이핑은 트래픽의 대역폭을 제어(bandwidth control)하는 두 가지의 대표적인 기술이다. 레잇 리미팅과 쉐이핑 모두 리키 버킷이나 토큰 버킷을 사용해서 구현된다.[4] 단지 차이가 있다면, 레잇 리미팅은 버퍼링을 사용하지 않고 쉐이핑은 버퍼링을 사용한다는 것이다. 이 절에서는 레잇 리미팅과 쉐이핑에 대해 자세히 살펴볼 것이다.

[4] 리키 버킷은 ATM 같이 고정길이 패킷을 사용하는 경우에 주로 사용되며, 토큰 버킷은 가변 길이 패킷에 대해 사용될 수 있다. 즉, 토큰 버킷은 버스트한 패킷 사이즈를 허용한다.

8.1 레잇 리미팅(Rate Limiting)

폴리싱(policing)이라고도 불리는 레잇 리미팅은 말 그대로 어떤 트래픽의 속도(대역폭)를 제한하는 기술이다. 속도를 제한하기 위해서는 기본적으로 입력되는 트래픽의 속도를 알 수 있어야 하므로 7.2절에서 설명된 미터(meter) 및 마커(marker)와 함께 사용된다. 레잇 리미팅의 동작은 그림 11에 보이는 것처럼 목표로 한 속도 이상으로 들어오는 트래픽은 모두 버리는 것이다. 즉, 과도한 트래픽을 버퍼링 하지 않고 그대로 드랍 시키는 것이다.

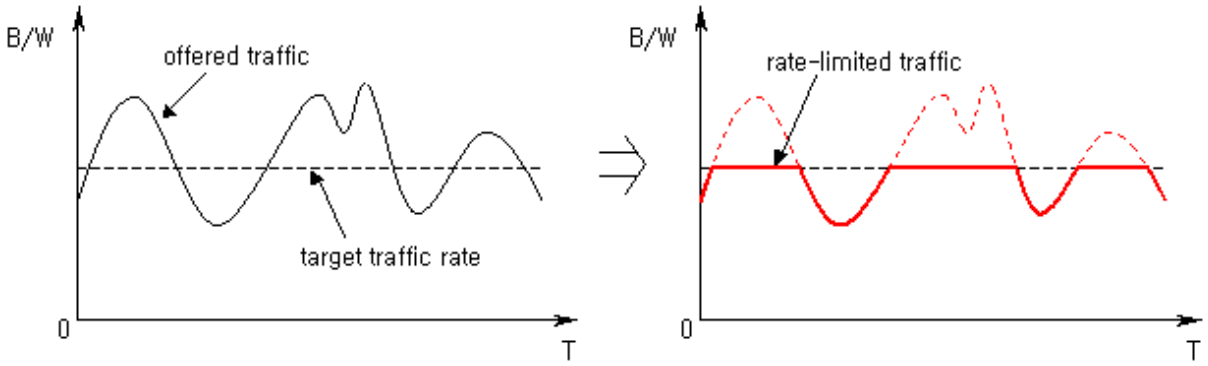


그림 11. 레잇 리미터의 동작.

일반적으로 레잇 리미터(rate limiter)는 장비의 입력 부분, 즉 스위칭 패브릭의 앞쪽에 위치한다고 알려져 있으나, 스위치 패브릭의 뒤쪽인 출력 모듈에 존재할 수도 있고 입출력 모듈 모두에 존재할 수 있다. 그러나, 8.2절에서 설명될 쉐이퍼가 주로 출력 모듈에 존재하기 때문에 레잇 리미터는 쉐이퍼와의 중복을 피해 버퍼를 사용하지 않는 입력 모듈에 위치하게 되는 것이다.

레잇 리미팅은 몇 가지 기준에 의해 구분할 수 있다. 먼저 레잇 리미팅이 적용되는 대상을 기준으로 플로우 기반 레잇 리미팅(flow-based rate limiting)과 포트 기반 레잇 리미팅(port-based rate limiting)으로 구분할 수 있다. 즉, 플로우 기반 레잇 리미팅은 각 어플리케이션 플로우에 대해 트래픽의 속도/대역폭을 제한하는 것이며, 포트 기반 레잇 리미팅은 포트나 인터페이스 단위로 트래픽의 속도/대역폭을 제한하는 것이다. 레잇 리미팅의 적용 방식에 따라 소프트 레잇 리미팅(soft rate limiting)과 하드 레잇 리미팅(hard rate limiting)으로 구분할 수도 있다. 하드 레잇 리미팅은 지금까지 설명한 레잇 리미팅 방법으로, 입력된 트래픽이 약속된 트래픽 프로파일을 따르지 않으면, 즉 약속된 대역폭 이상으로 들어오면 무조건 버리는 방식이다. 소프트 레잇 리미팅은 장비가 혼잡 상태에 있지 않은 경우 약속된 트래픽 프로파일을 일정 범위 내에서 초과해서 입력된 트래픽에 대해 낮은 우선순위로 마크다운(mark down) 한 후 다음 프로세스로 넘기는 것이다. 낮은 우선순위로 마크 된 패킷들은 출력 쪽에서 다른 트래픽들과 경쟁 하는 도중 버려질 수 있다. 소프트 레잇 리미팅은 서비스 제공업자가 서비스 차원에서 제공할 수 있다.

8.2 쉐이핑 (Shaping)

쉐이핑도 레잇 리미팅과 마찬가지로 어떤 트래픽의 속도/대역폭을 제한하는 기술이다. 레잇 리미팅과의 차이가 있다면, 버퍼링을 사용한다는 것이다. 버퍼링을 사용함으로써 목표 속도 이상으로 들어오는 트래픽을 잠시 저장했다가 나중에 서비스를 해 주게 된다. 이러한 쉐이핑의 동작 특성이 그림 12에 잘 설명되고 있다. 이러한 쉐이핑의 특성은 어느 정도 버스트 트래픽을 수용할 수 있다는 점과, 레잇 리미팅에 비해 패킷 손실률을 줄이고 전체적인 쓰루풋을 향상시킬 수 있다는 장점을 가지고

있다. 그러나, 버퍼링으로 인해 추가적인 딜레이가 더해질 수 있으므로 커다란 버퍼를 사용하는 경우 실시간 어플리케이션 트래픽에 영향을 미칠 수 있다.

쉐이핑은 버퍼링을 사용하기 때문에 버퍼가 존재하는 곳에서 구현된다. 입력 및 출력 쪽에서 모두 구현될 수 있지만, 주로 버퍼가 위치하는 출력 쪽에서 구현되는 것이 일반적이다. 이는 버퍼가 없는 입력 쪽에서 쉐이핑과 같은 기능을 수행하는 레잇 리미팅이 구현되는 것과의 중복을 피한다는 점에서도 이해될 수 있다.

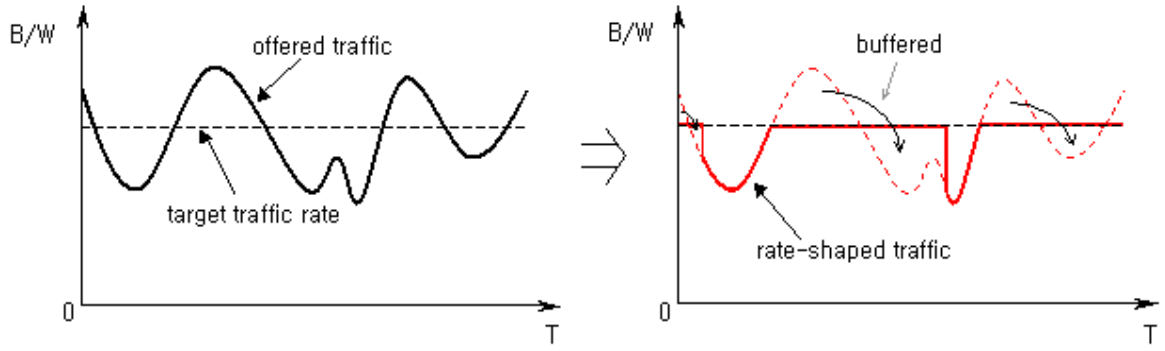


그림 12. 쉐이퍼의 동작.

9. 혼잡 제어 (Congestion Control) 및 흐름 제어 (Flow Control)

혼잡 제어(congestion control)는 네트웍 장비가 처리할 수 있는 능력 이상으로 입력된 트래픽 혹은 특정한 포트에 집중된 트래픽으로 인해 발생하는 혼잡 현상, 즉 장비가 계속해서 과부하 상태에 있는 것을 해결하는 기술을 말한다. 혼잡 현상을 해결하기 위한 방법에는 액세스 제어와 스케줄링 방식, 그리고 흐름 제어 방법이 사용될 수 있다. 액세스 제어는 장비로 입력되는 트래픽의 양을 제한하는 방법이며, 스케줄링은 장비에 입력된 트래픽을 효율적으로 서비스하는 방법이다. 흐름 제어(flow control)는 데이터 소스가 자신의 전송 속도를 네트웍과 수신기에서 현재 가용한 속도에 맞출 수 있도록 하는 일련의 기술들을 가리킨다.

액세스 제어는 CAC와 같이 혼잡이 발생했을 때 새로운 커넥션의 설정을 허용하지 않거나 우선순위가 낮은 커넥션의 트래픽의 양을 제한하는 방법으로 구현될 수 있다. 그러나, IP 기반의 네트웍에서는 액세스 제어를 통한 혼잡 제어가 일반적이지 않다. 이는 액세스 제어와 베스트 에포트의 개념이 서로 상충되기 때문이다. 스케줄링 방식은 다양한 스케줄링 기법을 통해 높은 우선순위를 갖는 트래픽에 대한 서비스를 보장해 주는 방식으로 10장에서 대표적인 몇 가지 방식들을 살펴볼 것이다. 흐름 제어를 통한 혼잡 제어 방식은 IP 기반 네트웍에서 가장 오래 전부터 사용되어 오던 방식으로 9.1절에서 설명될 것이다.

9.1 흐름 제어 (Flow Control)

흐름 제어는 데이터 소스가 혹은 데이터 소스로 하여금 전송 속도를 줄이도록 함으로써 혼잡 문제를 해결하는 방식이다. 흐름 제어가 적용되는 방식에는, 혼잡이 발생할 것을 예측하고 혼잡이 발생하기 이전에 흐름 제어가 수행되도록 하는 방법과, 혼잡이 발생한 것을 확인한 후 흐름 제어가 수행되도록 하는 방식으로 구분할 수도 있다. 이 경우, 전자를 Proactive 방식이라 하며, 후자를 Responsive 혹은 Reactive 방식이라고 한다. 어느 방식이든, 기본적으로 데이터 소스가 전송 속도를 조절할 수 있는 능력을 가지고 있어야 한다. 인터넷에서 전달되는 트래픽의 상당 부분은 TCP 트

래픽이며, TCP 서버는 이처럼 전송 속도를 조절할 수 있는 능력을 가지고 있어서 흐름 제어를 할 수 있다. 따라서, 이곳에서는 TCP의 동작을 먼저 살펴 본 후, TCP의 동작 특성을 이용한 흐름 제어 기법인 RED와 WRED에 대해서 살펴볼 것이다.

TCP의 동작은 슬로우 스타트(slow start)와 혼잡 회피(congestion avoidance)의 두 부분으로 동작한다. 슬로우 스타트는 TCP 센더(sender)의 윈도우 사이즈(CWND)가 쓰레시홀드(threshold)보다 적은 경우, 전송한 패킷에 대해 타임아웃(time out)이 발생하기 전에 자신이 보냈던 패킷에 대한 ACK를 받을 때마다 TCP 센더가 자신의 전송 속도를 두 배로 증가시키는 것을 말하며, 혼잡 회피는 TCP 센더의 윈도우 사이즈가 쓰레시홀드보다 큰 경우 1 MSS (maximum segment size) 만큼씩 전송 속도를 증가시키는 것을 말한다. 만약, 타임아웃이 발생하게 되면, 쓰레시홀드는 현재 윈도우 사이즈의 1/2로 줄어들고 윈도우 사이즈는 1 MSS로 리셋 된다. 즉,

- TCP 센더가 Timeout 이전에 ACK를 받을 때
  - if CWND ≤ Threshold → CWND = 2 * CWND
  - if CWND > Threshold → CWND = CWND + 1
- TCP 센더가 Timeout 이전에 ACK를 받지 못할 때
  - Threshold = CWND / 2
  - CWND = 1 MSS

그림 13은 이러한 TCP의 동작을 보여주고 있다. 1로 나타낸 구간은 슬로우 스타트에 해당하며, 2로 나타낸 구간은 혼잡 회피 구간이다.

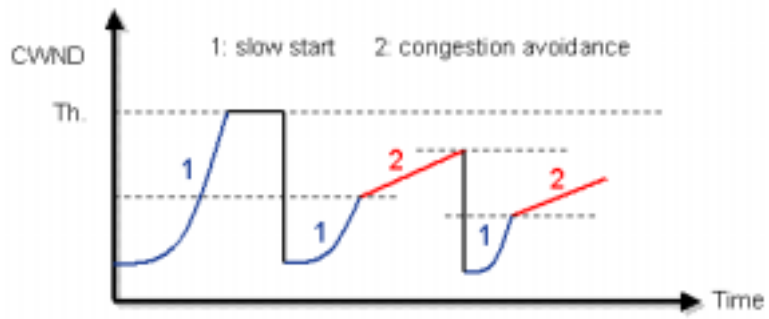


그림 13. TCP의 동작.

그렇다면, 어떻게 TCP 센더의 속도 조절 (흐름 제어) 기능을 이용해서 혼잡 제어를 수행할까? TCP 기반의 흐름 제어 시나리오를 살펴보기 위해서는 여러 개의 TCP 플로우가 하나의 버퍼를 사용하는 경우를 생각해야 한다. 그림 14는 이러한 시나리오를 보여주고 있다. 이런 경우 큐 사이즈는 빠른 속도로 증가하게 되며, 이내 버퍼 풀(buffer full)이 발생하고 오버플로우(overflow)가 발생하게 된다. 그러면, 모든 TCP 플로우들은 버퍼 풀 이후에 도착한 모든 패킷들을 잃게 되며, TCP 센더들은 timeout이 발생할 때까지 잃어버린 패킷들에 대한 ACK를 받지 못하게 된다. 결국, 모든 TCP 센더들은 자신의 전송 속도를 줄이게 되며 빠른 속도로 큐 사이즈는 줄어들게 된다. 그러나, 슬로우 스타트로 인해 큐 사이즈는 다시 증가하게 되고 오버플로우가 발생하며, 모든 TCP 플로우에 대해 동일한 과정이 반복되게 된다. 이러한 현상을 Global Synchronization이라 하는데, 트래픽 양의 급격한 출렁거림으로 인해 성능은 물론 네트웍 장비가 불안정해지는 원인이 된다. 이러한 Global Synchronization 문제는 공정한 방법으로 임의의 TCP 플로우를 선택해서 전송 속도를 줄이도록 함으로써 해결할 수 있다. 그런 방법 중의 하나이며 가장 대표적인 방법이 바로 RED (random early

detection/discard)에 의한 혼잡 제어 방법이다.

그림 14. 여러 TCP 플로우에 의해 혼잡이 발생하는 경우 – 임의의 TCP 플로우에 속한 패킷을 미리 버림으로써 혼잡이 발생하는 것을 사전에 막을 수 있다.

9.2 RED에 의한 혼잡 제어

Random Early Detection 혹은 Random Early Discard라고도 불리는 RED는 TCP 동작 특성을 이용한 대표적인 혼잡 제어 기법이다. 이름이 암시하는 것처럼, 혼잡이 발생하기 이전에 미리 랜덤한 방식으로 패킷을 버림으로써 특정한 TCP 플로우로 하여금 전송 속도를 줄이게 하는 방법이다. 기본적인 동작은 큐에 두 개의 쓰레시홀드 TH_min과 TH_max를 두고 세 구간에 서로 다른 드랍 확률을 적용하는 것이다. 이러한 과정이 아래 그림 15에 잘 설명되어 있다. 즉, 평균 큐 사이즈(AQS: Average Queue Size)가 TH_min보다 작은 경우에는 어떤 패킷도 버리지 않고 모두 받아 들인다. 큐 사이즈가 TH_min보다는 크지만 TH_max보다는 작은 경우 큐 사이즈에 따라 특정한 확률값을 가지고 패킷을 버린다. 큐 사이즈가 TH_max보다 큰 경우는 입력되는 모든 패킷을 버린다. 즉, 혼잡의 정도가 심해질수록 많은 패킷을 버림으로써 입력되는 트래픽의 양을 줄이려는 방법이다.

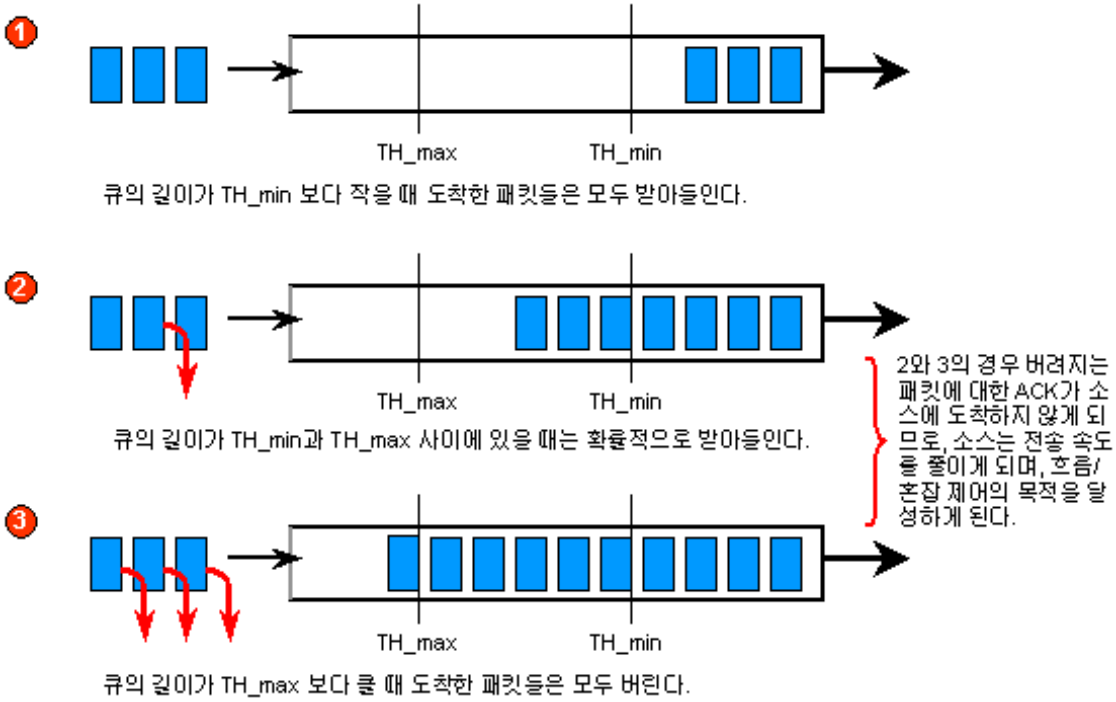


그림 15. RED의 동작 원리.

TH_min과 TH_max에 의해 구분되는 각 구간에서 패킷을 버리는 확률은 Drop Probability Function이라는 확률 함수 혹은 확률 테이블로 작성이 되며, 가능한 큐 사이즈에 대해 드랍 확률 값들을 정리해 놓고 있다. 그림 16은 RED 패킷 드랍 확률 함수의 일례를 보여주고 있다. K 는 최대 큐 사이즈이며 $p_{\max}$ 는 TH_min과 TH_max 사이의 기울기를 결정하는 값이 된다. 일반적으로, TH_max 값이 너무 작으면 패킷 드랍이 빈번히 발생해서 전체 성능에 심각한 영향을 줄 수도 있으

며, TH_max 이상에서 모든 입력되는 패킷을 버리는 동작이 버퍼 오버플로우에 의한 결과와 동일하기 때문에 TH_max를 큐의 최대 크기인 K 와 같거나 가까운 값으로 설정을 하게 된다. 또한, $p_{\max}$ 의 값이 너무 작으면 패킷이 드랍 되는 빈도가 낮아져 혼잡 제어 효과가 제대로 나타나지 않을 수도 있다. 따라서, RED를 사용할 때는 TH_min, TH_max, 그리고 $p_{\max}$ 값을 적절하게 설정해 주는 것이 중요하다.

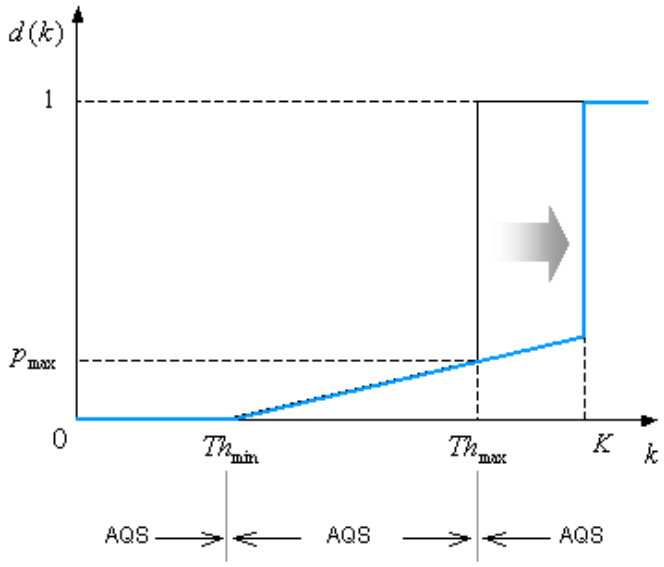


그림 16. RED 패킷 드랍 확률 함수.

9.3 WRED에 의한 혼잡 제어

Weighted RED는 하나 혹은 여러 개의 서로 다른 클래스 트래픽에 서로 다른 특성(profile or weight)를 갖는 RED 함수를 적용함으로써 혼잡 제어를 하는 것을 말한다. 서로 다른 특성을 갖는 RED 함수라는 것은 TH_min, TH_max, 그리고 $p_{\max}$ 값이 서로 다른 RED 패킷 드랍 확률 함수를 말한다. 동일한 클래스 트래픽에 WRED를 적용하는 경우는, 같은 클래스에 속한 서로 다른 우선순위의 패킷들에 서로 다른 RED 함수를 적용하게 된다. 예를 들면, DiffServ의 AFx 클래스의 경우 동일한 클래스에 3 개의 서로 다른 drop precedence가 존재하는데, 각각의 drop precedence에 대해 서로 다른 RED 함수를 적용할 수 있다.

그림 17은 이러한 WRED 패킷 드랍 확률 함수의 일례를 보여주고 있다. 이 그림은 DiffServ의 어떤 클래스 트래픽에 대해 green, yellow, 그리고 red로 마크된 패킷들에 대해 서로 다른 RED 함수를 적용할 수 있음을 보여주고 있다. 그림에서 보이는 것처럼 낮은 우선순위의 패킷 혹은 클래스에 대해서는 더욱 공격적인 패킷 드랍 확률 함수를 적용하게 되며, 우선순위가 높은 패킷 혹은 클래스에 대해서는 보수적인 패킷 드랍 확률 함수를 적용하게 된다. 그림에서는 우선순위가 낮은 C1에 대한 TH_max가 C2에 대한 TH_min보다 크게 설정된 경우를 보여주고 있으나, C1에 대한 TH_max가 C2에 대한 TH_min보다 반드시 작거나 같아야 하는 조건을 줄 수도 있다.

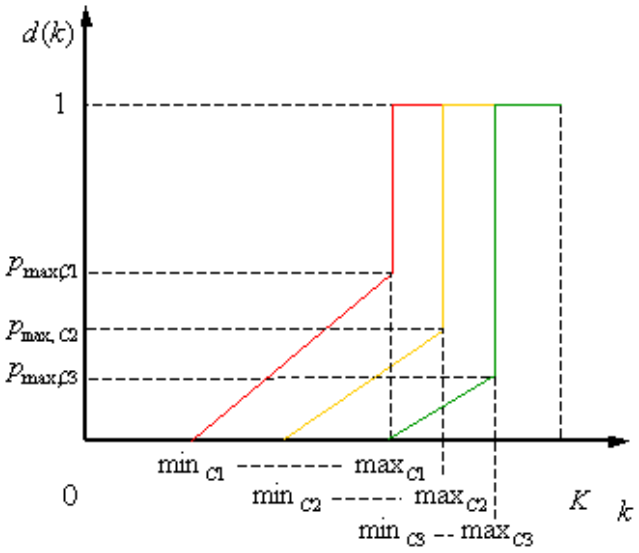


그림 17. 세 개의 클래스에 적용되는 WRED 패킷 드랍 확률 함수.

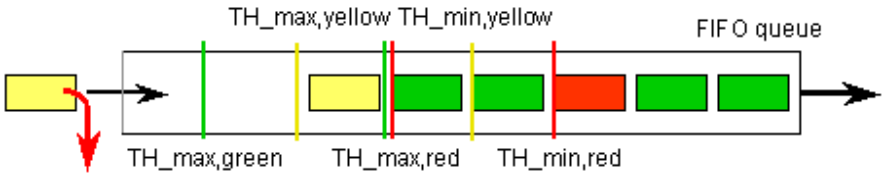


그림 18. 단일 클래스 트래픽에 대한 WRED의 적용.

그림 18은 단일 큐를 사용하는 하나의 클래스에 WRED를 적용하는 예를 보여주고 있다. 비록 같은 클래스에 속한 패킷들이지만, 드랍 우선순위가 다르며, 서로 다른 RED 드랍 확률 함수가 적용된다. 그림에서는 패킷이 TH_max,yellow 바로 아래까지 차 있으며, 새로 들어오는 yellow 패킷은 TH_max,yellow 쓰레시홀드를 넘어서게 되기 때문에 무조건 버려지게 된다. 만약, yellow 패킷 대신 green 패킷이 도착한다면, 이 패킷은 TH_min,green과 TH_max,green 사이에 존재하기 때문에 특정한 확률을 가지고 버려지게 될 것이다.

10. Queueing & Scheduling

큐잉(queueing)은 어떤 네트웍 장비가 처리(서비스)할 수 있는 것 이상으로 패킷이 도착하거나, 동시에 동일한 목적지로 향하는 패킷들이 존재할 때 발생하게 된다. 즉, 한꺼번에 처리할 수 없는 패킷들을 잠시 동안 버퍼에 저장을 해 두었다가 나중에 서비스를 하는 것을 큐잉이라고 한다. 스케줄링은 이렇게 버퍼에 저장된 패킷들을 서비스 하는 방식을 총칭하는 용어이다. 스케줄링 알고리즘은 간단하면서도 효율적이어야 하며 공정해야 한다. 큐잉과 스케줄링은 따로 떼어서 생각할 수도 있으나, 일반적으로 큐잉 방식에 의해 스케줄링 방식이 결정되게 된다. 이곳에서는 특정한 출력 포트에서 단일 FIFO 큐를 사용하거나 여러 개의 FIFO 큐를 사용하는 경우로 나누어서 생각할 것이다. 단일 혹은 여러 개의 FIFO 큐를 가정할지라도, 실제로 도착한 패킷들은 도착 순서에 상관없이 공유 메모리(Shared Memory)에 저장되며, 패킷이 저장되어 있는 메모리 정보만이 FIFO 방식의 가상 큐(virtual queue)로 저장되어 관리된다. 이곳에서 다룰 큐잉 및 스케줄링 방식은, FIFO 큐잉, Priority 큐잉, Fair Queueing, 그리고 Weighted Fair Queueing으로 제한할 것이다.

10.1 FIFO Queueing

FIFO 큐잉은 단일 FIFO 큐를 사용하는 것을 말한다. 일반적으로 하나의 큐에 하나 이상의 클래스가 매핑 되는 것과는 달리, FIFO 큐잉은 하나의 큐에 모든 클래스의 트래픽을 저장하게 된다. FIFO 큐잉에 사용되는 스케줄링 방식은 First Come First Serve. 즉, 패킷의 클래스나 우선순위에 상관없이 먼저 입력된 패킷을 먼저 서비스하게 된다. 베스트 에포트 서비스 모델만을 가지고 있는 전통적인 인터넷 망에 사용되는 큐잉 및 스케줄링 구조에 해당한다. 트래픽의 구분이 존재하지 않기 때문에, FIFO 큐잉을 사용하는 장비에서는 패킷 클래시피케이션이나 마킹처럼 클래스와 관련된 기능은 필요 없게 된다.

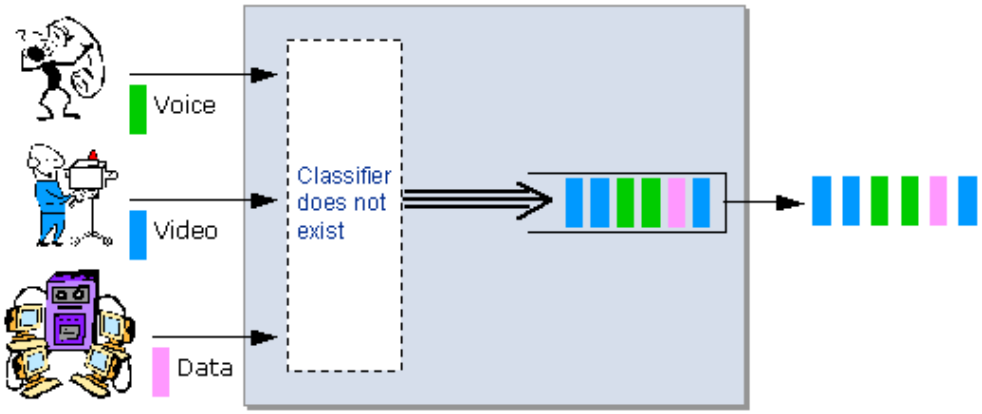


그림 19. FIFO 큐잉의 동작 – 클래스의 구분 없이 먼저 도착한 패킷을 먼저 서비스한다.

FIFO 큐잉의 장점은 구현이 간단하며 FIFO 큐의 동작이 예측 가능하다는 것이다. 즉, 패킷들의 순서가 유지되며, 패킷의 최대 딜레이는 큐의 최대 크기에 의해 결정된다. FIFO 큐잉의 단점으로는 클래스의 구분이 없기 때문에 차등화된 서비스를 제공하는 것이 불가능하다. 또한, 버스트 트래픽 서비스에 부적합 하며, 혼잡이 발생하는 경우 TCP보다 UDP 트래픽에 유리하다는 것이다. 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 되어 결국에는 TCP 트래픽의 서비스가 어렵게 된다.

10.2 Priority Queueing

Strict Priority 큐잉이라고도 불리는 Priority 큐잉은 여러 개의 FIFO 큐를 사용하는 방식이다. 여러 개의 FIFO 큐를 사용하므로, 각각의 큐가 서로 다른 트래픽 클래스에 매핑이 된다. Priority 큐잉을 사용하는 경우 스케줄링 방식은 아주 단순하다. 즉, 낮은 우선순위 큐에 저장되어 있는 패킷들은 높은 우선순위 큐에 저장되어 있는 패킷들이 모두 서비스 된 이후에나 서비스가 된다. 만약, 낮은 우선순위 큐에 저장되어 있는 패킷이 서비스 되는 도중 높은 우선순위 큐에 패킷이 입력되면, 낮은 우선순위 큐는 서비스를 잠시 멈추고 높은 우선순위 큐에 새로 도착한 패킷을 먼저 서비스 해 주게 된다.

Priority 큐잉 방식의 장점은 간단한 방법으로 차등화된 서비스를 제공할 수 있다는 것이다. 따라서, 실시간 어플리케이션을 지원할 수 있게 된다. 그러나, 높은 우선순위 큐에 패킷이 계속해서 입력되는 경우에는 낮은 우선순위 큐에 저장된 패킷이 서비스가 되지 못하는 스타베이션(starvation, 기근) 현상이 발생하는 문제점을 가지고 있다. 이러한 문제점은 뒤에서 설명될 WFQ을 사용해서 해

결 할 수 있다. 또한, 여러 트래픽이 하나의 동일한 큐에 저장되는 경우, 버스트 하게 입력되는 트래픽이 다른 플로우들을 압도하게 되는 문제가 발생한다.

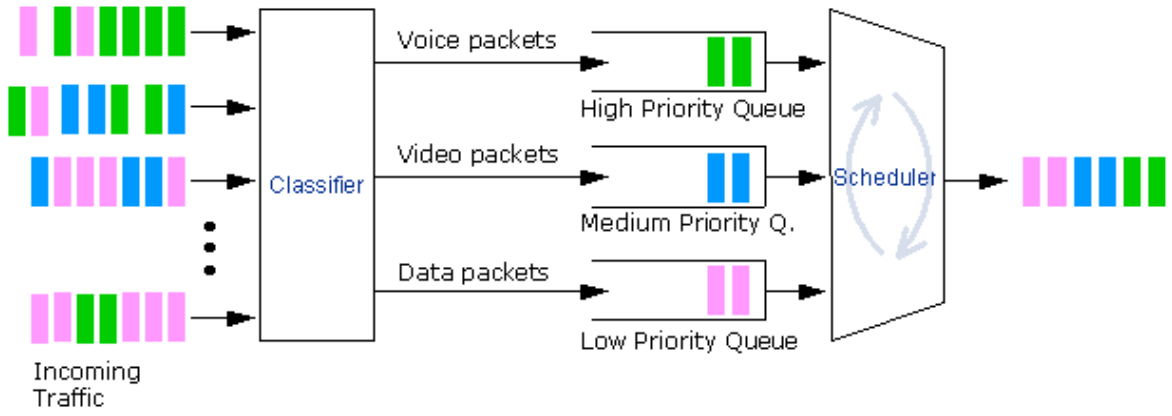


그림 20. Priority 큐잉 – 높은 우선순위의 큐부터 서비스 한다.

10.3 Fair Queueing

Fair 큐잉은 Priority 큐잉처럼 여러 개의 FIFO 큐를 사용하는 방식으로 각각의 큐는 하나 혹은 그 이상의 트래픽 클래스와 매핑이 된다. Priority 큐잉에서는 높은 우선순위 큐가 패킷을 가지고 있는 한 낮은 우선순위 큐에 대해 절대적인 서비스 우선순위를 갖기 때문에 낮은 우선순위 큐가 스타베이션 현상을 경험하게 되지만, Fair 큐잉에서는 모든 큐가 동일한 우선순위를 갖기 때문에 스타베이션 현상은 발생하지 않게 된다. 그러나, 이 방식은 트래픽의 특성을 고려치 않고 서비스 차원에서의 공정성만을 감안하고 있기 때문에, 스케줄링 차원에서 차등화된 서비스를 제공하는 것은 불가능하게 된다.

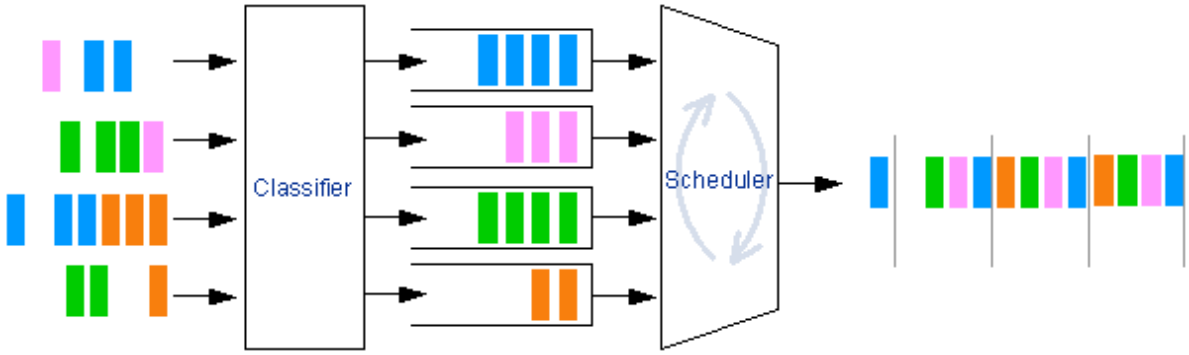


그림 21. Fair 큐잉 – 낮은 우선순위 큐의 스타베이션 현상을
막을 수는 있지만, 차등화된 서비스를 제공하는 것은 불가능하다.

10.4 Weighted Fair Queueing

WFQ은 Priority 큐잉 방식에서의 스타베이션 현상을 해결함과 동시에 Fair 큐잉 방식에서 차등화된 서비스를 제공하지 못하는 현상을 해소하기 위해 개발된 것이다. 이를 위해 각 큐는 웨이트(weight)를 할당 받게 되며, 할당 받은 웨이트에 비례하도록 스케줄링이 이루어진다.

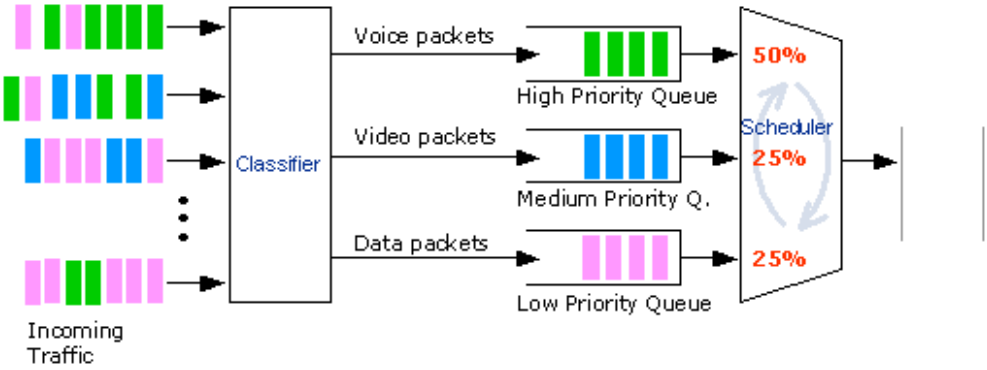
문서의 마지막에 있는 그림 22는 세 종류의 큐에 WFQ을 적용하는 예를 보여주고 있다. High Priority 큐에 50%, Medium 및 Low Priority 큐에 각각 25%의 웨이트를 할당했다고 가정하자. (b)

와 (c)에서는 모든 큐에 패킷이 존재하므로 웨이트 값에 비례하여 50:25:25의 비율로 서비스가 되는 것을 알 수 있다. 그러나, (d)에서는 High 큐에 패킷이 존재하지 않으므로 High 큐에 할당된 웨이트가 나머지 큐에 균등하게 혹은 각 큐에 할당된 웨이트 값에 비례하도록 할당되어 0:50:50의 비율로 서비스가 되는 것을 보여주고 있다.

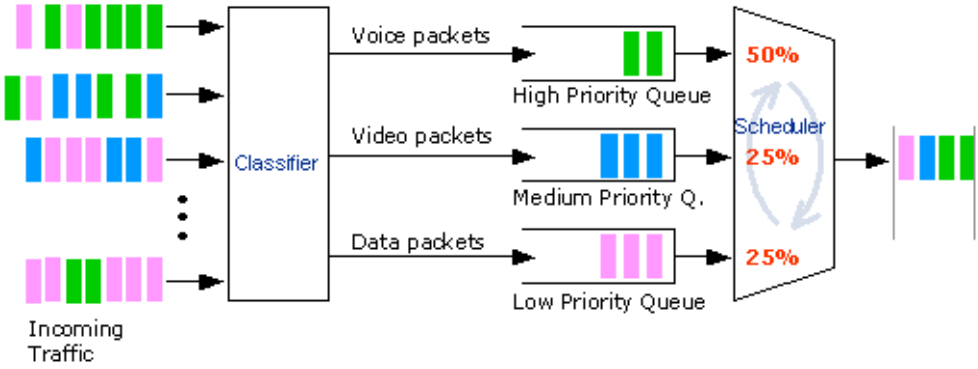
WFQ은 Priority 큐잉에서 발생하는 스타베이션 현상을 해결하려는 목적을 가지고 있기 때문에, 그림 22에 보이는 예에서처럼 단순히 웨이트 값만을 할당하기 보다는 반드시 보장되어야 하는 대역폭을 설정해 줄 수도 있다. 즉, High 큐에 대해서는 전체 대역폭의 25%를, Medium 큐에 대해서는 전체 대역폭의 15%를 반드시 보장해 줄 필요가 있는 경우를 생각해 보자. 이런 경우는 상기의 대역폭을 각 큐에 우선적으로 할당해 주고, 나머지 60%에 해당하는 대역폭을 웨이트 값에 따라 각 큐에 할당해주게 된다. 웨이트가 각각 50:25:25의 비율로 할당이 되었다면, High 큐는 나머지 대역폭의 50%에 해당하는 30만큼을, Medium 큐와 Low 큐는 나머지 대역폭의 25%에 해당하는 15만큼을 할당받게 된다. 결국, High 큐는 반드시 보장되는 25%와 웨이트에 따라 할당받은 30% 만큼, 즉 전체 대역폭의 55%만큼 서비스를 받게 된다. Medium 큐는 15%를 보장받았고 웨이트에 의해 15%를 할당 받았으므로 전체 대역폭의 30%를 서비스 받게 된다. Low 큐는 웨이트에 의해 할당받은 15%만을 서비스 받게 된다. 즉, 네트웍 자원을 최대로 활용하는 방향으로 WFQ의 웨이트 값들이 적용된다.

11. 맺음말

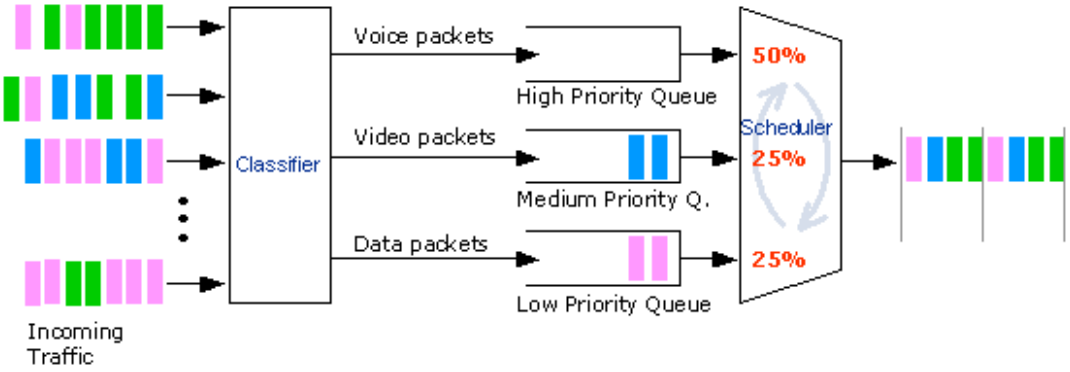
인터넷을 통해 전달되는 트래픽의 특성이 다양화 될수록 베스트 에포트의 단일 서비스 모델보다는 멀티 서비스 모델을 통한 차별화된 서비스의 제공이 중요해진다. 이 글에서는 기존의 인터넷 망에서 차등화된 서비스를 제공하는 IP QoS의 개념과 차등화된 서비스를 제공하는데 필요한 QoS 요소 기술들에 대해 살펴 보았다. 구체적인 기술 소개보다는 각 기술에 대한 개념을 이해하는데 초점을 맞추었다. 따라서, 관련 내용을 알고 있는 사람들에게는 너무 쉬운 이야기가 될 수도 있다. 그러나, IP QoS의 개념이 무엇인지, 그리고 어떤 기술들에 의해 어떤 구조로 적용되는지에 대해 개괄적으로 알고자 하는 사람들에게는 커다란 도움이 될 거라고 생각한다. 다음부터는 이 문서의 각 부분에 나와 있는 내용들, 특히 QoS 구현 기술들을 중심으로 좀 더 상세한 내용을 소개하려 한다. 이 글 및 앞으로 준비하려 하는 글에 대한 코멘트나 제안, 오류 사항 지적은 hykim@ieee.org로 해 주시길 바란다.



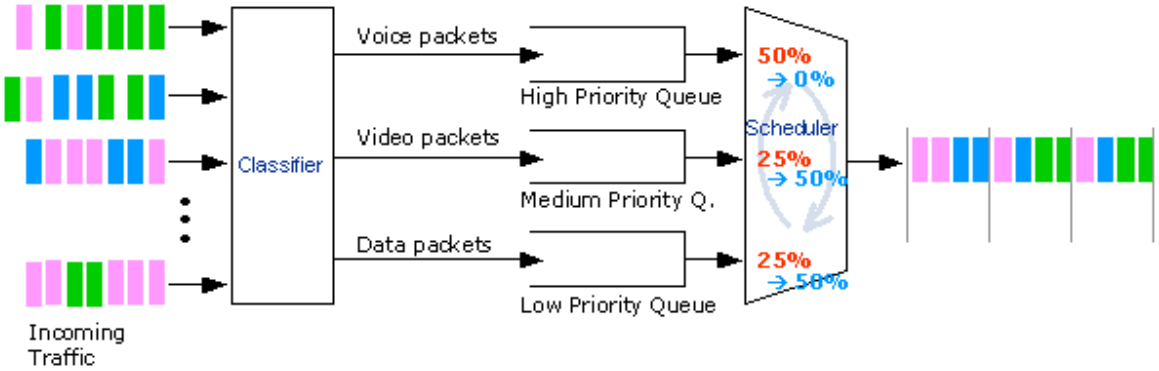
(a) 서비스 이전의 큐 상태.



(b) High:Medium:Low = 50:25:25의 비율로 서비스 된다.



(c) High:Medium:Low = 50:25:25의 비율로 서비스 된다.



(d) Medium 및 Low 큐에서 50%씩 서비스된다.

그림 22. WFQ의 동작 – 각 큐에 할당된 웨이트에 비례하도록 서비스 된다.