TCP/IP 흐름제어 & 혼잡제어

TCP/IP 정의

Internet Protocol Suite : 인터넷에서 컴퓨터들이 서로 정보를 주고받는 데 쓰이는 통신 규약(프로토콜)의 모음이다.
TCP/IP : Internet Protocol Suite 중 TCP/IP가 가장 많이 쓰인다.

TCP -> Transmission Control Protocol -> 전송 제어 규약
IP -> Internet Protocol -> *패킷 통신 방식의 인터넷 프로토콜

즉, IP 방식으로 통신을 하고, TCP가 전송에 대한 제어 규약을 가짐!

*패킷 :
정의 : 패킷은 정보 기술에서 패킷 방식의 컴퓨터 네트워크가 전달하는 데이터의 형식화된 블록이다. 패킷은 제어 정보와 사용자 데이터로 이루어진다.
장점 : 작은 블록의 패킷으로 데이터를 전송하며 데이터를 전송하는 동안만 네트워크 자원을 사용하도록 한다.

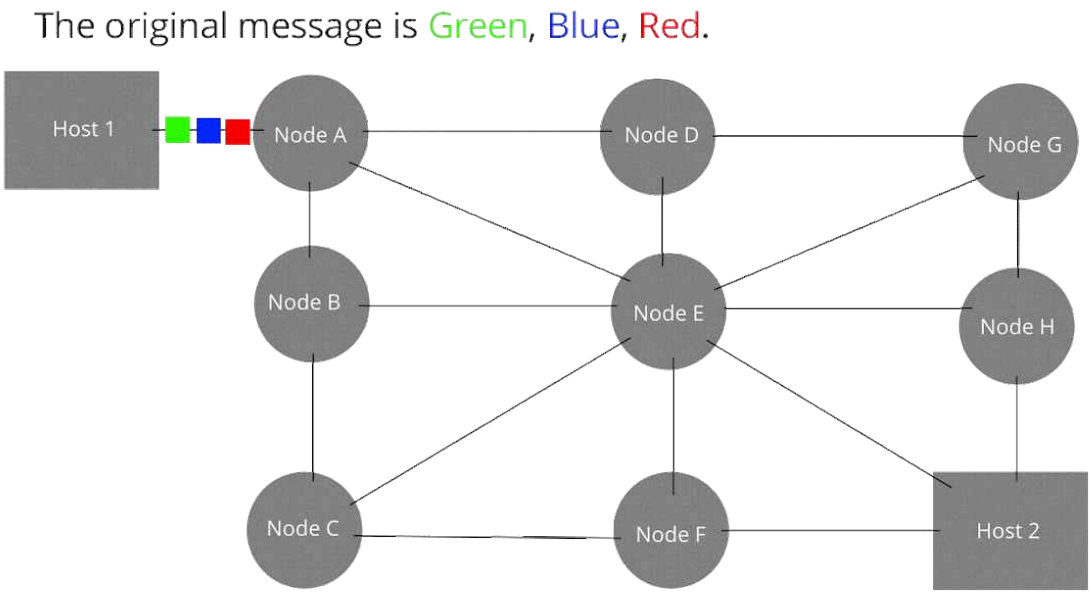


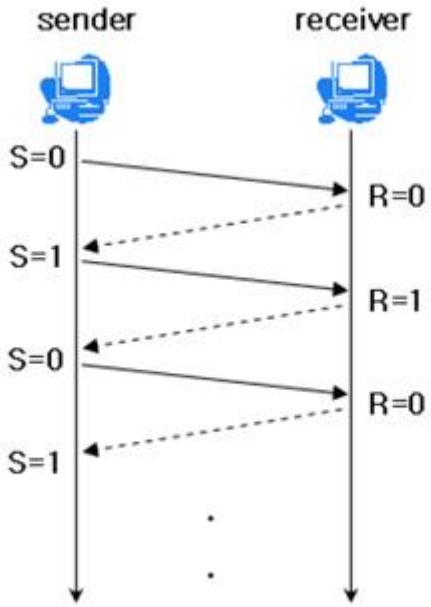
그림: :
https://ko.wikipedia.org/wiki/%ED%8C%A8%ED%82%B7_%EA%B5%90%ED%99%98

결국, TCP/IP는 송신자가 수신자에게 IP 주소를 사용하여 데이터를 전달하고
그 데이터가 제대로 갔는지, 너무 빠르지는 않는지, 제대로 받았다고 연락은
오는지에 대한 개념이다.

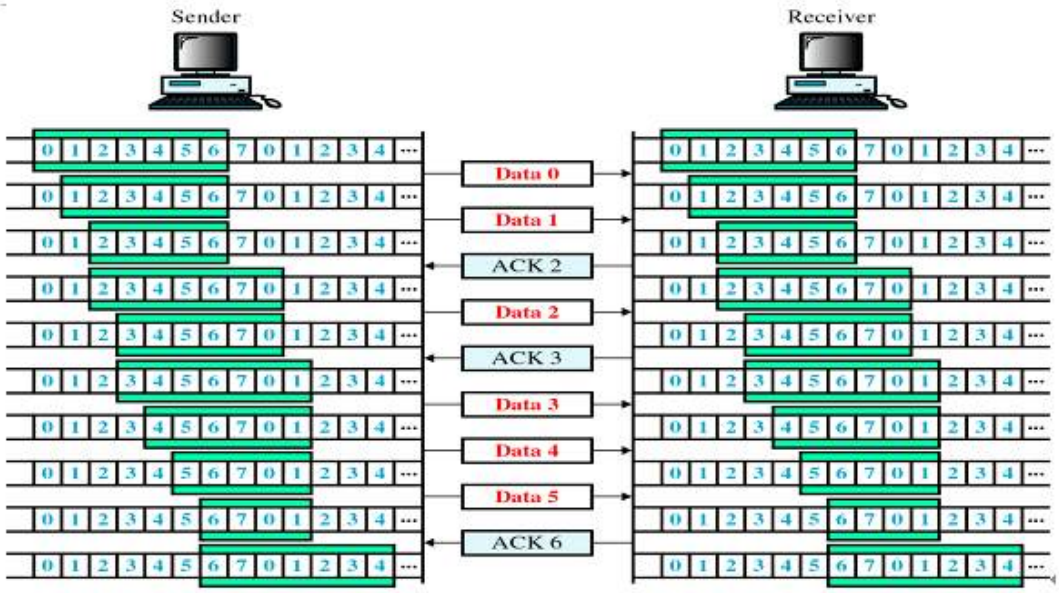
흐름제어

개념 : 수신 측이 송신 측보다 데이터 처리 속도가 빠르면 문제가 없지만, 송신 측의 속도가 빠를 경우 문제가 생긴다. 수신 측에서 제한된 저장 용량을 초과한 이후에 도착하는 패킷은 손실될 수 있으며, 만약 손실된다면 불필요한 추가 패킷 전송이 발생하게 된다.
흐름 제어는 위와 같이 송신 측과 수신 측의 TCP 버퍼 크기 차이로 인해 생기는 데이터 처리 속도 차이를 해결하기 위한 기법

1. Stop and Wait -> 매번 전송한 패킷에 대해 확인 응답(ACK)를 받으면 다음 패킷을 전송하는 방법
단점 : 패킷을 하나씩 보내기 때문에 비효율적인 방법이다.



2. Sliding Window -> 수신 측에서 설정한 **윈도우 크기 만큼** 송신 측에서 확인 응답(ACK)없이 패킷을 전송할 수 있게 하여 데이터 흐름을 동적으로 조절하는 제어 기법이다.



윈도우 크기

최초의 윈도우 크기는 호스트들의 '3 way handshaking'을 통해 수신 측 윈도우 크기로 설정되며,
이후 수신 측의 버퍼에 남아있는 공간에 따라 변한다. 윈도우 크기는 수신 측에서 송신 측으로 확인 응답(ACK)을 보낼 때
TCP 헤더(window size)에 담아서 보낸다. 즉, 윈도우는 메모리 버퍼의 일정 영역이라고 생각하면 된다.

동작 방식

윈도우에 포함된 패킷을 계속 전송하고, 수신 측으로부터 확인 응답(ACK)이 오면 윈도우를 옆으로 옮겨 다음 패킷들을 전송한다.

1. 최초로 수신자는 윈도우 사이즈를 7로 정한다.
2. 송신자는 수신자의 확인 응답(ACK)을 받기 전까지 데이터를 보낸다.
3. 수신자는 확인 응답(ACK)을 송신자에게 보내면, 슬라이딩 윈도우 사이즈을 충족할 수 있게끔 윈도우를 옆으로 옮긴다
4. 이후 데이터를 다 받을 때까지 위 과정을 반복한다.

재전송

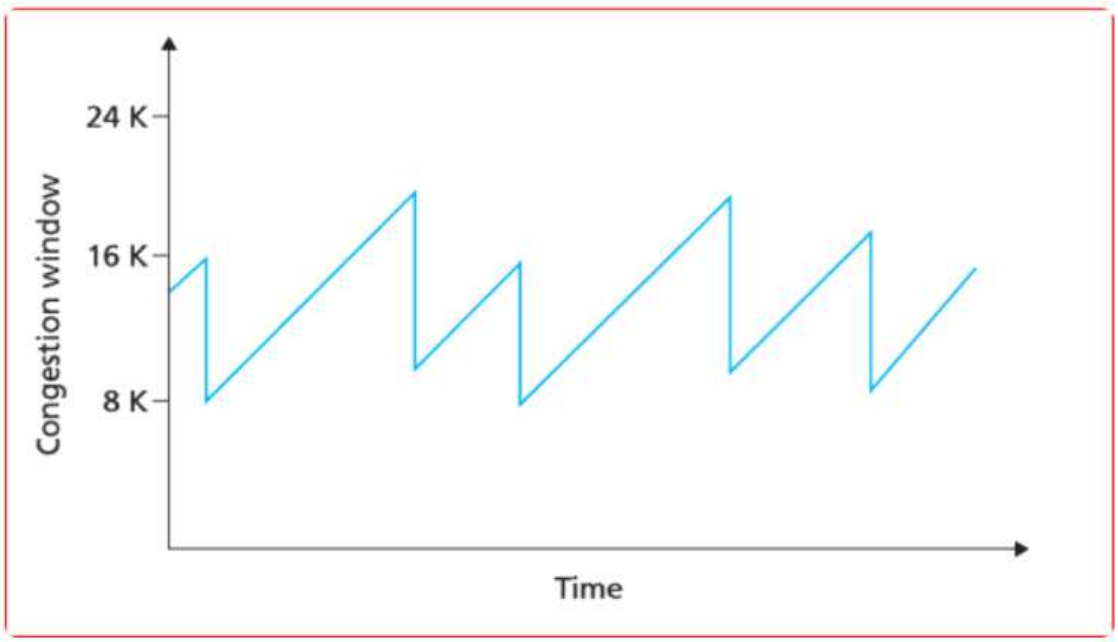
송신 측은 일정 시간 동안 수신 측으로부터 확인 응답(ACK)을 받지 못하면, 패킷을 재전송한다. 만약, 송신 측에서 재전송을 했는데 패킷이 **소실된 경우가 아니라** 수신 측의 버퍼에 남는 공간 없는 경우면 문제가 생긴다. 이를 해결하기 위해 송신 측은 해결 응답(ACK)을 보내면서 남은 버퍼의 크기 (윈도우 크기)도 함께 보내 준다.

혼잡제어

개념 : 송신측의 데이터는 지역망이나 인터넷으로 연결된 대형 네트워크를 통해 전달된다. 하지만 이러한 네트워크 상의 라우터가 항상 한가로운 상황은 아니다. 만약, 한 라우터에 데이터가 몰릴 경우, 다시 말해 혼잡할 경우 라우터는 자신에게 온 데이터를 모두 처리할 수 없다. 그렇게 되면 호스트들은 또 다시 재전송을 하게 되고 결국 혼잡을 가중시켜 오버플로우나 데이터 손실을 발생시킨다. 따라서, 이러한 네트워크의 혼잡을 피하기 위해 송신측에서 보내는 데이터의 전송 속도를 강제로 줄이게 된다.

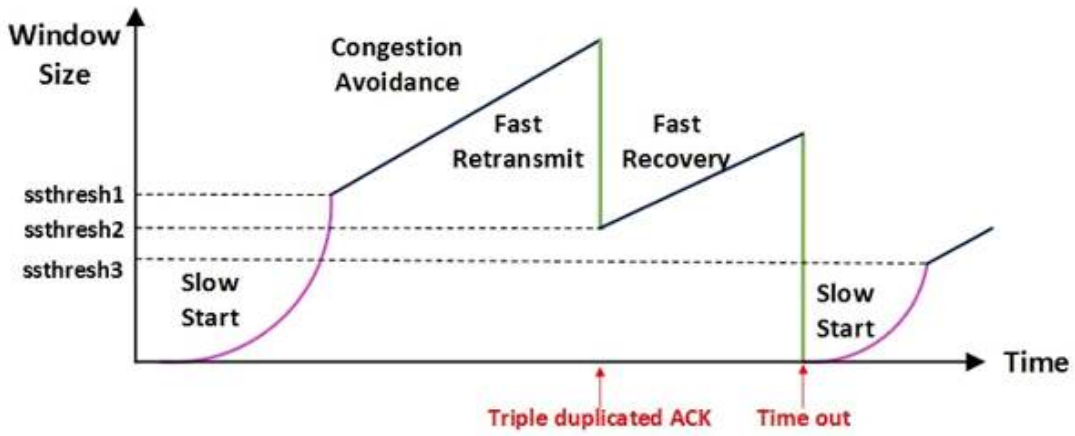
(1). AIMD (Addtive Increase Multicative Decrease)

합 증가 / 곱 감소 알고리즘이라고 한다
처음에 패킷 하나를 보내는 것으로 시작하여 전송한 패킷이 문제 없이 도착한다면 Window Size를 1씩 증가시키며 전송하는 방법이다. 만약, 패킷 전송을 실패하거나 TIME_OUT이 발생하면 Window Size를 절반으로 감소시킨다.
이 방식은 공평하다. *여러 호스트가 한 네트워크를 공유하고 있으면 나중에 진입하는 쪽이 처음에는 불리하지만, 시간이 흐르면 평형 상태로 수렴하게 되는 특징이 있다*
문제점은 초기 네트워크의 높은 대역폭을 사용하지 못하고 네트워크가 혼잡해지는 상황을 미리 감지하지 못하여 혼잡해지고 나서야 대역폭을 줄이는 방식이다.



(2). Slow Start

AIMD가 네트워크의 수용량 주변에서는 효율적으로 동작하지만, 처음에 전송 속도를 올리는 데 시간이 너무 길다는 단점이 있다
Slow Start는 AIMD와 마찬가지로 패킷을 하나씩 보내는 것부터 시작한다. 이 방식은 패킷이 문제 없이 도착하여 ACK 패킷마다 Window Size를 1씩 늘린다. 즉, 한 주기가 지나면 Window Size는 2배가 된다
따라서 그래프의 모양은 지수 함수 꼴이 된다
혼잡 현상이 발생하면 Window Size를 1로 떨어뜨린다
처음에는 네트워크의 수용량을 예측할 수 있는 정보가 없지만 한번 혼잡 현상이 발생하고 나면 네트워크 수용량을 어느정도 예상할 수 있으므로 혼잡 현상이 발생하였던 Window Size의 절반까지는 이전처럼 지수 함수 꼴로 Window Size를 증가시키고 그 이후부터는 완만하게 1씩 증가시키는 방식이다 .
미리 정해진 임계값(threshold)에 도달할 때까지 윈도우의 크기를 2배씩 증가시킨다.
Slow Start라는 이름을 사용하지만, 매 전송마다 2배씩 증가하기 때문에 전송되어지는 데이터의 크기는 **지수함수적**으로 증가한다. 전송되는 데이터의 크기가 임계 값에 도달하면 **혼잡 회피** 단계로 넘어간다



[혼잡 회피(Congestion Avoidance)]

윈도우의 크기가 임계 값에 도달한 이후에는 데이터의 손실이 발생할 확률이 높다
따라서 이를 회피하기 위해 윈도우 크기를 선형적으로 1씩 증가시키는 방법이다.
수신측으로부터 일정 시간 동안까지 ACK를 수신하지 못하는 경우* 타임 아웃의 발생 : 네트워크 혼잡이 발생했다고 인식 * 혼잡 상태로 인식된 경우 --> 윈도우의 크기 세그먼트의 수를 1로 감소시킨다. --> 동시에 임계값을 패킷 손실이 발생했을 때의 윈도우 크기의 절반으로 줄인다

[빠른 재전송(Fast Retransmit)]

패킷을 받는 수신자 입장에서는 세그먼트로 분할된 내용들이 순서대로 도착하지 않는 경우가 생길 수 있다.
예를들어 1, 2, 3, 4, 5... 번의 데이터가 순서대로 와야하는데 2, 3 다음 5번이 온 것이다.
이런 상황이 발생했을 때 수신측에서는 순서대로 잘 도착한 마지막 패킷의 다음 순번을 ACK 패킷에 실어서 보낸다. 그리고 이런 중복 ACK를 3개 받으면 재전송이 이루어진다.
송신 측은 자신이 설정한 타임아웃 시간이 지나지 않았어도 바로 해당 패킷을 재전송할 수 있기 때문에 보다 빠른 전송률을 유지할 수 있다.
참고로, 송신측에서 설정한 타임아웃 까지 ACK를 받지 못하면 혼잡(Congestion)이 발생한 것으로 판단하여 혼잡 회피를 한다.

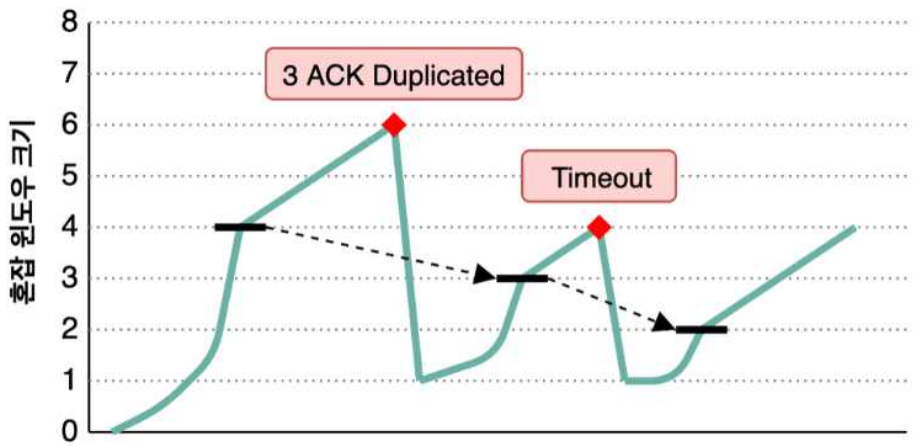
[빠른 회복(Fast Recovery)]

혼잡한 상태가 되면 Window Size를 1로 줄이지 않고 절반으로 줄이고 선형 증가시키는 방법이다
빠른 회복 정책까지 적용하면 혼잡 상황을 한번 겪고 나서부터는 순수한 AIMD 방식으로 동작하게 된다.

TCP 혼잡 제어 정책

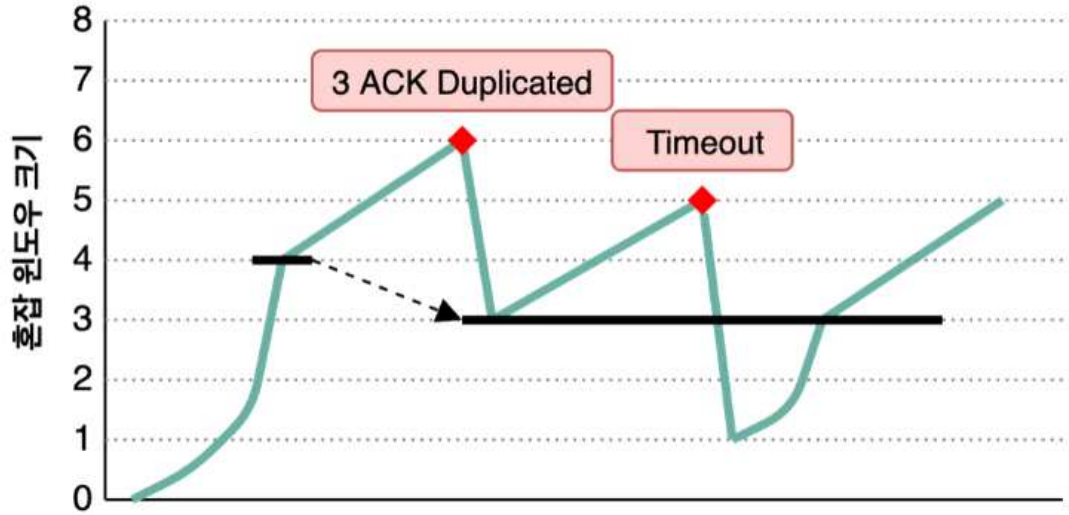
TCP Tahoe

TCP Tahoe는 처음에는 Slow Start를 사용하다가 임계점에 도달하면 AIMD 방식을 사용한다. 그러다가 3 ACK Duplicated 또는 타임아웃이 발생하면 혼잡이라고 판단하여 임계점은 혼잡이 발생한 윈도우 크기의 절반으로, 윈도우 크기는 1로 줄인다.
이 방식은 혼잡 이후 Slow Start 구간에서 윈도우 크기를 키울 때 너무 오래걸린다는 단점이 있다. 1부터 키워나가야 하기 때문이다.
그래서 나온 방법이 빠른 회복 방식을 활용한 TCP Reno 이다.



TCP Reno

TCP Tahoe와 마찬가지로 Slow Start로 시작하여 임계점을 넘어가면 AIMD 방식으로 변경한다.TCP Tahoe와의 차이점은 바로 3 ACK Duplicaed와 타임아웃을 구분한다는 점이다. TCP Reno는 3 ACK Duplicated가 발생하면 빠른회복 방식을 사용한다.
즉, 윈도우 크기를 1로 줄이는 것이 아니라 반으로 줄이고 윈도우 크기를 선형적으로 증가시킨다. 그리고 임계점을 줄어든 윈도우 값으로 설정한다.
만약, 타임아웃이 발생하면 TCP Tahoe와 마찬가지로 윈도우 크기를 1로 줄이고 Slow Start를 진행한다. 이때는 임계점을 변경하지 않는다.



흐름제어와 혼잡제어의 차이를 간단하게 설명한 그림![아래]

출처: https://rok93.tistory.com/entry/네트워크-TCP-흐름제어혼잡제어
출처: https://jsonsang2.tistory.com/17
출처 : https://steady-coding.tistory.com/507
출처 :
https://velog.io/@mu1616/TCPIP-%ED%98%BC%EC%9E%A1-%EC%A0%9C%EC%96%B4