Improving the Start-up Behavior of a Congestion Control Scheme for TCP

수학과 김호진 (2016314786)

[1. Introduction to TCP congestion]

1981년 전송 제어 프로토콜(TCP)의 지정 이후, TCP의 구현은 혼잡 방지 알고리즘(congestion control)과 같은 다양한 성능 향상 알고리즘을 통해 강화되었다. TCP 연결에서 혼잡 제어 체계(congestion control scheme)는 송신자가 네트워크로 송신할 수 있는 시기와 양을 결정하는 TCP 매개변수에 대한 합리적인 운영 값을 동적으로 계산하기 위해 수신자로부터 받은 acknowledgement(ACK)를 사용한다. Van Jacobson의 논문 'Congestion avoidance and control'을 통해 일반적으로 Slow-Start라고 알려진 혼잡 방지 알고리즘 및 제어 체계(congestion avoidance and control scheme)가 제시되었고, 이후 Fast Retransmit 및 Fast Recovery Algorithm이 포함되도록 체계가 수정되었다.

또한, Selective acknowledgement(SACK) 옵션을 TCP 표준에 추가하자는 의견이 제시되었다. SACK를 활용하면 수신자는 송신자에게 성공적으로 도착한 세그먼트에 대해 알릴 수 있게 되고, 이를 통해 송신자는 누락된 세그먼트에 대해서만 재전송할 필요를 가진다. Forward Acknowledgment congestion control algorithm 등 SAK와 함께 사용되도록 설계된 알고리즘들은 네트워크의 데이터 흐름에 대해 보다 정확한 제어를 가능하게 한다. SACK를 비롯해 이와 관련된 여러 알고리즘은 다중 패킷 손실로부터의 복구 문제를 해결했지만, 이들을 사용하기 위해서는 cooperative receivers가 필요하다. 그에 반해, 본 논문에서 주로 다루고 있는 Fast Retransmit 및 Fast Recovery algorithm은 TCP 구현에 있어 송신자 측의 간단한 변경만을 필요로 하며 기존의 TCP 사양과도 일치한다.

논문이 발표될 당시 TCP 구현에서 혼잡 제어 체계는 (1) Slow-start, (2) Fast Retransmit and Fast Recovery Algorithm이라는 두 가지 주요 부분으로 구성되었다. Slow-start를 사용하는 경우, TCP 송신자는 Congestion window를 하나의 세그먼트로부터 출발하여 지수적으로 증가시킨다. 이후 Congestion window가 ssthresh라는 경계에 도달하게 되면, 이를 선형적으로 증가시켜 사용가능한 네트워크 용량을 알아낸다. 그러므로 송신자가 패킷을 네트워크에 넣자마자 패킷이 네트워크를 떠날 수 있도록 equilibrium operating point 추정 값을 정하는 것이 알고리즘 성능에 있어 핵심적인 역할을 한다. 패킷 손실은 Fast Retransmit algorithm 또는 Retransmission timeout(RTO) 이후의 Slow-start를 통해서만 복구가 가능하다. 그런데 Fast Retransmit algorithm의 경우 다수의 패킷들을 복구할 수 없으므로 해당 경우에서 연결은 idle한 상태로 시간이 초과되기를 기다렸다가 1.5초가 지난 뒤 Slow-start를 통해 손실된 패킷을 복구할 수밖에 없다.

수신자는 지금까지 받은 것들 중 가장 높은 순서의 sequence number만을 인지한다. 그렇기 때문에 순서가 뒤바뀐 패킷을 수신하게 되면 가장 높은 sequence number에 대한 acknowledgement(duplicate ACK)를 다시 생성한다. 이에 기반하여 Fast Retransmission algorithm은 송신자가 세 개의 duplicate ACK를 받았을 때 해당 세그먼트가 손실되었다고 추론한다. 이렇게 손실되었다고 판단된 세그먼트는 duplicate ACK를 통해 승인된 바로 다음의 숫자를 sequence number로 가진다고 여겨지고, 송신자에 의해 재전송 된다. 그리고 ssthresh와 congestion window는 전송 속도를 늦추기 위해서 Fast Retransmit 이전 congestion window 크기의 약 절반 수준으로 낮아진다. 정리하자면, Fast Recovery algorithm은 Fast Retransmission 이후 congestion window를 지수적으로 증가시키는 대신 전송자가 속도를 늦추고 선형적으로 증가하도록 congestion window와 ssthresh를 조정하는 방식을 일컫는다. 본 논문에서는 단일 패킷이 손실된 상황과 동일한 window에서 패킷들이 다중 손실된 상황에서 호출된 각각의 Fast Retransmission algorithm이 어떻게 작용하는지 살펴보고, 이 상황에서 TCP가 가지는 문제점을 다루고 있다.

[2. Problems]

앞서 설명했듯이 TCP 혼잡 제어 체계는 TCP 매개변수에 대한 합리적인 운영 값을 동적으로 계산하기 위해 수신자로부터 받은 ACK를 사용한다. 그런데 처음 송신을 시작하게 되면, 송신자는 네트워크의 용량 및 수신자에 대해 어떠한 정보도 가지고 있지 않기 때문에 매개변수를 기본값(default value)으로 설정하게 된다. 이러한 기본값은 매우 임의적이므로 송신자는 너무 많은 패킷을 매우 빠르게 출력하게 되고, 결국 같은 window상의 여러 패킷을 잃게 만든다. 그리고 이러한 손실로부터의 복구는 불필요한 시간 비용을 소모한다.

또한 Fast Retransmission algorithm을 사용하는 경우, 하나의 패킷이 손실된 경우에서는 잘 작동하지만 다수의 패킷들이 손실되었을 때는 알고리즘에 의해 오직 하나의 패킷 손실만 복구되고 나머지 패킷들은 일반적으로 긴 시간의 RTO가 지난 후 Slow-start에 의해 복구가 되는 것을 확인할 수 있다. 세그먼트의 재전송을 발생시키기 위해서는 또 다른 Fast Retransmit algorithm이 활성화되거나 RT0를 지나야 하는데 연결이 idle하고 duplicate ACK가 예상되지 않기 때문에 Fast Retransmit에 의한 세그먼트 복구가 불가능한 것이다

송신자가 받는 각각의 duplicate ACK에 대하여 송신자는 하나의 세그먼트마다 congestion window를 열 수 있다. 그러나 데이터의 전체 window를 초과하여 가질 수는 없기 때문에 non-duplicate ACK가 들어오기 전까지 어떠한 세그먼트도 전송할 수 없다. 마지막 전송 이후 이미 Roundtrip Time(RTT)을 한참 지났기 때문에 패킷은 네트워크에 고착화되고 수신자에게 도달했을 가능성은 매우 낮아진다. 따라서, 추가전송 없이 ACK는 더 이상 유발되지 않으며 세그먼트를 재전송하는 데 Fast Retransmission를 사용할 수 없게 된다. 이러한 이유들로 인해 1.5초가 지난 후 발생하는 RTO을 기다리는 것이 재전송을 발생시키는 유일한 방법이 되었다.

이러한 시간 소모적인 손실 복구의 근본적인 문제는 exponential Slow-start mode에서 송신자가 대량의 패킷들을 보내게 하여 다수의 패킷 손실을 유발하는 ssthresh의 초기 임의 값과 Fast Retransmission algorithm의 손실된 패킷 복구 실패에서 비롯된다. 송신자에게 RTO를 오랫동안 기다리게 하는 이러한 상황은 start-up 기간 동안 TCP 성능을 대폭 감소시키는 주요 원인이 되었다.

경우에 따라, 세그먼트 손실이 없음에도 알고리즘이 활성화되는 False Fast Retransmits이 발생하기도 한다. 이러한 False Fast Retransmission은 실제로는 정체가 없어도 송신자에게 'linear mode'로의 전환을 강요한다.

[3. Solutions]

네트워크의 복잡도와 크기가 증가함에 따라 합리적인 운영 매개변수를 찾기 위해 네트워크를 조사하는 Start-up 기간이 길어졌다. 또한 FTP, HTTP 등 수많은 응용 프로토콜이 짧은 전송을 위해 TCP를 사용하면서 상대적으로 적은 수의 데이터 세그먼트가 전달되고 TCP가 안정화 상태에 도달하기 전 연결이 종료되었다. 이로 인해 TCP 연결을 통한 대부분의 데이터 전송에 있어 Start-up 기간은 중요한 역할을 하게 되었고, 해당 기간 동안의 TCP 성능 역시 매우 중요해졌다.

Start-up 기간 동안 TCP의 성능을 향상시키기 위해서는 시간 비용이 많이 소모되는 상황을 찾아야 한다. 패킷이 보내지는 양이 급증하면서 여러 패킷들이 손실되고, Fast Retransmission의 손실된 패킷 복구 실패는 전송자로부터 RTO를 기다리게 하기 때문에 TCP 성능을 저하시키는 대표적인 문제이다. 이러한 문제를 해결하기 위해서는 다중 패킷 손실을 유발하는 패킷 급증을 줄여야 하며, Fast Retransmit algorithm의 수정을 통해 동일한 window에서 발생한 다중 패킷 손실로부터의 복구를 가능하게 만들어 RTO를 기다릴 필요성을 줄여야 한다. 첫번째 방식을 위해서는 ssthresh의 더 나은 초기 값을 찾아야 하며, 두번째 방식은 손실된 세그먼트를 복구하기 위한 기존보다 더 적극적인 Fast Retransmit algorithm을 필요로 한다.

앞서 언급했듯이 다중 패킷 손실을 일으키는 패킷의 큰 급증을 피하기 위한 하나의 방법은 임의의 기본값보다 더 좋은 ssthresh를 선택하는 것이다. 그렇기 때문에 ssthresh의 초기 값은 매우 중요하다. lower ssthresh에서는 congestion window를 ssthresh에 도달할 때까지 지수적으로 증가시켜 열고, 이후 window당 segment 하나씩 추가적으로 열 수 있게 만들어 과잉공급 없이 파이프 용량을 알아낸다. 만약 초기 sstresh 값이 너무 작게 설정되었다면 전송자가 너무 이르게 additive increase mode로 전환하면서 성능이 저하될 수 밖에 없다. 그럴 경우, 패킷 손실은 없을지라도 송신자의 전송 속도가 너무 느리기 때문에 전송 시간이 오래 걸리게 된다. 따라서 전체 네트워크 용량에 근접하도록 threshold를 추정한 다음 속도를 줄여 남은 용량을 조사할 수 있도록 만들어야 한다. '대역폭-지연 곱(bandwidth-delay product)'은 이를 가능하게 하는 대표적인 추정 방법이다. 최악의 경우, 추정치가 너무 낮아지면서 송신자가 너무 보수적이게 되고 그 결과 여러 패킷을 잃은 뒤 이를 복구하기 위해서 RTO을 기다리는 것보다 성능이 더 안 좋게 나타날 수도 있다. 그러나 네트워크의 집합적 성능을 고려했을 때 지나치게 공격적인 송신자보다는 보수적인 송신자가 바람직하다.

다음으로 제안하는 변경사항은 Fast Retransmit algorithm을 수정하는 것이다. 이를 통해 ACK가 제공하는 단서와 의미를 더 효과적으로 사용할 수 있게 만들고, RTO를 기다리는 대신 Slow-start를 이용한 복구 프로세스가 더 빨리 시작되도록 한다. 또한 이렇게 만들어진 Fast Retransmit phase algorithm은 다중 패킷 손실에 대한 복구를 가능하게 한다. Fast Retransmit phase에 있다는 것은 동일한 Window에서 발생한 다중 패킷 손실이 아직 완벽하게 복구되지 않았음을 의미한다. 따라서 송신자는 phase에 있는 동안 Slow-start를 이용하여 모든 패킷이 복구될 때까지 패킷을 계속해서 재전송할 수 있다. 이러한 방식은 때때로 불필요한 재전송을 유발하기도 하지만, ACK의 단서를 통해 송신자에게 어떤 세그먼트가 수신되지 않았는지 제대로 알려줄 수 있어 불필요한 재전송의 수는 전반적으로 감소한다. 기존 알고리즘과 비교했을 때 Fast Retransmit phase algorithm은 더 공격적인 것처럼 보인다. Fast Retransmit phase algorithm에서 RTO는 백업 용도로 작동하며 다른 모든 방법들이 실패한 이후에 사용된다. Fast Retransmit phase에서 acknowledgement는 송신자에게 돌아오고 있기 때문에 여전히 네트워크에 데이터가 존재한다는 사실을 알려줄 수 있다. 이러한 이유들로 송신자는 모든 패킷들의 손실이 복구될 때까지 패킷을 계속 재전송할 수 있다. Fast Retransmit phase의 개념은 False Fast Retransmits 문제를 해결하는 데에도 사용된다. Fast Retransmit의 세그먼트와 동일한 window로부터 세그먼트를 인식하는 duplicate ACK는 지속적인 혼잡을 나타내지 않는다. 그러므로 송신자는 Fast Retransmit phase에 있는 한 모든 duplicate ACK를 무시하고 False Fast Retransmission을 제거할 수 있다.

다중 패킷 손실을 복구하는 동안, 큰 값의 ACK는 전송되는 세그먼트의 급증을 유발할 가능성이 있고 이러한 갑작스러운 증가는 세그먼트의 추가적인 손실을 일으킬 수 있다. 이러한 문제를 해결하기 위해서는 TCP가 연속적으로 출력할 수 있는 세그먼트의 수를 제한해야 한다. 본 논문에서는 이에 대해 추후에 논의하기로 언급하며 자세한 내용은 다루지 않았다.

실제 네트워크에서는 수많은 요소들에 의하여 throughput이 결정되므로 지금까지 제안된 변경사항이 적용된 TCP와 그렇지 않은 TCP의 성능을 비교할 때에는 많은 주의사항이 요구된다. 그렇기 때문에 본 논문에서도 수많은 실험들을 반복해서 진행했고, 그때마다 기울기와 패킷 손실 수가 조금씩 다른 그래프를 구했다. 하지만 패킷 손실에 대한 혼잡 제어 체계의 동작은 조금씩 다른 그래프들 사이에서도 유용하며, 논문에서 새롭게 제안된 알고리즘은 다중 패킷 손실로부터의 복구를 가능하게 만들기 때문에 변경된 사항들로 인해 Start-up 기간 동안 TCP가 더 나은 성능을 발휘할 수 있는 것은 분명하다. 또한 불필요한 timeout을 제거하면서 실제 네트워크의 경우에서도 start-up 기간 동안 TCP의 획기적인 성능 향상이 있다는 것 역시 증명해냈다. 많은 TCP 연결이 오래 지속되지 않으면서 Start-up 기간의 TCP 성능은 점점 더 중요해졌으며, 제안된 알고리즘은 송신자의 TCP 구현에 있어서 매우 간단한 변경만을 필요로 하는 장점을 가진다. 본 논문은 당시 TCP가 가진 혼잡 제어 문제를 정확히 지적해냈으며 제안된 변경사항들이 실제 TCP 구현에 적용되면서 Start-up 기간 중 TCP 혼잡 제어 체계의 성능을 향상시키는데 큰 영향을 주었다.