종단간 순방향/역방향 전송 지연에 따른 TCP Reno 혼잡제어 알고리즘 성능향상

한규형*, 김은기
*국립 한밭대학교 정보통신전문대학원
e-mail :dabak76@gmail.com, egkim@hanbat.ac.kr

Performance improvement of TCP Reno congestion control algorithm using end-to-end estimation of forward/backward delay variation

Kyu-Hyeong Han*, Eun-Gi Kim
Graduate School of Information & Communications, Hanbat National University

요 약

기존 TCP Reno 의 혼잡 제어는 트래픽에 수동적으로 동작하는 방법으로서 혼잡이 이미 발생한 상태에서 동작하게 되므로 발생 시점의 라우터 버퍼는 이미 최대치에 도달해 있게 된다. 따라서 이후에 도착하는 모든 패킷은 폐기되므로 이 패킷들을 전송한 모든 송신원은 거의 동시에 윈도우 크기를 줄이는 Slow-start 단계에 들어가게 되어 일시적으로 링크 사용률이 떨어지는 전역 동기화(global synchronization)가 발생하게 된다. 이러한 문제를 해결하기 위해서는 네트워크의 혼잡이 발생하기 전에 능동적으로 대처하는 방안이 필요하다. 본 논문에서는 새로운 RTT 계산 알고리즘인 순방향/역방향 전송지연 알고리즘을 이용하여 네트워크의 혼잡을 미리 예측하고 네트워크 혼잡에 능동적으로 대처할 수 있는 새로운 알고리즘을 제안한다. 본 논문에서는 리눅스(Linux) 커널(Kernel)의 TCP Reno 의 흐름제어 및 혼잡제어를 수정하여 네트워크 혼잡에 능동적으로 대처 할 수 있는 새로운 TCP Reno 를 설계, 구현하였다.

1. 서론

UDP 가 비연결지향 프로토콜로서 신뢰성이 보장되지 않는데 반하여 TCP 는 연결지향 프로토콜로서 신뢰성 있는 데이터 전송을 위해 누적 승인 방식(Cumulative Acknowledgement)사용, RTO(Retransmission Time-Out)계산과 재전송(Retransmission)을 수행하며, 성능 향상을 위해 흐름 제어(Flow Control) 그리고 혼잡 제어(Congestion Control)등의 알고리즘을 사용하고 있다.[1][2][3][4][5]

TCP 의 혼잡 제어는 다음과 같은 네 개의 알고리즘인 저속 출발(Slow start), 혼잡 회피(Congestion Avoidance), 빠른 재전송(Fast Retransmit), 빠른 복구(Fast Recovery) 알고리즘으로 구성된다.[1][2][3][4][5]

위와 같은 TCP 의 혼잡 제어는 트래픽에 수동적으로 동작하는 방법으로서 망의 혼잡도를 반영하지 못한다. 본 논문에서는 순방향과 역방향에서 어느 정도의 혼잡과 전송 지연이 발생하였는지 알 수 있는 새로운 RTT 측정 방식인 순방향/역방향 전송지연 알고리즘을 이용하여 TCP Reno 의 혼잡제어와 흐름제어를 수정하였다.

본 논문의 구성은 2 장에서 TCP Reno 의 기본적인 혼잡제어 알고리즘과 새로운 RTT 측정 방식인 순방향/역방향 전송지연 알고리즘을 설명하고, 3 장에서는 제안된 알고리즘을 설명한다. 4 장에서는 제안된 알고리즘을 실제 리눅스(Linux) 커널(Kernel)에 구현하는 방법을 소개하고 5 장에서는 본 논문의 결론과 향후 연구되어야 할 내용을 기술한다.

2. 관련 연구

2.1 TCP 혼잡제어 알고리즘

TCP 혼잡제어(Congestion Control Mechanism)의 주 목적은 송신원과 수신원 사이에 설정된 연결의 상태에 따라서 송신원의 전송 속도와 직결되는 송신원의 윈도우의 크기를 조절함으로써 전체 네트워크의 과부하로 인한 폭주(Congestion Collapse)를 미연에 방지하고, 패킷 손실이 발생한 경우 이를 빠른 시간 내에 복구 하는 것이다.

다음은 TCP Reno 의 기본적인 흐름제어 알고리즘과 혼잡제어 알고리즘에 대해 설명한다.

저속 출발(Slow Start) 알고리즘을 사용하기 위해 송신자는 수신자에 의해서 제공된 윈도우 이외에 혼잡 윈도우(cwnd: congestion window)라는 새로운 윈도우를 추가한다. 연결 설정이 이루어지고 나면, 초기 cwnd 값은 하나의 세그먼트로 설정된다. 송신자는 각 세그먼트의 확인 응답이 도착할 때마다 cwnd 값을 한 세그먼트 단위씩 증가하게 된다(cwnd = cwnd + 1).

혼잡 회피 알고리즘은 저속 출발에 사용되는 혼잡 윈도우(cwnd)와 저속출발 기준점인 ssthresh 라는 새로운 변수를 두고 망의 혼잡에 대해 다음과 같은 기능을 수행하여 적절히 대응한다.

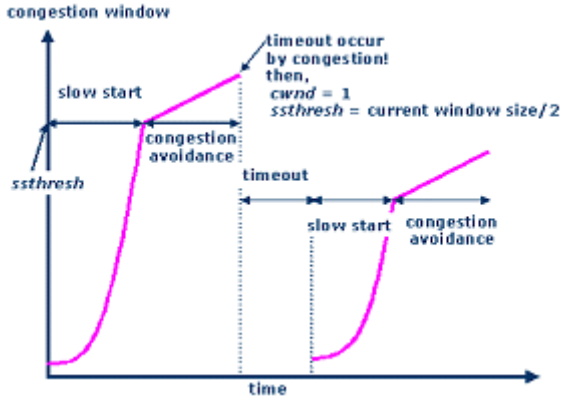
• 연결의 초기에 cwnd 는 1 세그먼트, ssthresh 는 무한대의 값으로 설정한다.

• 저속출발 도중 혼잡이 발생하면 현재 cwnd 크기의 1/2 을 ssthresh 에 저장하고, cwnd 는 1 세그먼트로 초기화 후 다시 저속출발을 수행한다.

• 저속출발 수행 중 cwnd 가 이전에 설정한 ssthresh 보다 커지게 되면 확인응답이 도착할 때 마다 다음 수식과 같이 증가하게 된다.

$$cwnd = cwnd + (segsize \times segsize / cwnd) + (segsize / 8)$$

이는 cwnd 를 1/cwnd 씩 증가하여 지수적인 증가에서 선형적인 증가로 변하도록 한다. 즉, 혼잡이 예상되는 부분에서 미리 송신되는 세그먼트의 양을 줄여 혼잡을 회피하게 된다. 그림 1 은 저속 출발과 혼잡회피의 기본 알고리즘을 도식화 하였다.

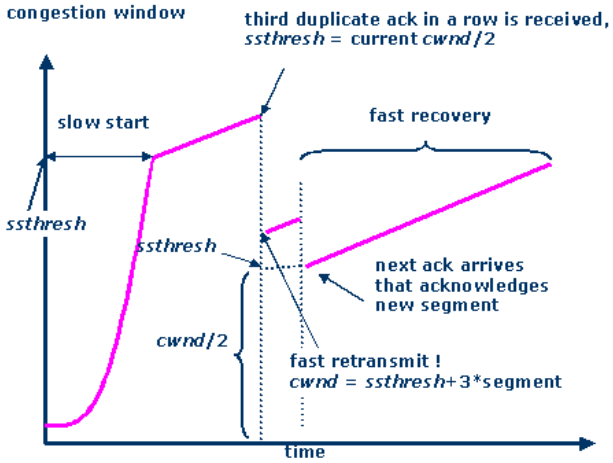


(그림 1) 저속 출발과 혼잡회피의 혼합된 동작

빠른 재전송과 빠른 복구 알고리즘은 연속적인 데이터를 전송하는 환경에서 망이 혼잡하여 세그먼트의 분실이 발생할 때 송신자는 분실된 세그먼트에 대해서 타임아웃이 발생되기 전에 수신자로부터 중복된 확인 응답을 받을 수 있다.

단순한 중복된 확인응답으로는 세그먼트가 분실이 발생한 것인지 또는 단지 망 내에서 세그먼트의 순서가 바뀌었기 때문인지 판단할 수 없다. 그러나 3 개의 중복된 확인 응답을 계속해서 받는다면 송신자는 해당 세그먼트가 분실되었을 가능성이 크다고 판단하게 되어 타임 아웃이 발생하기를 기다리지 않고 바로 재전송을 수행하게 되는데 이것을 빠른 재전송(Fast retransmit)이라 한다.

중복된 확인응답을 계속해서 수신할 수 있다는 것은 망의 혼잡상태가 심각하지 않는 다는 것을 의미하므로 재전송을 수행하고 난 후 저속 출발을 수행하여 갑작스럽게 흐름을 감소시키는 대신 혼잡 회피를 수행하여 망의 흐름을 어느 정도 유지시키게 되는데 이것을 빠른 복구(Fast Recovery)라 한다. 그림 2 는 빠른 재전송과 빠른 복구의 동작을 도식화 하였다.

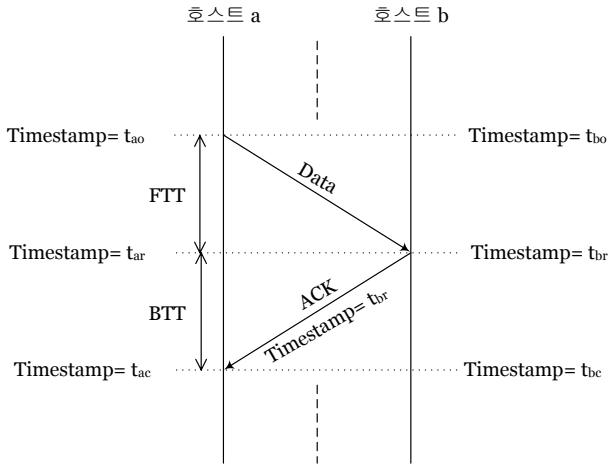


(그림 2)빠른 재전송과 빠른 복구의 동작

2.2 종단간 순방향/역방향 전송 지연 측정 알고리즘

현재 네트워크의 트래픽을 측정하기 위한 방법으로는 RTT 측정 방법이 있으며, 이 방법으로 망의 혼잡 정도를 측정할 수 있다. 그러나 RTT 는 패킷의 왕복 시간만을 측정하기 때문에 패킷의 송수신시 순방향과 역방향에서 어느 정도의 혼잡과 전송지연이 발생하였는가는 알 수가 없다.[6]

본 논문에서는 기존에 연구된 순방향/역방향 전송 지연 측정 알고리즘을 이용하여 TCP Reno 를 수정하였다. 여기서는 기존의 순방향/역방향 전송지연 측정 알고리즘을 설명한다. 송신원 호스트 a 와 수신원 호스트 b 가 있을 때, a 호스트에서 b 호스트로 데이터를 보낼 때의 데이터 경로를 순방향이라고 하고 반대의 경로를 역방향이라고 한다.



(그림 3) 종단간 순방향/역방향 전송 지연 측정 파라메타

그림 3 은 호스트 a 와 b 의 패킷 교환과 종단간 순방향/역방향 전송 지연 시간 측정을 위한 요소들을 자세하게 보여주고 있으며, 알고리즘을 설명하기 위하여 필요한 변수들을 다음과 같이 정의 한다.

- t_{hi} : 호스트 h 의 TSC 값, i=o(originate), r(receive), c(current)
- TD_{ab} : 호스트 a 와 b 에 세트 된 TSC 의 차이

- RTT_{cp} : 현재 패킷의 RTT 값.
- RTT_{min} : 지금까지의 RTT 측정값들 중 최소값.

- FTT_{cp} : 현재 패킷의 순방향 전송 시간(Forward Transmission Time).
- FTT_{ref} : RTT_{min} 일 때의 FTT_{cp} 값($FTT_{ref} = RTT_{min}/2$)
- FTT_{dif} : FTT_{ref} 와 비교한 FTT_{cp} 의 증가 또는 감소 값.

- BTT_{cp} : 현재 패킷의 역방향 전송 시간(Backward Transmission Time).
- BTT_{ref} : RTT_{min} 일 때의 BTT 값($BTT_{ref} = RTT_{min}/2$)
- BTT_{dif} : BTT_{ref} 와 비교한 BTT_{cp} 의 증가 또는 감소 값

그림 4 는 새로운 알고리즘을 의사 코드(pseudo code)로 나타낸 것이다. 알고리즘의 동작을 간단히 설명하면 다음과 같다. 3~6 번 줄에서는 지금까지 전송된 모든 패킷의 RTT 값들 중에서 가장 작은 값을 찾아, 이 값을 기준으로 FTT_{ref}, BTT_{ref}, TD_{ab} 값을 계산한다. 그리고, 8~11 줄에서는 현재 패킷이 전송되어 응답이 수신될 때까지의 여러 시간 값, 송수신 호스트가 갖는 시간 차, 그리고 FTT_{ref}, BTT_{ref} 등을 이용하여 현재 패킷의 순방향/역방향 전송 지연 시간의 변화를 계산한다.

```
01:  RTT_min = big number;
02:  until (end of packet) {
03:      if (RTT_min > RTT_cp) {
04:          RTT_min = RTT_cp;
05:          FTT_ref = BTT_ref = RTT_min / 2;
06:          TD_ab = t_ac - t_br - BTT_ref;
07:      }
08:      FTT_cp = t_br - t_ac + TD_ab;
09:      BTT_cp = t_ac - t_br - TD_ab;
10:      FTT_dif = FTT_cp - FTT_ref;
11:      BTT_dif = BTT_cp - BTT_ref;
12:  }
```

(그림 4) 순방향/역방향 전송 지연 측정 알고리즘

3. 순방향/역방향 전송지연을 지원하는 TCP 설계

본 논문에서는 기존 TCP Reno 의 흐름 제어 및 혼잡 제어를 기본으로 순방향/역방향 전송지연에 따라 흐름 제어 및 혼잡 제어방식을 수정하여 전송 효율이 높은 TCP 를 설계하였다.

3.1 저속 출발(Slow Start) 수정

기존 방식에서는 네트워크 혼잡도를 반영하지 않고 단순히 수신원에서 보내어 지는 확인 응답에 따라 cwnd 를 증가 시켰다. 하지만 네트워크 상태가 점점 혼잡해 지고 있는 상황에서 cwnd 를 과도하게 높인다면 네트워크에 혼잡을 초래할 수 있다. 그리고 RTT 가 늘어나는 상황이지만 순방향으로는 전송지연이 발생하지 않고 역방향으로 전송지연이 발생하고 있는 상황이라면 순방향으로는 더 많은 패킷을 보내도 된다. 본 논문에서는 순방향(FTT)으로 전송지연이 발생하면 cwnd 를 적게 증가시키고, 순방향으로 전송지연이 발생하지 않으면 cwnd 를 많이 증가시켜 전송속도를 높임으로써 네트워크의 혼잡도를 반영시켰다.

$$cwnd = cwnd + \left(1 - \left(\frac{cur\_FTT - pre\_FTT}{cur\_FTT}\right)\right)$$

(수식 1) 저속 출발 시 cwnd 증가식

수식 1 은 기존에 1 만큼 증가시키던 것을 순방향(FTT) 전송 지연의 변화량 만큼 증가 시키기 위함이다.

3.2 빠른 재전송과 빠른 복구 수정

3 개의 중복 확인 응답을 수신시 현재 cwnd 의 반을 ssthresh 에 저장하고, 3 개의 중복 확인 응답으로 패킷을 보내지 못했기 때문에 이에 대한 보상으로 cwnd 를 ssthresh + (segsize * 3)으로 해주었던 기존의 방법을 수식 2 처럼 수정하였다. 여기에서 ssthresh 는 빠른 복구의 시작점이 된다.

$$ssthresh = \left(\frac{cwnd}{2}\right) - \left\{\left(\frac{cur\_FTT - pre\_FTT}{cur\_FTT}\right) \times \left(\frac{cwnd}{2}\right)\right\}$$

$$cwnd = ssthresh + 3 \times segsize$$

(수식 2) 중복 확인응답 3 개 수신시

예를 들어 3 개의 중복 확인 응답이 발생했을 때, 순방향 전송지연이 낮아지고 있었다면, 즉, 순방향 전송지연(FTT)값이 120ms 에서 100ms 로 변하였다면, 순방향으로는 트래픽이 적게 발생된 상황이므로 ssthresh 값을 기존의 방식보다 증가시켜 Fast Recovery 의 시작점을 높였다.

이와 반대로 순방향으로 전송지연이 과도하게 발생하였다면 ssthresh 값을 기존의 방식보다 감소시켜 Fast Recovery 의 시작점을 낮추어 네트워크에 패킷의 전송률을 감소시켰다.

4. 순방향/역방향 전송지연을 지원하는 TCP 구현

리눅스 커널 2.6.10 을 수정하여 순방향/역방향 전송지연에 따라 흐름 제어와 혼잡 제어를 하는 TCP Reno 를 구현하였다.

4.1 새로운 RTT 계산 알고리즘 구현

새로운 RTT 계산 알고리즘을 위해 socket.h 에 새로운 구조체인 struct new_rtt 를 추가하고 내부 변수를 선언한다.

기존의 TCP 는 매 ACK 수신시 RTT 를 계산한다. 여기에 순방향(FTT)과 역방향(BTT) 전송 지연을 계산하는 함수를 추가한다.

```
void tcp_FTT_BTT_estimator(struct tcp_opt *tp,
u32 seq_rtt)
```

- tp : tcp 헤더를 가리키는 구조체
- seq_rtt : 계산된 RTT

4.2 저속 출발(Slow Start) 수정

TCP 에서 저속 출발과 혼잡 회피는 tcp_cong_avoid()에서 한다. 여기서 reno 버전으로 동작하는지 vegas 버전으로 동작하는지에 따라 다르게 분기 하는데 TCP Reno 버전은 tcp_reno_avoid() 에서 처리한다. tcp_reno_avoid()의 저속 출발부분을 이미 계산한 FTT 를 이용하여 수식 1 에서와 같은 방법으로 cwnd 를 증가 시킨다.

4.3 빠른 재전송과 빠른 복구 수정

중복 확인 응답 3 개 도착시 ssthresh 와 cwnd 를 수정하는 함수는 tcp_cwnd_restart()이다. 이 부분을 수식 2 에서와 같은 방법으로 수정한다.

5. 결론 및 향후 계획

본 논문에서는 기존의 TCP Reno 가 네트워크의 혼잡도를 수동적으로 반영하고 적절히 대응하지 못하는 문제점을 새로운 네트워크 혼잡 측정 방식인 순방향/역방향 전송지연 알고리즘을 이용하여 네트워크 혼잡도에 동적으로 동작하는 새로운 TCP Reno 를 제안한다. Slow Start 상태일 때 순방향으로 전송지연이 발생했는지에 따라 cwnd 의 증가 속도를 조절하였고 또한 빠른 재전송 시작점을 조절함으로써 동적으로 네트워크 상황에 대처하도록 하였다.

현재 본 논문에서 제안한 기법을 리눅스 커널 2.6.10 에 구현하였고, 구현된 TCP 를 이용하여 실제 네트워크에서 데이터를 전송할 때 전송효율을 기존의 TCP Reno 와 비교하는 실험을 할 것이다.

참고문헌

[1] W. Richard Stevens, “TCP/IP Illustrated, Volume1 The Protocols”, Addison-Wesley, 1994

[2] W. Richard Stevens, “TCP/IP Illustrated, Volume2 The Implementation”,Addison-Wesley, 1995

[3] W. Stevens, “TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms”, RGC2001, Jan.1997

[3] Behrouz A. Forouzan, “TCP/IP Protocol Suite”, McGraw-Hill, 2003

[4] RFC 793, “Transmission Control Protocol”, http://www.ietf.org

[5] RFC 2861, “TCP Congestion Windows Validation”, http://www.ietf.org

[6] 황순한, “A Study on the Measurement of end-to-end Forward/Backward Delay Variation”,한밭대학교,2005