단방향 지연 시간 추정 기법과 이를 이용한 응용

(One-Way Delay Estimation and Its Application)

최 진 희 † 유 혁 ††
(Jin-Hee Choi) (Hyuck Yoo)

요 약 컴퓨터 네트워크에서 단방향 지연을 추정하는 것은 해결하기 어려운 문제다. 단방향 지연 시간을 정확하게 추정하는 것은 이 값이 네트워크 성능과 이를 이용한 응용의 설계에 매우 중요한 역할을 담당하기 때문에 필수적으로 해결 해야 할 문제이기도 하다. RTT (Round Trip Time)가 자주 이러한 지연 시간의 근사값으로 사용되곤 하지만 이 값이 순방향과 역방향의 지연 시간의 합이기 때문에 실제 단 방향 지연 시간과는 큰 차이를 보이는 경우가 많다. 단방향 지연을 정확하게 추정하기 위해 이 논문에서는 분석적으로 순방향과 역방향 지연 시간을 유도해낼 수 있는 새로운 방법을 제안한다. 그리고 이를 바탕으로 비대칭 네트워크에서 TCP의 성능을 크게 향상시킬 수 있음을 보인다. 본 논문의 핵심은 논문에서 제안된 단방향 지연 시간 추정 기법이 RTT 추정 기법보다 정확하여 이를 이용한 TCP가초기 시작에서 네트워크의 가용 량을 보다 빨리 찾아내도록 할 수 있다는 것이다. RTT는 순방향과 역방향의 합이기 때문에 RTT에기반을 둔 어떤 프로토콜에도 본 논문에서 제시된 방법이 적용될 수 있다.

키워드 : 단방향 지연 시간, 성능 평가, TCP 성능, 비대칭 네트워크.

Abstract Delay estimation is a difficult problem in computer networks. Accurate one-way delay estimation is crucial because it serves a very important role in network performance and thus application design. RTT(Round Trip Time) is often used as an approximation of the delay, but because it is a sum of the forward and reverse delays, the actual one-way delay cannot be estimated accurately from RTT. To estimate one-way delay accurately, this paper proposes a new scheme that analytically derives one-way delay, forward and reverse delay respectively. We show that the performance of TCP can improve dramatically in asymmetric networks using our scheme. A key contribution of this paper is that our one-way delay estimation is much more accurate than RTT estimation so that TCP can quickly find the network capacity in the slow start phase. Since RTT is the sum of the forward and reverse delays, our scheme can be applied to any protocol that is based on RTT.

Key words : one-way delay, performance evaluation, TCP performance, asymmetric networks.

1. 서 론

이 논문에서 우리는 컴퓨터 네트워크의 기반적인 문제인 송신자(sender)와 수신자(receiver) 사이의 단방향 지연 시간을 추정하는 기법을 다룬다. 만약 송신자와 수신자 사이의 전역적인 시간 동기화(global time synchronization)가 성립되었다는 가정을 한다면 단방향 지연 시간을 얻는 것은 어려운 일이 아니다. 즉, 순방향 지연 시간은 단순히 수신자의 시간(clock)과 송신자의 타임스탬프(time stamp)의 차이를 구해서 얻을 수 있을 것이다. 그리고 이 값을 수신자가 ACK 패킷(packet)의 헤더(header)에 넣어 돌려주면 송신자가 순방향 지연 시간을 알 수 있게 된다. 마찬가지로 역방향 지연 시간도 쉽게 구할 수 있다. 하지만 불행하게도 많은 컴퓨터들의 시간이 서로 동기화되어 있지 않은 것이 일반적이기 때문에 이러한 방법으로 단방향 지연 시간을 얻을 수는 없다. 이미 수행되었던 많은 연구 성과들[1-3]에도 불구하고 클럭(clock) 동기화 문제는 패킷의 순방향과 역방향 지연을 계산할 정도까지 해결되지 않았다[4-6]. 게다가 인터넷 같은 이형적(heterogeneous)이고 거대한(massive) 네트워크라면 이러한 클럭 동기화는 더욱 어려운 문제가 된다.

† 비 회 원 : 고려대학교 컴퓨터학과
jhchoi@os.korea.ac.kr
†† 종신회원 : 고려대학교 컴퓨터학과 교수
hxy@os.korea.ac.kr
논문접수 : 2004년 12월 6일
심사완료 : 2005년 2월 16일

RTT는 보통 단방향 지연시간의 근사값으로 사용된다. RTT는 종단간 통신(end-to-end communication)에서 핵심적인 매개변수인데, 보통 송신자에서 타임아웃(timeout)을 결정하고 전송량(transmission rate)을 계산하는데 사용되어 왔다. 이러한 프로토콜에서 RTT를 사용하는데 있어 그 기본 가정은 대상 네트워크가 대칭적(symmetric)이라는 것이다. 환언하면, 순방향과 역방향 지연 시간이 RTT[7]의 1/2에 근사 한다는 것이다. 하지만 현재 흔히 볼 수 있는 많은 네트워크들-케이블 모뎀 네트워크, 여러 3G 데이타 네트워크, ADSL 네트워크, 그리고 위성 네트워크 등-은 보통 역방향 링크의 대역폭 보다 순방향 링크의 대역폭이 훨씬 크다[8-12]. 게다가 인터넷 라우팅에 대한 많은 연구 결과에서도 볼 수 있듯이 인터넷의 경로 자체가 비대칭적인 경우가 보통이다[13,14] (즉, 순방향에서 지나는 라우터와 역방향에서 지나는 라우터가 다른 경우가 많다는 의미이다). 또한, 같은 경로를 패킷이 거쳐 간다 할지라도 네트워크의 상황에 따라 그 패킷이 겪는 큐잉 지연 시간(queuing delay) 의 정도도 다를 수 밖에 없다. 결과적으로 이러한 다양한 특성들이 RTT와 단방향 지연 시간의 차이를 크게 한다.

만약 순방향 지연 시간이 RTT의 1/2과 너무 큰 차이를 보인다면 RTT에 기반한 많은 프로토콜들이 네트워크의 자원을 잘 못 사용하게 될 것이다. 가령, 역방향이 혼잡하면 역방향의 지연 시간은 순방향보다 커지는데, RTT가 두 지연 시간의 합이기 때문에 RTT에 기반한 프로토콜들은 이러한 상황을 적절히 반영하지 못하고 사용 가능한 대역폭을 부정확하게 추정하게 될 것이다[8,12]. 또한, 이러한 경향은 순방향과 역방향 경로의 대역폭이나 지연 시간의 차이가 큰 비대칭 네트워크에서 더욱 두드러지게 나타날 것이다.

이 논문은 분석적으로 순방향과 역방향 지연시간을 각각 계산해낼 수 있는 방법을 제시하며, 이 방법을 비대칭 네트워크(asymmetric network)에서의 TCP에 적용하여 정확한 단방향 지연 시간을 이용하는 TCP의 초기 동작이 RTT를 이용하는 것보다 보다 효율적으로 네트워크의 자원을 이용함을 보인다. 본 논문의 나머지 부분은 다음과 같이 구성된다. 2절에서는 새로운 단방향 지연 시간의 고안에 동기가 된 전형적인 단방향 지연 시간의 형태를 보여준다. 3절에서는 본 논문에서 제시한 단방향 지연 시간 추정 기법을 설명하며, 4절에서 이를 실제 프로토콜 형태로 제시한다. 5절에서는 이 방법을 비대칭 네트워크에서 활용한 응용의 예를 제시하며, 6절에서 네트워크 트래픽의 관점에서 우리의 방법이 어느 정도의 부가적인 트래픽을 만들어 낼 수 있는 지 설명한다. 그리고 7절에서 시뮬레이션 결과와 그 이유를 분석하며, 다음 절에서 우리의 방법으로도 해결하지 못한 문제에 대해 논의한 후, 9절을 끝으로 본 논문을 마무리 짓는다.

2. 논문의 동기와 배경 지식

Motivation: 단방향 추정 기법을 소개하기 전에 먼저 정확한 순방향 지연 시간과 TCP 송신자가 측정한 RTT의 차이를 관찰해 볼 필요가 있다. 그림 1은 일반적인 대칭 네트워크 토폴로지에서 수행한 TCP 통신의 한 예이다.

이 그래프에서 X 축은 시뮬레이션 시간이며, Y 축은 순방향 지연 시간과 RTT/2를 표기했다. 순방향 지연 시간은 송신자와 수신자의 클럭을 동기화시킨 후, 송신자의 타임 스탬프와 수신자의 클럭의 차이로 구했다. 그림 1을 보면 이와 같이 구한 순방향 지연 시간과 RTT/2가 대칭 네트워크에서조차 큰 차이를 보임을 알 수 있다. 즉, RTT/2가 순방향 경로의 혼잡 정도를 정확하게 보여주지 못하고 있음을 알 수 있다. 그리고 이러한 경향은 RTT가 네트워크 상황의 지시자(indicator)로 사용될 때, 각 경로의 상황을 오판하여 결과적으로 프로토콜이 네트워크 자원(network resource)를 적절하게 활용하지 못할 가능성이 있음을 보여준다. 각 단방향 지연 시간의 변화를 정확하게 추적할 수 있다면 이러한 문제를 해결할 수 있음은 자명하다.

단방향 지연 시간을 정확하게 추정하기 위해 제안되었던 기존의 연구들은 크게 두 개의 접근 방식으로 분류될 수 있다. 첫 번째 부류는 송신자와 수신자의 클럭을 먼저 동기화 시킨 후, 타임 스탬프의 차이를 이용하는 것인데, 이러한 방법들은 분산 컴퓨팅 환경이 커질수록 적용하기 어려운 것으로 알려져 있다[1,16]. 또 다른 한 부류는 타임 스탬프에서 클럭 스큐(clock skew)를 추정해 내어 단방향 지연 시간을 계산하는 것이다. 이와

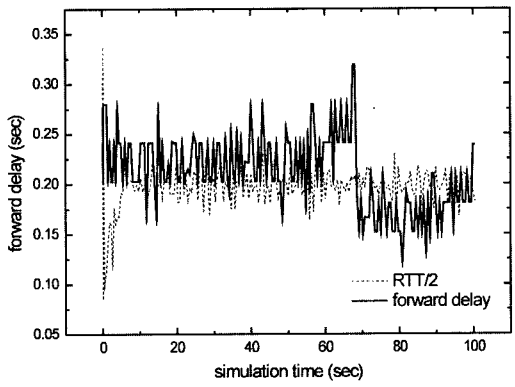


그림 1 순방향 지연 시간과 RTT/2

같은 접근 방식을 취한 대표적인 연구는 Paxson[1]과 Moon[2]이다.

Paxson의 연구는 두 노드 A와 B 사이의 측정된 단방향 지연 시간값 들에서 클럭 스큐를 추정해 내는 것을 핵심 골자로 하고 있다. 비록 알고리즘에서 휴리스틱한 팩터가 있기는 하지만 이 연구에서 제시된 알고리즘은 양 노드의 클럭이 동기화되어 있지 않다 해도 단방향 지연 시간을 상당히 정확하게 추정할 수 있다는 장점이 있다.

Moon의 연구는 Paxson의 연구와 본질적으로 같은 문제를 해결하고 있다. 차이가 있다면 측정값에서 클럭 스큐와 측정 오류를 추정해 내는데 선형 프로그래밍(linear programming) 기법을 이용하고 있으며, 그 추정 오차가 Paxson의 연구보다 작아졌다는 것이다.

두 연구의 공통된 가정은 두 노드 A와 B가 각각 자신의 클럭에 의해 이미 측정된 데이타를 확보하고 있어야 한다는 것이다. 이러한 제약은 네트워크의 성능을 측정하고 관리하는데는 큰 문제가 없으나 추정을 위해 많은 데이타가 확보되어야 하는 것은 물론, 이 측정값들이 별도의 처리를 거쳐야 하기 때문에 프로토콜과 같은 실시간 추정이 요구되는 곳에서는 이와 같은 접근 방식이 사용되기 어렵다.

본 연구와 기존 연구들과의 차이점을 정리하자면 다음과 같다.

- RTT를 단방향 지연 시간의 추정에 이용한 최초의 연구이다.
- 프로토콜과 같은 실시간 추정이 요구되는 곳에서 사용 가능한 연구이다.

부가적으로 본 연구에서 제시하는 새로운 추정 기법의 장점을 다음과 같이 정리할 수 있다.

- 송신자와 수신자 사이의 클럭 동기화를 가정하지 않는다.
- 네트워크의 트래픽 패턴이나 혼잡 여부 때문에 지연 시간이 동적으로 변화해도 순방향과 역방향 지연 시간의 변화를 정확하게 따라갈 수 있다.
- 충분히 짧은 시간 안에 지연 시간의 계산이 가능하기 때문에 네트워크의 상황을 반영하여 동작을 조절하는 프로토콜에도 이용 가능하다.
- 최대한 단순하게 설계되어 실제 구현하기 용이하다.

Basic clock terminology: 이 후의 논의에서 사용되는 클럭의 특징을 논의하기 위해 기본적인 용어를 정의하고 넘어갈 필요가 있다. Mill[6]은 이러한 클럭의 특징을 기술하기 위해 용어를 정리하였는데, 우리도 역시 다음과 같은 그의 용어를 따르기로 한다.

- 진행 속도(Frequency): 클럭이 진행되는 속도.
- 오프셋(Offset): 클럭에 의해 보고되는 시간과 국가 표준시와의 차이.
- 스큐(Skew): 클럭과 국가 표준시의 진행 속도 차이.

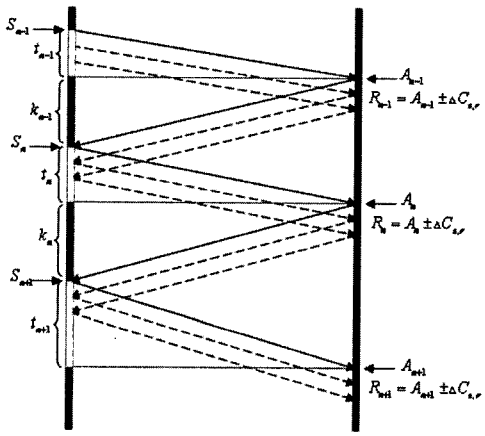


그림 2 전형적인 패킷 교환

3. 지연시간 유도

우리 방법은 지연 시간을 추정하기 위해 RTT당 하나의 패킷을 사용한다. 따라서, RTT동안 여러 개의 패킷이 전송된다면 혼동을 피하기 위해 그 중 첫번째 패킷에만 마크(mark)를 하고 마크된 패킷의 ACK를 추정에 이용한다. 그림 2는 전형적인 패킷과 ACK의 교환을 도식화한 것이다. 그림 2의 모델은 TCP의 일반적인 통신 과정에서 단방향 지연 시간과 송신자와 수신자가 각각 측정할 수 있는 RTT의 값사이에 연속적인 연결 고리가 존재함을 잘 보여준다. 이들의 관계는 다음과 같이 분석적으로 모델링될 수 있다.

이제 단방향 지연 시간 추정 기법의 유도에 사용될 클럭, 타임 스탬프, 그리고 지연 시간의 표기를 소개한다.

- C_s: 송신자의 클럭. 그림 2에서 왼쪽 노드(송신자)가 유지하고 있는 클럭을 의미한다.
- C_r: 수신자의 클럭. 그림 2에서 오른쪽 노드(수신자)가 유지하고 있는 클럭을 의미한다.
- S_n: C_s의 관점에서 본 송신자 단에서의 n번째 패킷 전송 시간. 그림 2에서 송신자가 보내는 패킷의 턴(turn)에 일련 번호를 부여할 때, C_s에 의해 나타나는 n번째 턴의 시작 시간을 의미한다.
- R_n: C_r의 관점에서 본 수신자 단에서의 n번째 패킷 도착 시간. 그림 2에서 수신자가 받는 패킷의 턴에 일련 번호를 부여할 때, C_r에 의해 나타나는 n번째 턴의 시작 시간을 의미한다.
- A_n: C_s의 관점에서 본 송신자 단에서의 n번째 패킷

도착 시간. 그림 2에서 수신자가 n번째 패킷을 받았을 때의 시간을 C_r이 아닌 C_s의 관점에서 확인한 값이다. 따라서, A_n과 R_n 사이에는 $\triangle C_{s,r}$만큼의 상대적 오프셋이 존재한다.

- t_n: C_s의 관점에서 본 n번째 패킷의 순방향 지연 시간, 즉 $A_n - S_n$. 그림 2의 송신자에서 클럭 C_s를 바탕으로 측정한 n번째 패킷 턴의 순방향 지연 시간을 의미한다.
- k_n: C_s의 관점에서 본 n번째 패킷의 역방향 지연 시간, 즉 $S_{n+1} - A_n$. 그림 2의 송신자에서 클럭 C_s를 바탕으로 측정한 n번째 패킷 턴의 역방향 지연 시간을 의미한다.
- $RTT(s,n)$: C_s의 관점에서 본 송신자 단에서의 n-1번째 패킷의 RTT. 그림 2의 송신자에서 클럭 C_s를 바탕으로 측정한 RTT를 의미한다.
- $RTT(r,n)$: C_r의 관점에서 본 수신자 단에서의 n-1번째 패킷의 RTT. 그림 2의 수신자에서 클럭 C_r를 바탕으로 측정한 RTT를 의미한다.
- $\triangle C_{s,r}$: C_r과 C_s의 상대적 오프셋.

단방향 지연 시간이 A_n에 기반을 두고 계산되며, 상대적 오프셋 $\triangle C_{s,r}$까지 A_n만 정확하게 계산할 수 있다면 알 수 있기 때문에 이 값을 정확하게 얻는 것은 매우 중요한 문제이다. 하지만 A_n을 수신자 단에서 알 수 있는 방법이 없으며, 이것이 바로 클럭 동기화 문제가 분산 환경에서 해결되기 어려운 이유이다. 따라서, 본 논문에서 제시하는 방법은 A_n의 정확한 값을 사용하는 대신 클럭 스큐에 상관없이 같은 시간의 길이를 이용하여 이 문제를 해결한다.

RTT의 의미를 보다 명확하게 정의하면서 우리의 해결 방법을 설명하도록 하겠다.

- 송신자가 측정한 RTT: $RTT(s,i)$

  송신자는 하나의 패킷을 전송하고 그에 대한 ACK가 도착하는 시간을 측정하여 전송 시간과 수신 시간의 차이로 RTT를 계산한다. 따라서 $RTT(r,n) = R_n - R_{n-1}$ $RTT(s,n+1) = t_n + k_n$으로 표기할 수 있으며, 이것은 송신자가 C_s을 이용하여 S_{n+1}에 측정한 RTT이다. 따라서, 다음과 같은 식을 얻을 수 있다.

$$RTT(s,n+1) = S_{n+1} - S_n = t_n + k_n \quad (1)$$

- 수신자가 측정한 RTT: $RTT(n,i)$

  수신자는바로 이전 턴(turn)에 보낸 ACK와 현재의 ACK의 시간 차이를 RTT로 측정한다. 따라서, $RTT(r,n) = R_n - R_{n-1}$으로 표기할 수 있으며, 이것은 수신자가 C_r을 이용하여 R_n에 측정한 RTT이다. 또한, $R_n = A_n \pm \triangle C_{s,r}$이고 $R_{n+1} = A_{n+1} \pm \triangle C_{s,r}$이기 때문에 $R_{n+1} - R_n = A_{n+1} - A_n$의 관계를 얻을 수 있다. 이 관계식으로부터 우리는 사용되는 수신자가 C_r을 이용하여 측정한 RTT를 이용해서 C_s의 관점에서의 순방향 지연시간을 얻는 것이 가능하다는 것을 알 수 있다 (인접한 RTT 구간에서는 상대적인 오프셋이 일정하다고 가정한다). 따라서, 우리는 다음과 같은 또 다른 관계식을 얻을 수 있다.

$$RTT(r,n) = R_n - R_{n-1} = t_n + k_{n-1} \quad (2)$$

송신자와 수신자가 각각 측정한 RTT 모두 t_n을 포함하고 있기 때문에 두 식(식 (1)과 (2))을 서로 빼면 다음과 같은 관계식을 얻을 수 있다.

$$RTT(s,n+1) - RTT(r,n) = k_n - k_{n-1} \quad (3)$$

이 관계식을 1에서 n일 때의 경우를 모두 더하면 다음과 같은 새로운 식을 얻을 수 있다.

$$\begin{aligned} &RTT(s,2) - RTT(r,1) = k_1 - k_0 \\ &+RTT(s,3) - RTT(r,2) = k_2 - k_1 \\ &+RTT(s,4) - RTT(r,3) = k_3 - k_2 \\ &\cdots\cdots\cdots \\ &+RTT(s,n+1) - RTT(r,n) = k_n - k_{n-1} \\ &\sum_{i=1}^{n} RTT(s,i+1) - \sum_{i=1}^{n} RTT(r,i) = k_n - k_0 \end{aligned} \quad (4)$$

n이 0일 때, 식 (1)은 $RTT(s,1) = t_0 + k_0$이기 대문에 식 (4)에서 k_n와 k_0를 각각 다음과 같이 얻을 수 있다.

$$k_n = RTT(s,n+1) - t_n,$$
$$k_0 = RTT(s,1) - t_0$$

이로부터 다음의 식을 얻는다.

$$\begin{aligned} k_n - k_0 &= \sum_{i=1}^{n} RTT(s,i+1) - \sum_{i=1}^{n} RTT(r,i) \\ &= RTT(s,n+1) - t_n - RTT(s,1) + t_0 \end{aligned} \quad (5)$$

따라서, 다음과 같이 순방향 지연 시간 t_n을 얻을 수 있다.

$$t_n = t_0 - \sum_{i=1}^{n} [RTT(s,i) - RTT(r,i)] \quad (6)$$

또한, 같은 방법으로 역방향 지연 시간 k_n을 얻을 수 있다.

$$k_n = -t_n + \sum_{i=1}^{n+1} RTT(s,i) - \sum_{i=1}^{n} RTT(r,i) \quad (7)$$

식 (6)과 (7)은 순방향과 역방향 지연 시간이 송신자가 측정한 RTT와 수신자가 측정한 RTT를 이용하여

계산될 수 있음을 의미한다. 또한, 이 식은 수신자가 RTT를 측정하여 돌려주기만 한다면 단방향 지연 시간의 변화를 정확하게 따라갈 수 있음을 의미하기도 한다.

4. 프로토콜의 설계와 구현

식 (6)을 구현하기 위해 우리는 세 개의 값을 계산할 필요가 있다. 그 첫 번째는 송신자가 측정한 RTT이며, 두 번째는 수신자가 측정한 RTT이다. 그리고 마지막이 순방향 지연시간의 초기값인 t_0이다. 이들 각각의 값을 TCP에서 어떻게 계산할 수 있는 지를 설명하고자 하며, 그 프로토콜은 그림 3에 기술되어 있다.

• $RTT(s,i)$: TCP의 모든 버전은 전송률(sending rate)이나 타임아웃값을 계산하기 위해 주기적으로 RTT를 측정한다. 따라서 송신자가 RTT를 측정하는 것은 쉬운 일이다.

• $RTT(r,i)$: 수신자에서 RTT를 측정하는 가장 직관적인 방법은 ACK의 간격을 측정하는 것이다. 식 (6)을 계산하기 위해서는 송신자가 $RTT(s,i)$와 $RTT(r,i)$를 모두 필요로 하기 때문에 수신자는 ACK 헤더(header)의 사용되지 않는 필드(field) 중 하나에 $RTT(r,i)$를 써서 알려줘야 한다.

• t_0: 식 (6)이 단방향 지연시간이 분석적으로 유도될 수 있음을 보여준다 해도 지연 시간 추정의 정확성은 순방향 지연 시간의 초기값 t_0에 의해 결정된다. 따라서 적절한 t_0를 얻어내는 방법에 대해 논의해볼 필요가 있다. TCP 세션이 처음 만들어질 때, 만약 이미 네트워크의 혼잡이 없었다면 t_0를 RTT/2로 시작하는 것은 추정 기법의 결과에서 큰 오차를 만들어내지 않을 것이다. 하지만 세션이 만들어질 때의 네트워크 상황을 미리 예측하는 것은 매우 어려운 일이다. 이러한 이유로 t_0의 오차 범위를 종단간 프로토콜이 수용할 수 있는 수준까지 줄일 수 있는 휴리스틱(heuristic) 방법이 필요하다. 먼저, t_0의 상위 경계 값(upper bound)를 결정하자.

```
Sender's protocol
On sending segment for RTT measurement:
        tcph.timestamp = current time;
        probe_sequence++;
On receiving new ACK:
        RTT_S_ = current time – ack.timestamp;
        if (ack.rtt_r_ <= 0)
                exit;
        RTT_R_ = ack.rtt_r_;
        if (probingPhase == true){
                calculate average d;
        }else{
                If(firstDataACK == true){
                        Make t0;
                Else{
                        fd_diff_ = fd_diff_ +
(fd_prev_rtt_s_ - ack.rtt_r_);
                        fd_prev_rtt_s_ = RTT_S_;
                }
                tn = t0 – fd_diff_;
        }
Receiver's protocol
On receiving TCP segment
        if (firstPacket == true){
                fd_prev_ack_ = current time;
                ack.rtt_r_ = -1;
        }else{
                ack.rtt_r_ = current time – fd_prev_ack_
                fd_prev_ack_ = current time;
        }
```

그림 3 단방향 지연 시간 추정 기법의 수도 코드

첫 순방향과 역방향 지연 시간의 차이를 d라고 정의하면 $(t_0 - k_0 = d)$ 다음과 같이 세 경우 마다 각각 상위 경계 값을 얻을 수 있다.

- case 1: $d>0$

이 경우 $t_0 > k_0$이기 때문에 $2t_0 > RTT(s,0) > 2k_0$가 성립한다. 양변에 t_1을 더한 식 $t_0 + RTT(s,0) > t_1 + 2k_0$에서 $t_1 + k_0 = RTT(r,0)$와 $t_1 = t_0 + RTT(s,1) - RTT(r,1)$의 조건을 이용하여 새로운 t_0의 범위를 다음과 같이 설정할 수 있다.

$$\frac{RTT(s,1) - RTT(r,1) - RTT(r,0)}{2} < t_0 < RTT(s,0) \tag{8}$$

- case 2: $d<0$

이 경우, $d>0$과 마찬가지 과정을 거쳐 다음과 같은 범위를 얻는다.

$$\frac{RTT(s,1) - RTT(r,1) - RTT(r,0)}{2} < t_0 < 0 \tag{9}$$

- case 3: $d=0$

이 경우, t_0와 k_0가 같기 때문에 초기값은 다음과 같이 설정할 수 있다.

$$t_0 = k_0 = \frac{RTT(s,0)}{2} \tag{10}$$

하지만 저자들이 아는 한에서는 아직 d의 부호(sign)을 알 수 있는 실용적인 방법이 없기 때문에 이 값을 예측하기 위해 간단한 휴리스틱을 이용한다.

이 휴리스틱 방법은 인터넷에서 단방향 지연시간의 특성인 큐잉 지연 시간의 변화가 짧은 순간에 급격하게 발생하지 않으며(steady) 전체 지연 시간 중 가장 큰 비중을 차지하는 것(dominant)이라는 성질을 이용하는 것이다. 따라서, d의 부호가 세션이 만들어질 때부터 측정된 몇 개의 샘플(sample)들의 평균과 크게 다르지 않을 것이라고 예측할 수 있다. 다시 말하면 순방향 지연

시간과 RTT의 비율이 α보다 크다면 t_0를 $\frac{RTT(s,1)-RTT(r,1)+RTT(r,0)+2RTT(s,0)}{4}$로 설정한다 (이 값은 $d>0$인 경우의 영역 평균(range average)에 해당한다). 반대로 비율이 β보다 작으면 t_0로 $\frac{RTT(s,1)-RTT(r,1)+RTT(r,0)}{4}$를 선택한다 (이 값은 $d<0$인 경우의 영역 평균에 해당한다). 그리고 만약 그 비율이 β보다 크고 α보다 작으면 t_0는 $\frac{RTT(s,0)}{2}$가 된다. 여기서 α와 β는 각각 0과, 그리고 $RTT(s,0)$의 중간(middle value)에 근사한 값이다. 우리는 이 프로토콜을 구현할 때, α와 β에 각각 0.75과 0.25을 사용했는데, 이 값은 RTT를 4등분하는 값이며, 다수의 시뮬레이션 결과 이 값이 대부분의 경우에 큰 오차를 보이지 않음을 확인하였다.

5. TCP에의 응용 예

이절에서는 앞서 소개한 단방향 추정 기법을 비대칭 환경(asymmetric environment)[1]에서의 TCP에 어떻게 적용할 수 있는 지를 소개한다. TCP는 비대칭 네트워크에서 심각한 성능 저하를 보이는데, 그 이유는 TCP가 ACK에 절대적으로 의존하여 동작하기 때문이다. 따라서 비 대칭성으로 인한 ACK의 도착 율(arrival rate)의 변화가 처리량을 크게 떨어뜨리게 되는 것이다. 이절에서 우리는 기존 연구에서 제안되었던 기존의 해결 방안보다 단방향 지연 시간을 활용한 방법이왜 더 효율적이고 실용적인지를 비교 설명할 것이다.

비대칭성에 대한 가장 직관적인 해결 방법은 ACK 흐름의 혼잡을 제어하거나 ACK 패킷의 수를 필터링(filtering)같은 방법을 이용하여 줄이는 것이다. Balakrishnan et al.[12]는 비대칭 네트워크에서 발생하는 TCP의 성능 저하 문제를 해결하기 위해 ACC(ACK Congestion Control)과 AF(ACK Filtering)을 제안하였다. 또한, delayed ACK 방법도 종단간 문맥(end-to-end context)에서 ACK의 수를 줄이는데 도움을 줄 수 있다. 이러한 ACK 제어 방법들은 ACK의 수를 줄여 역방향 병목 링크(reverse bottleneck link)의 혼잡 문제를 완화시켜준다 (즉, 하나의 ACK가 여러 데이타에 대해 수신 확인을 하게 된다). 하지만 이러한 방법들은 송신자가 패킷을 버스티(bursty)하게 전송하도록 만들고 TCP의 초기 단계에서 cwnd(혼잡 제어 윈도우)가 너무 늦게 증가하도록 만들며, 중복된 ACK에 의해 활성화되는 fast retransmission 메커니즘의 효율성을 떨어뜨린다는 문제가 있다[12].

SA(Sender Adaptation)[12]의 핵심 아이디어는 maxburst의 크기를 결정하여 cwnd가 maxburst 보다 크다 할지라도 maxburst 만큼의 데이타만을 보내게 억제하는 것이다. 또한, 보낼 데이타의 크기를 타이머를 두어 잘게 쪼갠다 ($\frac{cwnd}{RTT}$가 타이머가 완료될 때마다 보내는 양이다). 하지만 이 방법에서 TCP는 여전히 ACK가 도착할 때에만 cwnd를 증가시킨다. 그 결과 이 방법은 순방향 경로의 대역폭을 적절하게 사용하지 못하는 문제를 여전히 안고 있다. 만약 순방향 지연 시간을 알고 있다면 TCP는 ($2\times forwarddelay$)마다 cwnd를 증가시킬 수 있다 (적어도 TCP의 초기 단계에서라도[2]). 이러한 접근 방법은 1) cwnd를 빨리 증가시킬수 있으며 2) 보다 네트워크자원의 활용(utilization)을 효율적으로 할 수있다.

이러한 기본 개념은 이미 Raisinghani et al.[8]에 의해 제안된 바 있다. 하지만 그들은 MAF라 불리는 고정된 휴리스틱 변수를 이용하여 네트워크의 상황에 따라 다른 ($2\times forwarddelay$)를 추정하기 때문에 실제 환경에서 적용되기는 어려웠다. 즉, 시뮬레이션을 통해 접근 방법의 가능성은 보여주었지만 실제로 적용되기는 어려웠던 연구였다고 평가할 수 있다.

비대칭 네트워크에서 TCP의 성능 문제는 주로 부정확한 RTT로부터 기인한다. RTT는 순방향과 역방향 지연의합이기 때문에 데이타 흐름과 ACK 흐름 사이에는 도착 시간의 차이가 존재한다. 따라서 RTT에 기반한 TCP 적용 방법들은 비대칭 환경에서 잘못 동작할 수가 있다. 만약 합당한 수준으로 단방향 지연 시간을 추정할 수 있다면 TCP는 ACK의 도착율 변화에 따른 영향을 상당히 줄일 수 있을 것이다.

본 논문에서 제시하는 해결 방법의 핵심 아이디어는 TCP의 시작 단계(slow start)에서 ACK의 도착 여부에 상관없이 대역폭을 추정하는 것이다. 환언하면 TCP가 초기 단계에 ACK를 받지 못했더라도 ($2\times forwarddelay$)마다 cwnd를 증가시켜서 가용한 대역폭의 크기를 추정한다는 것이다 (그 결과 TCP는 대역폭을 전보다 빨리 찾을 수 있게 된다). 이후 혼잡 회피 단계(congestion avoidance phase)부터는 TCP의 원 메커니즘과 동일하게 ACK를 받았을 때에만 데이타를 전송한다. 이와 같이 동작하면 단방향 지연 시간의 비대칭이 심각할 때에만 메커니즘이 활성화되게 된다. 즉, FI(Fast Increase)는 송신자가 타이머(cwnd timer)가 만료되기 전에 계속 ACK를 받으면 활성화되지 않는다는 것을 의미한다. 이

1) TCP 성능의 관점에서 비대칭 네트워크란 그 처리량(throughput)이 순방향의 링크와 트래픽 특성만의 함수(function)이 아닌 역방향의 특성까지 포함한 함수인 네트워크를 말한다.

2) 초기 단계에서 TCP는 네트워크의 가용 대역폭을 추정한다.

프로토콜의 수도 코드(pseudo code)는 그림 4에 기술되어 있다. 7절에서는 다수의 시뮬레이션을 통해 이러한 TCP FI가 기존의 접근 방식보다 실용적이고 보다 간단하다는 사실을 보여준다.

```
ON receiving new ACK:
        IF((slow_start == true) &&
                (fd_estimation_available == true))
                fd = fd_estimation();
                IF(cwnd_timer_run == true)
                        Cwnd_timer_cancel();
                        Cwnd++;
                END
                cwnd_timer->timeout = fd*2;
        ELSE
                Normal_TCP_action();
        END

ON expiring cwnd_timer:
        Cwnd++;
```

그림 4 TCP Fast Increase 의 수도 코드

6. 단방향 지연 시간 추정 기법이 네트워크에 주는 영향 분석

이절에서는 단방향 지연 시간 추정 기법의 과부하(overhead), 즉 $RTT(s,i)$, $RTT(r,i)$ 그리고 $RTT(r,i)$의 결과를 쓸 ACKJ 헤더의 작은 공간 등의 자원을 확보하기 위해 부가적으로 발생하는 패킷의 양을 분석한다. 과부하의 정도는 추정 기법이 TCP와 함께 사용되는 경우와 독립적으로 순수하게 추정용 프로토콜로 구현될 경우에 따라 각각 다르다.

6.1 TCP와 함께 사용되는 경우

TCP는 RTO를 결정하기 위해 RTT를 주기적으로측정하고 있기 때문에 추정 기법에서 $RTT(s,i)$를 얻기 위해 부가적인 패킷을 보낼 필요는 없다. 또한 TCP 수신자도 $RTT(r,i)$를 얻기 위해 부가적인 패킷을 요구하지 않는데, 이것은 $RTT(r,i)$이 ACK들의 간격으로 계산될 수 있기 때문이다. 한가지 유의 해야 할 점은 TCP 송신자가 슬라이딩 윈도우(sliding window) 메커니즘에 의해 한 턴에 여러 개의 데이타 패킷을 보낸다는 것이다. 이것은 ACK의 간격을 계산하는데 혼란을 줄 수 있다. 이러한 모호함을 제거하는 가장 직관적인 방법은 TCP 송신자가 $RTT(s,i)$의 측정 대상 패킷의 헤더에 간단한 마크를 해서 보내는 것이다. 이것을 TCP 수신자가 받게 되면 해당 턴에 여러 개의 데이타 패킷을 받았다 할지라도 마크가 있는 패킷의 ACK만을 계산 대상으로 잡으면 모호함을 제거할 수 있다.

따라서 본 논문에서 제시한 추정 기법이 TCP와 함께 사용될 경우에는 네트워크에 아무런 부하도 주지 않음을 알 수 있다. 이것은 수신자가 delayed ACK를 사용하더라도 여전히 유효하다. Delayed ACK는 보통 2개의 데이타 패킷 당 하나의 ACK를 보내는 것인데, 일반적으로 최소 cwnd는 2 이상이기 때문이다. 이러한 이유로어떠한 경우에도 $RTT(r,i)$을 얻기 위해 부가적인 패킷이 사용될 필요가 없다는 사실을 알 수 있다. 이로부터 판단해 보건 데, 우리의 추정 기법이 TCP와 사용될 때에는 측정의 모호함을 제거하기 위해 사용되는 마크를 위한 필드와 $RTT(r,i)$의 결과값을 돌려주기 위해 필요한 필드의 약간의 공간적인 부하만이 요구된다.

6.2 ACK 없이 사용되는 경우

이 경우 양단에서 측정한 RTT를 얻기 위해 간단한 핑-퐁(ping-pong) 프로토콜이 필요하다. 따라서 핑-퐁 프로토콜에서 보내는 모든 패킷이 과부하라고 할 수 있지만 필요로 하는 것은 단지 데이타 없는 제어 패킷(control packet)이기 때문에 부하가 크다고 할 수는 없다. 또한, 보내는 패킷과 그에 대한 ACK 패킷의 크기가 동일하기 때문에 그만큼 추정 기법의 오차도 줄어들 것이다.

패킷의 크기를 40 bytes라고 가정하고 지연을 측정한 시간을 t라고 하면 다음과 같이 이 기법이 네트워크에 주는 부하를 계산할수 있다.

$$OverheadTraffic = \frac{1}{average(RTT)} \times 40bytes \times 2 \tag{11}$$

이식을 보면 평균 RTT가 200ms인 네트워크에서 100초 동안 측정을 한다 할지라도 네트워크로 투입되는 부하는 40 Kbytes 남짓에 불과하다.

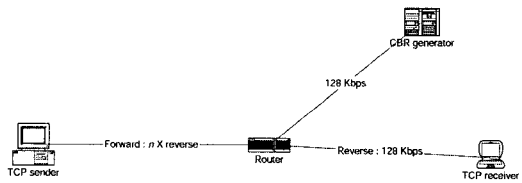


그림 5 ns-2 시뮬레이션을 위한 네트워크 토폴로지

7. 시뮬레이션과 분석

이절에서는 두 종류의 시뮬레이션 결과를 보고한다. 첫 번째 시뮬레이션은 본 논문에서 소개된 추정 기법의 결과값의 평균과 표준 편차를 계산하는 것이다. 이 시뮬레이션의 목적은 추정 기법이 네트워크의 변화 정도에 상관없이 순방향과 역방향 지연 시간의 변화를정확하게 따라가고 있음을 보여주는 것이다. 시뮬레이션의 장점을 이용하여 우리는 모든 노드의 클럭을 동기화시키고 실제 지연 시간(true one-way delay)을 측정할 수 있었으며, 이 값을 추정 값과 비교할 수 있었다. 추정 기법

은 3절에서 기술된 프로토콜의 형태로 구현되었다.

두 번째 시뮬레이션은 추정 기법을 응용한 TCP의 성능을 평가하는 것이다. 우리는 TCP Reno에 단방향 추정 기법을 적용하였으며, ns2 시뮬레이터[15]에서 그 성능을 평가하였다. 두 번째 시뮬레이션을 통해 우리는 순방향 지연시간을 사용하는 것이 TCP가 비대칭 네트워크에서 보인 문제를 완화시키는데 효과적임을 보여줄 것이다.

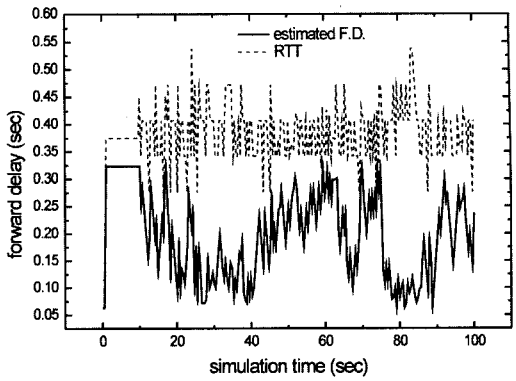
7.1 평균과 표준 편차에 기반한 분석

비대칭 네트워크를 위해 실험에서 사용된 네트워크 토폴로지는 그림 5에서 볼 수 있다. 역방향 채널의 대역폭은 128 Kbps이며 순방향의 대역폭은 $n \times 128Kbps$이다 (n은 0에서 10 사이의 값이며, 이를 비대칭 정도라 정의한다). 비대칭 정도는 순방향 채널의 대역폭이 역방향에 비해 얼마나 큰 지를 나타낸다. 가령, ADSL 링크가 8Mbps의 순방향 대역폭과 1Mbps의 역방향 대역폭을 갖고 있다면 이 링크의 비대칭 정도는 8이다. 마찬가지로 순방향 대역폭이 1Mbps이고 역방향 대역폭이 5Mbps라면 비대칭 정도는 0.2이다.

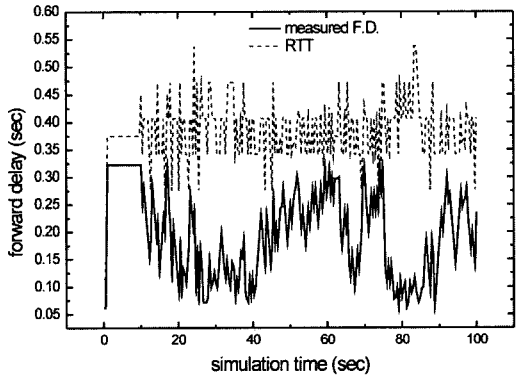
본 논문의 시뮬레이션에서 모든 링크의 전파 시간 (propagation delay)는 50ms이며 사용된 세그먼트의 크기는 1460 bytes이다. 또한, 라우터(router) 큐의 크기는 50 packets 이며 FIFO(First-In-First-Out)을 큐잉 정책으로 사용한다. 그리고 CBR(Constant Bit Rate) 응용이 역방향 채널의 혼잡을 만들어내기 위해 사용되었다.

그림 6은 비대칭 정도가 8이고 역방향 채널에 100Kbps의 트래픽이 있는 경우, 추정된 순방향 지연 시간과 측정된 순방향 지연 시간 (클럭을 동기화시키고), 그리고 RTT의 변화를 도식화한 것이다. 그림 6(a)는 RTT와 추정된 순방향 지연 시간의 결과를 보여준다. 그림 6의 두 그래프는 RTT가 네트워크의 상태를 잘 반영하지 못함을 보여준다. 또한, 두 그래프를 비교해 볼 때, 비록 절대값은 차이가 있을지라도 추정 기법에 의해 계산된 값과 실제 측정된 값이 거의 같은 형태를 띰을 알 수있다. 즉, 본 논문에서 제시된 추정 기법이 비대칭 네트워크 환경에서도 단방향 지연 시간을 잘 따라가고 있음을 확인할 수 있다. 또한, 이러한 경향은 비대칭 정도에 상관없이 일관되게 유지된다.

표 1, 2, 3은 비대칭 정도는 0.1에서 10까지 변화시켜 가면서 얻은 추정 기법의 결과와 측정 결과의 평균과 표준 편차를 정리한것이다(그림 7, 8, 9는 그 중 비대칭 정도 1에서 10까지의 경우를선별하여 도식화한 것이다). 또한, 지연 시간의 변화를 더 크게 만들기 위해 역방향에 부가적인 CBR 트래픽을 0에서 100Kbps까지 변화시켜 가면 추가하였다. 이러한 설정은 비대칭 정도와 네트워크 상태의 변화 여부에 상관없이 본 논문에서 제시된



(a) 추정된 순방향 지연 시간과 RTT



(b) 측정된 순방향 지연 시간과 RTT

그림 6 순방향 지연시간과 RTT의 비교

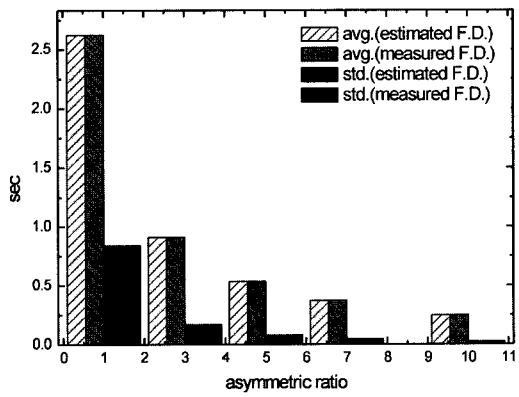


그림 7 역방향에 트래픽이 없을 때, 순방향 지연 시간의 평균과 표준편차

추정 기법이 잘 동작함을 보여주는데 유용하다. 평균과 표준 편차를 관찰해 보면 추정값과 측정값이 매우 근사하고 있음을 알 수 있다. 게다가 비록 평균값은 약간의 차이를 보이는 경우가 있지만 (기껏해야 0.002 sec 이다) 표준 편차는 두 값이 거의 차이가 없음을 알 수 있

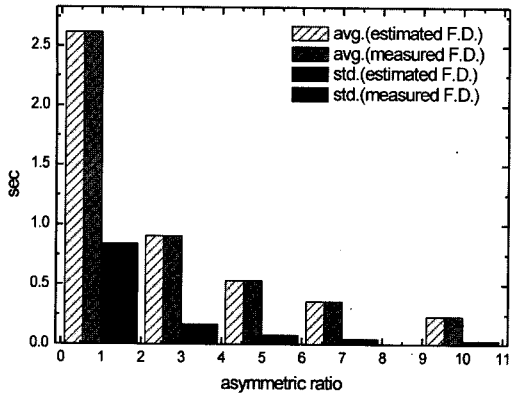


그림 8 역방향에 30Kbps의 트래픽이 있을 때, 순방향 지연의 평균과 표준편차

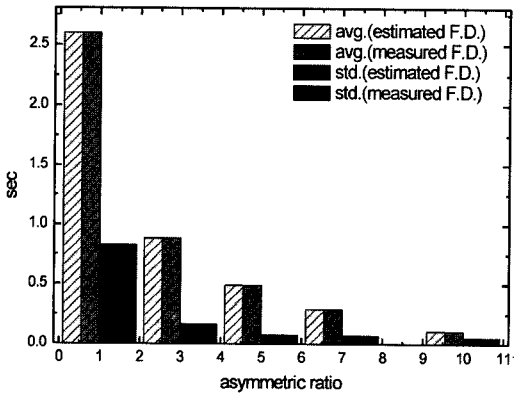


그림 9 역방향에 100Kbps의 트래픽이 있을 때, 순방향 지연의 평균과 표준편차

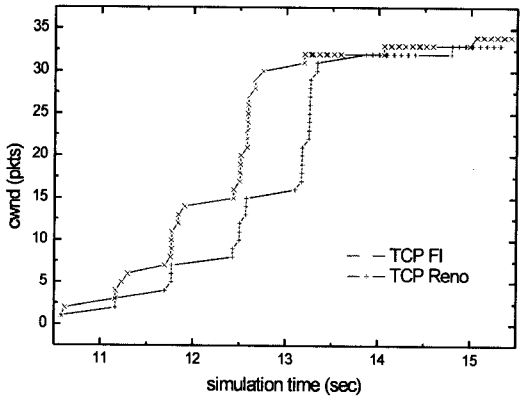


그림 10 TCP Reno와 FI의 초기 단계의 동작 모습

다. 이것은 추정 기법이 변화하는 환경에서도 단방향 지연 시간을 매우 정확하게 따라가고 있음을 보여준다.

측정값과 추정값 사이의 작은 차이는 전적으로 초기

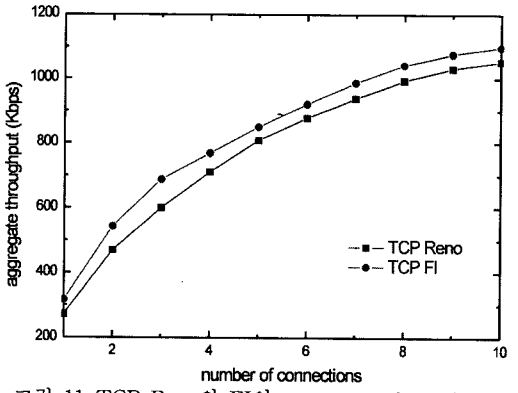


그림 11 TCP Reno와 FI의 aggregate throughput

값인 t_0에 의존하고 있다. 실험 결과, 최선의 경우에도 이 초기값의 영향이 완전히 없어지진 않았다. 하지만 결과에서 볼 수 있듯이 그 차이가 매우 미비하기 때문에 초기값을 결정하는 우리의 메커니즘도 수용할 만 하다는 사실을 확인할 수 있다. 실제로 순방향 지연시간의 변화 정도는 t_0의 차이보다는 변화하는 네트워크의 상황에 훨씬 더 큰영향을 받는다.

7.2 추정 기법의 응용을 위한 시뮬레이션 결과

평균을 이용한 분석에서 사용된 같은 토폴로지가 TCP의 성능 평가를 위한 시뮬레이션에도 사용된다. 순방향 경로의 전파 지연 시간을 제외하고는 세그먼트 크기나 라우터 큐 크기 등의 다른 모든 환경 변수는 동일하다. 역방향 채널의 링크 지연 시간을 조절하는 이유는 ACK의 도착율(arrival rate)을 보다 쉽게 제어하기 위한 것이다.

그림 10은 우리의 알고리즘이 TCP 초기 단계에서 보이는 동작을 보여준다 (TCP FI). 초기 단계에 초점을 두기 위해 시뮬레이션은 5초만 수행되었으며, 이 결과는 TCP FI가 ACK의 도착율에 상관없이 Reno 보다 빨리 cwnd를 증가시킴을 보여준다. 그 결과, 그림 11에서 확인할 수 있듯이 TCP FI가 Reno보다 약 17%의 처리량(throughput)을 향상시켜주었음을 알 수 있다. 또한, TCP 세션의 수가 증가할수록 이러한 처리량 향상의 경향도 일관되게 증가하였다. 이와 같은 결과로 판단 하건데, 평균적인 처리량의 관점에서 TCP FI가 처리량을 향상시킴은 분명하다고 판단할 수 있다.

8. 그 밖의 논의 사항

지금까지의 결과를 통해, 본 논문에서 제시한 추정 기법이 네트워크의 변화를 매우 잘 따라갈 수 있다는 사실을 확인할 수 있었지만 추정된 단방향 지연 시간이 고정된 순방향 지연 시간의 초기값에 영향을 받기 때문

에 여전히 측정값과의 오차가 존재한다. 시뮬레이션 결과, 표준 편차는 거의 같았지만 평균값에는 약간의 차이가 있음을 확인할 수 있었다. 본 논문에서 초기값을 결정하는 휴리스틱 방법을 제안하였고, 이 방법을 적용한 결과 오차의 범위를 많이 줄일 수 있었다. 또한 변화하는 다양한 네트워크 상황에 무관하게 이 방법이 수용할 수 있을 수준의 오차만을 발생시킴도 표 1,2,3으로부터 확인할 수 있었다. 하지만 t_0와 실제 초기 순방향 지연 시간의 차이는 여전히 존재하기 때문에, 오차의 범위는 세션이 만들어지기 전 양 방향 링크의 네트워크 혼잡 정도가 극심할 경우, 초기값을 정하는 휴리스틱 방법이 큰 오차를 낼 가능성은 충분히 있다. 따라서, 보다 정확한 결과를 얻기 위해 t_0를 결정하거나 동적으로 조절할 수 있는 효율적인 방법이 요구된다.

TCP 송신자의 타임아웃 값이 순방향 지연 시간에 기반하여 결정될 수도 있을 것이다. 현재의 TCP 구현은 RTT에 기반한 타임아웃 값을 이용하는데, 무선 네트워크를 포함한 몇몇 환경에서는 이러한 타임아웃 값이 적절하지 않게 설정되는 경우가 많다. 순방향과 양방향의 상황을 모두 알 수 있다면 타임아웃 값도 변화하는 네트워크의 상황에 근거하여 보다 능동적으로 결정할 수 있을 것이다.

9. 결 론

본 논문에서는 단방향 지연 시간을 분석적으로 유도할 수 있는 새로운 방법을 제안하였다. 단방향 패킷 전송 시간을 측정하고자 했던 기존의 연구들은 주로 클럭 동기화 문제에 초점을 맞추었지만 저자들이 아는 한 아직 실용적이고 확실한 방법이 제시되진 않았다. 따라서 우리는 기존의 연구들과 다른 접근 방법을 선택하였는데, 단방향 지연 시간의 절대값을 정확하게 계산하기 보다는 추정 값을 절대값에 근사 시키되 변화만은 정확하게 따라갈 수 있도록 하였다. 그 결과, 송신자와 수신자가 측정한 RTT의 차이가 약간의 시간 간격을 두고 변화하는 네트워크 상황을 정확하게 그리고 있음을 발견하였고, 이러한 개념 하에 새로운 지연 시간 추정 기법을 설계하였다.

시뮬레이션을 통해 이러한 단방향 지연 시간 추정 기법이 네트워크의 상황을 매우 잘 반영함을 확인할 수 있었으며, 이를 TCP Reno에 쉽게 구현할 수 있었다. 또한, 이 방법의 정확성을 검증하고 비대칭 환경에서의 TCP 문제를 완화시킬 수 있는 응용 예도 있음을 보였다.

참 고 문 헌

[1] V. Paxson: On Calibrating Measurements of Packet Transit Times, In Proc SIGMETRICS 1998, June 1998.

[2] S. Moon, P. Skelly, and D. Towsley: Estimation and Removal of Clock Skew from Network Delay Measurements, In Proc. INFOCOM 1999, March 1999.

[3] K. Anagnostakis, M. Greenwald, and R. Ryger: cing: Measuring Network Internal Delays using only Existing Infrastructure, In Proc. INFOCOM 2003, April 2004.

[4] D. Mills: Improved Algorithms for Synchronizing Computer Network Clocks, IEEE/ACM transactions on Networking, 3(3), June 1995.

[5] D. Mills: Network Time Protocol (Version 3): Specification, Implementation and Analysis, RFC 1305, Network Information Center, SRI International, Menlo Park, CA, March, 1992.

[6] D. Mills: Modelingand Analysis of Computer Network Clocks, Technical Report 92-5-2, Electrical Engineering Department, University of Delaware, May 1992.

[7] J. Postel: Transmission Control Protocol, RFC 793, September 1981.

[8] V. Rasinhani, A. Patil, and S. Iyer: Mild Aggregation: A new approach for improving TCP Performance in Asymmetric Networks, In Proc. AMOC 2000, October 2000.

[9] I. Ming-Chit, D. Jinsong, and W. Wang: Improving TCP Performance Over Asymmetric Networks, ACM Computer Communication Review, Volume 30, Issue 3, July 2000.

[10] M. Allman, S. Dawkins, D. Glover, J. Griner, D. Tran, T. Henderson, J. Heidemann, J. Touch, H. Kruse, S. Ostermann, K. Scott, and J. Semke: Ongoing TCP Research Related to Satellites, RFC 2760, February 2000.

[11] T. Henderson and R. Katz: Transport Protocol for Internet-Compatible Satellite Networks, IEEE Journal on Selected Areas in Communications, Vol 17, No. 2, February 1999.

[12] H. Balakrishnan, V. Padmanabhan, and R. Katz: The Effects of Asymmetry on TCP Performance, In Proc. Mobicom 1997, September 1997.

[13] M. Allman and V. Paxson: On Estimating End-to-End Network Path Properties, In Proc. SIGCOMM 1999, September 1999.

[14] V. Paxson: End-to-End Routing Behavior in the Internet, In Proc. SIGCOMM 1996, August 1996.

[15] ns2 Network Simulator version 2.26, http://www.-isi.edu/nsnam/ns, 2003

[16] A. Tanenbaum and M. Steen: Distributed Systems - Principles and Paradigms, Prentice Hall, 2002.

부 록

표 1 역방향 링크에 트래픽이 없을 경우, 추정값과 측정값의 비교

without background traffic

	estimated fd		measured fd	
	average	std.	avg.	std.
0.1	7.617188	1.781096	7.564688	1.781096
0.3	7.075357	3.982878	7.07244	3.982878
0.5	4.74325	2.145377	4.742	2.145377
0.7	3.596594	1.36899	3.597129	1.368989
1	2.623649	0.838992	2.623649	0.838992
3	0.911326	0.167597	0.910493	0.167597
5	0.53564	0.074833	0.53464	0.074833
7	0.370909	0.042493	0.36989	0.042493
10	0.245947	0.021901	0.244822	0.021901

표 2 역방향 링크에 30Kbps 트래픽이 있을 경우, 추정값과 측정값의 비교

with background traffic(30Kbps)

	estimated fd		measured fd	
	average	std.	avg.	std.
0.1	7.530813	1.79747	7.530778	1.79747
0.3	7.070144	3.981231	7.073061	3.981231
0.5	4.736642	2.142428	4.737892	2.142428
0.7	3.591718	1.366789	3.592253	1.366788
1	2.619188	0.837368	2.619188	0.837368
3	0.902363	0.166581	0.90153	0.166581
5	0.529321	0.075159	0.528321	0.075159
7	0.357442	0.044899	0.35637	0.044899
10	0.23584	0.028011	0.234715	0.028011

표 3 역방향링크에 100Kbps 트래픽이 있을 경우, 추정값과 측정값의 비교

with background traffic(100Kbps)

	estimated fd		measured fd	
	average	std.	avg.	std.
0.1	7.526613	1.799795	7.526255	1.799795
0.3	7.045031	3.963171	7.047947	3.963171
0.5	4.711177	2.12814	4.712427	2.12814
0.7	3.575713	1.359463	3.576248	1.359463
1	2.598705	0.829651	2.598705	0.829651
3	0.881034	0.163178	0.880201	0.163178
5	0.489119	0.076359	0.488119	0.076359
7	0.292937	0.071811	0.291866	0.07181
10	0.107206	0.054606	0.106081	0.054606

최 진 희

2000년 고려대학교 컴퓨터학과(이학사) 2002년 고려대학교 컴퓨터학과(전산학 석사). 2002년~현재 고려대학교 컴퓨터학과 박사 과정. 관심 분야는 커널 네트워킹, 프로토콜 설계 및 구현임

유 혁

1982년 서울대학교 전자공학과(공학사) 1984년 서울대학교 전자공학과(공학 석사). 1986년 University of Michigan(전산학 석사). 1990년 University of Michigan(전산학 박사). 1990년~1995년 Sun Microsystems 연구원. 1995년~현재 고려대학교 컴퓨터학과 교수. 관심 분야는 운영체제, 멀티미디어, 네트워크임