종단간 순방향/역방향 전송 지연 측정

황 순 한† · 김 은 기††

요 약

인터넷 망의 종단간 혼잡상태를 유추할 수 있는 가장 일반적인 방법은 RTT (Round Trip Time)값을 측정하는 것이며, 이 방법으로 망의 혼잡 정도를 측정할 수 있다. 그러나 RTT는 패킷의 왕복 시간만을 측정하기 때문에 패킷의 송수신시 순방향과 역방향에서 어느 정도의 혼잡과 전송지연이 발생하였는가는 알 수가 없다. 본 논문에서는 패킷이 전송될 때 순방향/역방향 전송 지연을 계산하여, 망의 혼잡 상태를 정확하게 유추할 수 있는 새로운 알고리즘을 제시한다. 본 알고리즘에서는 여러 RTT 값들 중에서 가장 작은 RTT 값을 기준으로 하여 기준이 되는 순방향/역방향 전송 시간을 결정하고, 이 값과 각 패킷이 전송될 때 측정된 전송 시간을 비교하여 순방향/역방향 전송 지연 시간을 계산한다. 본 연구에서는 NS-2에서의 시뮬레이션과 실제 네트워크 상에의 측정을 통하여 제안된 방법의 올바른 동작을 확인하였다.

키워드 : 순방향, 역방향, 전송지연, 혼잡

Measurement of End-to-End Forward/Backward Delay Variation

Soon-Han Hwang† · Eun-Gi Kim††

ABSTRACT

The measurement of RTT (Round Trip Time) can be used for the analysis of Internet congestion. However, simple measuring of RTT which measures only turn around time of a packet can not infer a packet forward/backward delay variation. In this thesis, we present a new algorithm which can be used for the estimation of forward/backward delay variation of packets. These delay variations are implication of network congestion state. In this algorithm, the reference forward/backward delay can be determined based on the minimum RTT value. The delay variation of each packet can be calculated by comparing reference delay with the packet delay. We verified our proposed algorithm by NS-2 simulation and delay measuring in a real network.

Key Words : RTT, Forward, Backward, Transfer Delay, Congestion

1. 서 론

인터넷의 발달로 인하여 사용자들에 의해 교환되는 데이터의 양이 점점 증가함에 따라서 인터넷 망 내의 트래픽(Traffic)이 지수적으로 증가하고 있는 추세이다. 패킷 교환망에서 패킷의 급속한 증가는 망을 혼잡 상태로 만들 수 있으므로, 데이터를 전송하는 노드에서는 인터넷 망의 혼잡 상태에 관한 정확한 정보를 알아내어 적절한 동작을 취하는 것이 필요하다[1].

망의 혼잡상태를 측정하기 위한 가장 일반적인 방법은 RTT(Round Trip Time) 측정 방법이며, 송신 노드에서는 패킷의 발신지로부터 목적지까지의 왕복 시간을 측정하여 패킷이 전송되는 경로의 혼잡 정도를 측정할 수 있다[1], [2].

그러나 기존의 RTT 측정 방법은 망 내에 혼잡이 발생했을 때 패킷의 왕복 시간만이 측정되므로 순방향(Forward) 경로에서 혼잡이 발생했는지 아니면 역방향(Backward) 경로에서 혼잡이 발생했는지의 여부는 알 수 없는 단점을 갖는다.

본 논문에서는 왕복 경로의 혼잡도 만을 측정할 수 있는 기존 RTT 측정의 단점을 보완하기 위해 기존의 RTT 측정 방법을 수정하여 패킷 송수신시 순방향과 역방향 경로에서 어느 정도의 혼잡이 발생하였는가를 측정할 수 있는 방법을 제시하고, NS-2를 이용한 시뮬레이션과 실제 네트워크 상에서의 측정을 통하여 제안된 방법의 올바른 동작을 확인하였다.

2. 종단간 순방향/역방향 전송 지연 측정 알고리즘

본 장에서는 논문에서 제시하고 있는 종단간 순방향/역방향 전송 지연 측정 방법에 대하여 기술한다. 1절에서는 종단간 순방향/역방향 전송 지연 측정에 관한 알고리즘을 간단히 설명하고, 2절에서는 알고리즘의 동작 방법을 상세히 설명한다. 끝으로 3절에서는 실제 망 환경에서 본 알고리즘

※ 이 논문은 2004년도 한발대학교 교내 학술연구비 지원을 받았음
† 준 회 원 : 국립한발대학교 정보통신 전문대학원 공학석사
†† 정 회 원 : 국립한발대학교 정보통신 · 컴퓨터공학부 부교수
논문접수 : 2004년 10월 11일, 심사완료 : 2005년 2월 11일

의 적용에 대하여 설명한다.

2.1 개요

각 호스트는 타임스탬프 카운터(TSC, Time Stamp Counter)에 의하여 표시되는 타임스탬프 값을 갖고 있으며, 이 값은 호스트의 현재 시간을 저장하는데 사용된다. TSC 값은 UTC(Coordinated Universal Time) 시간으로부터 경과된 밀리-초(milli-second) 단위의 시간으로, 송신 노드는 이 값을 이용하여 RTT 값을 측정할 수 있다[2], [5].

만약, 인터넷에 연결된 모든 호스트들의 TSC 값이 정확하게 동기화 되어 있다면 RTT 측정 만으로, 패킷의 순방향/역방향 전송 시간을 측정하는 것이 가능하다. 예를 들면, 송신측에서 어떤 패킷을 전송한 시각의 TSC 값이 10000, 수신측에서 이에 대한 응답을 전송한 시간이 10010 이고, 응답이 송신측에 도착한 시간이 10030이면 순방향 전송 시간은 10ms, 역방향 전송 시간은 20ms, 전체 RTT 시간은 30ms로 쉽게 계산될 수 있다. 그러나, 인터넷에 연결된 호스트들의 TSC 값은 동기화 되어 있지 않으므로, RTT 측정만으로 순방향 전송 시간과 역방향 전송 시간을 알아내는 것이 불가능하다[5].

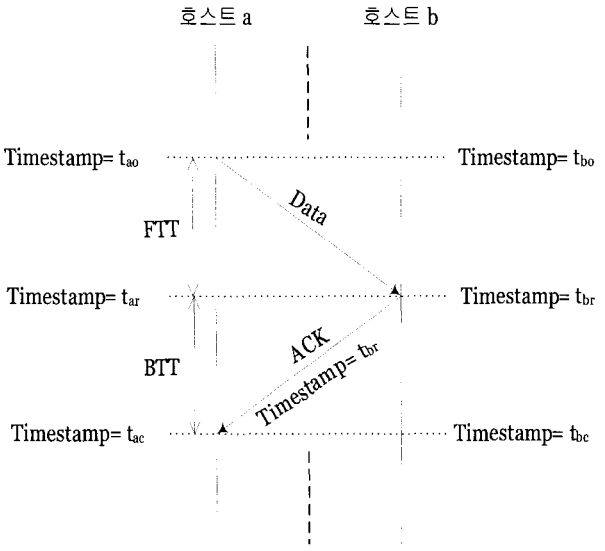
본 연구에서는 기존 RTT 측정 방법의 문제점을 해결하기 위하여 RTT 측정값이 가장 작을 때 - 이때의 순방향과 역방향 전송 시간도 가장 작다. - 를 기준으로 RTT 값의 증가 원인이 어느 경로에 있는지 측정하여 망에 혼잡이 발생했을 때 혼잡 원인이 되는 경로를 파악하기 위한 알고리즘 제안한다.

본 연구에서 제안한 알고리즘은 다수의 RTT를 측정하여, 가장 작은 RTT의 경우 왕복 전송 경로에 가장 혼잡이 없고, 이때의 순방향과 역방향 전송 시간이 동일 하다는 기본 가정으로부터 시작한다. 이러한 가정에서, 가장 작은 RTT 값을 갖는 경우로부터 계산된 순방향/역방향 전송 시간을 사용하여 호스트간 TSC 값의 차를 계산한다[2]. 그리고 계산된 호스트간 TSC 차이 값을 이용하여 현재 응답 패킷이 RTT 측정 값이 최소일 때 계산한 순방향/역방향 전송시간보다 얼마나 증감하였는가를 계산할 수 있게 된다.

본 연구의 기본 가정에서 사용된 최소 순방향/역방향 전송 시간이 실제 순방향/역방향 전송 시간과 일치하지 않는 경우에도 본 연구의 결과는 유효하게 된다. 이는, RTT 값이 최소일 때 기준이 되는 임의의 순방향/역방향 전송 시간으로 호스트간 시간차를 계산하고, 그 값을 이용하여 RTT 값이 증가하였을 때 순방향/역방향 전송 시간을 계산하기 때문에 최소 RTT의 증가치를 순방향/역방향 별로 각각 계산할 수 있기 때문이다. 따라서 패킷의 왕복 전송 경로가 비대칭(asymmetric routing path)이고, 그 결과 RTT가 최소 일 때 순방향과 역방향 전송 시간이 동일하지 않는 경우에도 본 알고리즘은 정상적으로 동작할 수 있는 장점을 갖는다.

2.2 알고리즘의 동작

본 절에서는 1절에서 간략하게 설명한 알고리즘을 상세히



(그림 1) 종단간 순방향/역방향 전송 지연 측정 파라메타

설명한다.

(그림 1)은 호스트 a와 b의 패킷 교환과 종단간 순방향/역방향 전송 지연 시간 측정을 위한 요소들을 자세하게 보여주고 있으며, 알고리즘을 설명하기 위하여 필요한 변수들을 다음과 같이 정의한다.

- t_{hi} : 호스트 h의 TSC 값, i=o(originate), r(receive), c(current).
- TD_{ab} : 호스트 a와 b에 세트 된 TSC의 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 2)는 본 논문에서 제안한 알고리즘을 의사 코드(pseudo code)로 나타낸 것이다. 알고리즘의 동작을 간단히 설명하면 다음과 같다. 3~6번 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:  RTTmin = big number;
02:  until (end of packet) {
03:       if (RTTmin > RTTcp) {
04:            RTTmin = RTTcp;
05:            FTTref = BTTref = RTTmin / 2;
06:            TDab = tac - tbr - BTTref;
07:       }
08:       FTTcp = tbr - tao + TDab;
09:       BTTcp = tac - tbr - TDab;
10:       FTTdif = FTTcp - FTTref;
11:       BTTdif = BTTcp - BTTref;
12:  }
```

(그림 2) 제안된 순방향/역방향 전송 지연 측정 알고리즘

패킷의 송수신시 순방향/역방향 전송 지연 시간(FTT_{dif}, BTT_{dif})을 구하는 방법은 다음과 같다.

1) 호스트 a는 데이터를 전송하고, 호스트 b로부터 ACK를 수신한다. (그림 1)
2) ACK를 수신 한 송신측 노드는 (그림 2)의 알고리즘에 따라서 FTT_{dif}, BTT_{dif}를 계산한다.

2.3 실제 망 환경에서의 적용

본 연구에서 제시한 알고리즘을 실제의 환경에서 적용하는 경우 시간이 경과함에 따라서 올바르지 못한 결과를 나타내는데, 이는 송신측과 수신측 호스트가 타임스탬프 시간차를 가질 뿐만 아니라 호스트간 타임스탬프 시간차가 변하기 때문이다.

호스트간 타임스탬프 시간차가 변하는 이유는 호스트간 타임스탬프 카운터의 틱(Tick) 간격 차이 때문이다[5]. 각 호스트는 틱 시간에 한번씩 TSC를 증가시켜 현재 시각을 설정하는데, 틱 간격이 실제 시간의 경과와 정확하게 일치하지 못하는 경우, 시스템의 시각은 시간이 경과함에 따라서 점차 실제 시각과 차이를 갖게 된다. 이러한 결과, 예를 들면, 현재 호스트 a와 b의 실제 타임스탬프 시간차 값이 100ms 인데 1시간 후에는 실제 타임스탬프 시간차 값이 110ms로 변해 타임스탬프 시간차가 10ms만큼 증가 할 수 있다.

결국, 호스트간 타임스탬프 카운터의 틱 간격 차이로 인하여 시간이 흐를수록 실제 호스트간 타임스탬프 시간차가 약간씩 증가 또는 감소 되어 본 연구에서 제시한 알고리즘의 정확도를 떨어뜨리는 결과를 나타나게 된다. 그러나, 이러한 타임스탬프 시간차의 변화는 아주 작기 때문에, 짧은 시간 (수 분~수 십분) 동안 경로 별 혼잡 정도를 파악하는데 큰 영향을 주지는 못한다. 그리고, 송신측과 수신측간의 타임스탬프 시간 차 변화를 빠르게 감지할 수 있도록 하기 위하여 (그림 2)에서 제시한 알고리즘을 (그림 3)과 같이 변경하고, (그림 4)와 같은 의사 코드로 타임스탬프 시간차가 1ms 증감하는데 소요되는 시간을 계산하여 (그림 3)의 알고리즘에 적용한다. (그림 3)의 알고리즘에서는 현재 패킷의 RTT 값이 기존의 RTT_{min}과 같거나 작을 때, 송수신 호스트간의 타임스탬프 시간차를 다시 계산하도록 하고 있다.

```
01:  RTTmin = big number;
02:  until (end of packet) {
03:       if (RTTmin ≥ RTTcp) {
04:            RTTmin = RTTcp;
05:            FTTref = BTTref = RTTmin / 2;
06:            TDab = tac - tbr - BTTref;
07:       }
08:       FTTcp = tbr - tao + TDab;
09:       BTTcp = tac - tbr - TDab;
10:       FTTdif = FTTcp - FTTref;
11:       BTTdif = BTTcp - BTTref;
12:  }
```

(그림 3) 실제 시스템에 적용하기 위한 알고리즘

(그림 4)의 타임스탬프 시간차가 1ms 증감하는데 소요되는 시간을 계산하기 위하여 필요한 변수들은 다음과 같이 정의한다.

- t_{ac_ref} : RTT_{min} 일 때의 t_{ac} 값.
- t_{ac_cp} : 현재 패킷의 t_{ac} 값.
- t_{ac_dif} : t_{ac_cp}와 t_{ac_ref}의 차.

- TD_{ab_ref} : RTT_{min} 일 때의 TD_{ab} 값.
- TD_{ab_cp} : 현재 패킷의 TD_{ab} 값.
- TD_{ab_dif} : TD_{ab_cp}와 TD_{ab_ref}의 차.
- TD_{ab_roc} : TD_{ab}가 1ms 변하는데 소요되는 시간.

```
01:  if (RTTmin == RTTcp) {
02:       tac_dif = tac_cp - tac_ref;
03:       TDab_dif = TDab_cp - TDab_ref;
04:       TDab_roc = tac_dif/TDab_dif;
05:  }
```

(그림 4) 실제 시스템간 타임스탬프 시간차 변화율 계산 알고리즘

3. 시뮬레이션 및 실제 환경에서의 테스트

본 연구에서는 제시한 알고리즘의 타당성을 확인하기 위하여 다음과 같은 세 가지 방식으로 테스트 하였다.

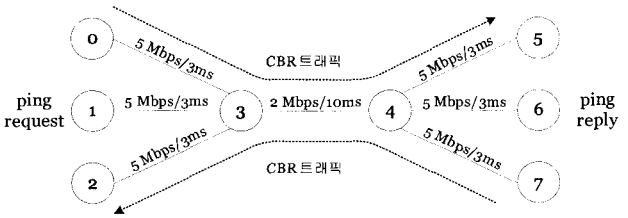
첫째, 인터넷 시뮬레이션 도구로 많이 사용되고 있는 NS-2를 이용하여 시뮬레이션 하였다.

둘째, 호스트와 호스트 간 홉 수가 1인 실제의 근거리 망에서 테스트 하였다.

셋째, 호스트와 호스트 간 홉 수가 22인 실제의 원거리 망에서 테스트 하였다.

3.1 NS-2에서의 시뮬레이션

(그림 5)는 NS-2에서 시뮬레이션을 위하여 구성한 망 환경을 보여주고 있다.



(그림 5) 시뮬레이션의 네트워크 구성

시뮬레이션에 이용된 노드(Node : 0, 1, 2, 5, 6, 7)와 라우터(Router : 3, 4)의 기본 큐(Queue) 동작 방식은 드롭-테일(Drop-tail)이다. 노드와 라우터 사이는 5Mbps의 대역폭(Bandwidth)과 3ms의 지연(Delay)이 발생하며, 라우터와 라우터 사이는 2Mbps의 대역폭과 10ms의 지연이 발생한다. 노드 0과 7에서 CBR(Constant Bit Rate)을 이용하여 순방향/역방향 경로에서 트래픽을 발생시켰다. 시뮬레이션에 이용된 응용(Application)은 NS-2에서 제공하는 ping 응용으로 이 응용에 본 논문의 알고리즘을 추가하여 사용하였으며, 또한 ping 응답 메시지를 전송할 때 응답 시간을 추가하여 전송하도록 수정하였다. 노드 1에서는 ping 요청(Request) 메시지를 전송하고, 노드 6에서는 이에 대하여 ping 응답(Reply) 메시지를 전송하도록 구성하였다.

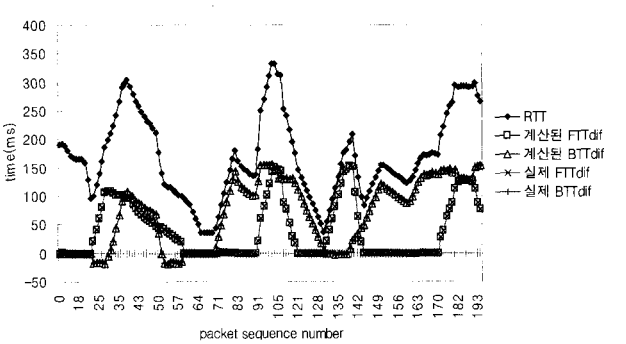
실제 네트워크와 유사한 트래픽을 발생시키기 위해 노드 0에서는 <표 1>의 각 경우를 무작위로 추출하여 1초 동안 추출된 경우의 트래픽을 발생하였으며, 전체 60초 동안 측정하였다. 노드 7에서 발생하는 트래픽은 노드 0에서 발생하는 트래픽과 동일한 방법으로 발생되며, 방향은 ping 응용의 관점에서 역방향으로 발생한다. 또한, ping 응용에서는 실험이 이루어지는 60 초 동안 1초에 10번의 요청과 응답이 발생하도록 하였으며, 이 결과 ping 응용에서 발생하는 순방향과 역방향 트래픽 량은 각각 180Bps (Byte per second, 18 × 10) 이다.

<표 1> 노드 0과 7에서 무작위(random)로 발생한 CBR 트래픽 량

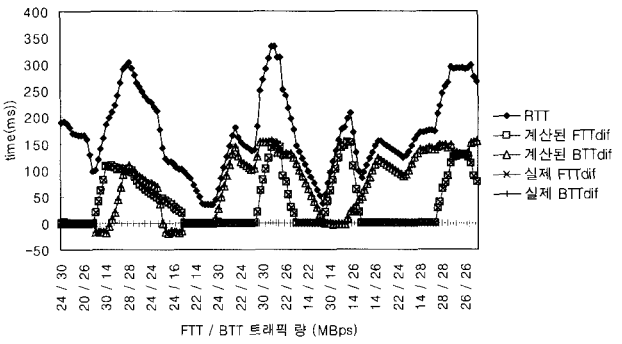
경우	패킷 크기 (Byte)	패킷 간격 (Second)	대역폭 (Bps)
1	600	0.005	12M
2	700	0.005	14M
3	800	0.005	16M
4	900	0.005	18M
5	1000	0.005	20M
6	1100	0.005	22M
7	1200	0.005	24M
8	1300	0.005	26M
9	1400	0.005	28M
10	1500	0.005	30M

(그림 6)은 본 논문에서 제안한 알고리즘으로 계산된 FTT_{dif}, BTT_{dif} 와 실제 FTT_{dif}, BTT_{dif}를 비교한 그래프로써 전체 60초 동안 측정한 결과값 중 처음 20초 동안의 결과치를 나타내고 있다. NS-2에서는 송신측과 수신측의 타임스탬프 카운터 값이 같기 때문에 실제의 FTT_{dif}, BTT_{dif} 값을 측정할 수 있게 된다.

(그림 6)의 (A)와 (B)는 같은 결과의 그림으로써 시뮬레이션 결과의 이해를 쉽게 하기 위해 X축 데이터를 각각 패킷의 순서번호, 순방향/역방향 경로의 트래픽 량으로 하였다. (그림 6)의 결과를 살펴보면 계산된 FTT_{dif}, BTT_{dif}가 실제 FTT_{dif}, BTT_{dif}와 동일하게 변하는 것을 알 수 있으며, 또한 FTT, BTT의 트래픽 량과 FTT_{dif}, BTT_{dif}가 비례하여 증감하는 것을 알 수 있다.



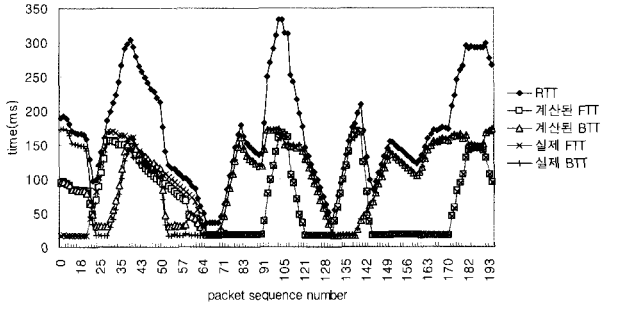
(A) 패킷의 순서번호에 따른 비교



(B) FTT / BTT 트래픽 량에 따른 비교

(그림 6) FTTdif, BTTdif 와 실제 FTTdif, BTTdif 비교

(그림 7)은 본 연구의 알고리즘에 의해 계산된 FTT, BTT와 실제 FTT, BTT 값을 비교한 그림으로써 RTT 값이 최소일 때 계산한 TD_{ab}를 이용하여 FTT, BTT 값을 계산하게 되며 실제 FTT, BTT 값과 거의 같게 증감하는 것을 보여주고 있다.

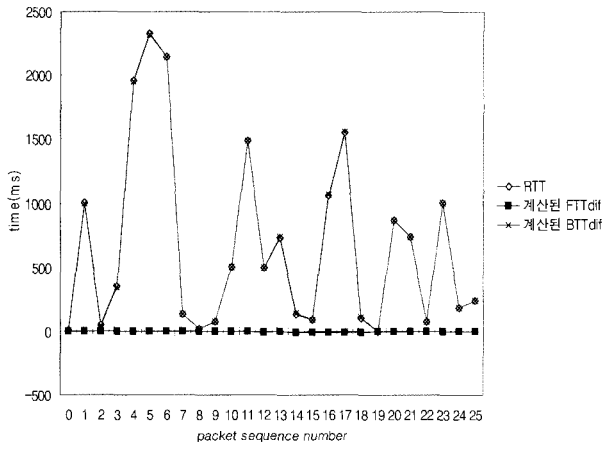


(그림 7) 계산된 FTT, BTT와 실제 FTT, BTT 비교

(그림 6)의 (A), (B)번을 비교하여 결과를 살펴보면, 패킷의 순서 번호가 66~69 일 때 RTT값이 최소 인 것을 알 수 있으며, 이 구간의 양방향 트래픽 량도 작아 패킷의 흐름이 원활하다는 것을 알 수 있다. 또한 (그림 7)에서도 순서번호 66번부터 계산된 FTT, BTT가 실제 FTT, BTT와 같게 증감하는 것을 볼 수 있다. 따라서, 순서번호 69번의 패킷을 기준으로 FTT_{ref}, BTT_{ref}, TD_{ab}를 계산하고, 이 계산 값을 이용하여 순서번호 70번부터의 패킷에 대한 FTT_{dif}, BTT_{dif}를 계산하면 더욱 정확한 경로 별 혼잡도를 측정할 수 있음을 나타내고 있다. 그리고 패킷의 순서번호가 23~29, 51~58인 구간에서 BTT_{dif}가 -18.1까지 감소하는 것을 볼 수 있는데, 그 이유는 58번 패킷 이전에 구한 FTT_{ref}, BTT_{ref} 값이 혼잡이 발생했을 때 구한 값이기 때문에 BTT_{ref} 값이 BTT_{cp} 값 보다 크게 되어 BTT_{dif} 값이 큰 음의 값이 되었음을 알 수 있다.

그러나, 66번 패킷은 혼잡이 가장 없을 때 전송된 패킷으로, 이때 계산된 FTT_{ref}, BTT_{ref} 값이 가장 낮은 FTT, BTT 값이 되어 이후부터는 BTT_{dif}, FTT_{dif}가 음의 값이 생기지 않는다. 그리고, 측정된 RTT 값이 200개가 되지 않는 이유는 혼잡이 발생했을 때 ping 패킷이 드롭(Drop)되었기 때문이다.

3.2 실제 근거리 네트워크에서 테스트

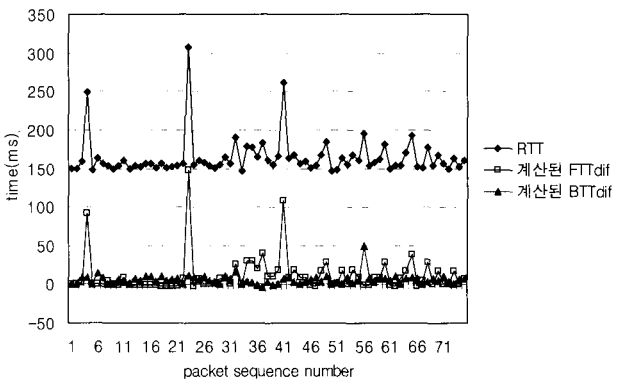


(그림 8) 실제 근거리 네트워크에서 FTT_{dif}, BTT_{dif}

(그림 8)은 동일한 이더넷 (Ethernet)에 연결된 송신측과 수신측 사이에서 데이터가 전송될 때 측정한 FTT_{dif}, BTT_{dif} 값을 보여주고 있다. 이때, ICMP timestamp 요청/응답 메시지를 이용하여 측정하였으며, 순방향으로는 ICMP timestamp 요청 이외의 다른 트래픽이 발생하지 않으며, 역방향으로는 ICMP timestamp 응답 메시지 이외에 추가의 FTP 트래픽을 발생시켜 인위적인 역방향 전송 지연을 유도하였다.

순방향 경로에선 트래픽이 발생하지 않아 RTT 값의 변화와 상관없이 FTT_{dif} 값이 거의 일정하며, 역방향 경로에선 트래픽이 발생되어 BTT_{dif} 값이 RTT 값의 변화와 비례하게 변하는 것을 볼 수 있다.

3.3 실제 원거리 네트워크에서 테스트



(그림 9) 실제 원거리 네트워크에서 FTT_{dif}, BTT_{dif} 측정

(그림 9)은 홉 수가 22인 원거리(galaxy.cs.berkeley.edu)에 있는 호스트에 패킷을 송신하고 수신하는 과정에서 측정한 FTT_{dif}, BTT_{dif}를 보여주고 있다. 그리고 테스트는 근거리 네트워크와 마찬가지로 ICMP timestamp 요청/응답 메시지를 이용하여 측정하였다.

실제의 순방향/역방향 경로에 트래픽을 발생시킬 수는 없지만, NS-2와 실제 근거리 네트워크에서의 시뮬레이션 결과를 토대로 볼 때, 패킷 순서번호 3, 22, 40번 패킷이 전송될 때 순방향 전송 지연의 증가로 인하여 전체 RTT가 크게 증가하는 것을 알 수 있다.

4. 결 론

본 논문에서는 패킷을 전송할 때 RTT가 증감하는 경우, 그 원인이 되는 순방향과 역방향의 전송 지연 변화를 측정할 수 있는 방법을 제시하였고, NS-2를 이용한 시뮬레이션과 실제 근거리/원거리 네트워크에서의 테스트를 통하여 제안된 알고리즘의 타당성을 확인하였다.

기존의 단순 RTT 측정 방법은 호스트간 타임스탬프 카운터의 차로 인하여 순방향과 역방향에서의 전송 지연 시간을 알 수 없는 문제를 갖는다. 따라서, 전송 경로에 혼잡이 발생했을 때 혼잡의 원인이 순방향 경로인지 아니면 역방향 경로인지 알 수가 없었으며, 적절한 대처 방안을 세울 수가 없었다. 하지만 본 논문에서 제시한 새로운 알고리즘을 기존 RTT 측정 방법에 추가하여 적용한 결과 손쉽게 경로별 혼잡 정도를 파악할 수 있었다.

추후, 본 연구의 알고리즘을 TCP에 적용하여 순방향/역방향 경로 별 혼잡에 따라 패킷 송신의 양을 조절하는 연구를 수행하려고 한다.

참 고 문 헌

[1] Behrouz A. Forouzan, "TCP/IP Protocol Suite", 2'nd Ed., McGraw-Hill, 2003.

[2] W. Richard Stevens, "TCP/IP Illustrated, Volume1 The

Protocols", Addison-Wesley, 1994.
[3] W. Richard Stevens, "TCP/IP Illustrated, Volume2 The Implementation", Addison-Wesley, 1995.
[4] W. Richard Stevens, "UNIX Network Programming", 2'nd Ed., Prentice Hall PTR, 1990.
[5] Daniel P. Bovet, Marco Cesati, "Understanding the Linux Kernel", O'Reilly, 2001.
[6] David L.Mills, "Network Time Protocol(Version 3): Specification Implementation and Analysis", RFC 1305, March, 1992.
[7] G. Almes, "A One-way Delay Metric for IPPM", RFC 2679, September, 1999.
[8] "NS-Tutorial", http://www.isi.edu/nanam/ns/tutorial/
[9] "NS by Example", http://nile.wpi.edu/NS/
[10] "NS-Manual", http://www.isi.edu/nsnam/ns/doc/index.html/
[11] V. Paxson, "End-to-End Routing Behavior in The Internet," IEEE/ACM Transactions on Network, Vol.5, No.5, pp.601~615, Oct., 1997.
[12] V. Paxson, "End-to-End Internet Packet Dynamics," IEEE/ACM Transactions on Networking, Vol.7, No.3, pp.277~292, June, 1999.

황 순 한

e-mail : hsh305@hotmail.com
1999년 배재대학교 정보통신공학과(공학사)
2005년 국립 한밭대학교 정보통신전문대학원(공학석사)
관심분야 : 인터넷프로토콜, 임베디드, 라우팅 프로토콜 등

김 은 기

e-mail : egkim@hanbat.ac.kr
1987년 고려대학교 전자공학과 (공학사)
1989년 고려대학교 전자공학과 (공학석사)
1994년 고려대학교 전자공학과 (공학박사)
1995년~현재 국립 한밭대학교 정보통신·컴퓨터공학부 부교수
관심분야 : 인터넷프로토콜, 무선랜, 보안 프로토콜, 라우팅 프로토콜 등