TCP Vegas RTT Ambiguity 문제와 그 해결

정회원 김 종 덕*, 김 종 권*

TCP Vegas RTT Ambiguity Problem and Its Solutions

Jong-Deok Kim*, Chong-Kwon Kim* *Regular Members*

요 약

TCP 구현의 하나인 Vegas는 RTT 측정값을 바탕으로 혼잡을 인지하며 윈도우 크기 등 혼잡제어를 위한 주요 인자를 결정한다. 이러한 Vegas가 기존 TCP 구현에 비해 우수한 성능과 높은 공평성을 가짐은 여러 논문을 통해 보고되었다. 그런데 우리는 Vegas의 혼잡회피 방안이 TCP 데이터 흐름의 비대칭적 특성을 제대로 반영하지 못하며, 이것이 양쪽 방향 상태를 반영하는 RTT 측정값을 순방향 링크의 상태 해석에 이용하기 때문임을 발견하였다. 우리는 이를 Vegas RTT Ambiguity 문제라 하고, 네트워크 내의 패킷 스케줄링 방법인 Small Get Priority Queue, TCP Timestamp Option을 이용한 수정 Vegas 구현 등 두 가지 해결 방안을 제시하였다. 우리는 시뮬레이션을 통해 Vegas RTT Ambiguity 문제와 제시한 해결 방안들을 검증하였다.

ABSTRACT

Vegas is an implementation of TCP using RTT measurement for the detection of network congestion and decides the congestion control parameters, such as window size, based on it. There are several research results showing that Vegas achieves better throughput and higher fairness than the existing TCP implementations, such as Tahoe, Reno. However we have noticed that Vegas poorly reflects the asymmetric characteristic of TCP data transfer because of the wrong interpretation of RTT which reflects backward link state as well as forward link state. We named this problem as "Vegas RTT Ambiguity Problem" and developed 2 solutions. one is "Small Get Priority Queue", a packet scheduling technique in router, and the other is "Modified Vegas" using TCP Timestamp Option. We used simulation to show the problem and to verify the solutions.

Ⅰ. 서 론

TCP 구현의 하나인 Vegas는, 패킷의 유실을 통해 망의 혼잡을 인지하는 Tahoe, Reno와 같은 기존 TCP 구현들과는 달리 RTT(Round Trip Time) 측정값의 변화를 분석하여 망의 혼잡을 인지한다. Vegas의 RTT 기반 혼잡 인지 방법은 혼잡의 초기 상황을 빠르게 인지할 수 있어, 혼잡 발생 이후에 인지하고 대처하는 기존 구현에 비해 패킷 유실이 적고 따라서 재전송량이 준다. 또한 혼잡이 발생할 때까지 꾸준히 혼잡 윈도우 크기를 늘이는 기존 구현과 달리, 네트워크 내에 버퍼링되는 패킷의 수를 특정 범위 이내로 제한하여 혼잡 윈도우 크기의 과대화를 막아, 높은 처리율을 유지하면서도 혼잡 상황을 억제하는 특징이 있다[1].

기존 구현을 이용할 경우 긴 RTT를 가지는 TCP 연결이 짧은 RTT를 가지는 TCP 연결에 비해 자원 이용에 있어 불리함이 [2][3] 등을 통해 지적되었다. 그러나 Vegas는 RTT 길이가 다른 연결들이 경쟁을 할 때 이들의 전송률이 특정 범위 내로 수렴하도록 동작하며, 그 결과 기존 구현들에 비해 공평성 측면에서 훨씬 우수함이 [4]를 통해 보고되었다.

* 서울대학교 컴퓨터공학부(ckim@popeye.snu.ac.kr)
논문번호 : 99481-1130, 접수일자 : 1999년 11월 30일
※ 본 연구는 정보통신부에서 지원하는 대학기초연구지원사업(99-024)으로 수행되었습니다.

하지만 앞에서 밝힌 Vegas의 우수성과 관련한 여러 내용과는 별개로, Vegas의 기본 아이디어라고 할 수 있는 RTT 측정값 해석에 오류가 있음을 우리는 인식하였다.

TCP는 양방향 데이터 전송을 지원하지만 실제 TCP의 이용 패턴은 일방향 데이터 전송 형태를 취한다. 이는 대부분의 인터넷 서비스가 서버-클라이언트 구조를 취하며 이 구조의 실제 정보흐름의 형태가 일방향의 비대칭적 형태를 띄기 때문이다. TCP 연결의 비대칭적 정보흐름은 비대칭적 네트워크 링크 이용으로 이어진다. 그림 1은 그 예로 교육망(KREN)의 미국 Sprint로의 T3 링크 유통량 정보를 보여주는 것인데, 가는선 및 채워짐으로 표현된 것은 Sprint로부터 교육망으로의 유통량을, 굵은 실선으로 표현된 것은 교육망으로부터 Sprint로의 유통량이다. 그림 1에서 링크 이용의 비대칭 성이 확연하게 드러나 있다.

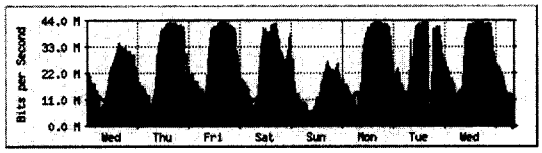


그림 1. 교육망 해외 인터넷 이용 유통량

링크가 비대칭적으로 이용되므로 순방향(Forward) 링크와 역방향(Backward) 링크의 혼잡 상태 역시 다를 수 있다. 순방향 링크에서 혼잡이 발생할 경우 순방향 데이터 전송을 줄이는 것은 혼잡 제어를 위해 타당하다. 그러나 역방향 링크에서 혼잡이 발생할 경우 순방향 데이터 전송을 줄이는 것이 옳지 않을 수 있다. 역방향 링크의 혼잡 원인은 일반적인 경우 역방향 데이터 흐름의 과다에 따른 것으로 이것의 해소는 역방향 데이터 흐름에 대해 작동하는 혼잡 제어 기능에 의해 이루어져야 한다(즉 역방향 데이터 흐름의 감소를 통해 이루어져야 한다). 물론 순방향 데이터 흐름의 감소를 통해 역방향으로 되먹임(Feedback)되는 ACK의 양을 줄일 수 있으며 이를 통해 역방향 혼잡의 해소에 기여할 수도 있을 것이다. 그러나 순방향 데이터 흐름의 감소량에 비해 역방향 정보량의 감소가 미미하여 역방향 링크의 혼잡은 해결하지 못한 채 순방향 링크의 이용률만 떨어뜨릴 수 있다.

Vegas는 혼잡 인지를 위해 RTT 측정값을 이용한다. 그런데 RTT 측정값은 양방향 링크의 속성을 나타내는 값으로 이를 이용하는 Vegas는 순방향 링크의 혼잡과 역방향 링크의 혼잡을 구별하지 못한다. 그 결과 Vegas는 역방향 링크의 혼잡에 과도하게 대응하여 순방향 링크의 이용률을 떨어뜨리는 문제점을 가진다.

이 문제를 해결하기 위해서는 ACK 패킷이 역방향 링크의 혼잡에 의해 지연이 되는 것을 방지하거나 또는 Vegas가 순방향 링크의 혼잡과 역방향 링크의 혼잡을 구별할 수 있어야 할 것이다. 우리는 이 논문에서 전자의 해결책으로 네트워크 내의 패킷 스케줄링 방법인 Small Get Priority Queue, 후자의 해결책으로 TCP Timestamp Option을 이용한 수정 Vegas 구현 등 두 가지 해결 방안을 제시하였다.

Ⅱ. Vegas RTT Ambiguity 문제

여기서는 앞서 정의한 Vegas RTT Ambiguity 문제를 구체적으로 살펴본다. 이를 위해 Vegas의 핵심 알고리즘인 혼잡회피(Congestion Avoidance) 방안[11]을 간단히 설명한다.

① 해당 TCP 연결에 대하여 BaseRTT를 구한다.

- 해당 TCP 연결이 혼잡을 겪지 않을 때의 RTT 값을 BaseRTT라고 정의한다. 하지만 정확한 BaseRTT 값을 알 수 없으므로 Vegas 구현에서는 실제 해당 연결의 모든 RTT 측정값들 중 최소 값을 BaseRTT로 지정한다. 대체로 해당 연결로 인해 라우터의 큐가 길어지기 전인 첫 번째 TCP 세그먼트에 대한 RTT 측정값이 BaseRTT가 된다.
- 이 TCP 연결의 현재 윈도우 크기가 W라면 이 연결에 대하여 기대할 수 있는 전송률, Expected는 다음과 같다.

$$\text{"Expected"} = W / BaseRTT$$

② 현재의 실제 전송률, Actual을 구한다.

- 실제 전송률은 W를 현재 RTT 측정값으로 나누어 구한다. 즉 Actual = W / RTT (식에 사용되는 RTT 값은 현재 RTT 측정값, 즉 가장 최근의 RTT 측정값이다)이다. BaseRTT ≤ RTT이므로 Actual ≤ Expected가 항상 성립한다.
- 실제 전송률 계산은 매 RTT 마다 한번 수행한다.

③ 실제 전송률과 기대 전송률을 비교하고, 이를 바탕으로 윈도우 크기 W를 적절하게 변경한다.

- Diff = (Expected - Actual) × BaseRTT로 정의

한다. Diff는 Expected, Actual의 정의에 따라 음수가 아닌 값이 된다.

- α , β 라는 두 가지 임계값을 정의한다. 직관적으로 α , β 값의 의미를 말하면, α 는 해당 연결이 너무 적은, β 는 해당 연결이 너무 많은 여분의 데이터를 네트워크 내에서 가지고 있음을 판단하는 기준 값이다. Vegas는 TCP 연결이 네트워크 내에서 가지는 여분의 데이터, 즉 라우터의 큐를 차지하는 데이터 양을 α 와 β 값 사이로 제한하고자 한다.

- 이를 위해 Vegas는 Diff < α 일 경우 W를 선형 증가시키고 Diff > β 일 경우 W를 선형 감소시킨다. Vegas의 선형 증가/선형 감소 방법은 기존 구현의 선형 증가/지수형 감소에 비해 TCP 연결의 W 변화가 급격하지 않아 연결의 전송상태가 안정적이며, TCP 트래픽의 Bursty 속성을 완화시켜 네트워크 역시 안정적인 동작을 할 수 있도록 한다.

우리는 TCP의 비대칭적 특성을 고려할 때 Vegas의 혼잡 회피 방안에 오류가 있을 수 있음을 인식하였다. 비대칭적 정보흐름 및 혼잡 상태를 고려하면 ③에서 혼잡 또는 네트워크 이용률의 척도로 쓰이는 Diff가 순방향 링크의 상태만을 표현하는 것이 타당하다. 이를 위해서는 Diff의 계산에 이용되는 두 인자 Expected와 Actual이 각각 순방향에 대한 기대 전송률과 실제 전송률이어야 한다. 구분을 위해 순방향 링크에 대한 기대 전송률과 실제 전송률을 각각 F_Expected, F_Actual이라 하자.

그런데 Expected의 경우 정의에 따라 역방향 및 순방향 모두에 대해 혼잡을 배제하였으므로 이를 F_Expected로 그대로 이용해도 무방하다. 그러나 Actual의 경우 계산 과정에 이용되는 RTT 측정값이 양방향의 상태 정보를 포함하고 있어 F_Actual로 이용함에 문제가 있다. F_Actual을 구하기 위하여 다음과 같은 변수들을 정의한다.

T_f : TCP 연결이 혼잡을 겪지 않을 때 순방향 링크에 대한 OTT (One-way Trip Time)

T_b : TCP 연결이 혼잡을 겪지 않을 때 역방향 링크에 대한 OTT

t_{fq} : 혼잡 회피 방안 적용 시점에서 측정된 순방향 링크 대기열 지연(Queueing Delay)

t_{bq} : 혼잡 회피 방안 적용 시점에서 측정된 역방향 링크 대기열 지연(Queueing Delay)

Vegas에서 이용된 BaseRTT= $T_f + T_b$로, RTT = $T_f + T_b + t_{fq} + t_{bq}$로, Expected = $\frac{W}{T_f + T_b}$로, Actual = $\frac{W}{T_f + T_b + t_{fq} + t_{bq}}$로 각각 표현할 수 있다. 그리고 F_Expected는 Expected를 그대로 이용하므로 $\frac{W}{T_f + T_b}$ 표현할 수 있다.

③에서 Diff = (F_Expected - F_Actual) × BaseRTT로 변경하면 Diff는 순방향 링크에 있는 여분의 데이터 량으로 생각할 수 있다. 여분의 데이터 Diff를 순방향 실제 전송률로 전송하는데 걸리는 시간을 순방향 링크의 대기열 지연이라 할 수 있으므로 $\frac{Diff}{"F_{Actual}} = t_{fq}$의 관계가 성립한다. 이를 앞의 식에 대입하면 F_Actual을 앞서 정의한 변수들로 다음과 같이 표현할 수 있다.

$$"F_{Actual} = \frac{W}{T_f + T_b + t_{fq}}$$

즉 위의 해석으로부터 F_Actual을 구하기 위해서는 RTT 측정값에서 역방향 링크의 대기열 지연을 제거하여야 함을 알 수 있다. 만약 이를 제거하지 않을 경우 역방향 링크의 혼잡으로 순방향 링크의 실제 전송률을 과소 평가할 수 있고 이는 ③의 적용에 의해 W를 불필요하게 낮추는 결과로 이어진다.

우리는 시뮬레이션을 통해 현 Vegas 구현의 이러한 문제점을 살펴보았다. 시뮬레이션 도구로는 인터넷 관련 주요 시뮬레이션 도구로 최근 관련 연구 논문 등에서 많이 활용되고 있는 NS[5]를 이용했다. 그림 2는 시뮬레이션에 사용된 네트워크 구성이다.

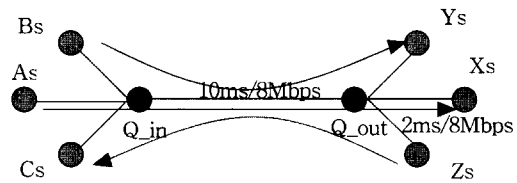


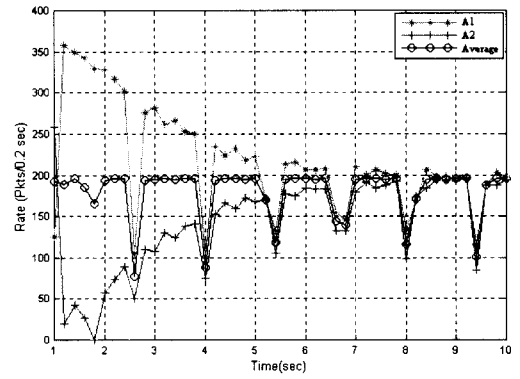
그림 2. 시뮬레이션 네트워크 구성

시뮬레이션에는 3가지 종류의 TCP 연결이 있는데 A→X는 관측 대상 TCP 연결이며, B→X는 관측 대상 TCP 연결과 같은 데이터 흐름 방향을 가지는 TCP 연결이며, Z→C는 관측 대상 TCP 연결과 반대의 데이터 흐름 방향을 가지는 TCP 연결이다. (nA, tA), (nB, tB), (nZ, tZ)를 각 연결 종류의 (연결 수, 첫 번째 연결의 시작 시간)이라고 하겠다.

동기화 효과(Synchronization Effect)를 해소하기 위해 모든 연결 종류에 있어 연결들간의 시작 시간에 사이에 간격을 두었다. 간격의 길이는 평균 50ms의 지수(Exponential)분포를 따르는 난수 발생기로부터 얻은 값을 이용하였다. TCP의 MSS(Maximum Segment Size)는 500bytes로 하였다. 역방향으로는 데이터를 Piggy-Backing하지 않고 ACK만 보내는 것으로 하였다. 따라서 ACK패킷의 크기는 TCP 헤더 크기만 반영하여 40bytes로 하였다. 시뮬레이션에 이용된 라우터의 기본 Queue 동작 방식은 Drop-tail이다.

실험 1) nA=2, nB=0, nZ=0

그림 3, 4는 두 개의 관측 대상 TCP 연결만을 생성하였을 때의 Tahoe와 Vegas의 성능을 보이는 실험 결과이다. 각 그림에서 위쪽은 매 0.2초 동안의 전송량을 바탕으로 구한 각 연결의 순간 속도(*, +)와 그 평균(o)이다. 아래 그림은 각 연결의 누적 전송량과 그 평균이다. Vegas가 성능 면에서 조금 더 우수하며 전송률도 안정적임을 확인할 수 있다.



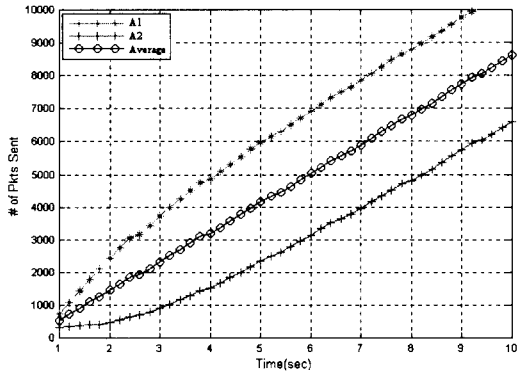
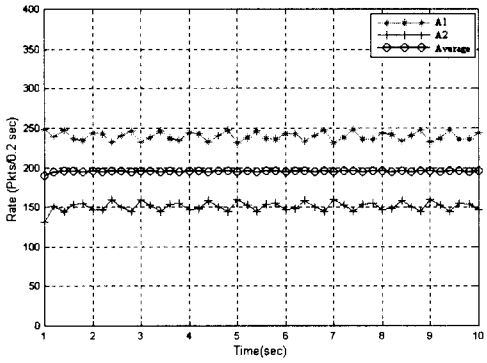


그림 3. 실험 1 구성에서의 TCP-Tahoe 동작 결과



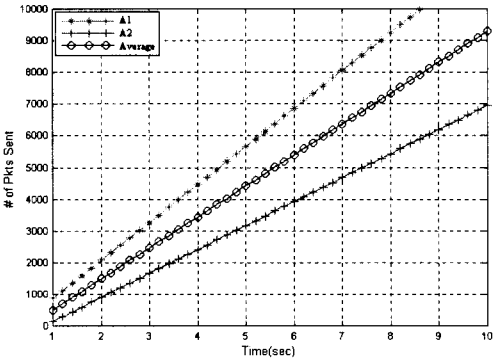
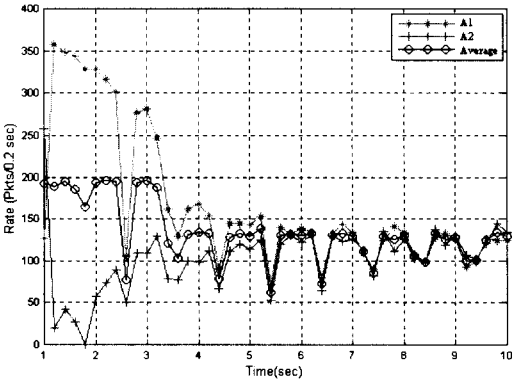


그림 4. 실험 1 구성에서의 TCP-Vegas 동작 결과

실험 2) (nA=2, tA=1.0), (nB=1, tB=3.0), nZ=0

관측 대상 TCP 연결의 데이터 흐름 순방향과 같은 데이터 흐름 방향을 가지는 새로운 TCP 연결을 추가하여 관측 대상 TCP 연결에 대하여 순방향 혼잡이 발생할 경우 Tahoe와 Vegas의 성능을 비교하기 위한 실험이다. 그림 5와 그림 6에서 보이듯 역시 Vegas가 우수하고 안정적인 성능을 보이며 순방향 혼잡에 잘 대처함을 알 수 있다.



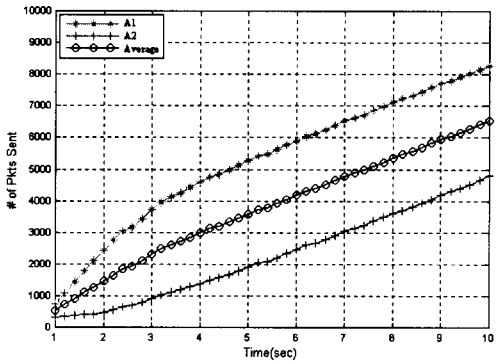
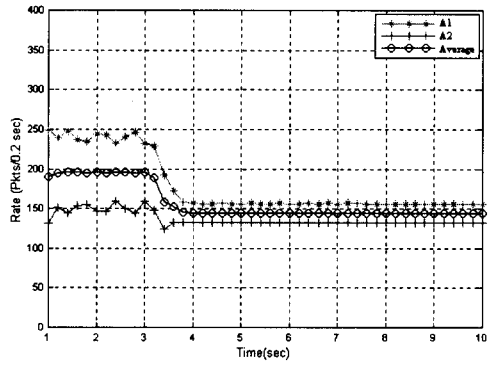


그림 5. 실험 2 구성, 순방향 링크 혼잡 발생 시 TCP-Tahoe 동작 결과



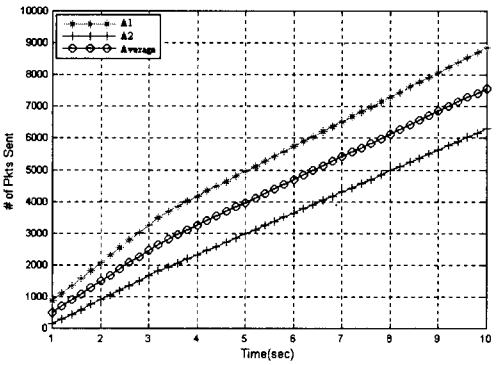
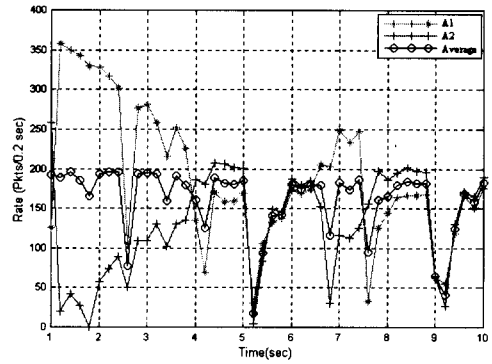


그림 6. 실험 2 구성, 순방향 링크의 혼잡 발생 시 TCP-Vegas 동작 결과

실험 3) (nA=2, tA=1.0) nB=0, (nZ=8, tZ=3.0)

관측 대상 TCP 연결의 데이터 흐름 순방향과 반대의 데이터 흐름 방향을 가지는 새로운 TCP 연결을 추가하여 관측 대상 TCP 연결에 대해 역방향 혼잡이 발생할 경우 Tahoe와 Vegas의 성능을 비교하기 위한 실험이다. 우리가 제기한 Vegas RTT Ambiguity 문제를 잘 보일 수 있는 실험이다. 그림 7에서 보이듯 Tahoe의 경우 관측 대상 TCP 연결의 성능이 (실험 1)에 비해 저하되었음을 알 수 있다. 이것은 추가된 역방향 연결들에 의해 순방향으로의 ACK 패킷이 증가함에 따라 발생하는 것으로 적절한 수준의 성능 저하라 할 수 있다. 그러나 Vegas의 경우 그림 8에서 드러나듯 급격한 성능 저하가 있다. ACK 패킷의 증가를 고려하더라도 그것의 순방향 링크 점유율(8개 연결의 ACK 패킷)이 (실험 2)의 순방향 링크 점유율(1개 연결의 MSS 크기만큼의 TCP 패킷)에 미치지 못할 것임을 고려할 때 너무 큰 성능 저하라 할 수 있다.



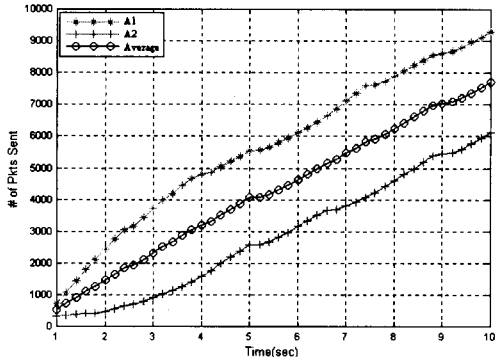
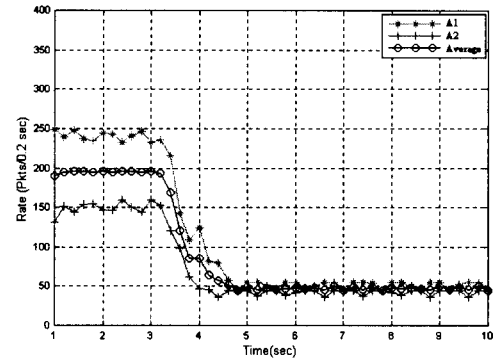


그림 7. 실험 3 구성, 역방향 링크 혼잡 발생 시의 TCP-Tahoe 동작 결과



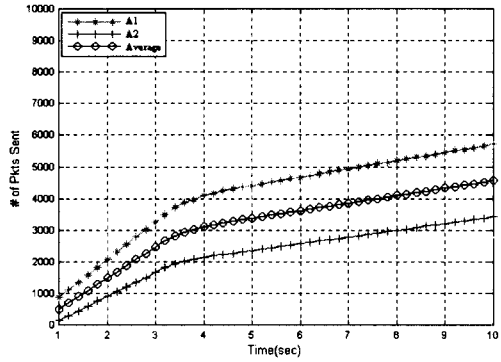


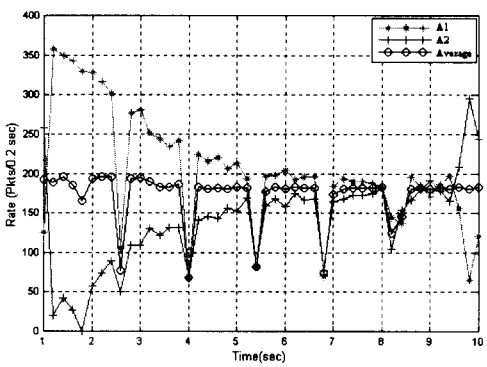
그림 8. 실험 3 구성, 역방향 링크 혼잡 발생 시의 TCP-Vegas 동작 결과

Ⅲ. 해결 방안

1. Small Get Priority Queue

이 방안은 네트워크 내에 있는 라우터의 패킷 스케쥴링 방법을 변경하여, TCP의 RTT 측정값이 순방향 링크의 상태만을 반영할 수 있도록 하는 것이다. 이를 위해 RTT에서 값을 원천적으로 제거하여, 즉 RTT = $T_f + T_b + t_{fq}$가 되도록 한다. 따라서 호스트가 기존 Vegas를 이용하여도 역방향 혼잡의 영향을 받아 순방향 전송률을 줄이는 문제를 일으키지 않는다.

이를 위해서는 역방향으로 전송되는 ACK가 대기열 지연을 겪지 않도록 라우터에서 ACK 패킷을 다른 데이터 패킷에 비해 우선 처리하여야 한다. 그런데 라우터가 ACK 패킷을 구별하고 이를 우선 처리하는데는 문제가 있다. 라우터의 라우팅 및 포워딩은 네트워크 계층, 즉 IP 계층에서 수행되는데 비해 ACK는 전송 계층, 즉 TCP 계층의 정보이기 때문이다. 그래서 우리는 Small Get Priority Queue를 대안으로 제시한다. Piggy-Backing이 아닐 경우 ACK는 데이터 패킷에 비해 훨씬 작은 크기이다 .



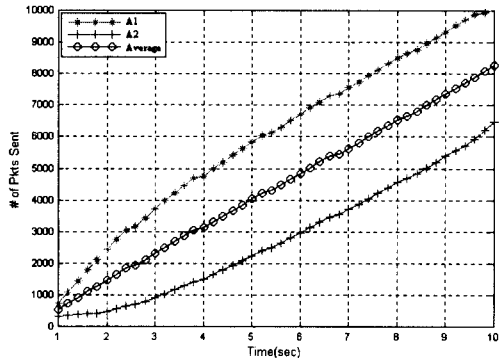
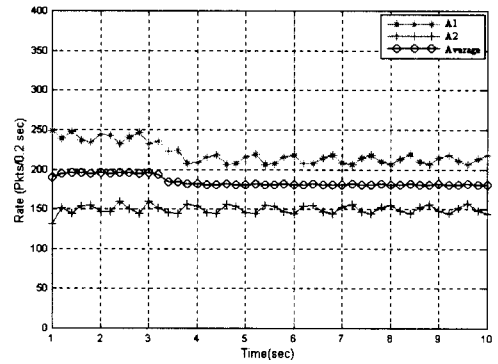


그림 9. 실험 3 구성, Small Get Priority Queue를 적용했을 때 TCP-Tahoe 동작 결과



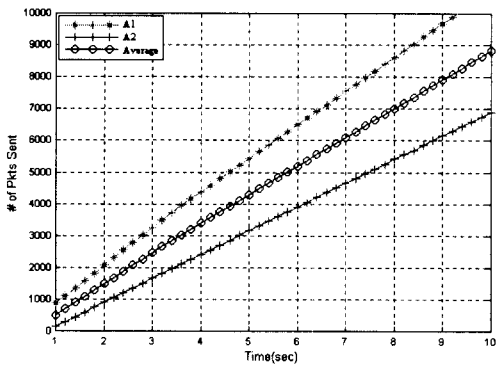


그림 10. 실험 3 구성, Small Get Priority Queue를 적용했을 때 TCP-Tahoe 동작 결과

따라서 지정한 크기 이하의 패킷에 대해 우선권을 줌으로써 ACK 패킷을 우선 처리하는 효과를 가져올 수 있다.

그림 9와 그림 10은 (실험 3)의 구성에서 라우터에 Small Get Priority Queue를 적용했을 때의 실험 결과이다. 두 구현 모두 기존 (실험 3)의 결과에 비하여 성능 향상이 있었다. 특히 Vegas의 경우 기존 (실험 3)의 결과에 비해 아주 좋은 성능을 보이

고 있다. 그러나 Small Get Priority Queue는 역방향 경로 상의 모든 라우터가 이를 지원하여야 한다는 문제점이 있다. 또 Piggy-Backing 되는 경우의 ACK를 구별할 수 없으며 라우터의 동작 방식이 악용 당할 소지가 있다는 등의 문제점이 있다.

2. Timestamp Option을 이용한 수정 Vegas

이 방안에서 ACK 패킷은 Small Get Priority Queue에서와는 달리 혼잡을 겪을 수 있다. 따라서 F_Actual을 구하기 위해서는 측정된 RTT 값에서 를 제거하여야 한다. 우리는 이를 위해 RFC-1323[6]에서 정의된 TCP Timestamp Option을 이용하였다. TCP Timestamp Option은 TCP 성능에 밀접한 연관이 있는 RTT의 정확한 측정을 위해 제안된 것이다. TCP Timestamp Option을 이용하면 ACK 패킷에 해당 ACK 패킷을 전송하는 시각을 수신자(ACK 송신자)가 기록하게 할 수 있다.

이를 이용하여 RTT에서 t_{bq}를 제거하는 간단한 방법을 다음에서 제시하겠다.

다음의 세 변수를 정의한다.

t_s : RTT 측정에 이용된 데이터 패킷의 송신자 측 전송 시각

t_r : RTT 측정에 이용된 데이터 패킷에 대한 ACK 패킷의 수신자 측 전송 시각

t_a : RTT 측정에 이용된 데이터 패킷에 대한 ACK 패킷의 송신자 측 도착 시각

이 변수들의 값은 Timestamp Option 및 현재 Clock 측정 등을 통해 Vegas 송신 측에서 모두 얻을 수 있다. 즉 송신 측은 Timestamp Option 등을 통해 $< t_s, t_r, t_a >$ 형태의 Sample을 얻을 수 있다. 기존 RTT 측정값은 $t_a - t_s$와 동일하므로 이 Sample을 통해 쉽게 구할 수 있다. 그러나 순방향 및 역방향 OTT는 그렇지 못하다. 직관적으로 순방향 OTT인 $T_f + t_{fq} = t_r - t_s$로, 역방향 OTT인 $T_b + t_{bq} = t_a - t_r$로 생각할 수 있지만 여기에는 오류가 있을 수 있다. t_r의 측정에 이용된 수신자 Clock과 t_a, t_s 측정에 이용된 송신자 Clock이 다를 수 있기 때문이다[8]. 그 대표적 원인으로 두 Clock 사이에 존재하는 Offset을 들 수 있다. 즉 한 Clock이 다른 Clock에 비해 Offset 만큼 빠르거나 또는 느릴 수 있다는 것이다.

Offset T_O를 다음과 같이 정의한다.

T_O : 절대시각 t 시점에서 송신자 Clock의 값을 S(t), 수신자 Clock의 값을 R(t)라 할 때

$$T_O = R(t) - S(t)$$

따라서 다음과 같은 관계식이 성립한다.

$$t_r - t_s = T_f + t_{fq} + T_O \quad \text{---} \quad ⓐ$$

$$t_a - t_r = T_b + t_{bq} - T_O \quad \text{---} \quad ⓑ$$

Vegas의 BaseRTT 측정 방법과 유사한 이유로 다음의 관계식이 성립한다고 가정한다.

$$\min\{ t_r - t_s \} \rightarrow T_f + T_O \quad \text{---} \quad ⓒ$$

$$\min\{ t_a - t_r \} \rightarrow T_b - T_O \quad \text{---} \quad ⓓ$$

ⓒ, ⓓ에서 얻어진 각 값을 T_{fO}, T_{bO}라고 정의한다. 위의 관계식을 이용하면 새로 정의된 값들을 이용하여 F_Actual 계산에 필요한 시간 정보를 표현할 수 있다.

$$\begin{aligned} T_f + T_b + t_{fq} &= (T_f + t_{fq} + T_O) + (T_b - T_O) \\ &= t_r - t_s + T_{bO} \end{aligned} \quad \text{---} \quad ⓔ$$

관계식 ⓔ로부터 T_{bO} 값 및 t_r, t_s 값을 통해 F_Actual 계산을 할 수 있음을 알 수 있다.

이로부터 우리는 다음과 같은 수정 Vegas 혼잡회피 방안을 제시한다.

① BaseRTT의 정의는 변함없다. 기존 Vegas 구현에서는 BaseRTT를 실제 구하기 위해 BaseRTT = min $\{ t_a - t_s \}$의 방법을 취했다고 할 수 있다. 그러나 수정 Vegas 구현에서는 BaseRTT = min$\{ t_r - t_s \}$ + min$\{ t_a - t_r \}$ = $T_{fO} + T_{bO}$로 변경한다. 그리고 $< t_s, t_r, t_a >$ Sample들로부터 T_{fO} 및 T_{bO}를 구하기 위한 두 개의 변수를 유지하여야 한다. 수정 Vegas의 BaseRTT 측정법은 기존 방법에 비해 항상 BaseRTT 정의에 보다 가까운 결과를 가진다.

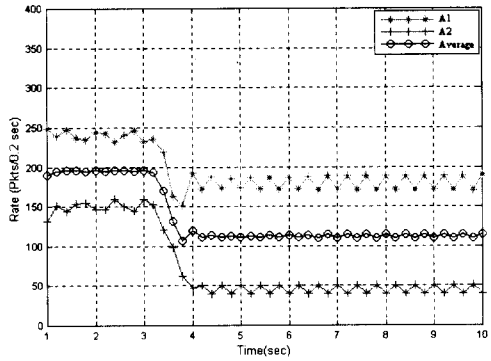
② Actual을 F_Actual로 변경한다. F_Actual은 가장 최근의 $< t_s, t_r, t_a >$ Sample과 ①을 통해 유지하고 있는 T_{bO}, 현재 윈도우 크기 W로부터 구할 수 있으며 다음과 같다.

$$\text{F\_Actual} = \frac{W}{t_r - t_s + T_{bO}}$$

③ Diff = (F_Expected - F_Actual) × BaseRTT

로 정의한다. 나머지는 기존 Vegas와 동일하다.

수정 Vegas 구현을 (실험 3)의 구성에 적용하여 실험하였다. 단 송신자와 수신자 Clock 사이에 난수 값 발생을 통해 임의의 Offset이 생기도록 하였다. 그림 11, 그림 12, 그림 13은 실험 결과 그래프이다. 그림 11과 그림 12는 관측 대상인 두 개의 TCP Vegas 연결 중 하나는 수정 Vegas 구현을 다른 하나는 기존 Vegas 구현을 이용할 경우의 결과이다. 그림 13은 두 개 모두 수정 Vegas 구현을 이용할 때의 결과이다.



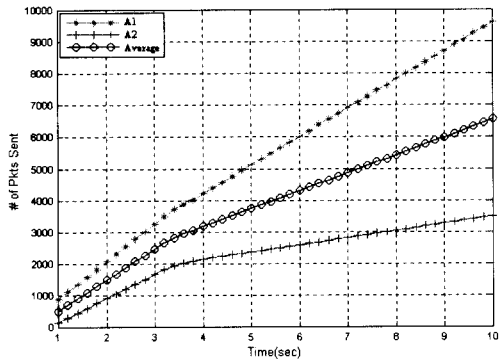
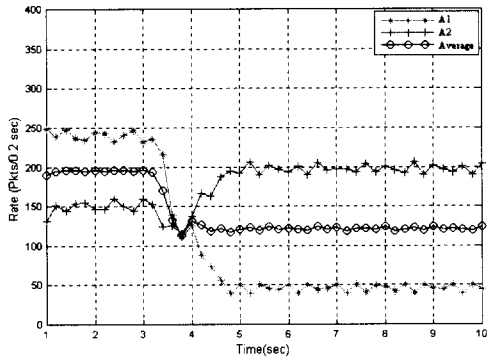


그림 11. 실험 3 구성, 수정 TCP를 먼저 시작한 nA 연결에만 적용한 경우



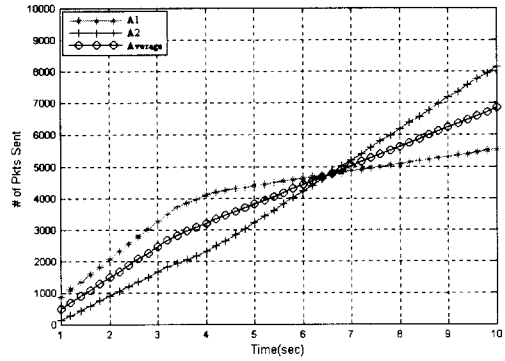
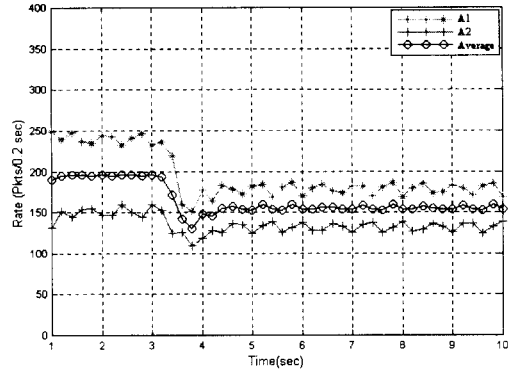


그림 12. 실험 3 구성, 수정 TCP를 뒤에 시작한 nA 연결에만 적용한 경우



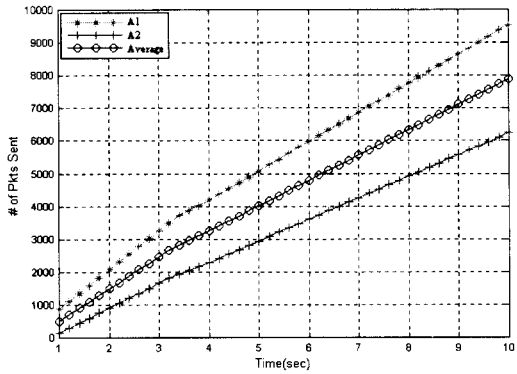


그림 13. 실험 3 구성, 수정 TCP를 nA 연결 모두에 적용한 경우

실험 결과를 통해 수정 Vegas 구현이 우리가 제기한 Vegas RTT Ambiguity 문제를 잘 해결하고 있음을 알 수 있다. 특히 수정 Vegas 구현은 앞서 제시한 Small Get Priority Queue와는 달리 네트워크 구성 요소의 기능을 변경시킬 필요가 없을 뿐 아니라 기존 TCP Timestamp Option을 이용하므로 Vegas 송신 측의 구현만을 변경하면 되는 장점이 있다. 또한 ACK Piggy-Backing시의 구별 불가, 동

작 방법의 악용과 같은 문제점도 갖지 않는다.

게다가 수정 Vegas 구현은, 이용되는 $< t_s, t_r, t_a >$ Sample이 송신자 및 수신자 측의 지역 Clock에 의한 측정값으로 양 Clock간에 Offset이 존재하여도 오류 없이 작동한다. 우리의 F_Actual 측정 작업은 OTT 측정을 위한 기존 연구들과 유사한 점이 있다. 그러나 이러한 기존 연구들의 경우 OTT 계산을 위해 순방향 및 역방향 링크가 대칭적이라는 가정을 요구한다[7]. (혼잡이 없을 경우 순방향 OTT와 역방향 OTT가 동일하다고 가정한다) 그러나 우리의 제안은 이러한 전제가 필요 없고 순방향 및 역방향 링크가 비대칭적일 수 있다. 수정 Vegas 구현은 Timestamp Option 이용과 T_{f0} 및 T_{b0} 유지를 위해 기존 Vegas에 비해 좀 더 많은 오버헤드를 필요로 한다. 하지만 이러한 오버헤드는 이를 통해 얻는 성능 향상에 비해 미약한 수준이라 생각한다.

Ⅳ. 결 론

우리는 Vegas가 TCP 데이터 흐름의 비대칭적 특성을 제대로 반영하지 못하며, 그 원인이 Vegas의 혼잡 회피 방안에 이용되는 RTT 값이 TCP 데이터 흐름의 비대칭적 특성을 제대로 반영하지 못함에 있음을 밝혔다. 그리고 이를 해결하기 위한 방안으로 Small Get Priority Queue와 수정 Vegas 구현 방안을 제시하였고 그 타당성을 시뮬레이션을 통해 검증하였다.

우리는 제시한 두 방안 중 수정 Vegas 구현이 현실 구현 가능성 측면에서 보다 나은 방안이라 생각한다. 그런데 수정 Vegas 구현에서 해결하여야 할 중요 문제는 기존 OTT 측정 연구에서 이미 제기된 송신자 Clock과 수신자 Clock간의 비동기화 문제이다. 비동기화의 원인으로는 Clock 간의 Offset 및 Skew, 그리고 Clock Adjustment 등을 들 수 있다. 수정 Vegas에서는 양 Clock간에 Offset이 존재하여도 추가의 정보나 전제 조건 없이도 해결할 수 있었다. 그러나 수정 Vegas는 Skew나 Clock Adjustment가 있을 경우는 고려하지 않고 있으며 이는 앞으로 해결해야 할 과제이다.

그리고 최근 가입자망을 중심으로 상, 하향 링크의 전송 속도가 비대칭적인 링크가 많이 포설되고 있다. 이러한 환경에서는 되먹임되는 ACK가 역방향 혼잡의 주요 원인이 될 수 있다. 순방향의 데이터 흐름을 줄여 역방향 되먹임을 줄일 수도 있겠지만 ACK 지연 인자 조정 등을 통한 ACK 감축(Suppression)이 보다 효과적인 방안일 수 있을 것이다. 순방향 링크의 이용률을 떨어뜨리지 않고 역방향의 혼잡을 해소하는 것은 Vegas를 포함한 모든 TCP 구현에서 검토되어야 할 문제라 할 수 있으며 앞으로 해결해야 할 과제이다.

참 고 문 헌

[1] L. Brakmo, S. O'Malley, and L. Peterson, "TCP Vegas : New Techniques for Congestion Detection and Avoidance", *Proceedings of ACM SIGCOMM*, pp 24-35, August 1994.

[2] D. Lin and R. Morris, "Dynamics of Random Early Detection", *Proceedings of ACM SIGCOMM*, pp. 127-137, 1997.

[3] Thomas R. Henderson, Emile Sahouria, Steven McCanne, and Randy H. Katz, "On Improving the Fairness of TCP Congestion Avoidance", *Proceedings of IEEE GLOBECOM*, pp 539-544, 1998

[4] J. Mo, R. J. La, V. Anantharam, and J. Walrand, "Analysis and Comparison of Reno and Vegas", *Proceedings of IEEE INFOCOM*, pp 1556-1563, 1999

[5] S. McCanne and S. Floyd, NS (Network Simulator), http://www-nrg.ee.lbl.gov/ns

[6] V. Jacobson, R. Braden, D. Borman, "TCP Extensions for High Performance", *IETF RFC-1323*, May 1992

[7] V. Paxson, "On Calibrating Measurements of Packet Transit Times", *Proceedings of SIGMETRICS '98*, June 1998.

[8] Sue B. Moon, Paul Skelly, Don Towsley. "Estimation and Removal of Clock Skew from Network Delay Measurements," *Proceedings of IEEE INFOCOM*, pp 227-234, 1999.

김 종 덕(Jong-Deok Kim) 학생회원
1994년 2월 : 서울대학교 전산과학과 졸업
1996년 2월 : 서울대학교 전산과학과 석사
1996년 3월~현재 : 서울대학교 컴퓨터공학부 박사 과정
<주관심 분야> 인터넷, 이동 통신

김 종 권(Jong-Deok Kim) 정회원
1981년 : 서울대학교 산업공학과 (학사)
1982년 : 조지아공대 산업공학과 (석사)
1987년 : 일리노이대 전산과학과 (박사)
1987년~1991년 : 벨 통신 연구소 연구원
1991년~현재 : 서울대학교 컴퓨터공학부 부교수.
<주관심 분야> 인터넷, 이동통신망, 초고속 통신,