DOI : 10.5392/JKCA.2011.11.2.069

UDT 환경에서 혼잡상황 예측 및 패킷손실을 고려한 성능향상 기법

A Performance Improvement Method with Considering of Congestion Prediction and Packet Loss on UDT Environment

박종선*, 이승아*, 김승해**, 조기환*
전북대학교 전자정보공학부*, 한국과학기술정보연구원**

Jong-Seon Park(jschris25@chonbuk.ac.kr)*, Seung-Ah Lee(salee86@chonbuk.ac.kr)*,
Seung-Hae Kim(shkim@kisti.re.rk)**, Gi-Hwan Cho(ghcho@chonbuk.ac.kr)*

요약

최근 네트워크 기술의 비약적인 발전으로 사용자가 이용할 수 있는 대역폭이 증가하고 있다. 이에 따라 대용량 고속 네트워크에서 보다 신속하고 안정적인 전송기법이 요구되고 있다. UDT(UDP based Data Transfer protocol)는 UDP 기반의 전송 프로토콜이며, 일정 SYN time마다 rate control을 진행함으로써 긴 RTT 환경에서 TCP 보다 두드러진 성능을 보인다. 하지만, NAK 발생시 고정적인 sendInterval의 증가와, 이전 시간의 RTT에 기반한 flow control로 인해 최적의 성능을 기대하기 힘들다. 본 논문에서는 실험에 의한 결과값을 바탕으로 NAK 수신시 RTT 구간에 따라 sendInterval을 조절하는 rate 제어기법을 제시한다. 또한 TCP vegas에 기반하여 네트워크 혼잡을 예측하는 향상된 flow 제어기법을 제시한다. 실제 Testbed를 구성하여 실험한 결과, 각각의 제안기법에 대해 약 20Mbps정도 향상된 throughput이 측정되었다. 또한, 두 기법을 혼합 적용한 결과에서는 최고 26Mbps의 성능 향상을 확인하였다.

■ 중심어 : | 대용량 전송 | 고속네트워크 | TCP Vegas | 혼잡예측 | UDT |

Abstract

Recently, the bandwidth available to an end user has been dramatically increasing with the advancing of network technologies. This high-speed network naturally requires faster and/or stable data transmission techniques. The UDT(UDP based Data Transfer protocol) is a UDP based transport protocol, and shows more efficient throughput than TCP in the long RTT environment, with benefit of rate control for a SYN time. With a NAK event, however, it is difficult to expect an optimum performance due to the increase of fixed sendInterval and the flow control based on the previous RTT. This paper proposes a rate control method on following a NAK, by adjusting the sendInterval according to some degree of RTT period which calculated from a set of experimental results. In addition, it suggests an improved flow control method based on the TCP vegas, in order to predict the network congestion afterward. An experimental results show that the revised flow control method improves UDT's throughput about 20Mbps. With combining the rate control and flow control proposed, the UDT throughput can be improved up to 26Mbps in average.

■ keyword : | Bulk Data Transmission | High-speed Network | TCP Vegas | Congestion Prediction | UDT |

* 이 논문은 2010년도 제4회 정보통신분야학회 합동학술대회 우수논문입니다.

접수번호 : #101202-004
접수일자 : 2010년 12월 02일
심사완료일 : 2011년 01월 14일
교신저자 : 조기환, e-mail : ghcho@chonbuk.ac.kr

I. 서 론

최근 들어서 백본 망뿐만 아니라, 지역 망도 비약적으로 발전함에 따라 사용자에게도 보통 수십에서 수백 Mbps급의 네트워크가 제공되고 있다. 그러나 현재 네트워크에서 주로 사용되고 있는 TCP는 저속 네트워크에 최적화되어 있어 증가된 네트워크의 성능을 충분히 활용하지 못하고 있다. 특히 대역폭이 수 Gbps에서 수백 Gbps에 이르는 연구망들이 개발되고 있어 이용효율은 더욱 감소될 전망이다. 따라서 큰 대역폭, 높은 RTT 환경에서의 데이터 전송에 관한 연구가 필요하다.

TCP는 AIMD(Additive Increase Multiplicative Decrease)를 사용하여 혼잡 윈도우 크기를 급격하게 감소시킴으로써 혼잡상황으로부터 빠르게 벗어날 수 있다. 그러나 급격한 윈도우 크기 변화로 인하여 네트워크 성능을 효율적으로 사용하지 못하고 있다. TCP의 성능 문제를 개선하고자 HSTCP[1], STCP[2], BIC-TCP[3]등 많은 연구들이 진행되어 왔다. 그러나 아직까지 네트워크 이용 효율이 저조한 편이다. 게다가 이러한 TCP 수정은 커널의 수정을 요구하기 때문에 일반 사용자들이 접근하기에는 사실상 어려움이 있다.

UDP는 비교적 전송 성능이 우수하지만 연결지향성이 아니기 때문에 안정성을 보장하지 못한다. UDP의 성능 개선을 위해 최근 RBUDP[4], Tsunami[5], UDT(UDP Based Data Transfer protocol)[6]와 같은 대용량 데이터 전송 기법들이 연구되고 있다. 특히, UDT는 높은 대역폭과 큰 RTT 환경을 고려한 어플리케이션 기반의 전송기법이다. UDT는 UDP 기반의 데이터 전송과 TCP 기반의 컨트롤 패킷 전송을 사용하여 신뢰성을 보장한다. UDT는 손실이 없는 환경에서는 매우 안정적이고 높은 성능을 보이지만 NAK수신시 rate control 문제와 SYN time동안 flow control문제에 있어 개선이 필요하다. 손실이 발생하는 환경에서 고정적으로 sendInterval을 증가시키고, SYN time동안 congestion window를 선정하는데 있어 이전 RTT를 사용하기 때문에 최적의 성능을 기대하기 힘들다. 본 논문에서는 NAK 수신시 고정적으로 sendInterval을 증가하지 않고 RTT 구간에 따라 다른 가중치를 적용한다. 또한, TCP vegas에서 착안한 혼잡예측기법을 적용하여 UDT 성능을 향상시켰다.

논문의 구성은 다음과 같다. 우선 2장에서 TCP Vegas와 UDT에 대해 살펴본 후, 3장과 4장에서는 성능을 향상시키기 위한 제안기법을 UDT에 적용한다. 마지막으로 5장에서 결론을 맺는다.

II. 관련 연구

1. TCP Vegas

TCP Reno는 TCP 구현 중 하나로 AIMD 혼잡제어 알고리즘을 사용한다. Reno는 윈도우 크기를 증가시키다가 혼잡이 발생하여 패킷이 손실되면 윈도우의 크기를 급격하게 감소시킴으로써 혼잡에서 벗어난다[7]. 이러한 Reno의 혼잡제어 방식은 혼잡 상황에서 빠르게 벗어날 수 있는 반면 전송률의 많은 변화로 전체적인 전송 효율이 감소하게 된다. TCP 의 다른 구현인 TCP Vegas는 Reno의 이러한 점을 보완하기 위하여 Brakmo와 Peterson에 의해 제안되었다[8]. Vegas는 Reno와 같은 NAK기반의 혼잡제어 방식이 아닌 RTT를 이용해 혼잡을 미리 예측하는 알고리즘을 사용한다. RTT가 높을수록 혼잡 정도가 증가한 것으로 판단하여 윈도우 크기를 감소시키고 반대로 RTT가 낮을수록 혼잡 정도가 감소한 것으로 판단하여 윈도우 크기를 증가시킨다. NAK 발생 후 윈도우를 감소시키지 않고 혼잡을 미리 예측하여 상황에 대처하기 때문에 패킷 손실률이 감소하게 되어 손실 패킷을 재전송하는 일이 줄어들게 된다. Vegas의 혼잡회피 알고리즘은 다음과 같다[9].

i) RTT의 최소값 baseRTT를 구한다.

대체로 최소 RTT 값은 패킷 손실이나 혼잡이 거의 발생하지 않을 첫 번째 TCP 세그먼트에 대한 값이 된다.

ii) 기대전송률인 Expected를 구한다.

$$\mathrm{Exp}ected = \frac{W}{baseRTT} \quad (1)$$

기대전송률은 혼잡윈도우의 크기 W를 baseRTT로 나누어 구한다.

iii) 실제전송률인 Actual을 구한다.

$$Actual = \frac{W}{RTT} \quad (2)$$

실제전송률은 혼잡윈도우의 크기 W를 현재의 RTT로 나누어 구한다.

iv) 두 전송률간의 차이 Diff를 구한다.

$$\begin{aligned} Diff &= (Actual - \mathrm{Expected}) \times baseRTT \\ &= \left(\frac{W}{baseRTT} - \frac{W}{RTT}\right) \times baseRTT \end{aligned} \quad (3)$$

baseRTT는 RTT들의 최소값이기 때문에 baseRTT < RTT 가 항상 성립한다. 따라서 Actual > Expected 이므로 Diff는 항상 0보다 큰 값을 갖는다.

v) Diff에 따라 윈도우 크기를 조절한다.

$$W = \begin{cases} W-1, & (\mathrm{if}\ Diff > \beta) \\ W, & (\mathrm{if}\ \alpha \le Diff \le \beta) \\ W+1, & (\mathrm{if}\ Diff < \alpha) \end{cases} \quad (4)$$

Diff의 값에 따라 상태를 구분하기 위하여 임계값 α와 β를 사용한다. 임계값 β보다 큰 경우에는 혼잡한 것으로 판단하여 W를 감소시키고, α보다 작은 경우에는 네트워크가 혼잡하지 않은 것으로 판단하여 W를 증가시킨다.

2. UDT(UDP based Data Transfer protocol)

UDT는 UDP기반 대용량 전송프로토콜이다. 응용계층상에서 별도의 라이브러리를 구현함으로써, 일반 사용자가 쉽게 접근 할 수 있는 특징을 지닌다. 또한, 데이터의 전송은 속도가 빠른 UDP채널을 활용하면서 TCP와 같이 컨트롤 정보를 제어함으로써 신뢰성을 제공한다. [Fig. 1]은 UDT의 소프트웨어 아키텍처를 나타낸다.

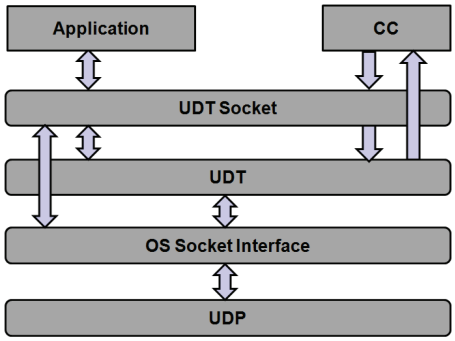

Fig. 1. UDT software architecture

UDT는 rate control 혼잡제어를 하는데 있어 TCP에서 사용되는 AIMD(Additive Increase Multiplicative Decrease)방법과 유사한 DAIMD(Decrease Additive Increase Multiplicative Decrease) 메커니즘을 사용한다. 이는 전송 시작 단계에서는 증가인자 값을 빠르게 증가시키고, 시간에 따라 증가인자 값을 점차적으로 감소시키는 방법이다. 이러한 전송방법은 대역폭이 큰 네트워크에서 가용대역폭에 신속하게 도달하게 하는 특징을 지닌다. UDT에서 사용되는 rate control은 sendInterval 조절을 통한 데이터 전송을 일컫는다. 여기서, sendInterval이란, 송신측이 패킷을 보내고 난 후 다음 패킷을 보내기 위한 시간의 간격이다. 즉, 일정 SYN time(0.01s)동안 패킷 손실이 없고 ACK를 수신하게 되면, 수신측으로부터 측정된 가용대역폭을 바탕으로 다음 SYN time동안 증가될 패킷의 양을 결정한다. 또한, 이를 이용하여 다음 SYN time의 sendInterval이 계산된다. 식 (5)와 식 (6)은 각각의 패킷 증가량과 sendInterval을 위한 계산식이다[10]. 식(5)에서 B는 가용대역폭, MSS는 최대 세그먼트의 크기이며 1500Byte를 기본 값으로 사용한다. 식(6)에서 P′는 현재 sendInterval이다.

$$inc = \max\left(10^{\lceil \log_{10} B \rceil - 9}, 1/1500\right) \times 1500/MSS \quad (5)$$

$$SYN/P = SYN/P' + inc \quad (6)$$

대역폭에 따른 증가량은 이해하기 쉽게 [Tab. 1]과 같이 표현되어 질 수 있다.

Tab. 1. UDT inc 파라메터 계산

B(Mb/s)	inc(packets/SYN)
B≤0.1	0.00067
0.1<B≤1	0.001
1<B≤10	0.01
10<B≤100	0.1
...	...

한편, UDT는 수신 측으로부터 NAK를 수신하면 전송량을 감소시킨다. 또한, 수신 측으로부터 수신한 윈도우 값을 바탕으로 다음 SYN time동안 전송될 윈도우의 양을 조절하는 flow control기법을 사용하는데 다음은 두 관점에서 살펴본다.

2.1 패킷손실 발생시 혼잡제어

SYN time동안 데이터 전송시 패킷손실이 발생하면 수신측은 손실된 패킷의 sequence 번호를 기록하여 NAK 패킷에 실어 송신측으로 보내게 된다. SYN time 동안 데이터를 전송하는 과정에서 패킷손실 발생에 의해 수신측으로부터 NAK를 수신하게 되면, 송신측은 전송량을 감소시킨다. 전송량을 감소시키는 방법은 ACK수신시와 마찬가지로 sendInterval을 조절한다. UDT는 패킷손실이 발생하면 일괄적으로 sendInterval을 1/8만큼 늘려 전송량을 조절한다. [Fig. 2]는 수신 측으로부터 NAK 수신시 UDT의 혼잡제어 알고리즘을 나타낸다.

UDT's Rate Control

```
avg_nak_num : average nak number
When NAK is occurred

onNak()
{
   snd_interval = snd_interval×1.125;
   avg_nak_num = update;
   nak_count = 1;
}
```

Fig. 2. NAK 수신시 혼잡제어 알고리즘

[Fig. 2]에서 NAK가 발생되면, 송신 측에서는 현재의 전송속도(snd_interval)에 가중치 1.125를 곱함으로써 전송 패킷의 간격을 늘리고 있다. 이는 곧, 전송속도를 현재의 전송속도에서 1/8만큼 감소시키는 의미와 동일하다.

하지만, 서로 다른 RTT와 패킷로스가 존재하는 상황에서 전송속도를 고정적으로 1/8만큼 줄이는 방법은 최적의 성능을 기대하기 힘들다. 따라서 패킷로스율과 RTT 구간에 따라 최적의 가중치를 적용함으로써 보다 향상된 성능향상 기법이 필요하다.

2.2 UDT 흐름제어

UDT가 일정한 SYN time동안 sendInterval을 통한 전송기법을 사용함에 있어 수신 측으로부터 수신한 윈도우 값은 rate control을 위한 flow control을 담당한다. 즉, rate control은 sendInterval을 조정해 패킷 전송 시간을 증가시킴으로써 혼잡을 제어하고, flow control은 윈도우 크기를 조절함으로써 혼잡을 제어한다. UDT의 기본 flow control은 [Fig. 3]과 같다

UDT는 ACK를 수신했을 때 슬로우 스타트(slow start)이면 윈도우의 크기를 ACK를 전송 받을 때까지의 기간 동안 수신된 세그먼트의 총 개수 S만큼 증가된다. S의 개수는 Sequence 번호의 차이로 계산할 수 있다. UDT는 매 데이터 전송마다 ACK와 NAK를 주고받지 않고 일정 시간(SYN time) 마다 한번 씩 전송 받기 때문에 UDT에서 수신된 Sequence의 개수 S는 TCP에서처럼 1이 아니다.

UDT's Flow Control

```
onAck()
{
  if (slow start)
        W= W+S;
 else
        W=RcvRate/(Interval+RTT)*1000000+16;
 .
 .
 .
 Modify Packet Sending Period;

 Set Maximum Transfer Rate;
}
```

Fig. 3. UDT 흐름제어

NAK를 전송받았을 때는 sendInterval을 조절하기 때문에 혼잡윈도우 크기를 직접적으로 감소시키거나

증가시키지 않는다. ACK를 수신하였을 경우 슬로우 스타트가 아니면 RcvRate(ReceiveRate)를 이용해 혼잡 윈도우의 크기를 증가시키고 감소시킴으로써 혼잡을 제어한다. RcvRate는 수신측에서 버퍼의 크기와 전송된 혼잡윈도우의 크기를 비교하여 결정되는 최소값이다. 따라서 ACK를 수신 받을 때마다 RcvRate에 따라 윈도우의 크기는 계속 변하게 되며 지난 전송시간에 얻어진 RTT 값을 이용하기 때문에 현재의 상황에 적절히 대처하지 못할 가능성이 있다. 본 논문에서는 슬로우 스타트가 아닌 부분에서 전송률과 RTT를 적용시켜 기존 UDT의 Flow 제어를 개선한다.

III. 제안된 성능향상 기법

1. 패킷손실을 고려한 최적의 혼잡제어 기법

기존 UDT에서 패킷손실에 의한 NAK 수신시 sendInterval을 고정적으로 1/8만큼 증가시킨다. 그러나 본 논문에서 제안한 기법은 각 RTT구간에 대해 최적의 가중치를 적용한다[11]. RTT구간은 실험을 통해 세 구간으로 나눠 구간별로 가중치를 달리 적용하여 sendInterval을 다르게 적용한다. 최적의 가중치는 네트워크 시뮬레이터인 NS-2를 사용하여 실험을 통해 구한다[12]. [Tab. 2]는 가중치를 구하기 위한 환경설정이다.

Tab. 2. 시뮬레이션 파라미터

parameter	value
link capacity(Gb/s)	1
MTU size(byte)	1500
RTT(m/s)	1, 25, 50, 75, 100, 150, 200
packet loss rate(%)	0.01, 0.001
simulation time(second)	60

최적의 가중치를 구하기 위해 패킷손실을 0.01, 0.001%로 설정한 후 RTT를 1에서 200ms 사이로 다양하게 변경하며 성능을 측정하였다. [Fig. 4]는 패킷손실이 0.01%인 환경에서의 성능그래프이다. 그래프에서 확인 할 수 있듯이 RTT가 대략 50ms 이하 구간에서는 기본으로 설정된 가중치보다 새롭게 적용된 가중치에서 향상된 성능이 측정된다. 특히 이 구간에서는 가중치 1.120에서 가장 두드러진 성능을 보인다. RTT가 50ms 이상 150ms 이하인 구간에서는 원래 가중치에서 가장 높은 성능이 측정된다. 그리고 RTT가 150ms이상인 구간에서는 기본적인 가중치보다 새로운 가중치에서 더 높은 성능이 측정된다. 특히 이 구간에서는 가중치 1.123에서 가장 높은 성능이 측정된다.

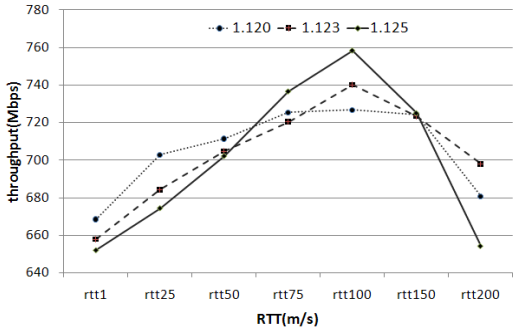

Fig. 4. 가중치에 따른 성능비교(loss rate 0.01%)

[Fig. 5]는 패킷손실이 0.001%인 환경에서의 성능그래프이다. 패킷손실이 적은 환경에서는 RTT가 증가함에 따라 전반적으로 성능도 작게 측정된다. 가중치별로 성능차이가 두드러지게 나타나지는 않지만, 이 상황에서 역시 기본 값으로 설정된 가중치보다는 향상된 성능이 측정된다.

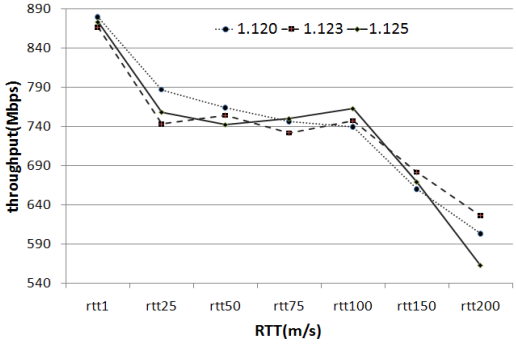

Fig. 5. 가중치에 따른 성능비교(loss rate 0.001%)

RTT가 50ms이하인 구간에서는 가중치 1.120에서 조

금이나마 향상된 성능이 측정된다. RTT가 50ms과 150ms 구간에서는 거의 비슷한 성능이 측정된다. 그리고 150ms 이상의 구간에서는 가중치 1.123에서 가장 높은 성능이 측정된다.

Modified UDT's Rate Control

```
onNak()
{
 if (RTT <= 50ms)
   snd_interval = snd_interval×1.120;

else if (RTT > 50 && RTT <=150)
   snd_interval = snd_interval×1.125;

else
   nd_interval = snd_interval×1.123;

   avg_nak_num = update;
   nak_count = 1;

}
```

Fig. 6. NAK 수신시 제안한 rate control 기법

실험을 통해 구한 최적의 가중치를 [Fig. 6]에서와 같이 세 구간에 대해 각각 적용한다. 기존 UDT에서 NAK 발생시 고정적으로 1/8로 sendInterval을 증가시킨 반면, 제안 기법에서는 실험을 통해 측정한 결과를 바탕으로 세 구간의 RTT에 대해 sendInterval을 조정하였다. 이는 패킷손실이 발생한 상황에서 RTT 구간을 고려함으로써, 각각의 구간에 대해 최적의 가중치를 적용하고 보다 향상된 성능을 측정하기 위한 것이다.

2. 혼잡예측 기반의 UDT 흐름제어 기법

기존의 UDT에서는 윈도우 크기를 한가지의 방법으로 변경시킨다. ACK를 수신하면 수신 측으로부터 전송받은 RTT와 RcvRate값을 이용하여 윈도우 크기를 변경한다. 따라서 RTT와 RcvRate의 영향을 크게 받는다. 지난 전송시간에 받은 정보를 이용하여 윈도우 크기를 변경하기 때문에 지금 현재의 네트워크 상태에 맞지 않을 수 있다. 현재의 네트워크 상황이 더 나빠진 경우, 지난 전송시간에 전달받은 정보를 이용하여 전송을 하게 되면 혼잡을 발생시키게 되거나 혼잡이 더욱 심해지게 되는 경우를 초래할 수 있다. 수신 측으로부터 정보를 전달받는데 까지는 RTT만큼의 지연이 일어나기 때문에 특히, RTT가 큰 환경일수록 그 지연 시간이 더욱 길어지게 된다. 이로 인해 혼잡을 발생시키거나 혼잡을 더욱 가중시키게 된다. 따라서 UDT의 흐름제어 기법에 혼잡을 미리 예측하여 혼잡을 사전에 방지하기 위한 기법을 도입한다.

본 논문의 기본적인 혼잡예측 기법은 TCP Vegas로부터 착안하였다. Vegas에서는 RTT의 최소값인 baseRTT를 사용한다. RTT의 최소값의 대체로 처음 연결이 설정되는 시점의 첫 번째 TCP 세그먼트에 대한 값일 경우가 대부분이다. 연결 최초의 세그먼트에 대해서 혼잡이나 NAK가 거의 발생하지 않기 때문이다. 따라서 baseRTT의 값은 뒤에 얻어지는 RTT값보다 거의 모든 경우에서 작기 때문에 적절하지 않다. 따라서 baseRTT와 RTT 값을 비교하지 않고 현재의 RTT와 이전에 얻어진 RTT 값인 prevRTT값을 사용한다[13].

알고리즘의 동작과정은 다음과 같다[14].

1) prevRTT를 구한다.

prevRTT는 이전 전송시간에 얻어진 RTT 값이다.

2) 기대전송률인 Expected를 구한다.

$$\mathrm{Exp}ected = \frac{W}{prevRTT} \tag{7}$$

기대전송률은 혼잡윈도우의 크기 W를 prevRTT로 나누어 구한다.

3) 실제전송률인 Actual을 구한다.

$$Actual = \frac{W}{RTT} \tag{8}$$

실제전송률은 혼잡윈도우의 크기 W를 현재의 RTT로 나누어 구한다. 실제전송률은 현재 전송할 수 있는 양이다.

4) 두 전송률간의 차이 nDiff를 구한다.

nDiff는 네트워크의 상태를 정량적으로 표현한 값으로 다음과 같이 구한다.

$$\begin{aligned} nDiff &= (Actual - \mathrm{Exp}ected) \\ &= \left(\frac{W}{prevRTT} - \frac{W}{RTT}\right) \end{aligned} \tag{9}$$

Vegas에서와는 달리 두 전송률의 차이에 RTT를 다시 곱하지 않는다. UDT는 매우 높은 대역폭 환경을 고려하여 제안되었기 때문에 RTT와 혼잡윈도우의 크기가 TCP에 비하여 매우 크다. 두 전송률의 차에 prevRTT를 곱하게 되면 Diff 값이 매우 커지게 되어 임계값을 적용하기 어렵다. 따라서 두 전송률의 차에 prevRTT를 곱하지 않은 nDiff값을 사용한다.

5) Diff에 따라 윈도우 크기를 조절한다.

$$W=\begin{cases} Decrease\ W, & (\text{if } nDiff > \beta) \\ W, & (\text{if } \alpha \le nDiff \le \beta) \\ Increase\ W, & (\text{if } nDiff < \alpha) \end{cases} \quad (10)$$

UDT의 흐름제어 알고리즘에 적용하면 [Fig. 7]과 같다. UDT는 RcvRate를 적용하여 윈도우 크기를 변경시킨다. RcvRate는 수신측으로부터 전달받는 정보로서, 수신측의 버퍼 크기와 수신측의 윈도우 크기 중에 작은 값이다. UDT는 ACK를 수신하는 경우 RcvRate를 이용하여 윈도우 크기를 항상 변경시킨다. 윈도우 크기의 빈번한 변경은 이용효율을 감소시키는 결과를 초래할 수 있으므로 이를 방지하기 위하여 α≤nDifff≤β에서는 현재의 윈도우 크기를 유지시키도록 한다.

Modified UDT's Flow Control

```
onAck()
{
 if (slow start)
    W= W+S;
 else
 {
    if  (α ≤ nDiff ≤ β)
       W=W;
    else if  (β < nDiff)
       W=RcvRate*0.9/(Interval+RTT)*1000000;
    else
      W=RcvRate*1.1/(Interval+RTT)*1000000;

 }
 .
 .
 .
 Modify Packet Sending Period;

 Set Maximum Transfer Rate;

 prevRTT=RTT;
}
```

Fig. 7. 혼잡예측 기반의 UDT 흐름제어 기법

알고리즘의 α≤nDifff≤β의 조건은 현재의 네트워크 상황이 안정적이라는 것을 의미한다. 따라서 현재의 윈도우 크기를 그대로 유지한다. α<nDifff 조건은 네트워크가 혼잡하지 않다는 것을 의미한다. 아직 대역폭을 더 사용할 수 있다는 것이므로 윈도우 크기를 증가시킨다. nDiff>β는 현재의 네트워크 상황이 혼잡하다는 것을 의미한다. 네트워크가 혼잡한 상태이므로 윈도우 크기를 감소시킨다. 예측된 세 구간에서 서로 다른 방식으로 윈도우 크기를 변경함으로써 좀 더 개선된 성능을 얻도록 하였다.

IV. 실험 결과

본 절에서는 제안한 기법을 실제 Testbed를 구축한 후 성능실험 결과를 확인하였다. 실험은 먼저 TCP vegas를 변경한 제안기법에 대해 살펴보고, NAK 발생 시 제안한 rate control 기법에 대해 살펴본다. 또한, 두 기법을 혼합 적용한 결과에 대해 살펴보도록 한다. 실험을 위한 환경 구성은 [Fig. 8]과 같다. 두 엔드 시스템은 1G 선으로 연결하였고, 엔드시스템은 Linux를 사용하였다. RTT와 손실률은 Linux에서 제공되는 netem[15]을 사용하여 RTT는 100ms, 손실률은 0.001%로 설정하였다.

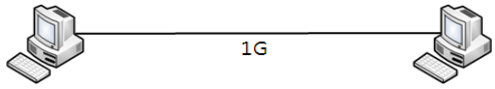

Fig. 8. 실험환경

[Fig. 9]는 시간에 따른 UDT의 전송률과 제안된 알고리즘을 적용한 UDT의 전송률을 보인다. 혼잡예측 알고리즘을 적용한 변경된 UDT의 성능이 좀 더 향상된 것을 확인할 수 있다. 특히 전송 초기의 혼잡이 예측되지 않는 구간에서 기존의 UDT보다 좀 더 공격적인 방법으로 윈도우를 증가시킴으로써 전송률을 높인다. 그 결과 기존의 UDT 보다 혼잡예측 알고리즘을 적용한 UDT가 더 높은 전송률을 보이며, 사용 가능한 대역폭을 더 효율적으로 사용할 수 있다. [Fig. 9]의 경우 임계값은 α=1.5, β=3.5로 설정하였다.

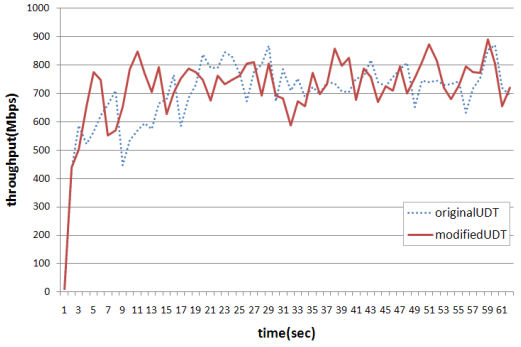

Fig. 9. 기존 UDT와 제안된 기법을 적용한 UDT의 진행시간에 따른 전송률

[Fig. 10]은 임계값 설정에 따른 평균 전송률을 나타낸다. 임계값 쌍 설정에 따라 기존의 UDT 보다 평균전송률이 증가하기도 하지만 반대로 감소하는 경우도 있음을 확인할 수 있다. 이는 임계값에 따라 혼잡 상황을 판단하여 구간을 설정하기 때문에 임계값 설정이 잘못된 경우에 잘못된 구간 설정으로 인하여 혼잡상황을 잘못 판단하기 때문에 기존의 UDT보다 오히려 전송률이 더욱 감소하게 되는 것을 의미한다. 임계값의 쌍이 (1.0, 3.0), (2.0, 3.5), (2.0, 4.0) 의 경우가 잘못된 예측을 하는 경우이다. 반대로 임계값 쌍이 (0.5, 3.0), (1.5, 3.5)인 경우는 기존의 UDT 보다 평균전송률이 증가하였음을 확인할 수 있다. 특히, (1.5, 3.5)의 경우에 뚜렷한 성능향상을 보인다. RTT가 100ms이고 손실률이 0.001%인 환경에서 혼잡예측 기반의 UDT가 기존의 UDT 보다 약 18Mbps 향상된 것을 확인 하였다.

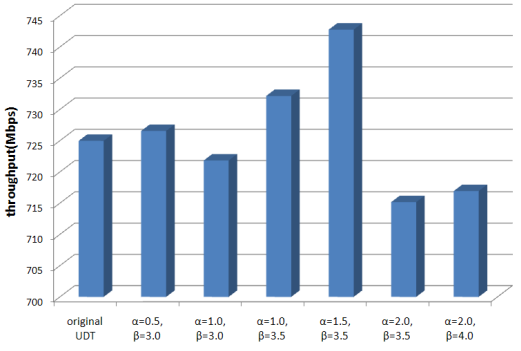

Fig. 10. 임계값 변화에 따른 평균 전송률

[Fig. 11]과 [Fig. 12]는 구간별 최적의 가중치를 적용한 후 기존 UDT와의 성능비교 실험이다. 실험은 적용된 가중치를 대표해 2개의 구간에서 기존 UDT와 비교하였다.

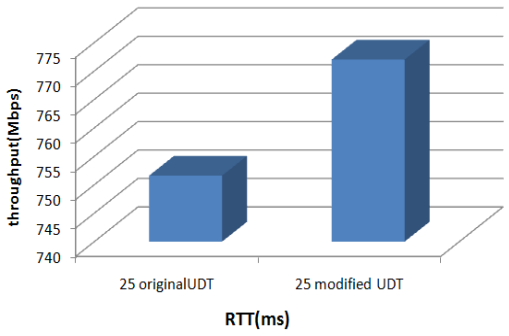

Fig. 11. 구간별 적용한 가중치와 기존 UDT의 성능비교 (25ms)

RTT가 50에서 150사이의 구간은 기존 제안된 가중치와 동일하기 때문에 여기에서는 2개의 구간에서만 실험한다. 실험은 제안한 혼잡제어 알고리즘을 성능 실험한 환경과 동일하다. [Fig. 11]에서 RTT는 netem을 사용해 25ms로 설정하였다. 이 경우 측정한 평균 성능은 기존 UDT에서보다 약 21Mbps의 성능향상을 보인다.

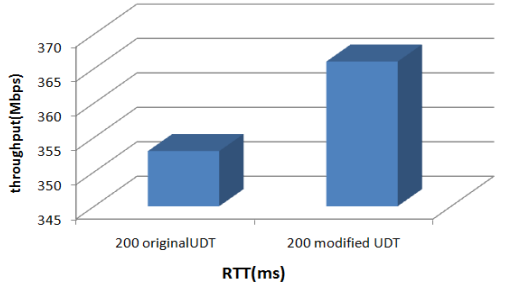

Fig. 12. 구간별 적용한 가중치와 기존 UDT의 성능비교 (200ms)

[Fig. 12]는 RTT 200ms구간에서의 성능비교 실험이다. 이 구간에서는 기존 UDT보다 약 15Mbps의 성능향상이 측정된다.

이 두 경우의 실험은 네트워크 시뮬레이터인 NS-2를

이용하여 먼저 구간에 따라서 최적의 가중치를 구한 후, 기존 UDT의 고정된 가중치를 구간에 따라 분리 적용한다. 성능실험을 통해 확인한 결과 기존 UDT보다 제안한 기법을 통해 향상된 성능이 측정되었다.

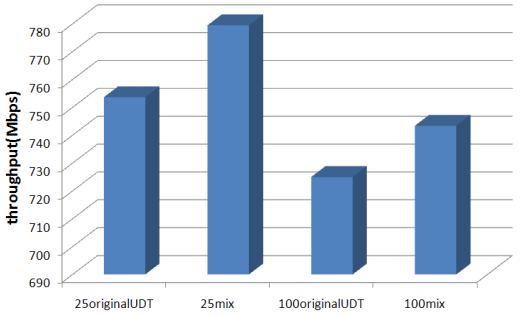

Fig. 13. 제안기법을 혼합 적용한 UDT와 기존 UDT의 성능비교

[Fig. 13]은 구간별 적용된 가중치와 혼잡예측 알고리즘을 혼합 적용한 후 실험한 그래프이다. 패킷 손실률을 0.001%로 설정한 후 RTT 구간을 25ms, 100ms에서 기존 UDT와 비교 실험하였다. RTT 100ms인 구간에서는 적용한 가중치에 의한 영향은 없을 것으로 예상된다. 하지만, 혼잡예측 알고리즘에 의해 약 19Mbps의 향상된 성능이 측정된다. RTT 25ms인 구간에서는 적용한 가중치와 혼잡예측 알고리즘에 의해 기존 UDT보다 향상된 성능을 보인다. 이 구간에서는 약 26Mbps정도의 성능향상을 확인 하였다.

V. 결 론

본 논문에서는 기존 UDT에서 NAK 수신시 고정적으로 sendInterval을 1/8만큼 증가시키는 반면 실험을 통해 구한 최적의 가중치를 세 단계의 RTT구간에 적용한다. 또한, TCP Vegas에서 제안한 혼잡 예측 기법을 UDT에 적용함으로써 향상된 흐름제어 알고리즘을 제안한다. RTT와 prevRTT를 이용하여 혼잡을 미리 예측하여 혼잡이 예상되는 구간에서는 혼잡윈도우를 감소시키고, 안정된 구간에서는 기존 혼잡윈도우를 유지한다. 그리고 나머지 구간, 즉 망상태가 양호한 구간에서는 윈도우 크기를 증가시킨다. 제안한 기법은 각각 실험을 통해 성능향상을 확인하였다. 또한, 혼합 적용한 기법 역시 기존 UDT보다 향상된 성능을 측정하였다. 제안한 방법으로 실험한 결과 성능 면에서 평균 15~30Mbps 정도의 성능향상을 보였다.

본 논문에서는 패킷손실에 따른 NAK 상황에서 RTT구간별 가중치를 변경하였지만, 향후연구에서는 가중치(sendInterval)가 성능에 미치는 영향에 대해 추가 연구가 필요하다. 또한 UDT 흐름제어에 영향을 미치는 RcvRate와 본 논문에서 제안한 알고리즘의 관계에 대해서도 연구가 필요하다.

참 고 문 헌

[1] S. Floyd, "HighSpeed TCP for Large Congestion Windows," IETF, RFC3649, 2003.

[2] T. Kelly, "Scalable TCP: Improving Performance in Highspeed Wide Area Networks," ACM Computer Communication Review, Vol.32, No.2, pp.83-91, 2003.

[3] L. Xu, K. Harfoush, and I. Rhee, "Binary Increase Congestion Control(BIC) for Fast Long-Distance Networks," IEEE INFOCOM'04, Vol.4, pp.2514-2524, 2004.

[4] E. He, J. Leigh, O. Yu, and T. A. Defanti, "Reliable Blast UDP: Predictable High Performance Bulk Data Transfer," International Conference on IEEE CLUSTER'02, pp. 317-324, 2002.

[5] M. R. Meiss, "Tsunami: A High-Speed Rate-Controlled Protocol for File Transfer," www.evl.uic.edu/eric/atpTSUNAMI.pdf/, 2009.

[6] Y. Gu and Robert L. Grossman, "UDT: UDP Based Data Transfer for High Speed Wide Area Networks," Computer Networks, Vol.51, No.7, pp.1777-1799, 2007.

[7] V. Jacobson and M. Kerels, "Congestion Avoidance and Control," ACM SIGCOMM'88, Vol.18, No.4, pp.314-319, 1988.

[8] V. Jacobson, "Modified TCP Congestion Avoidance Algorithm," LBNL Technical Report, 1990.

[9] L. Brakmo, S. O'malley, and L. Peterson, "TCP Vegas: New Techniques for Congestion Detection and Avoidance," ACM SIGCOMM'94, Vol.24, No.4, pp.24-35, 1994.

[10] Y. Gu, *"UDT: A High Performance Data Transport Protocol,"* Ph.D Thesis, Laboratory for Advanced Computing, Univ. of Illinois at Chicago, 2005.

[11] 박종선, 김승해, 조기환, "UDT 환경에서 패킷 로스를 위한 효율적인 혼잡제어 방안에 관한 연구," 제4회 정보통신분야학회 합동학술대회 논문집, pp.217-220, 2010.

[12] http://www.isi.edu/nsnam/ns/

[13] 이선헌, 송병훈, 정광수, "TCP Vegas에서 공정성 향상을 위한 혼잡제어 알고리즘," 한국정보과학회논문지, 제32권, 제5호, pp.583-592, 2005.

[14] 이승아, 김승해, 조기환, "혼잡예측 기반의 UDT 흐름제어 기법," 제 34회 한국정보처리학회 추계학술대회 논문집, 제17권, 제2호, pp.1019-1022, 2010.

[15] http://swik.net/netem

저 자 소 개

박 종 선(Jong-Seon Park) 준회원

- 2009년 : 조선대학교 전자공학과 학사졸업
- 2010년 ~ 현재 : 전북대학교 컴퓨터공학과 석사과정

<관심분야> : 컴퓨터통신, 센서네트워크

이 승 아(Seung-Ah Lee) 준회원

- 2010년 : 전북대학교 컴퓨터공학과 학사졸업
- 2010년 ~ 현재 : 전북대학교 컴퓨터공학과 석사과정

<관심분야> : 컴퓨터 보안, 클라우드

김 승 해(Seung-Hae Kim) 정회원

- 1996년 : 한남대학교 정보통신공학과 학사졸업
- 2003년 : 전북대학교 대학원 정보과학과 석사졸업
- 2008년 : 전북대학교 대학원 정보보호공학과 박사졸업
- 1996년 ~ 현재 : 한국과학기술정보연구원 선임연구원

<관심분야>

무선보안, IPv6, 미래 네트워크, 네트워크 운용/관리

조 기 환(Gi-Hwan Cho) 종신회원

- 1985년 : 전남대학교 계산통계학 학사 졸업
- 1987년 : 서울대학교 계산통계학과 석사 졸업
- 1996년 : 영국 Newcastle 대학교 전산학과 박사 졸업
- 1987년 ~ 1997년 : 한국전자통신연구원 선임연구원
- 1997년 ~ 1999년 : 목포대학교 컴퓨터과학과 전임강사
- 1997년 ~ 현재 : 전북대학교 컴퓨터공학부 교수

<관심분야> : 이동컴퓨팅, 컴퓨터통신, 분산처리 시스템, 무선네트워크보안, 무선네트워크