# 혼잡예측 기반의 UDT 흐름제어 기법

이승아*, 김승해**, 조기환*
*전북대학교 전자정보공학부
**한국과학기술정보연구원 융합자원 연구실
e-mail : *{salee86, ghcho}@chonbuk.ac.kr
**shkim@kisti.re.kr

# UDT Flow Control Method based on Congestion Prediction

Seung-ah Lee*, Seunghae Kim**, Gihwan Cho*
*Division of Electronics and Information Engineering, Chonbuk University
**Department of Computing and Networking Resources, KISTI

요 약

네트워크 기술의 발전으로 이용할 수 있는 대역폭이 증가하고 있다. 그에 따라 증가한 대역폭을 효율적으로 사용하기 위한 전송 기술이 요구되고 있다. TCP Vegas는 RTT(Round Trip Time)를 이용해 혼잡을 미리 예측하여 윈도우 크기를 조절하는 혼잡 제어 알고리즘을 사용한다. UDT는 높은 대역폭과 큰 RTT 환경에서 대용량 데이터를 전송하기 위해 제공된 응용 기반의 전송 프로토콜이다. 본 논문에서는 UDT에 혼잡예측 알고리즘을 적용한 새로운 UDT의 혼잡제어 알고리즘을 제안한다. 혼잡예측을 통해 혼잡한 구간, 혼잡하지 않은 구간을 나누어 혼잡윈도우를 갱신한다. 혼잡하지 않은 구간에서 혼잡윈도우를 증가시키고 혼잡한 구간에서 혼잡윈도우를 감소시킴으로써 기존의 UDT보다 성능이 개선되었음을 확인 할 수 있다.

## 1. 서론

광 통신의 비약적인 발전으로 네트워크의 성능은 매우 빠르게 증가하고 있다. 보통 사용자에게 제공되는 네트워크의 대역폭은 수십에서 수백 Mbps 급이다. 네트워크에서 주로 사용되고 있는 TCP는 저속 네트워크에서 최적화되어 있기 때문에 네트워크의 성능을 충분히 활용 하지 못하고 있다. AIMD(Additive Increase Multiplicative Decrease)를 사용하는 TCP는 윈도우 크기를 급격하게 감소시킴으로써 혼잡 상황에서 빠르게 벗어난다.

그러나 윈도우 크기가 급격하게 변화함으로써 네트워크를 효율적으로 사용하지 못하고 있다. TCP의 성능의 문제점을 개선하고자 많은 연구들이 진행되고 있지만 아직까지 네트워크 이용 효율이 저조한 편이다. 특히, 대역폭이 수 Gbps에서 수백 Gbps에 이르는 연구망들이 개발되고 있는 현재 상황으로 보았을 때 그 효율은 더욱 감소되고 있다. 또한 연구망은 국내뿐만 아니라 국외의 다른 연구망들과 연동될 전망이다. 그렇기 때문에 큰 대역폭, 높은 RTT에서의 데이터 전송에 대한 연구가 매우 필요하다.

따라서 어플리케이션 기반의 전송기법인 UDT(UDP - based Data Transfer)를 바탕으로 높은 대역폭과 큰 RTT 환경에서의 전송기법들을 분석했다. UDT는 손실이 없는 환경에서는 매우 안정적이고 높은 성능을 보이지만 손실이 존재하는 환경에서는 그렇지 않다. 그러므로 손실이 있는 환경에서 Vegas로부터 착안한 혼잡예측 알고리즘을 UDT에 적용하여 UDT의 성능을 향상시켰다.

논문의 구성은 다음과 같다. 먼저 2장에서 TCP Reno를 개선한 TCP Vegas와 높은 대역폭과 큰 RTT에서의 전송기법을 연구한 UDT에 대해 알아본다. 3장과 4장에서는 UDT의 흐름제어(윈도우 제어)에 Vegas에서 착안한 기법을 적용해본다. 마지막으로 5장에서는 결론을 맺었다.

## 2. 관련연구

### 2.1 TCP Vegas

TCP Reno는 AIMD 혼잡 제어 알고리즘을 사용한다. 따라서 Reno는 윈도우 크기를 증가시키다가 혼잡이 발생하면 윈도우 크기를 급격히 감소시킨다[1]. 이러한 Reno의 혼잡 제어 방식은 혼잡 상황에서 빠르게 벗어날 수 있다는 장점이 있지만 전송률의 변화가 심하다. 전송률의 많은 변화로 인해 전체적인 네트워크 효율이 감소하게 된다. TCP Vegas는 Reno의 이러한 문제점들을 해결하기 위해 Brakmo와 Peterson에 의해 제안되었다[2]. Vegas는 NAK 발생 시 혼잡 제어를 하는 것이 아니라 RTT 값을 이용해 혼잡 상황을 미리 예측하는 알고리즘을 사용한다. RTT가 커지면 네트워크의 혼잡 정도가 증가된 것으로 판단하여 미리 윈도우 크기를 감소시킨다. 반대로 RTT가 작아지면 혼잡 정도가 감소된 것으로 판단하여 윈도우 크기를 증가시킨다.

Vegas의 동작과정을 서술하면 다음과 같다[2][3].

ⅰ) RTT의 최소값 *baseRTT*를 구한다.

*baseRTT*는 혼잡이 없을 때의 RTT로, 대체로 첫 번째 TCP 세그먼트에 대한 RTT값이 *baseRTT*값이 된다.

ⅱ) 기대전송률인 *Expected*를 구한다.

$$\text{Exp}ected = \frac{CwndSize}{baseRTT} \quad (식\ 1)$$

*CwndSize*는 *baseRTT*동안에 전송할 수 있는 윈도우의 크기이다.

ⅲ) 실제전송률인 *Actual*을 구한다.

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

ⅳ) 두 전송률간의 차이인 *Diff*를 구한다.

*Diff*는 네트워크의 상태를 정량적으로 표현한 값이다. *Diff*는 다음 (식 3)과 같이 구한다.

$$Diff = (\frac{CwndSize}{baseRTT} - \frac{CwndSize}{RTT}) \times baseRTT \quad (식\ 3)$$

ⅴ) *Diff*에 따라 윈도우 크기를 조절한다.

*Diff* 값으로 상태를 구분하기 위해 임계값 α와 β를 사용한다. W는 윈도우 크기이다. Vegas에서 α와 β의 기본값은 α=1, β=3이다. *Diff*의 값에 따라 윈도우 크기 $W$의 조절은 다음과 같이 이루어진다.

$$W = \begin{cases} W-1, & \text{if } Diff > \beta \\ W, & \text{if } \alpha \le Diff \le \beta \\ W+1, & \text{if } Diff < \alpha \end{cases}$$

2.2 UDT(UDP-based Data Transfer)

UDT는 높은 대역폭, 큰 RTT를 갖는 네트워크 환경에서 대용량 데이터 전송을 위해 Yunhong Gu가 제안한 기법이다[4]. UDT는 UDP 기반의 데이터 전송을 수행하고 UDP의 단점인 신뢰성을 보장하기위해 일정 시간마다 ACK를 수신하는 기법을 사용하는 어플리케이션이다. UDP로는 데이터를 전송하고 일정 주기마다 ACK 또는 NAK를 수신 받아 손실된 패킷을 다시 전송하는 방법을 취하고 있다. UDT의 혼잡 제어는 Rate 제어와 Flow (Window) 제어를 함께 사용한다. Rate 제어에서는 전송 간격을 조정해 혼잡을 제어하고, Flow 제어에서는 전송 간격과 함께 윈도우 크기 조절을 함께 한다. UDT의 기본 Flow 조절 방식은 다음 (그림 1)과 같다.

슬로우 스타트(slow start) 동안 ACK를 전송받았을 경우 윈도우의 크기는 ACK를 전송 받을 때까지의 기간 동안 보낸 세그먼트의 총 개수 S만큼 증가된다. S의 개수는 Sequence 번호의 차로 구할 수 있다. UDT는 매 데이터 전송마다 ACK와 NAK를 주고받는 것이 아니라 일정 시간(SYN time) 마다 한번 씩 전송 받기 때문에 세그먼트의 개수는 TCP에서처럼 1이 아니다. NAK를 전송받았을 때는 전송 주기를 조절하기 때문에 혼잡윈도우 크기를 직접적으로 감소시키거나 증가시키지 않는다. 슬로우 스타트가 아닌 경우 ACK를 받으면 RcvRate(ReceiveRate)를 이용해 혼잡윈도우의 크기를 증가시키고 감소시킨다. RcvRate는 버퍼와 혼잡윈도우 크기 중의 최소값으로 수신측에서 결정된다. 따라서 ACK를 수신 받을 때마다 RcvRate에 따라 윈도우의 크기가 계속 변하게 된다. 제안하는 기법에서는 슬로우 스타트가 아닌 부분에서 전송률과 RTT를 적용시켜 기존 UDT의 Flow 제어를 개선한다.

***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;
}
```

(그림 1) UDT 흐름 제어

## 3. 제안기법

기존의 UDT에서 윈도우의 크기는 ACK를 수신 받을 때마다 RTT와 RcvRate를 이용해 윈도우의 크기를 다시 결정한다. 따라서 RTT와 전송률의 영향을 많이 받는다. RTT와 전송률이 빈번하게 변화하기 때문에 윈도우 크기 역시 빈번하게 변화한다. 윈도우 크기의 빈번한 변화는 전체적인 네트워크의 성능을 감소시킨다.

UDT는 전체적인 네크워크의 상황에 대한 정보를 수신측으로부터 받는다. 수신측의 제어 관련 부분에서 계산한 값을 송신 측으로 전송해 준다. 송신측은 전송받은 값을 다음 전송에 적용한다. 이러한 방식은 네트워크 혼잡이 일어났을 경우 혼잡 발생에 대해 뒤늦게 대처하게 한다. RTT 시간만큼 정보를 늦게 받아 볼 뿐 아니라 네트워크 상황에 대한 값을 계산하는 시간만큼 더 늦게 대처하게 된다. 즉, RTT 시간에 처리시간을 더한 만큼 혼잡에 대한 대처가 늦어지게 되는 것이다. 이러한 경우 RTT가 크면 클수록 네트워크 상황에 더욱 늦게 반응하게 되어 혼잡 상황을 늦게 벗어날 뿐만 아니라 혼잡을 더욱 가중시키는 상황을 야기하게 된다. 따라서 UDT의 기본 흐름제어 기법에 네트워크 혼잡을 미리 예측하여 혼잡을 미리 예방하기위한 기법을 도입한다.

기본적인 기법은 TCP Vegas에서 착안하였다. Vegas의 *baseRTT*는 RTT값들 중 가장 작은 값이 선정 된다. 대체로 네트워크에서 가장 작은 RTT 값은 통신이 시작 될 때 얻어지는 RTT값이다. 통신이 시작될 때는 데이터의 혼잡 상황이나 NAK가 거의 발생하지 않기 때문이다. 이렇게 얻어진 *baseRTT* 값은 뒤에 얻어지는 RTT의 값들보다 거의 모든 경우에 작기 때문에 적절하지 않다. 따라서 *baseRTT*와 비교하는 대신 현재의 RTT 시간 직전에 측정된 RTT와 현재의 RTT를 비교한다[5]. 이전에 얻은 RTT 값은 *prevRTT*라고 한다.

알고리즘의 동작은 다음과 같다.

ⅰ) *prevRTT*를 구한다.

*prevRTT*는 바로 이전의 전송시간에 얻어진 RTT이다.

ⅱ) 기대전송률인 *Expected*를 구한다.

$$\text{Exp}ected = \frac{CwndSize}{prevRTT} \quad \text{(식 5)}$$

*CwndSize*는 *prevRTT*동안에 전송할 수 있는 윈도우의 크기이다.

ⅲ) 실제전송률인 *Actual*을 구한다.

$$Actual = \frac{CwndSize}{RTT} \quad \text{(식 6)}$$

*CwndSize*를 현재의 *RTT*로 나누어 현재 전송할 수 있는 양을 얻는다.

ⅳ) 두 전송률간의 차이인 *nDiff*를 구한다.

*nDiff*는 네트워크의 상태를 정량적으로 표현한 값이다. *nDiff*는 다음 (식 7)과 같이 구한다.

$$nDiff = \left( \frac{CwndSize}{prevRTT} - \frac{CwndSize}{RTT} \right) \quad \text{(식 7)}$$

Vegas에서와 달리 두 전송률 차이에 *prevRTT*값을 다시 곱하지 않는다. UDT는 매우 높은 대역폭과 RTT 환경을 고려하여 설계되었다. 따라서 UDT는 TCP에 비해 혼잡윈도우의 크기와 RTT가 크다. *prevRTT*를 곱하게 되면 *nDiff* 값이 커져서 임계값을 적용하기 어렵다. 따라서 임계값 선정을 용이하게 하기 위해 *prevRTT*를 곱하기 전 값인 *nDiff*를 사용한다. 임계값 α는 하한값이고 β는 상한값이 된다.

ⅴ) *nDiff*에 따라 윈도우 크기를 조절한다.

$$CwndSize = \begin{cases} \text{mod}ify\ CwndSize & (\text{if } nDiff > \beta) \\ CwndSize & (\text{if } \alpha \le nDiff \le \beta) \\ \text{mod}ify\ CwndSize & (\text{if } nDiff < \alpha) \end{cases}$$

UDT는 윈도우 사이즈에 RcvRate를 적용한다. 따라서 단순히 윈도우 값만을 가지고 증가시키거나 감소시키지 않는다. 그러나 ACK를 받은 경우에는 RcvRate를 적용해 윈도우 크기를 항상 변경시킨다. 제안하는 기법에서는 RcvRate에 따른 윈도우 크기의 급격한 변화를 방지하기 위하여 *nDiff* 값이 임계값 사이에 존재하는 경우 현재의 윈도우 크기를 유지시킨다. 그 밖의 경우는 윈도우 크기를 갱신한다. 여기서 *CwndSize*는 식의 간략화를 위하여 W로 표현한다. 변경된 알고리즘은 다음 (그림 2)와 같다.

***Modified UDT's Flow Control***

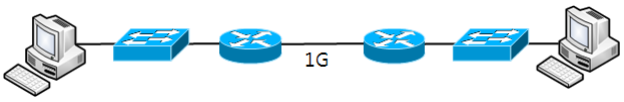
```
onAck()
{
  if (slow start)
    W= W+S;
  else
  {
    if  (α ≤ nDiff ≤ β)
      W=W;
    else if  (β < nDiff)
      W=RcvRate*0.9/(Interval+RTT)*1000000;
    else if  (α > nDiff)
      W=RcvRate*1.1/(Interval+RTT)*1000000;

  }
  .
  .
  .
  Modify Packet Sending Period;

  Set Maximum Transfer Rate;

  prevRTT=RTT;
}
```

(그림 2) 혼잡예측 기반의 UDT 흐름제어 기법

(그림 2)의 $a \le nDiff \le \beta$ 조건은 현재 네트워크의 상황이 안정적이라는 것을 의미한다. 따라서 기존의 윈도우 크기를 변경시키지 않고 그대로 유지 한다. $\beta < nDiff$의 조건은 네트워크의 상황이 혼잡한 것을 의미하므로 혼잡윈도우를 감소시키는 반면, $a > nDiff$ 조건은 네트워크가 혼잡하지 않으므로 혼잡윈도우의 크기를 증가시켜 네트워크의 이용률을 높인다. 기존의 UDT에서는 (그림 1)에서 볼 수 있듯이 else 구문에서 항상 윈도우 크기를 변경시키는 방식을 취한다. 그러나 (그림 2)의 알고리즘에서는 $a \le nDiff \le \beta$ 조건에서 현재의 윈도우 크기를 유지할 것을 제안한다. $a \le nDiff \le \beta$ 조건에서는 윈도우 크기를 유지하고 그 나머지의 경우에만 윈도우 크기를 갱신하는 기법을 도입하여 좀 더 안정적이고 개선된 성능을 얻도록 한다.

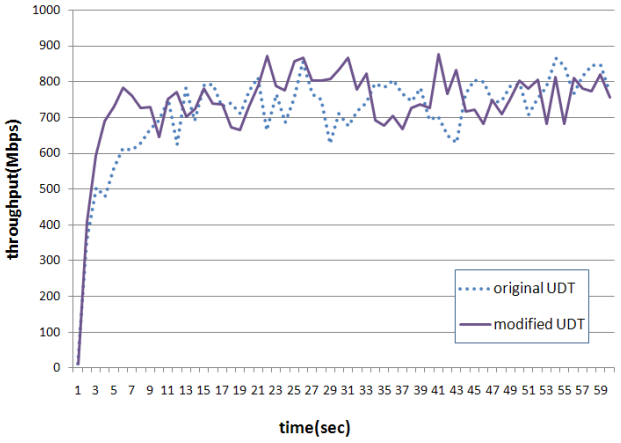
## 4. 실험 결과

본 절에서는 제안한 혼잡예측을 활용하여 혼잡예측 UDT 흐름제어 알고리즘을 실험하였다. 실험을 위한 환경 구성은 다음 (그림 3)과 같다.



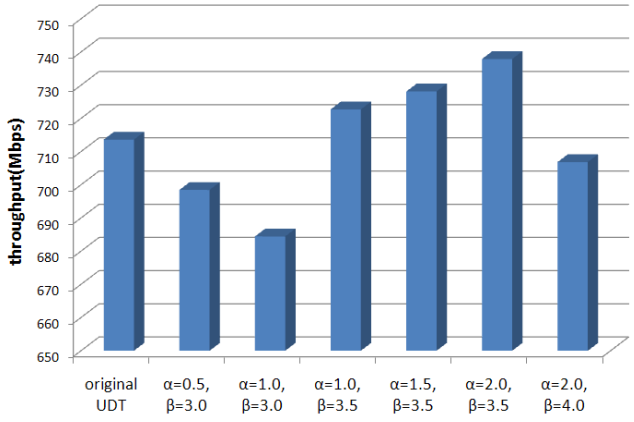

(그림 3) 실험 환경

각 스위치와 앤드 시스템은 1G로 연결되어 있고 각 시스템의 NIC(Network Interface Card)는 1G를 지원한다. 실험 환경의 RTT가 매우 작기 때문에 RTT는 netem을 사용하여 100ms로 설정하였다. 패킷 손실률은 netem을 사용하여 0.001%로 설정하였다. 각 실험은 60초씩 5번 수행한 결과의 평균을 적용하였다.




(그림 4) 기존 UDT와 제안된 기법을 적용한 UDT의 진행시간에 따른 전송률

(그림 4)는 시간에 따른 UDT의 전송률과 제안된 알고리즘을 적용한 UDT의 전송률을 보인다. 변경된 UDT의 그래프가 좀 더 향상된 것을 확인할 수 있다. 특히, 전송이 시작된 직후의 증가구간 동안의 전송률이 기존의 UDT보다 높다. 이는 혼잡이 예측 되지 않는 구간에서 좀 더 공격적으로 혼잡윈도우 크기를 증가시키기 때문이다. 혼잡이 예측되지 않는 시점에서 윈도우 크기를 증가시킴으로써 사용가능한 대역폭을 더 효율적으로 사용할 수 있다.




(그림 5) 임계값 변화에 따른 평균 전송률

(그림 5)는 제안된 알고리즘의 임계값 쌍을 변화 시켜 실험한 결과이다. 임계값 쌍은 여러 번의 실험을 통해 얻은 값이다. 실험 결과 중 (α, β)의 쌍이 (2.0, 3.5) 일 경우의 성능이 가장 좋은 것을 확인 할 수 있다. 임계값 쌍 (0.5, 3.0), (1.0, 3.0), (2.0, 4.0)의 경우는 기존 UDT 보다 성능이 저조하다. 이는 임계값 값의 설정에 따른 잘못된 구간 설정으로 잘못된 판단을 할 경우 윈도우 크기를 잘못 변경하게 되어 오히려 전송률이 떨어질 수 있다는 것을 의미한다. 실험 결과를 통해 제안된 알고리즘의 임계값 (α, β)의 값이 (2.0, 3.5)일 때 전송률이 기존의 UDT 보다 향상된 것을 확인할 수 있다. 손실률이 0.001%이고 RTT가 100ms 인 환경에서 혼잡예측 기반 UDT의 평균 전송률이 기존의 UDT 보다 약 24Mbps 향상된 것을 확인 할 수 있다.

## 5. 결론 및 향후 과제

본 연구에서는 TCP Vegas에서 제안된 혼잡예측을 통한 혼잡 제어 방식을 UDT의 혼잡 제어 방식에 적용하는 기법을 제안하였다. RTT와 *prevRTT*를 이용한 차이로 혼잡을 예측하여 혼잡이 예상되는 구간에서는 혼잡윈도우를 감소시키고, 안정된 구간에서는 기존 혼잡윈도우를 유지시켰다. 또, 나머지 경우에 혼잡윈도우를 증가시켰다. 실험결과를 통해 제안된 알고리즘을 적용한 UDT의 성능이 향상된 것을 검증하였다.

본 논문에서의 실험은 RTT가 100ms 손실률이 0.001%인 환경에서의 실험 결과만을 보았지만, 향후 연구에서는 다른 대역폭 환경을 고려한 연구가 필요하다. 또한 윈도우 크기에 영향을 미치는 UDT의 Receive Rate와 제안된 알고리즘의 관계에 대한 추가적인 연구가 필요하다.

## 참고문헌

[1] V. Jacobson and M. Kerels, “Congestion Avoidance and Control,” ACM SIGCOMM′88, pp. 314-319, Aug. 1988.

[2] V. Jacobson, “Modified TCP Congestion Avoidance Algorithm,” LBNL Technical Report, Apr. 1990.

[3] L. Brakmo, S. O′malley, and L. Peterson “TCP Vegas: New Techniques for Congestion Detection and Avoidance,” ACM SIGCOMM′94, pp. 24-35, Aug. 1994.

[4] Y. Gu and R. L. Grossman, “UDT: UDP-based Data Transfer for High-speed Wide Area Networks,” Computer Networks, Vol. 51, Issue 7, pp. 1777-1799, May. 2007.

[5] 이선헌, 송병훈, 정광수, “TCP Vegas에서 공정성 향상을 위한 혼잡제어 알고리즘,” 한국정보과학회논문지, 제 32권 5호, 2005.