유비쿼터스 컴퓨팅 환경에서 단말의 이동성을 지원하기 위한 DCCP기반의 혼잡 제어 정책

이태훈, 김성민, 박시용, 정기동
부산대학교 컴퓨터공학과
{withsoul, morethannow, sypark, kdchung}@pusan.ac.kr

DCCP based Congestion Control Scheme to support Mobility of Devices on Ubiquitous Computing Environment

Tae-Hoon Lee, Sung-Min Kim, Si-Yong Park, Ki-Dong Chung
Dept of Computer Engineering, Pusan National University

요 약

본 논문에서는 유비쿼터스 컴퓨팅 환경에서 단말들의 이동성에 따른 적응적인 혼잡제어 기법을 제안한다. 제안하는 혼잡 제어 기법은 무선환경의 특성에 따른 비트 에러와 혼잡에 따른 패킷 손실을 구별하기 위해서 역 혼잡 회피 단계를 도입하였다. 그리고 혼잡이 발생 했을 때, 대역폭 낭비를 최소화 할 수 있는 슬로우 스톱 단계를 추가하였다. 본 논문에서 제안하는 혼잡 제어 정책은 DCCP(Datagram Congestion Control Protocol)을 기반으로 설계하였고, 리눅스 커널 버전 2.4.19에서 구현하였다.

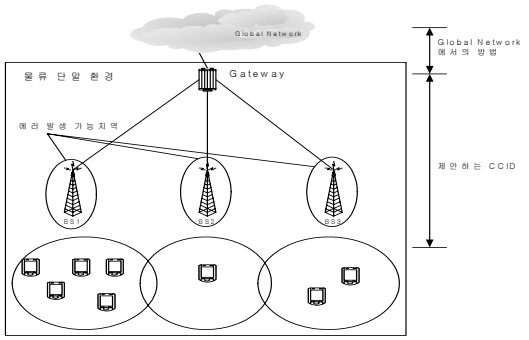
제안된 혼잡 제어 정책은 기존의 혼잡 제어 정책보다 적응성 있게 혼잡 상태를 제어하며, 실험 결과 무선에서뿐만 아니라 유선에서도 우수한 대역폭 이용률을 보였다.

1. 서론

현재의 컴퓨팅 환경은 이질적인 특성을 가진 다양한 네트워크가 유무선의 혼합 형태로 존재하고, 여러 종류의 컴퓨팅 능력을 가진 단말 장치들이 공존한다. 특히 유비쿼터스 네트워크에서는 여러 단말들의 이동성을 극대화하기 위해서 많은 무선 접속망이 포함되어 있다. 이와 같은 환경에서 대용량의 데이터를 효율적으로 전송하기 위해서는 소형화, 경량화된 편재형 단말 장치들을 고려해야 한다. 특히 편재형 단말 장치들은 기존은 단말 장치와 달리 H/W 자원이 매우 제한되어 있다. 그러나 TCP 프로토콜은 혼잡 제어와 흐름 제어, 그리고 재전송 기법들과 같은 많은 기능들로 인해 두터운 S/W 프로토콜 스택으로 구성되어있다. 그리고 UDP 프로토콜은 흐름 제어 등의 기능이 없는 경량화된 프로토콜인 반면에 신뢰성을 보장하지 못한다. 그래서 UDP에 약간의 신뢰성을 부여하기 위한 목적으로 TCP-like와 TCP-friendly와 같은 응용 계층의 혼잡 제어 기법들[1,2,4]이 제안되었으며 IETE(Internet Engineering Task Force)에서는 DCCP (Datagram Congestion Control Protocol)를 표준으로 제정 중에 있다[3,5].

본 논문에서는 DCCP를 기반으로 무선 접속망에 적합한 혼잡 제어 정책을 제안한다.

2. 무선접속망에 적합한 혼잡제어 정책



[그림 1] 혼잡제어 정책을 위한 네트워크 구조

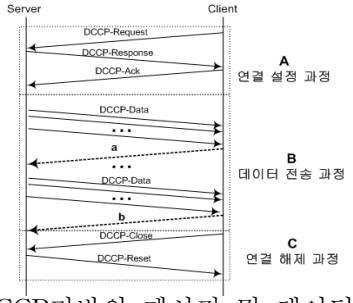
유비쿼터스 컴퓨팅 환경에서는 그림 1의 구조와 같이 무선단말과 유무선을 연결하는 게이트웨이 사이에 베이스

본 연구는 교육부에서 주관하는 “차세대 물류 IT 기술 연구 사업단”에 의해 지원받은 연구임.

스테이션만 존재하는 한 홉의 구조라고 가정한다. 중간노드에서는 라우팅 과정이 없고 모든 단말들이 이동할 수 있다. 이와 같은 환경에서는 기존의 유선망과는 달리 단말등의 접속점인 베이스 스테이션의 한계를 초과하는 무선 단말들이 연결할 때 혼잡이 발생한다.

또한 무선 네트워크에서 고려해야하는 또 다른 문제점은 비트에러에 의한 혼잡오류 현상이다. 만약 비테에서로 인하여 패킷이 손실되더라도 기존의 혼잡제어기법들은 이러한 패킷손실을 혼잡으로 오인한다.

무선망의 경우 게이트웨이를 중심으로 연결된 단말들이 하나의 홉(베이스스테이션)만을 거치기 때문에 중간 라우터의 패킷 손실은 무의미하며 모든 단말들이 게이트웨이의 통제 하에 있기 때문에 게이트웨이를 기반으로 효율적인 혼잡 제어를 수행 할 수 있다.



[그림 2] DCCP기반의 메시지 및 데이터 흐름

[그림 2]는 본 논문에서 제안하는 혼잡 제어 정책을 수행하기 위한 전체적인 흐름을 보인다.

A 구간은 데이터 연결 과정이고 B구간은 데이터 전송 과정, 그리고 C구간은 연결 해제 과정이다.

B구간에서는 전송 윈도우 크기만큼의 데이터 패킷들을 전송하고 이에 따른 Ack 벡터를 포함한 응답 패킷(그림3의 a,b)을 전송하는 과정을 반복적으로 수행한다. 그리고 서버는 혼잡제어의 결과에 따라서 전송 윈도우 크기를 조절하고 조절된 전송 윈도우 크기에 적합한 Ack 벡터의 크기 변화를 클라이언트에게 요구한다. 이때 Ack 벡터의 크기 변화 요구는 Change 옵션을 통해서 이루어진다. Ack 벡터는 송신측에서 보낸 패킷의 손실 유무를 나타내는 정보로서 본 논문에서는 전송 윈도우 사이즈를 Ack벡터의 사이즈로 정한다.

2.1 무선 환경에서 혼잡 제어 기법 설계 시 고려사항

유비쿼터스 네트워크의 무선 접속망에서 혼잡 제어를 수행하기 위하여 고려하여야 한 사항들은 다음과 같다.

- 무선의 특성 때문에 발생하는 비트에러에 따른 패킷손실과 실제 혼잡에 따른 패킷손실을 구분해야 한다.
- 기존의 프로토콜에서 혼잡 제어 시 전송률을 급격하게 하락시킴으로 인하여 발생하는 대역폭 낭비를 해결해야 한다.
- 패킷손실은 여러 단말이 동시에 접속해서 베이스스테이션의 처리 용량을 초과할 경우에도 발생한다. 이와 같은 경우 전송률을 적응적으로 조절해야 한다.
- 현재 서비스 중인 단말의 수와 각 단말들의 최대 전송 윈도우 사이즈 변화, 게이트웨이가 사용할 수 있는 전송 윈도우의 크기를 알 수 있고, 또한 모든 세션에 대해서 제어가 가능하다. 이를 기반으로 각 단말과 게이트웨이 사이의 전송량을 공평하게 조절할 수 있다.

2.2 임계값의 설정

본 논문에서 제안하는 혼잡 제어 정책은 TCP와 유사하게 슬로우 스타트 임계치와 최대 혼잡 윈도우 임계치를 가지며, 서비스중인 단말의 수와 에러의 상황에 따라 임계치를 조절한다. [그림 1]과 같은 구조에서 모든 세션들이 게이트웨이를 통하여 접속하기 때문에 게이트웨이를 통하여 임계치를 설정하는 것이 가능하다.

기호	의미
i	세션의 인덱스
N	게이트웨이를 통해서 서비스중인 모든 세션 수
$T^i_{\max}$	세션 i의 최대 혼잡 윈도우 임계치;
$W^G_{\max}$	게이트웨이의 최대 가용윈도우 크기;
W^i_{recv}	세션 i의 최대 수신 윈도우 크기;
T^i_{ss}	세션 i의 슬로우 스타트 임계치;
P^i_{loss}	세션 i의 패킷 손실 정도;
W^i_{inc}	세션 i의 증가된 혼잡 윈도우 크기;
W^i_{prev}	세션 i의 이전 혼잡 윈도우 크기;

먼저 세션 i의 최대 전송 윈도우 임계치(T^i_{Max})는 (1)와 같이 구한다.

$$T^i_{Max} = \min\ (\ W^G_{Max}/N,\ W^i_{Available}\ ) \qquad (1)$$

(1)에서는 게이트웨이가 세션 i에게 할당할 수 있는 최대 윈도우 크기와 세션 i의 수신 윈도우 크기 중에서 최소값을 최대 전송 윈도우 크기로 설정한다. 그리고 T^i_{Max}는 새로운 세션이 추가되거나 종료 될 때마다 N이 변화하기 때문에 (1)를 재계산하여 새로운 T^i_{Max}를 할당한다.

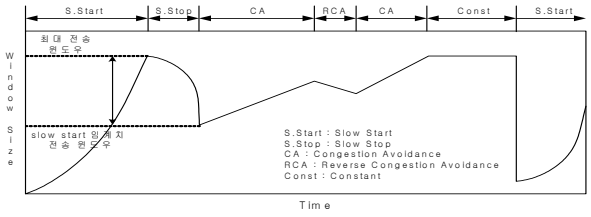
$$T^i_{SS} = \begin{cases} T^i_{Max}, & \sum_{i=1}^{N} T^i_{Max} < W^G_{Max} \\ (1-\frac{1}{N+1})\times T^i_{Max}, & \sum_{i=1}^{N} T^i_{Max} \geq W^G_{Max} \end{cases} \qquad (2)$$

(2)에서는 T^i_{SS}는 게이트웨이의 가용 윈도우에 여유가 있을 때는 T^i_{Max}가 할당되고 만약 가용 윈도우의 여유가 없을 때는 $(1-\frac{1}{N+1})\times T^i_{Max}$가 할당 된다. 이는 새로운 세션이 들어 올 여유 폭을 미리 남겨두어서 혼잡을 사

전에 방지하기 위한 것이다. 그리고 새로운 세션이 추가되면 (2)을 재계산하여 새로운 T^{i}_{SS}를 구한다. 만약 혼잡에 의해서 패킷 손실이 발생하면 T^{i}_{SS}는 (3)를 이용하여 새롭게 구한다. 그리고 T^{i}_{Max}와 T^{i}_{SS}는 게이트웨이에 의해서 결정된다.

$$T^{i}_{SS} = \frac{T^{i}_{Max}}{2} \quad (3)$$

3. 적응적인 혼잡제어 정책



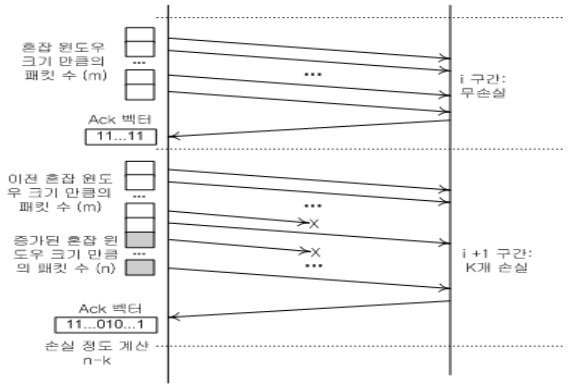
[그림 3] 무선 접속망 혼잡 제어 5 단계

기존의 TCP나 DCCP의 CCID 2, CCID 3은 슬로우 스타트, 혼잡 회피, Constant 단계와 같은 3단계 혼잡 제어 정책을 사용한다. 본 논문에서 제안하는 혼잡 제어 정책은 위의 3가지 단계 외에 비트 에러와 혼잡 상황에 따른 패킷 손실을 발견하기 위한 역 혼잡 회피 단계(RCA: Reverse Congestion Avoidance)와 약한 혼잡이 발생되었을 경우 낭비되는 대역폭을 방지하기 위한 슬로우 스톱(Slow Stop) 단계로 구성된다.

[그림 3]은 본 논문에서 제안하는 혼잡 제어 정책의 각각의 단계를 설명하고 있다. [그림 3]의 두 번째 구간에서 보이는 S.Stop은 슬로우 스톱 단계로서 혼잡 윈도우를 지수적으로 감소시키는 것을 보인다. 그리고 네 번째 구간의 RCA는 역 혼잡 회피 구간으로 혼잡 윈도우를 선형적으로 감소시키는 것을 보인다.

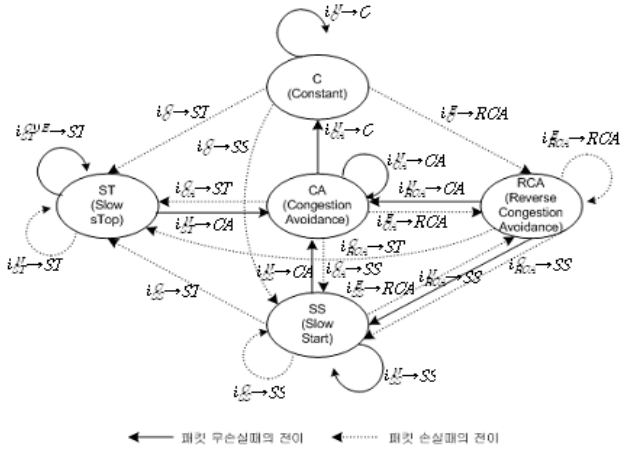
혼잡의 정도를 확인하기 위한 패킷의 손실 정도(P^{i}_{loss})를 측정하기 위해서 (4)를 이용한다.

$$P^{i}_{loss} = W^{i}_{inc} - AV^{i}_{loss} \quad (4)$$



[그림 4] 패킷 손실 정도 측정 과정

i구간에서 패킷 손실이 없었는데 i+1구간에서 k개의 패킷 손실이 발생하였다. 이는 i구간에서 무사히 전송된 m개 packet을 제외한 n개의 증가한 윈도우 패킷 때문에 발생했을 가능성이 크다. 그러므로 본 논문에서 새롭게 정의하는 패킷 손실 정도는 증가된 n개의 패킷에서 손실된 k개의 손실 패킷 개수를 감한 값으로 정의한다.



[그림 5] 혼잡 제어 정책의 상태 전이도

[그림 5]는 각 단계에서의 전이과정을 나타낸다.

a. 슬로우 스타트(SS)

혼잡 윈도우가 초기값부터 슬로우 스타트임계값까지 지수적으로 증가하는 단계이다.

패킷손실 발생 시, P^{i}_{loss}가 $\frac{W^{i}_{inc}}{2}$보다 작다면 역 혼잡 회피단계로, W^{i}_{inc}와 $\frac{W^{i}_{inc}}{2}$사이라면 슬로우 스톱단계로 가고, W^{i}_{inc}보다 크다면 슬로우 스타트단계로 전이한다.

b. 혼잡 회피(CA)

슬로우 스타트 임계값 이상부터 최대 전송 윈도우 값까지 선형적으로 증가하는 단계이다.

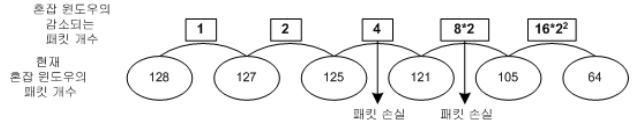
패킷손실 발생 시, P^{i}_{loss}가 T^{i}_{ss}보다 크면 슬로우 스타트 단계로, 2와 T^{i}_{ss}사이라면 슬로우 스톱단계로, 1이면 역 혼잡 회피 단계로 전이한다.

c. Constant(C)

CA 단계를 거쳐서 최대 전송 윈도우 값에 도달하면 더 이상 증가하지 않는 단계이다.

패킷 손실에 따른 혼잡 제어는 혼잡 회피 단계의 혼잡 제어 정책과 동일하다. 그 이유는 최대 혼잡 윈도우 임계치가 없다면 혼잡 회피 단계가 끝없이 진행될 것이므로 Constant 단계도 혼잡 회피 단계의 연속으로 가정하고 혼잡 회피 단계의 혼잡 제어 정책을 따른다.

d. 슬로우 스톱(ST)



[그림 6]슬로우 스톱 단계에서 패킷 손실에 따른 처리 과정

슬로우 스톱 단계에서는 지수적으로 혼잡 윈도우를 슬로우 스타트 임계치까지 줄여줌으로써 대역폭 낭비를 줄여준다.

슬로우 스톱 단계는 패킷 손실이후 반으로 줄이게 되는 일반적인 혼잡 제어 방법에 비해 느린 혼잡 제어가 제공될 수 있고, 슬로우 스톱단계에서 계속적으로 혼잡이 발생하여 정상적인 서비스를 방해할 수 있다. 따라서 슬로우 스톱 단계에서의 혼잡 발생은 에러 발생 횟수에 따라 2의 승수를 곱한 크기로 혼잡 윈도우 크기를 줄이게 되어 심한 혼잡 상황일 경우 일반적인 혼잡 제어 방법과 큰 차이 없이 현재 혼잡 윈도우 크기의 반으로 줄일 수 있다. [그림 6]는 혼잡윈도우가 줄어드는 과정을 보여주고 있다.

다음 단계는 슬로우 스톱단계로 전이되기 이전 단계로 전이된다.

e. 역 혼잡 회피(RCA)

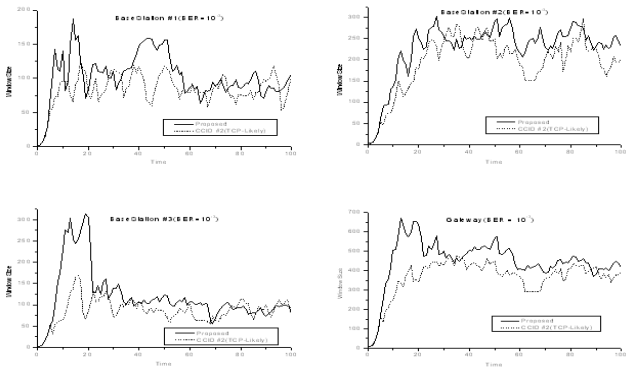
현재 전송되는 윈도우 크기를 하나씩 줄여가면서, 실제 혼잡 상황인지 단순한 비트 에러인지를 판단한다.

패킷손실 발생 시, P^{i}_{loss}가 1이면 RCA단계로, P^{i}_{loss}가 RCA 이전 단계에서 손실된 패킷의 수보다 작으면 ST단계로, 크면 슬로우 스타트단계로 가게 된다. 이 단계에서 패킷손실이 발생하지 않으면 역 혼잡 회피 이전 단계로 복귀한다.

4. 실험 및 성능 평가

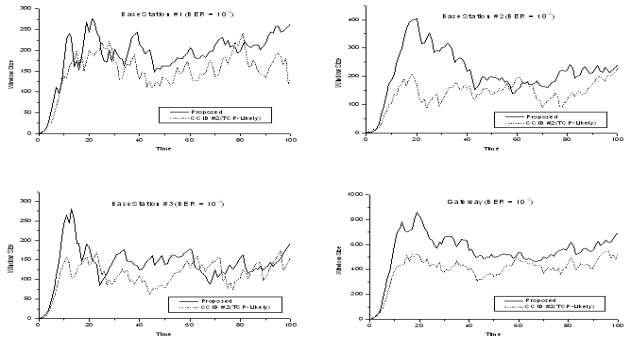
본 논문에서 제안한 혼잡 제어 정책에 대하여 시뮬레이션으로 성능을 평가하였다. 무선 실험망의 구조는 [그림 1]과 같다.

비트 에러율을 10^{-2}로 조절하여 혼잡 제어 성능을 확인하였고, 유선망에서의 성능도 측정하였다.



[그림 7]TCP-Like와 Throughput비교 (BER= 10^{-2})

[그림 7]에서 볼 수 있듯이, 본 논문에서 제안하는 혼잡 제어 정책은 TCP-Like인 CCID 2 혼잡 제어 정책에 비해서 훨씬 높은 최대 대역폭 활용도를 보여주고 있다. 이는 역 혼잡 회피 단계를 통해서 비트 에러와 혼잡에 따른 에러 상황을 구분해 내는 능력을 가졌기 때문이며, TCP-Like 혼잡 제어 정책에 비해 효율적인 혼잡 제어가 가능하다.



[그림 8] TCP-Like와 Throughput 비교 (유선 환경)

[그림 8]에서 알 수 있는 것과 같이 본 논문에서 제안하는 혼잡 제어 정책이 유선망에서도 성능이 CCID 2보다 우수하다. 먼저 혼잡의 상황을 동일하게 판단하지 않고 패킷 손실에 따라서 다양한 상황의 혼잡으로 가정하여 처리하기 때문이며 혼잡 제어 정책의 여러 단계 중에서 슬로우 스톱 단계와 RCA단계는 일반적인 유선 네트워크에서도 적용 가능하다.

앞의 실험 결과를 통해서 본 논문에서 제안한 혼잡 제어 정책이 TCP-Like 혼잡 제어 정책에 비해 훨씬 나은 성능을 보인다.

7. 결론

본 논문에서는 기존의 3단계 혼잡 제어 정책(슬로우 스타트, 혼잡 회피, Constant)에 무선망의 특성 때문에 발생하는 비트 에러를 기존의 혼잡 상태와 구분하기 위한 역 혼잡 회피 (RCA: Reverse Congestion Avoidance) 단계와 혼잡 제어에 따른 혼잡 윈도우 감소 시 발생하는 대역폭의 낭비를 줄이기 위한 슬로우 스톱 단계를 추가하였다. 그리고 본 논문에서 제안하는 혼잡 제어 정책은 패킷의 손실 정도에 따라서 비트 에러와 혼잡을 구분하며 다양하고 세분화 된 혼잡 제어 정책의 세부 단계 적용을 가능하게 한다.

참고문헌

[1] S. Floyd, M. Handley, J. Padhye, and J. Widmer, "TCP Friendly Rate Control (TFRC): Protocol Specification," Internet-Draft draft-ietf-tsvwg-tfrc-02.txt, October 2002.

[2]Sally Floyd, Eddie Kohler, "Profile for DCCP Congestion Control ID 2: TCP-like Congestion Control," Internet-Draft draft-ietf-dccp-ccid2-08.txt , November 2004.

[3] Eddie Kohler, Mark Handley, Sally Floyd, “Datagram Congestion Control Protocol (DCCP),” Internet-Draftdraft-ietf-dccp-spec-09.txt, November 2004.

[4] Sally Floyd, Eddie Kohler, Jitendra Padhye, "Profile for DCCP Congestion Control ID 3: TFRC Congestion Control," Internet-Draft draft-ietf-dccp-ccid3-09.txt, December 2004.

[5] T. Phelan, "Datagram Congestion Control Protocol User Guide," Internet-Draft draft-ietf-dccp-user-guide-02.txt,July 2004.