유비쿼터스 환경에서 DCCP기반의 적응적 혼잡제어 정책 구현

이태훈*O 임성렬**, 정기동*
부산대학교 정보컴퓨터공학과* 부산대학교 멀티미디어 협동과정**
{withsoul*O, kdchung*}@melon.cs.pusan.ac.kr syim **@wsu.ac.kr

Implementation of Adaptive Congestion Control Scheme based on DCCP on Ubiquitous Computing Environment

Taehoon Lee*O, Sungyeal Im**, Kidong Chung*
Dept. of Computer Engineering, Pusan National University*
Dept. of Multimedia, Pusan National University**

요 약

본 논문에서는 유비쿼터스 컴퓨팅 환경에서 단말들의 이동성에 따른 적응적인 혼잡제어 기법을 제안한다. 적응적 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. 적응적 혼잡 제어 정책은 DCCP(Datagram Congestion Control Protocol)을 기반으로 설계하였고, 리눅스 커널 버전 2.4.19에서 구현하였다.

적응적 혼잡 제어 정책은 기존의 혼잡 제어 정책보다 적응적으로 혼잡 상태를 제어하며, 실험 결과 무선에서 뿐만 아니라 유선에서도 우수한 대역폭 이용률을 보였다.

1. 서 론

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. 적응적인 혼잡제어 정책

기존의 TCP나 DCCP의 CCID 2, CCID 3은 슬로우 스타트, 혼잡 회피, Constant 단계와 같은 3단계 혼잡 제어 정책을 사용한다. 적응적 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.

적응적 혼잡 제어 정책은 TCP와 유사하게 슬로우 스타트 임계치와 최대 혼잡 윈도우 임계치를 가지며, 서비스중인 단말의 수와 에러의 상황에 따라 임계치를 조절한다.

이 논문은 교육인적자원부 지방연구중심대학육성사업(차세대물류IT기술연구사업단)의 지원에 의하여 연구되었음.

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;
AV^i_{loss}	Ack 벡터의 손실 패킷 개수;

[그림 1] 기호 정의

혼잡의 정도를 확인하기 위한 패킷의 손실 정도(P^i_{loss})를 측정하기 위해서 (1)를 이용한다.

$$P^i_{loss} = W^i_{inc} - AV^i_{loss} \quad (1)$$

a. 슬로우 스타트(SS)

혼잡 윈도우가 초기값부터 슬로우 스타트임계값까지 지수적으로 증가하는 단계이다.

b. 혼잡 회피(CA)

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

c. Constant(C)

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

d. 슬로우 스톱(ST)

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

ST단계는 패킷 손실이후 반으로 줄이게 되는 일반적인 혼잡 제어 방법에 비해 느린 혼잡 제어가 제공될 수 있고, ST단계에서 계속적으로 혼잡이 발생하여 정상적인 서비스를 방해할 수 있다. 따라서 ST단계에서의 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.

다음 단계는 ST단계로 전이되기 이전 단계로 전이된다.

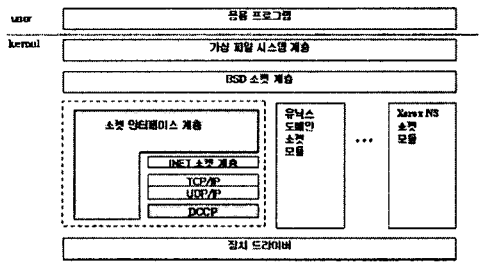
e. 역 혼잡 회피(RCA)

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 SS단계로 가게 된다. 이 단계에서 패킷손실이 발생하지 않으면 RCA 이전 단계로 복귀한다.

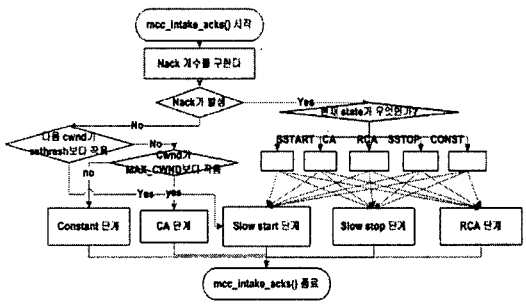
3. 혼잡 제어 프로토콜의 설계 및 구현

3.1. 전체 구조



[그림 2]리눅스의 계층적인 프로토콜 구조

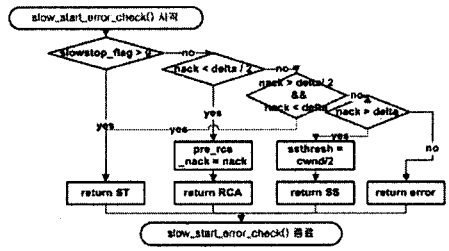
적응적인 혼잡 제어 기법은 네트워크의 프로토콜 계층 중에 전송 계층에서 제공된다. 이처럼 네트워크 전송 계층에 적응적 혼잡 제어 정책을 구현하기 위해서 [그림 2]와 같이 리눅스 커널 2.4.19 버전의 네트워크 부분을 수정 및 추가하였다. 그리고 DCCP 모듈은 기존의 PMcManux의 "Linux DCCP implementation"을 이용하여 적응적 혼잡 제어 정책을 추가하여 구현하였다.



[그림 3] mcc_intake_acks() 함수

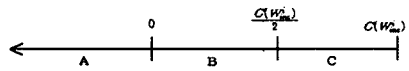
[그림 3]의 mcc_intake_acks()함수는 실제적으로 혼잡제어를 위해 window 값을 조절해주는 부분이다. packet loss가 발생하였으면 이전의 상태에 따라 알맞은 혼잡제어를 실행해준다. 혼잡제어는 이전 단계와 패킷 손실의 양에 따라서 어떤 단계로 전이될지 정해진다.

3.2 각 단계 함수의 설계 및 구현



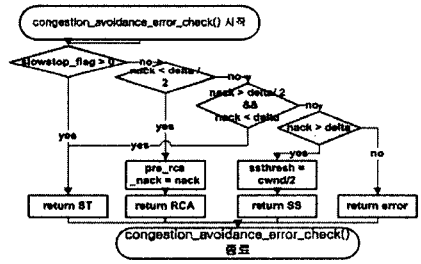
[그림 4] slow_start_error_check() 함수

SS 단계에서 nack가 발생 했다면[그림 4], 먼저 ST인지를 체크해본다. ST 단계일 경우에는 cwnd가 이전 cwnd의 절반으로 떨어질 때까지 수행해준다. ST가 아니라면, [그림6]의 범위를 이용해서 다음단계를 결정한다.



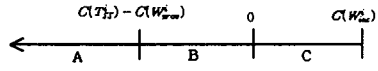
[그림 6]슬로우 스타트 단계에서의 혼잡 제어 과정

nack이 C 구간에 속하면 RCA단계로, B구간 이라면 ST 단계로, A구간일 때는 SS 단계로 전이한다.



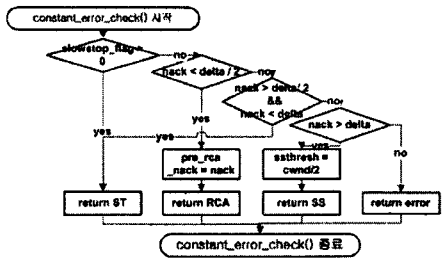
[그림 7] congsetion_avoidance_error_check() 함수

CA단계에서 패킷손실이 발생했다면[그림 7], 먼저 ST인지를 체크해본다. ST인 경우, ST 단계로, ST가 아니라면, [그림8]의 범위를 이용해서 다음단계를 결정한다.



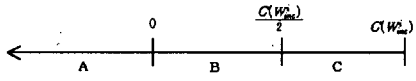
[그림 8] 혼잡 회피 단계에서의 혼잡 제어 과정

nack이 C 구간에 속하면 RCA단계로, B구간이라면 ST 단계로, A구간이라면 SS 단계로 전이한다.



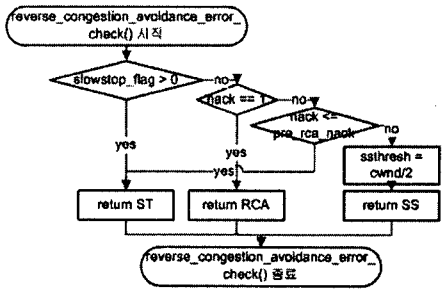
[그림 9]constant_error_check() 함수

Constant 단계일 때 nack이 발생하면[그림 9], 먼저 ST인지를 체크해보고, ST이면 ST 단계로 전이되고, 아니면 [그림 10]의 구간을 이용해서 혼잡제어를 한다.



[그림 10] Constant 단계에서의 혼잡 제어 과정

역시, nack이 C에 속하면 RCA단계로, B 구간이라면 ST 단계로, A구간에 속한다면 SS 단계로 전이한다.



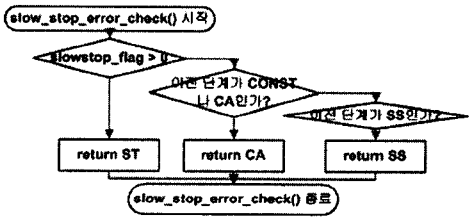
[그림11]reverse_congestion_avoidance_error_check()함수

RCA단계에서 패킷 손실이 발생하면[그림11], 먼저 ST인지를 체크해보고, ST이면 ST 단계로 전이되고, 아니면 [그림 12]를 이용해서 혼잡제어를 한다.

```
A : 현재 RCA 단계에서 손실된 총 패킷의 수
B : RCA 바로 이전 단계에서 손실 된 패킷의 수
if (A = 0)
  RCA 이전 단계 수행
else if (A = 1)
  RCA 단계 계속 수행
else if (A >= B)
  슬로우 스타트 단계 수행
else
  슬로우 스톱 단계 수행
```

[그림 12] 역 혼잡 회피 단계에서의 혼잡 제어 수행

[그림 12]에서 처럼 A와 B를 이용해서 다음 단계로 전이한다. RCA단계는 패킷 손실이 비트 에러인지 혼잡인지를 판별하기 위한 단계이기에 혼잡이 아닐 경우 RCA 이전 단계로 전이한다.

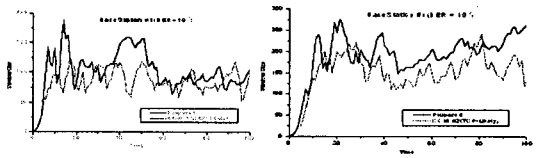


[그림 13] slow_stop_error_check() 함수

[그림13]은 현재 단계가 ST일 때 호출되는 혼잡제어 함수이다. 먼저 ST단계인지를 체크해고, ST인 경우 ST 단계로 전이된다. ST가 아닐 경우에는 이전 단계가 C나 CA일 때는 CA로, SS일 때는 SS로 전이한다.

4. 실험 및 성능 평가

적응적 혼잡 제어 정책에 대하여 시뮬레이션으로 성능을 평가하였다. 비트 에러율을 10^{-2}로 조절하여 혼잡 제어 성능을 확인하였고, 유선망에서의 성능도 측정하였다.



(a) BER = 10^{-2} (b) 유선환경

[그림 14]TCP-Like와 Throughput비교

[그림 14]에서 볼 수 있듯이, 적응적 혼잡 제어 정책은 TCP-Like인 CCID 2 혼잡 제어 정책에 비해서 훨씬 높은 최대 대역폭 활용도를 보여주고, 유선환경에서도 CCID 2보다 성능이 우수함을 보여주고 있다.

5. 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(ST: Slow sTop) 단계를 추가하였다. 그리고 적응적 혼잡 제어 정책은 패킷의 손실 정도에 따라서 비트 에러와 혼잡을 구분하며 다양하고 세분화 된 혼잡 제어 정책의 세부 단계 적용을 가능하게 한다.

향후에는 적응적 혼잡 제어 기법을 수신자 기반의 혼잡 제어 정책으로 수행 할 수 있는 절차를 추가할 것이고 또한 신뢰성 보장을 위한 다양한 기법에 대하여 연구를 실시할 것이다.

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.