유비쿼터스 컴퓨팅 환경에서 단말의 이동성을 지원하기 위한 DCCP기반의 혼잡 제어 정책

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

박 시 용 † 김 성 민 †† 이 태 훈 †† 정 기 동 †††
(Si-Yong Park) (Sung-Min Kim) (Tae-Hoon Lee) (Ki-Dong Chung)

요 약 본 논문에서는 유비쿼터스 컴퓨팅 환경에서 단말들의 이동성에 따른 네트워크의 혼잡 상태를 제어 할 수 있는 적응적인 혼잡 제어 기법을 제안한다. 제안하는 혼잡 제어 기법은 무선망의 특성상 발생할 수 있는 패킷 에러와 혼잡에 의한 패킷 손실을 구분하기 위해서 역 혼잡 회피 단계 및 혼잡 제어 시에 발생하는 비효율적인 대역폭 이용율을 최소화 할 수 있는 슬로우 스톱 단계를 기존의 혼잡 제어 정책에 새롭게 추가한다. 본 논문에서 제안하는 혼잡 제어 정책은 제 3의 전송 계층 프로토콜이라고 불리우는 DCCP(Datagram Congestion Control Protocol)를 기반으로 설계되었고 리눅스 커널 상에서 구현하였다. 제안된 혼잡 제어 정책은 기존의 혼잡 제어 정책보다 적응성 있게 혼잡 상태를 제어하며, 실험 결과 무선에서 뿐만 아니라 유선에서도 우수한 대역폭 이용율을 보였다.

키워드 : 유비쿼터스 컴퓨팅, 혼잡 제어, 모바일 컴퓨팅

Abstract In this paper, we propose a congestion control scheme to control the congestion due to the mobility of ubiquitous devices on ubiquitous computing environment. Especially, this congestion control scheme provides a reverse congestion avoidance state which can classify between packet error by features of wireless network and packet dropping by congestion. Also, it provides a slow stop state which can minimize bandwidth waste due to congestion control. The proposed congestion control scheme controls more adaptive than existing congestion control schemes. The proposed congestion control scheme is designed based on DCCP(Datagram Congestion Control Protocol) being proposed by IETF(Internet Engineering Task Force) and implemented on the Linux kernel. In simulation results, the proposed congestion control scheme provides good bandwidth throughput in wireless network as well as in wired network.

Key words : Ubiquitous Computing, Congestion Control, Mobile Computing

1. 서 론

유비쿼터스 환경은 누구나 언제, 어디서든지 어떠한 경로를 통해서라도 연결을 지속적으로 유지하여 서비스를 받을 수 있도록 하기 위해서 모든 IT 디바이스가 유무선 네트워크에 연결되어 있어야 한다. 웨어러블 컴퓨팅과 네트워크의 이동성을 극대화 시켜 어디서든지 컴퓨터를 사용할 수 있게 하는 노매딕(Nomadic)컴퓨팅, 그리고 모든 사물에 컴퓨터가 편재되는 퍼베이시브(Pervasive Computing)등이 유비쿼터스 컴퓨팅의 대표적 사례이다[1]. 이처럼 주목받고 있는 유비쿼터스 컴퓨팅 환경에는 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 포함될 전망이다. 이와 같은 환경에서 멀티미디어나 물류정보와 같은 대용량의 연속성을 가진 데이타를 효율적으로 전송하기 위해서는 경량 편재형 단말 장치들을 고려해야 한다. 특히 편재형 단말 장치들은 기존의 네트워크 단말 장치들과는 달리 H/W

· 이 논문은 교육인적자원부 지방연구중심대학육성사업(차세대물류IT기술연구사업단)의 지원에 의하여 연구되었음
† 학생회원 : 부산대학교 전자계산학과
sypark@melon.cs.pusan.ac.kr
†† 학생회원 : 부산대학교 컴퓨터공학과
withsoul@melon.cs.pusan.ac.kr
morethannow@melon.cs.pusan.ac.kr
††† 종신회원 : 부산대학교 컴퓨터공학과 교수
kdchung@pusan.ac.kr
논문접수 : 2005년 2월 28일
심사완료 : 2005년 11월 24일

자원이 매우 제한되어 있다. 그러나 현재 네트워크에서 표준처럼 사용하는 BSD 소켓 기반의 TCP 프로토콜은 혼잡 제어와 흐름 제어, 그리고 재전송 기법들과 같은 많은 기능들로 인해 두터운 S/W 프로토콜 스택으로 구성되어 있다. UDP는 흐름제어 등의 기능이 없는 경량화 된 프로토콜 스택인 반면에 신뢰성을 보장하지 못하기 때문에 어느 정도의 신뢰성을 부여하기 위한 목적으로 TCP-Friendly 혹은 TCP-Like와 같은 응용 계층의 혼잡 제어 정책들이 연구되었다[2-5]. 그리고 IETF (Internet Engineering Task Force)에서는 이러한 응용 계층의 혼잡제어 기능들을 포함하는 제3의 전송 계층 프로토콜인 DCCP(Datagram Congestion Control Protocol)를 표준으로 제정 중에 있다[4,6-9].

유비쿼터스 컴퓨팅 환경에서 멀티미디어나 물류 정보 데이타를 효율적으로 처리하기 위한 또 다른 고려사항은 단말 장치의 이동성 및 서비스의 이동성이다. 특히, 이질적인 특성을 가진 네트워크들 사이에서 서비스를 지속적으로 유지하기 위한 단말 장치의 이동성과 단말 장치들 간에 서비스의 지속성을 유지하기 위한 서비스의 이동성은 연속적인 QoS 보장을 어렵게 한다.

본 논문에서는 무선 접속망에서 단말 장치 및 서비스의 이동성으로 야기되는 혼잡의 상황을 정의하고 이러한 혼잡을 제어하기 위한 혼잡제어 정책을 제안한다. 먼저 유비쿼터스 네트워크를 그림 1과 같이 게이트웨이를 기준으로 단말들이 인터넷에 접속하기 위한 접속망과 인터넷으로 구분한다. 그리고 게이트웨이에는 인터넷과 접속망 사이의 서로 다른 프로토콜이나 데이타 트래픽을 제어하기 위한 기능, 즉 프로토콜 변환기 등이 있다고 가정한다.

접속망은 단말들의 연결 방법에 따라서 무선과 유선 접속망으로 나눈다. 유선 접속망에는 정해진 수의 단말들이 접속해 있기 때문에 인터넷이나 무선 접속망에 비하여 트래픽의 변동이 작지만 무선 접속망에는 단말들의 이동성에 때문에 실질적인 접속점인 베이스스테이션

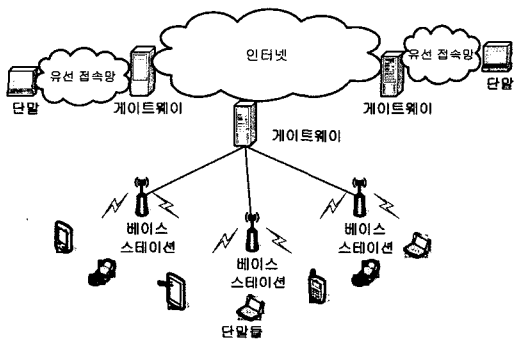

그림 1 유무선 네트워크가 혼합된 유비쿼터스 네트워크

에 접속할 수 있는 단말들의 수가 가변적이다. 그러므로 트래픽도 매우 가변적이다.

이동성을 가진 단말들은 네트워크들 간이나 혹은 동일 네트워크 내에서 접속 지점들 사이를 이동할 수 있다. 네트워크들 간의 이동뿐만 아니라 동일 네트워크 내에서의 이동의 경우에도 접속점의 네트워크 상태에 따라 혼잡의 상태가 달라진다. 서비스 이동성의 경우에서도 유선에서 무선으로 단말이 이동할 경우에는 무선 접속망의 트래픽이 증가한다. 이러한 현상은 하나 이상의 중간노드 즉 라우터 상에서 데이타그램의 과부하로 인하여 패킷 손실이 발생하는 유선 네트워크의 혼잡과는 다른 특성이다. 그러나 유비쿼터스 네트워크의 무선 접속망에서는 기존의 유선 혼잡과는 달리 단말들이 이동하기 때문에 하나의 베이스스테이션에 처리 용량을 초과하는 모바일 단말들이 존재하여 많은 수의 패킷들이 베이스스테이션의 처리 용량 초과로 인해 손실될 것이다. 이러한 이유 때문에 유선 네트워크의 혼잡은 하나의 중간 노드를 통과하는 것이 아니라 여러 개의 중간 노드들을 통과하기 때문에 패킷 지연이 중요한 인자이다. 그러나 무선망에서 단말의 이동에 의한 혼잡은 여러 중간 노드를 거치는 것이 아니라 게이트웨이에서부터 베이스스테이션만 거치기 때문에 패킷 지연이 중요한 인자가 아니라 무선망의 특성상 발생할 수 있는 패킷의 비트 에러에 의한 손실을 구분하는 것이 더 중요한 인자가 될 것이다.

본 논문에서 제안하는 혼잡제어 기법은 무선 접속망의 특성을 고려하여 패킷의 비트 에러와 혼잡에 의한 손실을 구분하여 적응적으로 혼잡을 제어할 수 있는 기능을 제공하고, 게이트웨이의 역할을 최대로 활용하여 접속망에 연결한 모든 노드들에게 높은 수준의 공정성을 제공한다.

본 논문의 구성은 다음과 같다.

2장에서는 혼잡 제어 정책과 전송 계층 프로토콜 구조 및 본 논문의 기본 배경이 되는 DCCP에 대하여 살펴보고 3장에서는 본 논문에서 제안하는 이동성을 지원하기 위한 혼잡 제어 기능에 대해서 소개한다. 그리고 리눅스 커널상에서 실제 구현된 혼잡 제어 구조를 설명하고 4장에서 실험 및 성능 평가를 보인다. 끝으로 5장에서는 결론 및 향후 연구 과제를 설명한다.

2. 관련 연구

TCP의 혼잡 제어 정책은 세부분으로 구성된다. 먼저 슬로우 스타트는 혼잡 윈도우가 슬로우 스타트 임계치(Slow Start Threshold)에 도달할 때까지 패킷 손실이 없으면 혼잡 윈도우를 지수적으로 증가시킨다. 그리고 혼잡 윈도우가 슬로우 스타트 임계치를 초과하게 되면

혼잡 회피 과정으로 전이한다. 두 번째 혼잡 회피 단계에서는 혼잡 상태를 사전에 방지할 목적으로 혼잡 윈도우는 각 RTT(Round Trip Time)마다 패킷 손실이 없으면 하나의 패킷 사이즈만큼 증가한다. 만약 타임아웃으로 인하여 패킷 손실이 확인 되면 전송 윈도우는 1로 설정되고 슬로우 스타트 임계치는 전송 윈도우의 절반으로 설정된다. 마지막으로 빠른 재전송 단계는 3개 이상의 연속된 Ack가 도착하면 이전의 패킷을 손실로 인식하고 빠른 재전송을 수행한다[10,11].

멀티미디어 트래픽의 혼잡 제어를 위한 목적으로 TCP-Friendly 혼잡제어 알고리즘들이 제안되었다[2,3,5,9]. 특히 [2,9]에서 제안한 TFRC(TCP-Friendly Rate Control)은 TCP의 전송율을 모델링한 비율 제어 등식에 의해서 TCP와 공정한 대역폭 분배가 가능하도록 하였다. RAP(Rate Adaptation Protocol)는 패킷 손실을 바탕으로 TCP-Friendly하게 응용의 전송율을 조절하는 기법이다. 전송율 조절을 위해서 AIMD(Additive Increase Multiplicative Decrease)알고리즘을 사용하고 IPG(Inter Packet Gap)에 의해서 AIMD는 제어된다[3]. LDA(Loss-Delay based Adjustment algorithm)는 패킷 손실 발생 유무와 함께 현재 전송량과 TCP트래픽의 전송 대역폭을 비교하여 네트워크의 혼잡 여부를 판단한다[5].

TCP의 혼잡 제어 정책 및 응용 계층 혼잡 제어 정책(TCP-like, TCP-friendly)은 유선 네트워크를 위하여 설계되었고, 상황에 따라서 혼잡 윈도우를 지수적으로 빠르게 증가시키거나 혹은 선형적으로 빠르게 증가시키는 정책을 사용한다[2,3,5,9-11]. 그리고 기본적으로 응용 계층 혼잡 제어 정책은 TCP와 같은 슬로우 스타트 및 혼잡 회피 단계로 구성되어 있다. 그러나 모바일 네트워크에서는 기존의 유선 네트워크의 라우터에서의 혼잡처럼 단순한 혼잡 유형만을 포함하고 있지는 않다. 비트 에러에 의한 손실, 단말들의 이동에 의한 베이스스테이션에서의 혼잡 등 다양한 혼잡의 상황을 유발시킨다. 이러한 다양한 혼잡의 상황에서 단순하게 혼잡 윈도우를 지수적으로 증가시키고, 선형적으로 감소시키는 정책은 오히려 혼잡을 더 유발하거나, 네트워크의 처리량을 떨어뜨리는 결과를 초래한다. 그러므로 모바일 네트워크와 같은 환경에서 다양한 혼잡의 상황을 제어할 수 있는 혼잡 제어 정책이 필요하다.

IETF에서 표준화 작업이 진행 중인 DCCP는 기존의 UDP를 기반으로 하는 응용 계층 혼잡 제어 정책을 전송 계층에서 수행하기 위한 프로토콜이다. 현재 DCCP의 혼잡 제어 정책은 지연 시간을 기반으로 한 혼잡 제어 정책인 CCID #3(TCP-Friendly Congestion Control)과 CCID #2(TCP-Like Congestion Control)를 표준으로 채택하고 있다. DCCP는 네트워크의 상태에 따라서 동적으로 혼잡 제어 정책을 교체할 수 있는 메커니즘을 제공한다[4,6-8]. DCCP는 UDP와 마찬가지로 비신뢰적 전송을 하지만 UDP와 달리 연결 설정과 해제 과정이 있으며 데이타 송수신 과정에서 혼잡 제어를 실시한다. DCCP의 표준에서 제공하는 혼잡 제어 메커니즘 중 CCID #2는 TCP-like한 혼잡 제어 메커니즘으로 TCP와 거의 유사하게 혼잡 제어를 실시하지만 Ack 벡터를 사용하여 Ack 메시지를 보내는 빈도를 줄인다[4]. CCID #3은 TCP-Friendly한 메커니즘으로 TCP와 친화적인 흐름을 지향하는 TFRC방식이다. CCID #3에서 전송율은 Loss Event Rate에 맞춰서 서서히 증가시키고, 몇 개의 연속적인 Loss Event 가 발생하면 전송율을 반으로 줄인다[7].

I-TCP(Indirect TCP)와 M-TCP(Mobile-TCP)에서는 무선망의 특성을 고려하여 무선망과 유선망을 분리하여 하나의 연결에서 두개의 TCP 세션이 존재하였다[12,13]. I-TCP는 기지국에 I-TCP의 Agent을 두고 유선망과 무선망의 서로 다른 연결을 포워딩한다. 그리고 무선망에서는 무선망에 적합한 변경된 TCP를 사용하였다[12]. M-TCP는 이동 단말이 핸드오프를 일으킬 때 나타나는 성능 저하 문제를 개선하기 위한 방안을 제시한다[13]. [14]에서는 기존의 UDP를 모바일 네트워크 환경에 맞추어 재구성한 M-UDP(Mobile UDP)를 발표하였다. M-TCP와 M-UDP 역시 I-TCP와 마찬가지로 이중 연결 구조를 사용한다.

3. 무선접속망에 적합한 혼잡 제어 정책

본 논문에서는 DCCP를 기반으로 무선 접속망에 적합한 혼잡 제어 정책을 제안한다. 유비쿼터스 네트워크 환경에서는 무선 단말과 유·무선을 연결하는 게이트웨이 사이에 베이스스테이션만 존재하는 한 홉의 구조라고 가정한다. 중간노드에서는 라우팅 과정이 없고 모든 단말들이 이동할 수 있으므로 이에 적합한 혼잡 제어 정책이 필요하다.

본 논문에서 제안하는 혼잡 제어 정책을 적용하기 위한 가상의 네트워크 구조는 그림 2와 같으며 유·무선이 혼재한 상황에서 무선접속망에서 적용할 수 있는 혼잡 제어 정책이다. TCP나 DCCP의 CCID #2, CCID #3은 각각 TCP-Like와 TCP-Friendly와 같은 유선 인터넷에서 사용하는 혼잡 제어 정책을 사용하기 때문에, 그림 2와 같은 네트워크 구조에서는 실질적인 효과를 기대하기 어렵다. DCCP 표준 또한 일반적인 인터넷 환경을 고려하고 있으며, TCP와 유사한 방법으로 전송률을 조절하기 위해서 중간 라우터에서 발생되는 패킷 손실만을 고려한다. 그러나 무선망의 경우 게이트웨이를 중심으로 연결된 단말들이 하나의 홉(베이스스테이션)만을

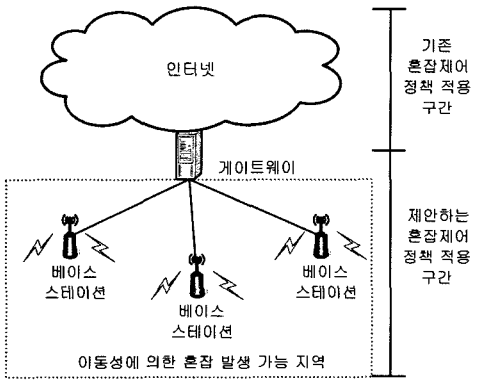

그림 2 혼잡 제어 정책을 위한 유비쿼터스 네트워크 구조

거치기 때문에 중간 라우터의 패킷 손실은 무의미하며 모든 단말들이 게이트웨이의 통제 하에 있기 때문에 효율적인 혼잡 제어를 수행할 수 있다. 게이트웨이는 모든 단말들이 전송과정에서 얻게 되는 전송 윈도우의 크기를 알고 있으며, 현재 게이트웨이가 서비스하고 있는 모든 단말들의 혼잡 윈도우의 총 크기도 알 수 있다. 따라서, 패킷 손실과 지연뿐만 아니라 게이트웨이가 관찰하는 각 단말의 윈도우 크기와 자신의 가용 윈도우 크기를 동시에 고려하여 혼잡 제어를 한다.

그림 3은 본 논문에서 제안하는 혼잡 제어 정책을 수행하기 위한 전체적인 흐름을 보인다. 메시지 및 데이타 흐름은 DCCP의 연결 설정 과정과 데이타 전송 과정 및 연결 해제 과정을 따른다. A 구간은 데이타 연결 과정이고 B구간은 데이타 전송 과정, 그리고 C구간은 연

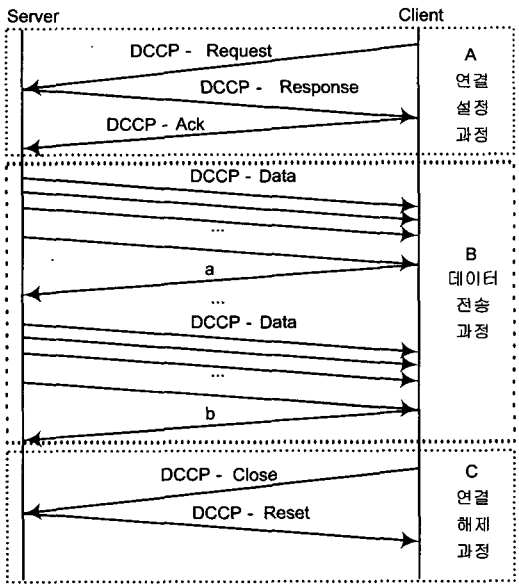

그림 3 DCCP기반의 메시지 및 데이타 흐름

결 해제 과정이다. DCCP에서는 일반적인 UDP기반의 프로토콜들과는 달리 신뢰성 있는 연결과 종료를 위한 핸드쉐이크 과정을 거친다. A구간에서의 연결 설정과정에서는 여러 특성값들에 대한 협상이 이루어지고, 데이타 전송 과정에서 사용할 혼잡 제어 정책도 이 협상 과정에서 선택된다. B구간에서는 혼잡 윈도우 크기만큼의 데이타 패킷들을 전송하고 이에 따른 Ack 벡터를 포함한 응답 패킷(그림 3의 a,b)을 전송하는 과정을 반복적으로 수행한다. 그리고 서버는 혼잡제어의 결과에 따라서 혼잡 윈도우 크기를 조절하고 조절된 혼잡 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 크기로 정한다. C구간은 연결 해제 과정으로 서버와 클라이언트 양쪽 모두 연결 해제를 요구 한다.

3.1 혼잡 제어 시 고려 사항

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

• 무선 환경이 가지는 특성에 따른 에러 상황의 검출

유선과 달리 무선에서 발생되는 비트에러에 따른 패킷 손실과 실제 혼잡에 따른 손실 제어 정책을 달리하는 것이 무선의 혼잡 제어 정책에 있어 가장 중요한 부분이다. 기존의 TCP와 응용 계층 혼잡 제어 정책에서는 에러 상황을 검출할 수 있는 기법을 포함하고 있지 않다. 본 논문에서 제안하는 혼잡 제어 알고리즘에서는 이와 같은 비트 에러에 따른 패킷 손실과 혼잡에 따른 패킷 손실을 다르게 처리할 수 있는 새로운 방법을 제시한다. 그리고 본 논문에서는 새로운 패킷 손실 정도를 측정하는 기법을 제공하여 무선 환경의 Burst Error특성을 고려할 수 있으며 패킷 손실 정도에 따라서 혼잡 제어 정책을 수행한다.

• AIMD에 의한 대역폭의 낭비

일반적으로 TCP나 유사한 정책을 표방하는 프로토콜의 혼잡 제어 정책에서 손실이 발생 하면, 전송 윈도우를 절반으로 줄인다. 만약 가벼운 혼잡이 발생한 경우에도 전송 윈도우의 크기가 큰 단말들이 동시에 혼잡 윈도우를 절반으로 줄이면, 혼잡을 해소하기 위한 대역폭의 양보다 더 많은 대역폭이 확보되고 대역폭 이용율은 저하된다. 그리고 혼잡이 해소된 후에 전송 윈도우는 선형적으로 증가함으로 상당한 시간 동안 네트워크 대역폭이 낭비된다. 기존의 기법들은 이러한 AIMD(Additive Increasement Multiple Decreasement)만을 지원하며 혼잡의 정도에 따라서 적응적으로 혼잡 윈도우를 조절하지 않기 때문에 가벼운 혼잡시에 대역폭의 낭비를 초래한다.

• 단말의 이동성에 따른 패킷 손실

무선 환경에서는 한 홉의 경로를 통해 패킷이 전달되고, 패킷 손실은 여러 단말이 동시에 접속된 자신의 처리 용량을 초과한 베이스스테이션에서 발생한다. 따라서 베이스스테이션에 발생된 패킷 손실이 게이트웨이나 무선 단말 측에서 감지 될 경우, 새롭게 손실이 추가된 양만큼의 혼잡 윈도우 크기를 조절하여 동시에 서비스 중인 단말들과의 혼잡을 피하고 네트워크 대역폭 이용율을 향상시키기 위한 방법이 필요하다. 무선 환경에서의 혼잡을 제어하기 위해서 기존의 기법들은 모든 패킷 손실 상황을 혼잡으로 간주하지만 본 논문에서는 증가된 혼잡 윈도우의 크기와 손실 패킷의 수에 따라서 혼잡의 정도를 측정한다.

• 게이트웨이에서 관리되는 전송 윈도우 크기

게이트웨이에 연결되어 있는 여러 베이스스테이션은 자신의 셀 영역내에서 다수의 무선 단말들을 서비스 하고 있다. 베이스스테이션은 라우팅 기능을 포함하지 않는 단순 전달 기능만을 포함하기 때문에, 게이트웨이는 현재 서비스 중인 단말의 수와 각 단말들의 최대 전송 윈도우 사이즈 변화, 게이트웨이가 가용할 수 있는 윈도우의 크기를 알 수 있고, 또한 모든 세션에 대해서 제어가 가능하다. 이를 기반으로 각 단말과 게이트웨이 사이의 전송량을 공평하게 조절할 수 있다. 기존의 혼잡 제어 정책은 종단간의 서버와 클라이언트에 의해서 적용되었지만 적응적 혼잡 제어 정책은 게이트웨이를 이용하여 적응적인 혼잡 제어를 실시한다.

3.2 적응적인 혼잡 제어 알고리즘

3.2.1 개요

본 논문에서 제안하는 혼잡제어 정책은 총 5개의 단계로 구성 된다. 먼저 슬로우 스타트 단계는 하나의 세션이 수립되거나 혹은 혼잡이 발생 한 후에 전이하는 단계이다. 슬로우 스타트 단계에서는 혼잡의 정도에 따라서 슬로우 스톱(Slow Stop) 단계와 혼잡 윈도우를 다시 초기화(전송 윈도우 크기 : 1)하는 슬로우 스타트(Slow Start) 단계로 회귀한다. 그리고 혼잡 상태와 무선망의 특성에 의한 패킷 손실을 구별하기 위해서 역 혼잡 회피(Reverse Congestion Avoidance) 단계를 거친다. 슬로우 스톱 단계는 혼잡 윈도우 크기를 지수적으로 서서히 감소시키기 때문에 혼잡 윈도우 크기가 갑자기 줄어들면서 생기는 클라이언트의 수신량 급감을 방지할 수 있으므로 네트워크의 대역폭 낭비를 개선할 수 있다. 두 번째로 혼잡 회피 단계는 슬로우 스타트 단계에서 혼잡이 발생하지 않고 혼잡 윈도우 크기가 일정한 임계값을 초과하면 전이하는 단계이다. 이 단계에서 혼잡이 발생하면 혼잡의 정도에 따라서 역 혼잡 회피 단계와 슬로우 스톱 단계 그리고 다시 초기 단계인 슬로우 스타트 단계로 전이 한다. 본 단계에서도 역시 혼잡 상태와 무선망에 의한 패킷 손실인지를 판단하기 위해서 역 혼잡 회피 단계를 사용한다. 그리고 마지막으로 Constant 단계는 혼잡 회피 단계에서 혼잡이 발생하지 않고 게이트웨이에서 할당하는 최대 혼잡 윈도우 임계치까지 혼잡 윈도우 크기가 증가하면 더 이상 혼잡 윈도우를 증가시키지 않는 단계이다. 이 단계에서는 혼잡 윈도우가 더 이상 증가하지 않는 것을 제외하면 혼잡 회피 단계와 동일하다.

본 논문에서 제안하는 혼잡 제어 정책은 기존의 혼잡 제어 정책에서 사용하는 슬로우 스타트 단계와 혼잡 회피 단계는 유사한 방법으로 사용하다. 그리고 Constant 단계로 전이하기 위하여 기존의 TCP가 흐름제어 정책의 수신 윈도우의 최대 크기를 임계치로 사용하는 반면에 본 논문에서는 게이트웨이의 전체 세션 관리 정책에 의해서 설정된 값과 수신 윈도우의 크기 중 최소값을 사용한다. 그림 4에서는 위에서 설명한 각 단계의 전이 과정을 보여준다.

그림 4에서 SS는 슬로우 스타트단계이고, CA는 혼잡 회피 단계, C는 Constant단계, ST는 슬로우 스톱 단계, RCA는 역 혼잡 회피 단계이다. 각 단계간의 전이는 (1)로 구분한다

$$i_B^A \rightarrow C \quad (1)$$

(1)에서 A는 혼잡인지 아닌지를 구분하는 인자이다. 혼잡인 경우에는 C(Congestion)이고, 비트 에러에 따른 패킷 손실인 경우에는 E(Error), 그리고 패킷의 손실이 없는 경우에는 N(Normal)이다. 그리고 $B \rightarrow C$는 B상태에서 C상태로의 전이를 나타낸다. 예를 들어 $i_{CA}^C \rightarrow ST$는 혼잡 회피 상태에서 혼잡이 발생하여 슬로우 스톱 단계로 전이하는 것을 의미한다.

3.2.2 임계값의 설정

TCP나 유사한 프로토콜의 혼잡 제어 정책은 그림 5와 같은 방법으로 혼잡 윈도우를 조절한다. 각 단계별 진행의 기준이 되는 임계치에는 슬로우 스타트 임계치와 최대 혼잡 윈도우 임계치가 있다. TCP는 전송초기에 슬로우 스타트 임계치를 최대 혼잡 윈도우 임계치의 절반으로 설정하여 전송을 시작한다. 이후 슬로우 스타트 임계치는 패킷 손실이 발견될 때, 혼잡 윈도우의 절반으로 줄여서 에러 상황에 따라 혼잡 윈도우를 조절할 수 있다.

본 논문에서 제안하는 혼잡 제어 정책도 TCP와 유사하게 슬로우 스타트 임계치와 최대 혼잡 윈도우 임계치를 가지며, 서비스중인 단말의 수와 패킷 손실 상황에 따라 임계치를 조절한다. 그림 2와 같은 구조에서 모든 세션들이 게이트웨이를 통하여 접속하기 때문에 게이트

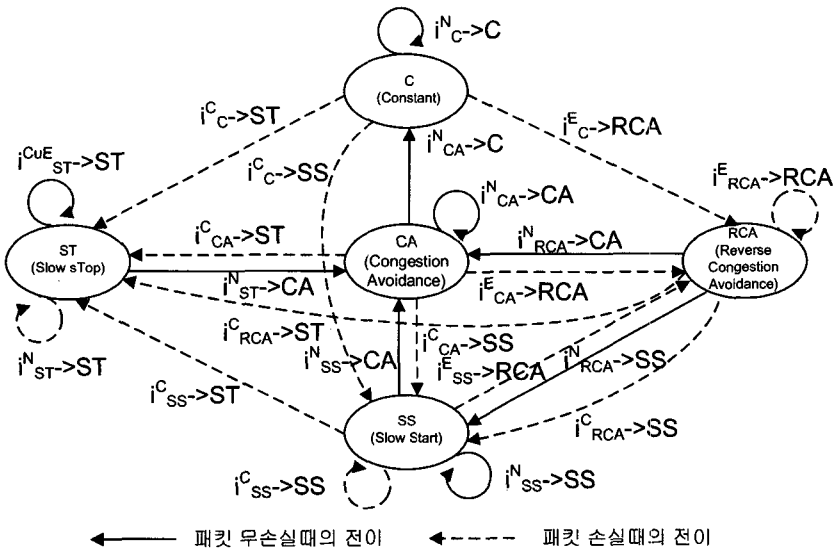

그림 4 혼잡 제어 정책의 상태 전이도

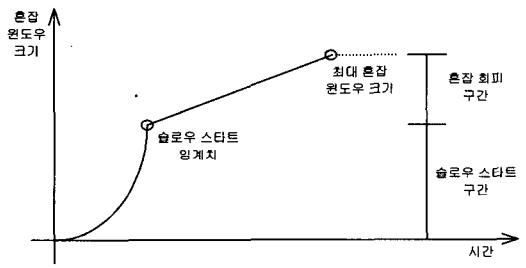

그림 5 TCP에서의 혼잡 제어

웨이를 통하여 임계치를 설정하는 것이 가능하다. 따라서 본 논문에서는 임계치 설정을 위하여 게이트웨이에서 수집된 정보를 이용한다. 표 1에서는 본 논문에서 사용하는 기호들을 정의한다.

먼저 세션 i의 최대 혼잡 윈도우 임계치(T^i_{Max})는 (2)와 같이 구한다.

$$T^i_{Max} = \min\left\{ W^G_{Max}/N,\ W^i_{recv} \right\} \quad (2)$$

(2)에서는 게이트웨이가 세션 i에게 할당할 수 있는 최대 윈도우 크기와 세션 i의 수신 윈도우 크기 중에서 최소값을 최대 혼잡 윈도우 크기로 설정한다. 그리고 T^i_{Max}는 새로운 세션이 추가되거나 종료 될 때마다 N이 변화하기 때문에 (2)를 재계산하여 새로운 T^i_{Max}를 할당한다.

$$T^i_{SS} = \begin{cases} T^i_{Max}\ , & \sum_{i=1}^{N} T^i_{Max} < W^G_{Max} \\ (1-\frac{1}{N+1}) \times T^i_{Max}\ , & \sum_{i=1}^{N} T^i_{Max} \geq W^G_{Max} \end{cases} \quad (3)$$

(3)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 세션을 위한 대역폭을 미리 남겨두어서 혼잡을 사전에 방지하기 위한 것이다. 그리고 새로운 세션이 추가되면 (3)을 재계산하여 새로운 T^i_{SS}를 구한다. 만약 혼잡에 의해서 패킷 손실이 발생하면 T^i_{SS}는 (4)를 이용하여 새롭게 구한다. 그리고 T^i_{Max}와 T^i_{SS}는 게이트웨이에 의해서 결정된다.

$$T^i_{SS} = \frac{T^i_{Max}}{2} \quad (4)$$

3.2.3 혼잡 제어 단계별 과정

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 3단계 외에 비트 에러와 혼잡 상황에 따른 패킷 손실을 구별하기 위한 역 혼잡 회피 단계(RCA: Reverse Congestion Avoidance)와 약한 혼잡이 발생되었을 때 낭비되는 대역폭을 방지하기 위한 슬로우 스톱(Slow Stop) 단계로 구성된다. 역 혼잡 회피 단계에서는 혼잡 윈도우를 선형적으로 감소시키면서 다음 단계의 패킷 손실 정도를 탐지하여 정확한 혼잡의 정도와 에러 상황을 인식한다.

그림 6은 본 논문에서 제안하는 혼잡 제어 정책의 각각의 단계를 설명하고 있다. 그림 6의 두 번째 구간에서

표 1 기호 정의

기호	의미
i	세션의 인덱스
N	게이트웨이를 통해서 서비스 중인 모든 세션의 수
$T^i_{\max}$	세션 i의 최대 혼잡 윈도우 임계치; 세션 i를 위해서 게이트웨이가 한번에 최대로 송신 할 수 있는 대역폭
$W^G_{\max}$	게이트웨이의 최대 가용 윈도우 크기; 게이트웨이의 최대 대역폭을 의미
W^i_{recv}	세션 i의 최대 수신 윈도우 크기; 세션 i가 한번에 받아 들일 수 있는 최대 윈도우 크기, 즉 수신 버퍼의 크기를 의미
T^i_{ss}	세션 i의 슬로우 스타트 임계치; 세션 i의 혼잡 윈도우가 지수적 증가에서 선형적 증가로 변화하는 대역폭의 기준값
$C(X)$	대역폭 X를 패킷의 개수로 나타내는 함수
$C(T^i_{ss})$	세션 i의 슬로우 스타트 임계치 패킷 개수; 세션 i의 슬로우 스타트 임계치만큼의 패킷 개수
$\sum_{i=1}^{N} T^i_{\max}$	게이트웨이를 통해서 연결 중인 모든 세션들의 최대 혼잡 윈도우임계치의 합
P^i_{loss}	세션 i의 패킷 손실 정도; 세션 i의 혼잡 윈도우에서 증가된 패킷 개수에서 손실된 패킷의 개수를 감해준 값
W^i_{inc}	세션 i의 증가된 혼잡 윈도우 크기; 이전 혼잡 윈도우 크기에서 혼잡 제어를 수행 한 후에 증가된 혼잡 윈도우의 크기
$C(W^i_{inc})$	세션 i의 증가된 혼잡 윈도우 패킷 개수; 이전 혼잡 윈도우 크기에서 혼잡 제어를 수행 한 후에 증가된 혼잡 윈도우의 패킷 개수
AV^i_{loss}	Ack 벡터의 손실 패킷 개수; 혼잡 윈도우 크기만큼의 패킷을 전송 한 후에 단말은 손실된 패킷과 손실되지 않은 패킷의 정보를 하나의 Ack 벡터를 통해서 전송 할 때, Ack 벡터가 나타내는 손실된 패킷의 개수
W^i_{prev}	세션 i의 이전 혼잡 윈도우 크기; 세션 i의 현재 혼잡 윈도우 크기에서 증가된 혼잡 윈도우 크기를 제외한 크기
$C(W^i_{prev})$	세션 i의 이전 혼잡 윈도우의 패킷 개수; 세션 i의 현재 혼잡 윈도우의 패킷 개수에서 증가된 혼잡 윈도우의 패킷 개수를 제외한 값

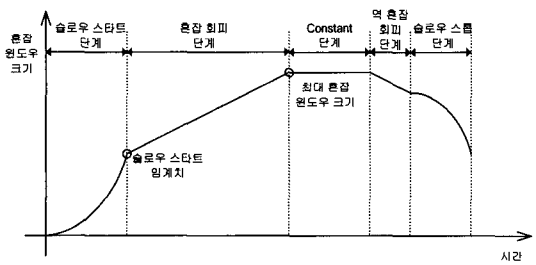

그림 6 적응적 혼잡 제어 정책의 혼잡 제어 5 단계

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.

본 논문에서 제안하는 혼잡 제어 정책은 TCP와 마찬가지로 새로운 세션이 도착했을 때는 슬로우 스타트 단계에서 시작한다. 슬로우 스타트는 혼잡 윈도우 크기를 1로 시작해서 혼잡 윈도우의 패킷 개수만큼의 무손실 Ack 벡터가 도착했을 때 혼잡 윈도우 크기를 2배로 증가해서 전송한다. 증가된 혼잡 윈도우 크기가 T^i_{SS}에 이를 경우 혼잡 회피 단계로 진행되며, T^i_{Max}와 T^i_{SS}가 동일할 경우 Constant 단계로 진행된다. 그리고 슬로우 스타트 단계에서 패킷의 손실이 발생한 경우에는 패킷의 손실정도에 따라 역 혼잡 회피나 슬로우 스톱 혹은 슬로우 스타트의 초기 단계로 다시 시작 할 수 있다. 본 논문에서는 혼잡의 정도에 따른 패킷의 손실 정도 (P^i_{loss})를 측정하기 위해서 (5)를 이용한다.

$$P^i_{loss} = C(W^i_{\in c}) - AV^i_{loss} \quad (5)$$

그림 7에서 i구간의 혼잡 윈도우의 패킷 개수는 m개이며, 손실이 없었기 때문에 Ack 벡터의 m비트는 모두 1로 채워진다. 그리고 $i+1$구간에서는 n개의 패킷 개수만큼 혼잡 윈도우가 증가하여 총 $m+n$개의 패킷을 전송한다. 그리고 $m+n$개의 패킷 중에서 m개의 패킷에

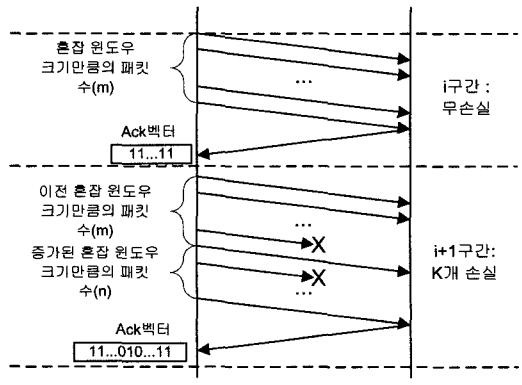

그림 7 패킷 손실 정도 측정 과정

대한 혼잡은 이미 i구간에서 혼잡이 아닌 것으로 확인하였다. 그러나 $i+1$ 구간에서는 k개의 패킷이 손실 되었다. 이 손실은 i구간에서 이미 무손실 전송을 확인 했던 m개의 패킷이 아닌 n개의 패킷이 새로 증가하였기 때문에 발생했을 가능성이 크다. 그러므로 본 논문에서 새롭게 정의하는 패킷 손실 정도는 증가된 n개의 패킷에서 손실된 k개의 손실 패킷 개수를 감한 값으로 정의한다.

그림 7과 같은 패킷 손실 정도 측정 과정에서는 무선 환경의 Burst Error 및 다양한 손실 정도를 측정할 수 있다. 그리고 각각의 단계에서는 패킷 손실 정도에 따라서 적절한 혼잡 제어 정책을 실시한다.

그림 8에서는 P^i_{loss}를 이용하여 다음 진행 과정을 결정한다.

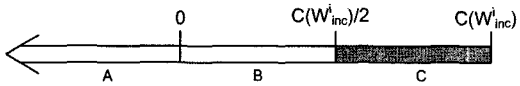

그림 8 슬로우 스타트 단계에서의 혼잡 제어 과정

그림 8에서 P^i_{loss}가 만약 A구간에 속한다면 증가된 혼잡 윈도우 크기보다 더 많은 패킷들이 손실되었다는 것을 알 수 있다. 즉 이것은 현재 세션 i가 속한 베이스 스테이션에서 혼잡이 발생하였을 가능성이 크다는 것을 의미하므로 슬로우 스타트 단계를 수행한다. P^i_{loss}가 B 구간에 속한다면 A구간보다는 적은 수의 패킷 손실이 발생하였지만 복수개의 패킷 손실이 발생했으므로 혼잡 상태로 가정한다. 그러나 이러한 경우에 A구간에 비하여 혼잡의 상태가 약하기 때문에 슬로우 스톱 단계를 수행한다. 슬로우 스톱 단계는 혼잡 윈도우 사이즈를 지수적으로 감소시키기 때문에 TCP의 슬로우 스타트처럼 혼잡 윈도우를 반으로 줄이는 경우보다는 혼잡 제어의 강도는 약하지만 낭비되는 네트워크 자원은 줄일 수 있다. 마지막으로 P^i_{loss}가 구간 C에 속한다면 이는 경미한 패킷 손실을 포함하는 경우이다. 이러한 경우는 혼잡이 아닐 가능성이 크기 때문에 역 혼잡 회피 단계를 거쳐서 혼잡 상황인지 아닌지를 판단하게 된다. 그림 8에서 B와 C구간을 구분하기 위한 인자로 $\frac{W^i_{inc}}{2}$를 사용한 이유는 슬로우 스타트 단계에서의 혼잡 윈도우 크기가 이전 단계에 비하여 두 배씩 증가하기 때문이다.

그림 9에서는 슬로우 스타트 단계에서 전이할 수 있는 각각의 단계를 설명한다. 슬로우 스타트 단계에서 슬로우 스타트 단계로 회귀하는 것은 두 가지 경우이다. 첫 번째는 슬로우 스타트 임계치보다 혼잡 윈도우 크기

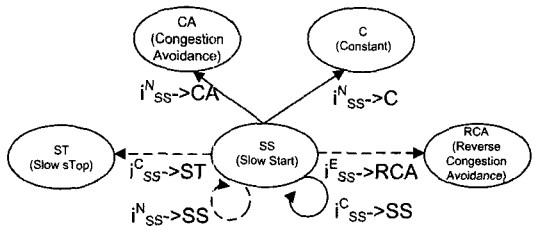

그림 9 슬로우 스타트 단계의 상태 전이도

가 작으면서 패킷의 손실이 없는 경우($i^N_{SS}\rightarrow SS$)이다. 두 번째는 슬로우 스타트 단계에서 혼잡이 발생하여 혼잡 윈도우를 초기화 시키고 슬로우 스타트를 수행하는 경우($i^C_{SS}\rightarrow SS$)(그림 8의 A구간)이다. 그리고 패킷 손실 정도가 그림 8의 B구간과 C구간에 속하여 슬로우 스톱과 역 혼잡 회피 단계로 전이하는 경우는 $i^C_{SS}\rightarrow ST$와 $i^E_{SS}\rightarrow RCA$이다. 만약 슬로우 스타트 단계에서 패킷 손실이 발생하지 않고 슬로우 스타트 임계치보다 혼잡 윈도우가 커진다면 혼잡 회피 단계나 Constant단계로 전이할 수 있다. 이 경우에 슬로우 스타트 임계치와 최대 혼잡 윈도우 임계치가 같다면 혼잡 회피 단계를 거치지 않고 Constant 단계로 전이하고, 그렇지 않다면 혼잡 회피 단계로 전이한다.

슬로우 스타트 단계에서 혼잡 윈도우 크기가 슬로우 스타트 임계치(T^i_{SS})에 도달 하면 혼잡 회피 단계로 진행한다. 이 단계에서는 윈도우 사이즈는 선형적으로 증가하며, 혼잡 윈도우의 크기가 최대 혼잡 윈도우 임계치(T^i_{Max})에 도달하면 Constant 단계를 수행한다. 혼잡 회피 단계에서도 패킷의 손실 정도(P^i_{loss})에 따라서 혼잡의 정도 및 상태를 판단하여 다음 수행 단계를 결정한다. 그림 10은 혼잡 회피 단계에서의 혼잡제어 정책 결정의 기준을 보인다.

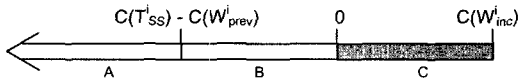

그림 10 혼잡 회피 단계의 혼잡 제어 과정

그림 10에서 A, B, C는 혼잡의 정도 혹은 혼잡 상태인지 아닌지를 판단하는 구간들이다. A구간은 혼잡 윈도우의 패킷들 중에서 손실되지 않고 전송 된 패킷의 개수가 슬로우 스타트 임계치값보다 적게 수신되었다는 것을 의미하는 영역이다. 그리고 많은 패킷이 손실되었기 때문에 슬로우 스타트 단계를 수행한다. B구역은 A구역의 임계치보다 크고 혼잡 윈도우가 증가되기 전의 크기보다는 작은 구역이다. 이 구역은 이전의 슬로우 스

타트 단계에서 벗어난 혼잡 회피 단계에서의 혼잡 윈도우 크기 변화량과 동일하다. 그러므로 패킷 손실 정도가 B구간에 속한다면 혼잡 회피 과정과 유연한 혼잡 제어를 동시에 수행하기 위해 슬로우 스톱 단계를 수행한다. C구간은 그림 8의 C구간과 동일한 의미이지만 혼잡 회피 단계에서는 전송 윈도우가 선형적으로 증가하기 때문에 1이고, 역 혼잡 회피 단계를 수행한다. 그림 11은 A, B, C 구간을 전체 혼잡 윈도우의 관점에서 설명한다.

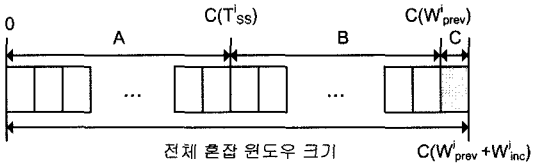

그림 11 혼잡 윈도우 관점에서 본 패킷 손실 구간

그림 11에서 A구역은 슬로우 스타트 임계치(T^i_{SS})보다 작은 양의 패킷이 수신측 송신된 경우이고, B구역은 슬로우 스타트 임계치(T^i_{SS})와 이전 혼잡 윈도우 크기 사이에 패킷이 전송된 경우이다. C구역은 단지 하나의 패킷이 손실 된 경우이다. C구역에서 하나의 패킷은 혼잡 회피 단계에서 증가된 혼잡 윈도우 크기를 의미한다.

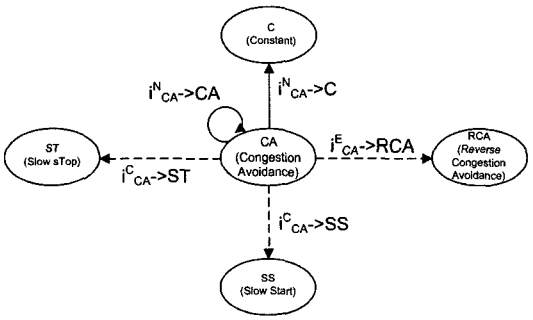

그림 12 혼잡 회피 단계의 상태 전이도

그림 12는 혼잡 회피 단계에서의 상태 전이를 나타낸다. 혼잡이 발생하면 혼잡 발생 정도에 따라서 그림 10과 그림 11의 B와 C구간에 해당하는 $i^C_{CA}\rightarrow SS$혹은 $i^C_{CA}\rightarrow ST$에 의해서 슬로우 스톱 단계나 슬로우 스타트 단계로 전이하고 패킷 에러인 경우에는 $i^E_{CA}\rightarrow RCA$에 의해서 역 혼잡 회피 구간으로 전이한다. 패킷 손실이 발생하지 않으면 최대 전송 윈도우 임계치에 따라서 혼잡 회피 구간을 계속 수행하거나 혹은 Constant단계로 전이한다.

혼잡 윈도우 크기가 최대 전송 윈도우 임계치(T^i_{Max})에 도달하게 되면 Constant 단계로 진행되며, 이 때부터는 더 이상의 혼잡 윈도우 증가 없이 현재의 혼잡 윈도우 크기를 유지한다. 그러나 게이트웨이를 통과하는 전체 세션들의 수가 변화함에 따라서 최대 혼잡 윈도우 임계치(T^i_{Max})가 변화하게 되면, 슬로우 스톱 단계 혹은 혼잡 회피 단계를 수행한다. 만약 최대 혼잡 윈도우 임계치가 증가하면 혼잡 회피 단계를 수행하고, 최대 혼잡 윈도우 임계치가 줄어든다면 슬로우 스톱 단계를 수행한다.

Constant단계에서 패킷 손실에 따른 혼잡 제어는 그림 10과 그림 11에 나타나 있는 혼잡 회피 단계의 혼잡 제어 정책과 동일하다. 그 이유는 최대 혼잡 윈도우 임계치가 없다면 혼잡 회피 단계가 계속 진행될 것이므로 Constant 단계도 혼잡 회피 단계의 연속으로 가정할 수 있기 때문이다.

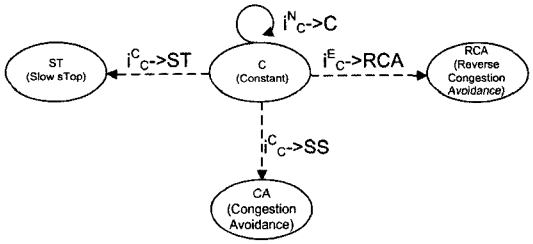

그림 13 Constant 단계에서의 상태 전이

그림 13은 Constant 단계에서의 상태 전이를 나타낸다. 그림 13의 A, B, C는 그림 12의 A, B, C와 같은 제약 조건에 따른 상태 전이를 나타낸다. 그리고 나머지 상태 전이 또한 혼잡 회피 단계와 같다.

무선 접속망은 유선과 달리 무선 매체에 따른 비트 에러가 빈번하게 발생하며, 이러한 비트 에러는 패킷 손실로 인식되어 실제 혼잡 제어 정책을 혼란스럽게 만드는 요인으로 작용한다. 이에 따라 패킷 손실이 발견되었을 경우 비트 에러로 인한 패킷 손실인지 실제 혼잡에 의한 패킷 손실인지를 구분하는 단계가 필요하다. 역 혼잡 회피 단계는 현재 전송되는 윈도우 크기를 하나씩 감소시키면서, 혼잡인지 비트 에러인지를 판단한다. 혼잡에 의한 패킷 손실이면 다수의 패킷이 손실될 것이고, 비트에러에 의한 손실이면 패킷 손실이 거의 발견되지 않을 것이다. 그러므로 경미한 패킷 손실이후에 이 단계를 수행함으로 정확한 에러의 원인을 발견할 수 있다. 그림 14는 역 혼잡 회피 단계에서 패킷 손실에 따른 처리 과정을 보여준다.

무선 단말과 게이트웨이간에 패킷이 정상적으로 전송

```
A : 현재 역 혼잡 회피 단계에서 손실된 총 패킷 수
B : 역 혼잡 회피 단계의 이전 단계에서 손실된 총 패킷 수

if (A = 0)
  역 혼잡 회피 단계의 이전 단계 수행
else if (A = 1)
  역 혼잡 회피 단계 계속 수행
else if (A = 1)
  슬로우 스타트 단계 수행
else
  슬로우 스톱 단계 수행
```

그림 14 역 혼잡 회피 단계에서의 혼잡 제어 수행

될 경우, 슬로우 스타트, 혼잡 회피, Constant 단계를 거쳐서 연결이 종료된다. 하지만, 게이트웨이와 무선 단말 사이에서 전송을 중계하는 베이스스테이션에서 전송 용량을 초과하는 무선 단말들이 집중 될 경우, 게이트웨이의 전송 능력에 상관없이 혼잡이 발생한다.

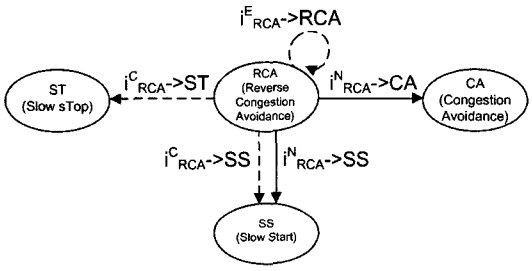

그림 15 역 혼잡 회피 단계에서의 상태 전이

그림 15는 역 혼잡 회피 단계에서의 상태 전이를 나타낸다. 그림 15에서 $i^{N}_{RCA}\rightarrow SS$ 와 $i^{N}_{RCA}\rightarrow CA$는 역혼잡 회피 단계에서 패킷 손실이 발생하지 않았기 때문에 역 혼잡 회피 단계로 전이하기 전의 단계인 슬로우 스타트 단계와 혼잡 회피 단계로 복귀하는 경우이다. 그리고 Constant 단계에서 역 혼잡 회피 단계로 전이한 후에 패킷 손실이 발생하지 않으면 Constant 단계로 복귀하지 않고 $i^{N}_{RCA}\rightarrow CA$에 의해서 혼잡 회피 단계를 거쳐서 Constant 단계로 복귀한다. 그리고 $i^{E}_{RCA}\rightarrow RCA$는 역 혼잡 회피 단계에서 계속하여 작은 수의 패킷 손실이 발생하여 계속 역 혼잡 회피 단계에 머물러 있는 것을 의미한다.

일반적으로 TCP와 같은 전송 프로토콜들은 혼잡 윈도우를 반으로 감소시키거나 혹은 패킷 하나의 크기로 줄여서 다시 슬로우 스타트 단계에서 진행하지만, 이 과정을 혼잡의 정도에 따라서 세분화하지 않는다. 그러나 혼잡 윈도우를 절반으로 줄이는 과정에서 게이트웨이가 제공할 수 있는 대역폭에서 낭비하는 부분이 발생할 수 있으며, 이는 전체적인 시스템 성능 하락의 원인이 된다. 그러므로 현재 혼잡 윈도우를 절반으로 감소시키는 과정을 한번에 수행하는 것이 아니라 슬로우 스타트와 반대되는 개념으로 지수적으로 혼잡 윈도우 크기를 감소하는 과정이 슬로우 스톱 단계이다. 이 과정을 통해서 대역폭의 낭비를 방지할 수 있다.

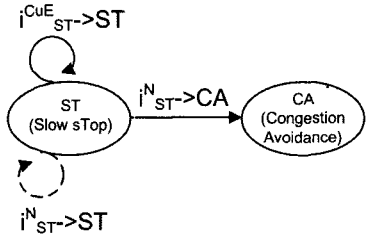

그림 16 슬로우 스톱 단계에서의 상태 전이

그림 16은 슬로우 스톱 단계에서의 상태 전이를 보이며 혼잡 윈도우를 절반으로 감소시키고 혼잡 회피 단계로 전이한다($i^{N}_{ST}\rightarrow CA$). 슬로우 스톱 단계 중에 계속하여 패킷 손실이 발생하면 강화된 슬로우 스톱 단계를 수행한다. 슬로우 스톱 단계는 패킷 손실이후 혼잡 윈도우를 절반으로 감소하는 일반적인 혼잡 제어 방법에 비해 상대적으로 느린 혼잡 제어가 제공되고, 슬로우 스톱 단계에서 계속적으로 혼잡이 발생하여 정상적인 서비스를 방해할 수 있기 때문에 슬로우 스톱 단계에서의 혼잡 발생($i^{N}_{ST}\rightarrow ST$)은 에러 발생 횟수에 따라 2의 승수를 곱한 크기로 혼잡 윈도우 크기를 감소하여 심한 혼잡 상황일 경우 일반적인 혼잡 제어 방법과 큰 차이 없이 현재 혼잡 윈도우 크기의 절반으로 줄일 수 있다.

그림 17은 슬로우 스톱 단계의 처리 과정을 보여준다. 그림 17에서 혼잡 윈도우 크기가 125인 지점에서 패킷 손실이 발생하였다. 그리고 혼잡 윈도우가 125가 될 때까지는 지수적으로 혼잡 윈도우가 감소되었다. 만약 패킷 손실이 발생하지 않았다면 다음 혼잡 윈도우 크기는 이전 혼잡 윈도우 크기의 2배인 8만큼 줄어들겠지만 패킷 손실이 125지점에서 한번 발생했기 때문에 2^{1}이 더 곱해진 16만큼 줄어든다. 그리고 전송 윈도우 크기가 121인 지점에서 두 번째 패킷 손실이 발생하였으므로 혼잡 윈도우의 크기는 2^{2}이 더 곱해진 64만큼 줄어든다. 이 때 줄어든 전송 윈도우 크기는 44가 되지만 슬로우 스톱 단계 초기에 설정된 한계 값이 64이므로 64로 설정된다.

이상의 5단계를 통해서 무선 접속망에서 발생되는 혼잡 상황에 대한 효율적인 제어가 가능하다. 그러나 각 단말의 최대 혼잡 윈도우 크기는 신규 세션의 연결 수

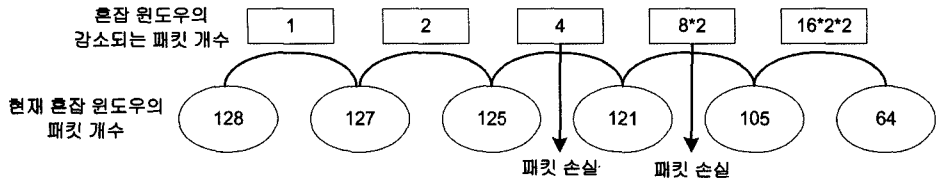

그림 17 슬로우 스톱 단계에서 패킷 손실에 따른 처리 과정

락에 의해서 감소된다. 이 때는 현재 혼잡 윈도우 임계치에서 감소된 최대 혼잡 윈도우 임계치까지 슬로우 스톱 단계로 진행한다. 이 과정을 통해서 최대 혼잡 윈도우가 변함에 따라 조정되어야 할 윈도우 크기를 정확하게 파악하며, 전체 시스템 성능에 영향을 미치지 않고 혼잡 제어를 할 수 있다.

3.3 혼잡 제어 정책의 구현

본 논문에서 제안하는 혼잡 제어 기법은 네트워크의 프로토콜 계층 중에 전송 계층에서 제공된다. 이처럼 네트워크 전송 계층에 본 논문에서 제안하는 혼잡 제어 정책을 구현하기 위해서 그림 18과 같이 리눅스 커널 2.4.19 버전의 네트워크 부분을 수정 및 추가하였다. 그리고 DCCP 모듈은 기존의 P. McManux의 "Linux DCCP implementation"을 이용하여 본 논문에서 제안하는 혼잡 제어 정책을 추가하여 구현하였다[15]. 리눅스 커널의 전송 계층은 BSD 소켓 계층과 INET 소켓 계층의 계층적인 인터페이스의 호출에 의해서 데이타가 전송되고, 또한 전송 계층은 IP 계층의 ip_local_deliver() 함수의 호출로 패킷을 전송하고, IP 계층의 ip_built_xmit() 함수의 호출에 의해서 패킷을 수신 할 수 있다.

그림 18에서는 리눅스의 전통적인 계층적 프로토콜 구조와 DCCP 프로토콜이 포함 된 전송 계층을 보인다. DCCP는 기존의 전송 계층 프로토콜인 TCP나 UDP와 같이 INET 소켓 계층의 직접적인 호출에 의해서 수행된다.

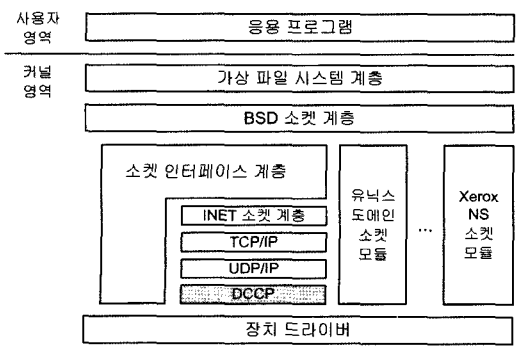

그림 18 리눅스의 계층적인 프로토콜 구조

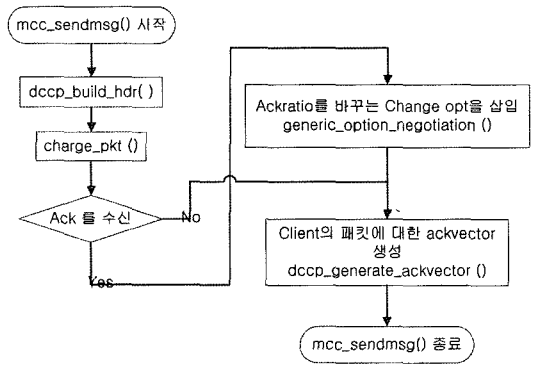

그림 19 적응적 혼잡 제어 정책의 sendmsg() 함수

IP 계층에서 전송 계층으로 데이타를 보내기 위해서는 전송 계층의 rcv() 함수를 호출하고 혼잡 제어 정책을 수행하지만 응용 계층의 send함수에 의해서 호출되는 전송 계층의 sendmsg() 함수는 혼잡 제어 정책을 호출하지 않는다. 송신인 경우에 데이타가 응용 계층에서 전송 계층으로 전송하여 설정된 혼잡 윈도우 값에 따라 패킷을 전송하기만 하고 수신인 경우에는 IP 계층에서 전송 계층의 rcv() 함수를 호출하여 수신된 Ack 벡터에 대한 혼잡 제어를 실시하기 때문이다.

DCCP에서는 다양한 혼잡 제어 정책을 사용할 수 있으므로 각 혼잡 제어 정책에 따른 전송 계층의 sendmsg() 함수와 rcv() 함수는 각각 존재하여야 한다. 그러므로 본 논문에서 제안하는 혼잡 제어 정책은 새롭게 구현된 sendmsg() 함수와 rcv() 함수에 의해서 호출된다.

그림 19는 sendmsg() 함수의 구조를 보이며 서버가 Ack 벡터 변경 옵션을 송신할 경우에 change 옵션에 포함시키는 기능을 generic_option negotiation() 함수에서 수행하고 클라이언트 패킷에 대한 ack 벡터를 생성하는 부분은 dccp_generate_ackvector() 함수에서 수행한다.

그림 20에서는 rcv() 함수의 구조를 보이며 generic_option_parse() 함수에서는 실제 수신된 Ack 벡터 변경 옵션에 대한 설정을 담당한다. 실제로 전송 윈도우 크기를 조정하는 혼잡 제어 정책을 수행하는 부분은 mcc_

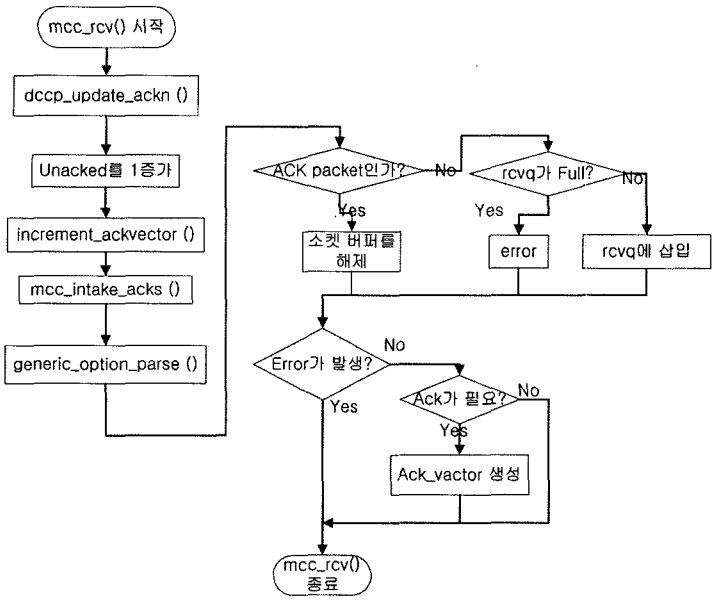

그림 20 적응적 혼잡 제어 정책의 rcv() 함수

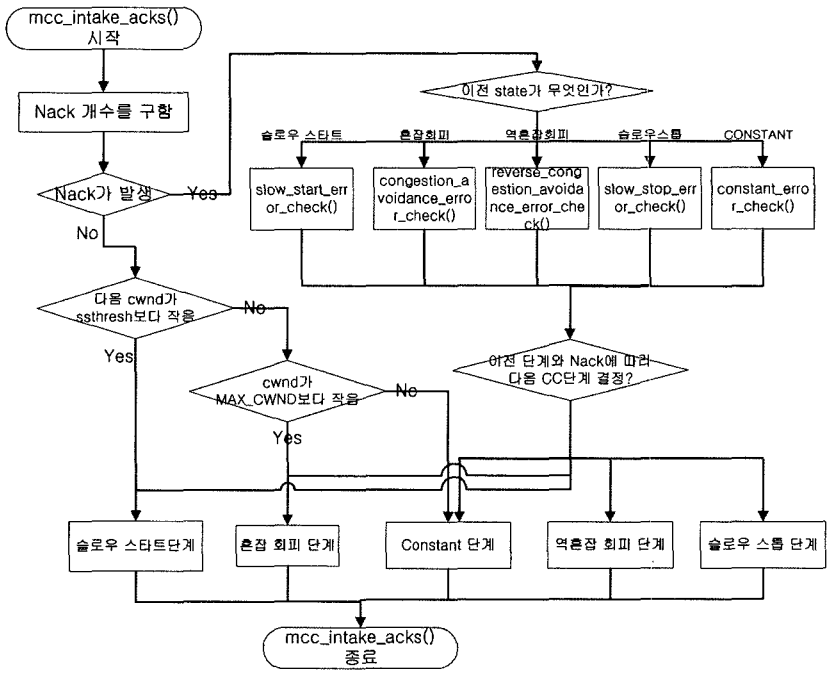

그림 21 mcc_intake_acks() 함수

intake_acks() 함수 부분이다.

그림 21은 mcc_intake_acks() 함수의 구조를 보이며 mcc_intake_acks() 함수에서는 본 논문에서 정의한 5상태에서의 혼잡 제어 처리과정을 보인다. 그리고 전송 윈도우 크기의 변경은 각각의 단계에서 수행한다.

4. 실험 및 성능 평가

본 논문에서 제안한 혼잡 제어 정책에 대하여 시뮬레이션으로 성능을 평가하였다. 표 2는 시뮬레이션을 위한 실험환경과 실험인자이다. 무선 실험망의 구조는 그림 2

표 2 실험환경

구분	실험 환경 인자	실험 측정 단위
게이트웨이	혼잡 윈도우 크기	2000
베이스스테이션 #1	혼잡 윈도우 크기	1000
	노드 수	10
베이스스테이션 #2	혼잡 윈도우 크기	300
	노드 수	25
베이스스테이션 #3	혼잡 윈도우 크기	700
	노드 수	10
	비트 에러율(Bit Error Rate)	10^{-3}, 5×10^{-2}, 10^{-2}

와 같으며 혼잡의 발생여부에 따른 성능의 평가를 위해 상대적으로 성능이 낮은 베이스스테이션에 혼잡이 발생토록 하였다. 본 논문에서의 혼잡 제어 정책은 무선 환경을 고려하고 있으므로, 무선 채널의 특성에 따른 임의의 비트 에러율을 10^{-3}~10^{-2} 사이에서 조절하여 혼잡 제어 성능을 확인하였다.

본 실험에서는 혼잡 제어 정책의 성능 비교를 위하여 TCP-like 혼잡 제어 정책(CCID #2)을 이용하였다.

그림 22~그림 24는 각각 비트 에러율이 10^{-3}, 5×10^{-2}, 10^{-2}일 경우의 성능을 비교하여 보여준다. DCCP에서는 TCP-like 혼잡 제어 정책(CCID #2)외에도 TCP Friendly 혼잡 제어 정책(CCID #3)을 표준으로 제정하였으나, 본 논문에서 제안하는 혼잡 제어 정책의 비교인자가 서로 다르기 때문에 성능 비교에서는 제외하였다. 본 논문의 혼잡 제어 정책은 지연 시간을 고려하지 않지만 CCID #3에서는 지연 시간을 고려하여 혼잡 제어 정책을 수행한다. 그러나 CCID #2는 본 논문에서 제안하는 혼잡 제어 정책과 유사하게 단말의 패킷 수신 정도에 따라 혼잡 제어 정책을 수행하므로 비교모델로 적합하다. CCID #2의 성능은 각 베이스스테이션별로 최대의 대역폭 활용도를 비교하였으며, 전체적인 게이트웨이의 최대 대역폭 활용도도 비교하였다. 그림 22는 비트 에러율이 10^{-3}으로 상대적으로 비트 에러가 낮은 경우에 해당하므로, 비트 에러에 따른 혼잡으로 오인할 확률이 떨어지므로, 베이스스테이션 2의 최대 용량까지 도달하여 혼잡에 따른 패킷 손실 상태가 계속됨을 확인할 수 있다. 이때 TCP-like 혼잡 제어 정책보다 제안하는 혼잡 제어 정책이 베이스스테이션 2의 대역폭을 최대에 가깝게 활용함을 확인할 수 있다.

베이스 스테이션 #1과 베이스 스테이션 #3은 최대 용

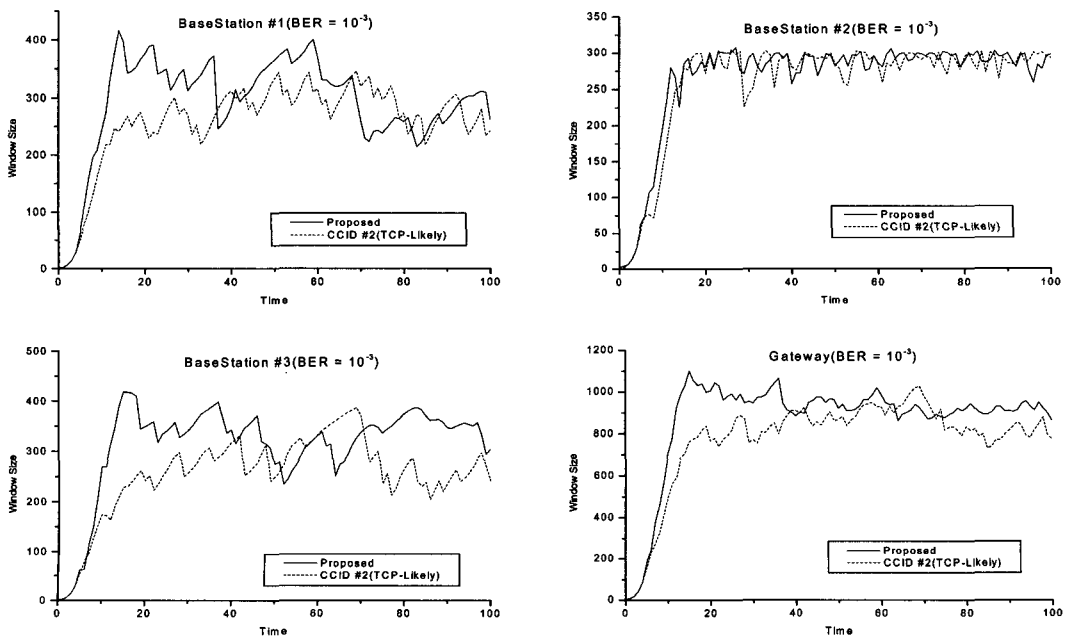

그림 22 TCP-like 혼잡제어 정책과의 Throughput 비교 (BER = 10^{-3})

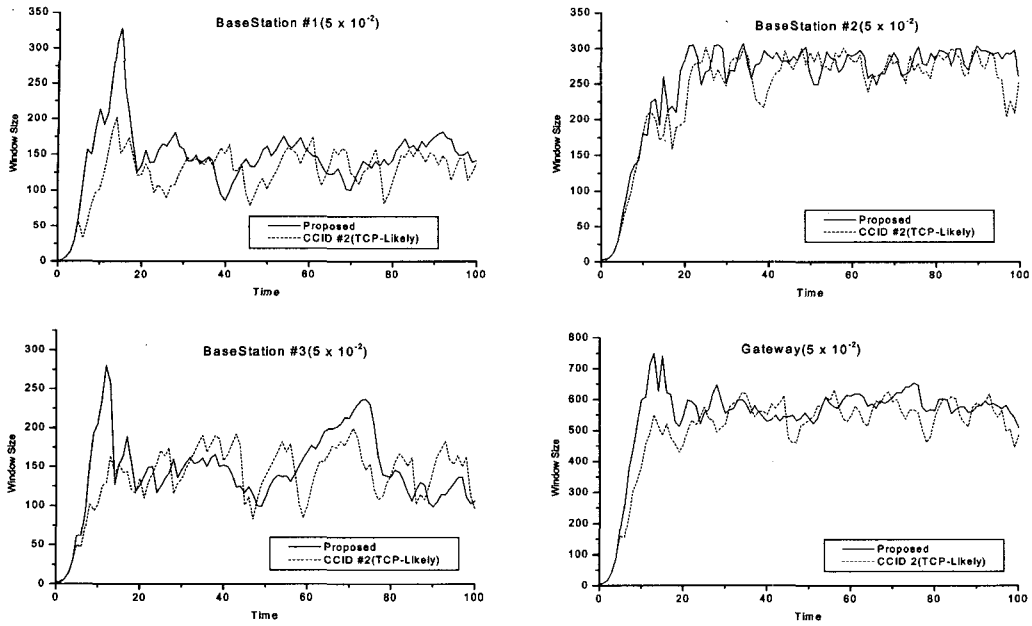

그림 23 TCP-like 혼잡제어 정책과의 Throughput 비교 (BER = 5×10^{-2})

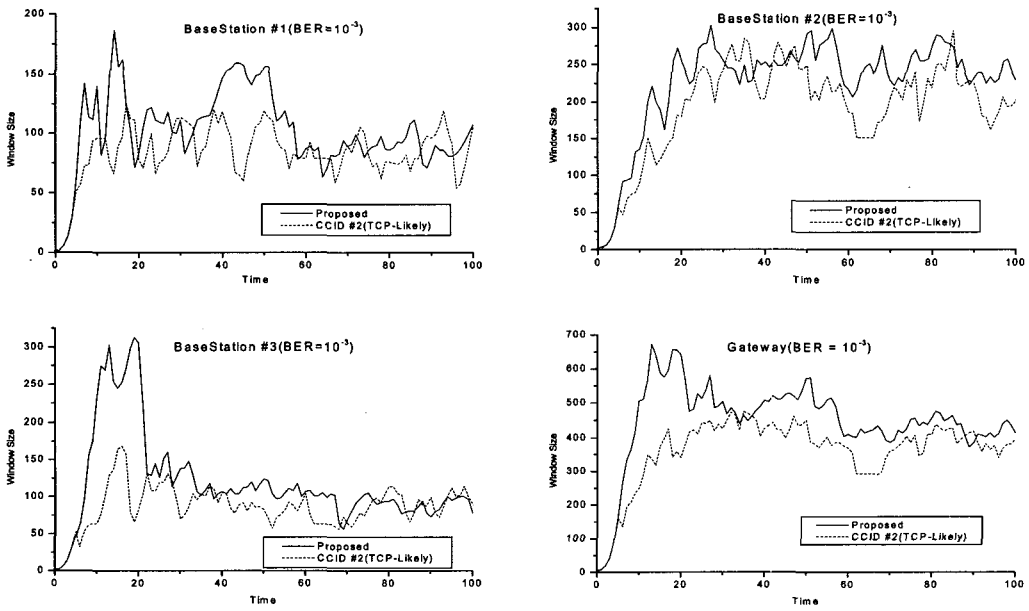

그림 24 TCP-Like 혼잡 제어 정책과의 Throughput 비교(BER = 10^{-2})

량까지 도달하지 않으므로, 혼잡에 의한 패킷 손실이 발생하지 않는다. 위의 그림을 통해서 확인할 수 있듯이, 제안하는 혼잡 제어 정책이 TCP-like 혼잡 제어 정책보다 더 좋은 대역폭 활용도를 보인다.

그림 23은 그림 22에 비해서 상대적으로 비트 에러율이 높아졌으나, 비트 에러에 의한 패킷 손실을 혼잡에 의한 패킷 손실로 오인할 확률보다는 혼잡에 따라서 실제적으로 손실이 발생될 확률이 높음을 보여주고 있다. 베이스스테이션 #2의 경우는 최대 대역폭 활용도까지 계속해서 대역폭을 이용하고 있음을 확인할 수 있다.

그림 24는 비트 에러율이 10^{-2}로 가장 높은 에러율을 보인다. 이 때에는 그림 22, 그림 23과는 달리 베이스스

테이션에서 거의 혼잡이 발생되지 않음을 확인할 수 있다. 이는 실제 혼잡에 따라 혼잡 제어를 하기보다는 비트 에러로 인해 혼잡 제어가 계속해서 진행되기 때문에, 참가하는 모든 노드들이 만족할 만한 서비스를 기대할 수 없게 된다. 그러나, 본 논문에서 제안하는 혼잡 제어 정책은 TCP-like 혼잡 제어 정책에 비해서 훨씬 높은 최대 대역폭 활용도를 보여주고 있다. 이는 역 혼잡 회피 단계를 통해서 비트 에러와 혼잡에 따른 패킷 손실을 구분할 수 있기 때문에 TCP-like 혼잡 제어. 정책에 비해 효율적인 혼잡 제어가 가능하다.

앞의 실험 결과를 통해서 본 논문에서 제안한 혼잡 제어 정책이 TCP-like 혼잡 제어 정책에 비해 훨씬 나은 성능을 보인다. 혼잡 제어를 수행하게 되는 게이트웨이의 혼잡 처리 시간을 비교해 봄으로써, 혼잡 제어기법의 효율성을 확인 할 수 있다.

표 3에서는 본 논문의 혼잡 제어 정책과 TCP-like 혼잡 제어 정책의 수행에 필요한 계산 시간을 나타내고 있다. 각 노드당 혼잡 제어를 위해서 필요한 평균적인 추가 시간은 3μs로 TCP-like 혼잡 제어 정책에 비해서 크지 않으며, 게이트웨이 관점에서는 무시할 만한 시간이다. 그리고 제안하는 혼잡 제어 기법은 게이트웨이에서 적용되며 유비쿼터스 환경에서 사용되는 경량 단말의 경우에는 단순한 패킷 수신 부분과 Ack 벡터 생성 부분만을 포함하기 때문에 다른 혼잡 제어 정책보다 실행 속도가 빠르다.

무선 접속망에서의 혼잡 제어뿐만 아니라 유.무선이

표 3 제안 기법과 TCP-Like의 혼잡 제어 처리 시간

구분	제안 기법	TCP-like
평균 처리 시간	39.9μs	36.9μs

혼재된 상황에서의 성능을 평가하기 위하여 여러 라우트를 거치면서 발생되게 되는 패킷 손실를 대상으로 실험하였으며, 패킷 손실 확률은 10^{-2}으로 설정하여 실험하였으며 실험 결과를 그림 25에 보인다.

그림 25에서 알 수 있는 것과 같이 본 논문에서 제안하는 혼잡 제어 정책이 유선망에서도 CCID #2보다 우수하다. 먼저 혼잡의 상황을 동일하게 판단하지 않고 패킷 손실에 따라서 다양한 상황의 혼잡으로 가정하여 처리하기 때문에 혼잡 제어 정책의 여러 단계 중에서 슬로우 스톱 단계와 역 혼잡 회피 단계는 일반적인 유선 네트워크에서도 적용 가능하다.

본 논문에서 제안하는 기법과 TCP-like 혼잡 제어 기법은 특정 시간에 전송되는 윈도우 크기 변화에 의한 네트워크 대역폭 이용률의 차이를 나타내므로, 기법에 따라 발생되는 에러의 구간이나 에러 발생률이 달라질 수 있다. 따라서 두 기법의 효율성을 조사하기 위해 수신측에서 전송받은 시퀀스의 특성을 비교하여 전송의 효율을 확인하였다. 실험결과는 표 4에 나타나 있다. 표 4에서 BER이 0.001인 경우에 제안 기법은 TCP-like에 비하여 약 8000개의 패킷을 더 전송할 수 있지만 패킷 손실은 단지 4개만 증가하였고 수신 불가능한 패킷율은 0.04%정도 향상되었다. 그리고 BER이 0.01인 경우에는

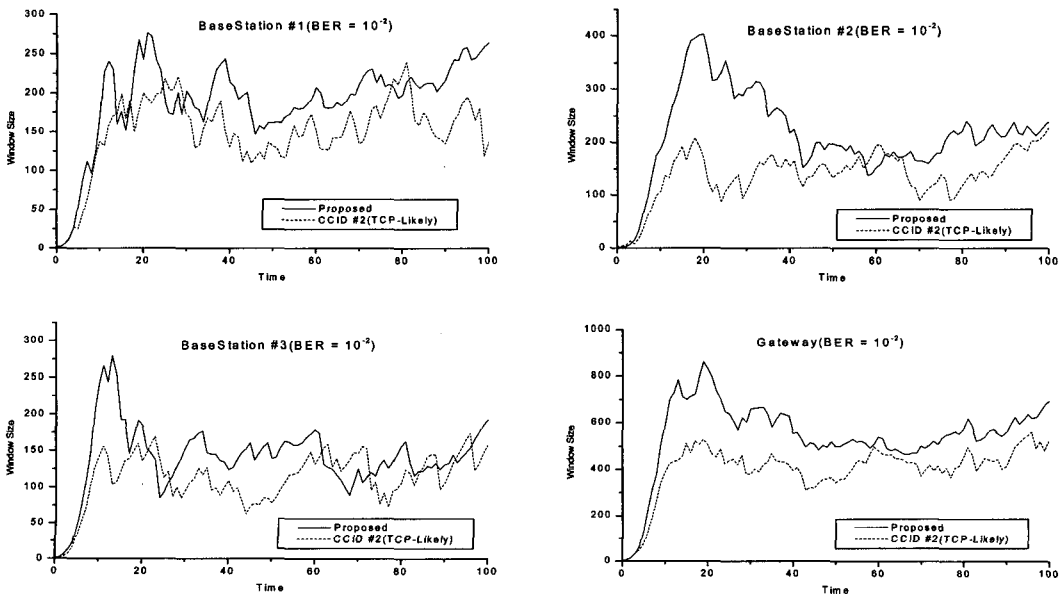

그림 25 TCP-like혼잡 제어 기법과의 Throughput 비교 (유선 환경)

표 4 제안 기법과 TCP-Like의 송수신된 데이타의 특성

구분	BER = 0.001		BER = 0.005		BER = 0.01	
	TCP-like	제안기법	TCP-like	제안기법	TCP-like	제안기법
전송한 총 패킷 수	85933	93468	52387	60876	40627	51167
사용 불가능한 패킷 수	87	91	280	302	425	564
수신 불가능 패킷율(%)	0.101	0.097	0.534	0.496	1.046	0.922

0.1%향상되었다. 이는 본 논문에서 제안하는 기법이 TCP-like에 비해 패킷 손실율을 향상시킬 뿐만 아니라 네트워크 대역폭 이용률이 현저히 개선시키므로, 효율적인 전송을 수행함을 확인할 수 있다.

5. 결론 및 향후 연구

본 논문에서는 유비쿼터스 컴퓨팅 환경에서 모바일 및 무선 접속망에 적합한 전송 계층의 혼잡 제어 기법을 제안하였다. 본 논문에서 제안하는 기법은 단말의 이동에 따른 혼잡 상태를 제어 할 수 있다. 그리고 이 기법은 IETF에서 표준으로 제정 중인 DCCP를 기반으로 설계하고 구현하였다. 본 기법에서는 기존의 혼잡 제어 기법에 무선망의 특성으로 인하여 발생하는 비트 에러를 기존의 혼잡 상태와 구분하기 위한 역 혼잡 회피 단계와 혼잡 제어에 따른 혼잡 윈도우 감소 시 발생하는 대역폭의 낭비를 줄이기 위한 슬로우 스톱 단계를 추가하였다. 그리고 제안하는 혼잡 제어 기법은 패킷의 손실 정도를 측정하는 새로운 방법을 포함한다. 또한 비트 에러와 혼잡을 구분하여 세분화 된 혼잡 제어 정책을 제공하고, 다양한 혼잡 상태에 따라서 적합한 혼잡 제어 정책을 적용한다.

본 논문에서는 제안하는 혼잡 제어 기법의 성능을 평가하기 위해서 DCCP의 CCID #2인 TCP-like 혼잡 제어 기법과 비교 실험하였으며, 실험 결과 본 논문에서 제안하는 기법은 빈번한 비트 에러가 발생하는 무선 접속망 환경에서 CCID #2보다 우수한 성능을 보였고, 또한 비트 에러가 많이 발생하지 않는 환경에서도 더 좋은 성능을 보였다. 그리고 두 기법간의 처리 시간을 비교한 결과 CCID #2가 약 3μs정도 빠른 수행 시간을 보였지만 이는 혼잡 제어 기법을 수행하는 게이트웨이에서는 무시할 만큼 작은 시간이다.

제안하는 기법은 현재 전송자 기반의 혼잡 제어 기법으로서 모든 혼잡 제어 정책을 게이트웨이에서 수행하기 때문에 각 단말은 Ack 벡터를 수집하여 전송하는 아주 간단한 기능만을 수행한다. 이러한 이유 때문에 단말의 프로토콜 스택은 경량화 될 수 있다. 향후 수신자 기반의 혼잡 제어 정책을 수행 할 수 있는 기능을 추가하고, 또한 신뢰성 보장을 위한 다양한 기법에 대하여 연구할 것이다.

참 고 문 헌

[1] 주상돈, "유비쿼터스 혁명이 시작되다." 전자 신문 2003.1.1.

[2] 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.

[3] R. Rejaie, M. Handley, and D. Estin, "Rap: An end-to-end rate-based congestion control mechanism for realtime streams in the internet," Proc. IEEE Infocom, March 1999.

[4] 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.

[5] D. Sisalem and A. Wolisz, "LDA+ TCP-friendly adaptation: A measurements and comparison study," Proc. International Workshop on Network and Operating systems Support for Digital Audio and Video , NOSSDAV'99, June 2000.

[6] Eddie Kohler, Mark Handley, Sally Floyd, "Datagram Congestion Control Protocol (DCCP)," Internet-Draft draft-ietf-dccp-spec-09.txt, November 2004.

[7] 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.

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

[9] S. Floyd, M. Handley, J. Padhye, J. Widmer, "Equation-Based Congestion Control for Unicast Applications," Proc SIGCOMM 2000, August 2000.

[10] W. Richard Stevens, "TCP/IP Illustrated, Volume1, The Protocols," Addision-Wesley.

[11] W. Richard Stevens, "TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms," RFC 2001, January 1997.

[12] K. Brown and S. Sin호, "M-TCP:TCP for Mobile Cellular Networks, ACM Computer Communication Review," May 1997.

[13] Ajay V. Bakre, and B. R. Badrinath, "I-TCP: Indirect TCP for Mobile Host," Proc 15th Int'l Conf. on Distributed Computing Systems, May 1995.

[14] Kevin Brown, Suresh Singh, "M-UDP: UDP for

mobile cellular networks," ACM SIGCOMM Computer Communication Review, vol. 26, pp.60-78, October 1996.
[15] "http://www.suck5ong.com:81/dccp," P.McManus-Linux DCCP implementation, 30 May 2002.

박 시 용

1997년 경성대학교 전자계산학과(학사) 2001년 부산대학교 대학원 멀티미디어과(이학 석사). 2003년 부산대학교 대학원 전자계산학과 박사과정 수료. 2005년 부산대학교 대학원 전자계산학과(이학 박사). 관심분야는 멀티미디어, 모바일 네트워크, 인터넷 QoS, 유비쿼터스 컴퓨팅

김 성 민

2001년 부산대학교 전자계산학과 졸업 2003년 부산대학교 전자계산학과 석사 2003년~현재 부산대학교 컴퓨터공학과 박사과정. 관심분야는 트랜스코딩, 멀티미디어 스트리밍, 모바일 네트워크

이 태 훈

2004년 부산대학교 정보컴퓨터공학과(학사). 2004년~현재 부산대학교 정보컴퓨터공학과(석사과정). 관심분야는 임베디드시스템, 무선 네트워크

정 기 동

1973년 서울대학교 응용수학과(학사). 1975년 서울대학교 대학원 전자계산학과(이학 석사). 1986년 서울대학교 대학원 전자계산학과(이학 박사). 1990년~1991년 MIT 대학 교환 교수. 1995년~1997년 부산대학교 전자계산소 소장. 1999년~2001년 부산대학교 BK21단장. 1978년~현재 부산대학교 전자계산학과 교수. 관심분야는 멀티미디어, 모바일 네트워크, 인터넷 QoS, 유비쿼터스 컴퓨팅