트리 기반의 다대다 신뢰적 멀티캐스트를 위한 효율적인 혼잡 제어 기법

(An Efficient Congestion Control Mechanism for Tree-based Many-to-many Reliable Multicast)

유 제 영 † 강 경 란 †† 이 동 만 ††
(Je-Young Yu) (Kyungran Kang) (Dongman Lee)

요 약 혼잡 제어는 에러 제어와 함께 신뢰적 멀티캐스트의 핵심적인 기능이다. MTCP나 TRAMCC와 같은 기존의 트리 기반의 혼잡 제어 기법은 일대다 신뢰적 멀티캐스트를 위하여 설계되어 다대다 신뢰적 멀티캐스트에 적용할 경우 몇몇 문제점들이 나타난다. 본 논문에서는 트리 기반의 다대다 신뢰적 멀티캐스트 프로토콜을 위한 효율적인 혼잡 제어 기법을 제안한다. 제안하는 기법은 혼잡 윈도우 기법을 기반으로 하며 전송률 제어기를 추가적으로 사용한다. 수신자들의 처리 부담을 최소화하기 위하여 추가적인 피드백 없이 에러 복구를 위한 피드백을 혼잡 제어를 위하여 이용하며, 동적으로 네트워크의 상태의 변화를 반영하는 ACK 타이머와 NACK 타이머 및 빠른 전송률 복구 기법 등을 통하여, 세션 내의 흐름들 간의 공평성을 제공한다. 네트워크 시뮬레이터를 사용해서 제안하는 기법이 세션 내의 흐름 간 공평성에 있어서 기존의 TRAMCC보다 효과적인 것을 보였으며, TCP-친화성, 응답성, 확장성에 있어서 만족할 만한 성능을 보임을 확인하였다. 그리고, 신뢰적 멀티캐스트 프로토콜인 GAM에 통합 구현하여 실험실 내 시험 네트워크 상에서 실험을 수행하였다.

키워드 : 혼잡 제어, 다대다 신뢰적 멀티캐스트

Abstract Congestion control is a key task in reliable multicast along with error control. However, existing tree-based congestion control schemes such as MTCP and TRAMCC are designed for one-to-many reliable multicast and have some drawbacks when they are used for many-to-many reliable multicast. We propose an efficient congestion control mechanism, TMRCC, for tree-based many-to-many reliable multicast protocols. The proposed scheme is based on the congestion windowing mechanism and a rate controller is used in addition. The feedback for error recovery is exploited for congestion control as well to minimize the overhead at the receivers. The ACK timer and the NACK timers are set dynamically reflecting the network condition changes. The rate regulation algorithm in the proposed scheme is designed to help the flows sharing the same link to achieve the fair share quickly. The performance of the proposed scheme is evaluated using ns-2. The simulation results show that the proposed scheme outperforms TRAMCC in terms of intra-session fairness and shows good level of responsiveness, TCP-friendliness, and scalability. In addition, we implemented the proposed scheme by integrating with GAM that is one of many-to-many reliable multicast protocols and evaluated the performance in a laboratory-wide testbed.

Key words : Congestion Control, Many-to-many Reliable Multicast

1. 서 론

인터넷이 확산됨에 따라 네트워크 가상환경, 네트워크 게임, 협동 웹 캐싱과 같은 다자간 협동 어플리케이션과 프로세스간 상호작용이 요구되는 어플리케이션에 대한 관심이 증가하고, 이에 따라 다대다(many-to-many) 신뢰적 멀티캐스트의 필요성이 증대되었다. 다대다 멀티캐스트를 요구하는 응용은 VOD, 데이타 분배 등에서와 같이 하나의 송신자와 다수의 수신자를 지원하는 일대다 멀티캐스트 응용과 달리 각 참가자가 수신자의 역할뿐만 아니라 송신자의 역할을 담당한다는

† 비 회 원 : 삼성탈레스 연구원
superjayyu@hotmail.com
†† 정 회 원 : 한국정보통신대학원대학교 공학부 교수
korykang@icu.ac.kr
dlee@icu.ac.kr
논문접수 : 2003년 2월 28일
심사완료 : 2003년 7월 21일

특징을 갖는다. 이러한 특징을 고려하여 다대다 세션을 위한 신뢰적 멀티캐스트 프로토콜로 Scalable Reliable Multicast SRM)[1], LORAX[2], Group Aided Multicast(GAM)[3]과 같은 Automated Repeat Request (ARQ) 기반의 기법들이 제안되었으며, 이들 중 트리 기반의 멀티캐스트 프로토콜이 전송량 및 대역폭 사용에 있어서 가장 효율적이라는 것이 이전 연구[4]에서 입증되었다.

혼잡 제어(congestion control)는 에러 제어와 함께 신뢰적 멀티캐스트 프로토콜의 핵심 기능 중 하나로 간주된다[5]. 적절한 혼잡 제어 기법을 가지지 않은 멀티캐스트 프로토콜은 TCP 흐름(flow)에 비해 네트워크 대역폭을 불공평하게 많이 점유하게 되고 결국에는 네트워크 동작이 중단되는 문제 상황(congestion collapse)을 일으킬 수 있다[6]. 그러므로, 혼잡 제어 기법에서는 회선 상에 공존하는 TCP 흐름과의 공평한 대역폭 사용을 보장할 수 있어야 한다. 또한, 멀티캐스트에서의 혼잡 제어는, 멀티캐스트 세션 내의 하나 이상의 수신자들이 서로 상이한 네트워크 환경에 존재하므로, 유니캐스트에서의 혼잡 제어 기법과는 달리 수신자의 네트워크 송수신 능력 등의 상이성이 중요하게 다루어져야 한다. 이러한 요구 사항들을 기반으로, 트리 기반의 신뢰적 멀티캐스트를 위해 MTCP[7]나 TRAMCC [8]와 같은 혼잡 제어 기법들이 제안되었다

다대다 신뢰적 멀티캐스트 세션에서는 참가자의 수가 늘어날수록 수신자의 처리 부담이 비례적으로 증가하게 되므로, 프로토콜 설계에 있어서 수신자의 처리 부담을 무시할 수 없다[10]. 그리고, 하나의 회선을 여러 데이타 흐름과 제어 흐름이 공유하게 되어 제한된 대역폭에 대한 경쟁이 일대다 세션에 비해 크게 증가하게 되고, 이로 인해 데이타의 손실뿐 아니라 피드백과 재전송 패킷의 손실을 가져오게 된다. 그러나, MTCP와 TRMACC는 근본적으로 일대다 신뢰적 멀티캐스트를 위하여 설계되어 다대다 세션의 경우에 효과적으로 대응하지 못한다. MTCP의 경우를 살펴 보면, 각 수신자가 매 송신자들에 대해 해당 송신자들이 전송률을 조절하게 하기 위하여 거의 모든 데이타 패킷에 대하여 ACK을 생성해야만 하므로, 수신자의 처리 부담이 송신자의 수가 증가함에 따라 함께 크게 증가하며, 특히 복구 트리의 중간에 위치한 수신자들은 개별 송신자로 전달되는 ACK들을 종합해야 하므로 처리 부담이 심각하게 증가한다. TRAMCC는 전송률 조절 기법 설계에서 한 회선 내에서 대역폭을 경쟁하는 흐름들을 고려하고 있지 않으므로, 서로 경쟁하는 흐름들이 공평한 대역폭 분할에 대응되는 전송률을 회복하는데 많은 시간이 걸리거나 공평한 대역폭 분할을 유지하지 못하게 된다.

본 논문에서는 앞서 제시한 혼잡 제어 기법의 기본적인 요구 사항을 만족시키며 기존의 혼잡 제어 기법이 갖고 있는 문제점들을 해결한 혼잡 제어 기법으로 Tree-based Many-to-many Reliable multicast Congestion Control(TMRCC)을 제안한다. 제안하는 기법은 *Hu*라 칭해지는 '유니캐스트 NACK/재전송, 정기적 ACK 기법'[10]을 사용하는 트리 기반의 다대다 신뢰적 멀티캐스트와 함께 동작하도록 설계되었다. *Hu*가 세션 전송량과 대역폭 사용에 있어서 ARQ 기반의 다대다 신뢰적 멀티캐스트에서 최적의 성능을 보이는 것으로 앞선 연구[10]에서 입증되었다. 제안하는 기법은 TRAMCC와 유사하게 전송률 제어기를 동반한 혼잡 윈도우 기법을 사용한다. 전송률 제어기를 사용함으로써 순수하게 윈도우 기법만을 기반으로 하는 경우 발생할 수 있는 자체 시간 조절 기능(self-clocking)의 상실 문제를 해결할 수 있다[12]. 수신자의 처리 부담을 최소화하기 위하여, 혼잡 제어를 위한 추가적인 피드백 없이, 손실 복구를 위한 피드백을 활용한다. 그리고, 피드백과 재전송 패킷의 손실에 의해 발생되는 문제 상황을 피하기 위하여, 다음의 두 가지 방법을 사용한다. 첫째, 수신자들로부터의 피드백이 도착하지 않아 데이타 패킷 전송이 일시적으로 중단된 경우, 송신자는 전송률을 조절하는데 있어서 실제 적용된 전송률과 예상 전송률을 기반으로 하여 다음 전송률을 계산한다. 동시에, ACK 타이머와 NACK 타이머의 값을 네트워크 상태에 따라 동적으로 조절함으로써, 중복된 재전송 및 피드백 요청이나 불필요한 대기 시간을 피하여 송신자가 전송률을 올바르게 조절할 수 있도록 한다. 네트워크 시뮬레이터 ns-2[11]를 이용하여 제안하는 기법의 성능을 평가하였으며, 실험 결과를 통해 세션 내 공평성에 있어서 TRAMCC는 세션 내 플로우들이 불균형된 전송률을 보인 반면 제안하는 기법은 플로우들이 유사한 전송률을 유지하였고, 만족할 만한 수준의 네트워크 대역폭 변화에 대한 응답성과 TCP 친화성 및 확장성을 보였다. 시뮬레이션과 별도로 TMRCC를 트리 기반의 다대다 신뢰적 멀티캐스트 프로토콜인 GAM[3]에 통합 구현하여 시험 네트워크 상에서 세션 내 공평성과 TCP 친화성을 실험하였으며, 시뮬레이션에서와 유사한 결과를 얻었다.

본 논문의 구성은 다음과 같다. 2장에서 기존의 트리 기반의 단일 전송률 신뢰적 멀티캐스트 혼잡 제어 기법을 살펴본다. 3장에서는 제안하는 혼잡 제어 기법인 TMRCC을 기술하며, 네트웍 시뮬레이터를 이용한 TMRCC의 성능 평가 결과를 4장에 보인다. 5장에서는 제안하는 혼잡 제어 기법 클래스 구현 및 시험망에서의 실험 결과에 대하여 기술하며, 마지막으로 6장에서 결론 및 향후 연구를 제시하는 것으로 본 논문을 맺는다.

2. 관련 연구

본 논문에서 제안하는 기법과 관련한 기존 연구로, 트리 기반의 단일 전송률 신뢰적 멀티캐스트를 위한 혼잡 제어 기법인 MTCP[7]와 TRAMCC[8]를 살펴 본다.

MTCP[7]는 윈도우 기반의 혼잡 제어 기법을 사용하는 신뢰적 멀티캐스트 프로토콜이다. 혼잡 제어를 위하여, MTCP는 복구 트리의 말단 노드를 제외한 모든 노드가 혼잡 윈도우와 전송 윈도우를 관리한다. 혼잡 윈도우의 크기는 슬로우 스타트 및 혼잡 회피 알고리즘을 포함하여 TCP[9]와 비슷하게 관리된다. 전송 윈도우는 자식 노드들로부터 응답되지 않은 패킷의 수를 나타낸다. 전송 윈도우의 크기는 송신자로부터 새로운 패킷을 받을 때마다 증가하며, 자식 노드로부터 ACK을 받을 때마다 감소한다. 각 노드는 복구 트리의 부모 노드에게 혼잡 요약(congestion summary)을 각 ACK에 포함하여 전송한다. 혼잡 요약에는 자신과 자신의 자식 노드들의 혼잡 윈도우 크기의 최소값(*minCwnd*) 및 전송 윈도우 크기의 최대값(*maxTwnd*)가 포함되며, 송신자는 *minCwnd*와 *maxTwnd*의 차이만큼의 데이타 패킷만 전송한다. 각 노드들은 매 송신자에 대해 매 패킷마다 혼잡 요약을 작성해서 보고해야 하므로 다대다 세션과 같이 송신자의 수가 늘어나게 되면 그 처리 부담이 비례적으로 증가하여 세션 전체의 성능을 떨어뜨리게 된다.

TRAMCC[8]는 TRAM[12]을 위한 혼잡 제어 기법으로, 윈도우 기반의 혼잡 제어 기법과 함께 전송률 제어기를 사용하므로 세션 참가자 수 증가로 인한 처리 부담 증가는 MTCP만큼 크지 않다. 송신자는 수신자들의 혼잡 윈도우가 계속 열려 있는 상태가 되도록 전송률을 조절한다. TRAMCC는 기본적으로 피드백 및 피드백 조합 부분, 윈도우 조절 부분, 전송률 조절 부분의 세 부분으로 이루어져 있다. 혼잡 윈도우의 크기는 각 수신자에서 관리되며, 수신자가 혼잡을 인식할 때마다 AIMD 방식으로, 증가 시에는 일정량만큼 증가시키며 감소 시에는 배수로 감소시킨다. ACK 윈도우는 수신자들이 하나의 ACK을 보내는 단위로써, 수신자는 ACK 윈도우 만큼의 연속적인 데이타 패킷을 수신했을 때에서야 복구 트리의 부모 노드에게 ACK을 전송한다. 현재의 ACK 윈도우 동안의 손실된 데이타 패킷의 수가 바로 전 ACK 윈도우보다 많고, ACK 윈도우의 25% 이상의 데이타 패킷을 손실했을 경우를 '혼잡'이라고 판단한다. 수신자가 혼잡을 인식하면, 혼잡 윈도우의 크기는 현재 크기의 4분의 1로 줄어들며, 그렇지 않을 경우, 두 패킷만큼 증가한다. 수신자로부터의 매 피드백에는 수신자의 혼잡 윈도우의 크기에 대한 정보가 연속적으로 받은 가장 높은 패킷 일련 번호와, 수신자가 현재 받을 수 있는 가장 높은 패킷 일련 번호로써 표현되어 포함되어 있다. 전송률은 최근 5 초 동안의 평균 전송률을 기준으로, 송신자가 매 ACK 윈도우만큼의 패킷을 보낼 때마다 조절된다. 어떤 순간에 송신자가 수신자들로부터의 ACK 없이 보낼 수 있는 최대 패킷 수를 '열린 윈도우'라고 정의하고, 이 열린 윈도우 크기의 평균이 지난 두 ACK 윈도우 동안 증가했을 경우, 전송률을 증가시키며, 그렇지 않을 경우 최근 5 초 동안의 평균 전송률을 그대로 사용한다. 이러한 전송률 조정 정책은, 특정 흐름이 일시적인 피드백 손실로 인하여 열린 윈도우의 크기가 0이 되고 이로 인해 실제 전송률이 급격히 떨어지는 경우, 다른 흐름과 유사한 전송률로의 회복이 어렵게 되는 문제점을 갖는다.

3. 트리 기반의 다대다 신뢰적 멀티캐스트를 위한 혼잡 제어 기법

3.1 고려사항

다대다 신뢰적 멀티캐스트 세션의 개별 참가자는 송신자임과 동시에 다른 참가자들의 수신자이므로 송신자의 처리 부담을 단순히 수신자들에게 분산시키는 기법이 언제나 전체 세션의 효율(throughput)을 증가시킨다고 볼 수 없다[10]. 그리고, 하나의 다대다 세션에서는 데이타 흐름(flow)과 제어 흐름들이 하나의 회선을 공유하며 대역폭을 경쟁할 수 있다. 이것은 피드백이나 재전송 패킷의 손실을 증가시킬 수 있으며, 순간적으로 일부의 데이타 흐름이 다른 데이타 흐름에 비해 더 많은 손실을 겪을 수 있다. 이러한 특징을 효과적으로 대처하지 못할 경우, 세션 내의 흐름 간 불공평한 전송률을 야기할 가능성이 크다. 정리하면, 트리 기반의 다대다 멀티캐스트 세션을 위한 혼잡 제어 기법 설계 시에 다음의 네 가지 사항이 고려되어야 한다.

첫째, 에러 복구와 함께 혼잡 제어를 위한 처리 부담은 각 참가자들에게 골고루 분산되어야 한다. 신뢰적 멀티캐스트 프로토콜은 반드시 최소한의 처리 부담을 가지며, 세션의 전체적인 처리 부담을 줄일 수 있도록 설계되어야 한다.

둘째, 신뢰적 멀티캐스트 프로토콜은 이미 에러 복구를 위한 피드백 패킷에 대한 처리 부담을 가지므로, 혼잡 제어를 위한 처리 부담을 추가하는 것을 피해야 한다. 즉, 혼잡 제어 기법은 에러 복구를 위한 기존의 피드백을 최대한 이용할 수 있어야 한다.

셋째, 피드백과 재전송 패킷의 손실을 효과적으로 대처하기 위하여 ACK/NACK 타이머를 사용하되, 각 타이머 값의 동적 추정 알고리즘이 제공되어야 한다. 너무 짧은 타이머 값은 네트워크를 중복된 피드백이나 재전송 패킷으로 채울 가능성이 있으며, 너무 긴 타이머 값

은 혼잡 제어 기법의 반응을 느리게 함으로써 네트워크 상태의 변화에 대한 적응을 느리게 만들 수 있다.

넷째, 하나의 회선에 공존하며 대역폭을 경쟁하는 흐름의 수가 증가함에 따라 순간적으로 하나의 흐름은 다른 흐름들보다 높은 손실률을 나타낼 수 있으며, 피드백 손실로 인해 송신자가 전송률 제어를 효과적으로 진행하지 못하는 경우가 발생할 수 있다. 따라서, 전송률 조절 알고리즘은 개별 흐름들이 세션 내 흐름들의 공평한 전송률로 빨리 복구될 수 있도록 설계되어야 한다.

다음 절에서는 제안하는 기법과 기법에서 가정하고 있는 트리 기반의 손실 복구 기법에 대하여 간략히 살펴보고, 이어서 제안하는 기법인 TMRCC를 상세하게 설명하도록 한다.

3.2 TMRCC의 개요

본 논문에서 제안하는 TMRCC는 TRAMCC와 같이 단일 전송률의 전송률 제어기를 가진 윈도우 기반의 혼잡 제어 기법이다. 혼잡 제어 기법과 함께 사용되는 에러 복구 기법으로써 *Hu* 기법을 가정한다. 신뢰적 멀티캐스트 프로토콜은 피드백과 재전송 방법을 기준으로 하여 몇 가지로 분류가 가능하며, *Hu*는 유니캐스트 NACK/재전송, 정기적 ACK을 사용하는 신뢰적 멀티캐스트 프로토콜을 가리킨다[10]. *Hu* 기법에서 송신자는 데이타 패킷을 세션에 멀티캐스트하며, 수신자는 패킷 손실을 발견할 때마다, NACK 타이머를 설정하고 복구 트리에서의 부모 노드에게 NACK을 전송한다. 부모 노드는 NACK을 받으면 자식 노드에게 복구 패킷을 유니캐스트한다. 복구 패킷을 수신하지 못하고 NACK 타이머가 종료되었을 경우, 수신자는 패킷 손실을 처음 발견한 때와 동일하게, 다시 NACK 타이머를 설정하고 NACK을 전송한다. 수신자가 정기적으로 ACK을 부모 노드에게 유니캐스트하면, 부모 노드는 유지하고 있던 데이타 패킷들을 안전하게 메모리에서 삭제할 수 있다.

TMRCC를 간략하게 요약해서 설명하자면, 수신자 피드백과 혼잡 윈도우 조절, 전송률 조절로 나누어서 설명할 수 있다.

수신자 피드백. TMRCC는 ACK과 NACK을 에러 복구뿐만 아니라 혼잡 제어를 위하여 함께 사용한다. 정상적인 상황에서는 각 수신자는 연속적으로 받은 데이타 패킷의 수가 ACK 윈도우에 도달할 경우, 복구 트리의 부모 노드에게 ACK을 보낸다. 데이타 패킷 손실이 발견되면, 수신자는 즉시 부모 노드에게 NACK을 전송한다. ACK과 NACK은 수신자의 혼잡 윈도우에 대한 정보로서 다음의 두 가지 값을 포함한다 수신한 연속적인 데이타 패킷의 마지막 일련 번호(W_L)와 자신과 자신의 자식 노드의(만약 존재한다면) 최대 허용 데이타 패킷 일련 번호의 최소값(W_R)이다[8]. 즉, 현재 혼잡 윈도우의 크기는 W_R과 W_L의 차이로 추정할 수 있다. 피드백은 복구 트리를 따라 종합되므로, 최종적으로 송신자에게 전달되는 W_R 값은 세션 내에서 해당 송신자에 대한 전체 수신자의 W_R 값 중 최소값이 된다.

피드백과 재전송 패킷의 손실에 대처하기 위해 송신자와 수신자에서 ACK 타이머와 NACK 타이머를 사용하며, 타이머 값은 네트워크 상태의 변화에 따라 동적으로 변경된다.

혼잡 윈도우 조절. 수신자는 각 송신자를 위하여 혼잡 윈도우를 관리한다. 혼잡 윈도우의 크기는 미리 지정된 최소 윈도우 크기와 최대 윈도우 크기 범위 안에서 그 값이 변경되며, TCP의 Additive Increase and Multiplicative Decrease (AIMD)[9] 규칙과 유사하게 조절된다.

전송률 조절. 송신자는 ACK을 수신하지 않고도 데이타 패킷을 연속적으로 W_R까지 전송할 수 있다. 전송률은 ACK 윈도우만큼의 데이타 패킷이 전송될 때마다 조절된다. 전송률 조절 단계에는 '슬로우 스타트 단계'와 '안정 상태 단계'의 두 가지가 있다. 슬로우 스타트 단계에서는 알맞은 전송률을 빠르게 찾아낼 수 있도록 전송률을 조절하며, 안정 상태 단계에서는 송신자가 연속적으로 전송할 수 있는 데이타 패킷의 수가 변화하는 경향에 따라 서로 다른 규칙을 적용하여 전송률을 조절한다. 이에 대한 자세한 알고리즘은 3.5절에 기술되어 있다.

3.3 수신자 피드백

TMRCC 세션의 수신자는 ACK 윈도우만큼의 연속적인 데이타 패킷을 성공적으로 받을 때 마다 복구 트리의 부모 노드에게 ACK을 전송하며, 데이타 패킷 손실이 발견될 때마다 NACK 을 전송한다. ACK을 전송하는 경우의 조건을 수식 (1)과 같이 표현할 수 있다.

$$W_L(n, i+1) = W_L(n, i) + AW \qquad (1)$$

$W_L(n,i)$는 i번째 ACK에서 보고하는, 연속적으로 수신된 패킷의 최대 일련 번호를 의미하며, AW는 ACK 윈도우의 크기를 나타낸다. 송신자 n에 대한 각 수신자의 W_R을 의미하는 $W_R(n)$은 $W_L(n)$과 송신자 n에 대한 혼잡 윈도우 크기의 합으로 계산된다. 혼잡 윈도우에 대해서는 3.4절에서 자세하게 설명된다. 각 수신자는 자신의 $W_L(n)$과 자신과 자신의 자식 노드의 $W_R(n)$ 값 중 최소값을 피드백에 포함하여 복구 트리의 부모 노드에게 전달한다. 송신자는 자신의 자식 노드들로부터 전달받은 $W_R(n)$값의 최소값을 자신이 전송할 수 있는 패킷의 최대 일련 번호로 사용한다.

TMRCC에서는 피드백과 복구 패킷의 손실에 효과적으로 대처하기 위하여 타이머의 값을 동적으로 변경한다. 송신자에서 ACK의 손실을 인지하기 위하여는

ACK 타이머가 사용된다. 타이머가 만료되었을 경우, 송신자는 복구 트리 상에서 아직 ACK을 전송하지 않은 자식 노드에게 ACK을 명시적으로 요청한다. 이것은, 모든 수신자로부터 ACK을 수신하게 될 때까지 복구 트리를 따라 반복하여 수행된다. ACK 타이머 값의 올바른 선택을 위하여, 'ACK 응답 시간(ACK Response Time, *ART*)'를 측정한다. ACK 응답 시간이란 ACK 윈도우의 마지막 패킷이 전송된 후부터 복구 트리의 모든 자식 노드들로부터 해당 ACK 윈도우에 대한 ACK을 받기까지의 시간을 의미한다. 다음 ACK 윈도우에 사용될 '추정 ART(ART_e)'는 수식 (2)에서 표현하는 바와 같이, 관측된 ART와 이전 ACK 타이머 값을 위해 계산된 추정 ART를 사용한 Exponentially Weighted Moving Average (EWMA)로 계산된다.

$$ART_e(i+1) = (1-\alpha) \times ART_e(i) + \alpha \times ART(i),$$

단, $0 < \alpha < 1$이고 i=0,1,2,...이다. (2)

일반적으로 네트워크상 노드들의 최대 RTT는 500 ms를 넘지 않는다고 가정할 수 있으므로 [13], 초기 추정 ART인 $ART_e(0)$는 500 ms로 설정한다. 제안하는 기법에서는 α를 TCP에서 ACK 타이머 값을 설정할 때 일반적으로 사용되는 값과 같이 0.1로 설정한다 [9]. *(i+1)*번째 ACK을 위해 실제 사용되는 ACK 타임아웃 값(*ATO*)은 수식 (3)과 같다:

$$ATO(i) = 1.5 \times ART_e(i),$$ 단, i = 0,1,2,...이다. (3)

NACK 패킷의 손실이나 복구 패킷의 손실에 대처하기 위하여, TMRCC는 NACK 타이머를 사용하며, NACK 타이머 값을 결정하는데 ACK 타이머 값을 결정하는 것과 유사한 방법을 사용한다. 즉, 각 NACK에 대해 측정한 'NACK 응답 시간(NACK Response Time, *NRT*)'을 활용함으로써 NACK 타이머 값을 네트워크 상태에 동적으로 대응하도록 갱신해 갈 수 있다. 여기서 NACK 응답 시간이란 NACK을 보낸 시점부터 보낸 NACK에 대한 첫번째 복구 패킷이 도착할 때까지 걸린 시간을 말한다. 송신자 n에 대해 *(i+1)*번째로 전송되는 NACK에 대한 추정 NRT인 $NRT_e(n, i+1)$는 수식 (4)와 같이 계산된다.

$$NRT_e(n, i+1) = (1-\alpha) \times NRT_e(n, i) + \alpha \times NRT(n, i),$$

단, $0 < \alpha < 1$ 이고, i = 0,1,2,...이다. (4)

초기 추정 NRT 값은 해당 세션에 대한 '멀티캐스트 RTT (MRTT)'[14]을 사용하여 수식 (5)와 같이 계산되며, 모든 송신자에 대해 동일한 값이 적용된다. 송신자 n에 대한 초기 추정 NRT 값은 수식 (5)와 같이 정의된다.

$$NRT_e(n, 0) = MRTT \times 2 \quad (5)$$

송신자 n으로부터의 데이타 패킷에 대한 $(i+1)$번째 NACK 타이머의 값, 즉 NACK 타임아웃(*NTO*) 값은 최종적으로 수식 (6)과 같이 계산된다:

$$NTO(n, i+1) = 1.5 \times \frac{s}{R(n)} NRT_e(n, i),$$

단, i = 0,1,2,...이다. (6)

여기서, s는 평균 패킷 크기를 의미하며, $R(n)$은 송신자 n에 대한 현재 측정된 수신률을 의미한다. NACK 타임아웃(*NTO*) 값을 결정하는데 있어서, 패킷 당 지연 시간($\frac{s}{R(n)}$)을 사용하는 이유는 하나의 NACK 패킷은 하나 이상의 복구 패킷에 대한 요청이 포함될 수 있기 때문이다.

3.4 혼잡 윈도우 조절

세션의 각 수신자는 각 송신자 n에 대하여 혼잡 윈도우 크기인 $CW(n)$을 관리하고, 매 피드백에 혼잡 윈도우 정보를 담아서 부모 노드에게 알린다. 세션이 시작될 때, 초기 혼잡 윈도우의 크기는 수식 (7)과 같이 설정되고, 수신자가 ACK을 복구 트리의 부모 노드로 전송할 때마다, 수신자는 혼잡 윈도우의 크기를 조절한다. 연속적으로 받은 패킷의 최대 일련 번호가 수식 (1)에서 정의하고 있는 $W_L(n)$에 도달할 동안 데이타 패킷의 손실이 없었다면, 혼잡윈도우의 크기는 수식 (8)과 같이 증가한다. 패킷 손실이 있었다면, 수식 (9)와 같이 감소한다. 이러한 TCP와 유사한 AIMD 윈도우 조절 정책은 다음 절에서 설명하는 전송률 조절 방법과 함께 TCP 친화성을 보장한다.

$$CW(n, 0) = 3 \times AW \quad (7)$$

$$CW(n, i+1) = CW(n, i) + IF,$$

단, $IF > 0$이고, i = 0,1,2,...이다. (8)

$$CW(n, i+1) = \frac{1}{DF} \times CW(n, i),$$

단, $DF > 1$이고, I = 0,1,2,...이다. (9)

여기서, $CW(n, i)$은 송신자 n에 대한 i번째 ACK 윈도우 동안 혼잡 윈도우의 크기를 의미하며, IF는 혼잡 윈도우 크기의 증가량을 뜻하고, DF는 혼잡 윈도우의 감소 비율을 의미한다.

3.5 전송률 조절

개별 송신자의 데이타 전송률은 최소 전송률 R_{min}과 최대 전송률 R_{max}로 제한되는 범위 안에서 결정되는데, R_{min}과 R_{max}는 응용 프로그램 개발자에 의해 설정될 수 있다. 송신자는 전송하는 데이타 패킷의 패킷 일련 번호가 ACK 윈도우의 배수에 이를 때마다 다음 ACK 윈도우의 데이타 패킷에 적용할 전송률을 결정한다. 세션 초기는 '슬로우 스타트 단계'로서, 최대로 적정한 전송률에 빨리 도달하기 위해 수식 (10)과 같이 전송률을 조절한다.

$$R_e(i+1) = R_e(i) + \Delta R(i+1),$$

$\Delta(i+1) = \Delta(i) + r$

단, i = 0,1,2,...이고, r = 1kbyte/sec이다. (10)

여기서, $R_e(i+1)$은 $(i+1)$번째 ACK 윈도우의 전송률을 의미하며, i번째 ACK 윈도우의 마지막 데이타 패킷 전송 후에 사용되는 수식으로, 초기 전송률 $R_e(0)$는 최소 전송률 R_{min}을 사용하고, 초기 전송률 증가분 $\Delta R(0)$는 0이다. 슬로우 스타트 단계는 다음의 조건 중 하나만 만족되면 중지되고, 곧바로 '안정 상태 단계'로 전환된다. 첫째, 전송률이 최대 전송률 R_{max}에 도달한다. 둘째, 전송하는 패킷의 일련 번호가 최대 허용 패킷 일련 번호 W_R에 처음으로 도달한다. 셋째, 송신자에게 NACK이 전달된다. 넷째, ACK 타이머가 만료된다.

안정 상태 단계에서는 최근의 두 ACK 윈도우 사이의 평균 열린 윈도우(*average open window*)의 변화 방향에 따라 다른 전송률 조절 정책이 적용된다. 열린 윈도우 크기는 TRAMCC에서 정의하고 있는 것과 동일한 의미로 사용되며, W_R과 송신자가 전송 중인 패킷의 일련 번호와의 차이로 계산된다. i번째 ACK 윈도우의 평균 열린 윈도우 크기 $W_o(i)$는 수식 (11)과 같이 계산된다.

$$W_o(i) = \frac{\sum_{j=(i-1)\times AW+1}^{i\times AW} \{W_R(j) - j\}}{AW},$$

단, i = 0,1,2,...이다. (11)

j는 데이타 패킷의 일련 번호를 가리키며, AW는 ACK 윈도우, $W_R(j)$는 일련 번호 j인 데이타 패킷이 전송되는 순간의 최대 허용 패킷 일련 번호를 의미한다. 이 값에 사용하여, 다음의 세 가지 경우의 전송률 조절 방법 중 하나를 적용한다.

경우 1 (빠른 복구). 열린 윈도우가 지난 ACK 윈도우 동안 한 번이라도 0이 되었을 경우, 수식 (12)와 같이 전송률이 계산된다.

$$R_e(i+1) = (1-\beta)\times R_e(i) + \beta \times R(i)$$

단, $0<\beta<1$이고 i = 0,1,2,...이다. (12)

여기서, $R(i)$는 지난 ACK 윈도우 동안의 실제 전송률을 의미하는데, 열린 윈도우가 0이 되면, 데이타 패킷이 전송되지 못하므로, $R(i)$는 $R_e(i)$보다 작은 값이 된다. ACK 윈도우가 0이 되는 경우는, 네트워크 혼잡으로 인해 데이타 패킷의 손실이 커서 수신자들로부터 ACK이 수신되지 않았거나, 수신자들이 전송한 피드백이 네트워크 상에서 손실되어 송신자에게 전달되지 못한 경우이다. 이 경우는, 앞서 제시한 바와 같이 다대다 세션에서 한 회선에 다수의 데이타 흐름과 제어 흐름의 수가 공존함으로써 발생한다. 이 때, 이전 ACK 윈도우에서 사용한 전송률을 그대로 적용하면 네트워크의 혼잡 상태를 반영할 수 없고, 실제 전송률만을 적용하면 낮은 값에서 머물게 되므로 세션 내 흐름 간 공평한 전송률에 도달하지 못하게 될 수 있으므로, 지난 ACK 윈도우에 대한 전송률 $R_e(i)$과 실제 측정된 전송률 $R(i)$의 EWMA로 새 ACK 윈도우에 적용할 전송률을 결정한다.

경우 2 (전송률 증가). $W_o(i) \geq W_o(i-1)$일 경우. 즉 열린 윈도우가 증가하는 것은 네트워크에서 손실되는 데이타 패킷의 수가 감소하고 있는 것이므로, 네트워크 상태가 현재의 전송률보다 더 높은 전송률을 허용하는 것으로 판단하여, 수식 (13)에 따라 전송률을 증가시킨다.

$R_e(i+1) = R_e(i) + \Delta R'$, 단, i = 0,1,2,...이다. (13)

$R_e(i)$은 i번째 ACK 윈도우에 적용하도록 계산된 전송률을 의미하며, $\Delta R'$은 전송률의 증가치로써 제안된 기법에서는 최대 패킷 크기와 같은 값을 사용한다.

경우 3 (전송률 감소). 만약 평균 열린 윈도우 크기가 감소할 경우, 즉 네트워크에서 손실되는 데이타 패킷의 수가 증가하고 있는 것이므로, 네트워크가 혼잡 상태에 있는 것으로 판단하여, 전송률을 수식 (14)에 따라 감소시킨다.

$R_e(i+1) = R_e(i) - \Delta R'$, 단, i = 0,1,2,...이다. (14)

4. 성능 평가

TMRCC의 세션 내 공평성, 네트워크 상태 변화에 대한 응답성, TCP 친화성 및 세션 규모에 대한 확장성이라는 관점에서 성능을 평가하기 위하여 TMRCC를 네트워크 시뮬레이터인 ns-2[11]를 이용하여 구현하고 시험을 실행하였다. 응답성 및 TCP 친화성, 확장성은 혼잡 제어 기법을 평가하는데 있어서 전형적인 평가 기준이다. '세션 내 공평성'은 다대다 멀티캐스트 혼잡 제어 기법에서 특별히 고려되어야 할 평가 기준으로서, 같은 회선 내에 존재하는 한 세션의 데이타 흐름들이 공평한 대역폭 분할에 해당하는 전송률을 유지하는가를 의미한다.

시뮬레이션에서 사용된 네트워크 토폴로지는 그림 1에 나타나 있다. 병목 회선의 대역폭은 3Mbps로 설정하였으며, 다대다 멀티캐스트 세션의 참여자는 병목 회

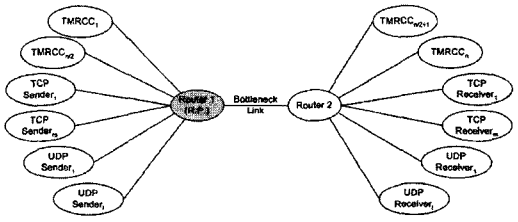


그림 1 시뮬레이션 토폴로지

선을 사이에 두고 양쪽에 대칭적으로 위치하고 있다. 라우터에서의 큐 관리 알고리즘으로 드롭 테일(drop-tail)을 사용하고, PIM-SM[15]과 같은 공유 멀티캐스트 라우팅 프로토콜을 사용한다. 병목 회선의 좌측에 위치한 Router1을 랑데부 포인트(Rendezvous Point, RP)로 설정하였다. 공유 멀티캐스트 라우팅 프로토콜에서는 송신자의 위치와 관계없이 RP가 패킷을 분배하기 때문에, 병목 회선에서 관찰할 때, Router1에서 Router2으로 전달되는 흐름의 수가 그 반대 방향의 흐름의 수보다 많다. TCP 친화성 및 응답성을 관찰하기 위하여, TCP 송/수신자 및, UDP 송/수신자 또한 토폴로지에 포함되어 있으며, 병목 회선의 좌측에 TCP와 UDP 송신자를, 우측에 TCP와 UDP 수신자가 위치한다. TMRCC의 실행 변수로, ACK 윈도우를 8로 설정하고, 최대 최소 혼잡 윈도우 크기를 각각 12와 48로 하였다. 그리고, IF와 DF를 각각 1, 2로 설정하였으며, $\Delta R'$을 1024 바이트로 설정하였다. 세션 내 공평성에 대하여 제안된 기법과 성능을 비교하기 위한 일대다 멀티캐스트 혼잡 제어 기법 TRAMCC가 사용되었으며 TRAMCC를 위한 시뮬레이션 코드는 [8]에 기술된 알고리즘을 기반으로 구현되었다. 시뮬레이션 시나리오와 그 의미를 간략하게 정리하면 다음과 같다.

시뮬레이션 1: 세션 내 공평성. 다대다 멀티캐스트 세션에는 하나 이상의 데이타 흐름과 제어 흐름이 존재하며, 전체 데이타 흐름의 수는 최대로 세션의 참가자의 수만큼 늘어날 수 있다. 따라서, 다대다 신뢰적 멀티캐스트 세션에서는 흐름들 간 대역폭 경쟁이 크게 발생할 수 있으므로, 데이타 흐름들이 공유하는 병목 회선 대역폭의 공평한 분할에 이를 수 있도록지원하는가를 관찰한다.

시뮬레이션 2: 네트워크 트래픽 상태 변화에 대한 응답성. 응답성은 혼잡 제어 기법을 설계하는데 있어 기본적인 고려 사항 중 하나로써, 네트워크 트래픽의 변화에 따라 얼마나 빠르고 적절하게 전송률을 조절하는가를 의미한다. UDP 흐름의 전송률은 시뮬레이션 상에서 쉽게 조절할 수 있으므로, 급변하는 네트워크 트래픽의 원천으로 사용될 수 있다. TMRCC 세션이 시작된 이후에 두 개의 UDP 세션을 각각 200초와 400초에 시작하며, 1024 바이트 크기의 패킷을 초 당 100 개씩 전송한다.

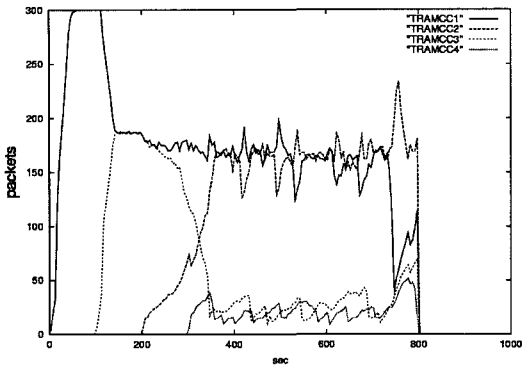
시뮬레이션 3: TCP 친화성. 혼잡 제어에서 TCP 친화성이란, 함께 존재하는 TCP 흐름들이 다른 프로토콜에 의해 제어되는 흐름들에 의해 불공평하게 다루어지지 않는 것을 뜻한다[6]. TMRCC 세션을 시작한 이후에 두 개의 TCP 세션을 각각 200초와 400초에 시작한다. TMRCC 세션이 TCP 세션과 공존할 때 각 세션 내 흐름들의 전송률 변화 추이를 관찰한다.

시뮬레이션 4: 세션 참가자 수에 대한 확장성. 이 실험에서는 세션의 참가자 수가 증가함에 따라 개별 TMRCC 흐름의 전송률의 변화 추이를 관찰한다. 병목 회선의 대역폭은 참가자 당 500 kbps로 보장할 수 있도록 설정하여 참가자의 수가 증가함에 비율적으로 증가하게 한다. 이렇게 함으로써 참가자 수의 증가에 따른 데이타 흐름의 전송률 변화를 쉽게 관찰할 수 있다. 참가자의 수는 8 개부터 32 개까지 두 배씩 증가시키며 송신자의 전송률 변화를 관찰한다. 각각의 시뮬레이션의 결과는 다음 절들에서 보여진다.

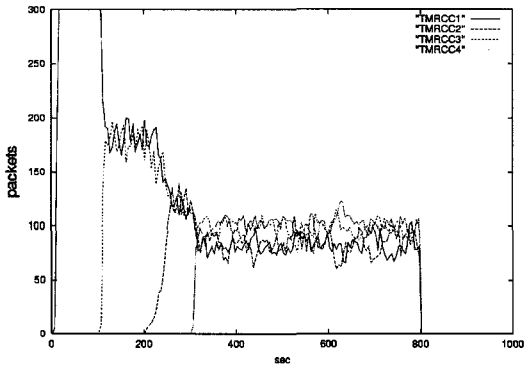
4.1 시뮬레이션 결과 1: 세션 내 공평성

네 개의 참가자가 있는 다대다 신뢰적 멀티캐스트 세션에서 각 참가자의 데이타 전송률을 측정한다. 세션의 두 참가자(TMRCC1, TMRCC2)는 병목 회선의 좌측에 위치하며, 다른 두 참가자(TMRCC3, TMRCC4)는 우측에 위치한다. 모든 참여자는 송신자의 역할을 하며, 0초, 100초, 200초, 300초에 각각 TMRCC1, TMRCC3, TMRCC2, TMRCC4가 데이타 전송을 시작한다. 따라서, 300초부터 병목 회선에 네 개의 데이타 흐름들이 동시에 존재한다. 기존에 제안된 혼잡 제어 기법과의 비교를 위하여 TRAMCC를 똑같은 참가자 조건에서 실행하였다. 다만, TRAMCC가 일대다 세션을 위해 제안된 기법이므로, TRAMCC의 경우에는, 참가자가 동일하고 송신자가 서로 다른 네 개의 일대다 세션이 시간 차를 두고 시작되는 방식으로 세션을 진행하였다.

그림 2(a)는 TRAMCC를 사용하였을 때 각 흐름의 전송률을 보여준다. 그림의 각 선은 각 송신자의 전송률을 표시한다. 그림 2(a)에서 TRAMCC3이 200초에 시작할 때, 혼잡으로 인하여 슬로우 스타트 단계가 빠르게 종료되고, 안정 상태에서는 최근 5초 동안의 평균 전송률을 사용하여 전송률을 변화함으로써 전송률이 매우 느리게 증가한다. TRAMCC2의 전송률이 증가하는 동안, 이미 네트워크에 존재하고 있는, 두 흐름은 새로 시작된 흐름들로 인하여 패킷 손실이 증가한다. 또한 TRAMCC3의 피드백은 다른 두 TRAMCC1과 TRAMCC2와는 달리 혼잡한 회선, 즉 Router1에서 Router2 사이의 회선을 이용하므로, TRAMCC3은 데이타 패킷 손실뿐만 아니라 피드백의 손실까지 겪게 된다. TRAMCC에서는 1초마다 정기적으로 ACK을 명시적으로 요청하므로 발생되는 피드백의 수가 더욱 증가시키고, 더불어 피드백 손실을 빨리 인지하게 되므로 그에 따라 전송률이 감소되어[12], 그림 2(a)에 나타나는 바와 같이 심각한 전송률 감소를 나타내게 된다. 그러나, 앞서 설명한 바와 같이 지난 5초 동안의 평균 전송률을 기반으로 하여 새로이 적용할 전송률을 계산하므로 전



(a) Multiple TRAMCC sessions



(b) A TMRCC session

그림 2 시뮬레이션 결과 1: 세션 내 공평성

송률이 감소된 흐름들은 타 흐름들과 유사한 전송률로 복구되는데 시간이 오래 걸리거나, 그림 2(a)에서 보는 바와 같이, 실패하게 된다.

그림 2(b)에 나타난 바와 같이, 제안하는 기법은 여러 흐름들 간의 공평한 전송률을 보여준다. 이는, TMRCC에서는 명시적으로 ACK을 요청하되 네트워크 상태를 반영하여 동적으로 ACK 타이머와 NACK 타이머 값이 변경되므로 피드백 손실이 발생한다고 해도 추가적으로 발생하거나 요구되는 피드백의 증가량이 TRAMCC보다 크게 작다. 그리고, 전송률 조절에 있어서, 빠른 복구 기법을 사용하므로, TRAMCC에서는 전송률이 급격히 감소할 수 있는 상황에서도, 제안하는 기법을 사용하면 보다 빨리 타 흐름들과 유사한 전송률로 복구가 가능하다.

4.2 시뮬레이션 결과 2: 네트워크 트래픽 상태 변화에 대한 응답성

그림 3에서 볼 수 있듯이, TMRCC 흐름은 네트워크 트래픽 상태의 변화에 빠르게 적응한다. 200초와 400초 사이의 UDP 세션이 존재할 때, 각 TMRCC 흐름은 병목 회선의 사용 가능한 대역폭의 공평한 양인 평균

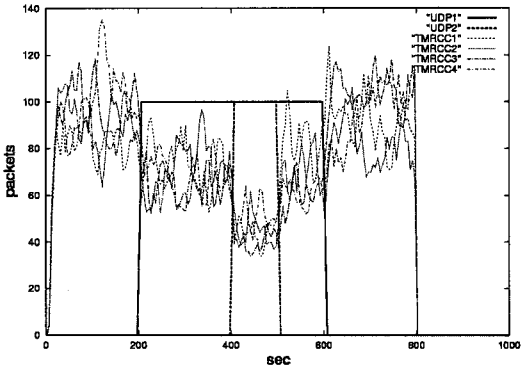


그림 3 시뮬레이션 결과 2: 네트워크 트래픽 상태 변화에 대한 응답성

초당 약 65 개의 패킷을 전송한다. UDP 세션이 시작되었을 때, 모든 TMRCC에서 즉시 패킷 손실이 발생한다. 패킷 손실은 각 수신자의 혼잡 윈도우 크기의 감소를 가져오며, ACK의 전달 지연 시간을 증가시킨다. 이로 인하여, 각 송신자는 전송률을 적당한 수준으로 감소시키게 된다. 400초와 500초 사이에 두 번째 UDP 세션이 시작될 때, 각 TMRCC 흐름은 전송률을 초당 약 40 패킷으로 감소시킨다. UDP 세션들이 차례로 마칠 때마다, TMRCC 세션 내 흐름들이 UDP 세션이 시작되기 이전의 전송률로 복구되는 것을 볼 수 있다.

4.3 시뮬레이션 결과 3: TCP 친화성

본 시뮬레이션에서는 두 개의 TCP 세션이 각각 200초와 400초에 시작되어 600초와 500초에 종료된다. 그림 1에서 보이는 바와 같이 'TCP sender'라고 표시된 노드에 TCP 세션의 송신자가 위치하고 'TCP receiver'라고 표시된 노드에 TCP 세션의 수신자가 위치하여 병목 회선을 TMRCC 세션 내 흐름들과 공유하게 된다.

TCP 친화성은 ACK 윈도우 크기, 혼잡 윈도우의 크기 등에 따라서 차이를 보인다. TCP보다 적극적으로 혼잡에 대응하게 되면 TCP 세션보다 전송률 변화의 폭이 커지게 되고, TCP보다 미진하게 혼잡에 대응하게 되면 TCP 세션보다 전송률 변화의 폭이 작아지게 되어 TCP 세션과의 공평한 전송률을 보이기가 어렵다. 본 실험에서 사용한 변수들의 값을 다음과 같이 설정하였다. ACK 윈도우의 크기를 8(packet)으로 하고, 혼잡 윈도우의 최대 크기를 12(packet), 최대 크기를 48(packet)으로 설정하고, IF를 1(packet), DF는 2(packet), $\Delta R'$을 1024(bytes)로 설정하였다. 동일한 조건에서 실험을 10회 반복적으로 수행하여 관찰된 평균 전송률을 그림 4에서 보이고 있다.

그림 4에서 보이는 바와 같이, TCP 세션이 시작되는 200초 이후부터 새로운 데이타 흐름으로 인해 혼잡이

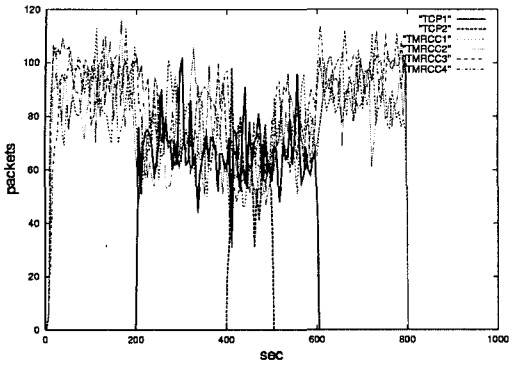


그림 4 시뮬레이션 결과 3: TCP 친화성

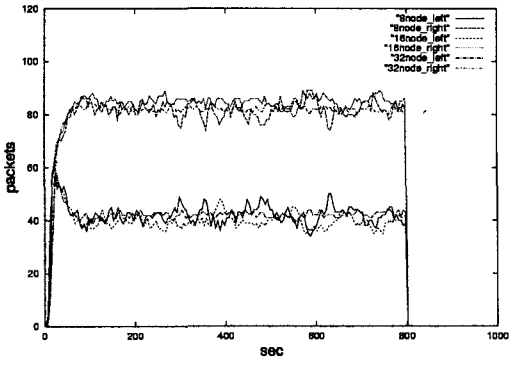


그림 5 시뮬레이션 결과 4: 세션의 크기에 따른 확장성

발생하게 되고 이에 대응하여 TMRCC 세션 내 흐름들이 전송률을 낮추게 된다. 결과적으로 TCP 세션들과 유사하게 대응하여 전송률을 조정함으로써 TMRCC 세션 내 흐름들의 평균 전송률이 TCP 세션의 전송률과 유사한 값을 보이는 것을 확인할 수 있다. 그리고, 새로운 TCP 세션이 추가로 시작되는 400초 이후에 전송률이 더욱 감소하는 것을 관찰할 수가 있다. TCP 세션이 종료되는 600초 이후에는, 병목 회선 내의 흐름 수가 감소하게 되므로 혼잡으로 인한 패킷 손실이 감소하여 TMRCC 흐름들이 전송률을 증가시키게 되고 TCP 세션이 시작되기 이전의 전송률을 회복하는 것을 관찰할 수 있다.

4.4 시뮬레이션 결과 4: 세션의 크기에 따른 확장성

그림 5에서 세션 크기에 대한 확장성을 평가한 시뮬레이션 결과를 확인할 수 있다. 그런데, 공유 멀티캐스트 라우팅 프로토콜이 병목 회선의 비대칭적인 사용 형태를 가져옴으로 인하여, 병목 회선의 좌측에 위치한 참가자들과 우측에 위치한 참가자들의 전송률에 차이가 나타난다. 이러한 차이는 멀티캐스트 라우팅 프로토콜과 함께 구성된 복구 트리의 형태와도 관련이 있다. 이 실험에서는 각 노드는 최대 네 개까지의 자식 노드를 가질 수 있게 하였으며, 각 송신자는 같은 쪽에 세 개의 자식 노드와 반대 방향에 한 개의 자식 노드를 갖도록 설정한 후, 다른 노드들은 같은 방향에 있는 노드를 부모 노드로 가지도록 하였다. 따라서, 송신자와 송신자의 한 개의 자식 노드만이 복구 트리 상에서 병목 회선을 공유하게 되므로 이 실험에서는 피드백 손실이 심각하게 나타나지 않는다. 하지만, 병목 회선의 좌측에 위치한 송신자들은 손실된 패킷을 재전송하는데 있어서, 우측에 위치한 송신자들과 달리, 병목 회선을 사용하게 되므로, 재전송 패킷의 손실이 그만큼 빈번하게 나타나게 된다. 이러한 재전송 패킷의 손실은 결국 그림 5에서 볼 수 있듯이 병목 회선의 좌측에 위치한 송신자와 우측에 위치한 송신자 사이의 전송률의 비대칭성을 야기하게 된다. 이러한 비대칭성과는 별개로, 그림 5는, 병목 회선의 대역폭이 참가자 수 증가에 따라 비례적으로 증가할 때, 참가자 수의 증가와는 관계없이 각 흐름들이 전체 대역폭의 동일한 비율을 차지한다는 것을 보여주고 있으며, 이는 TMRCC의 확장성을 입증하는 것이다.

5. 구현 및 시험 네트워크 환경에서의 실험 결과

본 장에서는, 트리 기반의 다대다 신뢰적 멀티캐스트 프로토콜 GAM [3]에 TMRCC를 통합하여 Linux 상에서 C++ 언어로 구현한 결과를 기술한다.

GAM은 *Hu* 기법을 기반으로 한 신뢰적 멀티캐스트 프로토콜로서, 다대다 멀티캐스트 세션을 위한 효율적인 복구 트리를 구성하기 위한 기법을 제안한다. 참가자를 여러 개의 작은 '지역 그룹'으로 나누고 개별 그룹마다 에러 복구를 관장하는 역할을 수행하는 '코어(core)'를 중심으로 공유 트리 형태의 복구 트리를 구성한다. 또한, 이와 별도로 '코어'들 간의 복구 트리를 구성하여 개별 그룹 내에서 변경되는 멤버쉽 정보를 지역 그룹 내로 제한함으로써 복구 트리를 관리하는 부담 및 에러 복구에 드는 처리 부담을 크게 줄였다.

5.1 GAMSocket 클래스

GAMSocket 클래스는 GAM 구현의 핵심 클래스로써 GAMSocket 클래스는 유니캐스트채널에 해당하는 GAMUnicastChannel 쓰레드와 멀티캐스트채널에 해당하는 GAMMulticastChannel 쓰레드, 패킷 버퍼 윈도우에 해당하는 GAMPacketWindow 쓰레드, 그리고 타이머에 해당하는 GAMTimer 쓰레드를 가진다. 그림 6은 GAMSocket 클래스의 구조를 나타낸다. GAMSocket 클래스가 초기화될 때, GAMUnicastChannel, GAMMulticastChannel, GAMTimer가 시작된다. GAMUnicastChannel, GAMMulticastChannel은 수신되는 패킷을 기다리며, GAMTimer는 ACK과 retrans-

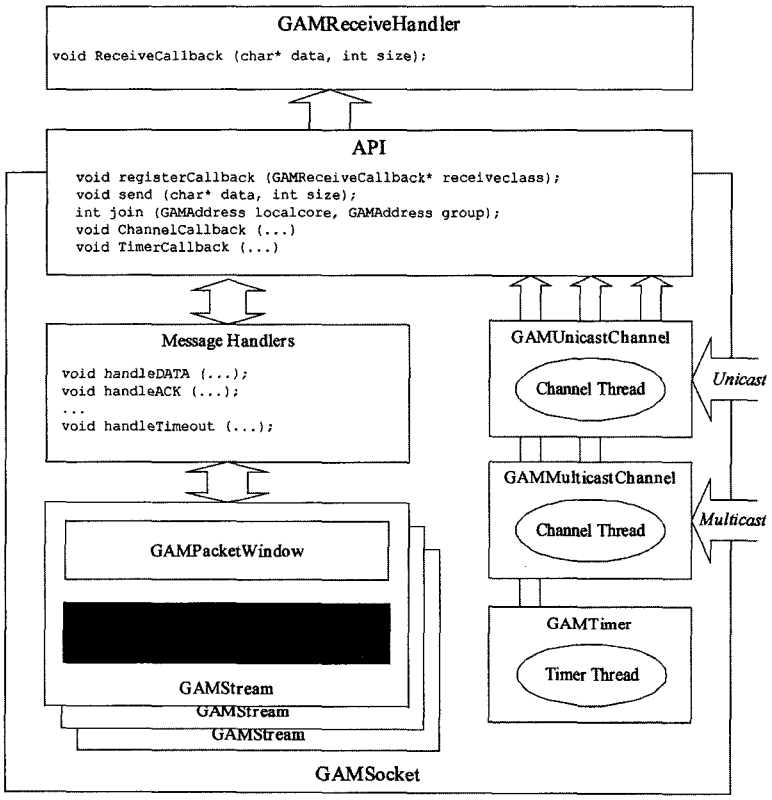


그림 6 GAMSocket 클래스의 구조

mission 등을 위해 설정된 타이머가 만료되었는지를 검사한다. GAMMulticastChannel로 데이타 패킷을 수신하거나 GAMTimer에서 타임아웃 등의 이벤트가 발생하는 경우, 각 쓰레드는 설정된 콜백 함수를 실행하여 해당되는 이벤트를 처리한다. 예를 들어, 데이타 패킷을 수신한 경우 *handleData()*가 실행되며, ACK 패킷을 수신하면 *handleACK()*이, 타임아웃 이벤트가 발생하며 *handleTimeout()*이 각각 실행된다.

이렇게 구성된 GAMSocket 클래스에 TMRCC를 반영하기 위해 다음과 같은 기능이 반영된다. 첫째, 주기적으로 전송률을 재조정하는 GAMRateCtlr 쓰레드가 포함된다. 둘째, GAMPacketWindow 쓰레드에 혼잡 윈도우에 해당하는 변수와 혼잡을 판단하기 위한 변수가 포함된다. 셋째, GAMTimer 쓰레드에 실제 ACK 응답 시간, NACK 응답 시간 측정을 위한 변수가 포함되고, *handleACK()*에서 GAMTimer에게 ACK 수신 및 재전송 패킷 수신을 통지할 수 있도록 한다. 넷째, 송신 함수인 *send()*를 혼잡 윈도우와 전송률 제어기에 의해 조절된 전송률에 따라 패킷을 전송하도록 한다. 다섯째, ACK과 NACK 패킷에 혼잡 윈도우 정보(W_L과 W_R)를 패킷 헤더에 추가한다.

5.2 시험 네트워크 환경에서의 실험 결과

그림 7은 실험실 내에 구축된 시험 네트워크 환경의 토폴로지를 보이고 있다. 실제 인터넷 환경을 모방하기 위하여, Router1과 Router2, 그리고 Router1과 Router3 사이의 회선 대역폭을 T1 회선에 해당하는 1.544 Mbps로 설정하였다. 시험 네트워크에는 두 개의 GAM 코어와 네 개의 세션 참가자가 존재하며, 그림에서 세션 참가자는 GAM으로 표시되었다. GAM 코어는 참가자들의 에러 복구를 전담하는 역할을 수행하며, 각각 Router1의 지역 네트워크와 Router2와 Router3의 지역 네트워크 내 참가자들을 서비스한다. IP 멀티캐스트 라우팅을 위하여 PIM-SM이 사용되었고 Router1을 RP로 설정하였다.

실용성의 검증을 위하여 세션 내 공평성을 측정하는 실험과 TCP 친화성을 측정하는 실험이 수행되었다. 세

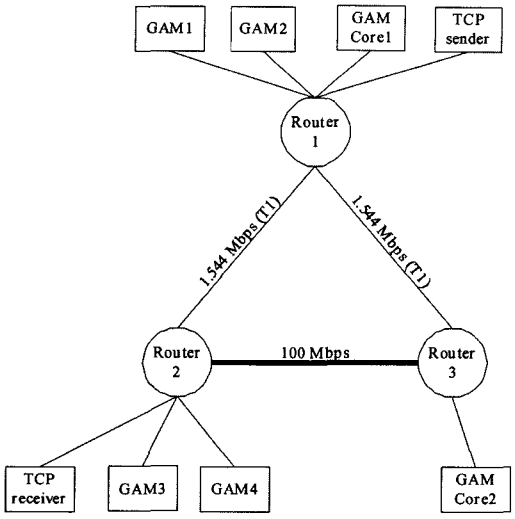


그림 7 실험을 위한 시험 네트워크 토폴로지

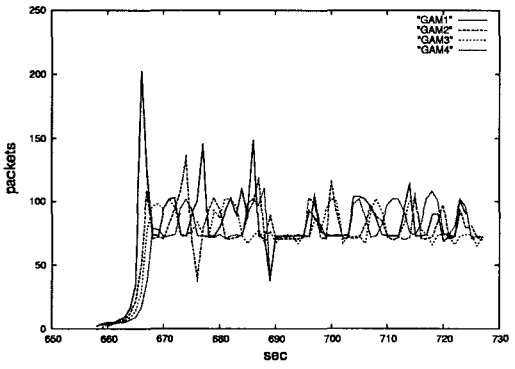


그림 8 세션 내 공평성 실험 결과

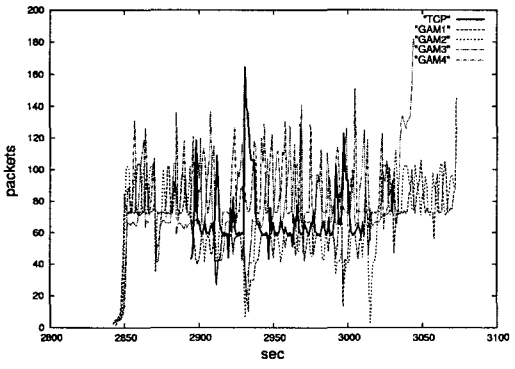


그림 9 TCP 친화성 실험 결과

션 내 공평성을 측정하는 실험에서, GAM/TMRCC 세션의 각 참가자들은 5M 바이트의 데이타를 전송하였다. 그림 8에서 실험 결과를 보여주고 있는데, 각 참가자들이 유사한 전송률을 유지하면서 데이타를 전송한다.

TCP 친화성을 측정하기 위한 실험에서, GAM/TMRCC 세션의 각 참가자들은 15M 바이트의 데이타를 전송하며, GAM/TMRCC 세션의 중간에 8M 바이트의 데이타를 전송하는 TCP 세션이 시작한다. TCP 세션의 송신자와 수신자의 위치는 그림 7에서 보는 바와 같고, GAM1, GAM2와 동일한 방향으로 데이타가 전달된다. TCP 친화성 실험 결과는 그림 9에 나타나 있으며, GAM1과 GAM2가 TCP 흐름의 전송률과 유사하게 전송률이 조절되는 것을 관찰할 수 있다.

6. 결론 및 향후 연구

본 논문에서는 트리 기반의 다대다 신뢰적 멀티캐스트 프로토콜을 위한 효율적인 혼잡 제어 기법으로 TMRCC를 제안하였다. 제안하는 기법에서는 혼잡 윈도우 기반의 기법으로 추가적으로 전송률 제어기를 사용한다. 혼잡 제어를 위해 추가되는 수신자의 처리 부담을 최소화하기 위하여 에러 복구를 위한 피드백을 혼잡 제어에 함께 사용하며, 네트워크 상태의 변화를 반영하여 동적으로 변화하는 ACK 타이머 및 NACK 타이머와, 피드백 손실 시 전송률 조절에서 네트워크 상태를 반영하며 동시에 급격한 전송률 변화를 방지하도록 함으로써 피드백 및 재전송 패킷의 손실에 효과적으로 대처한다. 제안하는 혼잡 제어 기법의 성능을 ns-2를 이용하여 세션 내 공평성, 네트워크 트래픽 변화에 대한 응답성, TCP 흐름에 대한 친화성, 세션 참가자 수에 대한 확장성이라는 관점에서 평가하였다. 평가 결과는 제안된 기법이 세션 내 공평성에서 기존에 일대다 세션을 위한 혼잡 제어 기법인 TRAMCC보다 나은 성능을 보여주고 있으며, 응답성 및 TCP 친화성, 확장성에서도 만족할 만한 성능을 보여주고 있다. 또한, TMRCC를 선행 연구 결과인 신뢰적 멀티캐스트 프로토콜 GAM과 통합하여 구현하였으며, 시험 네트워크 상에서의 실험을 통하여 시뮬레이션에서의 결과와 유사한 결과를 확인할 수 있었다.

향후 연구로는, 실제 인터넷 상에서의 적용을 고려하여, 응용 분야의 요구에 따라 제안된 기법의 파라미터들을 자동으로 적응시키는 방법을 강구해야 할 것이며, 보다 큰 규모의 세션에서의 TMRCC의 확장성에 대한 검증도 함께 이루어져야 할 것이다. 그리고, 연구실 규모의 시험 네트워크 뿐만 아니라 광역 네트워크 환경에서의 적용 및 실험을 추진할 것이다.

참 고 문 헌

[1] S. Floyd, V. Jacobson, C.-G. Liu, S. McCanne, and L. Zhang, "A reliable multicast framework for

light-weight sessions and application level framing," IEEE/ACM Transactions on Networking, Vol.6, No.5, pp.784-803, 1997.
[2] B. N. Levine, D. B. Lavo, and J. J. Garcia-Luna-Aceves, "The case for reliable concurrent multicasting using shared ack trees," Proc. of ACM Multimedia, pp.365-376, ACM Press, New York, 1997.
[3] W. Yoon, D. Lee, and S. Lee, "A combined group/tree approach for many-to-many reliable multicast," Proc. of IEEE INFOCOM '02, pp.1336-1345, New York, 2002.
[4] W. Yoon, D. Lee, and H.Y. Youn, "On the Scalability of Many-to-many Reliable Multicast Sessions," Proc. of IEEE International Parallel and Distributed Processing Symposium, pp.255-262, 2002.
[5] B. Whetten, L. Vicisano, R. Kermode, M. Handley, S. Floyd, M. Luby, Reliable Multicast Transport Building Blocks for One-to-Many Bulk-Data Transfer, Internet RFC 3048, 2001.
[6] S. Floyd and K. Fall, "Promoting the use of end-to-end congestion control in the internet," IEEE/ACM Transaction on Networking, Vol.7, No.4, pp.458-472, 1999.
[7] I. Rhee, N. Balaguru, and G. N. Rouskas, "MTCP: Scalable TCP-like congestion control for reliable multicast," Proc. of IEEE INFOCOM'99, Vol.3, pp.1265-1273, New York, 1999.
[8] D. M. Chiu, M. Kadansky, J. Provino, J. Wesley, H.-P. Bischof, and H. Zhu, "A congestion control algorithm for tree-based reliable multicast protocols," Proc. of INFOCOM '02, pp.1209-1217, 2002.
[9] V. Jacobson, "Congestion avoidance and control," Proc. of ACM SIGCOMM'88, pp.314-329, ACM Press, Stanford, 1988.
[10] W. Yoon, D. Lee, H. Y. Youn, and S. J. Koh, "Throughput analysis of tree-based protocols for many-to-many reliable multicast," Proc. of ICC 2002, Vol.4, pp.2523-2527, 2002.
[11] S. Bajaj, L. Breslau, D. Estrin, K. Fall, S. Floyd, P. Haldar, M. Handley, A. Helmy, J. Heidemann, P. Huang, S. Kumar, S. McCanne, R. Rejaie, P. Sharma, K. Varadhan, Y. Xu, H. Yu, and D. Zappala, "Improving simulation for network research," Technical Report 99-02b, USC, 1999.
[12] D.-M. Chiu, S. Hurst, M. Kadansky, and J. Wesley, "TRAM : A tree-based reliable multicast protocol," Technical Report TR-98-66, Sun Microsystems, 1998.
[13] M. Allman, "A web server's view of the transport layer," ACM Computer Communication Review, Vol. 30, No.5, pp.10-20, 2000.
[14] M. Luby, V. K. Goyal, S. Skaria, G. B. Horn, "Wave and Equation Based Rate Control using Multicast Round Trip Time," Proc. of ACM SIGCOMM'02, pp.191-204, ACM Press, New York, 2002.
[15] D. Estrin, D. Farinacci, A. Helmy, D. Thaler, S. Deering, M. Handley, V. Jacobson, C. Liu, P. Sharma, L. Wei, Protocol Independent Multicast-Sparse Mode (PIM-SM): Protocol Specification, Internet RFC 2362, 1998.

유 제 영

1998년 2월 동국대학교 컴퓨터공학과 학사. 2003년 2월 한국정보통신대학원대학교 공학 석사. 2003년 2월~9월 (주)코어세스 홈네트워크 개발실 주임연구원 2003년 9월~현재 (주)삼성탈레스 주임연구원. 관심분야는 멀티캐스트, 홈네트워크, 유비큐토스컴퓨팅

강 경 란

1992년 2월 서울대학교 계산통계학과 학사. 1994년 2월 한국과학기술원 석사 1999년 2월 한국과학기술원 박사. 1999년~2000년 한국전자통신연구원 선임연구원. 2000년~2002년 (주)디지탈웨이브 책임연구원. 2002년~현재 한국정보통신대학원대학교 연구조교수. 관심분야는 multicast, real-time collaboration, mobility

이 동 만

1982년 서울대학교 컴퓨터공학 학사 1984년 한국과학기술원 전산학 석사 1987년 한국과학기술원 전산학 박사 1987년~1988년 한국과학기술원 박사후과정. 1988년~1997년 Hewlett-Packard 책임연구원. 1997년~현재 한국정보통신대학원대학교 부교수. 관심분야는 multicast protocol, group communication, distributed virtual environment, pervasive computing