무선 이동 환경을 위한 개선된 TFRC 혼잡제어 메커니즘

(An Enhanced TFRC Congestion Control Mechanism for Mobile Environments)

최 미 라 † 이 미 정 ††
(Mira Choe) (Meejeong Lee)

요 약 멀티미디어 응용프로그램의 요구 조건을 만족시키면서 TCP 응용프로그램과 공존할 수 있는 혼잡제어 메커니즘으로 TFRC(TCP-Friendly Rate Control)가 제안되었는데, TFRC는 TCP 트래픽과 전송 대역폭을 공정하게 공유한다는 기본 특성을 지닌다[1-3]. 그런데, 실험을 통해 살펴본 결과 TFRC 혼잡제어 메커니즘이 무선 이동 환경에 적용될 경우에는 이동호스트가 핸드오프를 거듭할수록 처리율 및 공정성이 저하됨을 알 수 있었다. 이에 본 논문에서는 무선 이동 환경에서의 TFRC 성능 향상을 위해 TFRC가 핸드오프 시 발생한 패킷 손실을 처리하는 방안과 핸드오프 직후 구동하는 혼잡제어 방식을 제안하였다. 시뮬레이션에 의하여 연속적인 핸드오프가 발생하는 무선 이동 환경에서 제안하는 방안이 기존 TFRC에 비해 더 높은 처리율을 유지할 수 있고 공정성도 향상시킴을 볼 수 있었다.

키워드 : TFRC, 핸드오프, 공정성, 혼잡제어

Abstract TFRC(TCP-Friendly Rate Control) is proposed to satisfy the demands of multimedia applications while being reasonably fair when competing for bandwidth with TCP flows[1-3]. However, TFRC has a shortcoming that the fairness and throughput are degraded when the mobile host using TFRC experiences handoffs. This paper proposes a new control mechanism based on TFRC, which deals with the congestion caused by handoffs as well as the losses caused during the handoffs. The simulation results show that our mechanism achieves better throughput and fairness compared to TFRC for repeated handoffs.

Key words : TFRC, handoff, fairness, congestion control

1. 서 론

인터넷은 최선형 서비스(best-effort service)를 제공한다. 최선형 서비스에서 가장 중요하게 고려되어야 할 사항 중 하나가 혼잡제어이다. 혼잡제어란 네트워크에 혼잡이 발생하였을 때 전송률을 감소시켜 혼잡 붕괴 상황을 막는 것으로 현재 대부분의 인터넷 트래픽은 TCP 혼잡제어 메커니즘을 사용하고 있다. 그런데 TCP는 기존의 신뢰성 있는 전송을 요구하는 데이타 전송에는 적합하지만 스무드한 전송률 유지가 중요하고 전송 지연 제약이 엄격한 멀티미디어 응용의 경우에는 적합하지 않다. 데이타 응용은 전송 에러를 회복하는 메커니즘이 필수적이지만 멀티미디어 응용의 경우에는 재 전송된 데이타가 받아야 할 시간보다 늦어지면 소용없는 경우도 있다. 또한 TCP의 경우 하나의 패킷 손실에 대해 전송률을 반으로 줄여 혼잡상황에 잘 대처하지만 멀티미디어 응용의 경우 이와 같은 TCP의 급격한 전송률 변화는 사용자가 느끼는 오디오나 비디오의 품질을 떨어뜨리는 문제점을 갖고 있다. UDP를 사용하면 이와 같은 문제를 피할 수 있지만 이 프로토콜은 TCP와 같은 혼잡제어를 하지 않기 때문에 혼잡 상태를 가중시켜 대역폭을 낭비하게 만들뿐만 아니라 혼잡제어를 하는 프로토콜과의 대역폭 할당에서의 공정성에 문제점을 가지고 있다.

이에, 멀티미디어 트래픽의 혼잡제어를 위해 TCP-friendly 혼잡제어[1-6]의 일종인 TFRC(TCP-Friendly Rate Control)[1-3]가 제안되었다. TFRC는 TCP의 윈도우 기반 전송 알고리즘과는 대조적으로 율제어 등식

† 비 회 원 : 이화여자대학교 컴퓨터학과
mira.choe@samsung.com
†† 정 회 원 : 이화여자대학교 컴퓨터학과 교수
lmj@mm.ewha.ac.kr
논문접수 : 2003년 2월 13일
심사완료 : 2003년 7월 28일

기반 전송 알고리즘을 사용한다. TFRC는 TCP의 전송률을 모델링한 방정식을 율제어 등식으로 사용함으로써 TCP와 공정한 대역폭 분배가 가능하도록 한다.

그러나 시뮬레이션을 통해 살펴본 결과 TFRC 혼잡제어 메커니즘이 무선 환경에 적용될 경우에는 이동호스트가 핸드오프를 거듭할수록 처리율 및 공정성이 상당히 저하됨을 알 수 있었다. 이러한 문제는 핸드오프로 인해 발생한 패킷 손실 기록이 송신단의 전송률 도출 공식에 지속적으로 영향을 주는데서 기인한다. 또한 핸드오프로 인해 이동해간 셀에서 혼잡이 발생하는 경우 송신단 전송률이 크게 저하되는데, TFRC는 세션 시작 시에만 Slow Start로 전송률을 증가시키고 세션 진행 중 발생하는 혼잡에 대해서는 Congestion Avoidance에 의해 전송률을 증가시키기 때문에 결국 핸드오프 후에도 송신단에서 전송률을 빠르게 회복하지 못하게 되어 처리율과 공정성이 저하하게 된다.

TCP에 대해서는 무선 이동 환경에서의 성능 저하 문제를 해결하기 위한 방안들[7-10]이 다양하게 제안되었다. 예를 들면 I-TCP[7], Snoop TCP[8], Fast Retransmit[9], M-TCP[10] 등이 그것인데, 특히 Fast Retransmit와 M-TCP는 이동호스트가 핸드오프를 일으킬 때 나타나는 성능 저하 문제를 개선하기 위한 방안을 제시하고 있다. Fast Retransmit는 핸드오프로 인해 손실된 패킷들을 핸드오프 완료 시 빠르게 재전송하기 위한 방안이다. Fast Retransmit에서는 Mobile IP와 TCP의 연동을 통해 핸드오프가 완료되면 재전송 타임아웃이 발생하는 것을 기다리지 않고 바로 전송을 재개할 수 있도록 하며 통신 재개 시에는 윈도우 사이즈를 절반으로 떨어뜨리고 Slow Start 알고리즘을 사용한다. 이렇게 함으로써, Fast Retransmit는 핸드오프 완료 후에도 재전송 타임아웃이 발생할 때까지 통신을 재개하지 못해 처리율이 극도로 낮아지는 문제를 해결하였다. 이 방안은 재전송을 통한 신뢰성 있는 데이타 전송과 빠른 통신재개로 처리율을 향상시키는데 주안을 두고 있다. 한편, M-TCP는 이동호스트의 핸드오프가 수행되는 동안에 송신단이 데이타를 보내는 것을 막음으로써 핸드오프 시의 데이타 손실을 막고, 핸드오프 완료 후에는 핸드오프 이전 윈도우 사이즈로 데이타를 전송하도록 하여 불필요한 처리율 저하를 막는다.

그러나 TCP에 대한 이러한 연구들은 신뢰성 있는 데이타 전송 서비스를 위한 효율성에 초점을 맞추고 있어, 이들 연구를 TFRC에 그대로 적용시키는 것은 적합하지 않다. 이들 연구에서 제안하는 방안은 패킷 손실을 막기 위한 의도적인 통신 두절이나 손실된 패킷을 재전송 받기 위한 지연, 급격한 전송률의 변화 등을 발생시키기 때문에 멀티미디어 응용에 대한 서비스 품질 저하가 발생할 수 있기 때문이다. 따라서 본 논문에서는 TCP의 핸드오프 시 발생하는 문제점에 대한 해결책들과는 별개로 TFRC 혼잡제어 메커니즘에서 주요하게 다루고자 하는 성능 요구사항을 만족시킬 수 있는 해결방안을 찾아보고자 한다.

본 논문에서는 TFRC 혼잡제어 메커니즘을 사용하는 이동호스트가 무선 환경에서 핸드오프를 거듭할 때 야기되는 처리율과 공정성 저하 문제를 개선하기 위해 핸드오프 시 발생한 패킷 손실에 대한 처리 방안과 핸드오프 직후 구동되는 혼잡제어 방안을 제안한다. 먼저 수신단 TFRC는 핸드오프 시 발생한 패킷 손실에 대한 기록을 삭제하고 송신단에게 피드백하지 않음으로써 손실 이전 전송률로 데이타가 전송되도록 한다. 또한, 이동호스트의 핸드오프로 인해 혼잡이 발생한 경우에는 송신단 TFRC가 기존의 전송률 증가 패턴보다 좀 더 빠르게 전송률을 증가시키도록 하였다. 핸드오프가 발생한 경우는 세션 시작 시와 마찬가지로 새로운 환경에서 플로우가 전송되기 시작하는 것이므로 좀 더 적극적으로 전송률을 적응시키는 것이 적합하다고 보았기 때문이다.

본 논문의 구성은 다음과 같다. 1장 서론에 이어 2장에서는 무선 이동 환경에서의 TFRC의 문제점을 살펴본다. 3장에서는 무선 이동 환경에서의 효율적인 TFRC 동작을 위해 본 논문에서 제안하는 개선 방안을 자세히 설명한다. 4장에서는 제안하는 방안의 성능 평가를 위한 시뮬레이션 모델을 설명하고 시뮬레이션 결과를 분석한다. 마지막으로 5장에서는 본 논문의 결론을 기술한다.

2. 무선 이동 환경에서의 TFRC의 문제점

[1]에서 지적한 바와 같이 TFRC는 일시적인 대역폭 변화에 대해 전송률 변화를 절제하고 갑작스런 가용 대역폭 변화에는 느리게 반응한다는 문제점을 갖고 있다. 시뮬레이션을 통해 살펴본 결과 TFRC 혼잡제어 메커니즘을 사용하는 이동호스트가 무선 환경에서 다양한 가용대역폭의 셀 사이를 연속적으로 핸드오프할 때 처리율과 공정성이 상당히 저하됨을 알 수 있었다. 이와 같은 문제점을 살펴보기 위해 반지름이 200m이고 오버랩 구간이 50m인 두 개의 셀을 두고 한쪽 셀에만 TCP를 사용하는 이동호스트를 두 개 상주시켜 두 셀에서 가용한 대역폭에 차이가 생기도록 한 후, TFRC 이동호스트가 20m/s의 속도로 이동하여 20초 간격으로 두 셀을 번갈아 가며 방문하도록 하는 실험을 수행하였다. 이 실험의 설정은 인위적이지만 실제로 가용대역폭이 다양한 셀 사이를 연속적으로 핸드오프하는 상황을 단순화한 시뮬레이션 모델이다.

그림 1과 2는 이 실험의 결과로서, 각각 시간이 흐름

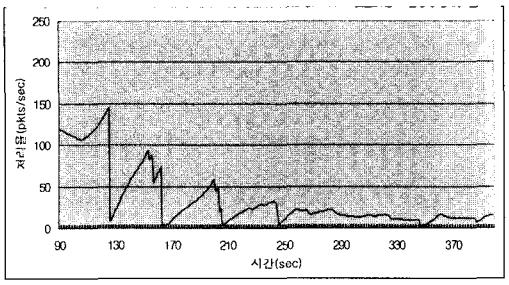


그림 1 연속적인 핸드오프와 송신측 전송률 변화

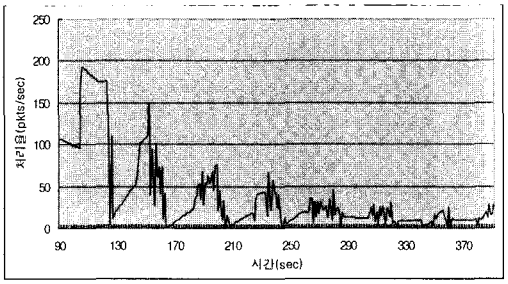


그림 2 연속적인 핸드오프와 수신측 처리율 변화

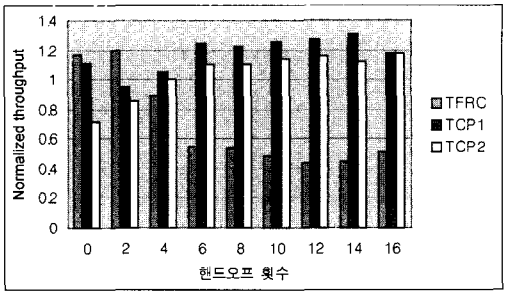


그림 3 각 플로우별 대역폭 할당 정도

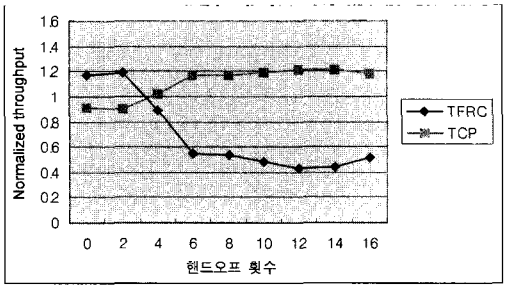


그림 4 TCP와 TFRC의 대역폭 할당 정도

에 따른 송신측 패킷 전송률 변화와 수신측 처리율 변화를 보여준 그래프이다. TFRC 혼잡제어 메커니즘을 사용하는 이동호스트가 핸드오프를 거듭할수록 점차 수신측 처리율이 저하하여 어느 시점부터는 초당 10개 내외의 패킷 수신율을 보이는 상태를 유지하게 된다. 이러한 저 처리율은 사용자가 느끼는 서비스 품질의 저하를 의미한다.

TCP와 공정하게 대역폭을 공유하는지 알아보기 위해 위의 실험에서와 동일하게 시뮬레이션 환경을 설정하고 TFRC 이동호스트가 연속적으로 몇 번의 핸드오프를 수행한 후 2개의 TCP 이동호스트가 상주하는 셀로 진입하여 300초간 머무르도록 하면서 각 플로우에 대해 처리율을 측정하고 이를 공정한 대역폭 몫을 기준으로 정규화한 값을 구하였다. 정규화된 처리율은 각 플로우의 대역폭 차지의 공정성 정도를 나타내는 값이다. 이 값이 1이면 해당 플로우가 공정한 몫을 차지한 것이고 이 값이 1보다 적은 것은 공정한 몫을 차지하지 못한 것이며, 1보다 큰 것은 공정한 몫 이상을 차지한 것을 의미한다. 그림 3과 4는 이 실험의 결과로서, TFRC 이동호스트가 2개의 TCP 이동호스트가 상주하는 셀로 진입하기 전 경험한 핸드오프 횟수에 따라 플로우별 정규화된 처리율이 어떻게 변화하는지를 보여준 것이다. 그림 3은 플로우별 대역폭 할당 정도를 각각 보인 것이고, 그림 4는 두 개의 TCP 플로우가 사용하는 대역폭의 평균 값과 TFRC 플로우의 대역폭 할당 정도를 비교한 것이다.

연속적인 핸드오프 횟수가 0에서 6까지 증가함에 따라 셀 내 존재하는 TCP 플로우들과의 공정성이 급격히 저하됨은 알 수 있다. 6회 이상부터는 연속된 핸드오프 횟수의 증가로 인해 공정성이 더 감소하지는 않았다. 이 결과를 통해, TFRC 혼잡제어를 사용하는 이동호스트가 핸드오프를 거듭하면 공정성이 저하됨을 알 수 있는데, 이것은 TCP-friendly 혼잡제어의 근본 의의에 위배된다.

그림 5는 그림 1, 2의 결과를 측정하기 위해 수행한 실험과 동일한 실험을 수행할 때 수신단에서 계산하는 손실 이벤트 속도 p가 어떻게 변화하는지를 보인 그래프이다. TFRC에서 손실 이벤트는 다음과 같이 정의된다[3]. 손실된 패킷 S_{old}가 어떤 손실 이벤트의 시작이고 S_{new}가 이후에 손실된 패킷이며 S_{old}와 S_{new}의 도착시각이 각각 T_{old}, T_{new}인 경우, (T_{old} + RTT) ≥ T_{new}이면 S_{new}는 S_{old}와 동일한 손실 이벤트에 속하게 된다. 그렇지 않으면 S_{new}는 새로운 손실 이벤트의 첫 번째 패킷이 되며 ($T_{new} - T_{old}$)가 이전 손실 이벤트의 시간 간격이 된다. 손실 이벤트 속도 p는 최근 n개(보통 8개)의 손실 이벤트에 대해 각 이벤트 시간 간격의 가중치 합을 구하고 그 평균의 역을 취한 것이다. 손실 이벤트 속도 p는 수신단 처리율 X_{recv}와 함께 송신단에게 피드

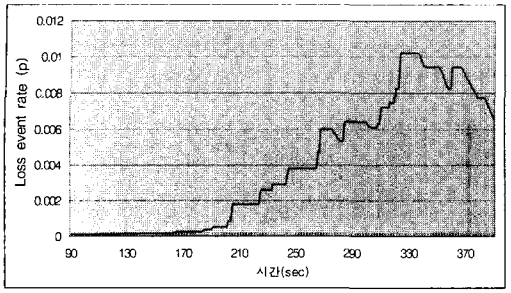


그림 5 연속적인 핸드오프와 손실 이벤트 속도 p의 변화

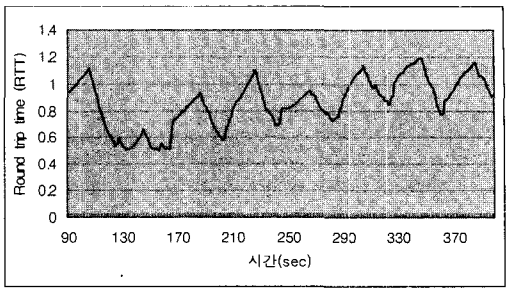


그림 6 연속적인 핸드오프와 TFRC 플로우의 RTT 변화

백 되고 이 두 값은 송신단의 율제어 등식에 적용되어 패킷 전송률을 조절하게 된다. 그림 5에서 핸드오프가 거듭됨에 따라 핸드오프로 인해 발생하는 패킷 손실 때문에 손실 이벤트 속도가 점차 증가하는 것을 살펴볼 수 있다.

그림 6은 역시 그림 1, 2에서와 동일한 실험을 수행하면서 TFRC 플로우의 RTT 값을 측정한 결과를 보인 것이다. RTT 값은 핸드오프가 거듭되어도 거의 변화가 없음을 알 수 있다. 따라서 본 논문에서는 핸드오프를 거듭할수록 처리율이 점차 낮아지는 현상의 주원인이 수신단에서 측정하는 손실 이벤트 속도의 증가라고 규정하고 문제를 해결해 보고자 한다.

3. 무선 이동 환경을 위한 TFRC 개선 방안

본 장에서는 2장에서 설명한 무선 이동 환경에서의 TFRC 혼잡제어의 문제점을 해결하기 위한 방안을 제시한다. 이 방안은 핸드오프 시 발생한 손실에 대한 처리와 핸드오프 완료 직후 적용하는 혼잡제어의 수정으로 나눌 수 있다.

3.1 핸드오프 시 발생한 손실에 대한 처리 방안

핸드오프는 연속적인 패킷 손실이라는 문제점을 안고 있다. 즉 이동호스트가 다른 셀로 이동하게 되면 기존 외부 에이전트로 전송된 패킷들이 경로 변화로 인해 목적지 호스트에 도달하지 못하고 손실되는 문제가 발생한다. 수신단 TFRC에서는 패킷 손실을 손실 히스토리에 기록하고 손실 이벤트를 구성하는데, 이와 같이 핸드오프 시 발생하는 패킷 손실도 구별 없이 손실 이벤트를 구성하게 된다. 새로 구성된 손실 이벤트는 손실 이벤트 속도 값을 변화시키며 이 값이 변화되면 수신단에서는 즉시 피드백 패킷을 송신단에게 보내어 혼잡에 대응하도록 한다. 따라서 핸드오프로 인한 손실이 발생한 경우에도 이러한 혼잡제어가 구동되어 불필요하게 전송률을 감소시켜 처리율이 낮아지게 된다. 특히 핸드오프가 거듭됨에 따라 손실 이벤트 속도값이 점차 증가하며 이 값이 송신측 전송률 공식에 지속적으로 영향을 주어 심각한 처리율 저하 현상이 발생하게 된다.

이를 해결하기 위해 제안하는 방안에서는 수신단 TFRC에서 핸드오프 시 발생하는 패킷 손실은 패킷 손실 히스토리에 기록하지 않고, 핸드오프가 일어나기 전의 손실 이벤트 속도 값을 그대로 피드백 하도록 하였다. 결국 핸드오프로 인한 패킷 손실을 송신단에게 숨김으로써 혼잡이 아닌 상황에서의 혼잡제어 구동을 방지하도록 한 것이다. 이 방법은 Mobile IP와 수신단 TFRC간의 연동을 통해 가능하다. 이동호스트의 Mobil IP는 기존에 유지하던 CoA(Care of Address)가 아닌 CoA를 담은 등록 응답 메시지를 받으면 핸드오프의 완료를 인식하고 이 정보를 수신단 TFRC에게 시그널을 통해 알린다. Mobile IP의 시그널을 수신한 TFRC는 자신이 유지하는 *handoff_flag*를 '1'로 바꾼다. 그리고 *handoff_flag*가 1인 상태에서 패킷을 수신했을 때, 기다리고 있는 패킷 순서 번호보다 더 큰 번호의 패킷이 도착하면 핸드오프로 인해 패킷 손실이 발생했다고 간주하고 손실 히스토리에 이를 손실로 기록하지 않는다. 따라서 손실 이벤트를 새로 구성하지 않으므로 손실 이벤트 속도는 핸드오프가 일어나기 전과 동일한 값으로 유지된다. 즉, 송신단에게 핸드오프 이전의 손실 이벤트 속도가 그대로 피드백되고, 송신단에서는 기존의 전송률로 패킷을 전송하게 된다. 그림 7은 이와 같은 메커니즘을 도식한 것이다.

3.2 핸드오프 직후의 혼잡 처리 방안

TFRC 혼잡제어를 사용하는 이동호스트가 상대적으로 가용대역폭이 큰 셀에서 작은 셀로 이동할 경우 가용대역폭 차이 정도에 따라 크고 작은 혼잡을 경험하게 된다. 이러한 혼잡은 패킷 손실을 야기하고 수신단의 손실 이벤트 속도를 변화시켜 결국 송신단의 전송률 저하를 가져오게 된다. 또한 수신단 처리율인 X_{recv}이 상당히 감소하게 되는데, 송신단 TFRC의 전송률이 $(2\times X_{recv})$로 제한되기 때문에 송신단의 전송률도 크게

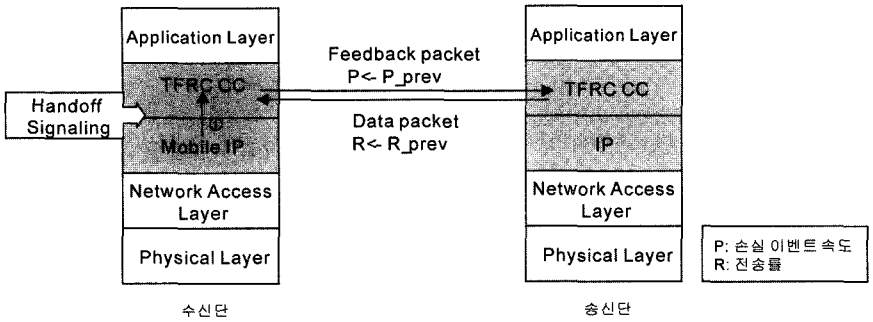


그림 7 핸드오프로 인한 패킷 손실 처리 방안

하락하게 된다. 그런데, TFRC는 세션 시작 시에만 Slow Start에 의하여 전송률을 증가시키고, 세션 진행 중의 혼잡으로 인한 전송률 하락에 대해서는 Congestion Avoidance 방식을 따라 전송률을 증가시키기 때문에 핸드오프로 인해 혼잡이 발생한 경우 새로운 셀에서 가용한 대역폭을 신속하게 차지할 수 없다.

이와 같은 핸드오프 직후의 혼잡으로 인한 전송률 저하에 대해서는 핸드오프 시 발생하는 손실에 대한 대처 방안과 동일한 방안, 즉 손실 기록을 삭제하여 송신단에게 알리지 않는 방법을 사용하는 것이 불합리하다. 이러한 정책은 이동한 셀에서의 과부하를 가중시켜 더 심각한 혼잡을 야기하기 때문이다. 따라서 핸드오프 직후 핸드오프로 인한 혼잡으로 수신측 처리율이 급격히 감소할 경우에는 좀 더 빠른 속도로 전송률을 회복하여 공정한 몫을 빠르게 점유할 수 있도록 하는 것이 더 적합하다. 또한, 핸드오프가 발생한 경우는 세션 시작 때와 마찬가지로 새로운 환경에서 플로우가 전송되기 시작하는 것이므로 전송률을 좀 더 빠르게 적응시키는 것이 필요하다고 보았다.

한편, 매 핸드오프마다 새로운 셀에서의 혼잡 발생을 피하기 위해 미리 전송률을 감소시키고 Slow Start를 적용하는 방법을 취하지 않고, 실제로 혼잡이 발생한 경우에만 사후 처리하도록 하는 방안을 사용하는 이유는 핸드오프로 인한 혼잡이 발생하지 않는 경우에 미리 전송률을 불필요하게 감소시키지 않기 위함이다. 멀티미디어 응용은 혼잡으로 인해 패킷이 손실되어도 재전송에 대한 요구가 엄격하지 않으므로 재전송으로 인한 네트워크의 혼잡 가중 현상이 심각하지 않다. 전송률을 좀 더 빠르게 회복하기 위한 방안은 구체적으로 다음과 같다.

기존 TFRC의 경우 송신단은 수신단으로부터 RTT 계산을 위한 타임스탬프 정보 외에도 수신측 처리율인 X_{recv}와 손실 이벤트 속도 p를 제공받는다. 이러한 정보를 이용해 송신단 TFRC는 현재 상황에서 적합한 전송률인 X_{calc}을 계산하여 X_{calc}가 현재 전송률인 X_{curr}보다 더 크면 RTT마다 최대 한 개의 패킷씩 전송률을 증가시킨다. 엄밀히 말하자면 RTT값이 고정되어 있고 history discounting이 없을 경우, TFRC 송신단은 RTT마다 약 0.14개의 패킷씩 전송률을 증가시킨다. 비혼잡 상황이고 history discounting이 시작되면 TFRC 송신단은 RTT마다 약 0.22개의 패킷씩 전송률을 증가시킨다[1]. 제안하는 방안에서는 핸드오프로 인한 혼잡으로 패킷 손실이 발생하였을 때는 좀 더 경쟁력 있게 혼잡 후 전송률을 증가시키기 위해서 다음의 두 가지 방식을 적용해 보았다. 수신단 처리율이 핸드오프 이전 처리율의 절반에 이르거나, 새로운 손실이 발생할 때까지 송신단에서 RTT마다 $\alpha(\alpha>=1)$개의 패킷을 증가시키는 방식과 기존 TFRC의 Slow Start 기법을 초기 전송률에서 그대로 적용하는 방법이 그것이다. 수신단 TFRC에서 피드백 패킷을 통해 송신단 TFRC에게 제공하는 정보에는 기존의 TFRC에서 요구하는 정보 외에 이동 수신단에서 핸드오프가 발생하였음을 표시하는 *handoff_flag*와 이전 피드백 패킷을 전송한 이후부터 패킷 손실이 있었는지를 표시하는 *loss_flag*를 추가시켰다. 송신단 TFRC는 수신한 피드백 패킷에 대해 매번 *handoff_flag*와 X_{recv}를 참조하는데, *handoff_flag*가 세트되어 있고 X_{recv} 값이 이전 셀에서의 처리율인 X_{prev}의 절반보다 작을 경우 송신단에서 유지하는 플래그인 *fast_increase_flag*를 '1'로 세트하고 전송률 증가 방식을 Fast Increase 제어모드로 전환한다. 이때 X_{prev}는 핸드오프 이전의 처리율 값으로 매 피드백 패킷이 올 때마다 피드백 패킷에 표시된 X_{recv}값으로 갱신하다가 일단 Fast Increase 제어모드가 되면 갱신하지 않고 값을 유지한다. 이러한 Fast Increase 제어모드는 전송률이 X_{prev}의 절반에 도달하거나 *loss_flag*가 세트될 때까지 계속되며 송신단이 Fast Increase 제어모드인 동안에는 RTT당 $\alpha(\alpha>=1)$개의 패킷을 증가시키거나 기존

TFRC의 Slow Start 기법을 사용한다. 아래의 표 1은 제안하는 방안에서 Fast Increase 제어모드를 위해 송신단에서 유지하는 정보와 수신단에서 피드백하는 정보, 그리고 Fast Increase 모드의 개시 및 종료 조건을 정리한 것이다.

표 1 Fast Increase 제어 모드

Fast Increase mode	
송신단에서 유지하는 정보	*fast_increase_flag*, X_{prev}
수신단에서 피드백하는 정보	*handoff_flag*, *loss_flag*
모드 개시 조건	(*fast_increase_flag* == 1) && ($X_{recv} < \frac{1}{2} X_{prev}$)
모드 종료 조건	($X_{recv} < \frac{1}{2} X_{prev}$) || (*loss_flag* == 0)

제안하는 방안을 포함하는 송신단 혼잡제어모드 변화를 도식화하면 그림 8과 같다. TFRC는 데이타 플로우의 초기화 시 TCP만큼 경쟁적으로 대역폭을 차지하기 위해 Slow Start 알고리즘을 사용한다. 이러한 Slow Start 제어모드는 *loss_flag*가 세트되면 종료되며 이후부터 기존 TFRC의 제어모드 혹은 Fast Increase 제어모드에서 동작하게 된다. Fast Increase 제어모드는 수신한 피드백의 *handoff_flag*가 '1'이고 현재 수신측 처리율인 X_{recv}가 이전 셀에서의 수신측 처리율인 X_{prev}의 절반보다 작은 경우 시작되어 송신단 전송률이 핸드오프 이전의 수신측 처리율인 X_{prev}의 절반 이상으로 회복되거나 *loss_flag*가 '1'로 세트될 때까지 계속된다. 제안하는 방안은 다음의 장점을 지닌다.

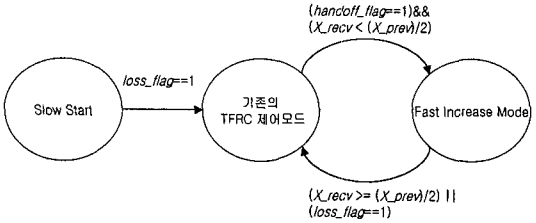


그림 8 송신단 혼잡제어 모드의 변화

i) 종단 호스트들의 소프트웨어를 최소한의 정도로 수정하면 된다.

ii) 기지국이나 다른 중간 라우터들과 같은 네트워크 요소들로부터 특별한 지원을 요구하지 않는다.

iii) 프로토콜 오버헤드의 크기에는 거의 변화가 없다. 즉 피드백 패킷에 *handoff_flag*와 *loss_flag* 필드 2비트를 추가하면 된다.

iv) 핸드오프가 발생한 경우에만 제안하는 방안이 구동되므로 기존에 검증된 유선망에서의 성능에 전혀 영향을 주지 않는다.

그림 9는 3.1과 3.2의 방안에서 사용한 *handoff_flag*와 *fast_increase_flag*의 상태 변화를 도식화한 것이다. 먼저 수신단의 Mobile IP는 CoA의 변화로 핸드오프를 인식하고 수신단의 TFRC에게 시그널을 통해 알리게 된다. 수신단의 TFRC는 자신의 *handoff_flag*를 '1'로 셋하고 피드백 패킷에 그 정보를 담아 송신단 TFRC에게 보내고 자신의 *handoff_flag*를 '0'으로 리셋한다. 송신단 TFRC는 피드백 패킷의 *handoff_flag*를 참조하여 '1'로

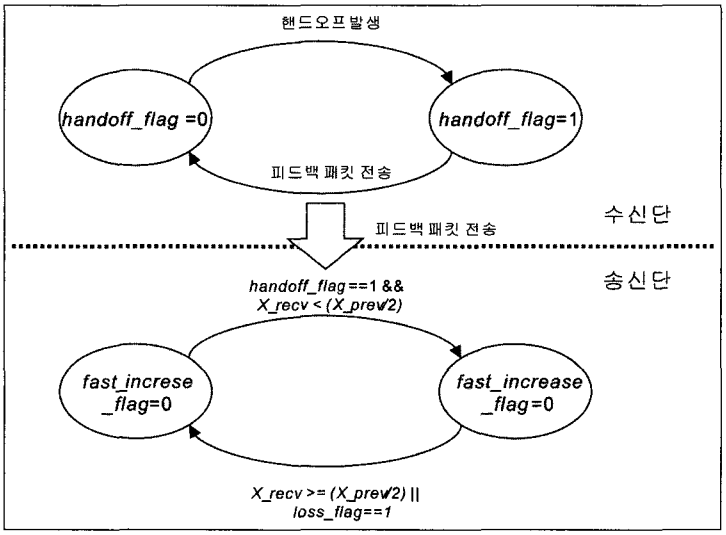


그림 9 플래그 상태 변화

세트되어 있으면 자신이 유지하는 *fast_increase_flag*를 '1'로 세트한다. 전송률이 핸드오프 이전 처리율의 절반으로 회복되거나 *loss_flag*가 세트되면 *fast_increase_flag*는 '0'으로 리셋된다.

4. 시뮬레이션

4장에서는 제안하는 방안의 성능을 평가하기 위한 시뮬레이션 모델을 설명하고 시뮬레이션 결과를 분석한다. 본 시뮬레이션은 Sun Solaris 워크스테이션에서 수행되었으며, 시뮬레이션 도구로는 대표적인 TCP/IP 네트워크 시뮬레이션에 적합한 도구인 버클리 대학(U.C. Berkely)의 NS(Network Simulator)-2[11]를 이용하였다.

4.1 시뮬레이션 모델

그림 10은 시뮬레이션에서 사용된 네트워크 토폴로지 모델을 보여주고 있다. 이 네트워크는 하나의 홈에이전트(HA, Home Agent)와 두 개의 기지국(BS, Base Station) 그리고 하나의 통신원(CN, Correspondent Node)로 구성된다. CN, HA 그리고 무선 게이트웨이는 인터넷을 통해 연결되었다고 가정하고 이를 전파지연과 대역폭이 각각 100ms와 10Mbps인 링크로 모델링하였다. 무선 게이트웨이에서 기지국까지의 한 홉에 해당하는 유선 링크의 전파지연과 대역폭은 각각 0.1ms와 10Mbps라 가정하였다. 기지국과 이동 호스트간의 무선 링크의 전파지연과 대역폭은 각각 25us와 2Mbps라 가정하였다. 셀 반지름과 셀 간 오버랩 구간은 각각 200m와 50m로 설정하였다. 무선 링크의 특성에 의한 패킷

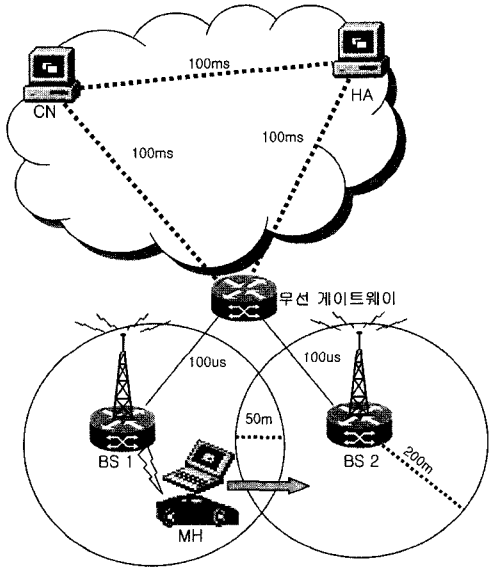


그림 10 시뮬레이션 네트워크 모델

손실을 방지하기 위해 유·무선 링크에서 발생하는 전송오류 발생 확률은 0이라 가정하였다.

기지국이 관할하는 각각의 셀에 TFRC 혹은 제안하는 혼잡제어를 사용하는 이동호스트(이들을 TFRC 호스트라 부르기로 함) 한 개와 셀 간 가용 대역폭 양의 차를 조장하기 위한 이동호스트(이들을 외부 이동호스트라 부르기로 함)를 배치하였다. 셀간 가용 대역폭 차이를 조장하기 위해 한 셀에 외부 이동호스트를 집중시켰으며, 외부 이동호스트로는 TCP 플로우가 전달되고 TCP 혼잡제어가 사용된다고 가정하였다. TCP 송신단은 무선 TCP 수신자에게 FTP로 파일을 전송하도록 했으며, TCP에서의 연속적인 데이타 전송을 위해 TCP의 윈도우 사이즈는 다음과 같이 설정하였다.

$$window\ size = \frac{RTT \times bottleneck\ bandwidth}{8 \times packet\ size}$$

또한 가용 대역폭 양의 차이 정도를 변화시켜 보기 위해 외부 이동호스트의 수를 0부터 4까지 변화시켜 보았다. TFRC 트래픽과 TCP 트래픽 모두 데이타 패킷의 크기는 1000Bytes이고 ACK 패킷의 크기는 40 Bytes라 가정하였다.

4.1.3 성능 측정 메트릭

TFRC 트래픽의 성능 측정을 위한 메트릭은 AT (Average Throughput), PLR(Packet Loss Ratio), 그리고 Normalized Throughput이다. Normalized Throughput은 TFRC와 TCP 플로우간 공정성을 측정하기 위한 성능치이다. 이들은 다음과 같이 정의된다.

$$AT = \frac{\text{수신한 총패킷의 양}}{\text{패킷 전송 시간}},$$

$$PLR = \frac{\text{손실된 패킷의 수}}{\text{총 발생된 패킷의 수}},$$

$$Nomalized\ Throughput = \frac{\text{전체 } flow\text{의 수} \times \text{각 } flow\text{의 전송률}}{bottleneck\ link\ bandwidth}$$

4.2 시뮬레이션 결과

4.2.1 Increase Factor α 선정

먼저, Fast Increase 제어모드에서 사용할 증가분 α 값의 적정치를 파악하기 위한 실험을 수행하였다. 그림 11은 α가 1~4인 각각의 경우에 대하여 20m/s의 이동속도를 갖는 TFRC 호스트가 가용대역폭이 다른 셀 사이를 8회 핸드오프 하는 동안의 평균처리율을 보인 그래프이다. 제안하는 방안의 처리율은 가용대역폭 차이에 상관없이 α가 클수록 높다는 것을 알 수 있다. 그러나 α=1인 경우와 α>1인 경우의 처리율 차이에 비해 α=2, 3, 4인 경우간의 처리율 차이는 상대적으로 적었다. 또한 α에 상관없이 가용대역폭 차이가 커질수록 처리율이 감소하는 추세를 보이지만 역시 감소의 정도도 α=1인 경우가 가장 크고, α>1인 경우들은 서로 큰 차이가 없었다.

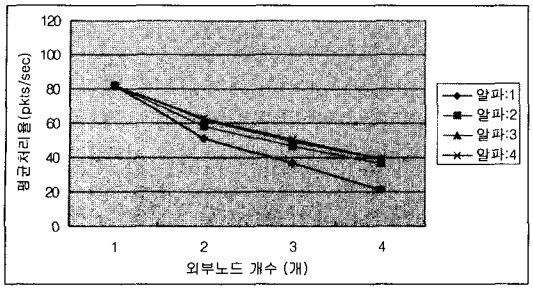


그림 11 α 변화에 따른 평균처리율 변화

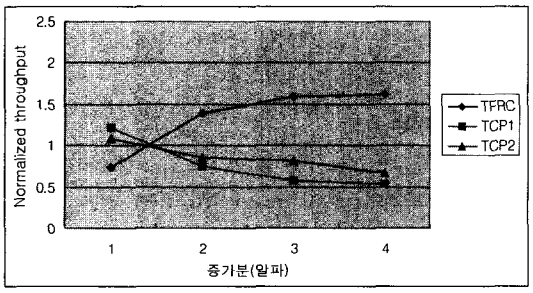


그림 12 α 변화에 따른 공정성 변화
(외부 이동호스트가 2개인 경우)

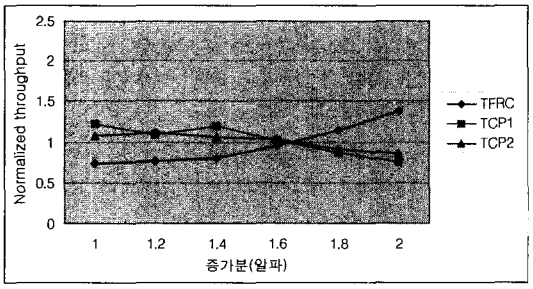


그림 13 α 변화에 따른 공정성 변화
(외부 이동호스트가 2개인 경우)

α가 공정성에 미치는 영향을 살펴보기 위해 TCP 혼잡제어를 사용하는 외부 이동호스트를 한 셀에만 2~4개 상주시켜 이동하는 셀 간 가용대역폭 차를 고정시키고, TFRC 이동호스트가 가용대역폭이 다른 셀 간에 8회에 걸쳐 핸드오프를 경험하도록 한 후 상대적으로 가용대역폭이 작은 셀 즉, 외부 이동호스트들이 통신하고 있는 셀로 진입해 300초간 머물도록 하면서 머무르는 동안의 평균처리율을 측정하고 정규화하였다. 그림 12~15에서 보듯이 실험한 모든 경우에 대하여 대략 α =1.6 인 경우 공정성 면에서 가장 우수한 성능을 보였다. 이에, 그림 12~15의 결과를 바탕으로 이후의 성능 비교실험에서는 모두 제안하는 방안의 α를 1.6으로 설정하였다.

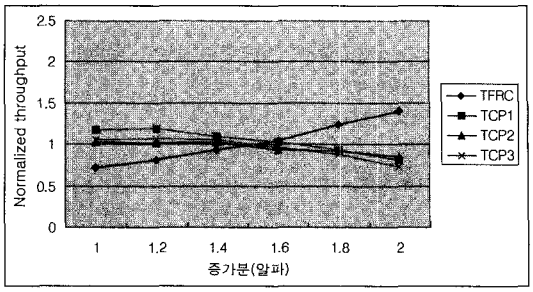


그림 14 α 변화에 따른 공정성 변화
(외부 이동호스트가 3개인 경우)

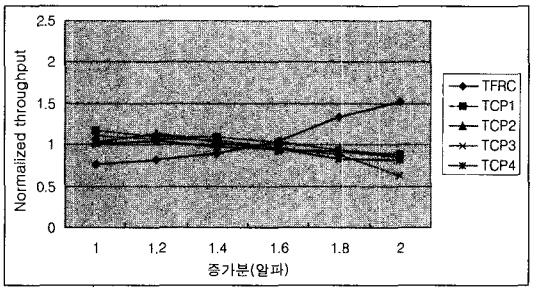


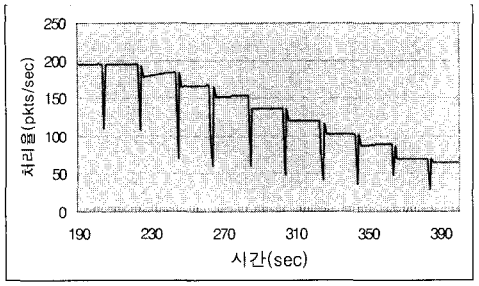
그림 15 α 변화에 따른 공정성 변화
(외부 이동호스트가 4개인 경우)

4.2.2 셀간 가용대역폭 차이가 다른 경우들에 대한 성능 측정

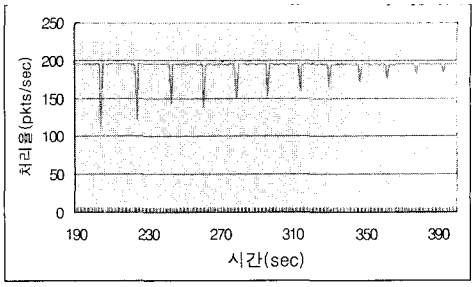
그림 16~18은 제안하는 방안을 적용하지 않은 경우와 적용한 경우 각각에 대하여 시간에 따른 수신측의 처리율 변화를 보인 것이다. 그림 16은 두 셀에 모두 외부 이동호스트가 없는 경우에 대한 결과이다. 그림에서 알 수 있듯이 TFRC는 핸드오프를 거듭할수록 핸드오프 시에 발생하는 패킷 손실로 인해 처리율이 점차 낮아짐을 알 수 있다(그림 16(a)). 그림 16(b)에서 제안하는 방안을 적용한 경우에는 거듭되는 핸드오프에도 처리율이 거의 일정함을 알 수 있다.

그림 17은 외부 이동호스트를 1개만 두어 상대적으로 셀 간 가용대역폭 차가 적을 때 시간에 따른 수신측 처리율 변화를 보인 그래프이다. 가용대역폭 차가 작을 때는 혼잡으로 인한 심각한 처리율 감소현상이 없기 때문에 제안하는 방안을 적용하지 않은 순수 TFRC는 그림 16의 경우와 비슷한 유형으로 처리율이 점차적으로 감소하는 추세를 보인다(그림 17(a)). 그림 17(b), 17(c)에서는 제안하는 방안을 적용함으로써 핸드오프가 반복되더라도 처리율이 거의 저하하지 않음을 볼 수 있다.

그림 18은 외부 이동호스트를 한 셀에 3개 상주시켜 상대적으로 셀 간 가용대역폭 차가 클 때 시간에 따른 수신측 처리율 변화를 보인 그래프이다. 가용대역폭 차

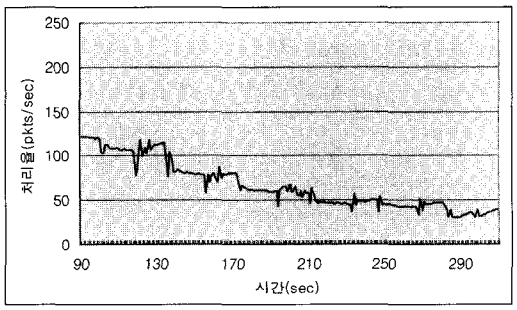


(a) 제안하는 방안을 적용하지 않은 경우

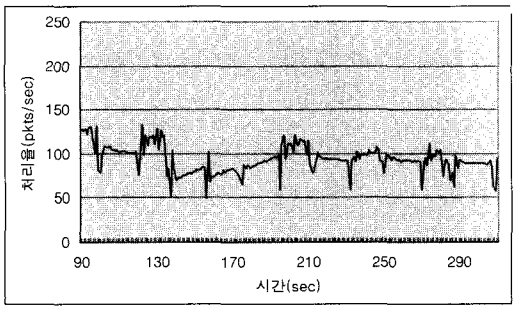


(b) 제안하는 방안을 적용한 경우

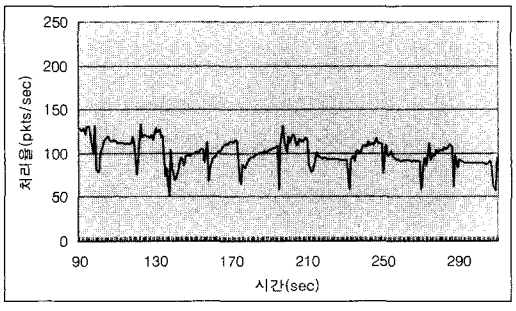
그림 16 가용대역폭 차가 없는 셀간 이동시 수신측 처리율 변화



(a) 제안하는 방안을 적용하지 않은 경우

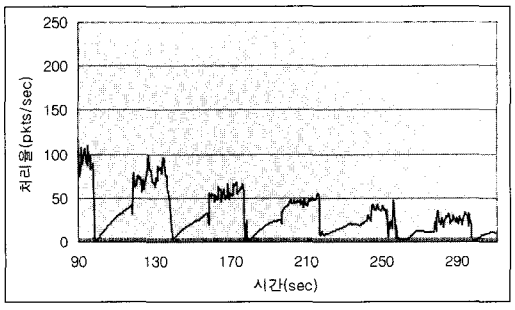


(b) 제안하는 방안을 적용한 경우(α 증가)

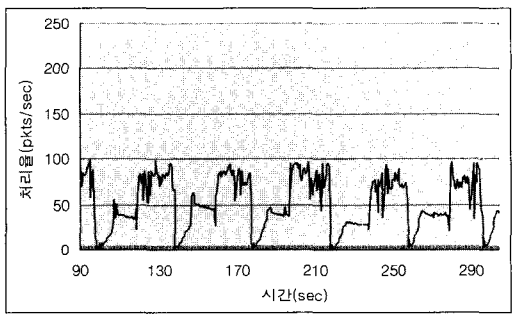


(c) 제안하는 방안을 적용한 경우(Slow Start)

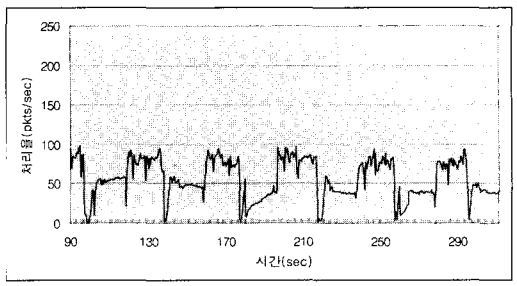
그림 17 외부 이동호스트 개수 차가 1개인 셀간 이동시 수신측 처리율 변화



(a) 제안하는 방안을 적용하지 않은 경우



(b) 제안하는 방안을 적용한 경우(α 증가)



(c) 제안하는 방안을 적용한 경우(Slow Start)

그림 18 외부 이동호스트 개수 차가 3개인 셀간 이동시 수신측 처리율 변화

가 클 때는 제안하는 방안을 적용하지 않은 순수 TFRC의 경우 그림 17에서의 유형과 다르게 혼잡으로 인해 심각한 처리율 저하문제가 발생하고, 가용대역폭이 큰 셀에서 작은 셀로 진입 시 수신측 처리율이 회복되는데 오랜 시간이 걸림을 볼 수 있다(그림 18(a)). 그림 18(b), 18(c)에서 제안하는 방안을 적용한 경우에는 Fast Increase 제어모드로 인해 혼잡으로 인해 하락한 처리율을 빠르게 회복함을 볼 수 있다. 특히 Slow Start를 적용한 경우는 α 증가보다 더 빠르게 처리율을 회복함을 볼 수 있다(그림 18(c)).

그림 19~21은 가용대역폭 차가 각각 다른 경우에 대해 제안하는 방안을 적용하지 않은 경우와 적용한 경우의 평균처리율, 패킷손실률, 공정성을 측정한 결과를 보

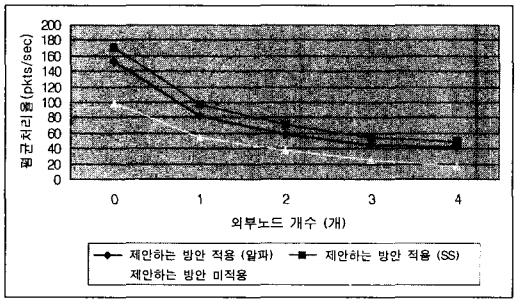


그림 19 가용대역폭 차이 변화에 따른 평균처리율 변화

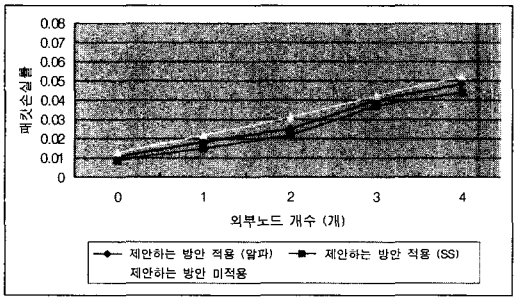


그림 20 가용대역폭 차이 증가에 따른 패킷손실률 변화

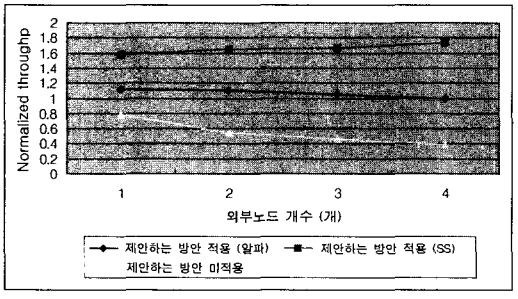


그림 21 가용대역폭 차이 증가에 따른 공정성 변화

인 그래프이다. 평균처리율과 패킷손실률은 20m/s의 이동속도를 갖는 TFRC 호스트가 용량이 다른 셀 사이를 8회 핸드오프 하는 동안에 측정한 값이다. 공정성은 8회 핸드오프 후 저용량 셀에서 300초 머무르는 동안의 평균처리율을 정규화한 값이다. 그림 19에서 가용대역폭 차가 클수록 두 방안 모두 처리율은 점차 낮아지는 경향을 보이나 제안하는 방안을 적용한 경우가 평균 처리율이 항상 높다는 것을 알 수 있다. 그림 20에서 제안하는 방안이 기존 TFRC에 비해 패킷손실률이 항상 낮게 나온다. 이것은 제안하는 방안이 손실 자체를 줄이기 위한 대책을 제공하는 것은 아니므로 손실된 패킷의 수는 그대로이며 송신단의 전송률 증가로 인해 총 발생된 패킷의 수가 증가하기 때문이다. 특히 Fast Increase 모드에서 Slow Start를 사용한 경우 평균처리율과 패킷손실률 면에서 성능이 우수함을 알 수 있다. 그림 21에서 기존 TFRC의 경우와 Fast Increase 모드에서 Slow Start를 적용한 경우 α 증가분을 사용한 경우에 비해 공정성이 훨씬 낮고 가용대역폭 차가 커짐에 따라 공정성이 점차 감소함을 알 수 있다.

4.2.3 이동호스트가 셀 내 머무른 시간이 다른 경우들에 대한 성능 측정

그림 22와 23은 TFRC 호스트가 각 셀 내에서 머무르는 시간을 변화시켜 보면서 제안하는 방안을 적용하지 않은 경우와 적용한 경우의 평균처리율, 패킷손실률을 측정한 결과를 보인 그래프이다. 이 실험에서는 한 셀에만 외부 이동호스트를 두 개 상주시켜 셀 간의 가용대역폭 차를 고정시켰다. 평균처리율과 패킷손실률은 20m/s의 이동속도를 갖는 TFRC 호스트가 가용대역폭이 다른 두 셀 사이를 8회 핸드오프 하는 동안 측정하였다. 그림 22에서 제안하는 방안을 적용한 경우나 하지 않은 경우 모두 TFRC 호스트가 각 셀에서 머무른 시간에 따라서는 처리율이 크게 달라지지 않음을 볼 수 있고, 제안하는 방안을 적용한 경우 처리율이 항상 더 높음을 알 수 있다. 그림 20에서 본 결과와 유사하게 패킷손실률은 제안하는 방안의 경우가 더 낮다. 특히 제안

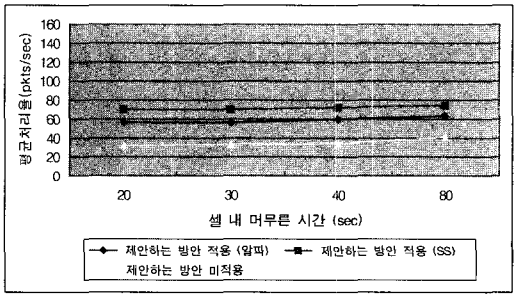


그림 22 셀 내 머무른 시간 증가에 따른 평균처리율 변화

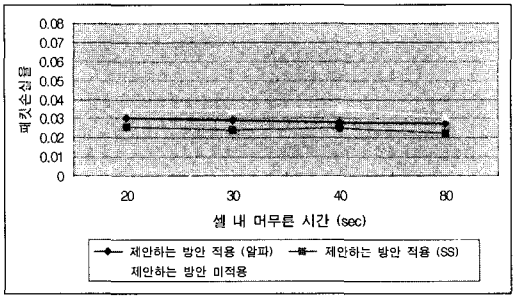


그림 23 셀 내 머무른 시간 증가에 따른 패킷손실률 변화

하는 방안 중 Fast Increase 모드에서 Slow Start를 사용한 경우 평균처리율과 패킷손실률 면에서 성능이 우수함을 알 수 있다.

4.2.4 핸드오프 횟수가 다른 경우들에 대한 성능 측정

그림 24~26은 핸드오프 횟수를 변화시켜 보면서 각각 평균처리율과 패킷손실률 그리고 공정성을 측정한 결과를 보인 그래프이다. 이 실험에서는 외부 이동호스트를 한 셀에 2개 상주시켜 셀 간 가용대역폭 차를 고정시켰고 TFRC 호스트의 속도는 20m/s로 고정시켰다. 그림 24에서 제안하는 방안을 적용한 경우나 적용하지 않은 경우 모두 핸드오프 횟수가 늘어날수록 처리율이 낮아지는 경향을 보임을 알 수 있다. 그러나 핸드오프 횟수에 상관없이 항상 제안하는 방안의 처리율이 더 높

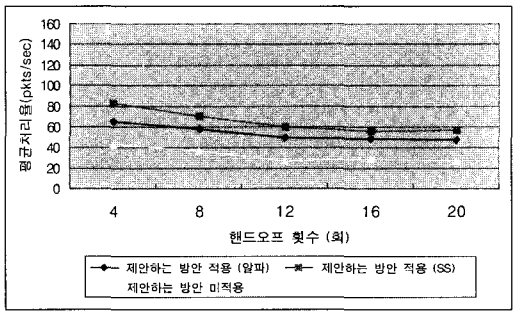


그림 24 핸드오프 횟수 증가에 따른 평균처리율 변화

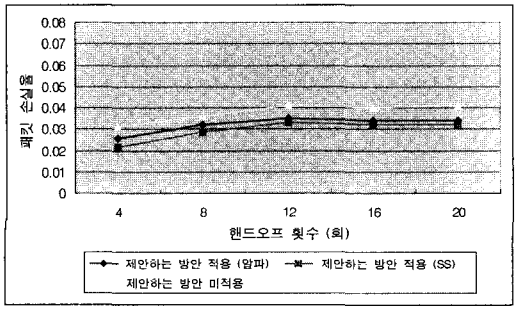


그림 25 핸드오프 횟수 증가에 따른 패킷손실률 변화

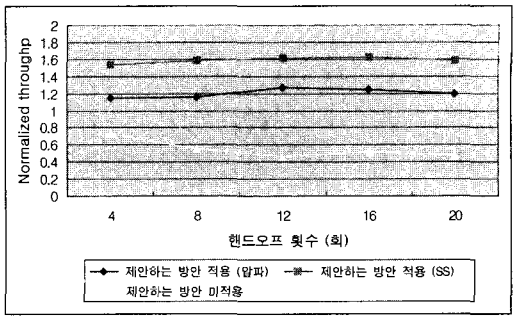


그림 26 핸드오프 횟수 증가에 따른 공정성 변화

다. 패킷손실률에 있어서는 앞에서의 결과들과 유사하게 제안하는 방안의 경우가 더 낮다. 특히 제안하는 방안 중 Fast Increase 모드에서 Slow Start를 사용한 경우 평균처리율과 패킷손실률 면에서 성능이 우수함을 알 수 있다. 그림 26에서 핸드오프가 거듭됨에 따라 기존의 TFRC는 TCP 트래픽에 비해 대역폭 할당에 있어서 더 불리한 입장이 되어 공정성이 저하됨을 알 수 있다. 또한 Fast Increase 모드에서 Slow Start를 사용한 경우 대역폭 할당에 있어서 더 유리한 입장이 되어 공정성이 저하됨을 알 수 있다. 그러나 Fast Increase 모드에서 증가분 α를 사용한 경우에는 공정성이 거의 그대로 유지됨을 알 수 있다.

5. 결 론

무선 이동 환경에서 TFRC 혼잡제어를 사용하는 이동호스트가 핸드오프를 거듭할 때 처리율 및 공정성에서 현저한 성능 저하가 있음을 확인하였다. 이러한 문제는 핸드오프로 인한 패킷 손실 기록이 송신단의 전송률 도출 공식에 지속적으로 영향을 주기 때문이다. 또한 이동해간 셀에서 핸드오프로 인해 혼잡이 발생하는 경우 핸드오프 후에도 전송률이 상당히 떨어지게 되는데 TFRC는 세션 진행 중의 혼잡에 대해서는 Congestion Avoidance에 의해 전송률을 증가시키기 때문에 핸드오프 후에도 전송률을 빠르게 회복하지 못하게 되어 처리율과 공정성이 저하하게 된다.

이에 본 논문에서는 무선 이동 환경에서의 TFRC 성능 향상을 위해 핸드오프로 인한 패킷 손실에 대해 대처하는 방안과 핸드오프 직후 구동되는 혼잡제어 방안을 제안하였다. 먼저 핸드오프로 인한 패킷 손실에 대해서는 수신단에서 그 기록을 삭제하여 송신단에게 피드백하지 않음으로써 손실 이전 전송률로 데이타를 송신할 수 있도록 하였고, 핸드오프 직후 발생하는 혼잡에 대해서는 기존 TFRC의 전송률 증가 패턴보다 좀 더 빠르게 전송률을 증가시킴으로써 지속적인 저 처리율과

공정성 저해 문제가 개선되도록 하였다. 시뮬레이션 결과 Fast Increase 모드에서 α 증가분과 Slow-Start를 사용하는 기법간에는 공정성과 처리율에 있어 성능차이가 있음을 볼 수 있었다. TCP 트래픽과의 대역폭 분할에 있어서 공정성 유지가 중요할 경우 Fast Increase 모드에서 α증가분을 사용하는 것이 좋으며, 공정성 저해에도 불구하고 더 높은 처리율을 요구할 경우 Slow-Start를 선택하는 것이 더 나음을 볼 수 있었다.

참 고 문 헌

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

[2] S. Floyd, M. Handley, J. Padhye, and J. Widmer, "Equation-Based Congestion Control for Unicast Applications: the Extended Version," ICSI tech report TR-00-03, March 2000.

[3] 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.

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

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

[6] J. Padhye, D. Kurose, and R. Towsley, "A model based TCP-friendly rate control protocol," Proc. International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV), June 1999.

[7] A. Bakre and B.R. Badrinath, "I-TCP: Indirect TCP for Mobile Hosts," Proc. 15th International Conf. on Distributed Computing Systems(ICDCS), May 1995.

[8] H. Balakrishnan, S. Seshan, and R.H. Katz, "Improving Reliable Transport and Handoff Performance in Cellular Wireless Networks," ACM Wireless Networks, 1(4), December 1995.

[9] Ramon Caceres and Liviu Iftode, "Improving the Performance of Reliable Transport Protocols in Mobile Computing Environments," IEEE Journal on selected areas in communications, vol. 13, No.5, June 1995.

[10] Kevin Brown and Suresh Singh, "M-TCP: TCP for Mobile Cellular Networks," ACM Computer Communication Review, 1997.

[11] http://www.isi.edu/nsnam/ns/index.html.

최 미 라

2001년 이화여자대학교 컴퓨터학과 졸업(학사). 2003년 이화여자대학교 과학기술대학원 컴퓨터학과(공학석사). 2003~현재 삼성전자 정보통신 총괄 근무. 관심분야는 QoS routing, 혼잡제어 메커니즘, VoIP

이 미 정

정보과학회논문지 : 정보통신 제 30 권 제 2 호 참조