R-CAT : P2P기반 스트리밍 환경에서 노드의 능력을 고려한 내구적 멀티캐스트 트리 생성 기법

김 은 석[†] · 한 세 영[††] · 박 성 용[†††]

요 약

최근 스트리밍 서비스는 인터넷 트래픽의 많은 부분을 차지할 정도로 인기 있는 서비스가 되었고, 확장성을 위해 P2P기반의 스트리밍 서비스가 제안되었다. P2P기반 스트리밍 환경은 빈번한 피어들의 떠남과 합류가 일어나므로, 이러한 멀티캐스트 그룹의 변화에 대처하기 위해서 다중 멀티캐스트 트리를 구축하게 된다. 하지만 기존의 방법들은 노드의 능력 차이를 고려하지 않았기 때문에 트리가 길어지고, 불안정해질 수 있다. 따라서 본 논문은 노드의 능력을 고려한 내구적 멀티캐스트 트리 생성 기법(R-CAT)을 제시하여 우수 노드를 트리의 상층부에 위치시킴으로써 트리의 길이를 줄이고 트리 내구성이 강한 멀티캐스트 트리를 구성하게 한다. 또한 시뮬레이션을 통해 제안한 R-CAT 기법이 기존의 SplitStream 기법에 비해 노드 합류에 드는 오버헤드는 늘어나지만 결과 트리의 종단간 평균 지연시간이나 손실되는 패키 수는 훨씬 적어 성능과 내구성 면에서 향상을 가져옴을 보일 수 있었다.

키워드 : 피투피 스트리밍, 멀티캐스트 트리, 안정성

R-CAT: Resilient Capacity-Aware Multicast Tree Construction Scheme

Eunseok, Kim[†] · Saeyoung Han[††] · Sungyong Park[†††]

ABSTRACT

Recently, streaming service accounts for large part of internet traffic and it is becoming the most popular service. Because of P2P's scalability, P2P-based streaming system is proposed. There are frequent leave and join of a node. To overcome the group dynamics, Multiple Multicast Trees Methods were suggested. However, since they did not consider discrepancy in peers' capacity, it may cause the trees to be long and unstable. So we suggest Resilient Capacity-Aware Multicast Tree construction scheme (R-CAT) that promotes superior peer to upper position in the tree and construct more stable and short multicast trees. By simulation we can show that R-CAT cost more overhead packets for tree joining process, but it reduce the end-to-end delay of the resulting tree and the number of packets lost during the node joining and leaving processes much more than SplitStream.

Key Words : Peer-to-peer Streaming, Multicast Tree, Resiliency

1. 서 론

최근 스트리밍 서비스는 웹 서비스와 더불어 인터넷 트래픽의 많은 부분을 차지하며 인기 있는 서비스가 되어가고 있다. 일반적인 서버-클라이언트 구조와는 달리 피어-투-피어 시스템의 확장성 높은 통신 방법으로 인해, 피어-투-피어 기반의 스트리밍 시스템에 대하여 많은 연구가 이루어졌다.

일반적으로 피어-투-피어 스트리밍 시스템은 멀티캐스트 구조를 생성하고 유지하는데 사용된 알고리즘에 따라 분산 알고리즘(Distributed algorithm)기반과 중앙집중식 알고리즘(Centralized algorithm)기반으로 분류될 수 있다. 중앙집중식 알고리즘 기반의 스트리밍 시스템에서는 모든 수신자들이 자신이 데이터를 전달 받아야 하는 노드(parent node, 부모노드)와 전달해야 하는 노드(children node, 자식노드)들에 대한 정보를 중앙 서버로부터 획득하는데, 대표적인 중앙 집중식 알고리즘 기반의 스트리밍 시스템으로는 CoopNet[6] 등이 있다. 이 시스템에서는 중앙 서버가 모든 피어들의 상황과 정보를 파악하고 있기 때문에 효율적인 멀티캐스팅 구조를 만들 수 있으나, 모든 피어들의 정보를 파악하기 위해서 많은 비용이 들고 부하가 집중되므로 확장성(scalability)이 떨어진다.

따라서 부모 노드와 자식노드를 각 수신자 자신이 결정(Self-organization)하는 분산 알고리듬기반이 연구되었는데, 대표적인 시스템으로는 SplitStream[7], SpreadIt[10] 등이 있다. 이 시스템은 각각의 피어들이 자신의 부모노드와 자식노드

※ 본 연구는 서강대학교 산업기술연구소의 지원으로 수행되었음.
† 정 회 원 : LG전자 기술원 연구원
†† 정 회 원 : 서강대학교 컴퓨터 학과 박사과정
††† 정 회 원 : 서강대학교 컴퓨터학과 교수
논문접수 : 2006년 2월 3일, 심사완료 : 2006년 3월 10일

에 대한 정보만 유지하면 되기 때문에 높은 확장성을 보장한다.

또한 분산 알고리듬 기반 스트리밍 시스템은 부하를 분산하고, 멀티캐스트 그룹 변화에 대처하기 위해 다중 멀티캐스트 트리를 적용했다[4, 5]. 이를 통해 노드의 떠남, 합류, 실패(failure), 병목현상 등으로 인해 영향 받는 노드의 수를 최소화하고 패킷손실을 줄여서 전체 시스템의 내구성을 높인다.

하지만 기존의 분산 알고리즘 기반 스트리밍 시스템에서는 멀티캐스트에 참여하는 모든 노드들의 능력이 동등하다 가정하고 그 차이를 고려하지 않았다. 그러나 일반적인 네트워크 트래픽 연구나 스트리밍 시스템의 연구를 통해 각 노드마다 사용 가능 대역폭 등의 능력 차이가 있다는 것을 알 수 있다[1, 2, 12]. 멀티캐스트 트리의 한 노드가 시스템을 떠나거나 병목현상을 일으킨다면, 그 노드를 루트로 하는 서브트리의 모든 노드들은 데이터를 받을 수 없게 되므로, 트리의 상부에 있는 노드일수록 전체 시스템의 내구성에 더 많은 영향을 미치게 된다. 따라서 각 노드들의 능력을 고려하여 능력이 큰 노드일수록 트리의 상부에 위치하게 한다면 트리의 내구성을 한 층 높일 수 있다.

따라서 본 논문에서는 스트리밍에 참여하는 피어들의 전달능력(forwarding capacity)의 차이를 고려해서 멀티캐스트 트리를 구성하는 기법인 내구적 멀티캐스트 트리(Resilient Capability-Aware Multicast Tree, 이하 R-CAT) 생성기법을 제안한다. R-CAT 기법은 기존의 다중 멀티캐스트 트리 기반의 SplitStream을 기반으로 지역정보 갱신 알고리듬, 등반을 통한 부모 선택 알고리듬, 적합성을 고려한 희생자 알고리듬을 추가하여 능력치가 높은 노드를 멀티캐스트 트리의 상층부에 위치할 수 있게 트리를 생성할 수 있게 한다.

R-CAT 기법의 유효성을 보이기 위하여 새로운 노드가 멀티캐스트 트리에 합류할 때 발생하는 오버헤드, 합류하기 까지 걸리는 홉수, 평균 종단간 지연시간, 그리고 손실되는 패킷 수를 측정하여, 기존의 SplitStream과 성능 및 내구성 등을 비교 측정하였다. R-CAT 기법을 추가할 경우 등반을 통한 부모 선택을 위하여 SplitSTream에 비해 추가적인 오버헤드가 발생하고 합류하기 까지 걸리는 홉수는 다소 증가하지만, 결과로 얻어지는 트리의 종단간 평균 지연시간과 합류과정에서 손실되는 패킷의 수는 현저히 줄어드는 것을 볼 수 있었다.

본 논문의 나머지 구성은 다음과 같다. 2장에서는 스트리밍 시스템의 관련 연구 중 하나인 SplitStream에 대해서 알아보고 기존 방법의 문제점을 알아본다. 3장에서는 기존의 문제점을 해결하기 위해 제안된 노드의 능력을 고려한 내구적 멀티캐스트 트리 생성 기법에 대해 알아본다. 4장에서는 시뮬레이션을 통해 기존의 SplitStream과 R-CAT 기법을 추가했을 경우를 성능과 내구성 측면에서 비교 분석하였다. 마지막으로 5장에서 논문의 결론과 향후 과제에 대해 논의 한다.

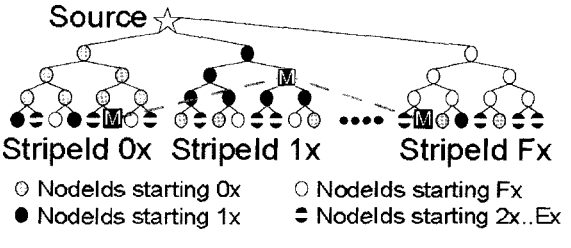
2. SplitStream

SplitStream[7]은 Pastry[9]와 Scribe[8]을 기반으로 하는 대표적인 스트리밍 시스템이다. SplitStream은 피어-투-피어 환경에서 미디어 전송과 파일 분배 모두에 사용될 수 있다.

2.1 서로 다른 내부노드를 갖는 다중 트리 구성(Interior-Node Disjoint Trees)

SplitStream은 보내고자 하는 데이터를 k개로 나누고, 각 세그먼트를 스트라이프(stripe)라 한다. SplitStream은 이 스트라이프 개수만큼의 서로 다른 내부 노드를 갖는 멀티캐스트 트리를 생성하고, 각 스트라이프를 해당 트리를 통해 전달한다. 하나의 스트라이프는 스트라이프 식별자(stripe id)로 구분한다. (그림 1)는 SplitStream이 데이터를 16개의 스트라이프로 나누어 다중 멀티캐스트 트리를 구성한 예이다.

SplitStream 노드 중 0으로 시작하는 노드 식별자(node id)를 가진 노드들은 첫 번째 스트라이프 트리에 소속된다. 같은 방법으로 다른 노드들도 각자가 가진 노드 식별자의 접두사(prefix)에 따라서 하나의 스트라이프에 속하게 되고, 노드 식별자의 첫 번째 자리와 일치하는 스트라이프 식별자를 가진 스트라이프나 트리를 그 노드의 으뜸 스트라이프(prime stripe)라고 한다. 각 노드는 으뜸 스트라이프 트리에서는 내부 노드(interior node)로서 다른 노드들에게 데이터를 전달하지만, 그 외의 스트라이프 트리에서는 단순히 다른 노드들에게서 데이터를 받기만 하는 단말 노드로 행동한다. 즉, (그림 1)에서 노드 M의 경우 스트라이프 식별자 1x에서는 내부 노드이지만, 0x와 Fx 등 다른 스트라이프에서는 단말 노드가 된다. 따라서 이렇게 구축되는 트리들은 서로 다른 내부노드를 갖게 된다.



(그림 1) SplitStream의 다중 멀티캐스트 트리 구성[7]

2.2 희생자 선택 방법

SplitStream은 부하를 분산시키기 위해서 자식연결의 개수(out-degree)를 제한한다. 만일 자식연결 개수를 제한하지 않는다면 자식 노드의 개수가 그 노드의 용량을 넘어서 병목현상을 유발할 수 있다. 따라서 SplitStream에서는 자식연결 개수가 한계에 다다른 노드가 다른 노드로부터 연결 요청을 받으면 다음과 같이 진행된다.

일단 연결을 요청한 노드를 자식노드로 받아들이고, 새로운 자식노드 집합 들을 평가하여 추방할 자식노드를 선택한다. 추방할 자식 노드의 선택 방법은 다음과 같다.

먼저 스트라이프 식별자와 노드 식별자가 같은 접두어(prefix)를 공유하지 않는 자식노드를 선택하여 추방한다. 만일 연결을 요청했던 노드가 선택된다면 바로 추방되고, 아니면 해당하는 자식 노드 중 임의의 노드가 선택되어 추방당한다.

만일 그런 노드가 하나도 없다면, 자식 노드 중 해당 스트라이프와 가장 짧은 접두어 공유(prefix match)를 가진 노드를 선택한다. 만일 요청을 희망한 노드가 여기에 해당된다면 추방하고, 아니면 해당하는 자식 노드들 중 임의의 노드를 선택해 추방한다. 이때 추방되도록 선택된 노드는 해당 스트라이프 트리에서 고아가 된다.

고아가 된 노드는 다시 자신의 부모 노드를 찾게 되는데, 다음의 두 단계를 거치게 된다. 먼저 기존의 스트라이프 트리에서의 형제 노드(sibling node)들 중 스트라이프 식별자와 같은 접두어 공유를 갖는 임의의 형제 노드에게 연결 요청을 보낸다. 연결 요청을 받은 노드는 앞에서와 같은 조건과 절차에 따라 이 연결 요청을 받아들이거나 거절하게 된다. 이 과정은 트리를 따라 내려가면서 고아 노드가 새로운 부모 노드를 찾거나 스트라이프 식별자와 접두어를 공유하는 노드가 없을 때까지 재귀적으로 진행된다. 만일 고아 노드가 끝까지 부모 노드를 찾지 못한다면, 두 번째 방법으로 여분 능력 그룹을 이용하게 된다.

2.3 여분 능력 그룹

여분 능력 그룹은 구축된 스트라이프 멀티캐스트 트리들 중에서 현재 자신의 전달 능력보다 적은 개수의 자식 노드를 갖고 있는 노드들로 이루어진 그룹이다. 따라서 여분 능력 그룹은 새로운 자식노드를 받아들일 수 있는 노드들로만 구성되고, 여분 능력이 없어진 노드들은 이 그룹을 탈퇴한다.

부모 노드를 찾지 못한 고아 노드가 여분 능력 그룹에 특정 스트라이프에 대한 합류 요청을 보내면, 이 요청 메시지는 애니캐스트(anycast)를 통해 여분 능력 그룹에 전달된다. 이때 여분 능력 그룹에 속한 노드들 중 합류를 요청한 노드와 실제적인 네트워크(physical network) 상에서 가장 가까운 노드가 이 요청을 받게 되므로, 이 요청을 받은 노드를 시작으로 DFS(depth-first search) 방식으로 요청하는 스트라이프 트리에 속한 노드를 찾는다.

2.4 문제점

이 SplitStream방법에서는 각 노드들의 능력의 차이를 고려하지 않기 때문에, 서로 네트워크 자원의 크기가 다른 노드들이 혼재하는 상황에서 네트워크 자원이 작은 노드들이 트리의 상층부에 배치될 수 있다. 트리의 상층부에 위치한 노드가 병목현상을 일으킨다면 그 노드로부터 시작하는 서브트리의 모든 노드가 데이터를 받을 수 없게 되므로 전체 시스템의 내구성이 크게 나빠지게 된다. 따라서 본 논문에서는 그러한 문제점들을 해결하기 위해서 노드의 능력을 고려하는 트리 생성 기법을 제안하고자 한다.

3. 내구적 능력고려 멀티캐스트 트리 생성 기법

본장에서는 내구적 능력고려 멀티캐스트 트리(Resilient Capacity-Aware multicast Tree, 이하 R-CAT) 생성 기법에 대해 설명하고자 한다. 먼저 R-CAT 생성 기법의 목표를 살펴보고, R-CAT 생성 기법을 적용할 멀티캐스트 트리 모델을 제시한 뒤, 내구적 능력고려 멀티캐스트 트리 생성 기법에 대해 설명하도록 하겠다.

3.1 설계 목표

본 논문에서 제시하는 멀티캐스트 트리 생성 기법은 크게 세 가지의 설계 목표를 가지고 있다.

첫째로, R-CAT 생성 알고리즘을 적용하는데 소요되는 비용을 최소화 하여야 한다. 현재 상황에서의 최적트리(optimal tree)를 생성하는 것이 가장 최선의 방법이겠으나, 최적트리를 생성하는 것은 피어-투-피어 환경에서 현실적으로 힘든 일이다. 피어-투-피어 기반의 스트리밍 환경에서는 일반 서버-클라이언트 시스템보다 많은 사용자들과 많은 리소스가 존재한다. 서버의 역할을 하는 노드의 합류와 떠남으로 인해 멤버십 정보(membership information)가 지속적으로 바뀌기 때문에, 최신 정보를 유지하기 위해서는 많은 오버헤드가 발생하게 된다. 따라서 R-CAT 생성 알고리즘은 추가적인 오버헤드를 최소화하여 현실적으로 적용할 수 있도록 해야 한다.

둘째로, R-CAT 생성 기법은 노드의 합류, 떠남, 그리고 병목현상 발생 등의 환경변화로 인해 발생하는 패킷 손실을 최소화해야 한다. 스트리밍 환경에서 지속적이지 못한 서비스는 사용자를 떠나게 하는 중요한 요인이 된다. 패킷 손실을 최소화 한다면, 이는 스트리밍 그룹의 안정화를 가져올 수 있다. 이를 위해서는 안정적인 노드가 멀티캐스트 트리의 상부에 위치하도록 해서 트리의 상부에서 불안정한 상황을 최소화해야 한다.

셋째로, R-CAT 생성 기법은 종단 간 지연시간(End-to-End delay time)을 최소화해야 한다. 종단 간 지연시간이 최소화 된다면, 이 또한 스트리밍 그룹의 안정화를 가져올 수 있다. 따라서 R-CAT 생성 기법은 노드가 합류할 때 부모노드와 자식노드간의 지연시간을 최소화 시켜야 한다.

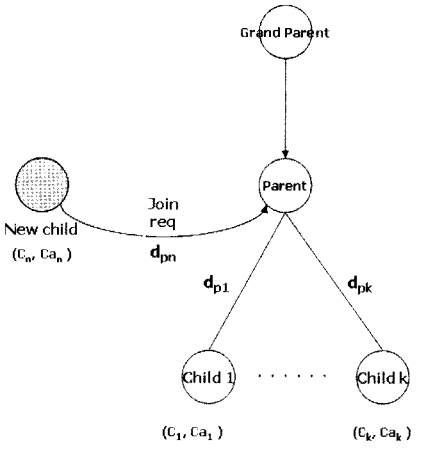
3.2 멀티캐스트 트리 모델

본 논문에서 피어-투-피어 네트워크에 존재하는 노드들의 능력 분포는, 피어-투-피어 시스템의 일종인 Gnutella를 통해 노드의 능력 분포를 분석한 결과인 <표 1>와 같다고 가정한다[2, 11, 12]. 이 표에서 능력이 작은 노드가 상당히 많이 존재한다는 것을 알 수 있다.

노드 i는 최대 능력치(max capacity) C_i와 사용가능 능력치(available capacity) C_{ai}를 가지며 항상 $C_i >= C_{ai}$를 만족한다. 최대 능력치는 Gnutella-like 노드 능력 분포에 따라 노

〈표 1〉 Gnutella-like 노드 능력 분포

Capacity level	Percentage of nodes
1x	20%
10x	45%
100x	30%
1000x	4.9%
10000x	0.1%



(그림 2) 멀티캐스트 트리 모델

드에 할당된 능력치를 의미하고, 사용가능 능력치는 멀티캐스트에서 업로드(upload)에 사용될 수 있는 능력치다. 각 노드가 데이터를 받을 수 있는 다운로드(download) 능력치는 충분하다고 가정한다. 따라서 이 멀티캐스트 트리 모델에서 사용가능 능력치는 다른 노드에게 데이터를 전달 할 수 있는 능력치로 정의한다.

멀티캐스트 트리는 (그림 2)과 같이 부모(parent), 조부모(grand parent), 자식(child)으로 구성된다. 새로운 자식노드가 멀티캐스트에 참여하기 위해서 이미 조직된 부모, 조부모, 자식 노드들로 구성된 트리에 합류요청(join request)을 한다. 이때 부모노드와 자식노드 간에는 단일한 양방향 연결(undirected edge)이 존재하는데, 이 연결들에는 각각 실제 네트워크(physical network) 상에서 부모노드 i와 자식노드 j 간의 종단 간 지연시간 d_{ij}가 존재한다.

특정 데이터에 대한 멀티캐스트 그룹은 그 스트리밍 서비스에 참여하고자 하는 노드들의 집합이다. (그림 2)의 부모노드와 자식노드는 같은 그룹 안에 포함되어 있으며, 그들은 서로 상대방의 생존여부를 알고 라우팅 테이블을 갱신하기 위하여 주기적으로 컨트롤 메시지를 주고받는다.

3.3 알고리즘

3.3.1 개요

앞서 제시한 세 가지 설계 목표를 달성하기 위한 R-CAT 기법은 크게 다음의 세 가지 알고리즘으로 구성된다.

- 지역정보 갱신 알고리즘
- 등반을 통한 부모 선택 알고리즘
- 적합성을 고려한 희생자 선택 알고리즘

R-CAT 생성 기법은 노드의 능력치를 고려해서 트리를 생성한다. 일반적으로 능력치가 높은 노드는 그렇지 않은 노드에 비해서 통신 기능을 수행할 때 더 안정적이므로, 상대적으로 능력치가 큰 노드들을 멀티캐스트 트리의 상층부에 배치해서 전체트리를 좀 더 안정적으로 만들고자 한다.

노드의 지역정보 갱신 알고리즘은 각 노드가 자신의 자식노드들의 정보를 항상 최신으로 유지하기 위해서 사용된다. 각 노드들이 자신의 상태정보를 다른 노드에게 전달할 때는 추가적인 오버헤드를 줄이기 위해서 주기적으로 주고받는 컨트롤 메시지에 피기-배킹(piggy-backing)을 통해서 전달한다. 이 지역정보를 기반으로 어느 부모를 선택할지, 또 어떤 노드를 자식으로 선택할지 결정하게 된다.

등반을 통한 부모선택 알고리즘은 새로운 노드가 멀티캐스트 그룹에 참여하기 위해서 합류요청을 할 때 사용된다. 새로운 노드가 이미 멀티캐스트 그룹에 참여해있는 부모노드 후보에게 합류메시지를 보내면, 처음 합류요청을 받은 부모노드 후보는 새로운 노드의 능력과 자신의 능력을 비교해서 자신보다 더 능력이 크다고 판단하면 자신의 부모노드에게 합류메시지를 전달하고, 그렇지 않은 경우 자신의 자식으로 추가를 시도한다. 이런 등반을 통한 부모선택 알고리즘을 통해 멀티캐스트 트리에서 상부구조의 안정화를 가져올 수 있다.

적합성을 고려한 희생자 선택 알고리즘은 부모노드가 새로운 자식을 받아들이고 자신의 수용능력을 살펴봤을 때 이미 수용가능 능력만큼의 자식노드를 수용하고 있어서 더 이상 자식노드를 수용할 수 없을 경우 가장 바람직한 선택을 하기 위해서 사용된다. 이 알고리즘을 통해서 새로운 자식노드의 합류요청을 다른 자식노드에게 전달하거나, 기존의 자식 집합에서 희생자를 선택하고 새로운 자식을 받아들임으로써 가장 최적의 자식노드를 유지하고 멀티캐스트 트리의 내구성을 높일 수 있다.

3.3.2 지역정보 갱신 알고리즘

R-CAT 생성 기법의 지역정보 갱신 알고리즘은 한 노드가 자신과 연결된 주변 노드들의 현재 상태를 인지하고, 그에 따라 정책 결정을 내리는 밑바탕이 된다. 따라서 효율적인 지역정보 갱신 알고리즘은 멀티캐스트에 참여한 모든 노드들이 자신의 주변상황을 보다 정확하게 파악할 수 있게 해주며, 이를 토대로 내리는 정책결정들은 성능을 더욱 향상시킨다.

주변상황 정보를 실시간으로 파악하기 위해서는 지역정보 갱신 알고리즘을 매우 자주 사용해야 한다. 이는 노드의 합류와 떠남이 빈번한 동적인 상황에서 주변상황에 대한 정확한 정보를 제공해주지만, 그에 따른 추가적인 오버헤드가 발생하게 된다. 그러나 앞에서도 언급했듯이 스트리밍 시스템을 확장성 있게 하기 위해서는 멤버십 관리 오버헤드를 최소화해야 한다. 따라서 지역정보 갱신 알고리즘의 오버헤드를 최소화하기 위해서 주기적인 그룹 메시지에 피기-배킹(piggy-backing)을 사용해서 노드의 정보를 교환하도록 설계했다.

본 논문에서 제안하는 지역정보 갱신 알고리즘은 크게 새로운 노드의 합류요청 상황과 일반적인 생사확인 메시지 교환 상황으로 나뉘며, 각 상황에 대해 다음과 같이 동작한다.

첫째로, 새로운 노드가 합류요청을 하는 상황에서 그 노드는 합류요청 메시지에 자신의 상태정보를 피기-배킹하여 부모노드 후보에게 보낸다. 이 부모노드 후보가 새로운 노드를 자식노드로 받아들일 경우는 합류승인 메시지에 부모노드의

정보를 포함시켜 보냄으로써 서로의 정보를 교환한다.

둘째로, 부모노드와 자식노드들 간의 생사확인 메시지를 교환하는 상황에서, 자식노드는 생사확인 메시지에 자신의 정보를 담아 부모노드에게 보내고, 부모노드 역시 자신의 정보를 보내어 서로의 지역정보를 갱신한다. 생사확인 메시지 교환의 주기를 π라고 한다면 자식노드는 매 π마다 생사확인 메시지를 부모에게 보낸다. 반면 부모노드는 가장 최근 자식노드로부터 생사확인 메시지를 받은 시간과 현재 시간과의 차를 구한다.

$$\Delta t = T_{current\ tick} - T_{recent\ update\ tick} \qquad <식\ 1>$$

<식 1>에서 만약 Δt가 π보다 작다면 기다리고, 그렇지 않다면 부모노드는 자식노드에게 생사확인 메시지를 보낸다.

노드는 지역정보 갱신 알고리즘을 통해 노드의 능력치(capacity), 생존성(aliveness), 적합성(goodness), 자신을 루트노드로 하는 부분트리(sub-tree)의 총 자손의 수(the number of sub-children), 이렇게 네 가지 정보를 교환한다. 각각의 정의는 <식2>, <식 3>, <식 4>, 그리고 (그림 3)에서 나타내었다.

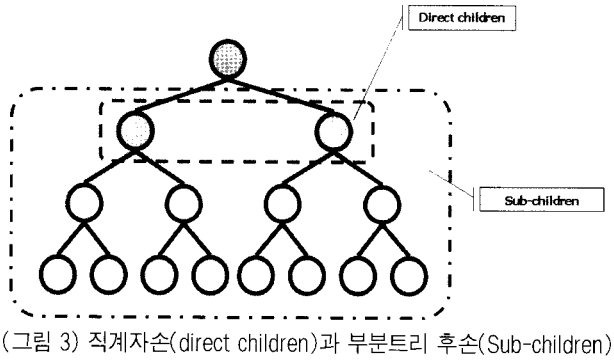
C_i : 노드 i의 최대 능력치

$A_i = 1 - \frac{1}{C_i + k}$: 노드 i의 생존성 (k : scaling factor) <식 2>

$S_i = \sum_{j \subset h_i} S_j + 1$: 노드 i를 루트노드로 하는 부분트리의 총 노드 수 <식 3>

h_i : 노드 i의 직계 자손(direct children) 집합

$G_i = A_i \times S_i$: 노드 i의 적합성 <식 4>



(그림 3) 직계자손(direct children)과 부분트리 후손(Sub-children)

노드 i의 능력치 Ci는 수용할 수 있는 자식노드의 수를 나타낸다. 어떤 노드도 자신의 능력치를 초과해서 자식노드를 수용할 수 없다. 즉 항상 $|h_i| <= C_i$가 성립한다.

노드 i의 생존성 A_i는 능력치가 큰 노드가 안정적이라는 연구결과를 이용해서 노드i가 병목현상 등을 발생시키지 않고 정상적으로 동작할 수 있는 수치를 나타낸다. 여기서 스케일링 인자(scaling factor) k는 노드의 능력치 Ci를 얼마나 생존성에 반영시킬지를 결정하는 상수이다. 예를 들어 k = 1로 가정하면, Ci 가 굉장히 크다면 A_i는 1에 수렴할 것이고, 반대로 C_i = 0이라면 A_i는 0이 된다.

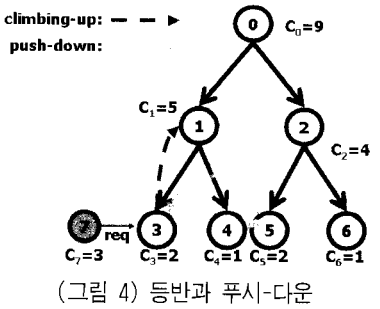
노드의 적합성 G_i는 노드 i의 생존성과 부분트리 후손을 모두 고려한다. 노드의 적합성은 생존성이 높거나 부분트리 후손이 많을수록 높아진다. 이 인자는 부모노드의 자식노드 선택 알고리즘에 사용된다.

피기-배킹을 사용하고 추가적으로 메시지를 주고받지 않으므로 지역정보 갱신 알고리즘으로 인한 추가적 컨트롤 메시지 오버헤드는 없다고 할 수 있다. 피기-배킹 때 생사확인 메시지의 크기가 커질 수 있으나, 전송해야하는 데이터의 양이 극히 작으므로 갱신을 위한 오버헤드는 없다고 할 수 있다.

3.3.3 등반(climbing)을 통한 부모 선택 알고리즘

일반적인 P2P 기반 스트리밍 환경에서 새로운 자식노드가 멀티캐스트 그룹에 합류하기 위해서는 이미 멀티캐스트 그룹에 합류해 있는 노드들 중에서 자신의 부모를 선택해서 허락을 받아야 한다. 자식노드의 멀티캐스트 그룹 합류과정은 자식노드가 그룹 내의 임의의 한 노드를 부모노드 후보로 선택하고 합류요청을 하면 선택된 노드는 새로운 자식을 수용할 수 있는 여분의 능력이 남아있는지 확인한 후, 여분의 능력이 있다면 새로운 노드는 이 후보노드의 자식으로 수용된다. 만약 그렇지 않다면 새로운 노드가 보낸 합류요청 메시지는 부모노드 후보의 부모, 혹은 자식노드에게 전달된다. 이 노드는 새로운 부모노드 후보가 되고 위의 과정을 반복한다.

본 논문이 제시한 알고리즘은 트리의 안정성을 높이기 위해서 가장 안정적인 노드를 트리의 최상층에 배치한다. 등반을 통한 부모 선택 알고리즘은 (그림 4)와 같이 동작한다. 새로운 노드 7은 부모 후보노드 3에게 합류요청 메시지를 보낸다. 노드 3은 자신의 능력치와 노드 7의 능력치를 비교한 후, 노드 7의 능력이 자신보다 크기 때문에 합류요청 메시지를 자신의 부모인 노드 1에게 전달한다. 이를 등반(climbing)이라고 한다. 노드 1은 노드 7보다 능력치가 높기 때문에 이 메시지를 등반시키지 않고 노드 7을 수용하려고 시도한다. 노드 7을 수용할 여분의 능력이 있다면 자신의 자식으로 등록하고, 만약 그렇지 않다면 기존 자식들을 대상으로 3.3.4 절에서 설명할 희생자 알고리즘을 통해서 희생자를 선택한다. 이때 선택된 희생자가 있다면 희생자를 방출하고 노드 7을 받아들이고, 만약 없다면 합류요청 메시지를 자신의 자식노드인 3번과 4번 노드에게 전달하는데, 이를 푸시다운(push-down)이라 한다. 합류요청 메시지는 그 노드가 받아들여질 수 있는 부모노드를 만날 때까지 푸시-다운된다.



(그림 4) 등반과 푸시-다운

```
i = prospective child's node id
p = parent's node id
msg = subscription request

if( Cp > Ci ) {
    if( |Hp| < Cp )
        addChild( the prospective child );
    else{
        victim = find_victim();
        if ( victim != null ){
            drop victim
        }
        else{
            add me to msg's visited list
            forward msg to one of my children
        }
    }
}
else {
    forward this join msg to my parent;
}
```

(그림 5) 등반을 통한 부모 선택 알고리즘

합류요청 메시지의 등반은 N을 전체 노드의 개수라 했을 때, 최악의 경우 O(logN)번 일어날 수 있다. 즉, 트리의 제일 밑바닥에서 루트까지 등반을 하는 것을 말한다. 이는 노드의 능력 분포를 고려했을 때, 최악의 경우는 그리 많이 일어나지 않을 것임을 알 수 있다. (그림 5)에서 등반을 통한 부모 선택 알고리듬을 수도코드로 나타내었다.

3.3.4 적합성을 고려한 희생자 선택 알고리즘

부모 노드는 새로운 노드가 합류요청을 했을 때, 합류요청 메시지를 등반시키지 않기로 결정했다면 자신의 여분의 수용능력을 확인한다. 부모 노드가 여분의 수용능력이 없을 경우, 새로운 자식노드와 기존의 자식노드 집합의 합집합에서 방출할 희생자(victim)를 선택해야 한다. 만약 선택된 희생자가 새로운 자식노드라면 자신의 자식노드들 중 하나에게 합류 요청 메시지를 전달한다. 그렇지 않을 경우 방출된 노드는 다시 합류요청 과정을 통해 멀티캐스트 그룹에 합류해야 한다.

앞서 언급한 본 알고리즘의 설계 목적을 만족시키기 위해 희생자 선택 알고리즘은 자식노드의 적합성(Goodness)과 부모 노드와의 종단간 지연시간(delay between parent and a child)을 고려한다. 내구성(Resilience)은 <식 5>로 표현될 수 있으며, 그룹변화 상황에서 확률적으로 패킷 손실을 줄일 수 있는 능력으로 정의한다. 희생자 선택 알고리즘은 내구성을 최대화 할 수 있는 방향으로 희생자를 선택한다.

Hi : 노드 i가 선택 가능한 모든 자식 집합의 집합

$$\forall h_i \in H_i,\ |h_i| \le C_i \qquad \text{<식 5>}$$

$$R_i = \max\left\{ \sum_{j \subset h_i} (G_j - d_{ij}) \right\}$$

$$h_i \in H_i$$

$$h_i^* = \arg max \left\{ \sum_{j \in h_i,\ h_i \in H_i} (G_j - d_{ij}) \right\} \qquad \text{<식 6>}$$

<식 5>와 <식 6>에서 희생자 선택 시 부모노드는 자식노드의 적합성과 부모노드와 자식노드간의 지연시간을 고려해서, 적합성은 최대화시키고 지연시간은 최소화시키는 자식노드 집합 h_i^*를 선택한다. 이를 현재 상태에서 내구성을 극대화시키는 최적 자식노드 집합(optimal children set)이라고 한다. 자식노드의 지연시간을 최소화 한다는 것은 크게 봤을 때, 전체 트리의 루트노드에서 말단노드까지의 지연시간을 최소화 한다는 것을 의미한다. 또한 적합성을 최대화 한다는 것은, 안정적이고 부분트리의 자식노드의 개수가 많은 노드를 먼저 희생자에서 제외함으로써 트리 전체의 패킷 손실을 줄인다. 즉, 등반을 통해서 우수 노드를 트리의 상층부에 배치하고 희생자 선택을 통해서 능력이 떨어지는 노드를 트리의 하부로 밀어낸다. 여기서 적합성과 지연시간의 중요도를 적절히 조절하는 트레이드-오프 파라미터(trade-off parameter) λ를 도입하면 위의 <식 5>와 <식 6>는 다음과 같이 다시 정의할 수 있다.

$$R_i = \max\left\{ \sum_{j \in h_i} (G_j - \lambda d_{ij}) \right\} \qquad \text{<식 7>}$$

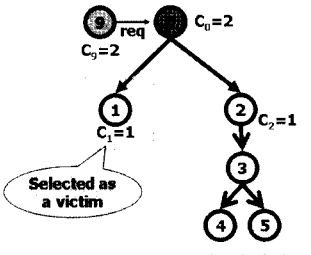
$$h_i \in H_i$$

$$h_i^* = \arg max \left\{ \sum_{j \in h_i,\ h_i \in H_i} (G_j - \lambda d_{ij}) \right\} \qquad \text{<식 8>}$$

희생자 선택 알고리즘을 통해서 최적 자식 집합 h_i^*를 구하면 나머지 이 집합에 속하지 않은 나머지 직계자손은 자연적으로 희생자로 선택된다. 희생자 선택 알고리즘을 적용하기 이전의 자식 집합을 h_i'이라 하고, 새로운 자식 노드를 j라 하면 희생자 노드는 다음과 같다.

$$victim = \{h_i' \cup j\} - h_i^* \qquad \text{<식 9>}$$

예를 들면, (그림 6)에서 보는 것과 같이 노드 9번이 노드 0에게 합류를 요청하면, 노드 0은 자신의 능력치가 새로운 노드에 비해서 작지 않고, 여분의 자식 수용능력이 없기 때문에 희생자 선택을 한다. 앞서 설명한 <식 7>, <식 8>, <식 9>를 통해서 어떤 노드를 방출할지 결정한다. 여기서 비교 대상은 새로운 노드 9와 기존 자식인 노드 1, 2 이다. 각각의 적합성(Goodness)은 0.67, 0.5, 그리고 2.0이라고 하고, 노드 9, 1, 2와 노드 0 사이의 지연시간은 모두 같다고 가정하면 λ와 상관없이, 희생자 알고리즘은 <식 7>, <식 8>, <식 9>를 통해서 노드 1을 희생자로 선택해서 방출하고 노드 9를 받아들인다. 희생자로 선택된 노드 1은 자신의 형제노드(sibling)에게 합류요청을 함으로써 새로운 부모를 찾는다. 적합성을 고려한 희생자 선택 알고리듬의 수도코드는 (그림 7)과 같다.



(그림 6) 적합성을 고려한 희생자 선택

```
if( can afford to add new child )
    add new child
else{
    victim = find_victim()
    if( victim == newChild )
        forward subscription message to one of my children
    else{
        eject the victim
        add new child
    }
}

find_victim(){
    p = parent
    k = newChild
    children = current children set
    iteration = 1 + size of current children set
    victim = newChild
    victim_factor = Gk - λ*dkp

    for( i = 0 ; i < iteration ; i++ ){
        tmp_factor = Gi - λ*dip
        if( tmp_factor < victim_factor ){
            victim = children[i]
            victim_factor = tmp_factor
        }
    }
    return victim
}
```

(그림 7) 적합성을 고려한 희생자 선택 알고리즘

4. 성능 평가

본 장에서는 앞서 제안한 R-CAT 생성 기법의 성능을 측정했다. 이를 위해서 Pastry 시뮬레이터로 잘 알려진 FreePastry [3]를 확장 개발하여 사용했다.

앞서 소개한 대표적인 피어-투-피어 기반의 스트리밍 시스템인 SplitStream과 거기에 R-CAT 기법을 추가한 경우의 성능과 내구성을 비교 측정했다. 제안한 알고리즘의 효율성을 측정하기 위해서 컨트롤 메시지 개수, 합류에 소요된 평균 홉수, 종단 간 평균 지연시간을 측정하였고, 내구성을 측정하기 위해서는 패킷 손실 개수를 측정했다.

4.1 시뮬레이션 모델

시뮬레이션을 위해서 SplitStream과 R-CAT은 하드웨어 사양이 같은 동질의 서버로 측정했다. 해당 서버의 하드웨어는 Pentium4 2.2G CPU, PC133 타입의 512MB 메모리, 그리고 ATA 133 규격의 7200 RPM을 지원하는 하드 디스크로 구성되었다. 운영체제로는 Redhat Linux 9을 사용했으며, 이 환경에서 Java 1.4.2, FreePastry 시뮬레이터 1.3.2 버전 하에서 시뮬레이션 했다.

피어-투-피어 기반의 스트리밍 시스템을 시뮬레이션하기 위해서 [1]에서 언급한 스트리밍 클라이언트의 노드분포를 적용하여 <표 2>와 같이 노드 능력치를 할당했다.

노드의 합류, 떠남, 실패, 병목현상의 그룹의 변화로 인해 발생되는 패킷 손실, 오버헤드 등을 측정하기 위해 시뮬레이터에서는 <표 3>과 같이 합류, 떠남, 실패, 병목현상의 빈도를 변화시키면서 실험한다. 이때 모든 노드들은 합류상태로 시작한다. 일반적인 스트리밍 환경에서 그룹의 변화가 가장 클 때 전체 노드 중 초당 약 1%의 노드가 합류하거나 떠난다. 따라서 테스트 케이스 5는 그룹의 변화가 심한 상황, 0번은 그룹에 변화가 없는 상황을 시뮬레이션하기 위해서 사용된다. 실패 확률도 일반적인 Gnutella에서 조사된 수치를 반영했다. 병목현상 확률은 일반적으로 연구된 자료가 없어 실패확률과 같다고 가정했다.

떠남, 합류, 실패, 병목현상 이벤트가 발생할 노드의 선정은 uniform distribution을 사용했다. 실패 노드로 선정된 노드는 자신의 부모나 자식에게 알리지 않고 시스템을 떠난다. 실패 노드의 부모는 지역정보의 마지막 갱신 시간과 현재시간의 차이가 앞서 설정한 π시간보다 큰 경우나 패킷을 보내는 시점에서 알 수 있다. 자식노드도 마찬가지로 π시간이 지나서야 자신의 부모가 떠났다는 것을 알 수 있다.

마지막으로 실험에 사용한 노드의 개수와 시뮬레이션 시간은 <표 4>와 같다.

〈표 2〉 스트리밍 시스템 클라이언트의 노드 분포

Contribution	Percentage (%)
0	56
1	21
2	9.5
3-19	5.6
20	7.9

〈표 3〉 Test cases 파라미터

Test cases	Departure rate	Failure rate	Bottleneck rate	Arrival rate
0	0	0	0	0
1	0.001	0.0005	0.0005	0.0015
2	0.002	0.0006	0.0006	0.0026
3	0.003	0.0007	0.0007	0.0037
4	0.004	0.0008	0.0008	0.0048
5	0.005	0.0009	0.0009	0.0059

〈표 4〉 시뮬레이터 파라미터

# of nodes	10,000
node capacity distribution	Gnutella-like node distribution
event distribution	uniform distribution
# of stripes	16
scaling factor k	1
λ	0.0001
π	1 simulation tick
Simulation duration	1000

• 합류로 인해 발생한 컨트롤 메시지의 개수

합류로 인해 발생한 컨트롤 메시지는 처음 그룹에 합류를 시도하거나 그룹에서 추방되었을 경우에 발생한다. 따라서 컨트롤 메시지의 개수는 해당 그룹이 얼마나 심한 그룹 변화를 겪고, 또 얼마나 효율적으로 변화를 극복했는지를 나타내주는 지표가 될 수 있다.

• 합류에 사용된 평균 홉수

합류에 소요된 홉수는 애니캐스트 메시지가 몇 개의 수신자에게 전달되었느냐에 따라 결정된다. 다시 말하면 홉수는

얼마나 빨리 합류에 성공했는지를 나타낸다. 만일 부모가 떠나서 재합류를 해야 하는 상황이라면 재합류에 성공하는 시점이 빠르면 빠를수록 그 동안 놓쳐버린 패킷의 수가 적을 것이다. 합류에 사용된 평균 홉수는 발생한 모든 메시지가 최종적으로 받아들여지거나 실패했을 때 거쳐 온 노드의 개수와 같다.

• 종단 간 평균 지연시간(Average End-to-End delay)

종단 간 지연시간은 부모노드와 자식 간의 지연시간을 측정했다. 종단 간 지연시간은 멀티캐스트 트리가 안정성만을 추구하고 지연시간을 등한시한다면 원활한 스트리밍 서비스는 힘들 것이다. 따라서 종단 간 평균 지연시간을 통해서 스트리밍 서비스의 질을 비교할 수 있다.

• 패킷 손실 개수(the number of lost packets)

패킷 손실은 멀티캐스트 트리에서 자신의 부모노드에 문제가 생겨서 자신이 패킷을 받지 못하는 상황을 측정했다. 만일 멀티캐스트 트리가 안정적으로 구성이 되었다면 부모노드의 문제로 인해서 영향 받는 부분트리 자식노드의 수가 줄어들 것이고 패킷 손실 개수는 줄어들 것이다.

4.2 성능 측정 및 분석

본 성능 측정에서는 앞서 제시한 측정 기준에 따라 각 알고리즘의 성능을 측정했다. 먼저 그룹의 변화가 없는 상황에서 각 알고리즘의 성능을 측정했다. 그리고 테스트 케이스 0~5에 대하여 총 노드의 개수를 증가시키면서 그룹의 변화가 각 알고리즘에 미치는 영향을 측정했다. 그룹의 변화가 지속적이지 못할 경우 정확한 성능측정이 어려우므로, 본 논문에서는 매 실험을 1000 시뮬레이션 시간(simulation tick) 동안 진행했다.

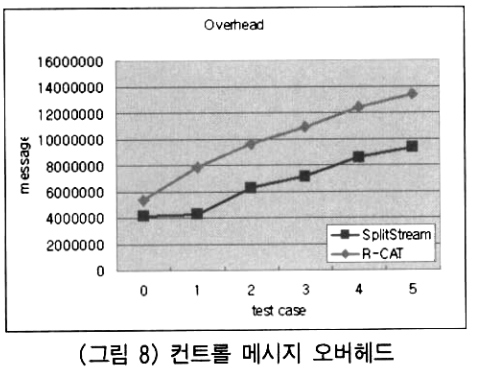
4.2.1 총 발생한 컨트롤 메시지

SplitStream과 R-CAT의 테스트 케이스 0~5까지 시뮬레이션 수행결과는 (그림 8)에서 보는 것처럼 전반적으로 R-CAT가 SplitStream보다 약 40% 정도의 추가적인 오버헤드가 발생했다. 그룹의 변화가 없는 0번 테스트 케이스에서는 R-CAT의 오버헤드가 SplitStream과 거의 차이가 나지 않았으나, 테스트 케이스 1~5번에서는 그룹의 변화가 지속적일 경우 오버헤드의 차이가 40%가량 꾸준히 차이를 보인다.

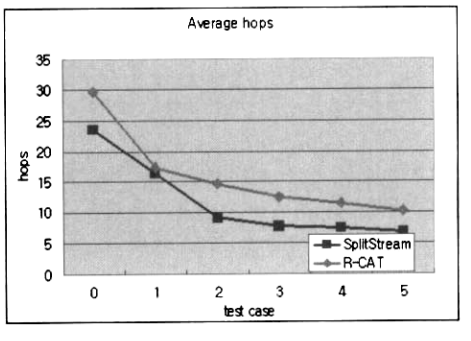
1~5번 테스트 케이스에서 보여주는 R-CAT의 오버헤드는 추가적인 프로토콜로 인해 발생한 오버헤드라 할 수 있는데, 이는 등반을 통해서 트리의 내구성을 높이는데 사용되었다. 등반을 통해 트리의 상층부에 올라간 노드는 그 곳에서 푸시-다운(push down)기법을 통해 합류에 성공할 때까지 등반한 지점부터 트리의 바닥까지 DFS를 이용해 합류를 시도한다. 따라서 등반한 노드가 높이 올라가면 갈수록 잠재적 오버헤드는 커지게 된다. SplitStream의 경우는 등반을 하지 않기 때문에 R-CAT에 비해 적은 오버헤드를 발생시켰다.

4.2.2 합류에 사용된 평균 홉수

평균 합류 홉수는 (그림 9)에서 그 결과를 나타내었는데, 그룹의 변동이 커질수록 작아지는 것을 볼 수 있다. 그룹이 안정되었을 때는 이미 부모노드에 자식 노드들이 가득 차 있어 더 이상 자식노드를 받아들일 수 없는 경우가 많다. 하지만 그룹의 변동이 심한 경우는 떠나는 노드들이 많아지면서 노드가 애니캐스트를 통해 합류를 시도할 때 처음 도착한 노드에 여분의 자식수용 능력이 남아있을 확률이 높기 때문에 합류에 소요된 홉수는 줄었다. 전체적으로 SplitStream이 R-CAT보다 5홉 정도 빨리 합류에 성공하는 것을 볼 수 있는데, 이는 R-CAT의 등반 알고리즘으로 인해서 합류메시지가 트리의 상층부까지 전달되는 과정에서 발생한 비용이다.



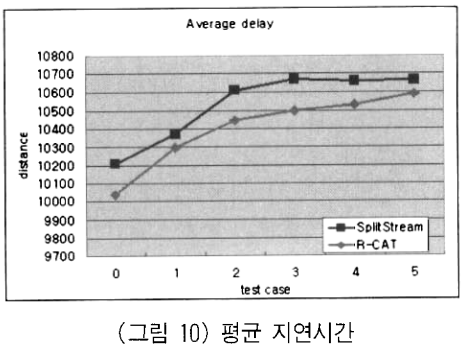
(그림 8) 컨트롤 메시지 오버헤드



(그림 9) 합류에 소요된 평균 홉수

4.2.3 종단간 평균 지연시간

평균지연시간은 패킷 손실율과 더불어 스트리밍 서비스가 얼마나 끊이지 않고 제공될 수 있는가에 큰 영향을 미친다. 평균 지연시간은 (그림 10)에서 볼 수 있는 것과 같이 전체적으로 R-CAT이 SplitStream에 비해서 좋은 결과를 보여주



(그림 10) 평균 지연시간

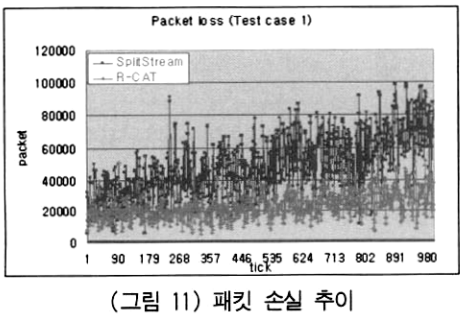
었다. 이는 R-CAT이 오버헤드를 희생하고 얻은 이득 중 하나이다. 노드가 높이 등반을 한다는 것은 그 만큼 많은 노드와 비교할 수 있다는 것을 의미한다. 따라서 작은 집합에서 최적 값을 선택하는 것보다 큰 집합에서 선택하는 것이 확률적으로 좀 더 최적화될 수 있다.

4.2.4 패킷 손실 개수

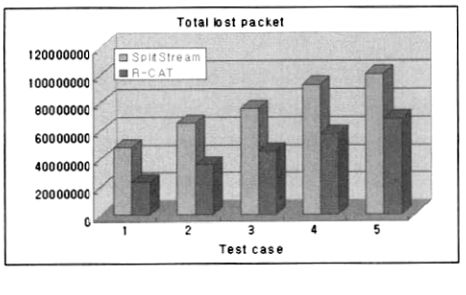
앞서 언급한 것처럼 그룹의 변동은 컨트롤 메시지의 증가뿐만 아니라, 패킷 손실율도 함께 증가시킨다. 따라서 그룹의 변동에 대하여 각 알고리즘의 내구성을 살펴보기 위해서 패킷 손실을 측정했다. (그림 11)은 시뮬레이션이 진행됨에 따라 전 구간에서 R-CAT에서 손실된 패킷의 수가 SplitStream보다 훨씬 적음을 보여준다. (그림 12)는 전체 패킷 손실 개수 역시 R-CAT이 SplitStream에 비해 50% 정도로 적음을 보여준다.

R-CAT이 SplitStream보다 그룹 변화에 대해서 패킷 손실률을 낮출 수 있었던 원인은 R-CAT의 등반 알고리즘과 적합성을 고려한 희생자 선택 알고리즘 때문이다. 적합성을 고려한 희생자 선택 알고리즘에서 선택된 희생자는 적합성이 의미하는 바와 같이 가장 덜 안정적이거나 가장 부분트리 자식의 개수가 적은 노드이므로, 이것은 패킷 손실을 낮추는 결과로 연결된다. 또한 그룹에서 떨어져 나갔던 안정적인 노드는 등반 알고리즘을 통해 트리의 상층부로 올라가므로, 트리는 그룹의 변동에도 안정성을 어느 정도 유지할 수 있었다.

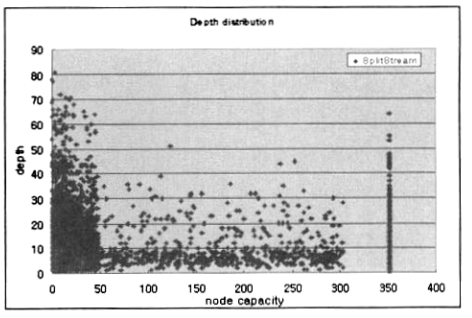
등반 알고리즘과 적합성을 고려한 희생자 선택 알고리즘이 트리의 안정성을 높였다는 것을 (그림 13)과 (그림 14)를 통해서 확인할 수 있다. (그림 13)은 SplitStream의 트리상의 노드분포를 보여주는데, SplitStream은 노드의 능력을 고려하지 않았기 때문에 많은 우수 노드들이 트리의 하단에 위치한 것을 보여준다. 반면 (그림 14)에서 R-CAT은 SplitStream에 비해 우수 노드들이 트리의 상층부에 더 많이 존재하고 전체



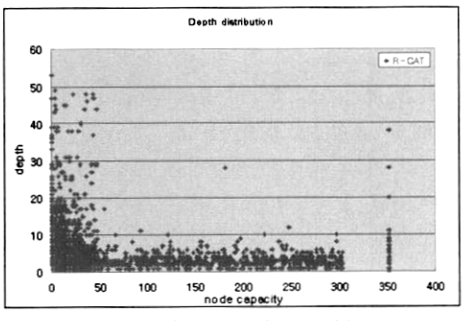
(그림 11) 패킷 손실 추이



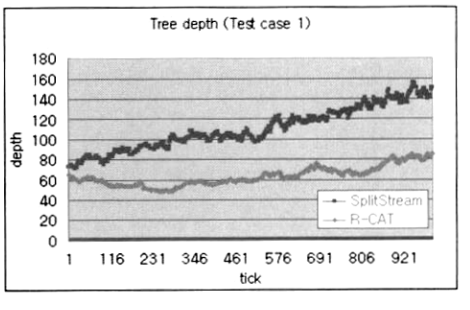
(그림 12) 총 패킷 손실



(그림 13) SplitStream의 노드 분포



(그림 14) R-CAT의 노드 분포



(그림 15) SplitStream과 R-CAT의 depth 비교

적으로 트리의 길이가 짧아졌음을 알 수 있다. 또한 (그림 15)에서 시뮬레이션 기간 동안 SplitStream은 점점 트리의 깊이가 길어지는 현상을 보인 반면, R-CAT은 거의 일정수준의 트리 깊이를 유지하는 것을 볼 수 있다. 이를 통해 R-CAT은 그룹변화 상황에서도 지속적으로 트리의 안정성을 높인다는 것을 알 수 있다.

5. 결론 및 향후 과제

기존의 피어-투-피어 기반의 스트리밍 시스템은 노드의 능력 차이에 대한 고려가 없이 연구가 진행되어 왔다. 따라서 지연시간만을 고려하거나 임의의 노드를 자식, 혹은 부모 노드로 선택했다. 또한 희생자를 선택함에 있어서 그 노드의 능력이나 부분트리 자식의 개수 등을 고려하지 않았으므로, 멀티캐스트 그룹의 노드들이 합류, 떠남, 실패 등을 반복할 때, 트리의 안정성을 떨어뜨리는 결과를 낳았다.

따라서 본 논문에서는 노드의 능력 차이를 고려해서 그룹의 변화에도 안정성을 유지할 수 있는 내구적 멀티캐스트 트리 생

성 알고리즘을 제시하였다. 노드의 정보 수집을 위한 오버헤드를 피기-배킹을 통해 최소화하는 알고리즘을 제시하였다. 또한 노드의 능력을 고려해 합류 시 등반을 통해 우수한 노드를 트리의 상층부에 위치시키고, 희생자 선택 시 노드의 적합성을 고려해서 내구적인 트리를 생성하는 알고리즘(Resilient Capacity-Aware multicast Tree, R-CAT)을 제시하였다.

본 논문에서는 제안한 알고리즘의 성능을 측정하기 위하여 피어-투-피어 기반의 스트리밍 환경에서 합류에 소요된 컨트롤 메시지 오버헤드, 합류에 소요된 평균 홉수, 결과 트리의 종단간 평균 지연시간, 손실된 패킷 수 등을 기존 연구 중 우수한 성능을 보인 SplitStream과 비교 측정하였다. 오버헤드 측면에서 R-CAT은 SplitStream과 정적인 상황에서는 비슷하고 동적은 상황에서 최대 40% 정도 증가함을 보였다. 그러나 종단간 지연시간 측면에서는 모든 테스트 케이스에서 향상된 모습을 보였고, 패킷 손실은 50%로 적었다. 또한 SplitStream의 경우 동적인 네트워크 상황에서 트리가 계속 길어지는데 비해, R-CAT의 경우 일정한 길이를 유지하는 안정적인 트리를 구축함을 볼 수 있다.

하지만 본 논문에서는 피기-배킹만을 의존해 지역정보를 수집함으로 인해 전역 최적화(global optimum)가 아니라 지역적 최적화(local optimum)만을 이루었다. 전역 최적화를 이루기 위해서는 효율적인 멤버십 프로토콜을 통해 지역정보를 보다 정확하고 더 넓은 범위에서 수집할 수 있어야 한다. 따라서 추후 효율적이고 확장성이 높은 멤버십 프로토콜에 대한 연구가 필요하다.

참 고 문 헌

[1] K. Sripanidkulchai, A. Ganjam, B. Maggs, and H. Zhang, "The feasibility of supporting large-scale live streaming applications with dynamic application end-points," SIGCOMM Comput. Commun. Rev. Vol.34, No.4, pp.107-120. Aug., 2004.

[2] S. Saroiu, P. K. Gummadi, and S. D. Gribble, "A measurement study of peer-to-peer file sharing systems," In Proceedings of the Multimedia Computing and Networking(MMCN). San Jose, CA, Jan., 2002.

[3] FreePastry, http://freepastry.rice.edu/

[4] D. A. Tran, K. A. Hua, and T. Do, "ZIGZAG: An Efficient Peer-to-Peer Scheme for Media Streaming," Twenty-Second Annual Joint Conference of the IEEE Computer and Communications Societies, IEEE Vol.2, pp.1283-1292, Apr., 2003.

[5] S. Banerjee, B. Bhattacharjee, and C. Kommareddy, "Scalable application layer multicast," In Proceedings of the 2002 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, SIGCOMM '02. ACM Press, New York, NY, pp.205-217, 2002.

[6] V. N. Padmanabhan, H. J. Wang, P. A. Chou, and K. Sripanidkulchai, "Distributing Streaming Media Content Using Cooperative Networking," ACM NOSSDAV, Miami Beach, FL, USA, May, 2002.

[7] M. Castro, P. Druschel, A-M. Kermarrec, A. Nandi, A. Rowstron and A. Singh, "SplitStream: High-bandwidth multicast in a cooperative environment," SOSP'03, Oct., 2003.

[8] M. Castro, P. Druschel, A-M. Kermarrec and A. Rowstron, "SCRIBE: A large-scale and decentralised application-level multicast infrastructure," IEEE Journal on Selected Areas in Communications(JSAC) (Special issue on Network Support for Multicast Communications), 2002.

[9] A. Rowstron and P. Druschel, "Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems," IFIP/ACM International Conference on Distributed Systems Platforms (Middleware), Heidelberg, Germany, pp.329-350, Nov., 2001.

[10] H. Deshpande, M. Bawa, and H. Garcia-Molina, "Streaming Live Media over a Peer-to-Peer Network," Technical Report, Stanford University, Aug., 2001. http://dbpubs.stanford.edu:8090/pub/2001-31

[11] K. Sripanidkulchai, B. Maggs, and H. Zhang, "An analysis of live streaming workloads on the internet," In Proceedings of the 4th ACM SIGCOMM Conference on Internet Measurement (Taormina, Sicily, Italy, October, 25-27, 2004). IMC '04. ACM Press, New York, NY, pp.41-54, 2004.

[12] Y. Chawathe, S. Ratnasamy, L. Breslau, N. Lanham, and S. Shenker, "Making Gnutella-like P2P Systems Scalable," In Proceedings of the 2003 Conference on Applications, Technologies, Architectures, and Protocols For Computer Communications (Karlsruhe, Germany, August, 25-29, 2003). SIGCOMM '03. ACM Press, New York, NY, pp.407-418, 2003.

김 은 석

e-mail : wannajump@dcclab.sogang.ac.kr
2004년 서강대학교 컴퓨터학과(이학사)
2006년 서강대학교 컴퓨터학과(공학석사)
2006년~현재 LG전자 근무중
관심분야 : 피어투피어 컴퓨팅, 피투피 스트리밍

한 세 영

e-mail : syhan@sogang.ac.kr
1991년 포항공과대학교 수학과(이학사)
2003년 서강대학교 정보통신대학원 (공학석사)
2004년~현재 서강대학교 컴퓨터학과 박사과정
1996년~현재 (주)이엔지 기술본부
관심분야 : 피어투피어 컴퓨팅, 분산처리 시스템

박 성 용

e-mail : parksy@sogang.ac.kr
1987년 서강대학교 컴퓨터학과(공학사)
1994년 미국 Syracuse University(석사)
1998년 미국 Syracuse University(박사)
1998년~1999년 미국 Bell Communication Research 연구원
1999년~현재 서강대학교 컴퓨터학과 부교수
관심분야 : Autonomic Computing, Peer to Peer Computing, High Performance Cluster Computing and System