Journal of The Korea Society of Computer and Information
Vol. 17 No. 5 , May 2012

2012-17-5-3-3

연결 정보를 이용한 P2P 스트리밍 네트워크 구조의 개선

이 상 훈*, 한 치 근*

An Improvement of the P2P Streaming Network Topology Algorithm Using Link Information

Sang-Hoon Lee*, Chi-Geun Han*

요 약

본 논문에서는 P2P스트리밍에서 peer간 연결 정보를 이용하는 방법을 기반으로 topology 최적화를 연구한다. 제안하는 방법은 mesh-network에서 사용된 link와 사용되지 않은 link의 수로 peer의 업로드 용량을 추정하는 방법을 기반으로 한다. 연결된 link의 정보를 사용하는 기존 방법은 peer의 자원 관리 측면에서 메시지 과부하를 줄이는데 효과적이다. 하지만 업로드 대역폭이 고려되지 않는 topology를 구성할 우려가 있다. 또한 서버에 가까운 peer에서 네트워크 오류 발생시 네트워크 전송성능이 저하될 수 있다. 본 논문에서는 기존 방법의 단점을 보완하는 방법을 제안한다. 기존 방법과 제안하는 방법을 시뮬레이션 하고 결과를 비교 분석한다.

▸ Keyword : P2P 스트리밍, 토폴로지 최적화

Abstract

In P2P streaming management, peer's churning and finding efficient topology architecture optimization algorithm that reduces streaming delay is important. This paper studies a topology optimization algorithm based on the P2P streaming using peer's link information. The proposed algorithm is based on the estimation of peer's upload bandwidth using peer's link information on mesh-network. The existing algorithm that uses the information of connected links is efficient to reduce message overload in the point of resource management. But it has a risk of making unreliable topology not considering upload bandwidth. And when some network error occurs in a server-closer-peer, it may make the topology worse. In this paper we propose an algorithm that makes up for the weak point of the existing algorithm. We compare the existing algorithm with the

• 제1저자 : 이상훈 • 책임저자 : 한치근
• 투고일 : 2011. 12. 28, 심사일 : 2012. 01. 14, 게재확정일 : 2012. 02. 13.
* 경희대학교 컴퓨터공학과(Dept. of Computer Engineering, Kyung Hee University)

proposed algorithm using test data and analyze each simulation result.

▸ Keyword :P2P streaming, topology optimization

I. 서 론

스트리밍을 위한 대안 중 전통적인 클라이언트-서버 방식은 스트리밍 서버에 각 클라이언트들이 직접 연결된다. 이 방식은 미디어 스트리밍을 전송 받는 방식 중 다른 방식들에 비해서 안정적이다. 하지만 서비스를 제공하기 위한 peer의 수에 비례하여 대역폭을 늘려야 한다는 단점이 있다. IP 멀티캐스트 방식은 같은 데이터의 중복 전송으로 인한 네트워크 대역폭 낭비가 적기 때문에 효율적인 전송 방법이다. 하지만 서버와 peer가 다른 네트워크 상에 있을 경우 해당 방법을 도입하기 위해서 트랜스미터와 리시버 사이에 존재하는 모든 라우터가 멀티캐스트 라우팅을 지원해야 한다는 단점이 있다.

이에 반해 P2P 방식은 사용자들의 자원을 공유하여 새로운 설비를 따로 설치하지 않아도 확장성이 좋다는 장점을 가진다. 하지만 P2P 방식을 활용하는 데에도 몇 가지 단점이 존재한다.

P2P 방식에서는 각 peer의 자원이 이질적이어서 각각에 대해 모두 서비스 품질을 만족시키기가 어려우며, peer의 편입 및 이탈이 잦아서 topology의 변경이 자주 일어난다[13]. 또한 스트리밍 환경일 경우 지연시간에 대해서 일반적인 P2P 파일 전송 조건보다 더 엄격한 제한을 가지게 된다[1-3].

이 논문에서는 P2P 방식에서 topology를 최적화하는 방안을 제안한다. 각 peer 간에 연결 및 전송 정보를 이용하는 방법[6]을 기초로 이를 개선한다. 메쉬-네트워크에서 각 peer에 연결된 link의 수를 이용하여 업로드 대역폭을 추정하는 알고리즘을 기반으로 각 연결 정보를 저장하는 queue를 추가한다. queue에 누적된 연결 정보는 peer간 연결의 생성 및 해제를 결정할 때 사용되어 topology에서 peer간 연결의 신뢰도 및 안정성을 높인다.

II. 관련 연구

1. P2P 네트워크 topology 종류별 특징

1.1 트리 오버레이

스트리밍을 위한 topology 중 트리는 가장 기본적인 구조로 서버에서 peer에게 전송하는 방식과 peer에서 peer에게 전송하는 방식이 동일하다. 각 peer는 따로 버퍼 맵 교환이나 데이터 요청(PULL)을 보내지 않고, 상위 peer가 스트리밍 데이터를 전송 받으면 PUSH 방식으로 하위 peer에게 전송한다. 제어 메시지가 적어서 topology가 정적인 상황에서는 효율적인 전송이 가능하나 상위 peer가 topology에서 통지 없이 갑작스럽게 이탈하게 될 경우 떠난 peer의 자식 peer들의 서비스 지속성을 저해할 수 있는 단점이 있다[1-3].

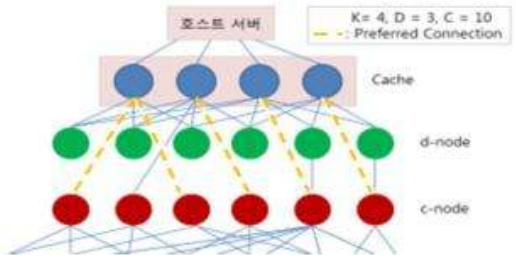
하지만 메시지의 오버헤드가 적고, 구현이 쉬우며 전송 효율이 좋다는 점에서 꾸준히 연구가 지속되고 있다[10].

1.2 메쉬 오버레이

메쉬 오버레이는 peer별 자원 활용도가 좋지 않고, 버퍼 맵 교환이나 데이터 요청 등 추가적인 메시지의 오버헤드가 발생하지만 peer의 갑작스런 이탈이 있을 때에도 다른 peer들의 서비스 지속성에 큰 영향을 미치지 않고 안정적이라는 장점을 가지고 있다[1-3].

각 peer의 연결된 link수만을 조절하여 메시지의 오버헤드를 줄이면서 전체 구조를 제어하는 방법이 제안되기도 하였다[9].

[그림 1]에서 새로 네트워크에 편입되어 D개의 이웃노드를 유지하는 노드와 연결된 이웃노드의 수가 C개가 되어 네트워크에서 분리되는 노드를 각각 d-node와 c-node로 명명한다. 또한 호스트 서버와 바로 연결되어 있는 노드들의 집합을 cache라 한다. 오직 cache에서만 peer의 확장을 처리하며 cache에서 C개의 연결을 유지하게 된 peer는 c-node가 되어 기존의 네트워크에서 독립하게 된다. 이 때 c-node에는 호스트 서버와의 연결점을 유지하기 위해서 preferred connection이라는 이름의 특수한 연결을 추가시킨다. c-node는 오직 한 개의 preferred connection을 가지며 만약 기존에 cache에 포함된 두 개 이상의 peer와 연결을 가지고 있었다면 한 개의 연결만을 남기고 나머지 연결은 제거한다. 이러한 과정들을 반복해서 수행함으로써 특별한 메시지 오버헤드 없이 전체 네트워크에 속해 있는 각각의 peer에 연결된 link의 수를 일정 범위 안에서 유지되도록 하여 전체 peer의 부하를 분산 시킬 수 있게 된다.

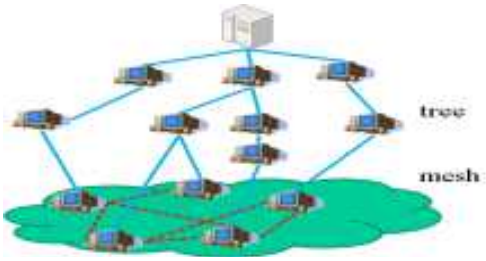


[그림 1] cache를 이용한 메쉬 오버레이 구성 방법
Fig. 1. Mesh overlay construction algorithm using cache

1.3 하이브리드 오버레이

[그림 2]와 같이 peer가 일정 수준 이상 연결을 유지하여 네트워크 신뢰성이 보장되었다고 판단되면 스트리밍 데이터를 제공하는 서버에 가까운 위치에 해당 peer를 트리형태로 연결시키고, 연결이 오래 되지 않은 peer들은 트리의 leaf부분에 메쉬 형태로 연결을 시키는 하이브리드 오버레이를 구성하는 방안이 제시된 바 있다[1-3].

이를 기초로 각 peer들을 유사한 지역별로 묶이도록 서버를 설정하여 메시지 오버 헤드를 줄이고, 업로드 대역폭이 넓은 peer들을 분산되도록 제한하여 자원 편중 현상에 대해서 조절할 수 있도록 하는 방법이 추가로 연구되기도 하였다[5].



[그림 2] 하이브리드 네트워크의 예
Fig. 2. An example of hybrid network

[5]에서는 기본적으로 mTreebone[4]의 특성을 이용한다. ISP 지역성으로 묶인 각 mesh들을 tree 형태로 유지되는 back-bone에 연결하여 topology를 구성한다. 부트스트랩 서버를 두어서 각 메시지 전달간 오버헤드 및 트리의 연결 변경시 중간 과정에서의 오버헤드를 줄이고 있다.

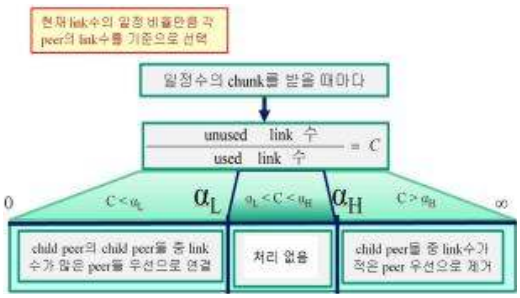
하이브리드 오버레이는 기존 오버레이들과 비교하여 각각의 장점을 차용하였다. 하지만 연결을 어떻게 보장하여 서비스 지속성의 저해를 최소화 할 것인지에 대한 추가적인 연구가 필요하다. 또한 메쉬 기반의 하위 네트워크에서 트리 기반의 상위 네트워크로 편입할 때 peer의 안정성을 평가하는 방법이 실제 P2P 스트리밍 상에서 운용될 때 어느 정도 부합될지 의문점을 가지고 있다. 그에 따라 현재에도 이를 극복하기 위한 다양한 연구들이 진행되고 있다[11-12,14].

2. 적응형 오버레이 알고리즘

적응형 오버레이 알고리즘은 P2P에서 스트리밍 데이터를 전송 받을 때 peer 스스로 각기 다른 peer의 업로드 대역폭을 고려하여 네트워크에 소속된 모든 peer들이 일정한 비율로 자원을 소모함으로써 사용되지 않고 낭비되거나 과부하가 걸리는 peer를 없애는 알고리즘이다[6]. 적응형 오버레이 알고리즘에서 가장 중요한 점은 각 peer의 업로드 대역폭 사용량을 판단하기 위한 메시지 오버헤드 최소화와 각 peer간 업로드 대역폭 사용률 유지이다.

연결 정보를 이용한 적응형 오버레이 알고리즘 연구는[6] 각 peer에서 연결된 link의 수와 사용여부만을 이용하여 업로드 대역폭을 추정하는 방안을 제시했다.

Lobb이 제안한 방법은 [그림 3]과 같이 다음과 같은 방식으로 동작한다[6]. chunk가 일정횟수 이상 도착할 때 연결된 link의 수를 조사하여 각 peer별로 최적의 이웃 peer 연결 상태를 구축한다. link의 수를 조사할 때는 전체가 아닌 이웃 peer의 이웃 리스트까지만을 메시지로 조사하여 속도를 향상시킨다. αL, αH은 확장과 축소를 결정하는 used link의 비율 상수로써 초기부터 일정한 값으로 주어진다.



[그림 3] Lobb이 제안한 적응형 오버레이 알고리즘
Fig. 3. Adaptive overlay algorithm proposed by Lobb.

[6]의 결과를 보면 실제 업로드 대역폭을 기준으로 만들어지는 topology와 거의 근접하면서 메시지 오버헤드를 크게 감소시키고 단순한 형태의 알고리즘으로 구현되어질 수 있다는 점에서 주목할 가치가 있다.

본 논문에서는 Lobb이 제안한 알고리즘에서 peer의 일시적인 네트워크 오류가 발생하였을 때 예상되는 문제점에 대해서 연결 정보를 담아두는 queue를 도입하여 개선된 방안을 제안한다.

III. 본 론

1. 연결정보를 이용한 적응형 오버레이 알고리즘

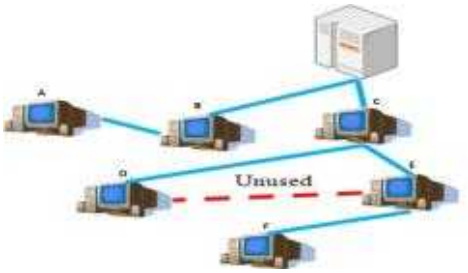
Lobb이 제안한 적응형 오버레이 알고리즘의 동작은 Algorithm 1에 기술되어 있다. 이 알고리즘은 미리 정해진 비율 αL과 αH 을 이용하여 δc개의 chunk가 peer p에 도달할 때마다 unused link 비율 card((p))을 used link card(U(p))와 두 비율의 곱 사이의 값으로 유지한다. card((p))가 αLcard(U(p)) 보다 낮아지면 peer p가 업로드 대역폭이 많이 남았다고 추정한다. 따라서 자신의 child peer 집합 N0(p)와 연결된 peer들의 집합 C(p)에서 현재 link수가 가장 많은 peer n+개를 선택하여 추가로 link를 연결한다.

반대로 card($\overline{U}$(p)이 αHcard(U(p))보다 낮아지면 요구량이 p의 업로드 대역폭을 넘었다고 추정하여 N0(p)중 현재 link수가 가장 적은 peer n-개를 선택하여 연결을 제거한다.

```
Algorithm 1: Lobb's Adaptive Overlay Algorithm
for every δc received chunks do
/* scan and adjust p's neighbourhood */
  if (card( Ū(p) ) < αL card(U(p)) then
/* grow the neighbourhood */
    Select n+ neighbors from (C(p) – N0(p) – {p})
    Add the n+ chosen neighbours to N0(p))
  end
  else if (card( Ū(p) ) > αH card(U(p)) then
/* Shrink the neighbourhood */
    Select n- neighbors from Ū(p) for culling
    Cull the n- neighbours from N0(p)
  end
  else
/* p's neighbourhood does not change */
  end
end
```

이 알고리즘은 스트리밍에서 주어진 연산만을 수행할 경우 업로드 대역폭과 무관한 형태로 네트워크 topology가 잘못 구성될 가능성을 가지고 있다. [그림 4]처럼 서버와 인접한 peer C와 인접하지 않은 peer E 양쪽에게 전송을 받는 child peer D가 있다고 가정한다. D는 이미 더 빠른 peer에게 전송을 받고 있기 때문에 인접하지 않은 peer E의 업로드 대역폭 여부와는 상관없이 D와 E간의 연결은 언제나 사용되지 않게 된다.

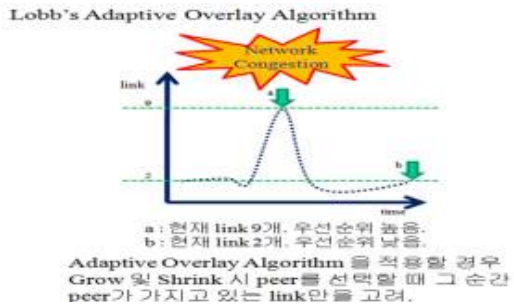
이러한 사용되지 않는 연결은 그 peer가 업로드 대역폭에 있어서 포화상태에 도달했다고 잘못 추정하여 네트워크 topology를 잘못된 방향으로 구성할 수 있다. 그림에서 점선으로 표시된 연결은 unused-link를 의미한다.



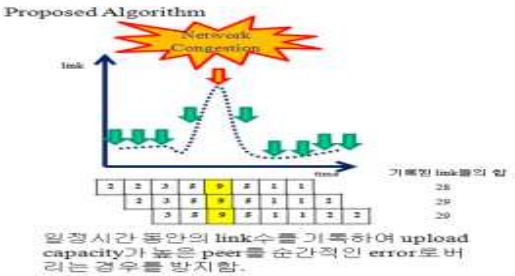
[그림 4] 스트리밍에서의 unused link 문제
Fig. 4. Problem of unused-link in streaming

제안하는 방법에서는 기존의 방법처럼 매 순간의 link수만을 이용하지 않는다. 대신 각각의 peer마다 queue를 두어서 일정한 시간동안 사용되었던 link 수를 저장하도록 한다. 이 때 각 peer에서 사용되었던 link들의 값을 기여도 라고 정의한다.

[그림 5]는 기존 논문의 네트워크 오류 상황을 표현하고 [그림 6]은 본 논문에서 제시하는 기여도의 사용 예를 나타내고 있다. 그림과 같이 기여도를 이용하면 순간적으로 발생한 네트워크 오류 때문에 높은 업로드 대역폭을 가지고 있는 peer를 저평가하여 연결을 끊는 상황을 회피할 수 있다.



[그림 5] 기존 알고리즘의 네트워크 오류 상황
Fig. 5. Network error situation in existing algorithm



[그림 6] 제안하는 알고리즘의 네트워크 오류 상황
Fig. 6. Network error situation in proposed algorithm

이 queue에 저장된 기여도들의 합을 이용하여 peer의 연결을 조절함으로써 일정 기여도 이상인 peer들을 서버에 가깝도록 배치하고, 기여도가 낮은 peer들은 하단의 topology에 묶이도록 하는 것이 가능해진다.

또한 오랜 시간 네트워크에서 전송을 하여 신뢰도가 있는 peer는 상대적으로 높은 기여도를 가진다. 이를 이용하여 신뢰도가 높은 peer들을 서버에 근접하여 위치시킬 수 있게 된다.

2. 제안 알고리즘

2.1 변수 정의

이 논문은 미디어 콘텐츠를 제공하는 스트리밍 서버와 각 peer들이 네트워크상에서 P2P로 연결되어 있는 환경을 가정한다. 각각의 peer들은 서버로부터 스트리밍을 통하여 데이터를 실시간으로 얻게 된다. 제안하는 방법의 알고리즘 기술에 관한 변수들의 정의는 아래와 같다.

L(p) : peer p에 연결된 link의 총 갯수
C_q : 각 peer별로 기여도를 저장할 순환 queue.
모두 0으로 초기화.
I : 각 peer별로 turn이 돌아왔을 때 queue에 저장할 위치.
sum : 순환 queue에 저장된 기여도의 총합
$\overline{U}$peer(p): peer p와 unused-link로 연결된 peer들의 집합
$\overline{U}$link(p): peer p와 연결된 unused-link의 집합
Upeer(p) : peer p와 used-link로 연결된 peer들의 집합
Ulink(p) : peer p와 연결된 used-link의 집합
α_L, α_H : 확장과 축소를 결정하는 used link의 비율 변수
sum_L : Shrink 연산 시 이웃 peer들에게 연결을 넘겨주는 기준이 되는 최소 sum

2.2 기여도를 적용한 topology 구성 알고리즘

제안하는 알고리즘을 전체적으로 pseudo code로 나타내면 algorithm 2와 같다. 전체적인 알고리즘은 기존의 방법과 유사하나 기여도 값을 계산하는 부분이 포함되었다는 점에서 차이가 있다. 제안하는 알고리즘은 queue에 일정시간동안 저장된 기여도를 통해서 누적된 결과로, peer의 Growth 및 Shrink 를 판단하기 때문에 기존 알고리즘보다 느리게 반응할 가능성이 있다. 이를 보완하기 위해 제안하는 알고리즘에서는 Growth Shrink 연산을 수행할 때 기여도가 더 높은 peer를 스트리밍 서버에 가까운연결을 가지도록 조절하는 기능을 수행한다.

Algorithm 2: Adaptive Overlay algorithm based on contribution principle

```
for(every δc received chunks) do
  /* 이번 turn의 used link를 Queue에 갱신 */
  p.sum += card(Ulink(p)) – p.Cq[p.I]
  p.Cq[p.I] = card(Ulink(p))
  p.I = ++p.I % MAX_Queue
   end
  /* scan and adjust p's neighborhood */
  if( card( Ū link(p) ) > αH card(Ulink(p) ) then
    /* Shrink the neighborhood */
    Algorithm 3 : Shrink algorithm
    end
  else if( card( Ū link(p) ) < αL card(Ulink(p) ) then
    /* grow the neighborhood */
      Algorithms 4: Growth algorithm
    end
```

2.3 Shrink 알고리즘

Shrink 알고리즘은 peer에 연결된 link의 비율이 높다고 평가되었을 때 수행된다. 기존 방법에서는 Shrink 알고리즘이 수행되면 기존에 peer가 가지고 있던 unused link들 중 기여도 합이 낮은 저평가된 peer와의 연결은 제거된다. 하지만 제안하는 알고리즘에서는 단순히 연결을 제거하지 않고 자신의 이웃 peer들에게 link를 넘겨주어서 자신이 해결하지 못했던 peer들의 분산을 돕는 기능을 추가로 수행한다.

```
Algorithm 3: Shrink algorithm pseudo code
/* 기여도 순으로 n- 개의 peer를 선택하여 연결 제거 후 이웃 peer에 편입 시킴*/
selection_queue = sort_by_ascending_use_sum( Ū peer(p))[0..n-]
  remain_neighborhood = {N0(p) - selection_queue - {p} }
  i = 0, j = 0
  for(; j < sizeof(remain_neighborhood); j++) then
    if(remain_neighborhood[j].sum > sumL)
       potential_parent_queue.add(remain_neighborhood[j])
    end
  end
  /* 선택된 이웃 peer의 수가 지나치게 적다면 전체 이웃 peer에 고르게 편입. */
  if (sizeof(potential_parent_queue ) < threshhold_num ) then
   remain_neighborhood =
   sort_by_descending_use_sum(remain_neighborhood)
   while( i < n- ) do
      add selection_queue [i] to N0(remain_neighborhood[i])
   end
 else
  /* 기여도 합을 이용하여 기여도가 높은 peer 우선으로 편입되도록 함. */
 sort_by_descending_use_sum( potential_parent_queue )
 while( i < n- ) do
  add selection_queue [i] to
  N0(potential_parent_queue[i++%size(potential_parent_queue)])
  end
 end
end
```

2.4 Growth 알고리즘

Growth 알고리즘은 peer에 연결된 link의 비율이 낮다고 평가되었을 때 수행된다. 기존 방법에서는 Growth 알고리즘이 수행되면 이웃 peer가 가지고 있는 이웃 peer와의 link들을 조사하여 새로 조사된 이웃의 이웃 peer들을 직접 연결하여 이웃 peer 리스트에 편입시킨다.

제안하는 알고리즘에서는 새로 편입되는 child들 중 parent peer보다 더 큰 기여도를 가지는 peer가 있다면 서버에 근접하도록 능동적으로 연결 관계를 바꿔준다. 큰 기여도를 가진 peer는 초기 연결 위치가 많은 연결을 거쳐서 chunk를 전송받는 서버에서 먼 위치에 있었다면 개선된 Growth 연산을 반복하게 되면 점점 더 서버에 가까운 위치로 접근하게 된다.

```
Algorithm 4: Growth algorithm pseudo code
/* 기여도 순으로 n+ 개의 peer를 선택 */
selection_Queue =
sort_by_descending_use_sum(C(p)- N0(p) -{p})[0..n+]
/* 만약 선택된 peer들 중 가장 기여도의 합이 큰 peer가 현
재 peer의 기여도 합보다 크다면 위치 교환 */
  if( selection_Queue [0].sum > p.sum ) then
    add selection_Queue[0] to p.parent
    add p to N0(selection_Queue[0])
  end
  add selection_Queue to N0(p)
end
```

IV. 시뮬레이션 결과

본 논문에서는 메쉬 기반의 P2P 스트리밍 환경을 Omnet++ 4.1과 denacast로 구현하였다. 입력되는 미디어 데이터는 Arizona university에서 제공하는 MPEG4 비디오 트레이스 파일들을 활용하였다. 시뮬레이션은 peer의 이탈이 잦은 네트워크 오류가 많은 모델과 peer의 이탈이 적은 모델 총 2개를 구현하고 있다. 각 모델에서 기존 알고리즘과 제안하는 알고리즘을 각각 실행시켜 나온 결과를 비교하는 형태로 실험을 진행한다.

peer의 이탈이 잦은, 네트워크 오류가 많은 모델은 Error Probability 변수 e의 값을 35%로 설정하여 전체적으로 오류가 많은 상황을 재현하고 있다. 이 모델은 제안하는 방법에서 연결의 강인함 및 네트워크 오류가 심할 때의 전송 효율이 개선되었는지 검증하기 위한 성능 비교 실험을 위해 제작되었다.

정적인 환경에서 시뮬레이션을 수행하는 목적은 추가한 알고리즘이 기여도 queue의 단점을 보완해주는지 검증하기 위함이다. 기존 방법과 비교하여 queue의 추가로 인한 오버헤드를 상쇄하였는지 성능을 비교하였다.

네트워크의 크기는 peer 수를 초기 150개로 시작한다. 점점 peer가 추가되어 시뮬레이션 내부 시간으로 120초가 될 때 약 1200개가 네트워크에 편입되도록 구성되었다. 트래커 서버와 스트리밍 서버를 제외한 모든 peer들의 경우 8~10ms의 회선 지연시간과 8Mbps의 다운로드 대역폭을 갖는다. 업로드 대역폭은 class별로 크게 A, B, C, D 4종류로 나누어 class A에 속한 peer에게는 5Mbps±10%, B는 1Mbps±10%, C는 0.5Mbps±10%, D는 0Mbps로 설정하였다. 전체 peer에서 각 class별로 존재하는 비율은 10%, 40%, 40%, 10%로 생성하였다. αL과 αH는 [6]에서 최적의 값이라고 밝힌 0.2, 0.3으로 실험하였으며 peer 생성 주기가 올 때마다 일정한 확률로 생성 대신 peer의 이탈이 일어나게 하였다. 기여도 queue의 사이즈는 10으로 설정하였고, Growth와 Shrink 연산 주기를 결정하는 chunk 수신수는 1000 으로 설정하였다.

표 1. 시뮬레이션 파라미터 설정
Table 1. System Environment

파라미터 항목	값
Start Peer Num	150
End Peer Num	1200
Simulation Time	120 sec.
rx Delay Time	8~10ms
Download Bandwidth	8Mbps
Upload Bandwidth for Class	A: 5Mbps±10% B: 1Mbps±10% C: 0.5Mbps±10% D: 0Mbps
αL, αH	0.2, 0.3
Size of Contribution Point Queue	10
Num of Chunks before Checking Growth Shrink Condition	1000
Error Probability e	5% , 35%

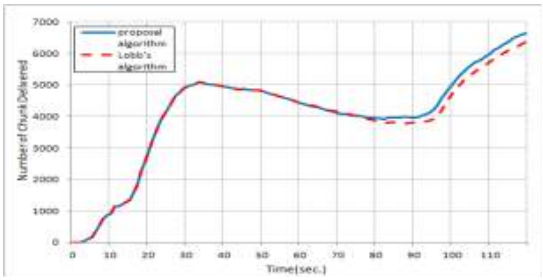
시뮬레이션이 진행되는 동안 각 class별로 연결된 link의 수와 각 class에 전송한 chunk의 수를 기록한다. message의 전송에 대해서는 따로 다루지 않는다. 만약 메시지 오버헤드가 성능 상에 영향을 미쳤다면 전체 네트워크의 chunk 전송량에 나타날 것이기 때문이다.

모든 시뮬레이션 결과는 그래프로 정리하였으며 실선 그래프가 제안하는 알고리즘의 결과이고, 점선 그래프가 기존 알고리즘의 결과값이다.

1. peer의 오류 발생률이 낮은 네트워크

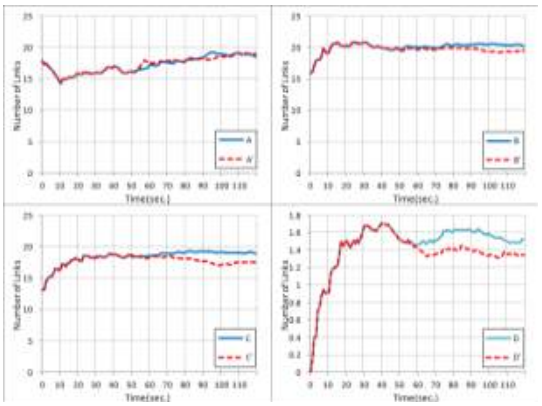
[그림 7]은 e = 5%인 환경에서 각 초마다 네트워크 전체에서 전송한 chunk의 양을 표현한다. 그래프를 보면 공통적으로 전송량이 감소하는 구간이 존재하는데, 이는 시뮬레이션

중간에 공통적으로 스트리밍 서버와 인접한 peer 몇 개가 이탈하면서 생긴 현상이다. 하지만 두 가지 알고리즘 모두 그 후에 해당 상황을 극복하여 전송량이 더 증가하고 있음을 알 수 있다. 또한 80초 이후 구간부터는 제안하는 알고리즘 쪽에서 확실히 더 많은 네트워크 전송량을 보이고 있음이 확인되었다.



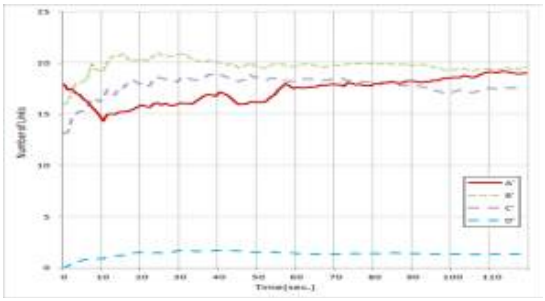
[그림 7] e = 5%, 초당 네트워크 총 전송량 변화
Fig. 7. Total num of transmitted chunks/sec. with e=5%

[그림 8]는 오류 발생률이 낮은 환경에서 각각의 class에서 peer당 가지는 link의 평균값을 시간 순으로 나타낸다. 공통적으로 전송 능력을 가지지 않는 class D의 peer가 낮은 link수를 가지고 있다. 하지만 제안하는 알고리즘이 class D에 link를 연결함에 있어 더 향상된 부분이 있으며 다른 class에서도 미세하지만 link의 수를 더 많이 유지하고 있음을 확인할 수 있다.

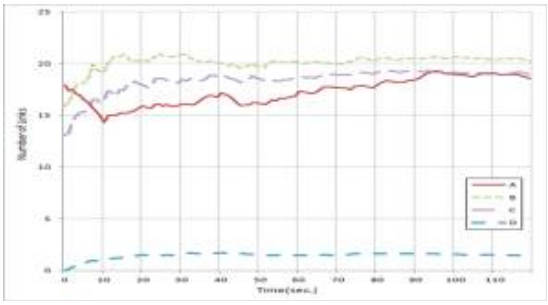


[그림 8] e = 5%, class의 각 peer별 평균 link수 비교
Fig. 8. Average num of links for a peer with e = 5%

[그림 9,10]은 e = 5%인 정적 모델에서 각 알고리즘을 수행했을 때 나온 link수 변화를 class별로 비교했다. [그림 9]는 기존 알고리즘이며 [그림 10]은 제안하는 알고리즘의 결과이다. 정적 모델에서는 시간이 흘러도 peer 평균 link수는 크게 변하지 않으며 또한 정적인 모델에서도 기존 알고리즘에 비교해 queue에 의한 오버헤드의 영향이 거의 없음을 알 수 있다.



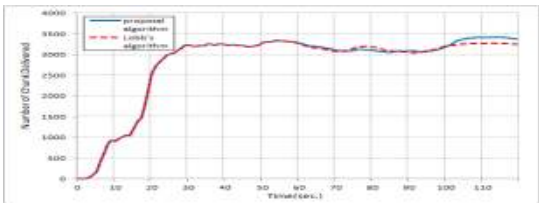
[그림 9] e = 5%, 기존 방식의 class별 link수 변화
Fig. 9. Num of links for each class using the existing algorithm with e = 5%



[그림 10] e = 5%, 제안하는 방식의 class별 link수 변화
Fig. 10. Num of links for each class using the proposed algorithm with e = 5%

2. peer의 오류 발생률이 높은 네트워크

[그림 11]은 e = 35%인 환경에서 각 초마다 네트워크 전체에서 전송한 chunk의 양을 표현한다. 그래프를 보면 [그림 7]의 안정적 모델에 비해 chunk 전송량이 전체적으로 감소했음을 알 수 있다. 100초 이후에는 제안하는 알고리즘이 안정적으로 더 많은 전송량을 보내고 있음이 확인된다.

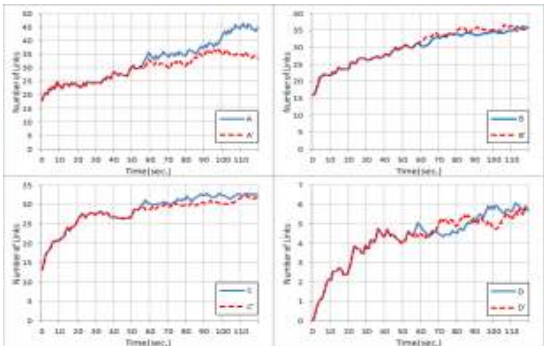


[그림 11] e = 35%, 초당 네트워크 총 전송량 변화
Fig. 11. Total num of transmitted chunks/sec. with e = 35%

[그림 12]는 e = 35%인 모델에서 각각의 class에서 peer당 가지는 link의 평균값을 시간 순으로 나타낸다.

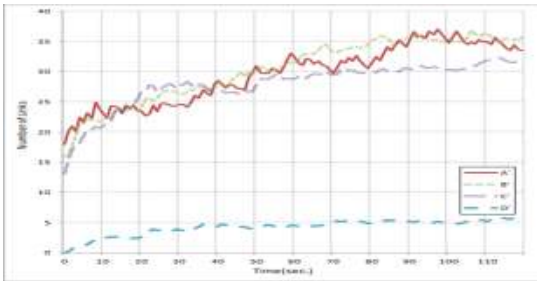
가장 업로드 대역폭이 큰 class A에서 제안하는 알고리즘

이 100초 구간을 전후해서 기존 알고리즘보다 더 크게 link 수가 증가한다. class A에서 큰 차이를 나타내기 시작한 100초 구간은 [그림 12]에서 제안하는 알고리즘이 기존 알고리즘보다 뛰어난 성능을 보이기 시작하는 구간이다. 따라서 100초 구간의 link수 증가가 성능의 향상을 가져왔다고 추측할 수 있다.

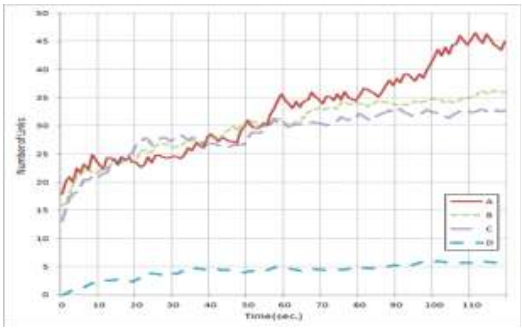


[그림 12] e = 35%, class의 각 peer별 평균 link수 비교
Fig. 12. Average num of links for a peer with e = 35%

[그림 13,14]는 e = 35%인 정적 모델에서 각 알고리즘을 수행했을 때 나온 link수 변화를 class별로 비교했다. [그림 13]이 기존 알고리즘이며 [그림 14]가 제안하는 알고리즘의 결과이다. [그림 9,10]과 비교하면 전체적으로 link수가 많고, 꾸준히 증가하고 있다는 것을 알 수 있다. 네트워크가 불안정할 때는 기존 알고리즘이나 제안하는 알고리즘 모두 최적의 link값을 찾기 위해 Growth와 Shrink 연산을 계속해서 수행한다. 이 과정 중에 제안하는 알고리즘은 queue에 담긴 기여도를 이용하여 peer의 확장 시에 더 많은 정보를 제공받을 수 있다.



[그림 13] e = 35%, 기존 방식의 class별 link수 변화
Fig. 13. Num of links for each class using the existing algorithm with e = 35%



[그림 14] e = 35%, 제안하는 방식의 class별 link수 변화
Fig. 14. Num of links for each class using the proposed algorithms with e = 35%

peer의 오류 발생률이 높은 환경에서 알고리즘을 실험하고 제안하는 알고리즘이 기존 알고리즘의 성능을 개선할 수 있음을 보였다. 또한 정적인 환경의 실험을 통하여 queue를 유지하는데 필요한 메시지 오버헤드가 실제 성능에 영향을 미칠 정도가 아니라는 점을 증명하였다. 마지막으로 불안정한 상황을 겪은 후 topology 전체의 성능을 회복함에 있어 제안하는 알고리즘이 기존 알고리즘보다 속도가 더 빠름을 보였다.

V. 결 론

본 논문에서는 peer간의 link 정보를 이용하여 업로드 대역폭을 추정하는 기존 알고리즘을 개선하였다. 제안하는 알고리즘은 일정량의 chunk를 전송받을 때마다 사용된 link의 정보를 일정시간 queue에 보관한다. 이 정보를 기여도라 정의하며 peer가 네트워크에 공헌한 정도를 표현한다. 기존 알고리즘에서 연결 확장이나 축소가 수행되어야 할 때 전체 link 대신 기여도 queue를 사용하여 peer를 판단한다. 이를 통해 peer의 일시적인 네트워크 지연 등 특수한 상황에서 기여도가 높은 peer에 우선순위를 주어 topology의 안정성을 얻을 수 있도록 보장하였다. 또한 Omnet++을 활용한 시뮬레이션을 통하여 기존 알고리즘과 제안하는 알고리즘의 동작을 검증하였다. 네트워크 오류가 잦은 모델에서의 실험을 통해 초기에 네트워크에 진입한 peer들이 네트워크를 재구축할 때 기존의 방법보다 제안하는 알고리즘에서 속도가 더 빨라졌음을 보일 수 있었다. 이를 통해 전체 네트워크의 전송 성능이 향상되었음을 확인하였다.

추후에는 제안된 방법으로 유지되고 있는 오버레이에서 특수한 예외 상황을 보장하기 위한 chunk 스케쥴링 전략을 연구할 것이다.

참고문헌

[1] F. Thouin, and M. Coates, "Video-on-Demand Newroks: Design Approaches and Future Challenges," IEEE Neworks, pp.42-48, March/April 2007.

[2] W.-P. Ken Yiu, X. Jin, and S.-H. Gary Chan, "Challenges and Approaches in Large-Scale P2P Media Streaming," IEEE Multimedia, pp.50-59, April-June 2007.

[3] D.-E. Meddour, M. Mushtag, and T. Ahmed, "Open Issues in P2P Multimedia Streaming," Proceedings of MultiComm'06, pp.43-48, June, 2006.

[4] Wang, Y. Xiong, and J. Liu, "mTreebone : A Hybrid Tree/Mesh Overlay for Application-layer Live Video Multicast," Proceeding of IEEE ICDCS'07, May, 2007.

[5] Haesun Byun, Meejeong Lee "A Hybrid P2P Overlay Architecture for Live Media Streaming," Journal of Korea Information Science Society, Vol.36, No.6, pp. 481-491, Dec. 2009.

[6] R. Lobb, A. P. Couto da Silva, E. Leonardi, M. Mellia, and M. Meo, "Adaptive Overlay Topology for Mesh-based p2p-tv Systems" in ACM NOSSDAV, June 2009.

[7] "Denacast" http://denacast.org

[8] "Omnet++ 4.1" http://omnet.org

[9] Pandurangan, G., Raghavan, P., Upfal, E. "Building Low Diameter Peer-to-Peer Networks," IEEE Journal on Selected Areas in Communications, Aug. 2003.

[10] B. Li and H. Yin, "Peer-to-Peer Live Video Streaming on the Internet: Issues, Existing Approaches, and Challenges," IEEE Communications Magazine, June, 2007.

[11] Q. Huang, H. Jin, and X. Liao, "P2P Live Streaming with Tree-Mesh based Hybrid Overlay," Proceeding of IEEE ICPPW'07, September, 2007.

[12] Hai Jin, Xuping Tu, Chao Zhang, Ke Liu, and Xiaofei Liao, "TCMM: Hybrid Overlay Strategy for P2P Live Streaming Services," GPC 2007, LNCS 4459, pp.52-63, 2007.

[13] Boon-Hee Kim, "Peer to Peer Search Algorithm based on Advanced Multidirectional Processing", Journal of The Korea Society of Computer and Information, pp.133-139, October 2009.

[14] Jin-Sung Kim, Dong-il Kim, Eunsam Kim, Sung-il Pae, "An Efficient Peer-to-Peer Streaming Scheme Based on a Push-Mesh Structure", Journal of The Korea Society of Computer and Information, pp.81-89, March 2010.

저 자 소 개

이 상 훈

2010: 경희대학교 컴퓨터공학과 공학사.

현 재: 경희대학교 컴퓨터공학과 석사 과정.

관심분야: 알고리즘, 그래프 이론, 메타휴리스틱 알고리즘

Email : a01b01c01@khu.ac.kr

한 치 근

1983: 서울대학교 산업공학과 학사

1985: 서울대학교 산업공학과 석사

1991: 미국 Pennsylvania 주립대학 Computer Science 박사

현 재: 경희대학교 컴퓨터공학과 교수

관심분야: 계산 이론, 알고리즘

Email : cghan@khu.ac.kr