韓國 컴퓨터情報學會 論文誌
第13卷 第2號, 2008. 3.

2008-13-2-3-4

네트워크 코딩에서의 유한필드 연산의 구현과 성능 영향 평가

이 철 우*, 박 준 상**

Performance Evaluation of Finite Field Arithmetic Implementations in Network Coding

Chul-Woo Lee*, Joon-Sang Park**

요 약

P2P(Peer-to-Peer) 시스템에서의 네트워크 코딩 기법의 사용은 파일전송시간을 단축할 수 있는 등 여러 장점들이 존재한다. 네트워크 코딩 방식과 기존의 통신 방식과의 가장 큰 차이점은, 발신지와 목적지 노드에서만 수행하던 데이터의 부호화와 복호화가 네트워크 코딩 방식의 경우 중간경유 노드들에서도 수행된다는 것이다. 그러나 네트워크 코딩 기법은 소프트웨어적으로 어떻게 구현하느냐에 따라 그 장점이 상쇄될 수 있는 많은 요소들은 존재한다. 먼저, 네트워크 코딩에서의 연산은 유한필드에서 정의되기 때문에 연산을 구현할 때 일반 연산명령을 이용할 수 없고 유한필드 연산 알고리즘을 필요로 하고 알고리즘의 선택이 시스템 전제척인 성능에 큰 영향을 준다. 또한, 필드의 크기 등 시스템의 성능에 큰 영향을 미치는 다른 요소들이 존재한다. 본 논문에서는 위와 같은 요소들이 네트워크 코딩의 성능에 미치는 영향을 살펴본다. 보다 구체적으로는 실험을 통해 이러한 요소들이 각각 2-5배정도의 성능 차이를 줄 수 있다는 사실을 보여주고 따라서 이러한 성능 분석을 토대로 네트워크 코딩을 이용한 시스템의 설계 시에는 가능한 한 큰 필드 크기를 선택하는 등 각 요소별로 시스템 성능의 최대화를 이룰 수 있는 선택을 할 것을 제안한다.

Abstract

Using Network Coding in P2P systems yields great benefits, e.g., reduced download delay. The core notion of Network Coding is to allow encoding and decoding at intermediate nodes, which are prohibited in the traditional networking. However, improper implementation of Network Coding may reduce the overall performance of P2P systems. Network Coding cannot work with general arithmetic operations, since its arithmetic is over a Finite Field and the use of an efficient Finite Field arithmetic algorithm is the key to the performance of Network Coding. Also there are other important performance parameters in Network Coding such as Field size. In this paper we study how those factors influence the performance of Network Coding based systems. A set of experiments shows that overall performance of Network Coding can vary 2-5 times by those factors and we argue that when developing a network system using Network Coding those performance parameters must be carefully chosen.

▸ Keyword : 네트워크 코딩(Network Coding), 유한필드(Finite Field), 가우스 소거법 (Guassian Elimination)

• 제1저자 : 이철우 교신저자 : 박준상
• 접수일 : 2008. 2. 26, 심사일 : 2008. 3. 4, 심사완료일 : 2008. 3. 18.
* 홍익대학교 컴퓨터공학과 학사과정 **홍익대학교 컴퓨터공학과 전임강사
※이 논문은 2007년 정부(교육인적자원부)의 재원으로 한국학술진흥재단의 지원을 받아 수행된 연구이고(KRF-2007-331-D00384) 2007학년도 홍익대학교 학술연구진흥비에 의하여 지원되었음

Ⅰ. 서 론

현재 인터넷에서 사용자간의 데이터 공유에 많이 쓰이는 P2P(Peer-to-Peer) 방식은, 지정된 서버에서 각 클라이언트들에게 데이터를 전송하는 기존의 클라이언트-서버 모델과는 달리, 클라이언트와 서버의 역할 구분 없이 모든 컴퓨터들이 클라이언트와 서버의 두 가지 기능 모두를 수행한다. 즉, 모든 컴퓨터들이 데이터를 다른 컴퓨터로 전송함과 동시에 다른 컴퓨터로부터 전송 받는다. 이러한 P2P 방식의 시스템의 한 가지 형태로, 보다 많은 컴퓨터(또는 노드)들이 참여할수록 시스템의 성능이 증가하는 구조인 단말시스템 협력 구조가 있다[1]. 이러한 구조를 이용한 대표적인 예로 비트토런트(BitTorrent)[2]를 들 수 있다. 비트토런트 시스템에서는 하나의 파일이 여러 개의 조각으로 나누고 각각의 개별 조각을 전송단위로 전송하기 때문에 동시에 여러 노드로부터 하나의 파일의 완성에 필요한 여러 조각 정보의 동시 수신이 가능하고 따라서 통신하는 노드 수에 증가함에 따라 파일 수신 속도가 증가한다. 이러한 비트토런트 시스템의 가장 큰 문제는 희소조각문제이다. 예를 들면, n개의 조각으로 나눠진 특정 파일을 가정할 때, 특정 노드가 이 파일의 n-1개의 조각들을 전송받는데 걸리는 시간보다 나머지 한 조각을 전송 받는데 걸리는 시간이 전체 파일 전송시간에서 더 큰 부분을 차지할 수 있다는 것이다. 이러한 문제에 대한 해결책 중 하나는 최근에 제안된 네트워크 코딩[3] 기법이다.

네트워크 코딩이란 일반적인 통신망 구조와는 달리 중간경유노드(또는 라우터)에서 서로 다른 패킷들을 혼합하는 기법을 지칭한다. 일반적인 통신망에서는 발신지에서 생성된 패킷의 내용은 중간경유노드(또는 라우터)에서 변경되지 않고 목적지 까지 전달되는데 반하여, 네트워크 코딩 기법을 사용하는 통신망의 경우 중간노드에서 서로 다른 패킷들이 혼합되는 등, 패킷의 내용의 변경을 허용한다. 네트워크 코딩 기법은 [3]에서 멀티캐스트의 성능향상을 위하여 처음 제안되었고 [1]에서 비트토런트 방식의 시스템에 적용할 것을 경우 파일의 다운로드 시간을 크게 줄일 수 있음을 보여주었다. 네트워크 코딩에 기반을 둔 비트토런트 시스템에서는 특정 파일의 원본 조각들을 전송하는 대신, 부호화(coding)된 조각을 전송한다. 그러한 코딩된 조각의 특징은 조각들 간의 구분이 없다는 점이다. n개의 조각으로 이루어진 특정 파일의 완전한 다운로드를 위해서는, 네트워크 코딩을 사용하지 않았을 경우, n개의 개별 조각들을 각각 전송받아야 하나, 네트워크 코딩 기법을 이용했을 경우 조각들 간에 구분이 없어 특정조건을 만족하는 n개의 조각만을 수집하면 된다. 따라서 희소조각문제가 발생하지 않아 비트토런트 시스템의 전반적인 성능향상을 꾀할 수 있다. 그러나 네트워크 코딩 기법을 소프트웨어적으로 구현할 경우 파일 디코딩 단계, 즉 모아진 파일 조각들로부터 원본 파일로 복구하는 단계의 실행 속도가 매우 느려 사용자의 전체적인 체감 성능이 떨어지는 현상이 발생할 수 있다. 다시 말하면, 네트워크 코딩을 이용하면 파일 조각들을 모두 전송받는 시간은 감소하나 파일 조각들로부터 원본 파일을 복구하는 시간이 크게 증가하여, 네트워크 코딩의 미사용 시와 비교하여 시스템의 전체적인 성능 향상이 크지 않을 수 있다. 따라서 네트워크 코딩의 구현에 관한 연구는 매우 중요하다 할 수 있겠다. 이에, 본 논문에서는 네트워크 코딩 기법에 기반을 둔 시스템의 구현 시 발생될 수 있는 성능 문제를 보여주고 그 성능 문제에 효과적으로 대응할 수 있는 방안들을 제시하고자 한다. 보다 구체적으로는, 먼저 시스템 성능에 가장 큰 영향을 미치는 요소는 유한필드(Finite Field) 연산 알고리즘임을, 그리고 필드 연산 알고리즘의 선택에 따른 시스템의 성능에 큰 차이가 있음을 보여준다. 또한, 네트워크 코딩의 성능에 상당한 영향을 미치는 다른 변수들과 그 변수들에 따라 시스템의 성능이 어떻게 달라지는지 보여준다.

본 논문은 다음과 같이 구성되어 있다. 2장에서는 네트워크 코딩 관련 연구들을 간략히 소개하고, 3장과 4장에서는 각각 네트워크 코딩 기반 파일 송수신 방식과 그 소프트웨어적 구현 방식을 설명한다. 5장에서는 네트워크 코딩의 성능에 영향을 주는 요소들과 그 영향을 분석하고, 마지막으로 6장에서는 결론을 도출한다.

Ⅱ. 관련 연구

네트워크 코딩은 [3]에서 처음 제안된 새로운 기법으로 유선 통신망에서 출발하여 그 응용 분야를 넓히고 있다. 네트워크 코딩 기법의 사용 시 얻을 수 있는 가장 큰 장점은 통신망의 전송량의 증가이다. 주어진 통신망에서 멀티캐스트 방식의 데이터 전송을 할 경우 최대 전송량을 달성하기 위해서는 네트워크 코딩 기법을 사용해야 한다는 사실, 즉 중간경유노드에서 서로 다른 경로를 통하여 전달된 서로 다른 데이터들을 혼합 또는 부호화(coding)해야 함이 [3]에서 증명하였다. 그러나 이런 네트워크 코딩 기법을 사용할 경우 가장 어려운 문제들 중의 하나는 어떤 중간경유 노드에서 어떻게 부호화를 할 것인가의 문제인데 주어진 네트워크에서 최적해를 구하기 위한 전제조건은 네트워크의 전체 위상과 간선들의 비용이 알

려져 있다는 것이다. 이후, [4]에서는 그러한 전제조건하에서 선형부호의 사용이 최적해 달성을 위한 충분조건이라는 사실이 증명하였으나, 네트워크의 전체 위상 및 간선 정보를 필요로 한다는 점이 여전히 네트워크 코딩의, 특히 위상이 변하는 동적 네트워크에서의, 실제 사용에 가장 큰 제약점이었다. 그러나 [5][6]에서 전체 위상정보를 필요로 하지 않는 랜덤 리니어 코딩(Random Linear Coding) 기법의 제안으로, 이후 실용적인 유무선 네트워크에서 사용이 가능한 몇몇 프로토콜(예, [5][7])이 개발될 수 있었고 비트토런트 시스템형태의 P2P 시스템에의 적용(예, [1])도 실험되었다. [1]의 제안으로 시작된 네트워크 코딩의 P2P 시스템에의 적용에 대한 연구로는 [8][9]등이 있다. [8]에서는 네트워크 코딩 기반 P2P 시스템에서의 보안문제에 대한 한 가지 해결 방안을 [9]에서는 무선환경에서 네트워크 코딩 기반 P2P 시스템 방식을 제안하였다.

유한필드 연산 구현에 관한 연구는 암호학 분야 등에서의 필요성으로 인해 다수의 연구들이 이루어졌다[10]. 암호학에서의 유한필드 연산 구현에 관한 연구로는 몽고메리 알고리즘[11]등이 있은데 이는 암호학 분야에서의 빠른 지수승의 계산의 필요성과는 달리 네트워크 코딩에서는 그 효용성이 존재하지 않는다. 본 논문에서는 유한필드 연산 구현 방법 중 네트워크 코딩에 적합한 알고리즘을 선택하여 제시하고 이의 구현 및 실험을 수행한다.

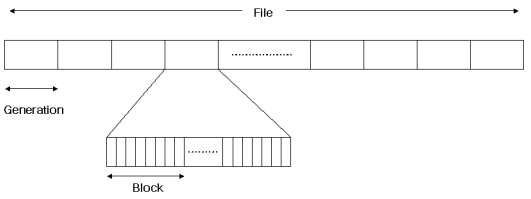

그림 1. 파일의 분할
Fig 1. Partition of File

III. 네트워크 코딩 기반의 파일 송수신

[1][3]에서 제안된 네트워크 코딩 기법에 따라 전송될 파일은 (그림 1)과 같이 분할된다. 네트워크 코딩에서의 연산은 유한필드에서 정의되어 있기 때문에 가장 작은 단위는 필드 크기를 가진 데이터이다. GF(2^8)를 가정하면 데이터의 최소 단위는 1 바이트이다. 이러한 필드 크기의 데이터들이 모여 하나의 블록을 이룬다. 즉, 필드 크기의 데이터들의 집합을 블록이라, 그리고 블록의 크기는 그러한 데이터들의 개수라 정의 할 수 있다. 네트워크상에서의 사용자간 데이터 전송단위는 이러한 블록이다. 이러한 블록들의 집합이 하나의 제네레이션(Generation)이다. 따라서 제네레이션의 크기는 특정 제네레이션을 이루는 블록의 개수이다. 단, 본 논문에서는 모든 제네레이션이 같은 크기를 갖는다고 가정한다. 제네레이션은 각 노드에서 인코딩 및 디코딩의 단위가 되고 전체적인 시스템의 성능에 가장 중요한 요인 중의 하나이다.

3.1. 인코딩과 송신

인코딩은 제네레이션 단위로 수행한다. 하나의 제네레이션에서 블록들을 읽어온 후, 읽어온 블록을 이용하여 다음과 같이 인코딩을 수행한다. 개략적인 네트워크 코딩에서의 인코딩 과정이 (그림 2)에 나타나 있다.

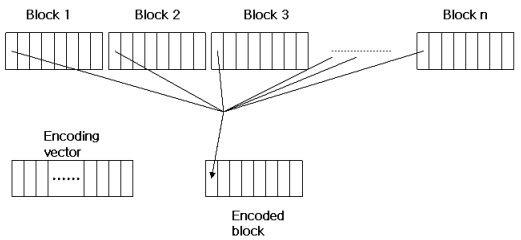

그림 2. 인코딩 과정
Fig 2. Encoding Process

(그림 2)에서는 제네레이션 크기와 블록 크기를 각각 n 과 8로 가정한다. 하나의 제네레이션에 속해있는 블록들의 집합을 $\mathbf{b^1 b^2 \ldots b^n}$로 나타날 때 하나의 블록은 벡터 $\mathbf{b^k} = [b_1^k, b_2^k, \ldots, b_8^k]$ (k = 1, 2, .., n)로 나타낼 수 있다. 인코딩을 위해서는 추가적으로 인코딩 벡터 $\mathbf{e} = [e_1, e_2, \ldots, e_n]$가 필요한데, 여기서 인코딩 벡터의 각 원소들은 정의된 유한 필드 내에서 무작위로 선택된다. 블록과 인코딩 벡터를 이용하여 생성되는 부호화된 블록을 벡터 $\mathbf{p} = [p_1, p_2, \ldots, p_n]$로 나타낼 때, 벡터 p의 각 원소는 다음과 같이 계산한다.

$$\mathbf{p_j} = \sum_{i=1}^{n} \mathbf{e_i b_j^i} \quad (j = 1, 2, .., 8)$$

앞서 밝힌 바와 같이, 인코딩 벡터의 원소들은 유한 필드 내에서 무작위로 뽑는다. 따라서 매번 원소들이 달라지고 이

러한 과정 때문에 랜덤 리니어 코딩[5][6]으로 지칭된다. 이러한 무작위성은 네트워크상에서 각 노드에서의 패킷 수락의 결정에 있어서 이점으로 작용한다. 네트워크 코딩을 쓰지 않는다면 각 노드에서는 정책에 따라 어떠한 블록을 받을 것인지에 대한 복잡한 결정해야 하는데 이는 추가의 비용이 드는 과정이다[1]. 이러한 결정 과정을 쓰는 이유는 각 블록이 네트워크상에서 중복되어 나타날 수 있기 때문이다. 네트워크 코딩에서 필드 크기가 커질수록 중복성이 일어날 가능성은 매우 낮아지기 때문에 위와 같은 추가 비용의 발생을 줄일 수 있다.

네트워크 코딩에서는 모든 연산이 유한필드에서 정의되고 수행되기 때문에 네트워크 코딩의 소프트웨어적 구현 시 일반 연산자의 사용이 불가하고 유한필드 연산 알고리즘의 사용이 필요하고 여기서 알고리즘의 선택의 문제가 나타난다. 유한필드 연산 알고리즘의 문제는 4장에서 다룬다.

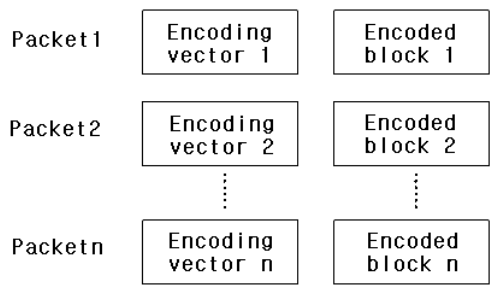

그림 3. 패킷의 구성
Fig 3. Organization of Packet

최종적으로 사용자간 데이터 교환에 사용될 패킷들의 모습은 (그림 3)과 같다. 생성된 패킷들은 부호화된 블록(그림 3에서는 encoded block로 표기)을 생성할 때 사용된 인코딩 벡터를 포함하고 있다. 인코딩 벡터가 패킷의 용량에서 차지하는 비율은 블록 크기, 필드 크기, 그리고 제네레이션 크기에 달려있으나 전체 패킷에서 인코딩 벡터의 용량이 차지하는 용량의 비율은 높지 않기 때문에 전체적인 성능에는 크게 영향을 미치지 않는다. 특정 노드는 다른 노드로부터 요청이 있을 때, (그림 3)와 같은 패킷을 개별적으로 전송한다. 특정 노드가 수신된 패킷으로부터 원본 파일을 복구해낼 수 있으려면 각 제네레이션 당 특정 조건을 만족하는 n개 이상의 패킷이 존재하여야 한다. 여기서 n은 제네레이션 크기이다. 자세한 복구(또는 디코딩) 과정은 다음절에 기술되어 있다.

3.2. 디코딩

디코딩은 다른 노드로부터 수신한 패킷들로부터 원본 파일을 복원해내는 과정이다. 디코딩 과정은 인코딩 과정에 비해 큰 리소스를 차지하는 비용이 큰 수행 과정이다. 특정 노드가 다른 노드로 부터 받은 패킷들의 집합이

$$\{(\mathbf{e}^1,\mathbf{p}^1),(\mathbf{e}^2,\mathbf{p}^2),.....,(\mathbf{e}^n,\mathbf{p}^n)\}$$

이라 가정하면 (여기서, 인코딩 벡터는 $\mathbf{e}$, encoded block은 $\mathbf{p}$, n은 제네레이션 크기), 수신된 패킷들로부터 원본 데이터를 얻어내기 위해서는 (그림 4)과 같은 식을 풀어내야 한다. 여기서 m은 블록 크기, n은 제레네이션 크기를 나타낸다. (그림 4)에 나타난 수식에서 알아내야 할 데이터는 원본 블록 벡터들 $\mathbf{b}^k$ (k = 1, …, n)이고 가우시안 소거법(Gaussian elimination)을 이용하여 좌변으로부터 우변을 만들어 냄으로서 알아낼 수 있다. 가우시안 소거법을 이용하여 $\mathbf{b}^k$ (k = 1,…, n)를 알기 위해서는 적어도 n개(또는 제네레이션 크기 또는 특정 제네레이션에 속한 원본 블럭의 개수)의 패킷들을 다른 노드들로부터 수집하여야 한다. 또 다른 필요조건은 수집된 인코딩 벡터들이 모두 일차독립(linearly independent)이어야 한다. 즉, 패킷을 수신할 때, 패킷에 포함된 인코딩 벡터가 디코딩 행렬의 계수(rank)를 증가 시키

$$\begin{Bmatrix} e_1^1 e_2^1 \ldots e_n^1 \\ e_1^2 e_2^2 \ldots e_n^2 \\ \cdots\cdots\cdots\cdots \\ e_1^n e_2^n \ldots e_n^n \end{Bmatrix} \begin{Bmatrix} p_1^1 p_2^1 \ldots p_m^1 \\ p_1^2 p_2^2 \ldots p_m^2 \\ \cdots\cdots\cdots\cdots \\ p_1^n p_2^n \ldots p_m^n \end{Bmatrix} \overset{\text{decoding}}{=\!=} \begin{Bmatrix} 1 0 \ldots 0 \\ 0 1 \ldots 0 \\ \cdots\cdots \\ 0 0 \ldots 1 \end{Bmatrix} \begin{Bmatrix} b_1^1 b_2^1 \ldots b_m^1 \\ b_1^2 b_2^2 \ldots b_m^2 \\ \cdots\cdots\cdots\cdots\cdots\cdots \\ b_1^n b_2^n \ldots b_m^n \end{Bmatrix}$$

그림 4. 디코딩 과정
Fig 4. Decoding Process

는지를 확인해야 한다. 디코딩 행렬의 계수를 증가시키는 패킷은 '혁신적이다'라고 한다. 패킷 수신 시, 계수 증가의 확인은 가우시안 소거법을 이용하기 위해서 필요한 조건이다. 혁신적이지 않은 인코딩 벡터는 가우시안 소거법에 의해 0인 행으로 되므로 이러한 인코딩 벡터를 포함한 패킷은 수신 후 폐기한다. 특정 파일을 복원하고자 하는 노드는 위의 조건을 만족시키는 패킷들을 충분히 수집하여 (그림 4)와 같은 디코딩 행렬을 만든 후, 가우시안 소거법을 이용하여 원본 블록들을 복원해 낸다. 이런 절차를 각 제네레이션 별로 수행한 후, 복원된 원본 블록들을 순서대로 나열하여 연결하면 원본 파일의 복원 과정이 완료된다.

가우시안 소거법을 살펴보면 디코딩 과정에서 많은 횟수로 연산들이 수행된다는 것을 알 수 있다. 가우시안 소거법 자체의 복잡도는 $O(n^3)$으로 더 이상의 성능 개선은 이루어지기 힘들 것이나 가우시안 소거법의 수행에서 사용되는 유한 필드 연산의 구현에는 넓은 선택의 폭이 존재한다. 다음절에서는 유한 필드 연산 알고리즘에 대하여 살펴보겠다.

Ⅳ. 네트워크 코딩의 구현

3장에서 설명된 인코딩/디코딩 과정을 소프트웨어적으로 구현할 때에는 전반적인 시스템의 성능에 영향을 미치는 요소들이 다수 존재한다. 그 첫째는 사용할 유한 필드의 선택이다. 유한 필드는 크게 소수 필드(Prime Field)인 GF(p)와 이진 확장 필드인 GF(2^k)로 나뉘는데, 소프트웨어적으로 구현할 경우 실행 속도를 높일 수 있는 이진 확장 필드를 많이 사용한다. 본 논문에서도 GF(2^k)를 사용하였다. 이 때, 어떤 k를 선택하느냐에 따라, 즉 어떤 필드 크기를 사용하느냐에 따라 시스템의 성능이 상당히 달라진다.

둘째는 유한 필드 연산 알고리즘의 선택이다. 유한 필드의 연산자 중 덧셈과 뺄셈의 연산은 XOR만으로 이루어지는데 반하여 곱셈연산은 보다 복잡한 알고리즘이 필요하다. 덧셈과 뺄셈은 모두 피연산자 a와 b가 주어졌을 때 a XOR b 를 통해서 결과를 얻어낸다. 유한 필드 곱셈연산은 GF(2^k)를 가정할 때 피연산자인 a를 $a_0a_1a_2....a_{k-1}$의 비트 스트링으로 표현된 $a_0X^0+a_1X^1+.....+a_{k-1}X^{k-1}$형태의 다항식으로 보아 두 개의 비트스트림 형태의 피연산자 a와 b의 곱을 통해서 얻어진 결과를 N으로 모듈러 연산을 취하면 완료된다. 여기서 N은 차수가 k인 기약다항식(irreducible polynomial)이다. 위와 같은 연산을 수행하기 위한 가장 기본적인 알고리즘은 쉬프트와 덧셈 연산의 반복만으로 구현이 가능하다.

유한 필드에서의 보다 효율적인 곱셈 연산 알고리즘은 테이블참조(Table Lookup) 알고리즘이다. GF(2^8)를 가정할 때, 미리 준비된 256 바이트 로그 테이블과 256 바이트 지수 테이블을 이용하여 두 피연산자의 곱셈 결과를 얻어낸다. 이 알고리즘은 추가적인 리소스를 필요로 하는데 그것은 위의 두 테이블을 저장하는 공간이다. 단순히 테이블에서 해당 값을 찾아내는 연산만이 필요하기 때문에 기본 알고리즘의 쉬프트와 덧셈 연산의 반복에 비해 보다 효과적이라 하겠다. 테이블 참조 알고리즘과 관련하여 필드 크기에 따른 성능 평가를 위한 실험을 위하여 필드GF(2^{16})을 위한 연산 알고리즘이 추가로 구현이 되었는데 이를 위해서는 (65536 x 2) 바이트 크기의 로그 테이블과 같은 크기의 지수 테이블을 이용하였다. 이들 기본 알고리즘과 테이블참조 알고리즘에 대한 개략적인 구현은 (그림 5)에 나타나있다. (보다 자세한 내용은 [12]를 참조한다.)

```
procedure Base(a, b)
{
  results = 0;
  while (a != 0) {
        if ((a & 1) != 0)
           results ^= b;
        overflow = b & 0x80;
        b <<= 1;
        if (overflow == true)
           b ^= 0x1d;
        a >>= 1;
  }
  return results;
}

procedure TableLookup(a, b)
{
  if (a == 0 || b ==0) return 0;
  s = logtable[a] + logtable[b] % Q; //Q=(필드크기-1)
  return exptonentiationable[s];
}
```

그림 5. 유한필드 연산 알고리즘
Fig 5. Finite Field Arithmetic Algorithms

마지막으로 제네레이션 크기 또한 시스템의 성능에 큰 영향을 미치는 요소이다. 다음 장에서는 위의 세가지 요소들이 시스템의 성능 영향에 미치는 영향을 살펴본다.

Ⅴ. 성능 분석

본 장에서는 네트워크 코딩 기법을 C언어로 구현하고 실제

실험으로 얻어진 결과를 토대로 성능분석을 수행한다. 실험은 리눅스를 운영체제로 사용하는 Intel Pentium 4 2.4GHz CPU, Cache size 512KB, Memory 1GB와 Intel Xeon 3.2GHz, Cache size 2048KB, Memory 4GB의 두 가지 서로 다른 환경의 컴퓨터에서 수행되었다. 비교의 척도가 되는 실행시간은 인코딩과 디코딩과정 모두를 포함하도록 다음과 같은 과정의 총 수행시간을 측정하였다. 처음 하나의 파일을 제네레이션 단위로 분할하여 각 제네레이션 단위로 3.1절과 같이 인코딩을 한다. 제네레이션 크기가 n일 경우 각 제네레이션 당 n개의 일차독립적인 인코딩 벡터에 의하여 생성된 n개의 부호화된 블록을 발생시킨다. 파일에 속한 모든 제네레이션에 대한 인코딩이 완료된 후, 부호화된 블록(또는 패킷)들에 대하여 각 제네레이션 별로 3.2.절에 기술된 방법으로 디코딩을 하고 디코딩된 제네레이션들을 합하여 원본 파일을 얻어낸다.

5.1 곱셈 알고리즘의 비교

앞서 살펴본 바와 같이 네트워크 코딩에서 유한 필드의 연산은 큰 부분을 차지한다. 제네레이션 크기를 n, 블록 크기를 m이라 할 때 인코딩과 디코딩에서 곱셈 연산 알고리즘은 각각 $O(n^2m)$, $O(n^2(n+m))$번 수행된다. 이렇게 알고리즘의 실행이 반복적으로 많이 일어난다는 것은 알고리즘의 성능이 전체적인 네트워크 코딩의 성능에 크게 영향을 준다는 것을 나타낸다. 본 절에서는 4장에서 살펴본 두 가지 유한필드 곱셈 연산 알고리즘에 따른 네트워크 코딩의 실행시간의 영향을 살펴본다. Intel Pentium4 2.4 GHz 환경과 Intel Xeon 3.2GHz 환경 모두에서 실험을 실시하였고 제네레이션 크기를 10으로 그리고 블록 크기를 1000으로 가정하였다. 각 점들은 10번의 실험을 통해 얻어진 값들의 평균값이고 표준편차비율은 모두 0.1% 이하로 매우 낮게 나타났다.

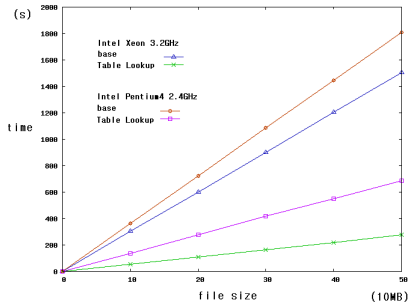

그림 6. 알고리즘 비교
(제네레이션 크기 = 10, 블록 크기 = 1000)
Fig 6. Comparison of Algorithm
(Generation size = 10, Block size = 1000)

알고리즘에 따른 영향은 (그림 6)에서 보여준다. 그래프 상에서 base로 표기된 기본 알고리즘에서는 쉬프트와 덧셈 연산이 반복적으로 일어난다. 이와 비교 대상이 되는 테이블참조 알고리즘은 단순히 테이블에서 해당 값을 찾아내는 연산만이 필요하기 때문에 기본 알고리즘의 쉬프트와 덧셈 연산의 반복에 비해 보다 효과적이다. 그래프를 살펴보면 기본 알고리즘을 사용한 전체적인 성능에 비하여 테이블참조 알고리즘을 사용했을 때가 실행시간측면에서 Intel Pentium4 2.4GHz의 경우 약 2.6배, Intel Xeon 3.2GHz의 경우 약 5.7배정도 뛰어나다. 또한 파일 크기의 증가에 따른 성능의 차이는 크게 나타난다. 작은 크기의 파일에 있어서는 알고리즘의 영향은 별로 크지 않았다. 하지만 파일의 크기가 커질수록 네트워크 코딩의 실행시간 차이는 크게 벌어짐을 알 수 있다.

결론적으로 네트워크 코딩의 전체적인 성능에 유한필드 곱셈 알고리즘이 미치는 영향이 크다. 때문에 효과적인 알고리즘의 사용은 네트워크 코딩을 사용하는 전체적인 시스템의 성능에 크게 도움이 된다. 특히 대용량의 파일을 다루는 경우 그 영향은 배가된다는 것을 본 실험을 통해서 알 수 있다. 최근의 P2P 방식의 파일의 교환의 경우 그 크기가 매우 크기에 네트워크 코딩에 기반을 둔 P2P 시스템의 구현 시 효율적인 유한필드 곱셈 알고리즘의 사용은 특히 중요하다 할 수 있겠다.

5.2 필드 크기

본 절에서는 필드 크기가 네트워크 코딩의 성능에 미치는 영향을 살펴본다. 실험에서는 Intel Xeon 3.2GHz 환경과 테이블참조 연산 알고리즘을 사용하였다. 필드 크기는 네트워크 코딩에서의 가장 기본적인 연산 단위가 되고 필드 크기에 따라 블록의 크기 또한 결정된다. 5.1절에서 수행된 실험은 필드 $GF(2^8)$만을 가정하였으나 필드 크기 또한 네트워크 코딩의 성능에 영향을 미치므로 필드 크기의 변화에 따른 네트워크 코딩 성능의 변화를 살펴보는 것은 중요한 일이다. $GF(2^8)$의 경우 기본 연산 단위가 1 바이트이고 $GF(2^{16})$에서의 기본 연산 단위는 2 바이트가 된다. 따라서 블록 크기가 1000으로 동일하면 $GF(2^8)$를 가정하면 블록 하나의 크기는 1000 바이트가 되고 $GF(2^{16})$의 경우 하나의 실제 블록 크기는 2000 바이트가 된다.

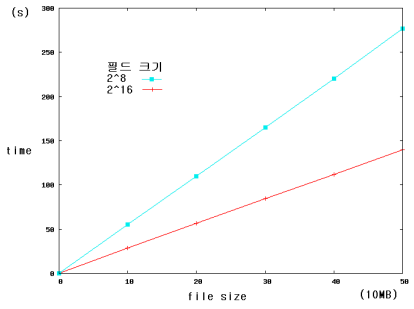

그림 7. 필드 크기에 따른 성능 비교
(제네레이션 크기 = 10)
Fig 7. Comparison of Perfomance
by Field Size
(Generation size = 10)

실제 시스템간의 교환에 사용되는 패킷의 실제 크기를 동일하게 하기 위하여 필드 GF(2^8)는 블록 크기를 1000, 필드 GF(2^{16})는 블록 크기를 500으로 설정하였다. 그래프에 나타난 수치들은 10번의 실험을 통해 얻어진 값들의 평균값이고 표준편차비율은 모두 0.1% 이하로 매우 낮게 나타났다. (그림 7)에 나타나 있는 실험 결과를 살펴보면 필드 GF(2^{16})가 필드 GF(2^8)에 비하여 약 1.9배 정도 성능이 좋은 것으로 나타난다. 이는 필드 GF(2^{16})가 전체 연산 횟수에서 필드 GF(2^8)의 전체 연산 횟수의 1/2정도가 되기 때문이다.

위의 결과를 통해 필드 크기가 클수록 전체 네트워크 코딩의 성능이 향상된다는 것을 알 수 있다. 하지만 필드 GF(2^{32})를 고려해 보면 필드 크기 증가에 제약이 있음을 알 수 있다. 테이블 참조 알고리즘을 위해 로그 테이블과 지수 테이블이 필요로 하다. 만약 필드 GF(2^{32})을 위한 테이블을 만든다면 각 테이블의 크기는 (4 x 4)GB가 된다. 최근 메모리의 용량이 늘어나고 있음을 감안한다 해도 위 테이블의 용량은 부담이 되는 요소로 작용한다. 따라서 현실적으로 일정 리소스 이상을 차지하는 필드 크기는 고려되기가 불가능하다.

5.3 제네레이션 크기

앞서 살펴본 알고리즘과 필드 크기 영향뿐만이 아니라 제네레이션 크기 또한 네트워크 코딩의 성능에 큰 영향을 미친다. 제네레이션의 크기에 따라 네트워크 코딩에서의 여러 조건들이 변화한다. 우선 인코딩과 디코딩에서 연산의 횟수가 결정된다. 앞서 기술된 바와 같이 제네레이션 크기를 n, 블록 크기를 m이라 할 때 디코딩의 연산은 $O(n^2(n + m))$로 제네레이션 크기의 영향이 크다는 것을 알 수 있다. 연산시간 측면에서만 보면 제네레이션 크기의 최소화가 전체 시스템의 성능향상에 도움을 준다고 보일 수 있으나 다른 문제가 발생한다. 제네레이션 크기가 작아질 경우 네트워크 코딩 기법의 도입의 목적이 상실된다. 즉, 비트토런트 시스템에서 나타나는 문제인 희소조각 문제가 나타나게 된다. 제네레이션 크기가 작다면 파일을 나누는 단위인 제네레이션의 개수가 증가하고 이는 희소 제네레이션이 나타날 확률이 높아지는 결과를 가져온다.

본 절에서는 제네레이션 크기 변화에 따른 성능의 변화를 살펴본다. 실험은 테이블참조 알고리즘을 사용하였다. 그래프의 각 점들은 10번의 실험을 통해 얻어진 값들의 평균값이고 표준편차비율은 모두 0.1% 이하로 매우 낮게 나타났다. (그림 8)에서 보이듯이 제네레이션 크기가 증가될수록 네트워크 코딩의 실행시간이 큰 폭으로 증가 되는 것을 볼 수 있다. 이 실험에서는 다른 모든 조건이 동일하므로 제네레이션 크기가 증가할수록 파일을 이루는 제네레이션의 개수가 감소한다.

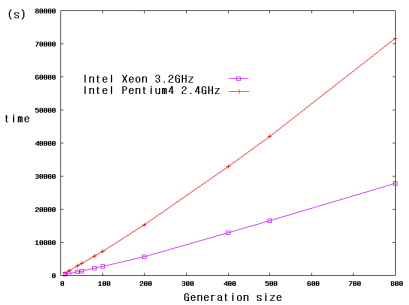

그림 8. 제네레이션 크기 변화에 따른 성능 비교
(파일 크기 = 500MB, 블록크기 = 1000)
Fig 8. Comparison of Performance
by variation of Generation size
(File size = 500MB, Block size = 1000)

(그림 8)로부터는 제네레이션 크기의 최대화를 통해서 시스템의 성능을 향상시킬 수 있다는 결론을 도출할 수 있으나 제네레이션 크기가 작아짐에 따라서 나타나는 문제점을 고려한다면 적정 수준의 제네레이션 크기를 유지해야 한다. 제네레이션 크기를 결정하는데 있어서 판단을 어렵게 하는 또 다른 요소는 네트워크에 참여한 각 시스템들의 성능이 다양하다는 점이다. 그림 8에서는 또한 같은 조건하에서 제네레이션 크기의 변화에 따라 두 시스템의 성능의 변화를 볼 수 있다. 그래프 상에서 두 시스템간은 약 2.5배정도의 성능의 차이를 보인다. 처리속도가 빠른 시스템에서는 큰 제네레이션 크기가

주어져도 빠른 시간 안에 네트워크 코딩 과정을 처리할 수 있으나 지나치게 큰 제네레이션 크기는 처리속도가 느린 시스템에서 네트워크 코딩 실행시간을 크게 늘릴 수 있고 따라서 전체 네트워크 시스템의 성능을 대폭 떨어뜨리는 결과를 가져올 수 있다. 때문에 다양한 리소스를 가진 각 노드들의 차이를 고려하여 네트워크 시스템내의 제네레이션 크기를 결정해야 한다.

Ⅵ. 결론

본 논문에서는 최근 크게 사용이 증가되고 P2P 시스템에서 사용될 수 있는 네트워크 코딩의 전체적인 성능에 영향을 미칠 수 있는 요소들에 대하여 알아보았다. 네트워크 코딩의 핵심인 인코딩과 디코딩에서 발생하는 성능상의 문제점은 전체 시스템의 성능에 직결되는 문제이다. 때문에 인코딩과 디코딩에서 성능에 영향을 미치는 요소들을 알아보는 것은 네트워크 코딩을 이용한 네트워크 시스템 설계 시 우선시 되어야 할 사항이다. 본 논문에서는 성능에 영향을 줄 수 있는 요소로 유한필드에서의 곱셈연산 알고리즘과 필드 및 제네레이션 크기를 살펴보았고 이들 요소의 영향은 전체 네트워크 코딩의 성능에 크게 영향을 준다는 것을 보여주었다. 특히 제네레이션 크기의 설정은 각 내부 노드뿐 아니라 전체 시스템에 영향을 주는 요소이므로 여러 변수들에 대한 고려 후 결정해야 한다.

참고문헌

〔1〕 C. Gkantsidis, P. Rodriguez, Network Coding for Large Scale Content Distribution, Proceedings of IEEE Infocom. Vol. 4, pp. 2235 - 2245, 2005

〔2〕 B. Cohen, Incentives Build Robustness in BitTorrent, Proceedings of Workshop on Economics of Peer-to-Peer Systems. Berkeley, CA. 2003.

〔3〕 R. Ahlswede, N. Cai, S. Li, R. Yeung, Network Information Flow. IEEE Transactions on Information Theory. Vol. 46, Issue. 4, pp. 1204 - 1216, 2000.

〔4〕 R. Koetter, M. Medard, An Algebraic Approach to Network Coding. IEEE/ACM Transacrions on Networking. Vol. 11, Issue. 5, pp. 782 - 795, 2003.

〔5〕 P. Chou, Y. Wu, K. Jain, Practical Network Coding, Proceeding of Allerton Conference on Communication, Control, and Computing. Monticello, IL. 2003.

〔6〕 T. Ho, R. Koetter, M. Medard, D. Karger, M. Effros, The benefits of coding over routing in a randomized setting. Proceeding of IEEE International Symposium on Information Theory (ISIT). Yokohama, Japan. pp. 442, 2003.

〔7〕 J. Park, M. Gerla, D. Lun, Y. Yi, M. Medard, CodeCast: A Network Coding Based Ad Hoc Multicast Protocol, IEEE on Wireless Communications. Vol. 13, Issue. 5, pp. 76-81, 2006.

〔8〕 C. Gkantsidis, J. Miller, and P. Rodriguez. Anatomy of a P2P Content Distribution system with Network Coding. Proceeding of the 5th International Workshop on Peer-to-Peer Systems (IPTPS 2006), 2006.

〔9〕 U. Lee, J. Park, J. Yeh, G. Pau, M. Gerl,. CodeTorrent: Content Distribution using Network Coding in VANETs. Proceedings of the First International Workshop on Decentralized Resource Sharing in Mobile Computing and Networking (MobiShare), 2006,

〔10〕 J. Guajardo, S. Kumar, C. Paar, J. Pelzl, Efficient Software-Implementation of Finite Fields with Applications to Cryptography, Acta Applicandae Mathematicae: An International Survey Journal on Applying Mathematics and Mathematical Applications, Vol. 93, No. 1-3, pp 3-32, 2006.

〔11〕 P. Montgomery, Modular Multiplication without Trial Division, Mathematics of Computation, Vol. 44, No. 170, pp. 519-521, 1985.

〔12〕 S. Trenholme, AES' Galois field, http://www.samiam.org/galois.html

저 자 소 개

이철우

2005년~현재: 홍익대학교 컴퓨터공학과 재학중

관심분야: 시스템 및 네트워크 기술

박준상

2006년: University of California, Los Angeles 전산학박사

2007년~현재: 홍익대학교 컴퓨터공학과 전임강사

관심분야: 유무선 통신 및 통신망