논문 2005-42TC-5-5

네트워크 프로세서에 적합한 개선된 AntNet기반 라우팅 최적화기법

(Optimized AntNet-Based Routing for Network Processors)

박 현 태*, 배 성 일*, 안 진 호*, 강 성 호**

(Hyuntae Park, Sung-il Bae, Jin-Ho Ahn, and Sungho Kang)

요 약

본 논문은 생태계 군집 시스템을 네트워크 기술에 응용한 적응형 라우팅 알고리즘인 AntNet을 기존의 상용 네트워크 프로세서 기반에서 최적화할 수 있도록 개선된 알고리즘을 제안하는 연구이다. 현재 사용되고 있는 네트워크 프로세서는 단순한 패킷 프로세싱만을 위해 설계되어 AntNet과 같은 복잡한 연산이 필요한 적응형 라우팅 알고리즘을 구현하는데 많은 문제점을 가지고 있다. 이를 분석하고 해결하기 위해 AntNet의 강화인자를 연산하는 부분을 중심으로 적응 성능은 유지하면서도 효율적으로 연산실행시간을 줄일 수 있는 개선된 AntNet알고리즘을 제안하였다. 이를 시뮬레이션을 통해 비교분석함으로서 제안한 개선된 AntNet알고리즘의 효용성을 검증한다.

Abstract

In this paper, a new modified and optimized AntNet algorithm which can be implemented efficiently onto network processor is proposed. The AntNet that mimics the activities of the social insect is an adaptive agent-based routing algorithm. This method requires a complex arithmetic calculating system. However, since network processors have simple arithmetic units for a packet processing, it is very difficult to implement the original AntNet algorithm on network processors. Therefore, the proposed AntNet algorithm is a solution of this problem by decreasing arithmetic executing cycles for calculating a reinforcement value without loss of the adaptive performance. The results of the simulations show that the proposed algorithm is more suitable and efficient than the original AntNet algorithm for commercial network processors.

Keywords : Network Processors, AntNet, Adaptive Routing, Reinforcement Value

I. 서 론

최근 이동성과 공유성에 대한 소비자의 욕구가 증가함에 따라 거대 네트워크 망이 구성되고 연동될 것으로 전망된다. 이러한 거대 네트워크망이 구성됨에 따라 엄청난 호스트를 지원하기 위한 확장성 문제, 급박한 상황에서의 데이터 및 서비스의 생존성과 고가용성을 보장하는 문제, 예기치 못한 추가적 요구나 기능에 대한 적응 문제 등이 발생할 수 있다. 이러한 경우 기존의 네트워크 기술로는 어느 시점 이후로는 이러한 문제에 효율적으로 대처하고 해결하기 어렵다. 따라서 차세대 네트워크 기술은 다양한 네트워크 환경에 대한 적응성과 생존성을 높이는 방향으로 발전할 것이다. 최근에 활발히 진행되고 있는 이러한 기술 연구는 적응성, 생존성을 높이는 네트워크 알고리즘 개발에 주력해 왔다. 그 중 대표적인 것이 스스로 확장을 조절하고 새로운 환경

* 학생회원, ** 평생회원 연세대학교 전기전자공학과
(Department of Electrical and Electronic Engineering, Yonsei University)
접수일자: 2004년5월25일, 수정완료일: 2005년5월11일

에 적응하며 생존하는 메커니즘을 가지고 있는 생태계 군집 시스템을 네트워크 기술에 응용하여 효율적인 네트워크 처리를 위한 적응 생존형 네트워킹 기술인 AntNet이다.[1]

그러나 기존의 AntNet 연구는 하드웨어 기반의 고려가 없이 알고리즘차원에서 연구되어 실제 네트워크에 구현 및 적용하기에는 많은 무리가 따른다. 뿐만 아니라 기존의 네트워크 프로세서는 데이터 패킷 처리의 성능 및 속도 향상에만 주력해 왔기 때문에 복잡한 연산이 필요한 적응형 라우팅 알고리즘을 적용할 때 이 연산부분에서 많은 성능저하를 가져온다. 따라서 이를 해결하기 위해서는 추가적인 알고리즘 연산을 위한 새로운 하드웨어 유닛의 설계 및 구현이 필요하거나 적응생존 모델을 유지하면서 기존 네트워크장비의 아키텍처에 적합하도록 알고리즘의 개선이 필요하다.

본 연구에서는 후자에 대한 논의로서 기존 네트워크 프로세서의 아키텍처를 고려한 AntNet 알고리즘의 개선 및 최적화를 하려고 한다. 실험은 상용 네트워크프로세서로서 널리 이용되고 있는 Intel사의 IXP1200 네트워크 프로세서를 이용하였으며 구현의 문제점 역시 IXP1200 중심으로 논의하겠다.

Ⅱ. 기존 AntNet알고리즘 분석 및 구현의 문제점

1. 기존 AntNet 알고리즘

AntNet은 네트워크의 최적화 문제를 해결하기 위해 군집개체인 개미의 행동패턴을 응용한 분산적, mobile agent 기반의 Monte Carlo시스템이다. 실제 개미의 행동패턴을 보면, 다른 개미들과 페로몬(pheromone)을 통해 의사교환을 하여 최단 이동거리를 찾아 나간다. 이를 응용하여 네트워크 상에 mobile agents(stigmergy 또는 ant)를 통해 수집한 정보를 간접적, 비동기적으로 각 노드에 전달하여 라우팅 문제를 해결하는 적응형 agent기반의 라우팅 알고리즘이다.[1]

각 노드의 라우팅 테이블의 선택 확률값은 역방향 agent에 의해 근원노드로 돌아가면서 갱신된다. 이 때 강화인자(r : reinforcement value) 를 통해 선택 확률값의 변화정도를 결정한다. 강화인자는 agent에 의해 수집된 시간지연 정보를 상대적으로 평가하여 구하며 AntNet의 적응형 성능에 가장 중요한 인자이다. 기존 알고리즘에서 제시하는 강화인자를 구하는 수식은 다음과 같다.

$$r = C_1\left(\frac{W_{BEST}}{T}\right) + C_2\left(\frac{I_{sup} - I_{inf}}{(I_{sup} - I_{inf}) + (T - I_{inf})}\right) \quad (1)$$

W_{BEST}는 순방향 agent가 목적노드까지 진행했던 시간 중에서 일정시간 내의 유효한 값으로 가장 좋은 값, 즉 best cost를 말한다. T 는 현재 agent에 의해 진행된 시간, 즉 current cost를 말한다. I_{sup}와 I_{inf} 는 진행했던 시간들의 평균값과 분산값에 의해 대략적인 신뢰구간의 제한으로서 얻어진 값들이다. C_1과 C_2는 두 부분간의 가중치이다.

각 노드의 라우팅 테이블에서는 이렇게 계산된 강화인자 r에 의해 다음 수식과 같이 갱신된다.

$$\begin{aligned} &P_{fd} \leftarrow P_{fd} + r(1 - P_{fd}) \\ &P_{nd} \leftarrow P_{nd} - rP_{nd} \\ &n, f \in N, n \neq f \end{aligned} \quad (2)$$

N은 이웃노드의 셋을 의미하고 f 는 현재 agent가 지나가고 있는 노드, n은 f 를 제외한 다른 이웃노드를 의미한다. P_{fd}는 라우팅 경로로 결정되어 r에 의해 증가될 f 노드의 확률값이고 P_{nd}는 라우팅 경로를 제외한 감소될 n노드로의 확률값이다. 식(2)에서 보면 agent가 노드 f 를 거쳐 선택한 경로에 대한 확률값은 강화인자에 비례하여 증가하고 f 를 제외한 노드 n를 거치는 경로의 확률값은 r에 비례하여 감소하게 된다. 이러한 과정을 각 agent가 반복하면 많이 선택한 경로의 선택 확률값이 커지게 되면서 최적 경로로 결정한다. 따라서 네트워크의 상태변화가 급변하더라도 agent에 의해 수집된 정보를 바탕으로 적응성이 강한 라우팅이 가능하게 된다.

2. 기존 네트워크프로세서에서 AntNet구현 문제점

가. 변수형 제한

그림 1의 마이크로엔진 아키텍처를 살펴보면 FPU와 같이 소수점을 연산할 수 있는 유닛이 없을 뿐만 아니라 정수형이 아닌 실수형 처리를 위한 고려도 없다. 실제로 IXP1200 마이크로엔진을 프로그래밍하기 위해 이용하는 Microengine-C언어는 오직 정수형만을 데이터

형으로 지원한다.[2] 반면에 기존 AntNet에서는 확률값으로서 0부터 1사이의 실수형을 사용하기 때문에 연산 과정에서도 소수점을 가지고 있는 실수형 타입을 필요로 한다. 따라서 기존 알고리즘의 실수형 연산을 정수형 연산으로 전환하여야 한다.

나. 연산 실행시간의 증가

그림 1에서 IXP1200 프로세서의 패킷처리를 전담하는 마이크로엔진의 구조를 보면 연산유닛 부분이 범용 프로세서에 비하여 상당히 빈약한 것을 알 수 있다.[3][4] 기존의 네트워크프로세서는 패킷의 헤더를 읽고 경로를 지정하는 단순한 패킷 프로세싱만을 빠르고 효율적으로 하기 때문에 복잡한 연산구조가 필요 없었다. 따라서 AntNet과 같은 복잡한 연산을 필요로 하는 라우팅 알고리즘을 그대로 구현하면 패킷 처리시간보다 경로선택을 위한 연산시간이 크게 증가하여 전체적인 라우팅 시간이 크게 증가하게 된다.

기존 AntNet알고리즘을 살펴보면 수식 (1)의 강화인자를 구하는 부분이 주 연산부분이 되는데 평균과 분산값 연산을 해야 하는 두 번째 항이나 임의의 나눗셈을 해야 하는 첫 번째 항 모두 기존의 마이크로엔진구조에서는 많은 연산 실행시간이 소요하게 된다. 따라서 이를 ALU와 Shifter 하나만으로 짧은 연산 실행시간 안에 수행될 수 있도록 개선하는 것이 필요하다. 즉, ALU

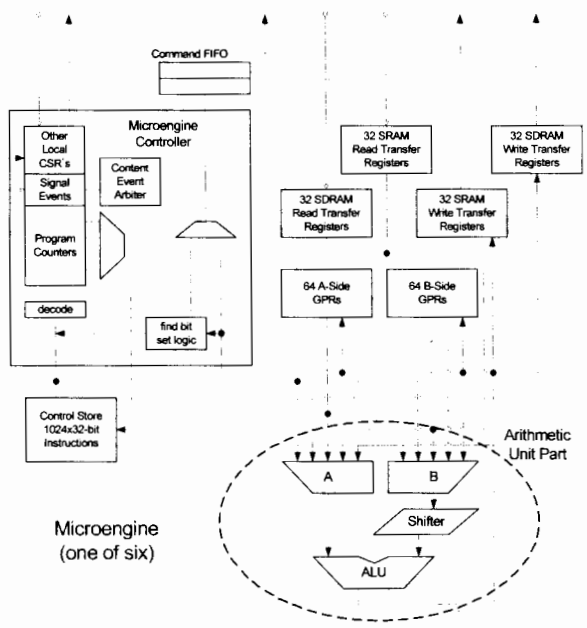


그림 1. Intel IXP1200 마이크로엔진 블록 다이어그램[4]
Fig. 1. Intel IXP1200 Microengine Block Diagram.

를 이용한 덧셈, 뺄셈과 Shifter를 이용한 특정 수(2^n)의 곱셈, 나눗셈으로 기존 알고리즘을 표현할 수 있다면 이 문제를 해결할 수 있을 것이다.

Ⅲ. 제안하는 개선된 AntNet알고리즘

1. 정수형 모델로 변환

가. 시간정보의 정수화

agent가 수집하는 네트워크상의 cost는 시간 지연값으로 갖기 때문에 이러한 시간정보를 하드웨어 처리가 가능한 정수형으로 바꾸어 주어야 한다.

나. cost의 범위제한 및 정수화

agent는 하나의 패킷이기 때문에 시간정보에 너무 큰 범위를 할당하는 것은 패킷 크기가 커지는 문제가 생긴다. agent가 갖는 시간지연이 너무 클 때에는 해당 링크의 패킷 전달이 거의 불가능하므로 지정된 최대값을 할당하여도 될 것이다. 따라서 최대로 지연되는 시간을 한정하고 이에 따라 시간 값을 일정범위의 정수 값으로 할당하여 정수형 cost를 가질 수 있도록 하였다.

다. 경로선택 확률값의 정수화

각 노드에 라우팅 테이블과 지엽적 트래픽 모델을 가지고 있는 기본 구조는 기존 알고리즘과 동일하다. 단 경로 선택의 기준이 되는 0과 1사이의 확률값은 소수를 갖는 실수형이므로 정수형으로 치환해야 한다. 정수로 치환된 확률값의 범위는 클수록 더욱 정밀한 계산을 할 수 있지만 하드웨어 구현 시 계산의 용이성과 처리해야 할 데이터의 크기 등을 고려할 때 8bit범위가 적당할 것이다. 따라서 라우팅 테이블과 지엽적 트래픽 모델에서 갖는 값도 이 범위의 정수값으로 한정한다.

2. 강화인자

강화인자의 연산은 네트워크 상태변화에 대한 AntNet의 적응형 라우팅 성능에 가장 많은 영향을 끼치는 요소이며 가장 많은 연산 실행시간이 필요한 부분이다. 따라서 제안하는 방식의 주안점은 AntNet의 네트워크 적응성능 저하 없이 연산 실행시간을 줄이는데 있다.

먼저 트래픽의 평균과 분산을 이용한 모델을 고려하지 않는다. 이는 기존 알고리즘에서 수식(1)의 강화인자

를 계산하는 부분에서 평균과 분산을 이용한 모델인 두 번째 항이 best cost에 의한 변화요인인 첫 번째 항보다 성능에 더 적은 영향을 미친다는 실험적 결과를 바탕으로 한다.[1] 물론 이에 따라 적응성의 정밀함은 다소 떨어지지만 연산실행의 많은 부담을 줄일 수 있다는 측면에서 충분히 trade-off가 성립한다. 그러므로 수식 (1)은 다음과 같이 간략화 된다.

$$r = norm\left(\frac{bCost}{curCost}\right) \tag{3}$$

즉, 강화인자 r은 $bCost$와 $curCost$의 상대적 비로서 나타낼 수 있다. 이는 의미적으로 가장 좋은 상태와 현재 상태의 상대적 관계를 통해 강화인자의 크기를 결정할 수 있다는 것이다. 그런데 이 역시도 임의의 수로 나눗셈을 하는 방식이다. 앞에서 언급했듯 IXP1200 마이크로엔진에서는 나눗셈알고리즘을 이용하지 않고 반복적인 덧셈 또는 뺄셈으로 이를 수행하므로 실제 구현 시 많은 연산 실행시간이 필요하게 된다. 이를 보완하기 위해 프로그래밍 차원에서 나눗셈알고리즘을 추가하더라도 원하는 수준의 연산 실행시간 감소는 이룰 수 없어 근본적인 해결책은 아니다. 또한 강화인자 r의 크기가 0~1사이의 실수형이 되기 때문에 상대적 비를 통한 나눗셈은 정수형 모델에도 적합하지 않다. 따라서 이 상대적 비를 통해 접근하는 방식 대신 뺄셈을 이용한 절대적 편차를 통한 다음의 접근 방식을 제안한다.

$$r' = 255 + (bCost - curCost) \tag{4}$$

$$r = norm\left(\frac{r' \cdot bCost}{C_{bCost} \cdot C_{curCost}}\right) \tag{5}$$

나눗셈은 두 수의 상대적인 관계를 나타내고 뺄셈은 두 수의 절대적인 관계를 나타낸다. 따라서 절대적인 값에 상대적인 의미를 부여하는 기법이 필요하다. 이러한 기법으로 $bCost$와 $curCost$의 값에 따라 특정 수로 나누고 가중치를 곱하는 방식을 제안한다. 이는 나눗셈의 제수가 일정한 몇 개의 특정한 수라면 임의의 수로 나누는 나눗셈연산보다 간단하고 연산 실행시간도 적기 때문에 이득이라 하겠다. 이러한 개념에서 뺄셈을 통한 절대적 편차를 어떤 조건에 따른 특정한 수로 나누어 상대적 비와 같은 효과를 낼 수 있다.

절대적 편차를 상대적 비와 같은 역할을 할 수 있도록 변환하기 위해 다음에 제안하는 규칙성을 통해 변환된 수식(5)를 제시한다.

· 규칙성 ① : 일정한 기준값에 $bCost$와 $curCost$의 음수 편차를 더해줘야 한다.

편차의 규칙성을 알아보기 위해 $curCost$값을 일정한 값으로 고정하여 영향을 무시하고 오직 $bCost$값과 수식 (4)처럼 변환된 편차 형태인 r'의 관계만을 고려해 보면 비례관계이다. 예를 들어 기존 알고리즘에서 $curCost$ = 100일 때, $bCost$ = 80인 경우와 $bCost$ = 90인 경우를 보면 r은 0.8과 0.9이다. 즉 $curCost$가 일정한 값일 때 $bCost$가 클수록 강화인자 r이 커진다. 따라서 $bCost$와 $curCost$의 편차를 나타낼 때 $bCost$는 양의 부호여야 한다. 즉 편차는 $bCost - curCost$가 된다. 그런데 항상 $curCost$가 $bCost$보다 크거나 같으므로 편차가 일정한 양의 범위를 갖기 위해서는 기준이 되는 상수값이 추가되어야 한다. 따라서 제안된 수식(4)와 같이 255를 기준으로 편차가 작을수록 더 큰 r'을 갖도록 하였다. 255라는 기준값은 실험적으로 적절한 기준값이며 r'의 최대값을 255로 제한하는 의미도 갖는다.

· 규칙성 ② : $bCost$, $curCost$크기에 따라 일정한 가중치 상수값으로 나누어줘야 한다.

$bCost$와 $curCost$의 편차와 강화인자의 관계로 따져보면 $bCost$나 $curCost$가 큰 수에서 형성되면 큰 편차가 나더라도 강화인자는 작은 값을 갖고 반면에 작

표 1. 구간에 따른 C_{bCost}와 $C_{curCost}$ 할당표
Table 1. Assign C_{bCost} and $C_{curCost}$ according to $bCost$ and $curCost$.

구 간	C_{bCost} or $C_{curCost}$
4 ~ 7	2
8 ~ 15	4
16 ~ 31	6
32 ~ 63	8
64 ~ 127	10
128 ~ 255	12
256 ~ 511	14
512 ~1024	16

은 수에서 형성되면 작은 편차가 나더라도 강화인자는 큰 값을 갖는다. 예를 들어 기존알고리즘에서는 $bCost$ = 5, $curCost$ = 10인 경우와 $bCost$ = 50, $curCost$ = 100인 경우 똑같이 r은 0.5이다. 이 경우 편차가 작은 수에서 형성되는 첫 번째 경우는 5로 두 번째 경우의 편차 50보다 훨씬 작은 것을 알 수 있다. 따라서 $bCost$, $curCost$크기에 따라 일정한 가중치 상수값으로 $bCost$와 $curCost$의 편차를 나누어 준다.

나누어 주는 수를 결정하는 방식은 $bCost$, $curCost$ 크기를 일정 구간에 비교하여 해당 구간에 할당되는 값으로 결정한다. 즉 표 1의 기준에 의해 $bCost$값의 구간에 따라 C_{bCost}를 할당하고 $curCost$값의 구간에 따라 $C_{curCost}$를 할당한다. 구간은 큰 수일수록 민감성이 더 적어지므로 더 넓은 범위를 할당하여도 무방하다. 따라서 2^n에 따른 구간으로 나누어 큰 수로 갈수록 더 큰 구간이 되도록 하였고 구간이 증가함에 따라 일정비율로 증가하여야 하므로 2의 배수로 할당 값을 정한다. 2의 배수를 할당함에 따라 두 가중치 $bCost$, $curCost$ 조합에 의해 제수가 결정되는데 하드웨어적으로 볼 때 이 값들은 2, 3, 5로 나누는 특정값 나눗셈기로만 구현되기 때문에 간단히 구현될 수 있다. IXP1200에서는 이러한 특정값으로 나눗셈하는 것도 고려되지 않았기 때문에 여기서 제안한 방식도 한계가 있다. 그러나 큰 임의의 수로 나누는 것보다는 일정하게 작은 수로 여러 번 나누도록 구현하여 연산 실행시간을 줄일 수 있다.

· 규칙성 ③ : 전체적으로 $bCost$ 자체를 가중치로 곱해줘야 한다.

규칙성①에서 $bCost$값과 변환된 편차형태인 r'의 관계만을 고려해 보면 비례관계라고 했었다. 이 때는 편차의 성질을 고려한 것이었지만 r'과 강화인자가 비례관계이므로 $bCost$값과 실제 강화인자의 관계도 역시 비례관계이다. $curCost$값의 영향을 일정한 기준으로 두지 않더라도 편차가 일정할 때를 고려해보면 이 비례관계를 쉽게 파악할 수 있다. 예를 들어 $bCost$ = 1, $curCost$ = 2인 경우와 $bCost$ = 99, $curCost$ = 100인 경우 똑같이 편차는 1이다. 그러나 기존 알고리즘에서 볼 때 강화인자 r값은 첫 번째 경우는 0.5이고 두 번째 경우는 0.99이다. 즉 $bCost$가 클수록 강화인자 r 값도 커지는 비례관계이고 이 영향이 매우 민감한 것을

```
reF : 강화인자 r
C_bCost : bCost 크기에 따라 할당된 가중치
C_curCost: curCost 크기에 따라 할당된 가중치
bCost : 유효시간 중 가장 좋은 cost값
curCost : 순방향 agent로부터 받은 현재 cost값
comptable : 구간에 따른 할당치

int comptable[] = { 2, 4, 6, 8, 10, 12, 14, 16 };
void setReinforcement ( int curCost ) {
      if( ! bCost의 유효시간 체크) {
        bCost reset;
      }
      if ( curCost < bCost ) {
        bCost = curCost;
        reF = r's maximum value;
      }
      else {                          // 2^n구간 비교에 의한 할당
        for(i=1;i=구간수;i++) {
         if( 2^(i+2) < bCost < 2^(i+3) ) C_bCost = comptable[i];
         if( 2^(i+2) < curCost < 2^(i+3) ) C_curCost = comptable[i];
        }
        reF = normal{ (255+(bCost - curCost))*bCost/
C_bCost/C_curCost };
      }
      if (reF>25) reF=25;     // reF 최대,최소값 제한
        else if (reF < 1) reF = 1;
```

그림 2. 제안한 강화인자 연산 알고리즘
Fig. 2. Pseudo Code for Suggested Reinforcement r Calculating.

알 수 있다. 따라서 곱해주는 가중치는 자세한 값일수록 더욱 좋은 성능을 보였다. 따라서 규칙성②에서 구간에 따라 나누는 수를 일정 수로 한정한 것과 달리 $bCost$값 자체를 곱하는 가중치로 사용하였다. 곱셈의 경우, 나눗셈과 달리 임의의 수에 대한 연산도 Booth Algorithm과 같은 곱셈알고리즘을 사용하면 상대적으로 작은 연산 실행시간을 가지므로 연산 실행시간이 크게 증가하는 문제는 발생하지 않는다.

이와 같이 제안한 알고리즘의 강화인자를 구하는 부분을 pseudo code로 나타내면 그림 2와 같다. 먼저 표1의 할당할 가중치 값을 저장해 놓고 $bCost$가 유효한지 체크한다. 이 후 넘겨받은 $curCost$과 비교하여 $curCost$가 더 좋은 값이면 $bCost$를 대체하고 아니면

표 1에 따라 비교하여 가중치를 할당한다. 이렇게 구해진 가중치를 가지고 강화인자를 구한다. 이렇게 구해진 강화인자가 최대, 최소범위 안의 값인지를 확인하고 범위를 넘으면 최소치 1, 최대치 255로 한정한다.

3. 라우팅테이블 갱신

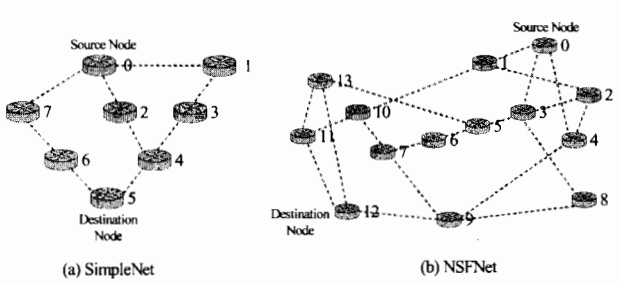
각 노드의 라우팅테이블은 agent에 의해 수집된 라우팅 정보와 강화인자에 의하여 갱신된다. agent가 선택했던 경로를 통해 거쳐가는 인접한 노드 f로는 강화인자에 따라 확률값을 증가시키고 나머지 인접 노드 n으로는 강화인자에 따라 확률값을 감소시킨다. 이를 수식으로 표현하면 수식(6)과 같다. 이는 확률값의 범위가 0~255의 정수형으로 전환된 형태일 뿐 의미나 역할은 기존 알고리즘의 수식 (2)와 같다.

$$
\begin{aligned}
&P_{fd} \leftarrow P_{fd} + (r \times (255 - P_{fd})/256) \\
&P_{nd} \leftarrow P_{nd} - (r \times P_{nd}/256) \\
&n, f \in N, n \neq f
\end{aligned} \tag{6}
$$

Ⅳ. 성 능 평 가

기존 알고리즘과 비교했을 때 제안한 개선된 알고리즘이 적응적이고 효과적으로 라우팅 경로선택을 하는지 실험을 통해 검증하였다. 기존 알고리즘은 PC상에서 C++를 이용하여 시뮬레이션하였고 개선된 알고리즘은 IXP1200상에서 Microengine-C를 통해 구현한 후 PC상에서 비교하였다.

비교실험을 위한 네트워크 구조는 라우팅 알고리즘 검증에 많이 사용되는, 단순한 모델인 SimpleNet과 인터넷의 전신인 백본망, NSFNet를 모델링하여 사용하였다. 그림 3과 같이 SimpleNet은 8개의 노드와 9개의 링크, NSFNet은 14개의 노드와 21개의 링크로 구성된다.



(a) SimpleNet (b) NSFNet

그림 3. 실험을 위한 네트워크 구조
Fig. 3. Network Topologies for the experiment.

네트워크 상황은 다음 4가지 경우에 따라 라우팅 소요시간, 즉 각 링크의 cost를 가정하였다.

- Light : 모든 링크의 트래픽 상황이 균일하여 홉 cost와 동일한 시간적 cost 상황
- Biasing : 홉 cost가 많은 특정 경로에 더 작은 시간 cost를 할당하여 특정 경로의 라우팅 소요시간이 작은 상황
- Heavy : Light와 같이 모든 링크의 트래픽 상황이 균일하지만 전체적으로 Light상황보다 10배 더 트래픽이 가중된 상황
- Dynamic : Light, Biasing, Heavy한 상황을 주기적으로 변화하여 네트워크 상황이 가변적인 상황

실험의 평가 기준은 각 노드의 라우팅 테이블 내 경로선택을 위한 확률값의 변화 추이이며 cost 10이 경과될 때 마다 새로운 agent를 생성 및 포워딩되도록 하였고 네트워크 상의 agent수는 30개로 제한하였다.

시뮬레이션 그래프는 agent에 의해 근원노드의 라우팅 테이블이 갱신되는 횟수를 x축으로 하고 근원노드에서 인접노드로 경로를 선택하는 확률값을 y축으로 나타내었다. 상대적으로 확률값이 크게 나타나는 노드를 통한 경로가 최적경로임을 나타낸다.

1. SimpleNet

근원노드를 0, 목적노드를 5으로 가정했을 때 0-1-3-4-5, 0-2-4-5, 0-7-6-5, 3개의 경로가 존재한다.

트래픽 상황이 Light인 경우, 0-2-4-5 경로와 0-7-6-5 경로가 동일한 cost를 갖으며 가장 좋은 경로이다. 기존 알고리즘의 경우, 두 좋은 경로가 cost차이가 없기 때문에 서로 경합하는 양상을 보이고 있고 0-1-3-4-5 경로는 제외되는 것을 알 수 있다. 개선된 알고리즘에서도 이러한 그래프 패턴을 뚜렷하게 나타남을 확인할 수 있었다. 단 경합하는 두 경로가 동등하게 경합되지 않고 어느 한쪽으로 쏠리는 경향이 있는데 이는 트래픽의 평균, 분산모델을 적용하지 않았기 때문에 랜덤한 선택이 누적되어 나타난 것이다. 그러나 경합된 두 경로가 모두 최적의 경로라면 어느 경로를 선택하든 라우팅 성능에는 영향을 주지 않는다.

트래픽 상황이 Biasing인 경우, 0-2-4-5 경로와

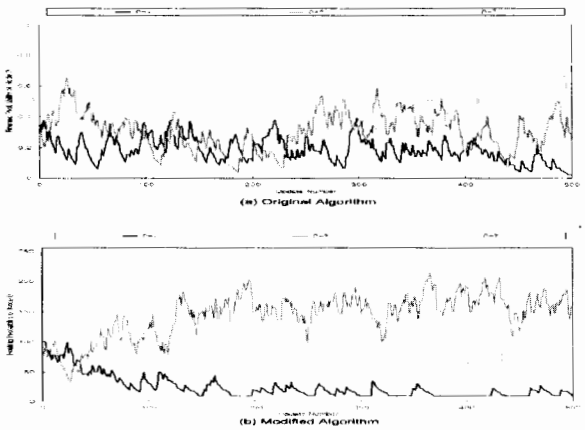


그림 4. SimpleNet 실험결과 : Light 상태
Fig. 4. SimpleNet Simulation Result : Light State.

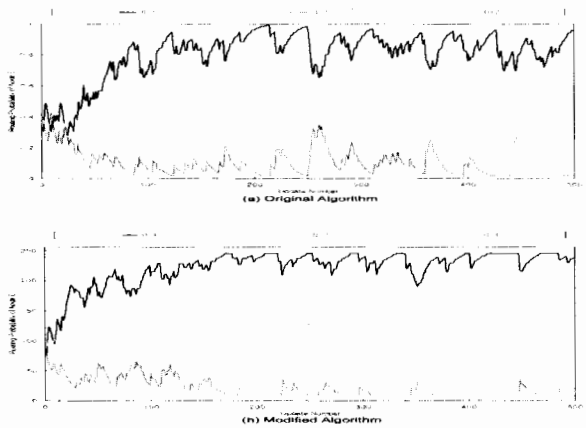


그림 5. SimpleNet 실험결과 : Biasing 상태
Fig. 5. SimpleNet Simulation Result : Biasing State.

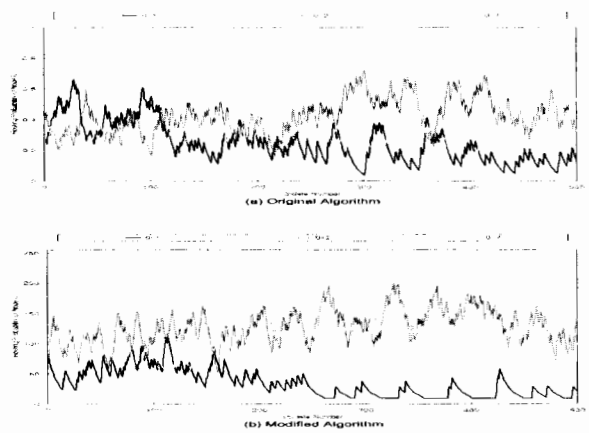


그림 6. SimpleNet 실험결과 : Heavy 상태
Fig. 6. SimpleNet Simulation Result : Heavy State.

0-7-6-5 경로보다 0-1-3-4-5경로가 더 라우팅 시간이 짧기 때문에 더 나은 경로이다. 이처럼 최적 경로가 확연히 존재할 때는 빠르게 그 경로를 라우팅 최적경로로

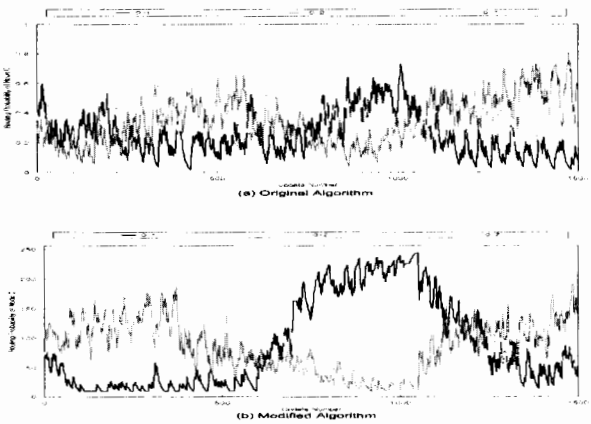


그림 7. SimpleNet 실험결과 : Dynamic 상태
Fig. 7. SimpleNet Simulation Result : Dynamic State.

결정해야 한다. 기존 알고리즘과 마찬가지로 개선된 알고리즘에서도 0-1-3-4-5경로를 비교적 적은 갱신시간 안에 확연히 결정하는 것을 볼 수 있다.

트래픽 상황이 Heavy인 경우, Light인 경우와 동일하게 트래픽이 균일하기 때문에 비슷한 그래프 패턴이 보여야 한다. 그림 6의 그래프에서 이를 확인 할 수 있다. 따라서 절대적인 트래픽 양과 상관없이 상대적인 각 링크의 트래픽 차이에 의해 경로선택이 이루어짐을 검증할 수 있었다.

트래픽 상황이 Dynamic인 경우, 갱신되는 횟수가 500회마다 주기적으로 세 가지 상황을 변화시켜 네트워크 상황 변화에 따른 라우팅 적응성을 확인한다. Light와 Heavy인 구간에서는 0-2-4-5 경로와 0-7-6-5 경로가 경합하고 0-1-3-4-5 경로가 상대적으로 선택 제외되는 양상을 보였다. 반면에 Biasing인 구간에서는 0-2-4-5 경로와 0-7-6-5 경로가 상대적으로 선택 제외되고 0-1-3-4-5경로로 수렴되는 양상을 보였다. 이는 기존 알고리즘에서 요구되는 것과 같이 개선된 알고리즘에서도 이런 양상을 보였다. 단 개선된 알고리즘에서는 전체적인 평균과 분산모델을 통해 완만히 적응하지 않기 때문에 랜덤한 선택에 의한 영향이 크게 나타난다. 그래서 어느 한 경로가 다른 경로에 비해 뚜렷한 우위가 있을 때 그 경로로 빠르게 수렴하려고 하는 성질을 보였다. 따라서 명확히 최적경로가 발생한 Biasing 구간에서는 기존 알고리즘보다 더 빠른 시간 내에 더 많은 선택 확률값의 차이를 보였다. 이처럼 최적 경로로 존재하고 이를 결정하는 면은 개선된 알고리즘이 더 나은 성능을 보였다.

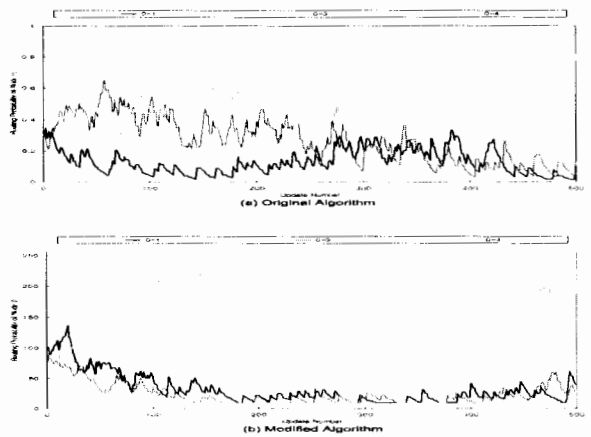


그림 8. NSFNet 실험결과 : Light 상태
Fig. 8. NSFNet Simulation Result : Light State.

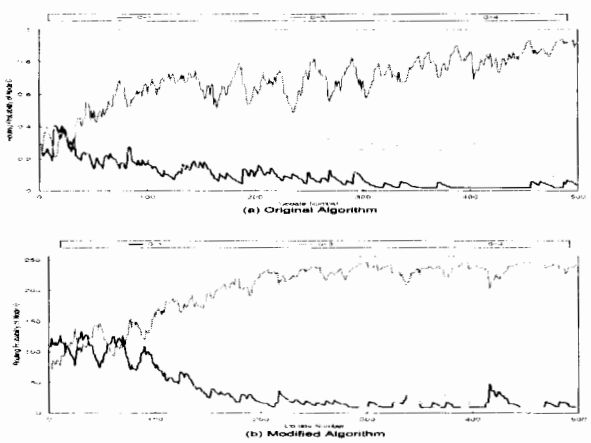


그림 9. NSFNet 실험결과 : Biasing 상태
Fig. 9. NSFNet Simulation Result : Biasing State.

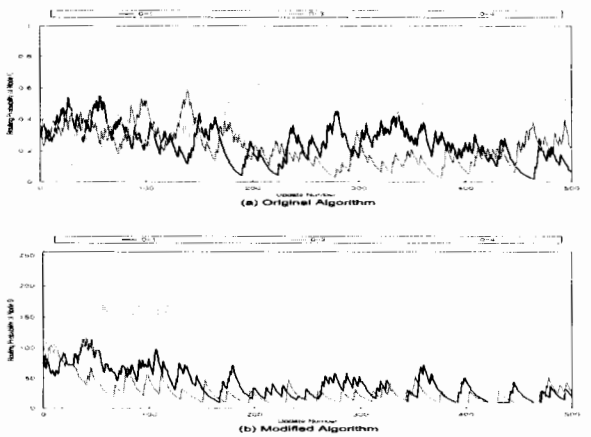


그림 10. NSFNet 실험결과 : Heavy 상태
Fig. 10. NSFNet Simulation Result : Heavy State.

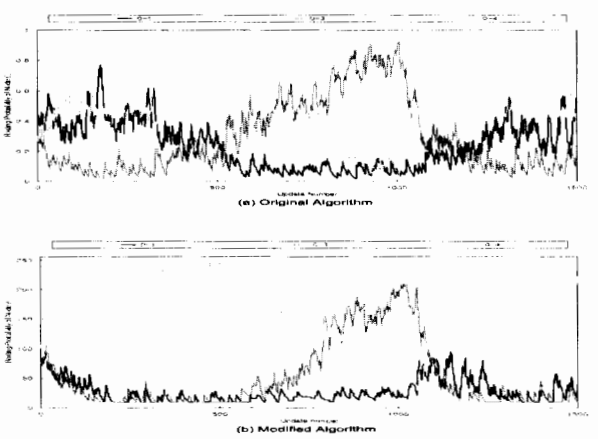


그림 11. NSFNet 실험결과 : Dynamic 상태
Fig. 11. NSFNet Simulation Result : Dynamic State.

2. NSFNet

NSFNet은 SimpleNet에 비해 더 많은 노드와 링크가 있지만 0-4-9-12의 최단경로가 존재한다. 따라서 각 링크에 트래픽이 균일한 Light와 Heavy한 경우 다른 경로와 cost차이가 크기 때문에 SimpleNet에 비해 더 뚜렷하게 0-4-9-12 경로를 최적 경로로 선택하는 것을 확인할 수 있다. 앞 서 SimpleNet의 Biasing상황에서 지적했던 것처럼 뚜렷한 최적 경로가 존재하는 경우에는 기존 알고리즘 보다 제안한 개선된 알고리즘이 더 빠르고 뚜렷하게 경로 선택하는 것을 확인 할 수 있었다. 또한 그 선택 확률값이 차이가 확연히 많이 났다. 이는 기존 알고리즘에 비해 개선된 알고리즘이 최적경로에 대해 더 신뢰하고 많은 패킷을 이 경로를 통해 보낼 것이라는 것을 의미한다. 만약 이로 인해 한 경로에 트래픽 과부하가 걸릴 수도 있으나 알고리즘에 따라 다시 적응하게 되므로 오히려 성능적 우위를 가져온다. 따라서 개선된 알고리즘은 이러한 개선된 알고리즘의 성능적 우위는 NSFNet의 네 가지 상황 모두에서 확인할 수 있었다.

이상에서와 같이 제안한 알고리즘을 다양한 가상 네트워크 시뮬레이션 환경에서 기존 AntNet알고리즘과 비교 분석한 결과 유사한 적응 성능을 보임과 함께 최적경로 선정에서는 더 나은 성능을 보이는 것을 확인할 수 있었다.

V. 결 론

앞으로 예상되는 네트워크 망의 확대와 트래픽 증가를 해결하기 위해서는 AntNet과 같은 적응형 라우팅 방식의 도입이 필요하다. 그러나 본 연구를 통해 지적

한 바와 같이 현재 사용되는 상용네트워크프로세서에서 기존 AntNet알고리즘을 구현하는 것은 많은 문제점을 안고 있었다.

특히 대부분의 연산 실행시간의 증가는 강화인자 r를 연산하는 부분에서 일어난다. 따라서 AntNet의 적응, 생존 능력을 최대한 유지하면서도 연산 실행시간이 적은 개선된 강화인자의 도출 알고리즘을 제안하였다. 기존 AntNet의 연산 실행시간 증가가 평균, 분산모델의 적용과 임의의 나눗셈에 의한 것이므로 적응성에 영향이 적은 평균, 분산모델을 고려하지 않고 임의의 나눗셈에 의한 상대적 수치를 뺄셈에 의한 절대적 수치로 전환하였다. 이 때 발생하는 의미적 차이를 $bCost$, $curCost$크기에 따라 일정한 가중치 상수값으로 나누어 주고 전체적으로 $bCost$ 자체를 가중치로 곱해줘서 보정하였다. 이를 통해 임의 수의 나눗셈를 고정된 간단한 수의 나눗셈과 임의 수의 곱셈으로 변환하여 연산 실행시간를 줄일 수 있다.

참 고 문 헌

[1] G. Di Caro and M. Dorigo, "AntNet: Distributed Stigmergetic Control for Communications Networks," *Journal of Artificial Intelligence Research 9*, pp. 317-365, Dec. 1998.

[2] Intel Corp., *Intel Microengine C Compiler Language Support*, Intel Press., December 2001.

[3] Intel Corp., *Intel IXP1200 Network Processor Family Hardware Reference Manual*, Intel Press, December 2001.

[4] Erik Johnson and Aaron Kunze, *IXP1200 Programming*, Intel Press, 2002.

[5] Intel Corp., *Intel IXP1200 Network Processor Family Microcode Programmer's Reference Manual*, Intel Press, December 2001.

[6] Intel Corp., *Intel Microengine C Compiler LIBC*, Intel Press, December 2001.

[7] Intel Corp., *Intel IXP1200 Network Processor Family Developer Workbench*, Intel Press, December 2001.

[8] J. F. Kurose and K. W. Ross, *Computer Networking*, Addison-Wesley, 2002.

저 자 소 개

박 현 태(학생회원)
2004년 연세대학교 전기전자 공학과 학사 졸업.
2005년 현재 연세대학교 전기전자 공학과 석사 과정.
<주관심분야 : SoC 설계 및 응용>

배 성 일(학생회원)
1998년 경북대학교 전자공학부 학사 졸업.
2000년 연세대학교 전기전자 공학과 석사 졸업.
2005년 현재 연세대학교 전기전자 공학과 박사 과정.
<주관심분야 : SoC 설계 및 응용>

안 진 호(학생회원)
1995년 연세대학교 전기공학과 학사 졸업.
1997년 연세대학교 전기공학과 석사 졸업.
2002년 LG전자 DTV연구소 선임연구원.
2005년 현재 연세대학교 전기전자공학과 박사 과정.
<주관심분야 : SoC 설계 및 응용, DFT, NOC>

강 성 호(평생회원)
1986년 서울대학교 제어계측 공학과 학사 졸업.
1988년 The University of Texas, Austin 전기 및 컴퓨터 공학과 석사 졸업.
1992년 The University of Texas, Austin 전기 및 컴퓨터 공학과 박사 졸업
1992년 미국 Schlumberger Inc. 연구원
1994년 Motorola Inc. 선임 연구원
2005년 현재 연세대학교 전기전자공학과 교수
<주관심분야 : SoC 설계 및 응용, DFT, SoC 테스트>