논문 01-26-12C-1 한국통신학회논문지 '01-12 Vol.26 No.12C

루프 방지를 위한 칼라 스레드 알고리즘의 개선

정회원 김 한 경*

Improvement of Colored Thread Algorithm for Loop Prevention

Han-Kyoung KIM* *Regular Member*

요 약

Ohba에 의해 제안된 칼라 스레드 알고리즘은 루프 경로 설정을 방지하기 위한 새로운 접근 방법으로서 효과적으로 인식되고 있다. 칼라 스레드 알고리즘에서는 스레드의 상태를 null, colored, transparent의 3 종류로 정의함에 따라, 동작의 내부 알고리즘이 복잡한데, 이를 4개 상태로 확대하여 유한상태 기계를 개선하였다. 또 루프 경로를 도출하기 위하여 스레드가 일단 중복이 되면 다시 새로운 칼라를 생성하여 다시 확장하고, 그것이 다시 접수되면 unknown 홉 카운트를 갖는 스레드로 재차 확장하는 방식을 기존의 칼라 스레드를 그대로 사용케 함으로써 네트워크 운행 횟수를 최소한 1회 절약되도록 하였으며 이벤트를 재정의하여 기존의 라우팅 프로토콜과의 결합이 용이하도록 하였다.

ABSTRACT

Colored thread algorithm suggested by Ohba is understood to be effective to prevent from establishing path in loop. 3 kinds of thread state, such as null, colored, transparent, defined in colored thread algorithm which makes difficult and complex to design actions of the thread events, is reconfigured and improved to a 4 states to make the finite state machines simple. The original colored thread algorithm finds loops by receiving thread of a same color that was generated by the initiator itself and confirms the loops by hop count of unknown thread traversed again along the looped path. This method is modified as to use the same color instead of generating new colored thread for the looped path, leading to save the traversing time. And also redefined the events to make easy coupling with existing protocols.

Ⅰ. 서 론

MPLS(Multi-protocol Label Switching) 네트워크에서 라우팅 정보를 변경하거나, 장애 등에 의해 라우팅 정보의 변경이 이루어지는 동안에, 일시적으로 네트워크에 존재하는 노드들 사이에 각기 저장된 라우팅 정보가 일치하지 않아 설정을 하려는 경로에 루프가 형성될 수 있다. 루프의 형성은 사용자가 발생시킨 데이터 혹은 제어 정보들이 목적지에 도착하지 못하고, 네트워크의 노드 사이를 돌아다니면서 네트워크 자원을 낭비 시키는 현상을 초래한다. 기존의 라우팅 프로토콜에서는 경로의 설정 시 루프를 허용하는 한편, 자원의 낭비를 최소화하기 위하여 패킷 헤더에 TTL(Time-to-Live) 값을 지정하고 이 값을 네트워크의 노드를 통과할 때마다 1씩 감소시켜 값이 0에 이르면 패킷을 제거하는 방법으로 루프를 허용하면서 영향을 완화 시키는 방식을 택하였다[1-3]. MPLS 망에서 이 문제를 해결하기 위하여 경로 벡터(path-vector/diffusion) 방법과 칼라 스레드(Colored thread) 방식의 알고리즘을 제안하고 있는데, 전자가 RIP 또는 RSVP에서 전통적으로 적용해 오던 방법임에 비해 후자는 이전에는 제안

* 창원대학교 컴퓨터공학과
논문번호: 010080-0426, 접수일자: 2001년 4월 26일
※ 본 연구는 2000년도 창원대학교 교내연구비로 수행됨

된 바가 없으며 그래서 실제 적용 사례가 없는 전혀 새로운 알고리즘이다[4-6].

MPLS 노드에 의해 LSP(Label Switched Path)를 설정하는 것은 소스 노드에 의해 경로 정보를 명세화하는 명시적 라우팅(explicit routing)과 계층3의 네트워크 라우팅 기능을 이용하여 다음 홉을 찾아가는 hop-by-hop 방법이 있다[6]. 특히, 후자는 LDP(label Distribution Protocol) 메시지를 사용하여 경로상의 각 노드에서 해당 FEC(Forwarding Equivalence Class)에 대한 레이블 바인딩 정보를 인접 노드와 교환하는데, 결과로 각 노드들은 LIB (Label Information Base)를 유지하게 된다. LIB를 구성할 때, 입력 링크의 state에 대하여 대응되는 출력 링크의 state를 정적으로 명시하는 방법과, 여러 입력 링크의 각 state들을 하나의 outgoing link state로 병합하는 방법이 있는데, VC-병합 기능을 활용하여 scalability를 향상 시키는 후자의 방법이 상대적으로 복잡하며, 특히 동시에 multiple state merging의 경우 특별한 주의가 요구된다[8-14].

Ⅱ. 칼라 스레드 알고리즘

2.1 개요

스레드란 LSP를 설정하기 위하여 ingress 노드에서 시작하여 다운스트림 방향의 노드들에게로 전달된 메시지의 열(sequence)이다[6,7]. 스레드 메커니즘은 ingress LSR이 특정 FEC에 대하여 다음 홉으로 새로운 칼라 스레드를 생성하여 다운스트림 방향으로 확장한다(extend). 스레드가 계속 다운스트림 방향으로 확장되면 결국에는 egress LSR에 도달한다. 스레드가 확장되는 과정에서, 스레드의 칼라가 이미 이전에 방문한 스레드의 칼라와 동일하면, 현재의 탐색된LSP에 루프가 형성되어 있다고 판단한다. 경로에 루프가 없음이 판단되면, egress LSR에서 ingress LSR까지 스레드를 역방향으로 되감는다(rewind). 스레드 되감기를 통해 레이블을 할당한다.

칼라 스레드의 확장은 다운스트림 방향의 노드들을 홉 카운트가 증가하는 방향으로 배열한다. 만일 동일 FEC에 대한 스레드가 존재할 때 새로운 칼라 스레드가 접수되면 스레드를 기존의 출력 링크로 병합한다(merge). 이 때 홉 카운트가 출력 링크 홉 카운트보다 크면, 다운스트림 방향으로 홉 카운트를 변경해야 하지만 그렇지 않은 경우는 병합만 이루어진다.

스레드의 칼라가 기존의 스레드와 동일한 경우에는 루핑이 형성된 것으로 판단하여 스레드를 가두고(stall), 해당 노드는 새로운 칼라 스레드를 생성하여 홉 카운트를 unknown으로 세트한 다음 다운스트림으로 확장한다. unknown으로 확장되는 스레드의 운행에 따라 루핑 경로를 도출할 수 있다.

Ohba는 이러한 과정의 수행을 위해 스레드의 동작을 스레드 확장, 스레드 병합, 스레드 가두기, 스레드 되감기, 스레드 철수의 6가지로 정의하고, 이러한 동작을 트리거 시키는 이벤트로는 received thread, next hop acquisition, rewound, withdrawn, next hop loss, reset to unknown을 [7]에서 상태 천이표를 통해 제시하였다. 특히 이러한 이벤트가 발생할 경우에 수행되는 동작에 의해 다음 이벤트가 발생할 때까지의 상태를 Null, Colored, Transparent의 세 가지로 정의하였다.

2.2 Ohba의 칼라 스레드 알고리즘

Ohba의 칼라 스레드 알고리즘에서 사용하는 기호는 다음과 같이 정의한다.

Hmax: 입력 링크의 홉 카운트 중 가장 큰 값

Ni: unstalled 입력 링크의 수

[7]에서 Ohba가 요약하여 제시하는 알고리즘은

1. 노드에서 다음 홉이 필요하면, 칼라 스레드를 생성하여 다운스트림으로 확장한다.
2. 확장중인 스레드가 다음 홉을 찾지 못하면, 노드는 스레드를 철수한다. 스레드의 철수로 인하여 노드가 취할 수 있는 동작은, 다음 홉으로부터 스레드를 철수하거나, 새로운 다음 홉으로 새로운 스레드를 확장하는 동작이다.
3. 칼라 스레드를 접수하면 취할 수 있는 동작은 가두기(stalled), 병합(mergcd), 되감기(rewound), 확장(extended)이다. TTL 값이 0이면 확장하지 않는다.
4. 수신된 스레드를 노드가 가두기를 시켜야 될 때, 노드가 leaf가 아니면서 Ni = 0이면, 스레드의 철수를 시작한다. 그러나 Ni > 0이면서 수신된 스레드의 홉 카운트가 unknown이 아니면, unknown 값의 홉 카운트를 갖는 칼라 스레드를 생성하여 확장한다. 만약 수신된 스레드 홉 카운트가 unknown이면, 스레드 확장은 없고, 더 이상의 동작을 취하지 않는다
5. 확장 중이던 스레드를 rewind할 때에, 만약 스레드의 홉 카운트가 Hmax보다 1이상 크면, 홉 카운트 값이 (Hmax+1)인 투명(transparent)한 칼라의 스레드를 다운스트림으로 확장한다.
6. 투명한 칼라의 출력 링크를 가진 노드에 투명한 칼라의 스레드를 접수하는 경우에, Hmax의 값이 감소된다면, 그 노드는 칼라의 변경 없이 다운스트림으로 그 스레드를 확장한다

그림 1. Ohba의 칼라 스레드 알고리즘.

(그림 1)와 같다.

2.3 Ohba의 유한상태기계

FSM에서 사용하는 주요 변수의 정의는 다음과 같다.

Hout : 다음 홉에 대한 출력 링크의 홉 카운트

Hrec : 수신 스레드의 홉 카운트

Ohba가 [7]에서 제시하는FSM은 Null, Colored, Transparent 세 개의 상태를 가진 구조로 제시되어 있다. Null은 출력 링크가 존재하지 않으며 Ni = 0 인 상태이다. Colored 상태는 노드가 다음 홉으로 연결되는 출력 링크를 통하여 칼라 스레드가 확장 중임을 의미하고, transparent 상태는 그 노드가 egress 노드이거나 출력 링크가 설정되었음을 의미한다. Ohba는 이런 상태를 바탕으로 상태의 천이를 일으키는 이벤트를 received thread, next hop acquisition, rewound, withdrawn, next hop loss, reset to unknown와 같이 정리하였다. 기본적인 동작으로는 extending, merging, stalling, rewinding, withdrawing으로 제시하였으나 FSM에서 실제의 기본적인 동작으로 존재하지 못하고 있다. 특히 Ohba는 FSM에서 가드 조건으로 부울 값을 갖는 3개의 플래그를 운용한다. 칼라(CL) 플래그는 접수한 스레드의 칼라 상태, 루프(LP) 플래그는 스레드의 루프 형성 여부, NL 플래그는 새로운 링크(NL)를 통하여 접수된 스레드일 때 세트 된다.

Ⅲ. Ohba 알고리즘의 개선

3.1 알고리즘의 개선점

Ohba는 첫째, 스레드가 일단 생성이 되면 상태를 colored란 단일 상태로 정의하여, 칼라 스레드 내부의 상태를 정의하지 않는다. 이것은 서로 다른 칼라 스레드의 상호 작용을 동작 내부에서 처리하고, 각 이벤트에 대하여 extend, stall, merge, withdraw 동작을 한꺼번에 고려해야 하기 때문에 CL, LP, NL 플래그를 운용하는 등 스레드의 동작 알고리즘이 복잡하다. 둘째, Ohba 알고리즘은 스레드의 병합 동작에서 새로운 칼라 스레드를 생성한다. 스레드의 병합은 동일 FEC에 대해 발생하므로, 새로운 칼라의 스레드를 생성하지 않고 홉 카운트만을 조정함으로써 논리적인 문제를 해결할 수 있다. 셋째, Ohba의 알고리즘에서는 가두기를 하는 한편 unknown 홉 카운트를 갖는 새로운 칼라 스레드를 생성 및 확장하여 루프를 도출한다. 이것을 next hop acquisition NACK 이벤트가 접수될 때 대안의 다음 홉을 탐색하도록 알고리즘을 보완해주면 별도의 동작과 상태를 정의하지 않아도 되며, (그림 2)의 4번 항목에서 제시한 알고리즘과 비교할 때 1차례의 루프 운행을 절약할 수 있다.

알고리즘의 개선을 위하여 next hop acquisition, rewind, withdraw, next hop loss, next hop acquisition NACK, updated와 같은 6가지의 기본적인 이벤트를 정의한다. 이벤트를 접수하여 노드가 취할 행동은 stall 동작을 제외하고, Ohba의 extend, merge, rewind, withdraw의 분류를 따른다. 6가지의 이벤트는 프로토콜의 기본 메시지를 고려하여 기존 프로토콜과 결합이 되도록 한다[10]. 이것은 Ohba의 received thread, next hop acquisition, rewound, withdrawn, next hop loss, reset to unknown의 이벤트로 정의할 경우 기존 프로토콜과의 호환성이 쉽지 않음을 고려한 것이다.

(그림 1)에 의해 요약된 Ohba의 칼라 스레드 알고리즘을 위에서처럼 이벤트를 다시 정의하고 이들 이벤트에 대한 동작 알고리즘을 3.2 절에서부터 3.6 절까지 이벤트 별로 동작 알고리즘을 제시한다.

3.2 next hop acquisition

업스트림 노드의 next hop acquisition에 의해 스레드가 확장된다. 이 이벤트는 leaf 노드 또는 중간 노드에서 새로운 경로의 탐색이 필요할 때, 다음 홉으로 발생시킨다. ingress 노드에서는 동일 FEC (egress 노드)에 대하여 경로가 설정된 바가 없으면, 새로운 칼라 스레드를 생성하고 다음 홉을 결정한다. 다음 홉의 탐색 여부에 따라 다운스트림으로 next hop acquisition 이벤트, 업스트림으로 next hop acquisition NACK이벤트를 트리거 할 수 있다 (그림 2).

새로운 칼라 스레드를 접수하면 동일한 egress 노드의 다른 스레드의 존재를 확인하고, 존재하면 스레드를 병합한다. egress 노드에서는 MPLS 네트워크 종단 처리를 수행하는 프로세스로 대치한다. 스레드의 확장은 H_{rec}보다 1만큼 큰 값을 H_{out}으로 결정하여 다운스트림으로 확장한다. 만일 동일한 칼라의 스레드가 존재하고 $H_{rec} > H_{out}$인 경우에는 루프가 형성된 것으로 인정한다. 이 프로세스는 새로운 칼라 스레드를 생성하지 않으며, 다운스트림으로 홉 카운트 변경 메시지를 송신하지 않아도 된다. 홉 카운트의 변경은 update 이벤트에 의해 단순히 홉 카운트가 (H_{max} + 1)의 값으로 변경된다.

```
for the next hop acquisition event
    if there is no thread or link going to the same egress
node(FEC)   /* null state */
        then if next hop is found
                then { initiate outgoing link to next hop;
                       extend the thread;
                       change state to extending;
                     }
                else { send next hop acquisition NACK to
upstream node;
                       terminate TCB;
                     }
        else if exist same colored thread
                then withdraw the thread; /* found looped
path */
                else
                     switch (state of the thread) {
                         case "extending" :
                              if Hrec < Hout
                                then {   merge the thread
to outgoing link;
                                         change state to
merging;
                                     }
                                else {   send next hop
acquisition NACK to upstream node;
                                         remove TCB;
                                     }
                         case "transparent" :
                              if Hrec < Hout
                                then {  merge the thread to
            outgoing link;
                                         change state to
transparent;
                                         rewind the thread;
                                     }
                                else {   update hop count
to downstream node;
                                         merge the thread;
                                         rewind the thread;
                                     }
                     }
```

그림 2. next hop acquisition 이벤트의 동작

3.3 rewind

스레드가 egress LSR에 도달하면 역방향으로 되감기를 한다. rewind 이벤트에 의해 되감기를 하는데 레이블 정보의 교환과 함께 스레드의 상태가 투명하게 변경된다(그림 3). 스레드의 상태를 투명하게 함으로써 경로가 설정되었음을 나타낸다.

```
if not exist colored thread for the rewind event
        then stop rewinding
        else
          switch(state of thread)
               case "extending" : {
                    send rewind to upstream node;
                    set thread state to transparent;
                    }
               case "merging" :  {
                    send rewind to each upstream node;
                    set thread state to transparent;
                    }
```

그림 3. rewind 이벤트에 대한 동작

3.4 withdraw

withdraw는 양방향 이벤트인데, 업스트림 노드로부터의 접수는 무조건적으로 경로 해제 작업을 수행하며, 다운스트림 노드로부터 접수될 때, H_{rec}가 unknown이면 업스트림으로 경로 해제 동작을 진행하며, known 값이면 확장한 다운스트림 경로를 해제하고 새로운 다음 홉으로 확장한다(그림 4).

```
for the thread on the withdraw event
if event is received from downstream node
   then {
      switch (state of the thread)
         case "extending" :
              if (Hrec = unknown)        /* 무핑 경로 */
                then {
                     send withdraw to upstream node;
                     remove TCB;
                     }
                else {   /* 새로운 next hop을 찾아 extend
*/
                     create new colored thread;
                     update hop count of outgoing link;
                     send next hop acquisition to alternative
downstream node;
                     }
         case "transparent" : {
              if (Hout = unknown)
                then do nothing;
                else {
                     send withdraw to upstream node;
                     update link state to release
                     remove TCB;
                     }
              }
         }
   else {          /* withdraw event form upstream node */
        send withdraw to downstream node;
        update link state to release;
        remove TCB;
        }
```

그림 4. withdraw 이벤트에 대한 동작

기 형성된 LSP에 장애가 발생하면 스레드의 철수를 요청하고 새로운 경로의 설정을 추진한다. 장애의 보고에 따라 next hop loss 이벤트를 자체적으로 발생시켜 기존의 다운스트림 LSP를 해체하는 한편, 자신의 노드에서부터 새로운 next hop을 찾아서 새로운 경로의 설정을 시도하던가, 아니면 새로운 next hop을 결정할 수 없을 때에는 업스트림 노드로 withdraw 이벤트를 트리거한다.

3.5 next hop loss

경로에 장애가 발생하면 자체적으로 next hop loss 이벤트를 발생시킨다. 이 이벤트는 다운스트림 방향의 스레드나 링크 정보를 해제시킨다(그림 5). 다음 홉을 변경할 경우에는 이벤트를 2가지로 분리

```
 for all of thread corresponding to the link  where the next
hop loss event generated
    if alternative next hop not exist
       then {
            send withdraw to upstream node with hop count
"unknown";
            remove TCB;
          }
       else {
            send next hop loss to downstream node;
            create new colored thread;
            extend thread to new next hop of downstream;
          }
```

그림 5. next hop loss 이벤트에 대한 동작

하는데, 기존의 다음 홉에 대해서는 next hop loss 이벤트를 발생시키고, 새로운 다음 홉에 대해서는 next hop acquisition 이벤트를 발생시킨다. 출력 링크와 관련되는 모든 LSP에 대하여 처리한다.

3.6 next hop acquisition NACK

노드 객체가 next hop acquisiton 이벤트를 접수하여 해당 FEC를 송수신할 다음 홉이 존재하지 않는 경우에는 업스트림으로 next hop acquisition NACK 이벤트를 발생한다. 임의의 한 노드가 스레드를 확장한 홉으로부터 next hop acquisition NACK를 접수하면, 또 다른 다음 홉이 존재하는지 확인하고, 존재하면 그 홉으로 스레드 확장을 계속한다. 만일 어떠한 다음 홉도 선택할 수 없다면, 이전 홉으로 next hop acquisition NACK 이벤트를 트리거 시킨다(그림 6).

```
for the thread on the next hop acquisition NACK event
  if alternative next hop not exist
    then {
          send withdraw to upstream node;
          remove TCB;
         }
    else {
          extend thread to new next hop of downstream;
         }
```

그림 6. next hop acquisition NACK 이벤트에 대한 동작

Ⅳ. 스레드 상태

4.1 상태의 개선

스레드 상태 null과 transparent는 초기 및 최종 상태이므로 이를 제외하면, 스레드의 진행과정은 colored라는 단일 상태로 표현되어 스레드의 상태로서 의미를 갖지 못한다. 이를 extending, merging 상태로 정의한다. 상태는 스레드의 상태와 링크의 상태로 구별하는데, 링크는 물리적으로 구축된 LSP별로 상태를 관리하기 위한 것으로 initialization, hello, alive 메시지에 의해 상태가 관리된다. 반면 스레드는 경로가 설정될 때까지의 상태를 관리하기 위한 것으로, 경로가 설정되면 링크 정보에 의해 경로가 유지된다.

extending 상태는 스레드 확장을 요구한 다음부터 새로운 이벤트가 발생할 때까지의 상태를 의미한다. merging 상태는 스레드 확장 요구에 의해 확장을 하는 중에 동일 FEC에 대한 LSP가 설정되어 있거나 또는 스레드가 이미 확장되고 있다면, 새로운 스레드를 기존의 스레드나 LSP에 병합 시키고자 하는 상태를 의미한다. 병합된 스레드는 기존 스레드가 되감기할 때 함께 되감기를 수행한다.

withdraw 상태는 상태로 정의가 가능하기도 하지만 transient 상태로 간주하였다. withdraw 상태를 정의하여 FSM이 동작하는 것을 설명하는 것이 용이할 수 있지만 이벤트 자체가 의미가 있을 뿐 이벤트의 처리 결과는 기존 상태로 남아있던지 아니면 null 상태로 돌아가기 때문이다. 각 상태의 천이 과정을 (표 1)에 나타내었다.

이를 지원하는 스레드 제어 블록은 3개의 객체로 정의한다. 스레드 정보 베이스, 입력 링크 베이스, 출력 링크 베이스로 구성되며 상호 포인터에 의해 연결된다. 스레드 정보 베이스는 FEC 별로 참조되며, 각 스레드의 상태 정보와 칼라 정보를 기록한다. 스레드 칼라가 투명하면 해당 LSP는 설정된 상태이며, 이 때는 링크 상태에 따라 트래픽 서비스가 제공된다. 스레드 상태는 LSP 설정 관리에 이용된다. 입력 링크 베이스는 이웃 노드의 주소, 링크의 상태, 홉 카운트, 레이블, M-플래그로 구성된다. 링크에 대한 주기적인 Hello 메시지 처리를 하며, 그에 따른 링크 상태를 유지한다. 각 LSP별 레이블 정보가 보관되어 출력 링크로의 레이블 스와핑이 이루어질 수 있다. 입력 링크와 동일한 애트리뷰트를 갖는 출력 링크 베이스는 단지 C-플래그에 의해

표 1. 칼라 스레드의 유한상태기계

	null		extending		merging		transparent	
	action	next state	action	next state	action	next state	action	next state
next hop acquisition	initiate	extending merging	withdraw	(loop)	X	X	X	X
rewind	X	X	rewind	transparent	rewind	transparent	X	X
withdraw	X	X	withdraw	null	withdraw	null	withdraw	null
next hop loss	X	X	X	X	X	X	withdraw and extend	null extending
next hop acquisition NACK	X	X	extend /NACK	-	extend NACK	null	X	X

연결 여부를 나타내는 것만 틀릴 뿐 데이터 유형은 동일하다.

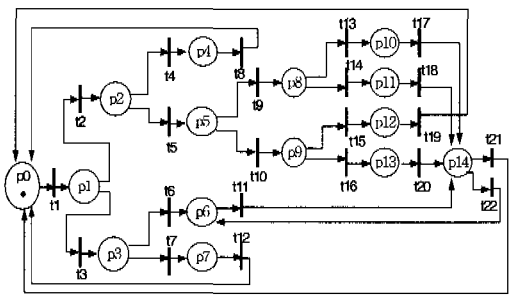

그림 7. 칼라 스레드 알고리즘의 페트리네트 모델

4.2 알고리즘 검증

개선된 알고리즘의 검증을 위한 페트리 네트 모델은 (그림 7) 및 (표 2)와 같으며, 이 모델에 대한 시뮬레이션을 통해 liveness, deadlock,reachability에 대한 문제가 없음을 확인하였다. 페트리 네트 시뮬레이터는 Visual Object Net ++ Version 2.0 (Evaluation)을 사용하였다. 이 모델의 초기 플레이스는 p0가 되며, p4에서 루프가 확인되는데, 이 플레이스에서 루프 회피 또는 루프 방지를 위한 동작을 수행할 수 있다. 각 노드에서 루프가 없는 정상적인 경로 탐색이 이루어지는 경우는 p0 → t1 → p1 → t3 → t6 → p6 → t11 → p14 → t21 → p0의 순서로 트랜지션이 일어나게 된다. 만일 이미 설정된 경로에 대해 확장 요구를 받으면(p2), 기 존재하는 스레드의 칼라를 점검하여(t4, t5), 새로운 칼라(p5)이면 스레드가 투명한지(t9) 아니면 확장 상태인지(t10) 구별하여 처리한다. 투명한 상태이면 입출력 링크의 홉 카운터 값에 따라 최대 홉 카운터 값을 변경하거나(p10), 스레드를 병합한다(p11). 다음 홉으로의 확장 요구를 접수하였을 때 만일 다음 홉으로의 링크가 존재하지 않는 경우(t7)에는 스레드를 제거하고 경로가 없음을 이전 홉으로 통보한다(t12). 페트리 네트 모델에서 동일한 칼라의 스레드가 접수되면(t4), 루프 경로가 형성된 것으로 판단하여 루프 제거를 위한 동작을 수행한다(p4). 이때 루프 방지 동작에 자원의 낭비를 방지하는 로직이 있는 경우에는 루프를 허용할 수도 있으며, 기존 칼라의 스레드와 병합 작업을 수행하도록 하는 것도 가능하다. 이는 구현할 때에 선택사항으로 결정할 수 있다.

V. 결 론

표 2. 플레이스와 트랜지션

플레이스			
번호	의미		
p0	*null state*		
p1	internal condition checks		
p2	color of the thread		
p3	nexthop searching		
p4	thread in loop		
p5	new colored thread		
p6	extending state		
p7	unavailable route		
p8	thread in transparent state		
p9	thread in extending state		
p10	hop count update		
p11	*merging thread*		
p12	remove TCB		
p13	merging state		
p14	transparent state		
트랜지션			
번호	의미	번호	의미
t1	next hop acquisition event	t12	withdraw&remove TCB
t2	exist threads for the FEC	t13	Hrec ≥ Hout
t3	1st thread for the FEC	t14	Hrec < Hout
t4	same colored thread	t15	Hrec ≥ Hout
t5	new colored thread	t16	Hrec < Hout
t6	next hop exist	t17	rewind event
t7	next hop not exist	t18	rewind event
t8	loop found(withdraw)	t19	withdraw
t9	thread in transparent state	t20	rewind event
t10	thread in extending state	t21	withdraw event
t11	rewind event	t22	next hop loss event

Ohba에 의해 제안된 칼라 스레드 알고리즘은 경로 설정에서 이전에는 전혀 제안된 적도 없는 새로운 접근 방법이며, 루프의 검출이나 방지에 효과적이고 새로운 알고리즘으로 인식되고 있다. 그러나 알고리즘에서 제시하는 FSM은 설계나 구현을 하는데 몇 가지 애매모호한 점이 있어 이에 대한 개선을 시도하였다.

첫째는, 스레드의 상태를 null, colored, transparent로 정의하였지만 사실상 colored란 단일 상태를 유지함으로써, 경로의 탐색 과정과 설정 과정에서 나타나는 시차적인 스레드의 상태를 표현하지 못하고, 또 이벤트에 대한 atomic 동작이 표현되지 않고, 그 동작에서 홉 카운트 점검과 세 가지 플래그를 설정하고 점검해야 하는 복잡성이 나타나는데, 이것을 extending, merging 상태로 세분하였다. 둘째는, 동일 FEC에 대하여 각 스레드의 특성을 칼라에만 초점을 맞추어 알고리즘을 제시함으로써, $H_{rec} > H_{out}$ 인 경우에는 새로운 칼라를 생성하고 이 칼라 스레드를 통하여 경로의 중복이 확인되면 다시 홉 카운

트를 unknown으로 세팅하고 루핑을 다시 한번 허용하였다. 이것을 새로운 칼라의 생성은 ingress LSR에서만 허용하고, core LSR에서는 기존의 칼라를 이용한 루프 검출을 수행하여 두 번의 노드 방문으로 가능하도록 하였다. 셋째, 기본적인 이벤트를 next hop acquisition, rewind, withdraw, next hop acquisition NACK, next hop loss으로 재조정하여 기존의 프로토콜과도 결합이 가능하도록 하였다.

개선된 알고리즘을 지원하는 FSM을 페트리 네트로 모델링을 하여 시뮬레이션하고 liveness, reachability, deadlock의 존재 등에 대하여 검증하였으며, 결과 구현이 가능한 것으로 판단하였다. 향후 이를 프로토콜에 적용할 수 있도록 구현 모델을 설계하고 적용 방안의 제시가 필요할 것으로 생각된다.

참 고 문 헌

[1] Bruce Davie and Yakov Rekhter, "MPLS Technology and Applications", *Morgan Kaufmann Publishers*, 2000.

[2] Y. Rekhter, B. Davie, D. Katz, E.Rosen, G. Swallow, "Cisco Systems' Tag Switching Architecture Overview", *IETF RFC* 2105, Feb.,1997.

[3] B. Davie, P. Doolan, Y. Rekhter, "Switching in IP Networks", *Kaufmann Publishers*, 1998.

[4] B. Davie, Y. Rekhter, E. Rosen, A. Viswanathan, V. Srinivasan, "Use of Label Switching with RSVP", *IETF Draft* draft-ietf-mpls-rsvp-00.txt

[5] E. Rosen, Y. Rekhter, D. Tappan, D. Farinacci, G. Fedorkow, T. Li, A. Conta, "MPLS Label Stack Encoding", *IETF RFC* 3032, Jan., 2001.

[6] Y. Ohba, "Issues on Loop prevention in MPLS Networks", *IEEE* Communication Magazine Dec., 1999.

[7] Y. Ohba, Y. Katsube, "MPLS Loop Prevention Mechanism", *IETF RFC* 3063, Feb., 2001.

[8] E. Rosen, A. Viswanathan, R. Callon, "Multiprotocol Label Switching Architecture", *IETF RFC* 3031, Jan., 2001.

[9] R. Callon, N. Feldman, A. Fredette, G. Swallow, A.Viswanathan, "A Framework for Multiprotocol Label Switching", *IETF Draft* draft-ietf-mpls-framework-03.txt

[10] L. Andersson, P. Doolan, N. Feldman, A. Fredette, B. Thomas, "LDP Specifica-tion", *IETF RFC* 3036, Jan., 2001.

[11] David McDysan, Darren Spohn, "ATM Theory and Applications", *McGraw-Hill, Signature edition*

[12] B. Davie, J. Lawrence, K. McCloghrie, E. Rosen, G. Swallow,Y. Rekhter, P. Doolan, "MPLS using LDP and ATM VC Switching", *IETF RFC* 3035, Jan., 2001.

[13] D. Awduche, J. Malcolm, J. Agogbua, "Requirements for Traffic Engineering Over MPLS", *IETF RFC* 2702, Sep., 1999.

[14] B. Jamoussi, "Constraint Based LSP Set up using LDP", *IETF Draft* draft-ietf-mpls-cr-ldp-01.txt

김 한 경(Han-Kyoung Kim) 정회원

1973년 : 서울공대 공학사

1987년 : 충북대 전산학 석사

1995년 : 충북대 전산학 박사

1983년~1997년 : 한국전자통신연구원 책임연구원

1997년~현재 : 창원대학교 컴퓨터공학과 부교수

<주관심 분야> ATM, MPLS, SDR, 소프트웨어공학