칼라 스레드 알고리즘을 이용한 네트워크 도달성 검사☆

Improvement of colored thread algorithm for network reachability test

김 한 경* 이 광 휘**
Han-kyoung KIM Kwang-Hui LEE

요 약

레이블 교환 망의 경로 설정을 위해 제안된 칼라 스레드 알고리즘을 패킷 교환 망에 적용하기 위하여 알고리즘 보완이 필요하다. 칼라 스레드 알고리즘은 스레드의 동작을 extend, rewind, stall, withdraw, merge로, 상태를 null, colored, transparent의 3종류로 정의하였지만 여기에 merged상태를 추가하였다. 또 루프 경로를 도출하기 위하여 스레드가 일단 중복이 되면 다시 새로운 칼라를 생성하여 unknown 홉 카운트를 갖는 스레드로 재차 확장하는 방식인데, 이를 수정하여, 업스트림 방향의 rewind 동작을 소스 노드에 의하여 다운스트림 방향으로 작동하도록 하였다. 스레드가 중복으로 접수되면 merge 동작에서 홉 카운트가 오름차 순이면 스레드의 병합을 시행하고, 그렇지 않는 경우에는 우선 스레드의 칼라가 투명해지거나 제거될 때까지 나중의 스레드를 stalling 시키도록 함으로써, unknown 홉 카운트를 갖는 새로운 칼라의 스레드 생성을 제거하였다.

ABSTRACT

Colored thread algorithm, suggested to be used for the label switching network, needs to be modified for the packet switching network. In this paper, it is recommended to add a merged state, besides the 3 states - null, colored and transparent - which are resulted from the behaviors of extend, rewind, stall, withdraw and merge events. The original colored thread algorithm is designed to generate a new thread and extend it to the downstream direction with unknown hop count when the thread has revisited the node that was visited. It also suggested rewinding the thread to the downstream direction by the source node, instead of rewinding it upstream direction by the revisited node. If a node received multiple threads which had a same forward equivalent class, then it checks first whether the hop counts are ascending or not. If it is in ascending order, then threads are merged. Otherwise the later thread is stalled until the former thread's color is to be changed to transparent or it is removed. This idea removes the effort of generating a new thread with unknown hop count.

☞ KeyWords : colored thread algorithm, loop avoidance, reachiability

1. 서 론

루프 경로가 설정되는 것을 허용하는 TCP/IP 라우팅 프로토콜은 네트워크 자원 고갈 현상을 방지하기 위하여. 패킷 헤더의 TTL(Time-to-Live) 필드를 이용한다. 이는 TTL 값이 0이 되면 패킷을 네트워크에서 제거하는 방법으로, 루프 형성에 따른 영향을 완화 시키는 방식이다[1]. MPLS(Multi-Protocol Label Switching) 망에서는 경로설정 방법으로 경로 벡터(path-vector/diffusion) 방법과 칼라 스레드(Colored thread) 알고리즘이 제안되어 있는데, 전자가 RIP 또는 RSVP에서 적용해 오던 방법인데 비해 후자는 새롭게 제안되어 실제 적용 사례가 없는 알고리즘이다[2]. 본 논문은 TCP/IP 망에서 칼라 스레드 알고리즘을 이용하여 도달성 점검을 수행하는 경우, 알고리즘의 변화와 도달성 시험 기능의 문제점을 검토한다.

[3,4]에서 스레드란 경로 설정을 위하여 ingress 노드에서 시작하여 다운스트림 방향의 노드들에게로 전달된 메시지의 열(sequence)을 의미한다. 칼라 스레드 알고리즘은 근원지 노드에서 목적지

* 정 회 원 : 창원대학교 컴퓨터공학과 교수
hkim@changwon.ac.kr(교신저자)
** 정 회 원 : 창원대학교 컴퓨터공학과 교수
khlee@changwon.ac.kr
[2008/11/07 투고 - 2008/11/12 심사(2009/03/20 2차) - 2009/04/14 심사완료]
☆ 이 논문은 2008년도 창원대학교 연구비에 의하여 연구되었음

노드로 가기 위하여 다음 홉이 필요하면, 칼라 스레드를 생성하여 다운스트림으로 확장(extend)한다. 확장중인 스레드가 다음 홉을 찾지 못하면, 노드는 스레드를 철수(withdraw)한다. 만일 한 노드가 이미 확장 중인 스레드를 가지고 있으면, 스레드의 상태에 따라 스레드를 병합(merge)하거나, 또는 확장이 완료될 때까지 가두기(stall)를 한다. 목적지 노드가 스레드 확장 메시지를 접수하면 역방향 즉, 업스트림 방향으로 되감기(rewind)를 하여 경로가 정상적으로 찾아졌음을 알려준다. 즉, 경로상의 임의의 노드가 취할 수 있는 동작은, 스레드의 확장, 철수, 가두기, 병합, 되감기이다.

2. 알고리즘의 수정

TCP/IP 프로토콜 망에 칼라 스레드 알고리즘을 적용은 순방향과 역방향 경로가 일치하지 않으므로 모든 정보는 소스 노드에서 확산되도록 한다. 기본적으로 경로 상에 위치하는 중간 노드들은 홉 카운트 값을 갖는데, 칼라 스레드의 확장은 목적지 네트워크 방향의 노드들의 홉 카운트가 증가하는 방향으로 배열된다. 만일 동일한 목적지 네트워크에 대한 스레드가 존재할 때, 새로운 칼라 스레드가 접수되면 새로운 스레드를 기존의 인터페이스 경로에 병합한다. 이 때 홉 카운트가 인터페이스 경로의 홉 카운트보다 크면, 다운스트림 방향으로 홉 카운트를 변경하고, 그렇지 않은 경우는 병합만 이루어진다. 그러나 새로운 스레드의 칼라가 기존의 스레드 칼라와 동일한 경우에는 경로에 루핑이 형성된 것으로 판단하여 스레드를 가두고(stall), 소스 노드에게 루프가 형성되었음을 통보하여, 소스 노드가 다운스트림 방향으로 각 노드들에게 스레드의 철수(withdraw)를 요구한다. 이는 루핑이 감지되면 홉 카운트가 “unknown”인 새로운 칼라 스레드를 생성하여 다운스트림 방향으로 확장하는 원래의 알고리즘을 변경한 것이다.

알고리즘 동작은 next hop acquisition, rewind, withdraw, merge, stall 이벤트에 의해 수행된다. 다음 이벤트가 발생할 때까지 이벤트의 동작에 의해 처리된 상태를 원래의 알고리즘과 동일하게 Null, Colored, Transparent의 세 가지로 정의하였다. 모든 프로시저들은 IP 계층에서 수행이 되며 자연스럽게 스레드 메시지들을 중계하여 주는 것을 기본으로 하였다. 소스 노드가 처리하여야 할 기능을 그림 1에 나타내었다.

```
begin          //reachability diagnosis
  request from application command:
    search routing table for the target network address.
    if found next hop then {
      search InterfacePath.LColor(routing table pointer)
for the target network address
      if thread pointer not exist then {
        set color with IP address and unique number
        set Hrec = 1
        set TTL to MAX
        send EXPAND
        save (color, Hop_count, TTL) to InterfacePath
        }
      else if InterfacePath.LColor(routing table pointer)
is not transparent then MERGE
        else quit diagnosis
  request by receiving rewind request message:
    send REWIND
  request by receiving withdrawal request:
    send WITHDRAW
end    // end of reachability diagnosis
```

(그림 1) 백그라운드 프로세스

각 노드의 라우팅 테이블에는 목적지 정보와 함께 스레드 정보 블록의 포인터를 운용한다. 그림 2에 입력 패킷에 포함되는 스레드 정보의 구조와 목적지로 향하는 인터페이스 별로 만들어지는 인터페이스 스레드 정보를 보여주는데, 칼라, 홉 카운터, TTL 및 링크 상태 정보로 이루어진다. 이 정보는 접수되는 스레드 정보와 비교하여 알고리즘의 운행 여부를 결정한다.

```
struct    Thread {
  unsigned  int      Color
  unsigned  int      Hrec
  unsigned  int      TTL
}

struct    InterfacePath[ ] {        //from routing table
  unsigned  int      LColor
  unsigned  int      Hop_count
      enum   State   {null, colored, merged, transparent}
}
```

(그림 2) 데이터의 구조

3. 네트워크 도달성 검사 알고리즘

3.1 EXPAND

next hop acquisition이벤트에 의해 스레드가 확장된다. 각 노드에서 새로운 경로를 탐색할 때, 다음 홉으로 이 이벤트를 전송한다. 이벤트가 접수되면 라우팅 테이블에서 목적지 네트워크를 검색하여 엔트리가 존재하면, 다음 홉을 결정하고 칼라 스레드를 확장한다. 목적지 네트워크가 존재하지 않으면, 중간 노드의 경우에는 가두기를 수행하거나, 소스 노드에 withdrawal request를 전송한다(그림 3).

새 스레드가 접수되었을 때 기존 스레드가 확장 중이면(colored state), 홉 카운트를 비교하여 순차적이면 스레드를 병합하고, 아니면 상태가 바뀔 때까지 대기한다(stall). 대기는 기존 스레드의 상태가 바뀌는 것에 따라 새로운 스레드의 동작을 결정하기 위한 것이다. 그러나 기존 스레드가 투명한 상태이면, 새로운 스레드를 병합 시키고 소스 노드에게 이를 통보하여 다운스트림 방향으로 rewind 시킨다.

```
procedure EXPAND          //next hop acquisition}
  search routing table for the target network address.
  if (target address = my address) then send rewind
request to source node  // Ack
  if not found next hop then send withdrawal request
to source node          // Nack
  else
    search InterfacePath.LColor(routing table pointer)
    if (InterfacePath.LColor = Thread.Color) then
LOOP_DISPLAY    // loop detected
    else
      case (InterfacePath.LColor) {
        null:
          set Hrec = Hop_count +1
          set TTL = TTL – 1
          if TTL = 0 then stop diagnosis.
          send next_hop_acquisition          //EXPAND
          save (color, Hop_count) to InterfacePath
          Change InterfacePath.State to colored
        colored:
          if Hrec < Hop_count then MERGE
          else STALL
        transparent:
          MERGE
          rewind to source node
      }                              //end of case
end       //EXPAND
```

(그림 3) next hop acquisition 동작

3.2 REWIND

MPLS에서는 스레드가 egress LSR(Label Switching Router)에 도달하면 역방향으로 되감기를 한다. rewind에 의해 되감기를 하면서, 레이블 정보를 교환하고 스레드의 상태를 투명으로 변경한다. 스레드의 상태가 투명함은 경로가 설정되었음을 나타낸다. 그러나 TCP/IP 프로토콜 스위트에서는 순방향과 역방향의 경로가 일치하지 않으므로, 경로가 설정이 되면 소스 노드에게 경로 상태를 통보하여, 소스 노드로 하여금 다운스트림 방향으로 칼라를 투명하게 바꾸도록 재차 확장을 시도하도록 rewind를 재정의하여 동작 방식을 다시 구성한다.

EXPAND 동작에서 목적지 노드는 소스 노드에게로 rewind request를 전송하고, 소스 노드는 이를 접수하면 다시 다운스트림 방향으로 rewind 이벤트를 전송함으로써 중간 노드들에게 경로 설정이 성공적임을 알려주어 칼라를 투명하게 바꾸어주면서 홉 카운트 점검도 동시에 수행한다. 아울러 NACK 응답에 대한 clear 동작을 수행하도록 확장하는 것도 가능하다. 각 노드에서는 칼라를 점검하여 투명 상태이면 병합된 것이므로 다운방향 확장은 중지한다. 홉 카운트가 논리적이지 않으면 에러 처리를 한다.

```
procedure REWIND      //make the path transparent
  case (received address) {
    source node:
      set Hrec = Hop_count
      send rewind              //REWIND
      change InterfacePath.LColor to transparent
    target address:
      stop threading           // stop job
    default:
      case (InterfacePath.LColor) {
        null:
          error processing type 1   //stop the job
        colored:
          if Hrec < Hop_count then {
            set Hrec = Hop_count
            send rewind
            change InterfacePath.LColor to transparent }
          else error processing type 2   // thread clear
        transparent:
          if (State = merged) then stop threading
          else error processing type 1       // stop the job
        }                          //end of case LColor
  }    //end of case (received address)
end     // REWIND
```

(그림 4) rewind 동작

3.3 MERGE

그림 5에서 보는 바와 같이 동일 목적지를 갖는 새로운 스레드를 접수하면, 경로 상태를 merged로 천이 시키고, 칼라에 따라 처리한다. 칼라가 확장 중이면 투명하게 바뀔 때까지 대기하였다가, 새로운 스레드의 소스 노드에게 rewind 요구를 한다. 물론 기 존재한 스레드의 칼라는 다운스트림 방향으로 투명하게 변할 것이다. 반면 병합 당시 기존 스레드의 칼라가 투명하다면, 새로운 스레드의 소스 노드로 rewind 요구를 하고 다운스트림 방향으로는 필요한 경우에 홉 카운트를 오름 순으로 재배열하게 된다.

```
procedure MERGE        //merging
  set InterfacePath.State merged
  case (InterfacePath.LColor) {
    colored:
      while (InterfacePath.LColor = transparent)
        rewind to source node of the additional
thread
    transparent:
      rewind to source node of the additional thread
      if Hrec ≥ Hop_count then update Hop_count to
downstream node
  }                          //end of case
end    //MERGE }
```

(그림 5) merging 동작

3.4 WITHDRAW와 STALL

MPLS에서는 next hop acquisition에 의해 해당 FEC(forward Equivalent Class)의 다음 홉이 존재하지 않으면 업스트림으로next hop acquisition NACK 이벤트를 전송하고, next hop acquisition NACK를 접수한 노드는 또 다른 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. 그러나 패킷 교환에서는 업스트림 방향의 경로 해제와 같은 동작은 필요치 않고 소스 노드에서부터 다운스트림 방향으로 경로 정보 해제와 같은 동작이 필요하다. 따라서 철수 동작은 소스 노드로부터 withdraw를 접수하면 무조건적으로 스레드 해제 작업을 수행한다(그림 6).

한편 확장 중인 스레드가 존재할 때, 기존의 스레드의 홉 카운트 보다 더 큰 홉 카운트 값을 갖는 새로운 스레드가 접수되면, 새로운 스레드를 가두기(stall)하고, 기존의 스레드 칼라가 투명하게 바뀌면 스레드 확장 동작에 의해 다운스트림 방향의 홉 카운트 업데이트가 이루어진다.

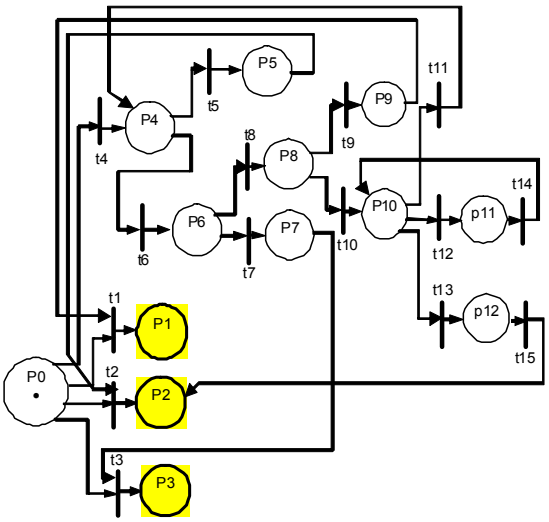
```
procedure WITHDRAW       //path not found
  clear correspond InterfacePath
end    //WITHDRAW

procedure STALL    //칼라 상태에서 큰 홉 카운트를 접수
  while (InterfacePath.State = transparent)
    EXPAND
end    //STALL}
```

(그림 6) withdraw 동작

4. 알고리즘 검증

알고리즘의 검증을 위한 페트리 네트 모델은 그림 7 및 표 1과 같으며, 이 모델에 대한 시뮬레이션을 통해 교착상태와 도달성에 문제가 없음을 확인하였다. 이 모델의 초기 플레이스는 p0가 되며, p5에서 루프가 확인되는데, 이 플레이스에서 루프 회피 또는 루프 방지를 위한 동작을 수행할 수 있다. 각 노드에서 루프가 없는 정상적인 경로 탐색이 이루어지는 경우는 p0 → t4 → p4 → t6→ p6 → t8 → p8 → t10 → p10→ t11 → p4→ t5 → p5→ t2 → p2의 순서로 트랜지션이 일어난다. 만일 이미 설정된 경로에 대해 확장 요구를 받으면(t4), 먼저 목적지 도달 여부를 점검하고(t5, t6), 목적지에 도달한 경우에는 경로가 찾아졌음(p2)을 확인한다. 목적지에 도달하지 못한 경우(t6)에는 다음 홉이 존재하는지 점검(t7, t8)하여 존재하지 않는 경우에는 스레드 철수에 의한 경로가 존재하지 않음(p3)을 확인한다. 다음 홉이 존재하면 인터페이스의 칼라를 점검하여(t9, t10), 같은 칼라(t9)이면 목적지로 가는 경로에 루프가 형성되어 있음(P1)을 알 수 있다. 인터페이스의 칼라와 틀리면(t10) 인터페이스의 상태에 따라(t11, t12, t13)

확장(p4), stall(p11), merge(p12) 작업을 수행한다. merge 처리는 스레드와 링크의 홉 카운터 값에 따라 홉 카운트를 갱신한다.



(그림 7) 단순화된 칼라 스레드 알고리즘의 페트리 네트 모델

(표 1) 플레이스와 트랜지션

플레이스		트랜지션	
번호	의미	번호	의미
p0	Reachiability test start	t1	received withdrawal signal(same color found)
p1	Loop found	t2	received rewind signal
p2	Path found	t3	received withdrawal signal(no next hop)
p3	Path not found	t4	next hop acquisition
p4	Next hop acquisition	t5	reached to destination
p5	Rewind	t6	unreached to destination
p6	expand	t7	not exist next hop
p7	withdraw	t8	exist next hop
p8	Expand thread	t9	same color with interface
p9	Same color with interface	t10	different color with interface
p10	Thread state check	t11	null state of interface
p11	stall	t12	colored state of interface
p12	merge	t13	transparent state of interface
		t14	changed state from colored to transparent
		t15	update hop count

5. 결 론

칼라 스레드 알고리즘은 경로 설정 시에 발생하는 루프의 검출 및 방지에 대하여 새로운 접근 방법이다. 그러나 이 알고리즘은 MPLS 망과 같은 레이블 교환망을 위한 알고리즘으로 패킷 교환 방식의 망에 적용하기 위해서는 개선이 필요하다.

Ohba는 스레드가 일단 생성이 되면 상태를 colored란 단일 상태로 정의하고, 칼라 스레드 내부의 변화를CL, LP, NL 플래그를 운용하였다. 이 알고리즘을 패킷 교환방식에 적용하기 위하여 스레드의 확장과 확장에 따르는 되감기 동작을 소스 노드에 의해 트리거되도록 하였으며, 스레드 병합 문제도 병합이 발생할 경우 새로운 칼라 스레드의 생성과 “unknown” 홉 카운트의 생성을 통하여 칼라 스레드에 의한 경로를 재차 운행하면서 홉 카운트를 “unknown”으로 변경하는 동작을 제거하였다. 이러한 동작의 제거를 보완하기 위하여 병합 동작에서는 접수한 홉 카운트와 경로 홉 카운트를 비교하는 기능과, 기존 스레드 칼라의 투명 여부에 따라 스레드 가두기를 하거나 또는 다운스트림 방향으로 홉 카운트를 조정하는 기능을 추가하였다.

스레드의 동작을 확장, 되감기, 병합, 가두기, 철수로 정의한 것은 그대로 유지하고, 스레드의 상태를 null, colored, transparent로 정의하였지만 동작 알고리즘을 개선하면서 merged 상태를 추가하였다. 또 Hrec, Hop_count로 정의되는 링크의 순차성을 유지하는 것이 중점을 두어, 홉 카운트를 “unknown”으로 설정하고 루핑을 다시 한번 허용하는 복잡성을 제거할 수 있었다. 향후 이 알고리즘을 이용하여 시스템을 구축하는 연구가 이어져야 할 것으로 본다.

참 고 문 헌

[1] Bruce Davie and Yakov Rekhter, “ MPLS Technology and Applications”, Morgan Kaufmann

Publishers, 2000.
[2] B.Davie, Y.Rekhter, E.Rosen, A.Viswanathan, V.Srinivasan, “Use of Label Switching with RSVP”, IETF Draft draft-ietf-mpls-rsvp-00.txt
[3] Yoshihiro Ohba, Yasuhiro Katsube, “ MPLS Loop Prevention Mechanism“, IETF rfc 3063, Feb., 2001.
[4] Yoshihiro Ohba, “Issues on Loop prevention in MPLS Networks”, IEEE Communication Magazine Dec., 1999.
[4] E.Rosen, Y.Rekhter, D.Tappan, D.Farinacci, G.Fedorkow, T.Li, A.Conta, ”MPLS Label Stack Encoding”, IETF rfc 3032, Jan., 2001.
[6] Bruce Davie, Paul Doolan and Yakov Rekhter, “ Switching in IP Networks”, Kaufmann Publishers, 1998.
[5] Y.Rekhter, B. Davie, D. Katz, E. Rosen, G. Swallow, “Cisco Systems’ Tag Switching Architecture Overview”, IETF rfc 2105, Feb., 1997
[6] Eric C. Rosen, Arun Viswanathan, Ross Callon, “Multiprotocol Label Switching Architecture”, IETF rfc 3031, Jan., 2001.
[7] R.Callon, N.Feldman, A.Fredette, G.Swallow, A.Viswanathan, “A Framework for Multiprotocol Label Switching”, IETF Draft draft-ietf-mpls-framework-03.txt
[8] L.Andersson, P.Doolan, N.Feldman, A.Fredette, B.Thomas, “LDP Specification”, IETF rfc 3036, Jan., 2001.
[9] David McDysan, Darren Spohn, “ATM Theory and Applications”, McGraw-Hill, Signature edition
[10] 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.
[11] D.Awduche, J.Malcolm, J.Agogbua, “Requirements for Traffic Engineering Over MPLS”, IETF rfc 2702, Sep., 1999.
[12] B.Jamoussi, “Constraint-Based LSP Setup using LDP”, IETF Draft draft-ietf-mpls-cr-ldp-01.txt

◐ 저 자 소 개 ◑

김 한 경
1973년 서울대학교 원자력공학과 졸업(학사)
1987년 충북대학교 대학원 전산통계학과 졸업(석사)
1996년 충북대학교 대학원 전자계산학과 졸업(박사)
1997～현재 창원대학교 컴퓨터공학과 교수
관심분야 : 소프트웨어공학, 프로젝트관리론
E-mail : hkim@changwon.ac.kr

이 광 휘
1983년 고려대학교 전자공학과 졸업(학사)
1985년 고려대학교 전자공학과 졸업(석사)
1989년 고려대학교 전자공학과 졸업(박사)
1988～현재 창원대학교 컴퓨터공학과 교수
관심분야 : 무선센서 네트워크, 네트워크 관리
E-mail : khlee@changwon.ac.kr