组织：中国互动出版网（http://www.china-pub.com/）
（http://www.china-pub.com/compters/emook/aboutemook.htm）
E-mail：ouyang@china-pub.com
译者：徐春红（airria　air_xu@263.net ）
译文发布时间：2001-3-29
版权：本中文翻译文档版权归中国互动出版网所有。可以用于非商业用途自由转载，但必须保留本文档的翻译及版权信息。

MPLS（多协议标签交换）环路预防机制

摘要

本文讲述了一种基于“线程”的、用于防止多协议标签交换协议（MPLS）设置含有环路的标签交换路径(LSP)的简单机制。此机制与虚电路（VC）的合并相兼容，但此兼容 u 并不是必需的。该机制还可用于下游按需等级分配也可用于下游等级分配。在协议消息中对要传输的信息进行了紧密的捆绑（也就是，不需使用路径矢量）。当一个节点需要转换到它的下一跳时，分布式程序被执行。不过，这只针对那些下游变化的节点。

目录

摘要 1
1.介绍 2
2. 基本的定义 3
3.线程基础 4
　　3.1　线程属性 4
　　3.2. 线程环 5
　　3.3. 线程的基本行为 5
　　3.4　线程基本行为实例 7
4. 线程算法 9
5. 算法的适用性 10
　　5.1　LSP 路由环的预防/检测 10
　　5.2 当新路径上有路由环时使用旧路径 10
　　5.3　如何处理下游等级分配 10
　　5.4. 如何实现负载的分离 11
6. 为什么算法是有效的？ 11
　　6.1　为什么一个带有未知跳数的线程被扩展 11
　　6.2. 为什么一个回绕的线程不能包含一个环？ 12
　　6.3. 为什么 L3 路由环被检测 12
　　6.4. 为什么 L3 不被错误地检测 12
　　6．5 一个滞留线程怎样自动地从环路中恢复 12
　　6.6. 为什么不同颜色的线程不能相互追赶？ 13
7. 环预防的例子 13
　　7.1 第一个例子 14

7.2. 第二个例子 16
8. 线程控制时钟 16
8.1. 有限状态机制 17
9. 与路径矢量/扩散方法进行比较 20
10 .安全考虑 20
附录 A--算法的进一步讨论 20
A.1. 环路预防的强制方法 21
A.2. 强制方法有哪些不妥？ 21
A.3. 线程跳数 22
A.4. 线程颜色 23
A.5. 颜色和跳数之间的关系 23
A.6. 检测线程环 23
A.7. 预防 LSPs 环路的建立 24
A.8. 撤销线程 25
A.9. 修改现有线程的跳数和颜色 25
A.10.什么时候没有下一跳？ 25
A.11. 下一跳的变化和先前存在的有颜色的输入线程 26
A.12. 一个环中有多少个线程在运行？ 26
A.13. 关于跳数 U 的一些特殊规则 27
A.14. 从环路中恢复 27
A.15. 继续使用旧路径 29

1.介绍

本文讲述了一种基于线程，用于防止 MPLS 设置具有路由环的标签转换路径（LSPs）的简单机制。

当一个 LSR(标签交换路由器)发现其有一个新的特定的跳向 FEC（等效前传类）[1]的下一跳时，它就创建一个线程并且将其扩展为下游。每一个这样的线程都被分配唯一的一种颜色来标识，这样就可保证网络上的任何两个线程都不会有相同的颜色。

对于一个给定的 LSP，若从节点到最远的上游节点上都没有跳数变化，那么一旦一个线程被扩展为一个特定的下一跳，其它的线程就不能再被扩展为这样的下一跳。与特定 LSP 的 特定的下一跳相关联的仅有的状态信息就是线程的颜色和跳数。

如果存在路由环，那么某一线程将会返回至它已经经过的 LSR 处。因为线程有特定的颜色，所以这一点很容易检测。

第三部分和第四部分提供了用于检测的没有路由环的程序。当线程被检测的时候，线程被回绕至其创建处。当他们被回绕时，标签被分配。因而，标签只有在保证自由路由环时才被分配。

当一个线程被扩展时，它所经过的 LSRs 必须记录它的颜色和跳数，但是当线程已被回绕时，LSRs 就只需要记录线程的跳数。

如果 LSP 中有一些，或全部，或根本就不存在 LSRs 支持 VC-合并，线程机制同样有效。它可以被用于请求的下游按需标签分配或者用于未经请求的下游标签分配[2,3]。该机制也可用于路由环检测，旧路由的保留和负载分离。

协议消息必须携带的并且必须保留在状态表内部的那些状态信息大小是固定的，与网络大小无关。因而线程机制比那些需带有路径矢量的选择对象更具有可伸缩性。

为了在路由变化后，建立一个新的 LSP，线程机制仅需要在变化点的下游节点之间进行通信，而不需要在变化点的上游节点之间进行通信。所以，线程机制比要求执行扩散算法的选择对象更加健壮。（参看第 9 部分）

2．基本的定义

LSP

我们将运用术语 LSP 参照一个根结点为出口点的多点到一点的树。参见 3.5[3]部分。

下面，我们假设网络中只设有一个 LSP 来讨论。这使得我们在谈到输入、输出链接时不需要老是说“对于同一个 LSP”这样的话。

输入链接，上游链接

输出链接，下游链接

在给定的节点处，一个 LSP 将有一个或更多个输入或上游链接和一个输出链接或下游链接。一个“链接”实际上是邻近 LSR 的一种抽象的联系；它是“树”的一个“边缘”，且就象一个“接口”一样不必是一个特别的具体实例。

叶节点，入口节点

没有上游链接的节点

符合条件的叶节点

能够成为叶节点的节点。例如，如果一个节点不能直接创建一个 L3 包或者在其输出链接上不能接收一个 L3 包，那么此节点就不是一个符合条件的叶节点。

链接跳数

每个链接用一个“链接跳数”标识。此“链接跳数”就是给定链接与其最远的上游叶节点之间的跳的数目。在任何一个节点上，下游链接的链接跳数总是比与之相关的最大的上游链接跳数大 1。

在一个给定节点上，我们用"Hmax"来定义节点上所有的输入链接跳数的最大值。注意：下游链接跳数等于 Hmax+1。在一个叶节点上，Hmax 被置为 0。

图 1 显示了一个链接跳数的例子。

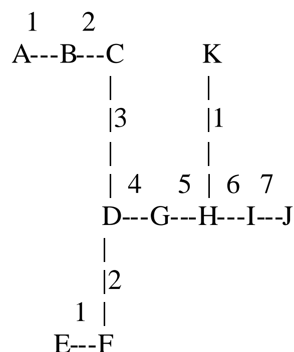


图 1 一个链接跳数例子

下一跳的获取

虽然以前节点 N 认为 FEC F 是不可到达的，但是现在它却可以有通向它的下一跳。

下一跳的丢失

节点 N 以前认为节点 A 是通向 FEC F 的下一跳，但是现在在节点 A 再也没有通向 FEC F 的下一跳了。无论何时下一跳向下传，节点 A 总会丢失一个下一跳。

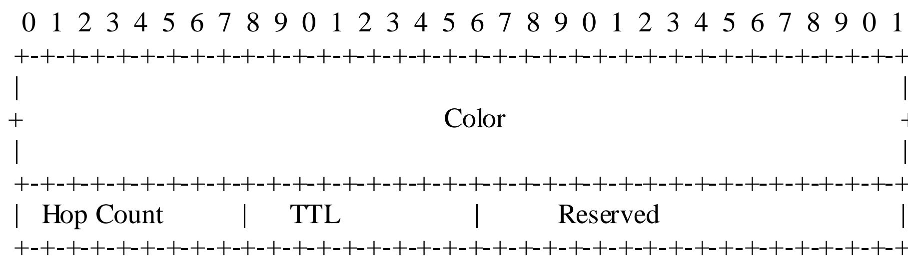
下一跳的变化
在节点 N，通向 FEC F 的下一跳从节点 A 变化到 B，这里节点 A 与 B 是不同的。下一跳的变化可以看作是老的下一跳丢失事件与新下一跳获取事件的结合。

3.线程基础

在下游按需等级分配形式（入口初始化分配控制）中，线程即是用于建立一个 LSP 的消息序列。

3.1　线程属性

线程有三个属性。这些属性可被编码到如下一个单独的线程对象中。



线程颜色
每当一个节点产生一个路径控制消息时，节点就会分配一种唯一的颜色给此消息。此颜色在时间和空间都是唯一的：它的编码包含一个节点的 IP 地址，而此地址与该节点所保留的编号空间的唯一的事件确认相关联。节点发送给下游的路径设置消息将包含该颜色。另外，当一个节点发送包含颜色的消息给下游时，节点将记录此颜色并且将此颜色设为下游链接的颜色。
当一个带有颜色信息的消息被接收时，它的颜色就成为输入链接的颜色。包含某一特定颜色消息的线程就被标识为具有那种颜色的线程。
特别的颜色值“透明色”（全 0）被保留。
分配唯一的颜色值的一个可行方法是：从事件标识符的初始值开始，每分配一种颜色值对其加 1（对最大值取模）。用这种方法，最初的事件标识符可以随机选取也可以分配一个比以前系统赋值所使用的最大的事件标识符更大的值。

线程跳数
为了保持链接跳数，我们需要在路径控制消息中携带跳数。例如，一个叶节点将分配跳数 1 给它的下游链接并将此值存储到它发往下游的路径建立消息中。当一个路径建立消息被发往下游时，一个节点将分配一个比其最大的输入链接跳数大 1 的跳数值给它的下游链接，它也会将此值存储到其发往下游的路径控制消息中。一旦跳数值被存储到路径控制消息中，我们就可将此值当作一个“线程跳数”。

一个特别的跳数值“未知的”（=0xff）比任何一个已知的值都要大，它被用于存在路由环的情况下。一旦线程跳数值成为“未知的”，那么当线程被扩展时，它的跳数值就不再会被增加。

线程 TTL

为了避免某些情况下的未定义的控制信息环，一个 TTL 线程就会被创建。当一个节点产生一个路径控制消息时并且将其发送到下游时，它就为其消息产生一个 TTL 线程。该 TTL 线程在每一跳上都被减 1。当 TTL 线程到达 0 时，消息就不能再向前传了。与线程跳数和线程颜色不同，TTLs 线程不需要被存储在输入链接中。

3.2. 线程环

当同一颜色的线程在多个输入链接中被接收到或者接收到的线程颜色是由接收节点分配时，我们就称线程形成了循环。通过检查接收线程颜色中的 IP 地址部分，一个线程创建者就可以判断出它是否分配了它接收到的线程的颜色。

3.3. 线程的基本行为

为了防止 LSP 路由环，通过使用线程：“扩展”、“回绕”、“撤销”、“合并”、“滞留”来定义线程的五个基本行为。本部分仅描述每个基本行为，并不描述这些基本行为是如何相互作用以及整个算法是如何起作用的。算法的主体部分将在每 4 部分描述。

线程的扩展(extending)

当一个节点开始发送一个带有一组线程属性的路径建立消息给它的下一跳时，我们就说“此节点创建了一个线程并将之向下游扩展”。当一个节点从其上游节点接收到带有一组线程属性的路径建立消息并将其发向它的下游时，我们就说“此节点接收到一个线程并将之向下游扩展”。线程的颜色和跳数就成为输出链接的颜色和跳数。在一个特定的链接处，不管何时接收到一个线程，线程的颜色都将代替链接先前所具有的任何颜色和跳数而成为该输入链接当前的颜色和跳数。

例如，当一个入口节点初始化一个路径建立时，它就创建了一个线程并通过发送一个路径建立消息将其扩展到下游。线程跳数被设置为 1，线程颜色用一个合适的事件标识符被设置为入口节点的地址，TTL 线程被设置为它的最大值。

当一个节点接收一个线程并且将之扩展至下游时，节点或者（i）不改变颜色扩展线程或者(ii)改变颜色扩展线程。如果线程是在一个新的输入链接上被接收并在一个已经存在的输出链接上被扩展，那么这个被接收的线程就会被改变颜色扩展，其他情况下，线程的扩展就不用改变颜色。当一个线程被改变颜色扩展时，一个新的颜色的线程就被创建和并被扩展。

线程的创建不仅仅只在叶节点才会发生。如果一个中间节点有一个输入链接，无论何时它获得一个新的下一跳时，它就会创建和扩展一个新的线程。

当一个节点注意到一个链接跳数减少的下一跳节点时，只要它不是在扩展一个带颜色信息的线程，一个透明线程就会被扩展。

线程合并(merging)

当一个具有颜色的输出链接的节点接收到一个新线程时，它不必一定要扩展新线程。相反，它可以将新线程“合并”到现有的输出线程中。在这种情况下，没有消息被送往下游。而且，如果一个新的输入线程被扩展至下游时，但同时还有其它的输入线程时，这些其它的输入线程将被考虑合并到新的输出线程中。

特别的是，如果具有所有下列情形，那么一个接收线程就被合并：

1) 一个带颜色信息的线程被节点 N 接收
2) 线程不形成循环
3) N 是一个出口节点
4) N 的输出链接是带有颜色的
5) N 的输出链接跳数至少比新近接收到的线程跳数大 1
6) 当一个输出线程回绕时，（参看下面），任何一个与之合并的输入线程也将回绕。

线程滞留(stalling)

当一个带颜色信息的线程被接收到时，如果线程形成了一个环路，则接收到的线程不被扩展，且它的颜色和跳数将被存储到接收线程的链接中。这是线程合并的特例，仅适用于形成环路的线程，并被称为“线程滞留”。存储滞留线程的输入链接被称为“滞留输入链接”。滞留输入链接与非滞留输入链接之间存在显著的差别。

线程回绕（rewinding）

当线程到达一个能实现特殊的自由环路节点时，该节点能够在线程被扩展的相反路径上返回一个应答消息给消息的产生者。应答消息的传送过程就是线程的“回绕”。

自由环路情况如下：

一个带颜色信息的线程被出口节点接收，或

包含下列所有情况：

(a) 一个带颜色信息的线程被节点 N 接收，并且
(b) N 节点的输出链接是透明色，并且
(c) N 节点的输出链接跳数至少比新近接收的线程跳数大 1

当一个节点回绕一个在特殊链接上接收到的线程时，它就将链接的颜色改为透明色。

如果从节点 M 到节点 N 有一个链接，并且 M 在此链接上扩展了一个线程到 N，而 M 确定了（通过接收 N 的一条消息）N 已经回绕了那个线程，那么 M 就设置输出链接的颜色为透明色。然后 M 继续回绕线程，除此而外，还回绕任何已经被回绕的线程合并的输入线程，其中包含滞留线程。

每一个节点均可在所有的输入输出链接变为透明色之后开始标签交换。

注意透明色的线程是已经被回绕的线程；因此，不存在回绕一个具有透明色的线程这样的事情。

线程撤销（Withdrawing）

从路径上退出对于一个线程而言是可能的。一个节点通过向它的下一跳发送一个退出消息从其下游路径的某一部分退出。这个过程被称为“线程撤销。”

例如，假设一个节点正试图建立一条路径，然后遇到了下一跳变化或者下一跳丢失的情况。它就会撤销掉那个向旧的下一跳扩展的线程。

如果节点 M 已经扩展了一个线程到节点 N 并且节点 M 然后又撤销了那个线程，那么现在节点 N 的输入链接就会比先前要少 1。如果现在节点 N 没有其它的非滞留线程并且也不是符合条件的叶节点，那么它就必须撤销掉它的输出线程。如果节点仍然还有一个非滞留输入链接或者节点 N 是一个符合条件的叶节点，它就可能（也可能不）需要改变输出链接的跳数。

下列情况下，节点 N 需要改变输出跳数：

1）刚消除的输出链接跳数比现存的任何输入链接的跳数都要大，并且

2）包含下列情形之一

a)输出链接是透明色的，或者

b)输出链接具有一个已知的跳数

如果输出链接是透明色的，那么它仍然保持透明色，不过，新的跳数仍需要被送至下游。如果输出链接是带有颜色信息的，一个新的线程（带有一种新颜色）需要被创建并且被扩展到下游。

3.4 线程基本行为实例

下面的符号用于描述线程基本行为的实例。

每个链接中的两个线程属性通过“(C,H)”来表示，这里 C 和 H 分别代表线程的颜色和线程跳数。

被标以“+”号的线程表示它正在被创建或接收。标以“-”号的线程表示它正在被撤销。

用一个方括号标记的链接（也就是，“[a]”）表示它是一个非滞留的链接。用一个花括号标记的链接（也就是，“{a}”）表示它是一个滞留链接。

图 2 显示了一个叶节点创建一个蓝色线程并将其向下游扩展的例子。

```
      (bl,1)
   A---[o1]--->
```

图 2．在叶节点扩展的线程

图 3 显示了在中间节点处不改变颜色对线程进行扩展的例子。假设节点 B 在接收到蓝色线程之前没有输入，输出链接。当节点 B 在一新链接 i1 处接收到一个跳数为 1 的蓝色线程时，它将线程扩展到下游并且不改变线程的颜色（图 3(a)）。在蓝色线程被扩展后，节点 B 又在输入链接 i1 处接收到一不知跳数的红色线程（图 3(b)）。红色线程也被不改变颜色地扩展，因为 i1 和 o1 都已经存在了。

```
  (bl,1)+      (bl,2)        (re,U)+      (re,U)
----[i1]--->B---[o1]---->   ----[i1]--->B----[o1]--->

    图 3(a)                  图 3(b)
```

图 3. 不改变颜色的线程扩展例子

图 4 显示了改变颜色的线程扩展例子。图 4(a)中只有一个输入链接 i1 和一个输出链接 o1。然后跳数为 3 的一个红色线程在新的输入链接 i2 处被接收。在这种情况下，接收的线程将会改变颜色进行扩展，也就是，一个绿色线程被创建并被扩展（图 4(b)），因为 o1 已经存在了。

```
   (bl,1)       (bl,2)        (bl,1)       (gr,4)
----[i1]--->B----[o1]--->   ----[i1]--->B----[o1]--->
                                        ^
                                        |
                            ----[i2]----+
                              (re,3)+

    图 4(a)                    图 4(b)
```

图 4. 改变颜色的线程扩展例子

图 5 显示了一个线程合并的例子。当一个节点 B 接收到一个跳数为 3 的红色线程时，因为输出链接跳数至少比接收的线程跳数大 1，故而此接收到的线程并不被扩展。当输出链接 o1 处的蓝色线程被回绕时，红色线程和蓝色线程都将被回绕。

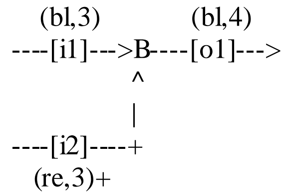


图 5. 线程合并

图 6 和 7 显示了线程滞留的例子。当一个节点 B 在图 6 中的输入链接 i2 处接收到一个跳数为 10 的蓝色线程时，由于蓝色线程形成了一个环路，它“滞留”了接收的线程。在图 7 中，一个叶节点 A 找到了其自身线程的环路。

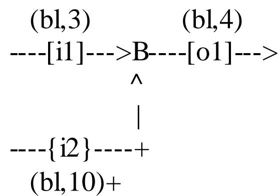


图 6 线程滞留（1）

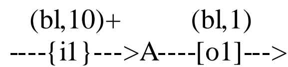


图 7 线程滞留（2）

图 8 显示了一个线程回绕的例子。当一个正被扩展的黄色线程被回绕时（图 8(a)），节点会将会把所有的输入输出线程的颜色改变为透明色，并且将线程的回绕传播至上游节点（图 8(b)）.

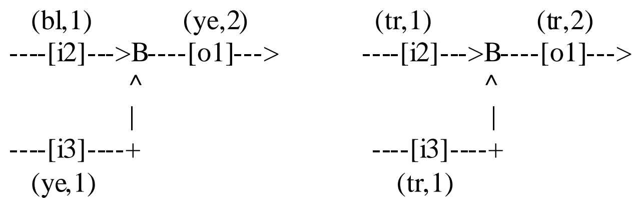


图 8(a) 图 8(b)

图 8 线程回绕

图 9 显示了一个线程撤销的例子。在图 9(a)中，在输入链接 i2 中红色线程被撤销了。然后 Hmax 从 3 减小至 1。节点 B 创建了一个新的绿色线程并且将它向下游扩展，如图 9(b)所示。

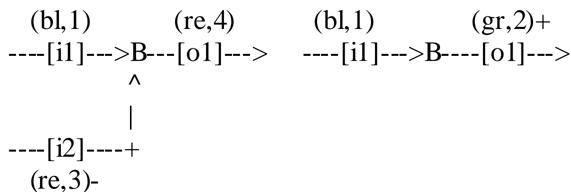


图 9(a) 图 9(b)

图 9 线程撤销(1)

图 10 显示了线程撤销的另一个例子。在图 10(a)中，输入链接 i3 处的红色线程被撤销了。在这种情况下，Hmax 从未知减至 1，然而，如图 10(b)所示，此时没有线程被扩展，因为输出链接有一个带有颜色信息的线程并且跳数是未知的。

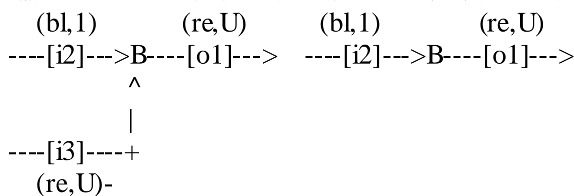


图 10(a) 图 10(b)

图 10 线程撤销(2)

图 11 显示了另一个线程撤销的例子。在图 11(a)中，输入链接 i3 上的透明线程被撤销了。在这种情况下，一个透明线程被扩展（图 11(b)）,因为 Hmax 减小了而且输出链接又是透明的。

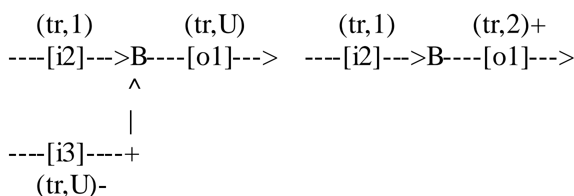


图 11(a) 图 11(b)

图 11 线程撤销(3)

4. 线程算法

这里假设为下游按需等级分配，然而，此算法也适用于下游等级分配，这将在第 5 部分讲述。

在此算法中，下一跳的变化事件将被分成两个事件：依次为旧下一跳上的下一跳丢失事件和新下一跳上的下一跳获取事件。

下面的符号被定义：

Hmax: 最大的输入链接跳数

Ni: 非滞留输入链接数目

线程算法描述如下：

当一个节点获得一个新下一跳时，它就创建一个事带有颜色信息的线程并且把它扩展到下游。

当一个节点丢失它向之扩展线程的下一跳时，它可能会撤销掉那个线程。正如第 3 部分所描述的那样，这将可能会也可能不会引起下一跳的某种反应。在这些反应中，下一跳采取的可能反应就是从它自己的下一跳中撤销掉线程，或者扩展一个新线程到它自己的下一跳。

一个收到的带有颜色信息的线程或者被滞留，或者被合并，或者被回绕，或者被扩展。一个带有 TTL 为 0 的线程永远不会被扩展。

当一个接收线程在一个节点被滞留时，如果 Ni=0 并且此节点不是一个符合条件的叶节点，就会产生线程的撤销。否则，如果 Ni>0 并且接收线程的跳数是未知的，一个带有颜色信息的线程被创建并被扩展。如果接收的线程跳数是未知的，则就不会有线程被扩展，节点也不会采取有进一步。

当一个正在被扩展的线程被回绕时，如果线程的跳数比 Hmax 还大 1，一个跳数为(Hmax+1)的线程将被扩展至下游。

当一个具有透明色的输出链接节点接收到一个有透明色的线程时，如果 Hmax 减少的话，那么节点将对它进行不改变颜色的下游扩展。

5. 算法的适用性

第 4 部分描述的线程算法可以运用于不同的 LSP 管理决策。

5.1 LSP 路由环的预防/检测

同一个线程算法既可以适用于 LSP 路由环的预防也可以适用于检测。

在路由环预防模式中，只有当节点为 LSP 回绕线程时，它才会传送一个标签映射（包含一个线程对象）给 LSP。只有线程被回绕时，才会有映射消息被发送。

另一方面，如果一个节点在路由环检测模式下，它在接收到一个带有颜色信息的线程时，它会立刻返回一个不含线程对象的标签映射消息。一个接收不含有线程对象的标签映射消息的节点不能回绕线程。

5.2 当新路径上有路由环时使用旧路径

当一个路由发生变化时，如果新路由是路由环时，你可能会想继续使用旧的路径。这很简单，只要将分配给旧路径上的下游链接的标签一直保留到新路由上被扩展的线程被回绕。这是一个执行选择。

5.3 如何处理下游等级分配

线程机制也可适用于下游等级分配模式（或由出口控制等级），但前提是将新近从下一跳接收一条标签映射消息的事件看作是下一跳获取事件。

注意一个没有输入链接的节点可看作是一个叶节点。在树刚被建立的情况下（例如，出口节点刚刚出现），每一个节点在短期内将依次被看作是一个叶节点。

5.4. 如何实现负载的分离

一个叶节点通过为同一个 FEC 建立两个不同的 LSPs 可以很容易的实现负载的分离。只不过两个 LSPs 的下游链接要被分配不同的颜色。这里的线程算法不但在两条路径上都可防止路由环，而且还允许两条路径有一个公共的下游节点。

若一些中间节点也想进行负载分离，则需做如下修改。假设同一个 FEC 有多个下一跳。如果一个特定的 FEC 有 n 个下一跳，那么为 FEC 的 LSP 的建立输入链接将被划分为建立 n 个子链接，每一子链接对应到一个不同的输出链接上。

这为为 FEC 提供了 n 个 LSPs。每一个这样的 LSP 为各自的输出链接使用不同的颜色。这里，线程算法不但防止了任何一条路径上的路由环，而且还允许其中两个路径或更多的路径有一个公共的下游节点。

在这种情况下，将会发生一个有趣的现象。我们用图 12 来说明这一点：节点 B 有两个输入链接,i1 和 i2,两个输出链接 o1 和 o2，这样 i1 就被映射到 o1,同时 i2 被映射到 o2。

若一在 i1 处被接收并在 o1 处被扩展的蓝色线程又在节点 B 处的 i2 被接收，这个蓝色线程并不被认为是形成了一个环，因为 i1 和 i2 被认为属于不同的子链接。另一种情况是，蓝色线程在 i2 被接收并在 o2 被扩展。如果在 o2 被扩展的线程被回绕，一个独特的穿过节占 B 的自由环路 LSP 就会被建立。

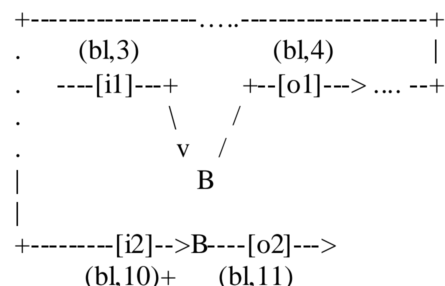


图 12　中间节点处的负载分离

还有一种类型的负载分离，在此类型中，到达单个输入链接中的包将被标签交换到任何一个多输出链接中。这种方法并不是一个好的负载分离的好方案，因为同一个 FEC 中的包的次序并没有被保存。因此，本文并不着重讲这种情况。

不管是不是一个好的负载分离方案，由于 ATM 交换不能在每一个包的基础上作转发决定，故而仍存在 ATM-LSRs 不能照前面的方案进行负载分离的实际情况。

6．为什么算法是有效的？

6.1　为什么一个带有未知跳数的线程被扩展

在算法中，当一个线程环被检测时，一个未知跳数的线程就被扩展。这样可以通过合并那些具有未知跳数流入流出路由环的线程来减少路由环预防信息的数目。参看附录 A.12

6.2. 为什么一个回绕的线程不能包含一个环？

6.2.1. 情况 1：具有已知跳数的 LSP

当输出链接跳数不是“未知的”时，我们怎样才能保证一个已被建立的路径不含有路由环呢？

考虑一个 LSRs 序列<R1,…,Rn>，这样在序列中就存在一个穿过 LSRs 的路由环。（例如，包从 R1 传到 R2,再传到 R3,等等，再到 Rn，然后再从 Rn 传到 R1）。

假设 R1 和 R2 之间的链接的线程跳数是 k,那么通过上述过程，在 Rn 和 R1 之间的跳数必定为 k+n-1。但是算法也保证了如果一个节点有一个输入跳数为 j，则它的输出跳数必须至少是 j+1。因此，如果我们假设由于线程回绕而建立的 LSP 具有一个路由环，在 R1 和 R2 之间的跳数至少是 k+n。从这一点来看，我们可能会得出一个很可笑的结论：n=0，我们因此也许会得出结论：根本不存在这样的 LSRs 序列。

6.2.2. 情况 2：具有未知跳数的 LSP

当输出链接跳数是“未知的”时，一个已建立的路径也不包含一个路由环。这是因为一个带有颜色信息和未知跳数的线程只有在它到达出口时才会被回绕。

6.3. 为什么 L3 路由环被检测

不管线程跳数是已知的还是未知的，如果存在一个路由环，那么环路中的某一节点将是通过一个新的输入链接接收线程的最后一个节点。这个线程将总是不改变颜色信息地回到那个节点。因此路由环总是可以通过环路中的至少一个节点被检测。

6.4. 为什么 L3 不被错误地检测

因为从没有一个节点会将具有同样颜色的线程向下游扩展两次，故只有存在 L3 路由环时，一个线程环才会被检测到。

6．5 一个滞留线程怎样自动地从环路中恢复

一旦线程在环路中被滞留，线程（或路径建立请求）有效地保留在环路中，所以路径的重配置（也就是，旧路径上的线程撤销和新路径上的线程扩展）来源于任何一个接收路由变化事件以打破环的节点。

6.6. 为什么不同颜色的线程不能相互追赶？

在算法中，如果同一时间内有几个节点开始扩展线程，那么就会发生多个线程颜色和/或者跳数的更新。我们怎样才能预防多个线程无限制地循环呢？

首先，当一个节点发现有一个线程形成环时，它就创建一个带有“未知的”跳数的线程。所有的此后到达节点，带有已知跳数的环线程都将被合并到这个线程。这样一个线程就象一个线程吸收器。

其次，“改变颜色的线程扩展”预防了两个线程的相互追赶。

假定一个接收线程总是被不改变颜色的扩展。那么我们就遇到下列的情形。

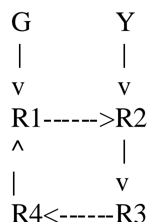


图 13 线程追赶的例子

在图 13 中，(1)节点 G 获得 R1 作为其下一跳并且开始扩展一跳数为 1 的绿色线程，（2）节点 Y 获得 R2 作为其下一跳，并且开始扩展一跳数为 1 的黄色线程。(3)节点 G 和节点 Y 会在这些线程向前流传之前撤销它们。

在这种情况下，黄色和绿色线程会回转并各自回到 R2 和 R1。然而当线程回到 R2 和 R1 时，存放线程颜色的输入链接已不存在了。结果，黄色和绿色线程将永远在环中相互追赶。

不过，既然我们有“改变颜色的扩展”机制，所以上述现象实际上不会发生。当 R2 处接收到一个绿色线程时，R2 通过改变其颜色来扩展它，也就是，创建一个新的红色线程并扩展它。类似地，当 R1 处接收到一个黄色线程时，R1 就创建一个新的紫色线程并扩展它。因此，甚至在节点 G 和 Y 已经撤销掉线程之后，线程环也会被检测到。这就确保了环附近的带有环中某节点所分配的颜色的线程被扩展。

至少还存在着一种甚至“改变颜色的扩展”也不能处理的情形。这就是“自追赶”。在自追赶中，线程的扩展和撤销都是根据环中相互追赶的同一个线程进行的。这种情况将发生在节点扩展一个线程到一个 L3 环后立刻撤销掉该线程时。

一种探测自追赶的方法就是在撤销线程的节点处延迟线程撤销的执行。不管怎样，TTL 线程机制可以删除任何种类的线程环。

7. 环预防的例子

本部分，我们将用两个例子来显示在给定网络中算法是如何实现 LSP 环预防的。

我们假定使用下游等级按需分配，并且所有的 LSPs 都与同一个 FEC 相关，而且所有的节点是能够 VC 合并的。

7.1 第一个例子

考虑图 14 所示的一个 MPLS 网，其中存在一个 L3 环。每个直接链接表示每个节点当前的 FEC 下一跳。这里，叶节点 R1 和 R6 产生 LSP 的创建。

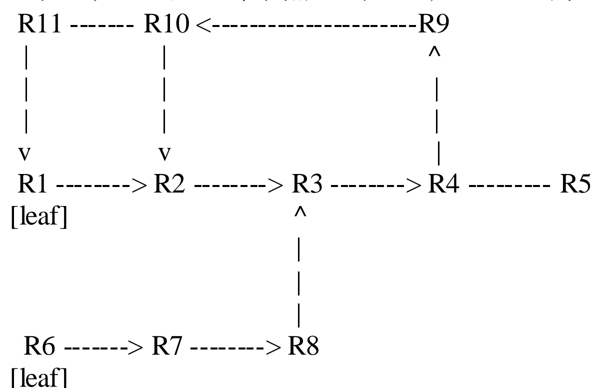


图 14 MPLS 网络的例子（1）

假设 R1 和 R6 同时发送一个标签请求消息并且将 TTL 线程初始化为 255。下面我们将先显示一个如何预防 LSP 环的例子。

线程属性的设置通过（颜色，跳数，TTL）来表示。

来自 R1 和 R6 的请求各包含（re,1,255）和（b1,1,255）。

假设 R3 在收到 R6 的请求之前收到了 R1 的请求。当 R3 接收到第一个带红色线程的请求时，R3 不改变它的颜色且赋之属性（re,3,253）转发，因为接收的输入和输出链接都是刚刚创建的。然后 R3 收到了第二个带蓝色线程的请求。此时，输出链接已经存在了。因而，R3 就执行改变颜色的线程扩展，也就是，创建一个新的棕色线程并赋予属性（br,4,255）转发。

当 R2 接收到 R10 的属性为（re,6,250）的请求时，它发现 R3 形成了一个环且滞留了红色线程。然后，R2 通过发送一个带属性为(pu,U,255)的请求给 R3，创建一个紫色带有未知跳数的线程并将它扩展至下游，这里“U”代表“未知的”。

此后，R2 从 R10 接收到另一个属性为（br,7,252）的请求。棕色线程被合并到紫色线程中。R2 不需要向 R3 发送请求。

另一方面，紫色线程通过现存的链接不改变颜色回转。R2 发现线程环并滞留紫色线程。因为接收到的线程跳数是未知的，因而再没有线程被创建。在这种情况下，没有线程回绕发生。当前的网络状态如图 15 所示。

*: 线程被滞留的位置

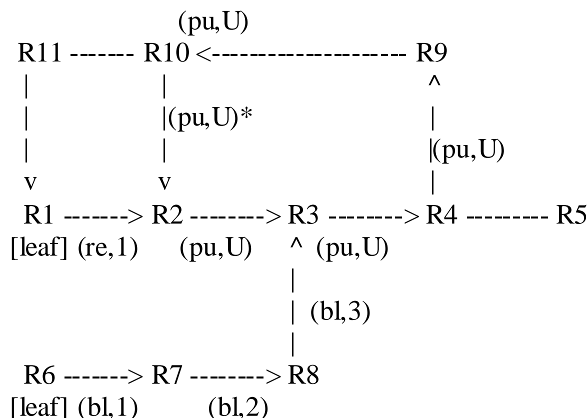


图 15 网络状态

然后 R10 将它的下一跳从 R2 换到 R11。

因为 R10 在旧的下游链接上有一个紫色线程，为了撤销紫色线程它要先发送一个拆卸消息给它的旧的下一跳 R2。接着，它创建一个未知跳数的绿色线程并发送属性为（gr,U,255）的请求给 R11。

当 R2 接收到来自 R10 的拆卸消息时，R2 就将 R10 和 R2 之间的被滞留的输入链接移去。

另一方面，绿色线程到达 R1 时，Hmax 从 0 更新到未知。在这种情况下，由于线程是在新的输入链接上被接收并在已存在的输出链接上被扩展，所以 R1 执行改变颜色的线程扩展。结果就是，R1 创建了一个桔黄色的具未知跳数的线程并将之扩展至 R2。

桔黄色的线程通过已有的链接进行不改变颜色的回转，最终它被滞留在 R1。

现在的网络状态如图 16 所示。

*: 线程滞留的位置

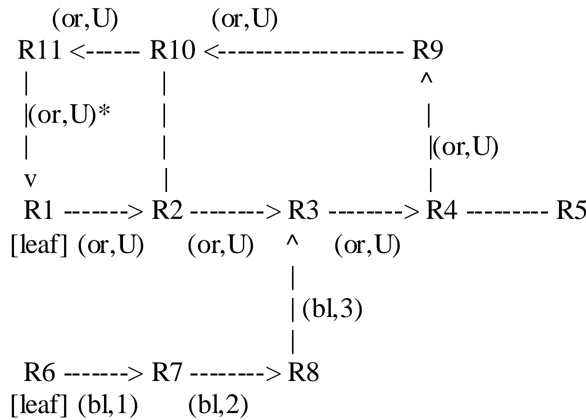


图 16 网络状态

然后 R4 将它的下一跳从 R9 换到 R5。

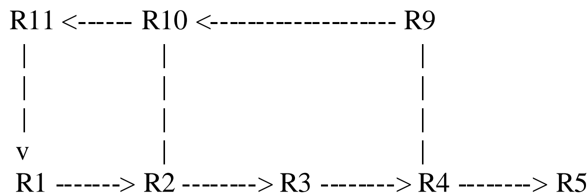
因为 R4 正扩展一个桔黄色线程，它先发送一个拆卸消息给旧的下一跳 R9 以撤销旧路由上的桔黄色线程。接着，它创建一个具未知跳数黄色线程并发送一带属性为（ye,U,255）的请求给 R5。

由于 R5 是出口节点，黄色线程就要回绕。R5 返回一个标签映射消息。当标签映射消息逐跳被返回至上游时，线程回绕程序就在每个节点处被执行。

如果 R1 在接收 R11 的桔黄色线程的撤销消息之前收到一个标签映射消息时，R1 就返回一个标签映射消息给 R11。一接收到桔黄色消息的撤销消息，R1 为了通知具已知跳数的下游，它会立刻创建一个透明色的线程并通过发送一带(tr,1,255)属性的更新消息来扩展它。

否则，如果 R1 在接收到标签映射消息之前就收到了桔黄色线程的撤销消息，R1 就会移去被滞留的输入桔黄色链接并等待输出桔黄色线程的回绕。最终，当 R1 从 R2 收到一标签映射消息时，它就创建一属性为(tr,1,255)透明色的线程并将它扩展至下游。

在两种情况下，一个合并的 LSP((R1->R2),(R6->R7->R8))->R3->R4->R5) 被创建并且每一个节点获得正确的跳数。最终的网络如图 17 所示。



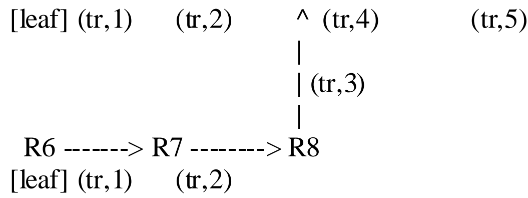


图 17 最终的网络状态

7.2. 第二个例子

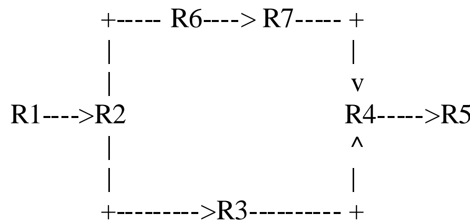


图 18 MPLS 网络的例子 (2)

假设在图 18 中存在一个已建立的 LSP R1->R2->R3->R4->R5,R2 的下一跳从 R3 变化到 R6。R2 发送一个带红色线程（re,2,255） 的请求给 R6。当带有属性（ re,4,253）的请求到达 R4 时，它改变线程的颜色并将之扩展到 R5。因此，一个新的绿色线程在 R4 处被创建并被通过发送一带属性为(gr,5,255)的更新消息扩展到 R5。

当 R5 收到一更新消息时，它就将输入链接的跳数更新为 5 并且返回一个更新应答（也就是带有成功代码的通知消息）。当 R4 收到一个更新应答时，它就返回一个标签映射消息给 R7。

当 R2 在新路由上收到一标签映射消息时，它就发送一个拆卸消息给 R3。当 R4 收到一个拆卸消息时，由于 Hmax 没有改变，故它并不发送更新消息给 R5。现在就得到了一个已建立的 LSP R1->R2->R6->R7->R4->R5。

然后，R2 处的下一跳又从 R6 变到 R3。

R2 发送一带蓝色线程(bl,2,255) 的请求给 R3。R3 将属性改为(bl,3,254) 转发请求给 R4。

当 R4 收到请求后，因此时 Hmax 没有改变，所以它立刻返回一个标签映射消息给 R3。

当 R2 在新路由上收到一标签映射消息时，它就发送一拆卸消息给 R6。拆卸消息到达 R4，引起一带透明色线程(tr,4,255) 的更新消息给 R5，因为此时 Hmax 从 4 减少到了 3。R5 将输入链接跳数更新为 4，而不返回应答。

8. 线程控制时钟

每一节点的每一个 LSP 中都有一个线程控制时钟（TCB），TCB 可能含有下面的信息：

- FEC

- 状态
- 输入链接

  每个输入链接有如下属性：

  邻居：上游邻居节点地址

  颜色：接收的线程颜色

  跳数： 接收的线程跳数

  标签

  S-标记: 表明是一个滞留链接
- 输出链接

  每个输出链接有如下属性

  邻居：下游邻居节点地址

  颜色：接收的线程颜色

  跳数：接收的线程颜色

  标签

  C-标记：表明当前下一跳的链接

如果在一个输入链接上接收到一个透明色线程，而该线程没有被分配标签，或者一个非透明色被存储，那么此线程会被丢弃而不进入 FSM。一个错误消息可能会返回给发送者。

无论什么时候输入链接收到一个线程，在进入 FSM 之前必须采取一些措施：(1)将接收线程的颜色和跳数存储在链接上以代替旧线程的颜色和跳数（2）设置下面的标记用于“Recv 线程”事件（参看 8.1）中的交换事件。

颜色标记 (CL-flag):

如果接收线程是有颜色的，则设置此标记

环标记 (LP-flag):

如果接收线程形成了一个环，则设置此标记

到达新链接标记 (NL-flag):

如果接收线程到达一个新的输入链接，则设置此标记

如果一个 LP-flag 被设置，除了接收链接外，则必定存在一个输入链接 L，存储与接收链接一样的线程颜色。属于链接 L 的 TCB 被看作是“检测 TCB”。如果接收 LSR 能够进行 VC 合并，则检测 TCB 与接收 TCB 是一样的，否则，两个 TCBs 不同的。

在执行线程扩展前，TTL 线程被减 1。如果最后 TTL 变为 0，线程不是被扩展而是被静静地丢弃。否则，线程被扩展并且扩展的线程颜色和跳数被存储到输出链接中。

当一个节点接收到一个线程回绕事件时，如果接收线程颜色和扩展线程颜色不同，节点会丢弃事件而不进入 FSM。

8.1. 有限状态机制

由 FSM 中一个动作预先按排的事件必须在动作完成后立刻被传递。

FSM 中所使用的变量如下：

Ni: 非滞留输入链接的数目

Hmax: 最大的输入链接跳数

Hout: 当前节点下一跳的输出链接跳数

Hrec: 接收线程的跳数

在 FSM 中，如果 Hmax=未知的，则值（Hmax+1）就是未知跳数加 1。例如，如果 Hmax=未知的=255，值（Hmax+1）就变为 256。

一个 TCB 有三个状态：Null，Colored 和 Transparent。当 TCB 在状态 Null 时，没有输出链接且 Ni=0。状态 Colored 意味着节点正在输出链接上为它的下一跳扩展一个有颜色的线程。状态 Transparent 意味着节点是出口节点或输出链接是透明的。

标记值为“1”代表标记被设置，“0”则代表标记未被设置，“*”意味着标记值或为 1 或为 0。

FSM 允许在旧的下一跳上有一个透明的输出链接和在当前下一跳上有一个带颜色的输出链接。然而，不允许在旧的下一跳上有一个带颜色的输出链接。

状态 Null:

事件 Recv 线程 标记 CL LP NL	动作	新状态
0 * *	无动作	无变化
1 0 *	如果一个节点是出口节点，它会产生一个回绕线程并将接收链接变为透明色。	Transparent
	否则，不改变颜色地扩展接收线程	Colored
1 1 *	滞留接收线程；如果 Hrec<unknown，则为检测 TCB 预先安排“初始化为未知值”事件	No change
下一跳的获得	如果是符合条件的叶节点，创建一带颜色线程并扩展它。	Colored
其它	静态忽略事件	No change

状态 Colored:

事件 Recv 线程 标记 CL LP NL	动作	新状态
0 * *	如果 Hmax+1<Hout<unknown，则创建一带颜色线程并扩展它。否则，什么都不做	No change
1 0 *	如果 Hmax<Hout, 合并接收线程。否则就扩展线程--如果 NL=1，则改变颜色如果 NL=0，则不改变颜色 。	No change
1 1 *	滞留接收线程，如果 Ni=0 并且节点是一个符合条件的叶节点，就引发线程的撤销。	Null
	如果 Ni>0 并且 Hrec<unknown, 为检测 TCB 预先安排“初始化为未知值”事件。	No change
	否则什么都不做	No change

回绕	将回绕线程传播至前面扩展一带颜色线程的跳；将所有的输入输出链接存储的颜色都改为透明色；如果 Hmax+1<Hout，扩展透明色线程。撤销那些 C-flag=0 的输出链接上的线程。	Transparent
撤销	移去相应的输入链接 如果 Ni=0 并且节点不是一符合条件的叶节点，将线程的撤销传播至所有的下一跳。	Null
	否则，如果 Hmax+1<Hout<unknown，创建一带颜色线程并扩展它。	No change
	否则，什么都不做。	No change
下一跳的获得	如果下一跳已存在一个输出链接，则什么都不做（这种情况只有在节点仍留在旧路径上时才会发生）	Transparent
	否则，创建一带颜色线程并扩展它。	No change
下一跳的丢失	如果输出链接是透明色的并且节点被允许留在链接中，而下一跳又是存在的，则什么都不做。否则，采取做如下动作：为下一跳产生一线程的撤销；如果节点变成一新的出口节点时，为这个 TCB 预先按排“回绕”事件。	No change
	如果 Ni=0，转移到 Null。	Null
	否则，什么都不做。	No change
初始化至未知值	创建一跳数为未知值的带颜色线程并扩展之。	No change
其它	静态地忽略事件。	No change

状态 Transparent

事件	动作	新状态
Recv thread Flags CL LP NL		
0 * *	如果 Hmax+1<Hout, 扩展一透明色线程。	No change
1 0 *	如果节点是出口节点或者如果 Hmax<Hout, 将接收链接的颜色改为透明色并引发线程回绕。	No change
	否则，扩展线程----如果 NL=1,则改变颜色，如果 NL=0，则不改变颜色。	Colored
撤销	移去相应的输入链接。 如果 Ni=0 且节点不是一符合条件的叶节点，	Null

	将线程的撤销传播至下一跳。	
	否则，如果 Hmax+1<Hout，创建一透明线并扩展它。	No change
	否则，什么都不做。	No change
下一跳的获得	创建一带颜色线程并扩展它。	Colored
下一跳的丢失	如果节点被允许留在输出链接中并且下一跳是存在的，则什么都不做。否则，采取下动作：引发线程的撤销。	
	如果 Ni=0，则转移到 Null。	Null
	否则，什么都不做。	No change
其它	静态地忽略事件。	No change

9. 与路径矢量/扩散方法进行比较

尽管路径矢量的大小是随着 LSP 的长度增加的，但是线程的大小却是一个常数。因此线程算法使用的消息的大小不受网络或拓扑结构的影响。此外，线程合并能力也减少了消息的数目。这些都将提高伸缩性。

在线程算法中，为特殊的 LSP 改变它的下一跳的节点只与那些在新路径上此节点与 LSP 出口之间的节点相互作用。但在路径矢量算法中，节点必须引发一个扩散估计以包含那些不在此节点与 LSP 出口之间的节点。

这个特征使得线程算法更加健壮。如果使用一扩散估计，甚至不在路径上的节点的错误动作也会延迟路径的建立。在线程算法中，唯一能延迟路径建立的节点是那些真正在路径 上的节点。

线程算法很好地适用于下游按需等级分配和下游等级分配。然而路径矢量/扩散算法中，只适用下游等级分配。

线程算法可以随意重试以获得快速地路径重配置。扩散算法则会延迟路径重配置时间，因为在路由变化点的一个节点必须与它的所有上游节点协商。

在线程算法中，如果在新路径上存在一个 L3 环，节点还能继续使用旧路径，这一点与路径矢量算法相同。

10 .安全考虑

本文档所使用的程序与任何 MPLS 程序使用中出现的程序在安全性上并不发生冲突。

附录 A--算法的进一步讨论

此附录的目的是为了给出一个较不正式和指导性质的算法描述并指出这些算法提出的原因。为了精确描述算法，FSM 应该被看作标准。

正如文档主体一样，我们讨论时就好象只存在一个 LSP，否则我们总是要说“对于同一个 LSP……”。只有算法用于预防环路而不是检测环路时，我们才会这样考虑。

A.1. 环路预防的强制方法

作为开始，我们考虑一个可称之为“强制预防环路”的 算示。在这个算法中，每一次路径建立的尝试都必须到达出口并返回建立路径。这个算法显然是自由环路，因为建立消息实际上送到出口又返回。

例如，考虑一个现有的指向出口节点 E 的 LSP B-C-D-E 。现在 A 试图加入 LSP。在这个算法中，A 必须发送一消息给 B，B 再给 C，C 再给 D，D 再给 E。然后消息被从 E 送回到 A。最终从 B 到 A 的消息包含一个标签绑定，清楚了此路径是一个自由环路，A 就可以加入 LSP 了。

运用我们的术语，我们就说 A 创建了一个线程并将它向下游扩展。线程到达出口并被回绕。

在前面的例子中，我们不需要假设 A 是一个入口节点。若对 LSP 有疑问，任何节点都可以获得或改变它的下一跳节点。

很明显，如果存在一个环路，则线程永不能到达出口，所以它不被回绕。怎么回事呢？路径建立消息一直在环路中。如果在消息中设置一个跳数，则当跳数达到最大值时，消息就会停止在环路中传送。也就是，当收到一具有最大值跳数的路径建立消息时，该路径建立消息不会被送往下游。

如何从一个环线程中恢复呢？对一个 L3 路由，为了打破环，环中的某一节点必须改变它的下一跳。此节点将把该线程从它的旧下一跳上撤销掉并扩展一线程给它的新下一跳。如果不存在环路，这个线程就可到达出口并被回绕。

A.2. 强制方法有哪些不妥？

看下面的例子：

```
A
|
B--D--E
|
C
```

如果 A 和 C 都试图加入已有的 B-D-E 路径，然后 B 和 D 必须保留两个路径建立尝试，一个来自 A，一个来自 C。也就是 D 必须跟踪两个线程—A 线程和 C 线程。一般说来，可能有更多的 B 的上游节点想加入已有的路径，而 D 也必须跟踪所有这些节点。

如果不使用 VC 合并，实际上情形并不是很差。若没有 VC 合并，不管怎样，D 实际上为每一上游节点都维持一 LSP。然而如果使用 VC 合并，每维持一 LSP 请求，就必须保存每个上游链接的状态。若预防环路技术也要求一个节点保存其上游链接所对应的状态信息量，而不是 LSP 中上游节点数，上述方法就比较好。

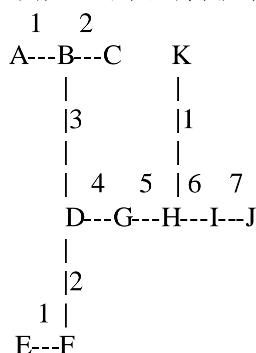
另一个问题是，如果存在一个环，路径建立消息就不停地循环。即使一个线程已两次通过某一节点，该节点也无法辨别当前接收的建立消息是否是它过去已接收的某一建立消息的一部分。

我们可以修改这个强制方案来消除这两个问题吗？可以。为了显示如何做，我们提出了两个概念：线程跳数和线程颜色。

A.3. 线程跳数

假设 LSP 树中每个链接被跳数标记，若想回到上游，你就可以从链接处穿过最远的上游节点到达。

例如，下面的树就采用了链接跳数标记：



称这些是“链接跳数”

链接 AB，EF，KH 都被标记为 1，因为你可以只走 1 跳就到达上游。链接 BC 和 FD 被标记 2，因为你可以走 2 跳从链接处到达上游。链接 DG 被标记为 4，因为可能需走 4 跳才能从链接处到达上游；等等。

注意任何一节点处，与下游链接相关的跳数比与上游链接相关的最大的跳数值大 1。

让我们来看一个保存这些跳数的方法。

为了保存链接跳数，我们需要在路径建立消息中携带跳数。例如，一个没有上游链接的节点将分配跳数值为 1 给它的下游链接，并将该值存储到它发往下游的路径建立消息中。一旦值存储到路径建立消息中，我们就可将称它有一“线程跳数”。

当一路径建立消息被收到时，线程跳数被当作消息被接收的上游链接的链接跳数存储起来。

当一路径建立消息被送往下游时，下游链接跳数（还有线程跳数）值被设置为最大的输入链接跳数加 1。

假设节点 N 有一些输入链接和一个输出链接，且链接跳数都被相应地设置，而且节点 N 现在有一个新的输入链接。只有当当新的输入链接跳数比现有的所有输入链接跳数还大时，下游链接跳数必须改变。这种情况下，必须发送一带有新的更大跳数值的控制消息给下游。

另一方面，若 N 获得一个其链接跳数小于等于其它所有的现有输入链接跳数的新的输入链接，下游链跳数保持不变，也不必发消息给下游。

假设 N 丢失了有最大跳数的输入链接。这种情况下，下游链接跳数必须变小且必须发送一消息给下游以说明这种情况。

如果我们不仅关心环路的预防，而且还关心跳数的保存。那么我们将采用下面的规则（这些规则用于合并点处）

A.3.1 当一个新的输入线程被接收时，只有当在它的跳数是所有输入线程中最大时才会被向下游扩展。

A.3.2 否则，回绕线程。

A.3.3 当然一个出口节点总是回绕线程。

A.4. 线程颜色

节点在改变下一跳或获得下一跳时创建线程。我们假定每当节点创建一个线程时，节点分配一种颜色给线程。这种颜色在时间空间上都是唯一的：它的编码包含一个节点的 IP 地址，而此地址与该节点所保留的编号空间的唯一的事件确认相关联。节点发送给下游的路径建立消息将包含该颜色。另外，当一个节点发送包含颜色的消息给下游时，节点将记录此颜色并且将此颜色设为下游链接的颜色。

当一个带颜色信息的消息被接收时，它的颜色就成为输入链接的颜色。包含某一颜色的消息的线程将被称这具有那种颜色的线程。

当一 线程被回绕（一个路径建立），颜色被移去。链接变为透明色。我们有时候称已建立的 LSP 为一个“透明”线程。

注意在一个带颜色链接上包不能被转发，而只有在透明链接上才可被转发。

注意如果是线程环，某节点将留意通过特定输入链接上的带节点已留意过的颜色信息的消息。节点或者产生那种颜色的线程，或者节点已有那种颜色的一个不同的输入链接。这个事实可被用来预防控制消息的循环。不过，节点必须记住所有通过它的且未被回绕或撤销的线程的颜色。（也就是，它将不得不在此过程中为每条路径记住一种颜色。）

A.5. 颜色和跳数之间的关系

通过将颜色机制和跳数机制联合起来，我们无需任何节点为每一条 LSP 上的每一个链接记住多个颜色和跳数，就能够预防环路的发生。

在前面我们已经提到，为了保存跳数，一个节点只有在线程具有最大的输入链接跳数时，才会扩展它。现在我们加入如下规则：

A.5.1　当向下游扩展一输入线程时，线程的颜色也被传到下游（也就是，下游链接的颜色将和具有最大跳数的上游链接的颜色相同）

注意在一个给定的节点，下游链接或者是透明的，或者有且仅有一种颜色。

A.5.2　如果一个链接改变颜色，就没有必须再记住旧的颜色。

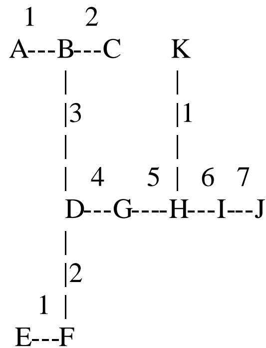
现在我们一定义“线程合并”的概念：

A.5.2　假设一个有颜色线程通过一个输入链接到达了一个节点，节点若已经有一个同样跳数或更大跳数的输入线程和一个输出的有颜色的线程。这种情况下，我们就可以说新的输入线程被“合并”到输出线程中。

注意当一个输入线程被合并到输出线程时，没有消息被发往下游。

A.6. 检测线程环

如果存在一个环，则总是存在某一节点具有同种颜色的两个输入线程，或者带颜色的线程将返回它的创建者。这部分，我们给出几个例子以直观地理解线程环是如何被检测的。



回到前面我们所讲的例子，如果 H 将它的下一跳从 I 换到 E，我们看看会发生什么？H 现在创建一个新的线程并分配一种颜色给它，譬如，红色。由于 H 有两个跳数分别为 1 和 5 的输入链接，它就分配 6 给它的新的下游链接并试图通过 E 建立一条路径。

E 现在有一跳数为 6 的红色输入线程。由于 E 的下游链接跳数现在仅为 1，它必须将红色线程跳数改为 7，然后扩展至 F。然后 F 再将跳数改为 8，扩展红色线程至 D。D 再以跳数 9 扩展至 G，G 再以跳数 10 扩展线程至 H。

红色线程现在已经回到了它的创建者处，环就被会检测到。

假设在红色线程回到 H 之前，G 将它的下一跳从 H 改变到 E。然后 G 将扩展红色线程至 E。但是 E 已经有了一个红色输入链接（来自 H），所以环又可被检测到。

现在让我们来定义一"滞留线程"的概念。一个滞留线程是一个即使输出线程有一个小的链接跳数，也被合并到输出线程中的线程。

当一个线程环被检测到时，线程就被滞留了。

A.6.1 当一个线程环是因某一颜色的线程两次通过一个节点被检测到时，我们就说线程在节点处被"滞留"了。更精确一点，这是线程被滞留的第二表现。注意如果线程被输入链接上的节点接收时，我们说一个线程两次通过一个节点。不过，节点或者有同种颜色的另一输入链接或者颜色是由节点自身分配的。

A.7. 预防 LSPs 环路的建立

用于预防 LSPs 环路建立的机制现在应该很明显了。如果节点 M 是节点 N 的下一跳且希望建立一个 LSP（或在 M 处合并到已有的 LSP 上），那么 N 扩展一线程到 M。

M 首先检查是否线程形成环（参看附录 A.6），如果形成了环，线程被滞留。如果没有形成环，将执行下面的过程。

A.7.1 如果 M 接收到这个线程且 M 有下一跳或者：
— M 没有输出线程
— 输入线程跳数比所有的其它输入线程跳数都大
那么 M 必须向下游扩展该线程

A.7.2 另一方面，如果 M 接收到这个线程且 M 有下一跳，而且还存在着一个更大跳数的输入线程，那么：

A.7.2.1 如果输出线程是透明的，M 就回绕新的输入线程。

A.7.2.2 如果输出线程是有颜色的，M 就将新的输入线程合并到输出线程中，但是不需要向下游发送任何消息。

A.7.3　如果 M 还没有分配标签给 N，那么只有当 M 回绕 N 向它扩展的线程时，M 才会分配一标签给 N。

A.7.4　如果 M 将新的线程合并到一现有的有颜色输出线程中，那么新的输入线程只有在输出线程回绕时，它才回绕。

A.8. 撤销线程

A.8.1　如果一特定节点有带颜色输出线程且丢失了或改变了它的下一跳，它就撤销掉输出线程。

　　假设节点 N 是节点 M 的直接上游且 N 已经向 M 扩展了一个线程。更一步假定 N 然后撤销了线程。

A.8.2　如果 M 有另一个更大跳数的输入线程，那么 M 不需要发送任何消息给下游。

A.8.3　然而，如果被撤销的线程具有任何输入线程中的最大跳数，那么 M 的输出线程将不再有正确的跳数和颜色。因此：

A.8.3.1 M 必须用最大跳数向下游扩展输入线程。（这将使它忘记旧的下游链接跳数和颜色。）
A.8.3.2 其它的输入线程则认为被合并到了扩展的线程中。

A.8.4　当最后一个非滞留的线程被撤销时，输出线程必须被撤销。

A.9. 修改现有线程的跳数和颜色

　　我们已经明白：一个线程的撤销可能会引起下游跳数和颜色发生变化。注意如果一个输出线程的跳数/或颜色发生变化，那么与之对应的下一跳上的跳数和颜色也会发生变化。

A.9.1　无论何时只要存在一输入线程的跳数发生变化，一个节点必须确定是否“所有输入线程中的最大跳数”发生了变化。如果是的话，输出线程的跳数或者是颜色也将发生变化，同时引发一消息送往下游。

A.10.什么时候没有下一跳？

A.10.1　如果一特定节点有一个带颜色线程，但是没有下一跳（或者丢失了它的下一跳），输入线程就被滞留。

A.11. 下一跳的变化和先前存在的有颜色的输入线程

在某时，当节点有带颜色输入线程时，节点将可能改变它的下一跳或者获得一个下一跳。在路径建立完成之前，当路由发生变化时也会发生上述现象。

A.11.1 在某时，当节点有带颜色的输入线程时，若一个节点改变它的下一跳或者获得一个下一跳，它将创建一个新颜色的线程，但是该线程的跳数比最大的输入链接跳数还大 1。然后节点向下游扩展该线程。

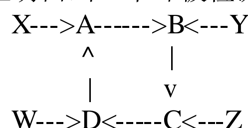
A.11.2 当这个新线程被创建并被扩展至下游，所胡的输入线程都被合并到该线程。任何一个此前被滞留的输入线程现在被看作是被“合并”而不是“滞留”。

也就是说，即使输出线程的颜色与任何一个输入线程的颜色不同，先前存在的输入线程也都看作是已经被合并到了新的输出线程中。这意味着当输出线程回绕时，输入线程也将回绕。

注意：当撤销线程被执行时，仍然需要区别滞留线程和非滞留线程。

A.12. 一个环中有多少个线程在运行？

我们已经明白当一个环被检测到时，环线程被滞留。然而，看下面的拓扑结构：



在这个例子中，存在一个环 A-B-C-D-A。另一方面，也存在一些从 X，Y，Z 和 W 进入环的线程。一旦环被检测到，

要做到这一点，我们引入“未知的”跳数概念，U。这个跳数值被认为是比任何跳数都要大。一个带跳数 U 的线程被称为“U-线程”。

A.12.1 当一个带已知跳数的输入线程被滞留且存在一个输出线程，我们分配跳数 U 给输出线程，同时也分配一个新颜色给输出线程。

结果，下一跳将有一个带新分配的颜色的输入 U-线程。因而这将使下一跳分配跳数 U 和上述新颜色给它的输出线程。我们给出的规则将会使环路中的每一链接被分配一新颜色和跳数 U。当该线程到达它的创建者时，或者到达任何一个有相同颜色的输入线程的节点时，线程被滞留。

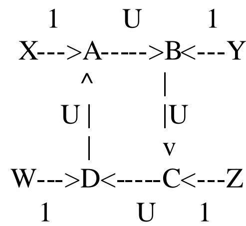
在我们前面所举的例子中，此规则将使得链接 AB，BC，CD 和 DA 被分配 U 跳数。

下面让我们再加入一个规则：

A.12.2 当一个有已知跳数的线程到达一带有颜色输出 U-线程的节点时，输入线程合并到输出线程中（实际上，这只不过是前面已给规则的一个推论，因为 U 比任何一个已知跳数大）。

那么如果 W，X，Y 或 Z 各自试图扩展线程至 D，A，B 或 C 时，那些线程都将被滞留。一旦所有的链接都被标以在环路中，环路上就再没有线程被扩展，也就是说，再没有建立消息通过环路。

下面是带有环中链接跳数的拓扑结构例子：



A.13. 关于跳数 U 的一些特殊规则

当一个 U-线程遇到一个带已知跳数的线程时，通常的规则是适用的，只要记着 U 比任何一个已知跳数值都大。

不过，我们需要增加一些特殊的规则以适用特殊的情形，如一个 U-线程遇到一个 U-线程。由于我们无法区别两个 U-线程哪个存在时间更长，我们就必须保证两个 U-线程都被扩展。

A.13.1 如果一个输入带颜色的 U-线程到达一个已有那种颜色的输入 U-线程的节点或者到达创建该 U-线程的节点时，线程被滞留。
（一旦环路被检测到，就没有必要进一步扩展线程。）

A.13.2 如果一输入带颜色 U-线程到达一具有指向它的下一跳的透明输出 U-线程的节点时，输入线程被扩展。

A.13.3 如果一输入带颜色 U-线程到达一带颜色输出 U-线程的节点时并且如果线程被接收的输入链接已经是 LSP 上的一输入链接时，线程被扩展。

A.13.4 如果一输入带颜色 U-线程到达一带颜色输出 U-线程的节点时并且如果线程被接收的输入链接还不是 LSP 上的一输入链接时，一个新的 U-线程被创建并被扩展。所有的输入线程都被合并到这个线程中。在文档的主体部分这被称为“改变颜色的线程扩展”。

这些规则确保了：一个输入 U-线程在没有形成环路时总是被扩展（或者被合并到一个可扩展的新的 U-线程中）

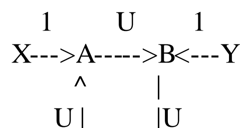
A.13.4 的目的何在？存在着一些环路形成的情况，但是产生环线程的节点并不是环中的一部分。规则 A.13.4 确保了当存在一个环时，总有一个环线程是由环路中某一节点创建的。这也因此确保了在 TTL 线程终止前，环路将被检测到。

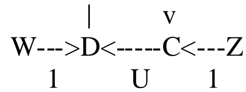
当扩展一带已知跳数的线程时，“改变颜色的线程扩展”规则也适用。

A.13.5 当一个带已知跳数的接收线程被扩展时，如果节点有一个输出线程，并且如果线程被接收的输入链接还不是 LSP 上的一个输入链接时，一个新线程被创建并被扩展。所有的输入线程都被合并到新线程中。这是 A.5.1 的一个异常情况。

A.14. 从环路中恢复

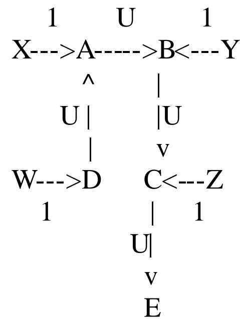
下面是我们的又一个拓扑例子（存在一个环）



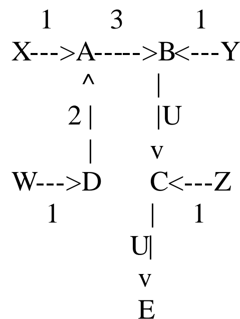


假设现在 C 的下一跳从 D 换到了某一其它节点 E，从而打破了环。为简明起见，我们将假定 E 是一个出口节点。

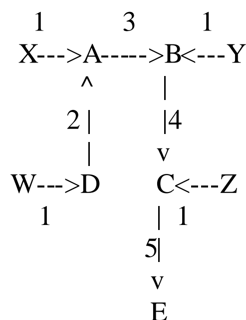
C 将从 D（9.1） 撤销掉它的输出 U-线程。它也将创建一个新线程(12.1)并分配一新颜色和跳数 U（C 的输入线程的最大跳数）给线程。C 还将合并它的其它两个输入线程到新线程中（12.2）并把新线程扩展至 E，结果如下：



当从 C 到 E 的线程回绕时，被合并的线程也回绕（8.4）。回绕过程一直进行下去直到回到叶节点。当这种情况发生进，当然 D 要指出它的输出线程跳数应该是 2，而不是 U 并且将产生变化（9.3）。结果，A 将指出它的输出跳数应该是 3，而不是 U 且也会产生变化。故而在将来某个时候，我们将明白下图所示的情况：



过一小段时间后，我们将看到如下所示的图：



因为所有的线程都是透明的，我们就完全建立了一个没有环路的路径。

A.15. 继续使用旧路径

由前面所讲，任何一个节点撤销一透明线程都没什么要求。现有的透明线程（已建立路径）能够继续被使用，即使一个新路径正在被建立。

如果前述过程被完成的话，那么某一节点可能同时有一个透明输出线程（旧路径）和一个带颜色线程（被建立的新路径）。这种情况只有在两个线程的下游链接不同时才会发生。当带颜色线程回绕时（变成透明色），以前的旧路径应该被撤销。