# ****不依赖IPv4/6 的FIB****

FIB 的主要功能如下：

## ****源的优先级****

一条路由可以通过不止实体或者源的方式加入到FIB中。所谓的源包括: API, CLI命令行接口，LISP(Locator/Identifier Separation Protocol RFC6830)，MAP等（所有的源可在fib\_entry.h中查看到）。每种源提供了该路由需要的确定FI(转发信息)。由于每种源使用不同的最佳路径和防止环路算法来确定FI，因而合并由多个源确定的FI将带来错误。FIB必须选择使用其中一个源提供的FI。这个选择是基于一个分配的静态优先级。例如：

如果是通过接口配置添加一个IP前缀(prefix)：

set interfacc address 192.168.1.1/24 GigE0

然后也通过命令行接口添加一个前缀

ip route 192.168.1.1/32 via 2.2.2.2/32

那么“接口”源会获胜，因而产生一条”本地”路由（命令行接口优先级低，它配置的路由竞争失败）。

FIB中总是安装竞争中获胜的源产生的FI，并且维护竞争失败的源产生的FI，以便在获胜的源失效被删除时可以使用竞争失败源的FI。

## ****adj-fib维护****

当ARP或ND在链路上发现一个邻居时，会产生一个表示邻居地址的临接（adjacency）信息。此时也需要在合适的FIB表中插入一条路由，这条路由关联了链路的VRF，是表示一个邻居的条目。该条目被成为adj-fib。Adj-fibs有一个专用的源：“ADJ”。

ADJ源比其他很多源的优先级都低。看以下的配置操作：

set interface address 192.168.1.1/32 GigE0

ip arp 192.168.1.2 GigE0 dead.dead.dead

ip route add 192.168.1.2 via 10.10.10.10 GigE1

这个操作将指向192.168.1.2的流量经过GigE1转发。因为通过控制平面添加的路由比通过ARP发现的临接更受青睐（优先级高）。控制平面有相关的认证机制，被认为是权威的源。

为了应对极端情况下临接注入，而导致极端情况的adj-fib添加到FIB中，FIB必须保证只有那些模糊覆盖前缀(less specific covering prefix，covering覆盖可以理解成掩码长度)是连通着的adj-fibs才可以加入到FIB中。这样就需要实现“覆盖跟踪”（cover tracking），也就是一条路由需要维护和它的模糊覆盖的依赖关系。当这个覆盖发生变化（如有了新的覆盖[掩码]路由）或者这个覆盖对应的路由信息发生变化，覆盖路由都要被通知到。

由于不支持重叠子网，因而adj-fib不可能有多个路径。

控制平面需要在接口改变VRF之前，删除接口的前缀配置。所以当下边的配置被应用时：

set interface address 192.168.1.1/32 GigE0

ip arp 192.168.1.2 GigE0 dead.dead.dead

set interface ip table GigE0 2（改变VRF）

这将得不到想要的结果。

## ****附加: 导出操作****

对以上adj-fib的进一步说明，考虑你以下的配置：

set interface address 192.168.1.1/24 GigE0

ip route add table 2 192.168.1.0/24 GigE0

指向192.168.1.2的流量在表（table）2中将会从GigE0上产生一个ARP请求。然而由于GigE0在表（table）0中，所有adj-fibs都会加入到FIB 0当中。所以，表2中子网的所有主机将不可达。为了解决这个问题，所有的adj-fib和本地前缀(prefixes)从表 0（称作导出表）中被导出（拷贝），导入到表 2（称作导入表）中。一个导出表可以导出到多个导入表中。

## ****递归路由解析****

一项递归路由有以下的形式：

1.1.1.1/32 via 10.10.10.10

也就是没有指定路由的出接口。为了将流量转发到1.1.1.1/32，FIB 必须首先确定如何将流量转发到10.10.10.10/32。这就是递归解析。

像常规解析一样，递归解析使用最长前缀匹配寻找中转地址10.10.10.10。注意，路由项的中转地址（via后边的地址）只能是一个地址（32位掩码或IPV6的128位掩码），不允许是有更短掩码的前缀（这种情况没有意义）。

FIB中用于解析递归路由的条目被称为中转条目（via-entry）。由于递归解析使用最长前缀匹配，当FIB中有其他路由项加入时，中转条目可能会发生变化。考虑上边递归路由，以及下边的非递归路由：

10.10.10.0/24 via 192.168.16.1 GigE0

条目10.10.10.0/24将会被选中为由于解析的中转条目。如果该条目被修改，也就是说：

10.10.10.0/24 via 192.16.1.3 GigE0

那么指向1.1.1.1/32的报文，必须被送往新的下一跳。   
现在考虑新增一条配置；

10.10.10.0/28 via 192.168.16.2 GigE0

更为精确的/28位掩码，是更好的最长前缀匹配，因而它就变成的中转条目。删除/28的条目，意味着/24位的条目将重新变为中转条目。这种变化的跟踪在FIB递归解析中是必须的。当中转条目的转发信息发生变化时，使用回退操作来更新递归路由的依赖。当新的路由项加入到路由表时，覆盖跟踪（cover tracking）特性会为中转条目路由提供必要的通知。

为1.1.1.1/32生成的邻接是一个递归邻接，它的下一个邻接将由中转条目贡献。维护递归邻接的有效性也是FIB必须要求的。

## ****避免递归产生的环****

考虑一下这组路由：

1.1.1.1/32 via 2.2.2.2

2.2.2.2/32 via 3.3.3.3

3.3.3.3/32 via 1.1.1.1

以上被称作一个递归环——所有的递归路由条目都没有被解析，因为他们没有一个解析出了邻接；然而所有每一条路由都被解析了，因为他的中转条目是确定的。非常重要的一点是，我们必须清楚的意识到控制平面对象和数据平面对象的区别（更多细节参考实现部分）。控制平面对象必须允许环的形成（也就是图变成了循环的），但是数据平面必须不能允许环的形成，否则报文无限循环，永远也不可能从设备中发送出去了——这将引起灾难。控制平面必须允许环形成，因为当环被打破时，所有的环中的成员需要被更新。形成环可以使依赖关系正确的建立。

VPP没有对递归的深度做限制，所以：

9.9.9.100/32 via 9.9.9.99

9.9.9.99/32 via 9.9.9.98

9.9.9.98/32 via 9.9.9.97

... turtles, turtles, turtles ...

9.9.9.1/32 via 10.10.10.10 Gig0

尽管有许多“接力条目”（turtles），但这样的配置是支持的。然后，当回走这个图时（本例中：从9.9.9.1/32回走到9.9.9.100/32,[9.9.9.1/32 变化是其他都要更新]），FIB需要区分这是一个很深的递归还是一个环。一个VPP使用的简单的办法是，限制回走的步数。VPP FIB限制是16。典型的BGP场景下，不会超过3（BGP Inter-AS option C）。

## ****快速收敛****

一个网络拓扑变化之后，会有一个收敛时间，它是从网络拓扑发生变化开始，直到路由器使用新的网络拓扑来完成第一次转发为止，所花费的时间。收敛时间是以下时间的总和；

* 探测到转发故障
* 计算新的最佳路径信息
* （从控制平面）下载新的最佳路径信息到数据平面
* 安装新的最佳路径到数据平面转发

最后两点是与VPP的架构相关联的。下载操作的API是二进制文件和批处理，细节这里不做讨论。这里没有硬件组件需要被重新编程，安装时间受限于内存分配、查表以及插入、访问等操作。

所谓快速收敛，它是基于这样一个技术理念，FIB用全心或部分精力用来重建转发表，同时以上的收敛操作同步进行。快速收敛技术可以进一步细分为PIC(Prefix Independent Convergence, 前缀独立收敛)和LFA-FRR(Loop Free Alternate path Fast re-route, 又是也成做IP-FRR) 分别影响递归和非递归路由。

## ****LFA-FRR****

考虑下边的网络拓扑：

C

/ \

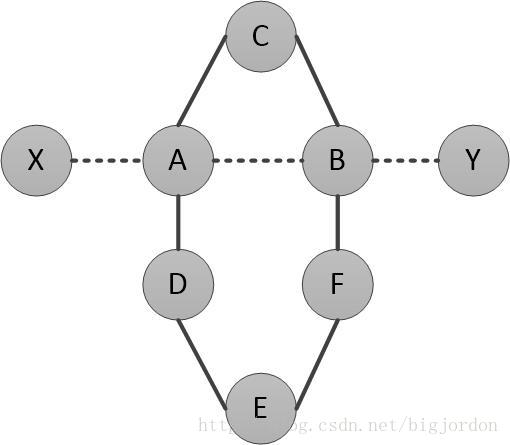
X -- A --- B - Y

| |

D F

\ /

E



所有的链路都有相同的开销，流量从X到Y。最佳路径是X-A-B-Y。还有两条可选的路径。一条通过C，一条通过E。如果其它两条可选路径上的路由器不会将流量转发的发送者，那个这两条路径是无环的。考虑路由器C，它到Y的最佳路径是通过B，所以如果A转发目的到Y的流量给C，那么C也会将流量转发给B——因而这是一个无环路径。相反地，考虑路由器D，D到Y的最短路径是通过A，如果A将目的为Y的流量转发给D，那么D将会把流量回送给A。这不是一个无环的路径。有几点是需要注意的：

* 我们考虑的是提前失效的路由拓扑
* A和B之间的所有等价路径都是LFA路径
* 为了A计算LFA路径，A必须知道，从D的角度来看到Y的最佳路径。这些计算限制了路由协议必须要了解整个网路拓扑的全貌，也就是像OSPF或SDN控制器这样的链路状态数据库协议。LFA保证了前缀是非递归的。

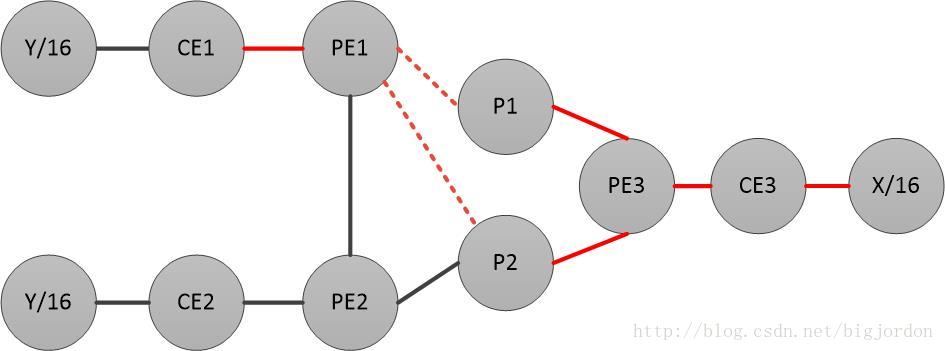
LFA被指定为一对一冗余的；一个首选路径只有一个LFA（又称备份）路径。在我的理解，这种限制是处于计算复杂性和容量规划的考虑，相对使用一对多的冗余而言。

当时间链路A-B失效时，可以使用通过C的可选路径。为了提供快速的失效备援功能，控制平面将会将首选路径和备份路径一起下载到FIB当中。这样的话就需要FIB实现失效备援（切换），可以从首选路径尽快地切换到备份路径上，并且不能有任何控制平面的介入。期望值是切换时间小于50毫秒（可以从VOIP QoS得到的值）。值得注意的是，切换时间依然包含链路故障检测时间，在某些环境下它是一个主要的因素。故障检测可以是链路失效（down），这会影响多路访问接口的多路径；也可能是通过特定路径的心跳（也就是BFD）。

****现在VPP还不支持LFA****，也就是它还不支持让一条路有上同时有首选路径又有备份路径。但是它支持ECMP（Equal-Cost Multipath），VPP FIB被设计成可以从ECMP集合中，快速删除失效路径。但是它不会将描述了受保护资源的共享对象插入到转发对象图中，因为这将产生转发/性能开销。失效备援时间是依赖路由的数量的。更多细节参看下边的实现部分。

## ****PIC(Prefix Independent Convergence)****

PIC指的是收敛时间应该独立于那些受链接故障影响的前缀(prefix)/路由。因而当网络中有大量的前缀（IP+掩码），也就是BGP网络，以及他们的递归前缀时，PIC就十分合适了。有几种不同类型的PIC，涵盖了不同网络位置的保护和故障场景。



CE = 客户边缘路由（customer edge）, PE = 供应商边界路由（provider edge）。   
external-BGP在客户和供应商之间运行，internal-BGP 在供应商之间运行。

### ****1）iBGP PIC-core****

考虑通过CE3,从CE1到X/16的流量。在PE1上的路由；

X/16 (以及成百上千和它相似的网络)

via PE3

并且

PE3/32（回环地址）

via 10.0.0.1 Link0 (this is P1)

via 10.1.1.1 Link1 (this is P2)

故障是link0和link1失效。

在所有的PIC场景中，为了提供前缀独立收敛（PIC）功能，FIB中指向X/16的路由（以及其他所有经由PE3的路由），都不应该被更新。FIB需要更新一个被所有路由共享的对象。一旦这个共享对象被更新了，所有使用这个共享对象的路由也就立即更新，并用上新的转发信息。这个例子中，这个共享对象是解析如何到达PE3的解析路由。一旦经由PE3的路由通过IGP（OSPF）被更新了，那些所有通过它来解析的递归路由也就被更新了。VPP FIB通过递归邻接（recursive-adjacency）实现这种场景。X/16以及它的兄妹路由共享一个递归临接，它是也连接到/指向一个普通的临接，这个临接由目的为PE3的路由贡献。一旦这个共享的递归邻接重新连接了，所有的路由将使用新的转发信息。如下边所示：

故障前：

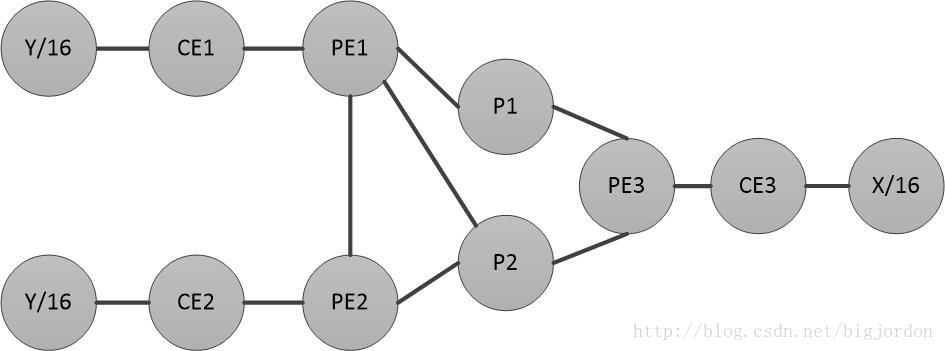
X/16 --> R-ADJ-1 --> ADJ-1-PE3 (multi-path via P1 and P2)

故障后：

X/16 --> R-ADJ-1 --> ADJ-2-PE3 (single path via P1)

R-ADJ-1（递归邻接）依然残留在转发图中，因此X/16（和他的兄妹）没有更新。由于X/16和它的兄妹网络共享共同的路径链（path-list），因而它们共享递归邻接。正是这个路径链贡献了递归邻接（详见下部分的描述）。

### ****2）iBGP PIC-edge****



流量从CE3到Y/16。在PE3上有以下路由配置；

Y/16 (以及成百上千和它类似的网络)

via PE1

via PE2

和

PE1/32 (PE1's loopback address)

via 10.0.2.2 Link0 (this is P1)

PE2/32 (PE2's loopback address)

via 10.0.3.3 Link1 (this is P2)

网络故障是到PE2网路不可达。这可能是链路P2-PE2引起，也可能是PE2节点引起。故障检测可以通过撤回PE2的回环路由，或其他形式的故障检查（BFD）方法。

VPP FIB 又一次发挥了PIC能力。它通过使用共享的递归邻接。Y/16和它的兄妹网络又一次共享了路径链{PE1，PE2}，这个路径链会产生一个多路径递归临接，也就是多路径邻接的每一个选择是另外一个邻接；

Y/16 -> RM-ADJ --> ADJ1 (for PE1)

--> ADJ2 (for PE2)

当到PE1的路由被撤销的时候多路径递归邻接更新为：

Y/16 --> RM-ADJ --> ADJ1 (for PE1)

--> ADJ1 (for PE1)

ECMP中的两个选择是一样的，因此所有的流量都转发到PE1去了。最终控制平面会下载更新一条通过PE1到Y/16的路由。此时的情形是：

Y/16 -> R-ADJ --> ADJ1 (for PE1)

在上述情景中，我们假设PE1和PE2是到Y/16的ECMP（等价多路径）。eBGP PIC核心（core）也阐述了这样的情形：一个PE是首选的，另一个是备用的，****但VPP FIB目前不支持这种实现****。

### ****3）eBGP PIC 边缘****

流量从CE3到Y/16。在PE1上配置的路由为:

Y/16 (and hundreds of thousands of others like it)

via CE1 (primary首选)

via PE2 (backup备份)

and

CE1 (this is an adj-fib)

via 11.0.0.1 Link0 (this is CE1) << this is an adj-fib

PE2 (PE2's loopback address PE2的回环地址)

via 10.0.5.5 Link1 (this is link PE1-PE2)

在eBGP PIC-edge的场景下，首选和备份路径是BGP计算出来的。每个对等端配置成总是将它的最好外部路径通告给它的iBGP对等端。备选路径将流量从备份PE发送到可替代的PE的核心。一个PE可能有多外部路径，也就是有多个直连CE；它也可能有多个备份PE，但是这两种备份没有关系。因而，不像LFA-FRR，这里的收敛模型是N-M；N个首先路径并有M个备份路径。只有当所有的首选路径失效时，才会切换到备份路径。注意，PE2必须有合适的配置来转发它从PE1接收到的外部路径上的流量。****VPP FIB不支持external-internal-BGP (eiBGP 外部到内部BGP)的负载均衡****。

就LFA-FRR来说，首选和备份路径的使用现在还没有被支持。但是递归多路径邻接以及一个合适受约束的，用来从首选和备份路径集中做出选择的哈希算法，会提供所需的共享对象，以及进一步的不依赖前缀规模的路径切换。

聪明的读者应该意识到，两种EGBP PIC场景只适用于无BGP核心（BGP free core）。

# ****特性与实现****

## ****快速收敛的实现选项有两个特性****

1）在数据路径中插入开关。这里的开关指的是受保护的资源。如果开关是打开的，那么首选路径生效，否则就是备份路径生效。测试数据路径中的开关，会带来性能的开销。一个数据包在转发过程中可能会遇到多个受保护资源。一旦受保护资源被检测到失效开关被触发，数据报就会在备份路径上转发，该方法使切换时间最小化。然而，这也带来一些性能的开销，并随着数据包遇到受保护资源的次数增加而增加。因而这种方法也是最适合LFA-FRR的，因为LFA-FRR受保护路由是非递归的（也就是会遇到较少的共享资源），期望切换时间更小（<50msecs）。

2）共享对象更新。确定数据路径中可以被多路径路由共享的对象，必须要知道是否满足快速收敛的要求。在受保护资源上生成一个对这些对象的依赖。当受保护资源失效时，每一个共享对象都要被更新，并且这个更新需要被共享对象的使用者一致地看到。这个方法在数据路径结构不发生变化时，不会有性能开销，但是但资源失效时，切换时间会随着处理处理资源失效工作时间的增长而增长。这种方式更适合于递归前缀（这种情况下，数据包会遇到多个受保护资源）和有更短切换时间的快速收敛技术（也就是PIC）。

## ****实现****

根据上边大纲的要求，不是所有FIB知道的路由的都被真正由于转发的（比如adj-fibs）。然而，万一情况发生变化，那些之前没有用上的路由就被用上了。这需要FIB维护两张表，每张都有VRF，AF（这张表用前缀索引）——转发表和非转发表[应该是这个意思]。

VPP为了DP（数据平面）速度，就希望在转发表中的查询结果直接就是ADJ。所以这两张表，一个包含所有的路由（一次查找产生一个fib\_entry\_t结构数据），另一个只包含转发路由（一次查找产生一个ip\_adjacency\_t结构数据）。后者被用在DP。

这使用内存换取了转发性能。在VPP操作环境中是一个不错的交易。

注意这些表都是只是有前缀来作为键值的。邻接的键值是元组{next-hop, address (and its AF), interface, link/ether-type}。

考虑以下较少见但是合法的配置：

set int ip addr 10.0.0.1/24 Gig0

set ip arp Gig0 10.0.0.2 dead.dead.dead

# 这个子网中的主机使用更好下一跳来路由 (它避免了一个大的二层的域)

ip route add 10.0.0.2 Gig1 192.168.1.1

# 这个递归的应该使用 Gig1

ip route add 1.1.1.1/32 via 10.0.0.2

# 这个非递归的应该使用 Gig0

ip route add 2.2.2.2/32 via Gig0 10.0.0.2

对于最后一条路由，使用{Gig0, 10,0,0,2}在前缀表了查找路径不会产生正确的结果。为了修正这个问题我们需要一个单独的邻接表。

## ****FIB 数据结构****

fib\_entry\_t:

- 标识一个路由项

- 有一个前缀

- 它维护了由不同的源得到的路径链的一个数组

- 最高优先级的源产生的路径链将得到一个邻接，它将这个邻接安装到转发表

fib\_path\_list\_t:

- 路径的链结构

- 路径链可能会被多个FIB实例共享。路径链被保存在数据库（DB）中。关键是多个路径相结合的描述。当路径链有助于收敛时，我们共享这个数据链。往数据库中加入那些从来不被共享数据链，或者不被前缀（这些前缀不隶属于PIC）共享的路径链，将会很不必要地增大数据库的体积，同时也会由于哈希碰撞增加搜索时间。

- 路径链为转发表中的实例提供合适的邻接。邻接可以是常规的，多路径的或者是递归的，这依赖于路径的数量和类型。

- 考虑到路径链是共享的，只需要创建一个实例来保存他们例。像这样的多路径邻接，它不需要单独的数据库。

有递归路径的路径链，以及递归邻接它们一同贡献了快速收敛体系主体的形成（正如前边描述的一样）。

fib\_path\_t:

- 如何转发流量的描述 (i.e. via {Gig1, K}).

- 路径描述了如何转发的意图。这和路径是如何迭代的不同。也就是说它可能根本就不迭代（接口被删除或down掉）。

- 路径有不同的类型，最显著的是递归和非递归型。

- fib\_path\_t贡献于合适的邻接对象。正是这份贡献，数据库图/链才会被建立。

- 如果一个路径是递归的，并且一个递归环被删除了，那么这个路径会贡献于这个特殊的被丢弃的（DROP）邻接。以这种方式，当控制平面图环路时，数据平面的图不会。

我们建立这些对象的图：

fib\_entry\_t -> fib\_path\_list\_t -> fib\_path\_t -> ...

对于递归路径（也就是fib\_path\_t后又接了fib\_entry\_t）：

fib\_path\_t -> fib\_entry\_t -> ....

对于非递归路径（fib\_path\_t后直接是接口）：

fib\_path\_t -> ip\_adjacency\_t -> interface

这些作为控制平面FIB的一个组成部分的对象，被用于表示路由的解析。整体来看，这个被称为控制平面图。有一个单独的数据平面图用于表示数据包的转发。在数据平面图中每一个对象表示一个动作，当数据包穿过这个图示，这个动作作用到这个包上。举例，在转发表中查找一个IP会得到以下的图：

recursive-adj --> multi-path-adj --> interface\_A

--> interface\_B

一个数据包穿过这个FIB数据平面图将会同时穿过一个VPP节点（函数）图：

ipX\_recursive --> ipX\_rewrite --> interface\_A\_tx --> etc

在FIB图中对象的分类是这样的，考虑：

A -->

B --> D

C -->

其中A,B,C是（比如）通过D解析到的路由。

父辈：D是A,B,C的父辈

孩子：A,B,C是D的孩子

兄妹：A,B,C互为兄妹。

所有FIB中的共享对象都有引用计数。这些对象的使用者需要使用add\_lcok/unlock语义（其中一个可能使用malloc/free）。

## ****遍历****

很必要对图做正向（从路由条目entry到接口）遍历，来实施分解或生成一个递归邻接；以及反向（从接口到路由条目）遍历，来实施更新，这个更新也就是当接口状态发生变化或者当递归路由解析更新发生时。   
正向遍历就是简单地通过一个对象的父辈指针来访问它的父辈对象。对于有多个父辈（如一个路径链）的对象，每一个父辈都会依次遍历。

为了支持反向遍历，需要维护对象之间的直接依赖关系，也就是在关系{A, B, C} –> D中，对象D会维护一串指向孩子{A, B, C}的指针。光秃秃的C语言指针是不被允许的，指针根据对象类型（如，entry, path-list,等）和索引来描述。这样就可以使对象从合适的池中恢复。为了实现大规模数据的快速收敛，需要维护一个链。当有数百万的或递归的前缀时，盲目的遍历表去查找实例（这个实例受给定的拓扑变化的影响）非常低效。最容易实现的优化是去除不需要的动作。所以，所有的反向遍历只需要通过那些直接由变化影响的对象。

PIC核心以及快速重路由依赖于FIB对接口状态变化的快速反应，并更新该接口使用的多路径邻接。一个例子如下：

E\_a -->

E\_b --> PL\_2 --> P\_a --> Interface\_A

... --> P\_c -\

E\_k --> \

Interface\_K

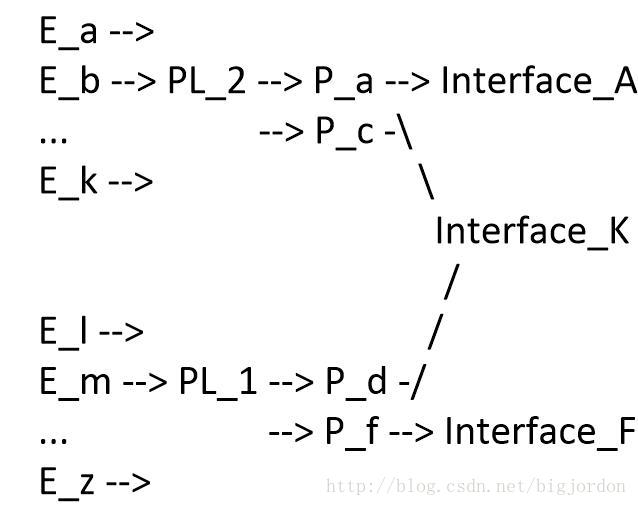
/

E\_l --> /

E\_m --> PL\_1 --> P\_d -/

... --> P\_f --> Interface\_F

E\_z -->



E = fib\_entry\_t   
PL = fib\_path\_list\_t   
P = fib\_path\_t

下标是随意写的只是为了区别不同的对象实例。

这个控制平面的图会产生一个数据平面的图：

M-ADJ-2 --> Interface\_A

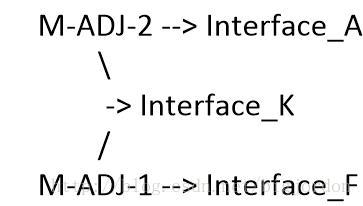
\

-> Interface\_K

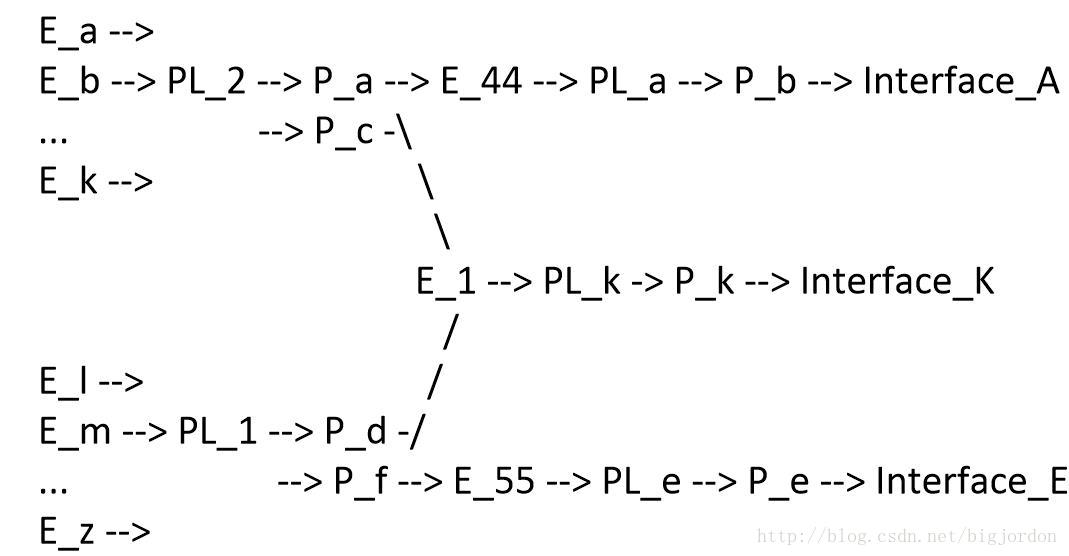
/

M-ADJ-1 --> Interface\_F

M-ADJ = multi-path-adjacency.



* 当接口K失效（down）时，反向遍历从它的控制平面依赖开始。反向遍历会到达PL\_1和PL\_2并且计算得到新的邻接（把接口K删除了）。遍历将向路由条目



故障场景是E\_1被移除，因而P\_c和P\_d变成不可解析的。为了实现PIC这两个共享的递归路径链PL\_1和PL\_2必须更新，把E\_1从递归多路径邻接上删除，这个更新操作在任何路由条目E\_a到E\_z更新之前。这意味着当更新在图中向反向传播时（从右向左），它必须做广度优先儿不能深度优先。注意，这个方法使得收敛时间依赖于路径链（path-list）的数目，以及PEs出口组合的数目——这个规模相比前缀的数目是非常低的，还是很令人满意的。

如果我们考虑这个图的另一部分，和上边很相似，有另外一个前缀E\_2在和E\_1类似的位置，同样有很多独立的孩子。我们可以想象一个很合理的场景，一个网络故障同时导致E\_1和E\_2不可达。这意味着在E\_1被删除后的更新下载到来之后，E\_2被删除后的更新也立即被下载。这要求在FIB在处理E\_1更新的时候不能把太多时间用在在反向回溯上，也就是反向回溯不可以一下到E\_a和他的兄妹那么远。因此，在反向回溯经过一代（广度优先）去更新所有的路径链（path-lists）后，必须停下（挂起），这样就可以处理更多的更新了。但更新队列为空的时候，被挂起的回溯就可以继续了。注意，多个更新影响同一个条目（E\_1）,这会触发多个相似的更新，这些被合并了，所有的孩子只更新一次。

多层递归PIC的存在也是个令人满意的特性。考虑上图的拓展，跟多的递归路由（E\_100 -> E\_200）加入到E\_a的孩子中：

E\_100 -->

E\_101 --> PL\_3 --> P\_j-\

... \

E\_199 --> E\_a -->

E\_b --> PL\_2 --> P\_a --> E\_44 --> ...etc..

... --> P\_c -\

E\_k \

E\_1 --> ...etc..

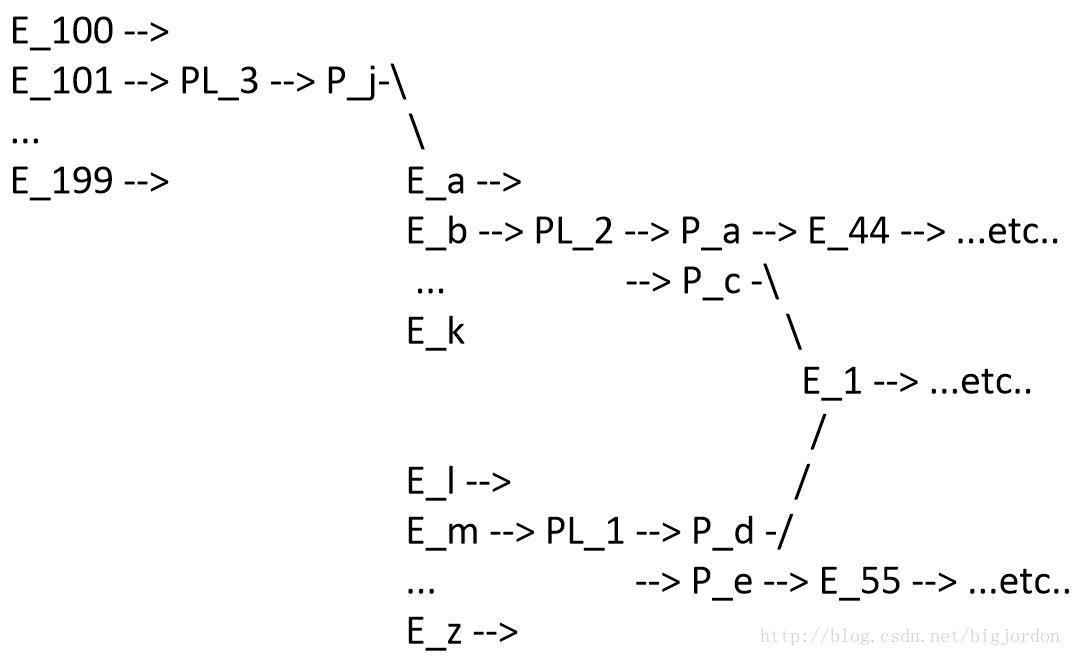
/

E\_l --> /

E\_m --> PL\_1 --> P\_d -/

... --> P\_e --> E\_55 --> ...etc..

E\_z -->



为了实现路由E\_100->E\_199的PIC，PL\_3需要在E\_b -> E\_z之前更新，深度优先在通过每一层时不会实现这点。遍历必须处理的更加智能。PL\_2的孩子们是排序的，因而那些有孩子的条目对象会出现在链的开始，没有孩子的在后边。在有还在的条目被遍历的时候，他们孩子的遍历被添加到后台队列。队列是优先队列。在广度优先处理独立条目对象E\_a到E\_k时，当到达第一个没有还在的条目（E\_b）时，遍历被挂起，并加入到队列后边。依循着优先化的方法，共享的路径链在所有非解析条目对象之前更新。   
处理更新的CPU/核/线程是处理反向遍历的相同的线程。处理比遍历有更新有更高的优先级，所以在有可用更新时，递归必须被打断/挂起。

****以下部分描述了，遍历其实不是上边描述的遍历****

在上图中E\_100是一个IP路由，然而VPP没有对独立FIB条目对象的类型做限制。FIB条目的孩子可以是GRE & VXLAN隧道端点，L2VPN LSPs 等。把所有类型加入到图中并扩展反向遍历，我们可以实现覆盖IP网络的快速收敛。

如果你认真阅读了上文你可能会想，”我不需要这些乱七八糟的，我只有一个我了解的路由，我只需要把他加进去“，那么fib\_table\_entry\_special\_add()就够了。