



(12) 发明专利

(10) 授权公告号 CN 110708241 B

(45) 授权公告日 2021. 10. 08

(21) 申请号 201910860198.3

(22) 申请日 2015.02.13

(65) 同一申请的已公布的文献号
申请公布号 CN 110708241 A

(43) 申请公布日 2020.01.17

(30) 优先权数据
61/939,487 2014.02.13 US

(62) 分案原申请数据
201580019119.6 2015.02.13

(73) 专利权人 洛克波特网络股份有限公司
地址 加拿大渥太华

(72) 发明人 D.奥普雷亚 A.卡塔纳
U.纽斯塔特

(74) 专利代理机构 北京市柳沈律师事务所
11105

代理人 陈丽来

(51) Int.Cl.
H04L 12/707 (2013.01)
H04L 12/721 (2013.01)

(56) 对比文件
US 7373543 B1,2008.05.13

审查员 张燕燕

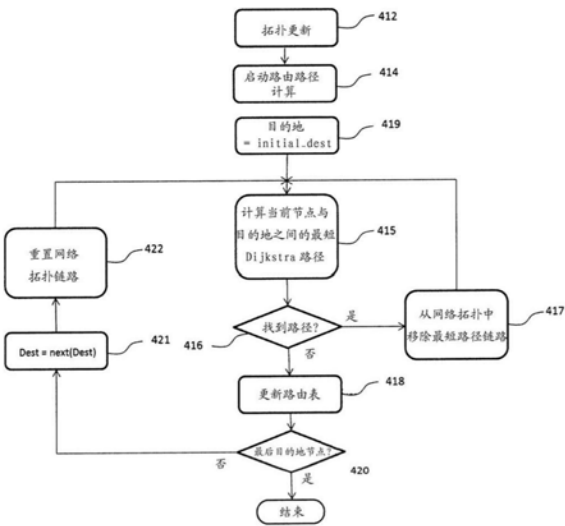
权利要求书2页 说明书14页 附图18页

(54) 发明名称

在分布式直接互连网络中路由分组的方法

(57) 摘要

本发明提供一种用于跨越具有低延迟、增加的吞吐量和业务分发的环形或高基数拓扑路由数据分组以避免热点发展的方法和设备。公开了一种在分布式直接互连网络中从源节点向目的地节点路由分组的方法，该方法包括以下步骤：发现所有节点和关联端口；更新数据库以包括网络拓扑中的所述节点和端口；计算从每个节点上的每个输出端口到所述拓扑中的每个其它节点的最短路径；在所述源节点的输出端口处将每个分组分段成微片；随着所述微片被分段，使用虫孔交换来沿着从所述源节点上的每个输出端口到所述目的地节点的最短路径分发所述微片，由此所述分组沿着所述网络拓扑中的替换的最大不相交路由被分发；以及在所述目的地节点处对所述分组进行重组和重排序，使得所述分组符合它们的原始次序/形式。



1. 一种在直接互连网络中从源节点向目的地节点路由分组的计算机实现的方法,该方法包括以下步骤:

在网络拓扑中发现所有节点以及每个节点上的所有输出端口;

将在所述网络拓扑中发现的节点和输出端口包括在拓扑数据库中以便允许所述节点和端口被包括在路径路由计算中;

基于包含在所述拓扑数据库中的那些节点和输出端口来计算从每个节点上的每个输出端口到所述网络拓扑中的每个其它节点的路径,其中每个这样的路径彼此不相交;

在每个节点上生成源路由数据库,其包含从每个节点上的每个输出端口到所述网络拓扑中的所有其它节点的最短最大程度不相交路径;

在所述源节点处接收一分组;

以循环方式或加权循环方式将所接收到的分组发送到选定的所述源节点的输出端口中的一个,由此所述接收到的分组此后沿着从所述源节点上的所选定的输出端口到所述目的地节点的所述最短最大程度不相交路径而被分发。

2. 根据权利要求1所述的计算机实现的方法,其中计算从每个节点上的每个输出端口到所述网络拓扑中的每个其它节点的所述路径的步骤包括计算不具有共同节点的不相交路径。

3. 根据权利要求1所述的计算机实现的方法,其中计算从每个节点上的每个输出端口到所述网络拓扑中的每个其它节点的所述路径的步骤包括计算不具有共同链路的不相交路径。

4. 一种在直接互连网络中从源节点向目的地节点路由分组的计算机实现的方法,该方法包括以下步骤:

在网络拓扑中发现所有节点以及每个节点上的所有输出端口;

将在所述网络拓扑中发现的节点和输出端口包括在拓扑数据库中以便允许所述节点和端口被包括在路径路由计算中;

基于包含在所述拓扑数据库中的那些节点和输出端口来计算从每个节点上的每个输出端口到所述网络拓扑中的每个其它节点的路径,其中每个这样的路径彼此不相交;

在每个节点上生成源路由数据库,其包含从每个节点上的每个输出端口到所述网络拓扑中的所有其它节点的多个最短最大程度不相交路径;

在所述源节点处接收多个分组;

以循环方式或加权循环方式将所接收到的多个分组发送到所述源节点的输出端口,由此所述接收到的多个分组此后沿着从所述源节点上的所述输出端口到所述目的地节点的所述多个最短最大程度不相交路径而被分发,使得所述多个分组从而沿着所述网络拓扑中的多个替换的路由而被分发。

5. 根据权利要求4所述的计算机实现的方法,其中计算从每个节点上的每个输出端口到所述网络拓扑中的每个其它节点的所述路径的步骤包括计算不具有共同节点的不相交路径。

6. 根据权利要求4所述的计算机实现的方法,其中计算从每个节点上的每个输出端口到所述网络拓扑中的每个其它节点的所述路径的步骤包括计算不具有共同链路的不相交路径。

7. 根据权利要求4所述的计算机实现的方法,其中所述分组还在所述目的地节点处被重新排序,使得所述分组符合它们在所述源节点处被接收的初始顺序。

8. 一种在直接互连网络中从源节点向目的地节点路由分组的计算机实现的方法,该方法包括以下步骤:

在网络拓扑中发现所有节点以及每个节点上的所有输出端口;

将在所述网络拓扑中发现的节点和输出端口包括在拓扑数据库中以便允许所述节点和端口被包括在路径路由计算中;

基于包含在所述拓扑数据库中的那些节点和输出端口来计算从每个节点上的每个输出端口到所述网络拓扑中的每个其它节点的路径,其中每个这样的路径彼此不相交;

在每个节点上生成源路由数据库,其包含从每个节点上的每个输出端口到所述网络拓扑中的所有其它节点的最短最大程度不相交路径;

在所述源节点处接收分组;

以循环方式或加权循环方式将所接收到的分组发送到所述源节点的输出端口,由此所述接收到的分组中的每一个此后在所述源节点的所述输出端口处被分段成微片并且沿着从所述源节点上的所述输出端口到所述目的地节点的所述最短最大程度不相交路径而被分发,使得所述分组从而沿着所述网络拓扑中的替换的路由而被分发。

9. 根据权利要求8所述的计算机实现的方法,其中计算从每个节点上的每个输出端口到所述网络拓扑中的每个其它节点的所述路径的步骤包括计算不具有共同链路或节点的不相交路径。

10. 根据权利要求9所述的计算机实现的方法,其中通过使用虫孔交换将所述微片转发到所述目的地节点。

11. 根据权利要求10所述的计算机实现的方法,其中所述微片还被重组成分组并且在所述目的地节点处被重新排序,使得所述分组符合它们在所述源节点处被接收的初始形式和顺序。

在分布式直接互连网络中路由分组的方法

[0001] 本申请是2015年2月13日提交的中国专利申请号为201580019119.6的题为“在分布式直接互连网络中路由分组的方法”的专利申请的分案。

技术领域

[0002] 本发明涉及用于数据中心和云数据中心服务器互连的计算机网络。特别地，本发明涉及用于在诸如“蜻蜓”布线结构的环形(torus)或高基数拓扑上实现的直接互连网络中路由分组的方法。

背景技术

[0003] 术语数据中心(DC)通常是指用来容纳全部通过大量的结构化电缆线路互连的大型计算机系统(常常被包含在容纳装置的机架上)及其关联组件的设施。云数据中心(CDC)是用来是指类似地存储实体的数据的大型的通常为备用设施的术语。

[0004] 网络交换机是链接网络设备以用于通信/处理目的的计算机联装置。换句话说，交换机是能够从连接到其上的任何设备接收消息并且将该消息发送到特定设备的电信设备，该特定设备是该消息将要被中继到的设备。网络交换机也通常被称为处理并路由数据的多端口网桥。这里，通过端口，我们是指交换机与它附连到的计算机/服务器/CPU之间的接口(用于电缆或插头的出口)。

[0005] 现今，DC和CDC通常使用一组二层交换机来实现数据中心联网。二层交换机在第2层(数据链路层)处处理并路由数据，所述数据链路层是在同一局域网上的节点(例如，服务器)或广域网中的相邻节点之间传送数据的协议层。然而，要解决的关键问题是如何构建大容量计算机网络，其能够承载包含非常大数目的端口(数千个)的非常大的聚合带宽(数千TB)、需要最小结构和空间(即，使对于用于容纳具有卡片机架的大量机柜的大房间的需要最小化)并且是容易地可扩展的以及可以有助于使功耗最小化。

[0006] 传统的网络拓扑实施例是基于按照如图1中所示的层次树结构而组织的完全独立的交换机。核心交换机2是具有非常大的交换容量的甚高速低计数端口。第二层使用聚合交换机4(具有大数目的端口的中等容量交换机)来实现，然而第三层使用较低速度、大端口计数(例如，四十/四十八)、低容量边缘交换机6来实现。通常边缘交换机是层2，然而聚合端口是层2和/或层3，并且核心交换机通常是层3。这个实施例在所提供的示例中在最多六跳链路(上至核心交换机2的三跳以及下至目的地服务器8的三跳)中提供任何服务器8到任何服务器连接性。出于冗余可靠性目的这样的层次结构也通常是重复的。例如，参考图1，在没有重复的情况下如果最右边的边缘交换机6出故障，则不存在到最右边的服务器8的连接性。至少，核心交换机2是重复的，因为核心交换机2的故障将产生总数据中心连接性故障。由于显然的原因，这个方法在解决将来CDC的挑战时具有显著的局限性。例如，因为每个交换机是完全自备式的，所以这增加复杂性、显著的楼层空间利用、易于人为错误的复杂电缆线路以及人工交换机配置/提供以及增加的能量成本。

[0007] 然而，已经做出许多尝试，以改进数据中心中的交换可扩展性、可靠性、容量和延

迟。例如,已经做出努力来通过使用统一控制平面(例如,来自Juniper Networks的QFabric系统交换机;参见例如<http://www.juniper.net/us/en/productservices/switching/qfabric-system/>)来实现更复杂的交换解决方案,但是这样的系统仍然使用并维持传统的层次架构。此外,考虑到系统用户以及要存储、存取和处理的数据的数目的指数增加,当确定计算机网络系统的性能要求时处理能力已成为最重要的因素。虽然服务器性能一直持续地改进,但是一个服务器不够强大到满足需要。这是并行处理的使用已变得至关重要的原因。结果,在多达80%的许多情况下,什么是占主导的北美业务流现在已主要成为东西方业务流。不管业务流的这种改变,网络架构尚未演进为对于这个模型来说是最佳的。它因此仍然是确定在并行处理通信期间CPU之间的交互的速度的通信网络(其互连计算节点(服务器))的拓扑。

[0008] 对于增加的东西方业务通信的这个需要导致较新的更平坦网络架构(例如,环形的/环形网络)的创建。环形互连系统是用于在并行计算机系统中以网状方式连接网络节点(服务器)的网络拓扑。环形拓扑能够具有布置在能够被虚拟化为阵列的2、3或更多(N)个维度上的节点,其中处理器/服务器连接到它们最近的邻居处理器/服务器,并且其中阵列的相对边缘上的处理器/服务器被连接。以这种方式,每个节点在N维环形配置中具有 $2N$ 个连接(图2提供了3-D环形互连的示例)。因为环形拓扑中的每个节点经由短电缆线路连接到相邻节点,所以在并行处理期间存在低网络延迟。实际上,环形拓扑以最小跳数提供对任何节点(服务器)的访问。例如,实现 $3 \times 3 \times 3 \times 4$ 结构(108个节点)的四维环形需要平均2.5跳以便提供任意点到任意点的连接性。图4提供 $6 \times 6 \times 2$ -D环形的示例,示出了从拐角节点1.6转向所有其它35个节点所需要的最小跳数。如所示,从节点1.6到达任何目的地所需要的跳数能够被标绘为在3跳(10个节点)处具有峰并且通常分别在5跳(4个节点)和1跳(4个节点)处具有尾的贝尔曲线。

[0009] 遗憾的是,大型环形网络实施例尚未实际用于DC或CDC的商业部署,因为大型实施例可能花费数月以来构建,电缆线路可能是复杂的(对于每个节点来说 $2N$ 个连接),并且它们在扩展是必要的情况下修改可能是代价高且麻烦的。然而,在对于处理能力的需要已经比商业缺点重的情况下,环形拓扑在超级计算机中的实现一直是非常成功的。在这方面,IBM的蓝色基因超级计算机提供3-D环形互连网络的示例,其中64个机柜容纳65,536个节点(131,072个CPU)以提供每秒千万亿次浮点运算(petaflops)处理能力(参见插图的图3),然而富士的PRIMEHPC FX10超级计算机系统是被容纳在包括98,304个节点的1,024个机架中的6-D环形互连的示例。虽然以上示例处理环形拓扑,但是它们同样地适用于其它平坦网络拓扑。

[0010] 本发明更具体地处理在环形或高基数网络结构中从节点到节点的数据分组遍历和路由的重要问题。在这方面,它是确定数据的分组在网络中从源转向目的地所采取的实际路径的路由。出于本文目的,延迟是指分组在网络中到达目的地所花费的时间,并且通常是从当报头到达源节点的输入端时到当它到达目的地节点的输入端时来测量的。跳计数是指在源与目的地之间遍历(经过)的链路或节点的数目,并且表示用于确定延迟的近似值。吞吐量是按照比特/每秒(bits/sec)测量的网络每输入端口/节点接受的数据速率。

[0011] 当路由时的有用目标是为了在节点当中均匀地分发业务(负荷均衡)以便避免热点发展(使用/需求已超过期望的或可接受的阈值的通路或节点区域)并且为了使争用(当

两个或更多个节点试图同时在同一条电线或路径上发送消息或分组时)最小化,从而改进网络延迟和吞吐量。为此选择的路由影响从节点到节点的跳数,并且可能在路由未被优化时潜在地甚至由此影响能量消耗。

[0012] 网络操作的拓扑还无疑影响延迟,因为拓扑影响节点之间的平均最小跳计数和距离。例如,在环形中,不仅存在分组能够为了到达目的地而采取的数个路径(即,在存在“路径分集”的环形中),而且在任何源和目的地对之间也存在多个最小长度路径。作为示例,图5示出了分组能够为了在2-D环形网格中从节点S11转向节点D12而采取的三个最小路由(3跳)的示例,同时也示出了5跳的较长的第四路由。通过路由完成的路径计算是基本上仅基于拓扑完成的-源路由在逐跳基础上动态地基于分组源-目的地对。

[0013] 利用路径分集的路由方法学在网络中具有更好的故障容错和更好的负荷均衡。然而,路由方法学不总是实现这样的目标,并且能够通常被划分成三个类别:确定性、无关和自适应。确定性路由是指一对给定节点之间的路由不顾网络的当前状态(即,不顾网络业务量)都被提前确定的事实。维序路由(DOR)是确定性路由的示例,其中从节点A到节点B的所有消息将总是遍历同一路径。具体地,消息逐个维度遍历(X-Y路由),从而在切换到下一个维度之前到达在一个维度上与其目的地匹配的纵坐标。作为一个示例,图6能够被用来示出DOR,其中分组首先沿着第一维度(X)行进与从节点1至5到9所需要的一样远,后面是沿着第二维度(Y)行进到目的地节点10。尽管这样的路由通常易于实现并且无死锁(死锁是指沿着从源到目的地的通路存在死循环的情形),然而未利用路径分集并且因此负荷均衡较差。

[0014] “无关(oblivious)”的路由算法是其中不顾网络的当前状态随机地做出路由判定的那些算法(确定性路由是无关路由的子集)。尽管这意味着无关路由可能实现起来简单,然而它也不能适于业务和网络情况。众所周知的无关路由方法的示例是Valiant算法(为本领域的技术人员所知)。在这个方法中,从节点A发送到节点B的分组被首先从A发送到随机地选择的中间节点X(一跳),并且然后从X发送到B。再次参考图6,Valiant算法能够将节点2随机地选择为从源节点1到目的地节点10的中间节点,意味着路径1-2-3-7-11-10例如能够被用于路由目的。这通常使任何业务模式随机化,并且因为所有模式似乎是均匀随机的,所以网络负荷是十分均衡的。事实上,Valiant算法一直被认为能够通常在几乎任何拓扑上针对任何业务模式使负荷均衡。然而,一个问题是Valiant路由通常是非最小的(最小路由是在源与目的地之间需要最小跳数的路径),这常常导致显著的跳计数增加,并且这进一步增加网络延迟并且可以潜在地增加能量消耗。然而,诸如在拥塞最小的网络中存在例外。非最小路由可能随着附加节点被遍历而显著地增加延迟并且可能增加功耗;另一方面,在经历拥塞的网络中,非最小路由可以实际上帮助避免节点或热点,并且从而实际上导致较低延迟。

[0015] 如果最小路由是期望的或必要的,则能够修改Valiant算法以通过指定中间节点必须位于最小象限内来将其随机判定限于最小路由/最短路径。作为示例,参考图6,节点1(具有节点10的目的地)的最小象限将包含节点2、5和0,并且将导致3的常用跳计数。

[0016] Valiant算法提供将降低热点发展的可能性的某种水平的路径选择随机化。然而,存在另一个难题,在影响这些效率情况下-Valiant路由仅在与DOR路由相结合地使用时才无死锁,其本身在负荷均衡和热点避免方面是失败的。

[0017] 自适应路由算法是其中路由判定是基于网络或网络业务的状态的那些算法。它们

通常涉及流控机制,并且在这方面常常使用缓冲器占用。自适应路由能够采用全局节点信息(其是成本性能合理的),或者能够使用来自仅本地节点的信息,包括例如用于测量网络拥塞的队列占用。使用单独来自本地节点的信息的问题是这有时可能导致次优选择。自适应路由也可能限于最小路径,或者它能够通过采用具有活锁(即,与分组行进未进行到目的地的死锁类似的情形;常常是资源缺乏的结果)的潜在性的非最小路径从而是完全自适应的(即,在采取最短路径时没有限制)。这有时能够通过每分组允许特定数目的误编路由并且通过将更高优先级给予被误编路由许多次的分组来克服。自适应路由的另一问题是它可能对保存数据分组排序造成问题-分组需要按照相同的次序到达目的地或者否则你需要实现分组重排序机制。

[0018] 最后,重要的是提及能够通过源表或本地表来实现路由。利用源表,整个路由在源处被指定,并且能够被嵌入到分组报头中。因此使延迟最小化了,因为路由不必在每个节点处被查找或者被逐跳路由。还能够使源表针对每目的地指定多个路由以能够管理故障,并且,在路由被随机地选择(即,无关路由)时,能够提高负荷均衡。利用本地节点表,另一方面,较小的路由表被采用。然而,分组将采取的下一个步骤在每个节点处被确定,并且这增加了每跳延迟。

[0019] 本发明设法克服现有技术中的缺陷并且改进在环形或高基数网络拓扑中路由分组的已知方法。

发明内容

[0020] 在一个方面中,本发明提供一种用于在环形网格或高基数拓扑上路由数据分组以提供低延迟、增加的吞吐量以及避免热点和死锁的新颖方法。

[0021] 在又一个方面中,本发明提供一种用于在环形网格或高基数结构中实现业务去相关(随机化)、带宽分配中增加的效率、负荷均衡和流量工程(traffic engineering)的方法。

[0022] 在一个实施例中,本发明提供一种在直接互连网络中从源节点向目的地节点路由分组的计算机实现的方法,该方法包括以下步骤:在网络拓扑中发现所有节点以及每个节点上的所有输出端口;将在所述网络拓扑中发现的节点和输出端口包括在拓扑数据库中以便允许所述节点和端口被包括在最短路径路由计算中;基于包含在所述拓扑数据库中的那些节点和输出端口来计算从每个节点上的每个输出端口到所述网络拓扑中的每个其它节点的最短路径;在每个节点上生成源路由数据库,其包含从所述每个节点上的每个输出端口到所述网络拓扑中的所有其它节点的最短路径;在所述源节点处接收分组;以循环方式或加权循环方式将所接收到的分组发送到所述源节点的输出端口,由此所述接收到的分组中的每一个此后在所述源节点的所述输出端口处被分段成微片(flits)并且沿着从所述源节点上的所述输出端口到所述目的地节点的多个最短路径被分发,使得所述分组从而沿着所述网络拓扑中的多个替换的路由而被分发;以及在所述目的地节点处对所述分组进行重组和重排序,使得所述分组符合它们的原始形式和次序。

[0023] 在另一个实施例中,本发明提供一种路由分组的方法,其中使用虫孔(wormhole)交换将微片转发到目的地节点。

附图说明

- [0024] 现在将参考附图通过示例对本发明的实施例进行描述,附图中:
- [0025] 图1是传统的数据中心网络实施例的高级视图;
- [0026] 图2是具有8个节点的3维环形互连的图;
- [0027] 图3是示出了采用环形架构的IBM蓝色基因处理单元的层次的图;
- [0028] 图4是显示了从拐角节点(1.6)开始到达35个节点中的任一个所需要的最小跳数的6x 6个2-D折叠环形中的36个节点的高级图;
- [0029] 图5是显示了在节点S的每个端口($x+$ 、 $x-$ 、 $y+$ 、 $y-$)上从节点S开始到节点D的数个路由的4x 4 2-D环形实施例的图;
- [0030] 图6是用于帮助示出各种路由算法如何能够从节点向节点路由分组的简单网络拓扑的图;
- [0031] 图7是根据本发明的系统的具有所有端口的高速网络接口卡的高级图;
- [0032] 图8是示出了根据本发明的系统的关键功能块的主机网络接口(HNI)卡的高级图;
- [0033] 图9是跨越环形网格实现分组传送所需要的分组分段的高级视图;
- [0034] 图10示出了在36节点2-D环形结构中从节点S(2.4)上的每个端口开始到节点D(5.2)上的每个端口的路由和跳数的图;
- [0035] 图11示出了根据本发明的呈现软件组件TD、SR、WS、FC的高级图;
- [0036] 图12示出了根据本发明的分组分段的过程的流程图;
- [0037] 图13显示了通过分组分段实现微片生成的伪代码;
- [0038] 图14显示了根据本发明的用于从每个端口准备源路由的处理步骤的流程图;
- [0039] 图15显示了根据本发明的实现路径计算的伪代码;
- [0040] 图16显示了根据本发明的通过路径计算生成的源路由表格式;
- [0041] 图17显示了根据本发明的在每个端口上传送“微片”的过程的流程图;
- [0042] 图18显示了根据本发明的在每个端口上传送“微片”的虫孔交换过程;
- [0043] 图19显示了根据本发明的实现如图17中的流程图中所描述的微片传送的伪代码;
- [0044] 图20显示了用于发现为了拓扑更新而触发拓扑发现过程的节点或链路的流程图;
- [0045] 图21显示了用于描述拓扑数据库更新的流程图;
- [0046] 图22示出了节点和/或链路故障检测和恢复的过程以及拓扑更新的过程的流程图;以及
- [0047] 图23显示了根据本发明的实现链路和/或节点故障检测的伪代码;

具体实施方式

[0048] 本发明利用环形网格或高基数布线来实现用于数据中心应用的直接互连交换。特别地,本发明尤其可与2014年8月29日提交的标题为“Method and Apparatus to Manage the Direct Interconnect Switch Wiring and Growth in Computer Networks”的相关PCT专利申请号PCT/CA2014/000652中所描述的环形系统一起使用,该PCT专利申请的公开内容通过引用结合在本文中。这样的架构能够在单个交换域中提供具有成千上万个服务器的高性能网络互连。

[0049] 控制交换机的软件遵循软件定义网络模型。交换机控制平面(交换机OS)优选地在

单独的服务器上运行,但是与在每个PCIe卡上运行的分布式处理紧密地相互作用。参考图11,关键软件组件包括拓扑发现(TD)、跨越拓扑的分组路由(SR)、虫孔交换(WS)、流控(FC)机制以及交换机状态监视和交换机管理。TD组件负责基本上发现节点存在(或不存在)于环形网格中(并且从而允许基于节点存在更新可能的路径);SR组件负责源路由计算(从源节点的每个端口到任意目的地节点可用的所有路径);WS组件负责基于路径向量的跨越环形的微片操纵,该路径向量被包括在微片报头中,具有最小延迟(在每个节点处的贯通模型);FC组件负责环形链路利用并且控制到源的消息以便通过输入排队机制来调节业务;交换机状态监视组件负责业务均衡、在故障条件下的业务重新路由以及对于交换机增长的支持;并且交换机管理组件负责维护交换机的完整映像,包括所有互连的服务器、到外部世界的所有互连的I/O链路以及连接性状态。

[0050] 本发明主要涉及本发明的新颖拓扑发现(TD组件)和路由协议(SR组件)-尤其可用于实现在PCT专利申请号PCT/CA2014/000652中所公开的新颖交换机架构上的关键功能。虽然通过环形结构路由数据分组的传统的实施方式做出隐式假定,即业务跨越目的地节点被随机地分发,但是由于跨越环形传播并且有能力完全禁用跨越环形网格传送分组的能力的热点发展,实际的实施例已经实际上在最大容量下的单个源/单个目的地连续分组流的情况下显示出了交换机故障。本发明因此更具体地参考用于诸如环形和高基数网络的直接互连网络拓扑的路由优化和实现,以便对于任何类型的业务解决性能问题和热点发展(无论是从任何源到任何目的地的大型流(高度相关的)还是到随机分布式目的地的业务(典型的互联网业务))。与如上面先前所描述的在环形中路由分组的已知方法相反,本发明描述了发现并更新拓扑中的所有节点的方法,以及向每个目的地节点路由分组的方法,其中最短路在每个节点处在每个输出端口(例如, $x+$ 、 $x-$ 、 $y+$ 、 $y-$ 、 $z+$ 、 $z-$ 、 $w+$ 和 $w-$)上被计算,并且由此分组被以循环方式或加权循环方式沿着从每个输出端口起的路径分发(即,促进多个替换的路径)以增加带宽和业务分发从而避免热点。流控(FC)机制被用来将用于分组路由的输出端口选择从简单循环连续的源节点输出端口选择(即,首先端口 $x+$,接下来 $x-$,然后 $y+$ 、 $y-$ 、 $z+$ 、 $z-$ 、 $w+$ 和 $w-$)改变为由FC标识为忙碌的链路被跳过的加权循环,或者那些端口上的业务量被降低至最大链路容量的低百分比(例如,10%)。

[0051] 数据分组重排序被实现在节点目的地处。这样的方法能够特别利用PCT专利申请号PCT/CA2014/000652中讨论并且在本文中进一步讨论的节点(服务器)发现的新颖方法。本发明因此解决针对诸如环形或高基数网络的直接互连网络拓扑的分组路由优化的问题,以便对于任何类型的业务解决性能问题和热点发展,而不管那是从任何源到任何目的地的大型流(高度相关的)还是到随机分布式目的地的业务(典型的互联网业务)。

[0052] 首先,图7显示了具有如下所有端口的高速网络接口卡10的高级别框图:用于访问外部世界的I/O端口13(N-S接口)、4D环形互连端口14、用于访问计算机服务器的PCIe端口15,以及控件19。本领域的技术人员将理解,本发明能够与具有更少或更多端口的HNI卡一起并且与其它环形或高基数拓扑(例如诸如“蜻蜓”)相关联地工作。

[0053] 图8利用关键功能块示出了根据本发明的系统的HNI卡10的高级别框图,所述关键功能块包括具有RAM 37和ROM 38的处理器31、交换块32、物理接口装置36、环形互连端口14、I/O端口13以及PCIe端口15。这个实施例的物理实现基于具有附连存储器的多核处理器以及用于实现环形互连的PHY装置。使用处理器的支持多个队列操纵的硬件加速辅助能力

来在软件中实现交换功能。本领域的技术人员将理解,本发明也将在物理实现方式不同的情况下(即,在交换功能加速用FPGA或ASIC实现的情况下)工作,或者,处理器不必驻留在与用于交换功能的卡相同的卡上。

[0054] 图10显示了根据本发明的用于在说明分组路由时使用的简单2D环形布线图。如图所示,该结构是其中每个连接的长度自始至终相等的折叠2D环形。此图中的每个节点表示经由被容纳在服务器中的PCIe交换卡(如上描述的)互连的服务器。该图示出了如在源节点S处计算的在各个维度($x+$ 、 $x-$ 、 $y+$ 、 $y-$)上开始从源节点S(2.4)的端口到目的地节点D(5.2)的最短最大程度不相交路径(无公共链路或节点)。在这方面,图10将三个计算的路径和路由向量示出为具有5跳的最小跳计数(从(S) 2.4起: $(x+, x+, x+, y-, y-)$; 从(S) 2.4起: $(y-, y-, x+, x+, x+)$; 并且从(S) 2.4起: $(x-, x-, y-, y-, x-)$),然而另一路径被计算为具有7跳的非最小跳计数(从(S) 2.4起: $(y+, y+, y+, x+, x+, x+, y+)$)。假定在源S节点与D节点之间有大业务量,分组/微片将被以循环方式或加权循环方式,沿着从每个输出端口起的路径在每个源节点S输出端口开始到节点D的最短路径上发送到目的地节点,从而一方面使用多个路径来增加带宽并且分发业务以避免热点发展(在源节点处对输出端口的加权循环选择将增加沿着特定路径的分组的数目)。第二替换路由也可以被计算并且被存储在每个节点处,由此,除第一跳之外,路由提供替换的单独的路径以在必要时在链路或节点故障的情况下帮助克服网络拥塞和热点。本方法出于负荷均衡目的而跨越环形或高基数网络分发业务,但是能够引入分组重排序。例如,因为在 $y+$ 上开始的路径(图10)比最小路径(5跳)长2跳,所以在这个路径上发送的所有分组将以可能产生分组重排序问题的额外延迟到达。根据本发明,因此必须在目的地节点处对分组进行重排序。这在本领域的技术人员的技能内。

[0055] 参考示出了分组分段的高级视图的图9,重要的是注意,遍历直接互连环形或高基数的分组在路由期间被分段成更小大小的分组(微片)。分组分段是跨越环形网格所需要的,以便避免“线端阻塞”(使用公共资源(链路和缓冲器)的分组网络中的常见情况)。微片的大小由业务类型确定以便使开销税最小化,并且实际值范围在64个字节与512个字节之间。重要的是注意,本发明能够处理以太网分组、Infiniband分组、PCIe流或NVMe事务等等。跨越环形结构的微片在透明地承载业务。外部接口I/O端口13特定于业务的类型并且终止用户业务,从而经由“微片”跨越环形将数据透明地传送到目的地PCIe端口15。重要的是注意,本发明能够处理跨越环形(经由以太网微片)的以太网分组。本文的关键优点是微片的可变大小,因为以太网分组的微片具有分组划界机制。因此,微片大小从64个字节(最小以太网分组大小)到512个字节(净荷和报头)是可变的。如果分组大于512个字节,则系统将会将原始分组分段成512个字节的微片:头微片、体微片和尾微片。

[0056] 图12提供了呈现分组分段的方法的流程图。一旦从主机(即,PCIe接口)接收到新分组(步骤402),HNI处理器就提取报头以便让处理器分析该分组的目的地(步骤404)和路由路径。在这个阶段,分组进入分段循环。字节计数器被初始化(步骤412)并且分组结束-EOP-分析器(步骤405)检测EOP,并且如果计数器小于或者等于476个字节(即,净荷小于或者等于476个字节),则分组被发送到微片生成器(步骤407)、报头生成器(步骤408)以及优先级判定块(步骤409)。微片此后被插入到指派的出口队列中(步骤410)以用于在环形网格上被发送到下一跳,并且我们注意看是否已接收到新分组(402)。字节计数器在每个单独的微片之后递增(步骤411),分组分段被发送到微片生成器(步骤407)并且过程遵循上面所描

述的步骤,直到分组中的最后微片到达并被发送为止。作为关键观察结果,环形链路上的微片优选地是后面有8个“空闲”字节的512个字节(476个字节的净荷和36个字节的报头)。如果分组比476个字节短,则微片(通常是尾微片)将具有不到512个字节,并且将使用“空闲”字节完成分组划界。图13提供了通过分组分段实现微片生成的伪代码。

[0057] 在目的地处,通过剥去环形报头并且对于微片在原始分组中的偏移使用“字节计数”,微片被组装成分组。本领域的技术人员将知道为目的地节点实现每源分组队列,创建分组分段的有序列表,并且一旦被按照原始顺序完全组装就释放分组。图14示出了根据本发明的呈现路由计算的流程图。本发明还在图15处提供伪代码用于计算在源节点的每个输出端口上最大不相交的(无公共链路和节点)从源节点到任何目的地节点的所有路径(源路由)。一检测到更新(即,每当已经添加/检测到节点或者存在与其相关联的临界问题/故障时)拓扑更新功能412(其发现活动服务器/节点及其活动端口)就将启动路由/路径计算(414)。如后面有步骤420和421(搜索拓扑列表中的所有目的地节点)的步骤419(在拓扑列表中找到第一目的地节点)中所描述的,根据本发明将在来自拓扑的当前节点与每个目的地节点之间使用Dijkstra算法来计算最短路径(步骤415)。在步骤422中,受步骤417影响的所有路径成本被重置为初始值。所计算的从源到目的地的最短路径将被存储在源路由数据库中并且它将通过将属于所存储的路径的所有链路上的成本增加到无限大而被从另外的最短路径计算中移除(以便提供不相交路径)(417)-这将具有从拓扑数据库中移除节点和链路以迫使不再用于不相交路径选择目的的实际效果。该过程继续直到所有输出端口将被使用并且源节点与目的地节点之间的所有可能的路径被存储在数据库中为止。结果然后作为每个节点的路由向量被概括在输出路由表(418)中,其中路由将首先从最短路径开始。最短路径计算它本身能够使用Dijkstra算法(路由协议实施中使用的众所周知的方法)来实现。本文特别新颖的是最短路径在每个输出端口处被确定并且相应地被使用。

[0058] 图16呈现路由输出表的格式。每个路由具有表示特定路径的跳数以及要在每跳上选择的输出端口的列表/向量的附连数目。特别地,图16呈现从源节点号1000到目的地节点0(节点1000和节点0的使用仅仅用于示例目的)的具有8个端口($x+$ 、 $x-$ 、 $y+$ 、 $y-$ 、 $z+$ 、 $z-$ 、 $w+$ 和 $w-$)的4-D环形($8 \times 8 \times 8 \times 8$)中的路由的输出表的示例快照。应当注意的是,8个路由的第一组中的8个路由(即, $x+$ 、 $y+$ 、 $z+$ 、 $w+$ 、 $x-$ 、 $y-$ 、 $z-$ 和 $w-$)的组中的第一输出选择例如对应于源节点的每个输出端口。8个路径完全不相交并且具有不同的长度:5跳、7跳或9跳。最重要的特征是跨越启用业务分发和可预测性能的环形的路径分发。图16中的8个路径的第二组表示路由路径的第二替代方案。该方法可以计算许多替换的路由,但是距离(跳数)将变得越来越大。由于实际的原因,该方法可以根据需要通常仅存储8个不同路径的第一选项。

[0059] 图17示出了根据本发明的表示分組交换中使用的步骤的流程图。

[0060] 随着新分組被推送到输出队列中,分組传送状态机启动跨越环形的传送。在这方面,讨论的(in question)输出端口“k”(端口1至端口8,即上面所讨论的输出端口: $x+$ 、 $y+$ 、 $z+$ 、 $w+$ 、 $x-$ 、 $y-$ 、 $z-$ 和 $w-$)被用来遵循虫孔模型传送微片(如图18中所示):头微片和后续微片递增(体微片),将被沿着从端口k起的同一路径传送,直到如图17中所示到达分組结束(尾微片)为止。一旦到达分組结束,计数器k就递增以指示下一个输出端口并且具有下一个分組标识符的下一个分組(被指定为下一个有序分組)使用这个虫孔交换模型在逐微片基础上在环形上被传送。一旦检测到分組结束,端口号就被再次递增(即, $k=k+1$)并且下一个分組

将被传送等等。这个实施例将在循环模型中在节点的所有输出端口14上分发源自一个节点的分组,以便使长的数据流与特定目的地去相关并且在环形网格中避免“热点”发展。本领域的技术人员将理解,能够根据需要修改这个分发方法以必要时针对不同类型的业务分发提高性能。例如,你能够用加权循环方法或任何其它用户定义的方法代替循环法。本发明的方法已被发现为比Valiant无关分发更好地执行,并且在结果方面是非常一致的且可预测的。这从网络的性能观点看是非常显著的。图19显示了实现如在图17处的流程图中所描述和示出的微片传送的伪代码。

[0061] 图20示出了根据本发明的呈现拓扑发现的方法的流程图。如上面所指出的,拓扑发现组件(TD)对于基本上发现节点(和节点端口)存在(或不存在)于环形网格中是重要的,所述节点/端口然后可以被用在以上路由分发中。在这方面,TD允许如PCT专利申请号PCT/CA2014/000652中特别公开的本发明的新颖网络以简化且高效方式缩放。TD基于“Hello”协议和数据库交换协议。特别地,在每个节点和每个端口上发送并接收“Hello”消息。一旦消息作为“Hello”被接收到,并且“Hello”分组中的节点ID被管理框架验证,则本地节点信息就被更新以指示到邻居的新链路的可用性。节点信息的收集在每个节点上被维护在邻居数据库中并且通过数据库交换协议与邻居同步。每个节点将它对邻居数据库的改变发送到邻居节点,从而逐步传播整个拓扑。在特定超时之后,如果对邻居数据库的改变发生了,则为路径计算生成功能调用。对于“Hello”消息交换,邻居到邻居永远以定义的时间间隔继续,以便检测拓扑中的改变。

[0062] 图21示出了根据本发明的呈现针对拓扑发现的DB交换更新的流程图。

[0063] 这基本上涉及负责在每个节点中维护交换机的完整映像的TD组件,包括所有经互连的服务器以及输入/输出链路和连接性状态。

[0064] 图22提供了呈现本发明的系统如何发现节点/链路故障检测的流程图以及重新路由以避免已出故障的节点的过程。随着链路故障被检测到602,直通块被编程使得去往已出故障的链路的业务被重新路由到CPU 603。业务源被通知以避免已出故障的链路发送业务604。这触发拓扑更新事件605。

[0065] 随着链路恢复被检测到606,直通块被编程使得由于已出故障的链路而去往CPU的业务被重新路由回到同一链路607。

[0066] 业务源被通知以链路故障之前的同一方式发送业务608。这触发拓扑更新事件609。在链路切断的情况下将发生类似的过程。图23提供了针对链路/节点故障检测和恢复的伪代码。

[0067] 本发明的一个实施例是用于新节点插入的方法。为了插入节点,互连现有节点的布线被中断(链路切断)。随着环形布线中断被检测到602,业务重新路由API被调用604。链路成本被设置为无限大并且拓扑被更新(为所有交换机节点计算最短(较低成本)路径)。这定义将被用来控制分组流以到达任何目的地的路由。新节点插入触发邻居发现。管理软件检查新节点ID并且与所提供的数据库进行比较。如果节点是有效的,则软件启动拓扑更新和路由计算的过程。管理数据库将随着新节点被更新并且节点状态将被设置为活动的。

[0068] 尽管已经描述了本发明的特定实施例,然而对于本领域的技术人员而言将显而易见的是,可以在以下权利要求的范围内做出对实施例的变化和修改。

[0069] 附图文字说明

[0070]	图1	
[0071]	PRIOR ART	现有技术
[0072]	图2	
[0073]	PRIOR ART	现有技术
[0074]	图3	
[0075]	PRIOR ART	现有技术
[0076]	Compute Chip	计算芯片
[0077]	2processors	2个处理器
[0078]	(compare this with a 1988Cray YMP/8at 2.7GF/s) 1988Cray YMP/8进行比较)	(在2.7GF/s下将此与
[0079]	Compute Card	计算卡
[0080]	I/O Card	I/O卡
[0081]	FRU(field replaceable unit)	FRU(现场可替换单元)
[0082]	2nodes (4CPUs)	2个节点 (4个CPU)
[0083]	Node Card	节点卡
[0084]	16compute cards	16个计算卡
[0085]	0-2I/O cards	0-2个I/O卡
[0086]	32nodes	32个节点
[0087]	(64CPUs)	64个CPU
[0088]	Cabinet	机柜
[0089]	2midplanes	2个中间平面
[0090]	1024nodes	1024个节点
[0091]	2,048CPUs	2,048个CPU
[0092]	System	系统
[0093]	64cabinets	64个机柜
[0094]	65,536nodes	65,536个节点
[0095]	131,072CPUs	131,072个CPU
[0096]	2.500sq ft.	2.500平方英尺
[0097]	MTBF 6.16Days	MTBF 6.16天
[0098]	图4	
[0099]	Number of hops	跳数
[0100]	Number of nodes	节点数
[0101]	Destination nodes starting from source node 1.6 的地节点	从源节点1.6开始的目
[0102]	图7	
[0103]	8port NIC Card	8端口NIC卡
[0104]	图9	
[0105]	MESSAGE/FLOW	消息/流
[0106]	PACKET	分组

[0107]	Seq nb.	顺序号
[0108]	data	数据
[0109]	Adr	地址
[0110]	FLIT	微片
[0111]	flit data	微片数据
[0112]	图10	
[0113]	36Nodes (6x 6)MDP fromS=2.4to D=5.2	从S=2.4到D=5.2的
	36个节点(6x 6)MDP	
[0114]	ROUTE 5 (x+,x+,x+,y-,y-)	路由5(x+,x+,x+,y-,y-)
[0115]	ROUTE 5 (x-,x-,y-,y-,x-)	路由5(x-,x-,y-,y-,x-)
[0116]	ROUTE 5 (y-,y-,x+,x+,x+)	路由5(y-,y-,x+,x+,x+)
[0117]	ROUTE 7 (y+,y+,y+,x+,x+,x+,y+)	路由7(y+,y+,y+,x+,x+,x+,y+)
[0118]	图11	
[0119]	TOPOLOGY DISCOVERY	拓扑发现
[0120]	PATH VECTOR COMPUTATION	路径向量计算
[0121]	HEADER NIBLE LOOKUP	报头半字节查找
[0122]	CONTROLFILT SENT TO SOURCE	控制发送到源的微片
[0123]	图12	
[0124]	402 接收到新分组	
[0125]	404 分析目的地	
[0126]	412 字节计数=0	
[0127]	407 创建微片	
[0128]	408 附接环形报头	
[0129]	409 分析优先级	
[0130]	410 插入在出口队列中	
[0131]	411 字节计数=字节计数+476	
[0132]	405 分组结束?	
[0133]	YES 是	
[0134]	NO 否	
[0135]	图14	
[0136]	412 拓扑更新	
[0137]	414 启动路由路径计算	
[0138]	419 目的地=initial_dest	
[0139]	415 计算当前节点与目的地之间的最短Dijkstra路径	
[0140]	422 重置网络拓扑链路	
[0141]	416 找到路径?	
[0142]	YES 是	
[0143]	NO 否	
[0144]	417 从网络拓扑中移除最短路径链路	

[0145]	418	更新路由表
[0146]	420	最后目的地节点?
[0147]	YES	是
[0148]	NO	否
[0149]	End	结束
[0150]	图16	
[0151]	4096 (8x8x8x8) MDP from S=1000to D=0	从S=1000到D=0的4096 (8x8x8x8) MDP
[0152]	ROUTE	路由
[0153]	图17	
[0154]	421	分组目的地“n”
[0155]	422	分组在队列中?
[0156]	yes	是
[0157]	no	否
[0158]	424	递增顺序号
[0159]	426	在端口k上路由微片i
[0160]	End of	packet分组结束?
[0161]	yes	是
[0162]	no	否
[0163]	图18	
[0164]	Wormhole Switching	虫孔交换
[0165]	Packet	分组
[0166]	FLIT	微片
[0167]	图20	
[0168]	Port UP event	端口UP事件
[0169]	Send Hello@time interval	以时间间隔发送Hello
[0170]	Hello received	接收到Hello
[0171]	YES	是
[0172]	NO	否
[0173]	New Nbr	新Nbr
[0174]	Nbr Valid	Nbr有效
[0175]	Process Link Master	处理链路主控
[0176]	Start DB Exchange Nbr	开始DB交换Nbr
[0177]	Port DOWN or retries exceeded	端口断开或者重试超过了
[0178]	Declare Link down	声明链路断开
[0179]	Update local NodeInfo	更新本地NodeInfo
[0180]	Stop DB_exchange_Nbr	停止DB_exchange_Nbr
[0181]	DB Exchange Nbr	DB交换Nbr
[0182]	DB update in progress	DB更新在进行中

[0183]	Link Master?	链路主控?
[0184]	YES	是
[0185]	NO	否
[0186]	Send DB summary For Entire DB	发送整个DB的概要
[0187]	Received DB summary updae?	接收到DB概要更新?
[0188]	YES	是
[0189]	NO	否
[0190]	Calculate DB delta	计算DB增量
[0191]	Send summary update-For old DB elements	发送概要更新-针对旧DB元素
[0192]	Send DB Elements List newer DB elements	发送较新DB元素的DB元素列表
[0193]	DB elements List received?	接收到DB元素列表?
[0194]	YES	是
[0195]	NO	否
[0196]	DB Update not in progress	DB更新不在进行中
[0197]	Wait to receive from Nbr DB Element list	等待从Nbr DB元素列表接收
[0198]	For Each DB Element inlist	针对列表中的每个DB元素
[0199]	DB Element List empty?	DB元素列表是空的?
[0200]	YES	是
[0201]	NO	否
[0202]	DB element Old?	DB元素是旧的?
[0203]	YES	是
[0204]	NO	否
[0205]	Return own version of DB Element	返回DB元素的自己版本
[0206]	DB element New?	DB元素是新的吗?
[0207]	YES	是
[0208]	NO	否
[0209]	Issue DB update event	发布DB更新事件
[0210]	图21	
[0211]	DB Exchange Process	DB交换过程
[0212]	Wait for DB update event	等待DB更新事件
[0213]	Any DB_Exchange_NBR updatein progress?	任何DB_Exchange_NBR更新在进行中?
[0214]	YES	是
[0215]	NO	否
[0216]	Start RouteCalc timer	启动RouteCalc定时器
[0217]	For Each DB_Exchange_Nbr	针对每个DB_Exchange_Nbr
[0218]	DB_Exchange_Nbr empty	DB_Exchange_Nbr为空
[0219]	YES	是
[0220]	NO	否

- [0221] Is Nbr src for update? Nbr src用于更新吗?
- [0222] Send new DB elements to Nbr 向Nbr发送新DB元素
- [0223] Updates Ack'ed 更新被确认
- [0224] YES 是
- [0225] NO 否
- [0226] Nbr still UP? Nbr仍然UP?
- [0227] YES 是
- [0228] NO 否
- [0229] 图22
- [0230] 602 链路断开
- [0231] 603 将直通块编程以向软件重新路由业务
- [0232] 604 通知业务发信方节点关于断开下行链路
- [0233] 605 触发拓扑更新
- [0234] 606 链路恢复了?
- [0235] YES 是
- [0236] NO 否
- [0237] 607 将直通块编程为向硬件重新路由业务
- [0238] 608 通知业务发信方节点关于链路恢复
- [0239] 609 触发拓扑更新

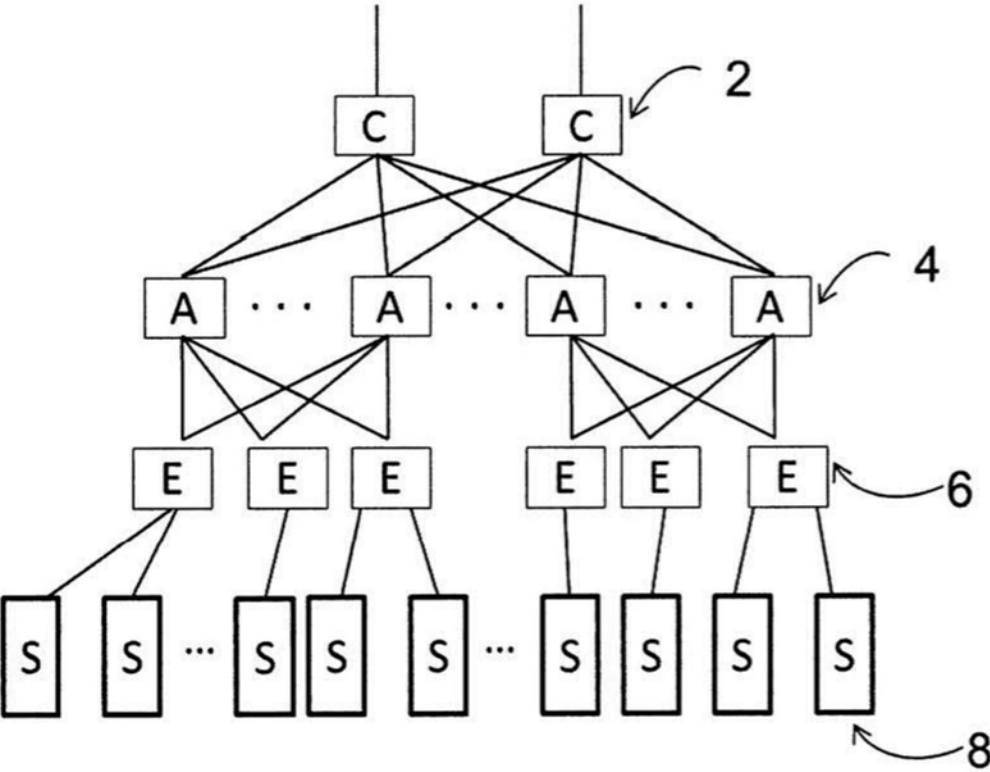


图1

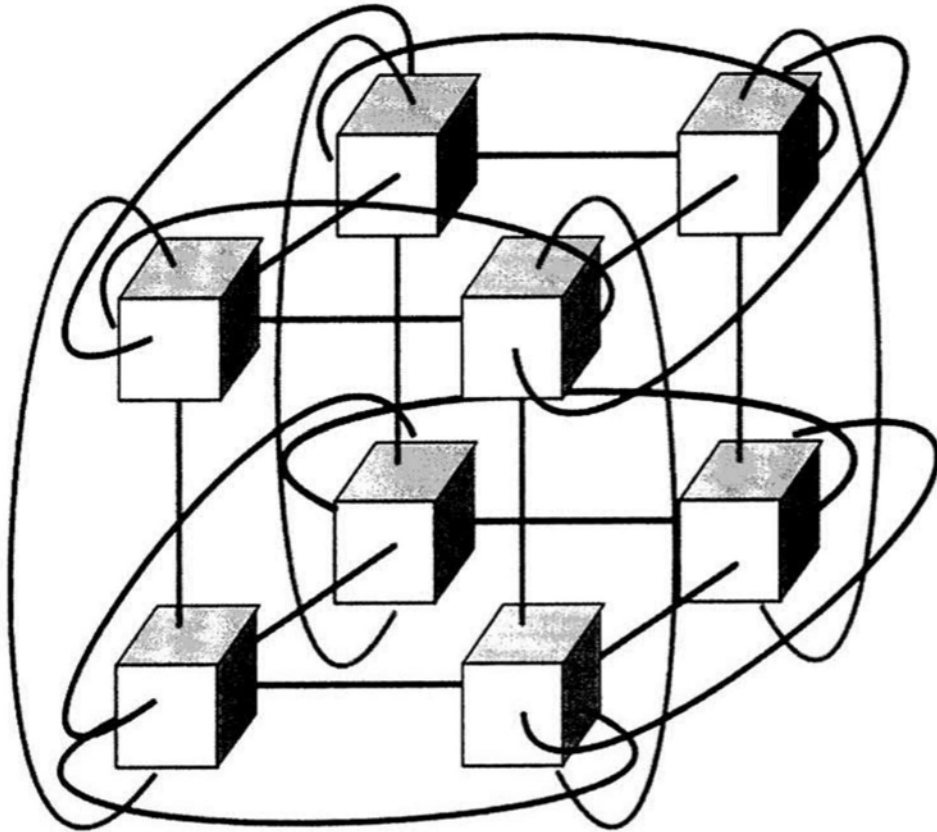


图2

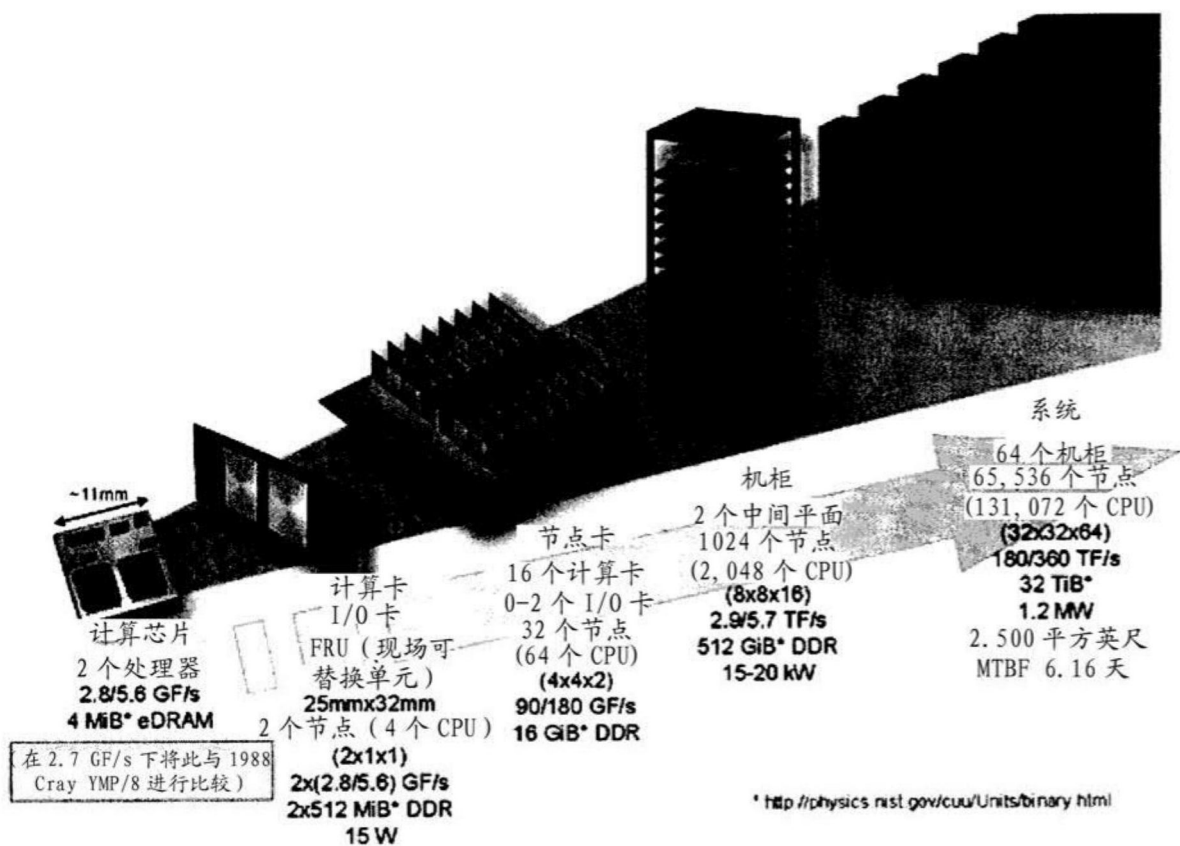
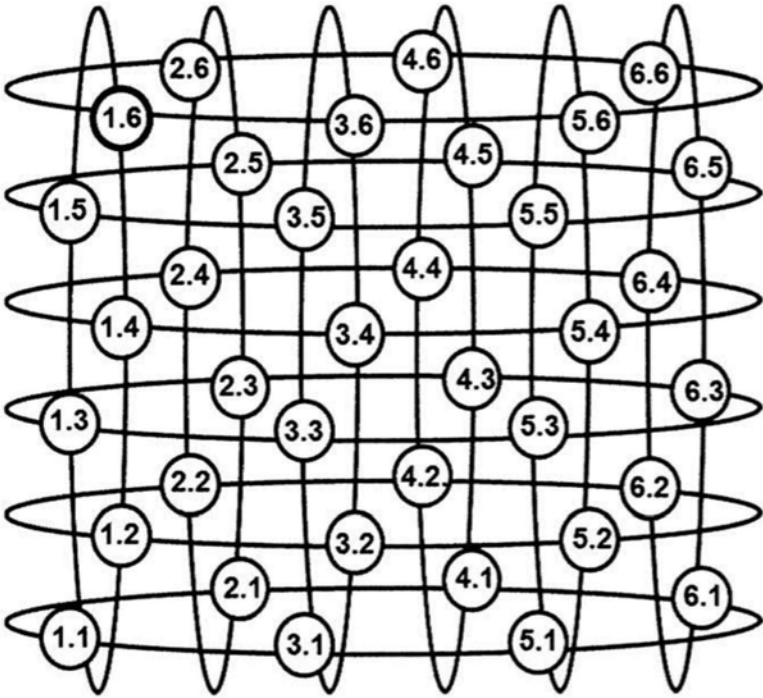


图3



1	2	3	4	5	6	跳数
4	8	10	8	4	1	节点数
2.6;1.5;1.4 3.6	2.4;2.5;1.2; 1.3;3.4;3.5; 4.6;5.6	2.3;2.2;1.1;3.3; 3.2;5.4;5.5;6.6; 4.4;4.5	2.1;6.5;6.4; 5.2;5.3;4.2; 4.3;3.1	5.1;4.1;.6;2 6.3	6.1	从源节点 1. 6 开始的目的地节点

图4

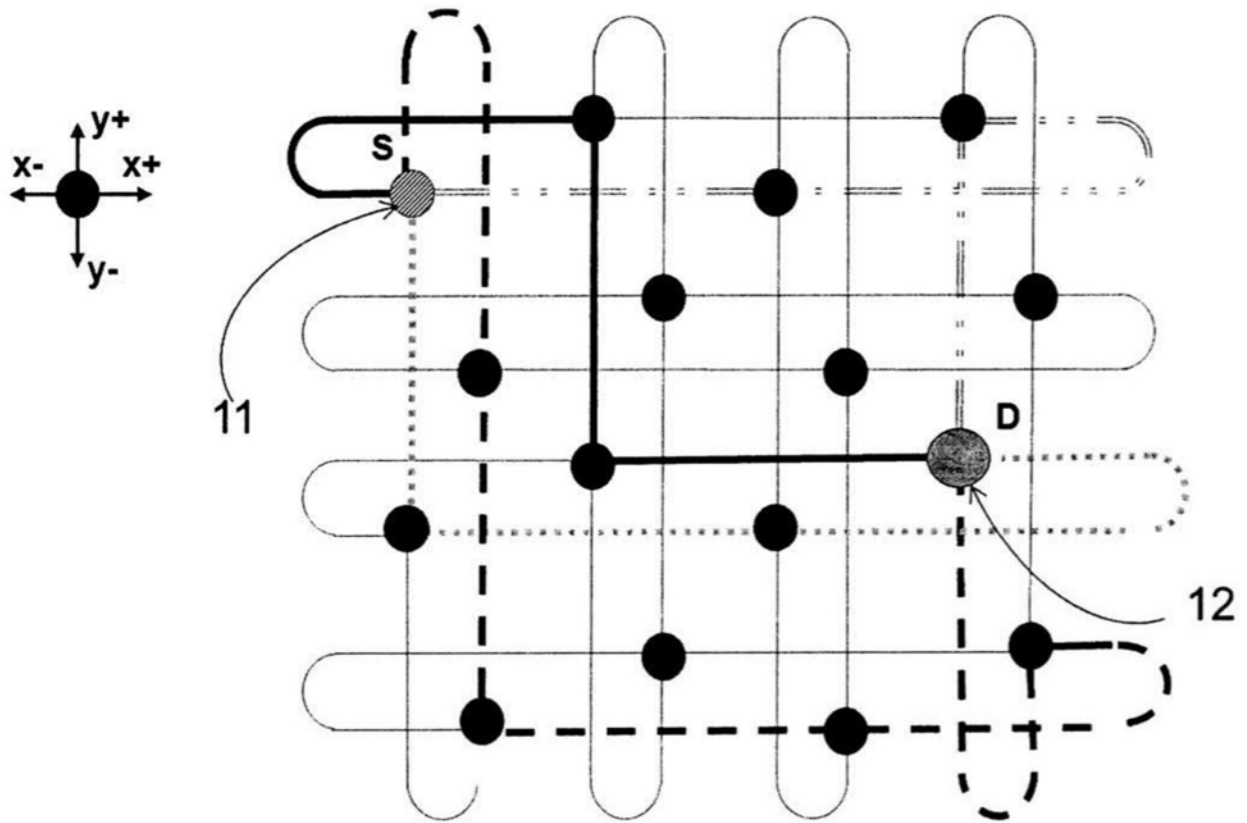


图5

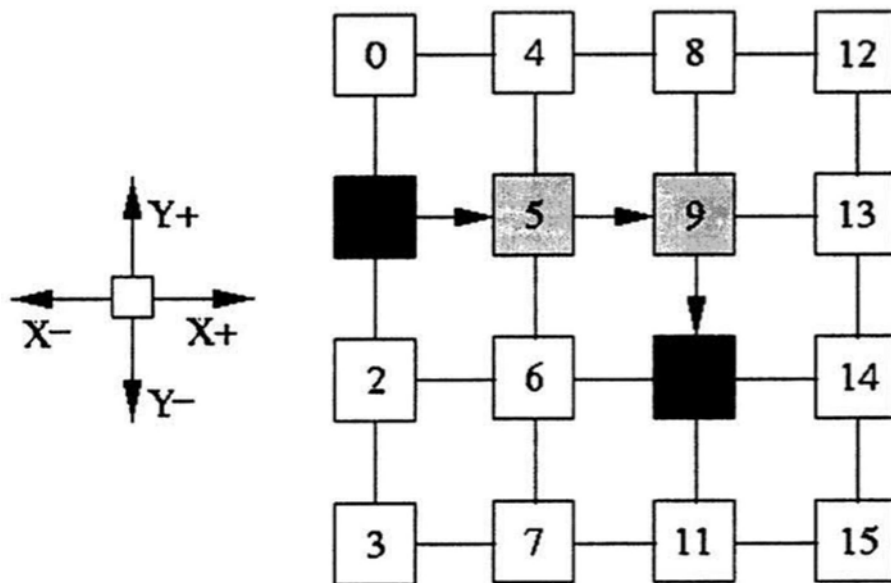


图6

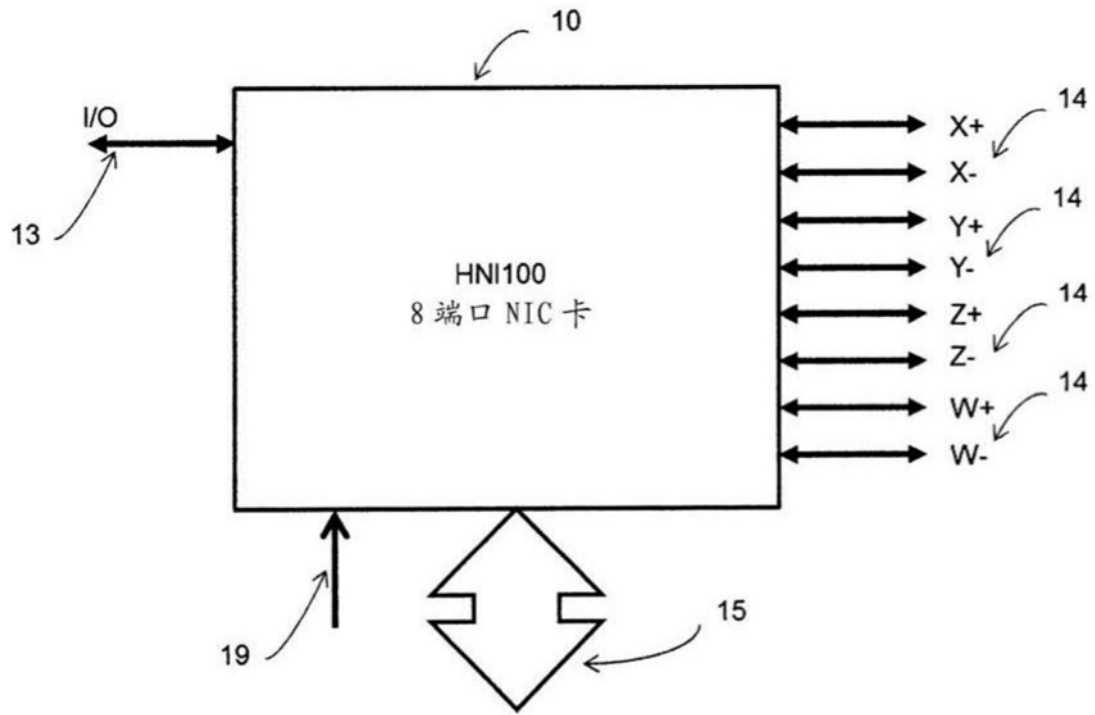


图7

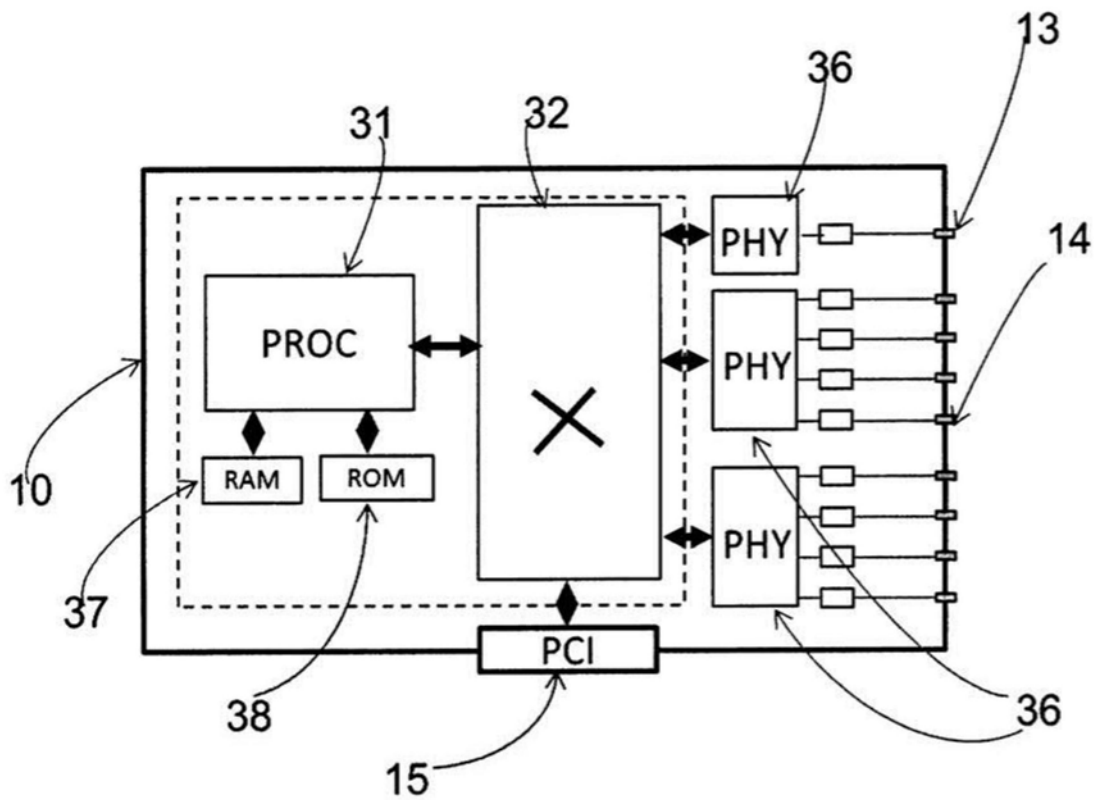


图8

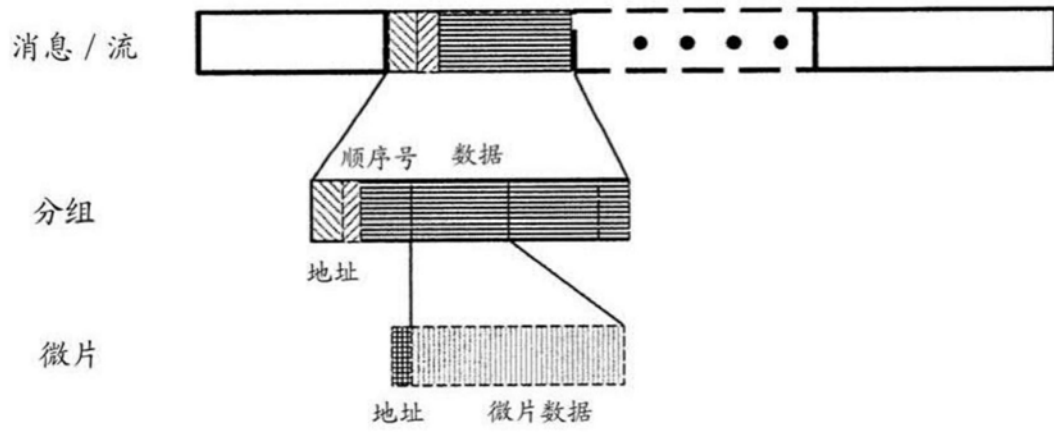
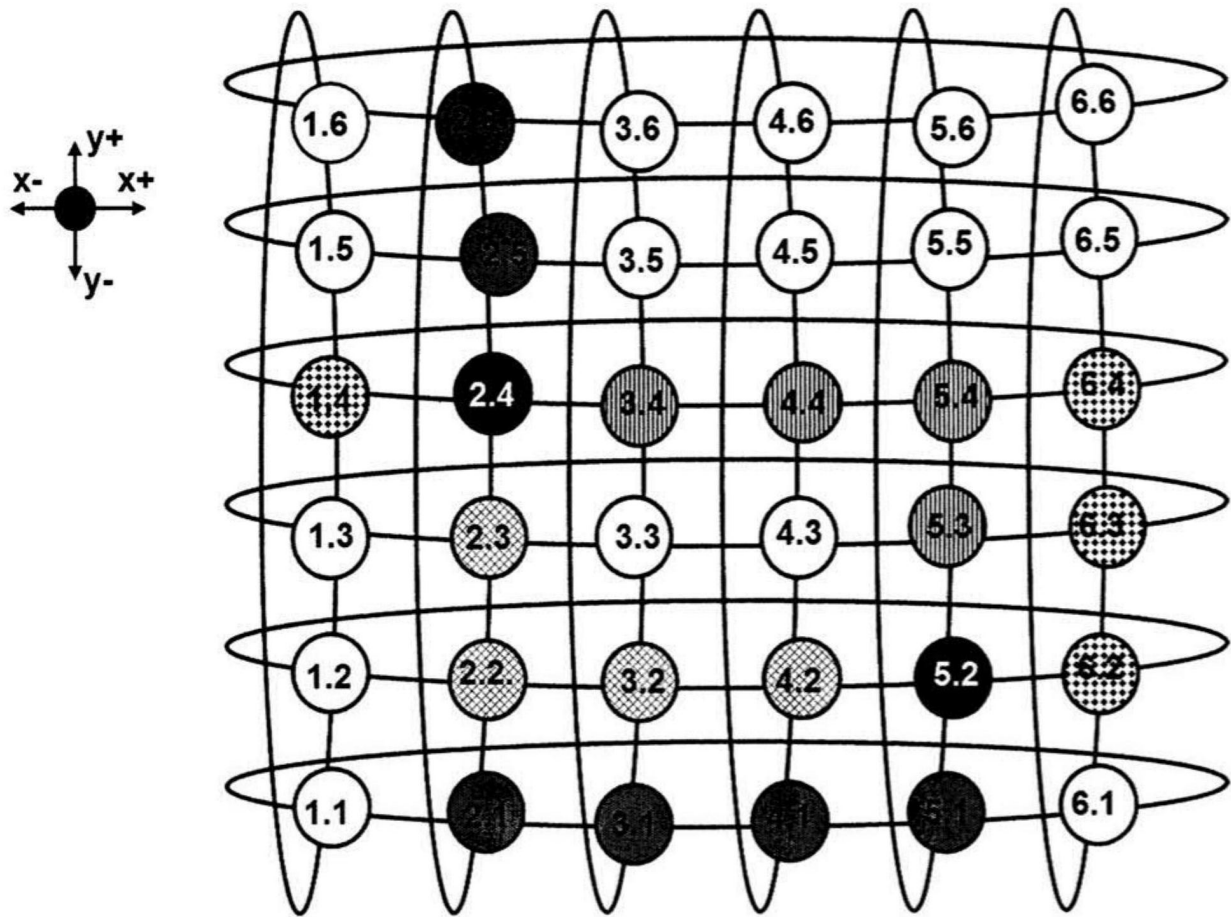


图9



从 S=2.4 到 D=5.2 的 36 个节点 (6 x 6) MDP

路由 5 ($x+, x+, x+, y-, y-$)

路由 5 ($x-, x-, y-, y-, x-$)

路由 5 ($y-, y-, x+, x+, x+$)

路由 7 ($y+, y+, y+, x+, x+, x+, y+$)

图10

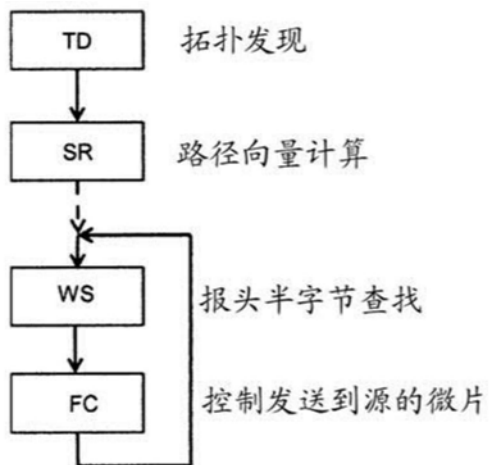


图11

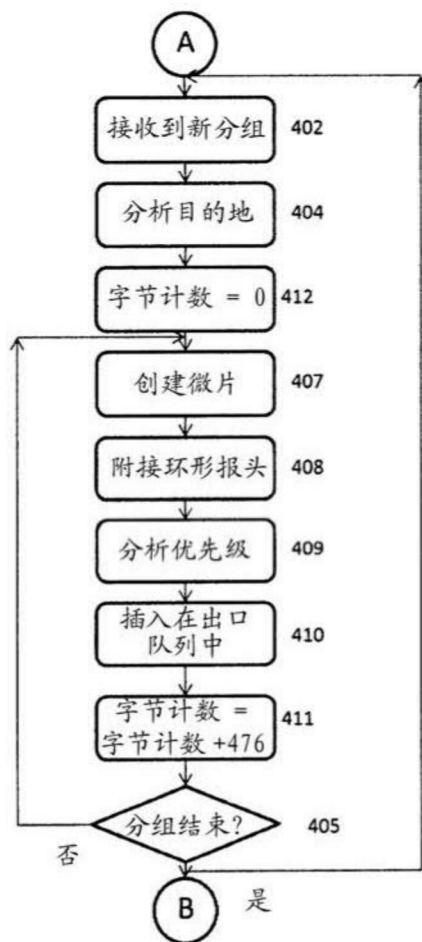


图12

```
1 function create_flit
2   packet = receive_new_packet
3   dest = analyze_destination(packet)
4   priority = analyze_priority(packet)
5   packet_pointer = 0
6   flit_length = 476
7   do
8     flit =
          create_flit(packet,min(flit_length,(end_of_packet -
          packet_pointer))
9         flit_packet = attach_torus_header(dest,flit,priority)
10        insert flit_packet in send_queue
11    packet_pointer += flit_length
12  while(not end_of_packet)
13 end function
```

图13

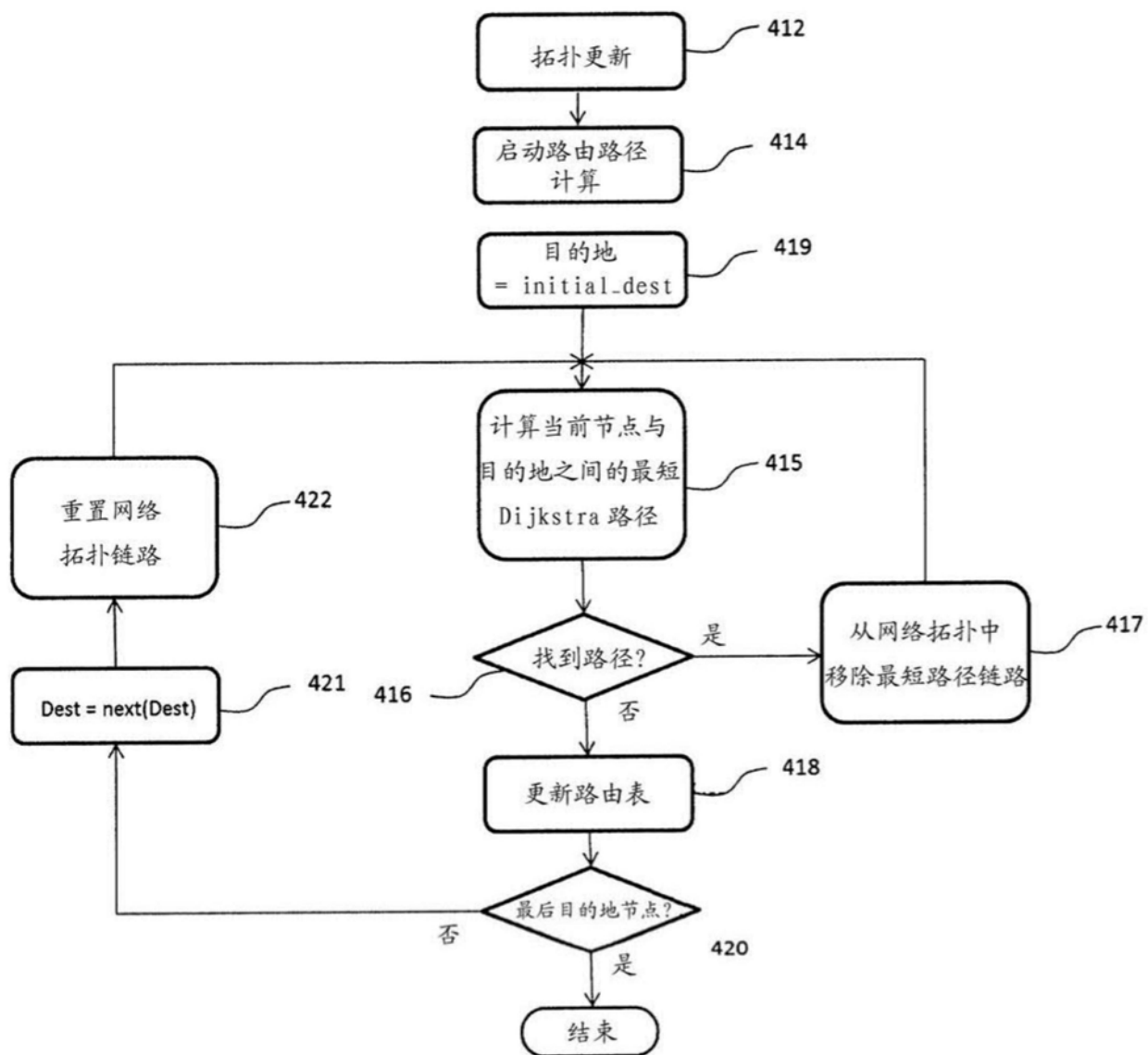


图14

```

1 function SourceRouting(Graph, source, destination):
2
3 while true:
4   path ← CalculateDijkstra(Graph, source, destination):
5   if path != NULL
6     for each link l of path:
7       cost(l) ← infinity
8     end for
9   else if
10    exit while:
11  end if
12  update_routing_tables
13 end while
14 end function

1 function CalculateAllRoutes(Graph, source):
2
3 initialize_routing_tables
4 for each Dest of topology_list
5   reset_topology_costs(Graph)
6   SourceRouting(Graph,source, Dest)
7 end for
8 end function

```

图15

4096 (8x8x8x8) MDP from S= 1000 to D=0

```

ROUTE 7 (x+,w+,y-,z-,w+,w+,y+);
ROUTE 7 (y+,w+,y-,z-,w+,w+,x+);
ROUTE 7 (z+,z-,y-,z+,z+,z+,z+);
ROUTE 5 (w+,z-,w+,w+,y-);
ROUTE 9 (x-,z-,y-,y-,y-,w+,w+,w+,w-);
ROUTE 5 (y-,w+,w+,w+,z-);
ROUTE 5 (z-,w+,y-,w+,w+);
ROUTE 9 (w-,y-,z-,y-,w+,z-,w+,w+,x-);

ROUTE 7 (x+,z-,w+,w+,w+,y-,y+);
ROUTE 7 (y+,z-,w+,w+,w+,y-,x+);
ROUTE 7 (z+,z+,z+,z+,z-,z+,y-);
ROUTE 5 (w+,w+,z-,y-,w+);
ROUTE 7 (x-,w+,w+,y-,w+,y-,z-);
ROUTE 9 (y-,z+,z+,z+,y-,z+,z-,z+,x-);
ROUTE 7 (z-,z+,z+,z+,z+,y-,z+);
ROUTE 9 (w-,w+,w+,w+,z-,z-,z-,y-,w-);

```

图16

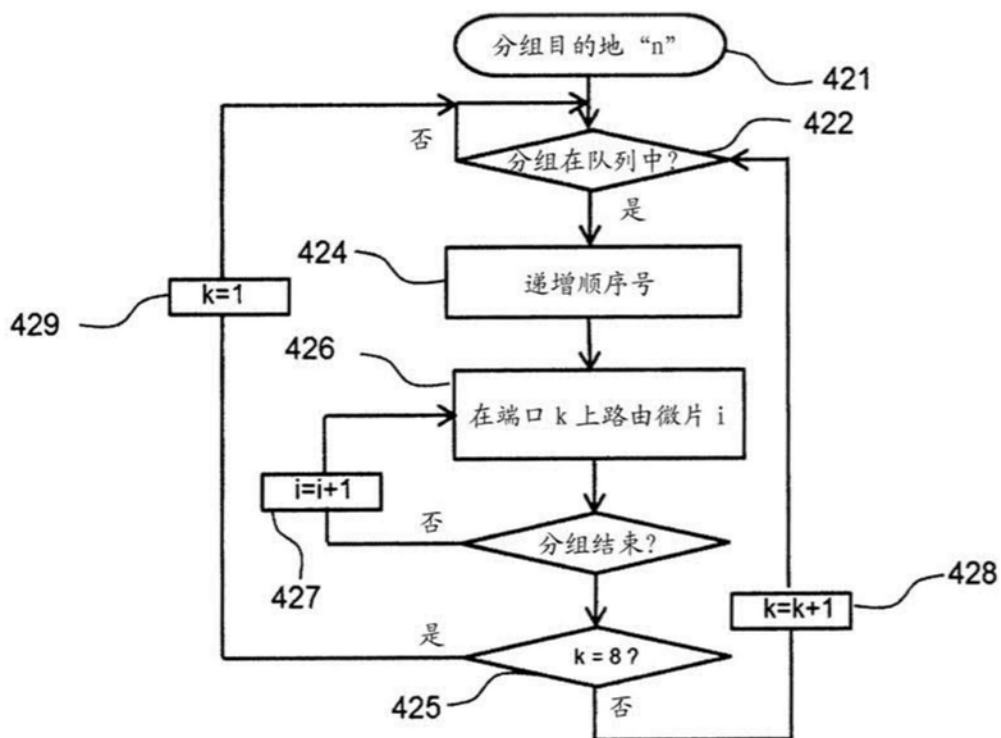


图17

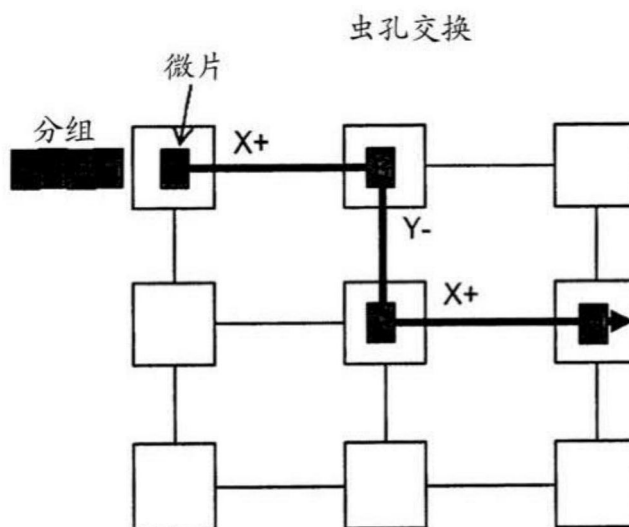


图18

```
1  function send_packet @ dest "n"
2    queue = packet_ready
3    for each dest
4      seq_number = increment_seq_number(dest)
5      for each packet
6        for each flit
7          flit = attach_header
8          port = extract_port_from_route(route)
9          send_packet(port)
10       end for
11       port=increment port
12       if port eq 8 port =1 end if
13     end for
14  end function
```

图19

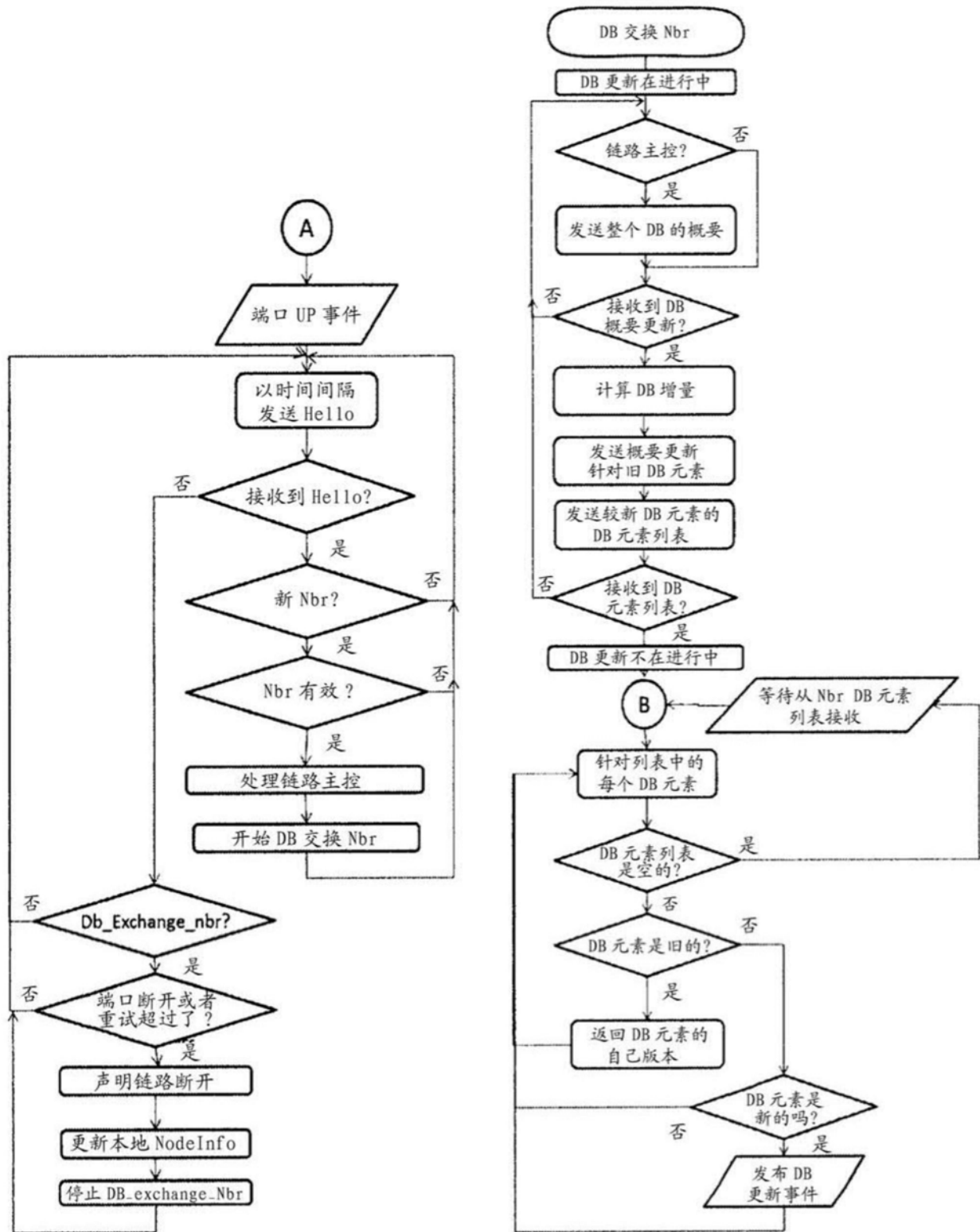


图20

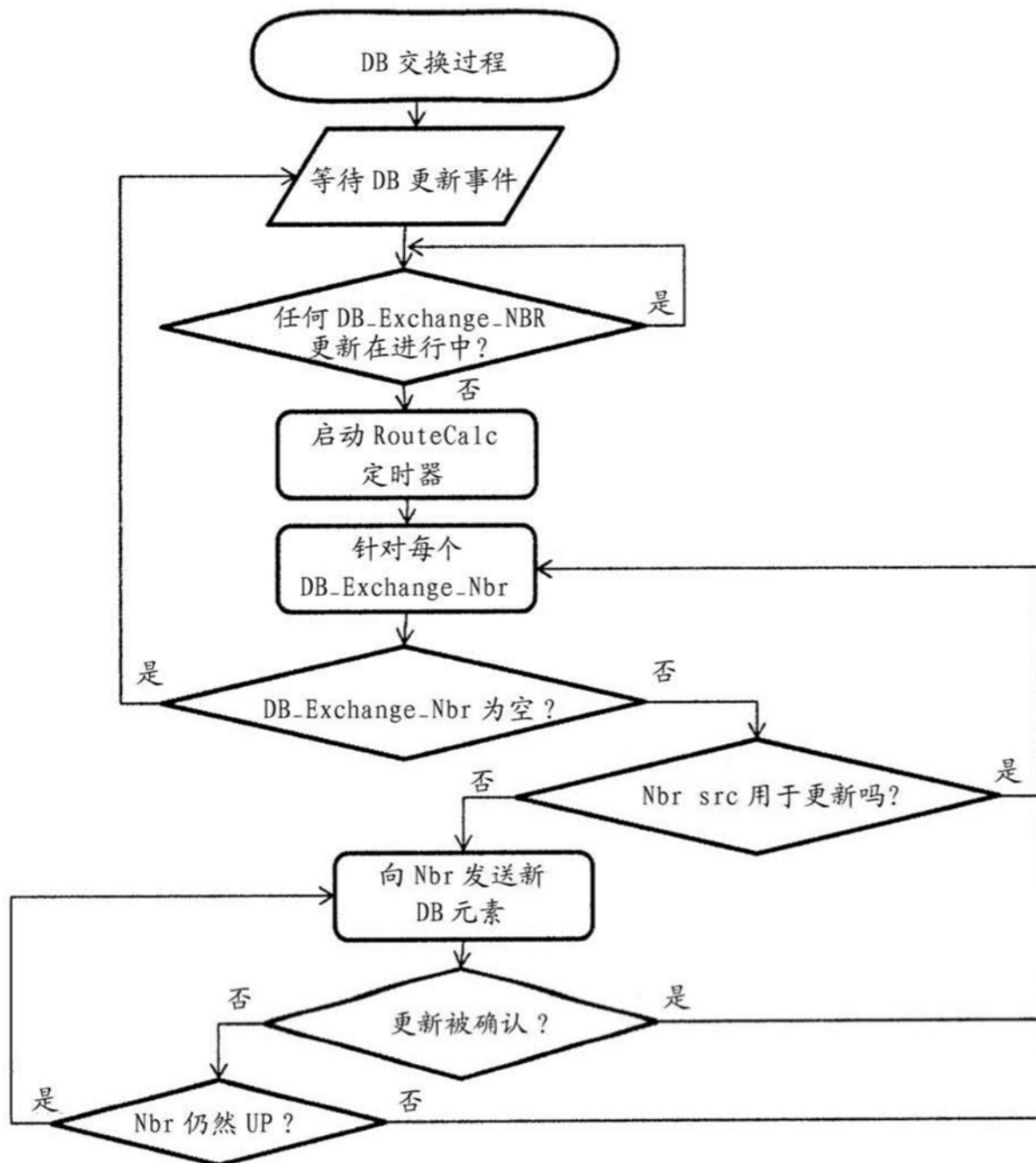


图21

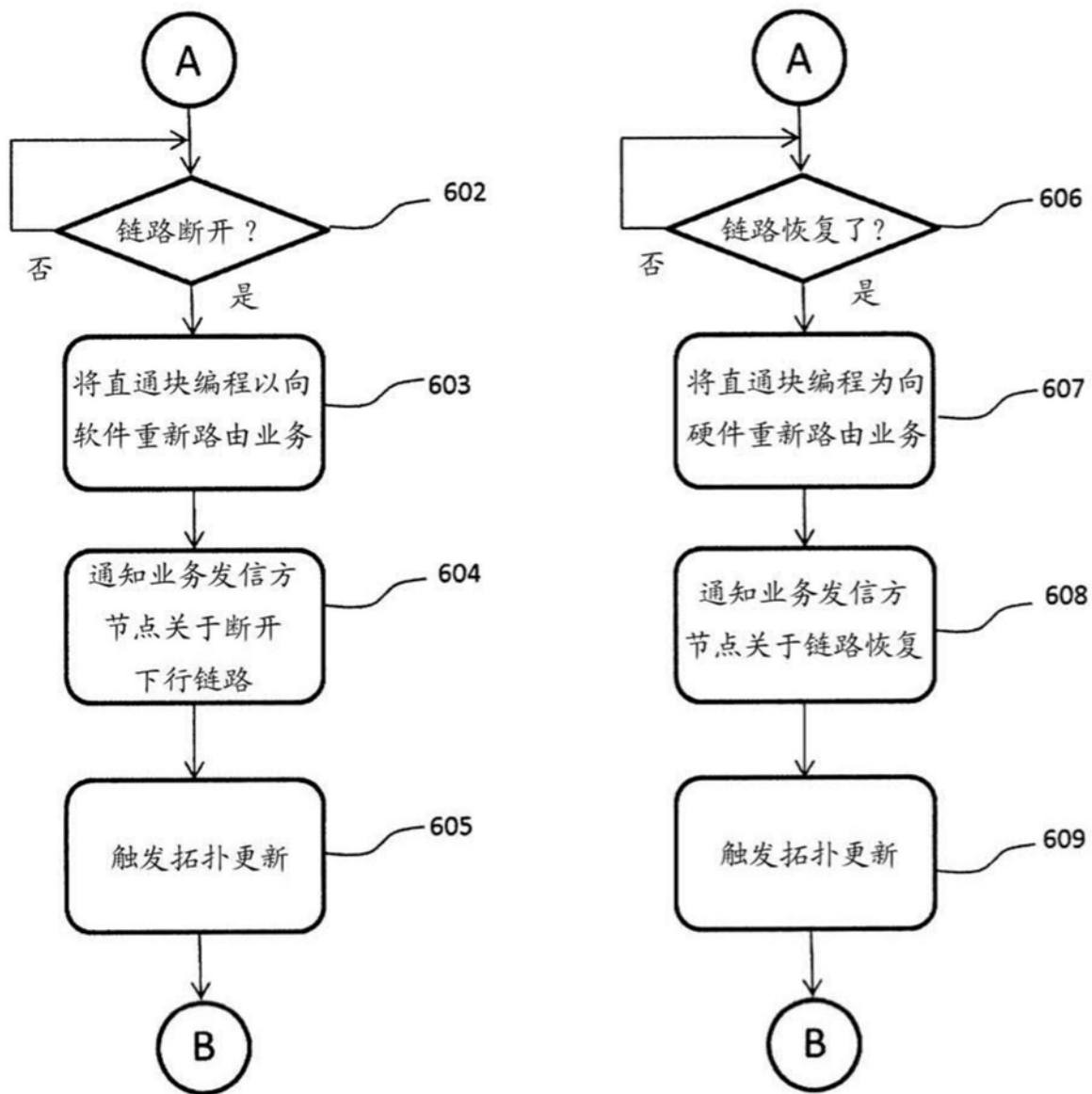


图22

```
1 function link_failure_protection
2   for each port
3     status = read_link_status
4     if status eq down
5       program_passthrough(traffic_to_cpu)
6       send_notification(originator, link_down)
7       topology_update
8     end if
9   end for
10 end function
```

```
1 function link_recovery
2   for each port
3     status = read_link_status
4     if status eq up
5       program_passthrough(traffic_to_hw)
6       send_notification(originator, link_up)
7       topology_update
8     end if
9   end for
10 end function
```

图23