### 三、DSR路由协议概述

本节概述DSR协议的操作。DSR的基本版本使用显式的“源路由”，其中每个发送的数据包都在其报头中包含一个数据包将通过的完整的、有序的节点列表。这种显式源路由的使用允许发送方选择和控制用于其自身数据包的路由，支持使用多条路由到任何目的地（例如，用于负载平衡），并允许简单地保证所使用的路由是无循环的；通过将此源路由包括在每个数据包的报头中，其他转发或无意听到任何数据包的节点也可以轻松地缓存此路由信息以供将来使用。3.1节描述了路由发现的基本操作，3.2节描述了基本的路由维护，第3.3和3.4节描述了DSR操作的这两个部分的附加特性。第3.5节描述了一个可选的、兼容的DSR扩展，即“流状态”，它允许在数据包中没有显式源路由头的情况下对大多数数据包进行路由，同

时仍然保留DSR操作的基本属性。

**3.1 基本DSR路由发现**

当某个源节点发出一个寻址到某个目标节点的新数据包时，源节点在数据包的报头中放置一个“源路由”，给出数据包到达目的地时要遵循的跳数序列。通常情况下，发送者将通过搜索其先前获得到的路由的“路由缓存”来获得合适的源路由；如果它的缓存中没有找到路由，它将启动路由发现协议来动态地找到该目标节点的新路由。在这种情况下，我们将源节点称为“启动器”，目标节点称为路由发现的“目标”。

例如，假设节点A试图发现到节点E的路由。本例中由节点A发起的路由发现将按以下方式进行：

^ "A" ^ "A,B" ^ "A,B,C" ^ "A,B,C,D"

| id=2 | id=2 | id=2 | id=2

+-----+ +-----+ +-----+ +-----+ +-----+

| A |---->| B |---->| C |---->| D |---->| E |

+-----+ +-----+ +-----+ +-----+ +-----+

| | | |

v v v v

为了启动路由发现，节点A将一个“路由请求”作为一个由目前在A无线传输范围内的(大约)所有节点接收，包括本例中的节点B的本地广播包发送。每个路由请求标识路由发现的发起者和目标，还包含一个由请求的发起者确定的唯一的请求标识(在本例中为2)。每个路由请求还包含一个记录，列出转发路由请求的特定副本的每个中间节点的地址。该路由记录由路由发现的发起者初始化为空列表。在本例中，路由记录最初只列出节点A。当另一个节点接收到此路由请求(例如本例中的节点B)时，如果它是路由发现的目标，它将向路由发现的发起者返回一个“路由应答”，提供来自路由请求的累积路由记录的副本；当发起者接收到此路由应答时，它会将此路由缓存在其路由缓存中，以用于向该目的地发送后续数据包。

否则，如果接收路由请求的节点最近看到来自此启动器的另一条路由请求消息具有相同的请求标识和目标地址，或者如果该节点自己的地址已经列在路由请求中的路由记录中，则该节点将丢弃该请求。（如果节点在其路由请求表中仍然有关于该请求的信息，则节点将考虑最近发现的请求）否则，该节点将自己的地址附加到路由请求中的路由记录中，并将其作为本地广播包(具有相同的请求标识)进行传输。在此示例中，节点B广播由节点C接收的路由请求；节点C和D也依次广播该请求，从而使节点E接收到该请求的副本。

在将路由应答返回给路由发现的发起者时，例如在本例中，节点E返回节点A，节点E通常会检查自己的路由缓存，以获得返回A的路由，如果找到，将使用它作为发送包含路由回复的数据包的源路由。否则，E应该对目标节点A执行自己的路由发现，但是为了避免路由发现的无限递归，它必须在包含它自己的路由请求A的数据包上携带这个路由回复。也还可以使用同样的机制在路由请求上携带其他小数据包，例如TCP SYN数据包。

相反，节点E可以简单地反转它试图发送的路由记录中的跳数序列，并将其用作承载路由回复本身的数据包上的源路由。对于像IEEE802.11这样需要双向帧交换并作为MAC协议一部分的MAC协议，在路由发现启动器开始使用路由之前必须以这种方式反转已发现的源路由以返回路由应答当测试已发现的路由并以确保它是双向的；此路由反转还避免了可能的第二路由发现的开销。

发起路由发现时，发送节点将原始数据包(触发发现)的副本保存在称为“发送缓冲区”的本地缓冲区中。发送缓冲区包含该节点无法传输的每个数据包的副本，因为它还没有到数据包的目的地的源路由。发送缓冲区中的每个数据包在逻辑上与将其放入发送缓冲区的时间相关联，并且在发送缓冲区中停留一段时间之后，将被丢弃。如果有必要防止发送缓冲区溢出，则还可以使用FIFO或其他替换策略在数据包过期之前将其排除。

当数据包保留在发送缓冲区中时，节点应该偶尔为数据包的目的地地址启动一个新的路由发现。但是，节点必须限制相同地址的这种新路由发现的启动速率，因为有可能目标节点目前无法到达。特别是由于无线传输范围有限以及网络中节点的移动，网络有时会被分割，这意味着目前没有可以通过这些节点转发数据包以到达目的地的节点序列。取决于网络中节点的移动模式和密度的网络分区可能是罕见的，也可能是常见的。

如果一个新的路由发现被在这样一个分区网络中的节点发送的每个数据包都启动了，那么大量的非生产性路由请求数据包将被传播到从此节点可到达的adhoc网络。为了减少这些路线发现的开销，节点应该使用指数后退算法来限制它对同一目标发起新路由发现的比率，并在为同一目标启动的每个连续发现之间加倍超时。如果节点试图超出限制的更频繁地向同一目标节点发送其他数据包，随后的数据包则应在发送缓冲区中缓冲，直到接收到一个到达此目的地的路由的路由应答。但是，在该目标的新路由发现之间的最小允许间隔达到之前，节点不能发起新的路由发现，这种对同一目标的最大路由发现速率的限制类似于在互联网节点上所要求的限制对任何单个目标IP地址发送ARP请求的速率的机制。

**3.2 基本DSR路由维护**

当使用源路由发起或转发数据包时，发送该数据包的每个节点负责确认数据可以通过该链路从该节点流到下一跳。例如，在下面所示的情况下，节点A使用应用中间节点B、C和D的源路由向节点E发起了一个数据包：

+-----+ +-----+ +-----+ +-----+ +-----+

| A |---->| B |---->| C |-->? | D | | E |

+-----+ +-----+ +-----+ +-----+ +-----+

在这种情况下，节点A负责从A到B的链路，节点B负责从B到C的链路，节点C负责从C到D的链路，节点D负责从D到E的链路。

ACK可以提供链路能够承载数据的证明，而在无线网络中，ACK通常是或作为使用中的MAC协议的现有标准部分（例如IEEE802.11定义的链路层确认帧），或通过一个“被动ACK”（例如，B通过偷听C在将数据包转发到D时发送数据包，从而确认在C处收到）而被免费提供。

如果内置确认机制不可用，则发送数据包的节点可以显式请求在路由沿线的下一个节点返回特定于DSR的software ACK；此software ACK通常将直接发送至发送节点，但是如果这两个节点之间的链接是单向的，这个software ACK则可以在不同的多跳路径上传输。

在从某个邻居节点收到ACK后，节点可以选择在一定时间内不需要来自该邻居节点的ACK，除非连接到该邻居节点的节点的网络接口总是接受响应于单播通信的ACK。

当一个software ACK已经被应用后，此确认请求应该被重传至最大次数。确认请求的重传可以作为单独的数据包发送，可以在原始数据包的重传上进行备份，或者在不包含software ACK的下一跳目的地的任何数据包上进行备份。

在确认请求达到重传的最大次数后，如果还未收到ACK，则发送方将指向下一跳目的地的链接视为当前“中断”。它应该从路由缓存中删除这个链路，并应该向每个从ACK被最后一次接受到以来，已经在该链路上发送数据包的节点返回一个“路由错误”。例如，在之前所示的情况下，如果C在一些请求之后没有收到来自D的ACK，则会返回一个“路由错误”给A和任何在上次C收到D的ACK后使用过由C到D的链路的任何其他节点。然后，节点A从其缓存中删除这个断开的链路；如果有必要，任何对于原始数据包的重传都可以由上层协议(如TCP)执行。为了向同一目的地E发送这样的重传或其他数据包，如果A的路由缓存中有另一条路由到E（例如，来自其早期路由发现的附加的路由答复，或从其他数据包中偷听到足够的路由信息），它可以立即使用新的路由发送该数据包。否则，它应该为这个目标执行一个新的路由发现（但必须符合3.1中的后退规则）。

**3.3附加路由发现的特点**

**3.3.1缓存听见的路由信息**

节点转发或以其他方式偷听到任何数据包时，应将来自该数据包的所有可用路由信息添加到其自己的路由缓存中。数据包中路由信息的有用性取决于物理介质的方向性特征，以及正在使用的MAC协议。具体而言，有三种不同的情况：

-网络中的链路通常只能单向(而不是双向)运行，并且在网络中使用的mac协议能够在单向链路上传输单播数据包。

-网络中的链接有时只能单向(而不是双向)运行，但是这种对任何链路的单向限制都不是永久的，几乎所有的链路都是物理上的双向的，网络中使用的mac协议能够在单向链路上传输单播数据包。

-在网络中使用的mac协议不能在单向链路上传播单播数据包；只能使用双向链路传输单播数据包。例如，ieee 802.11分布式协调功能(DCF)mac协议只能在双向链路上传输单播数据包，因为mac协议要求从接收方返回链路级别的确认数据包，并且还需要在传输节点与接收节点之间进行RTS与CTS数据包的双向交换。

例如，在上面的第一种情况下，数据包使用的源路由，路由请求中的累计路由记录或路由应答中返回的路由都应该被任何在“前向”方向上的节点缓存。任何节点都应该从接收到的任何这样的数据包中缓存这些信息，无论数据包是寻址到此节点，还是发送到广播(或多播)MAC地址，还是在节点的网络接口处于杂乱模式时被监听。但是，在这些数据包报头中标识的链接的“反向”方向不应该被缓存。

例如，在下面所示的情况下，节点A使用源路由与节点E通信：

+-----+ +-----+ +-----+ +-----+ +-----+

| A |---->| B |---->| C |---->| D |---->| E |

+-----+ +-----+ +-----+ +-----+ +-----+

当节点C沿着A到E的路由转发数据包时，它应该在缓存中增加“前向”方向链路的存在，这是它从这些数据包的头部、从自身到D和从D到E的方向链路获取到的。在这种情况下，节点C不应该缓存这些数据包报头中标识的链路的“反向”方向，从它自己回到B，从B到A，因为这些链路能是单向的。

在上面的第二种情况中，链路有时可以单向地运行，上面描述的链路应该在两个方向上缓存。此外，在这种情况下，如果节点X无意中听到(例如，通过杂乱模式)节点C发送的数据包使用从节点A到E的源路由，节点X也应该缓存所有这些链路，还包括它无意中听到了数据包的由C到X的链路。

在最后一种情况下，MAC协议要求单播操作具有物理的双向性，则源路由的链路应该在两个方向缓存，除非数据包还包含路由应答，在这种情况下，只应该缓存在此源路由中已经遍历的链路，但不应该缓存该路由中尚未遍历的链路。

**3.3.2 使用缓存路由响应路由请求**

一个接收到不以它为目标的路由请求的节点搜索其自己的路由缓存以寻找到请求的目标的路由。如果找到，该节点通常会将路由回复返回给启动器本身，而不是转发路由请求。在路由应答中，该节点设置路由记录以列出转发路由请求副本的跳数序列，并与从它自己的路由缓存获得的目标的源路由连接。

但是，在发送以这种方式使用其路由缓存中的信息生成的路由应答包之前，节点必须验证在此连接之后路由应答中返回的不包含路由记录中列出的重复节点的结果路由。例如，下图说明节点F已收到针对目标E的路由请求，而节点F在其路由缓存中已经有一条从自身到E的路由：

+-----+ +-----+ +-----+ +-----+

| A |---->| B |- >| D |---->| E |

+-----+ +-----+ \ / +-----+ +-----+

\ /

\ +-----+ /

>| C |-

+-----+

| ^

v |

路由请求 +-----+

路由 : A - B - C - F | F | 缓存: C - D - E

+-----+

将来自路由请求的累积路由记录与来自F的路由缓存的缓存路由连接起来，将包括一个从C传递到F并返回到C的重复节点。

在这种情况下，节点F可以尝试编辑路由，以及消除重复，得出从A到B，到C，到D，继续到E的路由，但是在这种情况下，节点F将不会出现在它自己的路由应答中返回的路由上。DSR路由发现禁止节点F从其缓存中返回这样的路由应答；这种禁止增加了产生的路由有效的概率，因为在这种情况下，如果路由之前已经停止工作，节点F将会收到一个“路由错误”。此外，这一禁令意味着，将来遍历路由的“路由错误”很可能通过发送路由回复的任何节点(包括节点F)，这有助于实现确保及时从缓存(如在 F)中删除陈旧数据；否则，由A发起的下一条路由发现，也有可能被F的载有相同的陈旧路由的路由应答所污染。由于基于路由缓存的信息返回路由应答的限制，如果节点F不返回这样的路由应答，节点F通常将会传播路由请求。

**3.3.3 路由请求跳数限制**

每个路由请求消息都包含一个“跳数限制”，可以用来限制允许转发路由请求副本的中间节点的数量。这种跳数限制是使用承载路由请求的数据包中的IP报头中的生存时间字段来实现的。当请求被转发时，这个限制就会减少，如果在找到目标之前限制达到零，那么请求数据包就会被丢弃。该路由请求跳数限制可用于在路由发现尝试期间各种控制路由请求传播的算法的实现。

例如，节点可以使用此跳数限制来实现作为路由发现的初始阶段的“非传播”路由请求。使用这种技术的节点发送其对某个目标节点的第一个路由请求尝试，其跳数限制为1，这样，接收到路由请求的初始传输的任何节点都不会通过重新广播将请求转发到其他节点。这种形式的路由请求称为“非传播”路由请求；它提供了一种廉价的方法，用于确定目标当前是否是发起者的邻居，或者邻居节点是否有到达缓存的目标的路由（有效地使用邻居的路由缓存作为发起者自己的路由缓存的扩展）。如果在短超时后没有收到路由回复，则该节点为目标节点发送“传播”路由请求（即具有由DiscoveryHopLimit配置变量的值定义的跳数限制）。

作为另一个例子，节点可以使用这个跳数限制来实现对目标的“扩展环”搜索。如上文所述，使用此技术的节点发送初始的非传播路由请求；如果没有收到它的路由应答，则节点发出另一个路由请求，其跳数限制为2。对于发起的每个路由请求，如果没有收到路由应答，则该节点会将上次尝试中使用的跳限制加倍，以不允许路由请求在整个网络上传播的条件下逐步探索目标节点。然而，这种扩展环搜索方法可能会增加路由发现的平均延迟，因为在发现到达目标节点的路由之前，可能需要多次发现尝试和超时操作。

**3.4 附加路由维护的特点**

**3.4.1 数据包回收**

当转发包的中间节点通过路由维护检测到该数据包的路由上的下一跳中断时，如果该节点在其路由缓存中有另一条路由到该数据包的目的地，则该节点应该“回收”该数据包，而不是丢弃它。为了挽救数据包，节点将数据包上的原始源路由替换为其路由缓存中的路由。然后，该节点将该数据包转发到沿此源路由指示的下一个节点。例如，在3.2节示例中显示的情况下，如果节点C有另一条缓存到节点E的路由，它可以通过将数据包中的原始路由替换为这个新的路由来从自己的路由缓存路由回收数据包，而不是丢弃包。

在回收一个数据包时，在它被回收的次数中会保持一个计数器，以防止一个包被无休止的回收。否则，由于每个节点只减少以此生存时间数（TTL），因此单个节点可以无限次回收一个数据包。我们选择在每次回收时减少生存时间数，回收数据包也是一项昂贵的操作，所以希望将数据包可以被回收的最大次数与数据包可以被遍历的最大跳数分开绑定。

如3.2节所述，在本例中，中间节点通过路由维护检测正在转发的数据包沿路由的下一跳中断，节点还应将“路由错误”返回给数据包的原始发送方，以标识无法转发数据包的链路。如果节点发送此路由错误，则在回收该数据包之前先发起路由错误。

**3.4.2 在中断链路上的排队数据包**

当转发包的中间节点通过路由维护检测到该数据包的路由上的下一跳链路中断时，除了处理为路由维护而定义的数据包之外，节点还应该以类似的方式处理它在这个新中断的链路上排队的所有挂起的数据包。具体来说，节点应该在其网络接口队列和维护缓冲区中搜索下一跳链路是该新中断链路的数据包。对于当前在此节点排队的每个此类数据包，节点应按以下方式处理该数据包：

-从节点的网络接口队列和维护缓冲区中删除数据包。

-使用8.3.4节中描述的过程，向数据包的原始发件人发出此数据包的“路由错误”，就像该节点已经达到了为路由维护而对该数据包进行重传尝试的最大次数。但是，在为排队的数据包发送这种路由错误以响应检测到的单个新中断链路时，节点应向这些数据包的每个原始发送方发送至多一个“路由错误”。

-如果该节点在其路由缓存中有另一条路由到该数据包的IP目的地地址，则该节点应该按照8.3.6节中的描述来回收该数据包。否则，节点应丢弃数据包。

**3.4.3 路由自动缩短**

如果不再需要路由中的一个或多个中间节点，则可以自动缩短正在使用的源路由。这种自动缩短使用中的路由的机制有点类似于“passive ACK”的使用。特别是，如果节点能够偷听到包含源路由的数据包（例如，通过在杂乱接收模式下操作它的网络接口），则此节点将检查该源路由的未扩展部分。如果此节点不是数据包的预定下一跳目的地，而是在数据包源路由的稍后未扩展部分中命名的，则可以推断源路由中自身之前的中间节点不再需要。例如，下图举例说明了节点D听到从B发送到C的数据包，以便稍后转发到D和E的示例：

+-----+ +-----+ +-----+ +-----+ +-----+

| A |---->| B |---->| C | | D | | E |

+-----+ +-----+ +-----+ +-----+ +-----+

\ ^

\ /

---------------------

在这种情况下，此节点(节点D)应向数据包的原始发送方(节点A)返回“无偿的”路由应答。路由应答给出了较短的路由，将原始源路由的一部分通过发送重传数据包的节点连接起来(节点B)，并加上原始源路由的后缀，以返回无偿路由应答的节点(节点D)开始。在本例中，由D发送到A的无偿路由应答消息中返回的路由给出了从A到B，到D，到E的跳数序列的新的路由。

当决定是否以这种方式返回一个无偿的路由应答时，节点可能会考虑额外的信息，而不仅仅是它能够偷听到数据包。例如，只有当接收到的信号强度或信噪比超过特定阈值时，节点才决定返回无偿路由应答。此外，每个节点维护一个无偿路由应答表，以限制其对同一返回路由的无偿路由回复的比率。

**3.4.4 增加“路由错误”消息的传播**

当源节点接收到它发起的数据包的路由错误时，此源节点通过在其下一个路由请求上支持它将该“路由错误”消息传播到其邻居。这样，源节点周围节点缓存中的陈旧信息将不会生成包含与此源节点接收路由错误相同的无效链路的路由应答。

例如，在第3.2节的示例中，节点A从C处的路由错误消息中了解到，从C到D的链路当前已中断。因此，它从自己的路由缓存中删除了这个链路，并启动了一个新的路由发现(如果它在路由缓存中没有到E的其他路由)。在发起此路由发现的路由请求数据包上，节点A携带此“路由错误”的副本，确保“路由错误”传播到其他节点，并保证它在响应此路由请求时收到的任何路由答复(包括来自其他节点的路由缓存)都不包含假定存在此断开链路的路由。

**3.5 可选DSR流态拓展**

本节描述了DSR协议的一个可选的、兼容的扩展，称为“流状态”，它允许在数据包中没有显式源路由头的情况下对大多数数据包进行路由。DSR流状态扩展进一步降低了协议的开销，但仍然保留了DSR操作的基本特性。一旦发送节点发现了源路由，例如通过DSR的路由发现机制，流状态机制将允许发送节点基于此源路由在网络中建立逐段转发状态，以使沿路由的每个节点根据节点自己对该包所沿的流的本地路由信息将包转发到下一个跳点。流状态由第一个数据包使用源路由动态初始化，然后，就可以在不使用数据包中的源路由报头的情况下沿着相同的流路由启动后续的数据包。在流程的每一跳处建立的状态是“软状态”，因此，在不再需要时自动过期，并且可以根据需要快速重新创建。扩展DSR的基本操作基于每个数据包路由的报头中的显式源路由，流状态扩展是一种“隐式源路由”，它保留DSR的基本操作，但从数据包中删除显式源路由。

**3.5.1 流的建立**

向某个目的节点发送分组的源节点可以使用这里描述的DSR流状态扩展来建立以流量的形式到该目的地的路由。“流”是表示为由沿途节点内的逐跳转发状态的从源到目的地的路由。每个流由源节点地址、目标节点地址和源节点选择的流标识符(流ID)的组合来唯一标识。

每个流ID都是一个16位的无符号整数。当比较两个不同的流ID时，必需要进行2的16次方运算。例如，使用C语言来进行实现，如果(Short)(A)-(B)>0，C语言“Short”数据类型被实现为16位有符号整数，则流ID值(A)大于另一个流ID值(B)。

数据包中的DSR流状态头部标识在转发该数据包时要遵循的流ID上。从给定的源到某个目的地，任何数量的不同流都可能存在并正在使用，例如，按照不同的跳序到达目的地。其中一个流可以被认为是从该源到该目的地的“默认”流。既没有DSR选项头指定要选择的路由(DSR选项头中有源路由选项)，也没有DSR流状态标头指定要遵循的流ID的接收数据包的节点沿着包的IP报头中指定的源地址和目标地址的默认流转发。

在建立新流时，源节点为此(源、目标)对生成一个大于任何未过期流ID的新的非零16位流ID。如果源希望此流成为默认流，则流ID必须设置低位(流ID为奇数)；不能设置低位(流ID是偶数)。

建立新流的源节点然后发送包含具有源路由选项的DSR选项报头的数据包；要建立流，源节点还必须在数据包中包含一个DSR流状态标头，其流ID字段设置为新流的所选流ID，并且必须在dsr选项标头中包含一个超时值选项，给出有关此流的状态信息将过期的生存时间。此数据包通常是从该发送方发送到接收方的正常数据包(例如，发现新路由后发送的第一个数据包)，但也被视为“流建立”数据包。

源节点在其流表中记录此流以供将来使用，将此流表项中的TTL设置为数据包IP头中TTL字段中使用的值，并将此项中的生存期设置为DSR选项头中超时选项中指定的生存期。TTL字段用于默认流转发，如3.5.3和3.5.4节所述。

在超时之前使用此流ID发送的任何其他数据包，如果还包含带有源路由选项的DSR选项报头，则必须在“源路由”选项中使用相同的源路由。

**3.5.2 接受和转发建立数据包**

如3.5.1所描述的一样，用于建立流的数据包内包含带有源路由选项的DSR选项头，并沿着指定的路由转发。当实现DSR流状态扩展的节点接受和转发这样的DSR数据包时，还在自己的流表中保持某种状态，以使其能够在仅指定流ID的情况下转发沿此流发送的未来数据包。具体来说，如果数据包还包含一个DSR流状态报头，那么该数据包应该在每个节点沿该数据包路由上的流表中为该流建立一个条目。

DSR流状态报头的跳数字段作为DSR选项报头中的指定生存时间选项也存储在流表中。

如果流ID为奇数且流表中没有流ID大于接收流ID的流，则将此对(IP源地址、IP目标地址)的默认流ID设置为接收到的流ID，并记录包的TTL。

流ID选项在包的最终传递之前被移除。

**3.5.3 沿已建立的流发送数据包**

当如3.5.1节所述建立流时，被发送的数据包在路由沿线的每个节点中建立状态。此状态是软的；也就是说，协议包含从丢失状态中恢复的机制。然而，这些机制的使用可能会导致在被遗忘状态下沿流发送的数据包的性能降低。因此，需要基于发送方是否合理地确定沿着路由的每个节点上存在流状态来区分行为。如果路由上所有节点的流表均包含该流的转发信息，则我们将流的状态定义为“端到端的建立”。虽然不可能在不发送数据包的情况下检测流的状态是否已达成端到端的建立，但是，实现可能会对流状态的保留和建立包被路由上的所有节点看到的概率做出合理的假设。

**8 详细操作**

**8.1 一般数据包处理**

**8.1.1 发包**