一种适用于 Ad Hoc 网络的拥塞控制算法*)

唐 伟 郭 伟 苏 俭

(电子科技大学通信抗干扰技术国家级重点实验室 成都 610054)

摘 要 Ad hoc 网络是一种无基础设施、无中心控制的分布式自组织网络,在紧急情况下能够迅速搭建。目前,在 IEEE802.11 协议基础上所搭建的 ad hoc 网络面临的主要问题是在信道达到饱和时,其链路层时延明显增加,以至于其上层的协议无法正常工作。本文提出了一种结合链路层及传输层的拥塞控制算法,通过对传输层拥塞窗口的控制、以及引入报文生命期及优先级,使得网络即使在大业务量时,链路层依然能够保持很低的时延,同时大幅度地提高传输层吞吐率。最后通过仿真,验证了该算法的有效性。

关键词 Ad hoc 网络,拥塞控制

A Congestion Control Algorithm for Ad Hoc Network

TANG Wei GUO Wei SU Jian

(National Communication Laboratory of UESTC of China, Chengdu 610054)

Abstract Ad hoc network is a kind of self-organizing network, and it requires no infrastructure, nor any central control. It can be rapidly constructed under emergency conditions. A major problem for IEEE802.11-based ad hoc network is the super high link-layer delay when the channel is fully loaded, which prevents upper-layer protocols from working properly. By introducing modifications on both transport layer and link layer, we propose a congestion control algorithm that employs congestion window, data packet lifetime management, and packet priority control. The algorithm significantly reduces the link-layer delay and greatly improves the throughput of the network under such conditions.

Keywords Ad hoc network, Congestion control

1 引言

随着 IEEE802.11 无线链路协议[1]的提出和完善,人们以之为基础设计出了无需基础设施的、可随时组建的网络——无线自组织网络,亦称 ad hoc 网络。这种网络具有分布式动态组网的能力,无需中心控制。每一个节点既是主机又要担负起路由器的责任,报文通过节点转发,每一次发送/转发称为一跳,所以这种网络也称为多跳网络。目前主要应用在紧急情况以及战场条件下的组网,具有生存期短、健壮性强等特点。正是由于 ad hoc 网络应用场合的特殊性,引发了人们对其网络性能的研究。

以下为了说明问题,首先引入几个概念。

· 吞吐率:成功发送的数据量÷总的时间。某一层(链路层、传输层等)的吞吐率,就是该层成功发送的数据量÷总的时间。网络吞吐率指传输层的吞吐率。

· 信道利用率:链路吞吐率和信道数据率的比值。

· 信道饱和:随着负载的增加,链路吞吐率也会随之增加,当网络负载增加到一定程度后,链路吞吐率就几乎不再增加,甚至下降,此时信道的传输能力达到极限,这种状态称为信道饱和状态。

· 链路层时延:从报文入链路层数据队列开始,到该报文传输到目的端链路层为止的时间。

· 信道接入时延:从报文入链路层数据队列开始,到该报文获得链路层信道占用权为止的时间。

· 报文传送时延:从获得链路层信道占用权后传输报文开始,到该报文传输到目的链路层为止的时间;也就是链路层时延和信道接入时延的差值。

· 网络拥塞:对于网络中的某个或某些节点,由于需要发送和转发的报文所要求的带宽大于其实际可用的带宽,而使得这些报文在这些节点处堆积,甚至于出现这些节点的数据队列溢出,这种状态就叫做网络拥塞。

网络拥塞包含两个因素,其一是由本节点产生的源发报文,其二是来自其它节点的转发报文。下面,首先讨论源发报文的情况。

在 ad hoc 网络中,送往链路层的每一个报文,都需要等待数据队列中先前的报文发送完毕,同时每个报文在发送之前又必须与邻居节点的报文竞争信道资源。当链路层的数据队列很长时,即使忽略由信道竞争所产生的时延,报文排队等待时延仍然很大。

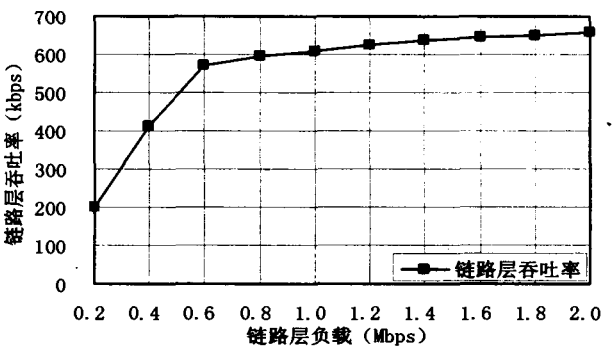

图 1 链路层负载与链路层吞吐率的关系

图 1 是在 20 个密集放置的节点(两两互为邻居),每个报

*)国家自然科学基金资助项目(60472052)。唐 伟 硕士研究生,主要从事无线自组织网络组网的研究。

文从 1kb 到 10kb，报文到达时间服从均值为 0.1 秒的泊松分布，每个节点链路层的负载分别从 10kbps 到 100kbps，信道数据率都为 1Mbps 的十个场景下，各仿真 3 分钟的统计结果，可以看到在负载超过 0.6Mbps 之后，吞吐率已没有明显增长，说明信道已达到饱和。

在信道饱和时，链路层时延超过了 6 秒，如图 2 所示。在如此高的链路层时延下，搭建在链路层以上的任何协议都难以正常运行。图 2 也显示出，想要有效地降低链路层的时延，只需减轻链路层的负载就可以了。对于源发报文，由于本节点传输层可以容易地控制其发送速率，这是很容易解决的。下面讨论转发报文的情况。

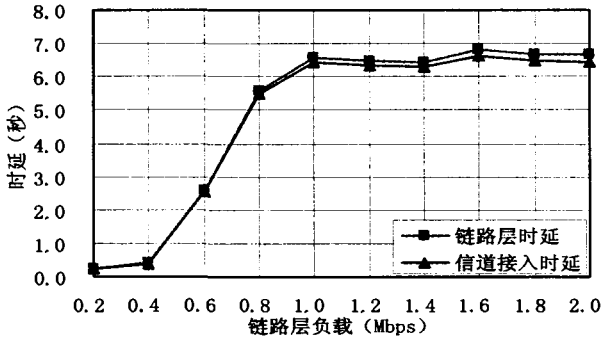

图 2　链路层负载与链路层时延及信道接入时延的关系

除了共享信道本身的竞争性，ad hoc 网络还有其自身的特点，尽管它是在共享信道上运行的，但是它不像传统的有线局域网，它是分布式的，每一个节点都在和自己的邻居节点争夺信道资源，某一个节点对信道资源的占用率变化，会引起其邻居节点对信道资源的占用率的变化，进而该变化将会连锁地扩展到更大的范围；而在有线局域网上网关可以起到隔离作用，一个子网内部的数据流量一般不会影响到其它子网。因此一个节点仅仅把握自己与周围邻居间的信道分配关系是不够的——一个节点不能因为自己能够占用的信道资源多，而一味地发送报文以试图用尽该资源，这样做可能引起甚至加重网络其它部分的拥塞，不仅不能增加报文的发送成功率，还浪费了宝贵的电池资源，作者称此为盲目发送问题。

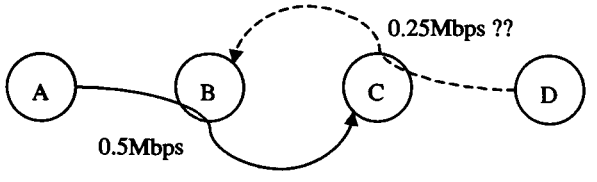

图 3　盲目发送问题的示意网络

为了演示盲目发送问题，举图 3 所示的简单无线网络。每个节点的信道数据率都是 1 Mbps，在信道公平竞争的前提下，考虑竞争信道不需要耗费带宽的理想情况。设节点 A 有 0.5 Mbps 的数据需要通过 B 发送给 C。此时 A 与 B 竞争各占用 0.5 Mbps 的带宽用来发送，网络刚好不拥塞。但是，对于节点 D 来说，它测出自己的带宽除了因为 B 在向 C 发送时自己不能发送而损失的 0.5 Mbps 外，还有 0.5 Mbps 可以用来发送数据。如果此时 D 有 0.25 Mbps 的数据需要通过 C 被转发到 B，问题就出现了——就算 D 完全趁着 A 向 B 发送数据的时候，向 C 发送了这 0.25 Mbps 的数据，但是因为 A 在向 B 发送时，B 和 C 都不能发送，可供 B 和 C 使用的带宽就只有 0.5 Mbps，然而需要发送的数据达到 0.75 Mbps，不能满足要求，网络势必产生拥塞。图 3 还只是最简单的情况，对于一个节点数很多且具有移动性的复杂网络，这种问题会更显突出。

转发节点通常遵循“谦逊地源发，慷慨地转发”(be conservative in what you do, be liberal in what you accept from others)的原则，因而由转发报文引起的链路层负载难以被转发节点控制，转发报文因素就成为了网络拥塞的最重要因素。所以解决盲目发送问题，是任何适用于 ad hoc 网络的拥塞控制算法的关键所在，这一点还会在后面的仿真结果中体现。基于这样的理由，并依据 ad hoc 网络的特点，本文提出了一种结合链路层及传输层的拥塞控制算法，该算法不仅大大地降低了链路层时延，还显著地提高了传输层吞吐率。

2　算法设计

减小链路层时延，目前已有的方法是采用链路层应答机制，其核心思想在于限制链路层数据队列的大小——当链路层认为该队列足够小时（通常是当队列中没有上层数据报文的时候），向传输层发送一个准入应答，传输层收到该应答后再把下一个数据报文交给链路层发送。实际上，这种方式和通常的不控制链路层数据队列大小的方式之间并没有本质的区别——两者的链路层发送完一个数据报文后，都立即取下一个进行发送，其区别仅在于数据报文暂存在传输层还是暂存在链路层。该机制没有对网络拥塞的感知能力，因而对解决网络拥塞无能为力。

借鉴传统的有线网络中已较完善的拥塞控制算法，比如 TCP 的拥塞控制算法[3]或者尚未形成标准的 DCCP 算法[4]，本文所提出的算法采用端到端时延来感知网络拥塞，于是引入了滑动拥塞窗口和数据报文的端到端应答。该算法面向报文传输，主要应用于 UDP 协议，亦可用于 TCP 协议。

在网络拥塞时，链路层时延很大，造成传输层报文超时，此时传输层会重发该报文，但是原先的报文并没有被中间节点丢弃，于是同一报文在网络中同时出现多个副本，浪费了网络资源，降低了传输层吞吐率。为避免这种情况，目前已有的方法，是采用逐节点应答机制，即一个节点的链路层将一个报文成功发送到下一个节点的链路层后，给传输层一个应答，传输层就认为该报文发送成功，不再重传，且将该报文从自己的数据队列中去掉；或者是反过来，链路层将传输失败的报文的序列号应答给传输层，传输层就认为之前的报文（序列号更早的报文）发送成功。该方法非常简单，也不需要应答机制，但是该方法也没有对拥塞的感知能力，而且该方法还有另一个问题，即报文可能因为中间的某个转发节点的传输层数据队列溢出而被丢弃掉，源节点对此却无能为力，这种情况恰恰出现在网络负载很大的时候。

对此，控制报文的生命期是最根本的解决办法。所谓控制报文的生命期，就是要让超时的报文从网络中消失掉。本算法在传输层报文中加入一个域，指明报文的生命期，并在网络不同步的情况下，对报文在各节点的时延进行估计，即估计一个报文从进入传输层数据队列到被链路层成功发送至下一个节点的这段时间。这段时延分两个部分——首先，是报文在传输层的时延，由于是在同一个节点中，这可以由传输层自己确定；第二，是报文在链路层的时延，精确地讲，该时延应该在网络同步的情况下，源链路层对报文打上时间戳，由目的链路层计算得出；在网络不同步的情况下，则只能依靠源链路层

估计。从前面的图 2 中看到，报文的链路层时延和信道接入时延几乎相等，而信道接入时延完全是由本节点自己确定的。这就是说即使在网络不同步的情况下，依然可以使用信道接入时延较精确地估计链路层时延。本算法更进一步，使用报文的最后一个分段的信道接入时延估计其链路层时延。

除了以上措施，本算法还考虑了不同报文之间的优先级问题。在 ad hoc 网络中，每个节点都在自由移动，路由层需要尽快地得知网路拓扑的变化，而传输层的应答报文也应该尽快传回源节点，如果简单地把控制报文和数据报文按时间顺序放入链路层的队列，显然是不妥的，控制报文应该优先被发送。所以本算法还引入了优先级控制机制。

经过上面的分析，最后将本文提出的拥塞控制算法归纳如下：

2.1 时延估计算法

$$Diff = RTT_{Sample} - RTT_i$$
$$RTT_{i+1} = RTT_i + \delta * Diff$$
$$Dev_{i+1} = Dev_i + \rho * (|Diff| - Dev_i)$$
If($Dev_{i+1} < 0$)
$$Timeout = RTT_{i+1} + Dev_{i+1}$$
Else
$$Timeout = RTT_{i+1} + \eta * Dev_{i+1}$$
If($Timeout > Timeout_{max}$)
$$Timeout = Timeout_{max}$$

其中，RTT_{Sample} 表示本次实际测出的端到端时延，RTT_i 表示端到端时延的估计值，而 *Timeout* 表示超时时延的估计值；依照标准[3]，取 $\delta = 2^{-3}$，$\rho = 2^{-2}$，$\eta = 3$，而 $Timeout_{max}$ 取为路由表项的最大生命期（过长的 *Timeout* 值，一味地等待，反而不利于及时地对网络变化情况作出反应，因此以路由表项的最大生命期为最大值）。在这里通过减缓超时时延的降低速率，来获得比较稳定的拥塞窗口（见下）。

2.2 拥塞窗口大小的控制算法

拥塞窗口大小 L_{CW} 初始值为 1，使用 *Timeout* 估计报文应答时延；如果超时，则加倍 *Timeout*，减半 L_{CW}；如果没有超时，则使用获得的 RTT_{Sample} 值估计下一次的 *Timeout*，并且将 L_{CW} 增加 1。

2.3 报文生命期控制算法

在传输层数据报文中加入生命期 *Lifetime* 域，源节点使用超时时延 *Timeout* 设置该域；如果该报文在某个转发节点的数据队列中排队时，生命期就完结了，则该节点丢弃此报文；否则在把该报文送入链路层时，将该报文的生命期减少其排队等待所花的时间。

传输层将报文送给链路层的同时，也将其时延告诉给链路层；如果该报文在某个链路层的数据队列中排队时，生命期就完结了，则丢弃此报文，并向传输层报告，传输层当作超时处理；否则在传输该报文最后一个分段前，携带该分段在链路层等待的时间（从报文进入链路层数据队列到它最后一个分段开始发送的这段时间），对方节点的链路层将这个时间作为链路层时延估计，并报告给其传输层，传输层再把该报文的生命期减少此时延。

2.4 报文的优先级控制

传输层将报文分为两个等级，给控制报文较高的优先级，而给数据报文较低的优先级，当传输层向链路层递交报文的时候，同时告知报文的优先级，链路层将优先级高的报文排放在数据队列中优先级低的报文之前，同等级报文依然按照 FIFO 的规则排列。

以上就是本文所提出的拥塞控制算法，下面通过就仿真结果来验证该算法的效能。

3 仿真实验

本文通过 OPNET 网络仿真软件对上面提出的算法进行仿真验证。在 1km² 范围内随机放置 30 个节点，每个节点的信道数据率都是 1Mbps，发射机信号覆盖范围为 300m，所有节点都以 20km/h 的速度自由移动；每个节点的网络负载为从 25kbps 开始，每次增加 25kpbs，直到 150kbps（每报文大小固定，从 2.5kb 开始，每次增加 2.5kb，直到 15kb，报文到达时间间隔是均值为 0.1 秒的泊松分布）；每个节点的每次会话包含 10～100 个报文，每次会话都随机选择目的节点；路由层采用 AODV 路由协议[2]；共 12 次仿真，每次网络运行时间都是 3 分钟，对比了没有拥塞控制和有拥塞控制的两种情况。

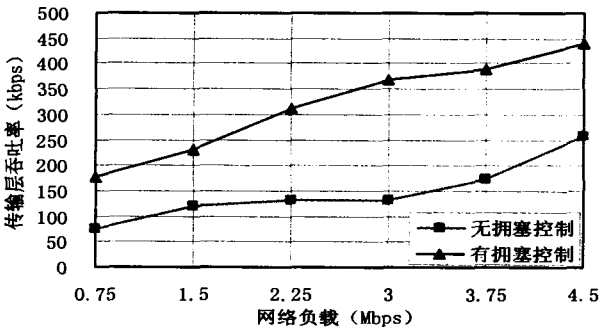

图 3 网络负载与传输层吞吐率的关系

图 3 比较了两者传输层的吞吐率。从仿真结果中清楚地看到，在加入了本文提出的拥塞控制算法后，传输层吞吐率增加了一倍。

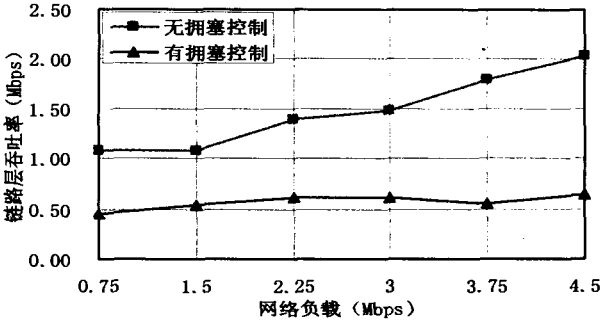

图 4 网络负载与链路层吞吐率的关系

图 4 对比了两者链路层的吞吐率。使用了本文提出的拥塞控制机制的网络，可以在使用不到一半的链路层吞吐率的情况下，获得两倍的传输层吞吐率。

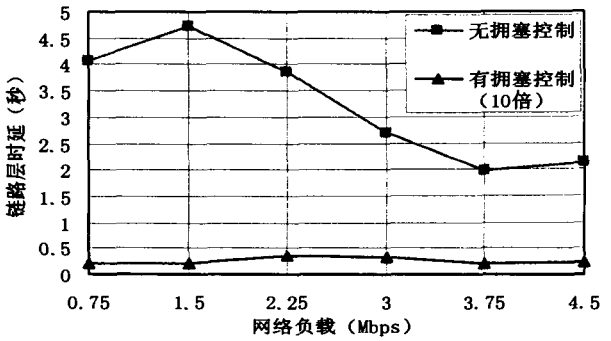

图 5 网络负载与链路层时延的关系

（下转第 107 页）

共享还是 BST 浓缩都能达到较好的效果。因此查询性能更好。但是在气象数据集上面，前缀立方的性能却是最差的，这是因为在真实数据集上数据比较稠密，选择度越低，按 z-order 码进行多维聚簇的效果越有利于范围查询。选择度较高时，如选择度为 1 时，要查询整个数据小方，这时，如果同一个数据小方的元组按维序被聚簇到一起，自然有利于查询，所以这时候前缀立方查询性能变得好一些。

结论 本文基于 CuboidTree 索引，提出了前缀立方的一种索引机制 Prefix-CuboidTree。总体来说，采用这种索引机制对前缀立方进行查询能取得较好的查询效果。实验表明，在气象数据集和自相似数据集上，其查询性能随着数据立方的维度增大而下降，随着选择度增加而上升。这主要是因为元组按维序进行聚簇会导致在维度增大时其物理存储邻近性和多维空间的点邻近性的差异增大。选择度增加，能平衡这方面的差异，从而使范围查询性能得到一定的改善。

参 考 文 献

1 Gray J，Bosworth A，Layman A，et al. Data Cube：A Relational Operator Generalizing Group-By，Cross-Tab，and Sub-Totals. In：Proc. of the Int. Conf. on Data Engineering，1996

2 Wang W，Feng J，Lu H，Yu J X. Condensed Cube：An Effective Approach to Reducing Data Cube Size. In：Proc. of the Int. Conf. on Data Engineering，2002

3 Sismanis Y，Deligiannakis A，Roussopoulos N，Kotidis Y. Dwarf：Shrinking the PetaCube. In：Proc. of the ACM SIGMOD Int. Conf. on Management of Data，2002

4 Lakshmanan L V S，Pei J，Han J. Quotient Cube：How to Summarize the Semantics of a Data Cube. In：Proc. of Int. Conf. on Very Large Data Bases，202

5 Lakshmanan L V S，Pei J，Zhao Y. QC-Trees：An Efficient Summary Structure for Semantic OLAP. In：Proc. of the ACM SIGMOD Int. Conf. on Management of Data，2003

6 Feng J，Fang Q，Ding H. PrefixCube：Prefix-sharing Condensed Data Cube. To appear：ACM International Workshop on Data Warehousing and OLAP，2004

7 Feng J，Si H，Feng Y. Indexing and Incremental Updating Condensed Data Cube. In：Proc of the Int. Conf. on Scientific and Statistical Database Management，2003

8 Orenstein J A，Merret T H. A Class of Data Structures for Associate Searching. In：Proc. of ACM SIGMOD-PODS Conf.，Portland，Oregon，1984. 294～305

9 Oracle INC. Oracle 9i Spatial，An Oracle Technical White Paper. 2001

10 Hahn C，Warren S，London J. Edited. Edited synoptic cloud report from ships and land stations over the globe，1982～1991

（上接第 43 页）

图 5 比较了两者的链路层时延。在引入了本文提出的拥塞控制后，缓解了链路层的压力，链路层时延变得很小，即使扩大成 10 倍，也远小于没有拥塞控制的情况。

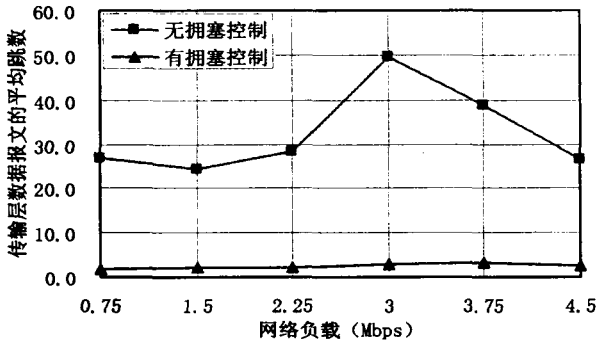

图 6 网络负载变化与传输层数据报文平均跳数的关系

图 6 对比了两者传输层报文的平均跳数，即传输层总的发送与转发次数÷成功发送的报文数目。两者相差相当悬殊，这说明加入了本文提出的拥塞控制机制后，传输失败的次数大大地减小了。

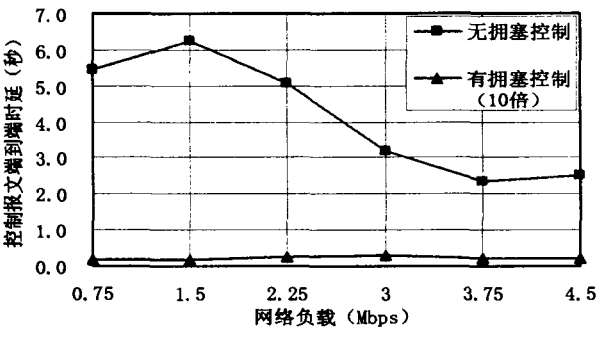

图 7 网络负载与控制报文端到端时延的关系

图 7 比较了两者路由层控制报文的平均时延。通过本文提出的拥塞控制算法，路由协议的运行不会受到过大的网络负载的影响。这里显示出了一个有趣的现象，在没有使用拥塞控制算法的情况下，控制报文的平均链路层时延居然大于总的平均链路层时延（如图 5），也就是说它大于数据报文的平均链路层时延。作者分析，这是由于盲目发送而引起的——有许多数据报文因为其发送/转发节点附近的链路层比较空闲而被发送了，这些数据报文的时延会比较短，加之数据报文的数量远大于控制报文，所以造成了这种现象。

结论 仿真结果验证了该算法的具有很好的效能——不但大大地降低了链路层时延，也显著地提高了传输层的吞吐率；同时也印证了前文对 ad hoc 网络的分析。制约 ad hoc 网络吞吐性能的因素主要有这两点：盲目发送问题以及同一报文的多个副本在网络中同时出现，这两个因素都无谓地加重了链路层的负担，特别是盲目发送问题。从结果图中可以看出，链路层的巨大网络负载带来了巨大的吞吐量（发射机工作时间长，电池能量消耗多），这加重了报文竞争信道时碰撞的可能性，造成发送失败率高，加大了发送时延，也就增加了链路层的时延，同时导致传输层的低吞吐率。而第二个问题，则是由链路层时延过大，又没有必要的控制措施而引起的。总之，要想解决这些问题，需要把握 ad hoc 网络的特点，实现链路层和传输层的通力配合，而传统的有线网络上的拥塞控制不能适应 ad hoc 网络的特点，这也正是本文提出的拥塞控制算法的优点所在。

参 考 文 献

1 IEEE 802.11 Working Group. http://grouper.ieee.org/groups/802/11/index.html

2 AODV Routing Protocol，rfc3561.txt. http://www.ietf.org/rfc/rfc3561.txt

3 Internetworking with TCP/IP Volume I：Principles，Protocols，and Architecture，3rd Edition. Douglas E. Comer，Department of Computer Sciences，Prudue University，West Lafayette

4 Datagram Congestion Control Protocol（DCCP）. http://www.ietf.org/internet-drafts/draft-ietf-dccp-spec-06.txt