基于 Qt 的图像数据网络传输应用研究

周名阳[1]， 韩秀玲[1,2]

(1. 东华大学 信息科学与技术学院，上海 201620；2. 数字化纺织服装技术教育部工程研究中心，上海 201620)

摘 要：基于 ARM 的嵌入式硬件平台和嵌入式 Linux 操作系统，利用 Qt 的 Socket 类及信号/槽机制，采用图像数据的分块算法及反馈控制机制，设计并实现了一个带有图形用户界面的嵌入式图像数据传输方案。该方案使用了异常处理机制，保证了传输的可靠与高效，为嵌入式网络应用开发提供了一个参考模型。系统运行实例结果表明了该方法的可行性。

关键词：图像数据； 网络传输； Qt； 信号与槽； 嵌入式系统

中图法分类号：TP393.09 **文献标识码**：A **文章编号**：1000-7024 (2011) 06-1901-05

Application research on image data network transmission based on Qt

ZHOU Ming-yang[1], HAN Xiu-ling[1,2]

(1. College of Information Science and Technology, Donghua University, Shanghai 201620, China; 2. Engineering Research Center of Digitized Textile and Fashion Technology, Ministry of Education, Shanghai 201620, China)

Abstract: Based on ARM embedded hardware platform and embedded Linux operating system, using the Qt's socket, Qt's signal/slot mechanism, block-based image data algorithm and the feedback control mechanism, a transmission scheme of embedded image data is designed and implemented. In this scheme an exception handling mechanisms are also used to ensure efficient and reliable transmission. The scheme provides a useful reference model for embedded web application development. Finally, a running example of the system is presented to verify the feasibility of the scheme.

Key words: image data; network transmission; Qt; signal/slot; embedded system

0 引 言

网络传输是近年来嵌入式领域的一个广泛应用，远程视频监控就是其中的一个典型应用实例。远程视频监控系统主要包括图像数据的采集、传输、处理及显示等，网络传输无疑是其中的重要环节，尤其是当有大流量和突发数据时，如何保证传输的可靠与高效性，是嵌入式应用领域的重要研究课题。本文利用嵌入式 Qt 的开发优势、运用图像数据的分块算法及反馈机制，设计并实现了一个带有图形用户界面的嵌入式图像数据传输系统，实现了图像数据的可靠与高效传输。文中给出了传输系统的详细设计与实现过程。

1 基于 Qt 的网络传输机制

在 Linux 下进行网络编程，我们可以使用 Linux 提供的统一的套接字(Socket)接口。但是这种方法牵涉到太多的结构体，比如 IP 地址，端口转换等，不熟练的人往往容易犯很多错误。而 Qt 中提供的 Socket 完全使用了类的封装机制，使用户不需要接触底层的各种结构体操作。并且，由于采用了其自身的信号和槽(signal-slot)机制，使编写的程序更容易被理解[1-2]。信号和槽是 Qt 的核心机制，应用于对象之间的通信，其基本思想是：当对象改变其状态时，信号就由该对象发射(emit)出去，这就是对象所要做的全部事情，它不知道另一端是谁在接收这个信号。这就是真正的信息封装，它确保对象被当作一个真正的软件组件来使用。槽用于接收信号，但它们是普通的对象成员函数，一个槽并不知道是否有任何信号与自己相连接。当信号被发射时，与之相关联的对象的槽函数就会被调用。

Qt 提供的 QSocket 和 QServerSocket 类用于编写 TCP 客户端和服务器应用程序。QSocket 类提供了一个有缓冲的 TCP 连接，可以用来实现其它标准协议也可以用来实现自定义的协议。QSocket 采用异步工作方式，它依靠 Qt 事件循环发现外来数据和向外发送数据，并以信号的方式报告状态改变或产生的错误，一旦网络的某一种状态发生(如网络断开)，信号就会发送(如 connectionClosed())，再通过信号与槽函数相关联进行处理，即调用相关联的槽函数[3-4]。由于 QSocket 类继承 QIODevice 类(QIODevice 类是输入/输出设备的基类)，因此可以使用 QTextStream 和 QDataStream 这样的流结构类，从而大大方便了 TCP 数据流的读写。

收稿日期：2010-06-26；修订日期：2010-08-30。

基金项目：上海市自然科学基金项目(08ZR1400400)。

作者简介：周名阳(1986－)，男，四川达州人，硕士研究生，研究方向为嵌入式系统、无线传感器网络； 韩秀玲(1957－)，女，辽宁人，博士，高级工程师，研究方向为计算机网络、嵌入式系统。E-mail：zhoumingyang96901@yahoo.com.cn

QServerSocket 类在服务器端处理外来的 TCP 客户端连接，在构造函数中设置 IP 地址和端口号，一旦设置好 IP 地址和端口号，QServerSocket 便能侦听所有连到服务器的用户，再由成员函数 newConnection(int socket) 对最新连接到的用户做出反应。QDataStream 类提供了读写数据的基本功能，先利用 QDataStream 类对 QSocket 类的 socket 对象进行封装，然后利用 QDataStream 类的 readRawBytes (char*s, uint len)、writeRawBytes(const char*s, uint len)、操作重载函数“<<”和“>>”等函数实现网络数据传输。

2 基于 Qt 的图像数据网络传输

本文在 ARM 嵌入式硬件平台和嵌入式 Linux 操作系统下，利用 Qt 的 Socket 类及信号与槽机制，实现了客户－服务器模式的图像数据网络传输，并通过采用分块算法、反馈机制以及出异常处理机制等，进一步保证了图像数据传输的可靠与高效性。以下是该方案的设计细节。

2.1 客户端与服务器通信的信号流程控制设计

客户－服务器模式是常用的网络数据传输模式，客户端首先要发起连接，在建立起连接后，发送所捕获的图像数据，因此客户端需要有 3 个操作流程，分别是发起网络连接，传输图像数据和关闭网络连接。服务端主要负责对数据的接收并反馈相关信息以传输后面的图像数据，或让客户端程序根据反馈信息判断传输是否出错，如果出错就进行异常处理。3 个阶段通信控制流程的设计细节如下：

2.1.1 网络连接

客户端发起网络连接过程如图 1 所示。图 1 中，ARM 为客户端硬件平台，PC 为服务器端硬件平台。双方的连接建立过程如下：

客户端首先触发 btnNetwork 按键以产生 clicked()信号，从而去执行与之相关联的槽函数 NetworkControl()，如图 1 中①所示。槽函数 NetworkControl()首先从界面窗口中获取服务端的 IP 地址和端口号，然后调用 Qsocket 类的成员函数 open()打开 socket 对象，最后调用 Qsocket 类的成员函数 connectToHost()准备连接到服务器指定的端口。

服务端自定义 MyServerSocket 类，从 QserverSocket 类继承而来，对 newConnection()成员函数重载(newConnection()成员函数中创建 Qsocket 对象 s 并发射信号 newConnected(s))，并且定义 newConnected (QSocket*) 信号，如图 1 中②所示。在 frmserver 类的构造函数中初始化系统和相关图像传输控制变量，创建 MyServerSocket 类的实例对象 server，并关联 server 对象所产生的 newConnected(QSocket*)信号与 newConnectionArrived(QSocket *)槽，如图 1 中③所示。

在服务端初始化完成后系统就处于侦听状态，一旦客户端发起连接，服务端 server 对象就会发射 newConnected (QSocket*)信号，从而执行 newConnectionArrived(QSocket *)槽函数，如图 1 中④所示。槽函数 newConnectionArrived(QSocket *)中分别关联 socket 对象的 connectionClosed () 信号和 socketConnectionClosed()槽(槽函数 socketConnectionClosed()用于关闭网络连接)；readyRead()信号与 socketReadyRead()槽(槽函数 socketReadyRead()用于读取网络数据)。一旦成功建立连接，客户端 socket 对象就会发射 connected() 信号从而去执行与之相关联的槽函数 SocketConnected()，如图 1 中⑤所示。在槽函数 SocketConnected()中首先把网络使能标志 NetworkEnable 置为 TRUE，标志网络可用，然后关闭 CameraTimer 定时器，以后的图像数据采集都是靠服务端的反馈信号来决定的而不是由客户端的定时信号来决定。这样客户端与服务器的连接就建立起来了并为后面的网络传输做好了准备。

2.1.2 图像数据的传输

图像数据的传输过程如图 2 所示。

在网络建立连接以后，就可以对采集到的图像数据进行传输了。本方案中，图像数据的采集由 OpenCV 来完成，OpenCV 一个是开放源代码的计算机视觉类库，其大部分库函数基本上采用 C 或 C++语言编写，因此可以方便地移植到其它如 ARM、MPIS 等嵌入式微处理器中。OpenCV 提供了专门的摄像头数据采集函数，在 Linux 平台上提供的 cvCaptureFromCAM() 函数可以直接使用 V4L(Video4Linux 是 Linux 下用于获取视频和音频数据的 API 接口) 或 FireWire (IEEE1394) 接口来为从摄像头采集到的视频流分配和初始化 CvCapture 结构，然后调用 cvGrabFrame()函数从摄像头或者视频文件中抓取帧[5]。然而，由于图像数据量过大，如果直接传输帧，容易导致网络拥塞，因此在本方案中采用了分块传输的方法：

(1)图像数据的分块

服务器端接收的数据主要有状态信息数据及图像数据，接收端利用其信息头加以区分。其中状态信息数据头为“Basic”，是基本的控制信号，可用于功能的扩展。图像数据头为“<PICTURE>”。实际中状态信息的数据内容较小，在传输过

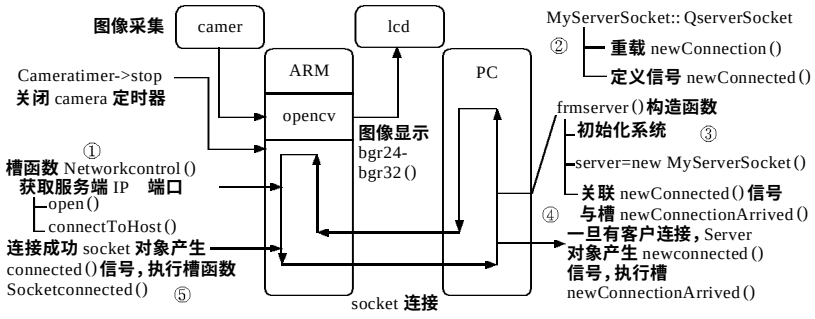

图 1 发起网络连接

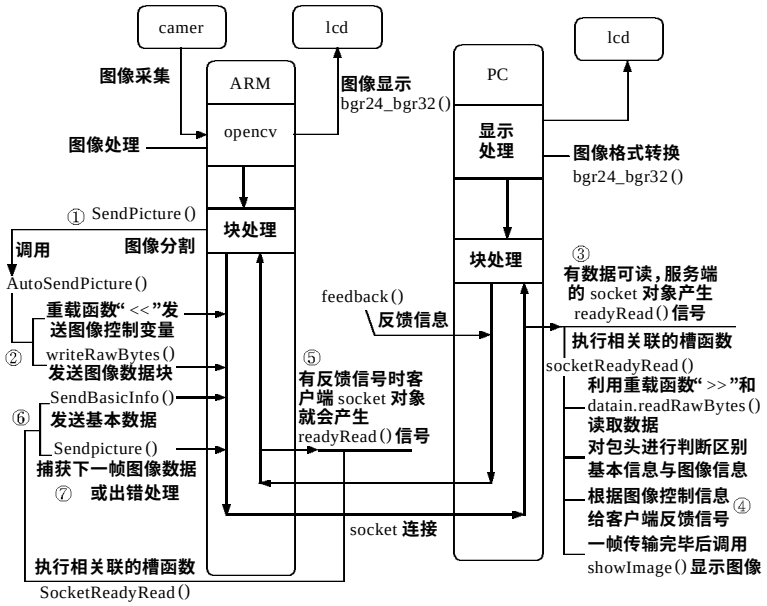

图 2　数据通信过程

程中不存在网络阻塞而丢数据包的情况，而对于图像数据而言，由于数据量较大，在传输过程中容易引起网络阻塞而丢数据包，最终导致视频图像无法恢复，因此在本方案中采用了分块传输的方法。所谓分块传输，即将需要传输的视频数据分成 n 个数据块，对每个数据块的传输进行通信握手，当一块数据完全传输到服务端后向客户端回送握手信号，客户端再发送下一个数据块，在一帧视频传输完毕后，再将视频数据进行合成。从而保证了数据传输的可靠性。

由于图像数据采用分块传输，因此在每接收一块图像数据之前都需进行通信握手。具体算法：将一帧 K 字节大小的视频图像分成 M 字节大小的数据块，需要传输的块数为 N = K/M + K%M。因此在每次数据传输过程中需要传输当前块、当前块的大小和需要传输总块数，接收端根据总的块数来判断一帧图像数据是否接收完毕。

(2)图像数据的传输

针对图 2，在网络处于连通的情况下，SendPicture()使用如上所述的图像分块算法实现图像分块并把分块后的数据存到发送缓冲区中，如图 2 中①所示。然后调用函数 AutoSendPicture()发送缓冲区中的图像数据及其控制信息，即利用 QDataStream 类的 writeRawBytes(const char*s, uint len)成员方法、重载函数“<<”往封装了 socket 的 QDataStream 类的 dataout 对象中写入要传输的数据，如图 2 中②所示。此时对于服务端来说 socket 中有数据可读，在服务端的 socket 对象就会产生 readyRead()信号从而执行相关联的槽函数 socketReadyRead()，如图 2 中③所示。在槽函数 socketReadyRead()中利用 QDataStream 类的重载函数“>>”、readRawBytes (const char*s, uint len)成员方法从封装了 socket 的 QDataStream 类的 datain 对象中读出客户端传输来的数据。首先对包头进行判断，如果为“Basic”，就获取基本的控制信息，该信息用于功能扩展，如果为“<PICTURE>”，下面获取的就是图像数据。在获取图像数据之前还要读到关于图像的控制信息 fileSize、times 和 send 的大小（它们用以判断传到一帧图像数据的第几块、是否传输完一帧图像数据以及是否有网络堵塞而丢包），服务端根据这些信息判断当前传输的是第几块数据，然后给客户端反馈传输下一块数据的握手信号，如图 2 中④所示。当一帧数据传输完毕后调用 showImage()函数在服务端显示该帧图像。showImage()的实现过程将在 2.2 节中加以说明。

(3)反馈机制

在服务端给客户端传输反馈信号后，对于客户端来说，socket 中有数据可读，客户端的 socket 对象就会产生 readyRead ()信号从而执行相关联的槽函数 socketReadyRead()，如图 2 中⑤所示。槽函数 socketReadyRead()首先读取反馈信号的包头是否为“<FEEDBACK>”，然后根据反馈的 send 判断一帧图像数据是否发送完毕，没有发送完就从发送队列中取出剩余的数据块继续发送该帧数据，发送完毕就调用 SendBasicInfo()发送基本的控制信息用于功能扩展，并且若摄像头处于工作状态就继续调用 SendPicture()函数去捕获下一帧图像数据，如图 2 中⑥所示。当获取到一帧新的图像数据后便按照如上所述的处理流程发送新的图像数据帧。

(4)异常处理

如果从反馈的信号判断出上一块数据传输失败就进行异常处理，这里采用重发数据块的方式进行异常处理。判断的依据是：客户端每发送一块数据就使当前发送的数据块变量 send 加一，准备发送下一块数据；服务端接收到数据块后，也会使当前接收的数据块变量 receive 加一，如果出错接收的数据块变量 receive 保持不变，当把变量 receive 的值反馈给客户端后，客户端比较反馈的变量的值是否等于 send 加一，如果相等就发送下一块数据，否则继续从发当前数据块。如图 2 中⑦所示。从而保证在出错的情况下能及时的纠错并且网络中的数据量会比重发整帧小得多。

以上是基于Qt的传输机制实现图像数据网络传输的基本思想与程序设计流程，其实就是通过往 socket 流中写数据，激发对方产生 readyRead()信号，从而去执行相应的处理函数[6]，在传输大量数据的时候通过采用分块算法、反馈机制与出错处理办法来保证数据传输的高效与可靠性。

2.1.3 断开网络连接

断开网络连接的处理流程如图 3 所示。

断开网络连接有两种情况，一种是客户端发起，另一种是由服务端发起，当一方关闭了 socket 连接后，会激发另一端的 socket 对象产生 connectionClosed()信号，从而执行相关联的槽函数关闭 socket 连接释放系统资源。具体设计流程如下：

(1)客户端发起断连

客户端触发 btnNetwork 按键后就会产生 clicked()信号，从而去执行与之相关联的槽函数 NetworkControl()，如图 3 中①所示。在槽函数 NetworkControl()中判断，如果之前网络处于连接状态就会调用 Qsocket 类的 close()方法关闭 socket 连接释放系统资源，如图 3 中②所示。当客户端关闭了 socket 连接后，服务端的 socket 对象会产生 connectionClosed()信号，从而执行相关联的槽函数 socketConnectionClosed()，如图 3 中③所示。在槽函数 socketConnectionClosed()中会把 socket 对象赋值为空以释放系统资源，如图 3 中④所示。

(2)服务端发起断连

当服务端首先断开连接时，客户端的 socket 对象同理也会产生 connectionClosed() 信号，从而去执行相关联的槽函数 ServerConnectionClosed()，如图 3 中⑤所示。在槽函数 ServerConnectionClosed()中也会关闭 socket 对象释放系统资源，如果摄像头可用的话就启动 CameraTimer 定时器去采集图像数据在本地显示，如图 3 中⑥所示。

2.2 服务端数据的恢复处理

服务端在得到数据后还需要对图像数据做相应的处理。本文仅对所获得的图像数据做了恢复处理使之在服务端显示出来，以验证本图像数据传输方案的运行效果。

服务端每接收完一帧图像数据便调用 showImage()函数以显示该帧图像。在 showImage()函数中首先定义 char 型指针变量 imagedata，使它指向存储图像数据的缓冲区。获取到图像数据帧以后，由于 IplImage 里的图像数据是 24 位真彩的三通道 BGR，而 QImage 为 32 位真彩的四通道(除 BGR 外，还有一个 a 通道)，所以在 Qt 中，需要对 IplImage 图像数据结构进行转换才能进行显示处理[7-8]，这里我们通过调用函数 bgr24_bgr32(int width,int height, char *src)实现了图像数据的转换，即通过循环将传入到函数 bgr24_bgr32()的图像数据“char *src”转化成为：width*height*4 的图像数据格式。转换代码如下：

```
char *frmclient::bgr24_bgr32(int width,int height, char *src){
  char *dst = 0, *tmp;
  dst = (char *)malloc(width*height*4);  //申请存储图像的数据空间
  tmp = dst;
  for(int y = 0; y < height;y++) {  //开始转化图像格式
    for(int x = 0; x < width; x++){
      for(int I = 0;I < 3;i++)   {
          *dst++ = *src++;
      }
      *dst++ = 0;
    }
  }
  return(tmp);
}
```

3 实验结果与分析

本文通过 Qpainter 构建了显示画板，并调用 Qpainter 的成员方法 drawImage()在基于 Qt 的图形界面中(如图 4 所示)显示了所传输的图像数据。

图 4 中右上角是图像数据恢复效果，其它部分是基本信号显示区(用于功能扩展)，实验结果表明，本方案是可行的，能有效地传输图像数据，在服务端得到了流畅的画面。

4 结束语

本文在基于 ARM 的嵌入式硬件平台和嵌入式 Linux 操作

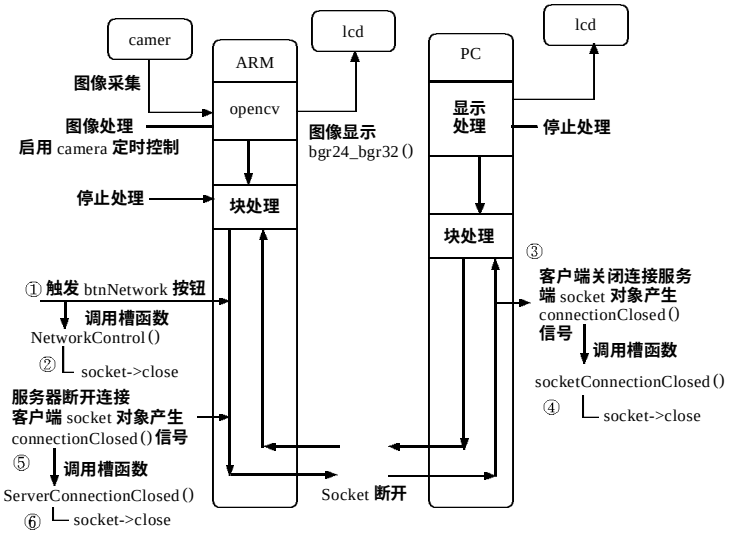

图 3 网络断开

图 4　服务端图像数据恢复效果

系统下，利用开放源代码的计算机视觉类库 OpenCV 捕获图像数据，并基于 Qt 的 Socket 类及信号与槽机制，设计了一个基于 Qt 的图像数据网络传输方案。文中给出了网络连接、数据通信、图像数据处理以及网络断开的详细设计与实现过程。为了实现数据的安全高效传输，还引入了图像数据的分块算法、反馈机制与异常处理机制。但在异常处理上本文仅仅采用了简单的基于数据块的重发机制，后期将进一步研究和实现基于字节的校验方法，以进一步提高系统的纠错和传输效率。

参考文献：

[1] Andrew S Tanenbaum.计算机网络[M].北京:清华大学出版社,2004.

[2] Richard Stevens W,Stephen A Rago.UNIX 环境高级编程[M].北京:人民邮电出版社,2006.

[3] 谭平,刘建新.基于 QT 的嵌入式 CAN 网络监控 GUI 系统[J].计算机工程与设计,2008,29(16):4147-4152.

[4] 白玉霞,刘旭辉,孙肖子.基于 Qt/Embedded 的 GUI 移植及应用程序开发[J].电子产品世界,2005,32(13):98-100.

[5] 马桂珍,朱玲赞,段丽.基于 OpenCV 的视频应用程序的开发方法[J].现代电子技术,2007,243(4):78-79.

[6] 刘爽,史国友,张远强.基于 TCP/IP 协议和多线程的通信软件的设计与实现[J].计算机工程与设计,2010,31(7):1417-1420.

[7] 刘余,孟小华.嵌入式智能家居终端通信模块的设计与实现[J].计算机工程与设计,2010,31(8):1689-1692.

[8] 张茁,孙洁.基于以太网的智能家庭网络系统设计[J].计算机工程与设计,2005,26(11):3133-3135.

(上接第 1900 页)

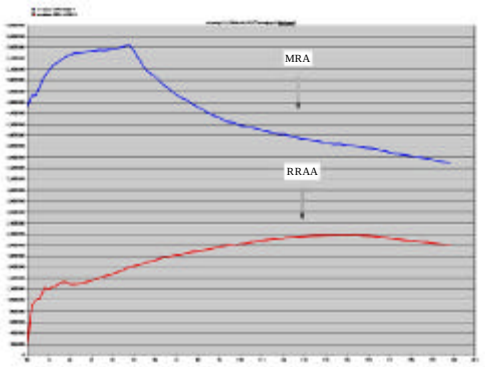

图 4　链路质量快速变化情况下的系统吞吐量

于其调整的滞后性，系统的吞吐量会大幅下降。而 MRA 则能够较好地应对链路质量的变化。如图 4 所示，链路质量快速变化情况下，MRA 仍然能够快速估计出一个较优速率。在仿真初期系统吞吐量迅速达到了一个较高的吞吐量，而 RRAA 则需要一段在时间后才能估计出合适的速率选择。由于 RRAA 在链路质量快速变化的情况下表现不佳，所以在快速变化情况下 MRA 的总体性能相对 RRAA 有将近一倍的提高。

4　结束语

目前对于 IEEE802.11 标准在物理层提供的多速率传输机制，需要有一种合适的算法来发挥其最大效能。MRA 算法利用 HPRA 子算法有效地适应了在快速变化的链路中的速率切换问题，通过 LPRA 子算法解决了链路传输中常出现的碰撞问题及 RRJ 问题。文章还对 LPRA 中的门限设定进行了分析。在仿真试验中，我们比较了 MRA 算法与当下知名的算法，其结果说明 MRA 算法能解决调制速率的快速切换、碰撞问题及 RRJ 问题，在性能上有其优越性。但是对于 MRA 算法中的一些参数的最优值设置，还需要进行进一步的数学分析及实验。

参考文献：

[1] 祁志娟,刘伟,张丽丽.无线 Ad Hoc 网络 MAC 层速率自适应技术研究[J].通信技术,2010,43(2):4-6.

[2] Judd G,Wang X,Steenkiste P.Efficient channel-aware rate adaptation in dynamic environments[C].Proc of the ACM MobiSys Conf,2008:118-131.

[3] 张军胜,孙沛,万毅.一种抗抖动的鲁棒性速率自适应算法[J].通信技术,2008,24(9):35-37.

[4] Lv S,Wang X,Zhou X.Self-learning rate adaptation in multi-rate 802.11 networks[C]. ShangHai, China: Proc IEEE International Conference on Wireless Communications,Networking and Mobile Computing,2007:2156-2159.

[5] Wong S,Yang H,Lu S,et al.Robust rate adaption for 802.11 wireless networks[C].Los Angeles,USA:Proc of ACM International Conference on Mobile Computing and Networking, 2006: 146-157.

[6] MADWIFI.Multiband Atheros driver for WiFi[EB/OL].http://madwifi. Source-forge.net,2005-12-04/2008-10-12.

[7] Bicket J.Bit-rate selection in wireless networks[D].MIT:Department of EECS,2005.

[8] 习勇,黄清艳,魏急波,等.基于 IEEE 802. II 高速无线局域网的速率自适应 MAC 协议研究[J]. 电子与信息学报, 2007,29(6): 115-120.

[9] 段中兴,张德运.多速率无线局域网的速率自适应算法[J].计算机工程,2007,33(8):33-35.