

文章编号:1001-9081(2010)06-1434-05

一种跨平台的自组网路由协议实现框架

张伟, 向勇, 李三立

(清华大学 计算机科学与技术系, 北京 100084)

(z-wei02@mails.tsinghua.edu.cn)

摘要:为了减少不同平台上自组网(MANET)路由协议的重复实现和保证协议在不同平台下实现的正确性与一致性,设计并实现了一种可以在 Windows、Linux,以及 NS-2 模拟器上运行的自组网路由协议框架,并用典型的自组网反应式路由协议 AODV 对其可行性进行了验证。该框架设计中将协议实体与周边环境抽象开来的方法具有通用性,可以适用于其他自组网路由协议甚至其他网络协议(例如 TCP),也可以扩展支持其他操作系统。

关键词:跨平台;无线 Ad Hoc 网络中反应式路由协议;自组网路由协议;适配器;协议实现

中图分类号: TP393 **文献标志码:** A

Implementation framework of cross-platform MANET routing protocol

ZHANG Wei, XIANG Yong, LI San-li

(Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China)

Abstract: In order to reduce repetition and ensure the accuracy and consistency in implementing Mobile Ad Hoc Network (MANET) routing protocol on different platforms, an implementation framework of MANET routing protocols running on Windows, Linux and NS-2 was designed and validated by implementing Ad Hoc On-Demand Distance Vector (AODV), a classic MANET reactive routing protocol based on it. The method by abstraction for separating protocol entity from environments is of generability in designing the framework. It can also be applied to other MANET routing protocols or other network protocols (e. g. TCP), and can also be extended to support other operating systems.

Key words: cross-platform; Ad Hoc On-Demand Distance Vector (AODV) routing protocol; MANET routing protocol; adapter; protocol implementation

0 引言

在网络协议开发的过程中,为了验证有效性、规模扩展性等原因常常需要在模拟环境下进行实验。作为与操作系统联系紧密的协议实现,很难用同一协议实现在模拟环境和实际操作系统下直接运行,通常在完成模拟环境下的实现和仿真后还需要在实际操作系统环境下重新开发。这对调试维护都带来了很大的不便,且多个平台下代码不一致也会使模拟结果的有效性大打折扣。即使同时运行在实际的操作系统中,由于不同的操作系统间存在较大差异,上述不一致的问题也会出现在这种跨平台开发的环境中。有线网络协议的开发基本都按照 OSI 分层结构或其简化结构进行,层间的接口比较清晰,跨层的协议也较少,因此有线网络协议开发时可以很容易构造一个软件中间件^[1],以适应不同操作系统;但是在自组网路由协议中,由于层间关系比较复杂,存在很多跨层操作,特别是反应式路由机制的存在,使得文献[1]中的做法不能满足自组网路由协议跨平台开发的要求。

目前大部分自组网路由协议实现都与平台密切相关,只可以运行在某一种操作系统下。比如 Bremen 大学开发了一个可以运行在 Windows 2000 下的 AODV (Ad Hoc On-Demand Distance Vector)^[2]路由协议,它不支持其他任何平台^[3];NIST (National Institute of Standards and Technology) 开发了一个可以运行在 Linux 2.4 下的 AODV 版本,同时还有一个单独实现

的可以在网络模拟器 OPNET 下运行的模拟版本^[4]。也有一些研究将协议实现在某一种操作系统与一种模拟环境下,比如 Uppsala 大学的 Erik Nordström 等人开发的 AODV-UU 就可以同时运行在 Linux 与网络模拟器 NS-2 下^[5],但它不能运行在 Windows 下;能够同时在多个操作系统平台以及模拟环境下运行的协议实现还很少。特别是没有可以同时运行在 Linux 与 Windows 系统下的 AODV 实现,Windows 作为一种使用率很高的桌面操作系统,并且受到代码不开源的影响,Windows 平台下的协议实现与 Linux 等开源操作系统有很大区别,其上的网络协议开发仍然是一个亟须解决的问题。

将这 3 种平台的协议开发统一在同一个框架中,对协议研究十分有意义,首先可以减少大量重复实现,有利于保证协议在不同平台下的正确性和一致性;其次协议实现可以更多地考虑到不同平台的限制,更接近于实际应用。由于 Windows 与 Linux 的内核机制差异较大,将这两者与模拟平台上的协议实现统一起来是一项有挑战性的工作。本文的研究目标就是在协议实现之初,尽可能减少不同平台(包括模拟器平台)上的重复开发和测试,保证协议实现正确性。

1 自组网路由协议跨平台实现框架及特点

1.1 跨平台框架结构

针对不同实际操作系统的差异,特别是针对实际操作系统与模拟环境的不同,自组网路由协议跨平台框架可分为协

收稿日期:2009-12-14;修回日期:2010-03-04。 基金项目:国家 863 计划项目(2008AA01Z220)。

作者简介:张伟(1985-),男,山东菏泽人,博士研究生,主要研究方向:无线网络; 向勇(1967-),男,重庆人,副教授,博士,CCF 会员,主要研究方向:无线自组网、计算机协同工作; 李三立(1935-),男,上海人,中国工程院院士,博士生导师,博士,主要研究方向:网络计算、计算机系统结构。

议实体与周边环境适配器两部分,如图1所示。其中,协议实体是自组网路由协议的主体逻辑,处理路由发现过程,对其他节点发来的路由消息进行处理,等等;它会通过周边环境适配器部分提供的接口来完成相应功能。周边环境适配器可以理解为在不同操作系统与协议实体间的一层中间件,使不同的操作系统相对于协议实体来说是透明的,与文献[1]中不同的是,它不仅封装了各种不同操作系统的下层接口(非协议相关接口),同时也封装了在自组网路由协议中需要用到的反应式路由发现等触发机制(协议相关接口)。

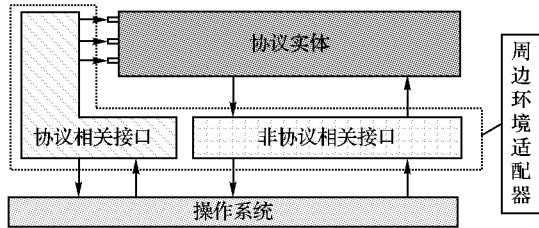


图1 跨平台协议实现框架结构

根据自组网路由协议的特点,周边环境适配器部分又可以分成如下几个功能模块:1)网络驱动模块;2)系统路由表通信模块;3)事件计时器模块。

网络驱动模块的主要作用包括:提供统一的数据包接收与发送机制;提供发送数据包无路由时触发路由发现的机制;提供路由发现过程中的包缓冲机制。系统路由表通信模块的主要作用是维护系统路由表与协议实体中包含的路由表一致,它提供了包括增加、删除和更新系统路由表的功能。事件计时器模块的主要作用是实现自组网路由协议中定义的一些

计时器,比如路由生命周期控制等。

在周边环境适配器的实现中,应尽量避免使用某个环境中独有的特性,以降低协议实体正确性的开发风险。在保证正确性的前提下,如果对某种平台上的性能有要求,还可以对其适配器做相应的针对性优化。

通过这样一系列的外部接口封装,协议实体就能够简洁地被抽象为几个主要的功能模块:路由发现(修复)过程;来自于其他节点的消息的处理过程;协议消息发送,协议内部路由表管理以及其他数据管理。

1.2 周边环境适配器的接口分类

通过如1.1节中所述的框架设计,就可以将协议实体与操作系统相分离,使得协议实体可以单独实现,而不必考虑操作系统的差异。上述周边环境适配器部分是系统相关的,对于不同的操作系统需要开发不同的周边环境适配器以隔离协议实体与系统。比如 AODV 路由协议在 Windows、Linux 与 NS-2 这3种平台下的框架结构如图2所示。

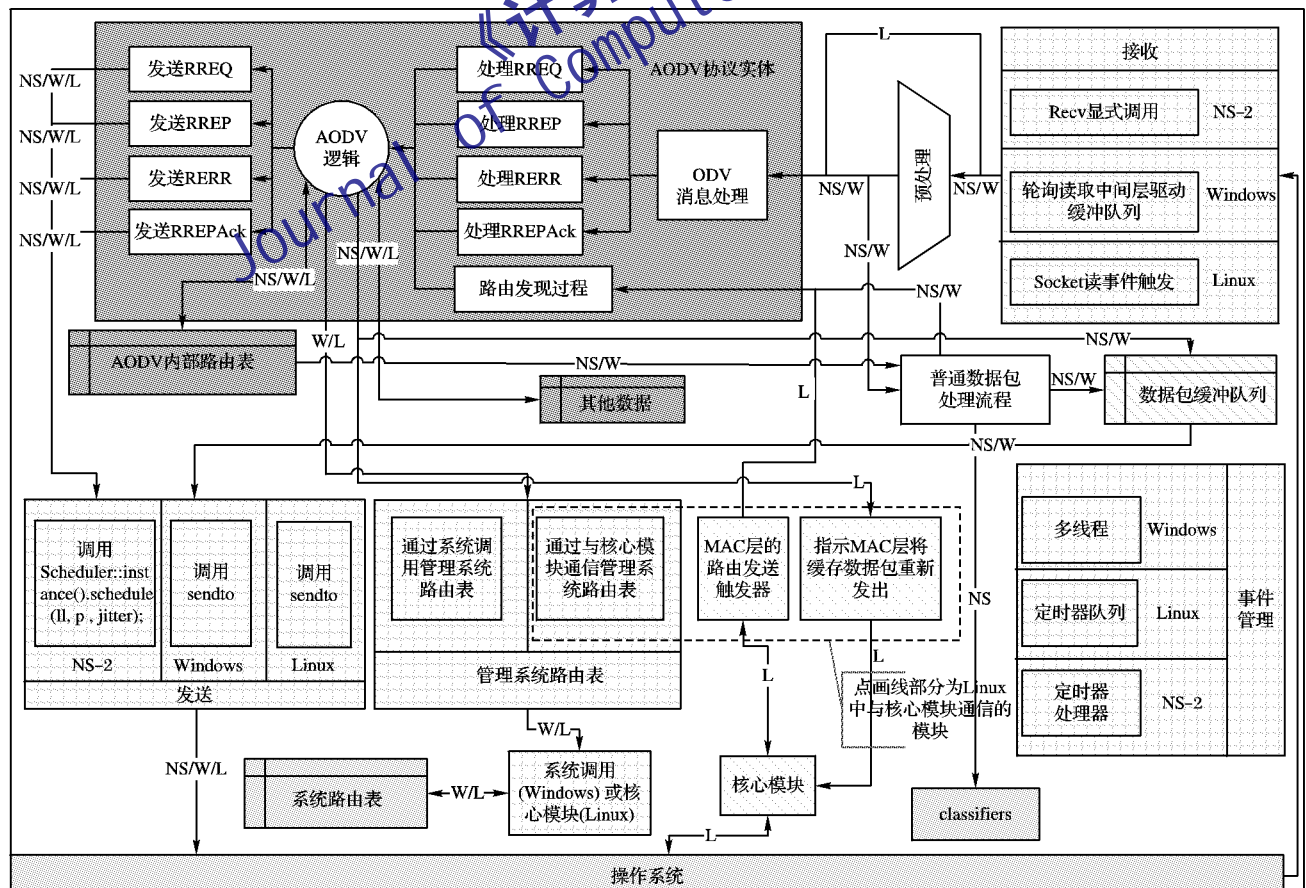
在此框架中,周边环境适配层的接口定义如下:

$$AdIF := IF_{nonrelative} \cup IF_{relative} \quad (1)$$

$$IF_{nonrelative} := IF_{socket} \cup IF_{packet} \cup IF_{rttable} \cup IF_{event} \cup IF_{otherutils} \quad (2)$$

$$IF_{relative} := IF_{triggers} \cup IF_{encapsulation} \cup IF_{others} \quad (3)$$

其中, $AdIF$ 表示适配层接口, $IF_{nonrelative}$ 表示非协议相关接口, $IF_{relative}$ 表示协议相关接口, IF_{socket} 表示管理 socket 的接口, IF_{packet} 表示接收和发送数据包的接口, $IF_{rttable}$ 表示管理系统路由表的接口, IF_{event} 表示事件管理以及定时器等接口, $IF_{otherutils}$



注: 协议相关接口 非协议相关接口 协议实体 操作系统
NS—NS-2 W—Windows L—Linux

图2 3种平台下 AODV 协议实现框架

表示其他的非协议相关接口, $IF_{triggers}$ 表示跨层触发器及相关接口, $IF_{crossmanage}$ 表示跨层数据管理接口, IF_{others} 表示其他的协议相关接口。

非协议相关接口仅是对系统提供的接口进行统一的封装,在3种平台下的实现各不相同:与系统路由表的交互在 Windows 中存在系统调用可以方便地使用,在 Linux 中也提供了一个用来管理系统路由表的核心模块,可以通过一个 netlink 连接与之通信实现系统路由表的更新,而在 NS-2 中不存在系统路由表,路由一般由协议代理直接完成;事件管理基本与协议实体本身的逻辑无关,具体可以参考 2.2 节;对协议包处理的具体的细节可参考 2.3 节中的说明。

协议相关接口则与协议实体的逻辑密切相关,比如在路由发现触发机制中,功能实体发现一个向外发送的数据包在路由表中没有相应的路由表项时,就要调用协议实体的路由发现处理过程;链路断开触发器发现一条与邻居节点的链路中断时,将调用协议实体的路由修复处理过程;在路由发现的过程中,协议实体需要将待发送的数据包缓存下来,在路由发现完成时再将缓存的数据包重新发出。在 Windows 与 NS-2 中,由于两者都必须处理节点流入流出的所有包,所以路由发现与包缓冲都是在更靠近协议实体的用户态完成的;而 Linux 则可以通过核心模块将这些触发器与缓冲机制实现在内核态运行的模块中,对协议实体来说,它们只需要提供用于回调的接口。

2 AODV 协议跨平台实现中的关键点

本文完成的在 Windows、Linux 与 NS-2 系统下的 AODV 协议实现,部分参考了 Uppsala 大学的 AODV-CU^[5] 以及 Bremen 大学的 UoBWinAODV^[3]。下面详细介绍 AODV 协议中路由发现的触发机制,不同平台下的事件管理,以及与网络层的交互等方面在实现过程中的若干关键点。

2.1 路由发现的触发机制

作为典型的反应式自组网路由协议,AODV 的路由发现触发机制是关系到协议性能的一个重要因素^[6]。

在 Windows 与 NS-2 中,可以采用监控流过网卡的所有数据包的方法,一旦发现发出的包的地址在内部路由表中没有相应表项,则将此数据包加入路由发现的缓冲队列,并触发路由发现过程,在收到路由回复后再将缓冲的数据包重新发出。在 Linux 系统中,由于在接收的时候仅仅将端口号为 654 (AODV 消息专用)的数据包转发给协议实体(如图 2),所以不能采用上述策略,而是构造一个核心模块(Kernel Module),由它监控 netfilter 中的 PRE_ROUTING 钩子,判断是否存在对应的路由表项,没有则将数据包缓存并利用 netlink 套接字向协议实体发送触发路由发现的消息;在协议实体收到其他节点发送的路由回复消息后,再向该核心模块发送路由命中的消息以重发缓冲中的数据包。

在 AODV 协议中,路由的局部修复与路由发现的过程是基本类似的,但是它的触发机制与路由发现略有不同。在 NS-2 中,无线链路断开是容易检测出来的,它可以利用链路层的 packetFailed 回调函数来进行错误处理,即发送路由错误消息以及触发路由局部修复。但是在 Windows 系统中,由于硬件与驱动程序的限制(仅有很少的网卡支持监控 802.11 的 CTS/RTS),无法了解到链路层的状况,很难在第一时间了解

到链路断开与否,只能通过各种定时器的维护来粗略了解链路状况,在现有的设计中,也主要是利用路由表项的活动周期来判定两个相邻节点间的链路断开。在 Linux 平台现有的 AODV 实现中,也都采用的是类似于 Windows 的做法,利用 Hello 包的定期发送来检测链路断开^[7],因此仍然沿用这种设计。从链路断开触发这点来看,模拟器平台对实际环境做了很多抽象,这为研究者提供了方便,但同时也导致了模拟环境与实际环境的巨大差异,使用上述跨平台框架,更有利于降低开发复杂度,发现因这种差异带来的问题。

2.2 事件管理

在 AODV 协议中,需要管理的事件基本上都可以用定时器来描述,即在既定时间后发生一个动作。比如当一个路由表项在一个生命周期内都没有活动过,那么在生命周期过后该路由表项将被置为非活动状态,且再启动一个定时器,在这个定时器到期后再将此路由表项彻底删除。

可以使用系统提供的定时器,也可以采用轮询队列的方式来实现这些定时器。在 Linux 中系统没有提供明显的定时器调用,因此框架中采用了轮询队列的方式;在 NS-2 系统与 Windows 系统中都提供了简明的定时器调用,但为了能与 Linux 系统的实现尽量一致,仍采用轮询的方法。对于轮询的实现方法,在各个系统上也存在着差异。Windows 与 Linux 都支持多线程编程,但是在 NS-2 中仅支持单线程,不过它提供了事件处理的机制,可以利用其提供的定时器处理器(TimerHandler)再次将事件队列封装以完成轮询。为了尽量保持实际系统与模拟系统实现的一致,在 Linux 系统中接收数据包是通过监听 AODV 端口实现的,所以可以将轮询与监听的循环写在一起单线程处理,而不采用多线程处理;但在 Windows 中由于不能仅监听 AODV 端口,否则无法缓存路由发现过程中的数据包,接收数据包必须使用的单独的线程通过与一个内核驱动通信获得,所以对定时器的处理也采用对每一类事件都新启用一个单独的线程来轮询触发。对于 3 种平台下的定时器队列的实现是基本一致的,仅仅是轮询的实现方式存在着上述差异,这也是为了尽可能提高协议跨平台实现的正确性,减少错误调试风险而对运行效率的一种折中。

2.3 与系统网络层的交互

NS-2 网络模拟器是一个模拟网络协议的运行离散事件模拟器,它与实际系统有着本质的区别,从图 3 中就可以看出,在 NS-2 与 Windows、Linux 等系统对数据包的处理流程有较大差异。

在 NS-2 中,所有进出本节点的数据包都会从一个进入点(entry point)进入处理流程,在有些节点中会使用地址分类器进行预处理,但一般情况下特别是在路由协议的实现中都是使用一个路由代理(Route Agent)来处理数据包。这样路由代理就可以取得通过该节点的所有数据包,那么它就可以根据它维护的内部路由表来判断数据包的流向:送向上层应用代理还是向下发送到链路,抑或没有相应的路由从而触发路由发现过程。

在 Windows 中并没有明显的接口可以使应用程序获得通过本节点的所有数据包,也不像 Linux 那样提供了方便的网络过滤钩子编程接口,所以实现时采用构造一个 NDIS 微端口驱动的方法,将通过本节点网卡的所有数据包截取下来

送往应用程序的方法来获得所有数据包。在截取的同时,被截取的数据包仍然像平常一样被系统处理,或向上传递至应用层,或向下到链路层,除非系统路由表中没有相应的路由表项。而路由代理在监控到一个数据包的目的地址并不存在于路由表中时,就触发了路由发现过程。

Linux 提供了丰富的网络编程接口,所以应用程序不必截取所有的数据包,而只需处理发送给 AODV 专用端口的那些 AODV 消息。触发路由发现与缓存路由发现时的数据包的工作则由一个核心模块完成,该核心模块的主要功能是实现了一个网络过滤中的两个钩子: `NF_IP_PRE_ROUTING` 和 `NF_IP_LOCAL_OUT`。前者作用主要是在数据包路由前判断是否存在指向该数据包目的地址的路由表项,不存在就丢弃该数据包,除非针对该目的地址的路由修复过程正在进行则缓存该数据包并返回。后者作用则是在不存在相应的路由表项时触

发路由发现过程;如果路由发现过程正在进行,则缓存该数据包并返回。除此之外,由于 Linux 系统路由表的接口也是由一个核心模块完成的,所以该核心模块还负责系统路由表的更新。

在两个实际的系统中, AODV 的协议实体基本都实现在用户态,因为这有利于调试。但是 Windows 较 Linux 而言,其很多周边环境适配器也都运行在用户态,比如无路由时的路由发现触发器等,这是由 Windows 内核固有的封闭性所决定的。

在这种框架下, AODV 协议实体结合相应的适配器就很容易运行在 3 种平台之上,我们已经在实际系统(Windows XP Service Pack 2, Linux 2.6.24, NS-2)中验证了其可行性。并且该系统对于其他的自组网路由协议甚至其他网络协议都具有可推广性。

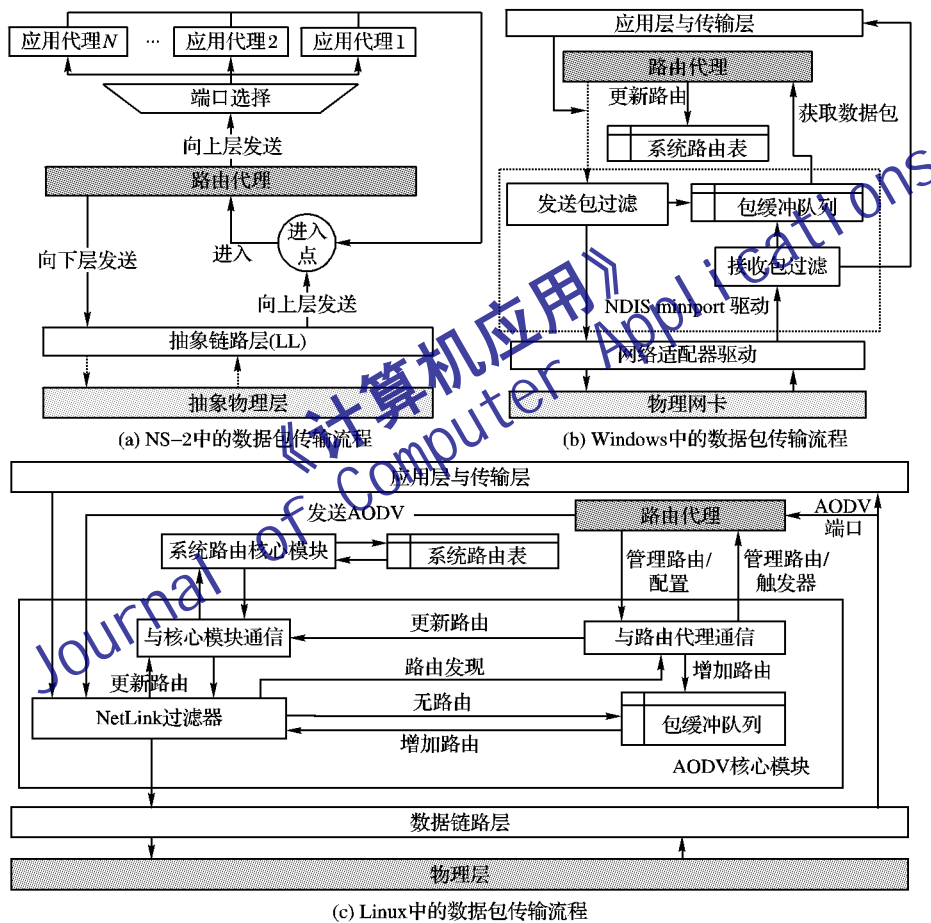


图3 3种平台下数据包传输流程

3 系统的可推广性

AODV 路由协议是典型的自组网反应式路由协议,前面介绍了 AODV 协议在 Windows、Linux 与 NS-2 这 3 种平台下的跨平台实现方案,对于其他自组网路由协议甚至其他类型网络协议的跨平台开发也具有可推广的意义。

如式(1)中把周边环境适配器分为两类接口:一类为非协议相关接口;另一类为协议相关接口。对于非协议相关接口来说,对于所有的自组网路由协议乃至其他的网络协议都是通用的,因为它们的实现与协议本身的逻辑并没有关系,比如接收发送数据包等功能。对于协议相关接口来说,本框架提供了路由发现触发、链路断开触发,以及路由发现时的包缓

冲等机制,基本可以涵盖一般的自组网反应式路由协议,比如 DSR 等对周边环境的交互要求。

对于自组网路由协议中与反应式路由相对的另一大类路由协议——主动式路由协议,它们对周边环境的要求较反应式更少,不需要反应式路由触发机制,比如 DSDV 主要依靠广播路由信息以交换路由,它们仍然可以使用本框架中协议相关接口中的链路断开触发器以在适当的时候回调其链路断开处理功能。

下面以 DSDV 为例讨论框架对其的通用性。记 DSDV 路由协议实体需要的周边环境适配器接口为 $AdIF_{DSDV}$, 协议相关接口为 $IF_{relative-DSDV}$, 非协议相关接口为 $IF_{nonrelative-DSDV}$ 。

因为 $IF_{socket-DSDV} = IF_{socket}$

$$\begin{aligned} IF_{\text{packet-DSDV}} &= IF_{\text{packet}} \\ IF_{\text{rttable-DSDV}} &= IF_{\text{rttable}} \\ IF_{\text{event-DSDV}} &= IF_{\text{event}} \\ IF_{\text{otherutils-DSDV}} &\subseteq IF_{\text{otherutils}} \end{aligned}$$

又根据式(2),所以

$$IF_{\text{nonrelative-DSDV}} \subseteq IF_{\text{nonrelative}} \quad (4)$$

因为 $IF_{\text{triggers-DSDV}} \subset IF_{\text{triggers}}$

$$IF_{\text{crossmanage-DSDV}} \subset IF_{\text{crossmanage}}$$

$$IF_{\text{others-DSDV}} \subseteq IF_{\text{others}}$$

又根据式(3),所以

$$IF_{\text{relative-DSDV}} \subset IF_{\text{relative}} \quad (5)$$

根据式(1)、(4)~(5),可得:

$$AdIF_{\text{DSDV}} \subset AdIF \quad (6)$$

所以该框架可以适用于路由协议 DSDV。

但是对于某些特殊的自组网路由协议,比如 GPS 辅助型路由协议^[8]需要使用到 GPS 信息,目前框架中并没有提供对这一类信息的封装,因此针对这种有特殊要求的协议,只要在周边环境适配器中加入对这一特性的支持即可使用该框架,即在 $IF_{\text{otherutils}}$ 及 IF_{others} 中添加相应的子接口。

另外,此框架目前仅支持 Windows、Linux 与网络模拟器 NS-2,如果需要向其他类型的操作系统或模拟器移植协议实现还需要为目标操作系统补充新的周边环境适配器。虽然自组网路由协议的功能逻辑各不相同,目标操作系统提供的功能也千差万别,在适应周边环境的时候上述框架可能需要添加一些新的接口,但是这种抽象的方法是通用的。

4 结语

上述自组织网络路由协议在 Windows、Linux 与 NS-2 系

统上的跨平台开发思想,已经在实际开发过程中得到了验证,同一个 AODV 路由协议实体搭配相应系统的适配器,在这 3 种平台下都可以良好运行。同时这个框架也可以作为其他自组网路由协议的跨平台开发框架,且其抽象的方法对其他网络协议的开发也具有指导作用。我们今后还将在此框架的基础上,实现 MAODV (Multicast Ad Hoc On-Demand Distance Vector) 组播路由协议以及一些网关协议,以进一步验证该框架的通用性。此外,针对特种平台的性能优化问题是在完成协议正确性研究后的后续工作,在本文框架中暂没有涉及。

参考文献:

- [1] MIGUEL C, CALAFATE T, MANZONI P. A multi-platform programming interface for protocol development [EB/OL]. [2009-10-10]. http://www.gre.upv.es/calafate/download/PICA_PDP2003.pdf.
- [2] RFC3561, Ad Hoc On-Demand Distance Vector (AODV) Routing [S]. 2003.
- [3] MobileIP [EB/OL]. [2009-10-10]. <http://www.aodv.org/>.
- [4] Kernel AODV [EB/OL]. [2009-10-10]. http://w3.antd.nist.gov/wctg/aodv_kernel/.
- [5] Ad Hoc on-demand distance vector routing [EB/OL]. [2009-10-10]. <http://core.it.uu.se/core/index.php/AODV-UU>.
- [6] DING S. A survey on integrating MANETs with the Internet: Challenges and designs [J]. Computer Communications, 2008, 31(14): 3547-3551.
- [7] CHAKERIS, ELIZABETH M. AODV routing protocol implementation design [C]// ICDCSW'04: Proceedings of the 24th International Conference on Distributed Computing Systems Workshops — W7: EC. Washington, DC: IEEE Computer Society, 2004: 698-703.
- [8] 苏静,郭伟. 无线移动自组织网络路由协议综述[J]. 中国测试技术, 2005, 31(2): 91-93.

(上接第 1433 页)

疏时,规约规则能有效地降低图的规模,减少后继求解算法的运算量;同时,优化规则也能起到很好的效果。而当图中的弧变得较为稠密时,近似算法的效果越来越好。因此,包含规约规则、近似算法和优化规则的求解算法对不同类型的有向图是有效的。

3 结语

有向图最多叶子生成树问题近年来引起了广泛的关注,较之无向图的最多叶子生成树问题更难解。该问题是否存在线性核,以及是否存在常数近似度的近似算法还是一个开放的问题。对此,借助参数算法中的一些技术设计了一个规约规则用以降低原问题的规模,然后设计了近似算法来求解,最后利用规约后的图的一些特性设计了优化规则,本文通过变换获得更多的叶子节点。不同类型随机图上的模拟实验表明规约规则、近似算法和优化规则是有效的。

参考文献:

- [1] GAREY M R, JOHNSON D S. Computers and intractability: A guide to the theory of NP-completeness [M]. New York: W. H. Freeman & Company, 1979: 206-207.
- [2] FELLOWS M R, LANGSTON M A. On well-partial-order theory and its application to combinatorial problems of VLSI design [J].

SIAM Journal of Discrete Mathematics, 1992, 5(1): 117-126.

- [3] BODLAENDER H L. On linear time minor tests with depth-first search [J]. Journal of Algorithms, 1993, 14(1): 1-23.
- [4] LU H, RAVI R. Approximating maximum leaf spanning trees in almost linear time [J]. Journal of Algorithms, 1998, 29(1): 132-141.
- [5] SOLIS-OLBA R. 2-approximation for finding trees with many leaves [C]// Proceedings of ESA. Berlin: Springer-Verlag, 1998: 441-452.
- [6] ALON N, FOMIN F V, GUTIN G, et al. Parameterized algorithms for directed maximum leaf problems [C]// Proceedings of ICALP. Berlin: Springer-Verlag, 2007: 352-362.
- [7] ALON N, FOMIN F V, GUTIN G, et al. Spanning directed trees with many leaves [J]. SIAM Journal of Discrete Mathematics, 2009, 23(1): 466-476.
- [8] KLEITMAN D J, WEST D B. Spanning trees with many leaves [J]. SIAM Journal of Discrete Mathematics, 1991, 4(1): 99-106.
- [9] FERNAU H, FOMIN F V, LOCKSHTANOV D. Kernel(s) for problem with no kernel: On out-trees with many leaves [C]// Proceedings of STACS. Berlin: Springer-Verlag, 2009: 421-432.
- [10] DALIGAULT J, THOMASSE S. On finding directed trees with many leaves [C]// Proceedings of IWPEC. Berlin: Springer-Verlag, 2009: 86-97.