

基于 NS-2 网络仿真协议的功能扩展

吴永华, 刘广钟

(上海海事大学 信息工程学院, 上海 200135)

摘要:网络仿真是网络协议设计的重要环节,也是通信网络性能分析的关键。基于 C++ 和 Tcl 脚本语言分裂层次实现的 NS-2 是一个开源、面向对象设计的多协议网络仿真软件,在网络仿真领域占有十分重要的地位。文中首先介绍 NS-2 的体系结构;然后介绍在 NS-2 环境下如何编程实现新协议,对 NS-2 功能进行扩展;最后利用前面实现的新协议,结合一个 4 节点的网络仿真实例,根据仿真结果比较 SFQ 和 DropTail 两种队列管理方式在选择数据包丢弃方面的公平性。

关键词:NS-2; C++; Tcl; SFQ; DropTail; 网络仿真; 协议

中图分类号:TP31

文献标识码:A

文章编号:1673-629X(2009)12-0063-04

Function Extension of Network Simulation Protocol Based on NS-2

WU Yong-hua, LIU Guang-zhong

(School of Information Science and Engineering, Shanghai Maritime University, Shanghai 200135, China)

Abstract: It is very important to simulate network in design and performance analysis of network. NS-2, with advanced C++ split-language and Tcl scripting languages programming idea, is an open-source and object-oriented design multiprotocol network simulation software. NS-2 has a very important position in the area of network simulation. Introduce the architecture of NS-2 and how to implement new network protocol in detail so as to extend the function of NS-2. Finally, using earlier implemented protocol, give an example which simulates four nodes of network, and then compare the fairness of SFQ and DropTail queuing when they are choosing the dropping data-packet.

Key words: NS-2; C++; Tcl; SFQ; DropTail; network simulation; protocol

0 引言

随着仿真技术快速发展,应用领域越来越广,作为仿真领域一个分支的网络仿真技术也获得了快速发展。

NS-2 是一个离散事件模拟器,具有开放性好、扩展性强的特点,是一个出色的研究网络拓扑结构、分析网络传输的仿真工具。NS-2 是一个开源、面向对象设计的多协议网络仿真软件;文中具体研究如何在 NS-2 中实现新的网络协议,并将新协议有效地融合到 NS-2 中,从而实现对 NS-2 功能的扩展;最后利用前面实现的网络协议进行具体仿真实验。

1 NS-2 体系结构

NS-2 是由两种语言 Otcl, C++ 编写而成的开源网络仿真软件,提供了模拟网络各层次所必需的基本

网络实体元素、环境仿真元素,但各层次元素之间的联系很松散,并没有像实际网络中紧密的联系。NS-2 中所有的网络元素都是抽象的,找不到任何实际的网络设备。NS-2 采用所谓“事件”驱动,因而是一个离散时间模拟器^[1,2]。NS-2 的模拟分为两部分:用 C++ 编写特定网络元素的实现;用 Otcl 编写模拟所需的模拟脚本文件,在文件中使用这些网络元素;二者之间的结合由 NS-2 负责完成,NS-2 规定一个固定的步骤,按照规定的步骤去做即可。

1.1 离散事件模拟

离散事件模拟,是几种常用的系统模拟模型之一。简单的说,事件规定了系统状态的改变,状态的修改在事件发生时进行。在一个网络模拟器中,典型的事件包括分组到达、时钟超时等。NS 是一个事件(Event)驱动的模拟器,其核心部分是一个离散事件模拟引擎。NS 中有一个“调度器(Scheduler)”类,是 NS 的核心,负责记录当前事件,调度网络事件队列中的事件,并提供函数产生新事件,指定事件发生的时间。需要注意的是,NS 只支持单线程,故在某一时刻只能有一个事件在执行,如果多于一个事件被安排在同一时刻,那么会

收稿日期:2009-04-09;修回日期:2009-07-18

基金项目:上海市教育科研计划项目(2008072)

作者简介:吴永华(1985-),男,浙江衢州人,硕士研究生,研究方向为计算机网络;刘广钟,教授,研究方向为计算机网络。

按照事件代码插入的先后次序执行。

1.2 NS-2 的层次结构

NS-2 后台用 C++ 实现网络协议,以 Otcl 解释器为前台,是面向对象的模拟器。该模拟器支持 C++ 中的类层次结构^[3](文中称为编译层次)和 Otcl 解释器中的相似层次结构(称为解释层次)。这两种层次结构紧密相关;从用户的角度看,解释层次中的类与编译层次中的类是一一对应的。这种层次的基类是 TclObject 类。用户进行仿真实验时通过解释器创建新的模拟对象;这些对象先在解释器中被实例化,然后由编译层次中相应的对象产生映射。解释类层次在 NS-2 加载时通过 TclClass 类中定义的方法自动创建,用户则是通过 TclObject 类中定义的方法映射这些实例化的对象,并进行变量绑定。C++ 代码和 Otcl 脚本里面还有其它的层次,但是这些层次并不会以 TclObject 的方式被映射。

NS 仿真器中用到了六种 Tcl 类,它们是:

(1)Tcl 类:封装了 Otcl 解释器实例,向外提供方法来访问解释器。

(2)TclObject 类:它是两种类结构中的大多数类的基类。图 1 是 NS 中部分继承于 TclObject 类的层次关系。

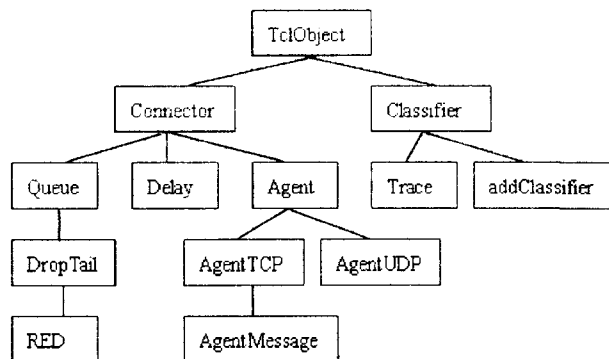


图 1 NS 类层次结构

(3)TclClass 类:把用户通过类 TclObject 在解释器中建立的类结构映射到 NS 编译类结构,提供方法来实际化新的对象。

```

class TclClass {
protected:
    TclClass(const char * classname);
    virtual TclObject * create(int argc, const char * const * argv) = 0;
};
  
```

上面摘录自 NS2.33 中的部分源码,TclClass 类的构造函数实现了解释层次类的注册,而 create 函数创建编译类层次的对象,所以说 TclClass 建立了解释层次对象与编译层次对象的映射关系。

(4)Tclcommand 类:提供一种机制使 NS 内核向解释器输出简单的命令。

(5)EmbedTcl 类:允许用户以编译代码或解释器代码来扩展 NS 功能。

(6)Instvar 类:定义了一些方法和机制,在编译类结构对象的成员变量和对应的解释器类结构对象的成员变量之间建立映射,使两类变量一致共享。

1.3 网络组件

NsObject(即 TclObject)是所有网络组件的基类,包括节点、链路、代理、业务记录和数据源等。节点、链路、代理同时继承了 NsObject 和事件处理器类 Handler,因为这三种对象要处理多种事件,其他对象则不需要。这些组件有的是简单组件,它与单一的 C++ 对象相联系,有的是由简单组件组合而成的复合组件;例如链路由延迟组件(模拟传播延时)和队列组件组成。一般来说,在 NS-2 初始化阶段,由 Tcl 解释器调用 Tcl 脚本(. /ns - allinone - 2. * /tcl/)来创建,配置这些组件。

NS 的组件库是一个层次结构,其中的组件通常都是由相互关联的两个类来实现的,一个在 C++ 中,一个在 Otcl 中。因此,NS 中就包含了一个 C++ 类的层次结构和一个 Otcl 类的层次结构。构件的主要功能通常在 C++ 程序中实现,Otcl 中的类则主要提供 C++ 对象面向用户的配置接口。在 NS 中,C++ 中的类和 Otcl 中的类通常具有对应的关系,两边的类的继承关系也通常是一致的,NS 通过 TclCL 把两种语言中的对象和变量联系起来,每当实例化一个组件时,都会同时创建一个 Otcl 中的对象和一个对应的 C++ 对象,并且这两个对象可以互操作。基于两方面的考虑,NS-2 在设计实现上使用了两种程序设计语言:C++ 和 Otcl。C++ 程序的运行速度比较快,是强制类型语言(进行严格的数据类型检查),容易实现复杂的数据类型,容易实现精确的、复杂的算法,因此适合用于实现具体的网络协议。Otcl 运行速度比较慢,但可以很方便地(并且是交互的)修改,不需要编译,而且 Otcl 不是强制类型的,不容易出错,适于用来进行网络仿真的配置。这种将 Tcl 脚本语言与 C++ 语言结合开发的软件可以参考 Ousterhout 的文章^[4]。

2 NS 中实现新协议

图 2 给出网络协议仿真的一般步骤。

首先在 NS 中添加或修改网络协议,并添加对应的分组及分组包头类型。这一部分的内容还包括相应的 Tcl 定义,以便使得修改后的内容被 NS 所接受并且在 Tcl 代码中使用。在完成了协议的定义^[5]和实现

后,需要对新增加的文件进行编译并链接到 NS 中去。下面结合实际例子 Agent/UDP,对相应过程进行分析。

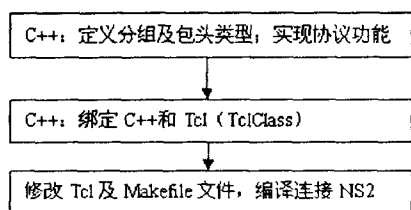


图 2 网络协议仿真的一般步骤

2.1 分组、包头定义及协议实现

UdpAgent 是一个网络组件, UdpAgent 继承于 Agent 类, 而 Agent 的继承关系是 TclObject/Connector/Agent, 前面提到过所有的网络组件都必须继承 TclObject 类。TclObject 类中 bind 成员函数实现 C++ 编译对象中成员变量与解释对象中变量的绑定; 具体可见 tclcl.h 和 tcl.cc, 以下代码是 UdpAgent 的声明代码。

UdpAgent 协议的具体实现主要在 upd.cc, udp.h 这两个文件中。理解下面代码还需要参考文件 agent.h, agent.cc 等。

```

class UdpAgent:public Agent {
public:
    UdpAgent();
    UdpAgent(packet_t);
    virtual void sendmsg(int nbytes,const char * flags=0)
    {
        sendmsg(nbytes,NULL,flags);
    }
    virtual void sendmsg(int nbytes,AppData * data,const char *
    flags=0);
    virtual void recv(Packet * pkt,Handler * );
    virtual int command(int argc,const char * const * argv);
protected:
    int seqno_;
};
  
```

在 upd.cc 文件实现中 UdpAgent 的默认构造函数, 可以看到 UdpAgent 的 type_ 成员变量被初始化为 PT_UDP; 这个变量需要在文件 packet.h 中增加。

```

UdpAgent::UdpAgent():Agent(PT_UDP,seqno_(-1))
{
    bind("packetSize_", &size_);
}
  
```

成员函数 virtual int command(int argc,const char * const * argv) 在解释层次对象 Agent/UDP 执行 Tcl 命令时被调用, Tcl 命令将以字符串的形式将参数传递给 command 函数。UdpAgent 其他的成员函数实现具体的网络协议功能。

2.2 绑定 C++ 和 Tcl(TclClass)

如前所述, C++ 编译对象与 Tcl 解释对象一一对应, 它们之间通过 TclClass 实现这种映射关系。

```

static class UdpAgentClass:public TclClass {
public:
    UdpAgentClass()
    TclClass("Agent/UDP") {}
    TclObject * create(int,const char * const * )
    {
        return (new UdpAgent());
    }
} class_udp_agent;
  
```

以上代码实现 C++ 编译对象 UdpAgent 与 Tcl 解释对象 Agent \ Udp 的映射, NS 初始化时候会调用这个声明为 static 类的构造函数, 此时它先调用 TclClass("Agent/UDP"), TclClass 会将解释类 Agent \ Udp 注册到解释器, 同时为这个类添加 create-shadow 和 delete-shadow 两个方法。create-shadow 这个方法在 Tcl 命令初始化解释对象时会被调用, 此时它会调用 UdpAgentClass::create() 成员函数返回 TclObject 对象, 这里就是 UdpAgent 对象, 从而实现 C++ 编译对象与 Tcl 解释对象的映射关系^[6]。

2.3 配置工作

在实现以上两个过程后, 还需要修改一些 Tcl 文件及 Makefile 文件。如 tcl/lib/ns-default.tcl 文件中需要增加代码: Agent/UDP set packetSize_ 1000, 为 Agent/UDP 数据包设定大小。修改 Makefile 文件, 为变量 OBJ_CC 添加目标文件: udp.o。在完成以上工作后, 可以重新编译得到扩充过的 NS-2。

3 仿真实现

3.1 场景设定

网络拓扑如图 3 所示, 包括 node0, node1, node2, node3 四个节点, node0→node2, node1→node2, node2→node3 之间有全双工的通信链路, 带宽都为 1Mb/s, 传播延时为 10ms。node0→node2 及 node1→node2 链路采用 DropTail 队列管理, 而对 node2→node3 之间的链路的队列管理分别设置为 DropTail 及 SFQ; node0→node3 及 node1→node3 之间有一个 UDP 数据流, 仿真目的一是实现仿真的可视化; 二是观察 DropTail 及 SFQ 队列管理在处理分组丢弃时的公平性。

以下是实现的部分 Tcl 代码:

```

$ns duplex-link $n0 $n2 1Mb 10ms DropTail
$ns duplex-link $n3 $n2 1Mb 10ms DropTail #或 SFQ; 以上
创建节点之间的链路
set udp0 [new Agent/UDP]
  
```

```
$ ns attach-agent $ n0 $ udp0 # node0 上创建 UDP agent
# 同样需要在 node1 上创建 UDP agent
set null0 [new Agent/Null]
$ ns attach-agent $ n3 $ null0 # node3 上创建 Null agent, 用于
接收 node0, node1 发送来的数据
```

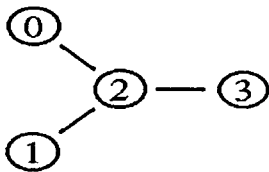


图 3 仿真拓扑

3.2 实例步骤及部分关键代码

(1) 建立一个仿真对象的实例并创建追踪文件。

```
set ns [new Simulator]
set nf [open out.nam w]
$ ns namtrace-all $ nf
```

(2) 建立一个名为 finish 的过程 (procedure), 用来关闭 trace 文件, 结束仿真过程, 并调用 nam 程序演示模拟过程。

(3) 建立如图 3 所示的拓扑, 同时分别为 udp0 及 udp1 创建 Application/Traffic/CBR 数据流。

```
set cbr0 [new Application/Traffic/CBR]
$ cbr0 set packetSize_ 500
$ cbr0 set interval_ 0.005
$ cbr0 attach-agent $ udp0
```

(4) 为不同 udp0 和 udp1 产生的数据流着不同的颜色, 以利用观察仿真结果。

```
$ udp0 set class_ 1
$ udp1 set class_ 2
```

(5) 模拟器在 0.5s 时启动 cbr0 先向 node3 发送数据, 在 1.0s 时启动 cbr1 向 node3 发送数据, 在 4.0s 停止 cbr0 的数据发送, 在 4.5s 停止 cbr1 的数据发送。5.0s 模拟器调度“finish”过程, 最后启动模拟器进行仿真。

3.3 NAM 动画演示

(1) 当 node2→node3 之间的链路的队列管理采用 DropTail 时, 仿真结果如下:

1.211453s 时, 网络运行情况见图 4, 可以看到此时没有发生丢包情况, 没有在 1.01s, node0 及 node1 同时向 node3 发送数据包时马上发生丢包现象。说明 node2→node3 之间的队列起到了缓冲数据包的作用。

1.518723s 时, 网络运行情况见图 5, 可以看到此时发生丢包情况, 而且丢包的全部是 node0 节点发送出来的数据包 (通过 Trace 功能)。说明 DropTail 队列管理在处理网络拥堵时没有公平选择丢弃的数据包。

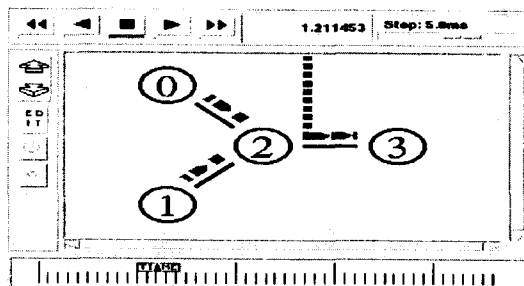


图 4 DropTail 队列运行

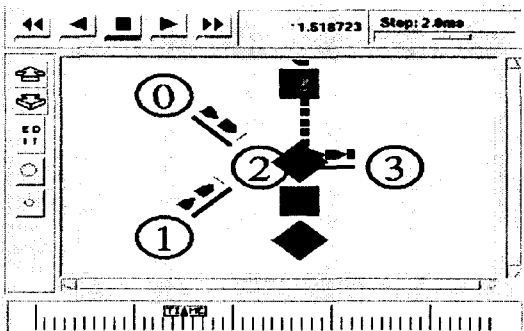


图 5 DropTail 队列运行丢包情况

(2) 当 node2→node3 之间的链路的队列管理采用 SFQ 时, 仿真结果见图 6。

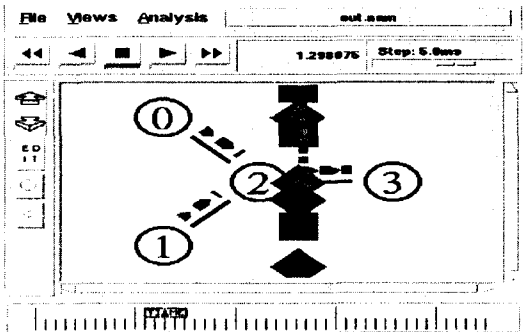


图 6 SFQ 队列运行丢包情况

1.298075s 时, 可以看到此时发生丢包情况, 而且丢弃的数据包有 node0 及 node1 节点发送出来的数据包 (通过 Trace 功能)。说明 SFQ 队列管理在处理网络拥堵时较公平选择丢弃的数据包。

3.4 结果分析

通过仿真得知, 队列管理能缓冲节点上接收到的数据包, 在节点不能正常接收发送数据包时。如还有数据包发送到当前节点, 节点会选择相应的数据包进行丢弃, 在选择数据包丢弃的过程中, 发现 SFQ 比 DropTail 更公平。

4 结束语

文中在研究分析 NS-2 网络仿真实现的原理的基础上, 对如何在 NS-2 上实现新协议进行具体详细

(下转第 70 页)

$EMPL < M[i] < EMPH$

则这一帧语音信号为有效语音信号,否则为噪声段或无声段。

第三步:重复第二步,直到语音结束。

(4) 提取语音特征参数。

根据文中第 1 节和第 2 节所述方法,分别提取 LPCC、LPCC + Δ LPCC、MFCC 及 MFCC + Δ MFCC 四种特征参数。

(5) DTW 识别算法的实现。

对 LPCC、LPCC + Δ LPCC、MFCC 及 MFCC + Δ MFCC 四种特征参数分别采用 DTW 识别算法进行语音识别。

该系统的 DTW 算法为两步:第一,计算参考模板和测试模板各帧之间的距离,即求出帧匹配距离矩阵。第二,在帧匹配距离矩阵中找出一条最佳路径。

4.2 实验结果及分析

该系统的识别结果如表 2 所示。

表 2 语音特征参数识别率的比较

特征参数	LPCC	LPCC + Δ LPCC	MFCC	MFCC + Δ MFCC
识别率	88%	90%	94.5%	95.1%

从表 2 看到 MFCC 的识别率较 LPCC 的识别率高,特征参数和它们的一阶差分组成的参数识别率较特征参数本身的识别率高。

LPCC 参数是根据声管模型建立的特征参数,主要反映声道响应,对噪声的影响特别敏感^[8],而 MFCC 参数基于人而的听觉效应,能对外来信号产生调节作用;LPCC 和 MFCC 参数反映了语音参数的静态特性,而它们的差分倒谱参数反映了语音的动态变化,二者结合起来既反映了语音的动态特性又反映了语音的静态特性,所以识别率较高。

(上接第 66 页)

介绍,最后结合上述新协议,对网络中的链路采用不同的队列管理策略(DropTail, SFQ^[7])进行仿真实验。文中虽然针对的是 NS-2 下有线网络的仿真,但是对于无线网络仿真也有一定的指导意义,更多关于无线网络仿真的信息可以参考 NS-2 无线网络仿真^[8]。

参考文献:

- [1] 李腊元. NS 的仿真机制及协议扩展[J]. 武汉理工大学学报, 2004, 28(2): 182-185.
- [2] 刘俊, 徐昌彪, 隆克平. 基于 NS 的网络仿真探讨[J]. 计算机应用研究, 2002(3): 54-57.
- [3] 孟艳君, 卞红雨. 水声通信网 MAC 层协议的研究与仿真

5 结束语

该系统对常用的语音特征参数 LPCC、LPCC + Δ LPCC、MFCC、MFCC + Δ MFCC 参数进行了研究,并用实验比较了它们的性能,得出 MFCC + Δ MFCC 参数的识别率最高,LPCC 的识别率最低。系统采用的语音库仅为 30 人,如果语音库加大,那么识别率将不会太理想,所以下一步将会探求更准确的特征参数比如混合特征参数、小波变换以及脉冲耦合神经网络(PCNN)^[9]的引入。

参考文献:

- [1] 易克初, 田斌, 付强. 语音信号处理[M]. 北京: 国防工业出版社, 2000.
- [2] 韩纪庆, 张磊, 郑铁然. 语音信号处理[M]. 北京: 清华大学出版社, 2004.
- [3] 荣薇, 陶智, 顾济华. 基于改进 LPCC 和 MFCC 的汉语耳语音识别[J]. 计算机工程与应用, 2007, 43(30): 213-215.
- [4] 刘雅琴, 裘雪红. 应用小波包变换提取说话人识别的特征参数[J]. 计算机工程与应用, 2006, 42(9): 67-69.
- [5] Li Fuhai, Ma Jinwen, Huang Dezhi. MFCC and SVM Based on Recognition of Chinese Vowels[C]//CIS 2005, Part II, LNAI 3802. [s.l.]: [s.n.], 2005: 812-819.
- [6] 何强, 何英. MATLAB 扩展编程[M]. 北京: 清华大学出版社, 2002.
- [7] 罗世谦, 冯子亮, 张恒. 一种基于能量聚类分析的句子语音端点检测法[J]. 计算机技术与发展, 2008, 18(4): 13-15.
- [8] 由守杰, 柏森, 曾辉. 鲁棒的混合域音频信息隐藏算法[J]. 计算机技术与发展, 2008, 18(3): 169-172.
- [9] 卢桂馥, 王勇, 宴易文. 一种新的基于 PCNN 的图像脉冲噪声滤波算法[J]. 计算机技术与发展, 2007, 17(12): 83-85.

[D]. 哈尔滨: 哈尔滨工程大学, 2006.

- [4] Ousterhout J. Scripting: Higher-level programming for the 21st century[J]. IEEE Computer, 1998, 31(3): 23-30.
- [5] 黄镇建. NS2 中新协议的实现[J]. 计算机系统应用, 2009(1): 119-121.
- [6] Fall K, Varadhan. The ns Manual[EB/OL]. 2008-12-10. <http://www.isi.edu/nsnam/ns/ns-documentation>.
- [7] Liebeherr J, Christin N. JoBS: Joint buffer management and scheduling for differentiated services[C]//In: Proceedings of IWQoS 2001. Karlsruhe, Germany: [s.n.], 2001: 404-418.
- [8] 张亚明, 陈绍炜, 夏林英. 基于 NS2 的无线网络仿真研究[J]. 通信技术, 2007(3): 58-60.