AQM算法在NS2中的实现及其性能评价

王秀利
(中央财经大学信息学院 北京 100081)

摘 要 为了评价一种新的主动队列管理(active queue management,简称AQM)算法而改变运行在成千上万个路由器上的软件是不现实的。研究人员被迫在模拟网络或私用网络上测试新算法。分析了网络模拟器NS2的模块组成,并以PSO-PID(Particle Swarm Optimization-Proportional Integral Differential)算法为例,重点描述新AQM算法如何在NS2中实现,详细介绍如何获取AQM算法的评价指标,如平均队列长度、队列抖动、丢包率、链路利用率等定量指标,并结合队列长度变化曲线对AQM算法性能进行评价。

关键词 网络拥塞控制,主动队列管理,网络模拟,性能评价,NS2
中图法分类号 TP393 **文献标识码** A

Implementation and Evaluation of the Active Queue Management Algorithm in NS2

WANG Xiu-li
(School of Information, Central University of Finance and Economics, Beijing 100081, China)

Abstract It's impossible to alter the software running on the thousands upon thousands routers to evaluate a novel active queue management (AQM) algorithm. Researchers have to test their algorithms on simulator or private network. The structure of NS2 was first described, and then the implementation of novel AQM algorithm in NS2 was presented with the PSO-PID (Particle Swarm Optimization-Proportional Integral Differential) algorithm as an example. Finally, the AQM algorithms were evaluated in terms of the queue length transformation and the quantity index, such as average queue length, the queue length standard deviation, the packet loss rate, and the link utilization.

Keywords Network congestion control, Active queue management, Network simulation, Evaluation, NS2

1 引言

近年来,主动队列管理(active queue management,简称AQM)[1]成为网络拥塞控制研究中的一个技术热点。它通过网络中间节点有控制的分组丢弃机制,实现了较低的排队延时和较高的有效吞吐量。研究人员提出了多种AQM算法,如RED[2], ARED[3], SRED[4], ERED[5], PI[6], Adaptive PI[7],PIP[8],DITAE-PID[9],PSO-PID[10]等。

如何评价新的AQM算法?直接在Internet上做实验,会因新的AQM算法涉及到改变路由器而使实验失败。为了评估一种新的AQM算法而改变运行在成千上万个路由器上的软件是不现实的。在这种情况下,网络设计者被迫在模拟网络或私用测试网络上测试新的算法。

目前,网络模拟器主要有OPNET(OPNET Modeler),GloMoSim(Global Mobile Information System Simulation Library),GaliLEO,NS2(Network Simulator version 2.x),它们的侧重点不一样,功能也不尽相同[11,12]。NS2免费使用并且完全开放源代码[13,14],使用者可以根据自己的需求编写代码,扩充NS2的功能,因此备受研究人员关注。

本文重点描述新的AQM算法(以PSO-PID算法为例)如何在NS2中实现,给出实现原理、实现步骤及代码结构,详细介绍如何获取AQM算法的评价指标,如平均队列长度、队列抖动、丢包率、链路利用率等量化指标,并结合队列长度变化曲线对AQM算法性能进行评价。

2 NS2模块组成及功能

NS-allinone2.33[13]包含11个模块,如表1所列。

表1 NS2模块组成

模块	版本号	必选/可选
Tcl	Tcl release 8.4.18	必选
Tk	Tk release 8.4.18	必选
OTcl	OTcl release 1.13	必选
Tclcl	Tclcl release 1.19	必选
Ns	Ns release 2.33	必选
Nam	Nam release 1.13	可选
Xgraph	Xgraph release 12.1	可选
Gt-itm	Georgia Tech Internetwork Topology Models	可选
SGB	Standford GraphBase	可选
Cweb	Cweb	可选
Zlib	Zlib release 1.2.3	可选

必选模块缺一不可,否则NS2不能正常工作,而可选模块能够增强NS2的功能。Tcl与Tk是安装在Unix/Linux环

到稿日期:2008-11-11 本文受国家自然科学基金项目(60743005,70872120,70872119),北京市自然科学基金项目(9092014),中央财经大学"中财121人才工程"青年博士发展基金项目(QBG0702)资助。
王秀利(1977-),男,博士,讲师,CCF会员,主要研究方向为计算机网络、信息安全、可信计算等,E-mail:xlwang.cufe@gmail.com。

境下的两个包，它们一起构成了一套开发系统应用程序和图形用户界面(Graph User Interface，简称 GUI)应用程序的环境；OTcl 是 MIT Object Tcl 的简称，是 Tcl 面向对象的扩展；Tclcl 模块包含 Tcl/C++的接口；Ns 是 NS2 的主体代码模块，内含一个节点移动产生器、两个传输事件产生器；Nam 即 Network Animator，它与 Ns 协同工作，将 Ns 仿真过程动态表现出来；Xgraph 是 X-Windows 应用程序，完成交互式测量，绘制动画效果，显示网络运行的数值特征；Gt-itm 产生模拟 Internet 网络结构的拓扑图；SGB 是图形产生器；Cweb 模块是与网页相关的工具；Zlib 是通用数据压缩库(data compression library)。

NS2 功能示意图如图 1 所示，粗框里为 NS2 的模块，方框外的 Script 为脚本文件。Ns 解释脚本，并将输出写到输出文件中，然后调用 Nam 或 Xgraph 显示输出文件。

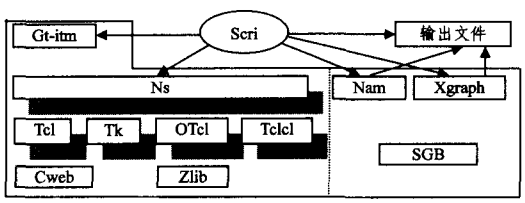

图 1　NS2 功能示意图

3　PSO-PID 算法描述

微粒群优化(particle swarm optimization，简称 PSO)算法是一种基于种群搜索的自适应进化计算技术。PSO 算法概念简单，实现容易，参数较少，能有效解决复杂优化任务。PSO-PID 是利用 PSO 算法整定比例-积分-微分(proportional-integral-differential，简称 PID)控制器参数[10]，PSO-PID 算法简要描述如下：

Step1　初始化。根据 PID 参数优化设计问题，设置初始参数，如种群规模、最大迭代次数、加速度常数、惯性权重的上下限、三个控制器参数的上下限、最大速度，并初始化种群中粒子的初始位置、初始速度、个体最佳位置 pbest 和全局最佳位置 gbest。

Step2　惯性权重更新。

Step3　个体最佳位置和全局最佳位置更新。根据适应度函数计算种群中每个粒子的适应度值并进行比较，最佳适应度值相应的位置即最佳位置，记为 pbest。所有粒子中最佳的 pbest 记为 gbest。

Step4　微粒速度和位置更新。

Step5　判断结束条件。如果已达到最大迭代次数或最优目标条件，跳转到 Step 6；否则，跳转到 Step 2，继续下一循环的优化求解。

Step6　求得最优值。通过最终的 gbest 可以求得控制器最优参数值 a,b,c。

4　算法实现

本节以 PSO-PID 算法为例，描述如何在 NS2 中实现新算法。

4.1　NS2 功能扩充原理

在表 1 所列的各个模块中，Ns 是最重要的部分。NS2 安装后，ns-2.33 目录包含许多子目录，它们各自实现了相应的一部分功能。比如 queue 子目录实现了路由器丢弃分组的算法，而与 TCP 有关的协议实现在 tcp 子目录下。此外，ns-2.33 目录下还有一些 makefile 文件。这些文件包含了每次编译、生成新的 ns.exe 时，需要编译的文件信息。

一般情况下，添加新协议可以在 ns-2.33 目录下找到实现相应功能的子目录，在该子目录下直接添加实现新协议的文件，然后再重新编译生成 ns.exe。新文件往往可以通过继承等方法部分利用 NS 已有的代码。

添加新协议的具体步骤如下：

1) 定义或继承 C++协议类；

2) 编写该类成员函数和协议算法；

3) 定义 Tcl 相关的类和变量；

4) 把 C++代码绑定到 Tcl；

5) 修改 makefile 文件，重新编译生成 ns.exe 文件。

4.2　算法代码结构

在 PSO-PID 主动队列管理算法中，当缓冲未满时，新到达的分组将按一定概率丢弃(或标记)。丢包概率的计算按照下式进行：

$$p=a\times(q-q\_ref)-b\times(q\_old-q\_ref)+c\times(q\_oldold-q\_ref)+p\_old \quad (1)$$

其中 q 为当前队列长度，q_ref 为期望队列长度，q_old 为上一时刻的队列长度，q_oldold 为上两个时刻的队列长度，a,b,c 为 PSO-PID 控制器的参数，由第 3 节描述的 PSO-PID 算法得出[10]，$a=3.7495\times10^{-6}$，$b=1.1300\times10^{-7}$，$c=3.3447\times10^{-6}$。

在 RED 算法中，每当有分组到达时路由器就对丢包概率重新计算。而 PSO-PID 则不同，它对丢包概率 p 的更新并不是时刻进行的。PSO-PID 中有一个采样频率，每过一个周期，算法才需要对丢包概率重新计算。与 RED 相比，这将省下路由器的大量资源。

下面具体说明 PSO-PID 算法在 NS2 中的实现。在此仅描述实现的步骤，并列举部分关键代码。

为实现新的 PSO-PID 算法，需要建立一个新的 PIDQueue 类。由式(1)可以看出，在 PIDQueue 的结构中应该有一组变量来记录相关参数的初始值，如 a,b,c 及采样频率等。此外，还需另一组参数记录 PIDQueue 的一些历史状态以计算新的丢包率，如上一时刻的队列长度及丢包率等。为此，本文提供两个结构：edp 和 edv。edp(early drop parameters)用来记录初始参数情况，这些参数的数值由用户提供，在程序中不再更改。edv(early drop variables)用来记录队列的历史状态信息，这些变量值在程序运行中根据队列状态动态更改，由 PIDQueue 自己维护。

edp 具体结构如下：

```
struct edp {
int mean_pktsize;    /* avg pkt size, linked into Tcl */
int bytes;   /* true if queue in bytes, false if packets */
int setbit;   /* true to set congestion indication bit */
double a,b,c;   /* parameters of PSO-PID controller */
double w;   /* sampling frequency (times per second) */
int q_ref;   /* desired queue size */
};
```

其中，*mean_pktsize* 为平均分组大小；*bytes* 和 *setbit* 为两个标

志，*bytes* 通过 0，1 设置指出队列长度是通过缓存中分组的个数来计算还是通过字节来计算，*setbit* 指出当检测到拥塞时是否采用位标记的办法；*a*，*b*，*c* 为 PSO-PID 控制器的参数；*w* 为采样频率；*q_ref* 为期望队列长度。

edv 具体结构如下：

```
struct edv {
TracedDouble v_prob;  /* prob of packet drop before "count" */
int count;  /* of packets since last drop */
int count_bytes;  /* of bytes since last drop */
int q_old,q_oldold;
};
```

其中，*v_prob* 为丢包概率，*count* 和 *count_bytes* 分别为目前丢弃分组的个数和丢弃分组字节数，*q_old* 为上一时刻的队列长度，*q_oldold* 为上两时刻的队列长度。

由于 PSO-PID 算法需要每隔一段时间对丢包概率更新一次，为了保证这个过程能周期性进行，需要建立一个 PIDCalcTimer 类，它是 TimerHandler 的子类。

PIDQueue 类无疑是最重要的类，它将完成 PSO-PID 的算法调度。以上结构是对其建立的必要支持。在 PIDQueue 类中，主要成员变量包括：PIDCalcTimer *CalcTimer*，edp *edp_*，edv *edv_*，int *curq*（当前队列长度），PacketQueue *q_*。

PIDQueue 类中的主要方法包括：

PIDQueue::PIDQueue(const char * *trace*);

PIDQueue 的构造函数，主要负责将一些 C++变量绑定到 Tcl 变量。如将 *edp_a* 绑定到 *a_*，将 *edp_q_ref* 绑定到 *q_ref_* 等。然后调用 *reset*()对相应变量进行初始化。

void *reset*();

reset 需要对队列中的变量进行初始化。包括：

edv_count = 0;

edv_count_bytes = 0;

edv_v_prob = 0;

edv_q_old = 0;

edv_q_oldold = 0;

*curq_* = 0;

calculate_p();

最后还需要调用 *calculate_p*()。在这里是第一次调用该方法，对丢包率进行了初始计算，此后对丢包率的更新计算周期进行。

void *enque*(Packet * *pkt*);

当分组到达路由器时调用此方法。它首先更新 *curq*，如果缓冲区已满，则只有丢弃分组；如果未满，调用 *drop_early* 方法，按一定概率丢掉或标记最近到达的分组。如果不丢弃分组，把新到达的分组加入缓冲区中。

int *drop_early*(Packet * *pkt*, int *qlen*);

drop_early 方法在缓冲未满时决定是否提前丢弃分组。它首先产生一个随机数，然后比较随机数和丢包率，当随机数大于丢包率时丢包，返回 1，否则返回 0。

Packet * *deque*();

当路由器中有分组被发送时调用此方法。返回出队列的分组，并对 *curq* 进行更新。

double *calculate_p*();

calculate_p 根据采样频率，每隔一段时间执行一次，然后按照式(1)对丢包概率重新进行计算。

程序中，首先得到当前队列长度：

```
int qlen = qib_ ? q_->byteLength():q_->length();
```

然后对丢包概率进行更新：

```
double p;
if (qib_) {
p = edv_v_prob +
edp_a × (qlen × 1.0/edp_mean_pktsize-edp_q _ref ) +
edp_b×(edv_q _old×1.0/edp_mean_pktsize-edp_q _ref) +
edp_c×(edv_q _oldold×1.0/edp_mean_pktsize-edp_q _ref);
}
else {
p = edv_v_prob + edp_a × (qlen-edp_q _ref) +
edp_b× (edv_q _old-edp_q _ref) +
edp_c× (edv_q _oldold-edp_q _ref);
}
```

其中 *qib_* 标志队列长度是用分组的个数来表示还是用字节数来表示。当 *qib_* 为 1 时，表示用字节数来表示队列长度。*edv_* 结构记录了一组状态变量。其中 *edv_v_prob* 为上一时刻的丢包概率，*edv_qold* 为上一时刻队列长度，*edv_qoldold* 为上两时刻队列长度。而 *edp_* 结构记录了一组初始的参数。其中 *edp_mean_pktsize* 为平均分组大小，*edp_qref* 为期望队列长度，*edp_a*，*edp_b*，*edp_c* 为算法和模型提供的参数。上面的代码和前面关于丢包概率的计算讨论是一致的。

随后，添加以下代码，以对状态变量进行更新：

edv_v_prob = *p*;

edv_qoldold = *edv_qold*;

edv_qold = *qlen*;

最后，将新得到的丢包概率作为返回值返回之前，调用 CalcTimer. resched (1.0/*edp_w*)；使得此方法在下一个周期能够再被自动调用。

4.3 Tcl 变量初始化

在 ns-2.33 目录下的 ns-default. tcl 文件里，对 PSO-PID 算法中的一些 Tcl 变量设置默认值。进行的设置如下：

```
Queue/PID set bytes_false
Queue/PID set queue_in_bytes_false
Queue/PID set w_160
Queue/PID set qref_150
Queue/PID set mean_pktsize_500
Queue/PID set setbit_false
Queue/PID set prob_0
Queue/PID set curq_0
```

这些语句对 PSO-PID 算法中的一些变量值进行了默认设置，包括：队列长度使用分组个数进行计算而不是使用字节，采样频率为 160Hz，期望队列长度为 150 个分组，平均分组大小为 500 字节，检测到拥塞时直接丢弃分组而不是采用位标记的办法，设置初始的丢包率和队列长度为 0。

4.4 重新编译

打开 ns-2.33 目录下的 makefile. vc 文件，在其输出文件列表中加入新协议的输出文件 queue/pid. o。然后在 ns-2.33 根目录下执行 nmake makefile. vc，此命令将对新添加的协议文件进行编译。之后，新的协议被加入到模拟器中，NS2 的功能得以扩充。

至此，已完成 PSO-PID 新算法在 NS2 中的实现。

5 性能评价

5.1 NS2 的跟踪机制

NS2 可以用以下两种机制在仿真后给出跟踪结果。

1) 通过跟踪变量的方法

NS2 源文件中有一种特殊的变量即跟踪变量。比如 queue 目录下的 red.h 文件中，*v_ave* 和 *v_prob* 均为 TracedDouble 型变量。这两个跟踪变量可以记录使用 RED 为 AQM 算法的情况下，路由器平均队列长度及丢包率的实时变化。使用这种方法可以通过编写仿真脚本，直接对需要跟踪的变量进行跟踪，运行仿真脚本的速度较快，运行后可以将跟踪变量各时刻的数值输出到指定文件中。但跟踪的内容受到跟踪变量的限制，一些统计变量，如一段时间内的链路利用率等数据无法直接得到。

2) 通过记录仿真过程到详细跟踪文件的方法

NS2 可以在仿真脚本中生成一个详细的跟踪文件。该文件将记录仿真过程中网络各节点和各数据分组的全部动作。有以下两种方式产生输出文件：

① namtrace-all 命令

把 NS2 工具包中的 nam 模块编译后，产生 nam.exe 文件。在仿真脚本中通过使用“ns namtrace-all $ outfile”命令，可以把仿真过程记录到 outfile 文件中。之后，Nam 可以通过 outfile 中的记录，用动画方式显示仿真的整个过程，比如：数据包的传输、链路断开、节点的移动、丢包等等。

② trace-all 命令

在仿真脚本中通过使用“ns trace-all $ outfile”命令把仿真过程记录到 outfile 文件中。这个记录文件与使用 namtrace-all 命令后得到的记录文件不同。它虽然不能以动画方式回放仿真过程，但是记录了各时刻网络各节点和各数据分组的全部动作。

5.2 本文采用的方法

由于 nam 提供的动画功能在文中不便展示，因此在本文的实验分析中并未使用。本文混合使用了跟踪变量及使用“trace-all”命令得到详细跟踪文件的两种方法，给出 AQM 算法运行下网络的性能评估指标，包括平均队列长度、队列抖动、丢包率、链路利用率，以此来对算法做出评价。

在给出 AQM 算法量化评估指标的同时，本文还给出了队列长度变化曲线图。因为通过队列长度变化曲线，可以对算法的实际运行情况进行感性观察。如直观上可以看出平均队列长度、抖动的情况。此外，也可以对丢包率和链路利用率做出评判。丢包通常会在两种情况下造成：一是算法主动对其丢弃；二是由于缓存溢出被迫将其丢弃。如果队列长度在某段时期内持续保持在缓冲能力的最大值，意味着在这段时间里到达的分组都将很可能因缓存溢出被丢弃。反之，如果队列长度在某段时期内持续保持在 0 附近，则这段时间内，链路有可能空闲，即利用率可能不高。

网络拓扑结构如图 2 所示，并将其和典型的 RED，PI 算法进行比较。

瓶颈链路位于节点 *A* 和节点 *B* 之间，链路容量 15Mbps (3750packets/s，分组缺省大小为 500bytes)，延时 5ms。所有业务源均为 FTP 业务源。它们与节点 *A* 之间的链路容量均为 10Mbps，各业务源节点到 *A* 节点的延时为 *t* ms。除节点 *A* 的队列分别由 RED，PI，PSO-PID 控制外，其余均为 Drop Tail。RED 的高低门限值分别为 100packets 和 200packets，PI 和 PSO-PID 的队列长度的期望值为 150packets。节点缓存大小均为 300packets，接收端窗口大小采用默认值 20packets。节点 *B* 和节点 *C* 之间的时延为 *d* ms。

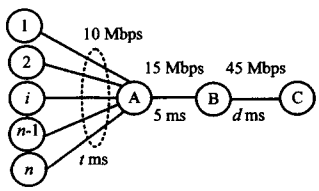

图 2 仿真网络拓扑结构

取 *n* 为 60，节点 *B* 和节点 *C* 之间的时延 *d* 为 150ms，各业务源节点到 *A* 节点的延时 *t* 分别为 $(2\times i)$ms，即从 2ms 到 120ms。所有 FTP 业务源均在 0 时刻启动。表 2 和图 3 给出了仿真结果。表 2 给出了平均队列长度、队列抖动、丢包率、链路利用率。图 3 中横坐标表示时间，单位为秒，纵坐标表示队列长度，单位为包的个数。

表 2 RED，PI，PSO-PID 性能指标

评价指标	RED	PI	PSO-PID
平均队列长度(packets)	71	148.5	147.6
队列抖动	67.9	54.4	47.6
丢包率(%)	1.30	0.97	0.78
链路利用率(%)	92.5	95.7	98.3

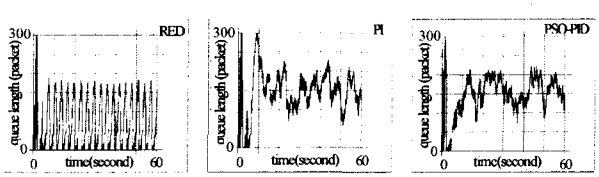

图 3 RED，PI，PSO-PID 队列长度变化

RED 的队列出现了大幅振荡。队列的大幅振荡一方面增加了端到端的时延抖动；同时由于空队列出现的概率加大，导致链路利用率降低，这两点都违背了最初提出的 AQM 设计目标。PI 出现了丢包现象，并且需要较长时间才能使队列长度趋于期望值，较慢的响应速度同样违背了 AQM 的设计目标，另外，如果瞬态过程太长，在千变万化的网络环境下，队列长度很难稳定在期望值附近。PSO-PID 则体现了较好的动态性能，链路利用率较高，丢包率较小，队列抖动较小。

结束语 本文分析了网络模拟器 NS2 的模块组成，并以 PSO-PID 算法为例，详细描述了 AQM 算法在 NS2 中的实现。最后，结合队列长度变化曲线，以平均队列长度、队列抖动、丢包率、链路利用率等为定量指标，对 AQM 算法性能进行了评价。

参 考 文 献

[1] Braden B, Clark D, Crowcroft J, et al. Recommendations on queue management and congestion avoidance in the Internet[EB/OL]. RFC2309, 1998. http://www.rfc.net/rfc2309.html

[2] Floyd S, Jacobson V. Random early detection gateways for congestion avoidance[J]. IEEE/ACM Trans. on Networking, 1993, 1(4): 397-413

[3] Feng WC, Kandlur DD, Saha D, et al. A self-configuring RED

gateway[A]// Proc. of the IEEE INFOCOM[C]. New York: IEEE Press, 1999: 1320-1328

[4] Ott TJ, Lakshman TV, Wong LH. SRED: Stabilized RED[A]// Proc. of the IEEE INFOCOM[C]. New York: IEEE Press, 1999: 1346-1355

[5] Liu S, Basar T, Srikant R. Exponential-RED: A stabilizing AQM scheme for low-and high-speed TCP protocols[J]. IEEE/ACM Trans. on Networking, 2005, 13(5): 1068-1081

[6] Hollot CV, Misra V, Towsley D, et al. On designing improved controllers for AQM routers supporting TCP flows[A]// Proc. of the IEEE INFOCOM[C]. Anchorage: IEEE Press, 2001: 1726-1734

[7] 卢锡城，张明杰，朱培栋. 自适应 PI 主动队列管理算法[J]. 软件学报，2005，16(5)：903-910

[8] Zhang HY, Liu BH, Dou WH. Design of a robust active queue management algorithm based on feedback compensation[A]// Proc. of the ACM SIGCOMM[C]. Karlsruhe: ACM Press, 2003: 277-285

[9] 王秀利，王永吉，周辉，等. 基于 D 稳定域和 ITAE 准则的主动队列管理算法[J]. 软件学报，2007，18(12)：3092-3103

[10] Wang X L, Wang Y J, Zhou H, et al. PSO-PID: a novel controller for AQM routers[A]// Proc. of the IEEE/IFIP WOCN[C]. Bangalore: IEEE Press, 2006: 1-5

[11] 王秀利，王永吉. 一种开放源代码的网络仿真器的原理与实现[J]. 计算机工程与应用，2004，40(15)：137-140

[12] 金信苗. 基于 ns2 的 LEO 卫星网络路由算法模拟[J]. 计算机科学，2007，34(1)：57-60

[13] Nsnam[EB/OL]. http://www.isi.edu/nsnam/

[14] Fall K, Varadhan K. The ns Manual[EB/OL]. The VINT Project: UC Berkeley, LBL, USC/ISI, and Xerox PARC. 2003

(上接第 35 页)

延迟时间，达到 1ms 左右。而 IPv4 端节点向 IPv6DNS 服务器发出的 ICMP 报文延迟则没有这种现象。这种区别是由 Napt-Pt 翻译网关的内部工作机制带来的。前者翻译 ICMP 报文时使用了端口映射机制，后者采用了 3.2 节描述静态地址绑定，其地址映射被固定在地址映射链表中，不需要获取新的地址映射 IPv4 地址，并对映射链表进行复杂的插入和删除。

这里特别指出，ICMP 报文中没有端口字段，但是有 Identifier 字段用以标示不同的 ping 进程。这里用 ICMP 报文中 Identifier 值作为映射机制中的目的端口和源端口。

表 1 IPv6_to_IPv4 的 ICMP 报文响应延迟时间

(单位毫秒)	最小时间	平均时间	最大时间	时间方差
穿越 Napt-Pt 的应答延迟	0.449	0.493	1.06	0.0136
纯 IPv6 环境下的应答延迟	0.269	0.2812	0.318	0.00012

表 2 IPv4_to_IPv6 的 ICMP 报文响应延迟时间

(单位毫秒)	最小时间	平均时间	最大时间	时间方差
穿越 Napt-Pt 的应答延迟	0.474	0.5256	0.650	0.0017
纯 IPv4 环境下的应答延迟	0.268	0.2795	0.282	0.0000129

3.3 FTP 传输性能分析

表 3 中用 IPv6_to_IPv4 来表示 IPv6 端节点向 IPv4 端节点发出的连接，用 IPv4_to_IPv6 来表示 IPv4 端节点向 IPv6 端节点发出的连接。我们发现经过 Napt-Pt 过渡的 FTP 连接响应时间和传输速度比纯 IPv4 和纯 IPv6 以太网环境下略有下降。这是因为 Napt-Pt 本身对分组的转换过程需要一定的时间，而且 Napt-Pt 工作于网络层，FTP_ALG 更是处在应用层，每个报文的翻译都要经过翻译网关从底层到高层的多层处理，造成时间上面的损耗。我们还发现 Napt-Pt 技术的双向连接性能存在一些差异，IPv6_to_IPv4 情况下 FTP 建立连接响应时间和文件传输速度稍慢于 IPv4_to_IPv6。由图 3 可知，程序首先查找地址映射链表，然后再查找端口映射链表，由结果来判断报文采用的映射机制。IPv4_to_IPv6 情况下采用的是地址映射机制，会直接从地址映射链表中得到查询结果，从而跳过了端口映射链表的查询，这使得 IPv4_to_IPv6 时查找链表的时间耗费更低，从而有更快的报文翻译速度。从表 3 中可以看到，经过翻译网关的网络传输速度与单纯网络环境下的传输速度的差距很小，翻译网关使用 2.8G 的 CPU 已经基本满足百兆网络满负荷传输的需要了。

这里还要特别指出，无论 FTP 协议采用何种工作机制，都有两条 TCP 连接，控制连接和传输连接。端口映射机制下会为这两条 TCP 连接分别建立两个映射节点。当传输连接对应的映射节点还存在的时候，其控制连接对应的映射节点绝不能因为长时间没有报文传输而被清除。也就是说与 FTP 传输相关的这两个端口映射节点的生命周期必须被同步更新。

表 3 FTP 传输性能比较

(单位毫秒)	以太网		基于 Napt-Pt 技术	
	纯 IPv4 环境	纯 IPv6 环境	IPv4_to_IPv6	IPv6_to_IPv4
连接响应时间	4.923ms	4.893ms	5.423ms	5.522ms
FTP 传输速度	11.1MB/s	11.2MB/s	11.0MB/s	10.0MB/s

结束语 本文在 Nat-Pt 动态地址映射的研究基础上添加了端口映射机制，实现了基于混合映射机制的 Napt-Pt 翻译网关。根据网络连接发起的方向不同，分别采用地址映射机制和端口映射机制，使用少量 IPv4 地址就能处理大量会话，扩展了翻译网关的传输能力。结合项目需求搭建了完善的实验环境。实验证明，Napt-Pt 翻译网关技术能有效地实现网络基本服务的过渡功能，而且对网络传输的影响非常有限。

本文结合项目要求实现了网络基本服务的过渡，然而网络服务的种类非常丰富，未来将对 Napt-Pt 翻译网关进行更多的扩展，并运用更多种类的网络服务测试来验证翻译网关的可靠性和扩展性。

参 考 文 献

[1] Bradner S, Mankin A. RFC 1752. The recommendation for the IP next generation protocol [S]. IETF, 1995

[2] Deering S, Hinden R. RFC 2460. Internet Protocol, Version 6 (IPv6) Specification [S]. IETF, 1998

[3] Davies J. Understanding IPv6. Second Edition [M]. Seattle: Microsoft Press, 2002: 76-83

[4] Gilligan R, Nordmark E. RFC 2893. Transition Mechanisms for IPv6 Hosts and Routers [S]. IETF, 2000

[5] Tsirtsis G, Srisuresh P. RFC 2766. Network Address Translation-Protocol Translation (NAT-PT) [S]. IETF, 2000

[6] Nat-PtImplementationIntro[OL]. http://www.eurescom.ed/~public-webspcae/P1000-series/P1009/ doc3_1.html

[7] Nordmark E. RFC 2765. Stateless IP/ICMP Translator (SIIT) [S]. IETF, 2000

[8] [Allman M, Ostermann S. RFC 2428. FTP Extensions for IPv6 and NATs [S]. IETF, 1998