ISSN 1000-9825, CODEN RUXUEW
Journal of Software, Vol.19, No.5, May 2008, pp.1234−1255
DOI: 10.3724/SP.J.1001.2008.01234

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel/Fax: +86-10-62562563

可用带宽度量系统中的若干基本问题*

周 辉[1,4], 李 丹[2], 王永吉[1,3+]

[1](中国科学院 软件研究所,北京 100190)

[2](清华大学 计算机科学与技术系,北京 100084)

[3](中国科学院 计算机科学国家重点实验室,北京 100190)

[4](中国科学院 研究生院,北京 100049)

Fundamental Problems with Available Bandwidth Measurement Systems

ZHOU Hui[1,4], LI Dan[2], WANG Yong-Ji[1,3+]

[1](Institute of Software, The Chinese Academy of Sciences, Beijing 100190, China)

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

[3](State Key Laboratory of Computer Science, The Chinese Academy of Sciences, Beijing 100190, China)

[4](Graduate University, The Chinese Academy of Sciences, Beijing 100049, China)

+ Corresponding author: E-mail: ywang@itechs.iscas.ac.cn, http://www.ios.cn/

Zhou H, Li D, Wang YJ. Fundamental problems with available bandwidth measurement systems. *Journal of Software*, 2008,19(5):1234−1255. http://www.jos.org.cn/1000-9825/19/1234.htm

Abstract: Based on the analysis of 13 well-known available bandwidth systems and the experience in designing, building, and deploying the BNeck system, the fundamental problems with available bandwidth measurement systems are analyzed comprehensively.

Key words: computer network; available bandwidth; network measurement; active probing

摘 要: 结合过去 3 年中设计、开发和部署 BNeck 系统的经验,分析了 13 个可用带宽度量系统的工作原理及软件实现,并在此基础上总结了可用带宽度量系统中存在的若干基本问题.对这些问题的研究,不仅有利于设计和开发出高效、准确的度量系统,也有利于推动网络管理和视频通信等多个相关领域的共同发展.

关键词: 计算机网络;可用带宽;网络度量;主动探测

中图法分类号: TP393 **文献标识码**: A

可用带宽是指在不影响网络路径中背景数据流的情况下,端到端通信所能获得的最大数据传输率[1].可用带宽是计算机网络的核心概念之一,它在协议设计、网络管理、QoS 部署、组播通信和多媒体传输等诸多方面

* Supported by the National Natural Science Foundation of China under Grant Nos.60673022, 60273026, 60473060 (国家自然科学基金); the National High-Tech Research and Development Plan of China under Grant Nos.2004AA1Z2100, 2005AA113140 (国家高技术研究发展计划(863)); the One-Hundred-People Program of the Chinese Academy of Sciences under Grant No.BCK35873 (中国科学院百人计划); the Chinese Academy of Sciences and the Royal Society of the United Kingdom Project under Grant Nos.20030389, 20032006 (中国科学院与英国皇家学会联合资助项目)

Received 2006-09-12; Accepted 2007-01-23

有着重要意义.例如,TCP 协议的一个主要设计目标就是在不堵塞网络的情况下,最大限度地利用可用带宽[2].在每一个网络中,管理员都需要监测各个链路的负载情况进而合理地配置网络资源,可用带宽正是网络监测和配置的重要参考依据.而且,能否获得预期的可用带宽,也是网络运营商(ISP)部署 QoS 的关键考核指标.与此同时,大量的网络应用也迫切需要具备快速度量可用带宽的能力.组播通信就需要在占用最小网络资源(包括可用带宽)的情况下,协同多个网络节点以维护高效的组播路由树[3].多媒体应用如点对点视频会议,同样需要快速感知各节点间可用带宽的变化,并据此及时调整数据压缩算法和传输策略.

在过去的 10 年里,为了快速、准确地度量可用带宽,国内外学者们基于主动探测机制研发了多个度量系统.以这些度量系统为主的设计、开发、实验、评估及相关的研究工作极大地推动了网络度量技术的发展.然而到目前为止,这些度量系统仍然处于实验研究的阶段,始终没有一个系统能够满足广域网环境中的度量要求[4],更没有一个系统被互联网用户广泛接受和推广.为此,迫切需要我们去研究度量系统中普遍存在的基本问题,找到限制系统适用范围的根本原因.

本文结合过去 3 年中设计、开发和部署 BNeck 系统[5]的经验,分析了 13 个常见度量系统的工作原理及软件实现,同时在不同软、硬件配置的单个节点内、机构内部搭建的实验网络中以及跨地域的覆盖型实验网络上,针对各种应用场景开展了大规模的实验,进而在理论分析和实验数据的基础上总结了可用带宽度量系统中普遍存在的基本问题.分析表明,正是这些基本问题极大地限制了度量系统的适用范围.深入研究这些问题,不仅有利于设计和开发出在复杂网络环境中高效运行的度量系统,而且有利于从整体上推动网络度量技术的发展.下文首先介绍可用带宽的基本定义,然后根据运行模式将度量系统分为单终端系统和双终端系统两类,并讨论它们的工作原理和假设前提.接下来从计时误差、低接入带宽和数据包异常等不同角度研究两类度量系统中共同存在的基本问题,然后,根据单、双终端系统各自的特点,进一步讨论两类系统中分别存在的基本问题.

1 可用带宽及其度量

1.1 基本定义

可用带宽是在网络路径的上下文中定义的.网络路径由一系列存储转发链路组成,它能够把数据包从源终端主机(R_0)通过一系列中间节点 $R_1,R_2,\ldots,R_{n-1}$ 传输到目标终端主机(R_n).链路 L_i 是从 R_i 到 R_{i+1} 的数据传输通道,这里,$0\le i\le n-1$.在 t 时刻,L_i 的利用率 $U_i(t)$ 为

$$U_i(t)=\begin{cases}0, & \text{当}L_i\text{空闲时}\\ 1, & \text{当}L_i\text{传输数据包时}\end{cases} \tag{1}$$

当 $U_i(t)=0$ 时,L_i 的传输速率为 0;反之,L_i 以固定的速率 C_i 传输数据包.这里,C_i 又称为 L_i 的物理传输速率或带宽容量(capacity),它是由链路的物理特性决定的,与时间 t 无关.端到端(R_0 到 R_n)数据传输率的上限 C_P 由最小的链路带宽容量决定,

$$C_P=\min_{i=0\ldots n-1}\{C_i\} \tag{2}$$

决定 C_P 的那段链路称为狭窄链路(narrow link)[1].同一网络路径中可能存在多条狭窄链路.

在某一时段$[t,t+\tau]$内,链路 L_i 的利用率为

$$U_i(t,t+\tau)=\frac{1}{\tau}\int_t^{t+\tau}U_i(t)\mathrm{d}t \tag{3}$$

易知 $U_i(t,t+\tau)\in[0,1]$.在时段$[t,t+\tau]$内,L_i 的平均数据传输率为

$$\lambda_i=C_i\cdot U_i(t,t+\tau) \tag{4}$$

一般称λ_i为 L_i 中背景数据流(cross-traffic)的传输速率.背景数据流即 L_i 正常传输的数据流,这样称呼是为了把它们与度量系统出于探测目的而发送的探测数据流(probe-traffic)区分开来.

在时段$[t,t+\tau]$内,L_i 的可用带宽 A_i 是指 L_i 的平均空闲传输速率,即

$$A_i(t,t+\tau)=\frac{1}{\tau}\int_t^{t+\tau}(C_i-\lambda_i(t))\mathrm{d}t \tag{5}$$

本文所关注的可用带宽并非单个链路的可用带宽,而是待测网络路径的端到端可用带宽 A_P,它由路径中最小的链路可用带宽决定:

$$A_P(t,t+\tau)=\min_{i=0\ldots n-1}\{A_i(t,t+\tau)\} \tag{6}$$

决定 A_P 的那段链路常被称为紧凑链路(tight link)[6]或瓶颈链路(bottleneck link)[5].与狭窄链路不同,在一个网络路径中,瓶颈链路往往只有一条,且其位置还会随着网络负载的不同而发生变化.

在实际度量过程中,时间 τ 并不是固定的值,按照惯例,它等于度量系统的一个度量周期(即运行一次系统以度量 A_P 所用的时间)[1].所以,对于不同的度量系统而言,时间 τ 通常不同;即使是同一个系统,在软、硬件配置或者待测路径的属性发生变化时,τ 也会随之变化.

图 1 是上述概念的简单示例,其中,R_0 和 R_n 作为源终端主机和目标终端主机分别位于路径的首、尾两端,链路 L_i 的可用带宽 A_i 等于带宽容量 C_i 减去背景数据流的传输速率 λ_i.对于一个中间节点而言,背景数据流可以有多个来源(source)和去处(sink).

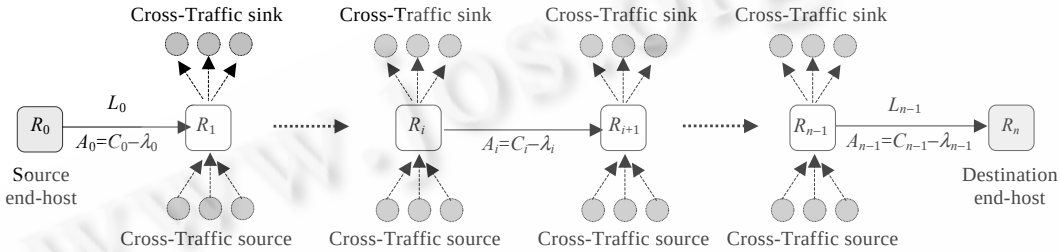


Fig.1 A simplified model of end-to-end network path

图 1 端到端网络路径的简化模型

1.2 度量系统

在过去的 10 年里,为了快速而准确地度量可用带宽,研究者们基于主动探测机制研发了大量的软件系统.其中常被讨论和比较的有 Abget[7],BFind[8],BNeck[5],CProbe[9],Delphi[10],IGI[6],Netest[11],Pathchirp[12],Pathload[1],Pipechar[11],SProbe[13],Spruce[14]和 TOPP[15]这 13 个系统(见表 1).按照运行模式的不同,这些系统可以分为单终端系统和双终端系统两类.其主要区别是,单终端系统仅需在 R_0 上安装软件,而双终端系统要求在 R_0 和 R_n 上都安装.相比之下,双终端系统的度量结果更准确,这是因为它能够利用 R_0 和 R_n 的协作简化度量过程,而不像单终端系统那样必须面对反向路径(R_n 到 R_0)的干扰.但是,双终端系统的应用范围非常有限,其原因是用户往往只能在 R_0(一般是用户自己的机器)上安装软件,而没有在远程的 R_n 上安装软件的权限.与此相反,单终端系统只需要安装在 R_0 上,就能用来度量以 R_0 为源终端主机的绝大多数网络路径的可用带宽.

第一个单终端系统是 CProbe[9],它同时也是最早的可用带宽度量系统.CProbe 充分利用了 ICMP 协议的请求-应答机制[16].具体地,CProbe 从 R_0 向 R_n 发送多组用于探测的 IP 包(IP 包中的 time-to-live 域被设置为 1,n 之间的值,且 destination port 域设置成一个非监听端口);这些 IP 包触发了节点的 ICMP 错误处理流程,所以,$R_1,\ldots,R_n$ 会向 R_0 发送 ICMP time-exceeded (TE)和 destination-unreachable (DU)包;CProbe 接收到 ICMP TE 和 DU 包后,将根据接收时间推算探测 IP 包在每段链路上传输速率的变化情况,进而估算可用带宽.在 CProbe 之后,Pipechar[11]也采用了基于 ICMP 协议的方法,而 SProbe[13]则是基于 TCP 和 HTTP 协议开发的,它们都有目的地利用了协议的特性以获取远程节点的反馈,并挖掘应答数据包的间隔变化规律进而估算可用带宽.但文献[17]中的工作证明了 CProbe 和 Pipechar 所度量的其实并不是可用带宽,而是处于可用带宽 A_P 和带宽容量 C_P 之间的非对称分散速率(asymptotic dispersion rate).所以,在 CProbe 和 Pipechar 之后的单终端系统,都必须证明它们所度量的确实是可用带宽.在这些后续的系统中,常被讨论的有 Abget[7],BFind[8]和 BNeck[5].

最典型的双终端系统是 Pathload[1].Pathload 的程序分为 *pathload_snd* 和 *pathload_rcv* 两部分,其中,*pathload_snd* 安装在 R_0 上,而 *pathload_rcv* 安装在 R_n 上.初始状态下,Pathload 认为可用带宽属于区间[0,C_0].然后,

pathload_snd 以速率 $x=C_0/2$ 发送 y 组探测数据包(默认情况下 $y=6$).R_n 上的 *pathload_rcv* 在接收到每一组数据包后,计算相邻探测数据包的间隔是否呈递增趋势.如果多组数据包的间隔都呈递增趋势,那么,Pathload 认为 $x>A_P$ 且 $A_P\in[0,x]$;否则,$x\le A_P$ 且 $A_P\in[x,C_0]$.假设在一次探测完毕后 $A_P\in[a,b]$,那么 *pathload_snd* 会以 $x=(a+b)/2$ 的速率发送 y 组数据包再一次探测网络路径并缩小可用带宽的所属区间.当 $(b-a)<\sigma$(或出现了意外)时,Pathload 才停止探测并输出 $[a,b]$,这里,σ 是事先设定的阈值.需要注意的是,Pathload 最终输出的是一个带宽区间,而不是单一数值.除了 Pathload 以外,常见的双终端系统还有 Delphi[10],IGI[6],Netest[11],Pathchirp[12],Spruce[14]和 TOPP[15].

Table 1 Measurement systems

表 1 度量系统

Type	System	Developer & affiliation	Used protocol*	Technical feature	Year
Single end-host system	Abget	Antoniades, *et al*., ICS, Greece	TCP, HTTP	Generates fake ACK packets to hack TCP connection	2005
	BFind	Akella, *et al*., Carnegie-Mellon Univ. and IBM Research	UDP, ICMP	Uses traceroute to monitor the queue size of middle nodes, linearly increases probe rate	2002
	BNeck	Zhou, *et al*., Institute of Software	UDP, ICMP	Gets the location, available-bw and capacity of bottleneck in one measurement cycle	2003
	Cprobe	Carter and Crovella, Boston Univ.	ICMP	Combines Bprobe[9], direct available-bw computation	1996
	Pipechar	Jin, *et al*., UC Berkeley	ICMP	Packet pair technique, used in NCS[11] infrastructure	2001
	SProbe	Saroiu, *et al*., Univ. of Washington	TCP, HTTP	Uses TCP protocol features to get designed TCP data	2002
Double end-host system	Delphi	Ribeiro, *et al*., Rice Univ.	UDP	Multifractal cross-traffic, link utilization estimation	2000
	IGI	Hu and Peter, Carnegie-Mellon Univ.	UDP	Exploits the relation between initial gap and bottleneck gap	2003
	Netest	Jin and Tierney, UC Berkeley	UDP	Monitors maximum burst size	2002
	Pathchirp	Ribeiro, *et al*., Rice Univ.	UDP	Uses self-induced congestion to locate the range of A_P	2002
	Pathload	Jain and Dovrolis, Georgia Tech.	UDP	Self-Loading periodic streams, heuristic inference	2001
	Spruce	Strauss, *et al*., MIT	UDP	Probe gap model, single bottleneck, and pre-known C_P	2004
	TOPP	Melander, *et al*., Uppsala Univ., Sweden	UDP	Trains of packet pairs, segmented regression	2005

* All systems implicitly use IP and lower layer protocols.

1.3 工作原理及假设

研究发现,上述 13 个系统都采用了探测-估算,再探测-再估算这样一种不断求精的主动度量方法,并且都基于本质相同的工作原理.简单地讲,即 R_0 以传输速率 x 向 R_n 发送一组探测数据包,如果 $x>A_P$,那么必定至少存在一个链路,它会以低于 x 的速率传输这组数据包;相反地,如果 $x\le A_P$,那么这组数据包到达 R_n 时的传输速率仍然是 x.通过发送不同速率的探测数据包,并监测数据包传输速率的变化,度量系统就能推断出探测速率与可用带宽的关系,进而估算出可用带宽(或其所属范围).下面展开来讨论本工作原理.

设有 m 个大小为 s 的探测数据包,被 R_0 以速率 x 发往 R_n.相邻两个数据包被 R_i 发送入 L_i 的平均时间间隔(也即相邻两数据包在 L_i 上传输的平均时间间隔)为 p_i,$0\le i\le n-1$.于是,这组数据包在链路 L_i 上的传输速率 μ_i 为

$$\mu_i=\frac{s}{p_i} \tag{7}$$

易知 $x=\mu_0$.当这组探测数据包通过 R_i 从链路 L_{i-1} 传送到 L_i 时,将面临两种情况:

第 1 种情况:$A_i<\mu_{i-1}$ $(1\le i\le n-1)$.因为 $A_i<\mu_{i-1}$ 即 $C_i-\lambda_i<\mu_{i-1}$,于是 $C_i<\mu_{i-1}+\lambda_i$.通过 R_i 进入 L_i 的所有数据包的速率之和已经超过了 L_i 的带宽容量 C_i,所以,R_i 无法将所有进来的数据包(背景数据包加上探测数据包)都以它们进入 R_i 时的速率发送到 L_i 中.在这种情况下,R_i 将缓存部分数据包等待稍后再发送.这意味着在 p_{i-1} 时间内,R_i

将不能发送 1 个以上的探测数据包,于是,相邻两探测数据包的间隔被 R_i(或 L_i)拉大了,即 $p_i>p_{i-1}$.根据公式(7),因为数据包大小 s 不变,于是$\mu_i<\mu_{i-1}$.

在一些简化的网络模型中[5],p_i 和 p_{i-1} 的关系还可以进一步量化为

$$p_i=\frac{\mu_{i-1}+\lambda_i}{C_i}\cdot p_{i-1} \tag{8}$$

但是在实际的网络环境中,公式(8)只能近似(而无法准确)地描述探测数据包间隔的变化[18].由于本文主要关注度量系统的基本问题,考虑的是对上述 13 个系统都适用的分析,所以在下面的讨论中,仅认为当 $A_i<\mu_{i-1}$ 时,有 $p_i>p_{i-1}$.

第 2 种情况:$A_i\geq\mu_{i-1}(1\leq i\leq n-1)$.因为 $A_i\geq\mu_{i-1}$ 即 $C_i-\lambda_i\geq\mu_{i-1}$,于是 $C_i\geq\mu_{i-1}+\lambda_i$.这说明 L_i 有能力在不减速的情况下传输所有数据包,于是,相邻两个探测数据包的间隔保持不变($p_i=p_{i-1}$),且探测数据包的传输速率也不变($\mu_i=\mu_{i-1}$).

现以图 2 为例说明上述两种情况.图中所有背景数据流都来自 R_C,并流向 R_3.源主机 R_0 以μ_0=3Mbps 的速率向目标主机 R_3 发送一组大小相同的探测数据包.因为 $A_1<\mu_0$ 属于上述分析的第 1 种情况,即探测数据包在 L_0 上的传输速率高于 L_1 的可用带宽,所以,数据包间隔被 R_1 扩大了($p_1>p_0$).接下来的 $A_2\geq\mu_1$ 属于第 2 种情况,所以,R_2 能够不减速地传输所有进来的探测数据包,结果是 $p_2=p_1$.图 2 参照了 NS2[19]的图形显示原则,其中重要的一点是数据包的显示宽度与所在链路的带宽容量成反比.

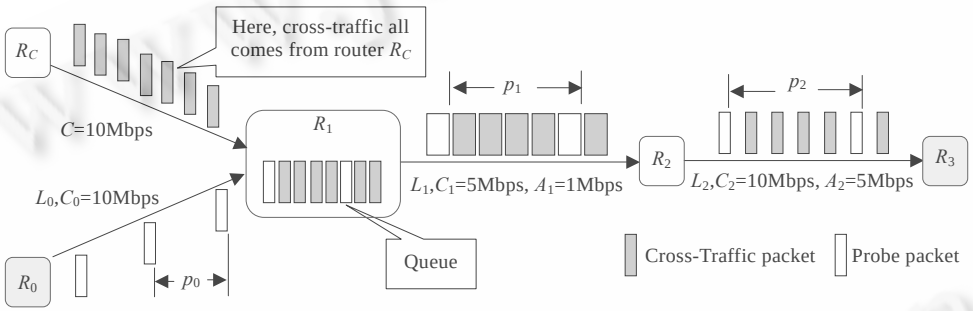


Fig.2 A typical probing process

图 2 一次典型的探测过程

在基于上述工作原理的同时,13 个度量系统还都(显式或隐式地)假设下面 4 点成立:

(1) 所有中间节点都采用先进先出(first-in-first-out,简称 FIFO)的缓冲队列管理算法;

(2) 背景数据流满足弗洛伊德(Fluid)模型,即数据包能够被拆分放入无限小的间隙中[14];

(3) 背景数据流的传输速率基本不变(或变化缓慢,以至于在一次度量周期内可以被认为是恒定的);

(4) 源主机(R_0)能够以高于可用带宽的速率发送探测数据包.

1.4 相关工作

Strauss 等人的研究表明,度量系统输出的结果总是存在难以消除的不确定性,这正是导致度量系统无法被广泛应用和推广的首要原因[14].Jin 等人首次证明了主机的处理能力是影响度量结果准确性的一个重要因素[20].随后,Jain 等人进一步细化了度量可用带宽的场景,并列举出度量过程中常见的 10 个概念误区[21].Lakshminarayanan 等人以实验数据说明,现有的度量系统大多只适用于传统网络环境[22],当系统被用于 802.11 无线网络环境(或有线与无线并存的混合网络环境)时,度量结果误差很大且无明显规律.这些工作为设计和开发准确、高效的度量系统提供了新的启示.不足的是,他们都只关注于度量过程的某些方面或仅针对一些具体的应用场景,而未能全面地考虑度量系统中普遍存在的基本问题.本文基于文献[23]的工作,从度量系统的工作原理、探测数据包在正反向路径上的传输过程、背景数据流和动态路由对度量过程的影响、各类实验平台上的大规模实验及相关数据分析等多个方面扩展了内容,更加深入地研究了度量系统的基本问题.

2 单、双终端系统共有的基本问题

下面,我们将度量系统中存在的基本问题分为 3 类:单、双终端系统共有的问题,单终端系统独有的问题以及双终端系统独有的问题.本节讨论单、双终端系统中都存在的 5 个基本问题:计时误差、低接入带宽、数据包异常、路由器队列管理、结果验证.

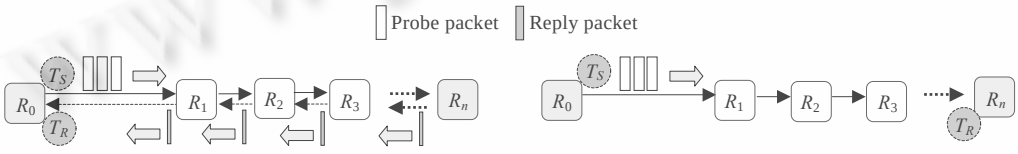
2.1 计时误差

度量结果的准确性在很大程度上取决于系统计时的精确度.几乎每一个度量系统都要求精确地控制数据包的发送时间,并准确获知数据包的接收时间.虽然单、双终端系统的计时过程稍有区别,但都存在计时误差的问题.在单终端系统中,数据包发送时间和接收时间的记录与维护全部由 R_0 负责,并且 R_0 所接收的数据包不是探测数据包,而是待测路径的中间节点 $R_1,\ldots,R_{n-1}$ 或目标主机 R_n 发送给 R_0 的应答数据包(reply packet).双终端系统由 R_0 控制每个探测数据包在何时发送,并由 R_n 记录探测数据包于何时接收.度量系统所有的后续计算(包括对可用带宽的评估),都基于系统在每次度量周期中维护的发送-接收时间集合 T_S 和 T_R.

$$T_S=\{s_1,s_2,\ldots,s_p\},\ s_i \text{ 表示第 } i \text{ 个指定数据包被发送的时间.}$$

$$T_R=\{r_1,r_2,\ldots,r_q\},\ r_i \text{ 表示第 } i \text{ 个指定数据包被接收的时间.}$$

其中,p 与 q 分别为一次探测中需要记录时间的数据包发送事件和数据包接收事件的数量.计时过程如图 3 所示,在单终端系统中,集合 T_S 和 T_R 都由 R_0 维护;在双终端系统中,集合 T_S 和 T_R 分别由 R_0 与 R_n 维护.



(a) Single end-host system (b) Double end-host system

(a) 单终端系统 (b) 双终端系统

Fig.3 The send-receive timing processes of a measurement system

图 3 度量系统的发送-接收计时过程

为了说明计时误差,设 t 是终端主机将数据包中的所有字节通过网卡放入物理传输媒介的时刻(或者是主机将数据包中的所有字节通过网卡从物理媒介上完整接收进来的时刻),并设 t_e 是系统所记录的它完成这一操作的时刻,则计时误差 ε 可以表示为

$$\varepsilon=|t-t_e| \tag{9}$$

计时误差的存在,意味着系统所记录的数据包发送(或接收)时刻其实并不是该数据包实际被发送(或接收)的时刻.计时误差 ε 越大,度量结果就越不准确.

计时误差不仅在每个度量系统中都存在,而且非常难以预测.数据包发送不及时是造成计时误差的一大原因.通常认为,让 R_0 在某个指定时刻发送出数据包,最好的方法是将度量系统的发送线程一直睡眠到那个时刻.可问题是,目前的主流操作系统(如 Windows 和 Linux)都只能提供毫秒级的计时器(timer)[24],这对于需要精确计时的度量系统而言是不够的.例如,为了获得 20Mbps 的探测速率,根据公式(7),度量系统计划每隔 200μs 发送一个 500 字节的 IP 包,为此,它在相邻发送操作之间调用睡眠函数 sleep 将发送线程休眠 200μs.为简化分析,假设发送操作所耗的时间可以忽略不计,且该度量系统在每次探测中仅发送一组共两个 IP 包.系统在某次探测中的实际发包过程如图 4 所示.由于计时器只能提供 1ms 的精度,在某次实验中,操作系统让发送线程实际睡眠了 700μs,这导致 ε=500μs 并且真正的探测速率仅有 5.7Mbps,这与期望的 20Mbps 相差相当大.

除了调用睡眠函数以外,在指定时刻准时发送数据包的另一种方法是使用不间断循环(non-blocking loop)[14],具体做法是,让发送线程一直保持活动状态(不断地检测当前时间,并执行一些无意义的计算指令),直到发送出数据包才释放 CPU(如图 5(b)所示的示例代码).与睡眠线程的方法相比,不间断循环确实能够更及时地

发送出数据包,但它也存在两大不足:一是发送线程强行占用了大部分CPU时间,因而会影响系统中其他功能的执行.这一点在单终端系统中尤其明显,因为数据包的发送和接收都由 R_0 来完成,所以,采用不间断循环的数据包发送方式必然严重影响接收线程的计时精度,进而导致 T_R 严重失真.二是不间断循环并不能 100%地按时发送数据包,它可能会被难以预测的系统事件(比如线程切换)打断.

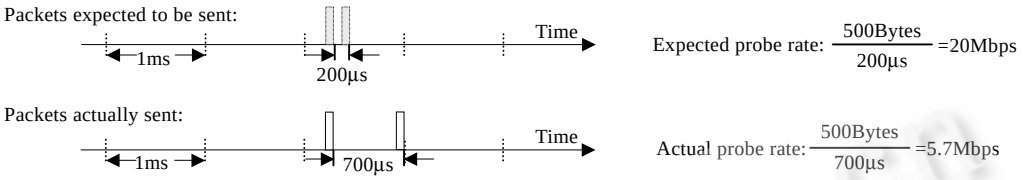


Fig.4 Timing error leads to unexpected probe rate

图 4 计时误差导致难以预测的探测速率

```
#1  gettimestamp(ts);   //get current time
#2  if (ts<expectedtime)
#3  {
#4     //sleep a while until the expected time
#5     sleep (expectedtime–ts);
#6  }
#7  sendto (Rn,packet);   //send out the packet
```

(a) Use sleep function to send a packet at specific time

(a) 使用 Sleep 方法在指定时刻发送数据包

```
#1  gettimestamp(ts);   //get current time
#2  while (ts<expectedtime)
#3  {
#4     //now it's early, just do something here
#5     x=random();
#6     x=(x+1)(x–1)/x^2;
#7     gettimestamp(ts);
#8  }
#9  sendto (Rn,packet);   //send out the packet
```

(b) Use non-blocking loop to send a packet at specific time

(b) 使用非中断循环在指定时刻发送数据包

Fig.5 The example code of two packet-sending methods

图 5 两种数据包发送方式的示例代码

线程切换也是造成计时误差的一大原因.操作系统通过多线程技术来支持并发任务的执行,它不断地切换线程,从而确保每个线程都能获得一定的执行时间.度量系统的线程在运行过程中也不可避免地会被线程切换打断;当它被打断时,核心资源(如 CPU)将被其他线程占用.线程切换带来的可能后果是:该发送数据包的时候不能发送,或者该接收数据包的时候不去接收.在这种情况下,时间集合 T_S 和 T_R 就是断裂的[18],即它们包含了不必要的线程切换开销.尽管可以在度量结束之后,根据操作系统的指令审计来检测相邻指令间是否存在异常间隔,从而判断是否发生了线程切换.但即使如此,目前仍没有可行的办法能够恢复已经断裂的 T_S 和 T_R[24].

网卡采用的中断技术是一个容易被忽略的计时误差来源.目前,不少高速网卡(比如 SysKonnect 9843)都采用了中断合并技术(interrupt coalescence)[25].传统网卡每接收到一个数据包就会产生一个中断信号,让系统读取它缓存中的数据.采用了中断合并技术后,当网卡接收到一个数据包时,它生成一个中断信号但并不立即发送该信号,相反,它会推迟片刻才发送这个信号.这样做的目的是,期望在这片刻中,网卡能够接收到额外的数据包,而这些额外的数据包将不会产生新的中断.因此在有限的时间内,多个数据包可以共用一个中断信号,从而节省了网卡的资源开销.文献[20,25]中的实验证明了中断合并技术确实能够有效提高网卡的吞吐量.不过,也正是因为该项技术的存在,使得操作系统无法保证它能获知所有数据包被接收的真实时间.比如,两个数据包共用了同一个中断信号,那么,操作系统能够获知的只是前一个数据包被接收的时间(它触发了中断信号),而无法得知后一个数据包是何时被接收的.这个问题可以通过修改网卡的驱动程序从而读取网卡寄存器(NIC register)来解决.但是,网卡的驱动随着生产厂商及主要处理芯片的不同而有很大差别,因此,在一台主机中的修改无法有效地应用到其他主机上.另外,修改驱动程序需要极高的系统权限,存在很大的安全风险,所以这个方案并不可行.

系统调用延迟也是计时误差的重要来源.系统调用延迟是指从调用操作系统函数到获得该函数返回值(包括 null)的这段时间.度量系统往往忽略了执行系统调用所需要的时间,或简单地假设系统调用延迟为 0,即函数

被调用的时刻也就是该函数执行完毕的时刻.但事实上,系统调用明显是耗费时间的,不同的系统调用以及同一系统调用在不同软、硬件平台上(或不同负载情况下)的时间开销也会不同.表 2 列出了度量系统中常用的 3 个系统调用:获取系统当前时间(get timestamp)、读网络套接字(read socket)和写网络套接字(write socket).后两个系统调用是接收和发送数据包的核心操作.表 2 给出了系统调用的最小(min)、平均(ave)和最大(max)延迟.最小和最大延迟分别是指连续 50 个系统调用的延迟中的最小值和最大值.这里重点介绍平均延迟的测量方法.现连续调用共 51 次 gettimestamp 函数(事后确认其间没有发生线程切换),假设第 i 次调用该函数后获得的时间标签为 t_i,那么,gettimestamp 的平均延迟 d_T 为

$$d_T=(t_{51}-t_1)/50 \tag{10}$$

Table 2 The delay of system call

表 2 系统调用延迟

Operating system*	CPU	Get timestamp (μs)			Read socket (μs)			Write socket (μs)		
		min.	ave.	max.	min.	ave.	max.	min.	ave.	max.
Linux 2.4.1	Intel Xeon 2.4 GHz	0.8	0.9	1.0	28.7	31.4	33.3	27.6	30.7	32.2
Linux 2.4.1	Intel P4 2.0 GHz	0.8	1.0	1.2	29.5	32.9	34.6	28.9	32.6	34.8
Linux 2.4.1	AMD AthlonXP 2500+	0.7	0.8	0.9	28.9	31.1	33.4	29.2	31.6	33.5
FreeBSD 4.8	Intel P3 700 MHz	4.3	4.6	5.0	45.2	48.9	50.5	45.1	47.0	49.5
Solaris 2.8	Sparc 400 MHz	0.5	0.5	0.5	33.6	36.8	37.9	32.3	35.2	38.1
Mac OS X 10.2	Power G4 1.0 GHz	1.8	1.9	2.0	43.2	45.8	48.9	40.2	43.5	45.3
Windows XP SP2	Intel P4 2.8 GHz	0.9	1.0	1.1	38.8	40.1	42.4	35.4	38.6	40.7

* For the first six OSs, we invoke *gettimeofday*() to get timestamp; for Windows, we invoke *QueryPerformanceCounter*(). We invoke socket functions *recv*() and *send*() for read and write socket system calls, respectively.

获取其他系统调用延迟的方法与获取 d_T 稍有不同.在度量 write socket 的平均延迟 d_W 时,我们连续不间断地执行 50 次 write socket 操作,每次操作都向 socket 中写入 500 字节的内容.紧接着在第 1 次操作前以及第 50 次操作后,我们调用了 gettimestamp 函数,得到的时间标签分别为 t_1 和 t_2.于是有

$$d_W=(t_2-t_1-d_T)/50 \tag{11}$$

计算 read socket 平均延迟的方法与计算 d_W 的方法相同.需要说明的是,由于事先分配了足够大的缓存空间,并且预写入了足够多的数据,所以每次调用 read socket 操作时,socket 缓存中总是有至少 500 字节的数据可读.总的来讲,为了降低计时误差,我们需要整合并优化从度量软件、操作系统到底层硬件等所有涉及到的软、硬件资源.比如,Cprobe 采用 SGI 公司的 IRIX 操作系统和硬件体系并做大量优化后,宣称能达到 40ns 的时间粒度.但从我们的实验看来,要想在通用操作系统如 Windows 和 Linux 上开发出具有同等计时粒度的度量软件还非常困难.

2.2 低接入带宽

回顾第 1.3 节,度量系统正常运行的前提之一是它能够以高于可用带宽的速率发送探测数据包.然而,过低的接入带宽(C_0)将严重限制度量系统评估网络路径整体带宽情况的能力.例如,对一个通过 56Kbps 调制解调器接入互联网的主机而言,度量系统输出的结果始终不会高于 56Kbps(如图 6 所示).这并不是度量系统出错了,相反,它是正确的.

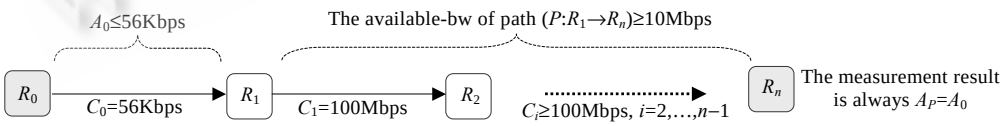


Fig.6 L_0 decides end-to-end available bandwidth when C_0 is comparatively low

图 6 接入带宽 C_0 过低时,可用带宽往往由 L_0 决定

根据公式(6),端到端可用带宽正是待测路径中最小的链路可用带宽.但问题是在低接入带宽的情况下,度量到的其实总是 A_0.这样一种度量结果往往不是用户想要的,此时,用户更希望了解其他部分网络路径的可用带

宽.低接入带宽问题是带宽度量研究中的一个富有挑战性的问题,尽管有些基于概率的方法(如 Monte Carlo[4])能够部分地解决这个问题,但至今没有相关的软件系统出现.

现在,大部分主机都能以 10Mbps,100Mbps,甚至 1 000Mbps 的速率访问互联网,这意味着在很多情况下,C_0 并不像图 6 中的 56Kbps 这么低.但是,源主机 R_0 作为待测路径中必须安装度量系统的终端节点,一般都处于网络分层体系的底部,它的接入带宽一般都低于上层 ISP 的带宽.从 2002 年起,大量 Tier-1 和 Tier-2 ISP 都开始基于 OC-192(10Gbps)和 OC-768(40Gbps)光纤链路来组建骨干网络[26].高速骨干网络带来的问题是,如果待测网络路径的大部分中间链路都属于高速率骨干网路,那么,度量系统很可能会由于接入带宽过低而无法有效地评估整个路径的带宽情况.

低接入带宽问题不仅取决于主机的网络配置,还取决于主机的数据包处理能力.在一个 1 000Mbps 的局域网中,我们简单地测试了 6 个不同的主机所能达到的最大传输速率.在实验中,每个系统连续不间断地发送出 3 000 个 1 500Bytes 的探测数据包,那么,传输速率就等于 4.5MBytes 除以发送所需的时间,测试结果见表 3.我们发现:

第一,主机的最大传输速率总是低于其网卡(NIC)的处理能力(这也是很自然的事情);

第二,网卡的利用率随着链路带宽的增长呈明显的递减趋势:当网卡是 10Mbps 时,其利用率高达 86%~89%;然而,当网卡为 1 000Mbps 时,其利用率只有 25%~34%.这意味着,受限于通用计算机系统的数据包处理能力,即使在配置了高带宽连接的情况下,主机也难以获得较高的传输速率.

Table 3 The maximum data-rate achieved by end-host

表 3 主机能达到的最大速率

Operating system	CPU	Achievable data-rate (Mbps)			
		NIC=10*	*NIC*=54	*NIC*=100	*NIC*=1000
Linux 2.4.1	Intel P4 2.0 GHz	8.8	38.3	77.2	323.0
Linux 2.4.1	AMD AthlonXP 2500+	8.9	39.1	78.3	340.2
FreeBSD 4.8	Intel P4 2.0 GHz	8.8	33.6	72.5	299.2
Solaris 2.8	Sparc 400 MHz	8.6	36.0	73.4	310.0
Mac OS X 10.2	Power G4 1.0 GHz	8.6	36.5	68.6	256.7
Windows XP SP2	Intel P4 2.8 GHz	8.8	36.3	70.6	280.2

* "*NIC*=*x*" means the NIC bandwidth is *x*Mb/s. The 10Mbps, 54Mbps, 100Mbps, and 1 000Mbps NICs are TP-LINK TE-2029 PCI, Intel Pro/Wireless 3945ABG, Realtek 8139, and SysKonnect 9843, respectively.

2.3 数据包异常

度量系统通过监控探测数据包来感知网络路径的带宽状况,因此很容易受到数据包异常(pathology)的干扰.数据包异常是指端到端数据传输过程中发生的异常事件,主要有包丢失(packet loss)、乱序传送(out-of-order delivery)、包重复(packet replication)和包破坏(packet corruption).在很多情况下,数据包异常会导致 T_R 中包含不合理的数据,进而影响度量过程的准确性.举例来说,如果探测数据包在传输途中被某个路由器给丢弃了(包丢失),度量系统就无法维持完整的时间集合 T_R;同样,如果数据包在途中被乱序传送,那么,根据 T_R 所推算出来的数据包间隔将出现难以理解的负值;另外,当包重复出现时,假设有两个完全一样的探测包抵达 R_n,系统将无法确认需要向 T_R 中放入哪一个探测包的接收时间(只能且必须放一个);最后,如果一个探测数据包在途中被意外地破坏了,那么,系统很可能由于无法识别它的特征信息而将它误认为是背景数据包,这同样会影响到系统对 T_R 的维护.

早在 1994~1995 年间,Paxson 就在分布于世界各地的 37 台主机上安装了 network probing daemon(NPD)程序,并在大规模网络实验的基础上统计了数据包异常现象发生的概率[18].2005 年 10 月,我们在 8 台主机联结而成的覆盖型网络(overlay network)上再次运行了 NPD,分两次(第 1 次 5 000 个,第 2 次 8 000 个)共收集了约 13 000 个 TCP 连接的记录(trace),并用 Paxson 的方法[18]统计了数据包异常的概率.表 4 给出了这 8 台主机所在机构的名称以及主机之间的跳转数(hop count),跳转数由 traceroute[27]获得.比如从 IOS_1 到 $GUCAS_2$ 的 IP 包要经过 11 个跳转,这说明从 IOS_1 到 $GUCAS_2$ 的网络路径共包含 11 个链路.部分主机(如 Telecom)不生成 ICMP 包,

所以到这些主机的路径无法用 traceroute 获知跳转数(用?标出).此外,两台主机之间往返路径的跳转数可能是不一样的(用斜体标出).

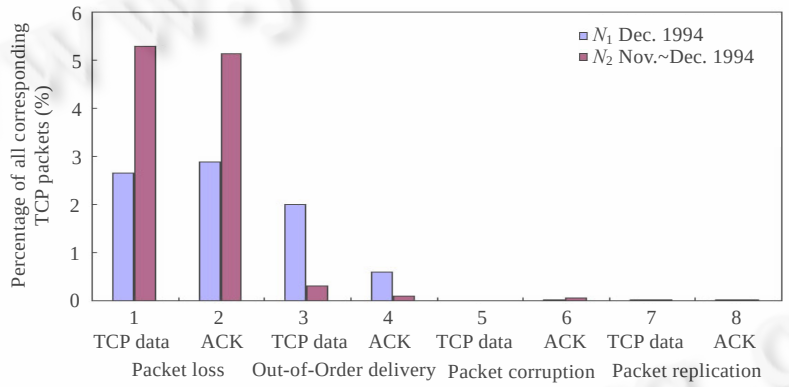
Table 4 The hop count between any two nodes of the overlay experimental network

表 4 覆盖型实验网络中任意两节点间的 IP 跳转数

	IOS$_1$	*IOS*$_2$	*GUCAS*$_1$	*GUCAS*$_2$	Tsinghua	PKU	USTC	Telecom
IOS$_1$	–	3	10	11	14	14	16	?
IOS$_2$	3	–	11	12	15	15	17	?
GUCAS$_1$	10	11	–	2	12	12	16	?
GUCAS$_2$	12	12	2	–	13	13	16	?
Tsinghua	14	15	12	?	–	13	16	?
PKU	?	?	?	?	13	–	16	?
USTC	16	17	15	16	16	?	–	?
Telecom	15	17	14	15	14	15	14	–

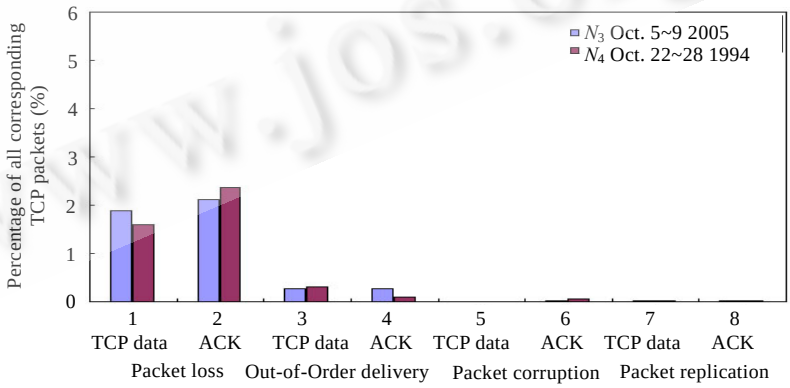
The meaning of symbols is as follows. –: no connection to itself; ?: hop count unknown; italic font: the forward & reverse paths between a pair of nodes comprise different number of routers. Specifically, IOS, GUCAS, Tsinghua, PKU, USTC, Telecom are Institute of Software, Graduate University of the Chinese Academy of Sciences, Tsinghua University, Peking University, University of Science and Technology of China, and Beijing Telecom, respectively.

图 7 给出了在 Paxson 的两个 TCP 连接记录集合(N_1,N_2)和我们的记录集合(N_3,N_4)上计算出的包异常概率.



(a) Pathologies measured by Paxson in 1994~1995

(a) Paxson 在 1994~1995 年间度量到的数据包传输异常



(b) Pathologies computed from our datasets in 2005

(b) 从 2005 年数据集中计算出来的数据包传输异常

Fig.7 The ratios of end-to-end packet pathologies

图 7 数据包异常事件发生的比率

集合 N_1,N_2,N_3 和 N_4 分别是在 1994 年 12 月、1995 年 11~12 月、2005 年 10 月初和 2005 年 10 月末收集的.需要注意的是,NPD 只能记录和分析 TCP 数据包(TCP data)与 TCP 应答包(ACK)的异常现象.但鉴于 TCP 是当今网络中最主要的通信协议,所以,基于 NPD 程序的分析基本上能够反映网络中数据包异常的真实情况.如图 7 所示,在所有的 TCP 数据包中的包丢失的概率分别为 2.7%(N_1),5.2%(N_2),1.9%(N_3)和 1.7%(N_4).不难看出,在这 4 个集合中,上述 4 种包异常现象都是存在的,其中以包丢失和乱序传送最为突出.

很难从根本上完全消除数据包异常对可用带宽度量过程的影响.根据第 1.3 节中介绍的工作原理,度量系统需要不断发送大量的探测数据包,并推断探测速率与可用带宽的关系,从而估算可用带宽.同时,度量系统还要求源主机的发送速率能够大于可用带宽,这意味着度量系统的主动探测将导致局部的网络拥塞,即至少 1 个路由器的缓冲队列会增大.当发生网络拥塞时,上述 4 种数据包异常现象(特别是包丢失)就有可能发生[18].关键的问题是,如果度量系统所发送的探测数据包引发了数据包异常现象,而这些现象又与待测路径本身就存在的数据包异常现象重叠在一起,那么,从终端主机(R_0 和 R_n)的角度要想准确获知造成包异常的原因,并正确判断探测速率与可用带宽之间的关系是非常困难的.

2.4 路由器队列管理

尽管度量系统都(显式或隐式地)假设中间路由器采用 FIFO 缓冲队列管理算法,但事实却并非如此[28].例如,已有相当多的路由器采用了主动队列管理(active queue management,简称 AQM)算法[29].与 FIFO 相比,AQM 的设计思想是尽可能提前避免网络拥塞,而不仅仅是顺序处理进入路由器的数据包.当数据链路即将拥塞时(链路的利用率高于某个预定的阈值,如 80%),AQM 路由器会主动丢弃部分数据包以降低拥塞的可能性.

但是,从带宽度量的角度来讲,AQM 将导致三大问题:第一,当探测数据包被丢弃时,网络拥塞可能只是将要发生而不是一定发生,所以当数据包被 AQM 路由器丢弃后,就推断拥塞已经发生(或认为探测速率高于可用带宽)是不准确的;第二,终端主机 R_0 和 R_n 无法预知 AQM 会导致多大的丢包率,也无法将 AQM 导致的丢包与节点或链路出错导致的丢包区分开来;第三,当探测数据流给待测路径带来局部拥塞时,AQM 路由器可能会丢弃路径中正常传输的数据流(即背景数据流).这违背了度量系统的设计初衷,即度量是为了更好地传输数据,而不是影响或损害背景数据流的传输.总之,如果瓶颈链路 L_i 的入口节点 R_i 采用的是 AQM 而不是 FIFO,那么,度量系统将难以通过观察探测数据包传输情况的方法来准确判断探测速率与可用带宽的关系.

然而,即使所有节点都采用 FIFO,在某些情况下,网络拥塞也会导致数据包间隔的异常变化,从而掩盖探测速率与可用带宽的真实关系.比较典型的是,尽管探测速率小于可用带宽,但是相邻数据包的间隔却被压缩了,而不是像第 1.3 节中分析的那样保持不变.如图 8 所示,R_0 向 R_3 发送两个探测数据包,这两个探测数据包在 3 个背景数据包之后进入 L_1,5 个数据包都将被 R_2 发送到 L_2 上.当第 1 个探测数据包进入 R_2 时,由于 R_2 还没有将 3 个背景数据包都发送出去,所以,这个探测数据包被缓存在 R_2 中,直到 3 个背景数据包都进入 L_2 后才被发送.当第 2 个探测数据包进入 R_2 时,由于前面的数据包都已经被处理了,因此,该数据包被 R_2 直接发送入 L_2.最后的结果是当 $A_2 \geq \mu_1$ 时 $p_2<p_1$,即数据包间隔被压缩了.

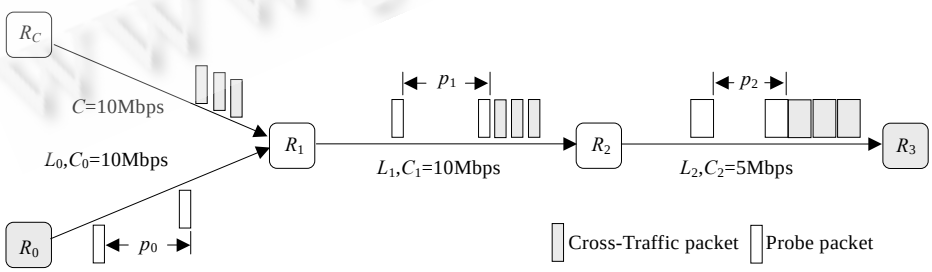


Fig.8 The compression of probe packet dispersion ($p_0=p_1>p_2$)

图 8 被压缩的数据包间隔($p_0=p_1>p_2$)