Accurate and Low-cost Scheme for Network Path Delay Measurement

Shiqiang Chen
College of Computer Science, Sichuan University, Chengdu, China
Hubei University for Nationalities, Enshi, China
Email: chensq8808@gmail.com

Junfeng Wang
College of Computer Science, Sichuan University, Chengdu, China
Email: wangjf@scu.edu.cn

Yifang Qin and Xu Zhou
Institute of Acoustics, Chinese Academy of Sciences, Beijing, China
Email: {qinyf126, zhoux_hpnl}@126.com

***Abstract*—Path delay between two end-hosts not only is one of the key metric for evaluating the performance of a communication network, but also plays a significant role in several overlay network construction protocols, peer-to-peer (P2P) applications, etc. A popular example is P2P networks which need to build an overlay between peers in a way that minimizes the message exchange delay among the peers. Moreover, the clock error and the location error are the main factors to influence the measurement accuracy in the network path delay measurement. In this paper, we present a new scheme for accurate path delay measurement. In the scheme, Khlifi's clock synchronization algorithm is used to eliminate the clock error between two end-hosts, and a kernel network driver (an intermediate driver based on Windows Network Driver Interface Specification) is utilized to eliminate the location error. The experimental results show that the new scheme can practically improve the measurement accuracy of network path delay, and make sure the measurement standard deviation to be lowered 10μs under different packet lengths and host loads. This scheme doesn't need additional softwares and hardwares, so it has lower cost and can be applied widely.**

***Index Terms*—network path delay, active measurement, measurement probe, clock error, location error, network kernel driver**

I. Introduction

In IP networks, delay is the key metric for evaluating the network performance. One-way delay (OWD) is the foundation for measuring network bandwidth, delay variation, packet loss rate, etc. Round-trip delay (RTD), i.e., path round-trip time (RTT), plays a crucial role in all reliable network transport protocols, peer-to-peer (P2P) applications, and proximity-based server redirection in such applications as content distribution networks and multi-player games. The IETF IP Performance Metrics (IPPM) Working Group has developed a set of standard metrics that can be applied to the quality, performance, and reliability of network data delivery services. OWD and RTD metrics of packets across Internet paths were defined respectively in rfc2679 and rfc2681 [1, 2]. Unfortunately, the OWD estimation is a difficult problem between end-hosts. The OWD value cannot be calculated accurately, because the clocks between two end-hosts are not synchronized (in this paper, the clock difference between two hosts is referred the clock error). Furthermore, in heterogeneous and massive networks, such as the Internet, it is even more difficult to guarantee the synchronized clock.

In order to reduce the clock error as far as possible, a number of researchers have done much work in the area of clock synchronization, such as J. H Choi [3], M. Aoki, E. Oki [4], D. Kim, J. Lee [5]. According to the way how they acquire synchronization between two clocks, the clock synchronization algorithms can be categorized into two basic types, which are end-to-end measurement based schemes and external source based schemes [3]. Typically, external source based schemes are the synchronization methods using centralized clock sources such as Network Time Protocol (NTP), Global Positioning System (GPS) and IEEE 1588. End-to-end schemes get synchronization information through active network measurements between two end hosts.

In addition to the clock error, the location error on the end-host is another significant factor which affects the measurement accuracy. What is the measurement location error? It means that the sending or receiving timestamp which marked by measuring application program is not the true moment when a probe packet departs from or arrives at the Network Interface Card (NIC) of an end-host. In other words, the location error is caused by the end-hosts. Therefore, the measurement results which contain the location error cannot represent the actual performance of the end-to-end path. In fact, the factors affecting the location error mainly include the hardware performance of an end host, the kernel complexity of an operating system, system load, interrupt response, kernel spin lock, and process/thread scheduling. So the location

doi:10.4304/jsw.8.6.1398-1404

error can reach dozens to hundreds of milliseconds. Especially, the big traffic flows lead to the end-host kernel appears the extreme case of interrupt-live-lock (ILL), which would be up to hundreds of milliseconds and even infinite [6, 7]. Therefore, an accurate path delay measurement system should reduce the clock error and location error as far as possible.

In this paper, a new delay measurement framework on Windows platform is presented. In our scheme, Khlifi's algorithm is adopted for clock synchronization between two end-hosts, and a kernel network driver is exploited and used to eliminate the position error. In addition, the User Datagram Protocol (UDP) is utilized to carry out the network path delay measurements. Our work makes several contributions as follows:

- An accurate timestamp collection driver (named *ATC-Driver* in this paper) based on Windows Network Driver Interface Specification (NDIS) is successfully exploited. By installing this driver on end-host, the collection position of probe time-stamp can be moved from the user application program to the data link layer of TCP/IP stack. The measurement location error can be basically eliminated based on this new method. To the author's knowledge, this work describes the first timestamp marked technology specifically tailored for the end-to-end latency measurement.
- We present a novel solution which substantially simplifies the development of accurate and low-cost path delay measuring application based on *ATC-Driver*. The experimental results show that it can be adapted to parallel and concurrent delay measurement tasks.
- The scheme uses the active network measurement technology, and is concurrently suitable for forward one-way delay (FOWD), reverse one-way delay (ROWD) and RTD measurements between two end-hosts.

The remainder of this paper is organized as follows. In Section II, some related work in this field of network delay measurement is briefly introduced. In Section III, the solution of accurate and low-cost delay measurement is presented. Experimental results and discussion are given in Section IV. The final section ends the paper with some concluding remarks.

II. Related work

During the past few years, many researchers have devoted their efforts on the issue of network delay measurement and therefore many valuable contributions are presented in the literatures. In 1983, Mike Muuss exploited the famous utility *ping* to diagnose the network connectivity from the path round-trip delay [8]. The *ping* utilizes the Echo request/reply mechanism defined in the Internet Control Message Protocol (ICMP) to carry out the path delay and connectivity measurement [9]. In [10], the authors proposed the method based on multicast protocol to estimate the network delay. In addition, other *ping-like* tools emerge, such as *TCP-ping* which employs the SYN/ACK mechanism in hand-shaking procedure of Transmission Control Protocol (TCP) [11, 12]. Above these measuring methods are typical active measurement technologies, which inject extra traffic into measured networks. Measuring delay passively doesn't generate additional traffic into measured network, but need to keep track the departure and arrival times for all probe packets at the measurement endpoints, and need to communicate with each other to correlate the timestamps for network delay calculation [13]. For this reason, passive measuring method for OWD is rarely deployed [14].

In order to measure path OWD, the clocks of two end-hosts must keep synchronized. A lot of researchers have focused on the clock synchronization algorithms with end-to-end measurement based schemes. Most of synchronization algorithms have focused on the detection and estimation of clock skew existing in the unidirectional path. Moon *et al*. focused on filtering out the effects of clock skew only with the unidirectional delay measurement to determine the variable portion of the delay [17]. Khlifi *et al*. proposed two offline clock skew estimation and removal algorithms [18]. In [19], L. Zhang *et al*. used their convex hull based algorithm to remove the skew from online delay. Above algorithms can be classified into two sub-groups according to their real-time applicability, i.e., offline method and online synchronization method.

Although there were many algorithms and tools for delay estimation, all or almost all of them focused on the design process of clock synchronization algorithms, i.e., many of them attempted to eliminate the clock error. To the best of our knowledge, few researchers have focused on the issue of eliminating the location error which is another key factor affecting the measurement accuracy. In [20], the authors presented a high performance data acquisition and generation (DAG) card which can capture network traffic with high accurate timestamps. But this card is too expensive and only suitable for the passive measurement mode. Berkeley Packet Filter (BPF) can capture the packet at the data link layer, and a timestamp can be returned to the application by *libpcap API* [21]. Similarly, this method is also only suitable for the passive measurement mode.

In this paper, our solutions not only pay attention to eliminate the clock error, but also in particular concerned about the elimination of the location error based on kernel driver technology.

III. Delay measurement solutions

A. Scheme Overview

Our goal is to exploit an accurate and low-cost delay measurement program, and to facilitate FOWD, ROWD, and RTD (i.e., RTT) measurements. The novel scheme of the network path delay measurement is shown in Figure 1. The solutions are presented as follows:

In order to measure FOWD and ROWD, the clocks of two end-hosts must keep synchronized each other. If the two clocks are perfectly synchronized, the OWD result can be calculated by subtracting the send timestamp from the receive timestamp and this value will be exactly the

true delay between the two end-hosts. Unfortunately, two clocks are rarely perfectly synchronized in the real systems. In this paper, an off-the-shelf, accurate and low-complexity algorithm is employed to eliminate the clock error from online delay measurements (this algorithm mainly removes the clock skew between two end-hosts): the *combined algorithm* [18], which is a mixed approach of the *sliding window mechanism* (SWM) and the *convex hull algorithm* (CHA). The SWM can quickly respond to skew effect, but it is less accurate than the CHA when long intervals are used. The *combined algorithm* also uses an exponential smoothing approach to improve the *convex hull* estimation of the current skew. The estimate of the skew at the end of the interval i, i.e., $\hat{\alpha}_i$ is defined as follows:

$$\hat{\alpha}_i = \omega \times \hat{\alpha}_{i-1} + (1-\omega) \times \alpha_i \quad (1)$$

where, $\hat{\alpha}_{i-1}$ denotes the estimate of the skew at the end of the interval $i-1$, α_i denotes the output of the CHA for the interval i, and ω denotes the weighting factor ($0 < \omega < 1$). This approach is of complexity $O(1)$.

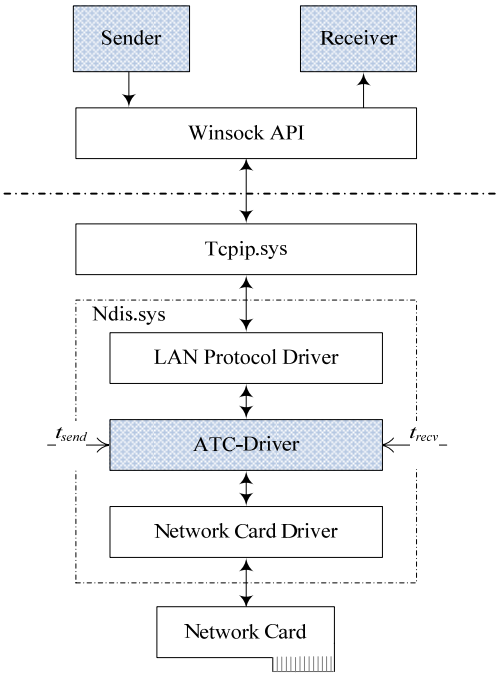

Figure 1. Scheme Overview

Currently, most network performance measurement tools, such as *ping*, *Iperl* and *Netperf* which were developed with user-level *C/C++* code written based on socket libraries and the timestamps are marked in user applications. Because of the process/thread context swap, the end host load, the interrupt response and many other factors, the timestamps suffer from an uncertain deviation from the expected wire departure/arrival times. Thus, the measurement results will contain the location error which reaches maybe from dozens to hundreds of milliseconds. In Figure 1, when the measurement probe application, for instance, the sender, sends a probe packet, the timestamp (denoted by t_{send_raw}), which marked by the sender, is not the true moment (denoted by t_{send}), which marked by the *ATC-Driver*. So the sending location error, denoted by Δt_{send}, can be described as follows:

$$\Delta t_{send} = t_{send} - t_{send\_raw}$$

where $t_{send} > t_{send\_raw}$. Similarly, the receiving location error, denoted by Δt_{recv}, is calculated as follows:

$$\Delta t_{recv} = t_{recv\_raw} - t_{recv}$$

where $t_{recv\_raw} > t_{recv}$. So the location error of a one-way path is presented as:

$$err_{loc} = \Delta t_{send} + \Delta t_{recv} \quad (2)$$

Therefore, the end-to-end OWD (denoted by owd_{raw}) which measured by the traditional method (i.e., The timestamps are marked in user applications), can be described as follows:

$$owd_{raw} = \sum_{i=1}^{n} q_i + \sum_{i=1}^{n} t_i + p_{const} + err_{lock} + err_{loc} \quad (3)$$

where, q_i and t_i ($i = 1,2,...,n$), denote the probe packet queuing delay and the transmission delay of the i^{th} router respectively, and t_i can be calculated by the formula of $Packet\_size / Capacity_{link}^{(i)}$. A constant propagation delay is denoted by p_{const}. The clock error and the location error of the one-way path are respectively denoted by err_{clock} and err_{loc}. Consequently, it can be seen that the owd_{raw} not only contains the clock error, but also may contain a large location error.

As the operating system kernel can provide high accuracy timestamp than user level, the mark position of probe timestamp can be moved from the user space to the kernel driver. To avoid the influence of the location error as much as possible, an accurate timestamp collection driver, which referred to as *ATC-Driver* is developed based on Windows Network Driver Interface Specification (NDIS). By installing this driver on end-host, the collection position of probe timestamp will be moved from the application program to data link layer of TCP/IP stack. In the section IV, the experimental results show that this method can basically eliminate the measurement location error.

As shown in Figure 1, the *Sender* and the *Receiver*, respectively, are the sending thread and the receiving thread inside the measuring probe, which is developed with standard *C++* code written based on Winsock library (i.e., *ws2_32.dll*). In this paper, the measuring probe (named *ATC-Measurer*) utilizes the protocol of UDP to carry out the end-to-end delay measurements. The *Receiver* receives measurement probe packets which are sent by the *Sender*, and reads the timestamps from these probe packets. These timestamps are marked and immediately written into the probe packets by the *ATC-Driver*. The advantage of this design is that the three types of measurement results, i.e., the FOWD, the ROWD and the RTT, can be obtained after one round-trip measuring process is accomplished.

B. ATC-Driver

To effectively reduce the measurement location error, the main idea is that we manipulate the position of probe

packet timestamps marked as close as possible to the host's NIC. On Windows NT 4.0 or later system, fortunately, the NDIS specification provides us with the possibility to achieve the innovation.

NDIS: is implemented by a Windows kernel file called *Ndis.sys*[1], and is a standard application programming interface (API) for the NICs. It allows computers to be connected to IP network with different communication protocols such as *TCP/IP*, *IPX*, *AppleTalk*, *NetBIOS*, etc. It supports three types of network drivers, i.e., the protocol driver, the intermediate driver, and the miniport driver (i.e., NIC driver). In fact, the intermediate drivers sit in-between the Media Access Controller (MAC) and Internet Protocol (IP) layers and can control all traffic being accepted by the NIC card.

In our solutions, the accurate timestamp collection driver, i.e., *ATC-Driver*, is an intermediate driver, and implements both miniport and protocol interfaces (i.e., *ProtocolXxx* and *MiniportXxx* functions). The NIC driver and protocol driver actually communicate with the corresponding protocol and miniport interfaces that reside in the *ATC-Driver*. When a communication packet arrives at the *ATC-Driver*, it should be checked by the *ATC-Driver* as a probe packet or not according to the port rules. If it is not a probe packet, it will directly be sent to the NIC driver or the protocol driver. On the contrary, the *ATC-Driver* will immediately generate a timestamp, and write the timestamp into the user data segment (UDS) of the probe packet. The IP probe packet structure is shown in Figure 2. The t_i $(i=1,2,3,4)$, respectively, is the measuring timestamp in one round-trip measuring process. When the UDS is constructed by the *Sender*, specifically, its first 128 bits are all set to 1 for simplifying the checksum recalculation.

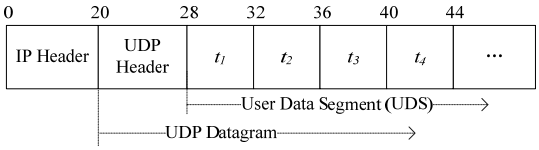

Figure 2. IP probe packet structure (46~1500 bytes)

NDIS defines a packet as a packet descriptor with a chain of one or more buffers containing the network packet data. The NDIS library describes the packet descriptor by defining the *NDIS_PACKET* structure. NDIS packet is allocated by a protocol driver, filled with data (i.e., The IP probe packet), and passed to the next lower NDIS driver so that the data can be sent on the network. When an NDIS packet (which contains an IP probe packet) is passed to the *ATC-Driver* from the protocol driver, or the lowest level NIC drivers which allocate an NDIS packet to hold received data and pass the packet up to the interested *ATC-Driver*, the *ATC-Driver* will immediately generate a timestamp. Therefore, the location error of delay measurement can be removed effectively. The procedure of removing the location error is described as follows:

[1]Ndis.sys is referred to as the NDIS wrapper, and defines the way protocols communicate with network adapters in Windows kernel.

Location error removal procedure

Inputs: *oPacket*: NDIS packet, *H_type*: end-host type, *bRecOrSend*: traffic direction

Begin

1. Initialize the environment and get the context.
2. Allocate a new packet descriptor, named *nPacket*, from the NDIS packets pool.
3. *nPacket* $\Leftarrow$ *oPacket*.
4. *pIPHeader* $\Leftarrow$ *Query_IP_Packet* (*nPacket*).
5. **if** *pIPHeader* $\rightarrow$ *protocol* equals to UDP **then** *pUDPHeader* $\Leftarrow$ (*pIPHeader*+1).
6. **else** goto step 10.
7. **if** *pUDPHeader* $\rightarrow$ *port* equals to rule_port **then**
   (1) Generate the timestamp t;
   (2) **if** *H_type* is the source host **then**
   if *bRecOrSend* is *dir_send* **then**
   Write t into the place *t*1 of the UDS;
   else
   Write t into the place *t*4 of the UDS;
   (3) **else**
   if *bRecOrSend* is *dir_recv* **then**
   Write t into the place *t*2 of the UDS;
   else
   Write t into the place *t*3 of the UDS;
   (4) Recalculate and update the checksum for the probe packet.
8. **if** *bRecOrSend* is *dir_send* **then** Sends *nPacket* to the NIC drive;
9. **else** Indicates *nPacket* to the protocol driver.
10. release *nPacket*.

End

Because the *ATC-Drive* modifies the UDS's contents with the timestamp information (see Figure 2), the checksum of the UDP datagram should be recalculated and updated. Here, we pay close attention to optimize the checksum algorithm to avoid the high overhead of recalculating a new checksum. Since UDP use a one's complement sum, it is sufficient to calculate the arithmetic difference between the before-modification and after-modification and add this to the checksum. When the time stamp information, such as t_i $(i=1,2,3,4)$, is inserted into the UDS, the checksum adjustment algorithm is described as follows:

Checksum adjustment algorithm

Inputs: *oChechsum*: old checksum, $t_i : t^{th}$ time-stamp

Output: *nChechsum*: new checksum

Begin

1. *oChechsum* $\Leftarrow \sim$ *oChechsum*. /* Bitwise complement operation*/
2. $t_i \Leftarrow \sim t_i$.
3. *oChechsum* $\Leftarrow$ *oChechsum* $\oplus$ $t_i[0]$. /* Binary addition operation; $t_i[0]$ is the low 16-bits part of t_i */

4. $oChechsum \Leftarrow oChechsum \oplus t_i[1]$. /* $t_i[1]$ is the high 16-bits part of t_i */
5. $nChechsum \Leftarrow \sim oChechsum$.

End

C. Measurement Application

The measurement application, i.e., the *ATC-Measurer* is associated with the *ATC-Driver*, and implements the following steps: (1) executive the clock synchronization process between the source end-host and destination end-host, (2) assign filtering rules to the *ATC-Driver*, (3) inject some probe packets into measured networks, (4) read the timestamp information from the probe-packets, and calculate the network path delay. After one round-trip delay measurement is accomplished, three types of delay values can be obtained, and described as follows:

$$\begin{cases} FOWD = t_2 - t_1 \\ ROWD = t_4 - t_3 \\ RTT = (t_2 - t_1) + (t_4 - t_3) = (t_4 - t_1) + (t_2 - t_3) \end{cases} \quad (4)$$

The round trip measurement processes of the *ATC-Measurer* are mainly described as follows:

- Initially, the *ATC-Measurer* executives the clock synchronization process between the source host (denoted as H_{sour}) and destination host (denoted as H_{dest}).
- The *Sender* (is the sending thread of the *ATC-Measurer;* see Figure 1), sends a UDP probe packet (named $Packet\_1$) to the H_{dest} from the H_{sour}, then the *ATC-Driver* can capture the packet, generate a timestamp t_1 at once, and insert it into the UDS of this packet.
- When $Packet\_1$ arriving at the H_{dest}, the *ATC-Driver* captures it, generates the timestamp t_2 rapidly, and inserts t_2 into the UDS of this packet.
- The *Receiver* fetches the $Packet\_1$, and sends it to the H_{sour} right away. Similarly, when this packet is transmitted back to the H_{sour}, the time stamps (i.e., t_3 and t_4) will also be inserted into the UDS.
- On the source-host side, the *ATC-Measurer* can get four time-stamps (t_1, t_2, t_3 and t_4), so the FOWD, ROWD, and RTT values can respectively be calculated by the Equation (4).

IV. Experimental Results and Analysis

A. Experiments Design

In order to evaluate the performance and accuracy of the *ATC-Measurer*, we designed a comparison program (named *U-Measurer*), which is not associated with the *ATC-Driver* but makes the timestamp information in its own process space. The experiments are completed with two general PCs which are linked by a shielded *CAT-5* twisted-pair network cable. The length of the cable is only 1 m. As the transmission delay of electrical signal equal to 1 ns on this cable, so the delay value can be ignored. These two PCs have the same configuration with *Pentium (R) Dual-Core E6700 3.2G CPU, 2GB memory, Intel PILA8460M-82551 1000Mbps Ethernet NIC*, and *Windows XP sp3 operating system*. We express them as PC_{sour} and PC_{dest} in this paper. The PC_{sour} denotes source host, and the PC_{dest} denotes destination host. The probe packet sizes (p_{size}) are typically 64 bytes, 512 bytes, and 1500 bytes respectively.

In addition, to avoid the interference of background traffic, other communications processes which were running at PC_{sour} and PC_{dest} were closed. The *ATC-Driver*, *ATC-Measurer* and *U-Measurer* were installed on PC_{sour} and PC_{dest} respectively. Those measurement results of path delay are calculated by Equation (4). Under these conditions, the FOWD, ROWD, and RTT meet the following criteria: FOWD=ROWD=RTT/2. For this reason, the value of RTT is used to evaluate the measurement accuracy and stability for the *ATC-Measurer*. In particular, our new solution can nearly eliminate the location error. In this paper, the evaluation parameters mainly include the maximum (Max), minimum (Min), mean (Mea) and standard deviation[2] (Std) based on the RTT measurement samples. Multiple sets of comparative experiments between the *ATC-Measurer* and *U-Measurer* were executed carefully. All experiments had been done from April.7, 2012 to April.15.

B. Light-load Measurement Evaluation

As a first test, the delay measurement accuracy was evaluated for the *ATC-Measurer* application compared the *U-Measurer* under the light-load conditions on PC_{sour} and $PC_{dest.}$ The detailed compounding of parameters are as follows: Case 1 set t_{out} to 1 s and p_{size} to 64 bytes; Case 2 set t_{out} to 1 s and p_{size} to 512 bytes; Case 3 set t_{out} to 1.5 s and p_{size} to 1500 bytes. The experiment results on *ATC-Measurer* and *U-Measurer* were persistently sampled 1000 numbers under different parameters cases, so six experimental result sets could be obtained. Table 1 lists the path RTT measurement results under three kinds of cases. Figure 3 displays that the time series of path round-trip delay under case 3.

TABLE I.

THE PATH RTT MEASUREMENT RESULTS UNDER THREE KINDS OF CASES

Case	*U-Measurer*				*ATC-Measurer*			
	Min	Mea	Max	Std	Min	Mea	Max	Std
1	169	203.7	237	28.1	161	165.3	201	7.4
2	241	280.2	319	23.9	233	241.1	247	5.2
3	322	348.9	395	22.7	252	256.8	265	4.8

As can be seen from Table 1, all the statistics of the path RTT measured by the *ATC-Measurer* are lower than those measured by the *U-Measurer*. For example, the mean RTT measured by the *ATC-Measurer* is 18.85%

[2] The formula of standard deviation is $\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2 / n}$, $\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$.

lower than that measured by the *U-Measurer* under case 1, 13.95% under case 2, and 26.40% under case 3. The RTT difference between maximum and minimum is 41.2% under case 1, 82.1% under case 2, and 82.2% under case 3. In addition, the results measured by the *ATC-Measurer* are more stable than that measured by the *U-Measurer* under all kinds of cases. For example, the difference of standard deviation is 73.7% when measured under case 1, 78.2% when measured under case 2, and 78.6% when measured under case 3. Furthermore, from Figure 3, we also noticed that the results measured under case 3 have particular difference. The testing reveals that the *ATC-Measurer* application based on the *ATC-Driver* can effectively reduce the location error, while it can improve and guarantee the accuracy of path delay measurement.

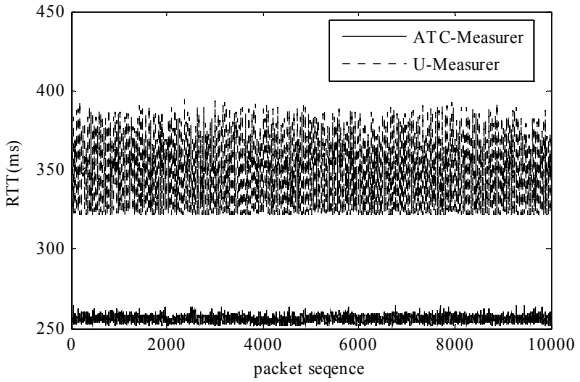

Figure 3. The time series of path round-trip delay under case 3

C. High-load Measurement Evaluation

In large-scale parallel measurement applications, one host usually provides network measurement services for the dozens or hundreds of target hosts. When the measurement host is running with a high load, the frequent scheduling of processes and threads may have a large effect on the user space application of network delay measurement. In order to evaluate the accuracy and stability of the *ATC-Measurer* for the parallel and concurrent delay measurements, we measured the path RTT using the *ATC-Measurer* compared the *U-Measurer* between the PC_{sour} and PC_{dest} in the high-load conditions (the CPUs utilization all are above 90%, and the probe packet size is equal to 1500 bytes, expressed as case 4). Table 2 lists the path RTT measurement results, and Figure 4 displays that the time series of path round-trip delay.

As can be seen from Table 2, all the statistics of the path RTT measured by the *ATC-Measurer* are also lower than those measured by the *U-Measurer*. The standard deviation of the RTT measured by the *ATC-Measurer* is lower than that measured by the *U-Measurer*. The standard deviation of the *U-Measurer* can achieve around 130μs, and the standard deviation of the *ATC-Measurer* can be less than 10μs. In addition, the mean difference can reach around 500μs. As can be seen from Fig. 6, the path RTTs measured using the *ATC-Measurer* is almost stable, but the RTT jitter using the *U-Measurer* is large. So the delay measurement using the traditional methods and tools will contain the location error. Hence, the results of network path delay measurement using the *ATC-Measurer* will be more accurate. Especially, the *ATC-Measurer* can be suitable for the large-scale path delay measurement.

TABLE II.

THE PATH RTT MEASUREMENT RESULTS UNDER CASE 4

Application	Min	Mean	Max	SD
ATC-Measurer	254	258.3	267	4.9
U-Measurer	427	817.6	1026	128.7

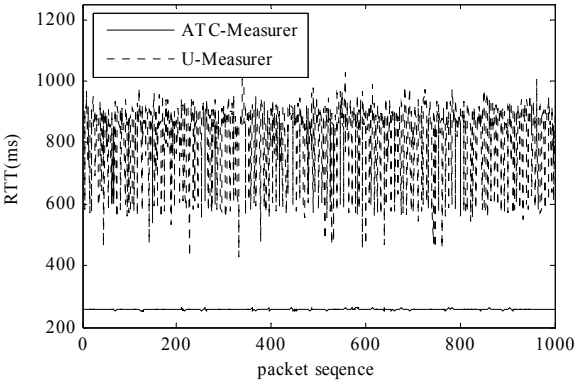

Figure 4. The time series of path round-trip delay under case 4

V. CONCLUSION

Understanding the network path delay is of crucial importance: first, it affects the quality of service of realtime applications and enables people to design an efficient congestion control mechanism for both realtime and non-realtime applications. In this paper, an accurate and low-cost delay measurement program is presented to facilitate FOWD, ROWD and RTD measurements. In our scheme, the *combined algorithm* is employed to eliminate the clock error from online delay measurements, and an NDIS intermediate driver is exploited to mark and collect the measurement timestamps. Especially, based on the kernel NDIS driver, the collection position of probe timestamp can be moved from the application program to data link layer of TCP/IP stack. The experimental results show that this method can nearly eliminate the location error. Meanwhile, the measurement accuracy can be practically guaranteed. In addition, the measurement scheme has low-cost in the investment, so it can be applied widely. In this paper, Windows XP is used as the target operating system. However, the exploited concepts are general and can also be adapted to other operating systems.

ACKNOWLEDGMENT

The work was supported by the Important National Science & Technology Specific Projects of China under grant no.2010-ZX03004-002-01, the National Natural Science Foundation of China (11102124, 61102076, and

60939002), the Program for New Century Excellent Talents in University of China (NCET-10-0604), the Youth Foundation of Sichuan Province (09ZQ026-028), and the Ph.D. Programs Foundation of Ministry of Education of China (20090181110053).

REFERENCES

[1] G. Almes, S. Kalidindi, and M. Zekauskas, “A one-way delay metric for IPPM,” [Online]. Available: http://www.ietf.org/ rfc/rfc2679.txt.
[2] G. Almes, S. Kalidindi, and M. Zekauskas, “A round-trip delay metric for IPPM,” [Online]. Available: http://tools.ietf.org/ html/rfc2681.
[3] J. H. Choi, and C. Yoo, "One-Way Delay Estimation and Its Application," Computer Communications, Vol. 28, No. 7, pp. 819-828, May 2005. doi:10.1016/j.comcom.2004.11.010
[4] M. Aoki, E. Oki, and R. Rojas-Cessa, "Measurement Scheme for One-Way Delay Variation with Detection and Removal of Clock Skew," ETRI Journal 32(6), pp. 854–862, 2010. doi:10.4218/etrij.10.0109.0611.
[5] D. Kim and J. Lee, "End-to-End One-Way Delay Estimation Using One-Way Delay Variation and Round-Trip Time," Proc. QSHINE '07. doi:10.1145/1577222.1577249.
[6] C. X. Guo and S. R. Zheng, "Analysis and evaluation of the TCP/IP protocol stack of LINUX," Proc.WCC-ICCT '2000, pp. 444-453. doi:10.1109/ICCT.2000. 889245.
[7] J. Mogul, k. Ramakrishnan, "Eliminating receive livelock in an interrupt -driven kernel," ACM Transactions on Computer Systems, Vol. 15, No. 3, pp. 217-252, 1997. doi:10.1145/263326.263335.
[8] The Real Outlook for Ping, [Online]. Available: http://www.articlekarma.com/Article/The-Real-Outlook-For-Ping/21799.
[9] J. Postel, "Internet Control Message Protocol," [Online]. Available: http://www.faqs.org/rfcs/rfc792.html.
[10] G. H. Lu, S. X. Sun, Z. L. Sao, and Yan. Z, "Multicast-Based measurement of network delay," Journal of Software, Vol. 15, No. 3, pp. 1704−1709, 2001. doi: RJXB.0.2001-11-018.
[11] Ping vs. Synack, [Online]. Available: http://www.iepm.slac.stanford.edu/~cottrell/pinger/synack/ping-vs-synack.html.
[12] J. Postel, "Transmission Control Protocol," [Online]. Available: http://www.faqs.org/rfcs/rfc793.html.
[13] J. F. Wang and M. T. Zhou, "Survey on the end-to-end internet delay measurements," High Speed Networks and Multimedia Communications Lecture Notes in Computer Science, 2004, Volume 3079/2004, pp. 155-166, doi: 10.1007/978-3-540-25969-5_14.
[14] Tamás Varga and András Oláh, "Quality of Service Monitoring in IP Networks by Using Two-way Active Measurements," Proc. EUNICE'2000. doi: 10.1.1.28.4512.
[15] V. Paxson, "On calibrating measurements of packet transit times," Proc. SIGMETRICS '98, pp. 11–21. doi: 10.1145/277851.277865.
[16] V. Paxson, "Measurements and Analysis of End-to-End Internet Dynamics," Ph.D. dissertation. Lawrence Berkeley Nat. Lab., Univ. California, Berkeley, 1998.
[17] S. Moon, P. Skelley, and D. Towsley, "Estimation and removal of clock skew from network delay measurements," Proc. INFOCOM '99. doi: 10.1.1.50.4299.
[18] H. Khlifi and J. C. Grgoirea, "Low-complexity offline and online clock skew estimation and removal," Computer Networks, Vol. 50, No. 11, pp. 1872–1884, 2006. doi: 10.1016/j.comnet.2005.08.009.
[19] L. Zhang, Z. Liu, and C. H. Xia, "Clock synchronization algorithms for network measurements," Proc. INFOCOM 2002, pp. 160–169, June 2002. doi: 10.1.1.16.7716.
[20] J. Micheel, S. Donnelly, and I. Graham, "Precision timestamping of network packets," Proc. of the 1st ACM SIGCOMM Workshop on Internet Measurement, pp. 273-277, 2001. doi: 10.1145/505202.505236.
[21] Packet Capture With libpcap and other Low Level Network Tricks. [Online]. Available: http://eecs.wsu.edu/~sshaikot/docs/lbpcap/libpcap-tutorial.pdf

Shiqiang Chen received the M.S. degree in Computer Software and Theory from Institute of Computer Software, Guizhou University, Guiyang, China, in 2005. He is currently a Ph.D. student in the College of Computer Science of Sichuan University. His research interests mainly focus on network measurement and monitoring, communication protocol, and network security technology.

Junfeng Wang received the M.S. degree in Computer Application Technology from Chongqing University of Posts and Telecommunications, Chongqing, China, in 2001 and Ph.D. degree in Computer Science from University of Electronic Science and Technology, Chengdu, China, in 2004. From July 2004 to August 2006, he held a postdoctoral position in Institute of Software, Chinese Academy of Sciences. From August 2006, Dr Wang is with the College of Computer Science, Sichuan University as a professor. His recent research interests include spatial information networks, network and information security, network measurement and monitoring, and intelligent transportation system.

Yifang Qin is an assistant professor in High Performance Network Lab., Institute of Acoustics, Chinese Academy of Sciences. He received B.S. degree from Huazhong University of Science and Technology, China in 2006, and Ph.D. degree in Signal and Information Processing from Institute of Acoustics, Chinese Academy of Sciences in 2011. His research interests include cognitive network, IP network management and their applications to communication systems.

Xu Zhou is an associate professor in High Performance Network Lab., Institute of Acoustics, Chinese Academy of Sciences. He received B.S. and M.S. degree in Mathematics from Sichuan University, China in 1998 and 2001, respectively, and Ph.D. degree in Computer Science from University of Electronic Science and Technology of China in 2004. His research interests span the areas of peer-to-peer network, mobile Internet, future network, scalable video coding, complex network optimization and their applications to communication systems.