

# A Rate and Resource Detection Based Receive Buffer Adaptation Approach for High-speed Data Transportation

Hao Liu, Yaoyue Zhang, Yuezhi Zhou and Ruini Xue

Key Laboratory of Pervasive Computing, Ministry of Education

Tsinghua National Laboratory for Information Science and Technology

Department of Computer Sci. and Tech., Tsinghua University, Beijing 100084, China

liuhao.buaa@gmail.com, zyx@moe.edu.cn, zhouyz@mail.tsinghua.edu.cn, xueruini@gmail.com

**Abstract**—With the development of computing devices and networks, several efficient and high performance UDP-based protocols have been proposed and employed in recently emerging computing paradigms, e.g., pervasive or cloud computing, to transport large data. However, since the server in such protocols uses a fixed-size memory buffer to hold received packets before handling them, the buffer will be exhausted if these packets cannot be handled as fast as they arrive, impairing the performance dramatically even if there is plenty of free memory.

To solve this problem, we propose a Rate and Resource Detection Based Buffer Adaptation Approach (RRDA). RRDA collects the difference between the server receiving and processing rate and the amount of free memory periodically. Based on these information, RRDA decides whether the receive buffer should be resized and if so, to what extent to adjust. RRDA can not only avoid the exhaustion of receive buffer when the server load is heavy, but also can free unnecessary memory when the load is low. Experimental results show that RRDA can reduce the occurrence of buffer exhaustion by a factor of 10 and improve the throughput remarkably, compared with the fixed-size buffer scheme.

## I. INTRODUCTION

With the development of various computing devices and network technologies, several emerging computing paradigms emphasizing “computing goes everywhere” have become hot topics in relevant research fields, for example, pervasive computing [1], grid computing [3], and more recently cloud computing [2]. By and large, these paradigms heavily rely on the availability of networks to perform better. Thus, high-speed networks, e.g., 1 or multi-10 Gbps, and relevant data transport protocols have been proposed to take the full advantage of high performance networks.

For example, UDP-based Data Transfer Protocol (UDT) [9] has been proposed to utilize the available bandwidth of wide-area high performance 10 Gbps networks and now has been employed in Open Cloud Testbed [10] to transport large data among cloud nodes [5], [6]. For another example, Performance Adaptive UDP (PA-UDP) [4] has been proposed to facilitate bulk data transfers in next generation large-scale science applications. Other UDP-based protocols for high-speed data transportation include Hurricane [15], Tsunami [16], etc.

However, these UDP-based high-speed protocols cannot

yet fully leverage the wide bandwidth of network. In these protocols, the client encapsulates protocol packets and sends them to the server, while the server has to enqueue these packets into a fixed-size memory buffer before delivering them to the dedicated handler. Since the buffer size is initially negotiated and then fixed later [9], [4], [15] (referred to as static buffer scheme hereinafter), it would be used up if the packet receiving rate is much faster than the processing rate of the server, even if there is sufficient free memory in the server at that time. Therefore, new packets would be discarded due to the exhausted buffer, decreasing the throughput remarkably.

To address this problem, we propose a Rate and Resource Detection Based Buffer Adaptation Approach (RRDA). RRDA dynamically determines buffer requirement and adaptation volume by detecting the difference between the server receiving and processing rate periodically. RRDA increases the buffer size when the receiving rate is constantly higher than the processing rate; otherwise, it reduces the buffer size. The extent of the increased or reduced buffer size primarily depends on the actual difference between these two rates, and it is also adapted according to the fluctuation of available free memory in the server, preventing the buffer adaptation from impairing the server performance.

However, there are still two main challenges for RRDA. First, how to decide whether the buffer should be resized. This is difficult because the packets receiving and processing rate varies along with the application execution, and thus it is difficult to observe which one is faster during a period of time. To get the above two rates’ accumulative effect on the buffer during a recent period, we adopt *Exponential Moving Average* (EMA) [13] transformation to deduce whether a buffer adaptation is needed. Second, how much to resize the buffer if an adaptation is required. The resizing extent tends to either too small to meet the data transfer requirement, or so large that the system overall performance would be obstructed. Therefore, we adopt a simple and efficient strategy, which adapts the increased or reduced buffer size according to system memory utilization. If there is enough free memory, the buffer size increases much or decreases less, and vice versa.

We have implemented RRDA within a simple UDP-based

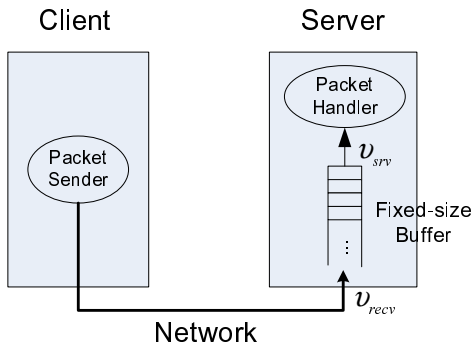


Fig. 1: The architecture of UDP-based high-speed protocol.

transport protocol, which employs a stop-and-wait [11] mechanism to send UDP data. Experimental results show that RRDA can reduce the occurrence of buffer exhaustion by a factor of 10 and improve the throughput by about 30% with only an overhead about 3%, against the static buffer scheme.

There are two main contributions in this paper. First, we propose a dynamic buffer adaptation approach, RRDA. RRDA adjusts the receive buffer size dynamically which can improve the throughput while keep the memory footprint as small as possible. Second, RRDA was extensively evaluated to verify its effectiveness. Experimental results show that RRDA can improve the throughput of data transportation with little overhead.

The paper is organized as follows. Section II describes the static buffer problem. Section III presents the design of RRDA, and Section IV explains several implementation issues. Section V provides a detailed evaluation of RRDA. Section VI discusses related work. Finally, Section VII concludes this paper.

## II. PROBLEM STATEMENT

In this section, we analyze the static buffer problem in existing high-speed UDP-based transport protocols.

### A. Background

Figure 1 shows the architecture of a typical UDP-based high-speed transport protocol. The client encapsulates user data into protocol messages and sends them to the server. Upon receiving the protocol messages, the server enqueue them in a buffer before delivering them to the packet handler. The buffer has a fixed size<sup>1</sup> and if it is exhausted, new messages will be discarded instead of being queued.

For the convenience of analysis, let  $v_{recv}$  be the receiving rate of the server, and let  $v_{srv}$  be the processing rate of the handler. From the view of long-term effect,  $v_{recv}$  should be equal to  $v_{srv}$ , whereas in a short run, there is a difference between the two. When  $v_{recv}$  is greater than  $v_{srv}$ , the data volume in the buffer increases, vice versa. Obviously, the data volume accumulated due to the short-term difference between  $v_{recv}$  and  $v_{srv}$  cannot go beyond the buffer size; otherwise a

buffer exhaustion would occur, and the subsequently arrived messages would be discarded even if there is free memory available, so that the throughput is restricted.

Let  $\lambda$  represent the relative difference between  $v_{recv}$  and  $v_{srv}$ , and  $L$  represent the buffer size, then the time needed for the accumulated messages to exhaust the buffer is  $L/(\lambda v_{recv})$ . Assume that  $v_{srv}$  is about 1% less than  $v_{recv}$  in a certain period of time on average, when  $v_{recv}$  reaches 100 Kbps, the time needed to exhaust the 1 MB buffer of UDT [21] is  $1MB/(100Kbps \times 1\%) = 10000s$ , whereas if  $v_{recv}$  is 1 Gbps, the time is as short as  $1MB/(1Gbps \times 1\%) = 1s$ . It is apparent that with the increase of data transmission rate, the probability of buffer exhaustion is increasing quickly, and this poses a severe problem for UDP-based high-speed protocols whose transmission speed is very fast.

### B. Dynamic Buffer Requirement

In order to avoid the buffer exhaustion, the buffer size can be manually adjusted to a larger value, but since it is difficult to estimate the actual buffer size needed, the fixed buffer size tends to be either too small to meet the requirement, or so large that the system memory would be wasted. Given that a larger buffer needs the more complicated data structures, a larger buffer would give rise to more memory management overhead. To minimize the memory usage while avoiding buffer exhaustion, the buffer requirement needs to be determined dynamically. For example, the buffer requirement can be affected by the processing power of the server (available CPU) and the number of simultaneous clients.

1) *Available CPU*: The buffer requirement is related to available CPU resources, which is fluctuating during system running, so the buffer requirement is also changing constantly. We verified the correlation of available CPU resources to the buffer requirement by the following experiment<sup>2</sup>. Thirty-five clients send data to the server simultaneously, and the server uses Windows Performance Monitor [12] to record the throughput. The results are presented in Figure 2. When available CPU resources amount to 4.26 GHz, the buffer with the size of 1 MB can reach the maximum throughput. And when the available CPU is adjusted to 3.2 GHz and 2.13 GHz, the server is only near its max throughput when the buffer size is increased to 10 MB and 100 MB, respectively.

2) *Number of Clients*: The buffer requirement is also related to the number of clients. Since the UDP buffer is allocated according to each UDP port so that all the clients share the same buffer of the server. Whereas TCP buffer is allocated according to each connection and different client has different buffer in the server [14]. In the UDP-based protocols, the buffer may be insufficient with the increase of number of clients, or it may become excessive when the number of clients decreases. We verified the correlation of the number of clients to the buffer requirement in the same experimental environment as mentioned above. The results are presented in Figure 3. A small buffer of 100 KB can achieve maximum

<sup>1</sup>The default size is 131 KB in Linux [20] and 1 MB in UDT version 4 [21].

<sup>2</sup>The experimental setup is the same as described in Section V-A.

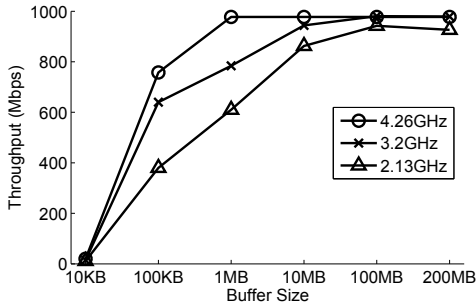


Fig. 2: Throughput for different CPU resource with the variation of buffer size.

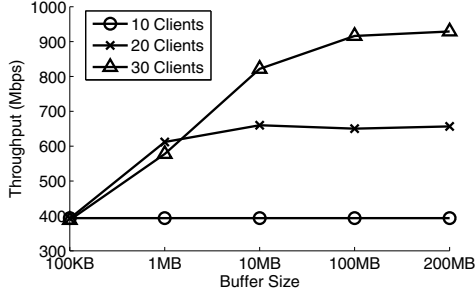


Fig. 3: Throughput for different client number with the variation of buffer size.

throughput when there are only 10 clients, whereas the number of clients increases, a larger buffer is needed to increase the throughput.

### III. DESIGN

As described in the previous section, the receive buffer requirement varies due to such factors as available CPU resources and the number of clients, which makes static buffer scheme hardly effective. Moreover, direct quantification of the buffer requirement is also difficult because available CPU resources and client number keep changing. Instead of devising complex models to model available CPU and number of clients, RRDA captures the general trend of the buffer requirement, based on which the buffer can be enlarged or shrunk.

RRDA determines the general fluctuation of buffer requirement by periodically detecting data receiving rate  $v_{recv}$  and data processing rate  $v_{srv}$ . The basic buffer adaptation principle is when  $v_{recv}$  is constantly greater than  $v_{srv}$ , the buffer size shall be increased, vice versa. The extent of such buffer size increase or reduction primarily depends on the actual difference between  $v_{recv}$  and  $v_{srv}$ , and it is also adapted according to the fluctuation of available free memory. Section III-A explains how to decide whether the buffer should be resized, and Section III-B presents how much the buffer should be adapted.

#### A. Buffer Adaptation Decision

Let  $v_b$  represent the difference between  $v_{recv}$  and  $v_{srv}$  ( $v_b = v_{recv} - v_{srv}$ ), the buffer adaptation decision actually depends

on the sign of  $v_b$  in consecutive epochs. However, it is difficult to observe  $v_b$  is constantly greater or less than 0 during a period of time, because  $v_{recv}$  and  $v_{srv}$  vary along with the execution of application.

We ran each of the following applications for more than forty times to explore  $v_b$ 's variation: (1) client sending rate sticks at 10 Mbps; (2) sending rate abruptly increases from 10 Mbps to 100 Mbps; (3) and sending rate abruptly decreases from 100 Mbps to 10 Mbps. The results show that there are generally three typical scenarios for  $v_b$ :

Scenario 1  $v_b$  constantly alternates between nearly the same positive and negative values. The buffer needs no adaptation.

Scenario 2  $v_b$  is constantly greater or less than 0 during several consecutive epochs. The buffer should be increased or decreased respectively.

Scenario 3  $v_b$  changes abruptly in certain discrete epochs or varies very much between positive and negative values. It is difficult to know how to resize the buffer.

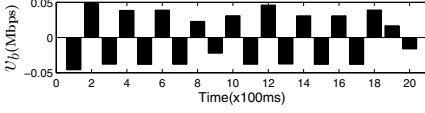
Figure 4 shows above three scenarios respectively. Figure 4a shows that  $v_b$  constantly alternates between nearly the same positive and negative values (Scenario 1), in which case the buffer requirement remains unchanged and the buffer needs no adaptation. Figure 4b shows that  $v_b$  is constantly greater than 0 during several consecutive epochs, thus the accumulative effects of these epochs lead to the increase of buffer requirement and the buffer needs increasing (Scenario 2). Figure 4c shows that  $v_b$  abruptly presents greater values in several epochs, e.g., the 1st and 8th epoch, so that it is difficult to know how to resize the buffer (Scenario 3). Therefore, we need to transform Scenario 3 to either Scenario 1 or Scenario 2 to expose the characteristics of  $v_b$  in order to apply the buffer adaptation principle directly.

The *Exponential Moving Average* (EMA) [13] transformation method is adopted. Let  $v_{b,n}$  represent  $v_b$  in the  $n$ th epoch, the EMA transformation of  $v_b$  is shown in (1). The new value  $v'_b$  is determined by both the detected rate during the current epoch and its value in the last epoch, of which the weight given to the value of the current epoch depends on the parameter  $\omega$  ( $\omega \in [0, 1]$ ).

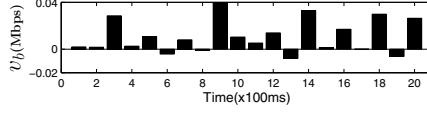
$$v'_{b,n} = \omega v_{b,n} + (1 - \omega)v'_{b,n-1} \quad (1)$$

Figure 5 shows the result of EMA transformation of Figure 4 ( $\omega = 0.5$ ). After the transformation, Figure 5a and Figure 4a, Figure 5b and Figure 4b still belong to the same scenarios respectively, while Figure 5c shows that Figure 4c is transformed from Scenario 3 to Scenario 2, and the buffer adaptation principle can be easily applied.

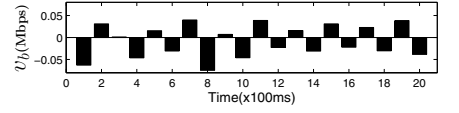
Let  $k$  represent the number of epochs which is based on to make buffer adaptation decision. Let  $f$  represent the fluctuations of buffer requirement.  $f = 0$  means the buffer needs no adaptation;  $f = 1$  means the buffer needs increasing; and  $f = -1$  means its decrease. Let  $v'_{b,i}$  represent  $v'_b$  in the  $i$ th epoch. Then the buffer adaptation principle can be expressed



(a) Scenario 1. The buffer needs no adaptation.

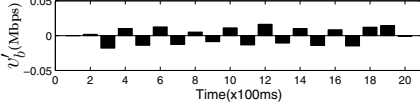


(b) Scenario 2. The buffer needs increasing.

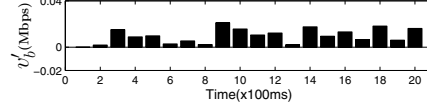


(c) Scenario 3. It is difficult to know how to resize the buffer.

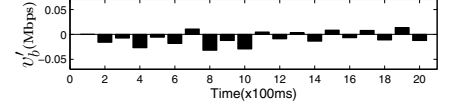
Fig. 4: Three typical  $v_b$  variation scenarios.



(a)  $v'_b$  variation is still Scenario 1. The buffer needs no adaptation.



(b)  $v'_b$  variation is still Scenario 2. The buffer needs increasing.



(c)  $v'_b$  variation is transformed from Scenario 3 to Scenario 2. The buffer needs decreasing.

Fig. 5:  $v'_b$  variation scenarios corresponding to Figure 4 after EMA transformation.

as (2).

$$f = \begin{cases} 1 & \text{for } \forall i, v'_{b,i} > 0, i \in [n-k, n-1] \\ -1 & \text{for } \forall i, v'_{b,i} < 0, i \in [n-k, n-1] \\ 0 & \text{others} \end{cases} \quad (2)$$

### B. Buffer Adaptation Strategy

Based on a series of  $v'_b$ , the buffer adaptation decision can be made, and the next challenge is how much the buffer should be adapted, which will be addressed in this section.

Let  $L_n$  represent the buffer size in the  $n$ th epoch. Basically, the buffer size after RRDA adjustment, i.e., the buffer size in the  $n+1$ th epoch can be calculated with (3).

$$L_{n+1} = L_n + f \times (\beta \times |v'_{b,n}|) \quad (3)$$

In (3), the product of the coefficient  $\beta$  and the rate  $|v'_{b,n}|$  determines the extent of increase or decrease of buffer. Such extent is correlative to the amount of free system memory, so that it is not appropriate to assume the same extent when available free memory varies. RRDA adopts different aggressive and conservative adaptation strategies according to memory abundance and insufficiency respectively. Let  $\alpha(n)$  represent the memory utilization (used memory/total memory) in the  $n$ th epoch, then (3) can be extended to (4) by taking system free memory into consideration.

$$L_{n+1} = \begin{cases} L_n + (1 - \alpha(n))\beta |v'_{b,n}| & \text{if } f = 1 \\ L_n & \text{if } f = 0 \\ L_n - \alpha(n)\beta |v'_{b,n}| & \text{if } f = -1 \end{cases} \quad (4)$$

Equation (4) leverages free system memory efficiently if there is abundant free memory and does not do harm to system performance otherwise. For example, if there is abundant free system memory,  $\alpha(n)$  is small, so that  $1 - \alpha(n)$  is relatively big. Thus, when the buffer is going to be enlarged,  $\beta |v'_{b,n}|$  is given a greater weight, and the size is increased by a greater margin than it is when there is insufficient free memory. When

the buffer is going to be reduced,  $\beta |v'_{b,n}|$  is given a smaller weight, and its size is decreased by a smaller margin than it is when there is insufficient free memory. When  $\alpha(n)$  approaches its two extremes, namely 1 and 0, (4) becomes (5) and (6), respectively.

$$\lim_{\alpha(n) \rightarrow 1} L_{n+1} = \begin{cases} L_n & \text{if } f = 1 \text{ or } f = 0 \\ L_n - \beta |v'_{b,n}| & \text{if } f = -1 \end{cases} \quad (5)$$

$$\lim_{\alpha(n) \rightarrow 0} L_{n+1} = \begin{cases} L_n + \beta |v'_{b,n}| & \text{if } f = 1 \\ L_n & \text{if } f = -1 \text{ or } f = 0 \end{cases} \quad (6)$$

Equations (5) and (6) show that the buffer is no longer enlarged when there is no available free system memory; and on the other hand, it will stop decreasing when the entire system memory is totally free.

## IV. IMPLEMENTATION

We have implemented RRDA within a simple UDP-based high-speed protocol as mentioned previously. Three threads were employed in RRDA: one collects server receiving rate; one monitors server processing rate; and one adapts the buffer. Based on the first two threads' output, the third thread makes the EMA transformation, queries the memory utilization  $\alpha(n)$  through system call and adapts the buffer. These threads were implemented with C++ code in Windows Server 2003 Enterprise Edition (SP2).

## V. EVALUATION

In this section, we evaluate the effectiveness of RRDA by answering the following questions (all are compared to the static buffer scheme):

- How much can RRDA reduce the probability of buffer exhaustion?
- How much can RRDA improve the data transportation throughput?
- What is the overhead of RRDA?

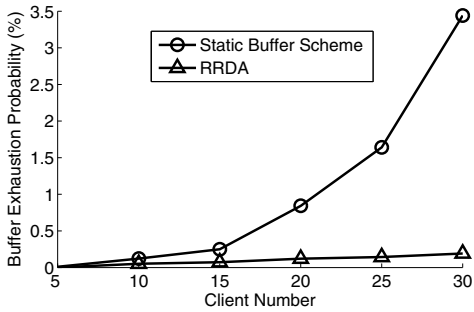


Fig. 6: Buffer exhaustion probability for RRDA and the static buffer scheme with the increase of the number of clients.

### A. Experimental Setup

There is a server and thirty-five clients in the experiment.

- Server: an Intel Xeon Dual Core 2.13 GHz CPU, 4 GB DDR2 667 MHz memory, one 1 Gbps Broadcom BCM5708C network card, a single Hitachi 160 GB 15000 rpm SATA hard disk and Windows Server 2003 Enterprise Edition (SP2).
- Client: Each client consists of an Intel Atom 1.60 GHz CPU, 512 MB DDR2 667 MHz memory, one 100 Mbps Realtek network card and Windows XP Professional (SP3). The clients and the server connect with each other through a TP-LINK TL-SG1048 gigabit switch.

In our tests,  $\Delta t = 100ms$ ,  $k = 3$ ,  $\beta = 16$ ,  $\omega = 0.5$ , and  $\alpha(n)$  can be easily calculated via the Windows system call `GlobalMemoryStatusEx`. The static buffer scheme employs a fixed-size buffer of 1 MB, which is the default buffer size of UDT version 4 [21].

### B. Buffer Exhaustion Probability

When the buffer exhaustion occurs, the newly arrived messages will be discarded, therefore the buffer exhaustion probability can be measured by the ratio of discarded messages in total messages. The result of buffer exhaustion probability when the clients send data to the server simultaneously is shown in Figure 6. With the use of static buffer scheme, when the number of clients exceeds 15, the buffer exhaustion probability increases rapidly with the increase of number of clients, whereas when RRDA is used, the probability is just slightly increased. RRDA reduces the buffer exhaustion probability by a factor of 10 compared to the static buffer scheme.

### C. Performance Improvement

Figure 7 shows the server throughput with different available CPU resources when thirty-five clients send data to server simultaneously. When the amount of available CPU resources is small, the throughput of RRDA is apparently higher than that of the static buffer scheme (30% on average). When the amount of available CPU resources reaches 4.26 GHz, the throughput of RRDA is equivalent to that of the static buffer scheme. This is because the CPU resources are relatively sufficient at this point, and the chances are slim that  $v_{recv}$

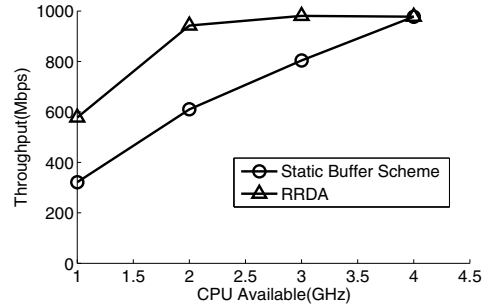


Fig. 7: Throughput for RRDA and the static buffer scheme with the variation of available CPU resources.

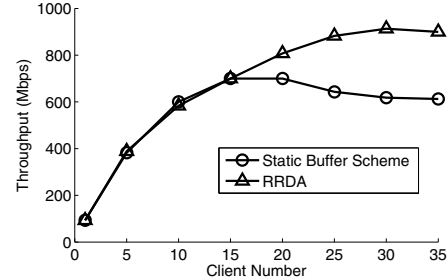


Fig. 8: Throughput for RRDA and the static buffer scheme with the increase of the number of clients.

persistently exceeds  $v_{srv}$ . Therefore, the buffer requirement can be easily met even when the buffer is small.

Figure 8 presents the server throughput with different number of clients. When the number of clients is less than 15, the server throughput in both cases are approximately the same. When the number of clients exceeds 15, however, the server throughput for RRDA is obviously higher than that of the static buffer scheme. This shows that RRDA improves the transport throughput as the number of clients increases.

### D. CPU Overhead

Sections V-B and V-C show that RRDA presents several benefits to the data transportation, and this section shows that RRDA only introduces very small CPU overhead. Since RRDA performs additional monitoring and calculation than the static buffer scheme, it requires more CPU resources inherently. Figure 9 shows the CPU usage per Gbps during a period of 260 seconds when thirty-five clients send data to the server simultaneously. The average CPU usages are 66.1% and 64.2% for RRDA and the static buffer scheme respectively, thus the overhead caused by RRDA is about 3%.

## VI. RELATED WORK

As a general problem, buffer adaptation relates to many topics. However, most of current work focuses on static buffer size optimization, while RRDA tries to address dynamic buffer adaptation for high-performance UDP-based transport protocols.

Cruz proposed an algorithm for the buffer allocation problem in general, finite, and single-server queuing networks [7].

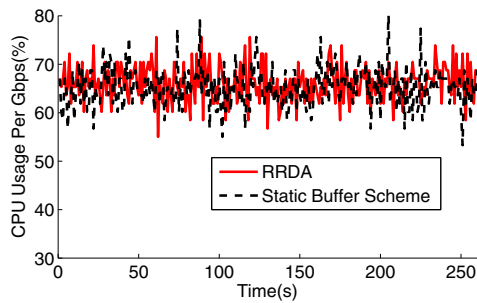


Fig. 9: CPU usage per Gbps during a period of 260 seconds.

Though the algorithm is simple and can be deployed easily, it is designed for blocking queuing system, in which case the clients are blocked if the server queue is full. However, this is different from our scenarios: packets will be discarded instead of being blocked if the buffer queue is full. Moreover, Cruz's algorithm aims to find the optimal buffer size to guarantee that the blocking probability never goes beyond a certain threshold, while RRDA tries to minimize the buffer size while maximizing the throughput. Cruz also proposed another algorithm to maximize the throughput while minimizing the buffer size, however, it still can only be applied to blocking queuing systems [8].

The most significant difference between RRDA and the previous approaches is that RRDA adjusts the buffer size dynamically, while the latter employs static buffer allocation optimization approaches. By static buffer allocation optimization, the previous approaches assume that system parameters are stable, which means packet arrival rates and service rates are almost fixed during the whole execution. Actually, new computing systems, such as cloud computing systems, do not obey these rules. Therefore, dynamic buffer adaptation is more efficient than the static buffer scheme.

There are also dynamic buffer adaptation approaches proposed [17], [18]. Unfortunately, these methods are designed for TCP-based protocols and cannot be adopted in high-performance UDP-based transport protocols that RRDA is devised for.

Several other researches also analyze the problem of the fixed buffer size in high-performance UDP-based transport protocols [19], [20]. But they only proposed to set the size manually but did not adapt the size dynamically. Due to the system dynamics, these manual results are far from the optimal size.

## VII. SUMMARY AND FUTURE WORK

We have studied the receive buffer exhaustion problem in high-performance UDP-based protocols and introduced RRDA, a dynamic buffer adaptation method based on rate and resource detection. By periodically detecting the difference between the data receiving and processing rate and the amount of available free memory, RRDA enlarges or shrinks the buffer as needed to satisfy the dynamic buffer requirement and at the same time to avoid the impact on the server performance.

We also have implemented and verified the effectiveness of RRDA through several experiments. The results show that RRDA can reduce the occurrence of buffer exhaustion and improve the throughput with small overhead. Future work includes exploring the mechanism in more high-performance transport protocols and in different local or wide area network based computing scenarios.

## ACKNOWLEDGMENT

This work has been supported in part by the National High Technology Research and Development Program of China (No.2009AA01Z151) and National Core-High-Base Major Project of China (No.2009ZX01036-002-3). We would like to thank the anonymous reviewers for their helpful comments on this paper.

## REFERENCES

- [1] V. Dhirga and A. Arora, "Pervasive Computing: Paradigm for New Era Computing", First International Conference on Emerging Trends in Engineering and Technology, Los Alamitos, CA, USA: IEEE Computer Society, 2008, pp. 349-354.
- [2] A. Weiss, "Computing in the clouds", *netWorker*, vol. 11, 2007, pp. 16-25.
- [3] I. Foster and C. Kesselman, *The Grid 2: Blueprint for a New Computing Infrastructure*, Morgan Kaufmann Publishers Inc., 2003.
- [4] B. Eckart, X. He, Q. Wu, and C. Xie, "A Dynamic Performance-Based Flow Control Method for High-Speed Data Transfer", *IEEE Transactions on Parallel and Distributed Systems*, vol. 21, 2010, pp. 114-125.
- [5] R. L. Grossman, Y. Gu, M. Sabala, and W. Zhang, "Compute and storage clouds using wide area high performance networks", *Future Gener. Comput. Syst.*, vol. 25, 2009, pp. 179-183.
- [6] National Science Foundation (NSF) News. "Data Travels Six Times Faster in the Clouds". [http://www.nsf.gov/news/news\\_summ.jsp?cntn\\_id=114229](http://www.nsf.gov/news/news_summ.jsp?cntn_id=114229).
- [7] F. R. B. Cruz, A. R. Duarte, and T. V. Woensel, "Buffer allocation in general single-server queueing networks", *Comput. Oper. Res.*, vol. 35, 2008, pp. 3581-3598.
- [8] F. R. B. Cruz, G. Kendall, and L. While. "A Throughput Maximization Problem with Minimization of Service Rates and Buffer Sizes". <ftp://est.ufmg.br/pub/fcruz/publics/mubap.pdf>.
- [9] Y. Gu and R. L. Grossman, "UDT: UDP-based data transfer for high-speed wide area networks", *Comput. Netw.*, vol. 51, 2007, pp. 1777-1799.
- [10] Open Cloud Testbed. <http://opencloudconsortium.org/testbed/>.
- [11] Wikipedia. "Stop-and-wait ARQ". [http://en.wikipedia.org/wiki/Stop-and-wait\\_ARQ](http://en.wikipedia.org/wiki/Stop-and-wait_ARQ).
- [12] Microsoft. "Windows NT Workstation Resource Kit: About Performance Monitor". <http://www.microsoft.com/resources/documentation/windowsnt/4/workstation/reskit/en-us/02perfmn.msp?mfr=true>.
- [13] Wikipedia. "Moving average". [http://en.wikipedia.org/wiki/Exponential\\_moving\\_average#Exponential\\_moving\\_average](http://en.wikipedia.org/wiki/Exponential_moving_average#Exponential_moving_average).
- [14] Wikipedia. "Internet socket". [http://en.wikipedia.org/wiki/Internet\\_socket](http://en.wikipedia.org/wiki/Internet_socket).
- [15] N. S. V. Rao, Q. Wu, S. M. Carter, and W. R. Wing, "Highspeed dedicated channels and experimental results with hurricane protocol", *Annals of Telecommunications*, vol. 61, 2006, pp. 21-45.
- [16] M. Meiss. "Tsunami: A high-speed rate-controlled protocol for file transfer". <http://www.evl.uic.edu/eric/atp/Tsunami.pdf>.
- [17] S. Tao, L. K. Jacob, and A. Ananda, "A TCP socket buffer auto-tuning daemon", *Proceedings of the 12th International Conference on Computer Communications and Networks*, 2003, pp. 299-304.
- [18] R. S. Prasad, M. Jain, and C. Dovrolis, "Socket Buffer Auto-Sizing for High-Performance Data Transfers", *Journal of Grid Computing*, vol. 1, Dec. 2003, pp. 361-376.
- [19] 29West. "UDP Buffering Background". <http://www.29west.com/docs/THPM/udp-buffering-background.html>.
- [20] 29West. "UDP Buffer Sizing". <http://www.29west.com/docs/THPM/udp-buffer-sizing.html>.
- [21] UDT Manual. <http://udt.sourceforge.net/udt4/index.htm>.