Buffer Bottleneck in UDT

Tapan Kumar Nayak[1], Anup Avistita[2], Somnath Kejriwal[3]

Abstract— UDT is built on top of UDP with reliability control & congestion control, which employs a fixed size application level buffer to hold the data before delivering it to upper layer protocols. The fixed size application level buffer leads to two different problems. Firstly, a fixed size buffer without any precise control flow mechanism discards the newly arrived packets when exhausted, decreasing the throughput sharply .Secondly, buffer wouldn't be exhausted but to maintain certain flow mechanism they have to send the data below a certain threshold. Thus UDT being high speed protocol is still unable to utilize the network bandwidth efficiently. We refer this problem of fixed size application level buffer to be "Buffer Bottleneck in UDT". The size of buffer being either too large or small impairs the performance of transport protocol. The factors which are responsible for deciding the buffer size varies dynamically, thus requiring a need for a mechanism which dynamically adapts the buffer size. In this paper, we propose a mechanism which dynamically adjusts the buffer size, allowing UDT to utilize the rapidly emerging high speed wide area optical networks effectively.

Index Terms—Buffer Bottleneck, Congestion Control, Exponential Moving Average, Fixed size Buffer, Transport Protocol, UDP, UDT

IJSER

1 Introduction

In the past few years, the network bandwidth has been increased rapidly. Five years ago, a 1 GB/s link could be regarded as an extremely fast link, but today, 10 GB/s links have become very common on high speed testbeds, and 40 GB/s links are emerging. The widespread use of high speed networks has enabled many new distributed data intensive applications that were impossible in the past.

Meanwhile, we are living in a world of exponentially increasing data. Examples of large volumetric datasets involved in these applications include satellite weather data, astronomy observation data, and network monitoring data. In the past these data were usually stored in local storage and then were delivered or processed in a batch mode. Today they can be transferred to a remote site in real time and be processed there.

Unfortunately, the high-speed networks have not been smoothly used by highly developing applications. The Transmission Control Protocol (TCP) fails to perform efficiently under these high speed networks leading to introduction of various variants in TCP which also underutilizes these network bandwidths [1]. Thus to achieve high speed data transfer a UDP based protocol has been proposed & deployed name UDT [2]. UDT utilizes the network bandwidth efficiently but is unable to improve the throughput which is restricted by the fixed size application buffer [3] deployed in the receiving side.

2 Problem Statement

UDT still cannot fully utilize these high bandwidth networks. In this protocol the receiver employs a fixed-size application level buffer to hold the data received before delivering it to upper layer protocols. As the buffer size is fixed, the buffer could be exhausted in protocols without a precise flow control mechanism [4]. The newly arrived packets would be discarded and the throughput will decrease dramatically. Protocols with precise flow control ensure that the buffer would not be exhausted, but they have to employ a conservative mechanism to send the data at a speed below a certain threshold level. In both the cases, the throughput would be limited by the fixed buffer size, which we refer to be **Buffer Bottleneck in UDT**.

A common practice to avoid buffer bottleneck is to manually set the receiving buffer size to a large value. However due to difficulty of estimating the actual buffer size needed, manual adjustments tends to be either too small to maximize the transfer throughput or too large to fully utilize the large amount of memory. Also a waste of memory in one application could significantly impair the performance of other applications.

A large buffer requires complicated data structures, resulting in high overhead in buffer management. When memory in receiver side is heavily loaded, a large buffer leads to serious performance drops. Therefore an optimal buffer size is needed to found.

It is practically found that the optimal buffer size varies dynamically during runtime depending on the factors such as available memory, no. of simultaneous transfers' & many more.

Thus a straight forward approach is to adapt the receiving buffer size dynamically with change in parameters the optimal buffer size depends upon.

3 Idea of Dynamic Buffer

In this paper we are discussing only about the buffer deployed in receiver end as, firstly, there is seldom concern about the sending buffer in UDP based protocols, secondly, the OS can manage the kernel UDP buffer on demand by simply monitoring the buffer occupancy, while in the receiving end buffer occupancy is not an indicator of buffer demand as precise flow control ensures that the buffer doesn't get exhausted.

There are no performance metrics to measure the buffer performance in receiving end as present in sending end buffer, such as TCP's dynamic adaptation scheme was proposed for sending buffer of I/O requests based on estimated data latency. These schemes aim at adapting the sending buffer, & use the perceived performance metrics such as packet loss, queuing delay & BDP to adjust sending buffer size. No such metrics exists for the buffer at the receiving end.

Various theoretical & experimental analyses have shown that the receiving buffer size depends upon: -

1. CPU Usage (Busy/Idle)
2. Available Memory
3. Available Bandwidth
4. No. of Simultaneous Transfers

When the CPU or disk becomes busy, increase the buffer; when the memory becomes heavily loaded, decrease the buffer; when the available bandwidth de-creases, decrease the buffer; and when the number of simultaneous transfers (transfer number hereinafter) de-creases, increase the buffer.

Moreover, it is difficult to deal with situations where more parameters change simultaneously. For example, there is a change (decrease) in memory utilization (which requires increasing the buffer) meanwhile there is another change (increase) in transfer number (which requires decreasing the buffer). Therefore, it is important to design an algorithm that can adapt to changes in all types of conditions.

All the factors described above except available memory ultimately lead to the variation of the following two rates: data arrival rate, v_{recv}, and data consumption rate, v_{srv}, in the buffer. When the disk becomes busy, v_{srv} decreases; when the available link bandwidth increase, v_{recv} increases; other condition changes, except the variation of available memory, can also be transformed to the variation of v_{recv} or v_{srv}.

Buffer adaptation decision is taken based on following three steps in Rate Detection based scheme.

Step 1: Periodic detecting the values of V_{recv} & $V_{srv.}$

Step 2: If V_{recv} is constantly larger than V_{srv}, the buffer must be increased; if V_{recv} is constantly smaller than V_{srv}, the buffer must be decreased; otherwise, the buffer does not need to be adapted.

Step 3: As the factor of available memory cannot be transformed to the variation of V_{recv} and V_{srv}, the adaptation extent in each buffer increase/decrease operation is also adapted according to the available memory.

4 An Approach to Dynamic Buffer

Dynamic Buffer Adaptation employs a rate detection based approach [5]. If the rate of data receival is greater or lesser than the rate of data consumption then the buffer size needs to be increased or decreased respectively. The implementation of this type of approach faces the following three subproblems:

1. It is difficult to decide when to de-cide adapt the buffer. However, it is not clear how to quantify that v_{recv} is constantly larger/smaller than v_{srv}.
2. There is any existing way about how much the adaptation extent should be or what kind of adaptation pattern should be employed.
3. The adaptation extent must be adaptive to the available memory in the receiver.

5 Buffer Adaptation Decision

Dynamic Buffer at receiving side employs a rate detection based buffer adaptation approach. Three sub problems lie in such a rate detection based scheme:

1. When to accept the buffer
2. To what extent buffer should be used
3. The factor available memory cannot be transformed to the variations of V_{recv} & V_{srv}.

5.1 When to accept?

- $v_b = v_{recv} - v_{srv}$
- Various experiments with a UDP based high speed protocol, as well as another UDP based high speed protocol shows that 3 categories of scenarios for v_b from the aspect of buffer adaptation for these protocols:
  1 **Scenario 1** v_b constantly alternates between nearly the same positive and negative

IJSER

values. The buffer requires no adaptation.

2 **Scenario 2** v_b is constantly positive or negative during several consecutive epochs. The buffer size should be increased or decreased respectively

3 **Scenario 3** Scenarios other than the above two. For example, v_b changes abruptly in certain discrete epochs or varies significantly between positive and negative values. It is not easy to decide how to resize the buffer.

Scenario 3 is needed to be transformed to either Scenario1 or Scenario 2. Some Moving Averages methods are useful in smoothing out short – term fluctuations & high lighting long term trends.

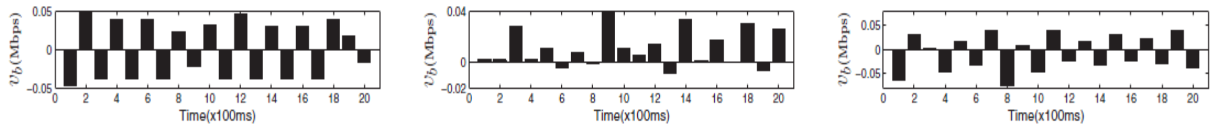
SMA (Simple Moving Average) treats all activities without discrimination, while EMA (Exponential Moving Average) emphasizes recent activities more than old activities. Thus, EMA performs better if a quick response to recent activities is required.To make the buffer adaptation more responsive, we also employ the EMA scheme to make a transformation of v_b. Let $v_{b,n}$ represent the value of v_b in the nth epoch. The EMA transformation of v_b is shown in Equation (1). The new value v_b is determined by both the detected rate difference in the current epoch and its value in the last epoch, of which the weight given to the current epoch depends on the parameter ω (ω ∈ [0, 1]).

$$V'_{b,n} = \omega V_{b,n} + (1-\omega)V_{b,n-1} \qquad \text{Eqn. 1}$$

Let k denote the number of epochs which the buffer adaptation decision is made based on, and let $V_{b,i}$ represent V_b in the ith epoch. Let f denote the buffer adaptation decision; the decisions to increase, decrease and make no adaptation are represented by f = 1, f = −1 and f =0 respectively. Then the decision of buffer adaptation can be expressed as follows: -

$$f = \begin{cases} 1 \; for \forall i, v'_{b,i} > 0, i \in [n-k+1, n] \\ -1 \; for \forall i, v'_{b,i} < 0, i \in [n-k+1, n] \\ 0 \; \text{Otherwise} \end{cases}$$

IJSER

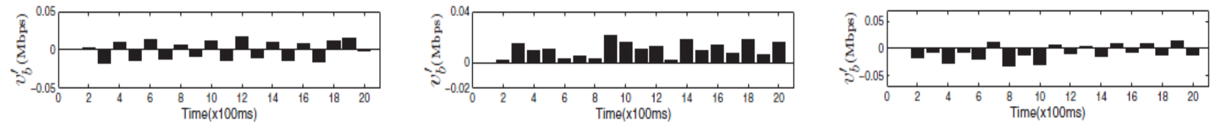


(a) Scenario 1. The buffer requires no adaptation.

(b) Scenario 2. The buffer must be increased.

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

Fig. 1: Three categories of scenarios for the variation of v_b.



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

(b) v'_b variation is still Scenario 2. The buffer must be increased.

(c) v'_b variation is transformed from Scenario 3 to Scenario 2. The buffer must be decreased.

Fig. 2: v'_b variation corresponding to Figure 1 after *EMA* transformation.

5.2 To What Extent?

TCP tries to additively increase the congestion window size to maximize the throughput, until the packet loss is detected; the congestion window is halved to avoid further packet loss. TCP employs AIMD (Additive Increase Multiplicative Decrease) [5] scheme. We observe that our motivation is opposite to that of TCP, i.e., the buffer size is expected to increase aggressively, so that the transfer throughput cannot be limited by the buffer size; it is also expected to decrease conservatively in case of large span rate (v_b) & a sharp increase in buffer demand afterwards.

However MIAD [6] scheme is not good in our scenario as it suffers from a sharp throughput drop in each one of the buffer increase operations. So we employ LAICD scheme in dynamic buffer adaptation. It avoids the overhead, but remains the property of aggressive increase & conservative decrease. The aggressive increase & conservative decrease are achieved by giving different adaptation extent to the increase & decrease operations. The increase operation is given a higher extent than the decrease operation, although both are linear. The following equation shows the above idea: -

$$L_{N+1} = \begin{cases} L_N + \beta_1 |v'_{b,n}| & \text{If } f = 1 \\ L_N & \text{If } f = 0 \\ L_N - \beta_2 |v'_{b,n}| & \text{If } f = -1 \end{cases}$$

IJSER

5.3 Memory Availability

An adaptation scheme without considering the available memory could impair the performance of the transfer as well as the whole system. For example 10 MB adaptation extent in 1000 MB available memory doesn't affect much but the same adaptation extent in 50 MB available memory will impair the performance.

Higher memory utilization requires smaller increase extent & bigger decrease extent whereas a lower memory utilization requires bigger increase extent & smaller decrease extent (i.e. if the buffer size needs to be increased but the available main memory is less than the buffer must be decreased and the sending rate must be decreased) [7]. The expected curve doesn't need to be Linear Monotonic as can be seen in following fig.

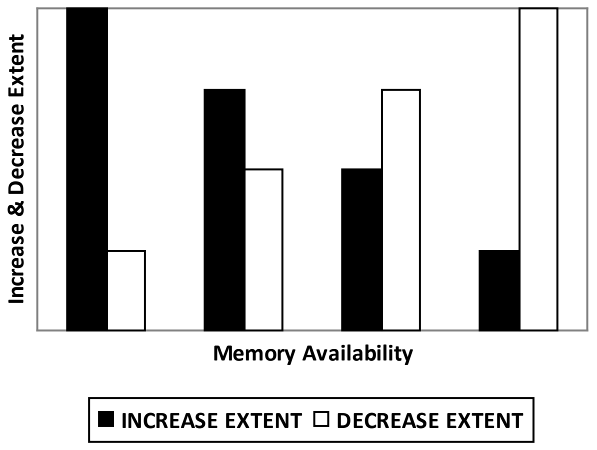


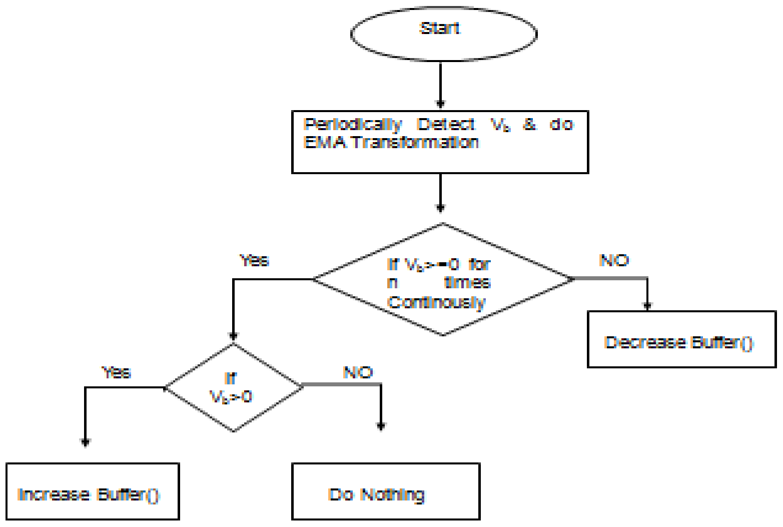
Fig.3. Graph showing increase decrease extent versus the memory availability in RAM

Adaptation extent makes a tradeoff between the demands of the transfer & receiving server based on the state of memory utilization. When the memory is heavily loaded, the receiving server is given a high priority; otherwise; the transfer is more highly prioritized. It can be achieved by a Weighted Mean Function (WMF) with the memory utilization as time varying weights. α(n) denote the memory utilization in the nth epoch, the buffer size after the increase & decrease operation can be expressed as: -

$$L_{N+1} = \begin{cases} L_N + (1-\alpha(n))\beta_1 |v'_{b,n}| & \text{If } f = 1 \\ L_N & \text{If } f = 0 \\ L_N - \beta_2 \alpha(n) |v'_{b,n}| & \text{If } f = -1 \end{cases}$$

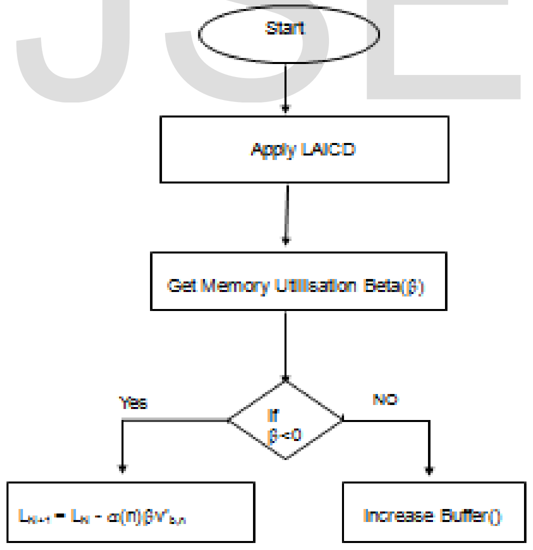
6 Idea to Implementation

In this section we are providing a way following which the idea of dynamic buffer can be implemented. The approach consistsof 3 functions linked to each other. The first one takes the buffer adaptation decision by periodically detecting the rate of consumption & rate of arrival of data whereas second & third functions perform the part of increasing or decreasing the buffer size as required. The above functions can be implemented through the flow chart below:

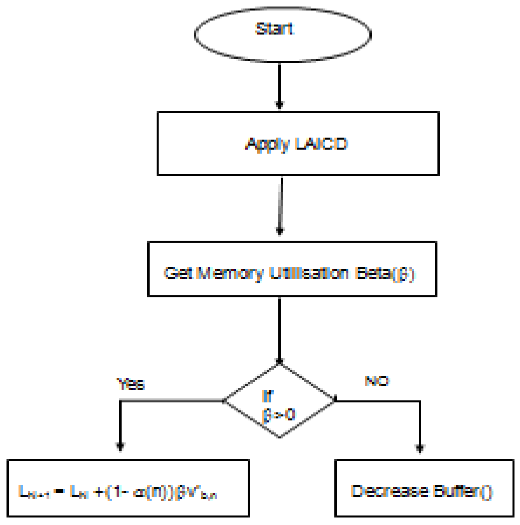

Start
Periodically Detect V_b & do EMA Transformation
If V_b>=0 for n times Continously
Yes
NO
Decrease Buffer()
If V_b>0
Yes
NO
Increase Buffer()
Do Nothing

FLOW CHART 1. BUFFER ADAPTATION DECISION

IJSER


Start
Apply LAICD
Get Memory Utilisation Beta(β)
If β<0
Yes
NO
$L_{N+1} = L_N - \alpha(n)\beta v'_{b,n}$
Increase Buffer()

FLOW CHART 2. DECREASE BUFFER


Start
Apply LAICD
Get Memory Utillisation Beta(β)
Yes
NO
If β>0
$L_{N+1} = L_N + (1-\alpha(n))\beta v'_{b,n}$
Decrease Buffer()

FLOW CHART 3. INCREASE BUFFER

IJSER

7 Conclusion

A number of high-bandwidth networks have been constructed, existing high-speed protocols cannot fully utilize the bandwidth of these networks, as their fixed size application level receiving buffers suffer from buffer bottleneck. A rate detection based buffer adaptation scheme is remedy to this problem.

- Exploring this dynamic buffer adaptation extent in other transfer protocols.
- Exploring this scheme in high speed Local & Wide Area Networks
- Implementing this scheme as libraries so that it can be easily ported in to the existing networks.

References

[1] W. Feng and P. Tinnakornsrisuphap: The Failure of TCP in High-Performance Computational Grids, Proc. of SuperComputing 2002.
[2] Yunhong, G., Robert G.: UDT: UDP-based Data Transfer for High-Speed WideArea Networks. University of Illinois, Chicago (2006)
[3] " Receiving Buffer Adaptation For High Speed Data Transfer" Hao Liu, Yaoxue Zhang, Yuezhi Zhou, Xiamong Fu, Laurenece T Yang (Vol 6, January 2012)
[4] Yunhong, G.: UDT - A High Performance Data Transport Protocol. University of Illinois, Chicago (2005)
[5] "An Analysis of AIMD Algorithm with Decreasing Increases" Yunhong Gu, Xinwei Hong, and Robert L. Grossman (Unpublished Publication)
[6] D. Chiu and R. Jain, "Analysis of the Increase/Decrease Algorithms for Congestion Avoidance in Computer Networks," *Journal of Computer Networks and ISDN*, Vol. 17, No. 1, June 1989, pp. 1-14.
[7] Data Communication & Comminication Network, 3rd Edition B. Forouzan.

1. *Tapan Kumar Nayak is currentlylecturer in Computer Science & Engineering department in College of Engineering & technology, Odisha, India. E-mail: chiku.tapan@gmail.com*
2. *Anup Avistita is currently pursuing bachelor degree program- in Computer Science & Engineering in Biju Pattnaik University of Technolgy, Odisha (India). E-mail: cet.anup@gmail.com*
3. *Somnath Kejriwal is currently pursuing bachelor degree program-in Computer Science & Engineering in Biju PattnaikUniversity of Technolgy, Odisha (India). E-mail: somnath.kejriwal@rediffmail.com*