멀티홈 모바일 호스트상에서 스트라이핑 전송계층 연결을 위한 적응형 버퍼튜닝기법

An Adaptive Buffer Tuning Mechanism for striped transport layer connection on multi-homed mobile host

파 라 즈* 허 의 남**
Faraz Idris Khan Eui-Nam Huh

요 약

최근 무선네트워크 기술은 이동 응용프로그램을 위해 이종통신망 연결패스 상에서 병렬로 스트라이핑 데이터 기술을 이용해 고속 데이터를 전달을 가능케 한다 (2). 전통적으로 대역폭지연프로덕트(BDP) 기반에서 고속전송은 송신자 측에서 다중 TCP 소켓의 튜닝을 요구한다. 더욱이, 메모리와 네트워크 요구의 균형을 유지하는 ATBT같은 기술은 유선기반의 단일 소켓상에서 하나의 플로우만 가정하여 설계되었다. 그러므로 본 논문은 여러 무선 패스를 경유하는 이종 무선네트워크 상에서 고속전송을 가능케하는 스트라이핑 전송기술에 적합한 버퍼튜닝 기술을 제안한다. 제안 기술은 이동성, 링크손실, 대역폭변동성 등의 특징을 지닌 무선 멀티홈 모바일 호스트상에서 작동하는 전송계층에서의 자원관리기술이다. 실험을 통하여 유선기반의 ATBT를 본 환경에 적용한 것보다 메모리, 평균 전송량에 있어 제안 기술의 성능이 우수하다.

Abstract

Recent advancements in wireless networks have enabled support for mobile applications to transfer data over heterogeneous wireless paths in parallel using data striping technique (2). Traditionally, high performance data transfer requires tuning of multiple TCP sockets, at sender's end, based on bandwidth delay product (BDP). Moreover, traditional techniques like Automatic TCP Buffer Tuning (ATBT), which balance memory and fulfill network demand, is designed for wired infrastructure assuming single flow on a single socket. Hence, in this paper we propose a buffer tuning technique at senders end designed to ensure high performance data transfer by striping data at transport layer across heterogeneous wireless paths. Our mechanism has the capability to become a resource management system for transport layer connections running on multi-homed mobile host supporting features for wireless link i.e. mobility, bandwidth fluctuations, link level losses. We show that our proposed mechanism performs better than ATBT, in efficiently utilizing memory and achieving aggregate throughput.

☞ keyword : socket buffer tuning, resource management, TCP, flow control, data stripping 소켓버퍼튜닝, 자원관리, TCP, 흐름제어, 스트라이핑

1. Introduction

Efforts to provide services to the astronomical mobile users led to the explosion of huge number of wireless access technologies. Each of these technologies exhibit unique and diverse network characteristics of coverage range and data rate. Thus, today there are various options available for access to a mobile user to choose from, ranging from Globstar for satellite access, General Packet Radio Service (GPRS), Wideband Code Division Multiple Access (WCDMA), Enhanced Data Rate for GSM Evolution (EDGE) for wide area access and IEEE 802.11 or HyperLAN for local area network access. Moreover, with the emergence of such diverse access

* 준 회 원 : 경희대학교 컴퓨터공학과
faraz@khu.ac.kr
** 종신회원 : 경희대학교 컴퓨터공학과 교수
johnhuh@khu.ac.kr(Corresponding author)
[2008/10/07 투고 - 2008/10/10 심사 - 2008/12/21 심사완료]

technologies an effort is made by network research community to utilize their coexistence to provide the best of the services to the mobile user.

The first ever proposal of a technology that realizes, the utilization of different access networks, is that of vertical handoff [1] providing ubiquity of service by handing off to a different network. It requires the usage of a single wireless interface by the application at a time. Lately, simultaneous usage of diverse multiple wireless interfaces to achieve high performance by aggregating bandwidth along different paths, is considered, as in [2]. In such scenarios, where high performance is required along diverse paths, a seemingly better solution of application layer stripping along different TCP sockets sharing same path, performs abjectly as shown in [2]. Hence, the network research community has come up with proposals to strip data at transport layer which lead to the stream of protocols such as parallel TCP (pTCP) [2], multiple TCP (mTCP) [3], Dynamic Multipath TCP (DMTCP) [4], Aggregate Bandwith Multihoming Support (AMS) [5], which are shown to perform well in case of diverse wireless path scenario. In all such protocols, the congestion control algorithm runs independently along multiple active paths in order to strip data according to the present network condition estimated by congestion window. It is important to note that the features of these protocols are inherited from TCP. In other words, these protocols are advance form of TCP, supporting striping.

The potential applications of the above mentioned protocols are distributed computing applications requiring high volume data transfer over high speed wireless connections such as 3G High Speed Uplink Access (HSUPA) or High Speed Downlink Access (HSDPA). For instance Mobile Data Grid, enabling heterogeneous grid resource, datawarehouse, access for huge amount of data over wireless, is a specific example of such distributed computing application. As TCP performs poorly across high capacity wide area network and requires proper tuning, proposals such as [6, 7] can be found whose ultimate target is to enable high performance data transfer. Moreover, in order to prevent tedious system tuning by the application developer, *Work Around Daemon (WAD)* [8] has been proposed which monitors network state and automatically configures buffer size for TCP connections at the host. All of such works are inspired by TCP performance degradation on high capacity wired infrastructure. And, the techniques employed to tune the connections considers congestion window as the estimate for the data in flight. However, in case of wireless connection congestion window is not a correct estimate, especially if the link is severely prone to errors. This is due to TCP's in ability to distinguish congestion losses from link level losses [9]. And, future generation mobile handset will encounter handoff both horizontal and vertical which requires notification from lower layers in order to enhance the performance of TCP. Also, as the mobile handsets are devices whose resources are to be utilized efficiently, therefore, such needs call for a resource management system which manages the TCP connections at the host efficiently utilizing the memory, as well as, by monitoring wireless link state.

ATBT [10] is the technique that balances memory usage, at the same time, meeting network target for multiple single flow TCP connections. With the advent of major TCP connection striped along diverse paths, a single socket will be logically shared by the micro flows. Hence, ATBT which assumes a single flow on a socket will perform poorly. Partially, the poor performance will be contributed due to in correct estimate of network target through

congestion window. Also, often at the host there are various small connections not utilizing the assigned buffer to the fullest. Hence, large connections such as striped connections requiring extra buffer for achieving high throughput can appropriate their space by applying greedy approach. On the contrary, during connection life time when the striped connection does not require extra buffer, it can allocate extra space to the other connections, running at the host, in proportion to their bandwidth.

Therefore, in this paper we propose a dynamic buffer tuning technique using cross layer communication for monitoring network state applicable to striped TCP connection along diverse wireless paths.

The contribution of this work is two fold

- *We propose a technique for tuning the buffer size of the connection by applying a controller often used in control theory to regulate the sizing process*
- *We set the network demand for the controller to tune the buffer; by using link layer information regarding dynamic changes in bandwidth and by monitoring the buffer occupancy of all the connections running at multihomed mobile host*

The rest of the paper is organized follows: Section 2 discusses related work in the direction of buffer tuning. Section 3 presents proposed system and discusses the details of tuning technique. Section 4 presents the simulation results. Section 5 concludes the paper with the discussion of future directions.

2. Related Work

2.1 Buffer Tuning Techniques

The advent of buffer tuning techniques arose with the need of improving the performance of the network applications on a high performance networks. Often the degradation of performance is experienced by the distributed code due to inability of TCP to open up its congestion window over WAN. In order to solve the problem, research community has come up with various buffer tuning techniques to avoid performance degradation [8]. In which the basic idea of tuning is to fill the BDP of the network path, the most important performance parameter in case of bulk data transfer.

The tuning mechanism in literature is classified as manual and automatic tuning mechanism. While, manual tuning is a tedious work leaving it to be a non-optimal solution.

ATBT in [10] is purely a sender – based approach where the sender uses TCP packet header information and timestamp to estimate the bandwidth delay product of the network which is used to resize the buffer, consequently leading to large sender window.

In contrast to ATBT, Dynamic Right Sizing (DRS) [11] is a receiver based approach, but like ATBT, it estimates the bandwidth delay product and the congestion control state of the sender using TCP packet header information and timestamps. It then advertises the window large enough that the sender is not flow window limited.

Linux auto-tuning is basically a memory management technique in stable Linux version 2.4. The buffer size is increased and decreased on the basis of available system memory and available socket buffer space. Unlike above mentioned mechanisms, bandwidth delay product is not estimated in this technique.

There are other techniques which performs the same task as manual tuning by running a daemon [8]. It gathers the network information of the hosts

among which the connections are to be tuned and saves it in a database. Host then lookup this information while opening the connection.

All, of the above mentioned schemes are conventional buffer sizing mechanism. They consider the scenario where we have a multiple single flow TCP connections running in a system and only high capacity wired network features are monitored. We will discuss, working of ATBT scheme as, it is closest to our work of efficiently utilizing memory with network target, with the detail critical analysis, in the following subsection

2.2. Automatic TCP Buffer Tuning Algorithm

As mentioned before that ATBT algorithm is a sender based approach which regulates the send buffer dynamics. The buffer size at any point in time is configured on the basis of three algorithms which are as follows

- First algorithm determines the target buffer size on the basis of network target
- Second algorithm attempts to balance memory usage
- Third algorithm calculates the fair share of memory for a single connection which acts as a hard limit preventing excess memory usage

It is necessary to discuss the variables used by the algorithm to calculate the optimal buffer size.

AUTO_SND_THRESH A constant which actually sets the memory threshold set by the system.

sb_net_target A variable that sets the target buffer size according to cwnd as it consider it to be true estimate of bandwidth delay product.

hiwat_fair_share A variable that holds the fair share of the memory calculated using max-min fair share algorithm for a single TCP connection.

sb_mem_target A variable that suggests a send socket buffer size by taking the minimum of sb_net_target and hiwat_fair_share.

The working of ATBT algorithm is shown in figure 1, where as the memory balancing algorithm is shown in figure 2.

```
1: Initialize sb_net_target = 2 * cwnd
2: Initialize hiwat_fair_share = calculateFairShare()
3: sb_mem_target = min(sb_net_target, hiwat_fair_share)
```

(Figure 1) ATBT Algorithm

The memory balancing algorithm runs twice in a second. At the same time, fairness is achieved by employing max-min fair share algorithm. In the second iteration the left over memory is divided equally among the large connections which require more memory.

```
1: Initialize sum = 0; sum1 = 0
2:        for each connection I
3:           if (sb_net_target_i < hiwat_fair_share_i)
4:                 sum++ ; M = M + sb_net_target_i
5:                 else
6:                       sum1++
7:                 end if
8:        end for
9:        if M ≥ AUTO_SND_THRESH
10:      for each connection i
11:           If (sb_net_target_i < hiwat_fair_share_i)
12:                                 hiwat_fair_share_i = (AUTO_SND_THRESH) / (sum+sum1)
13:                 end if
14:       end for
15:    else if  M < AUTO_SND_THRESH
16: for each connection i
17:          If  (sb_net_target_i > hiwat_fair_share_i)
18:                                 hiwat_fair_share_i = (AUTO_SND_THRESH − M) / sum1
19:                 end if
20:  end for
21:       End if
```

(Figure 2) Memory Balancing Algorithm

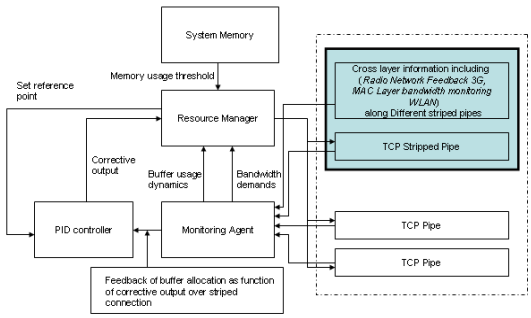
One of the obvious problems is that it is very rare that the large connections will have the same demand for memory. Thus, throughput of a

connection requiring more memory might be hindered as the excess share given to it in the second iteration might not be sufficient. If, in the second run, excess memory is assigned proportional to the BDP demand rather than dividing equally, large connections perform better than before.

At the same time, in case of wireless networks, the system might become unstable due to high frequency of congestion window (cwnd) oscillations or in correct estimate of bandwidth due to its dependency on probing interval to set sb_net_target. Moreover, in correct estimation might also result due to shrinkage of cwnd occurring because of link level losses, common in wireless environment, as mentioned in the introduction. Hence, we say that cwnd might not be an efficient estimate of BDP for a TCP connection running on wireless.

3. Proposed System

The basic assumption in our system is, existence of bottleneck at the gateway connecting the wired infrastructure and the wireless link. As a result, the performance of the TCP connections is dependent, on the dynamics of bandwidth available at the gateway. Our design, considers the constraint imposed by the system on memory usage, by defining a threshold under which all TCP connections can operate. Likewise, same kind of threshold on memory is used in [10] in order to avoid the exhaustion of mbuf clusters in the system. In figure 3, we show the flow of our system. The functionality of our design rests on the availability of memory space from the connections not utilizing assigned space to its maximum. And, regulation of buffer in our system is carried out by PID controller which suggests corrective output, which is analyzed with the available memory space, accordingly applied to the buffer. The feedback for the controller is the assigned buffer size. In simple words, the controller is managing the buffer size according to the dynamics of the available space in the system. In this section, we will define and describe the basic elements of our system.



(Figure 3) Flow of dynamic buffer size allocation for striped connection

3.1. Monitoring Agent

The monitoring agent is responsible for monitoring multiple TCP connections and maintaining variables related to them. The variables are updated after certain interval of time. For each TCP connection following variables are monitored.

AvgBuffOccup	A variable maintained for a single connection by the agent is calculated using *EWMA (Exponential Weighted Moving Average)* of ***BuffOccup*** variable. The goal is to estimate the buffer occupancy during the lifetime of the TCP connection.
BuffOccup	A monitoring variable that contains the current value of buffer occupancy of a single TCP connection.
Dev	A variable maintained to calculate the current deviation of ***BuffOccup*** from ***AvgBuffOccup***
AvgDev	A variable maintained by the agent for a single connection calculated using EWMA of ***Dev***. The concept behind

	maintaining such variable is to estimate the maximum deviation from the *AvgBuffOccup* by giving weight to the latest observation of Dev.
AssignedBuffSize	A variable containing the current buffer allocation for a single connection
proportion	A variable that is used by the resource manager, gives the proportion of connection estimated throughput over aggregate throughput demand of all the connections at the host

On the global level we maintain two variables for the whole system.

AggSpace	A global variable that is maintained to estimate the overall unused space based on the estimation of monitoring variables

It is necessary to describe certain conventions that we have used in the algorithms that will be described in the subsections of this section. We represent current time with the variable t. Then the value of any variable Var, for a connection represented by i at time t, is then represented as Var_{it}.

Figure 4 shows the algorithm running in the monitoring agent. The bandwidth changes are gathered using cross layer communication i.e. Radio Network Feedback in case of 3G as used in [15] and in case of IEEE 802.11 WLAN as monitored by MAC layer, wireless link is assumed to be bottleneck. These changes will be used to calculate BDP along a certain path. Thus, our system does not use cwnd to estimate the network demand. α and β are smoothing factors for EWMA and weights for the current observation. In case of our case, in order to give enough weight to historical observations the values choosen for α and β is 0.3.

1: *for each connection i*
2: $AvgBuffOccup_{it} = \alpha \times BuffOccup_{it} + (1-\alpha) \times AvgBuffOccup_{it-1}$
3: $Dev_{it} = BufferOccup_{it} - AvgBuffOccup_{it}$
4: $AvgDev_{it} = \beta \times Dev_{it} + (1-\beta) \times AvgDev_{it}$
5: $AvailableSpace_{it} = AssignedBuffSize_{it} - (AvgBuffOccup_{it} + AvgDev_{it})$
6: $AggSpace_t = AggSpace_t + AvailableSpace_{it}$
7: $AggregateBandwidth_{it} += Bandwidth_{it}$
8: end for
9: *for each connection i*
10: $proportion_i = \frac{Bandwidth_{it}}{AggregateBandwidth_{it}}$
11: end for

(Figure 4) Monitoring Agent Algorithm

3.2. Resource Manager

It is responsible for allocating extra memory space to adjust the assigned buffer size according to the current available memory space and the network memory demands. In brief, it consults the controller, running PID algorithm to regulate the assigned buffer for striped connection. Primarily, the PID (proportional integral derivative) controller is used in control engineering to control the measurable process variable in a factory by a constant feedback from the process. Algorithm running in the controller, then suggests a corrective output in order to bring the measurable process variable close to the reference point, set in the algorithm.

In network engineering realm, PID algorithm is often seen to be used in AQM (Active Queue Management) to manage the length of an outbound queue in a router by selectively dropping the packets to bias the behavior and performance of connections transiting the router during times of congestion [13].

During the life time of the connections there can be two scenarios: high network demand, low network demand such that the assigned buffer is in excess. In case of first scenario, first of all the threshold of the PID algorithm is set to the current network demand that is to be met. Then the

available space is obtained from the monitoring agent. Figure 5 shows the algorithm for resource allocation.

1: *Execute* $SetThresholdPID(BandwithDemand_t)$
2: *Initialize* $AvailableSpace = AggSpace_t$
3: *Initialize* $output = PID\_output(AssignedBuffSize_t)$
4: *if* $output - AvailabeSpace < 0$
5: $AssignedBuffSize_{t+1} = AssignedBuffSize_t + AvailableSpace$
6: *elseif* $output - AvailabeSpace \geq 0$
7: $AssignedBuffSize_{t+1} = AssignedBuffSize_t + output$
8: *end if*
9: *for each connection i*
10: $AssignedBuffSize_{it+1} = (Threshold\_Mem - AssignedBuffSize_{it+1}) \times proportion_i$
11: $AllocateBuffSize(AssignedBuffSize_{it+1})$
12: end for
13: *Execute* $AllocateBuffSize(AssignedBuffSize_{t+1})$

(Figure 5) Resource allocation algorithm case 1

PID algorithm suggests the corrective action that is necessary to be taken by the system in order to meet the demand of the network that is set. But, as the system is constrained by system memory, the corrective action at each cycle suggested for the assigned buffer of the striped connection cannot be as such directly applied. As a result, the output is compared with the available space at the moment from the already running connections. The allocation is suggested to be applied directly if there is enough available space. The additional memory is allocated according to the available space at the present cycle. In the end, the remaining space after the allocation of memory to the striped connection is proportionally re-assigned to other remaining connections.

In case of second scenario, at the end of step 4, the new buffer size will be calculated and assigned, as a result at step10; extra memory is proportionally allocated to the remaining TCP connections in proportion to estimated bandwidth demands from the network. At the same time, single transport connections can also estimate BDP (Bandwidth Delay Product) not necessarily on the basis of cwnd as in ATBT.

PID algorithm as shown in figure 6 generates corrective out put on the basis of three parameters

- Correction suggested in proportion to the error
- Aggregated error accumulated over a certain period of time
- Error difference between the present and the last error, thus giving the rate of error in feedback

Each of these parameters are multiplied by tunable constants k_p, k_i , k_d and then added together to generate the output for the controller. There are various sophisticated mathematical mechanisms to tune these constants. This is left as our future work. For this paper we have selected the constants which ensure stability of PID in most of the cases which are 0.05, 0.01, and 0.04 respectively. Threshold is set as in ATBT i.e. $2 \times BDP$. For a striped, connection bandwidth demand is aggregated over all the micro flows.

1: *procedure* $SetThresholdPID(BandwithDemand_t)$
2: $Threshold = 2 * BandwidthDemand * RTT$
3: *end procedure*
4: *procedure* $PID\_output(AssignedBuffSize_t)$
5: $Error_t = Threshold\_PID - AssignedBuffSize_t$
6: $output = k_p \times Error_t + k_i \times \sum_{i=1}^{t} Error_t + k_d \times (Error_t - Error_{t-1})$
7: *return output*
8: *end procedure*

(Figure 6) PID algorithm

4. Experiment

The first subsection presents the network topology used for the simulation. In the second subsection, the

simulation scenarios are discussed along with their results.

4.1. Network Topology

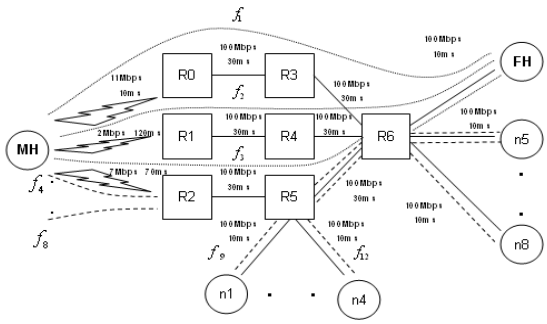
In this section, we present the topology used for our simulation. A mobile host (MH) uses transport layer striped connection, collectively, to transfer a file of size 10MB to a fixed host (FH) using FTP application. The data is striped along 3 wireless links

(a) MH – R0, with data rate of 11Mbps and propagation delay of 10ms in case of WLAN, where R0 is the access point or router for the link

(b) MH – R1, with data rate of 2Mbps and propagation delay of 120ms in case of GEO satellite network, where R1 is the access point or router for the link

(c) MH – R2, with data rate of 7Mbps and propagation delay of 70ms in case of 3G HSUPA (high speed uplink access), where R2 is the access point or router for the link

In figure 7, flows $f_1 \cdots f_3$ are the micro flows of a major TCP connection supporting transport layer stripping. As our mechanism is independent of the transport protocol used, for our simulation we ran TCP-SACK at micro flow level. Besides micro flows, there are other TCP flows, $f_4 \cdots f_8$, running on a mobile host which will vary according to the simulation scenarios discussed later. To simulate the background internet traffic, we ran 2 TCP flows and 2 UDP flows,f9 … f12, originating from sources $n1 \ldots n4$ with the ultimate destinations $n5 \ldots n8$ respectively. The access routers are connected with the background internet routers R_3, R_4, R_5 with 100Mbps data rate and 30ms propagation delay links connecting these routers. Ultimately, the backbone routers are connected to the gateway R_6 which is connected to FH with a link of 100Mbps data rate and 10ms propagation delay. Also, destination nodes for background internet flows are connected with R_6.



(Figure 7) Simulation topology

The entire simulation is carried out in ns-2 [14]. We have implemented our own module for monitoring the connections and maintaining global variables for entire system, as well as, per connection variables. Besides the monitoring module, a memory allocation agent is implemented which communicates with TCP, FTP application and assigns buffer size according to our proposed mechanism.

4.2. Simulation Results

We consider two scenarios for our experiment. For the first scenario, the performance is analyzed by ensuring more than enough available memory for the connections on the host by setting high memory threshold. In contrast, for the second scenario, we scrutinize performance when a host is constrained on available memory, by setting low memory threshold for the connections. For each scenario, we varied the single flow connections to eight on a mobile host, in order to prove the scalability of our proposed

mechanism.

The duration of simulation is 400 seconds. And, the single flows start after 10 seconds from the beginning of simulation. The configuration for the simulation of two scenarios is summarized in Table 1.

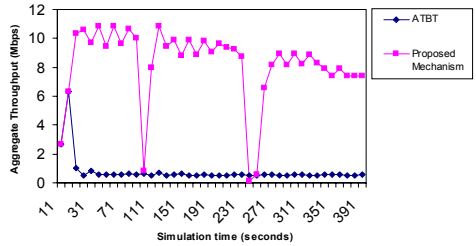
(Table 1) system configuration

	Memory Threshold (Mbytes)	File Size (Mbytes)	Default Socket Buffer Size (Kbytes)	Maximum Payload per Packet (Bytes)
High Memory Scenario	100	10	128	1460
Low Memory Scenario	2	10	128	1460

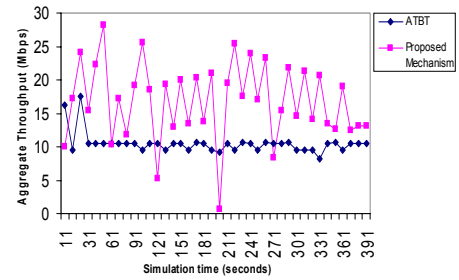
The performance is analyzed by observing the aggregate throughput of the striped connections both, at sender and receiver end, after every 10 seconds. We varied the bandwidth after 50 seconds, in order to simulate bandwidth fluctuations. At the same time, the buffer is monitored after 20 seconds. The results are discussed in the following subsections.

4.2.1 High Memory Scenario

Figure 8 and 9, respectively, shows the results at sender and receiver end for 3 micro flows constituting major stripped connection and 1 single flow, for an aggregate of 4 connections in the system. It is blatant, that our proposed mechanism simply out class ATBT mechanism in terms of enabling connections to open up their window to achieve maximum throughput. This trend is observable at both ends. The sender appropriated space from the 4th connection and maximized its sending throughput by transferring more data at a time. The bad performance of the connection under ATBT is expected, due to estimation of BDP on the basis of congestion window and its unnecessary cut down due to link level losses.



(Figure 8) Receiver side throughput for four connections

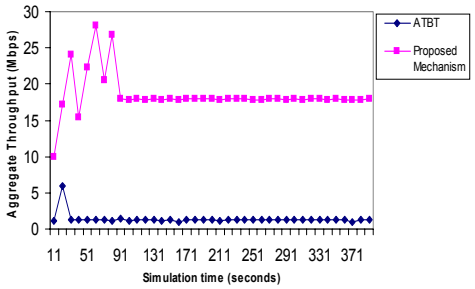


(Figure 9) Sender side throughput for four connections

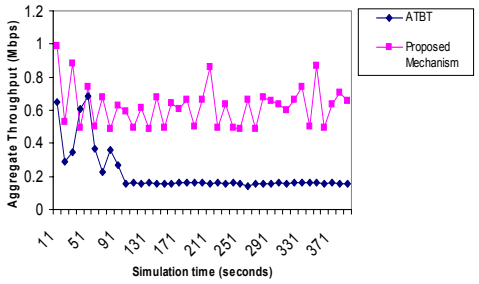
Likewise, the same behavior is observed even if the number of single flow connections is increased to an aggregate of 8 connections running at the host. Figure 10 and 11 shows the sender and receiver perceived aggregate throughput. The throughput achieved is lowered slightly which is due to high number of connections sharing common pool of memory. In addition, increased congestion in the network also contributed to lower aggregate throughput at the receiver.

Figure 12 shows the buffer size of the striped connection socket. The size of the buffer allocation remains higher than the allocation by ATBT. At the same time, memory constraint imposed by the system is preserved. The buffer allocations in case of

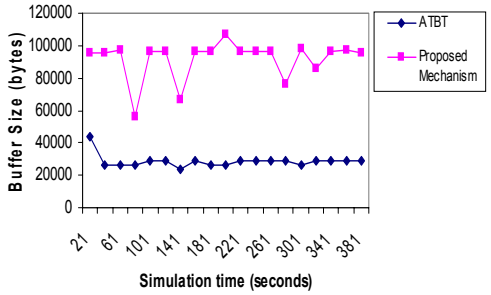
first scenario, for both ATBT and proposed mechanism, are also higher than the second scenario. This is due to low number of connections sharing common pool of memory, in case of first scenario. As a result higher amount of data is sent in flight and congestion window opens up fully leading to high throughput in case of proposed mechanism.



(Figure 10) Receiver side throughput for eight connections



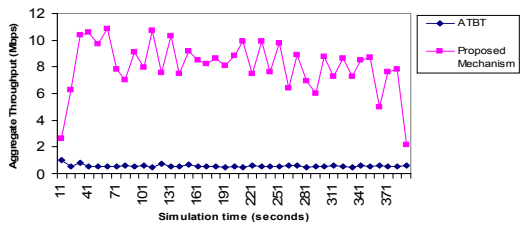
(Figure 11) Sender side throughput for eight connections



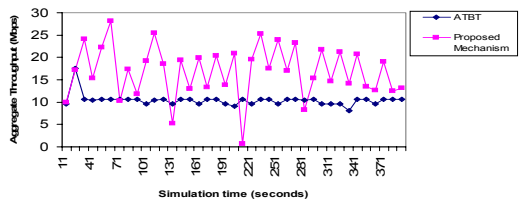
(Figure 12) Buffer size sender side for eight connections

4.2.2 Low Memory Scenario

In case of second scenario, for 4 aggregate number of connections running at the host. A large gap in achieved aggregate throughput, between the proposed and ATBT mechanism at the sender's end is seen. Figure 13 and 14 shows receiver and sender side aggregate throughput. Clearly, because of low memory the connections were unable to send large enough data. While, our proposed mechanism performed well in achieving high aggregate throughput at both end. It can be seen that at sender end, throughput remained nearly constant in case of ATBT while our mechanism showed great variation during the whole simulation time. This behavior is expected, as our mechanism, estimates available space from other connections. Thus, in this case we have only one single connection running at the host. As a result, aggregate throughput of the stripped connection is a function of the available space from 1 single connection in this case.



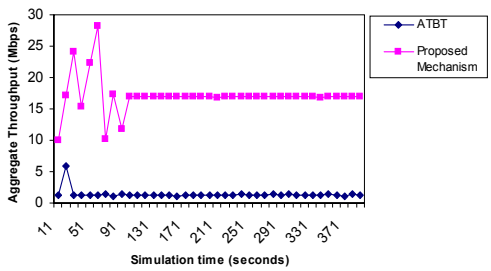
(Figure 13) Receiver side throughput for four connections



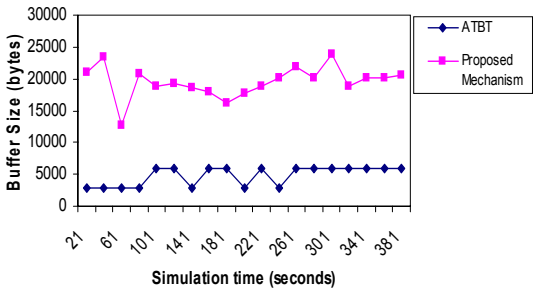
(Figure 14) Sender side throughput for four connections

In case of increasing the connections to 8 at the host, the same trend is observed, but lower than the first case. It is evident, in this case at the sender end; the stripped connection throughput is a function of the average available space across 5 single flow connections. In the beginning the throughput was high, as the time passes when the other 5 single flows begins sending data over the network, in that case the buffer allocations for the connections are reduced and contributes to nearly constant observed aggregate throughput. At the same time, at the receiver end ironically in the beginning a surge is seen in achieving aggregate throughput. The aggregate throughput achieved by sender and receiver is of the order of Kbytes, as our graph is drawn in the scale of Mbytes due to which the line nearly touches the x-axis. In the end again, due to congestion augmented with lower amount of data send by the sender, dramatically lowers the aggregate throughput.

Figure 15 and 16 shows the buffer allocation of the striped connection socket which is higher than ATBT is case of low memory scenario. While, the behavior observed in this case is similar to high memory scenario i.e. in case of 4 connections the allocation is higher than 8 connections regardless whether ATBT or proposed mechanism is applied. And, our proposed mechanism allocates higher buffer size than ATBT.



(Figure 15) Receiver side throughput for eight connections



(Figure 16) Buffer size sender side for eight connections

5. Conclusion

In this paper we propose a dynamic buffer tuning technique by applying PID controller, appropriating unused space from other connections and using cross layer communication for acquiring available bandwidth information. The technique is devised for striped transport layer connection in which a single socket buffer is shared by micro flows running on diverse wireless paths. In the end, we show that our proposed mechanism performs better than ATBT for achieving high aggregate bandwidth both at sender and receiver end.

Acknowledgment

This research was supported by the MKE (Ministry of Knowledge Economy), Korea, under the ITRC (Information Technology Research Center) support program supervised by the IITA (Institute of Information Technology Advancement) (IITA-2009-(C1090-0903-0011)).

Corresponding author is Prof. Eui-Nam Huh.

References

[1] M. Stemm and R. Katz. Vertical handoffs in wireless overlay networks. Mobile Networks

and Applications, 3(4): 335 – 350, 1998.

[2] H. Hsieh and R. Sivakumar. A transport layer approach for achieving aggregate bandwidths on multi-homed mobile hosts. Wireless Networks, 11:99-114, 2005.

[3] M.Zhang, J. Lei, A. Krishnamurthy, L. Peterson and R. Wang. A transport layer approach for improving end-to-end performance and robustness using redundant paths. Proceedings of the USENIX Annual Technical Conference, 8-8, 2004

[4] M. Tonouchi, H. Mineno, S. Ishihara, O. Takahashi, and T. Mizuno. A study on retransmission control of multipath-extended TCP. IPSJ SIG-MBL: Mobile Computing, 28(27), March 2004.

[5] S. Saito, Y.Tanaka, M.Kunishi, Y.Nishida and F. Teraoka. AMS: An adaptive TCP bandwidth aggregation mechanism for multi-homed mobile host. IEICE transaction of information and systems, 12:2838-2847, Dec. 2006

[6] W. Bethel, B. Tierney, J. Lee, D. Gunter, and S. Lau. Using high-speed wans and network data caches to enable remote and distributed visualization. Proceeding of the IEEE Supercomputing 2000 Conference, 2000.

[7] P. Steenkiste. Adaptation Models for Network-Aware Distributed Computations. In 3rd Workshop on Communication, Architecture, and Applications for Network-based Parallel Computing, 1999.

[8] T. Dunigan, M. Mathis, B. Tierney. A TCP tuning daemon. Supercomputing, ACM/IEEE 2002 Conference, 16-22, Nov 2002.

[9] A. Bakre and B.R. Badrinath. I-TCP: indirect TCP for mobile hosts. Proceedings of the 15th International Conference on Distributed Computing Systems, 136-143, June 1995.

[10] J. Semke, J. Mahdavi and M. Mathis. Automatic TCP buffer tuning. Proceedings of ACM SIGCOMM, 315 – 323, Sep 1998.

[11] E. Weigle and W. Feng. Dynamic Right-Sizing: A Simulation Study. Proceedings of IEEE International Conference on Computer Communications and Networks, 2001.

[12] E. Weigle and W. Feng. A comparison of TCP automatic tuning techniques of distributed computing. HPDC-11, 265-272, July 2002.

[13] F. Yanfie, R.Fengyuan and L. Chuang. Design a PID controller for active queue management. ISCC, 2:985-990,2003.

[14] The Network Simulator ns-2. http://www.isi.edu/nsnam/ns.

[15] N. Möller, I. C. Molero, K. H. Johansson,J. Petersson, R. Skog and Å.Arvidsson. Using radio network feedback to TCP performance over cellular networks. CDC-EEC '05, 7437-7439, Dec 2005.

◐ 저 자 소 개 ◑

파 라 즈(Faraz Idris Khan)

2005년 National University of Sciences and Technology 졸업(학사)
2008년 경희대학교 대학원 컴퓨터공학과 졸업(석사)
관심분야 : 무선통신, 4G, 자원관리, 이동통신 etc.
E-mail : faraz@khu.ac.kr

허 의 남(Eui-Nam Huh)

1990년 부산국립대학교 컴퓨터공학과 졸업(학사)
1995년 The University of Texas at Arlington 컴퓨터공과 졸업 (석사)
2002년 The Ohio University 컴퓨터공학과 졸업 (박사)
2005년 9월~현재 경희대학교 컴퓨터공학과 부교수
관심분야 : 클라우드, 분산컴퓨팅, 센서 네트워크, 보안. etc
E-mail : johnhuh@khu.ac.kr