AppTCP: The Design and Evaluation of Application-based TCP for e-VLBI in Fast Long Distance Networks

Guodong Wang[a,b], Yulei Wu[b,*], Ke Dou[a,b], Yongmao Ren[b], Jun Li[b]

[a] *Graduate University of Chinese Academy of Sciences, Beijing* 100049*, China,*
[b] *Computer Network Information Center, Chinese Academy of Sciences, Beijing* 100190*, China,*

Abstract

Electric Very Long Baseline Interferometry (e-VLBI) is a typical astronomical interferometry used in radio astronomy. It allows observations of an object that are made simultaneously by many radio telescopes to be combined, emulating a telescope with the size equal to the maximum separation between the telescopes. The main requirements of transporting e-VLBI data are the high and constant transmission rate. However, the traditional TCP and its variants cannot meet these requirements. In an effort to solve the problem of transporting e-VLBI data in fast long distance networks, we propose an application-based TCP (AppTCP) congestion control algorithm, using Closed-Loop Control theory to keep the stable and constant transmission rate. AppTCP can swiftly reach the required transmission rate by increasing the congestion control window, and keep the transmission rate and allows the other TCP flows to share the remaining bandwidth. We further conduct extensive experiments in both fast long distance network test-bed and actual national networks (i.e., from Beijing to Shanghai in China) and international networks (i.e., from Hongkong in China to Chicago in USA) to evaluate and verify the performance and effectiveness of AppTCP. The results show that the AppTCP can effectively utilize the link capacity and maintain the constant rate during the data transmission, and its performance significantly outperforms that of the existing TCP variants.

Keywords: Bulk Data Transmission, Congestion Control, High Speed Network, e-VLBI, TCP

1. Introduction

The efficiency of bulk data transmission over fast long distance networks (FLDnets) is critically required for many scientific applications, e.g., e-VLBI transmissions [1]. In order to support the e-VLBI experiments conducted by global radio telescopes, high volume e-VLBI data need to be transported globally. Although we can provide a link with a high bandwidth for e-VLBI data transmission, we still encounter the short-term congestion resulted from the burst flow in the IP link which greatly degrades the performance of transporting the e-VLBI data. That is because the current version of TCP cannot meet the requirements of e-VLBI data transmission, including reaching a high transmission rate swiftly by aggressively grabbing the available bandwidth from the link, maintaining the transmission rate as long as possible, and sharing the bandwidth friendly with other flows once the required transmission rate is obtained.

A number of improved TCP variants have been proposed to increase the efficiency of TCP in FLDnets, e.g., HSTCP [2], STCP [3], HTCP [4],Westwood [5], BIC [6], CUBIC [7], Yeah [8], ACP [9], etc. Although these TCP variants have achieved success in certain issues, there are still many challenges that cause researchers widespread concerns [10], such as the efficiency, stability and fairness of TCP for bulk data transmission in FLDnets. To the best of the authors' knowledge, there is no effective approach proposed in the current literature to meet the data transmission requirements of e-VLBI. Although Circuit TCP [11] made a step in this field, however, it completely removes the congestion control mechanism, which makes it only suitable for dedicated optical networks.

In an effort to tackle the problems of e-VLBI data transmission in fast long distance networks, this paper presents a new application-based TCP (AppTCP) based on the Closed-Loop Control theory [12]. With the proposed protocol, the transmission can swiftly reach the required rate by increasing the congestion control window (cwnd) aggressively; in addition, it keeps the transmission rate and allows the other TCP flows to share the remaining bandwidth. We further implement AppTCP on Linux platform, and conduct extensive experiments in the both FLDnet test-bed and the actual national and international networks to verify the efficiency of the proposed AppTCP. The experimental results show that the performance of AppTCP in bulk data transmission in FLDnets significantly outperforms that of the existing TCP variants.

The remainder of the paper is organized as the follows. Section 2 describes the related work. Section 3 presents the design of AppTCP algorithm. Extensive experiments of AppTCP in the FLDnet test-bed and in actual networks are conducted in Section 4 and Section 5, respectively. Finally, Section 6 concludes this study.

*Corresponding author
Email addresses: wangguodong@cstnet.cn (Guodong Wang), wuyulei@cstnet.cn (Yulei Wu), douke@cstnet.cn (Ke Dou), renyongmao@cstnet.cn (Yongmao Ren), jlee@cstnet.cn (Jun Li)

2. Related work

To increase the transmission efficiency of the traditional TCP in FLDnets, a number of TCP variants have been proposed. The proposed protocols can be classified into the following categories according to the congestion control strategies: Loss-based Congestion Algorithm (LCA), Delay-based Congestion Algorithm (DCA) and Compound Congestion Algorithm (C-CA) which combines the LCA and DCA. LCA adopts the packet loss as the only indicator of congestion, while DCA uses the variation of Round Trip Time (RTT) to reflect the network's condition. Table 1 shows the classified description of the popular TCP variants.

Table 1: Classification of TCP variants

Categories	Representatives
LCA	STCP [3], HSTCP [2], BIC [6], CUBIC [7]
DCA	Vegas [13], ACP [9]
CCA	CTCP [14], Yeah [8]

STCP [3] altered standard TCP's AIMD (Additive Increase Multiplicative Decrease) congestion avoidance scheme to MIMD (Multiplicative Increase Multiplicative Decrease). Specifically, STCP increases its cwnd by 0.01 times of the current size on each received ACK (acknowledgement) and decreases it to its 0.875 times upon a packet loss. In contrast, HSTCP [2] still adopts the AIMD scheme, but it polishes the increase and decrease parameters.

BIC [6] adopted a binary search growth and linear growth to adjust its cwnd. Binary search growth is similar to the classical binary search algorithm. If the distance between the current cwnd and the desired cwnd is too large, BIC adopts the linear growth strategy instead of the binary search growth. However, BIC's cwnd updating rate appears too aggressive, especially under short RTT or in low speed networks. CUBIC [7] is the revised version of BIC. It simplified the cwnd updating algorithm of BIC to increase the TCP friendliness.

Vegas [13] was a typical DCA TCP variant and its cwnd is adjusted by the RTT variation of the link. If the RTT is gradually increasing, the link is considered to be congested and the cwnd will be decreased; otherwise, the congestion is alleviated and the cwnd will be increased.

ACP [9] used the estimation of the bottleneck queue size and fairness ration to achieve high utilization and friendly share network resources.

CTCP [14] combined the loss based CAA (Congestion Avoidance Algorithm) and delay based CAA to achieve high bandwidth utilization and TCP friendliness. Delay based CAA is achieved by introducing dwnd (delay window), and then the sending window is controlled by both cwnd and dwnd. The delay-based component has a scalable window increasing rule that not only probes the link capacity, but also reacts early to avoid congestion by sensing the changes of RTT.

In [8], the network has two status: Fast and Slow. In Fast state, the increase of cwnd is expeditious just like STCP, while in Slow state, cwnd increases moderately as Reno dose to avoid network congestion.

Although these TCP variants have achieved certain success in their respective target areas [15, 16, 17], there is no effective approach which is proposed to meet the requirements for applications in e-VLBI environments. Therefore, in this paper we target this problem to propose an AppTCP to facilitate the data transmission for e-VLBI.

3. The proposed application-based TCP

3.1. Closed-Loop Control system

In this section, we first introduce the Closed-Loop Control system [12], a popular control system theory in Mechanical Engineering which can facilitate the design of AppTCP. A typical Closed-Loop Control system can be represented by the general block diagram shown in Fig. 1, where a *Feedback* component is applied together with the *Input*. The difference between the *Input* and *Feedback* is applied to the *Controller*. In responding to this difference, the *Controller* acts on the *Process* forcing *Output* to change in the direction that will reduce the difference between the *Input* and the *Feedback*. This, in turn, will reduce the *Input* to the process and result in a similar change in *Output*. This chain of events continues until a time is reached when *Output* approximately equals *Input*.

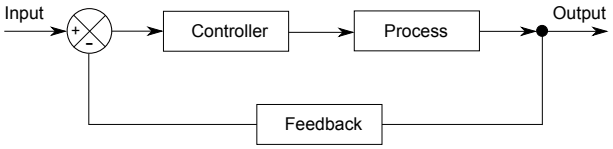


Figure 1: Typical Closed-Loop Control System

3.2. The theory of AppTCP

A Closed-Loop Control system is able to regulate itself in the presence of disturbance or variations in its own characteristics, and thus has obvious advantage in keeping the output at a constant rate. We, therefore, apply the Closed-Loop Control system theory to the design of AppTCP as shown in Fig. 2.

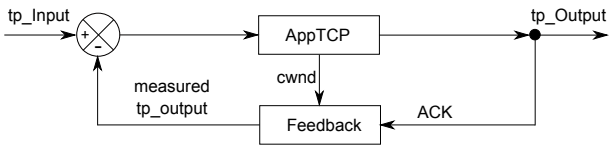


Figure 2: Our Closed-Loop Control System

In this model, *tp_Input* (throughput input) and *tp_Output* (throughput output) represent the required transmission rate of an application and the actual transmission rate we can obtain, respectively. AppTCP acts as both the *Controller* and the *Process* in our Closed-Loop Control system. The feedback of actual transmission rate (measured *tp_Output*) is applied with the required transmission rate *tp_Input*, and the difference between the *tp_Input* and measured *tp_Output* is applied to the AppTCP.

In responding to this difference, AppTCP acts on its congestion control algorithm, which forces *tp_Output* to change in the direction that will reduce the difference between the *tp_Input* and the measured *tp_Output*. This chain of events continues until a time is reached when *tp_Output* approximately equals *tp_Input*.

3.3. The implementation of AppTCP

In implementation of the AppTCP, the *tp_Input* can be obtained from the applications. To calculate the measured *tp_Output*, we get the current RTT (*RTT_current*) from the ACKs which are sent from the client, and the current cwnd from the AppTCP. For a cwnd number of packets are transmitted per RTT, the measured tp_Outut [13] can be described as

$$tp\_Output = \frac{cwnd \times MSS}{RTT\_current} \tag{1}$$

where the *MSS* is the Maximum Segment Size, with the default value of 1460 byte in the standard TCP [18].

Similar to the standard TCP [18], the Slow Start mechanism is adopted to aggressively grab the available bandwidth of the link, which means that AppTCP doubles its cwnd from the initial size within the time of RTT. In contrast, the Closed-Loop Control mechanism is introduced to keep the constant transmission rate, with which the cwnd of AppTCP is dynamically adjusted by the difference between the tp_Input and the tp_Output. Specifically, the cwnd will be changed with the aim of reducing the difference between the *tp_Input* and the *tp_Output*. The detailed congestion control algorithm of AppTCP can be given by

$$cwnd_{i+1} = \begin{cases} cwnd_i + 1 & \text{Slow Start} \\ cwnd_i \pm \frac{1}{AI\_cnt} & \text{Closed-Loop Control} \end{cases} \tag{2}$$

where the index i denotes the reception of the ith ACK and the *AI_cnt* is the Additive Increase count which is introduced to smooth the cwnd adjusting rate [3]. The cwnd will not be adjusted until *AI_cnt* number of ACK is received in the Closed-Loop Control phase of AppTCP. From Eq. (2), we can find that both the Slow Start and Closed-Loop Control result in a discrete exponential increase with RTT, but their bases are different. In Slow Start, the base is 2, while in Closed-Loop Control, the base is $1 \pm \frac{1}{AI\_cnt}$. The reason why we use the $\pm$ symbol is that the value of $(tp\_Output - tp\_Input)$ could be positive or negative. If the value is positive, the cwnd should be decreased, otherwise, the cwnd is increased. Consequently, the cwnd evolution in time can be given by

$$cwnd(t) = \begin{cases} 2^{\frac{t}{RTT}}, & \text{Slow Start} \\ \gamma \pm \frac{t-t_\gamma}{RTT} \times \frac{cwnd}{AI\_cnt}, & \text{Closed-Loop Control} \end{cases} \tag{3}$$

where t represents the elapsed time with the unit of RTT. t_γ and γ are the time and the cwnd, respectively, when AppTCP exits the Slow Start phase. Thus, it is easy to get $t_\gamma = RTT log_2 \gamma$. From Eq. (3), we can find that the larger the cwnd (the higher throughput), the greater the cwnd adjustment rate, which is helpful for AppTCP to effectively increase the cwnd in high-speed networks.

Algorithm 1: The congestion control algorithm of AppTCP

```
Initialization:
RTT_base ⟵ ∞        snd_ssthresh ⟵ ∞
/* get tp_Input from the application          */
tp_Input ⟵ application
On each ACK:
begin
    /* get the current RTT.                   */
    RTT_current ⟵ RTT_us
    RTT_base ⟵ min(RTT_base, RTT)
    /* calculated from Eq.(1)                 */
    tp_Output = cwnd × MSS/RTT_current
    if cwnd < cwnd_base then
        /* Slow Start                         */
        cwnd++
    else
        /* Closed-Loop Control                */
        if tp_input < tp_output then
            if cwnd_cnt ≥ AI_cnt then
                cwnd++
                cwnd_cnt = 0
            else
                cwnd_cnt++
            end
        else
            if cwnd_cnt ≥ AI_cnt then
                cwnd−−
                cwnd_cnt = 0
            else
                cwnd_cnt++
            end
        end
    end
end
Packet loss:
begin
    /* calculated from Eq.(4)                 */
    cwnd = tp_input × RTT_base/MSS
end
```

The pseudo code of AppTCP is presented in the Algorithm 1. On the initialization, the parameters are assigned with the values, and the AppTCP starts up from the Slow Start phase. On the receiving of each ACK, the *RTT_current* and the base RTT (*RTT_base*) can be obtained. The *RTT_base* denotes the propagation delay of the link which is used to determine the time when the AppTCP exits the Slow Start phase. The *RTT_current* represents the sum of the propagation delay and the queuing delay which is used to calculate the tp_Output in Eq. (1).

The Slow Start phase ends when the cwnd is equal to *cwnd_base*, which is the minimum cwnd (calculated using the *RTT_base*) that the AppTCP can achieve the required transmission rate. The *cwnd_base* can be expressed as

$$cwnd\_base = \frac{tp\_input \times RTT\_base}{MSS} \tag{4}$$

After that, the AppTCP enters the Closed-Loop Control phase, and the transmission rate is dynamically adjusted by the Closed-Loop Control mechanism of AppTCP. When packet loss happens, *cwnd* will be set to the *cwnd_base* and then dynamically adjusted by the Closed-Loop Control mechanism.

4. The performance of AppTCP in fast long distance network test-bed

In order to evaluate the performance of AppTCP, extensive experiments have been conducted in the both FLDnet test-bed and actual networks. This section investigates the performance of AppTCP in the FLDnet test-bed. The main components of the FLDnet test-bed include four terminals and one emulator, where the Sender 1 / Sender 2 and Receiver 1 / Receiver 2 are connected by the emulator. The emulator can control the bottleneck bandwidth, delay and packet loss probability of the links. The operating systems of all servers are CentOS with the Linux kernel of 2.6.18, and the specifications of the servers are shown in Table 2. Experiments are performed by sending 2 GB file from server to client. For AppTCP, as the Closed-Loop Control mechanism can restrain the throughput fluctuation, the *AI_cnt* is set to be 10 [3]. For the following experiments, each experiment is repeated 5 times, and thus we get 5 sets of data for each experiment. The largest and the smallest value of these data are dropped and the average value of the other three is taken as the valid data.

Table 2: Specifications of the servers in test-bed

Terminals	CPU	Memory	Disk Speed
Sender 1	CPU 2.00GHz Intel Xeon(R) (2×4 cores)	4.0GB	333MBps
Receiver 1			
Sender 2			
Receiver 2			
Emulator			

4.1. Required transmission rate vs. Actual throughput

Since the AppTCP is designed to meet the requirements of e-VLBI applications with the characteristics of constant transmission rate, the experiment is conducted to evaluate if AppTCP can effectively control the transmission rate. In this experiment, the RTT and bottleneck bandwidth of the link are set to be 300 ms and 622 Mbps, respectively. The required transmission rate varies from 100 Mbps to 600 Mbps to investigate if the actual throughput can be achieved precisely.

In Fig. 3, we compare the required transmission rate and the actual throughput we get from AppTCP. It is clearly to see that the difference between the required transmission rate and the actual throughput is quite small, which indicates that AppTCP can effectively control the transmission rate of the applications.

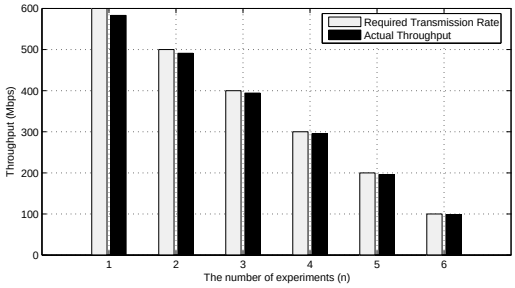


Figure 3: Required transmission rate vs. Actual throughput

4.2. RTT vs. Throughput

In this experiment, we set the bottleneck bandwidth to be 622 Mbps and varies RTT from 2 ms to 512 ms to investigate the effects of different RTTs on the efficiency of AppTCP's transmission.

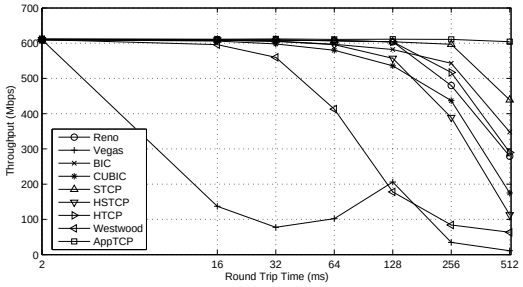


Figure 4: RTT vs. Throughput

Fig. 4 shows that with the increase of the RTT, nearly all the TCP variants show a downward trend in the throughput except for AppTCP, in which the throughput maintains around 600 Mbps. The results indicate that AppTCP achieves high efficiency for the transmission in FLDnet environments.

4.3. Link loss vs. Throughput

In this experiment, the bottleneck and RTT are set to be 622 Mbps and 300 ms, respectively, and the packet loss probability varies from 0 to 1E-4 to investigate the efficiency of AppTCP under different link loss probabilities.

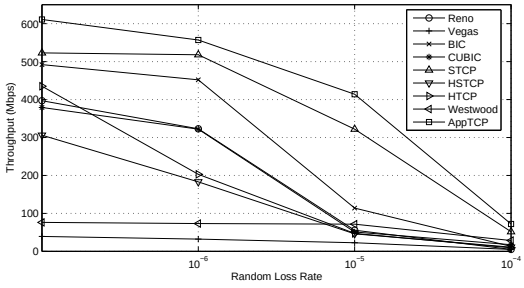


Figure 5: Link Loss vs. Throughput

Fig. 5 shows that when the loss probability is 0, AppTCP performs perfectly and the network throughput is around 610

Mbps. With the increase in loss probability, the performance of CUBIC and HSTCP degrades significantly, which indicates that they are sensitive to the link loss. The performance of AppTCP is undeniably affected by the increase of loss probability, but it is still higher than that of the other TCP variants regardless of the varying of the loss probability.

4.4. Friendliness evaluation

To evaluate the friendliness of AppTCP when it competes bandwidth with other TCP flows. Fig. 6 depicts the experiment where the RTT is fixed to be 300 ms and the bottleneck bandwidth is set to be 622 Mbps without any loss in the link. BIC is used as the competing flow because it is the default TCP variant in CentOS.

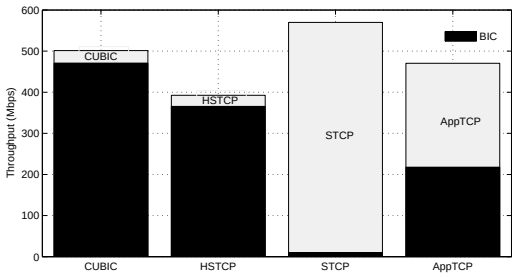


Figure 6: Friendliness evaluation

We can find from Fig. 6 that CUBIC and HSTCP are too mild when they compete with BIC. In contrast, the aggressiveness of STCP makes it grab excessive bandwidth from BIC. For AppTCP, when the transmission rate is set to be 256 Mbps, AppTCP only grabs the required 256 Mbps and shares the remaining bandwidth friendly with BIC.

5. The performance of AppTCP in actual international and national fast long distance networks

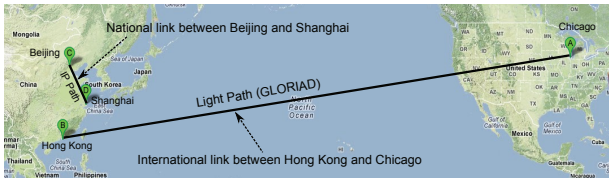
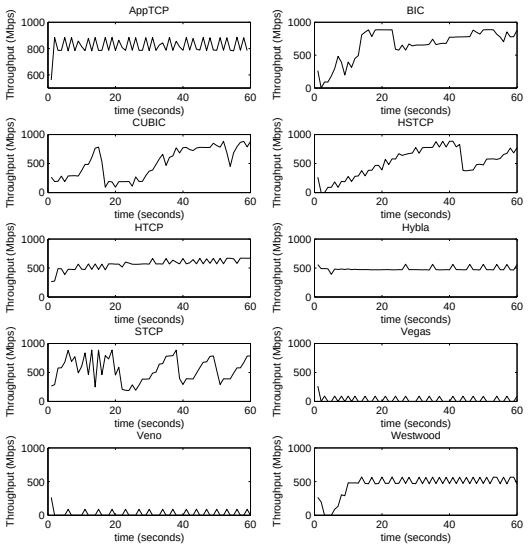


Figure 7: The actual network link for the performance evaluation of AppTCP

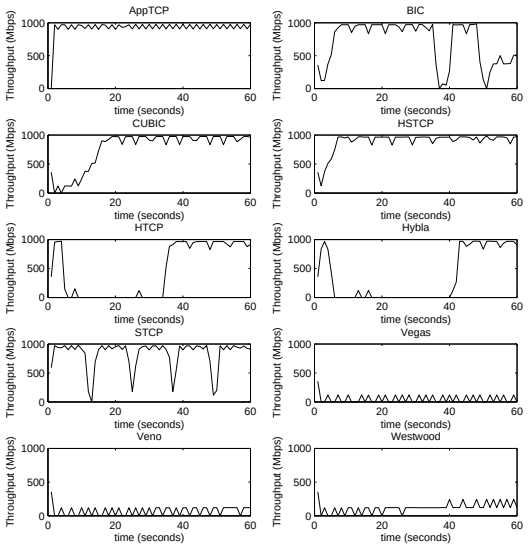
In this section, we mainly investigate the performance of AppTCP in actual network environments. To verify its effectiveness in different scenarios, we conduct our experiments both in an international network and a national network as shown in Fig. 7, respectively. The international network used in this section is the GLORIAD [19] link from Hong Kong to Chicago. The link bandwidth is 1 Gbps and the RTT is 139 ms (±1 ms). The national network adopted in this section is the link from Beijing to Shanghai. The link bandwidth is 1 Gbps and the RTT is 20 ms (±1 ms).

5.1. Dynamic transmission rate

One of the key contributions of AppTCP is that it can achieve high and stable transmission rate in FLDnets. In order to evaluate and verify this feature, the network testing tool Iperf [20] is adopted to measure the packet transmission rate. Each experiment runs for 60 seconds and the transmission rate is recorded every second. We compare the performance of all the TCP variants including BIC, CUBIC, HSTCP, etc. in Linux 2.6.18, and the results are shown in Fig. 8.



(a) Dynamic throughput in the international network



(b) Dynamic throughput in the national network

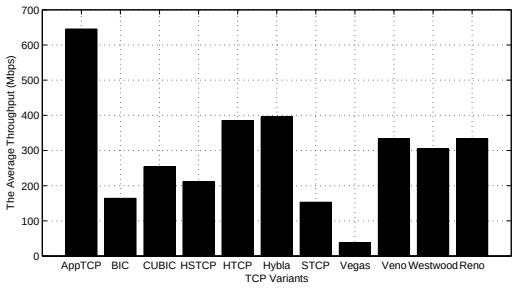
Figure 8: Dynamic throughput of AppTCP and other TCP variants in international and national networks

In the international network, the server in Chicago is installed with the AppTCP stack. It can be seen from Fig. 8(a) that

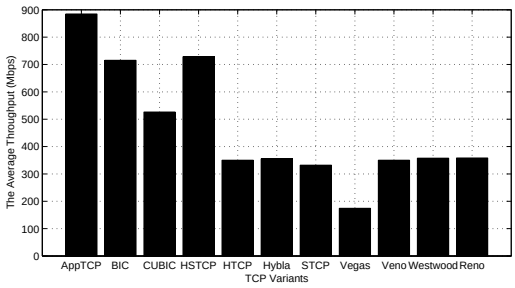
the transmission rate of AppTCP quickly rockets to around 850 Mbps at the very start of the test, and keeps the rate in the remaining of the experiment. HTCP, Hybla, and Westwood also achieve stable transmission rate, but their transmission rates are only about 500 Mbps, which is much lower than that of AppTCP. For the other TCP variants, they perform much poorer than AppTCP either in reaching high transmission rate, or in the ability to keep a constant rate. Vegas and Veno are the worst cases in this international network.

In the national network, the server in Beijing is installed with the AppTCP stack. It can be seen from Fig. 8(b) that AppTCP still quickly reaches and keeps a stable transmission rate over 900 Mbps. BIC, CUBIC, HSTCP, HTCP can also reach a high transmission rate, however, they require much longer time to go up to such a transmission rate, and the stability is not as good as the AppTCP. It is remarkable that like in the international network, Vegas and Veno remain the worst cases in this national networks. Westwood also performs poorly in this environment, and its transmission rate is much lower than that in the international network.

5.2. Average throughput



(a) Average throughput in international network



(b) Average throughput in national network

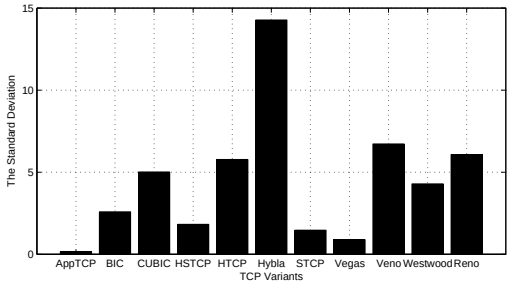
Figure 9: Average throughput of AppTCP and other TCP variants in international and national networks

Considering that FTP is widely used in bulk data transmission, in this section, FTP is employed to transport 2 GB files from Chicago to Hong Kong and from Beijing to Shanghai, respectively, to investigate the performance of AppTCP. We select VSFTP as the FTP server because it is the default FTP server in Linux CentOS. In order to obtain the accurate results, all experiments run for five times. After removing the maximum and minimum value, the mean of the other experiments is taken as the final result, which is presented in Fig. 9.

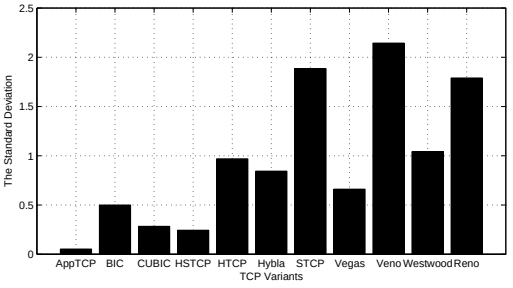
Fig. 9 shows that the performance of AppTCP is much better than the existing TCP variants both in the international and national networks. Specifically, in the international network, AppTCP can reach around 650 Mbps in average throughput, while in the national network, its average throughput is close to 900 Mbps. Hybla and HTCP perform well in the international network, since Hybla increases the cwnd update rate in networks with larger RTTs, and HTCP is designed to increase the transmission efficiency of TCP in FLDnets. The experimental results verifies their effectiveness. While in the national network BIC and HSTCP are efficient. Vegas is the worst case in transmission efficiency both in the international and national networks.

5.3. Stability

The stability can be depicted by the standard deviation of the throughput of the protocols. The standard deviation of the throughput of AppTCP as well as that of the other TCP variants are presented in Fig. 10.



(a) Standard deviation of the throughput in international network



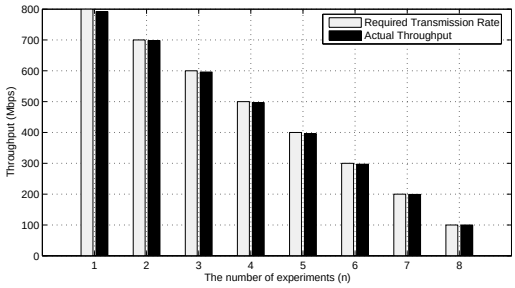
(b) Standard deviation of the throughput in national network

Figure 10: Standard deviation of the throughput

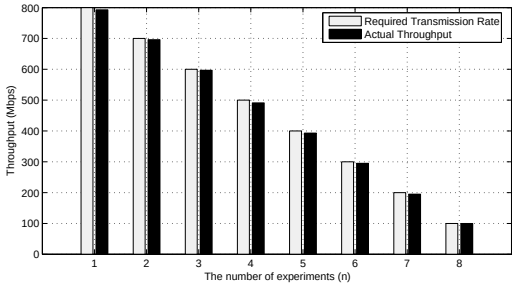
Fig. 10 shows that the standard deviation of the throughput of AppTCP is under 0.1 in the both international and national networks, which verifies that AppTCP's stability is the best among the evaluated TCP variants. It also can be seen from Fig. 10 that the stability of TCP variants in international networks is worse than that in the national network, which indicates that the RTT has significant effect on the TCP stability. Specifically, Hybla is the most unstable one in the international network, and its throughput standard deviation is close to 15. In the national network, Veno, STCP and Reno are also unstable in throughput.

5.4. Rate control

In addition to the advantages we mentioned above, another contribution of AppTCP is its ability to achieve the required transmission rate according to the requirements of the applications. To evaluate this feature, we vary the required transmission rate from 100 Mbps to 800 Mbps and compare the differences between the required transmission rate and the actual throughput in Fig. 11.



(a) Throughput comparison in international network



(b) Throughput comparison in national network

Figure 11: Required transmission rate vs. Actual throughput

Fig. 11 shows that the difference between the required transmission rates and the actual rates are quite small in the both international and national networks, which indicates that the demands of data transmission rate of the applications can be effectively satisfied using AppTCP.

6. Conclusions

In order to reduce the jitter of TCP transmission rate and meet the demands of transporting e-VLBI data, this paper has proposed the AppTCP for FLDnets. The Closed-Loop Control theory is adopted in AppTCP to keep the stable transmission rate. To evaluate the performance of AppTCP, extensive experiments have been conducted in the both FLDnet test-bed and in the actual networks, including an international network from HongKong in China to Chicago in USA and a national network from Beijing to Shanghai in China; the performance of the AppTCP has been compared with that of many existing TCP variants. Experimental results have verified that AppTCP can keep a high and constant transmission rate based on the requirements of applications, and can effectively utilize the link capacity and obtain the desirable transmission stability. In addition, AppTCP can also achieve a high degree of friendliness when the required transmission rate is satisfied. This work has its significance for reference of investigating the performance of the TCP owing to its comprehensive comparison with many existing TCP variants in both test-bed and actual networks.

Acknowledgment

This work is partially supported by the National Program on Key Basic Research Project (973 Program) under Grant No. 2012CB315803, the National Key Technology Research and Development Program of the Ministry of Science and Technology of China under Grand No. 2012BAH01B03 and the "Strategic Priority Research Program" of the Chinese Academy of Sciences under Grant No. XDA01020304.

References

[1] *eVLBI http://www.evlbi.org/.*
[2] S. Floyd, "HighSpeed TCP for large congestion windows," 2003.
[3] T. Kelly, "Scalable TCP: Improving performance in highspeed wide area networks," *ACM SIGCOMM Computer Communication Review*, vol. 33, no. 2, pp. 83–91, 2003.
[4] D. Leith and R. Shorten, "H-TCP: TCP for high-speed and long-distance networks," *PFLDnet'04*, 2004.
[5] S. Mascolo, C. Casetti, M. Gerla, M. Sanadidi, and R. Wang, "TCP westwood: Bandwidth estimation for enhanced transport over wireless links," in *Proceedings of the 7th annual international conference on Mobile computing and networking*. ACM, 2001, pp. 287–297.
[6] L. Xu, K. Harfoush, and I. Rhee, "Binary increase congestion control (BIC) for fast long-distance networks," in *INFOCOM 2004. Twenty-third AnnualJoint Conference of the IEEE Computer and Communications Societies*, vol. 4. IEEE, 2004, pp. 2514–2524.
[7] S. Ha, I. Rhee, and L. Xu, "CUBIC: A new TCP-friendly high-speed TCP variant," *ACM SIGOPS Operating Systems Review*, vol. 42, no. 5, pp. 64–74, 2008.
[8] A. Baiocchi, A. Castellani, and F. Vacirca, "YeAH-TCP: Yet another high-speed TCP," in *Proc. PFLDnet*, vol. 7, 2007, pp. 37–42.
[9] H. Jung, S. Kim, H. Yeom, S. Kang, and L. Libman, "Adaptive delay-based congestion control for high bandwidth-delay product networks," in *INFOCOM, 2011 Proceedings IEEE*. IEEE, 2011, pp. 2885–2893.
[10] M. Scharf, M. Welzl, B. Briscoe, and D. Papadimitriou, "Open Research Issues in Internet Congestion Control," *Internet Research Task Force (IRTF), RFC 6077*, February 2011.
[11] A. Mudambi, X. Zheng, and M. Veeraraghavan, "A transport protocol for dedicated end-to-end circuits," in *Communications, 2006. ICC'06. IEEE International Conference on*, vol. 1. IEEE, 2006, pp. 18–23.
[12] *Closed-Loop Control System, National Instruments tutorial, http://www.ni.com/white-paper/3096/en.*
[13] L. Brakmo and L. Peterson, "TCP Vegas: End to end congestion avoidance on a global Internet," *Selected Areas in Communications, IEEE Journal on*, vol. 13, no. 8, pp. 1465–1480, 1995.
[14] K. Song, Q. Zhang, and M. Sridharan, "Compound TCP: A scalable and TCP-friendly congestion control for high-speed networks," *Proceedings of PFLDnet 2006*, 2006.
[15] M. A. Mani and R. Mbarek, "Performance Evaluation of High Speed Congestion Control Protocols," *IOSR Journal of Computer Engineering (IOSRJCE)*, vol. 3, no. 1, pp. 12–19, 2012.
[16] J. B. Abed, L. Sinda, M. A. Mani, and R. Mbarek, "Comparison of High Speed Congestion Control Protocols," *International Journal of Network Security & Its Applications*, vol. 4, no. 5, 2012.
[17] C. Callegari, S. Giordano, M. Pagano, and T. Pepe, "Behavior analysis of TCP Linux variants," *Computer Networks*, vol. 56, no. 1, pp. 462–476, 2012.
[18] M. Allman, V. Paxson, W. Stevens *et al.*, "RFC 2581 TCP congestion control," 1999.
[19] *GLORIAD http://www.gloriad.org/gloriaddrupal/.*
[20] *Iperf http://iperf.sourceforge.net/.*