Network Working Group
Internet Draft
Intended status: Experimental
Expires: December 28, 2016

H.Zheng
CEIE,BJTU
C.Qiao
SUNY
K.Chen
Y.Zhao
CEIE,BJTU

June 25, 2016

An Effective Approach to Preventing TCP Incast Throughput Collapse for Data Center Networks

draft-zheng-tcpincast-00.txt

Status of this Memo

Copyright Notice

Abstract

This document presents an effective solution to the known TCP incast problem in data center networks. The incast problem refers to a drastic TCP throughput drop when the number of servers synchronically sending data to the same receiver is too large. Our idea is avoiding packet losses before TCP incast happens. The scheme is limiting the number of concurrent senders such that the link can be filled as fully as possible but no packet losses. In this document we examine the condition that the link can be saturated but no packet losses. Then based on the condition we propose an approach to estimates the reasonable number of concurrent senders. Our approach does not modify TCP protocol itself and can thus be applied to any TCP variant, and works regardless of the type of data center network topology and throughput limitation. Analysis and simulation results show that our approach eliminates the incast problem and noticeably improves TCP throughput.

Table of Contents

1. Introduction

Data centers are becoming one of hottest topics in both research communities and IT industry. It is attractive to build data centers atop standard TCP/IP and Ethernet due to their technological advantages and economy of scale. Although TCP has been used widely in the Internet and works well, TCP does not work well in data center networks. A reported open problem is TCP incast [1-4].

TCP incast, which results in gross under-utilization of link capacity, occurs in synchronized many-to-one communication patterns. In such a communication pattern, a receiver issues data requests to multiple senders. The senders respond to the request and return an amount of data to the receiver. The data from all senders pass through a bottleneck link in a many-to-one pattern to the receiver. When the number of synchronized senders increases, throughput observed at the receiver drops to one or two orders of magnitude below the link capacity.

Several factors, including high bandwidth and low latency of a data center network, lead to TCP incast [3]. The barrier synchronized many-to-one communication pattern triggers multiple servers to transmit packets to a receiver concurrently, resulting in a potentially large amount of traffic simultaneously poured into the network. Due to limited buffer space at the switches, such traffic can overload the buffer, resulting in packet losses. TCP recovers from most of packet losses through timeout retransmission. The timeout duration is at least hundreds of milliseconds, which is orders of magnitude greater than a typical round trip time in a data center network. A server that suffers timeout is stalled even though other servers can use the available bandwidth to complete transmitting. Due to the synchronized communication, however, the receiver has to wait for the slowest server that suffers timeout. During such a waiting period, the bottleneck link may be fully idle. This results in under-utilization of the link and performance collapse.

This document addresses the above TCP throughput collapse problem. We propose an approach to preventing TCP throughput collapse. We focus on avoiding packet losses before TCP incast happens. The basic idea is to restrict the number of servers allowed to transmit data to the receiver at the same time to a reasonable value so as to avoid TCP incast.

For given limited amount of data required by the receiver, too few servers send data at a time may not fully utilize the bandwidth on the link. Too many servers, however due to the limited switch buffer, simultaneously sending to the receiver will lead to overfilling of the buffer, resulting in TCP timeouts and then TCP incast. Accordingly the number of servers that are permitted to send data at the same time should be carefully determined. Our objective is to find a reasonable value of number of co-existing servers to ensure that the aggregate bandwidth is utilized efficiently but without packet losses.

Our approach is to calculate the reasonable number of servers allowed to send data at the same time. In this document, we give the condition that the bottleneck link can be saturated but no packet losses (SBNPL). We show that if there is sufficient amount of traffic on the bottleneck link while the backlog created at the switch buffer never exceeds the buffer size, the bottleneck link can be utilized effectively without losses. Based on this condition, we then propose an approach to calculating the reasonable number of concurrent TCP senders. The simulation results and analysis show our approach noticeably improves throughput performance.The approach is an application-layer mechanism. It can be a global scheduling application or an application staggering requests to a limited number of senders. It also can be implemented on senders, which can coordinately skew their responses. The advantage of our approaches is that it does not need to modify TCP protocol or the switch.

The reminder of this document is organized as follows. In section 2, we describe model discussed in this document, including TCP incast model and TCP flow characteristics in data center. We examine the condition that the bottleneck link can be saturated but no packet losses in section 3. Simulation results for evaluating performance are presented in section 4. In section 6 we conclude this document and discuss future works.

2. Model

2.1. TCP Incast Model

TCP incast occurs in synchronized many-to-one communication patterns. Research works have shown that many applications in data centers use many-to-one communication pattern [3,5,6]. A simplified model is originally presented in [1]. It is a distributed storage system, a file is split into data blocks. Each data block is stripped across N servers, such that each server stores a chunk of that particular data block, denoted by Sender Request Unit (SRU). We call the number of servers across which the file is stripped as stripping width. The receiver requests servers for that particular block; each server responds and returns the chunk, i.e., SRU, stored on it. Only after the receiver has gotten all chunks of the current block, it requests for the next block. Essentially, this is a Bulk Synchronous Model (BSP) [7].

Built atop TCP/IP and Gigabit Ethernet, the underlying network is constructed with high bandwidth (1~10 Gbps, even higher) and low latency (RTT of 10~100 microseconds). The link connecting the receiver and the switch is the only bottleneck.

In our model, each sender only opens one TCP connection for transmission. TCP connections are kept open until the transfer of the whole file completes, or the receiver does not request any more. It is because that it is not necessary to open an individual connection for transferring each data block of the file.

The transfer of a whole file consists of 'rounds'. A round begins when the receiver requests for a data block, and ends when the receiver has gotten all chunks of that block. In any round, each TCP connection transfers limited amount of data, which equals to the size of SRU, according to TCP congestion control algorithm.

Consequently the transfer of a SRU in each round can be viewed as a succession of 'sub-rounds'. At the beginning of any sub-round, each TCP sender transmits SEND W packets back-to-back, where SEND W is the current size of TCP send window. These packets are so-called 'outstanding packets', which have been sent but whose ACK's have yet to be received by the sender. If no packet losses, each outstanding packet must either be in the buffer queue, or be in the delay pipe (it is in the pipe from the sender to the receiver or its associated ACK packet is in the pipe in the other direction). Once all packets falling within the current send window have been transmitted, no other packets are sent until the first ACK is received for one of these packets. The reception of this ACK signals the end of the current sub-round and the beginning of the next sub-round. So the duration of a sub-round is a RTT. At the beginning of the next sub-round, a group of NEW SEND W new packets will be sent, where NEW SEND W is the new size of TCP send window. When all packets associated to that particular SRU have been acknowledged, the transfer of that particular block chunk completes. The next round starts when the slowest TCP connection completes the transfer of its corresponding SRU. Note that when the new round begins, the value of TCP congestion window is that of congestion window when the last round ends.

2.2. TCP rabbits in data centers

TCP has unique characteristics in the context of data centers. More specifically, a TCP flow is generally referred to as either a TCP mouse if it is short (so short that it never exits the slow-start phase), or a TCP elephant if it has a bulk of data to transfer and lasts very long.

From the discussion above, we observe that the size of a TCP flow is between a mouse and an elephant. In particular, a TCP flow may enter both the slow-start phase and the congestion avoidance phase during data transmission. We call such a TCP flow a “rabbit”.

Note that a rabbit TCP is elusive: it may exit slow start and enter congestion avoidance in the first round during which the first data block is transmitted (in the case of narrow striping) or in later rounds during which subsequent data blocks are sent (in the case of wider striping). While one may use different equations to model a TCP rabbit’s behaviors in different rounds, or for different widths of striping, it greatly complicates the implementation. Moreover when there are multiple co-active rabbits, the interaction between them will make it much harder to model and predict their behaviors. Our objective is to find an approach that: is easy to implement, works regardless of TCP phase, TCP parameters configuration and scalability of data center and does not need to modify TCP protocol itself.

3. The Condition SBNPL

Denote the reasonable number of servers as senders to simultaneously transmit with m. In this section at first we will explain why limiting the number of concurrent senders can guarantee throughput of the receiver. Then we will deduce the condition that the link can be saturated but no packet losses.

3.1. Why limit the number of concurrent senders

Given stripping width, observe throughput of the receiver varying the number of concurrent TCP senders, i.e., the value of m. Our experiment (for a block of 1 MB stripped across 256 servers with buffer size of 64 KB) shows that throughput firstly increases then drops with the increasing of m. Because the volume of data sent by a TCP sender is just limited to 4 KB, when there are few concurrent TCP senders, the total amount of data transmitted by these senders is not sufficient to saturate the link, such that throughput of the receiver is low. With the increasing of the value of m, there are more concurrent TCP senders pouring data onto the link, and throughput increases till it reaches a top. When the value of m becomes larger, the potentially large amounts of traffic contributed by these m concurrent senders exhausts the buffering capacity of the link, which leads to packet losses; the throughput falls from the top instead of rising. We call the top throughput as the optimal throughput, and the value of m corresponding to the optimal throughput as the optimal number of concurrent senders.

If the number of concurrent TCP senders exactly equals to the optimal value, the receiver can obtain optimal throughput. The simulation result also shows that even if the number of concurrent senders is not optimal, there exists a value range of m where the receiver obtains good throughput performance. By “good” throughput performance, we mean the receiver achieves 98 percent or above of the optimal throughput. Hence we view those values of m within this range as reasonable number of concurrent senders. It may be hard to find the exact optimal value of m such that the receiver achieves optimal throughput, while find the reasonable value of m is more feasible. In order to find the reasonable value of m, we deduce the condition SBNPL in later subsection.

3.2. Assumptions and notations

Ignoring the time that it takes to read data from disks or caches, once each sender receives a request from the receiver, it responds and returns its corresponding chunk. Thus at the beginning of any sub-round, all senders simultaneously inject data to the network. Because TCP connections are synchronized, an assumption also adopted in [8], they share both the buffer and the bottleneck link bandwidth quite equitably. That is, each TCP connection shares the buffer and the link bandwidth with B/m and C/m, respectively, where B is the buffer size and C is the bandwidth of the bottleneck link.

Assume that both TCP send and receive socket buffer sizes are large enough that TCP send window SEND_W is limited by its congestion window CON_W, i.e., SEND_W = CON_W. All links have same bandwidth C, and propagating delay d. The switch has buffer capacity B, and manages queue with Drop Tail algorithm.

3.3. The condition SBNPL

It is enough to analyze the transfer in a round, since for any round the data block’s size is same and the block is transferred according to TCP congestion control algorithm. It is worth noting that at the beginning of a round the TCP send window size is the one when the last round ends.

Due to synchronization, the windows evolutions of all TCP connections are synchronized, at any sub-round the total amount of data poured onto the bottleneck,M, is merely the sum of amount of data contributed by each connection at the current time, that is SUM_M.

As mentioned above, if measured in units of packets, these M/P outstanding packets, where P is the size of a packet, must either be in the buffer queue, or be in the delay pipe, assuming no packet losses. If M is not less than the bandwidth delay product of the bottleneck, the bottleneck can be saturated. When these m TCP connections share the bottleneck, each connection's bandwidth delay product equals to its share of the link bandwidth times RTT time that it traverses the bottleneck. Because they are synchronized, each connection's share of the bottleneck is equitable, (i.e., which is C/m), and the RTTs are similar, so the total bandwidth delay product is the sum of m connections' bandwidth delay product, which equals to the bottleneck bandwidth times the average RTT, i.e.,C*RTT AVR [9]. Ignoring the transmission time on the link between senders (or the receiver) and the switch, and processing time on the switch and/or the senders and the receiver, RTT AVR is four times of the propagating delay of the bottleneck.

In literature[10] it showed that when there is only one TCP connection on the bottleneck link, the link can be saturated but no packet losses, at each time as long as the amount of data poured onto the link satisfies the condition C*RTT AVR=M=C*RTT AVR+B. A simple extending of this condition to a version where m concurrent synchronized TCP connections share the link is C*RTT AVR=SUM M=C*RTT AVR+B. However, for TCP incast problem, this condition does not hold because of highly bursty traffic. At the beginning of a round, for each TCP connection it is possible that its window has increased enough large that SEND W packets are transmitted in back-to-back manner, resulting in flash crowd at the buffer. Such a traffic burstiness can lead to a backlog created at the buffer even when M<C*RTT AVR, and packet losses can happen even when M<C*RTT_AVR+B.

4. Performance evaluation

4.1. Simulation Configuration

We use network simulator NS-2 [11]to run simulation experiments, and leverage the NS-2 source code provided in [1,3] for our performance evaluation. Inspired by distributed storage and bulk block transfers in parallel processing applications such as MapReduce, we use the workload that is described in Section 2. Especially the data block size is fixed; it was thought to be representative of communication patterns in popular distributed storage systems [3,4].

The performance metric is throughput over the bottleneck link, given by the total bytes received by the receiver divided by the finishing time of the last sender. We explore throughput by varying parameters, such as the number of servers, switch buffer size, data blocks size and bandwidth delay product. As the number of servers increases, each server would transmit a decreasing chunk block, for the size of SRU is 1/n MB, where n is the number of servers. We vary the value of n from 4 till the value where the size of SRU is 2 KB, which is of representative of minor chunk of data. Data block sizes are 1MB and 4MB, respectively, and BDP varies from 12.5 to 25. We run 100 times simulation experiments, each simulation run 20 seconds. The throughput is averaged over 100 experiments.

4.2. Throughput performance

We fix the size of data blocks 1MB. The simulation result shows that our proposed approach improves incast throughput notice.

4.3. The effect of the limitation to the buffer size on throughput

We redo simulation experiments varying the buffer size B , given network bandwidth delay product of 12 KB. We consider three cases: B>=C*RTT_AVR and B<C*RTT_AVR-S .

The simulation result shows that our proposed approach can make sure the link utilized as fully as possible without packet losses. It is because sufficient enough data are poured onto the link. As long as B is not less than the bandwidth delay product, the utilization of the link can be 50% above.

4.4. The effect of BDP on throughput

The relation of bandwidth delay product to TCP throughput is well-known in the literature, such as [12,13]. We do simulation experiments varying bandwidth delay product, given the buffer size of 64 KB. The calculation of m with the improved approach is independent on network bandwidth delay product. The simulation result shows when bandwidth delay product increases, RTT increases and throughput decreases due to TCP itself congestion control mechanism. Nonethe-less, our approach improves incast throughput noticeably; the utilization of the bottleneck link is 70% above even when bandwidth delay product is higher.

5. Related Work

The idea of restricting the number of co-active servers (i.e., servers that send data simultaneously to the receiver) is originally suggested in [14]. However the authors did not give any clue on how to determine the value of the number of co-active servers. In [15], the authors proposed an approach to calculating this value. However, the assumption was that TCP always works in the slow-start phase and never enters congestion avoidance. Our work use a more realistic assumption: TCP does enter both phases. It will be shown that it brings more challenges for the calculating in next section.

The idea of avoiding packet losses so as to prevent throughput collapse due to incast is also used in ICTCP proposed in [16]. The difference between our work and ICTCP is that our approach works without requiring any changes made to TCP protocols while ICTCP modifies TCP receiver to avoid incast congestion.The authors of [18]provided an admission control to TCP senders in order to limit the number of concurrent senders. However, their scheme requests the switch can report the network situation.

DCTCP in [5] tries to improve TCP throughput based on the idea of reducing the switch buffer occupation. It achieves the goal by using marking scheme at switches and modifying TCP congestion control algorithm to react the marking scheme. Work [3] lowered TCP’s retransmission timeout value to improve TCP throughput, however, most systems lack the fine-grained TCP timer required for such a low RTO. Work [1] studied whether the TCP variants, such as TCP SACK and TCP New Reno, and further improvements to TCP loss recovery, such as Limited Transmit, can prevent the incast problem.

Some algorithms, such as Quantized Congestion Notification (QCN) [20]and Fair Quantized Congestion Notification (FQCN) [21]are developed to provide congestion control at the switch or Ethernet layer in data center networks. It has been shown that QCN can effectively control link rates very rapidly in datacenter networks. However, it performs poorly when TCP incast occurs. FQCN, as an enhanced QCN, improve fairness of multiple flows sharing the link capacity.

Different from those existing works which either improve TCP protocol itself or are done at the Ethernet level, work [22]proposed a control protocol that is customized for the datacenter environment. The proposed uses explicit rate control to apportion bandwidth according to flow deadlines, motivated by the soft real-time nature of large scale web applications in today’s datacenters.

Manish Jain et al. reported the condition that a bulk TCP transfer can saturate a bottleneck link but no packet losses [23]. Although their work can be extended to a version for many TCP transfer on a bottleneck link, in our work barrier synchronized many-to-one communication pattern in incast brings about differences and challenges for deriving the condition SBNPL in three aspects. First, in our work each TCP connection just has limited amount of data to transmit. Yet, a “bulk” TCP transfer in existing works has infinite data to send. Second, in our incast model TCP enters both the slow-start and congestion avoidance phases, while existing works assumed TCP works either in congestion avoidance [21,24] or slow-start phase [21,15]. Because of the barrier synchronized transfer, the successive data blocks transmissions are related: when the transfer of a particular data block begins, TCP stays at which phase, slow-start or congestion avoidance, not only depends on the last data block transmission but also the number of senders. Third, our objective is to determine how many TCP transfers can simultaneously exist on the bottleneck link so as to utilize the link as fully as possible; the objective of [21] is to determine TCP socket buffer size so that the TCP transfer receives its maximum feasible throughput.

The effect of the buffer on throughput, and the relations of bandwidth delay product and TCP congestion control algorithm to sizing the buffer are well-known in the networking literature. References [25,26] described the rule-of-thumb and square-root rule for determining the size of the buffer, respectively. Both of them come from a desire to keep a congested link, that is, there are packet losses on the link, as busy as possible. Different from these existing works, in this document we discuss sizing the buffer assuming no packet losses on the link.

6. Conclusions

By restricting the number of concurrent TCP senders that simultaneously transmit on the bottleneck is an effective method to avoid packet losses, thus TCP timeouts. This method eliminates the root cause of incast throughput collapse. However, it needs carefulness to find the condition that the bottleneck can be saturated but no packet losses when there are many TCP senders co-active on the bottleneck. Considering traffic burstiness and the delay pipe buffering ability, we examine this condition, and calculate the reasonable number of concurrent senders. Our approach is simple and easy to implement, and improve throughput performance.

It is hard to find the exact optimal number of concurrent TCP senders such that the bottleneck link is fully utilized. In the future works, we will devote ourselves to analyzing the possibility of finding the optimal number of concurrent TCP senders, and to finding it if it exists. We also will extend our work for more complicated data center networks, where the path from receiver to servers traverses multiple Ethernet switches, or there are multiple receivers on a single switch or multiple switches sending requests to a shared subset of servers.

7. Security Considerations

This document makes no changes to the underlying security of TCP. No new security issues are raised within this document.

8. IANA Considerations

This document includes no request to INNA. Existing IANA registries for TCP parameters are sufficient.

9. References

[1] A. Phanishayee, E. Krevat, V. Vasudevan, D. G. Andersen, G. R. Ganger, G. A. Gibson and S. Seshan, “Measurement and analysis of TCP throughput collapse in cluster-based storage systems,” FAST ’08, Feb. 2008, San Jose, CA.

[2] D. Nagle, D. Serenyi and A. Matthews, “The panasas activescale storage cluster: delivering scalable high bandwidth storage”, Proc. the 2004 ACM/IEEE conference on Supercomputing, Washington DC, USA, 2004.

[3] V. Vasudevan, A. Phanishayee, H. Shah, E. Krevat, D. G. Andersen, G.R. Ganger, G. A. Gibson and B. Mueller, “Safe and effective fine-grained TCP retransmissions for data center communication”, SIGCOMM'09, August 17-21, 2009, Barcelona, Spain.

[4] Y.Chen, R.Grith, J.Liu, A.D.Joseph, and R.H. Katz. Understanding TCP incast throughput collapse in data center rnetworks. In Proc. Workshop: Research on Enterprise Networking, Barcelona,Spain,Aug.2009.

[5] M.Alizadeh, A.Greenberg, D.A.Maltz, J.Padhye, P.Patel, B.Prabhakar, S.Sengupta, andM.Sridharan. Data Center TCP (DCTCP). Proceedings of ACM SIGCOMM, 2010.

[6] S. Kandula, S. Sengupta, A. Greenberg, P. Patel, and R. Chaiken. The Nature of Datacenter Traffic: Measurements & Analysis. In ACM IMC 2009, Nov 4-6, 2009, Chicago, USA

[7] BSP Website. (2012). Available: http://www.bsp-worldwide.org

[8] Jiao Zhang, Fengyuan Ren and Chuang Lin, “Modeling and Understanding TCP Incast in Data Center Networks,” Proc. IEEE INFOCOM 2011, July, pp.1377-1385, doi: 10.1109/INFCOM.2011.5934923.

[9] S. Shenker, L. Zhang, D. Clark, “some observation on the dynamics of a congestion control algorithm,” ACM Computer Communications Review, vol. 20, 1990, pp. 30-39.

[10] Manish Jain , Ravi S. Prasad and Constantinos Dovrolis, “The TCP Bandwidth-Delay Product revisited: network buffering, cross traffic, and socket buffer auto-sizing,” CERCS Technical Reports, 2003.

[11] The network simulator 2, http://www.isi.edu/nsnam/ns.

[12] S. Shalunov and B. Teitelbaum, “ Bulk TCP Use and Performance on Internet2,” 2002. Also see: http://netflow.internet2.edu/weekly/.

[13] D. Katabi, M. Handley and C. Rohrs, “Congestion Control for High Bandwidth Delay Product Networks,” Proc. ACM SIGCOMM 2002.

[14] E. Keevat, V. Vasudevan, A. Phanishayee, D. G. Andersen, G.R. Ganger, G. A. Gibson and S. Seshan, “On application-level approaches to avoiding TCP throughput collapse in cluster-based storage systems”, Supercomputing ’07, Nov. 10-16, 2007, Reno, NV.

[15] Yongxiang Zhao, Changjia Chen, “A scheme to solve TCP transmission problem in data center”, 2010 Cross-Strait Conference on Information Science and Technology, CSCIST 2010, July 9-11, 2010, Qinghuang Dao, China.

[16] H. Wu, Z. Feng, C. Guo and Y. Zhang, ICTCP: Incast congestion control for tcp in data center networks. In ACM CoNext 2010, Nov 30 - Dec 3, 2010 Philadelphia, USA

[18] Adrian Tam, Kang Xi, Yang Xu, H. Jonathan Chao, “Preventing TCP Incast Throughput Collapse at the Initiation, Continuation, and Termination”, Proc. IEEE/ACM International Workshop on Quality of Service (IWQOS 2012), Coimbra, Portugal, June, 2012.

[20] M. Alizadeh, B. Atikoglu, A. Kabbani, A. Lakshmikantha, R. Pan, B. Prabhakar, and M. Seaman, “Data Center Transport Mechanisms: Congestion Control Theory and IEEE Standardization,” Proc. the 46th Annual Allerton Conference, Illinois, USA, Sep. 2008, pp. 1270–1277.

[21] Yan Zhang, and Nirwan Ansari, “On mitigating TCP Incast in Data Center Networks,” Proc. IEEE INFOCOM 2011, Shanghai, China, July. 2011, pp. 51-55.

[22] Christo Wilson, Hitesh Ballani, Thomas Karagiannis and Ant Rowtron, “Better Never than Late: Meeting Deadlines in

Datacenter Networks," Proc. ACM SIGCOMM 2011, New York, USA, 2011, pp. 50-61.

[23] Manish Jain , Ravi S. Prasad and Constantinos Dovrolis, "The TCP Bandwidth-Delay Product revisited: network buffering, cross traffic, and socket buffer auto-sizing," CERCS Technical Reports, 2003.

[24] Jiao Zhang, Fengyuan Ren and Chuang Lin, "Modeling and Understanding TCP Incast in Data Center Networks," Proc. IEEE INFOCOM 2011, July, pp.1377-1385, doi: 10.1109/INFCOM.2011.5934923.

[25] C. Villamizar and C. Song, "High performance tcp in ansnet," ACM Computer Communications Review, vol. 24, pp. 45-60, 1994.

[26] G. Appenzeller, I. Keslassy and N. McKeown, "Sizing router buffers," ACM SIGCOMM Computer Communication Review, vol. 34, Oct. 2004, doi:10.1145/1030194.1015499.

10. Acknowledgments

We thank the research group lead by Professor David Andersen (Carnegie Mellon University) for sharing the NS-2 simulation codes and in particular, Amar Phanishayee (Carnegie Mellon University) for help with understanding these codes. We also thank Professor Yongxiang Zhao and Professor Changjia Chen of Beijing Jiaotong University for feedback and support. Discussing with Professor Hongfang Yu (University of Electronic Science and Technology of China) and Bingli Guo (Beijing University of Posts and Telecommunications) also help our work.

This document was prepared using 2-Word-v2.0.template.dot.

Authors' Addresses

Hongyun Zheng
CEIE, BJTU
Beijing,China
Email: hyzheng@bjtu.edu.cn

Chunming Qiao
SUNY
Buffalo,NY,U.S.A
Email: qiao@computer.org

Kai Chen
CEIE, BJTU
Beijing,China
Email: 10120063@bjtu.edu.cn

Yongxiang Zhao
CEIE, BJTU
Beijing,China
Email: yxzhao@bjtu.edu.cn