On-the-fly Data Compression for Efficient TCP Transmission

Min Wang[1, 2], Junfeng Wang[1], Xuan Mou[1] and Sunyoung Han[3]
[1] College of Computer Science, Sichuan University
Chengdu 610064, P.R. China
[e-mail: honghewangmin@sina.com.cn, wangjf@scu.edu.cn, mou0718@163.com]
[2] College of Computer Science and Information Technology, Yunnan Normal University
Kunming 650092, P.R. China
[e-mail: honghewangmin@sina.com.cn]
[3] Department of Computer Science and Engineering, Konkuk University
Hwayang, Gwangjin, Seoul 143-701, Korea
[e-mail: syhan@cclab.konkuk.ac.kr]
*Corresponding author: Junfeng Wang

Received October 14, 2012; revised February 8, 2013; accepted March 16, 2013; published March 29, 2013

Abstract

Data compression at the transport layer could both reduce transmitted bytes over network links and increase the transmitted application data (TCP PDU) in one RTT at the same network conditions. Therefore, it is able to improve transmission efficiency on Internet, especially on the networks with limited bandwidth or long delay links. In this paper, we propose an on-the-fly TCP data compression scheme, i.e., the TCPComp, to enhance TCP performance. This scheme is primarily composed of the *compression decision mechanism* and the *compression ratio estimation algorithm*. When the application data arrives at the transport layer, the *compression decision mechanism* is applied to determine which data block could be compressed. The *compression ratio estimation algorithm* is employed to predict compression ratios of upcoming application data for determining the proper size of the next data block so as to maximize compression efficiency. Furthermore, the assessment criteria for TCP data compression scheme are systematically developed. Experimental results show that the scheme can effectively reduce transmitted TCP segments and bytes, leading to greater transmission efficiency compared with the standard TCP and other TCP compression schemes.

Keywords: TCP data compression; long delay networks; limited bandwidth; compression decision mechanism; compression ratio estimation

This work was supported by the Important National Science & Technology Specific Projects of China (2012ZX10004-901001); the National Natural Science Foundation of China (11102124), the Program for New Century Excellent Talents in University, Ministry of Education of China (NCET-10-0604), the Provincial Key Science and Technology Research and Development Program of Sichuan, China (2013SZ0002).
http://dx.doi.org/10.3837/tiis.2013.03.004

1. Introduction

With rapid growth in the amount of users and network applications, Internet traffic has been increasing explosively. A tremendous growth in the application of remote computing technology can be witnessed and there has been substantial increase in the amount of data transmitted over long delay networks. Therefore, it is highly desirable to provide high-speed data communication for a variety of network applications.

However, the networks with low and limited network bandwidth, such as residential and wireless networks, cannot keep up with the increasing bandwidth need of network applications. Serious network performance deterioration due to link congestion or packet loss might occur when the high-bandwidth-consuming applications run over a low bandwidth link.

The transmission rate of transport layer entities is controlled by the transport layer protocols. The Transmission Control Protocol (TCP) [1] is widely used as the transport layer protocol for reliable data delivery on the Internet. Many TCP congestion control approaches [2-7] have been proposed to enhance TCP performance in different network environments. TCP Westwood [2] significantly improves the data transfer efficiency in error-prone networks (e.g., wireless) by estimating the last "good" flow rate and using this rate as a baseline to distinguish between congestion packet loss and random packet loss. TCP Peach [3] improves TCP performance in the satellite networks by introducing the "dummy" segments to probe the bandwidth availability. Podlesny et al. [4] propose an Asymmetric Queuing (AQ) mechanism that enables full utilization of the bottleneck access link in residential networks with asymmetric capacities. Leu et al. [5] propose an aggressive path switching scheme for SCTP and evaluate the scheme in terms of end-to-end delay, jitters and throughputs. ARROW-TCP [6] is proposed to address the issues of stability and convergence in existing transmission control protocols. It uses explicit rate pre-assignment mechanism to obtain ideal performance of zero queuing delay and free packet loss. Liu et al. [7] propose a new scheme to enhance TCP performance in the space networks, and the simulation results show that the scheme greatly improves the throughput and time delay. Although these solutions greatly improve the bandwidth utilization by optimizing congestion control algorithms, their throughput enhancements have still been bounded in the networks with bandwidth constrained (e.g., residential or wireless networks) or long delay (e.g., satellite) links. This is because it is still apt to lead to link congestion or packet loss when a large amount of network traffic is transmitted over the networks with low bandwidth or long delay links. In addition, due to the fact that the throughput of TCP is inversely proportional to the round-trip time (RTT) of the networks, the congestion window growth rate would be reduced on the long RTT connections, thereby resulting in significant throughput degradation in the long delay networks.

Since data compression may increase the amount of application data carried by network links, it is promising to improve transmission efficiency in the networks with limited bandwidth or long delay links. Some industrial solutions have included compression technique, however, due to the proprietary nature of implemented technology there has been limited published material on the combination of transport protocols and compression systems [8]. Many researches have been done for data aggregation and compression of wireless and satellite network data [9-12], but the in-depth analysis and design are not given.

Data compression can be deployed on different layers of the network protocol stack. Our scheme concentrates on the transport layer compression, especially TCP data compression because of the following advantages:

1) The transport layer compression is transparent to users and applications, while the application layer scheme needs to modify user applications.

2) Compression at the transport layer could potentially gain higher performance than the lower network and link layer compressions because it can use larger data block to compress and reduce the number of transmitted TCP segments, thereby reducing the overhead of TCP headers and IP headers. In addition, it need not work on hardware thus may not be under restriction on the processing time of compression scheme [13].

The transport layer compression can bring a lot of benefits, while there are many challenges in the practical on-the-fly TCP data compression. It could perform ineffective compression on the hard-to-compress data such as audio and video data, which has been previously compressed by external processes, since it is unaware of the characteristics of the application data. It is preferred to use larger data block in the compression to achieve great compression benefits, while the compressed data size is limited by the Maximum Segment Size (MSS) of TCP connection. If the compressed data size is larger than the MSS, the compressed data would be encapsulated into multiple TCP segments in sending, and the receiver would wait and gather all of the segments to decompress together. This would increase the delivering latency between the application layer peers. Therefore, the challenging issues for the on-the-fly TCP data compression include: 1) when to carry out compression, 2) what compression scheme should be applied to maximize compression efficiency, 3) how to systematically analyze and evaluate the performance of compression scheme.

In this paper, we propose TCPComp, the on-the-fly TCP data compression scheme. When arriving at the transport layer, the application data is divided into data blocks and compressed, and then each compressed data block is encapsulated into a TCP segment. We believe that TCPComp will result in an immense TCP performance enhancement in the networks with limited bandwidth or long delay links. Throughout the paper, the term compression ratio refers to the ratio between the original data size and the compressed data size, and the term compression unit refers to the application data block to be compressed.

Our main contributions are as follows: Firstly, based on the statistical investigation of the correlation between some popular network data types and their segment compression ratios, the compression decision mechanism is introduced to determine which data block can be compressed. Secondly, for the upcoming data to be compressed, the compression ratio estimation algorithm is adopted to predict its compression ratio so as to determine proper compression unit size. Lastly, the assessment criteria for the transport layer compression scheme are systematically developed.

This paper is organized as follows. In Section 2, we review previous progress on the network data compression and briefly describe the Kalman Filter. Section 3 details the design of the TCP data compression scheme. Then experimental results are presented and discussed in Section 4. Finally, Section 5 concludes the paper by briefly summarizing the main points and proposing future work.

2. Related work

The candidate layers where compression can be carried out are the application layer, network layer, link layer and transport layer. In this section, we compare these four candidates. In addition, the Kalman Filter, which is used to predict the compression ratios of upcoming application data, is also briefly described in this section.

2.1 Data Compression at the Application Layer

Application layer is the nature place to deploy compression, i.e., data is compressed before given to the operating system or the TCP/IP stacks for transmission. Compression is proposed in File transfer protocol (FTP) [14] to effectively reduce the size of printer files such as those generated by Remote Job Entry (RJE) hosts. Jeannot et al. [15] present an Adaptive Online Compression library that enables data transmission with compression to improve middleware performance. Gutwin et al. [16] propose an automatic compression system that minimizes verbose messages sent by groupware. Experimental results show that these approaches have high transmission efficiency on wired networks. As the application knows the content, the most suitable compression algorithm can be chosen to achieve high transmission performance. However, it is not transparent for both users and applications, and it is impractical to modify all the user applications in a real world.

2.2 Data Compression at the Network Layer

Network layer compression is applied to TCP segments before adding IP headers. The most notable packet compression technique is IPComp [17] that has been proposed in RFC 3173. The protocol is designed to increase the overall communication performance between a pair of nodes over slow or congested links via compressing IP packet payload. Tan et al. [18] propose a real-time adaptive packet compression scheme for bandwidth limited high latency networks. This scheme adopts block compression to increase the compression ratios and reduce network load. DART [19] is proposed as a rate-based congestion control and scheduling mechanism with integrated data compression, in which a number of TCP segments may be compressed jointly to maximize the compression efficiency in the compression engine. Since TCP header is compressed at the network layer, some of information about the TCP connection such as port number cannot be recognized when a packet passes a layer-4 switch or firewall. Moreover, the network layer has to carry out extra compression when a TCP segment is retransmitted.

2.3 Data Compression at the Link Layer

CCP [20] is a compression technique used at the link layer that has been proposed in RFC 1962. The whole packet is compressed before transmitted into the network link and decompressed at the other end of the link. Chawla et al. [10] design and implement a data packet compression mechanism for IEEE 802.11 networks and study its impact on the compression efficiency and throughput. The link layer compression can speed up the delivery process and provide more efficient utilization of available capacity. But there must be a compression and a decompression on every physical link in the networks, since the compression is applied to IP header, which is used for routing packets from sender to receiver. Compressing IP header on every physical link has a lot of overheads on the network performance and it must work on hardware..

2.4 Data Compression at the Transport Layer

Compression could be carried out on user payload within the transport protocols, such as TCP, before adding TCP headers. A large number of experiments are conducted in [21] to assess the performance of Cisco devices that employ flow optimization and data compression methods. Experimental results show multi-fold improvements are achieved in the throughput over the buffer-tuned TCP both for single and most multiple streams. Lee et al. [22] suggest a kernel-level TCP data compression scheme, which is transparent to the existing applications and can provide high-speed wireless communication. This scheme initially explores some of tricky issues resulting from the kernel-level compression, while it has not represented detail solutions and deep analysis.

In addition to the above methods, some approaches, independent of specific protocol layers, are also proposed. Reinhardt et al. [11] present a Squeeze.KOM compression layer for sensor networks that encapsulates all functions within a separate compression layer. It can be combined with application level data encoding, energy-aware MAC protocols, data aggregation mechanisms, or header compression. The aim of this scheme is to preserve energy by reducing packet sizes and thus minimizing activity periods of the radio transceiver. However, these complex functions adopted in the compression layer, if not designed carefully, would not work well together.

2.5 The Kalman Filter

The Kalman Filter [23]bases on the criteria of lease-mean-square error estimation to search for a recursive estimation algorithm. It supports estimations of past, present, and even future states, and it can do so even when the precise nature of the modeled system is unknown.

The equations for the Kalman filter fall into two groups: time update equations and measurement update equations. The specific equations are presented below.

Time Update Equations:

$$\begin{aligned} \hat{x}_k^- &= A_{k-1}\hat{x}_{k-1} + Bu_{k-1} \\ P_k^- &= A_{k-1}P_{k-1}A_{k-1}^T + Q_{k-1} \end{aligned} \tag{1}$$

Measurement Update Equations:

$$\begin{aligned} K_k &= P_k^- H_k^T (H_k P_k^- H_k^T + R_k)^{-1} \\ \hat{x}_k &= \hat{x}_k^- + K_k (z_k - H_k \hat{x}_k^-) \\ P_k &= (I - K_k H_k) P_k^- \end{aligned} \tag{2}$$

where

$\hat{x}_k^-$ is a priori estimate at step k+1given knowledge of the process prior to step k,

P_k^- is a priori estimate error covariance at step k+1,

$\hat{x}_k$ is a posteriori state estimate at step k,

P_k is a posteriori estimate error covariance at step k,

K_k is Kalman Filter gain or blending factor at step k,

A_{k-1}, B and H_k are the state matrixes,

z_k is a measurement variable at step k,

u_{k-1} is a control input at step k-1,

Q_{k-1} and R_k are the process and measurement noise covariance matrixes (respectively).

3. On-the-fly TCP data Compression Scheme (TCPComp)

3.1 Overview

In this section, we briefly describe the overall design of the TCPComp scheme. **Fig. 1** shows the position where TCPComp is applied in the TCP/IP stacks and the architecture of TCPComp. A 2-byte TCP compression header is employed in each TCPComp segment as presented in **Fig. 2**. The compression header is used to indicate the transmission form of the application data or the compression algorithms used in the TCPComp (e.g. 0 for no compression, 1 for compression algorithm A, and 2 for compression algorithm B, etc.). We define the size of TCP MSS excluding the compression header as *TCPComp_MSS size*, i.e.,

$$TCPComp\_MSS\_size = MSS - lengthof(Hdr) \tag{3}$$

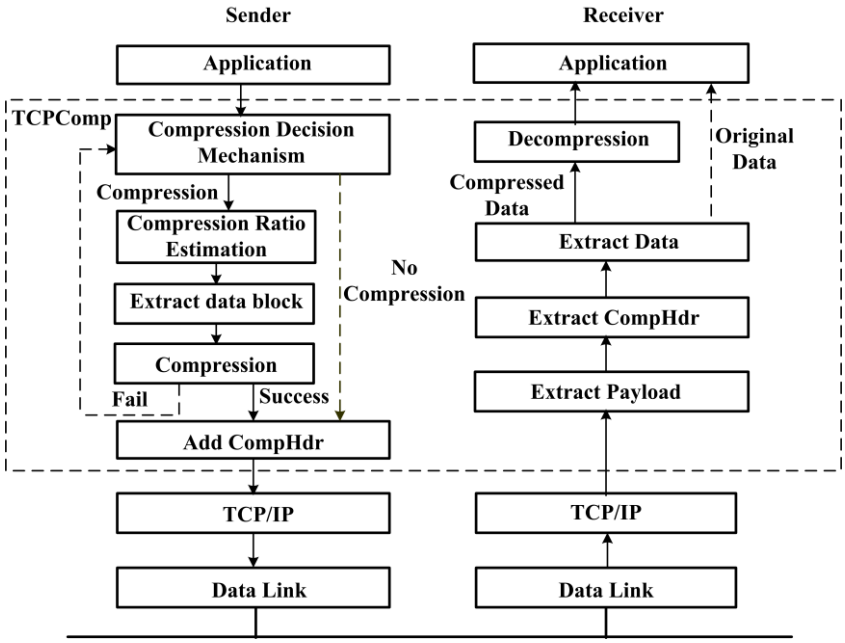

Fig. 1. The architecture of the TCPComp Scheme

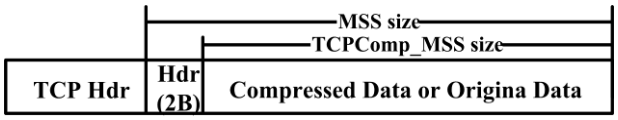

Fig. 2. The structure of the TCPComp segments

On the sender side, when application data arrives, it will be buffered within the socket layer. The compression decision mechanism is employed to determine whether to carry out the compression or not. If not, the data would be transmitted in the original form. Otherwise, a part of the application data is extracted from the buffer to be compressed and encapsulated in a TCP segment. The amount of the part of data, i.e., compression unit size, could be determined by many ways. One direct solution is to compress the application data in the TCPComp_MSS size, which avoids allocating the additional data buffer. However, the TCPComp_MSS size is smaller than the MSS size, and the MSS size is limited by the MTU size provided by network devices, which is around 1500 bytes in many cases. The compression ratios of the application data would be low when the compression unit size is too small. The TCPComp scheme determines the compression unit size according to the estimated compression ratio and obtains proper application data to compress. Meanwhile, it ensures that the compressed data size is no more than the TCPComp_MSS size so as to be encapsulated in a single TCP segment and then handed to the lower layer. For the application data of a given size, this method used to determine the compression unit size could not only achieve greater compression efficiency, but also reduce the amount of TCP segments and increase the amount of the application data transmitted in one RTT. If the compressed data size is more than the original data size or the TCPComp_MSS size, we think the compression has failed, and the data would be sent in the original form.

On the receiver side, the payload is extracted from a TCP segment, and then the compression header can be obtained. The total data is extracted and processed according to the compression header. If it is uncompressed, it would be directly delivered to the application layer. Otherwise, it would be decompressed according to the used compression algorithm and delivered to the upper-layer applications. In this study, we focus on the sender side and the basic idea can be easily extended to the receiver side.

3.2 Dynamical Compression Decision Mechanism

Since TCP is unaware of the characteristics of the application data, it would perform futile compressions on the data such as audio and video data, which probably has been encoded deliberately with less redundant information. In such cases, how to determine the data blocks that can be compressed is a challenge. If the application data is not compressible, the compression must stop in a timely manner so as not to affect transmission performance. For this, the compression decision mechanism is applied to determine when to carry out compressions. The details of the mechanism are described as below.

3.2.1 The Compression Ratios of Different Application Data Types

The type of application data transmitted on Internet usually includes text, audio and video, etc. We choose nine sets of data among which there are three sets of text data, three sets of audio data and three sets of video data. None of them has the same contents. The data is compressed using a constant segment length (equal to 1446 bytes) as compression unit, and then the first 10,000 compression ratios of each set of data are recorded. Confidence intervals with a confidence coefficient of 95% for the compression ratios of each set of data are

calculated. **Table 1** shows the confidence intervals of compression ratios. It is observed from **Table 1** that the compression ratios of video and audio data are lower than 1.2, while that of text data are higher than 1.5.

3.2.2 Compression Decision Mechanism

As mentioned before, futile compressions on the data, such as audio and video data, could be performed and thus great overheads would be brought to data transmission. In order to reduce the overheads brought by unnecessary compression attempts, the dynamic compression decision mechanism is applied in TCPComp to avoid futile compressions. The core of the mechanism is the backoff method. In most cases, the compression ratios of current compression units could be close to that of the adjacent compression unit due to the similarities of data from the same flow. If the compression ratios of the data in *n* consecutive compression units with the TCPComp_MSS size are lower than a threshold, named *CRmin*, it is possible that upcoming application data is also hard-to-compress. Therefore, the process of backoff starts, i.e., the next *m* segments are sent without attempting compression. If the compression ratio of the *(m+1)* th is still lower than *CRmin*, maybe there are more segments in upcoming data would be hard-to-compress. Hence, *m*, the backoff factor, is multiplied by 2 and the process of backoff continues. As long as a compression ratio is not lower than *CRmin*, the normal process of TCPComp restarts.

The value of *CRmin* is crucial for the compression benefits of the TCPComp scheme. If the value is too high, the chances that data is transmitted in the compressed form would be reduced, vice verse the compression benefits may be small due to the unnecessary compression attempts. An appropriate value should be chosen to obtain the greatest compression benefits but the least failure numbers. The value of *CRmin* in our scheme is suggested as 1.2 based on the results in **Table 1**.

Table 1. Confidence Interval of Compression Ratios

Data type	No.	Confidence Interval (Confidence Coefficient = 95%)
Text data	1	[1.5686, 1.5745]
	2	[1.6357, 1.6414]
	3	[2.4094, 2.4323]
Video data	1	[1.214, 1.2621]
	2	[1.0461, 1.0636]
	3	[0.9965, 1.0103]
Audio data	1	[1.0108, 1.0128]
	2	[0.9866, 1.0031]
	3	[1.003, 1.0376]

In the decision mechanism, once the compressed data size is larger than the original data size or the TCPComp_MSS size, the compression is thought to fail. Thus the data must be sent in the original form. As long as the compressed data size is smaller than the original data size, the data will be transmitted in the compressed form.

3.3 Compression Ratio Estimation Algorithm Based on Kalman Filter

The data compression ratios highly depend on the compression unit size. A larger compression unit generally results in a better compression ratio. An optimal compression unit size should

be utilized to achieve great compression efficiency. We estimate the compression ratios of upcoming application data based on Kalman Filter, and then the size of the *i*-th compression unit is determined as Equation (4)

$$orig\_size_i = target\_size * est\_CR_i,\ i = 1, 2, 3, \ldots \tag{4}$$

where est_CR_i is an estimated compression ratio of upcoming application data. The *target_size* is the expected size of the compressed data, and it should be no more than the TCPComp_MSS size according to the presentation depicted in the introduction. The optimal value of *target_size* in our scheme will be discussed in Section 4. In this section, we will explore the issue of the compression ratio estimation supposing the value of *target size* is fixed.

In most cases, the compression ratios of current compression units could be close to that of the nearest compression unit due to the similarities of data from the same flow. Therefore, a conservational estimation of compression ratios is expected to avoid compression failures induced by overestimated compression ratios. In Equations (1) and (2), we set the parameters A_k, B and H_k as unit matrixes, and u_{k-1}, Q_{k-1} and R_k as constants. Based on Kalman Filter, we derive the specific equations for estimating compression ratios as follows:

Time Update Equations:

$$\begin{aligned} est\_CR_k^- &= est\_CR_k + u_{k-1} \\ P_k^- &= P_{k-1} + Q_{k-1} \end{aligned} \tag{5}$$

Measurement Update Equations:

$$\begin{aligned} K_k &= P_k^- / (P_k^- + R_k) \\ est\_CR_k &= est\_CR_k^- + K_k (CR_{k-1} - est\_CR_k^-) \\ P_k &= (1 - K_k) * P_k^- \\ k &= 1, 2, 3, \ldots \end{aligned} \tag{6}$$

where

$est\_CR_k^-$ is a priori compression ratio estimate at step k,

est_CR_k is a compression ratio estimate at step k,

CR_k is a true compression ratio at step k,

The definitions of other parameters are same as that in Equations (1) and (2).

Once the data in the compression unit is transmitted in the compressed form, its true compression ratio is recorded and utilized to estimate the next one by Equations (5) and (6).

The new estimate will be adopted to calculate the next compression unit size by Equation (4). If the compression fails, the compression ratio estimation would be restarted.

4. Performance Evaluation

In this section, We follow a threefold experimental methodology: 1) three metrics are presented to evaluate the TCP compression scheme, 2)the optimal value of *target-size* is discussed according to the above presented metrics, 3) the performance of TCPComp is compared with the kernel-level compression scheme [22] and the standard TCP by the transmission time over real network environment.

4.1 Metrics Definition

Compression overheads differ from each other due to hardware and software related factors. Therefore a set of metrics for a TCP data compression scheme is required to provide a more timely understanding of data transmission in the view of compression benefits, and it is also possible to offer a chance to explore new approaches for more transmission performance. We present three metrics: *segment count*, *compression efficiency* and *compression failure number*, which can demonstrate the transmission performance of the TCP compression schemes systematically. The definitions of these metrics are as follows.

Definition1: If the compressed data size is no more than the TCPComp_MSS size, the TCPComp scheme would encapsulate the chunk of compressed data and the compression header with a TCP header, thereby forming a TCP segment. For certain application data of a given size, the total number of these segments is referred as *segment count*. The larger the *segment count* is, the greater the overheads of TCP and IP headers would be.

Definition2: For certain application data of a given size, when the TCPComp scheme is applied, the ratio between the amount of the application data and the sum of all transmitted TCP payloads excluding the compression headers is referred as *compression efficiency*. When TCPComp is adopted, the sum of all TCP payloads would be no more than the amount of uncompressed user data, i.e., *compression efficiency* should not be smaller than 1.0. The higher the *compression efficiency* is, the less the data transmitted over the links would be.

Definition3: The TCPComp scheme grabs the application data with a compression unit size to carry out a compression. If the compressed data size is more than the TCPComp_MSS size or the original data size, we think this compression process has failed. The third metric is the number of compression failures referred as *failure count*. For certain application data of a given size, the larger the *failure count* is, the greater the extra overheads of compression process would be.

4.2 Experimental System Setup

The TCPComp scheme is implemented in the Linux (kernel version 3.1.4). **Fig. 3** shows the experimental system setup. The three clients apply the standard TCP, kernel-level compression scheme and the TCPComp scheme, respectively, and all of them are equipped with Intel Pentium E5300 2.60GHZ processors and 2GB DDR2. The three servers are PCs equipped with Intel Xeon E7-8870 2.40GHZ processors and 1GB DDR3. The Internet access rates are 6 Mbps, and RTT between senders and receivers is about 100ms, which is the main characteristic of the high latency networks. Clients send data; the servers receive data and carry out corresponding decompression process.

Lossless compression schemes which can be applied to arbitrary data types with the ability to fully recover the contents by decompression. In our experiments, the Zip algorithm, an efficient lossless compression algorithm, is employed as the basic compression method. Other compression algorithms can be employed in a similar way.

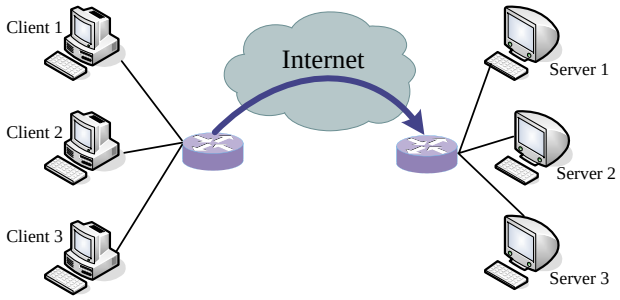

Fig. 3. Experimental system for TCPComp

4.3 Discussion of *target_size* Parameter

In order to optimize the performances of our scheme, we discuss the *target_size* based on the metrics presented in Section 4.1. As the compression ratios of text data are generally higher, it is possible for them to be sensitive to the value of *target_size*. Four text files are used in our experiments for determining the optimal value of *target_size*. The former three text files are obtained from the Canterbury Corpus [24], and the fourth is the Yahoo home page in the HTML format. The basic information about them is shown in **Table 2**. The MSS is 1448 bytes and TCPComp_MSS size is 1446 bytes in our experiment environment. **Fig. 4** shows the impact of the values of *target_size* on the performance of TCPComp as the *target_size* is enumerated from 100 bytes to 1600 bytes with step of 50 bytes.

Table 2. Basic information of the example files

Filename	File size (bytes)	Specification
bible.txt	4,047,392	The King James version of the bible
E.coli	4,638,690	Complete genome of the E.coli Bacterium
pi.doc	1,031,680	The first million digits of pi
Yahoo.htm	305,279	The Yahoo home page

As we can observe from **Fig. 4**, the segment count and failure number of all files decrease with the increases of *target_size* until it reaches 1400 bytes, meanwhile, the compression efficiency increases with the increases of *target_size*. Since the compression unit size would be increased with the increase of *target_size*, higher compression ratios could be achieved, i.e., higher performance in term of the three metrics. However, the situation changes when *target_size* is larger than 1400 bytes. Too large *target_size* would result in a consequence that compressed data could be larger than the TCPComp_MSS size due to the estimation errors, and thus this compression process fails. As a result, the segment count and failure number increase and the compression efficiency decreases when *target_size* is close to the TCPComp_MSS size. In this context, 1400 bytes is the optimal *target_size* in our experiment.

The values of the MSS could be distinct for different network conditions and so is the TCPcomp_MSS. Thus the value of target_size can be calculated by Equation (7). When

TCPComp_MSS size is 1446 bytes, 1400 bytes is the optimal *target_size* as shown in **Fig. 4**. Divide 1400 by 1446 and we get about 0.95 as the empirical value of "a" in Equation (7).

$$target\_size = a * TCPComp\_MSS,\ 0 < a < 1 \tag{7}$$

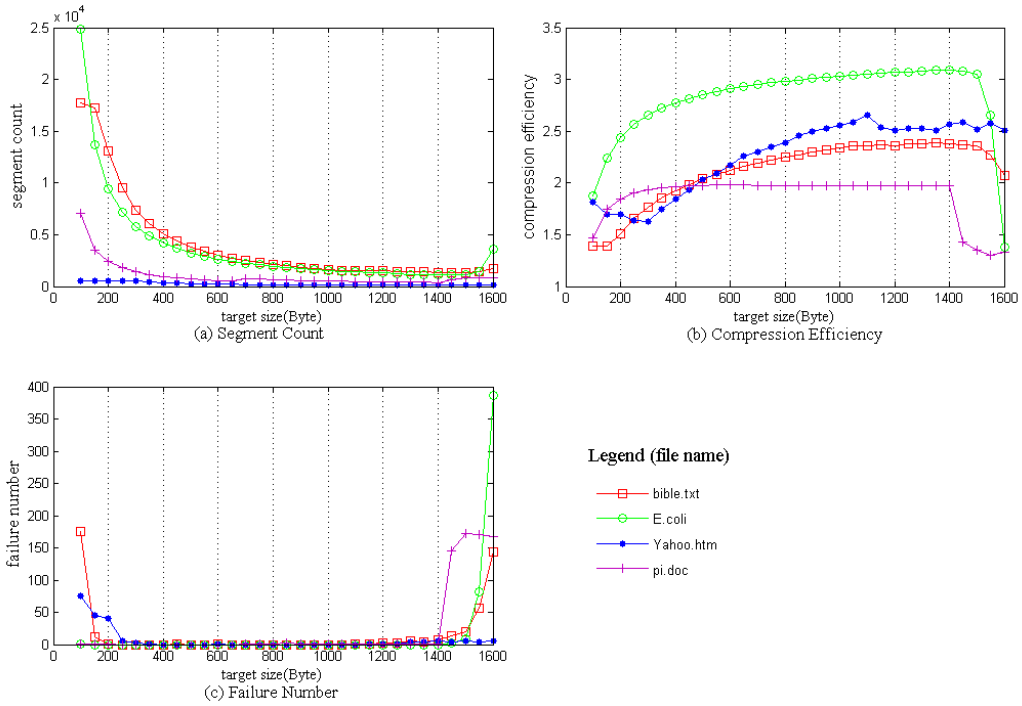

Fig. 4. Impact of the value of *target_size* on the performance of TCPComp

4.4 Experimental Evaluation of CR Estimation Algorithm

As mentioned in Section 3.3, TCPComp applies Kalman Filter to the compression ratio estimation. To evaluate the performance of the compression ratio estimation algorithm, all compression ratio estimates and the true compression ratios in the experiments are recorded and compared in **Fig. 5**. CR is referred as the compression ratio. The results show that the variation tendency of compression ratio estimates is overall in line with that of the true compression ratios, and the compression ratio estimates are basically near the true compression ratios. Enough application data can be obtained and compressed successfully due to the accuracy of the compression ratio estimation algorithm. Therefore, the compression ratio estimation based on Karlman Filter helps to increase compression efficiency and decrease compression failure number in the TCPComp scheme.

4.5 Performance Comparison with Other Schemes

For the evaluation of transmission performance, we employ three different kinds of data: text, multimedia, and hybrid data in the experiments. The text data comes from a few world classic novels in English; the multimedia data comes from a movie, The Godfather Part II, in the RMVB format; and the hybrid data is obtained from the webpage of Yahoo sports, including text, video, audio data and pictures. The clients deploy the standard TCP, the kernel-level

compression scheme and the TCPComp scheme, respectively. They send the same data to the servers at the same time. **Table 3** shows the parameters for TCPComp in our performance evaluation experiments.

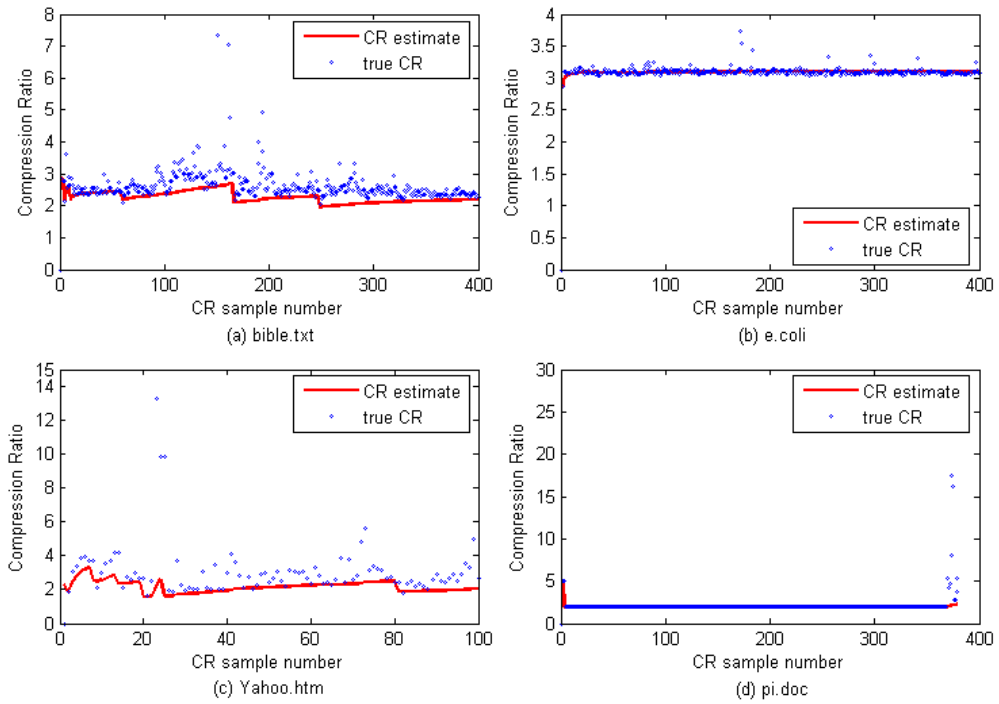

Fig. 5. Comparison between CR estimates and true CRs

Table 3. The parameter configuration for TCPComp

Component	Parameter	Parameter Value
Compression decision	*CRmin*	1.2
	n	5
	m	5
Compression ratio estimation	P_0	10
	u_k	10^{-3}
	Q_k	10^{-6}
	R_k	10^{-1}
	target_size	1400 bytes

We compare the performance of the three schemes in term of segment count, compression efficiency and transmission time. **Figs. 6-8** demonstrate that the performance of TCPComp on transmitting the text data. From **Fig. 6**, the average segment count in the TCPComp scheme is reduced by about 43.58% comparing with the standard TCP and about 43.74% comparing with the kernel-level compression scheme, which would result in the decrease of the overheads of TCP and IP headers. While the segment count in the kernel-level compression scheme is slightly more than that in the standard TCP. This is because the kernel-level compression scheme deploys the 4-byte compression headers, and the total size of the compression unit and the compression header is no more than the MSS size. Additional header overheads increase the amount of TCP segments. In **Fig. 7** the maximum CR denotes the

compression ratios of the whole experimental data compressed using the Zip algorithm at the application layer. Due to employing the compression ratio estimation algorithm based on Kalman Filter, the compression efficiency in the TCPComp scheme is twice as much as that in the standard TCP and a little higher than that in the kernel-level compression scheme. For the same application data, higher compression efficiency indicates that less data is transmitted over the links. This would help to alleviate network congestion and reduce the chances of packet drops and the amount of packet retransmissions at the same network conditions. Therefore, the transmission performance would be significantly enhanced. As shown in **Fig. 8**, the transmission time of text data in TCPComp is less than that in the kernel-level compression scheme and in the standard TCP, and it is worth noting that the transmission time in TCPComp is about half of that in the standard TCP.

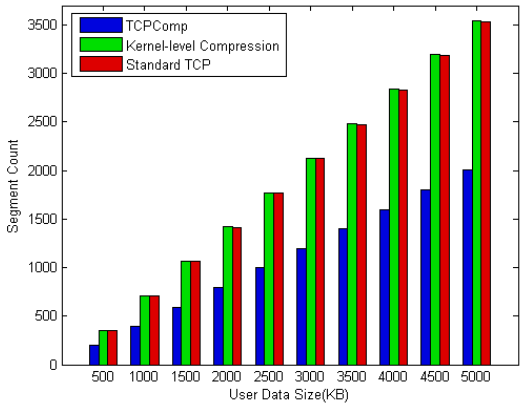

Fig. 6. The segment count of transmitted text data

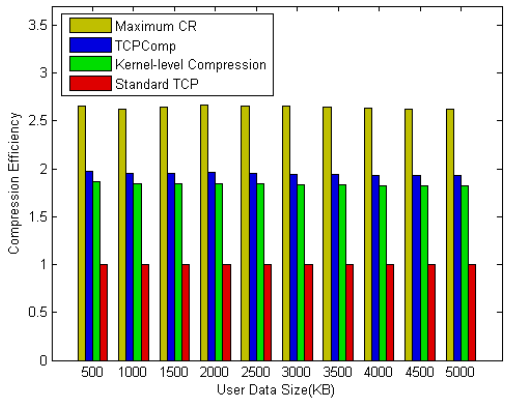

Fig. 7. The compression efficiency of transmitted text data

For hybrid data, which includes text and multimedia data, overall the performance of TCPComp is still better than that of two others. It is observed in **Fig. 9** that the maximum CR of hybrid data is significantly less than that of text data. This is because there are some non-compressible data such as videos and pictures in the hybrid data. The hybrids of application data with different types and context could result in a large number of estimation errors, thus the compression ratio estimation algorithm does not work as efficiently as in the case of text data. Nevertheless, the compression efficiency in TCPComp is higher than that in

two others because TCPComp can distinguish compressible data from the hybrid data and carry out compression by the compression decision mechanism proposed in Section 3.2. **Fig. 10** shows that the transmission time of hybrid data in TCPComp is still less than that in the kernel-level compression scheme and the standard TCP. Moreover, the segment count in TCPComp is comparable with that in two others, i.e., it does not increase the segment count.

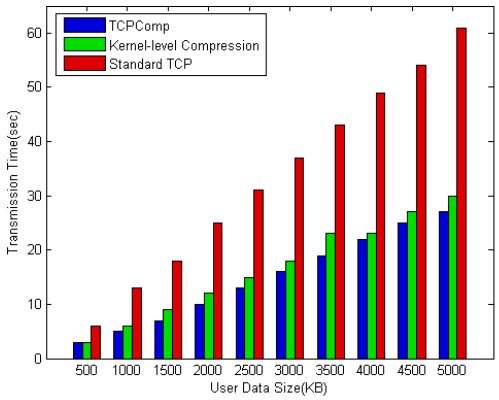

Fig. 8. The transmission time of text data

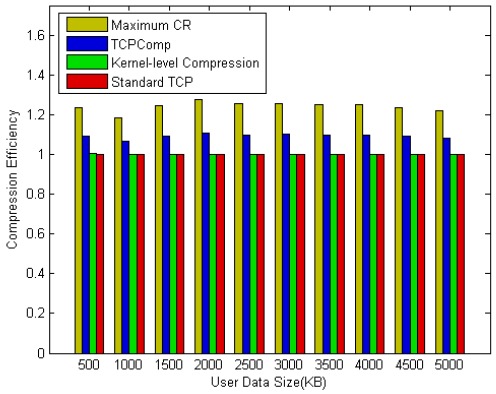

Fig. 9. The compression efficiency of transmitted hybrid data

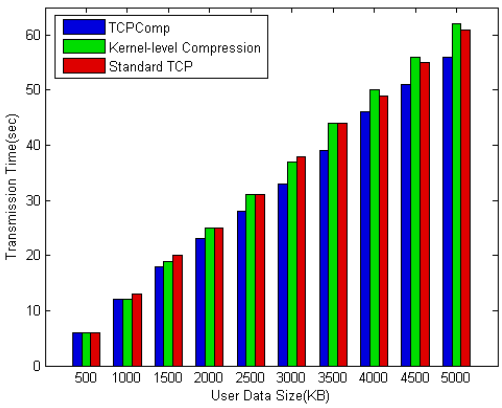

Fig. 10. The transmission time of hybrid data

For the multimedia data, which is usually hard to compress, the performance of TCPComp is comparable with two others as shown in **Fig. 11**. Extra process overheads are reduced because of the backoff method presented in Section 3.2.2. **Fig. 12** demonstrates that the backoff method significantly decreases the overheads for the multimedia data in term of the compression failure number.

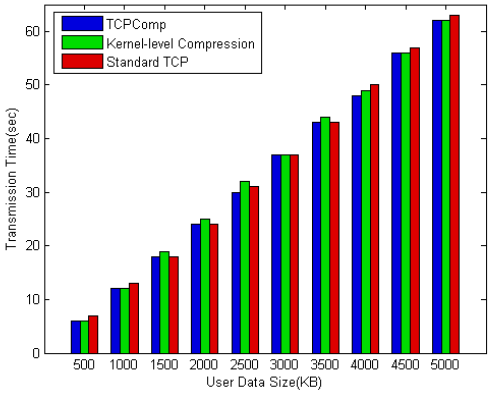

Fig. 11. The transmission time of multimedia data

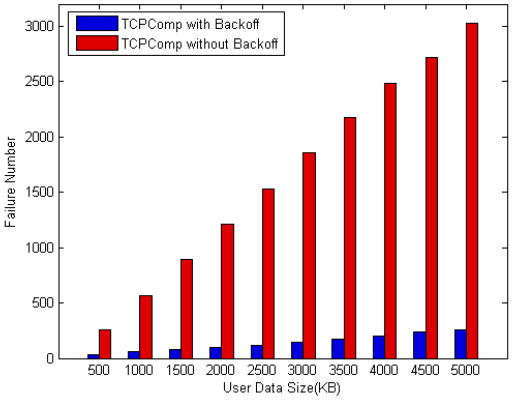

Fig. 12. The failure number comparison of TCPComp with and without backoff

5 Conclusions and Future Work

Data compression at the transport layer is potential to improve transmission efficiency in a low bandwidth or high delay network since it can reduce transmitted bytes over network links and increase transmitted application data in one RTT at the same network conditions. We propose a TCP data compression scheme (TCPComp) to enhance TCP performance. This scheme determines which compression unit can be compressed by the compression decision mechanism, and determines the compression unit size by the compression ratio estimation algorithm.

It is observed through experiments in the real network environment that though transmission time saving depends on statistics of the data and its content, overall TCPComp outperforms the kernel-level compression scheme and the standard TCP. When transmitting text data, huge performance gain can be brought due to the compression ratio estimation

algorithm. In the case of the multimedia data and the hybrid data, it achieves comparable performance with the two others due to efficient compression decision mechanism.

It is inevitable that compression would introduce some latency due to the intensive computation and memory access to compress and decompress data. However, since the backoff method in the compression decision mechanism could reduce extra process overheads, TCPComp can be applied in WANs, in particular the networks with long delay. Moreover, as TCPComp decreases the transmitted bytes over network links and thereby reducing the energy consumption of sensor nodes, it can also be applied in wireless sensor networks.

In our scheme, the Zip algorithm has been applied due to its high compression ratio. As future work, we intent to extend the TCPComp scheme based on its architecture by integrating a greater variety of compression algorithms.

References

[1] V. Jacobson, "Congestion Avoidance and Control," *ACM SIGCOMM Computer Communication Review*, vol. 18, no. 4, pp. 314–329, August, 1988. Article (CrossRef Link).

[2] S. Mascolo, C. Casetti, M. Gerla, M.Y. Sanadidi and R. Wang, "TCP Westwood: Bandwidth estimation for enhanced transport over wireless links," in *Proc. of the 7th annual international conference on Mobile computing and networking*, pp. 287–297, July 16-21, 2001. Article (CrossRef Link).

[3] I.F. Akyildiz, X. Zhang and J. Fang, "TCP-Peach+: enhancement of TCP-Peach for satellite IP networks," *IEEE Communications Letters*, vol. 6, no. 7, pp. 303–305, July, 2002. Article (CrossRef Link).

[4] M. Podlesny and C. Williamson, "Improving TCP performance in residential broadband networks: a simple and deployable approach," *ACM SIGCOMM Computer Communication Review*, vol. 42, no. 1, pp. 61–68, January, 2012. Article (CrossRef Link).

[5] F. Y. Leu, F. L. Jenq and F. C. Jiang, "A Path Switching Scheme for SCTP Based on Round Trip Delays," *Computers and Mathematics With Applications*, vol. 62, no. 9, pp. 3504–3523, November, 2011. Article (CrossRef Link).

[6] J. Wang, L. Rong, X. Zhang and J. Chen, "ARROW-TCP: Accelerating Transmission toward Efficiency and Fairness for High-speed Networks," in *Proc. of the IEEE Global Telecommunications Conference*, pp. 1–6, 30 Nov. - 4 Dec., 2009. Article (CrossRef Link).

[7] Y. Liu, C. He, X. Ge, Y. Dong and Z. Li, "A new scheme for improving the TCP transmission efficiency in space network," in *Proc. of the Second International Conference on Space Information Technology*, pp. 1–5, Nov. 10-11, 2007. Article (CrossRef Link).

[8] William B. Sebastian et al., *Methods and Systems for Performing TCP Throttle*, Patent No.: US 7911948B2, March 22, 2011.

[9] F. Marcelloni and M. Vecchio, "A simple algorithm for data compression in wireless sensor networks," *IEEE Communications Letters*, vol. 12, no. 6, pp. 411–413. June, 2008. Article (CrossRef Link).

[10] S. Chawla and B. S. Manoj, "Dynamic data compression in wireless networks," in *Proc. of IEEE 5th International Conference on Advanced Networks and Telecommunication Systems (ANTS)*, pp. 1–3, Dec. 18-21, 2011. Article (CrossRef Link).

[11] A. Reinhardt, M. Hollick and R. Steinmetz, "Stream-oriented lossless packet compression in wireless sensor networks," in *Proc. of the 6th Annual IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks (SECON'09)*,.pp. 1–9, June 22-26, 2009. Article (CrossRef Link).

[12] L. S. Tan, S. P. Lau and C. E. Tan, "Quality of Service Enhancement via compression technique for congested low bandwidth network," in *Proc. of IEEE 10th International Conference on Communications (MICC)*, pp. 71–76, October 2-5, 2011. Article (CrossRef Link).

[13] L. Wang and J. Manner, "Evaluation of data compression for energy-aware communication in

mobile networks," in *Proc. of International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC'09)*, pp. 69–76, October 10-11, 2009. Article (CrossRef Link).
[14] J. Postel and J. Reynolds, "FILE Transfer Protocol (FTP)," *IETF, Network Working Group, RFC959*, October, 1985. Article (CrossRef Link).
[15] E. Jeannot et al., "Improving Middleware Performance with AdOC: An Adaptive Online Compression Library for Data Transfer," in *Proc. of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05)*, pp. 70–79, April 3-8, 2005. Article (CrossRef Link).
[16] C. Gutwin, C. Fedak, M. Watson, J. Dyck and T. Bell, "Improving Network Efficiency in Real-Time Groupware with General Message Compression," in *Proc. of ACM 20th anniversary conference on Computer supported cooperative work (CSCW 2006)*, pp. 119–128, November 4, 2006. Article (CrossRef Link).
[17] A. Shacham et al., "IP Payload Compression Protocol," *IETF, Network Working Group, RFC 3173*, September 2001. Article (CrossRef Link).
[18] L. S. Tan, S. P. Lau and C. E. Tan, "Enhanced compression scheme for high latency networks to improve quality of service of real-time applications," in *Proc. of the 8th Asia-Pacific Symposium on Information and Telecommunication Technologies (APSITT)*, pp. 1–6, June 15-18, 2010. Article (CrossRef Link).
[19] T. Iyer, R. Boreli, G. Sarwar and C. Dwertmann, "DART: enhancing data acceleration with compression for satellite links," in *Proc. of IEEE Global Telecommunications Conference (GLOBECOM 2009)*, pp. 1–6, Nov. 30 -Dec. 4, 2009. Article (CrossRef Link).
[20] D. Rand, "The PPP Compression Control Protocol (CCP)," *IETF, Network Working Group, RFC 1962*, June 1996. Article (CrossRef Link).
[21] N. S. V. Rao, S. W. Poole, W. R. Wing and S. M. Carter, "Experimental analysis of flow optimization and data compression for TCP enhancement," in *Proc. of IEEE INFOCOM Workshops*, pp. 115–120, April 19-25, 2009. Article (CrossRef Link).
[22] M. Y. Lee, H. W. Jin, I. Kim and T. Kim, "Improving TCP Goodput over Wireless Networks Using Kernel-Level Data Compression," in *Proc. of the 18th International Conference on Computer Communications and Networks (ICCCN 2009)*, pp. 1–6, August 3-6, 2009. Article (CrossRef Link).
[23] R. E. Kalman, "A New Approach to Linear Filtering and Prediction Problems," *Journal of Basic Engineering*, vol. 82, no. 1, pp. 35–45, March, 1960. Article (CrossRef Link).
[24] The Canterbury Corpus file for testing new compression algorithms. Available at http://corpus.canterbury.ac.nz/index.html.

Min Wang received the M.S. degree in Computational Mathematics from Yunnan University in 2004. She is currently a Ph.D. student at the College of Computer Science, Sichuan University. Her research interests cover a wide variety of topics in wireless and satellite networks, with emphasis on design of transport layer protocols for wireless networks.

Junfeng Wang received the M.S. degree in Computer Application Technology from Chongqing University of Posts and Telecommunications, Chongqing in 2001 and Ph.D. degree in Computer Science from University of Electronic Science and Technology of China, Chengdu in 2004. From July 2004 to August 2006, he held a postdoctoral position in Institute of Software, Chinese Academy of Sciences. From August 2006, Dr Wang is with the College of Computer Science, Sichuan University as a professor. His recent research interests include spatial information networks, network and information security, and intelligent transportation system.

Xuan Mou received the B.S. degree in Information and Computing Sciences from Hubei University for Nationalities of China, Enshi in 2011. He is currently a M.S. student at the College of Computer Science, Sichuan University. His research interests include wireless and satellite networks, with emphasis on design of transport layer protocols for wireless networks.

Sunyoung Han received the B.S. degree in computer science from Seoul National University, and M.S. and Ph.D. degree in computer science from KAIST, Seoul, Korea, in 1977, 1979 and 1988, respectively. Since 1981, he is a professor of the Department of Computer Science and Engineering, Konkuk University, Seoul, Korea. He was a Dean of College of Information & Telecommunication, Konkuk University from Sept. 2004 to Feb. 2009. His research interests include overlay networks, future Internet, mobile Internet, IP Multicasting, vehicular networks and real-time distributed communication systems.