Hop-by-Hop TCP for Sensor Networks

Yao-Nan Lien

Computer Science Department, National Chengchi University, Taipei, Taiwan, R.O.C.
lien@cs.nccu.edu.tw

Abstract

Communication links in a sensor network are unstable such that running conventional TCP protocol over a high loss rate sensor networks will suffer from severe performance degradation. To handle a packet loss, conventional TCP retransmits the lost packet from its source. However, when error rate is high, it may have difficulty to deliver a packet to its destination. Considering that most applications on a sensor network prefer faster and reliable packet delivery to higher throughput, this paper proposes to use the Hop-by-Hop TCP protocol for sensor networks aiming to accelerate reliable packet delivery. Hop-by-Hop TCP makes every intermediate node in the transmission path execute a light-weight local TCP to guarantee the transmission of each packet on each link. It takes less time in average to deliver a packet in an error-prone environment. The performance of our approach is evaluated by simulation using NS-2 simulator. Our experiments show that Hop-by-Hop TCP outperforms TCP NewReno in both throughput and average packet delivery time. The fairness requirement is also achieved while Hop-by-Hop TCP coexists with other major TCP variants.

Keywords:

Sensor Network, TCP

1. Introduction

A wireless sensor network (WSN) is a wireless network consisting of spatially distributed autonomous devices using sensors to cooperatively monitor physical or environmental conditions, such as temperature, sound, vibration, pressure, motion or pollutants, at different locations [12].

Each node in a sensor network is typically equipped with a radio transceiver or other wireless communications device, a small microcontroller, and an energy source, usually a battery. Size and cost constraints on sensor nodes result in corresponding constraints on resources such as energy, memory, computational speed and communication bandwidth.

A sensor network normally constitutes a wireless multi-hop ad-hoc network. Without loss of generality, we assume that the data to be transmitted by a sensor node is in the form of packet; node ID w.r.t. the network is IP address; and the network is equipped with an appropriate routing mechanism that can adapt to the network dynamic. In other words, from the viewpoint of a transport protocol, the underneath network is an IP based full functional network. To assure a data packet to be delivered to the destination reliably, a transport layer protocol must be embedded between application and network layer.

The most popular reliable transport layer protocol is TCP. Unfortunately, it may perform poorly on an error-prone sensor network. In such an environment, communication links are unstable due to various reasons such as interference of radio signal, radio channel contention, and survival rate of nodes. Furthermore, the multi-hop feature increases the channel contention significantly that in turn increases error rate. In summary, in a sensor network, error rate is much higher and bandwidth is smaller than those of fixed networks. As a consequence, running conventional TCP protocol on a sensor network will suffer from severe performance degradation [4]. To handle a packet loss, conventional TCP retransmits the lost packet from its source. However, when error rate is high, it may have to take several retransmissions to deliver a packet to its destination successfully. Furthermore, packet losses may also activate TCP's congestion control causing too many what is so called "slow start" that will further impair the performance of packet delivery. As a result, the effective throughput is much lower and the average packet delivery time will be much longer. In the worst cases, a TCP may even completely stall when error rate is too high.

In fact, most applications on a sensor network prefer faster and reliable packet delivery to higher throughput. However, most versions of TCP are all designed to achieve higher throughput, not faster packet delivery. It may even completely stall in a highly error-prone environment. Therefore, it is beneficial to redesign a TCP by trading throughput for faster and reliable packet delivery.

Redesigning TCP may not be easy to implement on a WAN (Wide Area Network) because upgrading a large number of routers in a WAN is almost a business impossible. However, a sensor network has no such concern so that it is easy for a sensor network to embrace any new approach. This paper studies the Hop-by-Hop TCP protocol over sensor networks. Hop-by-Hop TCP was proposed by Lien [9] for Mobile Ad-Hoc Networks (MANET) aiming to accelerate reliable packet delivery. Hop-by-Hop TCP makes every intermediate node in the transmission path execute a local TCP to guarantee the transmission of each packet on each link. The retransmission of a lost packet is right at the sender end of the link where the packet is lost. It doesn't have to retransmit a lost packet all the way from its source node. It takes less time in average to transmit a packet to its destination in a high error rate environment.

One may argue that sensor network may not have sufficient computing power to implement the entire TCP/IP protocol. However, judging from the advancing speed of digital technology, we can anticipate that the computing power of sensor nodes will eventually be capable to support TCP/IP protocol.

The rest of this paper is organized as follow. In Section 2, we review the relative background and research regarding to TCP over a MANET, whose environment is similar to many sensor networks. We introduce our Hop-by-Hop TCP in Section 3 and evaluate it against others by simulation in Section 4. Finally, we conclude our main contribution of this paper and highlight some future work in Section 5.

2. Related Work

TCP is a connection-oriented reliable data delivery protocol. It provides a reliable transport service between pairs of hosts using the network layer service provided by the IP protocol. Whereas the IP protocol deals only with packets, TCP enables two hosts to establish a connection and exchange streams of data. TCP guarantees the delivery of data and also guarantees that packets will be delivered in the same order in which they were sent.

Packets may be lost in the transmission path due to various reasons such as network congestion and radio channel error. TCP adopts a complicated protocol to guarantee the delivery of packets. TCP modules reside at the both ends of a connection which may have quite a few intermediate nodes in between. Thus, the source node of a TCP connection must determine appropriate data rates based on its own poor knowledge of network status. Because most hosts on the network do not have a good knowledge of the network status, it is impossible for them to have a perfect data rate control, network will be congested from time to time. Network elements including routers and end terminals must work hard to avoid network congestion. The congestion control within a TCP plays a critical role in adjusting data rate to reduce network congestion. Based on some window-adjustment algorithm, a TCP not only guarantees the successful packet delivery, but also maintains an "appropriate" data rate [11].

The two indicators of network status are packet traveling time and the success/failure of package delivery. Therefore, most TCP variants count on these indicators to "guess" (estimate) the available bandwidth over the packet delivery path and to adjust its data rate accordingly. The accuracy and the promptness of bandwidth estimation are dependent on many factors such as traffic stability and path length. Not surprisingly, most TCP variants are suffering from some performance shortcomings, unnecessary network congestion and “slow start”. Unfortunately, a sensor network is generally slow and very unstable such that running TCP over sensor networks will suffer from even severe performance problems [4]. In summary, all TCP variants that are designed for conventional networks may not be appropriate for sensor networks. There is a need to redesign TCP protocol for sensor networks to improve its performance.

TCP-ELFN [5], TCP-F [1] and ATCP [10] are proposed to overcome the drawbacks of conventional TCP over MANET. They are mainly designed to improve throughput by dealing with the random packet losses caused by unreliable wireless links. They are based on the concept of freezing TCP states and keeping large congestion window without decreasing the transmission rate at the occurrence of routing change.

TCP Muzha [8] uses the assistance provided by routers to achieve better congestion control. To use TCP Muzha, routers are required to provide some information allowing

the source node to estimate more accurately the remaining capacity over the bottleneck link on the packet delivery path. With this information, TCP Muzha is able to enhance the performance of both TCP and network. TCP Muzha shows that having intermediate nodes provide assistance to the control of TCP is beneficial and practically feasible. It leads to the development of Hop-by-Hop TCP, which will be detailed in Section 3.

Split-TCP [7] is very similar to our Hop-by-Hop TCP. It divides a long path into several shorter segments and running a separated TCP in each segment. This approach can enhance the TCP performance. One drawback of Split TCP is that it is difficult to handle path change. In a Split-TCP session, all sub-TCPs running on path segments must coordinate with the upper level TCP closely. If any segment fails, the entire setup has to be changed accordingly. Unfortunately, nodes in a sensor network may be too weak to handle such a complicated reconfiguration task caused by frequent path failures.

3. Hop-by-Hop TCP

The design objectives of Hop-by-Hop TCP for sensor networks are as follows:

1. minimizing end-to-end packet delivery time without too much throughput degradation;
2. minimizing the number of retransmissions;
3. minimizing the occurrence of network congestion;
4. providing fairness service.

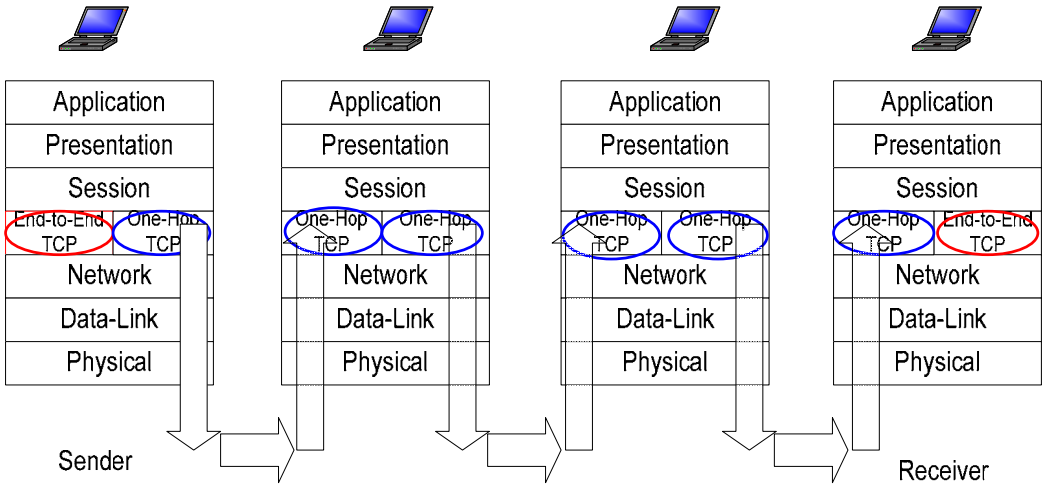


Fig. 1 Protocol Stack of Hop-by-Hop TCP

Hop-by-Hop TCP consists of two parts: an *End-to-End TCP* working on the source and destination nodes, and a *One-Hop TCP* working on every node as shown in Figure 1. The sender module of a One-Hop TCP is working at the sender end of a link, and the receiver module is working at the receiver end. Each link needs only one pair of One-Hop TCP for all End-to-End TCP sessions. However, for simplicity, we assume there is a One-Hop TCP for each End-to-End session in the following description.

3.1 End-to-end TCP

To avoid reinventing the wheel, we reuse an existing popular TCP protocol, NewReno, for the End-to-End TCP with several modifications. In fact, it can be replaced with any other version easily.

1. Instead of interacting with IP Layer, the sender module forwards packets to the One-Hop TCP module, and the receiver module receives packets from the One-Hop TCP module. One-Hop TCP in each node forwards data packets hop by hop to the destination node. Similarly, ACK packets for End-to-End TCP, called *End-to-End ACKs,* are forwarded to the source node using One-Hop TCP in the opposite direction.
2. Set a maximum threshold on the size of CWND to prevent it from over growth. The data rate of most sensor networks is extremely low and throughput is usually not a concern. Thus, there is no need to use a large CWND.
3. Set a larger initial RTO (Retransmission Time Out) value.

3.2 One-Hop TCP

One-Hop TCP is a light-weight version of TCP running on each node to forward received packets to the next node packet by packet reliably. Major modifications are as follows:

1. add the IP address (or alternative sensor ID) of current node to the packet header such that the receiver knows where to send Local ACK;
2. set the local RTO based on link characteristics;
3. set CWND to 1;
4. remove all CWND adjustment mechanisms;
5. set the upper threshold for the number of retransmissions.

Since the resource of a sensor node is usually very limited and the speed of One-Hop TCP is very critical to the performance of the entire TCP, One-Hop TCP must be very efficient and light weighted. Many TCP features, such as packetization and congestion control, are removed from One-Hop TCP. It is more like an enhancement to the link layer protocol to accommodate link failures. Furthermore, as mentioned earlier, only one session of One-Hop TCP is sufficient to manage all End-to-End TCP sessions that pass through the same link. Much overhead can be saved.

CWND is set to 1 for two reasons. First, the main reason that the sliding window mechanism is used in conventional TCPs is to allow more than one packet pending in the intermediate routers. However, there is no such need for a one-hop link. Secondly, if a succeeding packet were allowed to transmit before the ACK of the previously transmitted packet is received, the transmissions of the two packets will have to compete to each other for the radio channel. Thus, CWND is set to 1. In other words, after transmitting a packet, the sender module has to wait for the ACK packet before it can transmit next packet. In this way, complicated CWND size adjusting mechanisms become useless in One-op TCP and are all removed.

A packet is assumed lost if the corresponding ACK is not received before the local RTO expired. Once the link to the next node fails and the network layer performs a reroute, One-Hop TCP retransmits the packet transparently. Once the upper threshold of the number of retransmissions is reached, the retransmission stops. After the source node is aware of a packet loss (RTO expires or receiving of three duplicate ACKs), all corresponding actions will be taken.

3.3. Reduction of Control Messages

All control messages incurred by One-Hop TCP are extra overhead and must be minimized. Four different packet streams are flowing between source and destination nodes: (1) data packets, (2) local ACKs for data packets, (3) End-to-End ACKs, (4) local ACK for End-to-End ACKs.

The first and last types of messages are in the same direction. The other two types are in the opposite direction. Multiple messages in the same direction can be piggybacked together to reduce the number of messages. In a sensor network, each message may have to compete for radio channel such that the reduction of messages is very beneficial. To further reduce the number of messages, the packets of different End-to-End TCP sessions in the same link can all be piggybacked together as long as they are in the same direction. The piggybacking mechanism is shown in Figure 2.

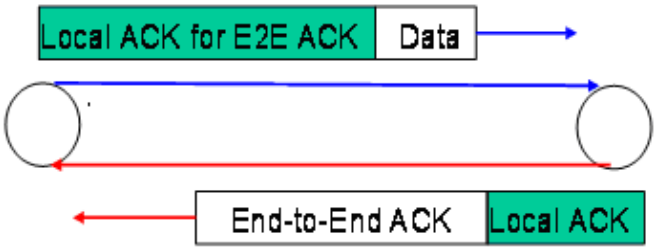


Fig. 2 Piggybacking Mechanism

The State Transition T able of One-Hop TCP at the sender end of a link is shown in Table 1.

3.4 Reduction of Duplicated Packets

The sender end of a TCP will transmit duplicated packets if they "thought" a packet is lost. However, the judgment of packet loss may not always accurate since a packet that were thought lost may be actually delayed for various reasons. Thus, the receiver end may receive duplicated packets. The receiver end of an End-to-End TCP is able to distinguish duplicated packets and to discard them. However, for efficiency reason, the receiver end of a One-Hop TCP doesn't memorize the sequence numbers of all received and forwarded packets such that it doesn't have the capability to distinguish duplicated packets. Although duplicated packets will eventually be discarded by the receiver end of the End-to-End TCP, they waste precious radio resources. Thus, we propose to

implement a short term memory to memorize necessary information of received packets and use it to distinguish duplicated packets.

Table 1 State Trans. Table of One-Hop TCP at Sender End

Event	Current State	Actions Taken	Next State
Receive a packet when the flag no_packet_outgoing is false	Ready	buffer the new packet; if there is any packet to be sent in another direction, piggyback it; if not, return LACK	Ready
Receive a packet and no packet is outgoing	Ready	buffer the new packet; if there is any packet to be sent in another direction, piggyback it; if not, return LACK, send the packet, and start Local RTO timer	Wait
Local RTO timer expires and retry count < 6	Wait	retransmit the lost packet; restart Local RTO timer	Wait
Local RTO timer expires and retry count > 5	Wait	purge the packet that retransmits over five times from buffer; stop Local RTO timer	Done
Receive a LACK from downstream node	Wait	purge the packet from buffer; stop Local RTO timer	Done
Other Packet in buffer	Done	send next packet from buffer; start Local RTO timer	Wait
Buffer empty	Done	set the flag no_packet_outgoing to true; wait the next packet	Ready

3.5 Network Fault Recovery

A sensor network is very unstable that a sensor node may be up and down according to various reasons such as power control. A transport protocol must be able to cope with this problem. Although a MAC layer protocol may be built-in with a reliable packet forwarding mechanism by local retransmission at each link, it will not be able to cope with link breakage. A reliable MAC protocol can coexist with Hop-by-Hop TCP. Assuming a packet is lost on a link due to a transient error, the Hop-by-Hop TCP will see a “good” link if the MAC layer protocol can recover from the error by retransmitting the lost packet. If the link can’t be recovered, e.g. next node is dead, the Hop-by-Hop TCP won’t receive the anticipated Local ACK in time. It will then retransmit the packet again and again until network routing protocol finishes its rerouting and updates its routing table. Nevertheless, Hop-by-Hop TCP may fail to retransmit the lost packet in extreme cases such as severe network breakage. In this case, the sender has to retransmit the packet after the corresponding End-to-End ACK misses its deadline.

4. Performance Evaluation

Hop-by-Hop TCP over sensor networks is evaluated using NS-2 network simulator under various conditions such as network topology, link reliability, network size, and

link bandwidth. Evaluation metrics in performance test are average packet delivery time, average throughput, average number of retransmissions, and the stability of CWND. Fairness test is executed in two different ways: coexistence with different version of TCP and coexistence with the same TCP of different initiation time. Hop-by-Hop TCP, TCP NewReno, TCP Vegas and TCP SACK are compared in the evaluation. Fig. 3 is the topology used in the performance test. Only one TCP session is injected to the network. The parameters used in the performance test are shown in Table 2.

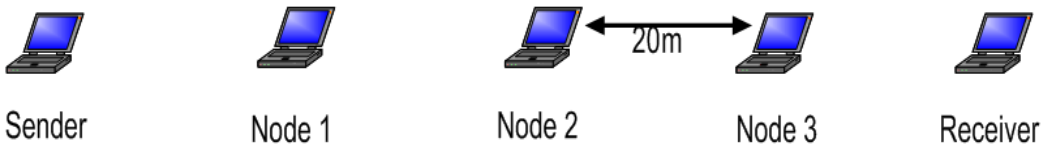


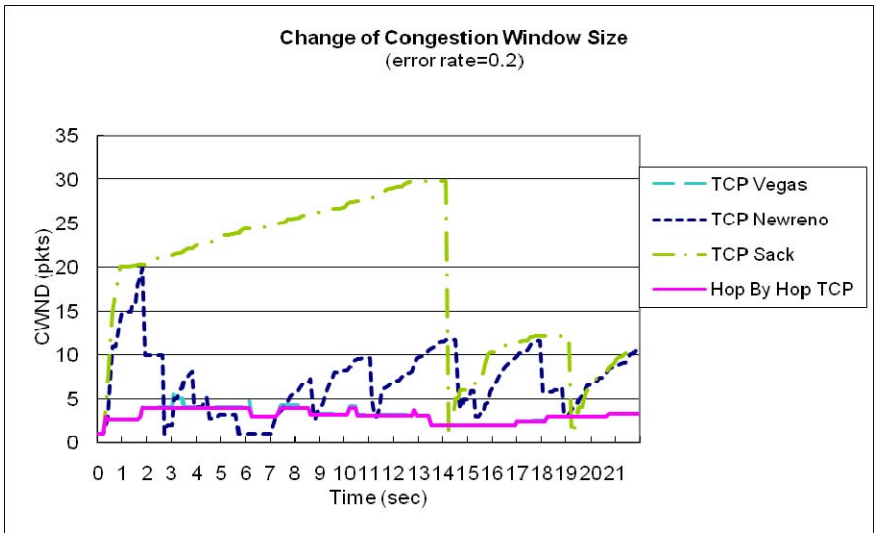
Fig. 3 Topology of Performance Test

Table 2 Parameters in Performance Test

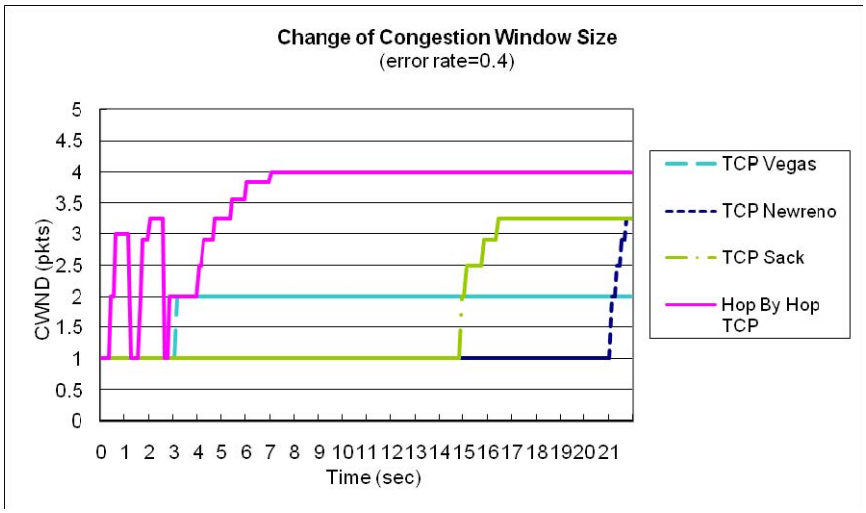
Parameter	Value
Packet Size	50 bytes
Buffer Size	10 packets
Buffer Management Scheme	DropTail
CWND Upper Bound	4
Link Bandwidth	5~10 Kbps
Error Rate	0.0~0.5
Number of TCP sessions	1
Number of Nodes	5~11
MAC Protocol	802.11
Routing Algorithm	DSR

4.1. Results of Performance Test

The results of performance test are shown in Fig. 4 to Fig. 7. As we can see from Fig. 4, the CWND of Hop-by-Hop TCP is kept at its upper bound, 4, for most of the time while others are fairly unstable. They even barely have a chance to reach 4. The stability of CWND at its upper bound implies high throughput and short delay time (packet deliver time) as we can see from Fig. 5 to Fig. 6. When error rate is high (0.4), Hop-by-Hop TCP not only have a higher throughput, its delay time is significantly lower than its counterparts when the number of hops is high. The number of retransmissions is significantly reduced as shown in Fig. 7.

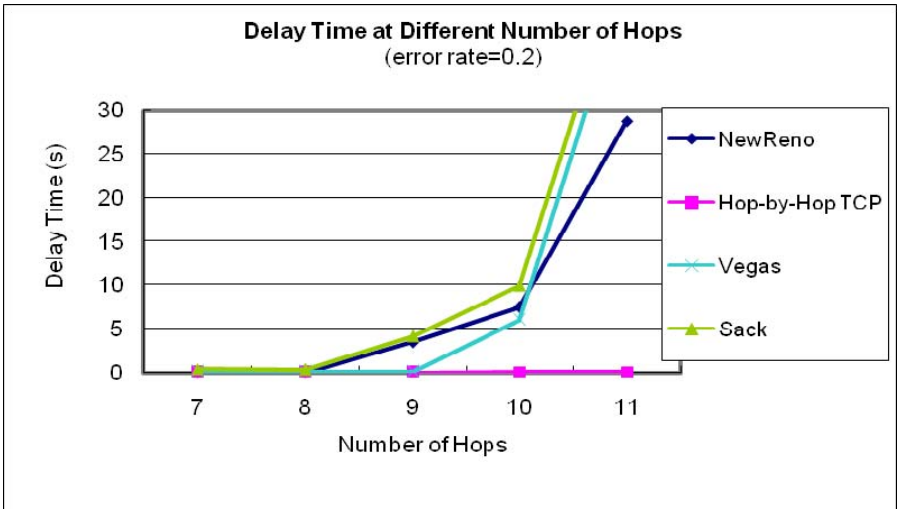


(a)

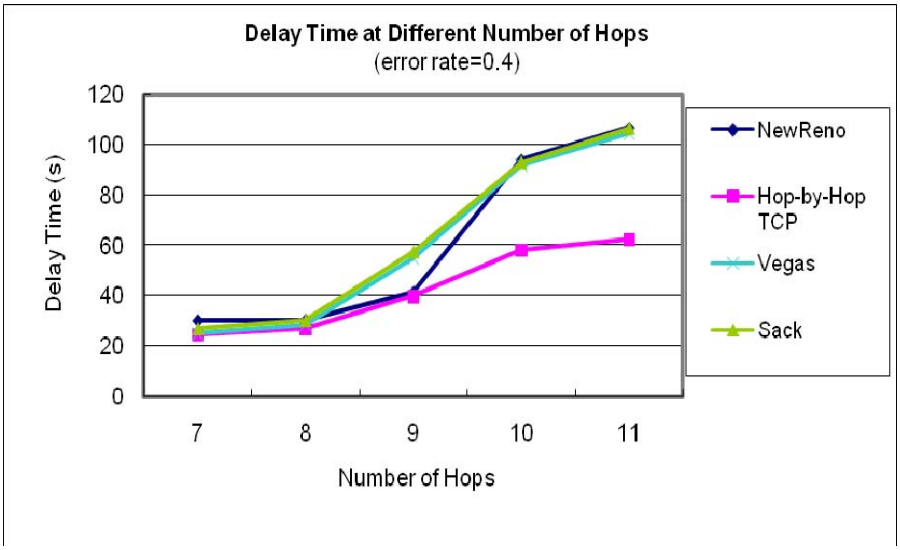


(b)

Fig. 4 Changes of CWND (a) error rate=0.2 (b) error rate=0.4

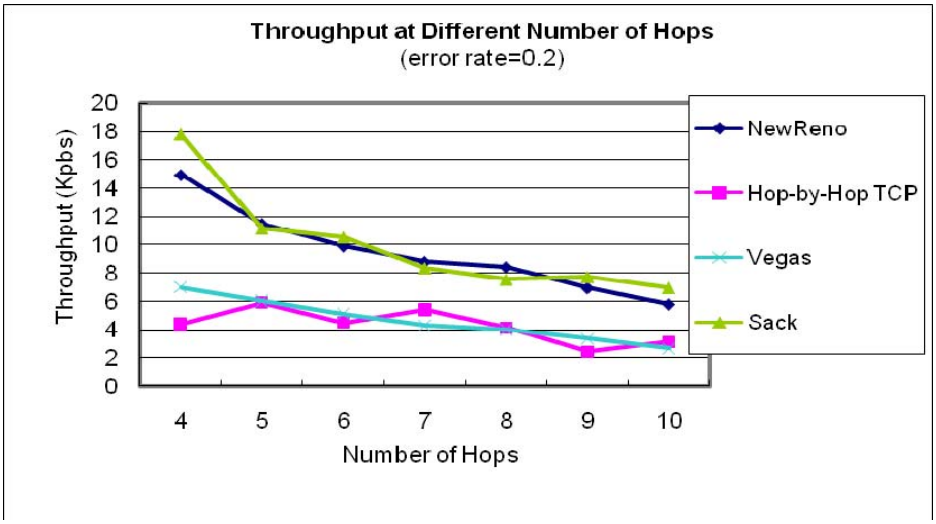


(a)

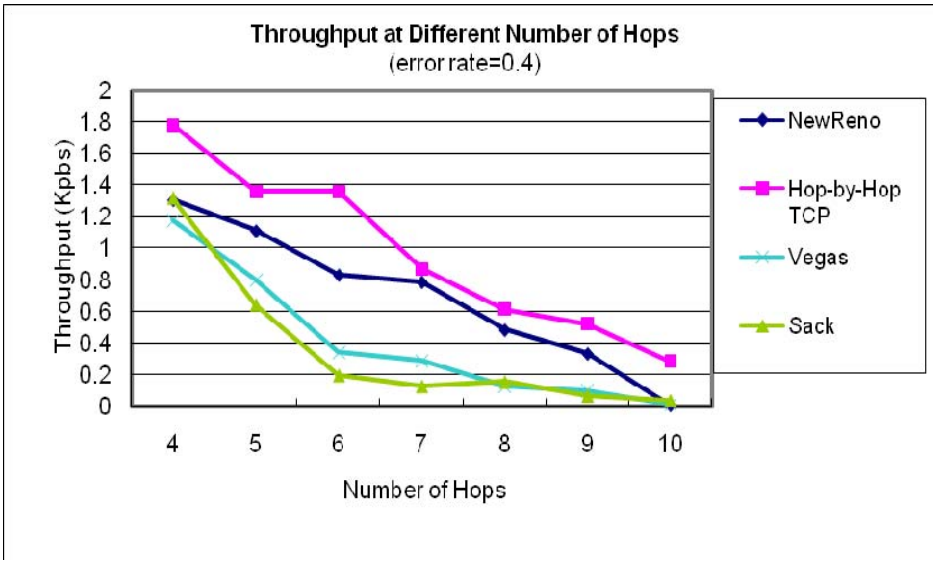


(b)

Fig. 5 Average Delay Time in Performance Test (a) error rate=0.2 (b) error rate=0.4

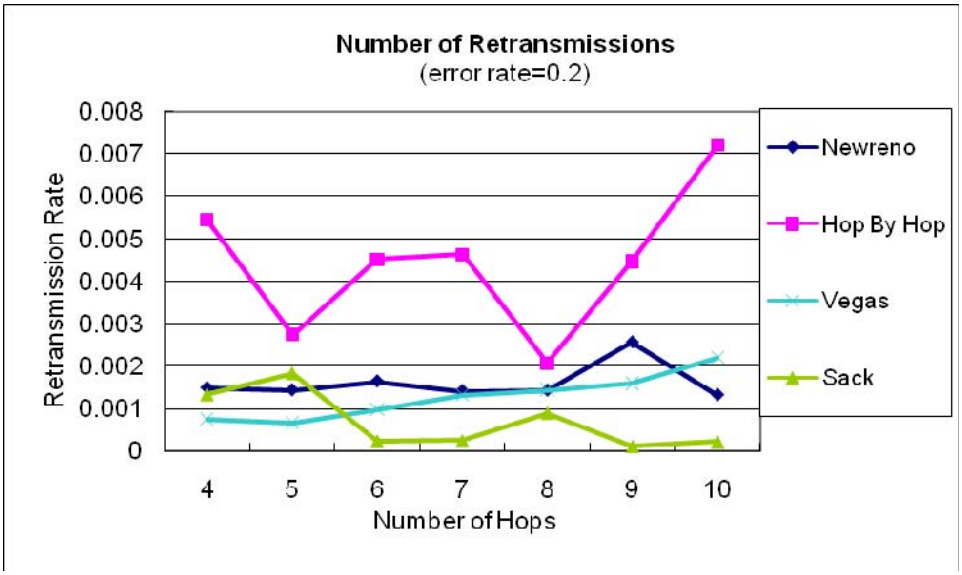


(a)

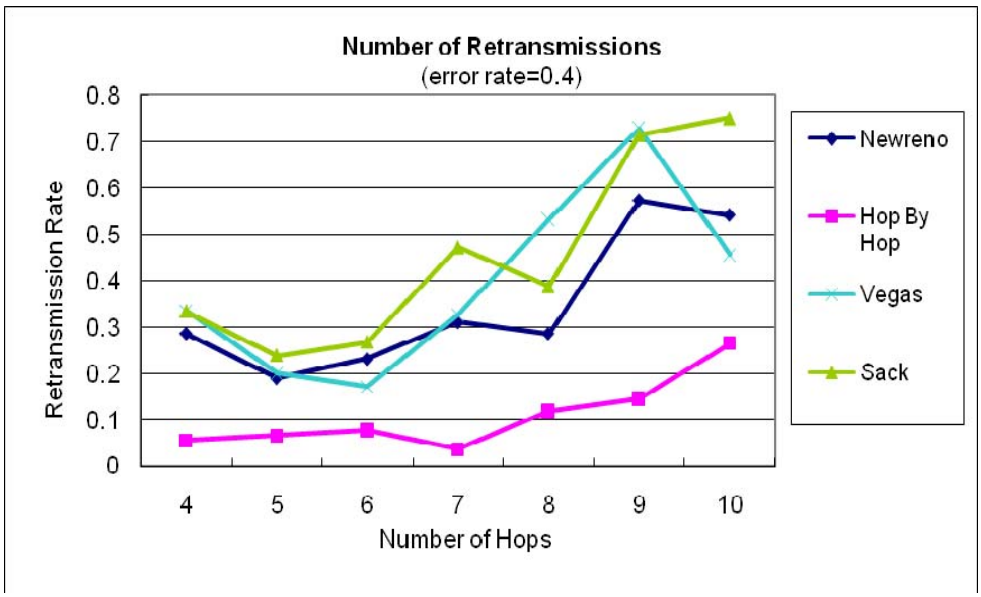


(b)

Fig. 6 Average Throughput in Performance Test (a) error rate=0.2 (b) error rate=0.4



(a)



(b)

Fig. 7 Number of Retransmissions (a) error rate=0.2 (b) error rate=0.4

4.2. Results of Fairness Test

Fig. 8 is the topology used in the fairness test. Two TCP sessions are injected into the network. One is from left to right and the other with different TCP version is injected from top to down. Both Hop-by-Hop TCP and TCP Vegas are tested under the coexistence of TCP NewReno. Jain's index [3], shown in Eq. 1, is used to measure fairness. The parameters used in the performance test are shown in Table 3.

$$\left(\sum_{i=1}^{n} x_i \right)^2 \Big/ n \sum_{i=1}^{n} x_i^2 \qquad (1)$$

As shown in Fig. 9, Jain's fairness index is very close to 1, when NewReno and Hop-by-Hop TCP are coexistent. TCP Vegas is obviously compromised if it coexists with NewReno. The second fairness test is to test self-synchronization capability. In this test, three streams of the same TCP are injected into the network at 0, 10, and 20 seconds to see how they are synchronized. Jain's index in every second is calculated and is shown in Fig. 10. From Fig. 10 we can see that NewReno has a very poor self-synchronization capability and Hop-by-Hop TCP has the highest one. There are two possible reasons that contribute to the high self-synchronization capability of Hop-by-Hop TCP. First, it has shorter RTT such that it can have faster and better congestion control. Secondly, it set an upper bound on the CWND window such that no TCP session can use excessive network resource.

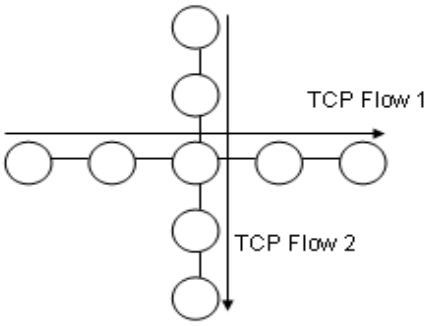


Fig. 8 Topology of Fairness Test

Table 3 Parameters in Fairness Test

Parameter	Value
Packet Size	50 bytes
Buffer Size	10 packets
Buffer Management Scheme	DropTail
Error Rate	0.4
Link Bandwidth	10 Kbps
Number of hops	4, 6, 8, 10
MAC Protocol	802.11
Routing Algorithms	DSR

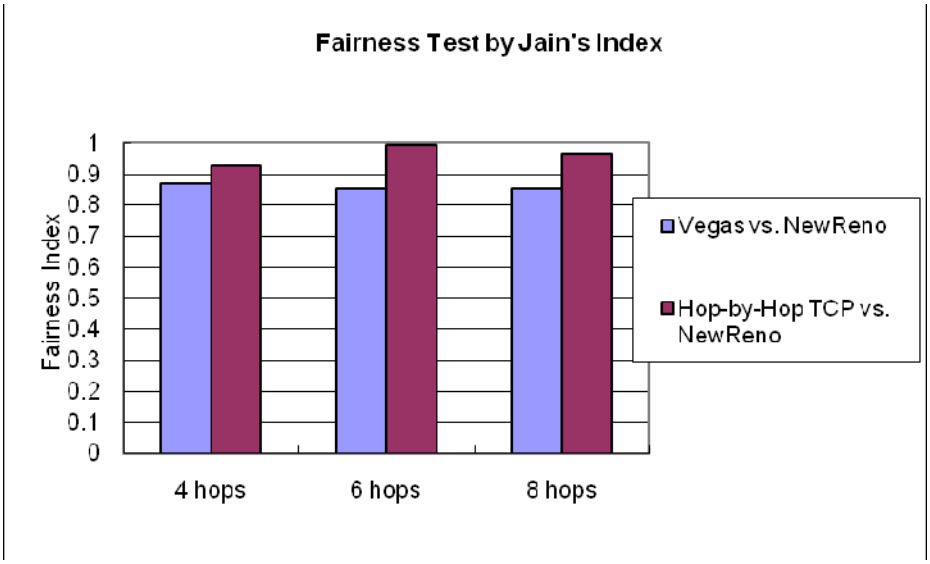


Fig. 9 Fairness Test 1

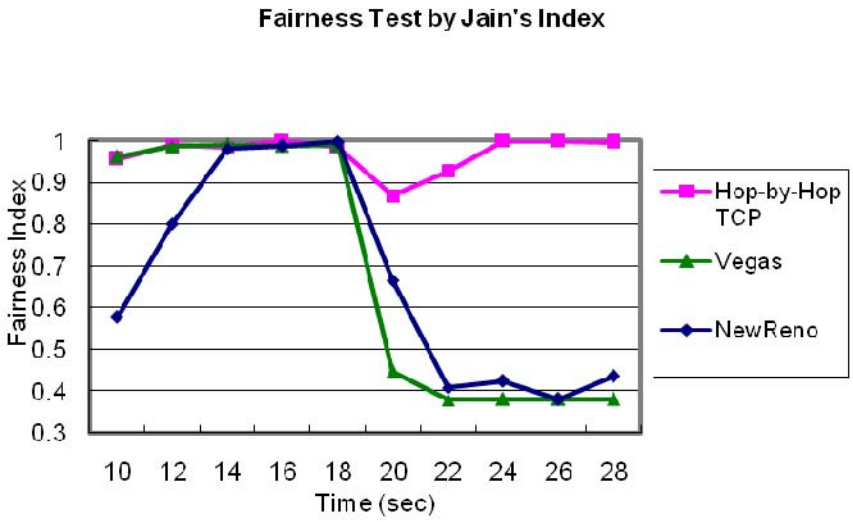


Fig. 10 Fairness Test 2

5. Concluding Remarks

In this paper, we apply the Hop-by-Hop TCP protocol to the sensor networks. Hop-by-Hop TCP makes every intermediate node in the transmission path of a TCP execute a local TCP (One-Hop TCP) to guarantee the transmission of each packet on each link. The retransmission of a lost packet is right at the transmitting end of the link where the packet is lost. It takes less time in average to deliver a packet in a high error rate and long path environment. The performance of our approach is evaluated by simulation using NS-2 simulator. Our experiments show that our proposed protocol outperforms TCP NewReno in throughput and average transmission time. The fairness requirement is also achieved while our proposed protocol coexists with other major TCP variants.

In the future, we can use One-Hop TCP to serve all TCP and even UDP so that the number of packets transmitted on the air can be greatly reduced.

References

1. K. Chandran, S. Raghunathan, S. Venkatesan, and R. Prakash, "A Feedback Based Scheme For Improving TCP Performance In Ad-Hoc Wireless Networks," *IEEE ICDCS*, vol. 8, no. 1, Feb. 2001, pp. 34-39.
2. K. Chen, Y. Xue, and K. Nahrstedt, "On setting TCP's congestion window limit in mobile ad hoc networks," *Proc. IEEE ICC 2003*, Anchorage, Alaska, May. 2003.
3. D. Chiu and R. Jain, "Analysis of the Increase and Decrease Algorithms for Congestion Avoidance in Computer Networks," *Computer Networks and ISDN Systems*, vol. 1, 1989, pp. 1-14.
4. Z. Fu, X. Meng, and S. Lu, "How bad TCP can perform in mobile ad-hoc networks," *Proc. IEEE ICNP'02*, Paris, France, 2002.
5. G. Holland and N. Vaidya, "Analysis of TCP performance over mobile ad hoc networks," *Proc. ACM Mobicom'99*, Seattle, WA, 1999.
6. R. Jain, D-M. Chiu, and W. Hawe, "A Quantitative Measure of Fairness and Discrimination for Resource Allocation in Shared Computer Systems," *DEC Research Report TR-301*, Sep. 1984.
7. S. Kopparty, S. Krishnamurthy, M. Faloutous, and S. Tripathi, "Split TCP for mobile ad hoc networks," *Proc. of IEEE GLOBECOM*, Nov. 2002.
8. Yao-Nan Lien and Ho-Cheng Hsiao, "A New TCP Congestion Control Mechanism over Wireless Ad Hoc Networks by Router-Assisted Approach," *Proc. of IEEE Workshop on Specialized Ad Hoc Networks and Systems*, Jun. 2007.
9. Yao-Nan Lien and Yi-Fan Yu, 2008, "Hop-by-Hop TCP over MANET", *Prof. of The First IEEE International Workshop on Wireless Network Algorithms (WiNA 2008)*, Dec. 9-12, 2008.
10. J. Liu and S. Singh, "ATCP: TCP for Mobile Ad Hoc Network," *IEEE Journal on Selective Areas of Communication*, vol. 19, no. 7, July 2001.
11. IETF RFC 2581, http://www.faqs.org/rfcs/rfc2581.html, Retrieved 1/15/2009.
12. http://en.wikipedia.org/wiki/Wireless_Sensor_Networks, Retrieved 1/15/2009.

Author

Yao-Nan Lien has been a professor of the Department of Computer Science at the National Chengchi University since 1995. He was the Chairman of the department from 1996 to 1999 and the Dean of the College of Science from 2000-2003.

He received his BS from National Cheng Kung University in 1979, and his MS and PhD degrees from Purdue University in 1981 and 1986, all in Electrical Engineering.

He was an assistant professor of Computer and Information Science at the Ohio State University from 1986 to 1989 and a Member of Technical Staff at AT&T Bell Laboratories from 1989 to 1993. From 1993 to 1995, he joined the Computer and Communication Research Laboratories, Industrial Technology Research Institute being the Deputy Director of the Computer Software Technology Division.

His research interests include mobile computing, communication networks, and database systems. He can be reached via email at lien@cs.nccu.edu.tw.