# Improving Performance of TCP-Vegas for Mobile IP Handoffs

**Cheng-Yuan Ho[1], I-Hsuan Chiu[2]**
[1]Department of Computer Science and Information Engineering, Asia University, Taichung City, Taiwan,
tommyho@asia.edu.tw
[2]Department of Civil Engineering, National Chiao Tung University, Hsinchu City, Taiwan,
sherry52168@g2.nctu.edu.tw

**ABSTRACT**

Mobile IP solves the primary problem of routing IP packets to mobile nodes. However, when using TCP Vegas over a mobile network, Vegas responds to a handoff by invoking a congestion control algorithm, thereby resulting in a degraded end-to-end performance in a mobile network. Furthermore, in Mobile IP handoffs, packets could be lost during movement detection and registration. These packet losses result in long communication pause, and successive timeouts. Accordingly, to reduce packet losses and timeout interval, we propose Mobility-Vegas, which uses Layer 2 trigger to detect handoffs, decreases the source's sending rate to reduce packet loss, and halts the retransmission timer to avoid increasing timeout interval. The proposed mechanism maintains end-to-end semantics, and operates under the existing network infrastructure. Mobility-Vegas presents a simple modification in the two end sides of a connection. Simulation result demonstrates that Mobility-Vegas features higher performance than Vegas in Mobile IP networks.

**Key words :** TCP, TCP Vegas, Mobile IP network, Handoffs, Transmission Control Protocol.

## 1. INTRODUCTION

Handoffs occur when the MN (mobile node) moves from its present location to a new network. If it moves from home network to the foreign network or from a foreign network to another foreign network, the MN must register its new location through the Registration Request and Registration Reply. However, if the MN moves from foreign network to its home network, the MN must deregister via the Home Agent. The communication will halt until the MN completes switching its point of attachment to a new IP subnetwork and registering its new location. The duration time between the MN losses the signal advertisement from an AR (access router) and completes the registration is called handoff duration.

MIP (Mobile IP) was originally designed to have the widest possible applicability without any assumptions about the underlying L2 (Layer 2) over which they would operate. This approach has the advantage of facilitating a clean separation between L2 and L3 (Layer 3) of the protocol stack; however, it results in the following built-in source delays:

(1) The MN may only communicate with a directly connected AR. This implies that a MN may only begin the registration process after an L2 handoff to a new AR is completed.

(2) The registration process takes some nonzero time to complete as the registration requests run through the network from the MN to the home agent. During this period, the MN is not able to receive IP packets.

The built-in source delay [1] degrades the handoff performance of MIP [7]. However, as it is well known, the built-in source delay can be reduced with information called an L2 trigger [1] which is sent from L2 to L3 to inform L3 of the occurrence of detailed events involved in the L2 handoff sequencing. One possible event is the completion of relocating a MN's L2 connectivity from an old AR to a new AR (L2 post-trigger). Another possible event is early notification of an upcoming change in the L2 connectivity of the MN (L2 pre-trigger).

In this paper, we focus on the performance of TCP Vegas during Mobile IP handoffs. We propose a modification of TCP Vegas, called Mobility-Vegas. Mobility-Vegas could detect the movement of a MN early by L2 trigger before handoffs, and then manage source's sending rate to prevent packet losses during handoffs, and finally set RTO (Retransmission Timeout) value in order to immediately resume the communication after handoffs. In addition, Mobility-Vegas is simple with very little overhead because only one bit is added in TCP option, and a small modification in the end side is made. This facilitates incremental deployment in today's Internet. Our intensive simulation shows that Mobility-Vegas significantly improves the overall TCP Vegas throughput in mobile environment.

The rest of this paper is organized as follows. Section 2 introduces TCP Vegas and Retransmission Timer. Related work is described in Section 3. We characterize motivation and Mobility-Vegas in section 4. Section 5 presents the simulation results and Section 6 summarizes this paper.

The work is supported by the Ministry of Science and Technology of Taiwan, the Improve AI Data Center Network Performance - Mitigate TCP Incast Problem by Dynamically Setting Threshold Values Project, under Grant no. MOST 108-2221-E-468-010 -, and partially supported by Asia University, Harnessing the Public Transportation System for Public Health Promotion Using Big Data Analytics of Passengers' Behavior Project, under Grant no. 107-ASIA-UNAIR-09.

## 2. TCP VEGAS AND RETRANSMISSION TIMER

### 2.1 TCP Vegas

TCP Vegas [5] uses the difference in the expected and actual flow rates to estimate the available bandwidth in the network. When the network is not congested, the actual flow rate would be close to the expected flow rate. On the other hand, if the actual rate is much smaller than the expected rate, it indicates that buffer in the network is filling up and the network is approaching congestion. This difference in flow rates can be calculated as *Diff = Expected - Actual*, where *Expected* and *Actual* are the expected and actual rates, respectively.

(i) Congestion Avoidance

In its congestion-avoidance phase, Vegas uses two threshold values, $\alpha$ and $\beta$ (whose default values are 1 and 3, respectively), to control the adjustment of the congestion window size at the source host. Let *d* denote the minimum observed packet round-trip time (also known as BaseRTT), *D* denotes the actual RTT (round-trip time), and *W* denotes the size of the congestion window size, then *Expected = W/d* and *Actual = W/D*. In addition, *W* is measured in segments as is normally done in any TCP version. The estimated backlog of packets in the network queues can then be computed as

$$\Delta = (Expected - Actual) \times BaseRTT = W \times \frac{(D-d)}{D}. \quad (1)$$

For every RTT, the congestion-avoidance algorithm adjusts *W* as follows:

$$W \leftarrow \begin{cases} W+1, & if \Delta < \alpha \\ W-1, & if \Delta > \beta \\ W, & otherwise (\alpha \le \Delta \le \beta). \end{cases}$$

Conceptually, Vegas tries to keep at least $\alpha$ packets but no more than $\beta$ packets queued in the network. Thus, when there is only one Vegas connection, *W* converges to a point that lies between *window*+ $\alpha$ and *window*+ $\beta$ where *window* is the maximum window size without considering the queuing in the network.

(ii) Slow Start

Like Reno, Vegas uses a slow-start mechanism that allows a connection to quickly ramp up to the available bandwidth. However, unlike Reno, to ensure that the sending rate will not increase too fast to congest the network during the slow start, Vegas doubles its congestion window size only every other RTT, and calculates the difference between the flow rates (*Diff*) and $\Delta$ given in (1) in every other RTT. When $\Delta > \gamma$ (whose default is 1), Vegas leaves the slow-start phase, decreases its congestion window size by 1/8 and enters the congestion-avoidance phase.

(iii) Retransmission

As in Reno, a triple-duplicate ACK always results in packet retransmission. However, in order to retransmit the lost packets quickly, Vegas extends Reno's fast retransmission strategy. Vegas measures the RTT for every packet sent based on fine-grained clock values. Using the fine-grained RTT measurements, a timeout period for each packet is computed. When a duplicate ACK is received, Vegas will check whether the timeout period of the oldest unacknowledgement packet is expired. If so, the packet is retransmitted. This modification leads to packet retransmission after just one or two duplicate ACKs. When a non-duplicate ACK that is the first or second ACK after a fast retransmission is received, Vegas will again check for the expiration of the timer and may retransmit another packet. Note that, packet retransmission due to an expired fine-grained timer is conditioned on received certain ACKs.

After a packet retransmission was triggered by a duplicate ACK and the ACK of the lost packet is received, the congestion window size will be reduced to alleviate the network congestion. There are two cases for Vegas to set the *W*. If the lost packet has been transmitted just once, the *W* will be three fourth of the previous congestion window size. Otherwise, it is taken as a sign for a more serious congestion, and one half of the previous congestion window size will be set into *W*. Notably, in case of multiple packet losses occurred during one RTT that trigger more than one fast retransmission, the congestion window will be reduced only for the first retransmission.

If a loss episode is severe enough that no ACKs are received to trigger fast retransmit algorithm, eventually, the losses will be identified by Reno-style coarse-grained timeout. When this occurs, the slow start threshold will be set to one half of *W*, and then the *W* will be reset to two, and finally the connection will restart from slow start.

### 2.2 Retransmission Timer

TCP provides connection-oriented and reliable services between two hosts that are responsible in ensuring the transfer of datagrams from source to the respective destination. TCP sends the data in variable length of segments. The sender will stop the transmission after all the bytes in the window has been sent. Eventually, a timeout will pass and the missing segment will be retransmitted. The RTO value depends on the RTT that can be defined as a measured elapsed time between sending a window data octets with a particular sequence number and receiving an acknowledgement [2]. Also, TCP is tuned to perform well in traditional wireline fixed networks where packet losses occur mostly because of congestion [3]. However, in the mobile network, packet losses usually occur due to a handoff rather than congestion.

During the handoff, an unsuccessful segment received at the

destination will retransmitted. The sender assumes that the segment was lost after the timeout expired. There are two types of RTO. One is specified in [2], and the other has been proposed by Jacobson in his paper [6]. Both of the standard RTO and Jacobson's RTO are using an exponential timeout backoff. This is because the exponential timeout backoff can avoid the traffic loaded with the unsuccessful transmission. However, the timeout value increases twice for each retransmission of the same segment and this may delay the retransmission. Therefore, the successive TCP timeouts then increase the TCP timeout interval such that, even after Mobile IP handoff has completed, TCP will not immediately resume the communication. In other words, in Mobile IP, a MN may inevitably have long handoff delay resulting in a long communication pause at the sender. For the TCP protocol, this service disruption is perceived as an indication of congestion that requires a TCP exponential backoff and slow-start.

## 3. RELATED WORK

J. W. Kwon et al. proposed two schemes, TCP-MD (Movement Detection) and TCP-R (Registration) in [8]. TCP-MD can detect the movement of a MN early on, whereas TCP-R can force the source to freeze data transmission during registration. However, only using TCP-MD or TCP-R is not enough to improve the performance of TCP. FxRTO (Fixed RTO) [9] is proposed to decrease the pauses in communication. The FxRTO allows many segments to be transmitted even during the handoff; however, the increase of segments sent may affect the dropping or losing. Another disadvantage is that the FxRTO of a sender may not suit for various environments.

K. Omae et al. proposed an MN extension [10] employing a buffering function to improve the handoff performance. Before L2 handoff, the MN extension sets pre-buffer timer and then buffers the ACKs during the pre-buffer time. After the MN completes its L2 and L3 handoffs, it sends all buffered ACKs to the CN to resume the communication. This way could reduce the source's sending rate and packet loss during handoffs, but it is difficult to set the value of the pre-buffer timer. It is because the pre-buffer time should be longer than RTT and the total of the pre-buffer time, L2 and L3 handoffs should be shorter than sender's retransmission timer. In addition, in traditional handoff mechanism, the time for L2 handoff is about 180 ms, and for L3 handoff is about 3 seconds.

Demo-Vegas [4] is proposed to improve the performance of Vegas after a Mobile IP handoff. Although, after handoffs, Demo-Vegas is able to detect the movement of both a sender and a receiver based on their COAs, and re-measure the BaseRTT if necessary, Demo-Vegas could not reduce packet loss and TCP timeout interval during handoffs.

## 4. MOBILITY-VEGAS

### 4.1 Motivation

TCP Vegas is a rate based mechanism, it adjusts the congestion window based on the current congestion window size, BaseRTT, and newly measured RTT. Vegas can successfully avoid the congestion in the network, so there are implementations of Vegas in some operating systems such as Linux and NetBSD. However, during and after handoffs, Vegas has some problems. As a result, on a Mobile IP network, Vegas may not utilize the bandwidth efficiently. We propose a variant of TCP Vegas, Mobility-Vegas, to solve this issue. Our method does not influence the original scheme on the wired network.

### 4.2 The Scheme of Mobility-Vegas

In order to propose a widely applicable solution that achieves better handoff performance over all L2 technologies, we make two assumptions related to the L2 trigger:

(1) The MN can acquire an L2 pre-trigger and L2 post-trigger over all access technologies.

(2) However, these cannot include a new AR IP address identifier as a parameter of the L2 trigger over all access technologies.

Based on these assumptions, the key idea of Mobility-Vegas is described as follows. Mobility-Vegas uses one bit in TCP option as the IWH (I Will Handoff) flag. Figures 1 and 2 show the operation of the receiver and the sender respectively. First, we see the timing diagram of the receiver handoff (Figure 1). After receiving L2 pre-trigger signal (and before L2 and L3 handoffs), the receiver will mark the IWH flag of ACKs in order to tell the sender. Then the sender will halt the retransmission timer and send one packet per RTT until receiving receiver's binding update, which includes the new COA (Care of Address). After that, the sender immediately resumes the communication. The reason of halting the retransmission timer is to avoid unnecessary exponential backoff and retransmitted packets, and it is good for performance when sending one packet per RTT because this may reduce the handoff latency, decrease the probability of packet loss and increase the throughput.

Similarly, the sender will mark the IWH flag of data packets when it gets L2 pre-trigger signal. While the receiver receives these packets, it knows the sender will handoff later, so it will buffer the ACKs. The sender halts the retransmission timer during L2 and L3 handoffs. It is because the sender cannot receiver any ACK in this period. After handoffs, the sender immediately transmits data packets to the receiver. Then, the receiver sends the buffered and new ACKs to the sender. All of the above are shown in the Fig. 2.

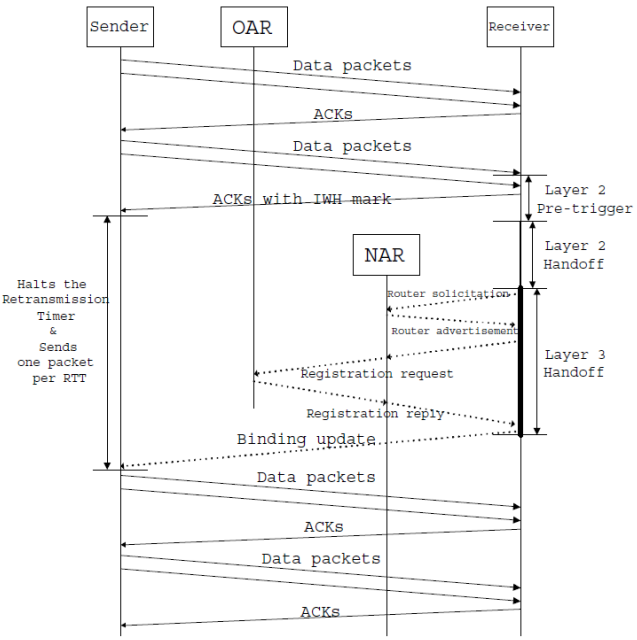


**Figure 1:** Timing diagram of the proposed receiver handoff.

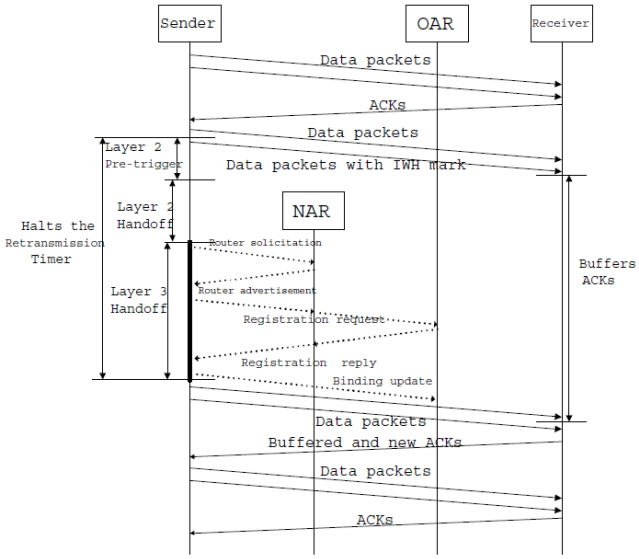


**Figure 2:** Timing diagram of the proposed sender handoff.

Next, in order to understand Mobility-Vegas more clearly, a step-by-step procedure for handoff is given below.

1. When the L2 decides to initiate L2 handoff, it sends an L2 pre-trigger to L3.

2. L3 sends the message "L2 handoff will initiate" to the transport layer when the L2 pre-trigger is received.

3. While that message is received, the transport layer marks the IWH flag of the packets for transmission.

4. If this procedure is running in a sender, it will halt the retransmission timer.

5. When the L2 handoff is completed, L2 sends an L2 post-trigger to L3.

6. L3 sends a router solicitation to the new AR and receives the router advertisement via L2 when the L2 post-trigger is received.

7. When a router advertisement from the new AR is received, L3 sends the Registration Request to the new AR and waits the Registration Reply.

8. L3 changes the default router to a new AR and sends the signal to the transport layer.

9. When the signal is received, the communication will be resumed.

Since the space is limited, we omit the pseudo codes in this section. The proposed scheme can improve the performance of TCP Vegas in the traditional Mobile IP networks based on the simulation result in the following section.

## 5. PERFORMANCE EVALUATION

### 5.1 The Simulation Environment

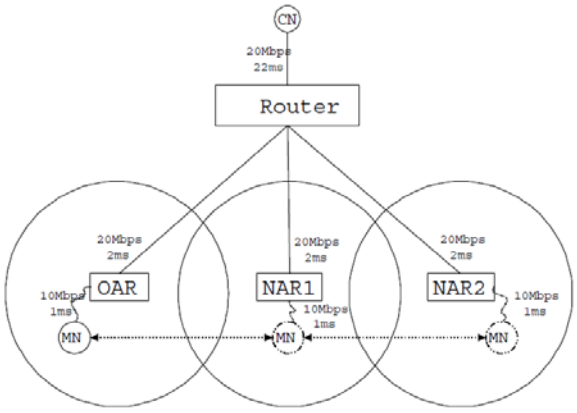


**Figure 3:** A simple topology for simulations.

The simulation experiments are conducted using the *ns2* [11], version 2.35. A simulation network topology is shown in Fig. 3, where CN and MN represent end hosts. CN and MN are the two end sides which execute Vegas, Demo-Vegas or Mobility-Vegas. The application service in our simulation is FTP. The receiver sends an ACK for every data packet received. For the convenience of presentation, we assume that all window sizes are measured in number of fixed-size packets, which are 1000 bytes. Router, OAR, NAR1 and NAR2 represent four finite-buffer gateways. The buffer size in each gateway is set to 10 packets. For the constant-load experiment, drop-tail gateways with FIFO service are assumed. The bandwidth is 20 Mbps for all wired links. The propagation delay is 22 ms from CN to the router, 2 ms from the router to an AR (OAR, NAR1, or NAR2), respectively.

From an AR to the MN, the bandwidth is 10 Mbps and wireless transmission delay is a multiple of 1 ms which includes both packet transmission delay and the propagation delay. The former may account the layer 2 retransmission due to unsuccessful frame delivery, while the later can be ignored because the propagation delay is much smaller comparing with the packet transmission delay. In addition, this is a two dimensional plane in topology. The distance of radio coverage for the agent is 90 meters. The positions of OAR, NAR1, and NAR2 are (200, 300), (350, 300), and (500, 300), respectively.

### 5.2 Numerical Result

#### 5.2.1 A fixed sender and a mobile receiver

In this simulation, CN is the sender and MN is the receiver. The BaseRTT is about 50 ms if the packet is transmitted successfully at the first time. The MN moves with a speed of 10 m/s from (150, 275) to (350, 275) at the 10$^{th}$ second, then moves to (550, 275) at the 35$^{th}$ second. It starts to come back (350, 275) at the 60$^{th}$ second, then return to (150, 275) at the 85$^{th}$ second. When the MN is in the foreign network, the datagrams are routed from CN to OAR, tunneled from OAR to NAR, and NAR de-tunnels these datagrams to the MN. The ACKs are routed directly from MN to CN through the NAR. The average throughput of Vegas, Demo-Vegas, and Mobility-Vegas are shown in Fig. 4, where we can observe that Mobility-Vegas outperforms the others during and after handoffs. It is because the average throughput of Vegas is about 5.55 Mbps, and the average throughput of Mobility-Vegas is 1.5 times as great as that of TCP Vegas, and 1.2 times as high as that of Demo-Vegas. In other words, the average throughput of Mobility-Vegas (about 8.26 Mbps) is the highest among these three Vegas versions.

Figure 5 shows the throughput of TCP Vegas, Demo-Vegas, and Mobility-Vegas. From this Figure, we could see the handoff time of Mobility-Vegas is decreased by about half from 7 (or 8) seconds, which is the handoff time of TCP Vegas or Demo-Vegas, to 3 (or 4) seconds. Moreover, the throughput of Mobility-Vegas is not 0 during handoffs. For example, the handoff time of our proposed mechanism is from the 21.3$^{th}$ second to the 24.9$^{th}$ second, and the average throughput is about 200 Kbps during this period; on the other hand, the TCP Vegas' handoff time is from the 21.3$^{th}$ second to the 28.5$^{th}$ second, and during this time interval, it retransmits the packets which may be lost after the timeout expired. The percentages of dropping or losing packets of three TCP versions during handoffs are illustrated in Table 1. Mobility-Vegas' percentage of dropping or losing packets is 3.2%, which means there are about 2 packet losses during handoffs. It is the lowest value compared to that of TCP Vegas and Demo-Vegas. In addition, the percentage of TCP Vegas is 5 times as much as that of Mobility-Vegas.

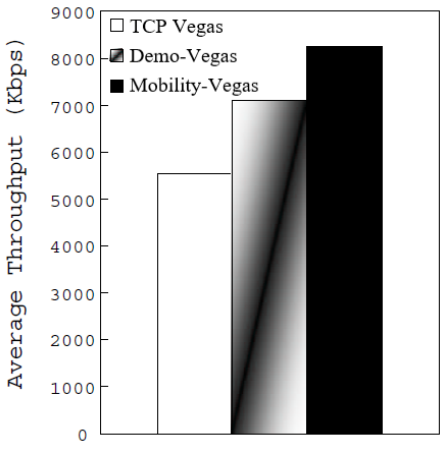


**Figure 4:** Average throughputs.

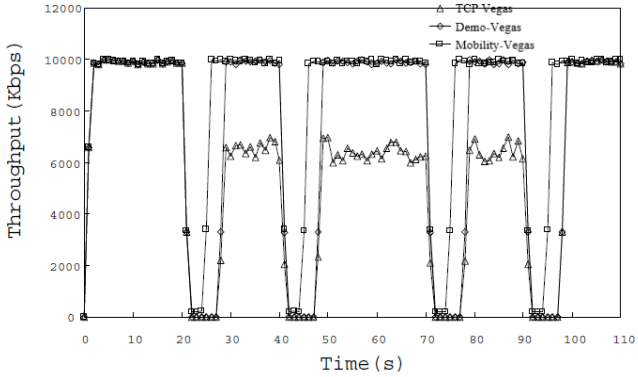


**Figure 5:** The throughput of TCP Vegas, Demo-Vegas, and Mobility-Vegas.

**Table 1:** Total number of packets drop or loss during handoffs

| Version | Packet Drop/Loss (%) |
|---|---|
| TCP Vegas | 16.1% |
| Demo-Vegas | 16.1% |
| Mobility-Vegas | 3.2% |

#### 5.2.2 A fixed receiver and a mobile sender

In this simulation, we investigate the performance of three Vegas versions when the sender is a mobile node and the receiver is fixed. Basically, the sender sends packets through the NAR1 or NAR2 (if NAR1 or NAR2 allows) to the receiver, then the receiver sends the ACKs through the OAR and NAR back to the sender with a triangular routing path when the sender is in a foreign network. The average throughput of Mobility-Vegas is about 8.5 Mbps, which is about 1.46 times as great as that of TCP Vegas, and 1.16 times as high as that of Demo-Vegas. In other words, the average throughput of Mobility-Vegas is the highest among these three Vegas versions. Moreover, the percentage of dropping or losing packets and throughput of three schemes are similar to those in the first simulation results.

Due to the limited space, we only show the results of configuration with a fixed node and a mobile node.

Furthermore, the diagrams of configuration with two mobile nodes are just like that with one fixed node and one mobile node. From simulation results, we could observe that the performance of Mobility-Vegas is much better than Vegas when one side is fixed and the other side is mobile. It is because Mobility-Vegas could halt retransmission timer, decrease the sending rate, and resume the communication in time. Thus, when both end sides are mobile, the throughput of Mobility-Vegas will be still better than Vegas.

## 6. CONCLUSION

We propose and evaluate a new variant of TCP Vegas, called Mobility-Vegas, to improve the performance during Layer 2 and Layer 3 handoffs. In this work, we achieve a significantly higher throughput comparing with TCP Vegas and Demo-Vegas on a Mobile IP network. Mobility-Vegas could detect handoffs by using Layer 2 trigger, reduce packet loss during handoffs, and immediately resume the communication when Mobile IP handoff has completed. From the simulation and numerical result, it shows that Mobility-Vegas is more suitable than Vegas on a Mobile IP network. In addition, Mobility-Vegas will still work well when the IPv4 environment changes to IPv6. Furthermore, the 'IWH' bit setting propagates past the NAT mechanism that enables mobile IP, to the remote host, forcing that host to do corresponding steps. The proposed scheme is simple and can be easily implemented on existing operating systems. We will focus on its coexistence with same or other TCP implementations and its performance with high speed movement in our future work.

## REFERENCES


1. K. El-Malki, P. R. Calhoun, T. Hiller, J. Kempf, P. J. McCann, A. Singh, H. Soliman, and S. Thalanany. **Low Latency Handoffs in Mobile IPv4**, draft-ietfmobileip-lowlatency-handoffs-v4-09.txt, June 2004.
2. J. Postel. **Transmission Control Protocol Specification**, IETF RFC 793, September 1981.
3. M. Allman, V. Paxson, and W. Stevens. **TCP Congestion Control**, IETF RFC 2581, April 1999.
4. C.-Y. Ho, Y.-C. Chan, and Y.-C. Chen. **An Efficient Mechanism of TCP-Vegas on Mobile IP Networks**, in *8th Proc. of the IEEE Global Internet Symposium in conjunction with IEEE INFOCOM*, March 2005.
5. L. Brakmo and L. Peterson. **TCP Vegas: End-to-End Congestion Avoidance on a Global Internet**, *IEEE J. Sel. Areas Commun.*, Vol. 13, No. 8, pp. 1465-1480, October 1995.
6. V. Jacobson. **Congestion Avoidance and Control**, in *Proc. of ACM SIGCOMM'88*, Stanford, August 1988.
7. H. Soliman, C. Castelluccia, K. Malki, and L. Bellier. **Hierarchical Mobile IPv6 mobility management (HMIPv6)**, draft-ietf-mipshop-hmipv6-04.txt, December 2004.
8. J. W. Kwon, H. D. Park, and Y. Z. Cho. **An Efficient TCP Mechanism for Mobile IP Handoffs**, TENCON'01, Vol. 1, pp. 278-281, August 2001.
9. Y. Mohamed, N. Fisal, and A. Mohd. **Performance of TCP on Mobile IP Network During Handoffs**, *in Proc. of SCOReD 2002*, pp. 390-393, July 2002.
10. K. Omae, T. Ikeda, M. Inoue, I. Okajima, and N. Umeda. **Mobile Node Extension Employing Buffering Function to Improve Handoff Performance**, in *Proc. of WPMC'2002*, Vol. 1, pp. 62-66, October 2002.
11. NS2. http://www.isi.edu/nsnam/ns/