# Smooth Handoff with Enhanced Packet Buffering-and-Forwarding in Wireless/Mobile Networks

Chun-Ting Chou

Philips Research USA
345 Scarborough Road,
Briarcliff Manor, NY 10510, USA
chun-ting.chou@philips.com

Kang G. Shin

Real-Time Computing Laboratory
The University of Michigan
Ann Arbor, MI 48109-2122, U.S.A.
kgshin@umich.edu

*Abstract*— **Packet buffering-and-forwarding is a simple but essential mechanism and has been widely used with other mechanisms to provide seamless handoffs in many wireless/mobile networks. However, some undesirable side effects of this mechanism, if not managed appropriately, can easily diminish its effectiveness in providing seamless inter-cell transitions during a handoff. We first examine these side effects and show how inappropriate buffer management by a mobility agent could affect the TCP performance. The throughput of TCP is then studied with special emphasis on the effects of a handoff. We enhance the conventional buffering-and-forwarding by proposing the Last-Come-First-Drop (LCFD) buffer management policy (to be employed by mobility agents) and post-handoff acknowledgement suppression (to be used by mobile nodes). Our enhancements are backward compatible and suitable for the gradual/incremental deployment. By deriving an analytical model and conducting numerical analysis, we show that our scheme can improve the TCP throughput up to 30%. Finally, we conduct the *ns-2* based simulation to confirm these numerical results, and demonstrate the applicability of the analytic model for predicting TCP throughput in other handoff schemes.**

*Index Terms*— **Smooth handoff, MobileIP, TCP congestion control, retransmission timeout, fast retransmit**

## I. Introduction

Most of the current Internet applications, such as HTTP, FTP, email, and telnet, use TCP for reliable transfer of data. The dominant usage of TCP in the wired Internet calls for smooth and transparent migration of TCP to the emerging wireless/mobile networks. As a prevailing transport-layer protocol, the TCP uses two different strategies for end-to-end reliable data transfer: timeout-based and duplicate-ACK-based packet retransmissions. The TCP assumes every packet loss to have resulted from network congestion, and uses these retransmission mechanisms to recover from packet losses. This assumption is reasonable because transmission errors occur rarely in wired networks. However, packet loss in a wireless network occurs much more frequently than in its wired counterpart due mainly to transmission errors over a wireless link or temporary disconnection from the network caused by user mobility. The TCP may misinterpret these packet losses as a result of network congestion and then invoke the TCP congestion control. This "over-triggering" of TCP congestion control, together with the accompanying reduction of TCP congestion window size, degrades the TCP throughput significantly.

Many variations of TCP have been proposed to avoid TCP congestion control triggered by packet losses over a wireless link. In the Indirect-TCP [1] and Snoop TCP [2], packet loss is concealed from the TCP sender by the base station (BS), and packet retransmission is handled by the BS. Some TCP options, such as Explicit-Congestion-Notification (ECN) [3] and Selective-Acknowledgment (SACK) [4], are also adopted such that the TCP sender can recover from packet losses more efficiently. Along with these transport-layer solutions, one may also rely on reliable link-layer transmission mechanisms, such as Forward Error Correction (FEC) and Automatic Repeat reQuest (ARQ), to make packet loss invisible to the TCP. A scheme that uses Delayed-Duplicate-ACKs [5] along with link-layer retransmission, was proposed to hold the third duplicate ACK such that the TCP sender will not fast retransmit the lost packet.

With these proposals to improve TCP performance over lossy wireless links, the remaining challenge will be to solve the problems caused by user mobility, which is the main intent of this paper. Let us consider the handoff in a MobileIP network as an example. The mobile node in a MobileIP network cannot "detect" its migration to a neighboring subnet until it misses three consecutive agent/router advertisements from its previous access router. The default advertisement period in the MobileIP draft is 1 second, implying that the temporary disconnection (namely, the handoff latency) could last for at least 3 seconds. Such a handoff latency can be long enough to cause many packet losses and trigger TCP retransmission timeout(s). In general, there are two ways to alleviate this problem: (1) using fast handoff to avoid TCP timeouts and (2) using smooth handoff to eliminate packet losses. Most fast-handoff schemes rely on different forms of "triggers" from link-layer handoffs [6], [7], [8], [9]. Since the link-layer handoff latency, typically ranging from 300 to 500 ms, is much smaller than the IP-layer handoff latency, one can reduce the IP-layer handoff latency and the subsequent TCP timeouts.

A smooth handoff is used to avoid packet loss during either a MAC-layer or IP-layer handoff. It can be realized in many different ways. The simplest solution is to use packet buffering-and-forwarding, in which in-flight packets are buffered in the old BS during a handoff and then forwarded to the new BS after the handoff is completed [9], [10], [11]. The concept of multicast can also be used to achieve a smooth handoff [2], [12]. The packets destined for a mobile node are multicast to its neighboring BSs before the mobile node actually moves into the coverage of a neighboring BS. This way, the packets that cannot reach the mobile node during a handoff can be ready for transmission to the mobile node via the new BS immediately after the handoff. The "persistent mode" of TCP is also used to improve the TCP performance. Either the mobile node [14] or the BS on behalf of the mobile node [15], will advertise a zero window size to the TCP sender before a handoff. Once the TCP sender receives an ACK indicating that the receiver window size is equal to zero, it will cancel the retransmission timer and stop transmission until the receiver's window is open again after the handoff.

Even though the link-layer-assisted fast handoff schemes can reduce the IP-layer handoff latency, it has been shown that packet losses can still occur because of the nonzero link-layer handoff latency [13]. It alone cannot solve the problem without

The work reported in this paper was supported in part by the US AFOSR under Grant No. F49620-00-1-0327.

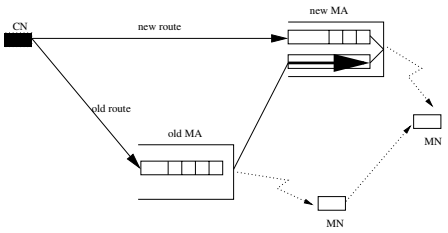


Fig. 1. The smooth handoff in a Mobile IP network

the help of some smooth handoff mechanism. Among the approaches to achieving a smooth handoff, the "persistent mode"-based schemes require the BSs or mobile nodes to send out zero window-size advertisements before a handoff. Nevertheless, detecting an upcoming handoff is difficult and relying on signal strength could be very risky. The signal strength may fall simply due to channel fading or "ping-pong" effects near the cell boundary. Any "false alarm" will force the TCP sender to cease its transmission, thus reducing the overall throughput. The multicast-based smooth handoff schemes consume more network bandwidth and buffer spaces in the BSs. In view of these facts, we focus our discussion on the relatively simple approach — packet buffering-and-forwarding. The simplicity and effectiveness (in avoiding packet losses) makes itself an essential element in the emerging high-speed, high-mobility wireless/mobile networks. Despite these advantages, such a simple approach may introduce some new problems. For example, buffer overflow may occur in the BS during a handoff in high-speed networks. In this paper, we will first show how a buffer management policy can affect the TCP throughput. Then, we propose two enhancements to fix these problems such that the TCP throughput can be improved.

The rest of this paper is organized as follows. Section II states the assumptions used in this paper and thoroughly investigates the TCP performance problem resulting from the packet buffering-and-forwarding. Section III presents our solution and provides a mathematical model for the TCP throughput analysis. The numerical results based on the analytical model are presented in Section IV, while in Section V, the numerical results are verified by the *ns-2*-based simulation. Finally, conclusions are drawn and the direction of our future work is discussed in Section VI.

## II. Problem Statement and Assumptions

Figure 1 shows a smooth handoff using packet buffering-and-forwarding in a wireless/mobile network. The Mobility Agents (MAs) take charge of packet buffering and forwarding for a mobile node. The MAs can "reside" in the BSs of a cellular network, the Access Routers (ARs) in a MobileIP network, or the Access Points (APs) in wireless LANs. The old MA will buffer in-transit packets and all other packets that cannot be transmitted during a handoff. The packet buffering could be initiated by the old MA when it receives the link-layer signal indicating a link-down. The packet forwarding is triggered either by (1) a link-layer signal indicating a link-up, or (2) a forwarding request from the mobile node via the new MA. The link-layer trigger helps speed up the resumption of data transfer when a previous link-down was due to channel fading, not a handoff. The packet forwarding for (1) actually means that the old MA resumes its packet transmission to the mobile node directly. The forwarding request of (2) simply performs packet forwarding via the connection between the MAs. This forwarding request may follow right after the mobile node's reconnection to the new subnet, e.g., after an association with a new AP or a binding update in a MobileIP network. This way the movement-detection algorithm in the original MobileIP can be preserved. Other than this packet buffering-and-forwarding, BSs, ARs and APs work as usual and are assumed to be TCP-unaware.

Since the MA may need to simultaneously handle many TCP sessions from many mobile nodes, the buffer could become a bottleneck. Thus, a buffer overflow may occur [16]. If the average end-to-end transmission capacity is high and the maximum TCP congestion window size is large, the old MA may have to buffer many packets for the mobile nodes, which is likely to result in a buffer overflow. Without appropriate buffer management, the buffer overflow can easily offset the benefits gained from packet buffering-and-forwarding. One of our goals in this paper is to study the impact of buffer overflow on the TCP performance and propose a better buffer management policy.

### A. Retransmission Timeout with Buffer Overflow

As mentioned in the Introduction, an immediate problem associated with a handoff is the potential retransmission timeout. For example, the MobileIP handoff latency can be up to 3 seconds, which can easily force the TCP to time out. Even though this handoff latency can be reduced by using some link-layer enhancements, some experiments show that TCP can still time out during a link-layer handoff [13]. This is because many operating systems have finer TCP timer granularity[1] so that the link-layer handoff latency is long enough to cause TCP retransmission time out. These handoff-induced timeouts, which should not trigger the TCP congestion control, will reduce the current TCP congestion window size (*cwnd*) to 1, and reset the slow start threshold (*ssthresh*) to $\frac{cwnd}{2}$. Consequently, it reduces the TCP throughput. To make it even worse, the exponential backoff of retransmission timeout (RTO) after consecutive timeouts may cause the TCP to stall for a longer period of time. As shown in Figure 2, the TCP sender will retransmit the first unacknowledged packet at time A and double the RTO. Without the packet buffering-and-forwarding, the TCP will wait for the second timeout and thus remain idle between time B and D even though the handoff is completed at time B. Fortunately, the old MA will forward the buffered packets to the mobile node. These packets will generate ACKs such that the TCP sender can resume transmission at time C. However, if the buffer overflow occurs during the handoff, inappropriate buffer management will cause new problems. For example, an intuitive buffer management policy is proposed in [17], where packets are replaced on a *First-Come-First-Drop* (FCFD) basis when the buffer allocated to the mobile node is full. According to this policy, the packets at the head of the buffer will be dropped first. The idea is that the TCP receiver can receive out-of-order packets right after a handoff, and then generate duplicate ACKs for such packets. Therefore, the TCP sender can resume transmission without waiting any longer. The same mechanism can be found in [14], [18] where three "spurious" duplicate ACKs are generated upon completion of a handoff such that the sender can fast retransmit the packet immediately without waiting for another timeout. Together with a retransmission timeout, this invocation of fast retransmission adversely reduces *ssthresh* to a half of the current window size (= 1 because of the timeout) and ends up with *ssthresh*=2.[2] This means that TCP will restart with a very small window size and enter the congestion avoidance almost right away.

### B. Receipt of Duplicate Packets

The packet buffering-and-forwarding in the MA could introduce some other new problems. If those packets that had already been received by the mobile node before a handoff are

[1] It could be 200 msecs in many Linux machines.

[2] Unless the bug-fix version of TCP [19] is implemented, this will be the case.

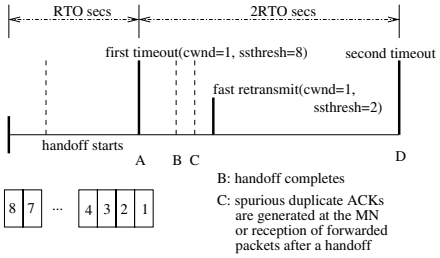


Fig. 2. Concurrence of TCP retransmission timeout and buffer overflow at the old MA

forwarded to the new MA,[3] the mobile node has to generate duplicate ACKs as it has to assume that ACKs before the handoff are lost. This will again trigger the TCP congestion control and result in throughput reduction. One way to avoid this problem is that the mobile node provides the old MA (via the new MA) some information about the received packets. In IPv4, the *16-bit ID* field in the IP header may be used for this purpose [10]. Since there is no such packet identifier in the IPv6 packet header, we will need the link-layer acknowledgement (e.g., in the IEEE 802.11 wireless LAN) between the old MA and the mobile to provide the old MA with the information needed for avoiding the mobile's receipt of duplicate packets after a handoff.

### C. Retransmission Ambiguity

The last problem associated with the packet buffering-and-forwarding is the retransmission ambiguity. Let us consider the previous example and assume packets 1–8 are buffered in the old MA. Packet 1 times out at $t = A$ and the TCP sender retransmits the packet. After all the packets are forwarded to the new MA (and then to the mobile node) at $t = C$, the TCP sender will receive an ACK for packet 1, which is the ACK for the original buffered packet 1, not the retransmitted one. Since the TCP sender is unable to distinguish them, it will just increase the window size by 1, and send packets 2 and 3 (i.e., the slow start after a timeout). This procedure will continue until packet 8 is retransmitted again. At the receiver end (i.e., the mobile node), receiving the duplicate packets 1–8 makes the mobile node generate duplicate ACKs for packet 8. This, in turn, invokes the fast retransmission at the TCP sender side and consequently, reduces *cwnd* and *ssthresh* again. A similar situation has been found in wired networks, where a severely-congested node may cause "spurious" timeouts such that the TCP sender unnecessarily retransmits packets causing the TCP receiver to generate duplicate ACKs. The TCP-Eifel [20] used a timestamp for every timed packet such that the TCP sender can determine, without ambiguity, which of the transmitted segments an ACK belongs to. However, a timestamp will occupy 12 bytes in each TCP packet and requires modification to the TCP sender. In our case, since this retransmission ambiguity occurs during a handoff, we use post-handoff acknowledgement suppression to solve this problem without the need of using a timestamp in each packet.

## III. Enhanced Packet Buffering-and-Forwarding for Smooth Handoff

In order to solve the aforementioned problems caused by the packet buffering-and-forwarding, two enhancements are proposed: one for the MA and the other for the mobile node. In what follows, we will introduce this proposed approach, and present a mathematical model to analyze the TCP throughput under the proposed handoff scheme.

[3]Note that MAs are TCP-unaware and may have no idea which packets have been received by the mobile node.

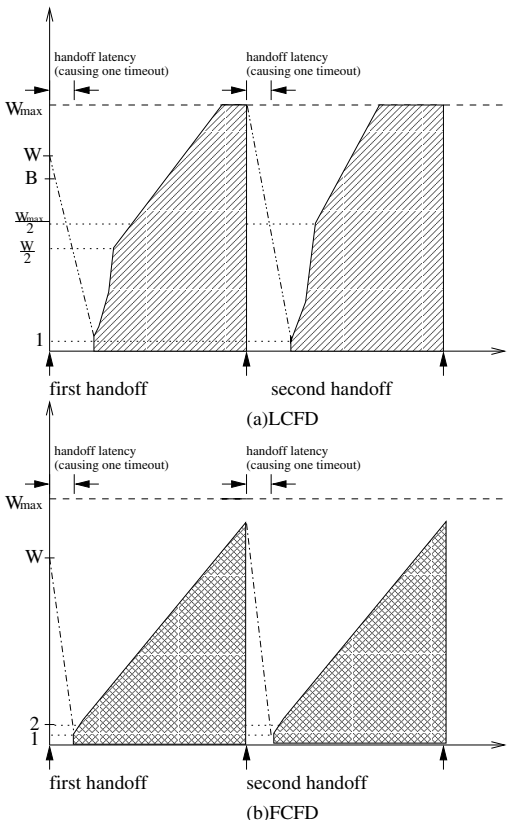


Fig. 3. The evolution of TCP congestion window size for 2 consecutive handoffs: $W = 24$, $B = 20$, and $W_{max} = 32$

### A. Last-Come-First-Drop Policy

The *Last-Come-First-Drop* (LCFD) policy is used for managing the buffered packets in a MA, instead of the intuitive FCFD policy. If no retransmission timeout occurs during a handoff, both policies have similar performance since any dropped packet results in duplicate ACKs. The TCP sender will simply invoke the fast retransmission after the handoff. If there are a handoff-caused retransmission timeout and some packet drops due to buffer overflow, the LCFD policy will outperform the FCFD policy. Let $W_{t_k} = w$ be the congestion window size at the $k$-th handoff. As shown in Figure 2, packet 1 times out at time $A$ and the TCP sender retransmits that packet (now the TCP sender is in the slow start with *cwnd*=1 and *ssthresh* $= \frac{w}{2}$). The handoff is completed at time $B$, and the buffered packets are forwarded to the mobile node via the new MA at time *C*. If packet 1 was dropped in case of buffer overflow under the FCFD policy, duplicate ACKs are generated upon receipt of packets 2–8. The duplicate ACKs will then invoke the fast retransmission of packet 1. This invocation of fast retransmission further reduces *ssthresh* to its minimum value (i.e., $\min(2, \frac{w}{2}) = 2$, where $w = 1$ is the window size after the first timeout). So, the TCP will restart with a small window size and then immediately enter the linear increase region, resulting in severe throughput degradation. Using LCFD, on the other hand, will not generate duplicate ACKs, because the older packets are kept in the buffer such that no "gap" exists. The resulting *ssthresh* is much higher (=$\frac{w}{2}$).

### B. Post-handoff Acknowledgement Suppression

The LCFD policy alone cannot solve the entire problem because of the potential retransmission ambiguity mentioned earlier. In order to improve the TCP performance further, a Post-Handoff Acknowledgement Suppression (PHAS) module is added to the mobile node. Here, we assume that the TCP receiver acknowledges every received packet. After a successful handoff, the PHAS module will delete all ACKs for the forwarded packets, except when it is a duplicate ACK or the ACK for the last forwarded packet. If it is a duplicate ACK, there must be some packet losses during the handoff and the TCP sender should receive such duplicate ACKs as soon as possible for error recovery. If it is the ACK for the last forwarded packet, the TCP sender need not care which TCP packets this received ACK belongs to, because the ACK sequence number must be no less than the sequence number of the retransmitted packet, which is the first packet buffered in the old MA during a handoff. This way, no retransmission ambiguity occurs. It is possible that TCP sender receives more than one ACK that acknowledges the last forwarded packet. This can occur if the TCP sender times out during a handoff. For example, if packets 1–8 are buffered during a handoff but packet 1 times out due to the handoff latency, packet 1 will be retransmitted by the TCP sender. In this case, the TCP receiver will generate two ACKs for packet 8: the first for receiving the forwarded packet 8 and the second for receiving the retransmitted packet 1. But a single duplicate ACK (i.e., the second ACK for packet 8) will not invoke any fast retransmission at the TCP sender side. Instead, this duplicate ACK helps ensure that the TCP sender will receive at least one of these important ACKs (since the ACK acknowledges all buffered packets) in case of transmission loss. In fact, one more duplicate ACK can even be sent by the PHAS to make sure that TCP sender will receive this important ACK. The PHAS can function properly even though some encryption like IPsec [21] is enforced for security reasons. One implementation is to install the PHAS module between the TCP agent and the port demultiplexer in the mobile node so that the PHAS module can update the TCP ACK number easily without compromising the end-to-end security.[4] At the end of packet forwarding, the old MA only needs to notify the mobile node, via the new MA, such that PHAS can know the last forwarded TCP packet.

With these two enhancements, the packet retransmission due to inadequate buffer management or retransmission ambiguity can be eliminated. Figure 3 illustrates the evolution of TCP congestion window under the conventional and the enhanced packet buffering-and-forwarding schemes. Let $B$ be the buffer size in the old MA. We assume $1 \ll B \leq W_{max}$ since a smooth handoff cannot be achieved if $B$ using too small. In Figure 3, we assume that the RTT is about 200 ms, the RTO is 2 secs, and the average sojourn time of a mobile node in each subnet is 10 secs. Further, we assume that the handoff takes 3 secs, triggering a timeout. Under the proposed scheme, since no more than 3 duplicate ACKs are generated by the mobile node upon receiving the buffered packets, the TCP will enter the congestion avoidance about 1 sec after resuming the transmission and will reach $W_{max}$ at $t \cong 8$. In contrast, the TCP under the conventional packet buffering-and-forwarding will enter the congestion avoidance with *ssthresh*=2 after the fast retransmission and fast recovery. The resulting throughput is significantly higher: about 800 packets are sent under our enhanced scheme while only about 680 packets are sent under the conventional one. The difference gets larger in the second handoff because the $cwnd$ value before the second handoff is even larger under our scheme than that under the conventional one. Of course, the improvement depends on the current window size $W$, buffer size $B$, the mobile node's mobility (i.e., the average sojourn time), and also the probability of retransmission timeout due to the handoff. One can expect that, if user mobility is high, or the buffer overflow occurs more frequently, the improvement will be more pronounced.

[4] We will show in Section V how this can be done easily in the ns-2 simulator.

### C. Analytical Model for the TCP Throughput

We now develop a mathematical model for the evolution of TCP NewReno congestion window size in a wireless/mobile network. Based on this model, we can analyze the average throughput of a TCP session when a mobile node undergoes frequent handoffs. We will use this model to show the TCP improvement under the proposed scheme. We will also show how this model can be applied to other handoff schemes such as multicast-based smooth handoff or link-layer-assisted fast handoff.

The basic idea of our model is simple — since the TCP congestion window determines the number of packets that a TCP sender can transmit within a packet round-trip time (RTT), the TCP throughput can be computed based on the congestion window size and the RTT. Let $T$ be the mobile node's sojourn time in a cell and $R$ the packet RTT. Here, the sojourn time is referred to as the time interval starting when the mobile node establishes a connection with the AP until that connection is torn down. Obviously, both $T$ and $R$ are random variables. We can compute the total number of TCP packets transmitted between two consecutive handoffs if we have the probability density functions of $T$ and $R$. For example, the TCP congestion window size is incremented by 1 every RTT if the TCP is in the congestion avoidance. Let $S$ be the normalized sojourn time and $S = \frac{T}{R}$. The number of packets transmitted during the mobile's stay in a cell is $N = \frac{S*(2w+S)}{2}$, assuming that the TCP is in the congestion avoidance after the handoff and $w$ is the contention window size right before the handoff.

In reality, the TCP could be in the slow start phase in case of a retransmission timeout during a handoff or in the fast retransmission phase in case of buffer overflow. Therefore, the number of TCP packets transmitted within the mobile's stay in a cell depends on the probability of TCP retransmission timeouts and the buffer-management policy. The probability of retransmission timeout, $p$, depends on the TCP retransmission timeout (RTO) value, the packet RTT and the handoff latency, and can be obtained as

$$p = P(RTO < T_{lat} + \frac{RTT_{new}}{2} + \frac{RTT_{old}}{2}), \qquad (1)$$

where $T_{lat}$ is the handoff latency and $RTT_{new}$ ($RTT_{old}$) is the packet RTT before (after) the handoff. Figure 4 shows how the TCP congestion window size evolves under different circumstances. For example, the TCP retransmission timeouts occur during the first and third handoffs so the TCP needs to restart with the slow start. For the second handoff, neither a retransmission timeout nor a packet loss occurs. The TCP congestion window size keeps increasing linearly as the TCP is already in the congestion avoidance when a handoff takes place. During the fourth handoff, some packets are dropped due to buffer overflow, so the fast retransmission kicks in. Let $W_k$ be the congestion window size right before the $k$-th handoff. The process $\{W_k : k = 1, 2, \cdots\}$ can be modeled as a discrete-time Markov chain. Given the p.d.f. of $S$, $S(s)$ and $p$, we can compute the transition probability as follows:

*1) $B \geq w$:* Since the buffer is larger than the current congestion window size, no packet in the current congestion window gets dropped during the handoff. Let $W_{k+1}$ be the congestion window size right before the next handoff. For

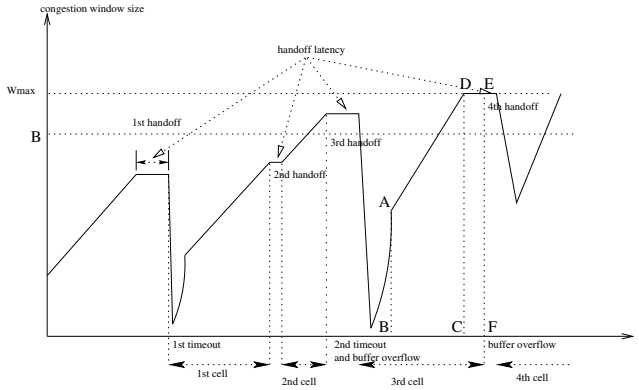


Fig. 4. TCP congestion window size

$w \leq W_{k+1} = j < W_{max}$, the transition probability under the proposed scheme can be computed as

$$P(W_{k+1} = j|W_k = w) = (1-p)\int_{s=j-w}^{j-w+1} S(s)ds + p\int_{s=log_2\frac{w}{2}+j-\frac{w}{2}}^{log_2\frac{w}{2}+j-\frac{w}{2}+1} S(s)ds. \quad (2)$$

The first integral represents the case when no retransmission timeout occurs during a handoff such that the TCP will not be affected by the handoff. Here we assume that the TCP is already in the congestion avoidance at the $k$-th handoff (i.e., $W_k \geq ssthresh$). Unless the mobile node's sojourn time in a cell is small (e.g., only several RTTs long), this assumption is reasonable. If the sojourn time is very small, the mobile node will be handed off immediately after completion of the current handoff, so it is almost impossible to achieve a smooth handoff. The second integral accounts for the case when a timeout occurs (with probability $p$). The TCP will restart with the slow start and enter the congestion avoidance after $log_2\frac{w}{2}$ RTTs. For $\frac{W_k}{2} \leq W_{k+1} < W_k$,

$$P(W_{k+1} = j|W_k = w) = p\int_{s=log_2\frac{w}{2}+j-\frac{w}{2}}^{log_2\frac{w}{2}+j-\frac{w}{2}+1} S(s)ds, \quad (3)$$

because the congestion window size will not shrink without a retransmission timeout. These equations can be applied to both the FCFD and LCFD policies since a buffer-management policy does not have any effect on the TCP if no buffer overflow occurs. For $W_{k+1} = W_{max}$, the transition probability can be obtained from Eqs. (2) and (3) as

$$1 - \sum_{j=\frac{w}{2}}^{W_{max}-1} P(W_{k+1} = j|W_k = w). \quad (4)$$

*2) $B < w$:* Some packets will get dropped according to the buffer-management policy when the buffer is smaller than the current congestion window. Under the proposed LCFD policy, the transition probabilities for $\frac{w}{2} \leq W_{k+1} < W_{max}$ can be computed as

$$P(W_{k+1} = j|W_k = w) = (1-p)\int_{s=T_{d1}+j-w_{s1}}^{T_{d1}+j-w_{s1}+1} S(s)ds + p\int_{s=T_{d2}+j-w_{s2}}^{T_{d2}+j-w_{s2}+1} S(s)ds. \quad (5)$$

The first integral considers the case when no retransmission timeout occurs. Since the packet loss invokes the TCP NewReno's fast retransmission, it takes $T_{d1} = (w - B)$ RTTs for loss recovery before resuming transmission in the congestion avoidance $w_{s1} = \frac{w}{2}$. The second integral represents the case of concurrence of retransmission timeout and buffer overflow. Using the LCFD and PHAS policies makes the TCP reduce *cwnd* and *ssthresh* only once as explained in the previous subsection. In this case, it takes $T_{d2} = log_2\frac{w}{2}$ RTTs before leaving the slow start and entering the congestion avoidance $w_{s2} = \frac{w}{2}$. If the FCFD policy is applied, the transition probabilities for $\frac{w}{2} \leq W_{k+1} < W_{max}$ takes the same form as Eq. (5) except that the TCP will enter the congestion avoidance $w_{s2} = 2$ (*ssthresh*=2) within $T_{d2} = 1$ RTT due to the double reduction of *cwnd* and *ssthresh* as explained earlier. For $2 \leq W_{k+1} < \frac{w}{2}$, the transition probabilities can be obtained by the second term of Eq. (5)

$$P(W_{k+1} = j|W_k = w) = p * \int_{s=T_{d2}+j-w_{s2}}^{T_{d2}+j-w_{s2}+1} S(s)ds, \quad (6)$$

because only the concurrence of buffer overflow and retransmission could make $W_{k+1}$ less than $\frac{w}{2}$. The transition probability, $P(W_{k+1} = W_{max}|W_k = w)$ can be obtained as in Eq. (4).

The next step is to compute the average number of packets transmitted between two consecutive handoffs. Here, we do not include the packets transmitted during the slow start, nor those during the fast recovery. The number of packets transmitted during the fast recovery is usually less than the current congestion window size, and are negligible. For the packets transmitted during the slow start, since the congestion window grows exponentially before the TCP enters the linear-increase region, the TCP stays in the slow start phase only for a short period of time (several RTTs). Moreover, the packets transmitted in a RTT during the slow start is far less than $W_{max}$. Therefore, the total number of packets transmitted during the slow start phase is very small as compared to that in the linear-increase region. For example, let $W_{max} = 36$, $W_k = 32$ and $E[S] = 25$. If a retransmission timeout occurs, the total number of packets transmitted during the slow start is 15 (in 4 RTTs) while the number of packets transmitted thereafter is about 520. That is, the slow start only accounts for less than 3% of the total transmitted packets. Of course, this ratio increases when the mobile node's sojourn time decreases. However, as we pointed out earlier, the smooth handoff is really difficult, if not impossible, to achieve under such frequent handoffs.

Let $N_w$ be the average number of packets transmitted between $t_k$ and $t_{k+1}$, given $W_k = w$. $N_w$ can be obtained by the following equation:

$$\begin{aligned} N_w = p\Bigg\{&\int_{s<T_{th1}} \frac{(\lceil s - T_{d1}\rceil + 2w_{s1})\lceil s - T_{d1}\rceil}{2} S(s)ds \\ &+ \int_{s\geq T_{th1}} \lceil s - T_{th1}\rceil * W_{max} * S(s)ds\Bigg\} \\ + (1-p)\Bigg\{&\int_{s<T_{th2}} \frac{(\lceil s - T_{d2}\rceil + 2w_{s2})\lceil s - T_{d2}\rceil}{2} S(s)ds \\ &+ \int_{s\geq T_{th2}} \lceil s - T_{th2}\rceil * W_{max} * S(s)ds\Bigg\}. \end{aligned} \quad (7)$$

Eq. (7) appears very complicated but it simply computes different areas under the curve of congestion window size shown

in Figure 4. First, let us consider the case when there is no buffer overflow during a handoff (i.e., $w \leq B$). If a timeout occurs (with probability $p$), the number of transmitted packets between handoffs can be calculated by the first term of Eq. (7). In case the normalized sojourn time is less than $T_{th1} = (W_{max} - \frac{w_k}{2}) + log_2 \frac{w}{2}$, the congestion window at $t = t_{k+1}$ cannot reach its maximum value $W_{max}$. The number of transmitted packets is equal to the area of the trapezoid A-B-C-D shown in Figure 4, and can be computed by the first integral with $T_{d1} = log_2 \frac{w}{2}$. Otherwise, the congestion window size will reach $W_{max}$ and more packets can be sent. The number of additional packets being transmitted is equal to the area of the rectangular C-D-E-F shown in Figure 4, and can be computed by the second integral. The total number of packets transmitted is the sum of these two integrals. If no timeout occurs (with probability $1 - p$), the total number of transmitted packets can be obtained by summing the two integrals in the second term of Eq. (7) with $T_{th2} = W_{max} - w$, $w_{s2} = w$, and $T_{d2} = 1$ since the TCP is not affected by the handoff. The same parameters (i.e., $T_{th1}$, $T_{d1}$, etc.) can be used for the case when the FCFD policy is applied since a buffer-management policy does not make any difference in the absence of buffer overflow.

Next, we consider the case when the buffer overflow occurs ($w > B$). Under the proposed LCFD policy, the buffer overflow will not invoke fast retransmission (due to the PHAS) if a retransmission timeout occurs during a handoff. The TCP will restart with the slow start. The number of transmitted packets can be computed as we did for the case of $B > w$. If there is no retransmission timeout during a handoff, the TCP will fast retransmit the dropped packets and the new *ssthresh* will be $\frac{w}{2}$. It takes one more RTT to initiate fast retransmission since only one ACK (for the entire buffered packets) is sent back, and $(w - B)$ RTTs to recover all dropped packets. The number of transmitted packets can be calculated by the second term of Eq. (7) with $T_{th2} = (W_{max} - \frac{w}{2} + w - B + 1)$, $w_{s2} = \frac{w}{2}$, and $T_{d2} = w - B + 1$. Under the FCFD policy, if both the buffer overflow and a retransmission timeout occur during a handoff, the TCP throughput will be reduced because of the double reduction of *cwnd* and *ssthresh*. The number of transmitted packets can be obtained by using the first term of Eq. (7) with $T_{d1} = 1$, $T_{th1} = (W_{max} - 2) + 1$ and $w_{s1} = 2$. Otherwise, the buffer overflow causes the TCP to fast retransmit the dropped packets as that under the LCFD policy, except that the fast retransmission can be detected earlier than the LCFD policy by about one RTT, because $B$ duplicate packets are received immediately after the handoff. In this case, $T_{th2} = W_{max} - \frac{w}{2} + w - B$, $w_{s2} = \frac{w}{2}$ and $T_{d2} = w - B$ in the second term of Eq. (7).

Finally, the average TCP throughput can be obtained by

$$TP = \frac{\sum_{w_k} \pi_{w_k} * N_{w_k}}{\eta},$$

where $\pi_{w_k}$ can be calculated by using Eqs. (2)–(6)

The above analysis considers the case where only the packet buffering-and-forwarding is used. The analytical model, nonetheless, is applicable to the other handoff schemes. In the next subsection, we will show that only minor modification is needed to apply the model for the TCP throughput analysis under different handoff schemes.

### D. Applicability of the Analytical Model

To demonstrate the applicability of our analytical model to other handoff schemes, we selected the fast handoff and multicast-based smooth handoff schemes as examples. These two schemes can, in fact, be regarded as the special cases of our model. If the fast handoff is used, the handoff latency can be reduced but buffer overflow may still be a problem. The transition probabilities can still be calculated by Eq. (2)–(6) with a different probability of TCP transmission timeout according to Eq. (1). Likewise, the average number of packets transmitted between handoffs can be obtained by using Eq. (7). If the multicast is used for achieving a smooth handoff, the problem of buffer management in the old MA is shifted to the neighboring MAs because each neighboring MA will receive a copy of the packet. The TCP throughput can still be obtained by using Eqs. (2)–(7) with or without considering the timeout, depending on the handoff latency.

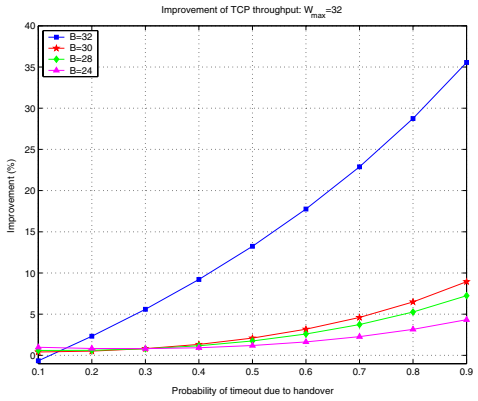


Fig. 5. Comparison of the TCP throughput with and without PHAS

## IV. Numerical Evaluation

We compare the TCP throughput under the conventional packet buffering-and-forwarding scheme, the conventional scheme with the PHAS module, and the scheme with both PHAS and LCFD. We evaluate the impact of three major factors on the TCP throughput: (1) the probability of retransmission timeout during a handoff, (2) the buffer size at each MA, and (3) user mobility. The distribution of mobile node's normalized sojourn time $S$ in a cell is assumed to be uniformly-distributed within $[T_{min}, T_{max}]$ with $T_{max}$ less than 100 (or $ET = 20$ seconds if $E[R] = 200$ msecs) for the following analysis because we are interested in high-mobility cases.

Figure 5 shows the improvement of TCP throughput under the packet buffering-and-forwarding with PHAS enhancement over the conventional scheme, both using the FCFD buffer-management policy. When $B = 32 = W_{max}$, no buffer overflow occurs. If there is a handoff-induced timeout, $cwnd$ will be halved twice under the conventional scheme (i.e., the retransmission ambiguity explained earlier). With the help of PHAS, however, such ambiguity can be resolved and the throughput can be improved significantly, especially when $p$ is large (e.g., 35% when $p = 0.9$). Even though the PHAS may introduce an extra delay of sending back the ACK, it only affects the throughput when $p$ is small and the degradation is almost negligible (e.g., $-0.7\%$ when $p = 0.1$). If $B$ is smaller than $W_{max}$, buffer overflow may occur. The concurrence of buffer overflow and retransmission timeout makes no difference to TCP throughput under these two schemes, because both will invoke the congestion control twice, resulting in the double reduction of $cwnd$ and $ssthresh$. As a result, the improvement is much smaller than that when $B = W_{max}$. This calls for the need of the LCFD buffer-management policy in each MA.

The effect of buffer management on the TCP throughput is plotted in Figure 6. If the buffer size is larger than the maximal window size, the default buffer-management policy (i.e.,

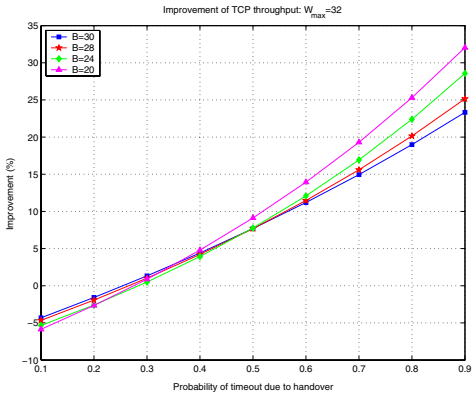


Fig. 6. Comparison of the TCP throughput under LCFD and FCFD policies – (1)

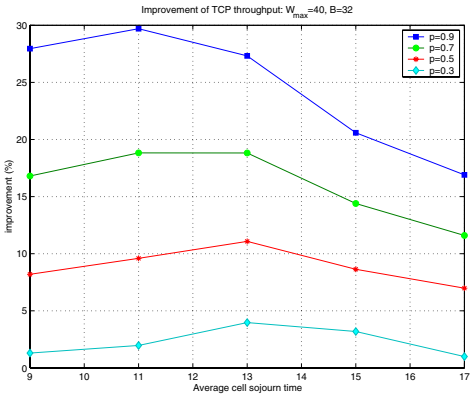


Fig. 8. Comparison of the TCP throughput under LCFD and FCFD policies – (2)

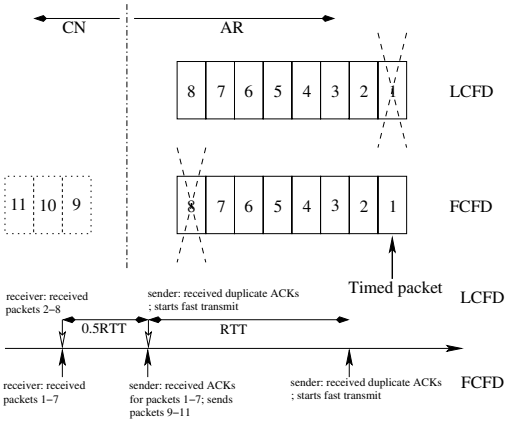


Fig. 7. Fast retransmission under FCFD and LCFD policies when buffer overflow occurs

FCFD) will have the same throughput as the proposed LCFD policy, because there will be no buffer overflow. Therefore, only the case of $B < W_{max}$ is considered. The improvement of throughput under the LCFD policy with PHAS is significant, ranging from 24 to 32% when $B$ is varied from 30 to 24 if $p = 0.9$. Further improvements can be made by decreasing the buffer size, because it is more likely to have concurrent TCP retransmission timeout and buffer overflow for a small buffer size, thus doubly reducing $cwnd$ under the FCFD policy. The difference of improvements by changing the buffer size is within 5% for a large range of $p$. This shows that, when the buffer is a limited resource in the MAs, allocating a smaller buffer space to each mobile node will not cause any significant throughput degradation. The only exception to the improvement by the LCFD policy is when $p$ is small. In this case, timeouts occur rarely, and hence, only buffer overflow will affect the TCP throughput. The FCFD policy enables the TCP sender to detect the packet losses resulting from buffer overflow sooner than the LCFD policy. That is, there is no extra ACK delay under the conventional packet buffering-and-forwarding. The buffer overflow simply results in out-of-order packet delivery, and forces the TCP receiver to generate duplicate ACKs for the first dropped packet. However, using the LCFD policy forces the TCP sender to take one more RTT to trigger the fast-retransmit as shown in the example of Figure 7. Since the difference of this delay is only 1 to 2 RTTs, the resulting throughput under LCFD is slightly lower than that under FCFD by about 5% and it only occurs when $p$ is very small.

This result, together with that in Figure 5, shows that the packet buffering-and-forwarding with both LCFD policy and PHAS enhancement can improve the TCP throughput substantially, regardless if buffer overflow occurs or not. When the buffer is not a limited resource in ARs, the PHAS helps increase the TCP throughput while the LCFD policy kicks in when buffer overflow occurs.

Figure 8 shows the improvement of TCP throughput under the proposed scheme, as compared to the conventional packet buffering-and-forwarding when different user mobility is considered. Here, we convert the normalized cell sojourn time $S$ to the real sojourn time $E[T]$ by using $E[R]$=200 msecs. It shows that the higher the probability of retransmission timeout, the greater the improvement of TCP throughput for a given sojourn time. This is because the double reduction of TCP congestion window size is more likely to occur under the conventional scheme when $p$ is larger. One can also observe that given a probability of retransmission timeout during a handoff, the improvement decreases if a mobile node stays longer in a cell. Given a larger sojourn time, the $cwnd$ under the FCFD policy will eventually reach its maximum value, even though the TCP's new $ssthresh$ is small (i.e., 2) so that it takes more time to reach $W_{max}$. The difference between the numbers of transmitted packets under the FCFD and LCFD policies only makes up a small portion of the total packets transmitted during a mobile node's stay in a cell. Consequently, the improvement due to our enhancements become smaller. On the other hand, if the average sojourn time is small, the improvement may also decreases because the mobile node may be handed off again before its $cwnd$ reaches $W_{max}$. Therefore, the advantage of using the LCFD policy — resuming transmission with larger $ssthresh$ and reaching $W_{max}$ faster than the LCFD policy — becomes less significant, and so does the improvement. In general, the improvement ranges from 10 to 30% in many other real scenarios. For a walking speed around 4-5 km/hour, for example, the average time of crossing a 20 m wireless LAN is 10-15 seconds while for a driving speed around 50 km/hour, it takes 10-15 seconds to traverse a 200-300 m micro cell. In these cases, the improvement could be very significant, depending on the probability of retransmission timeout.

## V. Simulation Results

To confirm the analytical results, we conducted simulation using the ns-2 simulator. A mobile node's handoffs between "cells" are realized by controlling the costs of links between the mobile node (MN) and the MAs: link-a and link-b. The packet routing is determined by the "session" routing protocol provided in ns-2 which updates all nodes' routing tables if a link's

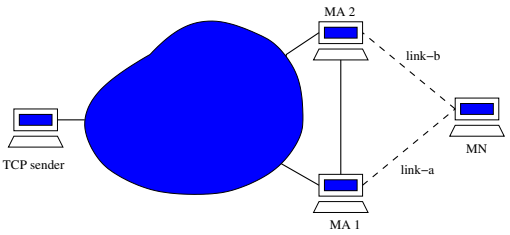


Fig. 9. Simulation setup

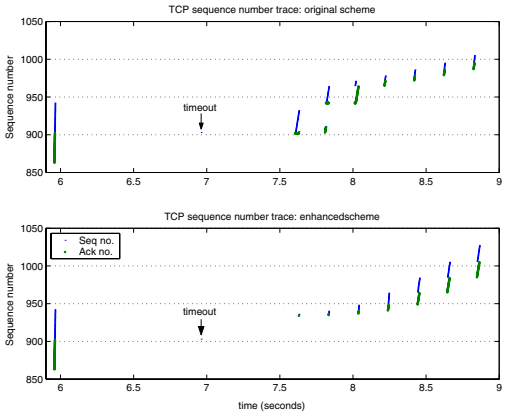


Fig. 10. Trace of TCP sequence numbers during a handoff: $W_{max} = 40$, $B = 32$

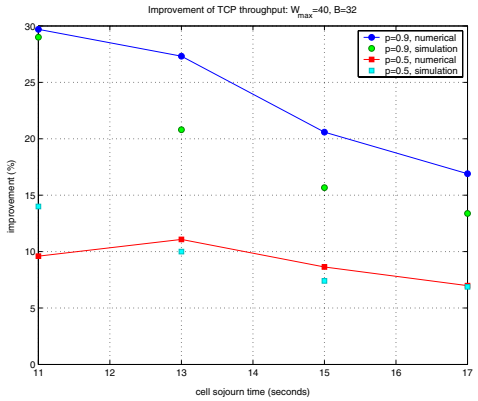


Fig. 11. comparison between numerical and simulation results

cost changes. For example, as shown in Figure 9, let link-a's initial cost be 1 and link-b's cost be 5. Packets will be transmitted to MN via MA1. This way, the MN is said to be in the "first cell." If link-a's cost is changed to $IN\_TRANSIT = 3$, the packets are still sent to MA1, but the MA1 will actually buffer them since the link cost indicates the MN's handoff. If link-a's cost is changed again to 5 while link-b's cost is changed to 1, all new packets will be sent to MA2, and MA1 will also forward the buffered packets to MA2. This completes the handoff process. By repeating this procedure, we can control the MN's handoffs using "link-layer indication" and emulate any desired cell sojourn time and handoff latency. The PHAS module is inserted between the TCP agent and the port demultiplexer of the ns-2 simulator. For the PHAS's operation, only one state variable — the TCP sequence number of the last forwarded packet — is needed. The PHAS can then be used with any existing version of TCP. Throughout our simulation, we use the TCP NewReno for its capability of recovering from multiple packet losses. Finally, the RTT is mainly controlled by the delays of the links between the TCP sender and MAs.

### A. Trace of TCP Sequence Numbers

Figure 10 plots the trace of TCP sequence numbers when a retransmission timeout occurs during a handoff. At $t \approx 7$, the packet is retransmitted. At $t = 7.6$, the TCP sender receives 32 duplicated ACKs for the timed-out packet under the conventional scheme. The packet is fast retransmitted and some other packets are also sent because of the window inflation during the fast recovery. Some of these packets will acknowledge the dropped packets during the handoff, but the others will generate duplicated ACKs for the last forwarded packet. This explains the duplicate acknowledgements around $t \approx 7.7$. After the fast recovery ($t \approx 8$), the TCP enters the congestion-avoidance phase and increases its $cwnd$ linearly (it is not easy to see that in the figure, but it can be inferred by the difference between largest TCP sequence number and ACK numbers).

Compared to the conventional packet buffering-and-forwarding, the trace of our enhanced scheme is much simpler. After a handoff, the TCP sender only receives the ACK for the last forwarded packet. This can be seen by the jump in ACK number between $t = 6.9$ and $t = 7.6$. After that, the TCP enters the slow start at $t = 7.6$ (note the difference between the largest sequence number and ACK number during the time interval [7.6, 8.6], and then enters the congestion avoidance at $t = 8.6$ (note that the $cwnd$ value is around 20, which is the new $ssthresh$). This trace verifies the occurrence of the problem discussed earlier and shows how our scheme can solve the problem. Since the $cwnd$ value under our enhanced scheme is lager than that under the conventional scheme, we will show next that a higher TCP throughput can be achieved.

### B. The Improvement of TCP Throughput

Both the numerical and simulation results are plotted in Figure 11. We only consider the case when mobility is not too high because a smooth handoff is not achievable if the node moves too fast, as we discussed earlier. In general, the numerical results match well the simulation results with the largest error around 7% when the average cell sojourn time is 13 secs. There are several factors contributing to this error. First, we do not consider the packets retransmitted during the fast recovery phase in our model. Even though the number of retransmitted packets is usually less than $cwnd$, it can account for 3–5% of the total packets for the case of $cwnd = 40$ and $\eta = 10$. Therefore, the total number of packets sent under the conventional packet buffering-and-forwarding scheme is actually larger than our estimated value. Second, it is possible that fast retransmit and fast recovery may occur more than once under the conventional scheme (also shown in Figure 10). In our model, we do not take this into account for simplicity of our derivation. Even in the presence of these factors, our model is still accurate, implying its applicability to the enhanced handoff schemes as suggested in Section IV.

### C. The Number of Retransmitted Packets

Finally, we evaluate the amount of resources that can be wasted by an inappropriate handoff scheme. This waste of resources is measured by the total number of retransmitted packets during a handoff. Figure 12 plots the number of retransmitted packets normalized by the maximum congestion window size, $W_{max}$, for both high and medium probabilities of retransmission timeout. It is surprising that the number of retransmitted packets when $p = 0.5$ is a little larger than that when $p = 0.9$ if we use the enhanced packet buffering-and-forwarding. This is because when $p$ is small, a retransmission

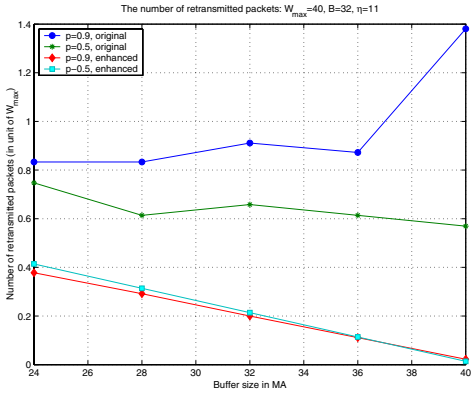


Fig. 12. Number of retransmitted packets during a handoff

timeout is less likely to occur. If a timeout does not occur, the ACK of the last forwarded packet (by the PHAS) will trigger the TCP sender to transmit new packets. These packets, after being received by the receiver, will generate duplicate ACKs for the dropped packets during a handoff and invoke TCP fast retransmit and fast recovery. However, if a timeout does occur, the ACK of the last forwarded packet will not trigger the TCP sender to transmit new packets but the first dropped packet during a handoff. No extra retransmission except for the dropped packets will be needed. But in both cases of $p = 0.5$ and $p = 0.9$, the numbers of retransmitted packets are very small, as compared to those under the conventional scheme.

Under the conventional scheme, the larger the probability of retransmission timeout, the larger the number of retransmitted packets. This is because the double reduction of $cwnd$ is likely to happen when $p$ is large. As a result, the number of retransmitted packets when $p = 0.9$ is larger than that when $p = 0.5$. Moreover, given a probability of retransmission timeout, the number of retransmitted packets should be smaller if MAs are equipped with large buffers. One may notice that there is an obvious exception shown in Figure 12 when $p = 0.9$ and $B = 40$. In this case, no buffer overflow will occur because of $B = W_{max}$. After receiving the forwarded packets, the receiver will generate ACKs for all these packets. Meanwhile, the TCP may already have timed out during a handoff and retransmits the timed-out packets. The TCP sender will also transmit all of the packets already received by the receiver again upon receiving the ACKs after a handoff. These retransmitted packets, upon their reception by the TCP receiver, will generate duplicate ACKs and then make the TCP sender fast retransmit the packets again. Therefore, the total number of retransmitted packets when the buffer size is $W_{max}$, is larger than that if the buffer size is smaller, when $p$ is large. If $p$ is small, the above situation will rarely occur since the retransmission timeout is more unlikely to occur. This result is consistent with the plots in Figure 5 and shows the importance of PHAS in improving the TCP throughput.

In summary, our enhanced packet buffering-and-forwarding not only improves the TCP throughput but also uses the network resources more efficiently (by reducing the number of retransmissions). It is also shown that substantial improvements can be achieved under many different circumstances.

## VI. Conclusion

In this paper, we examined the problem of TCP throughput during a handoff in a wireless/mobile network. We identified the problem of retransmission ambiguity due to timeout during a handoff as well as the double reduction of TCP congestion window size resulting from the concurrence of timeout and buffer overflow under the conventional packet buffering-and-forwarding. We proposed an enhanced scheme for the TCP throughput, using post-handoff acknowledgement suppression with the LCFD buffer management policy. An analytical model for the evolution of TCP congestion window is derived along with the average TCP throughput. By using numerical evaluation, the effects of buffer size, the probability of retransmission timeout resulting from a handoff, and user mobility are studied. The results show that when the probability of timeout is high, the proposed scheme can increase the TCP throughput by 20 to 30%. Since the proposed scheme requires no modification of the TCP and only a different buffer-management policy is needed in MAs, it is easy to implement and can be deployed gradually. The ns-2 simulation confirms our findings and demonstrates the applicability of our model to other schemes. In future, we would like to compare the improvement of the proposed scheme with that of the other schemes, such as the fast handoff or multicast-based scheme, while considering the complexity and extra network resources used.

## References

[1] A. Bakre and B. R. Badrinath, "I-TCP: Indirect TCP for mobile hosts," *Proc. International Conference on Distributed Computing Systems*, 1995, pp. 136–146.
[2] H. Balakrishnan, S. Seshan and R. H. Katz, "Improving reliable transport and handoff performance in cellular wireless networks," *Wireless Networks*, vol. 1, no. 6, 1995, pp. 469–481.
[3] S. Floyd, "TCP and Explicit Congestion Notification," *ACM Computer Communications Reviews*, vol. 24, no. 5, 1994, pp. 10–23.
[4] M. Mathis, *et al.*, "TCP selective acknowledge options," em RFC 2018, 1996.
[5] N.H. Vaidya, M.N. Mehta, C.E. Perkins, and G. Montenegro, "Delayed duplicated acknowledgements: a TCP-unaware approach to improve performance Of TCP over wireless," *Wireless Communications and Mobile Computing*, Vol. 2, no. 1, 2002, pp. 59–70.
[6] J. Kempf, *et al.*, "Bidirectional Edge Tunnel Handoff for IPv6," *IETF Internet Draft*, 2001.
[7] "Low latency handoff in mobile IPv4," *IETF Internet Draft*, 2001.
[8] H.Yokota, A. Idoue, T. Hasegawa, and T. Kato, "Link layer assisted mobile IP fast handoff method over wireless LAN networks," in *ACM Mobicom'02*, pp. 131–139.
[9] J. C. Wu, C. W. Cheng, N. F. Hunag, and G. K. Ma, "Intelligent handoff for mobile wireless Internet," *Mobile Networks and Applications*, vol. 6, no. 1, 2001, pp. 67–79.
[10] C. E. Perkins and K. Y. Wang, "Optimized smooth handoffs in mobile IP," *IEEE ISCC'99* , pp. 340–346.
[11] R. Caceres and V. N. Padmanabhan, "Fast and scalable wireless handoffs in support of mobile Internet audio," *Mobile Networks and Applications*, vol. 3, no. 4, 1998, pp. 351–363.
[12] S. Ohzahata, S. Kimura and Y. Ebihara, "A proposal of seamless handoff method for cellular Internet environments," *IEICE Transactions on communications*, vo1. E84-B, no. 4, Apr. 2001, pp. 752–759.
[13] M. Portoles and *etc*, "IEEE 802.11 Link-Layer Forwarding For Smooth Handoff", *PIMRC'03*.
[14] T. Goff, J. Moronski, D. S. Phatak and V. Gupta, "Freeze-TCP: A true end-to-end TCP enhancement mechanism for mobile environments," *IEEE Infocom'00*, pp. 1537–1545.
[15] C.F. Chan, H.K. Tsang and S. Gupta, "Impacts of handoff on TCP performance in mobile wireless computing", *IEEE ICPWC'97*, pp. 184–188.
[16] C. Blondia, N. Wijngaert, G. Willems, and O. Casals, "Performance analysis of optimizid smooth handoff in MobileIP", *ACM MSWIM'02*, pp. 22–29.
[17] G. Krishnamurthi, *et al.*, "Buffer management for smooth handoffs in IPv6," *IETF Internet Draft*, 2001.
[18] R. Caceres and L. Iftode, "Improving the performance of reliable transport protocols in mobile computing environments," *IEEE Journal on Selected Areas in Communications*, vol. 13, no. 5, 1995, pp. 850–857.
[19] S. Floyd, "TCP and successive fast retransmits", *URL:www.icir.org/floyd/papers/fastretrans.ps*.
[20] R. Ludwig and R. Katz, "The Eifel algorithm: Making TCP robust against spurious retransmissions," *ACM Computer communications Reviews*, vol. 30, no. 1, Jan. 2000, pp. 30–36.
[21] "Security Architecture for IP", *RFC 2401*, 1998.
[22] W. R. Stevens, "TCP/IP Illustrated. Volume 2", Addison Wesley, MA, 1994.