PRISM: Improving the Performance of Inverse-Multiplexed TCP in Wireless Networks

Kyu-Han Kim, *Student Member, IEEE* and Kang G. Shin, *Fellow, IEEE*

***Abstract*— Multi-homed mobile hosts in physical proximity may spontaneously team up to form a community and run high-bandwidth applications by pooling their low wireless wide-area network (WWAN) bandwidths together for communication with a remote application server. Utilizing their high-bandwidth wireless local-area network (WLAN), the thus-teamed mobile hosts can aggregate and distribute the application content among themselves. This paper first justifies the need for such a mobile collaborative community (MC^2), or a community, to improve user-perceived network bandwidth and utilization. Then, existing one-to-one communication protocols like TCP are shown to suffer significant performance degradation due to frequent out-of-order packet deliveries. To address this TCP problem, we propose a proxy-based inverse multiplexer, called *PRISM*, that enables TCP to efficiently utilize the community members' WWAN connections while avoiding the performance degradation. PRISM runs at the proxy's network layer as a routing component and stripes each TCP flow over multiple WWAN links by exploiting the transport-layer feedback information. Moreover, it masks a variety of adverse effects specific to each WWAN link via an intelligent ACK-control mechanism. Finally, PRISM enables TCP to respond correctly to dynamically-changing network states through a sender-side enhancement of congestion control. PRISM has been evaluated with experimentation on a testbed as well *ns-2*-based simulation. Our experimental evaluation has shown PRISM to improve TCP's performance by up to 310% even with two collaborative mobile hosts. Our in-depth simulation study has also shown that PRISM delivers a near-optimal aggregated bandwidth in the community, and improves network utilization significantly.**

***Index Terms*— Mobile collaborative community, multi-homing, bandwidth aggregation, TCP, out-of-order packet delivery**

I. Introduction

As wireless networks become omnipresent, mobile users are gaining access to the Internet via a variety of wireless networks. To keep pace with this trend, a mobile host is becoming multi-homed with multiple wireless network interfaces (e.g., GPRS, IEEE 802.11x, and Bluetooth). Based on the network and technology diversity, several approaches have been proposed to enhance network availability, focusing on concurrent (or alternative) use of multiple wireless technologies available on a host [18], a mobile user [12], or a designated mobile center [26]. Even though these approaches improve the network availability and performance, they only consider the use of a *single* individual multi-homed entity, realizing only partial benefits from the technology diversity.

It is important to note that collaboration among multi-homed mobile hosts significantly improves both user-perceived bandwidth and overall wireless network utilization. Mobile hosts in close proximity can spontaneously form a community, connected via a high-speed WLAN interface and sharing their WWAN link bandwidths with other members in the community. Each member within the same community receives a subset of contents from an Internet server and shares the contents with the other members (*content sharing*). On the other hand, one member uses the other members' bandwidths when it needs more bandwidth than its own WWAN link for applications like music/video file downloading and Hi-Definition TV live cast (*bandwidth sharing*).

We therefore advocate formation of a mobile collaborative community (MC^2) that is a user-initiated network model in order to make the best of WWAN diversity. Currently, a mobile host is forced to use a single WWAN link at a time and thus, suffers several limitations in capacity, coverage, and hardware. By contrast, in the mobile community, mobile users initiate new virtual WWANs that overcome such limitations by sharing their WWAN interfaces. Moreover, by adopting an inverse multiplexer [14], the community effectively aggregates and uses its members' WWAN bandwidths by inverse-multiplexing traffic over the shared links.

However, existing transport protocols, such as the Transmission Control Protocol (TCP), are optimized only for a single link and waste the available bandwidth of multiple links in the mobile community. Frequent out-of-order packet deliveries due to the heterogeneity of multiple WWAN links generate duplicate acknowledgments (ACKs), which, in turn, cause the TCP sender to over-reduce his congestion window size. There are several transport-layer approaches to aggregation of the bandwidths of multiple wireless links, such as those in [17], [18], [20]. However, their basic design considers aggregation of the interfaces of only a *single* host or user, and requires support from the network layer to route traffic to/from a group of multi-homed mobile hosts. Moreover, the development and deployment of a whole new transport protocol requires significant efforts on both content servers and mobile clients, and their operation incurs a high computational overhead to resource-constrained mobile hosts.

To solve these problems, we propose a proxy-based inverse multiplexer, called *PRISM*, that enables each TCP connection to utilize the entire community's aggregate bandwidth. As a protocol complementary to TCP, PRISM consists of (i) an inverse multiplexer (PRISM-IMUX) at a proxy that forwards TCP's data traffic through different WWANs and controls duplicate ACK traffic, and (ii) a new congestion control mechanism (TCP-PRISM) at the sender that expedites loss recovery. PRISM-IMUX stripes TCP traffic intelligently over multiple

The authors are with the Real-Time Computing Laboratory, Department of Electrical Engineering and Computer Science, The University of Michigan, Ann Arbor, MI 48109-2122, E-mail:{kyuhkim, kgshin}@eecs.umich.edu

WWAN links using up-to-date link state information such as link weight, the number of in-flight packets, and a round trip time on each WWAN link. Also, it masks the effects of out-of-order delivery by identifying spurious duplicate ACKs and re-sequencing them so that the TCP sender receives correctly-sequenced ACKs.

The second component in PRISM, TCP-PRISM, is the sender-side congestion control mechanism that reduces the loss-recovery time and accurately adjusts the congestion window size of TCP by using the loss information provided by PRISM-IMUX. It immediately dis-ambiguates real packet losses from out-of-order deliveries through negative loss information, and reduces the loss recovery time. Its proportional adjustment strategy of the congestion window size further improves link utilization by minimizing the effects of partial network congestion on un-congested links.

We evaluate the performance of PRISM using both experimentation and *ns-2*-based simulation. PRISM is implemented as a Linux-based loadable module and extensively evaluated on a testbed. Our experimental evaluation shows PRISM to improve TCP's performance by 208% to 310% even with two collaborative mobile hosts with heterogeneous link delays, loss rates and bandwidths. Moreover, our simulation study shows that PRISM effectively reduces the need for reordering packets and delivers a near-optimal aggregated bandwidth in the community that consists of heterogeneous mobile hosts.

The rest of this paper is organized as follows. Section II presents the motivation and the contributions of this work, and Section III provides an overview of the PRISM architecture. Sections IV–VI give detailed accounts of PRISM. Section VII describes our implementation and experimentation experiences. Section VIII evaluates the performance of PRISM using *ns-2*-based simulation. Related work is discussed in Section IX. Finally, Section X discusses a few remaining issues with PRISM and concludes the paper.

II. Motivation

We first present motivations for a mobile collaborative community. Then, we discuss basic functions for the community to work. Finally, we identify the problem of TCP in the community and introduce our approach to the problem.

A. Why a Mobile Community?

Wireless network services are becoming available anywhere and anytime. 2.5G and 3G wide-area cellular networks, such as GPRS, UMTS, and CDMA, are being deployed for more bandwidth and wider coverage. Moreover, WLANs (e.g., IEEE 802.11x) can provide high-speed wireless network services in small areas. At present, different wireless Internet Service Providers (ISPs) are co-located with different network technologies or frequency channels, and end-users are equipped with various wireless (e.g., WWAN, WLAN, and Bluetooth) interfaces and can select the best interface/channel available at a given place/time.

Although there exist various choices (i.e., different ISPs, technologies, and channels) in the current wireless network environment, they are not utilized efficiently due to the current ISP-centric service model. That is, most mobile users should use the same network, technology, or a single frequency channel for their connectivity. As a result, they suffer from various service limitations as described below.

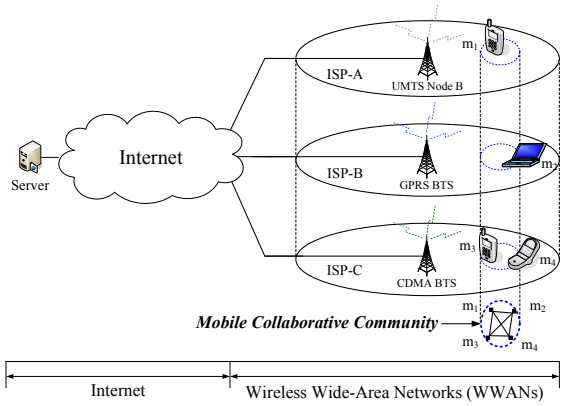


Fig. 1. *Target environment.* The environment includes various WWAN network services available and multi-homed mobile hosts equipped with both WWAN and WLAN interfaces. Mobile hosts in a WLAN range form a mobile community and collaborate to simultaneously use multiple WWANs.

- L.1 *Capacity limitation:* Mobile users may experience a low data rate from its own ISP while other ISP networks in the same area are idle or under-utilized.
- L.2 *Coverage limitation:* A user may find no service nearby from his own ISP while the other ISPs' services are available.
- L.3 *Hardware limitation:* A user cannot access a new service through his own interfaces while other nearby users can access the service by their new interfaces.

Let us consider the following scenario to have a feel for the above limitations. Sam is waiting at an airport for his flight, and wants to download his favorite movies to watch during his long flight. First, he tries to use his own WWAN interface, but finds that it will take longer than his waiting time for the flight (capacity limitation). Next, he decides to use his WLAN interface. However, the nearest WLAN hot-spot is too far away for him to return in time for the flight (coverage limitation). Finally, he finds another access network with high capacity, but his device does not support the access network's technology (hardware limitation). Therefore, Sam will not be able to download and watch the movies. Instead, Sam searches other nearby mobile users who are willing to share their interfaces for certain "rewards." He finds several mobile hosts whose interfaces have capacity, use different frequency channels, or support a high-rate wireless technology like IEEE 802.16. With the help of other mobile hosts, Sam can download movies in time, and enjoy them during his flight.

To realize a scenario like this, we construct a user-initiated collaborative wireless network model, called a *mobile collaborative community (MC^2)*. As shown in Figure 1, the community is composed of multi-homed mobile hosts in physical proximity. Community members are connected to the Internet via different WWAN ISPs (e.g., m_1, m_2) or different channels[1]

[1]We assume that the community is formed in such a way that its members have mutually exclusive frequency channels to make bandwidth aggregation practical if they subscribe to the same ISP.

of the same ISP (e.g., m_3, m_4), and locally communicate with each other via WLAN interfaces in ad-hoc mode.

B. How Does a Mobile Community Work?

For an MC^2 to work, it requires three basic functions: collaboration, multiplexing, and indirection.

1) Collaboration Among Mobile Hosts: The mobile community requires users to collaborate by sharing/pooling their communication channels. However, what are the incentives for users to collaborate? When only one host or a small set of members want to receive the contents at others' expenses, will the other members be willing to contribute their bandwidths to enable the small set of members to achieve statistical multiplexing gains?

A somewhat related debate is underway with regard to "forwarding incentives" in ad hoc network routing [11], [27], [34]. In ad hoc networks, the communication between endpoints outside of the radio transmission range relies on intermediate nodes on the path to forward packets for them. Some researchers suggest use of credit-based, or reputation-based, schemes to stimulate cooperation [11], [22]. Game-theoretic arguments have been used to show that collaboration on packet forwarding among all participating nodes maximizes network throughput [5].

Forwarding in ad hoc networks, however, is somewhat different from the collaboration we consider here. In ad hoc networks, nodes rely on each other to communicate amongst themselves. In a mobile community, nodes rely on each other, not for basic connectivity but for performance improvements. As we will see in Section III, a node completely controls access to its shared communication resources, and revokes access if its communication needs are not met by the community. Ultimately, it is the ability to opt-in to achieve better performance and the ability to opt-out when necessary, making link sharing a viable option. Nonetheless, communities are more likely to be formed within domains where a pre-existing trust (or cost-sharing) relationship exists. For example, an individual with multiple devices (e.g., cell phone, PDA, laptop) interconnected with a personal area network can benefit from resource sharing, as in the case of teams of people working together.

2) Multiplexing: Given shared links, how can the mobile community aggregate link bandwidths for a higher throughput? An inverse-multiplexer is a popular approach that aggregates individual links to form a virtual high-rate link [14]. For example, an inverse multiplexer stripes the traffic from a server over multiple wireless links of the community members, each of which then forwards the traffic to the receiver. Finally, the forwarded packets are merged and assembled in the receiver at the aggregate rate.

An important issue is then where to put the inverse multiplexer. The inverse multiplexer can be placed at (1) a performance-enhancing proxy (PEP [10]) by a network access provider, a wireless telecommunication service provider, or a content distribution network operator for downstream communications, and (2) one of community members for upstream communications.

Within a proxy or a host, the multiplexer can be placed at the network layer as a routing component with an efficient traffic filtering function as in the Network Address Translation (NAT) service. Or, the inverse multiplexer might run as an application like in an overlay network. However, multiplexing inherently requires responsive network state information, and additional packet-processing overheads at the application layer limit the performance of the inverse multiplexer [18].

3) Indirection: Traffic from an inverse multiplexer to community members is tunneled via Generic Routing Encapsulation (GRE) [15]. The inverse multiplexer encapsulates the traffic via GRE and routes it to the community members' WWANs. Upon its reception, each member de-capsulates the tunneled traffic, and forwards it to a destination via WLAN. Since the destination is oblivious to which member forwarded the data packets, no additional data reassembly functionality is required at the receiver. Furthermore, because GRE tunneling is supported by most operating systems (e.g., Linux, FreeBSD, the Windows), no system modification of mobile hosts is required.

C. Challenges in MC^2's Use of TCP

Our primary contribution in this paper is to enable *one-to-many-to-one* communication for a TCP connection to achieve high-speed Internet access in an MC^2. While traditional one-to-one communication of TCP limits its bandwidth to a single link's capacity, in an MC^2, we enable a TCP connection to achieve the members' aggregate bandwidth by inverse-multiplexing its packets over all available members' WWAN links.

In this communication model, however, we encounter several challenges. First, *scheduling* traffic over wireless links requires exact link-state information, such as data rate and delay, which varies with time and is usually expensive to obtain in mobile environments. Second, since WWAN links suffer from high and variable round trip times (RTTs), burstiness and outages, a large number of *out-of-order* packet deliveries—which occur frequently and generate spurious duplicate ACKs—degrade TCP's end-to-end performance significantly. Finally, TCP's *congestion control* mechanism does not fully utilize multiple links' bandwidths because it interprets a packet loss as the overall links' congestion, making over-reduction of its congestion window size. Also, frequent spurious duplicate ACKs with positive ACKs cause the sender to delay loss detection/recovery.

D. Improving TCP Performance in an MC^2

To overcome the above challenges, we propose a proxy-based inverse multiplexer, called *PRISM*, that effectively aggregates members' WWAN links bandwidths for a TCP connection. Specifically, we

C.1 devise an adaptive scheduling mechanism that efficiently stripes traffic without any link-state measurement overheads while maintaining full links utilization (Section IV);

C.2 construct an ACK-control mechanism that effectively masks the effects of out-of-order delivery without sacrificing end-to-end performance (Section V); and

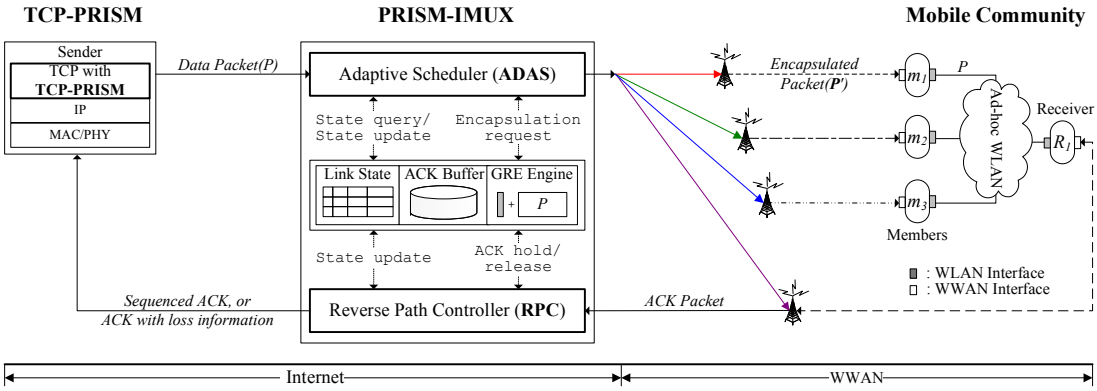


Fig. 2. *PRISM architecture*. PRISM consists of an inverse multiplexer at the proxy (PRISM-IMUX) and a sender-side congestion control mechanism (TCP-PRISM). PRISM-IMUX schedules data packets over multiple WWAN links of the community members (ADAS), and masks adverse effects from out-of-order packet deliveries (RPC). TCP-PRISM helps TCP accurately react to partial link congestion.

C.3 propose a new congestion-control mechanism that (i) is a sender-side optimization technique and (ii) improves link utilization by expediting loss recovery (Section VI).

The rest of this paper provides a detailed account of PRISM. The following assumptions are made for the basic design of PRISM and mobile community: (1) each mobile host has multiple (especially WWAN and WLAN) interfaces that can be used simultaneously for a single application connection; (2) a mobile host uses PRISM mainly for downstream communications[2] such as music/video file downloads; (3) a mobile community is formed via an application-layer daemon as in [30]; (4) an inverse multiplexer is located at PEPs of each 3G's access network; (5) for each PRISM flow, a PEP owned by the ISP that a receiver of the flow belongs to is chosen for inverse-multiplexing traffic, and it handles prior agreements on how to share other 3G's access networks (e.g., access control and pricing), as in the roaming of ATM networks; (6) inverse-multiplexed traffic from the proxy to mobile hosts may traverse different ISPs' infrastructure networks, but it is mainly dictated by bottleneck WWAN links as assumed in [13], [29]; and (7) GRE is enabled at each mobile host as default; and (8) both the sender and the receiver support TCP-SACK.

III. The PRISM Architecture

Figure 2 provides an architectural overview of PRISM and its operational environment. PRISM consists of a network-layer inverse multiplexer (PRISM-IMUX) at the proxy and a network-assisted congestion-control mechanism (TCP-PRISM) at the sender side. PRISM interacts with a mobile community of multiple multi-homed mobile hosts through multiple WWANs.

A. PRISM-IMUX

PRISM-IMUX is the routing component in a proxy that handles both the forward (data) and backward (ACKs) traffic of a TCP connection using up-to-date wireless links-state information. As shown in Figure 2, PRISM-IMUX captures the data traffic from a sender in the proxy's network layer,[3] and determines the best WWAN link for the next hop via the Adaptive Scheduler (ADAS). It also captures and controls ACK packets to mask the adverse effects of striping over multiple WWAN links via the Reverse Path Controller (RPC). Finally, PRISM-IMUX maintains a WWAN-links-state table, is equipped with a buffer for temporarily storing ACKs that need to be re-sequenced, and supports GRE for indirection. We will detail ADAS in Section IV, and RPC in Section V.

B. TCP-PRISM

TCP-PRISM is a new sender-side congestion-control mechanism that works with PRISM-IMUX to expedite loss recovery and to improve network utilization. TCP-PRISM reduces the loss recovery time via the *negative* ACK information shipped by RPC at the proxy to detect a packet loss. Also, it adjusts the congestion window size based only on the congested link bandwidth, thus preventing waste of uncongested links' bandwidth. This will be detailed in Section VI.

C. Mobile Community

A mobile community is formed voluntarily and incrementally. When a new mobile node wants to join an existing community, it first searches for communities nearby using the Service Location Protocol [30]. After determining the community of most interest to itself, the mobile joins the community and works as either a relay node or a receiver. The node receives packets from PRISM-IMUX via its WWAN link, and forwards packets to the receiver, through its WLAN interface in ad-hoc mode. Or, the node receives packets via multiple community members' WWAN links, and sends ACKs to the sender through one of the WWAN links.

IV. Scheduling Wireless Links:ADAS

A. Overview

Scheduling TCP packets over heterogeneous wireless links requires exact link-state information for a receiver to achieve

[2]This is for easy exposition of the PRISM architecture. We will extend this for upstream communications in Section X.

[3]Note that a magnified proxy in the figure shows only the PEP of a receiver's ISP (i.e., different PEPs are used by different ISPs), and direct links between a proxy and base stations are *logical* links that consist of multiple hops via different ISPs' access networks.

the optimal aggregate bandwidth, and obtaining the information is expensive, especially in mobile environments, due to the fluctuating traffic rate and wireless links' dynamics. As shown in Figure 3, the typical TCP traffic rate fluctuates as a result of its congestion and flow control. Similarly, the output rate varies due to the heterogeneity of wireless links and/or the processing power of each member device in a mobile community. Although it is possible to measure a channel's condition and report it to the proxy, frequent changes in the channel condition will incur significant report-processing overhead and transmission-power consumption to resource-limited mobile hosts.

ADAS is a new packet-scheduling algorithm that is adaptive to dynamic input/output rates, and that incurs the least cost in obtaining link-state information. ADAS maintains up-to-date link-state information—which is obtained by RPC (to be discussed in Section V) without incurring any reporting overhead to mobile nodes—and adaptively schedules packets over the best available links using the state information. Also, it uses packets' expected arrival times over each link not only to reduce out-of-order packet deliveries, but also to increase the end-to-end throughput. Finally, ADAS adaptively reacts to the congestion of a link via a TCP's AIMD-like traffic control mechanism.

B. Algorithm

ADAS consists of three scheduling rules and a dynamic link-weight adjustment mechanism. Algorithm 1 describes ADAS's scheduling rules. *Rule.1* is to give retransmissions priority in scheduling packets on a WWAN link. Under *Rule.2*, ADAS chooses the link with the most available bandwidth by using a normalized NIP (see below). Under *Rule.3*, if there are more than two links with the same NIP, then ADAS picks the link that has the smallest expected arrival time ($hRTT$).

1) Normalized Number of In-flight Packets (NIP): NIP enables ADAS to utilize multiple links fairly so as to maximize aggregate bandwidth. NIP_i is derived from the Weighted Round Robin (WRR) scheduling for its fairness. WRR divides the time into *rounds*, in each of which packets are assigned to a link based on its proportional bandwidth (or weight), and thus, all links are utilized fairly. Likewise, ADAS uses the link weight for link utilization, thus achieving long-term fairness as WRR does.

However, ADAS uses a different definition for fair link utilization: *while WRR keeps track of how many packets have been scheduled so far on each outgoing link, ADAS considers how many packets are currently in-flight on the link given its link-weight*. Because existing static scheduling algorithms like WRR assume accurate link-state information, using only the link weight in scheduling packets over wireless links cannot capture and adapt to network dynamics. Hence, in addition to the link weight, ADAS exploits the actual NIP, which automatically reflects unexpected delay or loss of a link in order to determine the best available link. Therefore, we define and use $\text{NIP} = \left\lfloor \frac{N_i}{W_i} \right\rfloor$, where N_i is the NIP over ℓ_i, and W_i (link weight) is the ratio of the link bandwidth to the least common denominator/factor of all links' bandwidths. N_i can be derived from scheduled packets' information and ACK-control information without incurring any additional cost, and W_i is given when member i joins the mobile community.

Algorithm 1 ADAS_Scheduling ($Packet$)

```
if Packet is a retransmission then
    // Rule.1: retransmit lost packets over a fast link
    l_next is the link with minimum hRTT in 𝕊
else
    𝕊_next = {l_i: links with minimum NIP_i from 𝕊}
    if |𝕊_next| == 1 then
        // Rule.2: use NIP_i
        l_next is the link (in 𝕊) with minimum NIP_i
    else
        // Rule.3: use an expected time of arrival and NIP_i
        l_next is the link with minimum hRTT in 𝕊_next
    end if
end if
Update N_i, NIP_i for l_next, and 𝕊
return l_next
```

$hRTT_i$	RTT from a proxy to a receiver over WWAN_i
N_i	Number of in-flight packets (NIP) on ℓ_i
l_{next}	WWAN link which a packet is scheduled to
NIP_i	Normalized N_i with respect to link weight
$\mathbb{S}$	A set of community members' WWAN links
$\mathbb{S}_{next}$	A set of links with the minimum NIP

Let's consider Case A in Figure 3 to see the effectiveness of using NIP. The ratio of the weight of link ℓ_1 to that of link ℓ_2 is assumed to be 1:2. ADAS schedules the third packet (p_3) on ℓ_2 because when p_3 arrives at the proxy, ADAS knows from an ACK packet (a_2) that p_2 has left ℓ_2, so ℓ_2 still has more available bandwidth than ℓ_1. In case of WRR, it assigns p_3 to ℓ_1 because the quantum of ℓ_2 has already exhausted by p_1 and p_2, wasting available bandwidth of ℓ_2.

2) Expected Arrival Time ($hRTT$): ADAS uses expected arrival time (RTT/2 or $hRTT$) along with NIP to further improve the overall link utilization and minimize the need for packet reordering. When more than one link ($\mathbb{S}$) have the same lowest NIP, ADAS selects the link that has the smallest expected arrival time in that subset of links (Rule.3). Due to a WWAN link's varying transmission delay or forwarding nodes' random processing delay, links with a similar utilization might experience different short-term rates or delay fluctuations which might not be immediately reflected into NIP. Using $hRTT$ ensures that ADAS transmits packets on the fastest link in a greedy fashion, thus not only increasing the overall short-term link utilization, but also reducing out-of-order packet deliveries at the receiver.

One might question why we do not use only $hRTT$ or only NIP without flooring (i.e., $\frac{N_i}{W_i}$). Using only $hRTT$ increases short-term link utilization and avoids fluctuating links. However, its long-term performance is limited by the capacity of the link that has the smallest RTT but low bandwidth because of frequent packet drops at a bottleneck queue. On the other hand, using only NIP contributes to adaptation to long-term fair link utilization, but it is not responsive to random short-term delays, reducing average link utilization. We will compare the performance of these two variants with that of ADAS in Section VIII-C.2.

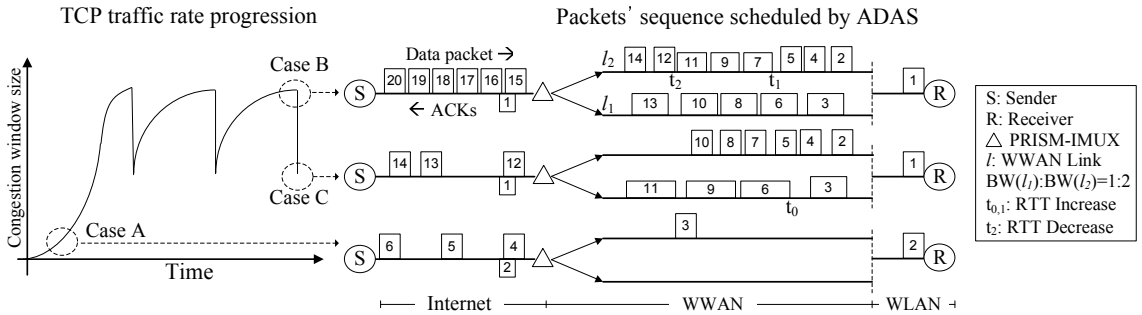


Fig. 3. *Three ADAS scheduling snapshots for different input/output rates.* The left graph shows the fluctuation of input rate from a TCP sender. The right three wide figures are snapshots of ADAS scheduling for different input/output rates. Case A shows Rule.2 (based on the number of in-flight packets), Case B shows Rule.3 (based on RTT as well as NIP$_i$), and Case C shows dynamic weight adjustments.

Let's consider Case B in Figure 3 to illustrate the effectiveness of using $hRTT$ along with NIP. Until p_7, ADAS has packets scheduled on each link with the same sequence as WRR does. However, at t_1, $hRTT$ of ℓ_2 increases and when p_8 is to be scheduled, the expected arrival time of p_8 via ℓ_2 becomes longer than that via ℓ_1. Besides, since the NIP values of both links are same, ADAS schedules p_8 on ℓ_1. If the packet is scheduled on ℓ_2 as WRR does, then p_8 might arrive later than p_9, and ℓ_1 could waste its bandwidth until the transmission of p_9 begins.

3) Dynamic Link-Weight Adjustment: ADAS adapts to each link's congestion without separate links-state probing or congestion-notification messages from the network by dynamically adjusting the congested link's weight. ADAS uses the loss information obtained by RPC (to be explained in the next section), and adjusts the link weight to approximate its instantaneous bandwidth by adopting the TCP's Additive Increase and Multiplicative Decrease (AIMD) strategy [16]. If the link experiences congestion, ADAS cuts the congested link's weight by half. Subsequently, the link's NIP$_i$ becomes larger, and no new packets are assigned to the link until it recovers from the congestion. This link weight is increased additively each time an ACK arrives on that link, without exceeding the original weight. Specifically,

(i) When a packet loss on link i, ℓ_i, is detected by RPC, the link's original $weight_i$ is stored as $\mathbb{W}_i$. Then, $weight_i$ is reduced to $\frac{weight_i}{2}$ (or $weight_i = 1$ if $\frac{weight_i}{2} < 1$);

(ii) When a new data packet traversed ℓ_i is acknowledged by the receiver, $weight_i$ is incremented by $\frac{1}{weight_i}$ each time corresponding ACK arrives, without exceeding $\mathbb{W}_i$; and

(iii) Whenever $weight_i$ is updated, the NIP of ℓ_i is recalculated and used to immediately reflect link-state information into the packet scheduling.

Case C in Figure 3 depicts ADAS's reaction to both delay fluctuations and packet losses. When p_6 is scheduled at t_0, ℓ_1 experiences increased $hRTT$. However, ADAS schedules p_6 on ℓ_1 based on NIP to maintain maximum network utilization even though it might cause packet reordering. On the other hand, right before scheduling p_{11}, ADAS detects and identifies the loss of p_0 on ℓ_2. It adaptively reduces the ℓ_2's weight using the link-weight adjustment algorithm, and assigns p_{11} to ℓ_1 based on the newly-computed NIP.

C. Complexity

The main computational complexity of ADAS comes from the sorting of links to find the best link. Since ADAS uses an ordered list, it requires $O(\log n)$ time complexity where n is the number of available links. Usually, n is less than 10, so its overhead is not significant. ADAS requires constant space complexity. ADAS maintains a link-state table as shown in Figure 2. It independently stores per-link information which includes only four variables (i.e., NIP$_i$, $hRTT_i$, W_i and N_i).

V. Handling Spurious Duplicate ACKs:RPC

A. Overview

Even though ADAS tries to minimize the need for packet reordering, data packets are sometimes scheduled out-of-sequence intentionally to fully utilize networks (e.g., Case C in Figure 3). Moreover, due to the delay fluctuations resulting from the aggressive local retransmission mechanism of 3G networks or a community member's processing delay, there could be unexpected out-of-order packets. In both cases, a receiver blindly generates duplicate ACKs, which we call 'spurious' duplicate ACKs, as a false sign of link congestion, and these ACKs, unless handled properly, significantly degrade TCP performance.

The Reverse Path Controller (RPC) is an intelligent ACK-control mechanism that hides the adverse effects of out-of-order packet deliveries to the receiver. RPC exploits TCP's control information which is carried by ACKs, to determine the meanings of duplicate ACKs and correct them, if necessary. Moreover, along with a scheduling history, RPC also infers the link condition such as its packet loss, delay, and rate. Finally, because RPC maintains each link's state information (including loss and instantaneous capacity), it provides such information to the sender's congestion-control mechanism so as to prevent one pipe from stalling other uncongested pipes, thus enhancing network utilization.

B. Algorithm

RPC consists of three ACK-controlling mechanisms: ACK identification, ACK re-sequencing, and loss detection. RPC first determines the meaning of an arrived ACK. Then, it decides whether this ACK needs to be re-sequenced or not. Finally, it differentiates duplicate ACKs caused by real packet

losses from spurious duplicate ACKs, and detects any packet loss.

1) ACK Identification: In order to determine the meaning of ACKs, this mechanism identifies the sequence number of a data packet that actually arrives at the receiver and causes an ACK to be generated. Assuming that the receiver supports the TCP-SACK mechanism, RPC traces the meta-state of the receiver buffer through SACK blocks and a cumulative ACK number, and finds the latest updated sequence number of the receiver buffer via the newly-arrived ACK. Because TCP-SACK conveys information of up to three data blocks when there are holes in the receiver buffer, and its first block[4] contains the sequence number of the recently-arrived data packet [23], RPC can infer the state of the receiver's buffer as follows.

A.1 *SACK block matching:* If an ACK delivers SACK information, RPC simply matches the SACK block(s) with the meta-state buffer and finds sequence number(s) that is newly covered by this SACK block.

A.2 *Cumulative ACK number scanning:* If an ACK sequence number is greater than the meta-buffer's cumulative sequence number, RPC scans a region between the two numbers, and finds the sequence number(s) that has not been covered before.

Figure 4 shows a series of snapshots that describe the two schemes of identifying a sequence number. For example, snapshots I, II, and IV show the SACK block matching scheme. Snapshots III and V illustrate how cumulative ACK numbers are scanned. Each snapshot contains the circular buffer representing the meta-state of the receiver buffer.

2) ACK Re-sequencing: After identifying the meaning of ACKs, RPC determines whether to release this ACK to the sender, or to hold it for re-sequencing as follows. If the identified sequence number proceeds towards a new unACKed sequence number, RPC starts releasing ACKs-on-hold including the one just arrived (e.g., snapshots III and V).

If arrived ACK packets are duplicates, then RPC re-sequences them in two different ways. First, if there is not any congested link, then RPC holds the ACK packet in the slot of the wrapped-around sequence number in the circular ACK buffer. Since RPC knows the meaning of each ACK, it corrects the cumulative ACK sequence number with the identified number of the ACK packet and stores it in the buffer (e.g., snapshots I, II, and IV).

Second, if there exists congested link(s), RPC releases ACKs-on-hold in their original form because duplicate ACKs have really resulted from packet loss(es), and because released duplicate ACKs can help the sender calculate the number of packets that have left the network.

3) Loss Detection: The remaining questions on the ACK re-sequencing mechanism are how to detect packet losses from congestion, and how to differentiate out-of-order packet arrivals from real packet losses. Assuming that packets scheduled on a link are delivered to the receiver in order, RPC detects packet losses if there are holes that are not sequentially acknowledged in a list of scheduled packets on the link. Snapshot VI shows an example of loss detection of RPC. Since packets 26, 28 were sent back-to-back via link 1, RPC

[4]It could be the second block when a DSACK option is used [33].

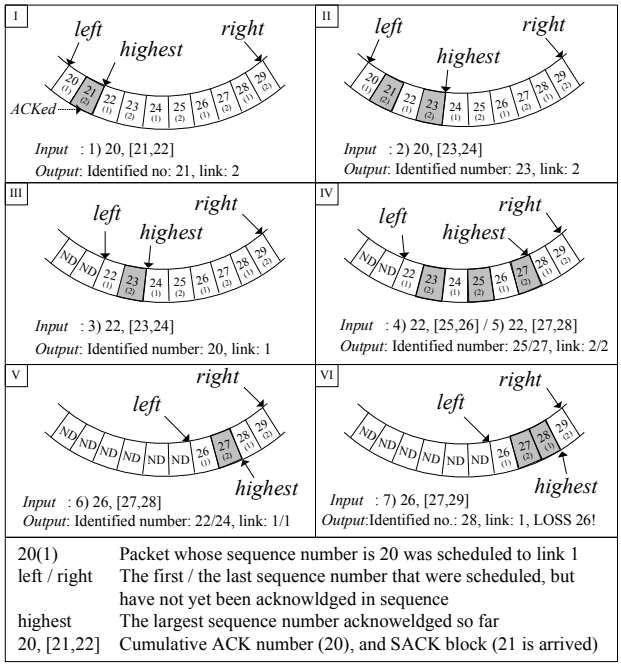


Fig. 4. *Snapshots for ACK identification and loss detection mechanism in RPC.* RPC determines the meaning of each ACK through SACK blocks (Snapshot I, II, and IV) or cumulative ACK numbers (Snapshot III, V). Also, it detects packet losses using the identified sequence numbers and scheduling history (Snapshot VI).

determines, from the arrival of ACK 28, that packet 26 is lost. This is different from the loss detection mechanism of TCP whose duplicate ACKs' threshold is 3. However, a more sensitive reaction on each link is desirable since it helps all connections avoid disrupting one another. Moreover, any threshold can be set based on the network characteristics. Packet losses can also be attributed to wire-line congestion. RPC detects such wire-line packet losses from an ACK packet whose corresponding sequence number in the circular ACK buffer is neither scheduled nor ACKed (called *No Data*).

As shown in the above example, the loss-detection mechanism and its accuracy depend on the information of ACK identification and scheduling history. Given the accurately-identified ACK sequence numbers, finding a hole in each link just requires one more comparison and is a clear indication of packet losses. On the other hand, the loss-detection mechanism may raise false alarms if there are more than three consecutive ACK losses, because ACK identification cannot fully construct the meta-state due to the limited number of SACK blocks carried by ACKs. However, three consecutive losses of small-sized ACKs rarely happen—throughout all of our evaluation, we have not observed it—and by using cumulative ACK number scanning, ACK identification can also complete the meta-state of the receiver's buffer and can quickly recover from a false alarm.

C. Complexity

RPC's complexity strongly depends on the number of duplicate ACKs. When there are no duplicate ACKs, RPC does not incur any overhead except for updating link-state

variables (NIP$_i$, N_i). However, if there are duplicate ACKs resulting from either out-of-order delivery or packet losses, then RPC needs time to figure out the ACK's sequence number and space to re-sequence ACKs. First, for computational complexity, ACK identification mainly consumes computation resources, and its sequence comparison incurs the computational overhead which is linear in the number of duplicate ACKs. However, this overhead can be minimized using an optimization technique, such as a hash function and a bit-operation, whose time complexity is constant.

In the worst case of space complexity, RPC may have to store all ACKs of a flow for ACK re-sequencing. Since the number of ACKs is limited by the number of outstanding packets in the network, $\frac{BW \times RTT}{MSS} \times S_{ACK}$ is the maximum required ACK re-sequencing buffer size. For example, assuming that aggregated bandwidth (BW) and average RTT are 5 Mbps and 120 ms, respectively, and a maximum segment size (MSS) is 1.5 KB and the size of ACK (S_{ACK}) is 60 bytes, the maximum space requirement is 3 KB. Note that even though the space complexity is linear in the number of flows, it can support a large number of PRISM flows (hence scalable). For instance, only a 60 MB footprint is necessary to support 20,000 simultaneous PRISM flows which deliver a 100 Gbps bandwidth.

VI. Expediting Packet Loss Recovery:TCP-PRISM

A. Overview

Along with ACK re-sequencing and loss detection, fast recovery from packet loss(es) and appropriate congestion control are critical to the overall TCP performance. Although many end-host congestion-control mechanisms, such as Reno, New-Reno and SACK, have been proposed for a *single* path congestion control, they are not optimized for *multiple* paths due mainly to the following two reasons. First, TCP's positive ACK mechanism (e.g., SACK block) consumes more time to detect/recover packet loss or out-of-order delivery from multiple and heterogeneous paths, resulting in frequent timeouts. Second, they over-reduce the window size upon congestion of one of multiple paths, reducing the overall link utilization.

In addition to the end-host approaches, there are other approaches to control congestion/channel-related loss at a base station or a proxy via local retransmission [8] or split-connection [7]. Local retransmission using another copy of each packet at the base station can handle channel-related losses, but it cannot effectively handle partial link congestion due to over-reduction of the window size at the end-host. Besides, it needs to store a copy of each data packet, causing a scalability problem. On the other hand, the split-connection scheme can handle both congestion and packet loss, but it violates TCP's end-to-end semantics [18] and requires an entirely new transport protocol at both the proxy and mobile hosts to support the use of multiple links.

To effectively recover from packet losses while fully utilizing multiple links, PRISM takes an end-host congestion-control approach and removes the limitations of existing solutions with the following mechanisms. The first mechanism provides exact loss/congestion information in a negative form to the TCP sender. The second is a sender-side congestion control mechanism (TCP-PRISM), which understands negative ACK information from networks and expedites loss recovery upon congestion of a link in one of multiple paths. Finally, TCP-PRISM is a simplified version of TCP-SACK, so it is easy to implement and deploy with other congestion-control mechanisms.

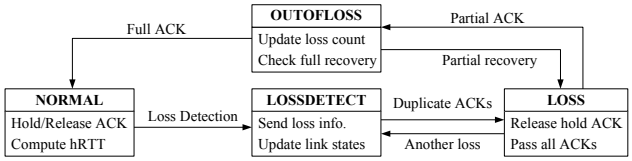


Fig. 5. *State machine of RPC.* Boxes with capital letters indicate states of RPC, and boxes with small letters list operations in each state.

B. Algorithm

This algorithm is invoked by RPC and the sender-side TCP when there is a packet loss(es). On detection of any packet loss, RPC ships loss information on ACKs. Using this delivered information at the sender, its congestion-control mechanism quickly reacts to packet losses.

1) Delivery of Loss Information: Figure 5 shows the state machine of RPC that describes loss information delivery in each state. In NORMAL and OUTOFLOSS states, RPC only updates state variables as described in Section V. In LOSSDETECT state, RPC sends the loss information to the sender, and switches to LOSS state. RPC in LOSS state releases all duplicate ACKs until all losses are recovered.

RPC provides loss information to the sender that includes: (i) which data packet is lost, (ii) which channel is congested to adjust the congestion window size, and (iii) how many packets have left the network. Once a packet loss is detected, RPC sends the lost packet's sequence number to the sender in the form of negative ACK. In addition, RPC ships the congested link's bandwidth information that is computed as the proportion (p) of congested link's bandwidth over total bandwidth (i.e., $p = 1 - \frac{B_i}{2\sum_{j=1}^{n} B_j}$, where i is the congested channel ID, B_j the bandwidth of channel j, and n the total number of active channels). Finally, after sending the loss information, RPC begins releasing ACKs-on-hold, if any, so that the sender can calculate the exact in-flight packet number, inflate the congestion window size, and send more data packets via other uncongested links.

2) Congestion-Control Mechanism: TCP-PRISM makes two major enhancements of existing congestion-control mechanisms. First, it reduces the fast retransmit time given partial link's congestion by using the loss information delivered from the proxy. TCP-PRISM just extracts lost packets' sequence numbers from the information and retransmits the corresponding data packets immediately. It does not wait for more duplicate ACKs, nor does retransmit all packets which are ambiguously believed to have been lost.

Second, it makes fast recovery accurately react to congestion, and thus, improves network utilization. TCP-PRISM re-

duces the congestion window size only by the proportion (p)—we call this adjustment *Additive Increase and Proportional Decrease* (AIPD). This adjusted window size allows the sender to transmit more data via uncongested links. If there are other congested links, TCP-PRISM performs the same procedure as the above. Other than the above two enhancements, TCP-PRISM works exactly the same way as the standard TCP-SACK.

C. Complexity

The complexity of TCP-PRISM is lower than that of the standard TCP-SACK thanks to the loss information available from RPC. TCP-SACK's scoreboard mechanism maintains positive ACK information based on a SACK block(s) from a receiver and then identifies lost segments by comparing it with a sender's outstanding packet list. Thus, TCP-SACK has to keep ACK information in the scoreboard until it receives full ACKs. Even though the size of the scoreboard is bounded by the number of in-flight packets, in the worst case, it has to repeatedly search its scoreboard to find lost segments upon receiving a SACK block(s). By contrast, TCP-PRISM simplifies the scoreboard mechanism by using negative loss information shipped by RPC, thus saving spatial (storage of positive ACKs) and computational (comparison of SACK blocks) costs in TCP-SACK.

VII. Implementation

We have implemented, and experimented with, PRISM. This section first presents our implementation details of each PRISM component. Then, it describes our testbed setup and presents the experimental results.

A. Implementation Details

1) PRISM-IMUX: PRISM-IMUX is implemented as a loadable kernel module in the network layer using Netfilter [1]. Netfilter provides a hook for packet filtering at the network layer, and hence allows users to dynamically register or unregister packet filters. That is, PRISM-IMUX is implemented as a filter with a back-end agent which includes ADAS and RPC.

Within the network layer, there are three places to register the PRISM-IMUX filter: at entrance, NF_IP_PRE_ROUTING; in the middle, NF_IP_LOCAL_OUT; and at exit, NF_IP_POST_ROUTING. The filter is registered at the layer's exit because such a placement minimizes the number of functions that PRISM-IMUX should incorporate, and also avoids the need for system modification. When PRISM-IMUX transmits multiple packets from its buffer, it can make a direct call to an interface function of the link layer, so it need not go through all the remaining network-layer functions.

Finally, the filter agent may sometimes need to store packets, and thus, stop the remaining packet-processing chain in the network layer. There are two options (NF_DROP and NF_STOLEN) from Netfilter to silently store a packet, and PRISM-IMUX uses NF_STOLEN as it does not incur any overhead, such as the buffer copying required in NF_DROP.

2) TCP-PRISM: We implemented TCP-PRISM in a Linux kernel-2.4's TCP protocol stack, and installed it in a server. As stated in Section VI, we have implemented a simplified scorebard mechanism as an extension of TCP-SACK, while preserving the original TCP-SACK. Specifically, TCP-SACK maintains sack-tag information in the scoreboard, which is initially cleared, and becomes "SACKED" when the corresponding sack information arrives. Based on the information of an un-sacked packet, TCP-SACK decides on packet loss. We modified this sack-tag mechanism so that the exact loss information provided by PRISM-IMUX is reflected immediately into the scoreboard, bypassing mechanisms for the scoreboard maintenance and the packet loss decision. TCP-PRISM is enabled through a socket option, and the normal TCP-SACK is performed if this option is disabled.

B. Testbed Setup

To evaluate our PRISM implementation, we have built a testbed that is composed of an Internet infrastructure, and a mobile community as shown in Figure 6. For the Internet infrastructure, we use one server (Pentium-IV 1.64 GHz CPU with 512 MB memory), one proxy, one WWAN emulator (both are a Pentium-III 865 MHz CPU with 256 MB memory), and one Ethernet switch. TCP-PRISM and PRISM-IMUX are installed on the server and the proxy, respectively. The emulator has NISTnet [2] to emulate WWAN networks of each member. The Ethernet switch works as a WWAN access point and splits traffic from the emulator to each community member. The server, the proxy, the emulator, and the switch are connected in a row via 100 Mbps Ethernet cables between adjacent components.

For the mobile community, we use three Dell latitude laptops (Pentium-III 865 MHz CPU with 256 MB memory) which have both built-in Ethernet interfaces (Realtek) and IEEE 802.11b (Orinoco) interfaces. Each Ethernet interface is connected to Ethernet switch's 100 Mbps cables and is used as a WWAN link. An ad-hoc mode WLAN interface is used for communication within the community.

All machines in the testbed use Redhat 9.0, and an ftp application between the server and a receiver is used to measure end-to-end throughput by transferring a 14 MB file.

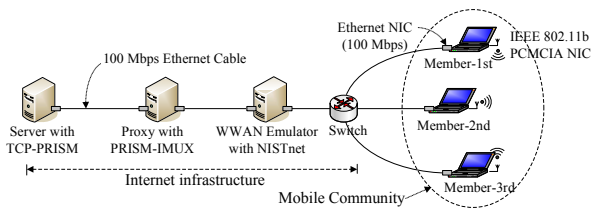
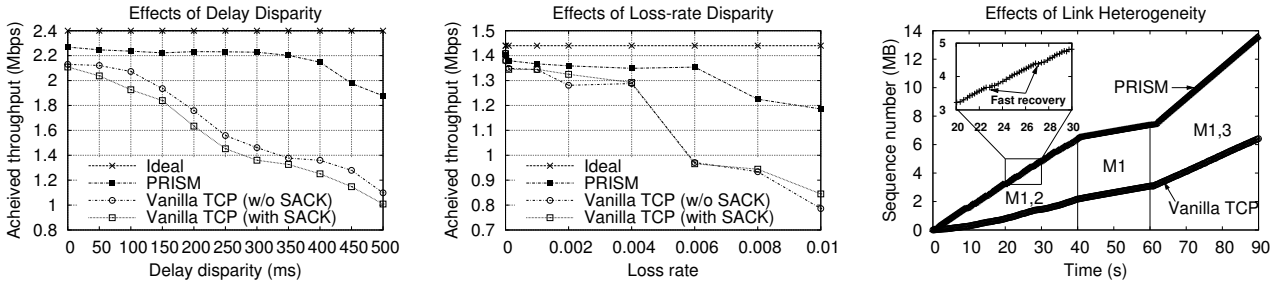


Fig. 6. *Testbed setup.* The testbed is composed of Internet infrastructure (a server, a proxy, a WWAN emulator and a switch) and mobile community (three laptops). Note that even though we show direct links between components in wired netwtorks to simplify our testbed, the links could consist of multi-hop links.



(a) Effects of delay disparity (b) Effects of loss-rate disparity (c) Effects of link heterogeneity

Fig. 7. *Experimental results.* We ran PRISM on our testbed and compared its performance with that of vanilla-TCP. Figure 7(a) shows PRISM's robustness to delay disparity among the links aggregated. Figure 7(b) also shows PRISM's robustness to loss disparity. Finally, Figure 7(c) shows PRISM's effectiveness in aggregating heterogeneous members' WWAN links.

C. Experimental Results

We present three experimental results, demonstrating PRISM's TCP performance improvement in the presence of delay disparity, loss-rate disparity, and links heterogeneity.

1) Effects of delay disparity: We evaluated the robustness of PRISM's performance to WWAN links' delay disparities. We use two community members that have different WWAN link bandwidths (1800, 600 Kbps) but initially have the same link delay, 500ms (average delay from the UMTS trace with the packet size of 1.4 KB) [5]. While increasing the delay of the second member's link up to 1000 ms in increments of 50 ms, we measure the end-to-end throughput. For a better comparison, we also run vanilla-TCP with and without SACK.[6]

PRISM effectively masks the effects of WWAN links' delay disparities by using the re-sequencing mechanism and provides an aggregated bandwidth. As shown in Figure 7(a), PRISM achieves a throughput equal to 95% of the total aggregate links capacity when the delay disparity is less than 400 ms. Beyond that point, it shows a little degradation because of deep-buffering for increasing duplicate ACKs. Vanilla-TCP suffers a significant performance degradation due to spurious duplicate ACKs. Furthermore, vanilla-TCP with SACK performs worse than that without SACK because the detailed SACK information delivered to the sender causes a significant number of false retransmissions.

2) Effects of loss-rate disparity: We measured the robustness of PRISM's performance to the WWAN links' loss-rate disparities. In a community of two members (with WWAN link bandwidths of 1080 and 360 Kbps), we fix the link delay of both members at 300 ms (average delay from the UMTS trace with 1KB packet size) and measure the end-to-end throughput while varying the loss rate from 0.001% to 1% of the second member (1% is a typical maximum loss rate of WWAN links [18]).

PRISM's fast-recovery mechanism indeed expedites loss recovery and increases link utilization even at a high loss rate. As shown in Figure 7(b), PRISM provides throughput equal to 94% of the aggregated links' capacity when loss rates are less than 0.8%. At the point of 0.8% or higher, PRISM's throughput decreases because the achievable link throughput also degrades due to frequent packet losses. Vanilla-TCP, however, experiences a severer performance degradation. Even though it shows a relatively good performance (i.e., 90%) at a low loss rate, vanilla-TCP's performance degrades as the loss rate increases due to the long loss-recovery time for one congested link and blockage of the uncongested link.

3) Effects of link heterogeneity: We evaluated PRISM's performance gain even in the case of heterogeneous community members. We construct a mobile community that consists of three members, all having different WWAN link characteristics (i.e., bandwidth, delay, and loss rate) as follows: member 1 (M1) has a reliable but slow link (360 Kbps, 300 ms, 0%); member 2 (M2) has a fast but unreliable link (1080 Kbps, 100 ms, 0.6%); and member 3 (M3) has a faster link (1800 Kbps, 100 ms, 0%) than others, but its bandwidth difference from M1's is large (5 times). Initially, M1 and M2 collaborate until 40 seconds, but encounter different delays and loss rates. Then, M2 leaves the community (at 40s). At 60s, M3 joins the community and collaborates with M1, but they have a large bandwidth disparity.

PRISM achieves the aggregated bandwidth of all WWAN links even in case of heterogeneous link characteristics. Figure 7(c) shows the sequence number progression of a sender's transport layer for both PRISM and vanilla-TCP. As shown in the figure, PRISM can achieve 310% throughput compared to vanilla-TCP in the presence of both loss-rate and delay disparities (from 0s to 40s) thanks to its fast loss-recovery mechanism (see the magnified graph in Figure 7(c)). Furthermore, PRISM yields a 208% performance improvement over vanilla-TCP in case of a large bandwidth disparity (ranging from 60s to 90s) using its effective scheduling mechanism and ACK re-sequencing mechanism.

VIII. Performance Evaluation

We also evaluated PRISM via in-depth simulation for diverse environments. We first describe our simulation models and then evaluate PRISM with respect to bandwidth aggregation, packet reordering, network utilization, and fairness.

[5]We use the on-line source of [26].

[6]Note that vanilla-TCP refers to TCP implementation in Linux kernel-2.4.20. Vanilla-TCP with (without) SACK means the use of the original TCP implementation with a SACK (NewReno) option.

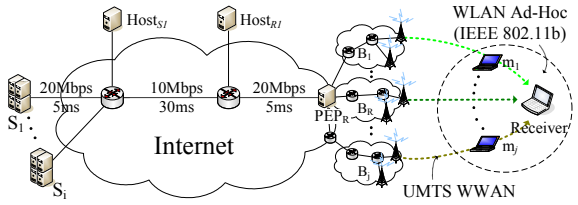


Fig. 8. *Simulation network topology.* S_i (server), $Host_{S,R}$ (background traffic generator), a PEP of a receiver's ISP, B_j (base station), m_k (community members), and R (receiver)

A. Simulation Models

We use *ns-2* [3] for our simulation study. The network topology in Figure 8 is used for all simulation runs and consists of Internet, WWAN and WLAN networks. First, the Internet is composed of fixed servers (sender S_i), Internet routers, a performance enhancing proxy (PEP), and other hosts ($Host_{S/R}$) for generating background traffic. The bandwidth between each host and its edge router is 20 Mbps, and the bandwidth between Internet routers is 10 Mbps. Next, for WWANs, we use the Universal Mobile Telecommunication System (UMTS) *ns-2* extension [6]. B_i is a WWAN access network that is connected to a couple of WWAN access points. Then, for WLANs, we use the IEEE 802.11b implementation in *ns-2*, and add the NOAH [32] routing protocol to simulate peer-to-peer communications among community members. Finally, for each community member m_j, including a receiver, we use an extended *ns-2* mobile node with multiple wireless interfaces.

Given the above topology, we implemented TCP-PRISM at the sender's transport layer and PRISM-IMUX at the proxy's network layer as explained in Section III. For indirection between the proxy and each mobile node, we used encapsulation/decapsulation modules in *ns-2*. Finally, we implemented a simple service location protocol [30] for bootstrapping the mobile community and for initiating the PRISM flow at mobile nodes.

We used the following parameter settings throughout the simulation. First, we use TCP-SACK as a receiver's transport protocol as it is. Second, we use UDP flows with a packet size of 1000 bytes as background traffic. Finally, we run an FTP application(s) for 150–500 seconds and average the results of 10 runs unless specified otherwise.

B. Achieving Bandwidth Aggregation

We measured PRISM's aggregated bandwidth gain while increasing the number of WWAN links. The bandwidth of each link is randomly chosen from the range [400Kbps, 2.4Mbps]. We first run an FTP session between a server (S_1) and a receiver (R_1) for 300 seconds under PRISM, and then run the same experiment without the proxy ('No Proxy'). For a better comparison, we also run the same experiment under a weighted-round-robin (WRR) striping agent without the ACK-control mechanism.

PRISM achieves bandwidth aggregation (i.e., summing the bandwidths of all links), and its performance scales well with various community sizes. Figure 9 plots PRISM's bandwidth aggregation gain, and confirms the performance gain and scalability with up to five community members[7]. By contrast, using the WRR striping agent, TCP performance degrades to the one that is worse than a single community member's throughput due to frequent out-of-order packet deliveries. Note that the "Ideal" case is defined as the sum of vanilla-TCP throughputs achieved via all WWAN links.

C. Minimizing Need for Packet Reordering

ADAS minimizes out-of-order packet deliveries by using link utilization and RTT. We show its effectiveness in the presence of bandwidth disparity and rate/delay fluctuations.

1) Bandwidth disparity: We evaluated how much ADAS reduces the need for packet reordering in the presence of disparity of WWAN links' bandwidths. We use three community members whose links bandwidth difference (say, $d\%$) increases from 0 to 70%, and measure the achieved aggregate throughput. We initialize the WWAN bandwidth of all members to 1.4 Mbps and then increase one member's bandwidth by $d\%$ of 1.4 Mbps and decrease the bandwidth of one of the remaining members by the same percentage. We disable RPC to isolate the performance benefit of ADAS, and run PRISM with other existing scheduling algorithms as well as ADAS.

ADAS reduces the number of packets delivered out-of-order by sensing bandwidth disparity, and achieves an up-to-280% improvement of link utilization over other scheduling algorithms (especially, *random*). Figure 10 shows the performance gain by reducing the need for packet reordering under various scheduling algorithms. The x axis represents the bandwidth disparity, and the y axis is the achieved throughput which is normalized by the ideal total bandwidth of WWAN links. Although the maximum ratio is below a half of the ideal bandwidth due to the absence of RPC, the figure shows that the throughput under ADAS improves as the bandwidth disparity increases by selectively using high-bandwidth links, i.e., ADAS reduces the number of out-of-order deliveries.

On the other hand, since other scheduling algorithms, such as WRR, RR and Random, blindly assign packets to all available links without any regard to their bandwidth disparity, their performance is degraded by a significant number of out-of-order deliveries that result from the use of low-bandwidth links.

2) Rate/delay fluctuations in WWAN links: We also evaluated ADAS's adaptivity to rate/delay fluctuations by examining the end-to-end throughput given dynamically-changing background traffic. For a system with three community members (whose WWAN bandwidths are 600, 900 and 1200 Kbps, respectively), we run one PRISM flow and two ON/OFF background traffic flows (one via the first member's WWAN link with 400 Kbps, and the other via the third member's WWAN link with 800 Kbps). We use a burst-time of ON/OFF traffic as the parameter of rate/delay fluctuations with a Pareto distribution and a one-second idle-time. In this experiment, we enable RPC to show the overall performance improvement.

[7]Note that we limit the maximum community size to 5 since the IEEE 802.11b provides up to 6 Mbps in terms of end-to-end throughput.

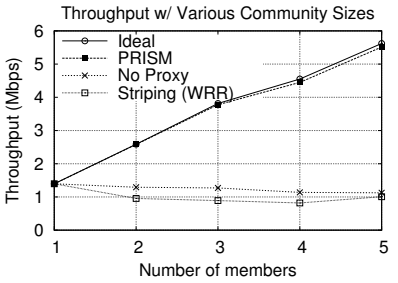


Fig. 9. Bandwidth aggregation in the mobile community with various sizes of community members.

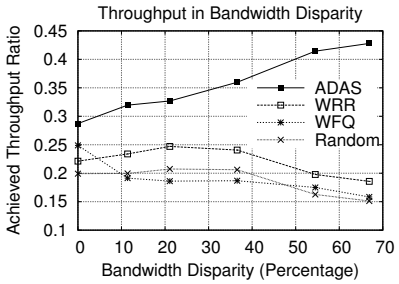


Fig. 10. Performance comparison in bandwidth disparity for different scheduling schemes without RPC.

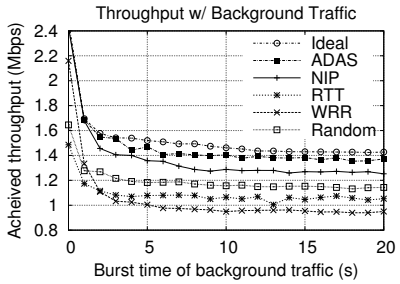


Fig. 11. PRISM's performance comparison under rate/delay fluctuations for different scheduling schemes.

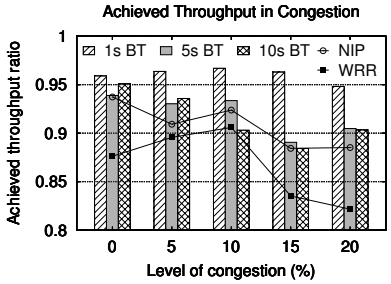


Fig. 12. Effects of congestion

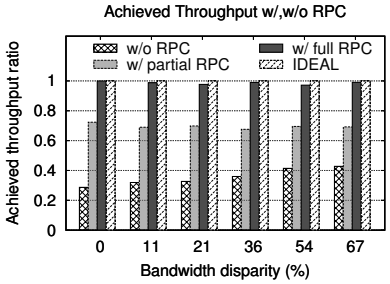


Fig. 13. Performance gain by RPC

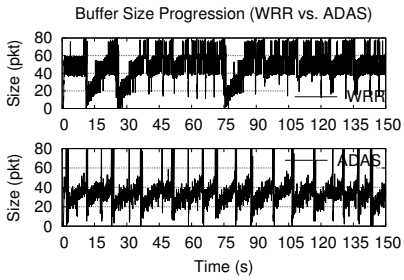


Fig. 14. Resequencing buffer size

ADAS adapts to the rate/delay fluctuations of WWAN links and reduces the need for packet reordering. As we will see in Section VIII-D.2, the thus-reduced number of out-of-order packet deliveries improves the end-to-end throughput, so we measured the throughput achieved by several scheduling algorithms while increasing the rate/delay fluctuations. As shown in Figure 11, ADAS outperforms the other scheduling algorithms by 12–47% and achieves 97.1% of the ideal performance in the presence of maximum background traffic. The "ideal" case is the sum of vanilla-TCP's throughputs achieved via all WWAN links with the same background traffic.

On the other hand, WRR performs worse than random scheduling in the presence of large fluctuations. $hRTT$-only scheduling performs worse (1.48 Mbps) than the others because the fast (i.e., small RTT) but low-bandwidth link limits the overall performance by dropping many packets. The NIP-only scheduling algorithm performs similarly to ADAS in a stable link state. However, as the rate/delay fluctuates more, the NIP-only scheduling becomes less responsive to short-term fluctuations than ADAS which adapts itself to the fluctuations by using RTT, thus achieving only 88% of ADAS's throughput.

3) Congestion over a path from a proxy to base stations: Although we have focused on characteristics of WWAN links (assumed to be a bottleneck), we also evaluate the effectiveness of PRISM over intermittent congestion in a path from a proxy to base stations. We use the same community settings as the previous experiment, and run UDP flows with a Pareto distribution to generate congestion over paths from a proxy to every base station as shown in Figure 8. As we increase the level of congestion by adjusting a burst time and a peak rate of the distribution, we measure the throughput of a PRISM flow. We also measure throughputs of PRISM with NIP and WRR for the same scenario.

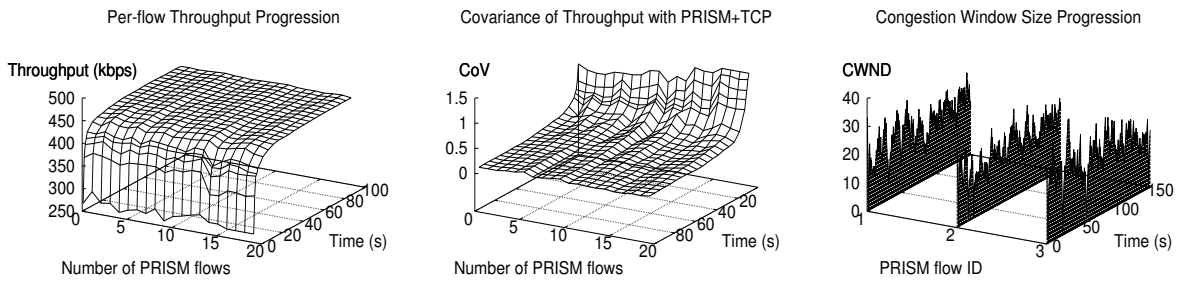
Figure 12 plots ratios of PRISM's achieved throughput to the "ideal" throughput, and demonstrates its effectiveness even in different levels of congestion. First, when the burst time (BT) is small (e.g., 1s-ON/10s-OFF), PRISM achieves 96.1% of the ideal throughput on average thanks to its adaptive weight adjustment and fast recovery. On the other hand, as the burst time increases (10s-ON), PRISM experiences slight performance degradation (e.g., 91.3%) due to the increased number of simultaneous packet drops. Next, when a bandwidth congestion level, which is defined as the percentage of WWAN link's bandwidth by which background traffic congests, rises, PRISM shows a slight performance degradation, but it still performs better than the other scheduling algorithms as shown in line points in Figure 12. Nonetheless, NIP-/WRR-based approaches show performance similar to ADAS thanks to fast loss recovery in RPC.

D. Maximizing Network Utilization

We show how much each component of PRISM contributes to the improvement of network utilization. We first show RPC's gain through ACK controls, and then show ADAS's contribution by reducing traffic burstiness.

1) Performance gain by RPC: We evaluated the RPC's improvement of network utilization under the same setting as in the bandwidth-disparity experiment, and compared three cases: PRISM without RPC, PRISM with only ACK re-sequencing (partial RPC), and PRISM with full RPC (including loss detection and fast loss recovery).

By effectively handling spurious duplicate ACKs, RPC maximizes network utilization which, in turn, enables PRISM to achieve a close-to-ideal aggregate bandwidth. Figure 13



(a) Per-flow average throughput (b) Covariance of throughput (c) Multiple PRISM flows

Fig. 15. *Bandwidth fairness of PRISM.* We introduce a various number of PRISM flows to vanilla-TCP flows (the total number of flows is 20), and measure (a) per-flow average throughput and (b) its covariance to show the TCP bandwidth fairness. Also, we run multiple PRISM flows in a community and (c) plot the congestion window size progression of each flow to show fair-share among multiple PRISM flows.

plots the performance gain by RPC. PRISM with the full RPC indeed achieves maximum network utilization even in the presence of large bandwidth disparities. By contrast, PRISM's performance without RPC is shown to be less than 50% of the ideal bandwidth. PRISM with a partial RPC makes, on average, only a 50% performance improvement since it should depend only on timeouts for packet-loss recovery.

2) Minimizing traffic burstiness: We evaluated ADAS's improvement of network utilization by measuring the degree of traffic burstiness that depends on the scheduling mechanism. We use four community members (whose WWAN bandwidths are 620, 720, 720, and 860 Kbps), and measure the size of re-sequencing buffer in PRISM-IMUX while running a PRISM flow under ADAS. We run PRISM with WRR for comparison.

ADAS reduces traffic burstiness by minimizing out-of-order deliveries, thus improving the overall network utilization. Figure 14 shows the progression of the re-sequencing buffer size which is defined as the distance between *left* and *highest* of the re-sequencing buffer. The average buffer size required by ADAS in the lower figure is 1.5 times less than that by WRR, meaning that ADAS generates less out-of-order packet deliveries than WRR. Also, the ADAS's smaller buffer size requirement implies less bursty traffic because PRISM-IMUX releases only a small number of stored ACKs to the sender. Our experimental results show that the throughput (2.9 Mbps) for less bursty traffic (scheduled by ADAS) improves by up to 16% over that (2.5 Mbps) for bursty traffic (scheduled by WRR).

E. Maintaining Bandwidth Usage Fairness

We show that PRISM maintains bandwidth fairness with other traffic and among PRISM flows. We study the effects of introducing PRISM on other vanilla-TCP performance and also the fairness among multiple PRISM flows.

1) Co-existence with TCP: We examined the fairness of PRISM with vanilla-TCP for the network topology in Figure 8. We introduce multiple independent TCP flows between $Host_{Si}$ and $Host_{Ri}$, and PRISM flows between S_j and M_j (the total number of flows including TCP and PRISM is fixed at 20), and compare per-flow throughput and its covariance. Note that each community member uses only its own WWAN link, and each flow competes with other PRISM and TCP flows on a shared bottleneck link (10 Mbps, 30 ms) in the Internet.

PRISM traffic co-exists well with (and hence, is friendly to) vanilla-TCP traffic without starvation or preemption on a shared link of the Internet. Whenever there is a shared bottleneck on a wired link, TCP-PRISM works exactly in the same way as the vanilla TCP (3 duplicate ACKs and halving the congestion window), and thus, is friendly to the vanilla TCP. Figure 15(a) shows per-flow throughput, and Figure 15(b)[8] shows the covariance of throughput as the number of PRISM flows increases. While per-flow throughput and its covariance fluctuate for the first 5 seconds due to the TCP's slow start, they quickly converge to their average values, demonstrating PRISM's fairness with vanilla-TCP.

2) Fair-share among multiple PRISM flows: We also evaluated the fairness among multiple PRISM flows where multiple receivers in a community initiate their own "aggregated" connection. We use four community members (whose bandwidths are 600, 900, 1200, and 1500 Kbps), and run three PRISM flows (from S_1, S_2, and S_3 to R_1, R_2, and R_3, respectively). Three PRISM-IMUX agents are invoked for each flow, and share four WWAN links.

PRISM preserves the end-side flow control, and its scheduling mechanism supports fairness among multiple PRISM flows. Figure 15(c) shows the progression of each sender's congestion window (CW) size. The three flows' CWs fluctuate with average sizes of 19.5, 18.4 and 21.8, and standard deviations of 4.6, 4.4 and 5.9, respectively.

IX. Related Work

A. Bandwidth aggregation in mobile hosts

Bandwidth aggregation in multi-homed mobile hosts is considered by several researchers. pTCP [18] is a new transport-layer protocol that is a wrapper around a slightly-modified version of TCP (called TCP-v). pTCP manages the send buffer across all the TCP-v connections and decouples loss discovery from congestion control, performs intelligent striping of data across the TCP-v connections, and does data reallocation

[8]Note that we reverse time progression for clarity. Also, we sample data once every second for the period of first 10 seconds and then once every 5 seconds.

to handle variances in the bandwidth-delay product of the individual connections. R^2CP [17] is also a transport-layer protocol that is a receiver-centric transport protocol that effectively addresses the wireless last-hop problem such as seamless handoff and server migration as well as bandwidth aggregation.

MOPED [12] is a framework to enable group mobility such that a user's set of personal devices appear as a single mobile entity connected to the Internet. MOPED provides the lightweight routing architecture (called MRCAP) that supports multipaths between the home agent and a receiver (network-layer approach), and also includes a new transport-layer protocol to aggregate the bandwidths of multiple links (transport-layer approach).

All of these approaches are efficiently enhancing mobile hosts to get better and more bandwidth. However, in PRISM, we extend bandwidth aggregation of multiple links from a single host to multiple collaborating mobile hosts.

B. Packet reordering

Packet reordering is a major problem in multi-path routing environments. DSACK [33] is a detection mechanism of spurious retransmissions on packet reordering for TCP. In DSACK, a sender receives spurious retransmission information from a receiver and reacts to it by restoring its congestion window to its value prior to the spurious retransmission.

TCP-Door [31] is another scheme for improving TCP in a MANET environment, which uses an additional sequence number, called *TCP packet sequence number*, to detect out-of-order packets by counting every data packet including retransmissions. Upon detecting out-of-order packets, the sender either temporarily disables congestion control or restores the state prior to entering the congestion avoidance if it is in the congestion-avoidance phase. These approaches address occasional out-of-order packet arrivals. However, they are reactive (as opposed to proactive) to out-of-order packets, and thus, channel bandwidths can be wasted on spurious retransmissions. This problem becomes severer when there are persistent out-of-order arrivals.

TCP-PR [9] addresses this problem using a timeout as an indication of packet loss instead of duplicate ACKs. It maintains 'to-be-acked' with timestamps, and detects packet loss if the current time t exceeds the packet's timestamp in the list plus an estimated maximum possible RTT. However, TCP-PR may still suffer from false timeouts that result from large RTT fluctuations. TCP-PR may increase maximum timestamps to reduce false timeouts, but it also requires a large buffer size to accommodate outstanding packets during such a long time.

C. Scheduling disciplines

Scheduling packets across multiple links is a well-known problem for which several solutions have been proposed. First, the round-robin (RR) scheduling guarantees long-term fairness, and is of low complexity. Surplus RR [4] adopts the weighted RR mechanism with a virtual buffer which resequences out-of-order packets. This approach is simple to implement, and provides long-term fairness among the links. However, like other RR mechanisms, it causes burstiness, and significant packet reordering may require a large resequencing buffer.

Second, the fair queueing attempts to approximate the Generalized Process Sharing (GPS) to achieve fairness and bounded delay, and there are several well-known solutions such as PGPS, WFQ, WF^2Q. However, these approaches assume that the exact bandwidths of each input and output link are known.

Third, a hybrid approach takes advantage of both RR and fair queueing. The stratified RR [25] classifies links into several groups based on their aggregate weight, and uses both earliest-deadline-first and RR. This algorithm removes the complexity of the timestamp approach. However, they also assume that the service rate and weight are fixed, and it is not desirable to change class whenever the link bandwidths change, especially in a mobile environment.

Finally, a wireless fair queueing approach addresses bursty errors and location-dependent losses. The authors of [24] evaluated various instances of wireless fair queueing and presented a unified architecture. Even though all of the instances work well in a wireless environment, they selectively require differentiated services (e.g., lagging and leading), a compensation model for fairness, and channel monitoring and prediction mechanisms. All these requirements need close interactions with base stations, incurring significant message exchange (with a proxy) and processing overheads.

X. Discussion and Conclusion

We first discuss a few issues associated with PRISM that have not been covered in this paper. Then, we make concluding remarks.

A. Discussion

PRISM can easily support upstream (from a mobile host to a server) traffic by placing PRISM-IMUX at a mobile node in the community. One mobile member in the community can work as the proxy and inverse-multiplex traffic over other community members. It might incur overheads to mobile hosts, but, as shown in Sections IV, V, and VI, the computational complexity of PRISM increases only on a log-scale, and its spatial complexity is also reasonable (3KB). Most of all, fast transmissions at an aggregate high data rate via members' collaboration contribute to the savings of a base power of mobile hosts. For example, let's assume Bob needs to upload a 10 MB file from his laptop, and Alice nearby Bob is reading an article in her laptop. If Bob uses own WWAN card (600 Kbps), the uploading takes 134 seconds, consuming 1,275 J (= (1.29 watt (WWAN) + 8.23 watt (system power)) $\times$ 134 s) of energy (based on measurements in [21], [28]). On the other hand, by using Alice's WWAN (600 Kbps) together, Bob can not only reduce uploading time to 67 s, but also reduce its energy consumption to 731 J (= (1.29 + 1.4 (WLAN) + 8.23) $\times$ 67 s). Finally, he will pay for Alice's bandwidth-related energy cost of 180 J (= (1.29 + 1.4) $\times$ 67 s) plus a premium for sharing a base power.

We also consider two different security-related issues: (i) what if the packet header is encrypted? and (ii) what if a community member behaves maliciously? Since PRISM exploits TCP information, it is critical for PRISM to extract the header information from each packet. As was done in [27], if we consider the proxy as a trusted party and let it hold the secret key for each connection, then the proxy can extract the header information from encrypted packets. This mechanism also helps prevent members' malicious behaviors from tampering with, or extracting data from, a packet. The other approach to the members' malicious behavior problem is to have a reputation and punishment system as in [11] to discourage such behaviors.

B. Concluding Remarks

In this paper, we first motivated the need for a mobile collaborative community: it improves the user-perceived bandwidth as well as the utilization of diverse wireless links. Then, we addressed the challenges in achieving bandwidth aggregation for a TCP connection in the community. Striping a TCP flow over multiple wireless WAN links requires significant scheduling efforts due to heterogeneous and dynamic wireless links, creates the need for frequent packet reordering due to out-of-order packet deliveries, and causes network under-utilization due to the blind reaction of the TCP's congestion-control mechanism. To remedy these problems, we proposed a proxy-based inverse multiplexer, called *PRISM*, that effectively stripes a TCP connection over multiple WWAN links at the proxy's network layer, masking adverse effects of out-of-order packet deliveries by exploiting the transport-layer information from ACKs. PRISM also includes a new congestion-control mechanism that helps TCP accurately respond to the heterogeneous network conditions identified by PRISM. Through experimental evaluation on a testbed and in-depth simulations, PRISM is shown to opportunistically minimize the need for packet reordering, effectively achieve the optimal aggregate bandwidth, and significantly improve wireless links utilization.

ACKNOWLEDGMENT

The work reported in this paper was supported in part by the AFSOR under Grant No. F49620-00-1-0327 and by the NSF under Grant No. CNS-0519498. This is an extended version of the paper [19] that appeared in ACM/USENIX MobiSys 2005.

REFERENCES

[1] Netfilter. http://www.netfilter.org.
[2] Nist net. http://snad.ncsl.nist.gov/nistnet.
[3] *ns-2* network simulator. http://www.isi.edu/nsnam/ns.
[4] H. Adiseshu, G. Parulkar, and G. Varghese. A reliable and scalable striping protocol. In *Proceedings of the ACM SigComm*, Stanford, CA, Aug. 1996.
[5] L. Anderegg and S. Eidenbenz. Ad hoc-VCG: A truthful and cost-efficient routing protocol for mobile ad hoc networks with selfish agents. In *In Proceedings of MobiCom*, San Diego, CA, Sept. 2003.
[6] A. Baiocchi and F. Vacirca. End-to-end evaluation of WWW and file transfer performance for UMTS-TDD. In *Proceedings of the IEEE GlobeCom*, Taipei, Nov. 2002.
[7] A. Bakre and B. R. Badrinath. I-TCP: Indirect TCP for mobile hosts. In *Proceedings of the 15th ICDCS*, May 1995.
[8] H. Balakrishnan, S. Seshan, E. Amir, and R. Katz. Improving TCP/IP performance over wireless networks. In *Proceedings of the ACM MobiCom*, Nov. 1995.
[9] S. Bohacek, J. P. Hespanh, J. Lee, C. Lim, and K. Obraczka. TCP-PR: TCP for persistent packet reordering. In *Proceedings of the 23rd ICDCS*, Rhode Island, May 2003.
[10] J. Border, M. Kojo, J. Griner, G. Montenegro, and Z. Shelby. Performance enhancing proxies intended to mitigate link-related degradations. Internet Request for Comments 3135 (rfc3135.txt), June 2001.
[11] S. Buchegger and J.-Y. L. Boudec. Performance analysis of the CONFIDANT protocol:cooperation of nodes. In *Proceedings of the ACM MobiHoc*, Lausanne, Switzerland, June 2002.
[12] C. Carter and R. Kravets. User device cooperating to support resource aggregation. In *Proceedings of the 4th IEEE WMCSA*, Callicoon, NY, June 2002.
[13] M. C. Chan and R. Ramjee. TCP/IP performance over 3G wireless links with rate and delay variation. In *Proceedings of the ACM MobiCom*, Atlanta, GA, Sept. 2002.
[14] J. Duncanson. Inverse multiplexing. *IEEE Communications Magazine*, 32(4), Apr. 1994.
[15] D. Farinacci, S. Hanks, D. Meyer, and P. Traina. Generic routing encapsulation (GRE). Internet Request for Comments 2784 (rfc2784.txt), Mar. 2000.
[16] S. Floyd. Congestion control principles. Internet Request for Comments 2914 (rfc2914.txt), Sept. 2000.
[17] H. Hsieh, K. Kim, Y. Zhu, and R. Sivakumar. A reciver-centric transport protocol for mobile hosts with heterogeneous wireless interfaces. In *Proceedings of the ACM MobiCom*, San Diego, CA, Sept. 2003.
[18] H. Hsieh and R. Sivakumar. A transport layer approach for achieving aggregate bandwidths on multi-homed mobile hosts. In *Proceedings of the ACM MobiCom*, Atlanta, GA, Sept. 2002.
[19] K. Kim and K. G. Shin. Improving TCP performance over wireless networks with collaborative multi-homed mobile hosts. In *Proceedings of the ACM/USENIX MobiSys*, Seattle, WA, June 2005.
[20] L. Magalhaes and R. Kravets. MMTP:multimedia multiplexing transport protocol. In *Proceedings of SigComm-LA*, San Jose, Costa Rica, Apr. 2001.
[21] A. Mahesri and V. Vardhan. Power consumption breakdown on a modern laptop. In *Proceedings of the IEEE International Symposium on Microarchitecture*, Portland, OR, Dec. 2004.
[22] S. Marti, T. Giuli, K. Lai, and M. Baker. Mitigating routing misbehavior in mobile ad hoc networks. In *Proceedings of the ACM MobiCom*, Boston, MA, Aug. 2000.
[23] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow. TCP selective acknowledgement options. Internet Request for Comments 2018 (rfc2018.txt), Oct. 1996.
[24] T. Nandagopal, S. Lu, and V. Bharghavan. A unified architecture for the design and evaluation of wireless fair queueing algorithms. In *Mobile Computing and Networking*, 1999.
[25] S. Ramabhadran and J. Pasquale. Stratified round robin: A low complexity packet scheduler with bandwidth fairness and bounded delay. In *Proceedings of the ACM SigComm*, Karlsruhe, Germany, Aug. 2003.
[26] P. Rodriguez, R. Chakravorty, J. Chesterfield, and I. Pratt. MAR: A commuter router infrastructure for the mobile internet. In *Proceedings of the ACM MobiSys*, Boston, MA, June 2004.
[27] N. B. Salem, L. Buttyan, J.-P. Hubaux, and M. Jakobsson. A charging and rewarding scheme for packet forwarding in multi-hop cellular networks. In *Proceedings of the IEEE/ACM MobiHoc*, Annapolis, MD, June 2003.
[28] E. Shih, P. Bahl, and M. J. Sinclair. Wake on wireless: An event driven energy saving strategy for battery operated devices. In *Proceedings of the ACM MobiCom*, Atlanta, GA, Sept. 2002.
[29] P. Sinha, N. Venkitaraman, R. Sivakumar, and V. Bharghavan. WTCP: A reliable transport protocol for wireless wide-area networks. In *Proceedings of the ACM MobiCom*, Aug. 1999.
[30] J. Veizades, E. Guttman, C. Perkins, and S. Kaplan. Service location protocol. Internet Request for Comments 2165 (rfc2165.txt), June 1997.
[31] F. Wang and Y. Zhang. Improving TCP performance over mobile ad-hoc networks with out-of-order detection and response. In *Proceedings of the ACM MobiHoc*, Lausanne, Switzerland, June 2002.
[32] J. Widmer. Network simulations for a mobile network architecture. http://www.icsi.berkeley.edu/widmer/mnav/ns-extension.
[33] M. Zhang, B. Karp, and S. Floyd. Improving TCP's performance under reordering with DSACK. Technical report, International Computer Science Institute, Technical Report ICSI TR-02-006, July 2002.

[34] S. Zhong, J. Chen, and Y. Yang. Sprite: A simple, cheat-proof, credit-based system for mobile ad-hoc networks. In *Proceedings of the IEEE InfoCom*, San Francisco, CA, Apr. 2003.

Kyu-Han Kim received the BS from the Department of Computer Science and Engineering at Korea University, Seoul, Korea in 2000 and the MS from the College of Computing at Georgia Institute of Technology, Atlanta, GA in 2003. Since 2003, he has been pursuing Ph.D degree in the Real-Time Computing Laboratory at the Department of Electrical Engineering and Computer Science, The University of Michigan, Ann Arbor, Michigan. His researh interests include quality of service support in wireless networks and distributed systems—wireless mesh networks, mobile ad-hoc networks, and cognitive radios. He is a recipient of government scholarship from the ministry of information and communication, Republic of Korea.

Kang G. Shin is the Kevin and Nancy O'Connor Professor of Computer Science and Founding Director of the Real-Time Computing Laboratory in the Department of Electrical Engineering and Computer Science, The University of Michigan, Ann Arbor, Michigan.

His current research focuses on QoS-sensitive networking and computing as well as on embedded real-time OS, middleware and applications, all with emphasis on timeliness and dependability. He has supervised the completion of 56 PhD theses, and authored/coauthored more than 650 technical papers (more than 230 of which are in archival journals) and numerous book chapters in the areas of distributed real-time computing and control, computer networking, fault-tolerant computing, and intelligent manufacturing. He has co-authored (jointly with C. M. Krishna) a textbook "Real-Time Systems," McGraw Hill, 1997.

He has received a number of best paper awards, including the IEEE Communications Society William R. Bennett Prize Paper Award in 2003 and an Outstanding IEEE Transactions of Automatic Control Paper Award in 1987. He has also received several institutional awards, including Distinguished Faculty Achievement Award in 2001, and Stephen Attwood Award in 2004 from The University of Michigan; a Distinguished Alumni Award of the College of Engineering, Seoul National University in 2002; 2003 IEEE RTC Technical Achievement Award; and 2006 Ho-Am Prize in Engineering. He is Fellow of IEEE and ACM, and member of the Korean Academy of Engineering.