Improving TCP Performance over Wireless Networks with Collaborative Multi-homed Mobile Hosts

Kyu-Han Kim and Kang G. Shin
Department of Electrical Engineering and Computer Science
The University of Michigan, Ann Arbor
{kyuhkim,kgshin}@eecs.umich.edu

Abstract

Multi-homed mobile hosts situated in physical proximity may spontaneously team up to run high-bandwidth applications by pooling their low wireless wide-area network (WWAN) bandwidths together for communication with a remote application server and utilizing their high-bandwidth wireless local-area network (WLAN) in ad-hoc mode for aggregation and distribution of application contents among the participating mobile hosts. In this paper, we first describe the need for such a mobile collaborative community, or a community, in which multi-homed mobile hosts exploit the diversity of WWAN connections to improve a user-perceived bandwidth and network utilization. Then, we show that existing one-to-one communication protocols like TCP suffer significant performance degradation due to frequent packet reordering and heterogeneity of WWAN links in the community.

To address the above TCP problem, we propose a proxy-based inverse multiplexer, called *PRISM*, that enables TCP to efficiently utilize the community members' WWAN connections. PRISM runs at a 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 variety of adverse effects specific to each WWAN link via intelligent ACK-control mechanism. Finally, PRISM includes a sender-side enhancement of congestion control, enabling TCP to respond correctly to dynamically-changing network states.

We have evaluated the PRISM protocol using both experimentation and *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 also shows that PRISM delivers a near-optimal aggregated bandwidth in the community formed by heterogeneous mobile hosts, and improves network utilization significantly.

1 Introduction

As wireless networks become omnipresent, mobile users are gaining access to the Internet via a variety of wireless networks. To keep pace with the trend, a mobile host is equipped with multiple wireless network interfaces (e.g., GPRS, IEEE 802.11x, and Bluetooth). Based on such diversity, several researchers attempted to enhance network availability, focusing on concurrent (or alternative) use of multiple wireless technologies available on a host, a mobile user or a designated mobile center [8, 12, 17]. That is, they have attempted to improve network availability within a single individual multi-homed entity.

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 share their WWAN link bandwidths with other members in the community. Possible applications of this include (i) contents sharing where each host with same interests receives a subset of contents from an Internet server and share the contents with other hosts, and (ii) bandwidth sharing where one host in a community needs more bandwidth than its own WWAN link for applications like video-on-demand or Hi-Definition TV live cast.

We, therefore, advocate formation of a mobile collaborative community that is a user-initiated network service model and that allows bandwidth sharing/pooling among multi-homed mobile users to make the best of their diverse WWAN links. The mobile community is different from a current WWAN service model, which forces a mobile host to use a single WWAN link at a time, causing capacity, coverage, and hardware limitations. By contrast, the community helps mobile users initiate new virtual WWANs that overcome such limitations by sharing their WWAN interfaces. Moreover, by adopting an inverse multiplexer [9], the mobile community effectively aggregates its members' WWAN bandwidths.

However, the existing transport protocols, such as the Transmission Control Protocol (TCP), are not aware of existence of multiple and parallel intermediate links, and thus, cannot exploit multiple available WWAN links in the community. Even with the help of a multi-path routing protocol, frequent out-of-order packet delivery due to the heterogeneity of WWAN links significantly degrades TCP's performance.

There are several transport-layer solutions for bandwidth aggregation of a *single* multi-homed mobile host such as those in [11–13], but 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. Furthermore, the development and deployment of a whole new transport protocol requires significant efforts on both content servers and mobile clients, and also incurs a high com-

putational overhead to resource-scarce 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 TCP's complementary protocol, PRISM consists of (i) an inverse multiplexer (PRISM-IMUX) that handles TCP's data and acknowledgment (ACK) traffic at a proxy, and (ii) a new congestion control mechanism (TCP-PRISM) that effectively handles, at a sender side, packet losses over the community's WWAN links.

The first component stripes TCP traffic intelligently over multiple WWAN links using up-to-date link utilization information and each packet's expected time of arrival at a receiver. Also, it masks the effects of out-of-order delivery by identifying spurious duplicate ACKs and resequencing them so that a TCP sender receives correctly-sequenced ACKs.

The second component in PRISM is a sender-side congestion control mechanism (TCP-PRISM) that reduces the loss recovery time and accurately adjusts a 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 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 kernel loadable module and extensively evaluated on a testbed. Our experimental evaluation has shown 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 packet reordering and delivers the near-optimal aggregated bandwidth in the community that is composed of heterogeneous mobile hosts.

The rest of this paper is organized as follows. Section 2 presents the motivation and the contributions of this work, and Section 3 provides an overview of the PRISM architecture. Sections 4–6 give detailed accounts of PRISM. Section 7 describes our implementation and experimentation experiences. Section 8 evaluates the performance of PRISM using *ns-2*-based simulation. Related work is discussed in Section 9. Finally, Section 10 discusses a few remaining issues with PRISM and concludes the paper.

2 Motivation

We first describe the motivation of a mobile 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.

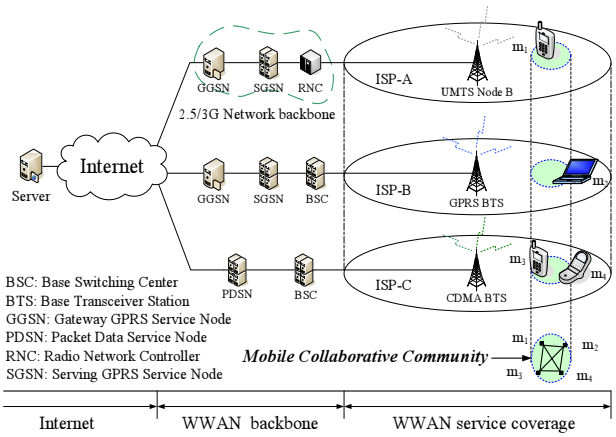


Figure 1: *Target environment.* The environment includes various WWAN network services available and multi-homed mobile hosts equipped with both WWAN and WLAN interfaces. Mobile hosts in a WLAN range form a mobile community and collaborate to simultaneously use multiple WWANs.

2.1 Why a Mobile Community?

Wireless network services become available anywhere and anytime. 2.5G and 3G wide-area cellular networks (e.g., GPRS, UMTS, and CDMA) are being deployed for more bandwidth and wider coverage. Moreover, WLANs (e.g., IEEE 802.11x) can now 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 interfaces (e.g., WWAN, WLAN, and Bluetooth) and can select the best interface/channel available at a given place/time.

Although there exist many 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 to get a common service. As a result, they suffer from various service limitations as follows.

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 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 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 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 movie files 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 (m_1, m_2) or different channels[1] of the same ISP (m_3, m_4), and forward packets via WLAN interfaces in ad-hoc mode.

2.2 How Does a Mobile Community Work?

As basic building blocks, a mobile community has three functions: collaboration, multiplexing, and indirection.

2.2.1 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 content at others' expenses, will the other members be willing to contribute their bandwidth 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 [7, 18, 23]. In ad hoc networks, the communication between end-points 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 [7, 14]. Game-theoretic arguments have been used to show that collaboration on packet forwarding among all participating nodes will yield maximum network throughput.

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 3, 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.

2.2.2 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 [9]. 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 in the receiver at the aggregate rate.

Then, an issue is where to put the inverse multiplexer. The inverse multiplexer can be placed at a performance-enhancing proxy by a network access provider, a wireless telecommunication service provider, or a content distribution network operator. Furthermore, it can be placed in a network layer as one routing component with an efficient traffic filtering function as in the Network Address Translation (NAT) service. On the other hand, 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 [12].

2.2.3 Indirection

Traffic from an inverse multiplexer to community members is tunneled via Generic Routing Encapsulation (GRE) [10]. The inverse multiplexer encapsulates the traffic via GRE and routes it to the community members. Each member de-capsulates the tunneled traffic, upon its reception, 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.

2.3 Challenges in MC^2's Use of TCP

Our primary contribution in this paper is that in an MC^2, we enable *one-to-many-to-one* communication for a TCP connection to achieve high-speed Internet access. 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 links.

In this communication model, however, we encounter several challenges. First, *scheduling* traffic over wireless links requires exact link state information such as data

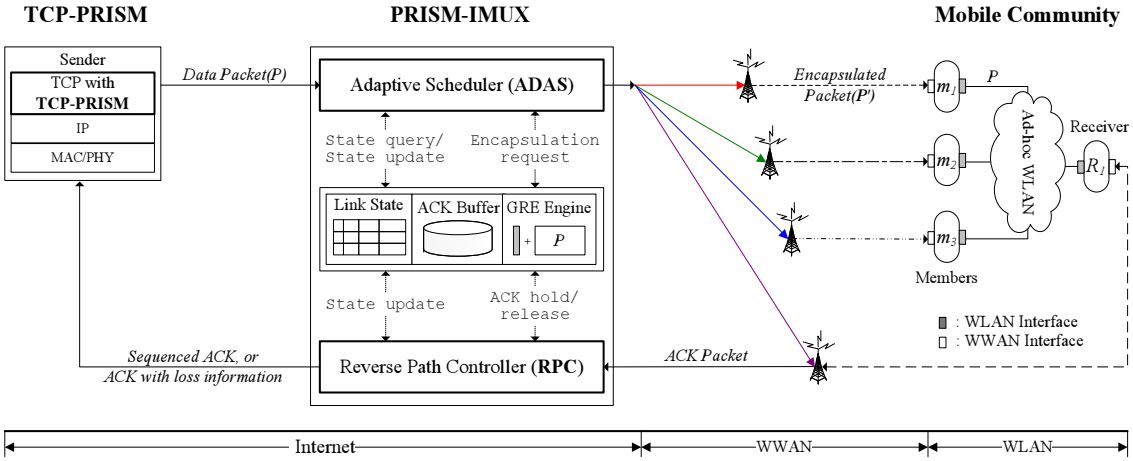


Figure 2: *PRISM architecture*. PRISM consists of an inverse multiplexer at the proxy (PRISM-IMUX) and a sender-side congestion control mechanism (TCP-PRISM). PRISM-IMUX captures each data packet in the middle of a sender and a receiver. After selecting the best WWAN link for each packet's next hop using the Adaptive Scheduler (ADAS), PRISM-IMUX forwards the packet to the WWAN link after encapsulating it via GRE. The encapsulated packet arrives at a community member via the WWAN link, and the member de-capsulates and forwards the packet to the receiver. Next, the receiver receives and processes the packet as a normal packet, and then returns an ACK packet to the sender. PRISM-IMUX again captures the in-transit ACK and decides whether the ACK is a spurious duplicate ACK or not, using the Reverse Path Controller (RPC). If PRISM-IMUX detects packet losses from duplicate ACKs, it releases the ACK with loss information to the sender, and TCP-PRISM at the sender uses the delivered loss information for fast loss recovery. If there is no loss, PRISM-IMUX holds or releases the ACK in a sequenced order.

rate and delay, which varies with time and is usually expensive to obtain in mobile environments. Second, because WWAN links suffer from high and variable round trip times (RTTs), burstiness and outages, the large number of *out-of-order* packet deliveries, which 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.

2.4 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 stripes traffic with the least cost while maintaining full links utilization;
- **C.2** construct an ACK-control mechanism that effectively shields the effects of out-of-order delivery without sacrificing end-to-end performance; and
- **C.3** propose a new congestion-control mechanism that is a sender-side optimization technique and that improves links utilization by expediting loss recovery.

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: (i) each mobile host has multiple (especially WWAN and WLAN) interfaces that can be used simultaneously for a single application connection; (ii) a mobile community is formed via an application-layer daemon; (iii) GRE is enabled as a default; and (iv) all hosts support TCP-SACK.

3 PRISM Architecture

Figure 2 depicts the 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 that is composed of multi-homed mobile hosts.

3.1 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. PRISM-IMUX captures data traffic from a sender in the proxy's network layer, and decides the best WWAN link as a next-hop through the Adaptive Scheduler (ADAS). It also captures and controls ACK packets to shield the adverse effects of striping over multiple WWAN links via the Reverse Path Controller (RPC). Finally, PRISM-IMUX maintains a WWAN links' state table, has a buffer that temporarily stores ACKs which need to be re-sequenced, and supports GRE for indirection. We will detail ADAS in Section 4, and RPC in Section 5.

3.2 TCP-PRISM

TCP-PRISM is a new sender-side congestion-control mechanism that works with PRISM-IMUX to expedite loss recovery, thus improving network utilization. TCP-PRISM reduces the loss recovery time via using the *negative* ACK information shipped by RPC at the proxy to detect a packet loss. Also, it adjusts the congestion window size according to the congested link bandwidth only, thus preventing waste of uncongested links' bandwidth. We will detail this in Section 6.

3.3 Mobile Community

A mobile community is formed voluntarily and incrementally. When a new mobile node wants to join an existing mobile community, it first searches for communities nearby using the Service Location Protocol [19]. After determining the community of most interest to itself, the mobile node/host 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.

4 Scheduling Wireless Links: ADAS

4.1 Overview

Scheduling TCP packets over heterogeneous wireless links requires exact link state information for a receiver to achieve the optimal aggregate bandwidth, and obtaining the information is expensive, especially in mobile environments due to dynamic 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. 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 overheads (e.g., message 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 is flexible enough to deal with the lack of rate information. ADAS maintains up-to-date links state which is inexpensively obtained by RPC (see Section 5), and adaptively schedules traffic 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 a link's congestion via a TCP's AIMD-like traffic control mechanism.

Algorithm 1 ADAS_Scheduling ($Packet$)

$hRTT_i$	RTT from a proxy to a receiver over WWAN $link_i$
l_{next}	Next link which a packet is assigned to
N_i	The number of in-flight packets on $link_i$
$\mathbb{S}$	A set of community members' WWAN links
$\mathbb{S}_{next}$	A set of links with the minimum utilization
U_i	Link utilization of $link_i$

```
1: if Packet is a retransmission then
2:    // Rule.1: send the retransmission over a fast link
3:    l_next is the link with minimum hRTT in S
4: else
5:    // Choose links with the most available bandwidth
6:    S_next = {l_i: links with minimum U_i from S}
7:    if |S_next| == 1 then
8:       // Rule.2: use link utilization (U_i)
9:       l_next is the link (in S) with minimum U_i
10:   else
11:      // Rule.3: use an expected time of arrival and U_i
12:      l_next is the link with minimum hRTT in S_next
13:   end if
14: end if
15: Update N_i, U_i for l_next        // update channel information
16: Update S                          // apply new U_i to the set S
17: return l_next
```

4.2 Algorithm

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

4.2.1 Link Utilization (U_i)

Link utilization enables ADAS to utilize multiple links fairly so as to maximize an aggregated bandwidth. U_i is derived from the Weighted Round Robin (WRR) scheduling for its fairness. WRR divides the time into a round, 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 fair link utilization, thus achieving long-term fairness as WRR does.

However, ADAS uses a different definition of 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*. Because existing static scheduling algorithms (e.g., WRR) assume accurate and stable links state information, the link utilization based on the algorithm might not be accurate due to network dynamics. ADAS exploits the actual number of in-flight packets, which can be derived from scheduled packets' information and ACK-control information, and which automatically reflect unexpected delay or loss

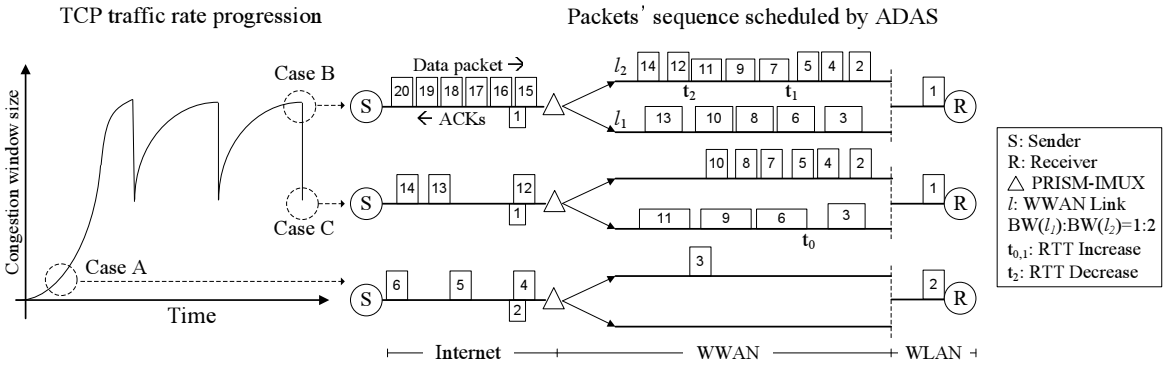


Figure 3: *Three ADAS scheduling snapshots in different input/output rates.* The left graph shows the fluctuation of input rate from a TCP sender. The right three wide figures are snapshots of ADAS scheduling in different input/output rates. Case A shows Rule.2 (based on link utilization), Case B shows Rule.3 (based on RTT as well as U_i), and Case C shows dynamic weight adjustment.

of a link, in order to determine the utilization of each link. Therefore, we define "link utilization" as $U_i = \left\lfloor \frac{N_i}{W_i} \right\rfloor$, where N_i is the number of in-flight packets over link i, and W_i (link weight) is the ratio of the link bandwidth to the least common denominator/factor of all links' bandwidths.

Let's consider Case A in Figure 3 to see the effectiveness of U. 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.

4.2.2 Expected Arrival Time ($hRTT$)

ADAS uses expected arrival time (half RTT or $hRTT$) along with U to further improve overall link utilization and minimize the need for packet reordering. When more than one link ($\mathbb{S}$) have the same lowest link utilization, 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' unexpected processing delay, links with similar utilization might experience different short-term rates or delay fluctuations which might not be immediately reflected into U. Using $hRTT$ ensures that ADAS transmits packets on the fastest link in a greedy fashion, and thus, not only increases the overall short-term links utilization, but also reduce out-of-order packet deliveries at a receiver.

Let's consider Case B in Figure 3 to illustrate the effectiveness of $hRTT$. 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 at the point of scheduling p_8, the expected arrival time of p_8 via ℓ_2 becomes longer than that via ℓ_1. Besides, since the U 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.

4.2.3 Dynamic Link-Weight Adjustment

ADAS adapts to each link's congestion without separate links state probing or congestion notification messages from networks 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. If the link experiences congestion, ADAS cuts the congested link's weight by half. Subsequently, the link's U_i becomes larger, and new packets are not assigned to the link until it recovers from the congestion. This link weight increases additively each time an ACK arrives on that link, without exceeding the original weight. This way, ADAS adaptively reacts to partial links' congestion and controls the amount of traffic to be allocated to each link without requiring expensive instantaneous bandwidth information.

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 an increased $hRTT$. However, ADAS schedules p_6 on ℓ_1 based on U to maintain maximum network utilization even though it might cause packet reordering. On the other hand, right before scheduling p_{11}, ADAS identifies a packet loss on ℓ_2. It adaptively reduces the ℓ_2's weight, and assigns p_{11} to ℓ_1 based on the new computed link utilization.

4.3 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 ta-

ble as shown in Figure 2. It independently stores per-link information which includes only four variables (i.e., U_i, $hRTT_i$, W_i and N_i).

5 Handling Spurious Duplicate ACKs: RPC

5.1 Overview

Even though ADAS tries to minimize the need for packet reordering, data packets are sometimes scheduled out-of-sequence explicitly 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 meaning of duplicate ACKs and correct them, if necessary. Moreover, along with the 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.

5.2 Algorithm

RPC consists of three different mechanisms: ACK identification, ACK re-sequencing, and loss detection. RPC accepts ACKs as input, and then holds or releases them based on the above three mechanisms. RPC first identifies the meaning of an arrived ACK. Then, it decides whether this ACK is normal or spurious. Finally, it differentiates duplicate ACKs caused by real packet losses from spurious duplicate ACKs, and detects any packet loss.

5.2.1 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[2] contains the sequence number of the recently-arrived data packet [15], RPC can infer the state of the receiver's buffer via following two ways.

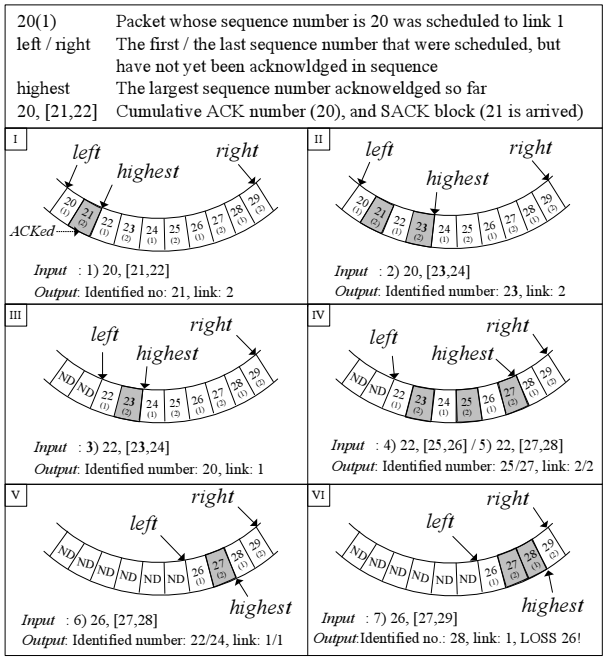


Figure 4: *Snapshots for ACK identification and loss detection mechanism in RPC.* RPC identifies the meaning of each ACK through SACK blocks (Snapshot I, II, and IV) or cumulative ACK numbers (Snapshot III, V). Also, it detects packet losses using identified sequence numbers and scheduling history (Snapshot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

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, the snapshot I, II, and IV show the SACK block matching scheme. The snapshot III and V illustrate how cumulative ACK numbers are scanned. Each snapshot contains the circular buffer representing the meta-state of the receiver buffer.

5.2.2 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 based on Algorithm 2. If the identified sequence number proceeds towards a new unACKed sequence number, RPC starts releasing ACKs-on-hold including the one just arrived.

If arrived ACK packets are duplicates (line 8), then RPC re-sequences them in two different ways. First, if there

Algorithm 2 ACK re-sequencing ($Packet$)

```
1: // N is the size of circular ACK re-sequencing buffer
2: wSeqNum = (cumulative ACK sequence number) mod N
3: if wSeqNum > left then
4:    left = wSeqNum        // in-sequence, new ACK
5:    put ACK at the end of the re-sequencing buffer
6:    release held ACK(s) up to left point with a peak rate
7: end if
8: if wSeqNum == left then
9:    switch RPC state      // duplicate ACK
10:       case NORMAL        // re-sequencing
11:          hold ACK with an identified number, break
12:       case LOSS          // no re-sequencing
13:          release ACK to a sender, break
14:   end switch
15: end if
```

is not any congested link (i.e., all links are in NORMAL state), then RPC holds the ACK packet in the slot of the wrapped-around re-sequence number ($wSeqNum$) in the circular ACK buffer. Since RPC knows the meaning of an ACK, it corrects the cumulative ACK sequence number with the identified number of the ACK packet and stores it in the buffer. Second, if RPC is in the LOSS state, it 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.

5.2.3 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 a 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. The snapshot VI shows an example of loss detection of RPC. Since packets 26, 28 were sent back-to-back via link 1, RPC 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.

5.3 Complexity

RPC's complexity heavily relies on the number of duplicate ACKs. When there is no duplicate ACKs, RPC does not incur any overhead except for updating link-state variables (U, N). However, if there are duplicate ACKs resulting from either out-of-order delivery or a packet loss, then RPC needs to figure out (compute) the ACK's sequence number and space to re-sequence ACKs.

Given duplicate ACKs, RPC only requires constant time complexity and 3KB space complexity per connection in the worst case. The most computation-intensive mechanism in RPC is ACK identification which requires extensive sequence comparison. However, this overhead can be minimized using such optimization techniques as a bit-operation, and thus requires constant time complexity. RPC may have to store all ACKs of a flow in the worst case. Since the number of ACKs is limited by the number of outstanding packets in the network, $\frac{BDP}{MSS} \times S_{ACK}$ is the maximum required ACK re-sequencing buffer size. For example, assuming that an aggregated bandwidth and average RTT are 5 Mbps and 120 ms, respectively, and MSS is 1.5 KB and the size of ACK (S_{ACK}) is 60 bytes, a maximum space requirement is 3 KB.

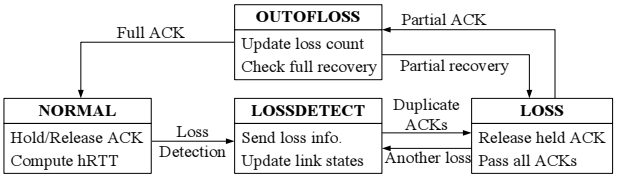


Figure 5: *State machine of RPC.* Boxes with capital letters indicate states of RPC, and boxes with small letters list operations in each state.

6 Expediting Loss Recovery: TCP-PRISM

6.1 Overview

Along with ACK re-sequencing and loss detection, fast recovery from loss(es) and appropriate congestion control at the sender side are critical to the overall performance of a TCP connection. Although many 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 mainly for 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 heterogeneous paths, resulting in frequent timeouts. Second, they over-reduce the window size upon congestion of one of multiple paths, reducing the overall links utilization.

PRISM addresses these problems by using two mechanisms. The first mechanism provides exact loss/congestion information in a negative form to a 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.

6.2 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.

6.2.1 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 updates state variables as described in Section 5. 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 which can be computed by $p = 1 - \frac{B_i}{2\sum_{j=1}^{n} B_j}$ 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 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.

6.2.2 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 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 reduces the congestion window size only by the proportion (p) of congested link's bandwidth over total bandwidth—we call this adjustment *Additive Increase and Proportional Decrease*. 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 first reduction step. Other than the above two enhancements, TCP-PRISM works exactly same as the way vanilla-TCP does.

6.3 Complexity

The complexity of TCP-PRISM is lower than that of the standard TCP-SACK. TCP-SACK's scoreboard mechanism maintains positive ACK information from a receiver and then identifies lost segments. It requires an extensive search to construct up-to-date blocks whenever a SACK block is delivered, and hence, may require more memory space. In contrast, TCP-PRISM uses a simplified version of scoreboard, which only maintains a list of lost packets from the negative loss information.

7 Implementation

We have implemented, and experimented with, PRISM. In this section, we first describe the implementation details of each PRISM component. Then, we describe our testbed setup and present the experimental results.

7.1 Implementation Details

7.1.1 PRISM-IMUX

PRISM-IMUX is implemented as a loadable kernel module in a network layer using Netfilter [1]. Netfilter provides a hook for packet filtering at the network layer, allowing users to dynamically register or un-register any filter. Thus, PRISM-IMUX is implemented as a filter with a back-end agent which includes such mechanisms as ADAS, RPC, and link's state maintenance.

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

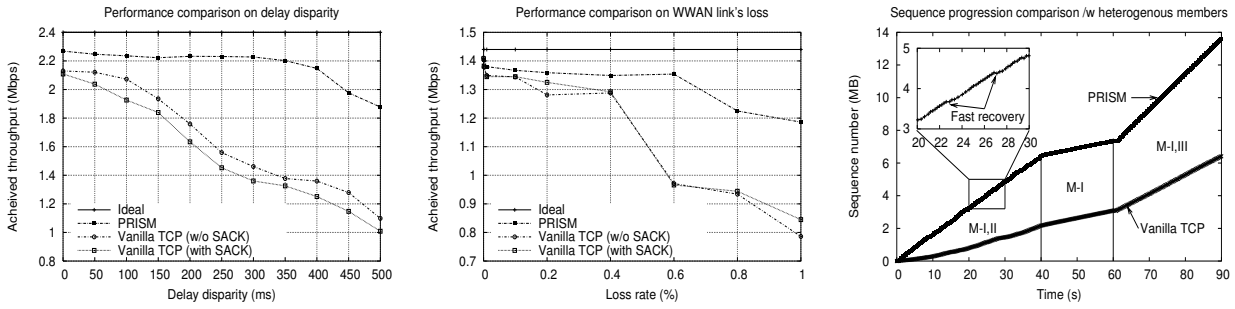
Finally, there is a case where the agent of the filter needs to store packets, and thus stop the remaining packet processing chain in the network layer. There are two options (NF_DROP and NF_STOLEN) from Netfilter to silently store a packet, and PRISM-IMUX uses NF_STOLEN because it does not incur any overhead such as buffer-copy which is required in NF_DROP.

7.1.2 TCP-PRISM

We implemented TCP-PRISM in a Linux kernel-2.4's TCP protocol stack, and deployed it in a server. As described in Section 6, TCP-PRISM is a simplified version of TCP-SACK and easily extensible from TCP-SACK implementation. TCP-SACK maintains sack-tag information, which is initially cleared, and becomes "SACKED" when the corresponding sack information arrives. However, determining an un-sacked packet as a lost packet is still not an easy problem, and thus, TCP-SACK has both a heuristic decision algorithm and an undo algorithm to fix any incorrect decision. We modified this sack-tag mechanism so that the exact loss information provided by PRISM-IMUX is immediately reflected into the sack-tag information.

7.2 Testbed Setup

To evaluate our PRISM implementation, we have built a testbed that is composed of an Internet infrastructure, and a mobile community.



(a) Effects of delay disparity

(b) Effects of loss-rate difference

(c) Effects of link heterogeneity

Figure 6: *Experimental results.* We run PRISM in our testbed and compare its performance with that of vanilla-TCP. In the experiment (a), we measure the throughput achieved by both PRISM and vanilla-TCP while increasing the bandwidth disparity of WWAN links. In the experiment (b), we also measure the throughput while increasing WWAN link's loss rates. In the experiment (c), we conduct the same experiment in a community consisting of heterogeneous members. Member 1 (M-I) with a slow link (360 Kbps) and member 2 (M-II) with fast but a lossy link (1080 Kbps, 0.6%) initially collaborate, then M-II leaves the community (at 40s), and member 3 (M-III) with a fast link (1800 Kbps) joins (at 60s) and collaborates with M-I.

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. The server and the proxy have TCP-PRISM and PRISM-IMUX installed, respectively. The emulator has NISTnet [2] to emulate WWAN networks (the proxy and the emulator each have two Ethernet interfaces to construct different networks). The Ethernet switch works as WWAN access networks and splits traffic from the emulator to each community member. The server, the proxy, the emulator, and the switch in a row are connected via 100 Mbps Ethernet cables between successive 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 interfaces (Orinoco). Each Ethernet interface is connected to Ethernet switch's 100 Mbps cables and is used as WWAN links. A WLAN interface in ad-hoc mode is used for communication within the community.

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

7.3 Experimental Results

7.3.1 Effects of delay disparity

We evaluated the performance tolerance of PRISM to the WWAN links delay disparity. We use two community members which have different 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).[3] While increasing one link's delay up to 1000 ms in increments of 50 ms, we measure end-to-end throughput. For better comparison, we also run vanilla-TCP with and without SACK.

PRISM effectively masks the delay disparity of WWAN links and provides an aggregated bandwidth through RPC's re-sequencing mechanism. Figure 6(a) shows that PRISM achieves 95% throughput of total aggregate link 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 significant performance degradation due to spurious duplicate ACKs. Furthermore, vanilla-TCP with SACK shows worse performance than that without SACK because detailed SACK information delivered to a sender causes significant false retransmissions.

7.3.2 Effects of loss-rate disparity

We measured the performance tolerance of PRISM to the WWAN links loss-rate difference. In the community with two members (whose bandwidths are 1080, 360 Kbps), we fix the link delay of both members at 300 ms (average delay from the UMTS trace with the packet size of 1 KB) and measure end-to-end throughput as we vary the loss-rate from 0.001% to 1% of the second member (1% is a typical maximum loss-rate of WWAN links).

The fast-recovery mechanism in PRISM indeed expedites loss recovery and increases link utilization even at a high loss-rate. As shown in Figure 6(b), PRISM provides 94% throughput of the total 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 relatively good performance (i.e., 90%) at a low loss rate, vanilla-TCP immediately shows degraded performance as the

loss-rate increases because of the long loss-recovery time for one congested link, blocking the uncongested link.

7.3.3 Effects of link heterogeneity

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

PRISM achieves the aggregated bandwidth of all WWAN links even in case of heterogeneous link characteristics. Figure 6(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% more throughput than 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 6(c)). Furthermore, PRISM yields 208% better performance than TCP in case of a large bandwidth disparity (ranging from 60s to 90s) from its effective scheduling mechanism and ACK re-sequencing mechanism.

8 Performance Evaluation

We also evaluated PRISM via in-depth simulation in diverse environments.We begin with a simulation model and then evaluate PRISM with respect to bandwidth aggregation, packet reordering, and network utilization.

8.1 Simulation Models

We use the *ns-2* [3] for our simulation study. The network topology in Figure 10 is used for this study and consists of Internet, WWAN, and WLAN networks and nodes. The Internet is composed of fixed servers (sender S_i), a proxy, and other hosts ($Host_{S/R}$) for background traffic. The bandwidth between hosts and their edge router is 20 Mbps, and the bandwidth between routers is 10 Mbps.

For WWANs, we use the Universal Mobile Telecommunication System (UMTS) *ns-2* extension [5]. B_i is a base-station node that has support for WWAN links. For WLANs, we use the IEEE 802.11b implementation in *ns-2*, and add NOAH [21] routing protocol to simulate peer-to-peer communication among community members. For each community member, we use an *ns-2* mobile node with extension for supporting multiple wireless interfaces.

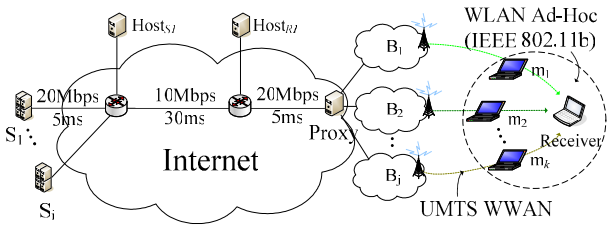


Figure 10: *Simulation network topology.* S_i (server), $Host_{S,R}$ (background traffic generator), a Proxy, B_j (base station), m_k (community members), and R (receiver)

We set up a PRISM flow(s) between a sender(s) and a receiver(s), and measure the end-to-end throughput between them. We implement TCP-PRISM (an extension of TCP SACK) for the sender's transport layer, and place PRISM-IMUX at the proxy. We run an FTP application between the sender and the receiver for 150–500 seconds.

8.2 Achieving Bandwidth Aggregation

We measured aggregated bandwidth gains by PRISM while increasing the number of WWAN links. For a scenario with i links, we randomly choose each link bandwidth between 400 Kbps and 2.4 Mbps. We first run an FTP application 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 better comparison, we also run the same experiment under a weighted-round-robin (WRR) striping agent without an ACK-control mechanism.

PRISM achieves the aggregated bandwidth that reaches the sum of link bandwidths, and its performance scales well with the community size. Figure 7 plots the bandwidth aggregation gain by PRISM and confirms the performance gain and scalability with up to five community members.[4] 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's throughputs achieved in each WWAN link.

8.3 Minimizing Need for Packet Reordering

8.3.1 Bandwidth disparity

We evaluated ADAS's performance in the presence of disparity between WWAN links' bandwidths. We use three community members whose 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. Then, we 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 benefits of ADAS, and run PRISM with other existing scheduling mechanisms as well as ADAS for comparison.

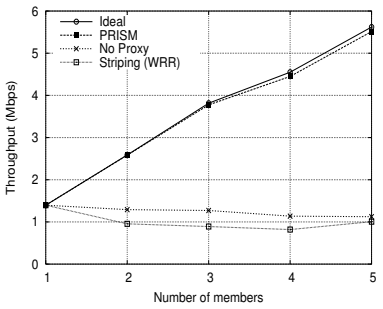


Figure 7: Bandwidth aggregation in the mobile community with different community sizes

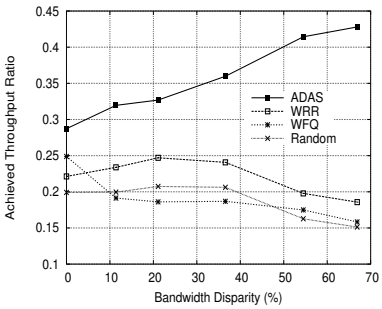


Figure 8: PRISM performance comparison in B/W disparity for different scheduling schemes without RPC.

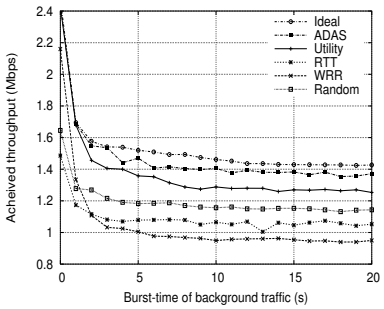


Figure 9: PRISM performance comparison under rate/delay fluctuations for different scheduling schemes.

ADAS reduces out-of-order packet deliveries by sensing bandwidth disparity, and improves links utilization. Figure 8 shows the performance gains on reducing the need for packet reordering under various scheduling mechanisms. The x axis represents the bandwidth disparity, and the y axis is an achieved throughput which is normalized by the ideal total bandwidth of WWAN links. Although the maximum ratio is below the half of ideal bandwidth due to the absence of RPC, the figure shows that the throughput by ADAS improves as the bandwidth disparity increases by selectively using high-bandwidth link bandwidth, meaning that ADAS reduces out-of-order deliveries.

On the other hand, since other scheduling mechanisms such as WRR and RR blindly assign packets to all available links without being aware of bandwidth disparity, their performance is degraded by the use of low-bandwidth links, which causes significant out-of-order deliveries.

8.3.2 Rate/delay fluctuation

We also evaluated ADAS's adaptivity to rate/delay fluctuations by examining end-to-end throughput given dynamic background traffic. Having three community members (whose WWAN bandwidths are 600, 900 and 1200 Kbps, respectively), we run one PRISM flow and two On/Off background traffic (one to the first member's WWAN link with 400 Kbps, and the other to the third with 800 Kbps). We use a burst-time of On/Off traffic as a parameter of rate/delay fluctuation with a Pareto distribution and a fixed idle-time (1s). At this time, we enable RPC functions to show the overall performance improvement.

ADAS adapts to the rate/delay fluctuation of WWAN links and reduces the need for packet reordering. As we will see in Section 8.4.2, reduced out-of-order packet deliveries makes an end-to-end throughput improvement, so we measured the throughput achieved by several scheduling algorithms while increasing rate/delay fluctuations. As shown in Figure 9, ADAS outperforms the other scheduling mechanisms by 12% to 47% in the presence of maximum background traffic.

On the other hand, WRR performs worse than random scheduling in the presence of large fluctuations. $hRTT$-only scheduling exhibits worse performance than the others because the fast but low-bandwidth link limits the overall performance by dropping most of packets. A utility-only scheduling algorithm provides similar performance to ADAS under stable links state. However, as the rate/delay fluctuates more, the U-only scheduling becomes less responsive to short-term fluctuations than ADAS which adapts itself to the fluctuations by using RTT, and thus achieves only 88% of ADAS's throughput.

8.4 Maximizing Network Utilization

8.4.1 Performance gains by RPC

We evaluated the RPC's benefits in network utilization. We use the same setting as in the bandwidth disparity experiment, and for better comparison, we compare three cases: PRISM without RPC, PRISM with only ACK resequencing (partial RPC), and PRISM with full RPC (including loss detection and fast loss recovery).

RPC achieves maximum network utilization by which PRISM can deliver almost ideal aggregated bandwidth. Figure 11 shows the performance gains achieved by RPC. PRISM with the full RPC indeed achieves maximum network utilization even in the presence of large bandwidth disparities. On the other hand, PRISM's performance without RPC shows less than 50% of ideal bandwidth. PRISM with a partial RPC yields, on average, only a 50% performance improvement since it should depend only on timeouts for packet-loss recovery.

8.4.2 Minimizing traffic burstiness

We evaluated the ADAS's contribution in 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 with ADAS. We run PRISM with WRR for comparison.

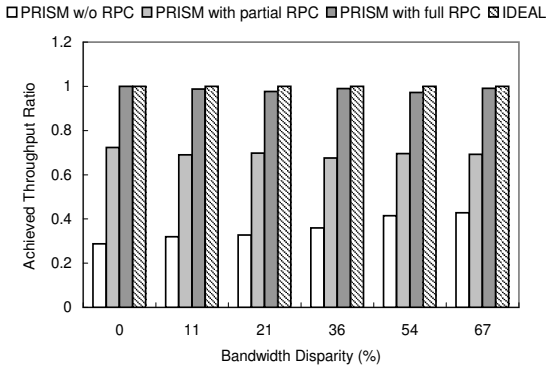


Figure 11: Performance gains by RPC

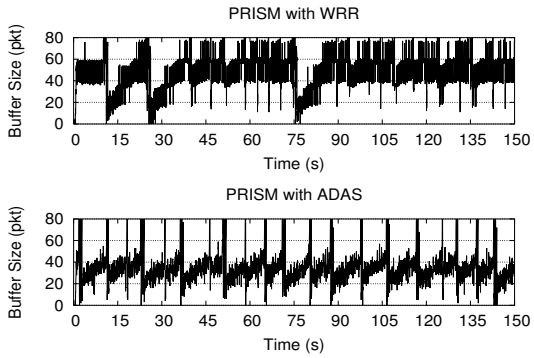


Figure 12: Re-sequencing buffer size progression

ADAS reduces traffic burstiness by minimizing out-of-order deliveries, and thus improves overall network utilization. Figure 12 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 a reduced chance for 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 up to 16% over the throughput (2.5 Mpbs) for bursty traffic (scheduled by WRR).

9 Related Work

Bandwidth aggregation in multi-homed mobile hosts is considered by several researchers. pTCP [12] and R^{2}CP [11] are transport-layer approaches to achieving aggregated bandwidth. They make a transport protocol have multiple states so that the transport layer can open multiple connections through multiple interfaces in a *single* mobile host. MOPED [8] is a framework to enable group mobility such that a *single* user's set of personal devices appear as a single mobile entity connected to the Internet.

Packet reordering is a major problem in multi-path routing environments. DSACK [22] in TCP is a detection mechanism of spurious retransmissions on packet reordering based on the information from a receiver via DSACK block. TCP-Door [20] is another scheme for detecting packet reordering in a MANET environment. This approach uses additional information, called *TCP packet sequence number*, to detect out-of-order packets. However, these mechanisms can solve occasional (but not persistent) out-of-order packet arrivals. TCP-PR [6] addresses this problem using a timeout as an indication of packet loss instead of duplicate ACKs, but it may still suffer from false timeouts that result from large RTT fluctuations.

Scheduling packets across multiple links is a well-known problem, and there are three approaches: Round-Robin (RR), fair-queuing, and hybrid. First, the RR scheduling guarantees long-term fairness, and is of low complexity [4]. However, RR inherently causes traffic burstiness which may require a large re-sequencing buffer. Second, the fair queuing attempts to approximate the Generalized Process Sharing (GPS) to achieve fairness (e.g., PGPS, WFQ, WF^{2}Q). However, these approaches assume that the exact bandwidths of each input and output link are known, which is expensive for resource-limited mobile hosts to obtain. Finally, a hybrid approach (e.g., [16]) removes the complexity of the fair-queuing approach, but it also assumes the known/fixed service rate.

10 Discussion and Conclusion

10.1 Discussion

PRISM can easily support upstream traffic (from a mobile host to a server) 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 4, 5, and 6, 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. Quantifying this benefits is part of our future work.

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 [18], 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 [7] to discourage such behaviors.

10.2 Concluding Remarks

In this paper, we first demonstrated 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.

Acknowledgement

The authors would like to thank Jack Brassil, Sung-Ju Lee, and Puneet Sharma of HP Laboratories for introducing the concept of mobile community. The work reported in this paper was supported in part by AFOSR under Grant No. F49620-00-1-0327.

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] 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.

[6] 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.

[7] 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.

[8] C. Carter and R. Kravets. User device cooperating to support resource aggregation. In *Proceedings of the 4th IEEE WMCSA*, Callicoon, NY, June 2002.

[9] J. Duncanson. Inverse multiplexing. *IEEE Communications Magazine*, 32(4), Apr. 1994.

[10] D. Farinacci, S. Hanks, D. Meyer, and P. Traina. Generic routing encapsulation (GRE). Internet Request for Comments 2784 (rfc2784.txt), Mar. 2000.

[11] 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.

[12] 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.

[13] L. Magalhaes and R. Kravets. MMTP:multimedia multiplexing transport protocol. In *Proceedings of SigComm-LA*, San Jose, Costa Rica, Apr. 2001.

[14] 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.

[15] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow. TCP selective acknowledgement options. Internet Request for Comments 2018 (rfc2018.txt), Oct. 1996.

[16] 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.

[17] 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.

[18] 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.

[19] J. Veizades, E. Guttman, C. Perkins, and S. Kaplan. Service location protocol. Internet Request for Comments 2165 (rfc2165.txt), June 1997.

[20] 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.

[21] J. Widmer. Network simulations for a mobile network architecture for vehicles. http://www.icsi.berkeley.edu/widmer/mnav/ns-extension.

[22] 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.

[23] 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.

Notes

[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.

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

[3] We use the on-line resource of [17].

[4] 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.