A Receiver-Centric Transport Protocol for Mobile Hosts with Heterogeneous Wireless Interfaces [*]

Hung-Yun Hsieh, Kyu-Han Kim, Yujie Zhu, and Raghupathy Sivakumar
School of Electrical and Computer Engineering
Georgia Institute of Technology
Atlanta, GA 30332, USA
{hyhsieh, zhuyujie, siva}@ece.gatech.edu

ABSTRACT

Numerous transport protocols have been proposed in related work for use by mobile hosts over wireless environments. A common theme among the design of such protocols is that they specifically address the distinct characteristics of the last-hop wireless link, such as random wireless errors, round-trip time variations, blackouts, handoffs, etc. In this paper, we argue that due to the defining role played by the wireless link on a connection's performance, locating the intelligence of a transport protocol at the mobile host that is adjacent to the wireless link can result in distinct performance advantages. To this end, we present a receiver-centric transport protocol called RCP (Reception Control Protocol) that is a TCP clone in its general behavior, but allows for better congestion control, loss recovery, and power management mechanisms compared to sender-centric approaches. More importantly, in the context of recent trends where mobile hosts are increasingly being equipped with multiple interfaces providing access to heterogeneous wireless networks, we show that a receiver-centric protocol such as RCP can enable a powerful and comprehensive transport layer solution for such multi-homed hosts. Specifically, we describe how RCP can be used to provide: (i) a scalable solution to support interface specific congestion control for a single active connection; (ii) seamless server migration capability during handoffs; and (iii) effective bandwidth aggregation when receiving data through multiple interfaces, either from one server, or from multiple replicated servers. We use both packet level simulations, and real Internet experiments to evaluate the proposed protocol.

Categories and Subject Descriptors

C.2.2 [**Computer-Communication Networks**]: Network Protocols

General Terms

Algorithms, Design, Performance

[*]This work was funded in part by NSF grants ANI-0117840 and ECS-0225497, Yamacraw, and Georgia Tech Broadband Institute.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
MobiCom'03, September 14–19, 2003, San Diego, California, USA.
Copyright 2003 ACM 1-58113-753-2/03/0009 ...$5.00.

Keywords

Heterogeneous wireless networks, multi-homed mobile host, seamless handoff, server migration, bandwidth aggregation

1. INTRODUCTION

The TCP (Transmission Control Protocol) transport layer protocol is a sender-centric protocol with the data sender performing all important tasks including congestion control and reliability. The receiver participates in the operation of the protocol, but contributes only by sending feedback in the form of acknowledgments. While numerous TCP variants and alternatives have been proposed for mobile hosts operating in a wireless environment, all such protocols still retain the sender-centric nature of TCP [2, 3, 11, 14, 30]. Although the role of the receiver is appreciably larger in some of the above protocols than in TCP, it is still limited to providing more meaningful feedback, with the sender having final control over all key tasks.

In this paper, we make the case for a receiver-centric transport layer protocol[1] for wireless packet data networks. While we present in-depth discussions in Section 2, our arguments are based on the following two factors:

1. Any transport protocol tailored for mobile hosts in a wireless environment has to tackle the unique characteristics of the "last-hop" wireless link, and the consequences of the end-point being mobile. In fact, the common theme between the wide variety of transport protocols proposed for different wireless environments, is indeed the notion of addressing the problems induced by the wireless last-hop. Despite the wireless-aware behavior of these transport protocols, the congestion control and reliability mechanisms of the connection are still predominantly controlled by the sender, a remote host in the backbone network. However, we argue that placing the transport protocol's intelligence at the mobile host, which is an end-point of the wireless link, can enable fundamentally smarter mechanisms for congestion control, loss recovery, and power management when compared to sender-centric approaches.

2. With the myriad of wireless networking technologies evolving to provide ubiquitous communication, a mobile user today has several options for wireless Internet access. Not

[1]In consideration of the typically prevalent server-client nature of traffic in the Internet, we term the protocol "receiver-centric" although precisely it is the mobile host that drives the protocol operation. Note that in this paper, we define the sender and the receiver of a connection with respect to the direction of the data flow.

surprisingly, mobile hosts are increasingly becoming multi-homed, possessing two or more interfaces. The distinct advantages offered by the different technologies further spur the need for mobile hosts to have multiple interfaces. For example, wireless LANs offer high bandwidths but suffer from low coverage areas, while wireless WANs offer larger coverage areas but cannot support as high bandwidths as their LAN counterparts. Thus mobile hosts are, or can increasingly be expected to be, equipped with heterogeneous wireless interfaces providing access to wireless networks that can potentially belong to even different autonomous domains. We argue that a receiver-centric transport protocol, where the receiver controls *how much* and *which* data to receive from the sender, will be able to fare better than traditional sender-centric approaches in addressing this heterogeneity at the receiver, and provide distinct advantages from the perspective of the transport layer functionality achievable.

To understand the above factors better, in Section 2, we provide detailed arguments for the specific benefits enabled when using a receiver-centric transport protocol for mobile hosts in a wireless environment.

We then propose a receiver-centric transport protocol called **RCP** (Reception Control Protocol), which is a TCP clone in its general behavior. RCP is TCP-friendly in its operations, but enables smarter transport layer mechanisms for congestion control, loss recovery, and power management. Briefly, the receiver in RCP controls all the key functionalities of the protocol including congestion control, flow control, and reliability, while the sender's role is minimized to that of responding to the receiver's directions. We evaluate RCP both to demonstrate its TCP-friendliness, and to highlight its unique benefits when compared to sender-centric transport protocols. We also show why the transposition of the key functionality to make the protocol receiver-centric, does not impose any appreciable increase in the CPU and energy consumption at the resource-constrained mobile host. We provide details of the RCP design, protocol, and its evaluation results in Section 3. While several protocols have been proposed with increased receiver participation [8, 12, 24, 30, 33, 38], to the best of our knowledge, this is the first effort to systematically investigate the benefits achievable using receiver-centric protocols in wireless networks.

Finally, we propose a purely receiver-only extension to RCP called **R^{2}CP** (Radial RCP), designed specifically for multi-homed mobile hosts. R^{2}CP is a multi-state transport protocol that effectively aggregates multiple RCP connections into one abstract connection for the higher layer application. An R^{2}CP connection has multiple independent RCP senders communicating with their corresponding RCP receivers, with the receivers coordinated by R^{2}CP. R^{2}CP facilitates several important transport layer functionalities for multi-homed mobile hosts with heterogeneous wireless interfaces including: (i) seamless handoffs, (ii) server migration, and (iii) effective bandwidth aggregation. We describe the different functionalities, and how R^{2}CP achieves them in Section 4. We also discuss why these functionalities cannot be supported, or not effectively supported, by sender-centric approaches.

The rest of the paper is organized as follows: In Section 2 we motivate a receiver-centric approach for mobile hosts in a wireless environment. In Section 3 we present details of the RCP protocol, and its performance gains. In Section 4 we extend RCP to a multi-state protocol called R^{2}CP that provides functionality gains to mobile hosts with heterogeneous wireless interfaces. Section 5 discusses overheads when using a receiver-centric protocol at the mobile host, and several RCP extensions. Finally, Section 6 discusses related work, and Section 7 concludes the paper.

2. WHY RECEIVER CENTRIC?

In this section, we discuss the benefits of using a receiver-centric transport protocol for mobile hosts in a wireless environment. We focus on a scenario where mobile hosts act as receivers for data sent from servers in the backbone network, and hence we use the terms "receiver" and "mobile host" interchangeably in the following discussions. While we explain in Section 3 various protocol functionalities that can be moved from the sender to the receiver, for purposes of discussions in this section, we assume that a receiver-centric transport protocol controls *how much data can be sent*, and *which data should be sent*, by the sender. The sender merely acts based on the requests from the receiver. We first discuss the *performance gains* for a mobile host by dealing with the characteristics of the wireless last-hop, and then discuss the *functionality gains* when the mobile host is equipped with multiple heterogeneous wireless interfaces.

2.1 Tackling the Wireless Last-Hop

2.1.1 Loss Recovery

TCP assumes that all losses are due to congestion, and hence it invokes its congestion control mechanisms when recovering from losses. In the presence of non-congestion-related losses introduced by wireless links such as channel errors, delay variations, blackouts, and handoffs, TCP suffers from performance degradation due to unnecessary window cutdowns. Hence, many approaches proposed to improve the performance of TCP in wireless environments have focused on providing TCP with information about the characteristics of the wireless link for it to distinguish the causes of losses and take appropriate actions. The information can be in the form of loss classification (whether a loss is due to congestion or corruption), RTT sample filtering (excluding RTT samples adversely inflated due to link retransmissions), channel states or potential link outages (handoffs or blackouts), etc [2–4, 11].

Since the mobile host is adjacent to the wireless last-hop, it is obviously better equipped to obtain first-hand knowledge of the above pieces of information. In TCP, since the loss recovery (including loss detection) is performed at the sender, the mobile host needs to convey the requisite information to the server for it to take "wireless-aware" actions. While this model of operation has predominantly been adopted in related work, it has some key limitations: (i) Providing feedback to the sender incurs a finite overhead in terms of the throughput consumed on the reverse path. This can translate into degraded performance for connections, especially when the forward and reverse traffic shares the same bottleneck channel (as is the case for the wireless last-hop), or when the feedback is lost. (ii) Providing all available information as feedback within a limited transport protocol framework can be unwieldy to achieve. For example, some mobile hosts might use a reliable link layer that affects the round-trip time of the connection, and hence might choose to feedback information to filter specific RTT samples. Other mobile hosts might have an unreliable link layer, but can provide feedback information about the reasons for losses (random or congestion-related). If transport protocol headers need to be changed to accommodate such information, how can the changes be made generic enough to accommodate any possible feedback information? (iii) Along the same vein, how can a sender be designed generically to operate with potentially a wide variety of such types of feedback coming from mobile hosts that use any arbitrary link layer protocol?

A receiver-centric transport protocol that performs loss recovery at the receiver, however, can *avoid the feedback overheads and latency*, and be responsive to the dynamics of the wireless link using the information obtained locally. Moreover, while any intelligence added to sender-centric approaches requires changing both the backbone server (for reaction) and the mobile host (for feedback), a receiver-centric approach involves *changing only the mobile host*. The backbone server that is not in charge of loss recovery does not need to be aware of the characteristics of the wireless link.

2.1.2 *Congestion Control*

The congestion control mechanism that TCP uses is designed for wired environments, without taking into consideration the characteristics of wireless environments. Related work that aims to achieve optimal performance in various wireless environments has proposed different congestion control mechanisms tailored to the characteristics of the specific target environment [14,21,30,38]. For example, WTCP [30] has been proposed for wireless WANs with very low bandwidths and reverse path congestion, while STP [14] has been proposed for satellite networks with highly asymmetric links and long propagation delays.

To achieve optimal performance, a mobile host should ideally use the congestion control mechanism (or transport protocol) designed for the specific wireless network it has access to. However, in sender-centric approaches, these network specific congestion control mechanisms need to be implemented at the backbone server. While it is conceivable that a mobile host has access to only a very limited number of wireless networks, a backbone server may need to support a significantly large amount of connections from mobile hosts belonging to any arbitrary wireless network. Given the increased heterogeneity of the wireless networks, the disadvantages of sender-centric approaches is pronounced in terms of its lack of deployability. *Not only is it infeasible for the server to implement all possible congestion control mechanisms designed for various wireless environments, but it is unscalable to require the server to change its protocol stack whenever a new congestion control mechanism optimized to a new wireless access technology is introduced*.

A receiver-centric protocol where the receiver is responsible for congestion control thus has unique advantages over a sender-centric one. Since the sender is not tasked with implementing the congestion control mechanism of the connection, its functionality can be significantly simplified and made transparent to the specific congestion control mechanisms used at the receiver.

2.1.3 *Power Management*

While a majority of work on the performance of TCP has focused on the throughput achievable, recently the energy efficiency of TCP has also gained attention [29,37,40]. It is shown in [37,40] that since channel errors tend to be bursty (correlated), it is energy-conserving to cut down the window size (and hence reduce the number of packets in flight) when wireless losses are detected. This is because packets retransmitted immediately after wireless losses are likely to be lost again, thus wasting the energy. While TCP-SACK achieves better throughput performance compared to other TCP variants, in fact it is the least energy-conserving protocol of all when the channel error rate is high [29].

Therefore, an energy-efficient transport protocol should avoid persistently accessing the channel when the channel condition is hostile, as energy consumed during this period for attempting to transmit or receive packets is likely to be wasted. Instead, it should adjust the retransmission policy according to the channel dynamics. While it is possible to implement such power management in a sender-centric transport protocol like TCP, there are several limitations to this approach: (i) While the receiver is more aware of the channel condition than the sender, any power-saving decision

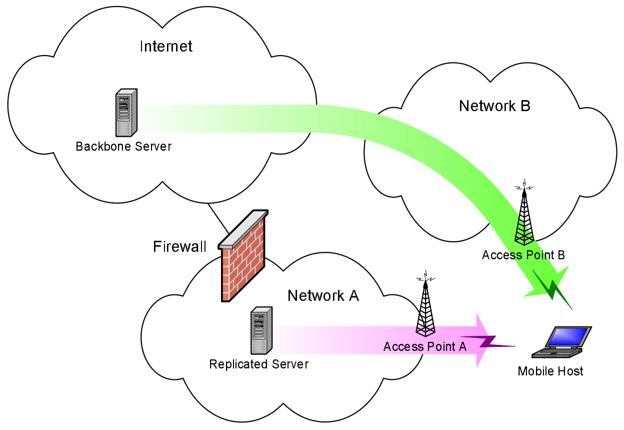


Figure 1: A Mobile Host with Multiple Wireless Interfaces

cannot be made locally at the receiver. This is because the sender is responsible for congestion control and loss recovery, and hence any "unexpected" prolonged delay incurred at the receiver (that decides to refrain from accessing the channel until the channel condition is more favorable) in receiving data packets or transmitting *ACKs* can easily cause the sender to timeout or wrongly inflate its RTT estimation. (ii) Even if the receiver decides to inform the sender of the power-saving decision, the feedback information will suffer from the same problems that we discussed in Section 2.1.1. More importantly, packets transmitted for conveying such feedback information incur extra energy consumption – especially if the channel condition is bad such that multiple retransmissions are required. The overheads incurred in sending the feedback information hence limit the granularity and effectiveness of any sender-centric power management scheme.

On the other hand, in a receiver-centric protocol the receiver decides which and how much data it needs to receive, and the sender merely responds based on the receiver's direction. Efficient power-conserving decisions can be made at the receiver without triggering any adverse reaction at the sender. Hence, the receiver has a higher degree of flexibility to control the transmission or retransmission decisions, without involving the sender.

2.2 Supporting Heterogeneous Interfaces

The primary reason for a mobile host to be equipped with heterogeneous wireless interfaces is the performance tradeoffs that different access technologies exhibit, in terms of mobility support, coverage area, network capacity, and transmission power. The availability of heterogeneous interfaces, however, has given rise to new challenges to existing transport protocols in terms of the functionalities they provide. In the following, we discuss the *functionality gains* that a receiver-centric transport protocol can achieve to leverage the existence of multiple interfaces at the mobile host.

2.2.1 *Seamless Handoffs*

When the coverage areas of different access technologies overlap, it is possible to achieve seamless handoffs at the link layer. However, such link layer handoffs do not necessarily translate into seamless handoffs at the transport layer. Specifically, when a mobile host handoffs from one interface to another with an IP address change handled by Mobile IP, the prolonged delay for registration with the home agent [27] can potentially introduce packet losses after the link layer handoff has completed. To prevent TCP from having adverse reactions due to packet losses during handoffs, the mobile host needs to inform the sender of the handoff decision. As we discussed in Section 2.1.1, whenever feedback information is required, a receiver-centric protocol has advantages over a sender-centric one due to the locality of information needed.

However, while it is possible to *freeze* TCP during handoffs [11], such a stall causes connection disruption and prevents users from enjoying seamless handoffs. One solution to avoid the handoff latency without relying on infrastructure support [9], is to use a mobility-enabled transport protocol for achieving end-to-end host mobility [26]. When the mobile host decides to perform a vertical handoff [34], it can create a new "data stream" for data transfer through the new address, as soon as the new interface becomes active. With an approach like [15], the mobile host can use multiple TCP pipes (streams) simultaneously without experiencing any connection stall as long as the link layer supports seamless handoffs.

A receiver-centric transport protocol thus has advantages over a sender-centric one in such a scenario, since the receiver can accurately control which and how much data to send through each pipe based on the status (say, signal strength) of each interface. Moreover, as we discussed in Section 2.1.2, when the receiver decides to switch to another interface specific congestion control mechanism after handoffs, such decision does not need to involve the sender, which otherwise would be tasked with, in addition to supporting a plethora of congestion control mechanisms, the seamless transition from one congestion control mechanism to another for a live connection.

2.2.2 Server Migration

Server migration is necessary for achieving service continuity when a mobile host handoffs from one network to another, and fails to connect to the original server using the new network address. For example, consider a mobile host with both WWAN (Network A) and WLAN (Network B) interfaces as shown in Figure 1. When it initially uses the WWAN interface to connect to the E! Online server, it is provided access to the proxy server inside the WWAN that mirrors the same content (e.g. consider a WWAN service provider such as EarthLink that teams with Akamai for improving the network service it provides [1]). When the mobile host moves to within the coverage area of the WLAN and undergoes a vertical handoff, it cannot connect to the original proxy server in the WWAN (due to, say, firewalls). However, it may have a connection to a different server through the WLAN interface, and hence can initiate server migration to enjoy service continuity.

Server migration may require support from the application [35] or the transport protocol [31], to synchronize the states between servers. A well-designed transport layer protocol can facilitate such a synchronization process. If a sender-centric transport protocol is used for server migration, states maintained at the server for performing congestion control and loss recovery need to be transferred from one server to another. Moreover, for TCP, after the new server assumes control of the connection, the mobile host has to flush any out-of-order data from its resequencing buffer, to avoid confounding the server by acknowledging data that the sender has not yet sent [31]. As much as a window's worth of data buffered at the receiver needs to be flushed after server migration.

On the other hand, in a receiver-centric transport protocol, since the states maintained for protocol operations are biased toward the receiver, overheads incurred in transferring protocol states from one sender to another are minimized. Moreover, since the receiver has access to the receive buffer and has control over which data to receive from the sender, there is no need to flush the buffer after migration. The receiver can simply request every "hole" in the receive buffer from the new sender.

2.2.3 Bandwidth Aggregation

When the coverage areas of different wireless networks a mobile host has access to, overlap, the mobile host can use multiple interfaces simultaneously, with the goal of enjoying the aggregate bandwidth available.

While approaches for achieving bandwidth aggregation on multi-homed mobile hosts using sender-centric transport protocols have been proposed [15, 20], they are limited to using only one server. Such point-to-point bandwidth aggregation, however, might not be possible or desirable in some cases. For example, as we discussed in Section 2.2.2, some proxy servers are accessible only through the designated wireless interface, and hence it is not possible to achieve bandwidth aggregation using additional wireless interfaces that have no access to the existing server. In such a scenario, a sender-centric approach would not be desirable since it would otherwise require explicit coordination between these geographically-spaced servers. Moreover, it is possible that mobile hosts want to leverage the existence of multiple active interfaces in an *opportunistic* fashion, based on the tradeoffs between different interfaces in terms of achieved throughput, power consumption, and the cost incurred. The decision to use or shut down an active interface thus can be dynamic, based on the channel conditions (e.g. loss rates and delays) and the policy of bandwidth aggregation that the mobile host desires.

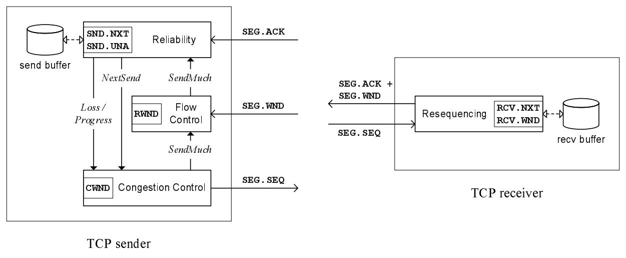
It is advantageous to use a receiver-centric transport protocol for achieving different instantiations of bandwidth aggregation. For multipoint-to-point bandwidth aggregation, since the receiver is the center of control, it can easily coordinate the transmission of multiple senders internally, without any explicit coordination between senders themselves. For policy-based bandwidth aggregation, any policy can be easily implemented and updated at the receiver based on the characteristics of the last-hop and the preference of the user.

3. RCP: RECEPTION CONTROL PROTOCOL

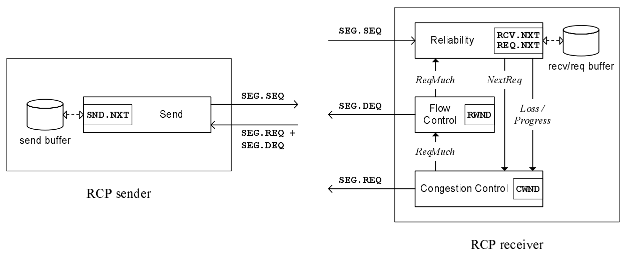
We now present details of a receiver-centric transport protocol called RCP. Briefly, RCP moves the responsibility for performing reliability and congestion control from the sender to the receiver. We first give a short review of the sender-receiver interaction in TCP, and how it is transposed in RCP. We then give an overview of the protocol operation in RCP, and present different protocol functionalities including connection management, congestion control, flow control, and reliability. Finally, we use simulation results to show that while RCP is indeed TCP-friendly, it achieves better performance in wireless environments in terms of intelligent loss recovery, scalable congestion control, and efficient power management.

3.1 Transposition of Functionalities

TCP is a connection-oriented transport layer protocol that provides reliable in-sequence data delivery to the application. Its protocol operation mainly consists of the following four functionalities: connection management, flow control, congestion control, and reliability. Figure 2(a) shows a schematic view of the sender–receiver interaction in TCP, along with several state variables using the notation introduced in [25]. The connection management is required by any connection-oriented protocol to synchronize connection states between the communicating peers. After the connection is established, the sender in TCP controls the progress of data transfer. The sender drains data from its buffer based on the amount of data that the receiver can accept (flow control), and the amount of data that the network can sustain (congestion control). The receiver



(a) TCP (Sender-Centric)



(b) RCP (Receiver-Centric)

Figure 2: Sender–Receiver Interactions

performs resequencing and acknowledges data received. Reliable data transfer is achieved through loss detection and loss recovery performed at the sender.

It is clear that the connection management cannot be implemented only at one side of the connection, but needs participation of both the sender and the receiver. For the other functionalities, while TCP uses a sender-centric approach, RCP delegates the responsibility to the receiver as shown in Figure 2(b). Briefly, while the receiver in TCP merely sends back *ACKs* with no control over which and in what sequence data is transmitted by the sender, in RCP the receiver explicitly controls these factors and the reliable delivery of data. Moreover, the RCP receiver also assumes total control over the bandwidth the connection can consume, using the same window based algorithm employed by the TCP sender. Finally, although flow control in TCP involves the sender, it is performed solely by the receiver in RCP. Therefore, the receiver in RCP determines *how much data* the sender can send (via congestion control and flow control), and *which data* the sender should send (via reliability).

3.2 Overview

In RCP, since the control of data transfer is shifted from the sender to the receiver, the *DATA–ACK* style of handshaking in TCP is no longer applicable. Instead, to mimic the self-clocking characteristics of TCP, RCP uses the *REQ–DATA* handshake for data transfer, where any data transferred from the sender is preceded with an explicit request (*REQ*) from the receiver. Equivalently, RCP uses the incoming data to *clock* the request for new data. The sender simply maintains the send buffer with one pointer (`SND.NXT`) indicating the maximum sequence number sent thus far.

After the connection is established, the receiver requests data from the sender based on the size of the initial congestion window. The progression of its congestion window follows the slow start, congestion avoidance, fast retransmit, and fast recovery phases just like in TCP. The key difference in the operation is that any trigger for performing congestion control is inferred based on the arrival (or non-arrival) of *data segments*. For example, a loss is inferred upon the arrivals of three out-of-order data segments – instead of *ACKs*. Upon detection of a segment loss, RCP cuts down its congestion window, and retransmits the corresponding *REQ* asking for the lost segment. Finally, the receiver performs data resequencing, and gives in-sequence data to the application.

3.3 Protocol

In the following, we present details of the RCP protocol in terms of the *REQ–DATA* handshake, and different functionalities including connection management, congestion control, flow control, and reliability. For simplicity of explanation, we assume a backlogged (network-limited) traffic source in the following discussions. We revisit the implications of other traffic types in Section 5.

3.3.1 REQ–DATA Handshake

In the *DATA–ACK* handshake, TCP uses the cumulative acknowledgment for achieving robustness to losses. To emulate this behavior and tolerate loss in the reverse path, RCP allows the receiver to send request either in a *cumulative* mode or in a *pull* mode, by appropriately setting the pull flag (`PUL`) in the packet header. The receiver by default uses the cumulative mode to requests for new data, and uses the pull mode only for retransmission of requests. When the sender receives a request with the pull flag set, it sends only the data segment indicated in the packet header. Otherwise, the sender cumulatively transmits data from `SND.NXT` that has not been sent yet. Hence, the loss of *REQ* in cumulative mode has similar impact to that of *ACK* loss in TCP. To protect *REQ* in the pull mode from losses, RCP also uses a similar mechanism used by TCP for protecting SACK from losses. The receiver puts the most recent blocks of sequence numbers (we use three blocks as proposed in the SACK option [23]) it requested in the *REQ* header. The sender, in addition to maintaining the send buffer, also maintains a cyclic buffer consisting of the most recent blocks of sequence numbers (three blocks) it sent out. Upon receiving the request from the receiver, the sender checks the consistency between the blocks in *REQ* and its cyclic buffer. Any mismatch is an indication of *REQ* losses, and will be recovered by the sender. Note that a request in the pull mode for a specific data segment will be carried in at least four *REQs*. We revisit the robustness of *REQ* in Section 3.4.1.

3.3.2 Connection Management

Just like in TCP, either the RCP sender or the receiver can initiate the connection setup. The setup process consists of the same *SYN – SYN+ACK – ACK* handshake as in TCP. However, once the connection is established, instead of the sender sending the first data segment, the RCP receiver transmits the first *REQ* with the initial sequence number. The sender then transmits the first data segment upon receiving the *REQ*. The connection teardown in RCP also follows that in TCP.

3.3.3 Congestion Control

In RCP, the receiver performs congestion control and maintains the congestion control parameters including the congestion window `CWND` and round-trip time information. Since RCP is a TCP clone, it adopts the window based congestion control used in TCP. The slow start, congestion avoidance, fast retransmit, and fast recovery phases are triggered and exited in the same fashion as in TCP. Note that while the same window adaptation algorithm (additive increase, multiplicative decrease) can be implemented either at the sender or at the receiver for performing congestion control, the semantics of the congestion window and the trigger for window increase or cutdown are different. In TCP, the size of the congestion window limits the amount of unacknowledged *DATA* in the network, and the sender uses the return of *ACKs* to trigger the progression of the congestion window. In RCP, the size of the congestion window limits the amount of outstanding *REQs* in the network, and the receiver uses the return of *DATA* to trigger the progression of the congestion window.

3.3.4 Flow Control

Flow control allows the receiver to limit the amount of in-transit data to the available buffer space at the receiver – when waiting for the application to read (and purge) in-sequence data, or waiting for the arrivals of out-of-order data. In RCP, a request is sent out only if the corresponding data, once received, does not cause buffer overflow at the receiver. This can be achieved by creating a "dummy" `sk_buff` [6] (that does not contain any data) in the receive buffer for each data segment requested. New requests are issued as long as new space is created in the buffer. Note that the sender in TCP relies on the window advertisement from the receiver to perform flow control. However, in RCP since the receiver maintains the receive buffer, and has total control over how much data the sender can send, flow control is internal to the receiver. Interestingly, RCP also needs a window field (`SEG.DEQ`) in the packet header to inform the sender of the highest in-sequence data received so far (which can be calculated at the sender using `SEG.REQ - SEG.DEQ`), thus allowing the sender to purge such data from its send buffer. The window scale option [17] used in TCP can also be applied to RCP in the same fashion.

3.3.5 Reliability

As Figure 2(b) shows, in RCP the resequencing and reliability functionalities are collocated at the receiver. Upon receiving a data segment from the sender, the receiver enqueues the data in the corresponding `sk_buff` (created when its request was transmitted), and updates `RCV.NXT` after the resequencing process. In TCP, since reliability is performed at the sender while resequencing is performed at the receiver, `RCV.NXT` is conveyed as the cumulative ACK to the sender for it to perform loss detection. However, `RCV.NXT` conveys limited information about the state of the receive buffer, and hence early implementations of TCP that rely on the cumulative ACK for performing loss detection, suffer from recovering at most one loss per round-trip time, in addition to incurring frequent timeouts [10]. The SACK option is proposed to address this limitation, using which the TCP sender aims to construct the bitmap of the receive buffer in the "scoreboard" data structure [23]. However, in RCP the receiver has direct access to the receive buffer, and hence it can timely and accurately perform loss detection and loss recovery without relying on the use of SACK.

While RCP can use any loss recovery algorithm optimized to the wireless environment, in the current implementation of RCP we adopt the algorithms proposed in [5, 22]. Briefly, (i) the same threshold in terms of the number of out-of-order arrivals is used for detecting all holes (not just the first one) in the receive buffer; (ii) RCP does not incur a timeout when a retransmitted segment is lost; and (iii) there is no need to clear the receive buffer upon a timeout (whereas a TCP sender using SACK should clear its scoreboard due to the possibility of receiver reneging [23]).

3.4 Performance Gains

In this section, we show through simulations the performance gains of a receiver-centric transport protocol over a sender-centric one. We first show that RCP is indeed a TCP clone and is friendly to TCP in the wired environment. We then show that in the wireless environment, RCP achieves better performance than TCP in terms of loss recovery, congestion control, and power management.

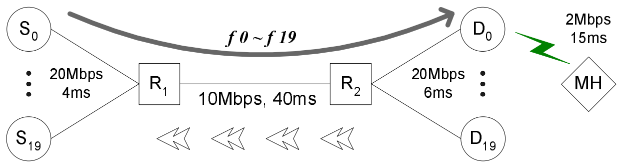


Figure 3: Network Topology

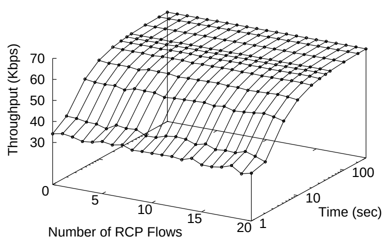
We use the *ns-2* network simulator [36] and a dumb-bell network topology shown in Figure 3 for performance evaluation. For fair comparisons between sender-centric and receiver-centric protocols, we modify the implementation of TCP-SACK in *ns-2* to include better loss recovery mechanism such as better estimation of the pipe size [22], and prevention of timeouts due to lost retransmissions (refer to Section 3.3.5). Unless otherwise specified, each data point in the figures is an average of 10 samples using random seeds, and each sample is run for 300s.

3.4.1 TCP Friendliness

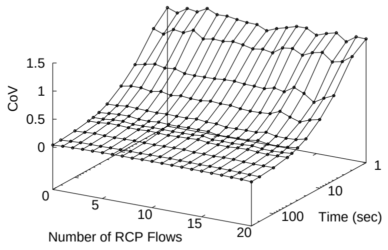
We introduce 20 flows between $S_{0...19}$ and $D_{0...19}$ with different proportions of TCP and RCP flows, to study the impact of RCP on existing TCP flows. We vary the number of RCP flows from 0 to 20 (the others are TCP flows), and observe for each scenario the short-term and long-term behaviors of all flows in the network. As shown in Figure 4(a) and Figure 4(b), we plot the mean and coefficient of variation of the per-flow throughput at different time instants after the simulation starts. The coefficient of variation (CoV) is obtained by dividing the standard deviation of the throughput by the mean throughput [13]. Note that for clarity of presentation, the direction of the time-axis in Figure 4(b) is reversed. We can find from both figures that the impact of introducing RCP flows on existing TCP flows in terms of the throughput re-distribution is minimal, as evident from the "flat" curve across different proportions of RCP flows. Specifically, since CoV is an index of unfairness in the network, it is clear from Figure 4(b) that TCP flows do not suffer from unfairness in the presence of RCP flows. (Otherwise, CoV would have increased with increasing number of RCP flows.)

To profile the robustness of the request mechanism used in RCP, we consider a scenario where there is significant loss in the reverse path. As indicated in Figure 3, we introduce 0 to 100 on/off UDP flows in the reverse direction of the bottleneck link to emulate the flash crowds (e.g. WWW-like traffic) in the Internet [13]. These on/off flows generate traffic based on the Pareto distribution, where the shape parameter is set to 1s, the mean idle time is set to 2s, the mean burst time is set to 1s, and the data rate during the burst period is set to 500Kbps. Such flash crowds introduce significant packet drops (to *ACK* or *REQ*) in the reverse path. For example, with 100 on/off traffic sources, each of the TCP (or RCP) flows experiences a packet drop rate of approximately 40% in the bottleneck link. We introduce 20 RCP flows in the forward path, and compare the per-flow throughput achieved against that of using 20 TCP flows. As Figure 4(c) shows, despite the heavy losses in the reverse path, the performance of RCP closely tracks that of TCP. This substantiates our argument made in Section 3.3.1 that the design of the cumulative request and the use of cyclic buffer help RCP tolerate losses in the reverse path.

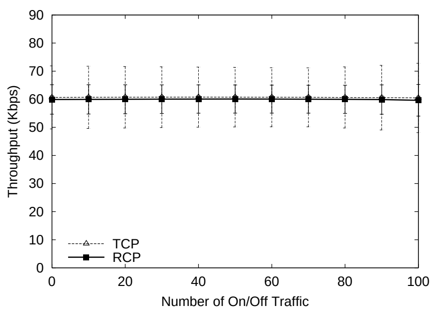
Thus far we have compared the performance of TCP and RCP only in a wired environment. In the following, we consider a mobile host with wireless access to the backbone network. As shown in Figure 3, the wireless link between the mobile host *MH* and the access point D_0 has a bandwidth of 2Mbps, and access delay of 15ms. It allows the mobile host to connect (using either TCP



(a) Mean Throughput



(b) CoV



(c) Reverse Path Loss

Figure 4: RCP is Friendly to TCP

or RCP) to the backlogged traffic source S_0 for, say, file download. We introduce random packet losses from 0.01% to 10% in the wireless link (in both directions), and use the achieved throughput and power consumption at the mobile host to compare the performance of the two protocols.

3.4.2 Intelligent Loss Recovery

Information from lower layers about the characteristics of the wireless link allows the transport layer protocol to identify the cause of losses, and hence to intelligently perform loss recovery. As we discussed in Section 2.1.1, a key motivation for using receiver-centric protocols at the mobile host is to avoid the feedback overheads and latency seen in sender-centric protocols, and to allow more flexible layer coordination without being limited by the format of the packet header. While it is not the focus of this paper to provide assorted instantiations of wireless-aware transport protocols leveraging such benefits, we use the following example to show the performance gain achievable when the information used for loss recovery is locally available.

Explicit loss notification (ELN) [3] has been proposed as a TCP option that allows TCP to distinguish wireless random losses from congestion losses. Mobile hosts, with or without the assistance from the base station [3, 4], keep track of packet drops due to wireless errors. When cumulative acknowledgments indicating the packet lost due to wireless errors are generated, the `ELN` flag in the packet header is set by the mobile host. Upon detecting a hole with the `ELN` flag set, the TCP sender retransmits the lost segment without cutting down its congestion window. As we can see in Figure 5(a), the performance of TCP improves substantially for loss rates between 0.2% and 2% when ELN is used. However, when the loss rate increases beyond 2%, the performance gain decreases rapidly. This is because the `ELN` bit allows the sender to identify only one wireless error, and hence when multiple wireless losses occur in one round-trip time, ELN fails to provide the sender with the necessary loss classification information, making the performance of TCP-ELN degrade to that of vanilla TCP.

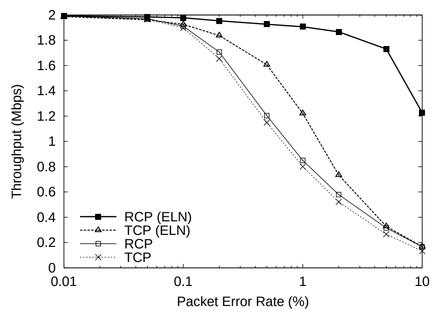
RCP with ELN, on the other hand, shows a much better performance even when the loss rate is high. The primary reason is that the wireless loss information maintained at the mobile host is directly accessible to RCP. Hence RCP has accurate information about the cause of losses for all holes in the receive buffer, which allows it to recover from each loss intelligently. While it is possible to couple SACK with ELN, and *redesign* the TCP packet header such that each *un-SACKed* segment has its own ELN flag, this approach in fact exposes the limitations of sender-centric protocols that we mentioned earlier in this section. We note from Figure 5(a) that even in the absence of ELN, RCP constantly achieves better performance than TCP, with the performance gain increasing as the packet error rate increases. The reason, as we discussed in Section 3.3.5, is because loss recovery and resequencing in RCP are collocated at the receiver. The effectiveness of the SACK blocks in helping the TCP sender construct the bitmap of the receive buffer, is impaired when both the data segments and *ACKs* suffer from high loss rates (recall that we introduce random losses in both directions of the wireless link).

3.4.3 Scalable Congestion Control

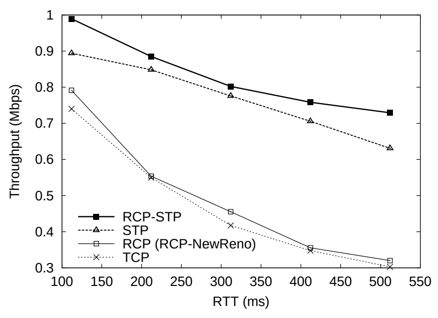
As we mentioned in Section 2.1.2, various congestion control mechanisms have been proposed for use with different wireless environments. Until a unified congestion control framework is available, to achieve optimal performance, a mobile host needs to use the congestion control mechanism designed for the specific wireless network it has access to. In sender-centric protocols, since congestion control is implemented at the sender, the backbone server is overloaded with supporting a plethora of congestion control mechanisms for all possible wireless networks mobile hosts might connect from. In this section, we present how a receiver-centric transport protocol like RCP can address this problem in a scalable way.

To start with, we consider a satellite environment with long propagation delay and highly asymmetric links, compared to the terrestrial wireless networks. The authors in [14] show that TCP (SACK) fares badly in such an environment. They propose a new transport protocol called STP (Satellite Transport Protocol) with improved performance. However, STP is a sender-centric protocol like TCP, and hence a mobile host using the satellite network to access the backbone server, cannot use STP unless it is implemented in the protocol stack of the concerned server. Now, by using the algorithm presented in [14], and the technique of functionality transposition discussed in Section 3.3, we transform STP into a receiver-centric protocol called RCP-STP. By virtue of the simple sender design in receiver-centric protocols, while STP uses a fundamentally different congestion control algorithm from TCP, the RCP-STP sender uses the same algorithm as the RCP sender. Hence, the backbone server as an RCP sender can communicate with any mobile host acting either as an RCP receiver, or as an RCP-STP receiver – depending on the access network the mobile host uses.

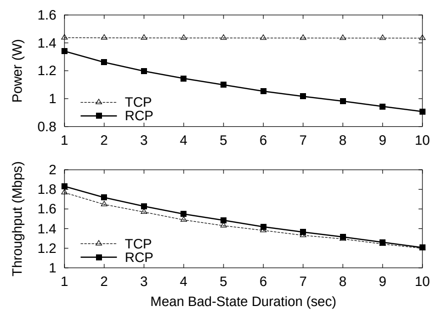
We use the same network topology and scenario used in [14] for evaluating the performance of STP, RCP-STP, TCP and RCP (RCP-NewReno). Briefly, the network topology resembles the dumb-bell topology in Figure 3 with the source (S_0) and destination (D_0) separated by a satellite link (link $R_1 - R_2$). The satellite link is the bottleneck link with a bandwidth of 1.5Mbps, and propagation de-



(a) Loss Recovery



(b) Congestion Control



(c) Power Management

Figure 5: Performance of RCP

lay ranging from 50ms to 250ms. Link asymmetry is emulated using backlogged TCP flows in the reverse direction. Four HTTP traffic sources are introduced in each direction to emulate the background traffic. As evident from Figure 5(b), the NewReno style of congestion control used in TCP (and RCP) does not perform well in the satellite environment, while STP and RCP-STP achieve a much better performance. We can make the following observations from the results: (i) the performance difference between TCP and STP substantiates the need to use network specific congestion control for achieving optimal performance; and (ii) the performance difference between STP and RCP-STP reinforces the benefits of receiver-centric protocols over sender-centric ones.

3.4.4 *Efficient Power Management*

We now show the performance of RCP in terms of facilitating power management at the mobile host. As we described in Section 2.1.3, when the channel condition is severe, it is not energy-efficient for a mobile host to persevere with persistent retransmissions. Since the mobile host is an end-point of the wireless last-hop, it is aware of the channel condition (via, say, measuring the signal strength in the received packets or beacons from the access point). Upon detecting a hostile channel state, the mobile host can save the battery power by reducing the amount of data in transit or refraining from transmissions. However, note that while significant energy savings can be achieved by operating the wireless interface card in the sleep mode, doing so without the sender being aware of such energy-conserving tactics may cause adverse reactions at the sender and cause performance degradation [19]. A receiver-centric protocol such as RCP does not have this problem since the mobile host has full control over *how much* data the sender should send.

To evaluate the performance benefits in power consumption, we use the IEEE 802.11b wireless card as a case study. The IEEE 802.11b card consumes 1.65W, 1.4W, and 0.045W when operated in the transmit, receive, and sleep modes respectively [19, 28]. We consider a two-state Markov error model for varying the channel condition of the wireless link [2]. The packet error rate in the good state is set to 0.01% and that in the bad state is set to 10% (for deep fades). The mean duration of the good state is set to 10s, while that of the bad state varies from 1s to 10s depending on the scenario. We assume an energy-frugal mobile host that enters the sleep mode whenever it detects the channel is in the bad state. Once in the sleep mode, all data transmissions and receptions are suspended (hence all packets in transit that arrive during this period are lost). The mobile host wakes up periodically every 100ms to listen to the beacons from the access point [16], during which it also measures the channel state using the received signal strength. The duration of the beacon is 2ms, and the power consumed for receiving the beacon is based on the value assumed for the receive mode. Once the mobile host decides that the channel is in the good state, it *defreezes* and resumes data transmission as usual.

Figure 5(c) compares the performance of RCP and TCP in terms of power consumption and achieved throughput when the mean duration of the bad state varies from 1s to 10s. We assume that the sender is unaware of the channel state, and hence when TCP is used, the mobile host receives data and transmits *ACKs* irrespective of the channel state. On the other hand, when RCP is used, the mobile host enters and leaves the sleep mode as mentioned before. The mobile host freezes the RCP timer when it enters the sleep mode. When it wakes up, RCP resumes data request based on the state (holes) of the receive buffer. As expected, the longer the mobile host stays in the sleep mode, the more energy savings it can achieve using RCP. While the energy savings are obvious, Figure 5(c) also shows an interesting result that compares the achieved throughput between TCP and RCP. Since the mobile host suspends all packet transmissions and receptions in the sleep mode, it obviously suffers from throughput loss in terms of giving up the data in transit and giving up the time to use the channel. However, for various conditions of the channel state, the throughput achieved using RCP is in fact no less than that when using TCP. The reason that TCP suffers from a more pronounced performance degradation is due to the adverse reaction of the congestion control mechanism in the presence of severe packet losses.

4. R^{2}CP: RADIAL RCP

We have shown in Section 3 that a receiver-centric transport protocol like RCP has performance gains over a sender-centric one, in terms of intelligent loss recovery, scalable congestion control, and efficient power management. However, the recent trends where mobile hosts are increasingly equipped with heterogeneous wireless interfaces, have severely exposed the limitations of the functionalities provided by existing transport protocols. Specifically, as we discussed in Section 2.2, when a mobile host handoffs from one interface to another during a live connection, it can benefit from the following functionalities the transport protocol supports: (i) seamless handoffs without relying on infrastructure support, (ii) server migration for achieving service continuity, and (iii) bandwidth aggregation using multiple active interfaces.

In the following, we present how a multi-state extension of RCP at the receiver called R^{2}CP can achieve the desired functionalities, without the requirement of changing the senders. We first discuss the design motivation of the R^{2}CP protocol, and then present the architectural overview and protocol details. Finally, we demonstrate the functionality gains achievable in R^{2}CP using network simulation and testbed emulation.

4.1 Design

4.1.1 *Receiver-Centric Operation*

To achieve optimal performance, a mobile host may need to use network (or interface) specific congestion control. When the mobile host is equipped with heterogeneous wireless interfaces, a receiver-centric protocol allows it to freely use the desired congestion control mechanism depending on the interface it chooses, or the access network it migrates to, without involving the remote server. In addition, during periods of mobility, the mobile host may need to handoff from one server to another (for service continuity), or change the number of servers it connects to (for bandwidth aggregation). It is thus advantageous for the mobile host to use a receiver-centric protocol with a simple sender design, allowing the mobile host to have control over the reliable delivery of data from the sender(s). RCP, being a receiver-centric protocol that allows the mobile host to drive the protocol operation such as congestion control and reliability, hence turns out to be an ideal protocol for the target environment.

4.1.2 *Maintaining Multiple States*

Existing transport protocols suffer from performance degradation during handoffs across heterogeneous networks due to the prolonged handoff latency Mobile IP introduces. While end-to-end host mobility without relying on the support from the infrastructure has been proposed [32], it does not fully address this problem due to the single-state design in TCP that maintains only one TCB [25] per connection. When link layer handoffs invalidate the state maintained at the transport layer (e.g. due to the change in IP addresses), the transport layer protocol needs to modify its state accordingly for achieving transport layer mobility. Although [32] intelligently performs connection migration, it introduces packet losses by "overwriting" the old state right after the new one is created. An ideal solution for achieving state migration, however, should allow the two states to co-exist in the connection for as long as it takes to handoff the states (considering packets in transit). Therefore, to support transparent host mobility without infrastructure support, a transport layer protocol should be able to handle multiple states. We hence build R^{2}CP as a multi-state extension of RCP. R^{2}CP dynamically creates and deletes RCP states according to the number of active interfaces in use. It effectively maintains multiple states at the mobile host without requiring explicit support from the remote server. *No change is necessary at the RCP sender to support the multi-state operation at the receiver.* R^{2}CP thus is different from related approaches [15, 26] that require changing both ends to support the multi-state operation. Since R^{2}CP is a receiver-only extension of RCP, it allows the mobile host to establish a multipoint-to-point connection to communicate with multiple servers, while in related work multiple states are confined to within a unicast connection.

4.1.3 *Decoupling of Functionalities*

An R^{2}CP connection with k active interfaces consists of k states at the receiver. Effectively, R^{2}CP maintains one RCP *pipe* per end-to-end path that exists between the receiver and the sender(s). R^{2}CP minimizes the overheads due to maintaining multiple states in a connection, by decoupling the transport layer functionalities associated with the per-pipe characteristics from those that pertain to the aggregate connection. For example, congestion control, being a per-pipe functionality, is handled by individual RCP pipes.

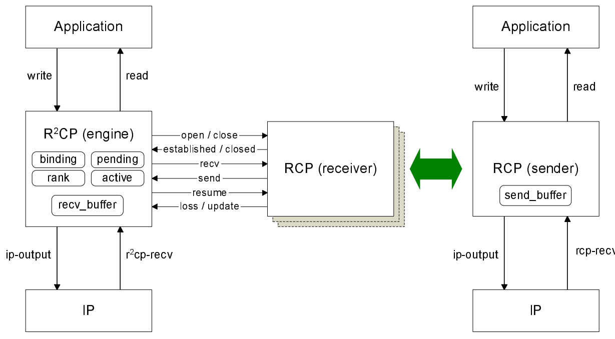


Figure 6: R^{2}CP Architecture

On the other hand, reliability and socket buffer management pertain to the aggregate connection, and hence are handled by R^{2}CP itself. Therefore, the R^{2}CP engine controls what data to request from each sender, and individual RCP pipes control how much data it can request along its path. The overheads due to repetitive implementations of transport layer functionalities are minimized.

4.1.4 *Effective Packet Scheduling*

A key challenge in maintaining multiple states in a connection is the effective multiplexing of pipes with mismatched characteristics in terms of bandwidths, delays, and loss rates. Specifically, since R^{2}CP uses multiple RCP pipes across heterogeneous interfaces to request data from one or multiple senders, data segments with smaller sequence numbers traversing the slower pipes may arrive later than those with larger sequence numbers traversing the faster pipes. Out-of-order arrivals at the receive buffer thus may cause head-of-line blocking and make the aggregate connection stall. R^{2}CP achieves effective multiplexing and bandwidth aggregation by scheduling transmissions (requests) based on the congestion window and the round-trip time of each RCP pipe. Briefly, R^{2}CP assigns the sequence of requests to each RCP pipe based on the (estimated) time the requested segment will arrive through the concerned pipe. Moreover, a request is assigned to an RCP pipe only when there is space in its congestion window. Any loss detected by individual RCP pipes is reported to R^{2}CP such that the corresponding request is reassigned to another pipe that has space in its window, to prevent the aggregate connection from stalling. Hence, head-of-line blocking due to segment losses, and bandwidth or delay mismatches of individual pipes is minimized.

4.2 Overview

Figure 6 presents an architectural overview of R^{2}CP and its key data structures. An R^{2}CP connection consists of one receiver, and one or multiple senders. Different senders of an R^{2}CP connection can be located at one or multiple hosts. While a unicast R^{2}CP connection is in fact equivalent to an RCP connection, a multipoint-to-point R^{2}CP connection can be considered as an aggregation of multiple RCP connections whose receiving ends are coordinated by an R^{2}CP engine at the receiver using the interface functions shown in the figure. We refer to the *virtual* connections that exist between the R^{2}CP receiver and individual senders as RCP pipes, and focus on the receiver for the following discussions.

When the application at the mobile host opens an R^{2}CP connection, initially one RCP pipe is created between the active interface and the remote server. When the mobile host handoffs from one interface to another, a new RCP pipe between the newly active interface and the server is created, after which the old RCP pipe is deleted. However, if bandwidth aggregation is possible (the old interface remains active after handoffs) and desirable (instructed by the application through a socket option), the old pipe is not deleted. If server migration is required when the mobile host handoffs to the new interface, the new RCP pipe is created between the newly active interface and the new server. The application can use a socket option to convey the address of the new server to R^{2}CP.

Whenever multiple RCP pipes co-exist in an R^{2}CP connection, the R^{2}CP engine performs transmission scheduling using the data structures shown in Figure 6, to minimize out-of-order arrivals due to data requested through different RCP pipes. Since multiple RCP pipes collaboratively request data for the same connection, it is possible that data requested through individual pipes is non-contiguous, depending on the transmission schedule used by the R^{2}CP engine. Hence, in R^{2}CP the request is always transmitted in the pull mode (refer to Section 3.3.1), such that the sender can transmit *only* the data requested. However, to facilitate loss detection and loss recovery, at the receiver each RCP pipe internally maintains a local sequence number space. Since the R^{2}CP engine controls the packet I/O (to and from the IP layer), it converts the local sequence number used by each RCP pipe to the global sequence number used by the aggregate connection before sending out the packet, and vice versa. We discuss in Section 4.3.1 how the conversion is achieved.

4.3 Protocol

In this section, we describe the key protocol functionalities in R^{2}CP including scheduling, connection management, congestion control, flow control, and reliability.

4.3.1 Scheduling

A key functionality in R^{2}CP is to perform packet scheduling across multiple RCP pipes. R^{2}CP (the R^{2}CP engine) maintains the following four key data structures for achieving this goal:

- `binding`: For each request sent out by one of the RCP pipes, R^{2}CP maintains the mapping between the local sequence number of the concerned RCP pipe, and the global sequence number of the aggregate connection in the `binding` data structure. The pipe through which the data segment is requested is also recorded in the `binding` data structure.
- `pending`: The ranges of sequence numbers for data yet to be requested are maintained in the `pending` data structure. It consists of the sequence numbers of data segments that need to be retransmitted (requested again), and sequence numbers greater than the highest sequence number requested so far.
- `rank`: For every outstanding request for segment i (one with starting sequence number i) sent by pipe j, an element is inserted into the `rank` data structure with a timestamp of $T^i + 2 * RTT_j$, where T^i is the time at which the request was transmitted, and RTT_j is the round-trip time of pipe j. The timestamp is reflective of the time the data segment requested in response to the arrival of segment i, is expected to arrive.
- `active`: When an RCP pipe issues a request to R^{2}CP for transmission, R^{2}CP can return with `FREEZE` to the corresponding RCP pipe due to unavailable space in the receive buffer. In such an event, R^{2}CP adds the concerned RCP pipe to the `active` data structure. When any space is created in the receive buffer, R^{2}CP issues a *resume*() call to each of the pipes in the `active` data structure.

We now explain how R^{2}CP uses these data structures to perform transmission scheduling and interacts with individual RCP pipes. When pipe j uses the *send*() call with RCP sequence number s for transmission request at time T, R^{2}CP locates the rank k of the request by comparing $T + RTT_j$ with existing entries in the `rank` data structure. Then it finds segment i as the k^{th} segment to request in the `pending` data structure, updates the entry for segment i in the `binding` data structure with (j, s), and inserts an entry (i, $T + 2 * RTT_j$) in the `rank` data structure. Finally, it uses the sequence number i in the request header, and sends out the request. When a data segment m arrives, R^{2}CP deletes the corresponding entry in the `rank` data structure, enqueues the data in the receive buffer, finds the corresponding RCP pipe and its local sequence number q based on the `binding` data structure, and passes q to the corresponding RCP pipe using the *recv*() call. The concerned RCP pipe then updates its states (e.g. congestion control parameters and the next sequence number to send), and determines whether it can send more requests or not. In case it can generate more requests, it uses the *send*() interface with the next RCP sequence number for transmission request as before.

Upon receiving the transmission request from any RCP pipe, if there is no available space for more data in the receive buffer, R^{2}CP returns with `FREEZE` to freeze the concerned RCP pipe, and puts it in the `active` data structure. If later any buffer space opens up due to, say, the arrival of the head-of-line segment, R^{2}CP uses the *resume*() call to de-freeze all pipes in the `active` data structure. Whenever an RCP pipe detects a loss, it uses the *loss*() call to inform R^{2}CP. R^{2}CP then unbinds the lost segment in the `binding` data structure, inserts the sequence number in the `pending` data structure, and deletes the corresponding entry from the `rank` data structure. Whenever an RCP pipe updates its RTT estimate, it uses the *update*() call to inform R^{2}CP, which then updates the `rank` data structure for pending requests pertaining to the concerned pipe.

4.3.2 Connection Management

When R^{2}CP creates an RCP pipe, it uses the *open*() call to make the RCP pipe start the connection setup procedure. The connection setup procedure for each RCP pipe is discussed in Section 3.3.2. When the RCP pipe is established, it uses the *established*() call to notify R^{2}CP. The R^{2}CP connection is established when any of the RCP pipe returns with the *established*() call. On the other hand, when R^{2}CP deletes an RCP pipe, it uses the *close*() call to make the RCP pipe enter the closing handshake. When all RCP pipes return with the *closed*() call, the R^{2}CP connection is closed.

4.3.3 Congestion Control

Congestion control in an R^{2}CP connection is performed on a per-pipe basis, where each RCP pipe is responsible for controlling the amount of data transferred through the respective path. R^{2}CP decides the congestion control mechanism to use for each wireless interface by opening an appropriate RCP pipe (e.g. RCP-NewReno, or RCP-STP as we discussed in Section 3.4.3). We assume the choice as to which congestion control scheme to use for each interface is an external decision, and is provided to R^{2}CP through a system configuration or a socket option.

4.3.4 Flow Control

Since R^{2}CP has control over the receive buffer, it is responsible for the flow control of the aggregate connection. R^{2}CP freezes a requesting RCP pipe if it finds that the number of outstanding data is equal to the available buffer space. It de-freezes concerned pipes through the *resume*() call when any space is created in the buffer. The flow control mechanism for individual RCP pipes that we discuss in Section 3.3.4 will not kick in since they do not deal with the actual data segments. Note that R^{2}CP is also responsible for appropriately informing the senders about what data to purge using the `SEG.DEQ` field in the RCP header that we discussed in Section 3.3.4.

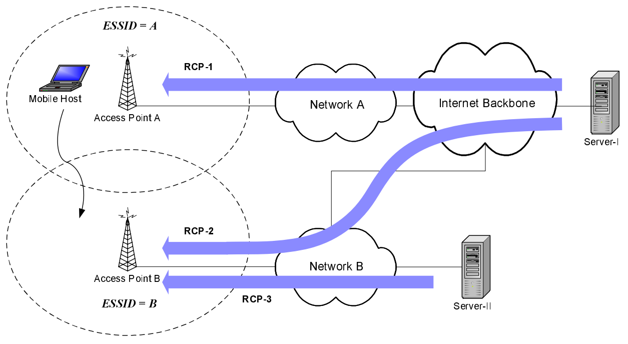


Figure 7: R^{2}CP Testbed Scenario

4.3.5 *Reliability*

R^{2}CP is primarily responsible for the reliable data transfer of the aggregate connection. It achieves this goal by maintaining the binding information for all data segments. Once a segment is bound to a particular RCP pipe, the concerned pipe will take over the responsibility (since RCP is a reliable protocol). However, note that when an RCP pipe detects a segment loss and reports to R^{2}CP using the *loss*() call, R^{2}CP will unbind the corresponding data segment, and delegate the reliable transfer of the lost segment to the next available pipe (according to the rank). While the original RCP pipe will still strive to deliver the same segment (in terms of the RCP sequence number) via retransmissions, it will be assigned a different data segment by R^{2}CP.

4.4 Functionality Gains

In this section, we show the functionality gains when using R^{2}CP at a mobile host with heterogeneous wireless interfaces. We use both network simulation and testbed emulation to present the results. While *ns-2* has been popularly used for network simulation, it can also be used as an emulator to interact with a live network. The protocol object developed in *ns-2* can tap into the device driver of the interface card (of the host where *ns-2* is running) to inject real packets to the network. Packets received by the interface card can also be dispatched to the target protocol object in *ns-2*. The advantage of using emulation is that packets generated by the emulator experience the same bandwidth fluctuations, round-trip time variations, and losses as any other live traffic in a real network. This is especially useful for evaluating the performance of the protocol in an uncontrolled wireless environment. We use the testbed shown in Figure 7 for performing emulation. The mobile host is an IBM Thinkpad T-20 laptop, and the servers are Dell Optiplex GX110 desktops. The mobile host is equipped with two IEEE 802.11b interfaces that allow it to connect to two WLANs belonging to different administrative domains (the two cards are associated with different ESSIDs, and assigned different IP addresses). Server-I and Server-II are replicated file servers. We also use simulation with controlled parameters (e.g. bandwidth and round-trip time) to show the performance of R^{2}CP in various environments.

4.4.1 *Seamless Handoffs*

When mobile hosts handoff between heterogeneous wireless networks, a key challenge in supporting seamless handoffs is the problem associated with address change and prolonged registration delay. Conventional approaches for performing vertical handoffs suffer from connection disruptions due to this problem. As we explained in Section 4.1, the multi-state design in R^{2}CP allows it to open multiple connections (pipes) associated with the wireless interfaces that become active during handoffs. By retaining the old connection (for as long as the link layer supports) during the initial setup delay of the new connection, the application can continue transmitting and receiving data from either or both interfaces without being disrupted during handoffs.

We show in Figure 8(a) the testbed results when the mobile host handoffs from one access network to another. The mobile host is initially connected to Server-I through network A, and hence one RCP pipe (RCP-1) is created in the R^{2}CP connection. At $t = 58s$, the mobile host decides to handoff to network B, so a second RCP pipe (RCP-2) is created (using the new network address). However, as the figure shows, RCP-1 is not closed until $t = 60s$ (a preset value), and hence during $t = 58s$ and $t = 60s$ two pipes co-exist in the connection to collaboratively deliver data for the application. Even if there is some setup or ramp-up (e.g. due to slow start) delay for the RCP-2 pipe, the existence of the RCP-1 pipe allows the aggregate connection to continue progressing without being disrupted. This is very different from related work that uses a single-state transport protocol for handoffs. Since R^{2}CP is a multi-state transport protocol, it is capable of maintaining multiple (interface specific) pipes effectively in a connection without suffering from problems due to packet reordering or duplicates. Note that the redundant striping technique proposed in [15] can also be used during handoffs for achieving better performance.

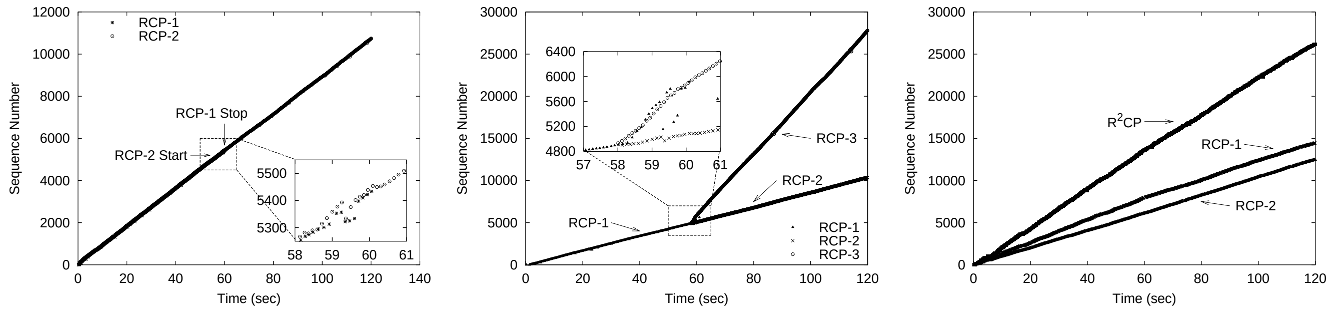
4.4.2 *Server Migration*

A key difference between R^{2}CP and other multi-state transport protocols is the ability to support end-point handoffs in R^{2}CP. By virtue of its receiver-centric design, the sender does not maintain any “hard” state (e.g. retransmission timers) of the connection. Since the mobile host controls which data to receive from the sender, handoffs from one server to another can be as simple as stop requesting data from the old server, and start from the new one. As we described in Section 2.2.2, server migration involves interaction between the transport layer and higher layer protocols. We focus in this section on the ability of R^{2}CP to facilitate server migration given sufficient support from the higher layers, and hence motivate its use as a valuable and effective building block for end-to-end mobility support frameworks.

As Figure 7 shows, when the mobile host moves to network B, it has access to a replicated server (Server-II). The end-to-end path from the mobile host (using interface B) to Server-II has a shorter round-trip time and a larger bandwidth, and hence the mobile host decides to perform server migration from Server-I to Server-II. Initially, the R^{2}CP connection creates an RCP pipe (RCP-1) using network address A and the address of Server-I. When the mobile host moves to network B, R^{2}CP creates a new RCP pipe (RCP-3) using network address B and the address of Server-II. Note that in Figure 8(b) we also show a contrasting scenario where the mobile host does not perform server migration, and hence the second RCP pipe created (RCP-2) is between network address B and the address of Server-I. After the new RCP pipe is established, the mobile host requests data that has not been delivered by Server-I, instead of requesting from the first byte of the data.[2] The difference between the slopes of RCP-2 and RCP-3 indicates that RCP-3 provides a larger bandwidth than RCP-2. Approaches used for achieving seamless handoffs discussed in Section 4.4.1 can also be used for achieving seamless server migration.

Based on the content of its receive buffer, R^{2}CP may request non-contiguous data from Server-II. Hence server migration using

[2]Note that the RCP sender on Server-II may need to purge from its send buffer the data that is not required by the receiver, with or without the application’s interaction.



(a) Seamless Handoffs (b) Server Migration (c) Bandwidth Aggregation

Figure 8: R^{2}CP Testbed Results

R^{2}CP does not not cause redundant transmissions compared to that using only TCP (the TCP sender delivers only in-sequence data stream). While support for selective pulling of data is provided by some applications (e.g. HTTP 1.1 Range Requests), it can be achieved in R^{2}CP with *no support from the server side application.*

4.4.3 Bandwidth Aggregation

When a mobile host handoffs between heterogeneous wireless networks, it is possible that the old connection remains active after the handoff is complete. In such a case, it would be advantageous for the mobile host to achieve aggregate bandwidths by simultaneously using both interfaces. Since R^{2}CP allows multiple RCP pipes to co-exist in one connection, and performs effective transmission scheduling for striping across multiple pipes, a mobile host using R^{2}CP can easily achieve bandwidth aggregation if desired.

We first consider the testbed scenario shown in Figure 7. While bandwidth aggregation can be achieved between the mobile host and one server (point-to-point), we consider a scenario where the two pipes connect to different servers (multipoint-to-point). The mobile host opens the RCP-1/RCP-2 pipe between network address A/B and the address of Server-I/Server-II respectively. However, instead of closing the RCP-1 pipe after RCP-2 is established, the mobile host keeps both pipes open during the period it is within the coverage of both WLANs. As shown in Figure 8(c), R^{2}CP can achieve the aggregate bandwidth of the two pipes.

We now use simulation to evaluate the performance of R^{2}CP in achieving effective bandwidth aggregation under various network conditions. We use a network topology similar to the testbed topology shown in Figure 7. The mobile host opens two pipes to aggregate bandwidths from different servers. We vary the characteristics of the two paths, in terms of the bandwidth of the bottleneck link, and the round-trip time of the entire path, to introduce bandwidth mismatches and delay mismatches. We also introduce bandwidth fluctuations by using on/off traffic sources as we described in Section 3.4. We compare the performance of R^{2}CP against the following approaches: (i) Ideal: the ideal performance of bandwidth aggregation, where the aggregate bandwidth equals the sum of bandwidths along the two pipes; (ii) APPS: an application layer striping approach (similar to the one used in [15]), where the application stripes across multiple RCP connections without using R^{2}CP; and (iii) R^{2}CP-s: a simplified version of R^{2}CP, where the data request is assigned to individual pipes on a first-come-first-served basis without considering the round-trip times.

Due to lack of space, we present only a subset of the performance results in Figure 9. In Figure 9(a), we vary the bandwidth of the two pipes such that the bandwidth of the first pipe is fixed at 4Mbps, while that of the second pipe varies from 1Mbps to 6Mbps. We observe that both R^{2}CP and R^{2}CP-s achieve the ideal performance irrespective of the bandwidth mismatches. The application striping approach fails to achieve the desired performance for the same reason explained in [15]. In Figure 9(b), we vary the round-trip time of the two pipes such that the RTT of the first pipe is fixed at 30ms, while that of the second pipe varies from 30ms to 210ms. We find that while the performance of R^{2}CP still closely tracks the ideal performance, R^{2}CP-s fails to scale when the RTT mismatch increases beyond 3. The performance degradation of R^{2}CP-s is due to the scheduling used that does not take into consideration the round-trip times of different pipes. While an FCFS style of striping policy works well when the round-trip times of different paths are comparable, as the RTT mismatches increase, it suffers from frequent out-of-order arrivals. Due to the limited space in the R^{2}CP receive buffer, head-of-line blocking eventually triggers the flow control of R^{2}CP and causes the progression of the aggregate connection to stall. We show in Figure 9(c) the percentage of packets that find the buffer 75% full upon arrivals, for three different striping approaches. The reason for the non-performance of the application striping approach is clear from the figure. While R^{2}CP-s manages to maintain a small queue size when the RTT mismatches are small, the queue builds up noticeably as the RTT mismatches increase. R^{2}CP, on the other hand, achieves better performance even with large RTT mismatches.

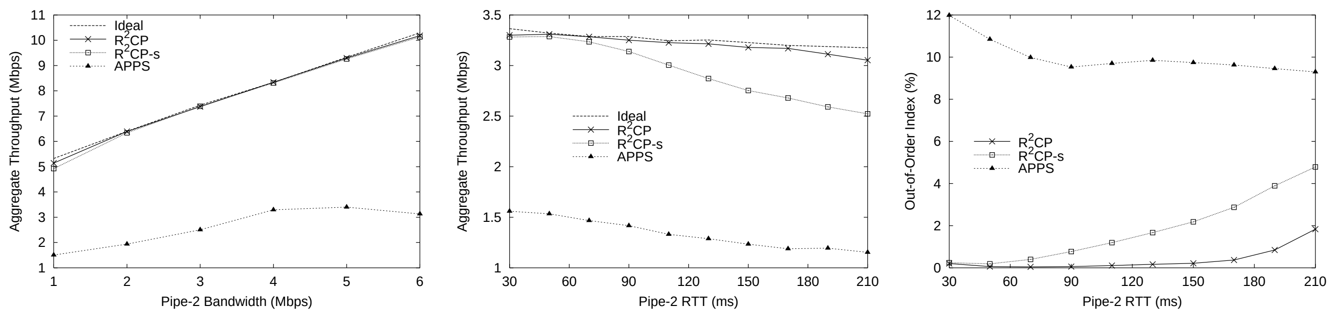
5. DISCUSSIONS

In this section, we first discuss the overheads and complexities at the mobile host when using a receiver-centric transport protocol like RCP. We then discuss several extensions for RCP when the mobile host also acts as the sender, and when the traffic source is not network-limited.

5.1 Mobile Host Overheads

A question that arises when moving the intelligence of the transport protocol from the server to the mobile host is: would such design demand a more sophisticated mobile host, or prove to be a drain on the precious battery resource at the mobile host?

Note that RCP is designed for mobile hosts that use TCP as the transport layer protocol. Since RCP is a receiver-clone of TCP, it merely transposes various functionalities performed in TCP from the sender to the receiver. As we showed in Figure 2, RCP does not increase the complexity of the protocol. Since TCP is a duplex protocol, any implementation of the TCP protocol stack at the



(a) Bandwidth Mismatch (b) RTT Mismatch (c) Out-of-Order Index

Figure 9: Performance of R^{2}CP Scheduling

mobile host already includes functionalities of both the sender and the receiver, irrespective of whether the mobile host acts as a TCP sender or receiver. Hence, in terms of the footprint, RCP does not introduce any overhead at the mobile host. In terms of complexities, it is obvious that the mobile host performs more functionalities as an RCP receiver than it does as a TCP receiver. However, in [7] the authors study the processing overheads of the TCP/IP protocol stack. They find that after a packet is dispatched to the target TCP socket, the TCP sender uses 448 (CPU) instructions to process the packet before sending it out, while the TCP receiver uses 421 instructions. Moreover, they find that after a packet is received by the interface card, the protocol-specific processing overheads (including TCP, IP, and ARP) for a 1460-byte packet constitute only 8% of the total processing overheads – the majority being the data touching (memory copy) and operating system overheads. The authors in [18] using packets of different sizes also make the same observation. Since the complexity of the RCP receiver is similar to that of the TCP sender, it can be expected that the processing overheads will increase only minimally when RCP is used at the mobile host.

In terms of energy consumption, since the mobile host performs more functionalities, it is intuitive that the mobile host will consume more power when acting as an RCP receiver (instead of a TCP receiver). However, note that such difference in power consumption exists only when the mobile host is transmitting or receiving packets (after which it will process the packet following the state machine of the transport protocol in use). We use the following numbers to explain that the power consumed for transmitting or receiving packets through the wireless interface significantly outweighs the extra power consumed due to the increased computation required by RCP. For a laptop with a Pentium-III 800MHz CPU, we measure the per-packet CPU cycles required for it to act as a TCP sender and a TCP receiver respectively. On a Linux operating system we find that the CPU cycles increase by about 5000 when the host is used as a TCP sender. Such extra CPU cycles can be translated to additional CPU energy consumption of about 9.38μJ. When using an IEEE 802.11b wireless card to transmit or receive a packet with a packet size of 1000 bytes, it consumes approximately 1.28mJ. Therefore, the per-packet energy consumption increases by about 0.7% when performing the functionalities of a TCP sender. We conclude that the increased energy consumption does not noticeably decrease the time the mobile host can have to transmit or receive data.

5.2 RCP Extensions

In Section 3 we present the RCP protocol by focusing on a scenario where the mobile host acts as the receiver, and the application used is backlogged (network-limited). We now briefly discuss the operations and extensions of RCP when used in other scenarios.

5.2.1 Upstream Traffic

While mobile hosts predominantly consume data retrieved from the backbone server, it is possible that they also need to upload data to the backbone server. In such a scenario, the mobile host has the following two options: (i) *Use RCP for Upstream Traffic:* Note that RCP is a duplex protocol like TCP, and hence the mobile host can use RCP to send or receive data. When uploading data to the server, the mobile host acts the RCP sender, and the server acts as the RCP receiver. While this option simplifies the design of the protocol, the performance and functionality gains that we discussed in Section 2 would not apply in such a scenario. (ii) *Use TCP for Upstream Traffic:* A duality exists between TCP and RCP for upstream and downstream traffic. As a data source, the mobile host can control how much and what data to send by using TCP. Therefore, using TCP for upstream traffic has the same advantages as using RCP for downstream traffic.

Seemingly, one solution for the mobile host to handle traffic in both directions is to implement both TCP and RCP. However, being a TCP clone, RCP can reuse many algorithms already implemented in TCP, such as congestion control and resequencing. We hasten to add that an ideal transport protocol for mobile hosts with heterogeneous wireless interfaces is one that is *transpositional*, with TCP and RCP standing at both ends of the spectrum. A transpositional transport protocol can dynamically redistribute the functionalities of the protocol to the sender or the receiver depending on, say, the direction of the traffic or the capability of the device. Note that such a transpositional protocol does not need to actually "move" the codes between the sender and the receiver, but simply changes the data path and/or control path in the protocol stack.

5.2.2 Application-Limited Traffic

In Section 3 we assume that the sender always has data available in the send buffer for the receiver to consume. However, it is possible that the application used is data-limited (e.g. telnet), or the sender needs to perform *reverse* flow control against the receiver due to its processing limit.[3] In these cases, like TCP, the sender can explicitly advertise flow control to throttle the amount of requests issued by the receiver. When the RCP receiver hits the limit

[3]However, note that an RCP sender is simpler than a TCP sender. A server running RCP thus can service more users than a server running TCP, if the bottleneck is the processing power.

imposed by the sender, it will enter the persist mode as in TCP [39]. Afterward, it can periodically probe the sender for request permissions, or wait for the explicit window update from the sender. Such flow control can also be used to notify the receiver when there is no more data to send at the sender.

5.2.3 Rate-Controlled Traffic

While we have consciously positioned RCP as a TCP clone that uses the window based congestion control, it is possible to extend RCP for traffic sources that need rate control. For example, the sender can maintain a send timer to control the rate at which it can burst out data. The send rate will be provided by the receiver through its congestion control mechanism. Note that since the receiver has control over the reliability mechanism of the protocol, it can switch between reliable (TCP-like) or unreliable (UDP-like) data delivery without changing the semantics at the sender. Our ongoing work also includes building R^{2}CP atop a reliable protocol such as RCP for streaming applications where prompt delivery, instead of reliable delivery, is more desirable. Using RCP to support an application that does not require full reliability is possible, as a retransmission in an RCP pipe does not necessarily mean a retransmission of the same application data.

6. RELATED WORK

While sender-centric approaches have prevalently been adopted in the design of transport layer protocols, a considerable amount of work has also focused on increasing receiver participation (compared to TCP) in the protocol operation. In the following, we discuss several related work that leverages the existence of the receiver for improving the performance and functionality of the protocol.

In [8], the authors propose a transport protocol called NETBLT for achieving high throughput performance in bulk data transmission. While NETBLT is in fact a sender-centric protocol, an interesting design is the relocation of the retransmission timer from the sender to the receiver. The authors contend that since the receiver knows which packet has been received and which has not, unnecessary retransmissions can be eliminated when timeout occurs. However, in NETBLT the sender is still predominantly responsible for performing loss recovery, and hence the receiver needs to use SACK for conveying other losses to the sender. WTCP, proposed in [30], is an example that uses the receiver for performing congestion control. The WTCP receiver calculates the rate at which the sender can send, based on the inter-packet delay of received packets. By maintaining the history of packet losses, the receiver can intelligently distinguish between the congestion losses and corruption losses, and adjust the send rate accordingly. While the receiver does control the send rate, WTCP is not a fully receiver-centric transport protocol. Reliability is still the role of the sender, and hence the receiver needs to periodically send back *ACKs* (CACK and SACK) to inform the sender of its buffer state. Other rate-based transport protocols such as TFRC [13] and TCP-Real [38] also use the receiver for tracking loss events or achieving better loss identification. Still, the functionality of the receiver is limited to providing a more accurate feedback or estimation of the transmission rate that the sender can use.

In [12], the authors propose a receiver-driven transport protocol called WebTP for optimizing the performance of Web data transfer. WebTP follows the request/response model used in HTTP, where the connection is initiated, controlled, and terminated by requests from the receiver and responses from the sender. Like RCP, WebTP is a fully receiver-centric transport protocol where the receiver is responsible for flow control, congestion control, and reliability, while the sender merely transmits whatever packet the receiver requests. However, since WebTP is primarily designed for the wired environment, it does not address the issue of request losses, which RCP handles through the use of the cumulative mode and the cyclic buffer as we discussed in Section 3.3.1. Moreover, while WebTP also follows the congestion control algorithms used in TCP, it employs a different timeout detection mechanism based on the packet inter-arrival times, and it does not cut down the congestion window in response to the detection of three out-of-order arrivals. It has been shown in [12] that WebTP is in fact more aggressive than TCP, whereas RCP is friendly to TCP as shown in Figure 4. Unlike RCP, the design of WebTP is closely coupled to the target application. WebTP processes the application data unit (ADU) directly, and performs reliability and priority control at the ADU level. Nonetheless, WebTP does make a case for using HTTP atop a receiver-centric transport protocol to address the inefficiencies associated with the use of a sender-centric protocol like TCP.

Finally, in [33], the authors consider receiver-based management of low bandwidth access links. They observe that since the mobile host increasingly tends to maintain several concurrent connections across the bandwidth-limited access link, it is important to prioritize connections depending on the type of the application used (e.g. an interactive application has a higher priority than file download). Since the mobile host is aware of the bandwidth of the access link and the relative importance of different connections, a receiver-based approach is ideal for bandwidth management. However, using TCP, the mobile host in the proposed approach needs to override the advertised window to indirectly control (through the server) the bandwidth used by different connections. We note that if RCP is used, such bandwidth management can be achieved at the mobile host without involving the server or changing the advertised window. The authors in [24] also consider a similar problem of sharing the bandwidth of the wireless last-hop among multiple TCP flows. The proposed approach adjusts the bandwidth share of each TCP flow by manipulating the round-trip time as well as the advertised window. While the use of RTTs allows more flexibility in controlling the per-flow bandwidth share, it requires the receiver to perform RTT estimation (already implemented at the sender side), and to artificially inflate RTTs by using delayed *ACKs*. It is obvious that a receiver-centric transport protocol like RCP can support such receiver-based bandwidth allocation better than TCP.

7. CONCLUSIONS

In this paper, we present a receiver-centric protocol called RCP that is a TCP clone in its general behavior. We show that for mobile hosts in wireless networks, RCP allows better loss recovery, congestion control, and power management mechanisms compared to a sender-centric transport protocol like TCP. More importantly, in the context of recent trends where mobile hosts are increasingly being equipped with multiple interfaces providing access to heterogeneous wireless networks, we show that RCP enables a powerful and comprehensive transport layer solution for such multi-homed hosts, including the ability to (i) enjoy seamless handoffs, (ii) use network specific congestion control schemes, (iii) facilitate server migration, and (iv) achieve flexible bandwidth aggregation. We use both packet level simulations, and real Internet experiments to evaluate the proposed protocol.

8. REFERENCES

[1] Akamai Technologies. *Akamai Accelerated Network Program*. http://www.akamai.com.

[2] B. Bakshi, P. Krishna, N. Vaidya, and D. Pradhan. Improving performance of TCP over wireless networks. In *Proceedings of IEEE ICDCS*, Baltimore, MD, USA, May 1997.

[3] H. Balakrishnan, V. Padmanabhan, S. Seshana, and R. Katz. A comparison of mechanisms for improving TCP performance over wireless links. *IEEE/ACM Transactions on Networking*, 5(6):756–769, Dec. 1997.

[4] S. Biaz and N. Vaidya. Discriminating congestion losses from wireless losses using inter-arrival times at the receiver. In *Proceedings of IEEE ASSET*, Richardson, TX, USA, Mar. 1999.

[5] E. Blanton, M. Allman, K. Fall, and L. Wang. A conservative SACK-based loss recovery algorithm for TCP. IETF Internet Draft; *draft-allman-tcp-sack-13.txt*, Oct. 2002.

[6] D. Bovet and M. Cesati. *Understanding the Linux Kernel*. O'Reilly & Associates, Sebastopol, CA, USA, Dec. 2002.

[7] D. Clark, V. Jacobson, J. Romkey, and H. Salwen. An analysis of TCP processing overhead. *IEEE Communications Magazine*, 27(6):23–39, June 1989.

[8] D. Clark, M. Lambert, and L. Zhang. NETBLT: A high throughput transport protocol. In *Proceedings of ACM SIGCOMM*, Stowe, VT, USA, Aug. 1987.

[9] ETSI. *BRAN; HIPERLAN/2; Requirements and Architecture for Internetworking between HIPERLAN/2 and 3rd Generation Cellular Systems*. TR 101 957, Aug. 2001.

[10] S. Floyd and T. Henderson. The NewReno modification to TCP's fast recovery algorithm. IETF RFC 2582, Apr. 1999.

[11] T. Goff, J. Moronski, and D. Phatak. Freeze-TCP: A true end-to-end TCP enhancement mechanism for mobile environments. In *Proceedings of IEEE INFOCOM*, Tel-Aviv, Israel, Mar. 2000.

[12] R. Gupta, M. Chen, S. McCanne, and J. Walrand. A receiver-driven transport protocol for the web. In *Proceedings of INFORMS Telecommunications Conference*, Boca Raton, FL, USA, Mar. 2000.

[13] M. Handley, S. Floyd, J. Pahdye, and J. Widmer. Equation-based congestion control for unicast applications. In *Proceedings of ACM SIGCOMM*, Stockholm, Sweden, Aug. 2000.

[14] T. Henderson and R. Katz. Satellite transport protocol (STP): An SSCOP-based transport protocol for datagram satellite networks. In *Proceedings of Workshop on Satellite-Based Information Services*, Budapest, Hungary, Oct. 1997.

[15] H.-Y. Hsieh and R. Sivakumar. A transport layer approach for achieving aggregate bandwidths on multi-homed mobile hosts. In *Proceedings of ACM MOBICOM*, Atlanta, GA, USA, Sept. 2002.

[16] IEEE. *Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications*. ANSI/IEEE Standard 802.11, Aug. 1999.

[17] V. Jacobson, R. Braden, and D. Borman. TCP extensions for high performance. IETF RFC 1323, May 1992.

[18] J. Kay and J. Pasquale. Profiling and reducing processing overheads in TCP/IP. *IEEE/ACM Transactions on Networking*, 4(6):817–828, Dec. 1996.

[19] R. Krashinsky and H. Balakrishnan. Minimizing energy for wireless web access with bounded slowdown. In *Proceedings of ACM MOBICOM*, Atlanta, GA, USA, Sept. 2002.

[20] L. Magalhaes and R. Kravets. Transport level mechanisms for bandwidth aggregation on mobile hosts. In *Proceedings of IEEE ICNP*, Riverside, CA USA, Nov. 2001.

[21] S. Mascolo, C. Casetti, M. Gerla, M. Sanadidi, and R. Wang. TCP-Westwood: Bandwidth estimation for enhanced transport over wireless links. In *Proceedings of ACM MOBICOM*, Rome, Italy, July 2001.

[22] M. Mathis and J. Mahdavi. Forward acknowledgement: Refining TCP congestion control. In *Proceedings of ACM SIGCOMM*, Palo Alto, CA, USA, Aug. 1996.

[23] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow. TCP selective acknowledgement options. IETF RFC 2018, Oct. 1996.

[24] P. Mehra, C. De Vleeschouwer, and A. Zakhor. Receiver-driven bandwidth sharing for TCP. In *Proceedings of IEEE INFOCOM*, San Francisco, CA, USA, Apr. 2003.

[25] J. Postel. Transmission control protocol. IETF RFC 793, Sept. 1981.

[26] M. Riegel and M. Tuexen. Mobile SCTP. IETF Internet Draft; *draft-riegel-tuexen-mobile-sctp-02.txt*, Feb. 2003.

[27] A. Sanmateu, L. Morand, E. Bustos, S. Tessier, F. Paint, and A. Sollund. Using Mobile IP for provision of seamless handoff between heterogeneous access networks, or how a network can support the always-on concept. In *Proceedings of EURESCOM Summit*, Heidelberg, Germany, Nov. 2001.

[28] T. Simunic, L. Benini, P. Glynn, and G. De Micheli. Dynamic power management for portable systems. In *Proceedings of ACM MOBICOM*, Boston, MA, USA, Aug. 2000.

[29] H. Singh and S. Singh. Energy consumption of TCP Reno, Newreno, and SACK in multi-hop wireless networks. In *Proceedings of ACM SIGMETRICS*, Marina Del Rey, CA, USA, June 2002.

[30] P. Sinha, N. Venkitaraman, R. Sivakumar, and V. Bharghavan. WTCP: A reliable transport protocol for wireless wide-area networks. In *Proceedings of ACM MOBICOM*, Seattle, WA, USA, Aug. 1999.

[31] A. Snoeren, D. Andersen, and H. Balakrishnan. Fine-grained failover using connection migration. In *Proceedings of USENIX USITS*, San Francisco, CA, USA, Mar. 2001.

[32] A. Snoeren and H. Balakrishnan. An end-to-end approach to host mobility. In *Proceedings of ACM MOBICOM*, Boston, MA, USA, Aug. 2000.

[33] N. Spring, M. Chesire, M. Berryman, V. Sahasranaman, T. Anderson, and B. Bershad. Receiver based management of low bandwidth access links. In *Proceedings of IEEE INFOCOM*, Tel-Aviv, Israel, Mar. 2000.

[34] M. Stemm and R. Katz. Vertical handoffs in wireless overlay networks. *Mobile Networks and Applications (MONET)*, 3(4):335–350, 1998.

[35] F. Sultan, K. Srinivasan, D. Iyer, and L. Iftode. Migratory TCP: Connection migration for service continuity in the Internet. In *Proceedings of IEEE ICDCS*, Vienna, Austria, July 2002.

[36] The Network Simulator. *ns-2*. http://www.isi.edu/nsnam/ns.

[37] V. Tsaoussidis, H. Badr, X. Ge, and K. Pentikousis. Energy/Throughput tradeoffs of TCP error control strategies. In *Proceedings of IEEE ISCC*, Antibes, France, July 2000.

[38] V. Tsaoussidis and C. Zhang. TCP-Real: Receiver-oriented congestion control. *Computer Networks*, 40(4):477–497, Nov. 2002.

[39] G. Wright and W. Stevens. *TCP/IP Illustrated, Volume 2*. Addison-Wesley Publishing Company, Reading, MA, USA, Oct. 1997.

[40] M. Zorzi and R. Rao. Is TCP energy efficient? In *Proceedings of IEEE MoMuC*, San Diego, CA, USA, Nov. 1999.