Inference and Evaluation of Split-Connection Approaches in Cellular Data Networks

Wei Wei[†], Chun Zhang[†], Hui Zang[‡], Jim Kurose[†], Don Towsley[†]
[†]Department of Computer Science
University of Massachusetts, Amherst, MA 01003
[‡]Sprint Advanced Technology Laboratories, Burlingame, CA 94010
UMass Computer Science Technical Report 2005-59

Abstract

Numerous mechanisms have been proposed for improving TCP performance over wireless links, including those in wireless cellular networks. In this paper, we infer the existence and investigate the performance of one class of these performance-enhancing approaches, split-connection approaches, in commercial cellular data networks. Special attention is given to the so called Split-TCP and TCP proxy approaches. We present inference techniques to identify whether a cellular provider implements Split-TCP or TCP proxy. Through end-to-end measurements over commercial cellular networks operated by three different providers (including two CDMA2000 networks and one GPRS network), we find that all three providers selectively implement Split-TCP or TCP proxy for certain applications (e.g., HTTP). We also find that the implementations differ from provider to provider. Experimental results demonstrate that the performance gains from Split-TCP depend on flow sizes and that TCP proxy using data compression outperforms Split-TCP.

I. Introduction

The explosive growth of wireless networks poses challenges to existing network protocols and applications that are designed for wired networks. For instance, TCP, the most widely used transport protocol in the Internet, has been shown to perform poorly in wireless networks [1]. Numerous mechanisms have been proposed for improving TCP performance over wireless links, including those in wireless cellular networks. These approaches fall broadly into three classes [2], [3], [4]: link layer approaches, end-to-end approaches, and split-connection approaches. Split-connection approaches insert a split point between the wireless and wired host, thus splitting the end-end TCP connection into two connections: one between the wired host and the split point, and the other between the split point and the wireless host (e.g., [5]). The main idea behind split-connection is to isolate wireless related issues from the TCP sender (which usually resides on the wired network). Our focus in this paper will be on two split-connection techniques: (1)*Split-TCP*, in which a split point splits the original TCP connection into two concurrent connections; (2)*TCP proxy*, in which a proxy, located at the split point, first receives all the data from the sender and only then forwards the data to the receiver. The proxy may also execute performance-enhancing functions such as caching or data compression before forwarding the data. Note that TCP proxy differs from Split-TCP in that the proxy obtains the data first and then forwards it to the receiver, while data flows simultaneously in the two connections in Split-TCP. In the rest of the paper, we refer to a TCP connection using the split-connection technique as a *split-connection* TCP and refer to the traditional end-to-end TCP implementation as *regular* TCP.

In this paper, we ask the following question: *Have split-connection techniques such as Split-TCP or TCP proxy been deployed in commercial cellular networks? If so, what are the performance gains from these techniques?* We answer this question by conducting end-to-end experiments in commercial networks operated by three different providers (including two CDMA2000 networks and one GPRS network). Our paper makes the following contributions:

- We design techniques to detect/infer the existence of performance-enhancing mechanisms such as Split-TCP and TCP proxy. Our empirical measurements indicate that all three providers selectively implement Split-TCP or TCP proxy for certain applications (e.g., HTTP). We also find that the performance-enhancing mechanisms deployed differ from provider to provider.
- Using end-to-end measurements, we study the performance gains provided by Split-TCP and TCP proxy. Empirically, we find that the performance gains can sometimes be dramatic.
- We characterize the behavior of Split-TCP, TCP proxy and regular TCP in cellular networks. Our observations are important for modeling TCP in cellular networks. We also explore practical issues in implementing Split-TCP, e.g., whether the TCP connections are terminated gracefully when a connection is aborted.

Most proposals to improve TCP performance in wireless networks have been evaluated through analysis and/or simulation; there are very few performance studies of real implementations of TCP or measurements in an operational network. In [3], the authors compare mechanisms for improving TCP performance over wireless links by conducting experiments on a testbed. The testbed environment, however, does not reflect the effects of scheduling in a cellular data network [6], and has wireless link bandwidth that is much higher than that typically available in today's cellular data networks. Our experiments, on the other hand, are conducted in commercial cellular data networks and thus directly consider the influence of various mechanisms (e.g., scheduling) that are used in practice in these data networks. A recent study measures TCP performance in GPRS networks by analyzing traces collected at the gateway of the wireless provider's network to the public Internet [7]. The behavior of the wireless network is inferred from the trace. We differ from this work in that our measurements are end-to-end, with a focus on end-to-end characteristics and the use of split-connection approaches.

The rest of the paper is organized as follows. Section II describes our experimental setup. The study of the deployment of split-connection approaches is described in Section III. A performance evaluation of Split-TCP and TCP proxy is presented in Section IV. Finally, Section V concludes the paper.

II. Experiment Configuration

We conduct experiments over three commercial cellular data networks operated by three service providers, named as CDMA-I, CDMA-II, and GPRS-I, based on the technology that they use. All of our experiments are TCP-based. Our goal is two-fold: (i) to detect whether split-connection techniques are used in these commercial networks, (ii) to characterize split-connection TCP and quantify the performance gains from split-connection techniques.

Figure 1 illustrates the basic configuration for our experiments. A wired host, shown as the desktop in Figure 1, is connected via an Ethernet interface to a LAN, which in turn is connected to the Internet via a high-speed wired link. A mobile station, shown as the laptop, is connected to the Internet via the cellular service provider's network using a commercial wireless data card (also referred to as an *aircard*) sold by the service provider. The desktop is a Linux

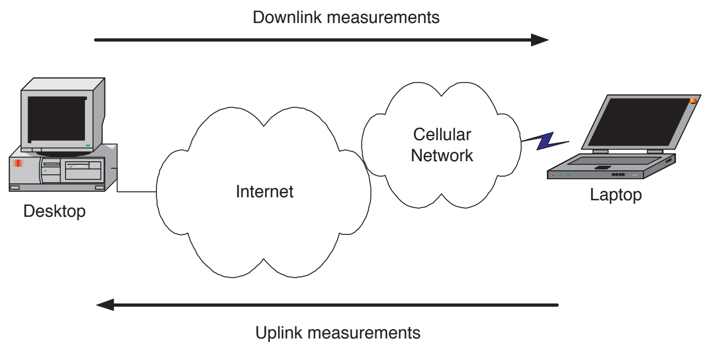


Fig. 1. Experiment configuration.

server running RedHat 9.0 and the laptop runs Windows XP. In the rest of the paper, we refer to the desktop as the *wired host* and the laptop the *mobile*.

We call an experiment a *downlink* experiment if the data packets (relative to ACKs) flow from the wired host to the mobile, and an *uplink* experiment otherwise. We conduct HTTP downloads (to the mobile), FTP downloads (to the mobile), FTP uploads (from the mobile), and a general TCP-based file transfer using our own program in both directions. For HTTP or FTP experiments, an Apache HTTP server or an FTP server runs on the wired host. For the general TCP-based file transfer, we set up a TCP server running on the wired host (or mobile) that accepts connections from the mobile (or wired host) on a pre-determined port.

In all experiments, we collect packet traces using *tcpdump* [8] on the wired host and *windump* [9] on the mobile. To obtain one way end-to-end delays, we use software on Linux and Windows XP to synchronize clocks at the sender and receiver. All experiments are performed from January to October, 2005.

III. Inferring Deployment of Split-connection Mechanisms

In this section, we describe passive and active techniques to detect the use of split-connection mechanisms. We then apply these techniques to determine whether TCP connections carrying different application-layer protocols (e.g., FTP, HTTP) are split inside the three providers' networks. We then illustrate the characteristics of Split-TCP and TCP proxy. Last, we explore an implementation issue, namely, graceful termination of Split-TCP.

A. Techniques to detect split-connection

In some cases, a split TCP connection can be detected by visually inspecting the *tcpdump* trace. For instance, the sender (or receiver) IP address on packets sent by the sender (e.g., the wired host) may differ from these on packets received at the receiver (e.g., the mobile). Also the payload size, receiver window size, IPID or even transport protocol may change. However, these are not necessary conditions for a split-connection. For example, we find that IP spoofing can be used by a split point to hide split-connection approaches. In the following, we describe passive and active methods to detect split TCP connections. The passive method only requires passively observing the timing information of a TCP connection while the active method requires actively delaying the ACKs at the receiver. We summarize the passive and active methods in Method 1, 2 and 3. A detailed discussion is omitted in the interest of space and can be found in [10].

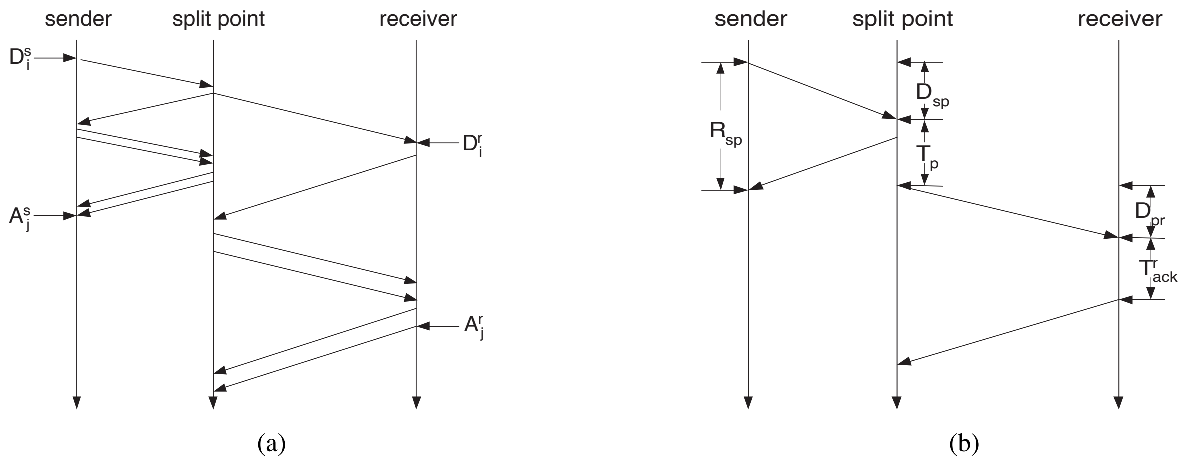


Fig. 2. Passive detection of split-connection: (a) detection method when the sender and receiver clocks are not synchronized; (b) condition for effective detection.

We first present our passive detection methods. Consider an arbitrary packet from the sender to the receiver[1]. After receiving the packet, the receiver generates an ACK and sends it to the sender at time A^r. Let A^s denote the receiving time of this ACK at the sender. Then we have the following result to detect split-connection:

Method 1: **(Passive detection method)** When the clocks at the sender and the receiver are perfectly synchronized, a split TCP connection can be detected if there exists a packet with $A^s < A^r$.

The main idea of this detection method is as follows. When the sender and receiver clocks are synchronized, for a regular TCP, we must have $A^s \geq A^r$. Therefore, a violation of this inequality indicates that the connection is split. When the clocks at the sender and receiver are not synchronized, we generalize the above result to detect a split-connection as follows. Consider two arbitrary packets, packet i and j, $j \geq i$, as shown in Fig. 2(a). For the i-th packet, let D_i^s be its sending time at the sender and D_i^r be its receiving time at the receiver. For the j-th packet, let A_j^r be the time when the receiver returns the corresponding ACK and A_j^s be the receiving time of this ACK at the sender. Then we have the following more generalized method to detect split-connection:

Method 2: **(Generalized passive detection method)** When the offsets of the clocks at the sender and the receiver are fixed, a split TCP connection can be detected if $A_j^s - D_i^s < A_j^r - D_i^r$ for any i, j, $1 \leq i < j \leq n$, where n is the length of the trace.

The main idea of this detection method is as follows. For a regular TCP, we must have $A_j^s - D_i^s \geq A_j^r - D_i^r$ since the i-th packet cannot reach the receiver before it is sent by the sender, and the ACK for the j-th packet cannot reach the sender before it is returned by the receiver. When applying Method 2, we set $i = 1$, vary j from 1 to n. Method 2 requires that the offsets of the two clocks are fixed. In practice, we assume that the offset variation between the sender and the receiver lies within a small error threshold. This is reasonable since our experiments are short (within tens of minutes).

We find that our passive detection methods are very effective in detecting downlink split-connection but ineffective for the uplink direction. We briefly explain this for the case that the sender and receiver clocks are synchronized; the case for unsynchronized clocks is similar. Consider an arbitrary packet in a Split-TCP connection, as shown in

[1]We refer to data packets (relative to ACKs) simply as packets.

Fig. 2(b). For this packet, let R_{sp} denote the RTT of the TCP connection between the sender and the split point; let D_{sp} be the one-way delay from the sender to the split point; let D_{pr} be the one-way delay from the split point to the receiver. After receiving a packet, if the split point cannot forward the packet immediately to the receiver, it queues the packet in a buffer. For the packet we consider, let T_p be its queuing delay at the split point. Finally, suppose that, after receiving the packet, the receiver returns the corresponding ACK after an interval of T^r_{ack}. Then the sufficient condition to detect split-connection (i.e., $A^s < A^r$ in Method 1) is equivalent to $R_{sp} < D_{sp} + T_p + D_{pr} + T^r_{ack}$. In the downlink direction, since the bandwidth from the sender to the split point is much higher than that from the split point to the receiver, the queuing (buffering) delay of a packet at the split point, T_p, is large (tens of seconds, see Section III-B.1). Therefore, the sufficient condition for detecting split-connection is easily satisfied. This is not true for the uplink direction since T_p is not sufficiently large.

We next describe an active detection method for the uplink direction.

Method 3: **(Active detection method)** At the receiver, we delay every ACK by a time interval of T before it is returned to the sender. Let $\{R_i\}_{i=1}^n$ denote the sequence of round-trip times (RTTs) measured at the sender, where n is the length of the trace. If $\min_{1 \le i \le n}\{R_i\} < T$, then the TCP connection is split.

Note that this active method requires no clock synchronization or fixed clock offsets between the sender and the receiver. For all our experiments, we combine our detection methods and visual inspection of *tcpdump* traces to determine whether a TCP connection is split. After detecting a split-connection, we further determine whether TCP proxy or Split-TCP is used. In the uplink direction, differentiating the use of a TCP proxy and Split-TCP is straightforward. When a TCP proxy is used, the receiver will experience a relatively long delay before receiving any data packets since the speed of the wireless connection is very low and the file downloading at the proxy is slow. In the downlink direction, we differentiate a TCP proxy from Split-TCP based on the receiver advertised window size (i.e., the available receiving buffer size at the receiver), which is embedded in the ACKs and received at the sender. When Split-TCP is used, we observe that the receiver window alternates between zero and the full size of the receiving buffer, indicating that the receiver buffer is filled and then depleted by the mobile user. When a TCP proxy is used, the receiver window size is essentially constant since the proxy consumes a packet as soon as the packet is received. Another indication of a TCP proxy is: when repeatedly downloading the same file, later downloads may not be from the original server; instead the file is retrieved from the cache at the TCP proxy.

B. Use of split-connection in commercial networks

We now report the prevalence of split-connections in the three providers' networks using the techniques described earlier. The results are summarized in Table I for easy reference.

1) Use of split-connection by FTP: In each provider's network, we run FTP in both the downlink and uplink directions. Using the methodologies described in Section III-A, we find that CDMA-I and CDMA-II use Split-TCP in both directions, while GPRS-I does not split TCP connections for FTP. We next describe the results for our two CDMA networks in more detail.

In CDMA-I's network, visual inspection of *tcpdump* traces does not reveal the use of split-connection. However, using our passive and active detection techniques, we find that split-connection is used in both the downlink and uplink

TABLE I

SPLIT-CONNECTION SCHEMES DEPLOYED BY DIFFERENT PROVIDERS

	CDMA-I	CDMA-II	GPRS-I
FTP	Split-TCP	Split-TCP	No
HTTP data	TCP proxy/Split-TCP	Split-TCP	Split-TCP
HTTP image	TCP proxy	TCP proxy	TCP proxy
TCP	No	No	Split-TCP(optional)

directions. For traces collected in the downlink direction, according to Method 2, we set $i = 1$ and vary j from 1 to the length of the traces and observe a significant number of packets (over 60%) satisfying the condition in Method 2. Hence, we conclude that a split-connection is used in the downlink direction. In the uplink direction, we delay ACKs by 10 seconds using *Nistnet* [11] at the receiver, which is a wired host at University of Massachusetts, Amherst (UMass). We then measure the RTTs at the TCP sender and find that they all fall below 10 seconds, which implies that the TCP connection is split (see Method 3).

In CDMA-II's network, we directly observe the IP address of the split point in received packets. Furthermore, the protocol used in the connection between the split point and the mobile is changed to UDP[2]. The above visual inspection indicates the use of split-connection techniques.

2) Use of split-connection by HTTP: We conjecture that different techniques may be applied to HTTP data and image objects in cellular data network due to the special characteristics of images. We therefore conducted two sets of experiments. In the first set of experiments, a mobile host requests an HTTP data object from a web host. In the second of experiments, an HTTP image is requested. Using the methodology in Section III-A, we find that CDMA-I uses a TCP proxy for both HTTP data and images[3]; CDMA-II and GPRS-I use Split-TCP for HTTP data and TCP proxy for HTTP image, as summarized in Table I.

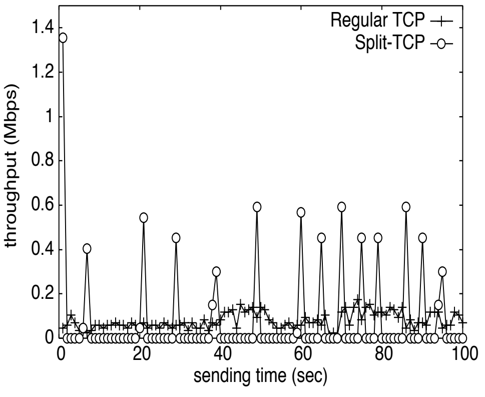
3) Use of split-connection by TCP: We also run TCP experiments using our own programs in both downlink and uplink directions, using a port different from that used by FTP and HTTP. For this case, we find that a regular TCP connection is used in CDMA-I and CDMA-II's network. GPRS-I allows the mobile user to explicitly split a TCP connection. When the user chooses to split the connection, the TCP connection is split. Otherwise, it is not split.

C. Characteristics of Split-TCP and TCP proxy

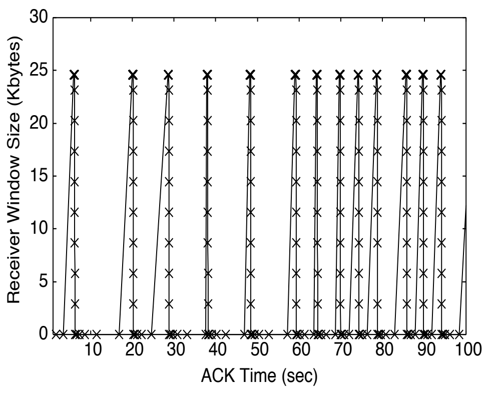
We now compare the sender throughput and end-to-end delay of Split-TCP, TCP proxy and regular TCP using traces collected in CDMA-I's network; similar results were observed in CDMA-II and GPRS-I network. The traces for Split-TCP, TCP proxy and regular TCP were collected using FTP, HTTP and our programs respectively. All traces were in the downlink direction. We observe that, when using Split-TCP, the throughput measured at the sender is very bursty: the sender alternates between transmitting for a very short period of time and not transmitting. One example is shown in Fig. 3(a), which plots a time series of the sender throughput averaged every second. Correlating the sender throughput and the receiver window size (returned from the split point) (plotted in Fig. 3(a) and Fig. 3(b) respectively), we find that the sender stops sending because the receiving buffer at the split point is full and resumes sending when there is

[2]We speculate that UDP tunneling is used between the split point and the mobile. That is, TCP packets are encapsulated into UDP packets.

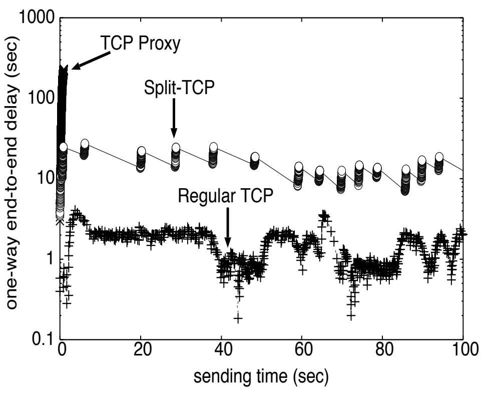
[3]We observe that CDMA-I originally used TCP proxy and switched over to Split-TCP for HTTP data after September, 2005.



(a)



(b)



(c)

Fig. 3. Characteristics of Split-TCP, TCP proxy and regular TCP: (a) sender throughput; (b) receiver windows size of Split-TCP; (c) end-to-end delay

space in the buffer. The sender throughput when using regular TCP is much less bursty than that when using Split-TCP (see Fig. 3(a)). When using TCP proxy, the sender transmits at a rate of approximately 17Mbps and completes the transfer quickly (in 1 second). The receiver completes the downloading much later (in around 200 seconds) due to the low bandwidth of cellular network.

Fig. 3(b) plots the end-to-end delays using Split-TCP, TCP proxy and regular TCP. Packets are indexed by their sending time at the sender. We observe that the end-to-end delays when using Split-TCP fluctuate and in general can be longer than those when using regular TCP: when using Split-TCP, most of the end-to-end delays lie above 10 seconds, while the end-to-end delays are below a few seconds using regular TCP. The very long end-to-end delays of Split-TCP (which were observed to be as long as 100 seconds in some traces) are caused by the buffering at the split point. When using TCP proxy, end-to-end packet delay varies from a few seconds to more than 200 seconds. Furthermore, packets in the later part of the file exhibit longer delay. This is expected since all packets reach the TCP proxy quickly and packets in the later part of the file need to wait longer at the proxy before being transmitted to the receiver.

D. Graceful Termination of Split-TCP

Since Split-TCP violates the end-end semantics of TCP, one potential concern is that TCP connections (between the sender and the split point and between the split point and the receiver) may not terminate gracefully when a TCP connection is aborted in the middle. We investigate this issue by downloading 2M bytes HTTP data objects from a UMass web server, and aborting the download while in progress. We observe that, for CDMA-II, both TCP connections terminate immediately after the downloading is aborted. For CDMA-I, the TCP connection between the split point and the client is terminated within a few seconds, while the TCP connection between the server and the split point is terminated within 40 seconds. For GPRS-I, only the TCP connection between the split point and the client is terminated. The TCP connection between the server and the split point continues for about 12 minutes. During this period, the server continually probes the available receiving buffer size at the split point, always receiving zero as the reply. This is because the TCP session between the split point and the client is already terminated and hence the data

TABLE II

PERFORMANCE IMPROVEMENT FROM SPLIT-TCP OVER REGULAR TCP

	West Coast		East Coast	
File Size (KBytes)	Throughput Improvement	Split-TCP Throughput(Kbps)	Throughput Improvement	Split-TCP Throughput(Kbps)
1	-24%	18	-32%	18
5	64%	59	26%	42
10	65%	72	56%	63
50	41%	87	97%	94
100	26%	111	51%	95
500	5%	119	13%	124
1000	0%	120	7%	123

in the receiving buffer cannot be removed.

IV. PERFORMANCE GAINS FROM SPLIT-CONNECTION TECHNIQUES

In this section, we quantify the performance gains from Split-TCP and TCP proxy techniques.

A. *Performance gains from Split-TCP*

For a fair performance comparison between Split-TCP and regular TCP, we compare them using the same application. From experiments, we learned that, in CDMA-I's network, a connection running FTP uses Split-TCP if the FTP server runs on (listens to) the regular FTP control port (port 21). Otherwise, the connection is not split. Based on this observation, we compare the performance of Split-TCP and regular TCP using FTP in CDMA-I's network.

We configure two FTP servers (wired hosts) running on different ports: one on port 21 and the other on a port other than 21. The two servers are co-located on the west coast. To explore the effect of the server-client distance on performance, we configure two clients (mobiles), one on the east coast and the other on the west coast. A client alternatively connects to each server to download a file. We vary the file size from 1 K to 1000K bytes. Each experiment is repeated at least 100 times. All experiments are conducted after midnight, in order to reduce the influence of rate fluctuation caused by cellular voice users. Our performance metric is average throughput.

Table II summarizes the results over all experiments. We observe that the average throughput increases with the file size for both Split-TCP and regular TCP. For very small file sizes, Split-TCP performs worse than regular TCP. This is because, for a very small file, (which can fit in one packet), a single packet is transferred and the split point in Split-TCP only introduces additional delay. In all the other cases, Split-TCP outperforms regular TCP. Furthermore, the performance gains from using Split-TCP depend on the file size: the performance gains initially increase with the file size, reach a maximum value and then decrease with the file size. This can be explained intuitively as follows. At the beginning of a connection, a Split-TCP sends faster than a regular TCP. This is because, when using Split-TCP, the RTTs of the connection between the split point and the mobile host are less than those of a regular TCP, leading to a faster ramp-up of the TCP window. Furthermore, the initial window size at a Split-TCP may be larger than that in a regular TCP [12], [13]. Channel scheduling inside a cellular network may further increase the rate difference between a Split-TCP and a regular TCP, since a connection with a higher sending rate may be assigned with a higher bandwidth, e.g. via the assignment of additional channels [14]. However, the throughputs of a Split-TCP and a regular TCP eventually become the same when both reach the maximum bandwidth allowed by the cellular network. Therefore, the throughput gains from Split-TCP initially increase and then decrease with the file size. From Table II, we also observe that Split-TCP provides more dramatic improvements when the server and client are far away from each other.

We next evaluate the performance gain using analysis. For ease of exposition, we refer to the TCP session between the sender and the splitting point as the *upstream* TCP and the other session between the splitting point and the receiver as the *downstream* TCP. Let v denote the mobile data rate assigned by the cellular network. Let r_1 and r_2 denote the RTT of upstream TCP and downstream TCP respectively. As the sending rate reaching v, Split-TCP and regular TCP have the same throughput. We therefore only focus on the time required for the sending rate to reach v using regular TCP and Split-TCP, denoted as $T_{regular}$ and T_{split} respectively. In regular TCP, we assume that the TCP session has not finished slow start when its sending rate reaches v. Let n be the number of round trip needed for the regular TCP throughput to reach v. Then we have $2^n s = (r_1 + r_2)v$, where s denotes the size of a packet. Hence, $n = \log(r_1 + r_2)v/s$. Then $T_{regular} = n(r_1 + r_2) = (r_1 + r_2)\log(r_1 + r_2)v/s$.

In Split-TCP, we first assume that the downstream TCP does not go through slow start. Instead, the initial window size is $r_2 v$. That is, the downstream TCP fully utilizes the sending rate. The upstream TCP performs slow start and has not finished slow-start when its sending rate reaches v. The upstream TCP needs $\log(r_2 v/s)$ rounds to reach this window size. The total time to reach this window size is $T_{split} = r_1 \log(r_2 v/s)$.

$$\begin{aligned}
T_{regular} - T_{split} &= (r_1 + r_2)\log[(r_1 + r_2)v/s] - r_1\log(r_2 v/s) \\
&= (r_1 + r_2)\log[r_2(1 + r_1/r_2)v/s] - r_1\log(r_2 v/s) \\
&= (r_1 + r_2)(\log(r_2 v/s) + \log(1 + r_1/r_2)) - r_1\log(r_2 v/s) \\
&\approx (r_1 + r_2)(\log(r_2 v/s) + r_1/r_2) - r_1\log(r_2 v/s) \\
&= r_1\log(r_2 v/s) + r_2\log(r_2 v/s) + r_1^2/r_2 + r_1 - r_1\log(r_2 v/s) \\
&\approx r_1 + r_2\log(r_2 v/s)
\end{aligned}$$

The approximations hold because r_1/r_2 is small.

When $v = 128$ kbps, $s = 1500$ bytes, $r_1 = 0.03$ second, $r_2 = 0.8$ second. We have $T_{regular} - T_{split} \approx 2.5$ seconds. The amount of data downloaded by regular TCP before the sending rate reaches v is 25K bytes without using delayed ACK and 50K bytes when using delayed ACK. This explains why the largest relative performance gain is observed when the file size is around 50K bytes. As the file size increases or decreases, the relative difference between regular TCP and Split-TCP decreases.

We now assume that the downstream TCP performs slow start. In this case, $T_{split} = r_2\log(r_2 v/s)$. We have

$$\begin{aligned}
T_{regular} - T_{split} &= (r_1 + r_2)\log[(r_1 + r_2)v/s] - r_2\log(r_2 v/s) \\
&= (r_1 + r_2)\log[r_2(1 + r_1/r_2)v/s] - r_2\log(r_2 v/s) \\
&= (r_1 + r_2)(\log(r_2 v/s) + \log(1 + r_1/r_2)) - r_2\log(r_2 v/s) \\
&\approx (r_1 + r_2)(\log(r_2 v/s) + r_1/r_2) - r_2\log(r_2 v/s)
\end{aligned}$$

$$
\begin{aligned}
&= r_1 \log(r_2 v/s) + r_2 \log(r_2 v/s) + r_1^2/r_2 + r_1 - r_2 \log(r_2 v/s) \\
&= r_1 \log(r_2 v/s) + r_1^2/r_2 + r_1 \\
&\approx r_1(\log(r_2 v/s) + 1)
\end{aligned}
$$

Note that, this difference is roughly proportional to r_1, which explains why the performance gain from Split-TCP is more dramatic when the server and client are far away from each other. Using the parameters in the earlier example, we have $T_{regular} - T_{split} \approx 0.12$ second, indicating only a slight performance improvement by using Split-TCP. However, when considering the effect of scheduling, this improvement might be larger, as discussed below. When using Split-TCP, data flow to the wireless network faster than using regular TCP, since the throughput in the upstream TCP session in Split-TCP is higher. Therefore, supplemental channels are more likely to be assigned to a connection using Split-TCP than one using regular TCP, leading to a higher data rate in Split-TCP.

B. *TCP proxy versus Split-TCP*

We next compare the performance of TCP proxy and Split-TCP by downloading an image and a data file with the same size using HTTP in CDMA-I's network (Note that in Table I, in CDMA-I's network, after September, 2005, TCP proxy is used for HTTP images while Split-TCP is used for HTTP data). The size of both files is 1.5M bytes. In the experiments, a wired host acts as the server and a mobile acts as the client. Both the server and the client are located at UMass. When using HTTP, the image is compressed to a lower-quality image of 150K bytes while no compression is performed for the data file. We conduct 40 back-to-back downloads for both files using HTTP. The average downloading times are 14 and 109 seconds using TCP proxy and Split-TCP respectively (with the standard deviations of 1 and 8 seconds respectively). We observe that TCP proxy with data compression can outperform Split-TCP significantly.

V. Conclusion

In this paper we describe passive and active techniques to detect whether split-connection techniques are used in cellular data networks. Through end-to-end experiments in three commercial networks, we identified that Split-TCP and TCP proxy have been implemented in all these networks for selected applications. We observed that Split-TCP can achieve significant throughput improvement over regular TCP for relatively small-sized flows. Furthermore, we demonstrated that TCP proxy is quite effective in improving user perceived performance via data compression.

References

[1] R. Caceres and L. Iftode, "Improving the performance of reliable transport protocols in mobile computing environments," *IEEE J. Select.Areas Commun.*, vol. 13, no. 5, pp. 850–857, 1994.

[2] H. Balakrishnan, S. Seshan, and R. H. Katz, "Improving reliable transport and handoff performance in cellular wireless networks," *ACM Wireless Networks*, vol. 1, no. 4, pp. 469–481, 1995.

[3] H. Balakrishnan, V. N. Padmanabhan, S. Seshan, and R. H. Katz, "A comparison of mechanisms for improving TCP performance over wireless links," *IEEE/ACM Transactions on Networking*, vol. 5, no. 6, pp. 756–769, 1997.

[4] H. Elaarag, "Improving TCP performance over mobile networks," *ACM Computing Suryeys*, vol. 34, pp. 357–374, SEP 2002.

[5] A. Bakre and B. R. Badrinath, "I-TCP: Indirect TCP for mobile hosts," *15th International Conference on Distributed Computing Systems*, 1995.

[6] B. H. Kim, I. Lee, and K. Chu, “End-to-end application performance impact on scheduler in CDMA-1XRTT wireless system,” in *IEEE 61st Semiannual Vehicular Technology Conference*, May 2005.

[7] P. Benko, G. Malicsk, and A. Veres, “A large-scale, passive analysis of end-to-end TCP performance over GPRS,” in *Proc. Infocom 2004*, March 2004.

[8] “Tcpdump,” http://www.tcpdump.org/.

[9] “Windump,” http://windump.polito.it/.

[10] W. Wei, C. Zhang, H. Zang, J. Kurose, and D. Towsley, “Inference and evaluation of split-connection approaches in cellular data networks,” Tech. Rep. 05-59, Department of Computer Science, University of Massachusetts, Amherst, 2005.

[11] “NIST Net,” http://snad.ncsl.nist.gov/itg/nistnet.

[12] H. Inamura, G. Montenegro, R. Ludwig, A. Gurtov, and F. Khafizov, “RFC 3481 - TCP over Second (2.5G) and Third (3G) Generation Wireless Networks,” *RFC 3481*, 2003.

[13] M. Allman, S. Floyd, and W. C. Partridge, “Increasing TCP’s Initial Window,” *RFC 3390*, 2002.

[14] V. Vanghi, A. Damnjanovic, and B. Vojcic, *The cdma2000 System for Mobile Communications*. Prentice Hall Communications Engineering and Emerging Technologies Series, 2004.