TCP-PARIS: a Parallel Download Protocol for Replicas

Roger P. Karrer
Deutsche Telekom Laboratories
TU Berlin
Berlin, Germany
roger.karrer@telekom.de

Edward W. Knightly
ECE Department
Rice University
Houston, TX
knightly@ece.rice.edu

Abstract

Parallel download protocols have the potential to reduce file download time and to achieve a server-side load balancing in replica systems, such as peer-to-peer networks, content distribution networks and mirrored servers, by simultaneously establishing connections to multiple replicas and downloading disjoint file parts in parallel. This paper presents TCP-PARIS, a novel parallel download protocol from multiple replicas to one receiver. Because the ideal partitioning of the transfer volume from each server is a dynamic and a difficult-to-predict function of network conditions, server load and data size, TCP-PARIS uses the stream segmentation of TCP and congestion window information to continuously adapt the assigned volume to each server in proportion to the bandwidth-delay product to best approximate the optimal data partitioning. Analytical results, simulation and Internet experiments with a transport-layer implementation characterize the performance and the resource requirements of TCP-PARIS and allow a comparison with related protocols. Extensive simulations with varying network and application parameters show download time reductions of up to 52% compared to single-flow downloads and up to 52% compared to related protocols.

1 Introduction

Data replication is a key building block for large-scale distributed storage systems such as Mirrors, Content Distribution Networks (CDNs) and peer-to-peer (p2p) file sharing applications. Parallel download protocols that establish multiple connections to distributed replica servers and simultaneously download individual parts of the data in parallel, as depicted in Figure 1, promise a reduction of the data download time by up to a factor of n when using n parallel connections, compared to a download from a single server only. Moreover, the failure of a replica during the download reduces the download rate, but does not interrupt the transfer. Finally, parallel download protocols shed load on multiple servers and therefore automatically achieve a load balancing, resulting in higher server availability.

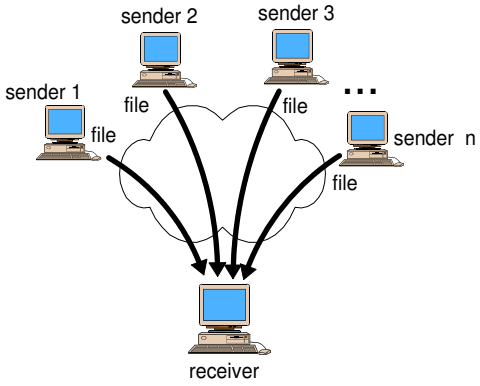


Figure 1. Parallel download from multiple servers

The challenges in the design of parallel download protocols arise because network conditions in the Internet are heterogeneous, dynamic and difficult to predict, and because file sizes follow a distribution that varies from application to application. Current parallel download protocols only exist as application-layer solutions that target specific application demands and network environments. CDNs download small pictures (ads) in parallel from low-latency servers.

Slicing protocols, such as BitTorrent [6], split multimedia files of several MB into smaller slices and coordinate the download of individual slices from multiple peers. Finally, Digital Fountain [2] targets unreliable environments (e.g., multicast), where the redundancy in the form of codes eliminates a download coordination. Unfortunately, the use of these parallel download protocols is limited to specific scenarios (e.g., CDNs are bound to http requests), and protocol performance is dismal if the assumptions about file size or bandwidth are not met.

In this paper we develop TCP-PARIS, a novel *PA*rallel download protocol for *R*epl*I*ca*S*. TCP-PARIS is a multipoint-to-point protocol that provides reliable file delivery by establishing n individual TCP subconnections and coordinating the download from these n replica servers. The key novelty of TCP-PARIS is to tightly couple the coordination with the TCP subconnections. In particular, TCP-PARIS uses the TCP segmentation to coordinate the download by assigning each server the responsibility to deliver only a subset of the segments. Moreover, TCP-PARIS uses congestion control information of the subconnections to dynamically adapt the downloaded volume from each server, allowing TCP-PARIS to achieve a near-optimal download independent of the network heterogeneity and dynamics.

This paper makes the following contributions. First, we develop the TCP-PARIS protocol. We show that the download coordination can be integrated with TCP congestion control with only two components: the partition rules that store for every sender which TCP segments it must deliver, and the Largest Assigned Sequence number (LAS), which allows the receiver to coordinate the download such that no segment is transmitted by two senders. While congestion control information is integrated to coordinate the download, TCP-PARIS does not modify congestion control of the subconnections.

Second, we describe two implementation variants of TCP-PARIS, one with receiver-based TCP connections and the other with sender-based TCP connections. The fact that the parallel download coordination is steered by the client simplifies the TCP-PARIS protocol with receiver-based TCP. We present implementations of TCP-PARIS in ns-2 and in the Linux kernel.

Finally, we perform an extensive set of simulation experiments to evaluate the impact of the systems' key performance factors. In particular, we assess the ability of TCP-PARIS to achieve a near-to-optimal download time for a single flow under varying network and application parameters and compare the results to slicing protocols. Moreover, we study the system performance of multiple interacting TCP-PARIS flows in a distributed environment and show that TCP-PARIS achieves download time reductions of up to 52% compared to single-server downloads and up to 52% compared to related parallel download protocols.

2 Design rationale

The ability of a parallel download protocol to cover a large span of network and application parameters is crucial for the download performance. As data replication is increasingly used in distributed systems, the benefits of parallel downloads can be exploited for a large variety of applications: Web (where parallel downloads can be made by a client browser or a proxy), peer-to-peer, mirrors, high-speed downloads in Grid environments, or even for multimedia real-time streaming. Therefore, a single parallel download protocol that can be used independent of a specific application scenario and that achieves the promised performance objective even under dynamic network conditions will significantly expand the use and the benefits of parallel downloads.

2.1 Requirements

In our quest to design a parallel download protocol, we will address the following protocol requirements

- **Bandwidth heterogeneity.** The protocol must accommodate a large variety of physical bandwidth rates, ranging from kb/sec connections (e.g., in p2p and CDNs) up to high-speed networks with Gb/sec speed [17, 16]. Moreover, the available bandwidth is a difficult-to-predict function of network conditions and server load.

- **File size heterogeneity.** The protocol must ensure an efficient download for different file size distributions. For CDNs, e.g., embedded objects may range from 100 KB (pictures) to several MB (large pictures, movies). Similarly, p2p file sharing applications may share files from 10s to 100s of MB. Finally, Grid applications may download files of several GB.

- **Reliability.** The protocol must guarantee to deliver the file even if servers suddenly and unexpectedly terminate a connection.

- **Generality.** The protocol must be broadly applicable to multi-point to point transfer. For example, the performance of real-time streaming multimedia downloads (applications or as embedded Web documents) could significantly be increased via parallel downloads, but only under the condition that the parallel streams synchronized in their delivery.

By focusing on these requirements, we will not discuss three important issues in this paper, as they can be addressed independently of TCP-PARIS. First, we ignore how an application learns the location of the servers on which the file is located. Current solutions range from explicit knowledge (e.g., a list of mirrors) to dynamic searches for servers or documents (as in p2p). Second, we assume that the application verifies that the documents on all servers are consistent. Ensuring consistency should be done prior to the download. Finally, we ignore that servers may be malicious and modify the content during the download. Solutions can be deployed independent of TCP-PARIS, or TCP-PARIS can be extended to check consistency, e.g. by downloading the same segments from multiple replicas.

2.2 Performance Objective

The performance objective of a parallel download protocol is to minimize the download time at a client. This minimization is determined by two factors: (i) the ability to efficiently use the available bandwidth and (ii) the ability to coordinate the volume downloaded from each server such that all downloads terminate at the same time. Consider a volume of data denoted by V (bits) to be downloaded by a client from servers $1, 2, \cdots, n$. Denote $V_i \geq 0$ as the volume transferred from server i with $\sum_i V_i = V$, and denote T_i as the download time for server i. Finally, denote $r_i(t)$ as the transmission bandwidth that a TCP flow obtains from replica i to the receiver during the download, and denote $R_i = \frac{1}{T_i}\int_0^{T_i} r_i(t)dt$ as the average download rate for the entire transfer. The download time of replica i is then

$$T_i = V_i/R_i \tag{1}$$

The first objective of a parallel download protocol is to utilize the transmission bandwidth $r_i(t)$ by ensuring that the server always has data to send at $r_i(t)$ to avoid starvation. If a server starves, the achieved transmission rate will be lower than $r_i(t)$ and, in some cases, even result in a termination of the connection. This objective is achieved if the *assigned volume* to server i, $v_i^*(t)$ is always larger than the bandwidth-delay product of the connection, i.e.,

$$v_i^*(t) \geq r_i(t) \cdot rtt_i(t) \tag{2}$$

The second objective requires that the difference in the total download time from each server must be minimized, as the download is not completed until the last byte is received:

$$\min(\max(T_1, \ldots, T_n)) \tag{3}$$

Obviously, this goal is achieved when all downloads terminate at the same time, i.e., $T_i = T_j$ for all replicas i, j.

In the simplest case in which the bandwidth R_i is static, known and homogeneous, the objectives are achieved by downloading a volume $V_i = V/n$. Moreover, if the bandwidth is known and heterogeneous, the solution to Equation (3) yields that $V_i/R_i = V_j/R_j$. The key challenge in protocol design is to achieve the objectives in the case that the rates are not known *a priori*, but change over time.

2.3 Overview of related protocols

Related work can be separated into 3 groups: parallel TCP connections between a single client-server pair, code-based protocols and coordination-based protocols.

First, parallel download protocols that establish multiple TCP connections between the same client-server pair have been proposed for high-speed networks [1, 7, 20]. TCP-PARIS takes parallel download to a next level by allowing a download from distributed replica servers.

Code-based protocols, such as Digital Fountain [4], SplitStream [5] and Bullet [13], add redundant code to each packet in the form of Tornado Codes [3] or Erasure Codes [2]. The redundancy enables the client to decode and reconstruct the original file when it receives *any* ηk packets of the original k packets. While no coordination is necessary as to which server must deliver which data, the overhead in redundant data averages 36% for Tornado Codes [3]. In networks with little packet loss, this overhead significantly increases the download time and reduces the goodput of a network.

Protocols where the receiver coordinates the download to ensure a non-disjoint data delivery have been proposed within different application contexts. First, Content Distribution Networks (CDNs) such as Akamai, download individual subdocuments embedded in an `html` request from different servers. As the volume of the subdocuments is fixed, CDNs must try to approximate, rather than optimize Equation 1. This approximation may be sufficient for small volumes V_i where R_i is dominated by the latency. However, significant performance drawbacks are expected for larger documents, and no parallel download support is provided for real-time streaming.

Alternatively, *slicing* protocols used in p2p applications split large files into multiple equally-sized slices that can be downloaded in parallel from different servers [14, 15]. The key problem for slicing protocols is to determine the best slice size: while small slices increase the flexibility of the protocol to meet the second performance objective, small slices are more likely to incur server starvation. Thus, the best slice size depends on dynamic run-time parameter, and a wrong selection leads to a significant increase in the download time [8]. BitTorrent [6] uses slice sizes of 256 KB, but additionally ensures that every server has at least k sub-pieces of x KB outstanding at the server. With typical values of $k = 5$ and $x = 16$ KB, BitTorrent generally avoids server starvation for average maximal congestion window sizes of 64 KB. While these parameter settings are efficient for most p2p downloads, slice sizes of 256 KB limit ability of the protocol to adjust the volume for small files and/or limit the number of parallel connections. Moreover, a non-trivial adjustment of the above parameters is required for high-speed environments for larger maximal congestion windows. Finally, slicing protocols lack real-time streaming support for multimedia.

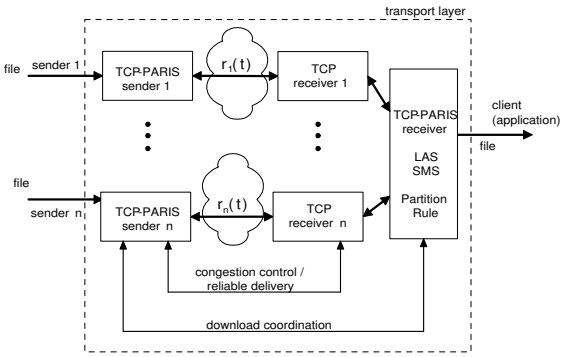


Figure 2. Parallel download with TCP-PARIS

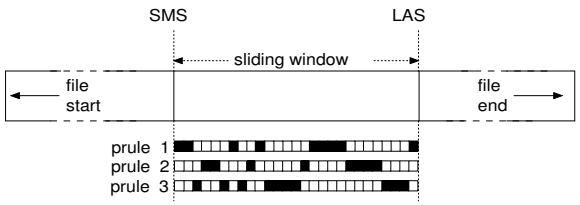


Figure 3. Sliding window

3 TCP-PARIS: a parallel download protocol for replicas

3.1 Coordinated download

TCP-PARIS is a multipoint-to-point protocol. A TCP-PARIS flow consists of n subconnections that connect a single receiver with n replica servers via point-to-point TCP subconnections, as depicted in Figure 2. The client sends a file request to the servers, which is acknowledged by the senders. Then, the TCP-PARIS receiver starts coordinating the parallel download by assigning every subconnection individual TCP segments to deliver [1]. Every server sends only those segments that are requested by the receiver.

The TCP-PARIS receiver coordinates the download of the subconnections using a sliding window protocol that reflects the status of the parallel download (i.e., of all subconnections), as depicted in Figure 3. The boundaries of the sliding window are maintained by 2 variables: the Largest Assigned Sequence number, *LAS* and the the Smallest Missing Sequence number, *SMS*. Segments are assigned sequentially to subconnections by the TCP-PARIS receiver whenever the congestion control of the subconnection allows the transmission of a new segment, ensuring that each segment is assigned to only one subconnection. The SMS is updated if the smallest missing segment of all subconnections is received, allowing all data up to the SMS to be passed to the application. The sliding window maintains a well-defined download status, as sequence numbers larger than the LAS have not been assigned yet, segments between LAS and SMS are in transit, and all segments smaller than the SMS have been successfully received by the TCP-PARIS receiver. Moreover, the sliding windows of each individual subconnection are contained in the boundaries set by LAS and SMS.

The coupling of the assignment with the congestion control ensures that the assigned volume v_i^* to server i equals the bandwidth-delay product of this subconnection throughout the download, i.e.

$$v_i^*(t) \sim r_i(t) \cdot rtt_i(t) \tag{4}$$

$r_i(t)$ is the rate achieved by each individual subconnection and is *not* modified by TCP-PARIS. Thus, at any time, each subconnection is TCP-friendly by itself, whereas the total rate of a TCP-PARIS connection is the aggregation of the individual rates. Thus, a TCP-PARIS connection can obtain an n-fold throughput of a single-flow download. In cases where the n subconnections share a common bottleneck, this aggregation is unfair and we will address this unfairness in future work. At the moment, we argue that TCP-PARIS is as unfair as other currently deployed parallel download protocol.

At any given time t, the assigned volume to a subconnection i corresponds to the size of the congestion window. $v_i^*(t)$ is therefore the smallest volume assigned to a server that ensures that the server never starves in its delivery. During an increase in the congestion window size, v_i^* is also incrementally increased. During a (multiplicative) decrease, however, v_i^* is temporarily larger than the product as the reduction of v_i^* requires several round-trip times. We will show that this delayed adaptation has an impact only if it occurs at the end of a transmission. Moreover, v_i^* is set to the size of the congestion window. Finally, the total downloaded volume $V_i^* = \int_0^{T_i} v_i^*(t)dt$ is proportional to the average download rate R_i as stated in Equation 1.

The second data structure to coordinate the download is the *partition rule*. For each segment between SMS and LAS, the partition rule stores to which subconnection the segment has been assigned. TCP-PARIS uses the reliable data delivery mechanism of each TCP subconnection to ensure the delivery of all (assigned) segments, i.e., that missing segments are detected and retransmitted. Thus, upon receiving a segment, a subconnection uses the partition rule to distinguish between missing and unassigned segments: if a subconnection receiver receives a segment with sequence number m and the last received segment was k, the receiver checks its partition rule whether any segment l with $k < l < m$ has been assigned to it. If so, the receiver must assume that segment l is lost and retransmit the segment.

[1] we assume that all subconnections use the same segment size

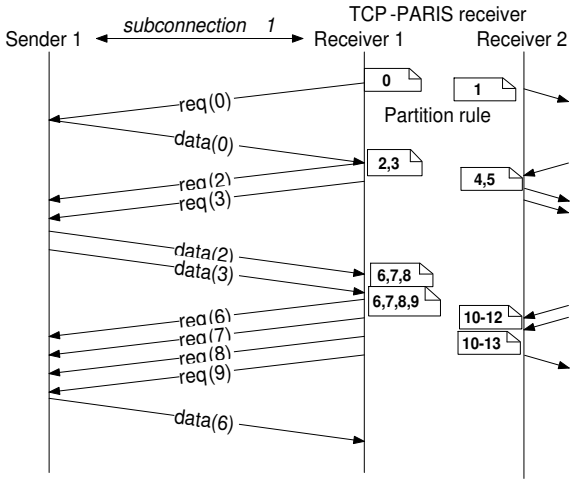
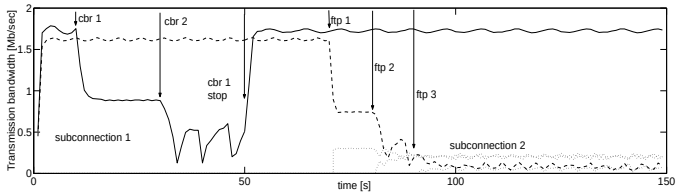
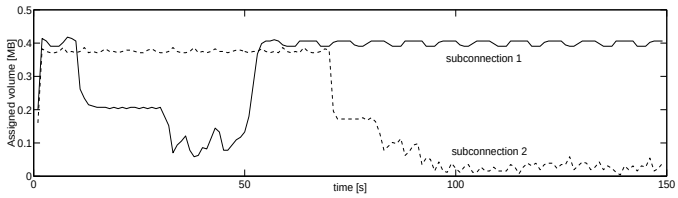


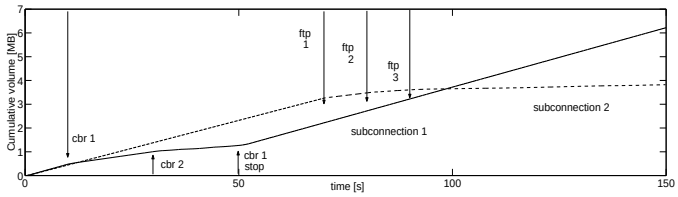
Figure 4. Data delivery with TCP-PARIS(R)



(a) Transmission bandwidth



(b) Assigned volume



(c) Cumulative Volume

Figure 5. Dynamics of a single flow

3.2 Integration with TCP

Here, we describe 2 realizations of TCP-PARIS, one with receiver-based and one with sender-based TCP.

Receiver-based TCP [10, 12] is a request-reply protocol designed to improve response times for web traffic [10], to improve performance for wireless links [12], and to reduce state and overhead for web servers [10]. In receiver-based TCP, the receiver is responsible for congestion control and reliable delivery by requesting individual segments from the sender. The number of segments allowed to be requested is controlled by a congestion window. Upon detecting missing segments, the receiver re-requests them and performs congestion window reductions and timeouts in analogy to sender-based TCP. The implementation of TCP-PARIS with receiver-based TCP subconnections is simplified by the local availability of all status information and all main events at the receiver. In particular, receiver-driven TCP's use of receivers to control the data delivery cycle provides a natural mechanism for TCP-PARIS' receiver to coordinate data delivery across subconnections. Figure 4 depicts the interactions for a TCP-PARIS receiver coordinating the download of two subconnections. The figure shows that the TCP-PARIS receiver coordinates the download via the partition rules. At the sender, no modifications are necessary for TCP-PARIS as the senders continue to deliver the packets that are required by the receiver.

Today, however, most TCP implementations have sender-based control. Thus, an information exchange between the TCP-PARIS receiver and each subconnection sender is necessary. First, the TCP-PARIS receiver must be informed about the number of segments it must assign. Therefore, we piggyback the congestion window size ($cwnd$) in every data segment from the sender to the receiver. Then, the receiver ensures that the partition rule contains $min(max\_cwnd, 2 \cdot cwnd)$ segments. Since the size of congestion window has an upper bound (max_cwnd) and the congestion window at most doubles in one RTT, this assignment ensures that the server does not starve and thus fulfills Equation 3. Second, a copy of the partition rule is piggybacked from the receiver to the sender in the acknowledgment that will allow the sender to determine which segments to send and to distinguish non-assigned from lost segments. Thus, with sender-based TCP, modifications are required at both the sender and receiver side. Both sender and receiver must store a copy of the partition rule, and the sender must determine whether a segment is in the partition rule prior to sending. Following our concept of designing a transport-layer protocol that requires as few changes in the application as possible, we integrated this check inside the sending routine of the TCP sender. Alternatively, the sender-side modifications could also be made on top of the transport layer: by exposing the partition rule beyond the transport layer, e.g., to the application, the application could send only those segments defined in the partition rule. Since this solution would only alleviate, but not eliminate the sender-side transport-layer changes, we opted for an implementation that maintains the transparency between

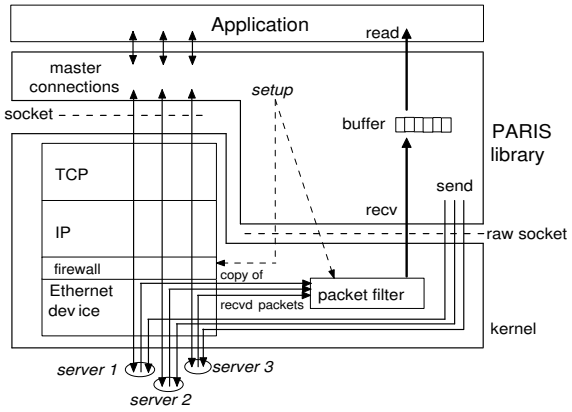


Figure 6. Implementation at the client

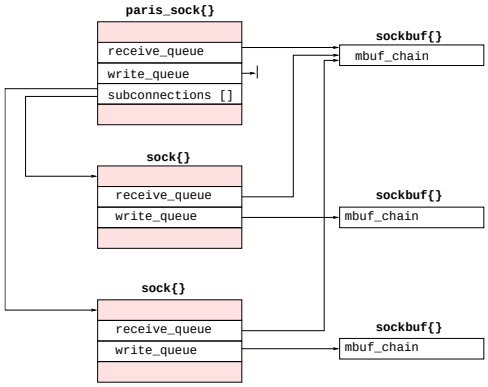


Figure 7. Socket I/O with TCP-PARIS (client side)

applications and the transport layer.

Figure 5 shows the TCP-PARIS behavior for a parallel download with 2 subconnections using ns-2. Figure 5(a) shows the transmission bandwidth $r_i(t)$ of the 2 TCP subconnections. The bandwidth of the subconnections varies as a function of the cross traffic: between $t = 10$ sec and $t = 50$ sec, a CBR flow with rate 768 kb/sec interferes with subconnection 1. At $t = 30$ sec, another CBR flow starts from server 1 with rate 128 kb/sec. At $t = 70$ sec, $t = 80$ sec and $t = 90$ sec, 3 ftp sessions are started that interfere with the bandwidth of subconnection 2. Figure 5(b) shows the *assigned* volume $v_i^*(t)$ of the subconnections during the download time. The assigned volume follows the transmission bandwidth in Figure 5(a). A slicing protocol, in comparison, would create a square-wave-like function that approximates the behavior of TCP-PARIS. Finally, Figure 5(c) shows the cumulative assigned volume during the download, $V_i^* = \int_0^T v_i(t)dt$. Initially, both volumes increase with the same rate. At $t = 10$ sec, the increase of volume 1 slows down because of the drop in available bandwidth rate. Between $t = 50$ and $t = 70$ sec, TCP-PARIS increases both volumes again at the same rate. Finally, after $t = 70$ sec, when the rate of subconnection 2 is reduced due to the ftp traffic, volume 2 increases only slightly compared to volume 1. Therefore, at the end of the download, the downloaded volumes of $V_1 = 6.1$ MB and $V_2 = 3.9$ MB reflect the average transmission bandwidth of the respective subconnections, as required in Equation 1, even though the transmission bandwidth varied dynamically during the download.

4 Implementation

We implemented both receiver- and sender-based TCP-PARIS for the Linux 2.4.19-web 100 kernel. Due to space limitations, we only highlight the parts that need special attention. The API of TCP-PARIS offers a new protocol type that is instantiated using the standard socket call: `psock = socket(AF_INET, SOCK_PARIS, 0)`. Moreover, the `connect` call is modified to pass multiple server addresses instead of one. Afterwards, data is exchanged via the standard `send` and `receive` calls. Thus, since the API is analogous to a single-flow download, an application can be migrated to parallel downloads with only two changes in the application code.

Figure 6 shows the implementation of client with receiver-based TCP subconnections. We first implemented receiver-based based on IP firewalls and packet filters, a frequently used technique, e.g., in Sting [18], and then extended it with the download coordination of TCP-PARIS. For various reasons (e.g., security), this implementation first establishes a set of master subconnections to the sender and exchanges initial requests via these connections. Then, after a successful completion, the firewall and the packet filter are set up and the data transfer is made via the raw sockets. Note that a single packet filter is used per download. Thus, the data from the different subconnections is automatically merged. The implementation of the sender is analogous.

The implementation of sender-based TCP-PARIS adds and modifies the data structures within the kernel. Figure 7 shows the modification of the read- and write queues of the subconnections at the receiver. Each subconnection has its own write queue to send information to the server. However, the read queue is shared, allowing merging the data from multiple subconnections to be passed to the application. Moreover, the information exchange between senders and receiver is implemented as two TCP options, requiring 8 bytes for the $cwnd$ and up to 40 bytes per ack for the partition rule.

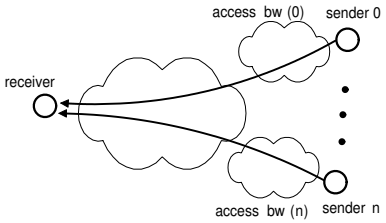


Figure 8. Setup for single downloads

#	bw_i [kbps]	1 MB file t_{PARIS}	1 MB file $t_{slicing}$
1	1024	8.27	8.27
2	56 / 968	14.3/7.67	36.35/6.43
2	128 / 896	8.92/7.9	15.92/6.94
2	256 / 768	8.71/7.82	8.07/7.98
2	512 / 512	8.82/8.8	8.85/8.79
4	56/128/256/584	13.4/12.8/8.2/7.1	36.3/15.9/8.0/3.5
4	56/128/328/512	13.8/13/8.6/7.3	36.3/15.9/6.2/4.0
4	56/56/56/856	12.9/12.9/12.9/7.03	36.3/36.3/36.3/2.4
4	256/256/256/256	8.3/8.3/8.3/8.3	8.3/8.3/8.3/8.3
8	8*128	7.99 - 7.79	n/a
8	7*56/632	11.66 - 6.28	n/a

Table 1. Download times for TCP-PARIS and BitTorrent for a 1 MB file, as a function of the number of subconnections and the bandwidth distribution

5 Evaluation

In this section, we systematically study the performance of TCP-PARIS using ns-2. First, we study the ability of a single TCP-PARIS download to achieve the performance objectives as a function of the volume and the bandwidth distribution. Second, we study the effects of multiple interacting TCP-PARIS downloads in a distributed environment. In both cases we compare TCP-PARIS to related protocols, such as different slicing implementations and coding protocols.

5.1 Single download

We simulate a parallel download with n subconnections, as depicted in Figure 8. Round-trip times are 20 ms, the core bandwidth is 100 Mb/sec, and we vary the server-side bottleneck bandwidth such that the total download bandwidth $bw = \sum_i bw(i)$ is 1Mb/sec. Thus, an optimal download protocol would result in the same download time for a given volume independent of the number of subconnections and their bandwidth distribution.

Table 1 shows the download times for TCP-PARIS and BitTorrent for a 1 MB file as a function of the downloaded volume, the number of subconnections and the bandwidth distribution. TCP-PARIS is able to achieve the performance objective within 10%, except for downloads with 56 kb/sec links. The larger differences of the latter, however, arise because of the simplicity of our simulation. With all queues initially empty, those slow connections open the congestion window far beyond the bandwidth-delay product and decrease it after receiving duplicate acks. As discussed in Section 3, the assigned volume v_i^* is adjusted only after several round-trip times, resulting in an increased download time. However, these effects disappear in simulations where the queues are not empty, as the congestion control finds its equilibrium faster. BitTorrent, in contrast, lacks the ability of a fine-grained volume adjustment. With 1MB files and 256kB slices, it is unable to use more than 4 parallel connections. Moreover, it must assign at least 25% of the volume to each server. Thus, the performance objectives are met only if the bandwidth distribution ratio matches the volume ratio, e.g. in row 4, $bw_1/bw_2 = 1/3$ and $v_1 = 25\%$ and $v_2 = 75\%$. For odd bandwidth ratios, however, TCP-PARIS outperforms BitTorrent by up to a factor of 4.3 (row 2).

Table 2 shows the download times for TCP-PARIS and BitTorrent for a *3* MB file as a function of the downloaded volume, the number of subconnections and the bandwidth distribution. Compared to the Table 1, where the file size was 1 MB, we note that the differences between TCP-PARIS and BitTorrent are less pronounced. The last 2 columns show the volume distribution of TCP-PARIS and BitTorrent: due to the larger file, BitTorrent is able to achieve a similar volume partitioning as TCP-PARIS. The exception is for $n = 8$ subconnections, where BitTorrent is unable to adjust the volumes to large number of subconnections. These results show that TCP-PARIS achieves a near-to-optimal download because of the integration of congestion control information. In contrast, the performance of BitTorrent depends on relationship of file size and bandwidth. While BitTorrent shows a good average performance in typical current peer-to-peer scenarios, the download time may be significantly higher for specific downloads. Moreover, we ignore how BitTorrent will perform in future high-speed networks where the assumption that the maximal congestion window is limited to 64 KB may no longer hold, whereas the integration with TCP-PARIS ensures a near-to-optimal download.

Finally, Figure 9 shows the download time of TCP-PARIS for larger file downloads. The x-axis denotes a file size multiplier x that is repeated in the labels of the individual lines. For example, for the line label denoting $x * 1MB$, the file size is x times 1 MB, where x is modified along the x-axis. The y-axis denotes the download time, each line represents a different bandwidth distribution (see legend). The main impression in this figure is that the download time increases linearly with the downloaded file size. Thus, this

#	bw_i [kbps]	3 MB file			
		t_{PARIS}	$t_{slicing}$	v_{PARIS}	$v_{slicing}$
1	1024	24.7	24.7	100	100
2	56/968	26.5/25.2	36.5/23.4	8/92	8/92
2	128/896	26.47/23.8	32.0/23.0	14/86	17/83
2	256/768	24.9/23.8	25.1/24.0	26/74	25/75
2	512/512	24.9/24.8	24.9/24.9	50/50	50/50
4	56/128/256/584	26.5/26.1/24.2/23.8	36.5/32.0/24.1/21.8	8.3/20/25/45	8.3/16.6/25/50
4	56/128/328/512	26.5/26.1/24.0/23.8	36.35/32.1/24.9/20.2	8.3/14/30/50	8.3/16.6/33.2/41.8
4	56/56/56/856	27.6/27.6/27.6/21.8	35.6/35.6/35.6/21.8	8/8/8/75	8/8/8/75
4	256/256/256/256	25.6 each	25.6 each	25 each	25 each
8	8*128	24.16	32.09-15.9	12.5 each	$4 * 8.3/4 * 16.6$
8	7*56/632	27.78 - 19.32	36.35-16.37	$7 * 8.1/43.3$	$7 * 8.3/42$

Table 2. Download times for TCP-PARIS and BitTorrent for a 3 MB file, as a function of the number of subconnections and the bandwidth distribution

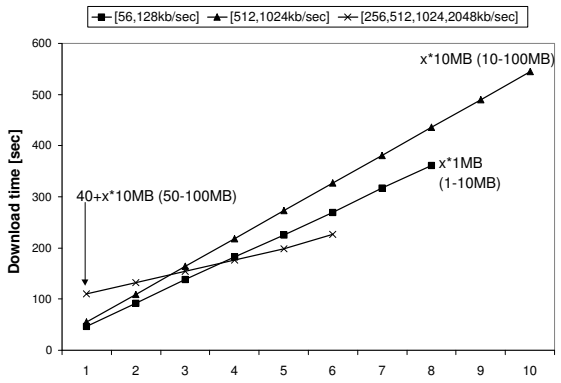


Figure 9. Download time for TCP-PARIS as a function of the bandwidth distribution and the volume.

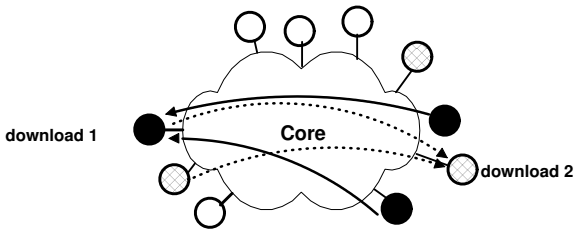


Figure 10. Setup for distributed system experiments

figure enhances the fact that TCP-PARIS is able to achieve the near-to-optimal download independent of the file size.

5.2 Evaluation of the system performance

In this section, we evaluate TCP-PARIS in a distributed system where multiple applications share network and server resources. As depicted in Figure 10, the system consists of a high-speed core of 100Mb/sec links and end systems attached via slower access links. 50 applications are randomly assigned to end systems, where each application consists of 1 client and n servers. Thus, clients and servers from different application may be located on the same end system, sharing server and network resources. The access link bandwidth has uniform distribution of among 56, 128, 256, 512 kb/sec and 1 Mb/sec, which approximates measurements from [19]. We perform experiments with 2 sets of file size distributions. The first distribution, denoted as *large*, approximates a peer-to-peer system [9]: 90% of the files have a uniform distribution of 10-100MB, 3% of 1-10MB and 7% of 100MB-1GB. The second distribution, denoted as *small*, uses the same distributions among file size ranges of 100KB-1MB (90%), 1-10MB (3%) and 10-100MB (7%). We assess the download time as a function of the number of parallel connections n for TCP-PARIS as well as BitTorrent, a Tornado Code based protocol, as well as an unoptimized slicing protocol with slice sizes of 50kB and 1MB slices. In these unoptimized protocols, the client requests the next slice only after the complete download of the previous slice. Thus, the two protocols show the effects of starvation for two different slice sizes.

Figure 11 shows the average download time in minutes as a function of the number of parallel connections n for the large distribution. Compared to download from a single peer, TCP-PARIS reduces the mean download time with $n = 2$ parallel connections from 6.4 min to 4.1 min, a net reduction of 36%. Using $n = 3$ instead of 2 parallel connections reduces the download time further to 3.7 min, a reduc-

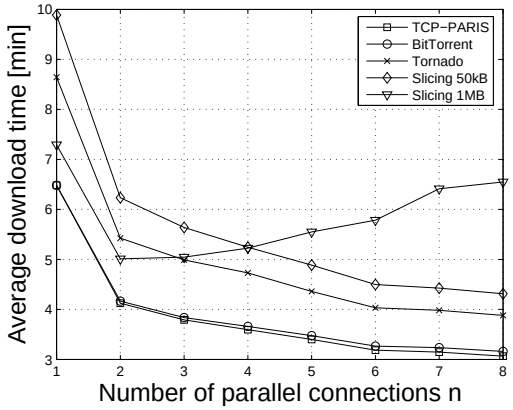


Figure 11. Average download time for large file size distribution

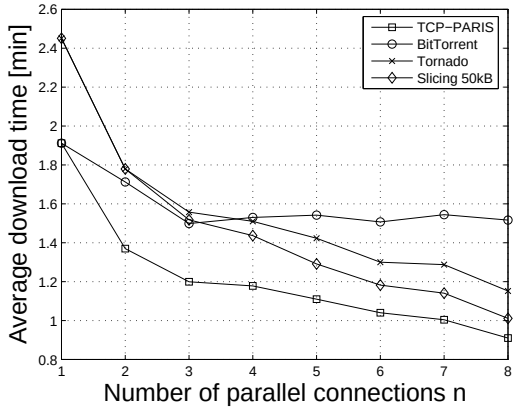


Figure 12. Average download time for small file size distribution

tion by 5%, and for $n = 8$, TCP-PARIS reduces the download time to 3 min, a total reduction compared to $n = 1$ by 52.8%. The reduction decreases with larger n in our simulation because the unused network bandwidth decreases steadily. With $n = 1$, every peer node statistically hosts an application end point. By increasing n but maintaining the number of peer nodes, no additional capacity is added to the network. Therefore, the performance gains of larger n are only due to a better use of the available network resources.

BitTorrent achieves a similar performance as TCP-PARIS, with differences increasing up to 3% when parallelism is increased. A protocol based on Tornado Codes, in contrast, shows a difference of 20-35%, as the volume to be downloaded increases due to the redundancy, a net difference of 1-2 minutes. Finally, the two non-optimized slicing protocols show the tradeoff between small and large slices: small slices incur frequent transmission gaps whereas large slices are less flexible to coordinate the volume. Therefore, with 50 KB slices, the download time difference decreases from 29% to 34% with increased parallelism, whereas it increases from 12% to 50% for 1 MB slices.

Figure 12 shows the average download time for the small distribution. Due to the smaller files, the average download time is reduced to seconds. Similar to the experiment with larger files, TCP-PARIS is able to reduce the download time by 51%, from 1.9 sec to 0.94 sec. In contrast, BitTorrent shows now differences of up to 50% for $n = 8$ compared to TCP-PARIS, first because the larger slice sizes limit the ability to use multiple streams (e.g., for a 1MB file, $n = 4$), and second because the larger assigned volume limits the ability to ensure that all subconnections terminate at the same time. The code-based protocol shows a similar difference compared to TCP-PARIS as with large files, with 20%-35% larger download times, and a protocol with

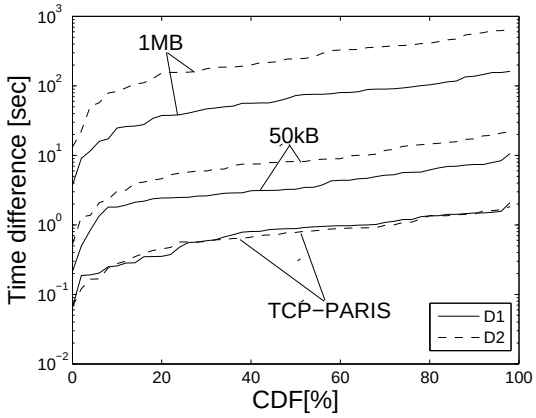


Figure 13. Differences in the download time among subconnections.

50 KB slices has a larger download time of 10%-42%.

These results show that TCP-PARIS is able to reduce the download time independent of the file size distribution, whereas other protocols, in particular BitTorrent, are limited in their ability to coordinate the download for smaller files, i.e., when the download time is in the order of seconds. In addition to the obvious advantage for the client, the faster download also implies that server-side resources (e.g., access link bandwidth) are freed faster.

Next, we assess the influence of the access link bandwidth distribution onto the protocol performance. As shown for a single flow, heterogeneous access link speeds increase the demand on a fine-grained volume adjustment. Here we compare the difference in the termination time of the subconnections of TCP-PARIS to slicing protocols with 50 KB

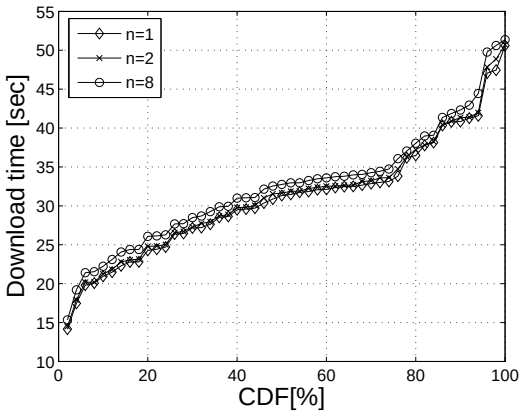


Figure 14. Download time for receiver-side bottlenecks

and 1 MB slices with 2 bandwidth distributions: $D1$ has a uniform distribution of among 56, 128, 256, 512 kb/sec and 1 Mb/sec (avg: 480 kb/sec), $D2$ has a uniform distribution of among 56 kb/sec and 1 Mb/sec (avg: 540 kb/sec). Figure 13 depicts the time differences on a log-scaled y-axis for $n = 2$ parallel connections. The fine-grained adjustment of TCP-PARIS limits the time differences to < 2 sec for both distributions. The differences for a slicing protocol with 50 KB exceed the differences of TCP-PARIS by almost an order of magnitude. The differences with $D2$ increase by a factor of 2 compared to $D1$. Finally, for 1 MB slices, the differences increase by another order of magnitude, and also the difference between $D1$ and $D2$ increases to a factor of 4. Thus, in accordance with the results for a single flow, these differences of 1 MB slices significantly hinder the protocol to achieve an efficient parallel download.

Finally, Figure 14 shows the download time distribution for TCP-PARIS for receiver-side bottlenecks. In particular, use the small file distribution and the same bandwidth distribution, but deliberately assign clients only to slow links. The figure, which plots the CDF of the download time of the 50 applications shows that TCP-PARIS does not decrease the download performance when using n parallel connections. In contrast, the download time is slightly increased, corresponding to results found in literature (e.g. [11]).

6 Conclusions

This paper presents TCP-PARIS, a novel protocol for parallel download from n replica servers to a single client. TCP-PARIS' integration of congestion control information ensures a near-optimal download coordination independent of file size and network bandwidth distributions. Protocol simulations as well as simulations with multiple interacting flows using TCP-PARIS show a download time reduction by up 52% compared to single download, and up to 52% compared to related parallel download protocols, such as code-based or slicing protocols.

Our implementation shows that TCP-PARIS can be integrated with minor changes in the application code. These changes are limited to the connection establishment, but the fact that the data is received from multiple sources is hidden from the application. Therefore, TCP-PARIS is easily integrated with new and legacy applications.

Finally, TCP-PARIS opens new opportunities to address fairness and performance in parallel downloads. So far, TCP-PARIS does not modify the rate of a subconnection and can thereby achieve an n-fold throughput of a single-flow download. In future work, we will study means to modify the throughput of the subconnections to mitigate performance and fairness. Such a modification is only possible with an integration of the transport layer and would contrast current peer-to-peer fairness model that base on tit-for-that fairness objectives. Moreover, we will exploit the streaming behavior to assess TCP-PARIS' ability to support real-time parallel downloads for multimedia content in future work.

References

[1] B. Allcock, J. Bester, J. Bresnahan, A. Chervenak, I. Foster, C. Kesselman, S. Meder, V. Nefedova, D. Quesnel, and S. Tuecke. Data management and transfer in high-performance computational grid environments. *Parallel Computing*, 28(5):749–771, May 2002.

[2] J. Byers, J. Considine, M. Mitzenmacher, and S. Rost. Informed content delivery across adaptive overlay networks. In *Proceedings of ACM SIGCOMM'02*, Pittsburgh, PA, Aug. 2002.

[3] J. Byers, M. Luby, and M. Mitzenmacher. Accessing multiple mirror sites in parallel: Using tornado codes to speed up downloads. In *Proceedings of IEEE INFOCOM'99*, New York, NY, Mar. 1999.

[4] J. Byers, M. Luby, M. Mitzenmacher, and A. Rege. A digital fountain approach to reliable distribution of bulk data. *Computer Communication Review*, 28(4), Oct. 1998.

[5] M. Castro, P. Druschel, A. Kermarrec, A. Nandi, A. Rowstron, and A. Singh. SplitStream: High-bandwidth multicast in a cooperative environment. In *Proceedings of SOSP'03*, Bolton Landing, NY, Oct. 2003.

[6] B. Cohen. Incentives build robustness in Bittorrent, May 2003. http://bittorrent.com/bittorrentcon.pdf.

[7] S. Floyd. HighSpeed TCP for Large Congestion Windows. Request for Comment 3649, Dec. 2003.

[8] C. Gkantsidis, M. Ammar, and E. Zegura. On the effect of large-scale deployment of parallel downloading. In *Proceedings of IEEE Workshop on Internet Applications (WIAPP'03)*, San Jose, CA, June 2003.

[9] K. Gummadi, R. Dunn, S. Saroiu, S. Gribble, H. Levy, and J. Zahorjan. Measurement, modeling and analysis of a peer-to-peer file-sharing workload. In *Proceedings of SOSP'03*, Bolton Landing, NY, Oct. 2003.

[10] R. Gupta, M. Chen, S. McCanne, and J. Walrand. A receiver-driven transport protocol for the web. In *Proceedings of the 4th INFORMS Telecommunication Conference*, Boca Raton, FL, Mar. 2000.

[11] T. Hacker, B. Athey, and B. Noble. The end-to-end performance effects of parallel TCP sockets on a lossy wide-area network. In *Proceedings of IPDPS'02*, Fort Lauderdale, FL, Apr. 2002.

[12] H. Hsieh, K. Kim, Y. Zhu, and R. Sivakumar. A receiver-centric transport protocol for mobile hosts with heterogeneous wireless interfaces. In *Proceedings of ACM MobiCom'03*, San Diego, CA, Sept. 2003.

[13] D. Kostic, A. Rodriguez, J. Albrecht, and A. Vahdat. Bullet: high bandwidth data dissemination using an overlay mesh. In *Proceedings of SOSP'03*, Bolton Landing, NY, Oct. 2003.

[14] P. Rodriguez and E. Biersack. Dynamic parallel-access to replicated content in the Internet. *ACM/IEEE Transactions on Networking*, Aug. 2002.

[15] P. Rodriguez, A. Kirpal, and E. Biersack. Parallel-access for mirror sites in the internet. In *Proceedings of IEEE INFOCOM'00*, Tel Aviv, Isr, Mar. 2000.

[16] S. Saroiu, P. Gummadi, R. Dunn, S. Gribble, and H. Levy. An analysis of Internet content delivery systems. In *Proceedings of OSDI'02*, Boston, MA, Dec. 2002.

[17] S. Saroiu, P. Gummadi, and S. Gribble. A measurement study of peer-to-peer file sharing systems. In *Proceedings of IS&T/SPIE Conference on Multimedia Computing and Networking (MMCN'02)*, San Jose, CA, Jan. 2002.

[18] S. Savage. Sting: a TCP-based network measurement tool. In *Proceedings of the 2nd USENIX Symposium on Internet Technologies and Systems (USITS'99)*, Boulder, CO, Oct. 1999.

[19] S. Sen and J. Wang. Analyzing peer-to-peer traffic across large networks. In *Proceedings of the Second SIGCOMM Internet Measurement Workshop (IMW 2002)*, Marseille, France, Nov. 2002.

[20] R. Sivakumar, S. Bailey, and R. Grossman. PSockets: the case for application-level network striping for data intensive applications using high speed wide area networks. In *Proceedings of the 2000 ACM/IEEE Conference on Supercomputing*, Dallas, TX, Mar. 2000.