On Transport Layer Support for Peer-to-Peer Networks

Hung-Yun Hsieh and Raghupathy Sivakumar
School of Electrical and Computer Engineering
Georgia Institute of Technology
Atlanta, GA 30332, USA
Email:{hyhsieh, siva}@ece.gatech.edu

Abstract

TCP is the transport protocol used predominantly in the Internet as well as in peer-to-peer networks. However, peer-to-peer networks exhibit very different characteristics from those of conventional client-server networks. In this paper, we argue that the unique characteristics of peer-to-peer networks render TCP inappropriate for effective data transport in such networks. Specifically, we motivate transport layer support for multipoint-to-point connections to address the problem of sources in peer-to-peer networks lacking server-like properties in terms of capacity and availability. We outline several key elements in designing a new transport protocol for supporting effective multipoint-to-point connections. Finally, we present a case study for a multipoint-to-point transport protocol that puts together these design elements in practice. We thus motivate further research along this direction.

1. INTRODUCTION

Over the last few years, the area of peer-to-peer networking has attracted considerable attention. The notion of end-users collaborating to create as well as to consume a richer set of services and contents has been quite well received. Resources shared in a peer-to-peer network are distributed in a decentralized fashion, and directly accessible to any host participating in the network.

While existing works in the area of peer-to-peer networking have typically focused on application layer approaches with vertically integrated solutions, the growing scale and diversity of peer-to-peer networks have called for a common platform to facilitate the development and interoperability of peer-to-peer applications. Several research endeavors have gone into building generic architectures, interfaces, and protocols that can support peer-to-peer networks more effectively [1]–[3].

In this paper, we argue for transport layer support for peer-to-peer networks. TCP (Transmission Control Protocol) has been the predominant transport protocol used in the Internet as well as in peer-to-peer networks. It is designed for a unicast connection between a server and a client. However, peer-to-peer networks exhibit very different characteristics from those of conventional client-server networks. On one hand, the existence of multiple peers with replicated content provides users with multiple potential sources to transfer data from. On the other hand, these peers that act as sources to supply the content typically do not exhibit "server-like" properties due to their limited capacity and transient availability [4]. We argue that using TCP not only prevents the requesting peer from leveraging the existence of multiple sources for achieving potential performance improvement, but also exposes it to the non-server-like behavior of individual sources thus causing performance degradation.

Toward this end, we first motivate the benefits of transport layer support for multipoint-to-point connections in peer-to-peer networks.[1] The transport layer plays a defining role in effective data transport between the source and the destination. However, existing transport protocols support only point-to-point and/or point-to-multipoint (multicast) connections. We discuss the performance benefits in enabling multipoint-to-point data transport in Section 2. We then proceed to outline several key components that should be considered in designing new transport layer protocols for supporting multipoint-to-point connections. We discuss these design elements in Section 3. Finally, we present a case study for a multipoint-to-point transport protocol called R^2CP (Radial Reception Control Protocol). We show in Section 4 that R^2CP encompasses the desired transport layer design and allows for further consideration in peer-to-peer networks. We discuss issues for multipoint-to-point transport protocols and summarize the paper in Section 5.

[1] In reference to the TCP/IP protocol model, the transport layer translates the services provided by the network (internet) layer for use by the application. We note, however, that the argument presented in this paper is also applicable to the session layer in the OSI model. The solution proposed thus can build atop conventional transport protocols, with sufficient support from the latter.

2. MOTIVATION

In this section, we present arguments for supporting multipoint-to-point connections in peer-to-peer networks, from the perspectives of the destination (requesting peer), the source (supplying peer), and the content. While there are several existing applications that can use multiple replicated sources for achieving better performance in terms of faster downloads or resilient streaming at the destination [5]–[8], we discuss why such application layer approaches cannot effectively support multipoint-to-point communication without any modification at the transport layer. We thus motivate transport layer support for multipoint-to-point connections in peer-to-peer networks.

2.1 The Destination (Requesting Peer)

From the perspective of the peer requesting the content, the benefit of maintaining multipoint-to-point communication is the potential for better resource aggregation and fault tolerance by tapping multiple sources, thus achieving *higher access performance*.

Hosts participating in peer-to-peer networks are typically located at the edges of the Internet. It is reported in [4] that more than 70% hosts in the Napster network measured have asymmetric links via dial-up (V.90), ADSL, or cable modems with the uplink bandwidths significantly lower than the downlink ones. Hence it is conceivable that a majority of connections in peer-to-peer networks is bottlenecked at the source, and the performance of these connections can be improved by using multiple sources with replicated content concurrently. Several peer-to-peer applications [5], [6] have started exploring along this direction to provide the requesting peer with better download or streaming performance. Note that, however, a key issue for the destination to support multipoint-to-point connections is the resequencing of data received from multiple sources. Existing approaches have relied on using the hard disk for offline buffering, where data resequencing is performed only after the entire content has been received [7], [8].[2] While such approaches can be used for content downloads, they are not applicable to most other applications that maintain a limited buffer and require the in-sequence delivery service from lower layers. It has been shown in [9] that for such applications the performance achieved can be throttled by the slowest link in the connection if application-layer striping is performed without transport layer support.

Using multipoint-to-point connections is not limited to bandwidth aggregation. It can also allow the requesting peer to mitigate performance degradation due to suboptimal peer selection and transient peer availability. The problem of selecting the "best" source to request data from is non-trivial, especially in peer-to-peer networks where many peers may not be encountered more than once [10] or may not be uniquely identified [11], and they may even provide inaccurate bandwidth information [4]. Similarly, peer transience due to dynamic peer arrivals and departures has been shown in related work to be a serious problem causing disrupted or even aborted communication [4], [11]. The ability to incorporate multiple concurrent sources in a connection thus manifests itself as an ideal solution for these problems, since the performance of a connection is no longer tied to the capacity or availability of individual sources. For example, the departure of any source(s) in the connection will not stall the content delivery at the requesting peer as long as there is still one source available for transmissions. Note that a transport layer protocol supporting multipoint-to-point connections can dynamically maintain the number of sources in the connection, and mask such artifacts in the peer-to-peer network from the application. However, any application layer approach will not be able to address these problems without exposing the same to the application.

2.2 The Source (Supplying Peer)

From the perspective of the peer supplying the content, the benefit of participating in multipoint-to-point communication is the potential for better load balancing in sharing its resource, thus resulting in a *lower average load* on the sources.

It has been reported in [4] that while multiple sources may be available for content supply in peer-to-peer networks, a significant portion of these hosts lacks the high-bandwidth, low-latency profile of a server. For example, in the Napster network measured, 22% of the peers have upstream (outbound) bandwidths that are lower than 100Kbps. Existing systems address this problem by dropping connections or bypassing hosts with low available bandwidths [12]. While such an approach can effectively prevent these low-profile hosts from becoming the bottleneck of the network, it also prevents them from contributing resources (bandwidths) to the network, thus potentially increasing the load on qualified sources. The inefficiency incurred

[2]We note that although the resequencing delay can potentially be reduced using smaller request blocks [7] with appropriate online scheduling (refer to Section 3.3), the communication overheads incurred that increase with decreasing block sizes can make such an approach undesirable.

in utilizing low-profile hosts, however, exists only for unicast connections where the performance of the connection is upper-bounded by the bandwidth of the source. A multipoint-to-point connection, on the other hand, can allow low-profile hosts to be aggregated to support a high-bandwidth connection. It hence makes use of all available resources (however small) that each host in the peer-to-peer network can provide, without sacrificing the quality enjoyed by the requesting peer.

Multipoint-to-point support in peer-to-peer networks not only allows low-profile hosts to contribute, but also encourages participation from hosts with relatively high bandwidths. This is because any content search in peer-to-peer networks typically results in the host with the "highest" bandwidth being chosen. Although it can be argued that hosts are willing to share unused resources [13], overloads in the uplink direction can potentially delay the acknowledgment packets sent by the downlink TCP traffic, thus decreasing the maximum utilization achievable in the downlink and increasing the disincentive to share [14]. It has been shown in [4] that hosts tend to *deliberately* misreport their bandwidths, so as not to serve too many requests from other peers. In a system with multipoint-to-point support, however, such "hot spots" can be alleviated, since the requesting peer, by aggregating resources from the runners-up, can achieve the same performance as that using the best source.[3] Note that a key issue for the source to support multipoint-to-point connections is that no data is redundantly transmitted across multiple sources to the destination. Application layer approaches that perform content coding at the sources have been proposed in content distribution networks [6], [8]. Although these approaches free the requesting peer from involving in coordinating the transmissions of multiple sources, they are not applicable to peer-to-peer networks, since they require the number and distribution (or characteristics) of the sources used to be known a priori for performing content coding or reducing the reception inefficiency. Moreover, it is difficult, if not impossible, to enforce source coding at all autonomous peers. On the other hand, as we discussed in Section 2.1, approaches such as [7] where the receiving application dynamically requests ranges of data from individual sources can suffer from high communication overheads and application complexities without transport layer support.

[3]We note that load balancing using multipoint-to-point connections is different from that using unicast connections, since the latter uses *only* the "second-best" host and hence can potentially result in decreased performance otherwise enjoyed by the requesting peer without load balancing.

2.3 The Content

From the perspective of the content itself, the benefit of using multipoint-to-point communication is the ability to *preserve the integrity* of the content better as it is propagated through the peer-to-peer network, which will in turn benefit both the sources and destinations.

Existing transport layer protocols provide a spectrum of reliability services, including unreliable, partially reliable, and fully reliable services, that can be used by different applications. However, an important issue in distributing the content and thus amplifying the capacity of a peer-to-peer network is reliable content replication. Consider a scenario where a video clip is streamed from one host to another, and the requesting peer later becomes the supplying peer serving other hosts. Since streaming applications typically choose timeliness over reliability, it is possible that the requesting peer has a lossy replication of the original video clip after the streaming is complete. If the lossy copy is streamed to another host without the missing information being restored, the quality of the video will continue to degrade after each replication – eventually rendering the clip unusable. An obvious approach to address this problem is for the application to open a reliable connection after streaming, and retrieves the missing information from the source. However, such an application layer approach increases implementation complexities (the application needs to implement loss detection and recovery) and communication overheads (consider the overheads incurred in TCP when retrieving data in a non-contiguous fashion). While it is non-trivial to design a transport protocol that can support 100% reliability without degrading the application performance otherwise attainable using a partially reliable transport protocol (since the bandwidth used for recovering the lost data may be wasted as far as the application is concerned), such functionality can be easily implemented in the context of multipoint-to-point connections. A transport protocol supporting multipoint-to-point connections can open one more source (in addition to the original data source) dedicated to loss recovery. Such out-of-band loss recovery allows loss recovery to take place without consuming the precious bandwidth available along the data path. In this way, the application can continue to receive data in a timely, partially reliable fashion, but when the connection completes, the data will be reliably replicated to the receiver.

3. TRANSPORT LAYER DESIGN

We have thus far motivated the benefits of transport layer support for multipoint-to-point connections

in peer-to-peer networks. In this section, we discuss several key components in designing a transport layer protocol that can support effective multipoint-to-point connections. We assume that the multipoint-to-point transport protocol needs to support the same in-sequence data delivery semantics as TCP.

3.1 Multiple States

TCP is designed for point-to-point connections where it assumes a single path between the source and the destination. TCP captures the characteristics of the path it traverses such as bandwidth and latency in the form of TCB (Transmission Control Block) state variables such as congestion window and round-trip time, for determining the send rate of the connection. In a multipoint-to-point connection, packets from different sources traverse different paths to the destination. Since hosts in peer-to-peer networks can exhibit a very high degree of heterogeneity in terms of the connection bandwidth and latency [4], maintaining only one set of TCB variables (single state) in a multipoint-to-point connection can render the send rate and hence the achieved throughput suboptimal. Therefore, a key design in a multipoint-to-point transport protocol is to maintain multiple states in accordance with the number of sources (paths) used in the connection. In the context of TCP, multi-state design allows TCP to maintain one TCB for each path, and hence different sources can use different send rates (depending on the characteristics of the underlying path) for transmitting packets to the destination. Note that out-of-order arrivals at the destination due to packets traversing multiple paths will not trigger unnecessary window cutdown in such a multi-state TCP, since congestion control is performed on a per-path basis. Vanilla TCP with single state, on the other hand, will fail to utilize even the slowest path in the connection when operated over multiple paths.

3.2 Decoupling of Functionalities

To incur minimum overheads resulting from the multi-state design, transport layer functionalities should be divided between those associated with individual paths and those pertaining to the aggregate connection. For example, congestion control estimates the bandwidth of the underlying path, and hence should be performed for each path in the connection. On the other hand, buffer management handles the socket buffer, and hence should not be repetitively implemented across multiple states. While the reliable delivery of data can be considered either as a per-path functionality (packets are reliably delivered along each path) or as a per-connection functionality (packets lost on one path can be recovered through retransmissions on another path), there are several advantages in designing reliability as a functionality of the aggregate connection: (i) Since loss recovery can take place along a path different from the one traversed by the lost packet, path shutdown due to the departure of an active source (e.g. peer transience or failure) does not interfere with the reliable delivery of data. (ii) One path can be dedicated to loss recovery, while the others provide only unreliable (or partially reliable) service. In this way, packets lost along the unreliable paths will be recovered "out-of-band" without stalling the progression of data delivery along these paths.

3.3 Packet Scheduling

Since packets traverse different paths from multiple sources to one destination in a multipoint-to-point connection, a key issue in providing in-sequence data delivery to the receiving application is packet scheduling. Out-of-order arrivals not only call for a large resequencing buffer at the receiver, but can also introduce *head-of-line* blocking. It has been shown in [9] that head-of-line blocking can cause significant performance loss in terms of achieving the aggregate bandwidth. As we mentioned in Section 3.1, different paths in a multipoint-to-point connection can exhibit very different characteristics in terms of bandwidth/latency mismatches and fluctuations. A scheduling algorithm that schedules packets based on a pre-determined bandwidth ratio of different paths [15] will apparently suffer from bandwidth fluctuations. Moreover, since different paths can exhibit latency mismatches by more than a factor of four [4], a scheduling algorithm based purely on the bandwidth ratio without taking into consideration the latency mismatch will fail to achieve the optimal performance in peer-to-peer networks. Note that the scheduling algorithm should also handle the dynamic arrivals and departures of sources in a multipoint-to-point connection.

3.4 Receiver-Driven Operation

While any data source may come and go, in a multipoint-to-point connection the invariant is the destination (receiver). Moreover, since the receiver is the common point of different paths in a multipoint-to-point connection, it manifests itself as an ideal location for coordinating packet transmissions of individual sources. Note that the role of the receiver is not limited to performing packet scheduling. The reliability

and congestion control functionalities of the transport protocol can also be driven by the receiver. If the receiver is primarily responsible for the reliable delivery of data from the sources, any failure at the source (e.g. due to peer departure) will have minimal impact on the connection. On the other hand, if congestion control is receiver-driven, the receiver will have instant knowledge of the congestion control parameters such as bandwidth and latency that can be used by the packet scheduling algorithm. In this way, the receiver controls *when*, *which*, and *how much* data should be sent from individual sources. Adding or deleting any source from the connection thus has the mere effect of increasing or decreasing the bandwidth available to the connection, without causing any other undesirable disruptions or stalls. Since the intelligence of the transport protocol is primarily located at the receiver, whenever server migration is possible and desirable, the overheads incurred in synchronizing the states maintained (if any) between the old and the new supplying peers will be minimized.

We have thus far identified *multiple states, decoupling of functionalities, packet scheduling, and receiver-driven operation* as the four key design elements in a multipoint-to-point transport protocol. We hasten to add that there do exist several additional transport layer elements for which an argument can be made in the context of peer-to-peer networks. Examples of such elements include run-time peer selection, server load distribution, and forced reliable replication. Consideration of these other elements is part of our ongoing work.

4. Case Study: The R^{2}CP Protocol

We now present a case study for a multipoint-to-point transport protocol called R^{2}CP (Radial Reception Control Protocol) that puts together the design elements outlined in Section 3 in practice. R^{2}CP was originally proposed in [16] for mobile hosts with heterogeneous (such as 3G and WiFi) wireless interfaces. In the following, we provide a synopsis of the R^{2}CP protocol, and show that R^{2}CP encompasses the desired transport protocol design, allowing it to be considered for peer-to-peer networks.

1. R^{2}CP is a receiver-driven, multi-state transport protocol that supports multipoint-to-point connections. The R^{2}CP destination (receiver) maintains multiple states, each of them corresponds to the single state maintained by individual sources (senders) in the connection. Figure 1 shows the architecture of the R^{2}CP protocol.

2. R^{2}CP is built atop a single-state, point-to-point transport protocol called RCP (Reception Control Protocol). RCP is a TCP clone in its general behavior, including the use of the same window-based congestion control mechanism. However, RCP *transposes* the intelligence of TCP from the sender to the receiver such that the RCP receiver is primarily in charge of the congestion control and reliability. The RCP receiver drives the progression of the connection, while the RCP sender merely responds to the instructions sent by the receiver. It is shown in [16] that RCP is indeed TCP-friendly.

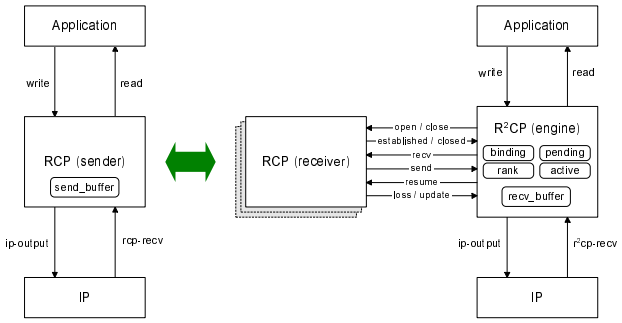

Figure 1. R^{2}CP Architecture

3. Multiple RCP pipes in an R^{2}CP connection are coordinated by the R^{2}CP engine at the receiver. The R^{2}CP engine is responsible for buffer management, flow control, and the reliable delivery of data to the application, while individual RCPs implement congestion control. Note that although RCP by itself is a reliable protocol like TCP, R^{2}CP allows data recovery to occur along the RCP pipe different from the one data was sent. This is achieved in R^{2}CP through *dynamic binding* of the application data (to be requested) and the RCP packets using the `binding` data structure (see Figure 1). Effectively, individual RCPs control *how much* data to request from each sender, while the R^{2}CP engine control *which* data to request from each sender.

4. The R^{2}CP engine performs packet scheduling to coordinate packet transmissions along individual RCP pipes. R^{2}CP uses an RTT-ranked, CWND-based packet scheduling algorithm to minimize out-of-order arrivals at the receiver. Upon receiving the $send()$ call from any RCP pipe that has space in its window for packet requests (note that each RCP pipe is self-clocked like TCP), the R^{2}CP engine uses the `rank` data structure to determine *which* data to request from the corresponding source such that the data requested will arrive in sequence. Since R^{2}CP provides in-sequence data delivery to the application, minimizing out-of-order arrivals can also minimize head-of-line blocking at the receive socket buffer (head-of-line blocking occurs when R^{2}CP is unable to bind more data to any RCP pipe for

requests due to the buffer being filled up). It has been shown in [16] that such a packet scheduling algorithm allows R^2CP to effectively aggregate the bandwidths available along individual paths with bandwidth/latency mismatches and fluctuations.

While we refer interested readers to [16] for a more detailed presentation of the R^2CP protocol and its performance, it is clear from the above discussion that R^2CP follows the design elements outlined in Section 3, allowing it to address the unique characteristics of peer-to-peer networks such as peer heterogeneity and peer transience. Our ongoing work includes developing R^2CP to support load balancing, peer selection, and reliable replication mechanisms for use with peer-to-peer networks.

5. ISSUES AND SUMMARY

While we have made a case for transport layer support for multipoint-to-point connections in peer-to-peer networks, there are several issues that need to be addressed such as potential server overload and network overload due to the greedy use of multipoint-to-point connections. In particular, the need for a sound fairness model to avoid network overload can be of importance when the bottleneck occurs in the backbone of the network. Related work [17] has investigated this issue and proposed several fairness models for use with multipoint-to-point connections – with no consensus reached yet. Note that the design elements outlined in Section 3 allow the fairness model to be seamlessly incorporated in a multipoint-to-point transport protocol. For example, the receiver is the ideal location to enforce the fairness model across multiple pipes. The "decoupling of functionalities" design allows different congestion control mechanisms such as [18] to be used for secondary pipes such that the aggregate connection does not exceed its fair share. We recall from Section 2 that even when the fairness model prevents the requesting peer from enjoying higher throughput than that achievable using a point-to-point connection, a multipoint-to-point transport protocol can still provide significant performance benefits.

In this paper, we argue for transport layer support for peer-to-peer networks. We focus on the benefits of enabling multipoint-to-point communication from the perspectives of requesting peers, supplying peers, and content replication. We first show that existing application layer solutions cannot effectively address the challenges in peer-to-peer networks and support multipoint-to-point communication, whereas a multipoint-to-point transport layer protocol can be a power building block in peer-to-peer networks. We then present several design components for developing a transport layer protocol with multipoint-to-point support. Finally, we present a multipoint-to-point transport protocol called R^2CP that shows potential in addressing the unique characteristics of the peer-to-peer networks. We hence motivate further investigation along this direction.

REFERENCES

[1] Global Grid Forum, http://www.gridforum.org.

[2] F. Dabek, B. Zhao, P. Druschel, J. Kubiatowicz, and I. Stoica, "Towards a common API for structured peer-to-peer overlays," in *Proceedings of International Workshop on Peer-to-Peer Systems (IPTPS)*, Berkeley, CA, USA, Feb. 2003.

[3] J. Eriksson, M. Faloutsos, and S. Krishnamurthy, "PeerNet: Pushing peer-to-peer down the stack," in *Proceedings of International Workshop on Peer-to-Peer Systems (IPTPS)*, Berkeley, CA, USA, Feb. 2003.

[4] S. Saroiu, P. Gummadi, and S. Gribble, "A measurement study of peer-to-peer file sharing systems," in *Proceedings of SPIE Conference on Multimedia Computing and Networking (MMCN)*, San Jose, CA, USA, Jan. 2002.

[5] Kazaa, http://www.kazaa.com.

[6] CenterSpan, http://www.centerspan.com.

[7] P. Rodriguez and E. Biersack, "Dynamic parallel-access to replicated content in the Internet," *IEEE/ACM Transactions on Networking*, vol. 10, no. 4, pp. 455–464, Aug. 2002.

[8] 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*, New York, NY, USA, Mar. 1999.

[9] 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.

[10] D. Bernstein, Z. Feng, B. Levine, and S. Zilberstein, "Adaptive peer selection," in *Proceedings of International Workshop on Peer-to-Peer Systems (IPTPS)*, Berkeley, CA, USA, Feb. 2003.

[11] R. Bhagwan, S. Savage, and G. Voelker, "Understanding availability," in *Proceedings of International Workshop on Peer-to-Peer Systems (IPTPS)*, Berkeley, CA, USA, Feb. 2003.

[12] LimeWire, http://www.limewire.com.

[13] D. Anderson, J. Cobb, E. Korpela, M. Lebofsky, and D. Werthimer, "SETI@home: An experiment in public-resource computing," *Communications of the ACM*, vol. 45, no. 11, pp. 56–61, Nov. 2002.

[14] M. Feldman, K. Lai, J. Chuang, and I. Stoica, "Quantifying disincentives in peer-to-peer networks," in *Proceedings of Workshop on Economics of Peer-to-Peer Systems*, Berkeley, CA, USA, June 2003.

[15] D. Xu, M. Hefeeda, S. Hambrusch, and B. Bhargava, "On peer-to-peer media streaming," in *Proceedings of IEEE ICDCS*, Vienna, Austria, July 2002.

[16] H.-Y. Hsieh, K.-H. Kim, Y. Zhu, and R. Sivakumar, "A receiver-centric transport protocol for mobile hosts with heterogeneous wireless interfaces," in *Proceedings of ACM MOBICOM*, San Diego, CA, USA, Sept. 2003.

[17] P. Karbhari, E. Zegura, and M. Ammar, "Multipoint-to-point session fairness in the Internet," in *Proceedings of IEEE INFOCOM*, San Francisco, CA, USA, Apr. 2003.

[18] A. Kuzmanovic and E. Knightly, "TCP-LP: A distributed algorithm for low priority data transfer," in *Proceedings of IEEE INFOCOM*, San Francisco, CA, USA, Apr. 2003.