[SZTST02] *Jens Schmitt, Michael Zink, Steffen Theiss, Ralf Steinmetz;* **Improving the Start-Up Behaviour of TCP-Friendly Media Transmissions;** International Network Conference (INC'02). Plymouth, UK, July 2002, S. 73-80.

Improving the Start-Up Behaviour of TCP-friendly Media Transmissions

Jens Schmitt, Michael Zink, Steffen Theiss, Ralf Steinmetz

KOM, Darmstadt University of Technology, Merckstr. 25, D-64283 Darmstadt, Germany
email: {Jens.Schmitt, Michael.Zink, Steffen.Theiss, Ralf.Steinmetz}@kom.tu-darmstadt.de

Abstract

The Internet has built its success story to a large degree on the Transmission Control Protocol (TCP). Since TCP still represents the by far most important transport protocol in the current Internet traffic mix, new applications like media streaming need to take into account the social rules implied by TCP's congestion control algorithms, i.e., they need to behave *TCP-friendly*. One problem of this insight is that these new applications are not always well served by inheriting TCP's transmission scheme. In particular, TCP's initial start-up behaviour is a problem for streaming applications. In this paper, we try to address this problem by proposing a *reflective server design* which allows to do inter-session congestion control, i.e., to share network performance experiences between sessions to make informed congestion control decisions. Since our target application is media streaming, we show the design in the framework of a media server, which means in particular that we employ not TCP itself but a TCP-friendly transmissions scheme.

Keywords

TCP-friendliness, Slow-Start, Media Streaming

1 Introduction

TCP's congestion control involves two basic algorithms: slow start and congestion avoidance. During slow start (SS) a sender exponentially increases its sending window during each round trip time (RTT)[1] starting with a window size of 1 to trial the available bandwidth in the network. It thus makes no assumptions and tries to find out fast what could be its fair share of the available bandwidth. Once it encounters an error, either due to a retransmission time-out or due to 3 consecutive duplicate acknowledgments (fast retransmit), it halves its slow start threshold (sstresh) and does another SS. This repeats until slow start reaches sstresh without any losses, then the congestion avoidance (CA) phase is started. In CA the sender still probes the network for more capacity but now at a linear increase per RTT.

For new multimedia applications like media streaming TCP has several drawbacks:

- *retransmissions* are unnecessary since old (retransmitted) data is usually worthless for streaming applications,
- the *bandwidth* resulting from TCP's window-based congestion control algorithms tends to *oscillate* too much for streaming applications,
- the *initial slow start behaviour* is the exact opposite of what streaming applications would desire, namely an initially high rate that allows to fill the playback buffer such that later rate variations can be accommodated by the smoothing effects of the buffer.

This is why many of these applications employ UDP (User Datagram Protocol) as a transport protocol. However, UDP does not have a congestion control. That is why a number of TCP-friendly congestion control schemes have been devised for UDP transmissions. The definition of TCP-friendliness is phrased as "achieve fairness with concurrent TCP transmissions, i.e., achieve the same long-term average throughput as a TCP transmission".

1. For simplicity we neglect the dampening effect on the exponential increase due to ACK clocking.

If TCP-friendliness is accepted as a MUST in the Internet it needs to be observed that, while TCP-friendly transmission schemes can avoid the problems of retransmissions and unsteady bandwidth availability to some degree (we will discuss some proposals below), all TCP-friendly protocols inherit TCP's gross start-up behaviour resulting from slow start. Note that TCP's start-up behaviour has for some time been realized as a problem for transfers of short duration as typically seen for HTTP requests. Yet, also for long-term streaming applications in contrast to long-term file transfers the initial transmission performance is of high importance, since they need to present transmitted data (more or less) immediately to the user and it might be especially dissatisfying if the start of a media transmission is badly disturbed or heavily delayed due to a slow filling of the playback buffer (which might make the consumer switch away again). Besides, promising optimizations of media distribution systems like patching may involve short transfers, too (Hua et al. 1998).

On a higher level of abstraction one could argue that TCP is transiently unfair to new sessions which are still in their probing phase. Ideally, one would wish for a new session to start sending with its fair share and immediately go into a CA-like phase. The question now is how could we make a step towards this ideal behaviour. Since our target application is media streaming, we may assume that we have high-performance servers streaming the media towards a large number of clients. We are thus in a situation where TCP's zero knowledge assumption about the network state at the start of a new transmission towards a client is unnecessarily limiting since such a server could take advantage of the probing of past and concurrent transmissions to the same or "*similar*" clients. The server could thus improve its congestion control decisions by *reflecting* on past decisions / experience and could start the transmission at a higher rate avoiding the SS phase altogether. Of course, care must be taken to back-off from this rate immediately if the estimation of available bandwidth turns out to be erroneous.

2 Related Work & Own Contribution

2.1 TCP-Friendly Transmission Protocols

The design of TCP-friendly transmission protocols has recently experienced a lot of attention. A nice overview can be found in (Widmer et al. 2001). Their basic rationale is to avoid retransmissions and to improve TCP's oscillating bandwidth behaviour by smoothing the available bandwidth to a session. There is mainly two flavours:

- window-based schemes like (Bansal and Balakrishnan 2001, Jin et al. 2001),
- rate-based schemes like (Rejaie et al. 1999, Floyd et al. 2000, Rhee et al. 2000).

While window-based schemes inherit TCP's favourable self-clocking characteristic and can generally be assumed to react faster to dynamic changes in available bandwidth, rate-based schemes usually achieve a smoother transmission scheme which makes them more favourable for streaming applications. Furthermore, the rate-based TFRC (TCP-Friendly Rate Control) has been shown to react relatively fast to changes and has been extended to the multicast case (Widmer and Handley 2001). For these reasons we chose TFRC as the TCP-friendly transmission scheme which shall be integrated into our reflective server design.

2.2 Inter-Session Congestion Control

Directly related to our work is what we call inter-session congestion control. These proposals consider network performance experience from other concurrent or past sessions for their congestion control decisions. (Savage et al. 1999) introduces what they call inter-host congestion control and give some nice introductory motivation for the efficiency gains that may be achievable. Their sharing of congestion control information is solely based on what they call *network locality*, i.e., only destinations that have a common 24-bit subnet mask share information. Their proposal is restricted to TCP transmissions. Along the same lines yet more detailed is (Zhang et al. 2000), which proposes the use of a gateway. Again this work is only suited for Web-like traffic since only TCP is considered and they only share information between destinations with common 24-bit subnet masks (network locality).

In conclusion, while the above proposals are very interesting, they are specialized for TCP transmissions and may require substantial infrastructure changes due to the gateway approach. Furthermore, they employ a simple rule for sharing congestion control information, which, while it is empirically shown to be a good rule (Zhang et al. 2000), may be too restrictive for the case of a server-based inter-session congestion control for media streaming, which involves compared to a Web server a lesser number of sessions.

3 Reflective Media Server Design for Inter-Session Congestion Control

In this section, we give an overview of the high-level design of our reflective media server. The underlying principles of our reflective media server proposal is to gather past bandwidth availability data, process these data intelligently in order to make more *informed decisions* when starting a new TCP-friendly streaming session.

3.1 Overall Design

Two different, concurrently performed areas of operation can be distinguished for the reflective media server: the actual handling of media requests and the reflection on the corresponding transmission observations. The latter process of reflection is further on called *data management* because it involves the gathering and processing of statistical data for past sessions. The results from the data management operations are then exploited in serving the media requests, i.e., in the congestion-controlled *transmission* of the media objects. The following subtasks for the data management component can be identified:

- *data gathering,* i.e., record the data from sessions periodically for subsequent sessions,
- *data clustering*, i.e., explore the data on past and concurrent sessions for similarities in order to find maximum sharing rules between the recorded data,
- *data prediction*, i.e., forecast fair bandwidth share for a session based on the sharing rules constructed in the preceding step.

With respect to the transmission component: as discussed above we focus on the improvement of the start-up behaviour for media streams, i.e., we introduce an *informed start* which contrasts to slow start by assuming knowledge when choosing an initial transmission speed.

The overall scheme of our refective media server design is depicted in Figure 1.

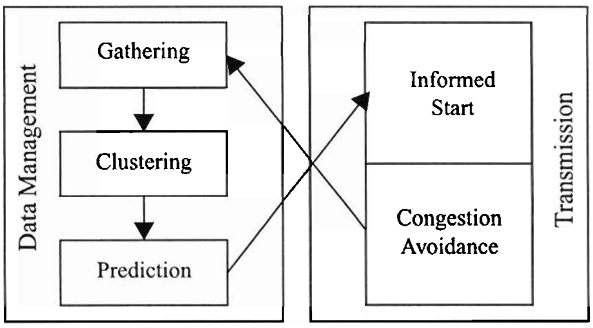

Figure 1: Reflective Media Server Design.

3.2 Data Management

Data Gathering: The major questions for data gathering are which data to gather and when. We decided to gather the available bandwidth values after they have reached a certain equilibrium state (i.e., the rate does not vary too much any more). The available bandwidth of each session (identified by the client's destination address) is tracked on two time-scales, one on the order of RTTs and one on the order of minutes. These serve different purposes. The longer time-scale values are used for identifying similar available bandwidth properties for different clients, i.e., they are input for the data clustering task. A time-scale on the order of minutes seems sufficient for that purpose based on the temporal stability observations reported in (Balakrishnan et al. 1997). The shorter time-scale values are used within the data prediction step and therefore need to be very recent.

Data Clustering: The data clustering subtask is a preparation step for the actual data prediction in order to make as much use as possible from the given data. In particular, we perform a cluster analysis along the available bandwidth samples of different sessions, which promises more comprehensive sharing rules than for a second-order criterion like network locality, since it allows to capture more similarities between clients / sessions, e.g., like the use of the same access technology which might always form a bottleneck or the situation when a transatlantic link is underdimensioned and a certain subset of clients is only reachable via this link. So, clients do neither need to share exactly the same bottleneck but only a structurally similar one, nor does the bottleneck need to be close to them. Furthermore, we cluster along available bandwidth trajectories, and not just single values, over relatively large time-scales (24 hours), which also allows to identify temporal similarities.

Since for individual clients the covering of a 24 hour interval by sampled available bandwidth values is likely to be insufficient, we first aggregate the samples of all clients of a network cloud defined by the common 24-bit subnet mask heuristic (network locality). The actual technique we then use for clustering the network clouds is so-called *agglomerative clustering* based on maximizing the inter-cluster distance while minimizing the intra-cluster distance where we use the euclidean distance norm for the bandwidth trajectories (Gordon 1999). The resulting clusters represent the sharing rules used for data prediction.

Data Prediction: The data prediction step takes as an input the short time-scale samples from the data gathering and uses the sharing rules resulting from the data clustering subtask to obtain a set of samples as large as possible to ensure an accurate prediction. The quantity to predict is the fair share of bandwidth available to a new media stream. Note that for different congestion control schemes this value may have to be transformed in the quantity that is relevant for the respective scheme, i.e., for a window-based scheme this has to be transformed into an initial window size (which would require to also sample the RTT, which for ease of discussion we have left out here since our focus is on rate-based schemes).

The actual prediction technique we use is an optimal linear predictor (Papoulis 1991), i.e., we make relatively little assumptions on the underlying stochastic process. This optimal linear predictor uses the existing realization of the stochastic process of the available bandwidth to set its linear coefficients such that the prediction error is minimized. This is only possible if the underlying process is ergodic, however, the results reported in (Balakrishnan et al. 1997) are encouraging with respect to this assumption. The number of linear coefficients that are employed depend on the number of samples that are available, the more samples are available the more linear coefficients are used resulting in a higher prediction accuracy. So, this is exactly the point where the maximization of the sharing rules is exploited.

3.3 TFRC Transmission Using *Informed Start*

Concurrently to the data management operations, the actual transmission of media streams takes place. As discussed above we chose TFRC as a (good) example of a TCP-friendly transmission protocol for media streams. Now, we describe how a TFRC-based media streaming can take advantage of the data gathered and evaluated by the data management component to improve a media stream's start-up behaviour by using what we call an *informed start* instead of the normal slow start algorithm. Therefore, we first discuss in a little bit more detail how TFRC works especially at start-up.

At the start-up of a session, TFRC mimics TCP's SS behaviour: it doubles its sending rate every RTT and even tries to emulate TCP's self-clocking characteristic by limiting the sending rate to two times the received bandwidth as reported by the receiver (which sends these reports every RTT). It does so until a loss event occurs. This enables then a receiver-based rough estimation of the loss rate as the corresponding loss rate for half of the current sending rate. This estimated loss rate is returned to the sender and used to compute the allowable sending rate using the TCP rate formula proposed in (Padhye et al. 1998). Furthermore, the sender then turns into a less aggressive CA-like behaviour which is again determined by the TCP rate formula: if the formula results in a higher value than the current sending rate then the sending rate is increased by one packet per RTT.

Assuming we have enough data to make a sensible fair share bandwidth prediction we can avoid the SS-like behaviour and start with the predicted bandwidth, i.e., perform an informed start (IS) and turn to the CA-like phase of TFRC directly. An IS requires, however, special care since the prediction might be wrong. In particular, IS works the following way:

- The transmission starts with the predicted rate. After 1.5 RTT the receiver calculates the corresponding loss rate from the inverse TCP rate formula and sends it towards the sender. It cannot invoke the TCP rate formula before because it requires an estimate of the RTT which is only determined after 1 RTT at the sender and then sent to the receiver (which takes another 0.5 RTT).
- Before the sender receives the first loss rate estimate the sender uses the minimum of predicted rate and received rate as reported by the receiver. This restriction minimizes the negative effect of a wrong prediction for the available fair share of the bandwidth.
- After it got the first loss rate estimation (after 2 RTT), the sender uses TFRC's normal CA-like behaviour further on.
- In case of packet loss two cases must be distinguished:
  (1) packet loss *before 2 RTT*: this indicates that the predicted available bandwidth was too optimistic and the sender should backoff immediately in order not to interfere with other TCP sessions. Of course, due to the packet loss we have a first estimate of the loss rate, however it is very likely to be too pessimistic since due to the overestimation of the allowed sending rate losses are probably excessive. Using that loss rate would consequently lead to an underestimation of the actual allowed sending rate. Fortunately, we also have the received rate as reported by the receiver as a further guide. While the received rate itself is obviously too high because we have been overly aggressive at the start-up, we can take a compromise between underestimating and overestimating the allowed sending rate by taking the mean of the fair bandwidth share as computed by the TCP rate formula and the received bandwidth. When the fair rate eventually becomes higher than the received rate we turn to CA-like TFRC behaviour.
  (2) packet loss *after 2 RTT*: here we just use normal TFRC behaviour, i.e., the loss rate is reported to the sender and the sender adapts its current sending rate.

A further question that comes up after this discussion is what happens if we underestimate the currently available bandwidth. Here, the problem is that since we do not use a SS-like trialling of the available bandwidth at the start of a new session we may remain in a state of underutilizing the fair share for that media stream. However, at least no other sessions suffer and probably for the case of media streaming we should actually reject the request for a new stream if the predicted available bandwidth is too low since we cannot expect our estimate to be too low and it is better not to start a session which can anyway not deliver the quality a user would expect.

4 Simulations

The aim of the following simulation experiments with the ns-2 simulator[2] is to show the basic improvements that can be achieved with an IS over the normal SS-like behaviour of TFRC. They are not about the analysis of the data management component of our reflective media server design, but make extreme assumptions on the outcome of the data management operations: the fair share bandwidth predictions are assumed to be either correct, far too high, or far too low. The simple simulation setup we used for these experiments is shown in Figure 2.

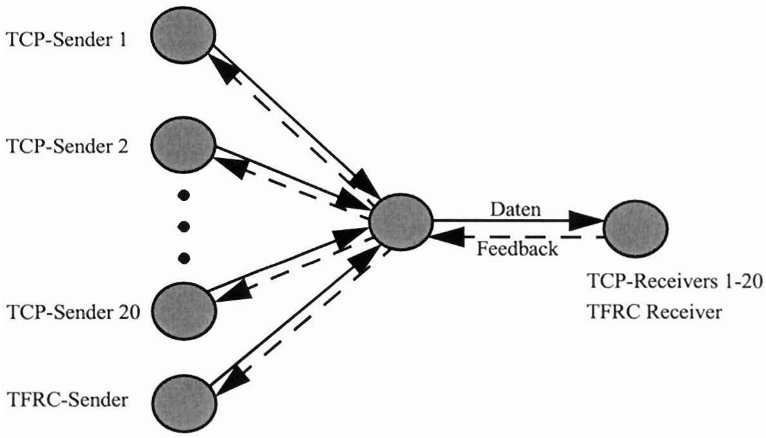

Figure 2: Simulation Setup.

The queue at the bottleneck link uses drop tail, all links are dimensioned at 10 Mb/s (or 1.25 MB/s) with a propagation delay of 10 ms. The TCP senders use TCP Reno (i.e., they employ fast retransmit) and all of them are all started at the beginning of the simulation runs (t=0s). At approximately t=4s they achieved an equilibrium state where they shared the available bandwidth at the bottleneck link fairly. Thus at t=4s we started our different versions of TFRC:

- TFRC with usual SS,
- TFRC with IS and correct prediction (CORR), i.e., the fair share bandwidth prediction is 1/21 of 1.25 MB/s (≈ 60 KB/s)
- TFRC with IS and far too high prediction (HIGH), in particular, the fair share bandwidth prediction is 3 times to high (≈ 180 KB/s)
- TFRC with IS and far too low prediction (LOW), in particular, the fair share bandwidth prediction is 3 times to low (≈ 20 KB/s)

In Figure 3, the simulation outcomes for the different scenarios are given. Here, we have depicted the sending behaviour of one of the TCP senders (TCP-Sender 1[3]) vs. the respective TFRC sending behaviour in the relevant time-scale (from 3s to 15s).

2. http://www.isi.edu/nsmam/ns
3. The others showed the same behaviour, though with some phase shifts.

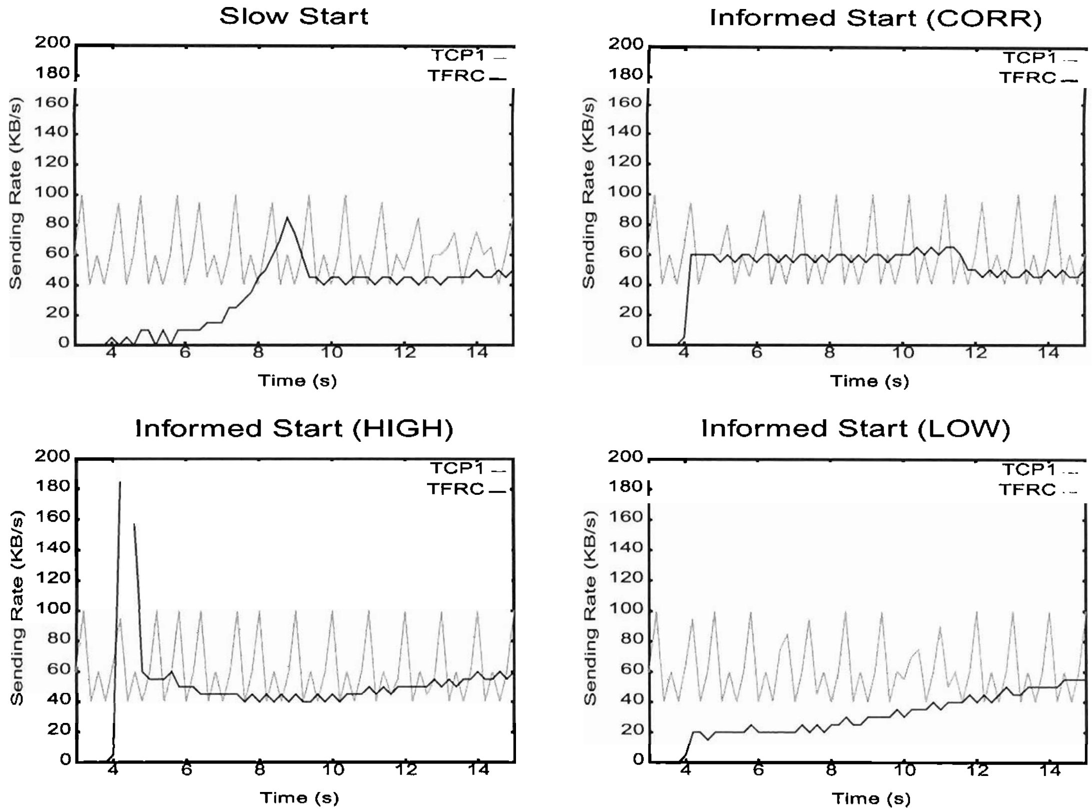

Figure 3: Slow Start vs. Informed Start TFRC with Differing Prediction Scenarios.

It is obvious that with a correct prediction we can substantially improve on TFRC's usual start-up behaviour resulting from SS: TFRC with SS took about 5s (from t = 4s to 9s) until it turns to a stable CA-like behaviour, whereas TFRC with IS(CORR) shows immediate stability from its start. Interestingly, also for a far too high prediction of the fair bandwidth share for the TFRC session, it takes only about 1s until a stable behaviour can be observed. So, we have achieved the goal of a fast reaction of the informed start on an overestimated bandwidth prediction. The case IS(LOW) shows that an underestimation requires a longer start-up phase until the fair bandwidth share is reached than the other cases (including the slow start case), yet it does so in a fairly smooth way which from the perspective of streaming applications should be desirable.

5 Conclusions

In this paper, we have investigated how TCP-friendly transmission schemes for media streaming could be enhanced to circumvent the inheritance of TCP's disadvantageous start-up behaviour by the use of inter-session congestion control. For that purpose we have introduced a reflective media server design and described its major functional components: data management and transmission. In contrast to previous work, we have focussed on the maximization of sharing rules between sessions by the use of cluster analysis techniques taking into account the specific requirements for media streaming servers. We have shown how TFRC, a special instance of a TCP-friendly transmission protocol can be extended to use an informed start based on the operations performed by the data management component of the reflective media server. By simulations we have shown the benefits of an informed start over the normal slow start-like behaviour of TFRC.

References

Balakrishnan, H., Stemm, M., Seshan, S., and Katz, R. H. (1997). Analyzing Stability in Wide-Area Network Performance. In *Proc. of the ACM SIGMETRICS, Seattle, WA*, pages 2–12.

Bansal, D. and Balakrishnan, H. (2001). TCP-friendly Congestion Control for Real-time Streaming Applications. In *Proceedings of the 20th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM'01)*. IEEE Computer Society Press, Anchorage, AK, USA.

Floyd, S., Handley, M., Padhye, J., and Widmer, J. (2000). Equation-Based Congestion Control for Unicast Applications. In *Proceedings of the ACM SIGCOMM '00 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication 2000, Stockholm, Sweden*, pages 43–56.

Gordon, A. (1999). *Classification*. Chapman-Hall.

Hua, K. A., Cai, Y., and Sheu, S. (1998). Patching: A Multicast Technique for True Video-on-Demand Services. In *Proceedings of the ACM Multimedia Conference 1998, Bristol, England*, pages 191–200.

Jin, S., Guo, L., Matta, I., and Bestavros, A. (2001). TCP-friendly SIMD Congestion Control and Its Convergence Behavior. In *Proceedings of ICNP'2001: The 9th IEEE International Conference on Network Protocols, Riverside, CA*.

Padhye, J., Firoiu, V., Towsley, D., and Kurose, J. (1998). Modeling TCP Throughput: A Simple Model and its Empirical Validation. In *ACM SIGCOMM '98 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, pages 303–314, Vancouver, CA.

Papoulis, A. (1991). *Probability, Random Variables, and Stochastic Processes*. McGraw-Hill.

Rejaie, R., Handley, M., and Estrin, D. (1999). RAP: An End-to-End Rate-based Congestion Control Mechanism for Realtime Streams in the Internet. In *Proceedings of the Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies 1999, New York, NY, USA*, pages 395–399.

Rhee, I., Ozdemir, V., and Yi, Y. (2000). TEAR: TCP emulation at receivers - flow control for multimedia streaming. Technical report, North Carolina State University.

Savage, S., Cardwell, N., and Anderson, T. (1999). The Case for Informed Transport Protocols. In *Proceedings of the Seventh Workshop on Hot Topics in Operating Systems, Rio Rico, AZ.*

Widmer, J., Denda, R., and Mauve, M. (2001). A Survey on TCP-Friendly Congestion Control. *Special Issue of the IEEE Network Magazine "Control of Best Effort Traffic"*, 15(3):28–37.

Widmer, J. and Handley, M. (2001). Extending Equation-Based Congestion Control to Multicast Applications . In *Proceedings of the ACM SIGCOMM '01 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication 2001, San Diego, CA*, pages 275–285.

Zhang, Y., Qiu, L., and Keshav, S. (2000). Speeding Up Short Data Transfers: Theory, Architecture Support, and Simulation Results. In *Proceedings of NOSSDAV 2000.*