Report Number: WUCSE-2004-50

2004-05-15

Selecting the Buffer Size for an IP Network Link

Sergey Gorinsky, Anshul Kantawala, and Jonathan S. Turner

In this paper, we revisit the problem of selecting the buffer size for an IP network link. After a comprehensive overview of issues relevant to the link buffer sizing, we examine usefulness of existing guidelines for choosing the buffer size. Our analysis shows that the existing recommendations not only are difficult to implement in the context of IP networks but also can severely hurt interactive distributed applications. Then, we argue that the networking research community should change its way of thinking about the link buffer sizing problem: the focus should shift from optimizing performance for applications of a particular... **Read complete abstract on page 2.**

Selecting the Buffer Size for an IP Network Link

Sergey Gorinsky, Anshul Kantawala, and Jonathan S. Turner

Complete Abstract:

In this paper, we revisit the problem of selecting the buffer size for an IP network link. After a comprehensive overview of issues relevant to the link buffer sizing, we examine usefulness of existing guidelines for choosing the buffer size. Our analysis shows that the existing recommendations not only are difficult to implement in the context of IP networks but also can severely hurt interactive distributed applications. Then, we argue that the networking research community should change its way of thinking about the link buffer sizing problem: the focus should shift from optimizing performance for applications of a particular type to maximizing diversity of application types that IP networks can support effectively. To achieve this new objective, we propose using small buffers for IP network links.

Department of Computer Science & Engineering

Washington University in St. Louis
School of Engineering & Applied Science

2004-50

Selecting the Buffer Size for an IP Network Link

Authors: Gorinsky, Sergey; Kantawala, Anshul; Turner, Jonathan;

May 15, 2004

Abstract: In this paper, we revisit the problem of selecting the buffer size for an IP network link. After a comprehensive overview of issues relevant to the link buffer sizing, we examine usefulness of existing guidelines for choosing the buffer size. Our analysis shows that the existing recommendations not only are difficult to implement in the context of IP networks but also can severely hurt interactive distributed applications. Then, we argue that the networking research community should change its way of thinking about the link buffer sizing problem: the focus should shift from optimizing performance for applications of a particular type to maximizing diversity of application types that IP networks can support effectively. To achieve this new objective, we propose using small buffers for IP network links.

Selecting the Buffer Size for an IP Network Link

Sergey Gorinsky
gorinsky@arl.wustl.edu
Anshul Kantawala
anshul@arl.wustl.edu
Jonathan Turner
jst@arl.wustl.edu

WUCSE-2004-50

May 15, 2004

Department of Computer Science and Engineering
Campus Box 1045
Washington University
One Brookings Drive
St. Louis, MO 63130-4899

Abstract

In this paper, we revisit the problem of selecting the buffer size for an IP network link. After a comprehensive overview of issues relevant to the link buffer sizing, we examine usefulness of existing guidelines for choosing the buffer size. Our analysis shows that the existing recommendations not only are difficult to implement in the context of IP networks but also can severely hurt interactive distributed applications. Then, we argue that the networking research community should change its way of thinking about the link buffer sizing problem: the focus should shift from optimizing performance for applications of a particular type to maximizing diversity of application types that IP networks can support effectively. To achieve this new objective, we propose using small buffers for IP network links.

Selecting the Buffer Size for an IP Network Link

Sergey Gorinsky and Anshul Kantawala and Jonathan Turner
Applied Research Laboratory
Department of Computer Science and Engineering
Washington University in St. Louis
St. Louis, MO 63130-4899, USA
{*gorinsky,anshul,jst*}*@arl.wustl.edu*

Abstract

In this paper, we revisit the problem of selecting the buffer size for an IP network link. After a comprehensive overview of issues relevant to the link buffer sizing, we examine usefulness of existing guidelines for choosing the buffer size. Our analysis shows that the existing recommendations not only are difficult to implement in the context of IP networks but also can severely hurt interactive distributed applications. Then, we argue that the networking research community should change its way of thinking about the link buffer sizing problem: the focus should shift from optimizing performance for applications of a particular type to maximizing diversity of application types that IP networks can support effectively. To achieve this new objective, we propose using small buffers for IP network links.

1. Introduction

The exponential growth of the Internet can be attributed to the simplicity of the Internet Protocol (IP) [23] which defines an addressing structure for *end systems* as well as format for *datagrams* used as packages for delivering data to addressed entities, potentially through intermediary *routers* interconnected by *links*. The only requirement that IP imposes on routers is an ability to forward a datagram to an output link according to the address field in the header of the datagram. To map IP addresses to output links, routers build forwarding tables by exchanging information via routing protocols. Except for routing, IP networks do not require any additional signaling between routers. The minimality of the coordination requirements empowers the quick deployment of IP networks. The owner of an IP network can build and operate the network with a great independence from the other networks in the Internet. In particular, the IP network owner can buy routers of various designs and from different manufacturers, install links with diverse bitrates, choose distinct methods for link access, and even compose a link from different transmission media connected by lower-layer devices such as hubs and bridges. On the other hand, the lack of coordination creates possibilities for mismatches between the traffic load on an IP router and the router ability to forward datagrams.

If the router receives a burst of datagrams destined for the same output link, some datagrams have to be discarded unless the router can store them in a buffer until the link becomes available.

In this paper, we examine the question of how much buffer an independent owner of an IP router needs to allocate for an output link to improve performance of distributed applications that run on end systems and communicate by sending datagrams through the link. The problem of the link buffer sizing can also be posed for other network architectures and can have very different answers than for IP networks. For example, one can imagine a network where all devices and transmission media operate according to a global clock and adhere to a global delivery schedule that voids a need for buffering in intermediate devices. Also, one can see a value in asynchronous networks with extensive signaling (such as hop-by-hop protocols for congestion control and reliability) that allows a distributed application to reserve a portion of the link buffer (and a share of the link bitrate) sufficient to achieve the desired end-to-end performance. By writing this paper, we do not argue against – or for – such other types of network architectures; discussions on their technical feasibility, practical usefulness, or economic viability lie beyond the scope of this work. We focus exclusively on IP networks because they are the mainstream reality of modern computer networking.

The dominance of IP networks is also a reason for our approach to addressing the buffer sizing problem. We strive for an optimal solution that the owner of an IP router can implement in practice without abandoning the independent spirit of IP networking; in particular, the router owner should be able to select the link buffer size without any additional signaling from other routers or end systems. Earlier attempts to solve the link buffering problem do not satisfy this requirement. For example, proposed guidelines for computing the optimal buffer size often require knowledge of the round-trip propagation delays for the served applications, information that is not available at the IP router. Furthermore, even if the router owner is somehow able to determine a value approximating a recommended setting for the buffer size, choosing this value usually improves performance only for a class of applications such as long file transfers over a standard end-to-end protocol for congestion control and reliability. The recommended setting, however, can have a devastating effect on other types of applications. For instance, extra queueing due to a large recommended buffer size can defeat interactive applications that require short end-to-end datagram delays.

Our proposal is to change the way the networking research community approaches the problem of selecting the buffer size for an IP network link. We believe that the focus should shift from optimizing performance for applications of a particular type to maximizing variety of application types that IP networks can support effectively. With this new goal in mind, we argue that IP network links should use small buffers in order to open the Internet for wide deployment of innovative delay-sensitive applications.

The rest of the paper is structured as follows. After Section 2 discusses briefly why buffering of datagrams in routers can be beneficial, Section 3 provides a comprehensive overview of issues related to selecting the link buffer size. Section 4 examines existing guidelines for setting the buffer size. Then, Section 5 argues that the existing guidelines are not only difficult to implement in practice but also detrimental to performance of applications that require short end-to-end datagram delays. Section 6 offers an alternative solution to keep the link buffer size small in order to expand the set of applications that IP networks can support effectively. Finally, Section 7 summarizes our findings.

2. Rationale for Buffering in Routers

Even when all output links of a router are lightly utilized on average, its input links can simultaneously deliver datagrams destined for the same output link. Since the router can forward only one datagram to the output link at a time, the router operator faces a dilemma of what to do with the extra datagrams. One simple option is to discard them and leave dealing with the loss to end systems. However, the discards waste the network capacity that the datagrams have consumed so far. Besides, since the end systems of distributed applications that require reliable delivery need to overcome the losses by transmitting additional datagrams, the discards can prolong the delivery. Another alternative is to store the extra datagrams in the router until the output link becomes available. If the allocated buffer is large enough, the router forwards the burst without losing any of the datagrams. Hence, buffering can help the router to avoid or reduce datagram losses during a transient overload.

3. Factors in Selecting the Buffer Size

In this section, we discuss factors that are relevant to the problem of determining an optimal buffer size for an IP network link. Whereas none of our observations is likely to be particularly new for an expert in computer networking, this thorough survey of the numerous factors is useful because it shows the complexity of the examined problem.

3.1. Network Load and Topology

The network load and topology are obviously the two major factors in the buffer sizing problem. Below, we consider these factors in more detail in the context of IP networks.

If end systems send a datagram only occasionally, a contention of datagrams for network links is unlikely, and link buffers are lightly utilized. As the amount of data generated by applications grows, the chances of such contention grow as well and create a need to address associated queueing and loss of datagrams. In IP networks, routing of datagrams is load-insensitive and based on relatively static metrics such as the minimal number of network links between communicating end systems. Dealing with excessive load is relegated to end systems that are supposed to regulate the amount of transmitted datagrams by adhering to congestion control algorithms. End systems in IP networks offer two options for transporting data of distributed applications:

- **Transmission Control Protocol** (TCP) [25] is a service for reliable in-order delivery of data and incorporates congestion control mechanisms. Popular versions of TCP congestion control include Tahoe [17], Reno [18], NewReno [11], and SACK [20] where the sending end increases its transmission until a datagram loss indicates that the capacity of the traversed route is exhausted. With these TCP versions, increasing the buffer size usually results in longer queues at the bottleneck link and higher end-to-end datagram delays. TCP Vegas [5] uses delay variations as an implicit signal about the route congestion status and, under certain conditions, is capable of controlling the transmission without overflooding the buffer of the

bottleneck link. A similar effect can be achieved by integrating TCP with Explicit Congestion Notification (ECN) [27] that allows the router of a congested link to notify end systems about the congestion by setting a bit in the headers of forwarded datagrams.

- **User Datagram Protocol** (UDP) [24] is a plain service that ships data of applications as datagrams into the network but offers neither congestion control nor reliability. Unlike with the TCP service which is typically provided by standard software or hardware of end systems, applications that communicate their data over UDP need to implement congestion control algorithms on their own. On the other hand, using UDP enables the applications to avoid undesirable effects (such as extra delays of retransmission-based reliability) caused by the TCP service.

Regardless of whether TCP or UDP transports data of an application over the network, end-to-end congestion control is usually based on perpetual probing for the available capacity on the traversed route: the sending end keeps increasing the load until the capacity is exhausted; then, the sender decreases the load and starts the next increase-decrease cycle. Hence, even when the set of applications and the amount of data generated by the applications do not change, end-to-end congestion control keeps the network load oscillating. The size of the load oscillations caused by end-to-end congestion control affects loss, delay, and throughput characteristics of the network performance and therefore is important for the problem of selecting the buffer size for the bottleneck link. A related issue is the dependence of the oscillations on the number of applications that use the bottleneck link concurrently. If a larger number of applications increases the load oscillations (as it happens when the applications rely on TCP congestion control [7, 15]), the number of concurrent applications becomes an important factor in the buffer sizing problem.

The network topology affects the buffer sizing problem by influencing which route the network picks for forwarding datagrams of an application. Together with the network load, the network topology determines how many bottleneck links the selected route traverses and thus how many times a datagram experiences queueing before reaching the receiving end of the application.

3.2. Buffer Management Policies

Whereas the size of the link buffer matters, it is also important how the router uses the buffer. Buffer management policies can be classified into two categories:

- **Link scheduling policy** specifies when and at which order the router forwards queued datagrams to the link. Routers in IP networks usually employ work-conserving scheduling disciplines that keep the link idle only if the link buffer contains no datagrams. With respect to the forwarding order, First-In First-Out (FIFO) is the simplest and the most common policy where the router forwards datagrams in the order of their arrival to the buffer. When an application deposits a burst of datagrams into a FIFO buffer, subsequently arriving datagrams from other applications might experience long queueing delay associated with forwarding of the burst, even if these applications transmit their datagrams only occasionally and at a smooth rate.

  To serve diverse applications better, numerous alternatives to FIFO have been proposed. Implementing an application-aware link scheduling policy in an IP router is a challenge because

IP networks do not provide the router with means for reliable mapping of a received datagram to the application that uses the datagram. A common approach is to group received datagrams into *flows* according to IP addresses, port numbers, and other fields in the datagram headers. Then, the router treats the datagrams of a flow as datagrams belonging to an application (or a class of applications). Note, though, that the grouping is not always precise; an application can – either inadvertently or intentionally – make routers assign its datagrams to multiple flows by placing different values in the same header field (e.g., the source port number) of different datagrams. With datagrams classified into flows, a router has a variety of options for flow-aware link scheduling. For example, the router can assign static priorities to flows and schedule datagrams according to these priorities. Alternatively, the router can adopt a fair queueing discipline that strives for fair sharing of the link capacity among all the flows that have datagrams backlogged at the link buffer [8, 16].

- **Datagram discard policy** determines which datagrams the router discards and when the discards take place. Droptail is a common discard policy where datagrams are dropped only upon arrival to the already full buffer. In Random Early Discard (RED) [13] and other active queue management schemes [9, 19], the router can discard a datagram even when the buffer has enough free space. Similarly to link scheduling, datagram discarding can exploit classification of datagrams into flows. For example, the router can choose to drop datagrams from a flow that contributes the most to the buffer occupancy [30].

Since FIFO Droptail routers are typical in contemporary IP networks, we focus on FIFO Droptail buffering in our subsequent analysis. However, one should keep in mind that the problem of the link buffer sizing can have different answers for different buffer management policies.

3.3. Local Impact of Buffering

The buffer size affects end-to-end performance of distributed applications via its local impact on datagrams belonging to the applications. The following three parameters capture average local performance of the router:

- **Link loss rate** for an output link is the fraction of discarded datagrams among the received by the router for forwarding to the link.
- **Link queueing delay** for datagrams forwarded to an output link is the average amount of time between the moments when the router receives a datagram and forwards it to the output link.
- **Link utilization** for an output link is the ratio of time when the router forwards datagrams to the output link.

In an ideal scenario, the router should forward datagrams to the output link without delay or loss. Besides, if the link is a bottleneck for delivering data of an application, the router should keep the link fully utilized. It is often impossible to attain all of the above goals. Although changing the buffer size can optimize the router performance with respect to one of the three parameters, the improvement usually comes at the expense of degrading another parameter.

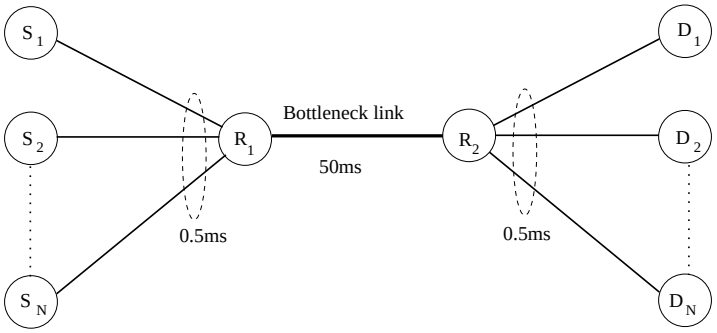


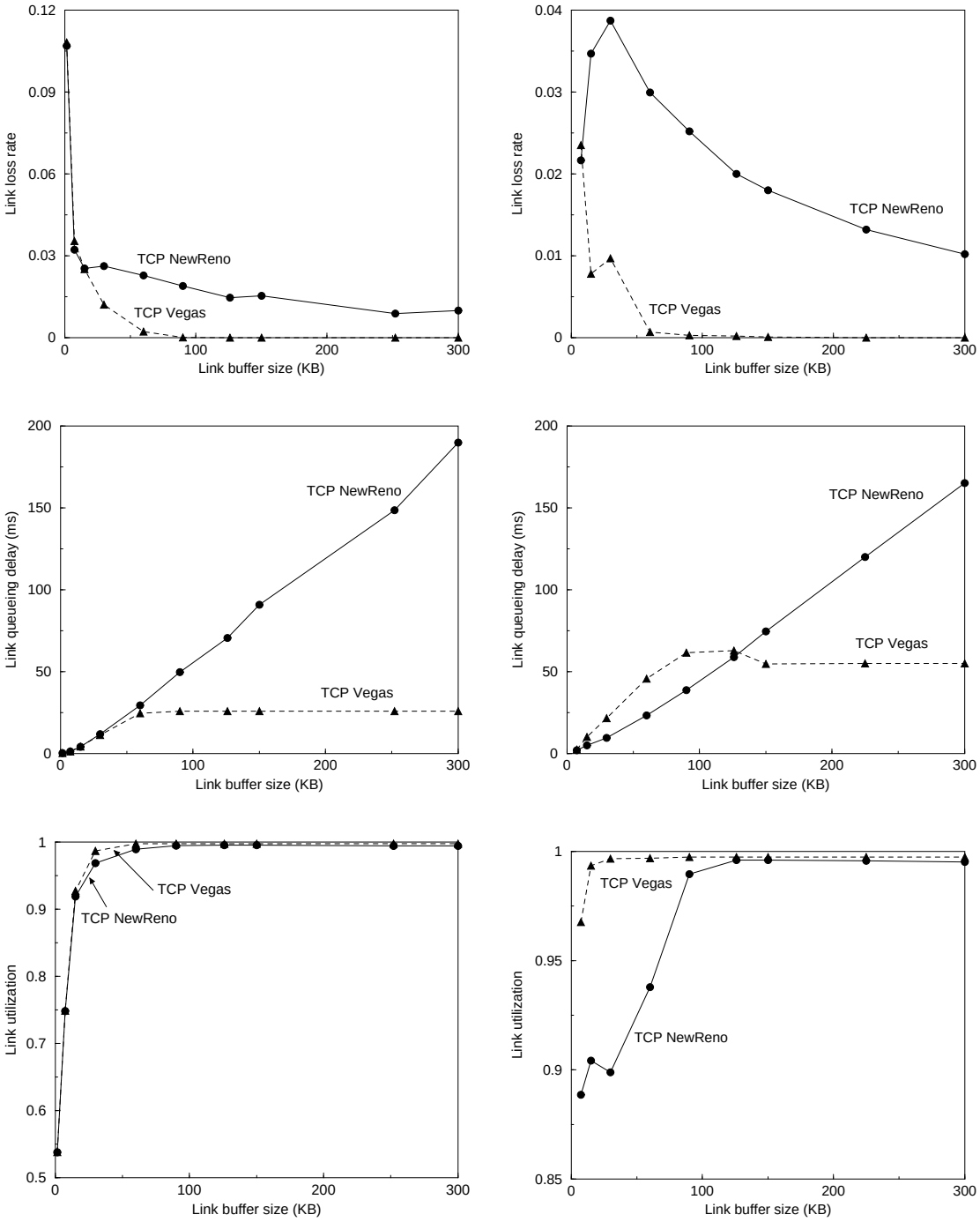
Figure 1: The single-bottleneck topology.

To explore the trade-off, we conduct NS-2 [22] simulations for the topology depicted in Figure 1. End systems S_i host the sending ends of distributed applications, and end systems D_i are the receiving ends for these applications (i varies from 1 to N). The link from router R_1 to router R_2 is a bottleneck for each of the applications. The bottleneck link has a propagation delay of 50 ms. Each of the other links has a propagation delay of 0.5 ms and a bitrate that is twice larger than the bitrate of the bottleneck link. All the applications use datagrams of size 1500 bytes. The applications that rely on TCP employ either NewReno or Vegas versions of it. Each of the links uses FIFO Droptail buffering. We measure the loss rate, queueing delay, and utilization for the 10 Mbps bottleneck link over the whole experiment duration of 200 seconds. Figure 2a reports the results for a scenario where the mix of applications consists of four long file transfers over TCP, a constant-bit-rate (CBR) 2 Mbps stream over UDP, and short web downloads (modeled as four Poisson processes generating datagrams with a mean rate of 1 Mbps each). As the buffer size increases, the link utilization improves until approaching 100% around the buffer size of 100 KB. With respect to the other two parameters, selecting the buffer size is subject to a trade-off between a smaller loss rate and a smaller queueing delay. Figure 2b plots the results for another scenario where all of the twenty applications are long file transfers. Once again, choosing the buffer size is constrained by a trade-off between the link loss rate, link queueing delay, and link utilization.

When a link carries datagrams from multiple applications, cumulative parameters of the link performance do not represent the treatment given by the router to datagrams of a particular application. For each characteristic of the overall local performance, one can define an application-specific equivalent:

- **Local loss rate** for an application at a link is the fraction of discarded datagrams among the received by the router from the application for forwarding to the link.
- **Local queueing delay** for forwarded datagrams of an application is the average amount of time between the moments when the router receives a datagram from the application and forwards the datagram to the output link.
- **Local throughput** of an application through a link is the rate at which the router forwards data from the application to the link.

Any of the above local characteristics can vary from one application to another. This phenomenon is not limited to advanced router architectures that isolate flows or provide some other explicit means



(a) Four long file transfers over TCP, one CBR stream over UDP, and short web downloads

(b) Twenty long file transfers over TCP

Figure 2: Trade-offs in the bottleneck link performance.

for handling different applications differently. Even in networks where routers employ simple FIFO Droptail buffering, applications of the same type can have very diverse experiences at a shared bottleneck link. For example, concurrent file transfers over TCP can experience substantially distinct loss rates at a shared bottleneck link due to stable long-term phase effects resulting from differences in propagation delays [12, 14]. Furthermore, even if the router enforces similar loss rates (e.g., by relying on FIFO RED buffering), file transfers with larger propagation delays achieve usually smaller throughputs because TCP uses the amount of outstanding data – rather than the transmission rate – as the congestion control parameter. Hence, even local performance of an application at a router depends not only on the router buffer size but also on global factors such as the network topology, amount of data generated by applications, and end-to-end congestion control.

3.4. Relationship between Local and End-to-End Performance

In Section 3.3, we showed that selecting the buffer size to improve local performance of a router is subject to a trade-off between the link loss rate, link queueing delay, and link utilization. We also discussed difficulties with translating the link performance into local experiences of datagrams belonging to a particular application. However, if the local performance of an application is already characterized for each link between its sending and receiving ends, one can easily express end-to-end treatment of datagrams belonging to the application by accounting for all the experienced queueing and losses:

- **End-to-end loss rate** for an application is the fraction of discarded datagrams among the transmitted from the sending end of the application.
- **End-to-end delay** for an application is the average amount of time to deliver a datagram from the sending end to the receiving end of the application.
- **End-to-end throughput** of an application is the rate at which the network delivers data from the sending end to the receiving end of the application.

Similarly to the performance of a link, the end-to-end performance of an application is constrained by a trade-off between the loss rate, delay, and throughput. However, a network-layer assessment of datagram experiences is insufficient for offering the application a right balance between the parameters. One also has to consider how the application uses the datagrams and which of the parameters are important for the application.

As an example, let us first consider a file transfer application. The main objective in this application is to deliver a complete file to its destination. Hence, it is reasonable to build the application on top of TCP which offers a retransmission-based service of delivering data reliably despite potential datagram losses in the network. However, it is also possible to implement file transfer applications over UDP, as in the Digital Fountain approach where forward-error correction techniques ensure reliable delivery [6]. Regardless of the implementation, what matters for the application is the overall time to deliver the complete file to the destination. Values for the achieved end-to-end datagram delay or loss rate are not significant by themselves. For a large file, the delivery time can be substantially longer than queueing delays experienced by datagrams at bottleneck links, even at those with big Droptail buffers. Also, discarding a datagram might have no impact on the file transfer

when the datagram carries redundant data; such instances include datagrams with duplicate TCP data segments or error correction codes for already delivered data. Furthermore, datagram losses are not a detrimental for the file delivery time if the application is able to communicate the lost data without causing an underutilization of the bottleneck link. Under these circumstances, the achieved end-to-end throughput becomes the dominant factor for the file transfer application.

Interactive audio streaming is an application with a very different set of performance priorities. In contrast to transferring a large file, audio streaming does not require a high throughput because the ends of the application generate data at low rates. On the other hand, it is essential to deliver the generated data within a time interval on the order of few hundred milliseconds – larger delays make a conversation uncomfortable for human perception. Whereas interactive audio can tolerate some datagram losses, resilience to high loss rates might require an incorporation of reliability mechanisms such as forward error correction. Since the retransmission-based reliability of TCP can introduce excessive delays, it is common to implement audio streaming on top of UDP. In general, if the objective is to improve performance of interactive audio streaming, the end-to-end delay is the crucial factor for deciding how much buffer the network links should have.

Our conclusion from the above discussion is that the problem of optimizing the link buffer size has different solutions for different applications.

4. Existing Guidelines for Setting the Buffer Size

As Section 3 showed, finding an optimal buffer size for an IP network link is a complicated problem affected by numerous parameters including the network topology, amount of data generated by applications, protocols employed for end-to-end congestion control and reliability, buffer management policies, and types of applications. Consequently, it is reasonable that existing approaches to this problem make simplifying assumptions about the impacting factors and propose approximate guidelines for the link buffer sizing.

Below, we consider three popular guidelines derived from scenarios where a single-bottleneck topology with FIFO Droptail buffering serves only applications that transfer files over a loss-driven version of TCP (such as Tahoe or NewReno):

- **Bitrate-delay product** [31] is a recommendation stemming from the following simple analysis for a situation where the only application is a long file transfer over a TCP connection. The analysis assumes that the connection stabilizes at the congestion-avoidance mode where the amount of unacknowledged outstanding data (controlled by the congestion window at the sender) oscillates between B and $2B$. To keep the bottleneck link fully utilized, B should be at least a bitrate-delay product defined as the link bitrate multiplied by the round-trip propagation delay of the TCP connection. Since increasing B beyond the bitrate-delay product increases the link queueing delay without any further improvement in the bottleneck link utilization, it is recommended to keep B equal to the bitrate-delay product. The buffer size for the bottleneck link in this setting also equals B. Hence, the guideline suggests to set the link buffer size to the bitrate-delay product.

- **Less than the bitrate-delay product** [3, 29] is a guideline that amends the above recommendation for a scenario where the number of the long file transfers is not one but many. The following argument justifies the amendment. When an overflow of the bottleneck link buffer causes a discard of datagrams, only a subset of TCP connections loose data and reduce their congestion windows in half to resolve the buffer congestion. The other TCP connections do not decrease their congestion windows. Then, the cumulative amount of unacknowledged outstanding data oscillates not between B and $2B$ but between B and X where $B < X < 2B$. Consequently, it is recommended to set the link buffer size to a smaller value than the bitrate-delay product.

- **Per-connection allocation** [21] is a guideline based on the observation that if the average congestion window of a TCP connection is less than few TCP data segments, the connection experiences high loss rate and suffers from frequent retransmission timeouts that knock the connection out from the congestion-avoidance mode. Considering such a timeout-ridden pattern of operation inappropriate, the guideline suggests that the bottleneck link buffer should provide enough space for at least few datagrams from each TCP connection. Hence, it is recommended to set the buffer size proportional to the number of TCP connections.

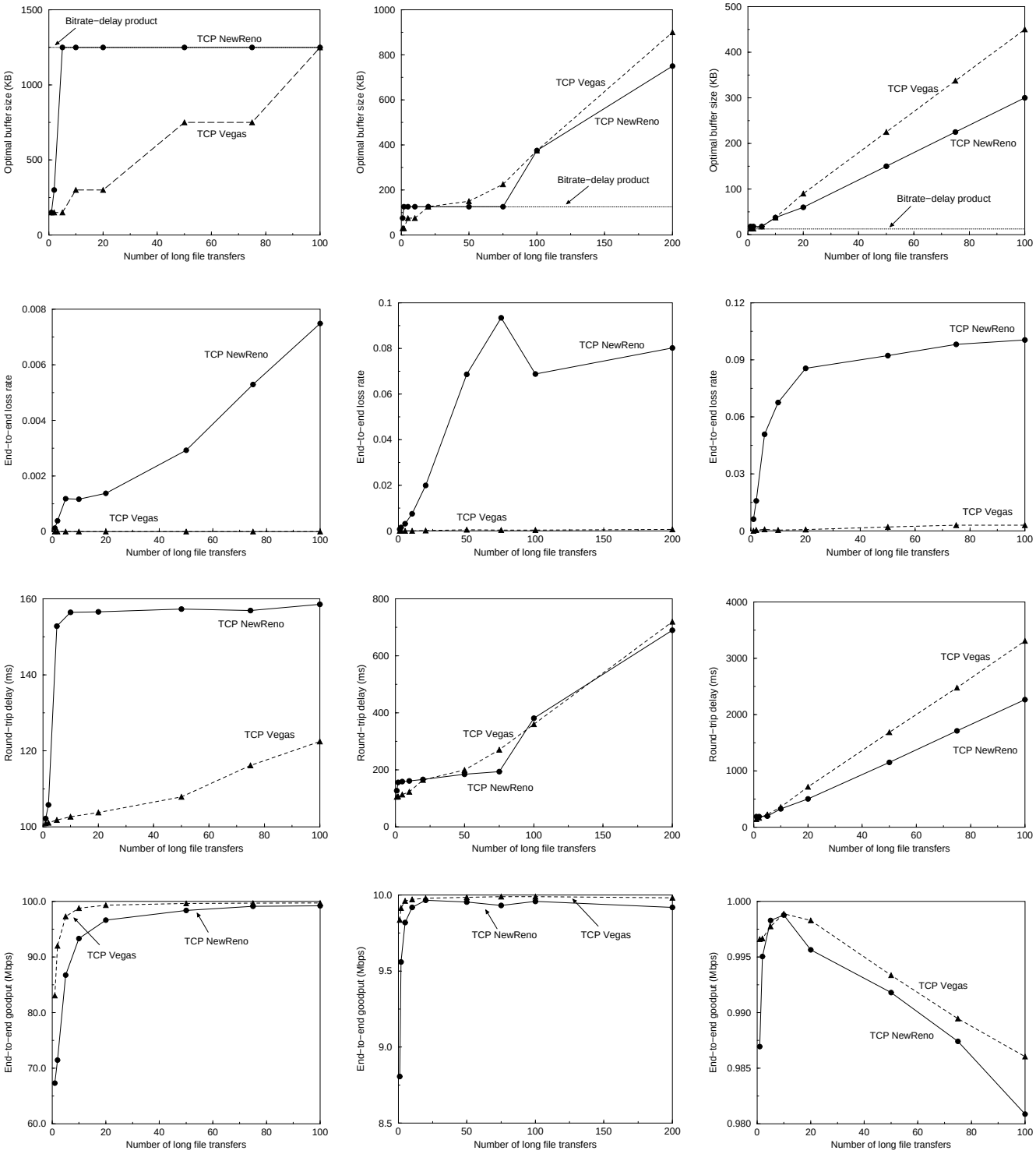
The above list of existing guidelines is interesting because they contradict each other. For example, whereas the recommendation to set the buffer size to the bitrate-delay product does not consider the number of connections, the guideline of per-connection allocation does not mention the link bitrate or network propagation delays. To check correctness of the guidelines, we conduct experiments reusing the single-bottleneck topology from Figure 1. All of the N applications in these experiments are long file transfers and rely on either NewReno or Vegas versions of TCP. The three examined values for the bottleneck link bitrate are 100 Mbps, 10 Mbps, and 1 Mbps.

To discount data that might be delivered redundantly (due to retransmission-based mechanisms employed by TCP), we report not the end-to-end throughput discussed in Section 3.4 but an end-to-end goodput defined as the ratio of the file size to the file delivery time. Then, since the main performance objective for a file transfer is to deliver the file as soon as possible, we define an optimal buffer size as the buffer size that provides the file transfers with the largest goodput. Figure 3 reports the optimal buffer size for the bottleneck link as well as the end-to-end loss rate, round-trip delay, and end-to-end goodput in the scenarios where the buffer size of the bottleneck link is set to this optimal value.

For TCP NewReno, the three top graphs in Figure 3 show that validity of the existing guidelines depends on the ratio of the bitrate-delay product to the number of file transfers. When the ratio is high (see the graph for the 100 Mbps bottleneck link), the bitrate-delay product approximates the optimal buffer size quite precisely. However, if the ratio is low (see the graph for the 1 Mbps bottleneck link), the optimal buffer size is proportional to the number of file transfers in accordance with the recommendation of per-connection allocation.

For TCP Vegas, the guideline of setting the link buffer size to the bitrate-delay product does not appear to be useful. In all of the examined scenarios, the optimal buffer is approximately proportional to the number of file transfers.

In the settings where the ratio of the bitrate-delay product to the number of file transfers is low, following the guideline of per-connection allocation limits the end-to-end loss rate and maximizes



(a) 100 Mbps bottleneck link (b) 10 Mbps bottleneck link (c) 1 Mbps bottleneck link

Figure 3: End-to-end performance of long file transfers over TCP.

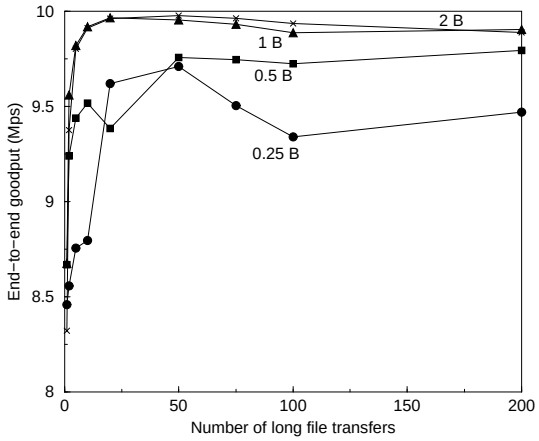


Figure 4: The impact of suboptimal buffer sizes on the goodput of file transfers over TCP NewReno.

the end-to-end goodput both for NewReno and Vegas. However, the improvements come at the price of increasing the round-trip delay. For example, the round-trip delay exceeds 2 seconds in the setting where 100 long file transfers compete for the 1 Mbps bottleneck link. Previous studies for such settings suggest that selecting a smaller buffer size does not necessarily cause a large degradation in the average goodput (even though the end-to-end goodputs of individual file transfers can become substantially more variable) [26]. To examine the sensitivity of the goodput to selecting a suboptimal buffer size, we repeat the experiments with NewReno when the buffer size for the 10 Mbps bottleneck link is set to different fractions of the bitrate-delay product which is denoted as B. As expected, Figure 4 shows that the goodput decreases when the fraction reduces from 2 to 1 to 0.5 to 0.25. However, the amount of the decrease is not substantial and does not grow with the number of applications. Hence, setting the buffer size for the bottleneck link significantly below the optimal value can provide the file transfers with reasonably good end-to-end performance.

5. Impracticality of the Existing Guidelines

As we saw in Section 4, the existing recommendations for the link buffer sizing are derived under many simplifying assumptions such as a single-bottleneck network where file transfers over a loss-driven version of TCP are the only sources of traffic. Despite the simplifications, the derived guidelines are difficult to implement in the context of IP networks where routers are not engaged in extensive signaling with other routers or end systems. Section 3.2 already discussed potential problems with attempts of an IP router to estimate the number of applications by classifying received datagrams into flows. The guideline of setting the link buffer size to the bitrate-delay product of a TCP connection is also problematic in practice. First, the IP router does not know the round-trip propagation delays for TCP connections passing through the router. Second, even if the router could somehow determine the round-trip propagation delays, which of the delays the router should use to compute the bitrate-delay product when the delays are different for different connections? The link buffer size that is optimal for one connection appears as suboptimal from the standpoint of another connection. In response to both concerns, analytical studies suggest that the goodput of file transfers

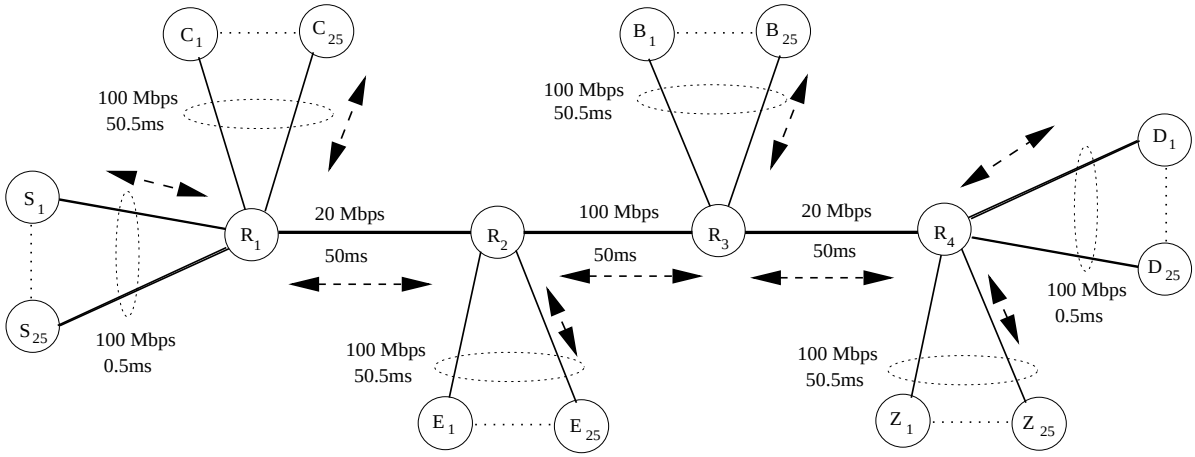
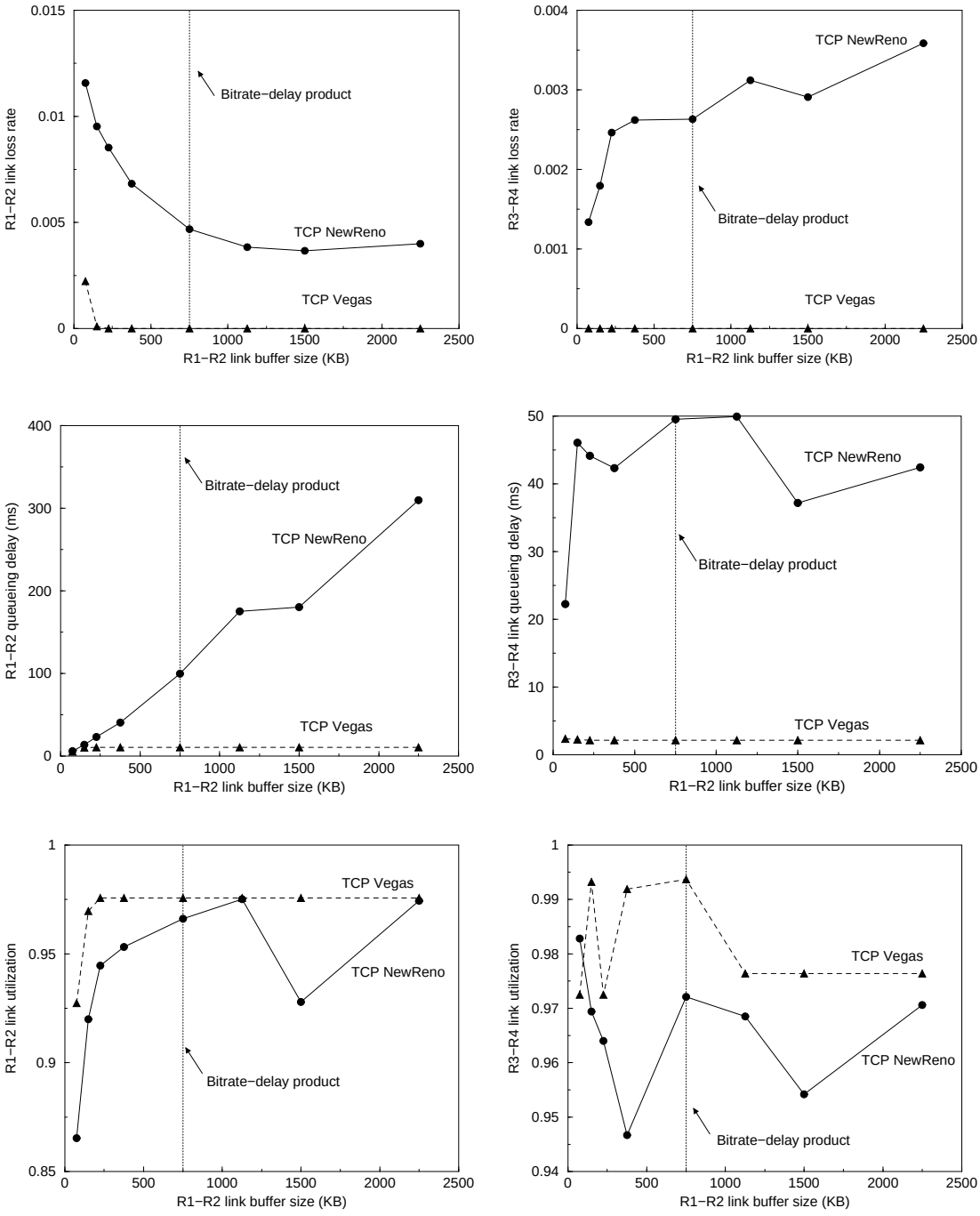


Figure 5: The multiple-bottleneck topology.

over TCP is less sensitive to overestimating the optimal buffer size than to underestimating it [4, 2]. The practice of configuring IP routers seems to follow this theoretical indication to favor large buffers over small buffers – according to anecdotal evidence, it is common for IP router operators to set the link buffer size to the bitrate-delay product where the delay factor is an arbitrarily-chosen large constant such as a "transoceanic delay" or 0.5 second. Existence of such configuration practices is also supported by end-to-end Internet measurements showing that datagrams of the *same* TCP connection can experience round-trip delays that differ by several seconds [1].

Overestimating the bandwidth-delay product is not the only likely reason for excessive queueing in modern IP networks. If the common assertion that links in the Internet core are lightly utilized is correct, then bottleneck links lie at the Internet edge, and a TCP connection can traverse several of them. Hence, we conduct experiments in the multiple-bottleneck topology depicted in Figure 5. The topology contains four routers R_1, R_2, R_3, and R_4 and carries balanced bidirectional traffic on each link. There are three groups of traffic: between end systems S_i and D_i, between end systems C_i and E_i, and between end systems B_i and Z_i (where i varies from 1 to 25). The following applications generate traffic in each group: an interactive video application transmitting one constant-bit-rate 2 Mbps stream over UDP in each direction, eight long file transfers over TCP (four in each direction), and short web downloads (twenty sources in each direction; every source periodically generates a 36-KB data burst, transmits it over TCP, and then goes idle for a time interval that has a duration distributed exponentially with the mean of 1 second). All the applications use datagrams of size 1500 bytes. The applications that rely on TCP employ either NewReno or Vegas versions of it. Each of the links uses FIFO Droptail buffering. Note that all the applications have the same round-trip propagation delay. Hence, the bitrate-delay product for a link is the same from the viewpoint of every application using this link. Our evaluation focuses on the applications transmitting data from end systems S_i. Links R_1-R_2, R_3-R_4, R_4-R_3, and R_2-R_1 are the four bottlenecks for these applications.

Figure 6 reports the link loss rate, queueing delay, and utilization (measured over the whole experiment duration of 200 second) for links R_1-R_2 and R_3-R_4 when the buffer size for link R_1-R_2 varies but the buffer sizes for all the other links are set to their bitrate-delay products. Figure 7 plots



(a) Performance of link R1-R2 (b) Performance of link R3-R4

Figure 6: Link performance in the multiple-bottleneck topology.

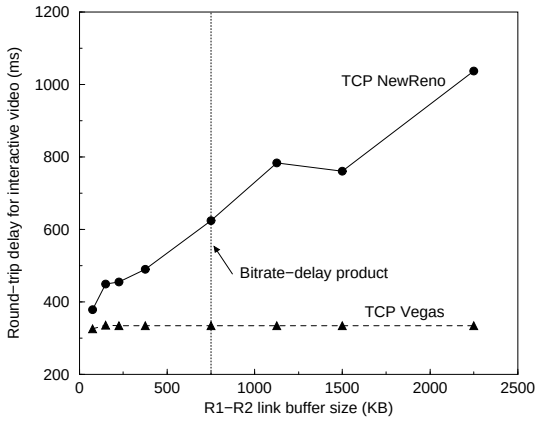
end-to-end performance metrics meaningful for each type of the examined applications: round-trip delay for the interactive video, goodput for the long file transfers, and delivery time for the short web downloads. For TCP NewReno, following the traditional recommendation to set the buffer size for link R_1-R_2 to its bitrate-delay product results in low link loss rates and high link utilizations for both links R_1-R_2 and R_3-R_4, large end-to-end goodputs for the file transfers, and small delivery times for the web downloads. However, the recommended setting inflates the round-trip delay for the interactive video in this multiple-bottleneck topology above 600 ms, a value that is already too large for the application.

6. Alternative Approach to Setting the Link Buffer Size

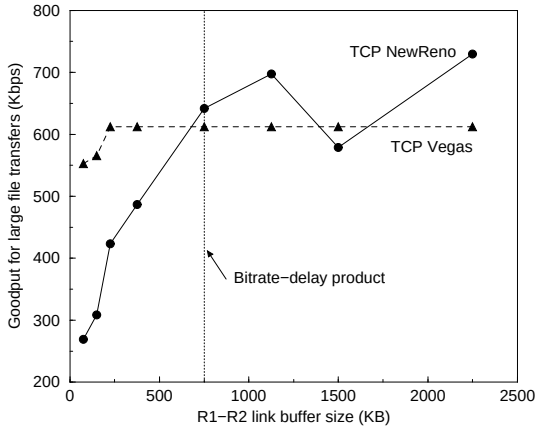
The traditional approaches to selecting the buffer size for an IP network link pursue the objective to minimize the delivery time for file transfers over loss-driven versions of TCP. Whereas the focus on TCP transfers is justifiable by the dominance of TCP traffic in the current Internet, we showed in Section 5 that the existing guidelines are hard to implement precisely in real IP networks and can severely hurt other types of applications such as interactive multimedia requiring small round-trip delays. We believe that the networking research community should change its way of thinking about the link buffer sizing problem. Instead of optimizing performance for applications of a particular type, the goal in solving the problem should be to maximize diversity of application types that IP networks can support effectively.

We also suggest that to achieve this new objective, IP networks should keep end-to-end delay close to its propagation component. Note that the existing guidelines for the link buffer sizing do not exhibit such property; for example, setting the buffer size for each bottleneck link to the bitrate-delay product adds queueing delay that can be as high as the round-trip propagation delay. In contrast, we propose using small buffers in order to enable effective operation of delay-sensitive applications over IP networks.

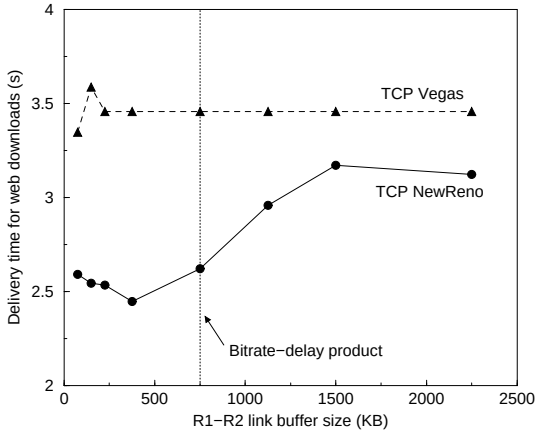
One might argue that our proposal to use small buffers merely shifts the trade-off between the link loss rate, queueing delay, and utilization to benefit interactive applications at the expense of traditional file transfers over TCP. Let us briefly explain why the proposed modification has substantially more important qualitative consequences going beyond a petty tussle of application types for minor improvements in own performance. Because bounded round-trip delays are essential for viability of interactive applications, excessive queuing in the modern Internet hampers wide deployment of such applications. Hence, small buffers can open IP networks to proliferation of innovative delay-sensitive applications. On the other hand, decreasing the link buffer sizes leaves the performance of traditional Internet applications qualitatively the same. The results that we reported for TCP NewReno showed that smaller buffer sizes do not lead to a significant degradation in the file delivery time. Furthermore, a simple analysis – similar to the one in Section 4 where we derived the guideline of the bitrate-delay product from the behavior of a TCP connection in the congestion-avoidance mode – shows that even with minimal buffering, long file transfers over a loss-driven version of TCP utilize at least 75% of the bottleneck link bitrate (more if only a subset of the TCP connections loose their data upon an overflow of the buffer). If the 25% decrease in the throughput of the file transfers is deemed unacceptable, options for improving their performance are not limited to extra queueing that defeats interactive applications. According to our results for TCP



(a) Round-trip delay for interactive video



(b) End-to-end goodput for long file transfers



(c) Delivery time for short web downloads

Figure 7: End-to-end performance for applications of different types.

Vegas, a smoother algorithm for end-to-end congestion control represents a promising alternative. Loss-driven TCP versions such as Tahoe, Reno, NewReno, and SACK can also benefit from this alternative solution; for example, in spite of minimal buffering in the network, a simple change of the decrease factor in the congestion-avoidance mode from 0.5 to 0.875 [10, 28] raises the bottleneck link utilization to 94%.

7. Conclusion

In this paper, we revisited the problem of selecting the buffer size for an IP network link. After a thorough overview of issues relevant to the link buffer sizing, we examined usefulness of existing guidelines for choosing the buffer size. Our analysis showed that the existing recommendations not only are difficult to implement in the context of IP networks but also can severely hurt interactive distributed applications. Then, we argued that the networking research community should change its way of thinking about the link buffer sizing problem: the focus should shift from optimizing performance for applications of a particular type to maximizing diversity of application types that IP networks can support effectively. To achieve this new objective, we proposed using small buffers for IP network links.

References

[1] J. Aikat, J. Kaur, D. Smith, and K. Jeffay. Variability in TCP Round-trip Times. In *Proceedings ACM SIGCOMM Internet Measurement Conference*, October 2003.

[2] E. Altman, K.E. Avrachenkov, and C. Barakat. TCP Network Calculus: The Case of Large Delay-Bandwidth Product. In *Proceedings IEEE INFOCOM 2002*, July 2002.

[3] G. Appenzeller, I. Keslassy, and N. McKeown. Sizing Router Buffers. In *Proceedings ACM SIGCOMM 2004*, September 2004.

[4] K.E. Avrachenkov, U. Ayesta, E. Altman, P. Nain, and C. Barakat. The Effect of Router Buffer Size on the TCP performance. In *Proceedings LONIIS Workshop on Telecommunication Networks and Teletraffic Theory*, January 2002.

[5] L.S. Brakmo, S.W. O'Malley, and L.L. Peterson. TCP Vegas: New Techniques for Congestion Detection and Avoidance. In *Proceedings ACM SIGCOMM 1994*, August 1994.

[6] J.W. Byers, M. Luby, and M. Mitzenmacher. A Digital Fountain Approach to Asynchronous Reliable Multicast. *IEEE Journal on Selected Areas in Communications*, 20(8):1528 – 1540, October 2002.

[7] D. Chiu and R. Jain. Analysis of the Increase and Decrease Algorithms for Congestion Avoidance in Computer Networks. *Journal of Computer Networks and ISDN*, 17(1):1–14, June 1989.

[8] A. Demers, S. Keshav, and S. Shenker. Analysis and Simulation of a Fair Queueing Algorithm. In *Proceedings ACM SIGCOMM 1989*, September 1989.

[9] W. Feng, D. Kandlur, D. Saha, and K. Shin. A Self-Configuring RED Gateway. In *IEEE INFOCOM 1999*, March 1999.

[10] S. Floyd, M. Handley, and J. Padhye. A Comparison of Equation-Based and AIMD Congestion Control. http://www.aciri.org/tfrc/, May 2000.

[11] S. Floyd and T. Henderson. The NewReno Modification to TCP's Fast Recovery Algorithm. RFC 2582, April 1999.

[12] S. Floyd and V. Jacobson. On Traffic Phase Effects in Packet-Switched Gateways. *Internetworking: Research and Experience*, 3(3):115–156, September 1992.

[13] S. Floyd and V. Jacobson. Random Early Detection Gateways for Congestion Avoidance. *IEEE/ACM Transactions on Networking*, 1(4):397–413, August 1993.

[14] S. Floyd and E. Kohler. Internet Research Needs Better Models. In *Proceedings HotNets-I*, October 2002.

[15] S. Gorinsky and H. Vin. Additive Increase Appears Inferior. Technical Report TR2000-18, Department of Computer Sciences, The University of Texas at Austin, May 2000.

[16] P. Goyal, H. M. Vin, and H. Cheng. Start-time Fair Queueing: A Scheduling Algorithm for Integrated Services Packet Switching Networks. In *Proceedings ACM SIGCOMM 1996*, August 1996.

[17] V. Jacobson. Congestion Avoidance and Control. In *Proceedings ACM SIGCOMM 1988*, August 1988.

[18] V. Jacobson. Modified TCP Congestion Control Algorithm. End2end-interest mailing list, April 1990.

[19] D. Lin and R. Morris. Dynamics of Random Early Detection. In *ACM SIGCOMM 1997*, September 1997.

[20] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow. TCP Selective Acknowledgment Options. RFC 2018, October 1996.

[21] R. Morris. Scalable TCP Congestion Control. In *Proceedings IEEE INFOCOM 2000*, March 2000.

[22] UCB/LBNL/VINT Network Simulator ns-2. http://www-mash.cs.berkeley.edu/ns, May 2004.

[23] Univeristy of Southern California. DoD Standard Internet Protocol. RFC 760, January 1980.

[24] J. Postel. User Datagram Protocol. RFC 768, October 1980.

[25] J. Postel. Transmission Control Protocol. RFC 793, September 1981.

[26] L. Qiu, Y. Zhang, and S. Keshav. Understanding the Performance of Many TCP Flows. *Computer Networks*, 37(3-4):277–306, 1999.

[27] K.K. Ramakrishnan and S. Floyd. A Proposal to Add Explicit Congestion Notification (ECN) to IP. RFC 2481, January 1999.

[28] K.K. Ramakrishnan and R. Jain. A Binary Feedback Scheme for Congestion Avoidance in Computer Networks with Connectionless Network Layer. In *Proceedings ACM SIGCOMM 1988*, August 1988.

[29] J. Sun, M. Zukerman, K.-T. Ko, G. Chen, and S. Chan. Effect of Large Buffers on TCP Queueing Behavior. In *Proceedings IEEE INFOCOM 2004*, March 2004.

[30] B. Suter, T.V. Lakshman, D. Stiliadis, and A.K. Choudhury. Buffer Management Schemes for Supporting TCP in Gigabit Routers with Per-Flow Queueing. *IEEE Journal on Selected Areas in Communications*, 17(6):1159–1169, June 1999.

[31] C. Villamizar and C. Song. High Performance TCP in the ANSNET. *ACM SIGCOMM Computer Communication Review*, 24(5):45–60, November 1994.