A Survey on TCP-Friendly Congestion Control

Joerg Widmer, Robert Denda, and Martin Mauve
Praktische Informatik IV, University of Mannheim, Germany

Abstract

New trends in communication, in particular the deployment of multicast and real-time audio/video streaming applications, are likely to increase the percentage of non-TCP traffic in the Internet. These applications rarely perform congestion control in a TCP-friendly manner; they do not share the available bandwidth fairly with applications built on TCP, such as Web browsers, FTP, or e-mail clients. The Internet community strongly fears that the current evolution could lead to congestion collapse and starvation of TCP traffic. For this reason, TCP-friendly protocols are being developed that behave fairly with respect to coexistent TCP flows. In this article we present a survey of current approaches to TCP friendliness and discuss their characteristics. Both unicast and multicast congestion control protocols are examined, and an evaluation of the different approaches is presented.

In the Internet, packet loss can occur as a result of transmission errors, but also, and most commonly, as a result of congestion. TCP's end-to-end congestion control mechanism reacts to packet loss by reducing the number of outstanding unacknowledged data segments allowed in the network. TCP flows with similar round-trip times (RTTs) that share a common bottleneck reduce their rates so that the available bandwidth will be, in the ideal case, distributed equally among them.

Not all Internet applications use TCP and therefore do not follow the same concept of fairly sharing the available bandwidth. Thus far, the undesired effect of the unfairness of these non-TCP applications has not had a heavy impact since most of the traffic in the Internet uses TCP-based protocols such as Hypertext Transfer Protocol (HTTP), Simple Mail Transfer Protocol (SMTP), or File Transfer Protocol (FTP). However, the number of audio/video streaming applications such as Internet audio players, IP telephony, videoconferencing, and similar types of real-time applications is constantly growing, and it is feared that one consequence will be an increase in the percentage of non-TCP traffic. Since these applications commonly do not integrate TCP-compatible congestion control mechanisms, they treat competing TCP flows in an unfair manner: upon encountering congestion, all contending TCP flows reduce their data rates in an attempt to dissolve the congestion, while the non-TCP flows continue to send at their original rate. This highly unfair situation can lead to starvation of TCP traffic, or even to a *congestion collapse* [1], which describes the undesirable situation where the available bandwidth in a network is almost exclusively occupied by packets that are discarded because of congestion before they reach their destination.

For this reason, it is desirable to define appropriate rate adaptation rules and mechanisms for non-TCP traffic that are compatible with the rate adaptation mechanism of TCP. These rate adaptation rules should make non-TCP applications *TCP-friendly*, and lead to a fair distribution of bandwidth.

In this article we present a survey of TCP-friendly congestion control schemes to summarize the state of the art in this field of research and motivate further research on TCP friendliness. We define TCP friendliness and outline the design space for TCP-friendly congestion control. Existing single-rate protocols are discussed, and a detailed survey of multirate protocols is given. The article contains an evaluation of the strengths and weaknesses of the mechanisms presented. We point to open problems and issues for future research and give some concluding remarks.

TCP and TCP Friendliness

TCP is a connection-oriented unicast protocol that offers reliable data transfer as well as flow and congestion control. TCP maintains a congestion window that controls the number of outstanding unacknowledged data packets in the network. Sending data consumes slots in the window of the sender, and the sender can send packets only as long as free slots are available. When an acknowledgment (ACK) for outstanding packets is received, the window is shifted so that the acknowledged packets leave the window, and the same number of free slots becomes available.

On startup, TCP performs *slowstart*, during which the rate roughly doubles each RTT to quickly gain its fair share of bandwidth. In steady state, TCP uses an additive increase, multiplicative decrease (AIMD) mechanism to detect additional bandwidth and to react to congestion. When there is no indication of loss, TCP increases the congestion window by 1 slot/RTT. In case of packet loss, indicated by a timeout, the congestion window is reduced to one slot and TCP reenters the slowstart phase. Packet loss indicated by three duplicate ACKs results in a window reduction to half of its previous size.

Robert Denda is also with ENDITEL Endesa Ingeniería de Telecomunicaciones.

0890-8044/01/$10.00 © 2001 IEEE

Modeling TCP Throughput

The throughput of TCP depends mainly on the parameters RTT t_{RTT}, retransmission timeout value t_{RTO}, segment size s, and packet loss rate p. Using these parameters, an estimate of TCP's throughput can be derived. A basic model that approximates TCP's steady-state throughput T is given by Eq. 1 [1]. This model is a simplification in that it does not take into account TCP timeouts.

Equation 2, presented in [2], gives an example of a more complex model of TCP throughput; b is the number of packets acknowledged by each ACK and W_m is the maximum size of the congestion window. Unlike the model presented by Eq. 1, the complex model takes into account rate reductions due to TCP timeouts. Thus, it models TCP more accurately in an environment with high loss rates.

$$T(t_{RTT}, s, p) = \frac{c \cdot s}{t_{RTT} \cdot \sqrt{p}},$$

where c is a constant value commonly approximated $1.5\sqrt{2/3}$; (1)

$$T(t_{RTT}, t_{RTO}, s, p) = \min\left(\frac{W_m \cdot s}{t_{RTT}}, \frac{s}{t_{RTT}\sqrt{\frac{2bp}{3}} + t_{RTO} \min\left(1, 3\sqrt{\frac{3bp}{8}}\right) p(1+32p^2)} \right). \quad (2)$$

Note, both models assume that the RTT and loss rate are independent of the estimated rate (i.e., they do not take into account that changing the rate can affect the RTT and loss rate). They work well in environments with a high level of statistical multiplexing such as the Internet, but care has to be taken when they are used as part of a protocol's control loop when only a few flows share a bottleneck link. In that case, changes to the sending rate alter the conditions at the bottleneck link, which in turn determine the sending rate through the equation. Such a feedback loop can render the results of both equations invalid.

TCP Friendliness

In [1], non-TCP flows are defined as TCP-friendly when "their long-term throughput does not exceed the throughput of a conformant TCP connection under the same conditions." We prefer to use a slightly more restrictive definition of the term. The definition used throughout this article focuses on the effect of a non-TCP flow on competing TCP flows rather than on the throughput of the non-TCP flow.

TCP Friendliness for Unicast — A unicast flow is considered TCP-friendly when it does not reduce the long-term throughput of any coexistent TCP flow more than another TCP flow on the same path would under the same network conditions.

TCP Friendliness for Multicast — A multicast-flow is defined as TCP-friendly when for each sender-receiver pair, the multicast flow has the property of being unicast TCP-friendly.

It should be noted that there is an ongoing debate on the correct definition of TCP friendliness for multicast. An alternative to the definition given above is to allow multicast flows to use a greater amount of bandwidth than unicast flows, since they serve multiple receivers. In [3] the term *bounded fairness* is introduced to define a situation where the following equation holds true:

$$a \cdot r_{TCP} \le r \le b \cdot r_{TCP},$$

where r is the rate of the multicast flow on the bottleneck link, r_{TCP} is the rate of a TCP flow under the same conditions, and a as well as b are functions of the number of receivers of the flow. For $b = 1$ the two definitions are equivalent. While the latter approach is perfectly valid, we prefer the definition that is more rigid in the protection of competing TCP flows.

With the above definition, TCP friendliness ensures that coexisting TCP flows are not treated unfairly by non-TCP flows. Note, however, that this does not necessarily mean that all TCP and TCP-friendly flows on a bottleneck link receive the same throughput. Even competing flows that use only TCP for congestion control will often not receive the same amount of bandwidth. For example, TCP flows with different RTTs or different numbers of bottlenecks nodes will transmit at different rates [4].

Classification of Congestion Control Schemes

Congestion control schemes can be classified with respect to a multitude of characteristics. In the following we briefly discuss various possible classification schemes for TCP-friendly approaches.

Window-Based vs. Rate-Based

One possible classification criterion for TCP-friendly schemes is whether they adapt their offered network load based on a congestion window or on their transmission rate.

Algorithms that belong to the window-based category use a congestion window at the sender or at the receiver(s) to ensure TCP friendliness. Similar to TCP, each packet transmitted consumes one slot in the congestion window, while each packet received or the acknowledgment of a packet received frees one slot. The sender is allowed to transmit packets only when a free slot is available. The size of the congestion window is increased in the absence of congestion indications and decreased when congestion occurs.

Rate-based congestion control achieves TCP friendliness by dynamically adapting the transmission rate according to some network feedback mechanism that indicates congestion. It can be subdivided into simple AIMD schemes and model-based congestion control. Simple AIMD schemes mimic the behavior of TCP congestion control. This results in a rate that displays the typical short-term sawtooth-like behavior of TCP. This makes simple AIMD schemes unsuitable for continuous media streams. Model-based congestion control uses a TCP model such as the one presented in [2] instead of a TCP-like AIMD mechanism. By adapting the sending rate to the average long-term throughput of TCP, model-based congestion control can produce much smoother rate changes that are better suited to the aforementioned type of traffic. Such schemes do not mimic TCP's short-term sending rate but are still TCP-friendly over longer timescales. However, the congestion control mechanism may not resemble TCP congestion control, and great attention has to be paid to the rate adjustment mechanism to ensure fair competition with TCP or other flows.

Unicast vs. Multicast

TCP friendliness is desirable for both unicast and multicast traffic. However, the design of good multicast congestion control protocols is far more difficult than the design of unicast protocols. Multicast congestion control schemes ideally should scale to large receiver sets and be able to cope with heterogeneous network conditions at the receivers. For example, if for all receivers the sender transmits packets at the same rate, care has to be taken as to how the sending rate is decreased in case of network congestion. This is nontrivial, since in large multicast sessions receivers may experience uncorrelated loss. It is therefore likely that most of the transmitted packets are lost to at least one receiver. If the sender responded to each of these losses by decreasing the congestion window, the transmission would likely stall after a certain length of time. This problem is known as the *loss path multiplicity problem* [5]. Whenever rate adjustment decisions are based not on congestion information from a specific receiver but on the overall congestion information present in the whole distribution tree, protocol performance can suffer considerably if the protocol has not been designed correctly.

Golestani and Sabnani discuss several important aspects of congestion control for multicast in a general fashion [6]. They investigate the different properties of rate-based and window-based approaches. In particular, they show that window-based congestion control can be TCP-friendly without knowing the RTT, whereas rate-based congestion control does need this information in order to be TCP-friendly. This is an important insight, since RTTs are difficult to obtain in a scalable fashion for multicast communication without support from the network.

Single-Rate vs. Multirate

A common criterion for classifying TCP-friendly multicast congestion control protocols is whether they operate at a single rate or use a multirate approach. Obviously, unicast transport protocols are confined to single-rate schemes. In single-rate schemes, data is sent to all receivers at the same rate. This limits the scalability of the mechanism, since all receivers are restricted to the rate that is TCP-friendly for the bottleneck receiver.

Multirate congestion control protocols allow for a more flexible allocation of bandwidth along the different network paths. Such schemes scale better to large receiver sets where increased heterogeneity among receivers is to be expected. A typical approach to multirate congestion control is to use layered multicast: a sender divides the data into several layers and transmits them to different multicast groups. Each receiver can individually select to join as many groups as permitted by the bandwidth bottleneck between that receiver and the sender. The more groups a receiver joins, the better the quality of its reception. For a video transmission an increased number of received layers may improve the video quality, while for reliable bulk data transfer additional layers may decrease the transfer time.

With layered multicast, congestion control is performed indirectly by the group management and routing mechanisms of the underlying multicast protocol. In order for this mechanism to be effective, it is crucial to coordinate join and leave decisions of receivers behind a common bottleneck: if only some receivers leave a layer while others stay subscribed, no pruning is possible and congestion cannot be reduced. In addition, receivers do not make efficient use of the multicast layers when they are not subscribed to a layer that is already present in their subpart of the routing tree. They could receive data at a higher rate at no additional cost. Therefore, receivers that share a bottleneck link should synchronize their decisions to join and leave layers. The *leave latency* is another issue of concern: pruning of the multicast tree upon receipt of leave messages for a layer can take considerable time, on the order of several seconds.

End-to-End vs. Router-Supported

Many of the TCP-friendly schemes proposed are designed for best effort IP networks that do not provide any additional router mechanisms to support the protocols. Thus, they can readily be deployed in today's Internet. These schemes are called *end-to-end* congestion control. They can be further separated into *sender-based* and *receiver-based* approaches.

In sender-based approaches the sender uses information about the network congestion and adjusts the rate or window size to achieve TCP friendliness. The receivers only provide feedback, while the responsibility of adjusting the rate lies solely with the sender.

Receiver-driven congestion control is usually used together with layered congestion control approaches. Here, the receivers decide whether to subscribe or unsubscribe from additional layers based on the congestion situation of the network.

The design of congestion control protocols and particularly fair sharing of resources can be considerably facilitated by placing intelligence in the network (e.g., in routers or separate agents). Congestion control schemes that rely on additional functionality in the network are called *router-supported*. Particularly multicast protocols can benefit from additional network functionality such as feedback aggregation, hierarchical RTT measurements, management of (sub)groups of receivers, or modification of the routers' queuing strategies. *Generic router assist* (GRA) [7], for instance, is a recent initiative that proposes general mechanisms located at routers to assist transport control protocols, which would greatly ease the design and implementation of effective congestion control protocols.

Furthermore, end-to-end congestion control has the disadvantage of relying on the collaboration of the end systems. Experience in the current Internet has shown that this cannot always be assumed: greedy users or applications may use non-TCP-friendly mechanisms to gain more bandwidth. As discussed by Floyd and Fall in [1], some form of congestion control should be enforced by routers in order to prevent congestion collapse. The authors present router mechanisms to identify flows that should be regulated: for instance, when a router discovers a flow which does not exhibit TCP-friendly behavior, the router might drop the packets of that flow with a higher probability than the packets of TCP-friendly flows. While ultimately fair sharing of resources in the presence of unresponsive or non-TCP-friendly flows can only be achieved with router support, this mechanism is difficult to deploy, since changes to the Internet infrastructure take time and are costly in terms of money and effort.

Classification Scheme

In the following sections we use the scheme shown in Fig. 1 to classify the different approaches. This classification distinguishes between single-rate and multirate congestion control at the top level and rate-based vs. window-based congestion control at the second level.

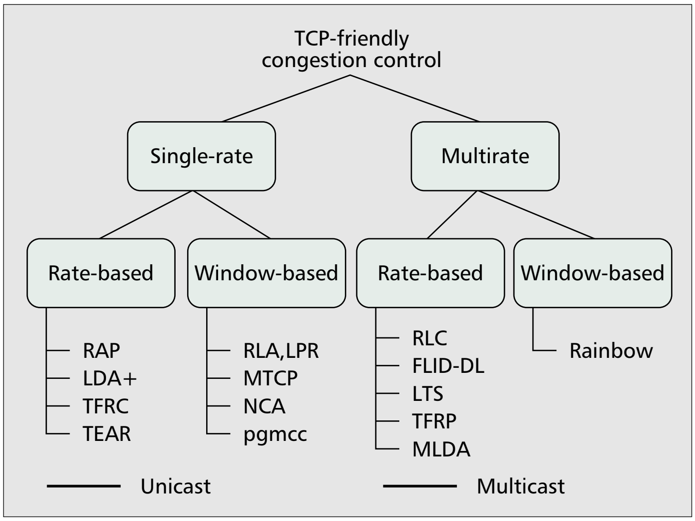
Single-Rate Congestion Control Protocols

In this section a selection of single-rate congestion control protocols is presented. A more complete overview can be found in the corresponding technical report [8].

Rate-Based Approaches

Many rate-based congestion control protocols mimic TCP's AIMD behavior to achieve TCP fairness, while others implicitly or explicitly adjust their rate according to a model of TCP traffic. A very obvious approach to TCP-friendly congestion control is to directly apply TCP's congestion control mechanism, but without the associated reliability mechanism. Early work in this area was presented in [9], where this approach is used to adjust the rate of a unicast video stream in order to adequately react to congestion.

RAP — The Rate Adaption Protocol (RAP) presented in [10] is a simple AIMD scheme for unicast flows. Each data packet is acknowledged by the receiver. The ACKs are used to detect



■ Figure 1. *A classification scheme for TCP-friendly protocols.*

packet loss and infer the RTT. When the protocol experiences congestion it halves the sending rate. In periods without congestion, the sending rate increases by 1 packet/RTT, thus mimicking the AIMD behavior of TCP. The decisions on rate increase or decrease are made once per RTT. To provide additional fine-grained delay-based congestion avoidance, the ratio of a short-term RTT average and a long-term RTT average is used to modify the interpacket gap between consecutive data packets. These fine-grained rate adjustments result in a smoother sending rate.

RAP achieves rates similar to TCP in an environment where TCP experiences no or few timeouts since RAP's rate reductions resemble TCP's reaction to triple duplicate ACKs. However, RAP does not take timeouts into account and is therefore more aggressive when TCP's throughput is dominated by timeout events.

LDA+ — Unlike many of the other schemes, the Loss-Delay Based Adaption Algorithm (LDA+) [11] does not devise its own feedback mechanism to control the sending rate but relies solely on the Real-Time Transport Control Protocol (RTCP) feedback messages provided by the Real-Time Transport Protocol (RTP) [12]. While LDA+ is essentially an AIMD congestion control scheme, it uses some interesting additional elements. The increase and decrease factors for AIMD are dynamically adjusted to the network conditions. An estimate of the bottleneck bandwidth is obtained using packet pairs.[1] The amount of additive increase is then determined as the minimum of three independent increase factors to ensure that:

- Flows with a low bandwidth can increase their rate faster than flows with a higher bandwidth.
- Flows do not exceed the estimated bottleneck bandwidth.
- Flows do not increase their bandwidth faster than a TCP connection.

If receivers report loss, the sending rate is decreased by multiplying by the factor $1 - \sqrt{(l)}$, where l is the loss rate. Additionally, the rate is reduced at most to the rate given by the TCP model as described in Eq. 2. Using the maximum of the AIMD rate and the equation rate may result in a long-term average that exceeds the average rate of the two separate schemes and can be more aggressive than TCP. While LDA+ is designed only for unicast communication, a protocol variant called MLDA can be used for multicast environments. MLDA is discussed later.

Simulations and network experiments the authors conducted show that LDA+ competes fairly with TCP in the investigated environments. The authors claim that the slow rate increase of LDA+ compensates for the fact that the rate is at most decreased to the throughput estimate of the complex TCP model. Using an existing report mechanism facilitates deployment of LDA+. Furthermore, LDA+ adapts its rate increase to prevent overshooting the bottleneck bandwidth. However, RTCP reports are generated infrequently (usually within several seconds). This makes LDA+ slow to react to changes in network conditions. Furthermore, the smallest loss rate standard RTCP can report is limited. In environments with a lower (but positive) loss rate, RTCP reports zero loss and LDA+ may claim more than its fair share of bandwidth.

TFRC — The TCP-Friendly Rate Control Protocol (TFRC) [13] evolved from the TFRCP protocol [14]. It is specified for unicast communication, although with some modifications it can be adapted to multicast. Similar to TFRCP, it adjusts its sending rate based on the complex TCP equation, Eq. 2, but uses more sophisticated methods to gather the necessary parameters. Several requirements for a loss rate estimator are formulated, and the authors settle on the average loss interval method which best fulfills these requirements. The loss rate is measured in terms of loss intervals, spanning the number of packets between consecutive loss events. A certain number of loss intervals is averaged, using decaying weights so that old loss intervals contribute less to the average. The loss rate is calculated as the inverse of the average loss interval size. The authors provide additional mechanisms to prevent the loss rate from reacting too strongly to single loss events and to ensure that the loss rate adapts quickly to long intervals without any losses. The RTT is measured by the standard method of feeding back timestamps to the sender.

Immediately after startup, the sender goes into a slowstart phase similar to TCP slowstart to quickly increase the rate to a fair share of the bandwidth. TFRC slowstart is terminated with the first loss event. Once per RTT the TFRC receiver updates its parameters and sends a state report to the sender. The sender then computes a new fair rate from these parameters and adjusts the sending rate accordingly. To improve protocol performance in environments that do not fulfill the assumptions of the complex TCP equation, TFRC supports additional delay-based congestion avoidance by adjusting the interpacket gap (i.e., the time interval between consecutive data packets).

A major advantage of TFRC is that is has a relatively stable sending rate while still providing sufficient responsiveness to competing traffic.

TEAR — TCP Emulation at Receivers (TEAR) [15] is a hybrid protocol that combines aspects of window-based and rate-based congestion control. TEAR receivers calculate a fair receive rate which is sent back to the sender, who then adjusts the sending rate. To this end, the receivers maintain a congestion window that is modified similarly to TCP's congestion window. Since TCP's congestion window is located at the sender, a TEAR receiver has to try to determine from the arriving packets when TCP would increase or decrease the congestion window size. Additive increase and window reductions caused by triple duplicate ACKs are easy to emulate. However, due to the lack of ACKs, timeout events can be estimated only roughly.

In contrast to TCP, the TEAR protocol does not directly use the congestion window to determine the amount of data

[1] *With packet pairs, the time interval between the receipt of two packets that were sent back-to-back is used as a hint of the current maximum rate for a flow.*

to send but calculates the corresponding TCP sending rate. This rate is roughly a congestion window worth of data per RTT. To avoid TCP's sawtooth-like rate shape, TEAR averages the rate over an *epoch*, which is defined as the time between consecutive rate reduction events. To prevent further unnecessary rate changes caused by noise in the loss patterns, a smooth rate is determined by using a weighted average over a certain number of epochs for the final rate. This value is then reported to the sender, which adjusts the sending rate accordingly. Since the rate is determined at the receivers and TEAR refrains from acknowledging packets, it can be used for multicast as well as for unicast communication, provided a scalable scheme to determine the RTT and report the rates is used in the multicast case. For multicast congestion control, the TEAR sender has to adapt the rate to the minimum of the rates reported by the receivers.

Due to the close modeling of TCP's short-term behavior, TEAR shows TCP-friendly behavior while avoiding TCP's frequent rate changes.

Window-Based Approaches

The domain of window-based unicast congestion control is well covered by TCP. There are two main problems that have to be solved in order to use window-based congestion control for multicast. First, protocols should prevent drop-to-zero of the rate due to the aforementioned loss path multiplicity problem. The second problem is how to free slots in the congestion window. Clearly it is not possible for the sender to receive ACKs for each packet from each receiver, since this would cause an ACK implosion.

In the following we will present several window-based congestion control approaches for multicast transmission. In particular, we will focus on how the two main problems are solved for each.

A Framework for Window-Based Congestion Control — Golestani and Sabnani propose to use a window-based approach where each receiver keeps a separate congestion window adjusted similar to the congestion window of TCP [6]. From the size of the window and the number of outstanding packets, each receiver calculates the highest sequence number it is able to receive without claiming an unfair amount of bandwidth.

This information needs to be communicated to the sender without causing a feedback implosion. As an example of how this can be done, the authors show that a tree structure formed by the receivers or other intermediate systems can be used to aggregate the information: each node takes the minimum sequence number contained in all incoming messages and forwards this sequence number to its parent. When the aggregated information reaches the sender, it is allowed to send packets up to the minimum sequence number it has received. Each receiver maintains its own congestion window, which circumvents the loss path multiplicity problem.

The observations made by Golestani and Sabnani form a theoretical background for window-based multicast congestion control. They need to be concretized by actual algorithms such as those that follow.

RLA and LPR — The Random Listening Algorithm (RLA) proposed by Wang and Schwartz [3] extends TCP selective ACK (SACK) by introducing some enhancements for multicast. For each receiver, the multicast sender stores the smoothed RTT and the measured congestion probability. A loss is detected by the sender via identification of discontinuous ACKs or via timeout. Based on these loss indications, the number of receivers *n* with a high congestion probability is tracked. If congestion is detected, the window is halved in the following two cases:

- If the previous window cut was made too long ago (the authors propose an interval of twice the moving average of the window size times the smoothed RTT of the corresponding receiver)
- If a generated uniform random number π is less than or equal to $1/n$

When a packet has been acknowledged by all receivers, the congestion window *cwnd* is incremented by 1/*cwnd*, identical to TCP. A TCP-like retransmission scheme with fast recovery is also included in RLA. With the above mechanisms, RLA avoids the loss path multiplicity problem, while achieving statistical long-term fairness. In [3] it is demonstrated that RLA is fair to TCP according to the definition of bounded fairness.

Linear Proportional Response (LPR), proposed by Bhattacharyya, Towsley, and Kurose [16], is a probabilistic loss indication filtering scheme that is an improvement over the corresponding RLA mechanism. The probability with which a multicast source reduces its congestion window size is proportional to the loss probability at the receiver; that is, the window size is halved when π is smaller than

$$X_i / \sum_{j=1}^{n} X_j,$$

where X_i is the number of losses at receiver i. The LPR scheme achieves better fairness of multicast sessions toward competing unicast sessions than does the window adjustment indication scheme of RLA. Even though we are not aware of any extensive measurements in real scenarios, the mathematical proofs and simulations in [16] give strong evidence that when combined with the window adjustment mechanism of RLA, LPR achieves good TCP friendliness.

MTCP — Multicast TCP (MTCP) [17] is a reliable multicast protocol that uses window-based congestion control to achieve TCP friendliness. MTCP groups the session participants into a logical tree structure where the root of the tree is the sender of the data. A parent in the logical tree structure stores a received packet until receipt is acknowledged by all of its children. Upon receiving a packet, a child (which may be a parent for other participants) transmits an ACK to its parent using unicast.

To control congestion, MTCP requires that each parent maintain two values: a congestion window and a transit window. The size of the congestion window is managed similarly to that of TCP, including slow-start and congestion avoidance. The main differences to TCP are:

- The congestion window is only incremented when ACKs from all children have been received.
- A packet is immediately (re)transmitted to a child if it indicates via a negative ACK (NACK) that it has not yet received the packet.

The size of the congestion window is halved when any child reports three consecutive NACKs or set to one when a timeout occurs because a child has not acknowledged a packet at all. The transit window keeps track of the amount of data the children of a parent node have not yet acknowledged.

With each ACK, a parent node transmits a congestion summary to its own parent. This congestion summary contains the minimum of its own congestion window size and those reported by its children, as well as the maximum of its own transit window size and those reported by its children. The sender is then allowed to transmit the difference between the minimum congestion window size and the maximum transit window size.

In MTCP, the loss path multiplicity problem is avoided by means of the aggregation at the intermediate nodes. Each node forwards the information about the bottleneck link of its children to its parent. Therefore, the sender will receive information about the overall bottleneck link rather than about uncorrelated packet loss. The main drawback to MTCP is its complexity and required setup of a tree structure where each node has to perform package storage, repair, and congestion monitoring functionality.

NCA and pgmcc — Nominee-Based Congestion Avoidance (NCA) presented in [18] and pragmatic general multicast congestion control (pgmcc) [19] are two approaches to congestion control that share the same fundamental idea: they select as a group representative the bottleneck receiver with the worst network connection. This receiver acknowledges every packet received and thereby allows the sender to use a TCP-style congestion control algorithm. It is important to note that in this approach congestion control and packet repair are treated independent of each other. Thus, the approach can be used in combination with a large number of mechanisms that establish reliability, as well as for unreliable data transmission.

The most challenging aspect of NCA and pgmcc is how to select the group representative. In both approaches, each receiver calculates the data rate at which it is able to receive by using a simple TCP rate formula. This formula takes into account the RTT and the loss rate experienced by the receiver. The information about the acceptable rate is conveyed back to the sender either piggybacked on NACKs (pgmcc) or accumulated in a tree structure of routers that always forward the report of the participant with the lowest acceptable data rate (NCA). From those reports the sender selects as the representative the participant with the lowest acceptable rate and uses a TCP-like congestion control mechanism to this participant.

This approach seems very promising, since it closely mimics the behavior of unicast TCP and therefore should lead to fairness with regard to TCP flows if the proper representative is chosen. The main problem is that the selection process is based on a rough estimate of the acceptable data rate. Further insight is needed as to whether network conditions exist where the wrong representative is selected. This could lead to unfair behavior against other flows. The author of pgmcc indicates that this may occur when a set of receivers has lossy links with a low RTT and congested links with a high RTT.

Multirate Congestion Control Protocols

In the following, a selection of promising multirate congestion control protocols is presented. Again, for a more complete overview please refer to [8].

Rate-Based Approaches

One of the first working examples of layered multicast transmission in the Internet was Receiver-Driven Layered Multicast (RLM) for the transmission of video, developed by McCanne, Jacobson, and Vetterli [20]. Their work did not focus on TCP friendliness but on how to provide each receiver with the best possible video quality in dependence on the bandwidth available between the sender and that receiver. In RLM the sender splits the video into several layers. A receiver starts receiving by subscribing to the first layer. When the receiver does not experience congestion in the form of packet loss for a certain period of time, it subscribes to the next layer. This is called a *join experiment*. When a receiver experiences packet loss, it unsubscribes from the highest layer it is currently receiving.

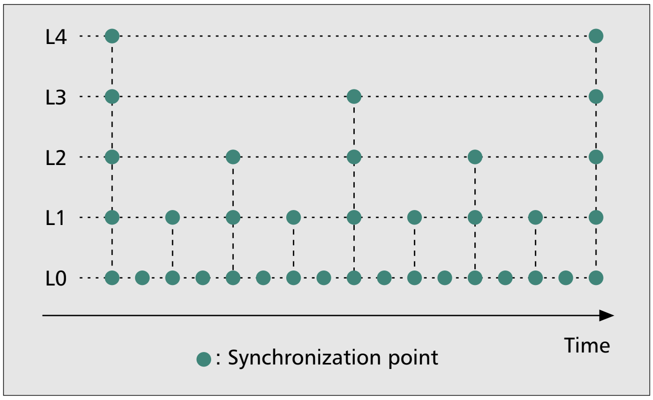
The use of RLM to control congestion is problematic since RLM's mechanism of adding or dropping a single layer based on the detection of packet loss is not TCP-friendly and can result in an unfair distribution of bandwidth among concurrent RLM sessions. Furthermore, leaving a multicast group may take a significant amount of time, usually on the order of several seconds. Failed join experiments (i.e., a receiver joining a layer immediately has to leave again because the necessary bandwidth is not available) are therefore very costly in terms of the additional congestion they may cause. As mentioned earlier, in order for layered schemes to be efficient, it is imperative that receivers behind the same bottleneck synchronize their join and leave decisions. Several protocols have been developed that improve the original concept of RLM.

RLC — Vicisano, Crowcroft, and Rizzo address most of these problems in their work on Receiver-Driven Layered Congestion Control (RLC) [21]. They propose to dimension the layers so that the bandwidth consumed by each new layer increases exponentially. Layer 1, for example, carries twice as much data in the same amount of time as layer 0. The time a receiver has to wait before being allowed to join a new layer also increases exponentially with each additional layer. On the other hand, a layer is dropped immediately when congestion becomes apparent in form of packet loss. This emulates the behavior of TCP since the increase in bandwidth is proportional to the amount of time required to pass without packet loss before being allowed to join the layer. At the same time the reaction to congestion is a multiplicative decrease, since dropping one layer results in halving the overall receive rate.

To improve synchronization between receivers, receivers may join a layer only at so-called synchronization points (SPs). SPs in higher layers are exponentially less frequent than in lower layers (Fig. 2). Thus, a receiver that has only subscribed to a small number of layers is likely to catch up with receivers with a higher subscription level. After some time, receivers that share the same bottleneck should be joining and leaving layers synchronously. In order to decrease the likelihood that a join experiment will fail, the RLC sender creates a short burst period before an SP. During this burst period the data rate is doubled in each layer. Only if a receiver does not experience any signs of congestion during the burst is it allowed to join the next higher layer.

Despite the improvements in the congestion control mechanism over RLM, RLC still has some drawbacks. The granularity at which the rate can be adapted to the network conditions is very coarse and may cause unfair behavior. The exponential distribution of the layers only allows doubling or halving the receive rate. The second problem is that the transmitted data must support layering. While this is true for video and bulk data transmission, streams that are more interactive like those produced by shared whiteboards cannot easily be separated into multiple layers. RLC does not take the RTT into account when determining the sending rate. This can lead to unfairness toward TCP since TCP is biased against connections with a high RTT. Furthermore, it is not guaranteed that the artificial bursts of packets introduced by RLC are acceptable for a broad range of applications that support layered transmission. A general point of controversy that applies to all layered congestion control schemes is whether it is acceptable to "abuse" network mechanisms like multicast routing to achieve transport layer functionality like congestion control.

FLID-DL — To address some of the deficiencies of RLC, Byers *et al.* propose Fair Layered Increase/Decrease with Dynamic Layering (FLID-DL) [22]. The protocol uses a digital fountain [23] at the source. With digital fountain encoding, the sender encodes the original data and redundancy information such that receivers can decode the original data once they have received a fixed number of arbitrary but distinct packets. Since it is not necessary to ensure delivery of specific packets, the layering scheme is much more flexible.



■ Figure 2. *Synchronization points in RLC.*

FLID-DL introduces the concept of dynamic layering to reduce the join and leave latencies associated with adding or dropping a layer. With dynamic layering, the bandwidth consumed by a layer decreases over time. Thus, a receiver has to periodically join additional layers to *maintain* its receive rate. The receive rate is reduced simply by not joining additional layers, whereas rate increase requires joining multiple layers. To reduce the total number of layers required by the mechanism, layers are reused after a quiet period where no data has been transmitted over the layers for a certain amount of time. This scheme provides an elegant solution to avoiding the effect of long leave latencies, provided that the quiet period is sufficient for normal leave operations to take effect.

Dynamic layering is complemented by an FLID scheme which results in a receive rate that is fair to a TCP flow with a fixed RTT experiencing the same loss rate. FLID retains RLC's concepts of sender-initiated synchronization points to coordinate receivers but refrains from packet bursts to probe for available bandwidth. FLID uses probabilistic increase signals such that receivers subscribe to additional layers only with a certain probability. These probabilities are chosen so as to achieve a rate compatible with TCP.

The FLID-DL protocol is a considerable improvement over RLC and can be considered to be state of the art for layered congestion control. It does not suffer from long leave latencies and is more flexible with regard to the bandwidth distribution on the layers. However, like RLC, FLID-DL does not take into account the RTT and thus exhibits unfair behavior toward TCP under certain network conditions. It also results in major overhead for the underlying multicast routing protocol since join and leave decisions occur much more frequently.

LTS and TFRP — Two similar congestion control protocols for the transmission of video streams are presented by Turletti *et al.* and Tan and Zakhor. The Layered Transmission Scheme (LTS) [24] and the TCP-Friendly Transport Protocol (TFRP) [25] both refrain from join experiments to probe for available bandwidth, using instead the simple TCP Eq. 1 to adjust the rate. Receivers simply adjust their subscription level to the rate given by the equation. The necessary parameters of loss rate and RTT are measured at the receivers in a straightforward fashion. While these protocols are easy to implement, they suffer from a multitude of drawbacks. Tan and Zakhor do not address the problem of how to measure the RTTs to the receivers in a scalable way. In LTS, the RTTs are measured simply by having the receivers send *RTT request messages* to the sender, which then multicasts the timestamps contained in those messages back to all receivers. This can pose a problem for very large receiver sets. The simple TCP equation gives only a reasonable estimate of TCP throughput under low loss rates. To prevent rate oscillations, it is necessary to accurately measure and smoothe loss and RTT values through filtering.

MLDA — The Multicast Loss-Delay Based Adaption Algorithm (MLDA) [26] is a congestion control protocol that uses layered multicast. It builds on the previously discussed LDA+ protocol, also using RTCP reports for the signaling between the sender and the receivers. MLDA retains the increase and decrease behavior of LDA+ but performs the rate calculation at the receivers. The receivers report the rate to the sender, avoiding feedback implosion by using exponentially distributed timers. The sender continuously adjusts the bandwidth distribution of the layers to support the reported rates. Independently, the receivers adjust their subscription level to the appropriate receive rate. Thus, MLDA combines the two concepts of sender- and receiver-based congestion control. To calculate the rate, the RTT has to be measured at the receivers. The authors present a complex mechanism to obtain sufficiently accurate RTT estimates in the face of very infrequent RTCP reports. At certain points in time, a receiver measures the RTT using the well-known scheme of having the sender feed back a timestamp value. This accurate measurement is then continuously modified using the one-way delay between the sender and the receiver. The authors take a possible offset between the clocks at sender and receiver into account and filter out irregularities in the one-way delay estimates.

The authors demonstrate the TCP-friendly behavior of MLDA through extensive simulations and compare the performance of their protocol to that of other layered congestion control schemes. By reducing the rate on a layer that causes congestion rather than waiting for all receivers behind a bottleneck to leave the corresponding multicast group, MLDA can react to congestion faster than other layered schemes. A major disadvantage of MLDA is the complexity of the protocol and the added complexity of the application that has to distribute the data onto the dynamic layers.

Window-Based Approaches

Rainbow — Rainbow [27] is a window-based congestion control scheme for the reliable transfer of bulk data. Like FLID-DL, the data is encoded using a digital fountain. Thus, it is not important what specific packets a receiver gets, but only how many distinct packets it receives.

The key idea of Rainbow is that receivers individually request the transmission of each data packet. Each receiver keeps a congestion window, and each request is marked with a label that essentially indicates the position of the request in the congestion window. If multiple requests with the same label arrive from distinct receivers, these requests are accumulated by intermediate routers. In addition, the routers store information about the requests they have received. This process is shown in Fig. 3. The router that is closest to the sender delivers the requests to the sender, which in turn sends a packet in response to each request. The packets are forwarded by the routers in the reverse direction of the requests. The routers delete the information about requests as the packet is forwarded toward the receivers. Thus, this congestion control scheme relies heavily on additional intelligence in the routers.

The congestion window in the receivers imitates the behavior of the TCP congestion window. When a data packet arrives with a label that falls in the current congestion window, a new request is immediately transmitted. The congestion window size is either increased by one for each data packet received (during slowstart) or increased by one when a full congestion window has been received (during congestion avoidance). When a packet loss is detected, the window size is halved.

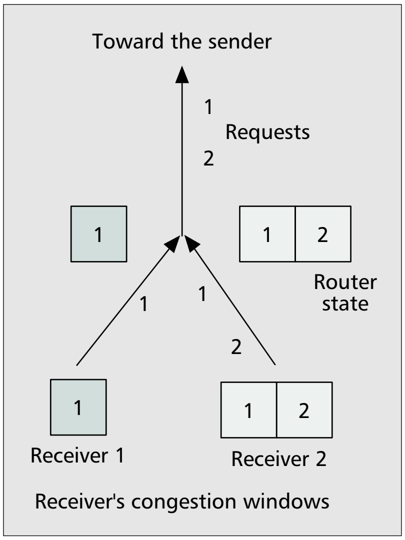
Rainbow is currently the only window-based congestion control approach that allows participants to receive data at different rates. This behavior is made possible by the special encoding of the data and the individual requests for data transmitted by each participant. There are two main limitations of Rainbow:

- It must be possible to use digital fountain encoding for the data.
- The routers must support the accumulation and storage of requests.

Protocol Evaluation

Which congestion control mechanism is suitable for a given task depends mostly on the network characteristics and the traffic requirements of the sending application.

In a controlled environment such as a company's intranet it is possible to implement solutions that require changes to the network infrastructure. However, deployment of these mechanisms in the global Internet is a much more difficult task that consumes time and is very costly. Thus, such solutions are likely to be used only if they offer vastly improved performance over solutions that can be used with today's Internet infrastructure. Among protocols that fall into the former category are tree-based protocols such as MTCP, and protocols that use a multicast service other than standard IP multicast such as Rainbow.



■ Figure 3. *Rainbow.*

The same considerations hold true for protocol complexity. The higher the complexity, the better the performance and fairness should be to justify the additional overhead. Simple rate-based AIMD schemes such as RAP require the least complexity but have the same large variations in the data rate as TCP. Furthermore, if they rely only on AIMD but do not take TCP timeouts into account, their TCP-friendliness is very limited. Because of the similarity of the congestion control mechanism, window-based congestion control schemes generally show good TCP friendliness. Their complexity, with regard to the congestion control mechanism, is comparable to rate-based AIMD schemes.

Model-based congestion control schemes require a moderately higher amount of complexity. In addition to the computation of the model, the measurement of the necessary parameters in a fashion that avoids unwanted behavior (e.g., rate oscillations) adds to the complexity, even more so in multicast environments. Schemes based on the simple TCP equation will estimate a sending rate that is too high in very lossy environments and thus will be more aggressive than TCP. Model-based schemes may fail to produce a fair sending rate when the network conditions do not comply with the assumptions made for the network model on which the equation is based.

Receiver and sender complexity can be reduced by moving intelligence into the network. However, as mentioned before, increased complexity in the network is even less desirable than increased complexity at the receivers and the sender. Usually, such router support is used for feedback suppression and for scalable aggregation of protocol information.

The layering of data further increases the level of complexity since the sender has to split up the original data, and the receivers again have to merge the layers they receive. While the overall throughput of a layered congestion control scheme is higher than that of a scheme that adapts to the worst receiver, it pays the price of reduced responsiveness. The mechanism of joining and leaving layers was not intended for congestion control. The long leave latencies of IP multicast prevent quick reaction to congestion unless an additional mechanism like dynamic layering is used. Furthermore, the granularity of the rate modification of layered schemes is very low compared to that of single-rate schemes. However, for very large heterogeneous receiver sets only multirate protocols provide a viable mechanism of rate control. Single-rate protocols suffer too much from their adaptation to the worst receiver in such environments.

The efficiency of layered schemes also depends on the type of underlying multicast routing protocol. While they work well with dense-mode multicast routing, sparse-mode routing protocols such as PIM-SM [28] can be problematic. In PIM-SM, the distribution tree for a group can change from a core-based tree (or rendezvous point tree) to a shortest path tree in the course of a session. This, combined with frequent joining and leaving of layers for congestion control, may result in unstable behavior.

The requirements of delay- and bandwidth-sensitive data streams make rate-based approaches more suitable than window-based approaches for applications that transmit audio or video. However, use of a rate-based congestion control scheme does not guarantee a smooth sending rate. For example, the sending rate of AIMD schemes resembles that of TCP and is thus not very suitable for applications requiring a stable sending rate.

Table 1 shows the main characteristics of the presented protocols. It classifies the protocols according to whether they support multicast and with regard to the type of their congestion control mechanism. Protocols that work end to end can be completely implemented in the end nodes and do not need additional support from the network. The complexity rating in the table takes into account only the complexity of the congestion control mechanism. Note that the overall complexity of the protocols also includes additional complexity required in the network or for layering of the data. The next column indicates whether the protocol can be used for applications that rely on a relatively stable sending rate. The rating refers to protocol behavior in steady state, given a static environment with periodic loss. Smoothness of the rate in real network environments depends largely on responsiveness and the specific parameters used for the increase and decrease mechanism; it is therefore difficult to predict. Generally, protocols with a "sawtooth-like" sending rate will show more rate oscillations. Smoothness for layered protocols depends on the number of layers used and the bandwidth distribution among them. Most layered protocols use a relatively small number of layers, which results in wide variations in the sending rate. TCP throughput degrades with higher RTTs. For that reason, a protocol that wants to comply with TCP throughput has to be biased against high-RTT connections. The RLC and FLID-DL protocols do not exhibit this bias and can thus be unfair, as explained in the detailed sections about the protocols.[2] The TCP friendliness rating is based on the evaluation by the authors of the protocol and on theoretical considerations such as whether the protocol takes TCP timeouts into account or has a rate increase that is more aggressive than that of TCP. Unfortunately, there exists no direct comparison of TCP-

[2] *Some research is concerned with removing TCP's bias against high RTTs. If those efforts turn out to improve TCP's behavior, the aforementioned protocols would be fair to TCP.*

friendly congestion control schemes in the form of standardized simulations. Therefore, this rating is — to a certain degree — subjective. Protocols rated "good" are expected to show no signs of unfair behavior toward TCP. Protocols rated "acceptable" are likely to show good TCP friendliness in general, but may be problematic in special cases. The "limited TCP friendliness" rating was given to protocols where there are clear signs of unfair behavior toward TCP also under not so unusual network conditions (e.g., high loss rates).

Areas of Future Research

As is always the case with an evolving research area, several unresolved issues remain. One particular problem is the lack of standard methods to compare congestion control protocols. Thus, an evaluation as in the previous section is often based on hints given in the corresponding papers and quite a bit of guesswork. A test environment (the *ns* network simulator comes to mind) with a standardized suite of test scenarios that investigate different important aspects such as fairness and scalability, combined with measures to directly compare the protocol performance, would be very handy. While such a testbed is not sufficient to explore all details of a specific protocol, it would provide a reasonable basis for more objective comparisons of the protocols.

In many cases the simulation scenarios presented by the developers of a protocol concentrate on a few general scenarios and are often too simple to capture protocol behavior in nonstandard situations. Claims about a protocol that are based purely on simulation results should be taken with a grain of salt. Traffic conditions in the Internet are too complex to be modeled in all aspects in a network simulator, making it important to evaluate protocols under real-world conditions as well.

We already discussed the characteristics of single-rate and layered congestion control. It may well be possible that different forms of congestion control are viable — maybe with router support — that do not exhibit the disadvantages of these approaches (e.g., the possibility of different rates to the receivers without the coarseness of layering and long leave latencies).

While TCP friendliness is a useful fairness criterion in today's Internet, it is well possible that future network architectures (in which TCP is no longer the predominant transport protocol) will allow or require different definitions of fairness. Also, fairness definitions for multicast are still subject to research. We presented one possible definition and also briefly addressed a different form where multicast flows are allowed to consume a higher percentage of bandwidth than are unicast flows, but these are by no means the only possible fairness definitions.

A further area of research is the improvement of the models for TCP traffic that are used for some of the rate-based congestion control mechanisms. Current TCP formulae are based on several assumptions that are often not met in real-world environments.

One aspect of congestion control that is not directly relevant to the traffic discussed in this article (i.e., streaming media traffic) but highly relevant to congestion control in general is how to treat short-lived flows that consist only of a few data packets. TCP congestion control, as well as the congestion control schemes presented in this article, require that flows persist for a certain amount of time; otherwise, those forms of congestion control are meaningless.

Many current congestion control protocols are still in the developmental phase, and little attention is paid to the fact that not all receivers share the same goal as the sender. It has been shown that conformant TCP senders can easily be tricked into sending at a higher rate by modifying the TCP

Protocol	Unicast/ multicast	Congestion control mechanism	Network support	Protocol complexity	Smoothness of the rate	Bias against high RTTs	TCP friendliness
Single-rate							
RAP	Unicast	Rate	End-to-end	Low	Sawtooth	Yes	Limited
LDA(+)	Unicast	Rate	End-to-end	High	Sawtooth	Yes	Acceptable
TFRC	Unicast	Rate	End-to-end	Medium	Smooth	Yes	Good
TEAR	Multicast	Rate	End-to-end	Low	Smooth	Yes	Good
RLA & LPR	Multicast	Window	End-to-end	Low	Sawtooth	Yes	Good
MTCP	Multicast	Window	End-to-end	Low	Sawtooth	Yes	Good
NCA	Multicast	Window	Required	Low	Sawtooth	Yes	Good
PGMCC	Multicast	Window	Required	Medium	Sawtooth	Yes	Good
Multirate							
RLC	Multicast	Window	Optional	Medium	Sawtooth	Yes	Good
FLID-DL	Multicast	Rate	End-to-end	Medium	Layer-dependent	No	Acceptable
LTS	Multicast	Rate	End-to-end	High	Layer-dependent	No	Acceptable
TFRP	Multicast	Rate	End-to-end	Medium	Layer-dependent	Yes	Acceptable
MDLA	Multicast	Rate	End-to-end	Medium	Layer-dependent	Yes	Acceptable
Rainbow	Multicast	Rate	End-to-end	High	Layer-dependent	Yes	Acceptable

Table 1. *Characteristics of the presented congestion control protocols.*

receiver [29]. The same holds true for most of the protocols presented here. Only single-rate multicast protocols with large receiver sets are usually immune since a single receiver that claims to be able to receive at a higher rate than it actually is will simply not contribute to the congestion control process. Before the large-scale deployment of new protocols it is necessary to also investigate the aspect of malicious receivers.

The form of congestion control that will eventually be used, be it end to end, router supported, or a hybrid of both, depends largely on if or when such support will be made available by router manufacturers. First efforts in the direction of router support are evidenced by the experiments of a major router manufacturer with the Pragmantic General Multicast protocol (PGM).

Conclusions

With this work, we present a survey on recent advances in the area of TCP-friendly congestion control. We discuss the need for TCP-friendly congestion control for both non-TCP-based unicast traffic and multicast communication, and give an overview of the design space for such congestion control mechanisms.

Throughout the article we analyze various approaches that provide TCP friendliness, by either restricting all receivers in such a way that TCP friendliness is achieved for the worst receiver (single-rate) or adapting the rate of each receiver individually in a TCP-friendly manner (multirate). Furthermore, we classified the protocols according to the type of their congestion control mechanism and their need for network support.

We believe that given the queuing and forwarding mechanisms of the current Internet, TCP friendliness is essential for end-to-end transport protocols. Eventually, router mechanisms that enforce TCP-friendly behavior and punish nonconformant streams will be necessary as an incentive for end-to-end congestion control. Appropriate enforcement mechanisms at routers need to be investigated, and although initial theoretical approaches to implement scalable and efficient algorithms exist, a great deal of work is necessary before they can be deployed. Until then, the efficiency of the Internet depends on the collaboration of applications by using TCP-friendly congestion control protocols.

Acknowledgments

We would like to thank Mark Handley, Wilbert de Graaf, and the anonymous reviewers for their very helpful comments on the article.

References

[1]S. Floyd and K. Fall, "Promoting the Use of End-to-end Congestion Control in the Internet," *IEEE/ACM Trans. Net.*, vol. 7, no. 4, Aug. 1999, pp. 458–72.

[2] J. Padhye *et al.*, "Modeling TCP Reno Performance: A Simple Model and Its Empirical Validation," *IEEE/ACM Trans. Net.*, vol. 8, no. 2, Apr. 2000, pp. 133–45.

[3] H. A. Wang and M. Schwartz, "Achieving Bounded Fairness for Multicast and TCP Traffic in the Internet," *Proc. ACM SIGCOMM*, 1998.

[4] M. Vojnovic, J. Y. Le Boudec, and C. Boutremans, "Global Fairness of Additive-Increase and Multiplicative-Decrease with Heterogeneous Round-Trip Times," *Proc. IEEEa INFOCOM 2000*, Tel Aviv, Israel, Mar. 2000.

[5] S. Bhattacharyya, D. Towsley, and J. Kurose, "The Loss Path Multiplicity Problem in Multicast Congestion Control," *Proc. IEEE INFOCOM*, New York, NY, Mar. 1999, vol. 2, pp. 856–63.

[6] S. J. Golestani and K. K. Sabnani, "Fundamental Observations on Multicast Congestion Control in the Internet," *Proc. INFOCOM '99*, Mar. 1999, vol. 2, pp. 990–1000.

[7] B. Cain, T. Speakman, and D. Towsley, "Generic Router Assist GRA Building Block Motivation and Architecture," Internet draft draft-ietf-rmt-gra-arch-01.txt, Mar. 2000, work in progress.

[8] J. Widmer, R. Denda, and M. Mauve, "A Survey on TCP-Friendly Congestion Control (Extended Version)," Tech. rep. TR-2001-002, Dept. of Math. and Comp. Sci., Univ. of Mannheim, Feb. 2001.

[9] S. Jacobs and A. Eleftheriadis, "Providing Video Services over Networks Without Quality of Service Guarantees," *W3C Wksp. Real-Time Multimedia and the Web*, Oct. 1996.

[10] R. Rejaie, M. Handley, and D. Estrin, "Rap: An End-to-End Rate-Based Congestion Control Mechanism for Realtime Streams in the Internet," *Proc. IEEE INFOCOM*, Mar. 1999.

[11] D. Sisalem and A. Wolisz, "LDA+ TCP-friendly adaptation: A Measurement and Comparison Study," *Proc. Int'l. Wkshp. Network and Op. Sys. Support for Digital Audio and Video*, June 2000.

[12] H. Schulzrinne *et al.*, "Rtp: A Transport Protocol for Real-time Applications," RFC 1889, Jan. 1996.

[13] S. Floyd *et al.*, "Equation-based Congestion Control for Unicast Applications," *Proc. ACM SIGCOMM*, Stockholm, Sweden, Aug. 2000, pp. 43–56.

[14] J. Padhye, D. Kurose, and R. Towsley, "A model based TCP-friendly rate control protocol," *Proc. Int'l. Wksp. Network and Op. Sys. Support for Digital Audio and Video*, June 1999.

[15] I. Rhee, V. Ozdemir, and Y. Yi, "TEAR: TCP Emulation at Receivers - Flow Control for Multimedia Streaming," Tech. rep., Dept. of Comp. Sci., NCSU, Apr. 2000.

[16] S. Bhattacharyya, D. Towsley, and J. Kurose, "A Novel Loss Indication Filtering Approach for Multicast Congestion Control," *J. Comp. Commun.*, Special Issue on Multicast, 2000.

[17] I. Rhee, N. Balaguru, and G. Rouskas, "MTCP: Scalable TCP-Like Congestion Control for Reliable Multicast," *Proc. IEEE INFOCOM*, Mar. 1999, vol. 3, pp. 1265–73.

[18] S. Kasera *et al.*, "Scalable Fair Reliable Mulitcast Using Active Services," *IEEE Net.* (Special Issue on Multicast), vol. 14, no. 1, Jan./Feb. 2000, pp. 48–57.

[19] L. Rizzo, "Pgmcc: A TCP-friendly single-rate Multicast Congestion Control Scheme," *Proc. ACM SIGCOMM*, Stockholm, Sweden, Aug. 2000, pp. 17–28.

[20] S. McCanne, V. Jacobson, and M. Vetterli, "Receiver-driven Layered Multicast," *Proc. ACM SIGCOMM*, Palo Alto, CA, Aug. 1996, pp. 117–30.

[21] L. Vicisano, J. Crowcroft, and L. Rizzo, "TCP-like Congestion Control for Layered Multicast Data Transfer," *Proc. IEEE INFOCOM*, Mar. 1998, vol. 3, pp. 996–1003.

[22] J. Byers *et al.*, "FLID-DL: Congestion Control for Layered Multicast," *Proc. 2nd Int'l Wkshp. Networked Group Commun.*, Palo Alto, CA, Nov. 2000.

[23] J. Byers *et al.*, "A Digital Fountain Approach to Reliable Distribution of Bulk Data," *Proc. ACM SIGCOMM '98*, Sept. 1998.

[24] T. Turletti, S. Parisis, and J. Bolot, "Experiments with a Layered Transmission Scheme over the Internet," Tech. rep. RR-3296, INRIA, France, Nov. 1997.

[25] W. Tan and A. Zakhor, "Error Control for Video Multicast Using Hierarchical FEC," *Proc. Int'l. Conf. Image Processing*, Oct. 1999.

[26] D. Sisalem and A. Wolisz, "MLDA: A TCP-friendly Congestion Control Framework for Heterogenous Multicast Environments," *8th Int'l. Wksp. QoS*, June 2000.

[27] K. Yano and S. McCanne, "A Window-based Congestion Control for Reliable Multicast Based on TCP Dynamics," *Proc. ACM Multimedia*, Oct. 2000.

[28] D. Estrin *et al.*, "Protocol Independent Multicast Sparse-mode (pimsm): Protocol Specification," IETF, RFC 2362, June 1998.

[29] S. Savage *et al.*, "TCP Congestion Control with a Misbehaving Receiver," *ACM Comp. Commun. Rev.*, vol. 29, no. 5, Oct. 1999, pp. 71–78.

Biographies

JÖRG WIDMER (widmer@pi4.informatik.uni-mannheim.de) received his M.S. degree in computer science from the University of Mannheim, Germany, in 2000. He is currently employed at the University of Mannheim while pursuing his PhD. His reserach interests include multicast communication, congestion control, and ad hoc networking.

ROBERT DENDA (denda@pi4.informatik.uni-mannheim.de) received his M.S. degree in computer science from the University of Mannheim, Germany, in 1998, and is currently pursuing his Ph.D. He is employed at the R&D department af Enditel, Endesa Ingeniería de Telecomunicaciones, Spain. His main research interests are fairness and quality of service in computer networks.

MARTIN MAUVE (mauve@pi4.informatik.uni-mannheim.de) received his M.S. and Ph.D. degrees in computer science from the University of Mannheim, Germany, in 1997 and 2000, respectively. He is currently an assistant professor at the University of Mannheim. His research interests include multimedia transport protocols, networked multimedia applications, and group communication.