Internet-Based Statistical Congestion Control for Non-Adaptable Multimedia Streams

Diplomarbeit

von

Jan Peter Damm

aus

Alheim

vorgelegt am

Lehrstuhl für Praktische Informatik IV

Prof. Dr. Effelsberg

Fakultät für Mathematik und Informatik

Universität Mannheim

April 2001

Betreuer: Dr. Martin Mauve, Dipl. Wirtsch.-Inf. Jörg Widmer

Abstract

This thesis presents a TCP-friendly congestion control scheme for non-adaptable flows. The main characteristic of these flows is that their data rate is determined by the producing application and cannot be adapted to the current congestion situation of the network. Typical examples of non-adaptable flows are those produced by networked computer games or live audio and video transmissions where adaption of the quality is not possible (e.g., since it is already at the lowest possible level). We propose to perform congestion control for non-adaptable flows by suspending them at appropriate times so that the aggregation of multiple non-adaptable flows behaves in a TCP-friendly manner. The decision whether a flow is suspended is based on random experiments. In order to allocate probabilities for these experiments, the data rate of the non-adaptable flow is compared to the rate that a TCP flow would have under the same conditions. Because random experiments are used to probabilistically adapt the aggregate rate of multiple flows, this congestion control scheme is called Probabilistic Congestion Control or PCC. We present a detailed discussion of the PCC scheme and an evaluation based on extensive simulations with the network simulator ns-2.

Acknowledgements

In its present form, this work was possible only with the support of a number of people to whom I would like to express my gratitude.

First of all, I would like to thank Prof. Dr. Effelsberg for putting his trust in me by consigning me to this challenging topic.

I would like to thank Martin Mauve and Jörg Widmer for the enthusiasm with which they continuously supported my work.

Furthermore, I would like to thank Achim Trabold for his helpful comments.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Congestion control is a vital element of computer networks such as the Internet. It has been widely discussed in literature - and experienced in reality - that the lack of appropriate congestion control mechanisms will lead to undesirable situations such as congestion collapse [1] of (parts of) a network. Due to congestion, the network capacity in these situations is almost exclusively used up by traffic that never reaches its destination, as packets are dropped by intermediate systems.

In the current Internet, congestion control is primarily performed by TCP. Furthermore, during recent years new congestion control schemes have been devised supporting networked applications that do not use TCP. Typical examples of such applications are audio and video transmissions over the Internet. One prime aim that the new congestion control schemes try to achieve is to share the available bandwidth in a fair manner with TCP-based applications, thus creating the category of *TCP-friendly* congestion control mechanisms.

TCP, as well as existing TCP-friendly congestion control algorithms, require that the data rate of an individual flow can be adapted to the congestion situation of the network. Using TCP, a fixed amount of data may take a variable amount of time to transmit, or with TCP-friendly congestion control the quality of an audio or video stream may be adapted to the available bandwidth.

While this not a limitation for a number of applications, there are cases where the data rate of an individual flow is determined by the application and cannot be adapted to the network conditions. Networked computer games are a typical example, considering the fact that players are very reluctant to accept the delayed transmission of information about a remote player's actions. Live audio and video transmissions with a fixed minimum quality below which reception is useless fall into the same category. For this class of applications there are only two acceptable states: either a flow is *on* and the sender transmits at the data rate determined by the application or it is *off* and nothing is transmitted at all. We call network flows caused by these applications *non-adaptable* flows.

This thesis presents a TCP-friendly end-to-end congestion control mechanism for non-adaptable unicast flows called *Probabilistic Congestion Control* (PCC). The main idea of PCC is to make aggregations of PCC flows rather than each single flow behave TCP-friendly at any time. This is achieved by regularly staging random experiments for each PCC flow based on probabilities calculated from the flow's actual send rate and an estimation of the current TCP-friendly rate in the network. Whenever such an experiment fails, the flow is turned off for a certain period of time.

The probabilities are chosen in such a way that the expected average send rate of each flow equals the TCP-friendly rate. We chose to use the name Probabilistic Congestion Control for this scheme because TCP-friendliness for aggregations of flows is achieved probabilistically through the use of random numbers.

The remainder of this thesis is structured as follows. Chapter 2 summarizes current approaches to congestion control. In Chapter 3 we examine non-adaptable flows in more detail. An extensive presentation of the PCC mechanism is given in Chapter 4, followed by a discussion of our protocol implementation in Chapter 5. The results of the simulation studies that were conducted are presented in Chapter 6. Chapter 7 proposes values to be used for numerous parameters. Chapter 8 discusses possible extensions to the PCC scheme, and we conclude the thesis with a summary and an outlook to future work in Chapter 9.

The main scientific contributions of this thesis are: the identification of the class of non-adaptable flows; the general idea of probabilistic congestion control for non-adaptable flows; a method to protect flows from unreliable network condition measurements at start-up in an overall fair manner; and continuous evaluation as a way to quickly adapt the aggregation of non-adaptable flows to degradations in network quality, while basing probability calculations on longer off times.

Chapter 2

Current Mechanisms

In the Internet, the Transmission Control Protocol (TCP) has proven to provide good congestion control. However, TCP is not suited for all types of network traffic. For example, TCP's re-transmission mechanism is too time-consuming for applications that need to obey real-time constraints, and acknowledgments and re-transmissions do not scale well for multicast flows. This section gives a short introduction to TCP and defines the term *TCP-friendliness* before briefly describing some major alternatives to TCP congestion control.

2.1 TCP

TCP is a connection-oriented unicast protocol that offers reliable data transfer [2]. TCP achieves reliability by employing mechanisms such as positive ACKs with re-transmission, sequence numbers, and a checksum.

In addition, TCP provides flow control and congestion control using a sliding window mechanism. The size of the window used for flow control is advertised by the TCP receiver, while the congestion window size regularly changes depending on indications for congestion. The minimum of the two window sizes determines the number of packets that may be sent before some of these packets are acknowledged by the receiver, and it thereby determines the send rate.

TCP rate change in response to congestion indications is described in [2] for the case that the advertised window does not pose a practical limitation: "On start-up, TCP performs *slowstart* during which the rate roughly doubles each round-trip time to quickly reach its fair share of bandwidth. In steady state, TCP uses an additive increase, multiplicative decrease mechanism (AIMD) to detect additional bandwidth: when there is no indication for loss, TCP increases the congestion window by one segment per round-trip time. When TCP experiences packet loss indicated by a timeout, the congestion window is reduced to one segment and TCP reenters the slowstart phase. Packet loss indicated by three duplicate ACKs results in a window reduction to half its previous size."

This behavior lets TCP flows adapt quickly to a worsening of network conditions, but also quickly search out free bandwidth. It also results in another property of TCP, which is the typical saw-tooth behavior with large variations of the send rate.

2.2 TCP-Friendliness

In [1], non-TCP flows are defined as being TCP-friendly when "their long-term throughput does not exceed the throughput of a conformant TCP connection under the same conditions".

Widmer et al. give a slightly different definition of *TCP-friendliness* (for unicast flows) in [2]:

> 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 do under the same network conditions.

For the purposes of PCC, where the goal is to make an accumulation of flows TCP-friendly rather than each single flow, we relax this definition to the following:

> An accumulation of unicast flows is considered *TCP-friendly* when it does not reduce the long-term throughput of any coexistent TCP flow more than an equal number of TCP flows on the same path would do under the same network conditions.

Due to the law of large numbers, this can be achieved if each flow's expected average rate is such that a flow sending constantly at this rate would be TCP-friendly in the sense of [2].

2.3 Classification of TCP-Friendly Congestion Control Schemes

The authors of [2] classify congestion control schemes by the following criteria:

- **Window-Based versus Rate-Based:** Window-based congestion control schemes use a congestion window to adapt their offered network load in much the same way as TCP does. Rate-based schemes employ some kind of feedback mechanism to measure network conditions, then adjust the transmission rate based on these measurements. A specific class of the latter type are model-based approaches, which use a TCP throughput model to determine a TCP-friendly send rate.

- **Unicast versus Multicast:** While TCP-friendliness is desirable for both unicast and multicast traffic, it is far more difficult to achieve for the multicast case. Multicast congestion control schemes ideally should scale to large receiver sets and be able to cope with heterogeneous network conditions at the receivers. For sender-based multicast congestion control, the *loss path multiplicity problem* [3] needs to be considered: with a large receiver set, it is likely that each transmitted packet is lost for at least one receiver. Packet loss reports can therefore not be used as congestion indications in the same way as in the unicast case.

- **End-to-End versus Router-Supported:** Many of the TCP-friendly schemes proposed are designed for best-effort IP networks such as today's Internet that do not provide any additional router mechanisms to support the protocols. However, the design of congestion control protocols and particularly fair sharing of resources can be considerably facilitated by placing intelligence in the network.

- **Sender-Based, Receiver-Based, or Router-Based Congestion Control:** The responsibility of enforcing congestion control can be located at the sender, at the receiver(s) or at intermediate routers. A common approach is that the sender uses information about network congestion to adjust the rate or window size in order to achieve TCP-friendliness, while receivers only give feedback. In the converse case, responsibility for rate adjustment lies at the receiver, which signals the sender as to what rate it is allowed to send at. The best fairness can be achieved with the help of router assistance (as detailed in [1]), but such mechanisms are difficult to deploy as changes to the Internet infrastructure take time and are very costly.

- **Single-Rate versus Multi-Rate:** This criterion is specific to multicast traffic, where a sender can choose to supply all receivers with the same amount of data or use multiple rates for different senders. In the unicast case with only one receiver per sender, all traffic is single-rate.

2.4 TEAR

TCP Emulation At Receivers (TEAR) [4] is a hybrid end-to-end protocol that combines aspects of window-based and rate-based congestion control. TEAR 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 acknowledgments, timeout events can only roughly be estimated. In contrast to TCP, the TEAR protocol does not directly use the congestion window to determine the amount of data to send but calculates the corresponding TCP send rate. This rate is roughly a congestion window worth of data per round-trip time. To avoid TCP's saw-tooth behavior, TEAR averages the rate over an *epoch*, where an epoch is defined as the time between consecutive rate reduction events. To further prevent 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 accordingly adjusts the send rate.

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 round-trip time 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 it avoids TCP's frequent rate changes.

2.5 TFRC

The TCP-Friendly Rate Control Protocol (TFRC) [5] evolved from the TFRCP protocol [6]. This rate-based scheme is specified for unicast communication although it can be adjusted to multicast with some modifications. TFRC adjusts its send rate to the complex TCP equation given in [7]

(Equation 5.4). The authors chose the Average Loss Interval method as loss rate estimator: 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 round-trip time is measured with the standard method of feeding back time-stamps to the sender.

The receiver sends state reports to the sender once per round-trip time. The sender then computes a new fair rate from the parameters and adjusts the send rate accordingly. Immediately after start-up, 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. This sender-based approach can also be transformed into a receiver-based approach.

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

2.6 Other Rate-Adapting Mechanisms

Apart from the mechanisms described in the two previous sections, work has been done on a number of other non-TCP mechanisms that attempt to adapt the rate of singular flows in a TCP-friendly manner. Among these are PGMCC [8], FLID-DL [9], and Rainbow [10]. For an overview of these and other congestion control mechanisms, see [2].

2.7 Reservation

An interesting approach to congestion control is reservation of bandwidth. The idea is that each flow needs to reserve the bandwidth it requires before being allowed to transmit any data. If sufficient resources are not available, the reservation request is denied and the flow is not allowed to transmit. If a pessimistic reservation mechanism is used that accepts requests only up to the point where the sum of all requested (maximum) rates equals the available bandwidth, congestion cannot occur.

However, pessimistic reservation has the disadvantage that bandwidth that is reserved but currently not used (by a variable bit rate flow that has reserved enough bandwidth to be able to transmit at peaks) is effectively wasted. Therefore, optimistic reservation mechanisms are often employed which accept more reservation requests to such an extent that the available bandwidth will not be exceeded *in most cases*. Here, however, a total absence of congestion can not be guaranteed.

One of the problems involved in the use of a reservation scheme is that each flow needs to know the peak rate at which it is going to send. Since this can not be determined for many applications, most flows will tend to reserve more than is actually needed. A more grave problem concerning the applicability of such a scheme to the Internet is that router support is imperative, since the end systems could not determine the global availability of free bandwidth locally in a scalable way. While ideas for such router support exist, such as Intserv/RSVP [11] or Diffserv [12], such mechanisms are not yet widely employed in the Internet.

Another disadvantage of a reservation scheme is that it treats flows unfairly in that some flows can transmit at exactly the rate they want, while others may not transmit any data at all. While this may be desirable for some classes of flows, where neither rate reduction nor transmission interruptions would be acceptable, it is less desirable for other classes.

Chapter 3

Non-Adaptable Flows

For the remainder of this thesis a non-adaptable flow is defined to be a data flow with a sending rate that is determined by the application and cannot be adapted to the level of congestion in the network. A non-adaptable flow has exactly two acceptable states: either it is in the state *on*, carrying data at the rate determined by the application, or it is *off*, meaning that no data is transmitted at all. Any data rate in between those two is inefficient, since the application is not able to make use of the offered rate.

Examples of applications producing non-adaptable flows are commercial networked computer games such as Diablo II, Quake III, Ultima Online, and Everquest. These games typically employ a client-server architecture. The data rate of a flow between client and server is determined by the fact that the actions of the players must be transmitted instantaneously. Similar restrictions hold for the flows between participants of distributed virtual environments without a centralized server. If a congestion control scheme delays actions by a considerable amount of time the application quickly becomes unusable. This can easily be experienced by experimenting with a state-of-the-art TCP-based networked computer game during peak hours. In reaction to this problem, a number of applications resort to UDP and avoid congestion control altogether.

A situation with either no congestion control at all or vastly reduced usability in the face of moderate congestion is not desirable. A much preferable approach is to turn the flows of some participants off *and* to inform the application accordingly, such that those participants are not unduly penalized (e.g., in the case of computer games players cannot be looted, killed, etc. during the off time). All other participants are not affected. On average, all users should be able to participate in the session for a reasonable amount of time between off periods to ensure usability of the application. At the same time, off periods should be distributed fairly among all participants.

Other examples of applications with non-adaptable flows are audio or video transmissions with a fixed quality. There are two main reasons why it may not be possible to scale down a media flow: either the user does not accept a lower quality, or the quality is already at the lowest possible level. The second reason indicates that a congestion control mechanism for non-adaptable flows can complement congestion control schemes that adapt the rate of a flow to the current network conditions.

Most existing congestion control mechanisms, some of which were described in the last chapter,

rely on adapting the send rate of the flow in question in some way. Either a congestion window is used, as for TCP, or the rate is adapted to a TCP-friendly value in some other way, such as using a TCP throughput model. By definition of non-adaptable flows, such mechanisms are not suitable for these flows.

One mechanism that could be used with non-adaptable flows is reservation. As long as flows are able to determine an upper bound to the required transmission rate at the time of reservation, they can send at the rate determined by the application, provided there is enough free bandwidth. However, with reservation, off periods may be distributed extremely unfairly, in that a flow is only off for the time it needs to get its reservation request through, and can then continue to send for as long as it chooses. The long wait times that may result from this may not be acceptable for most of the applications mentioned above. The most important practical drawback of reservation is that in today's Internet, the router support required for this mechanism is not in wide usage.

Chapter 4

PCC Protocol Overview

A congestion control scheme for non-adaptable flows cannot, by the very nature of these flows, involve adapting their rate. A reservation scheme, on the other hand, would require global knowledge and treat some flows unfairly.

The goal of our efforts was therefore to create a TCP-friendly congestion control scheme that is applicable to non-adaptable flows while relying solely on local knowledge and exhibiting good inter- and intra-protocol fairness.

In order to achieve this goal, the Probabilistic Congestion Control scheme (PCC) provides congestion control not by reservation or by adapting the rate at which any single flow is sending, but by turning flows on and off at appropriate times.

PCC is a rate-based, end-to-end congestion control scheme and does not require the support of routers or other intermediate systems in the network. While the current implementation of PCC is strictly receiver-based and for unicast use only, a sender-based implementation or an extension for multicast use are also possible. We call a flow that uses PCC a PCC flow. Note that the only extension PCC offers to an underlying protocol is congestion control. In particular, PCC does not offer any reliability mechanisms.

They key idea of PCC is that - as long as there is a high level of statistical multiplexing - it is not important that each single non-adaptable flow behaves TCP-friendly at any specific point in time. What is important is that the aggregation of all non-adaptable flows on a given link behaves as if the flows were TCP-friendly. More to the point, the sum of all traffic generated by PCC flows at a given point in time should be approximately equal to the traffic the same number of TCP flows would generate under the same circumstances. Due to the law of large numbers this can be achieved if (a) each link is traversed by a sufficiently large number of independent PCC flows and if (b) each PCC flow has an expected average rate which is TCP-friendly.

At a first glance (a) may be considered problematic because it is possible that a link is traversed only by a small number of PCC flows. However, further reflection reveals that in this case the PCC flows will only be significant in terms of network congestion if individual PCC flows occupy a high percentage of the link's bandwidth. We therefore relax (a) to the following condition (c): a single PCC flow is expected to have a rate that is only a small fraction of the available bandwidth on any

link that it crosses. Given the current development of available bandwidth in computer networks this is a condition that is likely to hold true.

In the following section we will define the requirements that need to be fulfilled in order for PCC to be applicable. The first one includes condition (c).

4.1 Requirements

There are a number of requirements that have to be fulfilled in order for PCC to be applicable:

R1: High level of statistical multiplexing. As it is usual for congestion control schemes, PCC requires a high level of statistical multiplexing. Moreover, in order to satisfy condition (c) from above, a single PCC flow is expected to have a rate that is only a small fraction of the available bandwidth on any link that it crosses.

R2: User acceptance of congestion control. Users typically perceive congestion control for adaptable flows as something bad and annoying, since it reduces the bandwidth that is available to them. The same holds true for non-adaptable flows, even more so, since the only way to react to congestion is to turn some flows off for some time. For the overall acceptance of PCC it is therefore important that the reason for this behavior is explained to the user in an appropriate way.

R3: No synchronization of PCC flows at start-up. PCC flows should be started independent of each other, no significant number of PCC flows may be synchronized at start-up time.

R4: The average rate of a PCC flow can be predicted. In order for PCC to work, it must be possible to predict the average rate of a PCC flow with reasonable accuracy. There are multiple ways how this can be done, ranging from using a constant bit rate flow where this prediction is trivial to the use of application level knowledge or prediction based on past bandwidth samples.

R5: The average rate of a TCP flow under the same conditions can be estimated. We expect that there is a reasonably accurate method to estimate the average rate at which a TCP flow would send under the same network conditions. This includes mechanisms to measure current network conditions, such as the loss event rate [1] and the round-trip time, as well as a method to estimate TCP's average rate under these conditions. However, it is important to notice that PCC is independent of the method used to estimate the throughput of a TCP flow for a given network situation.

The reasons for these requirements should become clear in the remainder of this chapter. Meanwhile, for the purpose of defining the PCC scheme, we assume that the requirements hold.

[1] The loss event rate is defined as the number of round-trip times in which at least one packet loss occurred divided by the total number of packets sent in the same period. This is identical to the definition of the loss rate but for the fact that multiple losses in one round-trip time are treated as a single loss. In this way the loss event rate behaves like the congestion indication mechanism of TCP, where loss events (one or more packet losses in one round-trip time) rather than losses are used to indicate congestion.

4.2 Turning Flows Off

The initialization process of a PCC flow is very simple: the sender starts to send packets at the rate determined by the application. The receiver then answers periodically with feedback control packets, containing whatever information the methods used to estimate send rate and TCP-friendly rate need. Periodic feedback takes place whenever the sender is transmitting.

In the receiver-based case, the control packets also contain the information whether the sender may remain on or whether it is expected to turn itself off. In the sender-based case, this information is generated by the sender and does not need to be transmitted.

The decision on whether to turn the flow off is made periodically. To this end, a random experiment is performed every T_{EXP} seconds. If the result of the experiment is positive, the flow may remain on up to at least the next experiment, otherwise it needs to be turned off for a period of T_{OFF} seconds. After this period, the sender may again start transmitting packets. Both T_{EXP} and T_{OFF} are parameters supplied by the application[2].

The random experiment is performed by calculating a probability value p_{ON} in [0;1] at which the flow is allowed to remain on, then generating a random number RAND in (0;1] and comparing these values. If $p_{ON} \leq$ RAND, the experiment fails and the flow needs to be turned off. The half open interval (0;1] for RAND is used so that with p_{ON} = 0, the flow will always be turned off while with p_{ON} = 1, it will always remain on. (Obviously therefore, if p_{ON} is not in (0;1), the generation of a random number is not necessary.)

In order to motivate our calculation of p_{ON}, we will first assume the following conditions (the ideal case):

C1: When a flow starts or is restarted after an off period, a good estimate for the TCP-friendly rate r_{TCP} is known instantly.

C2: An experiment period T_{EXP} equal to T_{OFF} is used.

In the following three sections, we will first explore the ideal case, then see how our calculation of p_{ON} is affected when the above conditions do not hold.

4.3 Experiments in the Ideal Case

As stated before, the goal of the PCC scheme is to ensure that the sum of all traffic generated by a number of PCC flows is (approximately) equal to the traffic the same number of TCP flows would generate under the same network circumstances.

This can be achieved when the average expected rate at which PCC flows send is equal to the average rate a TCP flow would send at under the same circumstances, i.e. the TCP-friendly rate r_{TCP}. As

[2]User acceptance (requirement R2) is somewhat facilitated by the fact that off times are always known in advance. Off time is either directly determined by the application parameter T_{OFF}, or it is adjusted prior to turning the flow off, as will be explained later on. In either case, the time until the flow is turned on again can be made known to the user, e.g., in the form of a count-down timer.

p_{ON} is the probability that the flow may remain on up to the next experiment (i.e. within the next T_{OFF}), this can be expressed as

$$p_{ON} \cdot T_{OFF} \cdot r_{NA} = T_{OFF} \cdot r_{TCP} \tag{4.1}$$

$$\Leftrightarrow p_{ON} \cdot r_{NA} = r_{TCP} \tag{4.2}$$

where r_{NA} is the actual (average) send rate at which the average non-adaptable PCC flow is sending when on.

Equation 4.2 gives the following value for p_{ON}:

$$p_{ON} = \frac{r_{TCP}}{r_{NA}} \tag{4.3}$$

This equation for p_{ON} exhibits some noteworthy properties:

P1: For $r_{NA} > r_{TCP}$, the probability for the flow to remain on is < 1.

P2: For $r_{NA} = r_{TCP}$, p_{ON} is exactly 1.

P3: For $r_{NA} < r_{TCP}$, p_{ON} is > 1.

P4: p_{ON} is always ≥ 0, as r_{NA} and r_{TCP} are always positive or zero (there is no such thing as a "negative transmission rate" in our context) [3].

Obviously, P1 is a property that needs to hold in all cases. If a flow is sending at a rate higher than the TCP-friendly rate, there must be some probability $p_{OFF} = 1 - p_{ON}$ that it will stop.

In the ideal case, the applicability of property P2 is also apparent. When a flow is sending at exactly the TCP-friendly rate, it should not be required to stop. We will see that this will be somewhat different in the non-ideal case.

As we see in property P3, Equation 4.3 does not always provide a value for p_{ON} in the range of [0;1]. However, we can treat values greater than one as one, because a probability value greater than one can mean nothing else than that the flow should not be required to stop, and this is already the case for $p_{ON} = 1$. When a flow transmits at less than the TCP-friendly rate, it can by Equation 4.3 transmit all the time. That is already the optimum.

P4 is a property that makes things easy in the ideal case. We will see that this property also is not ensured in the non-ideal case.

Up to this point we have been using average values for r_{NA} and r_{TCP}, thus global knowledge. Some thought reveals however that it is possible to use only local knowledge instead.

4.3.1 The Use of Local Knowledge

As stated in requirement R5, we expect a reasonably accurate method to estimate r_{TCP}. This method will be based on (explicit or implicit) measurements of network conditions such as the loss event rate and the round-trip time. When these measurements are based on local knowledge, r_{TCP}

[3] p_{ON} is never calculated when r_{NA} is zero.

will be over-estimated in some cases, while it will be under-estimated to the same extent in others. Ultimately, this means that some PCC flows will be more likely to stop than they should be, while others will be less likely to do so. Due to the high level of multiplexing (R1) and the law of large numbers, the average rate of a PCC flow will closely approach the value that could be obtained with the use of global knowledge.

The other part of the equation that is affected by the use of local knowledge only is the PCC flow's rate r_{NA}. While r_{NA} may vary greatly between different flows carrying different types of non-adaptable media, this has no effect on the average expected rate of each flow, which is $p_{ON} \cdot r_{NA} = r_{TCP}$ for $r_{NA} \geq r_{TCP}$, and thus on the average amount of bandwidth taken up by the aggregation of all PCC flows.

Using local values for r_{NA} and the estimate of r_{TCP}, Equation 4.2 can be restated as

$$\sum_{i=1}^{n} p_{ON}^{i} \cdot r_{NA}^{i} = \sum_{i=1}^{n} r_{TCP}^{i} \tag{4.4}$$

where n is the number of PCC flows.

Obviously, Equation 4.4 holds for $p_{ON}^{i} = \frac{r_{TCP}^{i}}{r_{NA}^{i}}$.

This shows that global knowledge about network conditions (and thus about the average expected TCP-friendly rate) is not necessary, though it would be useful if available. It is important to note however, that global averages of the PCC flows' send rate and their off and protected times should not be used even if they were available. The result of using these global values would be a globally uniform probability value p_{ON}. While this, too, would ensure that n PCC flows do not use up more than n times the TCP-friendly rate, this approach has important drawbacks:

- Using global averages would offset the less-than-TCP-friendly rate of some PCC flows against too high rates by others. When in a network with n PCC flows and a number of TCP flows, $\frac{n}{2}$ PCC flows choose to send very little for a while, the other $\frac{n}{2}$ PCC flows would be allowed to send at almost twice the TCP-friendly rate. The bandwidth freed by a self-limited TCP flow would however (through lower loss event rate and shorter round-trip time) benefit PCC flows as well as TCP flows, because the TCP congestion control mechanism is based on local knowledge only. When attempting to achieve TCP-like overall behavior, the bandwidth freed by a flow sending at less than TCP-friendly rate should benefit all flows in the network to the same extent. This is achieved by the sole use of local knowledge.

- Using global values would result in bad intra-protocol fairness: If all flows had the same probability to stop at each experiment, flows sending at a lower rate while on would also have a low average rate, while high-bandwidth flows could send at a higher than TCP-friendly average rate. The penalty for sending at a high rate would not be borne by the high-bandwidth flow alone, but shared across all PCC flows. Using local knowledge only on the other hand results in each flow's average expected rate being adjusted to the TCP-friendly rate.

4.4 Experiments after Protected Time

4.4.1 Protected Time

Condition C1 stating that a good estimate of r_{TCP} is instantly available is of course not realistic in all cases.

The current loss event rate and round-trip time are two network conditions that will always need to be measured - explicitly or implicitly - for any sensible attempt to estimate r_{TCP}. This is because TCP itself relies heavily on the (implicit) measurement of these conditions.

Naturally, the loss event rate can only be measured when the flow is transmitting. When an experiment is staged after the flow has already been transmitting for a sufficient time, the ongoing measurement of network conditions will be able to instantly provide PCC with a good estimate. When however the flow is only just starting or restarting after a period of being off, some time is necessary to reliably take measurements and arrive at a good estimate for r_{TCP}. It is not acceptable at this point to base the estimate solely on past measurements taken before the flow was turned off, as network conditions my have changed drastically in the meantime.

When measuring a network's current loss event rate ler, an average time of $\frac{1}{2 \cdot ler \cdot r_{NA}}$ seconds passes before the first loss event occurs and the first value for the measurement can be taken. One value is however not enough to generate a reliable estimate, so a multiple of that time is necessary. The average time that passes between two consecutive loss events is $\frac{1}{ler \cdot r_{NA}}$.

For a reliable measurement of the round-trip time RTT, some transmission of data and/or control packets is also necessary. This usually takes less time than an estimate for ler.

During the time it takes to arrive at a good estimate for r_{TCP}, a random experiment to calculate p_{ON} would be based on bad measurement values. Furthermore, if these measurements provoke a strong under-estimation of r_{TCP}, the experiment may well result in a PCC flow stopping that would not be required to stop in an experiment using a good estimate. This problem is not fully offset by potential over-estimations, because these will often result in $p_{ON} > 1$, which is treated as $p_{ON} = 1$, as explained above. Performing experiments with such bad measurement values would thus result not only in unnecessary variation, but also in actual under-allocation of PCC bandwidth.

To obviate this problem, it is necessary to protect the PCC flow from being required to stop for the time it takes to arrive at a good estimate. We call this time interval protected time (T_{PROT}). The duration of T_{PROT} can vary, especially with the current loss event rate. We propose to protect the (re-)starting PCC flow for a fixed number of loss events as well as for a fixed number of round-trip times. Protected time T_{PROT} will thus be as long as the maximum of the time T_{LER} it takes to get a reliable measurement for the loss event rate and the time T_{RTT} necessary for a good estimate of the round-trip time:

$$T_{PROT} = \max(T_{LER}, T_{RTT}) \tag{4.5}$$

With network conditions typically experienced in today's Internet, T_{PROT} can be expected to be in the range of a few seconds to half a minute. For some sample calculations of T_{PROT}, see Appendix A.

It may be considered problematic to let a flow send instantly with its full rate for T_{PROT} seconds, since this essentially violates the idea of exploring the available bandwidth as it is done, e.g., by TCP slowstart. However, requirements R1 (high level of statistical multiplexing) and R3 (no synchronization at start up) prevent that this causes excessive congestion. In addition the value of T_{PROT} will usually get smaller the more congested the network is, since loss events become more frequent and therefore the estimate converges more quickly.

Also during protected time, the average rate of the non-adaptable flow r_{NA} is calculated, e.g., using exponentially moving weighted averages, unless this rate is otherwise determined. Another possibility to determine r_{NA} is the use of application-level knowledge. In the trivial case of a constant bit rate flow, no further action is necessary.

4.4.2 Making Up for Protected Time

If the same formula as in the ideal case were used to perform the random experiment after protected time, the amount of data the flow already transmitted during T_{PROT} would not be accounted for, leading to over-allocation of PCC bandwidth.

To account for that data, a different formula is used that meets the following requirement: The expected average of the amount of data transmitted by any PCC flow in protected time and in the T_{OFF} interval afterwards (which is the interval this formula decides about) should be the same as the amount of data transmitted by a TCP flow within the same time interval of $T_{PROT} + T_{OFF}$:

$$T_{PROT} \cdot r_{NA} + p_{ON} \cdot T_{OFF} \cdot r_{NA} = (T_{PROT} + T_{OFF}) \cdot r_{TCP} \tag{4.6}$$

This results in a different formula for p_{ON} to be used in the first experiment after protected time:

$$p_{ON} = \frac{(T_{PROT} + T_{OFF}) \cdot r_{TCP} - T_{PROT} \cdot r_{NA}}{T_{OFF} \cdot r_{NA}} \tag{4.7}$$

Note that in analogy to Equation 4.4 for the ideal case, we can use local values for all elements of the formula.

With Equation 4.7, the average expected rate of a flow in the time spanning T_{PROT} and the first T_{OFF} after that can be set to the TCP-friendly rate. With Equation 4.3, the same can be done for any later interval T_{OFF}. As long as a smooth transition from the non-ideal phase to the ideal phase can be assured (how this is achieved will be explained in a later section), this means that the average expected rate of a flow within the time from the beginning of protected time up to the beginning of the next protected time after the flow has been turned off is TCP-friendly. It then follows that the flow's average expected rate is TCP-friendly over the complete time it is active.

4.4.3 Handling Negative Protected Times

After solving Equation 4.7, p_{ON} can be in any one of the three following ranges:

- $p_{ON} \geq 1$: The non-adaptable flow consumes less or equal bandwidth than a TCP flow would do under the same conditions. The flow should therefore stay on.

- $0 \leq p_{ON} < 1$: The non-adaptable flow consumes more bandwidth than a TCP flow would do under the same conditions. The flow should be turned off with the probability $p_{OFF} = 1 - p_{ON}$.

- $p_{ON} < 0$: The non-adaptable flow transmitted more data during T_{PROT} than a TCP flow would carry during $T_{PROT} + T_{OFF}$.

The third range is specific to the non-ideal case. As noted above, p_{ON} cannot be negative in the ideal case. When however p_{ON} is calculated to make up for the bandwidth used by the PCC flow during protected time, p_{ON} can be negative if during T_{PROT}, the non-adaptable flow has already transmitted more than a TCP flow would do during $T_{PROT} + T_{OFF}$.

For the purpose of the random experiment, p_{ON} can only be treated as 0 (there is no negative probability). However, the extra bandwidth the PCC flow used up still remains to be made up for. Since the data transmission of the PCC flow during T_{PROT} cannot be undone, we need to find some other way to do this.

The idea is to extend the time T_{OFF} the flow is required to stop to such an extent that re-calculating 4.7 with the resulting extended off time $T_{OFF,EXT}$ gives the value $p_{ON} = 0$, which is the one that is effectively used in the random experiment:

$$0 = \frac{(T_{PROT} + T_{OFF,EXT}) \cdot r_{TCP} - T_{PROT} \cdot r_{NA}}{T_{OFF,EXT} \cdot r_{NA}} \tag{4.8}$$

or

$$T_{OFF,EXT} = \frac{T_{PROT} \cdot (r_{NA} - r_{TCP})}{r_{TCP}} \tag{4.9}$$

Note that this is the same as to say the off time is extended so that during protected time plus off time, a TCP flow can transmit as much data as the PCC flow transmitted during T_{PROT}. This can be easily shown by restating 4.9 as follows:

$$(T_{OFF,EXT} + T_{PROT}) \cdot r_{TCP} = T_{PROT} \cdot r_{NA} \tag{4.10}$$

While there is the obvious need to adjust T_{OFF} in the case of $p_{ON} < 0$, such an adjustment is not necessary, or indeed possible in the opposite case of $p_{ON} > 1$. In that case, the flow does not stop, therefore an adjustment of T_{OFF} could only take effect the next time the flow is required to stop. At that point however, network conditions have changed, p_{ON} is again < 1, and the original value for T_{OFF} needs to be used.

It can be seen from Equation 4.10 that with $p_{ON} < 0$, the main requirement for PCC, that the expected average amount of data transmitted by a PCC flow be equal to that of a TCP flow, holds for time period $T_{PROT} + T_{OFF,EXT}$.

After the calculation of $T_{OFF,EXT}$, T_{OFF} can be adjusted to that value either temporarily or permanently:

Temporary Adjustment of T_{OFF}

Some applications using PCC may rely on being able to transmit at least some data after the preset off time of T_{OFF} in most cases. For these applications, a permanent adjustment of T_{OFF} to the new value is not acceptable. Thus, to make up for the unfair amount of data the flow transmitted in protected time, the off time immediately following the experiment that resulted in $p_{ON} < 0$ is set to $T_{OFF,EXT}$, while all further calculations are again based on T_{OFF}.

This assumes that T_{OFF} only needs to be extended in rare exceptional cases. With 4.7 (also compare 4.9), this means that $\frac{T_{OFF}}{T_{PROT}}$ should be greater or equal to $\frac{r_{NA}}{r_{TCP}} - 1$ in all but a few cases. If this is not the case, the application will need to decide whether it will permanently adjust T_{OFF} itself, accept the extended off times when they occur, or, if neither is feasible, stop sending altogether.

Permanent Adjustment of T_{OFF}

Some applications on the other hand may not mind a permanent extension of T_{OFF}, but may expect that parameter to remain as constant as possible. For this type of application, T_{OFF} should be adjusted to the new value of $T_{OFF,EXT}$, and this value should be used in further calculations, e.g. of p_{ON}. If network conditions do not worsen after that, p_{ON} can be expected to always be ≥ 0.

After permanently adjusting T_{OFF} it is however advisable to monitor whether p_{ON} would be < 0 again if the original value were used, and if this is not the case for a sufficient time, to revert back to the original value. This avoids having short and rare periods of high congestion permanently change T_{OFF} from its preset, and thus desirable, setting.

It should be noted here that using different values for T_{OFF} across different PCC flows does not harm overall fairness, since every flow calculates a probability value appropriate to its off time.

4.4.4 Maximum Value for Protected Time

It can be seen from Equation 4.9 that without a limitation on T_{PROT}, $T_{OFF,EXT}$ can also become very large. This may not be desirable for some applications that expect to be able to transmit at least some data at more or less regular intervals. Also, it has been noted that for applications adjusting T_{OFF} only temporarily on $p_{ON} < 0$, $\frac{T_{OFF}}{T_{PROT}}$ should in most cases be $\geq \frac{r_{NA}}{R_{TCP}} - 1$. Finally, large values for T_{PROT} may not be desirable in terms of fairness as they make PCC less robust against some amount of synchronization at start-up (R3).

Therefore, a limiting parameter $T_{PROT,MAX}$ may be be necessary that cuts off protected time even if the preset number of loss events and round-trip times has not yet occurred. While this limit can help to alleviate the above effects and allows the use of smaller values for T_{OFF}, these positive effects come at the cost of higher variance in the estimation of r_{TCP}, since there are fewer measurements of the network condition available. As has been noted in Section 4.4.1, a higher variance can also lead to under-allocation of PCC bandwidth. Therefore, $T_{PROT,MAX}$ should be chosen large enough that in most cases, enough measurements are taken to arrive at an acceptable estimate of r_{TCP}.

The minimum value for $T_{PROT,MAX}$ should be the number of round-trip times necessary to get a meaningful measurement for that value. This is not normally a strong limitation on any application (see Appendix A). However, there may well be the case that in protected time there occur no loss events at all up to the end of $T_{PROT,MAX}$. This should be handled by setting r_{TCP} to infinity, which results in $p_{ON} > 1$, thus the flow can stay on. This method seems reasonable if the value for $T_{PROT,MAX}$ has been chosen large enough to provide PCC with meaningful values in most cases, since in that case, no loss event during $T_{PROT,MAX}$ means that loss event rate is indeed low. Also, if $T_{PROT,MAX}$ is chosen very small, the flow will mostly avoid experiments using the formula for the non-ideal case, but the the amount of data transmitted in $T_{PROT,MAX}$ that this formula intends to make up for is not very relevant. In addition, this amount of data is offset by the chance that there is a loss event in that time period, a case that would lead to a strong over-estimation of the loss event rate, which usually leads to $p_{ON} < 1$.

4.5 Continuous Evaluation

If we could assume that the level of congestion stayed more or less constant, then the further behavior of PCC would be fairly simple. After T_{OFF} seconds a flow that is in the on state would have to repeat the estimation and random experiment step. This time the parameters r_{NA}, T_{PROT}, and r_{TCP} can be monitored during the previous T_{OFF} seconds, so we instantly have good measurement values at the time of the experiment, and there is no need for a protected time period ($T_{PROT} = 0$). This is the ideal case, so Equation 4.3 can be used to calculate p_{ON}.

Unfortunately, in a real network the level of congestion is not constant and may change significantly within a time frame much smaller than T_{OFF}. There are two cases to look at: the congestion situation may improve (i.e. the rate that a TCP flow would have under the same conditions increases) or the congestion situation may get worse.

The first case is not problematic since it does not endanger the overall performance of the network. In this situation, PCC flows have been treated unfairly in that they have been turned off with a higher probability than they should have been (in retrospect). However, after T_{OFF} they will be turned on again and react to the new congestion situation (i.e. the likeliness that they will be turned off again adapts to the new situation).

The second case is much more dangerous to the network. In order to prevent an unfair treatment of competing TCP flows or even a congestion collapse, it is very important that PCC flows react fast if the congestion situation becomes worse. Condition C2 (Section 4.2), that $T_{EXP} = T_{OFF}$ can no longer be assumed. Instead, a smaller T_{EXP} is used and PCC continuously calculates the value for p_{ON} every T_{EXP}, using Equation 4.7 if the flow is still in the first T_{OFF} after the initialization or Equation 4.3 if it is not. If $p_{ON} < 1$, the random experiment is continued in the same way as described above.

Obviously, it would not acceptable to simply re-calculate p_{ON} without accounting for the fact that the flow could have been turned off during another experiment in the previous T_{OFF}. This would treat PCC flows in an unfair manner compared to TCP flows: for a value of p_{ON} smaller than one PCC would continue to perform the same random experiment again and again, even if the congestion situation stayed constant. The probability of a flow being turned off within a time-frame of T_{OFF} would increase from $1 - p_{ON}$ to $1 - p_{ON}^{e}$, with e being the number of experiments performed in

T_{OFF}, $\frac{T_{OFF}}{T_{EXP}}$. Remember that the calculation of p_{ON} is based on the time period T_{OFF}.

In order to solve this problem, PCC modifies the value r_{NA} by taking into account all random experiments that have been performed for the flow during the last T_{OFF}. The general idea is that the modified value should represent the current expected average data rate of the flow. For this purpose PCC maintains a set P of the probabilities $p_{ON}^i \in (0;1]$ with which the flow stayed on during the previous $\frac{T_{OFF}}{T_{EXP}}$ random experiments. The rate r_{NA} of the flow is then adjusted based on this information. This adjusted rate is called effective rate (r_{EFF}). It is determined according to Equation 4.11. If P is empty then r_{EFF} is set to r_{NA}. For the continuous evaluation and random experiments r_{EFF} replaces r_{NA} in Equation 4.3.

$$r_{EFF} = \begin{cases} r_{NA} \cdot \prod_{p_{ON}^i \in P} p_i & \text{for } P \neq \emptyset \\ r_{NA} & \text{for } P = \emptyset \end{cases} \tag{4.11}$$

In Equation 4.7 for the non-ideal case, r_{EFF} is not used to replace all appearances of the variable r_{NA}. Instead, only the variable describing the data rate during T_{OFF}, which appears in the denominator of 4.7 is adjusted. In the numerator, the original value for r_{NA} is used, since this describes the data rate during T_{PROT}, which is completely unaffected by any experiments. Thus, Equation 4.7 is modified to Equation 4.12:

$$p_{ON} = \frac{(T_{PROT} + T_{OFF}) \cdot r_{TCP} - T_{PROT} \cdot r_{NA}}{T_{OFF} \cdot r_{EFF}} \tag{4.12}$$

A value for T_{EXP} should be chosen by dividing T_{OFF} by an integer. It is also possible to handle values where T_{OFF} is not a multiple of T_{EXP} in this context, but this would introduce additional mathematical complexity that is usually not necessary.

Since r_{EFF} is the expected average rate of the PCC flow, and the formulas used to calculate p_{ON} are designed to accomplish an expected average rate equal to the TCP-friendly rate (or slightly less, taking into account the protected time, in the case of Equation 4.7), p_{ON} will be exactly one in all further experiments until some values "drop out of" P, if network conditions and the flow's rate do not change. If conditions improve, p_{ON} will be greater than one, if they worsen, it will be less than one. Only in the last case will a random value need to be generated and the flow possibly be turned off.

By this method, the probability p_{ON} can be calculated based on the off period T_{OFF}, while being re-calculated at a much higher frequency. The first is necessary because T_{OFF} is the smallest granularity of time a decision can be made about, and the second is essential for a good reaction of PCC flows to increased congestion.

Some additional work is required to ensure a smooth transition from Equation 4.12 used in the first T_{OFF} after protected time to Equation 4.3 used subsequently. If network conditions remain stable, every further experiment should result in $p_{ON} = 1$. This should also be the case if some experiment in the continuous evaluation within the first T_{OFF} interval results in a probability value in $(0;1)$. This value is calculated from Equation 4.12, which ensures that if no further network changes occur, the following experiments using the same formula and the resulting r_{EFF} will give $p_{ON} = 1$. However, when, T_{OFF} after the end of protected time, Equation 4.3 were used with the values that were added to P before that transition, those (lower) values would result in $p_{ON} > 1$. While it may be argued that this has no different effect from using $p_{ON} = 1$ (the flow stays on), this

is still not acceptable, because this means that for a slight increase in congestion, the probability to remain on would still be greater or equal to one.

To solve this problem, during the transition, the values added to P during the first T_{OFF} interval after protected time need to be re-calculated in such a way that they represent the values that would have resulted from using Equation 4.3 instead of 4.12 during that time. Alternatively, a second set P^* can be used to calculate values with 4.3 in parallel. P would then be set to P^* during the transition.

The behavior of PCC using a second set P^* is summarized in the finite state machine given in Figure 4.1.

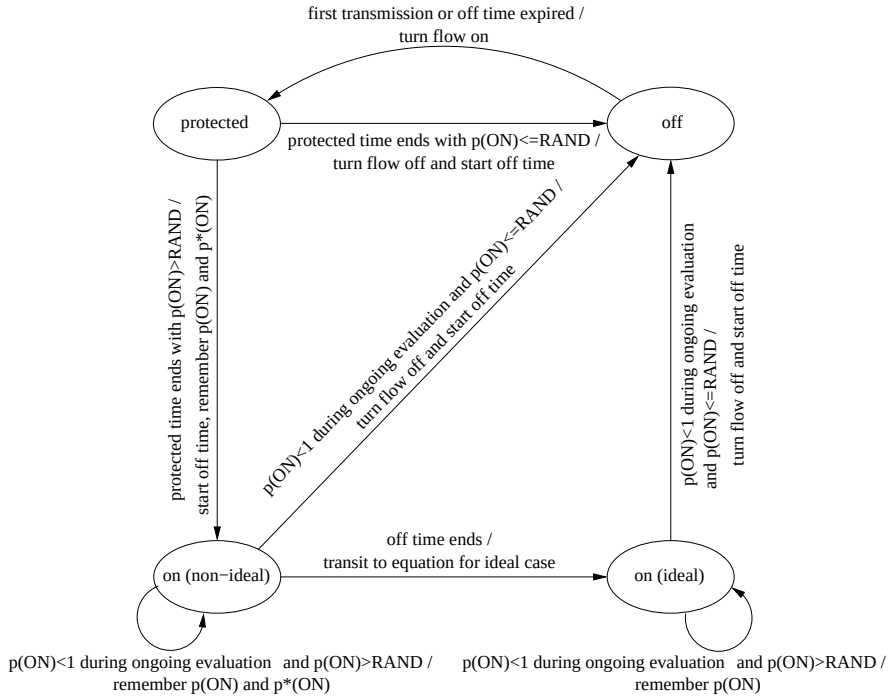

Figure 4.1: The PCC Scheme as a Finite State Machine

4.6 Misbehaving Participant Issues

In order to explore the vulnerabilities of the PCC scheme to misbehaving participants, the four cases of misbehaving senders or receivers in the sender- or receiver-based case shall be studied separately:

The two cases where the misbehaving participant is the one that has control over the congestion control scheme are trivial: since the well-behaved participant has only a passive role in the scheme, there is no way it can punish the other one for its behavior, or indeed even realize that it is misbehaving.

The only way for a receiver in a sender-based scenario to misbehave to its advantage is to “fake” information about the network condition, such as reporting fewer than the actual number of loss

events. This is hard to detect unless the sender has additional information about the network condition from other sources.

In a receiver-based scenario, the most obvious way for a sender to misbehave is to ignore any signals by the receiver to stop transmission. Such behavior is of course detected by the receiver, which receives additional data when it expects to receive none.

Receiving data while the flow should be off is not necessarily a sign for the receiver that the sender is misbehaving. Another reason this can happen is that the signal to stop has been lost. The receiver should therefore repeat the signal in such a case.

Another thing the receiver can do in response to a sender that does not react to off signals is to simply not accept data anymore, i.e. not to pass the excessive data on to the application. This would eliminate any profit the sender gets from misbehaving and thus the incentive to behave uncooperatively (see [1]).

However, this approach is highly debatable, since it involves discarding some amounts of data that have already got all the way through the network. Not only does the approach not benefit the network situation directly (only indirectly through demotivation of further malicious activity), it may also provoke a situation where this same data is re-transmitted, thus causing more network load than without the approach. Also, it is important to note that the sender usually draws little profit from transmitting more data, as it is the receiver that requests and ultimately uses the data. Other methods using router support in order to discard excess data early in the network may be more effective, though out of the scope of PCC.

Another way how a misbehaved sender can exploit PCC in order to send more than its fair share of data is to split up one large flow into many smaller flows that send at a rate smaller than the TCP-friendly rate and are thus not required to stop. This method assumes however, that the receiving application is able to make use of the partial flows.

It should be noted at this point, that the possibilities of exploiting PCC described above are no more and no less harmful than the possibilities of exploiting other protocols currently used in the Internet, such as TCP. A misbehaved TCP-sender might choose not to decrease its send rate in response to loss events, while a misbehaved TCP-recipient could make use of such mechanisms as implemented in TCP Daytona [13].

4.7 Example of PCC Operation

A brief example, depicted in Figure 4.2, shall illustrate the operation of PCC in the receiver-based case:

Given is a non-adaptable flow f_{NA}. After $T_{PROT} = 3$ seconds the receiver has determined an initial estimation of $r_{NA} = 100kbit/s$ and $r_{TCP} = 80kbit/s$. The application developer decided that for the given application a good value for T_{OFF} is 60 seconds. With this information we are able to derive the probability p_{ON} for keeping the flow in the on state from Equation 4.7.

Equation 4.7 yields the result $p_{ON} = 0.79$ when the parameters from flow f_{NA} are used. This is

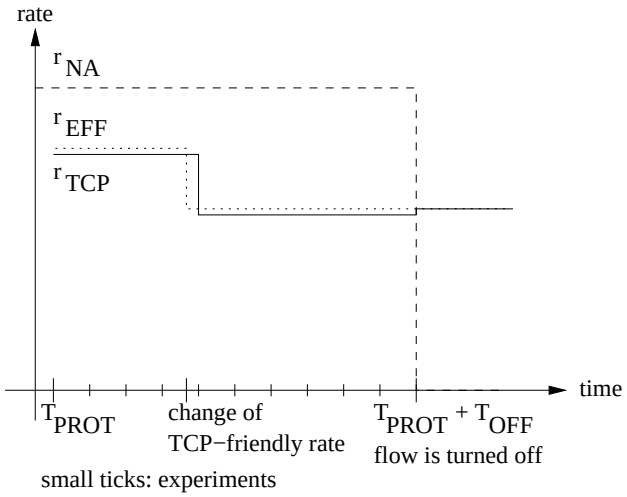

Figure 4.2: PCC Operation Example

reasonable considering that r_{TCP} is one fifth less than r_{NA} and there is a short protected time to be made up for. Now a random number is drawn from the interval $(0, 1]$ deciding whether the flow will stay on or be turned off. Given that we have a high number of statistical multiplexing (R1) this would result in roughly 1 out of 5 PCC flows being turned off, so that the remaining PCC flows use a fair, TCP-friendly overall share of the bandwidth.

Let us assume that for the flow we are examining the random number is 0.6. In this case the flow will continue to be on. p_{ON} is saved as the first value in the set P, while $p^*_{ON} = \frac{r_{TCP}}{r_{NA}} = 0.8$ is inserted into a temporary set P^*. Now consider the case that the congestion in the network increases after a few seconds and that the receiver will now estimate r_{TCP} to be only $60kbit/s$. In order to calculate a new p_{ON} for this situation PCC uses Equation 4.12 with $r_{EFF} = 0.79 \cdot r_{NA} = 79kbit/s$. This results in the new value for $p_{ON} = 0.73$. Since p_{ON} is now again in the interval $(0, 1)$ another random number is drawn to decide whether the flow should stay on or not. Let us assume that the random number is 0.4, so that the flow will stay on. p_{ON} is added to set P, making $r_{EFF} = 0.79 \cdot 0.73 \cdot r_{NA} = 58kbit/s$. $p^*_{ON} = \frac{r_{TCP}}{r_{EFF}} = \frac{60}{0.8 \cdot 100} = 0.75$ is added to P^*.

T_{OFF} seconds after the initial p_{ON} has been calculated, that value times out and is removed from the set P. For the same reason, the initial p^*_{ON} is removed from P^*, then P^* replaces P. The new value for p_{ON} is calculated based on Equation 4.3 (using r_{EFF} instead of r_{NA}), because the flow is no longer in the first T_{OFF} after its initialization. r_{EFF} is calculated to be $0.75 \cdot r_{NA} = 75kbit/s$, since 0.75 is now the only value in set P. If the value for r_{TCP} has not changed this results in a new value for $p_{ON} = \frac{60}{75} = 0.8$, which is inserted into P. This time 0.9 is drawn as the random number and therefore the flow is turned off for T_{OFF}, before it will start again with a period of protected time. At this point, r_{EFF} has the value of $0.75 \cdot 0.8 \cdot r_{NA} = 60kbit/s$.

Chapter 5

PCC Implementation for the Network Simulator

Testing a new protocol directly in the real life Internet would involve not only extensive and complex setups but also the considerable risk of severely damaging network conditions with errors in the protocol and/or the protocol implementation. For this reason, the network simulator ns-2 [14] was used to evaluate PCC fairness and throughput, and a special version of the protocol for that simulator was implemented.

Besides avoiding damage to a real life network, protocol testing with a simulator has further advantages, including the ability to control the network environment, the lack of limitation on the number of participating clients, and better comparability of results. The drawback is of course that simulations can always model real-life conditions only to a certain extent.

The following section gives a brief overview of the ns-2 Network Simulator, before the further sections go into details on the implementation of the protocol. For a more detailed version of the overview, see [15].

5.1 The Network Simulator ns-2

The ns Network Simulator (current version ns-2) is an event driven simulator for computer networks and network protocols. Since it is widely used in and contributed to by the research community, a large number of network components are available for ns. Amongst others, ns supports the following technologies:

- Point-to-point connections
- Different router discard strategies (DropTail, RED, etc.)
- IP
- Multicast

- TCP, UDP, and several experimental transport protocols
- Applications (Telnet, FTP, WWW-like traffic, etc.)
- Network emulation (i.e. interaction of the network simulator with a “real” operating network node)

The simulator framework uses a split-language programming approach. The simulator core, including the actual protocol implementations, is written in C++, while the control structure and the description of simulation scenarios is done in OTcl, an object oriented version of Tcl. This approach utilizes the flexibility advantages of a scripting language for scenario creation while taking advantage of the better efficiency of compiled code where necessary.

Any useful ns scenario consists of the following components:

- The topology
- Communication Patterns
- Events

The topology is made up of nodes with queues and links between the nodes, and defines network conditions such as the link bandwidth, delay, random losses, and queue sizes. Communication Patterns determine what kinds of traffic are transmitted between the nodes. Common events are starting and stopping a flow, or changing the random loss rate on a link.

A protocol implementation for ns mainly consists of two objects representing sender and receiver, with methods deciding when to send what kinds of packets and how to react to incoming packets.

5.2 Major Design Choices

Rather than trying to re-invent the wheel, PCC uses existing mechanisms to measure loss event rate and round-trip time, as well as to estimate the TCP-friendly rate from these measurements. We chose to use the mechanisms provided by TFRC [5], mostly because a TFRC implementation is available with ns-2, but also because of interesting possibilities for future extensions to PCC which will be discussed in Chapter 8. The PCC implementation reuses some TFRC methods with few or no changes, while other methods were largely rewritten.

An important design decision was to implement a receiver-based version of PCC. The general idea is to let the receiver perform as many congestion control related tasks as possible, most prominently the estimation of the TCP-friendly rate and the decision whether to turn a flow on or off. The main reason for this design decision is that it simplifies a future extension of PCC to multicast.

In the process of implementation, the receiver-based version of PCC evolved in three steps: First, the sender-based version of TFRC provided with ns-2 was rewritten into a receiver-based version, dubbed RTFRC. Next, this version of TFRC was stripped of all functionality not required by PCC, such as slowstart, resulting in a working, though in terms of congestion control useless basic protocol. Finally, PCC functionality was added to the basic protocol.

Further design choices were:

Support for $T_{PROT,MAX}$. Support for a maximum protected time application parameter is provided for the variety of reasons mentioned in Section 4.4.4. Most scenarios deployed in protocol evaluation use a value of $T_{PROT,MAX} = 30s$, since it is well below the 60s used for T_{OFF} and high enough to become relevant in only a few cases for most network conditions. A designer of a scenario who does not wish to use this parameter is free to set it to a value large enough never to become relevant (e.g., a value larger than the number of seconds the scenario is scheduled to run).

Temporary adjustment of T_{OFF}. We assume that for most applications using this protocol, the exact value of T_{OFF} given as a system parameter is important. Therefore, T_{OFF} is not adjusted permanently when p_{ON} is negative. Instead, the extended value is used only for the off time directly succeeding the experiment in question. While a real life implementation should probably support both types of adjustment, it can be expected that for purposes of evaluating protocol fairness and throughput, both alternatives are largely equivalent.

Application knowledge of send rate. In order to avoid the added complexity of measuring the send rate at the receiver, the use of application knowledge of that rate is assumed. This is straight-forward for the ns implementation, since the application and the protocol are integrated into the same objects. The send rate is determined by an application parameter at the sender that can be set from a scenario script by ns events, and passed on to the receiver as control information. Furthermore, in most test scenarios, this parameter is not changed, so that effectively a constant bit rate flow is used.

Avoiding phase effects. A highly deterministic model like the network simulator is prone to phase effects that can seriously degrade the quality of test results. Therefore, a random element is used to give some degree of variance to the otherwise constant send rate. The approach for this is taken directly from the TFRC implementation.

5.3 PCC Control Mechanisms

PCC control mechanisms rely mainly on the transmission of control data from receiver to sender and vice versa, and the utilization of various timers:

5.3.1 PCC Control Packets

As in TFRC, control packets are used by the receiver to pass information to the sender. In receiver-based PCC, this information includes a timestamp used for round-trip time calculation and, most importantly, the rate at which the sender is allowed to transmit packets. This rate is always either 0, meaning the flow is off and the sender may not transmit any data, or infinity (represented by a very large double), meaning the flow is on and the sender may transmit at any rate it chooses. While a binary flag would be enough to carry this information, the double value used by TFRC was retained in order to facilitate an integration of PCC and TFRC as described in Chapter 8.

Control packets by the receiver are transmitted at least once every round-trip time, if new packets arrive in that time. If no packets arrive after a control packet has been sent, there is usually no new

information to be sent. However, in order to take into account the possibility of losses on the path to the sender, control packets are transmitted even when no new data arrives, but at a much lower frequency. Control packets are not transmitted when the sender is a) supposed to be off and b) not transmitting data. A control packet with rate = infinity turns a flow that was off on again.

To minimize the delay between the decision to turn the flow off and the actual stop of transmission, control packets by the receiver are transmitted immediately when an experiment fails.

Control data going from the sender to the receiver is transmitted via header fields in every data packet. This includes timestamp information for round-trip time calculation as well as the current send rate (application level knowledge) and a sequence number used to detect packet loss.

Figure 5.1 visualizes the types of control data flowing between sender and receiver along with some of the main tasks performed by the participants.

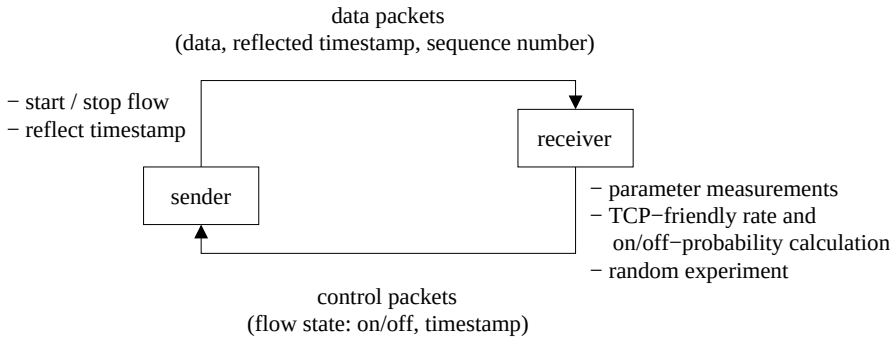

Figure 5.1: Flow of PCC Control Data

5.3.2 PCC Timers

The PCC receiver uses timers to decide when to send the next control packet, when to stage the next experiment, and when to turn a flow that was off back on:

The **control packet timer** is set to the current smoothed value of the round-trip time every time it expires. The decision whether to actually send a control packet is done at every expiration. When the flow is off, the control packet timer is not rescheduled, to save unnecessary overhead.

The **experiment timer** is set to the application parameter T_{EXP} after every successful experiment. After an unsuccessful experiment, the timer is only rescheduled after the experiment following the protected time after the flow has been turned on again. The expiration of the experiment timer invokes the next experiment.

The **off timer** is set to the application parameter T_{OFF} after an unsuccessful experiment. When it expires, the flow is turned back on by transmitting a corresponding control packet. Also, the control packet timer is rescheduled to turn regular transmission of control packets back on.

The PCC sender uses timers to decide when to send the next data packet, and to react to the continuous absence of expected control packets from the receiver:

The **send timer** is set to the inter-packet gap defined as packet size divided by the desired send rate whenever it expires. In addition, a random element is used that varies this gap by at most 50%[1] in order to avoid phase effects. When the timer expires, a data packet is sent. This method ensures an approximate send rate of the desired value.

The **no feedback timer** is set to the current smoothed round-trip time (which is part of the receiver's control data) multiplied by a constant (currently 24) whenever a new control packet provides feedback from the receiver. When the timer expires, the sender assumes extreme congestion in the network or failure of the receiver and stops transmission. The no feedback timer is based on the round-trip time, because the receiver intends to transmit one control data packet per round-trip time. The constant is meant to secure the mechanism against sudden increases in round-trip time (not yet reflected by the smoothed value) as well as the possibility of control data packet loss.

5.4 TFRC Mechanisms Used to Calculate the TCP-Friendly Rate

TFRC mechanisms are deployed by the PCC implementation for ns-2 to measure the round-trip time, to measure the loss event rate, and to estimate the TCP-friendly rate from these measurements. The full TFRC specification is available in [5]. The following subsections describe the mechanisms as they are used by PCC:

5.4.1 Measuring the Round-Trip Time

The sender and receiver cooperate to measure the round-trip time using timestamps. Every control packet by the receiver contains the time that packet was sent, and this timestamp is echoed by the sender with every transmitted data packet. In addition, the data packets contain an offset giving the time that passed from the moment the sender received the control packet until the data packet containing the echo was sent.

The round-trip time RTT is derived from the information in a data packet received at time t_{RCV} as $RTT = t_{RCV} - echo - offset$. The receiver then smooths the measured round-trip time using an exponentially weighted moving average (EWMA). The weight gives a trade-off between the responsiveness to changes in round-trip time and the reliability of the measurement.

Note that by this method, the one half of each round-trip time measurement pertaining to the path from receiver to sender remains constant for all measurements between two control packets, while there is a new value each time for the opposite path. An alternative to this is to only use one measurement per round-trip time. However, it seems reasonable to make use of the information the other, semi-current values provide as well, and to use a correspondingly smaller EWMA weight.

[1] The maximum variance can be set by an application parameter.

5.4.2 Measuring the Loss Event Rate

Loss events are used as an indicator of congestion rather than single losses since the same is true for TCP, where an adaptation of the send rate to a lower value in response to packet loss is performed at most once per round-trip time. See [5] for more discussion on this subject. Loss events are detected through the use of sequence numbers in data packets. A loss event is caused by any number of missing packets within one round-trip time.

A good method for measuring the loss event rate should respond quickly to an increase in the loss event rate, while also showing an acceptable response to a decreasing rate. The Average Loss Interval method used by TFRC and also employed by PCC was developed to meet these requirements.

The basic idea of this method is to count the number of packets between two loss events (the "loss intervals"), then adding up a certain number of loss intervals using weights, and finally taking the inverse value as the loss event rate. The number of loss interval samples used for the measurement in PCC is an application parameter (N_{SAMP}).

"There is a clear trade-off between measuring the loss event rate over a short period of time and responding rapidly to changes in the available bandwidth, versus measuring over a longer period of time and getting a signal that is much less noisy" [5].

The weights used by the Average Loss Interval method are designed to result in a behavior that is somewhere between that of the Dynamic History Window method (where every value has the same weight, and no distinction is made between more and less current values) and that of the EWMA method (where the weights effectively decrease from value to value). Equal weights are used for the newest $\frac{N_{SAMP}}{2}$ loss intervals, while the older intervals receive linearly decreasing weights. With $N_{SAMP} = 8$, weights would be 1 for the newest four intervals, then 0.8, 0.6, 0.4, and 0.2 for the rest (see Figure 5.2). The average loss interval is arrived at by dividing the sum of the weighted intervals by the sum of the weights, in this example by six. The main advantages of this method are

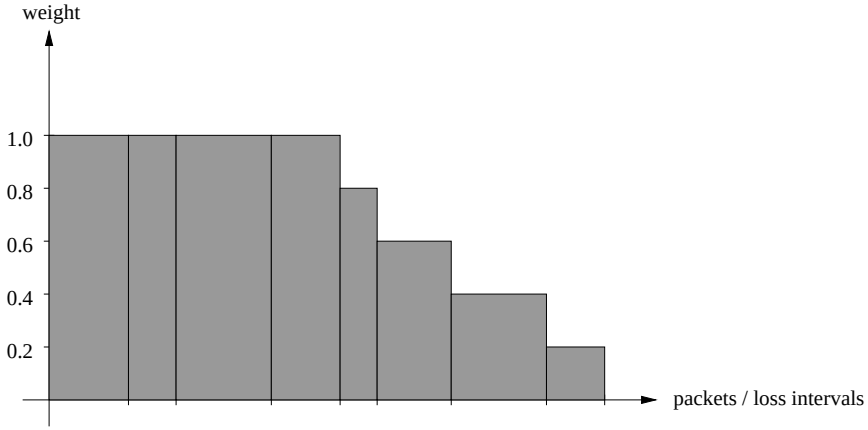

Figure 5.2: Weights Used in the Average Loss Interval Method

that the most recent interval is not over-emphasized and that the "use of a weighted average by the Average Loss Interval method reduces sudden changes in the calculated rate that could result from unrepresentative loss intervals leaving the set of loss intervals" [5].

For a given $n = N_{SAMP}$, the average loss interval $\hat{s}_{(1,n)}$ is calculated as a weighted average of the last n intervals s_i as given in [5]:

$$\hat{s}_{(1,n)} = \frac{\sum_{i=1}^{n} w_i s_i}{\sum_{i=1}^{n} w_i} \tag{5.1}$$

for weights w_i:

$$w_i = 1, 1 \leq i \leq n/2 \tag{5.2}$$

and

$$w_i = 1 - \frac{i - n/2}{n/2 + 1}, n/2 \leq i \leq n \tag{5.3}$$

An important note here is that this method also allows for calculating an average loss interval when fewer than N_{SAMP} intervals are available. In this case, the weights remain the same, but both sums in the fraction in 5.1 run only up to the number of available values. This possibility is important for PCC, where the application parameter giving the number of loss events expected for protected time is usually smaller than N_{SAMP} in order to keep protected time as short as necessary, and this option is in fact implemented in both PCC and TFRC.

The interval since the last loss event s_0 is dealt with in this context by only including it in the calculation if it is large enough so that this will increase the average. Optionally, when s_0 grows to more than twice the average loss interval, *history discounting* can be used for a faster response to the decrease in congestion responsible for this size of s_0. For more discussion on this subject, see [5]. *History discounting* is described in detail in [16].

5.4.3 Estimating the TCP-Friendly Rate

Whether the goal of a congestion control scheme is the TCP-friendly behavior of each single flow (as in TFRC) or of an aggregation of flows (as in PCC), a good estimation of the bandwidth a conformant TCP flow would use under comparable conditions is required. The designers of TFRC elected to use the TCP response function from Padhye et. al. [7] for this purpose:

$$r_{TCP} = \frac{s}{RTT\sqrt{\frac{2p}{3}} + RTOmin(1, 3\sqrt{\frac{3p}{8}})p(1 + 32p^2)} \tag{5.4}$$

This gives the average send rate of a conformant TCP flow in bytes/sec, assuming no limitation by the advertised window and a bulk data transfer, with packet size s, round-trip time RTT, steady-state loss event rate p, and the TCP retransmit timeout RTO. For simplicity, TFRC and PCC set RTO to $4 \cdot RTT$. This has proven to be sufficiently accurate in practice (compare [5]).

Equation 5.4 models a TCP flow that does not use delayed ACKs. A further parameter b, giving the number of packets that are acknowledged by a received ACK, is used in [5] to adapt the equation to other TCP versions that do use this mechanism. In heterogeneous networks where a relevant number of different versions of TCP coexist, a good average needs to be chosen for b. How this average can be arrived at is out of the scope of this work.

Note that with $RTO = 4 \cdot RTT$, r_{TCP} is inversely proportional to the round-trip time. Also, r_{TCP} decreases with increasing loss event rate p.

Since the requirements for a TCP-friendly rate estimator are similar in PCC and TFRC, PCC also uses response function 5.4 to estimate the TCP-friendly rate.

5.5 Activity Diagram for the Design of a PCC Receiver

The following UML activity diagram illustrates the flow of control within a PCC receiver, starting with the initialization of the receiver and ending when the sender closes the connection[2]. Since the chosen design is receiver-based, the sender side is much less interesting and therefore not presented in a diagram.

For better clarity, Boolean variables are used in the diagram, while these are, due to the lack of a Boolean type in C++, implemented as integers.

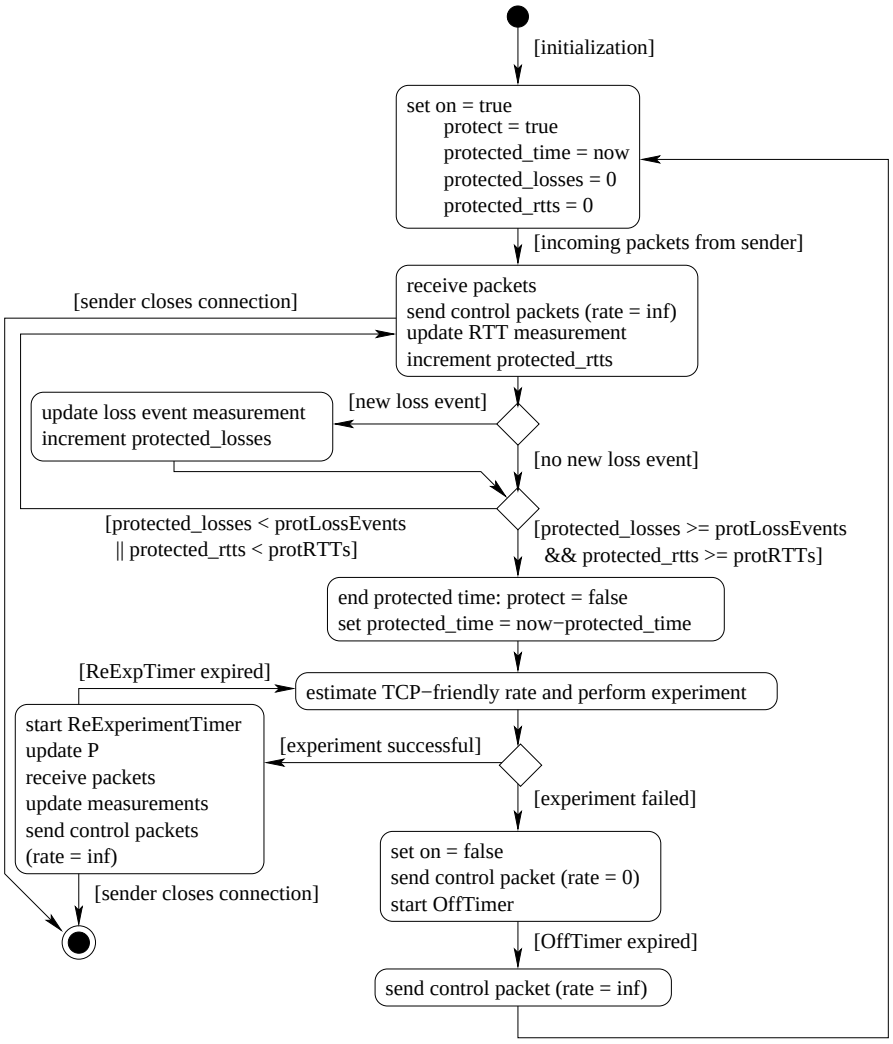

Figure 5.3: UML Activity Diagram of a PCC Receiver

[2]Receiver-initiated open and close of connection is not presented since this is out of the scope of the PCC congestion control protocol

Chapter 6

Simulation Results

The network simulator ns-2 has been used to simulate PCC flows competing with TCP traffic across a wide variety of network conditions. Most simulations showed very good results in terms of fairness and throughput. However, the simulations also proved very helpful for detecting some weaknesses of the PCC implementation connected with the use of the TFRC mechanisms for estimating the fair rate.

6.1 Simulation Settings

Table 6.1 gives the PCC and network settings used in the standard scenario. The standard scenario is meant to simulate an average PCC-based application in a stable network with a single bottleneck link exhibiting the bandwidth and round-trip time of a common Internet connection. Figure 6.1 shows the "dumbbell" topology that was used for all scenarios, with two PCC and two PCC senders and receivers[1].

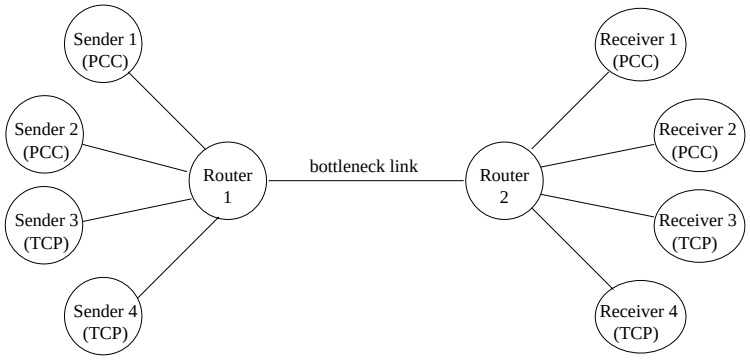

Figure 6.1: Dumbbell Topology for Simulations

All scenarios compare PCC flows with a fixed application send rate to TCP flows performing bulk

[1]In order to achieve a high level of statistical multiplexing (R1), the standard scenario uses 50 senders and receivers each

parameter	value
r_{NA}	32 packets/s
PCC Packet Size	1000 bytes
TCP Packet Size	1000 bytes
PCC Control Packet Size	40 bytes
RTT EWMA Weight	0.2
N_{SAMP}	24
History Discounting	used
RTTs for Protected Time	5
Loss Events for Protected Time	3
$T_{PROT,MAX}$	30 seconds
T_{OFF}	60 seconds
T_{EXP}	2 seconds
Simulation Run-Time	1800 seconds
Number of PCC flows	50
Number of TCP flows	50
Bottleneck Bandwidth	25600 kbit/s
t_{RTT}	100 ms
Router Discard Strategy	DropTail
Random Packet Loss	none

Table 6.1: PCC and Network Parameters Used in the Standard Scenario

data FTP transfers. With bulk data transfers, TCP send rate is limited only by network conditions, and not by the application. Note that the same is true for the effective send rate of a PCC flow, if $r_{NA} \geq r_{TCP}$.

In the following sections of this chapter, the term “simulation” will be used as describing a single run of the network simulator providing one fairness result. The term “simulation set” shall refer to a set of simulations using the same scenario but varying one or more parameters, resulting in a plot of fairness values. Also, in order to achieve good averages in the face of random effects and other variance, three simulations were usually staged per parameter value (or combination of parameter values) in a simulation set. The values displayed in the plot are the averages of the three simulations. In addition, to provide some information about variance, the three single simulation results were in most cases added to the plot. While fairness is the most prominent characteristic investigated, a number of other attributes such as throughput were measured and plotted.

6.1.1 Example of a Single Simulation

The following figure shows an exemplary simulation using a PCC application send rate of 48 packets/s and otherwise the parameters given above. Plotted is the throughput over time of a single PCC flow and of a single TCP flow. In addition, the average throughput and a line signifying the fair share for each class of flows is displayed. Throughput values are averaged over 10 seconds.

In this example, the PCC flow is turned off ten times; the third time it is turned off immediately

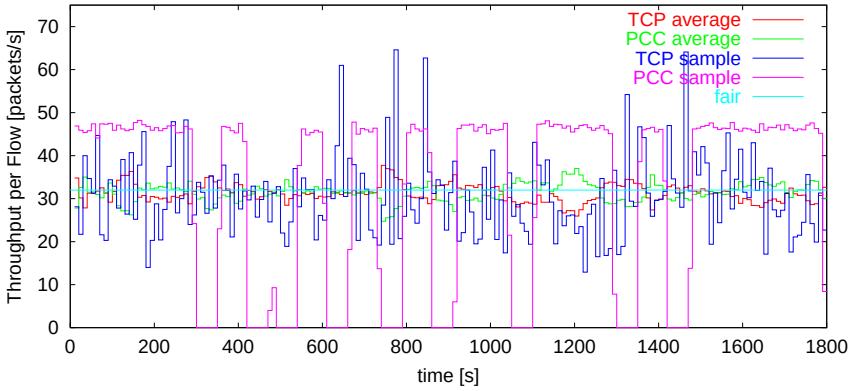

Figure 6.2: Throughput Over Time of a PCC and a TCP Flow

after protected time. Since in this example PCC and TCP throughput both oscillate around 50%, the fairness value in terms of PCC share of bandwidth resulting from this simulation is approximately 50%. For each of the simulation sets described below, three such simulations were run per parameter combination for in order to achieve reliable values.

6.2 Metrics for Fairness and Throughput

The goal of the simulations was to see how PCC behaves in terms of fairness and throughput when confronted with a variety of network conditions. We use the following metrics for measuring these characteristics:

6.2.1 Metrics for Fairness

An aggregation of flows is deemed fair to the same number of parallel TCP flows if both use the same amount of bandwidth. As there are only PCC and TCP flows in the network in our scenarios and most use an equal number of PCC and TCP flows, fairness in those scenarios is optimal if PCC gets exactly 50 percent of the bandwidth that is actually in use, as measured in the sum of packets received by all PCC receivers divided by the sum of packets received by all PCC and TCP receivers over the course of the simulation.

For scenarios with the number of PCC flows N_{PCC} equal to the number of TCP flows N_{TCP}, $\frac{Thp_{PCC}}{Thp_{PCC}+Thp_{TCP}}$ (with Thp_X being the aggregate throughput of all flows using protocol X) would therefore be an intuitive fairness metric, where 50 percent means completely fair, $> 50\%$ means PCC is overly aggressive, and $< 50\%$ means PCC is not aggressive enough. We call this metric the *"share of bandwidth"*.

While this metric is very intuitive for scenarios where the same number of PCC and TCP flows is used, it is less useful in cases where the number of PCC and TCP flows is different, or indeed varying, as in this case a bandwidth share of 25 percent may well be the fair share. To accommodate

these cases, the metric is normed using the average per-flow throughput of TCP and PCC:

$$PCC\ share\ of\ bandwidth = \frac{\frac{Thp_{PCC}}{N_{PCC}}}{\frac{Thp_{PCC}}{N_{PCC}} + \frac{Thp_{TCP}}{N_{TCP}}} \tag{6.1}$$

With this normed metric, PCC is fair at 50%, regardless of the relative numbers of PCC and TCP flows.

We chose to use this metric because it is intuitive for most of the simulation sets that were performed. Another advantage is that where the metric shows some degree of unfairness, the degree shown is meaningful. For example, a PCC *share of bandwidth* of 10% means that PCC receives only one fifth of the share it should, while a share of 60% means PCC is more aggressive than it should be.

In the following sections, the term “fairness” will be used synonymously with “share of bandwidth”.

While the *share of bandwidth* is very intuitive in showing the relative fairness of an aggregation of PCC flows compared to an aggregation of TCP flows, it says nothing about the fairness of one PCC flow to another. This so-called *intra-protocol fairness* is interesting, since it may not be desirable to have many PCC flows receive very little bandwidth over the course of the simulation while others send at the full rate most of the time.

In order to measure intra-protocol fairness, we chose to use the *fairness index* proposed by Jain, Chiu, and Hawe [17]:

$$f_A(x) = \frac{(\sum x_i)^2}{n \cdot \sum x_i^2} \tag{6.2}$$

where x is a set of values x_i, in our case a set of single flow throughput values over the course of a simulation.

The *fairness index* yields continuous values in $(0; 1]$ independently of the number of values in x and of the measurement unit. Other metrics, such as the *Coefficient of Variation* and the *Min-Max Ratio* have been studied in [17], but all lack at least one of these properties.

A drawback of the *fairness index* is some lack of intuitiveness due to the fact that relatively large values can be achieved with a relatively low degree of fairness. A number of flows with a throughput of zero and the same number of flows sending at full rate would for example still achieve a *fairness index* of 0.5, and some half-rate flows pitched against the same number of full-rate flows would get a *fairness index* as high as 0.9. Therefore, only values above 0.9 should be considered as showing good fairness.

The *fairness index* was also used to measure fairness across PCC and TCP flows, though this is less meaningful, as it aggregates the effects of PCC and TCP intra-protocol fairness as well as that of PCC fairness towards TCP.

6.2.2 Metrics for Throughput

To measure the average throughput of PCC and TCP flows as well as the average throughput of all flows in a way that makes it easy to compare results gained from different simulation sets, a metric

giving relative values was employed. In addition, the absolute average throughput of all flows was measured.

Relative Throughput. The average relative throughput of PCC flows, TCP flows or all flows is measured as the average number of packets actually transmitted in one second by all respective flows divided by the number of packets that would be transmitted by the same number of flows at exactly the fair rate. The fair rate is defined as the total bottleneck link bandwidth divided by the number of all active[2] flows. The resulting value is multiplied by 100 to give the throughput as a percentage of the fair bandwidth.

Absolute Throughput. The average absolute throughput of all PCC and TCP flows is measured simply as the average number of packets actually transmitted in one second divided by the number of flows.

6.3 Correspondence of Fairness with TCP-Friendly Rate Estimation

An obvious simulation set to stage is to take the setup presented at the beginning of the chapter and to vary the send rate of all PCC flows, testing PCC behavior at rates below the fair rate as well as rates above the fair rate. The expected fairness result of this simulation set is shown in Figure 6.3. Sending at a lower than fair rate, PCC should not be required to turn any flows off, and TCP, due to its property of quickly searching out free bandwidth, should send at an accordingly higher rate. If for example all PCC flows send at three quarters of the fair rate, an equal number of TCP flows can send at $1\frac{1}{4}$ times the fair rate, whereby the PCC *share of bandwidth* should be $\frac{0.75}{0.75+1.25} = 0.375$. Sending at the fair rate or a higher rate, PCC should receive exactly 50% of the bandwidth.

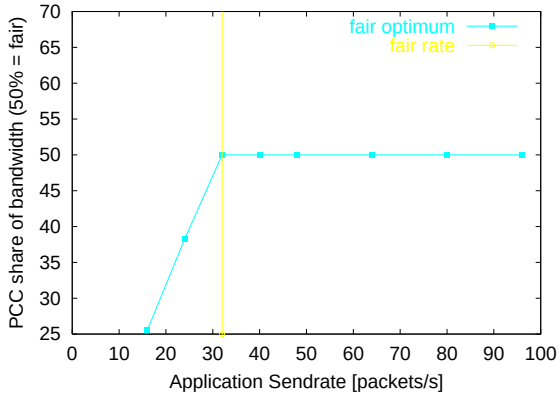

Figure 6.3: Expected Fairness when Varying Send Rate

While Figure 6.3 shows the expected fairness, the actual values can differ considerably due to random effects, even if the protocol implementation were working perfectly. Since the main goal of any effort in congestion control is to prevent congestion collapse, bandwidth shares in the range from 30% to 70% can be considered acceptable. This is also the range that other efforts such as

[2] A PCC flow is called active when it is transmitting data, but also when it is temporarily in the off state. The same holds for a TCP flow currently performing exponential back-off after a timeout.

TFRC have achieved. However, any clear tendencies revealed in a simulation set plot, especially one that uses averages over a number of similar simulations, need to be studied carefully as they are likely to be caused by some other effects than randomness.

Figure 6.4 shows the actual results of this simulation set superimposed on the optimum line from Figure 6.3. In addition to the average, the three single simulation results are displayed as the lines tagged i) through iii). While the actual fairness values correspond well to the expectations and

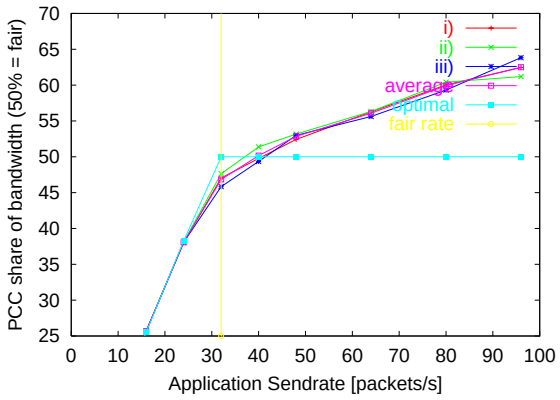

Figure 6.4: Actual Fairness when Varying Send Rate

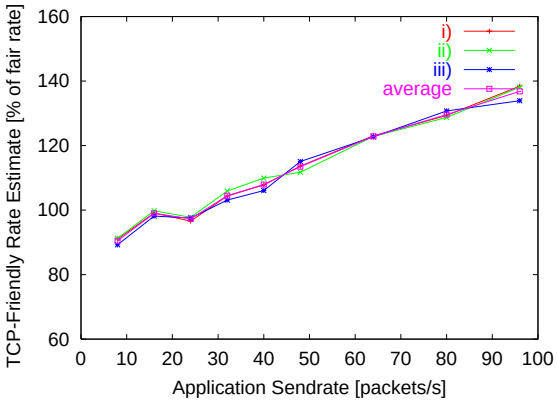

Figure 6.5: Estimated TCP-Friendly Rate when Varying Send Rate

all values from the fair rate onwards remain within the acceptable range, some other properties of Figure 6.4 point to potential problems:

- All single simulations have a fairness below 50% for the fair rate.
- The PCC share of bandwidth has a very clear tendency of increasing above 50% with an increasing application send rate.

These problems were in fact confirmed in most other simulation sets. They will be discussed in detail in the following two sections.

Another property of PCC that has been widely confirmed by other simulation sets can be seen when comparing Figures 6.4 and 6.5. The development of fairness values corresponds very closely to that of the average TCP-friendly rate estimate, normed to a percentage of the fair rate in Figure 6.5.

This property indicates a crucial point: The core PCC mechanism, which is responsible for adapting the average expected send rate to the estimated TCP-friendly rate by turning flows on and off appears to be working very well, while the estimation of the TCP-friendly rate used in the implementation seems problematic. The following sections will in fact point to possible causes of the problems seen in Figure 6.4 that are connected with the estimation of the TCP-friendly rate, more to the point, with the measurement of the loss event rate necessary for the estimation. It should also become clear why these mechanisms work well in TFRC and less well when used in the PCC context.

The attentive reader may have noticed that the correspondence of Figures 6.4 and 6.5, while apparent in tendency, is not so close in value. With the PCC mechanism working well, it should be expected that an estimate of above 100% would yield a fairness of above 50%, which is not the case e.g. for the fair rate of 32 packets/s. However, this upward shift in the estimate curve should be attributable

to the fact that PCC flows are more likely to be turned off when they encounter low TCP-friendly rate estimates. Therefore, flows temporarily under-estimating that rate will be turned off sooner, thereby producing fewer estimates, while flows temporarily over-estimating the rate will remain on and produce more estimates. For the measurement of the average TCP-friendly rate estimate, all single estimates of all flows are averaged, more of which come from flows with mostly higher than average estimates.

6.4 Effect of Variation in Loss Event Rate Measurement on PCC Flows

In a dynamic network, there is a certain degree of variation to most network parameters, most notably to the loss event rate. This is enhanced by the presence of TCP flows exhibiting TCP's well-known saw-tooth behavior. Only to some degree can this variation be compensated by a larger measurement interval (i.e. a larger number of samples), since a very large interval would base the measurement too much on old values that have little significance for the current network situation.

It is therefore inevitable that the loss event rate is at times over-, at other times under-estimated. In simulations even with a relatively stable network environment, this variation has commonly shown to be in the range of up to +/-40%.

The formula from [7] that is used by the current PCC implementation to estimate the TCP-friendly rate r_{TCP} is based on the measurement of the loss event rate and the round-trip time. Therefore, there will be a considerable amount of variation in this estimation and thereby in the probability value p_{ON} as well, even if it is assumed that the average round-trip time can be estimated well (an assumption that is supported by most simulation results).

As long as all estimations of p_{ON} are smaller than one, variation does not pose a problem. The statistical likelihood is that over a sufficient amount of time and/or with sufficient multiplexing, the effects of over- and under-estimations will cancel each other out. While some flows stop when they would not need to, other flows will remain on when they would normally be required to stop. Therefore, variation does not pose a problem in situations where the effective PCC send rate r_{EFF} is considerably higher than the fair rate. Note however that during continuous evaluation, r_{EFF} is usually close to the fair rate unless network conditions change or significant values are removed from set P.

Variation also does not pose a problem in situations where r_{EFF} is considerably lower than the fair rate. Here, all estimations of p_{ON} are larger than one, and no flow is required to stop.

A problem arises in situations where r_{EFF} is close to the fair rate. Consider the case of r_{EFF} = actual r_{TCP}. Here, an under-estimation of r_{TCP} will lead to $p_{ON} < 1$, so that there is some chance that the flow has to stop, whereas an over-estimation of r_{TCP} will increase p_{ON} to a value > 1, which merely has the same effect as $p_{ON} = 1$, that is the flow can remain on. This effect is present not only in the case of r_{EFF} = fair rate, but in any case where variation can cause some estimates to be < 1 and others > 1, though it has the strongest impact on fairness in that case. The upshot of this effect is that in these cases, PCC uses less than its fair share of bandwidth. This lack of aggressiveness at a send rate close to the fair rate can be observed well in Figure 6.4.

The existence of this effect is further evidenced by the following simulation set, wherein the number

of loss event samples is varied from 4 to 32 for PCC flows sending at exactly the fair rate. Obviously, with fewer samples, the variance in the loss event measurement and therefore the variance in the TCP-friendly rate estimate is higher, so the effect should be more visible. This can indeed be seen in Figure 6.6.

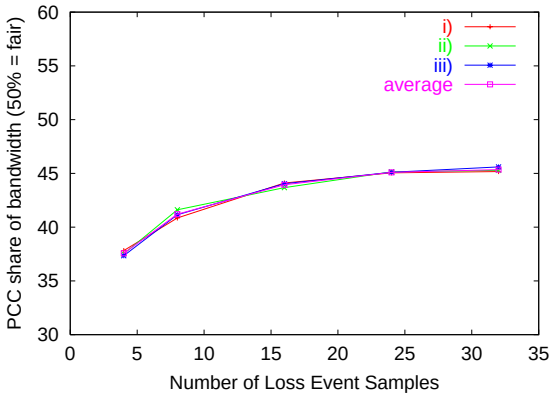

Figure 6.6: Fairness when Varying the Number of Loss Event Samples

The effect described in this section does not apply to the TFRC protocol, since it is a direct result of PCC treating probabilities > 1 as one, and TFRC does not use any probabilities. TFRC over-estimates r_{TCP} and thereby the rate a TFRC sender is allowed to send at to the same amount that it under-estimates that rate at other times. However, the effect is still caused by the measurement mechanism used for the loss event rate. If there was a mechanism at the disposal of a PCC implementation that always yielded the correct average loss event rate of the network, this effect would not persist. Due to the limited amount of measurement data that is available locally though, such a mechanism would probably need to involve global knowledge. It is very likely that it is better to either accept this (comparatively slight) effect or to work out a way to heuristically increase p_{ON} when the send rate is close to the fair rate, than to put up with the problems a global knowledge approach would pose.

6.5 Problems of TCP-Friendly Rate Estimation Based on the Loss Event Rate

The problem of a share of bandwidth increasing with the application send rate that was noticed in the simulation set described in Section 6.3 corresponds to increasing TCP-friendly rate measurements, which are in turn caused by a decrease in the loss event rate measured by the PCC flows. This can be seen by comparing Figures 6.4 and 6.5 with Figure 6.7. Notice that the loss event rate decreases with increasing send rate, whereas the loss rate actually goes up. While the increase in loss rate follows from the increased aggressiveness of the aggregation of PCC flows, as expressed by the share of bandwidth, the reasons for the decrease in loss event rate, which is clearly the cause of that aggressiveness, are more complex.

A PCC flow sending constantly at the fair rate is (apart from the effect described in Section 6.4) not required to stop, so it behaves like a normal constant bit rate flow. With the premise of random

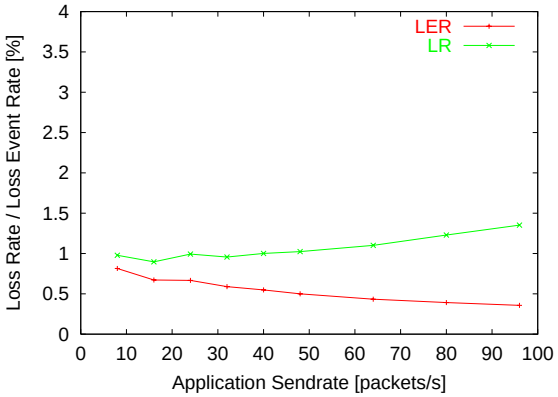

Figure 6.7: Loss Rate and Loss Event Rate when Varying Send Rate - Simulation Set with Some Correlated Losses

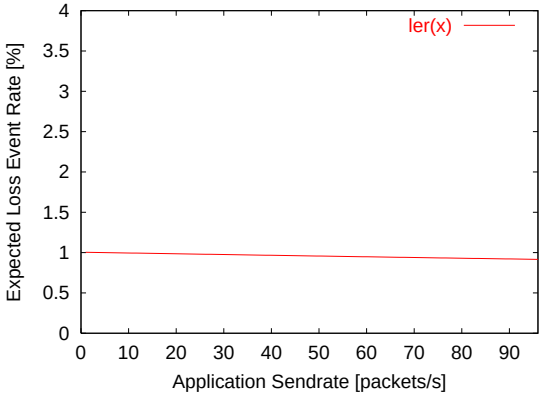

Figure 6.8: Expected Loss Event Rate when Varying the Number of Active Round-Trip Times - Random Losses, Loss Rate 1%

packet loss p (with very high multiplexing, packet loss in real-life networks is indeed close to random), the expected loss event rate l_{EXP} that such a flow experiences is given in [18] as

$$l_{EXP} = N \cdot \left(\frac{1-(1-p)^{\frac{T_{SIM} \cdot r_{NA}}{N}}}{T_{SIM} \cdot r_{NA}}\right) \tag{6.3}$$

where T_{SIM} is the run-time of the simulation and N the number of round-trip times in the simulation.

A PCC flow sending at a higher than fair rate is occasionally required to temporarily stop sending, while it is allowed to send at that higher rate otherwise, effectively transmitting the same overall number of packets as a flow sending at the fair rate. A higher send rate means that more packets are transmitted per round-trip time. No packets are transmitted in the round-trip times during off times, so fewer round-trip times are needed to transmit the same number of packets as in the fair rate case.

Obviously, when more packets are transmitted in a single round-trip time, the statistical likelihood of multiple losses in one round-trip time increases. By definition of the loss event rate, multiple losses in a single round-trip time are treated as a single loss event. Therefore, with the same number of losses as in the fair case, the number of loss events can be expected to be smaller. Figure 6.8 shows a plot of Equation 6.3 with $T_{SIM} \cdot r_{NA}$ = the overall number of packets sent being constant, and varying the number of round-trip times N[3] caused by the varying send rate (in addition, the network situation from the simulation set resulting in Figure 6.7 is assumed; in particular, the loss rate is set to 1%).

The discrepancy between the very slight decrease of the expected loss event rate and the comparatively strong decrease in the simulation (even in the face of an increase in loss rate) is attributable to the limited amount of multiplexing in the simulation, which causes packet loss to be correlated to a significant extent. Correlation intensifies the noted effect, since loss bursts are likely to account for fewer loss events if the number of packets transmitted in one round-trip time is large.

So the situation is this: a higher send rate results in a lower loss event rate measurement, which

[3]Only “active” round-trip times where at least one packet is transmitted are counted, since no losses can occur in others.

in turn causes a higher TCP-friendly rate estimate and therefore increasing aggressiveness of the PCC flow. Note that if a substantial amount of bandwidth is at any time taken up by PCC flows, the observed mathematical effect is dimmed by the increase in loss rate that this aggressiveness causes. This is the case when an overly aggressive behavior of PCC would be most dangerous to the network in terms of congestion collapse. Another hopeful sign for the deployment of PCC in real-life networks comes from Figure 6.8, which shows that the effect described in this section is much slighter when packet loss is closer to random. This assumption is supported by simulation sets performed with varying degrees of multiplexing.

However, with some correlation and at very high PCC send rates, the effect of a lower loss event rate can cause PCC flows to behave unfairly to a significant amount. Figure 6.9 shows that PCC flows at high send rates use up to 80% of the bandwidth, considerably discriminating TCP flows. On the other hand, it must be said that the send rates used in 6.9 are extreme cases that would not be used in any sensible PCC-based applications. A send rate of more than five times the TCP-friendly rate results in a PCC flow being off on average four fifths of the time, the remaining fifth being largely used for short periods of protected time. Hardly any application is conceivable that would profit from such behavior. In the range of up to three times the TCP-friendly rate, which can be seen as the maximum reasonable PCC send rate, PCC share of bandwidth is still good.

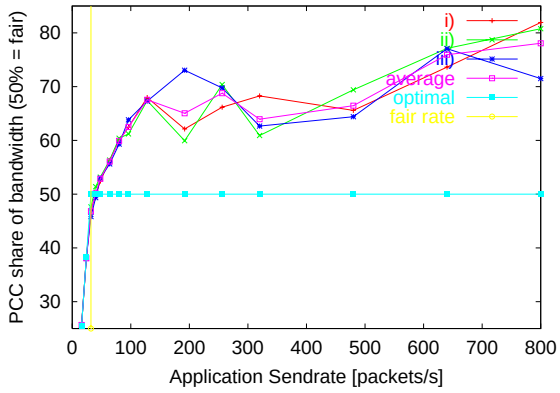

Figure 6.9: Fairness when Varying Send Rate up to 25 Times the Fair Rate

The question arises why PCC share of bandwidth is not larger than 50% already at slightly above the fair rate, even when disregarding the effect described in Section 6.4. The answer is of course that TCP flows also exhibit varying send rates, sending more than average packets in some round-trip times, while sending fewer or even none at all (during exponential back-off) in others. Therefore, the loss event rate experienced by TCP flows is also lower than the expected loss event rate given in 6.8, which assumes a constant bit-rate (CBR) flow. Sending at around the TCP-friendly rate, therefore rarely stopping transmission, PCC flows send more evenly than TCP flows. With a more even send rate, PCC flows measure a loss event rate that is closer to the value expected for CBR flows and therefore higher than that of TCP flows. Therefore, PCC under-estimates the TCP-friendly rate and behaves slightly too passively. This goes a long way in explaining the difference of 5% to the optimum value in Figure 6.6 even when a large number of loss event samples is used.

An interesting side note is that since the mechanism used to measure loss event rate in PCC is taken from TFRC, it is not unlikely that TFRC too suffers from this problem. In fact, since TFRC was designed to achieve a smoother send rate than TCP, it is quite likely that loss event rate is

over-estimated, much in the same way as it is for a PCC flow sending at near-TCP-friendly rate. However, the fundamental difference in approach that makes each single TFRC flow adapt its send rate to the TCP-friendly estimate lets this over-estimation of the loss event rate remain at more or less the same, relatively low percentage in all situations. A TFRC flow can never send at many times the fair rate in the way a PCC flow can (while on). Still, Jörg Widmer of the TFRC development team agrees that it might be interesting to run a series of tests to see if this is indeed an effect visible in TFRC.

6.5.1 A Quick and Dirty Solution

We implemented the following adjustment mechanism to alleviate the problem of PCC aggressiveness increasing with application send rate. It is based on the assumption of random packet loss and uses Formula 6.3 for the expected average loss event rate given in [18].

```
//compute TCP-friendly rate
tcprate = p_to_b(loss, rtt_, 4*rtt_, psize_, bval_);
if (adjust_rate) { //rate adjustment optional for application
  //use model from Ramesh and Rhee 1999 to compute expected loss
  //event rate using (erroneous) TCP-friendly rate as sendrate
  loss = (1 - pow((1-lossrate),(int)(rtt_*tcprate/psize_)))
         / (rtt_*tcprate/psize_);
  //re-compute TCP-friendly rate with this loss event rate estimate
  tcprate = p_to_b(loss, rtt_, 4*rtt_, psize_, bval_);
}
```

The idea is to use the estimate for the TCP-friendly rate that is based on the erroneous loss event rate measurement (but still relatively close to the actual value) to compute the average loss event rate expected for an application sending at that estimated rate, then to use this value to re-compute the TCP-friendly rate. This process could be repeated to achieve more exact results, but a number of numerical examples have shown that that would achieve little additional precision.

The mechanism has shown to be successful in a small simulation set that induced actual random loss by using an error model provided by ns-2, while setting bandwidth and queue sizes large enough to ensure that no significant number of possibly correlated losses occur due to packets being dropped from queues. A comparison of this simulation set using the mechanism in one case and not using it in the other is shown in Figure 6.10. On the other hand, when used in a scenario with a large number of correlated losses, the method naturally performs very badly. Since correlated losses lead to loss event rates that are considerably lower than the one expected with random loss, the high loss event rate computed by our mechanism heavily discriminates PCC flows (see Figure 6.11).

Therefore, this method can only be recommended if (close to) random loss can be assumed for a given network. Further work will be necessary to find a way to adjust the measured loss event rate taking into account correlated losses.

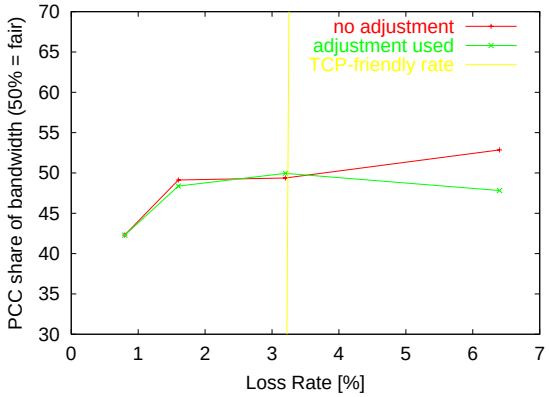

Figure 6.10: Share of Bandwidth with Varying Loss Rate (Random Loss), with and without Use of Adjustment Mechanism

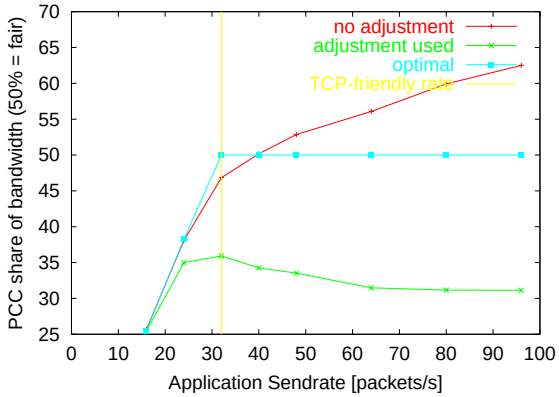

Figure 6.11: Share of Bandwidth with Varying Application Send Rate (Correlated Losses), with and without Use of Adjustment Mechanism

6.6 PCC Throughput

Since the share of bandwidth is defined on the average throughput of PCC and TCP flows, the throughput of PCC flows is similar to that of competing TCP flows when the share of bandwidth is 50%. Packet loss rates in a scenario with correlated losses have shown to be slightly higher for PCC flows than for TCP flows.

Another thing is how PCC behaves in terms of throughput in a scenario where the bottleneck link is shared by PCC flows only. In the optimal case, the average absolute throughput should be close to the fair rate independently of the application send rate. What we find however in the simulation set displayed in Figure 6.12 is that while throughput is optimal up to the fair send rate, it decreases notably with higher rates.

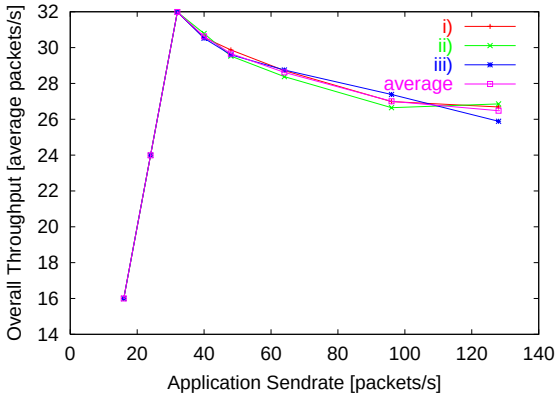

Figure 6.12: Absolute Throughput of an Average PCC Flow when Varying Application Send Rate

It is no surprise that throughput is optimal (all flows can continually transmit at their application send rate) up to the fair rate. While at fair rate there are some losses caused by slight variations in send rate, the number of resulting loss events is too small to ever yield a TCP-friendly rate low enough to cause the flow to stop.

On the other hand, at higher send rates, some loss is unavoidable, since this is the only valid local indicator of congestion available to the congestion control mechanism. The decrease of throughput with increasing send rate may be explained by the statistical likelihood that more than the fair number of flows are occasionally transmitting data. Such situations lead to high loss, which is not fully offset by situations where fewer than the fair number of flows are transmitting. This effect gets worse with increasing send rate, since loss in such situations becomes more drastic. Also, high loss rates result in higher probabilities to stop, so situations where fewer than fair PCC flows are transmitting may become more frequent. There may be a feedback problem here, since a large number of flows being turned off at high congestion would again cause high congestion if all restarted after the same T_{OFF}. A possible solution to this could be the introduction of a random element in the determination of the time a flow should remain off.

Note again however that even in this extreme scenario, the behavior of PCC with send rates in the range of up to three times the fair rate is still acceptable.

The above observations are supported by the following 3D plots of a simulation set varying the number of PCC flows and the number of TCP flows. The plots of the relative (for better comparison) PCC throughput also show that throughput is in the expected range when approximately one-fifth of all flows or more are TCP flows.

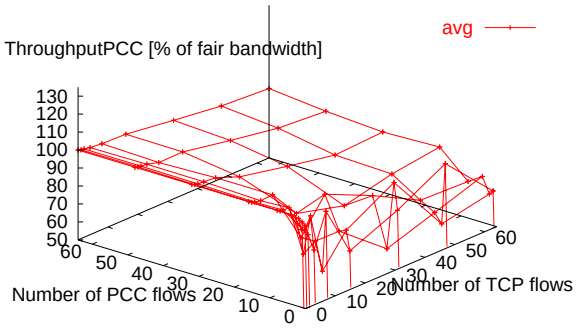

Figure 6.13: PCC Throughput at the Fair Rate Varying with the Number of PCC and TCP Flows

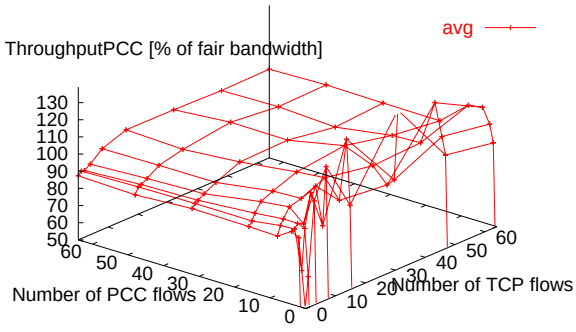

Figure 6.14: PCC Throughput at Twice the Fair Rate Varying with the Number of PCC and TCP Flows

The high level of variation for small numbers of PCC flows is a measurement problem. With few PCC flows, few measurements are available for PCC throughput. The fair bandwidth is defined as $\frac{N_{PCC}}{N_{PCC}+N_{TCP}} \cdot bandwidth$, where N_X is the number of flows using protocol X.

The following plots of the PCC share of bandwidth measured in the same simulation set also support much of what has been said before. Most importantly, they show that PCC fairness towards TCP does not depend on the ratio of the number of PCC flows to the number of TCP flows, as long as there are not too few TCP flows on the link.

The drastic decrease in PCC share for very few TCP flows in Figure 6.16 is a logical result of the decreasing PCC throughput for high send rates and mostly PCC flows explained at the beginning of this section, low multiplexing, and TCP's property of quickly searching out free bandwidth. When

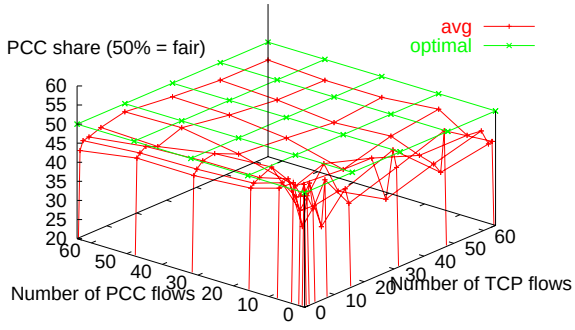

Figure 6.15: PCC Share of Bandwidth at the Fair Rate Varying with the Number of PCC and TCP Flows

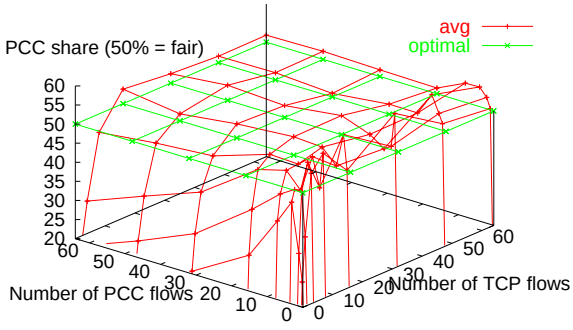

Figure 6.16: PCC Share of Bandwidth at Twice the Fair Rate Varying with the Number of PCC and TCP Flows

PCC throughput decreases, the TCP flows quickly take hold of any bandwidth freed by this. Since there are very few TCP flows, each single flow gets a relatively high amount of additional average throughput. The PCC share of bandwidth, comparing the average throughputs of PCC and TCP flows, therefore deceases rapidly.

6.7 Intra-Protocol Fairness

For send rates up to the TCP-friendly rate, PCC intra-protocol fairness has shown to be very close to one, since all flows are allowed to transmit (almost) all of the time, so each flow's throughput is almost the same. At send rates above the TCP-friendly rate, intra-protocol fairness decreases, since here flows are occasionally turned off.

In the scenario with competing TCP flows, the decrease is slight, with intra-protocol fairness remaining at good values above 0.96 for all of the most relevant cases (Figure 6.17). In the (problematic) scenario of only PCC flows, intra-protocol fairness decreases more rapidly (Figure 6.18).

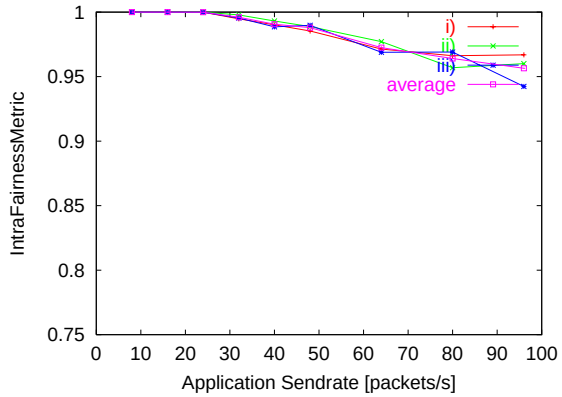

Figure 6.17: Intra-Protocol Fairness with Competing TCP Flows and Varying Send Rate

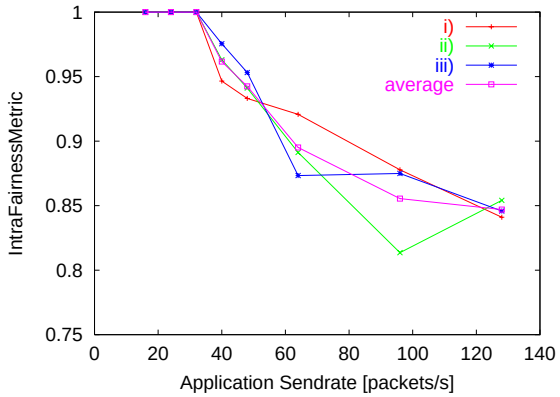

Figure 6.18: Intra-Protocol Fairness with only PCC Flows and Varying Send Rate

6.8 PCC Behavior at Different Round-Trip Delays

We have tested PCC at a wide range of average round-trip delays, ranging from values as low as four milliseconds which are typical for local area networks (LANs) to up to 1.6 seconds, which is an extreme value even for global wide area networks such as the Internet. Typical round-trip times in the Internet are between 50 and 400 milliseconds.

Note that the round-trip delay (RTD) comprises only the delay on the links, while the round-trip time (RTT) is also added to by queuing delays. This is most relevant at low RTDs, where round-trip time can be more than twice the RTD, while at the high RTD of 1.6 seconds, RTT in the simulation was about 1.8 seconds.

The PCC implementation depends on the RTT only as an input parameter to the TCP formula. As long as that formula accurately models TCP at all RTTs, PCC should be expected to perform well. Indeed, Figure 6.19 shows that the PCC share of bandwidth is in the expected range up to a round-trip delay of 800 milliseconds. The expected range is between 40% and 50%, because the fair rate is again used as the application send rate in this simulation set, so PCC's share is lowered by the effects described in prior sections. Note that a logarithmic scale is used for this plot, since exponentially increasing RTD values were employed.

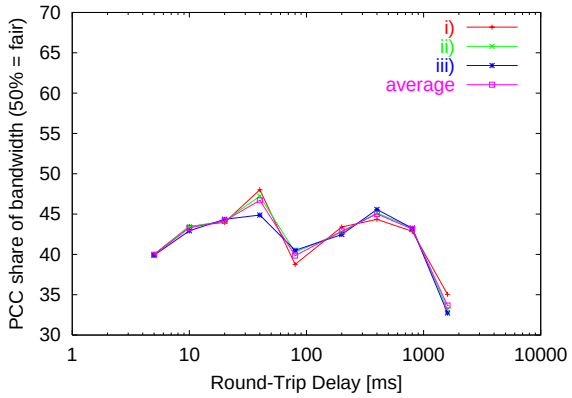

Figure 6.19: PCC Share of Bandwidth when Varying the Round-Trip Delay

However, at the high RTD of 1.6 seconds, PCC suffers considerably. We therefore assume that the TCP formula used by this implementation of PCC under-estimates the TCP-friendly rate at very high values for RTT.

6.9 Further Findings

6.9.1 Queue Types

It turns out that PCC Fairness improves when per-flow scheduling such as RED is used. Figure 6.20 shows that with RED queues, the PCC share of bandwidth at the fair rate is close to 50%, and the increase in aggressiveness is held considerably lower. However, the increase is still obvious - RED

queues alone do not solve the loss event rate estimation problem described in Section 6.5.

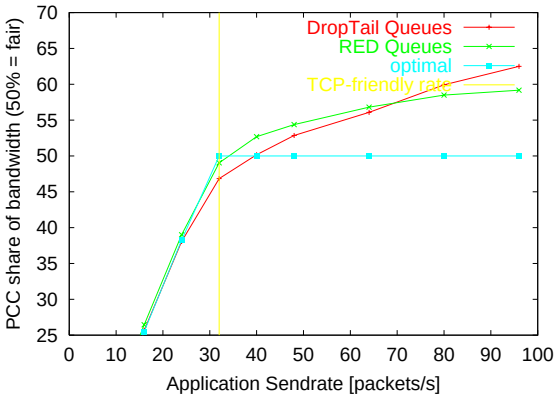

Figure 6.20: PCC Share of Bandwidth with Different Queue Types and Varying Send Rate

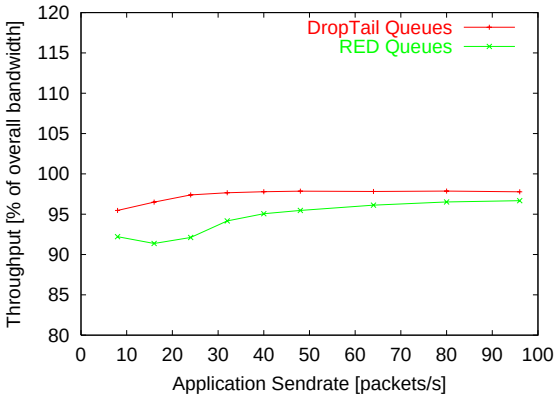

Figure 6.21: Overall Throughput with Different Queue Types and Varying Send Rate

More importantly, Figure 6.21 shows that the improvement in fairness comes at the cost of lower overall throughput, which is caused by the lower PCC throughput, from which TCP throughput hardly profits at all. Therefore, the use of RED queues does not seem to bring any real improvement in terms of TCP-friendliness.

6.9.2 Influence of Absolute Send Rate

The effects of a varying PCC application send rate with constant bandwidth (and thereby a constant fair rate, as the number of flows traversing the bottleneck link is not altered) have been studied in previous sections. The simulation set shown in Figures 6.22 and 6.23 varies both the bottleneck bandwidth and the application send rate. The vertical plane in 6.22 indicates the points where the send rate is the fair rate in relation to the bandwidth.

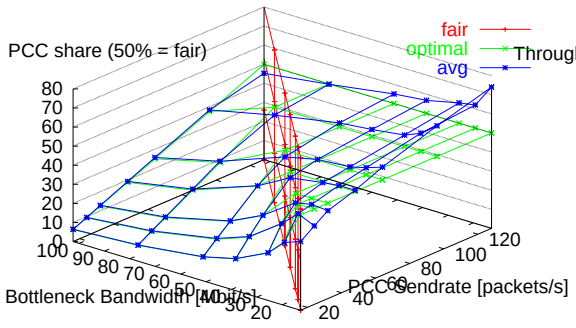

Figure 6.22: PCC Share of Bandwidth with Varying Send Rate and Bandwidth

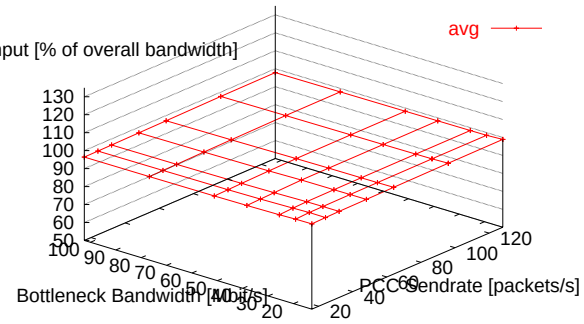

Figure 6.23: Overall Throughput with Varying Send Rate and Bandwidth

As expected, the absolute send rate does not have a large effect on PCC fairness. With send rates

below the fair rate, fairness is close to optimal, while the behavior beyond the fair rate is close to that described in Section 6.3 for any bandwidth. PCC share of bandwidth is highest in the right corner, since this is where the send rate is largest compared to the fair rate (at this point send rate is eight times the fair rate).

Overall throughput also is only marginally affected by the absolute PCC send rate. Throughput is slightly higher when PCC send rate is high relative to the fair rate.

6.9.3 PCC Behavior in an Unstable Environment

All simulation sets discussed so far have been performed in a very stable network environment where only random effects and TCP's saw-tooth behavior give any variation to the loss rate. This is not an altogether realistic setting. In real-life networks such as the Internet, the loss rate regularly undergoes changes, when some (large) flows start using the bottleneck link or others stop doing so. Drastic changes to the situation on a link can happen for example when a heavily used router decides to switch traffic to this link instead of some other that was used before.

To test PCC under such less stable conditions, a simulation set was prepared where random loss on the bottleneck link was occasionally changed from 1% to 10% for ten or fifty seconds, then back to 1%.

The expected behavior of TCP in such a situation is to adapt the average send rate of each single flow to the new situation almost immediately. The expected behavior for PCC is to adapt the average send rate of all flows together to a worsening situation within the time between two experiments, T_{EXP}. Adaptation of PCC to an improved situation on the link is expected to be slower, since a flow that stopped has to remain off for T_{OFF}, in this case 60 seconds.

Figure 6.24 confirms these expectations. It shows the average throughput over periods of ten seconds for TCP and PCC flows. PCC send rate is 128 packets/s, and thus slightly above the fair rate for a loss rate of 1%, and far above the fair rate for 10%. Figure 6.25 shows the same plot with the send rates of a single TCP and a single PCC flow superimposed.

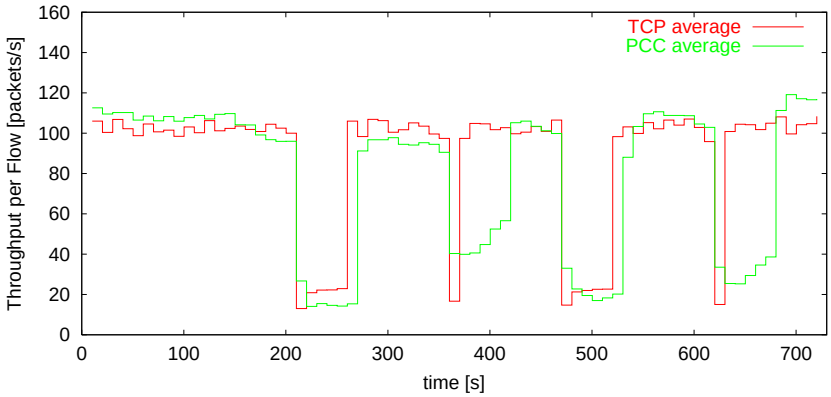

Figure 6.24: PCC and TCP Average Throughput over Time, with Loss Bursts

Note in Figure 6.24 that in the cases of short periods of high loss, the average PCC throughput

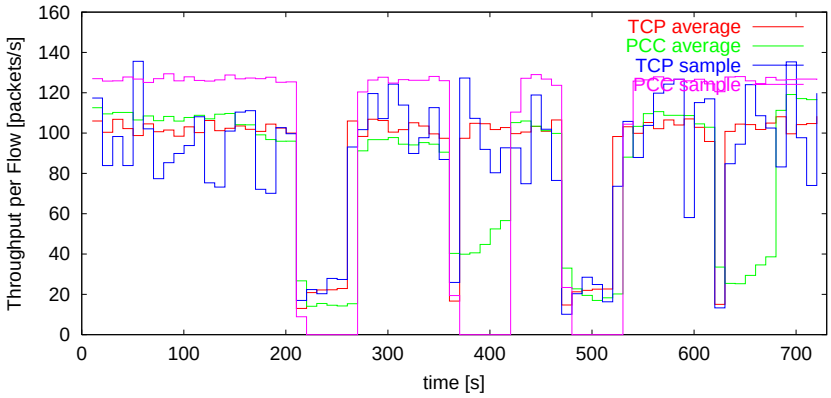

Figure 6.25: PCC and TCP Average and Single Flow Throughput over Time, with Loss Bursts

increases already before the off time ends for those flows that stopped during the loss burst. This is due to the fact that those flows that were already off at the time of the loss burst are starting up again. In the cases of long periods of hogh loss, this effect is offset by other flows exiting.

Note in Figure 6.25 that the sample PCC flow stops during the first three loss bursts but not during the fourth. At that time, it is one of the (few) PCC flows transmitting at the full application send rate.

6.9.4 Protected Time

At standard Internet-like network conditions such as given with the parameters in Table 6.1, the average protected time T_{PROT} is usually in the range of up to 30 seconds. T_{PROT} usually ends when a sufficient number of loss events have occurred (round-trip time measurements are mostly reliable prior to that). Therefore, protected time decreases when packets are transmitted at higher rates, since, the loss rate being equal, this means losses and therefore loss events at a higher frequency (in time). This can be seen in Figure 6.26.

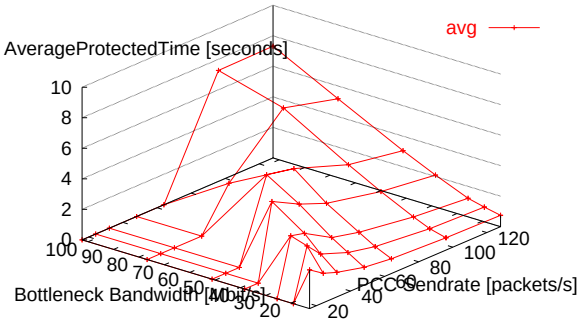

Figure 6.26: Protected Time with Varying Send Rate and Bandwidth

Note that protected time is given as zero when the send rate is well below the fair rate, in order to

signify that in this case, no protected time other than the one at start-up takes place, since no PCC flow has to stop transmission at such a rate. The protected time for such flows at start-up ends when $T_{PROT,MAX}$ is reached.

6.9.5 Application Parameters

Loss Events for Protected Time: In PCC, protected time usually ends when a preset number of loss events have occurred that are expected to be sufficient to give a relatively reliable loss event rate measurement. The round-trip time measurement, which is the other reason for protected time, is usually reliable before that. This number of loss events can be set by the application.

In a relatively stable environment, this parameter is almost irrelevant, as demonstrated in Figure 6.27, which shows a simulation set varying the number of necessary loss events. Here, even a single loss event appears to provide a measurement sufficiently accurate for the PCC scheme to achieve good fairness towards TCP. This is interesting, since experiments have shown that single loss intervals vary considerably (by about +/-70%) even in the given, relatively stable environment.

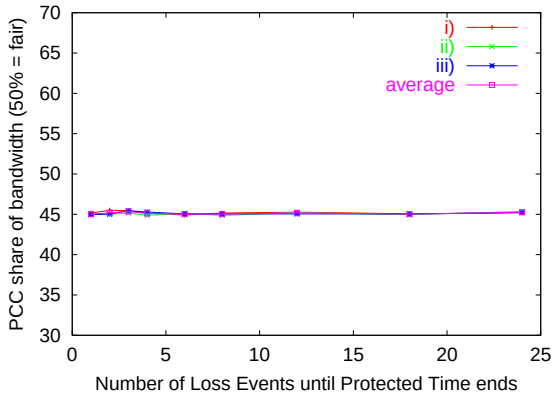

Figure 6.27: PCC Share of Bandwidth with a Varying Number of Loss Events for Protected Time

The number of loss events for protected time can be expected to become more important in a less stable environment such as the Internet. However, the results of this simulation set show that even then it may not need to be very large. Most simulation sets presented in this thesis used a value of three.

T_{OFF} *and* T_{EXP}: In an environment with a completely stable loss event rate, a random experiment is only performed once per T_{OFF} period, and all other experiments in continuous evaluation result in a probability to remain on of exactly one. In such an environment, T_{EXP} could be chosen as any value up to T_{OFF}.

However, in an environment with some variation in loss event rate, the choice of T_{EXP} does have an effect on both PCC fairness and on the responsiveness of TCP flows. Responsiveness is certainly the main issue. If network conditions vary quickly and strongly, a good responsiveness is crucial in order to avoid unfairness towards other flows. Therefore, T_{EXP} needs to be small. However, for reasons detailed in Section 6.4, a large number of experiments within one T_{OFF} period results in

a too passive behavior of PCC flows, especially when send rate is close to the TCP-friendly rate. Therefore, the relation of T_{EXP} to T_{OFF} needs to be chosen carefully to give a good balance of responsiveness and PCC share of bandwidth.

The simulation sets shown in Figures 6.28 and 6.29 were performed in the relatively stable environment of the standard scenario. While the relative stability of the loss rate should keep PCC passiveness in a reasonable range, the use of the fair rate results in a stronger passiveness than would occur at other send rates in the same environment.

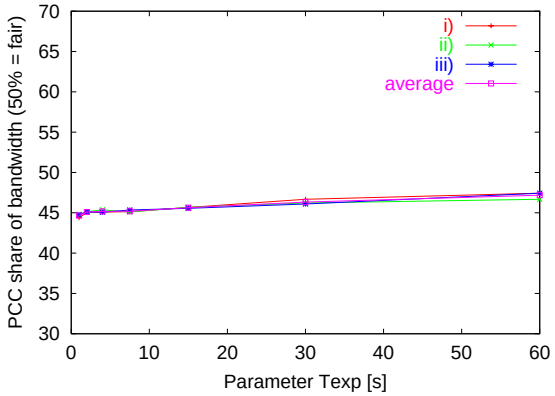

Figure 6.28: PCC Share of Bandwidth with Varying T_{EXP} and Constant T_{OFF}

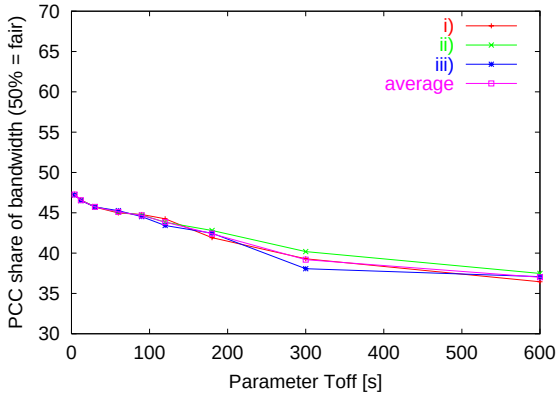

Figure 6.29: PCC Share of Bandwidth with Varying T_{OFF} and Constant T_{EXP}

The simulation set in Figure 6.28 varies T_{EXP} while keeping T_{OFF} constant at 60 seconds. The ratio of T_{OFF} to T_{EXP} is largest when an experiment is performed every second. With 60 experiments per off time, PCC share of bandwidth is approximately 2% lower than at a ratio of one to one.

Figure 6.29 shows the effect of varying T_{OFF} with a constant value of two seconds for T_{EXP}. With $T_{OFF} = 600s$, 300 experiments are performed per off time. This results in a PCC share of bandwidth that is about 10% lower than at the one to one ratio.

These simulation sets along with prior findings demonstrate that a PCC flow can be considerably discriminated when the following properties are given:

- T_{OFF} is large and T_{EXP} is small
- loss event rate varies considerably
- the PCC flow sends at a rate close to the TCP-friendly rate

A small value of T_{EXP} is common, even necessary for an environment with high loss event rate variation. On the other hand, if loss event rate varies considerably, it can be expected that the TCP-friendly rate does the same. Therefore, it is highly improbable that the PCC send rate stays very close to the TCP-friendly rate in most cases. Such extreme combinations should thereby be very rare.

Packet Size: In all previously described simulation sets, equal data packet sizes were used for PCC

and TCP. It may not be obvious that this is the only feasible setting, since the PCC implementation uses bytes/s (as opposed to packets/s) as a measure of send rate internally. However, Figures 6.30 and 6.31 show that for smaller packet sizes, PCC receives a lower than fair share of bandwidth, while larger packet sizes result in heavily increased aggressiveness in the case of PCC send rates above the fair rate.

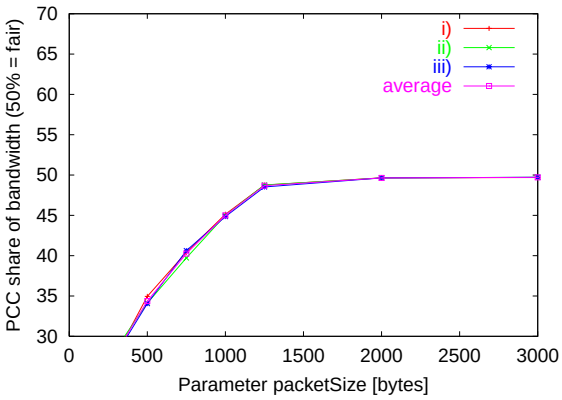

Figure 6.30: PCC Share of Bandwidth with Varying Packet Size at the Fair Rate

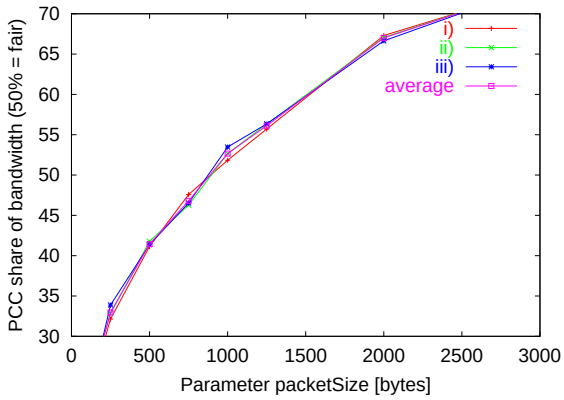

Figure 6.31: PCC Share of Bandwidth with Varying Packet Size at Twice the Fair Rate

In real-life networks where TCP packet size is not known it is therefore necessary to use a mechanism such as path MTU discovery [19] in order to determine this value, as is also proposed for other rate-based congestion control schemes such as TFRC.

Chapter 7

Discussion of PCC Application Parameters

A number of protocol parameters can be set by applications using PCC in order to account for specific application demands and network conditions. The following discussion of these parameters is meant to give some guidelines to future application developers employing the PCC congestion control scheme.

Table 7.1 at the end of the chapter proposes ranges of parameter values to be used in certain exemplary network conditions. These propositions are based on the experiences gained from extensive experimentation.

7.1 Off Time and Re-Experiment Interval

The time T_{OFF} that a PCC flow remains off when an experiment fails should be set according to the needs of the application. While some applications may depend on being able to send at least some data at relatively small intervals, it may be more favorable to others to transmit for long intervals and then to remain off for longer times as well.

For reasons detailed in Section 4.4.3, if the application is not robust against extensions of the actual off time, T_{OFF} should be chosen such that with common values for protected time T_{PROT}, TCP-friendly rate r_{TCP}, and application send rate r_{NA}, $\frac{T_{OFF}}{T_{PROT}}$ is greater or equal to $\frac{r_{NA}}{r_{TCP}} - 1$. This is usually not a strong limitation since protected time can be bounded by the use of $T_{PROT,MAX}$.

The re-experiment interval T_{EXP} determines the responsiveness of the flow to sudden increase in congestion. It should be chosen small (in the range of one second) if sudden sharp increases in congestion are likely to occur. On the other hand, a small interval and therefore more experiments in the same time increases the CPU time used up by the PCC scheme.

While it may be tempting to use small T_{EXP} to increase responsiveness even when a large off time is used, this is not advisable, since we have shown in Section 6.4 that a large number of experiments within T_{OFF} increases the chance that more flows than necessary stop transmission,

and that therefore PCC flows get less than their fair share of the bandwidth. This is especially true when variation in network conditions is high. With a large off time and a high degree of variation, the choice of T_{EXP} is therefore a trade-off between responsiveness and less self-discrimination.

7.2 Parameters Determining the Length of Protected Time

When a flow (re-)starts transmission, it is protected from having to stop until a certain number of round-trip time samples have been taken and a certain number of loss events have occurred, or until a time of $T_{PROT,MAX}$ is reached.

Since the round-trip time measurement is usually quite reliable after a few samples have been taken, it is not necessary to choose the number of round-trip time samples to be very large. This parameter is only relevant in determining T_{PROT} in the case of long round-trip times and high loss rates, in which three to five should be enough to determine a measurement reliable enough for the purposes of PCC.

The duration of protected time is more frequently determined by the number of loss events deemed necessary for a good loss event rate measurement. When variation in loss event rate is low, this parameter can be chosen very small. In experiments with low variation, even a value of one provided good results. When variation is higher, more samples are needed to achieve reliable results.

Protected time can be limited to a maximum of $T_{PROT,MAX}$. This is advisable if extensions of the preset off time are not desirable. This value should however be set with care, since cutting off protected time early leads to higher variation, which in turn leads to under-allocation of PCC bandwidth. Therefore, $T_{PROT,MAX}$ should be set such that in most cases, enough measurements are taken to arrive at an acceptable estimate of r_{TCP}, especially when variation in loss event rate is high. For more discussion on this subject, see Section 4.4.4.

7.3 Parameters Determining Smoothness of Measurements

Whenever multiple samples are used to arrive at a smooth measurement, there is the trade-off between reducing variation as much as possible and on the other hand keeping the measurement up-to-date. If too many samples are used, the influence of out-of-date values is too large, while smoothness suffers if too few are used. More samples should be used if short-term variation is high while medium-term variation is low, fewer should be used in the converse case.

In PCC, an exponentially weighted moving average (EWMA) is used to smooth the round-trip time measurement. A large weight for the current sample makes the measurement more up-to-date (but less smooth) and should therefore be used when short-term variation is usually low while some rapid, longer-lasting changes in round-trip time may occur. In our experiments, this parameter has proved to be of little relevance, good results have been achieved with weights between 0.05 and 0.5.

For smoothing the loss event rate measurement, PCC employs the Average Loss Interval method discussed in Section 5.4.2. The number of loss event samples N_{SAMP} used in this context can be set by the application. A value of 24 for this parameter has proved to achieve good results in

a setting with low medium-term variance, while it may be necessary to use a smaller value when sudden lasting decrease in the available bandwidth is frequent, in order to avoid unfairness towards other flows due to low responsiveness. However, the increase in variation of the loss event rate measurement and therefore of the TCP-friendly rate estimate resulting from a small N_{SAMP} leads to a more passive behavior of PCC, as has been discussed at length in Section 6.4. N_{SAMP} should therefore be set only as low as necessary to achieve acceptable responsiveness.

An application using PCC can chose whether to use history discounting [16] or not. It is usually advisable to use history discounting, since it improves both smoothness and responsiveness (to sudden long absence of congestion) of the loss event rate measurement.

7.4 Packet Size

As has been noted in Section 6.9.5, the size of PCC data packets should be the same as that of the average TCP data packet. It is the responsibility of the application to determine the TCP data packet size, e.g., by using path MTU discovery [19], and to set the PCC packet size accordingly.

medium-term congestion variation	low		high	
congestion	low	high	low	high
PCC Packet Size	same as actual TCP packet size in the network			
RTT EWMA Weight	0.1 - 0.3		0.2 - 0.4	
N_{SAMP}	16 - 24	24 - 36	8 - 16	16 - 24
History Discounting	used			
RTTs for Protected Time	4 - 8	3 - 5	4 - 8	3 - 5
Loss Events for Protected Time	1 - 3	2 - 4	3 - 5	4 - 6
$T_{PROT,MAX}$	>=20sec	>=10sec	>=30sec	>=20sec
T_{OFF}	determined by user expectations			
T_{EXP}	ca. 1/10 of T_{OFF}		1 - 3 seconds	

Table 7.1: Proposed PCC Application Parameter Values for Different Network Conditions

Note: The choice of $T_{PROT,MAX}$ depends mostly on the type of application. If the application expects off times to remain constant, $T_{PROT,MAX}$ should be chosen as discussed in Section 4.4.4.

Chapter 8

Possible Extensions to the PCC Scheme

A number of possible extensions to the PCC congestion control scheme have been investigated, though neither implemented nor experimentally evaluated as of yet. The examples given in this section are therefore potential subjects of further research.

8.1 Probe While Off

PCC flows may receive less expected average bandwidth than competing TCP flows, since a flow that has been turned off will switch to on only after T_{OFF} seconds, even if congestion improves beforehand. This may be problematic, especially if T_{OFF} is large. In order to increase PCC's fairness, flows that are off could monitor network congestion by sending probe packets at a low data rate from the sender to the receiver. The data rate r_{OFF} produced by the probe packets needs to be taken into account in Equations 4.2 and 4.6 by adding $(1 - p_{ON}) \cdot T_{OFF} \cdot r_{OFF}$ to the total amount of bandwidth used by a PCC flow.

If the loss rate and the round-trip time of the probe packets signal that r_{TCP} has improved sufficiently, a flow that has been turned off may be turned on again immediately, without waiting for the remainder of the off time to pass, and without performing an initialization step. This may be done only if, under the new network conditions, all experiments that were performed within the last T_{OFF} (before the flow was turned off) would have been positive. If the congestion situation gets worse later on, it must be checked if any of the experiments in that time would have failed. If this is the case, the flow must be turned off again. Only after the last entry that was included in set P before the flow was turned off has timed out, may the flow again resume normal behavior.

The biggest problem with this method is the amount of bandwidth that is used up by the probe packets. While the round-trip time can be determined with reasonable accuracy with just a few packets, many more are necessary to measure the loss rate. In order to measure a loss rate of 1%, for example, a minimum of 100 packets need to be transmitted, about 300 to achieve at least reasonable accuracy. If measurement values older than 30 seconds are considered out-of-date, this means 10 probe packets per second, which equals 80 kbit/s if the packet size is 1000 bytes. Smaller probe packets could be used to reduce the necessary data rate, but this would lead to an under-estimation of the loss rate, since smaller packets are less likely to be dropped.

The current version of PCC does not include Probe While Off. Apart from the waste of bandwidth, Probe While Off can lead to frequent changes between the states on and off which would be distracting for the user of the application. Probe While Off may be included in a later version of PCC as an option that can be turned on by the application. Furthermore, it could be helpful to add a threshold to Probe While Off, so that the flow is turned on again only if the available bandwidth increases significantly. This could improve the stability of the flow.

8.2 Probe Before On

In PCC the flow is turned on upon initialization. This has two drawbacks. First, it violates the idea of exploring the available bandwidth as in TCP slowstart. Second, the flow may be turned off immediately after the initialization is complete, so that the user perceives only a brief moment where the application seems to work, before it is turned off. An alternative would be to send probe packets at an increasing rate before deciding whether to turn the flow on or not. Only after the parameters have been measured and the random experiment has succeeded will real data for the flow be transmitted. The drawback of this method is that bandwidth is wasted by probe packets and that the initial start of a flow is delayed by T_{PROT}, with T_{PROT} being larger than with the normal method, since the rate starts off low.

8.3 Forward Error Correction

Since applications employing non-adaptable flows frequently have to obey real-time constraints, they benefit from forward error correction to compensate for packet loss. However, packet loss is typically a signal for congestion. Therefore it has long been considered inacceptable to compensate for congestion-based packet loss by increasing the data rate of a flow with redundant information for forward error correction. PCC does however enable the use of forward error correction in an appropriate way.

PCC's support for the usage of forward error correction is straight forward: when an application decides to employ forward error correction, the new r_{NA} is simply set to the rate of the flow including the forward correction information. From the perspective of PCC this is equivalent to an application increasing its sending rate and thus needs no special treatment. Increasing r_{NA} results in an appropriate decrease of p_{ON}, therefore this method is fair to competing streams.

8.4 Multicast PCC

The PCC congestion control scheme can be extended from the unicast case described in this thesis to the multicast domain. Given the efficiency savings of multicast, different notions of TCP-friendliness are possible in this domain. However, we take the conservative position formulated in [20], that "a multicast flow is acceptable if it achieves no greater medium-term (expected average) throughput to any receiver in the multicast group than would be achieved by a TCP flow between the multicast sender and that receiver."

Since different receivers may experience different network conditions, the estimation of the TCP-friendly rate and the calculation of p_{ON} must be performed at the receivers when using multicast. Two different general approaches are possible in how to use this value to achieve multicast TCP-friendliness.

8.4.1 Sender-Driven Multicast PCC

In sender-driven multicast PCC, the probability values are reported back to the sender, which then decides whether to turn the whole multicast flow off for T_{OFF}, using the lowest of the reported values. Probability values greater one do not need to be reported. A method to further suppress unnecessary feedback in order to avoid feedback implosion for large receiver sets can be found in [20]. The same paper also gives ideas on how to ensure that some feedback always reaches the sender so that the absence of feedback can be attributed to strong congestion on the feedback path.

The drawback of sender-driven multicast PCC is that decisions can only be made for the whole multicast flow. If congestion in some outer branch of the multicast tree is high, the sender has to stop often enough to ensure multicast TCP-friendliness for the receivers on that branch. Other receivers in the multicast tree experiencing much less congestion suffer from the same reduced expected average throughput.

8.4.2 Receiver-Driven Multicast PCC

In receiver-driven multicast PCC, the probability values are not reported to the sender but used for local decisions. If for any receiver the experiment using the locally calculated probability value fails, that receiver temporarily leaves the multicast group, before re-joining it after T_{OFF}.

When all receivers behind a congested link leave the multicast group, congestion control is performed implicitly by the multicast routing protocol, which prunes this branch from the multicast tree until some receiver re-joins.

Obviously, the random numbers used in the experiments can not be generated locally. Otherwise, the probability that at least one receiver behind the congested link remains in the group at any time is very high, and the congestion control effect on that link is almost non-existent. Therefore, random numbers are periodically generated by the sender and distributed as data packet header information to all receivers.

While this method operates at a much finer granularity than the sender-driven method, it also has several drawbacks. For one thing, probability values calculated by several receivers behind the same bottleneck link may differ slightly due to random losses on links behind that bottleneck, or due to different settings for certain application parameters, such as the number of loss event samples. Therefore it is possible that with p_{ON} close to the random number, some receivers leave the multicast group while others stay in. In such a case, throughput is reduced for some receivers without achieving any positive effect for the bottleneck link. The second drawback is that receivers in the off state miss out on data that is transmitted to the others. While this is reasonable for example for most real-time applications such as live video transmission, other applications may need to employ sophisticated dynamic join and leave algorithms to ensure consistency.

8.4.3 TCP-Friendly Rate Estimation

Most methods for the estimation of the TCP-friendly rate, such as those used in TEAR and TFRC, require the knowledge of the current round-trip time. In multicast, feedback packets can not be passed from each receiver to the sender and echoed back as frequently as in the unicast case, as this would lead to feedback implosion.

Therefore, a scalable way is necessary to measure round-trip time. In the multicast extension to TFRC, TFMCC, this is done by initializing the receiver's round-trip time measurement using synchronized clocks (if available) and adjusting that measurement using the one-way delay calculated from data packet header information. In addition, receiver feedback is echoed infrequently to all but one special receiver, which is the receiver that currently reports the lowest rate, the so-called *current limiting receiver* (CLR). Since the CLR usually determines the allowed rate, its measurement needs to be the most exact, therefore its feedback is echoed more regularly. For a more detailed description of this method, see [20].

For sender-driven multicast PCC based on TFRC mechanisms for the TCP-friendly rate estimation, this method can be copied. The current limiting receiver here is the receiver reporting the lowest probability value. In the receiver-driven case, the notion of a CLR does not make sense, because there is no 'global' rate that could be limited by some single receiver. Instead, each receiver determines its own average receive rate by deciding locally whether to temporarily leave the multicast group. Other elements of the method developed for TFMCC, such as round-trip time measurement initialization and adjustment, are however applicable to the receiver-driven case as well.

8.5 Combining PCC with other Congestion Control Schemes

So far we have distinguished between flows that allow their rate to be adapted to any extent and flows that do not allow any adaption at all, the so-called non-adaptable flows. It is not hard to imagine an intermediate class of flows, that allow their rate to be adapted up to a certain minimum, but cannot make use of any rate below that minimum. An example of such a "semi-adaptable" flow would be a live audio transmission that can be scaled in quality, while anything below a certain minimum quality would be considered unacceptable.

Such flows would profit from a combination of PCC and some other congestion control scheme that adapts the rate. As long as the TCP-friendly rate is greater or equal to the minimum acceptable rate, the rate is adapted. Once the TCP-friendly rate drops below the minimum, the send rate is set to the minimum and PCC is used to turn the flow off at appropriate times. It is advisable to use the same mechanism for estimating the TCP-friendly rate in both cases. The current PCC implementation could therefore be easiest combined with TFRC.

Chapter 9

Conclusion and Future Work

In this thesis we presented a congestion control scheme for non-adaptable flows. This type of flow carries data at the rate determined by the application. It cannot be adapted to the congestion situation of the network in any other way than suspending the entire flow. Existing congestion control approaches are therefore not viable for non-adaptable flows.

We proposed to perform congestion control for non-adaptable flows by suspending individual flows in such a way that the aggregation of all non-adaptable flows on a given link behaves in a TCP-friendly manner. The decision about suspending a given flow is made by means of random experiments, hence we chose the name Probabilistic Congestion Control or PCC for our approach.

In a series of simulations it has been shown that PCC displays a TCP-friendly behavior in a wide range of network situations. We also identified a number of possible extensions to the PCC scheme. The thesis shall conclude with a discussion of issues that will need to be investigated further in future work.

9.1 Future Work

Apart from investigating the merits of the extensions to PCC discussed in the last chapter, some amount of work is still necessary to further improve on PCC performance and to verify our simulation results under a greater array of network conditions.

9.1.1 Improving the Loss Event Rate Measurement

As we have seen in Chapter 6, the fairness of the PCC scheme is in certain cases harmed by false estimations of the TCP-friendly rate resulting from erroneous measurements of the loss event rate. These measurement errors are caused by the difference in smoothness of the rates of the average TCP and PCC flows, as was explained in Section 6.5.

One way to account for this difference in order to achieve better loss event rate measurements would be to find a heuristic function to adjust the measurement based on experimental findings. This

function would have to involve the ratio of the TCP-friendly rate (as estimated with the erroneous measurement) to the PCC flow's application send rate, as this determines how frequently the flow stops and therefore how smooth the effective rate is. However, more parameters will need to be considered, such as the correlatedness of loss events, the loss rate, and possibly the round-trip time.

Another very promising thought was brought forward by Martin Mauve when this thesis was nearly completed, so unfortunately there has not been enough time to verify it experimentally. Consider a PCC flow that sends at a rate considerably higher than the TCP-friendly rate when on. This flow will transmit more packets per round-trip time than a TCP flow, so due to a higher probability of multiple losses per round-trip time, the same number of losses for the same number of packets will result in fewer loss event, thereby a lower loss event rate.

The idea is now to artificially reduce the number of packets used for loss event rate measurement by only counting as many packets per round-trip time as would arrive at the estimated TCP-friendly rate, while ignoring all other packets, which count neither as losses nor as transmitted packets for the purpose of this measurement. Statistically, the subset of counted packets will experience the same loss rate as if all packets were used, but the probability of multiple losses within one round-trip time will be approximately the same as for a TCP flow. Therefore, the loss event rate should be much closer to that experienced by TCP flows, and the increase in aggressiveness with a higher ratio of PCC send rate to TCP-friendly rate that was noted in our experiments should not occur. On a side note, the number of packets counted per round-trip time in loss event rate measurement will need to be somewhat higher than with the TCP-friendly rate, to account for the fact that TCP itself does not send smoothly and therefore has an increased probability of multiple losses per round-trip time as well (as compared to a flow sending at a completely smooth rate). This however depends on how smooth the PCC send rate determined by the application actually is.

9.1.2 Randomizing Off Time

If many PCC flows use the same off time T_{OFF}, there is the danger that some synchronization of these flows may occur. This can happen if a large number of these flows exit at about the same time because of a sudden drastic decrease in available bandwidth. With the same T_{OFF}, these flows will also re-start at about the same time, impacting the congestion situation, thereby exiting again with slightly higher probability.

PCC fairness as well as throughput may therefore profit from some amount of randomization of T_{OFF}. Whether randomization is used or not and if so at what range should be up to the application, since some applications might not accept off times that differ much from the preset parameter. Some experimental evaluation is necessary to find out if randomized off times actually improve PCC performance, or if performance is on the contrary harmed by the additional element of variance.

9.1.3 "Deterministic Congestion Control"

The use of non-determinism in PCC is responsible for some amount of intra-protocol unfairness. Some flows will under the same conditions be forced to stop more frequently than others, though this effect will be less noticeable for longer run-times.

It is possible to devise a similar congestion control scheme that does not make use of non-determinism. Instead of calculating probability values and performing random experiments, the on time T_{ON} could be determined directly from Equation 9.2.

$$(T_{PROT} + T_{ON} + T_{OFF}) \cdot r_{TCP} = (T_{PROT} + T_{ON}) \cdot r_{NA} \tag{9.1}$$

$$\Leftrightarrow T_{ON} = \frac{(T_{PROT} + T_{OFF}) \cdot r_{TCP} - T_{PROT} \cdot r_{NA}}{r_{NA} - r_{TCP}} \tag{9.2}$$

As for PCC's continuous evaluation, T_{ON} can be regularly re-computed to account for changes in network condition. The flow stops when either it has been on for the on time that was last computed, or when a new time is computed that is smaller than the time the flow has been on.

While it would be interesting to implement such a deterministic congestion control scheme in order to see how well it actually performs, it should be reflected that for some applications, less regularity in the occurrence of off times, as produced by PCC, may be preferable. Also, synchronization might be more of a problem in the deterministic scheme.

9.1.4 Further Experimental Evaluation

Due to a limited amount of time, some simulations have not yet been performed that might provide interesting results towards a better understanding of PCC performance in real-life networks. Most notable among such simulations are

- **Complex topologies:** All simulations discussed in this thesis use the standard dumbbell topology of a number of senders on one side transmitting data across a single bottleneck link to a number of receivers on the other side. More complex topologies could involve several bottlenecks with flows traversing different numbers of these links.

- **Varying send rate:** The current version of PCC uses constant application send rates, as this was easiest to implement. For most applications, varying send rates are more realistic. This should be implemented and compared to the results provided in this thesis by means of re-running some of the simulation sets.

- **Different parameters across different flows:** In order to achieve significant average results, the same application parameters were used for all PCC flows in a simulation. A more realistic simulation would involve different values across different flows for application send rate, T_{OFF}, T_{EXP}, and other parameters.

- **Testing PCC against HTTP traffic:** In our simulations, we have compared PCC flows against TCP flows carrying unlimited bulk FTP data, since this makes simulation setup and evaluation easier. In the Internet, a large amount of TCP data is however transmitted by HTTP. It would therefore be interesting to test PCC flows against HTTP traffic.

- **Testing PCC against CBR and TCP traffic:** A significant amount of the traffic in today's Internet does not use any kind of congestion control mechanism at all. Some simulations should be performed pitching PCC flows against some constant bit-rate flows with no congestion control as well as some TCP flows, in order to see how PCC performs compared to TCP in such an environment.

9.1.5 Real-Life Implementation and Evaluation

Unless the above additional experiments indicate anything to the contrary, we believe that the PCC congestion control scheme as it is described in this thesis can be implemented for and tested in a real-life network such as the Internet without a significant risk of harming that network.

Appendix A

Calculation of Protected Time under Exemplary Network Conditions

The time T_{PROT} a (re-)starting flow needs to be protected from having to stop is the maximum of the time T_{LER} it takes to get a reliable measurement for the loss event rate and the time T_{RTT} necessary for a good estimate of the round-trip time, as was explained in Section 4.4.1.

A.1 Calculation of the Time Necessary for a Reliable LER-Measurement

With the number of loss events N_{LE} the application deems necessary for a reliable loss event rate measurement, the expected average of T_{LER} can be given as (compare Section 4.4.1)

$$T_{LER} = \frac{N_{LE} - \frac{1}{2}}{ler \cdot r_{NA}} \tag{A.1}$$

where ler is the actual loss event rate in the network.

It can be seen from Equation A.1 that T_{LER} is (almost) proportional to N_{LE} and inversely proportional to ler and r_{NA}. T_{LER} can get very large if the loss event rate is low. However, a low loss event rate also means that the TCP-friendly rate r_{TCP} is comparatively high. If r_{NA} is lower than r_{TCP}, the duration of protected time does not matter, since in such a case the flow will not be required to stop anyway. Therefore, only cases with $r_{NA} > r_{TCP}$ will be considered in the following.

Let N_{LE} be 3 (the value used in most of our simulation sets). Furthermore, consider a network situation with $ler = 0.01$ and $RTT = 100ms$. It follows from Equation 5.4 that r_{TCP} is roughly 112 packets per second (pps).

The relevant cases are therefore those with $r_{NA} > 112pps$. With this, we have $T_{LER} < \frac{2\frac{1}{2}}{0.01 \cdot 112} = 2.23sec$.

Table A.1 gives maximum relevant values for T_{LER} arrived at in the same manner using different values for ler and RTT.

	ler=0.001	**ler=0.01**	**ler=0.1**
RTT=100ms	6.51 sec	2.23 sec	1.41 sec
RTT=200ms	13.02 sec	4.46 sec	2.82 sec
RTT=400ms	26.04 sec	8.92 sec	5.64 sec

Table A.1: Exemplary Maximum Relevant Values for the Time to Arrive at a Reliable LER-Measurement

As can be seen from Table A.1, the maximum relevant T_{LER} is proportional to the round-trip time. This is due to the fact that r_{TCP} and therefore the minimum relevant r_{NA} is inversely proportional to the round-trip time (if a simple approximation to RTO is used that is a multiple of RTT).

Another result that can be drawn from Table A.1 is that (relevant) T_{LER} gets very large only under network conditions with a large round-trip time and at the same time a low loss event rate in the face of high application send rate. Since large round-trip time and loss event rate are in many cases both caused by full queues in intermediate systems, these situations can be expected to be rare.

A.2 Calculation of the Time Necessary for a Reliable RTT-Measurement

If one RTT-measurement value is taken per round-trip time, the time T_{RTT} necessary to arrive at a good estimate of the round-trip time is calculated simply as the product of RTT and the number of measurement values N_{RTT} considered necessary by the application for a reliable measurement. More than one measurement can be taken per RTT by using the one-way information provided with each data packet. In that case, T_{RTT} is much smaller.

With $N_{RTT} = 5$ (the value used in most of our simulation sets) and $RTT = 100ms$, T_{RTT} is 0.5sec. Other exemplary values are given in Table A.2.

	N_{RTT}**=2**	N_{RTT}**=5**	N_{RTT}**=10**
RTT=100ms	0.2 sec	0.5 sec	1.0 sec
RTT=200ms	0.4 sec	1.0 sec	2.0 sec
RTT=400ms	0.8 sec	2.0 sec	4.0 sec

Table A.2: Exemplary Values for the Time to Arrive at a Reliable RTT-Measurement

It can be seen from a comparison between Tables A.1 and A.2 that T_{RTT} is in most cases smaller than T_{LER}. Therefore, in most cases $T_{PROT} = T_{LER}$.

Appendix B

Installation Guide

In order to perform simulations with our implementation of PCC, the first requirement is to have a working installation of the network simulator ns-2. For the latest version of ns-2 and a detailed description of how to install it, see [14].

Once the simulator is installed, the following steps need to be performed to add PCC functionality to it:

1: Unzip and untar the code distribution package pcc.tar.gz. This should be available from the same source from which this thesis was obtained. The package also contains a file called *install.txt* that briefly lists the following steps.

2: Move the C++ source and header files to the main code directory of ns-2.

3: Modify the file *packet.h* found in the code directory: Add two lines for the new packet types PT_PCC and PT_PCC_ACK to enum packet_t. Also, add the following two lines to the constructor of class p_info:

```
name_[PT_PCC]= "probabCC";
name_[PT_PCC_ACK]= "probabCCCtl";
```

4: Modify the file *ns-default.tcl* in the *tcl/lib/* sub-directory to include the following defaults for the main PCC parameters:

```
Agent/PCC set AppRate_ 60000; # Constant Bitrate Application
Agent/PCC set overhead_ 0.5;  # dither outgoing packets if >0
Agent/PCC set packetSize_ 1000
Agent/PCC set ndatapack_ 0;   # Number of packets sent
Agent/PCC set printStatus_ 0

Agent/PCCSink set packetSize_ 40
Agent/PCCSink set InitHistorySize_ 100000
Agent/PCCSink set NumFeedback_ 1
Agent/PCCSink set printLoss_ 0
Agent/PCCSink set df_ 0.8;    # decay factor for RTT estimate
Agent/PCCSink set bval_ 1;    # Value of b for TCP formula
Agent/PCCSink set NumSamples_ -1
Agent/PCCSink set discount_ 1;# History Discounting
```

```
Agent/PCCSink set smooth_ 1;  # smoother Average Loss Interval
Agent/PCCSink set totalReceived_ 0
Agent/PCCSink set protLossEvents_ 3
Agent/PCCSink set protRTTs_ 5
Agent/PCCSink set protMaxTime_ 30
Agent/PCCSink set Toff_ 60
Agent/PCCSink set Texp_ 2
Agent/PCCSink set rateDiscount_ 0.8
Agent/PCCSink set adjustRate_ 0
Agent/PCCSink set runtime_ 600
```

5: Modify the file *tcl/lib/ns-packet.tcl* to include the following two lines in the list at the beginning of the file:

```
{ PCC off_pcc_}
{ PCCSink off_pcc_sink_}
```

6: Modify the file *Makefile.in* to include pcc.o and pcc-sink.o in the list of object files.

7: Run *make depend*.

8: Run *make*

After following these steps, PCC and PCCSink can be used like any other agents provided with ns-2. For an easy introduction on how to use ns-2 for simulations, read Marc Greis' Tutorial [21]. The PCC Agent produces a constant bit rate flow at the rate given in parameter AppRate_, while the PCCSink Agent receives this flow and performs congestion control in the way described in this thesis.

Ehrenwörtliche Erklärung

Hiermit versichere ich, die vorliegende Diplomarbeit ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die aus den Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Mannheim, den 27.4.2001 Jan Peter Damm

Bibliography

[1] Sally Floyd and Kevin Fall, "Promoting the use of end-to-end congestion control in the internet," *IEEE/ACM Transactions on Networking*, vol. 7, no. 4, pp. 458–472, Aug. 1999.

[2] Jörg Widmer, Robert Denda, and Martin Mauve, "A survey on tcp-friendly congestion control (extended version)," *Technical Report, Department for Mathematics and Computer Science, University of Mannheim*, Sept. 2000.

[3] Supratnik Bhattacharyya, Don Towsley, and Jim Kurose, "The loss path multiplicity problem in multicast congestion control," in *Proc. of IEEE Infocom*, New York, USA, March 1999, vol. 2, pp. 856 – 863.

[4] Injong Rhee, Volkan Ozdemir, and Yung Yi, "Tear: Tcp emulation at receivers - flow control for multimedia streaming," Tech. Rep., Technical Report, Department of Computer Science, NCSU, Apr. 2000.

[5] S. Floyd, M. Handley, J. Padhye, and J. Widmer, "Equation-based congestion control for unicast applications," *ACM SIGCOMM*, Aug. 2000.

[6] J. Padhye, D. Kurose, and R. Towsley, "A model based tcp-friendly rate control protocol," *Proc. International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV)*, June 1999.

[7] Jitendra Padhye, Victor Firoiu, Donald F. Towsley, and James F. Kurose, "Modeling tcp reno performance: a simple model and its empirical validation," *IEEE/ACM Transactions on Networking*, vol. 8, no. 2, pp. 133–145, april 2000.

[8] Luigi Rizzo, "pgmcc: A tcp-friendly single-rate multicast congestion control scheme," *ACM SIGCOMM*, August 2000.

[9] John Byers, Michael Frumin, Gavin Horn, Michael Luby, Michael Mitzenmacher, Alex Roetter, and William Shaver, "Flid-dl: Congestion control for layered multicast," *To appear in the Second Int'l Workshop on Networked Group Communication (NGC 2000)*, Nov. 2000.

[10] Koichi Yano and Steven McCanne, "A window-based congestion control for reliable multicast based on tcp dynamics," in *Proc. of ACM Multimedia*, Los Angeles, October 2000, ACM.

[11] R. Braden, L. Zhang, S. Berson, S. Herzog, and S. Jamin, "Resource reservation protocol (rsvp) – version 1 functional specification," RFC 2205, IETF Network Working Group, 1997.

[12] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss, "An architecture for differentiated services," RFC 2475, IETF Network Working Group, 1998.

[13] Stefan Savage, Neal Cardwell, David Wetherall, and Tom Anderson, "Tcp congestion control with a misbehaving receiver," *Computer Communication Review*, vol. 29, no. 5, 1999.

[14] Information Science Institute at the University of Southern California, "The network simulator main page," *http://www.isi.edu/nsnam/ns/*.

[15] Jörg Widmer, "A mobile network architecture for vehicles," Tech. Rep., Technical report TR-00-009, ICSI, University of Southern California, USA, May 2000.

[16] S. Floyd, M. Handley, J. Padhye, and J. Widmer, "Equation-based congestion control for unicast applications: the extended version," Tech. Rep., February 2000, URL http://www.aciri.org/tfrc/.

[17] Rajendra K. Jain, Dah-Ming W. Chiu, and William R. Hawe, "A quantative measure of fairness and discrimination for resource allocation in shared computer systems," Tech. Rep., Digital Equipment Corporation, Sept. 1984.

[18] Sridhar Ramesh and Injong Rhee, "Issues in model-based flow control," Tech. Rep. TR-99-15, Department of Computer Science North Carolina, Nov. 1999.

[19] J. Mogul, "Path mtu discovery," RFC 1191, IETF Network Working Group, Nov. 1990.

[20] Jörg Widmer, "Extending equation-based congestion control to multicast applications," in *to appear in: Proceedings of ACM SIGCOMM*, 2001.

[21] Marc Greis and VINT Group, "Tutorial for the network simulator "ns"," *http://www.isi.edu/nsnam/ns/tutorial/index.html*.