REIHE INFORMATIK
3/2001

Probabilistic Congestion Control for Non-Adaptable Flows

Jörg Widmer, Martin Mauve, Jan Peter Damm
Universität Mannheim
Praktische Informatik IV
L15, 16
D-68131 Mannheim

Probabilistic Congestion Control for Non-Adaptable Flows

(updated version)

Jörg Widmer, Martin Mauve, Jan Peter Damm

{widmer,mauve,damm}@informatik.uni-mannheim.de
Praktische Informatik IV, University of Mannheim, Germany

Abstract— **In this paper we present a TCP-friendly congestion control scheme for non-adaptable flows. The main characteristic of these flows is that their data rate is determined by an 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 adaptation of the quality is not possible (e.g., since it is already at the lowest possible quality 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 or not a flow is to be 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 achieve under the same conditions. We present a detailed discussion of the proposed scheme and evaluate it through extensive simulation with the network simulator *ns-2*.**

Index Terms— **Congestion Control, Non-Adaptable Flows, TCP-Friendliness**

I. Introduction

CONGESTION control is a vital element of computer networks such as the Internet. It has been widely discussed in the literature – and experienced in reality – that the lack of appropriate congestion control mechanisms will lead to undesirable situations such as a congestion collapse [1]. Under such conditions, the network capacity is almost exclusively used up by traffic that never reaches its destination.

In the current Internet, congestion control is primarily performed by TCP. During recent years, new congestion control schemes were devised, supporting networked applications that cannot use TCP. Typical examples of such applications are audio and video transmissions over the Internet. One prime aim that these congestion control schemes try to achieve is to share the available bandwidth in a fair manner with TCP-based applications, thus falling into 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 network conditions. Using TCP, it may take a variable amount of time to transmit a fixed amount of data, or with TCP-friendly congestion control, the quality of an audio or video stream may be adapted to the available bandwidth.

While for a large number of applications this not a limitation, there are cases where the data rate of an individual flow is determined by the application and cannot be adjusted 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 no data is transmitted at all. We call network flows produced by these applications *non-adaptable* flows.

In this paper we describe 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 calculate a probability for the two possible states (on/off) so that the expected average rate of the flow is TCP-friendly,
- to perform a random experiment that succeeds with the above probability to determine the new state of the non-adaptable flow, and
- to repeat the previous steps continuously to account for changes in the network conditions.

Through this mechanism it is ensured that the aggregate of multiple PCC flows behaves TCP-friendly.

The remainder of this paper is structured as follows. Section II summarizes related work. In Section III we examine non-adaptable flows in more detail. A thorough description of the PCC mechanism is given in Section IV. The results of the simulation studies that were conducted

are presented in Section V and we conclude the paper with a summary and an outlook on future work in Section VI.

II. Related Work

Much work has been done on TCP-friendly congestion control schemes for applications that cannot use TCP. Prominent examples of these schemes are PGMCC [2], TEAR [3], TFRC [4], and FLID-DL [5]. A discussion of such TCP-friendly congestion control mechanisms can be found in [?]. TCP, as well as all existing TCP-friendly congestion control schemes, requires that the bandwidth consumed by a flow be adapted to the level of congestion in the network. By definition, non-adaptable flows cannot use such congestion control mechanisms.

It is conceivable to use reservation mechanisms such as Intserv/RSVP [7] or Diffserv [8] for non-adaptable flows so as to prevent congestion altogether. However, these mechanisms require that the network supports the reservation of resources or provides different service classes. This is currently not the case for the Internet. In contrast, PCC is an end-to-end mechanism that does not require support from the network. With PCC it is possible to "partly" admit a flow and to continuously adjust the number of flows to network conditions.

We are not aware of any previous work that directly matches the category of probabilistic congestion control.

III. Non-Adaptable Flows

For the remainder of this paper, a non-adaptable flow is defined as a data flow with a sending rate that is determined by an application and cannot be adjusted to the level of congestion in the network. A non-adaptable flow has exactly two 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 states is inefficient, since the application is not able to utilize the offered rate.

Examples of applications using non-adaptable flows are commercial network games such as Diablo II, Quake III, Ultima Online, and Everquest. These games typically employ a client-server architecture. The data rate of the flows 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 the transmission of actions too long, 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. For this reason, a number of applications resort to UDP and avoid congestion control altogether.

A situation with either no congestion control at all or vastly reduced utility in the face of moderate congestion is not desirable. A much more preferable approach is to turn the flows of some participants off and to inform the application accordingly. All other participants do not need to react to the congestion. On average, all users should be able to participate in the session for a reasonable amount of time between off-periods to ensure utility 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 current network conditions.

IV. Probabilistic Congestion Control

The Probabilistic Congestion Control scheme (PCC) provides congestion control for non-adaptable unicast flows by suspending flows at appropriate times. PCC is an end-to-end mechanism and does not require the support of routers or other intermediate systems in the network.

The key aspect of PCC is that – as long as there is a sufficiently high level of statistical multiplexing – it is not important that each single non-adaptable flow behave TCP-friendly at any specific point of time. What is important is that the aggregation of all non-adaptable flows on a given link behave as if the flows were TCP-friendly. Due to the law of large numbers this can be achieved if (a) each PCC flow has an expected average rate which is TCP-friendly and if (b) each link is traversed by a sufficiently large number of independent PCC flows.

At first glance (b) 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 each individual PCC flow occupies a high percentage of the link's bandwidth. We therefore relax (b) 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.

A. 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.* Condition (c) discussed above is met.

R2: *No synchronization of PCC flows at startup.* PCC flows start up independent of each other.

R3: *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.[1]

R4: *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 bandwidth that a TCP flow would have under the same network conditions.

B. Architecture

A simple overview of the PCC architecture is depicted in Figure 1. A PCC sender transmits data packets at the rate determined by the application, while the PCC receiver monitors the network conditions by estimating a TCP-friendly rate using a model of long-term TCP throughput. Whenever a PCC receiver observes a degradation in network conditions, it conducts a random experiment to determine whether or not the flow should be suspended. In the case of a negative result, a control packet is sent to notify the sender that it is temporarily required to stop. After a certain off-period, a sender may then resume data transmission. For PCC, we chose to allocate as much functionality to the receiver as possible to facilitate a future extension of PCC to multicast.

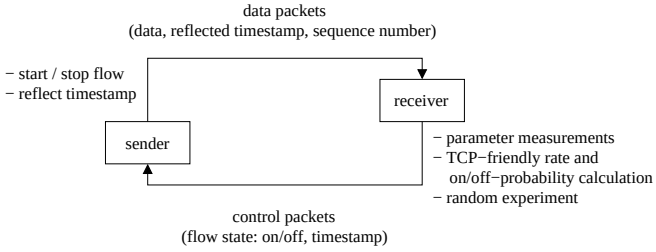

Fig. 1. PCC Architecture

While a flow is in the on-state, control packets are sent at certain time intervals. They allow to continuously measure the round-trip time required to determine the TCP-friendly rate and they serve as a backup mechanism in case of very heavy network congestion. In the absence of these periodic control messages, the sender stops sending, thus safeguarding against the loss of notifications to stop. As long as the flow is in the on-state, the data packets are transmitted at the rate determined by the application. Each data packet includes the timestamp of the most recent control packet that the sender has received in order to be able to determine the round-trip time. Each data packet also contains a sequence number to allow the receiver to detect packet losses.

For the remainder of this work we use the TCP throughput formula of Padhye et al. [9] to compute the TCP-friendly rate. In order to determine the parameters required for the formula, the current version of PCC uses the measurement mechanisms proposed for the TCP-Friendly Rate Control Protocol (TFRC) [4]. However, it is important to note that PCC is independent of the method used to estimate the throughput of a TCP flow for given network conditions. A possible alternative, for example, would be to use the rate calculation mechanism of TCP Emulation At Receivers (TEAR) [3].

C. Continuous Evaluation

To determine the probability with which a PCC flow is allowed to send for a certain time interval T, it is necessary to compare the average rate r_{NA} of PCC to the TCP-friendly rate r_{TCP}.

$$p \cdot T \cdot r_{NA} = T \cdot r_{TCP} \quad \Longrightarrow \quad p = \frac{r_{TCP}}{r_{NA}} \qquad (1)$$

where p denotes the ratio of r_{NA} to r_{TCP}. When solving the equation, two outcomes are possible:

- $p \geq 1$: The non-adaptable flow consumes less than or the same amount of bandwidth that would be TCP-friendly and should therefore stay on.
- $0 < p < 1$: The non-adaptable flow consumes more bandwidth than a comparable TCP-friendly flow. In this case, p is taken as a probability and the non-adaptable flow should be turned off with probability $1 - p$.

For $p \in [0, 1]$, a uniformly distributed random number x is drawn from the interval $(0, 1]$. If $x > p$ holds, the PCC flow is turned off for a time of T. After that time interval the flow may be turned on again. If $x \leq p$, then the flow remains in the on-state. Since we require a sufficient level of statistical multiplexing (R1) and because of the law of large numbers, the aggregation of all PCC flows behaves as if each of them were TCP friendly.

T is an application-specific parameter that is crucial for the utility of the protocol and thus for the user acceptance of the congestion control mechanism. For example, if short news clips are transmitted T should be equal to the length of these clips. If a networked computer game is played, T should be determined so that in "normal" congestion situations the player is able to perform some meaningful tasks during the average time the flow stays on. If the network is designed to carry the required traffic (i.e., congestion is low), then the average on-time will be a large multiple of T.

Under the assumption of a relatively constant level of congestion, the further behavior of PCC is very simple.

[1] There are multiple ways in which this can be done, ranging from a constant bit-rate, flow where this prediction is trivial, to the usage of application level knowledge or prediction based on past samples of the data rate.

After a time of T, a flow that is in the on-state has to repeat the random experiment using the same r_{TCP}. However, in a real network the level of congestion is not constant but may change significantly within a time frame much shorter than T. There are two cases to consider: network conditions may improve (increasing r_{TCP}) or the congestion may get worse.

The first case is not problematic since it does not endanger the overall performance of the network. PCC flows may be treated unfairly in that they are turned off with a higher probability than they should be. However, after a time of T the decision will be reevaluated with the correct probability and PCC will adjust to the new level of congestion.

The second case is much more dangerous to the network. In order to prevent unfair treatment of competing adaptive flows or even a congestion collapse, it is very important that PCC flows respond quickly to an increase in congestion. Therefore, PCC continuously updates the value for p and performs further random experiments if necessary.

Obviously, it is not acceptable to simply recalculate p without accounting for the fact that the flow could have been turned off during one of the previous experiments. Without any adjustments, PCC would continue to perform the same random experiment again and again and the probability to survive those experiments would drop to 0. The general idea of how to avoid this drop-to-zero behavior is to adjust the rate used in the equations to represent the current expected average data rate of the flow.

PCC modifies the value r_{NA}, taking into account the last random experiments that have been performed for the flow. To this end, PCC maintains a set P of the probabilities p_i with which the flow stayed on in the random experiments during the last T seconds.[2] The so-called effective rate r_{EFF} is determined according to the following equation:

$$r_{EFF} = \begin{cases} r_{NA} \prod_{p_i \in P} p_i & \text{for } P \neq \emptyset \\ r_{NA} & \text{for } P = \emptyset \end{cases} \quad (2)$$

For the continuous evaluation and the random experiments r_{EFF} replaces r_{NA} in Equation 1.

D. Initialization

At the initial startup and after a suspended flow restarts, a receiver does not have a valid estimate of the current condition of the network and thus is not able to instantaneously compute a meaningful TCP-friendly rate. To avoid unstable behavior, a flow will stay in the on-state for at least the protected time T', where T' is the amount of time required to get the necessary number of measurements to obtain a sufficiently accurate estimate of the network conditions.

After T', PCC determines whether it should cease to send or may continue. In order to take the data transmitted during the protected time into account, the probability of turning the flow off is increased during the first interval of T so that the average amount of data transmitted during $T' + T$ is equal to that carried by a competing TCP flow. Let r'_{NA} denote the average rate of the non-adaptive flow during the protected time and r'_{TCP} the average rate a TCP flow would have achieved during the same time. For

$$T' \cdot r'_{NA} + p' \cdot T \cdot r_{NA} = T' \cdot r'_{TCP} + T \cdot r_{TCP}$$

the adjusted ratio p' can be calculated as

$$\begin{aligned} \Longrightarrow \quad p' &= \frac{T \cdot r_{TCP} + T' \cdot (r'_{TCP} - r'_{NA})}{T \cdot r_{NA}} \\ &= p - \frac{T'(r'_{NA} - r'_{TCP})}{T \cdot r_{NA}} \end{aligned} \quad (3)$$

Again, for $0 \leq p' \leq 1$ we use p' as the probability for the random experiment. If the flow is turned off, the application may resume sending after it has been off for a least T seconds, starting again with the initialization step.[3] If the flow is not turned off, then the flow will stay on for at least T more seconds, provided that the congestion situation of the network does not get worse.

Note that it is now possible that $p' \leq 0$ if the non-adaptable flow transmits more data during T' than a TCP flow would carry during $T' + T$. Obviously, in this case p' cannot be used as a probability for the random experiment. Instead, it is necessary to turn the flow off and to increase T, so that $p' = 0$.

Through the above mechanism the excess data transmitted during the protected time T' is distributed over a time span of T. At time T', $r'_{TCP} = r_{TCP}$ and $r'_{NA} = r_{NA}$ but in contrast to r'_{TCP} and r'_{NA}, r_{TCP} and r_{NA} continue to be updated after T'.

When a random experiment has to be conducted, it is necessary to calculate not only p' but also the corresponding p. Each is included in their respective set P' and P. As long as PCC is in the first T slot and the protected time has to be accounted for, the values in P' are used to calculate the effective rate and thus the on-probability. Later on, the set P is used.

It may be considered problematic to let a flow send at its full rate for T' as this violates the idea of exploring the available bandwidth as is done, e.g., by TCP slow-start. However, requirements R1 (high level of statistical multiplexing) and R2 (no synchronization at startup) prevent

[2] Note that $p_i = 1$ if the corresponding $p \geq 1$.

[3] T can be adjusted by some random offset to prevent synchronization in case several flows with the same value for T were forced to cease sending simultaneously due to heavy congestion.

this causing excessive congestion. In addition, the value of T' will usually decrease the more congested the network is, since the actual measurement of the loss event rate makes up most of the time interval T'. Loss events become more frequent as congestion increases and therefore the estimate of the network conditions converges faster to the real value. While r_{TCP} is determined, the receiver also calculates the average rate of the non-adaptable flow r_{NA}.[4] Summing up, three important values are determined during initialization: r_{TCP}, r_{NA}, and T'.

E. State Diagram

A finite state machine of a PCC receiver is depicted in Figure 2.

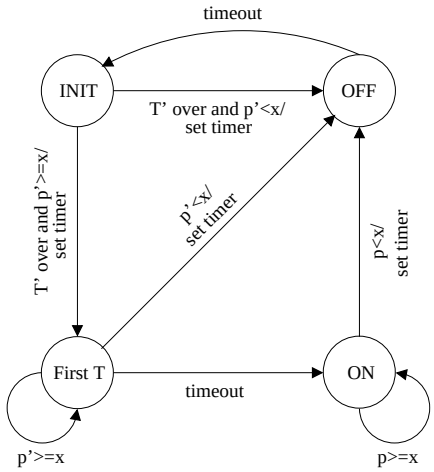

Fig. 2. Finite State Machine of a PCC Receiver

The runtime of the timer used in this state machine is always T.

F. FEC

Since applications generating non-adaptable flows frequently have to obey real-time constraints, they benefit from forward error correction to compensate for packet loss. However, packet loss typically signals congestion. Therefore it has long been considered unacceptable to compensate for congestion-based packet loss by increasing the data rate of a flow with redundant information for forward error correction.

PCC supports the use of forward error correction in a straightforward fashion: 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 and is therefore fair towards competing flows.

[4] In our implementation, we use an exponentially weighted moving average of past PCC rates, but as noted in requirement R3, other options are possible.

G. Example of PCC Operation

To provide a better understanding of the behavior of PCC, let us demonstrate how PCC operates by means of an example. As depicted in Figure 3, the sender starts transmitting at the rate determined by the application. After $T' = 10$ seconds the receiver arrives at an initial estimate of $r_{NA} = 100KBit/s$ and $r_{TCP} = 80KBit/s$. Furthermore, let us assume that the application developer decided that $T = 50$ seconds is a good value for the given application. Now p can be calculated as:

$$p = \frac{80\frac{KBit}{s}}{100\frac{KBit}{s}} = 0.8$$

The value of p is included in the set P and p' is calculated since we are in the first T interval and have to make up for the data transmitted during the protected time.

$$p' = p - \frac{10s \cdot (100\frac{KBit}{s} - 80\frac{KBit}{s})}{50s \cdot 100\frac{KBit}{s}} = 0.8 - 0.04 = 0.76$$

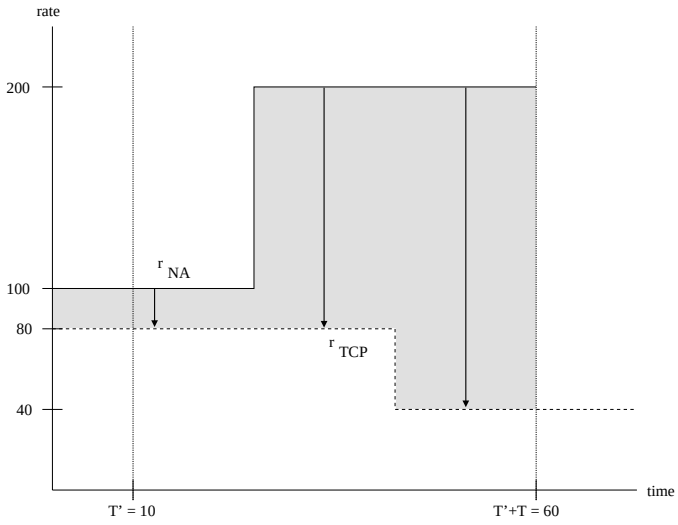

Fig. 3. Example of PCC Operation

Now a random number is drawn from the interval $(0, 1]$, deciding whether the flow will stay on or be turned off. Given a high level of statistical multiplexing, this will result in roughly 1 out of 4 PCC flows being turned off, with the aggregation of the remaining PCC flows using a fair, TCP-friendly share of the bandwidth.

Let us assume that the random number drawn is smaller than p' and that the flow will stay in the on-state. As depicted in Figure 3, at some later point in time the bandwidth required by the application increases

to $r_{NA} = 200KBit/s$. A new value for p is then calculated as follows:

$$p = \frac{80\frac{KBit}{s}}{200\frac{KBit}{s} \cdot 0.8} = 0.5$$

This value for p is saved to the set P for later use. The adjusted probability p' has to be calculated based on the past value of p'.

$$\begin{aligned} p' &= \frac{80\frac{KBit}{s}}{200\frac{KBit}{s} \cdot 0.76} - \frac{10s \cdot (100\frac{KBit}{s} - 80\frac{KBit}{s})}{50s \cdot 200\frac{KBit}{s} \cdot 0.76} \\ &= 0.5 \end{aligned}$$

Let the random number drawn for this decision be below 0.5 so that the flow remains on. A few seconds after this decision the rate a TCP flow would have under the same conditions drops to $r_{TCP} = 40KBit/s$. Consequently new values for p and p' are calculated:

$$\begin{aligned} p &= \frac{40\frac{KBit}{s}}{200\frac{KBit}{s} \cdot 0.8 \cdot 0.5} = 0.5 \\ p' &= \frac{40\frac{KBit}{s}}{200\frac{KBit}{s} \cdot 0.76 \cdot 0.5} - \frac{10s \cdot (100\frac{KBit}{s} - 80\frac{KBit}{s})}{50s \cdot 200\frac{KBit}{s} \cdot 0.76 \cdot 0.5} \\ &= 0.47 \end{aligned}$$

Again the value p is stored in P while the random number drawn is below p'. At $T' + T = 60s$ two things happen: first, the data transmitted during the protected time need no longer be accounted for since PCC has made up for that during the past T interval. Therefore p' is no longer calculated. Second, the first value within P times out and is removed from the set. If the network situation has not changed this will result in the following new value for p:

$$p = \frac{40\frac{KBit}{s}}{200\frac{KBit}{s} \cdot 0.5 \cdot 0.5} = 0.8$$

This time let the random number be larger than p. As a result the flow is suspended for the next T interval before it may start again with a protected time. It should be noted that this example was designed to demonstrate how PCC works. In reality, a situation where the rate of the non-adaptable flow is five times the TCP-friendly rate indicates that the network resources are not sufficient for this application.

H. Extensions

While the current version of PCC works as described above, there are a number of options and possible improvements that we have investigated. In the following we outline two modifications that have not yet been incorporated into PCC.

H.1 Probe While Off

PCC flows on average may receive less bandwidth than competing TCP flows, since a flow that has been turned off may resume only after a time of T, even if network conditions improve beforehand. This degrades PCC's performance, particularly if T is large. In order to improve average PCC throughput, flows that are off could monitor network congestion by sending probe packets at a very 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 the Equations 1 and 3 by including an additional factor $(1 - p) \cdot r_{OFF} \cdot T$.

If the loss rate and the round-trip time of the probe packets signal that r_{TCP} has improved, a flow that has been turned off may be turned on again immediately, without waiting for the remainder of the T to pass, and without performing an initialization step. This may be done only if, under the new network conditions, all experiments within the last T interval had been successful. If the congestion situation worsens later on, it must be checked whether any of the experiments during the last T interval had failed. If this is the case, the flow must be turned off again. Only after the last entry in set P has timed out may the flow resume normal operation. For *Probe While Off* to work correctly, it is of major importance that the estimate of the network parameters work independent of the data rate PCC is sending at.

The current version of PCC does not include Probe While Off, since it could lead to frequent changes between the states "on" and "off", which is likely to be distracting to the user of the application. Furthermore, probe packets waste bandwidth. Probe While Off may be included in a later version of PCC as an option for the application. The mechanism can be improved by including a threshold, so that the flow is turned on again only if the available bandwidth increases significantly. With this improvement, the number of state changes is reduced to improve stability.

H.2 Probe Before On

In PCC, a 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 or not to turn on the flow. Only after the parameters have been estimated and the random experiment has succeeded will real data for the flow be transmitted. The drawback to this method is that bandwidth is wasted by probe packets and that the initial startup of a flow is delayed.

H.3 Loss Rate Monitoring

PCC flows do not take into account the impact of their actions on the network conditions. Assume that the random experiments of a number of PCC flows fail due to increased congestion, but that the congestion was largely caused by these PCC flows. Then too many flows will be suspended since it is impossible to include the expected improvement in the network conditions in the calculation of the on-probability. Similarly, when the bandwidth consumed by PCC flows during the protected time is a significant fraction of the bottleneck link bandwidth, severe congestion may be inevitable. Even after the protected time, the changes in network conditions caused by PCC flows that consume a large fraction of the bandwidth are undesirable.

For these reasons it is vital that the condition of a sufficient level of statistical multiplexing holds and that the PCC flows do not consume too large a fraction of the bandwidth of the bottleneck link. By continuously monitoring the packet loss rate (e.g., through probe packets) and correlating it with the on- and off-times of the PCC flow, it is possible to estimate the impact of the flow on the network conditions. If the PCC flow causes very large variations in the loss rate, the flow should be suspended permanently. With this extension it is possible to use PCC in environments where it is unclear whether the condition of a sufficient level of statistical multiplexing is fulfilled.

V. Simulations

In this section, we use network simulations to analyze PCC's behavior. Simulations are based on the dumbbell topology (Figure 4) since it is sufficient to analyze PCC fairness and the results can be compared to those of other congestion control protocols evaluated with it. For the same reason, simulations were carried out with the *ns*-2 network simulator [10], commonly used to evaluate such protocols. Drop-tail queuing (with a buffer size of 50 packets) was employed at the routers. We used the standard TCP implementation of *ns* for the flows competing with PCC.

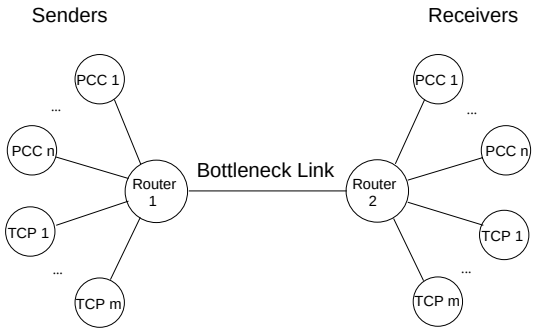

Fig. 4. Simulation Topology

A. TCP-Friendliness

A typical example of PCC behavior is shown in Figure 5. For this simulation, 32 PCC flows and 32 TCP flows were run over the same bottleneck link with 32MBit/s capacity. At an application sending rate of 750KBit/s, the PCC flows should ideally be in the on-state for two thirds of the time. In this example, T was set to $60s$, leading to an expected average on-time of $120s$. The graph depicts the throughput of one sample TCP flow and one sample PCC flow, as well as the average throughput of all 32 PCC flows. The starting time of the PCC flows is spread out over the first $50s$ to avoid synchronization.

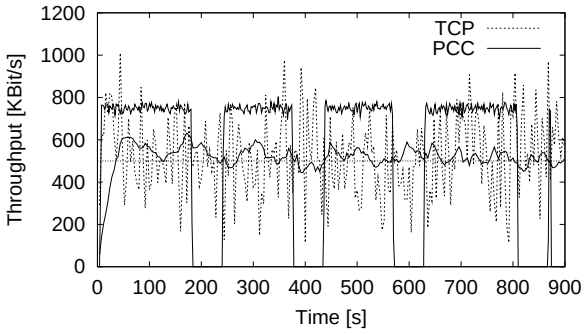

Fig. 5. PCC and TCP throughput

The TCP rate shows the usual oscillations around the fair rate of 500KBit/s. PCC's behavior is nearly perfect, with an average rate that closely matches the fair rate and an on-off ratio of two to one. Naturally, not all of the 32 PCC flows achieve exactly this ratio; some stay on for more, some for less time.

B. Intra-Protocol Fairness

Usually, it is desirable to evenly distribute the necessary off-times over all PCC flows instead of severely penalizing only few. To examine PCC's intra-protocol fairness, a simulation setup similar to the previous one was used, yet the number of concurrent PCC and TCP flows varied between 2 and 128. The probability density function of the throughput distribution from these simulations is shown in Figure 6. As expected, the throughput range is larger for PCC. The coefficient of variation (standard deviation over mean) for PCC throughput is 15% compared to a TCP coefficient of variation of only about 3%.

This results from the time frame for changes in the states of the PCC flows being $60s$ instead of a few RTTs for TCP flows. There is a direct tradeoff between the parameter T and the intra-protocol fairness. Longer on-times, achieved by a larger T, result at the expense of the flows that are suspended for a longer time, thus decreasing intra-protocol fairness. Taken to the extreme, for very

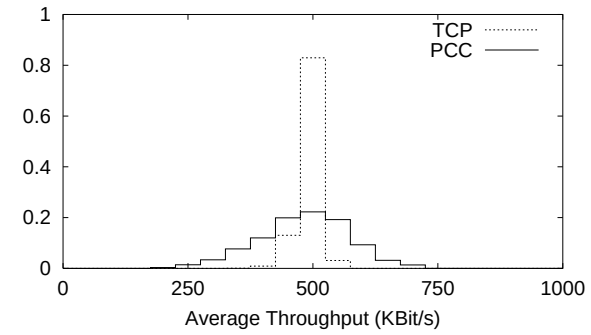

Fig. 6. Distribution of Flow Throughput

large T flows may stay on for the whole duration of the session or are not permitted at all, leading to a type of admission control scheme.

C. Responsiveness

In addition to inter- and intra-protocol fairness, sufficient responsiveness of a flow to changes in the network conditions is important to ensure acceptable protocol behavior. TCP adapts almost immediately to an increase in congestion (manifest in the form of packet loss). Through the continuous evaluation at timescales of less than T, as described in Section IV-C, PCC can react nearly as fast as TCP to increased congestion, however, it will react to improved network conditions on a timescale of T. Figure 7 depicts the average throughput of 32 PCC flows, again with parameter T set to $60s$, and 32 TCP flows. A rather dynamic network environment was chosen where the loss rate increases abruptly from 2.5% to 5% from time $200s$ to $300s$ and from time $400s$ to $420s$.

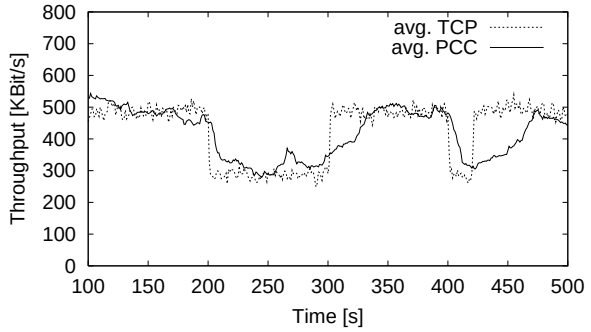

Fig. 7. Loss Bursts

When the loss rate changes at time $200s$, PCC does not adapt as fast as TCP but still achieves an overall average rate that is quite close to the TCP rate after only a few seconds. 60 seconds later we can see a little spike in the average PCC rate, resulting from the PCC flows that reenter the protected time to probe for bandwidth once their off-time is over. Since the loss rate is still high, the average PCC rate settles at the appropriate TCP-friendly rate shortly thereafter. As soon as the loss rate is reduced to its original value, the probability that suspended flows reentering the protected time will immediately be suspended again (and the probability that the random experiment of flows in the on-state will fail) decreases. Thus, after time $300s$, the random experiments of more and more flows succeed until about 50 seconds later the TCP-friendly rate is reached again. Although PCC reacts more slowly than TCP, the average throughput of TCP and PCC up to time $350s$ is very similar. In contrast to long high-loss periods, short loss spikes hurt PCC performance much more than TCP performance. When the loss rate increases again at time $400s$, suspended PCC flows will stay in the off-state for at least $60s$, while the actual congestion persists for only $20s$. From the time the congestion ends until the time the PCC flows are allowed to reenter the protected time, TCP throughput is considerably higher than PCC throughput. However, we can also see from the graph that during periods of congestion PCC throughput does not quite drop to the level of TCP throughput but remains slightly higher. In the following we will analyze this effect in more detail.

D. PCC Throughput for Different Application Sending Rates

Ideally, no PCC flows would be suspended as long as the PCC application sending rate is below the TCP-friendly rate. For higher application sending rates the average PCC rate should remain at exactly the fair rate through the use of the random experiments. From Figure 8 we take it that an average PCC rate of exactly the fair rate is not reached when the application sending rate equals the fair rate but for an application sending rate that is about 25% higher. The latter effect can be explained by PCC's susceptibility to dynamic network conditions. TCP's typical sawtooth-like sending rate results in variations in the network conditions which unduly cause suspension of PCC flows. When we compare the average PCC throughput to TCP throughput for high PCC application sending rates, we find that PCC throughput and thus PCC's aggressiveness continues to increase with the application sending rate once the fair rate has been reached.

The effect of increased aggressiveness at higher application sending rates can be attributed to the TCP model used by PCC. As stated in [9], the TCP model is based on the so-called loss event rate. A loss event occurs when one or more packets are lost within a round-trip time, and the loss event rate is consequently defined as the ratio of loss events to the number of packets sent. The denominator of the loss-event rate increases as more and more packets are sent during a round-trip time due to a higher application

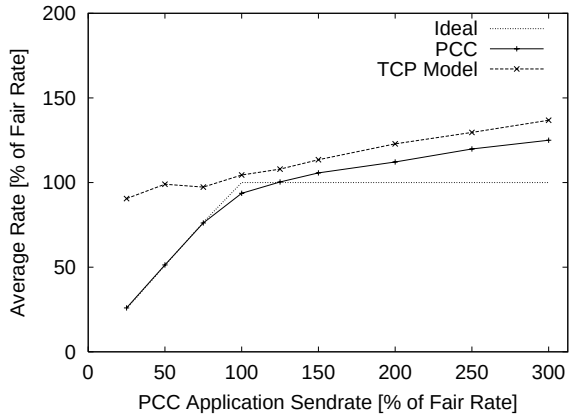

Fig. 8. Comparison with Estimated TCP-Friendly Rate

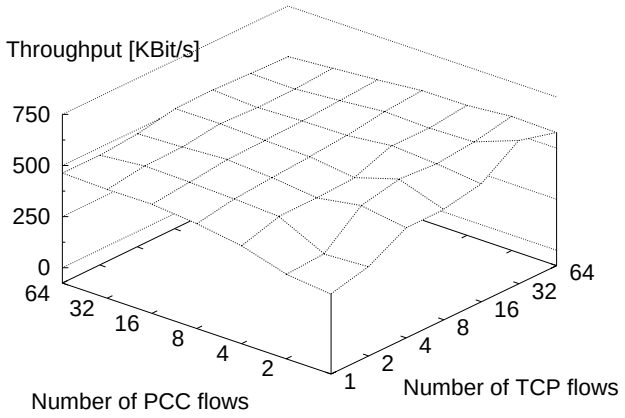

Fig. 9. Average PCC Throughput for Different Numbers of Flows

sending rate. At the same time, the number of loss events does not increase to the same extent since more and more lost packets are aggregated to a single loss event. An in-depth analysis of this effect can be found in [11]. When relating the estimated TCP-friendly rate at different application sending rates to the average PCC rate achieved in these simulations, it becomes obvious that PCC's aggressiveness is not caused by PCC's congestion control mechanism but by the dependence of the TCP model on the measurement of the loss event rate at sending rates close to the actual TCP rate (to ensure that for TCP and the TCP model the lost packets that constitute a loss event are the same). In addition to PCC's susceptibility to variations in the network conditions, the difference between the TCP-friendly rate and the average PCC rate is also caused by taking into account only the rate estimates of flows in the on-state.

E. PCC Fairness for Different Combinations of Flows

Figure 9 shows the average throughput achieved by PCC for different combinations of PCC and TCP flows when the fair rate is 500KBit/s and the application sending rate is 750KBit/s. Generally, PCC throughput increases with the number of TCP flows, since the higher the level of statistical multiplexing, the lower the variations in the network conditions that degrade PCC performance. This effect is more pronounced, the lower the number PCC flows is.

For a more detailed analysis of PCC and further network simulations we refer the reader to [12].

VI. Conclusions

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

We proposed to perform congestion control for such 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.

In a series of simulations we have shown that PCC displays a TCP-friendly behavior under a wide range of network conditions. We identified the conditions under which PCC throughput does not correspond to the TCP-friendly rate. To some extent, these effects on the average PCC sending rate cancel each other out. Nevertheless, the may degrade PCC performance.

We intend to include Probe While Off as an optional element in PCC, which would improve PCC's behavior in highly dynamic network environments. Furthermore, we are currently investigating a method to perform a more accurate estimate of the fair TCP rate if the loss event rate is measured at a sending rate that differs considerably from the TCP-friendly rate. Finally, we plan to evaluate if and how PCC can complement congestion control for multicast transmissions.

References

[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] Luigi Rizzo, "pgmcc: A TCP-friendly single-rate multicast congestion control scheme," in *Proc. ACM SIGCOMM*, Stockholm, Sweden, August 2000, pp. 17 – 28.

[3] Injong Rhee, Volkan Ozdemir, and Yung Yi, "TEAR: TCP emulation at receivers - flow control for multimedia streaming," Tech. Rep., Department of Computer Science, NCSU, Apr. 2000.

[4] S. Floyd, M. Handley, J. Padhye, and J. Widmer, "Equation-based congestion control for unicast applications," in *Proc. ACM SIGCOMM*, Stockholm, Sweden, Aug. 2000, pp. 43 – 56.

[5] John Byers, Michael Frumin, Gavin Horn, Michael Luby, Michael Mitzenmacher, Alex Roetter, and William Shaver, "FLID-DL: Congestion control for layered multicast," in *Proc. Second Int'l Workshop on Networked Group Communication (NGC 2000)*, Palo Alto, CA, USA, Nov. 2000.

[6] Jörg Widmer, Robert Denda, and Martin Mauve, "A survey on TCP-friendly congestion control (extended version)," Tech. Rep. TR-2001-002, Department for Mathematics and Computer Science, University of Mannheim, Feb. 2001.

[7] R. Braden, L. Zhang, S. Berson, S. Herzog, and S. Jamin, ," RFC 2205, IETF Network Working Group, 1997.

[8] 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.

[9] 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.

[10] Sandeep Bajaj, Lee Breslau, Deborah Estrin, Kevin Fall, Sally Floyd, Padma Haldar, Mark Handley, Ahmed Helmy, John Heidemann, Polly Huang, Satish Kumar, Steven McCanne, Reza Rejaie, Puneet Sharma, Kannan Varadhan, Ya Xu, Haobo Yu, and Daniel Zappala, "Improving simulation for network research," Tech. Rep. 99-702b, University of Southern California, March 1999, revised September 1999, to appear in IEEE Computer.

[11] S. Ramesh and I. Rhee, "Issues in TCP model-based flow control," Tech. Rep. TR-99-15, Department of Computer Science, NCSU, 1999.

[12] J. P. Damm, "Probabilistic congestion control for non-adaptable flows," M.S. thesis, University of Mannheim, Apr. 2001.