Traceable Congestion Control

Michael Welzl

Leopold Franzens Universität Innsbruck
Institut für Informatik
Technikerstr. 25/7, A-6020 Innsbruck, Austria
michael.welzl@uibk.ac.at

Abstract. A new, easily traceable congestion control scheme which only requires bandwidth information from rare packets that query routers is presented. In contrast with TCP, the (very simple) rate calculation can be performed by any node that sees the occasional feedback message from the receiver; this facilitates load based charging as well as enforcing appropriate behaviour. The control law, which is based on logistic growth, quickly converges to a fixed and stable rate. In simulations, the mechanism showed greater throughput than TCP while maintaining a very small loss ratio and a smaller average queue length.

1 Introduction

We present a slowly-responsive congestion control mechanism which only uses rare bandwidth querying packets to calculate the rate and needs no other feedback; since it does not depend on the loss ratio or the round-trip time (RTT), the (very simple) rate calculation can be performed by any network node which sees acknowledgments from the receiver. Our scheme has a number of additional advantages: it converges to a stable state instead of a fluctuating equilibrium, has a smooth rate and showed greater throughput and less loss than TCP over a wide range of parameters in simulations.

Signaling is carried out with the *Performance Transparency Protocol (PTP)*, which resembles ATM ABR Explicit Rate Feedback, but is scalable and lightweight: the code to be executed in routers is reduced to the absolute minimum and does not involve any per-flow state — all calculations are done at end nodes. PTP packets carrying information requests are sent from the source to the destination and are updated by intermediate routers. In order to facilitate packet detection, the protocol is layered on top of IP and uses the "Router Alert" option [1]. Upon detection, a router adds a timestamp, the address and nominal bandwidth of the outgoing link[1] and a traffic counter to the packet; single intermediate missing routers are detected via the "TTL" field in the IP header and compensated for by taking the incoming link into account. The receiver builds a

[1] This value represents the maximum bandwidth available to flows from a particular incoming link to a particular outgoing link; it may actually be less than the link bandwidth, but it is fixed and predefined.

table of available bandwidth information from two consecutive PTP packets and determines the available bandwidth at the bottleneck during the period between the two packets. The nominal bandwidth and traffic of the bottleneck dataset are fed back to the sender in a standardized manner, where they are used by the congestion control mechanism; feedback packets are just as easy to detect as any other PTP packet and can be used to calculate the rate similar to the sender somewhere else in the network. For an in-depth description of the protocol, the reader is referred to [2] and [3].

In the next section, we motivate the design of the endpoint control law and show that, assuming a fluid model and equal round-trip times, our mechanism converges to an asymptotically stable equilibrium point. We present simulation studies in section 3, usage scenarios for QoS support, differentiated pricing and incremental deployment in section 4 and conclude with an overview of related and future work.

2 Control Law

The endpoint congestion control scheme builds upon the *Congestion Avoidance with Proportional Control (CAPC)* ATM ABR switch mechanism, which solely relies on available bandwidth information — this is very uncommon for such schemes. Convergence to efficiency is achieved by increasing the rate proportional to the amount by which the traffic is less than the *target rate*, a rate that is slightly less than the bottleneck link bandwidth. If the total traffic is higher than the target rate, rates are decreased proportional to the amount by which the target rate was overshot. Additional scaling factors ensure that fluctuations diminish with each update step, while upper and lower limits for the multiplicative rate update factor increase robustness of the scheme [4].

Trivially, if all RTTs are equal, CAPC must also work if these calculations are done by end systems. The key to finding the necessary code changes for the mechanism to work in the asynchronous RTT case was a simulation based design method for studying two users with a fluid model and a single resource. A detailed description of this process, which is based on the vector diagram analysis of AIMD in [5], can be found in [6]. With this method, it became evident that in order to converge to fairness (equal sharing of the single resource), a user should not just adapt the rate proportional to the current load but to the relationship between the current rate and the available bandwidth. In other words, if a user whose load accounts for 90% of the current total load would increase the rate very slightly, and a user whose load accounts for 5% of the total load would increase the rate drastically, the system would converge to fairness. This relationship prevails even if both users experience a different feedback loop delay.

Applying these changes to CAPC led to a control law that is scalable because it does not depend on the round-trip time: the mechanism still works if users "artificially prolong" their RTTs by restricting the amount of generated measurement packets. If these packets do not exceed x% of the generated payload, PTP traffic does not exceed x% of the total traffic in the network; thus, it

scales linearly with traffic. In conformance with its distributed nature, we call our modified version of CAPC *"Congestion Avoidance with Distributed Proportional Control" (CADPC)*. The control law can be described with the following expression:

$$x_i(t+1) = x_i(t)\Big(2 - x_i(t) - \sum_{j=1}^{n} x_j(t)\Big) \tag{1}$$

Here, $x_i(t)$ is the normalized rate of user i at time t (the time between two PTP measurement intervals – a time unit – and the target rate are 1) and there are n users involved. The sum of the rates of all users models the measured traffic which is obtained from PTP at time t. Since all sources obey the same control law, the total traffic seen during the last measurement interval is simply n times the user's own rate in the synchronous case. The control law can then be shown to stem from the equation for logistic growth:

$$\dot{x}(t) = x(t)a\big(1 - x(t)/c\big) \tag{2}$$

For $a > 0$ and $c > 0$, this equation has the unstable equilibrium point $\overline{x} = 0$[2] and the asymptotically stable equilibrium point $\overline{x} = c$ [7]. It was popularized in mathematical biology and applies to a wide range of biological and sociotechnical systems; the equation can be used to describe the growth of populations ranging from bacteria colonies to animals and humans [8]. Logistic growth is characterized by its S-shape, which is the result of beginning exponentially and slowing the growth rate with the "negative feedback term" $(1 - x(t)/c)$. It seems to make sense that this equation, which generally models the growth of species populations with no predators, but limited food supply can be applied to congestion control in computer networks with limited bandwidth.

Replacing a with 1 and c with $1/(1+n)$, we get the continuous-time form of eqn. 1. In discrete time, a must be less than 2 in order for the mechanism to remain stable. In particular, for values above 2.5699456..., the equation yields a chaotic time series [9]. The rate of a user converges to c while the total traffic converges to $nc = n/(1+n)$, which rapidly converges to 1 as the number of users increases. [3] CADPC can also be expected to properly adapt to background traffic: if we assume a constant background load b, the measured traffic becomes $nx(t) + b$, which yields the equilibrium point $(1-b)/(1+n)$ for the rate of a user. The total traffic then converges to $1 - b$.

Even though the target rate should normally be less than the bottleneck link bandwidth to allow queues to drain, we can use the bottleneck link bandwidth

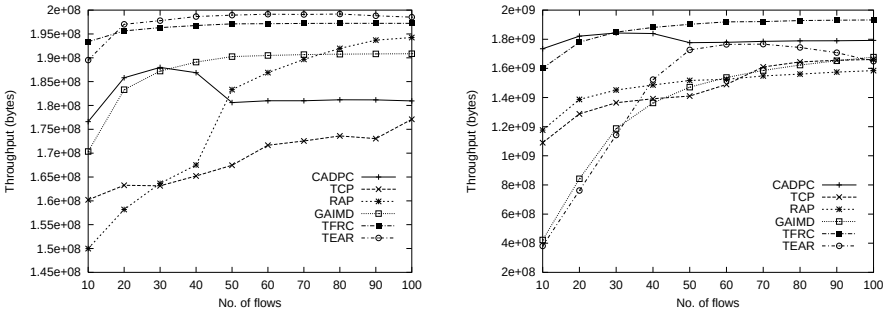
[2] This must be taken care of in implementations: the rate of a user can be arbitrarily small, but it must never reach 0.

[3] One might be tempted to replace eqn. 1 with the logistic growth equation $x_i(t+1) = x_i(t)\big(2 - \sum_{j=1}^{n} x_j(t)\big)$, which converges to $1/n$ in the synchronous RTT case; with this control law, the property of adapting to the relationship between the current rate and the available bandwidth is lost, leading to unfair rate allocations of a single resource in the asynchronous RTT case.

because the traffic will never reach the target rate. In spite of the low values for very few users, this behaviour should be significantly better than the behaviour of TCP: in real data networks, situations where only 2 or 3 users are actually able to use all of the available resources are very rare. We study the behaviour of CADPC under varying conditions by simulation.

3 CADPC performance

CADPC was implemented for the *ns* network simulator; PTP code for ns and Linux can be obtained from [3], where we will make CADPC available as well. During early tests, it became evident that receiving and reacting to feedback more often than every 2 RTTs causes oscillations. This is easy to explain: as a fundamental principle of control, one should only use feedback which shows the influence of one's actions to control the system [10]. Since CADPC differs from TCP in that it reacts more drastically to precise traffic information, increasing the rate earlier can cause the traffic to reach a level where the control law tells the source that it should back off and vice versa. The best results were achieved by calculating the PTP period (when we would send a new PTP packet) similar to the TCP retransmit timeout, which is approximately 4 RTTs; in fact, the scheme performed almost equally when we used exactly 4 RTTs. The round-trip time was estimated via PTP packets only.

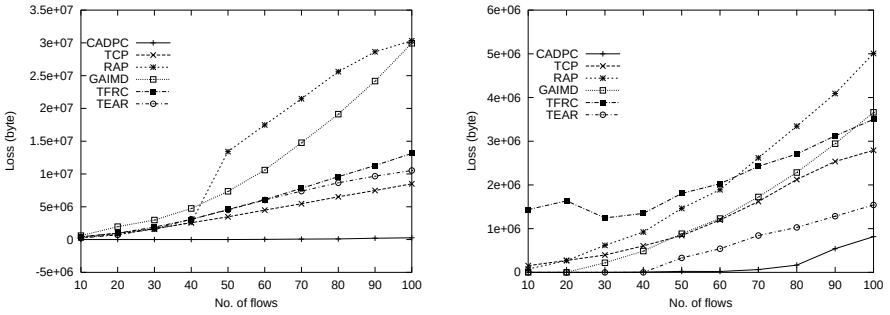


Bottleneck bandwidth: 10Mbit/s Bottleneck bandwidth: 100Mbit/s

Fig. 1. Throughput of CADPC and TCP-friendly congestion control mechanisms

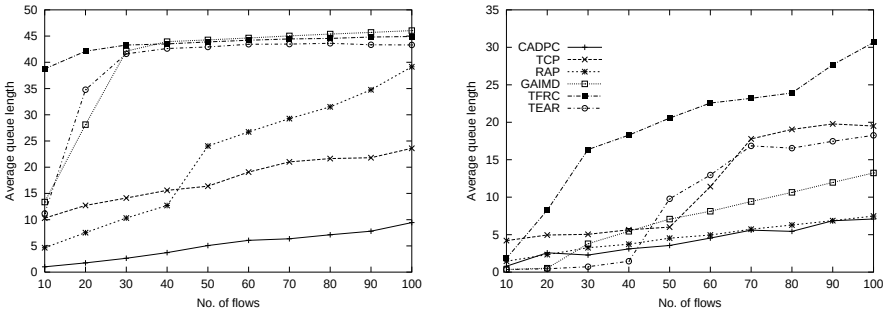
3.1 CADPC vs. TCP(-friendly) Congestion Control

To evaluate the performance of CADPC, its behaviour was compared with TCP Reno and several TCP-friendly congestion control protocols in simulations — RAP [11], GAIMD [12] with $\alpha = 0.31$ and $\beta = 7/8$ (which was realized by tuning RAP parameters), and the slowly-responsive TFRC [13] and TEAR [14].



Bottleneck bandwidth: 10Mbit/s Bottleneck bandwidth: 100Mbit/s

Fig. 2. Loss of CADPC and TCP-friendly congestion control mechanisms



Bottleneck bandwidth: 10Mbit/s Bottleneck bandwidth: 100Mbit/s

Fig. 3. Avg. queue length of CADPC and TCP-friendly congestion control mechanisms

Flows of one particular kind at a time shared a single bottleneck link in a "dumbbell" topology. For all the results in figures 1, 2 and 3, the simulation time was 160 seconds and link delays were 50ms each, which, neglecting queuing delays, accounts for a RTT of 300ms — thus, a simulation took approximately 500 RTTs when the network was lightly loaded. The bottleneck bandwidth was 10 Mbit/s for the diagrams on the left-hand side and 100 Mbit/s for the diagrams on the right-hand side while the bandwidth of all other links was 1000 Mbit/s so as to avoid limiting the rate anywhere else in the network. The queuing discipline was drop-tail, payload packet sizes were 1000 bytes and PTP was used in a mode where packets have an initial size of 32 bytes and grow by 16 bytes at each router [2]. Only the payload flow is shown in the figures.

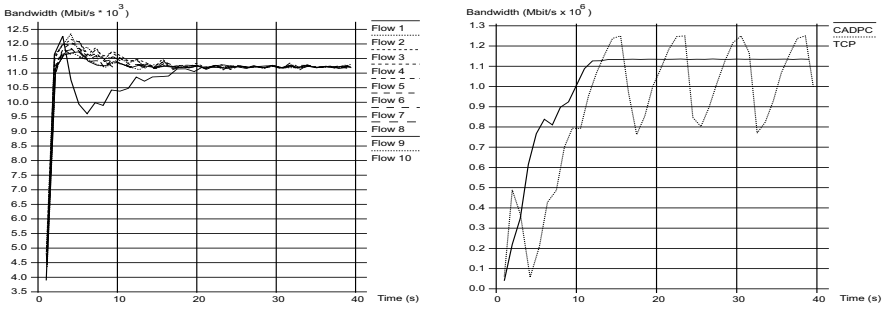
Note that in our simulation scenario, unresponsive senders which transmit at a high data rate would always achieve the greatest total throughput; therefore, high throughput can only be considered a good sign if the packet loss is low. For example, the high throughput of TFRC is not necessarily a good sign because it corresponds with high loss. Yet, TFRC appears to work better with many flows than RAP because RAP shows less throughput as well as more loss.

CADPC significantly outperformed TCP in our simulations: it achieved more throughput (and, in the case of the 100 Mbit link, more throughput than almost all other mechanisms) while maintaining the lowest loss. Also, while CADPC is the only mechanism that does not acknowledge every packet, it almost always showed the smallest average queue size — it should be no surprise that CADPC also worked very well in simulations with Active Queue Management, which we did not include due to space restraints.

3.2 Dynamic Behaviour of CADPC

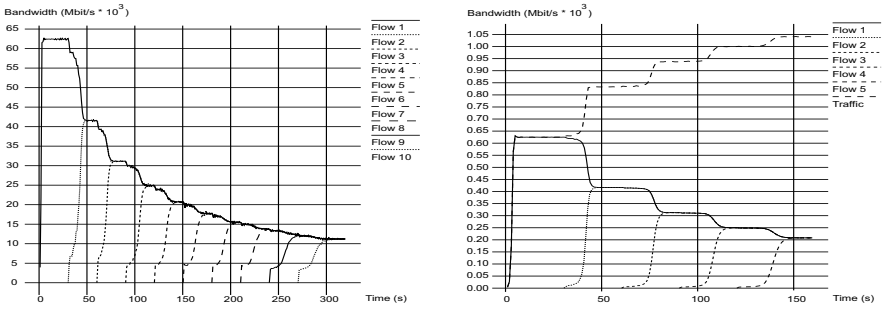
Figure 4 shows that CADPC is also superior in terms of convergence speed and rate smoothness. The outlier after approximately 6 seconds is the result of an overreaction of the very first flow. Since all flows are started at the same time but are nevertheless queued consecutively, only flow 1 sees a completely "empty" link and thus reacts more drastically than all other flows. This behaviour, which we only encountered in the somewhat unrealistic special case of flows starting at the same time, is more severe at higher link speeds and with a larger number of flows — in such a case, it can significantly delay convergence. It should be possible to eliminate this effect: simulation experiments have shown that CADPC can be tuned to behave less aggressively by changing the constant parameter a of the control law to a smaller value > 0 without disturbing the basic behaviour of CADPC (i.e. the point of convergence); it also remains intact if the rate increase or decrease factor is limited by a maximum or minimum value. Tuning these parameters represents a trade-off between convergence speed and robustness against various link delays and the level of statistical multiplexing.

A somewhat more realistic scenario is shown in fig. 5: here, CADPC flows are started with a delay of 30 seconds. In the case of a 1 Mbit/s bottleneck link, CADPC shows minor fluctuations; these are caused by the fact that packet sizes limit the granularity of rate adaptations in the case of a non-fluid model. These fluctuations are diminished at higher link speeds.



Bottleneck bandwidth: 1Mbit/s Bottleneck bandwidth: 10Mbit/s

Fig. 4. CADPC startup behaviour: 10 single flows, 10 * CADPC vs. 10 * TCP



Bottleneck bandwidth: 1Mbit/s Bottleneck bandwidth: 10Mbit/s

Fig. 5. Staggered start of sources

4 Usage scenarios

CADPC has two obvious disadvantages: i) it requires every other router to support the PTP protocol, and ii) any ISP interested in calculating the load should be in control of at least one router along the backward path — which is not always the case for highly asymmetric connections, where the uplink can be a modem and the downlink can be a satellite connection. CADPC will still work in such a scenario, but tracing its theoretical bandwidth becomes more difficult. In the following section, we briefly discuss how CADPC could be used within a controllable domain, where these two disadvantages are no issue. Then, we describe ideas which we think should be pursued in order to overcome the first problem.

4.1 CADPC in a controllable environment

In the simplistic scenario of a packet network where PTP could be used instead of TCP and an ISP is in control of all routers, CADPC can be expected to show the advantages seen in simulations: it should be more efficient than TCP in several aspects, feasible for streaming media applications and remain scalable. Additionally, appropriate behaviour can be enforced through traffic shaping or policing, and it is easy to detect malicious senders, which facilitates the prevention of denial-of-service attacks.

Load-based charging requires a router to trace the bandwidth of a flow — with CADPC, this can be done by merely executing the control law for every feedback packet. It should be noted that this approach may not work under conditions of extreme load because even PTP packets may be lost in such cases. While these occurrences can be expected to be very rare due to the small loss and average queue length seen with CADPC, it may be necessary to design a special back off behaviour for the case of a lost PTP feedback packet.

As an additional aid, if the number of sources is known, the rate which a flow is supposed to achieve (unless flows enter or leave the system) can be determined instantly from the solution of equation 2, which has the same point of convergence [8] as our discrete control law:

$$x(t) = \frac{c}{1 + e^{-at}} \tag{3}$$

Since the calculated rate of CADPC is deterministic and traceable, bandwidth pricing can be differentiated by allowing a sender to behave like m senders; this method, which is similar in spirit to the "MulTCP" approach described in [15], merely involves multiplying the rate by m in our case.

4.2 Incremental deployment ideas

One way of deploying CADPC in an environment where its behaviour is traceable and well-defined (i.e. protected from the more aggressive behaviour of TCP and unresponsive UDP flows) would be the definition of a new DiffServ class. While DiffServ ensures scalability by state aggregation, the necessity for congestion control remains within a behaviour aggregate. Using heterogeneous or TCP-friendly congestion control mechanisms within such an aggregate leads to traffic bursts and packet loss; therefore, it is difficult to provide meaningful per-flow QoS with DiffServ. This problem, which could be called "*QoS in the small*", might be solved by means of a scalable and enhanced congestion control mechanism which is supported by routers. The service could then include a rule such as "we offer QoS and provide router support, you use CADPC and get a good result (and we can calculate your rate, charge you accordingly and make sure that others behave well, too)". This kind of service would of course only be meaningful across domains which support it.

Another way of integrating CADPC with DiffServ would be signaling between edge routers in support of flow aggregates; a thorough description of such an approach can be found in [16]. In this scenario, CADPC is required to be advantageous even if flows are not greedy; this could be investigated by studying, for example, TCP with a rate limit from CADPC (TCP "over" CADPC). CADPC could then also be "translated" into TCP and vice versa by introducing an edge node which acts as both a TCP sender and CADPC receiver at the same time.

The mechanism in [17] is described to be deployable in a cloud-based approach similar to that proposed by Core Stateless Fair Queuing; due to the similarities between this scheme and ours, this method can be expected to apply to CADPC as well.

5 Related and Future Work

The advantage of moving a step further from binary to multilevel feedback is outlined in [19]. Actual ABR-like signaling based on RTCP is used with per-flow state in the "Adaptive Load Service" (ALS) [20] and with a window-based

approach, no per-flow state but more strenuous router calculations where the header of every payload packet is updated in [17]. In [16], the focus is on the interaction between ABR-like signaling and DiffServ. Work based on binary feedback includes TCP-friendly approaches with a smoother throughput such as TFRC and TEAR [13] [14]; while relying on packet loss instead of explicit signaling ensures high scalability, all TCP-friendly mechanisms need to exceed the available bandwidth in order to receive a congestion notification and react properly.

CADPC can be expected to converge to max-min fairness because, in an initially empty network, it works like the "progressive filling algorithm": all sources which share a bottleneck see the same feedback and their rates converge towards the rate of this bottleneck. From the perspective of these flows, other flows which traverse the same link but have a different bottleneck are merely background traffic. These flows, in turn, increase their rates until their respective bottleneck is filled, leading to a max-min fair rate allocation (or, more precisely, a rate allocation which converges to max-min fairness with a growing number of flows). Future work includes testing this claim via simulations with multiple bottlenecks.

A max-min fair rate allocation is achieved by maximizing the linear utility function of all users. Although this kind of utility function may in fact be feasible for streaming media applications, services like file transfer and web surfing have a logarithmic utility function; as of yet, it remains an open question whether CADPC could be extended to support such utility functions and converge to proportional fairness [21].

So far, CADPC was only studied with a single dumbbell where all sources behaved in a similar manner. Future work includes testing the mechanism with different background traffic, different topologies and CADPC flows which are not greedy; possible extensions to multicast scenarios will also be examined. Most importantly, it is planned to pursue the ideas for gradual deployment which were outlined in the previous section.

6 Conclusion

In this paper, a novel approach to congestion control based on explicit and lightweight feedback was presented. The control law converges to a fixed and stable point, which accounts for higher throughput, less loss, less fluctuations and thus better QoS than TCP. Moreover, the rate calculation is simple and can be performed wherever feedback packets are seen; it is easy to enforce appropriate behaviour and implement load-based and differentiated pricing. While our mechanism has the obvious drawback of requiring router support, we believe that the significant performance improvement combined with the ideas for gradual deployment can lead to a future service that is fit for the Internet.

References

1. D. Katz: *IP Router Alert Option*, RFC 2113, February 1997.

2. Michael Welzl: *A Stateless QoS Signaling Protocol for the Internet*, Proceedings of IEEE ICPADS'00, July 2000.
3. The PTP website: *http://fullspeed.to/ptp*
4. A. W. Barnhart: *Explicit Rate Performance Evaluations*, ATM Forum Technical Committee, Contribution ATM Forum/94-0983 (October 1994).
5. D. Chiu and R. Jain: *Analysis of the Increase/Decrease Algorithms for Congestion Avoidance in Computer Networks*, Journal of Computer Networks and ISDN, Vol. 17, No. 1, June 1989, pp. 1-14.
6. Michael Welzl: *Vector Representations for the Analysis and Design of Distributed Controls*, MIC 2002, Innsbruck, Austria, 18-22 February 2002.
7. David G. Luenberger: *Introduction to Dynamic Systems - Theory, Models, and Applications*, John Wiley & Sons, New York 1979.
8. Perrin S. Meyer, Jason W. Yung and Jesse H. Ausubel: *A Primer on Logistic Growth and Substitution: The Mathematics of the Loglet Lab Software*, Technological Forecasting and Social Change 61(3), pp.247-271, Elsevier Science 1999.
9. Mark Kot: *Elements of Mathematical Ecology*, Cambridge University Press, Cambridge, UK, 2001.
10. Raj Jain and K. K. Ramakrishnan: *Congestion Avoidance in Computer Networks with a Connectionless Network Layer: Concepts, Goals and Methodology*, Computer Networking Symposium, Washington, D. C., April 11-13 1988, pp. 134-143.
11. Reza Rejaie, Mark Handley, and Deborah Estrin: *RAP: An End-to-end Rate-based Congestion Control Mechanism for Realtime Streams in the Internet*, Proceedings of IEEE Infocom 1999, New York City, New York, 21.-25. 3. 1999.
12. Y. Richard Yang and Simon S. Lam: *General AIMD Congestion Control*, Technical Report TR-2000-09, Dept. of Computer Sciences, Univ. of Texas, May 2000.
13. Sally Floyd, Mark Handley, Jitendra Padhye, and Jörg Widmer: *Equation-Based Congestion Control for Unicast Applications*, Proceedings of ACM SIGCOMM 2000.
14. Injong Rhee, Volkan Ozdemir, and Yung Yi: *TEAR: TCP emulation at receivers – flow control for multimedia streaming*, Technical Report, Department of Computer Science, North Carolina State University.
15. Philippe Oechslin and Jon Crowcroft: *Differentiated End-to-End Internet Services using a Weighted Proportional Fair Sharing TCP*, ACM CCR, 1998.
16. Na Li, Sangkyu Park, and Sanqi Li: *A Selective Attenuation Feedback Mechanism for Rate Oscillation Avoidance*, Computer Communications, Vol. 24., No. 1, pp. 19–34, Jan. 2001.
17. Dina Katabi, Mark Handley, and Charlie Rohrs: *Internet Congestion Control for Future High Bandwidth-Delay Product Environments*, ACM SIGCOMM 2002, Pittsburgh, PA, 19-23 August 2002.
18. Ion Stoica, Scott Shenker, and Hui Zhang: *Core-Stateless Fair Queueing: Achieving Approximately Fair Bandwidth Allocations in High Speed Networks*, *ACM SIGCOMM 1998*, Vancouver, British Columbia, September 1998.
19. Arjan Durresi, Mukundan Sridharan, Chunlei Liu, Mukul Goyal and Raj Jain: *Traffic Management using Multilevel Explicit Congestion Notification*, Proceedings of SCI 2001, Orlando Florida 2001.
20. Dorgham Sisalem and Henning Schulzrinne: *The adaptive load service: An ABR-like service for the Internet*, IEEE ISCC'2000, France, July 2000.
21. Frank Kelly: *Charging and rate control for elastic traffic*, European Transactions on Telecommunications, 8. pp. 33-37. An updated version is available at http://www.statslab.cam.ac.uk/frank/elastic.html