Scalability and Quality of Service: A Trade-off?

Michael Welzl, Leopold Franzens University of Innsbruck
Max Mühlhäuser, Darmstadt University of Technology

ABSTRACT

During the last decade, two big efforts on Internet quality of service were made. The first, *IntServ*, promises precise per-flow service provisioning but never really made it as a commercial end-user product, which was mainly accredited to its lack of scalability. Its successor, *DiffServ*, is more scalable at the cost of coarser service granularity — which may be the reason why it is not yet commercially available to end users either. This leaves us with the question: is there a fundamental trade-off between QoS and scalability? A trade-off that, in the long run, could prevent deployment of QoS for end users altogether?

SCALE VS. SERVICE

Two years ago, I was faced with a peculiar situation at the University of Linz: I was one of two faculty members who were in charge of teaching approximately 300 students how to write simple programs in Java. With 150 students for each of us, this could have become a severe problem: by default, each of these students felt that we were the ones who would answer each and every question they had. While it would not be a problem to serve around 50 students in this manner, with a number as large as 150, it becomes impossible.

We dealt with this problem in a way that has been common at our university for ages: six tutors were employed, three for each of us. The most important task of these tutors was answering questions as we would personally had there been fewer students. Being students themselves, these tutors did not have the knowledge we had; thus, every once in a while they were not capable of answering a question. The rule we agreed upon was that they would collect questions they could not answer, meet us, and bring up these points on a weekly basis.

Now let us consider what happened on an abstract level: at first, we were experiencing problems due to the large *scale* of users we were facing; we could not provide the fine-grained *service* we were able to provide when there were more faculty members. As a logical layer between students and us, the tutors aggregated the questions and brought them to us. Our service had a *coarser granularity*, but, from our perspective, the number of students we had to communicate with was reduced. The total system worked — it was *scalable*. Notably, the *quality of service* experienced by individual students suffered because we could not answer each question personally.

A similar thing happened on the Internet. To stay with our metaphor, early quality of service (QoS) provisioning concepts simply assigned some students a higher priority than others; these students would obtain better treatment from my colleague or myself (e.g., while others would only be allowed to pay us a visit at specific office hours, questions from these students would be answered at any time). Depending on the number of high- and low-priority students, this method may or may not yield an acceptable service; since the quality of the service degrades in a linear manner as the number of students grows, the process obviously does not scale well (Fig. 1).

The network counterpart of our students being end-to-end traffic flows (source/destination pairs), the *integrated services* (*IntServ*) QoS model realizes just this type of mechanism: services are established by informing routers about the necessary details with the Resource Reservation Protocol (RSVP) signaling protocol; at intermediate routers encountered via standard routing, packets belonging to individual traffic flows are identified and handled appropriately. Separating packets that need preferential treatment from others can be a difficult task: in the case of end-to-end flows, tables of source/destination addresses and port numbers must be maintained (this process is known as *multifield classification*). Since port numbers are not part of the IP header, they may not be readable due to encryption mechanisms or IP packet fragmentation. Additionally, it may be necessary to save special information (e.g., the type of service requested) for each flow; such saved information is called *per-flow state* and grows linearly with the number of traffic flows just as the work for

0163-6804/03/$17.00 © 2003 IEEE

my colleague and me grows linearly with the number of questions we have to answer. Since RSVP typically sets up per-flow state, it is said to not scale well.

At this point, it is important to distinguish between scalability issues that have to do with service granularity and scalability of signaling: in the case of RSVP, the state associated with reservations can even grow like $O(N^2)$, where N is the number of individual traffic flows. As an alternative, more scalable routing protocols were proposed (e.g., [1]), some of which also provide functionality for state aggregation. Still, the inherent scalability of a signaling protocol is a fundamentally different issue than the scalability of a service model, which is our concern in this article.

The next milestone in the history of Internet QoS provisioning is *differentiated services* (*DiffServ*). Here, routers have different roles: *edge routers* (routers at domain endpoints) classify users into separate user groups in order to reduce the amount of state for *core routers* to a handful of user classes; these classes can be determined by looking at the *DiffServ code point* (*DSCP*) field in the IP header. Edge routers can be seen as the equivalent of tutors, while core routers act as my colleague and I did in our example. Thus, DiffServ achieves scalability through aggregation just like us at the University of Linz. To summarize, the following scalability facts can be learned from looking at the teaching example and at computer networks:

- For a mechanism to be scalable, the "work" (in terms of processing power, memory space, etc.) for an entity should not directly depend on the number of users.
- Aggregation is a key concept to reduce the number of users with which an entity has to deal.
- The scalability enhancement achieved by aggregation appears to come at the cost of degraded QoS.

In particular, the latter point is true in our example (students are given more useful answers when they ask us directly instead of asking tutors) and in DiffServ (the services defined by DiffServ are not as strict as IntServ services — DiffServ is an incremental improvement on the best effort Internet service model). Fundamentally, there seems to be a trade-off between the granularity of services and the scalability of the underlying QoS provisioning method. In the remainder of this article, we will take a closer look at this claim and question its validity by explaining three approaches to the problem; we refer readers interested in further detail on the IntServ and DiffServ QoS models to [2].

Approach 1: Combining IntServ and DiffServ

By design, the DiffServ architecture is relatively static. In particular, it lacks a signaling protocol to dynamically reserve resources; users may want to join and leave a particular behavior aggregate, and change their traffic profile at any time while

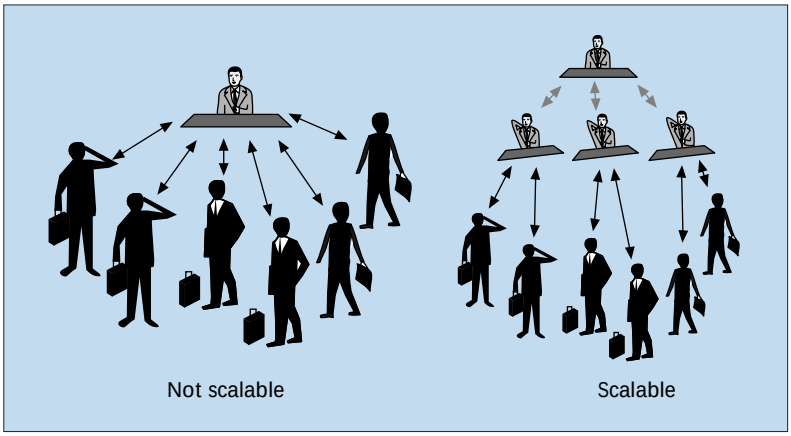

Figure 1. *Me serving six students in two ways.*

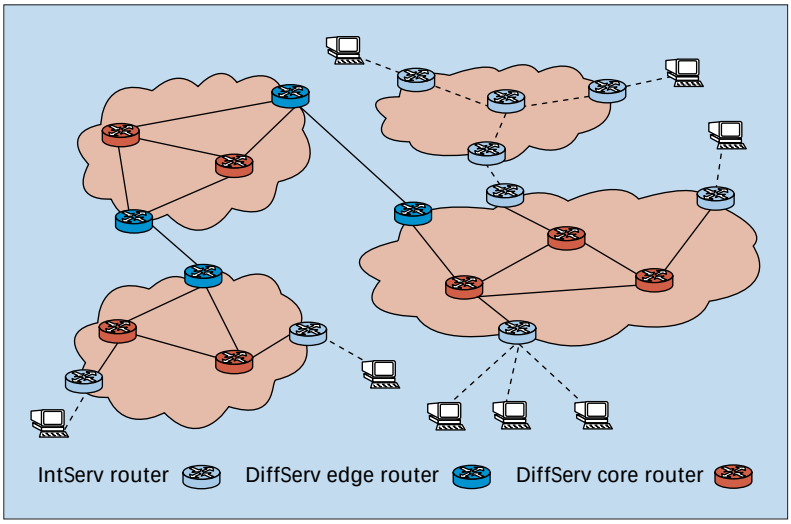

Figure 2. *An example IntServ over DiffServ scenario.*

the service provided through DiffServ is limited through static *service level agreements* (*SLAs*). IntServ, on the other hand, is more flexible but less scalable. As a result, several proposals for combining the *flexibility* of service provisioning through RSVP or a similar, possibly more scalable, signaling protocol with the *fine service granularity* of IntServ and the *scalability* of DiffServ have emerged; some examples are [3, 4], where a whole new simplified framework for guaranteed services is proposed.

Whether RSVP is associated with traffic aggregates instead of individual flows (so-called *microflows*) or a new signaling protocol is used, the scenario always resembles the example depicted in Fig. 2: signaling takes place between end nodes and IntServ edge routers or just between IntServ routers; some IntServ-capable routers act like DiffServ edge routers in that they associate microflows with a traffic aggregate, with the difference that they use the IntServ traffic profile for their decision. From the perspective of an IntServ network (e.g., a domain or a set of domains), these routers simply tunnel through a non-IntServ region. From the perspective of DiffServ routers, there is no such thing as an IntServ network: packets merely carry the information required to associate them

While it is not clear whether each and every stateful mechanism can be approximated in a way that yields satisfactory results, the DPS technique is certainly a promising alternative to the stateful QoS methods we have today.

with a DiffServ traffic aggregate. The network cloud in the upper left corner of Fig. 2 is such a DiffServ domain that is being tunneled through, while the domain shown in the upper right corner represents an independent IntServ network. It is up to the network administrators to decide which parts of their network should act as DiffServ tunnels and when the full IntServ capability could be used.

All such combinations of IntServ and DiffServ clearly represent a trade-off between service granularity and scalability: as soon as flows are aggregated, they are not as isolated from each other as they possibly were in the IntServ part of the network. This means that, for instance, unresponsive flows can degrade the quality of responsive flows. The strength of the IntServ/DiffServ combination is the fact that it gives network operators yet another opportunity to customize their network and fine-tune it based on QoS and scalability demands.

Approach 2: Dynamic Packet State

Dynamic packet state (*DPS*) is a simple and powerful concept that approximates the QoS achieved with per-flow mechanisms without requiring routers to maintain per-flow state. State is kept in packet headers instead; it influences packet processing and causes routers to update their internal state and the state in the packet before forwarding it. In [5], DPS is explicitly called an attempt to "bridge the long-standing gap between stateless and stateful solutions in packet switched networks such as the Internet."

The network scenario for DPS resembles the DiffServ scenario: edge routers — routers at edges of a contiguous trusted network region called the *stateless core* (SCORE) — differentiate between individual end-to-end flows to properly determine the state inserted in the header of a packet as it enters the network. Core routers apply the DPS technique and thus only require internal state that does not depend on the number of flows. As a packet leaves the SCORE, an edge router removes the state from the header. DPS-based mechanisms are designed in two steps. First, a reference stateful network is identified; second, this network is approximated with the DPS technique.

Let us take a closer look at two prominent examples: a realization of fair queuing called *core-stateless fair queuing* (CSFQ) and a method to provide guaranteed services.

Fair queuing is a queuing discipline where responsive flows are protected from the impact of nonresponsive ones in that each flow receives the same share of the link bandwidth. CSFQ, its DPS approximation, realizes a simple probabilistic dropping algorithm that only depends on state inserted by edge routers (a per-flow rate estimate) and the calculated fair share. The latter value is computed by core routers based on the information in packet headers. After the computation is finished, fair queuing is enforced at each core router by dropping certain packets.

The reference network for the guaranteed service model approximated with DPS consists of routers implementing the so-called Jitter-Virtual Clock (JVC) algorithm: as a new packet arrives, it is assigned an eligible time and a deadline; the packet is then held until the time is right. The scheduler orders the transmission of eligible packets according to their deadlines. While guaranteed services can also be realized with different algorithms, a reason to assume JVC was that it only relies on information about a single data flow, which makes it easier to approximate than algorithms that take all active flows into account. The distributed derivation of JVC based on DPS is called Core-Jitter-Virtual Clock (CJVC). Basically, it has edge routers precompute the necessary parameters, and core routers perform the packet scheduling based on these and additionally calculated data (once again, each packet state is updated by core routers as it traverses the SCORE). CJVC is shown to yield the same QoS as JVC in [5].

In addition to these two examples, [5] explains a distributed algorithm that performs relative service differentiation. While it is not clear whether each and every stateful mechanism can be approximated in a way that yields satisfactory results, the DPS technique is certainly a promising alternative to the stateful QoS methods we have today. Of all the approaches examined in this article, it is the most direct attempt to realize a mechanism in a scalable manner without sacrificing service granularity.

Approach 3: Congestion Control and QoS

As explained in [6], unresponsive traffic can lead to a phenomenon called *congestion collapse*: after a certain point, increasing rates of senders that do not perform congestion control causes the total throughput of a network to decrease. Therefore, congestion control is necessary for a network to operate in a stable and efficient manner. The mechanisms found in modern TCP variants represent an example of possibly the most scalable bandwidth management method ever: with the exception of the *explicit congestion notification* (ECN) bit, there is no explicit signaling between end nodes and interior network nodes at all; roughly, the work for routers is a function of the total amount of traffic and does not depend on the number of end-to-end flows. (At this point, we are ignoring details such as traffic phase effects that may render this claim false. However, while it cannot necessarily be reached under all circumstances, independence of the number of flows is clearly a goal of congestion control mechanism designers).

From a QoS perspective, TCP has several disadvantages. First of all, it provides a connection-oriented service (lost segments of a data stream are resent), which may not be desirable for a streaming media application — the type of application for which QoS architectures are mainly made. Properly adapting the rate of a sender is difficult because the bandwidth utilized by TCP

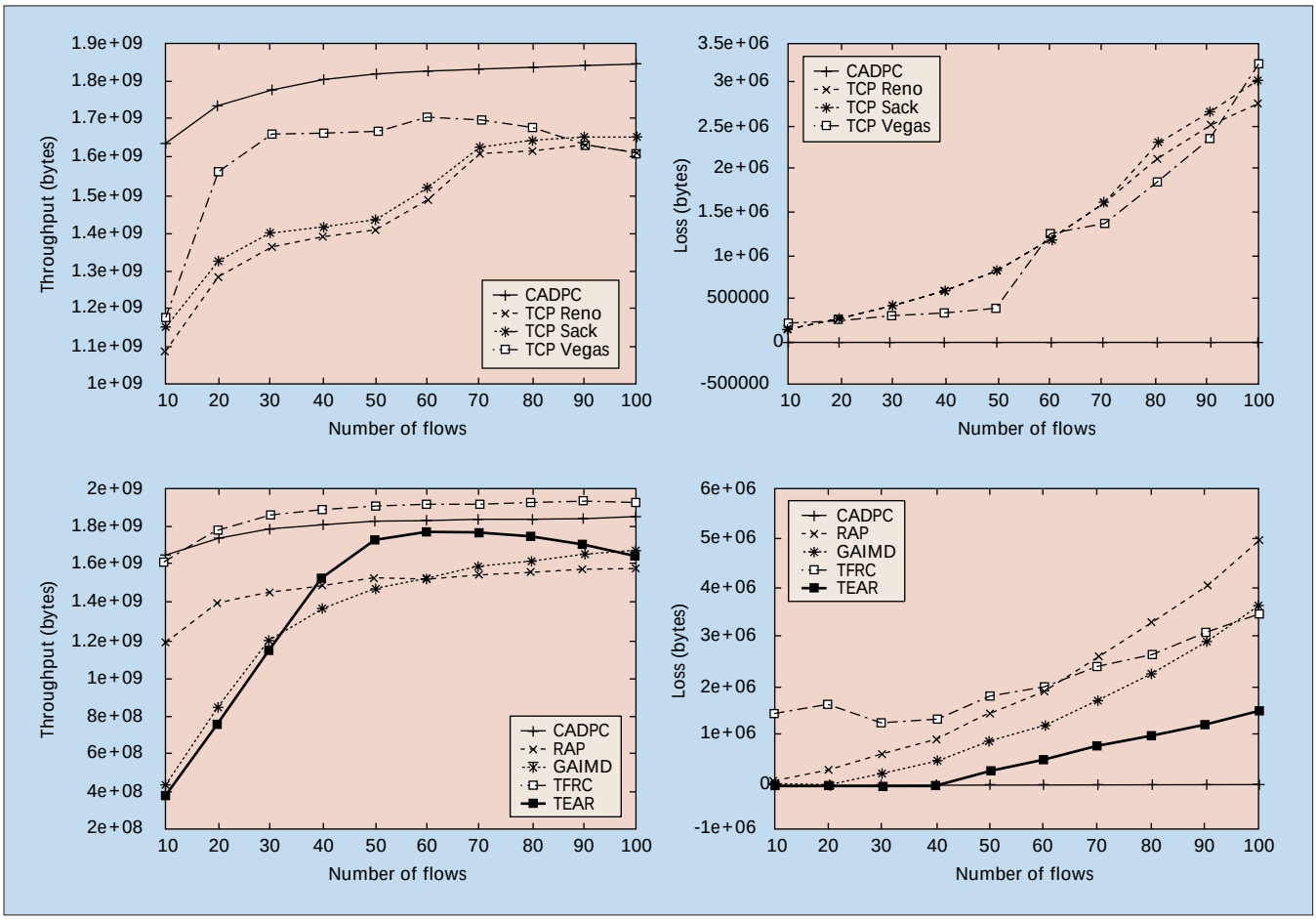

Figure 3. *QoS of CADPC compared to several TCP variants and TCP-friendly mechanisms.*

is constantly changing and can underlie wild fluctuations in a highly congested network; since, in the case of an interactive application such as IP telephony, too much buffering causes unwanted delay, it would be desirable to have a congestion control mechanism that changes its rate more smoothly. Moreover, the rate of TCP is hard to predict without monitoring every lost packet, which makes load (QoS)-based charging a difficult task.

In the world of congestion control research, the common solution is to enforce fairness toward TCP (so-called *TCP-friendliness*) and thus be downward compatible; an overview of mechanisms that conform to this rule can be found in [7]. A flow is said to be TCP-friendly if it is responsive to congestion notification and, in steady state, uses no more bandwidth than conforming TCP running under comparable conditions. Since the throughput of such a flow is roughly given by

$$Throughput = \frac{1.22\ Packetsize}{RTT\sqrt{PacketLossRatio}}, \quad (1)$$

this prescription does not necessarily lead to ideal link utilization. For instance, dependence on the round-trip time (RTT) may be unwanted; also, TCP (without ECN) loses packets on a regular basis, while it may be desirable to have a mechanism with a lower loss ratio. In other words, the limitation of TCP-friendliness hinders the deployment of new, fundamentally different congestion control mechanisms that yield significantly better QoS.

Two examples of mechanisms that outperformed TCP in simulations are the *Explicit Control Protocol* (*XCP*) [8] and *Performance Transparency Protocol* (*PTP*) with CADPC[1] end-to-end behavior [9]. Both use additional signaling of precise bandwidth information (XCP via an additional header in payload packets, PTP via extra packets), and both are scalable in that they do not require routers to keep per-flow state. Additionally, the functionality of CADPC does not depend on the frequency of PTP packets, which means its signaling traffic can be limited in a way that ensures independence of the number of flows: if PTP packets do not exceed *x* percent of the generated payload, PTP traffic in general will not exceed *x* percent of the total traffic in the network; thus, if properly restricted, the overhead from PTP scales linearly with traffic and not with the number of flows.

Neither mechanism is designed to be TCP-friendly, but when isolated from other traffic, they show (among other advantages) better link utilization while maintaining a smaller loss ratio than TCP in a wide range of scenarios; in the case of CADPC/PTP, this is shown in comparison with several TCP variants and TCP-friendly mechanisms in Fig. 3. The diagrams are results of *ns-2* simulations where

[1] CADPC, which stands for congestion avoidance with distributed proportional control, realizes logistic growth based on explicit performance data from PTP.

[2] Note that the best throughput could be achieved by simply sending at a very high rate and not performing congestion control at all in these simulations. Therefore, the high throughput of TFRC is not necessarily a good sign because it comes at the cost of increased loss.

Given the remarkably good results that can be achieved with each of these three approaches, the future looks promising; while it is obvious and rather straightforward to trade QoS for scalability, this does not necessarily have to be the only choice.

10–100 flows of one type at a time shared a single 100 Mb/s bottleneck link.[2] Payload packets consisted of 1000 bytes, each simulation lasted 160 s, all flows were started at the same time, and drop-tail queuing was used. GAIMD was realized by tuning RAP parameters (= 0.31, = 7/8). Notably, CADPC only required one message in the forward and backward direction every 4 RTTs to achieve this result; all other mechanisms acknowledged each and every packet.

Given these results (a performance study of XCP can be found in [8]) and the high scalability of both mechanisms, one obvious way to provide good QoS and remain scalable is to co-design a traffic class with a corresponding congestion control mechanism: within a particular fully isolated traffic aggregate, it should be mandatory to use a special (non-TCP-friendly) congestion control mechanism; the new *Datagram Congestion Control Protocol* (DCCP) could be used as a means to enforce conforming behavior [10]. An even better but presumably more complicated method would be to design a new framework for congestion control, a certain set of rules to be followed by mechanisms within a traffic aggregate instead of a single mechanism.

CONCLUSION

As we have seen, IntServ and DiffServ fundamentally represent a trade-off between fine service granularity and scalability. Three ways to alleviate this strict relationship were described:

- Combining both approaches enables a network administrator to fine-tune the relationship between scalability and QoS to suit customer demands and network capabilities.
- A totally different and highly scalable approach is Dynamic Packet State; it has the potential to eliminate per-flow state (and thereby any dependence on the number of flows) while approximating the service granularity achieved with stateful approaches. Its main disadvantage is that it is radically different than existing approaches, which potentially limits its chances of being gradually deployed — as some believe, a prerequisite for widespread use in the Internet.
- With the service isolation provided by DiffServ (or DPS, for that matter), it would be possible to use new and better congestion control mechanisms that rely on scalable signaling to achieve better QoS than traditional ones. Since some kind of congestion control is necessary as soon as flows are aggregated, it is better to use a traceable and predictable method that provides satisfactory QoS for, say, a streaming media application.

Given the remarkably good results that can be achieved with each of these three approaches, the future looks promising; while it is obvious and rather straightforward to trade QoS for scalability, this does not necessarily have to be the only choice. On the other hand, no matter how hard I try, I cannot come up with a way to apply these efforts to our initial example — for the University of Linz, using something like DPS to serve students in a scalable manner would just be too complicated. Can we then expect these concepts to work for the Internet? Personally, I have not given up hope: some complex things that work for computer networks just do not seem to work for other things in life. For example, I am still waiting for a traffic light to perform slow start and congestion avoidance: instead of going green for a fixed duration, it would allow only one car to pass into the inner city at first, then two, then four … until congestion is detected and the car rate is halved. It will be interesting to see whether such a traffic light or reasonable end-to-end QoS differentiation will be built and used first.

REFERENCES

[1] P. P. Pan *et al.*, "BGRP: Sink-Tree-Based Aggregation for Inter-Domain Reservations," *J. Commun. and Nets.*, vol. 2, no. 2, June 2000, pp. 157–67.

[2] G. Armitage, *Quality of Service In IP Networks: Foundations for a Multi-Service Internet*, Macmillan, Apr. 2000.

[3] L. Westberg *et al.*, "Resource Management in Diffserv (RMD): A Functionality and Performance Behavior Overview," *Proc. Protocols for High Speed Nets. 2002* , Berlin, Germany, 22–24 Apr., 2002.

[4] E. Ossipov and G. Karlsson, "A Simplified Guaranteed Service for the Internet," *Proc. Protocols for High Speed Nets, 2002*, Berlin, Germany, 22–24 Apr. 2002.

[5] I. Stoica, "Stateless Core: A Scalable Approach for Quality of Service in the Internet," Ph.D. thesis, Carnegie Mellon Univ., Pittsburgh, PA, Dec. 15 2000.

[6] S. Floyd and K. Fall, "Promoting the Use of End-to-End Congestion Control in the Internet," *IEEE/ACM Trans. Net.*, Aug. 1999.

[7] J. Widmer, R. Denda, and M. Mauve, "A Survey on TCP-Friendly Congestion Control," *IEEE Network*, Special Issue on Control of Best Effort Traffic, vol. 15, no. 3, May 2001.

[8] D. Katabi, M. Handley, and C. Rohrs, "Congestion Control for High Bandwidth-Delay Product Networks," *Proc. ACM SIGCOMM 2002*, Pittsburgh, PA, 19–23 Aug., 2002.

[9] Michael Welzl, "Traceable Congestion Control," in B. Stiller *et al.*, Eds., *QoS Provisioning to QoS Charging (Proc. QoFIS/ICQT 2002)*, Zürich, Switzerland, Oct. 2002.

[10] E. Kohler *et al.*, "Datagram Congestion Control Protocol (DCCP)," Internet draft, work in progress, draft-kohler-dcp-04.txt, Mar. 2003, http://www.ietf.org

BIOGRAPHIES

MAX MÜHLHÄUSER [M] (max@informatik.tu-darmstadt.de) is a full professor of computer science at Darmstadt University of Technology, Germany. He received his doctorate in informatics from the University of Karlsruhe and founded a research center for digital equipment. Since 1989 he has worked as either a professor or visiting professor at universities in Germany, Austria, France, Canada, and the United States. He has published more than 120 articles, co-authored and edited books about computer-aided authoring/learning and distributed/multimedia software engineering, and has patents in m-commerce pending. He is a member of GI and ACM.

MICHAEL WELZL (michael.welzl@uibk.ac.at) is a faculty member of the University of Innsbruck, Austria, since November 2001. He received his Master's degree from the University of Linz and passed his Ph.D. defense at the University of Darmstadt, Germany, with distinction; thesis publication is pending. From 1999 to 2001 he was a faculty member of the Telecooperation Department at the University of Linz.