Seamless Transport Service Selection by Deploying a Middleware [1]

Sven Hessler [*], Michael Welzl

University of Innsbruck, Institute of Computer Science, Technikerstr. 21a, 6020 Innsbruck, Austria

Abstract

Despite the many research efforts at the transport layer (SCTP, DCCP, etc.), new innovations in that area hardly ever make it into the TCP/IP stacks of standard end systems. We believe that this is due to lack of a flexible interface to the application as well as a lack of transparency – a problem that could be solved by introducing a middleware above TCP/IP. In this paper we present the architecture of our middleware, and show the importance of congestion awareness by simulations. In addition, we explain how to force congestion control on existing applications using our middleware, show the benefits of doing so with simulations, and finally discuss the impact of our middleware towards a better utilization of the network and a more suitable service for the user software.

Key words: congestion control, middleware, DCCP, protocols

1 Introduction

While many new Internet applications with specific requirements to the network emerge, ranging from peer-to-peer file-sharing tools to highly complex Grid computing software, the typical Internet end system provides only two different transport services:

(1) Unreliable datagram delivery (based on the User Datagram Protocol, UDP)

[*] Corresponding author.
Email addresses: sven.hessler@uibk.ac.at (Sven Hessler), michael.welzl@uibk.ac.at (Michael Welzl).

[1] This research is supported by the Austrian Science Fund (FWF).

(2) Reliable in-order delivery of a consecutive data stream (based on the Transmission Control Protocol, TCP)

This has not changed much since the 1970s; given the many new developments at the application layer, it seems hard to believe that these two services — TCP, which realizes reliable in-order delivery of a consecutive byte stream, and UDP, which merely brings the IP “best effort” service to the transport layer — will suffice forever. Yet, although an immense amount of research on transport protocol enhancements for potential successors of TCP and UDP is carried out, the results of this research hardly seem to make it into the operating systems of our personal computer. We believe that one of the reason for this is the socket interface, which lacks the flexibility and transparency that would be needed in order to gradually enhance the transport layer without having to change the applications that use it.

We propose to add a middleware on top of TCP/IP, thereby providing a unique service oriented interface and realizing a certain degree of transparency. We believe that our middleware could enable the deployment of new transport services, no matter if they are realized as TCP alternatives or even as actual end-to-end QoS with strict guarantees.

The rest of this paper is organized as follows: Section 2 gives a survey of the state of the art of research on Internet transport protocols and their prospective successors; Section 3 illustrates how the network and the applications benefit from congestion awareness; in Section 4 we first give a detailed description of the design and the functions of our middleware and present simulations showing how existing congestion-ignorant applications could be forced to use TCP-friendly transport protocols. Section 5 summarizes our findings, discusses potential implications of our middleware, and presents further steps of research.

2 State of the Art

According to an investigation in [1] TCP is still the prevalent transport protocol with a share of approximately 83%. Nevertheless the traffic composition changes all the time, and as the Internet grows, new applications arise. Following the traditional ones (file download and remote login) and the rise of the World Wide Web, an important application class that has emerged is real-time streaming media, encompassing live radio or TV transmissions, Internet telephones and video conferencing tools. Such applications differ from the typical earlier applications in that they have different requirements for the transport layer, e.g. they do not necessarily require reliability, but timely transmission of data.

While UDP merely adds identification of different communicating entities at a single site (port numbers) and data integrity (checksum) to the "best effort" service provided by the IP protocol, TCP encompasses a variety of functions:

Reliability – Connections are set up and torn down; missing segments are retransmitted. If the connections breaks, the application is informed.

Flow control – The receiver is protected against overload.

In-order delivery – Segments are ordered according to sequence numbers by the receiver.

Congestion control – The network is protected against overload by combining a variety of mechanisms, e.g.

- Ack-clocking helps to ensure network stability — roughly, in equilibrium, no packet enters the network unless a packet leaves the network.
- Rate control is used to reduce the sender's rate in response to congestion, following a rule called "Additive Increase, Multiplicative Decrease" (AIMD) for reasons of network stability and fairness.
- Round-trip-time estimation at the sender is carried out by monitoring received acknowledgments. This function is important to fine-tune the other mechanisms.

Congestion control consists of several intertwined algorithms; together, they make TCP a somewhat complex system with many setscrews to fine-tune it. This does not pose a problem in practice because its implementation is entirely left up to operating system designers and parameters are hardly changed.

As mentioned above, the most complex part of TCP is without a doubt its congestion control functionality. When it was added to TCP to protect the network from overload and ensure its stability [2], this appeared to be a sensible choice because TCP was the prevalent protocol, and congestion control in TCP is combined with its window-based flow control, which was already available in the protocol. In the case of UDP, implementing congestion control is entirely left up to application designers — and it is clearly necessary to have at least some kind of congestion control in order to maintain the stability of the Internet [3].

While applications using UDP should contain a congestion control mechanism that ensures fairness towards TCP ("TCP-friendliness"), the window-based stop-and-go behavior and heavy rate fluctuations of TCP are a poor match for real-time streaming media applications. Thus, a large amount of research on more suitable ("smoother") yet TCP-friendly congestion control has been carried out; some examples are TFRC [4], RAP [5], LDA+ [6] and Binomial Congestion Control [7]. An overview is given in [8].

2.1.1 The Datagram Congestion Control Protocol (DCCP)

The fact that the burden of implementing a congestion control mechanism is placed upon the application designer has only recently been acknowledged: the "Datagram Congestion Control Protocol" (DCCP), which is currently undergoing IETF standardization, is a means to support applications that do not require the reliable transport of TCP but should nevertheless perform (TCP-friendly) congestion control [9]. DCCP can be regarded as a framework for such mechanisms which provides a wealth of additional features, encompassing partial checksums — this makes it possible for an application to utilize corrupt data instead of not having it delivered at all (corrupt data are suitable for certain video and audio codecs). Combined with the "Data Checksum Option", partial checksums can also be used to circumvent the common problem of corruption being interpreted as a sign of congestion.

2.1.2 The Stream Control Transmission Protocol (SCTP)

The "Stream Control Transmission Protocol" (SCTP) is another new transport protocol which, like DCCP, is not yet commonly available in the Internet. Unlike DCCP, it is already standardized and ready for deployment [10]. SCTP was designed to transfer telephone signaling messages over IP networks, but is capable of broader applications. Among other things, it extends the TCP service model by separating reliability from in-order delivery. SCTP provides the following features [11]:

Multi-stream support – logically independent user messages can be delivered independently, which can alleviate the so-called "head-of-line-blocking-delay" caused by the strict sequence delivery of TCP. Nowadays, applications requiring this type of functionality typically utilize multiple TCP connections; this is similar to the SCTP feature from an application programmer's point of view, but different for the network as there is only one congestion control instance for multiple streams in the SCTP case.

Multi-homing support – with SCTP, a connection is not associated with a single IP address and a port number on each side but with a set of IP addresses and port numbers — thus, in the case of failure, a connection can transparently switch from one host to another. Such things are already done at a higher level for websites, but with SCTP, multi-homing is realized where it belongs: at the transport layer. This makes the feature independent from the application — FTP based on SCTP, for instance, would automatically use it.

Preservation of message boundaries – this is useful when application data are not a continuous byte stream but come in separate logically independent chunks.

Unordered reliable message delivery – data that come in logical chunks which do not require ordered transmission can sometimes be handed over to the application faster than with TCP (once again, this is due to elimination of the “head-of-line-blocking-delay” problem).

Notably, the congestion control behavior of TCP remains unchanged in SCTP, regardless of the type of application which uses the protocol.

2.2 The Future of Transport Services

As explained above, there will be a large number of transport protocols in the future. An Internet application programmer can be expected to face the following choice of transport services (provided by her operating system) within the next couple of years:

- Simple unreliable datagram delivery (UDP)
- Unreliable congestion controlled datagram delivery (DCCP)
  - with a choice of congestion control mechanisms
  - with or without delivery of erroneous data
- Reliable congestion controlled in-order delivery of
  - a consecutive data stream (TCP)
  - multiple data streams (SCTP)
- Reliable congestion controlled unordered but potentially faster delivery of logical data chunks (SCTP)

This is only a rough overview: each protocol provides a set of features and parameters that can be tuned to suit the environment or application — for example, the size of an SCTP data chunk (or UDP or DCCP datagram) represents a trade-off between end-to-end delay and bandwidth utilization (small packets are delivered faster but have a larger per-packet header overhead than large packets). Work based on TCP parameter tuning is described in [12,13].

Nowadays, the problem of the transport layer is its lack of flexibility. In the near future, this may be alleviated by the availability of DCCP and SCTP — but then, it is going to be increasingly difficult to figure out which transport protocol to use in what mode and how to tune its parameters. The problem will not go away. Rather, it will turn into the issue of coping with transport layer complexity.

3 The Benefits of Congestion Awareness

Congestion control is an important element of all the aforementioned transport layer protocols except for UDP, and it is an integral part of our middleware.

From the perspective of a single user, selfishly maintaining the rate could lead to more throughput than with congestion control, which reduces the rate when the network is overloaded. In this light, one might be tempted to ask the questions: Does a flow which reacts to congestion profit more than a "selfish", unresponsive UDP flow? How about the network? Are there good reasons to implement congestion control at all?

It is well known that unresponsive UDP flows could badly affect TCP flows and let them malfunction in the worst case [3]. Another less known, but highly interesting effect is congestion collapse as explained in [14]. In an environment of unresponsive UDP flows even the flows themselves could suffer from their unresponsiveness to congestion.

We show by simulation using the "ns-2" network simulator [2] that a congestion collapse will not occur when flows react to loss and congestion; this can be beneficial for the the flows themselves without causing any negative effect from the perspective of a single selfish flow. Figure 1 shows the Dumbbell topology

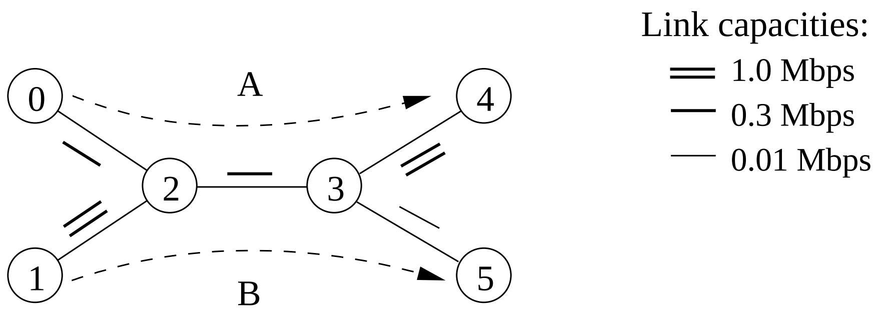


Figure 1. Congestion collapse test topology (Dumbbell)

which was used for testing. The topology contains two senders; one (0) is connected to the node 2 with a 0.3 Mbit link, the other (1) with a 1.0 Mbit link. There are two sinks (4, 5) which are connected to the node 3 with a 1 Mbit (node 3–4) and a 0.01 Mbit (node 3–5) link respectively. The inner nodes (2, 3) are interconnected with a 0.3 Mbit link. We performed two classes of simulations. The first consisted of two (unresponsive) UDP/CBR sources, and in the second test the sources used DCCP with TCP-like behavior. The nodes used a DropTail queuing strategy per default. For the validation of the results, i.e. to make sure not to be hoaxed by phase effects, the queuing strategy was changed to Random Early Detection (RED) with out-of-the-box parameters. We did not tune them because they do not have an impact on the effect we intended to show.

[2] ns-2 v2.27 with the DCCP v1.1 patch from `http://www.ns.dccp.org/`

3.1 Two UDP sources – congestion collapse

As mentioned, in the first test we used two UDP, constant bit rate (CBR) sources which started with an initial sending rate of 800 Byte/s. Both sources increased their sending rate two times per second by 800 Byte. Figure 2 shows

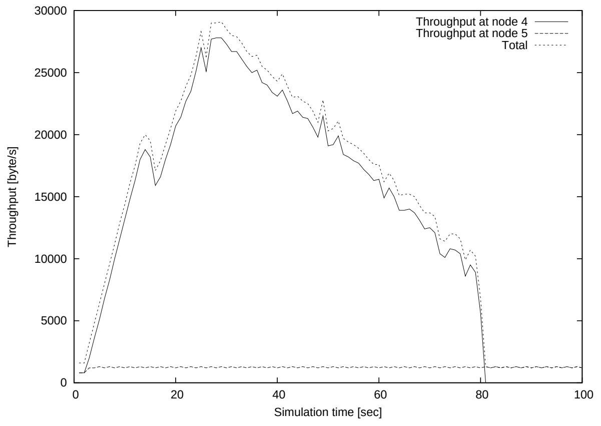


Figure 2. Congestion collapse caused by 2 UDP/CBR sources

the simulation result, i.e. the throughput measured at the destination links (connection 3–5, and 3–4). Both senders increase their rate continuously, which corresponds with increasing throughput until the 3rd second of the simulation. From there on, the link capacity of node 5 (connection 3–5) is saturated, but node 1 is still increasing its rate. Node 0 increases its sending rate too, and this causes the throughput at node 4 to grow until the 28th second of the simulation. The throughput at this moment (27800 Byte/s) is below the capacity of the bottleneck link between node 2 and node 3 (0.3 Mbit), and also lower than the link capacity between node 3 and node 4 (1 Mbit). Afterwards, the throughput at node 4 and also the total throughput decrease massively and return to zero on the 81st second of simulation. The overall usage of the link between node 2 and node 3 is only about 10 kbps from second 81 on.

The shown effect is called "congestion collapse" [3] and it is caused by the greedy and unresponsive UDP flow from node 1 to node 5.

3.2 Two DCCP sources

We used the same setup as in Section 3.1 but substituted the UDP sources with two DCCP Agents using TCP-like behavior. In Fig. 3 the results for this run are shown. Now the congestion collapse effect does not appear because DCCP

[3] Note that there are also other forms of congestion collapse, e.g. the effect described in [3]

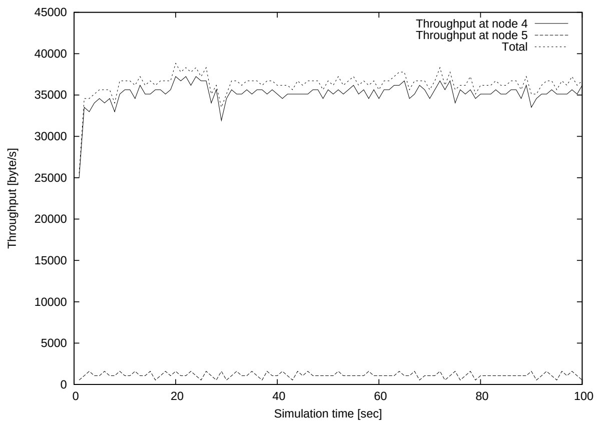


Figure 3. Simulation with 2 DCCP/TCP-like sources

is congestion-aware. Both senders increase their rate until they saturate the links connecting the sinks with node 3. After 3 seconds both outgoing links are nearly saturated and the total throughput fluctuates only slightly along the theoretical bottleneck link capacity of 0.3 Mbit. The fluctuation of both flows is typical for TCP-like behavior; both senders increase their sending rate until congestion occurs. Once packets get lost, the congestion avoidance algorithm is used, i.e. the sending rate will be reduced and carefully increased again later on, until congestion is detected, a.s.o.

3.3 Discussion

As shown above in an – admittedly – simple test setup, flows using a congestion-aware protocol like DCCP instead of plain UDP will improve network usage and be beneficial for all flows. As mentioned in [15] the increasing share of unresponsive UDP flows, e.g. Voice-over-IP (VoIP) connections, could harm an Internet service provider's (ISP) operativeness. It is therefore reasonable to expect that ISPs will limit UDP flows in some way in the future.

Taking this information into account, it would make sense to migrate applications based on UDP to congestion-aware transport protocols. Our middleware between application and transport layer may be a first step into this direction. It could enforce congestion-awareness of UDP applications by providing an additional layer which acts like a communication tunnel. The behavior for the application does not change, but all the UDP communication will be transfered through a tunnel using an appropriate transport protocol, e.g. DCCP. The decision which transport protocol is used for the tunnel is based on the application class, e.g. real-time streaming media or voice traffic, and the expected traffic characteristics.

4 The Middleware

An abstract view of our envisioned middleware is shown in Fig. 4. It hides transport layer details from the application and fulfills requests from the application level with best effort. This architecture simplifies the process of choosing and tuning transport services for applications and making them independent of a particular network infrastructure — as soon as a new mechanism becomes available, the middleware can use it and the application works better. In what follows, we will give a detailed description of the interfaces in both

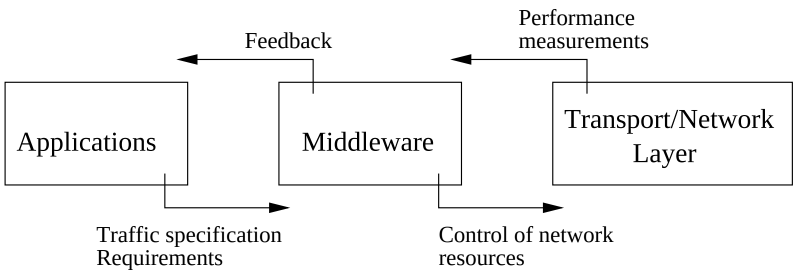
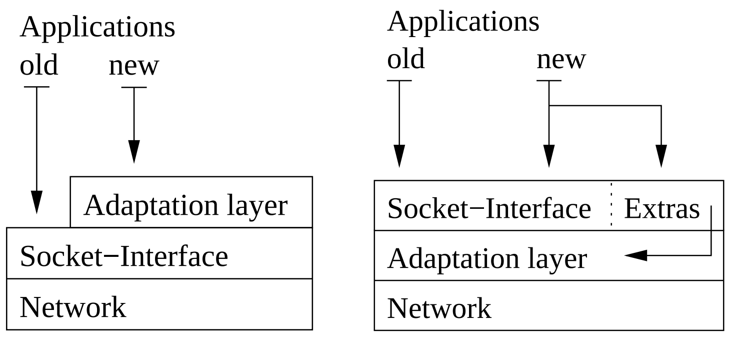


Figure 4. An abstract view of our middleware and its interfaces to the applications and the network

directions: from the application to the middleware, and from the middleware to the transport instance.

4.1 Interface to the application

The application which uses the middleware basically has to specify two things: its requirements to the network and the behavior it will show, i.e. what traffic pattern will be generated by the application. Figure 5 shows two different approaches of implementing a middleware to extend functionality of existing and new applications. Contrary to the common way (Fig. 5(a)), where the "adaptation-layer" – the element that realizes the new functionality – is above the socket interface, we propose to put this layer between the socket interface and the network (Fig. 5(b)). The common solution has the disadvantage of offering different programming interfaces to the applications, depending on whether the application already exists or it is a new one. While with the common approach only new applications could benefit from the middleware, our proposal (Fig. 5(b)) eliminates these drawbacks. It offers all application the same programming interface and lets them use the underlying adaptation-layer automatically and transparently. This means that old applications behave as usual (or better) and new programs could use the extensions to the socket interface to specify their needs, e.g. in terms of delay or throughput requirements.



(a) Common approach (b) Our proposal

Figure 5. Different ways to extend functionality by using a middleware

For an optimal adaptation of the application requirements to the network's options, it is very important that the middleware knows how the application behaves. Does it use all the resources it is given ("greediness")? How does the generated traffic pattern look like, e.g. are there alternating long transmission phases with short breaks? Is it predictable at all or could it be only based on statistics as the traffic pattern depends on user behavior?

For example, interrupting a TCP connection can cause the congestion window to decay [16]. Therefore, the underlying AIMD policy, which has the protocol slowly increase the rate, can lead to a severe throughput reduction that might be compensated for by the middleware via buffering i) if delay is known to be less important than bandwidth and ii) the data stream is known to have short interruptions.

As another example, an adaptive multimedia application (a real-time Internet application which adapts multimedia content based on network measurements) may not always be able to satisfy the criteria dictated by the network: in the case of layered data transmission, there may only be a limited number of rate steps (i.e. the application may have a coarser rate granularity), or there can be upper and lower bounds. When tuning the compression factor of a data stream to suit the available bandwidth, it may not even be possible to predict a precise rate but only estimate an average of some sort. To have knowledge of these things could be helpful for the middleware.

4.2 Functions of the Middleware

The middleware itself controls resources by choosing an appropriate transport instance and performing additional functions, e.g. buffering. Furthermore it needs to monitor the performance with respect to the requirements that were specified earlier. Besides, the middleware provides QoS feedback to the application, which can then base its decisions upon this tailored high-level information rather than generic low-level performance measurements.

Choosing a transport instance The middleware chooses and tunes an appropriate mechanism from the transport or network layer depending on the particular environment. This could mean to make a decision between different protocols, e.g. DCCP vs. UDP-Lite [17] and fine-tune their parameters, or even using an existing network QoS architecture such as IntServ or DiffServ and transforming the application's requirements to its specific attributes.

Additional tasks Beside the aforementioned, the middleware performs additional functions such as buffering or choosing the ideal packet size. Even this seemingly simple task is tricky, due to large packets generally cause less overhead as long as they do not exceed the fragmentation limit (the "Maximum Transmission Unit" (MTU) of a path), but small packets yield a shorter delay. The decision therefore depends on the bandwidth and delay requirements that were specified by the application. The packet size could, of course, also be mandated by the application or a congestion control mechanism.

Exclusions While the middleware is concerned with many complex tasks, some lower level issues remain out of its scope: retaining the stability of the network, for instance, can be left up to the transport- and network-layer mechanisms that it chooses, provided that the middleware does not switch too frequently; the determination of parameters to appropriate limit this frequency is a matter of future research.

4.3 Proof-of-concept

As a proof of our middleware concept, we show that is possible to force congestion control transparently to existing applications without modifications. We demonstrate this by means of an ns-2 simulation [4] where we forced VoIP flows to use congestion control so as to avoid the problem that an increasing share of unresponsive UDP flows could harm an ISP's operativeness as described in Section 3.3.

To this end, we used a modified TFRC agent (called "VoIP proxy" from now on) which enforces congestion awareness for unresponsive constant bit rate flows. We employed the same topology as used in Section 3 (Fig. 1). For simplification we will refer to the branch connecting node 0 with node 2 as

[4] Simulation scripts are available at `http://dps.uibk.ac.at/~sven/tcc/archive/middleware-latest.tgz`

“upper branch” and call the connection between node 1 and node 2 “lower branch”. Additionally we will define the tests using the VoIP proxies as “VoIP” and those without the middleware “UDP” although both tests use UDP/CBR senders.

4.3.1 Test setup

The middleware is implemented as a proxy which converts incoming CBR packets into TFRC packets and sends them to the sink.

Each source node (0,1) of Fig. 1 is connected to a CBR sender, either directly or via (n, m) VoIP proxies in which the intermediate link had a delay of 0 ms and a capacity of 1 Gbit. The destination nodes (4,5) are coupled with (n, m) sinks. The logical data flow goes from the UDP/CBR sender through the proxy (if it is used) and finally to the sink.

The parameters for the VoIP proxy are inherited by the TFRC agent, i.e.

```
set SndrType_ 1
set voip_ 1
set voip_max_pkt_rate_ 100
set packetSize_ 120
set maxqueue_ MAXQ
```

The UDP/CBR sources send with a packet rate of 100 packets per second, and a packet size of 120 Byte and for a duration of 100 seconds.

A central value for the behavior of a TFRC-sender is the maximum queue length (MAXQ) for incoming data from the application – in our case from the CBR sender – as it affects the end-to-end delay. We performed three tests with 3 CBR sources at the upper and the lower branch and chose a maximum queue length of 1, 10, and 25 packets. Figure 6 shows the results of these pretests using the VoIP-proxies, and additionally, the UDP curve as a base value. The mean one-way delay for the upper branch (node 0–2) fluctuates marginally for all selected values of the queue length. In contrast the delays at the lower branch (node 1–2) strongly depends on the maximum queue length (MAXQ). Taking these results into account we set the queue to one packet for all further tests as they are aimed at delay critical applications.

4.3.2 Test results

We performed an extensive series of simulations and varied the number of senders at the upper and the lower branch, enabled and disabled the VoIP proxies, and chose different queuing strategies (DropTail vs. RED). Some re-

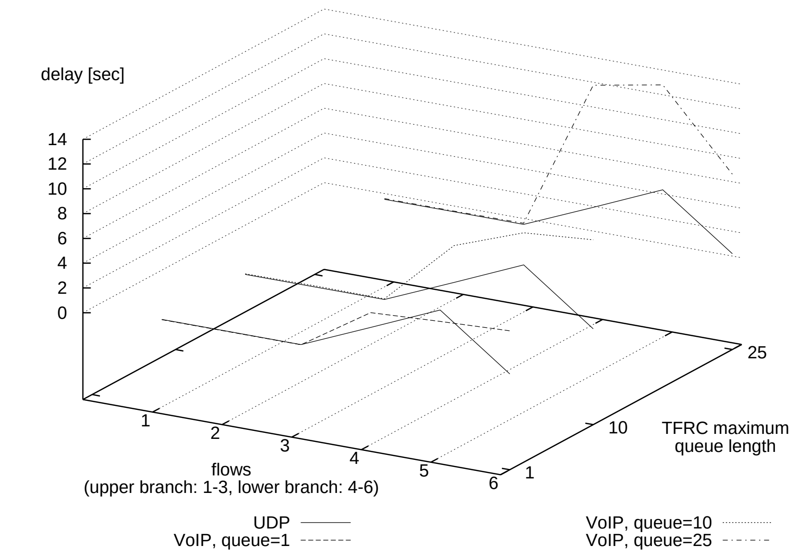


Figure 6. Mean one-way delay subject to the TFRC maximum queue length

sults of these tests are shown in Fig. 7 and Tab. 1. In Fig. 7 the throughput, i.e. the number of bytes received at node 4 and 5, is equally high no matter how many senders are involved for the tests where the unresponsive UDP/CBR flows are tunneled by a VoIP proxy. The overall throughput for these tests (ca. 35 kByte per second) is limited by the bottleneck capacity. Contrary but

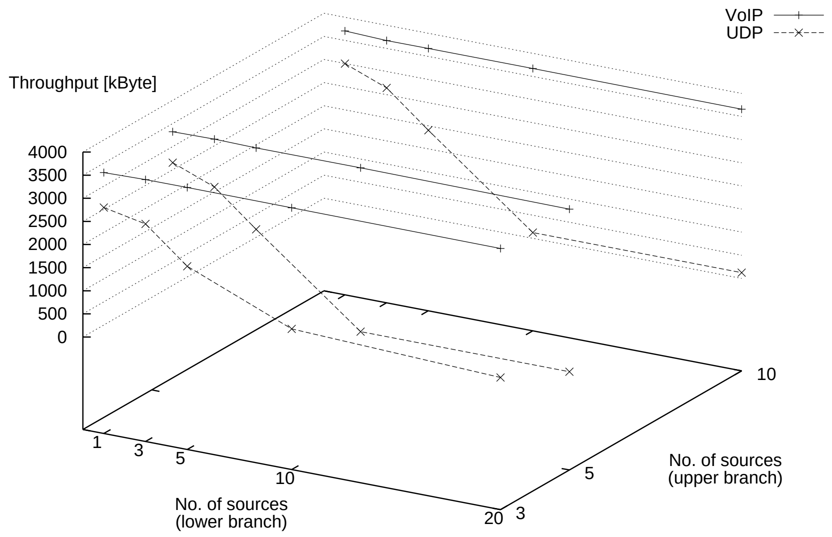


Figure 7. Throughput of a setup with and without using VoIP proxies

consistent with the tests in Section 3, the simulations without the middleware show the congestion collapse effect. The higher the number of senders, the lower the number of bytes received at node 4 and 5.

This behavior could be explained as follows. In both tests, the sources send with a constant rate. Packets get dropped typically at the bottleneck routers, i.e. node 2 or 3. For the UDP tests, i.e. the tests without the VoIP proxies, most of the packets from the senders at the lower branch get dropped in router 2, as the link capacity of the outgoing link (node 3–5) is much lower than the incoming link (node 1–2). The constant sending rate of a CBR source does not take congestion into account and yields congestion collapse in worst case. When the VoIP proxies are used, the dropping of packets moves from the routers to the proxies. The congestion-aware TFRC senders receive packets from the CBR senders but transmit data according to the congestion status of this connection, i.e. packets get dropped early and do not clog the bottleneck link.

number of flows at		UDP – delay [sec] at		VoIP – delay [sec] at	
upper	lower branch	upper	lower branch	upper	lower branch
1	1	0.029	4.787	0.031	4.203
1	3	0.063	4.873	0.031	4.314
3	1	0.070	4.814	0.068	4.404
5	5	0.098	4.864	0.106	4.690
10	10	0.066	4.888	0.163	5.775

Table 1
Mean of the one-way delay [sec] subject to the kind of test (UDP vs. VoIP)

Table 1 shows the mean of the one-way delay, measured in seconds, for both simulations scenarios. The results show similar values for all tests except that with a heavy congested network. In this particular case the one-way delay is slightly better, although the benefit is limited as only a very small percentage of the packets profit due to the very high loss rate (c.f. Fig. 7).

5 Conclusion and future work

In this paper, we have motivated and discussed the idea of placing a middleware on top of TCP/IP. We presented a ns-2 simulation to show how unresponsive flows could affect the network and concurrent streams. Further we argued for introducing an middleware on top of the socket interface, and justified our concept by means of simulations.

We showed that not only the network as a whole will benefit from our middleware but also a single, originally unresponsive, flow which changes its behavior to be congestion-aware.

The goal of our concept is to provide a unique and sufficiently flexible interface to all kinds of Internet applications, thereby enabling gradual deployment of a wide range of network services. We believe that such a middleware would have quite a positive impact on Internet applications and the network itself.

Clearly, introducing a layer above TCP/IP comes at the cost of service granularity — it will not be possible to specify an interface that covers each and every possible service and ideally exploit all available mechanisms without actually knowing which ones are available. On the other hand, this appears to be the only viable solution to the aforementioned long-standing deployment problems that may otherwise never be solved.

The fact that applications could transparently be forced to use a congestion-aware transport protocol, and use new services as they become available — much like SCTP-based FTP would benefit from multi-homing without requiring to actually implement the feature at the application level — makes it possible to incrementally enhance implementations of the middleware and thereby automatically further the quality attained by users.

Related research would encompass the “Congestion Manager” [18] — the idea of realizing a single congestion control instance for a single pair of communicating hosts no matter how many applications there are — and the question whether it would suffice to statically choose the right service or whether a middleware should dynamically adapt to changing network characteristics. While the latter would probably be more efficient, it brings up a few new questions. For instance, if the middleware switches between several TCP-friendly congestion control mechanisms depending on the state of the network, is the outcome still TCP-friendly? Despite these open problems, our proposal seems to be a promising and new way of realizing congestion control.

Since the middleware could be aware of the network infrastructure, it could also make use of TCP-unfriendly but more efficient transport protocols such as XCP, and CADPC/PTP — an overview of such alternatives can be found in [19].

Deployment of such schemes is a complicated issue in itself — clearly, there is a lot of need and scope for research in this area. For instance, any congestion control scheme requires feedback, and we have not decided how to realize this function in our middleware yet. We intend to tackle the issues one by one in the context of the project ”Tailor-made Congestion Control”; our results will continuously be made available to the public via the accompanying web page. [5]

[5] `http://dps.uibk.ac.at/~sven/tcc/`

References

[1] M. Fomenkov, K. Keys, D. Moore, k. Claffy, Longitudinal study of internet traffic in 1998-2003, Tech. rep., CAIDA, http://www.caida.org/outreach/papers/2003/nlanr/ (2003).

[2] V. Jacobson, Congestion avoidance and control, in: Proceedings of ACM SIGCOMM, 1988, pp. 314–329.

[3] S. Floyd, K. Fall, Promoting the use of end-to-end congestion control in the internet, IEEE/ACM Transaction on Networking 7 (4).

[4] S. Floyd, M. Handley, J. Padhye, J. Widmer, Equation-based congestion control for unicast applications, in: Proceedings of ACM SIGCOMM, 2000.

[5] R. Rejaie, M. Handley, D. Estrin, Rap: An end-to-end rate-based congestion control mechanism for realtime streams in the internet, in: Proceedings of IEEE Infocom, 1999.

[6] D. Sisalem, A. Wolisz, Lda+: A tcp-friendly adaptation scheme for multimedia communication, in: Proceedings of ICME, 2000.

[7] D. Bansal, H. Balakrishnan, Binomial congestion control algorithms, in: Proceedings of IEEE Infocom, 2001.

[8] J. Widmer, R. Denda, M. Mauve, A survey on tcp-friendly congestion control, IEEE Network Magazine: Special Issue Control of Best Effort Traffic 15 (3).

[9] E. Kohler, M. Handley, S. Floyd, Datagram congestion control protocol (dccp), internet-draft draft-ietf-dccp-spec-11.txt (March 2005).

[10] R. Stewart, Q. Xie, K. Morneault, C. Sharp, H. Schwarzbauer, T. Taylor, I. Rytina, M. Kalla, L. Zhang, V. Paxson, Rfc 2960: Stream control transmission protocol (October 2000).

[11] L. Coene, Rfc 3257: Stream control transmission protocol applicability statement (April 2002).

[12] W. chun Feng, M. K. Gardner, M. E. Fisk, E. H. Weigle, Automatic flow-control adaptation for enhancing network performance in computational grids, Kluwer Journal of Grid Computing 1 (1) (2003) 63–74.

[13] E. Weigle, W. chun Feng, A comparison of tcp automatic tuning techniques for distributed computing, in: Proceedings of High Performance Distributed Computing (HPDC), 2002.

[14] R. Jain, K. Ramakrishnan, Congestion avoidance in computer networks with a connectionless network layer: Concepts, goals, and methodology, in: Proceedings of Computer Networking Symposium, 1988.

[15] S. Floyd, J. Kempf, Rfc 3714: Iab concerns regarding congestion control for voice traffic in the internet (March 2004).

[16] M. Handley, J. Padhye, S. Floyd, Rfc 2861: Tcp congestion window validation (June 2000).

[17] L.-A. Larzon, M. Degermark, S. Pink, L.-E. Jonsson, G. Fairhurst, Rfc 3828: The lightweight user datagram protocol (udp-lite) (July 2004).

[18] H. Balakrishnan, S. Seshan, Rfc 3124: The congestion manager (June 2001).

[19] M. Welzl, M. Goutelle, E. He, R. Kettimut, S. Hegde, Y. Gu, W. E. Allcock, A survey of transport protocols other than standard tcp, gGF draft (work in progress) (February 2004).