A Stateless QoS Signaling Protocol for the Internet

Michael Welzl
University of Linz
Telecooperation Department
Altenberger Str. 69, A-4040 Linz, Austria
michael@tk.uni-linz.ac.at

Abstract

We describe a simple protocol that enhances the communication between end nodes and "the network". Other than the majority of QoS signaling systems, it achieves scalability by avoiding per-flow state. We also show how it can be used to decrease an adaptive multimedia application's packet loss ratio.

1. Introduction

The traditional black-box model of the Internet is not fit for today's world of heterogeneous services ranging from e-mail to distributed real-time multimedia applications. It remains an open question whether the continuously growing demand for QoS provisioning mechanisms will ever be satisfied; strict guarantees can hardly be given without violating Internet fundamentals. Adapting to the network's condition is a way out of this dilemma: it is a network-friendly method to enhance the quality perceived by the user — a tradeoff between bandwidth, delay and packet loss, adjusted to suit the user's requirements. The disadvantage is that being network-friendly may well result in a service that is just not good enough.

There are several reasons for this. One may be that there is too much other traffic disturbing the transmission. Another one certainly is the fact that there is not much support from the network: Simply put, adaptive applications are expected to "handle the situation on their own". If we take a look at some of the milestones in the evolution of Internet congestion control, we can make out a trend towards more deliberate information exchange between the layers three and four:

- In October 1986, the Internet experienced the first of what turned out to be a series of "congestion collapses"; the load became too much for the network [4].
- In 1988, the legendary paper "Congestion Avoidance and Control" summed up the changes that were made to TCP since 1986, including the well-known *Slow-Start* algorithm [4]. It represents the first major step towards adapting to the network: Packet loss got interpreted as a sign of congestion and forced TCP senders to reduce the data rate, which led to an overall Internet performance boost. Although it has undergone several changes since then, this is basically still the way TCP works.
- TCP's flow control got then supported by a complementary mechanism from within the network, namely *active queue management*[2]. The most popular active queue management algorithm is *Random Early Detection (RED)*, which makes the decision whether a packet should be dropped depend on the average queue length rather than solely queue overflows, thereby increasing the rate of packet drops when congestions occur and forcing TCP senders to adapt more quickly [3]. RED can be regarded as a reply to TCP's behaviour. As such, it establishes some basic form of communication between routers and end nodes via these packet drops.
- Reference [7] turns this into "real" communication (not just as a result of action): It states that the black-box model is inappropriate for delay- or loss-sensitive applications and introduces *Explicit Congestion Notification (ECN)* to the Internet. RED now sets a bit in packets it would normally drop; this way, receivers do not necessarily have to suffer from early packet loss, but the congestion control algorithms followed at the end-systems are expected to be essentially the same as the response to a single dropped packet [7]. ECN was extended with a "Backwards" functionality (informing senders directly by sending back an ICMP "Source Quench" message) by an Internet draft which has expired by the time of writing. Backwards notifications and Source Quench have a long history of discussion among the Internet community — related ideas come

up every once in a while and are usually rejected because of the increased backwards traffic and problems with scaling.

From this point of view, a protocol to support such information exchange in an efficient and scalable way seems to be the next logical step.

2 The "Performance Transparency Protocol" (PTP)

The most important principles we followed when designing PTP were to make it simple and scalable. Scalability can be achieved by avoiding to put stress on routers and, more importantly, avoiding per-flow state in routers; in conformance with the end-to-end argument, state is kept at the end points of the network [8]. No per-flow state means no state merging as in RSVP and no need for multicast support. PTP packets can be multicast via IPv6, but the communication model used in PTP is unicast.

A server sends a PTP request which is addressed directly to a client and treated by routers on-the-fly. PTP is layered on top of IP so that it can be identified by looking at the IP header's "protocol" field. Additionally, the *Router Alert Option* is used [5]. As far as packet treatment in routers is concerned, the protocol comprises two mechanisms:

1. *Forward Packet Stamping:* Somewhat similar to ATM's "Explicit Rate Feedback" mechanism, where a "Resource Management" cell gets updated as it traverses the network, routers add the requested information to a packet and forward it along the path.

2. *Direct Reply:* Here, a packet contains only one dataset with values set by the sender according to its performance requirements. Each PTP-compliant router compares the values with its own; the first router that cannot support the specified service updates and returns the packet. Since the redirection may put some additional stress on routers, implementation of this method is optional.

The requests are identified via so-called *Content Types* and subject to IANA standardization. The information is supposed to aid in QoS provisioning and can therefore be expected to concern links. Since each router is connected to two links that will be used by the packet, one PTP-compliant router can make up for one non-PTP-compliant router. Fig. 1 shows how this mechanism works: A packet contains a special field called TTL_{Check} that must be set to the value found in the IP header's *Time To Live (TTL)* field by all routers on forwarding. In the figure, it is abbreviated as *TTL-C*, a small / capital letter represents the address of an

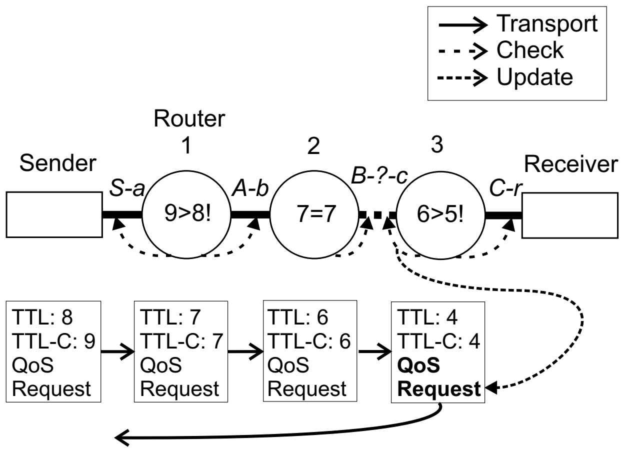

Figure 1. PTP's Direct Reply method

incoming / outgoing interface and the "?" in *B-?-c* stands for a single non-PTP-compliant router.

TTL is compared with TTL_{Check}; since TTL is reduced by at least one at each hop, $TTL_{Check} - TTL$ yields the maximum amount of routers the packet has encountered. If intermediate routers are detected, a router also uses information concerning the incoming link. In the case of Forward Packet Stamping, the receiver must be informed about the initial values so that the maximum amount of missing datasets can be calculated as

$$TTL_{Check} - TTL - DSCount \qquad (1)$$

(*DSCount* being the number of datasets). Depending on the data, the minimum or maximum value found in the datasets represents the "worst" service found along the path and should be used for further computations. If the information is incomplete (the result is not 0), it can be thought of as an upper limit for the path's capabilities. If, for instance, the information is incomplete and represents a link's nominal bandwidth, the available bandwidth can still be expected not to exceed the smallest value found in the datasets.

3 Performance Parameters

While PTP should be regarded as a generic QoS signaling protocol, its specification encompasses a number of predefined Content Types [11]. Using these Content Types, the most obvious QoS request can be made: *"What is the currently available bandwidth?"*

It is not easy to find an answer to this question. The definition of "currently" depends on the interval used, "available" does not necessarily mean "available to the application in question". By the time a reply returns, the situation may already have changed, so in fact, a prediction is needed. Reference [9] explains how such a prediction could be made; since we want to keep complexity at the network's end points, we leave the calculations up to the application and split the information request into parts. These parts are:

- A request for a link's nominal bandwidth $\mathcal{B}$ (defined as the bandwidth when there is no traffic at all) in bits per second in conjunction with the interface's address.
- A request for a traffic counter τ in conjunction with the interface's address and a timestamp. The traffic counters do not imply extra "work" as they are already available in the router's *Management Information Base (MIB)* [6].

While the available bandwidth corresponds with what is usually called a "QoS parameter", it would not make much sense to use a timestamp or traffic counters for application-level QoS specification. Therefore, we call this kind of information *performance parameters*. Since traffic counters cannot be compared without keeping per-flow state in routers, available bandwidth estimation can only be used with Forward Packet Stamping. We outline an example of the protocol's operation:

- The server sends a PTP packet with a request for a link's nominal bandwidth. This packet may additionally contain a request for traffic counters. All PTP-compliant routers along the path add their information to the packet.
- As it receives the packet, the client saves the information, determines the minimum nominal bandwidth $\mathcal{B}_{min}$ and sends it back to the server.
- The server now starts sending traffic counter request packets. To avoid overflows (the counters are 32-bit values), the time δ between two consecutive requests arriving at interface ι must not exceed

  $$\delta_{max} = 4294967296 * 8/\mathcal{B}_{\iota}. \quad (2)$$

  To compensate for end-to-end delay fluctuations, it is recommended to choose an interval i smaller than $4294967296 * 4/\mathcal{B}_{min}$.
- For each consecutive traffic counter request packet n and for all datasets j found in the packet, the client calculates the available bandwidth $\beta_{j,n}$ as

  $$\beta_{j,n} = \mathcal{B}_{j,n} - (\tau_{j,n} - \tau_{j,n-1})/\delta_{j,n} \quad (3)$$

  where $\delta_{j,n}$ can be calculated by subtracting the timestamp of dataset $(j, n-1)$ from the timestamp of dataset (j, n).
- If, in packet n, for all j, $\delta_j <= \delta_{max}$, the client reports the currently available bandwidth $\beta_n = min(\beta_{j,n})$ back to the server.

This procedure is path-sensitive and therefore begins anew if the number of datasets or the addresses change.

Network administrators may have their concerns about deliberately giving this information away; on the other hand, the information is already available via more bandwidth- and time-consuming methods [11].

4 Evaluation

We implemented PTP for the network simulator "ns-2" and simulated its behaviour with respect to available bandwidth determination across a transit-stub network with 100 nodes and a 3/2 mixture of Pareto traffic and FTP connections [12]. In order to see how PTP could be used, we implemented a "traditional" adaptive multimedia application which changed the bandwidth in discrete steps according to feedback and compared its behaviour with and without PTP.

Obviously, the available bandwidth obtained via PTP can not be interpreted as the bandwidth to be chosen for adaptation — if a router has three incoming lines and one outgoing line that is 50% "occupied", we cannot expect the other 50% to be available for our application. What we found is that if the data is interpreted as a trend, the percentage of lost packets can be drastically reduced.

Fig. 2 shows the bandwidth recorded by the receiver during one hour. In our case, i was twice the RTT and was used for the overall adaptation process. In the case without PTP, the application changed the bandwidth to the next multiple of the adaptation step below the measured bandwidth or increased it by one step if the results showed that the expected bandwidth was 100% utilized. In the PTP case, the application increased the bandwidth by one step if $\beta_n > \beta_{n-1}$ and decreased it by one step if $\beta_n < \beta_{n-1}$. Finally, we combined the methods by adding "normal" adaptation feedback and decreasing the bandwidth like we did in the case without PTP.

We found that the PTP feedback enabled us to build a very network-friendly application; in the example seen in Fig. 2 — the RTT was 3 seconds, the adaptation step was 5 kbit/s and the starting bandwidth was 56 kbit/s — using it reduced the ratio of lost bytes by 20%, while the combination of normal adaptation and PTP even reduced it by 42%. We ran various similar simulations with adaptation steps ranging from 1000 to 20000, i being half, once, twice or four times the RTT, packet sizes ranging from 100 to 5000 bytes, starting at various bandwidth levels including 0 and we found that the only way to make the PTP/adaptive combination lose more packets is to increase the amount of background traffic to a degree where most PTP packets get dropped.

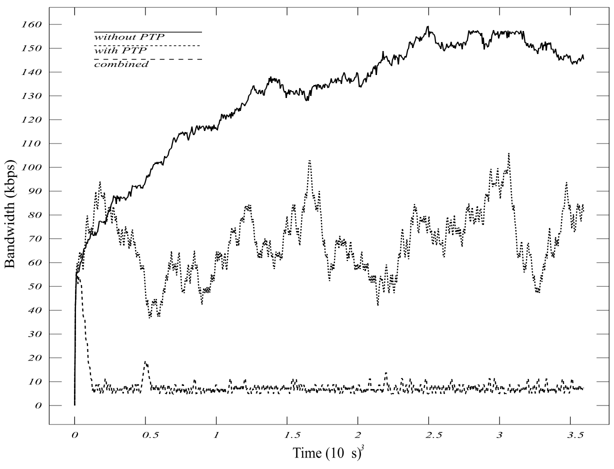

Figure 2. Bandwidth with PTP, without PTP and combined

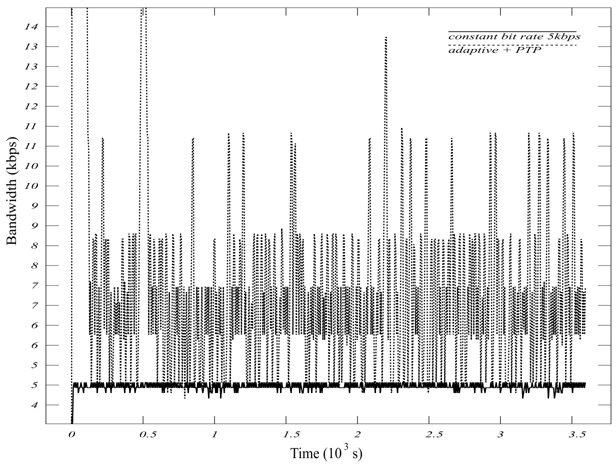

Figure 3. Bandwidth of a constant bit rate stream at 5kbps and adaptive+PTP.

It is important to be aware of the fact that the reduced packet loss ratio is not a plain result of PTP using less bandwidth; fig. 3 zooms in on our PTP/adaptive combination compared with a constant bit rate stream at 5 kbps. Although its behaviour *looks* more network-friendly, the constant bit rate stream's percentage of lost packets is 19.1% higher.

5 Conclusion

There is an obvious need for more communication between end nodes and the network; we proposed a lightweight, stateless and therefore scalable protocol to efficiently support this information. PTP has been proven to reduce the ratio of lost packets at the cost of reduced bandwidth, which is desirable for adaptive multimedia applications. In particular, we expect PTP to enhance the quality of voice over IP, where lost packets pose a big problem.

In our simulations, PTP was used in a very straightforward way; there are probably better, more sophisticated methods to use the average bandwidth feedback. Further, various other Content Types in support of well-known QoS parameters could be defined. We also expect that a combination of PTP and the *Real Time Protocol (RTP)*, which directly supports adaptive applications via RTCP "Receiver Reports", would be beneficial [10].

References

[1] Bolot, Jean-Chrysostome, "End-to-End Packet Delay and Loss Behavior in the Internet", *Proceedings of SIGCOMM 1993*, pp. 289-298. also in *Computer Communication Review* 23 (4), Oct. 1992.

[2] Braden, et al., "Recommendations on Queue Management and Congestion Avoidance in the Internet", *RFC 2309*, April 1998.

[3] Floyd, S., and Fall, K., "Promoting the Use of End-to-End Congestion Control in the Internet", *IEEE/ACM Transactions on Networking*, August 1999.

[4] Jacobson, V., and Karels, M. J., "Congestion Avoidance and Control", *Proceedings of SIGCOMM* 1988, pp. 314-329.

[5] Katz, D., "IP Router Alert Option", *RFC 2113*, February 1997.

[6] McCloghrie, K., and Rose, M.T., "Management Information Base for Network Management of TCP/IP-based internets: MIB-II", STD 17, RFC 1213, March 1991.

[7] Ramakrishnan, K., and Floyd, S., "A Proposal to add Explicit Congestion Notification (ECN) to IP", *RFC 2481*, January 1999.

[8] Saltzer, J.H., Reed, D.P., and Clark, D.D., "End-to-End Arguments in Systems Design", *ACM Transaction on Computer Systems*, 1984.

[9] Shu, Y., Jin, Z., Zhang, L., and Wang, L., "Traffic Prediction Using FARIMA Models", *Proceedings of ICC 1999*.

[10] Schulzrinne, H., Casner, S., Frederick, R., Jacobson, V., "RTP: A Transport Protocol for Real-Time Applications", *RFC 1889*, January 1996.

[11] Welzl, M., “The Performance Transparency Protocol (PTP)”, *Internet Draft*, draft-welzl-ptp-02.txt, available from http://www.ietf.org

[12] Zegura, Ellen W., Calvert, Ken, and Bhattacharjee, S., “How to Model an Internetwork”, *Proceedings of IEEE Infocom ’96*, 1996, San Francisco, CA.