The "best effort" service as a deployment success factor

Michael Welzl

Abstract

The "best effort" service model has certainly played a significant role for the success of the Internet – but in fact, the Internet only partially follows this model, and this has become a limiting factor for protocol deployment today.

Introduction: the importance of "best effort"

Internet routers "do their best" to deliver packets towards a destination, without making any promises about e.g. the delivery time or service rate. This behavior, which has been coined "best effort", allows for great flexibility – almost arbitrary strategies regarding e.g. routing, queuing, scheduling etc. can be locally chosen for each individual router without violating a service contract. This has given participants of the telecommunications market the necessary freedom in product differentiation ("my router is better than your router because it can do X-routing and Y-queuing"), resulting in the heterogeneous nature of today's Internet. However, certain parts of the Internet architecture have never followed the "best effort" model.

Case 1: Quality of Service (QoS)

QoS, by its very nature, is about giving promises. From the perspective of packets traversing the Internet, QoS mechanisms have been reasonably well defined. However, quoting [RFC 2990], *"for end-to-end service delivery it does appear that QoS architectures will need to extend to the level of the application requesting the service profile."*

Such extensions have not been standardized, which may be attributed to a general hesitance when it comes to APIs in the IETF. Had this work been done, the following dilemma would have become clear: either QoS usage is restrained to certain environments only, or – for the public "big I" Internet – the service model has to be "best effort". Here, "best effort" means that the system (e.g., the transport socket to which a request is made) can try to establish a certain form of QoS, but applications must be able to cope with a "no, QoS doesn't work, only best effort is available" situation.

With such an API that *provides QoS services on a best effort basis*, application programmers could (if the API is simple enough to use) make use of QoS straight away, when and where it is available. Possibly, providing such an API, and making it attractive enough by 1) keeping it simple and 2) demonstrating the benefits that can be attained with it, the following chicken-and-egg deployment problem from [RFC 2990] could have been prevented:

"No network operator will make the significant investment in deployment and support of distinguished service infrastructure unless there is a set of clients and applications available to make immediate use of such facilities. Clients will not make the investment in enhanced services unless they see performance gains in

applications that are designed to take advantage of such enhanced services. No application designer will attempt to integrate service quality features into the application unless there is a model of operation supported by widespread deployment that makes the additional investment in application complexity worthwhile and clients who are willing to purchase such applications."

It is interesting to note that now, some 15 years after the first QoS RFCs were published, application designers begin to realize "best effort QoS provisioning" themselves as they reach down to raw IP sockets and set a DSCP value directly, in the hope that certain routers or middle-boxes would somehow give them a slightly better service while not much harm would happen otherwise [draft-dhesikan-tsvwg-rtcweb-qos].

Case 2: transport protocols

Despite the design and deployment of a number of other transport protocols (SCTP, DCCP, UDP-Lite, ..), the transport service that application programmers can rely on has always been defined by two protocols: TCP and UDP. Most if not all of the services of the newer protocols are in fact semantically compatible with TCP and UDP. For example, SCTP provides faster data delivery for applications that can accept out-of-order chunk arrivals; certainly, such applications can also handle the ideal case of in-order delivery, which is provided by TCP at the expense of performance. Hence, if the transport API would provide a slightly more abstract service than what it offers today, offering out-of-order data delivery among other things, the operating system could provide these services "on a best effort basis" by trying to use e.g. SCTP and falling back to TCP or UDP (e.g. with a method like "Happy Eyeballs" [draft-wing-http-new-tech]) in case of failure.

Nowadays, such an API is not provided, and it is up to application programmers to implement such a fall-back mechanism themselves. This is not easy to do. It is also not necessarily clear that the potential gains would outweigh the effort, and hence SCTP is only used in special conditions where it is known that the protocol will work – or, as in rtcweb, it is applied in user space, over UDP, at the cost of application complexity and performance. A common alternative to such usage of e.g. SCTP is to simply construct the required service directly over UDP, inside the application – clearly, it would be a better situation if the right services would be made accessible via the transport API whenever they are available.

Conclusion: the way ahead

The "best effort" service model should be extended all the way to the application, by enriching the transport API with services of various transport protocols as well as QoS mechanisms *on a best effort basis* (i.e. with the possibility for the OS to fall back to standard TCP and UDP in case of a QoS-free Internet path). This would not only have the potential to improve the Internet's service via deployment of standards that are nowadays virtually unused, but also give OS deverlopers the freedom to implement complex service provisioning systems underneath the transport API. This would, in turn, allow for greater product differentiation among OS vendors, with potentially positive impacts on the OS market.

How to make this change happen is a bit harder to imagine. The idea is not new; the academic literature is full of similar suggestions, commonly following a "top down" approach where one starts with the question "what are the services that applications need?". The fact that so many different suggestions exist and nothing has led to a real change for the transport API that our applications use today indicates that such a "top down" approach may not be fruitful – making a real change needs a standard and implementations, and this requires an API design method that is based on strong consensus.

The author's suggestion is therefore to go "bottom up", i.e. starting with services that already standardized protocols and QoS mechanisms provide. The fact that these services exist means that there must have been preceding discussion about their need. For transport protocols, an example design based on such an approach is presented in [api], indicating that it is indeed possible to systematically arrive at a more generic, protocol-independent transport API. It is now about time to address this problem on a larger scale, in the IETF.

References

[RFC 2990]
G. Huston: "Next Steps for the IP QoS Architecture", RFC 2990, November 2000.

[draft-dhesikan-tsvwg-rtcweb-qos]
S. Dhesikan, D. Druta (ed.), P. Jones, J. Polk: "DSCP and other packet markings for RTCWeb QoS", Internet-draft draft-dhesikan-tsvwg-rtcweb-qos-02, 14 July 2013.

[draft-wing-http-new-tech]
D. Wing, A. Yourtchenko, P. Natarajan: "Happy Eyeballs: Trending Towards Success (IPv6 and SCTP)", Internet-draft draft-wing-http-new-tech-01, 20 August 2010.

[api]
Michael Welzl, Stefan Jörer, Stein Gjessing: "Towards a Protocol-Independent Internet Transport API", FutureNet IV workshop in conjunction with of IEEE ICC 2011, 5-9 June 2011, Kyoto, Japan.