How not to IETF: Lessons Learned From Failed Standardization Attempts

Michael Welzl
University of Oslo
michawe@ifi.uio.no

Jörg Ott
TU Munich
ott@in.tum.de

Colin Perkins
University of Glasgow
csp@csperkins.org

Safiqul Islam
Oslo Metropolitan University
safiqul.islam@oslomet.no

Dirk Kutscher
HKUST(GZ)
dku@ust.hk

***Abstract*—Protocol standards work is an interesting mixture of technical, political, financial, and human factors. Standardization processes require stamina as they may be lengthy, and they demand frustration resistance as they may hold surprises at all stages. While this certainly bears some similarity to academic endeavors, the need to build broader consensus and the potential of far reaching industry impact, among other factors, lead to different incentives and value systems. Peer review perspectives may also differ notably. In this paper, we discuss issues we came across in the past when trying to develop and advance technologies in the IETF or push presumed solid technology solutions towards standardization. We summarize our personal perspectives on the lessons learned.**

***Index Terms*—Standardization, Standards, Internet**

I. Introduction

For researchers working on network communications, the prospect of turning an idea into a standard can be alluring. Standards, when deployed, can have significant practical real-world impact. The Internet's main standards development organisation, the Internet Engineering Task Force (IETF), is a particularly attractive prospect for standardising the results of research, as it makes all materials freely available, allows open participation, and offers several ways to participate free of charge or at relatively low cost.

The authors of this article have contributed to the IETF in various ways over the last ∼25 years, holding a large variety of roles (document author, editor, working group chair, member of review directorate, etc.). As co-authors of 65 RFCs, we have had some success—but, naturally, we have also failed to standardize some of our ideas. Indeed, the IETF, as any (consensus-driven) organization, offers ample opportunity for failure, and one can fail for reasons that may seem unexpected, at least to academics. With "failure", we mean "failing to standardize" (publish as an RFC). It is not at all uncommon for accepted and published standards to "fail" in terms of deployment; such failures are out of scope of this paper, but discussed in [1]–[4].

It seems obvious that one needs to do the "homework". Approaching a standardization body with a suggestion that is not well thought out is bound to fail; such failure is unsurprising, and hence not very interesting. It is, however, quite possible to also fail with a proposal that may be (or, at least: appears to be, to the proponent) proven and tested, and technically sound. Here, we shed some light on this latter type of failure, in the hope that this can keep our peers from falling into certain less-than-obvious traps that we have fallen into.

In the next section, we present several of our failure stories, as examples from which we believe that lessons can be learned. We then summarize these lessons in section III. Section IV concludes.

II. Stories of Failure

We highlight our prior attempts to develop standards that ended in failure, covering transport and application protocols, moving from technical to less technical issues encountered.

A. TCP Corruption Notification Options

The presumption underlying this proposal was that, if bit errors happen in the payload, it is wrong for TCP to react by assuming that a packet was lost due to congestion, and hence a congestion control reaction is unnecessary and should be avoided. However, since TCP's checksum covers both the header and the payload, if corruption in the payload occurs, it is not even possible to correctly identify the host that should be notified of the error (as the port numbers may also be wrong).

The proposed fix, described in [5], keeps the original checksum unchanged, but attaches a new TCP option to the header, containing a checksum that covers only the header. If the "old" checksum fails but the "new" one does not, a bit error has occurred in the payload, and another new TCP option could be used by the recipient of the TCP segment to notify the peer of a corruption loss, provoking re-transmission but no congestion control reaction.

Balan et al. [6] have shown that this approach can be beneficial. This was done using a real-life implementation, where the Linux kernel was changed to support the necessary TCP options, and loss was artificially generated in the testbed.

1) Reason for failure: The major argument given upon presenting this proposal was that, in most practical real-life scenarios, errors do not occur in such a fashion. Instead, bit errors tend to affect blocks that are so large that they would always include the header, rendering this idea useless. When asked: "do you have proof?", the person with the counter-argument said "no, but do you have proof? You are the one proposing something"; a fair point indeed.

Later, we investigated how such errors really do play out in Wi-Fi networks. This involved disabling the link-layer CRC checksum (without tweaking the driver in this fashion, TCP

978-1-6654-5381-3/23/$31.00 ©2023 IEEE

would never even see any packets with errors). The results, reported in [7], confirm the suspicion of our IETF "opponent": the number of packets with payload-only errors was miniscule. For example, in an indoor test, approximately 1‰ of all packets were deemed corrupt and handed over by the driver, and 1% of these had errors in the payload only—30 out of 2.8 million packets.

Despite being limited to a specific link layer, these results did not exactly support our IETF proposal. We shared them with the IETF nevertheless, and received positive feedback for the very fact that we openly shared a negative result.

2) Take-aways: Simulations or artificial testbed scenarios are often not enough. Since a standard is supposed to operate in the real world, test the proposal under realistic conditions.

B. MulTFRC: TFRC with weighted fairness

TCP-Friendly Rate Control (TFRC) [8] is a well-known congestion control mechanism that is meant for multimedia content: it is TCP-friendly, i.e., it sends as much as TCP would under comparable circumstances, yet it exhibits a "smoother" behavior with less rate fluctuations. TFRC achieves this by calculating the well-known equation by Padhye et al. [9] that models TCP's long-term average throughput, and then sending data at the rate that the equation yields. We extended this model to N TCP connections, and applied it by replacing the equation in TFRC. The resulting mechanism, MulTFRC, can be tuned with a single parameter to be as aggressive as N TCP connections, with $N \in \mathbb{Q}_{>0}$. After publishing the equation and the MulTFRC mechanism [10], we decided to write a specification [11] and take it to the IETF.

1) Reason for failure: MulTFRC was reviewed in depth by several IETF peers, and generally quite well received. There was, however, one question that turned out to be devastating for the whole proposal: "how could this be implemented in the kernel"? Congestion control mechanisms were, at the time, routinely implemented in the kernel of Operating Systems such as Linux. Such kernels cannot handle floating point operations, requiring workarounds such as pre-calculated tables. We failed to adapt these workarounds to our model, and eventually dropped the ball on the MulTFRC IETF endeavor.

Today, it is not uncommon to implement congestion control mechanisms in user space (QUIC is a case in point [12]), and we could, in principle, try to proceed with MulTFRC—but "TCP-friendliness" is now an outdated concept. Instead, modern congestion control mechanisms for multimedia applications—such as the ones brought forward in the Real-Time Media Congestion Avoidance Techniques (RMCAT) IETF Working Group—strive to achieve a balance between competition with TCP on one hand, and latency minimization (via early congestion indicators such as delay growth) on the other.

2) Take-aways: Consider real-world implementation constraints for your algorithm, such as operating in the Linux kernel vs. in user space. Also, strike while the iron is hot.

C. LISA: A linked slow-start algorithm for MPTCP

Multipath TCP (MPTCP) [13] divides a TCP connection into multiple sub-flows, which are meant to traverse different paths. It has a form of "coupled congestion control" [14], where senders limit the total rate increase of sub-flows in Congestion Avoidance, so as to keep an MPTCP connection from being more aggressive than a single-path TCP connection. This prevents MPTCP from gaining an unfair advantage.

In the Slow Start phase, however, no coupling happens, and each subflow begins with its own TCP Initial Window (IW), which is now commonly set to 10 packets. This means that a single MPTCP connection with 4 subflows would transmit an initial burst of 40 instead of 10 packets, and this can harm other traffic and MPTCP itself. We have shown this problem in [15] and specified a solution—the "Linked Slow-start Algorithm" (LISA), which requires a new subflow to take a "credit" for its IW from the congestion window (cwnd) of an already existing subflow that is in Slow Start [16].

1) Reason for failure: The principles of MPTCP congestion control are laid out in RFC 6356 [14]: "resource pooling", "improve throughput", "do no harm" and "balance congestion". In response to the LISA proposal, we were informed about *another* principle, which is not documented anywhere: "the behavior of one subflow should not decrease the rate of another good[1] subflow". We showed that, due to the aggressive nature of Slow Start, the cwnd reduction caused by new subflows is not necessarily a disadvantage—even when the subflows do *not* share a bottleneck. Yet, the very idea of one subflow reducing the cwnd of another was a too strong divergence from the general behavior of MPTCP for the community to accept.

2) Take-aways: Get a thorough understanding of the underlying design principles of any mechanism or protocol that you are proposing to change. This concerns both written principles *and* unspoken rules. If you propose that protocol X should implement mechanism Y, then Y should be in line with what the designers of X intended (but, of course, knowing what they intended isn't necessarily straightforward).

D. Single-Path TCP Congestion Control Coupling

Combining the congestion control mechanisms of parallel TCP connections can yield several performance benefits. It eliminates competition between the flows, gives a large share of cwnd values to short flows to improve the overall start-up behavior and to skip Slow Start, fairly allocates available bandwidth, and reduces overall delay and loss. However, previous coupling mechanisms (e.g., the Congestion Manager [17]) have never been widely deployed because they require revamping the entire Internet protocol stack. To this end, we specified a lightweight congestion control coupling mechanism that combines the congestion control mechanisms of TCP connections when they traverse a same bottleneck [18]. We also proposed a lightweight, dynamically configured TCP-in-UDP (TiU) encapsulation scheme to enforce a common

[1]This was colloquial email text, where the word "good" was meant to indicate that the "other subflow" in question has not experienced congestion.

bottleneck by ensuring the same 5-tuple is used by multiple connections. We demonstrated the efficacy of our mechanism using ns2 and an implementation in FreeBSD [19], [20].

1) Reason for failure: We did not invest the effort into trying to get our code accepted in a major Operating System such as Linux (in which case, we may have tried harder to get this proposal standardized); instead, our hope was that our mechanism would be attractive enough for a hyperscaler to take on. However, such companies tend to run their servers load-balanced, on multiple machines, so that multiple TCP connections to the same client may not originate from the same physical device. Moreover, we were told that the need for TiU encapsulation gets in the way of hardware support for TCP such as TCP Segment Offloading (TSO).

2) Take-aways: Consider if the test environment matches the environment used by the target audience for the proposal (the IETF participants who should implement it). Consider also interoperability with current hardware.

E. Positive Acknowledgements for RTP Media Packets

The Real-time Transport Protocol (RTP) [21] and its original profiles for conferences with minimal control [22] were originally devised to support multicast-based conferences, in which information about multicast transport addresses for multimedia conferences would be shared, so that interested parties would join these addresses (e.g., via the multicast backbone, Mbone) to receive the corresponding media feeds, with security achieved by means of encryption. This led the design the of RTP Control Protocol (RTCP) to operate in a scaleable fashion and adjust the per-participant transmission of control information as a function of the observed group size, usually targeting to use no more than 5% of the total session data rate and a minimum interval of 2.5s between transmissions for control traffic.

This mechanism worked well for obtaining approximate group size information and sharing rough media reception (and thus "quality") statistics, deemed sufficient for larger groups. With the growing importance of voice-over-IP (VoIP) and thus point-to-point or small group calls, also including video, a demand arose to allow repairing packet loss. But the irregular transmission and especially the minimum transmission interval of the default RTP profile [22] made it effectively impossible to provide timely feedback to a media sender to immediately repair packet losses. As a remedy, a dedicated RTP profile was developed to enable more timely feedback [23], at least statistically for two-party calls and small conferences. This profile was designed with multicast in mind and hence provided mechanisms against feedback implosions. While numerous features for media feedback found broad support and made it into the final specification, published as RFC 4585 [23], one useful feature had to be removed: positive acknowledgements of received packets. Those would obviously bear the risk of ACK implosions if used in (larger) multicast groups, but the profile supported a dedicated point-to-point mode for faster feedback and simpler operation; yes, ACKs were even dismissed for this restricted usage. A similar mechanism was, however, found necessary to enable congestion control for real-time media and was only defined some 15 years later in RFC 8888 [24], after the advent of web-based real-time conferencing, *WebRTC*, suggested a large-scale uptake of multimedia conferencing, which required the development of real-time congestion control protocols, which ultimately took place in the RMCAT WG as noted above.

1) Reason for failure: The main cause for failure appears to have been of principled nature. The new RTP profile was not allowed *by default* to provide feedback once per RTT from the receiver to the sender (but the parameters for RTCP data rate could have been tuned in this way). Feedback per RTT would, however, be necessary to realize a suitable congestion control algorithm for real-time media. The fear was (or: appeared to be) that allowing lower-rate acknowledgments might lead implementers to mistake this feedback signal as suitable and sufficient for congestion control and thus support congestion control algorithms that could harm the Internet at large (as they, would, e.g., not be TCP-friendly [25], [26]). In hindsight, rather than removing ACKs altogether, it would have been more productive to understand in detail the concerns, and the real and perceived needs, and devise an extension to the same specification, or a complementary specification, that would specify exactly how to enable congestion control.

2) Take-aways: The main take-away is that principled issues are difficult to address. They require understanding the broader perspective, and developing possible mid- to long-term solutions to addressing the underlying concerns, rather than focusing on the specific technical issue at hand. This also requires "reading" the critics well and understanding their broader concerns, and being able to encourage them to assist in finding a way forward. This is doubly complex when parts of the community want a quick fix, to meet short-term business needs, while other parts raise broader issues of principle.

F. Session Description Protocol Next Generation (SDPng)

The Session Description Protocol (SDP) [27], [28] is used to communicate media descriptions of a multimedia session (such as a VoIP or video call) between peers. Such descriptions indicate which media codecs are to be used with which parameters, how media data is to be encapsulated into RTP, which RTP Profiles are used, which IP addresses and port numbers should be used by each peer. SDP was originally designed to minimally describe multicast-based multimedia conferences or distribution channels as used on the Mbone. But, with the advent of VoIP and the development of the Session Initiation Protocol (SIP) [29], [30] in the late 1990s, the *descriptive* nature of SDP and its interpretation was adapted to perform feature *negotiation* in the Offer/Answer model [29], [31]. This required not just a request-response style redefinition of its usage semantics but also more structure than a simple sequence of (key, value) pairs were able to deliver, so that sophisticated grouping, identification, and cross-referencing mechanisms were bolted on top of SDP. The simple design of SDP also led to conflating what functionality an endpoint *supports*, i.e., its *capabilities*, and which functionality an

endpoint intends to *use*, i.e., its initial media codec and transport *configuration*. Practical workarounds were found to address the most pressing issues within SDP, but the idea grew to define a more expressive replacement protocol.

SDPng [32] was devised to overcome the apparent shortcomings of SDP, offering structured descriptions based upon XML, which also supports grouping, naming, and referencing as well as a clean separation of capabilities, i.e., *potential configurations*, and choices for active use, termed *actual configurations*. The description language was first proposed in 2001 and revised repeatedly until 2005. In parallel, a transition document from SDP to SDPng was developed [33] to guide implementations moving from SDP to SDPng.

The SDPng proposal was formally adopted by the IETF Multiparty Multimedia Session Control Working Group, but did not receive sufficient industry support and the development was abandoned in the end. The SDPng transition document, for which even an RFC number—RFC 4637—was already allocated was never published, and RFC 4637 remains "not issued". Nevertheless, some concepts put forward in SDPng were picked up for SDP and documented as *Simple Capability Declaration* [34], also mechanisms for grouping and referencing media descriptions were developed for SDP [35].

1) Reasons for failure: While it is hard to pinpoint precisely the multi-faceted reasons for failure in retrospect, especially many years later, two aspects appear relevant. Firstly, SDPng used technology that was ahead of the time in IETF standards: XML as a data representation scheme. Industry stakeholders deemed XML too complex for the lightweight implementation needs of low-cost VoIP telephone devices. Interestingly, those critics overlapped with the ones supporting or developing the use of XML for further extensions to SIP not much later. Yet, XML may well have been a poor choice as it was clearly way more powerful than necessary to satisfy the needs at hand, thus violating the principle of using the least complex mechanism that suffices for a given task.

Secondly, beyond the choice of message encoding, SDPng was considered rather complex, while the promise of SIP (originally named SCIP, the *Simple* Conference Invitation Protocol) and SDP was to keep things simple, especially in contrast to the ITU-T efforts on H.323. The simple and easy to parse nature of SDP was in stark contrast to H.323's over-use of ASN.1, and SDP seemed to be good enough for what was needed at the time. Of course, "what was needed at the time" grew rapidly over the years, demanding further kludges to keep SDP functioning. The more of those got added, however, the less need was for a wholesale replacement by a next generation technology, ultimately rendering SDPng unnecessary.

One may add that the concept and details of SDPng itself evolved significantly during its IETF years, thus maybe not offering a sufficiently complete alternative from the outset. Looking back at the industry at the time, IP-based audiovisual communication, especially VoIP, was in the middle of the hype cycle and developing rapidly, so the industry players had a hard enough time keeping up with all the other SIP-related (standards) developments. This may explain the lack of interest or resources for exploring an unproven technology. SDPng never received sufficient buy-in from industry stakeholders.

2) Take-aways: Developing *next generation* specifications of a given technology may easily fall victim to competition with the legacy for many reasons, including that the new technology has unknown limitations while those of the existing version are known, and that there is experience with (and investment in) the legacy version. This experience allows for easily developing extensions that, over time, may marginalize the potential benefits of the next generation. Extending the present technology a bit at a time is often perceived to be simpler (and it indeed is for every single delta!) than embracing a radically different design, which leads to forces of inertia that must not be underestimated. Also timing is important: if the stakeholders don't have cycles for a refined technology, there won't be any buy-in.

G. Message Bus (MBus)

Multimedia conferencing systems may provide vastly different functions: audio and video communication, shared editing, and session control, among others. These features may be implemented in a stand-alone device or as code to run on any sufficiently powerful computer; in either case, following the separation of concerns, modules (with well-defined APIs) may implement these functions. The *Message Bus* [36], partly inspired by CCCP [37], was designed to allow flexibly interconnecting such modules via host/link-local multicast for a system design that could reach beyond a given physical device, allowing co-located devices and functions to form a coherent system. Mbus was developed partly in cooperation with industry, implemented, tested, and proven to work. Yet, it failed adoption in the IETF and was ultimately just published as an Informational RFC by the authors [36].

1) Reason for failure: Mbus received some pushback and not much support within the IETF. The pushback can partly be attributed to the encoding scheme chosen for Mbus: Scheme expressions of the style `method (arg arg arg)`, allowing for LISP-style nested structures and easy implementation of (key, value) pairs. This very notation, however, also could look like a function call to the casual observer, and there has been a strong opinion at the time that the IETF would not standardize APIs. Thus, choosing an unfamiliar encoding did not help getting people on board. More important was, however, the lack of industry support when Mbus was proposed: it was simply too early. As noted for SDPng, companies were busy specifying, following, and implementing SIP to get working products in the first place; concerns about advanced modularity (and functions) were likely not even on their radar.

2) Take-aways: Again, timing is everything. Moreover, a contributor should express herself in languages familiar to the target community—at a given time, as what is fashionable changes. People don't necessarily read specs but may quickly form—and articulate(!)—an opinion at first glance, no matter if their perception is right or wrong. This risk reduces if a spec language helps avoiding misunderstandings. This is even more important if tens of drafts compete for people's attention.

III. Lessons learned

After several decades experience working in the IETF, some successes, and many failures—some of which we describe in Section II—what have we learnt?

Firstly, that standards development can be an incredibly rewarding experience. The nature of this paper is that we focus on our failures, and try to learn from what didn't work. But we have also succeeded. The standards we have developed, with a wide range of industry collaborators, have been implemented and deployed in *billions* of devices and are in global use every day. As an academic researcher, if you want the results of your research to have real-world impact, standards development can be a very rewarding experience.

Secondly, we learn that standards development is *hard*. It's a slow and difficult process, that can be incredibly frustrating, and that often doesn't fit well with the dictates of an academic career. Standards development can take many years [1], [2], requires engagement with practical engineering and deployment concerns that get in the way of pure research, that no guarantee of success. In the following we explore these lessons and why they are both problematic and rewarding.

A. Understand Practical Deployment Concerns

The real-world Internet is a much larger and more complex system than any laboratory testbed or simulation. Participants in the IETF have long experience building and deploying protocols *at scale*, and hence are deeply familiar with issues that do not arise in the small-scale tests typically conducted by academic researchers. One of the benefits of engaging with the IETF is the ability to learn from this experience. As authors, we have received, and learnt from, detailed public explanations of why our research ideas won't work in practical deployments, and been forced to rethink and redesign our initial ideas. While such learning can be painful, we have all significantly benefited from these experiences. Systems research results are strengthened if they apply to the real-world, and not just to over-simplified experimental scenarios.

There are three key lessons around testbeds and deployment. Firstly, simulations or test-bed results are often not enough. Standards have to operate in the real world, so it's essential to test your proposals and ideas under realistic conditions, not just in idealised testbeds. You will be surprised at how often, and in what ways, the real Internet differs from simulations, controlled testbed experiments, and textbook examples.

Second, explore real-life implementation conditions, such as operating in the Linux kernel vs. user space. Consider whether your testing environment matches the deployment environment of the target audience—the IETF participants who should implement it—and to what extent those differences matter. Think about interoperability with current and developing hardware, programming models, and deployment realities.

Finally, ensure you have a thorough understanding of the underlying design principles of any mechanism or protocol that you are proposing to change. This concerns both the written principles *and* the unspoken rules. For better or worse, it is rare that an RFC tells the whole story about a protocol. Talk to developers of the RFCs, and those implementing and deploying the resulting standards, to understand what's really intended and how the RFC is used in practise.

B. Timing is Crucial

Getting industry engagement on the problem, and buy-in on the requirements analysis, is difficult when industry isn't (yet) experiencing the problem being solved. Academic work is supposed to be forward-looking and to address *future* problems. This makes it difficult to engage with the standards community when they are struggling to implement solutions to *today's* problems and customer needs. Solving problems before the industry realises they matter is the core of academic research, but can be a problem when engaging with the standards community.

This is tricky for academic research labs that need to have impact with a small team. You need to be early enough that the problem is apparent, and the community is small enough that your voice is heard, but not so early that the standards community doesn't yet understand the problem. Too early, and no-one cares about your ideas. Too late, and your PhD student is competing with the engineering might of a multinational corporation. We see this in our experience. Mbus was too early, solving a need industry did not yet have. In contrast, SDPng came right in the middle of the hype, when industry was fighting to keep-up with deploying what they had, even if it was clear that extensions would eventually be needed.

Alternatively, you may work with mature protocols, where the initial hype has dissipated and the protocol is widely used as an infrastructure component. This has the benefit that those implementing the protocol have calmed down enough to be willing to listen to outside input and ideas, but suffers because the full complexity of the network is now known and the deployment challenges are greater. Our experiences with TCP corruption notification and congestion control highlight this.

C. Abstractions, Generality, and Protocol Ossification

As academic researchers, we abstract and generalise. Those implementing products want something concrete and realisable today, and will worry about the broader issues later—if they're still in business. One of the problems we've seen several times is that, even when it's recognised that generalising the protocol will be necessary, persuading the standards community to spend the effort to look ahead is a difficult sell.

Equally, introducing hooks to build the more general framework early in the protocol design also may not work. Implementations tend to build what's needed today, leaving extension points unimplemented, ossified, and unusable in practise [38]. The SDPng transition is an example here.

The Internet only just works [39]. The protocols are a convoluted mess, patched together to solve numerous short-term issues, with limited effort spent to develop a coherent long-term architecture. This is *expected and normal*. The need to work with what is, rather than what should be, and to propose incremental improvements, that solve smaller problems than you desire, is a key lesson.

IV. Conclusion

The IETF is known as a somewhat intimidating environment, where harsh technical criticism is common. Thus, proponents of new ideas understand that they must first implement and evaluate their proposals before trying to standardize them. However, even when a proposal is sound, the IETF can present unwelcome surprises. For instance, as we have discussed, evaluations may need to be done differently than one might expect in a common academic setting—it is very important to consider the right environment, which may include considerations such as hardware support, kernel vs. user space implementation, and very realistic real-life network conditions. The notion of "realistic" can sometimes even be limited to deployment scenarios that are in use at the most influential industry players, as these might strongly oppose proposals that do not fit their conditions.

The IETF is a community in its own right—it is not simply an outlet that one should consider at the very last stage of development. Our advice is therefore to get involved in this community *early*, to gain an understanding of the direction of travel, the unspoken rules, and how people operate. We illustrate this point with an anecdote: at a recent academic conference, one of us was asked how to bring a specific proposal based on a well-accepted academic paper to the IETF. This proposal required to re-purpose a seemingly unoccupied bit combination in a protocol header. As a research idea, this makes sense. But, we were the first to inform the proponent that use of this particular bit combination had been under heavy debate in the IETF for the better part of a decade, and that an alternate proposal to use it had just succeeded.

Acknowledgements

Work supported, in part, by EPSRC grant EP/S036075/1.

References

[1] S. McQuistin, M. Karan, P. Khare, C. S. Perkins, G. Tyson, M. Purver, P. Healey, W. Iqbal, J. Qadir, and I. Castro, "Characterising the IETF through the lens of RFC deployment," in *Proceedings of the Internet Measurement Conference*. Online: ACM, Nov. 2021.

[2] P. Khare, M. Karan, S. McQuistin, C. S. Perkins, G. Tyson, M. Purver, P. Healey, and I. Castro, "The web we weave: Untangling the social graph of the IETF," in *Proc. ICWSM*. Atlanta: AAAI, Jun. 2022.

[3] M. Nikkhah, A. Mangal, C. Dovrolis, and R. Guérin, "A statistical exploration of protocol adoption," *IEEE/ACM Transactions on Networking*, vol. 25, no. 5, pp. 2858–2871, 2017.

[4] D. Thaler and D. B. D. Aboba, "What Makes for a Successful Protocol?" RFC 5218, Jul. 2008.

[5] M. Welzl, "TCP corruption notification options," Internet Draft draft-welzl-tcp-corruption-00.txt, Jun. 2004, work in Progress.

[6] R. K. Balan, B. Lee, K. R. R. Kumar, L. Jacob, W. K. G. Seah, and A. L. Ananda, "TCP HACK: TCP header checksum option to improve performance over lossy links," in *Proceedings of IEEE Infocom*, 2001.

[7] M. Welzl, M. Rossi, A. Fumagalli, and M. Tacca, "TCP/IP over IEEE 802.11b WLAN: the challenge of harnessing known-corrupt data," in *Proceedings IEEE ICC*. Beijing, China: IEEE, May 2008.

[8] S. Floyd, M. Handley, J. Padhye, and J. Widmer, "Equation-based congestion control for unicast applications," in *Proceedings of the SIGCOMM Conference*. ACM, 2000.

[9] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose, "Modeling TCP Reno performance: a simple model and its empirical validation," *IEEE/ACM Transactions on Networking*, vol. 8, May 2000.

[10] D. Damjanovic and M. Welzl, "An extension of the TCP steady-state throughput equation for parallel flows and its application in MulTFRC," *Networking, IEEE/ACM Transactions on*, vol. 19, no. 6, p. 1, Dec. 2011.

[11] M. Welzl, D. Damjanovic, and S. Gjessing, "MulTFRC: TFRC with weighted fairness," Internet Draft draft-irtf-iccrg-multfrc-01, Jul. 2010, work in Progress.

[12] J. Iyengar and I. Swett, "QUIC Loss Detection and Congestion Control," RFC 9002, May 2021.

[13] A. Ford, C. Raiciu, M. J. Handley, O. Bonaventure, and C. Paasch, "TCP Extensions for Multipath Operation with Multiple Addresses," RFC 8684, Mar. 2020.

[14] C. Raiciu, M. J. Handley, and D. Wischik, "Coupled Congestion Control for Multipath Transport Protocols," RFC 6356, Oct. 2011. [Online]. Available: https://www.rfc-editor.org/info/rfc6356

[15] R. Barik, M. Welzl, S. Ferlin, and O. Alay, "LISA: A linked slow-start algorithm for MPTCP," in *2016 IEEE ICC*, May 2016.

[16] R. Barik, S. Ferlin, and M. Welzl, "A Linked Slow-Start Algorithm for MPTCP," Jun. 2016, work in Progress.

[17] H. Balakrishnan and S. Seshan, "The Congestion Manager," RFC 3124, Jun. 2001. [Online]. Available: https://www.rfc-editor.org/info/rfc3124

[18] M. Welzl, S. Islam, K. Hiorth, and J. You, "TCP-CCC: single-path TCP congestion control coupling," Oct. 2016, work in Progress.

[19] S. Islam and M. Welzl, "Start me up: Determining and sharing TCP's initial congestion window," in *Proceedings of the 2016 Applied Networking Research Workshop*, 2016, pp. 52–54.

[20] S. Islam, M. Welzl, K. A. Hiorth, D. Hayes, G. Armitage, and S. Gjessing, "ctrlTCP: reducing latency through coupled, heterogeneous Multi-Flow TCP congestion control," in *Proceedings of the IEEE Global Internet Symposium*, Honolulu, USA, Apr. 2018.

[21] H. Schulzrinne, S. L. Casner, R. Frederick, and V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications," RFC 3550, Jul. 2003.

[22] S. L. Casner and H. Schulzrinne, "RTP Profile for Audio and Video Conferences with Minimal Control," RFC 3551, Jul. 2003.

[23] C. Burmeister, J. Rey, N. Sato, J. Ott, and S. Wenger, "Extended RTP Profile for Real-time Transport Control Protocol (RTCP)-Based Feedback (RTP/AVPF)," RFC 4585, Jul. 2006.

[24] Z. Sarker, C. Perkins, V. Singh, and M. A. Ramalho, "RTP Control Protocol (RTCP) Feedback for Congestion Control," RFC 8888, Jan. 2021. [Online]. Available: https://www.rfc-editor.org/info/rfc8888

[25] S. Floyd, M. Handley, J. Padhye, and J. Widmer, "Equation-based congestion control for unicast applications," in *Proc. SIGCOMM*. Stockholm, Sweden: ACM, Aug. 2000.

[26] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose, "Modeling TCP throughput: A simple model and its empirical validation," in *Proc. SIGCOMM*. Vancouver, Canada: ACM, Aug. 1998.

[27] C. Perkins, M. J. Handley, and V. Jacobson, "SDP: Session Description Protocol," RFC 4566, Jul. 2006.

[28] A. C. Begen, P. Kyzivat, C. Perkins, and M. J. Handley, "SDP: Session Description Protocol," RFC 8866, Jan. 2021.

[29] H. Schulzrinne, E. Schooler, J. Rosenberg, and M. J. Handley, "SIP: Session Initiation Protocol," RFC 2543, Mar. 1999.

[30] E. Schooler, J. Rosenberg, H. Schulzrinne, A. Johnston, G. Camarillo, J. Peterson, R. Sparks, and M. J. Handley, "SIP: Session Initiation Protocol," RFC 3261, Jul. 2002.

[31] H. Schulzrinne and J. Rosenberg, "An Offer/Answer Model with Session Description Protocol (SDP)," RFC 3264, Jul. 2002.

[32] D. Kutscher, J. Ott, and C. Bormann, "Session description and capability negotiation," Internet Draft draft-ietf-mmusic-sdpng-08, Feb. 2005.

[33] C. Perkins and J. Ott, "Sdpng transition," Internet Draft draft-ietf-mmusic-sdpng-trans-04, May 2003.

[34] F. Andreasen, "Session Description Protocol (SDP) Simple Capability Declaration," RFC 3407, Oct. 2002.

[35] H. Schulzrinne, G. Camarillo, G. A. Eriksson, and J. Holler, "Grouping of Media Lines in the Session Description Protocol (SDP)," RFC 3388, Dec. 2002. [Online]. Available: https://www.rfc-editor.org/info/rfc3388

[36] C. Perkins, D. Kutscher, and J. Ott, "A Message Bus for Local Coordination," RFC 3259, May 2002.

[37] M. Handley, I. Wakeman, and J. Crowcroft, "The conference control channel protocol (CCCP): a scalable base for building conference control applications," *ACM SIGCOMM CCR*, vol. 25, 10 1995.

[38] M. Thomson and T. Pauly, "Long-Term Viability of Protocol Extension Mechanisms," RFC 9170, Dec. 2021.

[39] M. Handley, "Why the Internet only just works," *BT Technology Journal*, vol. 24, no. 3, pp. 119–129, Jul. 2006.