Congestion-Control Throwdown

Michael Schapira
Hebrew University of Jerusalem
schapiram@huji.ac.il

Keith Winstein
Stanford University
keithw@cs.stanford.edu

ABSTRACT

Congestion control is a perennial topic of networking research. In making decisions about *who* sends data *when*, congestion-control schemes prevent collapses and ultimately determine the allocation of scarce communications resources among contending users and applications.

The field has seen considerable recent activity. Even after three decades of research, basic principles and techniques remain up for debate. In this throwdown-as-paper, the authors find themselves at loggerheads over the fundamental tenets of congestion control.

1 INTRODUCTION

After attending a research conference, the authors find themselves seated next to each other on a flight home to their respective employers.

Hamilton: What a coincidence that we ended up sitting together! I want to tell you about our vision for congestion control—it's pretty wild.

Burr: Yes, we keep meeting. I think it's only fair to tell you that I recently read a draft on this topic that sounds like it was yours, and it drove me to distraction in that way that only the topic of congestion control can do.

Hamilton: Well, we're strapped in, the seat-belt light is on, and you've got nobody else to talk to on this airplane—looks like we're stuck working this out. Let me try to make the case.

2 HAMILTON'S OPENING STATEMENT

First, let me tell you how I perceive the differences between three recently proposed paradigms for rate-control on the Internet, namely, Remy [8, 10], PCC [3], and BBR [1].

2.1 Remy *vs.* PCC *vs.* BBR

Remy takes as input *explicit assumptions* about the network, such as ranges of wire speeds, RTTs, number of senders on bottleneck links, and also the designer's *global optimization* objective, say, proportional fairness. Remy then generates a *model of the network* and seeks a "good" mapping from observed network state (average of packet ACKs inter-arrival times, ratio of current RTT and minRTT, etc.) to control actions (such as a multiplier/increment to the congestion window). BBR's design philosophy is different; BBR models the network pipe as a single link, repeatedly probes the bandwidth and RTT, and paces the rate so as to track the bottleneck link's bandwidth. Lastly, PCC continuously associates the sending rate with a numerical *utility* value that reflects a *local* performance objective (say, "high throughput and low loss rate"). This is accomplished through a *utility function*, which aggregates experienced performance-related statistics (e.g., goodput, loss rate, and latency) into a numerical value. PCC's control algorithm adapts the sending rate in the direction (faster/slower) that empirically yields better utility for the sender.

I see two fundamental distinctions between these three paradigms:

- **Reliance on a network model.** Remy takes as input *a priori* assumptions about the network and generates a model of the network, BBR generates a model of the network pipeline in an *online manner*, whereas PCC does not rely on a network model. In this sense, Remy and BBR both represent *white-box* approaches, whereas PCC takes a *black-box* approach.
- **Global vs. local optimization.** Remy explicitly aims to reach a *global* optimum. BBR and PCC, in contrast, optimize *local* performance.

I argue that designing rate-control schemes that *robustly* achieve high performance across a variety of network environments and, in particular, in the wild wild Internet, involves coming to terms with certain hard truths.

2.2 Generating an accurate network model of the Internet might not be feasible

The Internet is remarkably complex, maybe too complex for white-box approaches. Traffic flows enter and leave, network devices and links might fail, packets might traverse different communication media / organizations / numbers of hops, different end-hosts employ different rate-control protocols, latency increases and packet losses can result from temporary microbursts or PHY-layer corruption (as opposed to congestion), etc. Unless the network exhibits very high stability and predictability (as in, arguably, highly optimized WANs), which is rarely the case, *any* network model is likely to be

HotNets-XVI, November 30–December 1, 2017, Palo Alto, CA, USA

ACM ISBN 978-1-4503-5569-8/17/11…$15.00
https://doi.org/10.1145/3152434.3152446

highly inaccurate and, consequently, not a solid base for decision making. In fact, even when designers of white-box approaches argue for why these might fare well in practice despite the huge gap between their models and reality, the argument eventually boils down to empirical evidence [8].

Black-box approaches, in contrast, do not generate a model the network, but instead *seek good mappings from empirically-observed performance to changes in sending rate*. I believe that this design philosophy can provide inherently higher robustness to variability in network conditions.

2.3 You cannot reach a global optimum at Internet scale

Different applications have different performance requirements, typically unbeknownst to the rate-control protocol. Even defining the "right" global optimization objective is not obvious. Worse yet, even if such a notion exists, optimizing the objective function relies on the assumptions that (1) *all* end-hosts employ the prescribed protocol (and, possibly, even that all routers use the same queuing policy); and (2) traffic flows persist long enough for global optimization to be meaningful.

I believe that giving up on the aspiration of explicitly optimizing a metric of global network performance in the wild Internet is an unfortunate, yet unavoidable, consequence of the Internet's complexity. Instead, rate-control design should focus on achieving other desiderata such as optimizing local performance goals, fairness on bottleneck links, fast convergence to equilibrium for persistent flows, quick utilization of spare capacity, and more.

2.4 Rate-control via online learning is a promising direction

Out of the three rate-control schemes I mentioned, only PCC is both black-box *and* does not *explicitly* optimize global performance. I believe, however, that while PCC is a promising paradigm for next-generation congestion control, the specific manifestation of this paradigm in [3] is far from realizing its full potential. Now, let me tell you about a research direction that I am very excited about: applying ideas and machinery from the rich body of research on *online learning* in learning theory and game theory to the context of rate-control.

Online learning (a.k.a. "no-regret learning") provides a useful and powerful abstraction for decision making under uncertainty. In the online learning setting, a *decision maker* repeatedly selects an action from a set of available *strategies*. Only after selecting a strategy does the decision maker become aware of the implications of selecting that strategy, in terms of a resulting *utility* value. I posit that online-learning algorithms are particularly appropriate for rate-control. Specifically, online learning algorithms provide provable guarantees (namely, "no regret" [2, 4]) even under *complete* uncertainty about the environment, i.e., without assuming/inferring *anything* about the relation between choices of strategies and the induced utility values. Moreover, online learning algorithms "play well" together, in the sense that (under the appropriate conditions) global convergence to a stable outcome (equilibrium) is guaranteed when multiple decision makers apply online learning.

Rate-control can easily be cast as an online learning challenge: A traffic sender repeatedly selects sending rates. After sending at a certain rate and waiting "long enough," the sender learns the implications for performance of sending at that rate by translating aggregated statistics (e.g., achieved goodput, packet loss rate, average latency) into a numerical utility value, as in PCC (see [3] for details). Instead of the ad hoc choice of utility function and the fairly naive rate-control scheme used in [3], I argue that a principled approach, informed by online learning theory, should be applied. Preliminary experimentation with state-of-the-art online-learning-based rate-control schemes suggest that these outperform TCP, BBR, and PCC's realization in [3], in terms of achieved throughput and latency across highly variable network environments, adapting to rapidly changing network conditions, convergence rate and stability, and beyond.

So, what do you think?

3 BURR SPEAKS IN OPPOSITION

Hamilton, despite our disagreements, I couldn't have asked for a better colleague to be strapped down next to in an airplane. And I appreciate your efforts to frame a taxonomy of recent approaches to wide-area congestion control: BBR, PCC, and Remy. But (you knew there was a "but"), your view of the world doesn't make a whole lot of sense to me!

Let me make a few points in reply:

3.1 Local optimization = lousy outcomes

As I understand it, the innovation of PCC is that each flow explicitly performs *online* optimization of an objective function. This is distinguished from traditional TCP congestion-control algorithms (NewReno, Vegas, FAST, Cubic, the RemyCCs, etc.), where a human (or in the case of Remy, a computer program) first thought really hard *offline* to design a good congestion-control algorithm, and then deployed that algorithm on endpoints.

In PCC, because the optimization is online, the objective function necessarily only includes inputs that are locally perceptible to the individual flow. Two signals are considered: (1) how many bits per second is that flow sending, and (2) how many of those bits per second are making it through to that flow's receiver (i.e., the throughput).

My view: distributed hill-climbing of a locally perceptible utility function is not a sound foundation for Internet congestion control. To explain why I think this, let me bring up one of the most old-fashioned and classical congestion-control problems there is: a symmetric parking-lot topology (Figure 1). And to make it really traditional, I want to start with the assumption that all three flows are long-running and that if a link is oversubscribed, it simply drops packets proportionally to reduce its utilization to 100% (no queueing).

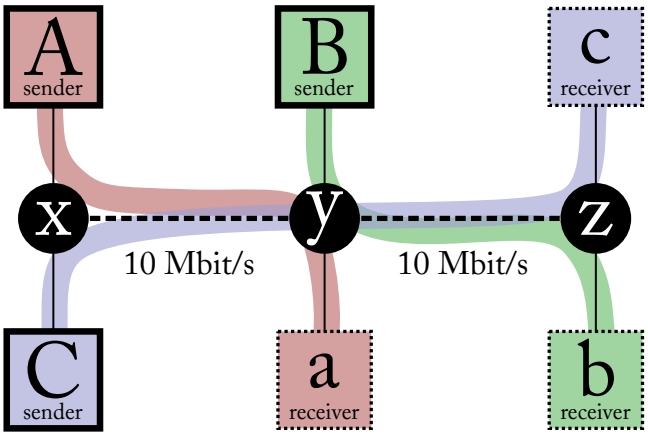

Figure 1: A parking-lot topology [9]. Routers x and y are connected by a 10 megabit-per-second link, and so are y and z. Long-running flows A and B each cross one of the links; flow C crosses both links. We'd like to deploy the same congestion-control algorithm at each of the three senders and have the network function sanely.

There's no one right answer to the question of how this network's resources should be divided among the flows. There is a family of Pareto-efficient fair solutions. The max-min fair solution has A, B, and C each getting 5 Mbit/s. The proportionally fair solution recognizes that C contends with twice as many flows of cross-traffic as A and B do, and so gives C half the allocation ($3\frac{1}{3}$ Mbit/s) of A and B ($6\frac{2}{3}$ Mbit/s each). The "max total throughput" solution simply gives A and B 10 Mbit/s each, leaving C with nothing.

Each of these solutions corresponds to maximizing a global total of each flow's individual utility, for various utility functions. For example, the proportionally fair solution maximizes $\log A + \log B + \log C$, where each flow's individual utility is the log of its own throughput.

Except: it wouldn't make sense to have each flow set its sending rate by *locally* maximizing its own utility function. If the objective is log(*throughput*), then each flow is just going to flood as fast as possible, in order to get the biggest possible share of the contended 10 Mbit/s links. The faster a flow's competitor sends into a congested link, the faster the flow itself needs to send in order to preserve its share. There's a tragedy of the commons in "online" optimization in this way.

Ah, you say, but this is not how PCC works. PCC has an objective function that penalizes loss *even if throughput is good*, so a flow will not want to flood the network with a huge sending rate just to eke out a little more throughput. The objective is $t \cdot \left[1 - \frac{1}{1+e^{-100(L-0.05)}}\right] - s \cdot L$, where s is the sending rate, t is the throughput, and $L := 1 - \frac{t}{s}$ is the loss rate.[1]

[1]This is the flow's optimization objective, but the presence of the loss penalty makes it, to me, not really a "utility" function. An application using TCP (with PCC as the congestion control) will have no idea and no reason to care what the underlying IP datagram loss rate is; TCP's job is to retransmit lost segments anyway. If PCC delivers good throughput to the application, that ought to be enough. Datagram loss itself is none of the application's business, and therefore none of a utility function's business. The need for this kind of penalty term limits PCC's agility to swap in a different function, because the designer needs to make sure that whatever function they choose has a term like this so it won't create a tragedy of the commons.

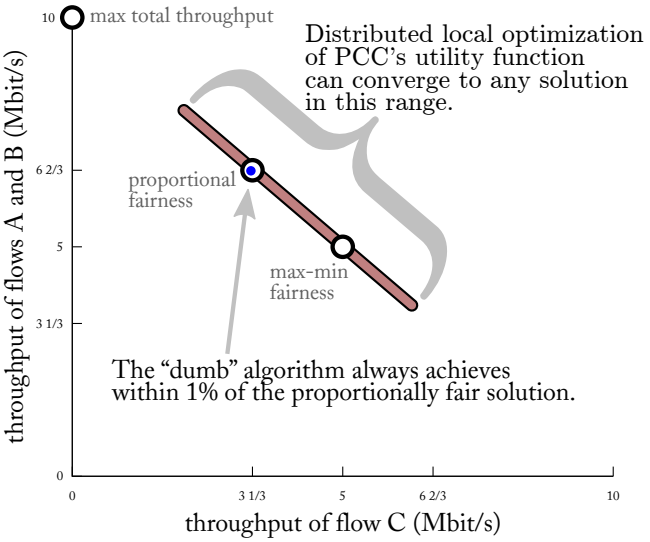

Figure 2: Convergence region of distributed optimization of PCC's objective function (red shaded strip) compared with a "dumb" algorithm on the topology of Figure 1. PCC always converges to something, but is sensitive to initial conditions—it can end up getting stuck in an allocation where flow C receives as little as 2.1 Mbit/s (upper-left of red shaded region) or as much as 6.3 Mbit/s (lower-right)—a threefold range. By contrast, a simple AIMD scheme consistently achieves close to the proportionally fair allocation (blue dot), no matter which flow starts first or at what rate.

There is great attraction to a design where we can say, "forget trying to design a good congestion-control scheme offline; simply specify an objective function and each flow will maximize it online, making no assumptions about the network and having no regrets." A lot of effort goes into designing these schemes; if they can be replaced by online maximization of an objective function, that would be awesome.

Unfortunately, this seems to me too ambitious, even for simple network scenarios. I performed a numerical experiment with an idealized PCC in the topology of Figure 1. The three flows start with a sending rate between 0 and 11 Mbit/s. At each step of the simulation, one of the flows tries to improve its situation by differentially adjusting its sending rate until it can no longer find an improvement in its objective function (akin to PCC's "micro-experiments"). This proceeds in round robin until all three rates have reached a fixed point where no scheme can improve by changing its rate up or down.[2]

The good news is that PCC always converges to *some* stable allocation of rates. The bad news is that the allocation it finds (and then preserves for eternity) can vary widely, depending on the initial rates or which flow starts first, even though they all last forever once they get started. The three flows

[2]Source code to replicate these results is at https://github.com/keithw/netsolve. The calculations are done in double-precision floating point, and the simulation is judged to have reached a fixed point when no flow wants to adjust its rate by more than $\pm 10^{-7}$ Mbit/s.

can stabilize in a solution where A=7.9, B=7.9, C=2.1, or a solution where A=3.7, B=3.7, C=6.3. Both of these, as well as every allocation in between, are fixed points of the distributed local optimization algorithm. Reasonable people can disagree about the "right" allocation of resources in this topology, but I don't think you can persuade me that a good scheme can be so sensitive to "which flow started first" that your own flow's allocation should vary by a factor of 3 for the rest of eternity.

The problem here is that each flow is only seeing its *own* throughput—it has no idea how happy or unhappy the other flows are, and once the three flows stabilize, there is no more loss and therefore no more interesting congestion signals to learn from. None of the flows has any cause for regret (there is no better rate for any of them to have chosen individually), so your emphasis on no-regret learning as the future of congestion control seems like a red herring.

Ultimately, when congestion-control decisions are decentralized (as they generally are outside of single-owner networks), I suspect that a good congestion-control scheme can't simply optimize a locally perceptible objective function. Call this **Burr's conjecture**: it is impossible for a decentralized congestion-control scheme to be globally asymptotically stable (never mind the quality of the outcome!) over a network with "dumb" bottlenecks (e.g. DropTail queues), if it operates by greedily optimizing an objective function whose only input is the fate of its own traffic (when were packets sent, and which arrived and when).[3]

Which is not to say that good congestion-control *algorithms* don't exist—only that good algorithms behave in a more sophisticated manner than simply optimizing a locally perceptible objective function. Good schemes must balance exploitation (of a current situation) with exploration (behavior that might be against the flow's self-interest in the short term, because of the chance that the status quo might not be good for other more-deserving traffic). The place where we design such algorithms is offline.

Consider, for example, a simple rate-based AIMD scheme. The scheme sends at a particular rate. At equal time intervals, it adjusts the rate based on whether it has seen any losses during the most recent interval. If there hasn't been any loss, it increases the rate by 10 kbps. Otherwise, it decreases it by a percentage between 0 and 1% (chosen uniformly at random).

This the dumbest of dumb schemes. The individual flows don't online-optimize any locally-measurable function, but they do have a pattern of exploration and exploitation behavior that was designed in upfront. And it turns out that on our classical congestion-control problem, the scheme performs a lot better than the idealized PCC. If run on the three flows in Figure 1, this scheme will achieve within 1% of the proportionally fair solution A=B=6.7, C=3.3 (shown with arrow and blue dot on Figure 2) over the long term, *no matter which flow started first or at what rate.* The dumb scheme does what it was designed—offline—to do. TCP Reno/Vegas/Cubic and the outputs of the Remy tool all represent attempts at coming up with these "dumb" schemes that yield good overall outcomes under an arbitrary pattern of flow arrivals.

Bottom line: just because a congestion-control scheme has each flow independently optimize a locally perceptible objective function, and just because that objective function is engineered so that this doesn't lead to a tragedy of the commons, and just because the scheme converges to some allocation of rates where no flow has reason for regret, we can't conclude that it's a good scheme.

3.2 Every scheme embodies assumptions about the network

To me, then, the burden is on those who propose "online" techniques to demonstrate that they work better in a realistic setting than in a toy example. If you don't think my classical parking-lot network is relevant, then the argument necessarily involves some statement of opinion about what a realistic network and workload look like. That's stating a model or an assumption!

Why don't we use the dumb scheme? One reason is that in real life, flows come and go, and convergence time matters. The tiny AI and MD constants (0–1% decrease if loss, 10 kbps increase if not) are great for achieving within 1% of the proportionally fair allocation over the long term, but bad for converging quickly, and terrible for allowing in newly joining flows when the network is already fully utilized. Those constants embody a designer's expectations about the networks and scenarios the scheme will face: how quickly will flows arrive or depart the network (leading to a need to converge to a new allocation), versus how important is it for long-running flows to do well?

PCC, also, carries its designers' assumptions about the network. The objective function assumes that loss will be provided as a useful congestion signal, and the constant of 0.05 represents an assumption that stochastic loss will be less than 5% if all is well. When these assumptions are violated, PCC—with its default objective function—performs poorly. On network paths with deep-buffered bottlenecks and no loss, such as some cellular networks, PCC blasts and builds up a standing queue. Across links with too much stochastic loss, PCC can't get anything done. TCP Reno represents some of the same assumptions, embodied in constants like the initial window and rate represent. Like PCC, BBR also has plenty of tuning constants.

There's no problem with tuning constants, but they do represent somebody's expectations about the conditions or workload that the scheme will ultimately face. When you say that "*any* network model is likely to be highly inaccurate," that may well be, but this affects *every* congestion-control scheme (perhaps every engineered system!). TCP Tahoe-like schemes worked well in the 1990s on shallow-buffered bottlenecks with many flows; they performed poorly in the 2000s on

[3] Classical results in Network Utility Maximization [7, 9] are compatible with this conjecture; e.g. the primal algorithm specifies a particular algorithm (not just a utility function) for each sender, and requires knowledge of overall link prices that I suspect cannot be learned from an individual flow's traffic over a DropTail queue, or a proportional-loss link as in the example.

deep-buffered bottlenecks with a single flow ("bufferbloat"). What changed was the quality of the match between Tahoe's implicit assumptions and the networks it faced.

You are certainly right to point out that modeling the entire Internet is infeasible, but there's no evidence you would need to. TCP Tahoe did well for a long time with a very simple model. (Of course, it took a decade from when Tahoe was implemented [6] until the theorists could tell us [7] what Tahoe's model *was*.)

Remy represented an effort to force designers to be explicit about their assumptions by specifying them in a structured format to a protocol-design tool that would then produce an algorithm and all its constants. Depending on how the model is stated—and how much uncertainty or confidence it has about the breadth of networks and workloads the protocol will ultimately encounter—maybe Remy would synthesize the "dumb" scheme from above, or TCP Reno, or a specialized scheme, or who knows.

Just because PCC is less explicit about its assumptions and model doesn't mean that it doesn't have them. The true distinction, in my view, isn't about "reliance on a network model" or the presence of assumptions; the difference is in whether those assumptions were written down, and whether the protocol's behaviors online (including all the constants, initial conditions, etc.) can be justified as being a function of the assumptions and goal of the protocol. If you find yourself tempted to start tuning an algorithm for different kinds of networks (cellular CC, underwater acoustic CC), that's a sign to me that it has, deep down inside, assumptions about the network it will ultimately be running over.

You believe that PCC does not rely on a network model and is therefore inherently robust to variability in network conditions, but how can this be true? How does PCC perform when the network loses 50% of its packets to stochastic loss?

I was reminded of the "AI koan" [5] on this topic:

> *In the days when Sussman was a novice, Minsky once came to him as he sat hacking at the PDP-6.*
> *"What are you doing?", asked Minsky.*
> *"I am training a randomly wired neural net to play Tic-Tac-Toe."*
> *"Why is the net wired randomly?", asked Minsky.*
> *"I do not want it to have any preconceptions of how to play."*
> *Minsky shut his eyes,*
> *"Why do you close your eyes?", Sussman asked his teacher.*
> *"So that the room will be empty."*
> *At that moment, Sussman was enlightened.*

4 HAMILTON REPLIES

Burr, I could not have hoped for a better sparring partner! *But*, I disagree with quite a few things you said.

4.1 White-box vs. black-box approaches are inherently different design philosophies

I do not agree that any design approach reflects a model of the network, and I think that this is actually important.

A useful analogy: identifying cats. Consider the analogy of identifying cats in images. One approach is "thinking really hard" (offline or online), coming up with an explicit model of "what's a cat" (e.g., "cats have pointed ears and sharp teeth"), and then basing decisions on this model. A different approach is applying machine learning (ML). Decisions under the ML approach need not necessarily be "interpretable", in the sense that a model of a cat can be reverse engineered from the ML scheme (say, trained neural network). In fact, the most accurate ML approaches to date aren't.

Why is this important? Because in computer vision, and other central areas in computer science, giving up on the aspiration of basing decisions on accurate explanatory models of complex phenomena has proven crucial for moving beyond the state of the art. Sure, in *some* cases white-box approaches for identifying cats might prove excellent (say, if all cats actually have pointed ears and no other animal does...). However, these are, in general, outperformed by black-box, empirical-evidence-driven approaches.

Now, back to rate control. Online-learning-based rate-control reflects a black box approach in the exact same sense as in the above computer-vision-related example; it does not base its decisions on an (offline/online-generated) model of the network, but instead adapts rates in response to empirically-observed performance derived from past rate selections. The results for PCC in [3], and preliminary experimentation with other black-box approaches to rate-control, provide initial evidence that in this context, too, black-box approaches can outperform white-box approaches. Importantly, unlike many ML techniques, online learning also comes with *provable* guarantees.

One size to fit them all? I am not arguing that any single black-box approach can be *optimal* across all network environments. I expect different choices of utility functions and of online learning algorithms to fare differently in different contexts. Choosing the online-learning method to use in a certain context should be driven by the designer's goal (optimize bandwidth? minimize latency?) and guided by empirical and theoretical explorations. I hope to see online learning rate-control applied to different network environments (wireless, data centers, etc.) in the near future. I conjecture that this could lead to better performance than that achievable by white box approaches even in the environments for which these white-box approaches were designed.

That said, I *do* hypothesize that black-box approaches to rate-control a la PCC can provide high performance in a more robust manner, obviating the need to design *numerous* different specialized protocols, each tailored for a fairly specific network environment, as is essentially the case with TCP.

What are network models good for? While I question reliance on a network model as a *design* philosophy, network modeling is, of course, indispensable for *analyzing* rate-control protocols.

4.2 Optimizing in wonderland?

When discussing rate-control on the parking-lot topology, you assume that all three flows are long-running and run the same protocol, all links drop packets proportionally, other flows do not enter and leave in the interim, links do not fail, no transient microbursts, and more. You also restrict your attention to the steady state (hopefully) *approximated eventually*. You might argue that this is for means of illustration, but I actually think that these assumptions typically underly white-box approaches. There are more things in heaven and earth, Burr, than are dreamt of in your network models!

Optimizing rate-control is plausible in sufficiently stable and predictable networks, and when the notion of optimum is obvious. The heart of our disagreement lies elsewhere: with the exception of very specific environments (highly-optimized private backbone networks?), are the above conditions satisfied? I believe that the answer is, in general, "No".

So, whether or not "Burr's conjecture" is true is beside the point. The point is that global optimization frameworks (such as Network Utility Maximization [7, 9]) fall apart when the actual network conditions violate their underlying assumptions, say, some connection employs a different protocol, or some router discards packets according to a different queueing policy. This motivates the need for other objectives and conceptual frameworks for protocol design.

4.3 What's next?

I believe the properties of PCC you point out as not *inherent* to the online learning paradigm, but merely properties of its *specific realization* in [3]. In my view, identifying the "right" algorithms and utility functions to build into online-learning rate control in different contexts is a conceptually fascinating and practically promising research agenda.

5 BURR'S CONCLUSION

Hamilton, this has been a fascinating plane ride. I apologize for all those times I hit the call button to try to get you moved to a different seat. I salute your willingness to shake things up in this 30+-year-old research area—one that governs when we speak and when we stay silent on the Internet, perhaps the most important communications medium since cuneiform. Let me try to summarize our disagreement:

The "white-box vs. black-box design philosophy" issue means a lot to you, and little to me. I think you view BBR and Remy, and even Reno, as arrogant: they presume to have an opinion about how the network will behave. Reno cuts the window if it encounters loss because it *assumes* that loss indicates overload. BBR *assumes* that it can infer the bottleneck bandwidth and rate. Remy wants the designer to specify a whole litany of assumptions upfront. Whereas it seems purer to simply have an objective function (even if it needs some carefully chosen constants) that the flow tries to optimize by wiggling its rate up and down.

To me, this philosophizing is beside the point. I don't think it's relevant what the designer was thinking; I care about what the code does at runtime. Every scheme we've named has situations where it does well and situations where it does poorly—these represent, ultimately, the scheme's assumptions about what the network and workload will look like. I don't think there's evidence that schemes that online-optimize a locally perceptible objective function perform better in a broader range of scenarios than other protocols.

You see more-principled learning methods as the future, and I'm not so sure. I look forward to "no regret" congestion control, but ultimately the test will be how the code performs at runtime. My numerical experiment on the parking-lot network showed problems with the distributed-online-optimization approach even where no flow had cause for regret. So I'm skeptical this is what's needed.

Is there still a need for a theory of congestion control? One thing we didn't discuss is the genesis of BBR. To develop this scheme and tune all of its constants and behaviors, Google says they tried many iterations of BBR on a fraction of the company's traffic, slowly ramping up the deployment as they searched for pathological behaviors or situations where BBR underperformed their status-quo scheme. This might be as close as you can get to "model-free" design, since they are designing the scheme live against Google's actual traffic—a substantial fraction of the whole Internet—and optimizing metrics of interest that, in the end, correlate with money made. It's not so easy for a skeptical outsider to go toe-to-toe with them because they have all that data and we don't. In a world where much of the Internet's traffic comes from a small number of big players who can confidently go their own way, is there still a role for theory, or for arguments like ours?

6 HAMILTON'S CONCLUSION

We're at the gate—I have to go. But rest assured, when our children tell our story, they'll tell the story of this flight.

ACKNOWLEDGMENTS

Hamilton thanks Brighten Godfrey for extensive brainstorming about the above-discussed topics, the rest of the PCC team at Hebrew U. and UIUC for valuable comments, and also Nathaniel Pendleton. Burr thanks Mohammad Alizadeh, Hari Balakrishnan, Anirudh Sivaraman, Nick McKeown, Philip Levis, and William P. Van Ness. Both authors thank the HotNets reviewers for their helpful comments. This work was funded in part by DARPA grant HR0011-15-2-0047.

REFERENCES

[1] N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh, and V. Jacobson. BBR: Congestion-based congestion control. *Queue*, 14(5):50:20–50:53, Oct. 2016.

[2] N. Cesa-Bianchi and G. Lugosi. *Prediction, Learning, and Games*. Cambridge University Press, New York, NY, USA, 2006.

[3] M. Dong, Q. Li, D. Zarchy, P. B. Godfrey, and M. Schapira. PCC: re-architecting congestion control for consistent high performance. In *12th USENIX Symposium on Networked Systems Design and Implementation, NSDI '15, Oakland, CA, USA, May 4-6, 2015*, pages 395–408, 2015.

[4] D. Fudenberg and D. Levine. *The Theory of Learning in Games*. The MIT Press, 1998.

[5] W. D. Hillis. AI Koans (ca. 1980), *available at* https://groups.google.com/d/topic/net.jokes/KrSD2dbCzQ8.

[6] V. Jacobson. Congestion Avoidance and Control. In *SIGCOMM*, 1988.

[7] F. P. Kelly, A. Maulloo, and D. Tan. Rate Control in Communication Networks: Shadow Prices, Proportional Fairness and Stability. *Journal of the Operational Research Society*, 49:237–252, 1998.

[8] A. Sivaraman, K. Winstein, P. Thaker, and H. Balakrishnan. An experimental study of the learnability of congestion control. In *SIGCOMM*, Chicago, Illinois, August 2014.

[9] R. Srikant. *The Mathematics of Internet Congestion Control*. Birkhauser, 2004.

[10] K. Winstein and H. Balakrishnan. TCP ex Machina: Computer-Generated Congestion Control. In *SIGCOMM*, Hong Kong, China, August 2013.