

Protocol Design Contests

Anirudh Sivaraman, Keith Winstein, Pauline Varley, João Batalha, Ameesh Goyal, Somak Das, Joshua Ma, and Hari Balakrishnan

Massachusetts Institute of Technology, Cambridge, Mass.

{anirudh, keithw, paulinev, johnnyb, ameesh, das, joshma, hari}@mit.edu

ABSTRACT

In fields like data mining and natural language processing, design contests have been successfully used to advance the state of the art. Such contests offer an opportunity to bring the excitement and challenges of protocol design—one of the core intellectual elements of research and practice in networked systems—to a broader group of potential contributors, whose ideas may prove important. Moreover, it may lead to an increase in the number of students, especially undergraduates or those learning via online courses, interested in pursuing a career in the field.

We describe the creation of the infrastructure and our experience with a protocol design contest conducted in MIT's graduate Computer Networks class. This contest involved the design and evaluation of a congestion-control protocol for paths traversing cellular wireless networks. One key to the success of a design contest is an unambiguous, measurable objective to compare protocols. In practice, protocol design is the art of trading off conflicting goals with each other, but in this contest, we specified that the goal was to maximize $\log(\text{throughput}/\text{delay})$. This goal is a good match for applications such as video streaming or videoconferencing that care about high throughput and low interactive delays.

Some students produced protocols whose performance was better than published protocols tackling similar goals. Furthermore, the convex hull of the set of all student protocols traced out a trade-off curve in the throughput-delay space, providing useful insights into the entire space of possible protocols. We found that student protocols diverged in performance between the training and testing traces, indicating that some students had overtrained ("overfitted") their protocols to the training trace. Our conclusion is that, if designed properly, such contests could benefit networking research by making new proposals more easily reproducible and amenable to such "gamification," improve networked systems, and provide an avenue for outreach.

CATEGORIES AND SUBJECT DESCRIPTORS

C.2.2 [Computer-Communication Networks]: Network Protocols

KEYWORDS

Design Contest, Protocol, Machine Learning, Congestion Control, Gamification

1. INTRODUCTION

Research in networked systems is carried out largely by professional researchers at academic institutions, industrial labs, and companies. Opening up research problems to a wider audience can only benefit the field by increasing the number of useful attempts at solving a problem. *Design contests* are one way to achieve this

goal by posing research problems in a manner that anyone with the right background can attempt them and compare their own solutions against other participants and the state-of-the-art.

Design contests benefit research in several ways (§5): they promote reproducible research, publicize research to a wider audience, and may even provide lasting advances. Other fields such as natural language processing and information retrieval have benefited from such contests. Similar efforts have not been adopted in our field, perhaps because of the perception that the design and implementation of networked systems is inherently harder to cast into small, well-defined problems with quantitative goals.

There is no doubt that networked systems are complex and often involve engineering trade-offs that are hard to justify on purely quantitative grounds. However, they also often involve components whose performance is measurable and matters. One such example is congestion control.

As part of MIT's graduate course in computer networks, 6.829 (<http://mit.edu/6.829>), we created and ran a design contest to develop the "best" congestion-control protocol over paths that traverse cellular wireless networks. Cellular networks are a challenging case for the many TCP congestion-control variants because link rates vary significantly with time and because the amount of buffering inside the network is often much higher than the prevailing (time-varying) bandwidth-delay product.

We asked students to develop a protocol to maximize the ratio of throughput to 95% end-to-end delay (defined precisely in § 2.4) measured over a long-running connection. The evaluation of each protocol was done over a packet trace we gathered independently. We provided a training trace to help the students design protocols, and then after the design was "frozen" and finalized, evaluated each protocol on a fresh test trace.

From this contest, we learned that:

1. Almost all students submitted protocols that performed competitively on the supplied trace, but ...
2. ... for some of the entries, there was a significant decrease in performance when run on a newly-collected trace used for the final evaluation (Figure 8), suggesting a degree of over-training.
3. The set of student protocols traced out an achievability region in the throughput-delay space (Figure 7). An analytic determination of the same achievability region is a much harder task, and our result shows that it might be feasible to estimate such achievability regions through such contests.
4. On the specific metric of $\log(\text{throughput}) - \log(95\% \text{ end-to-end delay})$, two student protocols performed slightly better than Sprout [12], a recently published

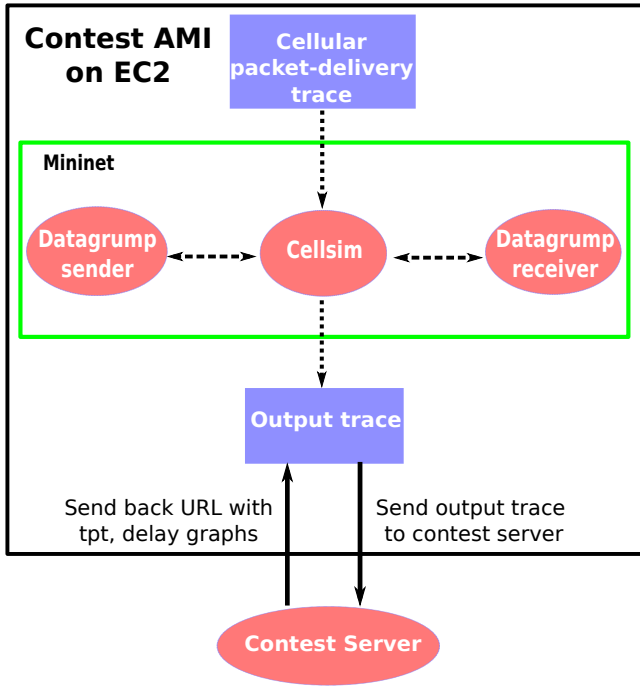


Figure 1: Block Diagram of Contest Infrastructure

protocol (§3.3). This result indicates that “crowdsourcing” well-defined components of research problems to a broader audience can lead to improved solutions.

2. CONTEST DESIGN

Prior to the contest, we taught students the main principles of congestion control, including the ideas in some TCP variants (Reno [9], Vegas [6], Compound [11], and Cubic [8]), non-AIMD protocols (binomial [5] and equation-based control [7]), notions of proportional and max-min fairness, and active queue management methods. We also exposed the students to Mininet [10] in a problem set exercise before the start of the contest.

The two-week contest included some warm-up exercises such as experimenting with fixed-window, AIMD, and delay-based congestion control to get students started with the process of designing a congestion-control protocol. For the contest, we picked two baseline protocols: Sprout, from one of our recently published papers [12], and the transport protocol used by Skype, a popular videoconferencing program.

Figure 1 shows a block diagram of the contest infrastructure. We provided students with an Amazon Machine Image (AMI), a disk image containing a basic implementation of a congestion-control protocol (§2.1), an evaluation harness (§2.2), and a packet-delivery trace from a cellular network (§2.3). Students modify this implementation as they see fit, and evaluate their protocols using the harness. Once the evaluation has run the duration of the trace, the output trace is automatically uploaded and scored by the contest server (§2.4). The contest server returns a URL displaying the throughput and end-to-end delay achieved by the protocol over time, and the position of the protocol relative to other students on a leaderboard (§2.5). After the contest, we use an analogous procedure to evaluate students on the testing trace (§2.6).

2.1 Boilerplate Protocol

To ensure that students focused only on the congestion-control protocol itself, and not on low-level details such as UNIX sockets,

we provided students with a customizable protocol implementation skeleton called *datagrump* (available at <https://github.com/keithw/datagrump>). The skeleton defines a Controller class that can be modified by the student to specify the entire congestion-control algorithm:

1. `Controller::packet_was_sent` specifies code that is executed whenever a packet is transmitted.
2. `Controller::ack_received` specifies code that is executed when an acknowledgment is received.

2.2 Evaluation Procedure

The *packet-delivery process* is the random process that determines the instants at which packets are delivered over a link. A packet-delivery trace is a particular instantiation of this random process. For a fixed-capacity link, any trace is an isochronous sequence separated by the transmission time. For a variable-capacity link (like a cellular network), however, the inter-delivery duration can vary arbitrarily depending on the prevailing link quality. To model this, we first capture a packet delivery trace from a real cellular network using a tool called *Saturatr* [12].

Saturatr runs on a laptop, tethered to two phones each with a cellular data connection: the loaded phone and the feedback phone. We operate a server in the Internet at a location with a low-latency path to the cellular provider. *Saturatr* attempts to fully utilize the cellular link on the loaded phone, by transmitting packets over the loaded phone between the laptop and the server in both directions (from the loaded phone to the server and back). *Saturatr* maintains a few seconds of queuing delay in each direction without allowing the queuing delay to grow indefinitely. The larger the queuing delay, the more unlikely that the link will starve for packets if the link quality suddenly improves. This, in turn, makes it more likely that *Saturatr* is fully utilizing available capacity. At the same time, an extremely large delay can result in the network operator blocking network access. The feedback phone is used exclusively for feedback in both directions between the server and the laptop. Because it is used only for acknowledgements, it has low queuing delays in either direction that expedite feedback.

The packet delivery trace from *Saturatr* is used as input to *Cellsim* [12], a network emulator that relays packets between two network interfaces according to the packet delivery trace recorded by *Saturatr*. Whenever *Cellsim* receives a packet from an application, it delays it by a user-specified amount (the minimum one-way delay), and then enqueues it into a FIFO. Based on the packet delivery trace, *Cellsim* then dequeues an MTU worth of bytes at every instant that a packet was delivered in the original trace. This accounting is done on a byte basis so that multiple smaller packets can be dequeued in lieu of one large packet on every packet delivery event. For all evaluations in this paper, we set the minimum one-way delay to 20 ms.

Our complete test harness (Figure 1) consists of three components: *Cellsim*, a *datagrump* sender, and a *datagrump* receiver. Each component runs within a separate Linux Container provided by Mininet [10], which is also responsible for routing packets from the sender, through *Cellsim*, to the receiver. To evaluate student submissions portably and repeatably, we created an Amazon AMI containing the test harness (the sender, receiver, and *Cellsim* all running within Mininet) to allow each student team to evaluate their protocol independently during the training period.

2.3 Training Trace

We released a training packet-delivery trace spanning 140 seconds to aid students in developing the congestion-control protocol.

The training trace is a snippet of a Verizon LTE trace used in our evaluations in [12].

Halfway through the training period, we also released a *validation* packet-delivery trace (a snippet of an AT&T 4G link), mimicking the actual evaluation procedure since the testing trace is different from the training trace. Using a validation trace allows students to develop protocols that aren't attuned to noise inherent in the trace files.

2.4 Performance Metrics

We measure the long-term throughput of the protocol over the entire trace, and the “95% end-to-end delay” (defined in the Sprout paper [12]), which we repeat here. We first record the arrivals of all packets at the receiver. Then, at every instant in time¹, we find the packet that was sent most recently among all packets that have arrived so far at the receiver. The time elapsed since this packet was sent is a lower bound on the playback buffer that a hypothetical video/audio conferencing system must impose to guarantee glitch-free playback at that instant. We compute this elapsed time at every instant in the received trace and call the 95th percentile of this quantity the “95% end-to-end delay”.

We use the 95% end-to-end delay instead of more conventional metrics such as tail per-packet delay because a clever sender could choose to hold off sending packets when the network's delays are already high. While holding off packets certainly mitigates congestion, it does not benefit the application because the delay simply builds up at the sender, instead of building up within the network. Measuring per-packet delay in such cases would be misleading. Using the end-to-end delay guards against this problem because it rises by 1 second for every second that the receiver doesn't receive any packets. Figure 5 illustrates this effect in a rather extreme case. If the sender sends exactly two packets: one at the very beginning and one at the very end of a trace, the end-to-end delay increases linearly², even though the per-packet delay of both these packets is 20 ms.

For every run of a submitted protocol, we upload an *output trace* describing the actual packet deliveries from the student's protocol to the contest server, where we plot a time series depicting the throughput in Mbits/sec of the student's protocol in non-overlapping 500 ms windows and compare that with the link capacity in the same window. Similarly, we plot a time series of the end-to-end delay sampled every millisecond. Four examples of the visualization are given in Figures 2, 3, 4, and 5, one each for every combination of low and high throughput, and low and high end-to-end delay.

To compare protocols, we combine these two metrics into a single score:

$$\log(\text{throughput}) - \log(95\% \text{ end-to-end delay}) \quad (1)$$

This objective captures a preference for both high throughput (for good quality) and low end-to-end delay (to maintain interactivity). Unless otherwise mentioned, all further mentions of delay refer to 95% end-to-end delay defined above.

A notable absentee from the evaluation procedure is cross-traffic. While most congestion-control protocols operate on a link with a FIFO queue shared among several flows, cellular networks typically provide per-user isolation preventing one user's aggressive behavior from adversely affecting another [12]. On networks where this assumption does not hold, the evaluation procedure would have to be revised to either include concurrent cross traffic or run several instances of the submitted protocol concurrently on the same link.

¹In practice, we sample this quantity at a constant interval of 1 ms.

²Note that the plot is log scale on the delay axis.

2.5 The Leaderboard

Once each team had submitted at least one protocol, we displayed all teams on a leaderboard (Figure 6), ordered by their $\log(\text{throughput}) - \log(95\% \text{ end-to-end delay})$ scores. We also display the throughput and 95% end-to-end delay achievable by an omniscient protocol: a protocol that utilizes every available packet delivery, and incurs a fixed per-packet delay of 20 ms on every packet. Even so, as Figure 6 shows, the omniscient protocol might have a 95 % end-to-end delay much larger than the minimum one-way delay of 20 ms because of outages in the trace.

Soon after releasing the leaderboard, the teams started working harder raising their scores to get to the top of the leaderboard. This happened despite our telling students that what really matters is the testing phase, and to guard against overtraining; several teams ended up training their protocol (§3) to noise, leading to worse performance on the test data.³

Username	Tpt (Mbps)	95% delay (ms)	Score
(omniscient)	3.41	69.00	-3.01
johnnybameesh	2.53	161.00	-4.15
sprout	2.03	139.00	-4.22
dasjoshma	2.52	174.00	-4.23
XXXXXX	2.32	184.00	-4.37
XXXXXX	1.98	162.00	-4.40
XXXXXX	2.21	199.00	-4.50
XXXXXX	2.27	222.00	-4.58
XXXXXX	2.46	244.00	-4.60
XXXXXX	1.86	199.00	-4.67
XXXXXX	2.02	221.00	-4.69
XXXXXX	1.70	221.00	-4.87
XXXXXX	1.65	226.00	-4.92
XXXXXX	2.18	360.00	-5.11
XXXXXX	2.84	472.00	-5.11
XXXXXX	2.72	471.00	-5.15
XXXXXX(this trace)	2.66	462.00	-5.16
XXXXXX	0.97	173.00	-5.19
XXXXXX	2.72	595.00	-5.39
XXXXXX	3.29	1193.00	-5.89
XXXXXX	0.24	150.00	-6.43
XXXXXX	3.07	7957.00	-7.86

Figure 6: Screenshot of leaderboard on testing trace showing protocol's position relative to others

2.6 Testing Trace

At the end of the training period, we collected a new set of cellular network traces using Saturatr. These traces were used to score the final submission from each team to determine the contest's winners. The evaluation was identical for the training and testing traces except for the choice of the trace itself. To determine the final student ranking on the testing trace, we ran each protocol three times on the testing trace and took the best of the three runs.

2.7 Prizes

³The concept of overfitting captures a similar notion for classification tasks. Our solutions, however, are judged by network-centric metrics such as throughput and end-to-end delay, and so we avoid using the term overfitting.

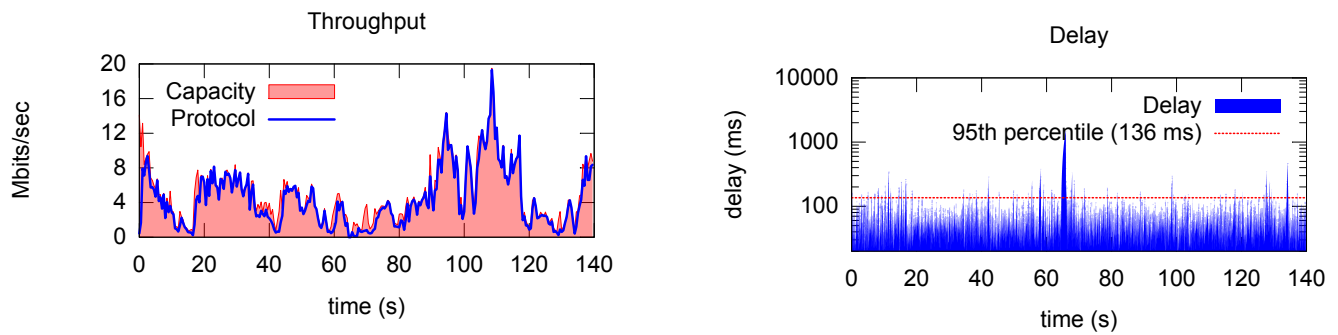


Figure 2: A protocol with high throughput and low end-to-end delay

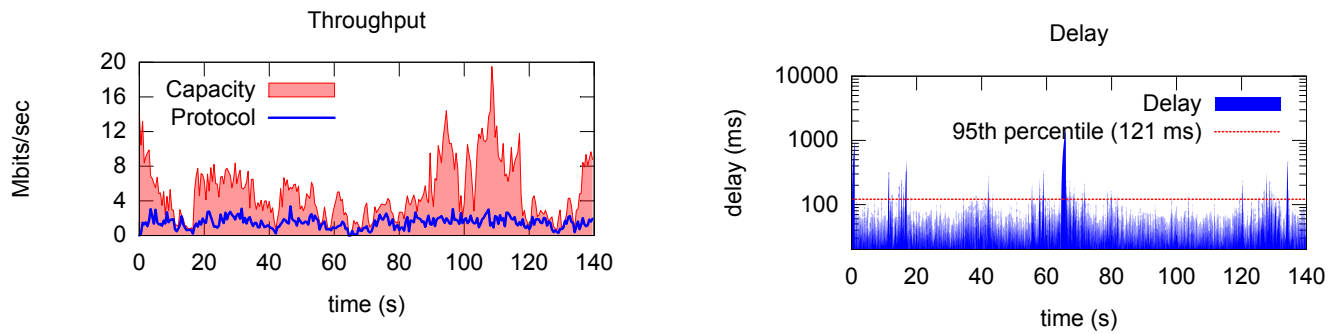


Figure 3: A protocol with low throughput and low end-to-end delay

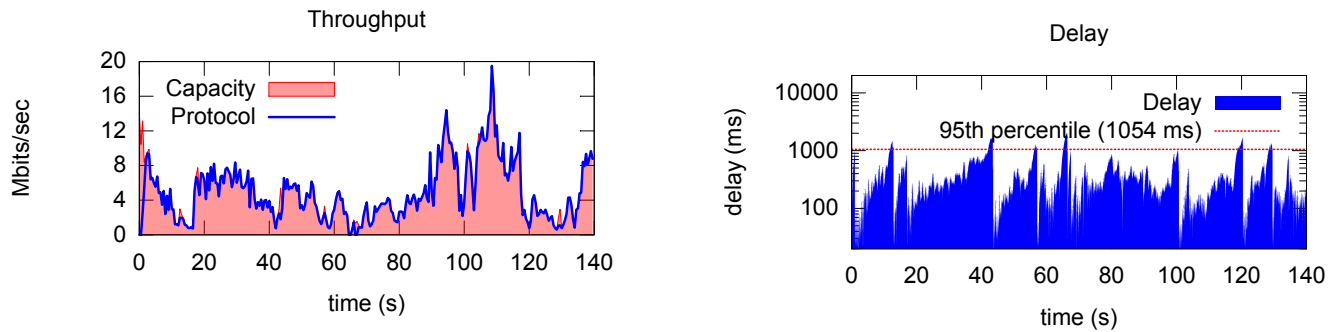


Figure 4: A protocol with higher throughput, but much higher end-to-end delay

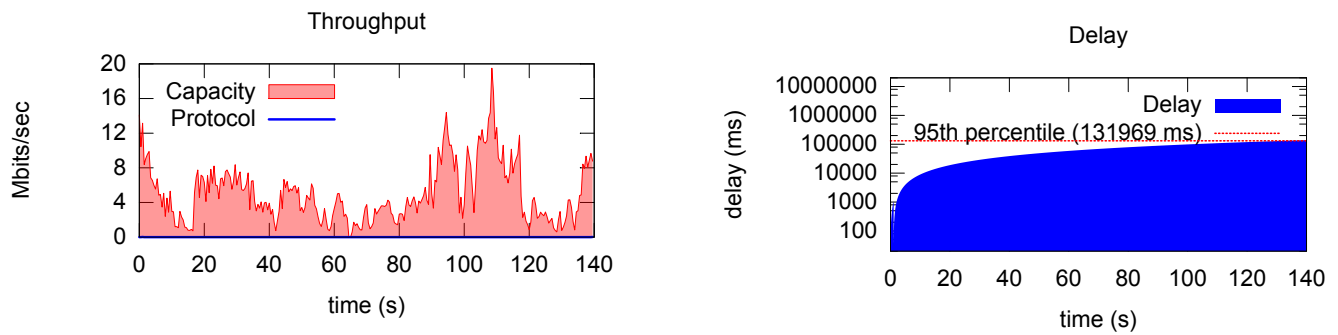


Figure 5: A protocol with low throughput and linearly increasing end-to-end delay

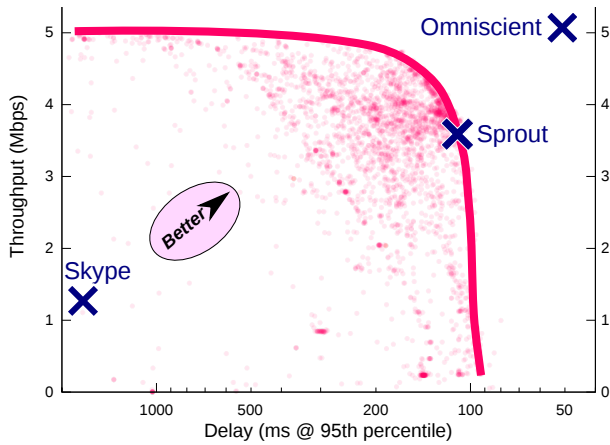


Figure 7: The 3000 student submissions

What’s a contest without prizes? When we announced the contest to the class, we offered modest cash prizes to the 4th, 3rd, and 2nd place finishers, and also announced the grand prize for the first place: co-authorship on a future paper if they developed a scheme that beat Sprout. That announcement was met with a moment’s silence, followed by a student’s question asking whether the grand prize could instead be in the form of cash as well. (In the end, we did give gift cards to the top four winning teams.)

3. RESULTS

3.1 Achievability Region

Figure 7 shows the performance achieved by the roughly 3000 student submissions on the training trace over the two-week contest period. The submissions trace out an empirical throughput-delay “achievability region” shown by the convex hull.

Sprout was on the frontier of this achievability region: other protocols achieved either lower delay or higher throughput than Sprout, but not both.

Without the contest to harness the ingenuity of 40 student protocol designers, it would have been much harder to form this conclusion because cellular packet-delivery traces do not lend themselves to an analytical treatment that could estimate the achievability region in closed form. Of course, we make no mathematical guarantee that the student-drawn achievability region in a two-week contest is the true achievability region for this problem, but the shape suggests that it might be. In any case, these are an inner bound on the true achievability region.

3.2 Training vs. Testing Results

Despite providing teams with a validation trace to avoid over-training, some teams created protocols that were highly tuned to the training trace. The result was degraded performance on the testing trace. We show this effect in Figure 8. We plot the utilization instead of throughput for both the training and testing traces because the capacity of the two traces are different. Most student protocols are close to each other and perform well on the training set. However, when evaluated on the testing set, these protocols disperse, indicating that several protocols had trained themselves to noise in the training trace.

3.3 Winning Entries

On the $\log(\text{throughput}) - \log(95\% \text{ end-to-end delay})$ metric, two student submissions did slightly better or about as well as Sprout. These two protocols are de-anonymized in Figure 8.

“dasjoshma” represents the protocol designed by Somak Das and Joshua Ma and “johnnybameesh” represents the protocol designed by João Batalha and Ameesh Goyal.

To understand this improvement, we briefly review Sprout’s protocol here. Sprout models the evolution of link rate as a random walk and maintains a probability distribution of the link rates at every instant. Sprout integrates the link-rate random variable forward in time for the next 100 ms to characterize the random variable representing the cumulative number of packet deliveries. It then computes the 95th percentile of this random variable, giving Sprout a cautious forecast for the number of packets that will be delivered in the next 100 ms, which Sprout uses as its current window. Sprout’s objective is not maximizing $\log(\text{throughput}) - \log(95\% \text{ end-to-end delay})$; instead, Sprout attempts to maximize throughput while ensuring that the probability of the per-packet delay exceeding 100 ms is under 95%. Sprout places greater emphasis on delay than throughput, while the objective in our contest accords equal importance to both quantities. We note that Sprout achieves close to the lowest delay on both the training and testing traces in Figure 8.

The two student submissions that improved on Sprout used a similar approach:

1. Estimate the received rate using an exponentially-weighted moving average (EWMA).
2. At the sender, multiply the received rate by a tunable target queuing delay.
3. Use this as the congestion window for the transport protocol.

The code for the two winning entries is available at <https://github.com/joshma/datagrump> (dasjoshma) and <https://github.com/ameeshg/wither> (johnnybameesh).

4. RELATED WORK

Data contests aiming to make sense of massive amounts of data are gaining in popularity. The Netflix grand challenge is a prominent example: Netflix awarded a \$1 million prize for an algorithm that improved its recommendation system by a specified amount on a well-defined scoring metric. Contests are also common in the area of natural language processing such as the Knowledge Base Population [4] competition, where participants populate existing knowledge bases such as Wikipedia from unstructured text.

Kaggle [2] is an online service that allows data producers to tap into a community of data scientists through contests. These contests span areas ranging from predicting loan defaults and insurance claims to flight simulation. Our contest also uses real-world data in the form of a packet-delivery trace, but our goal was not to develop a predictive algorithm to predict link capacity or minimum one-way delay on the link. Instead, our goal was to develop a congestion-control protocol measured using standard networking metrics such as throughput and delay.

Various DARPA grand challenges are other instances of competitions, which have sometimes led to the design of better systems (autonomous cars) or have demonstrated surprising capabilities (the balloon challenge). The on-going spectrum challenge [1] takes this approach toward physical layer radio communication.

Stanford University’s Reproducible Network Research class [3] takes published research and replicates their results over a semester-long class. Protocol design contests can take reproducible research one step further by not only reproducing existing research, but also improving upon them.

5. OUTLOOK

Design contests such as the one we conducted in MIT’s graduate networking class have several benefits, which we outline below:

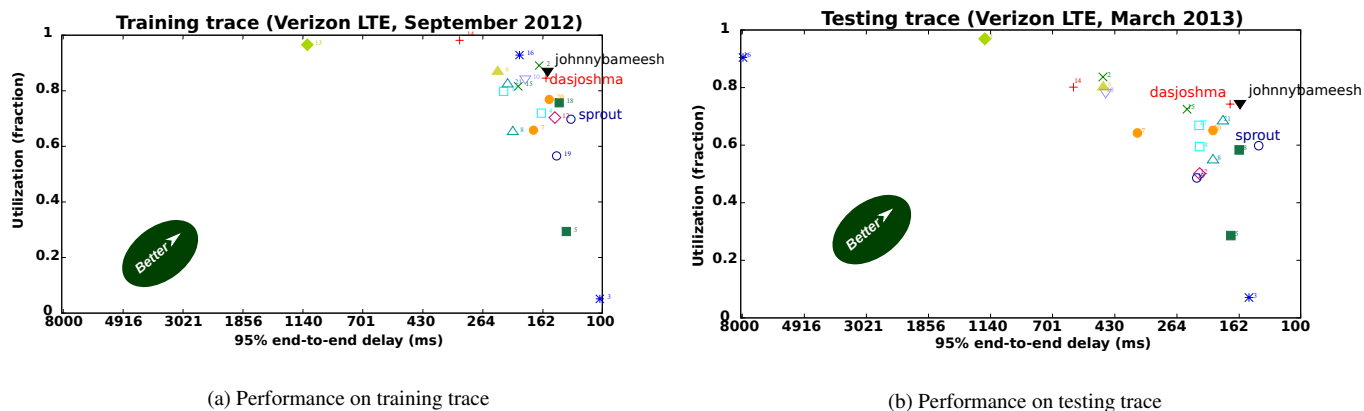


Figure 8: Student protocols cluster on the training trace and disperse on the testing trace

1. **Crowdsourced Protocol Design.** Tapping into a larger pool of human resources allows us to design better protocols, assuming an objective evaluation metric is known a priori. Crowdsourcing has largely been used for repetitive and tedious tasks by previous systems such as Mechanical Turk. Design contests allow us to use the wisdom of the crowd for considerably more involved problems.

2. **Reproducible Research.** Turning a research problem into a well-specified contest forces the researcher to clearly articulate the testing conditions and ensure that her protocol works reproducibly under those conditions. This, in turn, makes the protocol accessible to a wider audience of other researchers.

3. **Explicitness in problem specification.** “Gamifying” a research problem entails setting up a well-defined problem where the end objective is clearly specified. Several protocols today (for instance, TCP Cubic [8]) try to do something reasonable under all conditions without specifying their end goal. Specifying an objective explicitly has several benefits:

- (a) It affords a more directed search for the solution.
- (b) It forces the designer to think hard about whether the objective is one that an application truly cares about. The distinction between per-packet delay and end-to-end delay, discussed earlier, is a case in point.

4. **Achievable Region Estimation.** The achievable region (in our case, all achievable (throughput, delay) tuples) is mathematically intractable in several cases. Turning such problems into contests allows us to trace out a crowdsourced achievability region, assuming a large number of submissions.

Such contests have benefited several other fields and we think they confer similar benefits on Computer Networking as a whole. They also provide a means to engage students directly in the activity of research by simplifying a problem down to its essence such that it can be approached by a student with little domain expertise. Feedback from students taking the class was overwhelmingly positive, with several students singling out the contest for praise.

Code for all components of our evaluation infrastructure is available at <http://web.mit.edu/anirudh/www/contest.html>. In describing our experiences running a congestion-control protocol design contest, we hope to prod others into running similar contests centered around other classical, well-defined problems within networking such as routing, traffic engineering, and scheduling.

6. ACKNOWLEDGMENTS

We thank the 6,829 students for making this contest possible. Jonathan Perry, as the teaching assistant for 6.829, gave us valuable feedback on the design of the contest. We thank the anonymous CCR reviewers and Amy Ousterhout for feedback that significantly improved the quality of the paper.

7. REFERENCES

- [1] DARPA Spectrum Challenge. <http://www.darpa.mil/spectrumchallenge/>.
- [2] Kaggle: Go from Big Data to Big Analytics. <https://www.kaggle.com/>.
- [3] Reproducing Network Research — network systems experiments made accessible, runnable, and reproducible. <http://reproducingnetworkresearch.wordpress.com/>.
- [4] Text Analysis Conference (TAC) KBP 2013 Tracks. <http://www.nist.gov/tac/2013/KBP/>.
- [5] D. Bansal and H. Balakrishnan. Binomial Congestion Control Algorithms. In *INFOCOM*, 2001.
- [6] L. S. Brakmo, S. W. O’Malley, and L. L. Peterson. TCP Vegas: New Techniques for Congestion Detection and Avoidance. In *SIGCOMM*, 1994.
- [7] S. Floyd, M. Handley, J. Padhye, and J. Widmer. Equation-Based Congestion Control for Unicast Applications. In *SIGCOMM*, 2000.
- [8] S. Ha, I. Rhee, and L. Xu. CUBIC: A New TCP-Friendly High-Speed TCP Variant. *ACM SIGOPS Operating System Review*, 42(5):64–74, July 2008.
- [9] V. Jacobson. Berkeley TCP Evolution from 4.3-tahoe to 4.3-reno. In *Proceedings of the Eighteenth Internet Engineering Task Force*, 1990.
- [10] B. Lantz, B. Heller, and N. McKeown. A network in a laptop: Rapid prototyping for software-defined networks. In *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks, Hotnets-IX*, pages 19:1–19:6, New York, NY, USA, 2010. ACM.
- [11] K. Tan, J. Song, Q. Zhang, and M. Sridharan. A Compound TCP Approach for High-speed and Long Distance Networks. In *INFOCOM*, 2006.
- [12] K. Winstein, A. Sivaraman, and H. Balakrishnan. Stochastic Forecasts Achieve High Throughput and Low Delay over Cellular Networks. In *USENIX Symposium on Networked Systems Design and Implementation*, Lombard, IL, April 2013.