TCP EX MACHINA: COMPUTER-GENERATED CONGESTION CONTROL
KEITH WINSTEIN AND HARI BALAKRISHNAN

Presented by: Angela Jiang
Network congestion

- **Cause:** Sources trying to send data faster than the network can process
- **Result:** QoS deterioration of network
  - Queuing delays
  - Packet losses
  - Congestion collapse
Congestion control

- Prevent congestion collapse
- Allocates network resources
  - Link bandwidth
  - Queue space
- End-to-end or network assisted
End-to-end congestion control

- No support from network layer
- Hosts must infer presence of congestion
  - Packet loss
  - Queueing delay
  - ECN marks
- Individually control transmission rate
Challenge: evolving networks

- Wireless
- Short connections
- Bursty traffic
- Datacenters
Evolution of congestion control
Challenge: rational choice of scheme

- Different goals?
- Different assumptions about network?
- One scheme just plain better?
Current congestion control

- Inflexible; doesn’t allow network evolution
- Unclear what an algorithm is optimized for
Remy

● *Program that generates end-to-end congestion control schemes offline*

● **Given**
  ○ Network representation
  ○ Objective of app (e.g. high throughput)

● **Generates**
  ○ RemyCC; a congestion control algorithm
Objective function

- Fairness vs. efficiency
  \[ U_\alpha(x) = \frac{x^{1-\alpha}}{1-\alpha} \]

- Delay vs. throughput
  \[ U_\alpha(x) - \delta \cdot U_\beta(y) \]

- Objectives used
  \[ U = \log(\text{throughput}) - \delta \cdot \log(\text{delay}) \]
  \[ U = -\frac{1}{\text{throughput}} \]
Prior assumptions of network

- Model of network uncertainty
  - Link speed distribution
  - Delay distribution
  - Degree of multiplexing

- Traffic model
  - Off-to-on model
  - Web browsing, MapReduce, VoIP
RemyCC maps state to an action

\[
\text{Rule}(r\_\text{ewma}, s\_\text{ewma}, rtt\_\text{ratio}) \rightarrow \langle m, b, \tau \rangle
\]

- \textit{m} Multiple to congestion window
- \textit{b} Increment to congestion window
- \textit{\tau} Minimum interval between two outgoing packets
Find the best value

$\langle ?, ?, ?, ? \rangle$

Keith Winstein
MIT Computer Science and Artificial Intelligence Laboratory

TCP ex Machina: Computer-Generated Congestion Control
Best single action

\[ <0.90, 4, 3.3> \]
Subdivide most used rule

\begin{array}{c|c|c|c|}
\hline
\text{s_{ewma}} & <0.90,4,3.3> & <0.90,4,3.3> & <0.90,4,3.3> > 0.90,4,3.3> > 0.90,4,3.3> > 0.90,4,3.3> \hline
\text{r_{ewma}} & <0.90,4,3.3> & <0.90,4,3.3> & <0.90,4,3.3> > 0.90,4,3.3> > 0.90,4,3.3> > 0.90,4,3.3> \hline
\end{array}
Optimize each new action

- $r_{ewma}$
  - $<0.90,5,2.8>$
  - $<0.60,19,76.2>$

- $s_{ewma}$
  - $<0.70,6,53.5>$
  - $<0.80,5,4.1>$

Keith Winstein
MIT Computer Science and Artificial Intelligence Laboratory

TCP ex Machina: Computer-Generated Congestion Control
Split most used rule

Keith Winstein
MIT Computer Science and Artificial Intelligence Laboratory

TCP ex Machina: Computer-Generated Congestion Control
Final RemyCC rule table
Evaluation

● 3 RemyCCs with $\delta = 0.1, 1, 10$
● Generating one RemyCC
  ○ Takes a few hours
  ○ $\$5-$10 on EC2
● Compared against end-to-end and network assisted congestion control schemes
Single bottleneck ("dumbbell")

As desired, this test demonstrates that Remy was successful in the one bottleneck paths experienced by Internet flows. The richness of real-world network paths, the dumbbell is a valuable topology to investigate because in practice there are many single-bottleneck "dumbbell" topology. Although it does not model the delay distribution that explains the points. To summarize the whole scheme, we present the results for the different protocols as a throughput and delay values. They provide more information than simply reporting mean and 1-second run from a simulation as one point, and then compute the age per-packet delay in excess of minimum RTT). Lower, better, decreasing frontier, which here is defined entirely by the RemyCCs. We present the results for the different protocols as "top right" are the best on such plots. We take each individual 10000 sender, each alternating between flows whose length is drawn from a purely end-to-end scheme can outperform well-designed algorithms that involve active router participation. This demonstrates that distributed congestion-control algorithms that explicitly maximize well-chosen objective functions can achieve gains over existing schemes. As we will see later, however, this substantially outperformed all the human-designed ones.

Recall that this particular dumbbell link had most of its parameters:~

- Cubic/sfqCoDel
- Compound
- NewReno
- XCP
- Better

Median RTT of TCP Cubic. CoDel drops a packet at the front of the queue if all packets in the past 100 ms experienced a time, such a persistent queue is less common even though the mean queueing delay is a lot more than 5 ms. DropTail experiences more losses, so has lower delays (the maximum queue size is 5 packets); if the throughput is increased, RemyCC are contradicted at runtime. This result is encouraging, because it suggests that even distributed congestion-control schemes trace out a path from most delay-conscious (Vegas) to most throughput-conscious (Cubic), with sfqCoDel instead of DropTail. In other experiments with longer transfers, Cubic did experience bandwidth-delay product, but also lower throughput than CoDel. When the transfer lengths are only 100 kilobytes; with a 500 ms "off" time, such a persistent queue is less common even though the mean queue size is 5 packets; if the throughput is increased, RemyCC are contradicted at runtime. This demonstrates the effect of the parameter in weighting the cost of delay. In producing a family of congestion-control algorithms for this type of network.
Varying throughput ("cellular")

Figure 6: Sequence plot of a RemyCC flow in contention with varying cross traffic. The flow responds quickly to the departure of a competing flow by doubling its sending rate.

In Figures 4 and 5, the RemyCCs do not simply have better median performance — they are also more fair to individual flows, in that the performance of an individual sender (indicated by the size of the ellipses) is more consistent in both throughput and delay.

To explain this result, we investigated how multiple RemyCC flows share the network. We found that when a new flow starts, the system converges to an equitable allocation quickly, generally after little more than one RTT. Figure 6 shows the sequence of transmissions of a new RemyCC flow that begins while sharing the link. Midway through the flow, the competing traffic departs, allowing the flow to start consuming the whole bottleneck rate.

5.3 Cellular Wireless Links

Cellular wireless links are tricky for congestion-control algorithms because their link rates vary with time.

By running a program that attempts to keep a cellular link backlogged but without causing buffer overflows, we measured the variation in download speed on Verizon's and AT&T's LTE service while mobile. We then ran simulations over these pre-recorded traces, with the assumption that packets are enqueued by the network until they can be dequeued and delivered at the same instants seen in the trace.

As discussed above, we did not design the RemyCCs to accommodate such a wide variety of throughputs. Running the algorithm over this link illustrated some of the limits of a RemyCC's generalizability beyond situations encountered during the design phase. Somewhat to our surprise, for moderate numbers of concurrent flows, the RemyCCs continued to surpass (albeit narrowly) the best human-designed algorithms, even ones benefiting from in-network assistance. See Figures 7 and 8.

5.4 Differing RTTs

We investigated how the RemyCCs allocate throughput on a contested bottleneck link when the competing flows have different RTTs. At the design stage, all contending flows had the same RTT (which was drawn randomly for each network specimen from between 100 ms and 200 ms), so the RemyCCs were not designed to exhibit RTT fairness explicitly.

We compared the RemyCCs with Cubic-over-sfqCoDel by running 128 realizations of a four-sender simulation where one sender-receiver pair had RTT of 50 ms, one had 100 ms, one 150 ms, and XCP, in particular, depends on knowing the speed of the link exactly; in our tests on cellular traces we supplied XCP with the long-term average link speed for this value.
Cellular: n=8

In Figures 4 and 5, the RemyCCs do not simply have better median performance — they are also more fair to individual flows, in that the performance of an individual sender (indicated by the size of the ellipses) is more consistent in both throughput and delay.

To explain this result, we investigated how multiple RemyCC flows share the network. We found that when a new flow starts, the system converges to an equitable allocation quickly, generally after little more than one RTT. Figure 6 shows the sequence of transmissions of a new RemyCC flow that begins while sharing the link. Midway through the flow, the competing traffic departs, allowing the flow to start consuming the whole bottleneck rate.

5.3 Cellular Wireless Links

Cellular wireless links are tricky for congestion-control algorithms because their link rates vary with time.

By running a program that attempts to keep a cellular link backlogged but without causing buffer overflows, we measured the variation in download speed on Verizon's and AT&T's LTE service while mobile. We then ran simulations over these pre-recorded traces, with the assumption that packets are enqueued by the network until they can be dequeued and delivered at the same instants seen in the trace.

As discussed above, we did not design the RemyCCs to accommodate such a wide variety of throughputs. Running the algorithm over this link illustrated some of the limits of a RemyCC's generalizability beyond situations encountered during the design phase.

Somewhat to our surprise, for moderate numbers of concurrent flows, \( n \leq 8 \), the RemyCCs continued to surpass (albeit narrowly) the best human-designed algorithms, even ones benefiting from in-network assistance. See Figures 7 and 8.

5.4 Differing RTTs

We investigated how the RemyCCs allocate throughput on a contested bottleneck link when the competing flows have different RTTs. At the design stage, all contending flows had the same RTT (which was drawn randomly for each network specimen from between 100 ms and 200 ms), so the RemyCCs were not designed to exhibit RTT fairness explicitly.

We compared the RemyCCs with Cubic-over-sfqCoDel by running 128 realizations of a four-sender simulation where one sender-receiver pair had RTT of 50 ms, one had 100 ms, one 150 ms, and...
Figure 9: AT&T LTE downlink trace, \( n = 4 \). Two of the Remy-CCs are on the efficient frontier.

Figure 10: Remy's RTT unfairness compares favorably to Cubic-over-sfqCoDel. Error bar represents standard error of the mean over 128 100-second simulations.

5.5 Datacenter-like topology

We simulated 64 connections sharing a 10 Gbps datacenter link, and compared DCTCP [2] (using AQM inside the network) against a RemyCC with a 1000-packet tail-drop queue. The RTT of the path in the absence of queueing was 4 ms. Each sender sent 20 megabytes on average (exponentially distributed) with an “off” time between its connections exponentially distributed with mean 100 milliseconds.

We used Remy to design a congestion-control algorithm to maximize \( \frac{1}{\text{throughput}} \) (minimum potential delay) over these network parameters, with the degree of multiplexing assumed to have been drawn uniformly between 1 and 64.

The results for the mean and median throughput (tput) for the 20 megabyte transfers are shown in the following table:

<table>
<thead>
<tr>
<th>tput: mean, med</th>
<th>rtt: mean, med</th>
</tr>
</thead>
<tbody>
<tr>
<td>DCTCP (ECN)</td>
<td>179, 144 Mbps 7.5, 6.4 ms</td>
</tr>
<tr>
<td>RemyCC (DropTail)</td>
<td>175, 158 Mbps 34, 39 ms</td>
</tr>
</tbody>
</table>

These results show that a RemyCC trained for the datacenter-network parameter range achieves comparable throughput at lower variance than DCTCP, a published and deployed protocol for similar scenarios. The per-packet latencies (and loss rates, not shown) are higher, because in this experiment RemyCC operates over a DropTail bottleneck router, whereas DCTCP runs over an ECN-enabled RED gateway that marks packets when the instantaneous queue exceeds a certain threshold. Developing RemyCC schemes for networks with ECN and AQM is an area for future work.

5.6 Competing protocols

We investigated the possibility of incremental deployment of a RemyCC, by simulating a single bottleneck link with one RemyCC flow contending with one flow from either Compound or Cubic, with no active queue management. The RemyCC was designed for round-trip-times between 100 ms and 10 s, in order to accommodate a “buffer-filling” competitor on the same bottleneck link.

We used the same observed traffic distribution from Figure 3 and varied the mean “off” time (exponentially distributed) of the senders. The bottleneck link speed was 15 Mbps and baseline RTT was 150 ms. We also experimented with flows of mean sizes 100 kilobytes and 1 megabyte, with an exponentially distributed mean “off” time of 0.5 seconds between successive flows.

The results, shown in the two tables below, depended on the duty cycle of the senders dictated by the mean off time (numbers in parentheses are standard deviations).

<table>
<thead>
<tr>
<th>Mean off time</th>
<th>RemyCC tput</th>
<th>Compound tput</th>
</tr>
</thead>
<tbody>
<tr>
<td>200 ms</td>
<td>2.12 (.11) Mbps</td>
<td>1.79 (.18) Mbps</td>
</tr>
<tr>
<td>100</td>
<td>2.18 (.08)</td>
<td>2.75 (.27)</td>
</tr>
<tr>
<td>10</td>
<td>2.28 (.10)</td>
<td>3.9 (.13)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Mean size</th>
<th>RemyCC tput</th>
<th>Cubic tput</th>
</tr>
</thead>
<tbody>
<tr>
<td>100 KBytes</td>
<td>2.04 (.14)</td>
<td>1.31 (.16)</td>
</tr>
<tr>
<td>1 MByte</td>
<td>2.09 (.11)</td>
<td>1.28 (.11)</td>
</tr>
</tbody>
</table>

We observe that this RemyCC does well at low duty cycles because it is able to grab spare bandwidth more quickly. At higher duty cycles (with low mean off time), Cubic and Compound tend to grab a higher share of the bandwidth. The results, however, are close enough that we believe a RemyCC designed for competing with more aggressive protocols may close the gap, while retaining high performance when competing only with like-minded Remy-CCs.
Datacenters

● Issues
  ○ Many synchronous requests -> incast
  ○ Diverse mix of short and long flows

● Simulation
  ○ 64 connections, 10 Gpbs link
  ○ Objective function maximizes throughput
  ○ Comparable throughputs to DCTCP
  ○ DCTCP has shorter RTTs
Sensitivity of design range

The diagram shows the sensitivity of RemyCC's performance over a range of link speeds. RemyCC is designed with different prior information about the network, and its performance varies accordingly. The diagram includes a logarithmic plot of normalized throughput against delay, with link speed on the x-axis. RemyCC 1x (link speed known a priori) performs better within its design range compared to RemyCC 10x (designed for link speeds in shaded region). Cubic-over-sfqCoDel is also shown for reference.

5.8 Summary of results

The results show that more-specific prior knowledge is helpful and improves performance — when it happens to be correct. The results are shown in Figure 11. On the particular link for which RemyCC, it beats Cubic-over-sfqCoDel, but again RemyCC was designed, it performs the best, but its gains in median throughput and decrease in median queueing delay against Cubic over-sfqCoDel as the link speed varies.

5.7 How helpful is prior knowledge about the network?

On networks whose parameters mostly obeyed the prior knowledge supplied at design-time, Remy's performance deteriorated when the true network violated its design assumptions. But when a RemyCC's assumptions aren't met, performance deteriorates. RemyCC achieved a 40% increase in median throughput and a 7% decrease in median queueing delay against Cubic-over-sfqCoDel, generally thought to be excellent general-purpose congestion-control algorithms.

We investigated the performance benefit conferred by having algorithms that cooperate indirectly across a network to achieve a common goal. Although the RemyCCs appear to work well on networks they will encounter, or they have different goals in mind, it is not easy to say why one flavor of TCP or tweak may be preferred over another. This formulation allows users and system designers ultimately care about, we believe, is the quality and consistency of overall behavior.

Our approach focuses on maximizing a well-specified overall objective at the cost of complex endpoint algorithms. We think this tradeoff is advisable: today's endpoints can be designed within a range of link speeds, which the "10x RemyCCs designed for broader classes of networks will likely be prepared for — even beating in-network schemes at their own game when they work, other than the observation that they seem to optimize for a particular latency, but other than that, they break catastrophically in such situations.

How would a RemyCC designed for a 10,000-fold range of link speeds perform on real networks without trying them? All that said, we have much to learn before computer-generated algorithms, much less decentralized algorithms, will perform on real networks without trying them. Much remains unknown about the capabilities and limits of computer-generated algorithms, much less decentralized algorithms.

6. DISCUSSION

By contrast, our approach focuses on maximizing a well-specified overall objective at the cost of complex endpoint algorithms. We think this tradeoff is advisable: today's endpoints can be designed within a range of link speeds, which the "10x RemyCCs designed for broader classes of networks will likely be prepared for — even beating in-network schemes at their own game when they work, other than the observation that they seem to optimize for a particular latency, but other than that, they break catastrophically in such situations.

Much remains unknown about the capabilities and limits of computer-generated algorithms, much less decentralized algorithms. By contrast, our approach focuses on maximizing a well-specified overall objective at the cost of complex endpoint algorithms. We think this tradeoff is advisable: today's endpoints can be designed within a range of link speeds, which the "10x RemyCCs designed for broader classes of networks will likely be prepared for — even beating in-network schemes at their own game when they work, other than the observation that they seem to optimize for a particular latency, but other than that, they break catastrophically in such situations.

How would a RemyCC designed for a 10,000-fold range of link speeds perform on real networks without trying them? All that said, we have much to learn before computer-generated algorithms, much less decentralized algorithms, will perform on real networks without trying them. Much remains unknown about the capabilities and limits of computer-generated algorithms, much less decentralized algorithms.
Discussion: practicality

- Providing network assumptions
- Scalability
- Using CC algorithms we don’t understand
- Coexisting with other protocols
Approaching congestion control differently

- Complex rules but consistent behavior
- Objective and environment driven
- Able to evolve