Rethinking Transport for a Changing Internet

Keith Winstein

MIT Computer Science and Artificial Intelligence Laboratory

June 20, 2013

It's not 1995 anymore

Applications see:

- Intermittent connectivity
- Link speeds that vary in time
- Huge buffers
- Gigantic range of networks
  - 2G cellular to 40+ Gbps in datacenters

Our work

- Design for unreliability and mobility (**Mosh**)
- Design for variability (**Sprout**)
- Design for evolvability (**Remy**)

Secure Shell, 1995

- Connects local terminal to remote terminal.
- Conveys over TCP:
  - user keystrokes → server
  - octet stream (coded screen updates) → client terminal
- Connection endpoints dictated by IP:port on both sides

Post-1995 problems with SSH

- Can't roam:
  - ...across Wi-Fi networks.
  - ...from Wi-Fi to cell or vice versa.
- Times out.
- TCP responds poorly to packet loss.
- Reliable byte stream is wrong layer of abstraction.

Our solution: Mosh (mobile shell)

- State Synchronization Protocol
- Secure single-packet roaming
- Rolling latency compensation

KW and Hari Balakrishnan, Mosh: An Interactive Remote Shell for Mobile Clients, in *USENIX ATC*, June 13–15, 2012, Boston, Mass.

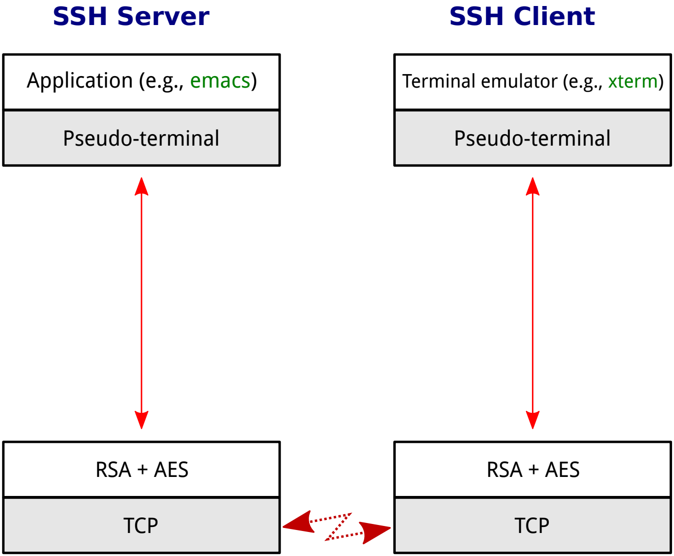
State Synchronization Protocol

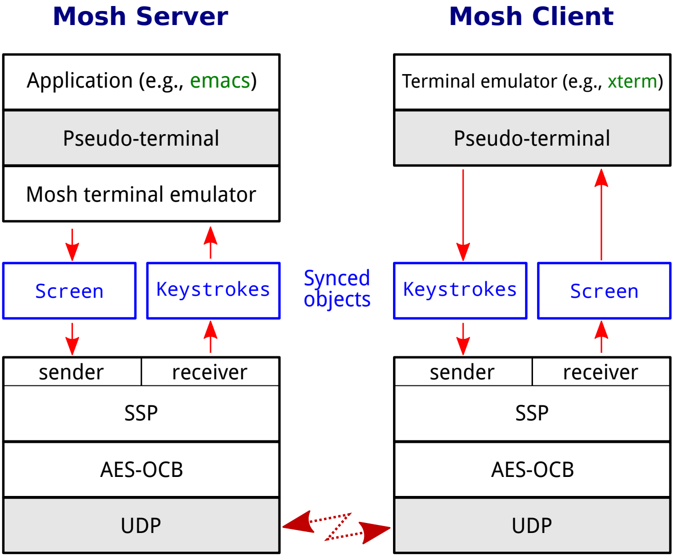
- Runs over UDP.
- Instead of synchronizing *octet streams*, synchronize *objects*.
- Implements simple interface:
  - `diff`: make vector from state $A \rightarrow B$
  - `patch`: apply vector to A, producing B
- Object implementation defines synchronization semantics.

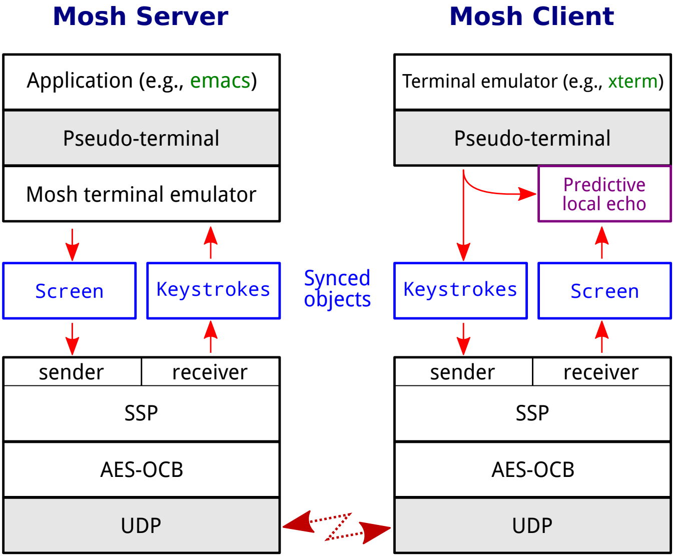
Secure single-packet roaming

- Every datagram is idempotent operation
- Protected by AES-OCB (offset codebook)
- Source addr of latest authentic packet $\Rightarrow$ new target.
- One packet is enough to roam.

SSH Server / SSH Client

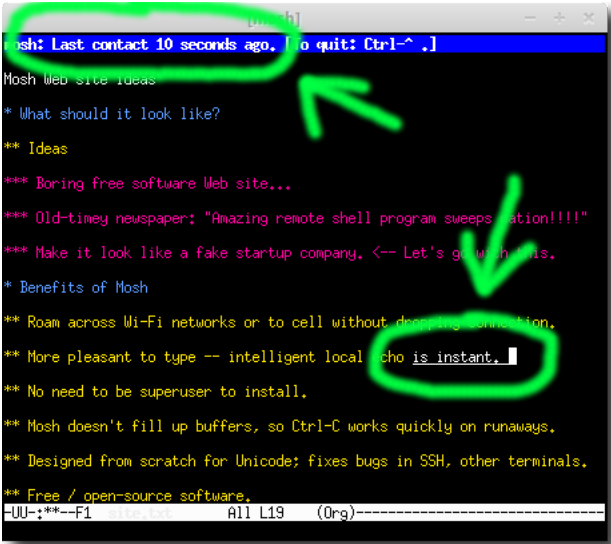



Mosh Server
Mosh Client
Application (e.g., emacs)
Pseudo-terminal
Mosh terminal emulator
Screen
Keystrokes
sender
receiver
SSP
AES-OCB
UDP
Synced objects
Terminal emulator (e.g., xterm)
Pseudo-terminal
Keystrokes
Screen
sender
receiver
SSP
AES-OCB
UDP


Mosh Server
Mosh Client
Application (e.g., emacs)
Pseudo-terminal
Mosh terminal emulator
Screen
Keystrokes
sender
receiver
SSP
AES-OCB
UDP
Synced objects
Terminal emulator (e.g., xterm)
Pseudo-terminal
Predictive local echo
Keystrokes
Screen
sender
receiver
SSP
AES-OCB
UDP

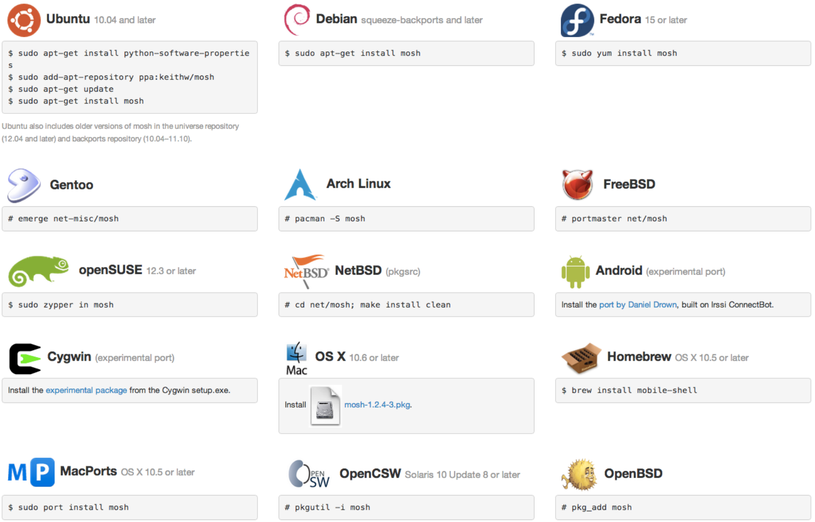
Rolling latency compensation

- Client anticipates server response.
- Runs predictive model in the background.
  - If user hits keystroke, predict key will appear where cursor was.
- Underline still-outstanding predictions.


mosh: Last contact 10 seconds ago. [To quit: Ctrl-^ .]
Mosh Web site ideas
* What should it look like?
** Ideas
*** Boring free software Web site...
*** Old-timey newspaper: "Amazing remote shell program sweeps nation!!!!"
*** Make it look like a fake startup company. <-- Let's go with this.
* Benefits of Mosh
** Roam across Wi-Fi networks or to cell without dropping connection.
** More pleasant to type -- intelligent local echo is instant.
** No need to be superuser to install.
** Mosh doesn't fill up buffers, so Ctrl-C works quickly on runaways.
** Designed from scratch for Unicode; fixes bugs in SSH, other terminals.
** Free / open-source software.
-UU-:**--F1 site.txt All L19 (Org)-----------------------------

Demo

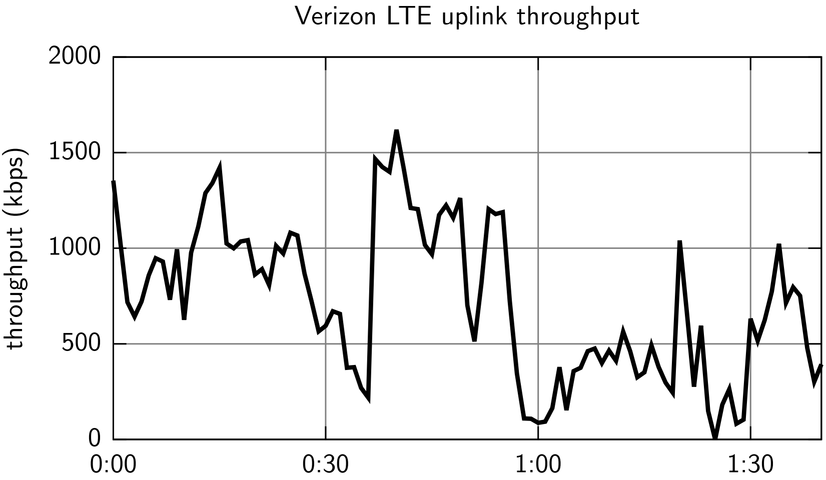
Deployment


Ubuntu 10.04 and later
$ sudo apt-get install python-software-propertie
s
$ sudo add-apt-repository ppa:keithw/mosh
$ sudo apt-get update
$ sudo apt-get install mosh
Ubuntu also includes older versions of mosh in the universe repository (12.04 and later) and backports repository (10.04–11.10).
Debian squeeze-backports and later
$ sudo apt-get install mosh
Fedora 15 or later
$ sudo yum install mosh
Gentoo
emerge net-misc/mosh
Arch Linux
pacman -S mosh
FreeBSD
portmaster net/mosh
openSUSE 12.3 or later
$ sudo zypper in mosh
NetBSD (pkgsrc)
cd net/mosh; make install clean
Android (experimental port)
Install the port by Daniel Drown, built on Irssi ConnectBot.
Cygwin (experimental port)
Install the experimental package from the Cygwin setup.exe.
OS X 10.6 or later
Mac
Install mosh-1.2.4-3.pkg.
Homebrew OS X 10.5 or later
$ brew install mobile-shell
MacPorts OS X 10.5 or later
$ sudo port install mosh
OpenCSW Solaris 10 Update 8 or later
pkgutil -i mosh
OpenBSD
pkg_add mosh

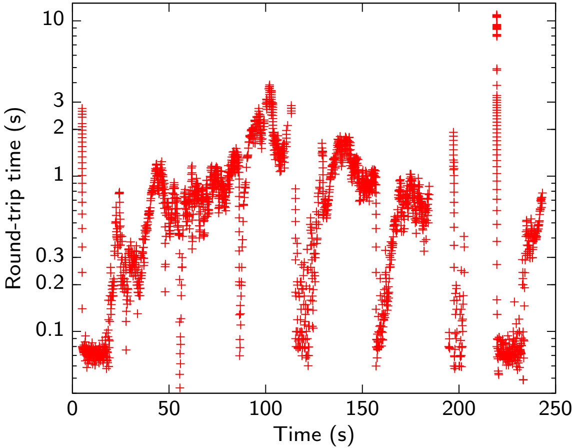
Sprout: When latency and throughput are both important

- Mosh is not a high-throughput app.
- Main concern: UI latency.
- What if application cares about latency **and** throughput?

Cellular networks are **variable**



Cellular networks are **too reliable**

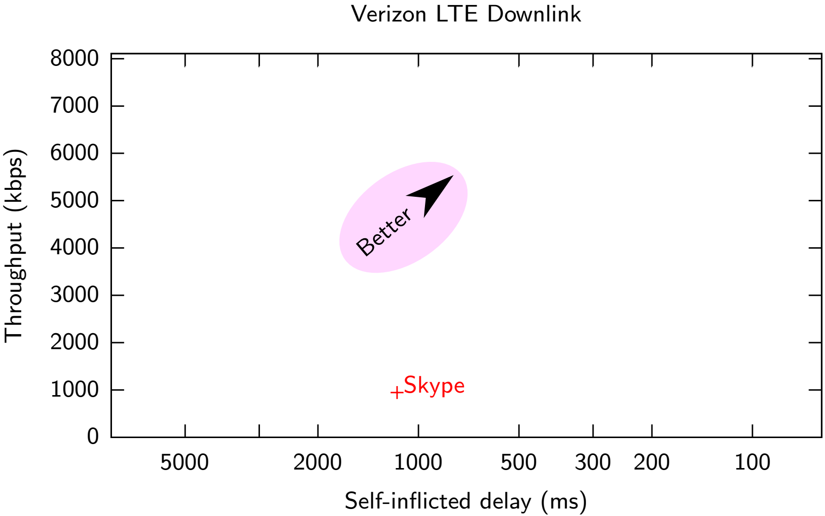


(Verizon LTE, one TCP download.)

Interactive apps work **poorly**

- We measured cellular networks while driving:
  - **Verizon LTE**
  - Verizon 3G (1xEV-DO)
  - AT&T LTE
  - T-Mobile 3G (UMTS)
- Then ran apps across replayed network trace:
  - **Skype** (Windows 7)
  - Google Hangout (Chrome on Windows 7)
  - Apple Facetime (OS X)

Performance summary



Why is performance so bad?

- Exiting schemes **react** to congestion signals.
  - Packet loss.
  - Increase in round-trip time.
- Feedback comes too late.
- The killer: **self-inflicted queueing delay**.
- Throughput overshoot means a queue filling up.

Sprout's goal

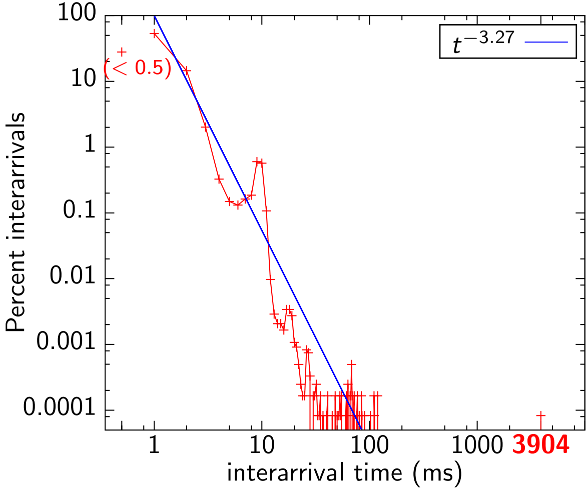
- Most throughput
- Bounded risk of delay > 100 ms

KW, Anirudh Sivaraman, and Hari Balakrishnan, Stochastic Forecasts Achieve High Throughput and Low Delay over Cellular Networks, in *NSDI*, April 2–5, 2013, Lombard, Ill.

Bounded risk of delay

- **Infer** link speed from interarrival distribution.
- **Predict** future link speed.
  - Don't wait for congestion.
- **Control:** Send as much as possible, but require:
  - 95% chance all packets arrive within 100 ms.

Infer: link speed from flicker noise process

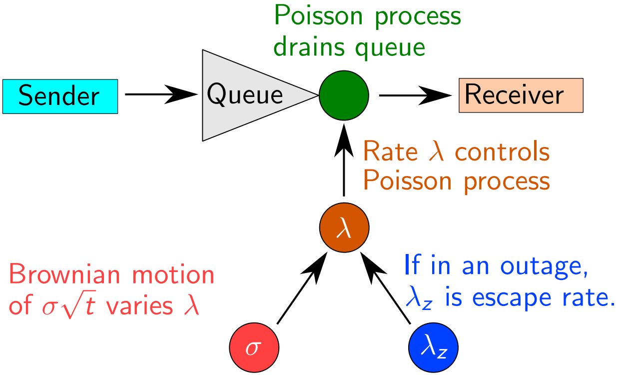


(Verizon LTE, phone stationary, 3 a.m.)

Predict: future link speed

- Model evolution of speed as **random walk**.
  - (Brownian motion)
- Cautious forecast: 5th percentile cumulative packets
- Receiver makes forecast; sends back to sender in ack
- Almost all precalculated

Sprout's model

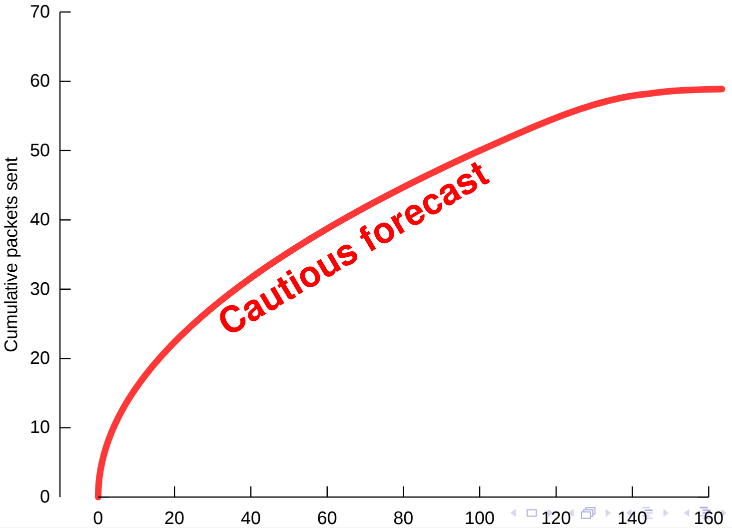


Parameters

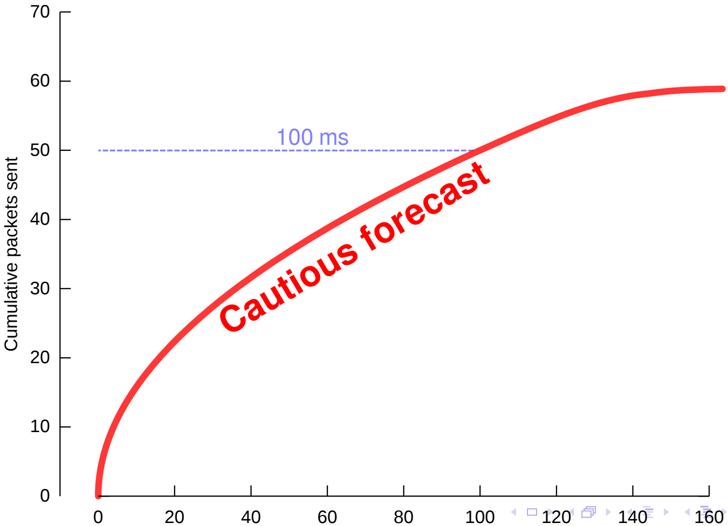
Volatility σ: fixed @	$200 \ \frac{\mathrm{pkts}/s}{\sqrt{s}}$
Expected outage time $1/\lambda_z$:	$1 \ s$
Tick length:	$20 \ ms$
Forecast length:	$160 \ ms$
Delay target:	$100 \ ms$
Risk tolerance:	5%

All source code was **frozen before data collection began**.

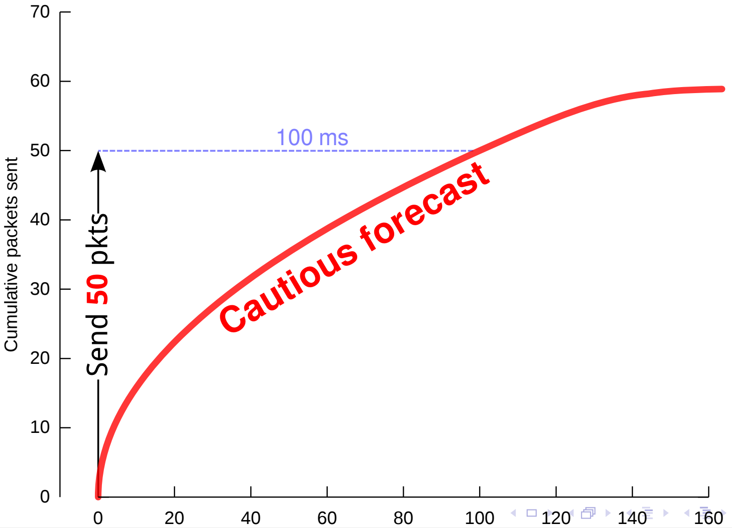
Control: fill up 100 ms forecast window



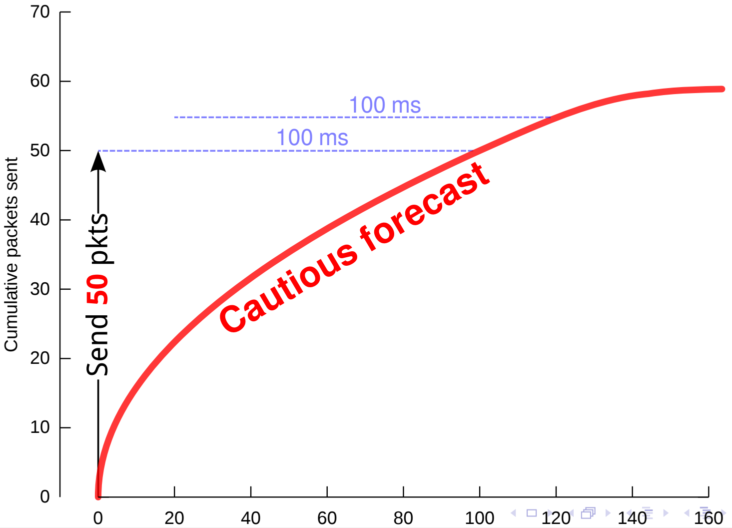
Control: fill up 100 ms forecast window



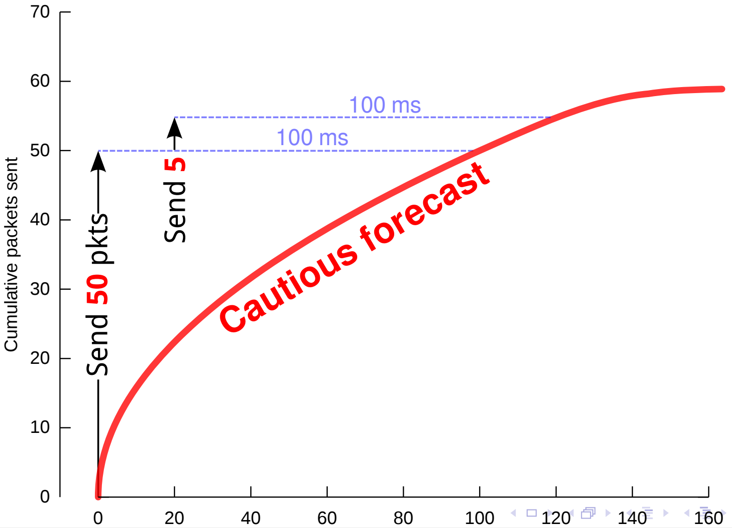
Control: fill up 100 ms forecast window



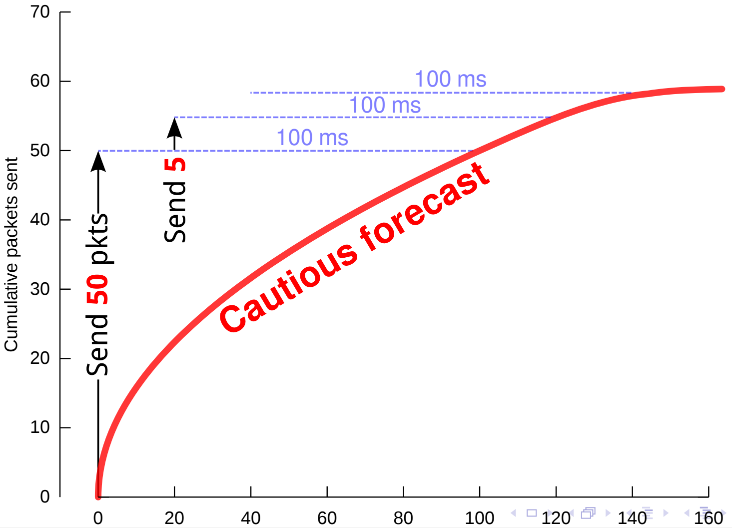
Control: fill up 100 ms forecast window



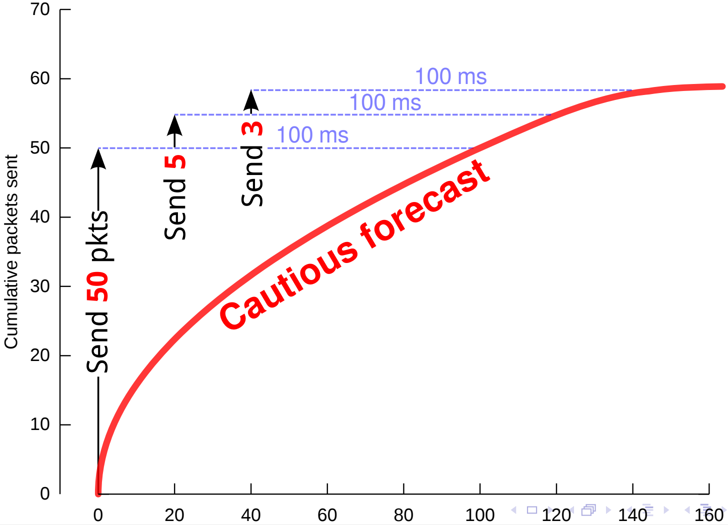
Control: fill up 100 ms forecast window

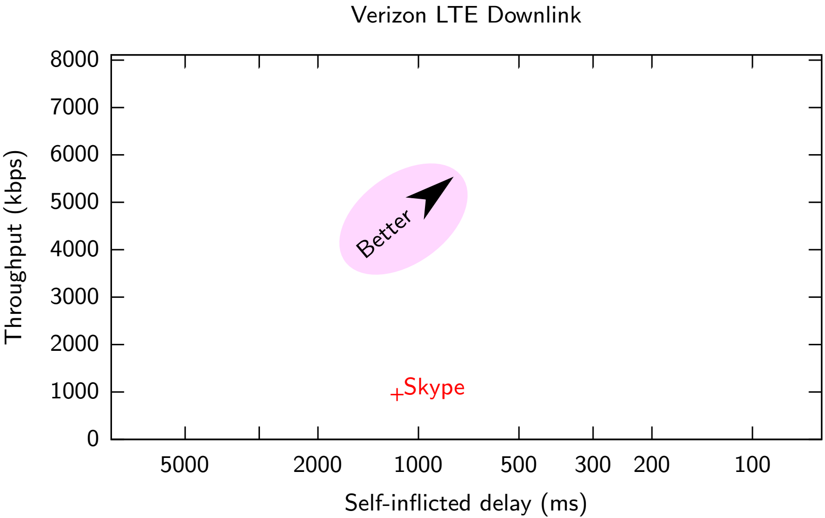


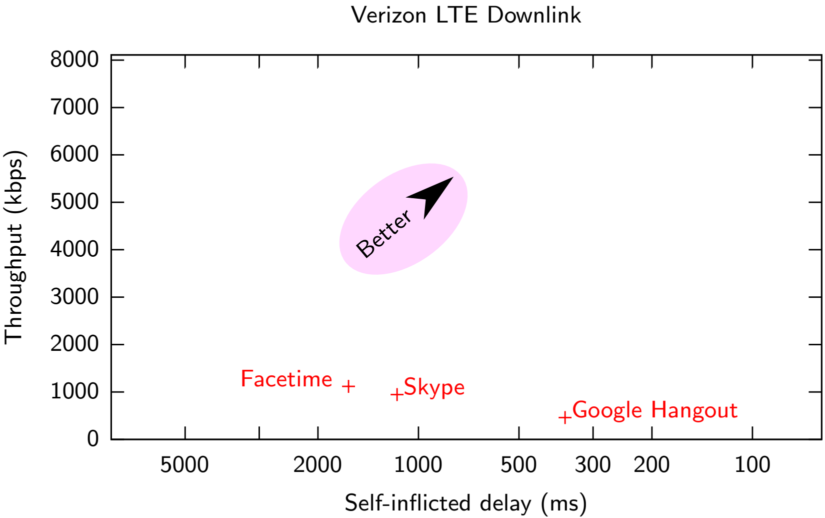
Control: fill up 100 ms forecast window

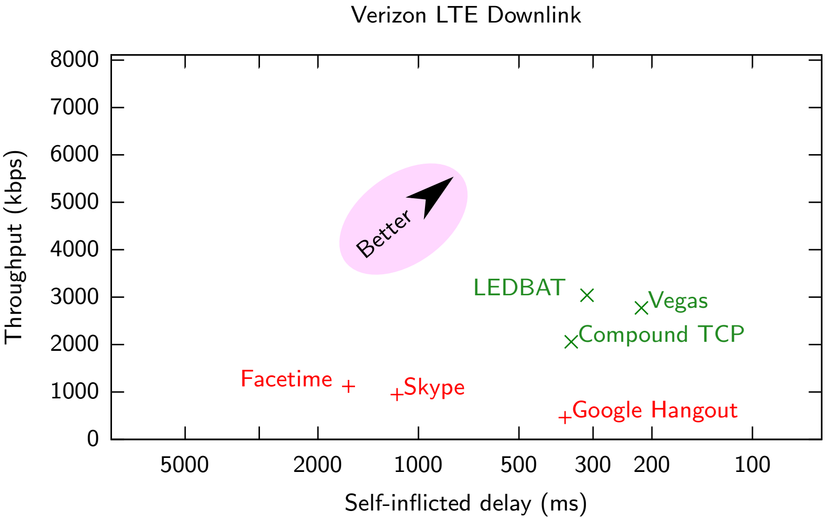


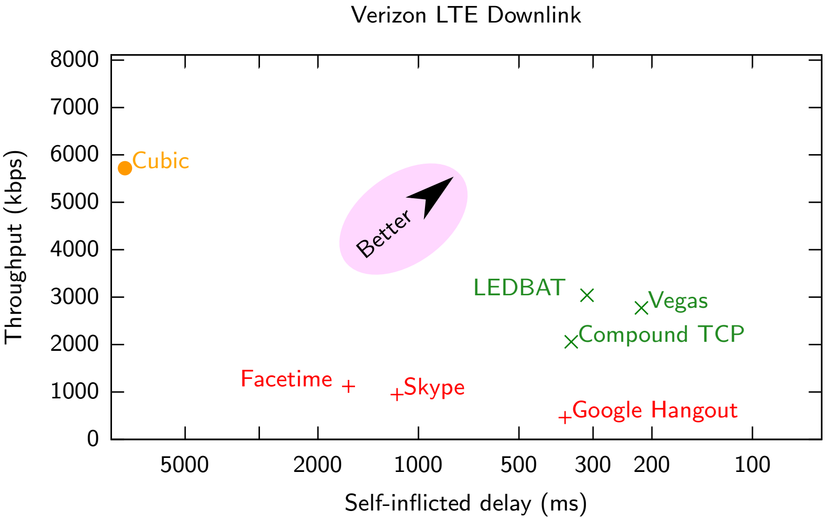
Control: fill up 100 ms forecast window

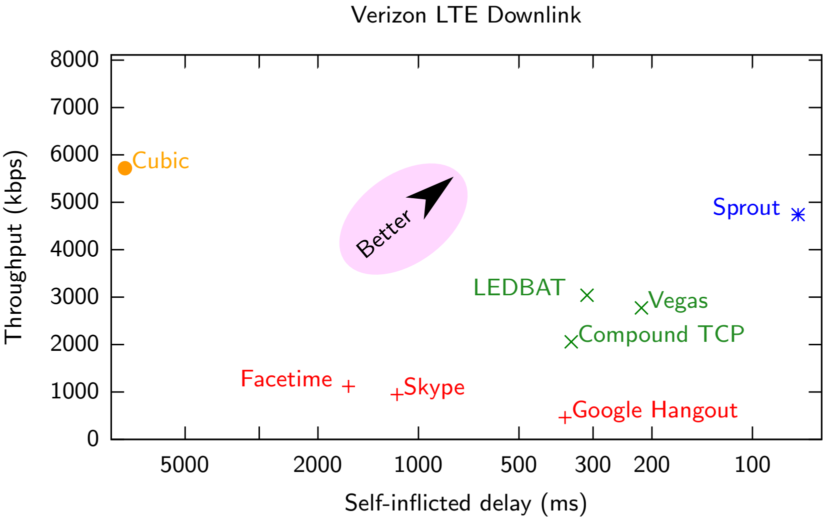


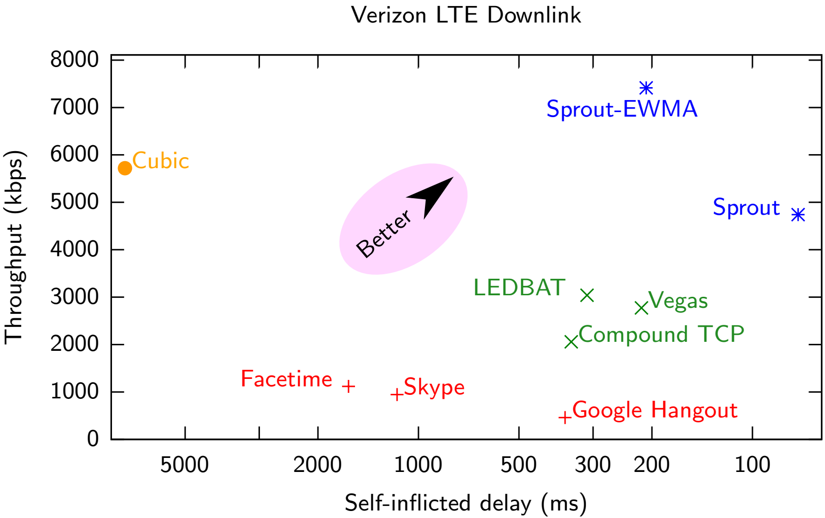

Verizon LTE Downlink
8000
7000
6000
5000
4000
3000
2000
1000
0
Throughput (kbps)
Better
Skype
5000
2000
1000
500
300
200
100
Self-inflicted delay (ms)

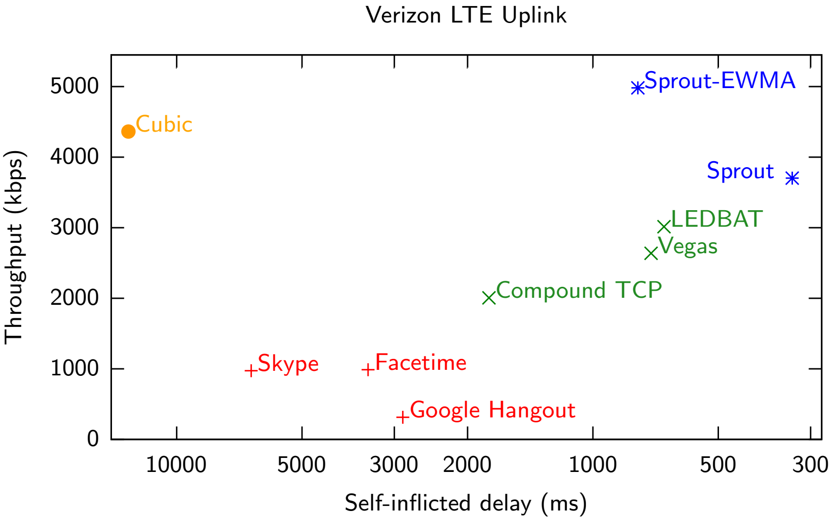

Verizon LTE Downlink
Throughput (kbps)
8000
7000
6000
5000
4000
3000
2000
1000
0
Better
Facetime
Skype
Google Hangout
5000
2000
1000
500
300
200
100
Self-inflicted delay (ms)


Verizon LTE Downlink
8000
7000
6000
5000
4000
3000
2000
1000
0
Throughput (kbps)
Better
LEDBAT
Vegas
Compound TCP
Facetime
Skype
Google Hangout
5000
2000
1000
500
300
200
100
Self-inflicted delay (ms)

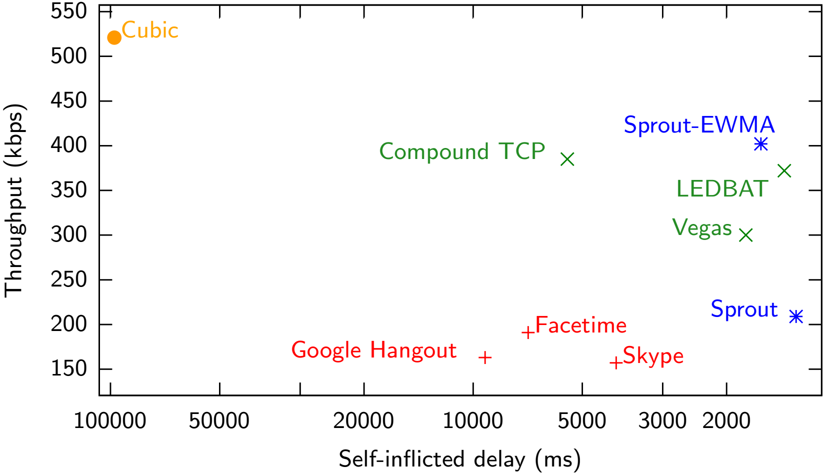

Verizon LTE Downlink
8000
7000
6000
5000
4000
3000
2000
1000
0
Throughput (kbps)
5000
2000
1000
500
300
200
100
Self-inflicted delay (ms)
Cubic
Better
LEDBAT
Vegas
Compound TCP
Facetime
Skype
Google Hangout

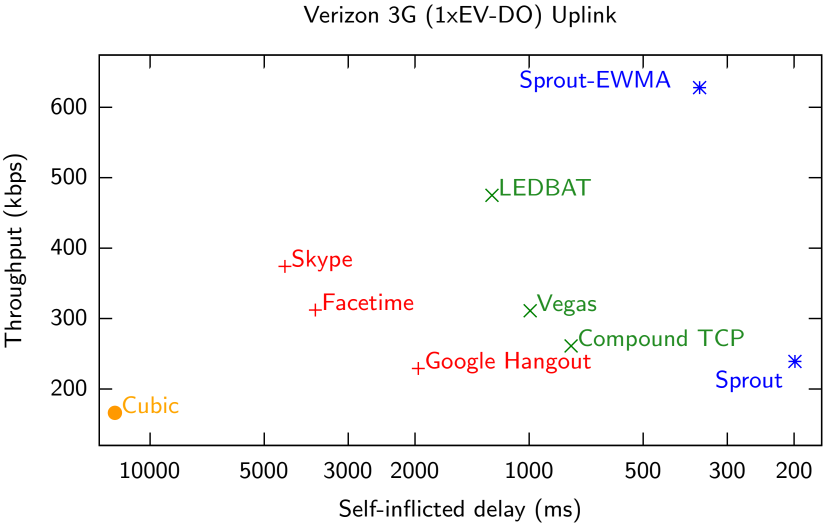

Verizon LTE Downlink
Throughput (kbps)
8000
7000
6000
5000
4000
3000
2000
1000
0
5000
2000
1000
500
300
200
100
Self-inflicted delay (ms)
Cubic
Better
Sprout
LEDBAT
Vegas
Compound TCP
Facetime
Skype
Google Hangout

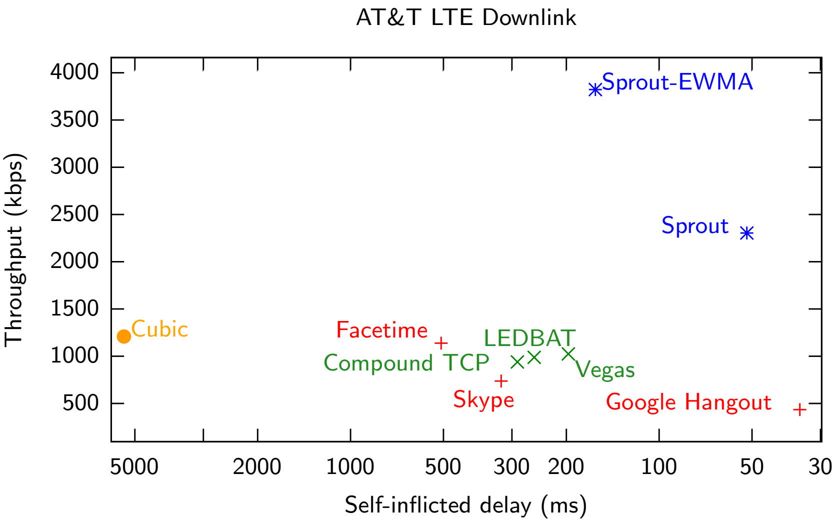

Verizon LTE Downlink
Throughput (kbps)
8000
7000
6000
5000
4000
3000
2000
1000
0
5000
2000
1000
500
300
200
100
Self-inflicted delay (ms)
Cubic
Sprout-EWMA
Better
Sprout
LEDBAT
Vegas
Compound TCP
Facetime
Skype
Google Hangout

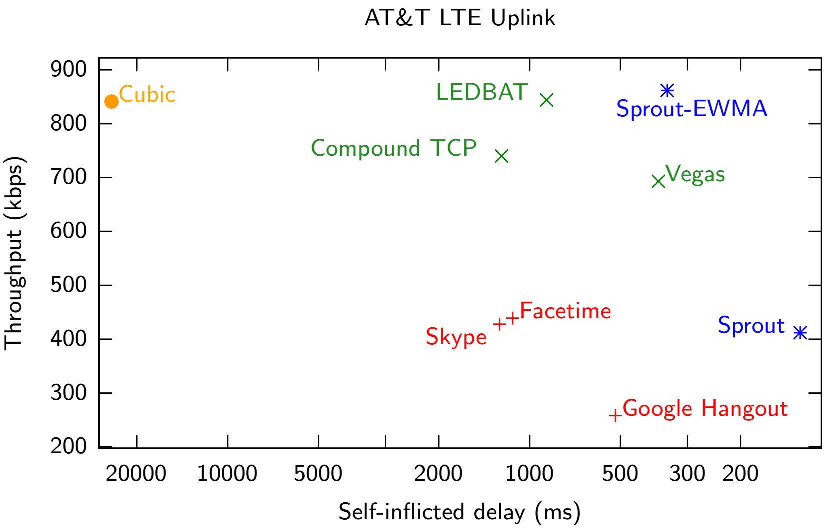

Verizon LTE Uplink
Throughput (kbps)
5000
4000
3000
2000
1000
0
10000
5000
3000
2000
1000
500
300
Self-inflicted delay (ms)
Sprout-EWMA
Cubic
Sprout
LEDBAT
Vegas
Compound TCP
Skype
Facetime
Google Hangout

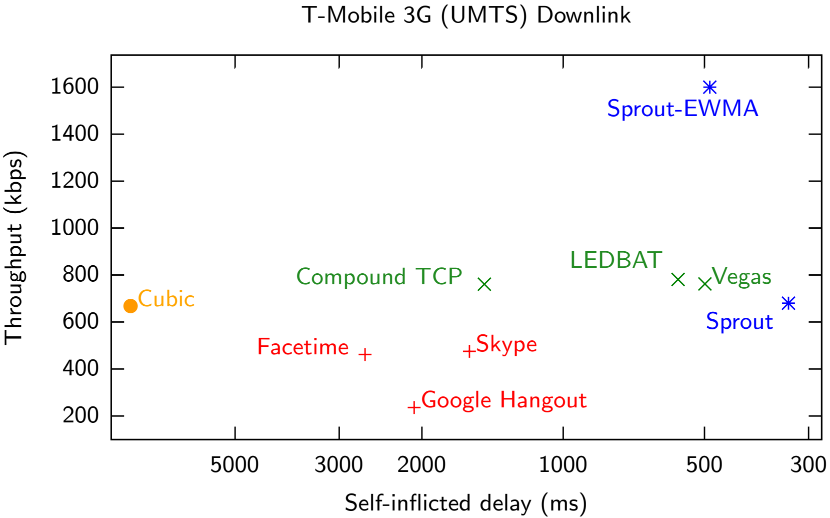
Verizon 3G (1xEV-DO) Downlink

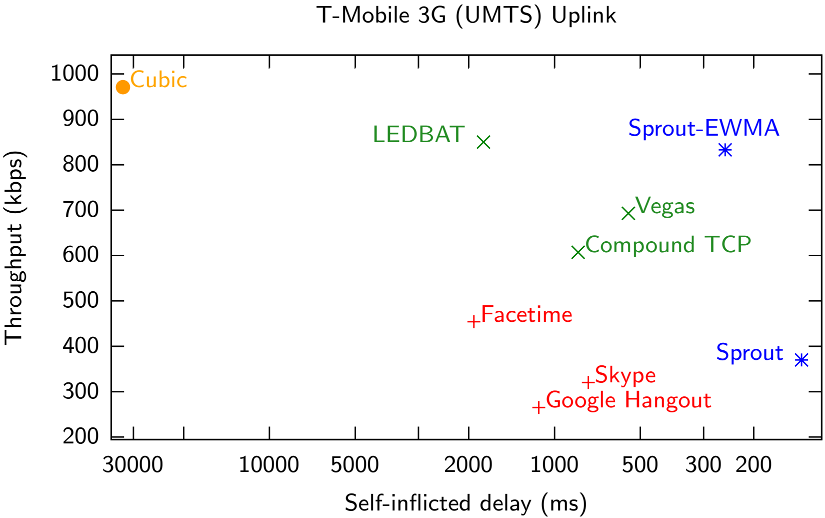



Verizon 3G (1xEV-DO) Uplink
Throughput (kbps)
600
500
400
300
200
Self-inflicted delay (ms)
10000
5000
3000
2000
1000
500
300
200
Sprout-EWMA
LEDBAT
Skype
Facetime
Vegas
Compound TCP
Google Hangout
Sprout
Cubic


AT&T LTE Downlink
Throughput (kbps)
4000
3500
3000
2500
2000
1500
1000
500
Self-inflicted delay (ms)
5000
2000
1000
500
300
200
100
50
30
Sprout-EWMA
Sprout
Cubic
Facetime
LEDBAT
Compound TCP
Vegas
Skype
Google Hangout


AT&T LTE Uplink
Throughput (kbps)
900
800
700
600
500
400
300
200
Cubic
LEDBAT
Sprout-EWMA
Compound TCP
Vegas
Facetime
Skype
Sprout
Google Hangout
20000
10000
5000
2000
1000
500
300
200
Self-inflicted delay (ms)

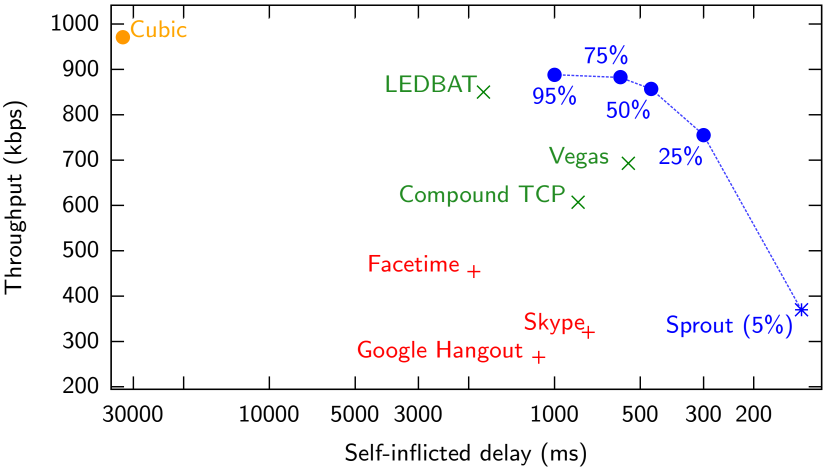

T-Mobile 3G (UMTS) Downlink
Throughput (kbps)
1600
1400
1200
1000
800
600
400
200
Self-inflicted delay (ms)
5000
3000
2000
1000
500
300
Sprout-EWMA
LEDBAT
Compound TCP
Vegas
Cubic
Sprout
Facetime
Skype
Google Hangout


T-Mobile 3G (UMTS) Uplink
Throughput (kbps)
Self-inflicted delay (ms)
1000
900
800
700
600
500
400
300
200
30000
10000
5000
2000
1000
500
300
200
Cubic
LEDBAT
Sprout-EWMA
Vegas
Compound TCP
Facetime
Sprout
Skype
Google Hangout

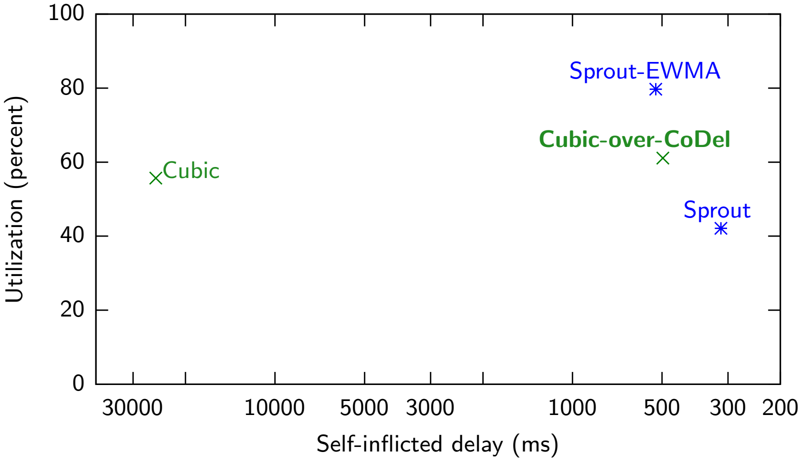
Overall results

Sprout vs.	Avg. speedup	Delay reduction
Skype	2.2×	7.9×
Hangout	4.4×	7.2×
Facetime	1.9×	8.7×
Compound	1.3×	4.8×
TCP Vegas	1.1×	2.1×
LEDBAT	Same	2.8×
Cubic	0.91×	79×

Varying risk tolerance



Competes with AQM even though end-to-end



Replication by Stanford students (February–March 2013)

- Alterman & Quach reproduced some of our measurements
- `http://ReproducingNetworkResearch.wordpress.com/2013/03/12/1216/`
- Won best project award in Stanford networking class!

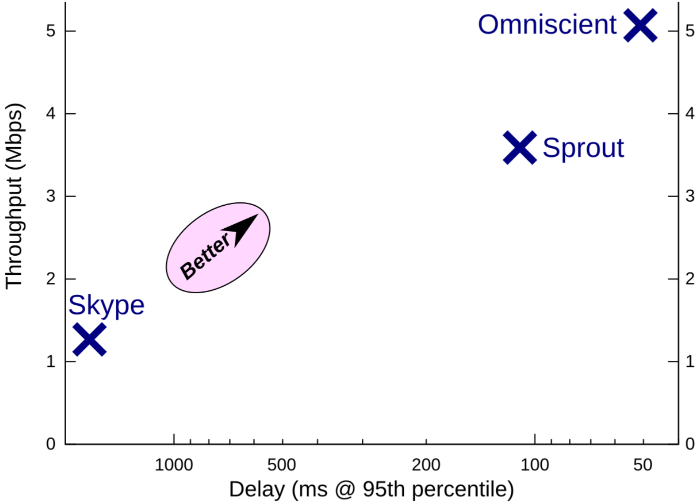
M.I.T. 6.829 contest (March–April 2013)

- Turnkey network emulator, evaluation
- Sender, receiver run in Linux containers
- 4th prize: \$20
- 3rd prize: \$30
- 2nd prize: \$40
- (If beat Sprout) 1st prize:

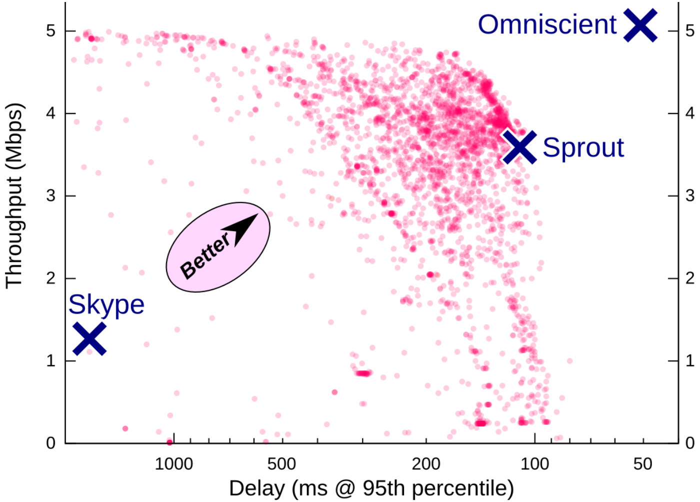
M.I.T. 6.829 contest (March–April 2013)

- Turnkey network emulator, evaluation
- Sender, receiver run in Linux containers
- 4th prize: \$20
- 3rd prize: \$30
- 2nd prize: \$40
- (If beat Sprout) 1st prize: **Co-authorship on future paper**

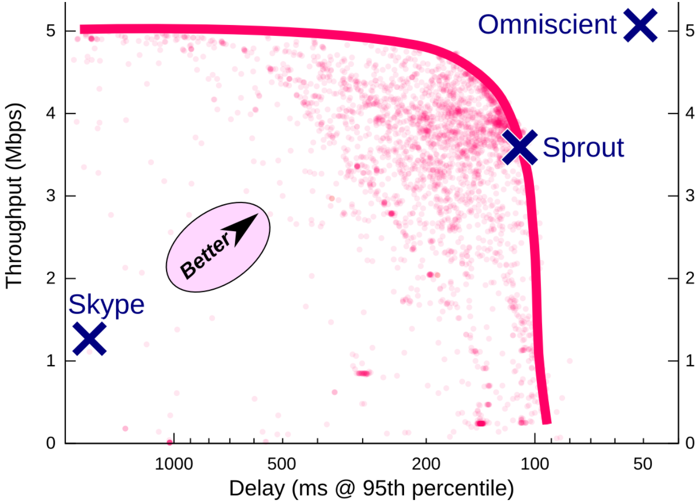
Baseline


Omniscient
Sprout
Skype
Better
Throughput (Mbps)
0
1
2
3
4
5
1000
500
200
100
50
Delay (ms @ 95th percentile)

Land of 3,000 student protocols



Sprout is on the frontier



Our approach

- Pick a model, any model.
- All models are wrong, but they help anyway!
- See if it lands on the frontier.*

* (On a large set of real network paths or newly-collected traces.)

- Kaizen for congestion

Sprout for controlled delay over cellular networks

- **Infer** link speed from interarrival distribution
- **Predict** future link speed
- **Control** risk of large delay with cautious forecast
- Yields 2–4× throughput of Skype, Facetime, Hangout
- Achieves 7–9× reduction in self-inflicted delay
- Matches active queue management **without router changes**
- http://alfalfa.mit.edu

Can we take humans out of the loop?

Sprout is a **human-designed protocol** for:

- **one** kind of network
- with **one** user
- for **one** goal.

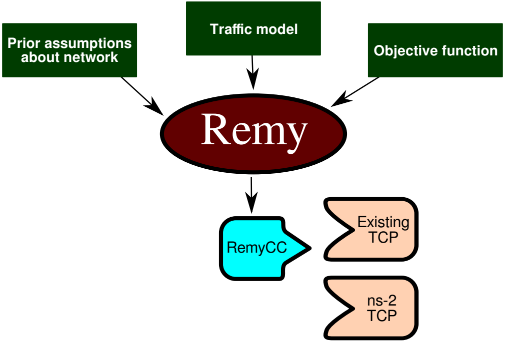
Congestion control: the march of mechanisms

- TCP Reno
- NewReno
- SACK
- Cubic (default in Linux, Android)
- Compound (Windows)
- XCP
- RCP
- DCTCP
- Sprout
- ...

What does congestion control achieve?

- Link layers try to accommodate TCP, but...
- "Teleology of TCP" is mostly unknown.
- Overall behavior is complex and unstable.
- Solutions for long-running flows only.

Remy: Start with goal, work backwards to algorithm



KW and Hari Balakrishnan, TCP ex Machina: Computer-Generated Congestion Control, forthcoming in *SIGCOMM*, August 12–16, 2013, Hong Kong, China.

Prior knowledge

- Uncertain, stochasic model for the network
  - Link speed distribution
  - Delay distribution
- Traffic model
  - "Conversation"-like (time-based)
  - Datacenter-like workload
  - Web browsing

Objective function

- Tradeoff between **throughput** and **delay**
- Tradeoff between **efficiency** and **fairness**
- Pareto-efficiency

Alpha-fairness

$$U_\alpha(x) = \frac{x^{1-\alpha}}{1-\alpha}$$

- "Most fair" Pareto-efficient utility function
- $\alpha = 0$: efficiency only
- $\alpha = 2$: min. potential delay fairness
- $\alpha \to \infty$: maximin fairness
- $\alpha \to 1$: proportional fairness ($\log(x)$)

Objective

$$\log(\text{throughput}) - \delta \log(\text{delay})$$

Other options:

- average flow completion time
- average transaction completion time
- 95th percentile transaction completion time
- ...

What is this problem?

- Decentralized end-to-end algorithm
- Routing is fixed
- Each sender only gets its own receiver's acknowledgements
- Decentralized partially-observable Markov decision process (Dec-POMDP)

Optimal solution is intractable

Arbitrary algorithm relates:

- Full history of acknowledgements
- Full history of packets sent

. . . to decision about when to send the next packet.

Search for algorithm is NEXP-complete.

Simplifying the state

Instead, keep limited state variables:

1. Moving average of interval between acknowledgements
2. Moving average of interval between sender timestamps reflected in acks
3. Ratio of latest RTT to smallest RTT seen so far

The action

1. Increment to congestion window
2. Multiple to congestion window
3. Upper bound on rate of sending

Remy's job

Rules relate sections of state space to actions.

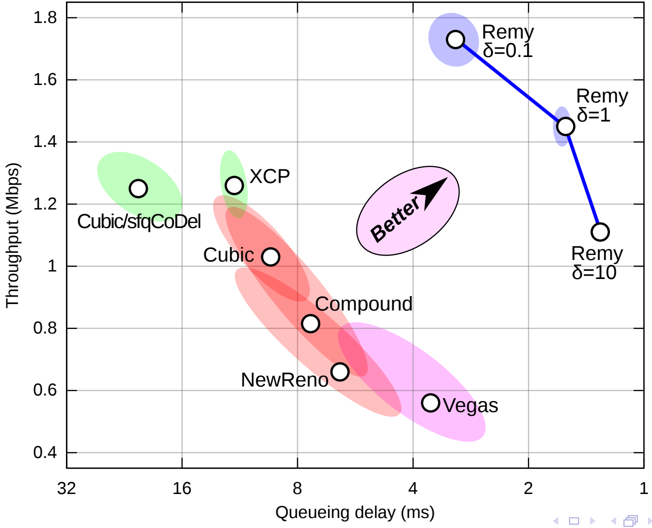
The task: find best set of rules to maximize expected value of objective function.

The algorithm

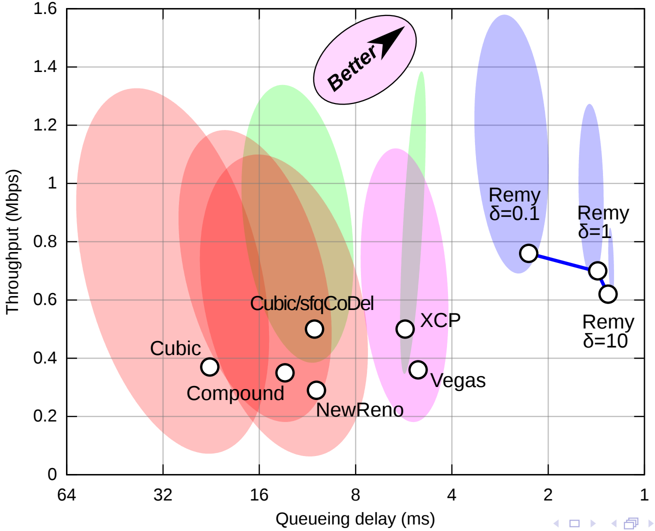
- Initially: one default rule for whole state space
- Find best action for whole state space
- Subdivide rule at median query $\rightarrow$ 8 new rules
- Repeat

Optimize existing rules and rule structure **in parallel**.

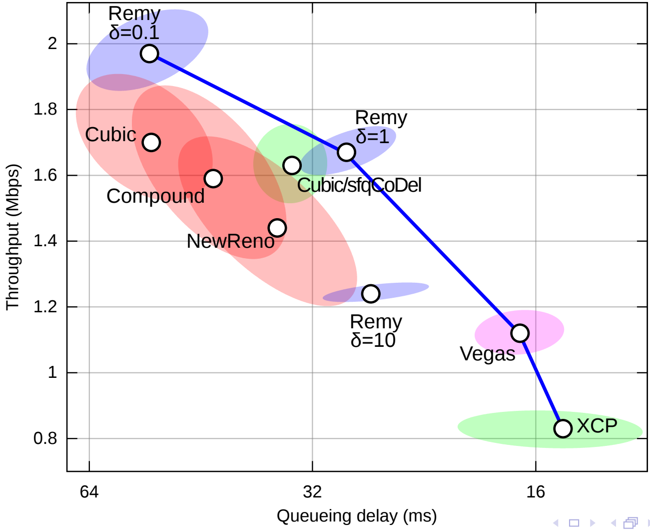
Fixed 15 Mbps link, 8 senders, flows exp-distributed



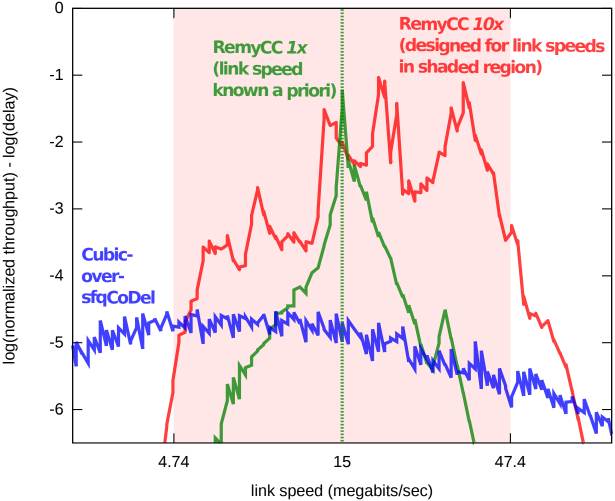
Fixed 15 Mbps link, 12 senders, heavy-tailed flows



Verizon LTE, 8 senders, flows exp-distributed



Prior knowledge is helpful, when correct



Why does Remy work?

- Not entirely clear!
- Need to reverse-engineer algorithms.
- Hundreds of rules — are they all necessary?

Goal-driven algorithm **moves** the complexity

Human-designed algorithm:

- Simple algorithm
- Complex and subpar emergent behavior
- . . . worse when implicit assumptions not met

Computer-designed algorithm:

- Complex algorithm
- Consistent and good emergent behavior
- . . . much worse when stated assumptions not met

Evolvability

Status quo:

- link layer constrained by need for TCP to perform
- apps add hacks to get around TCP

Evolvable transport:

- accommodate whatever link layer does & app wants

Conclusions

- Computer-designed > human-designed
- End-to-end > in-network
- Focus on goal and assumptions > focus on mechanism

Summary

Mosh: make every packet count on **unreliable** and **mobile** networks

Sprout: compromise throughput vs. delay on **variable** networks

Remy: find the best schemes for **evolving** networks and apps

http://mosh.mit.edu http://alfalfa.mit.edu keithw@mit.edu