Updating TCP to support Variable-Rate Traffic

draft-fairhurst-tcpm-newcwv-01.txt

Gorry Fairhurst

Israfil Biswas

{gorry, israfil}@erg.abdn.ac.uk

Electronics Research Group

School of Engineering

University of Aberdeen

Scotland, UK

TCP behaviour for variable-rate applications

"Traditional" TCP applications are "bulk" or "thin"

This differentiation is no longer true for many applications:

- Streaming over TCP
- "interactive applications"
- persistent web connections
- many more examples of variable-rate applications

Two characteristic behaviours:

- Burst applications with **idle periods**
- **Application-limited** transmission rate

Rethinking TCP CC for Applications

Should sender hold CC state for arbitrary time?

Burst applications with **idle periods:**

- *What is the allowed rate after an idle period?*
- Standard TCP – uses RW
- CWV experimental update to decay cwnd
- Neither really satisfactory for application

Application-limited transmission rate:

- *What value of cwnd is acceptable?*
- Standard TCP does not reduce cwnd
- CWV collapses cwnd each RTT

Congestion Window Validation

RFC 2861 published 2000

A quick survey of TCP implementations:

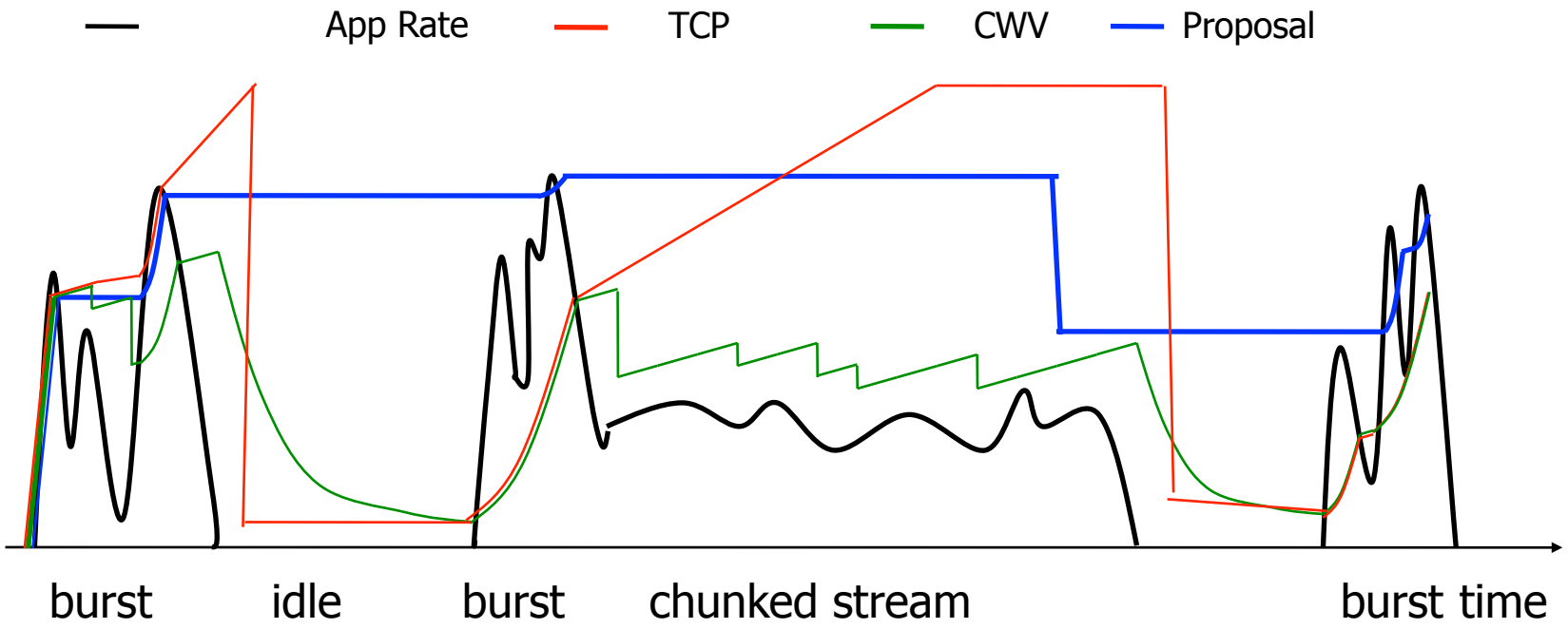
- Linux enables CWV by default
  - Many “interactive” apps disable CWV
- Vista, Net/FreeBSD restart from IW
- Mac OS does not reduce
- Solaris does not reduce

No single widely-used method

App-limited and ***idle*** are hard to separate in practice

How should cwnd be managed?

Simplified diagram showing the differences:



The proposal

Create two sender phases:

- **Normal (validated): flightsize ~ cwnd**

  standard management of cwnd

- **Nonvalidated: flightsize < (2/3)*cwnd**

  preserve cwnd (no growth)

  At the end of the Nonvalidated phase (6 mins):

  cwnd = max(2*w_used, IW) ; twice recently used capacity
  ssthresh = max(ssthresh, 3*cwnd/4) ; avoids excessive overshoot, as noted in RFC 2861

  Exits before 6 mins when:

  • Congestion feedback

  sender uses SACK to set cwnd; exits nonvalidated phase

  • RTO expires

  Resets cwnd to RW; exits nonvalidated phase

How long should we preserve cwnd ?

cwnd preserved during non-validated phase

Can't be an arbitrary period

There is no "optimum"

Ideal time is a compromise

Longer than "normal" periods of app inactivity

Not significantly longer than network regarded stable

Our first proposal is 6 minutes

- Most networks relatively stable (several minutes)
- Complex topologies do exits
- Fluctuation in path capacity uncommon (e.g. wireless)

Performance usually similar to current methods

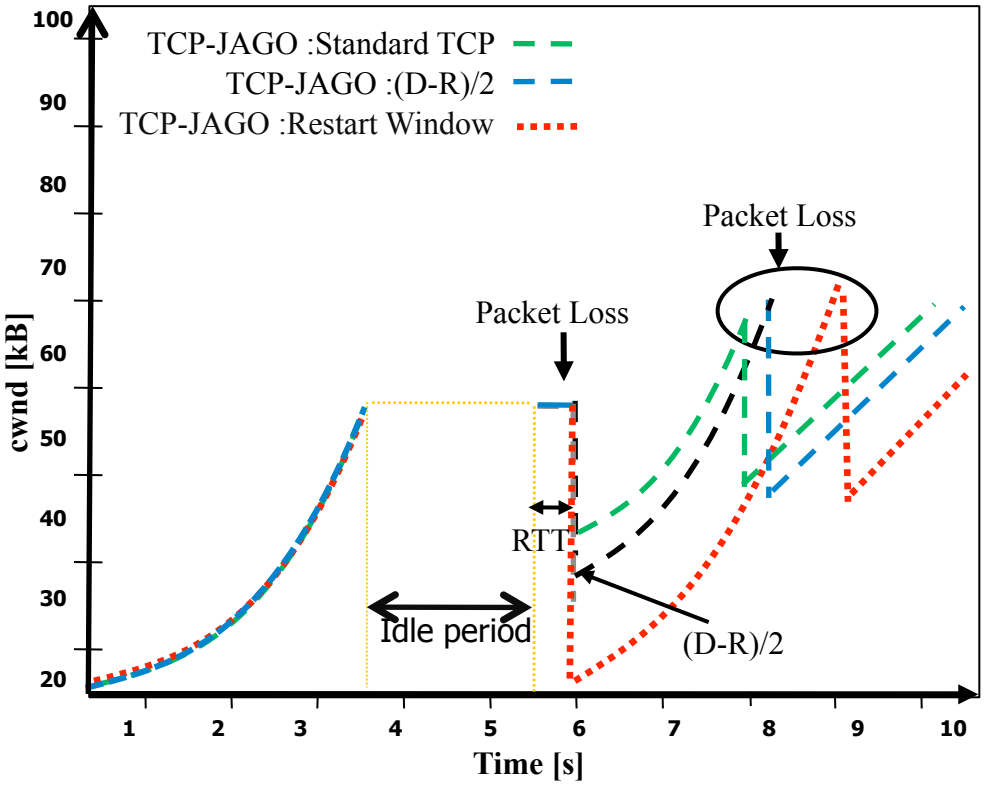
Selection of CC response in non-validated phase

A transient change in network state could reduce the available capacity

Evaluated 3 responses in Nonvalidated phase:

- Restart from RW
- Halve cwnd (as per TCP standard)
- Estimate successfully used capacity (D-R)*

Last method chosen, similar to Jump-start, uses SACK.



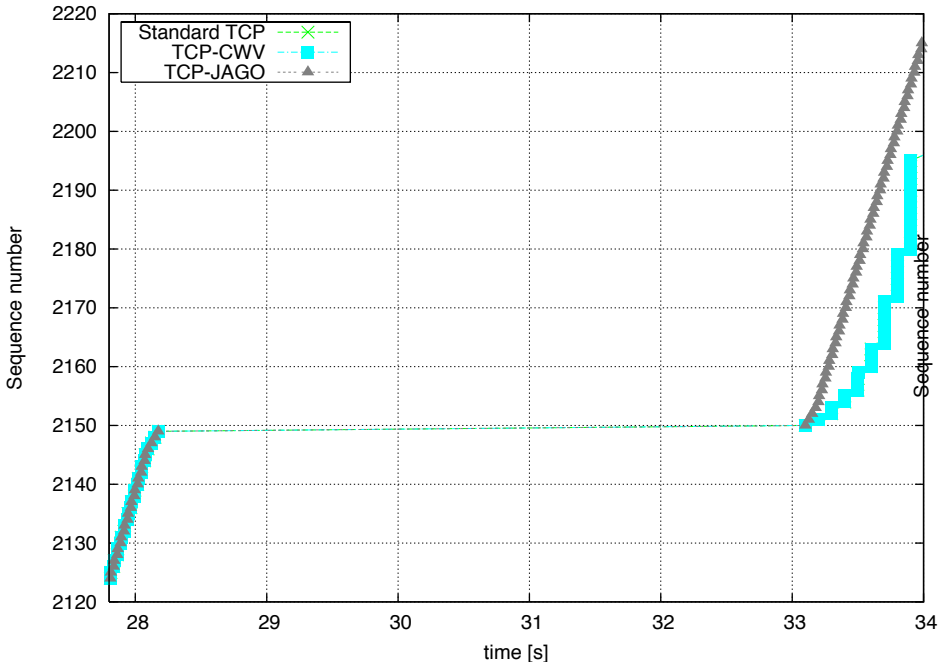
*

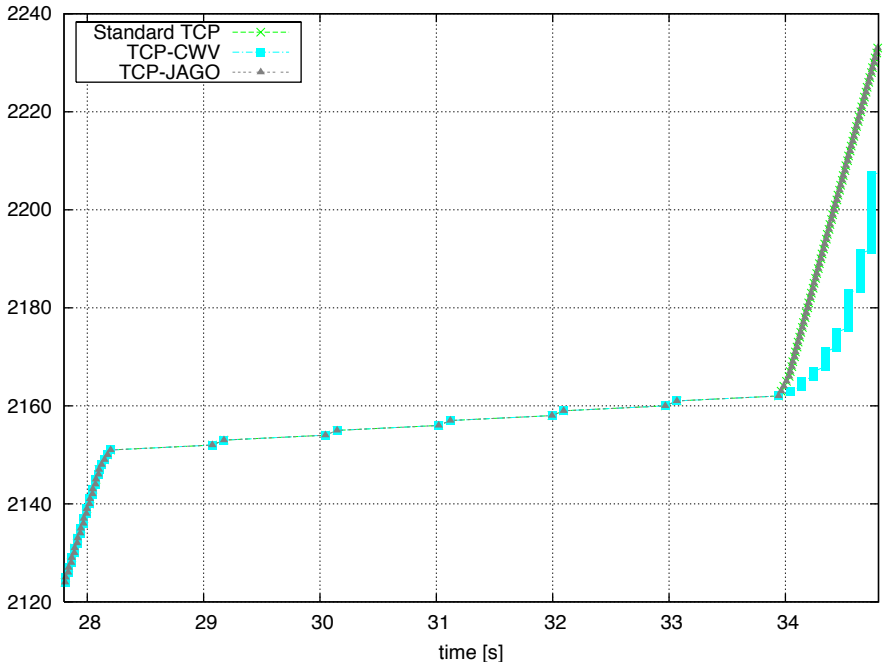
D flight size

R number of packets detected as lost

Benefits for application

Single bottleneck simulation topology, 200 ms path RTT, BDP router buffer, 100 Mbps Bandwidth, sending rate 512Kbps, MinRTO 1 sec





5 sec Idle period

5 sec App-limited period

After idle or an App-limited period this promptly resumes the previous sending rate and benefits application

Electronics Research Group

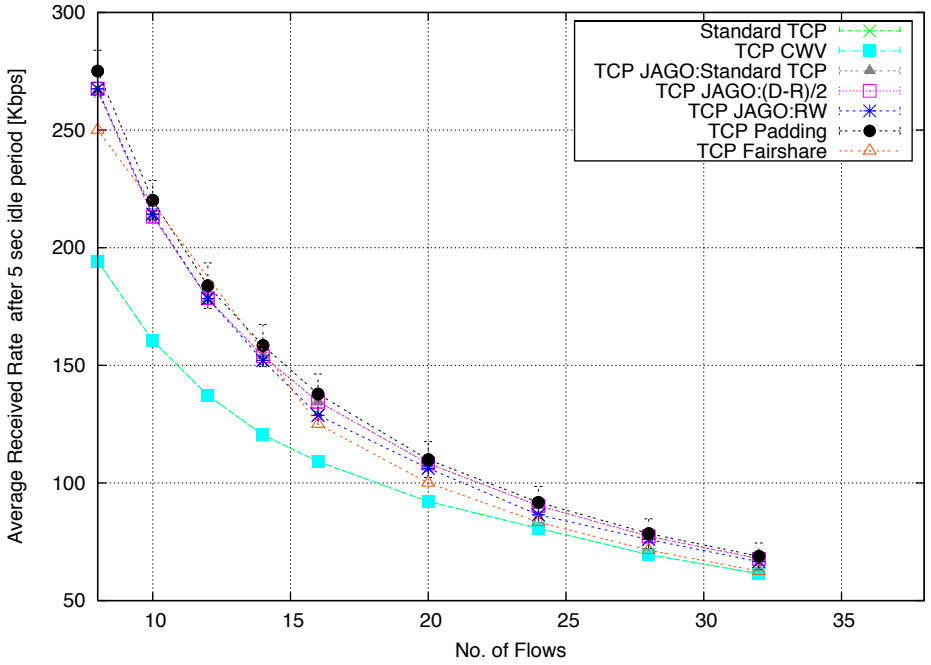
Simulations to explore fair share

Flow monitor: 5 sec Idle period case

200 ms path RTT, BDP router buffer, 100 Mbps bottleneck
Changed bottleneck to 2 Mbps , Flow monitored for 10 RTT

Updated behaviour is only 3% higher than TCP fair share, for heavy congestion (with 16 flows)

Average receive rate of all flows is less than or equal to fair share (less than 0.1% difference).



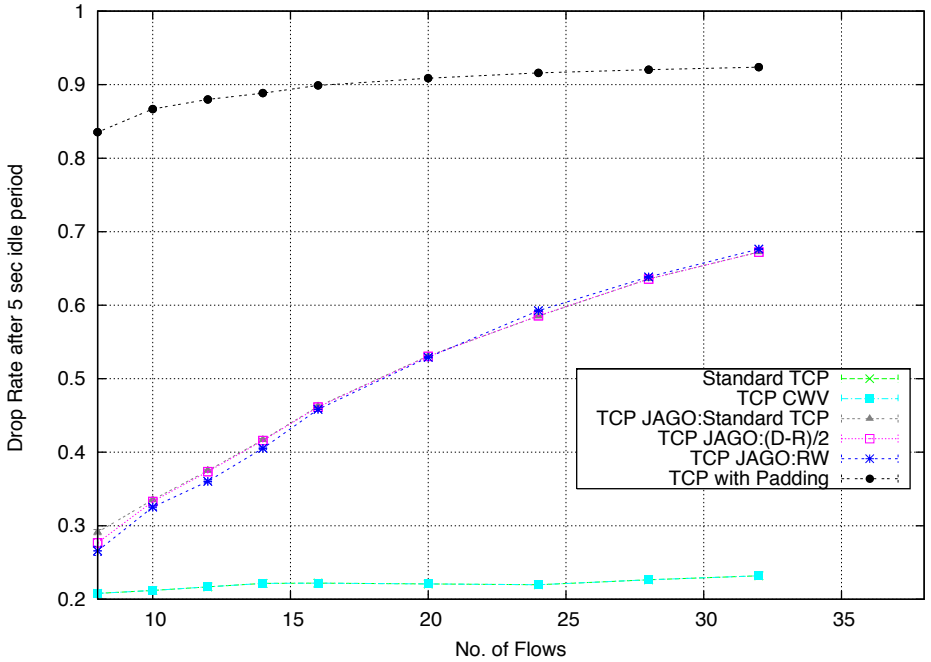
Electronics Research Group

Simulations to explore fair share …

Flow monitor: 5 sec App-limited period case

200 ms path RTT, BDP router buffer, 100 Mbps bottleneck, rate 512Kbps, Changed Bottleneck Bandwidth of 2 Mbps , Flow monitored for 10 RTT, rate during App-limited period 12 kbps

Higher average receive rate and better TCP fair share

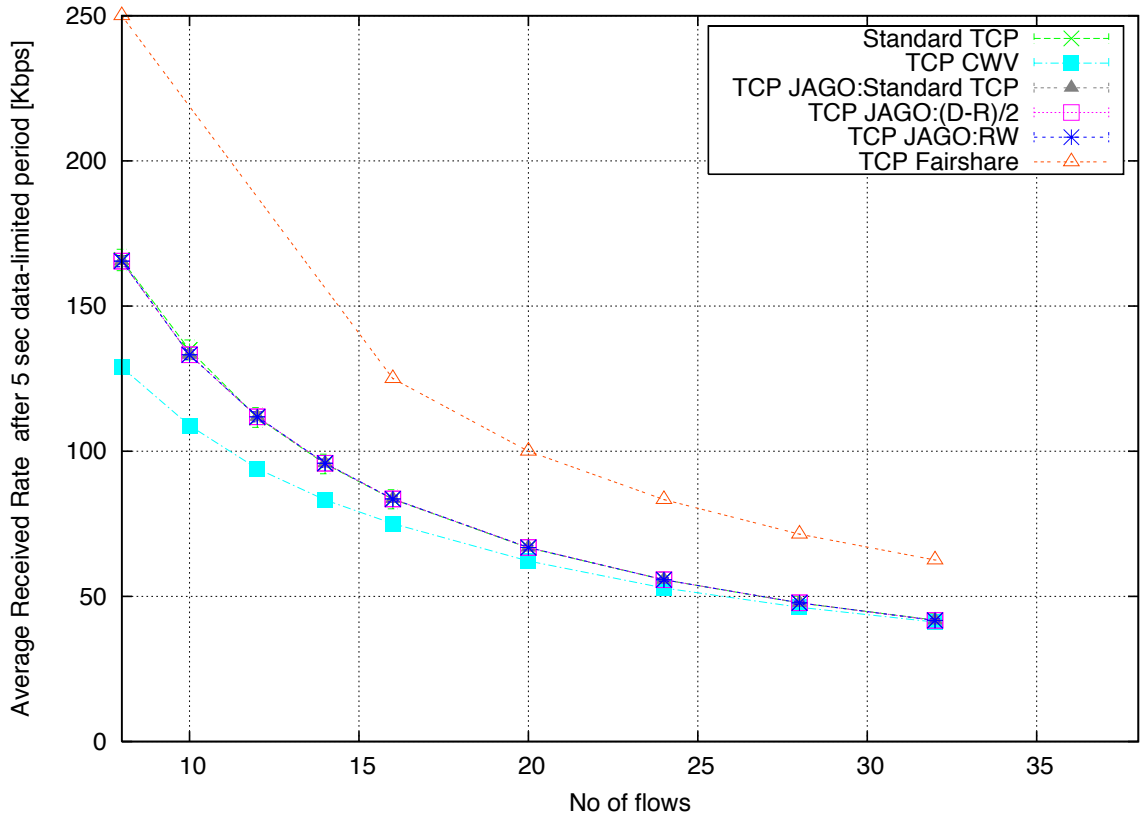


Electronics Research Group

Simulations to explore fair share …

Drop Monitor: 5 sec App-limited period

Offers higher receive rate with fewer packet drops than Standard TCP



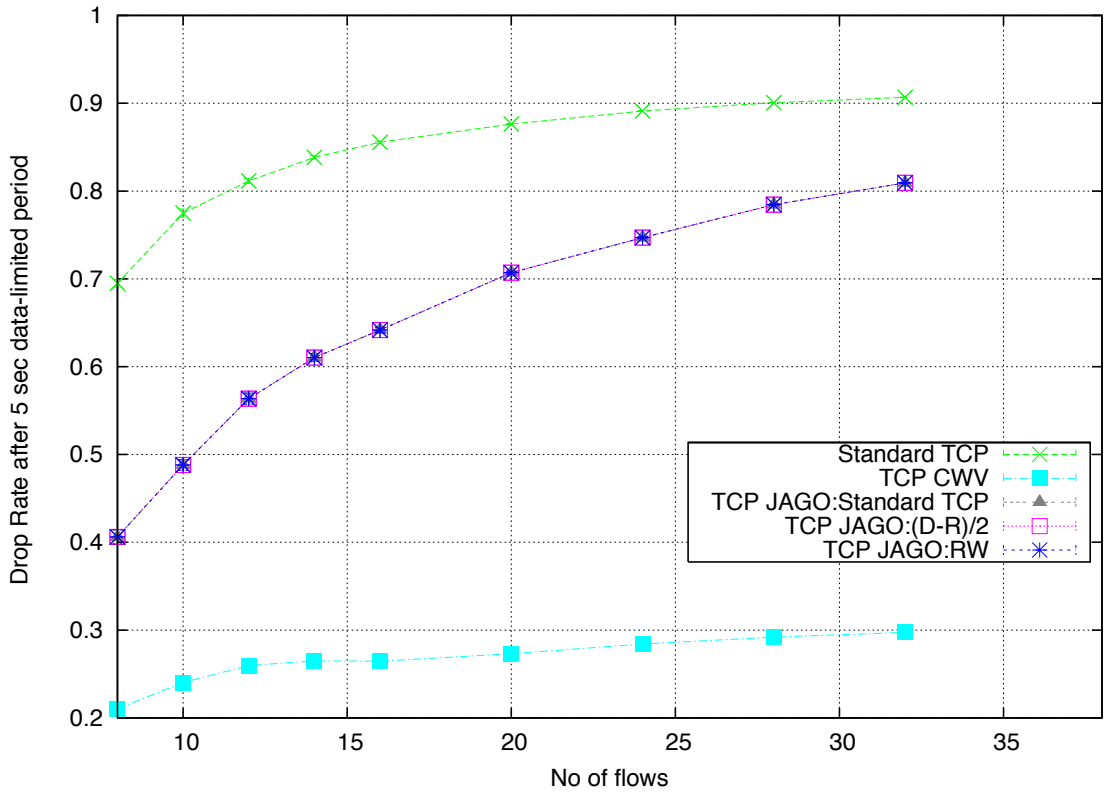
Simulations to explore fair share …

Drop Monitor: 5 sec Idle period

App advantage:

Quickly reduces rate after first RTT.

Drop rate lower than with padding



Electronics Research Group

Larger path RTT (600ms)

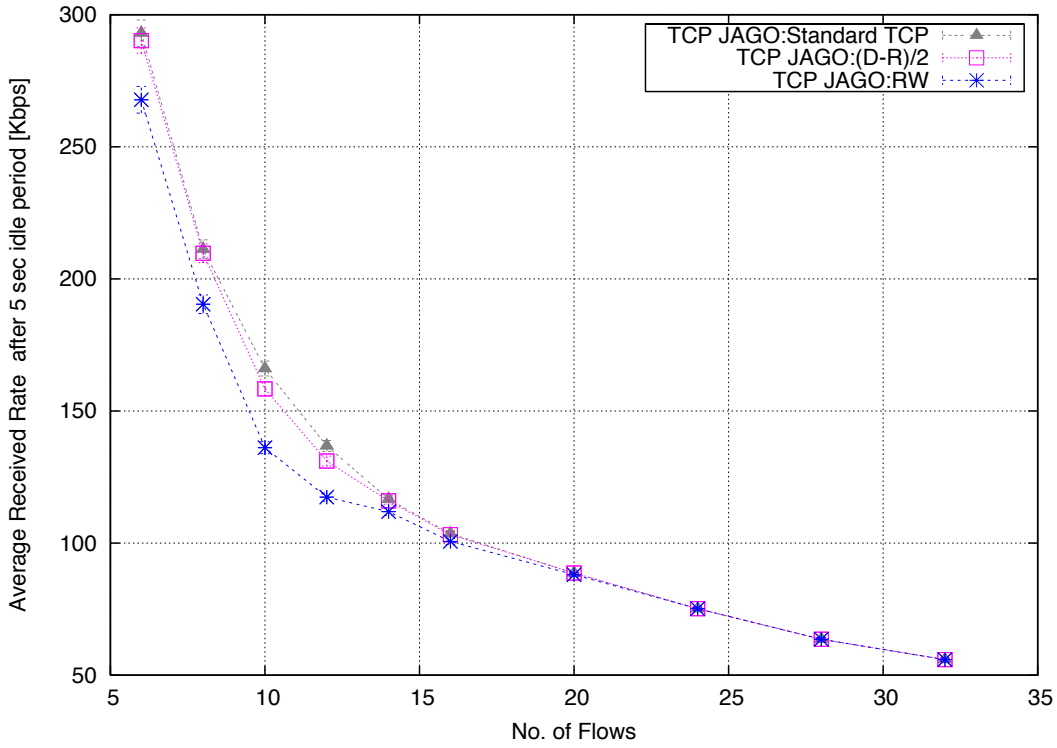
BDP router buffer, 100 Mbps bottleneck, rate 512Kbps, 5 sec idle,
Bottleneck change to 2 Mbps , Flow monitored 10RTT

Longer delay reveals different variants affect application performance

Reducing cwnd to RW, reduces receiver rate

Highest receive rate observed using cwnd/2

Best performance when cwnd reset to (D-R)/2



Electronics Research Group

Conclusions

CWV identified important issues – we agree

BUT, CWV can be both aggressive to network and too conservative for many applications!

- Congestion after idle is a problem for CWV

Our draft proposes a TCP update:

- Benefits variable-rate TCP applications
- Provides “appropriate” response to congestion
- Simulation evidence that this works

Is there interest in taking this further in IETF/IRTF?

- To obsolete CWV
- To replace with a less restrictive method

Extra Slides

Comparison

Approaches	Period	Standard TCP	CWV	Update
Benefit to Applications	Idle period	Reduces cwnd to RW and slow-starts. App does not benefit application	Reduces cwnd by half every RTO. Better for app than TCP.	Does not normally reduce cwnd. Apps benefit
	Apps-limited period	Increases cwnd each ACK. App benefits	Reduces cwnd by half every RTO. Conservative compared to TCP	Does not reduce cwnd. Apps benefit
Benefit to Network	Idle period	Conservative approach. Network benefits	Aggressive compared to TCP	More aggressive than TCP and CWV, converges to a safe cwnd
	Apps-limited period	Aggressive approach	Conservative compared to TCP	More conservative than Standard TCP. More aggressive than CWV, but converges to a safe cwnd