SWINBURNE
UNIVERSITY OF
TECHNOLOGY

FreeBSD-based TCP CC research: "Delay-gradient" TCP Modular congestion control framework

Grenville Armitage
Lawrence Stewart
Centre for Advanced Internet Architectures (CAIA)

Talk overview

- Purpose of this talk
- Delay-gradient TCP
- Modular CC framework in FreeBSD

Why here, why now?

- We've done some work enhancing FreeBSD's TCP stack
  - Supported by Cisco Systems (and later the FreeBSD Foundation)
  - Some code is now in FreeBSD, much remains work in progress
- Hope there's value to the IRTF/IETF community
  - Independent implementations of CUBIC, Vegas, ...
  - Some promising results with *delay-gradient* TCP (published in IFIP Networking 2011 conference, May 2011)
    - But no silver bullets for CC problems
  - Looking for more collaborators, testers

Background: "newtcp" project at CAIA

- FreeBSD implementations of new TCP algorithms
  - Modular congestion control (modCC) kernel framework
  - NewReno, CUBIC, HTCP, "Hamilton Delay" and Vegas implemented in modCC
  - Multipath TCP under development
- Improved tools for capturing behaviour of TCP state machine
  - Statistical Information For TCP Research (SIFTR)
- Improved loss-insensitive, delay-based TCP
  - "CAIA Hamilton Delay" (CHD), code released & in FreeBSD stack
  - "CAIA Delay Gradient" (CDG), code released as FreeBSD patches

Online at http://caia.swin.edu.au/urp/newtcp

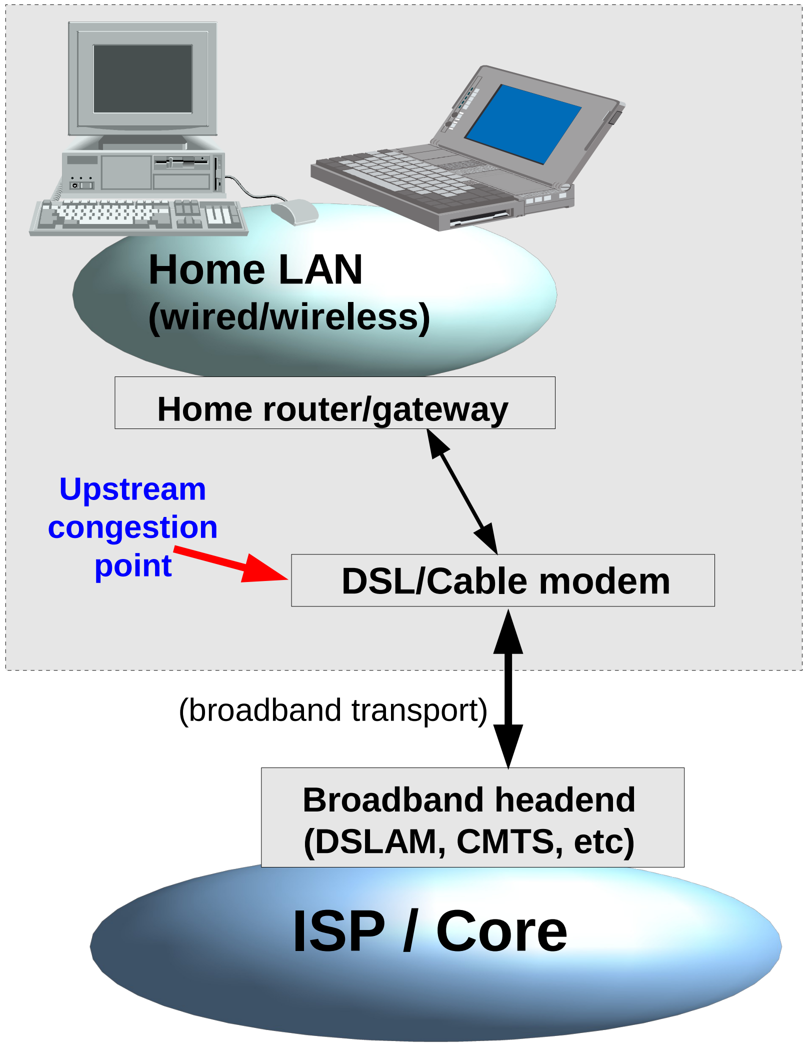
Talk overview

- Purpose of this talk
- Delay-gradient TCP
- Modular CC framework in FreeBSD

Why delay-based TCP?

- We all know letting queues build up is bad
  - collateral damage (induced latency and loss),
  - slows the feedback loop
  - “Buffer bloat”
- Packet loss is not a good congestion indicator over paths containing intrinsically lossy links (e.g. wireless)
- Idea: “delay-based” TCP that senses onset of congestion by variations in round trip time (RTT)
  - keeps queues small, lower induced delays, minimises packet losses
  - Jain, 1989, “Congestion Avoidance using Round-trip Delay”

Induced queuing delay – “Collateral damage”



Interactive flows sharing a congestion point with TCP flows experience cyclical “collateral damage”

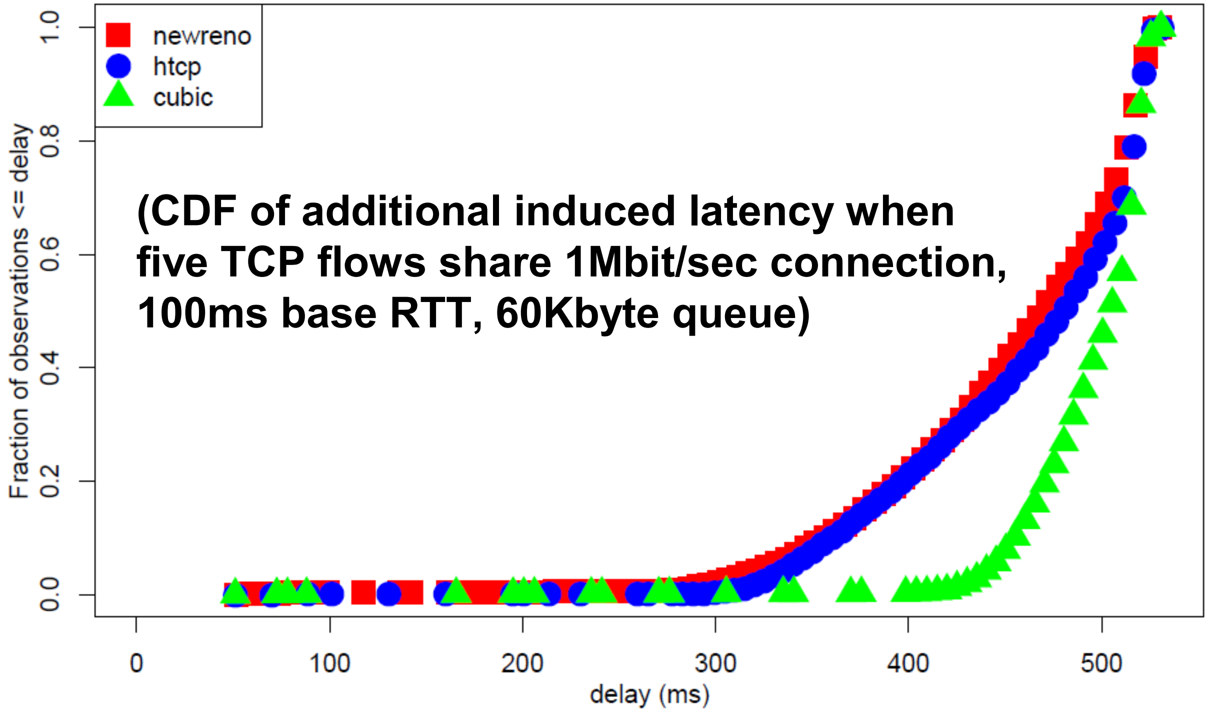
Certain newer CC algorithms are worse than NewReno

CUBIC – new TCP algorithm for high speed performance, default in Linux

NewReno – traditional TCP algorithm

HTCP – Hamilton Institute's experimental TCP algorithm for high speed performance

(there's also Compound TCP for Win7, etc....)



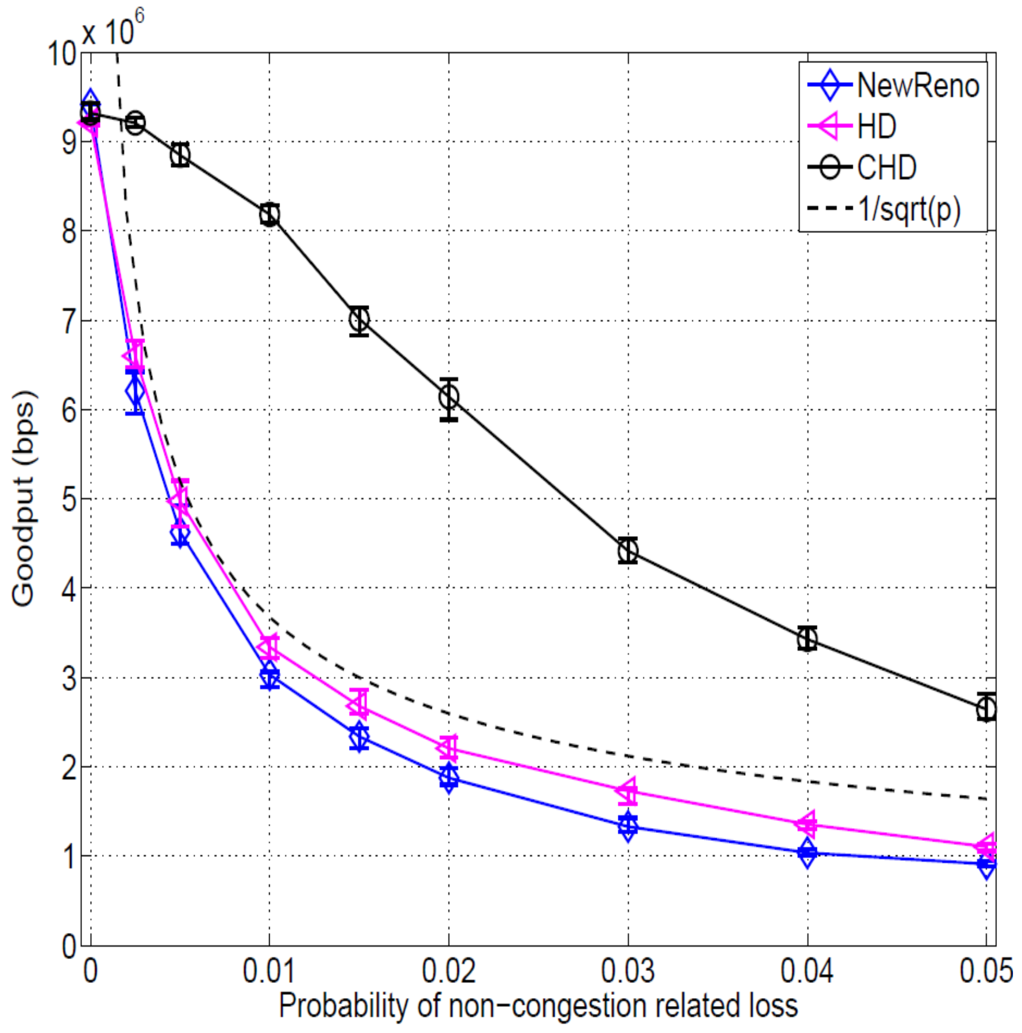
(CDF of additional induced latency when five TCP flows share 1Mbit/sec connection, 100ms base RTT, 60Kbyte queue)

Our first foray – “delay threshold” TCP

- Variant of Hamilton Institute's delay-based algorithm, with improved loss-tolerance (*CHD*)

  D.Hayes, G.Armitage, "Improved Coexistence and Loss Tolerance for Delay Based TCP Congestion Control," 35th Annual IEEE Conference on Local Computer Networks (LCN) **October 2010**

- However, limitations:
  - ☐ Requires knowledge of “baseRTT” (minimum RTT along path)
  - ☐ Requires knowledge of specific RTT 'thresholds' to control probabilistic backoff – hard without prior knowledge of each path

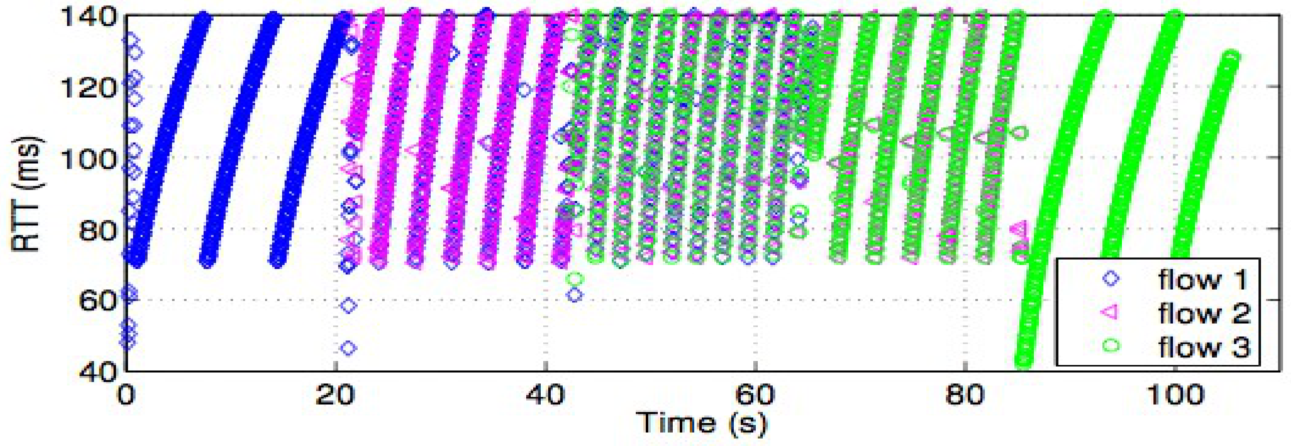


Delay-*gradient* TCP: Less configuration

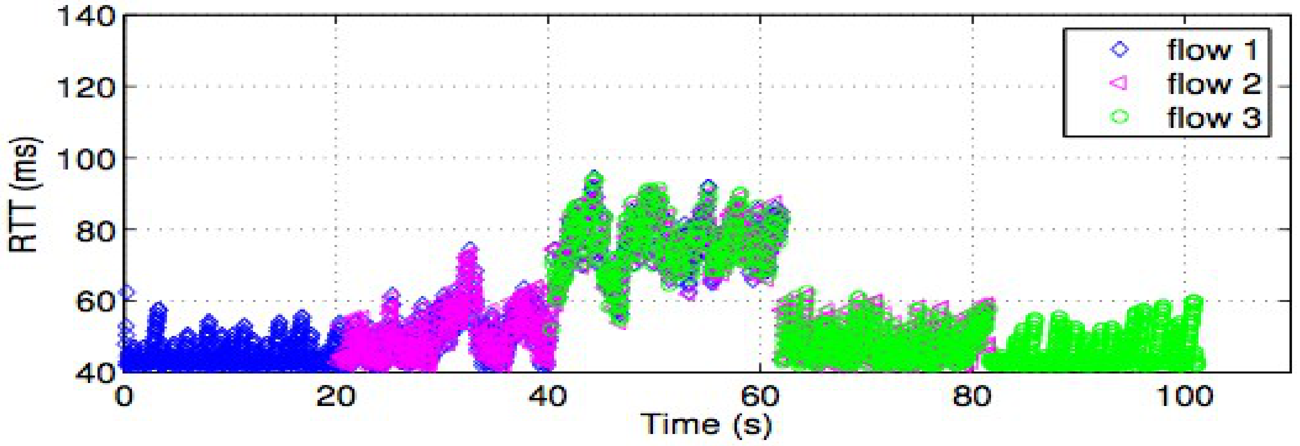
- "CAIA Delay-Gradient" – CDG (no budget for good acronyms)
  - D. Hayes, G. Armitage, "Revisiting TCP Congestion Control using Delay Gradients," IFIP/TC6 NETWORKING 2011 , Valencia, Spain, 9-13 May 2011
- CDG pursues the following goals:
  - Eliminate need for path-specific absolute RTT thresholds and accurate baseRTT estimation
  - Infer queue full and empty states → improve tolerance to packet loss
  - Co-exist with loss-based TCPs to some degree

CDG induces far less queuing delay

- Three 60sec flows through one queue
- Staggered starts, one flow every 20sec
- NewReno flows cycle queue, force packet losses
- CDG flows keep queue closer to empty throughout, "noisy" rather than cyclical queuing delays

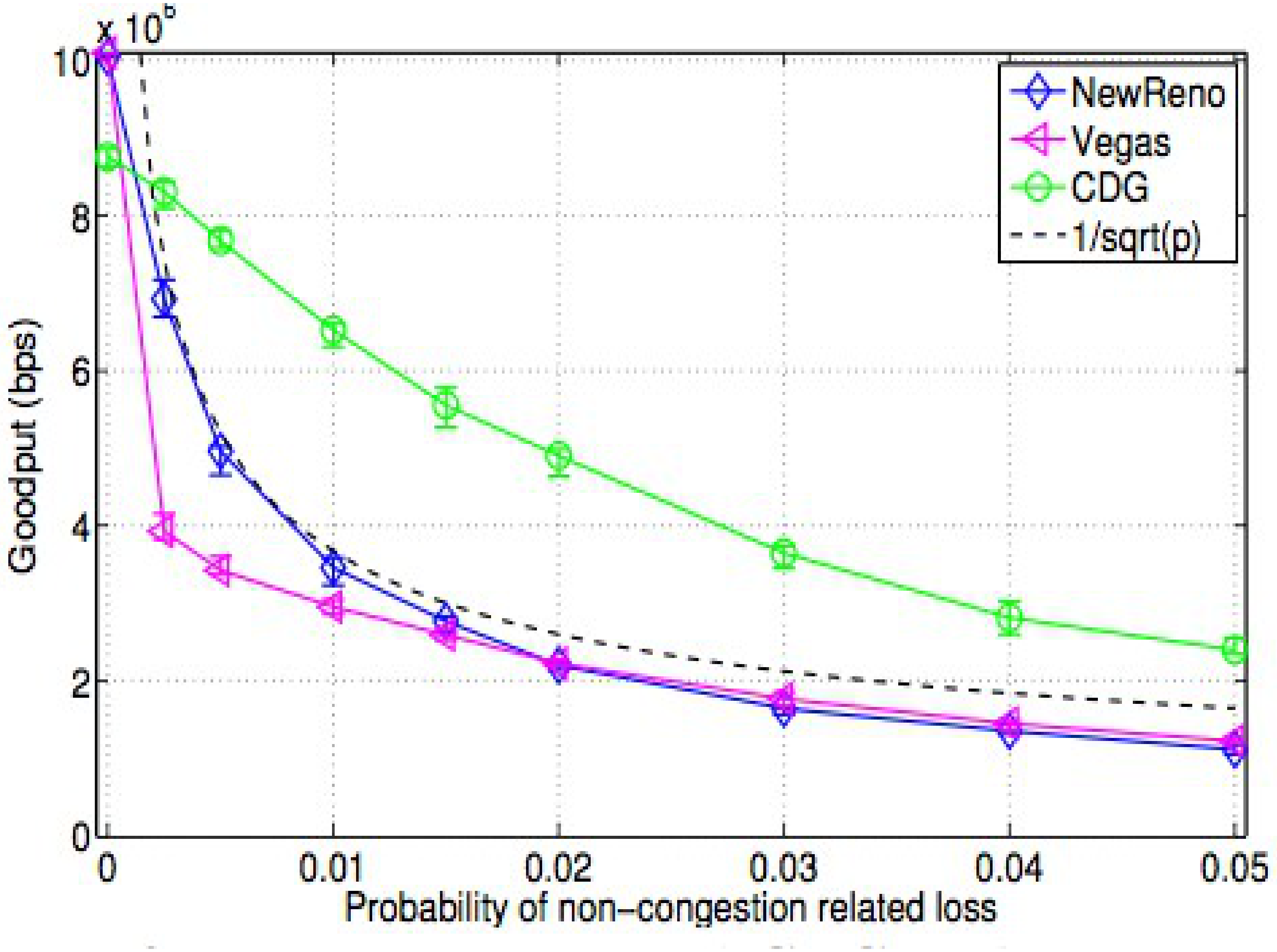


(a) NewReno RTT dynamics due to induced queuing delays



(b) CDG RTT dynamics due to induced queuing delays

CDG tolerant of intrinsic packet losses



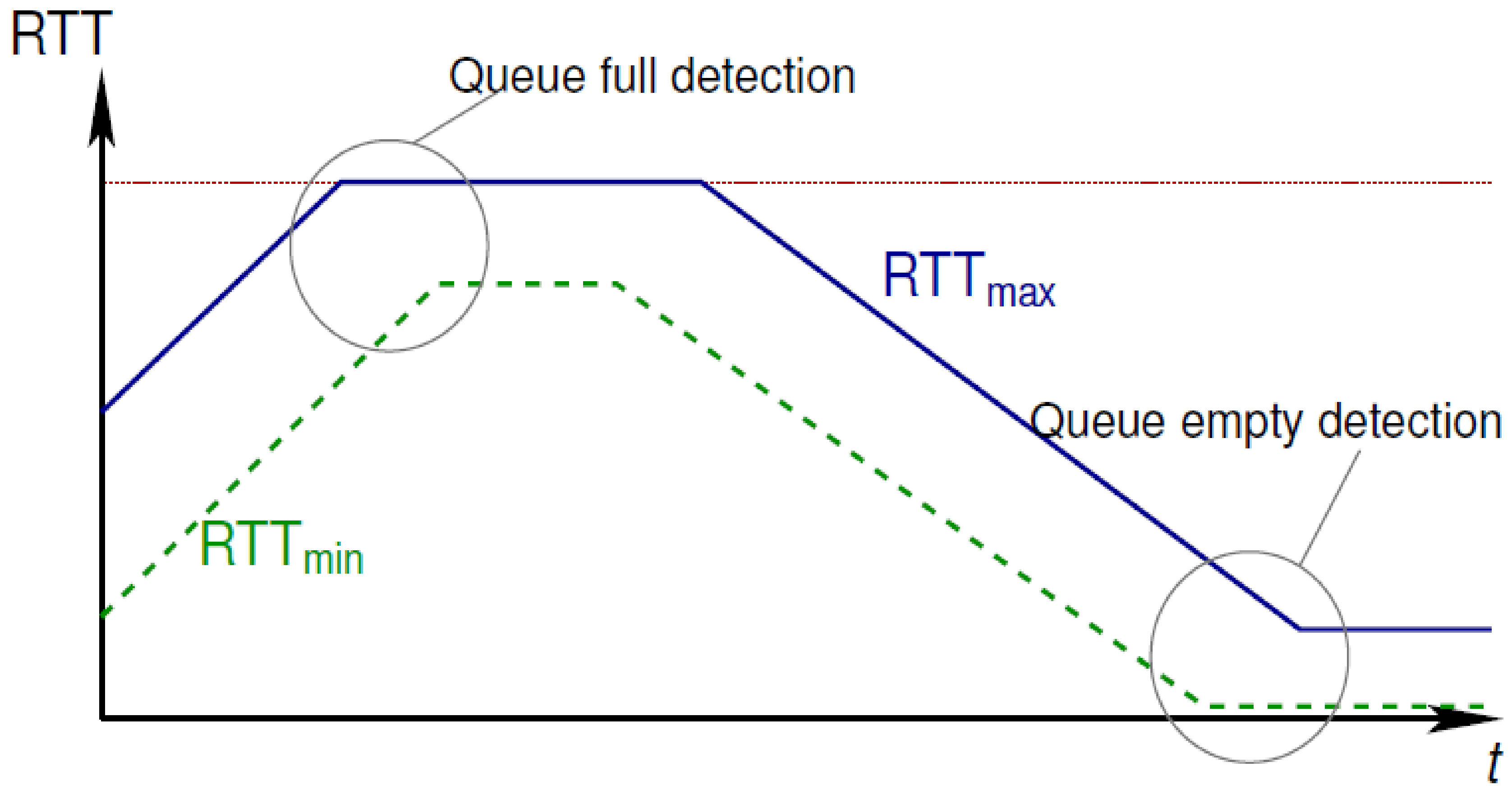
Delay gradient as a congestion signal

- RTT is a noisy signal
- We track RTT_{min} and RTT_{max} in a measured interval (1 RTT)

$$g_{\min,n} = \mathrm{RTT}_{\min,n} - \mathrm{RTT}_{\min,n-1}$$
$$g_{\max,n} = \mathrm{RTT}_{\max,n} - \mathrm{RTT}_{\max,n-1}$$

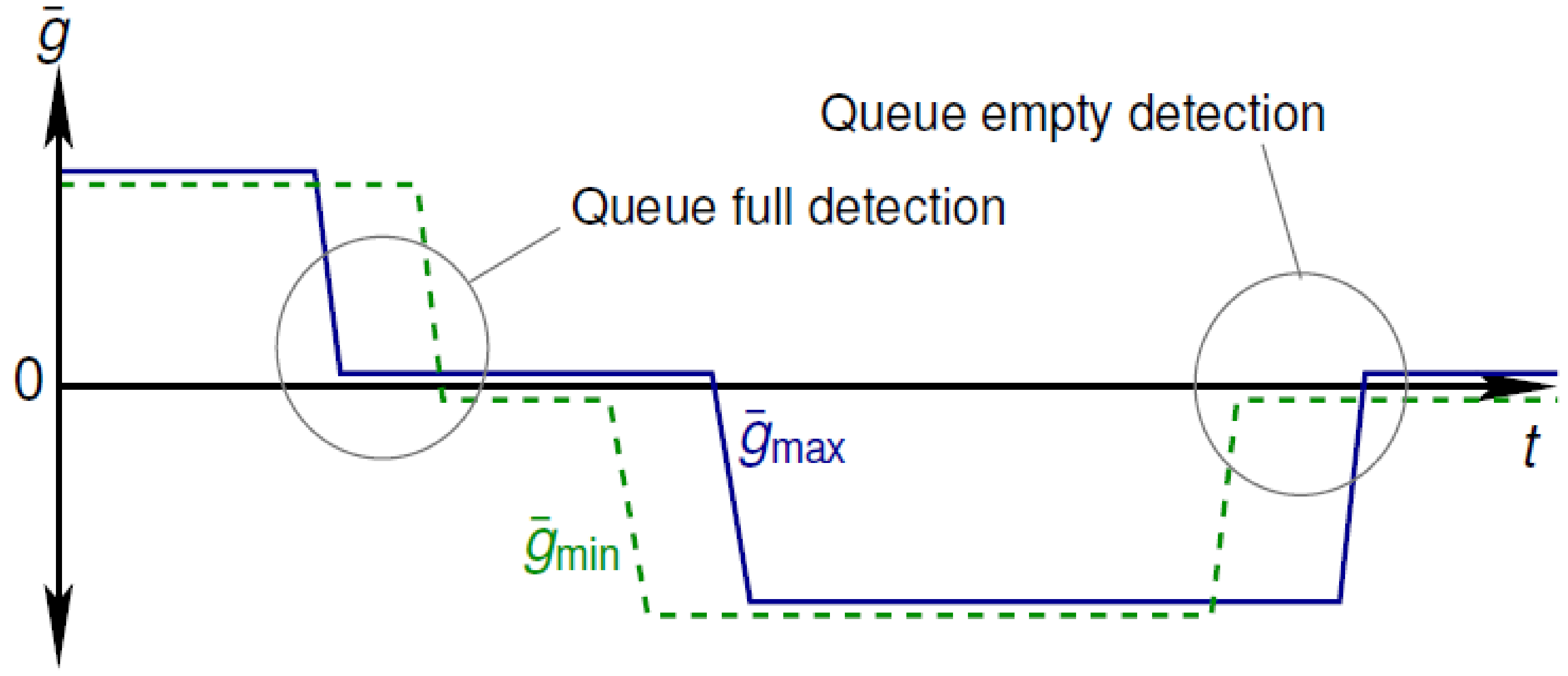
- Smoothed
  - moving average (configurable)
  - probabilistic back-off
- Path Queue State
  - Q is either {rising, falling, full, empty, unknown}.

Inferring queue states



Idealised RTT versus time

Inferring queue states



Idealised gradient versus time

- Packet loss now *only* indicates congestion if we think queue was full at the time

Window Progression when no packet loss

Decision made once per RTT:

$$w_{n+1} = \begin{cases} w_n\beta & X < P[\text{backoff}] \wedge \bar{g}_n > 0 \\ w_n + 1 & \text{otherwise} \end{cases}$$

- If gradient is $+$'ve AND random backoff
- Reduce w, with $\beta = 0.7$
- Otherwise increment window (one MSS of data)

Probabilistic 'back-off'

Why?

- Helps avoid synchronisation issues
- Helps smooth the noisy gradient signal.

Exponential?

- Decisions are made once per RTT
- $P[\text{backoff}] = 1 - e^{-(\overline{g}_n/G)}$
  - where $\overline{g}_n = \overline{g}_{\text{min},n}$, or $\overline{g}_{\text{max},n}$ if $\overline{g}_{\text{min},n} \leq 0$
  - G is a scaling parameter
- $\overline{P_{\text{RTT}}[\text{back-off}]} = \overline{P_{\text{RTT}}[\text{back-off}]}$

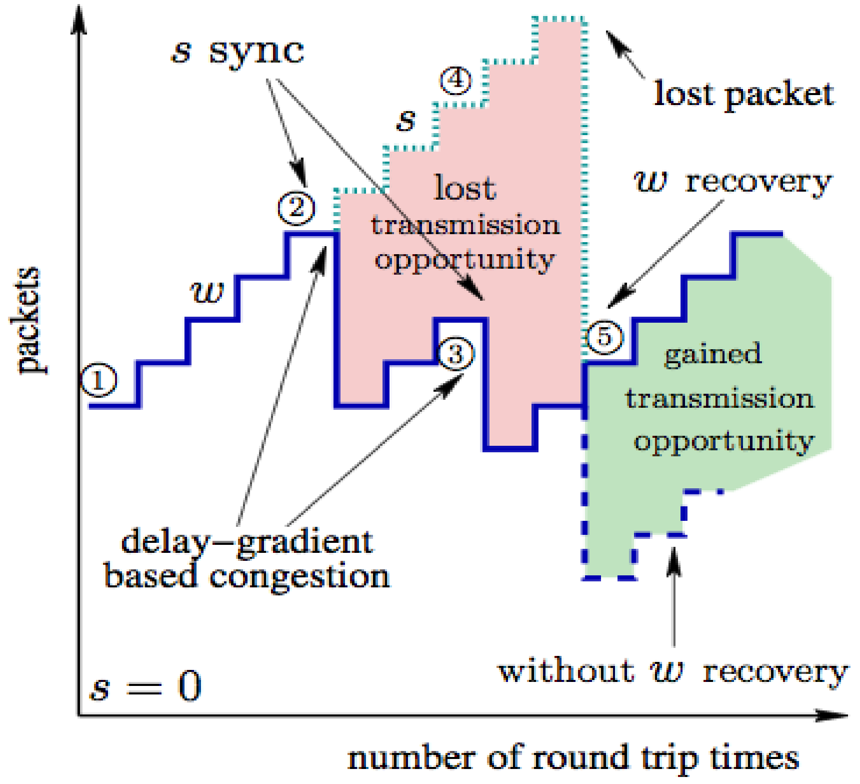
  Note: TCP's additive increase rate will still be RTT dependent.

Competing with loss-based flows #1

- CDG utilises CHD's "shadow window" to limit the loss of capacity relative to NewReno-like loss-based flows

Referring to the regions indicated by circled numbers:

1. w grows as normal for TCP congestion avoidance (one packet per RTT)
2. Delay-gradient congestion indication meeting Equation 6's criteria, s is initialised to w, then $w = \beta w$
3. w continues to react to delay-gradient congestion indications
4. s shadows NewReno-like growth
5. A packet loss occurs (Q = full), so w is set to $s/2$ rather than $w/2$ (per Equation 8)



s sync
lost packet
s
lost transmission opportunity
w recovery
w
gained transmission opportunity
delay–gradient based congestion
without w recovery
s = 0
packets
number of round trip times
1
2
3
4
5

Fig. 2: Behaviour of shadow window (s) and congestion window (w) when competing with loss-based TCP flows.

D.Hayes, G.Armitage, "Improved Coexistence and Loss Tolerance for Delay Based TCP Congestion Control," 35th Annual IEEE Conference on Local Computer Networks (LCN) **October 2010**

Competing with loss based flows #2

- "Ineffectual backoffs"

- If we back off b times and $\bar{g}$ remains positive,
  - Ignore the next b' congestion indications while $\bar{g}$ remains positive

An aside: our testbed

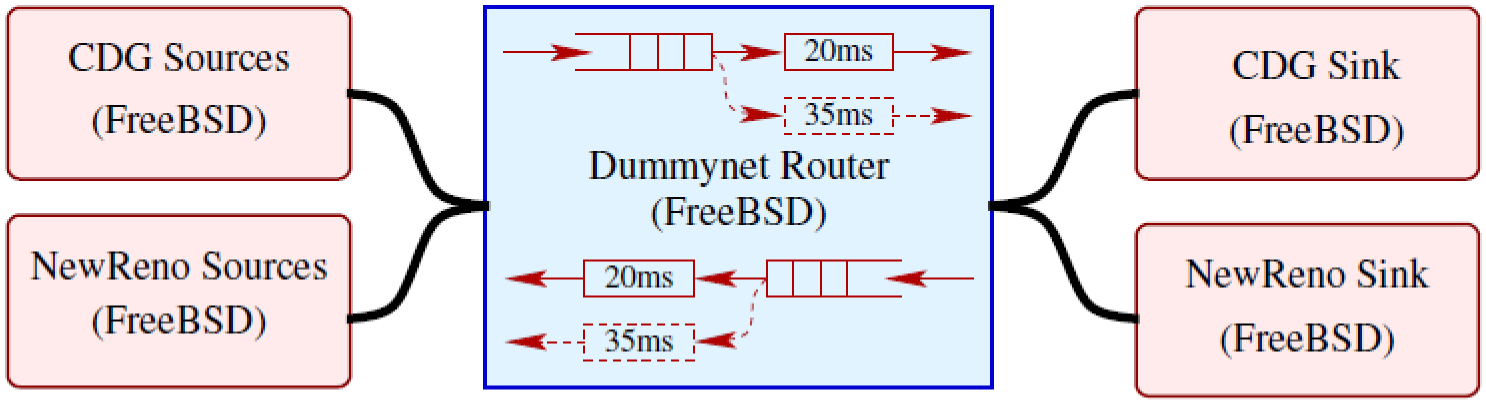
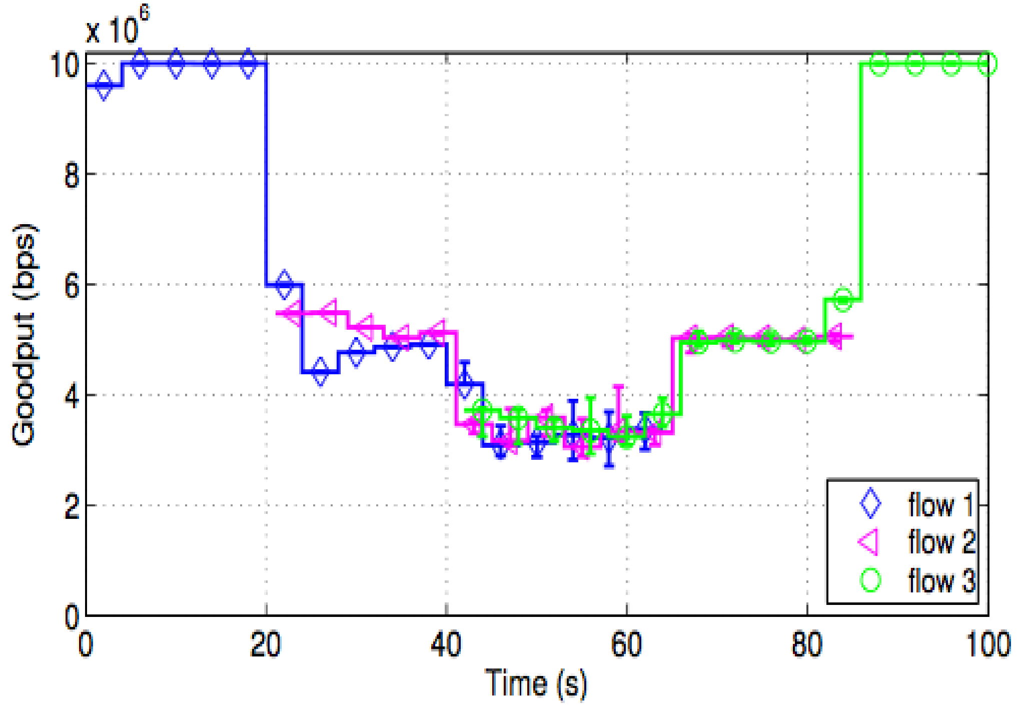


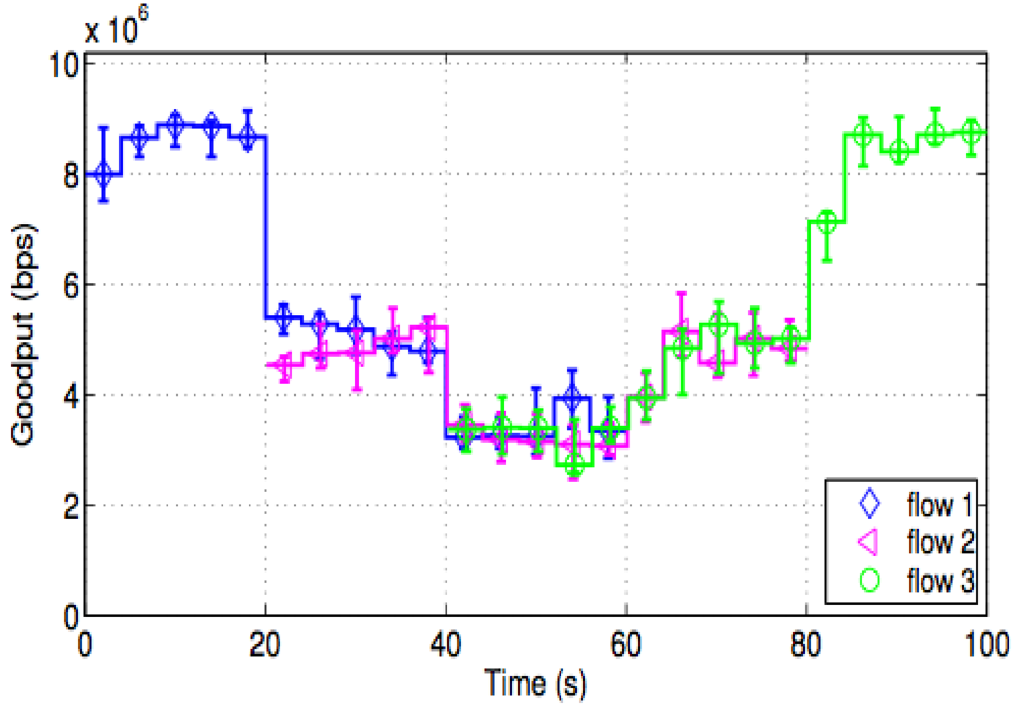
Fig. 3: Experimental Testbed

Sources: Modified FreeBSD 9.0 hosts
Sinks: Unmodified FreeBSD hosts
Links: 1Gbps ethernet
Dummynet: rate limit of 10Mbps,
queue of 84 packets (~100ms w/1500byte packets)

CDG achieves a reasonable goodput



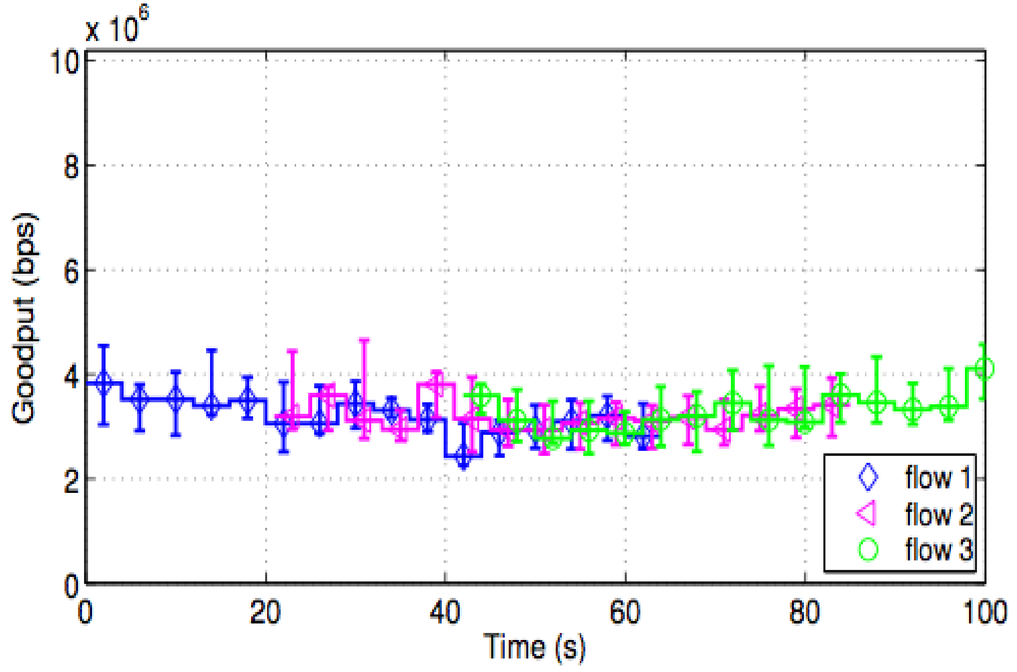
(a) Three NewReno flows sharing link



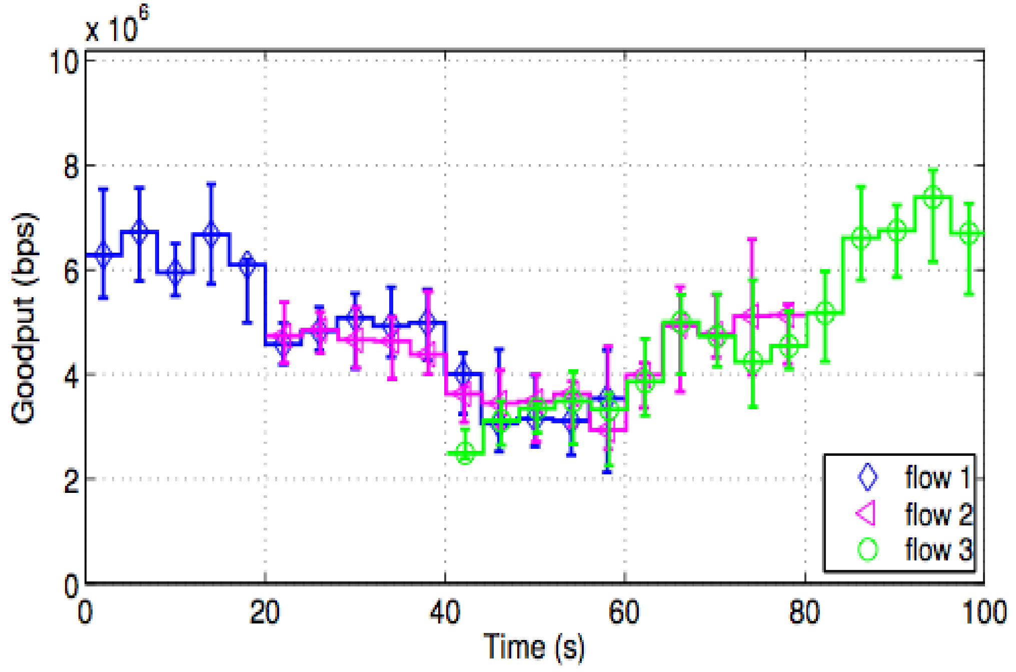
(b) Three CDG flows sharing link

(trials run through common 10Mbit/sec congestion point with no non-congestion losses)

CDG achieves better goodput with loss



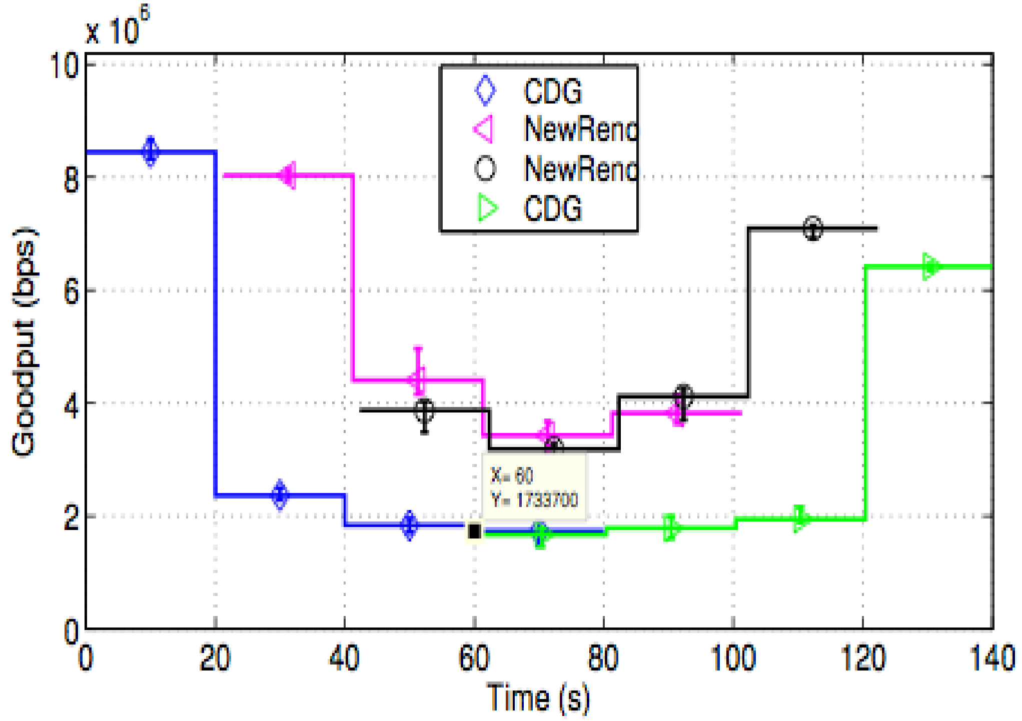
(a) Three NewReno flows sharing link



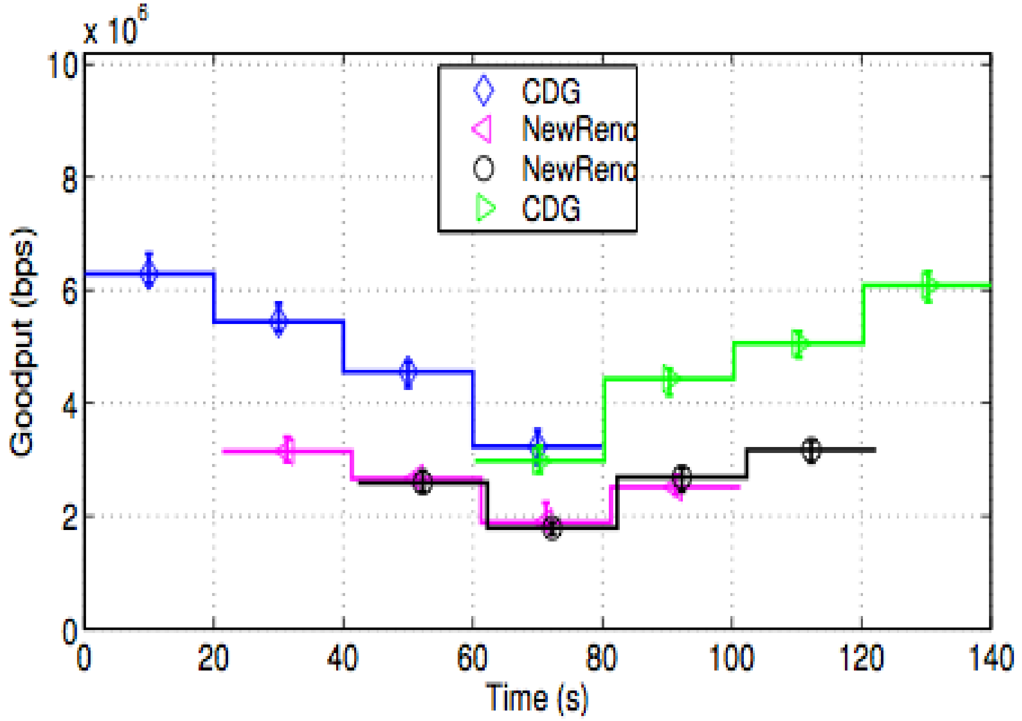
(b) Three CDG flows sharing link

(trials run through common 10Mbit/sec congestion point with 1% non-congestion losses)

Sharing between CDG and NewReno



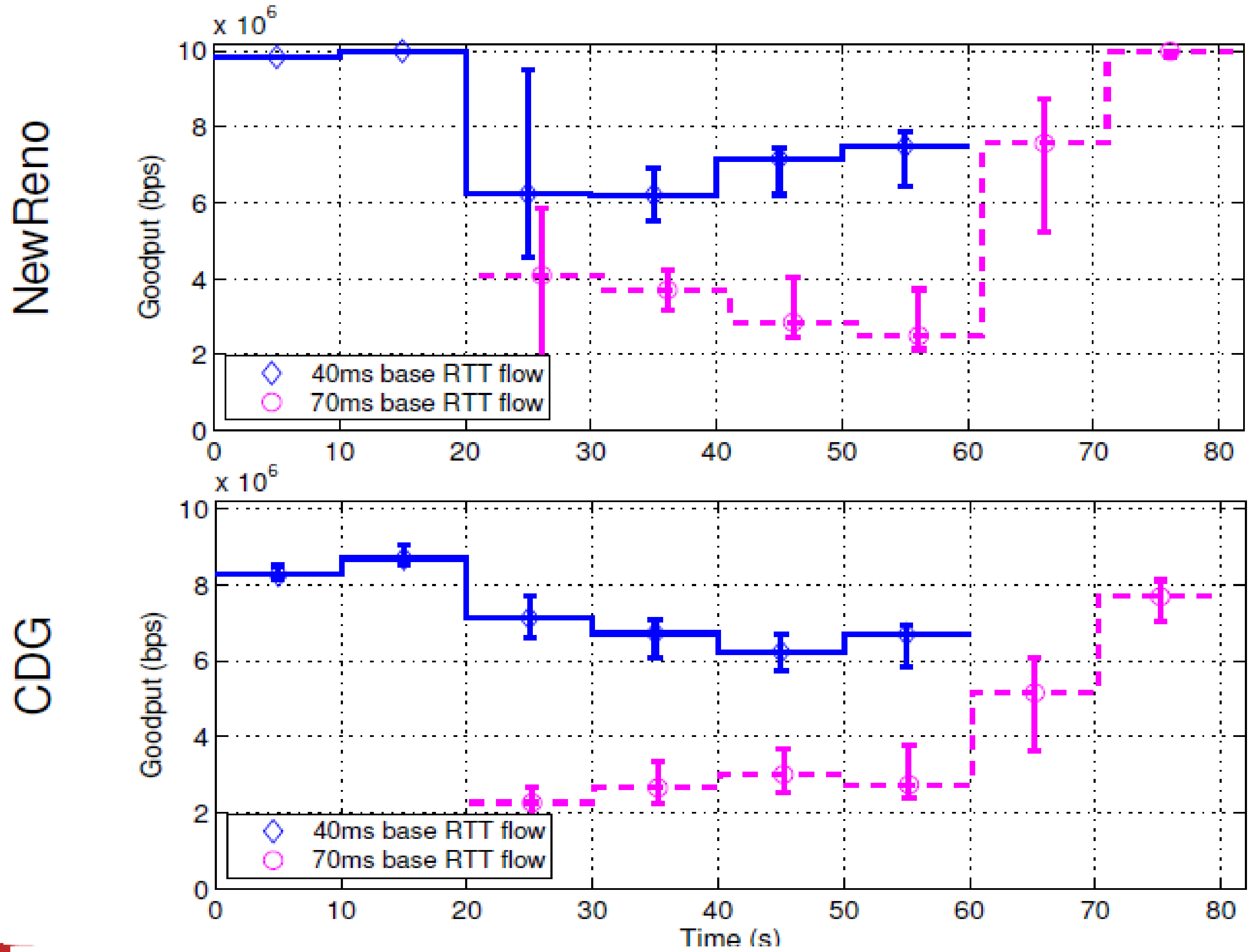
(a) No non-congestion losses.



(b) 1% random non-congestion loss.

(co-existence with and without non-congestion losses)

Sharing between flows with different RTT



Conclusions (regarding CDG)

- Using delay-*gradient* as congestion indication:
  - ☐ Avoids the need for an accurate estimate of baseRTT
  - ☐ Can be used to help tolerate non-congestion losses
  - ☐ Can coexist *somewhat* with loss-based TCP
  - ☐ Lots of future work on hybrid congestion control algorithms, evaluating CDG's detailed dynamic characteristics, ...
  - ☐ Implemented → FreeBSD kernel module code available online

Talk overview

- Purpose of this talk
- Delay-gradient TCP
- Modular CC framework in FreeBSD

FreeBSD As A R&D Platform

- Currently available in FreeBSD 8.3+, 9.0+
  - Modular congestion control framework: mod_cc(4), mod_cc(9)
  - Loss-based modules: cc_newreno(4), cc_htcp(4), cc_cubic(4)
  - Delay-based modules: cc_vegas(4), cc_hd(4), cc_chd(4)
  - Helper frameworks for extending stack functionality at runtime: khelp(9), hhook(9)
  - Enhanced RTT estimator Khelp module: h_ertt(4)
  - Event-based TCP data logging: siftr(4)

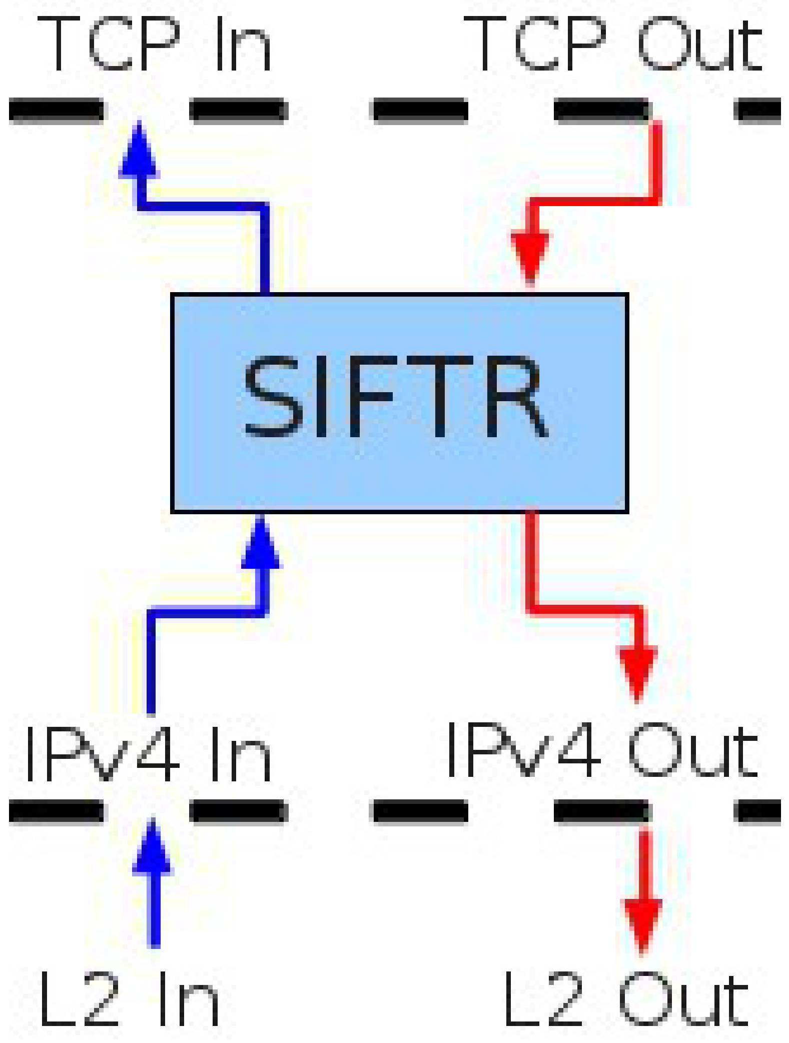
FreeBSD As A R&D Platform

- Available out-of-tree at http://caia.swin.edu.au/urp/newtcp
  - Delay based modules: cc_cdg
  - Assorted patches & scripts: SIFTR analysis, Iperf socket buffer management, deterministic packet discard, ...
- Coming soon
  - Research-focused multipath TCP implementation
  - Datacenter focused incast congestion control modules
  - NS-3 FreeBSD 9.x stack port

Modular CC

- Module consists of name + function pointers
  - mod_init, mod_destroy, cb_init, cb_destroy, conn_init, ack_received, cong_signal, post_recovery, after_idle
- Per-socket and can be changed at any time
- Can reuse code from other modules where useful
  - e.g. NewReno, Vegas, HTCP in ~ 88, 170, 200 LOC
- Introduces “congestion recovery” stack state to separate loss recovery from congestion recovery

SIFTR



```
root# man siftr
root# kldload siftr
root# sysctl net.inet.siftr.enabled=1
root# fetch http://caia.swin.edu.au
root# sysctl net.inet.siftr.enabled=0
root# cat /var/log/siftr.log | awk -F,
'BEGIN { print "direction,cwnd" }
{ print $1","$9 }'
    direction,cwnd
    o,1073725440
    i,1073725440
    o,4380
    o,4380
    ...
```

Khelp/Hhook

- Kernel Helpers Framework
  - Structured method for runtime kernel extension
  - Tightly coupled with hhook(9)
- Helper Hook framework
  - Arbitrary hook points in kernel code
  - Only executed if >1 khelp module registered for hook
  - Provides call site specific context to khelp modules
- First significant consumer: h_ertt(4)
  - Computes accurate & timely RTT estimate for CC
  - Hooks tcp_input()/tcp_output(), associates data with TCP control block

Links

- CDG
  - Networking 2011 paper: http://caia.swin.edu.au/cv/dahayes/content/networking2011-cdg-preprint.pdf
  - CDG related tech report: http://caia.swin.edu.au/reports/110729A/CAIA-TR-110729A.pdf
  - Part of NewTCP: http://caia.swin.edu.au/urp/newtcp/
  - Module patch: http://caia.swin.edu.au/urp/newtcp/tools.html
- FreeBSD-specific
  - CC code: http://svnweb.freebsd.org/base/head/sys/netinet/cc/
  - HTTP man pages: http://www.freebsd.org/cgi/man.cgi