MulTFRC: TFRC with weighted fairness

draft-welzl-multfrc-00

Michael Welzl, Dragana Damjanovic

75th IETF Meeting
Stockholm, Sweden
29 July 2009

Motivation

- TCP-friendliness has been criticized a lot
  - ICCRG has a design team now
  - Non-standard TCP variants are used
- Multimedia applications should use a reasonable (ideally IETF-approved) congestion control mechanism
  - Need to make this attractive
- Our suggestion: *N*-TCP-friendliness
  - Multiple TCPs do a better job; multiple flows are already used in practice for this reason
  - We already know that they don't cause much harm

What is MulTFRC?

- Like MulTCP: a protocol that is *N*-TCP-friendly
  - $N \in R^+$
  - Larger range of possible values for *N* than for others, e.g. MulTCP and CP
  - Yields flexible weighted fairness (e.g. priorities between users, or between flows of a single user)

- Based on TFRC
  - Easy to implement as an extension of TFRC code
  - Change the equation + measure “real” packet loss

The new equation

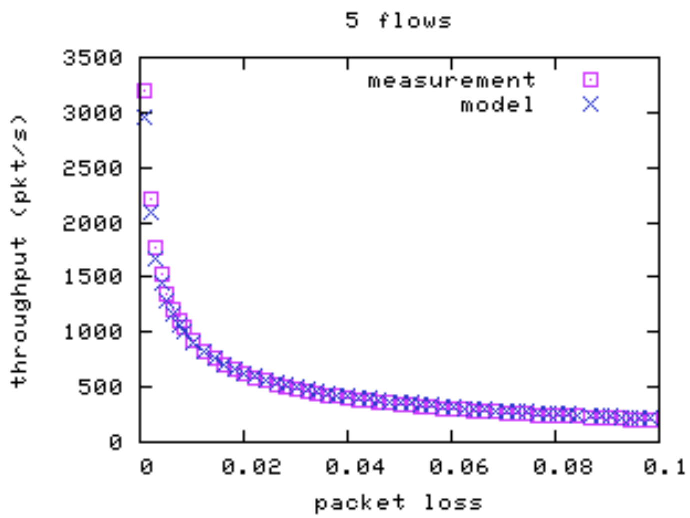
- Dragana Damjanovic, Michael Welzl, Miklos Telek, Werner Heiss: "Extending the TCP Steady-State Throughput Equation for Parallel TCP Flows", University of Innsbruck, Institute of Computer Science, DPS NSG Technical Report 2, August 2008.
  - available from http://heim.ifi.uio.no/michawe/research/projects/multfrc/index.html
  - also SIGCOMM'07 poster, 2-page text available from the same page

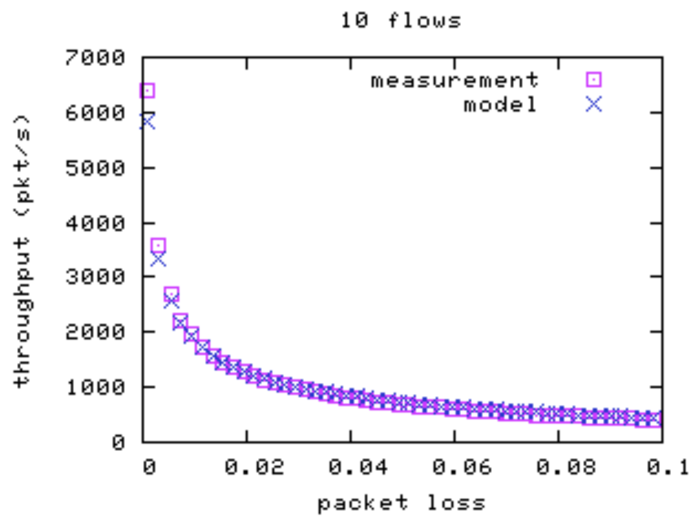
- Most readable in algorithm form
  - **n** - number of flows
  - **b** – no. of packets ACKed by one ACK
  - **RTT** - round-trip time
  - **T** - initial period of time (in TO phase) after which the sender retransmits unacknowledged packets
  - **p_e** - loss event probability of the cumulative flow
  - **p_r** - probability that a packet is lost

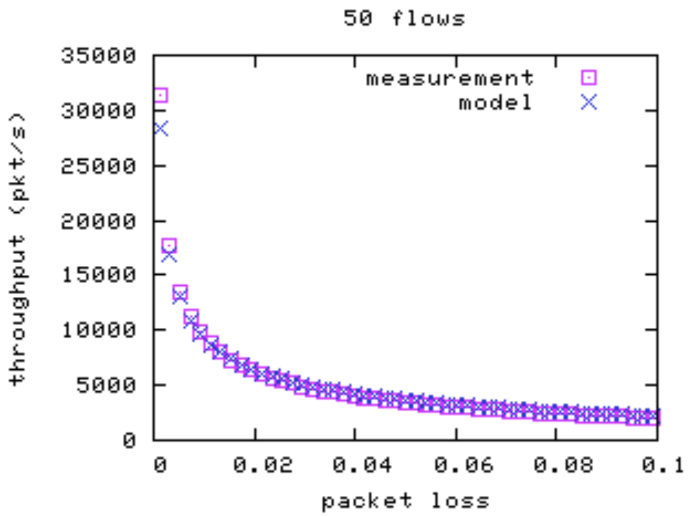
```
Require: n, p_e, p_r, b, RTT, T.
Ensure: The throughput B.
  j = p_r/p_e
  j1 = j
  if j1 > n then
    j1 = n
  end if
  a = sqrt(p_e*b*j1*(24*n*n+p_e*b*j1*(n*n-4*n*j1+4*j1*j1)))
  x = (j1 * p_e * b * (2 * j1 - n) + a)/(6 * n * n * p_e)
  w = n * x/(2 * b)(1 + 3 * n/j1)
  z = T * (1 + 32 * p_e * p_e)/(1 - p_e)
  q1 = j * n/w
  if q1 > 1 then
    q1 = 1
  end if
  if q1 * z/(x * RTT) >= n then
    q = n
  else
    q = q1 * z/(x * RTT)
  end if
  return (1 - q/n)/(p_e * x * RTT) + q/(z * (1 - p_e))
```

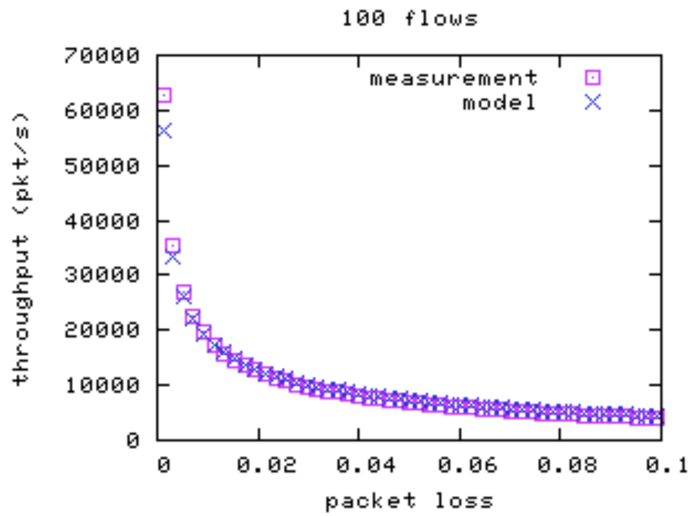
Equation validation

(ns-2 simulations; we also did real-life tests)





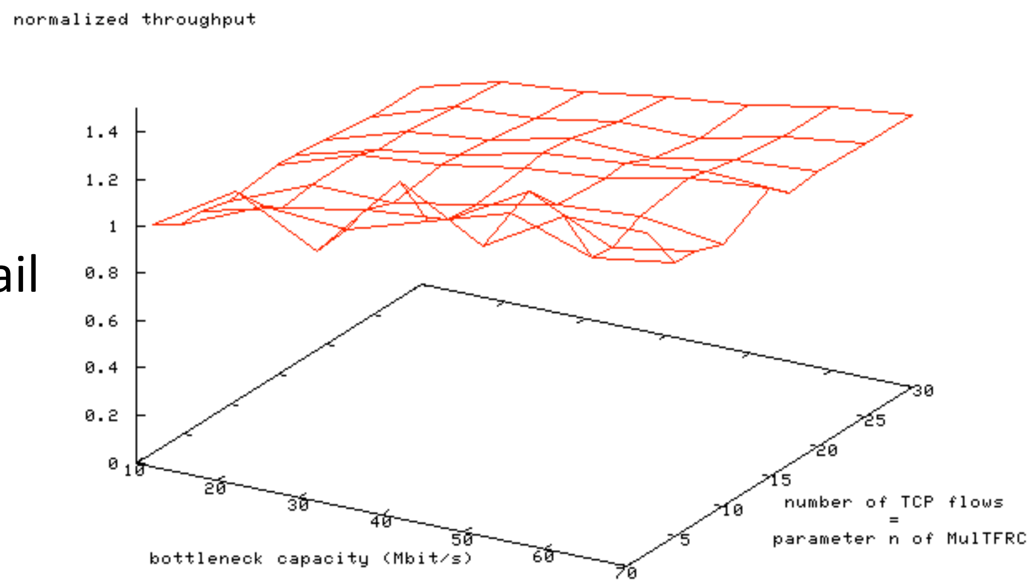


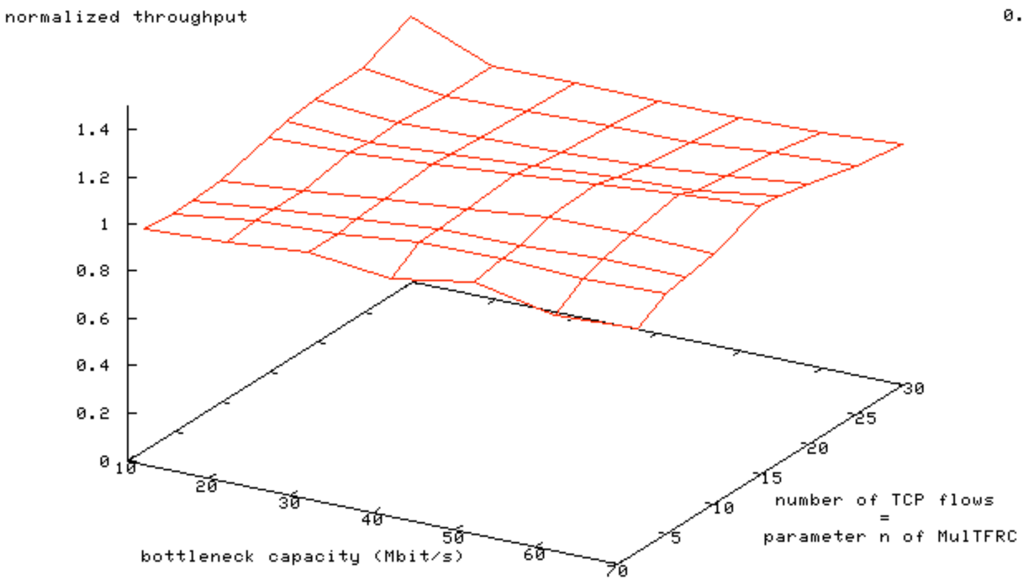


Some evaluation results

More: Dragana Damjanovic, Michael Welzl: "MulTFRC: Providing Weighted Fairness for Multimedia Applications (and others too!)", accepted for publication in ACM Computer Communication Review, 2009.

DropTail

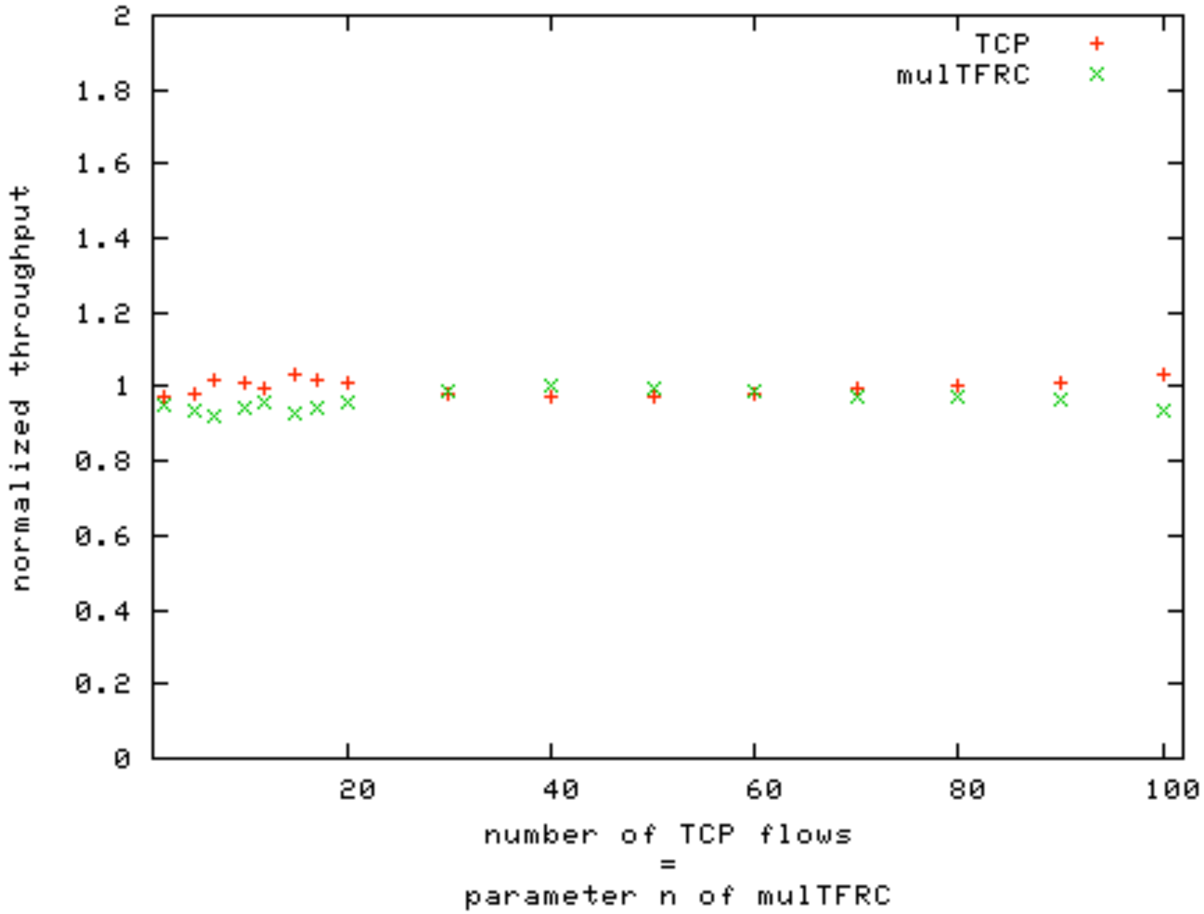




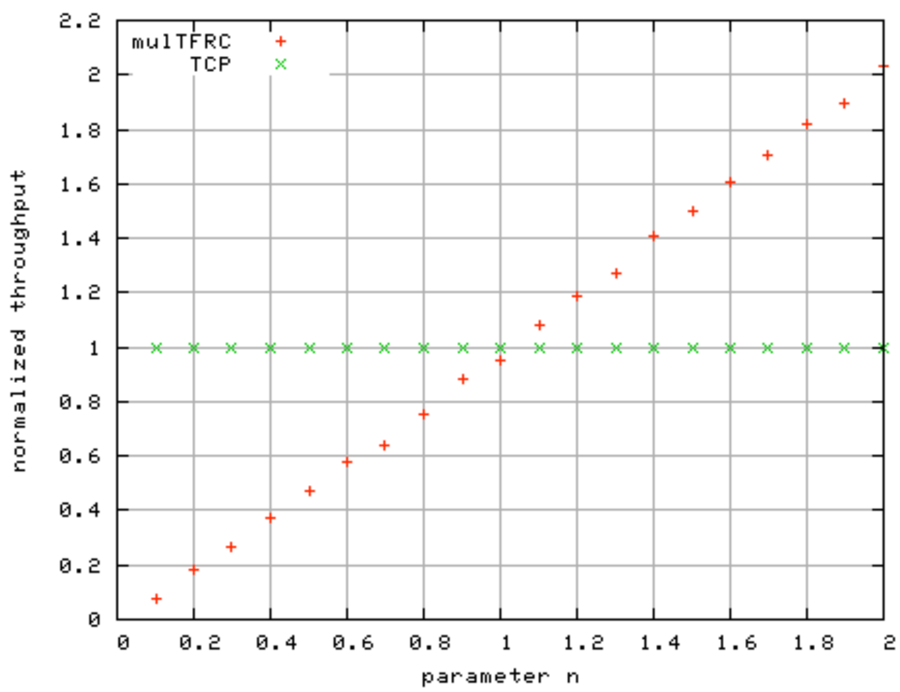
RED

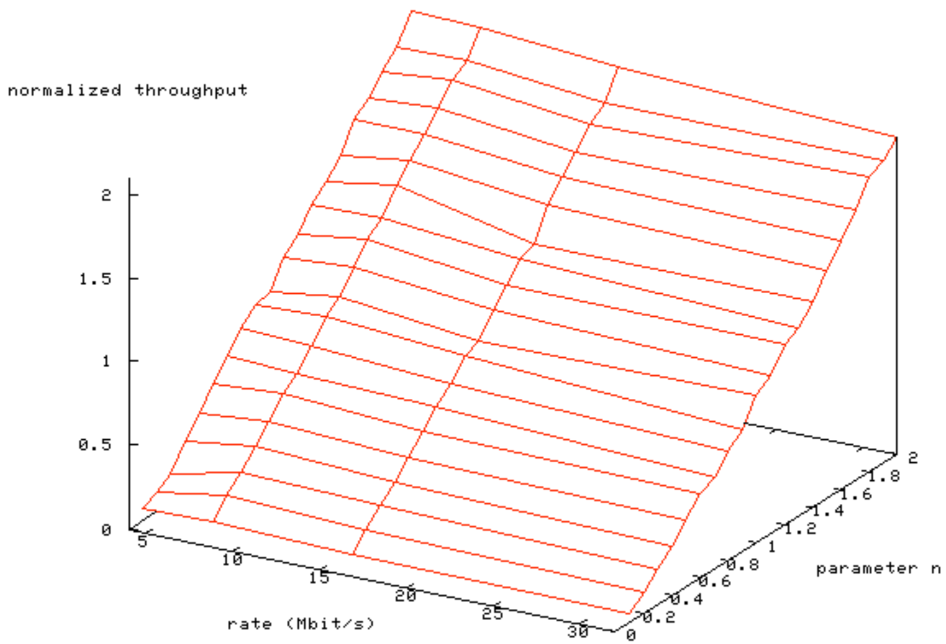
Real-life tests

(local testbed, bottleneck link 32Mbit/s)



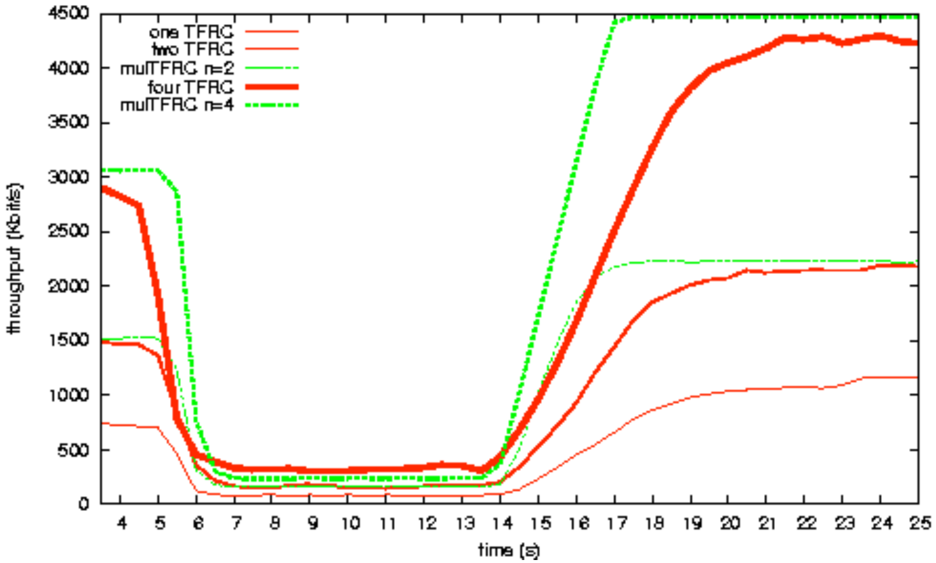
Zooming into the 0 ≤ *N* ≤ 2 range

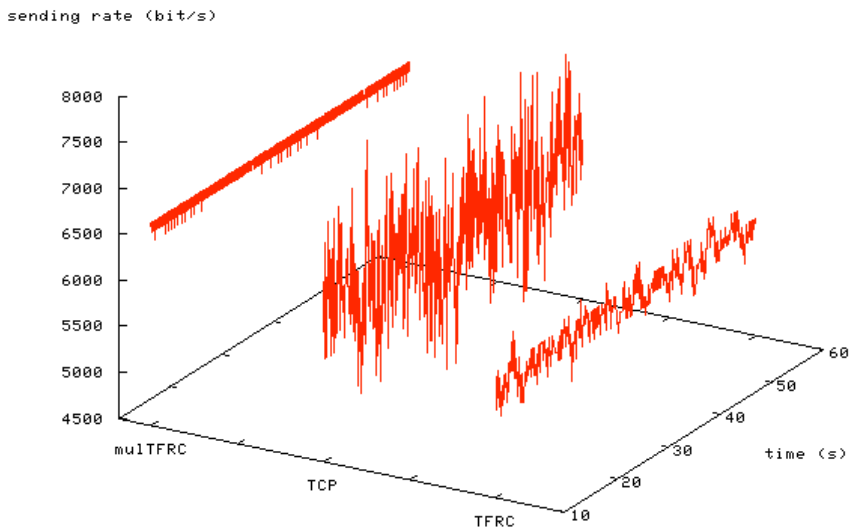




Bottleneck link capacity 4 Mbit/s

Responsiveness and smoothness





mulTFRC n=4; 4 TCP; 4 TFRC

Reasons to use MulTFRC

- *N*-TCP-friendly congestion control attractive
  - better performance for $N>1$

- Why is this better than multiple real TFRCs?
  - More reactive and smoother
  - Tunable congestion control with fine granularity and large range for *N*, including $0<N<1$
  - Less overhead (connection setup, teardown, state in end systems, ..)
  - No need to split data across multiple connections

How should it be used?

- MulTFRC also makes sense for reliable transfers (e.g. files)
  - But these apps don't need a smoother rate, and TFRC is generally less reactive than TCP...
- Setting *N*
  - Only at the beginning (otherwise: implications unknown)
  - Our suggestion: limited to 6
    - N TCPs alone: roughly up to 100-100/(1+3N) % bottleneck saturation
    - 1 flow 75%, 3 flows, 90%, 6 flows 95%
    - Gain decreases as N grows; from 1=>2 14.3 %, but less than 1% beyond 6
    - 6 TCPs from one host not uncommon
    - 6 will let users saturate their bottleneck (typically access link) better than one TFRC or TCP, larger numbers will still make the flow more aggressive when competing with others

Thank you