TCP Nice: A Mechanism for Background Transfers

Arun Venkataramani, Ravi Kokku,
and Mike Dahli
University of Texas at Austin, Austin, TX

ACM SIGOPS Operating Systems Review, 2002

Outline

- Introduction
- Design and implementation
  - Background
  - TCP Nice
  - Analysis
- Evaluation
- Conclusions

Introduction (1/2)

- Many distributed applications can make use of large *background transfers*
  - data that humans are not waiting for
  - non-deadline-critical
  - unlimited demand
- Hand tuning the background transfers risks
  - complicating applications
  - being too aggressive
  - being too timid

Introduction (2/2)

- Goal:
  - *manage network resources* to provide an abstraction of background transfers.

- TCP Nice:
  - interferes little with foreground flows
  - reaps a large fraction of spare network bandwidth
  - simplifies application

Design and implementation

- Background
  - existing algorithms
- TCP Nice
- Analysis

Background (1/2)

- Congestion control mechanisms in traditional TCP
  - *congestion signal* (packet loss)
  - *reaction policy* (AIMD)

- Problem: signal comes after damage done
- Solutions: proactively detects congestion
  - use *increasing RTT* as congestion signal

    congestion <=> increasing queue lengths
    <=> increasing RTT

Background (2/2)

- TCP Vegas
  - differs from TCP-Reno in its *congestion avoidance phase*

```
$E \leftarrow \frac{W}{minRTT}$                // Expected throughput
$A \leftarrow \frac{W}{observedRTT}$           // Actual throughput
$Diff \leftarrow (E - A) \cdot minRTT$

if ($Diff < \alpha$)
        $W \leftarrow W + 1$
else if ($Diff > \beta$)
        $W \leftarrow W - 1$
```

TCP Nice (1/4)

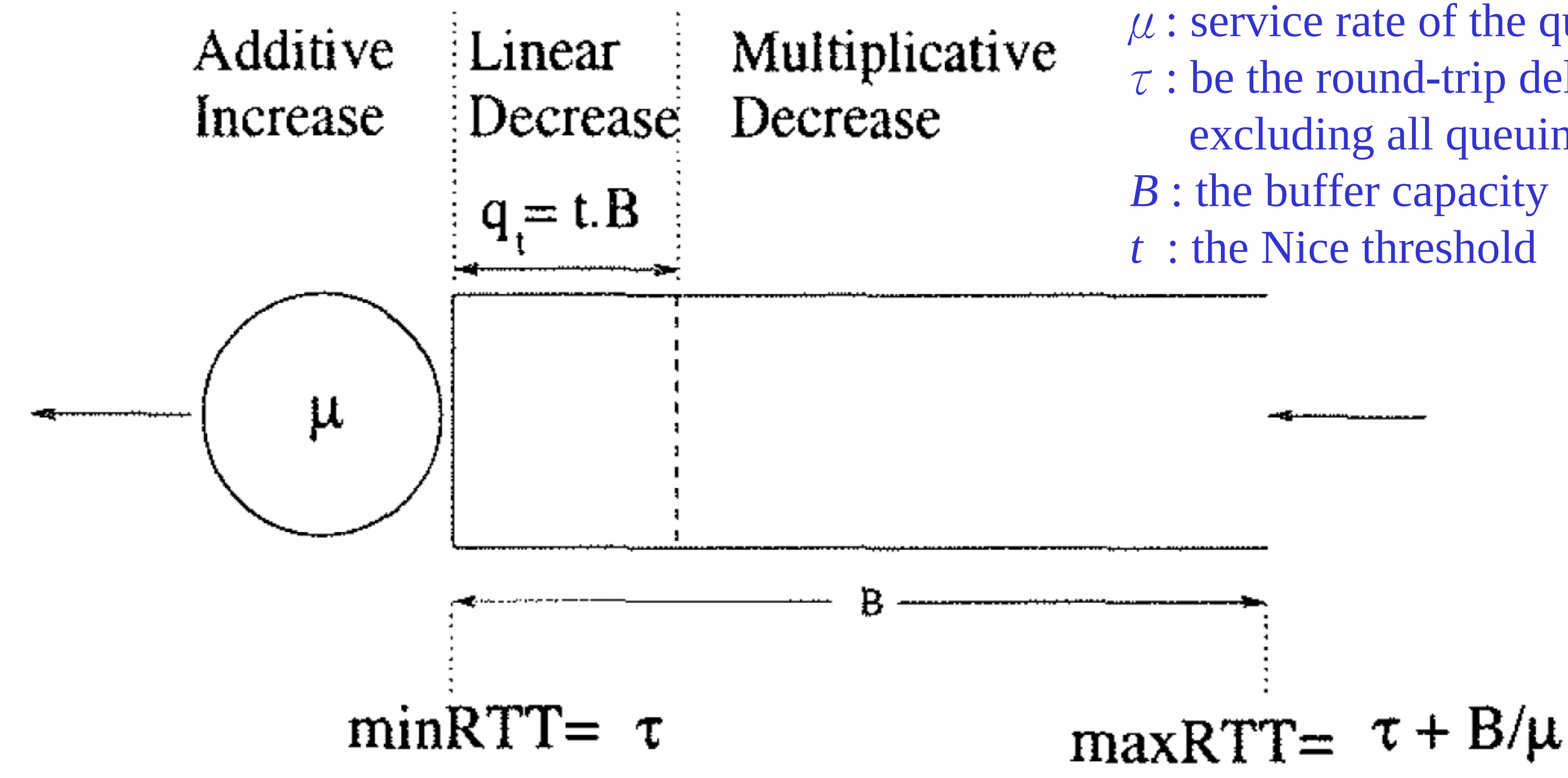
- Only modifies *sender-side* congestion control
- Adds three components to Vegas
  - more sensitive congestion detector
  - multiplicative decrease on early congestion
  - allow cwnd < 1.0

TCP Nice (2/4)

- Estimates the *total queue size* at the bottleneck
- Total queue size exceeds a fraction of the estimated maximum queue capacity
  - signals congestion

- In order to affect window sizes <1,
  - send a packet out after waiting for the number of smoothed round-trip delays.
  - act as network probes waiting for congestion to dissipate

TCP Nice (3/4)



μ : service rate of the queue
τ : be the round-trip delay excluding all queuing delays.
B : the buffer capacity
t : the Nice threshold

Fig.1 Nice Queue Dynamics

TCP Nice (4/4)

- per-ack operation:

```
if ($curRTT > minRTT + threshold * (maxRTT - minRTT)$)
   $numCong$++;
```

- per-round operation:

```
if ($numCong > f * W$)
   $W = W/2$
else { … Vegas congestion control }
```

Analysis (1/4)

- Prove small bound on interference
- Main result
  - interference *decreases exponentially* with bottleneck queue capacity, independent of the number of Nice flows
- Unrealistic model
  - Synchronous packet drop
  - consider fixed number of connections, *m* following Reno, and *l* following Nice

Analysis (2/4)

- We trace these window sizes across *periods*.

- The end of a period and the beginning of the next is *marked by a packet loss.*

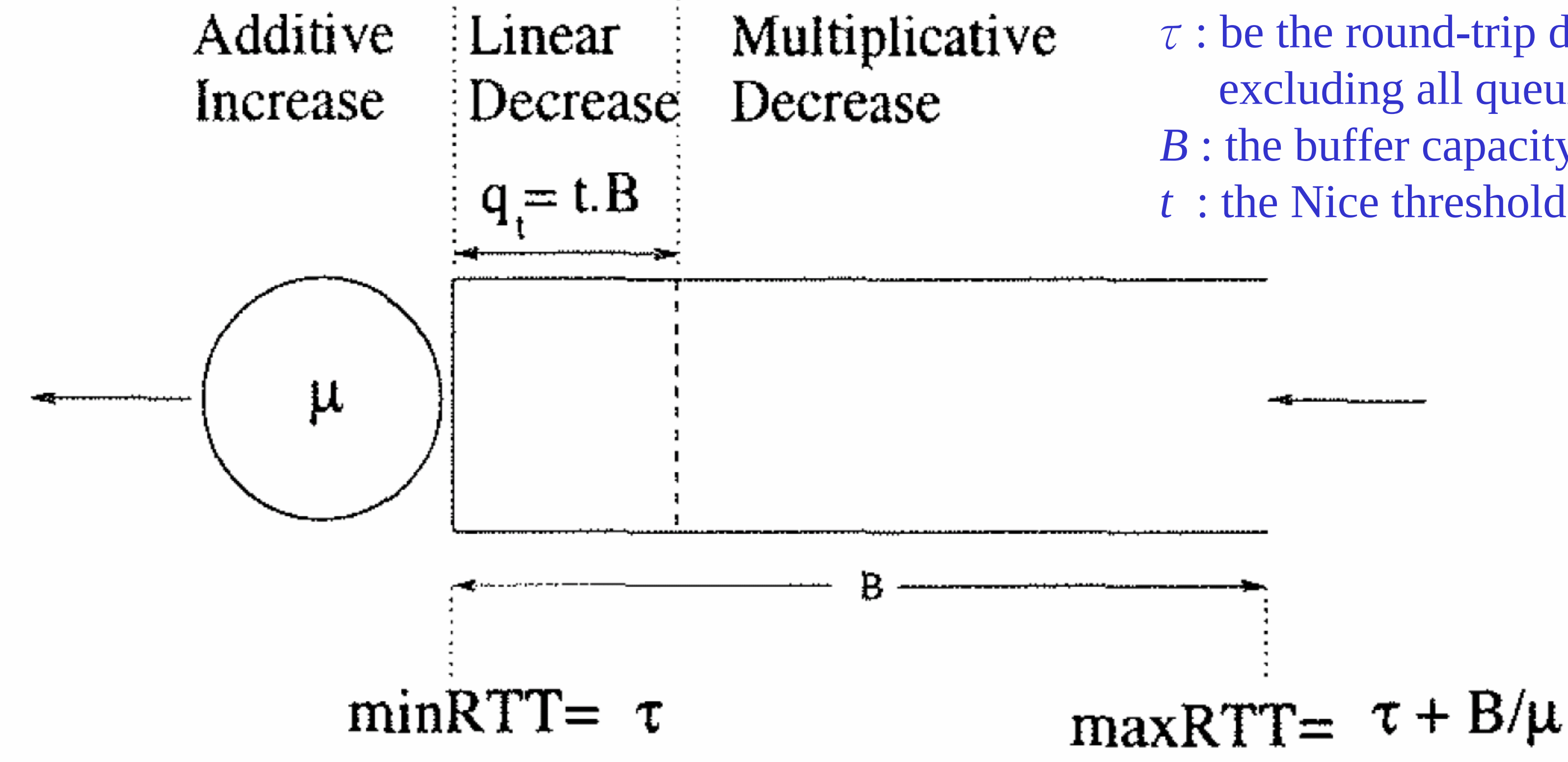
- $W_r(t)$ *and* $W_n(t)$: the total number of outstanding Reno and Nice packet at time t

Analysis (3/4)

- $W(t) = W_n(t) + W_r(t)$
- The window dynamics in any period can be split into three intervals
  - *Additive Increase, Additive Increase*
  - *Additive Increase, Additive Decrease*
  - *Additive Increase, Multiplicative Decrease*

$$I \leq \frac{4m \cdot e^{\left(-\frac{B(1-t)\gamma}{m}\right)}}{(\mu\tau + B)\gamma}$$

Analysis (4/4)



μ : service rate of the queue
τ : be the round-trip delay excluding all queuing delays.
B : the buffer capacity
t : the Nice threshold

Evaluation (1/5)

- Packet size: *512* bytes, propagation delay: 50ms.
- Single bottleneck topology.
- 15 minute section of a Squid proxy trace logged
- $t = 0.1$, $f = 0.5$ (default)
- Parameters
  - *spare capacity, number of Nice flows, threshold*
- Metric
  - average document transfer latency

Evaluation (2/5)

- Experiment 1: *vary the spare capacity*

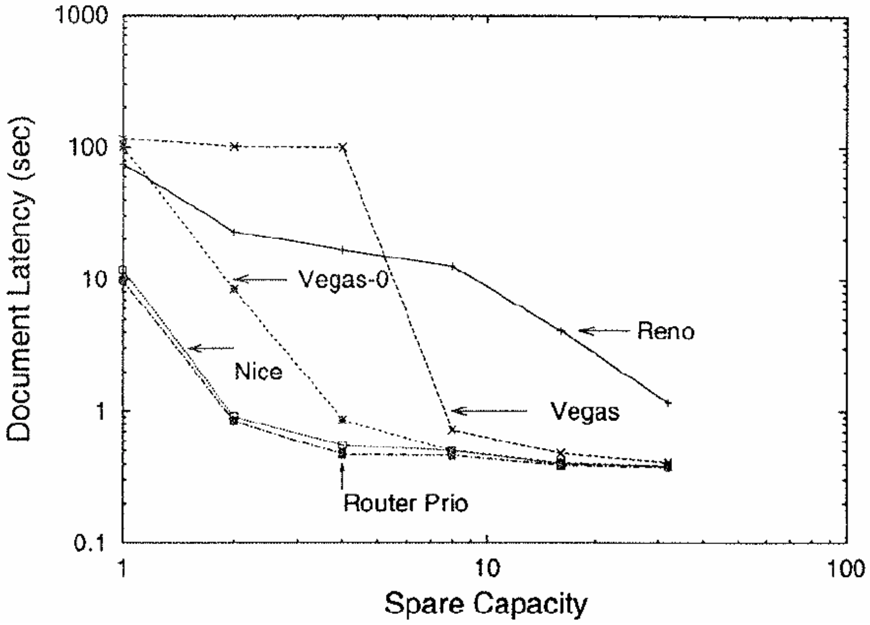


Fig.2 Spare capacity vs Latency. Nice causes low interference even when there isn't much spare capacity.

Evaluation (3/5)

- Experiment 2: *vary the number of background flows*

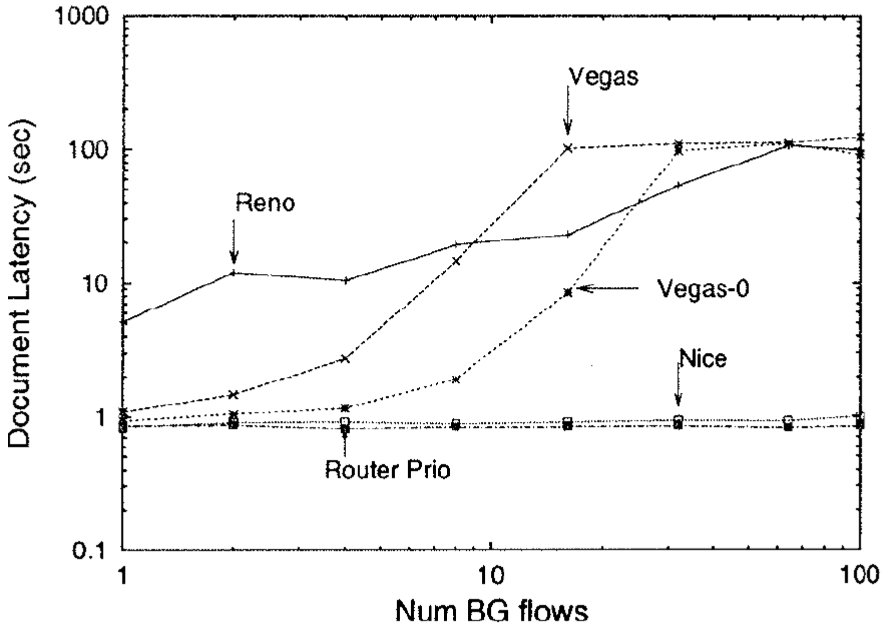


Fig.3 Number of BG flows vs Latency. W < 1 allows Nice to scale to any number of background flows.

Evaluation (4/5)

- Experiment 2: *vary the number of background flows*

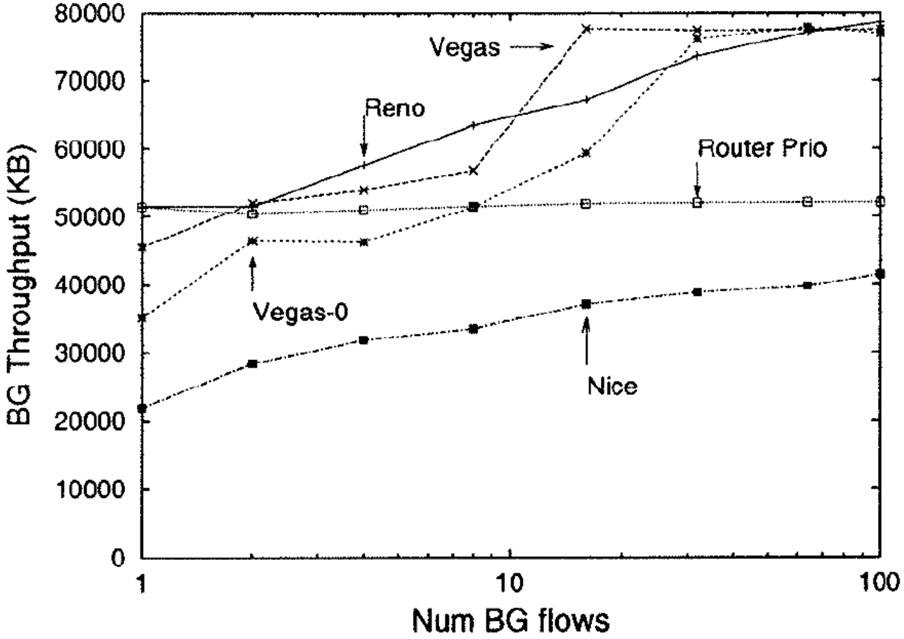


Fig.4 Number of BG flows vs BG throughput. Nice utilizes 50-80% of spare capacity *without stealing any bandwidth from FG.*

Evaluation (5/5)

- Experiment 3: *vary the threshold value*

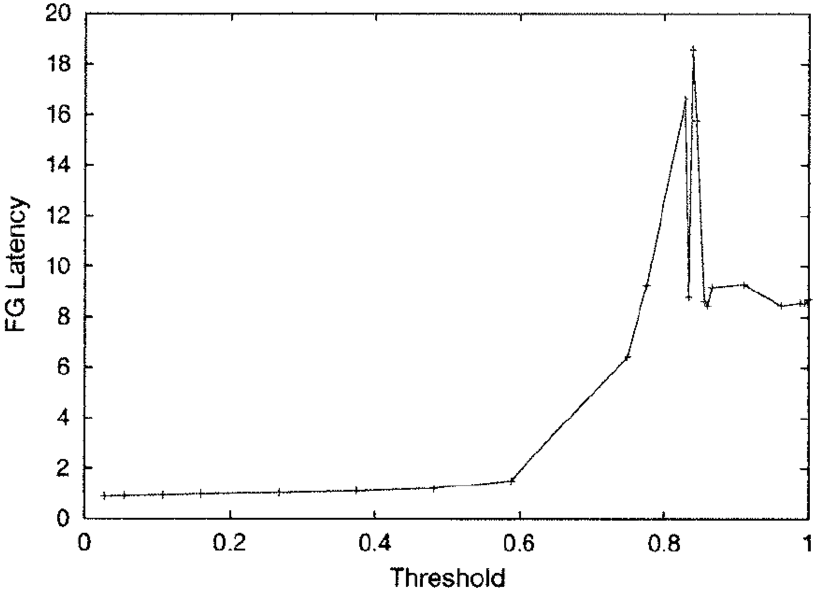


Fig.5 Threshold vs FG latency. Dependence on threshold is weak.

Conclusions

- An end-to-end strategy optimized to support background transfers.

- Enough usable spare bandwidth out there that can be nicely harnessed

- Nice makes application design easy