Application Level Congestion Control Enhancements in High BDP Networks

Anupama Sundaresan

Organization

- Introduction
- Motivation
- Implementation
- Experiments and Results
- Conclusions

ENABLE Overview

- Developing a “Grid” service
  - to provide a monitoring infrastructure
  - to provide the current network information to network-aware applications
- Network-aware applications will be able to obtain information about resource availability, in particular the network’s capabilities and status
- Applications will make informed QoS decisions based on the network monitoring information obtained from the database
- Once the application finds out the amount of network resources it has, the work in this thesis will help the application in maximizing the performance with the available resources

TCP Congestion Control

- Transmission Control Protocol (TCP) uses a set of Congestion Control algorithms to control the sending behavior
  - Slow Start algorithm - exponential increase in *CWND* from one
  - Congestion Avoidance - when *CWND* > *ssthres*h (slow start threshold) increase in *CWN*D is linear (1/CWND for every ACK)
- With a retransmission timeout, slow start is triggered again

HTTP Overview

- HTTP uses TCP as the transport protocol
- TCP's slow start phase predominates web flows which are of short duration
- HTTP 1.0 - A new connection is opened for each request
  - connection establishment latency and slow start reduces performance
- P-HTTP - Multiple requests are pipelined on a persistent connection
  - connection latency for each request is overcome
  - slow start on *each* request overcome

Problems of TCP on high BDP links

- 16 bits of advertised window in TCP header
  - $Throughtput_{max} = \frac{RcvBufSize}{RTT}$ overcome by window scaling extensions
- Startup behavior - Slow Start phase at the beginning of a connection
- Slow start time more than 1second on high latency links
- Short duration flows predominated by slow start and hence poor bandwidth utilization
- Occurrence of a ‘minor’ congestion event triggers congestion avoidance or slow start which in turn leads to inefficient utilization of bandwidth

Motivation and Solution

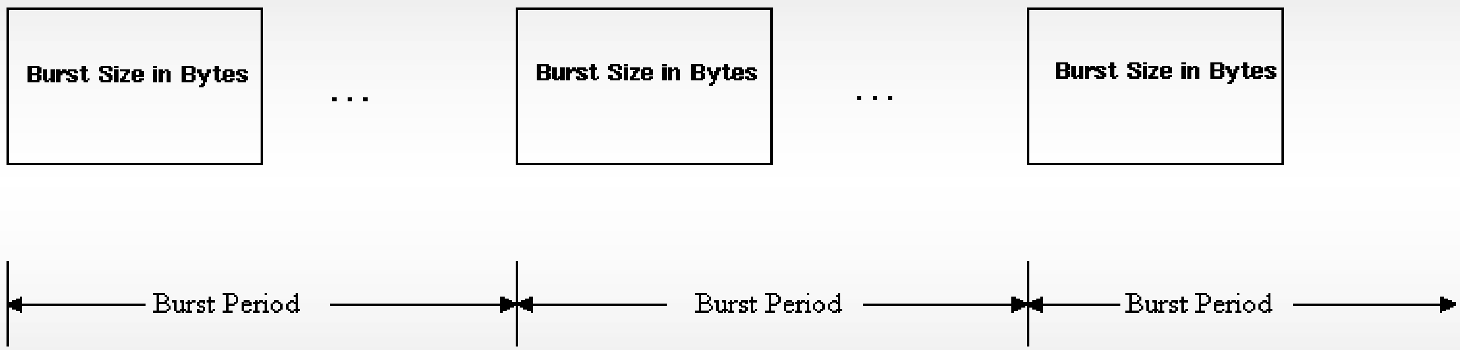
- Improving the performance of TCP flows especially short duration flows on high latency links
- Giving control to the application on the amount of bytes it writes on the network
- Due to the pitfalls of TCP on high bandwidth and high latency links, idea of experimenting with turning off congestion control(*NOCC*) in TCP came up
- *NOCC* is not limited by the *CWND* maintained by the TCP sender and sends up to the receiver's advertised window
- Pacing in the application along with *NOCC* gives the application the control of how much of data it is sending onto the network

Implementation

- Standard TCP implementation, sender sends a packet on the network if $\#pkts\_in\_flight < Min(\mathrm{Re}ceiver\_adv\_wnd, CWND)$
- In TCP with NOCC, application turns off congestion control through a *setsockop*t with *TCP_NO_CONGESTION* as a parameter, so sender sends a packet if $\#pkts\_in\_flight < \mathrm{Re}ceiver\_adv\_wnd$
- The *setsockopt* sets a flag *nocc* based on which modifications were made to the sending engine and retransmit engine of TCP on Linux 2.2.13
- A *setsockopt* to set the *CWND* to the initial value specified by the application with parameter *TCP_SET_CWND*
- A *setsockopt* to capture the number of retransmissions occurring on a connection with parameter *TCP_TOTAL_REXMITS*

Implementation (contd.)

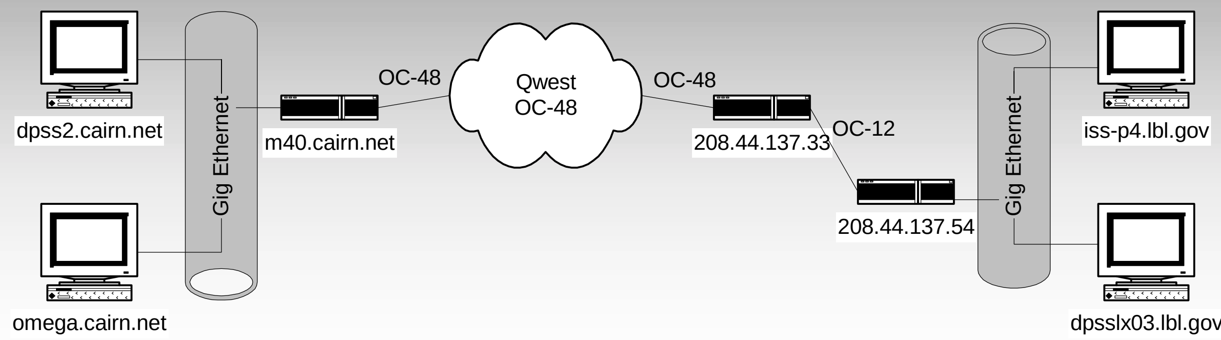
- The /proc interface was modified to display the retransmit information in */proc/net/tcp*
- *Pacing* was implemented in *Apache 1.3.12*



- Pacing parameters (*burst size* and *burst period*) are specified in *httpd.conf*
- Apache was modified to handle modified HTTP Get requests with burst size and burst period as parameters

Experiments and Results

Experimental Setup



TCP Transmitter	omega.cairn.net with Linux-2.2.13 with *NOCC*
TCP Receiver	iss-p4.lbl.gov
Round Trip Time	~67ms
Link Bandwidth	622Mbps
Web Server	Apache 1.3.12 on omega.cairn.net

Tools Used and Test Scenarios

- NetSpec, a traffic generation tool was used to generate Full and Burst traffic
- Apache for Linux was the web server used
- Web Server benchmarking tool Zeus was used to issue modified HTTP Get requests

NetSpec
- Startup Behavior
- Congestion Recovery Behavior

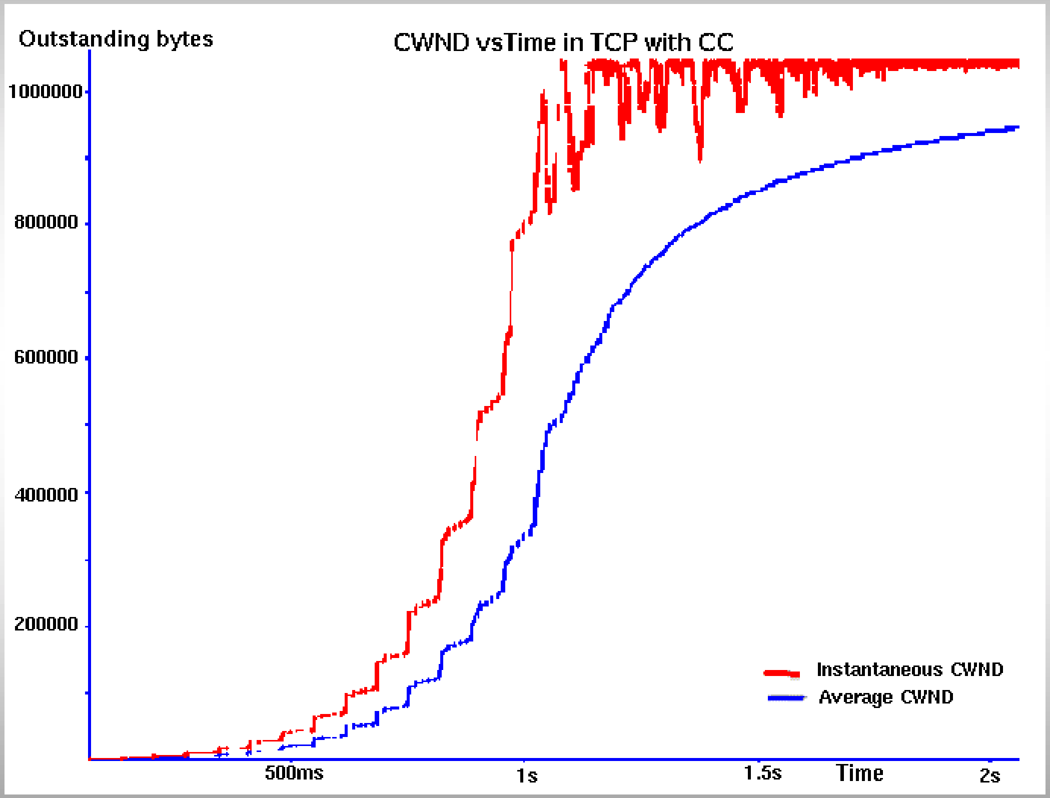
Apache
- Behavior for different flow durations HTTP1.0
- Behavior for different flow durations P-HTTP
- Performance during Congestion

Performance Metrics

- Outstanding Bytes
  - The number of packets in flight forms a direct measure of the Congestion Window
- Received Throughput
  - $\text{Re}ceived\_Throughput = \frac{Num\_bytes\_rcvd}{dur\_of\_transfer} Mbps$
- Offered Load
  - $Offered\_Load = Burst\_Size * 8 * \frac{1\sec ond}{Burst\_Period} Mbps$
- Response Time
  - This is the duration the client which, sends a HTTP Get request to the Web Server spends waiting before it can produce the requested web page to the end user.

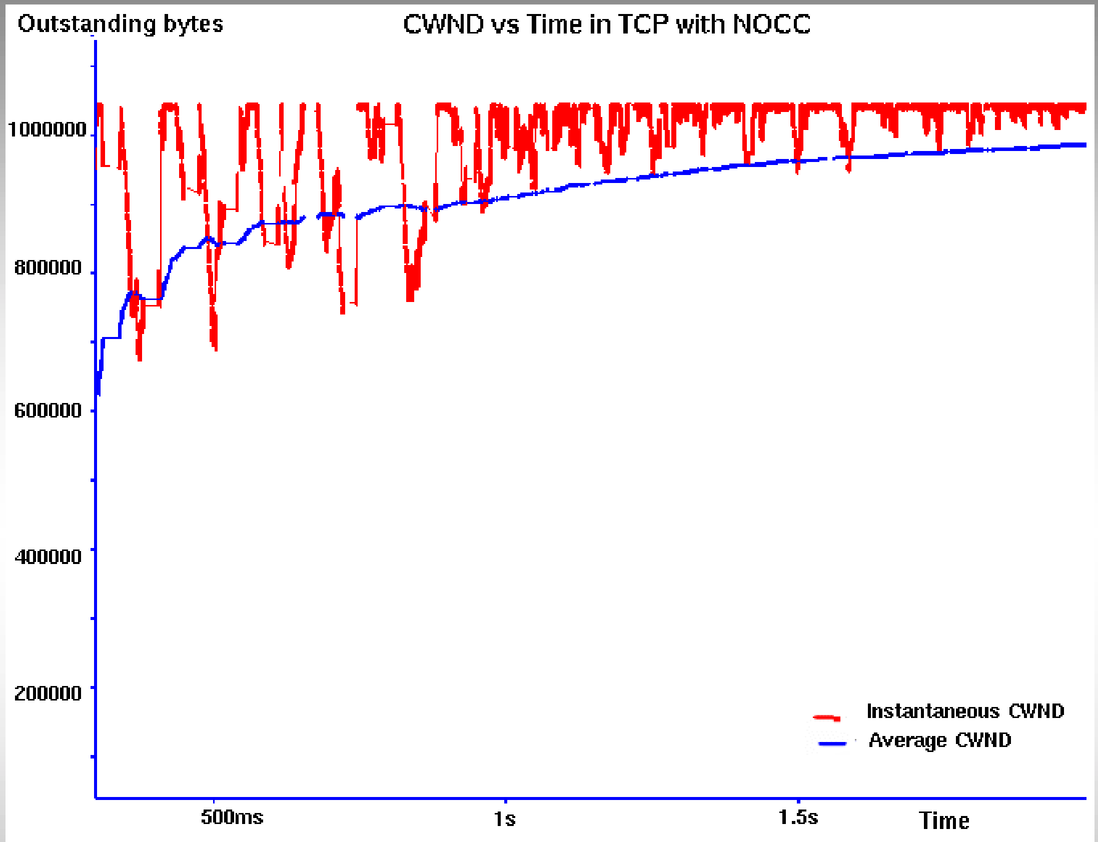
NetSpec Results - Startup Phase

Slow Start phase in TCP with CC



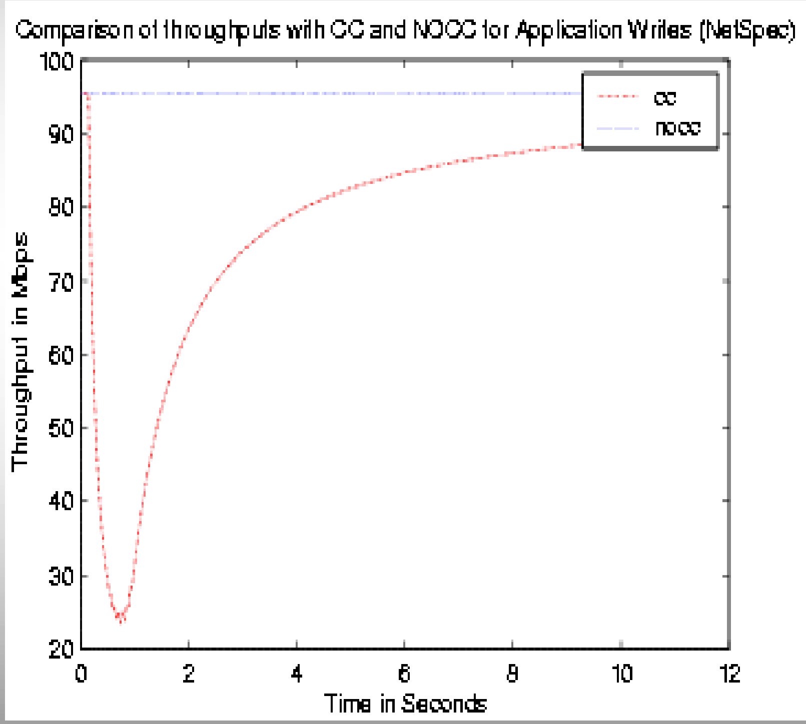
- Detrimental for short duration flows as the *CWND* takes more than a second to open out

TCP with NOCC - Startup Behavior



- Number of outstanding bytes on the network increases to the receiver’s advertised window as soon as the sender starts sending

Instantaneous Transmitted Throughputs

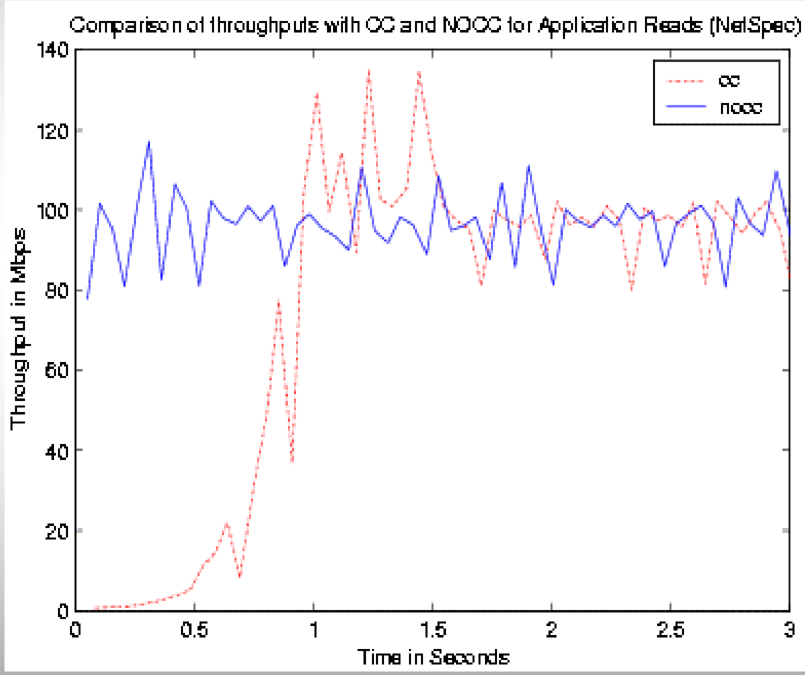


Burst Size = 128KB

Burst Period = 10ms

- *NOCC* transmits bursts without failed cycles
- CC is limited by the *CWND* => drop in transmitted throughput
- Throughput rises as *CWND* increases

Instantaneous Received Throughputs

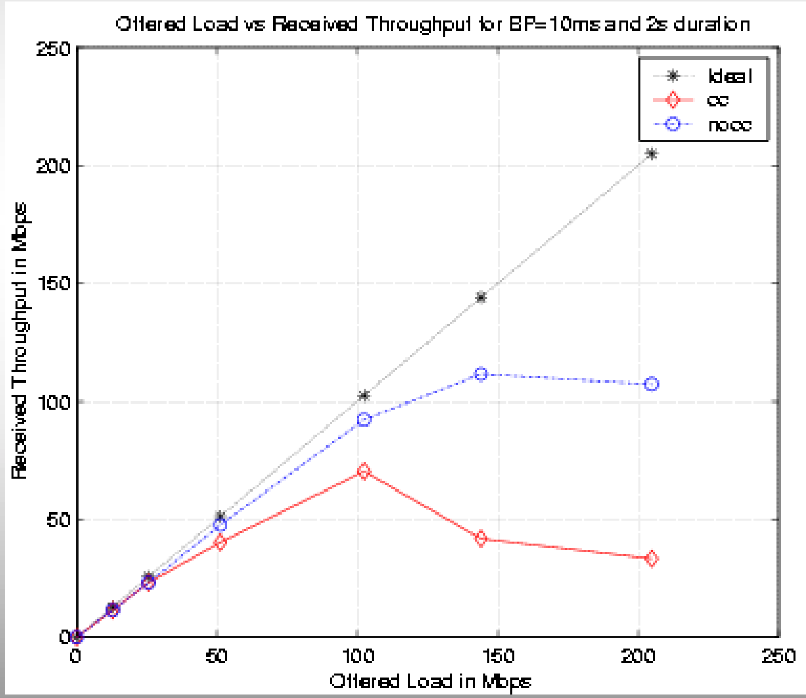


Burst Size = 128KB

Burst Period = 10ms

- CC shows a prominent startup phase
- NOCC shows steady behavior throughout the duration of the flow

Received Throughputs for Short Duration flows



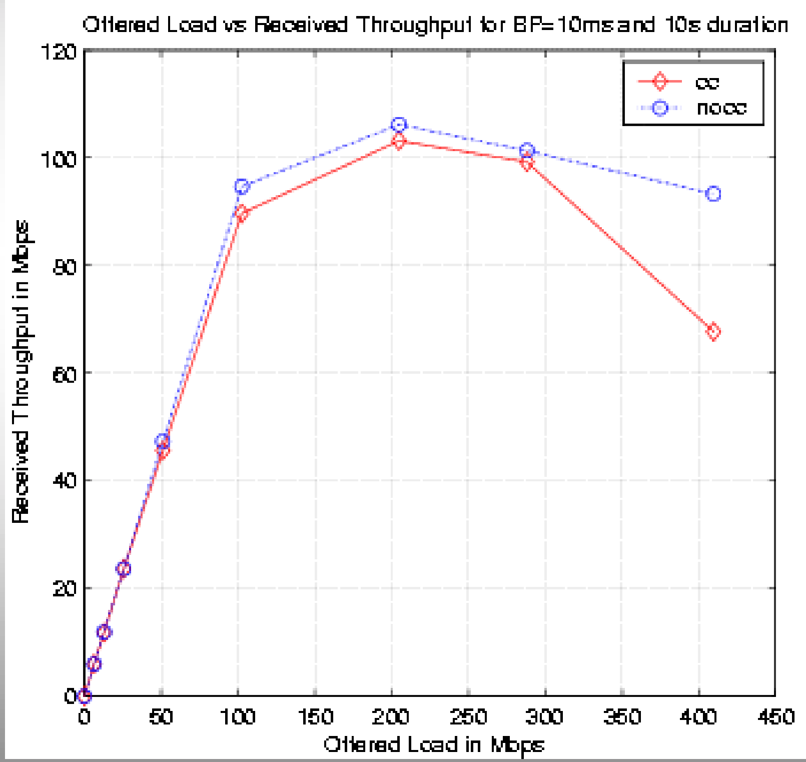
Burst Size=8KB,16KB,…256KB

Burst Period = 10ms

Duration = 2s

- *NOCC* performs significantly better than CC
- In CC, flow is mostly in slow start => under utilization of available resources

Received Throughputs for Long Duration flows



Burst Size=8KB,16KB,…256KB

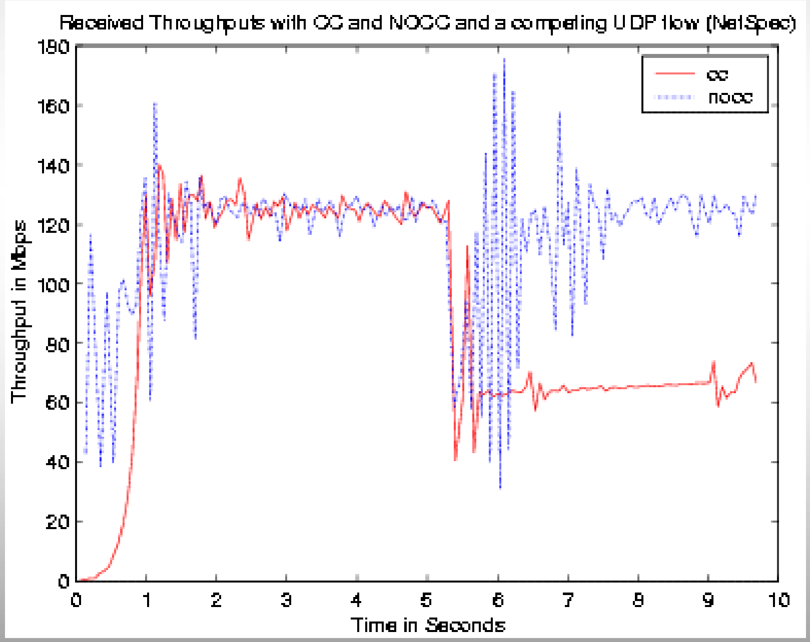
Burst Period = 10ms

Duration = 10s

- As offered load increases, *NOCC* performs better than CC
- CC is limited by *CWND*

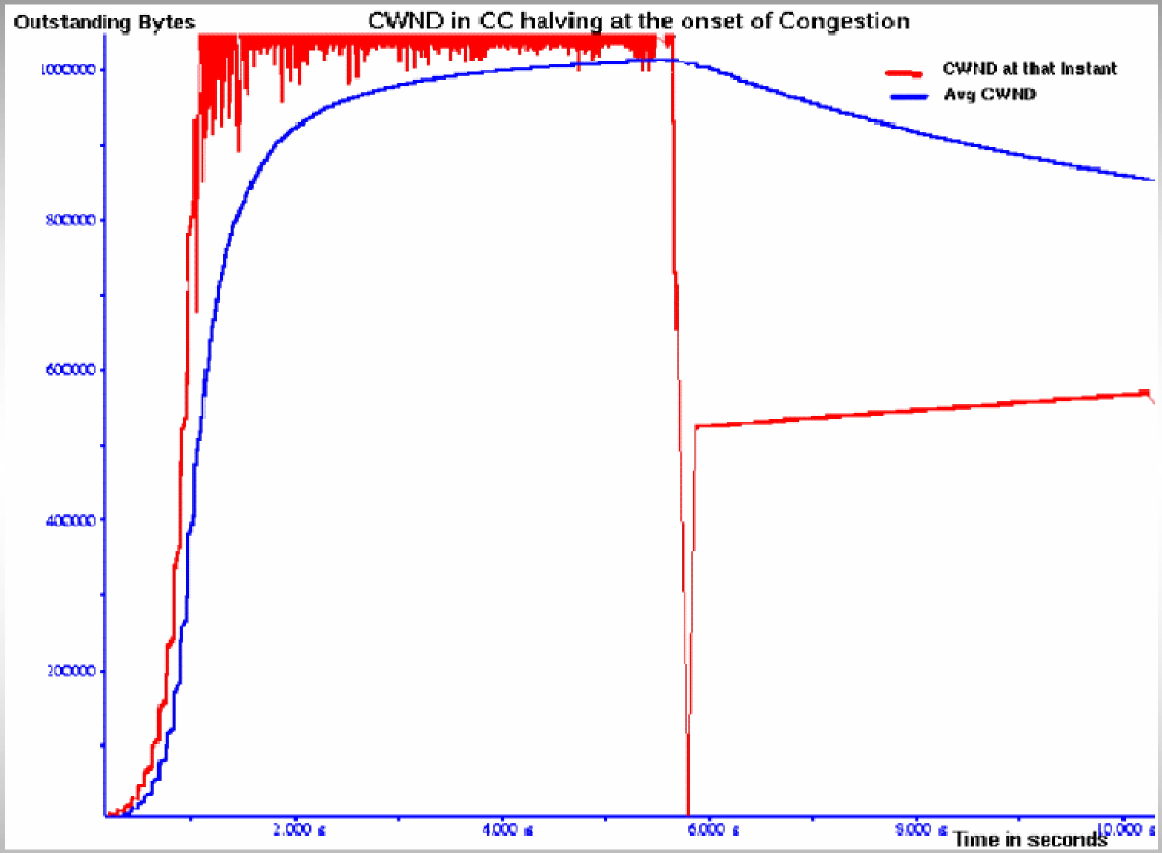
NetSpec Results - Congestion Recovery

CC and NOCC flows with 'minor' Congestion Event



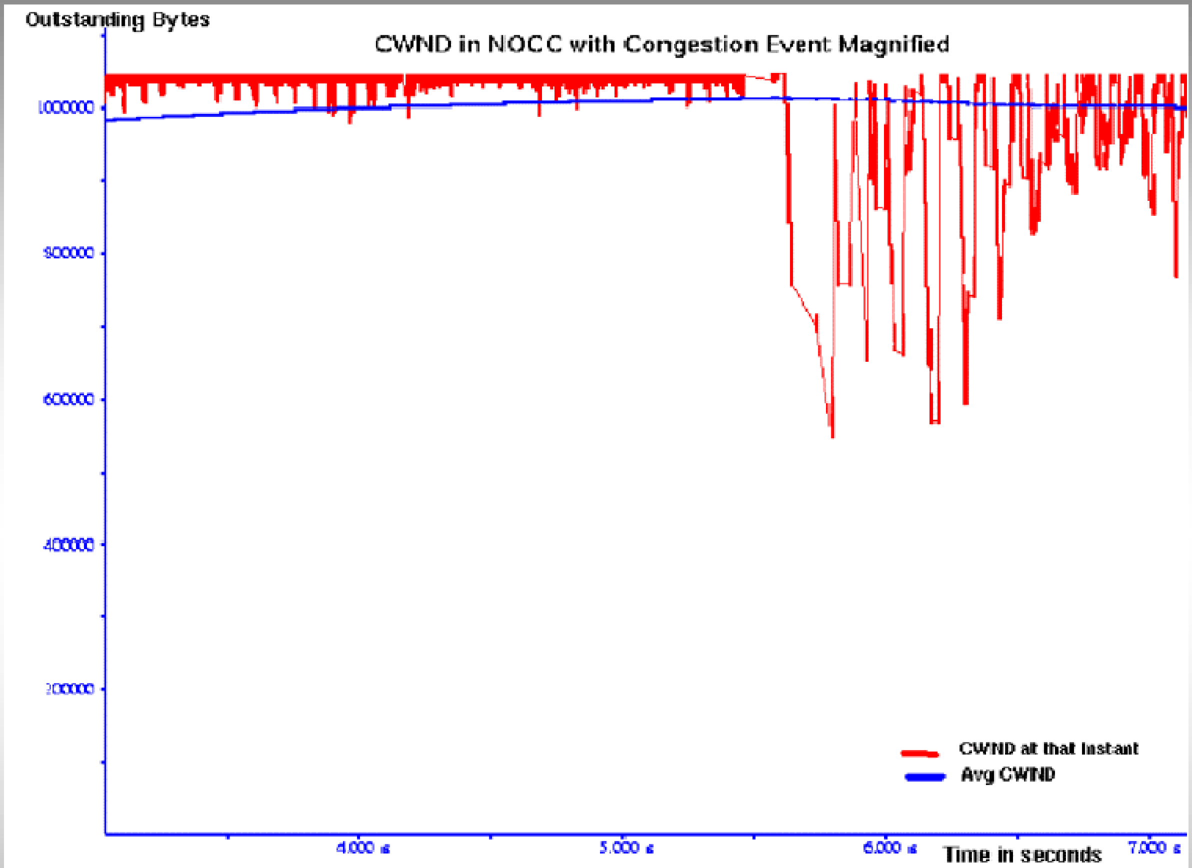
- A minor congestion event simulating a single bit error was introduced
- CC halves *CWND* and goes into Congestion Avoidance => halves sending rate
- *NOCC* is able to maintain the throughput at the same level

CWND in CC flow in Congestion



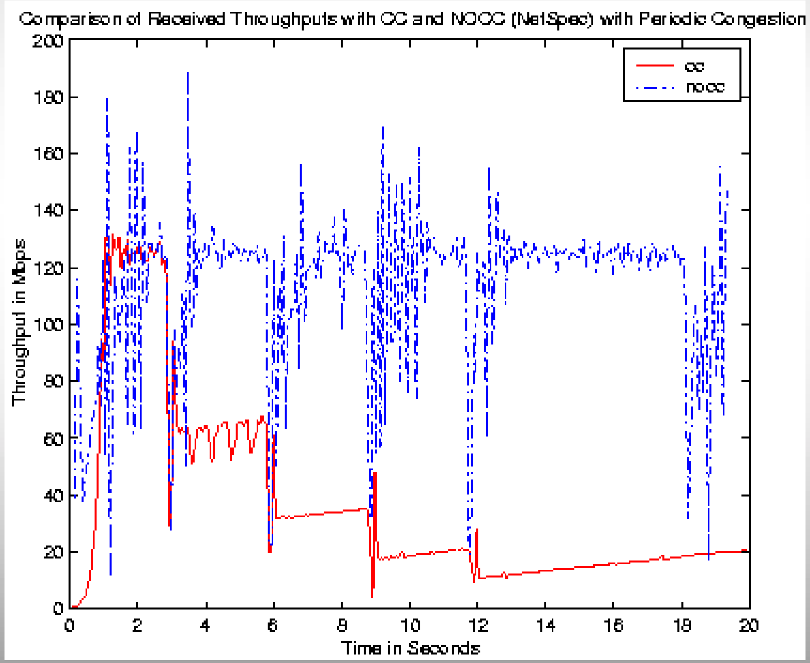
- *tcptrace* plot with *tcpdump* output showing CWND halving and Congestion Avoidance taking over

CWND in *NOCC* flow in Congestion



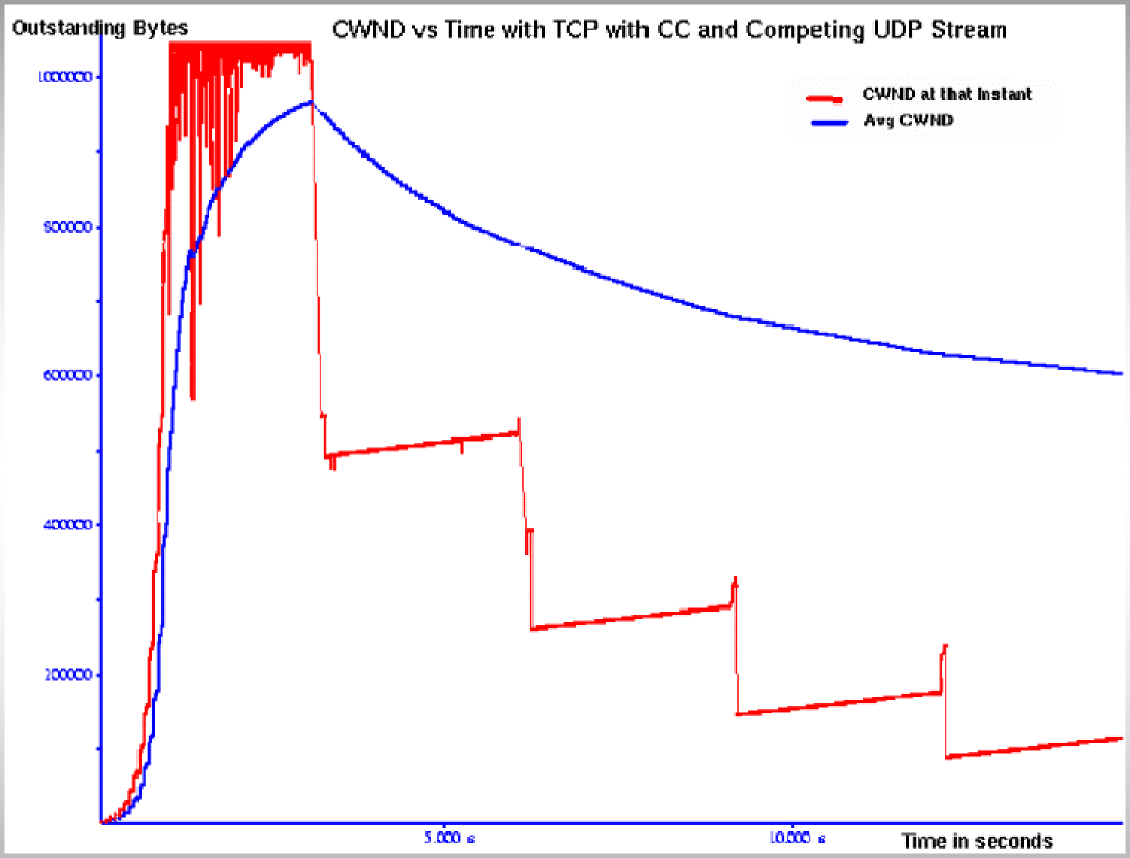
- A congestion event affects a NOCC flow but the CWND is not halved and the sender sends up to the receiver's advertised window at any instant

CC and *NOCC* with periodic congestion



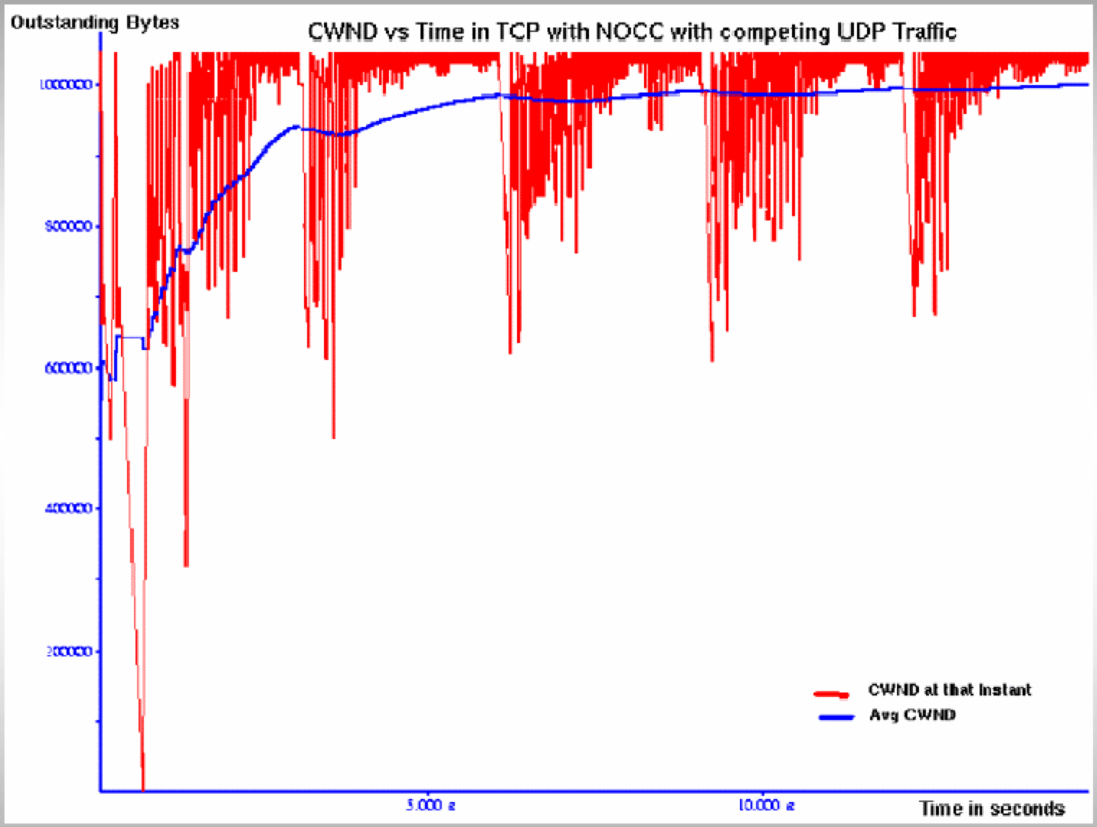
- UDP flow congests every 3 seconds
- CC halves sending rate => effectively achieves very little throughput
- NOCC achieves significantly better throughputs

CC flow with periodic congestion



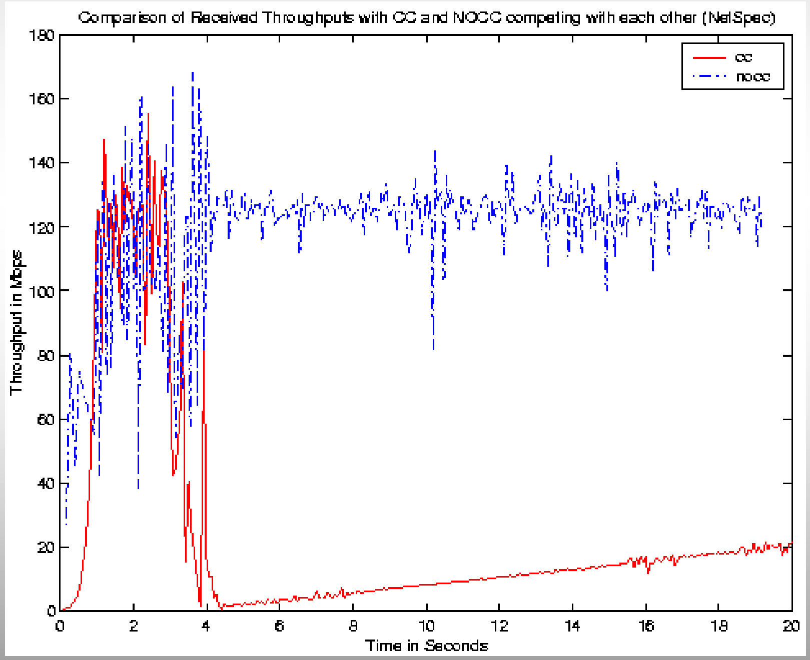
- *CWND* halves at every congestion event => average number of packets in flight decreases

NOCC flow with periodic congestion



- *NOCC* has a constant number of packets in flight

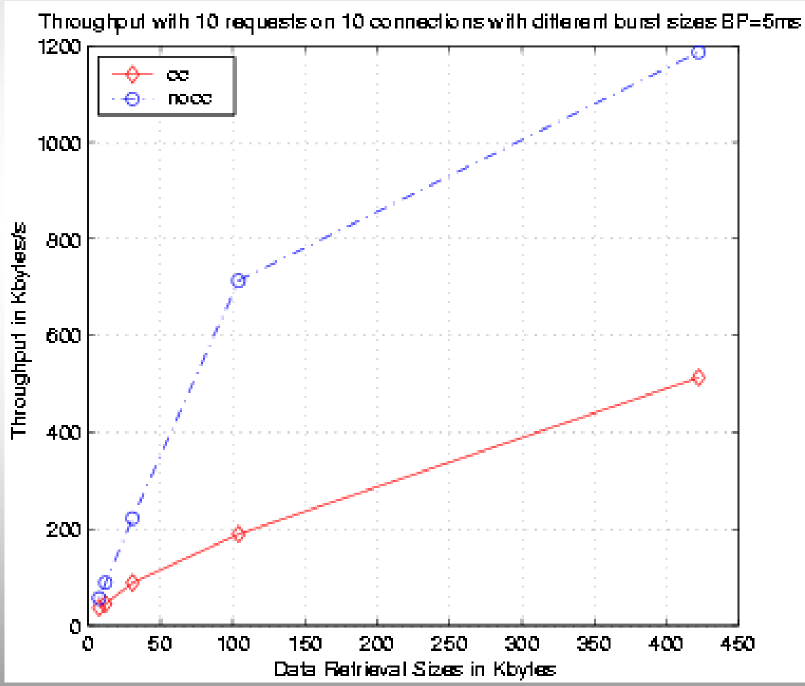
CC and *NOCC* flows



- *NOCC* is aggressive due to the lack of the *CWND* parameter
- CC flow is throttled and performs very poorly

Apache Tests

Burst tests with multiple connections HTTP1.0



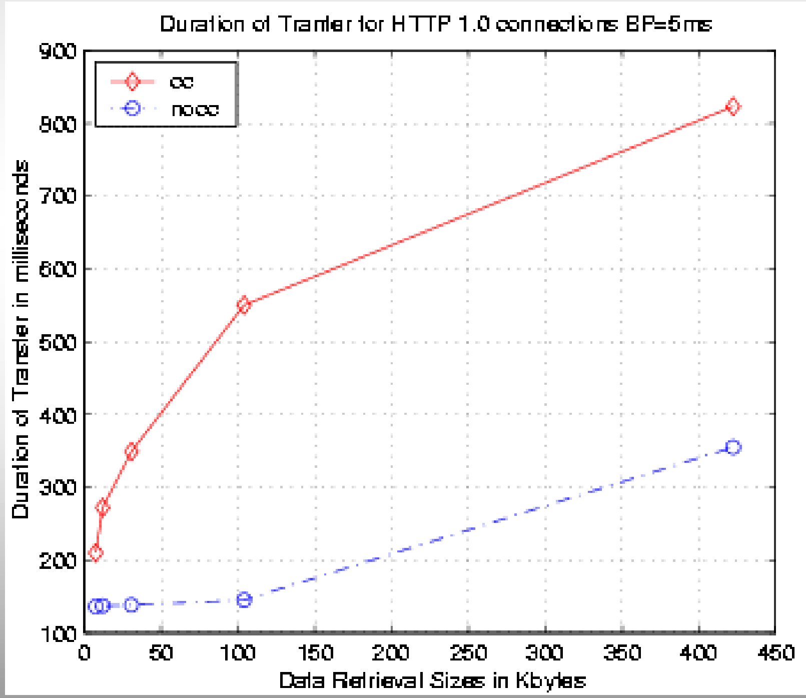
File Size in KB=7,10,30,100,422

Burst Size = 32KB, 64KB, 128KB

Burst Period = 5ms

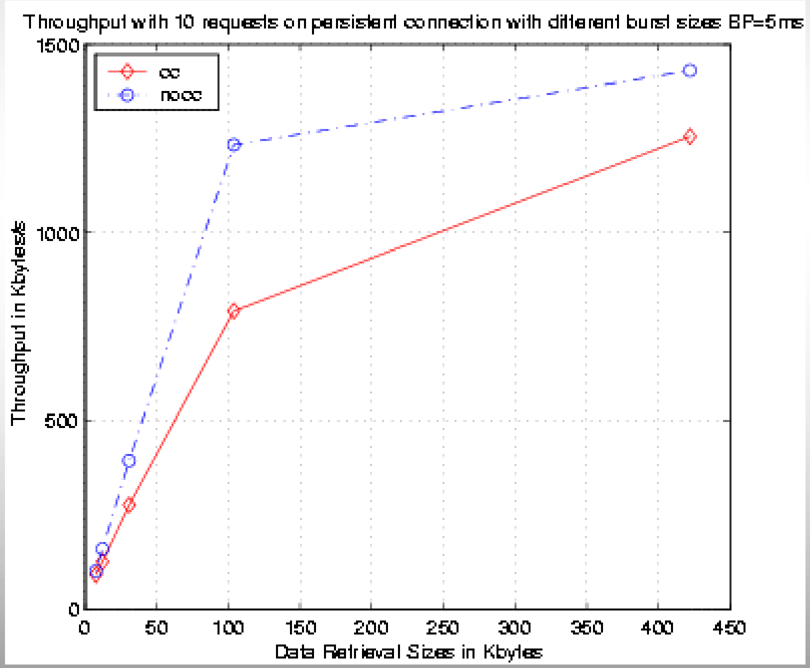
- *NOCC* does not wait for ACKs to increase *CWND* and so performs significantly better than CC
- The effectiveness of *NOCC* for short term flows is seen here

Burst Tests with HTTP 1.0 (contd…)



- The duration of transfer shows a significant reduction in *NOCC* case

Burst Tests with Persistent HTTP



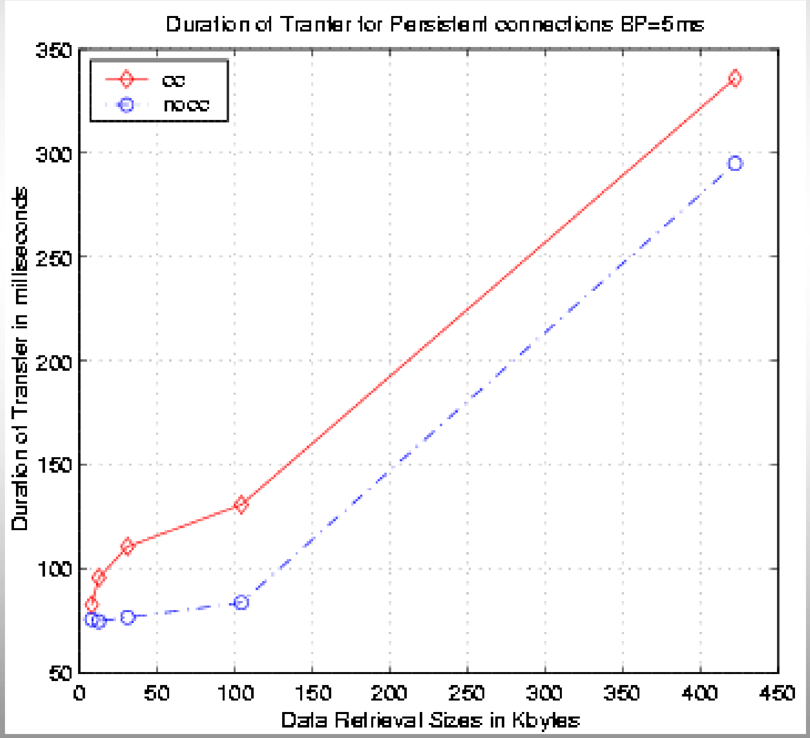
File Size in KB=7,10,30,100,422

Burst Size = 32KB, 64KB, 128KB

Burst Period = 5ms

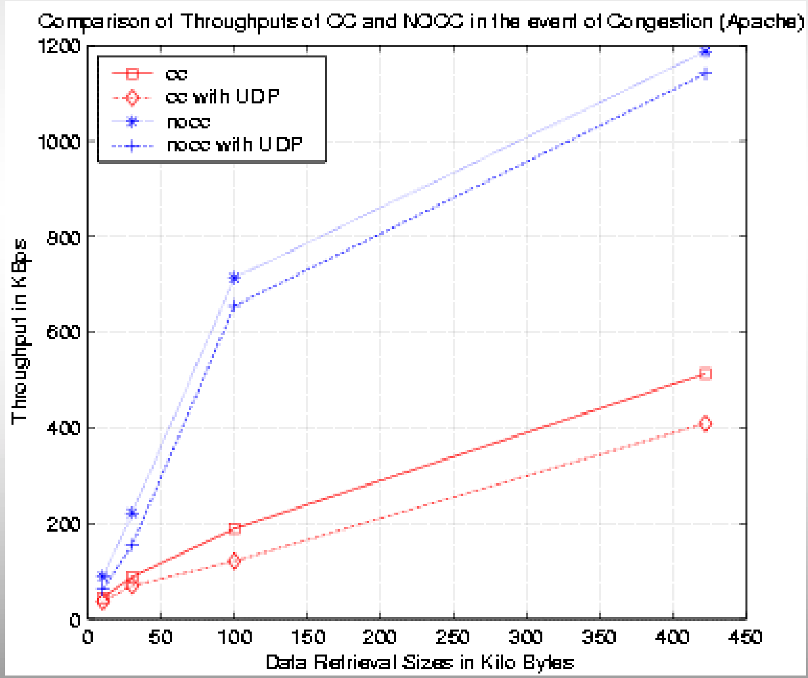
- P-HTTP was developed to overcome the connection request latency
- *NOCC* performs better than CC

Burst Tests with P-HTTP (contd.)



- The duration of transfer of *NOCC* is again seen to be significantly better than CC
- All requests sent on a single connection

CC and *NOCC* with Congestion



- Reduction in throughput in *NOCC* but performs significantly better than CC

Conclusions

- TCP's congestion control algorithms were designed for low bandwidth links prone to frequent congestion
- Slow Start causes an incredible startup phase problem which leads to poor utilization of the abundant bandwidth in high bandwidth links
- *NOCC* is advantageous to short term flows since it is not inhibited by the startup phase problem
- TCP reacts to single bit error losses adversely (Satellite links)
- *NOCC* does not halve the sending rate => gives better performance for random losses

Conclusions (contd…)

- In web flows *NOCC* gives considerable improvement in user perceived latency
- *NOCC* performs significantly better than CC in both HTTP 1.0 and P-HTTP cases