Resource Consumption and Fairness

Protocol Design – S-38.3157

© 2007 Jörg Ott & Carsten Bormann

Resource Consumption

- Execution of protocols consumes resources in
  - End hosts
  - The network
    - Links
    - Network elements
- Where resources are finite:
  - Use them for most important application objectives
  - Use them **productively** (throughput ≠ goodput)
    - Don't perform a protocol step that cannot be completed for lack of other resources
- Control of end system resources: implementation issue
- Control of network resources: shared between hosts and network
  - Internet tenet: Network does not know about application!

© 2007 Jörg Ott & Carsten Bormann

Case study: Internet congestion collapse

- 1988:
  - implementations sent data as they saw fit (BSD UNIX: LAN oriented)
  - Internet usage was growing
  - Where congestion occurred ➔ retransmissions ➔ **more** offered load
    - Previously non-congested links get congested, too
    - Collapse!
  - “Fixes” such as the Nagle algorithm only provided temporary relief
- Issue: How to decide whether to send another packet?
  - Based on the existing network?
  - Based on an upgraded network that provided more information?
  - Based on administrative control (“reserved” capacity)?

Flashback: X.25

- X.25 performed flow control per connection
  - Congestion: “Back-pressure” to previous network elements
  - Credit-based algorithm for per-connection network element buffer management
- Why not do this for IP as well?
  - Network would have to know about application layer connections
  - Large systems: Hard to control phasing/oscillation effects
- IP “back pressure”: ICMP Source quench
  - Send back information about congestion to source
  - “Request to slow down”
  - Issues:
    - Never quantitatively defined
    - Reverse congestion causes loss of source quench ➔ instability!

Van Jacobson's 1988 SIGCOMM paper

- Congestion information has to be carried forward to receiver and "reflected back"
- Receiver is already sending ACKs for packets that have arrived intact
- Remaining "small problem":
  - How to make use of that information in an effective control regime
- (you know the rest)

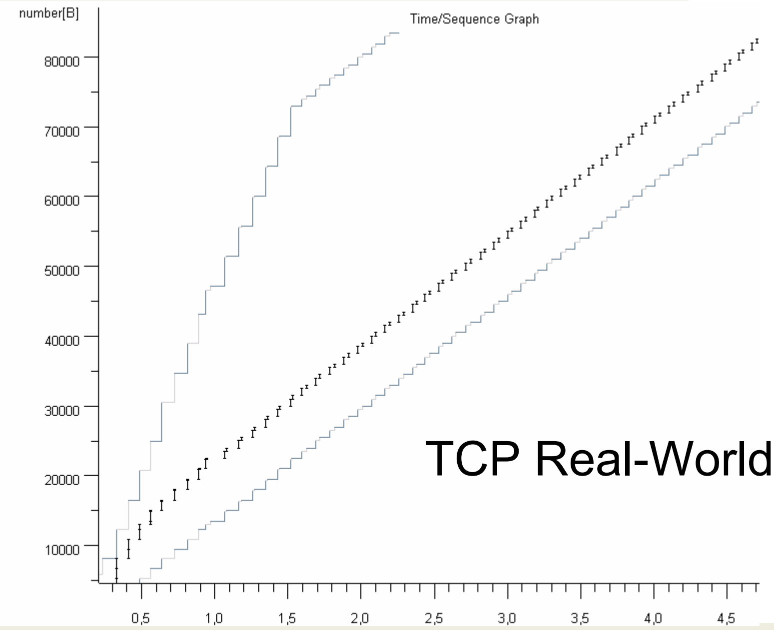
© 2007 Jörg Ott & Carsten Bormann

TCP Optimizations

- Fast retransmission
  - using only timeouts leads to long idle times
    - implementations can't estimate RTT that quickly (or don't at all)
  - receivers react to segments received out-of-order
    - acknowledge last correctly received segment again
    - keep out-of-order segments
  - sender acts upon reception of three duplicate acks
    - retransmit first non-acknowledged segment
    - probably fills the (single segment) gap at the receiver
- Fast recovery
  - duplicate ACKs indicate that most packets do get through (no timeout)
  - ➲ cut congestion window in half (instead of setting to 1 MSS)
  - ➲ don't slow start

© 2007 Jörg Ott & Carsten Bormann

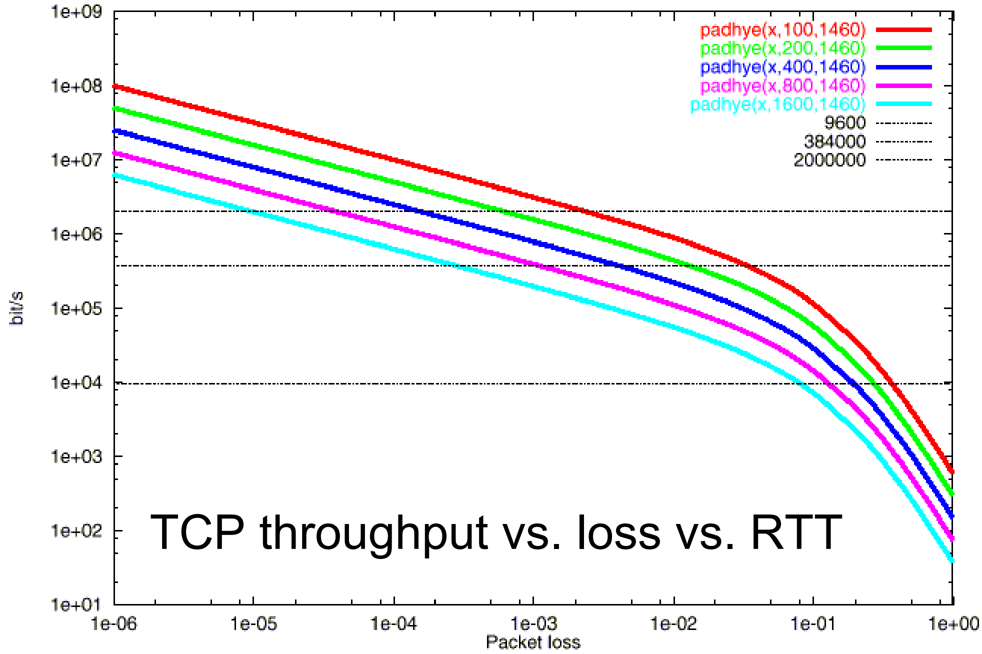
TCP Real-World Example



TCP Throughput Formula

- TCP throughput = f(RTT, MSS, p)
- Floyd approximation: Bit rate ~ $\frac{1}{\sqrt{p} \times RTT}$
- Padhye equation (b is implementation constant, usually 2)

$$B(p) \approx mss \cdot \min\left(\frac{W_{\max}}{RTT}, \frac{1}{RTT\sqrt{\frac{2bp}{3}} + T_0 \min\left(1, 3\sqrt{\frac{3bp}{8}}\right) p\left(1 + 32p^2\right)} \right)$$



TCP Congestion Control Summary

- TCP’s additional algorithms control transmission rate
  - Quickly respond to packet losses in the network (AIMD)
  - Optionally, may take advance measures if increasing RTT is observed
- TCP sender responds to incoming ACKs
  - Initiates transmissions or fast transmissions
  - “ACK Clocking”
  - Timeouts only used in rare circumstances if no packets get through
- Resulting transmission rate approximated by Padhye equation
- Measure for TCP fairness in the network
  - Fair sharing among TCP, UDP (e.g. RTP), and other flows
- CC may also be implemented as rate-based algorithm
  - E.g. TCP-friendly Rate Control (TFRC)

TCP-Friendly Rate Control (TFRC)

- Rate calculated at the sender
  - Based on a slight simplification of the Padhye equation
  - Closed loop algorithm
- Assumptions and Features
  - Usable only for streams with roughly constant packet size
  - Smoother reaction to congestion (does not half the rate upon loss)
  - Applicable to e.g. audio / video traffic
  - Not generally recommended for plain bulk data transfer
- Receiver provides feedback about loss event rate (p) and RTT
  - Provided about once per RTT (unless fewer data is sent)
  - Includes Explicit Congestion Notification (ECN) as observed loss
- Sender adjusts transmission rate according to feedback

© 2007 Jörg Ott & Carsten Bormann

Basic TFRC Operation

- Sender
  - Sends DATA packets
    - Sequence number, time stamp, current RTT estimate
  - Measures RTT from received feedback
    - Calculates weighted moving average
  - Calculates sending rate from received feedback
  - Adjusts transmission rate based upon feedback
    - Cuts rate in half if no feedback received for 2*RTT
    - Rate increase limited to factor 2 per RTT
- Receiver
  - Receives data packets, observes timing, losses
  - Aggregates individual losses per RTT into loss events
  - Return feedback frequently to sender
    - Sequence number, sender + reception timestamp (adjusted according to local delay)
    - Weighted packet loss event rate p

© 2007 Jörg Ott & Carsten Bormann

Congestion Control without Reliability

- Rate-based congestion control using TFRC
  - Requires regular and timely feedback from receivers: order of once per RTT
- Example: Datagram Congestion Control Protocol (DCCP)
  - Non-reliable transport protocol supporting congestion control
    - Was meant to address congestion-unaware UDP applications
  - No fixed congestion control scheme: uses pluggable modules instead
    - CCID2: TCP-like, CCID3: TFRC
- Example: TFRC Profile for RTP
  - Needs RTT readings at both sender and receiver
    - Sender calculates RTT and informs receiver
  - Introduces non-backward-compatible extensions to RTP and RTCP
    - Mechanisms based upon AVP Feedback profile (AVPF)
    - Increased feedback rate (once per RTT)
    - Current idea: smaller RTCP packets (reduce the number of compound packets)

Application Layer Congestion Control

- End-to-end principle: transport layer has only partial knowledge
- Typical assumption: continuous data transfer (“big file”)
- Unrealistic for many applications
  - Short message exchanges
  - Transport does not know much data outstanding for transmission
- Application semantics may implicitly provide congestion control
  - Lock-step protocols
    - SIP transactions in a dialog, e.g., MESSAGE exchanges
    - TFTP
    - Original NFS
- Anything beyond simple lockstep requires careful thought
  - So: Don’t try this at home: difficult to get it right…

What if we **can** modify the network?

But why bother (what are the remaining problems)?

- At low throughput, congestion signalling is wasteful (all the dropped packets are non-productive)
  - ECN: Explicit Congestion Notification
- At high throughput, the signalling rate is low ➔ slow convergence
  - Start at a higher rate (initial window ≅ 4 KB, RFC 3390)
  - Get more information about the path from the outset ("Quickstart")
- A congestion loss cannot be distinguished from a corruption loss
  - Corruption losses lead to lower throughput
    - (problem only if the corruption losses are higher than the congestion loss equivalent to the desirable throughput)
  - Hard to fix unless ECN became universally deployed
    - Would remove "emergency exit" packet drop from router's choices

Quickstart

- Idea: Let routers indicate available capacity
- IP option used in TCP handshake indicates sender's desired data rate
  - Each router in the path can reduce rate request to available capacity
  - Field is echoed back in SYN ACK at the transport layer (TCP option)
  - Also: use special Nonce to detect cheating in the receiver's report
- Backwards compatibility:
  Find out if there are non-participating routers in the path
  - Send with random TTL, also send a random "Quickstart TTL"
  - Approving router decrements both IP TTL and "Quickstart TTL"
  - Difference between (reduced) TTLs is echoed back
  - Old routers change the difference ➔ Sender abandons Quickstart
- Sender combines allowed rate with measured RTT to initialize congestion window
  - Sends "Report approved rate" to allow on-path routers to reduce allocation
  - Packets are then sent rate-paced (no ACK clocking available)

Guiding principles behind Quickstart

- Backwards compatibility, allowing gradual introduction
  - Quickstart must only be enabled if all routers agree
    - Pre-Quickstart routers cannot agree, so any old router on path disables Quickstart
  - Special considerations for tunnels and initial packet losses/ECN indications
- No incentive to cheat
  - Without the random nonces, a receiver could lie about
    - The fact that a quickstart rate was approved
    - The actual rate that has been approved
  - Make sure lying does not provide an advantage
- Generally: good idea

Some Issues with QuickStart

- Generally: adds complexity and introduces risks
- Hosts
  - How do you know your transmission rate in advance (what to indicate in the QS request)?
- Routers
  - How to allocate QuickStart shares fairly?
  - How much state to maintain (and much allocation to verify)?
  - How to clean up failed QS state?
    - Routers may no see packets in the reverse direction (path asymmetry)
  - Misconfiguration (or bugs) may lead to increased congestion
- Attacks
  - Hosts can generate many quickstart requests and thus increase router load
  - Hosts can request data rate and never utilize it (and may spoof IP src address)
    - Routers would require state to check
- Deployment
  - Needs to be deployed all the way along the path (= all routers): incentive?
  - Does this go through middleboxes?
  - Security/stability implications

ECN: Explicit Congestion Notification

- Replace packet loss as a signal by a special bit in each packet
  - "This packet would have been lost"
- Backwards compatibility:
  - Sender must indicate ECN capability of transport (ECT)
  - Non-ECT packets are dropped as previously usual
- No incentive to cheat
  - Unmarked packets carry a bit that would be destroyed by marking
    - "ECN Nonce", RFC 3540
  - Receivers echo back checksum (XOR value) of all these bits
    - Lying receiver is detected by sender detecting mismatch in Nonce echo
- Two bits in IP packet, four values:
  - 00 = old (non-ECT)
  - 01, 10 = ECT (the two possible values carry one bit of ECN nonce)
  - 11 = marked by router as "would have been lost" (destroys nonce bit)

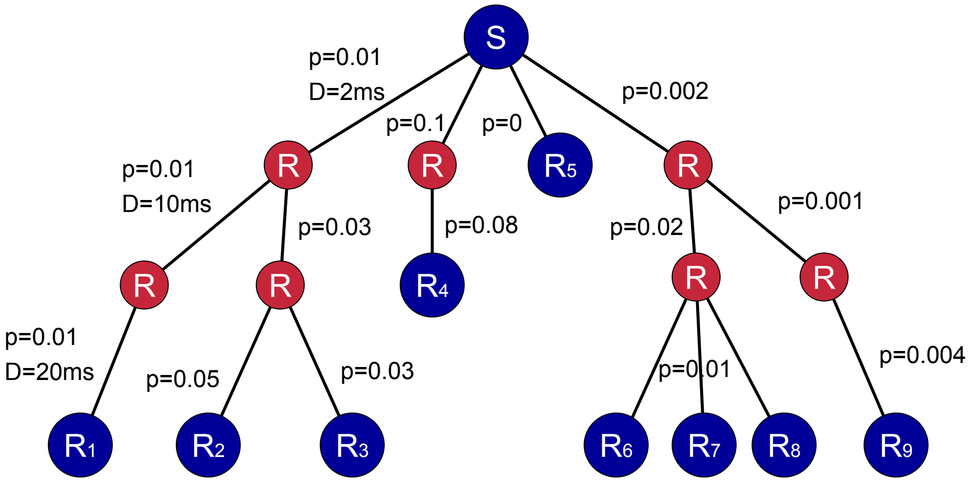
© 2007 Jörg Ott & Carsten Bormann

That was too easy…

- Multicasting
- Security
- Mobility

© 2007 Jörg Ott & Carsten Bormann

Congestion Control and Multicasting



Approaches to Multicast Congestion Control

- Rate-based congestion control based upon feedback
  - E.g. using TFRC mechanism
  - TCP-friendly Multicast Congestion Control (TFMCC) Building Block
  - To be used e.g. with NORM, RTP, …
  - Feedback loop from many receivers to sender
- Window-based mechanisms
  - TCP-style approach
  - Feedback loop from dedicated (possibly changing) receiver to sender
- Layered coding
  - Receiver-based congestion control without feedback loop
  - Receivers use IP multicast JOIN / LEAVE to control their reception rate
- Many of today‘s deployments don‘t use congestion control at all
  - Often deployed in controlled environment using a simple rate control

TFMCC

- Principle operation borrows from TFRC
  - Uses same formula, similar state variables, etc.
- But adjustments for multicast operation needed
  - Control the amount and type of feedback received
  - Distribute workload and amount of state to be maintained
- Receiver-based operation
  - Receivers have unique identifiers
  - Data rate calculations done at receivers X_r
    - Receivers need to measure RTT to sender
    - Send feedback with timestamps, echoed back by sender
  - Packet loss rate calculations as before
  - Feedback suppression scheme for all but worst receiver

© 2007 Jörg Ott & Carsten Bormann

TFMCC (2)

- Sender organizes feedback retrieval into rounds
  - Indicates feedback round number
  - Indicates Current Limiting Receiver (CLR)
  - Sender selects receivers to respond in each round
    - Receivers with measured RTT > MAX_RTT
    - Receivers with calculated rate X_r < X_supp (suppression threshold)
  - Includes reception timestamps for limited number of receivers
    - Enable RTT measurements
- Sender uses feedback to update transmission rate
  - Update CLR based upon feedback from last round
    - Decreases in the transmission rate take effect immediately
    - Also takes into account CLR crashes (e.g. no reports for > 10 RTTs)
  - Cuts transmission rate in half if no report is received for 4 RTTs

© 2007 Jörg Ott & Carsten Bormann

PGMCC

- Uses TCP-style window-based congestion control
- Dynamically determines a receiver for the control loop
  - Selected receiver: "ACKer"
  - Aims to locate the receiver which would have the lowest throughput if there was a TCP connection set up
  - Sender calculates transmission rate for each receiver based upon feedback
    - Using Padhye equation for TCP throughput
  - Chooses ACKer based upon this information
  - ACKer indicates if it is to leave the session
- Feedback from this ACKer control transmission rate
  - Window-based scheme

© 2007 Jörg Ott & Carsten Bormann

Layered Coding

- Multiple "layers" transmit data at different rates
  - Ordering / transmission rates need to be known up front
- Very simple receiver loop
  - JOIN layer n
  - Observe reception rate, packet loss
  - If no packet loss: n=n+1
  - If congestion observed: LEAVE layer n, n=n-1
- Past issue: LEAVE latency (IGMPv1 only)
  - Idea: transmit at constantly reducing data rate on each layer
    - Automatically makes reception rate drop to zero after some time
  - Congestion-free receivers continue JOINing new layers

© 2007 Jörg Ott & Carsten Bormann

Congestion Control and Security

Why would someone want to subvert Fairness?

- Sender: deliver better service (at cost of other senders)
- Network: deliver better service (at cost of other networks)
- Receiver: receive better service (at cost of other receivers)
- Sender: cause damage
- Receiver: cause damage

- Unlikely for large players (detection is almost assured)
  - Essentially rules out senders and network
- Receiver:
  - "TCP optimization software" to receive better service
  - (D)DoS sender by causing it to send to this receiver above fair share

Example: ACK attack

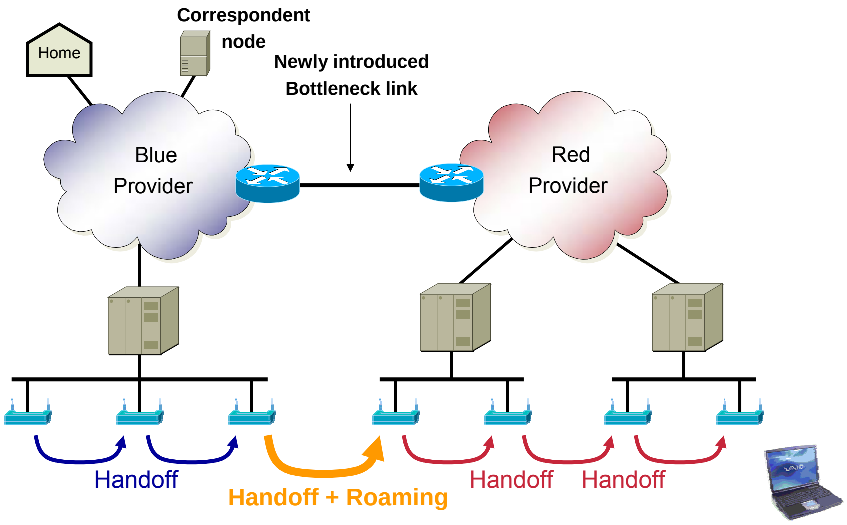
- Observation: TCP ACKs are not protected (no nonce etc.)
- Receiver could ACK everything (even if losses did occur)
  - Sender will ramp up quickly (exponential slow start) until first full RTT is lost
  - Not useful for receiving better service (there will be gaps)
  - Useful for causing damage

- Why doesn't this happen more often?
  - Changing OS's TCP is hard work
  - There are easier angles of attack
  - It's relatively easy to subvert a large number of consumer PCs (➔botnets)
  - It would be much harder to actually change all the various OS versions
  - Phew…

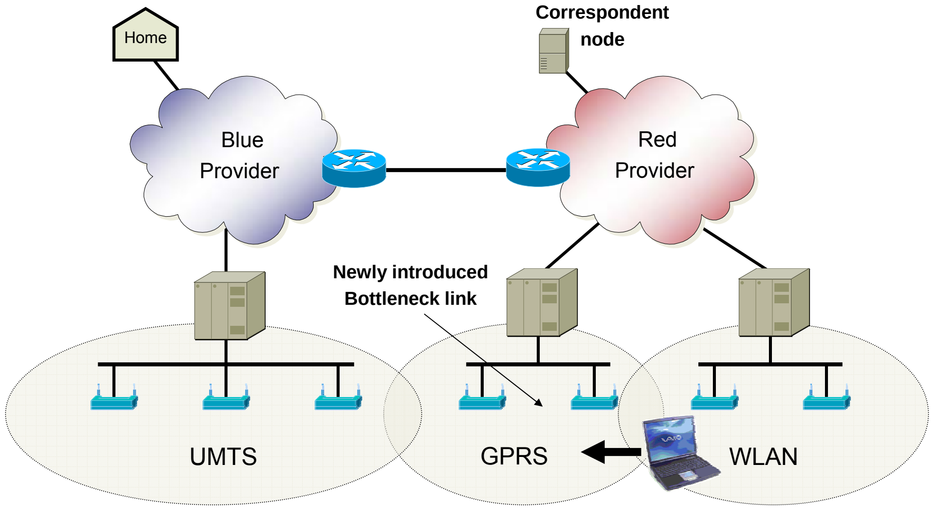
Congestion Control and Mobility

- Path characteristics in the Internet change
  - Filling and emptying queues lead to delay variation
  - Queue length determine congestion-induced packet losses
- Usually changes occur somewhat gradually
  - (relative to RTT)
  - Exception: route changes (rather infrequently, often involve other collateral damage)
- Mobility may lead to more drastic changes
  - Due to mobile IP route optimization
    - From indirect path via home agent to shortest path between CN and MN
  - Due to handover of a mobile node moving between different stub networks
  - Due to a mobile node's switching between different access technologies
- Changes invisible to the transport
  - Present architecture assumes that (mobile) IP and transport layer don't talk
  - Healthy: last hop may not be involved in mobility protocol

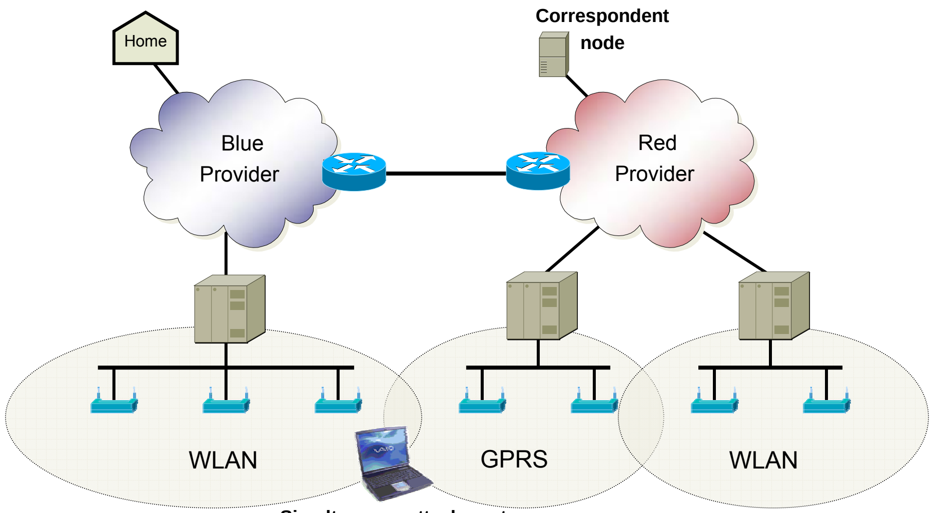
Mobility-induced Changing Path Properties (1)



Mobility-induced Changing Path Properties (2)



Multi-attachment problem



Limitations to Congestion Control

- Elastic applications
  - Typical reference: bulk data transfer
    - Not time critical + always data to send
- Inelastic applications
  - Real-time media streams
  - Limited number of operational points
  - Reducing rate below minimum may equate loss of service
- Example: File download and phone call share DSL access link
- So what does "fairness" really mean here?

© 2007 Jörg Ott & Carsten Bormann

Concluding Thoughts

- Congestion control *is* required
  - May serve the community (and you will get flak if you don't)
  - May improve performance of your own application
- Congestion scale may be limited by the application
  - Choice of (a few) discrete rates only
  - But: minimal QoS cannot be enforced by the application alone

© 2007 Jörg Ott & Carsten Bormann