# Internet Transport Protocols Today and Tomorrow

Michael Welzl

University of Innsbruck

# Outline

Focus on IETF standards!

**Note: only layer 4 TCP/IP technology**
**NOT layers below with all their influential factors!**

1. Internet transport today
    1. Overview
    2. UDP, TCP

2. Internet transport tomorrow
    1. SCTP
    2. UDP Lite
    3. DCCP

3. Example research effort: Tailor-made Congestion Control

# Internet Transport Today

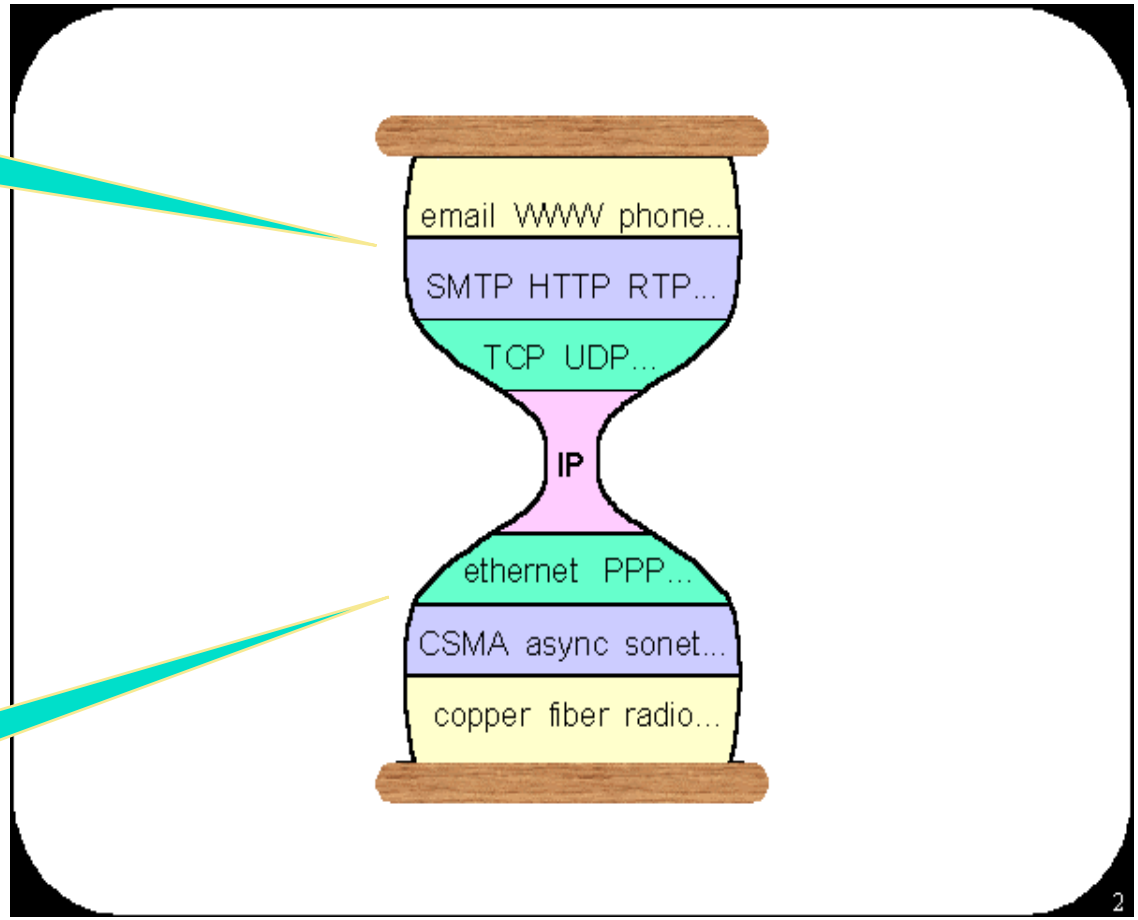
## Overview, TCP and UDP

# A shaky invariant: the Internet Hourglass

Everything Over IP

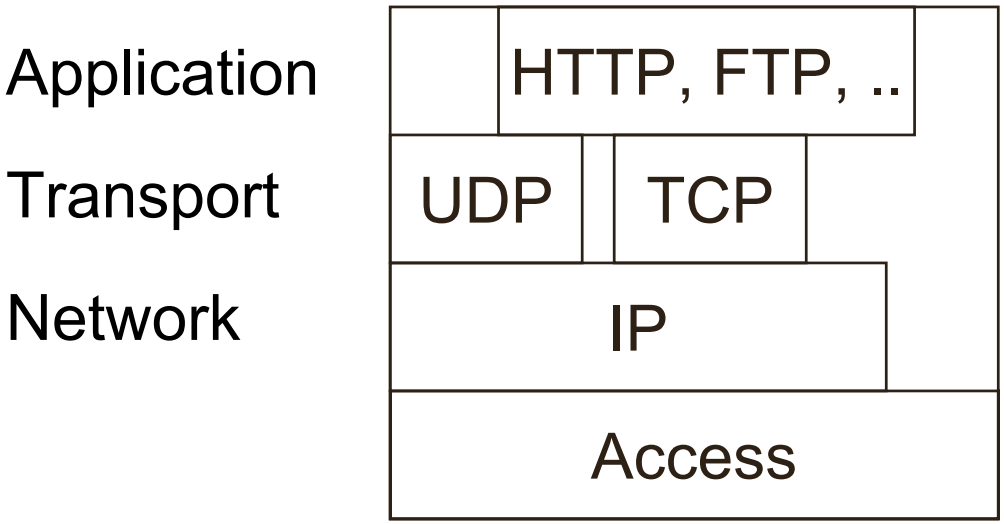
No assumptions
⇒ no guarantees!

IP Over Everything



# Bird's eye view of current TCP/IP stack

- IP: addressing, routing, fragmentation/reassembly, TTL
- UDP: ports, checksum
- TCP: UDP + lots of additional features



# Transport today: one size fits all

- UDP used for sporadic messages (DNS) and some special apps
- TCP used for everything else
  - now approximately 83 % according to:
    *Marina Fomenkov, Ken Keys, David Moore and k claffy, "Longitudinal study of Internet traffic in 1998-2003", CAIDA technical report, available from http://www.caida.org/outreach/papers/2003/nlanr/*
    - backbone measurement from 2000 said 98% ⇒ UDP usage growing
- Still, basically it's

  IP over everything, everything over TCP

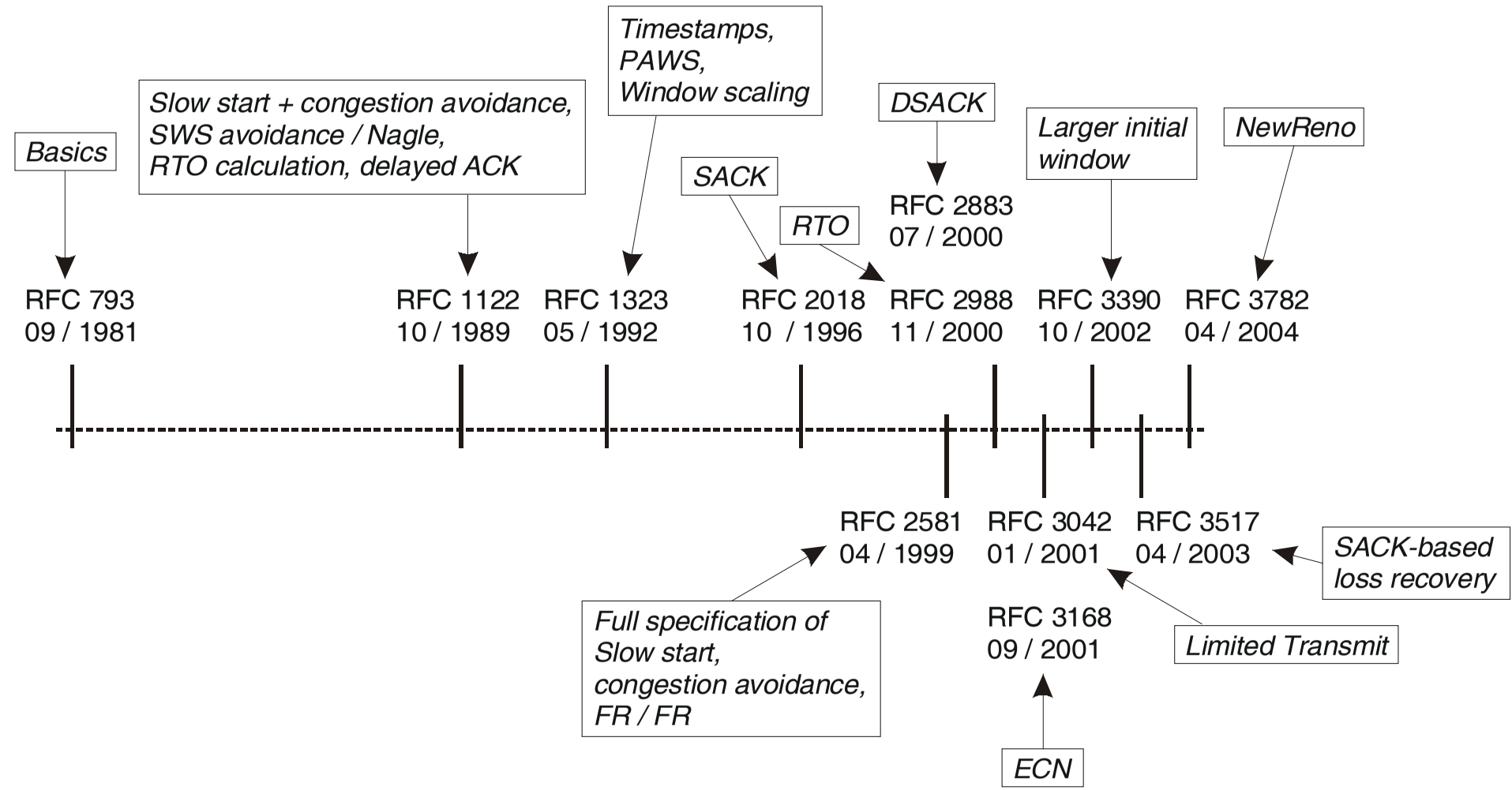
- Question: are all the features always appropriate?

# What TCP does for you (roughly)

- UDP features: multiplexing + protection against corruption
  - ports, checksum
- stream-based in-order delivery
  - segments are ordered according to sequence numbers
  - only consecutive bytes are delivered
- reliability
  - missing segments are detected (ACK is missing) and retransmitted
- flow control
  - receiver is protected against overload (window based)
- congestion control
  - network is protected against overload (window based)
  - protocol tries to fill available capacity
- connection handling
  - explicit establishment + teardown
- full-duplex communication
  - e.g., an ACK can be a data segment at the same time (piggybacking)

# TCP History

Standards track TCP RFCs which influence when a packet is sent



# Internet Transport Tomorrow

## SCTP, UDP Lite, DCCP

# Stream Control Transmission Protocol (SCTP)

# Motivation

- TCP, UDP do not satisfy all application needs
- SCTP evolved from work on IP telephony signaling
  - Proposed IETF standard (RFC 2960)
  - Like TCP, it provides reliable, full-duplex connections
  - Unlike TCP and UDP, it offers new delivery options that are particularly desirable for telephony signaling and multimedia applications
- TCP + features
  - Congestion control similar; some optional mechanisms mandatory
  - Two basic types of enhancements:
    - performance
    - robustness

# Overview of services and features

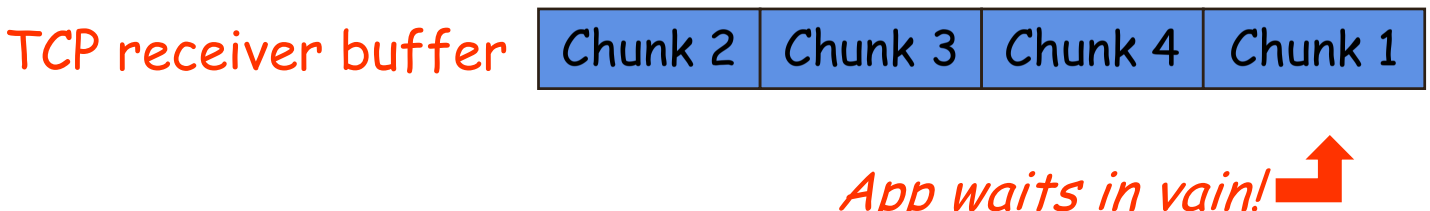
| Services/Features | SCTP | TCP | UDP |
|---|---|---|---|
| • Full-duplex data transmission | yes | yes | yes |
| • Connection-oriented | yes | yes | no |
| • Reliable data transfer | yes | yes | no |
| • Partially reliable data transfer | optional | yes | no |
| • Ordered data delivery | yes | yes | no |
| • Unordered data delivery | yes | no | yes |
| • Flow and Congestion Control | yes | yes | no |
| • ECN support | yes | yes | no |
| • Selective acks | yes | optional | no |
| • Preservation of message boundaries | yes | no | yes |
| • PMTUD | yes | yes | no |
| • Application data fragmentation | yes | yes | no |
| • Multistreaming | yes | no | no |
| • Multihoming | yes | no | no |
| • Protection agains SYN flooding attack | yes | no | n/a |
| • Half-closed connections | no | yes | n/a |

# Packet format

- Unlike TCP, SCTP provides message-oriented data delivery service
  - key enabler for performance enhancements
- Common header; three basic functions:
  - Source and destination ports together with the IP addresses
  - Verification tag
  - Checksum: CRC-32 instead of Adler-32
- followed by one or more chunks
  - chunk header that identifies length, type, and any special flags
  - concatenated building blocks containg either control or data information
  - control chunks transfer information needed for association (connection) functionality and data chunks carry application layer data.
  - Current spec: 14 different Control Chunks for association establishment, termination, ACK, destination failure recovery, ECN, and error reporting
- Packet can contain several different chunk types (extensible design)

# Performance enhancements

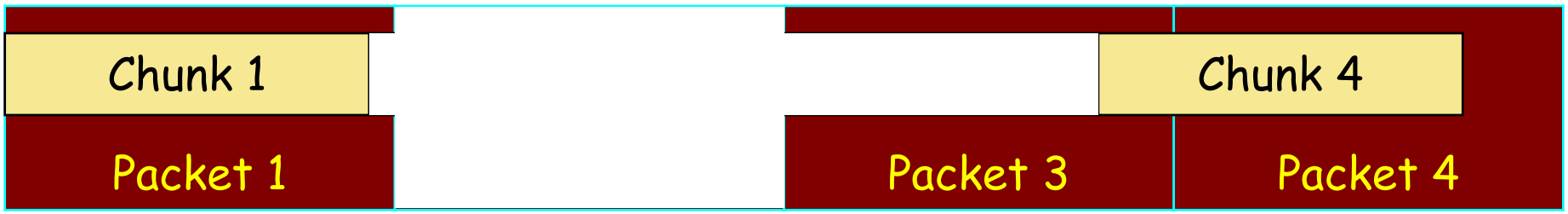
- Decoupling of reliable and ordered delivery
  - Unordered delivery: eliminate head-of-line blocking delay




- Application Level Framing
- Support for multiple data streams (per-stream ordered delivery)
  - Stream sequence number (SSN) preserves order *within* streams
  - no order preserved *between* streams
  - per-stream flow control, per-association congestion control

# Application Level Framing

- TCP: byte stream oriented protocol
- Application may want logical data units (“chunks“)
- Byte stream inefficient when packets are lost



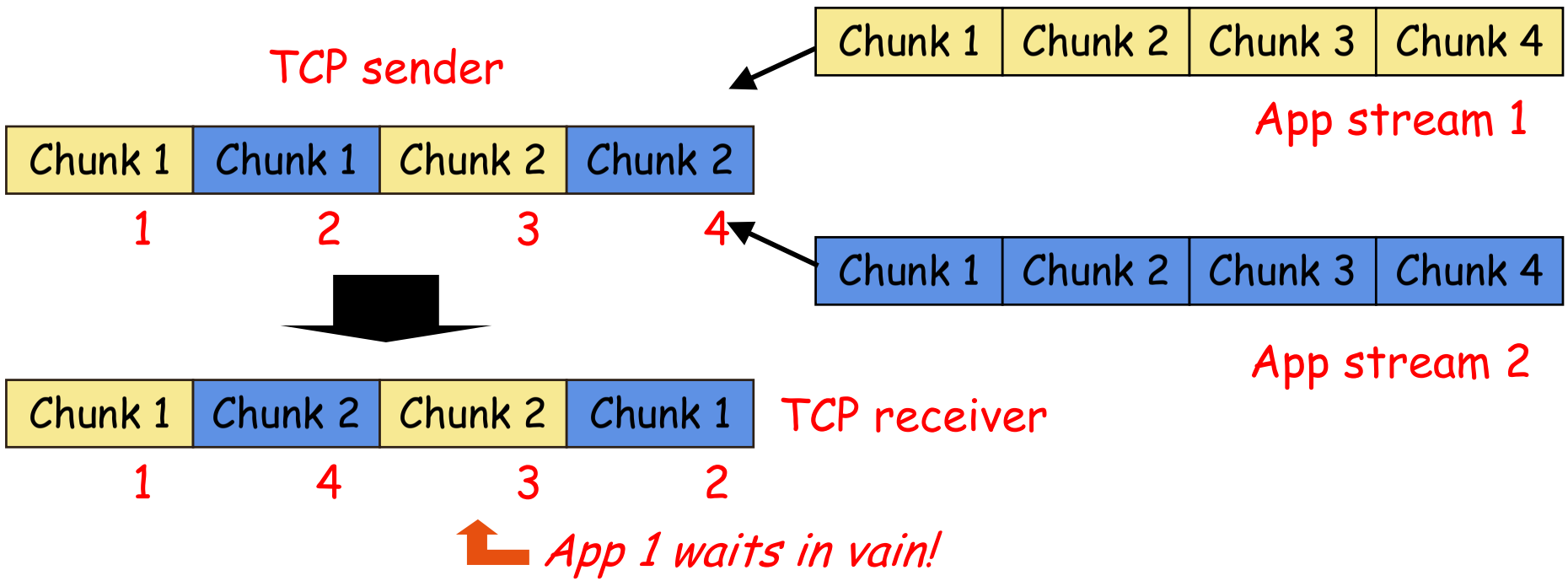

- ALF: app chooses packet size = chunk size

  packet 2 lost: no unnecessary data in packet 1,
  use chunks 3 and 4 before retrans. 2 arrives

- 1 ADU (Application Data Unit) = multiple chunks -> ALF still more efficient!

# Multiple Data Streams

- Application may use multiple logical data streams
  - e.g. pictures in a web browser
- Common solution: multiple TCP connections
  - separate flow / congestion control (Congestion Manager?)



# Multihoming

...at transport layer! (i.e. transparent for apps, such as FTP)

- TCP connection ⇔ SCTP association
  - 2 IP addresses, 2 port numbers ⇔ 2 sets of IP addresses, 2 port numbers

- Goal: robustness
  - automatically switch hosts upon failure
  - eliminates effect of long routing reconvergence time

- TCP: no guarantee for “keepalive“ messages when connection idle
- SCTP monitors each destination's reachability via ACKs of
  - data chunks
  - heartbeat chunks

- Note: SCTP uses multihoming for redundancy, not for load balancing!

# Association phases

- Association establishment: 4-way handshake
  - Host A sends INIT chunk to Host B, Host B returns INIT-ACK containing a cookie
    - information that only Host B can verify; no memory allocated
  - Host A replies with COOKIE-ECHO chunk; may contain A's first data.
  - Host B checks validity of cookie; association is established

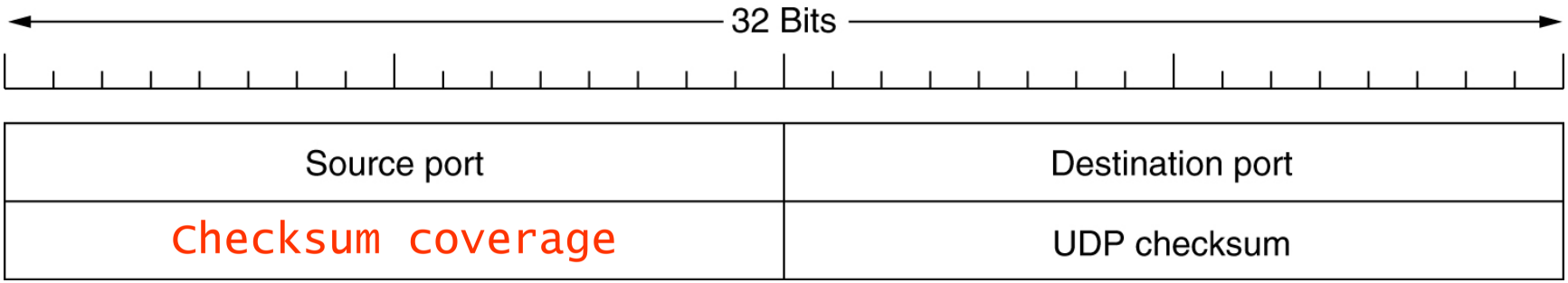
Avoids SYN flood attacks!

- Data transfer
  - SCTP assigns each chunk a unique Transmission Sequence Number (TSN)
  - SCTP peers exchange starting TSN values during association establishment phase
  - Message oriented data delivery; fragmented if larger than destination path MTU
  - Can bundle messages < path MTU into a single packet and unbundle at receiver
  - reliablity through acks, retransmissions, and end-to-end checksum

- Association shutdown: 3-way handshake
  - SHUTDOWN ⇒ SHUTDOWN-ACK ⇒ SHUTDOWN-COMPLETE
  - Does not allow half-closed connections
    (i.e. one end shuts down while other end continues sending new data)

# UDP Lite

# UDP Lite




- Checksum: Adler-32 covering the whole packet
  - UDP: checksum field = 0 ⇒ no checksum at all - bad idea!
- solution: UDP Lite (length := checksum coverage)
  - e.g. video codecs can cope with bit errors, but UDP throws whole packet away!
  - acceptable BER up to applications (complies with end-to-end arguments)
  - some data can be covered by checksum
  - apps can realize several or different checksums
- Issues:
  - apps can depend on lower layers (no more “IP over everything“)
  - authentication requires data integrity - not given with UDP Lite
  - handing over corrupt data is not always efficient - link layer should detect UDP Lite

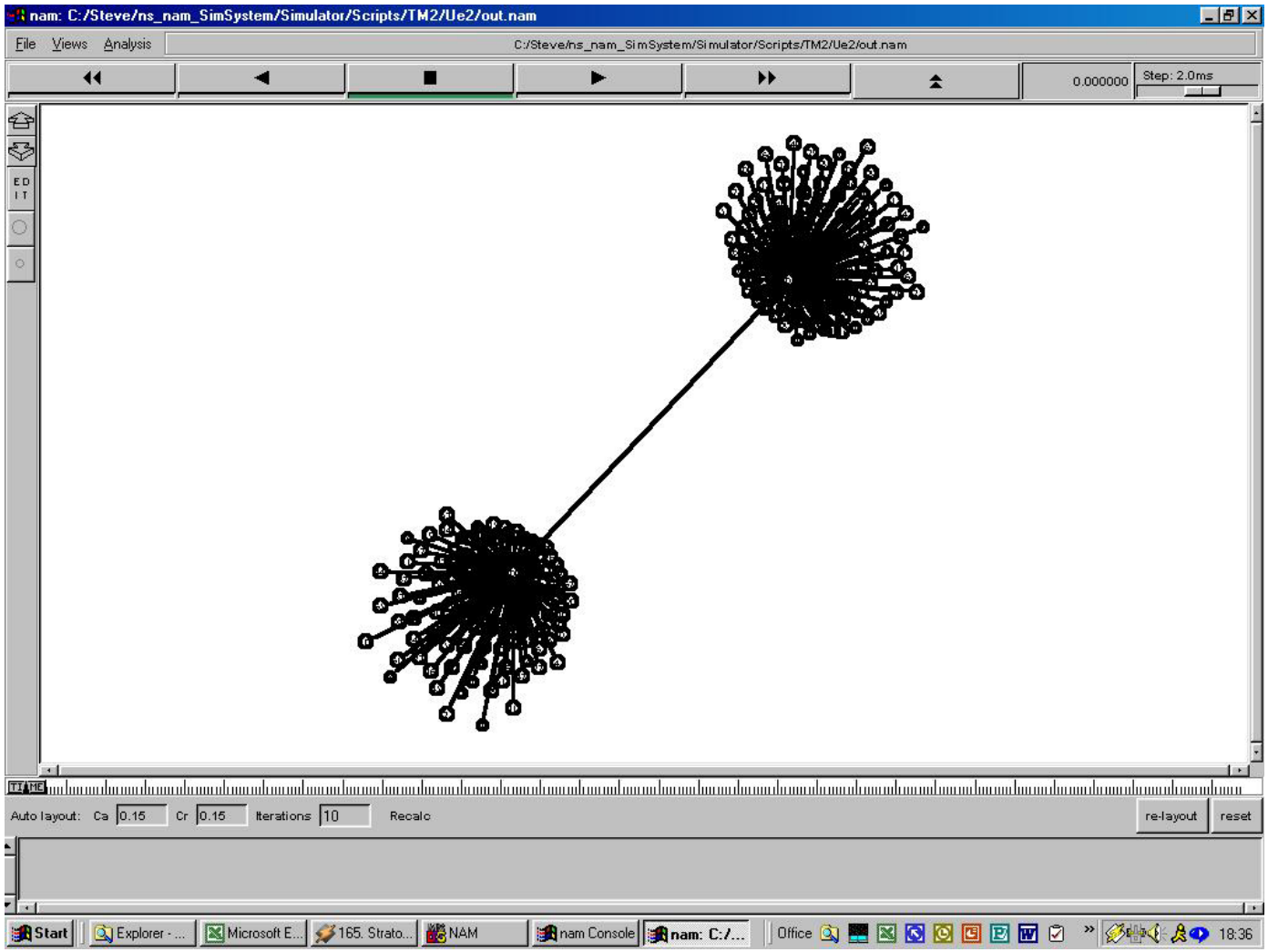
Inter-layer communication problem

# Datagram Congestion Control Protocol (DCCP)

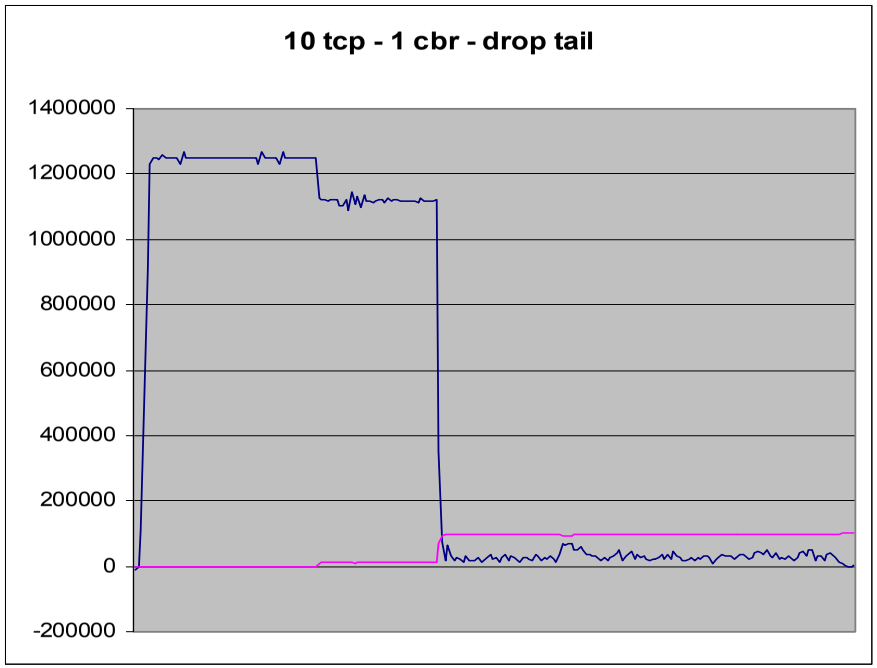
# Motivation

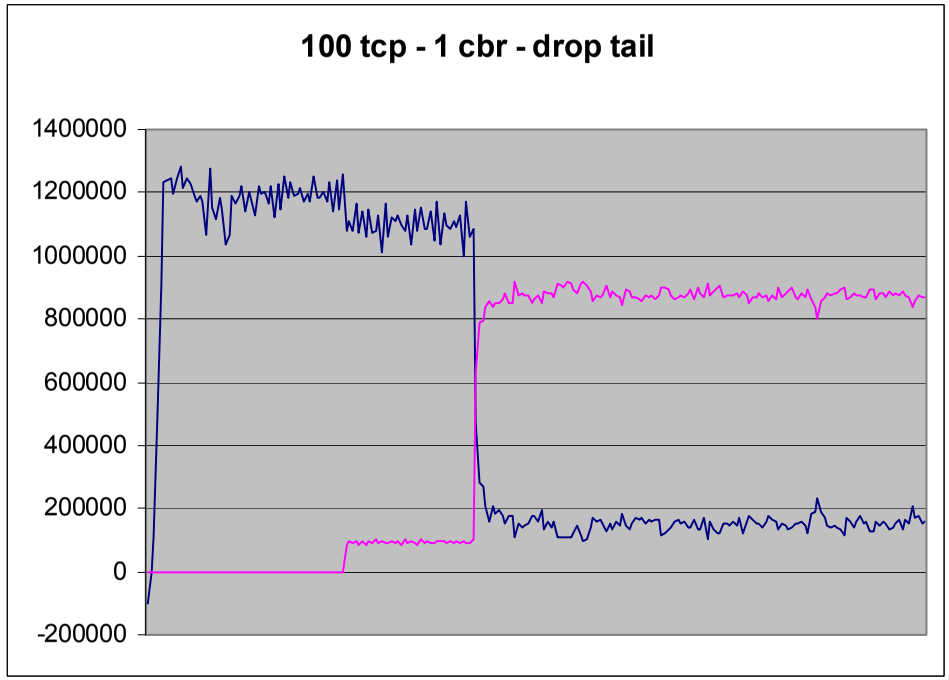
- Some apps want unreliable, timely delivery
  - e.g. VoIP: significant delay = ☹ ... but some noise = ☺
- UDP: no congestion control
- Unresponsive long-lived applications
  - endanger others (congestion collapse)
  - may hinder themselves (queuing delay, loss, ..)
- Implementing congestion control is difficult
  - illustrated by lots of faulty TCP implementations
  - may require precise timers; should be placed in kernel

# TCP vs. UDP: a simple simulation example



# It doesn‘t look good

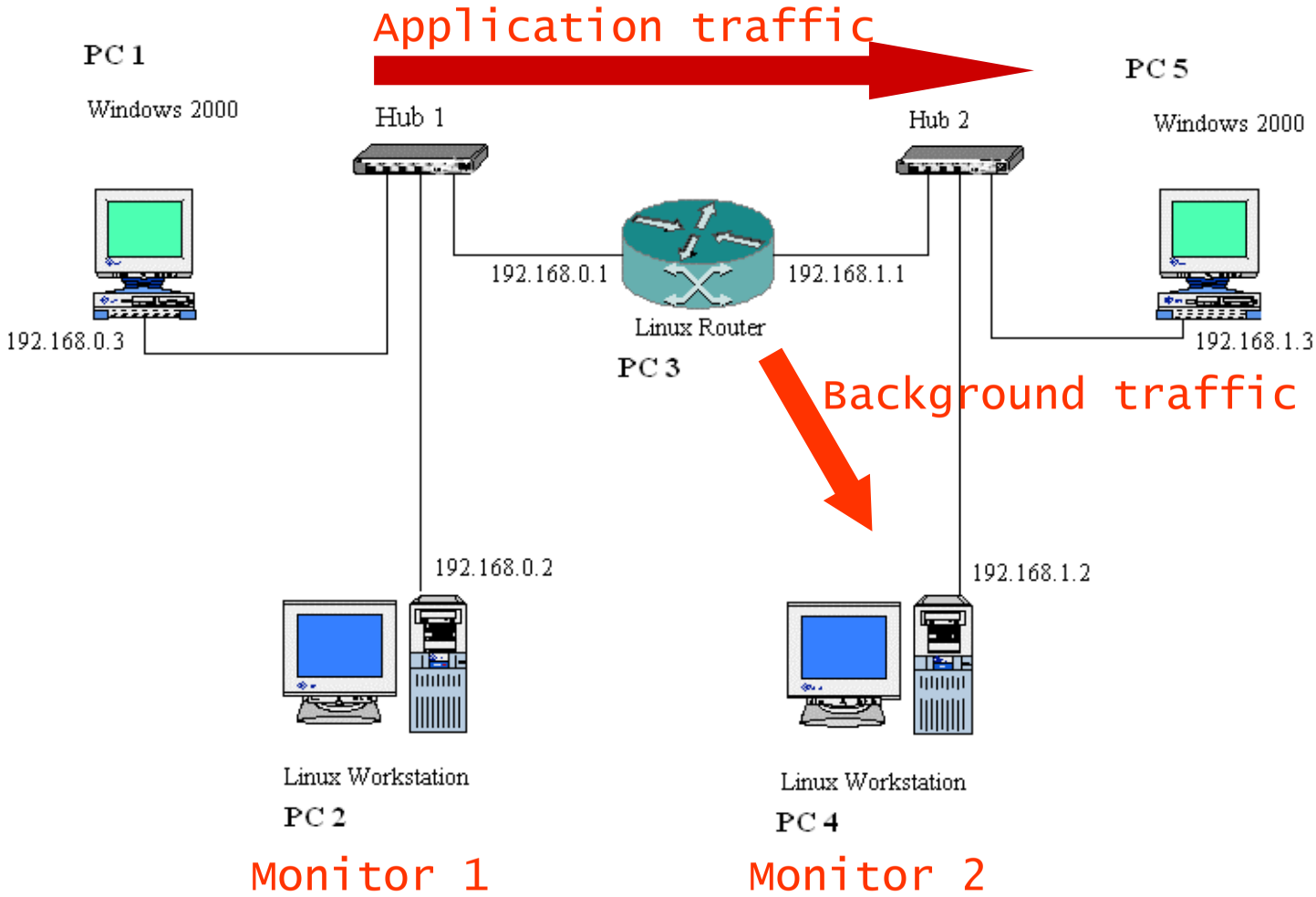


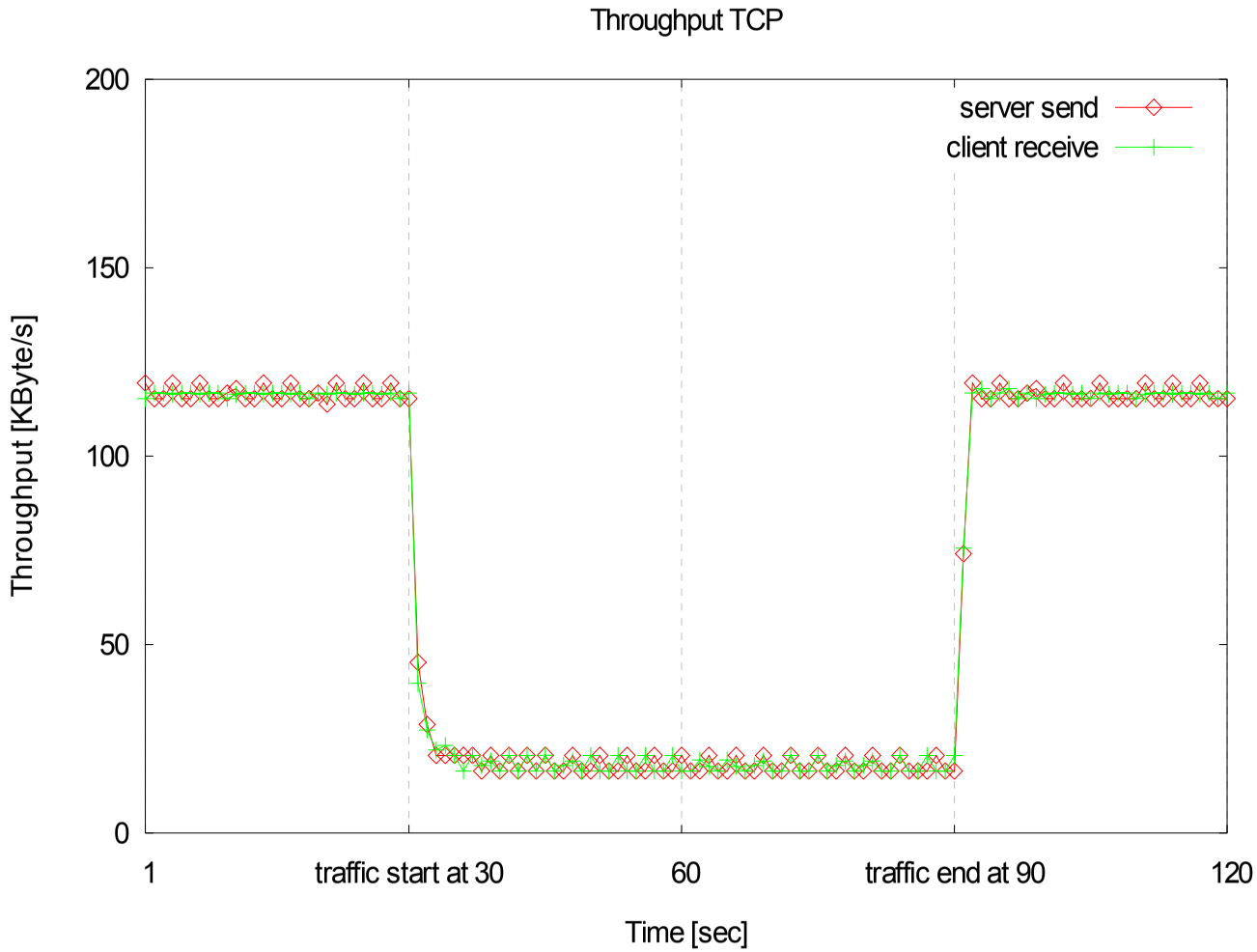

For more details, see:

*Promoting the Use of End-to-End Congestion Control in the Internet.*
*Floyd, S., and Fall, K..*
*IEEE/ACM Transactions on Networking, August 1999.*

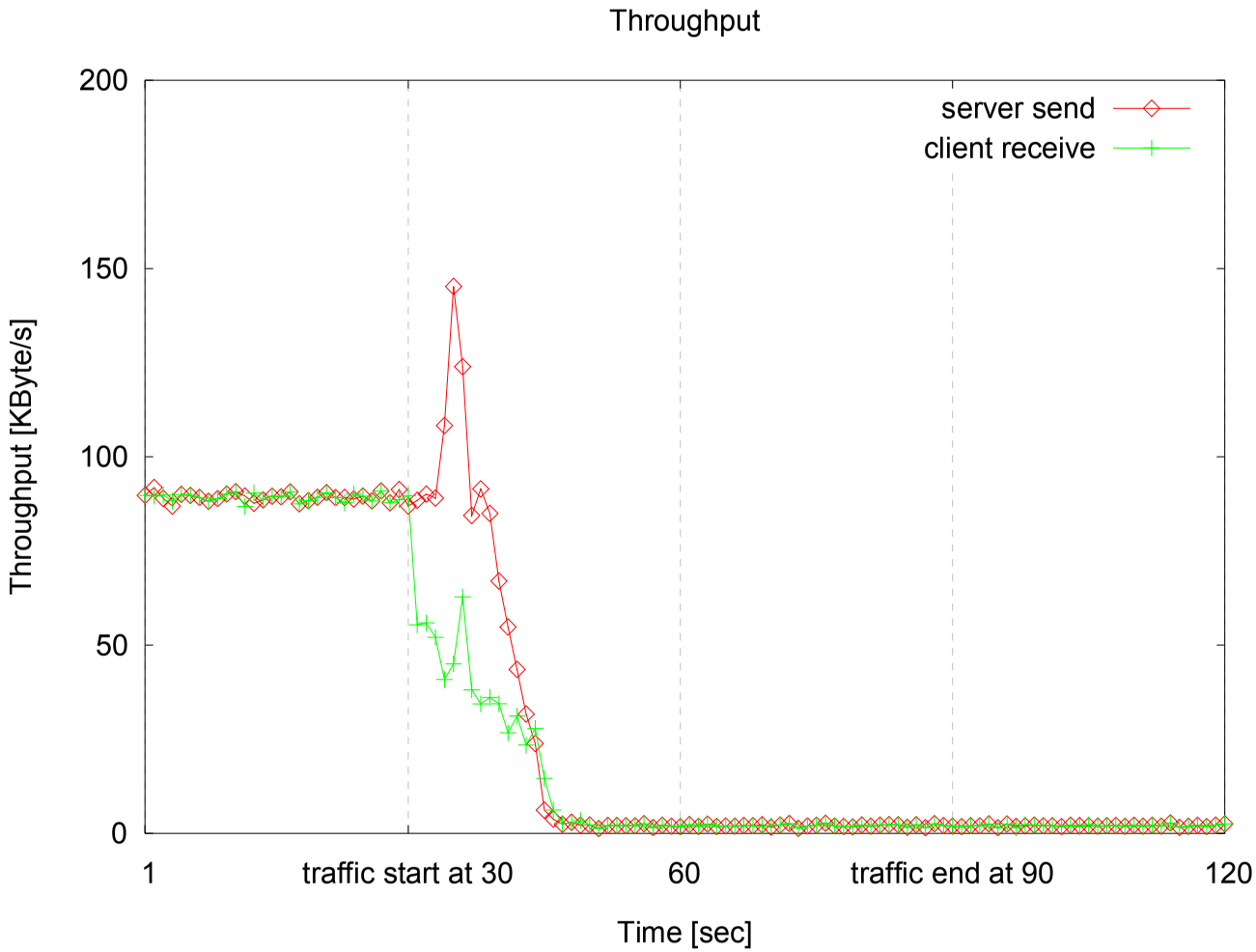
# Real behavior of today‘s apps



# TCP (the way it should be)

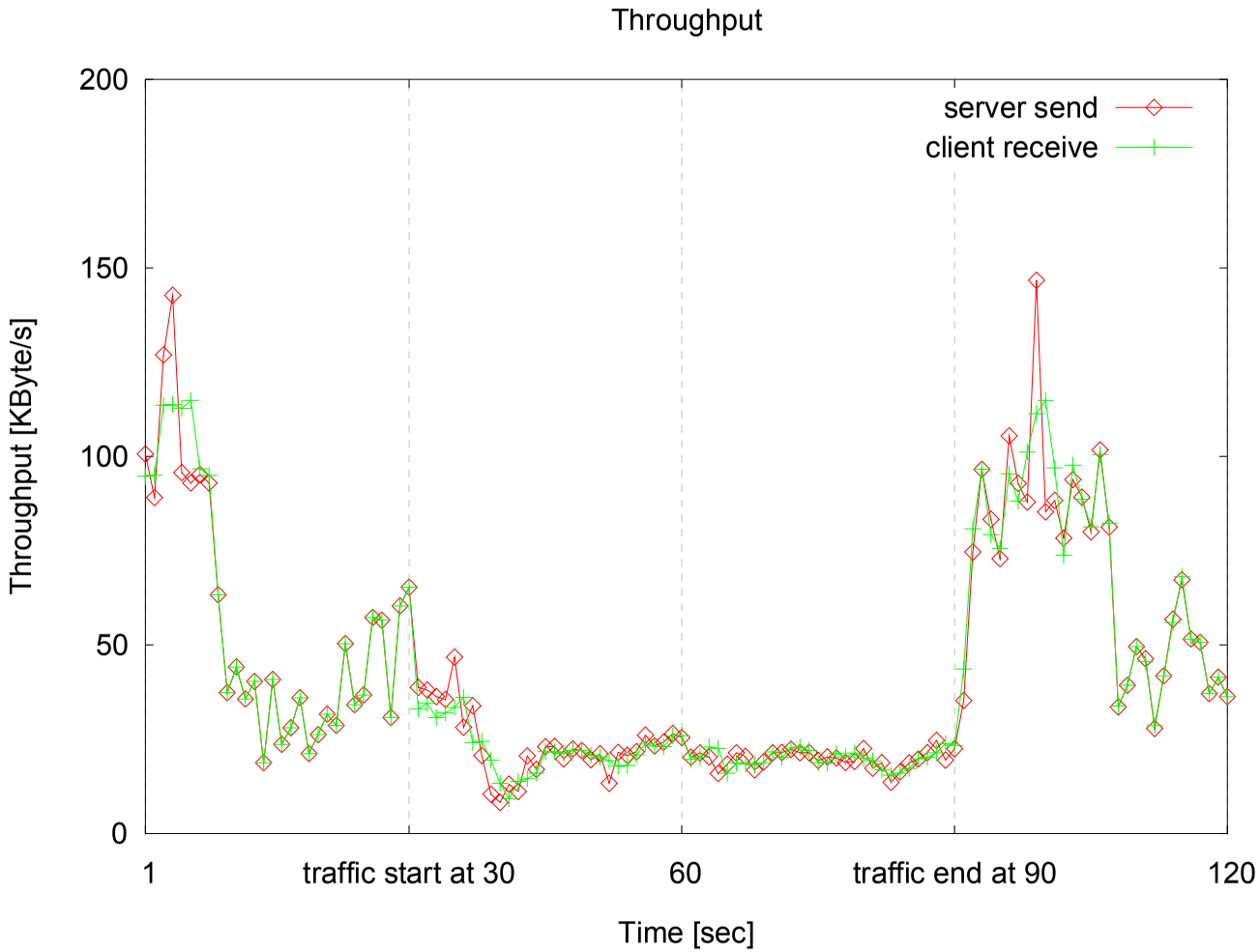


# Streaming Video: RealPlayer



# Streaming Video: Windows Media Player

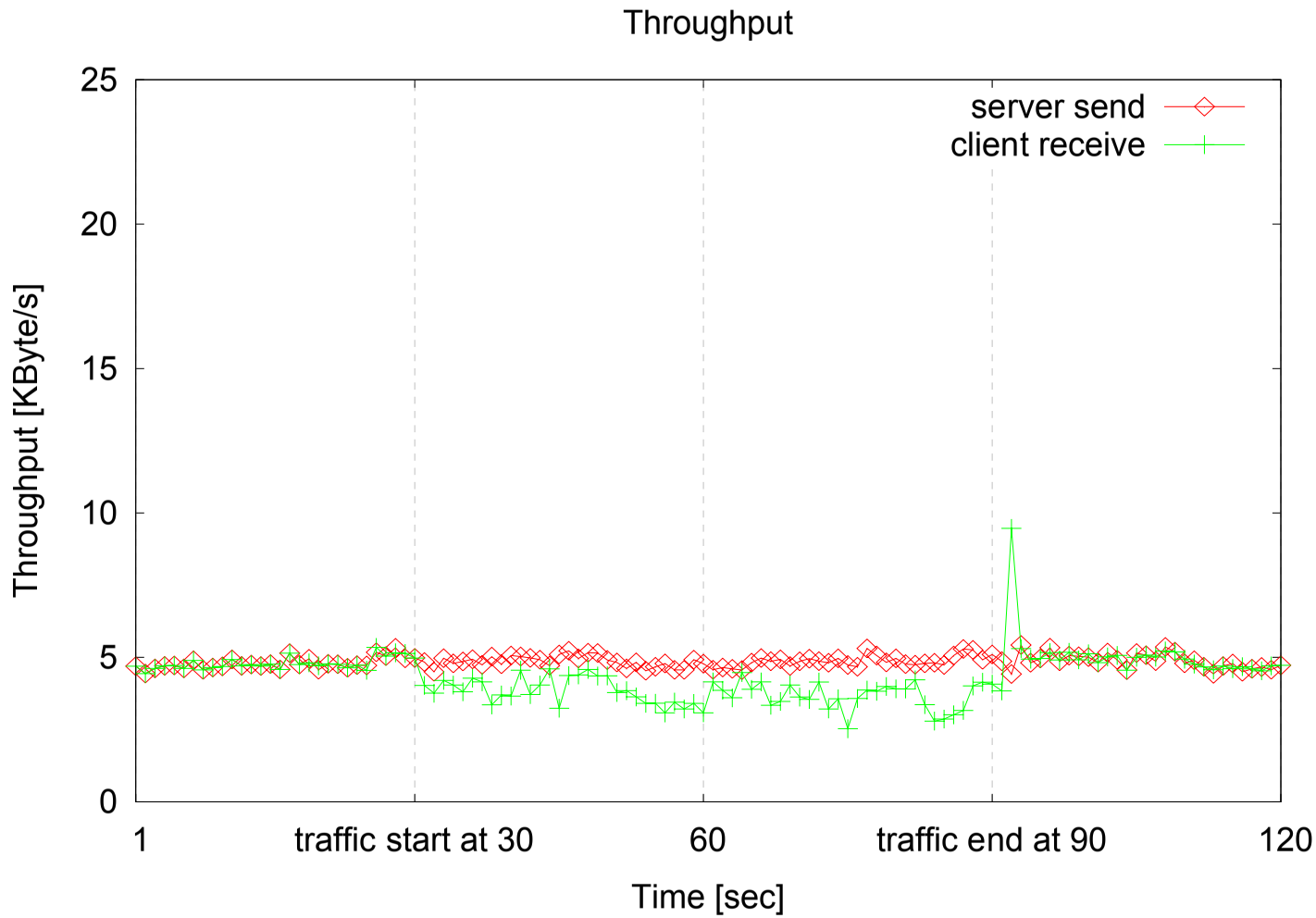
# Streaming Video: Quicktime



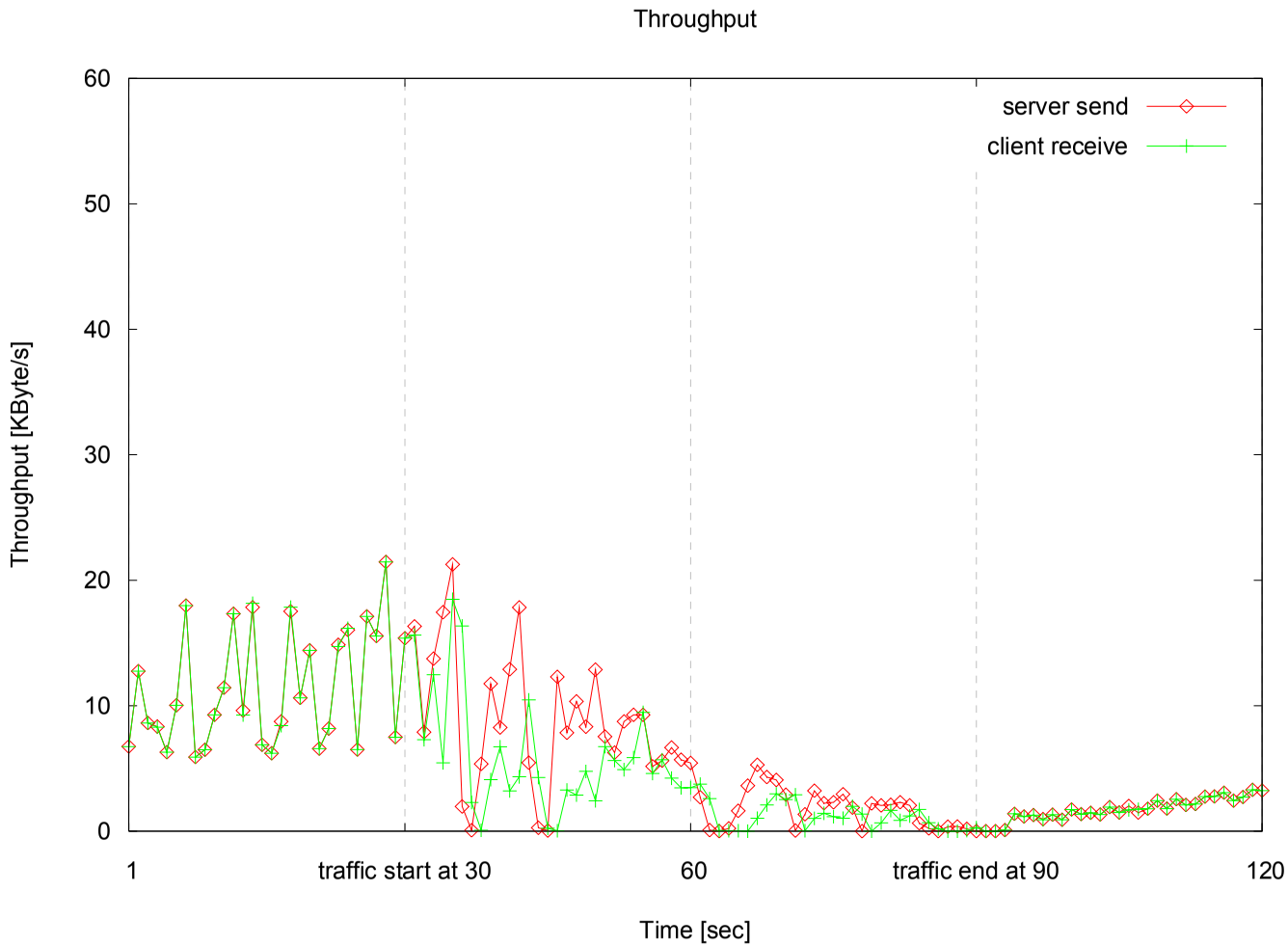
# VoIP: MSN


Throughput
server send
client receive
Throughput [KByte/s]
25
20
15
10
5
0
1
traffic start at 30
60
traffic end at 90
120
Time [sec]

# VoIP: Skype


Throughput
server send
client receive
Throughput [KByte/s]
25
20
15
10
5
0
1
traffic start at 30
60
traffic end at 90
120
Time [sec]

# Video conferencing: iVisit



# Observations

- Several other applications examined
  - ICQ, NetMeeting, AOL Instant Messenger, Roger Wilco, Jedi Knight II, Battlefield 1942, FIFA Football 2004, MotoGP2
- Often: congestion ⇒ increase rate
  - is this FEC?
  - often: rate increased by increasing packet size
  - note: packet size limits measurement granularity
- Many are unreactive
  - Some have quite a low rate, esp. VoIP and games
- Aggregate of unreactive low-rate flows = dangerous!
  - IAB Concerns Regarding Congestion Control for Voice Traffic in the Internet [RFC 3714]

# DCCP fundamentals

- Congestion control for unreliable communication
  - in the OS, where it belongs
- Well-defined framework for [TCP-friendly] mechanisms
- Roughly:

  DCCP = TCP – (bytestream semantics, reliability)
  = UDP + (congestion control with ECN, handshakes, ACKs)

- Main specification does not contain congestion control mechanisms
  - CCID definitions (e.g. TCP-like, TFRC, TFRC for VoIP)
- IETF status: working group, several Internet-drafts, thorough review
  - proposed standard RFC status envisioned

# What DCCP does for you (roughly)

- Multiplexing + protection against corruption
  - ports, checksum (UDP Lite ++)
- Connection setup and teardown
  - even though unreliable! one reason: middlebox traversal
- Feature negotiation mechanism
  - Features are variables such as CCID (“Congestion Control ID“)
- Reliable ACKs ⇒ knowledge about congestion on ACK path
  - ACKs have sequence numbers
  - ACKs are transmitted (receiver) until ACKed by sender (ACKs of ACKs)
- Full duplex communication
  - Each sender/receiver pair is half-connection; can even use different CCIDs!
- Some security mechanisms, several options

# Packet format

2 Variants; different sequence no. length, detection via X flag

| Source Port | | | | Destination Port |
|---|---|---|---|---|
| Data Offset | | CCVal | CsCov | Checksum |
| Res | Type | X = 1 | Reserved | Sequence Number (high bits) |
| Sequence Number (low bits) | | | | |

| Source Port | | | | Destination Port |
|---|---|---|---|---|
| Data Offset | | CCVal | CsCov | Checksum |
| Res | Type | X = 0 | Sequence Number (low bits) | |

- Generic header with 4-bit type field
  - indicates following subheader
  - only one subheader per packet, not several as with SCTP chunks

# Separate header / payload checksums

- Available as “Data Checksum option“ in DCCP
  - Also suggested for TCP, but not (yet?) accepted
  - Note: partial checksums useless in TCP (reliable transmission of erroneous data?)

- Differentiate corruption / congestion
  - Checksum covers all
    - Error could be in header
    - Impossible to notify sender (seqno, ports, ..)
  - Checksum fails in header only
    - Bad luck
  - Checksum fails in payload only, ECN = 0
    - Inform sender of corruption
    - No need to react as if congestion
    - Still react (keeping high rate + high BER = bad idea) ⇒ experimental!
  - Checksum fails in payload only, ECN = 1
    - Clear sign of congestion

# Additional options

- Data Dropped: indicate differentdrop events in receiver
  (differentiate: not received by app / not received by stack)
  - removed from buffer because receiver is too slow
  - received but unusable because corrupt (Data Checksum option)
- Slow receiver: simple flow control
- ACK vector: SACK (runlength encoded)
- Init Cookie: protection against SYN floods
- Timestamp, Elapsed Time: RTT estimation aids
- Mandatory: next option must be supported
- Feature negotiation: Change L/R, Confirm L/R

# DCCP usage: incentive considerations

- DCCP benefits (perspective of a single application programmer)
  - ECN usage (not available in UDP API)
  - scalability in case of client-server based usage
  - TCP-based applications that are used at the same time may work better
  - perhaps smaller loss ratio while maintaining reasonable throughput
- Reasons not to use DCCP
  - programming effort, especially if it is an update to a working UDP based application
  - common deployment problems of new protocol with firewalls etc.
  - less total throughput than UDP
- What if dramatically better performance than UDP is required?
- Can be attained using “penalty boxes“ - but:
  - requires such boxes to be widely used
  - will only happen if beneficial for ISP: financial loss from UDP unresponsive traffic > financial loss from customers whose UDP app doesn't work anymore
  - requires many apps to use DCCP
  - chicken-egg problem! Similar to QoS deployment towards end systems [RFC 2990]


FWF

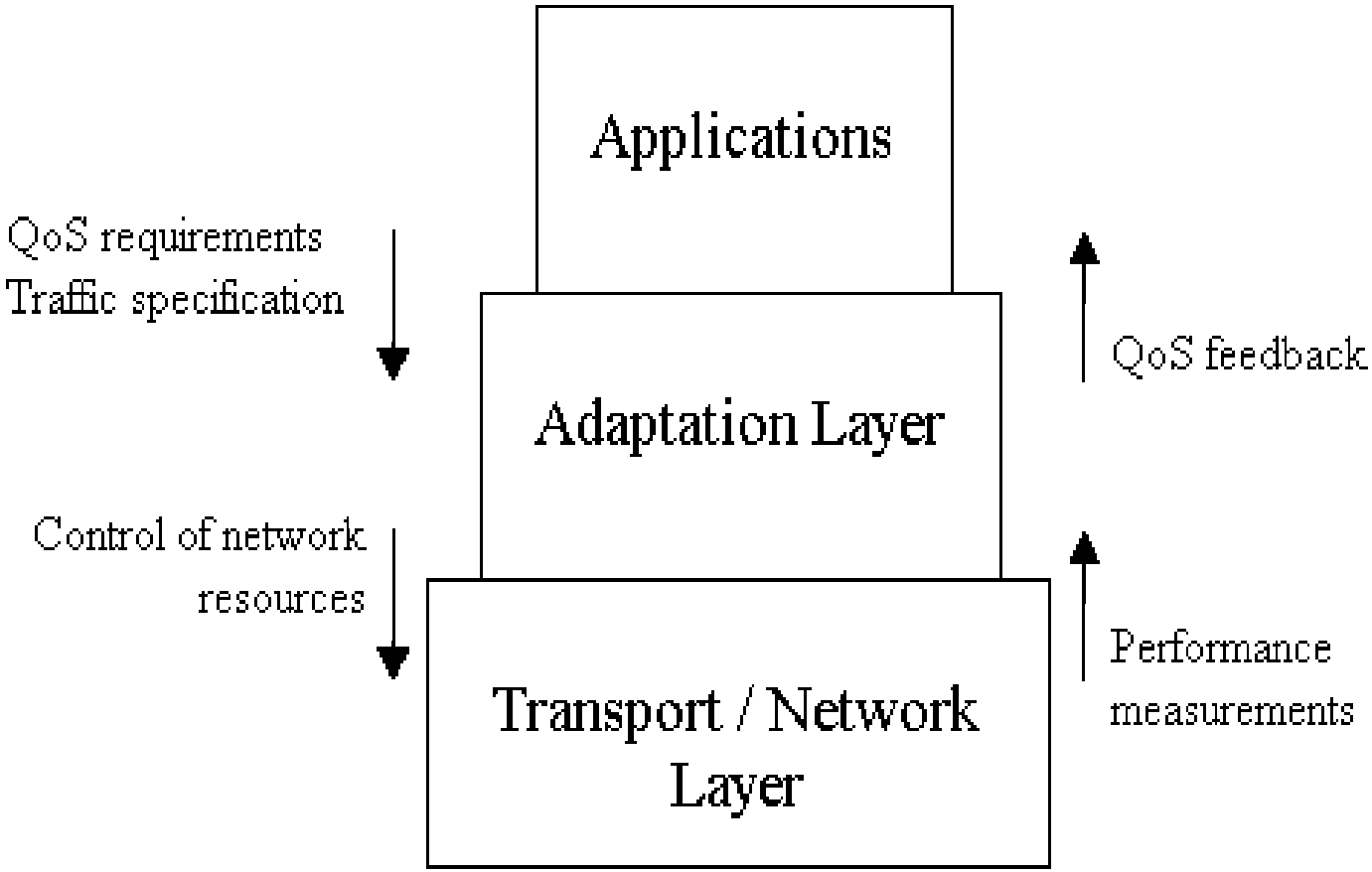
# Tailor-made Congestion Control

A research project at the
University of Innsbruck

# Two Internet deployment problems

- Deployment problem 1: Transport Layer Developments
  - Plethora of mechanisms out there (papers, proof, even code)
  - nobody seems to use them: app level implementation too complex!
  - Soon: TCP+UDP-Lite+SCTP+DCCP .. more complexity in the OS
    - does not *solve*, but *change* the problem: "how to choose the right protocol and parameters?"
- Deployment problem 2: End-to-end QoS
  - We all know it never happened...
  - IntServ/RSVP, DiffServ + SLAs + MPLS, but nothing for end users
  - Internet = too heterogeneous; flexible interface missing!

# *Proposed solution:*
# "Adaptation Layer"



# *Why* we need it

- Application relieved of burden
  - more sophisticated transmission mechanisms possible
  - tailored network usage instead of “one size fits all“ (just UDP / TCP)
- Network provides service - app specifies QoS requirements
  - Adaptation layer makes the most out of available resources
- Adaptation layer provides QoS feedback
  - Information logically closer to application
- Full transparency to application
  - gradual deployment of new transport mechanisms

# *How* it could work: application interface

- from application
  - QoS spec
    - apply weights to QoS parameters
    - goal: tune trade-offs (packet sizes, ..)
    - Examples:
      - reduced delay is more important than high throughput
      - I don‘t care about a smooth rate (I use large buffers)
  - Traffic spec
    - Example: long lasting stream, “greedy“
- to application
  - “video frame complete“ instead of “throughput = ... loss = ... “, ..

# *How* it could work: internals

- Control of network resources
  - Tune packet size
    - maximize throughput + minimize delay according to QoS spec
  - Choose protocol + tune parameters
    - TCP, UDP, but also:
    - DCCP: congestion control for datagrams (connectionless)
      - based on QoS-centric evaluation of mechanisms: RAP, TFRC, TEAR, LDA+, GAIMD, Binomial CC., ..
    - UDP Lite: transmission of erroneous payload
    - SCTP: transport level multihoming, reliable out-of-order transmission
  - Further functions: buffer, bundle streams, ..
    - Example: long-term stream, sporadic interruptions + delay not important ⇒ buffer, don‘t restart CC
- Performance measurements
  - use existing tools + passively monitor flows

# Implications

Pro‘s

- transparency enables apps to use new mechanisms automatically
- new competition for ISPs (reason to deploy QoS)
- possibile to use non-TCP-friendly mechanisms in special environments
- framework serving as a catalyst for new research (like ATM ABR)

Con‘s

- Loss of service granularity
- Difficulty of designing appropriate middleware (app interface, ..)
- Lots of open research issues, e.g.:
  - relationship with Congestion Manager
  - dynamically switching CC. mechanism

# Conclusion

- Internet transport layer growing
- More features, but also more complexity
- We work on this
  - Started September 2004
  - Currently developing on gradual deployment:
    transparently impose congestion control on standard UDP flows
    for the benefit of all; provide UDP interface + optional extras
  - Also, extra focus on Grid applications: how do they use the network,
    what do they need?

# Questions?

# Thank you for your attention!

# References (sources)

- Some pictures / slides from:
  - Max Mühlhäuser, Murtaza Yousaf
  - bachelor students: Muhlis Akdag, Thomas Rammer, Roland Wallnöfer
- IP hourglass picture from:
  - http://www.ietf.org/proceedings/01aug/slides/plenary-1/index.html
- Some content from:
  - Michael Welzl, "Network Congestion Control: Managing Internet Traffic", John Wiley & Sons, Ltd., July 2005
  - Various RFCs / Internet-drafts
- Recommended URLs:
  - http://www.ietf.org
  - http://www.icir.org/kohler/dccp/
  - http://www.sctp.org/