Internet Transport Tomorrow: Introducing SCTP, UDP-Lite and DCCP

Michael Welzl http://www.welzl.at

DPS NSG Team http://dps.uibk.ac.at/nsg
Institute of Computer Science
University of Innsbruck, Austria

CAIA guest talk
Swinburne University
Melbourne AUS
30 January, 2008

Outline

1. Internet transport today: too much, or not enough
2. Internet transport tomorrow
   1. SCTP
   2. UDP-Lite
   3. DCCP

Transport layer problem statement

- Efficient transmission of data streams across the Internet
  - various sources, various destinations, various types of streams
- What is “efficient“?
  - terms: latency, end2end delay, jitter, bandwidth (nominal/available/bottleneck -), throughput, goodput, loss ratio, ..
  - general goals: high throughput (bits / second), low delay, jitter, loss ratio
- Note: Internet = TCP/IP based world-wide network
  - no assumptions about lower layers!
  - ignore CSMA/CD, CSMA/CA, token ring, baseband encoding, frame overhead, switches, etc. etc. !

Internet transport today: one size fits all

- UDP used for sporadic messages (DNS) and some special apps
- TCP used for everything else
  - in 2003, 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
- Original Internet proposition:
  IP over everything, everything over IP
- Today‘s reality:
  IP over everything, almost everything over TCP, and the rest over UDP

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)

Are all these features always appropriate?

UDP, however…

32 Bits

Source port	Destination port
UDP length	UDP checksum

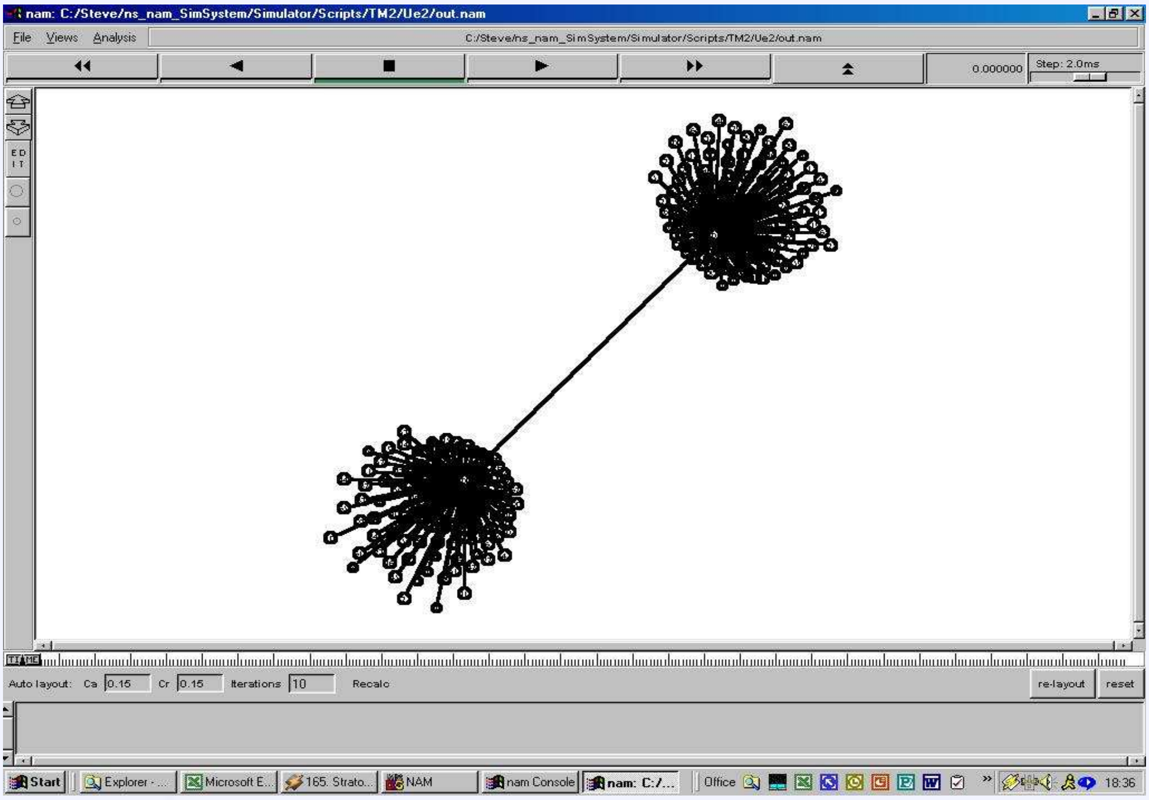
- RFC 768: three pages!

- IP + 2 features:
  - Multiplexing (ports)
  - Checksum

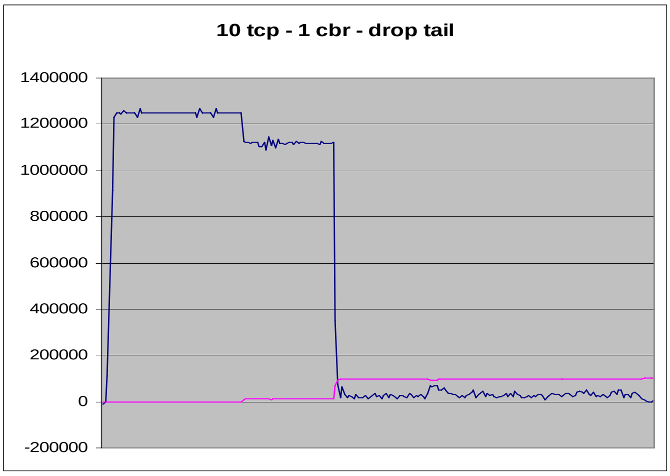
- Used by apps which want unreliable, timely delivery
  - e.g. VoIP: significant delay = ☹ … but some noise = ☺

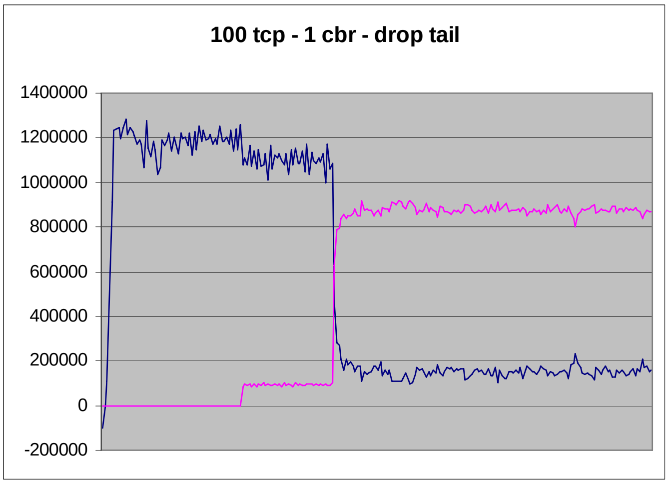
- No congestion control
  - fine for SNMP, DNS, ..

TCP vs. UDP: a simple simulation example



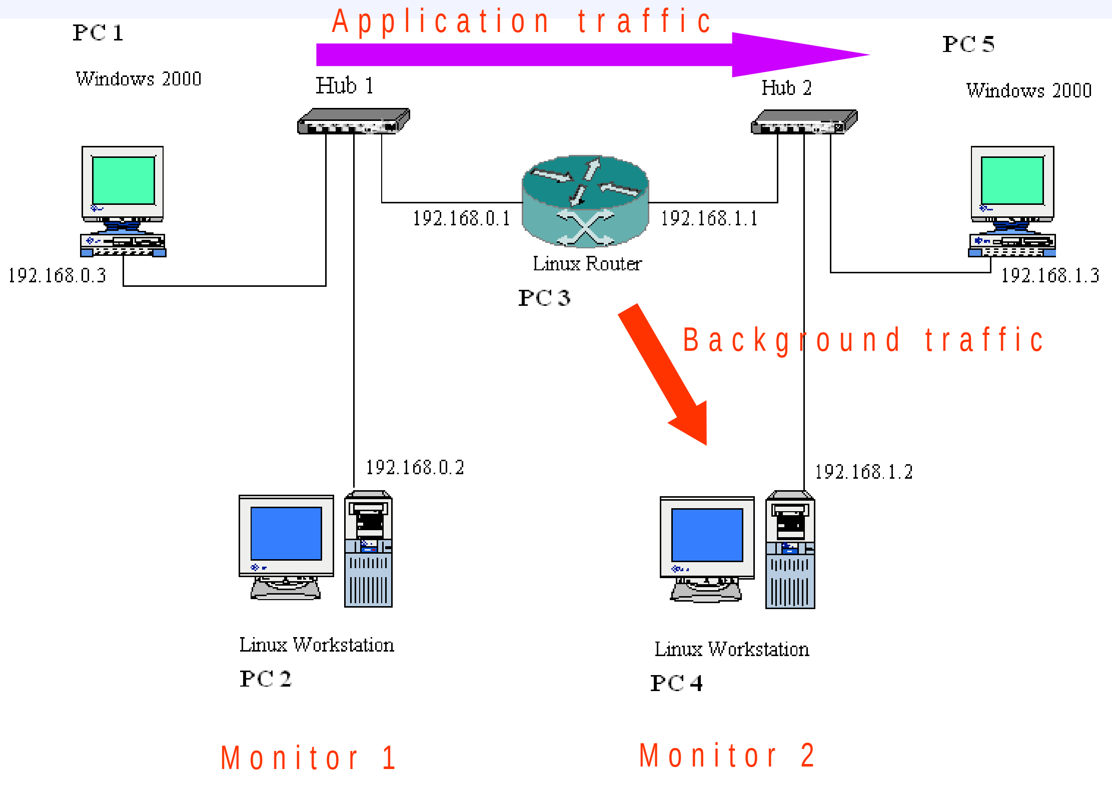
It doesn‘t look good



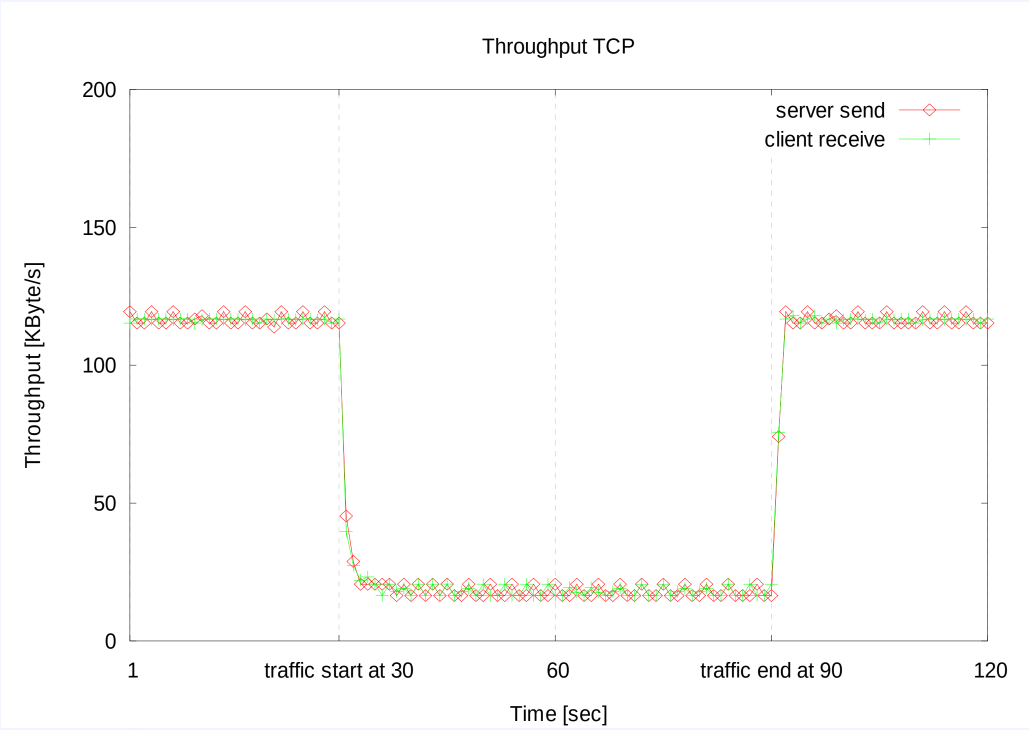


- For more details, see:
  Promoting the Use of End-to-End Congestion Control in the Internet. Floyd, S., and Fall, K..
  IEEE/ACM Transactions on Networking, August 1999.

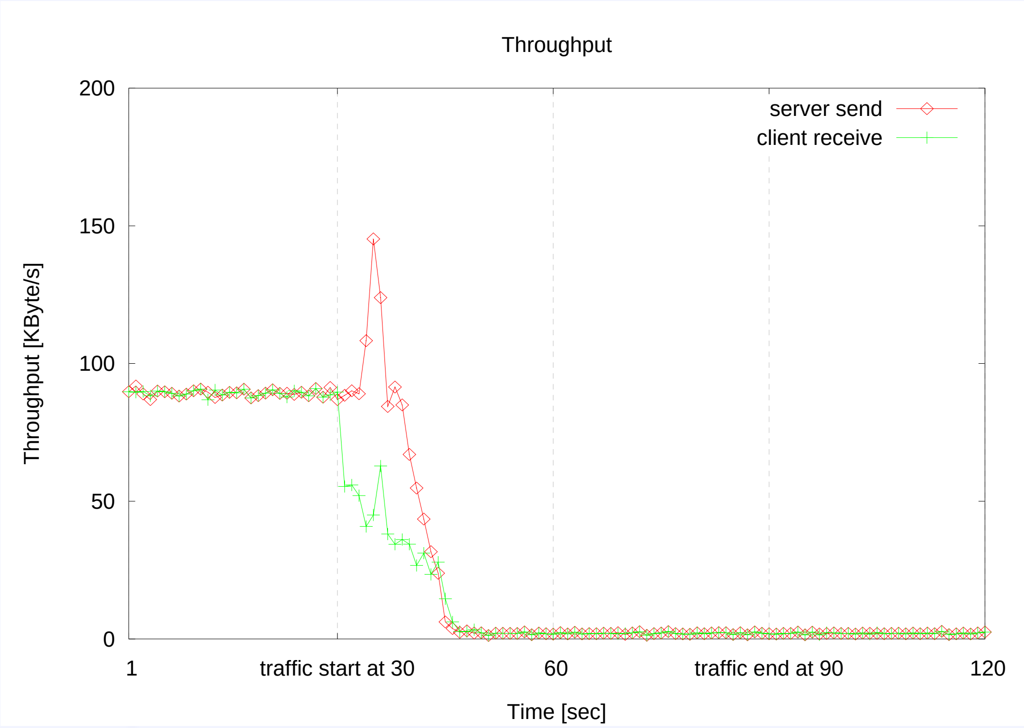
Real behavior of today's apps



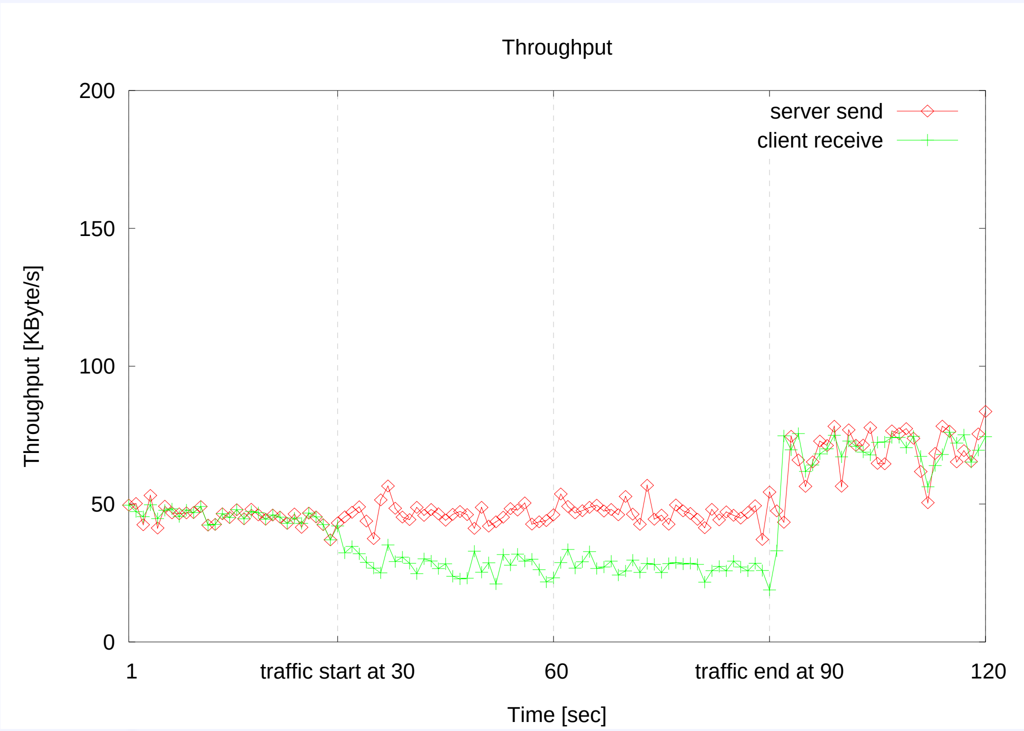
TCP (the way it should be)



Streaming Video: RealPlayer



Streaming Video: Windows Media Player

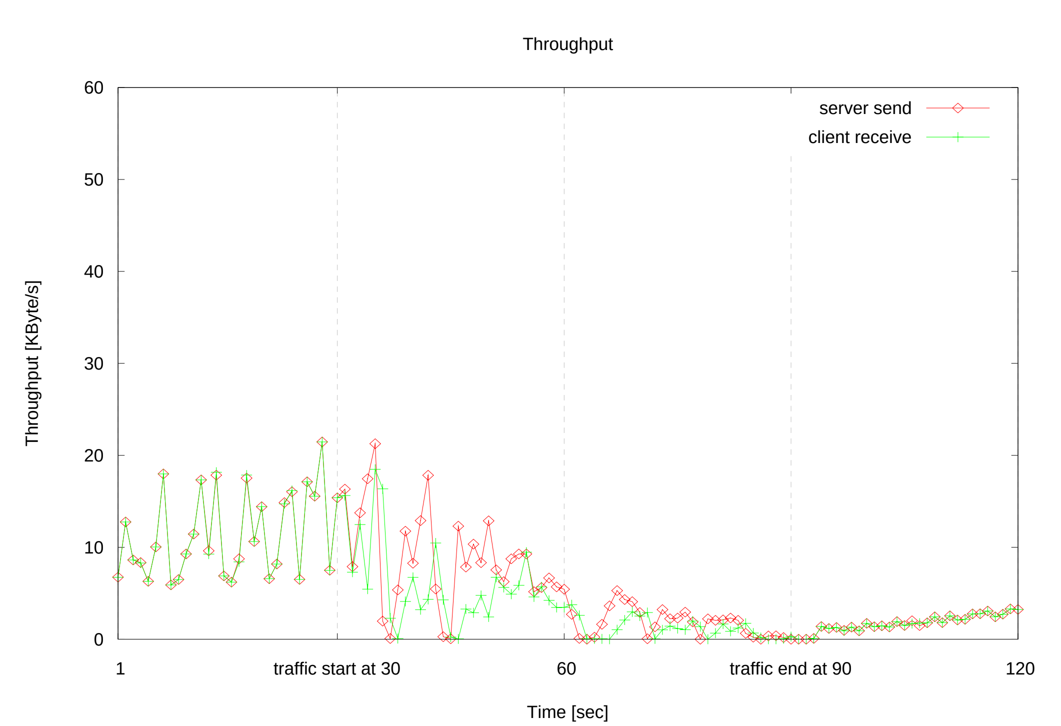


Streaming Video: Quicktime

VoIP: MSN

VoIP: Skype

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]

Conclusion

- TCP = too much
  - TCP++ (or rather TCP--) needed
- UDP = not enough
  - UDP++ needed
- We will see that, in fact, sometimes, even UDP = too much
  - UDP-- needed
- These gaps are filled by the new IETF transport protocols
  - TCP++ = SCTP
  - UDP++ = DCCP
  - UDP-- = UDP-Lite

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

SoA TCP + Extras

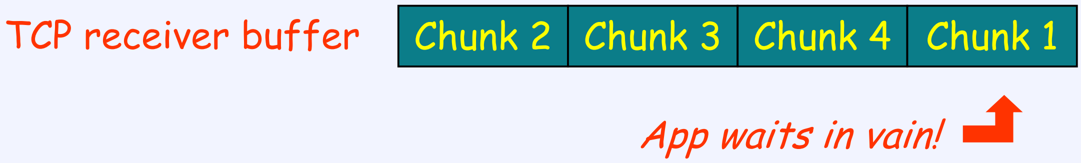
Services/Features	SCTP	TCP	UDP
Full-duplex data transmission	yes	yes	yes
Connection-oriented	yes	yes	no
Reliable data transfer	yes	yes	no
Unreliable data transfer	yes	no	yes
Partially reliable data transfer	yes	no	no
Ordered data delivery	yes	yes	no
Unordered data delivery	yes	no	yes
Flow and Congestion Control	yes	yes	no
ECN support	yes	yes	no
Selective acks	yes	yes	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
- SCTP is extensible

Performance enhancements

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



- Application Level Framing

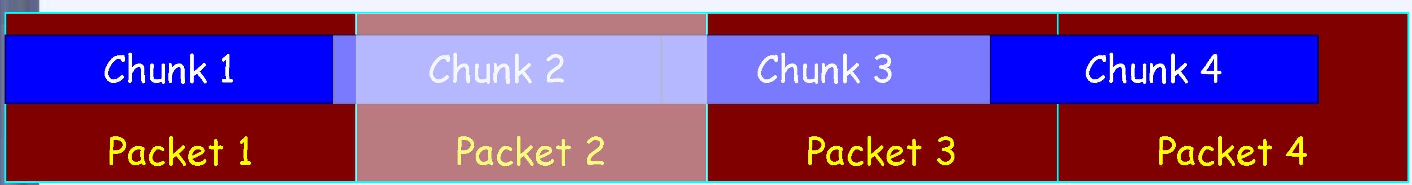
- Support for multiple data streams (per-stream ordered delivery)
  - Stream sequence number (SSN) preserves order *within* streams
  - no order preserved *between* streams
  - per-stream flow control, per-association congestion control

Application Level Framing

- TCP: byte stream oriented protocol

- Application may want logical data units (“chunks“)

- Byte stream inefficient when packets are lost

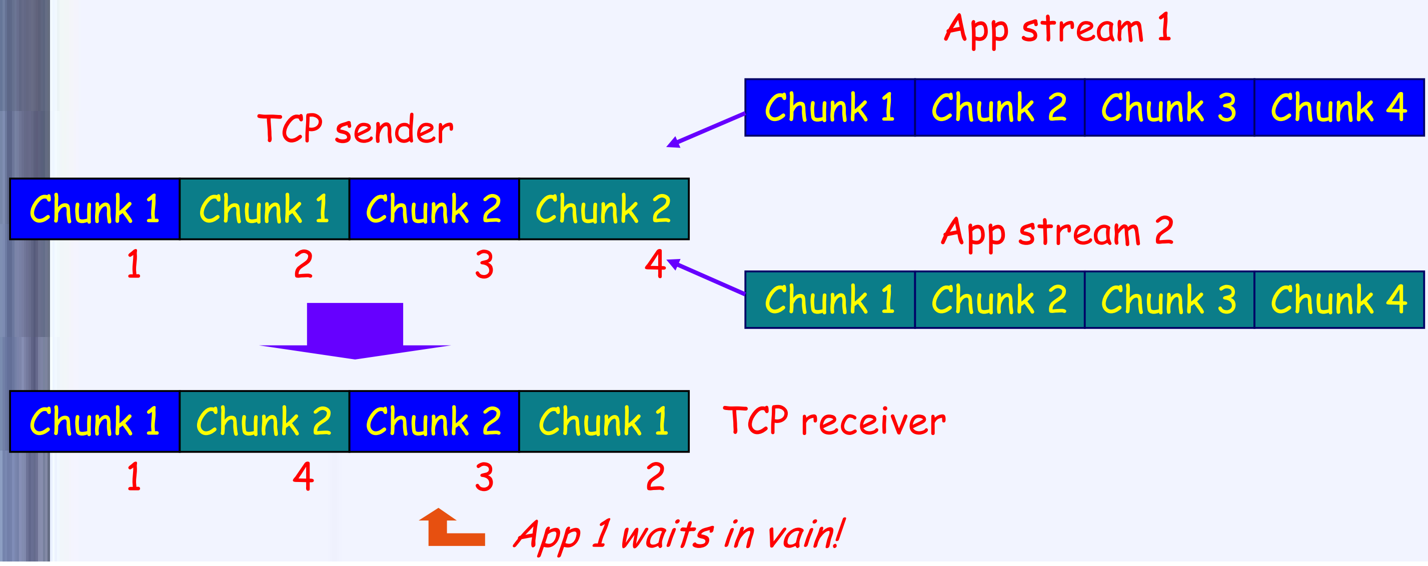


- ALF: app chooses packet size = chunk size
  packet 2 lost: no unnecessary data in packet 1,
  use chunks 3 and 4 before retrans. 2 arrives

- 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, overhead (connection setup/teardown, ..)



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 is allocated at this point!
  - 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
  - reliability 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 the other end continues sending new data)

UDP-Lite

UDP-Lite

32 Bits

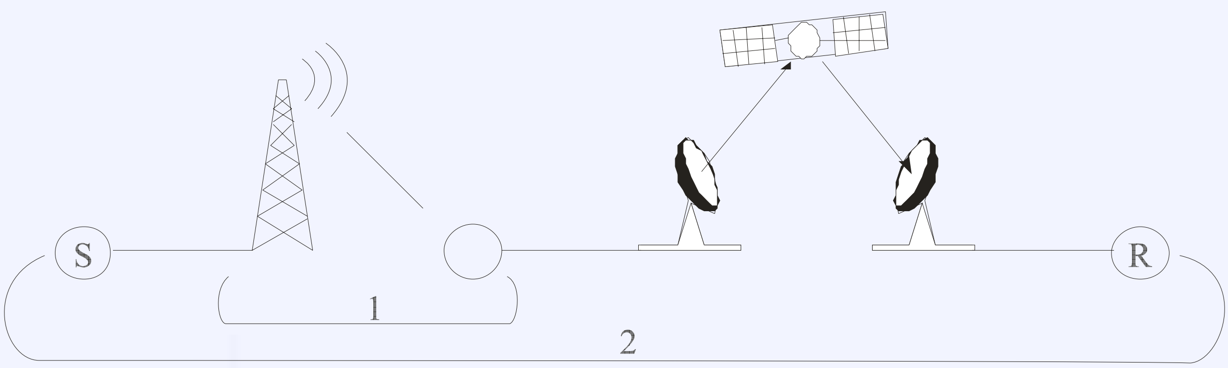
Source port	Destination port
Checksum coverage	UDP checksum

- 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

Link layer ARQ

- Advantages:
  – potentially faster than end-to-end retransmits
  – operates on frames, not packets
  – could use knowledge that is not available at transport end points
- example scenario: control loop 1 much shorter than 2



Link Layer ARQ /2

- Disadvantages:
  - hides information (known corruption) from end points
  - TCP: increased delay $\Rightarrow$ more conservative behavior
- Link layer ARQ can have varying degrees of persistence
- So what?
- Ideal choice would depend on individual end-to-end flows
- Thus, recommendation:
  - low persistence or disable (leave severe cases up to end points)
  - Give end points means to react properly (detect corruption)

Further details: RFC 3366

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

DCCP fundamentals

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

Not an explicit DCCP requirement, but a current IETF requirement

- 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
  - RFCs published in March 2006

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 a 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 follwing subheader
  - only one subheader per packet, not several as with SCTP chunks

Separate header / payload checksums

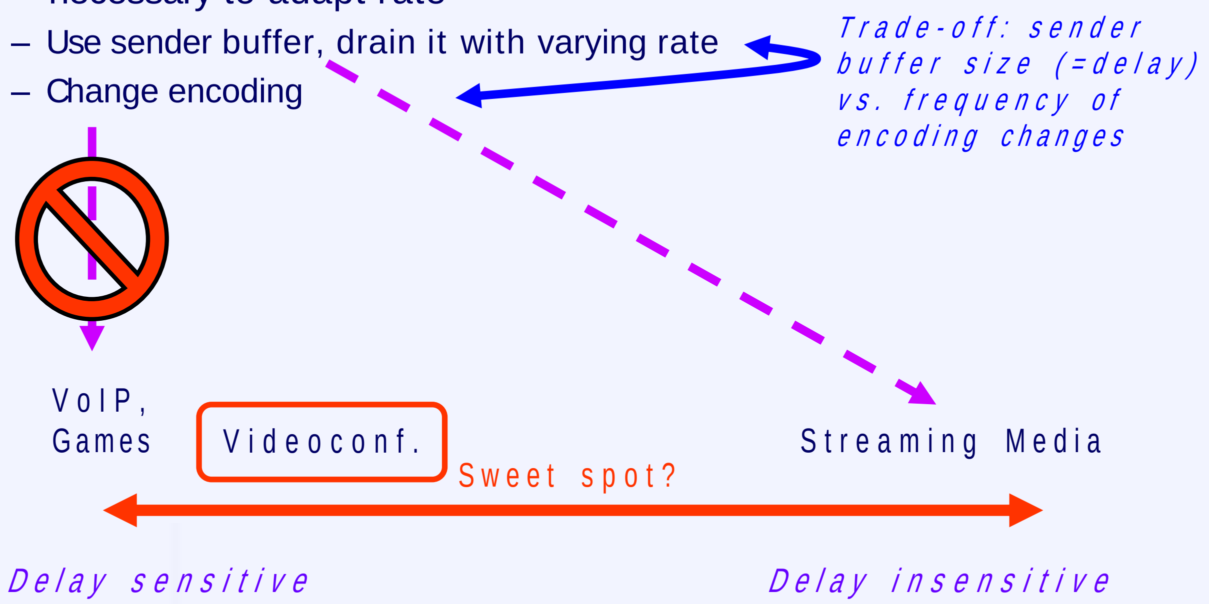
- Available as "Data Checksum option" in DCCP
  - Also suggested for TCP, but not (yet?) accepted
  - Note: partial checksums useless in TCP (reliable transmission of erroneous data?)
- Differentiate corruption / congestion
  - Checksum covers all
    - Error could be in header
    - Impossible to notify 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

Classifying DCCP applications

- Congestion control trade-off (selfish single-flow view):
  + reduced loss
  — necessary to adapt rate
    – Use sender buffer, drain it with varying rate
    – Change encoding



Is TCP the ideal protocol for one-way streaming media?

- Perhaps! Let's consider what happens…
- Remember: we're at the "buffering" side of the spectrum
  – Buffers (delay) don't matter
  – User perception studies of adaptive multimedia apps have shown that users dislike permanent encoding changes (big surprise :-))

  ⇒ no need for a smooth rate!
- Little loss case: TCP retransmissions won't hurt
- Heavy loss case:
- DCCP: 1, ~~2~~, ~~3~~, 4, ~~5~~, ~~6~~, 7, ~~8~~, ~~9~~, 10…
- TCP: (assume window = 3): 1, ~~2~~, ~~3~~, 2, ~~3~~, ~~4~~, 3, ~~4~~, ~~5~~, 4…
  – Application would detect: 4 out of 10 expected packets arrived ⇒ should reduce rate
  – Is receiving 1, 4, 7, 10 instead of 1, 2, 3, 4 really such a big benefit?
    - Or is it just a matter of properly reacting?
    - In RealPlayer and MediaPlayer, TCP can be used for streaming… seems to work well (also in YouTube!)

DCCP usage: incentive considerations

- Benefits from DCCP (perspective of a single application) limited
- Compare them with reasons not to use DCCP
  - programming effort, especially if updating a working application
  - common deployment problems of new protocol with firewalls etc.
- What if dramatically better performance is required to convince app programmers to use it?
- Can be attained using "penalty boxes" - but:
  - requires such boxes to be widely used
  - will only happen if beneficial for ISP:
    financial loss from unresponsive UDP traffic > financial loss from customers whose UDP application doesn't work anymore
  - requires many applications to use DCCP
  - chicken-egg problem!

References

- Michael Welzl: "Network Congestion Control: Managing Internet Traffic", John Wiley & Sons, July 2005.
- Randall R. Stewart, Qiaobing Xie: "Stream Control Transmission Protocols (SCTP)", Addison-Wesley Professional 2002.
- Key RFCs (main protocol specifications):
  - SCTP: RFC 2960; UDP-Lite: RFC 3828; DCCP: RFC 4340
- Recommended URLs:
  - SCTP, UDP-Lite:
    - http://www.ietf.org/html.charters/tsvwg-charter.html
  - SCTP:
    - http://www.sctp.org/
    - http://tdrwww.exp-math.uni-essen.de/inhalt/forschung/sctp_fb/
  - DCCP:
    - http://www.ietf.org/html.charters/dccp-charter.html
    - http://www.icir.org/kohler/dccp/