IASTED PDCN 2005 Tutorial:

Internet Transport
Today and Tomorrow

Michael Welzl  http://www.welzl.at

Institute of Computer Science
University of Innsbruck, Austria
Outline

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

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

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

3. Example research effort: Tailor-made Congestion Control
Internet protocol standardization

- Preliminary research in IRTF (http://www.irtf.org)
- Standards (RFCs) defined by IETF (http://www.ietf.org) - mostly Working Groups
- Decisions by IESG (as of Feb. 2001, 14 elected members)
- IAB stimulates IETF / IESG actions
  - Members elected by “Internet Society” (ISOC)
- RFCs have different status:
  - standard, proposed standard, draft standard
  - experimental, informational
- Internet-draft: preliminary - may turn into RFC
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

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

<table>
<thead>
<tr>
<th>Application</th>
<th>HTTP, FTP, ..</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transport</td>
<td>UDP</td>
</tr>
<tr>
<td>Network</td>
<td>IP</td>
</tr>
<tr>
<td>Access</td>
<td></td>
</tr>
</tbody>
</table>
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:
  - 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?
Transmission Control Protocol (TCP)
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 Header

<table>
<thead>
<tr>
<th>Source Port</th>
<th>Destination Port</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Sequence Number</th>
<th>Acknowledgement Number</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Header Length</th>
<th>Reserved</th>
<th>Window</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>C E U A P R S F</td>
<td></td>
</tr>
<tr>
<td></td>
<td>W C R C S S Y I</td>
<td></td>
</tr>
<tr>
<td></td>
<td>R E G K H T N N</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Checksum</th>
<th>Urgent Pointer</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Options (if any)</th>
<th>Data (if any)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Flags indicate connection setup/teardown, ACK, ..
- If no data: packet is just an ACK
- Window = advertised window from receiver (flow control)
TCP Connection Management

heavy solid line: normal path for a client

heavy dashed line: normal path for a server

Light lines: unusual events

Connection setup

teardown
Error Control: Acknowledgement

ACK ("positive” Acknowledgement)

Purposes:
- sender: throw away copy of SDU held for retransmit,
- time-out cancelled
- msg-number can be re-used

TCP counts bytes, not segments; ACK carries “next expected byte” (#+1)

ACKs are cumulative
- ACK n acknowledges all bytes "last one ACKed” thru n-1

ACKs should be delayed
- 1 ACK every 2 segments, at least 1 ACK every 500 ms (often set to 200 ms)
Error Control: Retransmit Timeout (RTO)

- RTO timer value difficult to determine:
  - too long ⇒ bad in case of msg-loss!
  - too short ⇒ risk of false alarms!
  - General consensus: too short is worse than too long; use conservative estimate

- Calculation: measure RTT (Seg# ... ACK#)

- Update RTO using Exponentially Weighed Moving Average (EWMA)

- Including variation (by Van Jacobson)
  \[
  \text{SRTT} = (1-\alpha) \text{SRTT} + \alpha \text{SRTT}
  \]
  \[
  \text{RTO} = \text{SRTT} + 4 \sigma
  \]
Window management

- Receiver “grants” credit (window)
  - sender restricts sent data with window

- Nagle algorithm: prevents Silly Window Syndrome (SWS)
  - sender waits until SMSS bytes can be sent
  - max. 1 smaller segment per RTT
A simple router model

- **Switching fabric** forwards a packet (dest. addr.) if no special treatment necessary: **fast path** (hardware)

- **Queues** grow when traffic bursts arrive
  - low delay = small queues, low jitter = minor queue fluctuations
  - Packets are dropped when queues overflow ("DropTail queueing")
    - low loss ratio = small queues
The congestion problem

• Congestion control necessary

• adding fast links does **not** help!

total throughput w/o cc.: 20kb/s

total throughput w/ cc.: 110kb/s
Congestion collapse

Goal: operation at the "knee"
Internet congestion control: History

- **1968/69**: dawn of the Internet
- **1986**: first congestion collapse
- **1988**: "Congestion Avoidance and Control" (Jacobson)
  Combined congestion/flow control for TCP
  (also: variation change to RTO calculation algorithm)

- **Goal**: stability - in equilibrium, no packet is sent into the network until an old packet leaves
  - ack clocking, “conservation of packets“ principle
  - made possible through window based stop+go - behaviour

- Superposition of stable systems = stable ➔ network based on TCP with congestion control = stable
TCP Congestion Control: Tahoe

- **Distinguish:**
  - *flow control*: protect receiver against overload
    (receiver "grants" a certain amount of data ("receiver window") )
  - *congestion control*: protect network against overload
    ("congestion window" (cwnd) limits the rate: min(cwnd,rwnd) used! )

- Flow/Congestion Control combined in TCP. Several algorithms:

  - (window unit: SMSS = Sender Maximum Segment Size, usually adjusted to Path MTU; init cwnd<=2 (*SMSS), ssthresh = usually 64k)
    - **Slow Start**: for each ack received, increase cwnd by 1 (exponential growth) until cwnd >= ssthresh
    - **Congestion Avoidance**: each RTT, increase cwnd by SMSS*SMSS/cwnd (linear growth - "additive increase")
Slow start and Congestion Avoidance

Sender
0
ACK 1
1
2
ACK 2
3
ACK 3
4
5
6
...
Sender

Receiver

Sender
0
ACK 1
1
2
ACK 2
3
ACK 3
4
5
6
...
Sender

Receiver
Tahoe vs. Reno

- Slow Start
- Congestion Avoidance
- 3 DupACKs
- ssthresh
- timeouts

Graph showing the comparison between Tahoe and Reno with various congestion control mechanisms.
One window, multiple dropped segments

- Sender cannot detect loss of multiple segments from a single window

- Insufficient information in DupACKs
  - NewReno:
    - stay in FR/FR when partial ACK arrives after DupACKs
    - retransmit single segment
    - only full ACK ends process

Example: ACK 2

Example: ACK 6
Selective ACKnowledgements (SACK)

- Example on previous slide:
  send ACK 1, SACK 3, SACK 5 in response to segment #4

- Better sender reaction possible
  - Reno and NewReno can only retransmit a single segment per window
  - SACK can retransmit more (RFC 3517)
  - Particularly advantageous when window is large (long fat pipes)

- but: requires receiver code change
Active Queue Management

• Today, TCP behaviour dominates the Internet (WWW, ..)

• (somewhat old) example backbone measurement: 98% TCP traffic

• 1993: Random Early Detection ("Discard", "Drop") (RED) (now that end nodes back off as packets are dropped, drop packets earlier to avoid queue overflows)

• Another goal: add randomization to avoid traffic phase effects!

• $Q_{avg} = (1 - Wq) \times Q_{avg} + Q_{inst} \times Wq$
  (Qavg = average occupancy, Qinst = instantaneous occupancy, Wq = weight - hard to tune, determines how aggressive RED behaves)
Active Queue Management /2

- Based on exponentially weighted moving average (EWMA) of instantaneous queue occupancy = low pass filter
  - recalculated every time a packet arrives

- $Q_{avg}$ below threshold $min\_th$: Nothing happens
- $Q_{avg}$ above threshold $min\_th$: Drop probability rises linearly
- $Q_{avg}$ above threshold $max\_th$: Drop packets

- RED expects all flows to behave like TCP - but is it fair?

- Variants: drop from front, drop based on instantaneous queue occupancy, drop arbitrary packets, drop based on priorities...
Explicit Congestion Notification (ECN)

- **1999**: Explicit Congestion Notification (ECN)
  Instead of dropping, set a bit

- End systems are expected to act as if packet was dropped
  ⇒ actual communication between end nodes and the network!

- ATM and Frame Relay: not only ECN but also BECN

- Internet BECN: often proposed and regularly discussed (ICMP SQ), but very unlikely - several reasons

- Quite popular among researchers - lots of ideas to exploit the bit!

- ECN cannot totally replace loss measurements!
ECN in action

1. Send packet with ECT = 1, CE = 0, nonce = random
   - Reduce cwnd, set CWR = 1

2. ECT = 1, so don’t drop update: CE = 1, nonce = 0

3. Set ECE = 1 in subsequent ACKs even if CE = 0
   - Only set ECE = 1 in ACKs again when CE = 1

- **Nonce** provided by bit combination:
  - ECT(0): ECT=1, CE=0
  - ECT(1): ECT=0, CE=1

- Nonce usage specification still experimental
TCP History

Standards track TCP RFCs which influence when a packet is sent

- **Basics**
  - RFC 793 09 / 1981
  - RFC 1122 10 / 1989
  - RFC 1323 05 / 1992

- **Slow start + congestion avoidance, SWS avoidance / Nagle, RTO calculation, delayed ACK**
  - RFC 2018 10 / 1996

- **Timestamps, PAWS, Window scaling**
  - RFC 2988 11 / 2000

- **SACK**
  - RFC 2988 11 / 2000

- **RTO**
  - RFC 2883 07 / 2000
  - RFC 3042 01 / 2001
  - RFC 3517 04 / 2003

- **DSACK**
  - RFC 3390 10 / 2002

- **Larger initial window**
  - RFC 3782 04 / 2004

- **NewReno**
  - RFC 3168 09 / 2001

- **Limited Transmit**

- **Full specification of Slow start, congestion avoidance, FR / FR**
  - RFC 3517 04 / 2003

- **SACK-based loss recovery**
  - RFC 3168 09 / 2001

- **ECN**

- **RFC**
  - RFC 2018 10 / 1996
User Datagram Protocol (UDP)
UDP

- 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..
Real behavior of today’s apps

Application traffic

Background traffic

Monitor 1

Monitor 2
TCP (the way it should be)

Throughput TCP

Throughput [KByte/s]

Time [sec]

Traffic start at 30

Traffic end at 90

Server send

Client receive
Streaming Video: RealPlayer

![Graph showing throughput over time for streaming video with RealPlayer. The graph indicates traffic start at 30 seconds and traffic end at 90 seconds. The x-axis represents time in seconds (1 to 120), and the y-axis represents throughput in KByte/s. There are two lines on the graph: one for server send and another for client receive.]
Streaming Video: Windows Media Player

Throughput

Throughput [KByte/s]

Time [sec]

server send
client receive

traffic start at 30
traffic end at 90
Streaming Video: Quicktime

Throughput [KByte/s] vs Time [sec]

- Server send
- Client receive

Traffic start at 30
Traffic end at 90
VoIP: MSN

Throughput

Throughput [KByte/s]

Time [sec]

Traffic start at 30
Traffic end at 90

Server send
Client receive
VoIP: Skype

Throughput [KByte/s]

Time [sec]

traffic start at 30
traffic end at 90

Throughput

server send

client receive

0 5 10 15 20 25

1 60 120

25 30 35 40 45 50 55 60

0 1
Video conferencing: iVisit

![Graph showing throughput in KBytes/s over time (seconds)]
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]
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

<table>
<thead>
<tr>
<th>Services/Features</th>
<th>SCTP</th>
<th>TCP</th>
<th>UDP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Full-duplex data transmission</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>Connection-oriented</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>Reliable data transfer</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>Partially reliable data transfer</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>Ordered data delivery</td>
<td>no</td>
<td>no</td>
<td>n/a</td>
</tr>
<tr>
<td>Unordered data delivery</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>Flow and Congestion Control</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>ECN support</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>Selective acks</td>
<td>yes</td>
<td>optional</td>
<td>no</td>
</tr>
<tr>
<td>Preservation of message boundaries</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>PMTUD</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>Application data fragmentation</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>Multistreaming</td>
<td>yes</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>Multihoming</td>
<td>yes</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>Protection against SYN flooding attack</td>
<td>yes</td>
<td>no</td>
<td>n/a</td>
</tr>
<tr>
<td>Half-closed connections</td>
<td>no</td>
<td>yes</td>
<td>n/a</td>
</tr>
</tbody>
</table>
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 containing 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

  ![TCP receiver buffer diagram](image)

  **App waits in vain!**

- **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?)

TCP sender

TCP receiver

App 1 waits in vain!
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

- **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**

- **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
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 ⇒ 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

• 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 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

<table>
<thead>
<tr>
<th>Source Port</th>
<th>Destination Port</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Offset</td>
<td>CCVal</td>
</tr>
<tr>
<td></td>
<td>CsCov</td>
</tr>
<tr>
<td></td>
<td>Checksum</td>
</tr>
<tr>
<td>Res</td>
<td>Type</td>
</tr>
<tr>
<td></td>
<td>X</td>
</tr>
<tr>
<td>1</td>
<td>Reserved</td>
</tr>
<tr>
<td></td>
<td>Sequence Number (high bits)</td>
</tr>
<tr>
<td></td>
<td>Sequence Number (low bits)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Source Port</th>
<th>Destination Port</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Offset</td>
<td>CCVal</td>
</tr>
<tr>
<td></td>
<td>CsCov</td>
</tr>
<tr>
<td></td>
<td>Checksum</td>
</tr>
<tr>
<td>Res</td>
<td>Type</td>
</tr>
<tr>
<td></td>
<td>X</td>
</tr>
<tr>
<td>0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Sequence Number (low bits)</td>
</tr>
</tbody>
</table>

- 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 different drop 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

- Benefits from DCCP (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]
Tailor-made Congestion Control

A research project at the University of Innsbruck
Current use of the Internet

• TCP
  - byte stream from source to destination
  - reliable, connection oriented service
  - all kinds of complex features
    • window based flow and congestion control
      - RTT estimation, self-clocking, parameters: max. / init. window size,…
      - slow start / congestion avoidance
    • flavors: Tahoe, Reno, NewReno, SACK, with and w/o ECN, ..

• UDP
  - connectionless service
  - ports and a checksum … that’s it :)  
    • simpler, but useless for reliable transport (DIY)
    • What about congestion control?
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: an "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)
• possible 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

- Idea requires:
  - IETF standardization
  - real-world deployment in common OS’s
  - new apps that use it ... or an upgrading strategy (realistic?)

- Quite a goal

- Okay, so this may never happen ... but:

- it is research worth pursuing - if approached with care
  - Started September 2004
  - Currently working on gradual deployment:
    transparently impose congestion control on standard UDP flows for the benefit of all; provide UDP interface + optional extras
References (sources)

- Some pictures / slides from:
  - Max Mühlhäuser, Murtaza Yousaf
  - bachelor students: Muhlis Akdag, Thomas Rammer, Roland Wallnöfer

- IP hourglass picture from:

- Some content from:
  - Various RFCs / Internet-drafts

- Recommended URLs:
  - http://www.ietf.org
  - http://www.icir.org/kohler/dccp/
  - http://www.sctp.org/
  - http://tdrwww.exp-math.uni-essen.de/inhalt/forschung/sctp_fb/
Thank you!