isslow:

Integrated Services System for LOW bitrate environments

Carsten Bormann (cabo@informatik.uni-bremen.de)

Universität Bremen

Overview

- Overview, pieces of the puzzle
- Status of header-compression proposals
- Requirements for a link-muxing protocol
- Relevant ITU low-level link management work
- Relevant ITU modem work
- Alternative link-muxing proposals

Integrated Services on low-bitrate links

- ***Integrated Services:***

→ ***transport of real-time multimedia information on the Internet***

- parameters for forwarding mechanisms (intserv)
- setup protocol to establish special forwarding treatment (RSVP)
- transport protocol for real-time information (RTP/RTCP in avt)

- ***Up to now: High-speed links only...***

Integrated Services on low-bitrate links

- ***Current low-bitrate environments***
– POTS: 14.4 to 28.8 kbit/s, erratic V.42 protocol
– ISDN: 56, 64, 2∗64 kbit/s, HDLC
– Low speed leased lines: 56, 64, 128 kbit/s; HDLC
– Fractional T1
→ Common protocol: PPP
– Frame Relay
– Mobile environments?
– Bandwidth on demand environments

Usage Scenarios

- ***Applications***
- – Internet Telephony
- $\rightarrow$ Internet Multimedia Conferencing
- ***Configurations***
- – Remote PC attached via modem to ISP/company
    - Including mobile case
- – Remote PC attached via Frame Relay to ISP/company
- – Remote PC attached via Leased Line (FT1) to ISP/company
- – Small Office router (multiple streams)

The Problem

– Low-speed links blocked by large frames:

1500 Bytes == 400 ms (@ 28.8 kbit/s)

→ Latency!

⇒ Provide fragmentation, suspend/resume

– Header Overhead:

44 Bytes @ 50 frames/sec == 17.6 kbit/s

→ need fewer, larger packets

→ Latency!

⇒ Provide good header compression

– Hard to reserve bandwidth with unknown header requirements

⇒ Obtain compressibility hints from application

Components of the ISSLOW Architecture

- ***a real-time encapsulation***
- – easy in principle,
- – hard to select right group of standards to be compatible with
- ***a header compression architecture***
- – see next slide
- ***a way to map IP flows on to encapsulation/compression***
- – identify realtime flows: RSVP
  - e.g., header compression hints in RSVP PATH messages
- – to do

Status of header-compression

- Hop-by-hop vs. end-to-end
- ***Hop-by-hop: IPv6*** (Degermark et al.: draft-degermark-ipv6-hc-01.txt)
  - Compresses successive headers identified by protocol field
  - Stops with UDP header (no further protocol field)
- ***Hop-by-hop: Casner/Jacobson*** (ftp://hydra.precept.com/pub/rtp/crtp.txt)
  - Can compress IP/UDP/RTP or just IP/UDP
    - Identify RTP by heuristics
  - More aggressive than Degermark et al.
  - Still loss-less (e.g., preserves UDP checksum, if present)
- ***End-to-end: Petrack*** (based on CJ draft)
  - Compresses just RTP (12 out of 44 bytes of overhead)

⇒ ***AVT WG on Tuesday afternoon (13–15)***

Requirements for a real-time encapsulation

- Minimize *worst case* end-to-end delay for real-time streams
  - Integrate low- and high-priority streams
  - Interrupt long low priority frame for timely high priority delivery
  - Provide usual MTU at low priority

  → How many levels of priority?
- Maximize available bandwidth, minimize overhead
  - Fixed cell size solutions are at disadvantage
  - Don't add further header overhead
- Be predictable enough for admission control
  - Provide a way to make use of application hints
- Robust against errors
  - Same level of error detection as PPP

Requirements for a real-time encapsulation

Fortsetzung

– Must cooperate nicely with PPP
  - Fall back to LCP without problems
  - Don’t forget Frame Relay, though
– Must work with existing chips and router systems
  - Synchronous: HDLC framing; hard to switch off per-frame CRC
  - Asynchronous: much more freedom in design
– Future proof
  - Modem Issues... V.42 vs. V.80 (async may become irrelevant)
  - Relationship to DSVD, H.324, ...?

Design avenues for real-time encapsulation

– abort/suspend → suspend

– cell-oriented + final bit vs. suspend/resume

→ Integrate suspend/resume with HDLC

– Which place for the suspend info?
  - Special flags (easy for async, V.76 → special chips)
  - byte at end of frame information field
  - bit at start of next frame

– How many priorities
  - Stacking vs. explicit stream identification

– Where to put error control
  - Per-frame CRC vs. per-packet CRC

H.223: Background

– H.324: Videoconferencing (Videophone) standard
  - Uses V.34 modem technology: 28800 kbit/s
  - Needs to multiplex voice, video, data
– Existing multiplex H.221 is ISDN-oriented
  - Works in increments of 8 kbit/s (too large)
  - Bitwise multiplexing expensive in software
– New multiplex for H.324:
  - H.223, to be defined Feb 1996
  - Uses HDLC framing/bitstuffing hardware
  – but not CRC, address/control fields, etc.

Design goals

- Integrate low- and high-priority streams
  - Interrupt long low priority frame for timely high priority delivery
  - Provide arbitrary frame lengths at low priority
- Minimal overhead
  - No CRC on streams that don't need it
  - 4-bit multiplex table index
- Allow error control
  - Allow classical CRC in data frames (do HDLC on top)
  - Provide limited retransmissions for video

H.223: A Multimedia Multiplex on HDLC

Architecture

- ***Multiplex layer***
  - HDLC frames (flags/bit stuffing) → 3 % overhead
  - Select multiplex table entry in 4 bits of first byte
  - Can end frame at any time
    - Bit in packet header of *next* frame: real end?
- ***Adaptation layer***
  - AL1: framed data, segmentable
    - Also provides “unframed transfer mode”
  - AL2: CBR streams, error indication only
  - AL3: VBR streams, limited error control

Multiplex layer

– HDLC frames, 1-byte header (MUX-PDU):

8	7	6	5	4	3	2	1
HEC			Multiplex code				PM
information							

– PM indicates whether the end of *previous* HDLC frame was the end of the segmentable data in there
  - Use 1-byte HDLC frame with PM=1 before period of quiescence

– HEC protects multiplex code

– information field contains sequence of *slots* from different streams

– multiplex code is index in *multiplex table*
  - index 0: control channel (update multiplex table, H.245)
  - index 1–15: sequence of defined length slots, plus one slot terminated by HDLC frame end

Adaptation layers

- ***AL1: Data***
  - variable length or unframed (byte stream)
  - segmentable (use PM to find out about end)
  - AL1-SDU end must be at end of HDLC frame
  - no error detecion (AL1-SDU may contain CRC or whatever)
- ***AL2: Audio***
  - generally unsegmentable (AL2-SDU ends at end of slot)
  - possibly variable length (end prematurely at frame end)
  - 8-bit CRC, optionally 8-bit sequence number
- ***AL3: Video***
  - variable length
  - 16-bit CRC, optionally 8- or 16-bit sequence number
  - optional retransmission based on negative acknowledgements

Recent ITU DSVD Modem Developments

- V.76: DSVD modem multiplex
- V.80: making synchronous modem streams accessible via async lines
- ***presentations by Fred Burg and Barry O’Mahony***

Alternative link-muxing proposals

ISSLOW Annex A RTE

- ***Adapt H.223 to PPP***
- – Remove AL3 negative acknowledgement mechanisms
- – Protect 0xFF initial header byte
- – Keep per-frame CRC
    - Need more than one *block* per frame for efficiency
- – Use H.223 style multiplex table for compression info, too

Alternative link-muxing proposals

SRTE: Simple real-time encapsulation

- ***Suspension***

– suspension flag in the last byte of each frame information field

- SUSPEND (TBD): suspend; discard
- TERMINATE (TBD): terminate; discard
- any other: terminate; keep the byte

– overhead:

- "1/128 byte" for non-suspended frames
- 1 byte for suspended frames

Alternative link-muxing proposals

SRTE: Simple real-time encapsulation

Fortsetzung

- ***Resumption***

– Header:

- resume correct one of possibly multiple suspended packets

$\rightarrow$ stream number (about 4 to 7)

- detect loss of frames reliably

$\rightarrow$ sequence number (3-bit)

SRTE: Simple real-time encapsulation

- ***Insertion overhead hack***
- header: optional additional byte: length of inserted block + stream number

  → overhead of insertion = size of SRTE header

  - deliver immediately (checksum?!)
- ***Header overhead hack***
- for each stream, can define byte string to be prepended by receiver

Alternative link-muxing proposals

SRTE: Simple real-time encapsulation

Fortsetzung

```
  0   1   2   3   4   5   6   7
+---+---+---+---+---+---+---+---+
| R | I |  sequence |  stream   |
+---+---+---+---+---+---+---+---+
:    length L  (if I)    | stream:
+.......................+.......+
:                               :
:        Inserted packet        :
:          (length L)           :
:                               :
+---+---+---+---+---+---+---+---+
|            data               |
:                               :
+...............................+
:       SUSPEND/TERMINATE       :
+---+---+---+---+---+---+---+---+
|             Frame             |
|              CRC              |
+---+---+---+---+---+---+---+---+
```

SRTE: Simple real-time encapsulation

- ***Information to be negotiated***

– for each of the seven streams:

  • implicitly prepended header bytes.

  • immediate delivery of inserted packets before frame end (no wait for CRC)

– for the entire link: use SRTE

  • fall back automatically on 0xFF (reserve stream 7)

The PPP multilink approach

- ***David Oran, Cisco***

– based on PPP multilink protocol (RFC1717)

  - sequence numbering (12 or 24 bit)
  - begin/end bits

  → fragmentation for multilink

– serial sequence numbering does not allow suspension

– but can send intervening non-multilink packets

Alternative link-muxing proposals

The PPP multilink approach

Fortsetzung

- ***Problems***

– only one level of suspension:

  - “Big” packets via multilink
  - “Small” packets outside of ML, not suspendable

– begin/end bits in the multilink *header*

  - send big packets with “end” off
  - if not suspended: close by zero-length fragment with “end” bit

  $\rightarrow$ 9 bytes per packet, six bytes per additional fragment

The PPP multilink approach

- ***Strengths***

– may fit a “mini-ATM” approach quite well, though

- slice all packets into regularly sized fragments
- minimum impact on serial queueing hard-/software
- 128 byte fragment size was suggested
  - 5–10 % overhead
  - 35 ms @ 28.8 kbit/s

– Good short-term approach

- uses existing standard, ML now widely deployed
- can interoperate with any ML-capable receiver

Where are we?

- ***Enlist:***
  - AVT: RTP compression (Casner/Jacobson, Petrack)
  - PPPEXT: framing, negotiation, interactions with other PPP layers
  - RSVP: header compression hint objects in PATH messages
  - ISSLL: keep the work coherent, actual intserv mappings
- ***Schedule:***
  - get implementable drafts out by August