Consistent Overhead Byte Stuffing

Stuart Cheshire & Mary Baker

{cheshire,mgbaker}@cs.stanford.edu

http://mosquitonet.stanford.edu/mosquitonet.html

Computer Science Department, Stanford University

Goal

- A byte stuffing algorithm which has
  - a simple efficient software implementation
  - consistent predictable overhead:
    - » always increases the packet length by about the same amount regardless of the packet contents

Talk Outline

- Framing & Byte Stuffing
- Byte Stuffing Example (PPP)
- Consistent Overhead Byte Stuffing (COBS)
- Theoretical Results
- Experimental Results
- Applications
- Conclusions

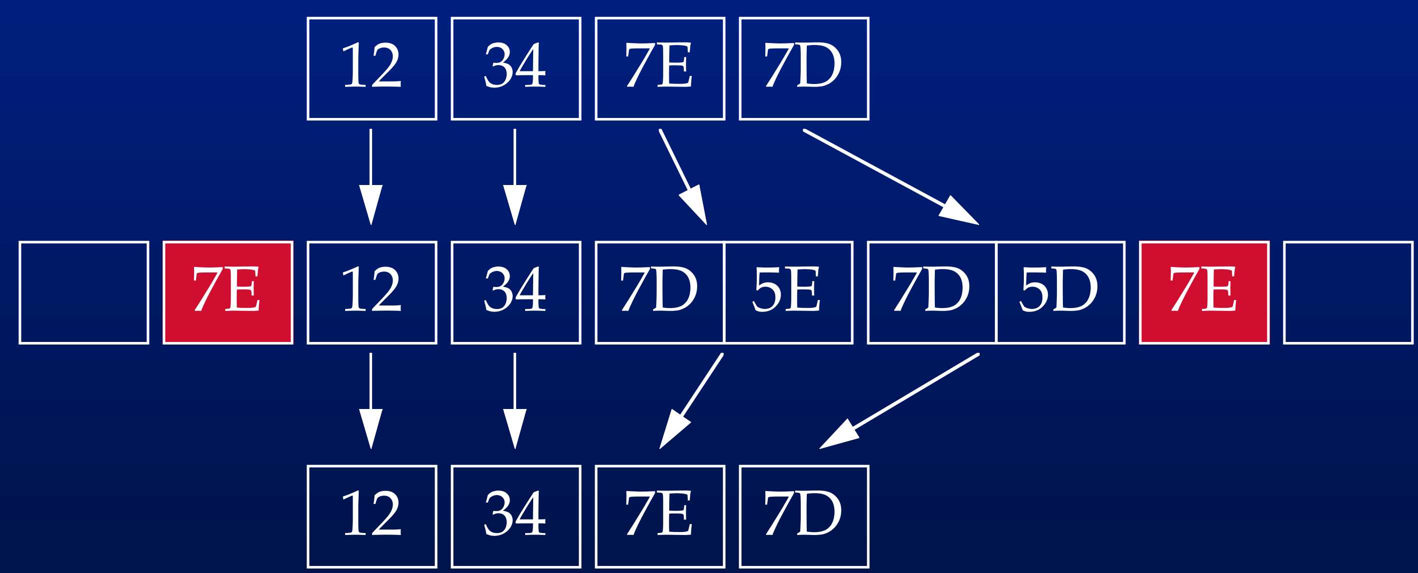
Framing & Data Stuffing

- Send packets over a serial medium — receiver needs to turn serial stream back into packets
- Need special character to mark packet boundaries
- Need to eliminate marker character from packet payload

Byte Stuffing Example (PPP)

- End-of-Packet marker: 0x7E
- Two-for-one substitution:
  - Escape Character: 0x7D
  - Replace 0x7E with: 0x7D 0x5E
  - Replace 0x7D with: 0x7D 0x5D

PPP Byte Stuffing Example



Byte Stuffing Overhead

- Have to engineer system assuming that any packet could double in size
  - Set IP MTU to half the true max packet size
  - Significantly degrades performance
- Packets that actually double in size are vanishingly rare
  - Performance sacrificed to accommodate packets that in practice hardly ever appear
  - Still, can't ignore those packets

Consistent Overhead Byte Stuffing

- Reversible data transformation to eliminate framing character from packet
- Any such transformation must, on average, increase size of packet
- With COBS the increase always falls in a very narrow range:
  - Minimum: One byte
  - Maximum: One byte per 254 bytes of data

How COBS Works

45	00	00	2C	4C	79	00	00	40	06	4F	37

45	00	00	2C	4C	79	00	00	40	06	4F	37	00

02	45	01	04	2C	4C	79	01	05	40	06	4F	37

	00	02	45	01	04	2C	4C	79	01	05	40	06	4F	37	00	

One Final Detail

- How do we encode
  a single (zero-terminated) block
  that is greater than 255 bytes long?
  - Code 0xFF represents 254 non-zero bytes
    without a zero on the end.

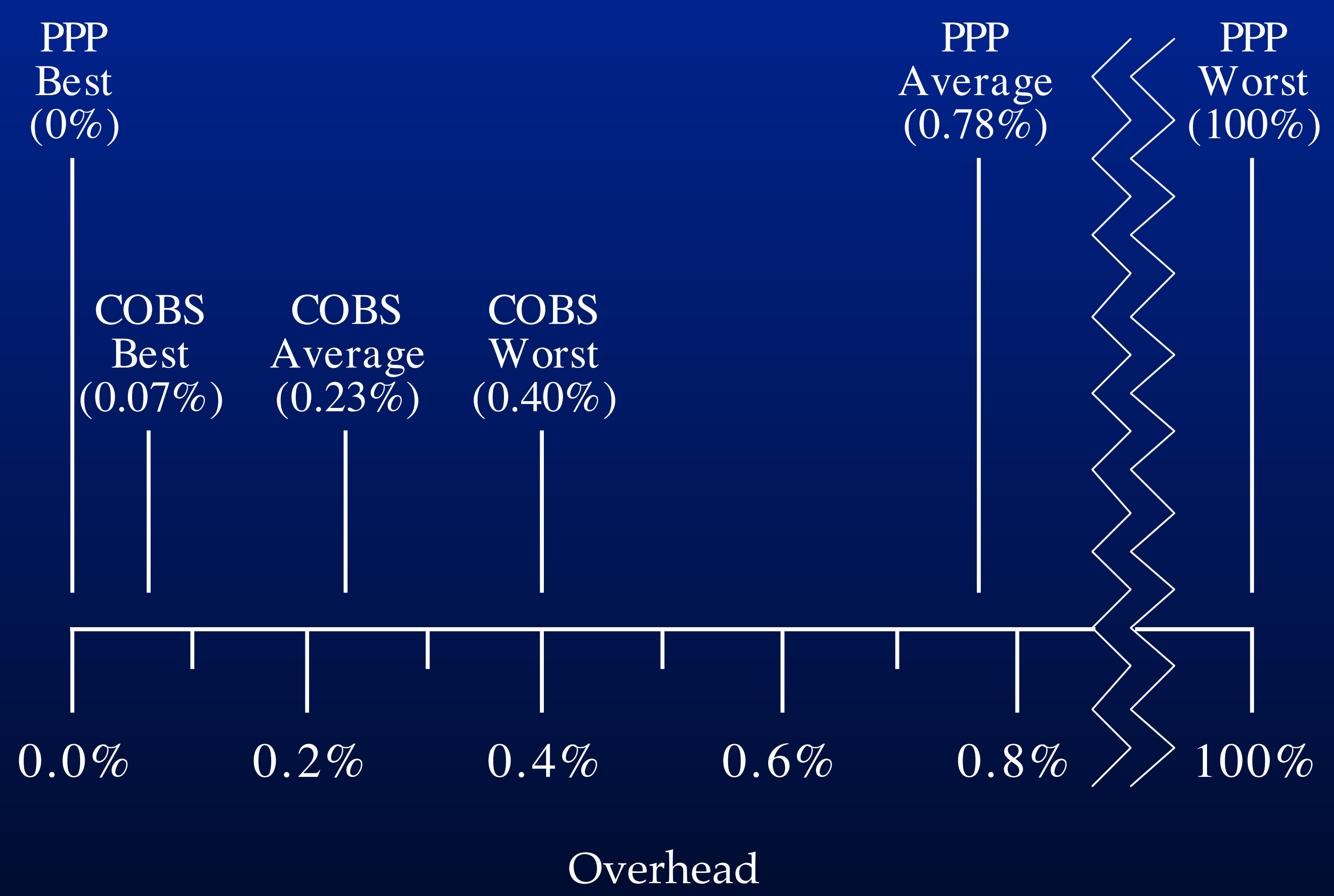
COBS Encoding Summary

- Code 0x00 not used
- Codes 0x01-0xFE mean
  - n-1 non-zero bytes
  - plus an implicit zero byte on the end
- Code 0xFF means
  - 254 non-zero bytes with no zero on the end
  - Encodes 254 bytes using 255
  - Increases data size by 0.394%

Theoretical Analysis

- Best Case
  - PPP: No overhead
  - COBS: One byte
- Average (Expected) Case
  - PPP: 1 byte in 128 (0.78%)
  - COBS: 1 byte in 436 (0.23%)
- Worst Case
  - PPP: Doubles packet size (100.00%)
  - COBS: 1 byte in 254 (0.39%)

Overhead for 1500 Byte Packet



Experimental Results

- 3-day user trace
  - 36,744 packets, 10,060,268 bytes
  - PPP overhead
    - Total overhead 36,069 bytes (0.36%)
    - Average 0.98 bytes per packet
    - Maximum observed overhead 53 bytes
  - COBS overhead
    - Total overhead 57,005 bytes (0.57%)
    - Average 1.55 bytes per packet
    - Maximum observed overhead 4 bytes

Experimental Results

- 15.3MB MPEG file transfer trace
  - 25,858 packets, 18,269,430 bytes
  - PPP overhead
    - Total overhead 101,024 bytes (0.55%)
    - Average 3.91 bytes per packet
    - Maximum observed overhead 20 bytes
  - COBS overhead
    - Total overhead 35,410 bytes (0.19%)
    - Average 1.37 bytes per packet
    - Maximum observed overhead 5 bytes

Applications

- IP over ATM
  - AAL5 assumes external framing marker
- MPEG video streams
  - Over satellite broadcast channels
  - On video CDs
- Records in a database file
  - Mail messages in a POP mbox
- Tape backup

Conclusions

- Benefits of COBS
  - Easy to implement efficiently in software
  - Average overhead comparable with PPP
  - Worst-case overhead much better than PPP (no more than 1 byte in 254)
    - » Improves network performance:
      Allows us to increase IP MTU from
      50% of underlying packet size to 99.6%