This full text paper was peer reviewed at the direction of IEEE Communications Society subject matter experts for publication in the ICC 2007 proceedings.

Bandwidth-efficient byte stuffing

Jaime S. Cardoso
Universidade do Porto / INESC Porto
Campus da FEUP, Rua Dr. Roberto Frias, nº 378
4200-465 Porto - Portugal
jaime.cardoso@inescporto.pt

***Abstract*— Byte stuffing is a technique to allow the transparent transmission of arbitrary sequences with constrained sequences. To date, most of the existing algorithms, such as PPP, attain a low average overhead by sacrificing the worst-case scenario. An exception is COBS which was designed for a low worst-case overhead; however, it imposes always a nonzero overhead, even on small packets. In this work is proposed a byte stuffing algorithm that simultaneously controls the average and worst-case overhead, performing close to the theoretical bound. It is shown analytically that the proposed algorithm achieves improved average and worst-case rates over state of the art methods. Furthermore, this technique is generalized to hybrid methods, with lower computing complexity. It is further analysed and compared experimentally the behaviour of the proposed algorithm against established algorithms in terms of byte overhead and computational time.**

I. Introduction

When sending packets over a serial medium the beginning and end of each frame must be recognizable to assist synchronization. A single flag byte is commonly used as the closing flag for one frame and the opening flag for the next. On both sides, communicating peers are continuously hunting for the flag byte to synchronize on the start of a frame. While receiving a frame, a peer continues to hunt for that flag to determine the end of the frame. Because a packet may contain an arbitrary byte (there is no restriction on the content) there is no assurance that the flag byte will not appear somewhere inside the frame, thus destroying synchronization. To avoid this problem, recovering data transparency, a procedure known as byte stuffing is used to remove from the user data the flag byte. [1]

PPP [2] eliminates the flag from the data payload using the following operation: everywhere that flag (0x7E) appears in the data it is replaced with the two-character sequence 0x7D, 0x5E. 0x7D is called the Control Escape byte. Everywhere that 0x7D appears in the data it is replaced with the two-character sequence 0x7D, 0x5D. The receiver performs the reverse process: whenever it sees the Control Escape value (0x7D) it discards that byte and XORs the following byte with 0x20 to recreate the original input. On average this byte stuffing does reasonably well, increasing the size of purely random data by 0.78%, but in the worst case it can double the size of the data being encoded. [3]

The large variability in the overhead of PPP, together with its high-cost worst-case, is source of some problems when designing a communication system, needing a conservative system design [3]. Other standards, such as SLIP [4] and HDLC [5], present the same behaviour.

COBS [3] attacks these problems by performing the stuffing operation on the packet level. COBS guarantees that for all packets, no matter what their contents, the overhead will be small. The occurrence of overhead in PPP is relatively rare but when it does occur the cost is high. With COBS, the overhead is not the rare case: it is the common case. However, in the worst case only one byte for every 254 bytes of data is added. But, even small packets with less 254 bytes will always suffer at least 1-byte overhead.

A. YABS

A proposal for yet another byte stuffing (YABS) algorithm can be presented. YABS divides a packet in chunks of 254 bytes (or fewer in the last chunk). Next, YABS encodes each chunk by adding a leading byte to the chunk: the added byte has to be different from the flag byte and can not be present on the chunk (always existing such byte for chunks with size up to 254 bytes). Finally, replace each occurrence of the flag byte on the chunk by the leading byte. The decoding process should be obvious. The YABS algorithm has the same worst-case performance as the COBS algorithm, although with a worst mean performance.

B. Beyond simple heuristics for byte stuffing

YABS and COBS have a clear problem for small packets, where the constant 1 byte overhead is noticeable. As exemplified, it is not difficult to construct an algorithm with a tight bound on the worst-case scenario; the challenge is on constructing algorithms simultaneously very efficient on the average and with low worst-case overhead. Instead of evaluating educated heuristics for byte stuffing, we introduce in the remainder of the paper an approach that is almost optimally bandwidth-efficient.

In section II the proposal algorithm, BABS, is described. Section III compares the overhead of PPP, COBS and BABS for stuffing the same data packets. We consider the expected and the worst-case overhead from a theoretical point of view. To overcome the time-complexity of BABS, section IV presents a Hybrid method, incorporating BABS and COBS in a single algorithm. Real time traffic is used to experimentally evaluate the different algorithms in section V. Finally, some conclusions are drawn and future work is oriented in the last section.

1-4244-0353-7/07/$25.00 ©2007 IEEE

II. Base Adaptation Byte Stuffing Algorithm

Babs performs a reversible transformation on a data packet to eliminate a single byte value from it. Byte stuffing is a process that takes as input a packet composed of characters from an alphabet of 256 possible symbols and gives as output a packet composed of characters from an alphabet of only 255 possible symbols. In the following description assume that the characters of the alphabet are the first integer numbers: $0, 1, 2, \cdots$. Now we can think on the data packet to send as a (big) number written in base 256; each byte is a character of the source alphabet. Applying standard arithmetic, we can perform a base conversion, changing from a number in base 256 to a number in base 255 [6]. With this simple operation, we have now a data packet making use of only 255 different characters (0..254), eliminating the byte 255 from the stream[1]. Fig. 1 shows some examples of data packets and the corresponding transformed packets.

It is well known that a number has a unique representation in a given base system. However, there is not a unique correspondence between packets and numbers. It suffices to note that appending zeros in the most significant byte positions does not change the number implicitly represented but changes the packet. To exemplify, consider Fig. 2, where all packets have the same representation as numbers in base 256.

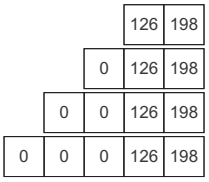


Fig. 2. Example of data packets corresponding to the same number in base 256.

To reestablish the reversibility of the transformation we just impose that both packets, the original and the stuffed packet, have the same number of trailing 0 characters. Fig. 3 illustrates the concept for some packets.

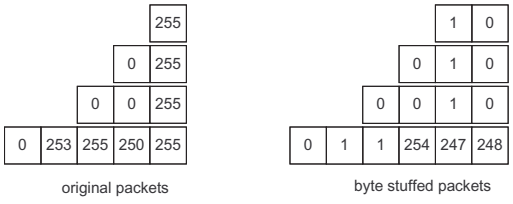


Fig. 3. Example of data packets and the corresponding BABS stuffed packets.

[1]If the byte to eliminate (the flag) is different from 255, then the base conversion operation should be followed by a convenient operation, such as LUT or XOR operations.

III. Theoretical Analysis of BABS

Let us address the theoretical performance of BABS. Denote by P a packet to stuff; size(P) the number of bytes of packet P; BABS(P) the stuffed packet; P^* the packet P normalized by removing the trailing zeros. The following properties should be easily grasped:

1) size(BABS(P)) − size(BABS(P^*)) = size(P) − size(P^*). The number of trailing zeros is the same, before and after stuffing.
2) size(BABS(P^*)) ≥ size(P^*). Normalized packets do not get smaller after stuffing.
3) size(BABS(P)) ≥ size(P). Packets do not get smaller after stuffing.
4) Let $P_N^{\max}$ be the packet corresponding to the biggest number representable with N characters. The value implicitly represented by $P_N^{\max}$ equals $\sum_{i=0}^{N-1} 255.256^i = 256^N - 1$. Then $N = \text{size}(P_N^{\max}) \Rightarrow \text{size}(\text{BABS}(P_N^{\max})) = M = \left\lceil N\frac{\log 256}{\log 255} \right\rceil$.
   That is because the biggest number representable with M characters in base 255 equals $\sum_{i=0}^{M-1} 254.255^i = 255^M - 1$. Then we must to have $255^M - 1 \geq 256^N - 1$ and the equality follows.
5) The byte overhead for a normalized packet with at most N bytes is less or equal than the overhead of the packet $P_N^{\max}$.
6) The byte overhead for a packet with at most N bytes is less or equal to the overhead of the packet $P_N^{\max}$.

Then, for packets' sizes less or equal to 1415 the BABS algorithm has an overhead of $\lceil 1415\frac{\log 256}{\log 255} \rceil - 1415 = \lceil 1415.999 \rceil - 1415 = 1$ byte in the worst-case. For packets' sizes between 1416 and 2831 the overhead is of 2 bytes in the worst-case.

It is also useful to calculate the expected performance for uniform random data, because data that is properly compressed and/or encrypted has a uniform distribution of byte values. It is possible to show (see appendix) that, for uniform random data, the expected size of the packet after BABS algorithm is, for $N \leq 1415$,

$$\text{mean}(M) = N + \frac{256}{255}\left(1 - 256^{-N}\right) - \frac{255}{254}\left(\left(\frac{255}{256}\right)^N - 256^{-N}\right) \quad (1)$$

And for $1416 \leq N < 2832$,

$$\text{mean}(M) = N + \frac{256}{255}\left(1 - 256^{-N}\right) - \frac{255}{254}\left(\left(\frac{255}{256}\right)^N - 256^{-N}\right) + 1 - \frac{255^{N+1}}{256^N} \quad (2)$$

In Table I is compared the expected overhead and the worst-case overhead for BABS, COBS[2] and PPP algorithms. Information theory tells us that for random data the average overhead must be at least $\frac{\log 256}{\log 255} - 1 \approx 0.07063\%$. This theoretical bound gives us a metric against which to judge

[2]The expected overhead of 0.2295% for COBS presented in [3] was derived for infinite-length packets.

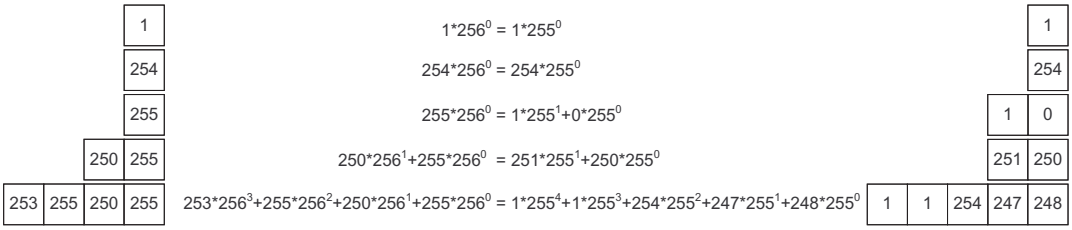


Fig. 1. Example data packets and the transformed packets after base conversion.

different byte stuffing schemes. Note that this is practically the performance achievable with BABS.[3]

N	mean(OH)			max(OH)		
	BABS	COBS	PPP	BABS	COBS	PPP
1	0.39062	100	0.78125	100	100	100
2	0.39062	50	0.78125	50	50	100
4	0.38948	25	0.78125	25	25	100
8	0.38665	12.5	0.78125	12.5	12.5	100
16	0.38078	6.25	0.78125	6.25	6.25	100
32	0.36927	3.125	0.78125	3.125	3.125	100
64	0.34756	1.5625	0.78125	1.5625	1.5625	100
128	0.30906	0.78125	0.78125	0.78125	0.78125	100
256	0.24817	0.53517	0.78125	0.39062	0.78125	100
512	0.16985	≥ 0.3008	0.78125	0.19531	0.58594	100
1024	0.09626	0.2295[4]	0.78125	0.09766	0.48828	100
1415	0.07067	0.2295[4]	0.78125	0.07067	0.42403	100
1416	0.07068	0.2295[4]	0.78125	0.14124	0.42373	100
1500	0.08546	0.2295[4]	0.78125	0.13333	0.4	100
2831	0.07065	0.2295[4]	0.78125	0.07065	0.42388	100

TABLE I
OVERHEAD: $OH = \frac{M-N}{N} \times 100\%$.

Observe the clear difference of performance between BABS and COBS, especially for small packets. For large packets BABS' performance converges to the theoretical optimum, while COBS stabilizes at about a $3\frac{1}{4}$ times worse performance. The worst case analysis is still more convincing, with BABS still converging to the optimum, while COBS' worst-case is roughly 0.4%. Comparing COBS with PPP it is apparent that for small packets COBS trades performance on the average for a tight control of the worst-case scenario. Unlike COBS, BABS attains an optimum average performance at an optimum worst-case behaviour.

IV. HYBRID BABS

One possible criticism of BABS stuffing is that its complexity may be unacceptable for some applications. In fact, the bandwidth efficiency of BABS is paid with a $\mathcal{O}(N^2)$ computing complexity, contrasting with the $\mathcal{O}(N)$ computing complexity of COBS and PPP. Noting that the biggest advantage of BABS over COBS occurs for small packets, where the constant overhead of 1 byte of COBS is much more noticeable, it is possible to formulate hybrid algorithms, keeping the best characteristics of both methods: excellent bandwidth efficiency with low computing complexity. Toward that end, an obvious solution is to stuff small packets with BABS and big packets with COBS. Adopting as threshold the value 255, a packet whose size is less or equal to 255 is encoded with BABS; packets with size larger than 255 are encoded with COBS. With pseudo-code the hybrid algorithm would come as illustrated in Listing 1.

```
Hybrid_Stuff()
{
      int threshold = 255;
      if(packet size <= threshold)
            BABS_encode ();
      else
            COBS_encode ();
}
```

Listing 1: Hybrid method encoding.

Note that, under this scheme, a packet encoded with BABS will have, at most, 256 bytes and a packet encoded with COBS will have, at least, 257 bytes. Then the hybrid encoding is inversely performed as listed with pseudo-code in Listing 2.

```
Hybrid_UnStuff()
{
      int threshold = 255;
      if(packet size <= threshold+1)
            BABS_UnStuff ();
      else
            COBS_UnStuff ();
}
```

Listing 2: Hybrid method decoding.

Adopting this hybrid scheme we control the computing complexity of the method (the threshold value can be adapted to the target computing capacity[5]), while maintaining a good bandwidth efficiency.

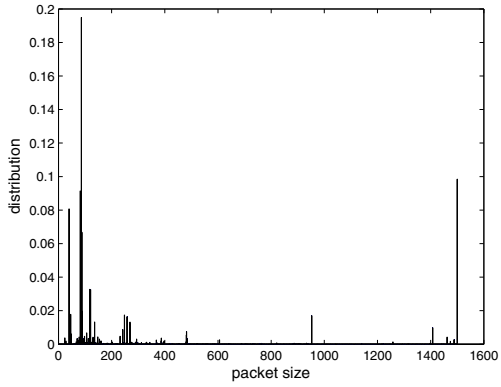
V. EXPERIMENTAL RESULTS

This section compares the stuffing of BABS with other protocols for real-world network traffic. We gathered a trace of network traffic using ethereal [7] and compared how efficiently these packets were stuffed by PPP, COBS, BABS and Hybrid BABS. Similarly to [3], the trace corresponds to a wireless

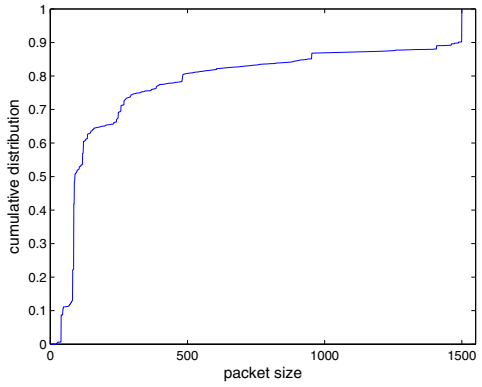
[3]Nevertheless, BABS does not perform optimally. Note that the packet with 1416 255's is coded with 1418 bytes, while the packet with 1417 0's is coded with 1417 bytes. That is, a less probable sequence is encoded with less bytes.

[4]Approximated value obtained from the infinite-length packet overhead.

[5]The threshold can be set as high as 1415 without ambiguity in the decoding process.



(a) Percentage distribution.



(b) Cumulative histogram.

Fig. 4. Histogram of packets' sizes.

traffic connection of a laptop during 3 days of normal use. The trace contains 88 544 packets, totalling 30 706 295 bytes of data. As visible in Fig. 4, the traffic consists of a mixture of small packets and full-sized packets (the MTU of the wireless interface was set to the default value of 1 500 bytes). Nevertheless most of the packets were not large; about 69% of the packets were shorter than 256 bytes. The results of this traffic trace are shown in Fig. 5.

PPP incurred a total overhead of 140 914 bytes (0.46%). PPP overhead is spread over a wide range, with a maximum of 47 bytes. Although this value is still far from the worst-case scenario, equipment implementing this protocol has to be designed for the worst-case scenario.

COBS incurred a total overhead of 139 160 bytes (0.45%), approximately the same overhead as PPP. On average, COBS does not show a clear improvement over PPP. However, COBS does succeed in controlling the worst-case behaviour, presenting a maximum overhead of 6 bytes.

Confirming the theoretical analysis, BABS attains a very low mean overhead — 44 639 bytes (0.15%) —, with only 2 bytes of overhead in the worst-case.

The byte overhead distribution for Hybrid BABS is essentially equal to COBS' distribution for overheads greater than 1 byte (corresponding mostly to packets with more than 255 bytes); however, for lower overheads Hybrid BABS is in advantage because most of the small packets do not see their size increased after stuffing — in opposition to COBS that always increases the size of the packet. BABS and Hybrid BABS presented systematically an overhead of 0 bytes for small packets.

A. Time complexity

The running time of the methods under evaluation were also compared. PPP and COBS stuffing take essentially the same time, confirming the $\mathcal{O}(N)$ complexity of both algorithms. BABS experienced a much higher computing time, approximately 400 slower. This a natural consequence of the $\mathcal{O}(N^2)$ complexity. Hybrid BABS recovered some of the computing efficiency, being only about 8 slower than COBS and PPP. As expected Hybrid BABS presents a trade off between the bandwidth-efficiency of BABS and the computing time of COBS.

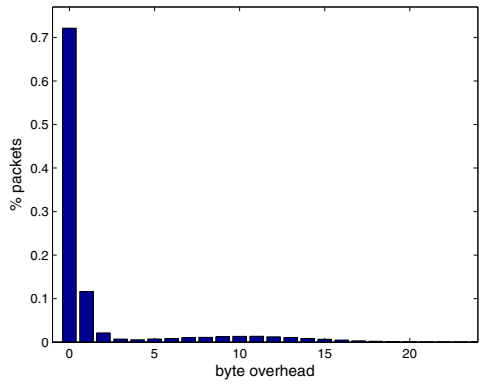
VI. Conclusion

In this paper we have introduced a new byte stuffing algorithm, BABS. The key idea for preventing the flag byte from being accidentally generated inside the payload is to convert the packet, interpreted as a huge number in base 256, to another packet written in base 255. This way, the encoded packet only makes use of 255 different characters.

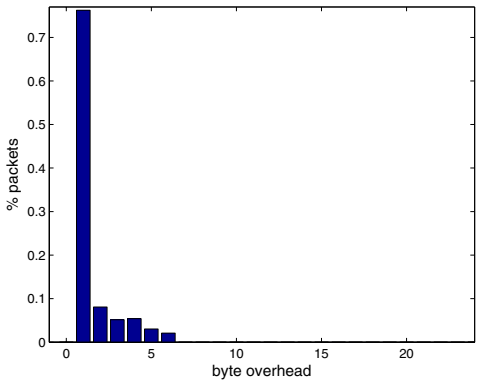
If it is true that the benefit of conventional PPP-like stuffing methods over COBS is established on the encoding of small packets with no overhead, that argument is no longer valid when comparing PPP with BABS. BABS presents the best average overhead with the minimum worst-case behaviour. The almost optimality of BABS in bandwidth-efficiency is penalized with a $\mathcal{O}(N^2)$ time complexity. Hybrid BABS mitigates this weakness with a balancing between bandwidth performance and computing complexity.

BABS is especially interesting for systems with more computing power than network bandwidth. Nonetheless, careful implementations or the development of specialized algorithms for converting between two consecutive bases may lessen the computational burden, enabling the deployment of the algorithm in a broader range of environments.

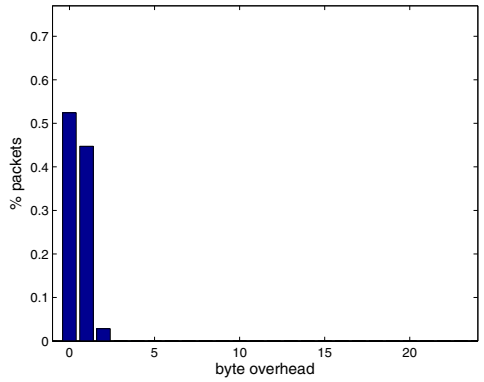
Although BABS was described in this paper to eliminate a single character from a packet, it is intrinsically adapted to eliminate one or more characters (a necessity when the start of transmission flag differs from the end of transmission flag). To eliminate two bytes from a packet, one just uses base 254 instead base 255 in the base conversion process. Likewise, it can the applied to other word lengths, other than eight-bit bytes.



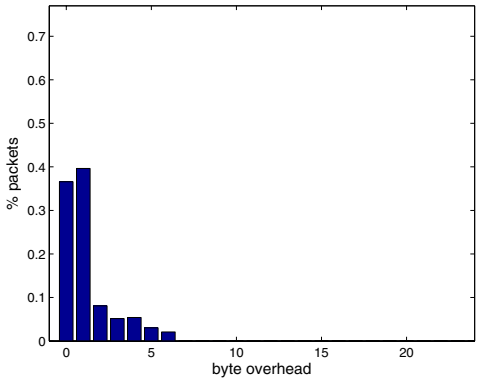
(a) PPP overhead. Maximum of 47 bytes, total of 140 914 bytes.



(b) COBS overhead. Maximum of 6 bytes, total of 139 160 bytes.



(c) BABS overhead. Maximum of 2 bytes, total of 44 639 bytes.



(d) Hybrid BABS overhead. Maximum of 6 bytes, total of 106 757 bytes.

Fig. 5. Stuffing overhead distribution.

APPENDIX I
EXPECTED PACKET SIZE AFTER BABS STUFFING

The biggest number representable with n characters in base 256 is $\sum_{i=0}^{n-1} 255.256^i = (256^n - 1)$. Likewise, in base 255 the biggest representable number is $\sum_{i=0}^{n-1} 254.255^i = (255^n - 1)$.

Then, any packet with n characters in base 256 corresponding to a number above $(255^n - 1)$ will need more than n characters in base 255.

Now, the number of packets with N characters that will have a nonzero overhead will be

$$\sum_{n=1}^{N} \left((256^n - 1) - (255^n - 1)\right) = \frac{256}{255}\left(256^N - 1\right) - \frac{255}{254}\left(255^N - 1\right) \tag{3}$$

because for $n < N$, appending $(N-n)$ zeros to a normalized packet of n characters that can not be represented with n characters in base 255, we obtain a packet with N characters that can not be represented with N characters in base 255.

For $N \leq 1415$, and because the maximum overhead is just one byte, the mean byte overhead is

$$\frac{1}{256^N}\left[\frac{256}{255}\left(256^N - 1\right) - \frac{255}{254}\left(255^N - 1\right)\right] \tag{4}$$

or, equivalently,

$$\frac{256}{255}\left(1 - 256^{-N}\right) - \frac{255}{254}\left(\left(\frac{255}{256}\right)^N - 256^{-N}\right) \tag{5}$$

For $1416 \leq N < 2832$ some of the packets that will have nonzero overhead (whose number is given by (3)) will have a overhead of two characters. The number of packets with two-byte overhead is

$$256^N - 255^{N+1}$$

Then it follows that the mean byte overhead is

$$\frac{256}{255}\left(1 - 256^{-N}\right) - \frac{255}{254}\left(\left(\frac{255}{256}\right)^N - 256^{-N}\right) + 1 - \frac{255^{N+1}}{256^N} \tag{6}$$

APPENDIX II
SOURCE CODE LISTING

It is presented in listing 3 a standard implementation of a base conversion scheme. A faster version of the same algorithm, using internally the fourth power of both bases to save on the number of operations, is available upon request to the author.

```
// BABS base adaptation byte stuffing
int BABS_Stuff_UnStuff(unsigned char* dataIn,
                       unsigned long lenIn,
                       unsigned char* dataOut,
                       UINT16 baseIn, UINT16 baseOut)
{   //BABS Stuff baseIn = 256 baseOut = 255
    //BABS UnStuff baseIn = 255 baseOut = 256
    int LimSup = lenIn-1;
    while (LimSup>=0 && dataIn [LimSup] == 0)
        LimSup--;

    int extraZeros = lenIn-1 - LimSup;
    int i;
    for (i = 0; LimSup>=0; i++)
    {
        UINT16 temp = 0;
        for (int j = LimSup; j >= 0; j--)
        {
            temp = temp*baseIn + dataIn [j];
            dataIn [j] = (UINT8) (temp/(baseOut));
            temp -= dataIn[j]*(baseOut);
        }
        dataOut [i] = temp;
        while ((LimSup>=0)&&(dataIn [LimSup] == 0))
            LimSup--;
    }
    for(; extraZeros>0; extraZeros--)
        dataOut [i++] = 0;
    return i;
}
```

Listing 3: BABS stuffing and unstuffing.

REFERENCES

[1] W. Stallings, *Data and computer communications*. Prentice Hall, 2004.
[2] W. Simpson, "The point-to-point protocol (PPP)," RFC 1661, 1994.
[3] S. Cheshire and M. Baker, "Consistent overhead byte stuffing," *IEEE Transactions on Networking*, vol. 7, pp. 159–172, Apr. 1999.
[4] J. Romkey, "Nonstandard for transmission of IP datagrams over serial lines: SLIP," RFC 1055, 1988.
[5] W. Simpson, "PPP in HDLC-like framing," RFC 1662, 1994.
[6] D. Knuth, *The Art of Computer Programming - Seminumerical Algorithms*. Addison Wesley, 1997, vol. 2.
[7] (2005) Ethereal. [Online]. Available: http://www.ethereal.com/