Network Working Group
Request for Comment: 468
NIC: 14742

R. Braden
UCLA/CCN

March 8, 1973

FTP DATA COMPRESSION

I. INTRODUCTION

APOLOGIA

Major design objectives of the proposed File Transfer Protocol (FTP) are reliability and efficiency for transmission of large files. Efficiency has two faces: efficiency of the host CPU's, and efficient use of the Network bandwidth. Block mode is intended to minimize CPU overhead for bandwidth efficiency, there is a mode called "HASP" in RFC 454. The "HASP" mode of FTP is really transmission with data compression, i.e., an encoding scheme to reduce the information redundancy in the messages.

RFC 454 contains no explicit definition of the "HASP" or compressed mode, but instead notes that a future RFC by yours truly will define the mode. Students of FTP may find this scarcely credible, but you are now reading the promised RFC. It turned out to be much farther in the future than any of us expected. Mea Culpa.

GENERAL CONSIDERATIONS

In the early years of the Network, its major uses have been remote terminal interactions and the small-to-medium-sized file transmission typical of remote job entry. As facilities such as the Illiac IV and the Data Machine become operational on the Network, and the Network community begins to include users with heavy data transmission requirements, large file transmission will become a major mode of Network use. For example, one user of CCN expects to send 2 x 10**8 bits of data _each_ _day_ over the Network.

Local byte compression of the type proposed here is particular effective for reducing the size of "printer" files such as those transmitted under the Network RJE protocol. Experience with CCN's RJS service has shown a typical compression of print files by a factor of between two and three. Since FTP was intended to contain the data transfer part of Network RJE protocol as a subset, it is appropriate to include a print file compression mechanism in FTP. These considerations led the FTP committee to include a compressed mode within FTP.

The two main arguments for data compression are economics and convenience (usability). Consider first economics, which is essentially a trade-off between CPU time and transmission costs. Of course, as long as Network use is a free commodity, the economics of data compression are all bad. That happy state won't last forever. What does data compression cost?

Let us consider only simple linear compression schemes, such as the one proposed here. By linear, I mean that the CPU time to examine a source record is proportional to number of bytes in the record. A simple linear scheme could detect repeated single characters, for example. One could imagine quadratic schemes, which detected repeated substrings; but except for possible special circumstance where the source stings have some structure known to the compression algorithm, the CPU economics don't favor quadratic compression.

Assuming a reasonable figure for large-scale CPU costs in the generation of CCN's 360/91, we concluded that an upper bound on CPU costs for total compression and decompression would be 5 cents per megabit; this is based on very loose coding of a simple linear algorithm. This may be compared with the projected Network transmission costs of over 30 cents per megabit (possibly a lot over).

Thus, the CPU time to conserve bandwidth costs significantly less than the bandwidth saved. Both CPU costs and bandwidth costs are trending downward, but it seems exceedingly unlikely that the ratio of CPU cost to bandwidth cost for linear compression will reverse in the next few years. On the other hand, this calculation clearly discourages one from using quadratic compression.

WHY HASP

CCN's batch remote job entry protocol NETRJS (see RFC #189, July 15, 1971) was designed to include two data transfer modes, truncated and compressed. The NETRJS truncated mode is essentially identical to current FTP block mode record structure (except for minor bit format differences). The compressed mode of NETRJS uses an adaptation of the particular compression scheme which is incorporated in the "Multileaving protocol" of the binary synchronous rje support in IBM's HASP system.

Although it isn't really necessary for the purpose of defining a compression scheme in FTP, I have included an appendix summarizing very briefly the nature of HASP and its rje package. That appendix may be considered cultural enrichment for those in the Network Community who have been denied the privilege of being an IBM customer. After all, I know a lot of HASP experts who never heard of

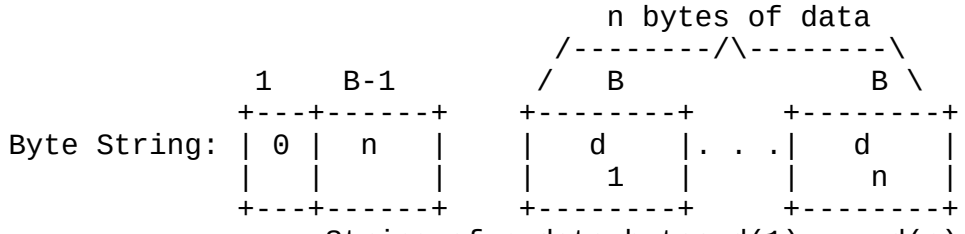
TENEX! More seriously, because HASP is widely used on IBM machines, the HASP compression scheme is also widely used. In designing NETRJS, we chose the HASP scheme of compression because of its ubiquity and plausibility.

However, certain details of the HASP bit formats are inappropriate or sub-optimal for FTP. Therefore, our proposal for compressed mode of FTP is only an adaptation of the HASP compression scheme.

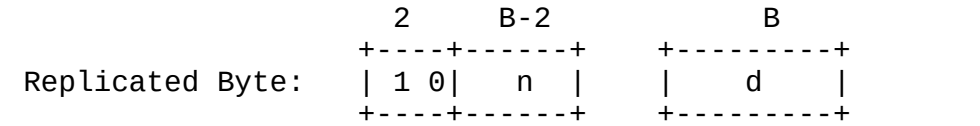
It should be clear from Appendix A that the compression scheme of HASP, even if used literally, is a very minor and incidental part of that system. Although we ought to properly credit the intellectual origin of FTP's compressed mode, it seems a little strange to call the compressed mode in FTP the "HASP mode". I trust this will be rectified by the forthcoming FTP meeting.

II. PROPOSED FTP COMPRESSED DATA MODE

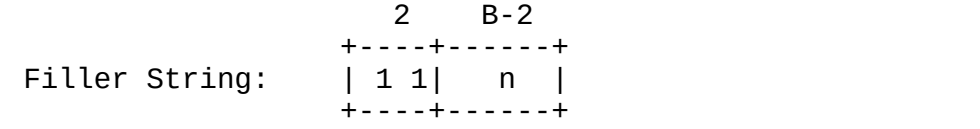
Byte size is B bits. Figures above boxes are field lengths in bits.



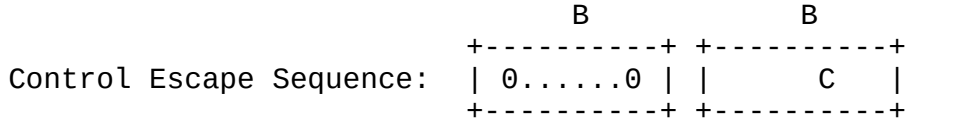
String of n data bytes d(1),...,d(n)
Count n must be positive



String consisting of n replications of the data byte d



String of n filler bytes. The filler byte is a "space" character for ASCII or EBCDIC type, or a binary zero byte for Image or Local Byte Type.



The control byte "C" which is the second byte of a control escape sequence is to have the same coding as the descriptor byte in Block Mode. This includes end-of-file and end-of-record indications. I will not specify this further because there is some question at present about the exact coding of the Block Mode descriptor byte.

Following the example of APL*, we have let the meaning of the filler (blank or 0) be determined by the type: character (ASCII|EBCDIC) vs. binary (Image|Local Byte). If byte size is equal to the word size of the transmitting host, the compressed mode allows one to send sparse notices with reasonable efficiency.

* Compare 1 (take) 0 1\‘A’ with 1 (take) 0 1\2

APPENDIX A: HASP MULTILEAVING

HASP (Houston Automatic Spooling Program) is a subsystem which essentially runs within OS/360 as a job but takes over the batch processing management functions from the operating system. That is, HASP handles spooling of card input and printer and punch output, queueing and scheduling of job execution, and the operator control interface. It is a tightly-written and efficient system for running a large and varied job load through a large-scale machine. The name results from the historical fact that HASP was originally by a local IBM group for one particular customer, NASA Houston.

HASP has always been an anomaly in the IBM scheme of things. The system was written around 1965 by two programmers; the HASP group has probably averaged three programmers during most of its life. The leader of the group has been "Mr. HASP", Tom Simpson. The HASP system spread rapidly through (more or less) underground channels to many of the medium and large scale 360's. At least once, only intense customer pressure prevented IBM from killing the HASP effort. HASP generated an astonishing emotional mystique among its users. The HASP sessions at SHARE Meetings were reminiscent of revival meetings. For years every SHARE Meeting has included HASP song sessions around the piano during the nightly open bar. HASP forms a fascinating chapter in the history of IBM's large machine business.

The core concepts in HASP are pseudo-devices, and the general technique of intercepting supervisor calls to augment operating system functions without changing the operating system itself. A generation of OS/360 system programmers learned these techniques from HASP. (These important techniques are hardly ever described in the literature, and "practical" programmers don't read the literature anyway).

When HASP starts up (in supervisor state), it overlays an instruction in the I/O Supervisor with a branch to its own code. A user program is written as if it were doing real I/O to card readers and printers. HASP intercepts and interprets these I/O operations to handle job I/O in a manner transparent to OS/360. It similarly intercepts and interprets operator console I/O.

HASP includes batch remote job entry using binary synchronous communication. The HASP communication protocol and message formats use a scheme developed by Simpson's group called "Multileaving Protocol". The HASP rje system, by far the best rje package IBM has produced, finally replaced two competitive IBM packages and has effectively become the IBM standard for rje. CCN's RJS system not only adopted the Multileaving Protocol but essentially copied its binary synchronous communication line handler directly form HASP.

The Multileaving Protocol is described in the HASP manual(1) as the "fully synchronized, pseudo-simultaneous, bidirectional transmission of a variable number of data streams between two or more computers using binary synchronous communications facilities". It allows a remote batch terminal to operate a variable number of card readers and printers simultaneously at different speeds over one communication line. It is not surprising that HASP Multileaving contains in miniature many of the features of IMP-IMP Protocol and a little host-host protocol. Specifically, Multileaving includes the following general features:

(1) "Conversational" transmission line protocol using transparency (DLE STX, etc.).

(2) "Strong" error control and retransmission using a 16-bit CRC and a modulo-16 block sequence number.

(3) Flow control for multiple streams in both directions. This includes the interchanging of matching control records ("RFC's") to open a stream, and a set of flow control bits in each block. Each flow control bit is logically equivalent to an ALLOcate command for one "message" (buffer) for a particular stream.

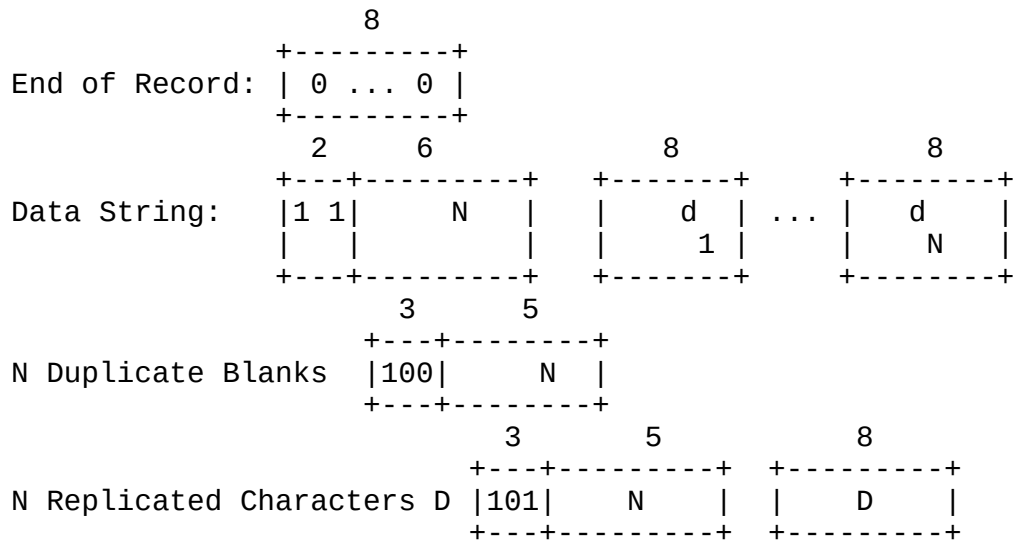
(4) Optional Special Control Information for remote devices. This includes printer carriage control, switching card reader hoppers, etc.

(5) Multiplexing ("multileaving") multiple streams into a single block for transmission.

(6) Marking end of file and ends of records within each stream.

(7) Compressing transmitted text by encoding repeated blanks and repeated single characters.

Finally, we have reached the (only) aspect of HASP relevant to FTP: its compression scheme. HASP uses the following encoding:



HASP is concerned only with 8-bit bytes. However, there is a provision (which was never implemented) in the Multileaving Protocol to set the unit of the counts N as 1 byte, 2 bytes, or 4 bytes.

Reference:

(1) HASP II System Manual, IBM Corporation (February 26, 1971)