



COPYRIGHT NOTICE



© 1992 IEEE. Personal use of this material is permitted. However, permission to reprint/republish this material for advertising or promotional purposes or for creating new collective works for resale or redistribution to servers or lists, or to reuse any copyrighted component of this work in other works must be obtained from the IEEE.

This material is presented to ensure timely dissemination of scholarly and technical work. Copyright and all rights therein are retained by authors or by other copyright holders. All persons copying this information are expected to adhere to the terms and constraints invoked by each author's copyright. In most cases, these works may not be reposted without the explicit permission of the copyright holder.



The V.42bis Standard for Data-Compressing Modems

The recently adopted CCITT V.42bis standard for data-compressing modems is a conservative and economically implementable scheme, one discussed here from algorithmic, experimental, practical, and marketing standpoints. V.42bis compresses text about as well as the Lempel-Ziv-Welch algorithm of the Berkeley Unix Compress utility; other Ziv-Lempel variants are discussed briefly. Even though better compression ratios can be obtained with these variants, V.42bis is eminently suitable for implementation on a contemporary modem.

Clark Thomborson

University of Minnesota
at Duluth

Many factors can affect modem throughput. Important factors include the protocols used for data compression, telephone signaling, error correction, and computer-to-modem transmission. To overcome some of these problems, the CCITT V.42bis standard details a textual substitution scheme that allows a well-implemented modem to achieve a 2.3:1 compression ratio on English text. A modem implementing this standard delivers higher bandwidth, on compressible files, than a noncompressing modem.

Two major V.42bis modem applications are terminal-to-computer and computer-to-computer communications over the telephone network. However, text transmission is only a fraction of current computer communication and this fraction is rapidly decreasing. Better algorithms for compressing text (and other types of data) are being developed, but I believe V.42bis will become the dominant method for text compression for at least the next 10 years.

Background

Briefly, the V.42bis data compression method is a textual substitution scheme based on Ziv and Lempel's second algorithm.¹ V.42bis incorporates Welch's coding scheme² as well as Miller and Wegman's³ (and, independently, Storer's^{4,5}) idea

for incremental modification of a full dictionary using a least recently used (LRU) heuristic. V.42bis also has an efficient cleartext escape mechanism, differing in implementation but not in spirit from that proposed by Fiala and Greene.⁶ (See the Standards box for more information.)

In 1990 users could buy a modem with the following characteristics:

- Price: \$670 (discounted) to \$1,150 (list)
- Modem-to-terminal or -computer connection: EIA-232-D (300 to 38,400 baud)
- Modem-to-telephone network connection: V.32 (300 to 9,600 bps)
- Error correction: V.42 (16-bit CRC with retransmission)
- Data compression: V.42bis (modified Ziv-Lempel)
- Implementation: 8-bit microprocessor (10-MHz Z80) with a 40-Kbyte RAM and 64-Kbyte ROM.

I will call such a device a V.42bis modem, although I caution the reader that other uses of the V.42bis standard are possible. The implementation and pricing information correspond to a MultiTech MT932EAB, introduced in late July 1990. Other manufacturers offered six similar products, according to an informative set of product reviews.⁷

Standards

I encourage expert readers to look at Byrd's article on V.42bis modems⁷ and/or Black's discussion of the EIA-, X-, and early V-series standards.⁸

Unfortunately, standards documents are very difficult to read, even for an expert in the field. Even more obtuse, but nonetheless important, are the US patents granted in the last decade for data compression methods. These patents appear to cover implementations of many of the algorithms I discuss. Michael Ernst of MIT, memst@theory.lcs.mit.edu, has collected a list of such patents. If you have Internet access, you may retrieve his list by anonymous ftp from mintaka.lcs.mit.edu: /mitlpf/ai/patent-list.

One year later, the first V.32bis modems became commercially available. All such modems include the V.42bis data compression feature, so it is superfluous to call them V.32bis/V.42bis modems. The advantage of a V.32bis modem over the older V.42bis modem is that the V.32bis protocol provides higher (14,400 bps) raw bandwidth on the telephone network than the older, slower 9,600-bps V.32 protocol of a V.42bis modem. A V.32bis modem can deliver 50 percent greater throughput than a V.42bis modem. However, as noted in the adjacent Terms box and elsewhere, telephone network bandwidth is only one of many possible bottlenecks.

PC World's December 1991 product reviews⁹ listed V.32bis modems at \$425 to \$1,345, while V.42bis modems cost from \$399 to \$1,395. The wide range of pricing is remarkable. The name-brand manufacturers charge the higher prices for their latest models. Available, however, are deep discounts, especially on discontinued lines. One distributor (Damark) of-

Terms

baud: The bandwidth of a communication channel, expressed in units of symbols per second. Baud is frequently assumed to be equivalent to bps, but this is not always true. If the symbols are taken from a binary alphabet, 1 baud is indeed 1 bps. However, if the symbols are taken from a 16-valued set, each signal carries four bits, and 1 baud is 4 bps.

BCH coding: Any of a variety of error-correcting codes based on work by Bose, Ray-Chaudhuri and Hocqneunghem. Reed-Solomon codes form a related class. These codes require much more computational power than the simple CRC-check-and-retransmit method of V.42. However, they impose much less overhead on the communication channel. Some of these codes can correct huge numbers of errors without retransmission, making them invaluable in satellite communication and other applications where errors are frequent and retransmission is infeasible.

bis: A suffix indicating a modified standard, as in V.32bis or V.42bis. Derived from the Latin *bis* meaning twice.

bps: Abbreviation for bits per second, a commonly used unit for expressing the bandwidth of a communication channel.

CCITT: Consultative Committee on Telephony and Telegraphy, a Geneva-based division of the International Telecommunications Union, a New York-based United Nations organization.

cleartext: Uncompressed data, often assumed to be English words in ASCII code.

cps: Bandwidth of a communication channel in (8-bit) characters per second. In this article, cps figures include

the overhead of timing and error control. Moreover, they are adjusted for data compression, so an 8-bps connection provides somewhat more or less than 1 cps of effective bandwidth.

CRC: Cyclic redundancy code, usually a 16-bit signature obtained by a simple hashing algorithm on keys with hundreds of characters. CRC forms the basis of V.42 error control: A modem sends a few hundred characters (or less, if there is not much data to send), followed by a 16-bit CRC. The receiving modem computes a CRC on the data characters it receives, comparing it to the CRC it receives. If the two CRC values are the same, it is unlikely that any transmission errors occurred.

EIA-232-D: An internationally recognized standard for the electrical and mechanical connections between a terminal and a modem, or between a computer and a modem. Data transmits bit-serially, typically in an asynchronous format of 8-bit characters with 2 or 3 timing bits per character. Other formats are possible. The maximum recommended bandwidth is 19,200 baud, or about 1,920 cps in an asynchronous format. Most V.42bis modems, and some high-speed terminals and computers, will communicate reliably over an EIA-232-D connection at 38,400 baud, or 3,840 cps. Some modems can communicate at 57,600 baud using an EIA-232-D connection to a special board that plugs directly into an IBM PC bus. This higher priced option, using a synchronous format for character transmission with little timing overhead, can provide 6,000 cps of bandwidth.

hash trie: A trie in which each node contains just one pointer, referencing the node's ancestor. The nodes are stored

ferred a V.42bis modem, with limited supply, for \$270. The overlapping price ranges for V.32bis and V.42bis modems indicate that the V.32bis modem is well on the way to supplanting the V.42bis modem, just as the V.42bis modem recently supplanted an earlier, non-V.42bis generation of high-speed modems.

Modem applications, performance, pricing

Terminal-to-computer and computer-to-computer communications employ the telephone network, either with a dial-up connection or a leased line.

The distinguishing feature of a V.42bis modem is that it can deliver higher bandwidth, in most applications, by the use of a good algorithm for data compression. Experts may wish to read the V.42bis Compression box on the next page for detailed information on this algorithm. Nonexperts need know only that the performance of a V.42bis modem will vary with the compressibility (or incompressibility) of the

data being transmitted. For example, approximately a two-fold improvement is typical in an application involving English text. Thus, when transmitting English text, a V.42bis modem will deliver somewhat more than 2,000 characters per second, up from about 1,000 cps on a V.42 modem. If relatively incompressible data is being presented to the V.42bis modem, or if some bottleneck exists in the system other than the modem, the V.42bis modem will not deliver performance improvement. On highly compressible data, such as spreadsheets, V.42bis data compression can deliver up to a fourfold improvement in bandwidth. Will this bandwidth improvement be achieved in a given application? Read on to find the factors, other than data compressibility, that can limit modem performance.

User's desired bandwidth. In a terminal-to-computer application, a user typically accesses a remote computer with a terminal in an office or home. All the user's keystrokes are

continued on p. 46

Terms (continued)

in an array, allowing rapid access if a node's index is known. Searching for a string in a hash trie involves hashing the string to find a set of possible indices for its node, then testing (by traversing the ancestor chains) whether one of the indexed nodes represents the input string.

hashing: The digital scrambling of a key (a series of data bits) into a signature (a shorter series of bits). The best hash functions give distinct signatures for almost all commonly encountered keys.

modem: A device performing the modulation/demodulation function necessary to transmit digital signals over an analog connection such as a telephone line.

Patricia trie: Binary trie in which each node contains an integer index and two subtree pointers. The integer is the index of the leftmost bit in which all keys in the left subtree differ from all keys in the right subtree. Complete keys are stored at the leaves of the tree.

pointer-list trie: A trie in which each node contains pointers to siblings (nodes at the same level), one or more descendants, and possibly to that node's ancestor. Searching for a string in a pointer-list tree involves the traversal of sibling lists and the following of descendant pointers.

trie: A tree-shaped data structure for the storage and reTRIEval of elements of a set of character strings. Typically, each node in a trie contains one character of data, namely the first character of all the strings representing that node's descendants. A trie node must also contain one or more pointers to other trie nodes, to define the connectivity of the tree and let the trie be traversed efficiently.

V-series interface: Any of a number of protocols for data communications over the telephone network.

V.32: An internationally recognized standard for sending digital information over a telephone connection with a modem. The maximum bandwidth is 9,600 bps. Each signal carries 4 bits; the signaling rate is $9,600/4 = 2,400$ baud.

V.32bis: A recent revision of V.32, allowing 14,400 bps.

V.42: A standardized method for error control over a V-series interface, providing for retransmission of information in case of error. This protocol typically has about 12 percent overhead, according to my MultiTech manual, so a 9,600-bps V.32 connection with V.42 error control has an effective bandwidth of $(9,600/8)/112$ percent = 1,060 cps. The overhead of the V.42 protocol will be larger on a noisy telephone line, for example, on an overseas connection, so the available bandwidth will actually be less than 1,060 cps.

V.42bis: A standardized method for data compression on a V-series interface. Single-bit errors can have catastrophic results, so V.42bis is normally used in conjunction with the V.42 method for error control. On English text that is compressible at 2.3 to 1, a 9,600-bps V.32 connection with V.42bis has a bandwidth limit of $(2.3) * (9,600/8)/112$ percent = 2,440 cps. A 14,400-bps V.32bis connection with V.42bis data compression has a bandwidth limit of about $(2.3) * (14,400/8)/112$ percent = 3,700 cps on English text. Other types of data give rise to different compression ratios, and thus to different bandwidth limits.

sent to the computer. The computer's response to a keystroke is variable: Sometimes no data is sent, sometimes a full-screen update of 2,000 characters is transmitted. Ideally, the computer's response to a user's keystroke would be displayed on the user's terminal within 0.1 second or so.

The desired peak bandwidth is thus about 2,000 characters in 0.1 second, or 20,000 cps, from the computer to the terminal. Much less data flows in the other direction, since few users type more than 10 characters per second. However, a terminal with programmable function keys can transmit 100 or more characters as a result of a single-user keystroke, for a peak (desired) bandwidth exceeding 1,000 cps.

I see no sign that these high-bandwidth standards are displacing the ubiquitous EIA-232 interface, nor have I seen any terminals with high-speed built-in modems.

Host computer bandwidth. When characters are sent from the computer to the terminal, an application program, such as an editor, running on that computer typically generates the characters one at a time. The application program may call an operating system routine to transmit each character. Alternatively, it may make one operating system call for each line of 64 characters, or possibly just one call per screenful of about $25 * 80 = 2,000$ characters. Obviously, the character-by-character processing in the application software, and in the operating system, is a potential communication bottleneck. If the application is complicated, if the computer is not very fast, or if the computer is multiprocessing many applications simultaneously, it may be unable to support the 20,000-cps bandwidth peaks desired by the user.

Computer-to-modem bandwidth. The next step in transmitting data from the computer to the user's terminal is for the operating system to send the character, or block of characters, to a modem attached to the computer. This modem then transmits the characters to the modem connected to the user's terminal, which then sends the characters to the user's terminal. Finally, the user's terminal displays the characters. Each of these steps is potentially a bottleneck. For example, there may be a low-bandwidth connection between the computer and its modem, a low-bandwidth modem-to-modem

connection, or a low-bandwidth modem-to-terminal connection. And the user's terminal may be incapable of accepting and displaying characters as rapidly as they are sent.

At my branch of the University of Minnesota, our campus mainframes and workstations are networked to a bank of modems. This computer-to-modem link has an upper limit of about 1,000 cps. One reason for this limit is applicable to many personal-computer-to-modem connections as well: Every character transmission causes an interruption of the computer's CPU. Most computers cannot handle more than about 1,000 interrupts per second.

Modem-to-modem bandwidth. The modem-to-modem connection is the easiest to specify and analyze. A V.42bis modem can send about 1,060 cps of uncompressed data on a typical telephone connection. If 2.3:1 compression is achieved, as is typical on English text, the available bandwidth rises to about 2,440 cps. However, if the telephone connection is impaired (but still within regulatory limits), the bandwidth will drop by 30 percent or more,⁷ or about 1,600 cps for text compressible at 2.3:1.

Modem-to-terminal bandwidth. This bandwidth will present a bottleneck in some cases. Typically, the interface is an EIA-232-D, which was originally designed to support at most 1,920 cps. Note that this is significantly lower than the 2,440-cps bandwidth of a V.42bis modem-to-modem connection when English text is transmitted. Fortunately, most V.42bis modems, and some high-speed terminals and computers, can communicate reliably at 3,840 cps, or more, over an EIA-232-D interface.

The EIA-232-D standard is a recent update of the venerable RS-232C convention, which itself has been updated several times. Yet another revision is in the works, but the document describing EIA-232-E was unavailable at press time.

Another way to obtain a higher modem-to-terminal bandwidth would be to avoid the EIA-232-D interface between terminal and the modem. One could build the modem into the terminal itself, in much the same way as an internal modem is attached to a bus in a personal computer. Alternatively, one could use a high-speed terminal-modem interface such as the 10-Mbaud EIA-422-A or the 100-Kbaud EIA-423-A. However, I see no sign that these high-bandwidth standards are displacing the ubiquitous EIA-232 interface, nor have I seen any terminals with high-speed built-in modems.

Terminal display bandwidth; flow control. Most modern terminals can receive and transmit short bursts of information at 3,840 cps but are unable to maintain this bandwidth for an extended period of time. Then, the following flow control mechanism is invoked to avoid losing data. When a terminal's input buffers become nearly full, it sends a pause signal to the modem. The modem's data buffers will then start to fill up, eventually causing it to send a pause signal to the transmitting modem. The transmitting modem's output buffers will then start filling up, possibly causing it in turn to

send a flow control signal to the transmitting computer.

A similar set of signals enables the transmission of data to be reinitiated between any pair of components in the path between the application program and the user's terminal. If the flow control mechanism has low latency, or if it is not invoked very often, this mechanism permits the transmission to proceed at the minimum bandwidth between any pair of components.

Performance gain from a V.42bis modem. As may be gathered from the foregoing discussion, it is not easy to know if the data compression feature of the V.42bis modem will result in a perceptible increase in performance over a noncompressing V.42 modem. V.42bis offers higher bandwidth for compressible data than does V.42, but unless the telephone connection is the bottleneck, there will be no increase in end-to-end bandwidth.

My experience may not be atypical: I have never observed more than 1,000-cps bandwidth on full-screen updates of a text editor on my home terminal (a Wyse-185) when it is connected with a V.42bis modem to any of the computers at my university. Since this bandwidth is below the 1,060 cps provided by V.42, I see essentially no benefit from V.42bis. One problem is that the connection from my workstation to my university's modem bank has a low bandwidth. I could sidestep this connection by installing my own modem on one of my workstations, or I could ask for an improvement in my workstation's connection to the modem bank, but frankly, this seems like more trouble than it is worth. I can tolerate a 1-second display time for a 1,000-character screen of text, and I am not willing to spend very much time or effort in improving this to 0.3 or 0.5 seconds.

However, some users have seen notable performance gains from V.42bis. High-speed personal computers with high-performance EIA-232-D ports exchange highly compressible (for example, spreadsheet) data at speeds approaching the 3,840-cps limit of the EIA-232-D interface.⁷ This nearly fourfold improvement over the 1,060-cps limit of a V.42 modem yields sizable cost savings when transmitting long files over a long-distance telephone connection. Even over a local telephone connection, there would be sizable equipment and personnel savings in any application that involved transmitting long, compressible files. A megabyte of highly compressible data might be transmitted in as little as four minutes over a V.42bis connection, instead of 15 minutes on a V.42 connection.

Data-compression software. From the preceding paragraph, one might assume that any application involving computer-to-computer communication of compressible files will benefit from V.42bis. This is not the case. The transmitting computer could run a compression algorithm on its data before sending it to a noncompressing V.42 modem; the receiving computer can run the corresponding decompression algorithm. This approach permits the user to select the compression method most suitable for the data being transmitted.

If a user transmits scanned image data, for example, bypassing the V.42bis feature helps more (10:1 compression is routine). Another advantage of compressing data before transmitting it to the modem is that an EIA-232-D connection is no longer a bottleneck: It is perfectly capable of supporting the 1,060-cps bandwidth of a noncompressing V.42 modem.

There are two disadvantages to this software data compression scheme. Someone must install and maintain appropriate software on both computers. In addition, the compression software will compete with the application software for CPU time (and memory space) on both computers.

***Each manufacturer must recoup
its engineering costs, marketing
costs, and approximately \$40,000
in V.42bis patent licensing fees.***

The V.42bis market. Due to the factors just discussed, the market for V.42bis modems is limited. In computer-to-computer applications, software data compression might offer better performance. In computer-to-terminal applications, the modem might not be the bottleneck. Thus there should not be a large price differential between V.42 and V.42bis modems. Indeed, one of the challenges to the V.42bis standards definition team was to develop a very low-cost data compression method.

In mid-1990, shortly after the first V.42bis modems hit the market, a typical V.42 modem listed for \$800 or more, but could be purchased for just over \$600. Additionally, about \$50 was charged for the V.42bis feature. In the case of the MultiTech product line, this feature was a minor modification to an existing V.42 modem. The modification involved adding another 8-Kbyte RAM chip and adding code to the ROM for the data compression software. The existing Z80 microprocessor apparently handled the extra CPU load of running the V.42bis data compression algorithm in addition to the V.42 error-correction protocol. The manufacturing cost of adding V.42bis to an existing V.42 modem was thus a few dollars. The profit margin on V.42bis might seem enormous to some readers, however each manufacturer must recoup its engineering costs, marketing costs, and approximately \$40,000 in V.42bis patent-licensing fees.

In a completely redesigned modem, the V.42bis feature is extremely low in production cost. Such a modem would have a 16- or 32-Kbyte RAM chip instead of the 8-Kbyte RAM chip in a V.42 modem. It might also have a slightly larger ROM to

accommodate the data compression software.

It is thus easy to understand why there were so few high-speed modems on the market in late 1991 that do *not* include the V.42bis feature. At worst (on incompressible data or on bandwidth-limited systems), V.42bis doesn't help; at best, it offers almost a fourfold improvement in bandwidth at a nominal increase in cost.

***It will soon become rare to
buy a high-speed modem
without BCH error control.***

V.32bis modems. The higher bandwidth V.32bis was recently approved by the CCITT, improving the raw bandwidth of a V.32 modem-to-modem telephone connection by 50 percent from 9,600 bps to 14,400 bps. Assuming the overhead of V.42 error correction remains at about 12 percent for this higher bandwidth connection, a V.32bis modem with V.42 error correction has a throughput limit of about $(14,400/8)/112\% = 1,610$ cps on uncompressed data. This bandwidth limit would rise significantly to 3,700 cps on data that is compressible at 2.3:1 when the V.42bis data compression feature is used.

Note that a V.32bis modem has no advantage over a V.42bis (V.32) modem whenever the V.32 modem-to-modem connection is not a bottleneck. This case is likely to occur whenever highly compressible data is being transmitted.

Measured bandwidths on real systems. No modem will run at its limiting bandwidth in a real system, with its nonideal flow control, limited buffer sizes, and slow EIA-232-D interfaces. For this reason, a number of researchers have constructed systems consisting of two PCs, two modems, a real or simulated telephone connection, and a collection of test files. As I discovered recently, the convention in this field is to measure bandwidths, in bits per second (bps), at the EIA-232-D interface.

For example, Glass measured throughputs of 22,000 bps on a text file with a V.32bis/V.42bis modem, and 16,000 bps with a V.32/V.42bis modem.⁹ Since 10 bit-times on the EIA-232-D interface are required to send an 8-bit character, Glass's measurement translates into *data* throughputs of 2,200 cps and 1,600 cps. These throughputs are about 40 percent lower than the ones I calculate here for text-file transmissions on these modems, leading me to conclude that Glass's system is bandwidth-limited somewhere other than in the modem-to-modem connection. My conclusion is supported by the fact that Glass measured only 24,000 bps (2,400 cps) on highly compressible database files, no matter which modem he used.

This is only 63 percent of the bandwidth limit of his EIA-232-D connection which, ideally, would run at full bandwidth on a highly compressible file transmission.

Using a different test system, Henderson measured 24,000-bps throughput on a text file transmitted on a V.32/V.42bis modem.¹⁰ This data throughput of 2,400 cps very closely agrees with the 2,440-cps bandwidth limit for V.32/V.42bis text transmissions (assuming 2.3:1 compressibility and 12 percent overhead for error correction). I conclude that Henderson's system does not suffer from bandwidth bottlenecks, other than the modem-to-modem connection under test.

The future. Another development in telephone network signaling, hovering on the near-future horizon for many years, is the Integrated Services Digital Network (ISDN). When and if local telephone loops are digitized, dial-up connections will run at 8,000 cps, without a modem. This is over twice the bandwidth of EIA-232-D, so this standard will finally be obsolete. One might make ISDN file-transfer connections with a special interface board on a personal computer, implementing V.42bis data compression and V.42 error correction. The result would be an effective bandwidth of nearly 20,000 cps for English text and 13 Kbytes/s for executables.

Even though digital signaling (ISDN) may never become widely available, other methods of analog signaling on the telephone network may become standardized soon. The 14,400-bps signaling of V.32bis has been surpassed by V.fast transmissions at 19,200 bps, and even 24,000 bps. Standardization of V.fast is expected soon. In conjunction with V.42bis data compression, data throughputs of up to 7,000 cps for English text are thus feasible. The slow EIA-232-D computer-to-modem interface becomes even more problematical at these data rates.

As computational power and memory become cheaper, more powerful methods for data compression and error control will no doubt become commonplace in the modem market. V.42bis could be revised to take advantage of recent work on data compression algorithms. Also, the V.42 error control protocol could be revised to incorporate a BCH or Reed-Solomon code that would greatly reduce the bandwidth and latency problems associated with V.42's retransmission after each single-bit error. One referee asserts that a BCH encoder/decoder chip would cost just \$50 at present, if purchased in quantity. With the price multipliers typical of a high-technology field, this would increase the cost of a modem by perhaps \$200. However, the cost of a BCH chip will no doubt drop rapidly in the next few years, possibly to the point where it will soon become rare to buy a high-speed modem without BCH error control.

Experimental results

John Copeland, of the V.42bis study group, kindly sent me C-language source code for an implementation of the V.42bis standard. The comments and experimental results in this sec-

tion are obtained from his version 6.83, dated July 16, 1989, of the hv42b3.c code. Hayes Microcomputer Products, Inc. owns the copyright to this code, which it says is "distributed for experimentation aimed at developing a CCITT V.42 data compression standard."

The Hayes code implements the V.42bis dictionaries as rchild/l sibling/rsibling/parent pointer tries. It sorts sibling lists by their suffix character.

When I ran the Hayes code on my Sun 3/50, it compressed my 100-Kbyte test files at rates of 6 to 10 Kbytes of input data per second. On a 10-MHz Z80, I estimate that the Hayes code would run two to four times slower, due to the lower MIPS rate and the narrower data bus. Some optimization of the code may thus be necessary to keep the CPU from becoming the bottleneck in a V.42bis modem, especially since the CPU in such a modem has a number of other tasks. The CPU must handle V.42bis compression on the transmission channel, V.42bis decompression on the receiving channel, as well as V.42 error correction and V.32 packetizing on both channels. (Perhaps the ATI 9600etc/e modem has a CPU bottleneck of this sort, which would explain why it was unable to deliver more than 1,977-cps throughput, on an experimental setup where its fastest competition ran at 3,440 cps.)⁷

I have run a number of compression measurements on the V.42bis implementation used by the study group. All measurements reported involve the following five files. The first, labeled csh, is the 122,880-byte executable C shell in my Sun 4.2 Unix. The second test file, Essays, consists of 176,369 bytes of concatenated freshman essays. It is cleartext English, containing no text-formatting commands. The third file, News, is 210,931 bytes of articles representing a (small) portion of a day's Usenet feed. This file is a good test of a compression algorithm's ability to respond quickly to a rapidly changing source, since the article header fields are quite different from the article bodies, and the article bodies are also variable (ranging from cleartext English to source code). The fourth file, Pascal, is 119,779 bytes of source code to a student-written program and is highly compressible, due to its repeated use of long identifiers. The fifth and final test file, Tex, is the 201,746-byte Latex manual, vintage 1984, containing both cleartext and text-formatting commands.

Table 1 shows the compression ratios obtained by the Hayes code on the five test files, when the length of the longest code word is 16 and the maximum length of the dictionary varied between 512 and 4,096. In this table, and throughout this section, I define compression ratio to be the length of the compressed file divided by the length of the input file. A compression ratio of 0.43 is thus the 2.3:1 compression I've already mentioned as the performance of V.42bis on a sample text file, Essays.

Note that increasing the size of the dictionary noticeably affects the compression ratio for all files save csh. Perhaps the most striking feature of Table 1, however, is that even a short,

Table 1. Effect of dictionary size on V.42bis compression ratio, when maximum code word length is 16.

Test files	Dictionary size			
	512	1,024	2,048	4,096
Csh	0.65	0.62	0.62	0.62
Essays	0.57	0.49	0.45	0.43
News	0.70	0.64	0.60	0.56
Pascal	0.52	0.41	0.35	0.31
Tex	0.59	0.50	0.45	0.42

Table 2. V.42bis compression ratio, for various dictionary sizes, when maximum code word length is 6.

Test files	Dictionary size			
	512	1,024	2,048	4,096
Csh	0.65	0.63	0.63	0.64
Essays	0.57	0.49	0.46	0.43
News	0.70	0.64	0.60	0.57
Pascal	0.53	0.43	0.38	0.35
Tex	0.60	0.51	0.46	0.44

512-code word dictionary is sufficient to obtain good compression on the test files. I've presented results only for test files long enough that large dictionaries may become useful. On short files, short dictionaries do just as well as long dictionaries.

Table 2 is identical to Table 1, except that the length of the longest code word is restricted to 6 characters. These figures are only slightly larger than those of Table 1, indicating that the code word-length parameter is not very important to V.42bis performance, at least for the five test files.

Table 3 on the next page shows the compression ratios obtained by the Unix Compress utility² on the five test files, as a function of dictionary size. Compress is a poor algorithm for small dictionaries. Even with a 64K dictionary, it is only slightly better than V.42bis with a 4K dictionary.

Tables 4 and 5 compare the performance of the Hayes code with a 4K-code word dictionary (maximum code word length is 16) with that of a number of other Ziv-Lempel variants. Where possible, I use the abbreviations of the Fiala-Greene work.

Table 4 consists of those algorithms which, like V.42bis, send code word indices in straight binary form. The algorithms of Table 5 gain compression, at some penalty in speed, by using Huffman or arithmetic codes to compress the code

Table 3. Unix-Welch compression ratio, for various dictionary sizes.

Test files	Dictionary size				
	512	1,024	2,048	4,096	65,536
Csh	0.98	0.82	0.89	0.75	0.64
Essays	0.67	0.53	0.49	0.46	0.40
News	0.91	0.73	0.71	0.67	0.54
Pascal	0.70	0.50	0.43	0.37	0.30
Tex	0.85	0.61	0.53	0.48	0.39

word indices.

Algorithms under text. Expert readers may be interested in the following paragraphs; others may skip to the Discussion section.

LZRW1 is a high-speed ZL1 scheme¹¹ recently released into the public domain by Ross Williams.¹²

Allb is a very high-speed ZL2 scheme¹ released in late 1990 into the public domain for research use only, by Brandon S. Allbery. This code can be retrieved under the name compact_sv, from volume 15, number 89, of the archives for the Internet news group, comp.sources.misc.

UW is the standard Unix-Welch Berkeley Compress utility, a ZL2 variant,¹ with 16-bit codes (64K dictionary entries).²

A1, B1, A2, B2, and C2 are Fiala and Greene's algorithms. A1 is a ZL1 scheme¹¹ with a 4K-character buffer, augmented with a literal transmission mode using a suffix trie to accelerate the buffer searches. Its compressed output contains two types of code words: literal *x*, meaning that the decompressor should pass the next *x* characters directly to the output, and copy *x y*, meaning that the decompressor should go back *y* characters in the output buffer and append the next *x* characters to the output buffer. B1 is similar to A1, but optimized for speed. It builds a Patricia trie with only some of the substrings contained in the last 4K characters of input. A2

and B2 are similar to A1 and B1 with 16-Kbyte buffers and a static Huffman back end. C2 is designed for maximum compression. Its Patricia trie contains all the substrings contained in the last 16K characters of input. It gains compression efficiency over the usual ZL1 scheme, since it does not waste code space on duplicate substrings. Because A2, B2, and C2 use a Huffman code to send trie indices, I have listed their compression ratios in Table 5.

MW1 and MW2 are the Miller-Wegman algorithms,³ as implemented by Dan Greene of Xerox PARC.¹ MW1 is a ZL2 variant, with a 4,096-entry dictionary, a least recently used replacement algorithm, and Welch-style output coding. MW2 is like MW1, augmented with string extension. It is thus an ID-LRU, in Storer's terminology.⁵

MWP, of Table 5, is a high-compression, high-speed implementation of MW2 by Roberto Pasqui of IBM Italy. It uses a static Huffman code with only three code lengths. The shortest codes refer to the most recently used set of 128 code words; medium-length codes refer to a set of less recently used 512 code words; and the longest codes refer to the oldest 4,096 code words.

Y64 and Y512 are the *yabba* coders, placed into the public domain recently by Dan Bernstein. Readers can retrieve these codes from volume 24, postings number 73 through 76, of the comp.sources.unix archives. *yabba* is a ZL2 variant¹ that adds one string *pc* to its dictionary for each input character *c*, where *p* is the longest suffix of the already processed input such that *p* is currently in the dictionary. The higher compressing Y512 has a 512K-entry dictionary; Y64 has a 64K-entry dictionary.

Like MWP, the Rogers-Thomborson code RT¹³ uses a back-end coder to transmit frequently used code words with fewer bits than the rarely used ones. However, since RT uses an adaptive arithmetic code rather than a static Huffman code, it is very slow. RT grows its dictionary in a similar fashion to the MW2 and MWP algorithms, except that it does not add new dictionary entries ending with a blank space when extending characters. In such cases, only string extension oc-

Table 4. Compression ratios of Ziv-Lempel algorithms that do not also employ Huffman or arithmetic coding. Boldface entries are row minima.

Test files	Algorithms									
	V42	LZRW1	Allb	UW	A1	B1	MW1	MW2	Y64	Y512
Csh	0.62	0.65	0.62	0.64	0.61	0.62	0.62	0.57	0.64	0.60
Essays	0.43	0.59	0.48	0.40	0.45	0.44	0.42	0.40	0.39	0.37
News	0.56	0.65	0.60	0.54	0.56	0.55	0.56	0.53	0.54	0.49
Pascal	0.31	0.39	0.35	0.30	0.25	0.25	0.31	0.17	0.25	0.22
Tex	0.42	0.52	0.46	0.39	0.41	0.39	0.41	0.35	0.40	0.35

curs. This heuristic tends to increase the fraction of dictionary entries that are complete English words or phrases, when the input is textual. Since there are usually few ASCII blanks in a binary file, their compression is not degraded much. Note that RT compresses slightly better than MWP, on all but the csh binary. Most of this improvement is due to the nonblank heuristic, not to the arithmetic coding; but this hypothesis deserves test.

LHarc (see Table 5) is a public-domain code from Japan in common use for distribution of personal computer software on floppy disks. To install such packages, users must typically wait minutes for the software to be decompressed onto their hard disk or alternate floppy drive. I obtained a Unix version of LHarc over the Internet, in volume 11, numbers 17-18, of the comp.sources.misc archives. On my Sun 3/50, LHarc runs at about 2 Kbytes/s, or about 15 times slower than the Unix Compress utility UW. As may be seen from Table 5, however, LHarc has better compression ratios. No documentation was included with my version of LHarc. It appears to run in two passes: a ZL1 coding¹¹ with a 4-Kbyte buffer followed by a dynamic Huffman recoding of the Ziv-Lempel indices. This Huffman back end probably accounts for most of the compression improvement and speed degradation, relative to UW.

Freeze is a public-domain code that recently emerged from the Soviet Union. It is archived in volume 17, numbers 68 and 74, of the comp.sources.misc news group. Like LHarc, Freeze is a ZL1 (first Ziv-Lempel) variant with a dynamic Huffman back end, although it has an 8-Kbyte buffer. Since single-character literals appear in the output stream, Freeze is more precisely an LZSS (Lempel-Ziv-Szymanski-Storer)¹⁴ variant.^{11,12} Its output coding appears to work a little better than that of LHarc, and it runs at about the same speed. Of a 512-symbol alphabet, the first 256 symbols denote single-character literals; one symbol marks an end-of-file; and the last 255 symbols indicate the length l of the match ($1 < l \leq 256$). If a multiple-character match is selected, the next bits in the compressed stream indicate the most-significant 6 bits of the position of the match. These bits are sent with a static Huffman code of 1 to 8 bits. The 7 least significant bits of the match position transmit in binary.

As may be seen in Table 5, the Bentley-Sleator-Tarjan-Wei code¹⁵ achieves very good compression ratios for every file except csh. Dan Greene's implementation provided the data in Table 5. This data agrees with my independently coded implementation to within 0.01.

BSTW's compression ratios are remarkable, it uses less than

Table 5. Compression ratios of algorithms employing Huffman or arithmetic coding. Boldface entries are row minima over Tables 4 and 5.

Test files	Algorithms								
	A2	B2	C2	MWP	RT	LHarc	Freeze	BSTW	ppmC
Csh	0.54	0.55	0.52	0.51	0.55	0.51	0.50	0.69	0.47
Essays	0.40	0.39	0.36	0.36	0.34	0.40	0.39	0.39	0.29
News	0.48	0.48	0.44	0.47	0.45	0.48	0.46	0.53	0.40
Pascal	0.15	0.14	0.13	0.17	0.16	0.19	0.15	0.24	0.18
Tex	0.34	0.33	0.31	0.32	0.31	0.36	0.33	0.38	0.27

6 Kbytes of space for its data structures. A BSTW compressor, in my implementation, uses two linked lists of 239 cells containing words of up to 16 characters each. One list contains only alphanumeric words; the other list contains nonalphanumeric words. The compressor parses the input file into an alternating sequence of maximal length alphanumeric and nonalphanumeric words. Zero-length words in the lists allow for the eventuality that the input contains alphanumeric (or nonalphanumeric) substrings of length greater than 16. A previously encountered word transmits with an adaptive Huffman encoding of the list position and then moves to the front of its list. A new word transmits with an escape code followed by an adaptive Huffman encoding of the cleartext word.

The Bell-Cleary-Moffat-Witten code ppmC^{16,17} of Table 5 is based on a variable-order Markov modeling of the source, with arithmetic coding used to communicate state transitions. It achieves impressive compression ratios, however it requires the largest amount of data space (730 Kbytes) of any algorithm in my tables.

Readers interested in more data on the relative performance of compression algorithms should access the comp.compression news group on Internet. One contributor, Peter Claus Gutman, is currently developing a large body of data on the Calgary corpus of test files.¹⁷

Discussion. As indicated in Tables 4 and 5, V.42bis obtains markedly worse compression ratios than many other algorithms. Let's look at competing algorithms to see if they offer a significant advantage in the V.42bis application.

The LZRW1 and Allb algorithms offer significantly worse compression ratios than V.42bis. However, they run two or three times more rapidly on my Sun 3/50 than any other algorithm I have tested. I observed speeds of 50-70 Kbytes/s and 90-120 Kbytes/s on this 1-MIPS machine, using Sun's optimizing compiler on these C codes. Even higher bandwidths could be obtained with CPU-specific code tuning. The LZRW1 and Allb algorithms would thus be preferable to V.42bis in applications that are cost-limited but not I/O chan-

**Readers interested in more
data on the relative performance
of compression algorithms
should access the
comp.compression news
group on Internet.**

nel-limited, for example, in a compressing interface to a low-cost tape or disk drive. For the low-cost modem application, however, the better compressing V.42bis algorithm is clearly preferable, since a 10-MHz Z80 is sufficient to keep up with a telephone modem's low-speed serial I/O channels.

The UW algorithm achieves slightly better compression than V.42bis on the test files, at the expense of 16 times the data space. Since a low-cost modem is limited in memory, UW is not an appropriate choice. Furthermore, as shown in Table 3, a memory-limited UW is not competitive in compression ratio with V.42bis.

Fiala and Greene's algorithm A1 requires 145 Kbytes to compress with a 16K-entry dictionary; their other algorithms require even more space. Further research is necessary to decide if any of these algorithms are competitive with V.42bis, when restricted to a 10- or 20-Kbyte dictionary and an 8-bit microprocessor. Still, the A2, B2, and C2 algorithms would be a promising starting point for research on a future data compression standard.

Despite its LRU-Huffman back end, MWP runs at a respectable 10-20 Kbytes/s on my test files, on an IBM PS/2 mod 80, 16-MHz 80386 under DOS 4.0, according to private correspondence from Victor Miller. By way of comparison, the Unix Compress utility UW runs at 25-50 Kbytes/s on the test files on my Sun 3/50, and my V.42bis implementation runs at only 6-10 Kbytes/s. Even though my V.42bis implementation is not optimized for speed, I believe MWP would be at least as fast as V.42bis, even on an 8-bit microprocessor such as a Z80. Despite its impressive speed and compression ratios, however, MWP is not immediately applicable to the design of today's low-cost V.42bis modems, since it requires 60 Kbytes of space for its data structures.

BSTW requires just 6 Kbytes of data space, so it would fit in a low-cost modem. However, its adaptive Huffman encoding would severely limit the bandwidth of any such modem's low-performance processor. A much more expensive modem could employ a special-purpose, high-bandwidth, Huffman- (or arithmetic-) encoder/decoder chip, as well as a

high-speed hardware implementation of BSTW.¹⁸ The poor performance of BSTW on nontextual data could pose a problem in many applications, although one might envision a revision to V.42bis allowing a compressor to enter BSTW mode whenever suitable data was received.


As noted earlier, a V.32bis/V.42bis modem is usually limited by factors other than its compression ratio. In high-performance, relatively cost-insensitive applications, bottlenecks such as the EIA-232-D can be removed, and other data compression schemes would be markedly superior to V.42bis. Thus MWP (perhaps with the RT conditional dictionary entry heuristic), the Fiala-Greene algorithm B2, BSTW, and ppmC deserve attention in the next round of standards-making activity.

THIS EXAMINATION OF SOME OF THE PRACTICAL, algorithmic, and marketing aspects of the V.42bis standard has not uncovered any serious flaws in its specification. On the contrary, the algorithm embodied in this standard is competitive with other Ziv-Lempel variants, if these are required to run in a small data space on an 8-bit microcomputer.

My data shows that a well-implemented V.42bis modem can achieve 2.3:1 compression on English text, 1.6:1 compression on 68000 object code, and 3.2:1 compression on Pascal source code. The best compression ratios I observed, for any data compression algorithm, were 3.4:1 on English text, 2:1 on object code, and 7:1 on my highly compressible sample of Pascal. The algorithms achieving these ratios require significantly more computational resources than does V.42bis, so they would not be viable in the current market for modems. Still, it is clear that a revision to the V.42bis standard could offer 50 percent or more additional bandwidth in many applications.

The future of data-compressing modems is uncertain. Microcomputers and their memory are rapidly becoming cheaper, by 20 percent to 30 percent per year. Thus the quick-and-inexpensive algorithmic choices embodied in V.42bis will soon be outmoded, and the more computationally expensive but better-compressing algorithms I've mentioned will become economical for text transmission. However, text transmission is only a fraction of current computer communication over the telephone network, and this fraction is rapidly decreasing. Source-specific compression algorithms (for voice, image, fax) can be run efficiently on a personal computer, achieving much higher compression ratios than are offered by any of the algorithms discussed here, for these nontextual sources.

Following this line of reasoning, the V.42bis standard will continue to be used in low-cost computer-to-ASCII-terminal communications, but it will find less and less applicability in computer-to-computer communications. I would not expect to see many modems making use of the V.42bis feature in the year 2000, when a 10-MIPS personal computer should be

only a little more expensive than an ASCII terminal. Source-specific data compression algorithms will run on such computers at high bandwidth, not only for communication but also for mass storage devices. 

Acknowledgments

Former UMD students Clyde Rogers and Julie Redland, or Dan Burrows and Bill Marko of UMD Information Services, ran many of the experiments I've described. I am indebted to John Copeland, of Hayes Microcomputer Products, Inc., for a copy of the source code to a V.42bis-compliant compressor and for the test files used by the V.42bis study group. I thank Dan Greene of Xerox PARC, Tim Bell of the University of Calgary, and Victor Miller of IBM for taking the time and trouble to run our test files through their compressors. Telephone conversations with Ashok Patel and Warren Henderson corrected my misunderstandings about their throughput measurement data. Last but not least, I thank the (four!) referees of my manuscript for their detailed and thoughtful reviews.

The National Science Foundation, through its Design, Tools and Test Program, supported this research under grant number MIP 9023238.

References

1. J. Ziv and A. Lempel, "Compression of Individual Sequences via Variable-Rate Coding," *IEEE Trans. Information Theory*, Vol. 24, No. 5, 1978, pp. 530-536.
2. T.A. Welch, "A Technique for High-Performance Data Compression," *Computer*, Vol. 17, No. 6, June 1984, pp. 8-19.
3. V.S. Miller and M.N. Wegman, "Variations on a Theme by Ziv and Lempel," in A. Apostolico and Z. Galil, eds., *NATO ASI Series*, Vol. F12, Combinatorial Algorithms on Words, Springer-Verlag, Berlin, 1985, pp. 131-140.
4. J.A. Storer, "Textual Substitution Techniques for Data Compression," *ibid.*, pp. 111-129.
5. J.A. Storer, *Data Compression: Methods and Theory*, IEEE Computer Society Press, Los Alamitos, Calif., 1988.
6. E.W. Fiala and D.H. Greene, "Data Compression with Finite Windows," *Commun. ACM*, Vol. 32, No. 4, Apr. 1989, pp. 490-505.
7. M. Byrd, "9600-bps Modems: Breaking the Speed Barrier," *PC Magazine*, Dec. 11, 1990, pp. 307-346.
8. U. Black, *Physical Level Interfaces and Protocols*, CS Press, 1988.
9. B. Glass, "High Speed Modems," *PC World*, Dec. 1991, pp. 236-242.
10. W.L. Henderson, Jr., and S.S. King, "Testing V.32 Modems—the Right Way ... (cont'd)," *Data Communications*, May 1991, pp. 99-105.
11. J. Ziv and A. Lempel, "A Universal Algorithm for Sequential Data Compression," *IEEE Trans. Information Theory*, Vol. 23, No. 3, 1977, pp. 337-343.
12. R.N. Williams, "An Extremely Fast Ziv-Lempel Data Compression Algorithm," *Proc. Data Compression Conf.*, IEEE, New York, Apr. 1991, pp. 362-369.
13. C. Rogers and C.D. Thomborson, "Enhancements to Ziv-Lempel Data Compression," *Proc. 13th Int'l Computer Software and Applications Conf.*, Sept. 1989, pp. 324-330.
14. J.A. Storer and T.G. Szymanski, "Data Compression via Textual Substitution," *J. ACM*, Vol. 29, No. 4, 1982, pp. 928-951.
15. J.L. Bentley et al., "A Locally Adaptive Data Compression Scheme," *Commun. ACM*, Vol. 29, No. 4, Apr. 1986, pp. 320-330.
16. J.G. Cleary and I.H. Witten, "Data Compression Using Adaptive Coding and Partial String Matching," *IEEE Trans. Computers*, Vol. COM-32, No. 4, Apr. 1984, pp. 396-402.
17. T.C. Bell, J.G. Cleary, and I.H. Witten, *Text Compression*, Prentice Hall, Englewood Cliffs, N.J., 1990.
18. C.D. Thomborson and B. W-Y Wei, "Systolic Implementations of a Move-to-Front Text Compressor," *Proc. 1989 ACM Symp. Parallel Algorithms and Architectures*, ACM, New York, June 1989, pp. 283-291.



Clark Thomborson teaches computer science and computer engineering at the Duluth campus of the University of Minnesota but is currently on a one-year sabbatical at MIT, where he will teach and conduct research. He has also served on the faculty of the University of California, Berkeley, Computer Science Division.

Thomborson received his bachelor's degree in chemistry, master's in computer science/computer engineering from Stanford, and doctorate in computer science from Carnegie Mellon University. He has published more than 40 articles on special-purpose hardware implementations of algorithms, VLSI theory, graph theory, algorithmic analysis and the effects of military funding on academic science and engineering. He is a member of the IEEE and ACM.

Direct any questions regarding this article to the author at the Massachusetts Institute of Technology, Laboratory for Computer Science, 545 Technology Square, Cambridge, MA 02139; or e-mail at cthombor@theory.lcs.mit.edu.

Reader Interest Survey

Indicate your interest in this article by circling the appropriate number on the Reader Service Card.

Low 165 Medium 166 High 167