LABORATORY FOR
COMPUTER SCIENCE

MASSACHUSETTS
INSTITUTE OF
TECHNOLOGY

MIT/LCS/TM-299

THE APPLICATION OF DIGITAL BROADCAST COMMUNICATION TO LARGE SCALE INFORMATION SYSTEMS

DAVID K. GIFFORD
JOHN M. LUCASSEN
STEPHEN T. BERLIN

APRIL 1986

545 TECHNOLOGY SQUARE, CAMBRIDGE, MASSACHUSETTS 02139

The Application of Digital Broadcast Communication to Large Scale Information Systems

DAVID K. GIFFORD, MEMBER, IEEE, JOHN M. LUCASSEN, MEMBER, IEEE, AND STEPHEN T. BERLIN

Abstract — **A new type of information system is described that combines personal computers, broadcast data communication, and bidirectional communication. The system is designed to use broadcast communication whenever possible to deliver information to personal computers, which are used for data storage, indexing, and retrieval.**

This paper starts with an overview of the system, and then discuss the problem of reliable digital broadcast communication in some detail. A parameterized broadcast protocol is described, and we show how to choose protocol parameters based on observed channel error characteristics. A flexible encryption-based protection system is included in the protocol. We discuss the implementation of the system on contemporary personal computers. A broadcast system based on these ideas is now operating in Boston area homes.

I. Introduction

THE goal of the research reported here is to use computers to improve communication between people. Our view is that the computer is an excellent communication medium because of its ability to process, index, edit, and display information, and that this capability can be well applied on a large scale to communication within a community.

However, building a computer system large enough to serve a community is a difficult problem. This paper describes a design for a decentralized system to meet this need. Our major design goals were the following. We wanted the system to

- economically serve a major metropolitan user community
- provide a high-quality user interface
- give its users access to a wide variety and a large volume of information
- allow its users to add value to the information provided by the system by specifying filtering and further processing
- safeguard the privacy of users
- be easily extensible to new services.

Two contemporary designs for community information systems, Teletex [5], and Viewdata [2], are now being tried out in various communities. Let us consider how these systems address the goals that we have established.

Teletex is the broadcast transmission of information for display in fixed-format pages on user's TV screens. Teletex succeeds in our goals of scale, economy, and privacy (there is no user-specific information to keep private), but it fails on several counts: the user interface is limited; the user has access to only a few hundred screenfuls of information; the information cannot be processed according to the user's specifications; and expansion to new services, such as interactive services, is not readily possible.

The second contemporary technology, called either Videotex or Viewdata, is based on central time-sharing systems that use TV-like terminals with a protocol that includes graphics [4]. The design gives users access to a wide variety of information and the services provided can be easily expanded. However, the Viewdata approach fails to some extent on all of our other goals. Viewdata relies heavily on large central systems, and thus it may not be an economical way to provide sophisticated services to an entire metropolitan area. Its user interface suffers from communication bandwidth problems and the response time limitations of time-sharing. Users are not able to add value to information by specifying additional processing and can not easily customize the system to their own interests. Finally, the system has troublesome privacy implications because all the user's requests can be monitored and recorded by the system.

Based on the analysis given above we have concluded that neither Teletex nor Viewdata, in their present forms, meet our goals. However, both have attractive properties. The broadcast nature of Teletex allows the system to economically accommodate an arbitrary number of users. With Viewdata, on the other hand, each user has direct access to the database, which ensures immediate access to a wide variety of information.

Our design seeks to combine the economy of Teletex with the broad-spectrum access provided by Viewdata. To meet this goal, our design combines personal computation and communication. A personal computer is located at every user site. Information is transmitted to these personal computers via broadcast communication. The personal computers retain information of interest to their owners

and provide a personalized information service. Because each user station has local processing and storage capability, the user can gain effective access to much more data than in the Teletex system without resort to the central per-user processing required for Viewdata.

The approach of sending information to the user's location and processing it there has a number of advantages. First, the central site can support any number of broadcast service users. Second, locating processing power with the user allows for a high-quality user interface. Third, local processing and storage can be used to assist the user in managing a larger volume of available information. Fourth, the user can choose how to add value to information, integrating received information with local computational tools and databases. Fifth, the local processing of information keeps private information confined to the user's site. Finally, because the personal computers are fully programmable, the system is easily extensible to new services.

We have built a prototype based on these ideas, namely the Boston Community Information System. The central site for the system is located at our laboratory and users are scattered throughout the Boston area. Presently, our digital broadcast channel is on the subcarrier of a normal FM broadcast station. The information that we transmit to our test audience includes two wire services (the New York Times and the Associated Press). We are presently integrating remote database access into our system, using dialup telephone lines for two-way communication. In the coming year we plan to expand our service to include computer software distribution, general interest bulletin boards, and local community and event information.

The remainder of the paper is organized in five sections. Section II describes the overall architecture and functionality of the system. Section III describes a parameterized protocol for reliable broadcast communication. Section IV describes our encryption-based protection scheme for use on a broadcast channel. Section V describes our implementation and operational experience. Section VI contains a brief summary and concludes with a look toward the future.

II. System Overview

We will provide an overview of the system in two stages. First, we will discuss the database component of the system, including the data model and the query facility that allows users to retrieve information of interest. The second part of our overview describes how this service is implemented on a decentralized hardware base.

When considering how to provide access to community information one soon realizes that it is a problem that cannot be solved by the application of standard commercial database techniques. Relational and hierarchical data models are far too restrictive to allow users to locate information of interest because of the relatively unstructured information (such as newspaper articles) that the system must handle.

We have chosen an approach that builds on ideas from full-text retrieval systems to fill our needs. Our database design is capable of handling a wide variety of information, including text-oriented data (news stories and electronic mail), information that has somewhat more structure (community event descriptions and city guides), and other kinds of data, including computer programs. In order to put all of this information into a single database we have adopted a fairly simple data model.

Every entry in our database, whether it be a *New York Times* article or a restaurant review, consists of a number of fields. The specific fields found in an entry depend on the type of the entry. For example, a newspaper article has a source identifier (e.g., "New York Times"), along with *category*, *subject*, *priority*, *section*, *title*, *author*, *date*, and *text* fields. The *text* field contains the body of the article. Likewise, an event listing includes *location*, *time*, *title*, and *abstract* fields.

A user can find out what information is available in the database by submitting queries that are Boolean combinations of words and phrases that may occur in various fields. Some fields can only contain certain words: for example, the *priority* field of a news article is chosen from the words *flash*, *bulletin*, *urgent*, *regular*, and *deferred*. Other fields, such as the *author* or *text* field, contain arbitrary text, and a user query can include arbitrary words and phrases. This is in contrast to controlled vocabulary systems where information is only indexed on a predefined set of index terms and the user is limited to these terms when formulating a query.

When the user submits a query to the system, a list of matching database entries is displayed along with a summary of each entry. Fig. 1 is a picture of the user interface that shows the result of entering the query "technology & (category financial)." Once the menu of available entries is displayed, the user may enter the number of the desired entry on the end of the query and press the return key. Fig. 2 shows the display of the second entry from the menu of Fig. 1.

We decided against basing our system solely on menus because we felt that free text searching provides more expressive power and is easy to understand and use. Other work [3] has suggested that novices may in fact prefer keyword-based searching to menu systems. Menu-based systems do have the advantage that the user can easily browse the database to see what is available without having something specific in mind. We have tried to retain this advantage of menu systems.

Our personal database system uses both free text and menu-based retrieval in an effective way. A user specifies what information should be kept in the local database by composing a set of free text queries. Because the system knows nothing *a priori* about the user's interests (and the user knows little about the system's capabilities and the scope of available information), unrestricted text is the more efficient medium for expressing such information filters.

However, when it comes to examining the local database that was compiled with the aid of these filters, the system "knows" what the user's interest profile is. This makes menu-based retrieval the more efficient medium. The information filters defined by the user serve as a menu of what is available in the local database; furthermore, the user can

```
5 matching articles found.                                  lines 1:18 of 18

1 sep 19, 10:48 (121 lines) regular (Financial)
    NEW YORK -- After a record year, the market for public stock
    offerings by private companies has gone into a slump, forcing many of
    these companies to bypass the new-issue market and seek capital --
    often through creative deals -- elsewhere.
2 sep 18, 22:37 (80 lines) regular (Financial)
    NEW YORK -- Technology stocks took a beating Tuesday, for two
    unrelated reasons, and helped to keep the market on the downside.
3 sep 18, 21:18 (82 lines) urgent (Financial)
    A digest of business and financial news for Wednesday, Sept. 19.
    1984:
4 sep 18, 18:22 (70 lines) urgent (Financial)
    NEW YORK -- Stock prices dropped Tuesday in accelerated trading, with
    some of the technology and large capitalization issues registering
    the biggest declines.
5 sep 18, 7:41 (113 lines) deferred (Financial)
    London - The American lawyer would have been rubbing his hands,
    except that he was jogging in Hyde Park, so he was swinging his arms.

technology & (category financial);
```

Fig. 1. User interface: menu screen (© 1984, *New York Times*).

```
Article N409187.727:                                        lines 1:23 of 80

type: New York Times general news copy
priority: regular
date: 09-18-84 2237edt
category: Financial
subject: MARKETPLACE
title: (BizDay)
author: DANIEL F. CUFF
source: (c)1984 N.Y. Times News Service
text:
     NEW YORK - Technology stocks took a beating Tuesday, for two
unrelated reasons, and helped to keep the market on the downside.
     First, worry over problems with a disk drive hurt Contral Data and
Burroughs. Second, the semiconductor issues were battered by a
bearish brokerage house report on Motorola.
     Burroughs opened down 2 3/8 Tuesday morning after an order imbalance.
The drop in Burroughs, which closed the day at 53, off 3 5/8, followed
Control Data's slide. On Monday, Control Data dropped 2 1/8, and it lost
an additional 3/8 Tuesday, to close at 26 1/8.
     Control Data, according to analysts, encountered problems with a
thin coating on the disk. ''If that chemical compound is not
virtually perfect, trouble ensues,'' said Ulric Weil, an analyst at
Morgan Stanley & Co. ''We are talking about tolerances the thickness
of a human hair.''

technology & (category financial);2
```

Fig. 2. User interface: article display (© 1984, *New York Times*).

use any of the filters, or any query, for example "(category news)," to obtain a list of matching articles, and then browse through that list. We find that most of our users use their information filters to browse the database.

The greatest asset of our data model is that it is simple to understand. Thus, our users have a good conceptual model of precisely what the system can and will do when they compose a query. This in turn allows them to use the system effectively. However, our users still need to know certain things about the database in addition to the access mechanisms. Consider the problem of locating movie reviews. Is the query "movie & review" appropriate, or might "(subject review) & movie" be better? To help users learn about the database and about composing queries, we provide printed and on-line documentation, and we also provide a library of preplanned queries that users can incorporate into their personal database system.

This concludes our discussion of the facilities that we provide for organizing and locating information in the system. We will now turn our attention to the implementation of the system.

As shown schematically in Fig. 3, information arrives at our central database machine from a variety of sources.

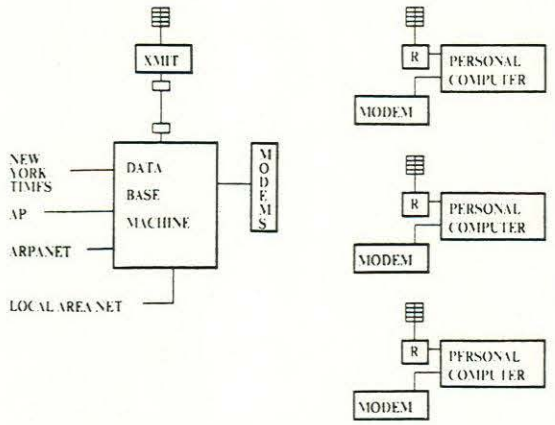

Fig. 3. Block diagram of the system.

Dedicated telephone circuits deliver information to our machine from the New York Times and the Associated Press. Our link to the Arpa Internet delivers certain Arpanet digests. We also have dial-up telephone lines that can be used to transfer input files from personal computers to the central database.

For each information source, there is a separate background process that parses information from that source and creates entries in a standard format, filling in field values as appropriate. This way, information from all sources is converted into a single stream of data. This stream serves as input to three other background processes: the clipping service, the central database system, and the scheduler.

The clipping service is a program that automatically forwards newspaper articles and other database entries to users via electronic mail based on each user's interest profile.

The central database system maintains a complete database of all information and a full-text index into this database. The clipping service and the central database system are not required for the operation of the broadcast system, but as we mentioned earlier, we are integrating access to the central database into our personal computer database system.

The scheduler transmits entries from the central database to personal computers over the broadcast channel. Currently the entire database is transmitted approximately every 4 h, but as we add new information sources this cycle time will increase. Information from the database is transmitted in round robin fashion. Newly arrived database entries are placed at the head of the transmit queue. High-priority database entries are transmitted more frequently. The latency of an entry from the time it arrives at the central system to the time it is transmitted is less than 5 min.

All of the central site software has been designed to recover in the event of a system crash. Database entries and transmit queues are maintained on secondary storage and are recovered upon system restart.

We will now turn our attention to the processing that takes place at the remote personal computers. The personal database system is a software package that runs on a user's personal computer and implements the user interface to the

community information system. When a user submits a request, the personal database system will display the list of local database entries that match the query. In our current prototype the personal database software uses only local information to satisfy user requests. When two-way communication is added, the system will attempt to determine whether a query can be completely answered with information from the local database, and if not, it will offer to search the central database as well.

The personal database system performs two tasks concurrently: receiving data and managing the user interface. These two processes share the personal computer by using a simple nonpreemptive multiprogramming system.

The reception process in the personal database system listens to the digital information broadcasts from the central system and keeps the local database up to date. As described in the next section, every database entry is transmitted as a series of packets. The personal computer software reassembles these packets into a complete database entry, decrypts the entry, filters the entry to see if it should be kept locally, and if so, enters it into the local database.

The user interface process is idle, except when the user submits a request through the keyboard. Requests are parsed and processed by the database component, and relevant information is displayed on the screen.

III. A Protocol for Reliable Broadcast Communication

In this section we present an algorithm for managing a digital broadcast communication channel. A digital broadcast channel delivers an unreliable byte stream in a very economical way to a large number of users. The problem is to make communication as reliable as desired without using acknowledgments.

The algorithms that we outline are independent of the precise type of digital broadcast medium that is employed. Current digital broadcast technology encompasses many alternatives. One example is the vertical blanking interval of TV transmissions. The resulting TV plus data transmissions can be delivered via a cable system or normal RF broadcast techniques. It is also possible to use an entire TV channel or other portion of the RF spectrum for higher bandwidth broadcast digital communication. One problem with high bandwidth communication is that contemporary personal computers can not process data at megabit rates. These channels could be employed by implementing packet selection in hardware, or by buffering information as it arrives and processing it later.

A. A Broadcast Transmission Protocol

We have designed and implemented a three-layer protocol for use on unidirectional byte channels characterized by burst errors. The protocol has a low implementation complexity, and is efficient enough to permit continuous error detection and correction at 4.8 kbits/s on a personal computer without any special-purpose hardware, using only a fraction of the available CPU power. The three layers of the protocol are depicted in Table I.

1) The Byte String Layer: The first and lowest layer of our protocol is the byte string layer. A byte string is defined as a finite sequence of arbitrary bytes. There is no guarantee that a byte string is delivered to the receiving sites or that it is delivered without errors, but all byte strings that are delivered are guaranteed to arrive in the order in which they were transmitted.

The byte string layer is implemented directly on top of the byte channel, and the end of each string and the beginning of the next is indicated by one or more bytes serving as separator tokens. Since the contents of the byte string can be arbitrary, any instances of the separator token in the byte string itself are mapped into other values by means of byte-stuffing.

2) The Packet Layer: The packet layer of the protocol is implemented on top of the byte string layer. It provides for transmission of packets of arbitrary contents, up to a certain length. (In our implementation, the length in bytes must be a multiple of four not exceeding 4*255.) The packet layer serves as an error detection layer: there is no guarantee that individual packets are delivered, but all packets that are delivered are guaranteed to be complete, free of errors, and in order. To accomplish this, the packet layer transmits each packet as a byte string, prefixing it with a length field and a checksum. The header format is given in Fig. 4.

If the length field of a received packet does not match the actual length of the received byte string, the packet is rejected. Otherwise, the checksum is computed. If the checksum does not match, the packet is also rejected. Otherwise, the packet is accepted.

Note that the guarantee of error-free transmission is merely probabilistic: if a packet has been corrupted, it will nevertheless be accepted if the length field and the checksum both match. With a 32 bit checksum, an error can go undetected with a likelihood of no less than 2^{-32}.

3) The Data Layer: The data layer of the protocol is implemented on top of the packet layer. It provides for the transmission of data blocks of arbitrary contents up to an implementation dependent length. The data layer serves as an error correction layer: although it provides the same guarantees as the packet layer (delivered data blocks are complete, free of errors, and in order), it has the potential for greater useful throughput when channel errors are likely to occur.

For transmission, each data block is divided into a number of fixed-sized packets. (The last packet may be shorter than the others if the block length is not an even multiple of the packet length.) The fragmentation is performed subject to the length constraint on packets and the implementation-defined limit (currently 100) on the maximum number of packets.

Recall that the underlying packet layer does not guarantee that individual packets are delivered. To increase the likelihood that a packet is delivered intact, it may be transmitted more than once. Because the physical channel

TABLE I
PROTOCOL LAYERS

Name of Layer	Unit of Transmission	Purpose of Layer
Data Layer	Data Blocks	Error Correction
Packet Layer	Packets	Error Detection
Byte String Layer	Byte Strings	Framing
(Byte Channel)	(Bytes)	(Transport)

The three layers of the broadcast protocol built on top of the byte channel.

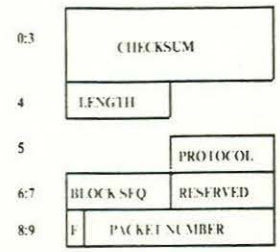

Fig. 4. The format of packet headers, including header information attached by both the packet layer (bytes 0–4) and the data layer (bytes 5–9).

exhibits burst errors (and no apparent periodicity), we create time diversity by interleaving these redundant transmissions: all the packets of the data block are transmitted once, and then this sequence is repeated as many times as is necessary to achieve the desired expected level of reliability.

The data layer appends a header containing reassembly information to each packet. The header format is as follows (see also Fig. 4).

- 4 bytes Checksum (part of the packet layer)
- 1 byte Length of packet in 4 byte words (part of the packet layer)
- 1 byte Reserved to specify the protocol that is used
- 1 byte Sequence number of the data block (mod 256)
- 1 byte Reserved for future use
- 2 bytes Packet number (within the data block). The high-order bit indicates whether this is the highest numbered packet of the data block.

Using this information, data blocks can be straightforwardly assembled as follows. Since packets are guaranteed not to be delivered out of order, and cannot contain erroneous data, the arrival of a packet with a data block serial number different from that of the previous packet signals the start of a new data block. The bit vector that records which packets of the data block have been received is initialized to all zeros and the expected number of packets (for the new data block) is initialized to "unknown."

For each packet that arrives, this bit vector is consulted. If the packet contents are already present, the packet is ignored and the system waits for the next packet. Otherwise, the packet contents are copied into the data buffer, starting at a location determined by the packet length and the packet number, and the packet's presence is noted in the bit vector. If the packet is tagged as the highest numbered packet of the data block, the expected number of packets is filled in. Each time a packet is copied into the data buffer, the bit vector and the expected number of packets are examined to determine whether a complete data block has been received.

Whenever a complete data block is received it is handed over for further processing. In our system this processing is performed in place, so control is not returned to the data layer until all processing has been completed. This may include decrypting the data block, scanning it to see if it matches the user's filter, and possibly saving the contents on disk. When the data layer regains control, it directs its attention to the incoming packet stream, and is ready to assemble the next data block.

Note that the correctness guarantee on data blocks is once again a probabilistic one: if any packet has been undetectably corrupted, the data block of which it is part may reflect the error, either directly or indirectly through incorrect assembly. Moreover, the one-byte data block serial number does not allow detection of transmission outages that last 255 data blocks (mod 256). Such outages must be detected by a timeout mechanism.

B. Engineering a Specific Channel

The digital broadcast system that we operate in Boston uses an FM Subsidiary Communications Authority (SCA) channel. SCA channels are subcarriers that can by used by an FM radio station without interfering with normal broadcasting. Typical uses of SCA channels include Muzak, stock quote services, and reading for the blind.

In our application the SCA of WMBR-FM is modulated with a frequency shift keyed signal (FSK) that carries RS-232 compatible asynchronous data at 4.8 kbits/s. An FSK system that uses asynchronous signalling is particularly simple to demodulate and to interface to existing serial ports on personal computers. The cost of the receiving equipment (not including the personal computer) is under $200 per receiver. A similar SCA data communication system is described by Anderson *et al.* [1].

The SCA channel transports a byte stream to our users. The time-average byte error rate of the channel varies, depending on the receiver configuration (the type of antenna and receiver circuit employed) and receiving conditions (distance between transmitter and receiver, multipath interference, interference from appliances, weather). The time-average byte error rate is given in Section III-C-2) for several receiver sites.

Currently, the transmission rate of our broadcast subsystem is limited to 4.8 kbits/s. This is not a limiting factor

since a typical personal computer can just accommodate continuous 4.8 kbit/s transmissions.

C. Choosing Values for Protocol Parameters

In this section, we describe the procedure used to determine appropriate values for the packet size and the number of repetitions to be employed by the data layer of the protocol. The parameters of the underlying byte channel, such as the transmission rate and the byte format, are presumed to be fixed and are not considered here.

1) Objective Functions: The first step of the process is to determine what objective function (of the parameters in question) we wish to maximize. We have considered the following functions in particular.

- The packet-level throughput rate: this is the ratio between the number of data bytes in a packet and the total packet size, times the probability that a packet arrives intact.
- The block-level throughput rate: this is the ratio between the number of data bytes in a data block and the total number of bytes it takes to transmit the data block, times the probability that the data block arrives intact. We will refer to this quantity as the channel utilization.
- The block delivery probability: this is the likelihood that a transmitted data block arrives intact.

The packet-level throughput rate would make a poor objective function because it reflects neither the number of repetitions nor the effects of the block size.

The block-level throughput rate, or channel utilization, makes a much better objective function. In fact, if a fixed set of data blocks is transmitted repeatedly in round robin fashion, and block delivery errors are statistically independent, we can minimize the mean latency from the time a data block is first transmitted until the time it is first correctly delivered by maximizing the channel utilization. Channel utilization will be our primary objective function.

The block delivery probability always approaches unity as the number of repetitions is increased. Therefore, it is not a useful objective function for choosing parameter values. It is, nevertheless, a useful quantity and we monitor it to avoid wasteful use of the channel.

2) Estimating Packet Error Rates: To calculate the channel utilization and the block delivery probability for a particular channel, we begin by estimating the packet error rate of the byte channel for various packet sizes. We define a k-burst (of errors) as a maximal-length sequence of bytes terminated by bytes transmitted incorrectly that does not contain a subsequence of k error-free bytes.

Fig. 5 shows the distribution of error burst lengths observed on our noisy channel for $k = 5$, along with the distribution that would be expected if byte errors were statistically independent, with the same byte error rate. The plateau in the expected distribution is an artifact corresponding to the definition of burst errors, but is nevertheless expected in the actual data. Instead, the observed distribution has a secondary peak at a burst length of 4 or 5, and has a much higher tail than the expected distribution. Therefore, we conclude that byte errors on the channel are not independently distributed, and that we cannot calculate packet error rates directly from the byte error rate of the channel.

Instead, we estimate packet error rates by means of a data collection program that maps the observed sequence of byte errors into (simulated) continuous packet streams of various packet sizes. We have run this program at several receiver sites, analyzing one Mbyte of received data at each site. The number of byte errors varied from 4870 (in a windowless, partially shielded, electrically noisy room near the transmitter) to 55 (in the suburbs 7.4 miles west of the transmitter), to 12 (7.5 miles north). A fourth test site, located 9.8 miles southeast of the transmitter, proved to have such poor reception that we could not deliver data blocks to that site reliably.

The relationship between packet size and packet error rate is determined by the extent to which byte errors are clustered. This dependence is best illustrated by our noisest set of observations. Table II shows the observed packet error rate as a function as packet size. For comparison, we have included estimated packet error rates based on the (erroneous) assumption that byte errors are independent with a byte error rate of 4870 errors per 1 Mbytes. It is clear from the table that byte errors are not independent, and that assuming independence would lead one to consistently overestimate the packet error rate. This confirms our earlier conclusion based on the distribution of error burst lengths.

The observed relationship between packet size and packet error rate indicates that even packet-level errors are not independently distributed over the time span investigated, up to 16K bytes. (This can be illustrated by comparing the delivery probability of two 1K packets, $(1-0.415)^2 = 0.342$, with the delivery probability of one 2K packet $(1-0.570) = 0.430$. Since a 2K packet can be viewed as two adjacent 1K packets, we find that the delivery probability of two adjacent 1K packets is greater than the delivery probability of two independently chosen 1K packets. Thus, packet errors are not independent.)

Because packet errors on the channel are not independently distributed, we cannot calculate the block delivery probability directly from the packet error rate of the channel. We could proceed as before and estimate the block delivery probability by mapping the observed sequence of packet errors into a (simulated) continuous stream of data blocks. However, to obtain accurate estimates in this way, we would need data collection runs lasting several orders of magnitude longer than those used to estimate the packet error rates.

Therefore, we are forced to estimate the block delivery probability directly from the packet error rates anyway, as a function of the block size, the packet size, and the number of repetitions. We have made these calculations using the channel model described below.

3) The Channel Model: In this section we describe the channel model used to choose suitable parameter values for the data level protocol. The model is based on the assump-

Burst Length:	No. of Bursts: Expected	Observed	Observed Bursts, * = 10
1	4400	736	**//*
2	20	54	*****
3	20	123	************
4	20	138	**************
5	20	137	**************
6	20	110	***********
7		90	*********
8		41	****
9		21	**
10-14		29	***
15-19		7	*
20-26		4	
65		1	

Fig. 5. Histogram of error burst lengths expected and observed on one megabyte of data received over a noisy channel.

TABLE II
ESTIMATED PACKET ERROR RATES BASED ON 4870 INDEPENDENT byte ERRORS PER MEGABYTE OF DATA, AND PACKET ERROR RATES ACTUALLY OBSERVED ON 1Mbyte OF DATA WITH 4870 bytes ERRORS

Packet Size (in bytes)	Estimated Packet Error Rate	Observed Packet Error Rate
4	0.018	0.009
8	0.037	0.014
16	0.072	0.025
32	0.138	0.044
64	0.258	0.075
128	0.449	0.125
256	0.696	0.193
512	0.908	0.286
1024	0.991	0.415
2048	1.000	0.570
4096	1.000	0.711
8192	1.000	0.835
16384	1.000	0.953

tion that packet errors are independent. To the extent that this is not true, our reliance on average values at the packet level tends to produce an overestimate of the error rates at the data block level, especially for small packet sizes and high repetition rates.

Let us call the block size N, the packet size n, the number of packets per data block k, and the number of repetitions r. Assume that the probability of packet error, $p(n)$, is given for all packet sizes of interest. Assume further that each packet contains a header of H bytes, leaving room for $(n-H)$ bytes of data. We will use $H=10$ throughout.

The number of packets per data block (not counting repetitions) is

$$k = \lceil N/(n-H) \rceil.$$

When the block size is not an exact multiple of the packet size, the last packet may be shorter than the rest. Because the packet error rate as a function of packet size is well-behaved, we can account for this by using a noninteger approximation of the number of packets per data block, namely the total number of bytes transmitted per repetition of the data block divided by the packet size:

$$k' = \frac{N + H\lceil N/(n-H) \rceil}{n}.$$

The potential channel utilization (which is achieved when there are no errors) is the ratio between the block size and the total number of bytes transmitted to get each data block across

$$\frac{N}{r(N + H\lceil N/(n-H) \rceil)}.$$

If each packet of a block is transmitted r times, the probability that all r copies of a given packet are damaged or lost in transmission is $p(n)^r$ (assuming independence of individual packet errors). Thus, the probability that at least one copy of a packet arrives intact (which is all that is needed) is $1-p(n)^r$. If a block is fragmented into k' packets (not counting repetitions), then the block delivery probability is equal to the probability that all k' packets can be reconstructed, or $(1-p(n)^r)^{k'}$.

The channel utilization is the product of the potential channel utilization and the block delivery probability:

$$\text{channel utilization} = \frac{N(1-p(n)^r)^{k'}}{r(N + H\lceil N/(n-H) \rceil)}.$$

Note that H is fixed, N is fixed for each block, N and n together determine k', and the function $p(n)$ is fixed given the receiver site. Thus, we can now maximize channel utilization over all possible values of n and r, subject to the restriction that $p(n)$ is known only for selected values of n.

4) Observations Made Using the Channel Model: Table III gives the combinations of packet size and number of repetitions that maximize the channel utilization for selected block sizes, for the high-error channel mentioned before. Packet sizes of successive powers of two were considered, up to 1024 bytes or the block size (whichever was less). Suboptimal parameter choices, included for comparison, are indicated with (<). Note again that this channel does not reflect realistic receiving conditions, which are up to three orders of magnitude better.

In this table we can distinguish three different operating regions, namely those with optimal repetition rates of 1, 2, and 3.

TABLE III
OPTIMAL PACKET SIZE AND NUMBER OF RETRANSMISSIONS, AND RESULTING BLOCK DELIVERY PROBABILITY AND CHANNEL UTILIZATION, AS A FUNCTION OF BLOCK SIZE FOR A PARTICULAR NOISY CHANNEL

Block Size	Optimal Packet Size	Optimal Number of Repetitions	Block Delivery Probability	Corresponding Channel Utilization	
256	256	1	.7940	.7365	
512	512	1	.7045	.6780	
1K	1K	1	.5789	.5678	
2K	128	2	.7615	.3500	
3K	128	2	.6645	.3054	<
3K	64	2	.7246	.3056	
4K	64	2	.6508	.2745	<
4K	128	3	.9349	.2871	
6K	128	3	.9039	.2774	
8K	128	3	.8740	.2684	
10K	128	3	.8452	.2597	
10K	64	3	.9227	.2594	<
12K	64	3	.9079	.2553	
14K	64	3	.8935	.2512	
16K	64	3	.8792	.2472	

TABLE IV
BLOCK DELIVERY PROBABILITY AND CHANNEL UTILIZATION AS A FUNCTION OF PACKET SIZE AND NUMBER OF REPETITIONS, FOR BLOCKS OF 4 kbytes TRANSMITTED OVER A PARTICULAR NOISY CHANNEL

	Number of repetitions (r):									
	r=1		r=2		r=3		r=4		r=5	
Pkt Size	Deliv. Prob.	Chann. Utiliz	Deliv. Prob.	Chann. Utiliz	Deliv. Prob.	Chann. Utiliz	Deliv. Prob.	Chann. Utiliz	Deliv. Prob.	Chann. Utiliz
16	0.0000	0.0000	0.6441	0.1207	0.9889	0.1236	0.9997	0.0937	1.0000	0.0750
32	0.0002	0.0001	0.6936	0.2381	0.9839	0.2252	0.9993	0.1715	1.0000	0.1373
64	0.0027	0.0023	0.6508	0.2745	0.9683	0.2723	0.9976	0.2104	0.9998	0.1687
128	0.0098	0.0090	0.5805	0.2674	0.9349	0.2871	0.9917	0.2284	0.9990	0.1841
256	0.0283	0.0272	0.5325	0.2557	0.8873	0.2840	0.9773	0.2346	0.9956	0.1912
512	0.0636	0.0622	0.4974	0.2434	0.8238	0.2687	0.9465	0.2315	0.9844	0.1927
1024	0.1141	0.1127	0.4651	0.2298	0.7406	0.2439	0.8852	0.2186	0.9511	0.1879

For very small data blocks, it does not pay to retransmit packets more than once because the error rate on single transmission is so low that a channel utilization greater than 0.5 can be achieved, which is not possible when each packet is transmitted more than once. For such small data blocks, the largest possible packet size is always optimal.

Next, there is a crossover region where $r = 2$ is optimal. Somewhere within this region, it pays to reduce the packet size from 128 bytes to 64 bytes, as indicated. The model does not allow for accurate comparisons between error rates obtained with and without retransmission, so it is hard to determine the precise block size at which retransmission becomes worthwhile.

Next, for block sizes of 4K and up, the model yields a remarkably stable optimum with $r = 3$ throughout our range of measurement. The model indicates that throughout our range of actual block sizes (4K–10K), a packet size of 128 bytes is optimal. For very large blocks, the packet size should be reduced to 64 bytes.

Table IV shows how the block delivery probability and the channel utilization vary with the packet size and the number of repetitions for a given block size (4K bytes). Note that for $r = 1$, there is no error correction and therefore channel utilization is maximized with the largest possible packet size, because it yields the most compact encoding and thus minimizes the probability of error.

For $r = 2$, the *block delivery probability* is maximized with a packet size of 32. However, the overhead on these packets due to headers is substantial. Indeed, the *channel utilization* is maximized with a packet size of 64, despite a somewhat lower block delivery probability. Note also that channel utilization is bounded above by 0.5.

For $r = 3$, the channel utilization is bounded above by 1/3. This value is most nearly reached with a packet size of 128. This combination of repetition rate and packet size happens to be the best value in Table IV. In particular, nothing can be gained by increasing r since this would limit channel utilization to 0.25 (for $r = 4$), 0.20 (for $r = 5$) and so on.

Finally, compare the data for $r = 2$ and $r = 3$ for a packet size of 64 bytes. The channel utilization rates differ only minimally, but the block delivery probability is much greater for $r = 3$. In fact, it is about 50 percent greater, which nearly compensates for the fact that each packet is transmitted three times instead of twice.

5) Choosing Appropriate Parameter Values: If we had complete and accurate information regarding packet error rates for all receiver sites, we might be able to choose the packet size and retransmission rate for each data block so as to maximize some function of the channel utilization observed at all receivers. A policy decision would have to be made regarding the relative level of service to be delivered to nearby and distant users.

Resource limitations have prevented us from collecting accurate packet error rate estimates for all receiver sites. Moreover, the channel utilization figures computed with the aid of the model are only approximate. We presently set our operating parameters on the basis of available packet error rate estimates for several receiver sites we believe to be representative.

Note that for large block sizes, the channel utilization and the delivery probability decline exponentially with increasing block size, regardless of the packet size and retransmission rate. Thus, the broadcast protocol without acknowledgments is not effective for very large block sizes.

IV. INFORMATION PROTECTION

Since our broadcast system uses a public medium, we cannot prevent unauthorized users from listening to the broadcasts. Yet, the dissemination of information must often be limited. For example, there may be copyrights and other restrictions attached to the information to be broadcast. Moreover, in a commercial environment it is often desirable to offer the service only to paying customers. To enable such control over the dissemination of information, we encrypt all data blocks that are intended for a restricted audience.

A. The Encryption Protocol

Each block is encrypted using a combination of a master key and a randomly generated data block key. The data block key is different for each data block, and is transmitted along with it. The master key is secret: it is made available only to the legitimate users of the service. In practice, we employ a table of master keys, identified by number. Each encrypted data block carries a number identifying the master key that was used to encrypt it. Unen-

crypted data blocks are identified by a key number of zero. This scheme has the following properties.

- The information that is broadcast can be thought of as being separated into distinct logical streams, each with its own master key. Consequently, users can subscribe to certain services without having access to all services.
- The master key used for the encryption of a certain information stream may be changed periodically, for example once a month. Paying subscribers can be provided with the keys for the duration of their subscription. Since the key number changes along with the key, the receiver will automatically switch to the new key whenever a switch takes place. The key numbers in the table may be reused.

B. The Encryption Algorithm

Because of the hardware limitations of our receiver stations, we are unable to utilize better-known encryption techniques such as DES or RSA. Instead, we have implemented an algorithm which is very efficient and appears to afford a level of security commensurate with the value of the information we seek to protect. At an effective data rate of 2.2 kbits/s, the decryption utilizes about 6.8 percent of the available CPU time on an IBM PC. (All performance figures pertain to implementations written entirely in C.)

Data are encrypted as follows. Without loss of generality, assume that the key number is given, so that the 64 bit master key is known to both sender and receiver. To encrypt a data block, proceed as follows.

- Generate a random 64 bit key (the data block key). Its purpose is to ensure that a different key stream will be generated for each data block that is to be encrypted. Encrypt this key with the master key, using some standard method. The encrypted value will be transmitted along with the data block.
- Load the data block key into a 64 bit linear feedback shift register. For implementation efficiency, this register will be shifted one byte at a time, rather than one bit at a time (see Fig. 6 and discussion below).
- On each shift of the register, a nonlinear function is used to obtain a 1 byte quantity based on the contents of the register. Successive bytes from this stream are used as a key stream, and are combined with the bytes of the data stream using XOR, yielding the ciphertext.

The procedure for decryption is identical, except for the fact that the block key is not randomly chosen; instead, the value that accompanies the data block is decrypted with the master key, and loaded into the shift register. Successive outputs of the nonlinear function are combined with the bytes of the ciphertext using XOR, yielding the original message text.

The security of the system depends on the period of the shift register, and on the ability of the nonlinear function to hide the contents of the register. Fig. 6 shows the byte-oriented shift register in detail. On each shift, the new byte in the shift register is obtained by tapping three bytes from the register, shifting one of them right, shifting another left, and combining the three resulting bytes using XOR. The resulting shift register assembly can be pictured as a bank of eight 8 bit shift registers, flanked by two 8 bit shift registers containing only zeros, where the new bit in each of the eight registers is an XOR combination of 1 bit from the previous register, 1 bit from the register itself, and 1 bit from the next register (with identical tap arrangements for each register).

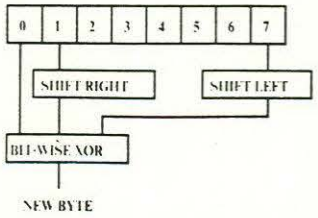

Fig. 6. The byte-oriented linear feedback shift register.

The tap positions were chosen to yield the longest period that could be obtained without shifting the bytes before the XOR operation. With shifting, the period has consistently been found to exceed 10^9.

The nonlinear filter is currently implemented as follows: four bytes from the register are combined into two 16 bit integers, which are multiplied; a single byte is extracted from the product, and returned. This method may be too structured to resist cryptanalytic attack. A system based on noninvertible table lookups may be more secure.

V. Implementation Experience

In this section, we describe our experiences in implementing the software of the receiver station on the IBM PC.

Throughout the design, we were constrained by the limited capabilities of contemporary personal computers (as embodied in the IBM PC). These limitations appeared in several areas.

1) Processing Power: It is essential that the system be able to keep up with transmissions: receiving characters, unstuffing control characters, checksumming packets, copying them into the data block buffer to assemble database entries, decrypting database entries, and matching them against the user-specified filter. We found that without resort to Assembly language programming, we were able to operate at a sustained data rate in excess of 4.8 kbits/s, except as indicated below. Responding to user queries did not significantly contribute to the computational load on the system.

2) Disk Access: The ultimate purpose of the software is to store database entries on the disk, and to read them back for display on request. We found that ordinary disk operations, such as opening a file, took several seconds. This posed a problem in the processing of incoming entries: while an entry is being saved on disk, the memory holding the entry cannot yet be released for incoming data, nor is the CPU available to process incoming data (except for individual characters, which are handled at interrupt level). Thus, a backlog of incoming data is created whenever an entry is written to disk or read from disk, or whenever the system state is checkpointed. Despite a buffer of 5000

bytes, capable of holding about 10 s worth of data, and despite a design that initiates a checkpoint only when the system is idle and the input buffer empty, packets are occasionally lost when the input buffer overflows. Due to error correction, this does not necessarily lead to loss of data.

3) No Multitasking with the Operating System: Some of the problems cited above would be alleviated if a user program could continue to run while a disk access is in progress. Unfortunately, the operating system does not allow this.

4) Disk Size: At present, a floppy disk can hold only about 320 kbytes. The object module and the on-line documentation file occupy about 70K. This leaves room for just about 40 average-size newspaper articles. We decided not to maintain a full-text index into those articles in part because of storage limitations. As more PC's become equipped with hard disks, this limitation will disappear.

5) Operating System Flexibility: The operating system does not facilitate integration of the receiver software into existing programs beyond the interface offered by the file system. Thus, our system has to provide all the functions that a user might desire, including performance analysis, building indexes to the database, and so on.

Despite these constraints, we feel that the system we have built is very usable. The most important performance problem in our first prototype was the delay due to disk access when reading files for display at the user's request. This has been remedied through the use of read-ahead and caching. Response times are now as follows:

- to create a menu of 5 database entry summaries and display it: < 0.5 s
- to create a menu of 25 database entry summaries and display the first page: < 2 s
- to fetch the first page of an article from the disk: < 3.5 s
- to display subsequent pages: < 0.5 s each provided read-ahead has quiesced.

VI. Summary

We have outlined a new type of mass communication that uses broadcast digital communication to transmit information of general interest to a very large user population at a low cost. Personal computers are used to isolate users from the time of transmission, to filter the incoming information, and to maintain a personalized database.

Our plans call for the addition of two-way communication between users and the central system to give users access to specialized and historical information as well as for interactive services. The two-way communication facilities will be integrated into our existing personal database system. Thus, users will not necessarily need to know whether their query is being processed locally or whether it has been forwarded to the main system.

This project brings together research from several disciplines. Based on the goals we outlined (Section I), we designed a database system that provides a hybrid of free-text and menu-based access (Section II). To deliver information reliably to remote computers over an unreliable broadcast channel, we use a parameterized communication protocol that incorporates packet-level redundancy (Section III). For one particular communication channel, we showed how to choose the parameters for the broadcast protocol by optimizing an objective function. For a given data block size the optimal packet size and number of packet transmissions depend on the error characteristics of the channel. Because the system uses broadcast communication, we have adopted a protection system based on encryption to allow fine grained access control (Section IV). We needed a bulk encryption algorithm that was reasonably secure yet that was efficient enough to be implemented in software on contemporary personal computers. The method that we chose uses a linear feedback shift register with a nonlinear output stage. Finally, we discussed our implementation experience (Section V). Despite the hardware limitations of contemporary personal computers, we feel that we have produced a very useful system, and technological advances will allow us to improve our user interface and maintain a larger local database.

Looking toward the future, we can see that the technology that we have used—digital mass communication and personal computation—can be employed for many applications in addition to home and business information services. One can imagine cars equipped with computers for monitoring traffic conditions, street corner kiosks that provide up-to-date information on community events, and electronic mail delivery to portable computers.

References

[1] H. Anderson and R. Crane, "A technique for digital information broadcasting using SCA channels," *IEEE Trans. Broadcast.*, vol. BC-27, pp. 65–70, Dec. 1981.

[2] R. D. Bright, "Prestel—The world's first public viewdata service," *IEEE Trans. Consum. Electron.*, vol. CE-25, pp. 251–255, July 1979.

[3] M. Lesk, "Combining data bases: National and cartographic files," in *Proc. Office Automation Conf.*, San Francisco, CA, Apr. 5–7, 1982, pp. 415–426.

[4] *Videotex Standard: Presentation Level Protocol*, AT&T, Bell Syst., Parsippany, NJ, May 1981.

[5] N. E. Tanton, "Teletex—Evaluation and potential," *IEEE Trans. Consum. Electron.*, vol. CE-25, pp. 246–250, July 1979.

David K. Gifford (S'78–M'81) received the Ph.D. degree from Stanford University, Stanford, CA, in 1982.

Since 1982 he has been with the Massachusetts Institute of Technology, Cambridge, where he is an Assistant Professor of Computer Science. From 1978 to 1982 he was with the Xerox Palo Alto Research Center, Palo Alto, CA. His research interests include computer systems and programming systems.

Dr. Gifford is a case study editor of the *Communications of the ACM*, and a member of the Association for Computing Machinery.

John M. Lucassen (M'84) received the S.B. and S.M. degrees in computer science and engineering from the Massachusetts Institute of Technology, Cambridge, in 1983, based in part upon research done at the IBM Thomas J. Watson Research Laboratory, Yorktown Heights, NY.

He is currently working toward the Ph.D. degree in computer science at the Laboratory for Computer Science at M.I.T. His interests include systems, programming languages, speech recognition, and policy issues.

Stephen T. Berlin received the B.A. degree in mathematics from Wesleyan University, Middletown, CT, in 1979.

He has been a researcher in the Laboratory for Computer Science at the Massachusetts Institute of Technology since 1981.

Mr. Berlin is on the Board of Directors of Computer Professionals for Social Responsibility.

Also published in IEEE Journal on Selected Areas in Communications, Vol. SAC-3, No. 3, May 1985.