# Network Servers for Symbolic Mathematics

Richard J. Fateman
University of California, Berkeley

December 9, 1996

**Abstract**

We describe advantages to using network socket facilities for communication and distributed computing from the perspective of symbolic mathematics systems. For some applications, an easily constructed Lisp server model provides a flexible portal between computer algebra and other programs, and one need not use new languages or new systems or write new stand-alone web-specific `cgi-bin` applications. Such socket programs can use, if necessary, HTML as a common transport encoding, but more efficient means are possible. We show that, rather than distributing all information to each computer algebra user's system, it makes sense to consider networking for accessing tables of information maintained at one or a few sites. Finally, we mention some consequences of the economic value of computation.

## 1 Introduction

We are faced with the availability of a variety of computer mathematics systems, commercial and academic, each of which contains some mix of services. The traditional symbolic computation systems provide algorithms for exact rational arithmetic, algebraic or differential equation solving, factorization of polynomials, symbolic integration, and so on. Newer systems tend to provide an environment that includes broader array of facilities taken from less-traditional areas such as numerical equation solving, high-precision, or interval arithmetic, typesetting, graphical display of functions in 2 or 3 dimensions, multi-media capabilities, library access to many application programs, formula table-lookup, and other capabilities.

If we can decide which single system has the best fit of features for us, then using that one system is, all things considered, easy: We buy a copy or download a free copy to our computer, and we learn how to use it from the manual, on-line help, courses, and experimentation.

If we find we need two different systems to solve a problem, we must face differing user interfaces, notations, and command semantics. We may also pay twice for the software (either financially or indirectly by suffering the additional hazards and costs of downloading two systems). If we need to run the systems simultaneously, we pay a penalty in memory and perhaps other resources.

Communicating between two systems, even those that have been defined to work together (such as the front/back end division used in several current CAS) is, at best, somewhat risky on several grounds: if they run on the same computer, they compete for resources, and in some unfortunate circumstances one can cause the other to crash. In the case of different system designs, they may have (perhaps subtlely) conflicting semantics for the same notations, or vice versa. To date, most efforts to establish standards for communication among systems have avoided discussion of the semantics. This natural, since it is a non-issue if only one system's front and back ends are being used: that system defines the semantics. As a consequence, some early inter-system communication designs tend to emphasize, as though it were the principle issue, the superficial language problems of syntax or elementary data-type issues. Other recent designs emphasize the human-computer interface, assuming that the semantics is solved if one can draw a picture of a mathematical expression. While a survey of these systems is beyond the scope of this paper, we mention just a few of them, for the reader interested in following links. We would include: Openmath[1], Polylith[2], MP[3] SGML-Math[4], ASAP[5], SWP[6], Mathlink[7], Zhao's system [9], GI/S, Mathscribe, etc. Saarela discusses SGML, HTML2 HTML3, PDF, Docbook, TEX[8][8].

We believe that this battle Roy-ale over what appears to be primarily notation, is a plausible as a first step but is hardly as problematical as semantics. This would, we believe, immediately surge forward as a more interesting and difficult problem if the syntax were to be settled in almost any

[1]http://www.can.nl/˜ abbott/OpenMath/index.html

[2]http://thumper.cs.umd.edu/files/docs/tr.html

[3]http://SymbolicNet.mcs.kent.edu/areas/protocols/mp.html

[4]SGML math DTD http://mish161.cern.ch/sc4wg6/math/math.htm

[5]http://www.inria.fr/cgi-bin/wais_rrrt-eng?PRJSAFIR

[6]http://www.thomson.com/tcisoft/default.html

[7]http:///www.wri.com

[8]http://www.cs.hut.fi/˜ jsaarela/thesis/thesis.html

of the numerous roughly equivalent ways. We too will not discuss semantics in detail in this paper, except to point out that when something *really useful* is available in a certain encoding, then interested parties will find it necessary to understand its syntax and semantics[9]. We hope the availability of useful results from TILU [2] will encourage such adoption, but that is a separate story.

In this paper we turn our attention to more basic questions that only touch on the semantics of what we are computing, but emphasize the nature of the communication.

- Why do it? In particular, what advantages accrue to the users? By contrast, another approach would be to avoid such interactions and proceed along the well-worn path for computer algebra systems: continue to accrete features into one monolithic system: kernel, libraries, more libraries, etc.

- How should we do it? In particular, we consider an engineering solution that make use of such communications without requiring each user to buy and install all the components.

We must emphasize that this differs from the usual "remote execution of a program." In our model the remote program is already executing before you use it, and may continue to execute (for someone else) after you are finished! The program package presumably is available on a *per-use basis*. Perhaps it can take advantage of unique (in the world) capabilities of libraries, algorithms, or special hardware.

In addition to these general issues, we describe a particular network application to table lookup of integrals.

## 2 Software server technology

Anyone who has used a time-shared system from a remote terminal has experienced one level of inter-system communication. Logging in to a remote computer to run a computer algebra system, or any other system, has certain benefits: the central computer may be fast and quite efficient — it may run re-entrantly many users of the same system. If there are enough cycles, and the connection bandwidth so very high that it can support needed interac-

[9] Just as we found it useful to learn about `html`

tion, this is a pretty handy model. It is one justification for supercomputer centers[10].

Many of us have experienced a client-server model, with local computation but perhaps remote file systems, or license servers. Computing power may be lower on the client, but each client runs only one user, and thus may provide better service. Since the client may be resource poor (e.g. memory capacity may be smaller) compared to the remote server, this may not be entirely effective.

A variation of these two models provides a front-end client program for input/output and communication, and a central server for "kernel" computations, whatever that may be. Sometimes the kernel computations are done in a separate process on the *client* computer. This can make for an uncomfortable shoving-match for resources when both components are near their limits.

The client-server paradigm may rely on a number of different techniques. One of them consists of maintaining a direct socket connection. We describe a variation on this, as well as two other models that are less common: a web-browser connection, and e-mail server. We have implemented the first, second, and (partially) the third.

Others have explored and continue to explore the territory of WWW use. For example a generalized scheme with a specific example for distributed statistical decision support via remote software modules is described by Günther *et al.* [5].

### 2.1 Internet Sockets and Lisp

The architecture we use in our experiments is a UNIX[11] process which is actually a Lisp program that listens on a particular internet (or WWW [1]) port for queries. Another computer can choose to read and write to this program by connecting to that port via `tcp/ip` from anywhere on the internet. The Lisp program, for its part, reads from the port, forks off an independent "light-weight" process (in our case to look up an answer) and the process, after a brief computation writes the answer back to that port. This is the primitive notion on which we build our service.

It differs from the previously mentioned client-server model of computer algebra systems front/back ends in that we maintain just ONE program on

[10] A better one is that it helps to have a way to hire excellent support personnel.
[11] Other network operating systems can do this as well, though the details differ.

ONE port that can handle any number of separate inquiries from separate clients[12]. We do not start up, *ab initio* on request, a new server "kernel" for each client, nor do we (ordinarily) offer to maintain any separate context over time for each client. Our server is more like a grand-master chess player playing any number of opponents each with a separate chess game. No opponent monopolizes the grand-master for more than a brief moment. Since our goal is to process most queries in under 10 milliseconds, it is clear that the time to start up a new server would exceed this threshold.

### 2.2 WWW browsers and servers

Handling sockets requires programs at both ends. A friendly and uniform user interface at the server end would be nice, but programming this for any possible platform and distributing it is difficult. Why not have access be from a standard network browser such as Mozilla (from Netscape), or Mosaic (NCSA), or Microsoft's Explorer, or the text-only browser, Lynx? This way everyone already has the interface, and our socket can be accessed from nearly every computer that is internet-capable.

This seem quite attractive: if we program our socket handler to deal with the clutter generated by a full WWW `http` connection, and then process the query extracted from the page, then finally return the result as an html page, we are all set. In fact, we were able to obtain an implementation of a WWW-server written in Lisp, and so the "usual" route of writing programs in C, C++, Perl, etc were quite unnecessary: the capabilities of computer algebra, table-searching, and WWW-serving are available in one program which silently lurks on an internet port until needed.

How does this work in more detail?

To support web browsers one ordinarily has a web "page" in a public (readable) areas that is merely a text file describing a form in the `html` language.

When a browser accesses certain kinds of forms, the user can be invited to fill out data and signal "submit". In this way, data can be transmitted back to the server. One scenario from that point on is to have the web server start a UNIX shell script (often called a `cgi-bin` since that is the directory where the scripts are stored) to execute a program on the server. This same script then reads the result of that computation, and generates

[12] perhaps in apparent simultaneous transactions

the appropriate html to display the result by transmitting it back to the web client's browser to display.

Rather than set up a cgi-bin program, we have the Lisp server directly listen to the port to analyze the input. This analysis includes the removal of irrelevant protocol chit-chat, the decoding of the vital informationhtml[13] computing the output, and sending a formatted html page back.

What happens if multiple queries may arrive at the same time? All tcp/ip implementation handle the problem of multiple connectors to a single port. We use Lisp's multiprocessing facility to run each problem as an independent thread within the same lisp image.

To illustrate the simplicity of the idea, here is the main loop of a server program written by J. Foderaro of Franz Inc. It just blocks (non-busy) and waits for a connection from the outside world. Then it starts a lisp thread to service that connection and the main lisp thread goes back to wait for another connection. As an option here, it also keeps track of how many connections have been made to date, in the variable con-count.

```
(defun www-server (port)
  (let ((socket (make-socket :connect :passive :local-port port))
        (con-count 0))
    (unwind-protect
        (loop
         (let ((con (accept-connection socket)))
           (mp::process-run-function
                (format nil "www-con-~s" (incf con-count))
                #'www-connection con)))
      (close socket))))
```

The program www-connection is handed a connection from which it can read, and to which it can write. After straining out the babble of headers and such, the material on a submitted page can be read, parsed, evaluated, etc.

Assuming that the processing is done without the use of global variables, and can be guaranteed to terminate (or to be cut off by a timer if it runs too

[13]Html defines a fairly verbose and clumsy language. Among other encoding tricks, it has a baroque escape mechanism. Instead of putting a string in quotes, all non-alphanumeric characters are sent as three characters encoding one byte in ASCII, except for a space which is "+". Go figure.

long), the possibly multiple threads are independent. If the Lisp program is cautious about guarding its host computer (by forbidding local input or output, or any command-shell interactions) it is also quite safe, at least from invasion. In debugging from remote sites, typically the worst that happens is that a thread fails in some fashion and does not return a page in a reasonable time. The remote user will eventually get an error message from the browser that the remote computer has not answered. (Or the impatient user will signal "stop" with the same result).

### 2.3 Using E-mail

The third way to connect to a computer without logging in (etc) is to send mail. Clients (users) can "connect" to the math server by sending e-mail to a designated address, and wait for a response. The server could be designed as a free-standing program that is invoked on receipt of mail, and which computes and returns an answer.

In our case, the server could be designed as a very brief free-standing program that starts up when mail is received and pokes the waiting Lisp process by connecting to a socket.

It is easy to specify in the sendmail `alias` file of a UNIX system that a program should be started on the receipt of mail, rather than the default behavior which just stores the mail in a file. The alias file might contain: `CASserve: "|/usr/bin/ConnectCASMailToPort"`. In this case, mail sent to `CASserve@machine` would be handed to the named program.

Although we have programmed this (in fact, we wrote it before the socket-based protocol) we have not bothered to debug it because it is much less fun than an interface from `http`.

## 3 Why do this?

Just because it is possible to set up a server for a computation does not justify its existence. It would certainly be silly to set up a server that took strings like "2+2" and returned "4".

We divide good reasons into *unique capabilities*, and *convenience*.

### 3.1 Unique Capabilities

Here are some examples of plausible uses.

**Integration lookup** As previously reported [2], we have produced a program named TILU that can rapidly look up integral expressions given to it, based on similarities to expressions gleaned from printed as well as electronic compilations of formulas.

While this work needs further refinement, and we are in the process of increasing the number of formulas, it can easily incorporate results not obtainable by algorithms in CAS today[14]. Building on past activities, we propose to make these results available on an as-requested basis via network access.

Why, you might ask, is this better as a network service than as a traditional program, distributed by network FTP, or traditional distribution channels (CDROM, floppies)?

1. The table can grow indefinitely at one site.

2. Corrections can be incorporated as they are discovered and take effect immediately and for all users.

3. The time to load a (potentially huge) program on a (potentially small) computer is avoided. This program, already loaded, awaits questions.

4. Submissions and suggestions for additional formulas can be considered in a timely fashion, perhaps tested for validity, and incorporated.

5. Downloading tables may be a violation of intellectual property rights. By contrast, the several publishers we have consulted with view the delivery of an answer, especially one that has been specialized to the user's variable names etc., as fair use of a table.

6. Expressing the answers, in what, on occasion, is a more sophisticated language than is used by CAS today, may be a driving force to make the CAS more capable. The appearance of special function names is one instance of this; we are also interested in providing answers requiring CAS to deal with domain restrictions such as "integer $n > 1$ and Real $z$ in $[-1, 1]$". Although we realize our results are now most often used directly by humans, our goal is to have CAS absorb the result and *continue to compute with them.*

[14] No commercial computer algebra system today appears capable of adequately representing the more sophisticated answers from large tables, much less computing them from first principles.

**Other tables** We can envision other uses for such mathematical tables, for example in classifications in group theory, specialized factorizations, or other kinds of searches. The most favorable kind of problem is one in which the question is small, and so is the answer. It is only worthwhile if the intermediate processing requires substantial specialized data on-line. Looking up information in a database (e.g. mapping names to telephone numbers, or using a web search engine) are by now classic though non-mathematical instances.

Another common application is that of a news server which redistributes newsgroup messages, probably of only temporary interest, on request. Here the request is small (e.g. "tell me if there are new messages on "sci.math.symbolic") and the response is typically moderate in size.

Along the lines of our integration server, we are exploring the possibility of closed-form differential equation solving. This has a substantially different feeling to it in the sense that many differential equations that do not "look the same" may, under some transformation, be quite similar. We are therefore looking at classification techniques that are based less on appearance than on invariants or symmetries. Presumably we would benefit in solving a larger set of differential equations, and storing a smaller number of distinct solutions.

### 3.2 Convenience, comparison shopping

**Timing Benchmarks** Before I buy a system, I might wish to try out a command to see how fast or capable it is.

**Accuracy Benchmarks** Before I buy a system, I might wish to try out a command to see if it can get the correct answer to a sample problem.

**Compilation requests** I don't want to buy your expensive compiler, or perhaps I do not have the capacity to run it in its entirely, but I would like to see the following routine expertly compiled and downloaded back into my system so I can make better use of the version of the system I bought. This or similar kinds of computation may be envisioned in some cases as a software "lease", if a charging mechanism is put into place.

**Consulting with an expert system** Is there a problem in your archives that looks anything like this? If so, please tell me about it.

**Download requests** Some computations make more sense to run locally, but it may be plausible to download the latest solution *methods* on short notice. Consider some CAS command that is still under development. The user can ask if a version has been archived for downloading that is more recent than the local one. If so, it could be downloaded and stored. The online libraries for Maple and Mathematica could be more useful by far if one could eliminate all bug-repair responses of the form "Your bug was present in an earlier, now obsolete, version of command X. To correct your problem, download the current file X.command". A similar version of this would be to have locally a cache of data (say, of the last 10 years of stock-market information), but update the information daily, or on request.

**Downloading macro-defined data** We consider the possibility that some solutions are very large, but one could download an encoding—a program and some initial data—to make it possible for the client to reproduce the answer locally. This could be done by downloading some "macro-expansion" programs for the client to run. This is easily done in communicating between Lisp systems, since one can down-load programs as easily as data (programs look just like data!). There are insecurities in allowing a server to download programs into a client environment (this is what Sun's Java is all about), but the prospects for useful computation are significant.

**Specialized computations** There are undoubtedly other briefly stated questions that can be of interest for remote computing, though any CAS "command" that can be briefly stated and has a smallish answer is grist for this mill. One can imagine a computer server equipped with a quantum "coin-tossing" device that could serve as a generator for random (not pseudo-random) sequences of numbers. A coordination server (like the please-take-a-number machine in a bakery) may have use as well.

# 4 Details of the integration table-lookup service

At this time we have a prototype server running on our integral table. A typical interaction spends about 0.005 to 0.01 seconds of CPU time for the table lookup, and an additional effort expended in display and formatting of the answer for human consumption. We allow user input in either of two forms, a Common-Lisp style expression form, and an expression form that is compatible with WRI's Mathematica, actually based on our own (free) Lisp implementation of a Mathematica-language parser.

Additional parsers for Maple, Macsyma, and other CAS probably should be added. Indeed, any other format that is compatible with ascii transmission such as SGML or as proposed by the OpenMath model should be possible. Yet other forms (e.g. encoded binary used by MP) should be possible as well.

Currently the answer is given in Lisp form as well as a 2-D display form. These could be easily changed, however.

The input page (which is experimental, and may not be up at any given time) should be available as `http://http.cs.berkeley.edu/~fateman/htest.html`.

Since arbitrary Lisp computations can be executed in the course of READING an expression, and we do, in the case of Common Lisp input, use the Lisp reader directly in our web server, the read-time macro expansion character `#` is disabled. Any Lisp expression can be submitted, although only those that can be interpreted in the "expression algebra" specified on the web page will be considered as legitimate inputs.

The Mathematica language has a UNIX shell escape character which we also ignore.

Since we wish to provide complete access to our data we must provide the opportunity for the user to utter additional specifications for the integral table-lookup application:

- If there are multiple hits in the lookup table, we must specify how many results should be supplied (default now: all, with appropriate "provisos" specified). As an example, in our current table there are overlapping formulas for indefinite integrals of $x^n \sin(ax)$ for positive integer $n$ since there is a general form as well as specific formulas for $n = 3$ and $n = 2$. Each, all, or perhaps none of such overlapping forms may be relevant to a given query. The given exponent might be unknown, might be 2 or 3, or not a positive integer at all (in which case no candidate in our table really matches). An earlier version of our program provided a rather verbose result including these many results; we've changed it so that you can see approximately what one gets from our current program (August, 1996). We are still experimenting, however.

```
;; query is x^n*sin(q*x), indefinite integral.
```

```
;; sample answer [omits some table data, acquisition numbers etc.]


 (+ (* (SUM
        (* (EXPT -1 R_) (EXPT Q (+ -2 (* -2 R_)))
           (EXPT X (+ -1 N (* -2 R_))) (FACTORIAL N)
           (EXPT (FACTORIAL (+ -1 N (* -2 R_))) -1))
        R_ 0 (FLOOR (* 1/2 (+ -1 N))))
       (SIN (* Q X)))
    (* (COS (* Q X))
       (SUM
        (* (EXPT -1 (+ 1 R_)) (EXPT Q (+ -1 (* -2 R_)))
           (EXPT X (+ N (* -2 R_))) (FACTORIAL N)
           (EXPT (FACTORIAL (+ N (* -2 R_))) -1))
        R_ 0 (FLOOR (* 1/2 N)))))
```

or displayed more readably:

```
      R   -2 - 2 R   N - 2 R - 1
    -1   Q          X              N!
Sum(---------------------------------, R, 0, FLOOR(1/2 (N - 1))) SIN(Q X)
            (-1 + N - 2 R)!

 +
       1 + R   -1 - 2 R   N - 2 R
     -1       Q          X         N!
 Sum(--------------------------------, R, 0, FLOOR(1/2 N)) COS(Q X)
               (N - 2 R)!
```

- If simplification of input and out is going to be taken care of by the client, thereby saving time in the lookup, this should be specified. (The TILU simplifier is not very sophisticated).
- If there are special comments that should be attached to the answer, the user may supply them on the input page, and they will be echoed on the return.
- The limits of integration must also be specified (default is indefinite

integral).

- The language for the input (and the output) can be specified.
- Special restrictions on variables (e.g. `n, m` are positive integers) can be specified.
- The verbosity of the result might be specified to include the provenance of the formula(s), the results in "typeset" style form, the explicit mapping between user-parameters and table-parameters, and other information in the data base that might be of interest.

Additional output information can also be supplied:

- Our acquisition number for the formula (which can be keyed to the source for the formula), but is our internal unique key.
- The source-internal page and numbering of the formula (its provenance).
- Assumptions or provisos as illustrated above.
- Notes (comments) supplied with the query.
- Time of arrival, elapsed (cpu and real) time for processing the query, etc.
- (perhaps) Billing information.

## 5 How would this be used?

At the moment, the interface is set up so that a user of a CAS on a networked workstation could copy an unsolved integral and paste it into our page, press a button and see if the answer can be found. For our local connections, this usually takes a second or two. The bulk of the time is in network transmission and local redisplay of the answer page, rather than CPU time.

At the moment, the result in Lisp form may be copied out of the answer page and pasted into the (Lisp-based) CAS. With a modest effort in programming, the result could also be supplied in the input-syntax of popular CAS.

This does not necessarily solve all problems inherent in the communication: syntax is only part of the issue. A more significant programming effort is needed on the CAS side to deal with the much richer *semantic* language of the published answers to integrals—domain restrictions in particular.

The page-in/page-out communication via a network browser is intended to be intuitive for individual users, but it requires the ability to type expressions in a linear syntax. To date it appears to be used mostly by unsophisticated web browsers or students seeking homework solutions. We hope it might be used by a pencil-and paper mathematician who would normally consult a printed table. But the interaction is rather chatty.

As indicated earlier, an alternative is to allow programs to generate and read e-mail, and do the whole transaction as a text-based "hands-off" interchange. The encoding of a problem in mail is fairly straightforward, although details remain to be specified. Most CAS programs can execute a "send-mail" command. Unfortunately, realizing that an answer has been received is more difficult, since CAS do not usually deal at all with asynchronous input from any device other than a keyboard or mouse. Thus the mail communication seems inadequate for systems, although the freshman calculus student with e-mail access only might find this interface acceptable.

A more direct and stream-lined system-to-system interface can be set up, but it involves a modest amount of system programming. This is not likely to be supported by the typical CAS user-style programming language. It involves opening an internet TCP/IP socket to a named server. The question is then written to this port, and the answer read from this port. In languages like C or Lisp, this is hardly distinguishable from file input and output.

On our (server) end we are already doing the corresponding reading and writing; The `http` interface is built upon this, providing a superstructure of (mostly irrelevant) chatter. All that `http` does is to send to the socket a line of text something like this:

```
theintegrand=a+x%5E3+%2Bb*x%5E2%2Bc&ChooseLang=Mathematica
&varint=x&ChooseLimits=Indefinite+integral&"
```

(we have inserted a new-line character for display here, but it is not be in the original). The obscure encoding is an artifact of `http`, not us. By by-passing an `http`-to-normal text translation filter, we could easily (and gladly) handle this equivalent message:

```
theintegrand=a x^3+b*x^2+c
ChooseLang=Mathematica
varint=x
```

```
ChooseLimits=Indefinite integral"
```

A correspondingly simple output form, absent the `html` blather to display it on a page can also (gladly) be returned. This has been made available used by Bronstein's $\sum^{it}$ differential equation solver.

# 6 Marketing, Payments

Payments for internet service is an area of great interest and there are several models available. We refer the reader to internet sites such as `http://www.digicash.com/ecash`, `http://www.NetBill.com` for examples of how payments can be transferred.

There seem to be a few models that make sense here. One is that all users in a domain can have access to a facility for an annual subscription fee. For example, my university subscribes to certain external on-line services. Any Berkeley client can access these services. Services may be provided "free" but only to members. My university provides full access to its own library but only to net clients from within the university.

Another model requires the provision, probably via some pseudo-secure transaction, a credit card number for billing. This is generally thought to be unsuitable for small charges since there is about a $0.30 minimum for the cost of billing.)

Micropayment issues are discussed by Hardy and Tribble in their Digital Silk Road proposal [6]. While we hypothesize a fixed, pre-agreed upon price for a transaction, a more elaborate solution is possible. Hardy and Tribble even suggest a computer-algebra (!) scenario:

> I want to integrate a function or factor a polynomial. I send the problem to a Mathematica server together with $.05 and get the response in a few seconds. I probably get the answer but the response may be that $.05 of computing yielded no answer but more might do the trick. For a while it will preserve the state of the computation for resumption if I wish to send more money. The response might alternatively be that it knows the answer but will only reveal it for $.25.

While the Silk Road proposal involves encoding cash in each network packet, other simpler forms of electronic cash have been proposed and in some cases implemented via direct encryption and verification.

NetBill supplies details for a possible internet billing protocol that does not require electronic cash, but relies on the user depositing credit to an account, typically in amounts of \$50, typically via a single credit card transaction. This could then be used for transactions of as little as \$0.05, (where about \$0.02 would be subtracted as a transaction charge)[15]. The NetBill protocol is suitable only for delivery of "information goods" which is precisely what we need.

Another model, used by information providers as well as commercial broadcast television and radio stations, is to provide a free service, however interspersed with paid advertisements. Is that the audience for integration results a good target for advertisers? For charges of below \$0.05, the advertising route is thought to be the best choice; this would be a low price to ad advertiser for delivering messages to a targeted audience[16].

From a commercial standpoint information is quite different from (say) selling pizza, which can presumably be eaten only once. We are offering information that can be delivered electronically, copied indefinitely without being destroyed, and whose provenance, accuracy, and ownership are all subject to dispute.

It is generally conceded that a mathematical formula, like a recipe, cannot itself be copyright protected; furthermore, looking up a formula, and one involving substitutions and translations, in a 1000 page book for someone else, seems like fair use. Not all net services would fit this model.

Whose rights are being trampled if we cloned our lookup service, and sold servers including the "translated tables" from published references, to clients? In effect, we would be selling vending machines, where the vending machines dispensed not coffee, but integrals.

## 7 Further Work and Conclusions

Network-based computing and information access is clearly here to stay. It appears that network servers accessible to Netscape/Explorer or similar browsers can provide a wealth of mathematical information rapidly, at low cost, in a useful form for end-users and systems. We've just touched the surface, and given an example of how this can be handled.

We barely touched on additional issues that range from the commercial to the intellectual:

[15]http://www.NetBill.com

[16]Assuming there were humans reading the result, and not just another computer!

- Standardizing communication of simple mathematical data.
- Identifying significant representation problems of non-standard data.
- Identifying other plausible services.
- Safeguarding intellectual rights.

Our current program successfully answers queries from numerous versions of browsers including Mozilla (Netscape), Mosaic (NCSA), IBM WebExplorer and Lynx. To the extent that we can decipher the nature of the recipients, they appear to be on Mac, Windows, and UNIX systems from a variety of manufacturers. As of November, 1996, some 5500 queries have been fielded.

The default input language parser accepts Lisp or at the user's option, Mathematica syntax. Other syntax consistent with (say) Macsyma[17], Maple, Reduce,Axiom, Derive, MuPAD, or other existing CAS systems could be added if, for each of these, requiring only that a parser, written in Lisp, were written or adapted for this use. A Java front-end down-loaded into a browser could help a user who is unskilled in such "linear syntax" for mathematics, construct via templates an expression. Our evidence to date is that the majority of users are unable to write in either of the provided modes at a first try[18].

In further work, we expect that we will see expansion of the particular integral server table, through manually entering new formulas, reading printed tables by OCR, and generating formulas algorithmically by various generating methods. We are especially interested in the CD-ROM produced by Academic Press of its Gradshteyn/Rhyzik reference [4] which includes TEX style data. While this is not ideal as input, it represents a post-OCR stage that can aid our efforts to digitize the formulas.

## 8 Highlights of Ancient History

Some of the earliest hosts on the Arpanet provided computer algebra service under timesharing. Circa 1970, the Macsyma system was on the MIT-ML,

[17] If we can successfully pry a Macsyma parser out of its system context for re-use, that may be the next language extension.

[18] This adds fuel to the fire about experiments in teaching calculus with computer algebra. It suggests that typing mathematics into a computer is an additional barrier — yet one more difficult topic to cover —in a calculus course!

and then MIT-MC computers. Reduce was also an early resident of the Arpanet. As far as pre-WWW widely-used network service, one standout is `netlib`[19] and its predecessor versions. This work is substantially centered on the distribution of mathematical software by ftp or email, but is rather general purpose.

The emergence of the WWW has changed the world of computer algebra, along with much else. The evolving activities and systems can be observed on-line[20] more easily than we can track them in a static note.

## 9 Acknowledgments

Thanks to Ted Einwohner for continuing improvements to the TILU program, and in pounding on the network socket code. Thanks to John Foderaro of Franz Inc, for providing critical pieces of Lisp software to implement the www-server program. Dan Halpern has also contributed to the design of Tilu. Additional thanks to Taku Tokuyasu, Paul Zimmermann and to many others who, generally anonymously, have used Tilu and provided grist for our mill. This work was supported in part by NSF Grants numbers CCR-9214963 and IRI-9411334, and by NSF Infrastructure Grant number CDA-8722788.

## References

[1] T. Berners-Lee, R. Caillaiau, A. Luotonen, H.F. Nielsen, A. Secret. "The World-Wide Web," *CACM 37 (8), 1994* 76—82.

[2] T. Einwohner and R. Fateman, Automated Integral Tables, ISSAC '95.

[3] Reduce differential equation mail server. (sci.math.symbolic announcement)

[4] Gradshteyn and Rhyzik. *Table of Integrals, Series, and Products* Academic Press (5th edition), (also CDROM) 1996.

[5] O. Günther, R. Müller P. Schmidt, H. Bhargava, R. Krishnan. "MMM: A WWW-based method management system for using software modules remotely," preprint. `http://icsi.berkeley.edu/~oliverg`.

[19] http://www.netlib.org

[20] http://SymbolicNet.mcs.kent.edu/

[6] Norman Hardy and Eric Dean Tribble. The Digital Silk Road, `http://www.agorics.com/agorics/dsr.html`.

[7] Anthony C. Hearn. Computer algebra and the internet, DISCO 96, Springer LNCS 1996.

[8] Janne Saarela. Using hypermedia to provide application support. Helsinki Univ. of Technology, 1995.

[9] Yanjie Zhao. *Knowledge-Based Parsing Method of Mathematical Notation for Improving Mathematical Computation Environment,* PhD Diss. Dept of Info. Engineering, Nagoya Univ., Sept, 1996.