A Case for Benevolent Viruses

by Dr. Frederick B. Cohen *

* This research was funded by ASP, PO Box 81270, Pittsburgh, PA 15217, USA

In recent months, a controversy has arisen in the electronic and print media as to the viability of benevolent computer viruses and the morality of a contest to find useful applications of this technology. In this paper, we discuss the issues related to applying computer viruses for good instead of evil.

We begin with some background on viruses and related topics in `life-like' computational organisms. Next we examine some of the major problems facing the current global computing environment and how viruses have the potential for helping to solve these problems. We then consider several widely stated arguments against the application of computer viruses for useful purposes and provide counterpoints.

Background

On March 22, 1991, the world high speed computing record was broken by a Massachusetts company specializing in parallel processing. (See the New York Times of that date for details.) The previous record holder, contrary to popular belief, was a computer virus written and distributed in the Internet, one of the World's largest computer networks, by Robert Morris, then a graduate student at Cornell University. (See Eichin and Rochlis, ``With Microscope and Tweezers: An Analysis of the Internet Virus of November 1988", IEEE-CS Oakland Conference, May, 1989.)

The `Internet Virus' was a relatively small computer program written in the `C' programming language. It was designed to replicate itself in computers networked to Mr. Morris's and use the processing and communication capabilities of these computers to spread to other networked computers. Because of a fundamental design flaw, the Internet Virus spread too quickly, and its exponential growth caused widespread denial of services to Internet users over a two day period. Eventually, Robert Morris, like his virus, was caught. Mr. Morris was tried and convicted of unauthorized access to `Federal Interest Computers', and to pay for his transgression, he had to pay a fine and perform community service, and was kicked out of graduate school, but his virus stands as a startling example of the potential of computer viruses for both malicious and beneficial purposes.

The design flaw that caused unchecked growth demonstrates one of the malicious aspects of this virus, and unfortunately, many of the other computer viruses we hear about are also malicious, but like any new technology, viruses are a two edged sword. Consider that the Internet Virus performed about 32 million operations per second on each of 6,000 computers, and another 3.2 million operations per second on each of 60,000 computers, for a grand total of 384 Billion operations per second! It took hundreds of person-years of work and millions of dollars to design the computer hardware and software that broke this processing record, while the Internet Virus was written by one graduate student in his spare time over a period of a few months. For pure processing cycles, computer viruses are some of the fastest distributed programs we know of, but unfortunately, we haven't yet grown enough scientifically or ethically to exploit their vast potential.

The same issues that make viruses a serious threat to computer integrity (F. Cohen, ``A Short Course on Computer Viruses", ASP Press, PO Box 81270, Pittsburgh, PA 15217, USA, 1990) make them a powerful mechanism for reliable and efficient distribution of computing. They distribute freely, easily, and evenly throughout a computing environment; they provide for general purpose

computerized problem solving; and they are very reliable even in environments where computer systems fail quite often.

Efficient and even distribution of computing between computers working together on the same problem is one of the hardest problems we face in parallel processing. For large parallel processors working on complex problems, it is sometimes more difficult to figure out an efficient way to distribute the problem solving among the available computers than it is to do the problem solving once the processing is distributed. With computer viruses, we get even distribution based on available processing because computers with less available processing tend to be slower to replicate viruses, while computers with more available processing tend to provide faster replication. Since viruses can spread wherever information spreads and is interpreted, (F. Cohen, ``Computer Viruses", ASP Press, 1985) viruses can eventually distribute themselves throughout networks regardless of how computers are connected. These two features eliminate the need to spend time figuring out how to distribute processing across computers.

Although general purpose problem solving is rarely a problem in computers today, reliability is particularly critical to large parallel processing applications because as the number of computers involved in problem solving increase, the likelihood of a failure during processing also increases. (See D. Sieweorek, ``The Theory and Practice of Reliable System Design" Digital Press, 1974) The Internet Virus continued processing even though many systems in the Internet were turned off and many subnetworks were disconnected in an effort to stop it. Few modern parallel processing applications could continue processing in this sort of environment. In fact, most parallel processing computers today could produce erroneous results without even producing an error message, much less processing correctly when some of the computers fail. Viruses on the other hand have inherent reliability because of their ability to replicate and spread. Most of the computer viruses we know about work correctly in a wide variety of computer makes and models, work in both networks and isolated systems, work in many different versions of operating systems, spread to backup tapes and are revived when backups are restored, work on floppy-disks and hard-disks, and survive system failures and reconfigurations. Some of them even operate across different operating systems and types of computers.

Efficient and reliable distribution of processing in itself is not always enough for efficient problem solving. For some problems, we have to be able to communicate results between processing components, while in other problems the time required for processing is too small to justify the time required for distribution. The problem of controlling virus growth must be addressed before widespread use of viruses in existing computer networks will become acceptable to the user community. Evolution of viruses over time will probably be a vital component to their long term utility. Many issues in viral computation are not yet resolved, but there is also a substantial body of knowledge and experience to draw from.

The use of self-replicating programs for parallel processing is not new. In fact, John von Neumann, one of the pioneers of the computer age, described reliable self-replicating programs in the 1940s. (J. Von Neumann, ``The Complete Works of John von Neumann') In many early works, the `living' computer program was not just a distant possibility, but the intent of the exercise. Early papers talked of `making reliable organisms out of unreliable organs', and `intelligent self-organizing systems with numerous components'. Over the intervening 50 years, many authors have reported isolated experiments, and slow but highly disorganized progress has been made.

The Worm Programs, Computer Viruses, and Artificial Life

In 1982, Shoch and Hupp reported a series of successful experiments with parallel processing using self-replicating programs that spread through the Xerox computer network. (J Shoch and J Hupp, ``The 'Worm' Programs - Early Experience with a Distributed Computation", CACM pp172-180, March, 1982.) To quote this paper:

> ``A *worm* is simply a computation which lives on one or more machines. ... The programs on individual computers are described as the *segments* of a worm ... the worm mechanism is used to gather and maintain the segments of the worm, while actual user programs are then built on top of this mechanism."

These so-called `worm' programs would install `segments' on computers which were not in use, each performing a portion of the parallel processing problem being solved. Whenever a person wanted to use a computer, they pressed the reboot button, and normal operations would resume. During the day, the worm would be trimmed back, running only on a few computers that were not in use, but at night, it would become active all over the network, performing tens of millions of useful calculations per second. Unfortunately, an error in one copy of the worm ended their experiments by causing the global Xerox network to reboot to the worm instead of the normal operating system. The entire network had to be restarted. Shoch and Hupp also reported a number of worm programs run on the Arpanet during the 1970s, some of them apparently capable of limited replication. For unspecified reasons, Shoch, Hupp, and the other worm researchers apparently stopped performing these experiments in the early 1980s.

In 1984, Cohen reported the first experiments with `Computer Viruses' as we usually think of them today (F. Cohen, ``Computer Viruses - Theory and Experiments", IFIP computer security conference, 1984, also appearing as an invited paper in IFIP `Computers and Security', V6\#1, Jan. 1987 pp23-35 and many other places.) To quote this paper:

> ``We define a computer `virus' as a program that can `infect' other programs by modifying them to include a possibly evolved copy of itself."

These `Viruses' had many implications for integrity maintenance in computer systems, and were shown to be quite dangerous, but Cohen also pointed out their potential for good, and even showed a practical virus which reduced disk usage of computer programs in exchange for increased startup time, a technique that is now commonplace.

A common misperception about viruses in the research community due to this definition is that they must modify other programs, but this special case is covered in the formal definition for viruses which encompasses all self-replicating programs and programs that evolve and move through a system or network. (F. Cohen, ``Computer Viruses", ASP Press, 1985, PO Box 81270, Pittsburgh, PA 15217, the formal definition from this work subsequently appeared in the IFIP journal `Computers and Security' under the title ``Computational Aspects of Computer Viruses" in 1989) This encompasses many of the `worm' programs under the formal umbrella of computer viruses. Cohen also pointed out the close link between computer viruses and other living systems, and even melded them into a unified mathematical theory of `life' and its relationship to its environment. Dr. Cohen's experiments were terminated rather forcefully because they were so successful at demonstrating the inadequacy of contemporary computer security techniques, that administrators came to fear the implications.

In the mid 1980s, Scientific American began publishing a series on a mathematical `game' called `core wars' (A. Dewdney, ``Mathematical Games - a series of articles in Scientific American, 1985-1988), in which two or more competing programs struggled for survival in a simulated computer. One of the most successful programs was a virus that replicated itself and then started executing code from the replicated copy, thus giving the appearance of `moving' through memory. The game of `Life' which simulates `living cells' in a cellular automata has existed for quite a long time. (J. Conway invented this game in 1970 while at the University of Cambridge) To the extent that they replicate and/or evolve within the environment, they meet the mathematical definition of a computer virus. These examples of viruses have met with no significant resistance, presumably because they have no widespread impact. The same cannot be said for more practical experiments.

In 1986, the first experiment with a PC based network virus was performed by several graduate students at the University of Texas at El Passo, who found that the virus spread to 60 computers in

30 seconds. These experiments were quickly terminated because they were so successful that researchers feared the consequences if the viruses escaped. In 1987, a graduate student researcher at Penn State University was forced to terminate experiments because of fears from members of the university community, even though there were no problems related to this research. In early 1988, a professor at the University of Cincinnati was forced off the University computer systems because a systems administrator saw the word `virus' in the professor's computer account and decided it was too much of a risk to allow that sort of work. These `knee jerk' responses have made the road for legitimate research on the beneficial implications of viruses very difficult, but despite these encumbrances, the research has continued.

In 1987, the first `Artificial Life' conference was held, with researchers gathering from around the world to present papers on a loosely knit combination of many independent research areas that seek to describe, simulate, analyze, or implement living or life-like systems. (C. Langton, ed. ``Artificial Life'' 1989, Addison-Wesley Publishing Company, New York, NY.) In recent years artificial life has received increased attention both in the popular and the scientific communities, partly because of the emergence of computer viruses in modern computer networks, partly because of the growing number of interested researchers with useful results to report, and partly because of the efforts of some members of the research community who have started to make communication between these divergent fields easier by providing common venues for publication and interaction.

This discussion leads us to ask some questions about the practicality of computer viruses for computation and the tradeoff between the potential benefits and the potential harm of using viruses for this purpose. In the remainder of this discussion, we will review some recent examples of practical computer viruses and some of the deep issues that come up in this research, and grapple just a bit with the issue of balancing the good with the bad.

Time's a Wastin

It turns out that in most of the world's computers, there is more unused processing time than used processing time. (The vast majority of the time, most computers are waiting for user input.) If a more reliable mechanism had been used in the early worm experiments, we might now have thousands of very reliable high speed parallel processing applications using this vast amount of unused computer time for useful purposes. One application that has been moderately successful is the `Lenstra-Manasse Creature' (A. Lenstra and M. Manasse, ``Factoring by Electronic Mail'' Eurocrypt, '89, appearing in ``Advances in Cryptology'', pp355-371, Springer Verlag, 1989.) which uses networked computers to factor large numbers.

This application runs under the Unix operating system and is distributed on major international research networks. It uses automated computer mail system to get permission from computer owners to cooperate in distributed processing, thus avoiding the problem of undesirable replication. Each `segment' of the Creature reports partial results to a central location via electronic mail, and the central location then distributes new problems, again using electronic mail for communication.

There are other applications of similar technologies in place, but I am unaware of any major papers in this area other than those in the Artificial Life conference cited earlier and those applications developed in my research which I shall now describe.

The Viral Bill Collector

One way to think of viral computation is in terms of an ecosystem. In the case of the viral bill collector, we have a set of `gene pools', each containing bill collectors of different sorts, and each available on their local machine for the collection of bills. When a bill is to be collected, a selection is made at (psuedo-)random from the `gene pool' on the current machine, that bill collector is `evolved' to contain the information relevant to the bill being collected, and the job of collecting

that bill is now that of the new bill collector.

As the bill is collected, the bill collector responsible for that bill `evolves' to contain updated information on the status of the case, schedules itself for awakening at various times related to deadlines in the collection process, and responds to actions related to that case via collector specific wake-up calls. After the collection process is complete, the bill collector reproduces in a quantity related to the success of the collection process, and the children (if any) are placed back in the `gene pool' and are thus made available for future collections.

Since replication is correlated to the success of the collector, the gene pool evolves over time to contain a higher portion of more effective bill collectors. Thus we have a form of selective survival which tends to improve system performance over time, assuming the gene pool has enough variation to cover a substantial portion of the space.

By contrast, a typical high volume collection system consists of a database management system with techniques for scanning periodically to determine which cases require action and performing collections on all applicable cases. For distribution among many computers, complex methods of database concurrency must be used. If a computer fails, special fault tolerant computing techniques must be applied. Timing between computers must be considered, and a scheme for simultaneous access must be devised. A wide variety of other issues come up, and thus, all of the non-viral bill collection systems on the market today which exploit a multitude of computers, use a centralized database and treat the remaining computers as terminals with some special data entry capabilities.

With the viral programming approach, we take another tactic altogether. Instead of creating a large centralized bureaucracy which controls and directs all activities, we distribute all functions to the individual bill collectors. Each bill collector only has information related to its own collection case and the ability to selectively call upon its various scenarios for bill collection. Instead of scanning a database for bills to be collected, each bill collector schedules a `wake up' call for when they next time it knows it has to do something. If some outside activity like a payment or a response to a previous action takes place, the human operators `wake up' the appropriate bill collector by sending it the new information. The collector then reacts to the situation by `evolving' its state of mind and line of pursuit to meet the new situation, reschedules its pending wake up calls, and goes back to sleep.

To collect statistics and resolve the status of cases in a centralized bill collection system, we normally implement a database scanning system using one computer for the analysis. In a viral system, the situation is much different because the data is distributed among all of the bill collectors. In order to do global searches or updates, we awaken all of the bill collectors and tell them what we are looking for or modifying, and they respond to the query or change request on their own.

There are clearly some tradeoffs between the viral bill collector and the centralized system. One of them is that the viral collector exploits the processing capabilities of a multitude of computers, while the centralized system exploits the communications advantage of a single system. The performance and costs associated with communication and computation dictate which is more cost effective. Another tradeoff is programming time. It turns out that it is very simple to write a viral bill collector, while writing a classical distributed database system is so difficult that it is rarely done. Another major advantage of the viral technique is that since we are running many very small programs instead of one large program, we can more easily distribute the computing load over a multitude of machines.

Our first viral bill collector was implemented by one programmer in one week, in 1986. It has evolved successfully in response to new needs without any global changes since that time. Although this sort of evolution requires human design, the amount of work is minimal. The bill collection viruses also coexist in the environment with a set of `maintenance' viruses that periodically awaken to perform cleanup tasks associated with systems maintenance.

Maintenance Viruses and the Birth/Death Process

Maintenance viruses, as a class, seem to be one of the most useful forms of computer viruses in existence today. Put in the simplest terms, computer systems are imperfect, and these imperfections often leave residual side effects, such as undeleted temporary files, programs that never stop processing, and incorrectly set protection bits. As more and more of these things happen over time, systems become less and less usable, until finally, a human being has to repair the problems in order to continue processing.

In the case of the viral bill collector, the design of the system is such that temporary files are stored under identifiable names, processing for each virus tends to be relatively short, and protection bits are consistently set to known values. To reduce manual systems administration, we decided to implement viruses that replicate themselves in limited numbers, seek out known imperfections, and repair them. Over time, we reduced systems administration to the point where the viral bill collector operated for over two years without any systems administration other than adding and removing users. The maintenance viruses were so successful that they even removed side effects of failed maintenance viruses.

In this case, it is convenient to look at the system as an ecosystem rather than as a computer system. Our maintenance ecosystem is powered in part by the `garbage' generated by the system, just as plants are powered in part by the CO$_2$ expelled by animals. The viruses that find an abundance of food replicate in larger numbers, while other viruses that find too little food die off. Those that replicate more are more abundant in the gene pool, and tend to be born more often.

Maintenance viruses have been designed for deleting old temporary files, repairing improperly set file protections, killing errant processes, awakening viruses that have slept too long, clearing temporary space, and checking file structures for errors. Like all programs, viruses can fail, but in the ecosystem formed by the maintenance viruses, when one virus fails, other viruses eventually consume it. For example, if a virus looking for errant protection settings gets into an unusual state, it may never terminate, but an errant process killing virus will eventually consume it. Thus we get a food chain wherein viruses eat other viruses that survive by eating other sorts of food.

To assure the continued survival of the maintenance viruses, they are born with a particular probability every time a user awakens a bill collector, and to assure they don't dominate processing by unbounded growth, they have limited life spans.

These ``birth/death'' processes are central to the problem of designing viruses that don't run amok, as well as to the evolution of viral systems over time. If it weren't for the death of old bill collectors and maintenance viruses, the system would eternally be collecting bills and performing maintenance under old designs, and the number of bill collectors and maintenance viruses would grow without bound. A global modification of all of the existing bill collectors would be required to make a system change, and this might be very hard to accomplish in a complex network. Birth and death processes are vital to optimization in viral systems where the environment changes dramatically with time, since what is optimal today may not even survive tomorrow.

This may even be used to explain why biological organisms that live in relatively dynamic environments (e.g. people on land) have limited life spans, while organisms that live in static environments (e.g. coral in an ocean) can live indefinitely without noticeable aging. There must be enough births to survive a reasonable number of deaths, death is necessary to prevent a population from expanding to a size where it consumes too many resources and dies from starvation. In a dramatically changing environment, the threats to survival change with time, and without active evolution, stagnant strains may eventually meet unsurvivable threats. The more dramatic the rate of environmental change, the more dramatic the evolution must be in order to survive, but we must be careful to understand that what seems to be dramatic change to one species may go completely unnoticed to another. For example, a 10 degree change in average local temperature has almost no

impact on people, and people regularly go through these changes when moving from city to city, but the same change in temperature dramatically impacts the plants that can survive, and many plants simply die out when moved to different climates.

Toward Random Variation and Selective Survival

We have spoken of evolution, but to many, this concept doesn't seem to apply to computer programs in the same way it applies to biological systems. In its simplest form, we speak of systems evolving through human reprogramming, and indeed, the term evolution seems to accurately describe the process of change a system goes through in its life cycle, but this is only one way that programs can evolve.

Consider a collection virus that uses pseudo-random variables to slowly change the weighting of different collection strategies from generation to generation, and replicates with a probability associated with its profitability (the net fee collected after all expenses of collection). In this case, assuming that the parameters being varied relate to the success of the collection process in an appropriate manner, the `species' of available viruses for the collection process will seem to `evolve' toward a more profitable set of bill collectors.

If we use less variation on bill collectors that are more successful, we may tend toward local optima. To attain global optima, we may occasionally require enough randomness to shake loose from the local optima. Over time, we may find several local optima, each with a fairly stable local population of bill collectors. Thus different species of bill collectors may coexist in the environment if there are adequate local niches for their survival. Cross breeding of species is feasible by taking selected parameters from different species to birth new bill collectors. Some will thrive, while some will not even survive. As the external environment changes, different species may perform better, and the balance of life will ebb and shift in response to the survival rates. This evolutionary process is commonly called ``random variation and selective survival", and is roughly the equivalent of biological evolution as we now commonly speak of it.

Complex Behaviors, Generating Sets, and Communication

The behavior we are discussing is getting complex, involving local and global optimization, evolution over time, and even the coexistence of species in an environment; and yet the computer programs we are discussing are still quite simple. Our viral bill collectors consist of only a few pages of program code, and yet they perform the same tasks carried out by much larger programs. The inclusion of evolution in experimental systems has been accomplished in only a few lines of program code. We create a small ``generating set" of instructions which creates a complex system over time through birth/death processes, random variation and selective survival, and the interaction of coexisting species in the environment. Even quite simple generating sets can result in very complex systems. In fact, in most cases, we cannot even predict the general form of the resulting system.

Our inability to accurately predict systemic behavior in complex systems stems, in general, from the fact that it is impossible to derive a solution to the `halting problem', which is one of the fundamental unsolvable problems of computer science. The problem of determining whether a computer program will ever complete processing a problem was proven `undecidable' for general purpose computers by Turing in 1936. (A. Turing, ``On Computable Numbers, with an Application to the Entscheidungsproblem", London Math Soc Ser 2, 1936.) Through a mathematical proof technique called reduction, most of the `interesting' behaviors of complex systems can be proven undecidable as well. More specifically, it was proven by Cohen (see earlier 1985 reference) that a virus can evolve in as general a fashion as a computer can compute, and therefore that the result of viral evolution is potentially as complex as Turing's computation.

It seems there is little we can do about predicting the behavior of general purpose evolutionary systems, but just as there are large classes of computer programs with predictable behavior, there are large classes of evolutionary systems with predictable behavior. Indeed, in the same way as we can generate computer programs from specifications, we can generate evolutionary systems from specifications, and guarantee that they will act within predefined boundaries. In the case of the maintenance virus, we can even get enhanced system reliability with viral techniques. Unfortunately, we haven't yet developed our mathematical understanding of viruses in an environment to make good predictive models of these sorts of systems, but there is a general belief that many important problems are not intractable at the systemic level, and if that is true, we may be able to make good predictive models of the behavior of viral system.

One of the ways we can design predictable viral systems is by adding communications. Completely deaf and dumb viruses have a hard time surviving because they tend to be born and die without any controlling influences, and we get unstable situations which either consume too many resources and ruin the ecology or die from lack of sufficient biomass. With even rudimentary communications, viruses can survive far better. For example, most real-world computer viruses survive far better if they only infect programs that are not yet infected. Too much communication also makes viruses inefficient, because they have to address an increasingly global amount of information. We suspect that communications is beneficial to viral survival only to the extent that it helps to form stable population relative to the resources in the environment, but in terms of predictability, communications seems to be key.

The maintenance viruses described earlier provide a good example of viral communication. There is no direct communication between the maintenance viruses, but they end up communicating in the sense that what each virus does to the environment alters the actions of other viruses. For example, a maintenance virus that deletes temporary files that haven't been accessed in 24 hours or more will not delete any files that a previous maintenance virus has already deleted, since the earlier virus already consumed them. Since the latter virus acts differently based on the actions of the earlier virus, there is a rudimentary form of communication between the viruses via changes in the environment. In fact, humans communicate in much the same way; by making changes in the environment (e.g. sound waves) that affect other humans. The net effect is that maintenance viruses act more efficiently because they rarely interfere with each other.

Toward Widespread Viral Computation

The possibilities for practical viruses are unbounded, but they are only starting to be explored. Unfortunately, viruses have gotten a bad name, partly because there are so many malicious and unauthorized viruses operating in the world. As of early 1991, there were over 500 real-world viruses, with a newly designed unauthorized virus being detected in the global computing environment more than once per day. If the computing community doesn't act to counter these intrusions soon, society may restrict research in this area and delay or destroy any chance we have at exploiting the benefits of this new technology. There are now many useful tools for defending against malicious viruses and other integrity corruptions in computer systems, and they can often be implemented without undue restriction to normal user activity, but perhaps another tactic would also serve society well.

The tactic is simple; instead of writing malicious viruses, damaging other people's computer systems, hiding their identity, and risking arrest, prosecution, and punishment; virus writers could be provided with a legitimate venue for expressing their intellectual interest, and can get both positive recognition and financial rewards for their efforts. By changing the system of rewards and punishment, we may dramatically improve the global virus situation and simultaneously harness the creative efforts of virus writers for useful applications. One instance of such a tactic is the `Computer Virus Contest'. The Computer Virus Contest gives an annual cash prize for the most useful computer virus submitted. The contest rules prohibit the use of viruses that have been

released into uncontrolled environments, viruses placed in systems without explicit permission of the owner, and viruses without practical mechanisms to control their spread.

The emerging computer virus technology, like all new technology, is a two edged sword. Just as biological viruses can cause disease in humans, computer viruses can cause disease in computer systems, but in the same sense, the benefits of biological research on the quality of life is indisputable, and the benefits of computer virus research may same day pay off in the quality of our information systems, and by extension, our well being. To the extent that we can learn about our ecosystem systems by studying the informational ecosystems formed by computer viruses, we may save ourselves from the sorts of mistakes we have already made in dealing with our environment.

Somebody once said that computer systems are one of the greatest laboratory facilities we have. Their ability to model and mimic life and life-like situations is astounding both in its accuracy and in its ability to allow experiments that would be unconscionable or infeasible in any other laboratory. We have the unique opportunity to use this laboratory to get at the fundamental nature of living systems, if we can only get past our biases and get on with the important work awaiting us.

Summary, Conclusions, and Further Work

There is clearly a tremendous potential for both harm and good in the ongoing pursuit of almost any research area, and as scientists we have a duty to consider the moral implications of our work. In the case of computer viruses, we have not adequately considered the potential for beneficial uses. Clearly, we can have very useful and well controlled viral computing environments, and clearly these environments have close ties to the living ecosystem that supports our very existence. Perhaps we will even learn something about ourselves and our environment through this effort, and perhaps we will not, but I don't think we can afford to ignore the implications.