Genetically Induced Communication Network Fault Tolerance

Stephen F. Bush and Amit B. Kulkarni

(Invited Paper: SFI Workshop: Resilient and Adaptive Defence of Computing Networks 2002)

Abstract— **This paper presents the architecture and initial feasibility results of a proto-type communication network that utilizes genetic programming to evolve services and protocols as part of network operation. The network evolves responses to environmental conditions in a manner that could not be pre-programmed within legacy network nodes *a priori*. *A priori* in this case means before network operation has begun. Genetic material is exchanged, loaded, and run dynamically within an active network. The transfer and execution of code in support of the evolution of network protocols and services would not be possible without the active network environment. Rapid generation of network service code occurs via a genetic programming paradigm. Complexity and Algorithmic Information Theory play a key role in understanding and guiding code evolution within the network.**

Index Terms— **Active Networks, Algorithmic Information Theory, Kolmogorov Complexity, Complexity Theory, Genetic Programming, Self-Healing Networks.**

I. Introduction

AC tive networking is a novel approach to network architecture in which network nodes – switches, routers, hubs, bridges, gateways etc. – perform customized computation on packets flowing through them. The network is called an "active network" because new computations are injected into the nodes dynamically, thereby altering the behavior of the network. Packets in an active network can carry fragments of program code in addition to data. Customized computation is embedded within the packets code, which is executed on the intermediate network nodes.

Many active network components and services have been designed, implemented, and are undergoing experimentation. The ABone (Active Network Backbone) implements a relatively large-scale (given the novelity of the technology) active network ($O(100)$ nodes). However, the fundamental science required to understand and take full advantage of active networking is lagging behind the ability to engineer and build such networks. In fact, the current Internet, whose protocols were built upon the ill-defined goal of simplicity are only slowly being understood. An outcry from the Internet community, with its carefully crafted, static protocol processing, with massive documentation ($O(4000)$ Request for Comments) of passive (non-executable) packets is that it is already "too" complex.

An adaptive fault tolerant system, no matter how resilient, would unlikely receive acceptance by industry or the community if it were considered "complex" in the colloquial sense. How can such systems, which require complexity to be adaptive, at the same time appear simple to understand and manage. Are active networks really more complex than the current Internet? Are adaptive applications built upon active networks any more or less complex than the same applications built upon the legacy Internet? Does a measure of complexity exist that would allow an objective comparison to be made? What are the benefits of an active network with respect to passive networks? While these are extremely difficult questions to answer, this paper attempts to lay the groundwork for answering these questions by proposing a complexity measure, Kolmogorov Complexity, and proposing an adaptation mechanism, Genetic Programming, based upon an analogy with biological systems.

Kolmogorov Complexity was applied in [1] as a measure of potential algorithmic information content for use in prediction and control of an active network. In the remainder of this paper, the term complexity will be used to indicate a particular form of complexity known as Kolmogorov Complexity. Kolmogorov Complexity is a measure of the length of the smallest program, such that, when executed upon a Universal Turing Machine, it generates a particular string of bits x. The length of such a smallest program $K(x)$ is the complexity of the bit-string, x. It should be noted that research has been performed in the use of genetic programming to evolve the smallest program for a given bit-string, and thus estimate $K(x)$. Complexity was applied in [1] to optimize the combined use of communication and computation within an active network; to determine the optimal amount of code versus data. It was shown that if the Kolmogorov Complexity of the information related to the prediction of the future state of the network is estimated to be high, then the ability to develop code, representing the non-random, or algorithmic portion, of that information is low. This results in a low potential benefit for algorithmic coding of the information; the benefit of having code within an active packet would appear to be minimal in such cases. Conversely, if the complexity estimate is low, then there is great potential benefit in representing information in algorithmic form within an active packet. In [1] it was suggested that if the algorithmic portion of information changes often and impacts the operation of network devices then active networking provides the best framework for implementing solutions. This is precisely the case in genetically programmed network services, a new class of services that are not pre-defined but those that evolve themselves in the network in response to the state of the network. In this paper, we will restrict this class to those services that are programmatic solutions for perceived faults that occur in a network. Further research is required to generalize this class to

Stephen F. Bush and Amit B. Kulkarni are with General Electric Global Research, 1 Research Circle, Niskayuna, NY 12309. Email: bushsf@research.ge.com, URL: http://www.research.ge.com/~bushsf/ftn

include other types of network services.

Frameworks for protocol and service composition have been developed for active networks, one of which is well described in [2]. Thoughts on the requirements for protocol and service composition are also discussed in [3]. However, the work done to date is lacking in that it does not address how active code will be generated rapidly enough to make dynamic injection of the code a significant factor. The argument against active and programmable networks is that, given enough time, memory, and processing power, legacy systems could eventually contain all the functionality that active networks could have injected. To do this, legacy developers would have to know *a priori* all possible functionality that would be required in the network. However, this paper demonstrates that it is possible for the network to generate code rapidly and in a manner that can never be known *a priori* for every possible condition. The inspiration for a genetic algorithm based approach to solution composition comes from nature in the form of the docking problem in molecular biology [4], [5], [6]. Solutions that efficiently match a particular fault should be able to "dock" with the fault. Prediction for successful docking in biology can be attempted by searching for minimal energy or minimal geometric construction combinations. Here we consider a genetic algorithm used to generate a solution for the self-composition of solutions to mitigate network faults. One goal of the experiment discussed later in this paper is to study the relationship between complexity and solution composition. In particular, it has been hypothesized that the complexity of the fault and potential solution will decrease as the optimal solution is composed. Specific examples of faults that could be simulated are:

- Network mis-configuration
- Bandwidth and Processor mis-allocation
- Faults caused by Distributed Denial of Service and virus attacks
- Poor Traffic shaping
- Routing problems
- Non-optimal fused data within the network
- Poor link quality in wireless and mobile environments
- Mal-composed protocol framework models in the network
- Poorly tuned components of network services

A simple fault, namely, mis-allocation of bandwidth and processing capability resulting in packet jitter, has been chosen as a working example. A fitness function defines a metric for "goodness" of a population. In this case, "goodness" is the reduction in the variance of packet arrival times. The fault is represented by the difference between the actual system and a minimum required fitness. Genetic material will evolve to minimize the effect of the fault. The complexity of the combined fault-solution pair should be at a minimum when the fitness is optimal. We will borrow a term from molecular biology and call a perfectly matched fault and solution a successful "docking".

II. Complexity and Evolutionary Control

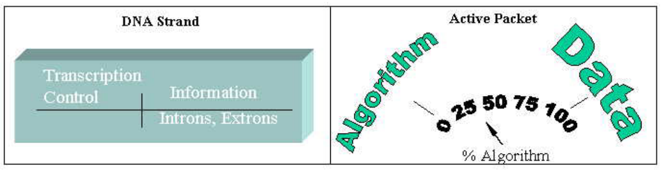
Complexity and evolution are intimately linked. Kolmogorov Complexity ($K(x)$) [7] is the optimal compression of string x. This incomputable, yet fundamental property of information has vast implications in a wide range of applications including system management and optimization [8], [9], security [10], [11], and Bioinformatics. Active networks [12] form an ideal environment in which to study the effects of tradeoffs in algorithmic and static information representation because an active packet is concerned with the efficient transport of both code and data. As noted in Figure 1, there is a striking similarity between an active packet and DNA. Both carry information having algorithmic and non-algorithmic portions. The algorithmic portion of DNA has transcription control elements as well as the codons [13]. The active packet has control code and may contain data as well.

Kolmogorov Complexity and Genetic Programming have complementary roles. Genetic Programming has been used to estimate Kolmogorov Complexity [14], [15]. Genetic Programming benefits from Kolmogorov Complexity as a measure and means of controlling not only the complexity, but the size and generality of the result [16]. One of the most obvious uses for complexity in networking is Programmatic Compression [17]. In this paper, the foundation is developed for the use of complexity to enable the network to self-heal. In the next section, a description of the Minimum Description Length algorithm and its role in Active Networks is explained.

III. The Application of Complexity in a Communications Network

The goal of the system that has been implemented is to utilize the benefit of an active network to automatically generate solutions that bring the network back into line with a healthy model of the system. The fitness function is used to describe the desired outcome. The concept of molecular docking, mentioned previously, requires a more precise measurement of the degree of "fit" in the docking of a fault and solution. In this project, we are exploring the use of Kolmogorov Complexity, estimated via the Minimum Description Length algorithm, as the means to measure the fit between the fault and the desired state. The next paragraph describes the Minimum Description Length complexity estimator and its relationship to active networking.

- Nucleotide Bases: A, C, G, U
- Triplets (Codons) result in translation to Amino Acids within the Ribosome
- Chromosome Structure: List of connected unit pairs. In eucaroytes these reside in the nucleus: ((Delay, Delay)(Join, Split)…)



- DNA Strand and Active Packet operate in the same manner: both carry control and data
- Dualism in genetic world (gene as information and algorithmic code) first noted by von Neumann

Fig. 1. DNA and an Active Packet.

A question active network application developers must answer is: "How can I best leverage the capabilities that active networks have to offer?" Because the word "active" in active networks refers to the ability to dynamically move code and modify execution of components deep within the network, this typically leads to another question: "What is the optimal proportion of content for an active application that should be code versus data?" A method for obtaining the answer to this question comes from direct application of Minimum Description Length (MDL) [18] to an active packet. Let D_x be a binary string representing x. Let H_x be a hypothesis or model, in algorithmic form, that attempts to explain how x is formed. Later in this paper, we view H_x as a predictor of x in the analysis of Active Virtual Network Management Prediction. For now let us focus on developing a measure of the complexity of x. MDL states that the sum of the length of the shortest encoding of a hypothesis of two components will estimate the Kolmogorov Complexity. The two components are the length of a model generating string x and the length of the shortest encoding of x using the hypothesis. This can be represented mathematically as $K(x) = K(H_x) + K(D_x|H_x)$. Note that error in the hypothesis or model must be compensated within the encoding. A small hypothesis with a large amount of error does not yield the smallest encoding, nor does an excessively large hypothesis with little or with no error. A method for determining $K(x)$ can be viewed as separating randomness from non-randomness in x by "squeezing out" non-randomness, which is computable, and representing the non-randomness algorithmically. The random part of the string, that is, the part remaining after all pattern has been removed, represents pure randomness, unpredictability, or simply, error. Thus, the goal is to minimize $l(H_e) + l(D_x|H_e) + l(E)$ where $l(x)$ is the length of string x, H_e is the estimated hypothesis used to encode the string (D_x) and E is the error in the hypothesis. The more accurately the hypothesis describes string x and the shorter the hypothesis, the shorter the encoding of the string. Choosing an optimal proportion of code and data minimizes the packet length.

The proposed hypothesis is that the Kolmogorov Complexity of a combined fault and solution description is minimized when the optimal solution to mitigate the fault is composed. A nearly trivial example can be seen with reverse code. Assume that fault data, F exists. Assume that the fault does not erase data but merely transforms it. Define the algorithmic description of the fault data $P_F()$. The reverse code for $P_F()$ will be labeled $RP_F()$. Assume $P_F()$ and $RP_F()$are minimal length programs. Then, $RP_F(P_F()) = \phi$, where ϕ is the empty set. RF is the data generated by $RP_F()$. Since the fault does not erase any data, the process is reversible [7] and therefore, $K(RF) - K(F) = 0$. The equivalence in complexity of RF and F follows because there is no loss or gain of complexity when the system is restored to a prior state using the anti-fault process RP_F; there is no work performed. The algorithmically reversed fault will be referred to as an anti-fault in this paper.

The descriptive complexity of the fault and the solution should ultimately be as low as possible and the Minimum Descriptive Length algorithm can be used, among other complexity estimators, as a technique to guide solution composition. In fact, this is the case with reversible code. Complexity is important information because it is an indicator of both the type of fault and level of difficulty in correcting the fault and the severity of the fault; fault severity is important in triage operations to optimize system health. Second, a more compact algorithmic representation of a fault will travel faster and more rapidly through the network; it is an efficient format for alerting system management and in triggering automated solutions. Third, it can be relatively easy to reverse the code of an algorithm, possibly generating an anti-fault, or solution to a problem in certain cases. Reversible code has been presented in previous work as a mechanism for generating anti-messages in Time Warp simulation [19].

Fault tolerant and self-healing systems should have the ability to self-compose solutions to faults. Ideally, composition should be an inherent part of system operation, rather than a structure imposed from "outside" the system. Genetic Algorithms are on the path towards self-composing solutions, however genetic algorithms, as implemented today, require external control to manipulate the genetic material. In other words, the genetic algorithm itself must be programmed into the system; if the genetic algorithm code failed, then the self-healing capability would fail. While this situation is not ideal, it is explored as a possible step towards a truly self-healing system.

One of the contributions of this paper is the study of complexity in genetic algorithms with the goal of eventually designing self-composing solutions. Genetic algorithms are widely known for their ability to find optimal solutions, avoiding local extrema, by using evolutionary-like processes dependent upon "random" mutation. Kolmogorov Complexity describes the randomness of information. The Kolmogorov Complexity of the genetic material during the evolution of a genetic algorithm can be estimated and yields interesting clues about the underlying physics of the information during its evolution towards a fitness function. It is our hypothesis that, as the evolution proceeds and the fitness level of the genetic material rises, the complexity decreases. This result yields an interesting insight that supports the hypothesis that "solutions" that self-compose to mitigate a fault will tend to decrease in complexity.

IV. The Genetic Algorithm

The goal of this study is to examine how complexity, specifically an estimate of Kolmogorov Complexity, relates to the evolution of a self-composing solution. We consider a genetic algorithm to be an approximation of a self-composing system. Details on the operation of genetic algorithms can be found in [20], [21], [22]. This paper assumes a basic understanding of genetic algorithm operation and provides only a brief overview. In this experiment a pre-existing Mathematica genetic algorithm package[1] is used. The decision to use Mathematica was based upon its combination of symbolic and arithmetic capabilities and because many of our research utilities, including Kolmogorov estimation functions are implemented in Mathematica.

The genetic algorithm package assumes a population of binary strings of preset size and whose values, when converted to a float type, are between zero and one. Similarly, the fitness function is assumed to accept and return values in the range from zero to one. Fitness values closer to one are assumed

[1]Written by Mats G. Bengtsson National Defense Research Establishment Box 1165, S-581 11 Linkoping Sweden email: matben@lin.foa.se

indicate more highly optimized results. A genetic algorithm consists essentially of three parts: selection, crossover, and mutation. In selection, each string is selected with a probability proportional to its fitness value. In crossover, a pair of selected strings is determined, a position along the string is chosen at random, and the right and left parts of each string are swapped. In mutation, each gene is changed at random with a low probability, in this case a probability of 0.002 was chosen based upon repeated experimentation. Each individual is coded as a binary string of length 10 bits. This length provides the size necessary to achieve numerical precision while being small enough to allow a large population size and without excessive overhead. The problem is limited to one-dimension with value x, which represents the real value of the bits in string x, that varies from zero to one. The first step is to create a random population. The population is defined on the real axis from zero to one. The random values are represented in the form of binary strings. Next a fitness function is defined. It is defined in the interval zero to one. The fitness function in this example is defined as $f(x) = \sin(\pi x)$. Thus, binary representations of values that are odd multiples of 0.5 will have maximal fitness.

A. Kolmogorov Complexity

This section discusses a general approach for self-composing solutions using lessons learned from the previous section. The approach can be described as the automated generation of a solution hypothesis $H_s = R(H_e - H_f)$, that is, the reverse of the algorithmic difference between the faulty and correct algorithmic representation of behavior by controlled means. As H_f deviates from H_e, complexity or heat as presented here, is generated. In [8] the relationship between fault and energy is explored and simulated (see [11], [10], [23] for recent work on complexity and energy and Information Assurance). The motivation for that experiment came from the relationship between Kolmogorov Complexity and entropy. The definition and application of Kolmogorov Complexity to vulnerability analysis identified how Kolmogorov Complexity can be used to determine vulnerabilities in a system as areas of low complexity. An underlying hypothesis of our work is that computation and communication are fundamentally related through complexity theory, and, thus, bandwidth and processing utilized in denial of service are fundamentally interrelated. Low complexity data or code consuming large amounts of bandwidth or processing indicates the likelihood of an attack. A model of complexity evolution within a closed system is described in reference [10]. That reference developed an abstract model with which to study complexity, specifically Kolmogorov Complexity, of information within an information system. That model explores $K(x)$, a measurement of length in bytes, and $K(x)/s$, a measure of the maximum increase in complexity of the system due to code entering a system such as code carried by active packets. The rate of complexity increase in terms of algorithmic active packet complexity in units of $K(x)/s$ within the closed system was measured. Significant changes in system complexity indicate the presence of faults. Reference [24] reported the results of Kolmogorov Complexity probes that detect Distributed Denial of Service attacks.

An active network environment is used to emphasize that information assurance laws must be able to deal with many alternative and dynamically changing representations of information. With regard to active packets and information theory, passive data is simple Shannon compressed data, and active packets are a combination of data and program code whose efficiency can be estimated by means of Kolmogorov Complexity [25]. The active network Kolmogorov Complexity estimator is currently implemented with a variety of compression estimators ranging from simple empirical entropy to more complex algorithms beyond the scope of this conference. The probe returns an estimate of the smallest compressed size of a string. The simplest estimator, trading accuracy for speed and low overhead, is based upon computing the entropy of the weight of ones in a string. Specifically it is defined in Equation 1 where $x\#1$ is the number of 1 bits and $x\#0$ is the number of 0 bits in the string whose complexity is to be determined. Entropy is defined in Equation 2. See [25] for other measures of empirical entropy and their relationship to Kolmogorov Complexity. The expected complexity is asymptotically related to entropy as shown in Equation 3. Observe an input sequence at the bit-level and concatenate with an output sequence at the bit-level. This input/output concatenation is observed for either the entire system or for components of the system. Low complexity input/output observations quantify the ease of understanding by a potential attacker. Previous work has demonstrated the use of Kolmogorov Complexity for Distributed Denial of Service (DDoS) attack detection [24].

$$\hat{K}(x) = l(x)H\left(\frac{x\#1}{x\#1 + x\#0}\right) + \log_2(l(x)) \tag{1}$$

$$H(p) = -p\log_2 p - (1.0 - p)\log_2(1.0 - p) \tag{2}$$

$$H(X) = \sum_{l(x)=n} P(X = x)K(x) \tag{3}$$

Because Kolmogorov Complexity was originally derived for the study of randomness, it is interesting to note that randomness plays a significant role in the operation of the genetic algorithm itself. The initial genetic material should be generated randomly. Selection of genes for mutation and crossover points should also be done randomly. Finally, selection of gene pairs is done randomly, but in proportion to their fitness value.

Given the randomly generated nature of the initial genetic material, one would expect the complexity of the genetic material to decrease as the genetic algorithm evolves. This is clearly the case in the initial steep downward spike shown in Figure 2. As the algorithm continues to evolve and the fitness of the genetic material improves, one would expect structure and order to appear. As mentioned earlier, in this specific case, the algorithm encourages the growth of binary strings that represent odd multiples of 0.5.

Figure 3 shows the complexity, estimated as the compressed size, of the genetic material as a function of evolutionary steps. Compare with Figure 4, which shows the sum of the fitness values as a function of evolutionary steps. The complexity decreases as the cumulative fitness function increases, then rises again while evolution continues however, the fitness function

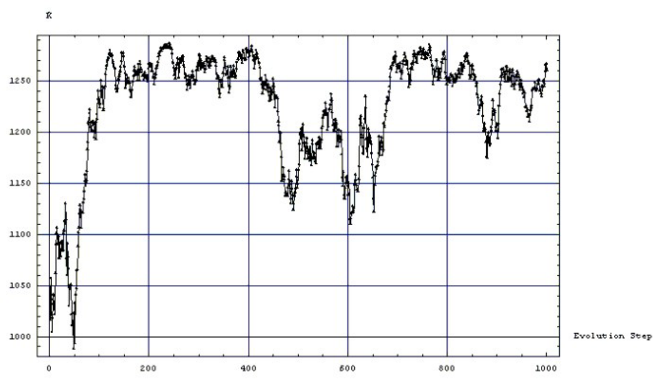


Fig. 2. Complexity of Genetic versus Evolutionary Time Steps with Population 128.

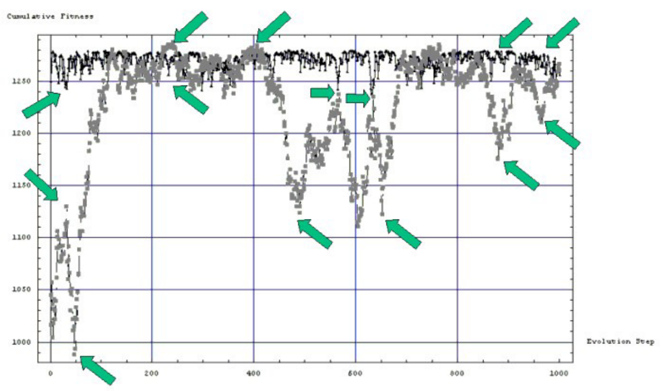


Fig. 4. Complexity and Fitness Comparison.

does not significantly increase. The complexity measure seems to indicate that the first optimal genetic composition was found near evolution step 50. As the genetic algorithm continued beyond that point, the genetic material became more complex again with no corresponding benefit in fitness. This result was unanticipated, but is plausible as new solutions evolve, with varying complexity, attempting to maximize fitness.

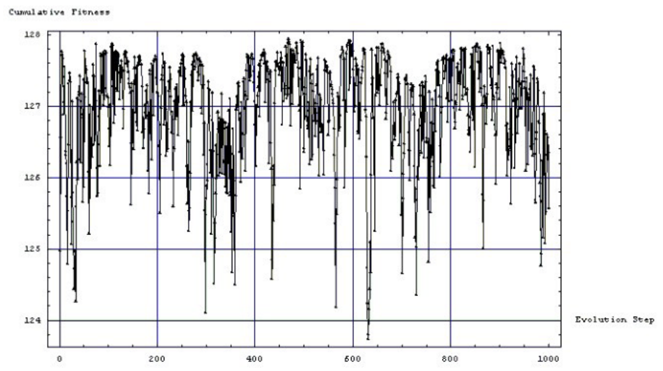


Fig. 3. Cumulative Fitness Function of Genetic Material with Population 128.

The cumulative fitness function results (multiplied by 10 to shift upward for easier comparison with the estimated complexity) are shown in Figure 4. Note that the points of high complexity always coincide with points of low cumulative fitness. Points of relatively low complexity correspond to high cumulative fitness. Arrows point to the extrema in the cumulative fitness function and estimated complexity that can be seen to align with extrema in the fitness function. In particular, minima in estimated complexity occur simultaneously with opposing maxima in the fitness. This indicates an inverse relationship between complexity and cumulative fitness extreme points.

Consider the complexity of the fitness function itself. The fitness function is an algorithmic representation of the fitness of a chromosome. The range resulting in maxima generated from the fitness function forms a string that represents the target complexity. In this particular genetic algorithm example, a solution of 0.5 for all 128 members of the population would yield an estimated complexity of 611.3. This low a level of complexity was never reached for two reasons: there are multiple optimal solutions, namely odd multiples of 0.5, and the algorithm never exactly achieved odd multiples of 0.5, but rather approximately close values. The remaining sections discuss how these concepts have been implemented to construct a fault tolerant network.

B. *Towards a Self-Evolving Network System*

Other papers from the Imperishable Networks Project have developed complexity-based techniques for fault detection and identification as discussed in [26] and [9]. The focus of this report is on progress towards self-composition of solutions assuming that other techniques, particularly complexity-based techniques, have identified faults. A problem with the genetic algorithm-based approach as previously described for use as a self-evolving system is that control is generally external to the genetic material and the genetic material is generally considered to be passive data. Instead the genetic material should be capable of being algorithmic information, that is, program code or objects. In addition, each chromosome, as an object, should contain the necessary capability to run the genetic algorithm. This would allow for a highly distributed and robust genetic algorithm capable of fault mitigation where the fault is represented through the fitness function.

A criticism of this approach might be that a genetically-engineered protocol stack will create a complex framework that will be difficult to understand and maintain. However, our approach is to compose the framework from simple components. Each of these components will be individually verifiable with respect to its properties and actions. As the components are arbitrarily composed to form a protocol stack, some protocol stacks may be generated that violate the principles of safety, consistency and correctness. One way to approach this is to define a fitness function that verifies the suitability of the stack with respect to the properties desired. Any mis-configured protocol stacks are automatically eliminated from consideration if the fitness function is carefully defined to check for the above-mentioned properties. However, this might make the definition of the fitness function itself cumbersome as every possible stack composition property will have to known a priori and an appropriate fitness "filter" defined. This will lead to a loss of elegance in the fitness function definition and consequently poor maintainability. A better approach would be to define syntactic and certain semantic composition properties in the individual components themselves, possibly in the form of logical expressions. These expressions will enforce constraints on the behavior of the components, which can be verified at run-time. The run-time system will embed a theorem-prover, which can be either a full-blown prover like PVS, NuPrl or SPIN or a reduced version of one, to systematically verify properties during composition itself. This reduces the burden on the programmer to define a proper fitness function that can catch and eliminate all types of composition errors.

C. Approach

Genetic material begins in a random state (M), and converges to the complexity of the optimal value produced by the fitness function. This enables true solution composition from a wide range of possible solutions. One problem with this approach is the time required evolving towards a feasible solution. Another problem is the fitness function itself has to be self-generated in some manner. Using Active Virtual Network Management Prediction [12], the fitness function exists in the form of H_e where H_e is the estimated correct operation hypothesis of the system as described in [1].

In summary, the experiment in this section has shown a relationship among fitness, complexity, and the evolution of genetic material. Complexity estimation probes have been embedded in the General Electric Global Research Center Active Network test-bed for use in security experimentation. The next section explains the framework developed to utilize the same complexity probes described in [24] to control the evolution of a genetic program within the active network. This makes the network highly resilient to faults by enabling the capability to adapt in a wide variety of ways.

V. Genetic Network Programming Architecture

The Magician Active Network [12] overlay network is used to test the feasibility of the genetically programmed network service concept. An active packet representing the nucleus (assuming network nodes are like eucaryotes- cells containing nuclei) is injected into all the ‘network nodes. The nucleus contains a population of chromosomes– strings of functional units. Operation of Genetic Network Programming begins with the injection of basic building blocks, known as functional units, into the network as shown in Figure 5. Currently, this “genetic material” is flooded into each active node. However, the material will remain inactive in each active node until a fitness function is injected into the network. Receipt of a fitness function will cause evolution to proceed.

Functional units are very small pieces of code blocks that perform simple, well-defined operations upon an active packet. Examples of functional units are *Delay*, *Split*, *Join*, *Clone*, and *Forward*. There is also a *Null* functional unit whose use is explained later. Chromosomes are strings of functional units as shown in Figure 6. Once a chromosome is assembled, the codons can be translated into Amino Acids at the Ribosomes. In other words, the string of functional units will operate upon active packets from other applications (or other functional units) that traverse through the node. The chromosome is represented

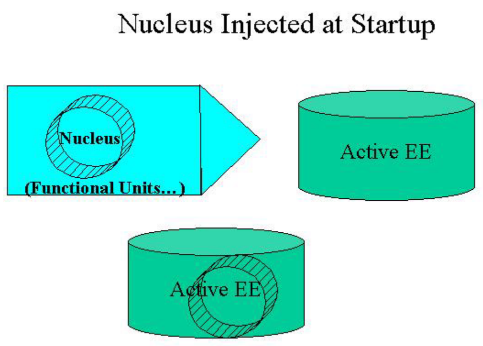


Fig. 5. Injection of the Nucleus.

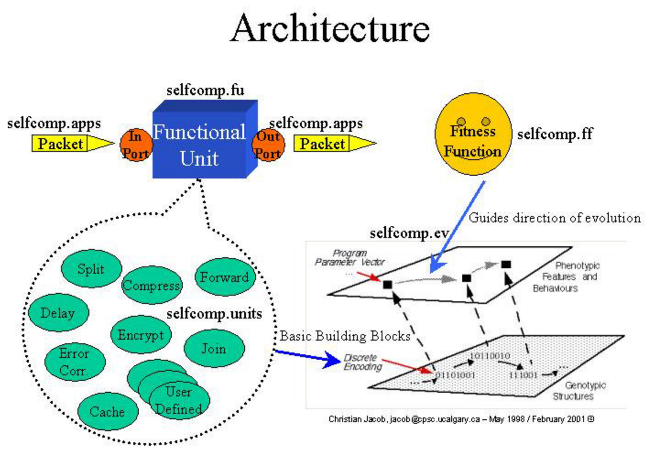


Fig. 6. Functional Units, Evolution, and Fitness.

in the code in a form similar to a Lisp symbolic expression, for example: *((Null Join Split) (Delay Split Join Delay))*.

Mutation and recombination occur among a population of genes. Mutation is a probabilistic change of a functional unit to another functional unit. Recombination is the exchange of chromosome sections from two different chromosomes. In Figure 7, a close-up of a single node can be seen containing a very short chromosome strand.

A single incoming traffic stream, as shown in Figure 8 entering the center node, is split into multiple streams. Each stream is processed by a different chromosome. Note that currently in our implementation, the full traffic stream is split along each chromosome, however, it is hypothesized that traffic sampling could be used to reduce the overhead in creating the multiple streams.

As shown in Figure 9, fitness functions can be designed to measure quality at different layers of the traditional protocol stack. In this particular case, fitness measures are shown at the Transport, Network, and Link Layers. As a particular example, jitter control might have a fitness function that minimized per frame variance at the Link Layer. The Network Layer would

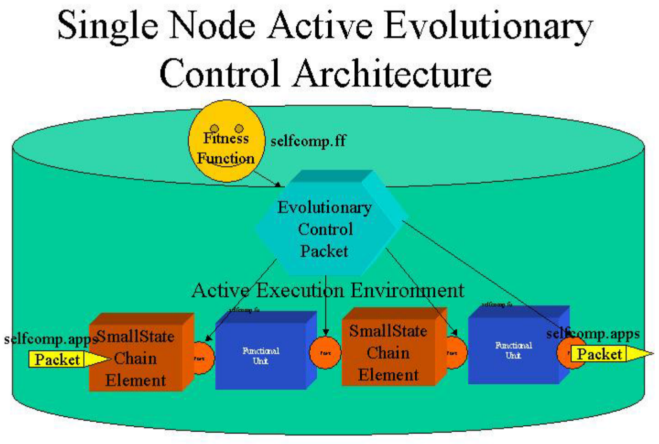


Fig. 7. Single Node Genetic Programming Architecture.

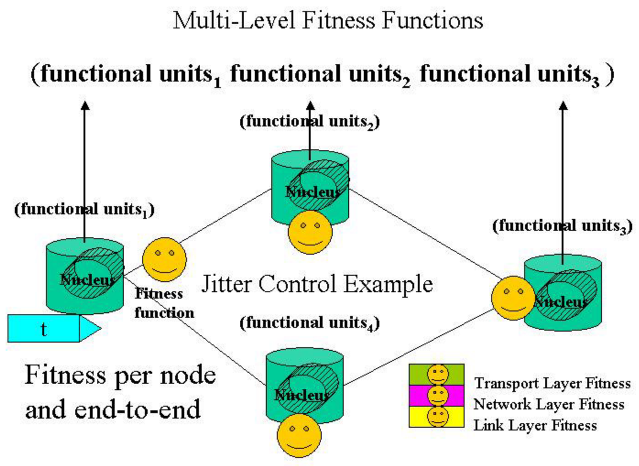


Fig. 9. Multiple Levels of Fitness.

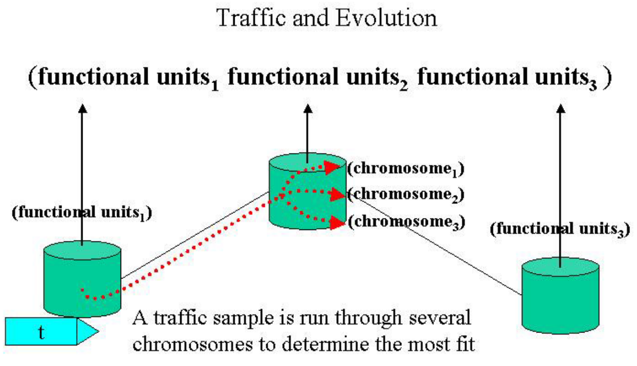

Traffic and Evolution
(functional units$_1$ functional units$_2$ functional units$_3$)
(chromosome$_1$)
(chromosome$_2$)
(chromosome$_3$)
(functional units$_1$)
(functional units$_3$)
t
A traffic sample is run through several chromosomes to determine the most fit

Fig. 8. Breeding Traffic Streams.

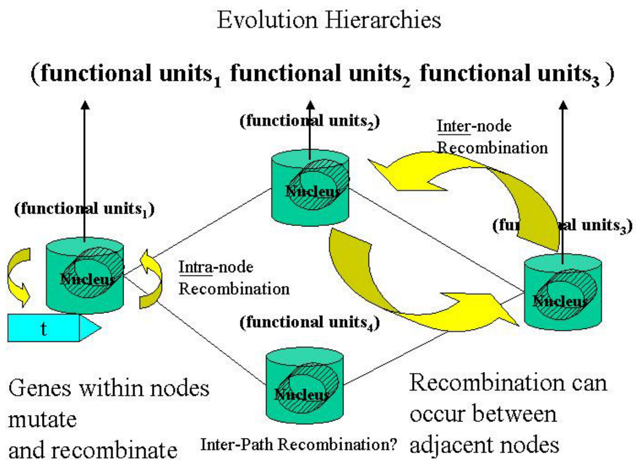

Evolution Hierarchies
(functional units$_1$ functional units$_2$ functional units$_3$)
(functional units$_2$)
Inter-node Recombination
(functional units$_1$)
(functional units$_3$)
Nucleus
Intra-node Recombination
(functional units$_4$)
t
Genes within nodes mutate and recombinate
Inter-Path Recombination?
Recombination can occur between adjacent nodes

Fig. 10. Recombination Levels.

attempt to maximize packet arrives at the destination in the reasonable time period, that is perform the routing function. The Transport Layer would have a fitness function that attempts to minimize end-to-end packet variance. The key is that each of these fitness functions need to work together towards reaching the stated goal in a reasonable manner. More will be said about the fitness function later.

In Figure 10, recombination can occur both within a node or between two nodes. In addition, as shown in Figure 11, changing the route of a packet also effectively accomplishes a recombination because the packet processing will be dependent upon the genetic material at each node traversed.

A key component of the evolutionary process is the fitness function. Fitness functions are "user" defined and injected into the network to control the evolution of the genetic population. For example, in our initial tests, minimizing variance in transmission time was used as a simple fitness function. However, initial experiments quickly demonstrated that the design of the fitness function is the most critical element. It reminds one of the saying, "Be careful of what you pray for..., because you might get it." Often the fitness was achieved, but in ways that

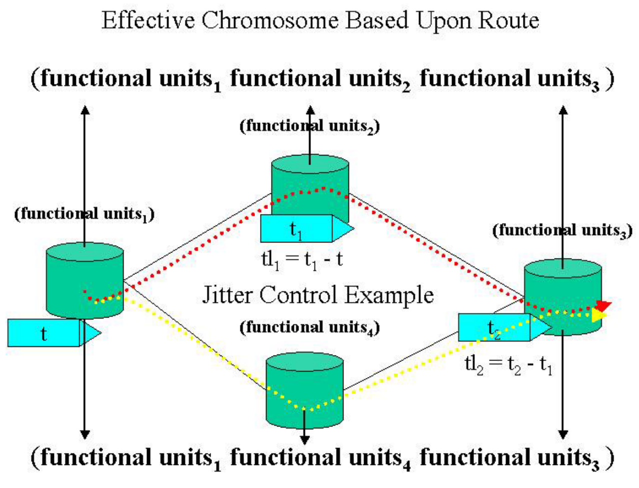

Effective Chromosome Based Upon Route
(functional units$_1$ functional units$_2$ functional units$_3$)
(functional units$_2$)
(functional units$_1$)
(functional units$_3$)
t_1
$tl_1 = t_1 - t$
Jitter Control Example
(functional units$_4$)
t
t_2
$tl_2 = t_2 - t_1$
(functional units$_1$ functional units$_4$ functional units$_3$)

Fig. 11. Chromosomes and Routing.

were unexpected and sometimes detrimental to the intended operation of the network. As a trivial example, the variance can be minimized by slowing the traffic to a near halt. Thus, a low latency term had to be added to the jitter control fitness function.

A. Genetically Programmed Active Network Jitter Control

As a feasibility test, an adaptive jitter control mechanism was developed on a fixed, wired active communication network having the topology shown in Figure 11. The genetic algorithm was implemented as an active application in the Magician Active Network Execution Environment [12]. Packets originate from the left-most node in Figure 11 and are destined for the right-most node in the figure. The dominant contributors to packet link transit time variability given the topology shown in Figure 11 are the fact that the active network is an overlay network that has unspecified lower-layer traffic and that packets are loaded and executed within a Java Virtual Machine residing in each node and are subject to Java garbage collection which runs at unspecified times.

The fitness function on all nodes returns a greater fitness as the result of a Simple Network Management Protocol query of an Object Identifier that measures packet link transfer time variance on the destination node is minimized. As previously mentioned, the fitness function is itself an active packet that consists of an objective function. The function is highly general and can be comprised of any mathematical function of accessible metrics.

Figures 12 through 14 show packet link transit variance through three of the chromosomes on the destination node and Figure 15 shows packet link transit variance without any jitter control mechanism at the destination node. Initial observation of the graphs shows that, overall, particularly as time progressed, the Chromosomes significantly reduced packet transit variance.

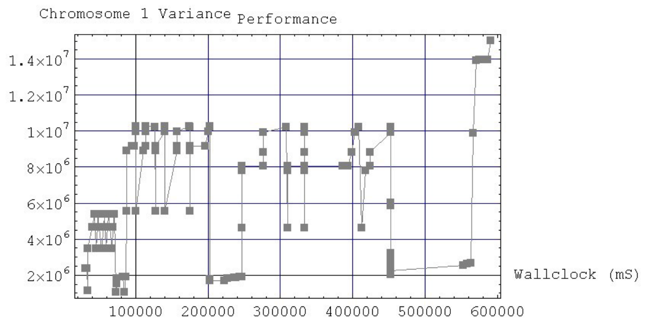


Fig. 12. Packet Link Transit Variance ($milliseconds^2$) on Destination Node Through Chromosome One.

Another observation of the experimental data is that the genetically programmed transit variance was initially worse than transit variance without any control mechanism. The reason for this is that the chromosomes begin operation with a random set of functional units and require time to converge to an optimal value.

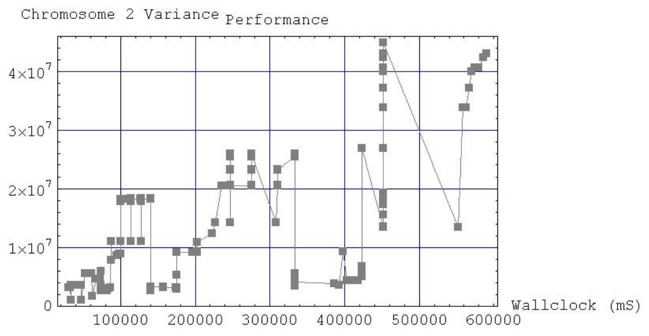


Fig. 13. Packet Link Transit Variance ($milliseconds^2$) on Destination Node Through Chromosome Two.

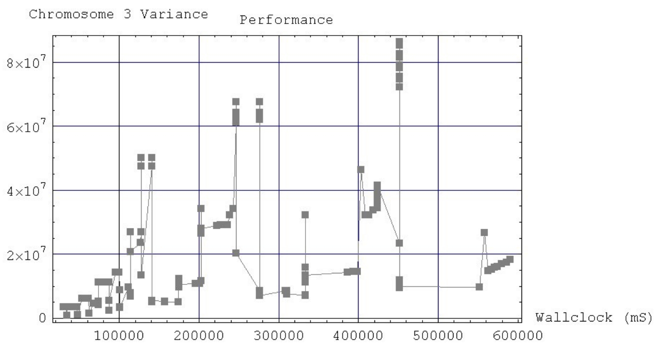


Fig. 14. Packet Link Transit Variance ($milliseconds^2$) on Destination Node Through Chromosome Three.

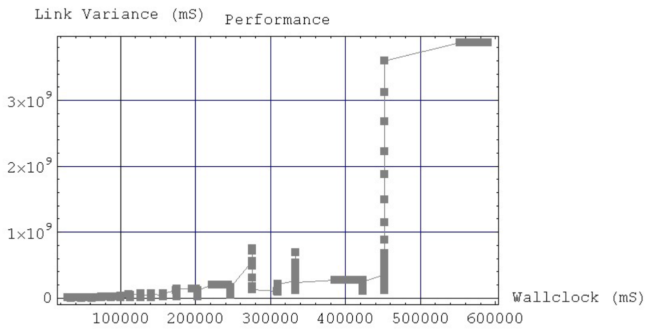


Fig. 15. Packet Link Transit Variance ($milliseconds^2$) on Destination Node Without Jitter Control.

VI. SUMMARY

This report has shown that a genetic algorithm shows sudden decreases in complexity of the population between generations as the algorithm evolves in response to the fitness function. Lower complexity correspond to greater homogeneity in the population and greater fitness to the chosen criterion. Thus it can be clearly seen that complexity can be used as one indicator of progress in evolution of the genetic algorithm. A framework for testing the injection of fitness functions into an active network that evolves solutions via a genetic programming technique has been implemented. Future work involves testing the response time to heal and the resiliency of the network in the presence of faults.

ACKNOWLEDGMENTS

The work discussed in this paper was funded in full by DARPA under the auspices of the Fault Tolerant Networks program. Our thanks go to Doug Maughan, the Active Networks program manager and Scott Shyne, Air Force Rome Labs for their generous support. We would also like to thank Nader Mehravari from Lockheed Martin Systems (Owego) for his support to this research.

APPENDIX

Jitter Control: A Simple Test Case

While *a priori* techniques have been developed for jitter control in legacy networks, jitter control forms a simple, easily measured and controlled application for the network genetic programming technique. The functional units injected into the network should allow evolution of a variety of interesting solutions to reduce variance, including adding delays, forward along different paths, or perhaps new ideas that have not been thought of yet.

REFERENCES

[1] Stephen F. Bush, “Active Virtual Network Management Prediction: Complexity as a Framework for Prediction, Optimization, and Assurance,” in *IEEE Computer Society Press, Proceedings of the 2002 DARPA Active Networks Conference and Exposition (DANCE 2002), San Francisco, CA*, May 2002, pp. 534–553, ISBN 0-7695-1564-9.

[2] S. Bhattacharjee, K. Calvert, Y Chae, S. Merugu, M. Sanders, and E. Zegura, “CANEs: An Execution Environment for Composable Services,” in *IEEE Computer Society Press, Proceedings of the 2002 DARPA Active Networks Conference and Exposition (DANCE 2002), San Francisco, CA*, May 2002, pp. 255–272.

[3] G. Minden, E. Komp, M. Kannan, S Subramaniam, S. Tan, S. Vallabhaneni, and J. Evans, “Composite Protocols for Innovative Active Services,” in *IEEE Computer Society Press, Proceedings of the 2002 DARPA Active Networks Conference and Exposition (DANCE 2002), San Francisco, CA*, May 2002, pp. 157–164.

[4] I. Kuntz, J. Blaney, S. Oatley, R. Langridge, and T. Ferrin, “A geometric approach to macromolecule-ligand interactions,” *Journal of Molecular Biology*, 1982.

[5] B. Shoichet, D. Bodian, and I. Kuntz, “Molecular docking using shape descriptors,” *Journal of Comp. Chemistry*, 1992.

[6] E. Meng, D. Gschwend, J. Blaney, and I. Kuntz, “Orientational sampling and rigid-body minimization in molecular docking,” *Proteins*, pp. 266–278, 1993.

[7] Ming Li and Paul Vitanyi, *Introduction to Kolmogorov Complexity and its Applications.*, Springer-Verlag, Aug. 1993.

[8] Amit B. Kulkarni and Stephen F. Bush, “Active network management, kolmogorov complexity, and streptichrons,” Tech. Rep. 2000CRD107, General Electric Corporate Research and Development, dec 2000, http://www.crd.ge.com/˜bushsf/ftn.

[9] Amit B. Kulkarni and Stephen F. Bush, “Active network management and kolmogorov complexity,” in *Proceedings of IEEE OpenArch 2001*, Apr. 2001.

[10] Stephen F. Bush and Scott C. Evans, “Kolmogorov complexity for information assurance,” Tech. Rep. 2001CRD148, General Electric Corporate Research and Development, 2001, http://www.crd.ge.com/˜bushsf/ftn.

[11] Scott C. Evans, Stephen F. Bush, and John E. Hershey, “Information assurance through kolmogorov complexity,” in *DARPA Information Survivability Conference and Exposition II (DISCEX-II 2001)*, June 2001, vol. II, pp. 322–331, http://www.crd.ge.com/˜bushsf/ftn.

[12] Stephen F. Bush and Amit B. Kulkarni, *Active Networks and Active Network Management: A Proactive Management Framework*, Kluwer Academic/Plenum Publishers, ISBN 0-306-46560-4, 2001, http://www.crd.ge.com/˜bushsf/ftn.

[13] James M. Bower and Hamid Bolouri, *Computational Modeling of Genetic and Biochemical Networks*, The MIT Press, 2001, 0-262-02481-0.

[14] M. Conte, G. Tautteur, I. De Falco, A. Della Cioppa, and E. Tarantino, “Genetic programming estimates of kolmogorov complexity,” in *Genetic Algorithms: Proceedings of the Seventh International Conference*, Thomas Back, Ed., Michigan State University, East Lansing, MI, USA, 19-23 1997, pp. 743–750, Morgan Kaufmann.

[15] I. De Falco, A. Iazzetta, E. Tarantino, A. Della Cioppa, and G. Trautteur, “A kolmogorov complexity-based genetic programming tool for string compression,” in *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2000)*, Darrell Whitley, David Goldberg, Erick Cantu-Paz, Lee Spector, Ian Parmee, and Hans-Georg Beyer, Eds., Las Vegas, Nevada, USA, 10-12 2000, pp. 427–434, Morgan Kaufmann.

[16] Byoung-Tak Zhang and Heinz Mühlenbein, “Balancing accuracy and parsimony in genetic programming,” *Evolutionary Computation*, vol. 3, no. 1, pp. 17–38, 1995.

[17] Peter Nordin and Wolfgang Banzhaf, “Programmatic compression of images and sound,” in *Genetic Programming 1996: Proceedings of the First Annual Conference*, John R. Koza, David E. Goldberg, David B. Fogel, and Rick L. Riolo, Eds., Stanford University, CA, USA, 28–31 1996, pp. 345–350, MIT Press.

[18] C. S. Wallace and D. L. Dowe, “Minimum message length and Kolmogorov complexity,” *The Computer Journal*, vol. 42, no. 4, pp. 270–283, 1999.

[19] C. Carothers, D. Bauer, and S. Pearce, “Ross: A high-performance, low memory, modular time warp system,” 2000.

[20] David E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley, 1989.

[21] David E. Goldberg, “Genetic and evolutionary algorithms come of age,” *Communications of the ACM*, pp. 113–119, 1994.

[22] M. Srinivas and Latit M. Patnaik, “Genetic algorithms: A survey,” *IEEE Computer*, pp. 17–26, 1994.

[23] Scott Evans and Bruce Barnett, “Network security through conservation of complexity,” in *MILCOM*, The Disneyland Resort, Anaheim, CA, USA, Oct 2002, IEEE.

[24] Amit B. Kulkarni, Stephen F. Bush, and Scott C. Evans, “Detecting distributed denial-of-service attacks using kolmogorov complexity metrics,” Tech. Rep. 2001CRD176, GE Global Research Center, Dec. 2001.

[25] Scott C. Evans and Stephen F. Bush, “Symbol compression ratio for string compression and estimation of kolmogorov complexity,” Tech. Rep. 2001CRD159, General Electric Corporate Research and Development, 2001, http://www.crd.ge.com/˜bushsf/ftn.

[26] Stephen F. Bush and Scott C. Evans, “Complexity-based information assurance,” Tech. Rep. 2001CRD084, General Electric Corporate Research and Development, Oct. 2001, http://www.crd.ge.com/˜bushsf/ftn.