

# Genetically Induced Communication Network Fault Tolerance

Stephen F. Bush, *Senior Member, IEEE*<sup>1</sup>

**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, and Self-Healing Networks.

## I. INTRODUCTION

ATROPOS, an in-line communication network prediction toolkit<sup>2</sup>, when injected with the proper predictive models, can provide early warning of potential network faults within a communication network. However, identifying a solution and marshaling automated solution entities within an active network has not yet been fully addressed. Initial work has begun to lay the groundwork for such automated composition of management solutions. This direction is being carried forward by exploration into the newly available processing power of Active Networks through the concept of Complexity and Algorithmic Information. This is achieved using small models designed for dynamic insertion into an active packet (called “Streptichrons” from classical Greek meaning to “bend time”).

Legacy networks, which are today’s passive networks, have been designed to optimize transmission of passive data using bit compression based upon the underlying notion of Shannon Entropy. ATROPOS has shown that active

networks allow for the possibility of executable models, and that the corresponding active, or executable network code packets, might be best studied with Kolmogorov Complexity as the underlying theory. It is fortunate that Complexity Theory has been receiving more attention lately and is making significant theoretical progress at the same time that research into active networking is taking place. Active networks provide a new paradigm and enhanced capabilities, which, when combined with ideas from Algorithmic Information Theory [22] may lead to superior, innovative solutions to network management problems.

One possible approach proposes combining Kolmogorov Complexity with the science of Algorithmic Information Theory (sometimes called Complexity Theory) to build self-managed networks that draw on fundamental properties of information to identify, analyze, and correct faults and security vulnerabilities in a distributed information system. [41] Specifically, we suspect that complexity measures can be used to detect and analyze problems in a network and to facilitate techniques to remedy network faults. We also envision that Kolmogorov Complexity can be applied directly to improve the performance of ATROPOS. In [20] the concept of monitoring the change in Kolmogorov Complexity of a system was first introduced for Information Assurance.

One potential drawback to ATROPOS is the fact that ATROPOS itself consumes resources in an effort to predict resource usage in a network. Resource consumption by ATROPOS is tied directly to accuracy. Higher accuracy costs more in terms of bandwidth utilization, associated with simulation rollbacks and the concomitant transmission of anti-messages. Despite this relationship, potential exists to nearly reach the theoretical minimum amount of bandwidth to achieve the maximal model accuracy. This possibility exists because ATROPOS consists of many small, distributed models (each a description of a theory) that work together in an optimistic, distributed manner via message passing (data). Each ATROPOS model can be transferred, using Active Networks, as a Streptichron [12], which is any message that contains an executable model in addition to data. Using Streptichrons, the optimal mix of data and model can be transmitted to closely approximate the minimum MDL. Achieving maximal model accuracy at minimal bandwidth and CPU usage provides the best ATROPOS accuracy at the least cost in ATROPOS resource consumption.

Other possibilities exist to exploit Kolmogorov Complexity to improve ATROPOS performance. For example, one can apply the MDL technique to the rollback frequency of all the ATROPOS enhanced nodes in a

<sup>1</sup> To appear in the *Complexity* Journal, vol. 9, issue 2, Nov-Dec 2003, Special Issue: “Resilient & Adaptive Defense of Computing Networks. Funded by the Defense Advanced Research Projects Agency (DARPA) contract F30602-01-C-0182 and managed by the Air Force Research Laboratory (AFRL) Information Directorate. Our thanks go to Doug Maughan, the Active Networks Program Manager and Scott Shyne, Air Force Rome Labs, for their generous support throughout our research.

Stephen F. Bush is with the GE Global Research Center, Niskayuna, NY, 12309, USA phone: 518-387-6827; fax: 518-387-4042; e-mail: bushsf@research.ge.com.

<sup>2</sup> See <http://www.comsoc.org/livepubs/ni/public/2002/jul/nisoft.html> for details on ATROPOS.

network. A low rollback complexity (which suggests a high compressibility in the observed data) would indicate patterns in the rollback behavior that could be corrected relatively easily by tuning ATROPOS parameters. High complexity (low compressibility) would indicate the lack of any computable patterns, and would suggest that little performance improvement could be achieved by simply tuning parameters. Thus, we hypothesize that our tuning gradient should be guided toward regions of high complexity, which suggests that one can tune parameters to improve the rollback frequency. The next section focuses upon experimental results relating prediction to complexity gathered from the operation of the ATROPOS system.

## II. ATROPOS AND KOLMOGOROV COMPLEXITY

In ATROPOS, information that impacts the network is transmitted based upon prediction at a low level within the network. Thus, ATROPOS allows experimentation in defining the boundaries within which active networking is beneficial. In Figure 1 an active and passive form of ATROPOS is represented. The passive case is represented in the upper portion of the figure. In the passive case, actual data ( $D_x$ ) is observed at the Driving Process. A hypothesis is formed about the data, and predicted data ( $D_y$ ) is generated in the form of static virtual messages. The term static indicates that information content within the message contains no executable code. When error in the hypothesis exceeds a preset threshold, ATROPOS causes rollbacks to occur in order to adjust for the inaccuracy. In the lower portion of Figure 1, the hypothesis is included within each packet and is used for encoding within the code portion of the active packet.

What is the relationship between the estimated operating hypothesis ( $H_e$ ) in the ATROPOS packet encoding and  $H_e$  as the predictor in the Driving Process? First, they are the same hypothesis. Second, it has been shown [45] that the shorter the packet, the better the predictor. Conversely, the worse the prediction, the longer the value is within the ATROPOS packet encoding.

ATROPOS benefits from the fact that the smallest algorithmic form is also the most likely predictor of a sequence. This can come about because Driving Processes and Streptichrons (active virtual messages anticipating events in the future) benefit by being both small and accurate, as shown in Figure 1. The objective is to increase the rate of convergence of the predictions held within the State Queue to converge to the actual value that will occur in the future, and to converge to the value before it actually exists. Actual and predicted values within a particular instance of a State Queue were shown in Figure 2. Let us examine ATROPOS results in light of complexity in more detail in the next section.

### A. Self-regulation via complexity

As predictions become more inaccurate in ATROPOS, virtual messages should slow down, rather than burden the system with potential rollbacks. Poorly predicted messages will naturally be larger in their minimum size ( $K(x)$ ), which

slows down their rate of propagation in proportion to their inaccuracy.

Another issue concerns a mechanism for feedback to the Driving Process in order to improve the Driving Process. Such a feedback mechanism can be based upon input from the complexity estimate, or minimum encoded packet size, of virtual messages. The hypothesis is adjusted in a manner that drives the system towards minimizing encoded virtual message size.

The next section takes self-regulation a step further and considers the role of complexity theory in a self-healing communication network.

## III. A SELF-HEALING COMMUNICATION NETWORK

Fault tolerant and self-healing systems should have the ability to self-compose solutions to faults. This 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 currently implemented 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.

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 novelty 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, is 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 the Internet is already “too” complex.

It is unlikely that an adaptive fault tolerant system, no matter how resilient, would 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 to answer these questions by proposing a complexity measure, Kolmogorov Complexity, and proposing an adaptation mechanism, Genetic Programming, based upon an analogy

with biological systems. Complexity was applied to optimize the combined use of communication and computation within an active network to determine the optimal amount of code versus data [20]. 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. 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 that evolve themselves 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 include other types of network services.

Frameworks for protocol and service composition have been developed for active networks, one of which is well-described reference [5]. However, research 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 are required in the network. 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, that is, a variable  $H_e$ .

The inspiration for a genetic algorithm based approach to solution composition comes from nature in the form of molecular biology's docking problem. Solutions that efficiently match a particular fault should be able to "dock" with that 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, which is discussed later in the 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: (1) Network mis-configuration (2) Bandwidth and processor misallocation (3) Faults caused by distributed Denial of

Service and virus attacks (4) Poor traffic shaping Routing problems (5) Non-optimal fused data within the network (6) Poor link quality in wireless and mobile environments (7) Mal-composed protocol framework models in the network (8) Poorly tuned components of network services.

A simple fault, namely misallocation 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".

#### IV. COMPLEXITY AND EVOLUTIONARY CONTROL

Complexity and evolution are intimately linked and descriptive complexity, specifically Kolmogorov Complexity, is one way to quantify this linkage. Kolmogorov Complexity ( $K(x)$ ) 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, [43], [2] security, [8], [29] and bioinformatics. Active networks form an ideal environment in which to study the effects of trade-offs 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 3, there is a striking similarity between an active packet and DNA. Active packet code can modify the operation of the network and effect transcription control on succeeding active packets. Both carry information having algorithmic and non-algorithmic portions. The algorithmic portion of DNA has transcription control elements as well as the Codons [6]. 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 [23][26]. Genetic Programming benefits from Kolmogorov Complexity as a measure and means of controlling not only the complexity, but also the size and generality of the result. One of the most obvious uses for complexity in networking is programmatic compression. In this paper, the foundation is developed for using 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. 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. Kolmogorov Complexity, estimated via the Minimum Description Length algorithm, can be a 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. As discussed earlier in this paper, direct application of Minimum Description Length (MDL) [52] can be applied to an active packet. Let  $D_x$  be a binary string representing  $x$ . Let  $H_x$  be a hypothesis or model, in algorithmic form, which 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  $\phi$  is a null instruction.  $RF$  is the data generated by  $RP_F()$ . Since the fault does not erase data, the process is reversible 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 is referred to as an anti-fault.

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, firstly, 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. Secondly, 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. Thirdly, 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 [21].

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, and avoid local extremes, by using evolutionary-like processes dependent on “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.

## V. 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 references [35] and [36]. 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 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 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 elected strings is determined, a position along the string is chosen t 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 his 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 no 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. Kolmogorov Complexity and Fitness. 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.

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 [43] the relationship between fault and energy is explored and simulated (see [29] and [28] for recent work on complexity, energy, and information assurance). The motivation for that simulation came from the relationship between Kolmogorov Complexity and entropy.

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.  $K(x)$ , 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 [44] reported the results of Kolmogorov Complexity probes that detect Distributed Denial of Service attacks. 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, gene pairs are selected 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 4. 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 5 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 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. The cumulative fitness function results (multiplied by 10 for easier comparison with the estimated complexity) are shown in Figure 6. 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.

#### A. Towards a Self-Evolving Network System

This paper focuses on progress towards self-composition of solutions assuming that other techniques, particularly complexity-based techniques, have identified faults. A previously described problem with using the genetic algorithm-based approach 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.

#### B. Genetic Network Programming Architecture

Genetic material begins in a random state ( $M$ ) and converges to the complexity of the optimal value produced by the fitness function. This enables solution composition from a wide range of possible solutions. One problem with this approach is the time required in evolving towards a feasible solution. Another problem is the fitness function itself must be self-generated. Using ATROPOS [10], the fitness function exists in the form of  $H_e$  where  $H_e$  is the estimated Virtual Network Management Prediction. In summary, the experiment in this section has shown a relationship among fitness, complexity and the evolution of genetic material.

The Magician Active Network [12][42] overlay network, a component of the ATROPOS system mentioned earlier, is used to test the feasibility of the genetically programmed network service concept. An active packet representing the nucleus (assuming network nodes are like eukaryotes- 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 injecting basic building blocks, known as functional units, into the network. 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. From the algorithm description in Table 1, Step 5 is similar to known genetic programming techniques. The primary difference from known genetic programming techniques is that it is occurring in real-time, during normal operation, within a communication network. Step 4 indicates that normal communication traffic flow occurs while Step 5 is repeatedly executed.

**Table 1: Adaptive In-line Genetic Communication Network Programming Algorithm.**

- |   |
|---|
| <ol style="list-style-type: none"> <li>1) Inject Functional Units             <ol style="list-style-type: none"> <li>a. Propagate to all nodes</li> </ol> </li> <li>2) Inject Nucleus             <ol style="list-style-type: none"> <li>a. Propagate to all nodes</li> </ol> </li> <li>3) Inject Fitness Function             <ol style="list-style-type: none"> <li>a. Propagate to all nodes</li> <li>b. Choose three functional units to form three independent chromosomes</li> </ol> </li> <li>4) Clone ‘normal’ packet traffic through each chromosome while simultaneously performing Step 5.</li> <li>5) Begin Operation             <ol style="list-style-type: none"> <li>a. If mutation event is to occur                 <ol style="list-style-type: none"> <li>i. Generate a random change in one of the functional units to another available functional unit</li> </ol> </li> <li>b. If recombination event is to occur                 <ol style="list-style-type: none"> <li>i. Select a cross-over point and swap portions of best parent chromosomes</li> </ol> </li> <li>c. Otherwise, for each chromosome, select an available functional unit at random and append to each chromosome</li> <li>d. Compute Fitness                 <ol style="list-style-type: none"> <li>i. ‘Normal’ packets flow through each chromosome</li> <li>ii. Compute a fitness value based upon monitoring desired chromosome objectives, such as latency or jitter</li> <li>iii. Return fitness value to the nucleus</li> </ol> </li> <li>e. Assign fitness to each chromosome</li> <li>f. Repeat from Step 5.a until a desired fitness threshold is reached</li> </ol> </li> </ol> |
|---|

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 7. Once a chromosome is assembled, the Codons can be translated into Amino Acids at the Ribosome. 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 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 8, 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 9, entering the center node is split into multiple streams. A different chromosome processes each stream. 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 10, fitness functions can be designed to measure Quality of Service (QoS) 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 uses its capabilities to maximize successful packet reception probability, in other words, perform the function of routing packets. 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 must work together towards reaching the stated goal in a reasonable manner. More will be said about the fitness function later. In Figure 11, recombination can occur both within a node and between two nodes. In addition, as shown in Figure 12, 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 were unexpected and sometimes detrimental to the intended operation of the network. As a trivial example, slowing the traffic to a near halt can minimize the variance. Thus, a low latency term had to be added to the jitter control fitness function.

### C. Jitter Control: Experimental Conditions and Topology

While *a priori* techniques have been developed for jitter control in legacy networks jitter control [16][51] forms a simple, easily measured, and controlled application for demonstration of the adaptive in-line 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. An adaptive jitter control mechanism was developed on a fixed, wired active communication network having the topology shown in Figure 12. The genetic algorithm was implemented as an active application in the Magician Active Network Execution Environment from the ATROPOS in-line communication network predictive system was used as the underlying code base. The experiment was run on a 10 Mbs Ethernet and four Sun Sparc Ultra 10 computers. Packets originate from the left-most node in Figure 12 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 12 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. Figure 13 through Figure 16 show packet link transit variance through three of the chromosomes on the destination node and shows packet link transit variance without any jitter control mechanism at the destination node. Initial observation of the graphs shows that, overall, the Chromosomes significantly reduced packet transit variance by a factor of 100. 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.

## VI. CONCLUSION

This paper presents initial results from a proto-type adaptive in-line communication network that evolves network services in a generic manner. It is demonstrated that complexity, specifically estimates of Kolmogorov Complexity, plays a critical role analyzing both fault analysis and in self-composing anti-faults.

A simple anti-fault is demonstrated via the composition of in-line communication services for jitter control. The demonstration shows a reduction in jitter by a factor of 100 which is achievable more quickly by *a priori* defined jitter control algorithms. However, the important point of these initial results is not improved jitter control, but the spontaneous composition, or creation, of a new service from a set of basic building blocks, in real-time, inside the network, without *a priori* knowledge of the particular domain. The result is extra-ordinary adaptive capability and fault tolerance.

## REFERENCES

- [1] Alexander D. S., Braden B., Gunter C., Jackson A., Keromytis A., Minden G., and Wetherall D., editors. *Active Network Encapsulation Protocol (ANEP)*. Active Networks Group, July 1997. Request for Comments: draft.
- [2] Kulkarni, A. B. and Stephen F. Bush, *Active network management and kolmogorov complexity*, in Proceedings of IEEE OpenArch 2001, Anchorage Alaska, April 27-28, 2001.
- [3] Barabasi, A.-L., V. W. Freeh, H. Jeong, and J. B. Brockman, Parasitic computing. *Nature*, 412:894–897, Aug. 2001.
- [4] Bengtsson, Mats G., National Defense Research Establishment, Box 1165, S-581 11, Linköping, Sweden, email: [matben@lin.foa.se](mailto:matben@lin.foa.se)
- [5] Bhattacharjee, S., Calvert, Chae, Y., Merugu, S., Sanders, M. and Zegura, E., *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, pp. 255–272, May 2002.
- [6] Bower, James M. and Bolouri, Hamid, *Computational Modeling of Genetic and Biochemical Networks*, The MIT Press, 0-262-02481-0, 2001.
- [7] Bush, Stephen, F. *Active Virtual Network Management Prediction*. In PADS '99, May 1999.
- [8] Bush, Stephen, F. and S. C. Evans. *Kolmogorov complexity for information assurance*. Technical Report 2001CRD148, General Electric Corporate Research and Development, 2001.
- [9] Bush, Stephen, F. A. B. Kulkarni, V. Galtier, K. Mills, and Y. Carlinet. Prediction and controlling resource usage in a heterogeneous active network, Dec. 2000. DARPA Active Networks PI Meeting and Demonstration (<http://www.research.ge.com/~bushsf/an>).
- [10] Bush, Stephen F., *Active Virtual Network Management Prediction*. In Parallel and Discrete Event Simulation Conference (PADS) '99, May 1999.
- [11] Bush, Stephen F., *Islands of Near-Perfect Prediction*. In *Virtual Worlds Simulation Conference '00 and 2000 Western MultiConference*, January 2000
- [12] Bush, Stephen F. and Kulkarni, Amit B., *Active Networks and Active Virtual Network Management Prediction: A Proactive Management Framework*. ISBN 0-306-46560-4, Kluwer Academic/Plenum Publishers. Spring 2001.
- [13] Bush, Stephen F., and Kulkarni, Amit B. (GE), Galtier, Virginie and Carlinet, Yannick and Mills, Kevin L. (NIST), Livio Ricciulli (Metanetworks). *Predicting and Controlling Resource Usage in an Active Network*. DARPA Active Networks PI Meeting, December 6-9, 2000, Atlanta, GA.
- [14] Bush, Stephen F., *Active Virtual Network Management Prediction*. In Parallel and Discrete Event Simulation Conference (PADS) '99, May 1999.
- [15] Bush, Stephen F., *Islands of Near-Perfect Prediction*. In *Virtual Worlds Simulation Conference '00 and 2000 Western Multi-Conference*, January 2000
- [16] Bush, Stephen F., *Active Jitter Control*. In Intelligence in Services and Networks (ISN) '00, February 2000.
- [17] Bush, Stephen F. and Barnett, Bruce. *A Security Vulnerability Technique and Model*. GE Corporate Research and Development, January 1998. Technical Report 98CRD028.
- [18] Bush, Stephen F., and Kulkarni, Amit B. (GE), Galtier, Virginie and Carlinet, Yannick and Mills, Kevin L. (NIST), Livio Ricciulli (Metanetworks). *Predicting and Controlling Resource Usage in an Active Network*. DARPA Active Networks PI Meeting, December 6-9, 2000, Atlanta, GA.
- [19] Bush, Stephen F. and Frost, Victor B. and Evans, Joseph B., *Network Management of Predictive Mobile Networks*. *Journal of Network and Systems Management*, Vol. 7, No. 2, June 1999.
- [20] Bush, Stephen, F. and Scott C. Evans, *Kolmogorov Complexity for Information Assurance*, GE Corporate Research and Development Technical Report 2001CRD148.
- [21] Carothers, C. D., Bauer, D. and Pearce, S., Ross: A High-Performance, Low Memory, Modular Time Warp System, 2000.
- [22] Chaitin, G. J., *The Limits of Mathematics*, Lecture Notes in Computer Science, volume 888, ISSN 0302-9743, 1995.
- [23] Conte, M., Tautteur, G., De Falco, I., Della Cioppa, A. and Tarantino, E., *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.
- [24] Cover, T. M. and Thomas, J. A., *Elements of Information Theory*, Wiley, NY, 1991.
- [25] Crutchfield, J. P., *The Calculi of Emergence: Computation, Dynamics and Induction*, *PhysicaD*, vol. 75, no. 1–3. Pages 11-54, 1994.
- [26] De Falco, T., Iazzetta, A., Tarantino, E., Della Cioppa, A. and Trautteur, G., *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.
- [27] Evans, Scott, Bush, Stephen F., and Hershey, John, *Information Assurance through Kolmogorov Complexity*, DARPA Information Survivability Conference and Exposition II (DISCEX-II 2001) 12-14 June 2001, Anaheim, California.
- [28] Evans, Scott C. and Barnett, Bruce, *Network security through conservation of complexity*, in MILCOM, The Disneyland Resort, Anaheim, CA, USA, Oct 2002, IEEE.
- [29] Evans, Scott, Bush, Stephen F., and Hershey, John, *Information assurance through Kolmogorov Complexity*, DARPA Information Survivability Conference and Exposition II (DISCEX-II 2001) 12-14 June 2001, Anaheim, California, Proceedings Vol. 2, pp 322-331, <http://www.research.ge.com/~bushsf/fin>.
- [30] Evans S. C. and S. F. Bush. Symbol compression ratio for string compression and estimation of Kolmogorov Complexity. Technical Report 2001CRD159, General Electric Corporate Research and Development, 2001.
- [31] Evans Scott, Bush Stephen F., and Hershey, John, *Information Assurance through Kolmogorov Complexity*. DARPA Information Survivability Conference and Exposition II (DISCEX-II 2001), 12-14 June 2001, Anaheim, California.
- [32] Galtier, Virginie and Mills, Kevin L. (NIST) and Carlinet, Yannick and Bush, Stephen F. and Kulkarni, Amit B., *Predicting Resource Demand in Heterogeneous Active Networks*, MILCOM 2001, McLean, VA, October 28-31.
- [33] Galtier, Virginie and Mills, Kevin L. (NIST) and Carlinet, Yannick and Bush, Stephen F. and Kulkarni, Amit B., *Prediction and Controlling Resource Usage in a Heterogeneous Active Network*, Third Annual International Workshop on Active Middleware Services 2001, San Francisco, CA, August 6, 2001.
- [34] Gao, Qiong and Li, Ming and Vitányi, Paul M., *Applying MDL to Learning Best Model Granularity*, arXiv: physics/0005062, May 23 2000.
- [35] Goldberg, David E., *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley, 1989.
- [36] Goldberg, David E., *Genetic and Evolutionary Algorithms Come of Age*. Communications of the ACM, pp 113-119, March 1994.
- [37] Howard, John, D, *An Analysis of Security Incidents on the Internet 1989-1995*, Theory, Vol. 44, July 1998, Ph.D. Thesis, Carnegie Mellon University, Apr. 1997.
- [38] Kircher, W., M. Li, and P. Vitanyi. *The Miraculous Universal Distribution*. The Mathematical Intelligencer, 1997.
- [39] Kulkarni, A. B. and S. F. Bush. Active network management and kolmogorov complexity. In *Proceedings of IEEE OpenArch 2001*, Apr. 2001.
- [40] Kulkarni, A. B. and S. F. Bush., *Detecting distributed denial of service attacks using kolmogorov complexity metrics*. Submitted to IEEE Computer Security Foundations Workshop, Feb. 2002.
- [41] Kulkarni, Amit B. and Bush, Stephen F., *Active Network Management, Kolmogorov, Complexity, and Streptichrons*. GE-CRD Class I Technical Report 2000CRD107 ([www.research.ge.com/~bushsf/fin](http://www.research.ge.com/~bushsf/fin)).
- [42] Kulkarni, A. B., Minden G. J., Hill R., Wijata Y., Sheth S., Pindi H., Wahhab F., Gopinath A., and Nagarajan A., *Implementation of a Prototype Active Network*. In *OPENARCH '98*, 1998.

- [43] Kulkarni, Amit B. and Bush, Stephen F., *Active network management, kolmogorov complexity, and streptichrons*, Tech. Rep. 2000CRD107, General Electric Corporate Research and Development, Dec. 2000, <http://www.research.ge.com/~bushsf/ftn>.
- [44] Kulkarni, Amit B., Bush, Stephen F. and Evans, Scott C., *Detecting Distributed Denial-of-Service Attacks using Kolmogorov Complexity Metrics*, GE Research Technical Report 2001CRD176. December 2001.
- [45] Li, Ming and Vitányi, Paul M., *Introduction to Kolmogorov Complexity and its Applications*, Springer-Verlag, August 1993.
- [46] Liu, G. Y. and G. Q. M. Jr. A Predictive Mobility Management Algorithm for Wireless Mobile Computing and Communications. In International Conference on Universal Personal Communications (ICUPC), pages 268,272, Nov 1995.
- [47] Rose, M. T., *The Simple Book, An Introduction to the Management of TCP/IP Based Internets*. Prentice Hall, 1991.
- [48] Thottan, Marina and Ji, Chuanyi, *Proactive Anomaly Detection using Distributed Intelligent Agents*. IEEE Network, Special Issue on Network Management, Sept/Oct 1998.
- [49] Thottan, Marina and Ji, Chuanyi, *Intelligent Processing Agents for Network Fault Detection*. IEEE Internet Computing, 2(2), March/April 1998.
- [50] Tinker, P. and Agra J., *Adaptive Model Prediction Using Time Warp*. In Proceedings of The Society for Computer Simulation (SCS), 1990.
- [51] Dinesh C. Verma, Hui Zhang, and Domenico Ferrari. Delay jitter control for real-time communication in a packet switching network. Technical report, International Computer Sciences Institute, 1991. TR-91-007.
- [52] Wallace, C. S. and D. L. Dowe. Minimum message length and Kolmogorov complexity. *The Computer Journal*, 42(4):270–283, 1999.



**Stephen F. Bush** (M'03-SM'03) Stephen F. Bush is internationally recognized with over 30 peer-reviewed publications. Dr. Bush has implemented a toolkit capable of injecting predictive models into an active network. The toolkit has been downloaded and used by more than 600 institutions. Stephen continues to explore novel concepts in complexity and algorithmic information theory to refine the toolkit for applications ranging from network management and ad hoc networking to DNA sequence analyses for bioinformatics applications. Dr. Bush has been the Principal Investigator for many DARPA and Lockheed Martin sponsored research projects including: Active Networking (DARPA/ITO), Information Assurance and Survivability Engineering Tools (DARPA/ISO), and Fault Tolerant Networking (DARPA/ATO). Stephen coauthored a book on active network management, titled *Active Networks and Active Network Management: A Proactive Management Framework*, published by Kluwer Academic Publishers. Before joining GE Global Research, Stephen was a researcher at the Information and Telecommunications Technologies Center (ITTC) at the University of Kansas. He received his B.S. in Electrical and Computer Engineering from Carnegie Mellon University and M.S. in Computer Science from Cleveland State University. He has worked many years for industry in the areas of computer integrated manufacturing and factory automation and control. Steve received the award of Achievement for Professional Initiative and Performance for his work as Technical Project Leader at GE Information Systems in the areas of network management and control while pursuing his Ph.D. at Case Western Reserve University. Steve completed his Ph.D. research at the University of Kansas where he received a Strobel Scholarship Award. Stephen is currently a Computer Scientist at General Electric Global Research in Niskayuna, NY.

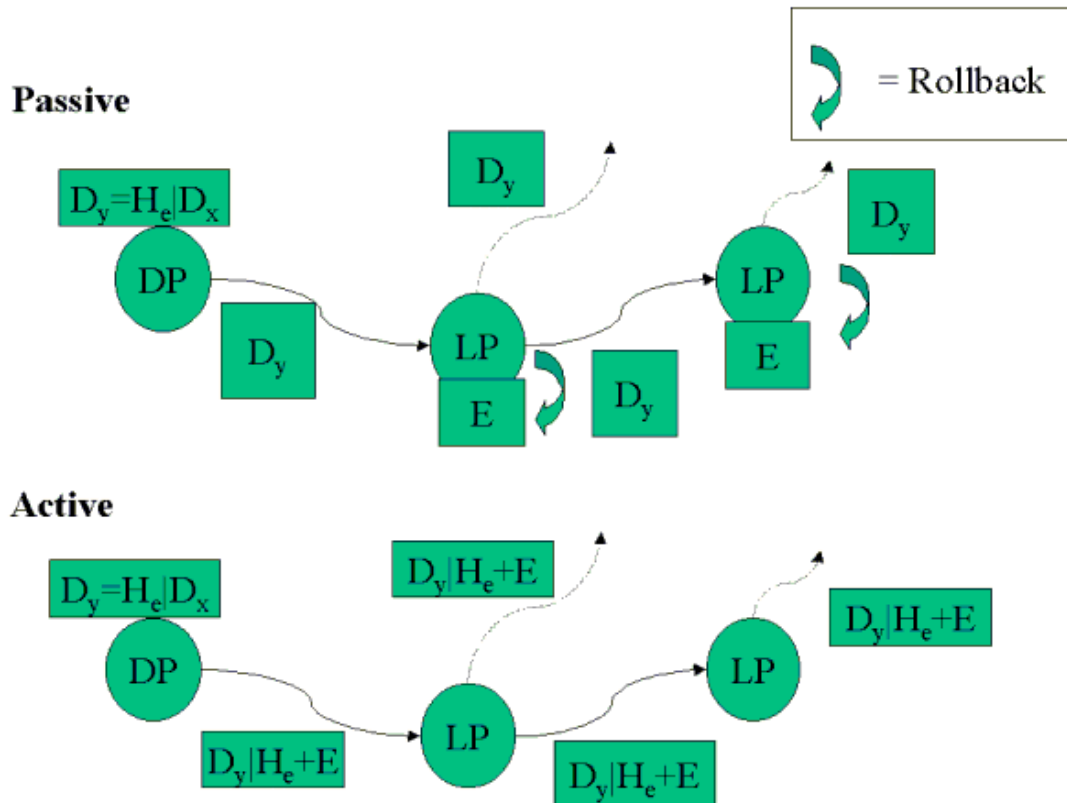


Figure 1. Active versus Passive Form ATROPOS.

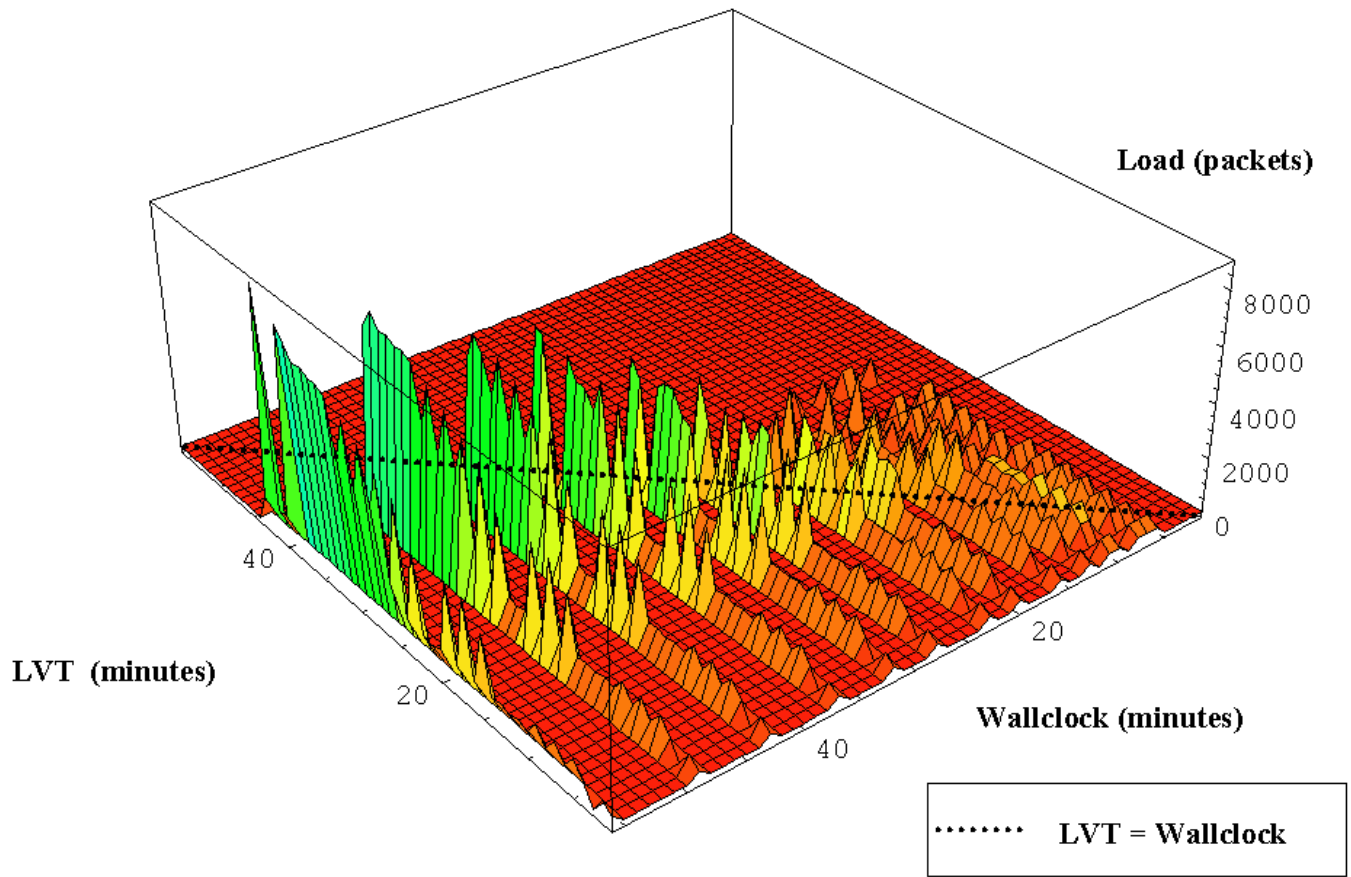


Figure 2: Load Prediction Application Results on a Single Node.

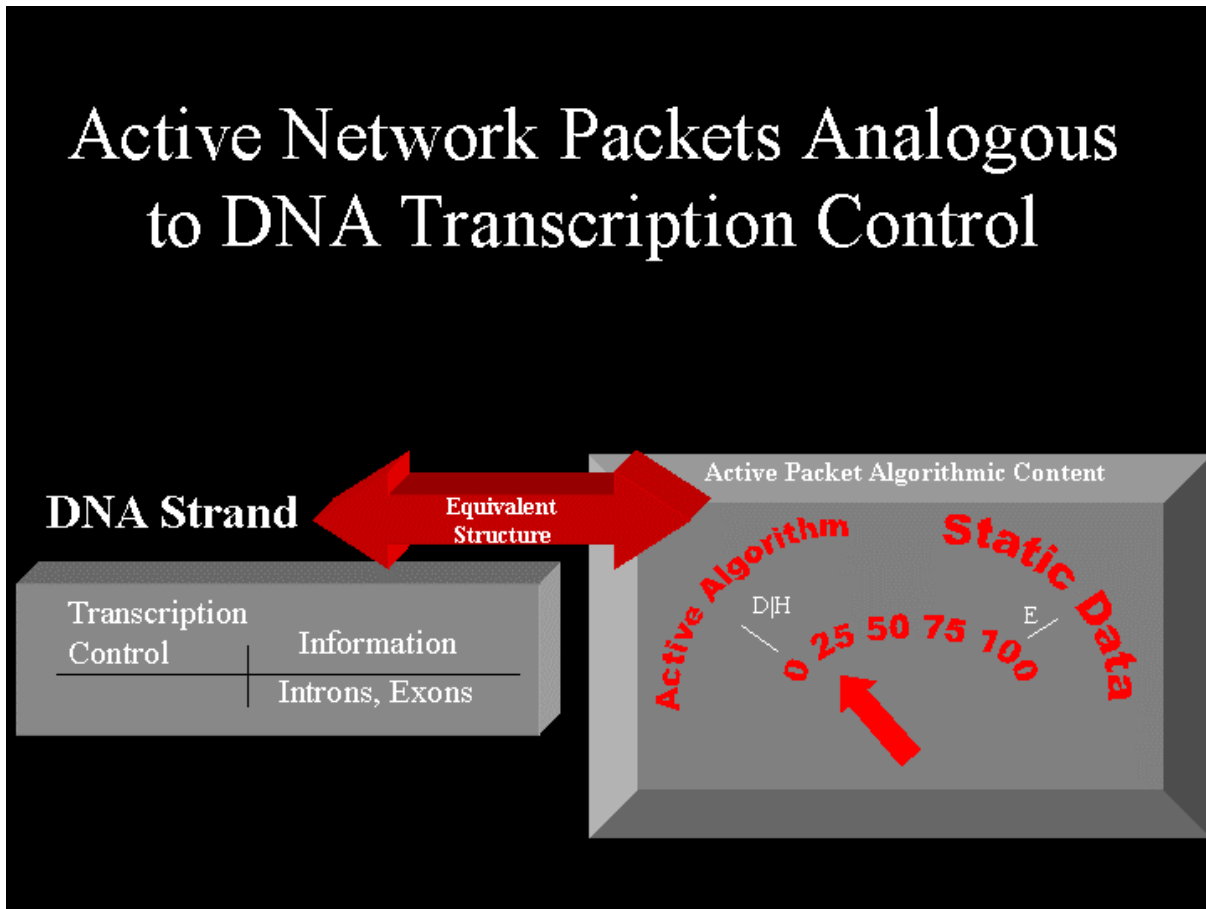


Figure 3. Active Packet as DNA.

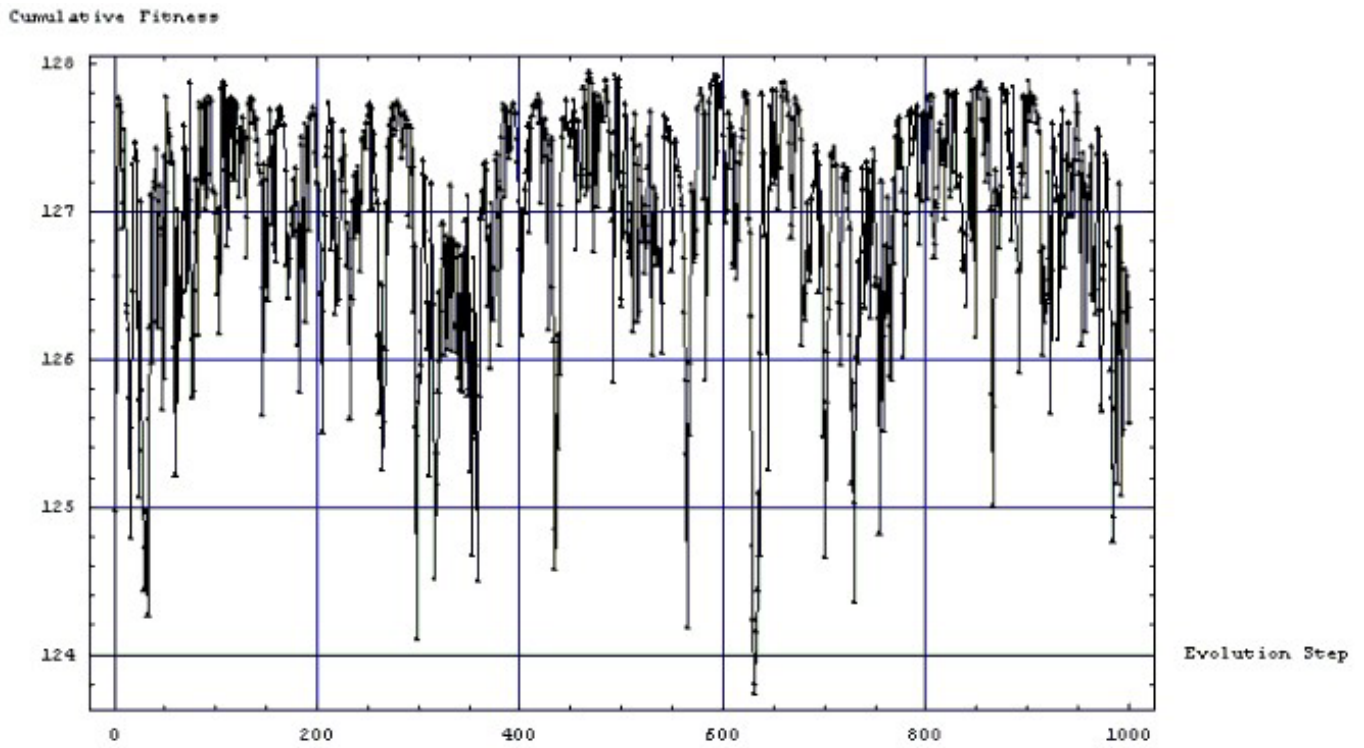


Figure 4. Cumulative Fitness of Genetic Material.

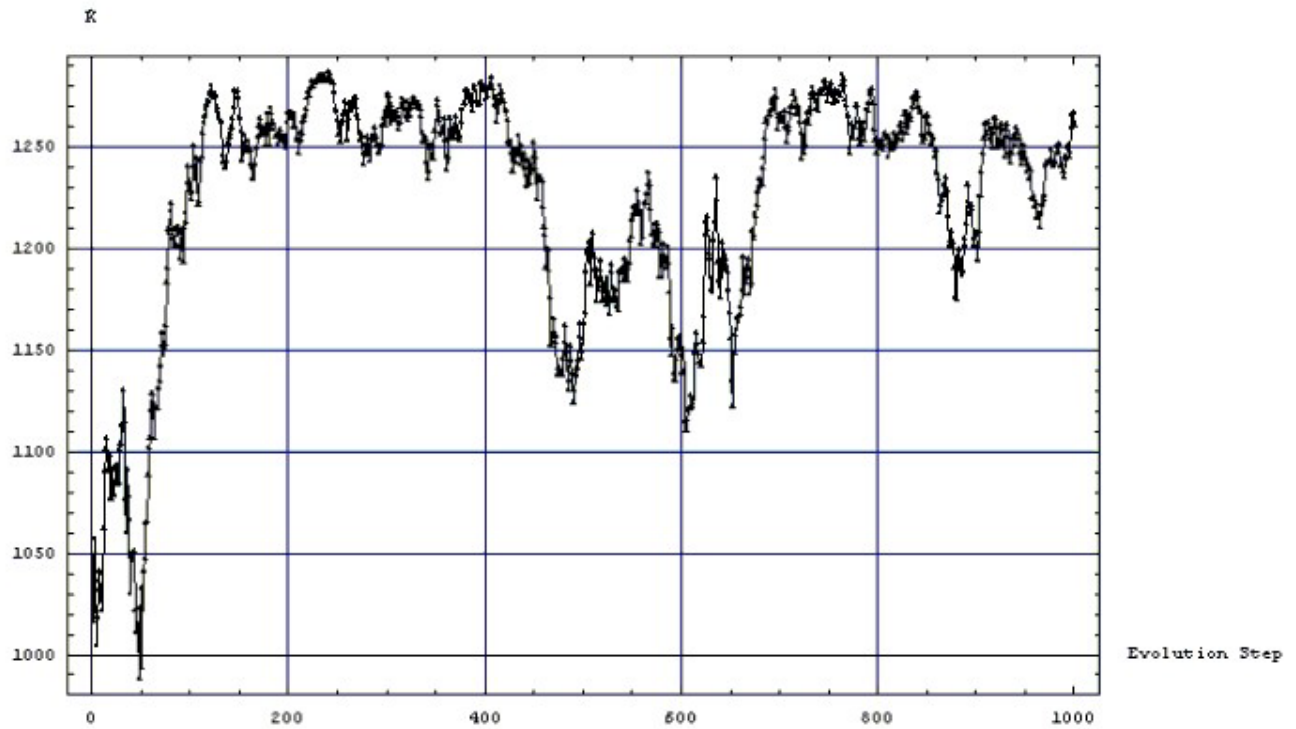


Figure 5. Complexity Estimate of Genetic Material Versus Time.

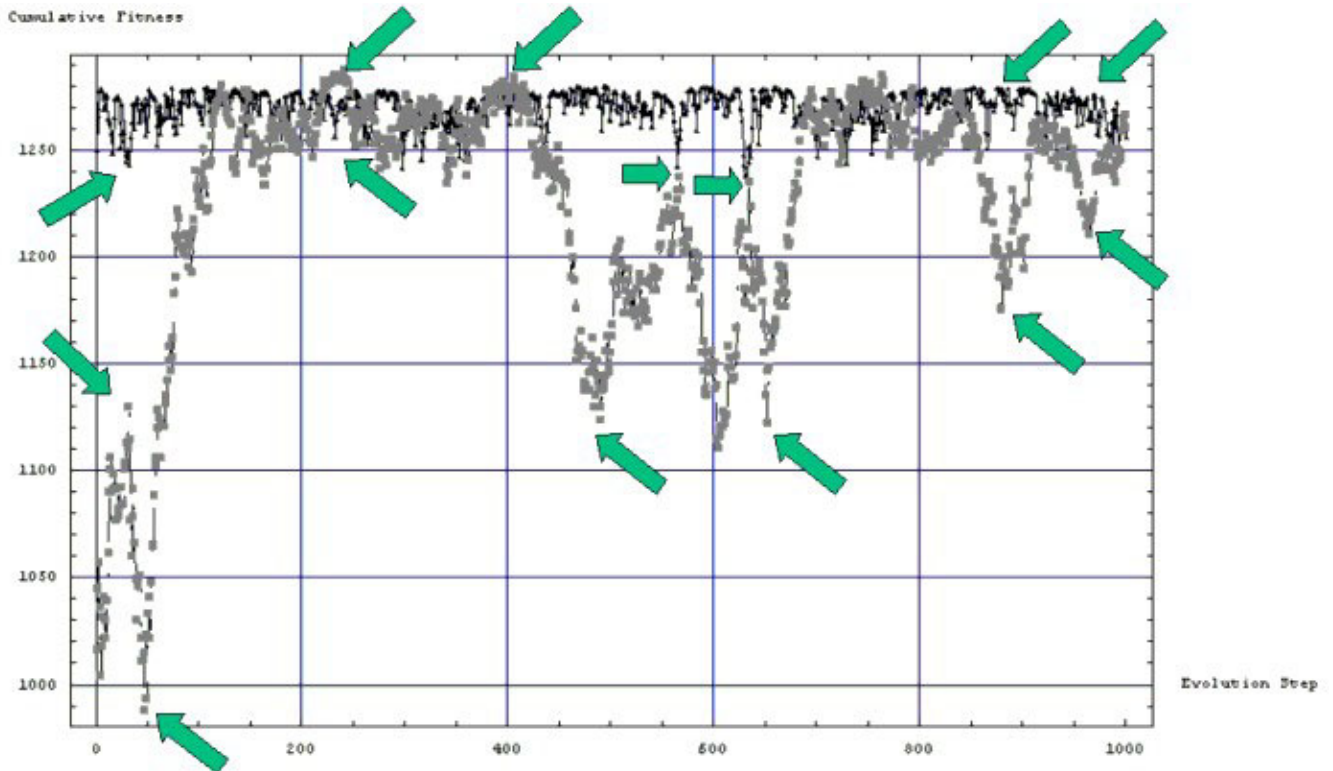


Figure 6. Correlation Between Fitness and Complexity Versus Time.

# Architecture

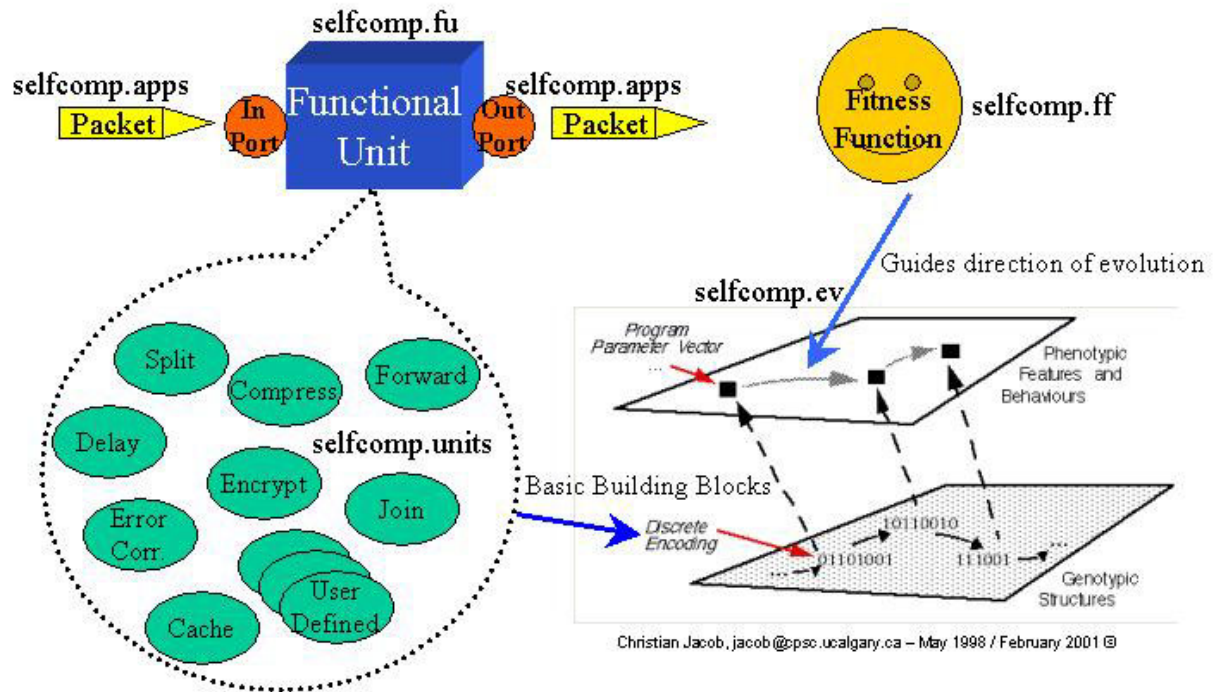


Figure 7. Functional Units, Evolution, and Fitness.

# Single Node Active Evolutionary Control Architecture

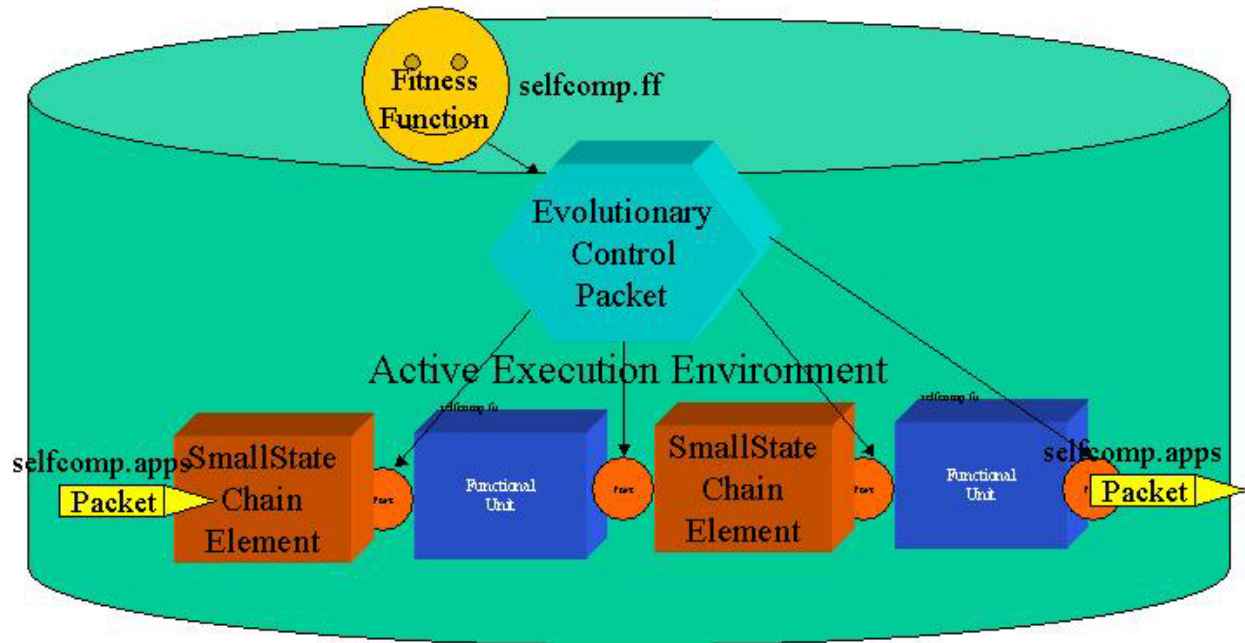


Figure 8. Architecture of a Single Node.

## Traffic and Evolution

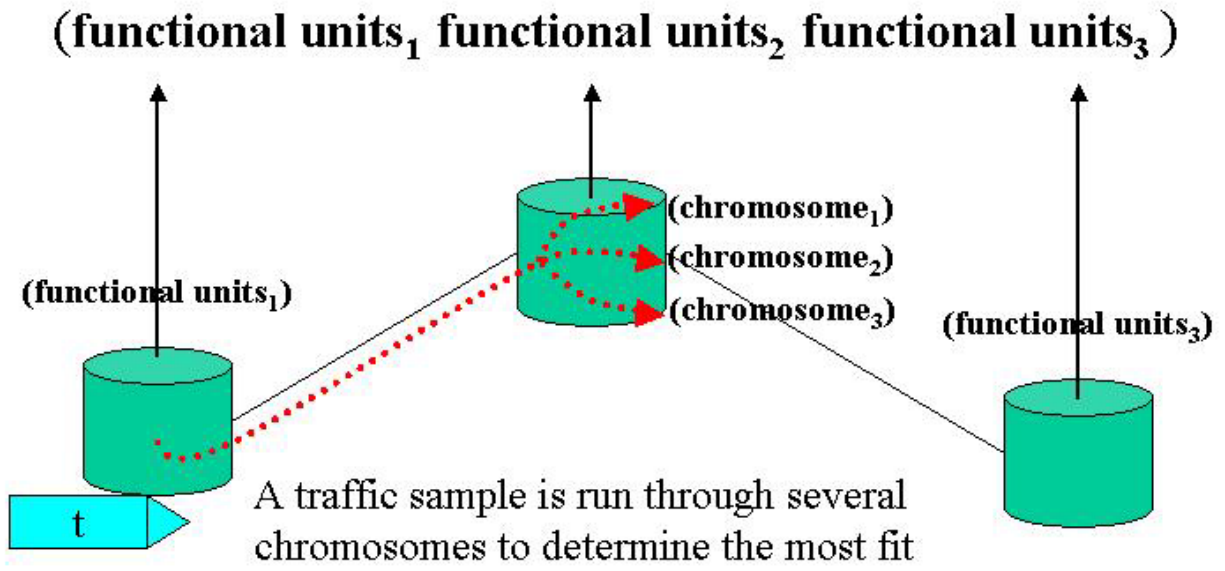


Figure 9. Breeding Traffic Streams.

### Multi-Level Fitness Functions

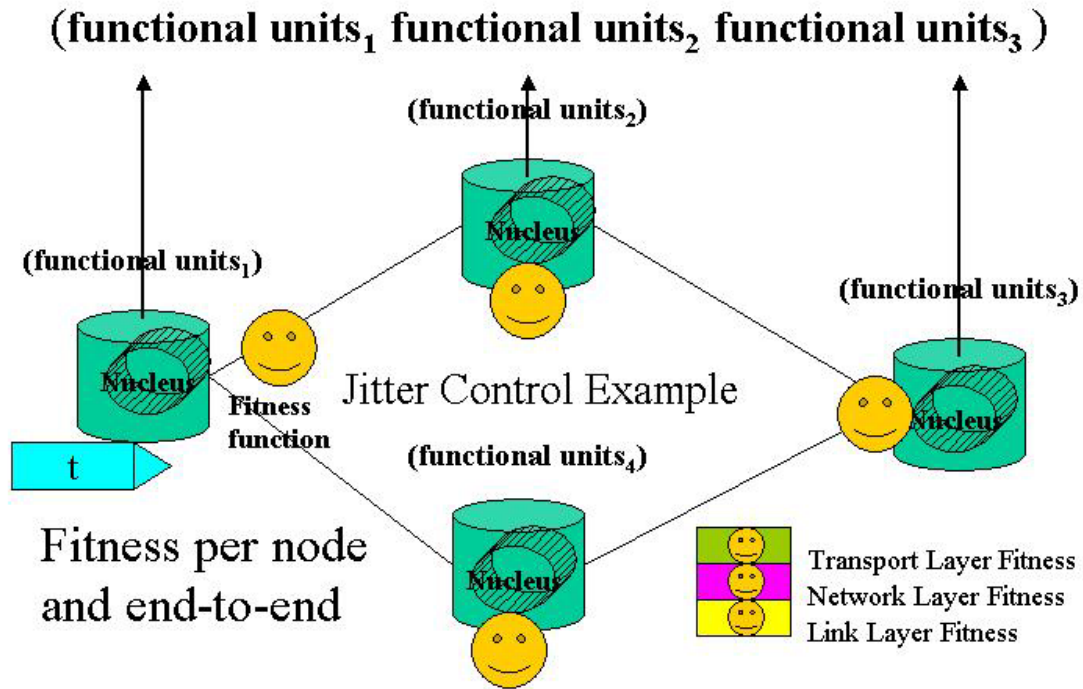


Figure 10. Multiple Levels of Fitness.

### Evolution Hierarchies

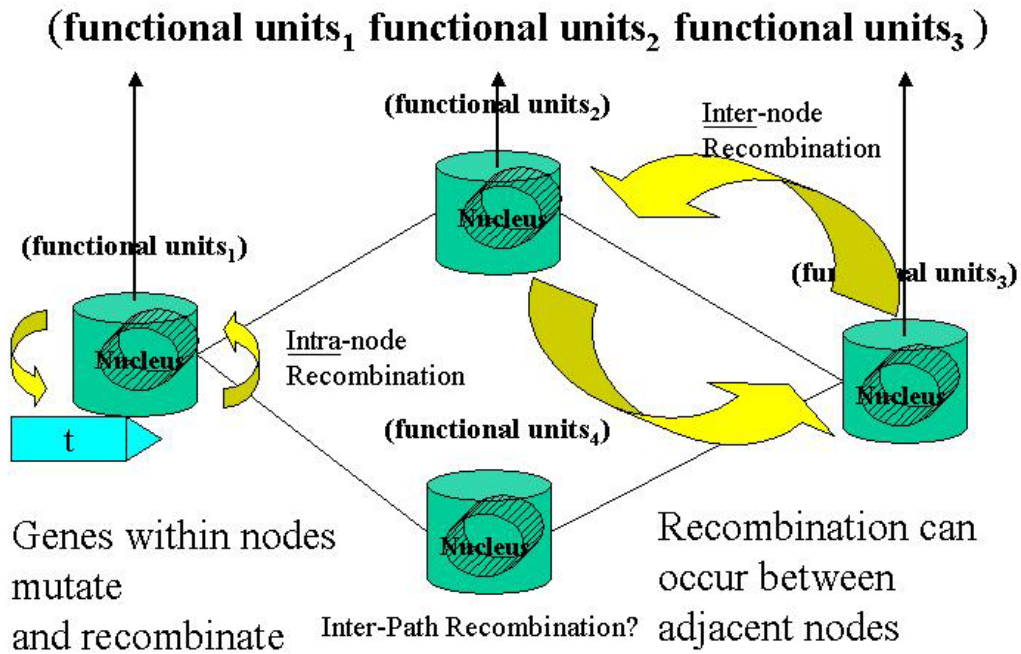


Figure 11. Types of Recombination.

## Effective Chromosome Based Upon Route

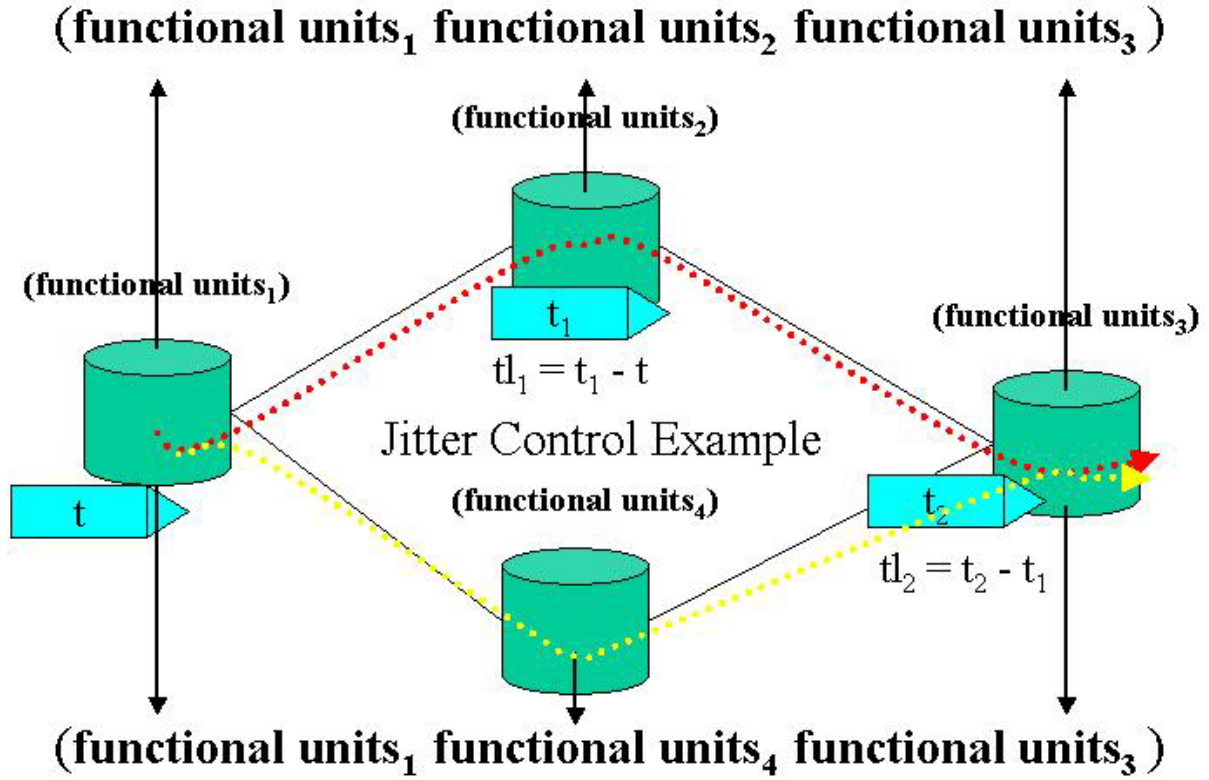


Figure 12. Chromosomes and Routing.

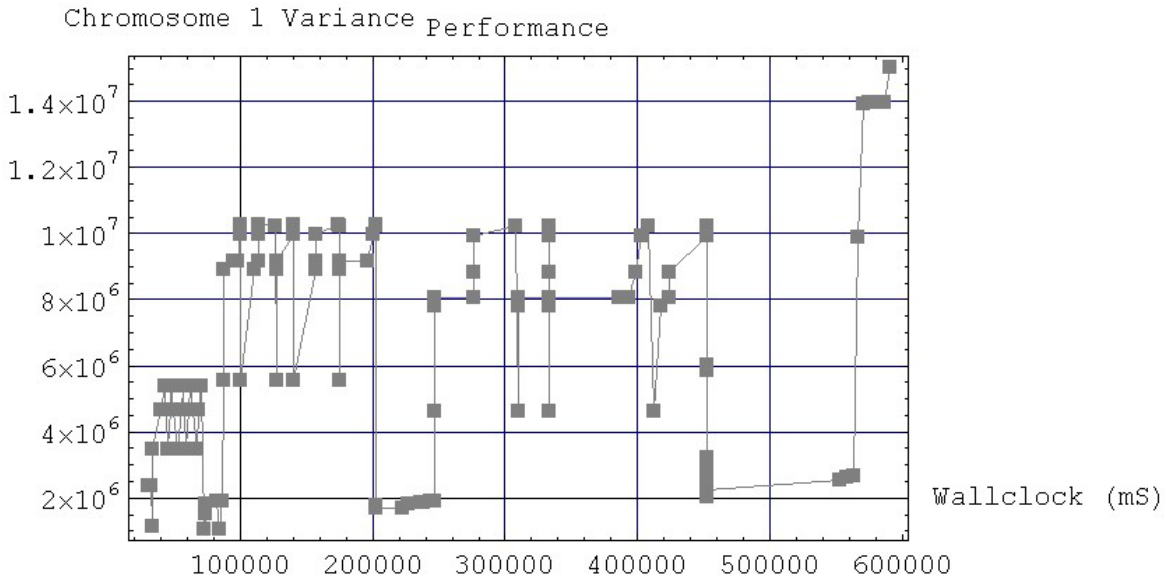


Figure 13. Packet Link Transit Variance (milliseconds<sup>2</sup>) on Designation Node Through Chromosome One.

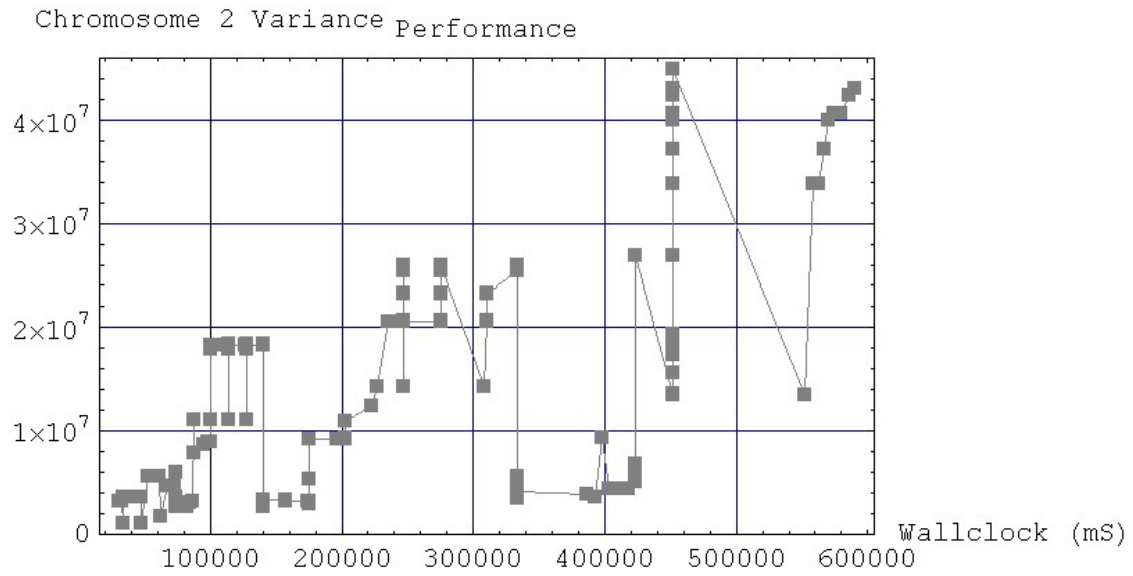


Figure 14. Packet Link Transit Variance (milliseconds<sup>2</sup>) on Designation Node Through Chromosome Two.

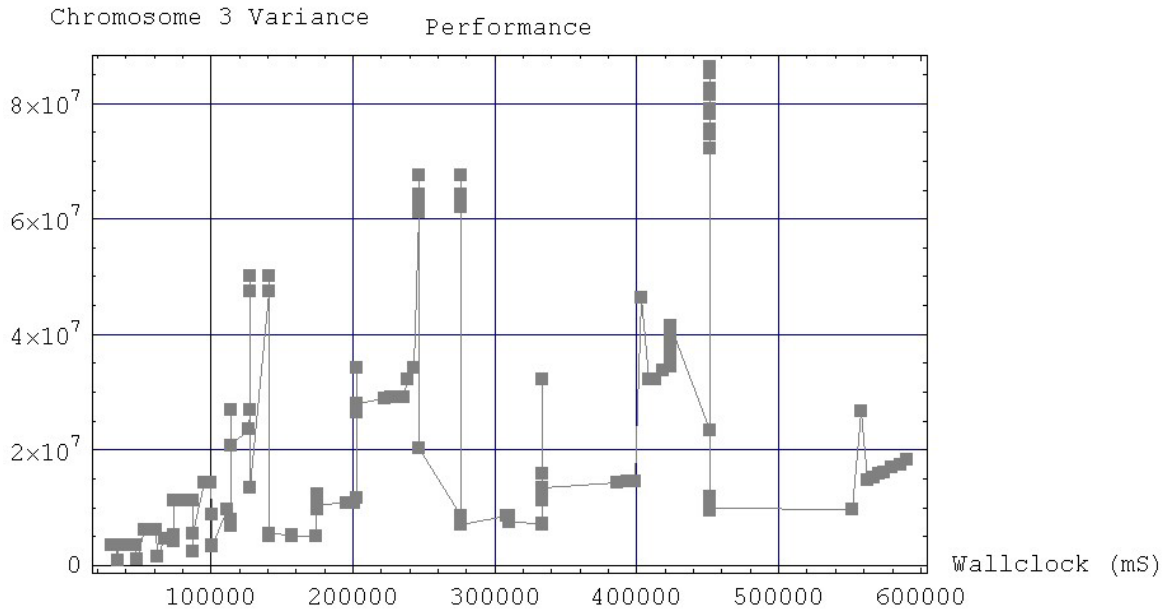
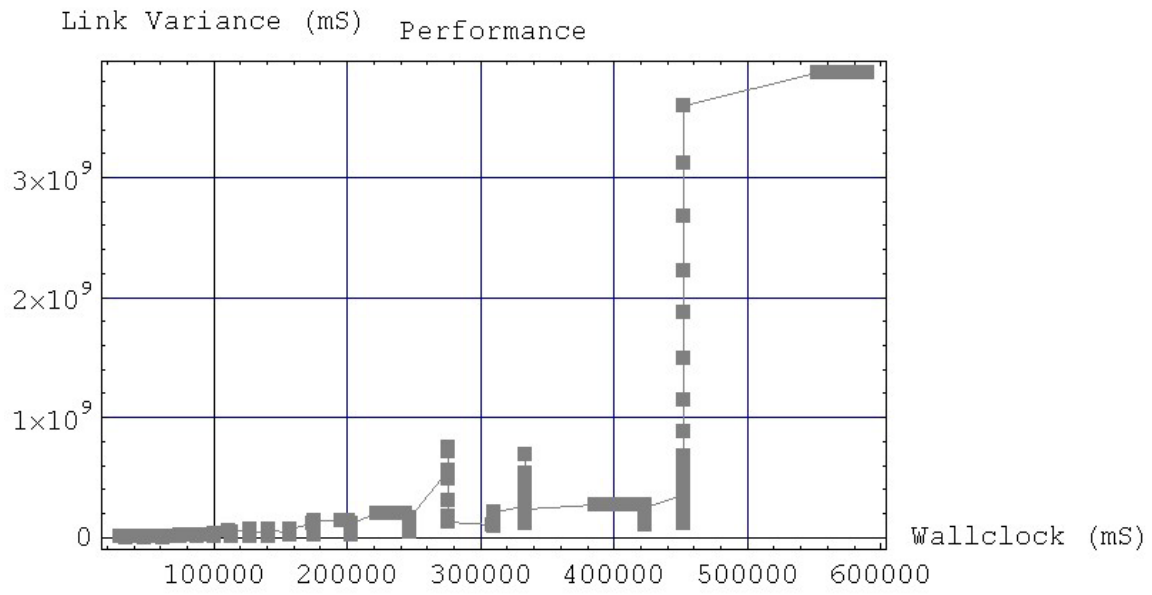


Figure 15. Packet Link Transit Variance (milliseconds<sup>2</sup>) on Designation Node Through Chromosome Three.



**Figure 16. Packet Link Transit Variance (milliseconds<sup>2</sup>) on Designation Node Without Jitter Control.**