# Adaptive Reconfiguration of Data Networks Using Genetic Algorithms*

**David Montana and Talib Hussain**
BBN Technologies
10 Moulton Street, Cambridge, MA 02138
{dmontana,thussain}@bbn.com

## Abstract

Genetic algorithms are applied to an important, but little-investigated, network design problem, that of reconfiguring the topology and link capacities of an operational network to adapt to changes in its operating conditions. These conditions include: which nodes and links are unavailable; the traffic patterns; and the quality of service (QoS) requirements and priorities of different users and applications. Dynamic reconfiguration is possible in networks that contain links whose endpoints can be easily changed, such as satellite channels, terrestrial wireless connections, and certain types of optical connections. We report preliminary results that demonstrate the feasibility of performing genetic search quickly enough for online adaptation.

## 1 INTRODUCTION

There is a growing need for networks to adapt to their operating conditions in order to maintain acceptable levels of performance. Networks must increasingly be able to continue to function effectively despite obstacles such as the disabling of portions of the network by cyberattacks or large fluctuations in the traffic patterns and service requirements. Network adaptation potentially enables not just fine-tuning in response to normal variations but also survivability of the network and its critical applications in the face of catastrophic failures and large-scale shifts in operating conditions.

Although dynamic routing solutions (e.g., [2]) to some of these problems exist, routing has natural limitations. For example, a routing algorithm cannot transmit data between nodes for which cyberattacks have disabled all connecting paths, nor can it transmit a high bandwidth of data between nodes which have only a low-bandwidth path between them. Robust network adaptation requires changes to the underlying network infrastructure (i.e. topology and link capacities) in response to changes in operating conditions.

Despite this need, the problem of automatic, dynamic redesign of functioning networks has received little attention. One reason for this is that network links were traditionally cables and hence not dynamically reconfigurable like satellite, wireless, and some optical links. Second, the optimization algorithms and computers of the past were not capable of finding a new network configuration fast enough to support adaptive reconfiguration. Third, factors driving the need for adaptive reconfiguration, such as cyberattacks and the desire to provide guaranteed and variable-level quality of service (QoS), have only recently become important.

In this paper, we investigate the use of a genetic algorithm to dynamically redesign a network with reconfigurable links. Before discussing our work, we provide a brief review of some previous related work. This includes work on the use of genetic algorithms for (static) network design as well as work on adaptively self-repairing systems.

### 1.1 PREVIOUS WORK: GENETIC ALGORITHMS FOR NETWORK DESIGN

There is not just one problem in network design but rather a whole family. There are three different components of a network architecture: the topology, the link capacities, and the routing policies. Different problems work with different subsets of these components. There are also three different basic criteria on which to judge a network: cost, reliability, and quality of service (QoS). Different problems use different subsets of

*This paper is an expanded version of one that appears in the GECCO-2002 proceedings [20].

these criteria, different measures of these criteria, and combine the criteria they do use in different ways.

A major focus has been minimal spanning tree problems (e.g., [22, 1, 6]). The only network component considered is the topology, and the topology is always a tree. There have been some novel chromosome representations used for these problems, including Prüfer encoding [22, 1] and Huffman trees [11].

When considering factors other than cost, the best topology is generally a graph rather than a tree. Different problems in optimizing non-tree topologies arise from different definitions of the evaluation criteria. For example, [17] and [10] use a probabilistic measure of reliability, while [13] and [24] use a measure of reliability based on redundancy. Given a numbering of all possible links between all pairs of nodes, graph topologies have been genetically represented as fixed length binary strings [17, 24] and as variable-length strings of unique integers [10].

With knowledge of the network traffic patterns, it is also possible to optimize the link capacities and routing policies. In early work, [7] used genetic algorithms to select a set of link capacities given a fixed topology. More recently, with the benefit of greater computational power, researchers have investigated using genetic algorithms to simultaneously optimize topology and link capacities [25, 13] or all three components of the network (topology, link capacity, and routing policies) [16, 15]. It is possible to use a single chromosome that represents all the required information about a network (e.g., [16]) or to use separate representations and solve for the different components in separate (nested) optimizations [15].

A broader view of the different techniques for static network design beyond genetic algorithms is given in [14]. An introduction to genetic algorithms is given in [18].

### 1.2 PREVIOUS WORK: ADAPTIVE, SELF-REPAIRING SYSTEMS

There is currently a growing interest in computing systems that are able to adapt and repair themselves. This general concept goes by a variety of names including *autonomic computing* [12] and *recovery-oriented computing* [23].

On the topic of network reconfiguration, there has been a lot of work on the redesign and adaptation of power system networks to compensate for failures in these networks [4, 8]. However, the problem of power system network design is very different from that of communications network design; with numerous linear constraints, power system design is well suited to classical linear programming techniques, while communications network design is not.

For communications networks, the focus of *self-healing networks* has been on rerouting packets around the failures [3]. As discussed above, this technique is greatly limited by the underlying network topology and link capacities. We now discuss the problem of how to adapt networks to failures by adapting the topology and link capacities.

## 2 ADAPTIVE REDESIGN

### 2.1 PROBLEM STATEMENT

The adaptive network redesign problem is that of determining a good configuration of the reconfigurable portions of the networking infrastructure (such as satellite channels, wireless connections, and some optical connections) for a network in operation. It is inherently a dynamic problem, since network traffic patterns, requirements and priorities, and available resources (primarily links and nodes) all change with time. Still, we solve the problem by considering snapshots at discrete times, performing the adaptation by solving for each snapshot largely independently. Our reasons for solving for discrete snapshots rather than continuously adapting are the following. First, the cost of network reconfiguration (updating routing tables, etc.) make it better not to update the network too often. Second, the hard-to-predict global effects on the network of local changes makes it better to perform a full network optimization rather than small changes. Third, waiting longer between adaptations allows time to gather data on persistent changes in the traffic and requirements, thus reducing the risk of reacting to transient effects.

However, there are still some important differences between this problem and the static network design problems. First, we do not completely redesign the network but only the reconfigurable components of the network, and even the reconfigurable portions tend to be more tightly constrained than in the static case. Second, the time for the optimization algorithm to find a solution becomes as important a consideration as the quality of the solution found, and this time is much more tightly constrained. (We somewhat arbitrarily have selected ten minutes as the maximum time allowed to find a solution.) Third, the current optimization should consider the previous network configuration, for the purposes of both minimizing changes and guiding its search.

The constraints on what reconfigurable links can be formed and what capacity they can have depend on the physical media and the low-level network protocols used. For the current work, we use a simple model of reconfigurable links based on the frequency-division multiplexing scheme used in satellites. There is a fixed amount of total reconfigurable bandwidth available. This bandwidth is unidirectional and is divided into identically sized chunks called *channels*. Reconfigurable links consist of one or more channels configured to have the same source node and destination node. The bandwidth of the channels of a reconfigurable link add, but the bandwidths of a reconfigurable link and a fixed link do not add. Instead, the link with the higher bandwidth is used and the other ignored. Each node has a limit on the number of channels it can send and receive, which is a type of node-degree constraint [6].

The givens of the problem include:

- **available nodes -** This is the set of all nodes not currently disabled by an attack or failure.
- **available fixed links -** This is the set of all fixed links not currently disabled by an attack or failure. Associated with each fixed link is a source node, destination node, capacity, and inherent transmission delay (which is the delay associated with the medium and does not include the delays due to queueing). Note that, for the purposes of our model, all fixed links are unidirectional; bidirectional links are decomposed into two unidirectional ones.
- **available channels -** For a given problem, the total number of channels, bandwidth per channel, and inherent channel transmission delay are fixed.
- **data flows -** Each data flow has associated with it the following information: source node, destination node, priority rating (a positive integer with smaller meaning higher priority), protocol (TCP or UDP), required transmission delay, required dropped packets, and the statistics of the generated traffic. We model the traffic as bursts of data of random number of bytes at random intervals, with Gaussian distributions for the number of bytes and the size of the interval. The mean and standard deviation are the required parameters for each of these distributions. Note that the quality of service (QoS) metrics (dropped packets and transmission delay, as described below) refer to the service as perceived by the application, not the network. In particular, a packet that is initially dropped but successfully retransmitted does not count as dropped but does register a long transmission delay. Hence, the dropped packets metric only applies to UDP flows, since TCP resends all dropped packets.

The variables over which to optimize are:

- **configuration of each channel -** Zero to all available channels may be added to the network topology. The source and destination nodes of each added channel must be specified.

The constraints to obey are:

- **send and receive limits -** The number of channels with a particular node as its source (destination) cannot exceed the send (receive) limit for that node.

The optimization criteria are:

1. **connectivity** - The measure of the degree to which flows are totally disabled due to lack of connectivity is the sum over all disconnected flows of $\frac{1}{\rho_i}$, where $\rho_i$ is the priority rating of the flow (recalling that a lower $\rho_i$ means a higher priority). Note that TCP flows will be disabled if there does not exist a path in both directions between the source and destination (to allow acknowledgements), while UDP flows only require a path in the forward direction.
2. **meeting transmission delay requirements** - The measure of the degree to which the network does not meet the transmission delay requirements is the sum over all connected flows for which the requirement is not met of $\frac{1}{\rho_i}(D_i - d_i)$, where $D_i$ is the average measured delay (in seconds), $d_i$ is the required delay (in seconds), and $\rho_i$ is the priority rating of the flow.
3. **meeting dropped packets requirements** - The measure for the dropped packets requirements is the sum over all connected flows for which the requirement is not met of $\frac{1}{\rho_i}(P_i - p_i)$, where $P_i$ is the average measured percent of packets dropped, $p_i$ is the required percent of packets dropped, and $\rho_i$ is the priority rating of the flow.
4. **minimizing changes** - The measure of the cost of reconfiguring the network is taken to be zero if all the reconfigurable links remain the same as before and one otherwise.

The three optimization criteria are combined into a single score using a weighted sum, $w_1S_1 + w_2S_2 + w_3S_3 + w_4S_4$, where $S_i$ is the score for the $i^{th}$ criterion. The goal is to minimize this combined score. For our experiments, we used $w_1 = 100$, $w_2 = 1$, $w_3 = 1$, and $w_4 = 0.001$.

## 2.2 GENETIC ALGORITHM

**Representation -** Each chromosome is a variable-length list of reconfigurable link *allocations*, where each allocation is a 3-tuple $(S, D, C)$ containing the source node ($S$), destination node ($D$), and the number of channels ($C$) connecting the source to the destination. Only allocations with one or more channels

are included in the list. For example, the chromosome [(6 3 1) (12 2 2)] indicates a reconfigurable link with 1 channel from node 6 to node 3, and a reconfigurable link with 2 channels from node 12 to node 2.

**Genetic Operators -** We use three operators:

- Crossover - Combine all the allocations from both parents into a single randomly sorted list. Proceed through this list including each allocation in the child chromosome if adding it does not violate any constraints and if no allocation with the same source and destination nodes has already been added.
- Local Mutation - Randomly select one allocation in the parent and randomly choose to either increase the number of channels by one or decrease it by one. If the choice was an increase and if this violates constraints, then attempt to assign the entire reconfigurable link allocation to a different source node or destination node; if none of these produces a legal chromosome, then discard the child.
- Global Mutation - Randomly select a number between half and all-but-one of the allocations in the parent. Randomly select this number of allocations from the parent and add them to the new child. Complete the child by randomly specifying the remaining available channels using the same algorithm as the initialization procedure, described below.

**Initialization -** The initialization procedure fills the initial population with randomly generated chromosomes. To generate a random chromosome, it specifies one channel at a time until some resource (total channels, node send limits, or node receive limits) has been fully exhausted. For each new channel, it randomly selects source and destination nodes that have not yet exhausted their send and receive limits, respectively. If there is an existing reconfigurable link allocation between the pair of nodes, it adds an additional channel to that allocation; otherwise, it creates a new allocation between the nodes containing one channel.

**Evaluation Function -** We have modified NS, version 2 [21], a packet-level network simulator, to compute the percentage of dropped packets and the average transmission delay exhibitted, on a per-flow basis, by a network during simulation. The evaluation function first converts the chromosome into a description of the represented network in the format expected by NS. It then starts the modified NS and sends NS the network data. NS performs the simulation and returns the QoS statistics. Finally, the evaluation function uses these statistics to compute the score given in Section 2.1. Note that we restart NS from scratch for every evaluation, and this creates a significant inefficiency because of the fixed startup time. (Attempts to avoid this failed due to a memory leak in the NS application.)

We use a packet-level network simulator rather than a computationally less expensive approach for a few reasons. First, it allows us to compute the QoS metrics more precisely. Second, NS can handle modeling of burstiness in the network traffic. Third, as an important future consideration, NS allows modeling of alternative routing algorithms such as multicast or adaptive routing as opposed to fixed routing tables. Since adaptive routing is another important piece of network adaptation, it is imporant to be able to handle it.

**Population Management -** The genetic algorithm uses steady-state, worst-one-out replacement. The population allows no duplicate members. Parents are selected probabilistically using *roulette-wheel* selection. Probabilities are distributed exponentially based upon rank. The search terminates when the number of evaluations reaches a threshold.

## 3 EXPERIMENTAL RESULTS

We had two goals for our experiments and two corresponding sets of experiments. The first was to provide concrete examples of the types of problems that adaptive network reconfiguration can solve. The second goal was to investigate the performance of the genetic algorithm, particularly concentrating on scaling with problem size and the tradeoff between the execution time and the quality of the solution found.

We approximate the search space size as

$$(M(M-1))^N/N! \tag{1}$$

where $M$ is the number of nodes and $N$ is the maximum number of channels. There are $M(M-1)$ possible ways to assign a source and destination node to each channel, and hence $(M(M-1))^N$ ways to assign sources and destinations to each of $N$ channels. However, the networks formed are not unique. For any network with no two channels sharing the same source and destination, there are $N!$ different ways to form this network; for other networks, there are less. Hence, Equation 1 is an underestimate but is a good approximation when $N \leq M/2$.

All of our timing results were performed on a single 850-MHz Pentium. All times are divided into two components: the number of total evaluations, which measures the effectiveness of the genetic algorithm search, and the average time per evaluation, which measures the efficiency of the evaluation function.

In our experiments, all fixed links have a capacity of

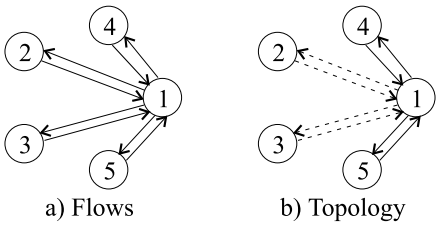


Figure 1: The first network in the sequence. Note that the dotted lines in the topology are the reconfigurable links and the solid lines are the fixed links.

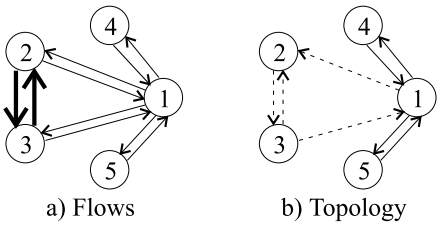


Figure 2: The second network in the sequence. Note that the heavier lines in the flows indicate higher priority traffic.

1000 kbits/sec (except in the random network experiment) and transmission delay of 10 msecs. Likewise, all channels have a capacity of 1000 kbits/sec and transmission delay of 10 msecs. (We chose these numbers to simplify theoretical analysis of the reconfiguration problems, and they are not based on the physical properties of any actual network links.)

### 3.1 ILLUSTRATIVE EXAMPLES

**A Sample Adaptation Sequence -** We start by examining a sequence of networks that could be snapshots of a single network as its operating conditions change with time. They illustrate, in a simple-to-understand scenario, the power of adaptive network reconfiguration.

There are three networks in the sequence, each with a maximum total of four channels. The first network in the sequence has five nodes. The traffic flows, pictured in Figure 1a, are typical of a server (node 1) with multiple clients (nodes 2-5). The clients communicate only with the server and not with each other. The server sends 400 kbits/sec to each client, while each client sends 40 kbits/sec to the server, all using the TCP protocol. The priorities are all 5, and the required latency is 10 msecs (since it is using TCP, dropped packets are not a criterion). There exist bidirectional fixed links between nodes 1 and 4 and between nodes 1 and 5. Each node has a limit of 3 send channels and 3 receive channels.

Clearly, the best solution is the one shown in Figure 1b, with four reconfigurable links, each containing 1 channel, that effectively form bidirectional links between nodes 1 and 4 and between nodes 1 and 5. The solution thus provides a single-hop path for all flows.

The second network in the sequence is the same as the first except for the addition of two high-priority (priority 1) flows, one from node 2 to node 3 and the other from node 3 to node 2 (e.g., a teleconference between two CEOs, or communication between two units in battle). The flows are shown in Figure 2a. The new optimal configuration is that shown in Figure 2b, since it provides full connectivity, a one-hop connection for all high priority flows, and a maximum delay of two hops for the lower priority flows.

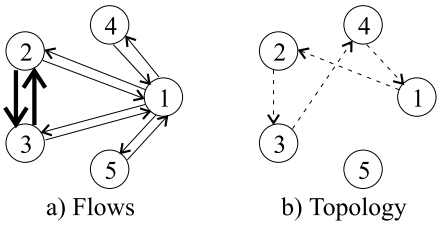


Figure 3: The third network in the sequence.

The third network in the sequence is shown in Figure 3a and is the same as the second except that all the fixed links have been disabled (e.g., due to a coordinated cyberattack). It is now impossible to fully connect all the nodes (4 unidirectional links can connect at most 4 nodes), so there is a choice about which node to leave out of the network. The best configuration is to use the channels to form a ring network between the four of the nodes, three of which must be nodes 1, 2 and 3. An example of such a network is shown in Figure 3b.

The genetic algorithm finds all solutions always in well under 500 evaluations. To perform 500 evaluations requires 4 minutes, or 0.48 secs per evaluation. Almost all of this time (between 0.4 and 0.45 seconds) is spent restarting NS; by eliminating this restart, we could get the runtime down to under 30 seconds.

**Bottleneck Networks -** We consider two more example networks, larger than the previous ones. Both have paths of fixed links with sufficient capacity to handle the traffic for any individual flow, but there exists a bandwidth bottleneck when considering the flows in aggregate. Reconfigurable links are used to relieve the bottleneck.

The first "bottleneck" network is shown in Figure 4. Each of nodes 1-5 sends data to each of nodes 6-10. All 25 flows are identical: transmitting an average of

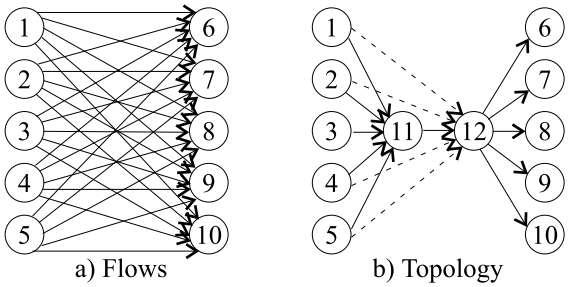


Figure 4: The first bottleneck network.

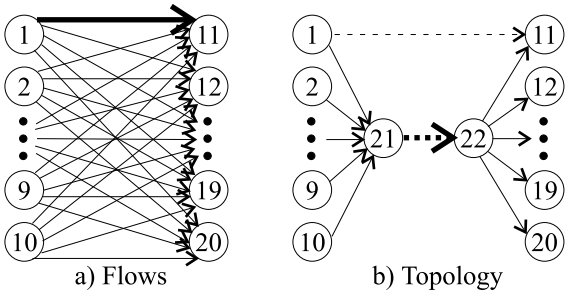


Figure 5: The second bottleneck network. Note that five channels form a single high-capacity link between nodes 21 and 22, replacing the original lower-capacity fixed link.

200 kbits/sec, using the UDP protocol, having priority 2, and requiring 0% dropped packets (with no requirement on latency). There are four available channels. Without the benefit of the reconfigurable links, all 5000 kbits/sec of the aggregate traffic would travel across the central link between nodes 11 and 12 (which has capacity of only 1000 kbits/sec). An optimal solution is shown in Figure 4, using the reconfigurable links (dotted lines) to relieve this bottleneck by bypassing the central link. (There are five equivalent solutions.)

The second bottleneck network is shown in Figure 5. Each of nodes 1-10 sends data to each of nodes 11-20. All but one of the 100 flows are identical: sending an average of 50 kbits/sec, using the UDP protocol, having priority 100 and required dropped packets 0% (with no requirement on latency). The flow between nodes 1 and 11 differs from the other flows in that it has priority 1, which is much higher than the others, and that it has a required latency of 10 msecs. There are six available channels. As with the first bottleneck network, without reconfigurable links, all 5000 kbits/sec of traffic would travel across the central link between nodes 21 and 22 (which has capacity of only 1000 kbits/sec). The solution is pictured in Figure-5b: use five channels to relieve the bottleneck by replacing the central link with a higher-capacity reconfigurable link, and use the sixth channel to directly connect nodes 1 and 11 and thereby provide the required latency. Note that in going from the first to the second network the optimal strategy changes from bypassing the central link to building up the central link.

The genetic algorithm consistently finds an optimal solution to the first bottleneck problem in under 1000 evaluations. These 1000 evaluations required 20.5 minutes, an average of 1.23 seconds per evaluation. According to Equation 1, the search space size is $1.3\text{x}10^7$. The genetic algorithm consistently finds the solution to the second problem in under 10,000 evaluations, requiring 394 minutes (2.36 seconds per evaluation). The search space size is $1.4\text{x}10^{13}$.

### 3.2 PERFORMANCE INVESTIGATIONS

We investigate the performance of the genetic algorithm on families of networks, where all the networks in a family have the same basic statistical properties but different sizes. This permits us to investigate the scaling properties of the genetic algorithm as a function of the size of the network. We used five different families, one family of "ring" networks plus four families of random networks.

For each network used to explore performance, we performed the same set of experiments, running the genetic algorithm ten times with each of the following sets of parameters:

- $popsize = 20$, $probdecay = 0.7$, $maxevals = 100$
- $popsize = 40$, $probdecay = 0.8$, $maxevals = 300$
- $popsize = 100$, $probdecay = 0.9$, $maxevals = 1000$
- $popsize = 300$, $probdecay = 0.967$, $maxevals = 3000$
- $popsize = 1000$, $probdecay = 0.99$, $maxevals = 10000$

Here, *popsize* is the population size, *probdecay* is the parameter that determines the exponential distribution of parent selection probabilities, and *maxevals* is the number of evaluations before terminating the run. A small population size and high selection pressure (small *probdecay*) mean that the genetic algorithm will converge (through loss of diversity) quickly, and are hence appropriate for a short run.

**Ring Networks -** We start by examining performance on a family of highly contrived networks, which we call "ring" networks. This family of networks has three important properties. First, it contains networks with arbitrarily large and small numbers of nodes and available channels, hence allowing an investigation of how the algorithm scales with network size. Second, each network has a known best solution and hence allows comparison with this known optimum. Third, the optimization problems are especially difficult for a genetic algorithm and hence provide worst-case scenarios.

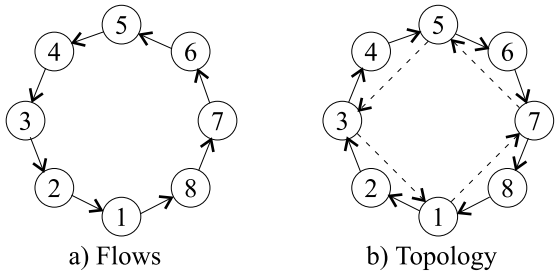


Figure 6: The 8-node, 4-channel ring network.

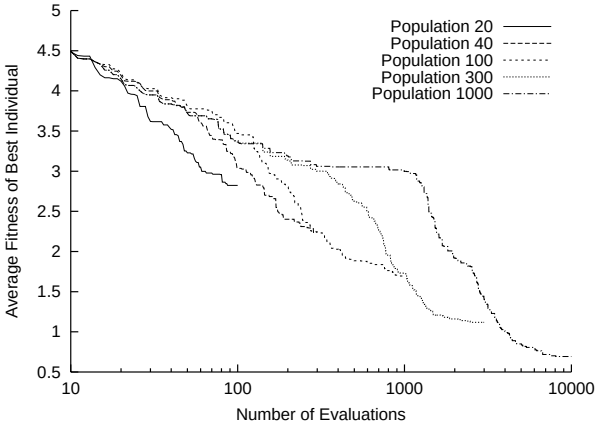


Figure 7: Average progress of the genetic algorithm for the five different parameter sets for ring network 16/8.

A member of this family has $N$ channels and a network with $M = kN$ nodes, where $k$ and $N$ are positive integers. There are $M$ identical flows, with one flow from node $i$ to node $(i-1)$ for each $i = 2, ..., M$ and one flow from node 1 to node $M$. Each flow uses the TCP protocol, has a required latency of 10 msecs, and transmits $\frac{800}{k}$ kbits/sec. There are $M$ fixed links, one from node $(i-1)$ to node $i$ for each $i = 2, ..., M$ and one from node $M$ to node 1. The fixed topology requires packets to travel $(M-1)$ hops. An optimal placement of reconfigurable links connects every $k^{th}$ node in reverse order from the fixed links and reduces the number of hops to $k$. Figure 6 shows this network when $M = 8$ and $N = 4$, along with an optimal solution. (The other optimal solution is obtained by rotating each reconfigurable link one node clockwise.)

This problem is very difficult for a genetic algorithm because of the existence of multiple completely distinct solutions (i.e. solutions that have no reconfigurable link in common). The genetic algorithm has trouble keeping the building blocks from these different solutions separate. This is generally not a problem in less contrived networks.

We ran experiments on ring networks with six different

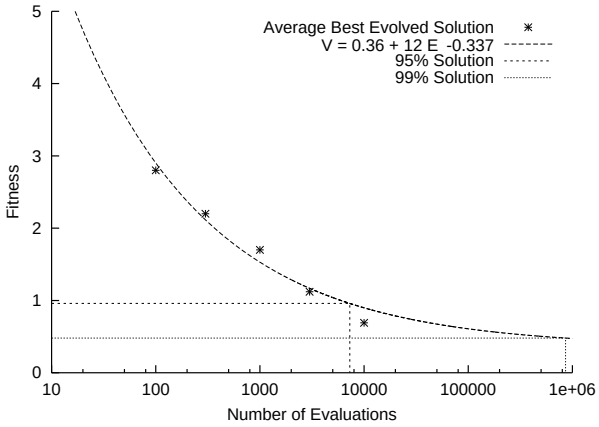


Figure 8: Modeling the tradeoff between solution quality and number of evaluations for the 16/8 ring network.

node/channel (i.e., M/N) configurations: 8/4, 12/6, 16/8, 20/5, 20/10, and 40/10.

For the 16/8 network and for each of the genetic algorithm parameter sets, Figure 7 shows the progress of a run (averaged over ten independent runs) plotted as the value of the best individual versus the number of evaluations (i.e., the number of configurations tried so far). Note how the smaller population with greater selection pressure starts out better but quickly stops making progress due to convergence. A larger population and smaller selection pressure requires longer to converge but eventually does better by exploring more of the space.

Table 1 provides a summary of the results for the different ring networks. Each row contains the results from one network. Column 1 contains the network name, and column 2 has the search space size as given by Equation 1. Columns 3-7 contain for each of the five parameter sets the value of the best individual at the end of a run averaged over the ten runs. The values in columns 3-7 provide five data points for the map between the number of evaluations performed and the quality of the solution.

As more evaluations are performed, the expected value of the best solution asymptotically approaches the optimal value. This leads us to a model for this relationship of the form

$$V = A + BE^{-C} \tag{2}$$

where $V$ is the expected value of the best individual, $E$ is the number of evaluations, and $A$, $B$ and $C$ are constants determined by the data. The constant $A$ is the value of the best possible solution, which is known for the ring networks. We use the five data points to do a least-squares regression analysis to find $B$ and $C$.

| Network | Search Space | 100 Evals | 300 Evals | 1000 Evals | 3000 Evals | 10000 Evals | A | B | C | $E_{95}$ | $E_{99}$ | Secs/ Eval |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 8/4 | 4.1E5 | .75 | .27 | .20 | .18 | .18 | .18 | 1100 | 1.65 | 6 | 16 | 0.69 |
| 12/6 | 7.3E9 | 1.90 | 1.14 | .80 | .48 | .31 | .27 | 22 | .559 | 210 | 3800 | 0.93 |
| 20/5 | 6.6E10 | 5.01 | 3.30 | 1.97 | 1.48 | 1.27 | .97 | 57 | .572 | 190 | 3100 | 1.28 |
| 16/8 | 2.7E14 | 2.8 | 2.2 | 1.70 | 1.12 | .69 | .36 | 12 | .337 | 7300 | 8.6E5 | 1.24 |
| 20/10 | 1.7E19 | 4.5 | 3.3 | 2.6 | 1.88 | 1.15 | .44 | 17 | .315 | 1.3E4 | 2.2E6 | 1.55 |
| 40/10 | 2.4E25 | 13.2 | 11.0 | 7.5 | 5.6 | 3.8 | 1.9 | 54 | .332 | 8300 | 1.1E6 | 3.20 |

Table 1: Results for the ring networks

We report $A$, $B$ and $C$ for each network in columns 8-10 of Table 1. Figure 8 shows an example graph of this curve for the 16/8 ring network.

The constant $C$ measures on average how quickly the search approaches the optimal solution. After $E$ evaluations, the search has roughly proceeded $1 - E^{-C}$ of the way from a random solution to the best solution. To find a solution that is a fraction $f$ of the way to the optimal solution therefore requires roughly $(1-f)^{\frac{-1}{C}}$ evaluations. In columns 11 and 12 of Table 1, we report the number of evaluations required to achieve 95% and 99% of the optimal solution, given by

$$E_{95} = 20^{\frac{1}{C}}, E_{99} = 100^{\frac{1}{C}} \quad (3)$$

Figure 8 shows these values for the 16/8 ring network.

**Random Networks -** We next examine optimization performance on a set of randomly generated networks. While ring networks provide a worst-case optimization problem, we also would like results for more typical networks. While we do not know the best solution for these networks a priori, we can still use a regression analysis similar to (although less accurate than) that used for the ring networks to estimate how performance varies with the number of evaluations.

Given a specified number of (i) nodes, (ii) available channels, (iii) bidirectional fixed links, and (iv) traffic flows, our software randomly generates a network with these dimensions. The random components include: (i) the fixed topology, (ii) the fixed link capacities (1000, 2000 or 3000 kbits/sec), and (iii) the source, destination, priority (1, 10 or 100), protocol (UDP or TCP), and bandwidth (100, 400 or 1000 kbits/sec) of each flow. Required latency and dropped packets were always 0.

For the experiments, we have used families of six networks. For each family, the number of nodes ($M$) and reconfigurable channels ($N$) are the same six pairs of values as for the ring networks, hence permitting comparisons of optimization performance between networks with the same search space size. The number of fixed links is $pM$ and number of flows is $qM$,

| Net | A | B | C | $E_{95}$ | $E_{99}$ | S/E |
|---|---|---|---|---|---|---|
| 8/4 | .064 | .91 | .697 | 74 | 740 | 0.69 |
| 12/6 | .038 | 150 | 1.63 | 6 | 17 | 0.90 |
| 20/5 | .201 | 2.8 | .599 | 150 | 2200 | 1.43 |
| 16/8 | .078 | 5.7 | .821 | 38 | 270 | 1.10 |
| 20/10 | .092 | 2.5 | .523 | 300 | 6700 | 1.49 |
| 40/10 | .30 | 11 | .512 | 350 | 8100 | 3.50 |

Table 2: Results for sparse/light random networks

| Net | A | B | C | $E_{95}$ | $E_{99}$ | S/E |
|---|---|---|---|---|---|---|
| 8/4 | .102 | 13 | 1.10 | 15 | 66 | 1.49 |
| 12/6 | 1.53 | 11 | .381 | 2600 | 1.8E5 | 1.86 |
| 20/5 | 2.74 | 7.5 | .364 | 3800 | 3.1E5 | 3.25 |
| 16/8 | .99 | 8.6 | .434 | 990 | 4.1E4 | 2.70 |
| 20/10 | 2.07 | 12 | .440 | 910 | 3.5E4 | 3.34 |
| 40/10 | 6.35 | 21 | .330 | 8800 | 1.1E6 | 7.04 |

Table 3: Results for sparse/heavy random networks

where $q$ and $p$ are constant for a family. We have used $q = 1$ and $q = 2$, referred to as "sparse" and "dense" respectively, and $p = 1$ and $p = 4$, referred to as "light" and "heavy" respectively, leading to four families of random networks: sparse/light, sparse/heavy, dense/light, and dense/heavy. For each of these families of random networks, we have done the same experiments and analysis as for the ring networks, except that we do not know apriori the optimal solution and hence the value to use for the A term. We instead estimate the optimal solution as the best solution found in any of the ten runs for any of the genetic algorithm parameters. The results are shown in Tables 2-5.

**Analysis of Results -** The central question is whether the optimization algorithm will support online adaptation by producing good enough configurations fast enough. While there is no clear threshold defining good enough or fast enough, we take 95% of the optimal solution in ten minutes to be our standard.

For small networks ($\leq 20$ nodes and $\leq 5$ channels), the optimization algorithm will support online adaptation.

| Net | A | B | C | $E_{95}$ | $E_{99}$ | S/E |
|---|---|---|---|---|---|---|
| 8/4 | .045 | 1.2 | .520 | 320 | 7000 | 0.75 |
| 12/6 | .024 | .65 | .760 | 52 | 430 | 1.02 |
| 20/5 | .089 | 1.2 | .640 | 110 | 1300 | 1.76 |
| 16/8 | .049 | 1.6 | .773 | 48 | 390 | 1.35 |
| 20/10 | .066 | 1.0 | .540 | 260 | 5100 | 1.77 |
| 40/10 | .159 | 1.9 | .382 | 2500 | 1.7E5 | 4.82 |

Table 4: Results for dense/light random networks

| Net | A | B | C | $E_{95}$ | $E_{99}$ | S/E |
|---|---|---|---|---|---|---|
| 8/4 | .103 | 150 | 1.48 | 8 | 22 | 1.62 |
| 12/6 | .55 | 120 | 1.16 | 13 | 53 | 2.19 |
| 20/5 | 1.04 | 23.6 | .823 | 38 | 270 | 4.16 |
| 16/8 | .33 | 4.1 | .576 | 180 | 3000 | 3.14 |
| 20/10 | 0.67 | 3.7 | .368 | 3400 | 2.7E5 | 4.04 |
| 40/10 | 1.67 | 42.3 | .696 | 74 | 750 | 9.59 |

Table 5: Results for dense/heavy random networks

It will consistently find the 95% solution in under 10 minutes. Once we fix the NS restart problem with the evaluations, it will do even better, potentially reaching the 98% or 99% solution in the given time.

For mid-sized networks ($\leq$ 40 nodes and $\leq$ 10 channels), the optimization algorithm will be sufficient for online adaptation only with the help of additional hardware to speed the optimization. Genetic algorithms are inherently parallelizable, with a near linear speedup as a function of the number of processors up to a large number of processors [5]. Assuming a 100-processor cluster providing a factor of 100 speedup, all of the reported networks would reach their 95% solution within ten minutes.

For larger networks, the highly superlinear (potentially exponential) scaling of the algorithm means that more hardware will not address the scaling problem. Instead, fundamental improvements to the algorithm are required.

One potential source of improvements to the genetic algorithm is to use the fact that network adaptation is a continuous process. A solution that was good a few minutes earlier is still most likely a good solution. Particularly for big networks, the current optimal configuration is likely only at most a small perturbation from the previously optimal configuration. By including in the initial population one or more individuals based on the previous best configuration, the genetic algorithm gets a big head start and can find a good solution in far less time [19]. While preliminary tests indicate that this approach does in fact speed up the search, we have not yet verified this with systematic testing.

Another approach to improving the genetic algorithm is to incorporate heuristics into the algorithm. These heuristics can be used both when generating the initial population and as part of the genetic operators, and will often improve the search by a large amount [9]. For example, we could define a mutation operator that first determines which links are overloaded as well as which reconfigurable links are underloaded and then probabilistically chooses an underloaded and reassigns is source and destination nodes so that it offloads an overloaded link.

A second question is how search time varies with the network. The size of the search space is the single biggest factor influencing search difficulty. It grows very quickly with the number of nodes and channels, resulting in a rapid growth in the number of evaluations required to find a good solution. (This growth is shown in Tables 1-5 as a general increase in $E_{95}$ and $E_{99}$ with search space size.) However, when we examine the random networks, we see that there are some networks with large search spaces that are much easier to solve than others with smaller seach spaces (e.g., see Table 4). Also, the random networks with heavy traffic tend to be more difficult to solve than networks with the same size search space but with light traffic. This is likely because more flows mean more tradeoffs and hence more difficult decisions. However, as both the ring and random networks show, even networks with light traffic can present difficulties.

## 4 CONCLUSION

We have introduced an important, little-investigated problem, that of determining at any given time the optimal configuration of a reconfigurable data network. Finding a good configuration is a critical part of the process of adaptively reconfiguring a network online. Adaptive network reconfiguration offers the benefits of survivability in the face of major changes in network operating conditions and performance fine-tuning in response to minor changes in operating conditions.

We have developed an algorithm for solving the problem using a genetic algorithm. In its current form, it is too slow for online adaptation. However, the simple step of distributing the evaluations of the genetic algorithm across many machines would make it fast enough for small and mid-sized networks. Improvements to the core algorithms of the genetic algorithm could potentially make it fast enough for networks with large numbers of nodes and reconfigurable links.

We have also created a set of benchmarks that are

interesting and scalable and that have analytical solutions. While we do not provide a comparison with other algorithms, these benchmarks provide a good way for other researchers utilizing other algorithms to compare their results against ours.

Future work should focus on making adaptive network reconfiguration a reality rather than just a possibility. This work has two main components. The first is speeding the optimization. There are major benefits to be gained by

- parallelizing the genetic algorithm and using many machines over which to distribute the computation
- making the evaluation function more efficient
- hybridizing the genetic algorithm with network design heuristics
- using results from the previous optimization to seed the current one

The second major component of future work is integration with actual networks. This integration requires a mechanism for collecting data on operating conditions, most importantly traffic flows, in "real time". The flow data includes both traffic statistics and required quality of service. The integration also requires some (potentially manual) mechanism to perform the reconfiguration indicated by the optimization algorithm.

## Acknowledgments

This work was supported by DARPA contract N66001-00-C-8042 as part of the SWWIM program. Thanks to Tushar Saxena for developing the interface with the NS simulator. Thanks to Ken Theriault and John Lowry for their guidance and for their recommending this problem.

## References

[1] Abuali, A., R. Wainwright, and D. Schoenfeld: 1995, 'Determinant Factorization: A new encoding scheme for spanning trees applied to the probabilistic minimum spanning tree problem'. *Proceedings of the Fifth International Conference on Genetic Algorithms*. pp. 470–477.

[2] Ash, G.: 1995, 'Dynamic network evolution, with examples from AT&T's evolving dynamic network'. *IEEE Communications Magazine* **33**(7), 26–39.

[3] Ayanoglu, E., C. Gitlin, and J. Mazo: 1993, 'Diversity coding for transparent self-healing and fault-tolerant communication networks'. *IEEE Transactions on Communications* **41**(11), 1677–1686.

[4] Butler, K., N. Sarma, and V. Prasad: 2001, 'Network Reconfiguration for Service Restoration in Shipboard Power Distribution Systems'. *IEEE Transactions on Power Systems* **16**(4), 653–661.

[5] Cantu-Paz, E.: 2000, *Efficient and Accurate Parallel Genetic Algorithms*. Kluwer.

[6] Chou, H., G. Premkumar, and C.-H. Chu: 2001, 'Genetic Algorithms for Communications Network Design - An Empirical Study of the Factors that Influence Performance'. *IEEE Transactions on Evolutionary Computation* **5**(3), 236–249.

[7] Coombs, S. and L. Davis: 1987, 'Genetic algorithms and communication link speed design: constraints and operators'. *Proceedings of the Second International Conference on Genetic Algorithms*. pp. 257–260.

[8] Curcic, S., C. Ozveren, L. Crowe, and P. Lo: 1996, 'Electric power distribution network restoration: A survey of papers and a review of the restoration problem'. *Electric Power Systems Research* **35**, 73–86.

[9] Davis, L.: 1991, *Handbook of Genetic Algorithms*. Van Nostrand Reinhold.

[10] Dengiz, B., F. Altiparmak, and A. Smith: 1997, 'Local Search Genetic Algorithm for Optimal Design of Reliable Networks'. *IEEE Transactions on Evolutionary Computation* **1**(3), 179–188.

[11] Elbaum, R. and M. Sidi: 1996, 'Topological design of local-area networks using genetic algorithms'. *IEEE/ACM Transactions on Networking* **4**(5), 766–778.

[12] Gibbs, W.: 2002, 'Autonomic computing'. *Scientific American*.

[13] Huang, R., J. Ma, and D. Hsu: 1997, 'A genetic algorithm for optimal 3-connected telecommunication network designs'. *Proceedings of the Third International Symposium on Parallel Architectures, Algorithms and Networks*. pp. 344–350.

[14] Kershenbaum, A.: 1993, *Telecommunications Network Design Algorithms*. McGraw-Hill.

[15] Ko, K.-T., K.-S. Tang, C.-Y. Chan, K.-F. Man, and S. Kwong: 1997, 'Using genetic algorithms to design mesh networks'. *Computer* **30**(8), 56–61.

[16] Konak, A. and A. Smith: 1999, 'A hybrid genetic algorithm approach for backbone design

of communication networks'. *Proceedings of the 1999 Congress on Evolutionary Computation*. pp. 1817–1823.

[17] Kumar, A., R. Pathak, M. Gupta, and Y. Gupta: 1993, 'Genetic algorithm based approach for designing computer network topology'. *Proceedings of the 1993 ACM Conference on Computer Science*. pp. 358–365.

[18] Mitchell, M.: 1996, *An Introduction to Genetic Algorithms*. MIT Press.

[19] Montana, D., M. Brinn, S. Moore, and G. Bidwell: 1998, 'Genetic Algorithms for Complex, Real-Time Scheduling'. *Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics*. pp. 2213–2218.

[20] Montana, D., T. Hussain, and T. Saxena: 2002, 'Adaptive Reconfiguration of Data Networks Using Genetic Algorithms'. *Proceedings of the Genetic and Evolutionary Computation Conference*. pp. 1141–1149.

[21] NS: 2001, 'The network simulator - ns-2'. http://www.isi.edu/nsnam/ns/.

[22] Palmer, C. and A. Kershenbaum: 1995, 'An Approach to a Problem in Network Design Using Genetic Algorithms'. *Networks* **26**(3), 151–163.

[23] Patterson, D.: 2002, 'Recovery-oriented computing: A new research agenda for a new century'. *Proceedings of the Eight International Symposium on High-Performance Computer Architecture*.

[24] Pierre, S. and G. Legault: 1998, 'A genetic algorithm for designing distributed computer network topologies'. *IEEE Transactions on Systems, Man and Cybernetics, Part B* **28**(2), 249–258.

[25] Sayoud, H. and K. Takahashi: 2001, 'Designing communication network topologies using steady-state genetic algorithms'. *IEEE Communications Letters* **5**(3), 113–115.