Optimizing QoS-Based Multicast Routing in Wireless Networks: A Multi-objective Genetic Algorithmic Approach*

Abhishek Roy and Sajal K. Das

Center for Research in Wireless Mobility and Networking (CReWMaN)
Department of Computer Scienece and Engineering
University of Texas at Arlington
Arlington, Texas, 76019-0015, USA
{aroy,das}@cse.uta.edu

Abstract. With increasing demand for real-time services in next generation wireless networks, quality-of-service (QoS)-based routing offers significant challenges. Multimedia applications like video conferencing, real-time streaming of stock quotes or processing of scientific images relayed from satellites require strict QoS guarantee (e.g. bandwidth, delay) while communicating among multiple hosts. This gives rise to the need for an efficient *multicast routing* protocol which will be able to determine multicast routes satisfying the different QoS constraints. Design of such protocol boils down to a multi-objective optimization problem, which is computationally intractable. In fact, discovering optimal multicast routes is an NP-hard problem when the network state information is inaccurate – a common scenario in mobile wireless networks. In this paper, we propose a novel multicast tree selection algorithm that determines near-optimal multicast routes on demand. Based on the multi-objective genetic algorithmic (MOGA) approach, our solution attempts to optimize multiple QoS parameters (e.g. end-to-end delay, bandwidth guarantee and residual bandwidth utilization) simultaneously. We mathematically analyze the performance and convergence of the developed algorithm. Simulation results demonstrate that our algorithm is capable of discovering on-demand a set of QoS-based, near-optimal multicast routes within a few iterations, even with imprecise network information. From these set of routes one can choose the best possible multicast route depending on the specified QoS requirements.

1 Introduction

Multicast routing is an effective way to communicate among multiple hosts in a network. It outperforms the basic broadcast strategy by sharing resources along common links, while sending messages to a set of predefined destinations. This is particularly true in wireless networks which suffer from resource (bandwidth)

* This work was supported by NSF grants EIA-0086260, EIA-0115885, IIS-0121297 and Texas Telecommunication Engineering Consortium (TXTEC)

E. Gregori et al. (Eds.): NETWORKING 2002, LNCS 2345, pp. 28–48, 2002.

scarcity and high bit error rate (BER). Furthermore, the growing demand for real-time multimedia communications like live video conferencing or streaming of stock quotes require strict *quality-of-service* (QoS) guarantee on such parameters as bandwidth, end-to-end delay, and delay jitter. An efficient allocation of network resources satisfying QoS requirements is the primary goal of QoS-based multicast routing [19]. However, individual QoS parameters may be conflicting and interdependent, thus making the problem even more challenging [15].

Further complications arise in wireless networks due to information (e.g. resource availability) inaccuracy caused by high BER and signal fading, leading to packet loss and hence higher packet delay and jitter. This effect can be reduced at the cost of extra bandwidth allocation. Thus, there exists a trade-off between bit error rate and bandwidth for a fixed radio spectrum. If we were to optimize a multicast route path with respect to a single QoS parameter, say bandwidth, then the problem can be solved in polynomial time even with uncertain network resources [8], by mapping it to a shortest path finding problem. On the otherhand, determining multicast routes satisfying different QoS parameters or constraints simultaneously, is an NP-hard problem [13]. The uncertainly of the network resources make such a problem more difficult. Therefore, various approximate algorithms have been proposed based on some heuristics.

Although QoS-Routing in wireless networks is an active research area in recent years, QoS-based multicasting is relatively a new research topic. The impact of information inaccuracy and uncertainty over QoS-routing has been investigated in [8,15] which proposes efficient heuristics to identify routes that are most likely to accommodate the desired QoS even with uncertain network state information. Using suitable probabilistic models it is shown that uncertainty is minimal for flows with only bandwidth requirements, but it makes path selections intractable when end-to-end delay is considered. A scalable, coarse-grained approach to control the mobile QoS is highlighted in [12]. The key technique used here is to aggregate a cluster of cells into a *Virtual Bottleneck cell* (VBC) in such a way that by controlling the parameters of VBC, specific QoS objectives of the system can be ensured without requiring the accurate prediction of the times and locations of each mobile user. The 3-level multi-agent architecture for QoS control in wireless ATM [11] provides a self-regulating network congestion control management by means of global network state awareness. A dynamic reconfiguration of the agents and an adaptive cell discarding scheme are performed to meet the end-to-end QoS requirements. The agents efficiently manage the buffer space to reduce the cell loss ratio while guaranteeing a bounded transit delay. In a completely different approach [16] multimedia streams are represented in terms of multiple substreams each with its own specified QoS and wireless network elements and protocols are made aware of the QoS requirements of such substreams. With the fluctuation of resource availability, using a fair scheduling algorithm the network selects and schedules substreams in order to meet an acceptable QoS. For effective multicast tree construction in interactive audiovisual communication, a heuristic has been proposed in [14] to compute low cost, delay-bound routes from source to each destination. Recently, the authors in

[1] demonstrated the efficiency of genetic-algorithm (GA) to obtain QoS-based multicast routes in computationally feasible time. With the help of evolutionary operations, the proposed algorithm is capable of optimizing multiple QoS parameters to generate a near-optimal multicast tree.

A careful analysis of the optimization schemes explored in QoS-routing in wireless as well as wirelined networks reveal that most of them suffer from the same drawback: multiple objectives are combined to form a *scalar single-objective* function on an ad hoc basis, usually through a linear combination (weighted sum) of multiple attributes. In these cases the solution not only becomes highly sensitive to the weight vector but also demands the user to have certain knowledge (e.g. priority of a particular objective, influence of an objective parameter over other) about the problem. Moreover, in the case of multi-objective optimization, a unique solution that optimizes all the objectives simultaneously will rarely, if at all, exist in practice. The user will therefore be more interested in obtaining a set of acceptable *non-dominated* solutions, one of which can be selected based on the specific problem requirements. We recognize that genetic algorithms can be readily modified to deal with multiple objectives by incorporating the concept of *Pareto-domination* (discussed in Section 2) in its selection operation [6].

In this paper, we use a *multi-objective genetic algorithm* (MOGA) technique to develop an efficient algorithm which determines multicast routes on-demand by simultaneously optimizing end-to-end delay guarantee, bandwidth requirements and residual bandwidth utilization without combining them into a single scalar objective function. Using suitable genetic operators, the algorithm is capable of finding near-optimal solutions within a few iterations. We have shown that with the increase in the number of nodes our algorithm performs better than existing algorithms based on scalar optimization. Although, it is impossible to provide a tight-bound for convergence of such an NP-hard algorithm, we have shown that asymptotically it can converge to the optimal point. From the experimental results it is clear that our algorithm is capable of obtaining more than 95% of the global optimal values for all three QoS parameters.

Section 2 reviews the basic concept of MOGA relevant in this context. The formulation of the required optimization functions and the proposed new algorithm are presented in Section 3. Section 4 highlights the power of the algorithm by analyzing the variation of some genetic operators and demonstrating its asymptotic convergence. In an attempt to evaluate the performance and the of the algorithm, a suitable model is developed and steady-state probabilities are calculated in Section 5. Simulation results in Section 6 corroborates the fast optimization of the required QoS parameters. Section 7 concludes the paper with pointers to the areas of future work.

2 Evolutionary Algorithms in Multi-objective Optimizations

Genetic algorithms (GA) provides a *guided random* search and optimization technique, based on the basic principles of *evolution: survival of the fittest* and

inheritance [7]. It uses *probabilistic transition rules* and a *payoff* function to guide the search. All generalized greedy and gradient descent search techniques suffer from getting stuck at a *local optimal* point. However, using the evolutionary techniques, GAs can overcome this limitation to provide a *near-optimal* solution in a few iterations. The steps involved in solving an optimization problem using GA can be briefly summarized as follows: (i) Random generation of a *population* of *chromosomes*, (ii) Decoding each chromosome to evaluate its *fitness*, (iii) Performing *selection, cross-over* and *mutation* operations, (iv) Repeating steps (ii) and (iii) until a stopping criterion is satisfied. To solve any optimization problem, GAs start with *chromosomal representation* of the parameter set. A set of such chromosomes or strings are termed as *population*. The *fitness/objective function* is chosen in such a way that the good points in the search space possess high fitness values. This is the so-called *payoff* information used by GAs. In short, GAs mimic the natural evolution process through its selection, cross-over and mutation operations as discussed below:

- **Selection**: The selection process copies parent strings into a tentative new population known as *mating pool*. Selection is usually proportional to an individual's fitness value and thus mimics the evolutionary selection process. *Roulette wheel* selection, *stochastic universal* selection and *tournament* based selections are the most widely used techniques [4].
- **Cross-over**: The key idea behind the cross-over is to exchange information between two randomly selected parent-strings to give birth to the offsprings for the next generation. The selected strings from the mating pool are paired at random and a particular *cross-over point* is selected uniformly at random between position 1 and the string-length. The offsprings are generated by swapping the respective portions of the strings after the cross-over point.
- **Mutation**: Mutation is the process of *random alteration* in the genetic structure to introduce *genetic diversity*. In adverse situation, when the global optimal solution resides in a particular portion of the search space not included in the population, then the mutation is the only way to direct the population to *jump out* from any *local optimal* solution by randomly altering the information in the string.

In addition to these basic concepts, generally the best string up to a particular generation is preserved in a location either within the population or outside it. This idea is known as *elitism* [7]. We are now in a position to digress into its multi-objective counterpart of GAs.

A careful look into many real world problems reveals the requirement of simultaneous optimizations of multiple objectives. In principle, multi-objective optimization is quite different from the single-objective optimization. In case of multiple objectives, there may not exist a single best solution with respect to all the objectives. In fact, there exists a set of solutions superior to the rest of the solutions in the entire search space when all objectives are considered. These solutions are termed as *Pareto-optimal solutions*. Since none of the solutions in this set is *absolutely* better than any other, any one of them will be an acceptable solution. Hence, the user is given the freedom to choose the best solution

from this set of Pareto-optimal solutions, defined below, to conform to specific requirements.

Pareto-optimal Front: This concept of *Pareto-optimality*, originally formulated by V. Pareto in the 19th century and constitutes by the origin of research in multi-objective optimization. We can say that a point x is *Pareto optimal* if for every x either, $\cap_i(f_i(x) = f_i(x^*))$ or, there is at least one i such that $f_i(x) > f_i(x^*)$, $\forall i \in \mathbf{I}$, where $f_i(x)$ is the *fitness function*. In other words, x^* is Pareto optimal if there exists no feasible vector x which would decrease some criterion without causing a simultaneous increase in at least one other criterion.

The *multi-objective genetic algorithm* (MOGA) varies from the ordinary GAs only in its selection operator. Before the selection is performed, using some suitable ranking schemes, the population is ranked on the basis of individual chromosome's or string's *non-domination*. The non-dominated strings from the current population are first identified to form the first *Pareto-optimal* front [5]. As MOGA iterates in every generation, the non-dominated, *Pareto-optimal* solutions are found and genetic operations are performed on these solution-sets to improve their fitness values. The non-dominated solution sets quickly proceeds towards the global optimal solution and gets saturated at a near-optimal solution-set. However, the tournament selection method used for ranking schemes can lead to a tie between two or more strings which is resolved by *Niche sharing* discussed below.

Niche Sharing on Non-dominated Frontier: Fitness sharing has already been applied to a number of real world problems. Given an optimization function having several peaks, the goal of fitness sharing is to distribute the population over the different peaks in the search space, where each peak receives a fraction of the entire population according to its height. The easiest way to achieve such fitness-sharing is to degrade an individual's *fitness*, f_i, by dividing it by a *niche count*, m_i, for that individual. The intuition behind the niche count is that it is a good estimate about how *crowded* the *neighborhood* of a particular individual i is [9], [20].

With these discussions we will now proceed to develop the multicast routing algorithm required for our protocol.

3 QoS-Based Multicast Routing Algorithm

The primary goal behind designing this algorithm is to find optimal multicast routes satisfying the necessary (QoS) parameters. Let us first discuss the different objective functions that the algorithm should try to optimize.

3.1 Objective Functions

Since wireless networks often suffer from uncertainty of resources, we design the algorithm in such a way that it can determine the multicast routes by probabilistically satisfying three major objective parameters: (i) end-to-end delay requirement, (ii) bandwidth guarantee and (iii) residual bandwidth utilization.

We represent the network by a graph $G = (V, E)$ where V is the set of nodes and E is the set of edges between the node-pairs. A path between a source (v_s) and a particular destination (v_d) is represented by a sequence of nodes $v_s, v_1, v_2, v_3, ..., v_d$ where $v_i \in$ V. There can be multiple such paths between a given pair of source and destination. For *unicast* routing the problem is to find the most efficient path between such a given pair of source and destination satisfying the required QoS constraints. However, in *multicast* routings, our focus is to find such paths between a single source and multiple destinations,which will simultaneously satisfy the above QoS parameters. These multicast paths essentially forms a *multicast tree* and we have multiple such trees.

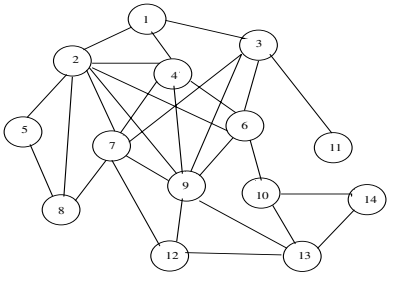

Fig. 1. A graph representing network

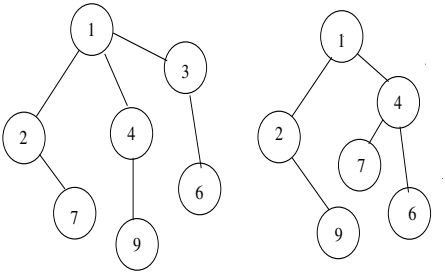

Fig. 2. Two different valid multicast trees

Figure 2 shows two possible multicast trees for finding routes from the source node 1 to destination nodes 6, 7, 9 for the input network of Figure 1. But, not all these paths can meet the desired QoS requirement. Our algorithm will look for the *set of non-dominated paths* that will satisfy the three different QoS parameters, namely end-to-end delay, bandwidth guarantee and residual bandwidth utilization. We assume the network to satisfy the following properties:

– The links are assumed to be *service queues* where packets are transmitted and get serviced. The *service rate* is assumed to follow *Poisson distribution* which makes the service time to obey *Exponential distribution*. The link delays introduced due to service time, should also follow an *Exponential distribution* with parameter equal to λ. Since the path consists of a chain of k hops, the delay along the entire path should follow *Erlang-K distribution* [17], which is the convolution of k independent random variables, each having the same exponential distribution. The probability that the delay (d_p) over a path p (from the source to one of the multicast destinations) of length k is less than t is given by: $Pr(d_p < t) = \frac{\lambda^k t^{k-1} e^{-\lambda t}}{(k-1)!}$. The probability that the delay (d) of the selected multicast tree ($\mathcal{T}$) will meet the specific delay constraint, can be obtained by taking the product of delays over individual paths in that multicast tree. This is expressed by: $Pr(d_{\mathcal{T}} < t) = \prod_{p \in \mathcal{T}} Pr(d_p < t)$. Our algorithm attempts to *maximize* this probability.

- To measure the second optimization factor, bandwidth guarantee, a similar model for the network links is assumed. Assume the *service* or *transmission rate*, a good measure of link bandwidth, follows a *Poisson distribution*. Then the probability that a link $l \in E$ is capable of providing a bandwidth of B is given by: $Pr_l(B) = \frac{\lambda^B e^{-\lambda}}{B!}$. The probability with which the bandwidth guarantee of B is satisfied for an entire multicast tree ($\mathcal{T}$) is given by: $Pr_{\mathcal{T}}(B) = \prod_{l \in \mathcal{T}} Pr_l(B)$. Our algorithm will try to *maximize* this probability also.
- Our third optimization factor is *residual bandwidth utilization*. Generally, the multicast path capable of providing *greatest residual bandwidth* is taken as the best possible choice. The total residual bandwidth in the network after allocating bandwidth for multicast is given by $\sum_{l \in E}(c_l - b_l)$, where c_l is the capacity of a link $l \in E$ and b_l is the bandwidth allocated for different hops along the multicast tree ($\mathcal{T}$). One can easily notice that $b_l = 0$ if $l \notin p$, where $p \in \mathcal{T}$. The fraction of total bandwidth available as residual bandwidth is given by: $R_b(\mathcal{T}) = \frac{\sum_{l \in \mathcal{T}}(c_l - b_l)}{\sum_{l \in \mathcal{T}} c_l}$. This measure is the third objective function that our protocol should try to maximize.
  We have also taken the *call blocking rate* as the measure of performance to compare our protocol with other existing ones. In order to determine the number of blocked calls, we first estimate the minimum available bandwidth for the multicast tree as $b^{min}_{avail} = \min_{l \in \mathcal{T}}(b^l_{avail})$, where $b^l_{avail} = c_l - b_l$ is the residual bandwidth on a network link belonging to the multicast tree $\mathcal{T}$. Any multicast session request is considered as *blocked* if its bandwidth requirement is more than b^l_{avail}.
  We now proceed to develop an efficient algorithm for on-demand QoS multicasting.

3.2 Proposed Algorithm

The underlying concept of the algorithm in Figure 3 is that it does not combine the three QoS objective functions on an ad hoc basis to form a scalar objective function, but attempts to tackle the problem from the perspectives of multi-objective optimizations. The motivation behind developing such an algorithm is to provide the user with a set of *Pareto-optimal* solutions, and give the liberty to choose the best solution from the set, depending on the specific requirements. We now discuss the implementation details of our algorithm and highlight the basic flow in Figure 4.

3.3 Implementation Details

The detailed implementation of the algorithm is discussed below.

Line 1: The *Network-generation* part of the algorithm takes the *number of nodes* as input and dynamically generates the graph using adjacency matrix representation with random connectivity.

MOGA-based Multicast-routing Algorithm

1. Generate a network (input: number of nodes) with random connectivity;
2. Obtain the initial set of multicast trees (input: source, destinations);
3. Map each of the multicast tree to a string sequentially consisting network nodes;
4. Generate the initial population by taking a specific number of such strings;
5. Repeat
6. Calculate the initial fitness values of three QoS parameters separately;
7. Generate the *comparison set* ($\mathcal{C}$) from population;
8. While (not all strings are examined)
9. Take out two strings at random;
10. Compare each of their fitness values with the strings in $\mathcal{C}$;
11. If (one string dominates the other (considering all fitness values))
12. Mark the non-dominated string;
13. End-If;
14. If (tie occurs (i.e. both the strings are dominated / non-dominated))
15. Calculate *niche count*;
16. Mark the string with lower niche count as non-dominated;
17. End-If
18. Add the non-dominated strings into *Pareto-Optimal* set ($\mathcal{S}$);
19. End-While
20. Perform cross-over and mutation operations;
21. Obtain the new set of strings to get new population;
22. Until $(\{fitness\}_{\mathcal{S}_{new}} - \{fitness\}_{\mathcal{S}_{previous}} < \epsilon)$;

Fig. 3. Multi-Objective QoS-Multicasting Algorithm

Line 2: The algorithm now takes as input the *source node* v_s and a *specific number of multicast destination nodes*, say, $v_{d_1}, v_{d_2}, ..., v_{d_n}$ and finds a set of possible multicast paths from v_s to each of $v_{d_1}, v_{d_2}, ..., v_{d_n}$, using the *depth first search* (DFS) algorithm. This gives the initial set of multicast trees. Our goal is to find the multicast trees which will satisfy the required QoS parameters. The next step is to map the problem in a search space suitable to MOGA.

Lines 3-4: Each of the generated multicast trees is mapped to a string consisting of the sequence of nodes along the path from the source v_s to each of the destinations $v_{d_1}, v_{d_2}, ..., v_{d_n}$. To mark the end of a path from a source to a single destination, we use -1 as the *sentinel*. Figure 5 gives depicts this scenario where the second multicast tree of Figure 2 is represented by a string. The set of all such initial strings constitute the *initial population*.

Line 6: The *fitness_computation* computes the values of the three pre-defined QoS parameters individually. The objective of the algorithm now boils down to a search for different multicast paths which will improve the values of these QoS parameters at each iteration.

Line 7: The key idea behind developing *Pareto-optimization* is to use a *ranking selection* method to emphasize the good points and incorporate the concept of *niching* to maintain *stable subpopulations* of good points. In order to achieve good selection, a *comparison set*, of individuals are picked at random

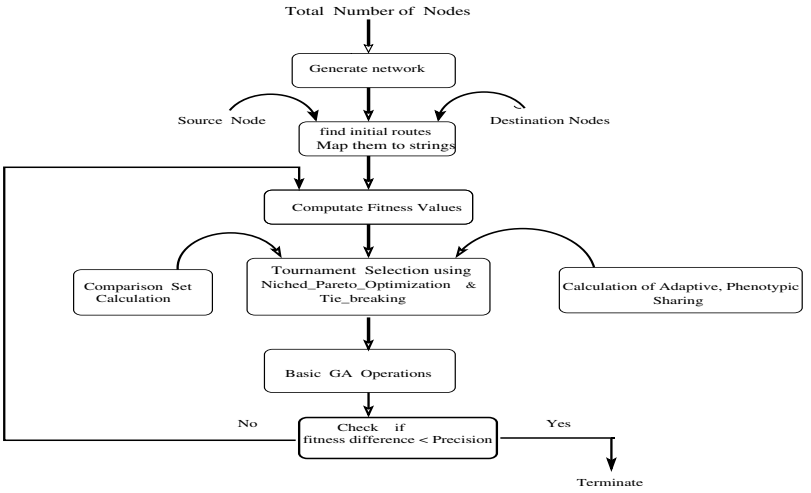

Fig. 4. Flowchart of the Algorithm

1	2	7	−1	1	4	9	−1	1	3	6

Fig. 5. String representing the first Multicast tree of Figure 2

from the population. The size of this comparison set, t_{dom}, gives us a good control over the selection pressure. If a small t_{dom} is chosen, only a few Pareto-optimal points would be found. Instead, choosing a very large t_{dom} might result into a *premature convergence*. In this algorithm we have taken $t_{dom} = 0.20 \times (\text{popsize})$.

Lines 8-16: From the population, two strings are randomly selected at a time and each of them is compared against each individual in the comparison set. If one candidate is dominated and the other is not then the latter is selected for selection. On the other hand, if both of the individuals are dominated or both non-dominated then we use *niche count* to resolve the tie. We compute the value of *niche count* for every individual string present in the population, is computed as:

$$m_i = \sum_{j=1}^{popsize} Sh[d_{s1,s2}], \tag{1}$$

where $d_{s1,s2}$is the distance between individuals $s1$ and $s2$ and $Sh[d_{s1,s2}]$ is the *sharing function*. For simplicity, triangular sharing function has been used:

$$Sh[d_{s1,s2}] = 1 - \frac{d_{s1,s2}}{\sigma_{share}} \tag{2}$$

for $d \leq \sigma_{share}$ and Sh[d] = 0 otherwise. Here σ_{share} is the *niche radius*, and it is a good estimate of *minimal separation* expected between the goal of solutions. Individuals within σ_{share} distance of each other degrade each other's fitness, as they are in the same niche. We introduce a new concept of *adaptive sharing*, i.e., the value of σ_{share} is no longer kept fixed. Depending on the fitness values of the particular string chosen and the population density in the search space, σ_{share} is dynamically updated in every iteration of the algorithm. We compute phenotypic Eucledian distance [9] between the different fitness values as a good measure of this σ_{share}.

$$d_{s1,s2} = \sqrt{(\delta_{delay_{s1,s2}})^2 + (\delta_{bw_{s1,s2}})^2 + (\delta_{bit_{s1,s2}})^2} \tag{3}$$

where $\delta_{delay_{s1,s2}} = Pr(d_{s1} < t) - Pr(d_{s2} < t)$, $\delta_{bw_{s1,s2}} = Pr_{s1}(B) - Pr_{s2}(B)$ and $\delta_{bit_{s1,s2}} = R_b(s1) - R_b(s2)$, B and R_b are the bandwidth and residual bandwidth respectively.

Similarly, we obtain the *niche radius*, σ_{share}, as some fraction (precisely half) of the maximum separation possible in the population, i.e.,

$$\sigma_{share} = \frac{\sqrt{(\delta_{delay_{max}})^2 + (\delta_{bw_{max}})^2 + (\delta_{bit_{max}})^2}}{2} \tag{4}$$

where $\delta_{delay_{max}} = Pr_{max}(d < t) - Pr_{min}(d < t)$, $\delta_{bw_{max}} = Pr_{max}(B) - Pr_{min}(B)$ and $\delta_{bit_{max}} = (R_b)_{max} - (R_b)_{min}$.

Lines 17-18: The cross-over and mutation operations are same as normal genetic algorithms. But a close look into the structure of the chromosome in Figure 5 reveals that these genetic operations can not be performed on any arbitrary gene (network nodes), as that may result in some illegal paths. Both the cross-over and mutation operations can only be performed at the end of an existing path, i.e., immediately after the particular *sentinel*, represented by -1. To give an equal probability to all such possible cross-over and mutation points, we randomly select one such point. To combine the good strings and simultaneously preserve the effective ones, we have taken the probability of cross-over as 0.7 and that of mutation as 0.1.

Loop 5-19: As the algorithm executes, at every iteration the genetic operations dynamically update the chromosomes (strings) and try to improve the corresponding probabilities until the difference of fitness values between the current Pareto-optimal set and the previous one is less than the precision ϵ.

3.4 Illustrative Example

Let us work out a small illustrative example to explain the essence of the algorithm, considering the network represented in Figure 1 with same source and destination nodes. The possible routes to nodes 6, 7 and 9 are respectively ($1 \rightarrow 2 \rightarrow 6$, $1 \rightarrow 3 \rightarrow 6$, $1 \rightarrow 4 \rightarrow 6$); ($1 \rightarrow 2 \rightarrow 7$, $1 \rightarrow 3 \rightarrow 7$, $1 \rightarrow 4 \rightarrow 7$); and ($1 \rightarrow 2 \rightarrow 9$, $1 \rightarrow 3 \rightarrow 9$, $1 \rightarrow 4 \rightarrow 9$). Thus, we have $3^3 = 27$ possible multicast

trees. We take ($1 \rightarrow 2 \rightarrow 7$, $1 \rightarrow 3 \rightarrow 6, 1 \rightarrow 4 \rightarrow 9$); ($1 \rightarrow 2 \rightarrow 9$, $1 \rightarrow 4 \rightarrow 7$, $1 \rightarrow 4 \rightarrow 6$); and ($1 \rightarrow 2 \rightarrow 6$, $1 \rightarrow 4 \rightarrow 7$, $1 \rightarrow 3 \rightarrow 9$) as our initial multicast trees which will form the initial strings in the population set. The three QoS parameters $Pr(d_{\mathcal{T}} < t)$ for end-to-end delay, $Pr_{\mathcal{T}}(B)$ for bandwidth guarantee and $R_b(\mathcal{T})$ for residual bandwidth utilization are evaluated on this set and the QoS-based fitness values obtained are shown in table 1.

Table 1. Initial Multicast Trees with QoS Parameters

Initial multicast trees	$Pr(d_{\mathcal{T}} < t)$	$Pr_{\mathcal{T}}(B)$	$R_b(\mathcal{T})$
$1 \rightarrow 3 \rightarrow 6$, $1 \rightarrow 2 \rightarrow 7$, $1 \rightarrow 4 \rightarrow 9$	0.4×10^{-3}	0.003	0.54
$1 \rightarrow 4 \rightarrow 6$, $1 \rightarrow 4 \rightarrow 7$, $1 \rightarrow 2 \rightarrow 9$	0.3×10^{-3}	0.001	$R(\mathcal{T})$=0.52
$1 \rightarrow 2 \rightarrow 6$, $1 \rightarrow 4 \rightarrow 7$, $1 \rightarrow 3 \rightarrow 9$	0.1×10^{-3}	0.004	0.46

As the initial population is too small to generate an effective comparison set, probabilistically both string-1 and string-2 is initially included in this set. We pick out two strings (1 and 2) randomly from the initial population and compare them with the string in the comparison set. From the QoS parameters it is clear that string-1 dominates the string-2. Hence, string-1 is now included in the non-dominated set. In the next trial strings 2 and 3 are randomly picked up and the same procedure results in a tie as the fitness values indicate both of them as non-dominated. Hence, as discussed in the algorithm, we calculate the niche count using Equations (1), (2), (3), (4) and obtain $d_{s1,s2} = 0.0204$, $d_{s2,s3} = 0.0101$, $d_{s1,s3} = 0.08105$ and $\sigma_{share} = 0.0405$, which leads to $m_2 = 0.2375$ and $m_3 = 0$, as $m_3 > \sigma_{share}$. The lower niche count of string 3 includes it in the non-dominated, Pareto-optimal front. Since, all strings of the population are examined, we now exit from the while loop.

Since the probability of cross-over is quite high, it is performed over both the pairs of strings $1, 2$ and $2, 3$ by selecting the cross-over points at nodes 6 and 7 respectively. The resulting four new strings are : ($1 \rightarrow 3 \rightarrow 6$, $1 \rightarrow 4 \rightarrow 7$, $1 \rightarrow 2 \rightarrow 9$), ($1 \rightarrow 4 \rightarrow 6$, $1 \rightarrow 2 \rightarrow 7$, $1 \rightarrow 4 \rightarrow 9$), ($1 \rightarrow 4 \rightarrow 6$, $1 \rightarrow 2 \rightarrow 7$, $1 \rightarrow 3 \rightarrow 9$), and ($1 \rightarrow 2 \rightarrow 6$, $1 \rightarrow 4 \rightarrow 7$, $1 \rightarrow 2 \rightarrow 9$). On the contrary, as mutation is a rare event it has not occurred in the first iteration. The above process is repeated at every iteration until the improvement is less than our precision. We tabulate the QoS based non-dominated, Pareto-optimal solutions of every iteration in Table 2. Within 4 iterations the improvement of the Pareto-optimal set becomes less than the precision and we conclude that the algorithm has obtained a good solution. The final non-dominated set of multicast trees are shown in Figure 6. From Table 2 it is clear that no single multicast tree gives the best solution in terms of all three QoS parameters, but the first, second and third multicast tree gives the best probabilities for meeting end-to-end delay, residual bandwidth utilization and bandwidth guarantee respectively.

Complexity of the Algorithm: The genetic operators *cross-over* and *mutation* requires $O(n)$ time, where n is the total number of network nodes. Since, the genetic operations are performed on every string in the population, the com-

Table 2. Chart of Multicast Trees with QoS Parameters

Multicast Trees in Different Iterations	$Pr(d_{\mathcal{T}} < t)$	$Pr_{\mathcal{T}}(B)$	$R_b(\mathcal{T})$
$1 \to 3 \to 6$, $1 \to 4 \to 7, 1 \to 2 \to 9$	0.4×10^{-3}	0.0045	0.58
$1 \to 4 \to 6$, $1 \to 2 \to 7$, $1 \to 4 \to 9$	0.6×10^{-3}	0.007	0.55
$1 \to 2 \to 6$, $1 \to 2 \to 7$, $1 \to 3 \to 9$	0.7×10^{-3}	0.005	0.575
$1 \to 3 \to 6$, $1 \to 2 \to 7$, $1 \to 4 \to 9$	0.55×10^{-3}	0.006	0.500
$1 \to 3 \to 6$, $1 \to 2 \to 7$, $1 \to 2 \to 9$	0.7×10^{-3}	0.006	0.565
$1 \to 4 \to 6$, $1 \to 2 \to 7$, $1 \to 2 \to 9$	0.5×10^{-3}	0.008	0.55
$1 \to 2 \to 6$, $1 \to 2 \to 7$, $1 \to 4 \to 9$	0.475×10^{-3}	0.0058	0.625
$1 \to 3 \to 6$, $1 \to 3 \to 7$, $1 \to 4 \to 9$	0.61×10^{-3}	0.0065	0.601
$1 \to 2 \to 6$, $1 \to 2 \to 7$, $1 \to 3 \to 9$	0.95×10^{-3}	0.008	0.645
$1 \to 4 \to 6$, $1 \to 4 \to 7$, $1 \to 3 \to 9$	0.775×10^{-3}	0.0095	0.635
$1 \to 4 \to 6$, $1 \to 3 \to 7$, $1 \to 4 \to 9$	0.821×10^{-3}	0.0081	0.665
$1 \to 3 \to 6$, $1 \to 2 \to 7$, $1 \to 4 \to 9$	0.95×10^{-3}	0.0082	0.641
$1 \to 4 \to 6$, $1 \to 3 \to 7$, $1 \to 3 \to 9$	0.90×10^{-3}	0.0090	0.667
$1 \to 2 \to 6$, $1 \to 4 \to 7$, $1 \to 3 \to 9$	0.88×10^{-3}	0.0097	0.655

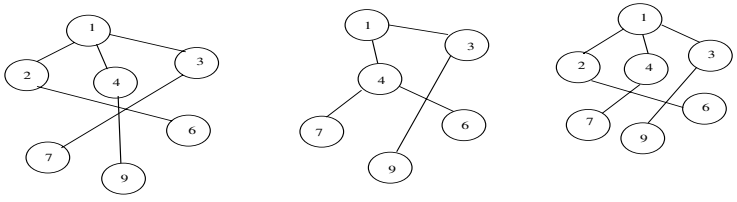

Fig. 6. Final Non-dominated Set of Multicast Trees

plexity of a single iteration of the algorithm will be: $O(\mathcal{P} \times n)$, where $\mathcal{P}$ is the population size. Finally, since, the algorithm is executed for g generations, the total complexity of the algorithm becomes $O(g \times \mathcal{P} \times n)$. The simulation experiments in Section 6 makes it clear that in most of the cases, only a few generations will give a near-optimal result. It is true that the number of iterations (g) varies with the population size ($\mathcal{P}$). A poor guess of choosing the initial population might increase the number of iterations leading to a relatively slower solution. However, such penalty is often tolerated while solving such a NP-hard problem.

Before going into the simulation results of the developed protocol, let analyze the algorithm to show its power, complexity and convergence.

4 Evolutionary Properties and Convergence

The general behavior of the algorithm depends on the fitness values of the individuals in the population. Using fitness distribution before and after the *selection*

operation, several properties of the algorithm can be unveiled to show its power. But before proceeding further, we need to define the following distributions:

Cumulative Fitness distribution: *Fitness distribution* is a function that assigns to each fitness value $f_i \in \mathbf{R}$, the number of individuals in a population P carrying this fitness value. If $\eta \leq N$ is the number of unique fitness values and $f_1 < f_2 < ... < f_\eta$ is the ordering of the fitness values, then the *cumulative fitness distribution* $S(f_i)$ is the number of individuals with fitness value f_i or worse, i.e. $S(f_i) = \sum_{j=1}^{j=i} s(f_j)$, for $0 \leq i \leq \eta$.

Expected Fitness distribution: A *selection method* $\mathcal{M}$ is a function that transforms a fitness distribution s into another fitness distribution s' such that $s' = \mathcal{M}(s, parameter - list)$. The *expected fitness distribution* $\mathcal{M}^*$ after allowing a selection method s to the original fitness distribution ($\mathcal{M}$) is given by $\mathcal{M}^*(s, parameter - list) = E(\mathcal{M}(s, parameter - list))$, [7]. However, for simplicity, the notation s^* is often used to represent this expected fitness distribution. We will try to predict it out of a given distribution. In the selection process used in the algorithm, an individual with fitness f_i or worse can win the tournament if all other individuals have a fitness of f_i or worse. Hence, we need to calculate the probability that all other t individuals have worse fitness. As the probability to choose an individual with fitness f_i or worse is $\frac{S(f_i)}{N}$, we can say $S^*(f_i) = N\left(\frac{S(f_i)}{N}\right)^t$. Now, combining this with the relation $s^*(f_i) = S^*(f_i) - S^*(f_{i-1})$ from definition of cumulative fitness distribution, we get the expected fitness distribution on the multi-objective tournament selection process as:

$$s^*(f_i) = \mathcal{M}^*(s,t)(f_i) = N\left[\left(\frac{S(f_i)}{N}\right)^t - \left(\frac{S(f_{i-1})}{N}\right)^t\right] \tag{5}$$

4.1 Analysis Using Continuous Distribution

We have assumed that the fitness values are continuously distributed. The continuous distribution $\bar{s}(f)$ will have the same range as its discrete counterpart. Hence, $\bar{S}(f) = \int_{f_0}^{f} \bar{s}(x)\partial x$ will be the expression for continuous cumulative distribution. We derive the probability of an individual with fitness f or worse to win the tournament as $\bar{S}^*(f) = N(\frac{\bar{S}(f)}{N})^t$. Again, as $\bar{s}^*(f) = \frac{\partial \bar{S}^*(f)}{\partial f}$, we obtain:

$$s^*(f_i) = \mathcal{M}^*(s,t)(f_i) = t\bar{s}(f)\left(\frac{\bar{S}(f)}{N}\right)^{t-1} \tag{6}$$

Selection Intensity: The *intensity* ($\mathcal{I}$) of the selection, defined as the expected average fitness value of the population after the iteration of the algorithm. Using normalized *Gaussian distribution* $G(0,1)(f) = \frac{1}{\sqrt{2\pi}} e^{\frac{-t^2}{2}}$ we have $\mathcal{I} = \int_{-\infty}^{\infty} f\bar{\mathcal{M}}^*(G(0,1))(f)\partial f$. Thus, the expression for selection intensity of our algorithm is given by

$$\mathcal{I}(t) = \int_{-\infty}^{\infty} tx \frac{1}{\sqrt{2\pi}} e^{\frac{-x^2}{2}} \left(\int_{-\infty}^{x} \frac{1}{\sqrt{2\pi}} e^{\frac{-y^2}{2}} \partial y \right)^{t-1} \partial x \tag{7}$$

We have varied the tournament-size t_{dom} and investigated the changes in $\mathcal{I}$. The plot in Figure 7 demonstrates that the intensity of selection increases with increasing tournament-size until the saturation arrives.

Selection Variance: The *selection variance* $\mathcal{V}$ is the expected variance of the fitness distribution of the strings after the algorithm completes its selection process over Gaussian distribution G(0,1). To calculate this variance with respect to our algorithm we evaluate the equation:

$$\mathcal{V}(t) = \int_{-\infty}^{\infty} t(x - I(t))^2 \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}} \left(\int_{-\infty}^{x} \frac{1}{\sqrt{2\pi}} e^{-\frac{y^2}{2}} \partial y \right)^{t-1} \partial x \tag{8}$$

Figure 8 shows the values of this selection variance with t_{dom}.

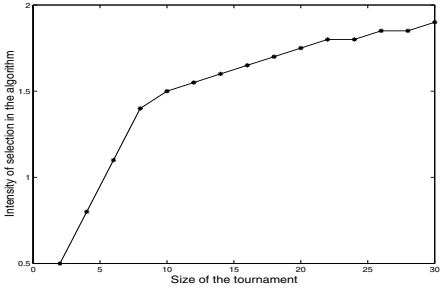

Fig. 7. Selection Intensity with respect to Tournament size

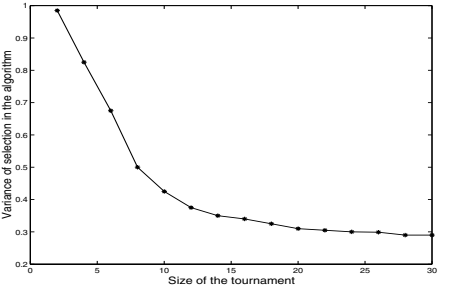

Fig. 8. Variance of the selection with respect to Tournament size

This provides us a trend of the selection pressure used in our algorithm. The selection pressure has its strong influence on selecting the good strings and punishing the bad ones, which eventually guides the improvement of the performance of our algorithm. Now, we will highlight on the convergence of the algorithm.

4.2 Convergence

While examining the convergence of the algorithm, we keep in mind that the proposed algorithm operates on the principle of *elitist GA*, i.e., in every iteration at least the current best individual strings survive. Intuitively, as the algorithm iterates, the fitness of the strings does not decrease. Let us assume that for every population P, there exists a non-zero probability Φ such that in the next generation the fitness of the population is better. Next, we divide the population into classes according to their fitness values. Suppose that the initial population

has fitness value f_{init}, and the optimal fitness value is f_{opt}. Moreover, there are $r \geq 1$ intermediate fitness values. Also, let p denotes the minimum of all the probabilities $\Phi(P)$.

Now, we will proceed to give bounds for the probability that our algorithm reaches optimality in at most t iterations. In, the worst case this optimum will be obtained in exactly $t \geq r-1$ generations, if the $(r-1)^{th}$ improvement takes place in the t^{th} iteration. In order to realize this, we need to pick up $r-2$ different values from the set $\{1, 2, 3, ..., t-1\}$. Indeed, these numbers correspond to the steps where an improvement takes place. Here, we deal with the worst case scenario, in which improvements are as small as possible.

The lowest probability that the algorithm takes precisely t steps equals $p^{r-1}(1-p)^{t-r+1}\binom{t-1}{r-2}$, since we have $r-1$ improvements in the worst case, and $t-(r-1)$ times we get no improvement, i.e., the strings stay in the same class with probability $1-p$. So, the probability that we reach the optimum in at most $t \geq m-1$ steps is bounded by $p^{r-1}\sum_{i=r-1}^{t}(1-p)^{i-r+1}\binom{i-1}{r-2}$.

Using elementary calculus this sum equals:
$p^{r-1}\frac{1}{(r-2)!}\frac{\partial^{r-2}}{\partial q^{r-2}}\left(\sum_{i=1}^{t} q^{i-1}\right) = p^{r-1}\frac{1}{(r-2)!}\frac{\partial^{r-2}}{\partial q^{r-2}}\left(\frac{1-q^t}{1-q}\right)$, where $q = 1-p$. Differentiating and taking limits for $t \to \infty$ we get,

$$p^{r-1}\frac{1}{(r-2)!}\frac{\partial^{r-2}}{\partial q^{r-2}}\left(\frac{1}{1-q}\right) = p^{r-1}\left(\frac{1}{1-q}\right)^{r-1} = 1, \tag{9}$$

since $\frac{\partial^{r-2}}{\partial q^{r-2}}\left(\frac{1}{1-q}\right) \to 0$ as $t \to \infty$.

Therefore, we can conclude that the algorithm converges asymptotically to provide the optimum solution. In the next section we develop a suitable performance model for the proposed algorithm.

5 Performance Modeling Using Markov Chains

Markov chains can be used to model each generation of the algorithm by combining the effects of various stochastic events like initial population generation, selection, cross-over, mutation [18]. However, the major difficulty of it is that the transition probability matrix becomes large and unwieldy. To make the analysis simpler, we encode the node numbers in *binary* form to represent every string by 0s and 1s.

For a binary string encoded population of size $\mathcal{P}$ and M different states, a particular state i in the model represents a population with exactly i ones and $(\mathcal{P}-i)$ zeroes. The algorithm chooses a member k of the current population to reproduce with probability proportional to its fitness relative to total fitness of the population. Thus, leaving the effect of niche counts, we can choose an individual k with probability $\frac{f_k}{\sum f}$, where f_k is the fitness of k and $\sum f$ is the sum of the all individuals in the current population. Now, if f_1 and f_0 denotes the fitness of "1" and "0" respectively, then the probability p_1 of choosing a 1 for the next generation's population will be: $p_1 = \frac{i*f_1}{i*f_0+(\mathcal{P}-i)*f_0} = \frac{\hat{r}*i}{\hat{r}*i+(\mathcal{P}-i)}$, where

$\hat{r} = \frac{f_1}{f_0}$ is the fitness ratio. Similarly, the p_0 probability of choosing a zero will be: $p_0 = \frac{\mathcal{P}-i}{i*\hat{r}+(\mathcal{P}-i)}$. The probability of going from a state of i 1s to a state with j 1s will be: $p_{i,j} = \binom{\mathcal{P}}{j}(p_1)^j(p_0)^{\mathcal{P}-j}$. Substituting the values of p_1 and p_0, we get, $p_{i,j} = \binom{\mathcal{P}}{j}\left(\frac{\hat{r}*i}{\hat{r}*i+(\mathcal{P}-i)}\right)^j\left(\frac{\mathcal{P}-i}{\hat{r}*i+(\mathcal{P}-i)}\right)^{\mathcal{P}-j}$.

The above equation gives a probability transition matrix for the population size and the fitness ratio. However, the absence of niche counts is not incorporated in the equation. Hence, our next objective is to extend the equation to include the niche count into feature and model the algorithm exactly. As discussed earlier, the niche GA seeks to maintain several subpopulations, or individuals at different good solutions and it gives a good view of the fitness landscape. Each peak of such landscapes forms a niche. The sharing values now will modify the fitness values to spread the population out in different peaks. The niche counts for 1s, will be $m_1 = i + (\mathcal{P}-i)(1 - \frac{1}{\sigma_{share}})$, since we have to take care of the shared value of each zero. Similarly, the niche count of 0s will be $m_0 = (\mathcal{P}-i) + i(1 - \frac{1}{1-\sigma_{share}})$. The fitness values will also be changed to f_1/m_1 and f_0/m_0 respectively. Substituting these degraded fitness values to the previous equation, we get

$$p_{(i,j)} = \binom{\mathcal{P}}{j}\left(\frac{1}{1+\frac{(\mathcal{P}-i)(\mathcal{P}+\frac{i-\mathcal{P}}{\sigma_{share}})}{i\hat{r}(\mathcal{P}-\frac{i}{\sigma_{share}})}}\right)^j\left(\frac{1}{1+\frac{i\hat{r}(\mathcal{P}-\frac{i}{\sigma_{share}})}{(\mathcal{P}-i)(\mathcal{P}+\frac{i-\mathcal{P}}{\sigma_{share}})}}\right)^{\mathcal{P}-j} \tag{10}$$

This equation gives the probability of the transition matrix as the algorithm iterates from one state to another.

Absorbing Markov Chain: While calculating such transition probabilities, before talking about steady states, we need to address the absorbing states [3]. Although transition matrix will tell that the *quasi-steady* states can not last, we usually do not wait long enough to see that the algorithm has reached the equilibrium. We keep ourselves satisfied with just a *noisy steady state*. One possible way to deal with this problem is to ignore the steady states and just analyze only the transient states. Applying the well known partitioning of states of an absorbing Markov chain,we get: $P = \begin{pmatrix} QR \\ 0I \end{pmatrix}$.

We take only the Q partition to be the entire matrix, ignoring $R, 0, I$, which consists of only the absorbing states. If we normalize the Q matrix, the resulting matrix Q_{norm}, is an ergodic Markov chain that allows us to calculate the steady state probabilities for all non-absorbing states. Before analyzing Q_{norm}, we will try to justify the "chopping off" the absorbing states. Intuitively, we can say that we are only looking for the *expected absorption time*.

Ergodic Markov chain: We now have an irreducible Markov chain, Q_{norm}, with all ergodic states. Calculation of the steady-state probabilities is quite straightforward. We seek the steady-state probability-vectors $\overrightarrow{\Pi} = \{\pi_1, \pi_2, ..., \pi_{\mathcal{P}-1}\}$, where π_j denotes the steady-state probability for state j. To

find $\overrightarrow{\Pi}$, we need to solve the equation $\overrightarrow{\Pi}Q = \overrightarrow{\Pi}$, where $\sum_{i=1}^{\mathcal{P}-1} \pi_i = 1$. Analyzing the vector $\overrightarrow{\Pi}$ helps us to understand the behavior of the steady state probabilities, plotted in Figure 9 against changing σ_{share} values.

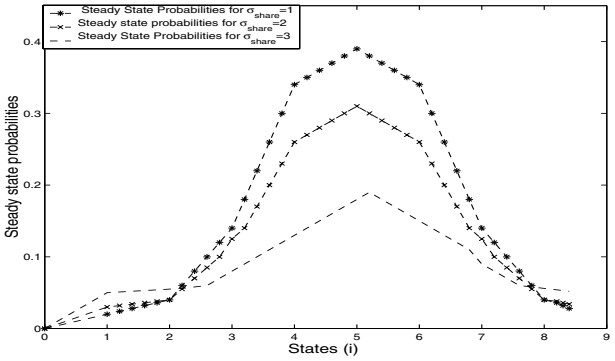

Fig. 9. Steady State Probabilities

All the steady state distribution curves are almost symmetric about the equilibrium point. As σ_{share} increases, the steady state distribution curve flattens and demonstrates the changing probabilities for different fitness sharing values.

6 Simulation Results

Simulation experiments are first performed over a network of $n = 100$ nodes with the number of multicast destination nodes being 10. The capacity of the network links are taken as uniformly distributed in the interval of [90-110]Mbps.Recall that our multicast QoS routing algorithm attempts to maximize the probabilities of meeting end-to-end delay, bandwidth requirement and bandwidth utilization within a few generations by building the Pareto- optimal fronts. We have compared the performance of our algorithm with an existing scalar-optimization [1] and heuristic algorithms [14] and observed that our algorithm performs better in terms of scalability and multicast call blocking rates.

An exhaustive search method, which finds the optimal values of the three QoS parameters by exhaustively searching them one after another is used to compare our results. The three plots (one for each QoS parameter) in Figures 10, 11 and 12 vividly explains how these Pareto-optimal fronts are developed and proceeded towards a global-optimal solution in a feasible time. The novelty of our algorithm is that it is capable of obtaining near-optimal values of all three QoS parameters simultaneously by building the non-dominated fronts. However, for the sake of clarity we have shown it in three different plots. Finally, after completing the execution of the algorithm, we get the final solution sets represented by the

Figure 13. From the above plots one can derive the amount of optimization obtained by our algorithm. Table 3 demonstrates that our algorithm is capable of obtaining more than 95% of the global-optimal values of end-to-end delay, bandwidth guarantee and residual bandwidth utilization within 100 iterations. As all the near-optimal solutions are achieved in a probabilistic approach, we conclude that our algorithm is robust enough to operate with imprecise network information. Note that the solution set may contain solutions which are not the best from any single objective's point of view, but is non-dominated by all three individual best solutions, when all three objectives are considered. Since the three individual best solutions will always be non-dominated, they are by default included.

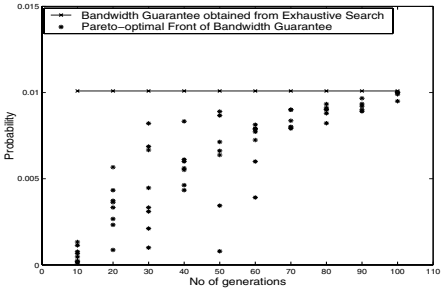

Fig. 10. Pareto-Optimal Bandwidth Guarantee with number of generations

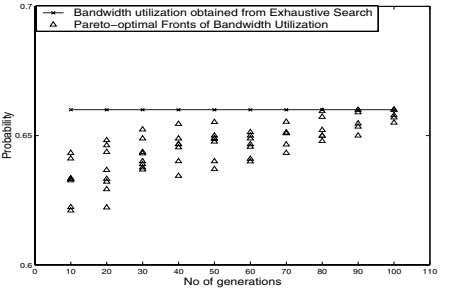

Fig. 11. Pareto-Optimal Bandwidth Utilization with number of generations

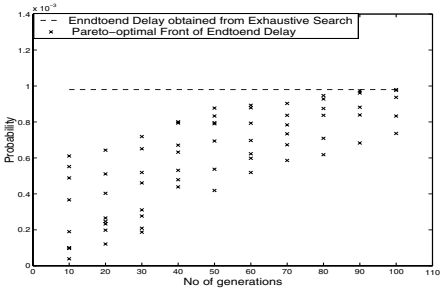

Fig. 12. Pareto-Optimal End-to-end Delay with no. of generations

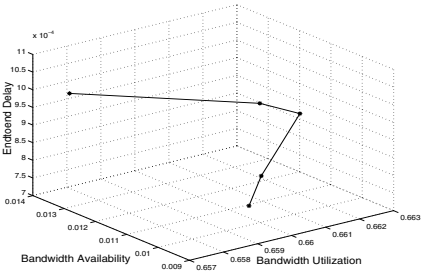

Fig. 13. Pareto-optimized Set of three QoS parameters

The solutions are provided in a generalized manner and a user can readily choose his choice-able solution depending on his needs. For example, in real time video transmission we are more careful about the end-to-end delay. Such a user

Table 3. Percentage of Global-optimal Solutions Obtained

End-to-end Delay	96.78
Bandwidth Guarantee	95.55
Bandwidth Utilization	98.39

will be having delay as a hard constraint and will choose the solution which will meet that constraint. On the other hand, while transmitting a scientific data from the remote satellite, the correctness is more important than the delay. So, for such cases an end-user will prefer to meet the bandwidth guarantee than the delay. We repeat the simulation with increasing number of network nodes and observe the efficiency of our algorithm. As the network becomes highly condensed, our algorithm exhibits a more linear and stable pattern than existing scalar optimization algorithm. This *approximate linearity* of the curve in Figure 14 corroborates the scalability of the algorithm. Finally, the non-dominated set of solutions are given as input to the call-blocking algorithm. Performance of our protocol is plotted against the increasing *call arrival rate* in Figure 15. The mean rate of arrival of multicast session request is assumed to be 10 requests/sec. Results show that the percentage of calls blocked in our protocol is less than the two existing QoS routing protocols based on scalar optimization [1] and heuristics [14]. The peak data rate for this comparison is taken as 35Mbps. Although the performance of all the schemes degrades with the increase of call arrival rate, our algorithm gains consistently over the existing ones. Hence, we can conclude that the designed protocol offers a *graceful degradation* of performance with increasing session arrival rates.

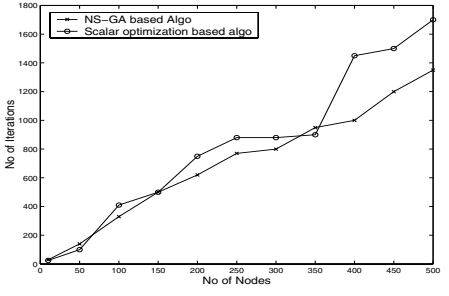

Fig. 14. Performance of the Algorithm with Increasing number of nodes

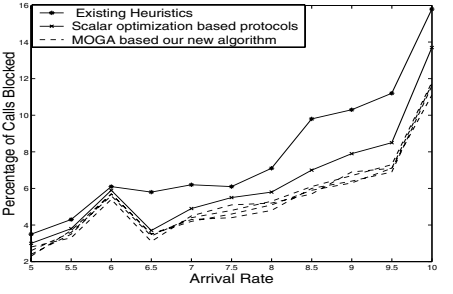

Fig. 15. Percentage of Calls Blocked with Call-Arrival Rates

7 Conclusions

On-demand multicast routing in networks is currently an active area of research. In most of the real world scenarios routings need to meet stringent measures of different quality of services. Seamless transmission of wireless audio and video traffic has already become a real challenge of current and future generation wireless systems. It is quite natural that real time multimedia traffic should meet a number of different and conflicting QoS issues. Optimizing a particular objective function may sacrifice optimization of another dependent and conflicting objective. In this paper, we studied QoS-based multicast routing problem from the perspective of multi-objective-optimizations. The blessing of multi-objective-genetic algorithms (MOGA) has paved the way to develop the algorithm for a new QoS-based multicast on-demand routing algorithm. The mathematical analysis shows the power of selection and complexity of the algorithm. We have also shown the asymptotic convergence of the algorithm to the optimal point. However, often we do not need to wait till the convergence and settle with a near optimal point. We have also developed a suitable model of the algorithm using Markov chains to track the transition probabilities and plot the steady state values of such probabilities. Simulation results delineates the efficiency, performance and scalability of the protocol. Our future interests is to adapt this technique to develop a mechanism for *renegotiable-QoS* in wireless multicasting. We expect our work will be helpful in solving some new problems in the domain of quality-of-service (QoS) routing.

References

1. N. Banerjee and S. K. Das, "Fast Determination of QoS-based Multicast Routes in Wireless Networks using Genetic Algorithms" *International Conference for Communication*, vol. 8, pp.2588-2592, 2001.
2. C. A. C. Coello, "An Updated Survey of GA-Based Multiobjective Optimization Techniques," *Technical Report, Laboratorio Nacional de Informatica Avanzada (Lania), Xalpa, Veracruz, Mexico*, pp. 1-45, June 1998.
3. J. N. Darroch and E. Seneta, "On quasi-stationary distributions in absorbing discrete-time finite Markov chains," *Journal of Applied Probability*, pp. 88-100, 1965.
4. L. Davis, "Handbook of Genetic Algorithms," *Van Nostrand Reinhold*, New York, 1991.
5. K. Deb and D. E. Goldberg "An investigation of niches and species formation in genetic function optimization," *Proceedings of the third International Conference on Genetic Algorithms*, pp. 42-50, 1991.
6. C. M. Fonesca and P. J. Fleming, "Genetic Algorithms for Multiobjective optimization: formulation, discussion and generalization," *Proceedings of the Fifth International Conference on Genetic Algorithms*, pp. 416-423, 1993.
7. D. E. Goldberg, " Genetic Algorithms : Search, Optimization and Machine Learning," *Adison-Wesley*, 1989.
8. R. A. Guerin and A. Orda, "QoS Routing in networks with Inaccurate Information: Theory and Algorithms," *IEEE/ACM Transactions on Networking*, vol. 7, no. 3, pp. 350-364, June 1999.

9. J. Horn, N. Nafpliotis and D. E. Goldberg, " A Niched Pareto Genetic Algorithm for Mutiobjective Optimization," *IEEE Conference on Evolutionary Computation*, New Jersey, vol. 1, pp. 82-87, 1994.
10. P. G. Harrison and N. M. Patel, "Performance modeling of Communication Networks and Computer Architectures," *Addison Wesley*, Reading, MA, 1989.
11. Y. Iraqi, R. Boutaba and A. Leon-Garcia, "QoS Control in Wireless ATM," *Mobile Networks and Applications*, vol. 5, pp. 137-145, 2000.
12. R. Jain, B. Sadeghi and E. W. Knightly, "Towards Coarse-Grained Mobile QoS," *Workshop on Wireless Mobile Multimedia*, pp. 109-116, 1999.
13. V. P. Kompella, J. C. Pasquale and G. C. Polyzos, "Multicasting for Multimedia Applications," *Proc. of IEEE Infocom 92*, Florence, Italy, vol.3, pp. 2078-2085, May 1992.
14. V. P. Kompella, J. C. Pasquale and G. C. Polyzos, "Multicast Routing for Multimedia Communications," *ACM/IEEE Transaction on Networking*, vol. 1, no. 3, pp. 286-292, Jun. 1993.
15. D. H. Lorenz and A. Orda, "QoS Routing in networks with Uncertain Parameters: Theory and Algorithms," *IEEE/ACM Transactions on Networking*, vol. 6, no. 6, pp. 768-778, Dec. 1998.
16. M. Naghshineh and M. W. Wilebeek-LeMair, "End-to-End QoS provisioning in Multimedia Wireless/Mobile Networks Using an Adaptive Framework," *IEEE Communications Magazine* vol. 6, no. 6, pp. 72-79, Nov. 1997.
17. R. Nelson, "Probability, Stochiastic Processes, and Queueing Theory," *Springer-Verlag*, 1995.
18. A. E. Nix and M. D. Vose, "Modeling Genetic Algorithms", *Analysis of Mathematics and Artificial Intelligence*, vol. 5, 1992.
19. S. Shenker, C. Patridge and R. Guerin, "Specification of Guaranteed Quality of Service, " Request for comments *RFC 2212, Internet Engineering Task Force*, Sept. 1997.
20. N. Srinivas and K. Deb, " Multiobjective Optimization Using Nondominated Sorting in Genetic Algorithms" *Journal of Evolutionary Computation*, vol. 2, no. 3, pp. 221-248, 1995.