AnyTraffic Labeled Routing

Dimitri Papadimitriou[1], Pedro Pedroso[2], Davide Careglio[2]
[1] Alcatel-Lucent Bell, Antwerp, Belgium
Email: dimitri.papadimitriou@alcatel-lucent.com
[2] Universitat Politècnica de Catalunya, Barcelona, Catalonia, Spain
Email: {ppedroso,careglio}@ac.upc.edu

Abstract—**This paper investigates routing algorithms that compute paths along which combined unicast and multicast traffic can be forwarded altogether, i.e., over the same path. For this purpose, the concept of AnyTraffic group is introduced that defines a set of nodes capable to process both unicast and multicast traffic received from the same (AnyTraffic) tree. The resulting scheme is referred to as AnyTraffic routing. This paper defines a heuristic algorithm to accommodate the AnyTraffic group and to find the proper set of branch nodes of the tree. The algorithm supports dynamic changes of the leaf node set during multicast session lifetime by adapting the corresponding tree upon deterioration threshold detection. Studies are performed for both static and dynamic traffic scenarios to i) determine the dependencies of the algorithm (node degree, clustering coefficient and group size); and ii) evaluate its performance under dynamic conditions. Initial results show that the AnyTraffic algorithm can successfully handle dynamic requests while achieving considerable reduction of forwarding state consumption with small increase in bandwidth utilization compared to the Steiner Tree algorithm.**

I. Introduction

WITH the advent of multimedia video stream/content, multicast distribution from a source to a set of destination nodes is (re-)gaining interest as a bandwidth saving technique competing or complementing cached content distribution. Nevertheless, the problems faced in the 90's when multicast received main attention from the research community are still present. Routing protocol dependent multicast routing schemes (such as Distance Vector Multicast Routing Protocol and Multicast Open Shortest Path First) have been replaced by routing protocol independent routing schemes such as Protocol Independent Multicast (PIM) and Core Base Trees. However, number of entries in routing and forwarding tables (states) and their maintenance is and remains a major problem. Two forwarding approaches are commonly used in current datagram networks, namely 1) forwarding on a set of point-to-point (P2P) paths to encapsulate whether unicast or multicast traffic (i.e., multicast traffic is replicated as many times as the number of edge nodes processing multicast traffic); and 2) forwarding on a set of dedicated P2P paths for unicast traffic and dedicated point-to-multipoint (P2MP) paths for multicast traffic. The latter can be either root-initiated as in source-specific multicast or leaf-initiated as in any-source multicast. Regardless of the underlying forwarding paradigm, a router must maintain membership state for each multicast group. Multicast membership states are stored as entries in the routing table that is subsequently used to derive a forwarding table. The latter determines the actual forwarding of an incoming packet to a router's outgoing interfaces. However, unlike unicast routing, there is no natural aggregation in multicast forwarding states thus a router may take a long time to look up the forwarding state for each arriving packet when there are a large number of multicast group [1]. This results in limited scalability of any multicast routing deployment. Several research efforts have proposed methods to reduce the number of multicast forwarding state through aggregation [2] [3], tunneling [4], and no-branching state elimination [5] [6]. Even if some of these techniques might be considered as acceptable, the proposed concept is totally new and tries to improve such existing algorithms by achieving lower state consumption at low bandwidth consumption increment.

In this paper, we investigate routing algorithms that compute paths along which both unicast and multicast traffic can be forwarded. Section II details the newly introduced concept together with the joint forwarding process. The AnyTraffic heuristic algorithm is formulated in Section III for both static and dynamic routing. Performance results obtained for both cases together with their analysis are presented in Section IV. Future work and conclusions are drawn in Section V.

II. AnyTraffic Routing Concept

This paper proposes a traffic routing approach, whose computed paths enable forwarding of both unicast and multicast traffic together, i.e., over the same path. For this purpose, the concept of *AnyTraffic group* is introduced that defines a set of nodes capable to process both unicast and multicast traffic received from the same distribution tree, the *AnyTraffic tree*. The resulting routing scheme is referred to as *AnyTraffic routing*. This paper defines a heuristic algorithm to accommodate the AnyTraffic group and to find the proper set of branch nodes of the AnyTraffic tree. It also provides for a performance evaluation of the proposed scheme against two commonly used approaches.

Introducing an AnyTraffic distribution tree to a group aims at reducing the total number of forwarding states by maintaining (as much as possible) a single path for both unicast and multicast traffic forwarding altogether. In other terms, a single state allows for both unicast and multicast traffic forwarding. The idea is to perform label-based forwarding (where labels encode topological information) using a single forwarding table entry for both unicast and multicast labeled traffic directed toward the same "label". Network nodes are named with destination labels that encode topological information. These labels are used in the forwarding decision process: each datagram carries the chosen destination in its header. At intermediate nodes, a discriminator (set at network ingress node) allows selecting either a unicast (one-to-one) or a multicast (one-to-many) forwarding entry associated to the same label entry. Differentiation between incoming unicast and multicast traffic is simply performed by means of a (single-bit) discriminator. Thus, the proposed routing scheme is applicable to any label-based forwarding technology as long as the following conditions are met: i) capability to distinguish multicast from unicast traffic by inspecting other header information than the destination address (e.g. label flag to discriminate between unicast and multicast traffic following the same path); and ii) de-multiplexing of traffic at destination nodes relies on the information encoded as part of other header information not processed by each network node. This scheme can be seen as a unification of the locator/identifier split concept where the locator value space names topological end-points that are able to terminate any traffic and the identifier value space allows distinction (at the edges) between unicast and multicast traffic. Ingress edge nodes upon multicast traffic identification tag this traffic as part of the label. Based on this indication, branch nodes along the AnyTraffic tree replicate the multicast traffic onto outgoing interfaces towards edge nodes registered for the corresponding multicast group(s). On the other hand, the unicast traffic directed to these edge nodes is not replicated at branch nodes but follows "as short as possible" paths. The salient feature of the proposed scheme is that the multicast traffic does not require any additional forwarding entry on intermediate network node to reach the topological location where the traffic is then natively processed.

The aim of the proposed routing scheme is to achieve better system resource consumption (for state maintenance) while limiting the network resource consumption, i.e., mitigate the state vs. bandwidth resource tradeoff by increasing the "common path" stretch. The proposed approach keeps the forwarding state maintenance overhead as low as possible while avoiding bandwidth waste by i) avoiding replication of multicast traffic at branch nodes, and ii) keeping unicast traffic forwarding over "as short as possible" paths. To meet this objective combined with the decrease in hop count of P2P paths, a deficit factor and an adaptive threshold function for the selection of the branch nodes are specified to decide where to separate the unicast from the multicast forwarding path (i.e., the placement of a branch node). This algorithm is also designed so as to efficiently operate in a dynamic environment where receivers are joining and releasing multicast sessions. Beside the reduction of the number of states, the AnyTraffic routing scheme can also handle more efficiently join and leave requests. Here, as both types of requests may deteriorate the system resource performance compared to the optimal case, a readjustment mechanism is designed so as to accommodate actual receivers' dynamics by adapting the multicast tree. Finally, resulting from the type of routing information it processes, the proposed routing scheme applies typically within network partitions (intra-domain).

III. AnyTraffic Routing Algorithm

Consider a network modeled by a directed graph G = (N, L), where N represents the finite set of nodes, and L represents the finite set of links. Let s, d $\in$ N denote a source and a destination node, respectively. Each link l $\in$ L might have an associated capacity b(l), and cost c(l). Let $p_{i,j}$ and $p_{i,k,j}$ both denote a path from node i to node j, where k is an intermediate node with $i \neq j \neq k$ .Let $T_{s,M} = (N_T, L_T)$ be a connected sub-graph without cycles (i.e., a tree) of G, source-initiated at s, and with the set of destination nodes $M \subseteq N_T \setminus \{s\}$, $M \neq \emptyset$. Hereafter, M is referred to as the AnyTraffic group, and $T_{s,M}$ as the AnyTraffic tree.

A. *Static AnyTraffic Heuristic Algorithm*

Let $\phi_{s,M}$ denote a traffic request between source s and a AnyTraffic group M, where $M \subseteq N_T \setminus \{s\}$, $M \neq \emptyset$. If $|M| = 1$, $\phi_{s,M}$ is a request for unicast traffic. The objective of the AnyTraffic routing algorithm is to construct a graph $T_{s,M}$ for a given source s and AnyTraffic group M, such that $T_{s,M}$ supports both unicast and multicast traffic requests. The graph $T_{s,M}$ is constructed by successive selection of branch node, n* $\in$ N. At a given source node s, processing of the request $\phi_{s,M}$ depends on its nature, i.e., it is either a unicast or multicast traffic request. We have the following alternatives: i) if a multicast traffic request $\phi_{s,M}$ arrives and an AnyTraffic tree $T_{s,M}$ is available, then the request is supported by $T_{s,M}$; otherwise, the AnyTraffic routing algorithm is executed (see Section III.A.2) to establish a new AnyTraffic tree; ii) if a unicast traffic $\phi_{s,d}$ request arrives, three situations can occur: (a) d $\in$ M and $T_{s,M}$ (with $|M| > 1$) is available and $\phi_{s,M}$ is supported by $T_{s,M}$; (b) d $\in$ M but $T_{s,M}$ is not yet created and thus a shortest path must be setup; or (c) d $\notin$ M and thus a shortest path must be setup. The AnyTraffic routing algorithm comprises two phases, namely, the initialization and tree computation phase.

1) *Initialization Phase*

Let $x_{i,j}$ denote the cost of the shortest path from node i to j, $i \neq j$, as computed by the Dijkstra algorithm on the (positive integer) link cost c(l), l $\in$ L. The hop count is used as tie-breaker. Methods for computing the cost c(l) of each link l $\in$ L can be found in [7]. Accordingly, $x_{s,d}$ denotes the cost of the shortest path from the source s to the destination d $\in$ M. Among all path costs, c_{max} corresponds to the shortest path of maximum cost. Let F(x): $\mathbb{R}^+ \rightarrow \mathbb{R}^+$ be defined as:

$$F(x) = x\left(1 + e^{\frac{-g(x)}{c_{\max}}}\right) \qquad (1)$$

This function specifies the threshold for the maximum cost of an alternative path to the path of cost x [7]. In particular, $F(x_{s,d})$ limits the acceptable cost deviation of an alternative path $p_{s,d}$, d $\in$ M, from the path given by the Dijkstra algorithm. The function g(x): $\mathbb{R}^+ \rightarrow \mathbb{R}$, is defined as $g(x) = \alpha x - \beta$, where parameters $\alpha, \beta \in [0,1[$ define the shape of the threshold function. High values for α and β would make the function too restrictive, allowing only for low path stretch. On the other hand, low values would result into too high path stretch. Having defined F, we can specify the *maximum deficit factor* $\Delta M_{s,d}$ for each shortest path $p_{s,d}$,:

$$\Delta M_{s,d} = F(x_{s,d}) - x_{s,d} = x_{s,d} e^{\frac{-g(x)}{c_{\max}}} \qquad (2)$$

This factor determines the acceptable cost increment(s) of an alternative path against the shortest path, i.e., it quantifies the tolerable cost deviation when forwarding both multicast and unicast traffic on that path without incurring too much damage compared to unicast traffic forwarding along the shortest path. Being only dependent on the topology, $x_{s,d}$, c_{max}, $g(x_{s,d})$, and $\Delta M_{s,d}$ can be computed during the initialization phase for any pair (s,d) ∈ N (values remain constant during the tree computation phase).

2) Tree Computation Phase

Let's define a leaf as the tuple $\omega_{\upsilon,\Lambda} = \{\upsilon, \Lambda\}$, where υ ∈ N is a leaf seed and Λ ∈ M is a subset of the AnyTraffic group. We define **Ω** as the set of leaves remaining to be processed. At the beginning, this set comprises only the initial leaf, $\Omega = \{\omega_{s,M}\}$, where s is the seed from where computation is initiated, which comprises all destination nodes M. We also define the initial graph $T_{s,M} = (\{s\}, \varnothing)$. The algorithm terminates when there is no leaves left in **Ω** and all destinations d ∈ M can be reached from s in $T_{s,M}$. At each iteration step, an arbitrary leaf $\omega_{\upsilon,\Lambda}$ is pulled out from **Ω** and the algorithm searches for a branch node n* ∈ N to be included in $T_{s,M}$ such that *s* is connected through n* to a subset of nodes comprised in Λ. For this leaf $\omega_{\upsilon,\Lambda}$, a set of candidate branch nodes A_ω is found. The set A_ω is restricted to unvisited nodes in previous iterations that are adjacent to υ and have a node degree equal or greater than three, i.e., the nodes that have at least two outgoing links, apart from the outgoing link to node υ. In case the node degree of an adjacent node a is two, the first node with node degree equal or greater than three and laying on a path going from υ through a is included into A_ω. At each candidate branch node n ∈ A_ω being evaluated, one alternative path $p_{v,n,d}$ per destination d ∈ M, starting at υ but forced to pass through n is computed. A pruning condition determines if the set of alternative paths from υ to each d ∈ L and passing through *n* could be accepted. Indeed, each path $p_{v,n,d}$ may introduce higher cost ($x_{v,n} + x_{n,d}$) when compared to the cost $x_{v,d}$ of the shortest path $p_{v,d}$. Therefore, a *local deficit* $\Delta L_{v,n,d}$ is computed for each d ∈ Λ by means of:

$$\Delta L_{\upsilon,n,d} = \left(x_{\upsilon,n} + x_{n,d}\right) - x_{\upsilon,d} \tag{3}$$

For each d ∈ Λ, a *cumulative path deficit* $\Delta P_{s,d}$, sums up, at node n, the local deficits produced by the alternative path $p_{s,d}$ passing by already accepted branch nodes n_0 (= s), $n_1,\ldots,n_u$ of $T_{s,M}$ and the candidate branch node n_{u+1} (= n):

$$\Delta P_{s,d} = \sum\nolimits_{i=0}^{u} \Delta L_{n_i,n_{i+1},d} = \sum\nolimits_{i=0}^{u} \left(x_{n_i,n_{i+1}}\right) + x_{n,d} - x_{s,d} \tag{4}$$

Then, for each d ∈ Λ, a comparison between the cumulative path deficit (Eq.4) and the maximum deficit (Eq.2) is performed. If the maximum deficit constraint $\Delta P_{s,d} \le \Delta M_{s,d}$ is verified, i.e., if the cumulative deficit of an alternative path $p_{s,d}$ does not exceed the maximum deficit, the alternative path can be accepted. Otherwise, the algorithm removes node d from $\omega_{\upsilon,\Lambda}$ and creates a new leaf. When all candidate branch nodes have been evaluated, branch node selection can be performed by running the pruning condition for each d ∈ Λ. The decision is taken by considering the minimum total deficit among all candidate branch nodes n ∈ A_ω. However, to reach decision fairness, considering the deficit based on the cost metric only is insufficient. Hence, we further ponder the deficit of each candidate branch node n by i) summing a fraction γ of the local deficit (Eq.3) to a fraction (1 − γ) of a local deficit $\Delta H_{v,n,d}$ defined as Eq.3 but using the hop count instead of the cost metric; ii) multiplying by a factor σ the ratio r, defined as the number of alternative paths meeting the pruning condition divided by the total number of paths that can reach all destinations, i.e., |Λ|, via n ∈ A_ω. The parameter γ governs the selection trade-off between longer but lower cost paths among the successful alternative paths meeting the pruning condition (high γ values), and shorter paths thus, lower probability to aggregate paths (low γ values). The parameter σ is a weight factor that favors candidate branch nodes with a higher number of successful alternative paths. For each candidate branch node, a *candidate deficit* $\Delta C_{n,\omega}$ is computed as:

$$\Delta C_{n,\omega} = \sum \left[\gamma \Delta L_{\upsilon,n,d} + (1-\gamma)\Delta H_{\upsilon,n,d}\right] - \sigma r \tag{5}$$

The candidate branch node n with the lowest deficit is selected as a branch node n*. Accordingly, $T_{s,M}$ is updated with all links and nodes that lay on the path from υ, the seed of the currently processed leaf $\omega_{\upsilon,\Lambda}$, to n*. Then, two new leaves may be created, $\omega_1 = \{n^*, \Lambda_{n^*}\}$ (leaf with the subset of destination nodes Λ_{n^*} that accepted n* as branch node), and $\omega_2 = \{\upsilon, \Lambda \setminus \Lambda_{n^*}\}$ (leaf with the destination nodes removed by the pruning condition). Leaves ω_1 and ω_2 are conditionally added to the set **Ω** for further processing, respectively, if $|\Lambda_{n^*}| > 1$ and $|\Lambda \setminus \Lambda_{n^*}| > 1$. If either $|\Lambda_{n^*}| = 1$ or $|\Lambda \setminus \Lambda_{n^*}| = 1$, $T_{s,M}$ is updated with all links and nodes along the shortest path, respectively, from n* to d ∈ Λ_{n^*} and from υ to d ∈ $\Lambda \setminus \Lambda_{n^*}$. Branch node n* is excluded from the set of adjacencies of υ, i.e., $A_{\omega 2} = A_\omega \setminus \{n^*\}$. Branch selection is repeated for each leaf left in **Ω**.

3) Complexity Analysis

The complexity of this algorithm is O (|M| · A · H), where |M| is the size of the AnyTraffic group, A is the maximum node degree, and H is the hop distance between the source and the most distant destination node. This bound comes from the fact that at each hop towards the destination all adjacent nodes are checked as candidate branch node for the destination nodes belonging to M. In a regular connected network (A << |N|), the complexity is low and it may be further reduced by limiting the set of adjacent nodes that are within a given perimeter with respect to the next node along the shortest path to each destination node. Results achieved by applying this method show no performance degradation while significant reduction of running time.

B. Execution Example

Fig.1 shows two consecutive steps of the branch node evaluation mechanism. We consider as initial leaf $\Omega_{s,M}$, where *s* is the node processing the incoming requests for both unicast and multicast traffic, and the AnyTraffic group M={d1,d2}. Fig.1a represents the evaluation of the set A_ω of adjacent branch nodes for destination d1. The set A_ω={1,2,3} corresponds to the adjacent nodes of *s* with a node degree equal or higher than three. The path through node 1 is the shortest path and its cost $x_{s,d1}$ gives the value of the maximum deficit factor $\Delta M_{s,d1}$. For each of the remaining adjacent nodes, an alternative path is computed. The alternative path through node 2 and the one through node 3 may introduce higher cost with respect to the shortest path; these costs define the cumulative path deficit factor $\Delta P^1_{s,d1}$ and $\Delta P^2_{s,d1}$ respectively. At this step, an adjacent node is accepted as candidate branch node if the pruning condition is verified; for example node 2 is accepted if the cost of its alternative path $\Delta P^{1}_{s,d1}$ is lower than $\Delta M_{s,d1}$. The same evaluation is then repeated for each remaining destination node. Fig.1a also depicts evaluation for destination node d2. Once all adjacent nodes have been evaluated through the pruning condition, branch node selection is performed. For each accepted candidate branch node, the candidate deficit factor $\Delta C_{n,\omega}$ is computed. The node with the lowest value is selected as branch node. Assuming that the candidate branch node 2 is selected which corresponds to this situation depicted in Fig.1b, nodes s and 2 are added to $T_{s,M}$. The leaf seed is now $\omega_{2,M}$, the set of adjacent nodes of node 2 is A_{ω}={1,3,4,6}, and three alternative paths are considered (the path through node 4 was already accepted from the previous step). The same process described in the previous iteration is then repeated.

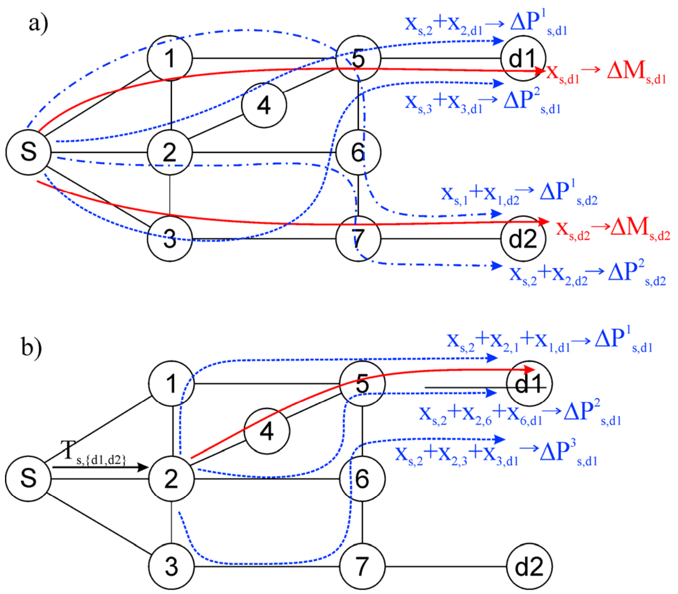


Fig.1: Two consecutive steps of the branch node evaluation mechanism

C. *Dynamic AnyTraffic Heuristic Algorithm*

Let $\phi_{s,d}$ and $\varphi_{s,d}$ denote respectively, a join and a leave request between source s and destination d. Graph $T_{s,M} = (N_T,L_T)$ maintenance by the AnyTraffic algorithm under dynamic traffic requests conditions consists in appending node $d \notin N_T$ to the graph $T_{s,M}$ when a join request $\phi_{s,d}$ arrives, and releasing node $d \in M$ from $T_{s,M}$ when a leave request $\varphi_{s,d}$ arrives. Graph $T_{s,M}$ is built up by iterative selection of branch nodes $n \in N_T$.

1) *Join request*

As regards join requests, the unicast traffic is forwarded over either i) a shortest path if the receiver is not a member of any AnyTraffic group, or ii) an existing AnyTraffic tree. As regards multicast, two other situations can occur: i) initiate a new AnyTraffic tree; or ii) join an existing AnyTraffic tree at one of its branch node. In any case, if an existing P2P path, rooted at the same source node, is up for such receiver, it is aggregated to the established AnyTraffic tree. A possible approach for a receiver to join an existing tree would be to re-compute the entire tree as if a new group request would arrive (using the static algorithm). Such computation is optimal for a given receivers group and can thus be considered as an upper bound on the algorithm performance. The disadvantage is the need to re-establish the entire tree each time a join request is received. To overcome this situation, we propose an extended mechanism to update the tree without re-computing the entire tree. Using this approach, deviation from the best case (as given by the static algorithm) is controlled by the mechanism of Section III.C.3. Let's assume a new receiver node $d \notin N_T$ attempts to join the AnyTraffic group M supported by the tree $T_{s,M}$. Updating the tree "on-the-fly" lies in joining the closest node of the tree under the maximum deficit constraint. The algorithm performs the following steps: 1) A Breadth-First Search algorithm is executed to find a set of candidate branch nodes $A_{\omega} \in N_T$ with the shortest hop count to node d; 2) For each node $n \in A_{\omega}$, find the shortest path $p_{n,d}$ to the receiver. For each path $p_{s,n,d}$ obtained by splicing path $p_{s,n}$, which is determined by the tree $T_{s,M}$, and $p_{n,d}$, calculate its deficit ¨ $P_{s,d}$. Then, among all these paths, select the path with the smallest deficit $\Delta P_{s,d}$, such that it satisfies the constraint $\Delta P_{s,d} \leq \Delta M_{s,d}$; 3) If such path $p_{s,d}$ is not found, step 1 is repeated by excluding the already processed nodes from the set of candidate nodes A_{ω}; 4) Once these steps are completed, as the receiver may still have unicast connectivity rooted at the source node of the AnyTraffic tree, the corresponding forwarding table entries are removed and traffic is forwarded over the tree.

2) *Leave/Prune request*

When a multicast traffic receiver wants to leave an AnyTraffic tree, the simplest operation consists of pruning the leaves of the tree which are not used by any other remaining receivers. This leads to two cases: the leaf node could be either a destination node or an intermediate node of the tree. Let's assume a receiver $b \in T_{s,M}$, attempts to leave the AnyTraffic group M. The following operations must be performed: 1) if node b is a leaf node, then the path from branch node n to node b must be pruned; 2) if node b is an intermediate node, the entry for this node must be removed from forwarding table. The forwarding state is not removed because some receivers are still active along the path. In both cases, a check is performed to verify if any P2P path is up for the leaving receiver. In case the receiver is a member of other AnyTraffic group and the releasing branch node crosses one of the corresponding AnyTraffic tree, unicast traffic may be redirected over one of the existing trees. Concerning unicast release requests, if the receiver is a member of a multicast session, then the request does not result into any state update if the corresponding path shares the same forwarding state with an AnyTraffic tree.

3) *Deterioration Control*

In a dynamic environment, after a certain period, join and leave requests deteriorate the entire AnyTraffic trees, due to the unpredictability of events. The process of locally re-adapting the tree pursues the detection of deterioration, i.e., deviation of the re-adapted tree from the best case. The deviation is computed by the formula $w\,D_s + (1-w)\,D_b$ (Eq.6), where D_b and D_s accounts respectively for the bandwidth and state consumption differences. To penalize higher state consumption, D_b and D_s are weighted asymmetrically by the weight factor w. Higher w values imply re-computation when excessive states are used compared to the best case; lower values mean that deviation in terms of consumed bandwidth leads to re-computation. The pre-determined deterioration threshold value is used to decide to either continue with the on-the-fly adapted tree (up to a certain deviation from the best case) or instead shift to a full tree re-computation. A high threshold value means less re-computation; on the contrary, a low value means stricter control, avoiding bandwidth and state consumption at the expense of more computation.

4) Complexity Analysis

The time complexity is O ($A^{H'} \cdot |N_T|$), where A is the maximum node degree, H' is the hop distance between the node to be attached to the tree and the most distant node of the tree, and $|N_T|$ is the number of nodes of the tree. Indeed, the number of iterations the algorithm performs depends mainly on the candidate branch node search, implemented by the Breadth-First Search algorithm. At each hop, the algorithm explores adjacent nodes looking for candidate branch nodes. Then, for each candidate node, the constraint compliance procedure is applied. In the worst case, all nodes have to be checked. The time complexity can be approximated by O($|N_T|$) since any node of the graph G is visited at most only once.

IV. Performance analysis

Simulations are performed to estimate the performance of the AnyTraffic algorithm in terms of bandwidth and state consumption, under the following scenarios: i) non-blocking static traffic; ii) dynamic traffic with limited capacity per link. Two reference approaches are used for comparison purpose: approach 1 (AP1) that forwards both unicast and multicast traffic along "as short as possible" paths (shortest path routing); and approach (AP2) that makes use of shortest path routing for unicast traffic and replication of multicast traffic at branch points of a tree as computed by the minimum-cost path algorithm, a Steiner Tree Heuristic (STH) [8]. For the dynamic scenario, the latter has been extended with a Greedy tree-based algorithm [9] to process dynamic requests.

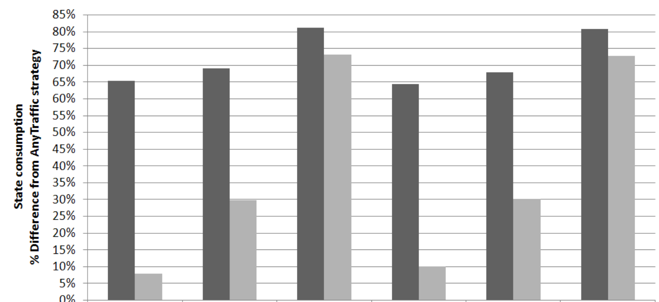
A. Experimental Setup

Experiments are executed on the ad-hoc simulator developed in [7] and here extended to operate on an event-basis. In order to determine their topological dependencies, different network topologies are considered: 37-nodes Cost266 [10]/Rand37 and 50-nodes Germany50 [10]/Rand50 networks. Rand topologies are generated from a random sequence of node degrees [11]. Each network node is an ingress-egress node generating, in a bound and discrete process, 150 (200) traffic requests for the static (dynamic) scenario. Both unicast and multicast traffic are generated within a range of two discrete traffic classes: class 1 of 2 Mbps, and class 2 of 8 Mbps. Different unicast and multicast traffic percentages are considered, namely 50%-50%, 75%-25%, and 95%-5%. For each multicast session, the size of the destination node set |M| ranges between $\log_2(N)$ and $[\log_2(N)]^2$, where N represents the number of nodes. After performing a number of experiments, we set α=0.7 and β=0.3 in the function F(x) (Eq.1), and selected the values γ=0.5 and σ=2 for the candidate deficit $\Delta C_{n,\omega}$ (Eq.5). State consumption measures the number of forwarding states required to accommodate each traffic ratio. We also define the relative gain as the percentage of performance gain in terms of either bandwidth or state consumption, attained when AnyTraffic routing is compared to AP1 and AP2. A negative gain means a loss for the AnyTraffic algorithm.

B. Results

1) Static Scenario

The simulation steps consist in i) creating the network entities (trees) for the AnyTraffic groups, and then ii) processing the unicast requests looking for the minimum cost path among the created trees. Fig.2 shows the results obtained for the Cost266 and Rand37 networks in terms of relative gain in state consumption with respect to the generated percentage of unicast and multicast traffic ratios. Bandwidth consumption figures are not shown but results analyzed here below.



	50% - 50% Cost266	75% - 25% Cost266	95% - 5% Cost266	50% - 50% Rand37	75% - 25% Rand37	95% - 5% Rand37
Approach 1	65,34%	69,02%	81,23%	64,28%	67,90%	80,83%
Approach 2	7,97%	29,75%	73,07%	10,09%	30,23%	72,80%

Unicast and Multicast Traffic Rate(%)

Fig.2: State Consumption for Cost266 and Rand37 networks

From Fig.2, AnyTraffic routing shows good performance in terms of state consumption compared to both AP1 and AP2 for the range 50% to 95% of unicast traffic. As the latter increases, the gain increases (from approximately 8 to 73% in the AP2 case). However, in terms of bandwidth consumption, AnyTraffic routing does not lead to the same performance gain due to the longer paths followed by unicast traffic. For example, when compared with AP2, it consumes from 4.4% to 7.2% more bandwidth in the Cost266 network, and from 4.5% to 5.6% in the Rand37 network. AnyTraffic routing shows better performance compared with AP1: it consumes between 25% and 80% less bandwidth than AP1. The same behavior is observed for both German50 and Rand50 networks although a bit less favorable. In terms of state consumption, the gain decreases from 32% to 6% for the range 50% to 95% of unicast traffic when compared to the Cost266 network. This observation reflects that more nodes with higher node degree influence the performance of the AnyTraffic algorithm. We also observe, for networks of identical size but lower clustering coefficient, that the algorithm performs better because it favors path aggregation at a lower deficit $\Delta P_{s,d}$.

The dependency of AnyTraffic routing with respect to the size of the group has also been studied. In terms of state consumption, although the gain is always positive for the whole set of unicast-multicast traffic pairs, a decreasing trend is observed. As the group size increases, the created AnyTraffic trees are pushed up to their limits (i.e. stretched) by becoming longer and thus requiring more states. As regards bandwidth consumption, the worst gain is never lower than

-8%. The concave shape observed for the 50%-50% and 75%-25% traffic pairs is steeper as the percentage of unicast traffic increases (as longer tree branches increase the bandwidth consumption).

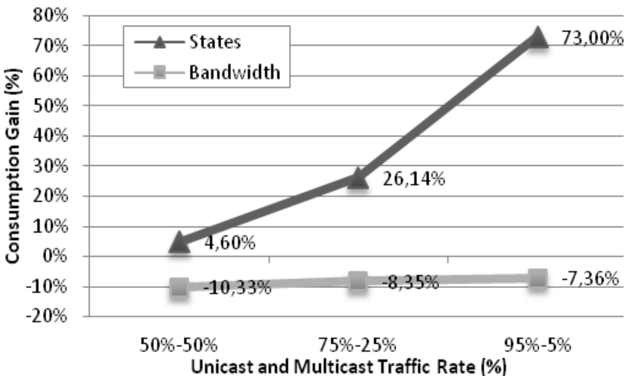


Fig.3: 100-node network: State and Bandwidth consumption gain

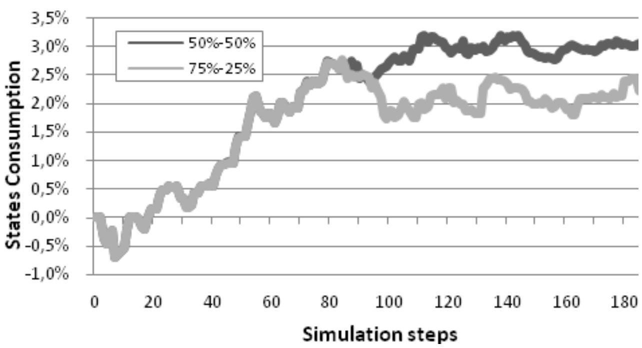


Fig.4: Cost266 network: State consumption gain for the AnyTraffic

The overall algorithm's behavior for larger topologies is close to the small topologies experimented in [7]. Fig.3 shows the performance of the AnyTraffic algorithm for a 100-node network topology. To reduce computational time, only 25% of nodes as ingress-egress nodes are considered. As regards state consumption, the algorithm demonstrates good performance compared with AP2 in the range 50%-95% of unicast traffic. The gain curve follows an exponential growth from 4% up to 70%. In terms of bandwidth consumption, AnyTraffic routing shows worst performance (up to -10%) for unicast traffic due to the longer paths followed by this traffic. Although the bandwidth consumption gain remains negative, it shows a positive trend from -10% to -7%. Indeed, as less AnyTraffic trees are created by multicast requests, forwarding unicast traffic requires more P2P shortest paths (consuming less bandwidth). Therefore, the value of -10% can be considered as an upper bound in terms of bandwidth consumption.

2) Dynamic Scenario

Simulations consist in processing several dynamic requests in which receivers join and leave an AnyTraffic group during its lifetime. Such scenario is modeled as a sequence of join and release requests where the bandwidth resources are limited to a maximum link capacity set to 10Gbps. Each simulation step represents one request processing for every node in the network. A probability that follows a non-stationary distribution is associated to each join/release request. This distribution starts with a 100%-0% join/leave probability up to a 50%-50% balanced stage, after several simulation steps. Fig.4 shows the state consumption gain of the AnyTraffic algorithm when performing with control of deterioration (Section III.C.3). The deterioration threshold to decide either to continue with on-the-fly tree setup or to perform the entire tree re-computation is set to 20%. After several simulation iterations, we set w=0.6 in Eq.6. From Fig.4, it can be observed that re-computation gain gradually grows to stabilize around 3% for the 50%-50% traffic pair and around 2% for the 75%-25% pair. The difference in the percentage of multicast requests explains this 1% gain variation. Fewer trees decrease the number of common forwarding entries for both unicast and multicast traffic. The low gain values obtained when re-computing the entire tree means that deviations from the optimum are not significant. Note that similar behavior is observed for the bandwidth consumption (not shown). In order to avoid waste of computational resource, a periodic deviation control can be performed when computing the tree on-the-fly.

V. CONCLUSION

The initial results obtained with the AnyTraffic routing algorithm, when applied to labeled-based forwarding (labels encode topological information) are promising. By stretching the shortest path for unicast traffic forwarding, common forwarding entries can be shared for both unicast and multicast traffic forwarding along the AnyTraffic tree toward the labeled topological locations, and thus the number of forwarding states significantly reduced. Future work includes i) investigation of other maximum deficit function; ii) execution of the algorithm on Internet-like topologies (power law random graphs) with increasing number of nodes up to O(10k) to further determine the dependencies of the algorithm on the node degree, and the clustering coefficient; and iii) elaborate on distributed processing (in particular, under dynamic conditions).

REFERENCES

[1] T. Wong, R. Katz, "On Analysis of Multicast Forwarding State Scalability", *IEEE Int'l Conf. Network Protocols (ICNP 2000),* Osaka, Japan, Nov. 2000.

[2] D. Thaler, M. Handley, "On the aggregatability of multicast forwarding state", *IEEE Infocom 2000*, Tel Aviv, Israel, Mar. 2000.

[3] P.I. Radoslavov, R. Govindan, D. Estrin, Exploiting the bandwidth-memory tradeoff in multicast state aggregation", *Technical Report 99-697*, Comp. Science Dep., University of Southern California, Jul. 1999.

[4] L. Blazevic, Y.-L. Boudec, "Distributed core multicast (DCM): a multicast routing protocol for many groups with few receivers", *Proc. Networked Group Communication Workshop.*, Pisa, Italy, Nov. 1999.

[5] J. Tian, G. Neufeld, "Forwarding state reduction for sparse mode multicast communications", *IEEE Infocom 1998*, San Francisco, CA, USA, Mar. 1998.

[6] D.-N. Yang, W. Liao, "Optimizing state allocation for multicast communications with explicit multicast forwarding", *IEEE Trans. Parallel Distr. Sys.*, vol. 19, no. 4, Apr. 2008.

[7] P. Pedroso et al., "Anytraffic routing algorithm for label-based forwarding", *IEEE Globecom*, Honolulu, Nov. 2009.

[8] F.K. Hwang, (Ed.), "The Steiner Tree Problem", Annals of Discrete Mathematics, vol. 53, Amsterdam, North-Holland Editions, 1992.

[9] Z. Fei, M. Ammar, E. Zegura, "Multicast server selection: problems, complexity, and solutions", *IEEE J. Select. Areas Commun.*, vol. 20, no. 7, Sep. 2002

[10] R. Inkret, A. Kuchar, B. Mikac, "Advanced infrastructure for photonic networks European research project", *Extended Final Report of COST 266 Action*, ISBN 953-184-064-4, 2003.

[11] H. Kim, (Ed.), "On realizing all simple graphs with a given degree sequence", *Discrete Mathematics*, Apr. 2008.