Two SDN multi-tree approaches for constrained seamless multicast

Constant Colombo, Francis Lepage, René Kopp, Eric Gnaedinger

Two SDN Multi-tree Approaches for Constrained Seamless Multicast

Constant Colombo
Université de Lorraine
CNRS, CRAN,
F-54000 Nancy, France
constant.colombo@univ-lorraine.fr

Francis Lepage
Université de Lorraine
CNRS, CRAN,
F-54000 Nancy, France
francis.lepage@univ-lorraine.fr

René Kopp
TDF SAS
Technical & Innovation Directorate
F-57000 Metz, France
rene.kopp@tdf.fr

Eric Gnaedinger
Université de Lorraine
CNRS, CRAN,
F-54000 Nancy, France
eric.gnaedinger@univ-lorraine.fr

***Abstract*—In Content Delivery Networks (CDN), Quality of Experience (QoE) provides two major performance indicators that are availability and continuity of service. As a consequence, network robustness has become a major concern for network operators. TDF operates a traditional transport network for video and audio transport through multicast. Any failure on the network causes a recovery time implying loss and an impact in the content viewing. This illustrate that service continuity is a direct consequence of network availability. This work aims to propose a Software Defined Networking (SDN) architecture [1] in which a central controller uses its knowledge of the performance and bandwidth allocation to compute redundant disjoint multicast trees. Two maximally independent trees carrying the same stream over the network are deployed. When a failure occurs, at least one of the trees is still active, eliminating any discontinuity on the content viewing. This paper is focused on the Path Computation Element (PCE), which is based on previous works and the Suurballe-Tarjan algorithm [2]. Two algorithms are presented in this paper, which both fulfill the requirement of the architecture.**

***Index Terms*—Routing, Disjoint trees, QoE, Seamless, Multicast, Real-time stream, Delay Constrained, SDN.**

I. Introduction

In Content Delivery Networks (CDN), robustness is one of the main problematic, as it is directly responsible for network availability and thus service continuity.

TDF is a French network operator, mainly providing transport for real-time audio and video streams nationwide. Today, one of its most important activity is audiovisual transport, streaming most of DTT channels (Digital Terrestrial Television) and national radios from studios to broadcasting antennas all over the territory. TDF operates its own network which is independent from the Internet, but can be considered a CDN.

The carried traffic is a real-time stream. As a consequence, data cannot be re-transmitted in case of loss and must be transported as fast and as steady as possible. Any method associated with TCP cannot be used in this case. Also, any loss in the network may cause a multiplied impact in the content viewing, depending on the receiving equipment: Quality of Experience (QoE) [3] is one of TDF's main concern. As service continuity is a consequence of network availability, Quality of Service (QoS) [4] is to be considered for QoE issues to be solved: commitments on availability, latency, jitter, and loss are specified for each business customer in the Service Level Agreement (SLA).

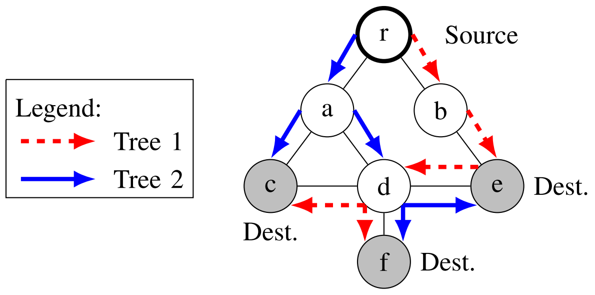


Fig. 1. Maximally independent delay constrained trees

To ensure robustness in case of failure, traditional IP solutions rely on rerouting based on the Interior Gateway Protocol (IGP). TDF current solution is the standard Multicast VPN (MVPN) [5] over MPLS on a traditional IP network. MVPN is composed of two protocols IGMP and PIM that both relies on the IGP. Those protocols ensure the building of a multicast tree and its resiliency through rerouting. Such resiliency schemes necessarily imply micro-cuts of the service, and thus impact on the content viewing. End equipment are able to buffer and absorb a part of the rerouting time, which must be inferior to the buffer size plus the delay of the path. But end equipment's' buffers are very small due to the real-time constraint, and only rerouting performance can be improved, which is very costly.

Inspired from Seamless Protection Switching (SPS) [6], we aim to propose a new architecture based on redundancy for reliable transport. SPS is a standard defining how two redundant streams should be combined into a single coherent stream, without any cut if any stream fails. Indeed, redundancy and specifically seamless switching can tend to the micro-cuts issue while assuring the continuity of service in case of failure.

The main idea of the proposed architecture is to build two maximally independent redundant trees from the source of the stream to the destinations as illustrated in Fig. 1.

Both of the trees are active and carry the data to the destination. Basically, the destination is capable of swapping seamlessly from one path stream to the other. This way, whenever a failure occurs, only one path is impacted per destination. There is no backup tree: both of them are active at the same time. The main drawback is the bandwidth utilization: carrying twice the data on disjoint trees means using much more network resources. As a consequence, one of the main goals is to minimize the total bandwidth utilization of the two trees.

As it is presented, the architecture is a proactive protection, implying offline computation of the trees. The complete robust scheme also ensure each tree robustness, accounting for multiple network failures.

Software Defined Networking (SDN) [1] is a new paradigm in which networks activities are separated in the control plane and the data plane. A controller handles decisions while network nodes only perform forwarding. This way, any functionality can be implemented seamlessly through controller programming, and various algorithms can be implemented in a centralized fashion. Fig. 2 illustrates the architecture of this paper's solution. The network is separated between three main planes : Control plane, responsible for decision elements, the Management plane, responsible for managing the network, and the Data plane which applies Control plane decisions and forwards traffic accordingly. An SDN controller usually contains Control and Management plane, while network nodes only support Data plane.

This paper is focused on the Path Computation Element (PCE) of the presented architecture. It is organized as follow: in Section II industrial constraints are described and transposed into a graph theory formulation. Section III reviews the related works on multicast reliability and the multi-tree computation problem. In Section IV, two algorithmic approaches are presented. Finally, both algorithms are experimentally evaluated in Section V.

II. MIDCMPT Problem

A. Problem statement

Multicast is a method to send data to a group of receivers in a single transmission, rather than multiple transmissions.

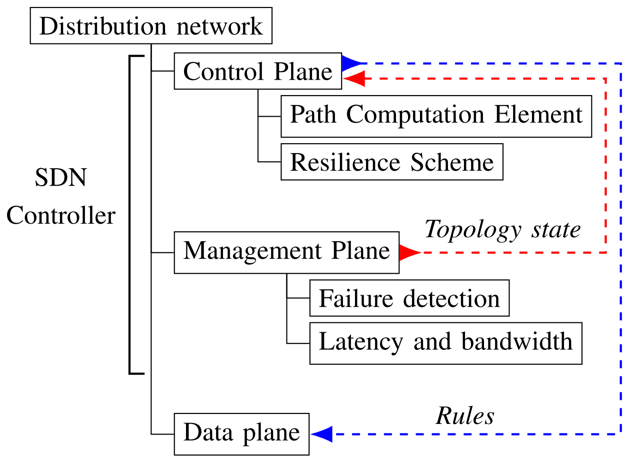


Fig. 2. Architecture Overview

To perform multicast over a given network, the creation of a tree is needed. Network topology can be modelled as a graph, and tree computation is a well-known field of graph theory. In simple cases, Spanning Trees are usually easy to compute and efficient enough.

TDF's constraint are the following: the data is a real-time continuous stream, and as such should be carried as fast as possible without loss. Also, network bandwidth should be managed: the network should be able to carry the streams and every other customer's traffic. The network is not dedicated to this single usage. As a consequence, minimising the total bandwidth usage is a major concern.

Links that do not have enough bandwidth for the stream are removed from the problem. Each edge of the graph receives a weight of 1: intuitively, saving network resources is equivalent to using as less links as possible. But this weighting scheme does not account for existing traffic nor network heterogeneity. A naive approach would be to use bandwidth consumption as weight, but given the heterogeneity of the network, it should be considered to prioritise some links : for example, a 10 Gb/s backbone link used at 10% should be less costly than a half full 1 Gb/s access link, even though the bandwidth consumption is greater.

As stated before, the real-time streaming constraint implies a constrained delay between the source and each destination. Each edge of the graph is set with the maximum link delay measured over a significant period. This ensures that over the real network, constraint is not violated.

At any time, any destination should be reached by two disjoint paths carrying the same stream, to ensure that in case of failure the stream reaches destination. This is called path diversity. More redundancy could be considered : if a destination si reached by more than two path, it is even less likely to be unreachable in case of link failure. But in the proposed architecture, all paths carry the stream, and thus consume bandwidth. Using more than two paths is too costly in network usage compared to the robustness gain.

Given the gathered data for the studied network, links are more likely to fail than nodes. Paths are then only required to be edge-disjoint. Node-disjointedness is stronger, because node-disjoint paths are inherently edge-disjoint. Nevertheless, the proposed algorithm can ensure node-disjointedness.

Given real network topologies, disjoint paths are not always possible. For geographic reasons, some areas may not be reachable by two distinct paths. This means a Single Point of Failure exists for such areas of the network, and cannot be bypassed. As a consequence, disjointedness is not required to be strict. The problem is then stated as finding Maximally Independent Delay Constrained Minimal Pair of Trees (MIDCMPT).

B. $\mathcal{NP}$-hardness

The MIDCMPT problem is $\mathcal{NP}$-hard and this section focuses on its proof. First, the Maximally Independent Delay Constrained Paths problem (MIDCP) is shown to be $\mathcal{NP}$-hard by reduction to the Maximally Independent Paths problem

(MIP). Then, MIDCP is generalised to MIDCMPT. Node and edge-disjointedness are similar from the problem formulation point of view.

The MIDCP problem can be formulated as follows : for a graph G_0, find a pair of (s,t)-paths such as those path are maximally independent, delay constrained and lightest possible. An auxiliary graph G_1 is created by removing from G_0 all edges belonging only to (s,t)-paths violating the delay constraints. One can solve the MIDCP problem over G_0 by solving the MIP problem over G_1. Since the MIP problem is known to be $\mathcal{NP}$-hard [7], then by reduction the MIDCP is $\mathcal{NP}$-hard.

MIDCP is a specific case of the MIDCMPT problem. Using the following MIDCMPT formulation, it appears that MIDCP is the $N = 1$ case : find a pair of trees from a source to N destinations such as those trees are maximally independent, delay constrained and lightest possible. By generalisation of MIDCP which is $\mathcal{NP}$-hard, the MIDCMPT problem is $\mathcal{NP}$-hard.

III. Related works

A. On Multicast reliability

Network survivability has been studied in [8] within the same context, providing tools for modelling MVPN solutions and evaluating its reliability. It shows that even though improvement of the traditional architecture is possible, impact still exists.

In [9], several paths are deployed proactively on network nodes, and activated quickly in case of failure, thanks to OpenFlow Fast Failover groups [10]. This solution improves service availability. However, this scheme uses multiple paths, which is not as bandwidth-efficient as a multicast distribution.

[11] proposes a scheme based on multiple trees to ensure reliable multicast distribution in a datacenter. The main point is to avoid loss and enable rapid path rerouting. Constraints and goals may differ from our problem, but the approach is similar, though our proposed scheme uses simultaneously the two multicast trees.

A genetic algorithm is proposed in [12] to deploy two spanning trees on Networked Controlled Systems to ensure communication reliability. The usage of a similar method will be considered in future works.

B. On MIDCMPT computation

To the best of our knowledge, no method has yet been proposed to compute MIDCMPT. To solve this problem, many heuristics have been proposed such as in [13], and even an exact solution in [14].

[15] provides an algorithm to compute two trees in such a way that a node will always be reachable through one of the trees in an edge redundant network. It can account for costs and for a delay constraint. The first drawback of this work is that it is based on undirected graph, and cannot account for different costs on each direction. Given the industrial context of unidirectional transmission, network links are to be considered directed. The second drawback is that this algorithm works on an edge redundant graph, which means that, for any given pair of vertices, there exists at least two strictly independent paths between those vertices. In practice this is not the case.

[16], [17], [18] intended to provide a new algorithm, based on directed graphs and resulting in a close to optimal solution. This method does not account for delay constraint and works on graphs with specific properties. Author's previous work intended to propose a new version of this algorithm, delay constrained and graph agnostic. Adaptation of [16], [17], [18] was not trivial, had a high computational cost and results showed that the algorithm often did not provide valid solutions.

IV. Proposed algorithms

A. Red Tree First - RTF

The concept of this algorithm is quite simple: a first delay constrained tree is computed iteratively using a delay constrained shortest path algorithm. Then a second tree is computed accounting for the first tree.

1) Delay constrained shortest path algorithm: Two delay constrained shortest path algorithm are selected for this work : Lagrange Relaxation based Aggregated Cost (LARAC) [19] based on Dijkstra algorithm [20] and Constrained Bellman-Ford algorithm (CBF) [21] based on Bellman-Ford algorithm [22].

The LARAC procedure can be described as follows : for a given delay constraint, Dijkstra algorithm is applied iteratively to reweighted versions of the graph until weight and delay conditions are met. The weighting depends on previously computed paths, weight and delay.

The CBF algorithm is a breadth-first search, discovering paths with increasing delay while recording the shortest path to each node visited, until the exploration delay exceeds the delay constraint.

2) RTF explanation: The Red Tree First (RTF) procedure consists in computing one tree first, then the second one. A tree is computed as follows : for each destination not yet reached, a delay constrained shortest path algorithm is applied from source. Only the lightest path found is added to the tree. Every edge of the graph belonging to the tree receives a null weight. Delay is not altered. This operation is repeated until all destinations are reached by this first tree. Once the tree complete, all edges of the graph belonging to the tree receive a maximal weight. This maximal weight is the total weight of the graph. The second tree is then calculated using the same process over the modified graph.

RTF can also aim for maximal node-disjointedness. Before the second tree computation, all incoming edges of nodes belonging to the first tree must receive the maximal weight.

This procedure tends to find lightest trees under delay constraint. Independence is not guaranteed, as the lightest path is not always part of the maximally independent pair of paths. Nevertheless, by weighting the edges of the first tree with the total weight of the graph, the procedure ensures that those links will be used as last resort to build the second tree. This is easily proven, as any other path is lighter than the complete graph.

This scheme ensures that two trees are found, fitting the delay constraint. Independence and cost are accounted for, but not a priority. The proposed RTF procedure is detailed in Algorithm 1, and illustrated in Fig. 3.

Algorithm 1 Red Tree First algorithm

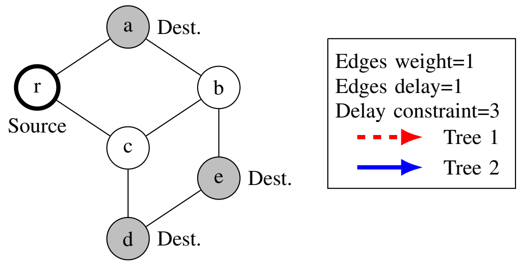
Input: A directed graph $G = (V, E)$ weighted with cost C and delay D, a source s, a set of destinations T, a delay constraint Δ.

Output: Two delay constrained maximally edge-disjoint lightest trees A_r, A_b from s to every $t \in T$.

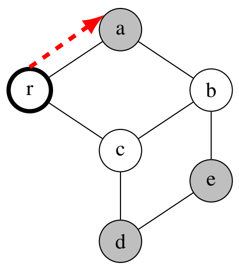
```
1:  $T_r \leftarrow T$
2:  while $T_r \neq \emptyset$ do
3:      $P_{min} \leftarrow E$
4:      for $t \in T_r$ do
5:          $P_t \leftarrow LARAC(s, t, \Delta)$
6:          if $C(P_t) \leq C(P_{min})$ then
7:              $P_{min} \leftarrow P_t$
8:              $t_{min} \leftarrow t$
9:          end if
10:     end for
11:     $T_b \leftarrow T_r - \{t_{min}\}$
12:     $A_r \leftarrow A_r \cup P_{min}$
13:     for $e \in P_{min}$ do
14:         $C(e) \leftarrow 0$
15:     end for
16: end while
17: for $e \in A_r$ do
18:     $C(e) \leftarrow \sum_{i \in E} C(i)$
19: end for
20: $T_b \leftarrow T$
21: while $T_b \neq \emptyset$ do
22:     $P_{min} \leftarrow E$
23:     for $t \in T_b$ do
24:         $P_t \leftarrow LARAC(s, t, \Delta)$
25:         if $C(P_t) \leq C(P_{min})$ then
26:             $P_{min} \leftarrow P_t$
27:             $t_{min} \leftarrow t$
28:         end if
29:     end for
30:     $T_b \leftarrow T_b - \{t_{min}\}$
31:     $A_b \leftarrow A_b \cup P_{min}$
32:     for $e \in P_{min}$ do
33:         $C(e) \leftarrow 0$
34:     end for
35: end while
```

B. Iterative SHERPA - IS

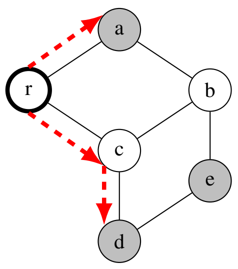
The second proposed algorithm is based on a multipath algorithm SHaring-Edges Restrained PAths (SHERPA) [23]. IS computes iteratively pairs of paths using SHERPA, and assigns them to the trees depending on the maximum independence affectation.



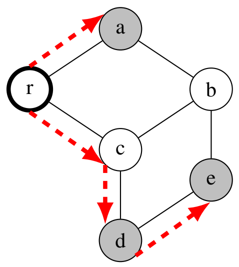
(a) Base graph



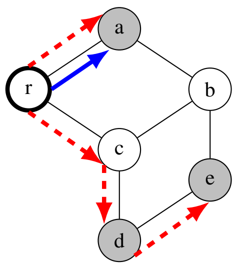
(b) Closest node a



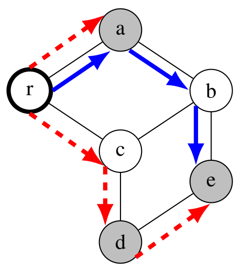
(c) Closest node d



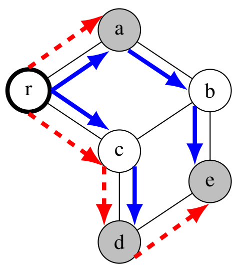
(d) Closest node e



(e) Closest node a



(f) Closest node e



(g) Closest node d

Fig. 3. RTF Illustration

1) SHERPA: This algorithm is to be published by the authors in [23], but this section describe the main idea. In a graph $G = (V, E)$, for a given pair $(s, t) \in V \times V$, the SHERPA algorithm finds two delay constrained maximally edge-disjoint paths from s to t. It is based on the Suurballe-Tarjan algorithm [2], which does not account for delay constraint and ensures only strict disjointedness. In short, the alterations to this original algorithm are: the replacement of the shortest path algorithm by a delay constrained algorithm; a reweighting scheme instead of the deletion of some links to provide relaxed disjointedness.

Based on Suurballe-Tarjan algorithm variation for node-disjointedness, SHERPA also propose a variation to find delay constrained maximally node-disjoint paths. This variation can be used in IS to find for node-disjoint trees.

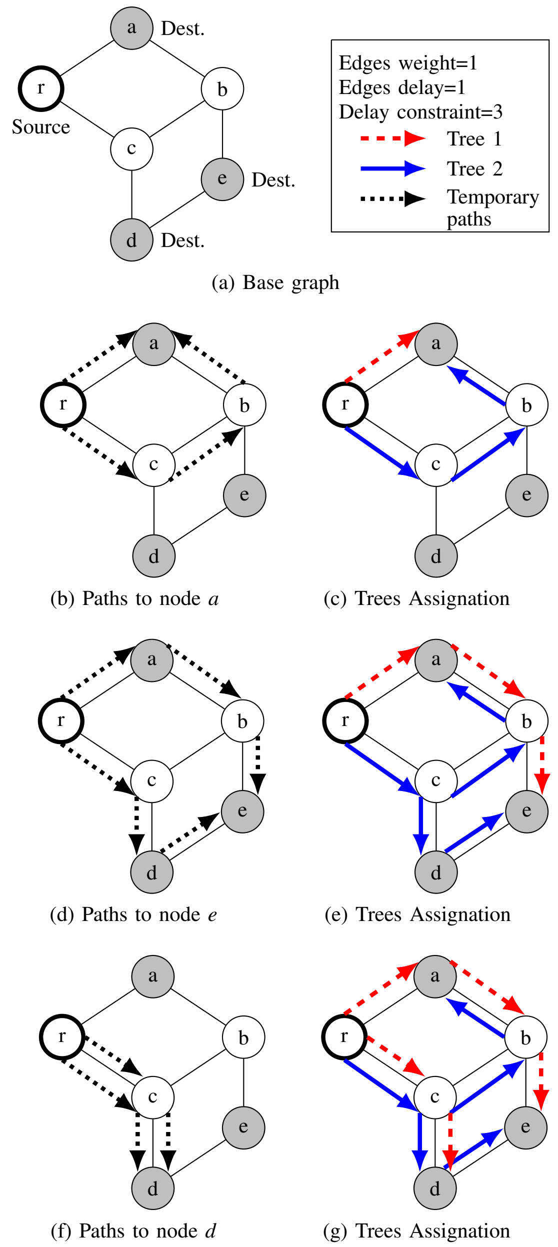
2) IS explanation: The IS procedure iterates SHERPA algorithm instances as follows : for each destination not yet reached by the trees, SHERPA algorithm is applied from the source to find a pair of Maximally Independent Delay Constrained Lightests Paths. Each path is assigned to a tree, depending on which affectation provides maximal independence. When no decision can be made for a given destination (i.e. provided independence is equal for both affectations), it is skipped and treated later. If no decision can be made for any of the remaining destinations, one random affectation is made, and the process restarted for every other not reached destinations. This process is repeated until all destinations are reached. Note that the order in which destinations are treated does not matter on the final result.

Algorithm 2 Iterative SHERPA algorithm

```
Input: A directed graph $G = (V, E)$ weighted with cost $C$
    and delay $D$, a source $s$, a set of destinations $T$,
    a delay constraint $\Delta$.
Output: Two delay constrained maximally edge-disjoint
    lightest trees $A_r, A_b$ from $s$ to every $t \in T$.
 1: $A_r \leftarrow \emptyset$
 2: $A_b \leftarrow \emptyset$
 3: $previous \leftarrow |T|$
 4: while $T \neq \emptyset$ do
 5:   if $|T| = previous$ then
 6:     $u \leftarrow rand(T)$
 7:     $P_1, P_2 \leftarrow SHERPA(s, u, \Delta)$
 8:     $A_r \leftarrow A_r \cup P_1$
 9:     $A_b \leftarrow A_b \cup P_2$
10:   end if
11:   $previous \leftarrow |T|$
12:   for $t \in T$ do
13:     $P_1, P_2 \leftarrow SHERPA(s, t, \Delta)$
14:     if $|(A_r \cup P_1) \cap (A_b \cup P_2)| > |(A_r \cup P_2) \cap (A_b \cup P_1)|$
        then
15:       $A_r \leftarrow A_r \cup P_1$
16:       $A_b \leftarrow A_b \cup P_2$
17:       $T \leftarrow T - \{t\}$
18:     else if $|(A_r \cup P_1) \cap (A_b \cup P_2)| < |(A_r \cup P_2) \cap (A_b \cup P_1)|$
        then
19:       $A_r \leftarrow A_r \cup P_2$
20:       $A_b \leftarrow A_b \cup P_1$
21:       $T \leftarrow T - \{t\}$
22:     end if
23:   end for
24: end while
```

Node disjointedness can be achieved by using the node-disjoint variation of SHERPA algorithm, and to check for common vertices instead of common edges for path assignation.

The proposed IS scheme ensures maximal path diversity for every destination. Path cost is minimized, but total tree cost is not guaranteed to be minimized, as path affectation only depends on path diversity. The complete procedure is detailed in Algorithm 2, and illustrated in Fig. 4.



(a) Base graph

(b) Paths to node *a*

(c) Trees Assignation

(d) Paths to node *e*

(e) Trees Assignation

(f) Paths to node *d*

(g) Trees Assignation

Fig. 4. IS Illustration

C. *Validity of the solutions*

Nothing guarantees the absence of cycle or the uniqueness of path from source to a vertex in a tree. That is to say a solution provided by one of the algorithms is a pair of valid trees. The only guarantee at this point is that the delay constraint is respected on at least one of the paths for each destination.

To ensure the validity of the results, two steps are necessary : removing cycles, and selecting unique paths.

Cycles can be easily found through Depth First Search (DFS). When an already visited node is met by the DFS, this means the last edge belongs to a cycle, goes towards the source, and should be removed. The DFS is then restarted. A specific case is met when the cycle is between two nodes, with an edge in each direction. These 2-length cycles are first treated as any other cycle, but kept in memory. It has been observed in practica that such cases necessitates a specific treatment. After treating all cycles, a procedure must be applied to each 2-length cycle met to select which direction must be kept. Algorithm 3 details the procedure. Basically, either one direction is needed to keep the tree continuous, or the shortest delay path between the two nodes is kept, to ensure that the delay constraint is not violated. This scheme always works if a single 2-length cycle is met, but might fail when multiple 2-length cycle appear.

Algorithm 3 2-length cycles elimination procedure

```
Input: A directed graph $G = (V, E)$ weighted with cost $C$
    and delay $D$, a source $s$, a set of destinations $T$,
    a delay constraint $\Delta$. A pseudo-tree $A \in E$ without cycle.
    A list $B$ of 2-length cycles.
Output: A pseudo-tree $A$ without cycle.
1:  $P_A(v)$ denotes the predecessors of $v$ in set of edges $A$
2:  for $(x, y) \in B$ do
3:      $A \leftarrow A \setminus (x, y), (y, x)$
4:      if $|P_A(x)| = 0$ then
5:          $A \leftarrow A + (y, x)$
6:      else if $|P_A(x)| = 0$ then
7:          $A \leftarrow A + (x, y)$
8:      else
9:          $s_x \leftarrow$ Shortest delay path from $s$ to $x$
10:         $s_y \leftarrow$ Shortest delay path from $s$ to $y$
11:         $d_x \leftarrow D(s_y) + D((y, x))$
12:         $d_y \leftarrow D(s_x) + D((x, y))$
13:         if $d_x < d_y$ then
14:             $A \leftarrow A + (y, x)$
15:         else
16:             $A \leftarrow A + (x, y)$
17:         end if
18:     end if
19: end for
```

Once cycles are removed, to ensure the validity of the results, unique path to destinations must be selected. For any vertex with more than two predecessor in a tree, the lightest-delay path from source is selected and only the matching predecessor is kept. All other unnecessary edges are removed. Algorithm 4 details the procedure.

In summary, to obtain a valid solution one must follow the process:

- DFS to find and break cycles while keeping in memory 2-length cycles met;
- Treat 2-length cycles using Algorithm 3;
- Eliminate multiple paths using Algorithm 4.

Algorithm 4 Multiple path elimination procedure

```
Input: A directed graph $G = (V, E)$ weighted with cost $C$
    and delay $D$, a source $s$, a set of destinations $T$,
    a delay constraint $\Delta$. A pseudo-tree $A \in E$ without cycle.
Output: A valid tree $A$ from $s$ to every $t \in T$.
1:  Function single_path($v$)
2:  $P_A(v)$ denotes the predecessors of $v$ in set of edges $A$
3:  for $p \in P_A(v)$ do
4:      $d[p] \leftarrow 0$
5:      $c \leftarrow p$
6:      while $c \neq s$ do
7:          if $|P_A(c)| > 1$ then
8:              single_path($c$)
9:          end if
10:         We have $P_A(c) = \{b\}$
11:         $d[p] \leftarrow d[p] + D(b, c)$
12:         $c \leftarrow b$
13:     end while
14: end for
15: $u \leftarrow \min_{p \in P_A(v)}(d[p])$
16: for $p \in P_A(v) \setminus \{u\}$ do
17:     $A \leftarrow A \setminus \{(p, v)\}$
18: end for
19: end Function
20:
21: Let $M$ be the set of vertices with more than one prede-
    cessor in A
22: while $M \neq \emptyset$ do
23:     single_path($rand(M)$)
24:     $M$ is re-estimated
25: end while
26: Let $L_A \in V$ be the set of leaves of $A$
27: $N \leftarrow L_A \setminus T$
28: while $N \neq \emptyset$ do
29:     for $n \in N$ do
30:         Let $E_n \in E$ be the set of edges to vertexs $n$
31:         $A \leftarrow A \setminus E_n$
32:     end for
33:     Let $L_A \in V$ be the new set of leaves of $A$
34:     $N \leftarrow L_A \setminus T$
35: end while
```

V. Algorithm Evaluation

A. Experimentation Scheme

The two proposed algorithms are here evaluated in terms of performance and scalability.

1) Problem generation: Random graphs are generated given a number of nodes following an Erdõs-Rényi model. Three types of instances will be studied : small graphs $n = 20$, medium graphs $n = 100$ and TDF's network scale large graph $n = 800$.

Random links are added between connected component to ensure the connectivity of the whole graph. It could be argued that this is not the best model for networks. Statistical significance is discussed later.

The probability p of edge creation is chosen according to the expected number of edges. For small graphs, different degrees will be used. Sparse and dense graphs will be generated with $p = 0.1$ and 0.6 for $n = 20$. Medium graphs will be generated with $p = 0.01$ for $n = 100$. For large graph, around 1600 edges are wanted, which matches $p = 0.002$, to reflect a topology fo the same scale as TDF's network.

Edges are randomly assigned a class to reflect the heterogeneity of the topology, as stated in II. Each class matches an intervals for random delay and a weight.

One node is randomly selected as source and a set of different nodes as destinations. Around 10% of nodes are selected as destinations. Such random selection is not representative of a hierarchical network. On a Internet network, source and destinations are usually on the edge of the network while on an operator network such as the studied one, source is located on the network's core while the destinations are on the edge.

Network core is usually a highly connected part of the network while network's edge are less connected.

Using a heuristic such as the one proposed in [24], it is possible to identify the "denser" region of the graph, that can be assimilated to the network's core. If the source is selected in this region while the destinations are selected outside of it, the problem is closer to the studied case. One could go even further and use classification or partition methods, but problem generation is not the core of this work.

To determine the delay constraint value, an assumption is made : the network is designed such as all nodes are reachable from source under delay constraint. Among the shortest-delay paths to every node, the one with the higher value still fits the delay constraint. It is a lower bound of the delay constraint, and it is chosen as constraint for the corresponding graph.

This way, a problem is guaranteed to have at least one solution, where the two trees are strictly identical and match the shortest-delay tree from source to each destination.

2) Indicators and significance: The observed performance indicators are :

- Computation time;
- Success : is a valid constrained tree found;
- Tree sharing: the proportion of shared edges of the smaller tree, as it will be the higher proportion;
- Which algorithm gives a better result: this is estimated thanks to an objective function defined in equation (1).

Let β denote the number of common edges between the trees. Objective function is defined by :

$$\mathcal{F}(A_r, A_b) = C(A_r) + C(A_b) + \beta * 2 \sum_{i \in E} C(i) \qquad (1)$$

It is easy to prove that this objective function aims for greater independence before lighter trees. All tree found by the algorithms match the delay constraint, that's why it does not appear in the function.

Experiments were led on a 2 core 4 Thread 2.9GHz CPU and 16 Go RAM machine, code implemented in Python 2.7.

TABLE I
ALGORITHMS PERFORMANCE (INTERVALS AT 99%)
PER INSTANCE SIZE

Instance Size (n/p)	20 0.1	20 0.6	100 0.01	800 0.002
Number of edges	45	228	199	1689
±	0.4	1	0.1	2
IS Computation Time (s)	0.008	0.02	0.40	9.45
±	10^{-4}	10^{-3}	10^{-2}	0.3
RTF Computation Time (s)	0.006	0.02	0.22	361
±	10^{-4}	10^{-4}	10^{-3}	8
IS success (%)	99	100	99	90
±	0.3	0	0.3	2
RTF success (%)	99	100	99	80
±	0.3	0	0.3	3
IS Tree sharing (%)	80	27	98	72
±	2	3	0.4	0.4
RTF Tree sharing (%)	80	29	99	72
±	2	3	0.4	0.4
Similar solution found (%)	91	73	93	0
±	2	4	2	0
IS strictly better (%)	7	18	7	7
±	2	0.3	2	2
RTF strictly better (%)	2	9	1	93
±	2	0.2	0.7	2

B. Results

Table. I illustrates mean performance confidence interval at 99% probability for the tested characteristics over different instances sizes.

Over N problem instance, let X denote the tested performance indicator and $\overline{X}$ its mean. Tested population is random, of high size and its standard deviation is unknown. For an small sample $30 < n < 0.05 * N$ it comes that the $1 - \alpha$ confidence interval is defined by :

$$\overline{x} - \frac{L}{2} \leq \overline{X} \leq \overline{x} + \frac{L}{2} \qquad (2)$$

$$\frac{L}{2} = t_{1-\frac{\alpha}{2},n-1} * \sqrt{\frac{\sum_{i=0}^{n}(x_i - \overline{x})^2}{n(n-1)}} \qquad (3)$$

with $x_i, i \in [1; n]$ the sample values, $\overline{x}$ the sample mean and $t_{1-\frac{\alpha}{2},n-1}$ the Student variable.

Confidence intervals of a proportion P of the population are defined by :

$$\overline{p} - \frac{L}{2} \leq \overline{P} \leq \overline{p} + \frac{L}{2} \qquad (4)$$

$$\frac{L}{2} = u_{1-\frac{\alpha}{2}} * \sqrt{\frac{\overline{p} * (1 - \overline{p})}{n}} \qquad (5)$$

with $\overline{p}$ the sample proportion and $u_{1-\frac{\alpha}{2}}$ the normal variable. Each instance type was generated and tested 1000 times.

C. Analysis

According to data, both of the algorithm perform well over small graphs. Few differences are observed, although IS is slightly better than RTF for a similar computation time. Overall, computation time and success rate are very satisfying.

On larger graphs, matching topologies of the same scale as TDF's network, significant differences can be observed.

First, the computation time is around fourty times higher for RTF than for IS. This is due to the fact that each iteration, RTF evaluates multiple possibilities, and its complexity grows faster than IS. On the other hand, RTF often gives better results than IS, for a similar tree independence. This means that the solution found is close to the same local optimum, but RTF often finds closer results. Finally, success rate drops significatively for both algorithms.

In any case, both algorithm provide a solution in reasonnable operationnal time. Note that when a solution is not found, it is because of the appearance of more than one 2-length cycles in the process that is not yet well accounted for. A more complex solution could raise the success rate, while increasing the computation time.

D. Weaknesses of the evaluation

The main weakness of this experimental scheme is the network model itself. A first approach to improve the scheme is to make Erdõs-Rényi parameters vary to observe many network types, but this creates a large number of different tests types. Plus, as shown in [25], random graphs are not an ideal way to evaluate algorithm as they may not be representative. As a consequence, it could be argued that our results are not completely representative of algorithm performance.

VI. Conclusion

In this paper we presented a new SDN architecture for Seamless Multicast distribution, which is based on redundant trees to ensure that a real-time stream is carried without interruption from a source to multiple destinations. We proposed two centralized algorithms called RTF and IS to compute maximally edge or node independent delay constrained trees and evaluated their performance. Both algorithms account for global bandwidth utilization. We showed that, though these algorithm do not guaranty an optimal solution, they meet the requirement of the multiple constraints within reasonable time. RTF complexity grows faster with higher instances, but it gives more optimal results in terms of bandwidth consumption.

The mentionned architecture must guaranty continuity of service in case of failure in order to be more efficient than traditional solutions in terms of QoE. This paper detailed the PCE element, and future works will aim to complete the architecture : trees resiliency, failure recovery behaviour, management plane elements...

References

[1] S. Denazis, E. Haleplidis, J. H. Salim, O. Koufopavlou, D. Meyer, and K. Pentikousis, "RFC 7426 : Software-Defined Networking (SDN): Layers and Architecture Terminology," IETF, Tech. Rep., 2015.

[2] J. W. Suurballe and R. E. Tarjan, "A quick method for finding shortest pairs of disjoint paths," *Networks*, vol. 14, no. 2, pp. 325–336, 1984.

[3] "E.800: Terms and definitions related to quality of service and network performance including dependability," ITU-T, Tech. Rep., 2009.

[4] "G.1080: Quality of experience requirements for IPTV services," ITU-T, Tech. Rep., 2008.

[5] E. C. Rosen and R. Aggarwal, "RFC 6513 Multicast in MPLS/BGP IP VPNs," IETF, Tech. Rep., 2012.

[6] "ST 2022-7:2013 - SMPTE Standard - Seamless Protection Switching of SMPTE ST 2022 IP Datagrams," SMPTE, Tech. Rep. 2022-7:2013, 2013.

[7] M. T. Omran, J.-R. Sack, and H. Zarrabi-Zadeh, "Finding paths with minimum shared edges," *Journal of Combinatorial Optimization*, vol. 26, no. 4, pp. 709–722, Nov. 2013.

[8] S. Pirlot, E. Gnaedinger, R. Kopp, and F. Lepage, "IP/MPLS network modeling using Bayesian networks to improve double failure recovery," *2015 International Conference on Industrial Engineering and Systems Management (IESM)*, 2015.

[9] A. Ghannami and C. Shao, "Efficient Fast Recovery Mechanism in Software-Defined Networks," *The 11th International Conference for Internet Technology and Secured Transactions (ICITST -2016)*, 2016.

[10] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: Enabling Innovation in Campus Networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.

[11] X. Xiong and T. Chen, "MTM: A Reliable Multiple Trees Multicast for Data Center Network," *Networking, Architecture, and Storage (NAS), 2017 International Conference on*, 2017.

[12] S. Kubler, J. Robert, J.-P. Georges, and . Rondeau, "Dual path communications over multiple spanning trees for networked control systems," *Engineering Applications of Artificial Intelligence*, vol. 25, no. 7, pp. 1460–1470, Oct. 2012.

[13] R. Forsati, M. Mahdavi, A. T. Haghighat, and A. Ghariniyat, "An efficient algorithm for bandwidth-delay constrained least cost multicast routing," in *Electrical and Computer Engineering, 2008. CCECE 2008. Canadian Conference on.* IEEE, 2008, pp. 001 641–001 646.

[14] G. Feng, "Delay constrained multicast routing: What can we learn from an exact approach?" in *Global Communications Conference (GLOBECOM), 2012 IEEE.* IEEE, 2012, pp. 2809–2814.

[15] M. Mdard, S. G. Finn, R. A. Barry, and R. G. Gallager, "Redundant Trees for Preplanned Recovery in Arbitrary Vertex-Redundant or Edge-Redundant Graphs," *IEEE/ACM TRANSACTIONS ON NETWORKING*, vol. 7, no. 5, 1999.

[16] Y. Bejerano and P. V. Koppol, "Optimal construction of redundant multicast trees in directed graphs," in *INFOCOM 2009, IEEE.* IEEE, 2009, pp. 2696–2700.

[17] Y. Bejerano, S. Jana, and P. V. Koppol, "Efficient Construction of Directed Redundant Steiner Trees," in *Local Computer Networks (LCN), 2012 IEEE 37th Conference on.* IEEE, 2012, pp. 119–127.

[18] Y. Bejerano and P. V. Koppol, "Link-coloring based scheme for multicast and unicast protection," in *High Performance Switching and Routing (HPSR), 2013 IEEE 14th International Conference on.* IEEE, 2013, pp. 21–28.

[19] A. Juttner, B. Szviatovski, I. Mcs, and Z. Rajk, "Lagrange relaxation based method for the QoS routing problem," in *INFOCOM 2001. Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, vol. 2. IEEE, 2001, pp. 859–868.

[20] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Mathematik*, vol. 1, no. 1, pp. 269–271, Dec. 1959.

[21] R. Widyono, *The design and evaluation of routing algorithms for real-time channels.* International Computer Science Institute Berkeley, 1994.

[22] R. Bellman, "On a routing problem," *Quarterly of Applied Mathematics*, vol. 16, no. 1, pp. 87–90, Apr. 1958.

[23] C. Colombo, F. Lepage, R. Kopp, and E. Gnaedinger, "Sherpa : a SDN Multipath Approach to Eliminate Resilience Impact on Video Streams," Oct. 2018, to appear in ICCT 2018 proceedings.

[24] M. Charikar, "Greedy Approximation Algorithms for Finding Dense Components in a Graph," in *Approximation Algorithms for Combinatorial Optimization*, G. Goos, J. Hartmanis, J. van Leeuwen, K. Jansen, and S. Khuller, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2000, vol. 1913, pp. 84–95.

[25] M. E. J. Newman, "Random graphs as models of networks," in *Handbook of Graphs and Networks*, S. Bornholdt and H. G. Schuster, Eds. Weinheim, FRG: Wiley-VCH Verlag GmbH & Co. KGaA, Dec. 2004, pp. 35–68.