SHERPA: A SDN multipath approach to eliminate resilience impact on video streams

Constant Colombo, Francis Lepage, René Kopp, Eric Gnaedinger

SHERPA: a SDN Multipath Approach to Eliminate Resilience Impact on Video Streams

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 network, 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. 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 the controller uses its knowledge of the performance and bandwidth allocation to compute redundant disjoint paths. Two maximally link-independent paths carrying the same stream over the network are deployed. When a failure occurs, at least one of the paths is still active, eliminating any discontinuity on the content viewing. This paper is focused on the Path Computation Element, which is based on Suurballe-Tarjan algorithm [2].**

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

I. Introduction

In Content Delivery Network, 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).

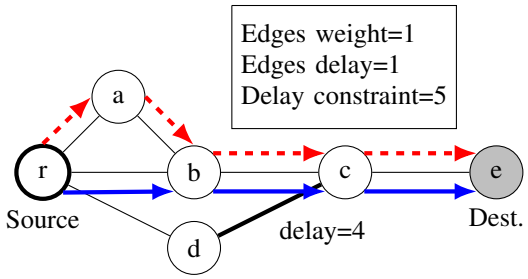


Fig. 1. Maximally independent delay constrained paths

To ensure robustness in case of failure, traditional IP solutions rely on rerouting based on the Interior Gateway Protocol (IGP). 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) [5], 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 paths from the source of the stream to the destination as illustrated in Fig. 1.

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

As it is presented, the architecture is a proactive protection,

implying offline computation of the paths. The complete robust scheme also ensure each paths robustness, accounting for multiple network failures.

Note that such multipath computation problems can be found in the literature for different applications like ECMP [6], Multipath TCP [7] or packet scheduling to gain performance [8]. Other issues addressed by multipath schemes are presented in [9], as well as four major requirements for such schemes: scalability, path disjointedness, computational complexity and path cost.

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. OpenFlow [10] is nowadays one of the most popular SDN communication protocol implementation. Fig. 2 illustrates the architecture of this paper's solution.

This paper is focused on the Path Computation Element of the proposed 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 the multipath computation problem. In Section IV, an algorithmic approach called SHERPA is explained. Algorithm performance is evaluated in Section V. Finally, Section VI reviews the other elements of the complete architecture.

II. Problem Statement

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 shall be managed so that not only the path can carry the streams, but also every other customer's traffic. 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.

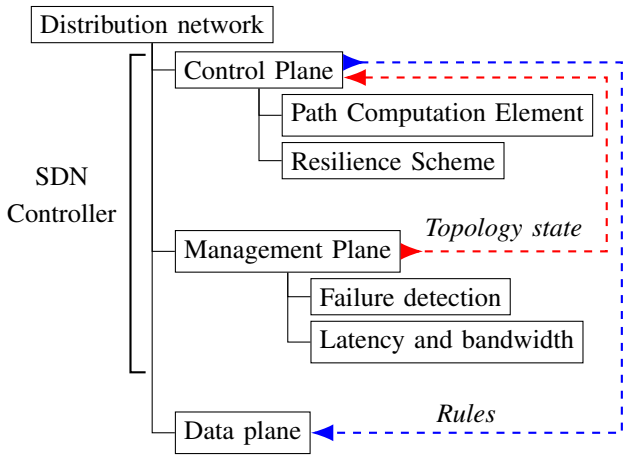


Fig. 2. Architecture Overview

[9] presents four major requirements for multipath schemes: scalability, path disjointedness, computational complexity and path cost. In this paper, a path delay constraint is added, and path disjointedness is only required to be maximal (as opposed to strict).

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

At any time, the 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.

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. As a consequence, disjointedness is not required to be strict. The problem is then stated as finding Maximally Independent Delay Constrained Paths.

III. Related works

It is shown that the problem of finding maximally disjoint paths is NP-hard in [11]. An interesting review of the subject can be found in [12].

One of the first algorithm for the disjoint path problem is the Suurballe-Tarjan algorithm [2]. As this work presents a variation of this algorithm, further details will be given later.

The MADSWIP algorithm [13] is the closest solution to the problem addressed in this paper: it finds for a pair of maximally disjoint paths, optimized in cost, and constrained by a minimum required bandwidth for each path. The main difference in the studied case is that cost accounts for the bandwidth, and the constraint is on the total delay of each path. Similarly, the algorithms presented in [14] find solutions for the strict disjointedness problem with constraints.

The subject of multipath computation is also widely covered in the literature for MANETs (Mobile Ad-hoc NETworks). Those networks are dynamic, highly connected, and control is decentralized. This has the following consequences:

- proposed solutions are usually distributed protocols [15] or distributed algorithm [16];
- path disjointedness is treated as a strict constraint, given that it is often possible in such networks as in [17];

- a single weight constraint is considered as in [18]. Delay constraint is accounted for as a weight to be minimized, not as an upper limit.

IV. SHERPA

A. Useful elements

To fully understand the algorithm presented in this paper, the following algorithms require a short description.

1) Suurballe-Trajan Algorithm: The proposed algorithm is based on Suurballe-Tarjan algorithm [2]. In a graph $G = (V, E)$, for a given pair $(s, t) \in V \times V$, the algorithm finds, if possible, two totally edge-disjoint paths from s to t.

The main steps of this algorithm are presented in Algorithm 1.

Algorithm 1 Suurballe-Tarjan algorithm

Input: A graph $G = (V, E)$, a source s, a destination t.
Output: Two totally edge-disjoint shortest paths from s to t.

1: Using Dijkstra algorithm [19], a tree T is built containing the shortest path from s to every node $v \in V$.
   This is called a shortest path tree (SPT).
   The cost of each path is noted $d_0(v)$.
   By definition it contains P_1 the shortest path from s to t.
2: Graph edges' weights are replaced as follow: for each edge $(a, b) \in E$ the new weight is:
   $w_1(a, b) = w_0(a, b) - d_0(b) + d_0(a)$
   with w_0 the weight of an edge in the original graph,
   and d_0 the shortest distance from source to the node.
3: Edges following path P_1 towards s are removed, and the direction of the edges belonging to P_1 is reversed.
   The produced graph is called Residual Graph.
4: Using Dijkstra algorithm, a path P_2 from s to t is found in this new graph.
5: Edges that exists in opposite directions in P_1 and P_2 are pruned.
6: $\{P_1, P_2\}$ contains the two paths.

In short, the algorithm finds a shortest path using Dijkstra's algorithm, and removes it from the graph. The reversed edges of this path are weighted 0, and a second path is computed. The two path are merged and edges going in reverse directions are deleted.

A variation exists for node-disjointedness that requires a simple graph transformation. Each node of the graph is replaced by two nodes, each carrying either all the incoming edges of the node or the outgoing edges with their respective weights. Those two nodes are linked by an edge, from the "in" node to the "out" node, with no weight.

2) LARAC and CBF: To account for the delay constraint while minimizing the cost, shortest path constrained algorithms are needed. Two algorithms are selected for this work: LARAC [20] based on Dijkstra algorithm [19] and 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.

Even though these algorithms are less optimal than Dijkstra's algorithm in terms of final cost, they provide a shortest path following a delay constraint.

B. Proposed algorithm

The proposed algorithm is based on Suurballe-Tarjan algorithm. First, to account for a delay constraint, the shortest path algorithm used in the original algorithm is replaced by a delay constrained shortest path algorithm.

1) Delay constraint: Dijsktra algorithm naturally builds a shortest path tree (SPT) as needed in Suurballe-Tarjan. A SPT is the tree containing the shortest paths from the source to every node. An adaptation is required to use either LARAC and CBF.

A first approach is to iteratively apply LARAC or CBF for each node in the graph, and to add the computed path to the SPT. But this does not guaranty that a single route exists in the SPT for each node, e.g. this does not form a tree as there can be multiple predecessor to a node. This is actually not an issue, as only finding the minimum distance to the source matters.

The main disadvantage of this procedure is that it requires running an algorithm for each node, which may imply a high number of algorithm instances.

A second approach is exploiting CBF. As it is a bread-first search, it visits all reachable nodes and finds the shortest path under delay constraint. This way, it is easy to find the shortest delay constrained distance to source for each node, in only one instance.

As stated in [21] and [20], the time complexity of LARAC is slightly superior to the complexity of CBF, even though CBF running time is less stable depending on graphs. For the experimentation in Section V, LARAC is used to ensure algorithm stability. Its higher complexity is not an issue given that the application requires offline computation.

2) Maximal Independence: The second alteration of the Suurballe-Tarjan algorithm is to permit edges to be shared between path if needed. To relax the strict independence requirement, the step 3 of Algorithm 1 where edges of the first computed path are pruned is removed. Instead, those edges are weighted using the total weight of the original graph's edges. This way, they will only be used in last resort if no other path is available. This is easy to prove, as any simple path (i.e. without cycle) is lighter than the total weight of the graph's edges. This alteration provides maximally independent paths.

3) Proposed Algorithm: The proposed algorithm is called SHERPA (SHaring-Edges Restrained PAths), and is described in Algorithm 2 and illustrated in a simple case in Fig. 3.

Algorithm 2 SHERPA

Input: A graph $G = (V, E)$, a source s, a destination t and a delay constraint Δ.

Output: Two maximally edge-disjoint delay constrained shortest paths from s to t.

1: Using a delay constrained shortest path algorithm, paths from s to every node $v \in V$ are computed. Their cost is noted $d_0(v)$. By definition this contains P_1 the delay constrained shortest path from s to t. If no path P_1 is found, then there is no solution.

2: Graph edges' weights are replaced as follow: for each edge $(a, b) \in E$ the new weight is: $w_1(a, b) = w_0(a, b) - d_0(b) + d_0(a)$ with w_0 the weight of an edge in the original graph, and d_0 the delay constrained shortest distance from source.

3: For each edge $(a, b) \in E \cap P_1$ the new weight is w_{max}, the total weight of the graph, and the opposite edge (b, a) is weighted 0. The produced graph is called Residual Graph.

4: Using a delay constrained shortest path algorithm, a path P_2 from s to t is found in this new graph.

5: Edges that exists in opposite directions in P_1 and P_2 are pruned.

6: $\{P_1, P_2\}$ contains the two paths.

Similarly to the original Suurballe-Tarjan algorithm, node-disjoint paths can be found by the proposed algorithm. The variation requiring replacing each node in the graph by an "in" node and an "out" node can be identically applied. Edges delays are transposed like edges weight, and the unidirectional link between the two nodes has a null delay.

In step 2 of Algorithm 2, edges of the paths computed in step 1 may have a zero weight. A loss in optimality could then appear in step 4: multiple paths fitting the delay constraint may exist, and the selected one may contain those 0-length edges while not being shortest path.

This algorithm is intended for graphs that are not fully meshed. As not many paths may exist, this loss in optimality will often not be observed. For higher connectivity graphs, Section III reviews some interesting solutions.

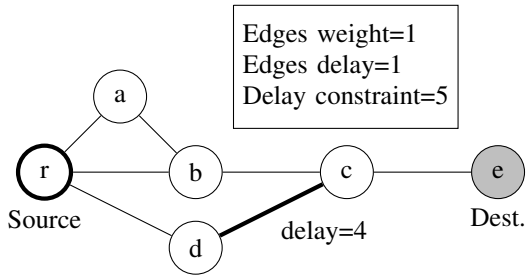
V. Algorithm Evaluation

A. Experimentation Scheme

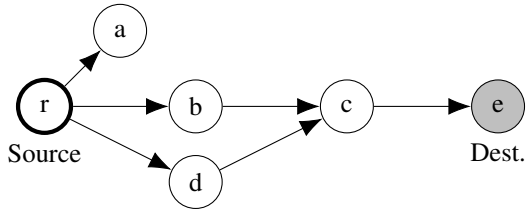
SHERPA algorithm is here evaluated in terms of performance and scalability.

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 $n = 800$.

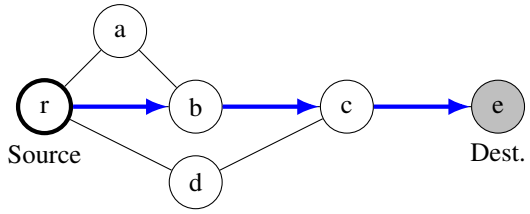
Random links are added between connected component to ensure the connectivity of the whole graph. It could be argued that this is not the best model for networks. Statistical significance is discussed later.



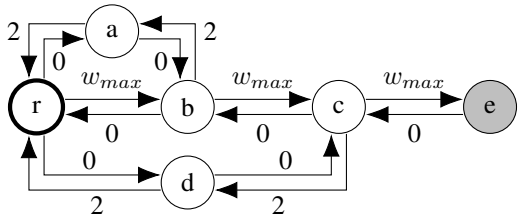
(a) Base graph



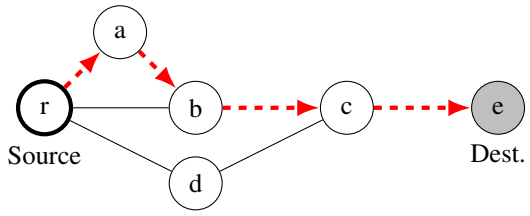
(b) Delay Constrained Shortest Path Tree - Alg. 2 step 1



(c) P1 - Alg. 2 step 1



(d) Residual graph - Alg. 2 step 2 and 3



(e) P2 - Alg. 2 step 4

Fig. 3. SHERPA illustration

The probability p of edge creation is chosen according to the expected number of edges. For small and medium graphs, different degrees will be used. Sparse and dense graphs will be generated with $p = 0.1$ and 0.6 for $n = 20$, and $p = 0.01$ and 0.3 for $n = 100$. For large graph, around 1600 edges are wanted, which matches $p = 0.002$.

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

One node is randomly selected as source and another node as destination. Such random selection is not representative of a hierarchical network. On a Internet network, source and destination 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 [23], 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 destination is selected outside, the problem is closer to the studied case. One could go even further and use classification and partition methods to define a graph hierarchy, but problem generation is not at 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 paths are strictly identical and match the shortest-delay path from source to destination.

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

- Computation time;
- Path sharing: the proportion of shared edges of the smaller path, as it will be the higher proportion;
- Completion: did the algorithm provide a valid solution where destination is reached within delay constraint.

Some of these variables are quantitative while some are binomials. In any case, the mean of the variables will be studied.

Number of test instances is chosen to be representative of the total population, but small enough to be tested in reasonable time. This study is about connected graphs. The number of possible linear graphs is $n!$ for a given number of nodes n. The number of possible simple graphs (not necessarily connected) for a given number of nodes n is $2^{\binom{n}{2}}$, as there are $\binom{n}{2}$ possible edges, and the number of possible subsets of a N-sized set is 2^N. It comes that the size of the population $|P|$ follows $n! \leq |P| \leq 2^{\binom{n}{2}}$. This illustrates the high size of the studied population. For example, for small graphs with $n = 7$ there are $5040 \leq |P| \leq 2^{21}$ connected graphs.

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

B. Results

Table. I illustrates mean performance confidence interval at 99% probability for the tested characteristics over different instances sizes. Table. II shows comparative performance to a brute force solution, over small size instances.

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

Instance Size (n/p)	20 0.1	20 0.6	100 0.01	100 0.3	800 0.002
Number of edges	45	228	199	2975	1687
±	0.5	1	0.1	5	2
Computation Time (s)	0.002	0.035	0.032	1.531	0.919
±	10^{-4}	10^{-3}	10^{-3}	10^{-2}	10^{-2}
Path sharing (%)	76	24	96	6	39
±	2	3	1	2	2
Completion (%)	100	100	100	100	100
±	0	0	0	0	0

TABLE II
SHERPA VS BRUTE FORCE (INTERVALS AT 99%)
FOR N=20 AND P=0.1 INSTANCES

	Time (s)	Completion (%)	Optimum found (%)
Brute force	0.003	100	100
±	7.10^{-4}	0	0
SHERPA	0.002	100	96
±	10^{-4}	0	2

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

$$\overline{x} - \frac{L}{2} \leq \overline{X} \leq \overline{x} + \frac{L}{2} \tag{1}$$

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

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's variable.

The completion is studied in proportion P of the population, which confidence intervals are defined by:

$$\overline{p} - \frac{L}{2} \leq \overline{P} \leq \overline{p} + \frac{L}{2} \tag{3}$$

$$\frac{L}{2} = u_{1-\frac{\alpha}{2}} * \sqrt{\frac{\overline{p} * (1-\overline{p})}{n}} \tag{4}$$

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

C. Analysis

From experimental results, it appears that the algorithm does always find a valid solution when one exists. Depending on the graph connectivity, the paths are more or less independent as expected.

Data also shows the scalability of the algorithm: as computation times grows with the number of edges in the graph, but stays within reasonable values, even for an online computation scheme.

Comparative data shows that the algorithm often finds an optimal solution. Further validation is required on higher instances, but the brute force solution is not scalable.

D. Weaknesses

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 [24], random graphs are not an ideal way to evaluate algorithm as they may not be representative.

VI. Architecture Elements

In this paper, the Path Computation Element of the architecture was presented. For the architecture to be complete, others implementation issues are yet to be addressed in future works, see Fig. 2.

In particular, for the network architecture to be fully robust, a path resiliency scheme must be implemented. If one path is still active, it must not be altered to preserve continuity of service. As a consequence, it is not advised to restart Algorithm 2. The failed path can be either repaired through segment rerouting, or fully recomputed through path rerouting.Either way, a new segment or path should be computed over a reweighted graph. Active path edges are weighted using the total weight of the graph, and a delay contrained routing algorithm is applied, to find a new segment or path maximally independent from the active path.

Management plane elements can be implemented in various ways, and subjects such as retrieving delay and bandwidth values or detecting link failures are covered in the literature. Future works shall present a complete architecture where all elements are consistently chosen.

VII. Conclusion

In this paper we presented a new architecture for Seamless distribution, which is based on redundant paths to ensure that a real-time stream is carried without interruption to destination. We proposed an algorithm called SHERPA to compute maximally edge or node independent delay constrained paths and evaluated its performance. We showed that, though this algorithm does not guaranty an optimal solution, it meets the requirement of the multiple constraints within reasonable time.

The proposed architecture guarantees continuity of service in case of failure, and thus is more efficient than traditional rerouting schemes in terms of QoE. Architecture resiliency also accounts for multiple network failures.

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

[6] C. Hopps, "Analysis of an Equal-Cost Multi-Path Algorithm," IETF, Tech. Rep. RFC2992, Nov. 2000.

[7] A. Ford, C. Raiciu, M. Handley, and O. Bonaventure, "TCP Extensions for Multipath Operation with Multiple Addresses," IETF, Tech. Rep. RFC6824, Jan. 2013.

[8] J. P. Rohrer, A. Jabbar, and J. P. Sterbenz, "Path diversification: A multipath resilience mechanism." IEEE, Oct. 2009, pp. 343–351.

[9] J. Tapolcai, G. Retvari, P. Babarczi, E. R. Berczi-Kovacs, P. Kristof, and G. Enyedi, "Scalable and Efficient Multipath Routing: Complexity and Algorithms." IEEE, Nov. 2015, pp. 376–385.

[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] 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.

[12] F. Iqbal and F. A. Kuipers, "Disjoint paths in networks," *Wiley Encyclopedia of Electrical and Electronics Engineering*, 2015.

[13] N. Taft-Plotkin, B. Bellur, and R. Ogier, "Quality-of-service routing using maximally disjoint paths," in *Quality of Service, 1999. IWQoS'99. 1999 Seventh International Workshop on*. IEEE, 1999, pp. 119–128.

[14] A. Orda and A. Sprintson, "Efficient algorithms for computing disjoint QoS paths," vol. 1. IEEE, 2004, pp. 727–738.

[15] S.-J. Lee and M. Gerla, "Split multipath routing with maximally disjoint paths in ad hoc networks," vol. 10. IEEE, 2001, pp. 3201–3205.

[16] R. Ogier, V. Rutenburg, and N. Shacham, "Distributed algorithms for computing shortest pairs of disjoint paths," *IEEE Transactions on Information Theory*, vol. 39, no. 2, pp. 443–455, Mar. 1993.

[17] S.-r. Jung, J.-h. Lee, and B.-h. Roh, "An Optimized Node-Disjoint Multipath Routing Protocol for Multimedia Data Transmission over Wireless Sensor Networks." IEEE, Dec. 2008, pp. 958–963.

[18] H. Zafar, D. Harle, I. Andonovic, and M. Ashraf, "Partial-Disjoint Multipath Routing for Wireless Ad-hoc Networks." IEEE, Oct. 2007, pp. 258–259.

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

[20] 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.

[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] 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.

[24] 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.