Pulse+: DetNet Routing Under Delay-diff Constraint

Shizhen Zhao, Ximeng Liu, Tianyu Zhu, Xinbing Wang

***Abstract*—Deterministic Networking (DetNet) is a rising technology that offers deterministic delay & jitter and extremely low packet loss in large IP networks. To achieve determinism under failure scenarios, DetNet requires finding at least two paths with close end-to-end delay, i.e., a *delay-diff* constraint, for mission-critical flows. However, how to find two routing paths subject to the *delay-diff* constraint remains open. We study the DetNet routing problem in two scenarios. First, given a primary path, we propose Pulse+, which finds a secondary path whose end-to-end delay is within a range determined by the end-to-end delay of the primary path and the delay-diff requirement. Second, we propose CoSE-Pulse+, which integrates Pulse+ with a divide-and-conquer approach to find a pair of paths that meet DetNet's delay-diff constraint. Both Pulse+ and CoSE-Pulse+ guarantee solution optimality. Notably, although Pulse+ and CoSE-Pulse+ do not have a polynomial worst-case time complexity, their empirical solver running time is better than that of other algorithms. We evaluate Pulse+ and CoSE-Pulse+ against the K-Shortest-Path and Lagrangian-dual based algorithms using synthetic test cases generated over networks with up to 10000 nodes. Both Pulse+ and CoSE-Pulse+ can solve more test cases than other algorithms under a predefined time limit. Compared to the second best algorithm, Pulse+ achieves an average-time speedup of $5\times$ and CoSE-Pulse+ achieves an average-time speedup of $22\times$. Our code and test cases are available at [1].**

***Index Terms*—DetNet, Routing, Delay-diff, Bounded Jitter.**

I. Introduction

Modern mission-critical real-time network applications, e.g., telesurgery [2], PLC remote control [3], etc., require bounded end-to-end delay & jitter and extremely low packet loss rate even under extreme scenarios with network failures. However, existing internet is designed on a best-effort basis, and thus may not be able to support such applications with stringent Quality-of-Service (QoS) requirements. To deal with the above challenges, DetNet Architecture [4] was proposed to offer bounded delay and bounded delay jitter guarantee. DetNet aims to achieve bounded delay through explicit routes, and guards against network failures using service protection. However, it remains an open problem to find routing paths that meet DetNet's stringent delay requirements.

We briefly describe the routing requirements of DetNet as follows. Shared risk link group (Srlg) is a widely adopted concept to guard against network failures. An Srlg contains a set of links that share a common physical resource (cable, conduit, node, etc.). An Srlg is typically considered as an independent failure domain, and a failure of an Srlg will cause all links in this Srlg to fail simultaneously. To achieve service protection in DetNet, we need to find at least two paths that do not share any Srlg. We focus on finding two paths in this paper. With two paths, DetNet packets are replicated at the source and then transmitted along both paths and finally de-duplicated at the destination. To ensure deterministic delay under path failures, the end-to-end delay of both paths cannot differ too much. This introduces a *delay-diff* constraint to DetNet's routing problem. Unfortunately, existing routing algorithms either do not support the delay-diff constraint or incur high computational complexity.

We solve DetNet's routing problem in two steps. First, given a delay diff δ and a primary path with end-to-end delay d, we solve the Delay-Range Constrained Routing (DRCR) problem to find an Srlg-disjoint secondary path whose end-to-end delay is within $[d-\delta, d+\delta]$. Second, given a delay upper bound U and a delay diff δ, we solve the Srlg-disjoint DRCR problem to find two Srlg-disjoint paths at the same time such that both paths' end-to-end delays are no larger than U and their delay diff is no larger than δ. Notably, both of the DRCR and the Srlg-disjoint DRCR problems are NP-Complete.

The main challenges of the DRCR and the Srlg-disjoint DRCR problems come from the delay lower bound introduced by DetNet's delay-diff requirement. If there were no delay lower bound, the DRCR problem degenerates to the classical Delay Constrained Routing (DCR) problem. Although the DCR problem is NP-Complete [5], many algorithms have been proposed to solve the DCR problem with efficacy and these algorithms can be generally grouped into four categories: 1) the K-Shortest-Path (KSP) approaches; 2) the Lagrangian-dual approaches [5]–[7], 3) the dynamic programming approaches [6], [8]–[10] and 4) the Pulse approaches [11]–[13]. We tried to extend these approaches to handle the delay lower bound. Unfortunately, both the KSP and the Lagrangian-dual approaches may have to explore a large number of paths before finding a valid path, and thus can be slow in practice; the dynamic programming approaches cannot avoid routing loops when a delay lower bound exists. The pulse approach is promising, but if we directly apply it to the DRCR problem, the optimal solution may be incorrectly skipped. The Srlg-disjoint DRCR problem is even more difficult than the DRCR problem. In addition to the challenges faced by the DRCR problem, the Srlg-disjoint DRCR problem may also encounter a "trap" problem, i.e., many paths found do not have an Srlg-disjoint path. Although researchers have proposed "conflict set" to solve the trap problem [14], [15], existing conflict-set

This work was supported by the NSF China under Grant 62272292. (*Corresponding author: Shizhen Zhao.)*

Shizhen Zhao, Ximeng Liu, Tianyu Zhu, Xinbing Wang are with Shanghai Jiao Tong University, Shanghai 200240, China (e-mail: shizhenzhao@sjtu.edu.cn, liuximeng@sjtu.edu.cn, zhutianyu5@163.com, xwang8@sjtu.edu.cn).

solvers cannot handle delay constraints.

We propose Pulse+ and CoSE-Pulse+, to solve the DRCR problem and the Srlg-disjoint DRCR problem with optimality guarantee. The detailed contributions are as follows:

(1) For the DRCR problem, we propose Pulse+, the first Pulse-like algorithm that handles delay lower-bound constraints. Similar to Pulse, Pulse+ is a deep-first-search based path-finding algorithm, with pruning strategies to improve search efficiency. The novelties of Pulse+ are two folds: 1) we show that the "dominance check" pruning strategy adopted by the original Pulse is incompatible with the delay lower-bound constraint, and then prove that after disabling this pruning strategy, Pulse+ guarantees solution optimality; 2) we mutate the deep-first-search order to "Large-Delay-First" and show that the Large-Delay-First sorting strategy improves the searching efficiency of Pulse+.

(2) For the Srlg-disjoint DRCR problem, we propose CoSE-Pulse+, the first algorithm that finds two paths with similar end-to-end delay. The key challenge of solving the Srlg-disjoint DRCR problem is that many paths do not have an Srlg-disjoint path with similar end-to-end delay and getting "trapped" by these infeasible paths would dramatically increase the solver running time. We solve the above challenges from two aspects: 1) we adopt "conflict set" to overcome the "trap" problem and propose the first conflict-set-finding algorithm that can handle delay constraints, i.e., Conflict-Pulse+; 2) we develop a divide-and-conquer approach to ensure that all possible paths are explored and prove that the resulting CoSE-Pulse+ algorithm guarantees solution optimality.

(3) We generate synthetic test cases based on real Internet topologies and randomly-generated topologies with up to 10000 nodes, and evaluate Pulse+ and CoSE-Pulse+ against Delay-KSP, Cost-KSP and Lagrangian-KSP algorithms. Both Pulse+ and CoSE-Pulse+ can solve more test cases than other algorithms under a predefined time limit, and the completion-rate improvement becomes more prominent as the network size increases. Compared to the second best algorithm, Pulse+ achieves an average-time speedup of $5\times$ and CoSE-Pulse+ achieves an average-time speedup of $22\times$.

II. Background

In order to support time-sensitive applications in large IP networks, DetNet was proposed to offer a strict guarantee on end-to-end delay, delay jitter and packet loss [4]. Experimental implementation has been conducted in two of China's real networks, including CENI and YZNET [3]. In this section, we give an overview of DetNet's design and pinpoint its challenges.

A. Bounded Delay & Jitter with Explicit Routes

To avoid temporary interruptions caused by the convergence of routing and bridging protocols, the RFC of DetNet suggests establishing explicit routes for DetNet flows [4]. Explicit routes can be established in various ways, e.g., with RSVP-TE [16], with Segment Routing (SR) [17], via an SDN approach [18], etc. In order to meet DetNet's QoS requirements, for each explicit route, 1) the DetNet flows along this route must be rate limited; 2) each node needs a finite buffer to avoid packet loss, while guaranteeing finite latency at each hop; 3) the end-to-end latency must be within a predefined range. Rate limiting can be implemented using traffic policing. There are also many buffer-management approaches, e.g., CQF [19], CSQF [20], etc., that offer deterministic latency at each hop. However, how to find a path that meets a delay range requirement remains open.

B. Low Packet Loss Upon Failures with Service Protection

Network failures are common and packet loss could severely hurt the determinism of packet deliveries. DetNet proposes Service Protection to mitigate or eliminate packet loss due to network failures [4]. The simplest approach for Service Protection is 1+1 path protection. In 1+1 protection, two paths are used to route DetNet flow packets. At the source node, there is a Packet Replication Function (PRF) that duplicates the mission-critical packets onto two egress ports that forward the packets to both paths. At the destination node, the received packets are de-duplicated using a Packet Elimination Function (PEF).

In order to guarantee reliable recovery in case of packet loss, an efficient approach is to ensure that the delay diff between the two paths is smaller than the smallest time gap between adjacent packets (See Section 2.2 in [21] for the detailed explanation). This approach could guarantee low delay diff and in-order packet delivery even if one packet in the fast path is lost. However, the main challenge is on the algorithm side: finding two paths with similar end-to-end lantency.

Another approach to guarantee such packet-delivery determinism is adding a Packet Re-ordering Function (PROF) at the receiver side. The PROF would require additional buffering to store the received packets and additional computation to de-duplicate & re-order the recieved packets. However, this approach will not only increase the implementation complexity, but also hurt the end-to-end latency. In addition, having an increased end-to-end latency can be detrimental to some mission-critical applications, such as remote surgery.

III. Mathematical Model

We model a network using a directed graph $G = (V, E)$, where V is the set of nodes and E is the set of directed links. Each directed link $e \in E$ is associated with a delay $d(e)$ and a cost $c(e)$. We use From(e) and To(e) to denote the two ends of the link e. Given a source node $s \in V$ and a destination node $t \in V, s \neq t$, a link sequence $P = [e_1, e_2, ..., e_h]$ is called a path from s to t if and only if $\text{From}(e_1) = s, \text{To}(e_1) = \text{From}(e_2), ..., \text{To}(e_{h-1}) = \text{From}(e_h), \text{To}(e_h) = v$. A path P is called elementary if no vertex is repeated in the path. We are interested in finding elementary paths to avoid routing loops. (The IP based forwarding will fail if a path contains a loop.) The delay and cost of a path P are denoted by $d(P) = \sum_{e \in P} d(e)$ and $c(P) = \sum_{e \in P} c(e)$, respectively.

We use Shared risk link group (Srlg) to model network failures. Let R be the set of Srlgs in the network $G = (V, E)$. Each Srlg $r \in R$ contains a set of links that share a common physical resource (cable, conduit, node, etc.). Thus, a failure

of r will cause all links in this Srlg fail simultaneously. Each link $e \in E$ may belong to multiple Srlgs. We use $\Omega(e) \subseteq R$ to denote the set of Srlgs that contain the link e. Then, for each path P, $\Omega(P) = \cup_{e \in P} \Omega(e)$ represents all the Srlgs that contain at least one link in P. To guard against network failures in DetNet, we study two problems as follows.

Delay-Range Constrained Routing (DRCR) Problem: Given two distinct nodes $s, t \in V$ and a delay range $[L, U]$, find a min-cost path subject to the delay range constraint:

$$\begin{aligned} \min_{P} \quad & c(P) = \sum_{e \in P} c(e), \\ \textbf{s.t.} \quad & P \text{ is an elementary path from } s \text{ to } t, \\ & L \leq d(P) = \sum_{e \in P} d(e) \leq U. \end{aligned} \tag{1}$$

This formulation applies to the scenarios where we have a primary path P_a and want to find an Srlg-disjoint path P_b with end-to-end delay satisfying $d(P_a) - \delta \leq d(P_b) \leq d(P_a) + \delta$, where δ is the maximum allowable delay diff. Such a formulation was also adopted in [22].

Srlg-disjoint DRCR Problem: Given two distinct nodes $s, t \in V$, a delay upper bound U and a delay diff δ, find a pair of Srlg-disjoint primary and secondary paths such that the primary path has the minimum cost and the delay diff of the two paths does not exceed δ, i.e.,

$$\begin{aligned} \min_{P_a, P_b} \quad & c(P_a) = \sum_{e \in P_a} c(e), \\ \textbf{s.t.} \quad & P_a, P_b \text{ are two elementary paths from } s \text{ to } t, \\ & d(P_a) \leq U, d(P_a) - \delta \leq d(P_b) \leq \min\{U, d(P_a) + \delta\}, \\ & \Omega(P_a) \cap \Omega(P_b) = \emptyset. \end{aligned} \tag{2}$$

This formulation is used if we want to find the primary path and the secondary path at the same time. Note that the delay-diff constraints can be also formulated using integer linear programming (ILP) [21]. However, the ILP-based formulation incurs significant computational complexity and could only handle very small network topologies.

Remark on the objective function of (2): In DetNet, there are still many best effort packets, which can tolerate delay jitters and packet loss. Since 1+1 protection is expensive as it doubles the traffic in the network and introduces extra processing cost, in practice 1+1 protection is only enabled for mission-critical packets. As a result, the primary path is used all the time, while the secondary path is only used for mission-critical packets. Therefore, we decide to optimize the cost of the primary path, rather than optimizing the sum cost of both paths.

Remark on the generality of finding Srlg-disjoint paths: In some circumstances, one may care about finding link-disjoint or node-disjoint paths instead of Srlg-disjoint paths. We argue that finding link-disjoint or node-disjoint paths is a special case of finding Srlg-disjoint paths. Specifically, if every Srlg contains only one link, then finding Srlg-disjoint paths degenerates to finding link-disjoint paths; if every node in G, except the source node s and the destination node t, corresponds to an Srlg, and each Srlg $u \in V, u \neq s, t$ contains all the ingress and egress links of u, then finding Srlg-disjoint paths degenerates to finding node-disjoint paths.

Remark on Algorithmic Complexity: Both the DRCR and the Srlg-disjoint DRCR problems are NP-Complete. By setting $L = 0$, the DRCR problem degenerates to the Delay Constrained Routing (DCR) problem, which was proven to be NP-Complete in [5]. Thus, the DRCR problem is also NP-Complete. In addition, given a DCR problem instance, if we create a side link e' from s to t with $d(e') \leq U$ and a large $c(e') > \sum_{e \in E} c(e)$, let this link e' form a separate Srlg, and set $\delta = U$, then this DCR problem instance will reduce to an Srlg-disjoint DRCR problem instance. Therefore, the Srlg-disjoint DRCR problem is also NP-Complete.

Since the DRCR and the Srlg-disjoint DRCR problems are NP-Complete, it is impossible to design polynomial algorithms unless $P = NP$. The objective of this paper is thus to design computational efficient algorithms for these two problems, and demonstrate that they are empirically efficient to support large DetNets with thousands of nodes and links.

IV. Algorithm Design Principles

Due to the presence of the delay-diff constraint, there exists no algorithm that meets the routing requirement of DetNet. (See Section X for a more detailed literature review). Fortunaltely, the exisiting literature could still offer some insights and helps identify a few promising directions.

A. DRCR Problem

To the best of our knowledge, there exists only one paper [22] that directly studied the DRCR problem. However, the algorithm proposed in [22] is merely a heuristic solution with no optimality guarantee. Nevertheless, if there were no lower-bound constraint on the end-to-end delay, the DRCR problem degenerates to the classical DCR problem[1]. Existing solutions to the DCR problem can be grouped into 4 categories. We examine these solutions one by one to identify promising algorithm-design directions for the DRCR problem.

1) K-Shortest-Path (KSP) approaches [23]: The key idea is to examine all the paths with cost ordered from low to high, and the first path that meets the delay constraint gives the optimal solution. This approach is efficient if the KSP algorithm can terminate with a small k value. However, when the delay bound is tight (i.e., $U - L$ is small), finding a path that meets the delay constraint may take a large number of iterations, which makes the KSP algorithm prohibitively expensive. (See Appendix B-A1 in our technical report [24].)

2) Lagrangian-dual approaches [5]–[7]: The key idea is to run the KSP algorithm based on a combined weight function $w_\lambda(e) = c(e) + \lambda d(e)$, where $c(e)$ and $d(e)$ are the delay and the cost of the link e. By properly choosing λ, the Lagrangian-dual approach could dramatically reduce the number of iterations required to find the optimal path. The Lagrangian-dual approach is effective in dealing with the delay upper bound. However, as we apply this approach to handle delay lower bound in the DRCR problem, we may

[1]Also known as the Constrained Shortest Path (CSP) problem in literature.

need to use a *negative* value for λ in certain cases. When λ is negative, the weight function $w_\lambda(e)$ may become negative and the KSP algorithm no longer applies. (See Appendix B-A2 in our technical report [24].)

3) Dynamic programming approaches [6], [8]–[10]: For a given destination node t, let $\psi(u,T)$ be the minimum cost of all the paths from u to t whose end-to-end delay is no larger than T. Then, $\psi(u,T) = \min_{e=(u,v)}\{\psi(v, T-d(e)) + c(e)\}$. Then, starting from $\psi(t,0)=0$, we can compute each $\psi(u,T)$ and the corresponding min-cost path from u to t using dynamic programming. When there is no delay lower bound, all the min-cost paths found must be elementary, i.e., every node is visited at most once. Otherwise, by removing a cycle from the resulting path, a lower cost path can be found. Unfortunately, when a delay lower-bound exists, such approaches cannot guarantee the optimal path to be elementary. Hence, we decide not to pursue this direction.

4) Pulse approaches [11]–[13]: These approaches use depth first search or KSP search to find a solution to the DCR problem, and adopts several pruning strategies to accelerate the search. Such approaches are the most efficient in solving the DCR problems among all the approaches. However, when a delay lower bound exists, some pruning strategies in Pulse may fail, which reduces the pruning efficiency. Nevertheless, Pulse offers a promising framework for solving DRCR problems, and the challenge is to develop new optimization techniques to improve the pruning efficiency.

B. Srlg-disjoint DRCR Problem

To the best of our knowledge, finding Srlg-disjoint path pairs with delay requirements has never been studied before. Nevertheless, if we remove the delay constraint, the degenerated problem did receive much attention in the past decades. We examine different solutions to find the promising algorithm-design directions and identify the corresponding challenges.

1) Primary-Path-First approaches [25], [26]: These approaches first compute a primary path without considering the need to find a secondary path, and then try to find an Srlg-disjoint path by removing those links affected by the primary path. If there exists no Srlg-disjoint path, these approaches may try a different primary path or stop based on certain criterion. In this paper, we tried two such approaches, one uses the KSP algorithm to find primary paths (see Appendix B-A1 in our technical report [24]) and another one uses the Lagrangian-dual algorithm to find primary paths. Despite of the simplicity of these approaches, they may suffer from the so-called *trap* problem [14], i.e., many primary paths do not have an Srlg-disjoint path due to some special network structure (see an example in Section VI-A) and blindly trying different primary paths can be highly inefficient.

2) Conflict-Set based approaches [14], [15], [27]: The concept of "conflict set" was proposed in [14] to solve the *trap* problem. Given a primary path P_a, if there exists no Srlg-disjoint path, one can always find a small Srlg set $T \subseteq \Omega(P_a)$, such that every primary path whose Srlg set contains T does not have an Srlg-disjoint path. This set T is called a "conflict set". If we could avoid finding primary paths whose Srlg set contains a conflict set, then it would be much easier to find an Srlg-disjoint path. Here, the key is to compute the conflict set. Unfortunately, existing solutions [14], [15], [27] only focused on the unconstrained routing scenarios without delay constraints, and thus cannot be used to find conflict sets for the Srlg-disjoint DRCR problem.

V. DRCR Algorithm

We propose Pulse+ to solve the DRCR problem in this section. Let $P_{s\to t}^{\text{min\_delay}}$ be the elementary path from s to t with the minimum delay and let $P_{s\to t}^{\text{min\_cost}}$ be the elementary path from s to t with the minimum cost. Clearly, the end-to-end delay of the first path is no larger than that of the second path, i.e., $d(P_{s\to t}^{\text{min\_delay}}) \le d(P_{s\to t}^{\text{min\_cost}})$.

All the DRCR problems can be grouped into the following six cases according to the relationship between the delay upper bound U, delay lower bound L, the min-delay path's delay $d(P_{s\to t}^{\text{min\_delay}})$ and the min-cost path's delay $d(P_{s\to t}^{\text{min\_cost}})$:

Case 1 (Infeasible): $L \le U < d(P_{s\to t}^{\text{min\_delay}}) \le d(P_{s\to t}^{\text{min\_cost}})$. It is impossible to find a path with delay smaller than the minimum delay $d(P_{s\to t}^{\text{min\_delay}})$.

Case 2 (Degenerated Case): $L \le d(P_{s\to t}^{\text{min\_delay}}) \le U < d(P_{s\to t}^{\text{min\_cost}})$. All paths can meet the delay lower bound. Thus, the delay lower bound can be ignored and this case can be solved by the original Pulse algorithm [11].

Case 3 (Trivial): $L \le d(P_{s\to t}^{\text{min\_delay}}) \le d(P_{s\to t}^{\text{min\_cost}}) \le U$. The min-cost path $P_{s\to t}^{\text{min\_cost}}$ is optimal.

Case 4 (Non-trivial): $d(P_{s\to t}^{\text{min\_delay}}) < L < U < d(P_{s\to t}^{\text{min\_cost}})$.

Case 5 (Trivial): $d(P_{s\to t}^{\text{min\_delay}}) < L < d(P_{s\to t}^{\text{min\_cost}}) < U$. The min-cost path $P_{s\to t}^{\text{min\_cost}}$ is optimal.

Case 6 (Non-trivial): $d(P_{s\to t}^{\text{min\_delay}}) < d(P_{s\to t}^{\text{min\_cost}}) < L < U$.

A. Review of the Pulse Algorithm

In Case 2, the DRCR problem degenerates to the Delay Constrained Routing (DCR) problem:

$$\min_{P} \quad c(P) = \sum_{e\in P} c(e) \ \textbf{s.t.} \ d(P) = \sum_{e\in P} d(e) \le U. \tag{3}$$

The DCR problem has been studied in the literature. Among all the algorithms proposed, Pulse [11] performs the best. The pulse algorithm (see Algorithm 1) adopts a branch-and-bound method to find the optimal solution of (3). It defines global variables tmp_min_cost and $P_{s\to t}^{\text{opt}}$ to track the best path found, and then performs depth first search using a stack. In the depth first search, lines 5-11 check the path found and update the best path found so far; lines 12-14 adopt three pruning strategies to cut some search branches; lines 15-17 iterate through all the egress links of the node u and add the new branches to the stack. The optimal path $P_{s\to t}^{\text{opt}}$ must be an elementary path. Otherwise, $P_{s\to t}^{\text{opt}}$ will contain at least one cycle, and by removing this cycle from $P_{s\to t}^{\text{opt}}$, we could obtain another path with lower end-to-end cost.

We delve into the details of the three pruning strategies below. The first strategy "$d(P_{s\to u}) + d(P_{u\to t}^{\text{min\_delay}}) > U$" prunes branches by feasibility. It indicates that it is impossible to

Algorithm 1: Pulse Algorithm [11]

Data: A network $G(V, E)$, a source node s, a destination node t, and a delay upper bound U.
Result: The optimal path $P^{\text{opt}}_{s\to t}$ from s to t.

Use tmp_min and $P^{\text{opt}}_{s\to t}$ to track the best path found. Initialize tmp_min $= +\infty$.
Use a stack S to store all the branches to be explored. Initialize $S = \{\text{empty\_path}\}$.
// Use deep first search to find $P^{\text{opt}}_{s\to t}$.
while *S is not empty* **do**
  Let path $P_{s\to u} = S$.pop(). Let u be the end node of $P_{s\to u}$. Set $u = s$ if $P_{s\to u}$ is empty.
  if $u == t$ **then**
    if $d(P_{s\to u}) \le U$ *and* $c(P_{s\to u}) <$ *tmp_min* **then**
      tmp_min $= c(P_{s\to u})$;
      $P^{\text{opt}}_{s\to t} = P_{s\to u}$;
    end
    continue;
  end
  // Cut branches when possible.
  if $d(P_{s\to u}) + d(P^{\textit{min\_delay}}_{u\to t}) > U$ *or* $c(P_{s\to u}) + c(P^{\textit{min\_cost}}_{u\to t}) \ge$ *tmp_min or* $CheckDominance(u, P_{s\to u}) == true$ **then**
    continue;
  end
  // Add new branches.
  for *every egress link e of the node u* **do**
    S.push($P_{s\to u} \cup \{e\}$);
  end
end
return $P^{\text{opt}}_{s\to t}$;

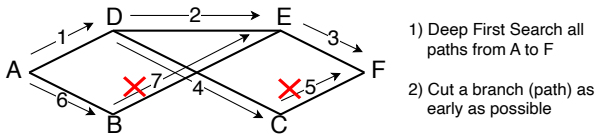


Figure 1. Illustration of Pulse-like Algorithms. The index on each link indicates the search order.

obtain a path with end-to-end delay no larger than U through this branch. The second strategy "$c(P_{s\to u}) + c(P^{\text{min\_cost}}_{u\to t}) \ge$ tmp_min" prunes branches by optimality. It indicates that it is impossible to obtain a path with lower cost through this branch. The third strategy "$CheckDominance(u, P_{s\to u}) == true$" prunes branches by dominance. Given two paths $P^1_{s\to u}$, $P^2_{s\to u}$ from s to u, $P^1_{s\to u}$ dominates $P^2_{s\to u}$ if and only if $d(P^1_{s\to u}) \le d(P^2_{s\to u})$ and $c(P^1_{s\to u}) \le c(P^2_{s\to u})$. Then, if we have searched the branch $P^1_{s\to u}$, searching the branch $P^2_{s\to u}$ cannot yield a better solution and thus can be skipped.

Illustration of Pulse-like Algorithms: The original Pulse algorithm can be easily generalized to meet different routing design objectives. Figure 1 illustrates the two key components in the design of Pulse-like algorithms. First, we need to determine a deep-first-search order for all the paths from the source to the destination. With exhaustive deep-first-search, the solution is guaranteed to be optimal. Second, we need to find an efficient approach to cut a branch (path) as early as possible to accelerate deep first search.

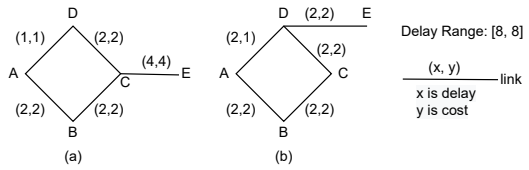


Figure 2. Dominance check is unsafe for DRCR.

B. Pulse+: Handling the Delay Range

We propose Pulse+, an enhanced Pulse algorithm, to compute the optimal solutions for the general DRCR problems. In this section, we detail the key difficulties encountered and the optimization techniques proposed for Pulse+. (We also studied the KSP-based approach and the Lagrangian-Dual based approach in this paper. Since these two approaches are less efficient than Pulse+, we put the detailed design in Appendix B-A of our technical report [24] for reference.)

1) Dominance Check is Unsafe: The efficiency of the Pulse-like algorithms heavily relies on the pruning strategies. The original Pulse algorithm adopts three pruning strategies, i.e., "$d(P_{s\to u}) + d(P^{\text{min\_delay}}_{u\to t}) > U$", "$c(P_{s\to u}) + c(P^{\text{min\_cost}}_{u\to t}) \ge$ tmp_min" and "$CheckDominance(u, P_{s\to u}) == true$". The first two pruning strategies are still valid, but the third one may prune a branch incorrectly for the DRCR problem and result in a sub-optimal solution.

We use two examples in Figure 2 to demonstrate the incorrectness of the Dominance check strategy. In the two examples, we need to find a min-cost path from A to E, such that the end-to-end delay is 8 (or the delay range is $[8, 8]$). Suppose that we have explored the branch $P_1 = A \to D \to C$, and we are to examine the path $P_2 = A \to B \to C$. In Figure 2(a), $d(P_1) = 3 < 4 = d(P_2), c(P_1) = 3 < 4 = c(P_2)$, and thus P_2 will be pruned by the dominance check. Clearly, after pruning P_2, we can no longer find a path from A to E that meets the delay range constraint. Note that the path $A \to B \to C \to E$ meets the end-to-end delay requirement.

The example in Figure 2(a) hints us to modify the dominance check as follows. Given two paths $P^1_{s\to u}, P^2_{s\to u}$ from s to u, $P^1_{s\to u}$ dominates $P^2_{s\to u}$ if and only if $d(P^1_{s\to u}) = d(P^2_{s\to u})$ and $c(P^1_{s\to u}) \le c(P^2_{s\to u})$. Unfortunately, this modified dominance check is still incorrect. Consider the example in Figure 2(b). Suppose that we are to examine $P_2 = A \to B \to C$ after exploring $P_1 = A \to D \to C$. Since $d(P_1) = 4 = d(P_2), c(P_1) = 3 < 4 = c(P_2)$, P_2 will be pruned by the dominance check, and then we can no longer find the optimal solution $A \to B \to C \to D \to E$. Apparently, this optimal solution is attained by concatenating P_2 and $C \to D \to E$. However, $C \to D \to E$ cannot be concatenated with P_1 because the node D has already been visited by P_1.

Admittedly, if the two paths $P^1_{s\to u}$ and $P^2_{s\to u}$ contain the same set of nodes and satisfy $d(P^1_{s\to u}) =$

Algorithm 2: Pulse+ Algorithm

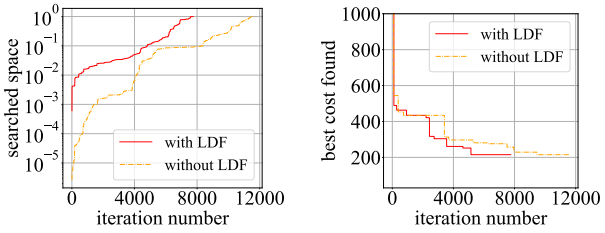
Data: A network $G(V, E)$, a source node s, a destination node t, and a delay range $[L, U]$.
Result: The optimal path $P^{\text{opt}}_{s\to t}$ from s to t.

1 Use tmp_min and $P^{\text{opt}}_{s\to t}$ to track the best path found. Initialize tmp_min $= +\infty$.
// Sort egress links to accelerate Pulse+.
2 For every node $v \in V$, sort all the egress links of v from lowest to highest based on the weight $w(e) = d(e) + d(P^{\text{min\_delay}}_{\text{To}(e)\to t})$.
// Use depth first search to find $P^{\text{opt}}_{s\to t}$.
3 Use a stack S to store all the branches to be explored. Initialize $S = \{\text{empty\_path}\}$.
4 **while** *S is not empty* **do**
5 Let path $P_{s\to u} = S$.pop(). Let u be the end node of $P_{s\to u}$. Set $u = s$ if $P_{s\to u}$ is empty.
6 **if** $u == t$ **then**
// Validate the path found.
7 **if** $\boxed{L \le d(P_{s\to u}) \le U}$ **then**
8 **if** $c(P_{s\to u}) <$ *tmp_min* **then**
9 tmp_min $= c(P_{s\to u})$;
10 $P^{\text{opt}}_{s\to t} = P_{s\to u}$;
11 **end**
12 **end**
13 **continue**;
14 **end**
// Cut branches when possible.
15 **if** $\boxed{P_{s\to u}\ \textit{should be pruned}}$ **then**
16 **continue**;
17 **end**
// Add new branches.
18 **for** *every egress link e of the node u* **do**
19 **if** *the node To(e) is not visited in $P_{s\to u}$* **then**
20 S.push($P_{s\to u} \cup \{e\}$);
21 **end**
22 **end**
23 **end**
24 return $P^{\text{opt}}_{s\to t}$;

$d(P^2_{s\to u}), c(P^1_{s\to u}) \le c(P^2_{s\to u})$, then $P^1_{s\to u}$ will dominate $P^2_{s\to u}$. However, this pruning strategy requires memorizing (delay, cost) pairs for all the visited node sets and the total number of different node sets grows exponentially with respect to the network size, making it scale poorly.

Based on the above considerations, we decide to remove the "dominance check" pruning strategy in the Pulse+ algorithm. Thus, the detailed pruning strategy of Pulse+ (see the box in line 15 of Algorithm 2) becomes

$$\boxed{\begin{array}{l} d(P_{s\to u}) + d(P^{\text{min\_delay}}_{u\to t}) > U \\ \text{or}\ \ c(P_{s\to u}) + c(P^{\text{min\_cost}}_{u\to t}) \ge \text{tmp\_min} \end{array}} \quad (4)$$



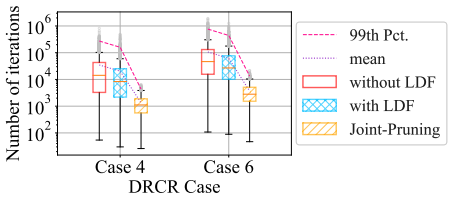
(a) Searched space size vs. iteration (b) Cost of best path vs. iteration

Figure 3. LDF accelerates Pulse+ search.

2) Visited Node Tracking is Necessary: Unlike the DCR problem, given a path $P_{s\to t}$ with duplicated nodes and $L \le d(P_{s\to t}) \le U$, we cannot remove cycles from $P_{s\to t}$ to obtain a lower-cost path, as the resulting path may violate the delay lower bound. As a result, if we do not enforce that each node can only be visited once, the resulting optimal path may not be an elementary path. Take Figure 2(b) for example. If we allow visiting a node more than once, the optimal solution would be $A \to D \to C \to D \to E$, which has an end-to-end cost of 7. In contrast, the optimal elementary path is $A \to B \to C \to D \to E$, which has an end-to-end cost of 8. According to the above analysis, we decide to explicitly track the visited nodes and make sure that no node is visited more than once (see line 19 in Algorithm 2).

3) Largest-Delay-First Searching Strategy: Having removed the "dominance check" pruning strategy, the pruning efficiency can be impaired dramatically. We thus propose the Largest-Delay-First (LDF) Searching strategy to improve the pruning efficiency for Pulse+. At the beginning, since tmp_min $= +\infty$, we can only rely on the feasibility pruning strategy "$d(P_{s\to u}) + d(P^{\text{min\_delay}}_{u\to t}) > U$" to cut branches. The LDF searching strategy explores egress links with higher end-to-end delay to the destination ($w(e) = d(e) + d(P^{\text{min\_delay}}_{\text{To}(e)\to t})$) first. The high-priority branches in the Pulse+ search either can be cut by the feasibility pruning strategy, or yield paths with end-to-end delay close to the delay upper bound U. As a result, the tmp_min value can be effectively reduced in the early stages of the Pulse+ DFS search, and then the optimality pruning strategy "$c(P_{s\to u}) + c(P^{\text{min\_cost}}_{u\to t}) \ge$ tmp_min" becomes more effective. Note that we use a stack to perform DFS, and a stack is last-in-first-out. Hence, to implement LDF, we need to sort all the egress links of a node $u \in V$ in an increasing order of the end-to-end delay from a link e to the destination node t (see line 2 in Algorithm 2).

We use a randomly selected test case to illustrate why LDF searching strategy could accelerate Pulse+ search. This test case is generated in a network with 4000 nodes and 99779 links. In order to quantify the progress of Pulse+ search, we introduce a new concept called *searching space size* (S^3) for every partial path $P_{s\to u}$ in the stack S (see line 3 in Algorithm 2). The first partial path in S is an empty path. An empty path means that Pulse+ needs to explore the whole searching space. Therefore, we set $S^3(\text{empty\_path}) = 1$. Every partial path $P_{s\to u}$ may generate a number of sub-paths in lines 18-21 of Algorithm 2. We set $S^3(P_{s\to u} \cup \{e\}) = S^3(P_{s\to u})/n$, where n is the number of sub-paths of $P_{s\to u}$. We say $P_{s\to u}$



	Case	without LDF	with LDF	Joint-Pruning
50th Pct.	Case 4	14199	8323	1606
	Case 6	46413	26719	6811
75th Pct.	Case 4	43528	25218	2610
	Case 6	131327	76254	12043
99th Pct.	Case 4	817173	489734	8392
	Case 6	1912911	1242840	57716

Figure 4. Strategies to reduce the number of iterations.

is explored if and only if all of its sub-paths are explored. In Figure 3(a), we plot the total searched space size of all the explored partial paths versus the number of iterations of the **while** loop (lines 4-23 in Algorithm 2). We can see that the searched space size increases much faster after enabling the LDF searching strategy. As a result, Pulse+ with LDF requires fewer number of iterations to find the optimal solution (see Figure 3(b)).

We generate DRCR test cases (see Section VII-A1), each of which belongs to either Case 4 or Case 6. For each test case, we record the number of iterations in Pulse+ search and summarize the average value and the percentile values in Figure 4. We can see that enabling LDF reduces the number of iterations by about 40%.

Another approach to accelerate Pules+: LDF is not the only approach to accelerate Pules+. In Appendix B-B of our technical report [24], we offer a joint-pruning approach, which could achieve even higher pruning and searching efficiency than LDF. However, the joint-pruning approach requires calculating a cost function beforehand, which incurs significant overhead. (For each test case, this overhead accounts for nearly 90% of the total computation time.) After weighing the pros and cons, we set LDF as the default search acceleration strategy for Pulse+.

4) Algorithmic Complexity and Optimality Guarantee: The algorithmic complexity of Pulse+ comes mainly from the **while** loop (lines 3-18 in Algorithm 2). In each while loop, the complexity of lines 4-14 is $O(1)$ and the complexity of lines 15-17 is $O(\deg_{\max})$, where $\deg_{\max}$ is the max node degree of the network. Let $K_{\text{pulse+}}$ be the total number of iterations in the while loop. Then the algorithmic complexity of Pulse+ is $O(K_{\text{pulse+}}\deg_{\max})$. In theory, $K_{\text{pulse+}}$ grows exponentially with respect to the network size. Nevertheless, thank to the high pruning efficiency, the values of $K_{\text{pulse+}}$ do not grow that fast in practice. We will numerically study $K_{\text{pulse+}}$ in Section VII-B.

In addition, Theorem 1 guarantees the optimality of Pulse+.

Theorem 1: (See Appendix A-C for the proof) For any DRCR problem instance, the solution generated by Pulse+ is optimal.

VI. SRLG-Disjoint DRCR Algorithm

We propose CoSE-Pulse+, to solve the Srlg-disjoint DRCR problem in this section. As discussed in Section IV-B, the key to CoSE-Pulse+ is the design of a conflict-set finding algorithm subject to delay constraints, which is described first below.

A. Conflict-Pulse+: Find a Conflict Set

1) Why do We Need Conflict Sets?:

Definition 1: (Conflict Set) Given a path P_a, its conflict set T is a subset of $\Omega(P_a)$ such that every path P whose Srlg set $\Omega(P)$ contains T cannot find an Srlg-disjoint path.

The concept of *conflict (Srlg) set* was proposed to solve the "trap" problem encountered in the link/Srlg-disjoint path finding problems, especially when the delay diff is small (which is common in DetNet). When trap happens, we get "trapped" in an infeasible solution space and cannot step out without tremendous searching.

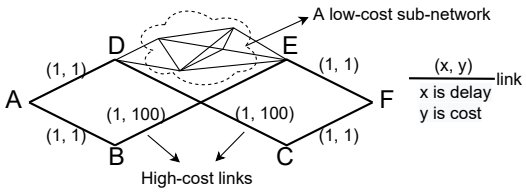


Figure 5. The Trap Problem and the Conflict Srlg Set.

Figure 5 shows an example of the trap problem. In this example, each Srlg only contains one link and thus we can use a link to represent an Srlg. The objective is to find two Srlg-disjoint paths from A to F such that the primary path attains the minimum cost. One natural idea is to find a sequence of primary paths with end-to-end cost sorted from low to high, and test if it is possible to find an Srlg-disjoint path. However, this approach can be extremely inefficient for the example in Figure 5. Note that the links CD and BE have very high cost, the low-cost paths from A to F would be of the form $A \to D \to E \to F$ ($D \to E$ actually consists of multiple links in the low-cost sub-network in Figure 5). However, none of the paths of the form $A \to D \to E \to F$ can find an Srlg-disjoint path. In this example, we are "trapped" in an infeasible solution space and have to do many iterations to step out.

In Figure 5, the Srlg set $\{AD, EF\}$ forms a conflict Srlg set. No path containing $\{AD, EF\}$ could find an Srlg-disjoint path. **Having found a number of Conflict (Srlg) Sets, if we could avoid finding an primary path P_a such that $\Omega(P_a)$ contains a conflict set, we could avoid the "trap" and accelerate the search of a feasible pair of primary and secondary paths.**

2) How to Find a Conflict Set for an Active Path P_a?: The problem of finding a conflict set has been studied in [14], [15], [27]. However, their approaches cannot handle delay constraints. Specifically, if a primary path P_a only has Srlg-disjoint path, but this path violates the delay constraint, then the existing conflict set finding algorithms in [14], [15], [27] will fail to find a conflict set, because these algorithms could incorrectly identify a Srlg-disjoint path for P_a.

According to Definition 1, given a conflict set T, if for every Srlg $r \in T$, we disable all the links in r, then we cannot find an Srlg-disjoint path for P_a. Based on this insight, we design Conflict-Pulse+. Conflict-Pulse+ performs a pulse-like search for a path that meets the delay constraints. Whenever a path $P_{s\to t}$ is found, we pick an Srlg $r \in \Omega(P_{s\to t}) \cap \Omega(P_a)$ and disable all the links in r. After exploring all the searching branches, all the chosen Srlgs form a conflict set. Due to space constraint, we put the detailed design of Conflict-Pulse+ in Appendix A-B.

B. CoSE-Pulse+: Solve Srlg-Disjoint DRCR

Based on the concept of the conflict (Srlg) set, we propose CoSE (**Co**nflict **S**rlg **E**xclusion)-Pulse+ to find Srlg-disjoint paths with delay constraints. CoSE-Pulse+ adopts a similar divide-and-conquer approach as CoSE [15]. The key difference is that CoSE uses the shortest path algorithms, e.g., Dijkstra [28], A^* [29], etc., to compute primary/secondary paths, while CoSE-Pulse+ uses variants of the Pulse+ algorithm to compute primary/secondary paths that meet the delay constraints and conflict sets to avoid the "trap" problem.

CoSE-Pulse+ defines a set of sub-problems $I = (In, Ex)$, where $I.In$ is the set of Srlgs that must be included, and $I.Ex$ is the set of Srlgs that must be excluded (see line 1 in Algorithm 3). Then, the original problem is the sub-problem $I = \{\emptyset, \emptyset\}$. For each sub-problem $I = (In, Ex)$, CoSE-Pulse+ first uses **AP-Pulse**+ (this algorithm is similar to Pulse+, and thus we put the details in Appendix A-A) to find a primary path P_a such that $d(P_a) \le U$, $I.In \subseteq \Omega(P_a)$, $I.Ex \cap \Omega(P_a) = \emptyset$, and $\Omega(P_a)$ does not contain any conflict set found (see line 8 in Algorithm 3). If there exists an Srlg-disjoint path P_b for P_a, CoSE-Pulse+ updates the best path pair found so far. Otherwise, CoSE-Pulse+ computes a conflict Srlg set T and uses this set to create new problem instances (see lines 17-27 in Algorithm 3). More specifically, let $\{r_1, r_2, ..., r_N\}$ be the set of Srlgs in T but not in $I.In$. Since $\{r_1, r_2, ..., r_N\} \subseteq T \subseteq \Omega(P_a)$ and $I.Ex \cap \Omega(P_a) = \emptyset$, we must have $I.Ex \cap \{r_1, r_2, ..., r_N\} = \emptyset$. Then, we can divide the sub-problem $I = (In, Ex)$ into $I_1 = (I.In, I.Ex \cup \{r_1\})$ and $I_1^{'} = (I.In \cup \{r_1\}, I.Ex)$; $I_1^{'}$ can be further divided into $I_2 = (I.In \cup \{r_1\}, I.Ex \cup \{r_2\})$ and $I_2^{'} = (I.In \cup \{r_1, r_2\}, I.Ex)$; $I_2^{'}$ can be further divided into $I_3 = (I.In \cup \{r_1, r_2\}, I.Ex \cup \{r_3\})$ and $I_3^{'} = (I.In \cup \{r_1, r_2, r_3\}, I.Ex)$; and so on. Note that $I_N^{'} = (I.In \cup \{r_1, r_2, ..., r_N\}, I.Ex)$ is an infeasible instance, because the conflict set $T \subseteq I_N^{'}.In$. Hence, we obtain N sub-instances $I_1, I_2, ..., I_N$ for the sub-problem I. Note that there is a corner case where Conflict-Pulse+ fails to compute a conflict set. In this case, we simply use a trivial conflict set, which contains all the links of P_a (see line 22). After exploring all the sub-problems in Q, CoSE-Pulse+ either finds an optimal Srlg-disjoint path pair, or concludes that such an Srlg-disjoint path pair does not exist.

1) Algorithmic Complexity and Optimality Guarantee: As shown in Algorithm 3, for each problem instance, CoSE-Pulse+ calls AP-Pulse+ in line 8, calls Pulse+ in line 12 and calls Conflict-Pulse+ in line 17. Let $K_{\text{cose-pulse+}}$ be the total number of iterations of AP-Pulse+, Pulse+ and Conflict-Pulse+ for all the problem instances. Then, the algorithmic complexity of CoSE-Pulse+ is $O(K_{\text{cose-pulse+}} \deg_{\max})$. We will numerically study $K_{\text{cose-pulse+}}$ in Section VII-B.

Algorithm 3: CoSE-Pulse+

Data: A network $G(V, E)$, a source-destination pair (s, t), a delay upper bound U and a delay diff δ.
Result: The optimal primary path P_a^{opt} and an Srlg-disjoint path P_b.

1 Introduce a special Srlg $r_e = \{e\}$ for each link e.
2 Define a problem instance as $I = (In, Ex)$, where $I.In$ is the set of Srlgs that must be included, and $I.Ex$ is the set of Srlgs that must be excluded.
3 Use $\mathcal{T}$ to denote the conflict sets found. Init $\mathcal{T} = \emptyset$.
4 Define a problem instance queue Q. Init $Q = \{(\emptyset, \emptyset)\}$.
5 Use tmp_min and P_a^{opt} to track the best primary path found. Use P_b^{opt} to track the secondary path. Init tmp_min $= \infty$.
6 **while** Q *is not empty* **do**
7 Let $I = Q$.pop();
8 Try using the AP-Pulse+ algorithm to find a min-cost path P_a from s to t such that $d(P_a) \le U$, $I.In \subseteq \Omega(P_a)$, $I.Ex \cap \Omega(P_a) = \emptyset$, and $T \subsetneq \Omega(P_a)$ for any $T \in \mathcal{T}$.
9 **if** P_a *is not found* ***or*** $c(P_a) \ge$ *tmp_min* **then**
10 **continue**;
11 **end**
12 Try using Pulse+ to find an Srlg-disjoint path P_b from s to t such that $\Omega(P_a) \cap \Omega(P_b) = \emptyset$ and $d(P_a) - \delta \le d(P_b) \le \min\{U, d(P_a) + \delta\}$.
13 **if** *Pulse+ returns a feasible Srlg-disjoint path* P_b **then**
14 tmp_min $= c(P_a)$, $P_a^{\text{opt}} = P_a$, $P_b^{\text{opt}} = P_b$;
15 **continue**;
16 **end**
17 Use Conflict-Pulse+ to find a conflict set T for P_a.
18 **if** T *is not empty* **then**
19 Add T to $\mathcal{T}$;
20 Let $\{r_1, ..., r_N\}$ be the Srlgs in T but not in $I.In$;
21 **else**
22 Let $\{r_1, ..., r_N\} = \{r_e : e$ is a link of $P_a\}$;
23 **end**
24 **for** $n = 1, 2, ..., N$ **do**
25 Construct a new problem instance $I_n = (I.In \cup \{r_1, r_2, ..., r_{n-1}\}, I.Ex \cup \{r_n\})$;
26 Q.push(I_n)
27 **end**
28 **end**
29 Return P_a^{opt} and P_b^{opt}.

Theorem 2 guarantees the optimality of CoSE-Pulse+.

Theorem 2: (See Appendix A-D for the proof) For any Srlg-Disjoint DRCR problem instance, the solution generated by CoSE-Pulse+ is optimal.

VII. Evaluation

A. Generate Test Problem Instances

1) DRCR Cases:: We generate non-trivial DRCR test cases (see Section V) below.

Generate Topologies: We do not find any open source data for DetNet topologies. Although there are experimental deployments of DetNet in China's CENI and YZNET networks, we do not have the detailed topologies. In this paper, we use the topologies in Topology Zoo [30], an ongoing project to collect data network topologies from all over the world. Up to now, Topology Zoo contains hundreds of different topologies, and we pick 7 topologies (Cogentco, GtsCe, Interoute, Kdl, Pern, TataNld and VtlWavenet2008) of different sizes with $|V|$ =87-754 for evaluation. These topologies contain hundreds of nodes and links, which are similar in size to the CENI network[2].

CENI is just an experimental network in China. If DetNet starts offering service to public customers, the scale of DetNet will quickly increase. Indeed, there is a recent Internet Draft discussing how to scale DetNet so that multiple domains can be stitched to form a large DetNet [31]. In order to test the performance of our algorithm in large scale networks, we generate random topologies with different number of nodes and different node degrees. The generated topologies have 6 different sizes: 1) $|V| = 1000$; 2) $|V| = 2000$; 3) $|V| = 4000$; 4) $|V| = 6000$; 5) $|V| = 8000$; 6) $|V| = 10000$. For each size, the topologies have 3 different node degrees: 1) degree $= \ln|V|$; 2) degree $= 2\ln|V|$ 3) degree $= 3\ln|V|$. In order to reduce the randomness of the experimental results, we generate 10 topologies for any given values of $|V|$ and degree.

Generate Source-destination Pairs and Delay Ranges: For each topology, we randomly select a number of connected source-destination pairs (s,t). For each pair (s,t), we use Dijkstra algorithm to compute the min-delay path $P_{s\to t}^{\text{min\_delay}}$ and the min-cost path $P_{s\to t}^{\text{min\_cost}}$, and then assign different delay ranges $[L,U]$ randomly to form problem instances that belong to either of the two non-trivial cases. To meet DetNet's routing requirement, we set $L \approx 0.9U$.

2) Srlg-Disjoint DRCR Cases:: We generate both trap cases and non-trap cases below.

Generate Topologies: We use the topologies in DRCR for the Srlg-disjoint DRCR problem and add Srlgs to the links. We add Srlgs in two styles: the star style and the non-star style [27]. The star style is generally applied in optical networks while the non-star style can be used in other forms of network, such as the overlay network. We adopt different strategies to generate Srlgs of the two forms. For the star style, we randomly select the egress links of a node to be in a Srlg and the size of a Srlg is randomly determined based on the average degree of the topology. For the non-star style, we randomly select links in all the links to be in a srlg until every link is in at least one Srlg. The size of each Srlg is a random number in a given range, e.g. $[1, 40]$ in our implementation.

[2]CENI contains 88 backbone network nodes, 133 edge network test nodes, and four cloud data centers [3].

Generate Source-destinations Pairs and Delay Ranges: For each topology, we randomly select a number of connected source-destination pairs (s,t). For each pair (s,t), we use the Dijkstra algorithm to compute the min-delay path $P_{s\to t}^{\text{min\_delay}}$, and assign the delay upper bound as $U = 2.5d(P_{s\to t}^{\text{min\_delay}})$. Then we use CoSE-Pulse+ to test whether the test instance has a feasible solution (Other algorithms, such as KSP, may run indefinitely when a problem instance does not have a feasible solution.) and classify the test cases into trap and non-trap scenarios.

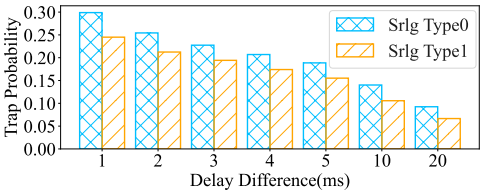


Figure 6. Trap Probability in Srlg-disjoint DRCR.

Trap problem in Srlg-disjoint DRCR. We conduct experiments to test the probability of encountering traps in the Srlg-Disjoint DRCR problems. As shown in Figure 6, the trap probability increases as the delay diff decreases. Recall from Section II-B that the primary and secondary paths in DetNet cannot have a large delay diff; otherwise the PEF may not guarantee deterministic delay in case of network failures. In our evaluation of Srlg-Disjoint DRCR problems, we set the delay diff of each flow as $1ms$ and the end-to-end delay upper bound in the range of $[50ms, 200ms]$. In this case, about 10% of all test cases encounter trap problem. We will evaluate different algorithms for both the trap and the non-trap cases.

B. Solving DRCR Problems

We compare Pulse+ with another three algorithms designed for the DRCR problem: 1) Cost-based KSP (see Appendix B-A1 in our technical report [24]), 2) Lagrangian-Dual based KSP (see Appendix B-A2 in our technical report [24]) and 3) Delay-based KSP (see Appendix B-A1 in our technical report [24]). We have tried our best to optimize the code for all these algorithms to improve efficiency. For example, we have adopted the A* algorithm to accelerate the shortest path search, which benefits all the KSP-based algorithms (see our source code in [1] for details). We use these these algorithms to solve all the problem instances. All experiments use a single thread of the AMD Ryzen 5600 @3.60GHz CPU on an Ubuntu workstation. Note that different test cases have different running times. Here, we calculate the percentile values and summarize the results in Figure 7. We can see that Pulse+ can finish all the test cases within 200 milliseconds, while other algorithms cannot with a time limit of 10 seconds (LagrangianKsp, CostKsp, DelayKsp achieve completion rates of 95%, 75%, and 24% respectively). In Figure 9(a), we summarize the completion rate versus the network size for all the algorithms. Compared to CostKsp and DelayKsp, we can see that as the network size increases, the advantage of Pulse+ becomes more evident than other algorithms. For LagrangianKsp, the completion rate only decreases slightly

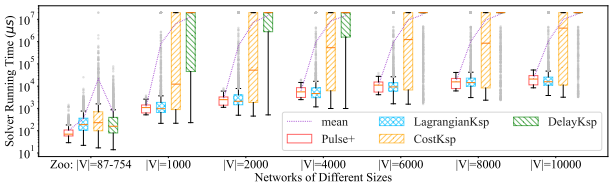


Figure 7. Percentile Values of the Solver Running Time for DRCR problems (μs).

with respect to the network size. However, the tail solver running time of LagrangianKsp is much higher than Pulse+. For each algorithm, we calculate the average solver running time (excluding the problem instances that cannot be solved in 10 seconds) as below: Pulse+ takes $10ms$, LagrangianKsp takes $53ms$, CostKsp takes $227ms$, and DelayKsp takes $429ms$.

Why Pulse+ Perform Better? According to Section V-B4 and Appendix B-A2 of our technical report [24], the algorithmic complexities of Pulse+, DelayKSP, CostKSP and LagrangianKsp are $O(K_{\text{pulse+}}\deg_{\max})$, $O(K_{\text{delay-ksp}}NH_{\max}(\log|V|+\deg_{\max}))$, $O(K_{\text{cost-ksp}}NH_{\max}(\log|V|+\deg_{\max}))$ and $O(K_{\text{lag}}NH_{\max}(\log|V|+\deg_{\max}))$, respectively. Here $|V|$ is the network size and $H_{\max}$ is the number of hops of the longest path from source to destination. We numerically evaluate all the K's in Figure 10(a). Statistically speaking, K_{lag} is smaller than $K_{\text{delay-ksp+}}$ and $K_{\text{cost-ksp+}}$. This explains why LagrangianKsp performs better than DelayKSP and CostKSP. For Pulse+, the polynomial part $\deg_{\max}$ is much smaller than the polynomial part $NH_{\max}(\log|V|+\deg_{\max})$ of LagrangianKsp, while the median value of $K_{\text{pulse+}}$ is about 2-3 orders larger than that of K_{lag}. This explains why the median solver running time of Pulse+ is similar to that of LagrangianKsp (see Figure 7). In addition, the tail of K_{lag} is larger than $K_{\text{pulse+}}$, which explains why the tail solver running time of LagrangianKsp is much larger. In the worst case, both Pulse+ and LagrangianKsp need to explore all the paths from the source to the destination. Fortunately, the overhead of exploring a path in Pulse+ is much smaller than that in LagrangianKsp, because Pulse+ does not need to sort the paths. This could be another reason that Pulse+ runs faster than LagrangianKsp in the worst case.

In the above evaluation, all the algorithms are actually proposed by ourselves. In the literature, there is indeed an algorithm proposed by Celso that aims at solving the DRCR problem [22]. However, Celso's algorithm cannot guarantee optimality. We test Celso's algorithm on the 7 topologies chosen from the Zoo dataset. As shown in Table I, we can see that Celso's algorithm yields sub-optimal solutions for a majority of the test cases. (Celso's algorithm achieves 100% accuracy in the Pern topology. The reason is that the Pern topology only contains 4 loops and is much simpler than other topologies.) Moreover, the solver running of Celso's algorithm is over $100\times$ longer than that of Pulse+. Therefore, Pulse+ is better than Celso's algorithm in terms of both optimality and solver running time.

Topology	Cogentco	GtsCe	Interoute	Kdl	Pern	TataNld	VtlWavenet2008
Accuracy of Celso	75/740	128/1344	107/408	38/1012	88/88	78/410	6/12
Average solving time of Celso (μs)	13227.01	11715.26	11329.48	75008.63	9281.61	13845.19	9487.25
Average solving time of Pulse+ (μs)	80.39	77.04	62.1	218.82	59.73	66.18	41.75

Table I
COMPARISON OF PULSE+ AND CELSO'S ALGORITHM.

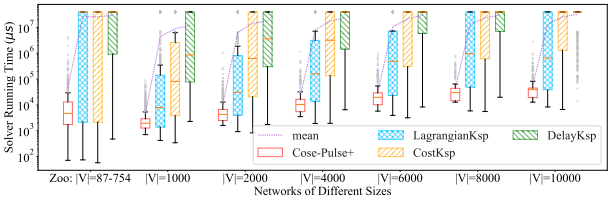


Figure 8. Percentile Values of the Solver Running Time for Srlg-disjoint DRCR problems (μs).

C. Solving Srlg-disjoint DRCR Problems

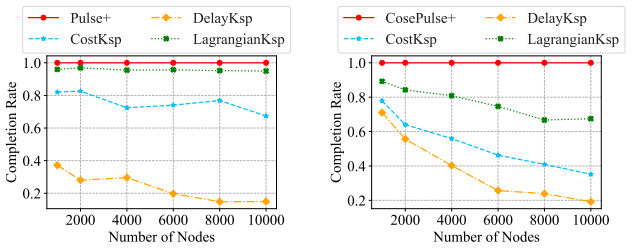
We compare CoSE-Pulse+, Cost-KSP, Lagrangian-KSP (Algorithm 8 in Appendix B-A2 of our technical report [24]) and Delay-KSP. Again, we set a time limit of 10 seconds for each problem instance. The experiment results are summarized in Figure 8 and Figure 9(b). CoSE-Pulse+ successfully solves all the test cases within 10 seconds. In contrast, the overall completion rates of LagrangianKsp, CostKsp, and DelayKsp are only 77%, 53%, and 39%, respectively. Moreover, as the network scale increased, the completion rates also decrease. For each algorithm, the average solver running time (excluding the problem instances that cannot be solved in 10 seconds) is summarized below: CoSE-Pulse+ takes $28ms$, LagrangianKsp takes $630ms$, CostKsp takes $867ms$, and DelayKsp takes $1490ms$.

Why CoSE-Pulse+ Perform Better? According to Section VI-B1 and Appendix B-A2 of our technical report [24], the algorithmic complexities of CoSE-Pulse+ and LagrangianKsp are $O(K_{\text{cose-pulse+}}\deg_{\max})$ and $O(K_{\text{lag}}NH_{\max}(\log|V|+\deg_{\max}))$. From Figure 10(b), we can see that the median values of $K_{\text{cose-pulse+}}$ and K_{lag} are on the same order and the tail of $K_{\text{cose-pulse+}}$ is much smaller. In addition, the polynomial part $\deg_{\max}$ of CoSE-Pulse+ is much smaller than the polynomial part $NH_{\max}(\log|V|+\deg_{\max})$ of LagrangianKsp. Therefore, CoSE-Pulse+ runs much faster than LagrangianKsp, as depicted in Figure 8.

D. Memory Consumption of Pulse+ and CoSE-Pulse+

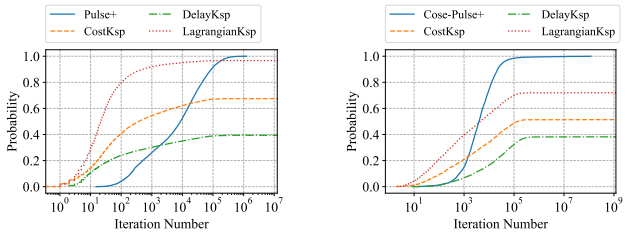
In Pulse+, we use a stack S to store all the branches to be explored (see line 3 in Algorithm 2). The number of branches to be explored can be huge. Thus, readers may worry about the memory consumption of Pulse+ and CoSE-Pulse+.

We evaluate the memory consumption of Pulse+ and CoSE-Pulse+ for all the test cases generated in Section VII-A. For each topology size, we average over all the test cases to obtain the average memory consumption of Pulse+ and CoSE-Pulse+. As shown in Figure VII-D, the average memory consumption of Pulse+ and CoSE-Pulse+ is under 100 MB, which is acceptable for today's computers. Note that, Pulse+ and CoSE-Pulse+ can be also implemented using recursive calls. However, the overhead of recursive calls would be much higher. That is why we use a stack to eliminate recursion.



(a) DRCR problem. (b) Srlg-disjoint DRCR problem.

Figure 9. Completion rates of different algorithms. The time limit is 10 seconds for each test case.



(a) CDFs of $K_{\text{pulse+}}$, $K_{\text{delay-ksp+}}$, $K_{\text{cost-ksp+}}$ and K_{lag} for the DRCR problem.

(b) CDFs of $K_{\text{cose-pulse+}}$, $K_{\text{delay-ksp+}}$, $K_{\text{cost-ksp+}}$ and K_{lag} for the Srlg-disjoint DRCR problem.

Figure 10. Compare the total number of iterations.

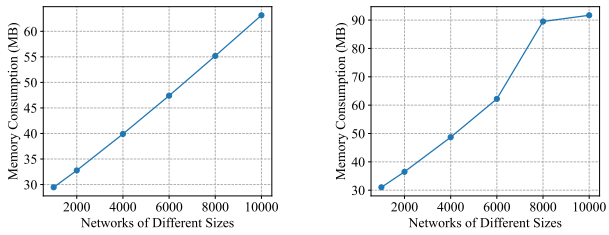
E. Computing DetNet Routing Paths in Batches

In the previous sections, we evaluate the solver running time for one request at a time. In practice, multiple DetNet flows may arrive roughly at the same time and are processed in one batch. The DetNet controller should be able to compute paths for all the DetNet flows in a short time.

We perform a batch test for different solvers. First, we generate three 1000-node topologies using the same approach in Section VII-A. Then, for each topology, we randomly generate 50 DetNet flows for both the DRCR problem and the Srlg-disjoint DRCR problem. For each DetNet flow, we set a limit of 10 seconds. If a solver cannot compute a solution for a DetNet flow within 10 seconds, it will skip this flow. After finishing all the 50 test cases, we record the total solving time and the completion rate for each solver. The batch test is repeated 3 times, each with a different topology, and the results are averaged over the 3 runs. As shown in Figure VII-E, Pulse+ and CoSE-Pulse+ can finish all the test cases with the least amount of time, with over 10× speedup compared to the second best algorithm.

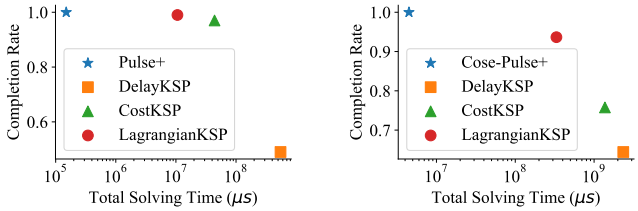
VIII. Do we really need DetNet?

Applications like telesurgery [2], PLC remote control [3], etc., require deterministic end-to-end delay. However, is DetNet the only choice to achieve deterministic delay? Currently, DetNet is just a *Proposed Standard* [4]. (According to RFC 6410 [32], two separate implementations and widespread use are required to advance an RFC from *Proposed Standard* to *Internet Standard*.) In contrast, current Internet already supports priority queuing. If we assign the highest priority



(a) Pulse+ (b) CoSE-Pulse+

Figure 11. Memory consumption of Pulse+ and CoSE-Pulse+.



(a) DRCR (b) Srlg-disjoint DRCR

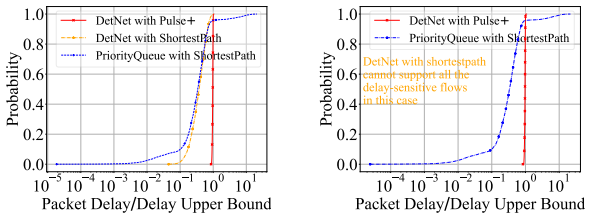
Figure 12. Batch test for DRCR and Srlg-disjoint DRCR.

to mission-critical flows, can we also achieve deterministic delay?

To answer this question, we perform packet-level simulations. We implement DetNet routing, including the Pulse+ routing algorithm, on top of NetBench [33]. The network topology is one of the generated topology described in Section VII, consisting of 1000 nodes and 6998 directed edges. Each edge has a propagation delay of 1-100μs and a bandwidth of 100Gbps. The network has a total of 10,000 flows, with $10000p$ DetNet flows and $10000(1-p)$ best-effort flows ($0 < p < 1$). For DetNet, each switch port contains multiple gate-controlled queues and a regular queue. The gate-controlled queues are used to handle DetNet flows, while the regular queue is used for best-effort flows. The gate-controlled queues open in a round-robin fashion. In each switch cycle, only one gate-controlled queue is open. Each cycle lasts 2.4μs, being able to transmit 20 1500-byte packets in one cycle. To achieve deterministic delay, we specify the queue each DetNet flow needs to enter at each switch and ensure that all the packets in the open gate-controlled queue can be transmitted within one cycle by proper rate limiting. To avoid link under utilization, when an open gate-controlled queue finishes transmitting all its packets, the packets in the regular best-effort queue can be transmitted. For DetNet flows, packets are generated periodically, and the Pulse+ algorithm is used to generate paths for each DetNet flow. For best-effort flows, shortest paths are adopted and TCP Reno is used for congestion control.

We compare the above "DetNet with Pulse+" setting with two other settings under the same topology and the same number of delay-sensitive and best-effort flows.

1) "DetNet with ShortestPath": All the flows take the shortest paths.
2) "PriorityQueue with ShortestPath": All the flows take the shortest paths. Each switch port contains a high-



(a) Number of Delay-Sensitive Flow: 4000

(b) Number of Delay-Sensitive Flow: 6000

Figure 13. Comparing the normalized packet delay under differnet queueing and routing policies.

priority queue and a low-priority queue; DetNet flows enter the high-priority queue, while best-effort flows enter the low-priority queue.

We plot the CDF (cumulative distribution function) of the normalized end-to-end packet delays (the end-to-end packet delay divide the delay upper bound) for all the delay-sensitive flows in Figure 13. It can be observed that DetNet can indeed guarantee deterministic delay for delay-sensitive flows. In contrast, some packets could miss its deadline under the priority queue scheme, i.e., the normalized end-to-end delay could be large than 1. In addition, when shortest path routing is used with DetNet, the end-to-end delay upperbound can be met. However, 1) many packets may arrive early, which could hurt network determinism and more importantly, 2) the shortest path routing has lower path diversity and thus cannot support as many DetNet flows as Pulse+.

IX. Discussion on DetNet

Is it practical to obtain global topology information? As mentioned in RFC 8655 [4], DetNet is designed for networks that are under a single administrative control or within a closed group of administrative control. This allows DetNet to obtain global network information and use Pulse+ or CoSE-Pulse+ to compute routing solutions. Such a centralized control can be realized using SDN technologies [34].

How to handle network changes? For sudden network changes such as link failures, node failures, etc., 1+1 protection can be used to mitigate the impact of failures. If the network changes is not transient, DetNet will resort to the central controller to recompute a pair of paths using Pulse+ and CoSE-Pulse+.

How to determine the delay requirements for DetNet flows? The actual end-to-end delay requirements are determined by the applications, rather than the network. Given an infeasible delay requirement, the DetNet controller is responsible to notify the application. The application can make decisions on whether to relax the end-to-end delay requirements or cancel the flow.

How to schedule transmission cycles for DetNet flows? After obtaining the routing paths for each DetNet flow, we could use a central controller to schedule transmission cycles following the design principles below. The packets allocated to each cycle at each output interface cannot exceed the maximum number of packets that can be sent in a cycle. Otherwise, contention and deadline miss may happen. This requirement also enforces each DetNet flow to regulate its traffic using certain rate limiting and shaping functions [4].

How to improve DetNet flow's admission rate? As the number of admitted DetNet flows increases, some links may not have sufficient resources to schedule additional DetNet flows. In this case, we could increase the cost of the congested links. Then, CoSE-Pulse+ or Pulse+ will avoid these links. This simple scheme could achieve better load balance and increase DetNet flow's admission rate.

How to avoid link under-utilization? DetNet flows reserve transmission cycles to achieve deterministic delay and jitter. Due to the rate fluctuation, some cycles may not have enough DetNet packets to send. In this case, best-effort packets can be transmitted.

X. Related Work on DetNet Routing

Pulse+ and CoSE-Pulse+ meet all the routing requirements of DetNet flows in large networks with thousands of nodes and links. To the best of our knowledge, none of the existing solutions could achieve this objective.

Most works on DetNet routing and scheduling did not account for network failures [35]–[40]. The RFC standard of DetNet proposed using multiple paths to protect against network failures [4]. A recent paper [21] formulated the primary/secondary path finding problem using integer programming, but the computational complexity is too high.

From the pure algorithm design's point of view, the link/Srlg-disjoint path finding problems have been studied with an objective to minimize 1) the sum cost of both paths [41]–[44] or 2) the min cost of the two paths [14], [15], [25]–[27], [45]. However, none of these works could handle delay constraints.

The DRCR problem arises as a sub-problem of the Srlg-disjoint DRCR problem. Due to the delay diff requirement, a delay lower bound is imposed. Most existing literature on delay constrained routing does not account for the delay lower bound constraints [5], [7]–[13]. Although the algorithm proposed in [22] directly handles delay lower bounds, it cannot guarantee optimality.

The DRCR problem is similar to another line of research works [46]–[49], i.e., the Vehicle Routing Problem with Time Windows (VRPTW). Given a graph $G(V, E)$, each link $e \in E$ is associated with a delay-cost pair $(d(e), c(e))$ and each node $v \in V$ is associated with a time window $[L_v, U_v]$. The objective is to deliver a service from s to t, such that the delivery time is in $[L_t, U_t]$. Note that the service in the VRPTW problem is allowed to arrive at a node v earlier than L_v and then wait until L_v to start its next delivery. In contrast, our DRCR problem does not allow early arrival. In DetNet, network switches may not have enough memory to buffer the early-arrival packets.

XI. Conclusion

DetNet introduces stringent routing requirements to achieve deterministic end-to-end delay under both normal and failure scenarios. We propose Pulse+ and CoSE-Pulse+ to solve DetNet's routing challenges. Pulse+ and CoSE-Pulse+ not

only have theoretical optimality guarantee, but also exhibit lower algorithmic complexity compared to other algorithms. Pulse+ and CoSE-Pulse+ make it possible to achieve fast routing computation in large-scale DetNets with thousands of nodes and links.

APPENDIX A

A. AP-Pulse+: Active Path Search

Given a sub-problem instance $I = (In, Ex)$ and a set $\mathcal{T}$ of conflict Srlg sets, we use AP-Pulse+ to search for the min-cost primary path P_a that satisfies the following constraints (see line 7 in Algorithm 3):

1) No Srlg in $I.Ex$ is included in $\Omega(P_a)$: $I.Ex \cap \Omega(P_a) = \emptyset$;
2) Delay constraint: $d(P_a) \leq U$;
3) All the Srlgs in $I.In$ must be in $\Omega(P_a)$: $I.In \subseteq \Omega(P_a)$;
4) For every conflict Srlg set $T \in \mathcal{T}$, $T \subsetneq \Omega(P_a)$.

To obtain AP-Pulse+, we modify Pulse+ as follows:

1) **Preparation stage**: For every Srlg $r \in I.Ex$, disable all the links contained in the Srlg r. This step ensures that the constraint (1) is met, i.e., $I.Ex \cap \Omega(P_a) = \emptyset$.
2) **Path validation** (the box in line 7 of Algorithm 2):

$$\boxed{\begin{array}{l} d(P_{s\to t}) \leq U \text{ and } I.In \subseteq \Omega(P_{s\to t}) \text{ and} \\ T \subsetneq \Omega(P_{s\to t}) \text{ for any } T \in \mathcal{T}. \end{array}} \quad (5)$$

   This step ensures that the constraints (2)-(4) are met.
3) **Prune strategy** (the box in line 15 of Algorithm 2):

$$\boxed{\begin{array}{l} d(P_{s\to u}) + d(P^{\text{min\_delay}}_{u\to t}) > U \\ \text{or } c(P_{s\to u}) + c(P^{\text{min\_cost}}_{u\to t}) \geq \text{tmp\_min} \\ \text{or } \exists T \in \mathcal{T} \text{ such that } T \subseteq \Omega(P_{s\to u}). \end{array}} \quad (6)$$

   Compared to the original pruning strategy (4), the above pruning strategy introduces the "conflict check": If a branch $P_{s\to u}$ contains a conflict Srlg set, then this branch is skipped.

B. Conflict-Pulse+

The detailed algorithm of Conflict-Pulse+ is shown in Algorithm 4. In Conflict-Pulse+, the conflict Srlg set T is initialized as an empty set (see line 1). When Conflict-Pulse+ finds a path $P_{s\to t}$ satisfying $d(P_{s\to t}) \leq U$, it checks if this path $P_{s\to t}$ is an Srlg-disjoint path of P_a. If it is, then Conflict-Pulse+ fails to find a conflict set (see lines 10-12); otherwise, Conflict-Pulse+ picks an Srlg $r \in \Omega(P_{s\to t}) \cap \Omega(P_a)$, disable all the links in r and insert r to T (see lines 13-15). Note that, when Conflict-Pulse+ generates new searching branches, only **active** egress links are explored (see lines 22-26). When Conflict-Pulse+ encounters a branch with disabled links, it will directly cut this branch (see lines 5-7). If Conflict-Pulse+ can reach line 28, then the resulting Srlg set T is a conflict Srlg set. This is guaranteed by the following theorem.

Theorem 3: Given a primary path P_a, if Algorithm 4 reaches line 28, the resulting set T must be a conflict set.

Proof 1: Since every Srlg $r \in T$ is chosen within the set $\Omega(P_a)$, we must have $T \subseteq \Omega(P_a)$. We next show that every path P satisfying $T \subseteq \Omega(P)$ does not have an Srlg-disjoint path that meets the delay-range requirement, i.e., T is a conflict set.

We prove by contradiction. Suppose that P'_a is a path satisfying $T \subseteq \Omega(P'_a)$ and P'_b is an Srlg-disjoint path of P'_a that meets the delay requirement. Clearly, $T \cap \Omega(P'_b) = \emptyset$.

Consider the searching branch that yields the path P'_b in Algorithm 4. This branch must be able to reach its final stage (lines 10-15). According to Algorithm 4, a branch can be only cut in two places: lines 5-7 and lines 19-21. First, P'_b does not contain any link e such that e belongs to an Srlg in T. Hence, the searching branch of P'_b cannot be cut at lines 5-7. Second, $d(P'_b) \leq U$. Thus, the branch of P'_b cannot be cut at lines 19-21, either.

Algorithm 4: Conflict-Pulse+

Data: A network $G(V, E)$, a source-destination pair (s, t), a delay upper bound U and a path P_a.
Result: A conflict Srlg set T.

1 Initialize the conflict Srlg set $T = \emptyset$.
`// Perform deep first search.`
2 Use a stack S to store all the branches to be explored. Initialize $S = \{\text{empty\_path}\}$.
3 **while** *S is not empty* **do**
4 Let path $P_{s\to u} = S$.pop(). Let u be the end node of $P_{s\to u}$. Set $u = s$ if $P_{s\to u}$ is empty.
5 **if** *there exists a disabled link in* $P_{s\to u}$ **then**
6 **continue**;
7 **end**
8 **if** $u == t$ **then**
`// Validate the path found.`
9 **if** $d(P_{s\to u}) \leq U$ **then**
10 **if** $\Omega(P_{s\to u}) \cap \Omega(P_a) = \emptyset$ **then**
`// Fail to find a conflict set.`
11 return an empty set;
12 **end**
13 Pick an Srlg $r \in \Omega(P_{s\to u}) \cap \Omega(P_a)$ such that r contains the largest number of links.
14 Disable all the links in the Srlg r.
15 T.insert(r);
16 **end**
17 **continue**;
18 **end**
`// Cut branches when possible.`
19 **if** $d(P_{s\to u}) + d(P^{min\_delay}_{u\to t}) > U$ **then**
20 **continue**;
21 **end**
`// Add new branches.`
22 **for** *every egress* ***active*** *link e of the node u* **do**
23 **if** *the node To(e) is not visited in* $P_{s\to u}$ **then**
24 S.push($P_{s\to u} \cup \{e\}$);
25 **end**
26 **end**
27 **end**
28 return the conflict set T;

When the P_b' branch reaches the final stage, it will not enter line 11; otherwise Algorithm 4 will fail to return a conflict set. Then, at line 13, an Srlg $r \in \Omega(P_b') \cap \Omega(P_a)$ will be chosen and added to T. This contradicts to the fact that $T \cap \Omega(P_b') = \emptyset$. Hence, T must be a conflict set.

Remark: From the above proof, we can see that the Srlg selection strategy in line 13 of Algorithm 4 is not critical for the correctness of Theorem 3. We pick the srlg that contains the largest number of links, because this choice could generate a relatively small conflict set.

C. Proof of Theorem 1

Proof 2: We prove by contradiction. Suppose that the solution $P_{s\to t}^{\text{opt}}$ returned by Pulse+ is not optimal. Then, there must exist another path P satisfying $L \le d(P) \le U$, such that $c(P) < c(P_{s\to t}^{\text{opt}})$. Consider the searching branch along the path P. At every intermediate node u of the path P, we must have $d(P_{s\to u}) + d(P_{u\to t}^{\text{min\_delay}}) \le d(P) \le U$ and $c(P_{s\to u}) + c(P_{u\to t}^{\text{min\_cost}}) \le c(P) < c(P_{s\to t}^{\text{opt}}) \le$ tmp_min. Hence, it is not possible to prune the path P's branch based on the strategies in (4). Hence, Pulse+ should be able to find a solution with cost no more than $c(P)$. This leads to a contradiction.

D. Proof of Theorem 2

Proof 3: We prove by contradiction. Suppose that the solution $(P_a^{\text{opt}}, P_b^{\text{opt}})$ returned by CoSE-Pulse+ is not optimal. Then, there must exist another pair of Srlg-disjoint path (P_a, P_b) satisfying $d(P_a) \le U$ and $d(P_a) - \delta \le d(P_b) \le \min\{U, d(P_a) + \delta\}$, such that $c(P_a) < c(P_a^{\text{opt}})$. Since P_a has an Srlg-disjoint path, for every conflict set T found in CoSE-Pulse+, we must have $T \subsetneq \Omega(P_a)$.

We have assumed that CoSE-Pulse+ terminates with a solution. Then, the total number of problem instances explored by CoSE-Pulse+ must be finite. We denote the set of explored problem instances by $\mathcal{I}$. Let $\mathcal{I}(P_a) \subseteq \mathcal{I}$ be the set of I's such that $I.In \subseteq \Omega(P_a), I.Ex \cap \Omega(P_a) = \emptyset$. Clearly, $\mathcal{I}(P_a)$ is not empty because $(\emptyset, \emptyset) \in \mathcal{I}(P_a)$. Within $\mathcal{I}(P_a)$, there must be an $I' \in \mathcal{I}(P_a)$ such that for every $I \in \mathcal{I}(P_a)$ and $I \ne I'$, $I'.In$ is not contained in $I.In$. Consider the problem instance I'. Let P_a' be the AP-Pulse+ solution of I'. Since P_a satisfies all the requirements of I', we must have $c(P_a') \le c(P_a) < c(P_a^{\text{opt}})$. In addition, P_a' does not have an Srlg-disjoint path; otherwise, P_a^{opt} would not be the optimal solution. Consider line 20 and line 22 of the CoSE-Pulse+ algorithm. Since $T \subsetneq \Omega(P_a)$ and $P_a \ne P_a'$, there must exist a $k \in \{1, ..., N\}$ such that $\{r_1, ..., r_{k-1}\} \subseteq \Omega(P_a)$ and $r_k \notin \Omega(P_a)$. According to lines 24-26 of the CoSE-Pulse+ algorithm, a new problem instance $I_k' = (I'.In \cup \{r_1, ..., r_{k-1}\}, I'.Ex \cup \{r_k\})$ will be generated. It is easy to verify that $I_k' \in \mathcal{I}(P_a)$ and $I'.In \subseteq I_k'.In$. This contradicts to the choice of I'.

REFERENCES

[1] https://gitee.com/zsz2019_shizhenzhao/drcr, Pulse+ Code Base.

[2] Z. Zhang, Y. Wang, Z. Zhang, J. Zheng, Z. Su, H. Gui, W. Jiao, X. Yang, and H. Niu, "Application of deterministic networking for reducing network delay in urological telesurgery: A retrospective study," *International Journal of Medical Robotics and Computer Assisted Surgery*, vol. 18, 2022.

[3] S. Wang, B. Wu, C. Zhang, Y. Huang, T. Huang, and Y. Liu, "Large-scale deterministic ip networks on ceni," in *IEEE INFOCOM Workshops*, May 2021.

[4] N. Finn, P. Thubert, B. Varga, and J. Farkas, *RFC 8655: Deterministic Networking Architecture*. https://datatracker.ietf.org/doc/html/rfc8655/, 2019.

[5] G. Y. Handler and I. Zang, "A dual algorithm for the constrained shortest path problem," *Networks*, vol. 10, pp. 293–310, 1980.

[6] J. E. Beasley and N. Christofides, "An algorithm for the resource constrained shortest path problem," *Networks*, vol. 19, 1989.

[7] L. Santos, J. Coutinho-Rodrigues, and J. R. Current, "An improved solution algorithm for the constrained shortest path problem," *Transportation Research Part B: Methodological*, vol. 41, pp. 756–771, 2007.

[8] I. Dumitrescu and N. Boland, "Improved preprocessing, labeling and scaling algorithms for the weight-constrained shortest path problem," *Networks*, vol. 42, pp. 135–153, 2003.

[9] X. Zhu and W. E. Wilhelm, "A three-stage approach for the resource-constrained shortest path as a sub-problem in column generation," *Computers & Operations Research*, vol. 39, pp. 164–178, 2012.

[10] B. W. Thomas, T. Calogiuri, and M. Hewitt, "An exact bidirectional a* approach for solving resource constrained shortest path problems," *Networks*, vol. 73, pp. 187–205, 2019.

[11] L. Lozano and A. L. Medaglia, "On an exact method for the constrained shortest path problem," *Computers and Operations Research*, vol. 40, pp. 378–384, 2013.

[12] A. Sedeo-Noda and S. Alonso-Rodrguez, "An enhanced ksp algorithm with pruning strategies to solve the constrained shortest path problem," *Applied Mathematics and Computation*, vol. 265, 2015.

[13] N. Cabrera, A. L. Medaglia, L. Lozano, and D. Duque, "An exact bidirectional pulse algorithm for the constrained shortest path," *Networks*, vol. 76, pp. 128–146, 2020.

[14] D. Xu, Y. Chen, Y. Xiong, and C. Qiao, "On finding disjoint paths in single and dual link cost networks," in *IEEE INFOCOM*, April 2004.

[15] M. J. Rostami, S. Khorsandi, and A. A. Khodaparast, "Cose: A srlg-disjoint routing algorithm," in *Fourth European Conference on Universal Multiservice Networks (ECUMN'07)*, February 2007.

[16] D. O. Awduche, L. Berger, D.-H. Gan, T. Li, V. Srinivasan, and G. Swallow, *RFC 3209: RSVP-TE: Extensions to RSVP for LSP Tunnels*. https://datatracker.ietf.org/doc/html/rfc3209/, 2001.

[17] C. Filsfils, S. Previdi, L. Ginsberg, B. Decraene, S. Litkowski, and R. Shakir, *RFC 8402: Segment Routing Architecture*. https://datatracker.ietf.org/doc/html/rfc8402/, 2018.

[18] D. Ceccarelli and Y. Lee, *RFC 8453: Framework for Abstraction and Control of TE Networks (ACTN)*. https://datatracker.ietf.org/doc/html/rfc8453/, 2018.

[19] N. Finn, J.-Y. L. Boudec, E. Mohammadpour, J. Zhang, and B. Varga, *RFC 9320: Deterministic Networking (DetNet) Bounded Latency*. https://datatracker.ietf.org/doc/rfc9320/, 2022.

[20] M. C. Chen, X. Geng, Z. Li, J. Joung, and J.-d. Ryoo, *Segment Routing (SR) Based Bounded Latency*. https://datatracker.ietf.org/doc/html/draft-chen-detnet-sr-based-bounded-latency, 2023.

[21] G. P. Sharma, W. Tavernier, D. Colle, and M. Pickavet, "Routing and scheduling for 1+1 protected detnet flows," *Computer Networks*, vol. 211, 2022.

[22] C. C. Ribeiro and M. Minoux, "A heuristic approach to hard constrained shortest path problems," *Discrete Applied Mathematics*, vol. 10, pp. 125–137, 1985.

[23] J. Y. Yen, "Finding the k shortest loopless paths in a network," *Management Science*, vol. 17, pp. 712–716, 1971.

[24] S. Zhao, X. Liu, T. Zhu, and X. Wang, *Technical Report*. Online:https://jhc.sjtu.edu.cn/%7eshizhenzhao/pulse.pdf, 2024.

[25] D. Xu, C. Qiao, and Y. Xiong, "An ultra-fast shared path protection scheme-distributed partial information management, part ii," in *ICNP*, June 2002.

[26] G. Li, D. Wang, C. Kalmanek, and R. Doverspike, "Efficient distributed path selection for shared restoration connections," in *IEEE INFOCOM*, June 2002.

[27] K. Xie, H. Tao, X. Wang, G. Xie, J. Wen, J. Cao, and Z. Qin, "Divide and conquer for fast srlg disjoint routing," in *48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, June 2018.

[28] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Mathematik*, vol. 1, pp. 269–271, 1959.

[29] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Trans. on Systems Science and Cybernetics*, vol. 4, pp. 100–107, 1968.

[30] http://www.topology-zoo.org/, The Internet Topology Zoo.

[31] P. Liu, Y. Li, T. Eckert, Q. Xiong, J.-d. Ryoo, S. Zhu, and X. Geng, *Requirements for Scaling Deterministic Networks*. https://datatracker.ietf.org/doc/draft-ietf-detnet-scaling-requirements/02/, 2023.

[32] D. Crocker, E. Burger, and R. Housley, *RFC 6410: Reducing the Standards Track to Two Maturity Levels*. https://datatracker.ietf.org/doc/html/rfc6410, 2011.

[33] https://github.com/ndal-eth/netbench, Netbench.

[34] M. Boucadair and C. Jacquenet, *RFC 7149: Software-Defined Networking: A Perspective from within a Service Provider Environment*. https://datatracker.ietf.org/doc/html/rfc7149/, 2014.

[35] J. Falk, F. Drr, and K. Rothermel, "Exploring practical limitations of joint routing and scheduling for tsn with ilp," in *2018 IEEE 24th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, 2018.

[36] N. G. Nayak, F. Drr, and K. Rothermel, "Incremental flow scheduling and routing in time-sensitive software-defined networks," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 5, pp. 2066–2075, 2018.

[37] N. G. Nayak, F. Dürr, and K. Rothermel, "Routing algorithms for ieee802.1qbv networks," *SIGBED Rev.*, vol. 15, no. 3, aug 2018.

[38] E. Schweissguth, D. Timmermann, H. Parzyjegla, P. Danielis, and G. Mhl, "Ilp-based routing and scheduling of multicast realtime traffic in time-sensitive networks," in *2020 IEEE 26th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, 2020.

[39] S.-H. Chang, H. Chen, and B.-C. Cheng, "Time-predictable routing algorithm for time-sensitive networking: Schedulable guarantee of time-triggered streams," *Computer Communications*, vol. 172, pp. 183–195, 2021.

[40] J. Krolikowski, S. Martin, P. Medagliani, J. Leguay, S. Chen, X. Chang, and X. Geng, "Joint routing and scheduling for large-scale deterministic ip networks," *Computer Communications*, vol. 165, p. 3342, Jan. 2021.

[41] J. W. Suurballe and R. E. Tarjan, "A quick method for finding shortest pairs of disjoint paths," *Networks*, vol. 14, pp. 325–336, 1984.

[42] J. Q. Hu, "Diverse routing in optical mesh networks," *IEEE Transactions on Communications*, vol. 51, pp. 489–494, 2003.

[43] T. Gomes, C. Simes, and L. Fernandes, "Resilient routing in optical networks using srlg-disjoint path pairs of min-sum cost," *Telecommunication Systems*, vol. 52, pp. 737–749, August 2011.

[44] J.-C. Bermond, D. Coudert, G. DAngelo, and F. Z. Moataz, "Finding disjoint paths in networks with star shared risk link groups," *Theoretical Computer Science*, vol. 579, pp. 74–87, May 2015.

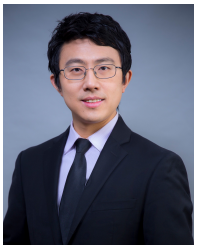
[45] B. Vass, E. Brczi-Kovcs, . Barabs, Z. L. Hajd, and J. Tapolcai, "Polynomial-time algorithm for the regional srlg-disjoint paths problem," in *IEEE INFOCOM*, May 2022.

[46] M. Desrochers, J. Desrosiers, and M. Solomon, "A new optimization algorithm for the vehicle routing problem with time windows," *Operations Research*, vol. 40, pp. 342–354, 1992.

[47] D. Feillet, P. Dejax, M. Gendreau, and C. Gueguen, "An exact algorithm for the elementary shortest path problem with resource constraints: Application to some vehicle routing problems," *Networks*, vol. 44, pp. 216–229, 2004.

[48] L. Lozano, D. Duque, and A. L. Medaglia, "An exact algorithm for the elementary shortest path problem with resource constraints," *Transportation Science*, vol. 50, pp. 1–10, 2015.

[49] L. Costa, C. Contardo, and G. Desaulniers, "Exact branch-price-and-cut algorithms for vehicle routing," *Transportation Science*, vol. 53, pp. 946–985, 2019.



Shizhen Zhao received the bachelors degree from Shanghai Jiao Tong University (SJTU) in 2010 and the Ph.D. degree from Purdue University in 2015. He is currently a Tenured Associate Professor with the John Hopcroft Center, SJTU. Before joining SJTU, he was with Googles Networking Team, managing Googles hyper-scale data center networks. He has published papers in top-tier conferences and journals, including SIGCOMM, NSDI, SIGMETRICS, MOBICOM, ICNP, INFOCOM, IEEE/ACM TRANSACTIONS ON NETWORKING, etc.

Ximeng Liu received the bachelors degree from Xi'an Jiao Tong University. He is currently pursuing his Ph.D. at the John Hopcroft Center for Computer Science, Shanghai Jiao Tong University. His research interests include traffic routing and traffic engineering.

Tianyu Zhu received the bachelors degree from Shanghai Jiao Tong University (SJTU). He is currently a graduate student at Shanghai Jiao Tong University. His primary research interests include computer networks and graph algorithms.

Xingbin Wang (Senior Member, IEEE) received the B.S. degree (Hons.) from Shanghai Jiao Tong University in 1998, the M.S. degree from Tsinghua University in 2001, and the Ph.D. degree from North Carolina State University in 2006. Currently, he is a Professor with the Department of Electronic Engineering, Shanghai Jiao Tong University. He has been a member of the technical program committees of several conferences, including MobiCom, MobiHoc, and INFOCOM. He has been an Associate Editor of IEEE TRANSACTIONS ON NETWORKING and IEEE TRANSACTIONS ON MOBILE COMPUTING.