# RFT: Toward Highly Reliable Flow Data Transmission in Network Measurement

Xi Sun, Xiang Chen, Di Wang, Zhengyan Zhou, Xinyue Jiang, Wenhai Wang, Chunming Wu, Haifeng Zhou
Zhejiang University, Zhejiang, Hangzhou, China

***Abstract*—How to satisfy the latency and reliability requirements of flow data transfer is an essential problem. To address this problem, we propose RFT, a framework that aims to satisfy the user-specified latency and reliability requirements of flow data transfer, especially in the situation where the network resources are insufficient. Firstly, we formulate the problem of satisfying the user-specified latency and reliability requirements of data transfer via mixed integer linear programming (MILP), and a heuristic algorithm is then designed to solve it in a polynomial-time. Secondly, to satisfy these requirements under insufficient network resources, we proposed a greedy-based algorithm used to select the minimum number of links added to the network, which can be deployed with low cost, especially in production networks such as data centers. Finally, we have implemented RFT on a 64×100 Gbps Intel Barefoot Tofino switch. Our experimental results indicate that RFT satisfies the user-specified latency and reliability requirements in all test cases at acceptable costs, even when the network resources are insufficient.**

## I. Introduction

In production networks, e.g., data center and ISP networks, network measurement plays a vital role in various network applications and management tasks, e.g., load balancing, heavy hitter detection, and DDoS detection [1]–[3]. In a measurement system, measurement points perform flow data collection tasks in the data plane, and deliver the collected flow date to monitoring servers in the control plane. The monitoring servers then transform the received flow data into traffic statistics, which is an essential input for numerous network applications and management tasks. Since the effectiveness of these network applications and management tasks heavily depends on the accuracy of the flow data, how to guarantee the flow data transfer in a low latency and high reliability way is one of the main concerns when designing a network measurement system in practice.

Current network measurement approaches can be classified into two categories according to their characteristics. The first category of approach allocates adequate transmission paths for the flow data during compile-time, e.g., [4]–[8]. Nevertheless, this category of approach is based on a potential assumption that network bandwidth resources are sufficient. As a result, if bandwidth resources are not adequate, the transfer latency and reliability requirements cannot be satisfied. The second category of approach saves limited network resources by reducing the amount of the flow data at runtime, e.g., [9]–[14]. However, these approaches also rely on a potential assumption that pre-switch resources are sufficient. For instance, approaches such as Marple [11] and *Flow [14] take advantage of in-switch memory to build least recently used (LRU) caches for temporarily storing flow data in measurement points until they can be safely emitted without incurring network congestion or suffering from potential data loss. Nonetheless, if the pre-switch resources are insufficient, the measurement points can only cache a small amount of data, and most of the data is under unreliable transmission.

Despite significant advances in network measurement, it is important to note that current approaches are built on a potential assumption that network resources for flow data transfer are sufficient. Consequently, how to guarantee flow data transfer to satisfy the low latency and high reliability requirements in the situation that network resources are insufficient remains a challenging task.

In this paper, we propose RFT, a network measurement framework that is capable to guarantee the user-specified transfer latency and reliability requirements of flow data even in the situation that network resources are insufficient to offer enough transfer paths. Firstly, to avoid network congestion and monitoring server overload, the selection and allocation of transfer paths for flow data based on limited network resources is formulated as a mixed integer linear programming (MILP), which is an NP-hard problem, and a heuristic algorithm is explored to solve it in polynomial time. Secondly, to handle the situation that network resources are insufficient to provide adequate transfer paths for flow data transfer, we propose to add a minimal number of extra links to the network for obtaining more transfer paths, when considering that the solution of this kind leaves low overheads and owns high feasible in many network scenarios. Furthermore, for satisfying transfer latency and reliability requirements, a greedy-based algorithm for the selection of the minimal number of links needed to add is designed. Finally, we implement RFT on a 64×100 Gbps Intel Barefoot Tofino switch. According to our experimental results in the 67 test cases, the deployment acceptance rate is 100% for RFT, has a 51% higher than current representative approaches, and the average run time for RFT is 0.32s, which is acceptable. All these experimental results indicate that RFT is competent to satisfy the user-specified transfer latency and reliability requirements in all test cases with acceptable costs, even if network resources are inadequate.

**Contributions.** To conclude, the main contributions of this paper are as follows.

- We analyze existing network measurement approaches and find that these approaches are based on a potential assumption

that network resources for flow data transfer are sufficient.

- We formulate the problem of selecting and allocating transfer paths under the transfer latency and reliability requirements of flow data as a mixed integer linear programming, propose to add extra links to the original network for satisfying the transfer latency and reliability requirements when limited network resources are incapable to afford enough transfer paths, and propose a greedy-based algorithm for selecting the minimal number of links needed to add.
- We have implemented RFT atop a Tofino-based hardware switch. Our experimental results indicate that RFT satisfy the user-specified transfer latency and reliability requirements in all test cases.

## II. PROBLEM FORMULATION

### A. Model of Network Measurement

The network measurement system consists of four parts: measurement points, monitoring servers, switches and links. The measurement point is a device used to monitor network performance, which can monitor network throughput, delay, packet loss rate and other parameters; the monitoring server is a server used to store the data collected by the measurement point, and can analyze, store and display. The switch is an important part of the network, responsible for forwarding the data packets in the network so that the data can be transmitted to the destination. Link refers to the communication link between two network nodes, which is the basis of data transmission. It can be represented by an undirected graph $G = (M_G, L_G, V_G, E_G)$. Among them, $M_G, L_G, V_G$ and $E_G$ respectively represent the collection of measurement points, monitoring servers, switches and links in the network measurement system.

- For each measurement point $m \in M_G$ is a device that runs one or more measurement tasks on the data plane (for example: a programmable switch). When running, the maximum rate at which the measurement point sends events to the monitoring server is denoted by $R(m)$ (in Gbps). To meet the requirements of both low latency and high reliability, the latency of allocated events sent by these tasks should be less than the cut-off time $\Phi_T(m)$ (in ms), the probability of event loss should be less than the threshold $\Phi_R(m)$.
- For each monitoring server $l \in L_G$ is used to collect and process the data sent by the measurement point, its processing capacity is $b_s(l)$ (in Gbps), and its current workload is $w_s(l)$ (in Gbps).
- For each switch $v \in V_G$ has four properties, including processing capacity $b_v(v)$ (in Gbps), current workload $w_v(v)$ (in Gbps), processing latency $t_v(v)$ (in ms) and the probability of being operational $r_v(v)$ $(0 < r_v(v) < 1)$.
- Each link $e_{(u,v)} \in E_G$ links two network nodes $u$ and $v$ $(u, v \in M_G \cup L_G \cup V_G)$, which has five properties. (1) $b_e\left(e_{(u,v)}\right)$ denotes the total bandwidth capacity (in Gbps). (2) $w_e\left(e_{(u,v)}\right)$ denotes the current workload. (3) $t_e\left(e_{(u,v)}\right)$ denotes the transmission latency. (4) $\rho_e\left(e_{(u,v)}\right)$ denotes the transmission reliability. (5) $r_e\left(e_{(u,v)}\right)$ $(0 < r_e\left(e_{(u,v)}\right) < 1)$ denotes the probability of being operational.

TABLE I: Notation of symbols

| | |
|---|---|
| $G$ | Substrate network $G = (M_G, L_G, V_G, E_G)$. |
| $m$,$l$,$v$ | Measurement point, Monitoring server, Data plane switch |
| $P(m,l)$ | Set of network paths connecting $m$ and $l$. |
| $e$,$p$ | A link in network, a path in network. |
| $\varepsilon(u,p)$ | Variable indicating if u exists in the path p. |
| $b_s$,$b_v$,$b_e$,$b_p$ | Total processing/bandwidth capability of $l$, $v$, $e$ and $p$. |
| $w_s$,$w_v$,$w_e$,$w_p$ | Current workload (in Gbps) of $l$, $v$, $e$ and $p$. |
| $t_v$,$t_e$,$t_p$ | Transmission latency (in milliseconds) of $v$, $e$, $p$. |
| $r_v$,$r_e$,$r_p$ | Probability of switch $v$, $e$ and $p$ of being operational. |
| $\rho_e$,$\rho_p$ | Transmission reliability of link $e$ and path $p$. |
| $\Phi_T(m)$ | Deadline of event collection for measurement point $m$. |
| $\Phi_R(m)$ | Probability of event loss for measurement point $m$. |
| $Y$ | Sum of available bandwidth capacity of operational paths. |
| $x_l$ | Variable indicating if server l is selected. |
| $y_l^m$ | Variable indicating if point m sends events to server l. |
| $\alpha_m(p)$ | Load of collecting events sent by m in path p. |
| $\beta_m(l)$ | Load of processing the events sent by m in l. |

The properties of workload, latency and operation probability are obtained by existing methods. (1) We invoke a central controller to measure workload [15]–[17]. (2) We use a low-cost probe-based approach to measure latency [18], [19]. (3) Reliability and probability are calculated based on historical network maintenance statistics [20]–[22].

Furthermore, for each pair $(m,l)$, the set $P(m,l)$ is used to denote all paths connecting $m$ and $l$. The path $p \in P(m,l)$ is a set of links and switches, using the 0-1 variable $\varepsilon(u,p)$ to indicates whether the link or switch $u \in (V_G \cup E_G)$ exists in the set $p$. If $u$ exists in $p$, $\varepsilon(u,p)$ = 1; otherwise, $\varepsilon(u,p)$ = 0. For each path $p$ there are five attributes:

(1) $b_p(p)$ denotes the total bandwidth capacity of $p$, which is equal to the minimum bandwidth capacity of the switches and links in path $p$:

$$b_p(p) = \min(\min(\sum_{\forall v \in V_G} \varepsilon(v,p) \cdot b_{\mathrm{v}(v)}), \min(\sum_{\forall e \in E_G} \varepsilon(e,p) \cdot b_e(e)))$$

(2) $w_p(p)$ denotes the current workload of $p$, which is equal to the maximum of the current loads of the switches and links in path $p$:

$$w_p(p) = \max(\max(\sum_{\forall v \in V_G} \varepsilon(v,p) \cdot w_{v(v)}), \min(\sum_{\forall e \in E_G} \varepsilon(e,p) \cdot w_e(e)))$$

(3) $t_p(p)$ is the transmission latency of $p$, which is the sum of the latency of all the links and switches resided in $p$:

$$t_p = \sum_{\forall v \in V_G} (\varepsilon(v,p) \cdot t_v(v)) + \sum_{\forall e \in E_G} (\varepsilon(e,p) \cdot t_e(e))$$

(4) $\rho_p(p)$ is the reliability of $p$ and it can be used to indicate the probability that the path will not fail.

$$\rho_p(p) = \prod_{\forall e \in E_G} \rho_e(e)$$

(5) $r_p(p)$ denotes the probability of $p$ of being operational, which is the product of the probability of every link or switch in $p$ of being operational:

$$r_p(p) = \prod_{\forall v \in V_G, \varepsilon(v,p)=1} p_v(v) \cdot \prod_{\forall e \in E_G, \varepsilon(e,p)=1} p_e(e)$$

### B. Problem Formulation

Given a network $G = (M_G, L_G, V_G, E_G)$, the problem is to select the monitoring servers from the set $L_G$ and assign the load of event collection to the network paths and the servers.

The solution contains four sets of decision variables. (1) Each variable $x_l$ in $\{x_l\}$ represents whether server $l$ has been selected. If $l$ is selected, $x_l = 1$; otherwise, $x_l = 0$. (2) Each variable $y_l^m$ in $\{y_l^m\}$ indicates whether the tasks running at measurement point $m$ send events to server $l$. If the tasks are sending events to $l$, $y_l^m = 1$, otherwise, $y_l^m = 0$. (3) Each variable $\alpha_m(p)$ in $\{\alpha_m(p)\}$ records the expected load of transferring sent by $m$ assigned to path $p$. (4) Each variable $\beta_m(l)$ in $\{\beta_m(l)\}$ records the expected load assigned to server $l$ for processing the events from $m$. The solution describes that a single measurement point can send events to multiple servers while a single server can receive events from multiple measurement points.

Next, the objective is set to minimize the number of servers in use to reduce monetary costs [8]:

$$min \sum_{\forall l \in L_G} x_l \tag{1}$$

We need to consider three types of constraints: (1) Sufficient network resources, including paths and servers, are available to collect events. (2) Paths and servers will not be overloaded. (3) Low latency and reliability of event collection are guaranteed.

(1) **Load assignment in the network path:** For each measurement point $m \in M_G$ with its rate $R(m)$ of sending events, the sum of the load of transferring the events sent by $m$ in network path should be at least $R(m)$:

$$\sum_{\forall l \in L_G} x_l \sum_{\forall p \in P(m,l)} y_l^m \cdot \alpha_m(p) \geq R(m), \forall m \in M_G \tag{2}$$

(2) **Load distribution in the monitoring server:** For each measurement point $m \in M_G$ with its rate $R(m)$ of sending events, the sum of the load of processing the events sent by $m$ in monitoring servers should be at least $R(m)$:

$$\sum_{\forall l \in L_G} x_l \sum_{\forall p \in P(m,l)} y_l^m \cdot \beta_m(l) \geq R(m), \forall m \in M_G \tag{3}$$

(3) **Avoiding path overload:** Given an arbitrary measurement point $m$ and an arbitrary monitoring server $l$, transferring events from $m$ to $l$ should not overload any path:

$$w_p(p) + x_l \cdot y_l^m \cdot \alpha_m(p) \leq b_p(p), \quad \forall m \in M_G, \forall l \in L_G \quad \forall p \in P(m,l) \tag{4}$$

(4) **Avoiding server overload:** Given an arbitrary measurement point $m$ and an arbitrary monitoring server $l$, processing the events sent by $m$ should not overload the server $l$:

$$w_s(l) + x_l \cdot y_l^m \cdot \beta_m(l) \leq b_s(l), \forall m \in M_G, \forall l \in L_G \tag{5}$$

(5) **Low latency:** For the measurement point $m \in M_G$, the latency of collection an event from $m$ to monitoring servers should be less than the deadline $\Phi_T(m)$:

$$\forall m \in M_G, \forall l \in L_G, \forall p \in P(m,l)\ x_l \cdot y_l^m \cdot t_p(p) \leq \Phi_T(m) \tag{6}$$

(6) **High reliability:** For the measurement point $m \in M_G$, we need to bound the probability of event loss incurred by network failures to at most a small value $\Phi_R(m)$ in order to guarantee high reliability. In other words, the probability that the sum of available bandwidth capacity of operational paths connecting $m$ and monitoring servers is less than the rate $R(m)$ of collecting events sent by $m$ should be less than $\Phi_R(m)$. Formally, let $Y$ denote the sum of available bandwidth capacity of operational paths. Then the above constraint is described as:

$$P[Y < R(m)] < \Phi_R(m) \tag{7}$$

A lower $\Phi_R(m)$ implies strict reliability guarantee at the expense of more bandwidth resources. However, solving this constraint is non-trivial given that each path has a probability of being operational. In response, we approximate this constraint via the Chernoff (lower-tail) bound [23]:

$$P[Y < (1-\delta)\mu] \leq \exp\left(-\frac{\mu}{2} \cdot \delta^2\right)$$

where $\mu$ is the expectation of $Y$:

$$\mu = E[Y] = \sum_{\forall l \in L_G} x_l \cdot y_l^m \sum_{\forall p \in P(m,l)} r_p(p) \cdot \alpha_m(p)$$

Next, we set $\delta$ to $1 - \frac{R(m)}{\mu}$. Then we have:

$$P[Y < R(m)] \leq \exp\left(-\frac{\mu}{2} \cdot \left(1 - \frac{R(m)}{\mu}\right)^2\right)$$

$$\exp\left(-\frac{\mu}{2} \cdot \left(1 - \frac{R(m)}{\mu}\right)^2\right) \leq \Phi_R(m) \tag{8}$$

### C. Challenge of Problem Solving

*Theorem* 1: Using mixed integer linear programming to solve the problem is NP-hard.

*Proof.* To prove Theorem 1, we consider a special case of the problem. This case applies three restrictions that simplify our problem. First, it restricts that each network path has infinite bandwidth, i.e., $b_p(p) = +\infty$ ($\forall p$). Second, it restricts that the probability of each path of being operational is one, i.e., $r_p(p) = 1$ ($\forall p$). Third, it restricts that the deadlines of event collection are unlimited, i.e., $\Phi_T(m) = +\infty$ ($\forall m \in M_G$). These restrictions reduce the problem to the NP-hard bin packing problem [24]. Thus, Theorem 1 follows. □

The problem can be solved by ILP solver. However, when the number of nodes and links grows, the execution time of ILP solvers becomes non-trivial and unacceptable. More specifically, the total number of decision variables for $x_l$ and $y_l^m$ is $2^{|M_G|} + 2^{|M_G| \cdot |L_G|}$ and the total number of constraints is $O(|M_G| \cdot |L_G| \cdot |E_G|)$. The ILP solver needs to enumerate all decision variables while explicitly checking all constraints for each enumeration. This yields a time complexity of $O((2^{|M_G|} + 2^{|M_G| \cdot |L_G|}) \cdot |M_G| \cdot |L_G| \cdot |E_G|)$. In addition, since it does not consider the case where the existing paths does not have enough bandwidth, in such a situation it cannot find a solution that satisfies the constraint (6). In our experience,

for a network with 100 nodes and 300 links, after running the ILP solver for 3-4 days, we can only get one feasible solution, which is inefficient. Thus, problem solving is challenging.

## III. Problem Solving

### A. Overview

Our goal is to provide a solution for the path assignment of measurement events that satisfies effectiveness and timeliness.

- **Effectiveness:** The collection of measurement events should satisfy low latency and high reliability.
- **Timeliness:** The solution should satisfy the worst-case running time within a polynomial.

To achieve our goals, we have divided the entire solition into two parts: (1) The process of path assignment is to select enough paths to ensure that the events collected by measurement points are transferred to monitoring servers while preserving low latency and high reliability. (2) The process of adding links is used when the existing paths are not enough to satisfy the requirements of low latency and high reliability, it adds links to the network so that it can use more paths to achieve our goals.

### B. Path Assignment

To address the problem of inefficient ILP solving, we design a heuristic algorithm for this problem, which aims to reduce the complexity of the algorithm at the expense of the optimality of the solution. The workflow of the algorithm is as follows:

(1) **Network initialization and path pruning:** Algorithm 1 takes the network model $G$ as input and records its decisions in $\{x_l\}$, $\{y_l^m\}$, $\{\alpha_m(p)\}$ and $\{\beta_m(l)\}$. In addition, it ignoring all paths of excessive length from the measurement point $m$ to the monitoring server $l$. Thus, constrain (5) can be satisfied.

(2) **Path selection:** It enumerates every path connecting $m$ to $l$ and ranks these paths according to their reliability. Thus, constraint (6) can be satisfied. For path $p \in P(m,l)$, it first determines whether $p$ has available bandwidth. If not, it turns to the next path. Otherwise, it assigns a fraction of the load of the transfer events $\alpha_m(p)$ sent by $m$ to path $p$, $\alpha_m(p)$ calculated as the minimum of the available bandwidth $b_p(p)$ of $p$, the remaining load $R(m)$ of $m$ and the available processing capability $b_s(l)$ of server $l$ to avoid overloading the path or the server. Thus, constraints (3) (4) can be satisfied.

(3) **Server selection:** The algorithm sorts these servers in ascending order of available processing capacity. For each measurement point $m$, the events are assigned to these servers based on the result of the sorting. Thus, constraint (2) can be satisfied. Next, it determines if $l$ has available capacity. If not, it turns to the next server. Otherwise, it assigns a fraction of the load of the transfer events to $l$ and sets $y_l^m$ to 1. Note that we heuristically reuse servers that have already been used. This reduces the number of servers needed and therefore reduces the cost.

*Theorem* 2: In the worst case the time complexity of Algorithm 1 is $O(|M_G| \cdot |L_G| \cdot (|P_{max}(m,l)|^2 + E_G + V_G))$.

**Algorithm 1** Path Assignment

**Input:** Network $G = (M_G, L_G, V_G, E_G)$
**Output:** Sets $\{x_l\}, \{y_l^m\}, \{\alpha_m(p)\}, \{\beta_m(l)\}$.

```
1:  function PATH_ASSIGNMENT(G)
2:      Initialization: set {x_l}, {y_l^m}, {α_m(p)} and {β_m(l)}
3:      for m ∈ M_G do
4:          H ← ∅
5:          for l ∈ L_G do
6:              H[(m,l)] ← Sorted_Paths(P(m,l))
7:          for (m,l_i), P(m,l_i) ∈ H.items do
8:              x_l ← 1
9:              if R(m) == 0: break
10:             if b_s(l_i) == 0: break
11:             y_l^m ← 1
12:             for p ∈ P(m,l_i) do
13:                 if R(m) == 0 or b_s(l_i) == 0: break
14:                 if b_p(p) == 0: continue
15:                 α_m(p) ← min(R(m), b_s(l_i), b_p(p))
16:                 R_m ← R_m − α_m(p)
17:                 b_s(l_i) ← b_s(l_i) − α_m(p)
18:                 b_p(p) ← b_p(p) − α_m(p)
19:                 β_l(m) ← β_l(m) + α_m(p)
20:     return {x_l}, {y_l^m}, {α_m(p)}, {β_m(l)}
```

*Proof.* Algorithm 1 enumerates every measurement points. For $m \in M_G$, it enumerates all servers and ranks the paths connecting to server $l$. The complexity of this process is $O(|P(m,l)|^2)$. For each pair $m$ and $l$, the followings must be checked when assigning paths: (i) whether $m$ has unassigned load. (ii) whether path $p$ is overloaded. (iii) whether server $l$ has remaining load. Then it decides $\alpha_m(p)$. This process is relevant to $|P(m,l)|$ (the number of paths from $m$ to $l$) so the time complexity is $O(|P(m,l)|)$. Additionally the time complexity of solving all the paths from $m$ to $l$ is relevant to the number of nodes and edges of the network as $O(|E_G| + |V_G|)$. In summary, the time complexity of Algorithm 1 in the worst-case is $O(|M_G| \cdot |L_G| \cdot (|P_{max}(m,l)|^2 + E_G + V_G))$. □

### C. Adding Links

If all the paths from the measurement point $m$ to the monitoring servers are not enough to transmit all the events, or if the assignment does not preserve the constraints of low latency and high reliability. We add a num of links to the network to increase the number of paths from measurement points to monitoring servers. So we can use more paths to transfer events.

To evaluate the fitness of the adding a link $e_{(u_a,v_a)}$ to the network, we define that the output of the fitness function is the sum of number of the added paths for each pair of a measurement point and a monitor server.

To find the best strategy of adding links, we explore a number of algorithms as follows:

**Brute-and-Force**: Enumerate all possible solutions for adding links to the network and find the optimal solution by calculating the fitness of each solution.

*Theorem* 3: The time complexity of this algorithm is $O(|M_G|^3 \cdot |L_G|^3 \cdot |V_G|^2 \cdot (|E_G| + |V_G|))$ with large time consumption.

*Proof.* The algorithm enumerates all possible cases of links with a total of $\frac{(|M_G|+|L_G|+|V_G|)^2}{2}$. For each link, it calculates its fitness using Algorithm 2 and record the best link, the complexity of this operation is $O(|M_G| \cdot |L_G| \cdot (|E_G| + |V_G|))$. Thus, the complexity is $O(|M_G|^3 \cdot |L_G|^3 \cdot |V_G|^2 \cdot (|E_G| + |V_G|))$. □

The running time of the algorithm is strongly correlated with the number of nodes in the network. If it is very large, this algorithm will consume a lot of computing resources. Through experiments, we have found that it takes 2-3 hours to run on a network with 100 nodes and 300 links which is inefficient.

**Particle Swarm Optimization:** Particle Swarm Optimization (PSO) is an optimization algorithm based on swarm intelligence [25]. It can be used to find the optimal solution for adding links. The basic idea of this algorithm is to simulate the motion of a swarm of particles in space. The workflow is as follows:

(1) **Initialise the particle swarm**: A particle swarm is a group of several particles, in this case, each particle has a position to represent the solution and a velocity to represent the direction in which the particle will be updated in the solution space.
(2) **Calculate fitness**: Update the current optimal position and the global optimal position of the particle swarm according to the fitness.
(3) **Update particle swarm**: In each iteration, it updates the position and velocity of the particles based on their current position, the global optimal position, and the group optimal position. After that, it moves on to the next round of iterations.

*Theorem* 4: The time complexity of the PSO is several orders of magnitude lower than the Brute-and-Force.

*Proof.* The algorithm calculates the fitness of each particle in each iteration and updates the global optimal solution, the group optimal solution and the position of the particles. So the time complexity of the algorithm is $O(N_{iters} \cdot N_{pop} \cdot (|M_G| \cdot |L_G| \cdot (|E_G| + |V_G|) + N_{pop}))$. □

Our experimental results indicate that this algorithm is around 160 times faster than the Brute-and-Force while its solutions are close to the optimal.

**Improved particle swarm algorithm:** The PSO algorithm is based on a global search and relies on the cooperation between particles to find the optimal solution. Thus, in the process of solving the problem, it often converges to the local optimal solution and fails to find the global optimal solution [26].

Genetic algorithms (GA) can improve PSO using natural genetic mechanisms. The basic idea is to simulate the genetic and evolutionary processes of organisms in nature to solve optimization problems. Unlike the PSO algorithm, it uses genetic evolution mechanisms such as selection, exchange, and mutation to find the global optimal solution, and uses a strategy of keeping the best individuals in the search process, thus reducing the number of cases where the solution falls in a local optimum. We can combine their advantages by using the global search capability of the particle swarm algorithm and the local search capability of the genetic algorithm.

**Algorithm 2** The PSO algorithm and its improvement

**Input:** Network $G = (M_G, L_G, V_G, E_G)$
**Output:** global_best_position, global_best_fitness

```
1:  function PSO(G)
2:      initialise_particles()
3:      while not_stopping_criteria() do
4:          particles ← ITERATION(particles)
5:      return global_best_position, global_best_fitness
6:  function PSOGA(G)
7:      initialise_particles()
8:      while not_stopping_criteria() do
9:          particles ← ITERATION(particles)
10:         GENETIC_OPERATORS(particles)
11:     return global_best_position, global_best_fitness
12: function ITERATION(particles)
13:     for particle ∈ particles do
14:         particle.fitness← Fitness(particle)
15:         if particle.fitness > particle.best then
16:             update individual fitness
17:         if particle.fitness > global_best_fitness then
18:             update global fitness
19:     for particle ∈ particles do
20:         particle.update_velocity()
21:         particle.update_position()
22:     return particles
23: function GENETIC_OPERATORS(particles)
24:     parents ← select_parents(particles)
25:     offspring ← crossover(parents)
26:     offspring← mutate(offspring)
27:     return offspring + particles
```

*Theorem* 5: This algorithm has the same complexity as PSO.

*Proof.* In the algorithm 2, line 10 runs the genetic algorithm with the crossover, mutate operations, which time complexity is $O(N_{pop})$, so the time complexity of this algorithm is $O(N_{iters} \cdot N_{pop} \cdot (|M_G| \cdot |L_G| \cdot (|E_G| + |V_G|) + N_{pop}))$. □

**Heuristic algorithm for adding links:** The strategies of the above algorithms rely on the function to calculate the fitness for the link $e_{(u_a,v_a)}$. However, this process is time-consuming. Thus, if the network is very large (e.g., 10,000 nodes), none of the above algorithms will converge in an acceptable time.

To cope with this situation, we propose a heuristic algorithm for adding links. The main idea of the algorithm is to find the network nodes that are critical for forwarding events sent by measurement points, and add a number of links to these nodes. The workflow of the algorithm is as follows:

(1) **Network pruning**: Prune the nodes that are not related to the forwarding of measurement events and obtain a simplified network $G_{prun}$.

(2) **Node sorting**: Sort the nodes by the number of edges connected to them and obtain the nodes that are critical for routing in the network $G_{prun}$.

(3) **Adding links**: For each measurement point, check whether the assigned result satisfies the constraints. If not, add a link to connect the measurement point to one of the nodes that gets from step 2.

**Algorithm 3** Heuristic for adding links

**Input:** Network $G = (M_G, L_G, V_G, E_G)$
**Output:** Sets $\{N_h\}$

**function** GET_HUB_NODE($G$)
  $N_p, E_p \leftarrow$ NETWORK_PRUNING($G$)
  **for** $N \in N_p$ **do**
    $N.degree \leftarrow 0$
    **if** $N \in M_G$ **then**
      continue
    **for** $E \in E_p$ **do**
      **if** $N \in E$ **then**
        $N.degree \leftarrow N.degree + 1$
  $N_p \leftarrow$ sorted$(N_p, key = N.degree)$
  $N_h \leftarrow N_p[: len(M_G)]$
  **return** $N_h$
**function** NETWORK_PRUNING($G$)
  $N_p \leftarrow \emptyset$, $E_p \leftarrow \emptyset$
  **for** $p \in P(M_G, L_G)$ **do**
    **for** $N \in p$ **do**
      $N_p.append(N)$
    **for** $E \in p$ **do**
      $E_p.append(E)$
  **return** $N_p, E_p$

*Theorem* 6: Algorithm 5 has the worst-case time complexity of $O\left(|M_G| \cdot |L_G| \cdot (|P_{max}(m,l)| + V_G + E_G) + |N_P|^2\right)$.

*Proof.* In the network pruning, to find $N_p$ and $E_p$, it must traverse all the paths of each pair of a measurement point and a monitoring server. The time complexity of this operation is $O(|M_G| \cdot |L_G| \cdot (|P_{max}(m,l)| + V_G + E_G))$. The time complexity of node sorting depends on the number of nodes $N_p$, it is equal to $O(|N_p|^2)$. So the time complexity of the algorithm is $O\left(|M_G| \cdot |L_G| \cdot (|P_{max}(m,l)| + V_G + E_G) + |N_P|^2\right)$. □

Our experimental results indicate that for a network with 10,000 nodes and 500,000 edges, the algorithm can find a solution to add links in an acceptable time (5-10s). This verifies the performance and feasibility of the algorithm.

## IV. IMPLEMENTATION

We have implemented a prototype of RFT in Python. There are three components, including the frontend module, the algorithm module and the backend module.

**Frontend.** The frontend module is used to collect statistics about the network and build the network model defined in II.A from these statistics, and feed this model to the algorithm module. Statistics on the switches in the network are obtained by direct calls to the RYU controller's API. These statistics include the current workload $w_v(v)$ for each switch and the per-switch packet processing delay $t_v(v)$ for each switch. Through network management tools, such as Simple Network Management Protocol (SNMP), can be used to obtain statistics on the links in the network, including the current workload per link $w_e(e)$ and the transmission latency per link $t_e(e)$. Reliability $\rho_e(e)$ is obtained through historical network maintenance data. In addition, it allows the network administrator to set user-defined parameters such as time and reliability thresholds for transferring events.

**Algorithm.** Using the network model constructed by the frontend module as input to the algorithm, the model generates the following five sets of decision variables (1) the set $\{x_l\}$ denotes whether server $l$ is selected, (2) the set $\{y_l^m\}$ denotes whether a measurement point $m$ sends events to monitoring server $l$, (3) the set $\{\alpha_m(p)\}$ denotes the amount of load transferring the events sent by $m$ assigned to path $p$, and (4) the set $\{\beta_m(l)\}$ denotes the amount of load processing the events sent by $m$ assigned to server $l$. (5) the set $\{e_{(u_a,v_a)}\}$ denotes the links added to the network.

**Backend.** We deploy RFT on a Tofino-based 64×100 Gbps hardware switch based on the decision variables output by the algorithm module. It activates the selected servers specified by the set $\{x_l\}$. For each measurement point $m$, it locates the target servers according to the set $\{y_l^m\}$ and transfers the events sent by $m$ to these servers via the paths identified by the set $\{\alpha_m(p)\}$. It also ensures that the load assigned to any path $p$ is less than $\{\alpha_m(p)\}$ and the load assigned to any server $l$ is less than $\{\beta_m(l)\}$. Finally, we add links in the data plane according to $\{e_{(u_a,v_a)}\}$.

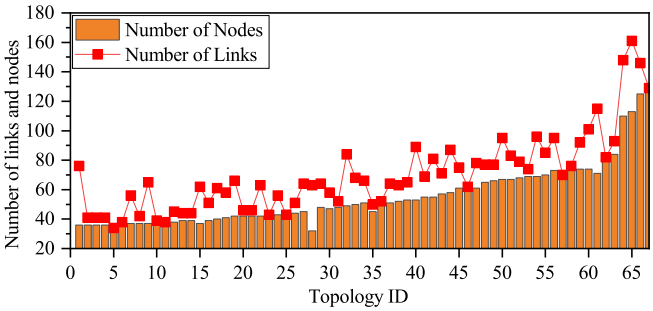
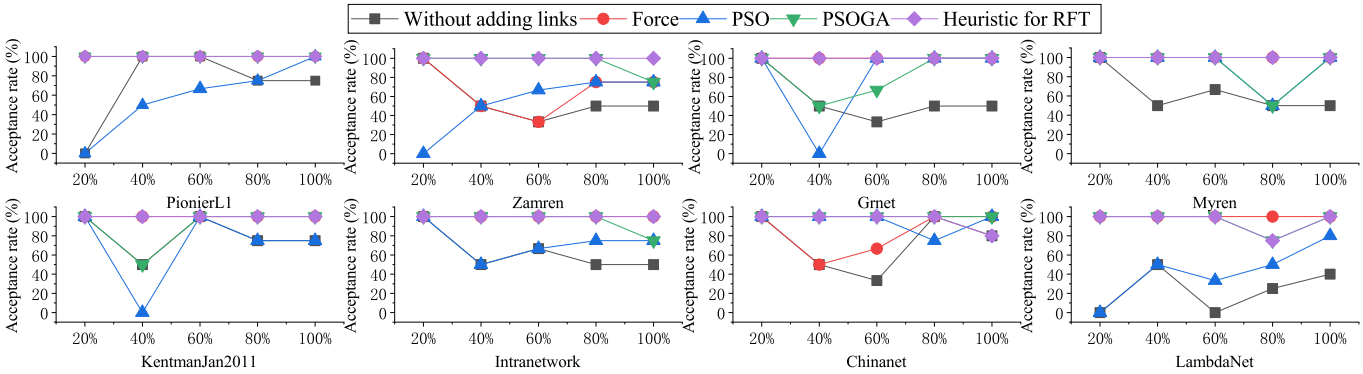


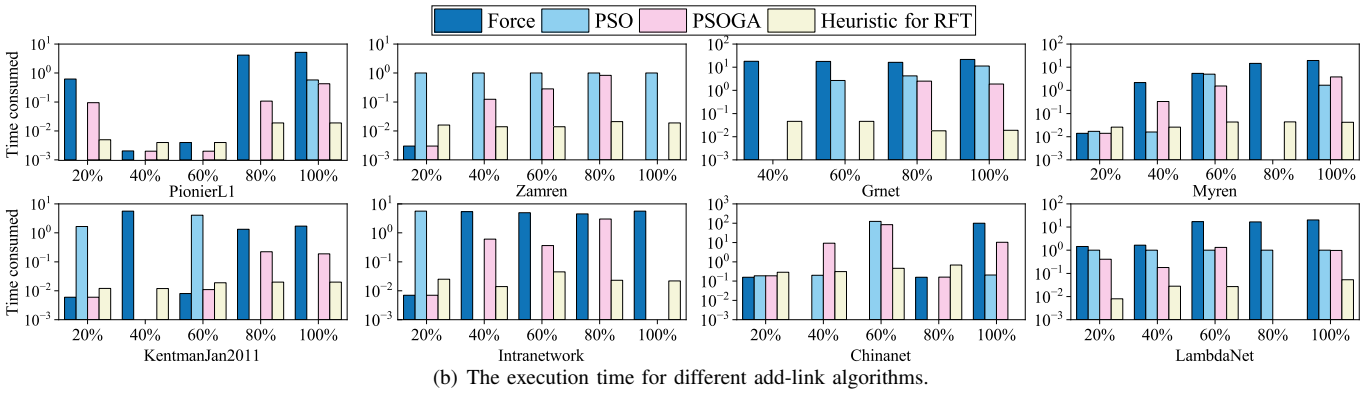
Fig. 1: Number of nodes and links in topologies collected in topology zoo.

## V. EVALUATION

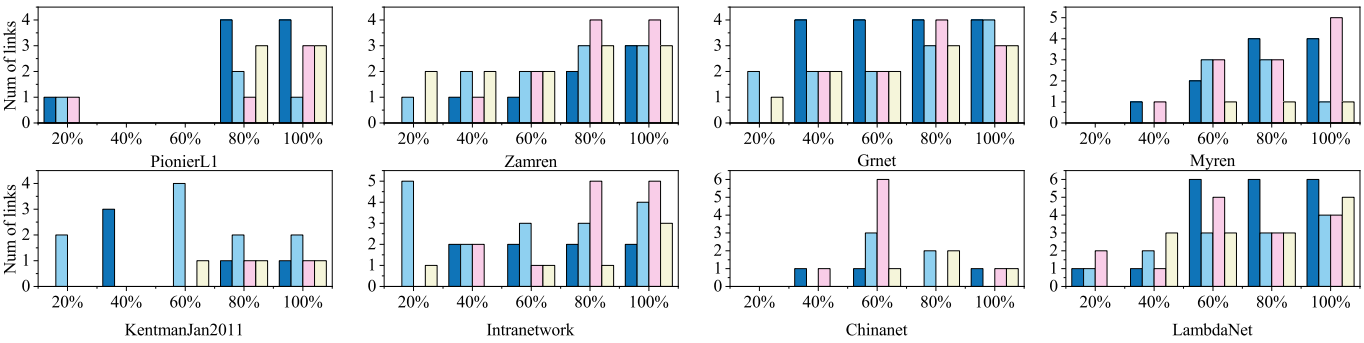
First, we conduct microbenchmarks to compare the heuristic of RFT with other algorithms, we observe that the heuristic performs better in terms of acceptance rate, time consumed and number of links added. Thus, we use the heuristic to add links in RFT. Second, we conduct macrobenchmarks to compare RFT with existing solutions. We observe that in the 67 test topos collected in topology zoo [27], the acceptance rate was 100% for RFT, 90% for both MTP and ILP, and 49% for Marple, and we have marked in the figure the test cases where no feasible solution could be found using other solutions. In




(a) The acceptance rate using different link-adding algorithms.




(b) The execution time for different add-link algorithms.




(c) Number of links added by different algorithms.

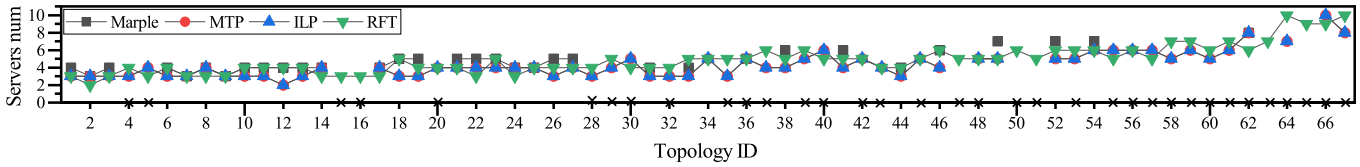
Fig. 2: The result of Microbenchmarks. The x-axis represents the percentage of OD-pairs that are required to satisfy the reliability.

terms of time consumption, the average run time was 0.32s for RFT and 0.007s, 3.20s, and 3.09s for Marple, MTP, and ILP, respectively. In addition, RFT preserves the low latency and reliability requirements in all tests.

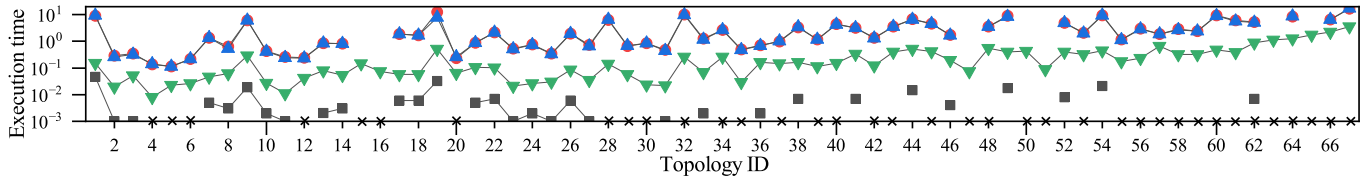
### A. Experimental Setup

**Simulate.** We simulate 67 topologies in the Internet topology zoo (No.1-67). By default, we randomly select 10% and 20% of the nodes as measurement points and candidate monitoring servers, respectively, and other nodes are data plane switches. Each data plane switch $v$ has a bandwidth $b_v(v)$ of 100 Gbps and a transmission latency $t_v(v)$ of 1 µs [8]. For each measurement point $m$, we set its rate $R(m)$ of sending events to be uniformly distributed between 10 Gbps and 40 Gbps [28], [29]. Each point only runs a single task. We set its deadline $\Phi_T(m)$ randomly between 5 ms and 25 ms while its reliability requirement $\Phi_R(m)$ is uniformly distributed between 0.01 and 0.5 [8]. The latency $t_e(e)$ of each link $e$ is randomly distributed between 1 ms and 10 ms. The bandwidth capacity $b_e(e)$ of a link is set to 100 Gbps, and the workload of a link or a data plane switch is uniformly distributed between 10 Gbps and 90 Gbps [29]. In addition, the reliability of the links or switches in operation is randomly distributed between 0.990 and 0.999, and the processing capacity $b_s(l)$ of each server is set to 40 Gbps while the initial workload $w_s(l)$ is set to 0 [8], [28], [29]. Figure. 1 summarises the size of these networks.

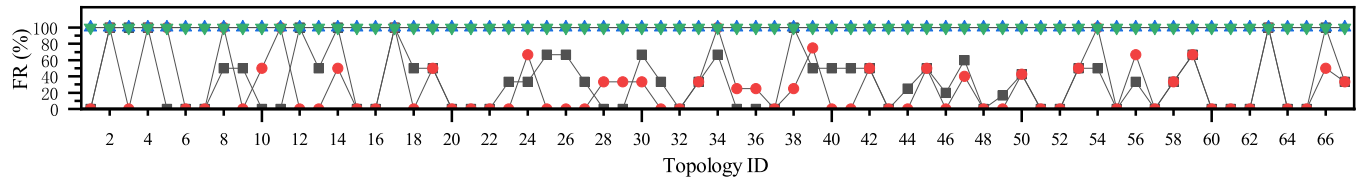
**Comparison solutions.** We compare RFT with two-pronged solutions. The first class consists of other algorithms of adding links, including (1) Brute-and-Force that enumerates all possible solutions to find the added links. (2) The population optimization based algorithms include PSO, PSOGA to optimize the search process and find near-optimal solutions through an evolutionary strategy. The second class consists of three representative solutions, including a measurement system Marple [11] that is implemented by using functional constructs like



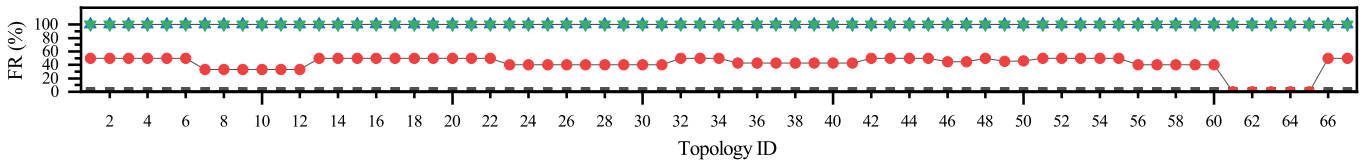

(a) Number of servers occupied by RFT. The **x** means the test cases which no feasible solution could be found by the comparison solutions.




(b) Execution time for RFT. The **x** means the test cases which no feasible solution could be found by the comparison solutions.




(c) Fulfillment of latency requirements.




(d) Fulfillment of reliability requirements.

Fig. 3: The result of Macrobenchmarks.

map, filter and zip, a placement framework MTP [8], and an ILP-based solution that optimally solves the problem. We assume that for each topo, the measurement points and the monitoring servers have been specified. We implement these solutions in Python and integrate them into the control plane.

### *B. Microbenchmarks*

**(Exp1) Acceptance rates.** We define the acceptance rate as the proportion of the origination-destination pairs (OD-pairs) that satisfy the constraints to the total number of the OD-pairs. In our experiments we evaluated the acceptance rate using different link-adding algorithms by specifying from 20%, 40% to 100% of the OD-pairs required to satisfy the reliability, respectively. As shown in Figure. 2(a), the heuristic algorithm performs better in every test sample.

**(Exp2) Execution time.** Using the same setup as Exp1, we evaluated the execution time of using different link-adding algorithms while specifying different proportions of OD pairs to satisfy the reliability. As shown in Figure. 2(b), for any given topo, the heuristic algorithm can complete the load assignment in less time while still satisfying the reliability requirements. For example, for the topo PionierL1 (38 nodes, 45 edges), the heuristic takes only 0.005 seconds when 20% of the OD pairs are specified to satisfy the reliability. For the topo Intranetwork (39 nodes, 51 edges), the heuristic takes only 0.02 seconds, several orders of magnitude less than the Brute-and-Force, while the PSO and PSOGA algorithms are unable to find a feasible solution in the given time.

**(Exp3) Overhead.** Using the same setup as Exp1, we evaluated the number of links required by different link-adding algorithms for specifying different proportions of OD pairs to satisfy reliability. The lower the number of links added, the lower the cost of operation. As shown in Figure. 2(c), the heuristic is close to the optimal solution for each of the topologies.

### *C. Macrobenchmarks*

**(Exp4) Number of servers occupied by RFT.** From experiments 1 to 3, we find that the heuristic has the lowest running cost and the fastest running time among these algorithms, so we use the heuristic as the algorithm for adding links in RFT. We quantify the cost by calculating the number of monitoring servers occupied. Comparing Marple [11], MTP [8] and ILP, as shown in Figure. 3(a), RFT's solutions are close to optimal, in some topologies such as Globenet (topo 50), Missouri (topo 51) and Deltacom (topo 65), the available network resources do not allow the distribution of measurement events, therefore these compared solutions could not find a feasible solution, while RFT found it.

**(Exp5) Execution time for RFT.** Using the same settings as exp4, we measured the execution time of RFT. As shown in Figure. 3(b), the execution time of RFT is lower than all of the comparison solutions. For the largest topology Pern (127 nodes, 129 edges), the execution time of RFT is 3.541 seconds,

while MTP and ILP execute in 16.879 and 17.058 seconds, and Marple is unable to find a feasible solution.

**(Exp6) Fulfillment of latency requirements.** Using the same settings as Exp4, we evaluate whether RFT can preserve the latency deadline. We measure the fulfillment rate (FR) as the ratio of the number of tasks that meet the requirements to the total number of tasks. We use RFT and compare solutions to make decisions, respectively. We count the number of tasks for which deadlines are preserved and calculate the fulfillment rate for each solution. Figure. 3(c) shows that RFT preserves all deadlines, while the other solutions fail to meet most deadlines. This is because RFT ranks the paths using transfer latency as a metric, ensuring that their collection can be completed by the deadline.

**(Exp7) Fulfillment of reliability requirements.** Using the same settings as Exp4, we evaluated whether RFT could meet the reliability requirements of the measurement task. As shown in Figure. 3(d), RFT can successfully limit the probability of event loss in all cases because it selects paths with reliability above a threshold, in contrast to other solutions whose event collection cannot meet most reliability requirements because they do not account for network failures.

## VI. CONCLUSION

We present RFT, a framework for achieving low-latency and reliable event transmission in network measurements. RFT strictly enforces user-specified reliability and latency requirements at the cost of adding a small number of links to the network topology. We implemented RFT on a Tofino-based hardware switch and evaluated RFT through extensive testbed experiments, demonstrating that RFT guarantees low latency and high reliability while ensuring high application-level accuracy. In the future work we will evaluate the performance of the proposed approaches in more tasks.

## VII. ACKNOWLEDGMENT

This work was supported in part by the National Key R&D Program of China under Grant 2022YFB2901305, in part by the Natural Science Foundation of China under Grant 61902362, in part by the Provincial Key R&D Program of Zhejiang under Grant 2020C01038, in part by the Fundamental Research Funds for the Central Universities (Zhejiang University NGICS Platform), and in part by the Major Science and Technology Infrastructure Project of Zhejiang Lab (Large-scale experimental device for information security of new generation industrial control system).

## REFERENCES

[1] C. Guo, L. Yuan, D. Xiang *et al.*, "Pingmesh: A large-scale system for data center network latency measurement and analysis," in *ACM SIGCOMM*, 2015, pp. 139–152.

[2] M. Yu, L. Jose, and R. Miao, "Software defined traffic measurement with opensketch." in *USENIX NSDI*, vol. 13, 2013, pp. 29–42.

[3] C.-W. Chang, G. Huang, B. Lin, and C.-N. Chuah, "Leisure: Load-balanced network-wide traffic measurement and monitor placement," *IEEE Transactions on Parallel and Distributed Systems*, vol. 26, no. 4, pp. 1059–1070, 2013.

[4] L. Jose, L. Yan *et al.*, "Compiling packet programs to reconfigurable switches," in *USENIX NSDI*, 2015, pp. 103–115.

[5] A. Sapio, I. Abdelaziz, A. Aldilaijan, M. Canini, and P. Kalnis, "In-network computation is a dumb idea whose time has come," in *ACM HotNets*, 2017, pp. 150–156.

[6] T. He, A. Gkelias, L. Ma *et al.*, "Robust and efficient monitor placement for network tomography in dynamic networks," *IEEE/ACM TON*, vol. 25, no. 3, pp. 1732–1745, 2017.

[7] C. Caol, W. Gong, W. Dong *et al.*, "Network measurement in multihop wireless networks with lossy and correlated links," in *IEEE INFOCOM*. IEEE, 2018, pp. 1340–1348.

[8] X. Chen, Q. Huang, P. Wang *et al.*, "Mtp: Avoiding control plane overload with measurement task placement," in *IEEE INFOCOM*. IEEE, 2021, pp. 1–10.

[9] X. Yu, H. Xu, D. Yao *et al.*, "Countmax: A lightweight and cooperative sketch measurement for software-defined networks," *IEEE/ACM TON*, vol. 26, no. 6, pp. 2774–2786, 2018.

[10] E. Song, T. Pan, C. Jia, W. Cao, J. Zhang, T. Huang, and Y. Liu, "Int-label: Lightweight in-band network-wide telemetry via interval-based distributed labelling," in *IEEE INFOCOM 2021-IEEE Conference on Computer Communications*. IEEE, 2021, pp. 1–10.

[11] S. Narayana, A. Sivaraman, V. Nathan *et al.*, "Language-directed hardware design for network performance monitoring," in *ACM SIGCOMM*, 2017, pp. 85–98.

[12] K. Yang, Y. Li, Z. Liu *et al.*, "Sketchint: Empowering int with towersketch for per-flow per-switch measurement," in *IEEE ICNP*. IEEE, 2021, pp. 1–12.

[13] Y. Zhou, C. Sun, H. H. Liu *et al.*, "Flow event telemetry on programmable data plane," in *ACM SIGCOMM*, 2020, pp. 76–89.

[14] J. Sonchack, O. Michel, A. J. Aviv *et al.*, "Scaling hardware accelerated network monitoring to concurrent and dynamic queries with* flow," in *USENIX ATC*, 2018, pp. 823–835.

[15] T. Pan, N. Yu, C. Jia *et al.*, "Sailfish: Accelerating cloud-scale multi-tenant multi-service gateways with programmable switches," in *ACM SIGCOMM*, 2021, pp. 194–206.

[16] X. Chen, Q. Huang, P. Wang *et al.*, "Lightnf: Simplifying network function offloading in programmable networks," in *IEEE/ACM IWQOS*. IEEE, 2021, pp. 1–10.

[17] X. Pu, L. Liu, Y. Mei *et al.*, "Understanding performance interference of i/o workload in virtualized cloud environments," in *IEEE International Conference on Cloud Computing*. IEEE, 2010, pp. 51–58.

[18] L. Liao and V. C. Leung, "Lldp based link latency monitoring in software defined networks," in *IEEE CNSM*. IEEE, 2016, pp. 330–335.

[19] Y. Li, Z.-P. Cai, and H. Xu, "Llmp: exploiting lldp for latency measurement in software-defined data center networks," *SCIENCE PRESS/Springer JCST*, vol. 33, pp. 277–285, 2018.

[20] P. Gill, N. Jain, and N. Nagappan, "Understanding network failures in data centers: measurement, analysis, and implications," in *ACM SIGCOMM*, 2011, pp. 350–361.

[21] A. Okabe, H. Yomono, and M. Kitamura, "Statistical analysis of the distribution of points on a network," *Geographical Analysis*, vol. 27, no. 2, pp. 152–175, 1995.

[22] L. Fang, X. Zhang, K. Sood *et al.*, "Reliability-aware virtual network function placement in carrier networks," *Journal of Network and Computer Applications*, vol. 154, p. 102536, 2020.

[23] M. Mitzenmacher and E. Upfal, *Probability and computing: Randomization and probabilistic techniques in algorithms and data analysis*. Cambridge university press, 2017.

[24] E. C. man Jr, M. Garey, and D. Johnson, "Approximation algorithms for bin packing: A survey," *Approximation algorithms for NP-hard problems*, pp. 46–93, 1996.

[25] F. Marini and B. Walczak, "Particle swarm optimization (pso). a tutorial," *Chemometrics and Intelligent Laboratory Systems*, vol. 149, pp. 153–165, 2015.

[26] K. Premalatha and A. Natarajan, "Hybrid pso and ga for global maximization," *Int. J. Open Problems Compt. Math*, vol. 2, no. 4, pp. 597–608, 2009.

[27] S. Knight, H. X. Nguyen, N. Falkner, R. Bowden, and M. Roughan, "The internet topology zoo," *IEEE Journal on Selected Areas in Communications*, vol. 29, no. 9, pp. 1765–1775, 2011.

[28] H. Liu, X. Chen, Q. Huang *et al.*, "Escala: Timely elastic scaling of control channels in network measurement," in *IEEE INFOCOM*. IEEE, 2022, pp. 1848–1857.

[29] X. Chen, H. Liu, Q. Huang *et al.*, "Speed: Resource-efficient and high-performance deployment for data plane programs," in *IEEE ICNP*. IEEE, 2020, pp. 1–12.