On Routing Optimization in Networks with Embedded Computational Services

Lifan Mei*, Jinrui Gou†, Jingrui Yang‡, Yujin Cai§, and Yong Liu *Fellow, IEEE*¶
Department of Electrical and Computer Engineering, New York University
Email: *lifan@nyu.edu, †jr6226@nyu.edu, ‡jy2823@nyu.edu, §yc4090@nyu.edu, ¶yongliu@nyu.edu

***Abstract*—Modern communication networks are increasingly equipped with in-network computational capabilities and services. Routing in such networks is significantly more complicated than the traditional routing. A legitimate route for a flow not only needs to have enough communication and computation resources, but also has to conform to various application-specific routing constraints. This paper presents a comprehensive study on routing optimization in networks with embedded computational services. We develop a set of routing optimization models and derive low-complexity heuristic routing algorithms for diverse computation scenarios. For the dynamic demands, we also develop an online routing algorithm with performance guarantees. Through evaluations over emerging applications on real topologies, we demonstrate that our models can be flexibly customized to meet the diverse routing requirements of different computation applications. Our proposed heuristic algorithms significantly outperform baseline algorithms and can achieve close-to-optimal performance in various scenarios.**

***Index Terms*—Routing, Edge Computing, In-Network Computation, Network Function Virtualization**

I. Introduction

Modern communication networks are increasingly equipped with in-network computational capabilities and services. Software-Defined Networking (SDN) [1] [2] technology can decouple data plane and control plane, and the routing decision can be made in a centralized fashion rather than hop-by-hop, utilizing more information for better routing decision. Traffic flows traversing such networks are processed by different types of middle-boxes in-flight. For example, in a 5G [3] core network, traffic from/to mobile user devices must pass through special network elements, including eNodeB, serving gateways, and packet data network gateways. To improve security and/or boost application performance, an application flow may also traverse other types of middle-boxes for application-specific processing, e.g., intrusion detection and prevention, content caching to reduce latency and network traffic, rendering of VR/AR objects to offload user devices, object detection from video/lidar camera data acquired by autonomous vehicles, etc. In a cloud-native network, to improve performance and resilience, middle-boxes are replicated throughout the network, and can be elastically provisioned on commodity servers through Network Function Virtualization (NFV) [4] [5].

Routing is a critical component of networking. The main goal of the traditional network routing is to forward user traffic to their destinations with the lowest possible delay, while maintaining the network-wide load balance and resilience. Routing in networks with embedded computational services is significantly more complicated. It has to find a path for each flow that simultaneously has sufficient bandwidth and computation resources to meet the flow's traffic and computation demands. Load balance and resilience have to be maintained on both communication links and computing nodes. To further complicate matters, application-specific computational services will introduce diverse additional routing requirements. Some application flows have to traverse multiple types of middle-boxes in certain preset orders, and the routing path may have to contain cycles. The traffic volume of a flow might increase or decrease after processing, consequently, the flow conservation law no longer holds. Some applications require the computation to be done on a single computation node, while some applications can split their computation load to multiple paths and multiple nodes to achieve the parallelization gain. How traffic and computation are split directly impacts the load balance and resilience of the whole network. The existing routing models cannot directly address these new challenges and requirements. *The goal of our study is to comprehensively explore the design space of routing in networks with embedded computational services. We develop a set of routing optimization models and derive low-complexity heuristic routing algorithms for diverse computation scenarios.* Towards this goal, we made the following contributions.

1) For routing with non-splittable flows, we show that the problem is NP-Hard, and develop a loop-friendly mixed integer program (MIP) model to characterize the interplay between traffic routing, computation load distribution, and network delay performance. We further design a Metric-TSP type of heuristic algorithm to achieve close-to-optimal performance.
2) When flows can be arbitrarily split, we prove the equivalence between the routing with computational services problem and the regular routing problem using the *segment routing* idea. We develop a Linear Program (LP) routing optimization model by extending the classic Multi-Commodity-Flow (MCF) model to work with heterogeneous middle-boxes and traffic scaling resulting from the processing. The LP model can be further extended to study the joint optimization of traffic routing and computation resource provisioning.
3) For come-and-go dynamic traffic demands, we convert the online routing problem into a flow packing problem, and develop a primal-dual type of online routing

algorithm with performance guarantees.

4) We evaluate the developed models and algorithms using emerging computation applications over real network topologies. Through extensive experiments, we demonstrate that our models can be flexibly customized to meet the diverse routing requirements of different computation applications. Our algorithms significantly outperform baseline routing algorithms, and can achieve close-to-optimal performance in various scenarios.

II. Related Work

For routing with in-network processing, there are existing studies to address various application scenarios with different assumptions. In the context of edge/fog computing, in [6], besides the computation allocation, they also considered mobility and privacy in the joint optimization problem. In [7], authors focused on the service allocation problem for AR offloading. Some papers, [8] [9] [10], focused on approximation algorithm. The authors of [8] [9] used randomized rounding with linear programming relaxation as the key idea to deal with the non-splittable flows. Authors of [10] developed a fully polynomial-time approximation scheme (FPTAS) for an NP-Hard problem in IoT scenarios. In [11] [12], the problem is studied without hard link capacity constraints. For the middle-box traversal order, [13] focused on the case where the traversal order is fixed. In the [14], authors used graph layering to conform to the order of traversal. Authors of [15] studied the case with arbitrary traversal order. For routing with cycles, one strategy is to calculate all paths with a certain number of cycles in advance [15] [16] [17], and then use the path-based routing formulation to find the optimal traffic routing and demand allocation. The number of candidate paths increases exponentially and the pre-calculated paths may miss some good paths. Among all these studies, the one that is closest to ours is [15]. The main assumption of their work is that flows are infinitely splittable. In real world applications, it is equally important to study non-splittable and finitely splittable flows. For infinitely splittable flows, their model assumes universal middle-boxes while our model can handle heterogeneous middle-boxes. Our segment routing based formulation also makes it easy to study traffic scaling after processing and joint routing and provisioning. Table I summarizes the main differences between our work and the most related studies.

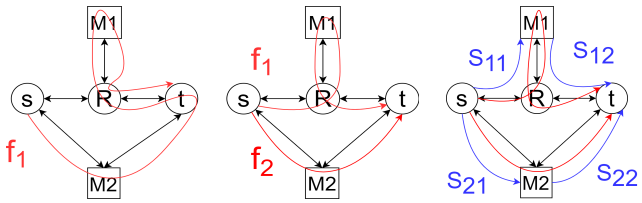
TABLE I
Key Differences from Most Related Works

Traffic Flow	**Non-Splittable**			**Infinitely-Splittable**			**Scaling**
Middle-box Univ./Hetero.	**Univ.**	**Hetero.**		**Univ.**	**Hetero.**		
Traversal Order **F**ixed or **N**ot	n/a	**F**	**N**	n/a	**F**	**N**	
[15]				✔			
[14] [13]	✔	✔					
Our Paper	✔	✔	✔	✔	✔		✔

III. Routing with In-network Processing Problem

We consider a generic network represented by a directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ denotes a set of nodes, and $\mathcal{E}$ denotes a set of directed links connecting the nodes. The graph $\mathcal{G}$ is assumed to be mesh-connected, having multiple paths connecting each pair of nodes. Every link e is associated with bandwidth capacity of C_e. The node set $\mathcal{V}$ contains standard routers and special middle-boxes attached to routers. There might be different types of middle-boxes. For a type-r middle-box z, its processing capacity is N_z^r. A set of flows are to be routed and processed in the network. Each flow d is characterized by its source node s_d, destination node t_d, traffic volume h_d, and its demand for type-r processing W_d^r. The **Routing with In-Network Processing (RINP)** problem is to find paths with sufficient bandwidth and processing resources for all flows, subject to bandwidth/resource capacities on all links/nodes. There are different variations of RINP along different dimensions.

1) *Universal vs. Heterogeneous middle-boxes:* In the traditional networks, different types of middle-boxes are designed for specific processing tasks. The new trend is to implement middle-box functions on generic computing servers using NFV. Each computing node can be configured to process any demand. The capacity of a middle-box can be measured by its universal computation power N^r, and the flow processing demand can be characterized by the total computation power needed.
2) *Non-splittable vs. Infinitesimal Flow/demand:* When the flow granularity is small, e.g., one flow for each user application session, splitting the flow to multiple network paths and multiple middle-boxes will be inefficient/impractical. On the other hand, in a backbone network, each flow is indeed the aggregated user traffic from city A to city B. It is therefore more flexible to split the traffic as well as the associated processing demand to multiple paths, and if a path contains multiple middle-boxes, the processing demand can be further split to utilize all available resources.
3) *Constant vs. Elastic Traffic Volume:* Certain types of in-network processing will increase or decrease the traffic volume of the processed flow. For example, after an edge server rendered online game updates, the size of the rendered video stream will typically be larger than the game updates. On the other hand, after an edge server processed the Lidar data captured by an autonomous vehicle, it only needs to upload the learned representations to the cloud server, which has a volume much lower than the raw data.
4) *Fixed vs. Reconfigurable Processing Capacity:* The traditional middle-boxes have fixed capacities, while the software based virtual middle-boxes can be reconfigured on-demand to match the processing needs. The RINP problem can be more efficiently solved by jointly routing flows and provisioning middle-boxes.



(a) Non-splittable (b) Infinitely-splittable (c) Segment Routing

Fig. 1. Routing with In-Network Processing (RINP) Basic Examples

TABLE II
KEY PARAMETERS AND NOTATIONS

Symbol	Description
V	set of all nodes
$P^r \subset V$	subset of computing nodes with type-r resources
E	set of links in a graph
D	set of demand flows
T	time horizon
a_{ev}	1 if link e originates from node v; 0 otherwise
b_{ev}	1 if link e terminates at node v; 0 otherwise
s_d	source node of flow d
t_d	terminal node of flow d
h_d	traffic volume of flow d
h_d^z	traffic volume of virtual sub-flow of d processed by z
W_d^r	type-r resource demand of flow d
C_e	bandwidth capacity on link e
f_e	total traffic rate on link e
x_{ed}	traffic of flow d allocated on link e
x_{ed}^{zs}	traffic of segment s of d's subflow processed by z allocated on link e
u_{ed}	integer, number of times flow d traverses link e
ε_{ed}	binary, whether or not flow d traverses link e
N_z^r	type-r resource capacity on node $z \in P^r$
ρ_r	upper bound of type-r resource utilization
N_z	resource capacity on node z with universal resources
w_{zd}^r	type-r resource consumption of flow d on node z
y_{ed}^r	unprocessed type-r demand of flow d on link e
$\tau_d, \tau_d^s, \tau_d^f$	duration, start time, and finish time of d
$\beta_{d\tau}$	binary, $= 1$ if $\tau_d^s \le \tau \le \tau_d^f$, $= 0$ otherwise
P_d	set of candidate paths for demand d
δ_{edp}	non-negative constant, the fraction of traffic of candidate path p routed on link e
y_{dp}	binary variable, whether demand d is routed on candidate path $p \in P_d$
$D(e)$	variable, set of demands routed on link e
$P(e)$	variable, set of selected paths ($y_{dp} = 1$) passing through link e

IV. RINP Optimization Models

The optimization objective of RINP depends on the actual situation. Some popular objectives are to minimize the link/node delay, minimize the maximum link/node utilization, maximize the processed flows, and certain combinations of them. In this paper, we use minimizing network delay as an example objective for static optimization and maximizing the processed flows for dynamic optimization. The developed formulations can be easily customized for other objectives.

A. Non-splittable Flow

We start with the simple case that each flow can only take a single path, i.e., non-splittable. An example is shown in Fig. 1(a), there are only two middle-boxes, each with a capacity of 1, to complete the processing demand of 2, the flow has to take a path with cycles to pass through the two middle-boxes before reaching its destination. Similarly, if the two middle-boxes are of different types, and the flow has to be processed by both types, the flow again has to take a path with cycles. In general, we will first show that the non-splittable RINP problem is NP-Hard by reducing the well-known Metric-TSP to a simple version of non-splittable RINP. We then develop a Mixed-Integer Program (MIP) to study the interplay between traffic routing, demand processing and network delay optimization.

Theorem 1: Non-splittable RINP with constant link delays is NP-Hard. □

Proof: Given a set of nodes and the distances between them, the Traveling Salesman Problem (TSP) is to find an optimal order of traversing all the nodes with the shortest total distance. Metric-TSP is a special case of TSP where the distances between nodes form a metric to satisfy the triangle inequality: $d(v_1, v_2) \le d(v_1, v_3) + d(v_3, v_2)$. For a given graph $G = (V, E)$ and the distance metric $d(\cdot)$, we construct an special instance of non-splittable RINP on G by: 1) placing one unit of computing capacity on each node, 2) setting the propagation delay on link (v_1, v_2) to $d(v_1, v_2)$, 3) creating one flow with the same source and destination node, 4) setting the flow's traffic volume to $\epsilon << C_e$ so that the congestion delay is negligible, and setting the flow's processing demand to $|V|$. Obviously, to satisfy the flow's processing demand, the flow has to visit all nodes in the graph, and to minimize the total delay RINP has to find the path with the shortest distance. The only potential discrepancy between RINP solution and the Metric-TSP solution is that Metric-TSP solution can only visit each node once while in principle RINP solution may have to visit a node multiple times, as illustrated in the example in Fig. 1(a). However, due to the metric distance, we can easily show that the RINP solution for the specially constructed network can be guaranteed to be cycle-free. Suppose the RINP solution visits some nodes more than once, without loss of generality, let k be the first node that is visited twice, i and j are the nodes visited before the after the second visit to k. By removing the second visit of k, i.e., replacing the path segment $i \to k \to j$ with $i \to j$, the total path length can potentially be reduced due to the triangle inequality. Using this process, we can remove all the duplicate visits to get a cycle-free RINP path that has either the same length or a shorter length than the original path. This path is a cycle-free solution for Metric-TSP. ■

For more generic non-splittable RINP, we develop a MIP model to analytically investigate how traffic routing and demand splitting impact network-wide performance. The goal of the modeling is to obtain understandings to develop heuristic single-path RINP algorithms.

$$\textbf{MIP-RINP:} \quad \min_{\{u_{ed}, y^r_{ed}, w^r_{zd}\}} \quad \sum_{e \in E} \frac{f_e}{C_e - f_e} \tag{1a}$$

subject to

$$\sum_{d=1}^{|D|} x_{ed} = f_e \leq C_e, \quad \sum_{d=1}^{|D|} w^r_{zd} \leq \rho_r N^r_z, \tag{1b}$$

$$\sum_{e \in E} a_{ev} u_{ed} - \sum_{e \in E} b_{ev} u_{ed} = \begin{cases} 1 & if\ v = s_d \\ 0 & if\ v \neq s_d, t_d \\ -1 & if\ v = t_d \end{cases} \tag{1c}$$

$$x_{ed} = u_{ed} h_d \tag{1d}$$

$$\varepsilon_{ed} \leq u_{ed}, \quad u_{ed} \leq B\varepsilon_{ed}, \quad y^r_{ed} \leq B\varepsilon_{ed} \tag{1e}$$

$$\sum_{e \in E} a_{ev} y^r_{ed} - \sum_{e \in E} b_{ev} y^r_{ed} = \begin{cases} W^r_d & if\ v = s_d \\ w^r_{vd} & if\ v \neq s_d, t_d \\ 0 & if\ v = t_d \end{cases} \tag{1f}$$

$$u_{ed} \geq 0 \text{ integer}; \quad \varepsilon_{ed} \text{ binary}; \tag{1g}$$

$$w^r_{vd} \geq 0, w^r_{vd} = 0, \forall v \notin P_r, \quad y^r_{ed} \geq 0, \tag{1h}$$

where (1a) is the total network delay modeled using the M/M/1 formula. (1b) describes the total traffic rate on a link can not exceed its bandwidth capacity, and the total type-r resources consumed on a node can not be larger than its type-r resource capacity discounted by the target utilization ρ_r. (1c) is the flow conservation for single non-splittable path. As illustrated in Fig. 1(a), non-splittable flow may have to take a path with cycles. To allow a flow to traverse a link/node multiple times, routing variable u_{ed} is configured to be non-negative integer, instead of binary. On a relay node, the total number of times its outgoing links are traversed should equal to the total number of times its incoming links are traversed. The difference between the two numbers should be 1 for the source node, and -1 for the destination. (1d) calculates the total traffic carried by link e for demand d. Since we only have a single-path for d, whenever the path traverses link e, the total demand volume of h_d will be added to the link. So the total traffic carried by a link is proportional to the number of times the link is traversed by the single path. Meanwhile, the resources on a node can be accessed by a flow only if the flow passes the node through links attached to it. Therefore, we want to know whether or not flow d traverses link e, which is indicated by a binary variable ε_{ed} in (1e). ε_{ed} is forced to take value 1 if $u_{ed} > 0$, and value 0 if $u_{ed} = 0$ (B is a large positive constant). Finally, (1f) is the conservation law for resources: all the resource demands exit from the source node through its outgoing links, the unsatisfied demand at the destination is zero, and on an intermediate node, the difference between the incoming unprocessed resource demand and the outgoing unprocessed resource demand is the amount of demand w^r_{vd} processed on that node. Similar to x_{ed} in (1d), if the path visits a link multiple times, y^r_{ed} is the sum of the unprocessed demand from all the visits. (1g) are the traffic routing variables, and (1h) are the resource demand splitting variables. The two sets of variables are coupled through (1e). By replacing the convex delay function in (1a) with a piece-wise linear function, the optimization problem becomes a mixed-integer program.

If *the processing demand is also non-splittable*, we can replace w^r_{vd} by $W^r_d b^r_{vd}$, where k^r_{vd} is a binary variable indicating whether type-r processing of d is done on node v. Then y^r_{ed} and constraint (1f) can be taken out from the formulation. The added new constraints are:

$$\sum_v k^r_{vd} = 1, \quad k^r_{vd} \leq \sum_{e \in E} a_{ev} \varepsilon_{ed},$$

where the first equation requires the processing will be done on exactly one node, and the second inequality dictates a node's processing resources can be used if and only if it is visited by the flow.

B. Infinitely Splittable Flow

Non-splittable flow is too limiting for large flows. The traditional Traffic Engineering (TE) of backbone networks assumes arbitrary traffic splitting and develops Multi-Commodity Flow (MCF) based convex/linear programming to optimize network design objectives. The obtained optimal routes are typically cycle-free. For RINP, if a flow can be arbitrarily split to multiple paths, the traffic and processing demand splitting become more flexible. For the example in Fig. 1(b), with traffic splitting, the flow can utilize two paths, one for traversing each middle-box. While the bottom path is cycle-free, the upper path still contains a cycle, i.e., not a simple path. Even with infinitely splittable flow, RINP still cannot be directly solved using the MCF model.

We address this challenge using the *segment routing* idea as shown in Fig. 1(c) . We start with universal middle-box. Flow splitting leads to both traffic and processing demand splitting among multiple paths. We assume that the processing demand allocated on a path is proportional to the traffic volume allocated on that path. If there are multiple middle-boxes on a path, the processing demand allocated on that path can be further arbitrarily split among these middle-boxes. In general, a legitimate RINP path can "stop" at multiple middle-boxes to get processing done before reaching the final destination. We define an *n-stop* RINP path as a path on which n middle-boxes process the flow. Note, since a middle-box can simply forward traffic without processing it, an *n-stop* path may traverse more than n middle-boxes. The single RINP path in Fig. 1(a) is *2-stop*, while the two RINP paths in Fig. 1(b) are both *1-stop*.

Theorem 2: Any *n-stop* RINP path can be decomposed to n *1-stop* RINP paths. □

Proof: Let $s_d \rightsquigarrow z_1, \cdots, \rightsquigarrow z_n \rightsquigarrow, t_d$ be any *n-stop* RINP path. The traffic flow on this path is f and the total processing demand is w, and the processing demand allocated to middle-box z_i is w_i, and $\sum_{i=1}^n w_i = w$. For any z_i, we can generate a *1-stop* RINP path that follows exactly the same route as the

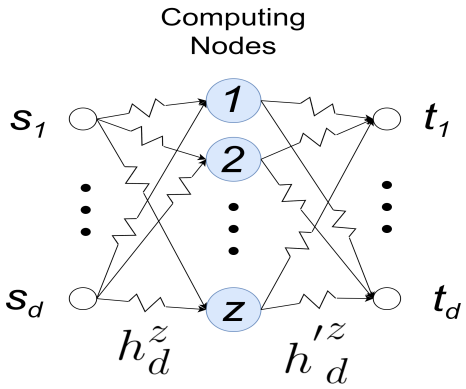


Fig. 2. With universal middle-boxes, any RINP flow can be implemented by two sets of regular traffic paths

n-stop RINP path, but only stops at z_i and gets w_i amount of processing done, and all the other middle-boxes only forward the traffic. To follow the proportional demand splitting rule, the traffic allocated on this path is fw_i/w. It is easy to check that the n *1-stop* RINP paths carry the total traffic of f to t_d, and all the processing demand of w are processed. ■

Theorem 3: Traffic routing subproblem of RINP can be optimally solved by an equivalent Multi-Commodity-Flow routing problem for pure traffic flows. □

Proof: We prove this by construction. Let $\mathcal{R}_d$ be a set of legitimate RINP routes for flow d with total traffic volume h_d and total processing demand W_d. And the total processing demand allocated to middle-box z_i is w_i. According to Theorem 2, all RINP paths in $\mathcal{R}_d$ can be decomposed to *1-stop* RINP paths. Any *1-stop* RINP path stopping at z_i can be decomposed into two segments, $s_d \rightsquigarrow z_i$ and $z_i \rightsquigarrow t_d$. The first segments of all the *1-stop* RINP paths stopping at z_i share the same source s_d and destination z_i, and the total traffic flow must be $h_d w_i/W_d$ (due to proportional demand splitting). Similarly, the second segments of all the *1-stop* RINP paths stopping at z_i share the same source z_i and destination t_d, and the total traffic flow is $h_d w_i/W_d$. In other words, the bandwidth consumed by $\mathcal{R}_d$ on all links can be used to carry traffic for $2|P|$ pure traffic subflows $\{s_d \rightarrow z_i, z_i \in P\}$ and $\{z_i \rightarrow t_d, z_i \in P\}$, both with traffic demand $h_d w_i/W_d$. Meanwhile, it is obvious that any MCF routing solution for the $2|P|$ traffic subflows, can be used to carry traffic for $\mathcal{R}_d$ to enable processing w_i on z_i. ■

As illustrated in Fig. 2, based on Theorem 3, we can replace each RINP flow by two sets of traffic subflows, one set consists of subflows from the source to all middle-boxes, the other set consists of subflows from all middle-boxes to the destination. The traffic volumes of the subflows are set to be proportional with the processing demand split among the middle-boxes. The following is an MCF-type convex/linear program for infinitely splittable RINP.

$$\textbf{SR-Infinite:} \quad \min_{\{h_d^z, x_{ed}^{z1}, x_{ed}^{z2}\}} \quad \sum_{e\in E} \frac{f_e}{C_e - f_e} \tag{2a}$$

subject to

$$\sum_{z\in P} h_d^z = h_d, \quad \sum_{d\in D} \frac{h_d^z}{h_d} W_d \le \rho N_z \tag{2b}$$

$$\sum_{d\in D}\sum_{z\in P} (x_{ed}^{z1} + x_{ed}^{z2}) \le C_e \tag{2c}$$

$$\sum_{e\in E} a_{ev} x_{ed}^{z1} - \sum_{e\in E} b_{ev} x_{ed}^{z1} = \begin{cases} h_d^z & if\ v = s_d \\ -h_d^z & if\ v = z \\ 0 & otherwise \end{cases} \tag{2d}$$

$$\sum_{e\in E} a_{ev} x_{ed}^{z2} - \sum_{e\in E} b_{ev} x_{ed}^{z2} = \begin{cases} h_d^z & if\ v = z \\ -h_d^z & if\ v = t_d \\ 0 & otherwise \end{cases} \tag{2e}$$

$$x_{ed}^{z1} \ge 0, x_{ed}^{z2} \ge 0, h_d^z \ge 0, \tag{2f}$$

where (2b) is the allocation of processing demand/traffic among all compute nodes. Here we assume the resource demand splitting is proportionally to the traffic splitting. (2c) dictates that the traffic of the first and second segments share link capacity, (2d) is the flow conservation for the first segment demand from s_d to z, while (2e) is the flow conservation for the second segment demand from z to t_d. Both segments have identical volume of h_d^z. If the traffic volume increases/decreases after being processed, we only need to change h'^z_d to $\phi_d h_d^z$ in (2e), where ϕ_d is the ratio of the traffic volume after processing over the original volume. After being processed by compute node z, all traffic going to the same destination can be aggregated and routed together. The total volume of the aggregated demand from z to v is $\sum_{d:t_d=v} h_d^z$. To further reduce the number of routing variables and routing constraints in the formulation, we can replace flow-based routing variables with *destination-based routing variables*. Let η_{ev} be the amount of post-processing traffic (regardless of the processing node) destined to node v (regardless of the source), the aggregated second-segment routing constraints on any node $v' \in V$ can be rewritten as:

$$\sum_{e\in E} a_{ev'}\eta_{ev} - \sum_{e\in E} b_{ev'}\eta_{ev} = \begin{cases} \sum_{d:t_d=v} h_d^{v'} & if\ v' \in P \\ \sum_{d:t_d=v} h_d & if\ v' = v \\ 0 & otherwise, \end{cases} \tag{3}$$

where the left-hand side is the difference between the outgoing traffic destined to v and incoming traffic destined to v, the right-hand side means if v' is one of the compute node, the difference is simply the total post-processing traffic from v' to v, if v' is the destination itself, then the difference is all post-processing traffic to v, for other relay nodes, the difference should be zero. The number of routing variables is reduced from $|V|^2|P||E|$ in (2e) to $|V||E|$ here. The number of routing constraints is also reduced from $|V|^3|P|$ to $|V|^2$. Both are reduced by a factor of $|V||P|$. By replacing (2e) with (3), and replacing (2c) with

$$\sum_{d\in D}\sum_{z\in P} x_{ed}^{z1} + \sum_{v\in V} \eta_{ev} \le C_e,$$

we have a more compact optimization problem.

C. Joint Routing and Resource Provisioning Problem

The routing optimization models can be easily extended to study the joint optimization of traffic routing and middle-box provisioning by making the middle-box capacity N_z^r in (1b) and N_z in (2b) variables under some total resource budget constraint. Flexible resource provisioning can make the routing job easier, and can play an important role in network failure recovery. We will demonstrate this through case studies in Section VII-C.

D. Heterogeneous Middle-boxes

When middle-boxes are heterogeneous, each flow can have multiple types of processing demands.

1) Non-splittable Flow: The MIP optimization model in Sec. IV-A has already considered different types of demands. In the optimal solution, the traversal order of different types of middle-boxes can be arbitrary. This may be not acceptable for certain application scenarios. For example, in cellular core networks, there are predefined orders for middle-box traversal, e.g., mobile traffic has to first go through a firewall before being routed to a load balancer, and there are multiple instances of each type of middle-box. RINP with predefined middle-box traversal order was studied in [14] using a graph layering approach.

2) Infinitely Splittable Flow: When the traversal order of heterogeneous middle-boxes is predefined, the LP formulation for homogeneous middle-box can be extended to study heterogeneous middle-boxes. If there are k types of middle-boxes, let the middle-box index i represent its order of traversal. Extending the 2-segment routing idea, each RINP path now consists of $k+1$ segments, $s_d \rightsquigarrow z^{(1)} \cdots \rightsquigarrow z^{(k)} \rightsquigarrow t_d$, where $z^{(i)}$ is some type-i middle-box. Similar to Theorem 3, each RINP demand can be replaced by $k+1$ sets of pure traffic subflows, one set for each segment. The number of subflows at segment k is $|P_{k-1}||P_k|$. An example of two-types of middle-boxes is illustrated in Fig. 3. MCF linear/convex program model can be established in a similar fashion to (2a). The flow conservation holds from the two segments in Fig. 2 to the three segments in Fig. 3. The first segment is from the source nodes to the Type I computing nodes; the second segment is from the Type I nodes to the Type II nodes; the final segment is from the Type II nodes to the destination. On each segment, the numbers of allocation variables are equal to the product of starting point numbers and endpoint numbers. The model can still handles the flow size change in the two segments when flow entering and exiting the Type I and Type II computing nodes, and the change ration can be different for two types of computing.

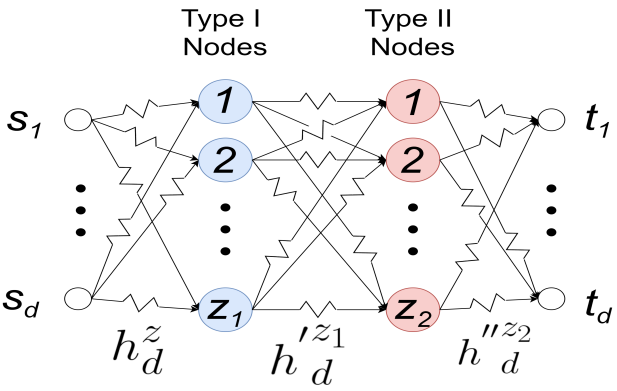


Fig. 3. With two types of middle-boxes, any RINP flow can be implemented by three sets of regular traffic paths

V. Fast Heuristic Routing Algorithms

A. Heuristic for Non-splittable Flow

Even though the non-splittable version of RINP is NP-Hard, we can still leverage on the solution of the infinitely splittable RINP to develop a close-to-optimal single-path solution. More specifically, we relax the non-splittable requirement and get the optimal infinitely splittable solution by solving the LP defined in (2a). While the paths cannot be used, the processing demand allocation, namely $\{\frac{h_d^z}{h_d} W_d\}$ can be used for the single-path solution. For all computing nodes with non-zero processing demand allocation, we then find with a *close-to-shortest* path to traverse them using an approximation algorithm solving the Metric-TSP problem, as illustrated in Algorithm 1:

1. We sort all flows by their demand volumes;
2. We solve the LP of segment routing with the infinitely splittable flow, denoted by SR in line 3, to get a set of computing nodes $\mathcal{Z}_d$ to be used by each flow d, and the demand allocation on each computing node $\{\frac{h_d^z}{h_d} W_d, z \in \mathcal{Z}_d\}$;
3. Process all flows in the decreasing order of volume. For each flow d, find the path with enough capacity by removing links with a capacity less than its volume h_d (line 6). Generate an overlay graph G^z consisting of the source, destination and all the used computing nodes for demand d (line 7). The distance between two overlay nodes is the current shortest path in the underlay network (considering the congestion delay) (line 8).
4. Find the shortest path in the overlay graph G^z to traverse all computing nodes $\mathcal{Z}_d$ using M-TSP approximation algorithm. (line 9). We use Christofides algorithm [18] here.
5. Map the overlay path back to the underlay network (line 10), take the bandwidth consumed by flow d out of the underlay network (line 11), and process the next flow.
6. Evaluate the network delay and return.

B. Finitely Splittable Flow

Real situations may lie between the infinitely splittable flow and the non-splittable flow. While non-splittable is too rigid, splitting flow to too many paths will introduce much overheads for implementation, such as too many flow entries in the routing table. Finitely splittable means that the maximum number k of paths each flow can be split to is controlled. One way to get a *k-split* solution is to evenly split traffic and computation demands of a flow equally and get k subflows, each of which can be treated as an independent flow and obtain the RINP solution using the MIP formulation in (1a). We call this scheme **MIP-K**. However, such an approach has to work with a set of $k|D|$ subflows. It is impractical to solve MIP for any reasonably large network.

Algorithm 1: HEURISTIC: SR-TSP

Input : Underlying Network $G = (V, E)$, Set of Computation Flows D
$D = Sort(D, h_d)$
$\{\mathcal{Z}_d, \forall d \in D\} \Leftarrow SR(G, D)$
for d in D **do**
  $G' = G$
  $G'.remove(\{e | e.remain\_capacity \leq h_d\})$
  $G^z.V = \mathcal{Z}_d \bigcup \{s_d, t_d\}$
  $d^z(v, v') = Shortest\_Path(G', v, v')$, $\forall v, v' \in G^z.V$
  $\mathcal{P}^z(s_d, t_d) =$ M-TSP(G^z, s_d, t_d)
  $\mathcal{P}(s_d, t_d) = Recon(G, \mathcal{P}^z(s_d, t_d))$
  $G.Update(\mathcal{P}(s_d, t_d), h_d, \mathcal{Z}_d)$
end for
return $\sum_{e \in E} e.delay$

Algorithm 2: SR-ITERATION FOR K-SPLIT

Input : $G = (V, E), D$
Equally split each flow in D to k subflows, get a new demand set D^k
$D^k = Sort(D^k, h_d)$
for d in D **do**
  $\mathcal{Z}_d = \{v \in G.V | v.remain\_comp > W_d\}$
  $BPC = +\infty$
  for z in $\mathcal{Z}_d$: **do**
    if $BPC > SPC(s_d, z) + SPC(z, t_d)$ **then**
      $BPC = SPC(s_d, z) + SPC(z, t_d)$
      $BP = SP(s_d, z) \bigcup SP(z, t_d)$
    end if
  end for
  $G.update(Path, h_d, W_d, BP)$
end for
Merge subflows of the same flow $d \in D$
return $\sum_{e \in E} e.delay$

1) Heuristic: To address this scalability issue, we came up with a fast heuristic algorithm. Motivated by that Segment Routing can lead to the optimal solution for the infinitely splittable flow case, our heuristic algorithm also follows the segment routing framework. We first equally divide each flow into k subflows, then iteratively find the shortest two-segment path for each subflow, and update the computation and bandwidth resource capacities. If two subflows of the same original flow share the same path, they will be merged them back into a larger subflow. The pseudo-code is in Algorithm 2. For each demand list, we sort the subflows in descending order of their volumes. In each iteration, find all the available computing nodes, $\mathcal{Z}_d$, whose available computation resource is more than the current computation demand. For every 2-segment path through $z \in \mathcal{Z}_d$, find the shortest (smallest delay) path from s_d to some computing node z, and then to t_d. In the pseudo-code, BP and BPC denote the best path, and the best path cost. SP and SPC are the functions to computing the shortest path and shortest path cost in the current graph (considering the link congestion delay).

VI. ONLINE ROUTING ALGORITHM WITH PERFORMANCE GUARANTEE

The previous formulations assume the application demands are known, and can be used for routing optimization in long-term traffic engineering. In practice, application flows come and go with a finite duration. Whenever a new application flow joins the network, online routing algorithm has to find a feasible routing path for it without knowing the future application flows.

A. Online Routing Optimization Models

Similar to other online routing studies, e.g. [14], [19], we want to accept as many flows as possible over time. The objective of online routing optimization is to maximize the total accepted flows, while complying with the capacity constraints on computation nodes and communication links.

Node Capacity Constraint Conversion: To simplify the problem, we convert computation node capacity constraints into virtual link capacity constraints. More specifically, each computing node is split into two virtual routing nodes: the inbound node connects to the computing node's inbound links, and the outbound node connects to the computing node's outbound links. A virtual link connects is introduced from the inbound node to the outbound node. The bandwidth capacity of the virtual link is set as the computing node capacity divided by the resource demand of per unit traffic. Consequently, if the traffic flow traversing the virtual link is bounded by the link capacity, the resource demand of the flow is bounded by the computing node capacity. In Fig. 4, the purple nodes are routers with computing resources capacity of 100 and 200 respectively. Each computing node is split into two copies in the extended virtual graph. Solid lines and dotted lines mark physical and virtual links, respectively. The demand is from $Source$ to $Terminal$, and its traffic volume is 8 units, and needs 160 units of computing resources. The resource demand of each traffic unit is $160/8 = 20$. The capacity of virtual links introduced for the two computing nodes are $100/20 = 5$ and $200/20 = 10$, respectively. After such conversion, the bottleneck of the upper path is 5, and only the lower path is feasible for the demand.

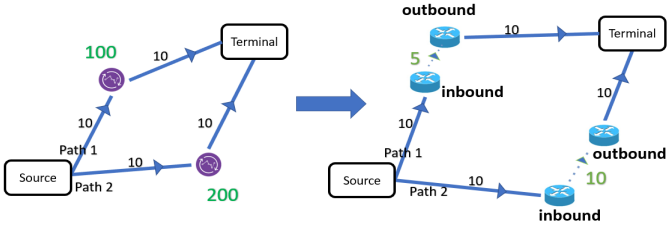


Fig. 4. Model Computing Node Capacity using Virtual Link

Feasible Path Generation: After the conversion at the previous step, not all the paths complying with the link capacity constraints are feasible. For example, if a flow does not traverse any virtual link, its computing resource demand will not be satisfied. To address this issue, we introduce two

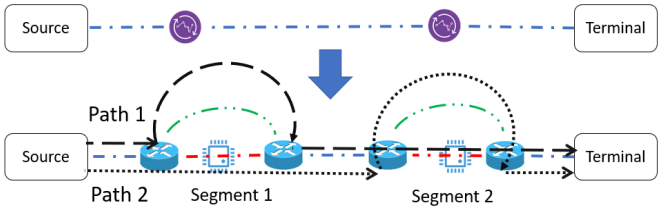


Fig. 5. Generating Candidate Path with Computing Resources

virtual links from the inbound copy to the outbound copy of each computing nodes: if a flow traverses the red link, it consumes the computing resources on the node; if a flow traverses the green link, it is only forwarded by the node without consuming any computing resources. The red virtual link capacity is set according to the computing node capacity, the green link capacity can be set to a large value so that it is never a bottleneck. By requiring each candidate path to contain exactly one red link, we can make sure that a flow will be processed correctly when routed on a candidate path. Fig. 5 shows an example about how to generate candidate paths with computing resources for a graph with two computing nodes. Two computing nodes are for two segments, and on each segment, the flow should follow one of two links. After removing the path with two red links and no red links, there are 2 valid paths. $Path\ 1$ goes green link for Segment 1 and red link for Segment 2, and $Path\ 2$ goes red link and green link for Segment 1 and 2, respectively.

After the previous conversion and candidate path generation, we now present the routing model to maximize the total accepted demands:

$$\max_{y_{dp}} \sum_{d} \tau_d h_d \sum_{p \in P_d} y_{dp} \tag{4a}$$

subject to

$$\sum_{p \in P_d} y_{dp} \leq 1, \quad \forall d \in D, \tag{4b}$$

$$\sum_{d} h_d \beta_{dt} \sum_{p \in P_d} \delta_{edp} y_{dp} \leq C_e, \quad \forall e \in E, \forall t \in T, \tag{4c}$$

where P_d is the set of candidate paths for flow d, and y_{dp} is a binary variable indicating whether d is routed on path p. Other notations are defined in Table II

By introducing Lagrange multipliers $(z_d, d \in D, x_{et}, e \in E, t \in T)$ for the constraints, the Lagrangian function $L(\{z_d\}, \{x_{et}\})$ is:

$$\sum_{d} \tau_d h_d \sum_{p \in P_d} y_{dp} - \sum_{d} z_d \left(\sum_{p \in P_d} y_{dp} - 1 \right)$$
$$- \sum_{e,t} x_{et} \left(\sum_{d} h_d \beta_{dt} \sum_{p \in P_d} \delta_{edp} y_{dp} - C_e \right),$$

The corresponding dual optimization problem becomes:

$$\min_{z_d, x_{et}} \sum_{d} z_d + \sum_{e,t} C_e x_{et} \tag{5a}$$

subject to

$$z_d + \sum_{e,t} x_{et} h_d \delta_{edp} \beta_{dt} \geq \tau_d h_d, \quad \forall d \in D, \forall p \in P_d \tag{5b}$$

Due to the strong duality theorem, any feasible solution of the dual problem is upper bound of the optimal solution of the routing problem.

B. Online Primal-Dual Algorithm

While the optimization problem can be solved offline, we need an online routing algorithm to route each flow as it arrives without knowing the future flows. It is expected that the total accepted flow volume by the online routing algorithm is lower than the offline optimal solution. Following the general framework of online approximation algorithm [19], we develop the following online routing algorithm with a guaranteed ratio over the offline optimal solution.

Algorithm 3: ONLINE ROUTING ALGORITHM

1: **Input** : $G = (V, E)$
2: **Initialize:** $x_{et} = 0, z_d = 0, y_{dp} = 0$
3: Whenever a new demand d is introduced,
4: **if** there is a path $p \in P_d$, so that
5: $\sum_{e,t} x_{et} \delta_{edp} \beta_{dt} < \tau_d$ **then**
6: Set $y_{dp} = 1$, $z_d = \tau_d h_d$
7: **for** each link e on path p **do**
8: $D(e) = D(e) + \{d\}$
9: $P(e) = P(e) + \{p\}$
10: $x_{et} = x_{et} \left(1 + \frac{h_d \delta_{edp}}{C_e}\right) + \frac{h_d \delta_{edp}}{C_e \Delta_{dp}}$,
11: $where\ \Delta_{dp} \triangleq \sum_{e \in E} \delta_{edp}$
12: **end for**
13: **else**
14: Reject the demand d
15: **end if**

For the Algorithm 3, when a new demand d is introduced, first check if there exists a path that satisfies the dual constraint. If so, that path is picked, and update the dual variable x_{et} and decision variables, $D(e)$ and $P(e)$, accordingly. If not, reject the demand d. Due to Line 5 in Algorithm 3, at any step of the update, before the update, $x_{et}(start) \leq \gamma_e \triangleq \max_{p \in P(e)} \frac{\tau_d}{\delta_{edp}}$. After the update, assuming $h_d \delta_{edp} < C_e$, we have

$$x_{et} \leq 2\gamma_e + 1. \tag{6}$$

C. Performance Guarantee

Theorem 4: The accepted demands by the online routing algorithm is $\geq \frac{1}{3}$ of the offline optimal solution. □

Proof: Obviously, the (z_d, x_{et}) generated by the algorithm is a feasible solution for the dual problem $(z_d = \tau_d h_d)$. After each demand is introduced, the increase in the routing

objective is simply $\tau_d h_d$, resulting from y_{dp} increases from 0 to 1; The increase in the dual objective is:

$$\tau_d h_d + \sum_{e,t} C_e \Delta x_{et} = 2\tau_d h_d + \sum_{e,t} x_{et} h_d \delta_{edp} \beta_{dt} \leq 3\tau_d h_d, \tag{7}$$

where the last inequality is due to Line 5 in Algorithm 3. After all iterations, the objective value of the dual solution is less than three times the objective of routing solution. Since dual feasible solution is the upper bound of the accepted demands of offline optimal routing solution, so the accepted demands of the online routing solution is no less than $\frac{1}{3}$ of the offline optimal routing solution. ■

Theorem 5: Upper bound for the link capacity violation of online routing algorithm is $W_e \log(\Delta_e(2\gamma_e + 1) + 1)$ □

Proof: It is clear that for any $h_d, C_e \geq 1$ (Taylor Expansion),

$$1 + \frac{h_d \delta_{edp}}{C_e} \leq \left(1 + \frac{1}{C_e}\right)^{h_d \delta_{edp}}.$$

We can choose W_e so that

$$1 + \frac{h_d \delta_{edp}}{C_e} \geq \left(1 + \frac{1}{W_e C_e}\right)^{h_d \delta_{edp}}, \forall p \in P(e). \tag{8}$$

In other words,

$$W_e \triangleq \max_{p \in P(e)} \frac{1}{C_e \left(\left(1 + \frac{h_d \delta_{edp}}{C_e}\right)^{\frac{1}{h_d \delta_{edp}}} - 1\right)} \tag{9}$$

$$= \frac{1}{C_e \left(\left(1 + \frac{\alpha_e}{C_e}\right)^{\frac{1}{\alpha_e}} - 1\right)}, \tag{10}$$

where $\alpha_e \triangleq \max_{p \in P(e)} h_d \delta_{edp}$, and the last equality holds due to function $(1 + \frac{x}{c})^{1/x}$ is a decreasing function of x. □

Proof: $(1 + \frac{x}{c})^{1/x}$ *decreases when* $x > 0$

$$\Leftarrow ((1 + \frac{x}{c})^{1/x})' < 0 \tag{11}$$

$$\Leftarrow \frac{(1 + \frac{x}{c})^{\frac{1}{x}}(x - (c + x)log(1 + \frac{x}{c}))}{x^2(c + x)} < 0 \tag{12}$$

$$\Leftarrow d(x) \triangleq x - (c + x)log(1 + \frac{x}{c}) < 0 \tag{13}$$

$$\Leftarrow d'(x) < 0 \; and \; d(0^+) < 0 \tag{14}$$

We can easily find that $d'(x) = -log(\frac{x}{c} + 1) < 0$ and $d(0) = 0$. ■

For the case, $h_d, C_e < 1$, (8) holds by setting $W_e = 1$ (Bernoulli's Inequality). Now define another sequence $x'_{e,t}$ and update it synchronously with $x_{e,t}$ as:

$$x'_{et} = x'_{et} \left(1 + \frac{1}{W_e C_e}\right)^{h_d \delta_{edp}} + \frac{1}{\Delta_e} \left(\left(1 + \frac{1}{W_e C_e}\right)^{h_d \delta_{edp}} - 1\right), \tag{15}$$

where $\Delta_e \triangleq \max_{p \in P(e)} \Delta_{dp}$. Comparing Line 10 in Algorithm 3 with (15), due to (8), both the multiplicative increase factor (the first term) and the additive increase (the second term) of x_{et} are larger than x'_{et} at any update step. So we can conclude $x_{et} > x'_{et}$.

Meanwhile, for each updating step, (15) can be transformed into:

$$x'_{et}(end) + \frac{1}{\Delta_e} = (x'_{et}(start) + \frac{1}{\Delta_e}) \left(1 + \frac{1}{W_e C_e}\right)^{h_d \delta_{edp}},$$

together with $x'_{et}(0) = 0$, at the end of the online algorithm,

$$x'_{et} = \frac{1}{\Delta_e} \left(\left(1 + \frac{1}{W_e C_e}\right)^{\sum_{p \in P(e)} h_d \delta_{edp}} - 1\right)$$

Then we have:

$$\sum_{p \in P(e)} h_d \delta_{edp} = \frac{\log(\Delta_e x'_{et} + 1)}{\log(1 + \frac{1}{C_e W_e})} \approx C_e W_e \log(\Delta_e x'_{et} + 1),$$

if $C_e >> 1$.

Finally, due to $x_{et} > x'_{et}$ and (6), we have

$$\sum_{p \in P(e)} h_d \delta_{edp} \leq C_e W_e \log(\Delta_e(2\gamma_e + 1) + 1), \tag{16}$$

In other words, the capacity violation factor on any link is bounded by $W_e \log(\Delta_e(2\gamma_e + 1) + 1)$. ■

To eliminate link capacity violation, we revise our online routing algorithm so that a path is picked only when the capacities of all links on the path will not be violated after the new flow is admitted. This change will potentially reduce the accepted traffic ratio. As will be shown in the experiment section, the loss of accepted traffic can be well justified by completely avoiding link capacity violation.

VII. Evaluation

We now evaluate the proposed models and algorithms using emerging applications over real network topologies.

A. Non-spilttable Flow – VR Rendering

In the VR scenario, one user communicates with another user through realtime 360 degree video streaming. Rendering of 360 video is computation-intensive, it is therefore beneficial to offload the computing from user device to edge computing nodes. The realtime video stream has to be processed as a whole, i.e. the flow is Non-Spilttable, follows a single path.

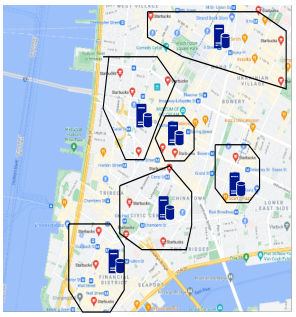
To generate a realistic VR scenario, we used the method in [7] to generate our data. For topology, we use the locations of Starbucks stores in the Lower Manhattan of New York City as the locations for users. Each store is connected to a close-by computing node, forming a star topology as shown on Fig. 6(a). The computing node is connected to two or three nearby computing nodes. There are 24 nodes and 56 links in the network. We synthesize four sets of VR flows between Starbucks customers.

To evaluate the SR-TSP heuristic, in Table III, we compare its solution with the optimal MIP solution on different datasets. The average performance loss of SR-TSP is about 8% from the

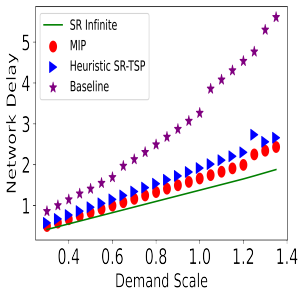
global optimal. In Fig. 6(b), we evaluate how the performance gap increases as traffic and computation demands scale up. We also develop a greedy baseline algorithm in which each source node finds the nearest computing node with sufficient computation capacity for processing, then the processed traffic is routed to the destination following the shortest path. SR-Infinite is the LP solution for infinitely splittable flow, which serves as a lower bound for the MIP solution. It can be seen that as the demands scale up, the network delay increases for all the algorithms. The gap between baseline and other algorithms is also getting larger and larger. SR-TSP uses the optimal infinitely splittable solution to get computation demand allocation, and then uses mature TSP Heuristic to solve the problem of compute node traversal. It achieves a good tradeoff between time complexity and routing performance.

TABLE III
EVALUATION ON NON-SPLITTABLE VR FLOWS

	Data 1	Data 2	Data 3	Data 4	Average
MIP-RINP	2.087	2.436	2.266	2.426	2.304
SR-TSP	2.214	2.589	2.615	2.622	2.510
Gap	5.77%	5.92%	13.36%	7.48%	8.23%



(a) Starbucks Locations



(b) Network delay for various Demand Scale

Fig. 6. Routing of Non-splittable VR Flows

B. Finitely Splittable – Smart City

In the case of a smart city, there are many cameras on the streets. The rapid processing of the information captured by the cameras provides a great help for the city's emergency response. For fire alarm, proactive and rapid fire detection can reduce casualties and property losses. As for the police department, it is critical to respond quickly to crimes. For example, on the Amber system, quickly extracting vehicle plates from street camera video can save lives of kidnapped children. Edge computing can support in-network video analytics for fast response. Unlike the non-splittable VR flows, video from one intersection is separable, because there are typically multiple camera videos from different viewing angles, even video from a single viewing angle can be divided and processed in parallel.

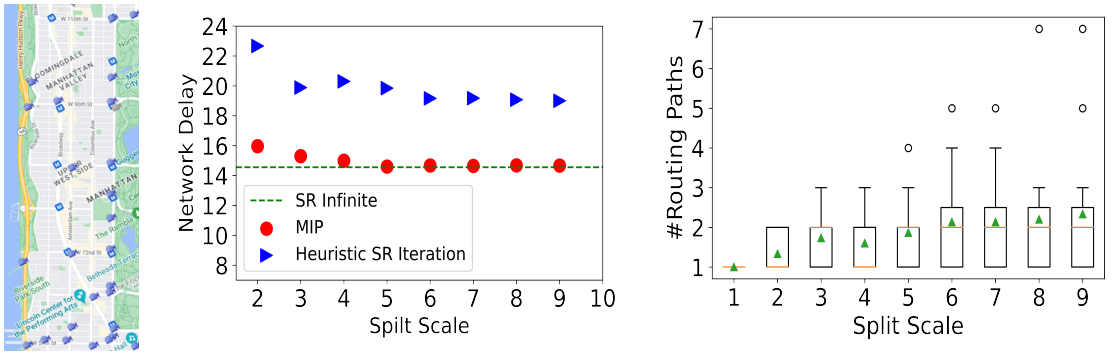
Real video camera data from New York City was used for evaluation. Fig. 7(a) shows the distribution of cameras from 115th to 59th Street in the Upper West Side of New York City. For the same camera, there are two video analytics tasks, one for the Fire department and one for the Police. We assume that the computing power required for fire detection is proportional to the number of nearby houses. Based on the data of Zillow, we counted the number of houses with a radius of half a block. Assuming that the computing power required by the police camera is related to the traffic flow of the street, we conducted the query of the precise location traffic flow and the reasoning of the fuzzy location traffic flow according to [20] [21] [22] to complete our data set. It contains 40 nodes and 15 cameras. A total of 30 streams flow from cameras to nearby fire and police stations.

We focus on the *k-split* routing. For each test, we randomly select 60% of demands for evaluation, the representative result shown in Fig. 7(b), as the split scale k increases, the performance gets better. For non-splitting, the performance is bad: the network delay of MIP and SR-Iteration are 245.87 and 247.79 respectively. But once the spilt degree goes to 2, the performance improved a lot! SR-Infinite is the optimal solution when the splitting can go to infinity. We can see that as the k goes to 4, the solution from MIP can already match the optimal. This finding is important, since MIP incurs high complexity. In our example, MIP with $k = 4$ uses 12.63s, but MIP with $k = 9$ uses 189s, while their solution quality is almost the same. Secondly, when the network and demands are large, an alternative method is use SR-Iteration to find a quick solution, even though it incurs 20% higher delay than MIP. The solution quality of SR-Iteration also improves a lot from $k = 1$ to $k = 9$, but saturates when k is large. In Fig. 7(c), for the same dataset, we draw the boxplot to show how split scale affects the number of paths taken by each flow. The green arrow is the average number of paths taken under k. We can see that most flows only take less than three paths even though they are allowed to take up to k paths. But some demands can still take advantage of large k. In the split scale $k = 9$, one demand is routed to seven different paths! *This suggests that we can assign different splitting degrees to different demands.*

C. Infinitely Splittable and Placement – WAN

A flow in Wide-Area-Networks (WAN) typically represents a set of user flows share the same ingress and egress points. It is often treated as infinitely splittable in traffic engineering. We evaluate RINP with infinitely splittable flow in the context of WAN.

Dataset. We obtain two WAN datasets. The first is obtained from [23]. It presents a publicly available dataset from GÉANT, the European Research and Educational Network. The GÉANT topology consists of 23 nodes with 74 directed links. The average node degree is 3.217. We randomly sample 12 node-to-node demands in the given traffic matrix. We set link capacity to 80000 for each link and pick three nodes not in the source node set and not in the destination set as compute nodes. The second dataset is obtained by [24]. We pick topology Abilene, which represents a high-performance backbone network across USA. It consists of 12 nodes with 30 directed links. The average node degree is 2.5. For the demand setting, we randomly sample 6 node-to-node demands in the traffic matrix. The link capacity is 40000 for each link

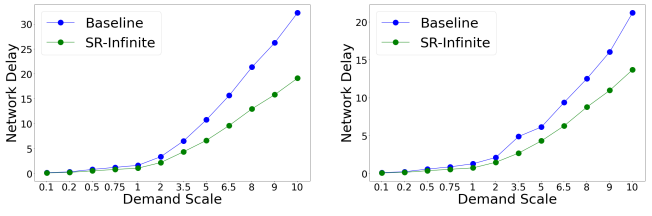


(a) Smart City Topology (b) Network Delay for Various Spilt Scales (c) Number of Paths per Flow

Fig. 7. K-split RINP Routing for Video Analytics Flows

as described in the dataset. We also pick two nodes SNVAng and IPLSng as compute nodes. Without loss of generality, for both datasets, we assume the computational resource demand for each flow equals its traffic volume.

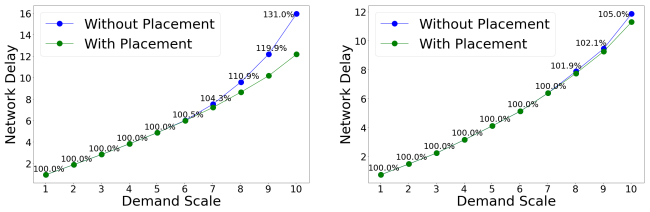
Greedy Computation Allocation vs. Joint Optimization. To demonstrate the importance of joint optimization of traffic routing and demand allocation, we develop a baseline algorithm that greedily allocates the flow with the largest computation load to the most powerful compute node, then conducts LP based routing optimization for the greedy computation load allocations. The result is shown in Fig. 8. For topology GÉANT, the network delay of baseline is always above the LP solution of segment routing. The average increment is about 57.58%. For topology Abilene, the network delay of baseline is also always above segment routing. The average increment is about 55.29%.



(a) GÉANT (b) Abilene

Fig. 8. Greedy Computation Allocation v.s. Joint Optimization

With Placement vs. Without Placement. We compare segment routing when the computational capacity is fixed for each computational node with when the computational capacity can be freely allocated among computational nodes as long as the sum of their computational capacity is the same as the fixed case. The result is shown in Fig. 9. The percentage represents the ratio of the network delay without placement over that with placement. For both topologies, the network delay is the same when the demand scale is not very large. The reason is that when demand scale is small, the original fixed computational capacity distribution is enough to handle all demands that follow the best routes. For topology GÉANT, with placement version outperforms the fixed version when the demand scale goes up to 6. When demand scale is 10, the network delay of the fixed version is about 131.0% of the

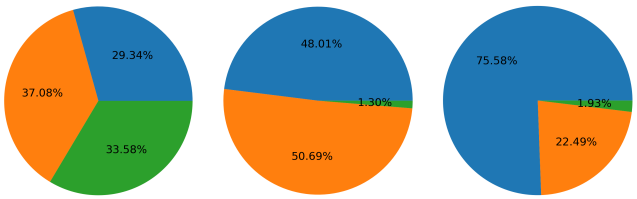


(a) GÉANT (b) Abilene

Fig. 9. Effect of Flexible Computation Capacity Placement

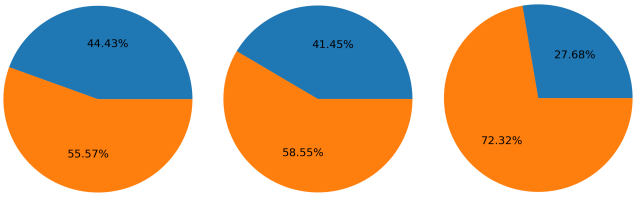
with placement version. For topology Abilene, with placement version outperforms the fixed version when the demand scale goes up to 8. When demand scale is 10, the network delay of the fixed version is about 105.0% of the network delay of the with placement version.

Adaptation to Link Failures. We run segment routing when computational capacity can be freely allocated among computational nodes under a given budget. For both topologies, we run it on three different configurations by disabling some links. The pie chart shows the ratio of the computational capacity allocated to each computation node. The result for GÉANT is shown in Fig. 10. Blue represents node 3, orange represents node 13, green represents node 16. Initially, the allocation of the computational capacity among node 3, 13, 16 is shown as Fig. 10(a). After cutting three bidirectional links connected to node 16, the allocation is shown as Fig. 10(b). The computational capacity allocated to node 16 drastically reduces. After cutting two more bidirectional links connected to node 13, the allocation is shown in Fig. 10(c). The computational capacity allocated to node 13 also drastically reduces and most computational capacity is allocated to node 3. The result for Abilene is shown in Fig. 11. Blue represents node IPLSng and orange represents node SNVAng. Initially, the allocation of the computational capacity among node IPLSng and node SNVAng is shown as Fig. 11(a). After cutting one bidirectional link connected to node IPLSng, the allocation is shown as Fig. 11(b). The computational capacity allocated to node IPLSng slightly reduces. After cutting one more bidirectional link connected to node IPLSng, the allocation is shown as Fig. 11(c). The computational capacity allocated to node IPLSng reduces further and most computational capacity



(a) Initial Topology (b) Cut Three Links (c) Cut Five Links

Fig. 10. Capacity Placement Change after Link Failures for GÉANT



(a) Initial Topology (b) Cut One Link (c) Cut Two Links

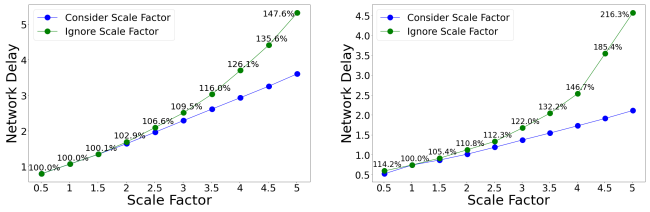
Fig. 11. Placement Change after Link Failures for Abilene

is allocated to node SNVAng.

Impact of Traffic Scaling after Processing. In this part, we run segment routing when the flow size will change after going through a computational node. We first take the scale factor into account in our model for different scale factors. We then compare it with the case when the traffic scaling is ignored when calculating the optimal segment routing. The result is shown in Fig. 12. The percentage represents the ratio of the network delay when ignoring traffic scaling to the network delay when considering traffic scaling. For topology GÉANT, the network delay is shown in Fig. 12(a). Except for scale factor 0.5 and 1, the network delay of ignoring scale factor is higher than that of considering scale factor. For topology Abilene, the network delay is shown in Fig. 12(b). Except scale factor 1, the network delay of ignoring scale factor is higher than that of considering scale factor.

D. Online Routing Algorithm with Guarantee

Dataset. To evaluate online routing algorithm, we synthesize dynamic traffic demands. The topology is the same as the Smart City problem in Section VII-B. In the graph of 40 nodes, 14 nodes were selected as computing nodes. Without loss of generality, the ratio between the computing demand and the flow size is $1:1$. Each link capacity is 550 units, and node capacity is 300 units. We pick 8 source-destination pairs. New flows arrive at the network according to Poisson process,



(a) GÉANT (b) Abilene

Fig. 12. Consider Traffic Scaling vs. Ignore Traffic Scaling

and are assigned to each source-destination pair in round-robin fashion. The time duration of each flow follows lognormal distribution, i.e., $\tau = e^{\mu+\sigma Z}$, where Z is the standard normal variable. When a flow is active, the traffic volume follows Gaussian distribution.

Performance. For online algorithm, we compare two versions, one with link violation, and one without violation. We also use the shortest path algorithm as the baseline, and the upper bound is obtained from the MIP model implemented by Gurobi. In Fig. 13, the flow arrival rate is two flows/minute, the lognormal duration parameters are $\mu = 0.974$ minutes, and $\sigma = 0.5$ minutes. The average flow volume is 85 units. To evaluate the impact of flow volume variations, we conducted two experiments with flow volume standard deviation of 10 and 20, respectively. Our online algorithms significantly outperform the shortest path routing. The accepted traffic of online algorithm without violation is only slightly lower than the MIP solution. With link violation allowed, the online algorithm can accept a little bit more traffic than MIP. Fig. 14 shows the total traffic of active flows over time. This suggests that our online algorithm can efficiently utilize available communication and computation resources in the network to support dynamic flows.

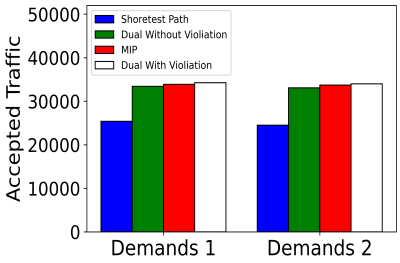


Fig. 13. Total Accepted Traffic of Dynamic Flows

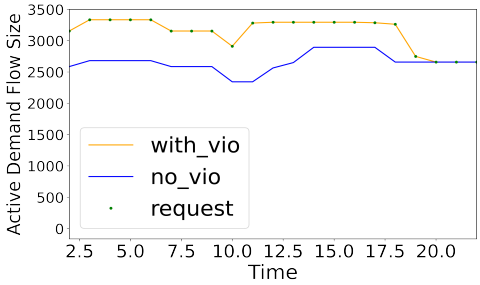


Fig. 14. Active Flow Size Over Time

Violation. In the online routing algorithm, the link violation theoretical upper bound has been given. But in practice, within a reasonable demand flow size range, the violation is rare. To check the actual violation, Fig. 15 plots the Median and Max link utilization over time. We also plot the boxplot distribution of link utilization in Fig. 16 and Fig. 17 for different demand parameter settings. For all the experiments, the flow arrival rate is still two flows/minute. There are three settings of flow time duration parameters (μ, σ): $(0.974, 0.5)$, $(0.598, 1.0)$, $(0.379, 1.2)$, respectively. Under those settings

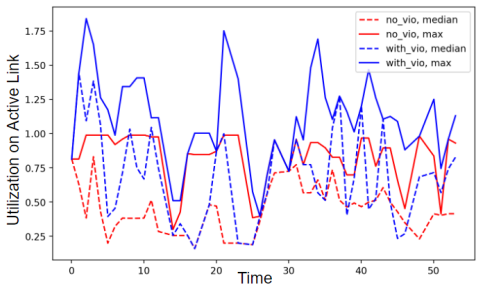


Fig. 15. Link Violation Over Time

for lognormal distribution, the average duration is the same, but the variance increases from first setting to the last setting. We also try three average flow sizes of 70, 85, and 100, with the standard deviation fixed at 10. It is worth noting that with flow size expectation of 85, the network capacity is reached [1]. In Fig. 16, as the average flow size increases, the link utilization increases. With the same average flow size, as the duration variance increases, the link utilization decreases, this is because more dynamic flow duration leads to more rejected flows by the online routing algorithm. Even though the maximum utilization can go up to 1.75, utilization on most of the links are still below one. In Fig. 17, with capacity violation control, no utilization goes over 1, and the differences between different flow sizes and different duration variations become smaller, while the overall trends agree with Fig. 16.

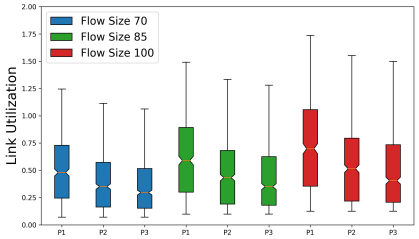


Fig. 16. Link Utilization for Online Routing with Violation

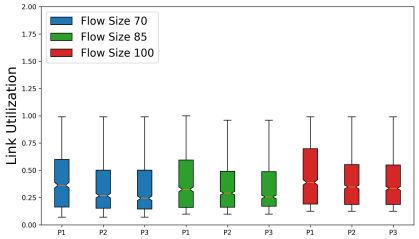


Fig. 17. Link Utilization for Online Routing without Violation

VIII. Conclusion

In this paper, we studied optimal routing in networks with embedded computational services. We developed routing optimization models and fast heuristic algorithms that take into account the unique features of routing with in-network processing and various routing requirements resulted from the computation needs of diverse applications. For the dynamic demand scenario, we came up with an online routing algorithm with a performance guarantee. We demonstrated through evaluations that our models and algorithms are highly customizable and can achieve close-to-optimal performance in a wide range of application scenarios. While the current work is focused on optimizing the network performance, we will investigate the robustness and resilience of RINP in our future work. In particular, we are interested in exploring how traffic routing and computation resource provisioning can help each other to quickly recover from major failures in networks with embedded computational services, as demonstrated in our preliminary results in Section VII-C.

[1]For each demand pair, we calculate its long-term average traffic volume based on the average flow arrival rate, mean flow duration, and mean flow size. We then check the feasibility of the long-term average traffic matrix of all pairs by solving the static LP routing optimization.

References

[1] N. Feamster, J. Rexford, and E. Zegura, "The road to sdn: an intellectual history of programmable networks," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 2, pp. 87–98, 2014.

[2] N. B. Truong, G. M. Lee, and Y. Ghamri-Doudane, "Software defined networking-based vehicular adhoc network with fog computing," in *2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)*. Ieee, 2015, pp. 1202–1207.

[3] M. Shafi, A. F. Molisch, P. J. Smith, T. Haustein, P. Zhu, P. De Silva, F. Tufvesson, A. Benjebbour, and G. Wunder, "5g: A tutorial overview of standards, trials, challenges, deployment, and practice," *IEEE journal on selected areas in communications*, vol. 35, no. 6, pp. 1201–1221, 2017.

[4] A. Basta, W. Kellerer, M. Hoffmann, H. J. Morper, and K. Hoffmann, "Applying nfv and sdn to lte mobile core gateways, the functions placement problem," in *Proceedings of the 4th workshop on All things cellular: operations, applications, & challenges*, 2014, pp. 33–38.

[5] J. G. Herrera and J. F. Botero, "Resource allocation in nfv: A comprehensive survey," *IEEE Transactions on Network and Service Management*, vol. 13, no. 3, pp. 518–532, 2016.

[6] X. Xu, C. He, Z. Xu, L. Qi, S. Wan, and M. Z. A. Bhuiyan, "Joint optimization of offloading utility and privacy for edge computing enabled iot," *IEEE Internet of Things Journal*, vol. 7, no. 4, pp. 2622–2629, 2019.

[7] L. Wang, L. Jiao, T. He, J. Li, and M. Mühlhäuser, "Service entity placement for social virtual reality applications in edge computing," in *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*. IEEE, 2018, pp. 468–476.

[8] M. Rost and S. Schmid, "Charting the complexity landscape of virtual network embeddings," in *2018 IFIP Networking Conference (IFIP Networking) and Workshops*. IEEE, 2018, pp. 1–9.

[9] ——, "Virtual network embedding approximations: Leveraging randomized rounding," *IEEE/ACM Transactions on Networking*, vol. 27, no. 5, pp. 2071–2084, 2019.

[10] R. Yu, G. Xue, and X. Zhang, "Application provisioning in fog computing-enabled internet-of-things: A network perspective," in *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*. IEEE, 2018, pp. 783–791.

[11] R. Cohen, L. Lewin-Eytan, J. S. Naor, and D. Raz, "Near optimal placement of virtual network functions," in *2015 IEEE Conference on Computer Communications (INFOCOM)*. IEEE, 2015, pp. 1346–1354.

[12] T. Lukovszki, M. Rost, and S. Schmid, "Approximate and incremental network function placement," *Journal of Parallel and Distributed Computing*, vol. 120, pp. 159–169, 2018.

[13] V. Valls, G. Iosifidis, G. de Mel, and L. Tassiulas, "Online network flow optimization for multi-grade service chains," in *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*. IEEE, 2020, pp. 1329–1338.

[14] Z. Cao, S. S. Panwar, M. Kodialam, and T. Lakshman, "Enhancing mobile networks with software defined networking and cloud computing," *IEEE/ACM Transactions on Networking*, vol. 25, no. 3, pp. 1431–1444, 2017.

[15] M. Charikar, Y. Naamad, J. Rexford, and X. K. Zou, "Multi-commodity flow with in-network processing," in *International Symposium on Algorithmic Aspects of Cloud Computing*. Springer, 2018, pp. 73–101.

[16] W. Chen, V. Faber, and E. Knill, "Restricted routing and wide diameter of the cycle prefix network," *DIMACS Series in Discrete Mathematics and Theoretial Computer Science*, vol. 21, pp. 31–46, 1995.

[17] V. Heorhiadi, M. K. Reiter, and V. Sekar, "Simplifying software-defined network optimization using {SOL}," in *13th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 16)*, 2016, pp. 223–237.

[18] N. Christofides, "Worst-case analysis of a new heuristic for the travelling salesman problem," Carnegie-Mellon Univ Pittsburgh Pa Management Sciences Research Group, Tech. Rep., 1976.

[19] N. Buchbinder, J. S. Naor *et al.*, "The design of competitive online algorithms via a primal–dual approach," *Foundations and Trends® in Theoretical Computer Science*, vol. 3, no. 2–3, pp. 93–263, 2009.

[20] NYDoT, "Traffic volume counts (2014-2019)," *https://data.cityofnewyork.us/Transportation/Traffic-Volume-Counts-2014-2019-/ertz-hr4r*, 2020.

[21] K. Okamoto, "What is being done with open government data? an exploratory analysis of public uses of new york city open data." *Webology*, vol. 13, no. 1, 2016.

[22] NYDoT, "Nyc street camera real time traffic information," *https://nyctmc.org/multiview2.php*, 2021.

[23] S. Uhlig, B. Quoitin, J. Lepropre, and S. Balon, "Providing public intradomain traffic matrices to the research community," *ACM SIGCOMM Computer Communication Review*, vol. 36, no. 1, pp. 83–86, 2006.

[24] S. Orlowski, M. Pióro, A. Tomaszewski, and R. Wessäly, "SNDlib 1.0–Survivable Network Design Library," in *Proceedings of the 3rd International Network Optimization Conference (INOC 2007), Spa, Belgium*, April 2007, http://sndlib.zib.de, extended version accepted in Networks, 2009. [Online]. Available: http://www.zib.de/orlowski/Paper/OrlowskiPioroTomaszewskiWessaely2007-SNDlib-INOC.pdf.gz