Optimum Transmission Delay for Function Computation in NFV-Based Networks: The Role of Network Coding and Redundant Computing

Behrooz Tahmasebi, Mohammad Ali Maddah-Ali, Member, IEEE, Saeedeh Parsaeefard, Member, IEEE, and Babak Hossein Khalaj

Abstract—In this paper, we study the problem of delay minimization in network function virtualization-based networks. In such systems, the ultimate goal of any request is to compute a sequence of functions in the network, where each function can be computed at only a specific subset of network nodes. In conventional approaches, for each function, we choose one node from the corresponding subset of the nodes to compute that function. In contrast, in this paper, we allow each function to be computed more than one node, redundantly in parallel, to respond to a given request. We argue that such redundancy in computation not only improves the reliability of the network but also, perhaps surprisingly, reduces the overall transmission delay. In particular, we establish that by judiciously choosing the subset of nodes which compute each function, in conjunction with a linear network coding scheme to deliver the result of each computation, we can characterize and achieve the optimal end-to-end transmission delay. In addition, we show that using such technique, it is possible to significantly reduce the transmission delay as compared to the conventional approaches. In fact, in some scenarios, such reduction can even scale with the size of the network, where by increasing the number of nodes that can compute the given function in parallel by a multiplicative factor, the end-to-end delay will also decrease by the same factor. Moreover, we show that while finding the subset of nodes for each computation, in general, is a complex integer program, approximation algorithms can be proposed to reduce the computational complexity. In fact, for the case where the number of computing nodes for a given function is upper bounded by a constant, a dynamic programming scheme can be proposed to find the optimum subsets in polynomial times. Our numerical simulations confirm the achieved gain in performance in comparison with conventional approaches.

Index Terms—Delay-computation trade-off, network coding, network function virtualization (NFV), network optimization, redundancy, reliability.

I. INTRODUCTION

NETWORK function virtualization (NFV) is the state-of-the-art architecture for future data networks. NFV is an enabler to network slicing and cloud over core in 5G which can considerably improve the efficiency of resource utilization [1]–[7]. In NFV-based networks, network functions are virtualized and the resulting virtual network functions (VNFs) can be computed at any node where they are placed. This modification in the architecture of data networks leads to high flexibility and improvement in performance measures [8]. Consequently, resource management will become a key problem in NFV realizations. In the resulting network function virtualization resource allocation (NFV-RA) problem, new challenging sub-problems such as chain composition, forwarding graph embedding, and scheduling will arise and need to be addressed efficiently [8].

In such networks, the ordering of the functions that are computed in the network is known as the service function chaining problem [9]–[11]. The forwarding graph embedding problem is to assign each VNF to one node, based on the set of requests. This problem is known to be NP-hard [8]. Some architectures are proposed to address this issue [12]–[14]. The final stage in the NFV-RA is the problem of scheduling, where we want to assign the execution of a given VNF to one node of the network, in addition to setting the execution times of those functions at the network nodes, given the set of the requests. This problem is formulated in a basic form in [15] and [16], and some solutions are proposed in [17]. Joint NFV-RA and admission control problem have been recently proposed in the literature to increase the resource efficiency, e.g., [18].

In this paper, we deal with the same problem of resource allocation in NFV-based networks, with the objective of minimizing the end-to-end transmission delay. However, we address the problem through a completely novel analytic approach. The key contribution of this work is to demonstrate that under certain conditions, it is possible to reduce the end-to-end delay for computation of VNFs over the network if functions are allowed to be executed redundantly and in
parallel over a number of nodes, an assumption that has not been properly exploited earlier in the literature. Another key observation in this paper is that by allowing a function to be computed redundantly over a number of nodes, a trade-off between overall computational power and end-to-end delay is observed. To the best of our knowledge, this is the first time that such trade-off is investigated in VNF computation. To achieve this trade-off, we rely on two components: (1) redundancy in computation as mentioned above, (2) network coding in delivery. It is interesting to note that, recently, the role of coding for achieving fundamental trade-offs between computation and communication has been investigated in other problems [19]–[30]. Li et al. [19] consider the framework of Map-Reduce, which is a known framework for distributed computing, and show that coding techniques can improve the performance of the system significantly. Such improvements in performance measures via coding theory are also observed in wireless distributed computing problem [22]. The problem of network stragglers is also addressed through use of coding theory for computational tasks. For example, in the problems of distributed matrix multiplication [20] and coded Fourier transform [21], the use of coding theory for computational tasks can alleviate the problem of network stragglers. Also in the problems of distributed optimization, where an optimization problem is divided into sub-optimization problems to reduce the computational complexity and decrease the delay using parallel servers, the problem of straggler servers is challenging. To deal with this challenge, recently, encoded distributed optimization is introduced [25]–[27]. Another example is to use the coding techniques in distributed gradient descent algorithm, which is an important algorithm in many problems of machine learning and data sciences [28]–[30]. The benefit of gradient coding is also for avoiding the stragglers. Also in the NFV-based systems, the coding theoretic approaches are recently used in [23] and [24] to reduce the effect of stragglers in VNF computing nodes. In this work, we introduce network coding to NFV-based networks to achieve optimum transmission delay.

For finding the minimum end-to-end transmission delay analytically, we propose an auxiliary multicast problem, which captures all the assumptions we have made before. This auxiliary problem shows that, we can optimize the end-to-end delay by applying the single-source multicast theorem [31] to the model considered in this paper. Thus, the problem of minimization of the end-to-end transmission delay can be formulated mathematically using the capacity of the auxiliary multicast problem, which is known to be equal to the min-cut in the network coding terminology. In this way, we formulate the problem of finding the subsets of nodes that must compute functions in each round by an optimization problem, specifically as an integer programming problem. Following the results of network coding theory, it is shown that the minimum end-to-end delay can be achieved only via linear codes [32], [33]. Actually, there are polynomial time algorithms for network code construction to exploit the capacity of the system [34]. We also notice that in our model, we do not assume that the functions have any structure, for example they are linear. Our results hold for arbitrary functions.

To use these results in practice, as we stated before, a complex integer programming problem needs to be solved. This problem corresponds to obtaining the set of nodes that will compute any specific function in parallel. While finding the exact solution of this optimization problem has high computational complexity in general, we propose an approximation algorithm for finding the solution with a moderate polynomial complexity. We show that the proposed algorithm outperforms the traditional no-redundancy approaches via numerical simulations. Furthermore, if we have an upper bound of \( \alpha \) on the number of nodes that compute the given function at each round of function computation, where \( \alpha \) does not scale with the size of the network, an optimal algorithm is also provided with polynomial complexity. This algorithm is based on the dynamic programming.

At the final stage, we present some numerical simulations and show the gain of the proposed scheme in several cases, specially in comparison with no-redundancy scheme. This is observed, both theoretically and numerically, that we have a trade-off between the end-to-end delay and the processing cost which means the maximum number of nodes that can compute a given function in parallel. It is interesting to note that, in fact, there are some scenarios that by increasing the number of nodes that can compute a given function in parallel by a multiplicative factor of \( \alpha \), the end-to-end delay will also decrease by the same factor. To the best of our knowledge, it is the first time in the literature that such trade-off is addressed. Note that a natural question about our method is whether it is possible to bound the gain of redundant computing by a constant multiplicative factor for all networks. As a contribution, we have developed some examples in which the gain of redundant computing concatenated with network coding scales with the size of the network. Hence, such constant does not exist.

The rest of this paper is organized as follows. In Section II, we define the problem mathematically. In Section III, we state the main result. The proposed algorithms can be found in Section IV. We present the numerical simulations in Section V and finally, Section VI concludes the paper.

**Notation:** For any positive integer \( K \), we define \( [K] := \{1, 2, \ldots, K\} \). Also, vectors are denoted by bold letters, like \( \mathbf{x} \).

### II. Problem Statement

Consider a communication network modeled by a directed weighted graph. We represent the set of nodes in this network by \( V \) and the set of directed links by \( E \subseteq V \times V \). For any directed link \( e = (u, v) \in E \), its communication capacity is denoted by \( w(e) \). This means that it is possible to send a packet of length \( \lfloor Tw(e) \rfloor \) bits from \( v \) to \( u \) via this link, during \( T \) time slots and assuming that the links are error free. Let us denote this network by \( G = (V, E, w(.) ) \), which corresponds to the network function virtualization infrastructure (NFVI) in the literature.

There is a library of functions denoted by \( F \) which indicates the set of VNFs in our model. We assume that each function\(^1\)

\(^1\)In this paper, we utilize VNF and function interchangeably.
function $f \in \mathcal{F}$ is a deterministic mapping from the set \{0, 1\}^{U_f} to the set \{0, 1\}^{L_f}, for some positive integers $U_f$ and $L_f$, respectively. We denote the set of nodes that have ability to compute a function (or equivalently a VNF) $f \in \mathcal{F}$ by $\mathcal{V}_f \subseteq \mathcal{V}$ for any $f \in \mathcal{F}$. This means that in the placement phase of VNFs, we place any function $f$ into a subset of the network nodes, denoted by $\mathcal{V}_f \subseteq \mathcal{V}$. In this setup, we assume that $\mathcal{V}_f$ is predetermined and the run-times of a function (or VNF) at different nodes are not different, i.e., the processors are assumed to be homogeneous.

In our model, we do not put any restriction to the number of functions that a node can compute. However, for simplicity, we assume that any node can compute at most one function, i.e., $\mathcal{V}_f \cap \mathcal{V}_g = \emptyset$ for any distinct functions $f$ and $g$. This assumption does not affect the generality of the problem. Note that, for any node $v \in \mathcal{V}$ that can compute $k > 1$ distinct functions (VNFs), we can replace it by $k$ new nodes $v_1, v_2, \ldots, v_k$ that can run exactly one of the $k$ functions and form a new directed weighted graph as follows. All the links from or to $v$ in the previous network are considered from or to $v_i$ in the new network. Also, for any distinct $k_1, k_2 \in [K]$, we connect $v_{k_1}$ to $v_{k_2}$ by a directed link with infinite capacity, i.e., $w(v_{k_1}, v_{k_2}) = \infty$. The other nodes, links and weights are remained unchanged. With this modification, the resulting network and the original network are equivalent in terms of delay and the achievable scheme. This shows that the condition is not restrictive.

There is a request tuple $\mathcal{R} = (s, d, x, (f_1, f_2, \ldots, f_K))$ for the computation in network which is defined as follows. The transmitter node or source node $s \in \mathcal{V}$ has the data packet $x \in \{0, 1\}^{L_0}$, where $L_0$ is an arbitrary positive integer. For any $k \in [K]$, $f_k \in \mathcal{F}$ is the function that must be computed in the $k^{th}$ round of the function computation problem. The receiver or destination node $d \in \mathcal{V}$ is interested to be delivered the sequentially computed result $f_K(f_{K-1}(\ldots(f_1(x))\ldots))$.

We assume that the chain is predetermined and the order cannot be changed. Note that the output of the $k^{th}$ function $f_k$ must match in the size with the input of the $(k + 1)^{th}$ function $f_{k+1}$. This shows that there is a sequence of positive integers $L_0, L_1, L_2, \ldots, L_K$ such that for any $k \in [K]$ we have $f_k : \{0, 1\}^{L_k} \rightarrow \{0, 1\}^{L_{k+1}}$. Throughout this paper, we assume that each $L_k$ is large enough. This is essential to establish the results.

The problem is as follows. Given the network graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, w, (\cdot))$, the library of functions $\mathcal{F}$, a request tuple $\mathcal{R} = (s, d, x, (f_1, f_2, \ldots, f_K))$ and the sets $\{\mathcal{V}_f\}_{k \in [K]}$, how can the network use its resources to response $\mathcal{R}$, in order to minimize the end-to-end delay? First the notions of responding to a request and end-to-end delay must be clarified.

Considering a request $\mathcal{R} = (s, d, x, (f_1, f_2, \ldots, f_K))$, in our model, a responding scheme to this request consists of two steps. In the first step, a sequence of non-empty sets $(S_1, S_2, \ldots, S_K)$ must be selected, where $S_k \subseteq \mathcal{V}_{f_k}$ for any $k \in [K]$. For any $k$, $S_k$ is the set of the nodes that will compute the function $f_k$ in the $k^{th}$ round of the function computation problem. In other words, all of the nodes in $S_k$ must compute the function $f_k$. Apparently $|S_k| \geq 1$. Later, we will see why it may be beneficial to have $|S_k| > 1$. Also for the simplicity in notation, we define $S_0 = \{s\}$ and $S_{K+1} = \{d\}$.

The second step consists of $K + 1$ transmission rounds. In the first round, the data packet $x$ must be delivered from the source node $s$ to all of the nodes of the set $S_1$ simultaneously, using the network links. After this round, all of the nodes of the set $S_1$ compute the function $f_1(x)$ simultaneously and in parallel manner. Now $f_1(x)$ is available in all of the nodes of the set $S_1$. In the second round, $f_1(x)$ must be delivered from the set of nodes $S_1$ to each node of the set $S_2$, separately. Therefore, in the $k^{th}$ round, the nodes of the set $S_{k-1}$ have $f_{k-1}(f_{k-2}(\ldots(f_1(x))\ldots))$ and they must deliver this common data to each node of the set $S_k$, using the network links. Note that by this definition, the functions must be computed independently and sequentially. In other words, the order of function computing is important. Also, for any $k \in [K]$, the value of $f_k(f_{k-1}(\ldots(f_1(x))\ldots))$ cannot be computed, unless the value of $f_{k-1}(f_{k-2}(\ldots(f_1(x))\ldots))$ is available at a node that can compute $f_k$.

More formally, in the second step, the sets $(S_1, S_2, \ldots, S_K)$ are available. The data must sequentially be delivered from one set to another. Hence, we assume that a responding scheme must include a sequence $(D_1, D_2, \ldots, D_{K+1})$ of delivery schemes. For any $k \in [K+1]$, $D_k$ is a transmission policy of the common content $f_{k-1}(f_{k-2}(\ldots(f_1(x))\ldots))$, from the set $S_{k-1}$ to the set $S_k$. Notice that all of the nodes of the set $S_{k-1}$ have the common data $f_{k-1}(f_{k-2}(\ldots(f_1(x))\ldots))$ and they want to deliver this common data to the nodes of the set $S_k$ where all of the nodes in $S_k$ must be delivered the content $f_{k-1}(f_{k-2}(\ldots(f_1(x))\ldots))$. In this paper, we consider a multistage graph where in each stage, i.e., between $S_{k-1}$ and $S_k$, there exists the infrastructure network $\mathcal{G}$ (see Figure 1). Therefore, in Figure 1, there exist $K + 1$ copies of infrastructure network for $K + 1$ stages where each cloud is utilized to demonstrate this copy of $\mathcal{G}$ for each stage.

For a responding scheme which corresponds to $(S_1, S_2, \ldots, S_K)$ and $(D_1, D_2, \ldots, D_{K+1})$, the end-to-end delay is defined as follows. For any $k \in [K+1]$, assume that the delay of the transmission round $k$ is denoted by $D_k$ which depends on the request and the responding scheme. More precisely, $D_k$ is a function of the sets $S_{k-1}$ and $S_k$, and the delivery scheme $D_k$. The end-to-end transmission delay $D$ is defined as the summation of the transmission round delays, i.e., $D := \sum_{k=1}^{K+1} D_k$. In this paper, our objective is to minimize $D$, subject to the network considerations and the request. The design parameters are the sets $\{S_k\}_{k \in [K]}$ and the delivery schemes $\{D_k\}_{k \in [K+1]}$.

Note that in the traditional responding schemes, any set $S_k$ is a one-element set which means that $f_k$ is computed at exactly one node. We call this scheme no-redundancy approach. In this paper, we omit this restriction and allow the responding schemes to compute a function redundantly in several nodes of the network in parallel. We will show that via redundancy in function computing, the network delay can decrease, compared to the no-redundancy schemes. We note
that this approach for responding a request, however, may increase the delay in the first round of the data transmission, because the content must be delivered to more than one node. On the other side, for the second round of data transmission, the content $f_1(x)$ is available in more than one node. This means that redundantly function computing may decrease the delay in the second round of data transmission. Similarly, this phenomenon is observed in all rounds of computation. This shows a trade-off for choosing one-element sets or larger sets.

III. MAIN RESULTS

In this section, we state the main results of this paper. In particular, we propose a scheme that minimizes the end-to-end delay $D$. Furthermore, we prove that the proposed scheme is optimal, for the objective of delay minimization. First we give some examples to explain the method.

A. Illustrative Examples

Example 1: Consider the network in Figure 2 (a) and the request tuple $\mathcal{R} = (s, d, x, (f_1, f_2))$. We notice that the data $x$ is available in the source node $s$. There are two functions $f_1$ and $f_2$. The destination node $d$ is interested in receiving the computed result of $f_2(f_1(x))$. We assume that the functions $f_1$ and $f_2$ can be computed in the nodes $V_{f_1} = \{v_{11}, v_{12}\}$ and $V_{f_2} = \{v_{21}, v_{22}\}$, respectively. The capacity of each link is equal to one, i.e., $w(e) = 1$ for any link $e$. Also, we assume that $L_0 = L_1 = L_2$. One can see that using no-redundancy scheme for function computation, the normalized delay of $D/L_0 = 3$ is achievable (see Figure 2 (b)). Furthermore, it is obvious that using only one-element sets for $\mathcal{S}_1$ and $\mathcal{S}_2$, we cannot have an end-to-end transmission delay which is smaller than $3L_0$. On the other side, if we redundantly compute the functions $f_1$ and $f_2$, we can achieve a more small delay. In particular, consider $\mathcal{S}_1 = \{v_{11}, v_{12}\}$ and $\mathcal{S}_2 = \{v_{21}, v_{22}\}$, which means that the functions $f_1$ and $f_2$ are computed in two nodes simultaneously and in parallel. In this case, we can prove that the total delay decreases (see Figure 2 (c)). It is shown in Figure 2 (c) that using this method, the normalized delay of $D/L_0 = 2$ is achievable. This means that in this example, redundantly function computation outperforms the traditional no-redundancy method, by a factor of $3/2$.

Remark 1: Note that in Example 1 and also in the next example, there is a bipartite graph between nodes computing $f_k$ and nodes computing $f_{k+1}$. However, in the rest of this paper, we consider a general scheme where we assume existence of an arbitrary network between nodes in layer $k$ and layer $k + 1$ which may not be necessarily one-hop (see the clouds in Figure 1). In other words, we do not restrict ourselves to a specific network topology.

Example 2: Consider the network in Figure 3 (a) and the request tuple $\mathcal{R} = (s, d, x, (f_1, f_2, \ldots, f_K))$. Similar to Example 1, there is a data packet $x$, available in the source node $s$. The destination node $d$ is interested in receiving the computed result of $f_K(f_{K-1}(\ldots f_1(x) \ldots))$. Assume that any function $f_k$ can be computed in the nodes $V_{f_k} = \{v_{k1}, v_{k2}, \ldots, v_{kN}\}$. In addition, let $w(e) = 1$ for any link $e$. In this example, assume that $L_0 = L_1 = \ldots = L_K$. Similar to Example 1, $D/L_0 = K + 1$ is the best achievable normalized delay using no-redundancy approaches for the function computation (see Figure 3 (b)). However, one can compute the functions redundantly in the nodes. In particular, assume that to respond the request, any function $f_k$ is computed in all of the nodes which are capable to compute it, i.e., $\mathcal{S}_k = V_{f_k}$. Via this approach, the network delay decreases. From Figure 3 (c), the normalized delay of $D/L_0 = (1 + K/N)$ is achievable by this method. This shows that using the proposed approach for function computation, the end-to-end delay decreases by a factor of $\frac{1+K}{1+K/N}$. For large values of $K$, we have $\frac{1+K}{1+K/N} \approx N$. This means that the gain of the redundantly function computing can increase as the size of the network increases.

B. Main Results

In the previous subsection, the main idea of redundant function computation has been discussed. For the statement of the main result for general networks, first we need a definition.

Definition 1 (Cut): For a directed weighted graph $G = (\mathcal{V}, \mathcal{E}, w(\cdot))$, and any $\mathcal{S} \subseteq \mathcal{V}$, cut$(\mathcal{S})$ is defined as

$$\text{cut}(\mathcal{S}) = \sum_{e=(u,v) \in \mathcal{E}, u \in \mathcal{S}, v \notin \mathcal{S}} w(e).$$
In the next theorem, we state a tight lower bound on $D_k$.

**Theorem 1**: For any $k \in [K+1]$, any $S_{k-1} \subseteq \mathcal{V}_{f_{k-1}}$, and any $S_k \subseteq \mathcal{V}_{f_k}$, the delay of the $k^{th}$ transmission round can be lower bounded as follows

$$D_k \geq \frac{L_{k-1}}{\min_{S_{k-1} \subseteq S \subseteq \mathcal{V}} \text{cut}(S)}.$$  

\(1\)
Moreover, the lower bound is tight, i.e., there exists a transmission scheme $\mathcal{D}_k$ that achieves the lower bound.

Proof: See Appendix A.

Remark 2: To prove Theorem 1, we utilize the results from network coding, specifically the capacity of the single multicast problem [31]. We show that the delivery problem in each round can be reduced to a single-multicast problem where the capacity is equal to the min-cut, and can be achieved by linear codes.

Remark 3: The above result shows that routing is not generally optimum. In particular, for delivery network coding should be applied.

Corollary 1: From Theorem 1, for a given sequence of sets $(S_1, S_2, \ldots, S_K)$, the end-to-end delay is lower bounded by

$$D \geq \sum_{k=1}^{K+1} \min_{S_k} \left\{ \sum_{v=1}^{K} \frac{L_{k-1}}{\min_{S_v} \sum_{S \subseteq V} \text{cut}(S)} \right\},$$

(2)

In addition, the lower bound is achievable.

Theorem 2: The optimum end-to-end delay of the problem, denoted by $D^*$, is equal to

$$D^* = \min_{\forall k \in [K]} \left\{ \sum_{v=1}^{K} \frac{L_{k-1}}{\min_{S_v} \sum_{S \subseteq V} \text{cut}(S)} \right\}.$$  

(3)

In addition, the optimum sets $(S_1^*, S_2^*, \ldots, S_K^*)$ are arguments of the above optimization.

Proof: See Appendix B. □

Remark 4: To achieve $D^*$, we let $S_1 = S_1^*, S_2 = S_2^*, \ldots, S_K = S_K^*$ and we develop linear codes in delivery rounds.

Remark 5: Via Theorem 2, the minimum achievable end-to-end delay of the problem for a general network is characterized. However, to achieve this minimum delay, a complicated optimization problem must be solved to obtain the sets $\{S_k\}_{k \in [K]}$. Solving this optimization problem is not straightforward. In the next section, we propose some approximation algorithms to solve this problem.

We note that the size of the set $S_k$ corresponds to the processing cost of $f_k$. Hence, it is rational to have constraint on the cardinality of each set $S_k$. The following corollary states the optimization problem with the processing power constraint on the sets. This means that we restrict the optimization to the cases that each set $S_k$ has a bounded size.

Corollary 2: The problem of minimization of the end-to-end delay, subject to limit on processing power in each round is formulated as

$$\min_{\forall k \in [K]} \left\{ \sum_{v=1}^{K} \frac{L_{k-1}}{\min_{S_v} \sum_{S \subseteq V} \text{cut}(S)} \right\},$$

(4)

where $\alpha_k$ denotes the maximum processing cost of $f_k$.

Remark 6: Note that the above formula for the end-to-end delay shows that by increasing the processing costs, i.e., $\alpha_k$’s, the end-to-end delay can decrease, because the set of feasible solutions of the optimization enlarges. This shows that there is a trade-off between end-to-end delay and processing cost of each request.

For the rest of this section, we explain the results of redundantly function computation in a special case of complete graphs. The main goal of this example is to demonstrate that there are also fully connected networks (not necessarily one-hop like Example 2) that the gain of the proposed scheme, i.e., redundantly computation and network coding in delivery, scales with the size of the network. Also, we attend to demonstrate the achieved trade-offs between computation and delay via this example.

C. Another Example

Consider the network $\mathcal{G} = (V, E, w(\cdot))$ where $V$ is a finite set of nodes and $E$ contains all of the distinct pairs of the nodes, i.e., the graph is fully connected. In this network, assume that there exists a request of computation as $\mathcal{R} = (s, d, x, (f_1, f_2, \ldots, f_K))$, and for any $k \in [K]$, $\mathcal{V}_{f_k} = \{v_{k1}, v_{k2}, \ldots, v_{KN}\}$ is a set containing $N$ nodes for a positive integer $N$. This means that the network consists of $K \times N$ function computing nodes, in addition of other nodes as relays. Suppose that we have at least $N + 1$ relays. Also, two nodes $s$ and $d$ cannot compute any function. In this case, for any $e = (u, v) \in E$ we define

$$w(e) = \begin{cases} \epsilon & \text{if } u \in \mathcal{V}_{f_k} \text{ for some } k \in [K], \text{ or } u = s, \\ 1 & \text{otherwise}, \end{cases}$$

where $\epsilon$ is a small enough positive. In this fully connected network, let us assume that there is an upper bound on processing cost as $|S_k| \leq \alpha$ where $\alpha \leq N$. Now we apply results of Theorem 2 and Corollary 2 to this network.

We want to compute the lower bound in Corollary 2. Note that we have

$$\text{cut}(S_k) = \epsilon \times |S_k| \times (|V| - |S_k|),$$

(5)

for any $k \in [K] \cup \{0\}$. To derive $D^*$ from Theorem 2, in this setup, we introduce the following lemma.

Lemma 1: Assuming small enough $\epsilon$, for the mentioned network we have

$$\min_{S_k \subseteq S \subseteq V} \text{cut}(S) = \epsilon \times |S_{k-1}| \times (|V| - |S_{k-1}|),$$

(6)

for any $k \in [K + 1]$.

Proof: See Appendix C. □

Now based on Theorem 1, we write (1) as

$$D_k \geq \frac{L_{k-1} \min_{S_k \subseteq S \subseteq V} \text{cut}(S)}{\epsilon \times |S_{k-1}| \times (|V| - |S_{k-1}|)},$$

(7)

where $a$ follows by Lemma 1. Therefore

$$D \geq \sum_{k=1}^{K+1} \frac{L_{k-1}}{\epsilon \times |S_{k-1}| \times (|V| - |S_{k-1}|)},$$

(9)

The need for $\epsilon$ to be small enough is due to such requirement for the proof of Lemma 1.
The R.H.S. of (9) is minimized when $\mathcal{S}_k$ is an arbitrary subset of $\mathcal{V}_{f_k}$ with size $\alpha$ for any $k \in [K]$. Hence, we have

$$D_{Optimum} \times \epsilon = \frac{L_0}{|\mathcal{V}| - 1} + \frac{1}{\alpha \times (|\mathcal{V}| - \alpha)} \sum_{k=1}^{K} L_k. \quad (10)$$

Consider $L_0 = L_1 = \ldots = L_K$. In this case, we conclude that by redundantly function computation, one can achieve the normalized delay of $D_{Optimum} \times \epsilon / L_0 = \frac{1}{|\mathcal{V}| - \alpha} \sum_{k=1}^{K} L_k$. Also, if we utilize no-redundancy approach, we achieve the normalized delay of $D_{No-reduancy} \times \epsilon / L_0 = \frac{1}{|\mathcal{V}| - \alpha} \sum_{k=1}^{K} L_k$, where we have $|\mathcal{S}_k| = 1$ for any $k \in [K]$ in this approach. If $K$ is large, and also $\alpha \ll |\mathcal{V}|$, then we conclude that the improvement is approximately equal to $\alpha$, i.e.,

$$\frac{D_{No-reduancy}}{D_{Optimum}} \approx \alpha.$$

We notice that while the parameter $\alpha$ indicates the available processing cost of each function, in this example, it can simultaneously model the reliability measure. In other words, if we set $\alpha$ to be larger, then any function is computed in more nodes of the network and hence, the system is more reliable against the node failures in functions computation. This means that as $\alpha$ increases, the system becomes more reliable.

Now we notice that for this example, the fundamental trade-offs among end-to-end delay, reliability and processing cost are obtained. In Figure 4, we plot the normalized delay in two cases; the no-redundancy approach and the proposed method. Here, an order-wise improvement in the normalized end-to-end delay is observed. Note that in this plot, $\alpha$ represents the processing cost and simultaneously it models the reliability in the function computation.

4This is due to the fact that the function $f(x) = x(|\mathcal{V}| - x)$ for $x \in \{1, 2, \ldots, \alpha\}$ is increasing since $\alpha \leq |\mathcal{V}|/2$.

4This fact is due to equation (9).

IV. ALGORITHMS

In this section, we propose our algorithms for solving the optimization problems (3) and (4). First we propose an approximation algorithm for solving (3), called greedy algorithm. Then, we propose an algorithm for solving (4) in polynomial times. This algorithm is called $\alpha$-optimal algorithm. The third algorithm is the same as the greedy algorithm, except it aims to solve (4). We note that all the proposed algorithms are based on dynamic programming.

A. Greedy Algorithm

In this part, we propose our algorithm for solving the following optimization problem

$$D^* = \min_{\forall k \in [K]: \mathcal{S}_k \subseteq \mathcal{V}_k} \left\{ \sum_{k'=1}^{K+1} \min_{S_k' \subseteq \mathcal{V}_k} \left[ \frac{L_{k'\leftarrow 1}}{\min_{S_{k'\leftarrow 1} \subseteq \mathcal{S}_{k'\leftarrow 1} \subseteq \mathcal{V}} \text{cut}(S)} \right] \right\}. \quad (11)$$

In this optimization problem, our aim is to find the sets $\{\mathcal{S}_k\}_{k \in [K]}$ in order to minimize the end-to-end delay. First we note that the optimization problem

$$\min_{\forall k \in [K]: \mathcal{S}_k \subseteq \mathcal{V}_k} \left[ \frac{L_{k'\leftarrow 1}}{\min_{S_k' \subseteq \mathcal{V}_k} \text{cut}(S)} \right] = \min_{S_k' \subseteq \mathcal{V}_k} \text{cut}(S). \quad (12)$$

is known as Max-flow problem. This problem can be solved via efficient known algorithms such as Ford-Fulkerson algorithm [35]. Therefore, the optimization problem (11) can be rewritten as

$$\min_{\forall k \in [K]: \mathcal{S}_k \subseteq \mathcal{V}_k} \left\{ \sum_{k'=1}^{K+1} \min_{S_k' \subseteq \mathcal{V}_k} \text{cut}(S_{k'\leftarrow 1}; S_k') \right\}. \quad (13)$$

Let us define

$$C(\mathcal{S}_k) := \min_{\forall k' \in [k-1]: \mathcal{S}_k' \subseteq \mathcal{V}_{f_k'}} \left\{ \sum_{k''=1}^{k} \min_{S_k'' \subseteq \mathcal{V}_{f_k''}} \text{cut}(S_{k''\leftarrow 1}; S_k'') \right\}. \quad (14)$$

for any $\mathcal{S}_k \subseteq \mathcal{V}_{f_k}$ and any $k \in [K]$. Considering $C(\{s\}) = 0$ and $D^* = C(\mathcal{S}_{k+1}) = C(\{d\})$ from (14), we achieve the following recursive equation

$$C(\mathcal{S}_k) = \min_{\mathcal{S}_{k-1} \subseteq \mathcal{V}_{f_{k-1}}} \left\{ C(\mathcal{S}_{k-1}) + \frac{L_{k-1}}{\min_{\mathcal{S}_{k-1} \subseteq \mathcal{V}} \text{cut}(\mathcal{S}_{k-1}; \mathcal{S}_k)} \right\}. \quad (15)$$

for any $k \in [K+1]$. The recursive equation (15) shows that the problem of finding the optimum subsets is in the form of dynamic programming and can be solved recursively [36, Ch. 6].

Now for estimation of the solution of (13), first we use dynamic programming, according to the above discussions, to obtain one-element sets with minimum delay. In other words, we initiate the algorithm by choosing optimum one-element sets. Then, we enlarge the sets in a greedy manner as long as the overall delay decreases. More precisely, at each round, we choose one function computing node from all the function computing nodes which have not been selected yet, and then, add it to the corresponding subset when this change decreases the end-to-end delay. This procedure is performed, again, via dynamic programming (see lines 8-16.
Algorithm 1 Greedy Algorithm

1: procedure GREEDY(G, s, d, \{L_k\}_{k=0}^{K} \{V_f_k\}_{k \in [K]})
2: for k ∈ [K + 1] ∪ {0} do
3: \(P_k \leftarrow \emptyset\)
4: \(B_k \leftarrow \{S \subseteq V_f_k : |S| = 1\}
5: end for
6: D \leftarrow \infty
7: while 1 do
8: \(OPT(\{s\}) \leftarrow \{s\}\)
9: \(C(\{s\}) \leftarrow 0\)
10: for k ∈ [K + 1] do
11: \(\bar{S} \leftarrow \argmin_{T \in B_{k-1}} \{C(T) + \frac{L_{k-1}}{\mincut(T,S)}\}\)
12: \(C(S) \leftarrow C(\bar{S}) + \frac{L_{k-1}}{\mincut(S,S)}\)
13: \(OPT(S) \leftarrow (OPT(\bar{S}),S)\)
14: end for
15: if D ≤ C(\{d\}) then
16: break
17: end if
18: \((P_0, P_1, \ldots, P_{K+1}) \leftarrow OPT(\{d\})\)
19: D \leftarrow C(\{d\})
20: for k ∈ [K + 1] ∪ {0} do
21: \(B_k \leftarrow \{S \cup P_k : S \subseteq V_f_k : |S| = 1\}\)
22: end for
23: return OPT(\{d\})
24: end while
25: end procedure

Algorithm 2 \(\alpha\)-Optimal Algorithm

1: procedure \(\alpha\text{-OPTIMAL}(G, s, d, \{L_k\}_{k=0}^{K} \{V_f_k\}_{k \in [K]}, \alpha)\)
2: for k ∈ [K + 1] ∪ {0} do
3: \(B_k \leftarrow \{S \subseteq V_f_k : |S| \leq \alpha\}\)
4: end for
5: \(OPT(\{s\}) \leftarrow \{s\}\)
6: \(C(\{s\}) \leftarrow 0\)
7: for k ∈ [K + 1] do
8: for S ∈ \(B_k\) do
9: \(S' \leftarrow \argmin_{T \in B_{k-1}} \{C(T) + \frac{L_{k-1}}{\mincut(T,S)}\}\)
10: \(C(S) \leftarrow C(S') + \frac{L_{k-1}}{\mincut(S,S)}\)
11: \(OPT(S) \leftarrow (OPT(S'),S)\)
12: end for
13: end for
14: return OPT(\{d\})
15: end procedure

B. \(\alpha\)-Optimal Algorithm

In this part, we aim to solve the following optimization problem:

\[
\min_{\forall k \in [K] : S_k \subseteq V_k} \left\{ \sum_{k=1}^{K+1} \frac{L_{k-1}}{\mincut(S_{k-1};S_k)} \right\}. \tag{16}
\]

Without loss of generality, assume that \(\alpha_k = \alpha\) for any \(k\).
This is just for simplicity in notation and the proposed algorithm can be considered for the general case.

To solve (16), we propose an algorithm in this part, which is called \(\alpha\)-optimal algorithm. This algorithm is based on dynamic programming to obtain the sets \(\{S_k\}_{k \in [K]}\). The algorithm searches among all the subsets of \(V_f_k\) with at most \(\alpha\) elements, for any \(k\), and chooses the best set to minimize the end-to-end delay, according to the recursive equation of dynamic programming. Similar to (15), the recursive equation of (16) is given by

\[
C(S_k) = \min_{S_{k-1} \subseteq V_{k-1}} \left\{ C(S_{k-1}) + \frac{L_{k-1}}{\mincut(S_{k-1};S_k)} \right\}. \tag{17}
\]

The \(\alpha\)-optimal algorithm computes the function \(C(\cdot)\) for each set with at most \(\alpha\) elements by dynamic programming. The details can be found in Algorithm 2. The notations used in this algorithm are also the same as Algorithm 1.

We note that the complexity of the \(\alpha\)-optimal algorithm is \(A \times O((K + 1)|V|^3)\) where \(A\) is the complexity of the algorithm which is developed to find the min-cuts. The remarkable feature of Algorithm 2 is that it is an optimal algorithm with polynomial complexity of the network size.

Let us provide more explanation about the complexity of this algorithm.

In this algorithm, we use dynamic programming only once. In that procedure, the function \(C(\cdot)\) must be computed for each layer \(k\), and for each possible \(S_k\), where \(k \in [K + 1]\). We note that the number of subsets of \(V_f_k\) with at most \(\alpha\) elements, for any \(k\), is \(O(|V|^\alpha)\). Hence, by taking lines 7-13 of Algorithm 2 into consideration, we conclude that it is necessary to compute
Algorithm 3 \( \alpha \)-Greedy Algorithm

1: procedure \( \alpha \)-GREEDY\(( \mathcal{G}, s, d, \{ L_k \}_{k = 0}^{K} \mathcal{V}_{f_k}, k \in [K]; \alpha \) \)
2: \hspace{1em} for \( k \in [K + 1] \cup \{0\} \) do
3: \hspace{2em} \mathcal{T}_k \leftarrow \emptyset
4: \hspace{2em} \mathcal{B}_k \leftarrow \{ \mathcal{S} \subseteq \mathcal{V}_{f_k} : |\mathcal{S}| = 1 \}
5: \hspace{1em} end for
6: \hspace{1em} \mathcal{D} \leftarrow \infty
7: \hspace{1em} while 1 do
8: \hspace{2em} OPT\((\{ s \}) \) \leftarrow \{ s \}
9: \hspace{2em} C(\{ s \}) \leftarrow 0
10: \hspace{2em} for \( k \in [K + 1] \) do
11: \hspace{3em} for \( \mathcal{S} \in \mathcal{B}_k \) do
12: \hspace{4em} \hat{\mathcal{S}} \leftarrow \arg\min_{T \in \mathcal{B}_{k-1}} \{ C(T) + \frac{L_{k-1}}{\operatorname{mincut}(T, \mathcal{S})} \}
13: \hspace{4em} C(\mathcal{S}) \leftarrow C(\hat{\mathcal{S}}) + \frac{L_{k-1}}{\operatorname{mincut}(\mathcal{S}, \hat{\mathcal{S}})}
14: \hspace{4em} OPT(\mathcal{S}) \leftarrow (OPT(\hat{\mathcal{S}}), \mathcal{S})
15: \hspace{3em} end for
16: \hspace{2em} end for
17: \hspace{1em} if \( \mathcal{D} \leq C(\{ d \}) \) then
18: \hspace{2em} break
19: \hspace{1em} end if
20: \hspace{1em} (\mathcal{P}_0, \mathcal{P}_1, \ldots, \mathcal{P}_{K+1}) \leftarrow OPT(\{ d \})
21: \hspace{1em} \mathcal{D} \leftarrow C(\{ d \})
22: \hspace{1em} for \( k \in [K + 1] \cup \{0\} \) do
23: \hspace{2em} if \( |\mathcal{P}_k| < \alpha \) then
24: \hspace{3em} \mathcal{B}_k \leftarrow \{ \mathcal{S} \cup \mathcal{P}_k : \mathcal{S}, \mathcal{P}_k \subseteq \mathcal{V}_{f_k}, |\mathcal{S}| = 1 \}
25: \hspace{2em} end if
26: \hspace{2em} end if
27: \hspace{1em} end while
28: \hspace{1em} return OPT(\{ d \})
29: end procedure

min-cuts for at most \( O((K + 1) \times |\mathcal{V}|^\alpha \times |\mathcal{V}|^\alpha) \) times. This shows that the complexity of the proposed algorithm is \( A \times O((K + 1)|\mathcal{V}|^\alpha \).

C. \( \alpha \)-Greedy Algorithm

We notice that the complexity of the \( \alpha \)-optimal algorithm is not equal to a polynomial of \( \alpha \). To solve this issue, we introduce \( \alpha \)-greedy algorithm as an approximation algorithm. This algorithm is exactly like Algorithm 1, except it does not consider subsets with more than \( \alpha \) elements (see lines 23-25 of Algorithm 3). Details of this algorithm can be found in Algorithm 3. We note that the complexity of this algorithm is at most \( A \times O((K + 1)|\mathcal{V}|^\alpha) \), like the greedy algorithm.

V. NUMERICAL SIMULATIONS

In this section, we evaluate the performance of the proposed algorithms by numerical simulations. To this end, we consider a network which is generated at random and then compare the performance of the proposed algorithms with conventional no-redundancy approach. We note that although random network construction is not necessarily the same as a given practical scenario, it is a reasonable assumption for arguing that the algorithms lead to reasonable performance in typical network scenarios.

Our setup for simulations in this section is similar to Example 2 (see Figure 3), except we assume that the capacities are not equal. The reason that we choose this model for our simulations is that the performance of our algorithms must be evaluated in the cases that varying the subsets changes the flow of the network as much as possible. This makes it possible to evaluate the performance, in the worst-case scenario.

We assume that \( L_0 = L_1 = \ldots = L_K \). Also we assume that the capacities of the links are chosen randomly and independently. The capacity of each link is chosen uniformly from the set \((1 - U, 1 + U)\), with probability \( p \), and is set to be zero, with probability \( 1 - p \). Here \( U \) and \( p \) are two parameters. For the initialization, we assume that \( N = K = 10 \) and \( \alpha = 2 \). Also we set \( p = 1 \) and \( U = 0.5 \). In all of the simulations, we iterate the algorithms \( 10 \) times for the independent inputs and then we consider the average performance as the output.

In Figure 5, we examine the performance of the proposed algorithms versus \( N \). Remember that the parameter \( N \) corresponds to the number of nodes that can compute a
specific function. From Figure 5, we observe that the 2-greedy algorithm approaches the solution of the 2-optimal algorithm. Also, it is observed that the greedy algorithm can improve the performance of the system when $N$ enlarges. This fact is also motivated theoretically (see Figure 4).

In Figure 6, we investigate the performance of the proposed algorithms versus $K$ which is the number of functions that must be computed in the chain. From Figure 6, the result of the 2-greedy algorithm approaches to the 2-optimal algorithm. Also, the greedy algorithm outperform one-element sets or two-elements.

Figure 7 demonstrates the end-to-end delay versus the parameter $\alpha$. It is observed that the greedy algorithm is near optimal, where for large $\alpha$, the performance of the greedy algorithm is close to the optimal case.

Figure 8 demonstrates the performance of the proposed algorithms versus $p$. Note that $p$ corresponds to the connectivity of the network. The simulations show that the proposed algorithms have a good performance, even when the network is sparse (small values for $p$).

At the end, we examine the algorithms for the case that the capacities have large variance, i.e., $U$ is set to be large. It is observed that in this case the performance of the algorithms does not decay and they have gains near to the theoretical analysis. This shows that the performance of the algorithms is not sensitive to the homogeneity of the capacities.

For the end of this section, we note that our proposed greedy algorithms are not optimal in general, but our simulations reveal that they have acceptable performance for most scenarios.

VI. CONCLUSION AND DISCUSSION

In this paper, we investigated the fundamental limits of end-to-end delay minimization in NFV-based networks. It was observed that traditional no-redundancy approaches are not enough for achieving the optimum delay since there are some examples that show the inefficiency of them. To exploit the capacity of such networks, in this paper, it has been assumed that to respond a request, the functions can be computed redundantly in parallel, in addition to use the network coding in the system. It was showed that redundantly function computation and using network coding for delivery, not only improves the reliability measures, but also decreases the transmission delay. In some cases, this gain would also scale by the size of the network, in comparison with the traditional no-redundancy approaches. The sufficiency of linear codes to exploit the capacity of the system was also shown. Then, it was observed that the problem of finding the subsets of the nodes that must compute the functions at each round in general is related to a complex integer programming problem. To this end, an approximation algorithm was proposed. The optimal algorithm with polynomial complexity for the case that there is an upper bound on the number of nodes computing functions in each round was also provided. The performances of the algorithms were evaluated through numerical simulations in several cases, where this was showed that they reach
order-wise improvements in comparison with no-redundancy approaches.

For future work, a number of problems might be of interest. First, we note that in this paper, we proposed a method to decrease the end-to-end delay through a theoretical modeling, and argued that our algorithms lead to good performance for typical network topologies. In this direction, an important question is to evaluate performance under more realistic network models. As an example, an interesting direction is to consider central cloud-edge cloud scenarios where the computing capacities of the nodes and link capacities may vary significantly, and in some scenarios, lead to less connectivity in the network. Therefore, it is fair to say that understanding the benefit of redundant computing in specific network topologies remains an open problem.

On another direction, we note that in this paper, we have considered the class of networks which are described by directed graphs. However, undirected network models have also been of interest. A natural question is that what will happen if we consider such network models? We note that if the network consists of two-way links with fixed capacities, then the results of this paper are still valid. However, if we have constraints on the sum capacities of undirected links, then the problem is challenging and can be addressed as future work. Another future direction is to consider more than one request which must be served simultaneously.

We should also note that from a practical point of view, our results suggest that one should consider the trade-off between delay and computation/communication cost in real-world designs. We suggest that for each network topology and network functions class, the achievable gain in decreasing the delay, in terms of computation/communication cost should be evaluated. Also there is a number of challenges for implementing the redundant computation. For example, we note that in this paper, the servers are assumed to be synchronized. However, they may have different processing times and this leads to its own implementation challenges.

Another interesting question that can be investigated in future is how to address the placement of VNFs, or design of the network infrastructure, in order to potentially achieve high gains through the proposed model of redundant computation? The answer to such question would naturally have significant effect on choice of system parameters for a given practical application and the resulting quality of service and experienced at the user side.

APPENDIX

A. Proof of Theorem 1

For the proof of Theorem 1, we use the single-source multicast theorem [31]. First we add an extra node, denoted by $EN$, to the infrastructure graph $\mathcal{G}$ and form a new directed graph $\tilde{\mathcal{G}}$. In $\tilde{\mathcal{G}}$, assume that $EN$ is connected to each node of $S_{k-1}$ via directed links with infinite capacity.

Consider a single-multicast problem from $EN$ to all of the nodes in $S_k$. We claim that the resulting auxiliary multicast problem is equivalent to the original problem. To show this fact, first we note that each solution of the auxiliary multicast problem, is also a solution of the original problem. This is due to the fact that for each solution of the auxiliary problem, the source node $EN$ first sends the data to the nodes in $S_{k-1}$, because they are neighbors of $EN$ in $\tilde{\mathcal{G}}$. This procedure does not experience any delay, because the corresponding links have infinite capacities. Subsequently, the nodes in $S_{k-1}$ deliver the data to the destination nodes $S_k$. For converse, consider a solution for the original problem. We construct a solution for the auxiliary problem with the same delay as the original problem. Assume that first the node $EN$ transmits all the data to each node in $S_{k-1}$. Again, note that this procedure does not have any delay, due to the infinite capacity of links. Consequently, we apply the solution of the original problem to the auxiliary problem, such that nodes in $S_{k-1}$ deliver their common data to each node in $S_k$. This completes the proof of the equivalency of two problems.

Now we apply the single-source multicast theorem [31] to the auxiliary problem to compute the minimum delay. In the network coding terminology, this theorem shows that the capacity of such system is equal to the capacity of min-cut in $\tilde{\mathcal{G}}$. Hence, we have

$$D_k \geq \min_{v \in S_k} \min_{v \in S_{k-1}} \min_{v \in S_k} \min_{S_{k-1} \subseteq S \subseteq V} \text{cut}(S),$$

where the notation “mincut” is defined in (12). Now we notice that if $S_{k-1} \not\subseteq S$, then $\text{cut}(S)$ is equal to infinity since the capacity of directed links between $EN$ and each node in $S_{k-1}$ is infinity. Hence, we have

$$D_k \geq \min_{v \in S_k} \min_{S_{k-1} \subseteq S \subseteq V} \min_{v \in S_k} \min_{S_{k-1} \subseteq S \subseteq V} \text{cut}(S),$$

This completes the proof. We emphasize that the capacity of the single-source multicast problem can be achieved using linear codes [32]. This proves the achievability of the lower bound, by using the optimal linear network codes in the corresponding transmission round. Note that in the single-source multicast theorem, for achieving the capacity, it is essential to use the network coding and routing is not generally optimum. In addition, the message size (or equivalently $L_{k-1}$) needs to be large enough, in order to ensure existence of capacity achieving network codes.

B. Proof of Theorem 2

Based on Theorem 1, we have

$$D = \sum_{k=1}^{K+1} D_k \geq \sum_{k=1}^{K+1} \min_{S_{k-1} \subseteq S \subseteq V} \text{cut}(S).$$

(22)
Also, the above lower bound is tight, by using the achievable scheme of Theorem 1 in all rounds of data transmission. Hence, the above achievable lower bound on end-to-end delay just depends on the subsets \(\{S_k\}_{k \in [K]}\). This means that the optimum delay can be achieved by minimizing the above equation as a function of the subsets. This completes the proof.

### C. Proof of Lemma 1

First notice that for \(S = S_k-1\), we have
\[
\text{cut}(S) = \epsilon \times |S_k-1| \times (|V| - |S_k-1|).
\]

In order to prove the lemma, it suffices to show that for any \(S \subseteq V\) such that \(S_k-1 \subseteq S\) and \(S_k \not\subseteq S\), we have \(\epsilon \times |S_k-1| \times (|V| - |S_k-1|) \geq \text{cut}(S)\).

Note that if \(|S_k-1| \leq |S| - |S_k-1|\), then we have
\[
\text{cut}(S) \geq \epsilon \times |\{(u, v) \in E : u \in S, v \not\in \tilde{S}\}|,
\]
and in this case the proof is completed.

Now consider the case that \(|S| < |S_k-1|\) or \(|S| > |V| - |S_k-1|\). The case \(|S| < |S_k-1|\) is impossible, due to the fact that \(S_k-1 \subseteq S\). Hence, we assume that \(|S| > |V| - |S_k-1|\) \(\geq K + N + 1 - N = K + N\). This means that \(S\) contains at least one relay node which is distinct of \(s\), such as \(\tilde{u} \in S\).

Since \(S_k \not\subseteq S\), there is a node \(\tilde{v} \in S_k\) such that \(\tilde{v} \not\in S\). Let us define \(\tilde{e} = (\tilde{u}, \tilde{v})\). Therefore, we write
\[
\text{cut}(S) \geq w(\tilde{e}) \geq 1 \geq \epsilon \times |S_k-1| \times (|V| - |S_k-1|),
\]}

where (a) follows from the fact that \(\epsilon\) is set to be small enough. This completes the proof.
Behrooz Tahmasebi received the B.Sc. and M.Sc. degrees from the Department of Electrical Engineering, Sharif University of Technology, Tehran, Iran, in 2016 and 2018, respectively. From 2017 to 2018, he was a Research Scientist with the Department of Communication Technologies, Iran Telecommunication Research Center, Tehran. His research interests include information theory, probability theory, game theory, and bioinformatics.

Mohammad Ali Maddah-Ali (S’03–M’08) received the B.Sc. degree in electrical engineering from the Isfahan University of Technology, in 1997, the M.A.Sc. degree in electrical engineering from the University of Tehran, in 2000, and the Ph.D. degree from the Coding and Signal Transmission Laboratory, Department of Electrical and Computer Engineering, University of Waterloo, Canada, in 2007. From 2007 to 2008, he was with the Wireless Technology Laboratories, Nortel Networks, Ottawa, ON, Canada. From 2008 to 2010, he was a Post-Doctoral Fellow with the Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, Berkeley, CA, USA. Then, he joined Bell Labs, Holmdel, NJ, USA, as a Communication Research Scientist. He is currently a Faculty Member with the Sharif University of Technology.

Dr. Maddah-Ali was a recipient of the NSERC Postdoctoral Fellowship in 2007, the IEEE Communications Society and IEEE Information Theory Society Joint Paper Award in 2015, and the IEEE Information Theory Society Joint Paper Award in 2016. He received the Best Paper Award from the IEEE International Conference on Communications in 2014.

Saeedeh Parsaeefard (S’09–M’14) received the B.Sc. and M.Sc. degrees from the Amirkabir University of Technology (Tehran Polytechnic), Tehran, Iran, in 2003 and 2006, respectively, and the Ph.D. degree in electrical and computer engineering from Tarbiat Modares University, Tehran, in 2012. She was a Post-Doctoral Research Fellow with the Telecommunication and Signal Processing Laboratory, Department of Electrical and Computer Engineering, McGill University, Montréal, Canada. From 2010 to 2011, she was a Visiting Ph.D. Student with the Department of Electrical Engineering, University of California, Los Angeles, CA, USA. She is currently an Invited Associate Researcher with the Electrical and Computer Engineering Department, Carleton University, and a Faculty Member with the Iran Telecommunication Research Center. Her current research interests include multi-domain resource management in software-defined-based wireless networking, Internet-of-Things and fifth generation of wireless networks, as well as applications of robust optimization theory and game theory in wireless networks. She received the Women in Engineering Award from the IEEE Iran Section in 2018.

Babak Hossein Khalaj received the B.Sc. degree in electrical engineering from the Sharif University of Technology, Tehran, Iran, in 1989, and the M.Sc. and Ph.D. degrees in electrical engineering from Stanford University, Stanford, CA, USA, in 1993 and 1996, respectively. At Stanford University, he has been among the pioneering team focusing on the adoption of multi-antenna arrays in mobile networks. In 1995, he joined KLA-Tencor as a Senior Algorithm Designer, focusing on the advanced processing techniques for signal estimation. From 1996 to 1999, he was with a number of companies in Silicon Valley. Since 2000, he has been a Senior Consultant in the area of data communications, and a Visiting Professor with CEIT, San Sebastian, Spain, from 2006 to 2007. He has co-authored many papers in signal processing and digital communications and holds three U.S. patents. He was a recipient of the Alexander von Humboldt Fellowship from 2007 to 2008 and the Nokia Visiting Professor Fellowship in 2018. He was a Co-Editor of the Special Compatibility Standard Draft for the ANSIT1E1 Group from 1998 to 1999.