Optimizing Virtual Backup Allocation for Middleboxes

Yossi Kanizo, Ori Rottenstreich, Itai Segall, and Jose Yallouz

Abstract— **In enterprise networks, network functions, such as address translation, firewall, and deep packet inspection, are often implemented in middleboxes. Those can suffer from temporary unavailability due to misconfiguration or software and hardware malfunction. Traditionally, middlebox survivability is achieved by an expensive active-standby deployment where each middlebox has a backup instance, which is activated in case of a failure. Network function virtualization (NFV) is a novel networking paradigm allowing flexible, scalable and inexpensive implementation of network services. In this paper, we suggest a novel approach for planning and deploying backup schemes for network functions that guarantee high levels of survivability with significant reduction in resource consumption. In the suggested backup scheme, we take advantage of the flexibility and resource-sharing abilities of the NFV paradigm in order to maintain only a few backup servers, where each can serve one of multiple functions when corresponding middleboxes are unavailable. We describe different goals that network designers can consider when determining which functions to implement in each of the backup servers. We rely on a graph theoretical model to find properties of efficient assignments and to develop algorithms that can find them. Extensive experiments show, for example, that under realistic function failure probabilities, and reasonable capacity limitations, one can obtain 99.9% survival probability with half the number of servers, compared with standard techniques.**

Index Terms— **Network function virtualization, middleboxes, fault tolerance.**

I. Introduction

ENTERPRISE networks often rely on middleboxes to support a variety of network services such as Network Address Translation (NAT), load balancing, traffic shaping, firewall and Deep Packet Inspection (DPI). Typically, a middlebox is dedicated to implement solely a single function without a common standardization among different vendors, raising the complexity and cost of middlebox management.

Middlebox failures might occur frequently, inducing great degradation of service performance and reliability [2]. Failures might occur due to several reasons, such as connectivity errors (e.g., link flaps, device unreachability, port errors), hardware faults (memory errors, defective chassis), misconfiguration (wrong rule insertion, configuration conflicts), software faults (reboot, OS errors) or excessive resource utilization. Temporary inability to support network functions can lead to packet drops, reset of connections, redundant packet delays, and in some cases even to security issues. A common approach for achieving middlebox reliability is through redundancy, where an additional instance of each middlebox operates as a backup upon a failure of the first [3], [4]. This approach is often referred to as active-standby deployment, where one (the "active") instance handles all the traffic, while a second (the "standby") instance is mostly inactive, until a failure occurs. Once a failure is identified, all traffic is diverted to the standby instance, which has to immediately handle the functionality of the active one. While this approach enables simple dealing with a failure in each of the active middleboxes, it is often too restrictive and demands excessive redundancy in practice, especially since the frequency of failures can vary for different middleboxes, in addition to that usually not all the middleboxes encounter problems concurrently. Therefore, a milder and more flexible survivability approach is called for, which would relax the rigid requirement of an entire dedicated copy per active function.

Network Function Virtualization (NFV) [5] is a rising paradigm by which network functions are executed on commodity servers managed by a centralized software defined mechanism. Such virtual middleboxes are often also referred to as Virtual Network Functions (VNFs). By its elastic nature, NFV can considerably reduce operating expenses. Specifically, we show how the flexibility of NFV can be utilized for planning of efficient backup schemes for running functions, while better utilizing resources to support recovery from failures.

A significant cost reduction can be achieved through sharing of resources among the standby instances of the different functions. Implementing the active-standby approach in NFV would require holding a virtual machine (VM), or an otherwise dedicated set of resources, for the standby copy of each function. For generality, we refer to this VM or set of resources as *a server* in this paper. Note that this does not necessarily refer to a physical server. Our suggested backup solution consists of multiple such servers, with a typical number smaller than the number of existing middleboxes. The proposed solution relies on the observation that while each such server is equipped with the resources required to *run* a single function, acting as a standby copy requires less resources. Therefore, a server is assumed to have the capability to backup several functions,

Manuscript received October 25, 2016; revised March 8, 2017 and April 25, 2017; accepted May 3, 2017; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor X. Liu. Date of publication June 5, 2017; date of current version October 13, 2017. A preliminary version of this article appeared in the IEEE International Conference on Network Protocols (ICNP), Singapore, November 2016 [1]. *(Corresponding author: Ori Rottenstreich.)*

Y. Kanizo is with the Department of Computer Science, Tel-Hai Academic College, Upper Galilee 1220800, Israel (e-mail: ykanizo@telhai.ac.il).

O. Rottenstreich is with the Department of Computer Science, Princeton University, Princeton, NJ 08540 USA (e-mail: orir@cs.princeton.edu).

I. Segall is with Nokia Bell Labs, Kfar Saba 4464321, Israel (e-mail: itai.segall@nokia-bell-labs.com).

J. Yallouz is with the Department of Electrical Engineering, Technion, Haifa 32000, Israel (e-mail: jose@tx.technion.ac.il).

Digital Object Identifier 10.1109/TNET.2017.2703080

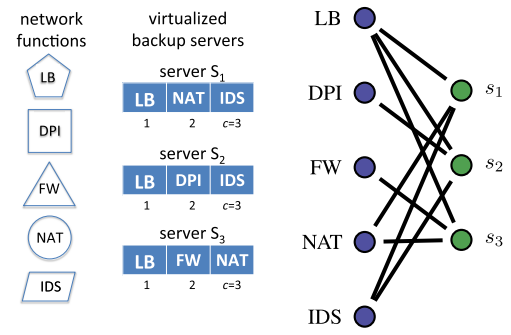
namely some specific selected subset of functions. As soon as a failure occurs, a server is turned into an active instance of the function of the failed middlebox, discarding the other backup capabilities of that server. Note that as a standby machine, the server can backup multiple functions. Once becoming active, only one of those functions can be served by this server. We show that while causing a very low penalty in terms of survivability, our suggested backup scheme enables a significant reduction in the amount of required resources in comparison with the conventional active-standby approach.

Many middlebox applications are stateful, i.e., maintain a state that changes over time and is required to enable correct operation. While in some cases the state is shared by all traffic (e.g., the rules in a firewall), in other scenarios it can be specified to every flow (e.g., automaton state in DPI) or shared among a subset of the traffic [6], [7]. In order to be able to backup a non-operating middlebox, a server must track the changes in the state of the middlebox, and thus has to be synchronized with such information frequently. It is expected that middlebox state will continue to increase in terms of diversity and complexity as middlebox vendors try to specialize their products. Thus, despite being in a standby mode, the number of functions that can be backed up by a server, staying in standby to replace any one of them, is limited and is expected to remain relatively low.

Network management under this aggregated standby mode should consider an important decision, namely the selection of the set of functions to be backed up in each of the servers. We demonstrate how this assignment affects the backup capabilities, and determines the possible combination of middlebox failures that can be resolved. Recall that in the case of two functions with faulty middleboxes, a single server that supports both functions cannot run more than one of them. In this work, we focus on aiming to optimize the backup scheme for maximum survivability under a given resource allowance. We detail several metrics describing goals that a network manager should take into account to achieve better survivability utilizing the constrained amount of resources.

Fig. 1(a) illustrates such a survivability architecture with five functions and three servers for backup. Each of the servers keeps the updated information required to recover three of the functions. In this example, for the load balancing function (LB), for instance, there is a backup in all the three servers and accordingly it can be recovered by any of them while for the deep packet inspection function (DPI) there is a backup in only one of the servers. Of course, information regarding the failure models and in particular their failure probabilities can be considered in the assignment of functions to servers to achieve higher survivability.

We investigate how to provide the best survivable solution supported by this approach of shared resources. To that end, we formalize two optimization problems describing possible goals that a network manager would often like to achieve in Section II. These problems consider the probability of full recovery from failures, as well as the expected number of functions that are available following a set of failures and the recovery process. In Section III and Section IV we establish fundamental properties of the problems while relying



(a) Architecture of $k = 5$ functions and $n = 3$ servers

(b) Bipartite graph representation. Left: network functions, right: Servers.

Fig. 1. (a) Illustration of architecture for middlebox survivability. A server can be used to backup one of $c = 3$ predetermined functions. To do so, the server must be frequently synchronized about the state of each of the functions it can backup. (b) Bipartite graph representation of the assignment of functions to servers. The degree of a server node is restricted to the capacity c.

on tools from graph theory. We describe characteristics of optimal solutions, and show bounds on the performance of optimal solutions. We show a family of instances for which the two problems collide, having the same set of optimal solutions but prove that the problems differ in the general case. We demonstrate that both problems are NP-hard by mapping different families of their instances to the known NP-hard number partitioning problem. Then, based on this analysis, we provide in Section V several algorithms to find efficient assignments. In Section VI we further discuss our findings about the problems. We study providing guarantees on the performance of the solutions and how to calculate the value of an assignment. In Section VII we demonstrate the advantages of our solutions through comprehensive simulations. Related work can be found in Section VIII. Finally, Section IX summarizes our results and discusses directions for future work. For space constraints, some proofs are omitted and can be found in [1].

II. Model and Problem Definition

We analyze this backup scheme through a graph framework. We denote the set of all functions by F and their number by $k = |F|$ such that $F = \{f_1, \ldots, f_k\}$. Likewise, let n be the number of servers and let c be the constrained capacity of a server, i.e., the maximal number of functions it can support. We refer to the set of servers by S satisfying $|S| = n$. We view an instance of the model as a bipartite graph $F \bigcup S$ with $k+n$ nodes. The assignment of functions to servers determines its edges. An edge between a server to a function illustrates the support of the function by the server. Fig. 1(b) shows the bipartite graph that corresponds to the assignment from Fig. 1(a). The capacity of the server limits its degree to at most c while the degree of a function equals the number of servers implementing it with a value in $[0, n]$.

A function $f_i \in F$ is associated with a failure probability p_i. We assume that these probabilities are mutually independent as well as that servers are always operational, i.e. do not

encounter failures. These probabilities determine the possibility for every set of functions to be the set of faulty functions.

Let $H \subseteq F$ be the set of functions with a failure. Given such a set of functions H, we are often interested in the induced graph of the set of faulty functions H, their outgoing edges and the connected set of servers. Operational functions do not have to be recovered. A matching M, representing a subset of the edges of the bipartite that corresponds to the assignment, describes the possibility to recover $|M|$ of the functions in H, such that a function can be recovered by its matched server. A matching of size $|M| = |H|$ would enable recovering all faulty functions.

Consider for instance the functions and assignment from Fig. 1. Assume that the functions LB, FW and NAT encounter a failure while the functions DPI and IDS are operational. In this case $H = \{\text{LB, FW, NAT}\} \subseteq F$ and $|H| = 3$. For such a set H we can find a matching M in the bipartite from Fig. 1(b) of the form $M = \{\text{LB} - s_2, \text{FW} - s_3, \text{NAT} - s_1\}$. Since $|M| = |H| = 3$ we can recover all the three failing functions such that a function is recovered by its matched server, e.g., the function LB by the server s_2. For a set of failing functions H for which a matching of size $|H|$ cannot be found, it is impossible to recover all the failing functions.

While managing the network, various goals considering survivability can be taken into account. We rely on the above probabilistic model, to quantify the performance of the backup mechanism. Following the distribution of the set of faulty functions we can examine the probability for specific events or concentrate on other characteristics calculated on expectation. Ideally, we would like to verify that all functions have an operational instance. This would enable to successfully serve all traffic, regardless of the network functions that a specific flow requires to apply. In interesting scenarios the number of backup servers is smaller than that of the functions and it is thus impossible to guarantee that all functions are operational or can be recovered. To that end, we would like to maximize the probability of the scenario that all functions have an operational instance, as expressed in the first optimization problem studied in this work.

Problem 1: Given are a set of functions F with function failure probabilities, a number of servers n, each with capacity c. The aim is to find an assignment that maximizes the probability for a recovery of all faulty functions. In such a recovery, each server is matched to at most one failed function, and each failed function is matched to one server.

We denote by P_{OPT} the optimal, maximal possible probability. Finding an optimal assignment that maximizes the probability for a recovery of all faulty functions is often very challenging. Moreover, in many cases, merely evaluating a solution, i.e. given a backup scheme – an assignment of functions to servers, calculating the probability for full recovery for this assignment can be a difficult task that requires considering a large number of combinations of faulty functions.

Sometimes there exists a benefit from a partial recovery of the faulty functions. In security applications for instance, availability of part of the security functions can increase the achievable level of defense. The Snort intrusion detection system [8] defines the notion of severity level where rules are categorized into levels such that examining a subset of them can be helpful. More generally, this partial recovery can be useful to serve all service chains that do not include the non-recovered functions. Similarly, a recent study [9] describes degrees of flexibility in the functions of a service chain, where a chain can be served with one among several functions or where the availability of some functions just contribute to improve the service quality or reduce the service time of service chains that can be completed with other functions. This motivates us in the definition of the following problem.

The next problem tries to maximize the number of available functions, i.e. the number of functions with operating middleboxes in addition to the maximal size set that can be recovered.

Problem 2: Given are a set of functions F with function failure probabilities, a number of servers n, each with capacity c. The aim is to find an assignment maximizing the expected number of operating functions.

This problem is equivalent to trying to maximize the expected number of functions that are recovered. The two metrics differ by a constant, equal to the expected number of operating functions without the recovery. We denote by E_{OPT} the optimal, maximal expected number of operating functions.

In both problems, finding a recovery is performed for a given status of the middleboxes, i.e. indication which of them is not operating. In other words, we assume that all functions fail at once. When the assignment of functions to servers is known, as well as the set of failed functions, a recovery of a maximal number of failed functions, expressed by a matching in the induced graph, can be easily found (in polynomial time) using algorithms for maximum matching in bipartite graphs (e.g., [10], [11]). In case a failing function with an implementation in more than a single server, the servers should agree in which of them the function should be recovered. This can be achieved through an identical implementation of the matching algorithm by the different servers. Accordingly, while as mentioned a frequent synchronization between a function and its implementing server is crucial, an additional synchronization between the servers, e.g. to agree on the server used for recovery, is not required.

Along most of the paper, we follow the assumption that a server cannot recover simultaneously more than a single function. As mentioned in the introduction, we assume enough resources in each server to recover a single function. We later provide experiments examining the benefit of being able to recover within the same server more than a single failing function to study cases when this assumption can be relaxed.

Our model has several additional simplifying assumptions, often required for the analysis. First, it does not consider a potential correlation between failures of different functions. In addition, it assumes that all functions require the same capacity when being backed up by a server. The model also assumes that the physical location of a function implementation does not affect the performance. We discuss a possible dependency between failure probabilities in Section VI-D and leave model generalizations for future work.

III. Problem 1: Full Recovery Probability

In this section we address Problem 1. We aim to find an assignment that maximizes the probability that all functions are operational or can be recovered. While the problem is rather easy when each server has a capacity of 1, we show that for a larger capacity the general problem is NP-hard.

For the set of k functions $F = \{f_1, \ldots, f_k\}$ let $\{p_1, p_2, \ldots, p_k\}$ be the respective function failure probabilities. W.l.o.g. assume functions have non-zero failure probabilities (a function that never fails does not require a backup) that are ordered in a non-decreasing order $p_1 \leq p_2 \leq \cdots \leq p_k$, i.e., the earlier functions are more reliable. Denote by p the product of those k probabilities. The set of n servers is denoted by $S = \{s_1, s_2, \ldots, s_n\}$, each with capacity c.

A simple observation is that it is impossible to guarantee a recovery of any combination of failed functions with less than $n = k$ servers. Due to the independence of functions, they can all fail together even if this scenario has a very small probability. It is also easy to guarantee that with at least k servers by dedicating a server to each function.

Corollary 1: The optimal full recovery probability satisfies $P_{OPT} < 1$ when the number of servers is $n < k$ and $P_{OPT} = 1$ when $n \geq k$.

We now turn to discuss the simple case of $c = 1$, where a server cannot backup more than one function.

A. Simple Case: $c = 1$

Theorem 2: In case $c = 1$, the optimal assignment of functions to servers is by assigning the n servers with the n functions with top failure probability, each server to a different function. Consequently, the optimal full recovery probability satisfies $P_{OPT} = \prod_{i=1}^{k-n}(1 - p_i)$, i.e. the probability that all first $k - n$ functions do not fail.

Proof: Once a function is recovered by some server, all other servers that are assigned to this function are useless (each server is assigned with only one function in our case). Therefore, all n servers should be assigned to distinct functions. Then, the first part of the theorem easily follows. Finally, full recovery is achieved if and only if none of the other $k - n$ functions fail, which yields the second part of the theorem. □

The simplicity of the case of $c = 1$ allows us to derive this result that does not hold for larger values of c. For such larger values, in the decision of an assignment it is required to determine not only which functions would have backups but also how many backups each function should have. Furthermore, there is also an importance for deciding about instances of what functions would be assigned to the same server.

B. Problem Hardness

We now focus on the case where $c = 2$. This case is much more complicated to analyze and to solve. In fact, a capacity of $c = 2$ suffices to show that the problem is NP-hard.

Our proof for the NP-hardness of this problem provides an intuition on how it can be solved approximately and inspired us in the design of the approach we take to tackle this problem later. The basic idea is to focus on the *connected components* of the underlying bipartite graph. In each connected component, nodes represent either servers or functions. To prove the following two properties we rely on [12]. The first property follows immediately from Lemma 1 there.

Property 3: In case $c = 2$, each connected component with t servers has at most $t + 1$ functions.

We also show the following.

Property 4: In case $c = 2$, let C be a connected component with t servers and $q \leq t + 1$ functions. Then for each subset of size (at most) $\min\{t, q\}$ of functions, there is a matching containing all of the functions.

Proof: If $q \leq t$ then $\min\{t, q\} = q$. Then, by [12, Lemma 2], the q functions have a matching of size $q = \min\{t, q\}$. Thus, any subset of the functions has also a matching (using the same edges as in the matching of size q). Otherwise, $q = t + 1$ and so $\min\{t, q\} = t$. Let $H \subseteq F$ of size $t = q - 1$. We now show it has a matching of size $|H| = t$ (and by the same argument as above any subset of H has a matching).

Let F_i be the only function in $F \setminus H$, and consider the subgraph induced by H and the server set. This subgraph contains t functions and t servers, and has one or more connected component. We show that for each such connected component the number of functions equals the number of servers. Assume on the contrary that some connected component has a number of function that is not equal to the number of servers, and consider a connected component for which the number of functions is larger than the number of servers. Such a connected component must exist since the number of functions in H equals the number of servers (in H).

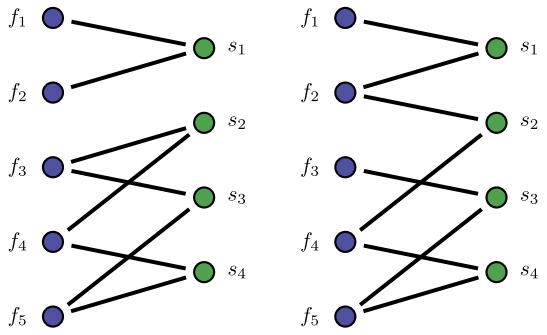
Let a be the number of servers in this connected component. By the connectivity of G, function F_i must be connected to one of those a servers. Thus, the number of functions connected to those a servers is at least $a + 2$. By Property 3 such a case cannot exist. Therefore, all connected components in H have a number of functions that is equal to the number of servers, and by [12, Lemma 2] have a perfect matching each, implying that the maximum matching size of H is t. □

The next theorem is easily followed:

Theorem 5: In case $c = 2$, and $n < k$, i.e., the number of servers is lower than the number of functions, each connected component in an optimal solution contains a number of functions that is larger by one than the number of servers.

Proof: Consider an optimal solution. Since $n < k$ there is at least one component with a number of servers smaller than the function number by exactly one. Difference cannot be larger by Property 3. We denote one such component by C_1. If the current theorem does not hold, there exists a second component C_2 with t servers and $q \leq t$ functions. We show that the two components can then be merged to increase the probability for recovery of all failing functions, in contradiction to the optimality of the solution.

By Property 4 all functions in C_1 can be recovered as long as not all of them fail together. We say then that the survival probability of the component is $1 - P_{C_1} = 1 - \prod_{i \in F_{C_1}} p_i$. Similarly, by the same property, we can always recover any set of failing functions from C_2, even if they all fail. Thus all functions in C_1, C_2 can be recovered with probability $1 - P_{C_1}$.



(a) non-optimal assignment I, 2 connected components (b) optimal assignment II, a single connected component

Fig. 2. Illustration of the structure of the bipartite graph in an optimal assignment for $c = 2$ (Theorem 5). Left: network functions. Right: Servers. Survival probability is improved when reducing the number of connected components as in a transition from (a) to (b) with the change of a one edge.

For an arbitrary spanning tree of C_2, consider an edge of the connected component that is not included in the tree. Since there are $c \cdot t = 2t$ edges, there must be such an edge not included in the at most $t+q-1 \leq 2t-1$ edges in the spanning tree. Omitting this edge does not eliminate the connectivity of C_2. We can therefore merge the two components C_1, C_2 by connecting one such edge to an arbitrary function in C_1, without changing the server it is connected to. Then, we obtain one connected component C instead of the two C_1, C_2, with t' servers and $q' \leq t'+1$ functions. Failures within its functions can be fully recovered with probability $1 - P_C = 1 - \prod_{i \in F_C} p_i = 1 - \prod_{i \in F_{C_1} \bigcup F_{C_2}} p_i > 1 - \prod_{i \in F_{C_1}} p_i$, i.e., larger than before the merge operation. Recovery probabilities for the other connected components are not affected by the change. Such a change is illustrated in Fig. 2 (for an arbitrary order of the functions) where in (a) there are two connected components C_1 with functions $\{f_1, f_2\}$ and a single server and C_2 with functions $\{f_3, f_4, f_5\}$ and three servers. In (b), by changing a single edge between s_2 to function f_2 instead of f_3, the number of connected components is reduced to one, improving the survival probability. □

By Theorem 5, the total number of functions equals the number of servers plus the number of connected components. This enables us to express the number of connected components in an optimal solution.

Theorem 6: In case $c = 2$, and $n < k$, the number of connected components in an optimal solution is $k - n$. This is the minimal possible number of components under these assumptions.

We now turn to compute the survival probability of a connected component.

Theorem 7: In case $c = 2$, and $n < k$. Let C be a connected component in an optimal assignment, and let $\{g_1, \ldots, g_q\}$ be the set of all function indices in C. Then the survival probability of the connected component is given by $1 - \prod_{i=1}^{q} p_{g_i}$.

Proof: By Theorem 5 we get that the number of functions in C is $t+1$ while the number of servers is t, for some $t \in \mathbb{N}$. Therefore, for each subset of functions of size t there is a matching in C (Property 4). Consequentially, the only case where not all functions can be recovered is the case where all of them fail. This happens with probability $\prod_{i=1}^{q} p_{g_i}$. The survival probability follows by taking the complement. □

Next, one may compute the overall survival probability, which is given by the product of the survival probability of all connected components. This follows immediately from the independence of function failure probabilities.

Theorem 8: The overall survival probability is given by the product of all connected component survival probabilities.

Note that the last theorem is valid also for arbitrary values of c, n, k. We begin to have a clear picture on how an optimal assignment looks like, as an intuition on how to split the connected components in the graph. Informally, we would want the connected components' survival probabilities to be balanced. We start with the special case where an optimal solution has two connected components.

Intuitively, there is a dependency between functions within the same connected component such that it might be impossible to recover one upon some scenario of others. Consider two functions i, j with failure probabilities $p_i > p_j$. Assume their backup should be selected from two available server slots, each translated to adding the function to one of two connected components C_1, C_2 with current failure probabilities $P_{C_1} > P_{C_2}$. Since function i fails more often, we wish to prioritize it. Thus it would be better to place it in the connected component with more reliable existing functions, i.e. in C_2. The other function, j, which encounters failures less frequently, will be placed in C_1 with existing functions that are more failure-prone. This addition contributes for balancing the two failure probabilities of the two connected components.

Theorem 9: In case $c = 2$, and $k = n + 2$, the optimal solution has exactly *two connected components. Among all possible assignments such that the bipartite graph has two connected components C_1 and C_2 with respective connected component survival probabilities $1 - P_{C_1}$ and $1 - P_{C_2}$, the optimal assignment is the one for which $\frac{\max\{P_{C_1}, P_{C_2}\}}{\min\{P_{C_1}, P_{C_2}\}}$ is minimized. In particular, if $P_{C_1} = P_{C_2}$ the assignment is optimal.*

First, since $k = n + 2$, by Theorem 6 the number of connected components is $k - n = 2$. Let A be an assignment that results in two connected components C_1 and C_2 with respective connected component survival probabilities $1 - P_{C_1}$ and $1 - P_{C_2}$. Also, let $r = \frac{\max\{P_{C_1}, P_{C_2}\}}{\min\{P_{C_1}, P_{C_2}\}}$. Assume that there exists an assignment A' with the corresponding C'_1, C'_2, $1 - P'_{C_1}$, $1 - P'_{C_2}$, and $r' = \frac{\max\{P'_{C_1}, P'_{C_2}\}}{\min\{P'_{C_1}, P'_{C_2}\}}$, such that $r' < r$. Since there exist exactly two connected components in each assignment, a function not included in one component must be included in the second and $C_1 \setminus C'_1 = C'_2 \setminus C_2$ and $C_2 \setminus C'_2 = C'_1 \setminus C_1$.

We will show that the survival probability of the assignment A' is larger than the corresponding probability of the assignment A, i.e. that $(1 - P'_{C_1})(1 - P'_{C_2}) > (1 - P_{C_1})(1 - P_{C_2})$.

To show this, we assume w.l.o.g. that $P_{C_1} \geq P_{C_2}$ and $P'_{C_1} \geq P'_{C_2}$. This implies that $\frac{P_{C_1}}{P_{C_2}} > \frac{P'_{C_1}}{P'_{C_2}}$.

We define $\alpha = P_{C_1 \cap C'_1}$ and $\beta = P_{C_2 \cap C'_2}$. In words, α is the product of probabilities of the functions that are assigned to the

first connected component in both solutions, and similarly β for the second component.

We further define $a = P_{C_1 \setminus C'_1} = P_{C'_2 \setminus C_2}$ and $b = P_{C_2 \setminus C'_2} = P_{C'_1 \setminus C_1}$. That is, a is the product of probabilities of the functions that are assigned to the first connected component in the solution A, but not to the first in A'. Note that $P_{C_1} = a \cdot \alpha$. Therefore, $P_{C'_2} = a \cdot \beta$. Symmetrically, $P_{C_2} = b \cdot \beta$ and $P_{C'_1} = b \cdot \alpha$.

$$\frac{P_{C_1}}{P_{C_2}} > \frac{P'_{C_1}}{P'_{C_2}} \Rightarrow \frac{a\alpha}{b\beta} > \frac{b\alpha}{a\beta} \Rightarrow a^2 > b^2 \Rightarrow a > b$$
$$P'_{C_1} \geq P'_{C_2} \Rightarrow b \cdot \alpha \geq a \cdot \beta > b \cdot \beta \Rightarrow \alpha > \beta$$

By combining both of the above, $a(\alpha - \beta) > b(\alpha - \beta)$

$$\Rightarrow a\alpha + b\beta > b\alpha + a\beta$$
$$\Rightarrow P_{C_1} + P_{C_2} > P_{C'_1} + P_{C'_2}$$
$$\Rightarrow 1 + p - P_{C'_1} - P_{C'_2} > 1 + p - P_{C_1} - P_{C_2}$$
$$\Rightarrow (1 - P'_{C_1})(1 - P'_{C_2}) > (1 - P_{C_1})(1 - P_{C_2}). \qquad \square$$

The last theorem demonstrated the necessity in increasing the balancing of the failure probabilities of connected components for maximizing the probability for a recovery of all functions. Intuitively, assume that we consider connecting a function (with its corresponding server) to one of the two connected components. Such an addition would reduce the failure probability of the selected component. A larger reduction is achieved for components with a larger failure probability. Similarly, a better balancing is achieved when the failure probability selected to be decreased is that of the component with a larger failure probability. The scenario this addition can make a difference is when the new function does not fail while all other functions in its connected component fail. The probability for that is maximized in a component with a larger failure probability.

By reducing the number partitioning problem [13] to our problem it is possible to show that the discussed problem is NP-hard. In the number partitioning problem, the goal is to divide a multiset of positive integers into two disjoint subsets while minimizing the absolute value of the difference of the subset sums.

Theorem 10: Finding an optimal assignment for Problem 1 is NP-hard (even for $c = 2$).

While many of the results described in this section were suggested for $c = 2$, they often do not apply for larger values of c. We can easily extend the more simple of them for such values. We can show, for instance, similarly to Property 3, that in a connected component with t servers there are at most $c + (t-1) \cdot (c-1) = t \cdot (c-1) + 1$ functions. However, it seems impossible to derive a similar complete analysis for such values. The reason is that while in the case of $c = 2$ we managed to express the survival probability of a connected component based only on its functions (Theorem 7), for $c \geq 3$ this cannot be done. Often the survival probability of a component is not well defined by its number of servers and its functions and can be influenced by the internal structure of the component.

C. Bounds

In this section we describe an upper bound on the value of the optimal full recovery probability P_{OPT} for the case of $c = 2$.

We start with the following simple lemma.

Lemma 1: For u_1, u_2, u_3, u_4 satisfying $0 < u_1 < u_2 \leq u_3 < u_4 < 1$ and $u_2 \cdot u_3 = u_1 \cdot u_4$. Then, $u_2 + u_3 < u_1 + u_4$.

We can also suggest the following upper bound.

Theorem 11: In case $c = 2$, the optimal full recovery probability satisfies $P_{OPT} \leq \left(1 - p^{\frac{1}{k-n}}\right)^{k-n}$, where $p = \prod_{i=1}^{k} p_i$ is the product of all function failure probabilities.

IV. Problem 2: Expected Number of Recovered Functions

Sometimes there can be a benefit from any available function that is running. In this section we study Problem 2, where we try to maximize the number of such functions, i.e. the functions with an operating middlebox together with the set of functions that can be recovered. We recall that we refer to the optimal (maximal) value of the expected number of such functions as E_{OPT}.

We can easily observe that E_{OPT} increases for larger number of servers n and that $E_{OPT} = k$ only for $n \geq k$ (assuming $p_i > 0$), i.e. when the number of servers equals at least the number of functions. A given assignment determines the connected components in the bipartite graph. Then, by the linearity of expectation the expected number of operating functions is given by the sum of their number in each connected component.

Problem 1 and Problem 2 are similar since in both problems it is more important to be able to recover a function that fails more frequently, i.e. with a larger failure probability. The difference comes from the fact that the first problem does not benefit from a partial recovery of the non-operating middleboxes. While we show that the problems coincide for some families of inputs we demonstrate that they differ in the general case.

A. Similarity of Problem 2 to Problem 1 and Its NP-Hardness

The two problems collide for the case $c = 2$ and $k = n + 2$ and their optimal solutions then coincide. This allows us to deduce that Problem 2 is also NP-hard. To show that we can rely on Property 3 and Property 4 that hold for general bipartite graphs (with $c = 2$).

Theorem 12: In a connected component C of t servers and $q = t + 1$ functions F_C, the expected number of operating functions in this component is $E_C = q - \prod_{i \in F_C} p_i$.

Similarly, Theorem 5 holds also for this problem. Also for this problem, it does not hold for $c > 2$.

Theorem 13: In case $c = 2$, and $n < k$, i.e., the number of servers is lower than the number of functions, each connected component in an optimal solution to Problem 2 contains a number of functions that is larger by one than the number of servers.

By Theorem 13, we get that also for Problem 2, if $c = 2$ and $n < k$ in an optimal solution the number of connected components is $k - n$.

It is possible to show that for $c = 2$, and $k = n + 2$, the optimal solutions of Problem 1 and Problem 2 coincide.

Theorem 14: In case $c = 2$, and $k = n + 2$, the set of (one or more) optimal solutions is the same in Problem 2 and Problem 1.

Finally, based on the above one can deduce that Problem 2 is also NP-hard.

Theorem 15: Finding an optimal assignment for Problem 2 is NP-hard (even for $c = 2$).

B. Bounds

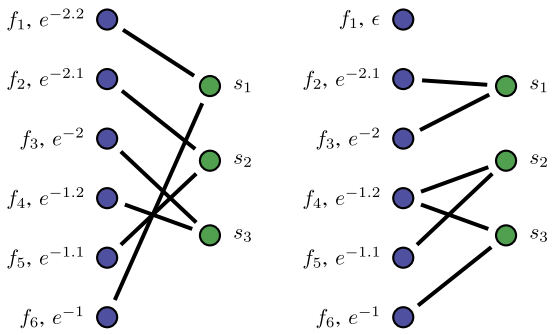
We can derive an upper bound on the expected number of available functions.

Theorem 16: In case $c = 2$, the optimal expected number of available functions satisfies $E_{OPT} \leq k - (k - n) \cdot p^{\frac{1}{k-n}}$, where p is the product of all function failure probabilities.

C. Between the Two Optimization Problems

Based on the similarity between Problem 1 and Problem 2 as we detailed, one might wonder whether the two problems are identical. It turns out that: a) the problems are very similar, and in the vast majority of cases, their optimum solutions coincide. This will be demonstrated further in Section VII, however: b) the problems are not identical. There exist cases where the optimal solutions for the two problems differ.

The next example demonstrates that the two problems differ from each other by showing how to find an instance where optimal assignments for the two problems vary. Let $k = 6$ and $n = 3, c = 2$, and first consider the failure probabilities be $p_1, \ldots, p_6 = e^{-2.2}, e^{-2.1}, e^{-2}, e^{-1.2}, e^{-1.1}, e^{-1}$. An optimal assignment to Problem 1 (assignment I) is with connected components $C_1 = \{f_1, f_6\}, C_2 = \{f_2, f_5\}, C_3 = \{f_3, f_4\}$ (achieved for servers implementing functions $\{f_1, f_6\}, \{f_2, f_5\}, \{f_3, f_4\}$). It achieves full recovery probability of $P_{OPT} = (1-p_1 \cdot p_6) \cdot (1-p_2 \cdot p_5) \cdot (1-p_3 \cdot p_4) = (1-e^{-3.2})^3 = (1-p^{\frac{1}{3}})^3$ for $p = \prod_{i=1}^{k} p_i$. The solution must be optimal since it achieves the upper bound from Theorem 11. This is also an optimal solution for Problem 2 achieving an expected number of $k-p_1 \cdot p_6-p_2 \cdot p_5-p_3 \cdot p_4 = 6-3 \cdot e^{-3.2} = k - (k - n) \cdot p^{\frac{1}{k-n}}$ available functions as in the upper bound from Theorem 16. If we set $p_1 = \epsilon$ (i.e., arbitrarily small), for both problems, there is no need to backup the first function and for both problems an optimal assignment (assignment II) has the connected components $C'_1 = \{f_1\}, C'_2 = \{f_2, f_3\}, C'_3 = \{f_4, f_5, f_6\}$ (achieved for instance for servers implementing functions $\{f_2, f_3\}, \{f_4, f_5\}, \{f_4, f_6\}$). Moreover, for $p_1 \in [\epsilon, e^{-2.2}]$ the transition value of p_1 between the two optimal solutions is different for the two problems. For Problem 1 we compare $(1 - p_1 \cdot p_6)(1 - p_2 \cdot p_5)(1 - p_3 \cdot p_4) = (1 - p_1)(1 - p_2 \cdot p_3)(1 - p_4 \cdot p_5 \cdot p_6)$ and get a transition point of $\frac{p_2 \cdot p_3 - p_2 \cdot p_5 - p_3 \cdot p_4 + p_4 \cdot p_5 \cdot p_6 + p_2 \cdot p_3 \cdot p_4 \cdot p_5 - p_2 \cdot p_3 \cdot p_4 \cdot p_5 \cdot p_6}{p_6 + p_2 \cdot p_3 - p_2 \cdot p_5 \cdot p_6 - p_3 \cdot p_4 \cdot p_6 + p_4 \cdot p_5 \cdot p_6 - 1} \approx 0.044390$. For Problem 2 we compare $6 - p_1 \cdot p_6 - p_2 \cdot p_5 - p_3 \cdot p_4 = 6 - p_1 - p_2 \cdot p_3 - p_4 \cdot p_5 \cdot p_6$ and get a transition point of $\frac{p_2 \cdot p_3 - p_2 \cdot p_5 - p_3 \cdot p_4 + p_4 \cdot p_5 \cdot p_6}{p_6 - 1} \approx 0.044404$. If p_1 is set to be between the two transition points, we have different optimal solutions for the two problems. The two assignments are illustrated in Fig. 3.



(a) optimal assignment I (b) assignment II, optimal for small ϵ

Fig. 3. Illustration of the distinction between Problem 1 and Problem 2 (Section IV-C). Left: network functions. Right: Servers. The failure probability appears for each function. When changing the failure probability of the first function p_1 from $e^{-2.2}$ to a small $\epsilon > 0$, the optimal assignments for each of the two problems changes from assignment I (in (a)) to assignment II (in (b)).

V. SOLUTIONS

We suggest various approaches to tackle the two problems. Due to the similarity between the problems, as discussed in Section IV-C, the approaches are agnostic to the problem at hand, and compute a solution that is expected to be effective for either problem. The heuristics presented in this section rely on the intuition developed in previous sections. Specifically, from Theorems 6 and 9 (resp.), we extract a general guideline to: a) reduce the number of connected components as much as possible, and b) keep the product of failure probabilities in the components as balanced as possible.

A. Simple Random Assignment (SRA)

The first approach is mainly used as a reference to the algorithms described in Section V-B and Section V-C.

In the SRA algorithm, we pick, randomly and independently, for each server c *different* functions that it backups, where the probability to choose any function is proportional to the specific function failure probability. The exact steps are described below:

- Compute the proportional probability vector defined by $\{p_1, p_2, \ldots, p_k\} / \sum p_i$.
- For each server $s_j, j \in [1, n]$:
  - Let $v_j = v$.
  - For each of the c backup slots: Randomly and independently pick a function using the distribution defined in v_j. Update v_j by removing the function that was picked and normalize back to a sum of 1.

B. Constrained Random Assignment (CRA)

Based on the intuitions obtained above we perform a slight modification to the SRA algorithm where the effect of the value of the failure probability of a function on the number of instances is moderate.

The basic idea is to decide in advance how many backups each function has, and then distribute the backups to the servers. The number of backups should be as even as

possible, with a maximal difference of one. The exact steps are described below:

- Decide on the number of backups b_i for function $f_i \in F$. Start with $\lfloor n \cdot c/k \rfloor$ backups for each one. Add an extra backup to the functions with the top $(n \cdot c) \mod k$ failure probabilities.
- For each server $s_j \in S$, select randomly c among the $n \cdot c$ instances, avoiding multiple selections of the same function by the same server.

C. Balanced Assignment (BA)

The following approach is also based upon the intuitions from the analysis. It consists of three phases:

- Decide on the number of connected components. As was shown, a crucial aspect in the effectiveness of a backup scheme is the partitioning of the corresponding bipartite graph into connected components. Therefore, in the first phase we choose the number of such components to partition the graph into.
- Partition the functions into the connected components. As discussed, the aim is to partition the functions such that the product of failure probabilities for functions in each component is balanced.
- Construct the actual backup scheme, by assigning functions to servers.

The purpose of the algorithm is to construct a backup scheme that keeps the number of connected components in the corresponding bipartite graph small, and is balanced in terms of products of function probabilities – both between connected components, and between servers within a component. Its details are given below.

1) Deciding on the Number of Connected Components: Generalizing on Theorem 6, we wish to minimize the number of connected components. Therefore, given k functions, and n servers with capacity c, we choose to construct $t = \max(1, k-(c-1)\cdot n)$ connected components.

2) Partitioning the Functions Into the Connected Components: Generalizing on Theorem 9, the aim of the partitioning is to balance as much as possible the product of failure probabilities of functions in each partition. We choose a greedy approach to this problem. First, functions are sorted in non-decreasing order of their failure probability. Then, each function is assigned to the partition with the largest product of failure probabilities so far.

In order to allow a mapping of the functions in each partition to servers, that is consistent with the choice of t above, we require the size of each partition to be $(c-1)\cdot w+1$ for some integer w. If in the result of the greedy algorithm this is not the case, we shift functions between partitions. In order for this to affect the balance between the partitions the least, we choose to shift the functions with the highest failure probability.

3) Constructing Connected Components From the Partitions: Each set of functions partitioned together in the previous step should now get a set of servers, and mapping between the functions and servers, such that a connected component is formed in the corresponding bipartite graph. Clearly, the construction of each component is independent of the other components, so we focus on a single given component.

Assuming the connected component consists of f functions, let $z = \frac{f-1}{c-1}$ be the minimal number of servers required in order to construct a connected component for these functions. We re-partition the f functions into z partitions, under the constraint that no partition contains more than c functions, using the same greedy heuristics as before. Each such partition is then mapped to a single server. For each remaining backup slots, i.e. for each server that received less than c functions, we connect the functions with the maximal failure probability from the previous partitions until c functions are backed up. This guarantees the connectivity within the component.

4) Assigning Functions to Remaining Servers: For the case where all functions are grouped into a single connected component, i.e. when $k-(c-1)\cdot n \leq 1$, the number of servers, z, in the previous step, might be smaller than n. Then, we have servers that were not assigned to any function at this point. Leaving unused servers is not optimal, so to those servers not in use, we assign functions in decreasing order of failure probabilities. We skip the functions with highest probability, as they were already assigned an additional backup when connecting the component in the previous step.

VI. Discussion

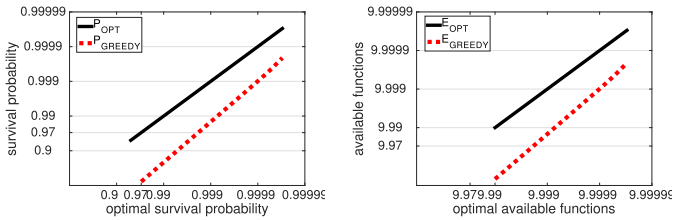
A. Approximation Guarantees

We notice that the balancing algorithm, presented in Section V-C above is a generalization of a greedy algorithm. For the special case of $c = 2$ and $k-n = 2$, where exactly two connected components are constructed, the algorithm is in fact the simple greedy algorithm that constructs two partitions by sorting the functions in non-decreasing order of their failure probability, and assigning each function to the partition with the largest product of failure probabilities so far.

This algorithm is equivalent to applying the well-known greedy algorithm for number partitioning, on inputs equal to minus the log of failure probabilities, similar to the reduction in Theorem 10. This algorithm is known to achieve $7/6$ approximation of the size of the larger partition [14]. One might ask how that known approximation ratio translates into guarantees on the effectiveness of the balancing algorithm, at least for this special case of $c = 2$ and $k - n = 2$. Unfortunately, the result does not imply a fixed approximation ratio, but enables us to derive bounds on the performance of the greedy algorithm that vary for different instances.

Assume an instance with $p = \prod_{i=1}^{k} p_i$ for failure probabilities $p_1, \ldots, p_k$. Let $P^o_{C_1}, P^o_{C_2}$ be the product of failure probabilities of functions in the two connected components in an optimal solution such that $P_{OPT} = (1-P^o_{C_1})\cdot(1-P^o_{C_2})$ is the optimal survival probability. We derive P_{GREEDY}, a lower bound on the survival probability of the greedy solution.

We denote P_{C_1}, P_{C_2} the products corresponding to the two connected components in the greedy solution. We assume w.l.o.g. $P^o_{C_1} \geq P^o_{C_2}$ and $P_{C_1} \geq P_{C_2}$. They satisfy $P^o_{C_1} \cdot P^o_{C_2} = P_{C_1} \cdot P_{C_2} = p$. The $7/6$ approximation ratio on the size of the larger partition in number partitioning gives us the following relation between $P^o_{C_2}$ and P_{C_2}: $P_{C_2} \geq (P^o_{C_2})^{7/6}$.



(a) Problem 1 (b) Problem 2

Fig. 4. Guarantees on the results of the greedy algorithm based on the optimal values. (a) Problem 1 with $p = 10^{-10}$, (b) Problem 2 with $p = 10^{-10}$, $k = 10$.

Here $P^o_{C_2} < 1$ and P_{C_2} satisfies $(P^o_{C_2})^{7/6} \leq P_{C_2} \leq P^o_{C_2}$. For a given value of P_{OPT} we solve the two equations $(1 - P^o_{C_1}) \cdot (1 - P^o_{C_2}) = P_{OPT}$, $P^o_{C_1} \cdot P^o_{C_2} = p$ to derive $P^o_{C_2}$. The minimal survival probability is obtained by the greedy algorithm when the lower bound on P_{C_2} is tight and $P_{C_2} = (P^o_{C_2})^{7/6}$. Then we can calculate $P_{C_1} = p/P_{C_2}$ and $P_{GREEDY} = (1 - P_{C_1}) \cdot (1 - P_{C_2})$ as a function of p, P_{OPT}.

In Fig. 4(a) we demonstrate this by setting a constant $p = 10^{-10}$, and leaving P_{OPT}, the survival probability of the optimal solution, as an independent variable. We plot P_{GREEDY} as a function of P_{OPT} for this case, as well as the identity function of P_{OPT} for comparison. Several points can be observed:

- For low values of P_{OPT}, below roughly 0.962724 in this case, the value of P_{GREEDY} is negative (hence not plotted in this figure). This means that the approximation is too loose to be of any value.
- Above a certain value of P_{OPT}, there is no real solution for $P^o_{C_2}$, i.e., no real inputs p_i exist for this case. In our example with $p = 10^{-10}$, no inputs exist for which $P_{OPT} > 1 + p - 2\sqrt{p} = 1 + 10^{-10} - 2 \cdot 10^{-5} \approx 0.99998$.
- As P_{OPT} approaches its highest possible values, the ratio between the bound on the probability of the greedy and the optimal one is improved (i.e., larger). For this example with $p = 10^{-10}$, the ratio is 0.9863 for $P_{OPT} = 0.999$ and 0.9998 for $P_{OPT} = 0.99997$.

For Problem 2, very similar derivations apply. Here we need to assume as given, in addition to p and E_{OPT}, also k, the number of functions. We start from $E_{OPT} = k - P^o_{C_1} - P^o_{C_2}$, assume worst case values for P_{C_2} and derive a formula for E_{GREEDY} as a function of p, E_{OPT} and k. Fig. 4(b) plots this for $p = 10^{-10}, k = 10$.

B. Calculating the Value of an Assignment

The number of possible assignments can be large, i.e. exponential in the number of servers and their capacity c. Moreover, for both problems calculating the value of a given assignment, i.e. the survival probability in Problem 1 or the expected number of available functions in Problem 2 is not an easy task. The number of scenarios with different set of failing functions that have to be taken into account is 2^k, i.e., exponential in the number of functions. This often restricts the number of functions we can consider in the experimental settings. We develop dynamic-programming based techniques to accelerate this calculation.

Our approach relies on the so called *defect* version of Hall's theorem [15], also known as Ore deficiency theorem [16]. Given a bipartite graph G with bipartation (U, V), the deficiency of the graph G, is given by

$$\text{def}\,(G) = \max_{A \subseteq U} \{|A| - |D\,(A)|\}\,,$$

where the operator $D\,(A)$ extracts the set of neighbors of all vertices in A. The defect version of Hall's theorem states that the maximum matching size of the graph G is exactly $|U| - \text{def}\,(G)$.

We go over all function subsets ordered by their subset size, maintaining the deficiency of the graph induced by each subset, the probability that this subset of functions fail (and all other do not fail), and the set of neighbors. An update of the deficiency of a subset A, with $|A| = a$, is performed by computing the maximum deficiency of all subsets of A of size $a - 1$, and a minus the number of adjacent servers of A.

Since the maximum matching size of the graph induced by any subset can be obtained by the collected data, we avoid computing the maximum matching size independently for each subset. Moreover, we are able to compute both the survival probability and expectation in the same (single) pass.

Note that, in light of these considerations, when the number of functions is large (e.g., over 20) we are not able to exactly compute the overall survival probability and expected number of operating functions.

C. Estimating Failure Probabilities

Our approach assumes that the failure probabilities of the middleboxes are known. Of course, this information is not always easy to obtain. Two recent studies [2] and [17] examined the number of failures within a period of a year and seven years, respectively, as well as the time it takes to repair them for middleboxes and cloud services, respectively. We can use them as a guidelines to have some intuition on typical numbers. In [2], a mean number of 1.5-3.5 failures/year is described for 4 types of middleboxes such that roughly half of the failures are fixed within an hour but the 95th and the 99th percentiles are much longer and equal about 2.5 and 17.5 days, respectively. Similar results can be derived from [17], while considering the outage of popular virtual cloud services. Here, a median of 3 failures/year is showed for all measured outage services and the annual downtime of 95% of the cloud services is upper bounded by roughly 3 days. Following the similarity of the two studies, we obtain the results of the first [2] for deriving the failure probability values we examine in the experimental settings selecting them uniformly over the range $[0.025, 0.175]$ where $0.025 \approx \frac{3.5 \cdot 2.5}{365}, 0.175 \approx \frac{3.5 \cdot 17.5}{365}$.

D. Dependent Failure Probabilities

Our model assumes that a function f_i has a failure probability p_i such that there is no dependency in the failures of different functions. One might wonder how our approach should take care of such dependency if exists. We provide intuitive ideas for such a scenario.

While considering two binary variables Y_i, Y_j that describe failure events for functions i, j we have $E(Y_i) = p_i, E(Y_j) = p_j$. Their dependency can be described through the notion of the covariance $\text{cov}(Y_i, Y_j) = E(Y_i \cdot Y_j) - E(Y_i) \cdot E(Y_j)$. Consider an extreme case, where $p_i = p_j$ and a pair of functions either operate or fail together. Thus $\text{cov}(Y_i, Y_j) = p_i - p_i^2 > 0$. Assume that each function has a single backup and they are located on the same server. In case they both fail, there is no option to recover both functions by the same server, thus there is no option for a full recovery in such an assignment. Accordingly, for Problem 1, these backup slots do not contribute to the target function. It would have been more beneficial to locate these copies in distinct servers. On the contrary, when the functions cannot fall together (and $\text{cov}(Y_i, Y_j) < 0$), it can be beneficial to locate them in the same server. On intermediate situations, we suggest to consider their dependency to impact the probability for the functions to be located together.

VII. Experimental Results

We now turn to evaluating our algorithms described in Section V. We first compare the different algorithms in Section VII-A, and then in Section VII-B we show the effect of the number of backups per server, c, on the number of servers required to achieve a target survival probability. The simulation is based on the technique described in Section VI-B that computes the overall survival probability and expected number of available functions for a given backup assignment. We compare between this greedy algorithm performance, the optimal values, and its guarantees.

According to [18], the number of middleboxes is often equivalent to the number of routers in a network. Therefore, in our experiment setup we assume a medium size network containing 20 middleboxes functions, i.e. $k = 20$. Nevertheless, we can expect the same behavior for larger networks. The server capacity c is obtained from the results in Section VII-B.

A. Competitive Evaluation

We first compare the algorithms proposed in Section V.

For the ease of presentation we define the *overall failure probability* as the probability that at least one function cannot be recovered, i.e., the overall failure probability equals one minus the survival probability. Likewise, we consider in this section the *unavailability expectation* defined as the expected number of unavailable functions. Since both new definitions complement the original ones, our goal is to minimize those quantities.

Fig. 5 and Fig. 6 show the overall failure probability and the unavailability expectation, as a function of the number of servers, with $c = 4$, and $c = 5$, respectively. We set $k = 20$ in both cases and compared the algorithms as n, the number of servers, grows. To cope with the fact that the algorithms performance depends on the actual input, and that some of the algorithms are random, we rely on average case analysis. Hence, for each $n \in \{0, 1, \ldots, k-1\}$, we created 500 random instances of failure probabilities vectors, where each failure probability is uniformly distributed over the range $[0.025, 0.175]$. This range was selected by the considerations presented in Section VI-C.

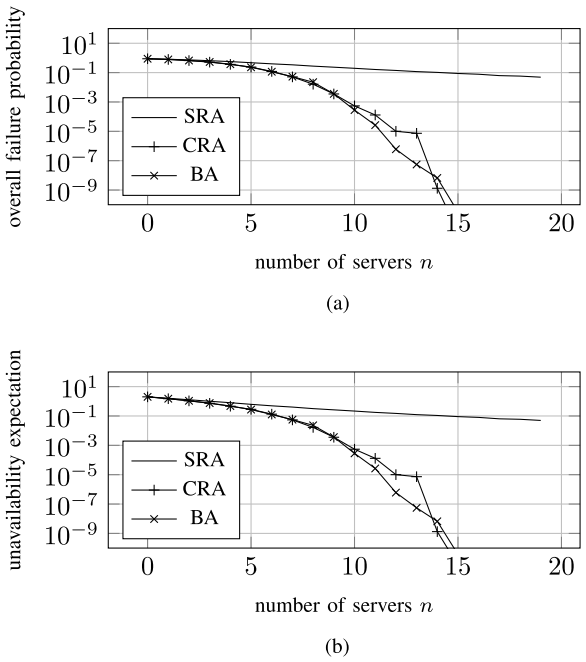


Fig. 5. Performance comparison of the algorithms presented in Section V with $c = 4$ and $k = 20$. (a) overall failure probability. (b) overall expected number of unavailable functions.

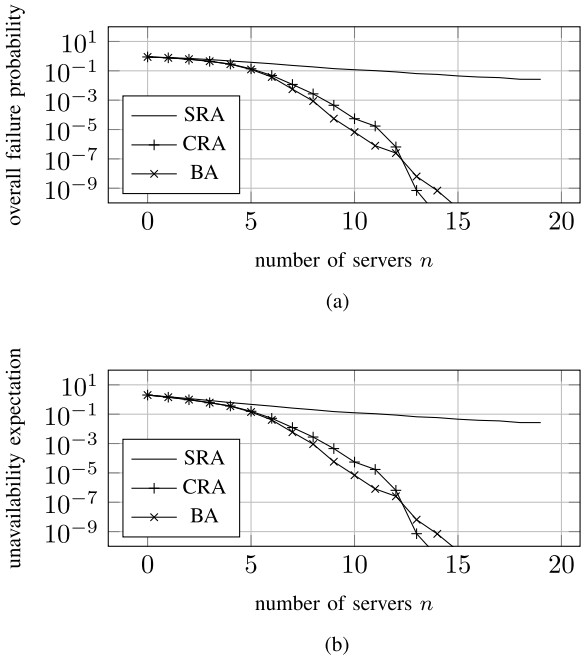


Fig. 6. Performance comparison of the algorithms presented in Section V with $c = 5$ and $k = 20$. (a) overall failure probability. (b) overall expected number of unavailable functions.

As depicted in Fig. 5 and Fig. 6, the CRA and the BA algorithms outperform the SRA algorithm for all values of n. While their performance seems similar for small values of n, the CRA algorithm is slightly better than the BA algorithm in this case. This is due to the fact that the BA algorithm balances the connected component probability based on their failure probabilities product, which is suitable for $c = 2$, while the CRA algorithm basically assigns backups to the

TABLE I

COMPARISON OF CRA AND BA FOR $c = 4$. THE TABLE SHOWS THE OVERALL FAILURE PROBABILITY (PROB.) AND THE UNAVAILABILITY EXPECTATION (EXP.)

n	CRA prob.	BA prob.	CRA exp.	BA exp.
9	$3.51 \cdot 10^{-3}$	$3.56 \cdot 10^{-3}$	$3.56 \cdot 10^{-3}$	$3.64 \cdot 10^{-3}$
10	$5.41 \cdot 10^{-4}$	$2.8 \cdot 10^{-4}$	$5.45 \cdot 10^{-4}$	$2.84 \cdot 10^{-4}$
11	$1.3 \cdot 10^{-4}$	$2.64 \cdot 10^{-5}$	$1.3 \cdot 10^{-4}$	$2.66 \cdot 10^{-5}$
12	$1.01 \cdot 10^{-5}$	$6 \cdot 10^{-7}$	$1.01 \cdot 10^{-5}$	$6.03 \cdot 10^{-7}$
13	$7.32 \cdot 10^{-6}$	$5.48 \cdot 10^{-8}$	$7.32 \cdot 10^{-6}$	$5.51 \cdot 10^{-8}$

TABLE II

COMPARISON OF CRA AND BA FOR $c = 5$. THE TABLE SHOWS THE OVERALL FAILURE PROBABILITY (PROB.) AND THE UNAVAILABILITY EXPECTATION (EXP.)

n	CRA prob.	BA prob.	CRA exp.	BA exp.
7	$1.18 \cdot 10^{-2}$	$5.59 \cdot 10^{-3}$	$5.41 \cdot 10^{-2}$	$5.6 \cdot 10^{-2}$
8	$2.69 \cdot 10^{-3}$	$8.68 \cdot 10^{-4}$	$1.67 \cdot 10^{-2}$	$2.3 \cdot 10^{-2}$
9	$4.48 \cdot 10^{-4}$	$5.5 \cdot 10^{-5}$	$3.56 \cdot 10^{-3}$	$3.64 \cdot 10^{-3}$
10	$5.46 \cdot 10^{-5}$	$6.69 \cdot 10^{-6}$	$5.45 \cdot 10^{-4}$	$2.84 \cdot 10^{-4}$
11	$1.72 \cdot 10^{-5}$	$7.85 \cdot 10^{-7}$	$1.3 \cdot 10^{-4}$	$2.66 \cdot 10^{-5}$

functions with the top most $c \cdot n$ failure probabilities (for $k \geq c \cdot n$). However, this range of values for the number of servers n is not of much interest since it typically results in too high overall failure probability and unavailability expectation. Note that if one is interested in extremely low overall failure probability (and unavailability expectation), then the CRA algorithm shows better results also for this case, where the number of servers is large.

Note also the similarity between the results for the two problems, both in Fig. 5 and in Fig. 6 (between the two sub-figures in each figure). This demonstrates the extreme similarity of the two problems, as discussed in Section IV-C.

Table I and Table II[1] distinguish between the overall failure probability and unavailability expectation of the CRA and BA algorithms for interesting values of n, when the overall failure probability and unavailability expectation obtained values are reasonable. The BA algorithm shows better results for all such values of n. As can be seen in the two parts of the tables, this is true for both problems.

Moreover, Tables III and IV show the standard deviation of the overall failure probability. The quality of the BA algorithm is demonstrated in a two-folded way. First, it shows better results for all (reasonable) values of n. Second, as Tables III and IV show, the CRA algorithm encounters a relatively higher standard deviation in results. A low-standard deviation in results is crucial when the number of functions is large and it may be hard to evaluate the exact survival probability and expectation, as discussed in Section VI-B.

B. Effect of the Number of Backups per Server

We now examine the effect of the number of backups per server c. Fig. 7 shows the number of servers required in order to achieve a survival probability of 0.999 and 0.9999 as a function of the number of backups per server, c, where

[1]For the journal version of this paper, we re-ran our simulations, increasing the number of instances, and fixed some minor bugs in the computer program. These modifications only slightly change the results reported in the conference version experiment section, where none of these changes is significant.

TABLE III

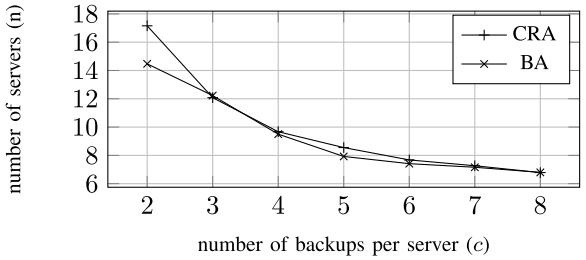
STANDARD DEVIATION OF THE SURVIVAL PROBABILITY P WITH $c = 4$

n	CRA prob. std	BA prob. std
9	$2.5 \cdot 10^{-3}$	$1.69 \cdot 10^{-3}$
10	$1.28 \cdot 10^{-3}$	$1.59 \cdot 10^{-4}$
11	$6.39 \cdot 10^{-4}$	$1.89 \cdot 10^{-5}$
12	$6.35 \cdot 10^{-5}$	$5.58 \cdot 10^{-7}$
13	$1.16 \cdot 10^{-4}$	$1.26 \cdot 10^{-7}$

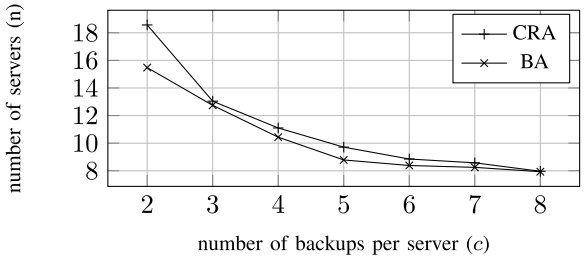
TABLE IV

STANDARD DEVIATION OF THE SURVIVAL PROBABILITY P WITH $c = 5$

n	CRA prob. std	BA prob. std
7	$5.7 \cdot 10^{-3}$	$2.63 \cdot 10^{-3}$
8	$3.31 \cdot 10^{-3}$	$4.54 \cdot 10^{-4}$
9	$1.62 \cdot 10^{-3}$	$4.28 \cdot 10^{-5}$
10	$4.06 \cdot 10^{-4}$	$5.86 \cdot 10^{-6}$
11	$3.33 \cdot 10^{-4}$	$7.56 \cdot 10^{-7}$



(a)



(b)

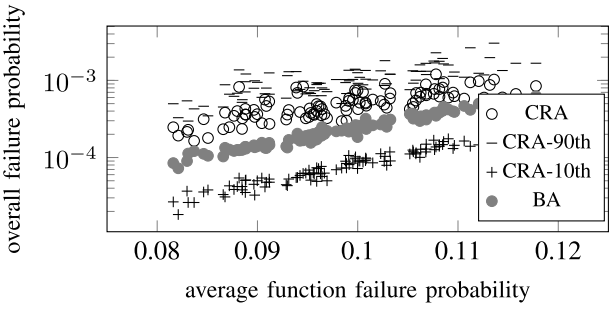
Fig. 7. Number of servers needed to obtain a desired survival probability P_d as a function of c, with $k = 20$ functions.(a) $P_d = 0.999$. (b) $P_d = 0.9999$.

$k = 20$. We used the CRA and BA algorithms which showed best results, and ran our simulations for each c and k with 500 random instances of failure probabilities vectors, distributed as in Section VII-A.

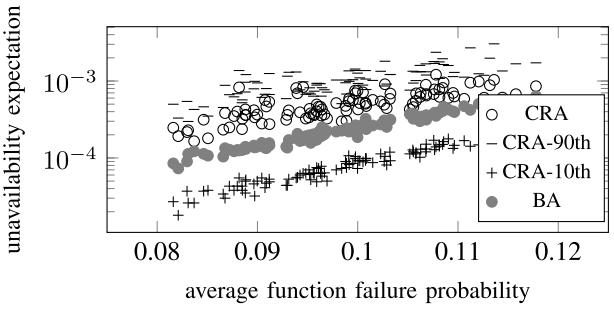
For example, Fig. 7 demonstrates that for 20 functions (i.e. $k = 20$), a survival probability of 0.9999 requires approximately 11 servers, each capable of backing up 4 functions (i.e. $c = 4$) using CRA, or only approximately 10 servers using BA. If each server can have 5 backups then we can save one server for each case, resulting in approximately 10 and 9 servers, respectively. From Fig. 7, we can conclude that most of the improvement in the number of required servers is achieved for $c = 4, 5$ which justifies the selection of these values in the rest of the simulations.

C. Between CRA and BA

In section VII-A we showed that while the BA algorithm only slightly improves upon the CRA algorithm, CRA shows large variance compared to BA. In this section we show that

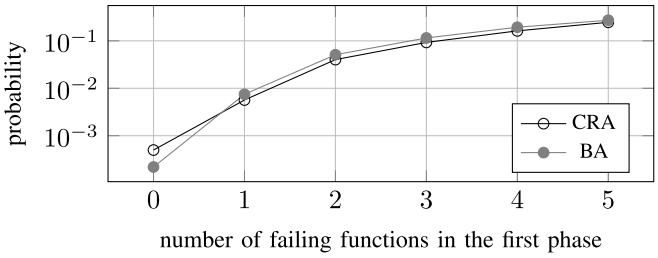


(a)

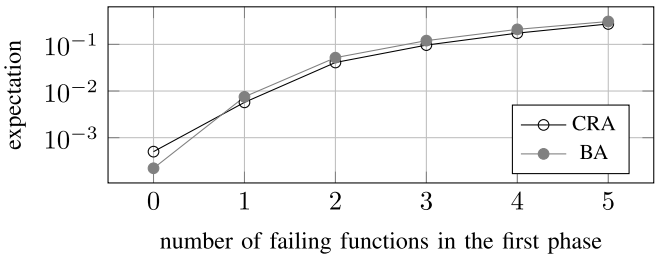


(b)

Fig. 8. Performance comparison of the CRA and BA algorithms with $c = 4$ and $n = 10$ and $k = 20$. (a) overall failure probability. (b) overall expected number of unavailable functions.



(a)



(b)

Fig. 9. Overall failure probability and unavailability expectation in a two-phases failure scenario, with $c = 4$ and $n = 10$ and $k = 20$. (a) overall failure probability. (b) overall expected number of unavailable functions.

the large variance of CRA can be exploited in cases where the *exact* overall survival probabilities and the unavailability expectation can be computed, i.e., where the number of functions is small.

Fig. 8(a) and Fig. 8(b) show statistics of the overall failure probability and the unavailability expectation as a function of the average function failure probability. We set $k = 20$, $c = 4$, $n = 10$, and created 100 random instances of failure probabilities vectors, where each failure probability is uniformly distributed over the range $[0.025, 0.175]$ (as in Section VII-A). For each such vector, we ran 100 times the CRA algorithm, gathering its 10th percentile, 90th percentile, and its mean (for both the overall survival probability, and the unavailability expectation), then we ran BA once. Since BA is a deterministic algorithm, no statistics gathering required (or possible). Note that Fig. 8(a) and Fig. 8(b) show the complement of the above quantities, that is, the overall failure probability and the unavailability expectation. Therefore, the 10-th percentile appears lower than the 90-th percentile.

As both Fig. 8(a) and Fig. 8(b) show, if the number of functions is low so that exact overall survival probability and unavailability expectation can be computed, one may run the CRA algorithm multiple times, saving the construction at each time, and then taking the construction which led to the best results. This process makes CRA perform better than BA. However, when the number of functions is larger, assuming that the difference between CRA mean performance and BA performance is also valid, BA should be favored.

D. Post Failure Recovery Probability

In this section we consider a scenario where the failures of the functions occur in two phases. In the first phase some functions fail and then recovered, that is, specific servers are used to recover the functions, where the assignment of function backups to other servers is *unchanged*. Then, in the second phase, other functions may fail. These functions can be recovered only by servers that were not used for recovery in the first phase.

Fig. 9(a) and Fig. 9(b) show the overall failure probability and the unavailability expectation at the start of the second phase (that is, after the first-phase failure and recovery), as a function of the number of functions that fail in the first phase. As in previous subsections, we set $k = 20$, $c = 4$, $n = 10$, and created 500 random instances of failure probabilities vectors, where each failure probability is uniformly distributed over the range $[0.025, 0.175]$. For each such vector and for each $q \in \{0, \ldots, 5\}$, we picked q functions uniformly at random, considered them as the first phase failing functions and recovered them using servers according to the specific assignments retrieved by CRA and BA. Then, we determined the overall failure probability and the unavailability expectation in each of the assignments (when disregarding the q picked functions and the q corresponding servers used for recovery).

As both Fig. 9(a) and Fig. 9(b) show, the overall failure probability and the unavailability expectation in the second phase gets significantly larger when the number of functions that fail in the first phase increases. For instance, while for both CRA and BA the overall failure probability and the unavailability expectation is below 10^{-3}, for $q = 3$ the corresponding values are approximately 0.1.

E. Servers Recovering Multiple Functions

Until this point we focused on the case where each server is capable of recovering exactly one function upon its failure, here we consider the case where each server is capable of recovering up to r functions. In this case, we basically use

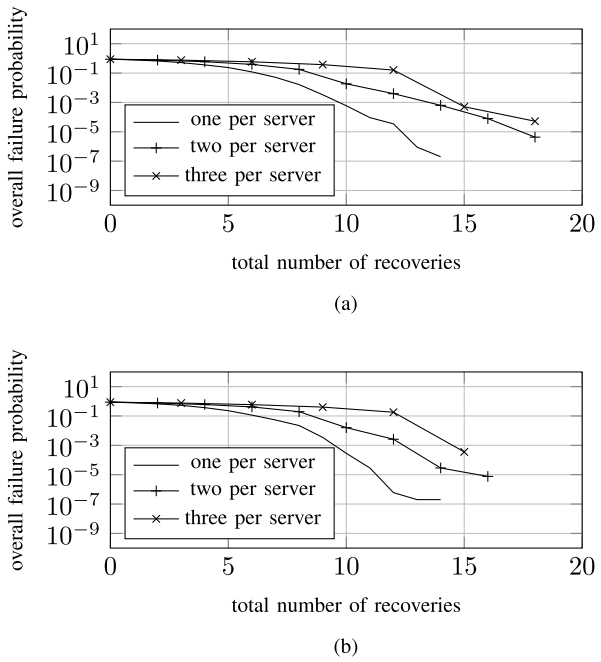


Fig. 10. Overall failure probability of CRA an BA when each server is capable of recovering more than one function with $c = 4$ and $k = 20$. (a) CRA. (b) BA.

the exact same solutions proposed for the single recovery per server case.

Fig. 10(a) and Fig 10(b) show, for both CRA and BA, respectively, the overall failure probability as the function of total number of recoveries the system is capable of, that is, the number of servers multiplied by the number of recoveries per server $n \cdot r$. We set $k = 20$ functions, and for each $n \in \{0, 1, \ldots, k-1\}$, we created 500 random instances of failure probabilities vectors, where each failure probability is uniformly distributed over the range $[0.025, 0.175]$. Then for each such vector we applied CRA and BA and determined their overall failure probability for $r \in \{1, 2, 3\}$ recoveries per server. Note that for the sake of fair comparison, we plot the results as a function of the total number of recoveries. As expected, since having n servers with $r > 1$ recoveries per server instead of just $n \cdot r$ servers results in loss of freedom in assigning function backups to servers, we witness a degradation in performance in the former case. However, having high performance servers capable of recovering $r > 1$ failing functions dramatically reduces the number of servers needed to achieve reasonable overall failure probability. For instance, using 5 servers only with $r = 3$ recoveries per servers suffices to achieve an overall failure probability lower than 10^{-3}, where for the case where $r = 1$, 10 servers are needed.

VIII. Related Work

Various studies have considered the design and implementation of high availability systems [3], [7], [19], [20]. Remus [3] and Colo [19] are general approaches offering fault tolerance for any VM-based system running a standard operating system.

Remus is based on checkpoint strategy, recovering a failed system to the last saved state, while Colo is also based on parallel execution of identical VMs. Pico [20] was the first fault tolerance designated system for middleboxes. Its maintenance operates at the flow level state enabling lighter checkpoints. In [7], Fault-Tolerant middlebox system for rollback recovery is introduced employing ordered logging and parallel release for low overhead middlebox fault-tolerance. In [2], a large-scale empirical study of middlebox failures was performed in a service provider network showing that failures can significantly impact hosted services. Moreover, [17] presents an extensive cloud outage study of common function services and their availability. The deployment of resilient functions in the Cloud is studied in [21] considering fault domains and communication delay. These studies rely on middlebox redundancy, either in terms of checkpoint copies or parallel execution, while our study considers the crucial aspect of limited backup resources.

Similar to our work several works in NFV are based on resource allocation problems. In [22], known methods for solving the Generalized Assignment Problem (GAP) [23] problem were employed in order to achieve a bi-criteria approximation to the performance of the virtual functions placement. In [24], the authors considered also the chaining policy constraints and proposed an Integer Linear Programming (ILP) model to allocate the virtual functions. Encoding schemes for sets and sequences have been suggested to represent middlebox service chains [25]. Recently, [26] studied optimizing delay in an architecture of pipelines shared among flows with different required function chains.

While we relied our theoretical discussion on the traditional 2-way number partitioning problem [13], our algorithm in fact performs multi-way partitioning, which for a given multiset of n positive integers aims to assign each integer to one of k subsets such that the largest sum of the integers is minimized. Several advances in the study of this extended problem have been published recently [27]–[30]. These studies explore the space of potential pairwise decompositions by generating entire subsets of numbers to be partitioned independently.

The problem described in this work is also related to the maximum matching in bipartite random graphs. This was studied in [31], which showed that, for given two disjoint sets L,R with $|R| \geq |L| = n$, in order to achieve with high probability a maximum matching of size n, the number of randomly-chosen neighbors has to be at least 3.

IX. Conclusions and Future Work

In this work we study how the flexibility of NFV can be leveraged for cost-effective survivable deployment of network functions. Following a formal analysis of two problems in this domain, we propose algorithms for computing an effective deployment, given the resource constraints. The algorithms are based on the intuitions obtained in the theoretical analysis of the problems. In a series of experiments, we show the effectiveness of these algorithms, and demonstrate how high levels of survivability can be guaranteed with significant reduction in the amount of required resources.

This paper examines two specific optimization goals – the probability of full survival, and the expected number of operating functions following failures and recovery. Another goal that can be considered is a deployment that fully guarantees

recovery from a restricted small number of failed functions. One can also consider an online variant of the problem, where the failures are given one at a time. For each failure we are required to *online* choose a server on which to run the function (out of those implementing the function), while trying to optimize some goal function globally. This is a variant of online matching in bipartite graphs. The difference here is that the edges to all left-hand-side nodes are known in advance, as well as the probability for each such node to appear in the input. One can also augment this problem by allowing the management system to update its backup scheme following each failure, with or without restrictions on the amount of changes allowed.

REFERENCES

[1] Y. Kanizo, O. Rottenstreich, I. Segall, and J. Yallouz, "Optimizing virtual backup allocation for middleboxes," in *Proc. IEEE ICNP*, Nov. 2016, pp. 1–10.

[2] R. Potharaju and N. Jain, "Demystifying the dark side of the middle: a field study of middlebox failures in datacenters," in *Proc. ACM IMC*, 2013, pp. 9–22.

[3] B. Cully *et al.*, "Remus: High availability via asynchronous virtual machine replication," in *Proc. USENIX NSDI*, 2008, pp. 161–174.

[4] D. J. Scales, M. Nelson, and G. Venkitachalam, "The design of a practical system for fault-tolerant virtual machines," *Oper. Syst. Rev.*, vol. 44, no. 4, pp. 30–39, 2010.

[5] M. Chiosi *et al.*, "Network functions virtualization," ETSI, Darmstadt, Germany, White Paper, Oct. 2012," [Online]. Available: https://portal.etsi.org/NFV/NFV_White_Paper.pdf

[6] A. Gember, P. Prabhu, Z. Ghadiyali, and A. Akella, "Toward software-defined middlebox networking," in *Proc. ACM HotNets*, 2012, pp. 7–12.

[7] J. Sherry *et al.*, "Rollback-recovery for middleboxes," *ACM Comput. Commun. Rev. SIGCOMM*, vol. 45, no. 4, pp. 227–240, 2015.

[8] J. Koziol, *Intrusion Detection With Snort*. Indianapolis, IN, USA: SAMS, 2003.

[9] H. Moens and F. De Turck, "Customizable function chains: Managing service chain variability in hybrid NFV networks," *IEEE Trans. Netw. Service Manage.*, vol. 13, no. 4, pp. 711–724, Dec. 2016.

[10] J. E. Hopcroft and R. M. Karp, "An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs," *Proc. SIAM J. Comput.*, vol. 2, no. 4, pp. 225–231, 1973.

[11] B. G. Chandran and D. S. Hochbaum. (May 2011). "Practical and theoretical improvements for bipartite matching using the pseudoflow algorithm." [Online]. Available: http://arxiv.org/abs/1105.1569

[12] Y. Kanizo, D. Hay, and I. Keslassy, "Maximizing the throughput of hash tables in network devices with combined SRAM/DRAM memory," *IEEE Trans. Parallel Distrib. Syst.*, vol. 26, no. 3, pp. 796–809, Mar. 2015.

[13] S. Mertens, "The easiest hard problem: Number partitioning," in *Computational Complexity and Statistical Physics*, A. Percus, G. Istrate, and C. Moore, Eds. London, U.K.: Oxford Univ. Press, 2003, pp. 125–139, ch. 5.

[14] R. L. Graham, "Bounds on multiprocessing timing anomalies," *J. Appl. Math.*, vol. 17, no. 2, pp. 416–429, 1969.

[15] P. Hall, "On representatives of subsets," *J. London Math. Soc.*, vol. 10, no. 1, pp. 26–30, 1935.

[16] L. Lovász and M. D. Plummer, *Matching Theory*. vol. 367, Providence, RI, USA: American Mathematical Society, 2009.

[17] H. S. Gunawi *et al.*, "Why does the cloud stop computing? Lessons from hundreds of service outages," in *Proc. ACM Symp. Cloud Comput. (SOCC)*, 2016, pp. 1–16.

[18] J. Sherry *et al.*, "Making middleboxes someone else's problem: Network processing as a cloud service," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 42, no. 4, pp. 13–24, 2012.

[19] Y. Dong *et al.*, "COLO: Coarse-grained lock-stepping virtual machines for non-stop service," in *Proc. ACM SOCC*, 2013, p. 3.

[20] S. Rajagopalan, D. Williams, and H. Jamjoom, "Pico replication: A high availability framework for middleboxes," in *Proc. ACM SOCC*, 2013, p. 1.

[21] M. Schöller, M. Stiemerling, A. Ripke, and R. Bless, "Resilient deployment of virtual network functions," in *Proc. Int. Congr. Ultra Modern Telecommun. Control Syst. Workshops (ICUMT)*, 2013, pp. 208–214.

[22] R. Cohen, L. Lewin-Eytan, J. Naor, and D. Raz, "Near optimal placement of virtual network functions," in *Proc. IEEE INFOCOM*, May 2015, pp. 1346–1354.

[23] R. Cohen, L. Katzir, and D. Raz, "An efficient approximation for the generalized assignment problem," *Inf. Process. Lett.*, vol. 100, no. 4, pp. 162–166, 2006.

[24] M. C. Luizelli, L. R. Bays, L. S. Buriol, M. P. Barcellos, and L. P. Gaspary, "Piecing together the NFV provisioning puzzle: Efficient placement and chaining of virtual network functions," in *Proc. IFIP/IEEE IM*, May 2015, pp. 98–106.

[25] R. MacDavid *et al.*, "Concise encoding of flow attributes in SDN switches," in *Proc. ACM SOSR*, 2017, pp. 48–60.

[26] O. Rottenstreich, I. Keslassy, Y. Revah, and A. Kadosh, "Minimizing delay in network function virtualization with shared pipelines," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 1, pp. 156–169, Jan. 2017.

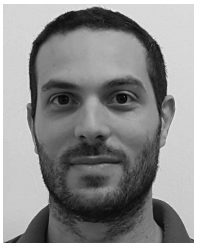
[27] R. E. Korf, "Multi-way number partitioning," in *Proc. Int. Joint Conf. Artif. Intell. (IJCAI)*, Jul. 2009, pp. 538–543.

[28] R. E. Korf, "A hybrid recursive multi-way number partitioning algorithm," in *Proc. Int. Joint Conf. Artif. Intell. (IJCAI)*, Jun. 2011, pp. 591–596.

[29] M. D. Moffitt, "Search strategies for optimal multi-way number partitioning," in *Proc. Int. Joint Conf. Artif. Intell. (IJCAI)*, Aug. 2013, pp. 623–629.

[30] R. E. Korf, E. L. Schreiber, and M. D. Moffitt, "Optimal sequential multi-way number partitioning," in *Proc. Int. Symp. Artif. Intell. Math. (ISAIM)*, 2014.

[31] A. M. Frieze and P. Melsted, "Maximum matchings in random bipartite graphs and the space utilization of cuckoo hash tables," *Random Struct. Algorithms*, vol. 41, no. 3, pp. 334–364, 2012.



Yossi Kanizo received the B.Sc. degree in computer engineering and the Ph.D. degree from the Computer Science Department, Technion, Haifa, Israel, in 2006 and 2014, respectively. He is currently a Lecturer with the Computer Science Department, Tel Hai Academic College, Israel. His main research interests are software defined networking, hash-based data-structures, and switch architectures.

Ori Rottenstreich received the B.Sc. (*summa cum laude*) degree in computer engineering and the Ph.D. degree from the Technion, Haifa, Israel, in 2008 and 2014, respectively. He is currently a Post-Doctoral Research Fellow with the Department of Computer Science, Princeton University, working with Prof J. Rexford. He was a recipient of the Rothschild Yad-Hanadiv Postdoctoral Fellowship and the Google Europe Ph.D. Fellowship in Computer Networking. He also received the Best Paper Runner Up Award at the 2013 IEEE INFOCOM conference and the Best Paper Award at the 2017 ACM Symposium on SDN Research.

Itai Segall received the B.A. degree from Technion, and the M.Sc. and Ph.D. degrees from the Weizmann Institute of Science. He joined Nokia Bell Labs in 2014, where he is currently the Site Leader of Nokia Bell Labs, Israel, and the Department Head of the Autonomous Software Systems Research Tel-Aviv Department. His research interests lie in software engineering and systems research, specifically in the context of Cloud and NFV.

Jose Yallouz received the B.Sc. and Ph.D. degrees from the Electrical Engineering Department, Technion, in 2016 and 2008, respectively. He is mainly interested in computer networks, algorithm design, survivability, and SDN and NFV architectures. He was a recipient of the Israel Ministry of Science Fellowship in Cyber and Advance Computing Award.