SDN-Based Private Interconnection

Shlomi Dolev
Department of Computer Science, Ben-Gurion University of the Negev, Israel.
Email: dolev@cs.bgu.ac.il
Shimrit Tzur-David
Department of Computer Science, Ben-Gurion University of the Negev, Israel and Software Engineering Department, Azrieli - College of Engineering, Israel.
Email: tzurdavi@cs.bgu.ac.il

Abstract—Private interconnection between datacenters is an essential goal due to the popularity of IaaS (Infrastructure as a Service) and SaaS (Software as a Service) architectures. Datacenters intercommunication is needed when an enterprise want to “stretch” its datacenter capacity by extending it with another datacenter on the cloud. This interconnection has to be private so this “stretch” will be considered only virtual. Our research focuses on achieving that privacy on top of SDN-based network. This privacy is achieved without the need to use keys. Namely, information theoretic secure rather than only computational secure. The general idea is to use SDN to enable the creation of several tunnels between each pair of datacenters that intercommunicate. The source uses secret sharing technique to encrypt its data and create \( n \) shares. In order to reconstruct the data, the destination needs to have at least \( k \) shares out of the \( n \) shares that were sent by the sender. We design an algorithm that creates these tunnels with the constraint that only less than \( k \) shares of the same information can reach a single router. This way we achieve a private and secure interconnection between the datacenters.

I. INTRODUCTION

Cloud computing is one of the fastest growing opportunities for enterprises and service providers. Enterprises use the Infrastructure-as-a-Service (IaaS) model to build private and public clouds that reduce operating and capital expenses and increase the agility and reliability of their critical information systems. In order to fulfil these needs, service providers build public clouds to offer on-demand, secure, multi-tenant IT infrastructure to potential customers that now can use cloud services using a public cloud infrastructure.

Due to the growth of businesses and the advent of Big Data, the private clouds are augmented with external resources known in the public clouds. The use of public cloud requires better connectivity between private and public clouds. This resulting “hybrid” cloud should provide transfer and sharing of data. Furthermore, this transfer has to be private.

The Open Networking Foundation (ONF) report on the infrastructure between datacenters [22] states that fast-changing and demanding enterprise and carrier business requirements force changes in network architecture. However, these changes were focused on datacenter server and storage virtualization, while the underpinning network architectures have stagnated with respect to both scalability and manageability.

SDN (Software Defined Network) is an emerging architecture that enables more deterministic, more scalable, more manageable and as we present in this paper, also private virtual networks. This architecture decouples the network control and forwarding functions enabling the network control to become directly programmable and the underlying infrastructure to be abstracted for applications and network services. SDN architecture is presented at Figure 1. With these capabilities, SDN can construct private virtual network between the local datacenters that reside in the private cloud, to the public resources in the public cloud. These virtual networks are called the hybrid cloud. Figure 2 presents the private and public cloud architecture.

Today, the hybrid cloud is not necessarily managed by SDN but we believe that with SDN advantages, most ISPs will adopt this innovative architecture.

In this paper we present a private hybrid cloud in which all the information that passes across the cloud is information theoretic secured. I.e., unless there is no coalition of several routers in the cloud, the information cannot be revealed. This is done by using secret sharing scheme together with SDN to ensure privacy. Encryption with \((n,k)\) secret sharing scheme \((n \geq k)\) is done by creating \(n\) shares from the data such that only by having at least \(k\) shares, the data can be decrypted. In the cloud notations, assume that the data has to be sent from the private datacenter. The source in the private datacenter creates \(n\) shares from the data and sends them to the destination at the public cloud through the hybrid cloud. The SDN controller manages the routes of these shares such that no router sees \(k\) or more shares. This way, we ensure that only the destination at the public cloud that gets all the shares, can decrypt the data, resulting in a private channel in the hybrid cloud. When \(n > k\), we allow \(n-k\) shares to get lost, due to congestion or even by malicious routers.

A. Our Contribution

The contributions of our paper are as follows:

1) To the best of our knowledge, we are the first to use secret sharing for a unicast communication over...
SDN architecture. We show that secret sharing can be very useful to achieve private channel when two parties communicate over a multipath network.

2) Whereas many papers have some contribution on the area of public cloud security, i.e., securing data in the cloud, we are the first to target the problem of \textit{theoretical secured channel} to the public cloud. Note, that this is differ from the known VPNs that supply computational secured tunneling that requires key-based encryption.

3) We show that even if there is a probability to the existence of coalition of several routers in the hybrid cloud, we can still bound the probability for privacy violation.

II. RELATED WORK

Since enterprise customers use public cloud services along with their privately-owned (legacy) datacenters to enable federation between on- and off-premise infrastructures for hosting Internet-based applications, the hybrid cloud has to be very efficient and the traffic has to pass across it between both, private and public clouds as fast as possible. Therefore, extensive research was done to make the hybrid cloud efficient, all the papers that target this goal deal with load balancing, e.g., [2] and efficient migration of applications and services, e.g., [15]. All these works are orthogonal to our work.

The IDC presents a cloud challenges survey among enterprise panel of IT executives [16]. As presented in the survey, security is the top challenge, i.e., the top opportunity for IT suppliers to tackle in the cloud era. Although security is a major concern, most research, e.g., [6], [17], [28], [29] focus in security or privacy in the public cloud rather than security \textit{on-the-way} to the public cloud, thus, also at this case, our work is orthogonal to these works.

CloudBridge is a technology presented by Citrix [1], it connects across third-party public cloud and private networks, offers a platform for cloud-enabling third-party applications. The privacy of CloudBridge is achieved by standard VPN, namely, computational secured channel.

Secret-sharing was first, independently, introduced by Shamir [24] and Blakley [4]. Secret sharing scheme is a tool used in many cryptographic protocols. A secret-sharing scheme involves a dealer who has a secret, a set of n parties, and a collection of subsets of k parties. A secret-sharing scheme is a method by which the dealer distributes shares to the parties such that: (1) any subset of k parties can reconstruct the secret from its shares, and (2) any subset with less than k parties cannot reveal any partial information on the secret.

Secret-sharing schemes have numerous applications in cryptography and distributed computing including secure information storage, Byzantine agreement [23], secure multiparty computations [3], [5], [7], threshold cryptography [9], access control [21], attribute-based encryption [12], [30], and generalized oblivious transfer [25], [27]. None of this works deal with the problem of source-destination communication over several optional paths. We use secret sharing [24] and SDN to enable the secret sharing scheme to be managed by the SDN controller.

Kurihara et al. [18] present an efficient implementation of the secret sharing scheme. They show that the distribution algorithm requires an average of $O(kn \log_2 |S|)$, where |S| is the length (bit-size) of the secret.

In [8], the author proves that the upper bound on the size of the shares is $\Omega(|S| n / \log n)$.

MPLS-based Virtual Private Networks (MPLS VPNs)

![Fig. 1: SDN Architecture](image)
using OpenFlow was previously demonstrated, i.e., in [26]. By using SDN-based MPLS VPNs, one can get easy maintenance and updating of services but the privacy is still achieved by using keys in the edges.

There are also few papers that deal with security issues that arise when using SDN, for example the paper presented at [20] uses multiparty computation technique to secure the data in the controller.

The paper presented at [11] focuses on the effects of end-to-end encrypted networks on Network Intrusion Detection Systems (NIDS) operations. Since encrypted data is very popular these days, standard NIDS cannot identify malicious patterns in the data. In the presented paper, all traffic sent to a receiver by a sender must be replicated and forwarded also to a Central IDS (CIDS), i.e., the sender sends the packet to a proxy and the proxy forwards it to the receiver as well as to the CIDS. Each connection is secured by VPN but thus unauthorized network sniffing is prevented, however, proxies are still able to access network packets relayed through them, which may expose the network packets to unwanted scrutiny.

To ensure confidentiality with respect to the proxies, the authors use secret-sharing. The sender splits the packet to \( n \) shares and sends the shares to \( n \) proxies, the proxies further send the shares to the receiver and the CIDS. There is a major difference between the presented work to our work. The goal of this work is to protect data while it goes through a proxy. The presented approach does not suggest an alternative to a standard VPN, moreover, the proposed approach works on top of the VPN encrypted network. Furthermore, since VPN routes may cross a common network components, the security of the scheme suggested is essentially identical to the security of VPN.

Our work can be used as an information theoretically secure alternative to the standard computational secure VPN.

### III. The SDN-Based Architecture

#### A. Problem Definition

As mentioned, our goal is to provide private interconnection between datacenters. In that case, we have a service provider that uses the cloud to extend its available services by using virtualized servers on the cloud. This service provider has to be able to guarantee to its customers that the interconnection between those datacenters is private, and therefore, to the eyes of any potential customer, these interconnection is transparent. We would like to have a solution that is information theoretic secured, i.e., there is no key that by revealing it, privacy is compromised.

A secret-sharing scheme is a method by which a dealer distributes \( n \) shares to parties such that only authorized subsets of at least \( k \) parties can reconstruct the secret. In our case, when one datacenter intends to communicate with another datacenter, it creates \( n \) shares of its message and distributes these shares to the network. The SDN controller routes these shares to the target datacenter in a way that no router on the way sees \( k \) or more shares. This way, we achieve a private channel between the datacenters. In our case, \( k \) can be equal to \( n \) if we assume that the channels are reliable and nodes do not omit or corrupt shares. Since only \( k \) shares are needed no reconstruct the secret, when \( n > k \), approach similar to forward erasure correcting or error correcting, such as the Berlekamp Welch technique [31] can be used to overcome erasures and even corruptions of at most \( n - k \) shares.

We examine the problem as graph theory problem. We are given a graph \( G = (V, E) \), a source node \( s \), and a sink node \( t \). Each node \( v \in V \) has a determined non-negative capacity \( c_v \). Our goal is to push as much flow as possible from \( s \) to \( t \) in the graph. Each path \( p_i \) has a flow \( f_{p_i} \), the rule is that the sum of the flows of all paths that each node sees cannot exceed its capacity, formally, for each node \( v \),

\[
\sum_{p_i | v \in p_i} f_{p_i} \leq c_v.
\]
B. Reduction to the Maximum Flow Problem

The problem presented at Section III-A can be reduced to the known maximum network flow problem where each edge \( e \) in \( E \) has an associated non-negative capacity \( c_e \), where for all non-edges it is implicitly assumed that the capacity is 0. In this problem, the goal is to push as much flow as possible from \( s \) to \( t \) in \( G \). The rules are that no edge can have flow exceeding its capacity, and for any vertex except \( s \) and \( t \), the flow into the vertex must equal the flow out from the vertex.

The original graph, \( G \), does not have to be directed. In order to reduce our problem to the maximum flow problem, we expand each vertex \( v \) in \( G \) to an directed edge \((v_1,v_2)\). The expansion algorithm is presented at Algorithm 1 and illustration of an undirected graph and its corresponded expanded graph is presented in Figure 3. The input of the algorithm is the graph \( G \), the source and sink, \( s \) and \( t \), and the threshold of the secret sharing scheme, \( k \).

The general idea is first to add a directed edge \((v_1,v_2)\) for each vertex \( v_i \) in \( G \) (except \( s \) and \( t \)), for example, \((v_1,1,v_1,2)\) in the figure, the capacity of this new edge is the capacity of the vertex \( v_i \), \( c_{v_i} \). Then each edge \((s,v_1)\) from the source is replaced by the directed edge \((s,v_1,1)\) (edges \((s,v_1,1)\) and \((s,v_1,2)\) in the figure). Respectively, each edge directed to the sink \((v_j,t)\) is replaced by \((v_j,2,t)\) and \((v_j,s)\). Then for each other edge \((v_i,v_j)\) in \( G \), we add, in \( G \), two directed edges, \((v_i,2,v_j,1)\) and \((v_j,1,v_i,2)\), for example, \((v_1,2,v_2,1)\) and \((v_2,1,v_1,2)\). In order to eliminate the possibility that edge sniffer will reveal the secret, the capacity of these edges is \( k-1 \).

In our paradigm, since the capacity of the nodes in the original graph also represents \( k-1 \), all the capacities of all edges are equal. If \( |V| \) is the number of vertexes and \( |E| \) is the number of edges in \( G \), the resulted directed graph, \( G' \) is a graph with \( (|V|−2) \times 2 + 2 \) vertices and up to \( (|E|−2) \times 2 + 2 \) edges. Note, that the expanded graph is directed to force the paths to include the edges of the expanded vertexes but the input graph can be either directed or undirected. In case of a directed graph, we still add a directed edges \((v_1,v_2)\) for each vertex \( v \), but for each other directed edge \((v_i,v_j)\), we add only the edge \((v_i,2,v_j,1)\) (and not also \((v_j,2,v_i,1)\) as we did for an undirected \( G \)).

We will start by defining the maximum flow problem. One of the most efficient solutions for that problem was proposed by Yefim Dinitz [10]. We choose Dinitz’s algorithm since it is strongly polynomial. The algorithm uses shortest augmenting paths and its complexity is \( O(V^2E) \). Another similar algorithm is the Edmond-Karp [32] algorithm which runs in \( O(VE^2) \). A common assumption is that there are more links than routers, therefore we choose Dinitz algorithm over Edmond-Karp algorithm.

**Definition 1** Let \( G = (V,E) \) be a network with \( s,t \in V \) being the source and the sink of \( G \) respectively. The *capacity* of an edge is a mapping \( c : E \to \mathbb{R}^+ \), denoted by \( c(u,v) \). It represents the maximum amount of flow that can pass through an edge.

<table>
<thead>
<tr>
<th>Algorithm 1 The Expansion Process</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input</strong>: ( G = (V,E),s,t,k-1 )</td>
</tr>
<tr>
<td><strong>for all</strong> ( u \in V</td>
</tr>
<tr>
<td>( V = V \setminus {u} )</td>
</tr>
<tr>
<td>( V = V \cup {u_1,u_2} )</td>
</tr>
<tr>
<td>( E = E \cup {(u_1,u_2)} )</td>
</tr>
<tr>
<td>( c_{(u_1,u_2)} = c_u )</td>
</tr>
<tr>
<td><strong>for all</strong> ( e = (s,v) \in E ) <strong>do</strong></td>
</tr>
<tr>
<td>( E = E \setminus {e} )</td>
</tr>
<tr>
<td>( E = E \cup {(v_1,v)} )</td>
</tr>
<tr>
<td>( c_{(v_1,v)} = k-1 )</td>
</tr>
<tr>
<td><strong>for all</strong> ( e = (u,t) \in E ) <strong>do</strong></td>
</tr>
<tr>
<td>( E = E \setminus {e} )</td>
</tr>
<tr>
<td>( E = E \cup {(u_2,v)} )</td>
</tr>
<tr>
<td>( c_{(u_2,v)} = k-1 )</td>
</tr>
</tbody>
</table>

A flow is a mapping \( f : E \to \mathbb{R}^+ \), denoted by \( f(u,v) \), subject to the following two constraints:

1) \( f(u,v) \leq c(u,v) \), for each \( (u,v) \in E \) (capacity constraint: the flow of an edge cannot exceed its capacity)

2) \( \sum_{v \in V \setminus \{s,t\}} f(u,v) = \sum_{v \in V \setminus \{s,t\}} f(v,u) \), for each \( v \in V \setminus \{s,t\} \) (conservation of flows: the sum of the flows entering a node must equal the sum of the flows exiting a node, except for the source and the sink nodes)

The value of flow is defined by \( |f| = \sum_{(s,v) \in E} f(s,v) \), where \( s \) is the source of \( G \). It represents the amount of flow passing from the source to the sink.

The maximum flow problem is to maximize \( |f| \), that is, to route as much flow as possible from \( s \) to \( t \).

Dinitz uses few definitions in his solution. The first is the *residual capacity*, which is computed as \( c_f(u,v) = c(u,v) - f(u,v) \) and \( c_f(v,u) = f(u,v) \) for each \( (u,v) \in E \). The residual graph is the graph \( G_f = ((V,E_f),c_f|_{E_f},s,t) \), where \( E_f = \{(u,v) \in V \times V : c_f(u,v) > 0\} \). The level graph is defined by \( G_L = (V,E_L,c_f|_{E_L},s,t) \), where \( E_L = \{(u,v) \in E_f : \text{dist}(v) = \text{dist}(u) + 1\} \). The last definition is \( \text{dist}(v) \) which is the length of the shortest path from \( s \) to \( v \) in \( G_f \). By using these definitions, Dinitz shows that the maximum flow problem can be solved in \( \Theta(V^2E) \). We use Dinitz approach to create the flow. This resulting flow defines the paths that will serve us to send the shares.

The maximum flow problem is related to the minimum-cut problem. The problem of \( s-t \) minimum cut in a flow network, is finding the minimum number of edges in the cut-set that creates a cut where the source and the sink are in different subsets. The cut-set only consists of edges
going from the source’s side to the sink’s side. In other words, if the edges in the cut-set are removed, then flow form the source to the sink is completely cut off. The cut-set value is the sum of the flow capacities in the source-to-sink direction over all of the edges in the cut-set. The minimum cut problem is to find the cut-set that has the minimum cut value over all possible cuts in the network.

For any network, having a single source and a single sink, the maximum possible flow from source to sink is equal to the minimum cut value for all cuts in the network. The proof stems from the fact that the maximum flow through a series of linked pipes equals the maximum flow in the smallest pipe in the series, i.e., the flow is limited by the bottleneck pipe. The full proof can be found at [13].

Figure 4 presents the expanded graph of an input graph, for simplicity, the input graph in this example is directed. In order to determine the minimum value of \( k \), we compute the \( s-t \) minimum cut on our expanded graph. If the minimum cut-set contains \( t \) edges, then \( n/k \) has to be less than \( t \). As we can see, the cut-set in our example consists of two edges, \((v_4,1,v_4,2)\) and \((v_5,1,v_5,2)\), this implies that in this graph, \( n/2 \) must be lower than \( k \). In our example, we choose \( n = 10 \) and \( k = 7 \), i.e., each router can see at most 6 shares and therefore this is the capacity on the \((u_1,u_2)\) edges. Figure 4(b) presents the resulting flow and the paths of the different shares.

C. The SDN-Based Solution

The SDN controller defines the route of each flow that occurs in the data plane. The controller computes a route for each flow, and adds an entry for that flow in each of the routers along the path. With all complex functions subsumed by the controller, routers simply manage flow tables whose entries can be populated only by the controller. Communication between the controller and the routers uses a standardized protocol and API. Most commonly this interface is the OpenFlow specification [19]. In an SDN architecture, each router forwards the first packet of a flow to the SDN controller, enabling the controller to decide whether the flow should be added to the router flow table. When a packet of a known flow is encountered, the router forwards it out the appropriate port based on the flow table. The flow table may include some additional information dictated by the controller. With the decoupling of the control and data planes, SDN enables applications and services to deal with a single abstracted network device without concern for the details of how the device operates. Network services see a single API to the controller. Thus, it is possible to quickly create and deploy new applications to orchestrate network traffic flow to meet specific enterprise requirements for performance or security.

In our case, after creating the \( n \) shares, the sender adds an index \( i \), \( 1 \leq i \leq n \) to each share, this index becomes a part of the flow id and thus for each original flow, the controller handles \( n \) “first” packets of these \( n \) initiated flows. Furthermore, each router may have more than one entry but up to \( k-1 \) entries for each flow, where \( k \) is the threshold of the secret sharing scheme. This index can be added either by adding this field into the matching fields structure in the SDN controller platform which causes a change also at the routers, or to use vendor extensions which allow different kinds of matching. By creating the paths based on the maximum flow algorithm, we can
program the controller to route the shares such that no router sees $k$ or more shares. In our example, if $v_{1,1}$ in Figure 4(b) gets the first 4 shares then the forwarding table of $v_{5,2}$ may contain the entries at Table I. Note that there is no limit on the number of packets that go through a single router as long as it does not see $k$ or more packets of the same flow.

### Table I: Forwarding Table of $V_{5,2}$.

<table>
<thead>
<tr>
<th>Flow Id</th>
<th>Next Hop</th>
</tr>
</thead>
<tbody>
<tr>
<td>$f_1$</td>
<td>$v_{7,1}$</td>
</tr>
<tr>
<td>$f_2$</td>
<td>$v_{7,1}$</td>
</tr>
<tr>
<td>$f_3$</td>
<td>$v_{6,1}$</td>
</tr>
<tr>
<td>$f_4$</td>
<td>$v_{6,1}$</td>
</tr>
</tbody>
</table>

**IV. OVERHEAD AND COMPLEXITY**

Network overhead is one of the concerning implications of our paradigm. Each packet that is sent between the source and the destination is split into $n$ shares. Assuming that each share is sent within a packet, $n$ packets are sent for a single packet of data.

If $m$ packets of data should be sent from the source to the destination, $n \times m$ packets are sent. This might be considered impractical and a high volume of data is sent in the network in general and in particular between the two nodes. Therefore, if this is the case, our paradigm can be used for exchanging keys for the use of standard VPNs. This way, the privacy of the keys is theoretically secured and we eliminate man-in-the-middle attacks.

By using the $(n,k)$ secret sharing scheme, we evaluate a polynomial of degree $k - 1$ at $n$ points. By Horner scheme [14], this polynomial can be expressed as multiple linear operations and it requires $O(n)$ multiplications. In order to recover the packet, the destination uses Lagrange interpolation polynomial with complexity of $O(k^2)$.

The controller operates maximum once for each flow to construct the maximum flow algorithm on the network topology. In practice, it will compute the algorithm only a few times for each couple of source and destination since the network is typically undergo only limited and infrequent dynamic changes. The computational complexity of the controller depends on the network size and it is $O(V^2E)$. In the realistic case where several controllers manage the network where each controller has its own part of the network, this computation can be done in parallel, thus $V$ and $E$ are nodes and edges at each part of the network.

**V. COALITIONS AND TRUST**

Our paradigm works as long as the nodes in the paths do not share their data. Our building block assumption is that the network is reliably managed by the controller, i.e., the controller can be trusted. If this is not the case, either by a malicious controller or by an adversary that controls
the controller, the controller can simply forward all shares to a single router to reveal the secret. In addition to the controller, we also assume that the routers (the nodes in the graph notations) behave according to the controller routes. If there is a coalition of two or more nodes, the current solution might fail. Assume that, with probability $p$, nodes $v_3$ and $v_5$ in Figure 4(a) share their data, in that case, they should be considered as one node and therefore the cut-set of the corresponding expanded graph contains only one edge, hence, there is no solution for any $n$ or $k$. Thus, the probability that there is no solution for private channel is $p$. On the other hand, if, with probability $p$, nodes $v_2$ and $v_3$ share their data, then, after merging these two nodes, the cut-set contains two edges, which implies that there are two paths from the source to the sink, i.e., the $n$ shares are now sent through two paths.

In the general case, if before merging two nodes in a coalition, the cut-set contains $\ell$ edges, we first need to calculate the number of possibilities to distribute the $n$ shares over the $\ell$ paths with the constraint that each edge in the cut-set sees at most $k-1$ shares; we call this set of possibilities $\Lambda$. This is equivalent to distributing $n$ balls in $\ell$ cells such that no cell contains more than $k-1$ balls, or by rephrasing, all the possibilities to distribute $n$ balls to the $\ell$ cells minus the possibilities that any of the cells contains $k$ or more balls. The number of possibilities in $\Lambda$ is

$$\binom{n+\ell-1}{n} - \ell \times \binom{n-k+\ell-1}{n-k}. \quad (1)$$

Note that for simplicity we assume that there are no cases in which two or more cells contain $k-1$ balls, i.e., $n < 2k - 2$; this assumption is reasonable since, as we will see at the end of this analysis, in order to maximize the probability for a private interconnection, the best value for $k$ is $n$ and certainly $k > n/2 + 1$. Out of the possibilities in $\Lambda$, we need to find those that fulfill the constraint that the merged nodes see at most $k-1$ shares, or with the balls notations, the sum of balls in the two merged cells is less than or equal to $k-1$. If $t$ is the sum of balls in the merged cells, then there are $\ell - 2$ more cells that each of which can contain up to $k-1$ balls, thus, the lower bound of $t$ is

$$\text{Max}\{n-(\ell-2)(k-1),0\}, \quad (2)$$

we call this bound $t_{ib}$. For each possible value of $t$, we have to calculate the number of possibilities of distributing $t$ balls in the two merged cells $\ell$ times the number of possibilities to distribute the other $n-t$ balls in the other $\ell-2$ cells, formally

$$\sum_{t_{ib} \leq t \leq k-1} \binom{t+1}{t} \times \binom{n-t+\ell-3}{n-t} \times \binom{n-k+\ell-1}{n-k}. \quad (3)$$

Therefore the probability for a private interconnect in the general case is

$$p \times \sum_{t_{ib} \leq t \leq k-1} \binom{t+1}{t} \times \binom{n-t+\ell-3}{n-t} \times \binom{n-k+\ell-1}{n-k} + (1-p) \quad (4)$$

By choosing $n=k$ and large $n$, this probability tends to 1.

For example, if $n=k=10$ and $\ell=3$, then $t_{ib}=1$ and the probability to get a private interconnection is

$$p \times \frac{\sum_{t_{ib} \leq 9} \binom{t+1}{(10-t)} \times \binom{10-(t+1)}{10-t}+ (1-p)}{63} + (1-p) =$$

$$p \times \frac{\sum_{t_{ib} \leq 9} \binom{t+1}{(10-t)}+ (1-p)}{63} =$$

$$p \times 54/66 + (1-p) =$$

$$1 - 0.182p$$

If we choose $n=k=20$, the probability becomes $1 - 0.0833p$. Note, that if the sender knows about the possibility of a specific coalition and the probability to find a flow given that coalition is positive, it can first merge the nodes of the coalition and then find a flow that fulfills the sharing scheme constraints.\(^1\) The above analysis was done for a single coalition of two nodes, in case there are more nodes in the coalition or another coalition, the same process can be done recursively.

At mentioned at Section II, the time computation of the secret sharing scheme depends on $n$ and $k$. Therefore, when there is full trust on the routers, $n$ and $k$ should be relatively low. In order to increase the probability of a private channel, in case there is some positive probability for a coalition in the network, the source may choose larger $n$ and $k$, even in the price of compromising the algorithm efficiency.

VI. CONCLUSIONS

We have proposed a scheme that enables fully private interconnection between datacenters on top of a SDN-based hybrid cloud. In order to ensure this privacy, we use a $n$-$k$ secret sharing to encrypt the data. In our scheme, the source creates $n$ shares of its data and sends them to the network. The SDN controller manages the paths such that no $k$ or more shares pass the same router. This way, the interconnection between the datacenters is theoretic secured, i.e. unless two or more routers share their data, the encrypted data can never be revealed. We showed that

\(^1\)If the probability to find a flow is positive, there is a flow that fulfil the constraint so the max-flow algorithm finds it.
the problem of finding these paths can be reduced to the maximum flow problem by expanding the original graph. By having a centralized controller, we can compute the flow on the network and determine a bound on the ratio $n/k$, which is the number of unique paths that are required to apply the $n$-secret sharing scheme on the network. Once we have the flow and the values of $n$ and $k$, the sender creates $n$ shares of each packet such that each flow is expanded to $n$ flows where the flow id is the flow data together with the share index. This way, the controller, which gets the first packet of each flow (corresponding to our new definition of flow), routes the shares to the corresponding paths and we obtain the private channel between the datacenters.

REFERENCES