A greedy approach for minimizing SDN control overhead

Mathis Obadia, Mathieu Bouet, Jean-Louis Rougier, Luigi Iannone

A Greedy Approach for Minimizing SDN Control Overhead

Mathis OBADIA[12] and Mathieu Bouet[2] and Jean-Louis Rougier[1] and Luigi Iannone[1]

[1]*Télécom Paristech France* [2]*Thales Communications and Security*

L'approche SDN implique une séparation entre le plan de données et le plan de contrôle centralisé. Pour des raisons de passage à l'échelle et de fiabilité ce plan de contrôle centralisé doit être physiquement distribué dans le cas de déploiements de grande envergure. Dans cet article nous abordons la question de la minimisation de la surcharge de trafic que l'utilisation d'un plan de contrôle distribué implique. Nous résolvons ce problème de manière optimal pour de petites instances grâce à l'utilisation de technique d'optimisation linéaire et introduisons un algorithme glouton capable d'obtenir des résultats proches de 3% de l'optimal en un temps moindre et sur des topologies bien plus grandes.

Keywords: "SDN", "Distributed Control Plane", "Communication Overhead"

1 Introduction

The Software Defined Network paradigm advocates for a logically centralized control plane. When a physically centralized control plane is not feasible, for instance because of scalability and/or resiliency issues, physically distributed controllers needing to act like a logically centralized control plane are used. This means sharing information among themselves in order to keep a global consistent view of the network. The implication is that a lot of data is exchanged over the control plane. While this might not be an issue in a data center network, where bandwidth is cheap, in large scale WANs the issue of control overhead may become predominant, especially in constrained networks [ea14].

In this paper we focus on SDN controller placement and control plane topology adaptation so as to minimize control overhead. We first model the problem as a Mixed Integer Program (MIP), in order to get the optimal controller placement and the optimal spanning tree topology formed by active controllers. Such model is used as reference point because of its optimality. But it uses too much time and computation resources to be used on large topologies. We then introduce a greedy algorithm able to compute quasi-optimal placement and control plane topology in a very short time, with a less than 3% margin from the optimal computed via the time consuming MIP approach.

The controller placement problem in distributed SDN architectures has been studied mostly by trying to minimize switch-controller delay [ea12], which is a very important metric in an architecture where switches sometimes need to buffer flows while waiting for the controller reply. Other extensions of the problem have tried to address the reliability issue with different placement mechanism and related metrics [eaa] [eab]. Differently from the above-mentioned proposals, our work focuses on the problem of efficiently sharing relevant network information between controllers and how their placement can impact the overhead on the control plane. This scheme includes the controller placement but also the minimum spanning tree among controllers. The latter is a major difference from other publications where all inter-controller distance is considered. This is especially relevant in locally controlled SDN architectures [ea13] and when using constrained control links [ea14].

The rest of the paper is structured as follows. Sec. 2 introduces our model,the MIP and the greedy algorithm. Both models are then compared in Sec. 3. Sec. 4 concludes the paper.

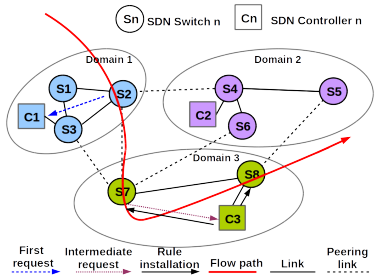


FIGURE 1: Flow requests and installations in distributed SDN networks.

2 Mixed Integer Program Model and Greedy heuristic

Hereafter we first provide the general network model and then define the cost function used to formally quantify the control overhead. We then present our greedy heuristic that aims at minimizing overhead.

2.1 Network Model

The network graph is $G = (S, E)$: composed by SDN-enabled switches S and the physical links between them E. Controllers can be placed on $C \subseteq S$. We introduce the following binary decision variables : $x_i = 1$ if controller i is active. The network can be seen as split in different *control domains*, each domain representing the set of network elements controlled by the same controller, represented by $y_{i,m} = 1$ if switch i is assigned to controller m. Since most of the data sent by a controller has to be sent to all the other controllers, the most efficient way to share information over the overlay is a spanning tree, modeled as follows : $u_{m,h} = 1$ if the distance in the spanning tree between m and the root is equal to h. If n is the predecessor of m in the spanning tree then $p_{m,n} = 1$.

Whenever a packet arrives at a switch, if the packet does not match any entry in the flow table, a `packet_in` is sent to the controller of this switch and the flow is buffered until a rule is installed. The number of `packet_in` sent to the controller highly depends on the way the network is operated and what is the flows traffic matrix. We indicate by $T = T_{i,j}$ the flows traffic matrix (in flow/s). Where $T_{i,i}$ is the number of flows without a match coming from the hosts connected on switch i, while $T_{i,j}$ is the number of flows without a match coming from switch i to switch j.

2.2 MIP

We shall define the overhead traffic as composed by all the traffic in the control plane, that is all switch-controller communications and controller-controller communications. It is composed by (in order of appearance in the long sum for the MIP problem) :

Data collection cost :SDN switches make available a number of statistics, the controller has to periodically (with f_{dc} bit-rate) ask switches to send the relevant data.

Cost of route request and route installation : A number of different control plane communications need to happen when a new flow is generated to install the right rules on all the switches where the flow will transit, those communications are represented in Fig.1. When a new flow coming from an host arrives at a switch, a `packet_in` (size f_p) is generated and sent to the controller of this switch. When a flow arrives from another domain, the controller discovers this flow, so the same process has to be applied. Finally the controller needs to install rules on all the switches in its domain via a `flow_mod` (size f_{fm}) packet.

Synchronization cost : The data collected by a controller m (with f_{syn} bit-rate) that needs to be sent to all the other controllers is proportional to the number of switches inside the domain of m and to the size of the spanning tree between all controllers.

We obtain the following MIP problem :

Minimize : $F = f_{dc} \sum_{m \in C} \sum_{i \in S} d_{i,m} y_{i,m} + f_p \sum_{m \in C} \sum_{i \in S} T_{i,i} d_{im} y_{i,m} +$
$f_p \sum_{i \in S} \sum_{j \in S} \sum_{m \in C} \sum_{n \in C, m \neq n} T_{j,i} y_{i,m} y_{j,n} d_{i,m} + f_{f,m} \sum_{i \in S} \sum_{j \in S} \sum_{m \in C} T_{j,i} d_{i,m} y_{i,m} + f_{syn} N \sum_{m \in C} \sum_{n \in C} p_{m,n} d_{m,n}$
with the following constraints :

$$\sum_{m \in C} y_{i,m} = 1 \quad \forall i \in S$$ Every switch is controlled by one controller exactly

$$y_{i,m} <= x_m \quad \forall i \in S m \in C$$ Switch i is controlled by m only if m is active

$$x_{i,m} delay_{i,m} <= \delta \quad \forall i \in S m \in C$$ Maximum switch-controller distance is less than δ

$$\sum_{l=1}^{H} u_{m,l} = x_i \quad \forall m \in C$$ Single loop free path from each controller to root

$$\sum_{m \in C} u_{m,0} = 1$$ Root uniqueness

$$\sum_{m \in C} p_{m,n} = x_j - u_{j,0} \quad \forall n \in C$$ Active controllers have one predecessor except for root

$$p_{m,n} <= 1 - u_{j,l} + u_{i,l-1} \quad \forall n \in C m \in C l = 0..L$$ m predecessor of n if his distance to root is 1 less

Where $d_{i,m}$ is the distance (in hops) between node i and node m, f_p is the packet size (in bytes) of the `packet_in`, $f_f m$ is the average size (in bytes) of a `flow_mod` packet.

2.3 Algorithm

To solve the problem formalized in the previous paragraph, we propose to use a greedy heuristic. The algorithm starts with all possible controllers activated, on each iteration it tries to turn off one of the controllers and computes the associated overhead cost. The first step of this calculation is choosing controller domains : each switch is controlled by the closest (in term of delay) activated controller. Each time a switch is associated with an active controller, the algorithm checks that the switch-controller delay is less than the maximum delay tolerated by the model. If it is true costs calculations can continue, the configuration is skipped otherwise. The cost of *data collection* and *route requests and installation* can be computed. The minimum spanning tree between controllers is calculated using Prim's algorithm and the cost of synchronization can also be computed [ead]. If the calculated cost is less than the previous minimum this new configuration is saved. The algorithm tries to shut down each possible controller on every iteration and uses the configuration with the minimum cost as a starting point for next iteration. The algorithm stops when stopping any controller causes the cost of overhead to grow.

3 Evaluation

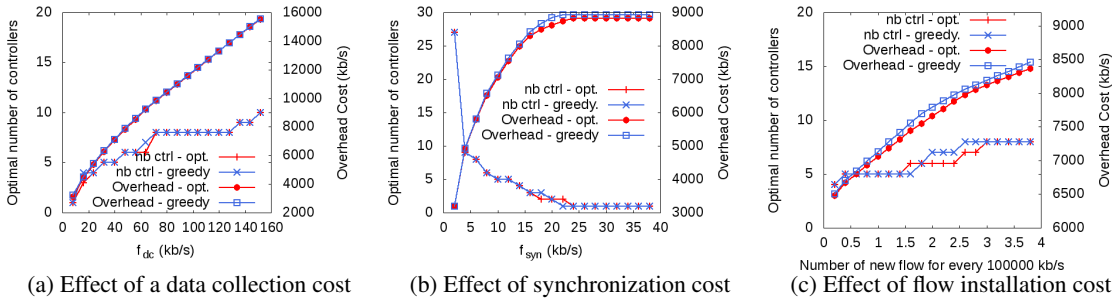
3.1 Implementation and Experimental Setup

In order to compute cost function defined in our model, we need realistic cost weights for as well as a realistic topology and the related traffic matrix. f_p and f_{fm} are openflow packets with size 80 bytes. The data collection cost highly depends on how much monitoring controllers need to react quickly to have a responsive network we used 40 KB/s in this simulation. The synchronization traffic includes reachability information, heart beat messages for failure detection [eac] and route quality information. The synchronization traffic sent by a controller to all the other controllers is an aggregation of the collected data $f_{syn} = f_{dc}/10 = 4KB/s$. We used the GEANT [ea06] topology as the network topology, assuming that each router is replaced by a SDN switch, there is one new flow every 100 000 KB exchanged, and $C = S$.

3.2 Evaluation Results

The baseline costs we chose are an example of a realistic setup, but can widely change according to network applications running on the controllers. To show the influence of those costs we evaluated the variation of the costs and of the optimal number of controllers according to those parameters.

Fig. 2a shows the impact of data collection cost F_{dc} on the overall cost. When the data collection cost grows because of a higher weight f_{dc}, the impact of switch-controller distance is stronger and the optimal placement needs controllers close to the switches, which causes the optimal number of controllers in the



(a) Effect of a data collection cost (b) Effect of synchronization cost (c) Effect of flow installation cost

FIGURE 2: Effect of a flow installation cost on the overhead cost and the number of controllers on the GEANT topology

network to grow. On the contrary, when the cost of synchronization F_{syn} grows, the size of the spanning tree between controllers becomes the predominant factor in the total overhead cost, thus, the number of controllers has to be reduced, as shown in Fig. 2b. The cost of flow requests and installation grows like the cost of data collection, with the exception that instead of having a fixed per-switch cost, the cost depends of the traffic coming from other domains and hosts directly connected to the switch. When the traffic generates more new flows, the switch-controller distance has greater impact and has to be smaller, hence, the number of controllers higher.

To test the effect of the network size on the overhead cost and optimal number of controllers, we generated random graphs using the Barabasi Algorithm [Bea02]. In all experiments, results showed that our greedy algorithm is very close to the optimal for the number of nodes tested, with a deviation from the optimal of less than 3% for all evaluated topology and at least 10 times quicker. Results are not available for topology bigger than 60 nodes because the CPLEX solver ran out of it's 4GB allowed memory. Our algorithm provides a near-optimal solution in less than a minute even for topologies of a hundred nodes.

4 Conclusions and perspectives

In this paper we tackle the issue of minimizing communication overhead in the control plane of distributed SDN networks Such goal is achieved by control plane topology adaptation (i.e., carefully placing the controllers) including a building minimal spanning tree between controllers. We developed a MIP based linear formulation that can be used to find the optimal solution on topologies with a moderate size. We also presented an heuristic based on a greedy algorithm to find a near optimal controller placement and its associated spanning tree topology. We showed numerically that this algorithm gives results very close the optimal in a reasonable time on real world as well as random topologies of different sizes.

Références

[Bea02] A. Barabási et al. Statistical mechanics of complex networks. *RMP*, 74(1) :47, 2002.

[eaa] Hock et al. Pareto-optimal resilient controller placement in sdn-based core networks. In *ITC, 2013 25th International*.

[eab] Hu et al. Reliability-aware controller placement for software-defined networks. In *Integrated Network Management (IM 2013) IFIP/IEEE*.

[eac] Mathis Obadia et al. Failover mechanisms for distributed SDN controllers. In *SCNS'14*.

[ead] Moret et al. An empirical assessment of algorithms for constructing a minimum spanning tree. *in proc of DIMACS 1994*.

[ea06] Uhlig et al. Providing public intradomain traffic matrices to the research community. *ACM SIGCOMM CCR*, 2006.

[ea12] Heller et al. The controller placement problem. In *Proc. ACM HotSDN*, 2012.

[ea13] Schmid et al. Exploiting locality in distributed sdn control. In *Proc. ACM HotSDN*, 2013.

[ea14] Phemius et al. DISCO : Distributed multi-domain SDN controllers. In *IEEE/IFIP NOMS*, 2014.