

WoS-CoMS : Work Stealing-based Congestion Management Scheme for SDN programmable networks

Yannick Florian Yankam (✉ yyankam@yahoo.fr)

University of Dschang

Vianney Kengne Tchendji

University of Dschang

Jean Frédéric Myoupo

University of Picardie Jules Verne

Research Article

Keywords: Work stealing, Congestion control, SDN, Task scheduling, Quality of service

Posted Date: August 2nd, 2023

DOI: <https://doi.org/10.21203/rs.3.rs-3213604/v1>

License:  This work is licensed under a Creative Commons Attribution 4.0 International License.

[Read Full License](#)

Additional Declarations: No competing interests reported.

Version of Record: A version of this preprint was published at Journal of Network and Systems Management on January 13th, 2024. See the published version at <https://doi.org/10.1007/s10922-023-09798-1>.

WoS-CoMS : Work Stealing-based Congestion Management Scheme for SDN programmable networks

Yannick Florian Yankam^{1*}, Vianney Kengne Tchendji² and Jean Frédéric Myoupo³

^{1*}Department of Telecommunication and Network engineering, IUT-FV, University of Dschang, 134, Bandjoun, Cameroon.

²Department of Mathematics and Computer Science, Faculty of Science, University of Dschang, 67, Dschang, Cameroon.

³Computer Science Lab-MIS, University of Picardie Jules Verne, 33 rue St. Leu, 80039, Amiens, France,.

*Corresponding author(s). E-mail(s): yyankam@yahoo.fr;
Contributing authors: vianneykengne@yahoo.fr;
jean-frederic.myoupo@u-picardie.fr;

Abstract

In recent years, the SDN (Software-Defined Networking) paradigm emerged as an easy way to manage large-scale network infrastructures through programmability brought out and its control plane/data plane decoupling logic. This enables infrastructure and service providers to have a global view of the network and track traffic flows from a remote controller. However, congestion control remains a concern due to the evolution of increasingly complex and resource-intensive user requirements (virtual reality, metaverse, Internet of Things (IoT), Artificial Intelligence (AI), Cloud, ...) on network infrastructures. This server state leads to high latency in request processing and data loss. This paper proposes in such controller-supervised environment, a congestion management scheme within network service servers to maintain acceptable quality of service. The strategy relies on work stealing to ensure better workload balancing. Simulations show that the proposed solution can reduce congestion load into the servers by up to 22%, depending on request grain size, within a shorter latency than other works in the literature. Moreover, the proposed solution allows stolen tasks to be completed within a shorter timeframe.

Keywords: Work stealing, Congestion control, SDN, Task scheduling, Quality of service

1 Introduction

Most of the modern network services offered by large-scale network operators require very low response times to user requests, whatever the state of the infrastructure (IoT, 5G, gaming, ...) [1]. As a result, infrastructure providers must constantly focus on developing architectures and management policies to meet these high responsiveness targets. But this requires greater control of the infrastructure, allowing changes to be detected quickly (link failure, congestion, etc.) and relevant measures to be taken in a real timeframe. Over recent years, SDN has gained a firm foothold in this area. SDN assigns the definition of network control rules to a controller through the control plane, and the implementation of these rules to managed equipment (openflow switches) through the data plane [2, 3]. All these features provide a global view of network device activities, including the servers receiving user requests. Figure 1 shows the SDN paradigm.

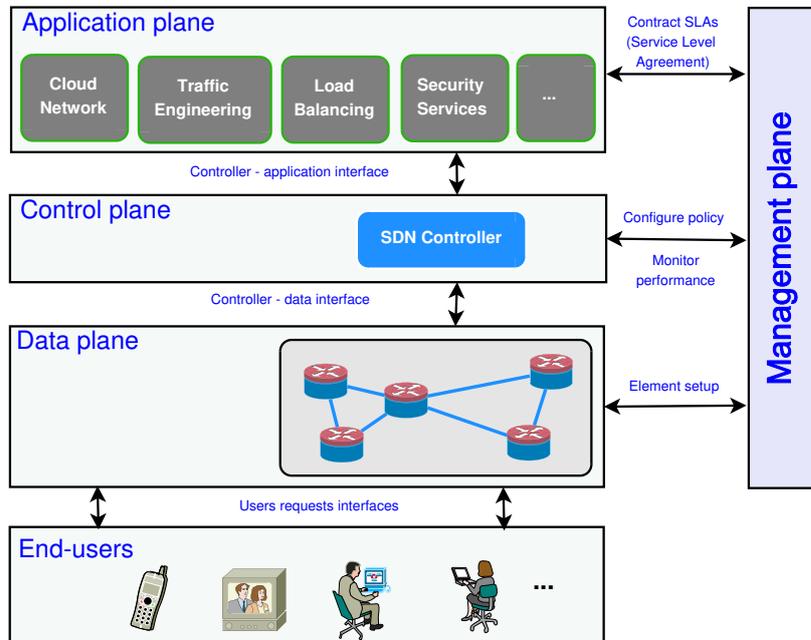


Fig. 1: SDN architecture.

Because of the high added value of the services offered to users, requests are increasingly numerous and difficult to satisfy by the servers, resulting in frequent congestion.

While the controller can quickly detect congestion, it is harder to end it immediately [4]. To achieve this, other servers in the relevant domain are usually involved. To this end, [5–7] proposed, in a virtualized environment supervised by a controller, a server replacement scheme, which replace a congested server to another with more available resources. But this approach requires migrating a lot of data, which overloads the links during migration. Nevertheless, response times after repositioning are better than the load balancing proposed by Seddiki [8]. Seddiki’s load balancing algorithm constantly looks for a balance of requests between virtualized servers. This method keeps these servers working continuously to process requests and execute the load balancing algorithm.

Our motivation is to provide a solution to server congestion in SDN environments by overcoming the limitations of the work cited above [5, 7, 8]. The proposed solution uses work stealing [9, 10] to mitigate workload on congested servers to the benefit of other servers. Work stealing involves an idle processor stealing one or more tasks from its neighbor’s waiting list [9, 11, 12]. This method already proved its ability to achieve lower latencies in single parallel job execution on multiple processors [9, 10, 13]. But the work-stealing strategies presented in previous work are still to our knowledge largely unused in server congestion scenarios in SDN networks. We describe our contribution in three points:

- A task selection method : we define some key metrics that we use to identify a ”stealable” task, such as independence, priority and work weight.
- A server selection algorithm allowing execution of stolen tasks. We rely on the network overview provided by the SDN controller [1, 14] to identify these server(s).
- An automated task transfer algorithm to the execution server, enabling low transfer time and task’s queue consistency.

The remainder of this paper is organized as follows : in Section 2, we present a brief review of congestion management in SDN networks, then describe our solution for allocating tasks to servers in Section 3; next, our simulation results and discussion are presented in Section 4. Finally, Section 5 presents our conclusions and directions for future work.

2 State of the art

Congestion refers to the state of a system whose capacity to handle incoming user requests has been exceeded [15, 16]. This situation is recurrent in single or multi-service environments shared between several users, such as in the SDN-Based Clouds [17, 18].

Overcoming server congestion in SDN networks has been tackled in several directions. Some of these works have investigated the load balancing direction [8, 17] over several servers in order to avoid congestion as long as possible. To ensure this balancing, [17, 18] prioritizes specific requests over others. The metric used is the type of service and the number of incoming requests. A service with higher bandwidth requirements (multimedia [17], games [19], etc) deserves higher priority. It is the same for a large number of requests [18]. These works highlight the importance of load balancing across multiple servers, based on shortest-path flow routing policies, in order to achieve

lower latency. Such policy biases light services against heavy ones. Server replacement [5–7] solves this problem with a migration of the hosted service to another server with more resources, thereby enhancing the processing ability of the congested server. Thus, all requests related to different services can be quickly processed, regardless of the task’s granularity. However, this result can only be achieved once the service has been completely replaced. A more cost-effective approach would be to execute queries while minimizing the number of data migrations. Work stealing could be the solution.

So far, work stealing has been used to provide better task management between the different cores of a processor in multiprocessor systems. For event-driven systems, [20] proposed task stealing algorithms such as Cache-aware stealing, Batch stealing and Handler pinning which can improve web data server performance by 15% compared with previous algorithms. In the Libasync-smp algorithm [21], each user request is saved in an event list. Event processing refers to task execution. One or more tasks can be executed in a thread. When a thread is out of tasks to execute, it checks if any other threads still have events in their queue. If this is the case, unoccupied threads try to steal some of them: it starts by choosing a color (group of requests) to steal (excluding those currently running) and moves all events of this color into its queue. In this way, the mutual exclusion guaranteed for events of the same color is preserved. When several threads are potentially stealable, the stealing algorithm chooses the one with the most events in its queue. However, the major drawback of this algorithm lies in its significant drop in performance when scaling up in multi-core environments and, by extension, in multiserver environments. [20] solves the scaling problem by integrating the ability to steal multiple tasks, even if the number does not exceed 50% of available tasks. For task processing in a network environment, [20] selects any server that accepts TCP (Transmission Control Protocol) connections easily. The "Handler pinning" version with the worker accepting TCP connections produces a higher acceptance rate (+12%) than the original version of task stealing (-8.5%). However, the TCP incoming connection acceptance metric is not sufficient on its own to successfully process stolen tasks.

[9] focuses on task weights and avoids capturing lightweight tasks in the stealing process. This trick reduces the number of work steals and subdivisions. Each server self-assesses the priorities and weights of the competing tasks. This algorithm uses these two metrics (priority and weight) to determine the victim. However, a restriction is placed on the number of tasks that can be stolen from an external node. In multiprocessor systems, this restriction can be useful for reducing power consumption, which is a major requirement for improving battery life, reducing cooling costs and improving overall system scalability and reliability.

Concerning the task processing order for congested data centers, [4] show that this problem stems from the scheduling policy used on the one hand for packet transfer by switches, and on the other hand for request processing within servers. [22] previously addressed congestion control and task scheduling only at the switch level for better packet routing. [22] focused on bandwidth usage and hop count as key criteria for good scheduling. To improve control on servers, [4] proposes to prioritize tasks with the shortest remaining execution time (Shortest Remaining-Processing-Time (SRPT)) when scheduling, as opposed to a FIFO (First In First Out) or Fair Queuing approach,

and to progressively move tasks out of the queue of tasks to be processed. However, heavier tasks may never be executed, or executed very late.

In section 3 we present our task stealing scheduling scheme to solve this balancing problem in query processing in SDN environment, which is at the best of our knowledge the first one using this approach with a SDN controller. Most previous work on task stealing has been carried out in unsupervised distributed environments. Table 1 compares the theoretical approach of our solution with the literature.

3 A new congestion management algorithm

In this section, we present WoS-CoMS, our task stealing-based server congestion management algorithm. We consider a set S of servers providing homogeneous services in an infrastructure supervised by a SDN controller. Our work stealing scheme investigates three key points: the granularity of stolen works, the stealing strategy (job request or snatch) and the selection of the runtime server.

3.1 Works granularity

Queries which cause congestion on a S_i server can be either dependent or independent. We consider both levels of granularity. Let's consider a set of tasks $J_i, i \in 1..3$. We consider the dependency types shown in figure 2. Singletons are tasks that have no dependencies on each other. They are therefore the easiest to steal, as their granularity level is null.

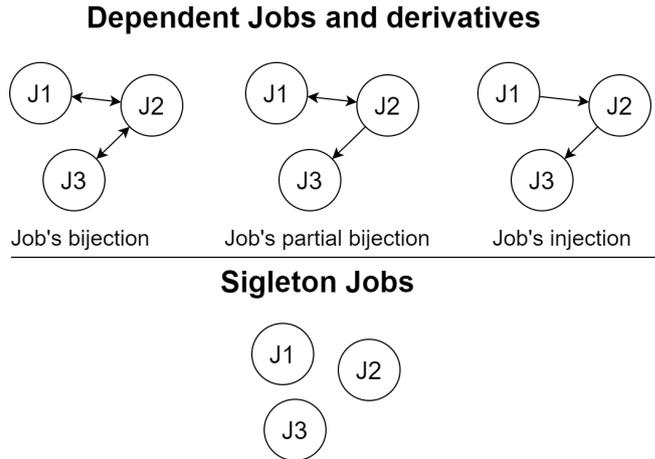


Fig. 2: Types of Dependencies.

Queries dependent on each other have weights according to the number of linked queries [24]. So, stealing a task that depends on other tasks must be followed by stealing those tasks as well. When the steal is completed, the requesting tasks are removed from the source server's queue and placed on the processing target server's queue.

Table 1: Comparison of congestion control schemes.

| Authors | Issues | | Scheduling criteria | | | | Application field | | | Optimization | | Environment | |
|-------------------------|--------|--------|---------------------|------|-----|--------|-------------------|--------|-----|--------------|----------|-------------|-------|
| | CC | Sched. | BW. | Hops | RRT | Weight | Switch | Server | CPU | SR | TCP Acc. | Dist. | Cent. |
| WoS-CoMS | ✓ | ✓ | ✓ | × | ✓ | ✓ | × | ✓ | × | ✓ | ✓ | × | ✓ |
| ERASE [23] | ✓ | ✓ | × | × | × | ✓ | × | × | ✓ | ✓ | × | ✓ | × |
| Nakashima et al. [9] | ✓ | ✓ | × | × | × | ✓ | × | × | × | ✓ | × | ✓ | × |
| Mushtaq et al. [4] | ✓ | ✓ | × | × | ✓ | ✓ | ✓ | ✓ | × | × | × | ✓ | × |
| D. Shah and Q. Xie [22] | ✓ | ✓ | ✓ | ✓ | × | × | ✓ | × | × | × | × | × | ✓ |
| Handler pinning [20] | × | ✓ | × | × | × | ✓ | × | ✓ | × | × | ✓ | ✓ | × |

Legend

1- CC : Congestion control

2- Sched. : Scheduling

3- BW : Bandwidth

10- Cent : Centralised environment

4- Hops : number of Hops

5- RRT. : Running Remaining Time

6- Weight : task weight

7- SR : Stealing Rate

8- TCP Acc. : TCP Acceptation

9- Dist. : Distributed environment

A stealing log is kept at server level in order to preserve the related task processing consistency.

Concerning independent requests, we can steal them anytime an execution platform is available. However, this availability is not a sufficient condition to perform a successful steal. The source server may deny the task to another server.

3.2 Stealing strategy

The work stealing strategy starts as soon as a server's congestion threshold (80% according to [10]) is reached. We propose two stealing strategies : the supervised approach and the unsupervised one.

3.2.1 Supervised Work stealing method

The S_{init} server sends a STB.REQUEST (Stealing-Task Broadcast Request) to the controller looking for a backup server. The controller broadcasts the request in the network in search of an available server, and retrieves the offers ST_OFFER (Stealing Task Offer) submitted by the applicant servers. Figure 3 present the proposed structure of these packets. Based on the network configuration, the controller selects the first valid offer and links S_{init} and S_i . S_i can then initiate task stealing. Algorithm 1 describe our backup server selection strategy perform by the controller with a time complexity of $m \times \mathcal{O}(k + n \log n)$, where m is the offers queue size, k is the number of arcs in the network and n is the number of vertices. If the controller previously saved the computed shortest paths from any source to any destination, the algorithm 1 will achieve a time complexity of $\mathcal{O}(m)$.

Algorithm 1: Controller's backup server selection algorithm.

```

input : List of ST_OFFER  $LO$ ,
        Task to Steal  $ST$ ,
        network graph  $G$ 
output: A valid ST_OFFER
1 Let  $P$  be an empty routing path;
  Let  $SO$  be an empty ST_OFFER;
  if  $empty(LO)$  then
2   | return  $SO$ ;
3 else
4   | foreach  $SO_i \in LO$  do
5     |   | if  $SO_i \rightarrow proposed\_runtime \approx 0$  then
6       |   |   |  $P \leftarrow G \rightarrow$  Shortest path from  $S_{init}$  to  $SO_i$  Server; /* Using
7         |   |   |   | Dijkstra's algorithm
8         |   |   |   | if  $P_i \neq null$  then
9           |   |   |   |   | return  $SO_i$ ;
10        |   |   |   |   |
11        |   |   |   |   |
12        |   |   |   |   |
13        |   |   |   |   |
14        |   |   |   |   |
15        |   |   |   |   |
16        |   |   |   |   |
17        |   |   |   |   |
18        |   |   |   |   |
19        |   |   |   |   |
20        |   |   |   |   |
21        |   |   |   |   |
22        |   |   |   |   |
23        |   |   |   |   |
24        |   |   |   |   |
25        |   |   |   |   |
26        |   |   |   |   |
27        |   |   |   |   |
28        |   |   |   |   |
29        |   |   |   |   |
30        |   |   |   |   |
31        |   |   |   |   |
32        |   |   |   |   |
33        |   |   |   |   |
34        |   |   |   |   |
35        |   |   |   |   |
36        |   |   |   |   |
37        |   |   |   |   |
38        |   |   |   |   |
39        |   |   |   |   |
40        |   |   |   |   |
41        |   |   |   |   |
42        |   |   |   |   |
43        |   |   |   |   |
44        |   |   |   |   |
45        |   |   |   |   |
46        |   |   |   |   |
47        |   |   |   |   |
48        |   |   |   |   |
49        |   |   |   |   |
50        |   |   |   |   |
51        |   |   |   |   |
52        |   |   |   |   |
53        |   |   |   |   |
54        |   |   |   |   |
55        |   |   |   |   |
56        |   |   |   |   |
57        |   |   |   |   |
58        |   |   |   |   |
59        |   |   |   |   |
60        |   |   |   |   |
61        |   |   |   |   |
62        |   |   |   |   |
63        |   |   |   |   |
64        |   |   |   |   |
65        |   |   |   |   |
66        |   |   |   |   |
67        |   |   |   |   |
68        |   |   |   |   |
69        |   |   |   |   |
70        |   |   |   |   |
71        |   |   |   |   |
72        |   |   |   |   |
73        |   |   |   |   |
74        |   |   |   |   |
75        |   |   |   |   |
76        |   |   |   |   |
77        |   |   |   |   |
78        |   |   |   |   |
79        |   |   |   |   |
80        |   |   |   |   |
81        |   |   |   |   |
82        |   |   |   |   |
83        |   |   |   |   |
84        |   |   |   |   |
85        |   |   |   |   |
86        |   |   |   |   |
87        |   |   |   |   |
88        |   |   |   |   |
89        |   |   |   |   |
90        |   |   |   |   |
91        |   |   |   |   |
92        |   |   |   |   |
93        |   |   |   |   |
94        |   |   |   |   |
95        |   |   |   |   |
96        |   |   |   |   |
97        |   |   |   |   |
98        |   |   |   |   |
99        |   |   |   |   |
100       |   |   |   |   |
101       |   |   |   |   |
102       |   |   |   |   |
103       |   |   |   |   |
104       |   |   |   |   |
105       |   |   |   |   |
106       |   |   |   |   |
107       |   |   |   |   |
108       |   |   |   |   |
109       |   |   |   |   |
110       |   |   |   |   |
111       |   |   |   |   |
112       |   |   |   |   |
113       |   |   |   |   |
114       |   |   |   |   |
115       |   |   |   |   |
116       |   |   |   |   |
117       |   |   |   |   |
118       |   |   |   |   |
119       |   |   |   |   |
120       |   |   |   |   |
121       |   |   |   |   |
122       |   |   |   |   |
123       |   |   |   |   |
124       |   |   |   |   |
125       |   |   |   |   |
126       |   |   |   |   |
127       |   |   |   |   |
128       |   |   |   |   |
129       |   |   |   |   |
130       |   |   |   |   |
131       |   |   |   |   |
132       |   |   |   |   |
133       |   |   |   |   |
134       |   |   |   |   |
135       |   |   |   |   |
136       |   |   |   |   |
137       |   |   |   |   |
138       |   |   |   |   |
139       |   |   |   |   |
140       |   |   |   |   |
141       |   |   |   |   |
142       |   |   |   |   |
143       |   |   |   |   |
144       |   |   |   |   |
145       |   |   |   |   |
146       |   |   |   |   |
147       |   |   |   |   |
148       |   |   |   |   |
149       |   |   |   |   |
150       |   |   |   |   |
151       |   |   |   |   |
152       |   |   |   |   |
153       |   |   |   |   |
154       |   |   |   |   |
155       |   |   |   |   |
156       |   |   |   |   |
157       |   |   |   |   |
158       |   |   |   |   |
159       |   |   |   |   |
160       |   |   |   |   |
161       |   |   |   |   |
162       |   |   |   |   |
163       |   |   |   |   |
164       |   |   |   |   |
165       |   |   |   |   |
166       |   |   |   |   |
167       |   |   |   |   |
168       |   |   |   |   |
169       |   |   |   |   |
170       |   |   |   |   |
171       |   |   |   |   |
172       |   |   |   |   |
173       |   |   |   |   |
174       |   |   |   |   |
175       |   |   |   |   |
176       |   |   |   |   |
177       |   |   |   |   |
178       |   |   |   |   |
179       |   |   |   |   |
180       |   |   |   |   |
181       |   |   |   |   |
182       |   |   |   |   |
183       |   |   |   |   |
184       |   |   |   |   |
185       |   |   |   |   |
186       |   |   |   |   |
187       |   |   |   |   |
188       |   |   |   |   |
189       |   |   |   |   |
190       |   |   |   |   |
191       |   |   |   |   |
192       |   |   |   |   |
193       |   |   |   |   |
194       |   |   |   |   |
195       |   |   |   |   |
196       |   |   |   |   |
197       |   |   |   |   |
198       |   |   |   |   |
199       |   |   |   |   |
200       |   |   |   |   |
201       |   |   |   |   |
202       |   |   |   |   |
203       |   |   |   |   |
204       |   |   |   |   |
205       |   |   |   |   |
206       |   |   |   |   |
207       |   |   |   |   |
208       |   |   |   |   |
209       |   |   |   |   |
210       |   |   |   |   |
211       |   |   |   |   |
212       |   |   |   |   |
213       |   |   |   |   |
214       |   |   |   |   |
215       |   |   |   |   |
216       |   |   |   |   |
217       |   |   |   |   |
218       |   |   |   |   |
219       |   |   |   |   |
220       |   |   |   |   |
221       |   |   |   |   |
222       |   |   |   |   |
223       |   |   |   |   |
224       |   |   |   |   |
225       |   |   |   |   |
226       |   |   |   |   |
227       |   |   |   |   |
228       |   |   |   |   |
229       |   |   |   |   |
230       |   |   |   |   |
231       |   |   |   |   |
232       |   |   |   |   |
233       |   |   |   |   |
234       |   |   |   |   |
235       |   |   |   |   |
236       |   |   |   |   |
237       |   |   |   |   |
238       |   |   |   |   |
239       |   |   |   |   |
240       |   |   |   |   |
241       |   |   |   |   |
242       |   |   |   |   |
243       |   |   |   |   |
244       |   |   |   |   |
245       |   |   |   |   |
246       |   |   |   |   |
247       |   |   |   |   |
248       |   |   |   |   |
249       |   |   |   |   |
250       |   |   |   |   |
251       |   |   |   |   |
252       |   |   |   |   |
253       |   |   |   |   |
254       |   |   |   |   |
255       |   |   |   |   |
256       |   |   |   |   |
257       |   |   |   |   |
258       |   |   |   |   |
259       |   |   |   |   |
260       |   |   |   |   |
261       |   |   |   |   |
262       |   |   |   |   |
263       |   |   |   |   |
264       |   |   |   |   |
265       |   |   |   |   |
266       |   |   |   |   |
267       |   |   |   |   |
268       |   |   |   |   |
269       |   |   |   |   |
270       |   |   |   |   |
271       |   |   |   |   |
272       |   |   |   |   |
273       |   |   |   |   |
274       |   |   |   |   |
275       |   |   |   |   |
276       |   |   |   |   |
277       |   |   |   |   |
278       |   |   |   |   |
279       |   |   |   |   |
280       |   |   |   |   |
281       |   |   |   |   |
282       |   |   |   |   |
283       |   |   |   |   |
284       |   |   |   |   |
285       |   |   |   |   |
286       |   |   |   |   |
287       |   |   |   |   |
288       |   |   |   |   |
289       |   |   |   |   |
290       |   |   |   |   |
291       |   |   |   |   |
292       |   |   |   |   |
293       |   |   |   |   |
294       |   |   |   |   |
295       |   |   |   |   |
296       |   |   |   |   |
297       |   |   |   |   |
298       |   |   |   |   |
299       |   |   |   |   |
300       |   |   |   |   |
301       |   |   |   |   |
302       |   |   |   |   |
303       |   |   |   |   |
304       |   |   |   |   |
305       |   |   |   |   |
306       |   |   |   |   |
307       |   |   |   |   |
308       |   |   |   |   |
309       |   |   |   |   |
310       |   |   |   |   |
311       |   |   |   |   |
312       |   |   |   |   |
313       |   |   |   |   |
314       |   |   |   |   |
315       |   |   |   |   |
316       |   |   |   |   |
317       |   |   |   |   |
318       |   |   |   |   |
319       |   |   |   |   |
320       |   |   |   |   |
321       |   |   |   |   |
322       |   |   |   |   |
323       |   |   |   |   |
324       |   |   |   |   |
325       |   |   |   |   |
326       |   |   |   |   |
327       |   |   |   |   |
328       |   |   |   |   |
329       |   |   |   |   |
330       |   |   |   |   |
331       |   |   |   |   |
332       |   |   |   |   |
333       |   |   |   |   |
334       |   |   |   |   |
335       |   |   |   |   |
336       |   |   |   |   |
337       |   |   |   |   |
338       |   |   |   |   |
339       |   |   |   |   |
340       |   |   |   |   |
341       |   |   |   |   |
342       |   |   |   |   |
343       |   |   |   |   |
344       |   |   |   |   |
345       |   |   |   |   |
346       |   |   |   |   |
347       |   |   |   |   |
348       |   |   |   |   |
349       |   |   |   |   |
350       |   |   |   |   |
351       |   |   |   |   |
352       |   |   |   |   |
353       |   |   |   |   |
354       |   |   |   |   |
355       |   |   |   |   |
356       |   |   |   |   |
357       |   |   |   |   |
358       |   |   |   |   |
359       |   |   |   |   |
360       |   |   |   |   |
361       |   |   |   |   |
362       |   |   |   |   |
363       |   |   |   |   |
364       |   |   |   |   |
365       |   |   |   |   |
366       |   |   |   |   |
367       |   |   |   |   |
368       |   |   |   |   |
369       |   |   |   |   |
370       |   |   |   |   |
371       |   |   |   |   |
372       |   |   |   |   |
373       |   |   |   |   |
374       |   |   |   |   |
375       |   |   |   |   |
376       |   |   |   |   |
377       |   |   |   |   |
378       |   |   |   |   |
379       |   |   |   |   |
380       |   |   |   |   |
381       |   |   |   |   |
382       |   |   |   |   |
383       |   |   |   |   |
384       |   |   |   |   |
385       |   |   |   |   |
386       |   |   |   |   |
387       |   |   |   |   |
388       |   |   |   |   |
389       |   |   |   |   |
390       |   |   |   |   |
391       |   |   |   |   |
392       |   |   |   |   |
393       |   |   |   |   |
394       |   |   |   |   |
395       |   |   |   |   |
396       |   |   |   |   |
397       |   |   |   |   |
398       |   |   |   |   |
399       |   |   |   |   |
400       |   |   |   |   |
401       |   |   |   |   |
402       |   |   |   |   |
403       |   |   |   |   |
404       |   |   |   |   |
405       |   |   |   |   |
406       |   |   |   |   |
407       |   |   |   |   |
408       |   |   |   |   |
409       |   |   |   |   |
410       |   |   |   |   |
411       |   |   |   |   |
412       |   |   |   |   |
413       |   |   |   |   |
414       |   |   |   |   |
415       |   |   |   |   |
416       |   |   |   |   |
417       |   |   |   |   |
418       |   |   |   |   |
419       |   |   |   |   |
420       |   |   |   |   |
421       |   |   |   |   |
422       |   |   |   |   |
423       |   |   |   |   |
424       |   |   |   |   |
425       |   |   |   |   |
426       |   |   |   |   |
427       |   |   |   |   |
428       |   |   |   |   |
429       |   |   |   |   |
430       |   |   |   |   |
431       |   |   |   |   |
432       |   |   |   |   |
433       |   |   |   |   |
434       |   |   |   |   |
435       |   |   |   |   |
436       |   |   |   |   |
437       |   |   |   |   |
438       |   |   |   |   |
439       |   |   |   |   |
440       |   |   |   |   |
441       |   |   |   |   |
442       |   |   |   |   |
443       |   |   |   |   |
444       |   |   |   |   |
445       |   |   |   |   |
446       |   |   |   |   |
447       |   |   |   |   |
448       |   |   |   |   |
449       |   |   |   |   |
450       |   |   |   |   |
451       |   |   |   |   |
452       |   |   |   |   |
453       |   |   |   |   |
454       |   |   |   |   |
455       |   |   |   |   |
456       |   |   |   |   |
457       |   |   |   |   |
458       |   |   |   |   |
459       |   |   |   |   |
460       |   |   |   |   |
461       |   |   |   |   |
462       |   |   |   |   |
463       |   |   |   |   |
464       |   |   |   |   |
465       |   |   |   |   |
466       |   |   |   |   |
467       |   |   |   |   |
468       |   |   |   |   |
469       |   |   |   |   |
470       |   |   |   |   |
471       |   |   |   |   |
472       |   |   |   |   |
473       |   |   |   |   |
474       |   |   |   |   |
475       |   |   |   |   |
476       |   |   |   |   |
477       |   |   |   |   |
478       |   |   |   |   |
479       |   |   |   |   |
480       |   |   |   |   |
481       |   |   |   |   |
482       |   |   |   |   |
483       |   |   |   |   |
484       |   |   |   |   |
485       |   |   |   |   |
486       |   |   |   |   |
487       |   |   |   |   |
488       |   |   |   |   |
489       |   |   |   |   |
490       |   |   |   |   |
491       |   |   |   |   |
492       |   |   |   |   |
493       |   |   |   |   |
494       |   |   |   |   |
495       |   |   |   |   |
496       |   |   |   |   |
497       |   |   |   |   |
498       |   |   |   |   |
499       |   |   |   |   |
500       |   |   |   |   |
501       |   |   |   |   |
502       |   |   |   |   |
503       |   |   |   |   |
504       |   |   |   |   |
505       |   |   |   |   |
506       |   |   |   |   |
507       |   |   |   |   |
508       |   |   |   |   |
509       |   |   |   |   |
510       |   |   |   |   |
511       |   |   |   |   |
512       |   |   |   |   |
513       |   |   |   |   |
514       |   |   |   |   |
515       |   |   |   |   |
516       |   |   |   |   |
517       |   |   |   |   |
518       |   |   |   |   |
519       |   |   |   |   |
520       |   |   |   |   |
521       |   |   |   |   |
522       |   |   |   |   |
523       |   |   |   |   |
524       |   |   |   |   |
525       |   |   |   |   |
526       |   |   |   |   |
527       |   |   |   |   |
528       |   |   |   |   |
529       |   |   |   |   |
530       |   |   |   |   |
531       |   |   |   |   |
532       |   |   |   |   |
533       |   |   |   |   |
534       |   |   |   |   |
535       |   |   |   |   |
536       |   |   |   |   |
537       |   |   |   |   |
538       |   |   |   |   |
539       |   |   |   |   |
540       |   |   |   |   |
541       |   |   |   |   |
542       |   |   |   |   |
543       |   |   |   |   |
544       |   |   |   |   |
545       |   |   |   |   |
546       |   |   |   |   |
547       |   |   |   |   |
548       |   |   |   |   |
549       |   |   |   |   |
550       |   |   |   |   |
551       |   |   |   |   |
552       |   |   |   |   |
553       |   |   |   |   |
554       |   |   |   |   |
555       |   |   |   |   |
556       |   |   |   |   |
557       |   |   |   |   |
558       |   |   |   |   |
559       |   |   |   |   |
560       |   |   |   |   |
561       |   |   |   |   |
562       |   |   |   |   |
563       |   |   |   |   |
564       |   |   |   |   |
565       |   |   |   |   |
566       |   |   |   |   |
567       |   |   |   |   |
568       |   |   |   |   |
569       |   |   |   |   |
570       |   |   |   |   |
571       |   |   |   |   |
572       |   |   |   |   |
573       |   |   |   |   |
574       |   |   |   |   |
575       |   |   |   |   |
576       |   |   |   |   |
577       |   |   |   |   |
578       |   |   |   |   |
579       |   |   |   |   |
580       |   |   |   |   |
581       |   |   |   |   |
582       |   |   |   |   |
583       |   |   |   |   |
584       |   |   |   |   |
585       |   |   |   |   |
586       |   |   |   |   |
587       |   |   |   |   |
588       |   |   |   |   |
589       |   |   |   |   |
590       |   |   |   |   |
591       |   |   |   |   |
592       |   |   |   |   |
593       |   |   |   |   |
594       |   |   |   |   |
595       |   |   |   |   |
596       |   |   |   |   |
597       |   |   |   |   |
598       |   |   |   |   |
599       |   |   |   |   |
600       |   |   |   |   |
601       |   |   |   |   |
602       |   |   |   |   |
603       |   |   |   |   |
604       |   |   |   |   |
605       |   |   |   |   |
606       |   |   |   |   |
607       |   |   |   |   |
608       |   |   |   |   |
609       |   |   |   |   |
610       |   |   |   |   |
611       |   |   |   |   |
612       |   |   |   |   |
613       |   |   |   |   |
614       |   |   |   |   |
615       |   |   |   |   |
616       |   |   |   |   |
617       |   |   |   |   |
618       |   |   |   |   |
619       |   |   |   |   |
620       |   |   |   |   |
621       |   |   |   |   |
622       |   |   |   |   |
623       |   |   |   |   |
624       |   |   |   |   |
625       |   |   |   |   |
626       |   |   |   |   |
627       |   |   |   |   |
628       |   |   |   |   |
629       |   |   |   |   |
630       |   |   |   |   |
631       |   |   |   |   |
632       |   |   |   |   |
633       |   |   |   |   |
634       |   |   |   |   |
635       |   |   |   |   |
636       |   |   |   |   |
637       |   |   |   |   |
638       |   |   |   |   |
639       |   |   |   |   |
640       |   |   |   |   |
641       |   |   |   |   |
642       |   |   |   |   |
643       |   |   |   |   |
644       |   |   |   |   |
645       |   |   |   |   |
646       |   |   |   |   |
647       |   |   |   |   |
648       |   |   |   |   |
649       |   |   |   |   |
650       |   |   |   |   |
651       |   |   |   |   |
652       |   |   |   |   |
653       |   |   |   |   |
654       |   |   |   |   |
655       |   |   |   |   |
656       |   |   |   |   |
657       |   |   |   |   |
658       |   |   |   |   |
659       |   |   |   |   |
660       |   |   |   |   |
661       |   |   |   |   |
662       |   |   |   |   |
663       |   |   |   |   |
664       |   |   |   |   |
665       |   |   |   |   |
666       |   |   |   |   |
667       |   |   |   |   |
668       |   |   |   |   |
669       |   |   |   |   |
670       |   |   |   |   |
671       |   |   |   |   |
672       |   |   |   |   |
673       |   |   |   |   |
674       |   |   |   |   |
675       |   |   |   |   |
676       |   |   |   |   |
677       |   |   |   |   |
678       |   |   |   |   |
679       |   |   |   |   |
680       |   |   |   |   |
681       |   |   |   |   |
682       |   |   |   |   |
683       |   |   |   |   |
684       |   |   |   |   |
685       |   |   |   |   |
686       |   |   |   |   |
687       |   |   |   |   |
688       |   |   |   |   |
689       |   |   |   |   |
690       |   |   |   |   |
691       |   |   |   |   |
692       |   |   |   |   |
693       |   |   |   |   |
694       |   |   |   |   |
695       |   |   |   |   |
696       |   |   |   |   |
697       |   |   |   |   |
698       |   |   |   |   |
699       |   |   |   |   |
700       |   |   |   |   |
701       |   |   |   |   |
702       |   |   |   |   |
703       |   |   |   |   |
704       |   |   |   |   |
705       |   |   |   |   |
706       |   |   |   |   |
707       |   |   |   |   |
708       |   |   |   |   |
709       |   |   |   |   |
710       |   |   |   |   |
711       |   |   |   |   |
712       |   |   |   |   |
713       |   |   |   |   |
714       |   |   |   |   |
715       |   |   |   |   |
716       |   |   |   |   |
717       |   |   |   |   |
718       |   |   |   |   |
719       |   |   |   |   |
720       |   |   |   |   |
721       |   |   |   |   |
722       |   |   |   |   |
723       |   |   |   |   |
724       |   |   |   |   |
725       |   |   |   |   |
726       |   |   |   |   |
727       |   |   |   |   |
728       |   |   |   |   |
729       |   |   |   |   |
730       |   |   |   |   |
731       |   |   |   |   |
732       |   |   |   |   |
733       |   |   |   |   |
734       |   |   |   |   |
735       |   |   |   |   |
736       |   |   |   |   |
737       |   |   |   |   |
738       |   |   |   |   |
739       |   |   |   |   |
740       |   |   |   |   |
741       |   |   |   |   |
742       |   |   |   |   |
743       |   |   |   |   |
744       |   |   |   |   |
745       |   |   |   |   |
746       |   |   |   |   |
747       |   |   |   |   |
748       |   |   |   |   |
749       |   |   |   |   |
750       |   |   |   |   |
751       |   |   |   |   |
752       |   |   |   |   |
753       |   |   |   |   |
754       |   |   |   |   |
755       |   |   |   |   |
756       |   |   |   |   |
757       |   |   |   |   |
758       |   |   |   |   |
759       |   |   |   |   |
760       |   |   |   |   |
761       |   |   |   |   |
762       |   |   |   |   |
763       |   |   |   |   |
764       |   |   |   |   |
765       |   |   |   |   |
766       |   |   |   |   |
767       |   |   |   |   |
768       |   |   |   |   |
769       |   |   |   |   |
770       |   |   |   |   |
771       |   |   |   |   |
772       |   |   |   |   |
773       |   |   |   |   |
774       |   |   |   |   |
775       |   |   |   |   |
776       |   |   |   |   |
777       |   |   |   |   |
778       |   |   |   |   |
779       |   |   |   |   |
780       |   |   |   |   |
781       |   |   |   |   |
782       |   |   |   |   |
783       |   |   |   |   |
784       |   |   |   |   |
785       |   |   |   |   |
786       |   |   |   |   |
787       |   |   |   |   |
788       |   |   |   |   |
789       |   |   |   |   |
790       |   |   |   |   |
791       |   |   |   |   |
792       |   |   |   |   |
793       |   |   |   |   |
794       |   |   |   |   |
795       |   |   |   |   |
796       |   |   |   |   |
797       |   |   |   |   |
798       |   |   |   |   |
799       |   |   |   |   |
800       |   |   |   |   |
801       |   |   |   |   |
802       |   |   |   |   |
803       |   |   |   |   |
804       |   |   |   |   |
805       |   |   |   |   |
806       |   |   |   |   |
807       |   |   |   |   |
808       |   |   |   |   |
809       |   |   |   |   |
810       |   |   |   |   |
811       |   |   |   |   |
812       |   |   |   |   |
813       |   |   |   |   |
814       |   |   |   |   |
815       |   |   |   |   |
816       |   |   |   |   |
817       |   |   |   |   |
818       |   |   |   |   |
819       |   |   |   |   |
820       |   |   |   |   |
821       |   |   |   |   |
822       |   |   |   |   |
823       |   |   |   |   |
824       |   |   |   |   |
825       |   |   |   |   |
826       |   |   |   |   |
827       |   |   |   |   |
828       |   |   |   |   |
829       |   |   |   |   |

```

| | | | | |
|--|-----------|-----------|----------|---|
| op (1) | htype (1) | hlen (1) | hops (1) | op: Message type. 1 = REQUEST (for STB_REQUEST), 2 = REPLY (ST_OFFER) option: [MessageInfos="STB_REQUEST" ST_OFFER", runtime, ...] htype, hlen: Hardware address type and length of an applicant client. hops: Number of relay agents a request message traveled. xid: Transaction ID, a random number chosen by the runtime server to identify an offer selection secs: Filled in by the runtime server, the number of seconds elapsed since this server's offer has been selected. Currently this field is reserved and set to 0. flags: The leftmost bit is the BROADCAST (B) flag. If this flag is set to 0, the congested server send a reply back by unicast; if this flag is set to 1, the congested server send a reply back by broadcast. The remaining bits of the flags field are reserved for future use. The numbers in brackets indicate the size of each field in bytes |
| xid (4) | | | | |
| secs (2) | | flags (2) | | |
| ciaddr (4) (applicant server IP address) | | | | |
| siaddr (4) (congested server IP address) | | | | |
| chaddr (16) (applicant server IP address) | | | | |
| sname (64) (congested server host name for which an ST_OFFER is created) | | | | |
| file (128) (for extra data) | | | | |
| options (variable) | | | | |

Fig. 3: STB_REQUEST and ST_OFFER packets structure.

S_i sends a job request to S_{init} . If S_{init} accepts the incoming TCP request, it scans its job queue and performs the job release. The IP packet generated by S_{init} is then routed to the nearest OpenFlow switch. The packet header contains the address of the destination server in the destination field. This switch uses the routing information (FIB) previously provided by the controller to redirect the packet to the appropriate output port. Figure 4 illustrates this process.

If S_{init} rejects the TCP connection, WoS-CoMS allows S_i to abort the request and notify the controller. As a result, the controller can retrieve the task without any authorization from S_{init} . This steal cannot be performed by S_i due to its accreditation level, which is similar to S_{init} . The controller, because of its high privilege levels on supervised switches and terminals, can forcibly recover the task even in different environments and operating systems. Thus, the steal takes place asynchronously without disturbing S_{init} , which may be running user tasks. Figure 5 shows our stealing strategy details.

3.2.2 Unsupervised Work stealing method

The congested server (S_{init}) sends a STB_REQUEST to all the servers in the network, looking for a valid candidate. Each server receiving a STB_REQUEST message builds and sends a ST_OFFER response message detailing its offer. Each received ST_OFFER are queued. After a delay, the source server S_{init} launches algorithm 2 to select a valid offer among those received. At the same time, the same server S_{init} keeps processing the user requests queue in order to reduce latency, disrupted by the congestion state. When a server S_i is selected, S_{init} opens a TCP port enabling tasks to be stolen, and notifies S_i . Then, S_i performs the stealing process of figure 6.

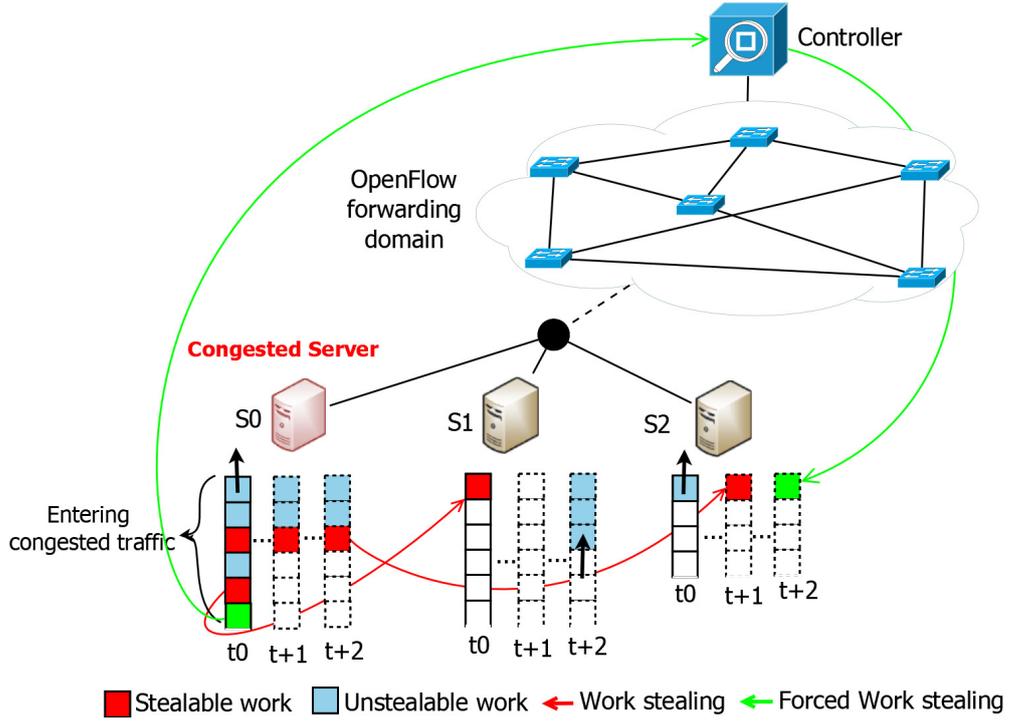


Fig. 4: Our global task stealing process

Algorithm 2: S_{init} 's unsupervised backup server selection algorithm.

input : List of ST_OFFER LO ,
Task to Steal ST ,
network graph G

output: A valid ST_OFFER

- 1 Let SO be an empty ST_OFFER;
- 2 **if** $empty(LO)$ **then**
- 3 **return** SO ;
- 4 **else**
- 5 **foreach** $SO_i \in LO$ **do**
- 6 **if** $SO_i \rightarrow proposed_runtime \approx 0$ **then**
- 7 **return** SO_i ;

3.3 Server lookup area

A stolen task can be processed by the server to which it is assigned, or recursively by another server. However, recursivity in the allocation of a stolen task to another

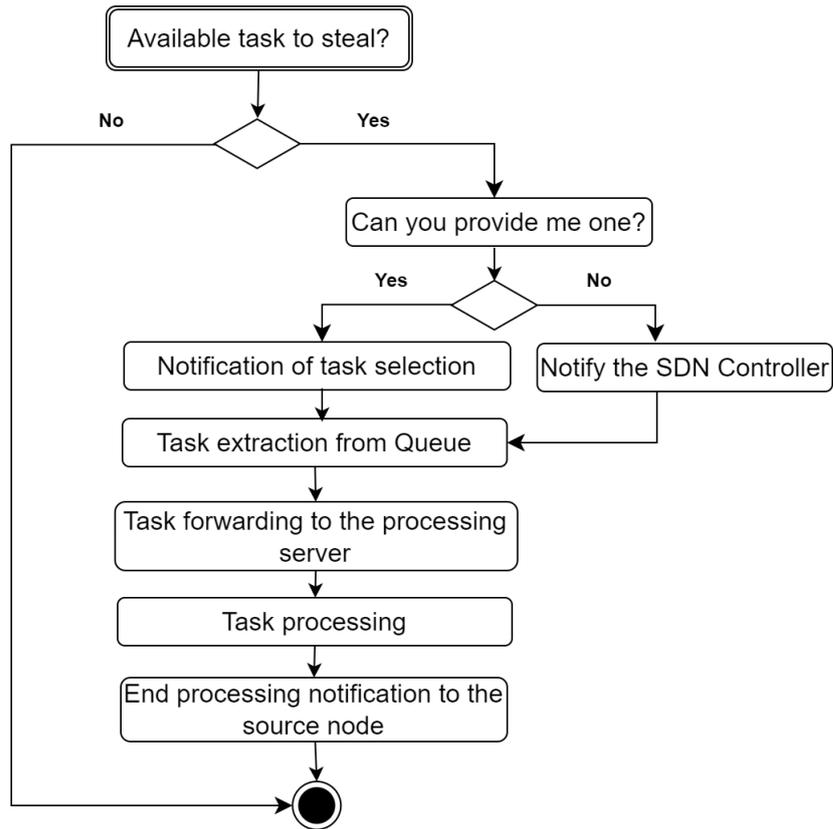


Fig. 5: Detailed stealing process between two servers in supervised method.

can compromise the performance the task process performance [9]. Therefore, before stealing any task, the stealing algorithm executed by the controller checks that the following constraints are met:

- there is at least one server with an empty execution queue;
- there is at least one available routing path for the task;
- the runtime server has enough resources for tasks processing;
- Once a stolen task has been received, it must be prioritized over subsequent ones;
- only servers with a positive previous performance record can receive newly stolen tasks.

In the runtime server lookup, we consider two areas: the intra-server level and the inter-server level. At the intra-server level, theft takes place between the physical server and its virtual machines, or between virtual machines on the same physical server. At this level, the virtual servers are all located at the same hop and their activity is supervised by the hypervisor, enabling to observe the performance metrics of each virtual server. A stolen task is assigned to the virtual server with the lowest additional resource requirements (CPU, RAM, storage).

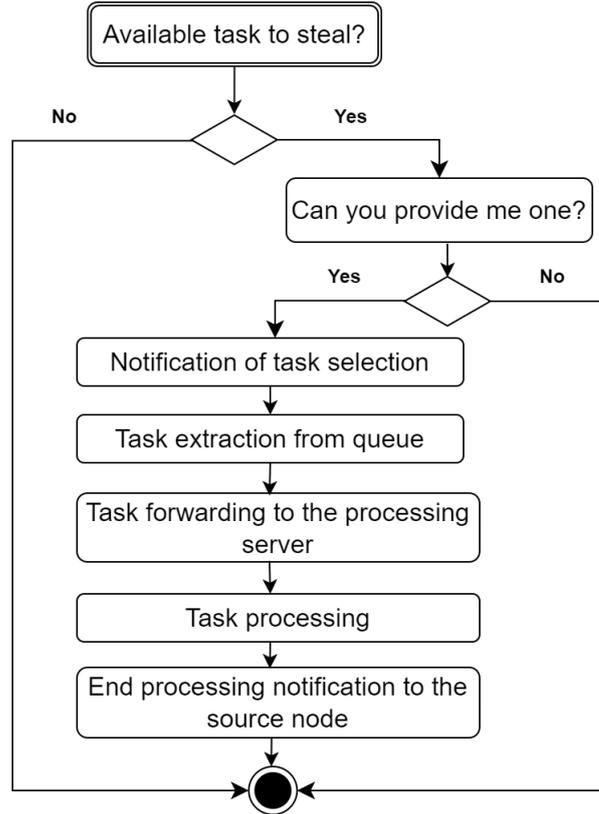


Fig. 6: Detailed stealing process between two servers in unsupervised method.

At the inter-server level, the controller needs to find the server that meets the above criteria, as well as the best path to transfer the job. The best path is the one which provides the minimum transfer time through the fewest possible hops. A minimum number of hops reduces the number of switches involved in the process, and also potentially limits cases of traffic congestion within OpenFlow switches.

4 Simulation results and discussion

In order to investigate the effectiveness of our stealing strategy, we carried out simulations in the OMNet++ version 5 environment running on an Intel Inspiron 2.70Ghz quad-core processor computer with 16GB of memory and a Linux 21.10 operating system. We assumed topologies with two(02), four(04), eight(08), twenty(20), thirty(30) and fifty(50) servers for scalability requirements.

The simulations focused on congested servers and the controller’s workload ramp-up during stealing (number of instructions executed). We also compared our WoS-CoMS task stealing solution with the server replacement scheme proposed by [7] and

the Handler pinning stealing algorithm [20]. Each server uses a 1MB list (queue) to store excessive requests, sources of congestion.

4.1 Server performance

To appreciate the performance of the different data servers, we ran the simulations sequentially on a 60-nodes network under the following respective conditions: without a stealing algorithm, in the presence of the Handler pinning algorithm, in the presence of the FSB-DReViSeR bandwidth replacement algorithm and finally with our WoS-CoMS task stealing algorithm. Traffic was generated according to a Poisson distribution to highlight the randomness of users' requests. Five servers were selected to study their reaction.

Figure 7 shows that WoS-CoM performs better than other approaches in reducing the activity of a congested server from 12% to 22% of the exceeding requests that cause congestion. This is achieved because of the lower granularity of stolen works than those involved in server replacement. Indeed, repositioning moves all user requests and service data to a faster working server with more resources.

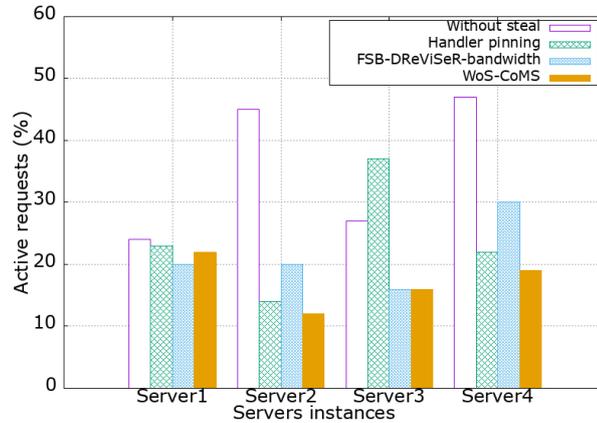


Fig. 7: Individual server performances.

4.2 Stealing ratio

Figure 8 shows that WoS-CoMS steals more jobs on average than Handler pinning [20] on a large number of servers. This is because the controller's presence enable to perform some stealing instances that can't take place only between a source server and a target server. This is the case, for example, when the sender's TCP connections to retrieve the task are not accepted by the source host. In many stealing cases, the Controller's involvement can be used to forcibly retrieve resistant tasks. However, this action does not always end with the task processing.

Evaluation of the stealing ratios provide the data in table 2. Figure 9 shows the results of successful steals compared with predictions. Globally, the stolen job losses

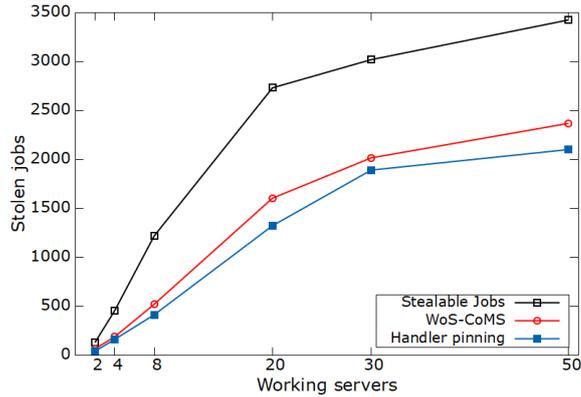


Fig. 8: Performed Stealing ratio.

are almost nil in the presence of a low number of servers, against the case where there are higher. Indeed, a large number of servers potentially implies a higher number of users and traffic. As a result, incoming TCP connection failures become significant. However, the number of stealing failures with WoS-CoMS in the presence of a large number of servers is still lower (0-5%) than with the Handler pinning heuristic (0-9%), thanks to controller involvement. The counterpart of this action is an increase in controller latency in the overall request processing, as shown in figure 10.

Table 2: Stealing performance ratio

| Stealing Schemes | WoS-CoMS | | | Handler pinning | | |
|------------------|----------|-------|--------|-----------------|-------|--------|
| | Workers | Brute | Forced | Ratio (%) | Brute | Forced |
| 2 | 67 | 65 | 0 | 42 | 42 | 0 |
| 4 | 190 | 180 | 5 | 163 | 163 | 0 |
| 8 | 521 | 517 | 1 | 414 | 385 | 8 |
| 20 | 1604 | 1584 | 2 | 1323 | 1270 | 4 |
| 30 | 2015 | 1936 | 0 | 1890 | 1713 | 9 |
| 50 | 2367 | 2321 | 2 | 2100 | 2011 | 4 |

4.3 Impact of task granularity

We evaluated the influence of granularity on our stealing strategy using the following grain levels: 5, 25 and 35. Figures 11a, 11b and 11c show the congestion rate reduction within a few servers with granularities 5, 25 and 35 respectively. We note that under mixed congestion conditions with non-zero granularity requests, the different decongestion approaches offer performance close to that of servers in the absence of task stealing or replacement strategy. This convergence increases when granularity is important. Indeed, multiple interdependent requests reduces the number of stealing

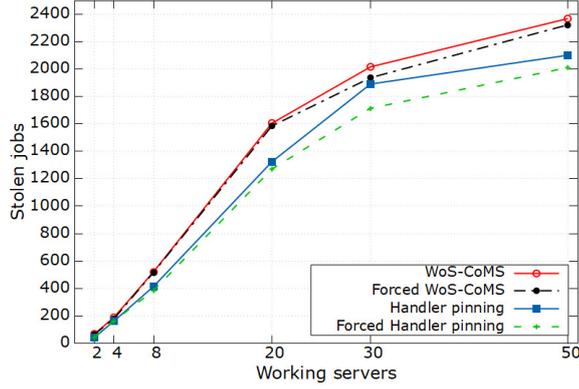


Fig. 9: Completed steals vs. confirmed steals

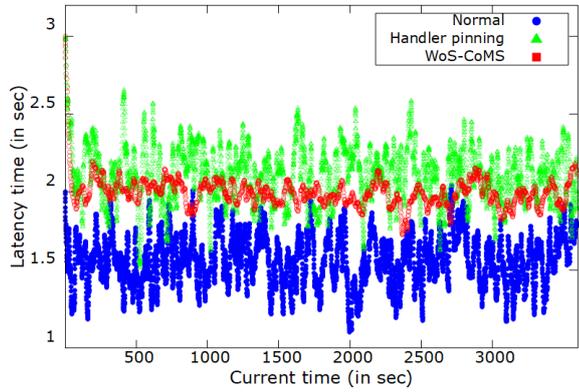


Fig. 10: Increasing controller workload.

instances performed and the ability to find suitable servers with enough resources supporting the weight of these dependent requests.

5 Conclusion and future work

The main motivation for this work was to provide a solution to server congestion in SDN-based network architectures. This network paradigm is highly sought-after today by major operators for their large-scale network management. The key feature of this approach is the ability to successfully mitigate some task stealing rejections, which are often behind low decongestion rates. This is achieved through a controller-based approach to these rejection cases, which easily outperforms other works in the literature by reducing rejection rates to a maximum of 5% and congestion rates to 22%. In the future, we intend to include a machine learning feature in our protocol to predict server availability.

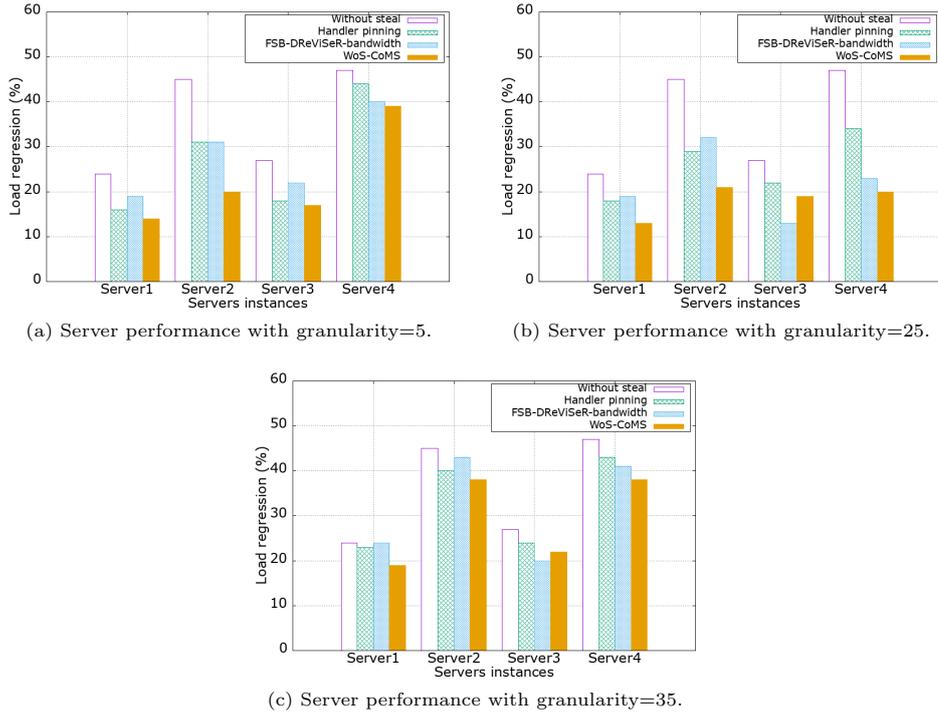


Fig. 11: Server performance vs Granularity factor.

Declaration

Funding

No funding was received for conducting this study.

Conflict of interest/Competing interests

The authors have no competing or conflict of interests to declare that are relevant to the content of this article.

Authors's contributions

Yannick Florian Yankam conceptualize this study, setup the methodology, carried out analysis and performed the experiments. Vianney Kengne Tchendji and Yannick Florian Yankam wrote the first draft of this work and worked for the revised version. Jean Frédéric Myoupo supervised this study and worked on the revised version. In addition, all authors read and approved the work.

References

- [1] Zhou, Y., Ren, B., Xie, J., Luo, L., Guo, D., Zhou, X.: Enable the proactively load-balanced control plane for sdn via intelligent switch-to-controller selection strategy. *Computer Networks* **233**, 109867 (2023) <https://doi.org/10.1016/j.comnet.2023.109867>
- [2] Liatifis, A., Sarigiannidis, P., Argyriou, V., Lagkas, T.: Advancing sdn from open-flow to p4: A survey. *ACM Comput. Surv.* **55**(9) (2023) <https://doi.org/10.1145/3556973>
- [3] Nadeau, T.D., Gray, K.: *SDN: Software Defined Networks: An Authoritative Review of Network Programmability Technologies*, pp. 9–46. "O'Reilly Media, Inc.", 1005 Gravenstein Highway North, Sebastopol, CA 95472 (2013)
- [4] Mushtaq, A., Mittal, R., McCauley, J., Alizadeh, M., Ratnasamy, S., Shenker, S.: Datacenter congestion control: Identifying what is essential and making it practical **49**(3), 32–38 (2019) <https://doi.org/10.1145/3371927.3371932>
- [5] Horiuchi, S., Tachibana, T.: Dynamic replacement of virtual service resources based on tree topology for mobile users in virtual networks. *Journal of Computers* **13**(12), 1335–1349 (2018) <https://doi.org/10.17706/jcp.13.12.1335-1348>
- [6] Myoupo, J.F., Yankam, Y.F., Tchendji, V.K.: On the dynamic replacement of virtual service resources for mobile users in virtual networks. *J. Comput.* **15**(1), 10–21 (2020)
- [7] Myoupo, J.F., Kengne Tchendji, V., Yankam, Y.F., Tagne, J.C., Tan, L.: Fsb-dreviser: Flow splitting-based dynamic replacement of virtual service resources for mobile users in virtual heterogeneous networks. *J. Comput. Netw. Commun.* **2020** (2020) <https://doi.org/10.1155/2020/8891481>
- [8] Seddiki, M.S.: *Allocation dynamique des ressources et gestion de la qualité de service dans la virtualisation des réseaux*. PhD thesis, Université de Lorraine (2015)
- [9] Nakashima, R., Yasugi, M., Yoritaka, H., Hiraishi, T., Umatani, S.: Work-stealing strategies that consider work amount and hierarchy. *Journal of Information Processing* **29**, 478–489 (2021) <https://doi.org/10.2197/ipsjjip.29.478>
- [10] Li, J., Agrawal, K., Elnikety, S., He, Y., Lee, I.-T.A., Lu, C., McKinley, K.S.: Work stealing for interactive services to meet target latency. *SIGPLAN Not.* **51**(8) (2016) <https://doi.org/10.1145/3016078.2851151>
- [11] Lin, C.-X., Huang, T.-W., Wong, M.D.F.: An efficient work-stealing scheduler for task dependency graph. In: 2020 IEEE 26th International Conference on Parallel and Distributed Systems (ICPADS), pp. 64–71 (2020). <https://doi.org/10.1109/ICPADS51040.2020.00018>

- [12] Larkins, D.B., Snyder, J., Dinan, J.: Accelerated work stealing. In: Proceedings of the 48th International Conference on Parallel Processing. ICCP '19. Association for Computing Machinery, New York, NY, USA (2019). <https://doi.org/10.1145/3337821.3337878>
- [13] Kehrer, S., Blochinger, W.: A survey on cloud migration strategies for high performance computing. In: Proceedings of the 13th Symposium and Summer School on Service-Oriented Computing (SummerSoc19), pp. 57–69 (2019). IBM Research Division
- [14] Nunes, B.A.A., Mendonca, M., Nguyen, X.-N., Obraczka, K., Turletti, T.: A survey of software-defined networking: Past, present, and future of programmable networks. *IEEE Communications Surveys & Tutorials* **16**(3), 1617–1634 (2014)
- [15] Servin, C.: Réseaux et Télécoms-4ème édition, pp. 188–190. Dunod, Paris (2013)
- [16] Tanaka, D., Kawarasaki, M.: Congestion control in named data networking. In: 2016 IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN), pp. 1–6 (2016). <https://doi.org/10.1109/LANMAN.2016.7548848>
- [17] Andjamba, T.S., Zodi, G.-A.L.: A load balancing protocol for improved video on demand in sdn-based clouds. In: 2023 17th International Conference on Ubiquitous Information Management and Communication (IMCOM), pp. 1–6 (2023). <https://doi.org/10.1109/IMCOM56909.2023.10035591>
- [18] Alwada'n, T., Al-Tamimi, A.-K., Mohammad, A., Salem, M., Muhammad, Y.: Dynamic congestion management system for cloud service broker. *International Journal of Electrical and Computer Engineering (IJECE)* **13**(1), 872–883 (2023). This is an open access article licenced under a CC-BY-SA license, <https://creativecommons.org/licenses/by-sa/4.0/>
- [19] Marchal, X., Graff, P., Ky, J.R., Cholez, T., Tuffin, S., Mathieu, B., Festor, O.: An analysis of cloud gaming platforms behaviour under synthetic network constraints and real cellular networks conditions. *J. Netw. Syst. Manage.* **31**(2) (2023) <https://doi.org/10.1007/s10922-023-09720-9>
- [20] Gaud, F., Geneves, S., Mottet, F.: Vol de tâches efficace pour systèmes événementiels multi-cœurs. In: Conférence Française en Systèmes d'Exploitation (CFSE'7) (2009)
- [21] Zeldovich, N., Yip, A., Dabek, F., Morris, R.T., Mazières, D.: Multiprocessor support for Event-Driven programs. In: 2003 USENIX Annual Technical Conference (USENIX ATC 03). USENIX Association, San Antonio, TX (2003). <https://www.usenix.org/conference/2003-usenix-annual-technical-conference/multiprocessor-support-event-driven-programs>
- [22] Shah, D., Xie, Q.: Centralized congestion control and scheduling in a datacenter.

arXiv preprint arXiv:1710.02548 (2017)

- [23] Chen, J., Manivannan, M., Abduljabbar, M., Pericàs, M.: Erase: Energy efficient task mapping and resource management for work stealing runtimes. *ACM Trans. Archit. Code Optim.* **19**(2) (2022) <https://doi.org/10.1145/3510422>
- [24] Tchendji, V.K., Yankam, Y.F.: Dynamic resource allocations in virtual networks through a knapsack problem's dynamic programming solution. *Revue Africaine de Recherche en Informatique et Mathématiques Appliquées* **Volume 31 - 2019 - CARI 2018** (2020) <https://doi.org/10.46298/arima.5321>

Yannick Florian Yankam is an Assistant Lecturer of Computer Science in the department of Telecommunication and Network engineering at the University of Dschang, Dschang, Cameroon. He received his PhD in computer science from the University of Dschang in June 2021. His current research interests focused on management of networks and services, resource management in wired and wireless networks applied in network virtualization, IoT, software-defined networks and cloud computing.

Vianney Kengne Tchendji is Associate Professor of Computer Science at the University of Dschang, Dschang, Cameroon. He received his PhD in computer science from the University of Picardie-Jules Verne, Amiens, France, in June 2014. Prof. Kengne Tchendji has served as member committee of international conferences, and reviewer for many international journals in computer science. His current research interests include parallel algorithms and architectures, network management and cybersecurity.

Jean Frédéric Myoupo is Professor of computer science in the University of Picardie-Jules Verne, Amiens, France since 1994, where he heads the parallel and Mobile computing research group in the Computer Science Lab-MIS. Professor Myoupo is the former dean of the Faculty of Mathematics and Computer Science of UPJV from 1999 to 2002. He received his Ph.D. in applied mathematics from the Paul Sabatier University of Toulouse, France in 1983, and his Habilitation in Computer science from the University of Paris 11, Orsay, France, in 1994. He held many positions in mathematics and computer science departments such as lecturer or Associate Professor in different universities, such as the University of Sherbrooke, Québec, Canada, the University of Yaounde, Cameroon, the University of Paris 11, Orsay, France and the University of Rouen, France. Professor Myoupo has served as member of program committee of international conferences as PDPTA, CIC, OPODIS, IPDPS workshops, HPCS, AP2PS, WINSYS, ICNS, ICWN, IEEE-RIVEF, IEEE ISSPIT, ISCA-PDCS. He is an Associate Editor of "ISCA International Journal on Computers and their Applications" and a Member of the editorial board

of "Studia Informatica Universalis". His current research interests include parallel algorithms and architectures, mobile computing and network management.