# Minimal Coflow Routing and Scheduling in OpenFlow-based Cloud Storage Area Networks

Chui-Hui Chiu, Dipak Kumar Singh, Qingyang Wang, Kisung Lee, Seung-Jong Park
Division of Computer Science and Engineering, Center for Computation and Technology
Louisiana State University, Baton Rouge, LA, USA 70803
E-mail: {cchiu1, dsingh8, qwang26, klee76, sjpark}@lsu.edu

***Abstract*—Researches affirm that coflow scheduling/routing substantially shortens the average application inner communication time in data center networks(DCNs). The commonly desirable critical features of existing coflow scheduling/routing framework includes (1) coflow scheduling, (2) coflow routing, and (3) per-flow rate-limiting. However, to provide the 3 features, existing frameworks require customized computing frameworks, customized operating systems, or specific external commercial monitoring frameworks on software-defined networking(SDN) switches. These requirements defer or even prohibit the deployment of coflow scheduling/routing in production DCNs. In this paper, we design a coflow scheduling and routing framework, `MinCOF` which has minimal requirements on hosts and switches for cloud storage area networks(SANs) based on OpenFlow SDN. `MinCOF` accommodates all critical features of coflow scheduling/routing from previous works. The deployability in production environment is especially taken into consideration. The OpenFlow architecture is capable of processing the traffic load in a cloud SAN. Not necessary requirements for hosts from existing frameworks are migrated to the mature commodity OpenFlow 1.3 Switch and our coflow scheduler. Transfer applications on hosts only need slight enhancements on their existing connection establishment and progress reporting functions. Evaluations reveal that `MinCOF` decreases the average coflow completion time (CCT) by 12.94% compared to the latest OpenFlow-based coflow scheduling and routing framework.**

***Index Terms*—Application-aware Networks; Data Center Networks; Software-defined Networking; OpenFlow; Coflow Scheduling; Storage Area Networks; Cloud Computing**

## I. Introduction

Nowadays, researchers widely utilize the enormous computational power of computer clusters to analyze big data. Many cluster computing frameworks [9], [15] emerge to unify and simplify the development of the analytic cluster applications. These frameworks transform application logics into an iteration of alternating and mutual blocking computing and communication stages (e.g., MapReduce jobs). In the communication stage, typically, multiple hosts exchange the result from the previous computing stage with multiple other hosts using multiple communication flows. The computing framework suspends the next computing stage until the all communication flows finish.

Coflow scheduling [3], [5] is introduced to optimize the network usage and help shorten the completion time of the communication stage. It bundles communication flows into **coflow**s according to their logical semantics in the application programs and sophisticatedly allocates network resources to coflows for shortening the overall completion time of all associated communication flows. Shortening the coflow completion time(CCT), from the beginning of the first associated flow to the ending of the last associated flow, shortens the communication stage. If a cluster application spends a substantial portion of time in communication stages [4] (e.g., intermediate data sizes of a MapReduce job is large), coflow scheduling shortens the application completion time. Successive researches incorporate network routing to coflow scheduling [19]. The outcome coflow scheduling and routing framework has even better performance.

By investigating existing coflow scheduling/routing frameworks, we summarize 3 common critical correlated features as follows.

1) **Coflow Scheduling:** Determining the order of coflows and the rate of each flow in coflows to transmit according to the coflow information synchronized from all hosts.
2) **Coflow Routing:** Determining the path for each flow in coflows to transmit.
3) **Per-flow Rate-limiting:** Enforcing the rate allocation to each flow by the coflow scheduler to prevent individual flows from aggressively seizing network bandwidth.

Some existing frameworks provide the 3 features by customizing operating systems(OSs) on hosts to synchronize coflow information to the coflow scheduler and limit flows′ rates such as Varys [5] and Rapier [19]. This strategy may prevent the adoption of coflow scheduling/routing in a DCN if the hosts require another customized OS which results in incompatibility issues. Other existing frameworks rely on external commercial monitoring tools on switches to synchronize coflow information such as Tailor [13] which requires the proprietary sFlow-RT [8]. However, deploying commercial tools in a production environment is likely to cost additional budget. Tab. I summarizes the features of existing frameworks. Details are included in Sec. V.

To avoid all the obstacles to deploying coflow scheduling/routing in production, we design, `MinCOF`, a coflow scheduling and routing framework which imposes minimal requirements on hosts and thoroughly takes advantage of the standardized mature commodity OpenFlow Switches. `MinCOF` migrates all unnecessary requirements away from the switch/host and efficiently provide them using the OpenFlow 1.3 Switch and our coflow scheduler (III). `MinCOF`

TABLE I: Comparison of features of coflow scheduling/routing frameworks.

| Framework | Coflow Scheduling | Coflow Routing | Per-flow Rate-limiting | Unmod. Host OS | Coflow Info. Sync. |
|---|---|---|---|---|---|
| Varys[5] | V | | V | | 2-way |
| Rapier[19] | V | V | V | | 2-way |
| Tailor[13] | Limited | 1 flow | | V | Proprietary |
| MinCOF | V | 1 coflow | V | V | 1-way |

is especially suitable for cloud SANs storing big data. For example, the scientific cloud object storage [12], [11] involves massive coflow communication scenarios (II), and its big data transmissions which utilize few concurrent long flows can be processed by current OpenFlow SDN implementations. Widely used transfer applications in cloud SANs only have to augment small segments of code to their original connection establishment and progress reporting functions to be integrated.

We measure the performance of `MinCOF` on our DCN testbed with physical software OpenFlow Switches and hosts. The results illustrate that `MinCOF` largely outperforms the existing framework without the routing feature (Varys) and further shortens the average CCT by up to 12.94% compared to the existing framework without rate limiting (Tailor). `MinCOF` also maintains the lead under various workload compositions and network topologies.

The rest of the paper is organized as follows. Section II provides the reasons which motivate us to create an OpenFlow-based coflow scheduling and routing framework for cloud SANs. Section III explains the architecture of `MinCOF` in details. Section IV compares the performance of `MinCOF` to that of existing similar frameworks by conducting experiments on real hardware. Section V distinguishes our work from previous works. Section VI concludes and gives an insight of future development.

## II. Motivation

Two observations motivate us to create a new coflow scheduling and routing framework for OpenFlow-based cloud SANs, (1) the cloud SAN has high similarity to the DCN for cluster computing in which existing coflow scheduling/routing frameworks operate, (2) the OpenFlow SDN is suitable for processing the traffic load intensity of the cloud SAN.

For the similarity of cloud SAN and DCN for cluster computing, both networks have similar underlying DCN topologies and similar communication patterns. The improvement brought by coflow scheduling/routing on cluster computing is likely to be reproducible in cloud SAN. Both networks are built on topologies with multiple alternative paths between a pair of hosts such as the fat-tree [1]. Both networks carry communication patterns involving a group of hosts concurrently communicating with another group [3]. The communication stage in cluster computing and data exchange/replication in cloud SAN all prefer multiple communication flows in the same logical operation finishing at close time instants. An example in cloud SAN is that a file transmission completes fast only if all parallel transferring flows complete fast because the slowest flow determines the overall file completion time.

For processing the traffic load with the OpenFlow SDN, the cloud SAN typically serves limited number of concurrent flows and experiences less frequent flow arrival/completion compared to DCN for cluster computing. The transfer applications in cloud SANs usually establish few parallel flows for data exchange. The flow table capacity of a commodity OpenFlow Switch is able to contain the number of concurrent flows [17]. When the cloud SAN stores big data, total files is relatively few and individual files are typically large so that parallel flows carrying files are long-lived. There are few flow arrivals and completions. The undesirable overheads of processing those events in the OpenFlow SDN are infrequently encountered.

The 2 observations suggest the feasibility of porting coflow scheduling and routing to OpenFlow-based cloud SANs, thus we design `MinCOF`.

## III. Framework Design

We design a coflow scheduling and routing framework, `MinCOF`, which is expected to achieve the following objectives in cloud SANs.

1) **Short Average CCT:** Our framework should minimize the average CCT of transfer applications to increase the throughput of the object storage. Shortening the CCT of the transfer application implies shortening the turn-around time of the storage I/O operation.
2) **Starvation Free Scheduling:** Our framework should prevent a coflow from being delayed for an uncertain amount of time.
3) **Work Conserving Scheduling:** Our framework should allocate any available network resource if the resource trigers a coflow to progress.
4) **Backward Compatibility:** Our framework should efficiently operate with flows without coflow affiliation.
5) **Proprietary System Avoidance:** Our framework should use mostly open source components to avoid licensing costs when deployed on production environments.
6) **Immediate Deployablity:** Our framework should impose least change on existing software and be built on commodity hardware.

The designs which ensure these objectives are highlighted in the following sub-sections.

We first give a high-level description of the procedure for the framework components to cooperate and dive into details in each subsequent sub-sections.

### A. Framework Workflow

Fig. 1 illustrates the targeted spine-leaf cloud SAN topology and all framework componets of `MinCOF`.

The framework initialization steps are

1) Each link from a leaf switch to a spine is sliced into queues for prioritizing coflows (Fig. 1(d)). (III-B)
2) Coflow-aware transfer application is deployed on each host (Fig. 1(c.c1) and (c.c2)). (III-C)

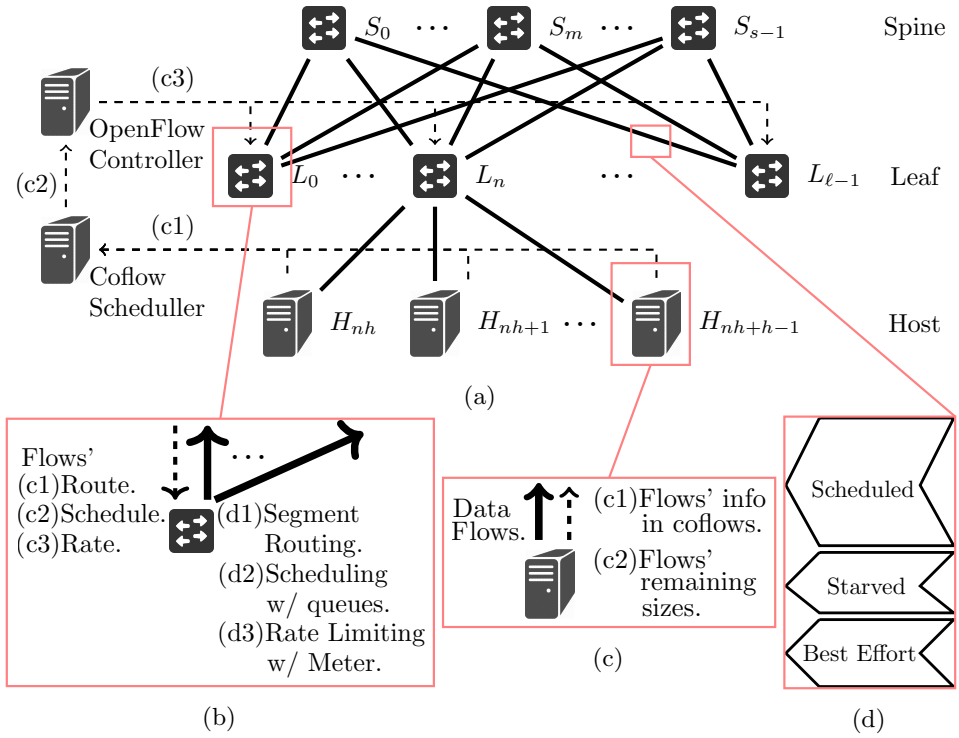



Fig. 1: (a) The overall architecture of MinCOF. Dashed links carry control messages. To save space, hosts connecting to leaf switches other than $L_n$ are omitted. (b) The functions on the leaf OpenFlow Switches. (c) The functions on the hosts. (d) The slices in links from leaf OpenFlow Switches to spines for prioritizing coflows.

The initialization steps for each coflow are

3) Unique coflow IDs are given to start transfer applications on hosts. Transfer applications report per flow information in coflows to the coflow scheduler (Fig. 1(c.c1) and (a.c1)). (III-C)
4) The scheduler schedules the new coflows. (III-D)

The repetative steps during the live time of each coflow are

5) Transfer applications update remaining data size of each flow to the scheduler (Fig. 1(c.c2) and (a.c1)). (III-C)
6) The scheduler calculates a new coflow schedule at coflow arrival and finishing. The scheduler releases per flow route, per flow queue schedule, and per flow rate to the OpenFlow Controller (Fig. 1 (a.c2)) to achieve short average CCT, starvation free, and work conserving objectives. (III-D)
7) The controller enforces the new per flow properties on leaf OpenFlow Switches (Fig. 1 (a.c3) and (b)).

### *B. OpenFlow 1.3 Switch*

MinCOF requires all leaf switches to be OpenFlow 1.3 Switches. Coflow schedules, coflow routing, and per-flow rate-limiting are enforced on OpenFlow Switches.

For enforcing coflow schedules, MinCOF takes advantage of slicing links from each leaf switch to spines for different classes of traffic. The slicing is achieved by dividing each physical egress port into 3 queues, Scheduled queue(50% port capacity), Starved queue(20%), and Best Effort queue(30%) shown in Fig. 1(d). When coflow scheduler pass a new coflow schedule, coflows with exclusive bandwidth allocations are placed in Scheduled queue with rate limiting configured, and the others are moved into the Starved queue. **To ensure starvation free scheduling**, Starved queue has identical priority to transfer data as the Scheduled queue. **To ensure work conserving scheduling**, any traffic exceeding the capacity of the Scheduled queue or Starved queue are moved into the Best Effort queue. Flows in the Starved queue and Best Effort queue freely compete for bandwidth. If fair share among flows is preferred, active queuing disciplines such as FaLL[18] should be deployed. The best effort queue length is set to 5% Bandwidth-delay Product(BDP) to keep queuing delay low. **To provide backward compatibility**, flows which do not have coflow affiliations are placed in the best effort queue.

For coflow routing, MinCOF adopts MPLS segment routing(SR). Each scheduled flow is attached with an MPLS label assigned by the scheduler when first reaching a leaf switch. Routing between leaf switches is based on MPLS labels. The benefit of using MPLS SR is that any conventional flow matching using the 5-tuple, <Protocol, Src. IP Addr., Dst. IP Addr., Src. Port, Dst. Port>, is reduced to a matching using only the MPLS label. Spine switches can be regular switches which are able to forward packets accroding to MPLS labels.

For per-flow rate-limiting, MinCOF utilizes the OpenFlow Meter. The coflow scheduler passes the corresponding rate limit of each flow in scheduled coflows in every new coflow

schedule. The rate limit is configured on the OpenFlow Meter table to control the transmission speed of flows.

### C. Coflow-aware Transfer Application

**To avoid using proprietary systems**, we create the coflow-aware transfer application by making a regular transfer application synchronize its own coflow information to our coflow scheduler. No proprietary flow monitoring tool is needed. We develop a compact 1-way egress synchronization protocol from the transfer application to the coflow scheduler. The benefit of our 1-way protocol is the simplicity. The 2-way coflow synchronization in existing frameworks requires a server which accepts incoming traffic on each host. Much more security and system management policy configurations have to be changed. **To provide immediate deployability**, our protocol only requires a regular UDP socket.

Our protocol can easily be integrated into the typical workflow of popular transfer applications as follows.
The typical workflow of a transfer application is

1) Receiving command line arguments.
2) Creating parallel TCP flows and evenly distributing data to TCP flows.
3) Periodically collecting statistics from all flows and reporting aggregated transmission progress.

The workflow after integrating our protocol is

1.1) Receiving command line arguments.
1.2) Receiving coflow ID and coflow scheduler location.
2.1) Creating parallel TCP flows and evenly distributing data to TCP flows.
2.2) Initially synchronizing the flows in the coflow information to coflow scheduler using 1 egress UDP segment.
3.1) Periodically collecting statistics from all flows and reporting aggregated transmission progress.
3.2) Synchronizing flows in the coflow information to the coflow scheduler using 1 egress UDP segment.

An example integrated transfer application is the customized `BBCP` of the BIC-LSU big data storage area network[2].

### D. Coflow Scheduler and OpenFlow Controller

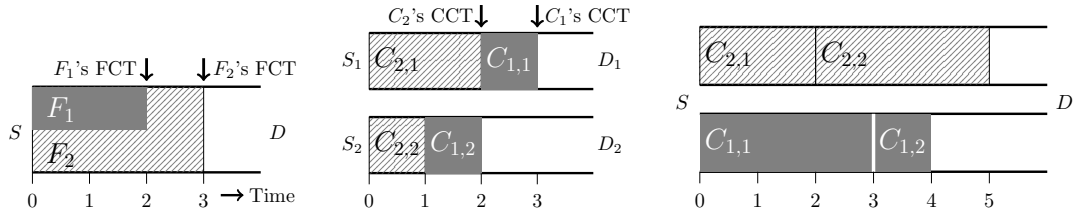
The core control components of `MinCOF` are a logically centralized coflow scheduler and an OpenFlow Controller. The coflow scheduler sophisticatedly allocates network resources to ensure that design objectives are achieved. OpenFlow Controller enforces the allocations by coflow scheduler on corresponding OpenFlow switches.

Our coflow scheduler is equipped with the 3 critical desirable features, coflow scheduling, coflow routing, and per-flow rate-limiting, of coflow scheduling/routing frameworks so that it can produce efficient or even optimal network resource allocations. We use 3 comprehensive examples to illustrate the importance of the critical features in shortening average CCT or flow completion time (FCT) by applying the shortest job first heuristic. All paths in examples have the bandwidth $\frac{1}{\text{unit time}}$ and all flows/coflows arrive at Time 0. **First**, we introduce the per-flow rate-limiting. In Fig. 2(a), 2 flows, $F_1$(Size 1) and $F_2$(Size 2) share 1 path. The optimal schedule is achieved in Fig 2(d). Per-flow rate-limiting can arbitrarily change the order of flow transmissions and enables the other 2 features. **Second**, we explain the coflow scheduling. In Fig. 2(b), 2 coflows, $C_1$ with Flow $C_{1,1}$(Size 1) and Flow $C_{1,2}$(Size 1) and $C_2$ with Flow $C_{2,1}$(Size 2) and Flow $C_{2,2}$(Size 1) share 2 independent paths. Coflow scheduling schedules all flows in one coflow as a logical unit. The optimal schedule is achieved in Fig 2(e). **Third**, we elaborate the coflow routing. In Fig. 2(c), 2 coflows, $C_1$ with Flow $C_{1,1}$(Size 3) and Flow $C_{1,2}$(Size 1) and $C_2$ with Flow $C_{2,1}$(Size 2) and Flow $C_{2,2}$(Size 3) share 2 alternative paths. Coflow routing finds the best route all flows in one coflow as a logical unit. The optimal routing decision is achieved in Fig 2(f).

Our scheduler maintains a Data Structure $Path$ **to avoid using proprietary systems** for collecting coflow information. $Path$ tracks the bandwidth usage in the Scheduled queue in each leaf-to-leaf path. The per-flow rate-limiting ensures that the usage information in $Path$ is precise. The total path tracked is the product of total leaves by total spines. In Fig. 1, the total path tracked is $s \cdot \ell$. As the blocking ratio ("number of links to hosts" to "number of links to spines" ratio on the leaf switch.) in the cloud SAN is typically high (10:1), the size of $Path$ is limited and can be efficiently managed using hash table. Function PathRem($i$,$j$,$p$,$h$) returns the remaining available bandwidth from Host $i$ to Host $j$ via Path $p$.

Our coflow scheduler is able to estimate the capacity gain of re-routing a TCP flow. We assume that every host uses the default Ethernet Maximum Transmission Unit (MTU) size 1500 bytes. Due to the stably short queue at each switch port (III-B), BDP can be estimated, so can the TCP's congestion window size (Cwnd) using the theoretical model in [16], Cwnd $\approx$ BDP/MTU. The duration of a re-routed flow is reasonably assumed to be the inter-coflow arrival time. Fig. 3 illustrates the worst impact of re-routing. The area below the curve is the effective capacity (BDP) of the flow. $C_O(C_N)$ is the effective capacity on the old(new) after re-routing and is calculated in Eq. 2(Eq. 4) respectively. The overall capacity gain of re-routing a TCP flow to a faster path is calculated by $C_N - C_O$.

Our coflow scheduler executes Alg. 1 to serve coflows at each coflow arrival/completion by invoking SERVECOFLOW. The new coflows first are routed through DCN for maintaining load balancing (Line 17-23). A new coflow schedule is generated for all coflows (Line 24). We define the bottleneck of a coflow to be the completion time of its slowest flow, which is calculated using Eq. 1. The coflow with the smallest bottleneck is re-routed if there are paths through which the coflow can earlier complete (Line 2-5) measured by the estimation mechanism explained in the previous paragraph. **To shorten average CCT**, coflows receive bandwidth allocation in the shortest bottleneck first (SBF) order (Line 6-12). Coflows with all flows which receive sufficient exclusive bandwidth allocation to continue are categorized as Scheduled Coflows ($C_{sch}$). The others are categorized to as Starved Coflows ($C_{str}$). The Scheduled Coflows in the latest coflow schedule are placed in the Scheduled queue on their paths (Line 25-



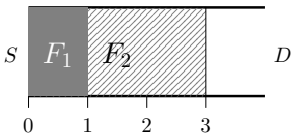

(a) 2 flows must fairly share one path without per-flow rate-limiting. Avg. FCT = (2+3)/2 = 2.5.



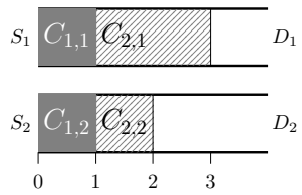

(b) Random schedule of 2 coflows. Avg. CCT = (2+3)/2 = 2.5.



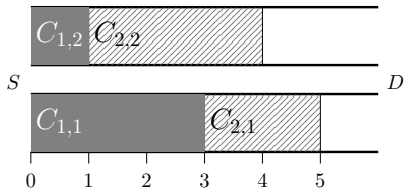

(c) Random routing of 2 coflows via 2 alternative paths. Avg. CCT = (4+5)/2 = 4.5.




(d) 2 flows optimally share one path with per-flow rate-limiting. Flows can transfer in arbitrary order. Avg. FCT = (1+3)/2 = 2.




(e) Optimal coflow schedule. Avg. CCT = (1+3)/2 = 2.




(f) Optimal coflow routing. Avg. CCT = (3+5)/2 = 4.

Fig. 2: (a)(d) Impact of per-flow rate-limiting. (b)(e) Impact of coflow scheduling. (c)(f) Impact of coflow routing. Each path has bandwidth $\frac{1}{\text{unit time}}$. The 3 features enable coflow scheduler to apply the shortest job first heuristic and its variations.

30) to receive a large portion of link bandwidth. **To ensure starvation free scheduling**, the Starved Coflows are moved to the Starved queue on their paths (Line 31-35) to share a small portion of link bandwidth (III-B).

The OpenFlow Controller configures OpenFlow Switches to enforce coflow scheduling, coflow routing, and per-flow rate-limiting using the OpenFlow protocol. **To provide immediate deployability**, `MinCOF` only includes hardware features which are provided by commodity OpenFlow Switches and scheduler functions which can be implemented on any up-to-date OpenFlow Controller frameworks.

$$\Gamma = \max_i(\max_j(\max_p \frac{d_{ij}}{PathRem(i,j,p,h)})) \quad (1)$$

where $d_{ij}$ is the summation of the remaining data size of the current coflow from Host $i$ to Host $j$.

$$C_O = D\frac{W_O}{2} + (D \textbf{ div } \frac{W_O}{2})\frac{W_O^2}{4} + \frac{(D \textbf{ mod } \frac{W_O}{2})^2}{2} \quad (2)$$

$$D_h = D - \frac{W_N}{2} \quad (3)$$

$$C_N = \frac{W_N^2}{4} + \frac{W_N}{2}D_h + (D_h \textbf{ div } \frac{W_N}{2})\frac{W_N^2}{4} + \frac{(D_h \textbf{ mod } \frac{W_N}{2})^2}{2} \quad (4)$$

where **div** calculates the quotient of integer division and **mod** calculates the remainder of integer division.

## IV. Evaluation

To evaluate the improvement of `MinCOF`, we compare the results of running `MinCOF` and previous works, Varys with ECMP routing and Tailor, on a DCN testbed with synthesized

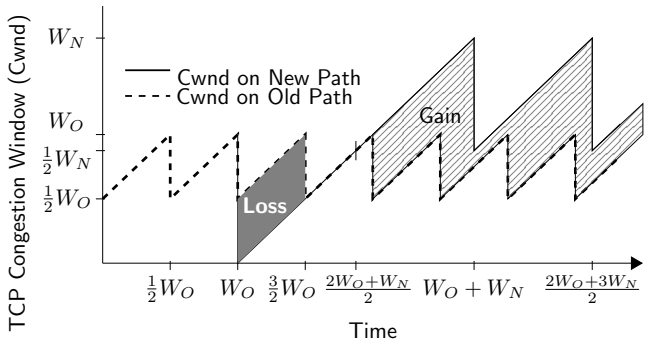



Fig. 3: The worst impact of switching paths on the congestion window size of a TCP flow. The flow re-routes to a faster path at Time $W_O$. Transmission capacity loss because of TCP re-ordering (shaded area) and gain (slashed area) are marked.

SAN workloads. Due to the complexity, we do not include the Rapier in our evaluation. Each result is the average of 5 runs and normalized to the result of Varys with ECMP routing.

### A. Testbed Environment

All scheduling frameworks run on a physical DCN testbed with the spine-leaf fat-tree topology. There are 4 spine switches, 4 leaf switches, and 16 hosts on the testbed shown in Fig. 4. Commodity servers are used to emulate both switches and hosts. Each switch and host has 8 Intel Xeon 2.33GHz CPU cores, 8GB main memory. Each switch has 2 Intel I350-T4 network interface cards. All servers run the Ubuntu Linux 16.04. For the simplicity of configuration, we deploy Open vSwitch (OVS) OpenFlow Switch [10] on every switch

**Algorithm 1** Coflow Scheduling & Routing in OpenFlow-based spine-leaf DCN.

**function** SCHEDULE(Coflows $\gamma$, PathRem(.))
  $C_{sch}$ = Sort all Coflows in $\gamma$ in SBF order
  **if** Rebalancing shortens 1st coflow's CCT $\in C_{sch}$ **then**
  ▷ Measured using Eq. 1, Eq. 2, and Eq. 4.
    Rebalance flows $\in$ 1st coflow.
  **end if**
  **for all** coflow C in $C_{sch}$ **do** ▷ allocate BW.
    Calculate $\Gamma$ using Eq. 1
    **for all** flow in C **do**
      $rate \leftarrow$ (flow's remaining size) / $\Gamma$
      Update PathRem(src host,dst,p,h) with $rate$
    **end for**
  **end for**
  $C_{str}$ = starved coflows in $C_{sch}$
  **return** $C_{sch}$, $C_{str}$
**end function**

**procedure** SERVECOFLOW(Coflows $\gamma$, PathRem(.))
  **if** new coflow arrival **then**
    **for all** flow $\in$ new coflow **do**
      **if** flow traverses across leaf SWs **then**
        Place flow on least used inter-leaf path.
      **end if**
    **end for**
  **end if**
  $C_{sch}$, $C_{str}$ = SCHEDULE($\gamma$, PathRem(.))
  **for all** flow in $C_{sch}$ **do**
    ▷ place scheduled flows on src leaf SWs.
    Configure new segment routing path
    Migrate flow to Scheduled queue at egress port
    Configure flow's OpenFlow Meter entry
  **end for**
  **for all** flow in $C_{str}$ **do**
    ▷ place starved flows on src leaf SWs.
    Migrate flow to Starved queue at egress port
    Remove flow's OpenFlow Meter entry
  **end for**
**end procedure**

even though MinCOF does not require the spine switches to be OpenFlow Switches. The coflow scheduler is written in Python, and the OpenFlow Controller is built on the Ryu framework [6]. However, the OpenFlow Meter is not implemented on OVS. We create a primitive control framework for the OpenFlow Controller to manipulate the *tc* command on each switch to mimic the behavior of the OpenFlow Meter using the Linux kernel packet scheduler. All network interfaces are throttled to 100Mb/s to ensure that the processing power of the CPUs on the switch is competent for forwarding network traffic. The RTT at a host is around 350$\mu$s.

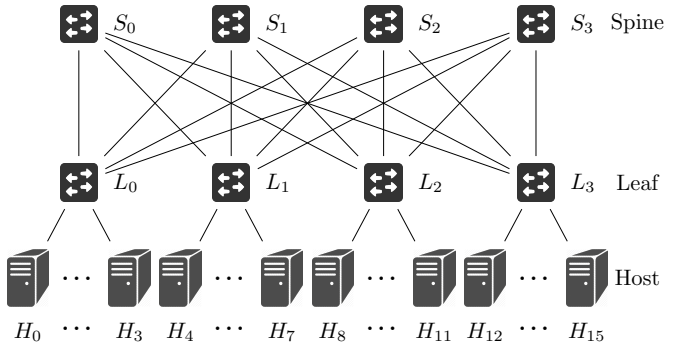



Fig. 4: The spine-leaf fat-tree experiment DCN consists of 4 spines, 4 leaves, and 16 hosts. Each link is 100Mb/s.

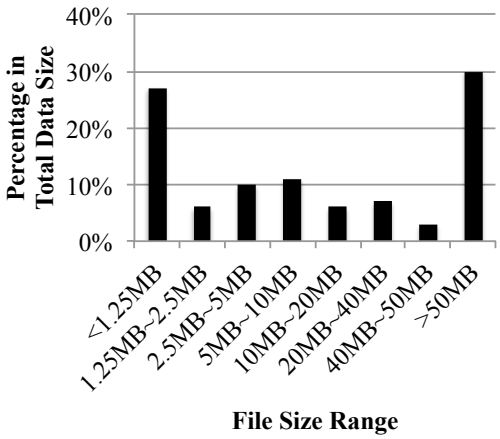



Fig. 5: The downscaled file size distribution of Data Oasis.

### B. Workload

To the best of our knowledge, there are very few publicly available statistics of data size distribution in cloud storage. Since the design of MinCOF targets the big data cloud storage, we adopt the scientific big data size distribution in the Data Oasis storage cluster [14]. We scale down the file size by multiplying the testbed bandwidth to Data Oasis' SAN bandwidth ratio, $\frac{\text{Exp. Net BW}}{\text{Data Oasis Net BW}} = \frac{100Mb/s}{10Gb/s} = 0.01$. The scaled file size distribution is summarized in Fig. 5. To increase scalability, MinCOF schedules only at coflow arrival/finishing and only considers the coflows carrying the large files (>2.5MB) which already account for around 70% of total data size. We generate 64 coflows with the arrival times distribution from the Facebook trace in [5]. Each coflow involves 3 through 12 normally distributed host-to-host file transmissions. Each file transmission uses 4 through 16 flows.

TABLE II: Statistics of coflow inter-arrival time in the Facebook traffic trace.

| Mean | Median | Mode | Std. Dev. |
|---|---|---|---|
| 6.91s | 3.06s | 1.58s | 9.76s |

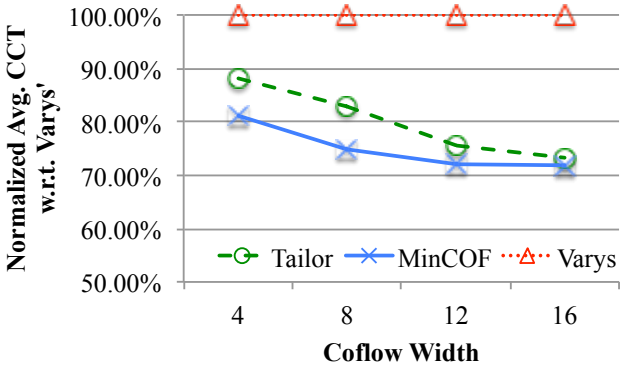



Fig. 6: Impact of Coflow Width. Shorter CCT is better.

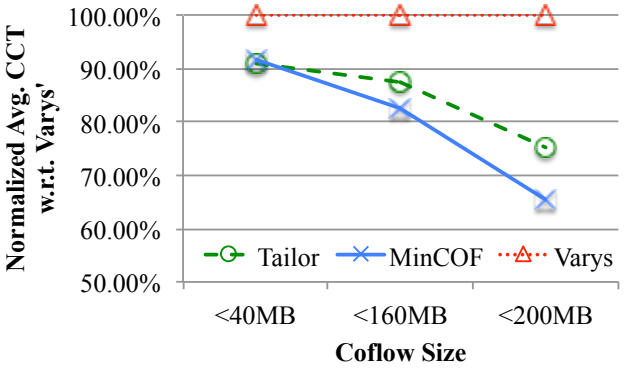



Fig. 7: Impact of Coflow Size. Shorter CCT is better.

### C. Impact of Coflow Width

In this scenario, we test coflow scheduling/routing frameworks under various coflow width, number of flows in coflows. We intentionally generate coflows all with only 4, 8, 12, or 16 flows. The result is provided in Fig. 6. `MinCOF` and Tailor both more effectively shorten the average CCT compared to Varys. `MinCOF` further improves by at most 9.81% compared to Tailor in the case of 8 flows. The trend for saturation with larger number of flows is because of the overhead of our workaround implementation of the OpenFlow Meter. Each flow becomes shorter and faster finishes when the coflow width increases. The coflow scheduler is not able to instantaneously adjust the OpenFlow Meter entries to the most adequate values. We conjecture that the saturation can be alleviated if `MinCOF` runs on dedicated hardware OpenFlow Switches because the OpenFlow protocol message format and message passing mechanism is standardized and highly optimized.

### D. Impact of Coflow Size

We vary coflow size, the aggregated size of all associated flows, in the experiment. We divide the file size range into 3 categories, <40MB(4x10MB), <160MB(4x40MB), and <200MB(4x50MB). The last range ends at 50MB because we simulate the scaled down file size upper limit in the OpenStack Swift cloud object storage, 5GB. Fig. 7 presents the result. `MinCOF` demonstrates more advantage with larger coflow size, up to 12.94% compared to Tailor, if the coflow can be as large as 200MB. The overhead problem mentioned in Sec. IV-C, again, results in the indistinction between `MinCOF` and Tailor in the case of small coflow.

### E. Impact of Network Blocking Ratio

In this experiment, we observe the performance of coflow scheduling/routing frameworks in the spine-leaf fat-tree DCN with various blocking ratios. A higher blocking ratio reflects that communications between hosts connecting to different leaf switches are more difficult. The result is summarized in Fig. 8. With high blocking ratio, the severe congestion in the network lessens the improvement of routing so that 3 frameworks have similar performance. With low blocking ratio, `MinCOF` effectively routes flows among multiple available paths so that achieves the largest improvement of 11.33% compared to Tailor.

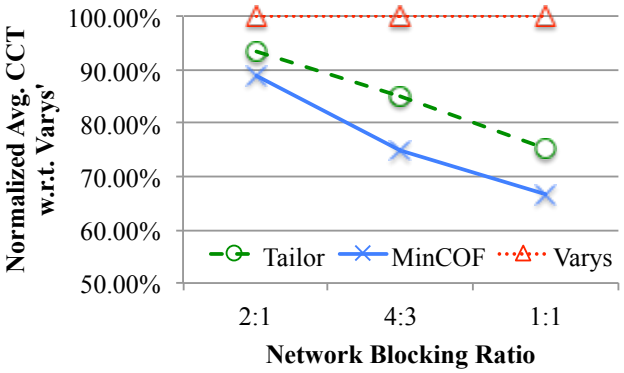



Fig. 8: Impact of Network Blocking Ratio. Shorter CCT is better.

### F. Impact of Background Traffic

We randomly select coflows to not synchronize with the scheduler as background traffic to examine the resilience and backward compatibility of the coflow scheduling/routing frameworks. Background traffic in practice is from applications which are not yet integrated with the coflow scheduling/routing framework. The percentage of background traffic gradually increases from 10% through 50%. The result is shown in Fig. 9. Tailor and `MinCOF` outperform Varys because of their routing feature. However, Tailor saturates with 40% background traffic. `MinCOF` profits from the sliced leaf-to-spine links (III-B) which confine the bandwidth consumed by background traffic and still improves under heavy background traffic. The maximum improvement is 15.21% with 50% background traffic.

## V. Related Work

Varys is the coflow scheduling framework which totally operates on hosts and is not aware of the network condition. Varys requires a customized OS on each host for a 2-way

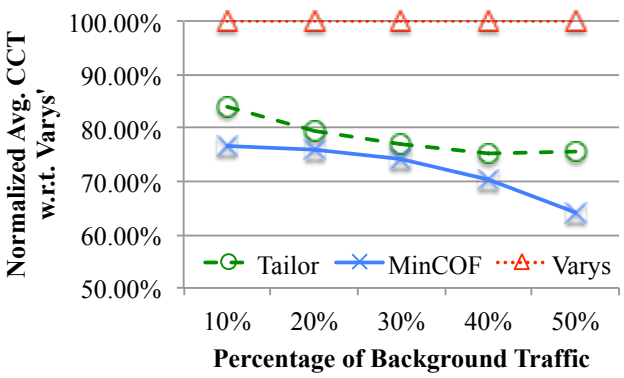



Fig. 9: Performance under Background Traffic. Shorter CCT is better.

interact to a logically centralized coflow scheduler to monitor network interface usage, synchronize coflows' information, and enforce coflow scheduling decisions. The critical coflow scheduling features of Varys include coflow scheduling and per-flow rate-limiting.

Rapier inherits all features from Varys and incoporates coflow routing while generating coflow schedules. Rapier also needs a customized OS to enfource per-flow rate-limiting.

Tailor is a simple coflow scheduling and routing framework built upon OpenFlow-based DCN. Limited coflow scheduling and coflow routing are included. Tailor adopts proprietary sFlow-RT to collect coflows' information and network condition for trimming the completion time of the slowest flow. Per-flow rate-limiting is not considered. Tailor does not require any features provided by a customized OS. Applications can profit from Tailor coflow scheduling and routing with simple modification.

`MinCOF` synthesizes all desirable critical features of all previous works including coflow scheduling, simplified coflow routing, and per-flow rate-limiting. `MinCOF` also gets rid of undesirable requirements such as customized OS and proprietary system component. An OpenFlow-based DCN and extended transfer applications for the 1-way coflow information synchronization are the only requirements.

## VI. Conclusion and Future Works

Existing coflow scheduling/routing frameworks require customized OSs on hosts or proprietary network monitoring tools to fully function. Those requirements hinder the adoption of coflow scheduling which does effectively reduce the communication and completion time of applications in DCNs. We design `MinCOF`, a coflow scheduling and routing framework which imposes only minimal modifications to applications and accommodates all critical features of coflow scheduling/routing targeting the OpenFlow-based cloud SANs. Being built upon open source software and standardized commodity OpenFlow Switches, `MinCOF` is easily deployable in production cloud SANs. Experiment results using software OpenFlow Switches confirm that `MinCOF` shortens the CCT by up to 12.94% compared to latest OpenFlow-based framework. `MinCOF` in the cloud SANs with dedicated hardware OpenFlow Switches is expected to perform even better. As this prototype implementation illustrates consistent improvements, we plan to port `MinCOF` to the BIC-LSU 10/40 Gb/s OpenFlow-based spine-leaf big data SAN.

## Acknowledgment

This work was partially funded by NIH LBRN grant (P20GM103424), NSF CC-NIE award (#1341008), NSF MRI grant (MRI-1338051), NSF IBSS-L (#1620451), NSF CISE grant CNS-1566443, Louisiana Board of Regents under grant LEQSF(2016-19)-RD-A-08, and LEQSF(2015-18)-RD-A-11.

## References

[1] M. Al-Fares, A. Loukissas, and A. Vahdat. A scalable, commodity data center network architecture. In *ACM SIGCOMM Computer Communication Review*, volume 38, pages 63–74. ACM, 2008.

[2] C.-h. Chiu, N. Lewis, D. K. Singh, A. K. Das, M. M. Jalazai, R. Platania, S. Goswami, K. Lee, and S.-J. Park. Bic-lsu: Big data research integration with cyberinfrastructure for lsu. In *Proceedings of the XSEDE16 Conference on Diversity, Big Data, and Science at Scale*, XSEDE16, pages 28:1–28:8, New York, NY, USA, 2016. ACM.

[3] M. Chowdhury and I. Stoica. Coflow: A networking abstraction for cluster applications. In *Proceedings of the 11th ACM Workshop on Hot Topics in Networks*, pages 31–36. ACM, 2012.

[4] M. Chowdhury, M. Zaharia, J. Ma, M. I. Jordan, and I. Stoica. Managing data transfers in computer clusters with orchestra. In *ACM SIGCOMM Computer Communication Review*, volume 41, pages 98–109. ACM, 2011.

[5] M. Chowdhury, Y. Zhong, and I. Stoica. Efficient coflow scheduling with varys. In *ACM SIGCOMM Computer Communication Review*, volume 44, pages 443–454. ACM, 2014.

[6] R. S. F. Community. *Ryu SDN Framework*, 2017.

[7] R. C. Computing. *OpenStack*, 2017.

[8] I. Corp. *sFlow-RT*, 2017.

[9] J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[10] L. Foundation. *Open vSwitch*, 2017.

[11] O. Foundation. *OpenStack Swift*, 2017.

[12] R. L. Grossman, M. Greenway, A. P. Heath, R. Powell, R. D. Suarez, W. Wells, K. P. White, M. P. Atkinson, I. A. Klampanos, H. L. Alvarez, C. Harvey, and J. Mambretti. The design of a community science cloud: The open science data cloud perspective. In *SC Companion*, pages 1051–1057, 2012.

[13] J. Jiang, S. Ma, B. Li, and B. Li. Tailor: Trimming coflow completion times in datacenter networks. In *Computer Communication and Networks (ICCCN), 2016 25th International Conference on*, pages 1–9. IEEE, 2016.

[14] G. K. Lockwood, M. Tatineni, and R. Wagner. Storage utilization in the long tail of science. In *Proceedings of the 2015 XSEDE Conference: Scientific Advancements Enabled by Enhanced Cyberinfrastructure*, XSEDE '15, pages 32:1–32:8, New York, NY, USA, 2015. ACM.

[15] G. Malewicz, M. H. Austern, A. J. Bik, J. C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski. Pregel: a system for large-scale graph processing. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, pages 135–146. ACM, 2010.

[16] M. Mathis, J. Semke, J. Mahdavi, and T. Ott. The macroscopic behavior of the tcp congestion avoidance algorithm. *SIGCOMM Comput. Commun. Rev.*, 27(3):67–82, July 1997.

[17] S. McGillicuddy. *Pica8 doubles flow rule capacity in its new OpenFlow 1.3 switch*, 2014.

[18] L. Xue, C.-H. Chiu, S. Kumar, P. Kondikoppa, and S.-J. Park. Fall: A fair and low latency queuing scheme for data center networks. In *Computing, Networking and Communications (ICNC), 2015 International Conference on*, pages 771–777. IEEE, 2015.

[19] Y. Zhao, K. Chen, W. Bai, M. Yu, C. Tian, Y. Geng, Y. Zhang, D. Li, and S. Wang. Rapier: Integrating routing and scheduling for coflow-aware data center networks. In *Computer Communications (INFOCOM), 2015 IEEE Conference on*, pages 424–432. IEEE, 2015.