SmartRep: Reducing Flow Completion Times with Minimal Replication in Data Centers

SmartRep: Reducing Flow Completion Times with Minimal Replication in Data Centers

Fuguang Wang[†], Zhuzhong Qian[†], Sheng Zhang[†], Mianxiong Dong[‡], and Sanglu Lu[†]
[†]State Key Laboratory for Novel Software Technology, Nanjing University, China
[‡]Muroran Institute of Technology, Japan
Email: [†]{wfg2008913, zhangsheng}@dislab.nju.edu.cn, {qzz, sanglu}@nju.edu.cn, [‡]mx.dong@ieee.org

***Abstract*—To improve users' experience, TCP short flows that are heavily used in interactive services should be completed as soon as possible. In current data centers, large flows and head-of-line blocking in switches hinder short flows from completion, which leads to long-tailed flow completion times (FCT). Replicating short flows with multiple equal-cost paths is a promising way to reduce FCT. However, the original flow and its replicated one are quite likely to be routed to the same path (ECMP hash collision), which increases both the mean and 99-percentile FCT significantly. What's more, inadequate replication leaves many other less-congested equal-cost paths unused and limits the performance while excess replication degrades throughput of large flows. To solve these problems, we propose SmartRep, a scheme consisting of an efficient and effective *traceroute* based hash collision avoidance method and an algorithm to decide the optimal number of replicated flows for different short flows. SmartRep can be easily implemented in software and readily deployed in data centers. Extensive NS2 simulations show that our approach improves previous replication-based work by 25%-50% in both mean and 99th percentile FCT, and meanwhile imposes negligible impact on large flows.**

I. Introduction

Nowadays, numerous online services are hosted in data centers. Many services, such as social networking, online shopping and web search have strict requirements on latency. Even a small increase in response latency to the user's request may lead to a large financial loss [**?**].

Modern applications in data centers are usually constructed in Partition/Aggregate pattern [**?**] [**?**], in which aggregator servers partition a large task into small ones and distribute them to worker servers. Worker servers achieve these small tasks and return results to aggregators. The finish time of each large task is largely up to the flow completion times (FCT) of the delay-sensitive short flows generated in this process. However, short flows are very likely to be queued behind bursts of packets from large flows, and they may experience more than 2x larger mean FCT than its theoretical minimum [**?**] [**?**] and even more than 10x larger than the average [**?**] [**?**].

To reduce FCT of short flows in data centers, a lot of approaches were proposed in recent years. However, most of them are need to modify either switches or TCP protocol [**?**] [**?**] [**?**] [**?**], which makes them difficult to deploy in current large-scale production data centers. To avoid modifying TCP or physical layer mechanism, replicating short flows with multiple equal-cost paths, which is easily implemented as libraries or middleware at the application layer, is a promising way to reduce FCT. Typically, each short TCP flow is replicated by creating another TCP connection to the receiver, and sending identical packets for both flows [**?**]. The application uses the result from the flow which completes first. Thus a task undertook by a flow originally is performed by two flows now, which makes the task less likely to be blocked by large flows than before when the two flows traverse different paths. However, it is observed that *many short flows and their replicated ones are routed to the same path because of hash collision in ECMP* (Equal-Cost Multi-Path routing), and this leads to a big performance degradation. Further, *even though the replicated flow and its original one traverse different paths, they are very likely to be blocked by large flows at the same time when the network load is high.*

In this paper, we introduce SmartRep, a new scheme to overcome the limitations mentioned above and maximize the improvement of replication to FCT of short flows. SmartRep is composed of a hash collision avoidance (HCA) method and an efficient algorithm (RepNumAssign) to find the optimal number of replicated flows created for every short flow while minimizing the overhead. In HCA, we use *traceroute* based path probe to find the route of a flow with a specific TCP/IP 5-tuple, and allocate a carefully selected TCP/IP 5-tuple to a replicated flow of the original one, which guarantees the two flows traverse different paths. In designing RepNumAssign, we model the process that the original flow and its replicated ones traverse the network as Bernoulli trials and then try to understand the problem that how to assign the optimal number of replicated flows to each short flow. SmartRep does not modify switch or TCP stack and can be easily implemented and readily deployed in data centers. Extensive NS2 simulations with a variety of workloads show SmartRep outperforms previous replication-based solution by 25%-50% in both mean FCT and 99-percentile FCT.

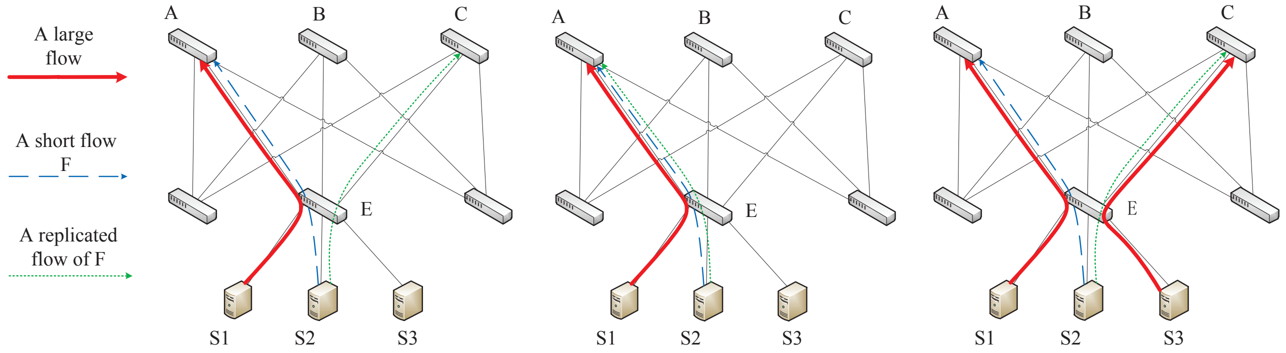
In summary, our work makes three main contributions: (i) We show by theoretical analysis that hash collision between original flow and its replicated one has a great impact on FCT of short flows and the potential of cutting down FCT of short flows by creating multiple replicated flows; (ii) We design SmartRep, a new scheme to use replication to reduce FCT of short flows, which consists of HCA and RepNumAssign; and (iii) We implement SmartRep in NS2 and conduct extensive simulations to evaluate it.

The rest of the paper is organized as follows. We describe the motivation of SmartRep in Section II. Then we present the design of SmartRep in Section III. In Section IV, SmartRep is evaluated. Finally, we conclude the paper in Section V.

II. Motivation

A. Related Work

Many of previous work reduce the FCT of short flows by reducing the occupations of queue in switches [**?**] [**?**], explicit rate control [**?**], or favouring short flows by well-designed flow



(a) An ideal case

(b) A hash collision

(c) An extreme case

Fig. 1: Hash collision illustration

scheduling algorithms [?] [?]. These methods require switch or TCP modification, which makes them difficult to deploy in current large-scale production data centers. Recently, some replication based approaches is proposed to reduce the completion times of tasks [?] [?] [?], which can be implemented at application layer and avoid changing switch or TCP protocol stack. The idea of these approaches is to initiate redundant operations across diverse resources or issue redundant requests to multiple servers. Based on this idea, RepFlow [?] creates two flows to finish a task, which was accomplished by a flow originally. One of the flows is known as the original flow and the other as the replicated flow. The replicated flow is exactly the same as its original one except that its destination port is set equal to its original one's plus one. The application uses the result from the first finished flow. When the original flow and its replicated one traverse different paths, which depends on ECMP heavily, the task is less likely to be blocked by large flows than before.

B. Impact of Flow Hash Collision

Our work is closely related to ECMP and we introduce it briefly next. ECMP is widely deployed in current data centers to utilize multiple equal-cost paths [?] [?]. To avoid out-of-order, which causes obvious throughput degradation of TCP [?], ECMP ensures the packets belonging to the same TCP flow traverse the same path by hashing TCP/IP 5-tuple (source IP address, destination IP address, source port, destination port and protocol number) fields of each packet to one of the available equal-cost routes to the destination [?]. The widely used hash algorithm is Hash-Threshold [?]. It first selects a key by performing hash over TCP/IP 5-tuple of a packet. Then the size of the hash function output space is divided by the number of available next hops to get the region size. Lastly the hash key is divided by region size to get the output port number. Note that the algorithm doesn't specify the hash function to obtain the key but typically uses Cyclic Redundancy Check (CRC) [?].

In ideal situations, the replicated flow and its original one are hashed to different equal-cost paths by ECMP and avoid being blocked by large flows, as Fig. 1(a) shows. Here by "different equal-cost paths" we means paths that don't share links. If a short flow and its replicated one both traverse a switch-to-switch link, we say that a flow hash collision happens (Fig. 1(b)). When a flow hash collision happens, replication becomes useless and even counterproductive because the output queue caching the original flow is more congested than before due to the replicated flow. Unfortunately, the flow hash collision is quite likely to happen in current replication based design. Through our simulations, over 25% of original flows will collide with its replicated ones when using CRC16 as the hash function, and the percentage exceeds 50% when using CRC32 hash function. The impact of flow hash collision on FCT of short flows is presented in Theorem 1. Note that throughout this paper we consider FCT as the flow's completion time normalized by the possible completion time without contention. Both mean and tail (99-th percentile) FCT is studied.

Theorem 1. *Let p be the probability that a short flow and its replicated one are routed to the same path, and ρ be the load of the data center network. Then the probability that both of the flows meet a large flow is $P = (1+\epsilon)\rho[p+(1+\epsilon)\rho(1-p)]$, where ϵ is the fraction of total bytes from replication.*

Proof: The proof is given in Appendix. ■

We learn from [?] that the mean FCT of short flows is

$$FCT_{mean} = \frac{pM}{2(1-p)} \int_0^{S_L} \frac{log_2(x/k+1)}{x} \frac{f(x)}{F(S_L)} dx + 1 \tag{1}$$

the 99-th percentile FCT of short flows is $FCT_{99} =$

$$\frac{pM}{2(1-p)} \int_0^{S_L} \frac{log_2(x/k+1) - \ln\frac{100}{e}}{x} \frac{f(x)}{F(S_L)} dx + 1, \tag{2}$$

the mean FCT of large flows is

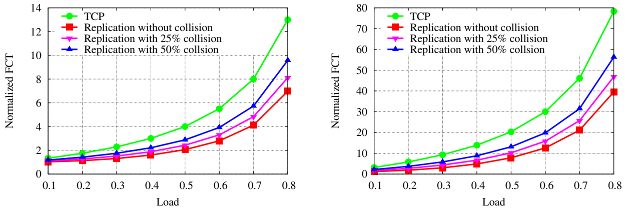
$$FCT^L_{mean} = \frac{p}{2(1-p)} + 1, \tag{3}$$

the 99-th percentile FCT of large flows is

$$FCT^L_{99} = (1 + \int_{S_L}^{\infty} \frac{(2\ln 10 - 1)f(x)}{(1-F(S_L))x} dx) \frac{p}{2(1-p)} + 1, \tag{4}$$

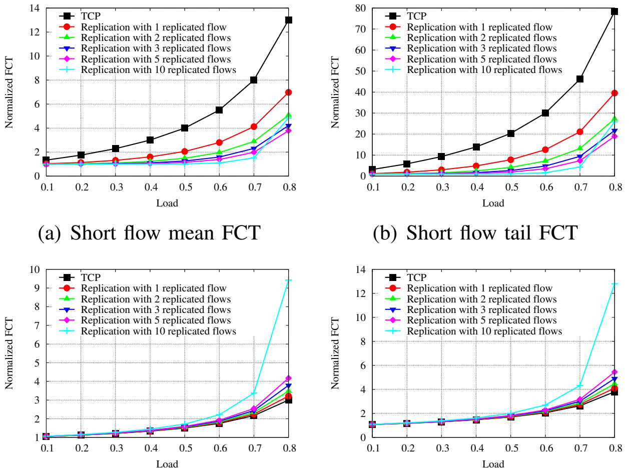
where p is the probability a short flow meets a large flow, and other notations are listed in Table I. According to the Theorem 1, Equation 1 and Equation 2, we plot the mean and 99-th percentile FCT of short flows under different levels of flow hash collision with the web search workload [?] in Fig. 2. As Equation 3 and 4 show, the mean and tail FCT of large flows is decided by the load increase from replication, which keeps constant under different levels of hash collision, so they are not affected by flow hash collision.

Fig. 2 shows when flow hash collision happens in probability 0.25, both the mean and tail FCT of short flows with replication will increase by about 15% at high load, compared to replication without flow hash collision, and when



(a) Short flow mean FCT (b) Short flow tail FCT

Fig. 2: FCT contrast under different levels of hash collision



(a) Short flow mean FCT (b) Short flow tail FCT

(c) Large flow mean FCT (d) Large flow tail FCT

Fig. 3: FCT contrast with different numbers of replicated flows

the probability increases to 0.5, the FCT increases by over 30%. The results demonstrate the great negative impact of flow hash collision on FCT of short flows.

C. Gain of Creating Multiple Replicated Flows

In a worse case, both of the short flows will collide with large flows when the network load is high, even if they traverse different paths (Fig. 1(b)). The reason is that creating one replicated flow is insufficient to utilize the multiple equal-cost paths in data center networks.

In the following analysis, we assume that there are enough equal-cost routes for each short flow and that replicated flows and its original one traverse different paths. Let ϵ be the fraction of total bytes from replication and ρ be the network load without replication. When we create $n-1$ replicated flows for every short flow, the network load increases from ρ to $(1+(n-1)\epsilon)\rho$ and the probability that all of them collide with a large flow is $(1+(n-1)\epsilon)^n\rho^n$, where ϵ is below 0.035 in the empirical workloads [?] [?] [?] used by us.

Using Equation 1, 2, 3 and 4, we plot the mean and 99-th percentile FCT of short flows and large flows with different number of replicated flows using the web search workload [?] in Fig. 3. As Fig. 3(a) and 3(b) show, creating 5 replicated flows can reduce both the mean and tail FCT by roughly 50%, compared to creating one. Meanwhile, excess replication cause large flows to finish late, as Fig. 3(c) and 3(d) show. So it is important to find the suitable number of replicated flows.

D. Goals and Challenges

The problems discussed above motivate us to design a new replication based method to reduce the FCT of short flows. However, it is not easy to find a suitable way to avoid the hash collision between flows or leverage multiple equal-cost paths by creating multiple replicated flows. Specifically, we are faced with the following challenges: (i) Though simple hash collision avoidance methods like modifying hash algorithm in switches exist, we do not use them because they are hard to deploy in production data centers; (ii) Short flows are very delay-sensitive and impose strict requirements on the efficiency; (iii) Simply creating multiple replicated flows for every short flow is not feasible because it has a large impact on performance of large flows and also suffers hash collisions.

Notations	Notes
M	maximum window size (64KB,44 packets)
ϵ	the fraction of total bytes from short flows
W	queuing delay of the M/G/1-FCFS queue
$\rho \in [0,1)$	overall traffic load
S_L	threshold for large flows(100KB)
$F(\cdot), f(\cdot)$	flow size CDF and PDF
k	initial window size in slow-start (12 packets)

TABLE I: Notations

III. Design Details

In this section, we elaborate on SmartRep targeting to maximize the capabilities of replication. SmartRep (Algorithm 1) consists of two components: HCA which avoids the hash collision by selecting suitable source ports for replicated flows and the original flow, and RepNumAssign which finds the optimal number of replicated flows for each short flow. The number of replicated flows for each short flow is computed according to its flow size in advance by RepNumAssign and stored in a set $repNum$. For each short flow, if its ends are located under the same top of rack, no replicated flow will be created for it because only one path exists between the two hosts. Otherwise, the number of replicated flows created is decided according to its flow size k (line 5). This observation saves much overhead because over 75% of the whole traffic happened in the same rack in most data centers [?]. Lastly, HCA is invoked to assign source ports for n replicated flows and the original one, them traverse different paths.

Next, we describe in details HCA and RepNumAssign.

Algorithm 1 SmartRep

```
1: procedure SmartRep(srcIP,dstIP,srcPort,dstPort,k,repNum)
2:     $n \leftarrow 0$
3:     if $srcIP$ and $dstIP$ not under the same ToR then
4:         $n \leftarrow repNum_k$
5:     end if
6:     $P \leftarrow HCA(srcIP, dstIP, P \cup p_0, n+1)$
7:     return $P$
8: end procedure
```

A. Hash Collision Avoidance

Our principle is to avoid switch and TCP modification and keep universal. Without hardware modification, we can not control the path of a flow. There are diverse implementations of ECMP so we can't assume a specific hash algorithm is used. We have to vary the changeable fields in TCP headers of a flow, test its route and check if the setting of the fields meets our demand. In the fields of a TCP/IP 5-tuple, we can change only the TCP source port because IP addresses, protocol number and the destination port are all fixed once the client and the server are decided. We try to find suitable source ports for replicated flows and the original one. To this end, we design a mechanism using path probing technique (Algorithm 2).

1) Traceroute based port selection: We choose *traceroute* like method to probe the path that a flow with a predefined TCP/IP 5-tuple will traverse (procedure $traceRoute$ in Algorithm 2). Procedure traceRoute constructs a packet using the input and set TTL of the packet as 1. When the packet arrives at its first hop switch, its TTL is decremented to zero and the switch will notify the sender with a packet carrying its address according to Internet Control Message Protocol. So the sender knows which path the flow with the 5-tuple will traverse. We consider data centers with fat-tree topology in this paper and we assume the ECMP algorithm in each switch is the same so that once the first hop of a flow is decided the route of the flow is decided. In data centers with other types of topology, the setting of TTL may be different.

When a host will create a short flow, SmartRep passes procedure HCA the IP addresses and the destination port of the flow, and the number of flows to be created, k, including replicated flows and the original one. HCA assign source ports to the k flows one by one. For those flows whose source ports have been assigned, the identity of the paths they will traverse is stored in $Addr$. For each next flow to be assigned a source port, a iteration will be done to find a source port, making the flow's path different with previous flows' (line 5-8). Finally, the k source ports are returned. It's clear that any two of k flows will traverse different paths.

Algorithm 2 Hash Collision Avoidance Algorithm

```
 1: procedure HCA(srcIP,dstIP,dstPort,k)
 2:     $p \leftarrow the\ biggest\ used\ port$
 3:     $Addr \leftarrow \phi, srcPorts \leftarrow \phi$
 4:     while $k > 0$ do
 5:         repeat
 6:             $p \leftarrow p + 1$
 7:             $addr \leftarrow traceRoute(srcIP, dstIP, p, dstPort)$
 8:         until $addr \notin Addr$
 9:         $Addr \leftarrow Addr \cup \{addr\}$
10:         $srcPorts \leftarrow srcPorts \cup \{p\}$
11:         $k \leftarrow k - 1$
12:     end while
13:     return $srcPorts$
14: end procedure
15:
16: procedure TRACEROUTE(srcIP,dstIP,srcPort,dstPort)
17:     create a TCP packet $pkt$ with IPs and Ports from input
18:     set the TTL of $pkt$ to 1
19:     send $pkt$ to get ICMP error packet $reply$
20:     return source IP address of $reply$
21: end procedure
```

2) Efficiency of HCA: The running time of HCA is up to procedure traceRoute. When n replicated flows are created for a short flow, procedure traceRoute will be invoked at least $\sum_{i=1}^{n} i = n(n+1)/2$ times. It takes $2*(link\ delay) + 2*(server\ processing\ delay) + switch\ processing\ delay \approx 1 + 18 + 11 = 30\mu s$ to finish one traceRoute. However, for a large data center, n may exceed 10 and HCA will run for over $1.6ms$. It is unacceptable to finish such a time-consuming operation every time a short flow lasting tens of μs is created.

Fortunately, HCA can be done in advance and the results (TCP/IP 5-tuples and its corresponding paths) can be stored in a database. This is because applications in data centers usually work in a Partition/Aggragation way, where the destination servers' addresses of short flows keep constant and ECMP ensures that the flows with the same TCP/IP 5-tuple traverse the same path at any time. Therefore, for each TCP/IP 5-tuple, procedure traceRoute runs only once and subsequent query about the tuple can be replaced by a fast database query. Thus, procedure traceRoute needs to run for $N = n+\sum_{k=1}^{n}\theta_k$ times, where θ_k is the number of iteration in line 5-8 of procedure HCA when the k^{th} flow is created.

If we assume the trials in the loop are independent from each other, θ_k follows a geometric distribution. The expectation of θ_k is $E[\theta_k] = 1/(1-p)^k$, where p is the probability that any two flows have a hash collision. Therefore, the expected number of running traceRoute when creating n replicated flows for a short flow is $E[N] = n+\sum_{k=1}^{n}E[\theta_k] = n+\sum_{k=1}^{n}1/(1-p)^k = n + \frac{1-(1-p)^n}{p(1-p)^n}$. For a fat-tree data center with 16000 hosts, there are 20 equal-cost paths between any pair of hosts at different pods and $n \leq 20$. Using CRC16 as the ECMP hash function, $p \approx 0.25$ and the expected time of running HCA once is no longer than $38ms$. It is observed that the number of active flows at any given interval is less than 10,000 [**?**], so it will take about $10000 \times 38ms = 380s$ to finish all the running of HCA on average.

B. Assignment of Replicated Flows

Multiple equal-cost paths exist between a host pair in typical data centers while one replicated flow is created, which is not enough at high network load. We try to create multiple replicated flows for a mice flow. However, such design may incur high overhead, reducing throughput of large flows and aggravating incast [**?**]. To reduce the overhead, we do not create the same number of replicated flows for each flow. The specific number of replicated flows for each flow depends on the distribution of the flow size and its size.

1) Model of the assignment: The impact of replication on large flows is decided by the fraction of total bytes from replication. To reduce the impact of replication on large flows, the following constraint is imposed on the number of replicated flows:

$$\sum_{i=1}^{S_L} i * \pi_i * x_i < \varepsilon * \Theta,$$

$$\forall i \in \{1, 2, ..., S_L\}, x_i \in \{0, 1, ..., ecpNum - 1\}$$

where Θ is the total traffic of all flows from historical data, π_i is the number of flows with i KB in the historical data, x_i is the number of the replicated flows that will be assigned to short flows with i KB traffic. The left of the inequality is the total traffic from replicated flows. $\varepsilon \in (0, 1)$ is used to limit the traffic from replication to an acceptable percentage of the total traffic. We limit $repNum_k$ to be less than $ecpNum - 1$, where $ecpNum$ is the number of different equal-cost paths between any pair of hosts at different pods, because when over $ecpNum$ replicated flows are created, there is at least a replicated flow in each equal-cost path.

We model the process that a short flow and its x_i replicated ones traverse data center network as $x_i + 1$ Bernoulli trials. In each Bernoulli trial, it's regarded as a failure that all of the $x_i + 1$ flows collide with a large flow. Let p be the probability that failure happens in a trial. If all of the $x_i + 1$ flows experience a large flow, the replication is meaningless and the gain of the replication is 0. Otherwise the gain is 1. We try to maximize the expected total gain for each short flow$\sum_{i=1}^{S_L}\pi_i(1 - p^{x_i+1})$, which is equivalent to minimize the following formula: $\sum_{i=1}^{S_L} \pi_i p^{x_i+1}$.

Algorithm 3 Replicated Flows Number Assignment

```
procedure RepNumAssign($\varepsilon$, $\Theta$, $\pi$, $S_L$)
    for $i \leftarrow 1$ to $S_L$ do
        $\eta_i \leftarrow i * \pi_i$
        $repNum_i \leftarrow 0$
    end for
    $sum \leftarrow 0$
    $T \leftarrow \{1, 2, 3, ..., S_L\}$
    repeat
        $k \leftarrow \arg\max_{i \in T}\{\pi_i * p^{repNum_i+1}\}$
        while $sum + \eta_k > \varepsilon * \Theta$ do
            $T \leftarrow T - \{k\}$
            $k \leftarrow \arg\max_{i \in T}\{\pi_i * p^{repNum_i+1}\}$
        end while
        if $\pi_k = 0$ or $repNum_k \geq ecpNum$ then
            $T \leftarrow T - \{k\}$
        else
            $repNum_k \leftarrow repNum_k + 1$
            $sum \leftarrow sum + \eta_k$
        end if
    until $T == \phi$
    return $repNum$
end procedure
```

2) A greedy solution: We devise an algorithm to solve the optimization problem (Algorithm 3). The basic idea is to increase the $repNum_k$ corresponding to the most $\pi_k * p^{repNum_k+1}$ by one because it reduces the current objective function value the most.

The time complexity of the algorithm is $O(\varepsilon\Theta)$. The algorithm is quick enough because SmartRep does not require it to reflect the pattern of new data in real-time. This algorithm are ran when new historical data are collected and the period between two runs can be long.

3) Setting of parameters: When the services hosted in a data center don't change, the traffic pattern will keep constant and historical data, which is easily gathered in private data centers, can be used to reflect the traffic patter of the data center. When p is too high, no replicated flows will be created for the minority of the short flows, which increase the tail FCT of short flows so much. Therefore, we set p as 0.1. In our experience, when ε is set below 0.03, the throughput degradation of large flows will be very small.

IV. Evaluation

Now we evaluate SmartRep through NS-2 [**?**] simulations.

A. Evaluation Settings

We used a fat-tree topology, which is commonly used in current data centers [**?**] [**?**]. The addressing method designed in [**?**] is used. The fabric consists of 10 pods, each of which contains 5 edge switches and 5 aggregation switches. Each switch in the fabric has 10 10Gbps ports. The end-to-end round-trip time is $12\mu s$. There are 5 different equal-paths for any pair of hosts at different pods.

Two empirical workloads are used to reflect traffic patterns in production data centers. One is from a data center mostly running web search [**?**], and the other is from a data center mostly running data mining jobs [**?**]. Both workloads are mix of mice flows and elephant flows and follow a long-tail pattern. Flows are generated between random pairs of hosts following a Possion process with load varying from 0.1 to 0.8. We simulate 2s worth of traffic at each run and 10 runs for each load.

We choose CRC16 as the hash function of ECMP. The source ports of replicated flows for each short flow is obtained in advance using HCA. Our method is mainly compared with the following ones: (i) **TCP:** Standard TCP-New Reno with DropTail queues is used as the baseline of our evaluation; (ii) **RepFlow:** One replicated flow is created for each short flow. The replicated flow is exactly the same as the original flow except that its destination port is equal to its original one's plus one; (iii) **FullRep:** The number of replicated flows created for each short flow is set equal to the number of equal-cost paths between any pair of hosts at different pods.

B. Evaluation Results

SmartRep on RepFlow and TCP: Fig. 4 and Fig. 5 show the FCT performance of different flows with the web search and data mining workloads under different levels of network load. With the data mining workload, SmartRep reduces the mean FCT of short flows by more than 70% and 50%, and improve the tail FCT of short flows by more than 50% and 40% at nearly all loads, compared with TCP and RepFlow, respectively. With the web search workload, SmartRep reduces the tail FCT of short flows by over 50% and 25%, and improve the tail FCT of short flows by over 50% and 35% at nearly all loads, compared with TCP and RepFlow, respectively. All of these simulation results demonstrate the ability of SmartRep to avoid the flow hash collision and use multiple equal-cost paths. Meanwhile, large flows under SmartRep suffer a negligible FCT increase, as Fig. 4(c) and Fig. 5(c) show.

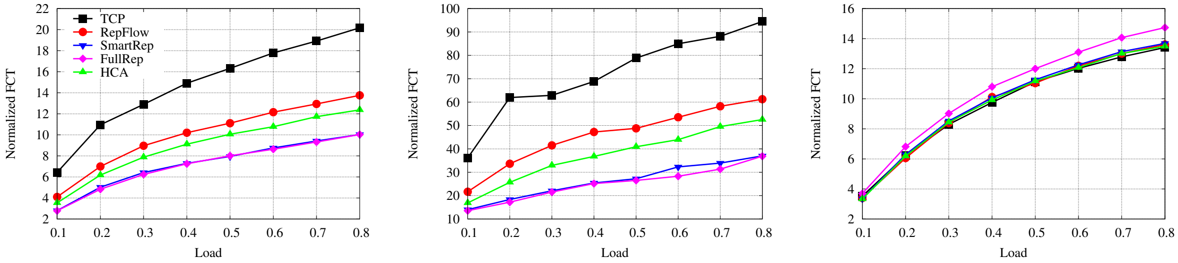
SmartRep on FullRep: As Fig. 5 shows, FullRep achieves almost the same mean and tail FCT as SmartRep. However, FullRep increases the mean FCT of large flows by over 10% for loads from 0.4 to 0.8 while SmartRep has little impact on the mean FCT of large flows. Note that we don't plot the FCTs under FullRep with the data mining workload, because all the FCTs under FullRep are exactly the same as SmartRep. The reason is that the fraction of traffic from short flows is very small and it is allowed to create enough replicated flows for each flow to utilize much more or even all of the equal-cost paths while imposing negligible increase. This is also why SmartRep has a better FCT improvement to RepFlow with the data mining workload than with the web search workload.

Performance and efficiency of HCA: We also evaluate the performance of HCA. Fig. 4 and Fig. 5 show HCA improves RepFlow by about 15% in both mean and 99th percentile FCT. This result confirms the analysis in Section II.

According to the analysis in Section III, the running time of HCA is up to the times of invoking procedure traceRoute. The numbers of replicated flows assigned to short flows by SmartRep vary from 0 to 5, and the average number of invoking procedure traceRoute is 15.1.

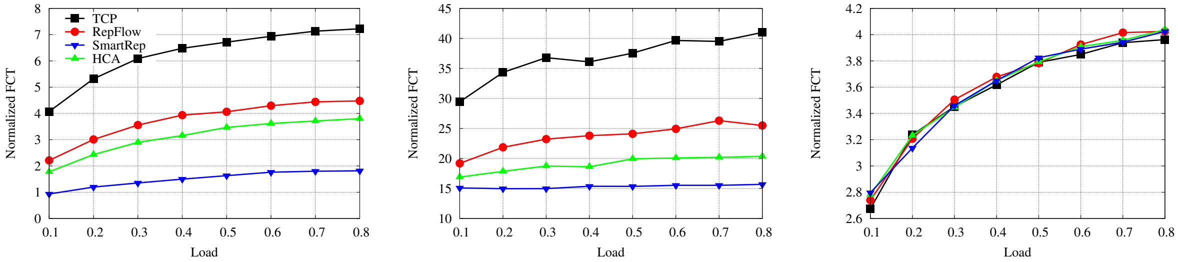
V. Conclusion

In this paper, we firstly study the effect of hash collision on performance of replication based FCT method and the benefit of creating multiple replicated flows for each short flow. Then we propose SmartRep, consisting of a *traceroute* based method HCA to overcome the hash collision problem and RepNumAssign to find the optimal number of replicated flows created for each short flow while limiting the overhead. SmartRep is evaluated in extensive simulations and improves prior replication based work by 25%-50% in both mean and 99-percentile FCT with negligible impact on large flows.



(a) Short flow mean FCT (b) Short flow tail FCT (c) Large flow mean FCT

Fig. 4: Normalized FCT breakdown for different flows with a 10-pod Fat-tree and web search workload in NS-2.



(a) Short flow mean FCT (b) Short flow tail FCT (c) Large flow mean FCT

Fig. 5: Normalized FCT breakdown for different flows with a 10-pod Fat-tree and data mining workload in NS-2.

Acknowledgment

This work is partially supported by the National Natural Science Foundation of China under Grant No. 61202113, 61472181, 61321491, 91218302; Jiangsu Key Technique Project (industry) under Grant No. BE2013116; EU FP7 IRSES MobileCloud Project under Grant No. 612212; the Fundamental Research Funds for the Central Universities under Grant No. 20620140513; JSPS KAKENHI Grant Number 25880002, 26730056 and JSPS A3 Foresight Program. This work is also partially supported by Collaborative Innovation Center of Novel Software Technology and Industrialization.

Appendix

Proof of Theorem 1: When one replicated flow is created for each short flow, the load increase from ρ to $(1+\epsilon)\rho$, where ϵ is the fraction of total bytes from short flows. It means that when a flow traverse a path, the flow meets a busy queue with probabilty $(1+\epsilon)\rho$. Let a be a short flow and b be its replicated flow. Let ε be the event that both flows meet a busy network, ε_1 be the event that both a and b are routed to the same path, i.e., hash collision happens, and ε_2 be the event that a and b traverse different paths. Then $Pr[\varepsilon_2] = 1 - Pr[\varepsilon_1]$ and $Pr[\varepsilon] = Pr[\varepsilon|\varepsilon_1] \cdot Pr[\varepsilon_1] + Pr[\varepsilon|\varepsilon_2] \cdot Pr[\varepsilon_2]$. When a and b traverse the same path, they meet a busy queue with probability $Pr[\varepsilon|\varepsilon_1] = (1+\epsilon)\rho$. When a and b traverse different paths, they are independent and the probability that both of them meet busy queues is $Pr[\varepsilon|\varepsilon_2] = (1+\epsilon)^2\rho^2$. Let $Pr[\varepsilon_1] = p$, then the probability that both a and b meet busy queues is

$$\begin{aligned} Pr[\varepsilon] &= (1+\epsilon)\rho p + (1+\epsilon)^2\rho^2(1-p) \\ &= (1+\epsilon)\rho[p + (1+\epsilon)\rho(1-p)] \end{aligned}$$