Scheduling Mix-flows in Commodity Datacenters with Karuna

Li Chen, Kai Chen, Wei Bai, Mohammad Alizadeh(MIT)
SING Group, CSE Department
Hong Kong University of Science and Technology

ABSTRACT

Cloud applications generate a mix of flows with and without deadlines. Scheduling such *mix-flows* is a key challenge; our experiments show that trivially combining existing schemes for deadline/non-deadline flows is problematic. For example, prioritizing deadline flows hurts flow completion time (FCT) for non-deadline flows, with minor improvement for deadline miss rate.

We present Karuna, a first systematic solution for scheduling mix-flows. Our key insight is that deadline flows should meet their deadlines while *minimally* impacting the FCT of non-deadline flows. To achieve this goal, we design a novel Minimal-impact Congestion control Protocol (MCP) that handles deadline flows with as little bandwidth as possible. For non-deadline flows, we extend an existing FCT minimization scheme to schedule flows with known and unknown sizes. Karuna requires no switch modifications and is backward compatible with legacy TCP/IP stacks. Our testbed experiments and simulations show that Karuna effectively schedules mix-flows, for example, reducing the 95th percentile FCT of non-deadline flows by up to 47.78% at high load compared to pFabric, while maintaining low ($<5.8\%$) deadline miss rate.

CCS Concepts

•**Networks** → **Transport protocols;**

Keywords

Datacenter network, Deadline, Flow scheduling

1. INTRODUCTION

User-facing datacenter applications (web search, social networks, retail, recommendation systems, etc.) often have stringent latency requirements, and generate a diverse mix of short and long flows with strict deadlines [3, 22, 38, 39]. Flows that fail to finish within their deadlines are excluded from the results, hurting user experience, wasting network bandwidth, and incurring provider revenue loss [39]. Yet, today's datacenter transport protocols such as TCP, given their Internet origins, are oblivious to flow deadlines and perform poorly. For example, it has been shown that a substantial fraction (from 7% to over 25%) of flow deadlines are not met using TCP in a study of multiple production DCNs [39].

Meanwhile, flows of other applications have different performance requirements; for example, parallel computing applications, VM migration, and data backups impose no specific deadline on flows but generally desire shorter completion time. Consequently, a key question is: how to schedule such a mix of flows with and without deadlines? To handle the mixture, a good scheduling solution should *simultaneously*:

- Maximize deadline meet rate for deadline flows.
- Minimize average flow completion time (FCT) for non-deadline flows.
- Be practical and readily-deployable with commodity hardware in today's DCNs.

While there are many recent DCN flow scheduling solutions [3–5, 22, 30, 38, 39], they largely ignore the mix-flow scheduling problem and cannot meet all of the above goals. For example, PDQ [22] and PIAS [5] do not consider the mix-flow scenario, while pFabric [4] simply prioritizes deadline flows over non-deadline traffic, which is problematic (§2). Furthermore, many of these solutions [4, 22, 30, 39] require non-trivial switch modifications or complex arbitration control planes, making them hard to deploy in practice.

We observe that the main reason prior solutions such as pFabric [4], or more generally, EDF-based (Earliest Deadline First) scheduling schemes, suffer in the mix-flow scenario is that they complete deadline flows *too* aggressively, thus hurting non-deadline flows. For example, since pFabric strictly prioritizes deadline flows, they aggressively take *all* available bandwidth and (unnecessarily) complete far before their deadlines, at the expense of increasing FCT for non-deadline flows. The impact on non-deadline flows worsens with more deadline traffic, but is severe even when a small fraction (e.g., 5%) of all traffic has deadlines (see §2.2).

SIGCOMM '16, August 22–26, 2016, Florianopolis, Brazil

 ISBN 978-1-4503-4193-6/16/08…$15.00

DOI: http://dx.doi.org/10.1145/2934872.2934888

Our key insight to solve the mix-flow scheduling problem is that deadline flows, when fulfilling their primary goal of meeting deadlines, should *minimally* impact FCT for non-deadline flows. This is based on the assumption that deadlines reflect actual performance requirements of applications, and there is little utility in finishing a flow earlier than its deadline. To this end, we design MCP, a novel distributed rate control protocol for deadline flows. MCP takes the minimum bandwidth needed to complete deadline flows barely before their deadlines (§4), thereby leaving maximal bandwidth to complete non-deadline flows quickly.

MCP flows walk a thin line between minimal-impact completion and missing deadlines, and must therefore be protected from any aggressive non-deadline flows. Thus, we leverage priority queues available in commodity switches and place MCP-controlled deadline flows in the highest priority queue. For non-deadline flows, we place them in the lower priority queues and use an aggressive rate control (e.g., DCTCP [3]) to take the bandwidth left over by MCP. Further, we extend the PIAS scheduling algorithm [6] to jointly schedule non-deadline flows with known or unknown sizes among the multiple lower priority queues, in order to minimize their FCT (§5.2).

Taken together, we develop Karuna, a mix-flow scheduling system that *simultaneously* maximizes deadline meet rate for deadline flows, and minimizes FCT for non-deadline flows. Essentially, Karuna trades off higher FCT for deadline flows, for which the key performance requirement is meeting deadlines, to improve FCT for non-deadline flows. Karuna makes this tradeoff deliberately to tackle this multi-faceted mix-flow problem. Karuna does not require any switch hardware modifications or complex control plane for rate arbitration, and is backward-compatible with legacy TCP/IP stacks. We further identify and address a few practical issues such as starvation and traffic variation (§6).

We implement a Karuna prototype (§7) and deploy it on a small testbed with 16 servers and a Broadcom Gigabit Ethernet switch. On the end host, we implement Karuna as a Linux kernel module that resides, as a shim layer, between the Network Interface Card (NIC) driver and TCP/IP stack, without changing any TCP/IP code. On the switch, we enable priority queueing and Explicit Congestion Notification (ECN), which are both standard features on current switching chips. Our implementation experience suggests that Karuna is readily deployable in existing commodity datacenters.

We evaluate Karuna using testbed experiments and large-scale ns-3 simulations with realistic workloads (§8). Our results show that Karuna maintains high deadline completion while significantly lowering FCT for non-deadline flows. For example, it reduces the 95th percentile FCT of non-deadline flows by up to 47.78% at heavy load compared to a clean-slate design, pFabric [4], while still maintaining low (<5.8%) deadline miss rate. Furthermore, our simulations show that Karuna is effective in handling starvation, and is resilient to traffic variations and multiple bottlenecks.

2. BACKGROUND AND MOTIVATION

To motivate our design, we identify 3 types of flows in DCNs, and show performance trade-offs with existing scheduling schemes for mix-flows.

2.1 Application examples of 3 flow types

Type 1: Flows with deadlines: Applications such as web search, recommendation, and advertisement usually generate flows with deadlines [38].[1] User experience of these applications is affected by latency, and hence they enforce strict deadlines on network flows. The flows are useful to the application if, and only if, they complete within the deadline [39]. In other words, the primary performance metric for this type of traffic is the *deadline miss rate*, i.e. the fraction of flows that miss their deadline.[2] If the traffic is only of this type, EDF-based proposals (e.g., pFabric [4] with priority given to flows with shorter deadlines, or PDQ [22] using remaining deadline as the flow criticality) are the best known schemes to minimize deadline miss rate.[3]

Type 2: Flows without deadlines but known sizes: Applications such as VM migration and data backup generate flows without strict latency requirements, but generally desire short completion times. Further, the sizes of these flows are usually known before transmission. If the traffic is only of this type, SJF-based proposals (e.g., PASE [30] and pFabric with priority given to flows with smaller sizes, or PDQ and pFabric using remaining size as flow criticality) are the best known schemes to minimize average FCT (AFCT).

Type 3: Flows without deadlines or known sizes: A number of applications are unable to provide size/deadline information at the start of their flows, e.g. database access and HTTP chunked transfer [6]. If the traffic is only of this type, best-effort schemes like DCTCP [3] are the predominant solutions, while recently PIAS [6] achieves better FCT than DCTCP by emulating SJF without knowing flow sizes.

Observation 1: The 3 types of flows coexist in DCNs. Each type alone has well-known scheduling solutions (based on SJF or EDF). But there is little prior work on how to schedule a mix of flows with different types.

2.2 Trade-offs in different scheduling schemes

We use ns-3 [33] simulations to show that applying criticality-based scheduling schemes (SJF or EDF) hurts the performance of different types of flows in mixed scenarios. In these experiments, the sender and receiver are connected to a switch, and the NIC capacity of both servers are 1Gbps. We use DCTCP for the end host rate control. We simulate a query/response application for the deadline flows. The

[1]In most applications where flows have deadlines, the sizes of these flows are also known in advance [39].

[2]This is equivalent to the *application throughput* metric (fraction of flows that meet their deadline) used in prior work [4, 22, 39].

[3]EDF is optimal because if there exist any scheduling discipline that can satisfy a set of deadlines, EDF also satisfies them. [29]

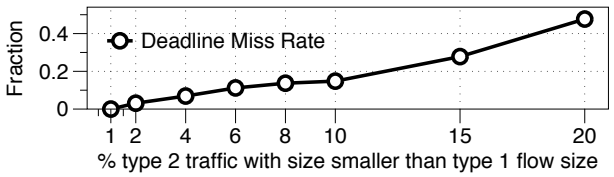

Figure 1: SJF hurts type 1 flows. Background flows sizes are drawn from the Data Mining workload in Figure 12. Type 1 flows are generated with deadline of 10ms, and their sizes are exactly the x-th percentile of the type 2 flows.

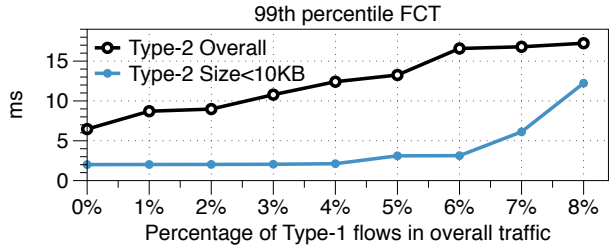

Figure 2: EDF hurts type 2 flows, especially smaller ones. Type 2 flows are scheduled using SJF, and their sizes are drawn as in Figure 1. Sizes of type 1 flows are 5KB, and their deadlines are drawn from an exponential distribution with mean 10ms. We vary the percentage of type 1 flows, and collect the FCT of type 2 flows.

queries are generated with a Poisson process, and we control the flow sizes and deadlines of response flows (type 1). Background flows (type 2) are generated using a Poisson process with flow sizes drawn from realistic traffic traces, and the average load of background traffic is 800Mbps (80% load). There are no type 3 flows in these experiments.

Case against pure SJF: In the first experiment, we schedule flows strictly based on their sizes (SJF). Figure 1 shows that the deadline miss rate of deadline flows is undermined by SJF. When the sizes of deadline flows are smaller than 1% of the type 2 flows, the deadline miss rate is 0; when the sizes are the 20th percentile (13KB), more than 40% of the responses misses their deadlines.

Observation 2: Simply applying SJF hurts type 1 flows. This is because size alone decides which flow goes first, which prevents deadline flows, especially the larger ones, from completing before their deadlines.

Case against pure EDF: Next, we instead use EDF to schedule the deadline flows, and let the type 1 flows have strict priority over type 2 flows. Type 2 flows are still scheduled using SJF. This is the strategy adopted in prior works [4, 22, 30]. In Figure 2, we observe that the tail latency of type 2 flows increases with the percentage of deadline traffic. For short type 2 flows (latency-sensitive), we observe a $\sim 5\times$ increase in tail latency when the percentage of type 1 traffic increases from 0 to 8%. Because type 1 traffic is prioritized and aggressively takes up available bandwidth using DCTCP, the more type 1 flows, the worse the performance for other types of traffic.

Observation 3: Simple combination of EDF (for type 1 flows) and SJF (for type 2&3 flows), with priority given to type 1 flows, hurts type 2&3 flows. Type 1 flows complete quickly with small FCT, which is unnecessary for meeting their deadlines, at the cost of higher tail latency for other flows.

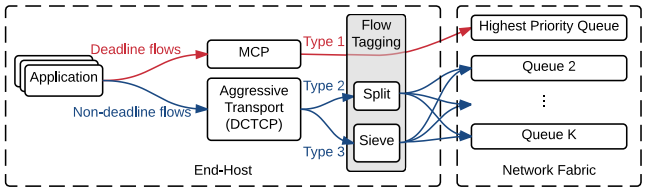

Figure 3: Karuna system overview.

3. SYSTEM OVERVIEW

This section outlines our design, Karuna (Figure 3). Karuna resolves the tension between different types of flows according to their respective goals. Since the primary goal of type 1 flows is to meet their deadlines, Karuna uses the minimum bandwidth required for these flows to complete just before their deadline, thus leaving *maximal* bandwidth to type 2&3 flows to optimize their FCT.

Deadline flows: A naive rate-control scheme to achieve near-deadline completion is to always set the rate of a deadline flow to its expected rate, M/δ, where M is the remaining size, and δ the remaining time to deadline. However, this scheme fails [39] when many deadline flows collide due to lack of congestion control. Moreover, a similar but more sophisticated rate-control scheme, D^3 [39], has a "priority-inversion" problem [22] due to its greedy algorithm (details in §4). Thus, to achieve the desired behavior, we design a new minimal-impact congestion control protocol for deadline flows, MCP, which reacts to network congestion and controls the rate conservatively to *just* meet the deadline (§4). We place deadline flows in the highest priority queue, so they are protected from the aggressive non-deadline flows.

Non-deadline flows: Type 2&3 flows reside in multiple lower priority queues (Queue 2–K)[4] and are regulated by aggressive protocols (e.g., DCTCP [3]) at end hosts to use all bandwidth left over by type 1. We further schedule, *i.e. split* or *sieve*, them among these multiple lower priority queues based on their sizes to minimize FCT:

- If their sizes are known a priori (type 2 flows), they are directly *split* into different priority queues based on their sizes in the spirit of SJF .
- If their sizes are unknown (type 3 flows), they are gradually *sieved* from higher priority queues to lower priority queues according to the number of bytes sent, which effectively emulates SJF without knowing flow sizes.

How to sieve type 3 flows has been explored in [6]. However, in Karuna, we need to handle both type 2 and 3 flows, which is a different problem. We thus extend the technique

[4]K is the number of priority queues supported by commodity switches, usually 4–8 [5, 6].

in [6] to jointly solve the problem of splitting type 2 flows and sieving type 3 flows (see §5 and Appendix A).

4. HANDLING DEADLINE FLOWS WITH MCP

Deadline flows are given the highest priority in our design, and their rates are throttled so that they finish transmission just before the deadlines. The key question is how to throttle the flows to just meet the deadlines in an environment where flow arrive and depart dynamically.

At first glance, D^3 [39], which sets the flow rate to $\gamma = M/\delta$ plus the fair share of the remaining link bandwidth after subtracting the demand of all deadline flows, seems to be a suitable solution. However, D^3 suffers from the priority inversion problem [38], as shown in the example in Figure 4. D^3 greedily allocates rates to flows that arrive earlier[5]. In Figure 4(a), flow C misses its deadline because the earlier flows A&B do not relinquish their bandwidth; an optimal schedule in (b) shows that flows A&B can give up bandwidth for flow C to complete before its deadline while still meeting their own deadlines. D^{2}TCP [38] overcomes this problem with a deadline-aware congestion window update function, which allows each flow to achieve its deadline *on its own*. Nonetheless, D^{2}TCP is not suitable for use in the highest priority in Karuna, because it aggressively takes over all available bandwidth, affecting non-deadline flows.

Therefore, we proceed to design MCP[6] for Karuna, which allows flows to achieve deadlines while minimally impacting non-deadline flows. In what follows, we formulate the near-deadline completion congestion control problem as a stochastic optimization problem, and solve it to derive MCP's congestion window update function.

4.1 Problem formulation

We first introduce the system model. Then, we formulate the problem and transform it to a convex problem (§4.1.1). By solving the transformed problem, we derive the optimal congestion window update function (§4.1.2).

System model: Consider L logical links, each with a capacity of C_l bits per second (bps). In the network, the total number of active sessions is S. At time t, session s transmits exactly one flow at a rate of $x_s(t)$ bps. The remaining flow size is denoted as $M_s(t)$, and the remaining time to deadline is $\delta_s(t)$. Applications pass deadline information to the transport layer (§7) in the request to send data. Define $\gamma_s(t)=M_s(t)/\delta_s(t)$ as the expected rate for session s at time t. The expected rate in the next Round Trip Time (RTT) is $\gamma_s(t+\tau_s(t))=\frac{M_s(t)-\tau_s(t)x_s(t)}{\delta_s(t)-\tau_s(t)}$, where $\tau_s(t)$ is the RTT of flow s at t. We assume that the flow from session s is routed through a fixed set of links $L(s)$. For link l, denote y_l as the aggregate input rate, $y_l=\sum_{s\in S(l)}x_s$, where the set of flows that pass through link l is $S(l)$.

[5]In dynamic setting, the allocation of rates to maximize deadline completion is NP-Complete [9], and D^3 chooses a greedy approach.

[6]MCP was first explored in our earlier paper [11] with preliminary simulation results for only type 1 flows.

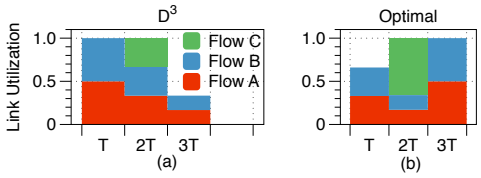

Figure 4: Link capacity is C. Flows A&B have deadline $3T$, size CT, and arrive at t=0. Flow C has deadline T, size $\frac{2CT}{3}$, and arrives at t=T. We assume immediate convergence.

Minimal impact: Our objective in designing MCP is to limit the impact of deadline flows on other traffic. Instead of using aggregate rates of deadline flows, we choose per-packet latency introduced by deadline flows to quantify their impact, because short non-deadline flows are more sensitive to per-packet delays and suffer the most from deadline flows in the high priority queue, as was shown in Figure 2.

We therefore use the long term time-averaged per-packet delay as the minimization objective. Denote $d_l(y_l)$ as the delay that a packet experiences on link l with aggregate arrival rate y_l. For session s, the average packet delay is $\sum_{l\in L(s)}d_l(y_l)$. We assume infinite buffer for all links, and the $d_l(y_l)$ is a positive, convex, and increasing function. We define the objective function as the average of the summation of per-packet delay of every source over time.

$$P(\mathbf{y}(t))=\lim_{T\to\infty}\frac{1}{T}\sum_{t=0}^{T-1}\sum_s\{\sum_{l\in L(s)}d_l(y_l(t))\} \quad (1)$$

where $\mathbf{y}(t)=[y_l(t)]_{l=1}^{L}$, a $L\times 1$ vector.

Network stability: To stabilize the queues, we require that each source control its sending rate $x_s(t)$, so that the aggregated rates at each link l, $y_l(t)=\sum_{s\in S(l)}x_s(t)$, will satisfy: $y_l(t)\le C_l, \forall l$. In practice, temporary overloading is allowed due to buffering in switches, thus we relax this constraint into the objective with a penalty term, μ, so flows that exceed the link capacity are penalized.

$$\tilde{P}(\mathbf{y}(t))=\lim_{T\to\infty}\frac{1}{T}\sum_{t=0}^{T-1}(\sum_s\{\sum_{l\in L(s)}d_l(y_l(t))\}+\mu\sum_l(y_l(t)-C_l)) \quad (2)$$

Deadline constraint: To complete within a flow's deadline, we require the transmission rate to be larger than or equal to the expected rate, $x_s(t)-\gamma_s(t)\ge 0$, $\forall s,t$.We relax this constraint with its long term time-average:

$$\lim_{t\to\infty}\frac{\sum_0^t(\gamma_s(t)-x_s(t))}{t}\le 0, \forall s \quad (3)$$

which essentially says that, for every flow that requires γ_s, the transmission rate x_s is on average larger than γ_s to complete before its deadline. This is a relaxation, as realistic flows will not last forever.

Formulation: Our goal is to derive optimal source rates ($\mathbf{x}(t)=[x_s(t)]_{s=1}^{S}$, a $S\times 1$ vector) to minimize long-term per-packet delay while completing within deadlines. Thus, we formulate the following stochastic minimization problem (4)

to encapsulate the above objective and constraints.

$$\min_{\mathbf{x}(t)} \tilde{P}(\mathbf{y}(t))$$
$$\text{subject to } x_s(t)>0,\forall s;\ y_l(t)=\sum_{s\in S(l)} x_s(t),\forall l; \quad (4)$$
$$\lim_{t\to\infty}\frac{\sum_0^t(\gamma_s(t)-x_s(t))}{t}\leq 0,\forall s$$

4.1.1 Application of Lyapunov optimization

Next, we apply the Lyapunov optimization framework [32] to transform this minimization problem to a convex problem, and then derive an optimal congestion window update function (§4.1.2) based on the optimal solution to the transformed convex problem. The *drift-plus-penalty method* [32] is the key technique in Lyapunov optimization, which stabilizes a queueing network while also optimizing the time-average of an objective (e.g. per-packet latency).

Here we explain the application of drift-plus penalty method to Problem 4 to transform it into a convex programming problem. To use this framework, a solution to our problem must address the following aspects:

Queue stability at all links: We first define a scalar measure $L(t)$ of the stability of the queueing system at time t, which is called *Lyapunov function* in control theory. For our model, we use the quadratic Lyapunov function: $L(t)=\frac{1}{2}\sum_l Q_l(t)^2$. The Lyapunov drift is defined as $\Delta(t_k)=L(t_{k+1})-L(t_k)$, the difference between 2 consecutive time instants. The stability of a queueing network is achieved by taking control actions that make the Lyapunov function drift in the negative direction towards zero. With drift-plus-penalty method, MCP controls the transmission rates of the sources to minimize an upperbound to the network Lyapunov drift, so as to ensure network stability.

Deadline constraint: To handle the deadline constraints in (4), we transform them into virtual queues [32]. Consider a virtual queue $Z_s(t)$ for flow s at time t, where the expected rate is the input and the actual rate is the output.

$$Z_s(t+\tau_s(t))=[Z_s(t)+\gamma_s(t)-x_s(t)]^+,\forall s \quad (5)$$

For the virtual queues to be stable, we have:

$$\lim_{t\to\infty}\sum_0^t\gamma_s(t)/t\leq\lim_{t\to\infty}\sum_0^t x_s(t)/t \quad (6)$$

Similar to the packet queues at the switches, the virtual queues can also be stabilized by minimizing the Lyapunov drift. To include the virtual queues, the Lyapunov function becomes $L(t)=\frac{1}{2}(\sum_l Q_l(t)^2+\sum_s Z_s(t)^2)$. If the virtual queues are stabilized, the deadline constraint (3) is also achieved, because the input $\gamma_s(t)$ of the virtual queue is on average smaller than the output $x_s(t)$.

Minimization of impact (per-packet latency): The above two points concern the "drift". We also use a "penalty" term to achieve MCP's goal of minimizing impact to other traffic. We formulate the drift-plus-penalty as $\Delta(t_k)+V\tilde{P}_0(\mathbf{y}(t_k))$. where V is a non-negative weight chosen to ensure the time average of $\tilde{P}_0(t)$ is arbitrarily close (within $O(1/V)$) to optimal, with a corresponding $O(V)$ tradeoff in average queue size [31]. By minimizing an upperbound of the drift-plus-penalty expression, the time average of per-packet latency can be minimized while stabilizing the network of packet queues and virtual queues.

Convex Problem: Finally, we arrive at the following covex problem:

$$\min_{\mathbf{x}(t)}\sum_s\{V\sum_{l\in L(s)} d_l(y_l(t))+\frac{Z_s(t)\gamma_s(t)}{x_s(t)}+\sum_{l\in L(s)}(Q_l(t)+\mu)x_s(t)\} \quad (7)$$
$$\text{subject to } y_l(t)=\sum_{s\in S(l)}x_s(t),\forall l$$

At a high-level, we transform the the long term ($t\to\infty$) stochastic delay minimization problem (4) into a drift-plus-penalty minimization problem (7) at every update instant t. To solve the transformed problem, we develop an adaptive source rate control algorithm.

4.1.2 Optimal congestion window update function

By considering the properties of the optimal solution and the KKT conditions [8] of the above problem, we obtain a primal algorithm to achieve optimality for (7). Eq.(8) stabilizes the queueing system and minimizes the overall per-packet delay of the network:

$$\frac{d}{dt}x_s(t)=f_s'(x_s(t))-\sum_{l\in L(s)}\lambda_l(t), \quad (8)$$

where $f_s(x_s)=-Z_s(t)\frac{\gamma_s(t)}{x_s(t)}-Q_s(t)x_s(t)$, $\lambda_l(t)=d_l'(y_l(t))$. Interested reader may refer to MCP technical report [10] for derivation.

Each flow should adjust its transmission rate according to (8), which can be re-written as:

$$\frac{d}{dt}x_s(t)=\Theta(\gamma_s(t),x_s(t))-\sum_{l\in L(s)}(Q_l(t)+\lambda_l(t)), \quad (9)$$

where $\Theta(\gamma_s(t),x_s(t))=\frac{Z_s(t)M_s(t)}{\tau_s(t)x_s^2(t)}=\frac{Z_s(t)\gamma_s(t)}{x_s^2(t)}$.

We then can derive the equivalent optimal congestion window update function:

$$W_s(t+\tau_s(t))\leftarrow W_s(t)+\tau_s(t)(\Theta(\gamma_s(t),\frac{W_s(t)}{\tau_s(t)})-\sum_{l\in L(s)}(Q_l(t)+\lambda_l(t))) \quad (10)$$

Consider the two terms that constitute the difference between window sizes:

- The first (*source term*), $\Theta(\gamma_s(t),x_s(t))$ where $x_s(t)=\frac{W_s(t)}{\tau_s(t)}$, is an increasing function of γ_s, and a decreasing function of x_s. A large γ for a flow means that this flow is more urgent, *i.e.* it has large remaining data to send and/or an imminent deadline. This term ensures that the flow will be more aggressive as its urgency grows.
- The second (*network term*), $\sum_{l\in L(s)}(Q_l(t)+\lambda_l(t))$, summarizes the congestion in the links along the path. If any link is congested, sources that use that link will reduce their transmission rates. This term makes MCP flows react to congestion.

Combining these two terms, the update function allows deadline flows meet their deadlines, while impacting the other flows as little as possible.

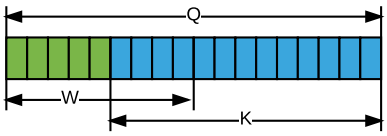

Figure 5: Queue length approximation.

4.2 MCP: From theory to practice

We now turn Eq.(10) into a practical algorithm.

4.2.1 ECN-based network term approximation

The source term can be obtained using information from upper layer applications (§7). However, obtaining the network term is not easy, as the sum of all link prices, λ_l, and queue lengths, Q_l, are needed along the path, and this aggregated path-level information is not directly available at the source. This sum can be stored in an additional field in the packet header, and each switch adds and stores its own price and queue length to this field for every packet. However, current commodity switches are not capable of such operations. For implementation, we use the readily available ECN functionality in commodity switches to estimate the network term.

Estimating queue lengths: The focus of our approximation is the aggregated queue lengths for each flow, Q. We denote F $(0\leq F\leq 1)$ as the fraction of packets that were marked in the last window of packets, and F is updated for every window of packets. Both DCTCP and D^{2}TCP compute F to estimate the extent of congestion, and MCP further exploits F to estimate queue lengths.

For our estimation, we abstract the DCN fabric as *one* switch. Current data center topologies enable high bisection bandwidth in the fabric, which pushes the bandwidth contention to the edge switches (assuming load-balancing is done properly) [4, 24]. In particular, the bottleneck link usually occurs at the egress switch of the fabric. Our estimation scheme therefore models the queueing behavior in the bottleneck switch.

Figure 5 illustrates how a source s estimates the queue length based on F. Assume the ECN threshold is K, the current queue length is Q, and the last window size of s is W. The fraction of packets in W of s that are marked by ECN should be $Q-K$. Therefore, we have $F\approx\frac{Q-K}{W}$, and thus $\hat{Q}\approx K+F\times W$, which is the estimate we use for the aggregated queue length for each source.

Estimating link prices: The link price represents the level of congestion at the bottleneck link, and, for mathematical tractability, we make the simplifying assumption that the link is an M/M/1 queue [27], $d(y)=1/(C-y)$. Therefore, the price of the link is proportional to the derivative of the delay function, $d'(y)=(C-y)^{-2}$.The arrival rate can be directly obtained by two consecutive queue estimations at the source: $\hat{y}(t)=\frac{\hat{Q}(t)-\hat{Q}(t-\tau_s(t))}{\tau_s(t)}$.

4.2.2 Practical MCP algorithm

Using the above estimation and Eq.(10), the congestion window update function of a practical MCP therefore is:

$$W_s(t+\tau_s(t))+=\tau_s(t)(\Theta(\gamma_s(t),\tfrac{W_s(t)}{\tau_s(t)})-(K+F_s(t)W_s(t)+\lambda(t))) \quad (11)$$

where $\lambda(t)=(C-\frac{F_s(t)W_s(t)-F_s(t-\tau_s(t))W_s(t-\tau_s(t))}{\tau_s(t)})^{-2}$.

We evaluate this algorithm in experiments (§8.1) and simulations (§8.2).

4.2.3 Early flow termination

Some flows may need to be terminated before their deadlines in order to ensure that other flows can meet theirs. Optimally selecting such flows has been shown to be NP-hard [22]. We propose an intuitive heuristic for MCP to terminate a flow when there is no chance for it to complete before its deadline: when the residual rate of the flow is larger than the link capacity, the flow will be aborted: $Z_s(t)>\min_{l\in L(s)}C_l$, where $Z_s(t)$ is the virtual queue of the flow, which stores the accumulative differences between the actual rates and the expected rates. $Z_s(t)$ is therefore a past performance indicator for this flow. This criterion implies that the capacity of the path is no longer sufficient for finishing before the deadline. Early termination of flows gives more opportunities for other flows to meet deadlines [39]. We evaluate this criterion in §8.1.3.

5. HANDLING NON-DEADLINE FLOWS

To consumer the bandwidth left over by type 1 flows, Karuna employs aggressive rate control such as DCTCP [3] for type 2&3 flows. Further, it leverages multiple lower priority queues in the network to minimize FCT of these flows.

5.1 Splitting type 2 flows

Since the sizes for type 2 flows are known, implementing SJF over them is conceptually simple. Karuna splits these flows to different priority queues according to their sizes: Smaller flows are sent to higher priority queues than larger flows. In our implementation, using limited number of priority queues, Karuna approximates SJF by assigning each priority to type 2 flows within a range of sizes. We denote $\{\beta_i\}$ as the splitting thresholds, so that a flow with size x is given priority i if $\beta_{i-1}<x\leq\beta_i$ ($\beta_0=0$ and $\beta_K=\infty$). With this, we formulate and solve an optimization problem to obtain the optimal splitting thresholds for different priorities (see Appendix A for details). Karuna effectively performs quantized SJF on type 2 flows using these thresholds.

5.2 Sieving type 3 flows

Type 3 flows differ from type 2 flows in that their sizes are unknown. As a result, there is no ground-truth for Karuna to directly split type 3 flows into different priority queues for approximating SJF. Inspired by PIAS [6] and Least Attained Service scheduling [15, 16], Karuna addresses this issue by sieving type 3 flows through multiple priority queues, which emulates SJF without knowing flow sizes.

Specifically, in the lifetime of a type 3 flow, Karuna sieves it from higher priority queues to lower priority queues based on the number of bytes it has sent. In this process, smaller flows are likely to complete in the first few priority queues,

whereas long flows eventually sink to the lowest priority queues. In this way, Karuna ensures that short type 3 flows are generally prioritized over long flows. All type 3 flows are at first given the highest priority, and they are moved to lower priorities as they send more bytes. The sieving thresholds are denoted as $\{\alpha_i\}$. A flow, which has transmitted x bytes, is given priority i if $\alpha_{i-1}<x\leq\alpha_i$.

The idea of sieving type 3 flows to minimize FCT has been well studied in [6]. However, in Karuna, we need to address both type 2 and type 3 flows together, which is a different problem. We reformulate the threshold optimization problem in [6] to jointly solve for the splitting thresholds for type 2 flows, and the sieving thresholds for type 3 flows. We pose this as a sum-of-quadratic-ratios problem, for which the solution in [6] is not applicable. Therefore, we relax the problem to a quadratic programming problem with linear constraints, and solve the relaxed problem (see Appendix A). We further investigate the effectiveness and robustness of the optimized thresholds in §8.2.1 and §8.2.3.

6. PRACTICAL ISSUES

We further examine several practical issues with Karuna, and discuss how to solve them.

Starvation: Using strict priority queueing in switches can potentially starve certain flows. A key benefit of Karuna is that it throttles deadline flows in the first priority queue using conservative rates, leaving the rest of bandwidth to non-deadline flows. In the extreme case, if deadline flows have to take up all the bandwidth for their deadlines, non-deadline flows will starve. There is not much a transport mechanism can do in such case, and the operators should consider increasing the network capacity.

In another scenario, deadline flows and small non-deadline flows in the higher priority queues can starve large non-deadline flows in the lowest priority queue. To counter this, we employ *flow aging* to elevate the priority of the flows that are being starved. Karuna identifies starved flows at end-hosts by observing time-out events. For example, when a flow experiences κ TCP timeouts, Karuna elevates this flow to a higher priority. In our implementation, if it is a type 2 flow, we re-split it to the queue based on the remaining size; if it is a type 3 flow, we move it to the highest priority queue for non-deadline flows (Queue 2) and let it re-sieve. We pick κ from $[2,10]$ uniformly at random for each flow, so that two long flows with similar size can avoid synchronization and congestion collapse [12]. Lifting the priorities of different flows with a random κ allows some of them to have higher priority earlier (and thus finish earlier). We note that such priority elevation may potentially lead to packet re-ordering, but it is not a serious issue, since TCP can handle it well for long flows. In our experiments, we found that flow aging is effective in solving starvation, and priority elevation does not have negative side-effects (see §8.1.3 and §8.2.2).

Traffic variation across time and space: Traffic in DCNs can vary across both time and space. Fortunately, such traffic variation does not affect type 1 flows, because type 1 flows are protected in highest priority queue. However, it does potentially affect types 2&3 flows, because they need to be split or sieved into multiple queues according to respective thresholds derived from a global traffic distribution. As a result, Karuna needs to dynamically update the thresholds as traffic varies.

It is challenging to accurately match the thresholds to the traffic. First, the distribution is always changing, and it takes time to collect sizes and distribute thresholds. Second, traffic also varies in space, and thresholds derived from a global distribution may not be perfect for each switch. When there is a mismatch between the traffic and thresholds, either packets of long flows are mis-split (type 2) or stay too long (type 3) in the higher priority queue, or packets of short flows are mis-split (type 2) or get prematurely sieved (type 3) to the lower priority queue. In both cases, the outcome is that short flows may queue behind long flows, increasing their latency.

We find that ECN used in network term estimation can also be used to to address this problem. With ECN, we can effectively keep low buffer occupation and minimize the impact of long flows on short flows. In our evaluation, we find this effectively addresses such thresholds-traffic mismatch and makes Karuna resilient to traffic variation (see §8.1.3 and §8.2.3).

One benefit of Karuna's resilience to traffic variation is that Karuna can afford to update thresholds infrequently. Thus, we periodically update the thresholds at a fixed time period, determined by the time it takes to collect/distribute information from/to the network (which depends on its scale). However, our threshold computation (see Appendix) is fast, taking at most seconds, irrespective of network scale.

Traffic statistics collection & threshold distribution: Computing thresholds requires flow size information from the entire network. It is impractical and time-consuming to collect and analyze complete traffic traces in large DCNs [34]. Instead, we design our end-host module (§7) to be capable of collecting flow information including sizes for all flows, and reporting to a centralized entity which computes the thresholds. The reporting and computation are done periodically, and in each period, a new set of thresholds are distributed to the end-host modules.

Coflow scheduling in Karuna: Coflow [13, 14, 17] is an important abstraction that identifies the inter-dependencies between flows. Karuna can facilitate coflow scheduling by exposing priorities in the network layer. Coflows with deadlines can be simply treated as type 1 flows in Karuna, and their deadlines can be met with the highest priority.

For the other 2 types, coflow scheduling requires application level coordination across multiple servers to determine the schedule–the order of transmission of coflows. With Karuna, such an order can be easily expressed with priorities in packets, and a similar idea, Smart Priority Class, has already been explored in a recent proposal (Baraat [17]) for decentralized coflow scheduling, where coflows mapped to a higher priority class get strict precedence over those in a lower priority class, and flows of the same class share bandwidth. Karuna can be readily employed in Baraat.

7. IMPLEMENTATION

We have implemented a Karuna prototype. We describe each component of the prototype in detail.

Information passing: For type 1 and 2 flows, Karuna needs to get the flow information (i.e., sizes and deadlines) to enforce flow scheduling. Such information is also required by previous works [4, 22, 30, 38, 39]. Flow information can be obtained by patching applications in user space. However, passing flow information down to the network stack in kernel space is still a challenge, which has not been explicitly discussed in prior works.

To address this, in our implementation of Karuna, we use `setsockopt` to set the `mark` for each packet sent through a socket. `mark` is an unsigned 32-bit integer variable of `sk_buff` structure in Linux kernel. By modifying the value of `mark` for each socket, we can easily deliver per-flow information into kernel space. Given that `mark` only has 32 bits, we use 12 bits for deadline information (ms) and the remaining 20 bits for size information (KB) in the implementation. Therefore, `mark` can represent 1GB flow size and 4s deadline at most, which can meet the requirements of most data center applications [3].

Packet tagging: This module maintains per-flow state and marks packets with a priority at end hosts. We implement it as a Linux kernel module. The packet tagging module hooks into the TX datapath at `Netfilter Local_Out`, residing between TCP/IP stacks and `TC`.

The operations of the packet tagging modules are as follows: 1) when a outgoing packet is intercepted by `Netfilter` hook, it will be directed to a hash-based flow table. 2) Each flow in the flow table is identified by the 5-tuple: src/dst IPs, src/dst ports and protocol. For each new outgoing packet, we identify the flow it belongs to (or create a new flow entry) and update per-flow state (extract flow size and deadline information from `mark` for type 1&2 flows and increase the amount of bytes sent for type 3 flows).[7] 3) Based on the flow information, we modify the the DSCP field in the IP header correspondingly to enforce packet priority.

Today's NICs use various offload mechanisms to reduce CPU overhead. When Large Segmentation Offloading (LSO) is enabled, the packet tagging module may not be able to set the right DSCP value for each individual MTU-sized packet with one large segment. To understand the impact of this inaccuracy, we measure the lengths of TCP segments with payload data in our 1G testbed. The average segment length is only 7.2KB which has little impact to packet tagging. We attribute this to the small TCP window size in the data center network with small bandwidth delay product (BDP). Ideally, packet tagging should be implemented in the NIC hardware to completely avoid this issue.

Rate control: Karuna employs MCP for type 1 flows and DCTCP [3] for type 2&3 flows at end hosts. For DCTCP implementation, we use DCTCP patch [2] for Linux kernel 2.6.38.3. We implement MCP as a `Netfilter` kernel module at receiver side inspired by [40]. The MCP module intercepts TCP packets of deadline flows and modifies the receive window size based on the MCP congestion control algorithm. This implementation choice avoids patching network stacks of different OS versions.

MCP updates the congestion window based on the RTT and the fraction of ECN marked packets each RTT (Eq.(11)). Therefore, accurate RTT estimation is important for MCP. We can only estimate RTT using TCP timestamp option since the traffic from the receiver to the sender may not be enough. However, the current TCP timestamp option is in millisecond granularity which cannot meet the requirement of data center networks. Similar to [40], we modify timestamp to microsecond granularity.

Switch configuration: Karuna only requires ECN and strict priority queueing, both of which are available in existing commodity switches [4, 5, 30]. We enforce strict priority queueing at the switches and classify packets based on the DSCP field. Like [3], we configure ECN marking based on the instant queue lengths with a single marking threshold.

We observe that some of today's commodity switching chips provide multiple ways to configure ECN marking. For our Broadcom BCM#56538, it supports ECN marking on different egress entities (queue, port and service pool). In per-queue ECN marking, each queue has its own marking threshold and performs independent ECN marking. In per-port ECN marking, each port is assigned a single marking threshold and packets are marked when the sum of all queue sizes belong to this port exceeds the marking threshold. Per-port ECN marking cannot provide the same isolation between queues as per-queue ECN. Interested readers may refer to [7] for detailed discussions on ECN marking schemes.

Despite this drawback, we still employ per-port ECN for two reasons. First, per-port ECN marking has higher burst tolerance. For per-queue ECN marking, each queue requires an ECN marking threshold h to fully utilize the link independently (e.g, DCTCP requires h=20 packets for 1G link). When all the queues are active, it may require the shared memory be at least the number of queues times the marking threshold, which cannot be supported by most shallow buffered commodity switches. (*e.g.* our Gigabit Pronto 3295 switch has 384 queues and 4MB shared memory for 48 ports in total). Second, per-port ECN marking can mitigate the starvation problem, as it pushes back high priority flows when many packets of low priority flows get queued in the switch (see §8.1.3).

8. EVALUATION

We evaluate Karuna using testbed experiments and ns-3 simulations. The result highlights include:

- Karuna maintains low deadline miss rate ($<5.8\%$) while greatly reducing the 95th percentile FCT of non-deadline flows by up to 47.78%, compared to pFabric, at heavy load (§8.2.1, we attribute this result to the facts shown in §8.1.1 and §8.1.2).

[7] For persistent TCP connections, we can periodically update flow states (*e.g.* , reset bytes sent to 0 for type 3 flows that are idle for some time).

Flow#	Size	Deadline	Start Time
1	14.4MB	20ms	1ms
2	48MB	120ms	1ms
3	3MB	5ms	50ms
4	0.5MB	10ms	80ms

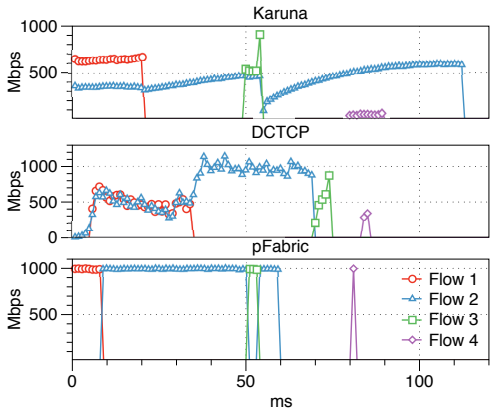

Figure 6: Karuna completes type 1 flows conservatively.

- The aging mechanism effectively addresses starvation and reduces FCT for long type 2&3 flows (§8.2.2).
- Karuna is resilient to traffic variation. Type 1 flows adapt to traffic dynamics well and keep close to 0 deadline miss rates in all scenarios. For type 2&3 flows, Karuna performs the best when the thresholds match the traffic, but it slightly degrades when mismatch occurs (§8.2.3, we attribute this partially to the fact in §8.1.3).
- While queue length estimation becomes inaccurate in extreme scenarios (oversubscribed network with multiple bottlenecks), Karuna still shows low (<7.9%) deadline miss rate for 2 bottlenecks at full load (§8.2.4).

8.1 Testbed experiments

Our testbed experiments focus on micro benchmarks using synthetic traffic. The main goal is to show how Karuna works, thus leading to the simulation results in §8.2.

Setting: We built a small testbed that consists of 16 servers, each with a 4-core Intel 2.8GHz CPU and 8G memory. The servers run Debian 6.0-64bit with Linux 2.6.38.3 kernel and are equipped with a Broadcom BCM5719 NetXtreme Gigabit Ethernet NICs. NIC offload mechanisms are enabled by default to reduce the CPU overhead. All the servers are connected to a Pronto 3295 48-port Gigabit Ethernet switch with 4MB shared memory. Our switch supports ECN and strict priority queuing with at most 8 class of service queues [1]. Our base RTT is $\sim$100μs.

Karuna uses 8 priority queues by default. We set per-port ECN marking threshold to be 30KB as [3] recommends. We develop a client/server model to generate traffic and measure the FCT on application layer. The client application, running on 1 server, generates requests to the other 15 servers to fetch data. Following [4, 6, 30], the requests are generated with a Poisson process.

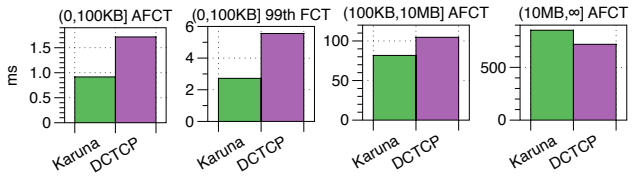

Figure 7: Karuna emulates SJF for type 2&3 flows.

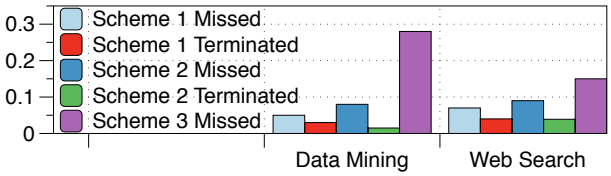

Figure 8: Effect of early flow termination.

8.1.1 *How Karuna meets deadlines of type 1 flows?*

MCP takes just enough bandwidth to meet the deadlines of type 1 flows, so that it can leave more bandwidth to type 2&3 flows. To demonstrate this, we showcase Karuna's behavior using a simple testbed experiment in Figure 6. In this experiment, four flows sharing a 1Gbps link. We observe that deadline flows in Karuna proceed conservatively as expected, and finish right before their deadlines. However, for DCTCP, lows 1 and 3 miss their deadlines by 21ms and 13ms, whereas flows 2 and 4 finish much earlier before their respective deadlines using more bandwidth. pFabric[8] meets all deadlines by consuming full bandwidth.

8.1.2 *How Karuna minimizes FCT of type 2&3 flows?*

Karuna optimizes FCT for type 2&3 flows by emulating SJF. Type 2 flows, given priorities based on size, are scheduled with quantized SJF. We proceed to show that type 3 flows are also indeed scheduled in SJF-like fashion. We run Web Search workload (Figure 12) at 80% load and compare Karuna with DCTCP, a fair-sharing scheme. Figure 7 shows the FCT results across different sizes. We observe the trend that, for small to medium sized flows Karuna is better than fair-sharing, and for larger sized flows Karuna is worse than fair-sharing. This indicates that Karuna emulates SJF on type 3 flows even though their sizes are not known beforehand.

8.1.3 *Deep dive*

Effect of flow termination: Early termination of type 1 flows based on its residual rate gives other flows more bandwidth to achieve their deadlines. We evaluate 3 schemes of flow termination in Figure 8: 1) termination based on $Z(t)$ (§4.2.3), 2) termination based on expected rate (*i.e.* when $\gamma(t)>C$, similar to [39]), and 3) no termination. We observe that Scheme 1 has overall better performance: it terminates more flows than Scheme 2, but has fewer deadline misses

[8]Approximated by giving flows pre-determined priorities.

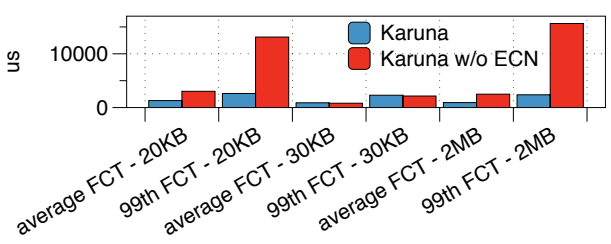

Figure 9: Effect of ECN.

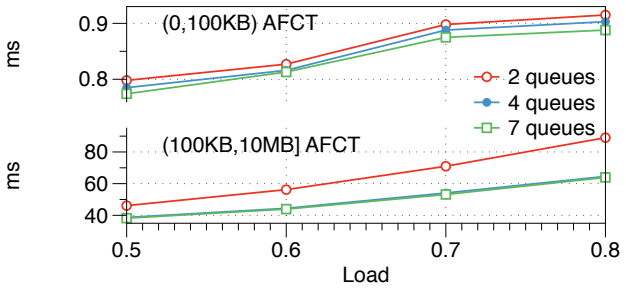

Figure 10: Effect of queue numbers.

(terminated flows count as miss). This shows that Scheme 2 is too lenient in termination, and some flows still send when they cannot meet their deadline, wasting bandwidth.

Effect of ECN: To evaluate the effect of ECN in handling the threshold-traffic mismatch, we create a contrived workload where 80% of flows are 30KB and 20% are 10MB and conduct the experiment at 80% load. We assume all the flows are type 3 flows and allocate 2 priority queues. Obviously, the optimal sieving threshold should be 30KB. We intentionally run experiments with three thresholds 20KB, 30KB and 2MB. In the first case, the short flow sieves to the low priority too early, while in the third case, the long flows over-stay in the high priority queue. In both cases, packets of short flows may experience large delay due to the queue built up by long flows. Figure 9 shows the FCT of 30KB flows with and without ECN. When the threshold is 30KB, both schemes achieve ideal FCT. Karuna w/o ECN even achieves 9% lower FCT due to the spurious marking of per-port ECN. However, with a larger threshold (2MB) or a smaller threshold (20KB), Karuna achieves 57%∼85% lower FCT compared to Karuna w/o ECN at both average and 99th percentile. With ECN, we can effectively control the queue build-up, thus mitigating the effect of threshold-traffic mismatch.

Effect of number of queues: In Figure 10, we inspect the impact of queue number on FCT of type 2&3 flows. For this experiment, we use traffic generated from Web Search workload and consider 2, 4 and 7 priority queues (the first queue is reserved for type 1 flows). We observe that: 1) more queues leads to better average FCT in general. This is expected because, with more queues, Karuna can better segregate type 2&3 flows into different queues, thus improving overall performance; 2) the average FCT of short flows are

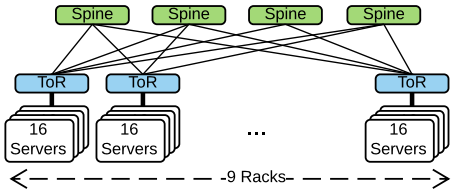

Figure 11: Spine-leaf topology in simulation.

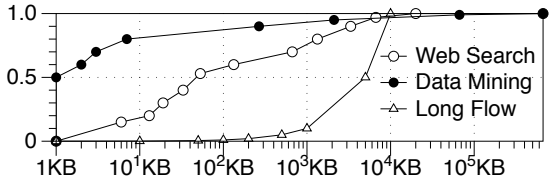

Figure 12: Workloads in simulation.

comparable in all three cases. This indicates that with only 2 queues, the short flows benefit most from Karuna.

8.2 Large-scale simulations

Our simulations evaluate Karuna using realistic DCN workloads on a common DCN topology. We test the limits of Karuna in deadline completion, starvation, traffic variation, and bottlenecked scenarios.

Topology: We perform large scale packet-level simulations with ns-3 [33] simulator, and use fnss [35] to generate different scenarios. We use a 144-server spine-and-leaf fabric (Figure 11), a common topology for production DCNs [4] with 4 core switches, 9 ToRs, and 16 servers per ToR. It is a multi-hop, multiple bottleneck setting, which complements our testbed evaluations. We use 10G link for server to ToR links, and 40G for ToR uplinks.

Traffic workloads: We use two widely-used [3, 6, 20, 30] realistic DCN traffic workloads: a web search workload [3] and a data mining workload [20]. In these workloads, more than half of the flows are less than 100KB in size, which reflects the nature of DCN traffic in practice. However, in some parts of the network, the traffic may be biased towards large sizes. For a more comprehensive study, we also create the "Long Flow" workload to cover this case. In this workload, the size is uniformly distributed from 1KB to 10MB, which means that half of the flows are larger than 5MB. The CDFs of flow sizes from the 3 workloads are shown in Figure 12. Unless specified, each flow type (§2.1) amounts to $1/3$ of overall traffic. As in [4, 6, 30], flow arrival follows a Poisson process and the source and destination for each flow is chosen uniformly at random. We vary flow arrival rate (λ_{arr}) to obtain a desired load (ρ=$\lambda_{arr}\cdot E(F)$, where $E(F)$ is the average flow size for flow size distribution F).

We compare Karuna with DCTCP, D^{2}TCP, D^3, and pFabric. To compare with DCTCP, we follow the parameter setting in [3], and set the switch ECN marking threshold as 65 packets for 10Gbps links and 250 packets for 40Gbps links. We implemented D^{2}TCP and D^3 in ns-3, including the packet format and switch operations in [39]. Following [38], $0.5\leq d\leq 2$ for D^{2}TCP, and the base rate for D^3 is

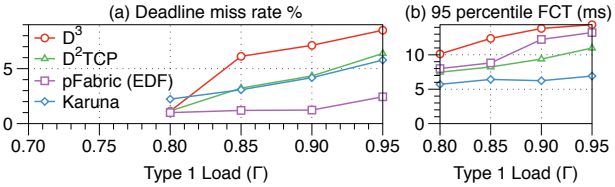

Figure 13: Karuna vs other schemes.

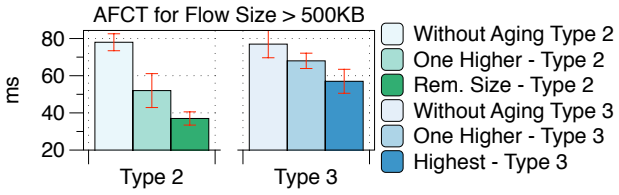

Figure 14: Aging against starvation in Karuna.

Scenario Index	WS (80%)	DM (80%)	LF (80%)
Set 1: thresholds for WS 60% load	1	5	9
Set 2: thresholds for WS 80% load	***2***	6	10
Set 3: thresholds for DM 60% load	3	7	11
Set 4: thresholds for DM 80% load	4	***8***	12

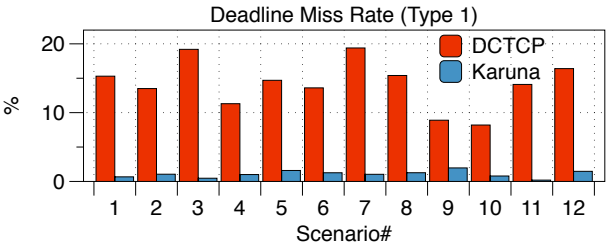

Figure 15: Deadline miss rates in different scenarios.

one segment per RTT. For pFabric, we follow the default parameter setting in [30], and it runs EDF scheduling as in §2.2. Each simulation runs for 60s (virtual time).

8.2.1 *Key strength of Karuna*

Karuna reduces FCT for non-deadline flows without sacrificing much for deadline flows. To show this, we compare Karuna with deadline-aware schemes, D^3, D^2TCP, pFabric (EDF). In this simulation, we choose flow sizes from data mining workload, and source-destination pairs are randomly chosen. We control the load of type 1 flows (total expected rate Γ) by assigning deadlines as follows: we record the total expected rates of all active type 1 flows $\tilde{\Gamma}$, and for each new flow, if $\tilde{\Gamma}<\Gamma$, we tag this flow as type 1 and assign a deadline to achieve Γ as much as possible (minimum deadline is 5ms); otherwise we tag it as type 2 or 3 uniformly at random. We vary Γ from 80% to 95%. The total network load is fixed at 95%.

In Figure 13(a), we observe that Karuna maintains low ($<5.8\%$) deadline miss rate, comparable to D^2TCP (only losing to optimal pFabric (EDF)), while in (b) Karuna achieves 51.92%, 37.07%, and 47.78% less the 95th percentile FCT than D^3, D^2TCP, and pFabric respectively, at maximum load. In fact, Karuna completes $\sim 100\times$ more non-deadline flows (15123 compared to 1338 for pFabric, 834 for D^2TCP when Γ=95%) in the simulation. We note that MCP is not an optimal deadline scheduling discipline like EDF, and is bound to miss some deadlines which only EDF can satisfy. However, the value of Karuna is that it allows non-deadline flows to reduce their FCT in presence of deadline flows, while achieving comparable deadline completion to previous deadline-aware schemes. Therefore, Karuna can provide substantial benefits to applications in §2.1.

8.2.2 *Aging mechanism against starvation*

In this simulation, we use the Long Flow (LF) traffic at 80% load, because we observe very few cases of starvation in the other two realistic workloads. We compare the following schemes for aging as shown in Figure 14: for type 2 flows: 1) "One Higher", which elevates the flow one priority higher (the common approach in OS [37]); 2) "Rem. Size", which elevates the flow to the priority corresponding to its remaining size (Karuna's approach); for type 3 flows: 3) "One Higher" is the same as 2); 4) "Highest", which elevates the flow to the highest priority (*i.e.* Queue 2; Karuna's approach).

In Figure 14, we find that, with aging, the FCT of large flows is much smaller than that without aging for both type 2&3 flows. We also observe that scheme 2 achieves better performance than scheme 1, and scheme 4 achieves better performance than scheme 3. This is because, in this multi-priority queueing system, moving upward for one priority does not always stop starvation. When starvation occurs, the starved flow may be blocked by flows that are a few priorities above, so the flow may still starve with just one priority up. In summary, aging effectively handles starvation in Karuna, and therefore improves FCT for long flows.

8.2.3 *Resilience to traffic variation*

We study Karuna's sensitivity to threshold settings, which include the splitting thresholds $\{\beta\}$, and the sieving thresholds $\{\alpha\}$. Specifically, we calculate 4 sets of $[\{\alpha\},\{\beta\}]$ thresholds: Set 1 and Set 2 are the thresholds calculated for the web search (WS) workload at 60% and 80% load; and Set 3 and Set 4 are the thresholds calculated for the data mining (DM) workload at 60% and 80% load, respectively. For these 4 sets of thresholds, we pair them with different workloads (all at 80% load) to create the 12 scenarios shown in Figure 15 (table at the top). Among these, except scenarios #2 and #8, all the other 10 scenarios create threshold-traffic mismatch. Each type contributes $1/3$ of the overall traffic.

First, we check deadline completion for type 1 flows for all scenarios in Figure 15. Karuna achieves close-to-zero deadline miss rates for type 1 flows in all the scenarios. This is because type 1 flows reside in the highest priority queue, thus can be protected from traffic variations.

Second, we examine the FCT for type 2&3 flows. Figure 16 shows the average FCT of type 2 flows. For WS, the thresholds match the traffic only in scenario #2, and this scenario has the lowest FCT. We also find that scenario #1 has comparable FCT to scenario #2, while scenarios #3 and #4

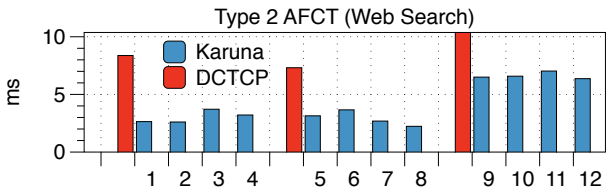

Figure 16: AFCT performance for type 2 flows (The same trend applies to type 3 flows).

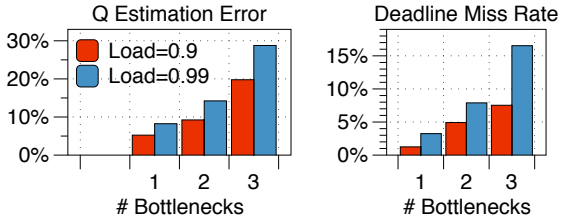

Figure 17: Karuna in bottlenecked environment.

have worse FCT, but not significant. For DM, the matched case is scenario #8, which also has the lowest FCT, whereas the FCTs for other scenarios are relatively worse. For LF, the thresholds are mismatched in all the scenarios, and the FCTs are longer compared to the first two groups. In all cases, Karuna achieves better FCT than DCTCP. The similar trend applies to type 3 flows as well (omitted for space).

In summary, for type 2&3 flows, Karuna performs the best when the thresholds match the traffic, which demonstrates the utility of the optimizations in Appendix A. When the thresholds do not match the traffic, the FCT degrades only slightly (but still much better than DCTCP), which shows that Karuna is resilient to traffic variation, partially because it has employed the ECN-based rate control to mitigate the mismatch (as validated in §8.1.3).

8.2.4 *Karuna in bottlenecked environments*

All the above simulations assume a full bisection bandwidth network, which fits the *one switch* assumption in estimating network term in Eq.(11). To evaluate network term estimation, we intentionally create high loads for cross-rack deadline flows on 1 (destination ToR), 2 (source & destination ToRs), and 3 (source & destination ToRs, and core) intermediate links. We obtain ground-truth queue length and the estimated queue length in MCP in the simulator.

In Figure 17, for different loads on the bottleneck links, we show the average queue estimation error ($100\% \times |\frac{\hat{Q}-Q}{Q}|$) and average deadline miss rates. We observe that the queue estimation error increases when the setting deviates more from our assumptions in (§4.2.1)—both load and number of bottlenecks negatively affect the estimation accuracy. However, Karuna still manages to achieve $<7.9\%$ miss rate for 2 bottlenecks at 99% load. This is because inaccurate estimation leads to accumulation of residual rates, and when the deadline is near, the source term (Eq.(11)) drives up sending rate for the flow to finish.

9. RELATED WORKS

There has been vast literature space on transport design. Here we review works that are closely related to Karuna.

DCTCP [3] is a transport protocol designed for DCN. We employ DCTCP in handling type 2&3 flows since its congestion control scheme works well with ECN. Compared to Karuna, DCTCP is deadline-unaware and unable to simulate SJF because DCTCP flows share bandwidth.

For type 1 flows, D^{2}TCP [38] adds deadline-awareness to DCTCP, but it does not address type 2&3 flows. D^3 [39] deals with the deadline flows using a greedy approach, which leads to priority-inversion problem (§4) and requires heavy modifications to switches. A flexible transport framework, FCP [21], also implements D^3 with a pricing mechanism. In contrast, Karuna ensures the completion of most deadline flows, and also optimizes FCT for other types of flows.

PDQ [22] and pFabric [4] are both criticality-based flow scheduling schemes, and they may hurt other types of flows (see §2.2). In contrast, Karuna not only maintains high deadline meet rate of type 1 flows, it also leaves as much bandwidth as possible for other flows, achieving lower FCT for type 2&3 flows.

PASE [30] combines previous transport layer strategies to reduce average FCT, but does not directly address the mix-flow scheduling problem. Also, PASE requires coordinated rate arbitration in the network control plane, whereas Karuna requires only ECN support in the network.

PIAS [5] is an information agnostic flow scheduling scheme that simulates SJF without knowing the flow sizes. PIAS is effective for type 3 flows, but does not account for the other types. Every flow in PIAS is treated as a type 3 flow, which hurts the performance of the other 2 types. The sieving operation in Karuna is inspired by PIAS, but Karuna adds support for type 1&2 flows.

It is worthwhile to note that MCP-like behaviors (just-in-time strategy and smoothening out link usage) has been explored in areas other than flow scheduling: e.g. traffic engineering [26] and guaranteeing job latencies [18].

For application developers, Karuna is flexible in terms of information needed for scheduling. Most of the above protocols require developers to provide full information about deadlines and sizes [4, 21, 22, 30, 38, 39]; on the other hand, some cannot benefit from flow information [3, 6], even if developers can provide them. In contrast, Karuna can take advantage of any available information given by developers to achieve performance benefits for all types of traffic.

10. CONCLUDING REMARKS

In this paper, we focused on how to schedule a mix of flows in DCNs. This is an important and practical problem, but has been neglected by prior work in this field. Karuna resolves the tension between different types of flows with a joint design of rate-control (MCP) and priority-based flow scheduling with limited priorities in commodity switches. Karuna is not designed to be an optimal flow scheduling algorithm, but a mix-flow scheduling system that balances the interests of deadline and non-deadline flows. At a high

level, Karuna trades off the average performance of one type of traffic (type 1 flows)to improve the average and tail performance of other traffic (type 2&3 flows).

Future work: We intend to explore different formulations of the mix-flow problem with the goal of improving average FCT for all types of flows, subject to deadline constraints for type 1 flows. This formulation is more suitable if deadlines represent the worst case requirements (*e.g.* Service Level Agreement), not the expected performance as we have assumed in the paper. For the current formulation, we plan to improve the queue length estimation using models with less assumptions (*e.g.* M/G/1). In addition, we intend to verify the safety of the relaxations and approximations with perturbation analysis.

Acknowledgments

This work is supported in part by the Hong Kong RGC ECS-26200014, GRF-16203715, GRF-613113, CRF-C703615G, and the China 973 Program No.2014CB340303. We thank our shepherd, Nandita Dukkipati, and the anonymous SIGCOMM reviewers for their valuable feedback. We also thank Haitao Wu for insightful discussions on DCN transport.

11. REFERENCES

[1] http://www.pica8.com/documents/pica8-datasheet-picos.pdf.

[2] DCTCP Patch. http://simula.stanford.edu/~alizade/Site/DCTCP.html.

[3] ALIZADEH, M., GREENBERG, A., MALTZ, D. A., PADHYE, J., PATEL, P., PRABHAKAR, B., SENGUPTA, S., AND SRIDHARAN, M. Data center tcp (dctcp). In *ACM SIGCOMM '10*.

[4] ALIZADEH, M., YANG, S., KATTI, S., MCKEOWN, N., PRABHAKAR, B., AND SHENKER, S. pfabric: Minimal near-optimal datacenter transport. In *ACM SIGCOMM '13*.

[5] BAI, W., CHEN, L., CHEN, K., HAN, D., TIAN, C., AND SUN, W. Pias: Practical information-agnostic flow scheduling for datacenter networks. In *HotNet 2014*.

[6] BAI, W., CHEN, L., CHEN, K., HAN, D., TIAN, C., AND WANG, H. Information-agnostic flow scheduling for commodity data centers. In *NSDI 2015*.

[7] BAI, W., CHEN, L., CHEN, K., AND WU, H. Enabling ecn in multi-service multi-queue data centers. In *NSDI 16*.

[8] BOYD, S., AND VANDENBERGHE, L. *Convex Optimization*. Cambridge University Press, New York, NY, USA, 2004.

[9] CHEN, B. B., AND PRIMET, P. V.-B. Scheduling deadline-constrained bulk data transfers to minimize network congestion. In *IEEE CCGRID 2007*.

[10] CHEN, L., HU, S., CHEN, K., WU, H., AND ALIZADEH, M. MCP: Towards minimal-delay deadline-guaranteed transport protocol for data center networks (technical report). "http://goo.gl/ncZKGT".

[11] CHEN, L., HU, S., CHEN, K., WU, H., AND TSANG, D. H. K. Towards minimal-delay deadline-driven data center tcp. In *HotNets-XII* (2013).

[12] CHEN, Y., GRIFFITH, R., LIU, J., KATZ, R. H., AND JOSEPH, A. D. Understanding tcp incast throughput collapse in datacenter networks. In *Proceedings of the 1st ACM WREN*.

[13] CHOWDHURY, M., AND STOICA, I. Efficient coflow scheduling without prior knowledge. In *ACM SIGCOMM '15*.

[14] CHOWDHURY, M., ZHONG, Y., AND STOICA, I. Efficient coflow scheduling with varys. In *ACM SIGCOMM '14*.

[15] COFFMAN, E. G., AND DENNING, P. J. *Operating systems theory*, vol. 973. Prentice-Hall Englewood Cliffs, NJ, 1973.

[16] CONWAY, R. W., MAXWELL, W. L., AND MILLER, L. W. *Theory of scheduling*. Courier Corporation, 2012.

[17] DOGAR, F., KARAGIANNIS, T., BALLANI, H., AND ROWSTRON, A. Decentralized task-aware scheduling for data center networks. In *ACM SIGCOMM '14*.

[18] FERGUSON, A. D., BODIK, P., KANDULA, S., BOUTIN, E., AND FONSECA, R. Jockey: Guaranteed job latency in data parallel clusters. In *EuroSys '12*.

[19] GRANT, M., BOYD, S., AND YE, Y. Cvx: Matlab software for disciplined convex programming, 2008.

[20] GREENBERG, A., HAMILTON, J. R., JAIN, N., KANDULA, S., KIM, C., LAHIRI, P., MALTZ, D. A., PATEL, P., AND SENGUPTA, S. VL2: A scalable and flexible data center network. In *ACM SIGCOMM '09*.

[21] HAN, D., GRANDL, R., AKELLA, A., AND SESHAN, S. Fcp: a flexible transport framework for accommodating diversity. In *ACM SIGCOMM CCR (2013)*.

[22] HONG, C.-Y., CAESAR, M., AND GODFREY, P. B. Finishing flows quickly with preemptive scheduling. In *ACM SIGCOMM '12*.

[23] HOU, X.-P., SHEN, P.-P., AND WANG, C.-F. Global minimization for generalized polynomial fractional program. *Mathematical Problems in Engineering 2014*.

[24] JEYAKUMAR, V., ALIZADEH, M., MAZIERES, D., PRABHAKAR, B., KIM, C., AND GREENBERG, A. Eyeq: Practical network performance isolation at the edge. In *NSDI '13*.

[25] JIAO, H., WANG, Z., AND CHEN, Y. Global optimization algorithm for sum of generalized polynomial ratios problem. *Applied Mathematical Modelling, 2013*.

[26] KANDULA, S., MENACHE, I., SCHWARTZ, R., AND BABBULA, S. R. Calendaring for wide area networks. In *ACM SIGCOMM '14*.

[27] KLEINROCK, L. *Theory, volume 1, Queueing systems*. Wiley-interscience, 1975.

[28] KLEINROCK, L. *Queueing systems: volume 2: computer applications*, vol. 82. John Wiley & Sons New York, 1976.

[29] LIU, C. L., AND LAYLAND, J. W. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM (JACM), 1973*.

[30] MUNIR, A., BAIG, G., IRTEZA, S., QAZI, I., LIU, I., AND DOGAR, F. Friends, not foes: synthesizing existing transport strategies for data center networks. In *ACM SIGCOMM '14*.

[31] NEELY, M. J. *Dynamic power allocation and routing for satellite and wireless networks with time varying channels*. PhD thesis, Massachusetts Institute of Technology, 2003.

[32] NEELY, M. J., MODIANO, E., AND ROHRS, C. E. Dynamic power allocation and routing for time-varying wireless networks. *IEEE JSAC, (2005)*.

[33] RILEY, G. F., AND HENDERSON, T. R. The ns-3 network simulator modeling and tools for network simulation. In *Modeling and Tools for Network Simulation, 2010*.

[34] ROY, A., ZENG, H., BAGGA, J., PORTER, G., AND SNOEREN, A. C. Inside the social network's (datacenter) network. In *ACM SIGCOMM '15*.

[35] SAINO, L., COCORA, C., AND PAVLOU, G. A toolchain for simplifying network simulation setup. In *SIMUTOOLS '13*.

[36] SHEN, P., CHEN, Y., AND MA, Y. Solving sum of quadratic ratios fractional programs via monotonic function. *Applied Mathematics and Computation, 2009*.

[37] SILBERSCHATZ, A., GALVIN, P. B., GAGNE, G., AND SILBERSCHATZ, A. *Operating system concepts*. 1998.

[38] VAMANAN, B., HASAN, J., AND VIJAYKUMAR, T. Deadline-aware datacenter tcp (d2tcp). In *ACM SIGCOMM '12*.

[39] WILSON, C., BALLANI, H., KARAGIANNIS, T., AND ROWTRON, A. Better never than late: meeting deadlines in datacenter networks. In *ACM SIGCOMM '11*.

[40] WU, H., FENG, Z., GUO, C., AND ZHANG, Y. Ictcp: Incast congestion control for tcp in data center networks. In *Co-NEXT '10*.

Appendix

A. OPTIMAL THRESHOLDS

We describe our formulation to derive optimal thresholds for splitter and sieve to minimize the average FCT for type 2&3 flows.

Problem formulation: We take the flow size cumulative density functions of different types as given. Denote $F_1(\cdot)$, $F_2(\cdot)$, and $F_1(\cdot)$ as the respective traffic distributions of the three types, and $F(\cdot)$ as the overall distribution. Thus, $F(\cdot)=\sum_{i=1}^{3}F_i(\cdot)$.

As in §5, type 2 flows are split into different priorities based on their sizes with $\{\beta\}$ as splitting thresholds, and type 3 flows are sieved in a multi-level feedback queue with $\{\alpha\}$ as sieving thresholds. We assume flows arrival follows a Poisson process, and denote the load of network as ρ, $0\leq\rho\leq1$. For a type 2 flow with priority j, the expected FCT is upper-bounded by [28]:

$$T_j^{(2)}=\frac{\rho(F_2(\beta_j)-F_2(\beta_{j-1}))}{1-\rho(F_1(\alpha_K)+F_2(\beta_{j-1})+F_3(\alpha_{j-1}))}$$

For a type 3 flow with size in $[\alpha_{j-1},\alpha_j)$, it experiences the delays in different priorities upto the j-th priority. An upper-bound is identified as [5]: $\sum_{l=1}^{j}T_l^{(3)}$, where $T_l^{(3)}$ is the average time of a type 3 flow spent in the j-th queue. Thus:

$$T_l^{(3)}=\frac{\rho(F_3(\alpha_l)-F_3(\alpha_{l-1}))}{1-\rho(F_1(\alpha_K)+F_2(\beta_{l-1})+F_3(\alpha_{l-1}))}$$

We identifies the problem as choosing an optimal set of thresholds $\{\alpha,\beta\}$ to minimize the objective: the average FCT of type 2&3 flows in the network:

$$\begin{aligned}\min_{\{\alpha\},\{\beta\}}\quad & \sum_{l=1}^{K}T_l^{(2)}+\sum_{l=1}^{K}(F_3(\alpha_l)-F_3(\alpha_{l-1}))\sum_{m=1}^{l}T_m^{(3)})\\ \text{subject to}\quad & \alpha_0=0,\alpha_K=\infty,\alpha_{j-1}<\alpha_j,j=1,...,K\\ & \beta_0=0,\beta_K=\infty,\beta_{j-1}<\beta_j,j=1,...,K\end{aligned}$$

To simplify the notations, we define $\phi_j=F_2(\beta_j)-F_2(\beta_{j-1})$ and $\theta_j=F_3(\alpha_j)-F_3(\alpha_{j-1})$. Thus, ϕ_j denotes the percentage of type 2 flows with sizes in $[\beta_{j-1},\beta_j)$, and θ_j denotes the percentage of type 3 flows with sizes in $[\alpha_{j-1},\alpha_j)$. And the objective can be transformed equivalently to:

$$\min_{\{\phi\},\{\theta\}}\ \rho\sum_{l=1}^{K}\frac{\phi_l+\theta_l\sum_{i=1}^{l-1}\theta_i}{1-\rho\hat{F}_1-\rho\sum_{i=1}^{l-1}(\theta_i+\phi_i)}$$

where $\hat{F}_1=F_1(\infty)$, the fraction of type 1 flows.

Solution method: We identify this as a quadratic sum-of-ratios problem (due to the term $\theta_l\sum_{i=1}^{l-1}\theta_i$), which has been thoroughly investigated by applied mathematical modeling and optimizations communities [23, 25, 36]. We use relaxation technique, and solve for the lower-bound of the objective. Notice that $1-\hat{F}_1-\rho\sum_{i=1}^{l-1}(\theta_i+\phi_i)$ is strictly smaller than 1, thus $\rho\sum_{l=1}^{K}\phi_l+\theta_l\sum_{i=1}^{l-1}\theta_i$ must be the lower bound of the objective for all possible choices of the thresholds. Therefore, we look for a set of percentages $\{\phi\}$ and $\{\theta\}$ that minimize the lower-bound. The problem then becomes:

$$\begin{aligned}\min_{\{\theta\},\{\phi\}}\quad & \sum_{l=1}^{K}(\phi_l+\theta_l\sum_{i=1}^{l-1}\theta_i)\\ \text{subject to}\quad & \sum_{i=1}^{K}\phi_i=F_2(\infty),\sum_{i=1}^{K}\theta_i=F_3(\infty)\end{aligned}$$

Now the problem is relaxed into a quadratic programming problem with linear constraints, which can be solved using semidefinite programming packages available in many solvers. We use the CVX toolbox [19] for MATLAB to solve the above problem. Since the complexity of the problem is related to the number of queues in the switches, the scale of the network is irrelevant, and we can solve it in under 10 seconds on a testbed machine.