***ATOM: Automatic Traffic Optimization using Machine learning***

*An end-end automatic traffic optimization tool for data centers*

Indukuri. Surya Sai Harischandra Prasad

Department of Electronics and Communication Engineering National Institute of Technology

Warangal, India. [suryasai678@gmail.com](mailto:suryasai678@gmail.com)

Ramavath Balu Rathod

Department of Electronics and Communication Engineering National Institute of Technology

Warangal, India. [ramavathbalu123@gmail.com](mailto:ramavathbalu123@gmail.com)

# ABSTRACT

Traffic optimizations (TO, e.g. flow scheduling, load balancing) in data centers are difficult online decision- making problems. Previously, they are done with heur- istics relying on operators’ understanding of the work- load and environment. Designing and implementing pr- oper TO algorithms thus take at least weeks. Encourag- ed by recent successes in applying Reinforcement Lea- rning (RL) techniques to solve complex online control problems, we study if RL can be used for automatic TO without human-intervention.

Here, we used a two-level RL system, ATOM, mimic- king the Peripheral & Central Nervous Systems in ani- mals. Peripheral System(PS) reside on the end hosts, c- ollect flow information, and make TO decisions locally with minimal delay for short flows. PS’s decisions are informed by a Central System(CS), where global traff-ic information is aggregated and processed. CS further makes individual TO decisions for long flows. With CS&PS, ATOM is an end-to-end automatic TO system that can collect network information, learn from past d- ecisions, and perform actions to achieve operator defi- ned goals. We implement ATOM with popular machi-ne learning frameworks and can be deployed on a 32-server testbed. ATOM reduces TO turn-around time fr-om weeks to ~100 milliseconds while achieving super-ior performance. Based on the theoretical Calculations, It demonstrates up to 58.14% reduction in average flow completion time (FCT) over existing solutions.

# CCS CONCEPTS

## Networks → Network resources allocation; Traffic engineering algorithms; Data center networks;

* **Computing methodologies → Reinforcement learning;**

**KEYWORDS**

**Data center Networks, Reinforcement Learning, Traffic Optimization**

1. **INTRODUCTION**

Data center traffic optimizations (TO, e.g. flow/coflow scheduling [1, 4, 5, 8, 9, 10, 12, 20], congestion con-trol

[3, 6], load balancing & routing [2]) have signifi-cant impact on application performance. Currently, TO is de- pendent on hand- crafted heuristics for varying traffic lo- ad, flow size distribution, traffic concentration, etc. Wh- en parameter setting mismatches traffic, TO heuristics may suffer performance penalty. For example, in PIAS [5], thresholds are calculated based on a long term flow size distribution, and is prone to mismatch the current or true size distribution in run-time. Under mismatch scen- arios, performance degradation can be as much as 38.46% [5]. pFabric [4] shares the same problem when implemented with limited switch queues: for certain cases the average FCT can be reduced by over 30% even if the thresholds are carefully optimized. Furthermore, in coflow scheduling, fixed thresholds in Aalo [9] depend on the operator’s ability to choose good values upfront, since there is no run-time adap-tation.

Apart from parameter-environment mismatches, the turn- around time of designing TO heuristics is long—at least weeks. Because they require operator insight, application knowledge, and traffic statistics collected over a long period of time. A typical process includes: first, deploying a monitoring system to collect end-host and/or switch statistics; second, after collecting enough

data, operators analyze the data, design heuristics, and test it using simulation tools and optimization tools to find suitable parameter settings; finally, tested heuri- stics are enforced1 (with application modifications [10, 20], OS kernel module [5, 8], switch configurations [6],

or any combinations of the above).

Automating the TO process is thus appealing, and we desire an automated TO agent that can adapt to volum- inous, uncertain, and volatile data center traffic, while achieving operator-defined goals. In this paper, we in- vestigate reinforcement learning (RL) techniques [18], as RL is the subfield of machine learning concerned with decision making and action control. It studies how an agent can learn to achieve goals in a complex, unce-rtain environment. An RL agent observes previous en- vironment states and rewards, then decides an action in order to maximize the reward. RL has achieved good results in many difficult environments in recent years DeepMind’s Atari results [16] and AlphaGo [17] used RL (RL) algorithms which make few assumptions about their environments, and thus can be generalized in other settings. Inspired by these results, we are motivated to enable RL for automatic data center TO.

We are trying to implement a flow-level centralized TO system with a basic RL algorithm called as Thom-pson Sampling. The key to ATOM’s scalability is to detach time-consuming RL processing from quick action taking for short flows. To achieve this, we adopt Multi-Level Feedback Queueing (MLFQ) [5] for PS to schedule flows guided by a set of thresholds.

# BACKGROUND AND MOTIVATION

In this section, we first overview the RL background. Then, we describe and apply a basic RL algorithm, Thompson Sampling to enable flow scheduling in TO.

## Thompson Sampling

It is one of the best RL algorithm available in Machine Learning so far. Its working can be explained as:

* + - Let a dataflow gets rewards from Bernoulli distribution p(**y**/θi) ≈ β(θi).
    - θi is unknown but we set its uncertainty by assuming it has a uniform distribution p(θi) ≈ U([0,1]), which is the prior distribution.
    - Bayes Rule: we approach θi by the posterior
* We get p(θi/**y**) ≈ *β*(number of successes+1, number of failures+1)
* At each round n, we take a random draw θi(n) from the posterior distribution p(θi/**y**), for each network *i.*
* At each round *n*, we select the ad *i* that has the highest *θi(n).*

## RL for Flow scheduling

As an example, we formulate the problem of flow scheduling in data centers as a RL problem, and describe a solution using the Thompson Sampling algorithm based on Equation.

**Flow scheduling problem** We consider a data center network connecting multiple servers. For simplicity, we adopt the big-switch assumption by previous works in flow scheduling [4, 8], where the network is non- blocking with full- bisection bandwidth and proper load- balancing. Following this assumption, the flow scheduling problem is simplified to the problem of deciding the sending order of flows. We consider an implementation that enables pre-emptive scheduling of flows using strict priority queueing. We create K priority queues for flows in each server [11], and enforce strict priority queuing among them. K priority queues are also configured in the switches, similar to [5]. The priority of each flow can be changed dynamically to enable pre- emption. The packet of each flow is tagged with its current priority number, and will be placed in the same queue throughout the entire data center fabric.

## RL formulation

*Action Space:* The action provided by the agent is a mapping from active flows to priorities: for each active flow f, at time step t, its priority is pt (f )∈[1,K].

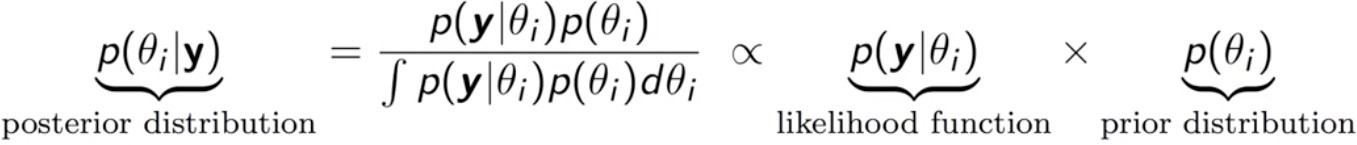
*State space:* The big-switch assumption allows for a much simplified state space. As routing and load ba- lancing are out of our concern, the state space only in- cludes the flow states. In our model, states are represe-

nted as the set of all active flows, Fat , and the set of all

distribution

finished flows, F

dt , in the entire network at current time

step t. Each flow is identified by its 5-tuple [5, 15]:

source/destination IP, source/destination port numbers, and transport protocol. Active flows have an additional attribute, which is its priority; while finished flows have two additional attributes: FCT and flow size.

## ATOM DESIGN

* 1. **Overview**

The key problem of current RL systems is the long latency between collection of flow information and generation of actions. In modern data centers with ≥ 10Gbps link speed, to achieve flow-level TO operat- ions, the round-trip latency of actions should be at least sub-millisecond. Without introducing specialized hard- ware, this is unachievable (§2.2). Using commodity har- dware, the processing latency of RL algorithm is a hard limit. Given this constraint, how to scale RL for data center TO?

Recent studies [3, 7, 13] have shown that most data center flows are short flows, yet most traffic bytes are from long flows. Informed by such long-tail distribution, our insight is to delegate most short flow operations to the end-host, and formulate RL algorithms to generate long-term (sub-second) TO decisions for long flows.

We design ATOM as a two-level system, mimicking the Peripheral and Central Nervous Systems in animals. As shown in Figure 3, Peripheral Systems (PS) run on all end-hosts, collect flow information, and make TO decisions locally with minimal delay for short flows.

Central System (CS) makes individual TO decisions for long flows that can tolerate longer processing delays.

Furthermore, PS’s decisions are informed by the CS where global traffic information are aggregated and processed.

# Peripheral System

The key to ATOM’s scalability is to enable PS to make globally informed TO decisions on short flows with only local information. PS has two modules: an enforcement module and a monitoring module

**Enforcement module** to achieve the above goal, we adopt Multi-Level Feedback Queueing (MLFQ, introduced in PIAS [5]) to schedule flows without centralized per-flow control. Specifically, PS performs packet tagging in the DSCP field of IP packets at each

end-host as shown in Figure 4. There are K priorities, Pi,1≤i≤K, and (K−1) demotion thresholds, αj,1≤ j≤K−1. We configure all the switches to perform strict priority

queueing based on the DSCP field. At the end host, when a new flow is initialized, its packets are tagged with P1, giving them the highest priority in the network.

As more bytes are sent, the packets of this flow will be tagged with decreasing priorities Pj (2≤j≤K),thus they are scheduled with decreasing priorities in the network.

The threshold to demote priority from Pj−1 to Pj is αj−1.

With MLFQ, PS has the following properties:

* It can make instant per-flow decisions based only on local information: bytes-sent and thresholds.
* It can adapt to global traffic variations. To be scalable, CS must not directly control small flows. Instead, CS optimizes and sets MLFQ thresholds with global information over a longer period of time. Thus, thresholds in PS can be updated to adapt to traffic variations. In contrast, PIAS [5] requires weeks of traffic traces to update the thresholds.
* It naturally separates short and long flows. As

shown in Figure 5, short flows finished in the first few queues, and long flows drop to the last queue. Thus, CS can centrally process long flows individually to make decisions on rout- ing, rate limit, and priority.

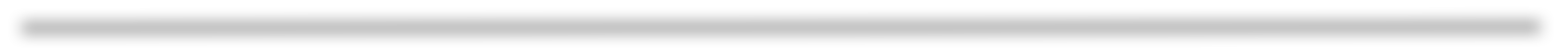
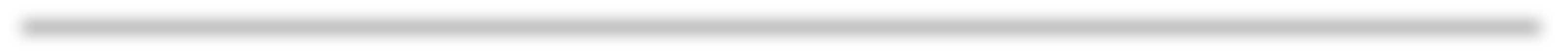
* **Monitoring module** For CS to generate thres- holds, the monitoring module collects flow sizes and completion times of all finished flows, so that CS can update flow size distribution. The monitoring module also reports on-going long flows that have descended into the lowest priority on its end-host, so that CS can make individual decisions.

# Central System

The CS is composed of two RL agents (RLA): short flow RLA (sRLA) is for optimizing thresholds for MLFQ, and long flow RLA (lRLA) is for determining rates, routes, and priorities for long flows. sRLA atte- mpts to solve a FCT minimization problem, and we develop a Random Forest Regression and K-means clustering algorithms for this purpose. For lRLA, we use a Thompson Sampling algorithm (§2.2) to generate actions for the long flows. In the next section, we describe the two RL problems and solutions.

**Algorithm 1:** Thompson Sampling reward system.

1. At each round n, we consider two numbers for each network flow *i*:



* + Ni1(n) – the number of times the network

flow i got reward 1 up to round n.

* + Ni0(n) – the number of times the network flow i got reward 0 up to round n.

1. For each network flow *i*, we take a random draw from the distribution below:

θ (n) = *β*( N 1(n) + 1, N 0(n) + 1)

i i i

# DRL FORMULATIONS AND SOLUTIONS

In this section, we describe the two RL algorithms in CS.

# Optimizing MLFQ thresholds

We consider a data center network connecting multiple servers. Scheduling of flows is imposed by using K strict priority queues at hosts and network switches by setting the DCSP field in each of the IP headers. The longer the flow is, the lower priority is assigned to the flow as it is demoted through host priority queues in order to approximate Shortest-Job-First (SJF). The packet’s priority is preserved throughout the entire data center fabric till it reaches the destination. One of the challenges of MLFQ is the calculation of the optimal demotion thresholds for the K priority queues at the host. Prior works [5, 9, 8] provide mathematical analysis and

models for optimizing the demotion thresholds: {α1 ,α2

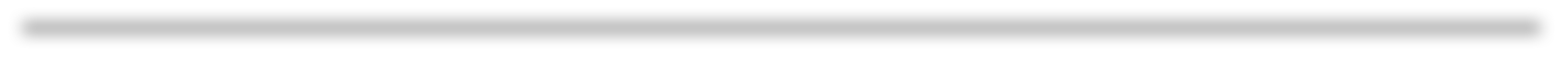
,...,αK-1}. Bai et al. [9] also suggest weekly/monthly re- computation of the thresholds with collected flow-level traces. ATOM takes a step further and proposes a RL ap- proach to optimizing the values of the α’s. Unlike prior works that used machine learning in data center problems [5, 14, 10], ATOM is unique due to its target - optimization of real values in continuous action space.

We formulate the threshold optimization problem as an RL problem and try to explore the capabilities of

Thompson Sampling for modelling the complex data center network for computing the MLFQ thresholds.

1. We select the network that has the highest θi(n).

# Optimizing Long Flows



The last threshold, αK−1, separates long flows from short flows by sRLA, thus αK−1 is updated dynamically according to current traffic characteristics, in contrast to prior works with fixed threshold for short and long flows [1, 23]. For long flows and lRLA, we use a PG algorithm similar to the flow scheduling problem in §2.2, and the only difference is in the action space.

Action Space: For each active flow f , at time step t , its cor- responding action is {Priot (f ),Ratet (f ),Patht (f )}, where Priot (f ) is the flow priority, Ratet (f ) is the rate limit, and Patht (f ) is the path to take for flow f . We assume the paths are enumerated in the same way as in XPath [22].

State space: Same as §2.2, states are represented as the set of all active flows, Fat , and the set of all finished flows, Fdt , in the entire network at current time step t .

Apart from its 5-tuple [5, 15], each active flow has an additional attribute: its priority; each finished flow has two additional attributes: FCT and flow size.

Rewards: The reward is obtained for the set of finished flows Fdt . Choices for the reward function can be:

difference or ratios of sending rate, link utilization, and

throughput in consecutive time steps. For modern data centers with at least 10Gbps link speed, it is not easy to obtain timely flow-level in- formation for active flows. Therefore, we choose to compute reward with finished flows only, and use the ratio between the average throughputs of two consecutive time steps as reward, as

in Equation (3). The reward is capped to achieve quick convergence [21].

# IMPLEMENTATION

We have implemented the code in python and used libraries that are built into the python to make our task easier.

First, we have searched for the dataset for a datacenter and finally got one, which describes the following:

* It is a routing table from one PARTICULAR IP to another IP via data center having 10 routers.
* It has the 10,000 packet routing history of that particular transaction.

Now, We have given a priority for each packet and sorted them based on the priority by using a Priority Queue. We

sent the resulting dataset to our Central System that decides

# whether it is a small data flow or large data flow on it size of packet. Here, we used sorting technique to do so namely merge sort. Sorted it in ascending order and given the preference to shorter data flow.

# The code for this project is in GitHub.

# Link: <https://github.com/surya1304/CN_Project>

# Using this we have achieved an efficiency of 25% for the long flows. And a massive which is great considering the time it takes to get train. And in the short flows, we got 69% as the efficiency.

# A future of turning the machine learning model into the deep learning model is in progress…

# REFERENCES:

1. Mohammad Al-Fares, Sivasankar Radhakrishnan, Barath Raghavan, Nelson Huang, and Amin Vahdat. [n. d.]. Hedera: Dynamic Flow Scheduling for Data Center Networks. In USENIX NSDI ’10.
2. Mohammad Alizadeh, Tom Edsall, Sarang Dharmapurikar, Ramanan Vaidyanathan, Kevin Chu, Andy Fingerhut, Francis Matus, Rong Pan, Navindra Yadav, George Varghese, et al. [n. d.]. CONGA: Distributed congestion-aware load balancing for datacenters. In ACM SIGCOMM ’14.
3. Mohammad Alizadeh, Albert Greenberg, David A Maltz, Jitendra Pad- hye, Parveen Patel, Balaji Prabhakar, Sudipta Sengupta, and Murari Sridharan. [n. d.]. Data Center TCP (dctcp). In ACM SIGCOMM ’10.
4. Mohammad Alizadeh, Shuang Yang, Milad Sharif, Sachin Katti, Nick McKeown, Balaji Prabhakar, and Scott Shenker. [n. d.]. pFabric: Minimal Near-Optimal Datacenter Transport. In ACM SIGCOMM ’13
5. Wei Bai, Li Chen, Kai Chen, Dongsu Han, Chen Tian, and Weicheng Sun. [n. d.]. Information-Agnostic Flow Scheduling for Data Center Networks. In USENIX NSDI ’15.
6. Wei Bai, Li Chen, Kai Chen, and Haitao Wu. [n. d.]. Enabling ECN in Multi-Service Multi-Queue Data Centers. In USENIX NSDI ’16.
7. Theophilus Benson, Aditya Akella, and David Maltz. [n. d.]. Network Traffic Characteristics of Data Centers in the Wild. In ACM IMC’10.

Li Chen, Kai Chen, Wei Bai, and Mohammad Alizadeh. [n. d.]. Sched- uling Mix-flows in Commodity Datacenters with Karuna. In ACM SIGCOMM ’16.

1. Mosharaf Chowdhury and Ion Stoica. [n. d.]. Efficient Coflow Sched- uling Without Prior Knowledge. In ACM SIGCOMM ’15.
2. Mosharaf Chowdhury, Yuan Zhong, and Ion Stoica. [n. d.]. Efficient coflow scheduling with Varys. In ACM SIGCOMM ’14.
3. Linux Foundation. [n. d.]. Priority qdisc - Linux man page. https: //linux.die.net/man/8/tc-prio. ([n. d.]). (Accessed on 04/17/2017).
4. Chi-Yao Hong, Matthew Caesar, and P Godfrey. [n. d.]. Finishing flows quickly with preemptive scheduling. In ACM SIGCOMM ’12.
5. Srikanth Kandula, Sudipta Sengupta, Albert Greenberg, Parveen Pa- tel, and Ronnie Chaiken. [n. d.]. The Nature of Datacenter Traffic: Measurements and Analysis. In ACM IMC’09.
6. Hongzi Mao, Mohammad Alizadeh, Ishai Menache, and Srikanth Kan- dula. [n. d.]. Resource Management with Deep Reinforcement Learning. In ACM HotNets ’16.
7. N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peter- son, J. Rexford, S. Shenker, and J. Turner. 2008. OpenFlow: Enabling innovation in campus networks. ACM CCR (2008).
8. Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioan- nis Antonoglou, Daan Wierstra, and Martin Riedmiller. 2013. Playing atari with deep reinforcement learning. arXiv preprint arXiv:1312.5602 (2013).
9. DavidSilver,AjaHuang,ChrisJMaddison,ArthurGuez, LaurentSifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. 2016. Mastering the game of Go with

deep neural networks and tree search. Nature (2016).

1. Richard S. Sutton and Andrew G. Barto. 1998. Introduction to Reinforcement Learning.
2. Neeraja J. Yadwadkar, Ganesh Ananthanarayanan, and Randy Katz. [n. d.]. Wrangler: Predictable and Faster Jobs Using Fewer Resources. In SOCC ’14.
3. Hong Zhang, Li Chen, Bairen Yi, Kai Chen, Mosharaf Chowdhury, and Yanhui Geng. [n. d.]. CODA: Toward automatically identifying and scheduling coflows in the dark. In ACM SIGCOMM ’16.
4. Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. 2015. Contin- uous control with deep reinforcement learning. CoRR abs/1509.02971 (2015). arXiv:1509.02971 <http://arxiv.org/abs/1509.02971>
5. Shuihai Hu, Kai Chen, Haitao Wu, Wei Bai, Chang Lan, Hao Wang, Hongze Zhao, and Chuanxiong Guo. 2016. Explicit path control in com- modity data centers: Design and applications. IEEE/ACM Transactions on Networking (2016)
6. Nathan Farrington, George Porter, Sivasankar Radhakrishnan, Hamid Hajabdolali Bazzaz, Vikram Subramanya, Yeshaiahu Fainman, George Papen, and Amin Vahdat. [n. d.]. Helios: A Hybrid Electri- cal/Optical Switch Architecture for Modular Data Centers. In ACM SIGCOMM’10.