Preemptive efficient queueing for scheduling deadline-bound data center flows

VinayKrishna GopalaKrishna

New Jersey Institute of Technology

ABSTRACT

PREEMPTIVE EFFICIENT QUEUEING FOR SCHEDULING DEADLINE-BOUND DATA CENTER FLOWS

by
VinayKrishna GopalaKrishna

This thesis introduces a new deadline aware flow scheduling scheme called Preemptive Efficient Queuing (PEQ). Unlike other schemes which use policies like EDF, SJF or fair share, this scheme aims to procrastinate flows as much as they can afford to, while keeping with their deadlines. Thereby, PEQ prioritizes more urgent flows that have shorter time to procrastinate, or cushion time. Results of PEQ are compared with state-of-the-art schemes for the transport of data-center flows, such as Preemptive Distributed Quick (PDQ) and Deadline-Driven Delivery (D^3). The results show that PEQ outperforms D^3 and PDQ. We identify results of an optimal scheme and show that PDQ's performance is close to that, yet not quite the same. Therefore, PEQ is a heuristic scheme with the equivalent complexity of PDQ but its performance is closer to the optimal solution than that of PDQ. The presented results show that when short flows have longer deadlines and long flows have stricter deadlines, the performance of PEQ, measured in terms of application throughput (i.e., the number of flows completed on time), is about twice that of PDQ.

PREEMPTIVE EFFICIENT QUEUEING FOR SCHEDULING DEADLINE-BOUND DATA CENTER FLOWS

by
VinayKrishna GopalaKrishna

A Thesis
Submitted to the Faculty of
New Jersey Institute of Technology
in Partial Fulfillment of the Requirements for the Degree of
Master of Science in Internet Engineering

Department of Electrical and Computer Engineering

May 2017

APPROVAL PAGE

PREEMPTIVE EFFICIENT QUEUEING FOR SCHEDULING DEADLINE-BOUND DATA CENTER FLOWS

VinayKrishna GopalaKrishna

BIOGRAPHICAL SKETCH

Author:	VinayKrishna GopalaKrishna
Degree:	Master of Science
Date:	May 2017

Undergraduate and Graduate Education:

- Master of Science in Internet Engineering,
  New Jersey Institute of Technology, 2017
- Bachelor of Engineering in Electrical and Electronics Engineering,
  Visvesvaraya Technological University, Belgaum, India, 2008

- **Major:** Internet Engineering

To my Parents,

ACKNOWLEDGMENT

I would like to express my deepest appreciation to Professor Roberto Rojas-Cessa, who not only served as my Thesis Advisor, providing valuable and countless resources, insight and intuition, but also constantly gave me support and encouragement. Special thanks to my committee members Dr. Nirwan Ansari and Dr. Qing Gary Liu for actively participating.

Also from our research group, Yagiz Kaymak who deserved recognition for his time, constant support and guidance.

Finally, I would like to express my gratitude to my parents who have supported me in every walk of my life for my success and happiness.

TABLE OF CONTENTS

TABLE OF CONTENTS
(Continued)

LIST OF TABLES

LIST OF FIGURES

LIST OF FIGURES
(Continued)

Figure **Page**

CHAPTER 1

INTRODUCTION

Data centers are warehouses with large number of servers and networking equipment, logically connected. They not only provide data for businesses, but also assure security, safety and scalability for managing large amounts of data. It is very important to ensure the servers in a data center communicate with each other efficiently. Hence, Data Center Networks (DCN) are crucial for attaining these communications [3]. Businesses may keep their data on a third party infrastructure to increase security, reduce management costs, or to overcome the lacking of own cyber infrastructure. These third party infrastructure providers own data centers and are expected to ensure that they provide the required level of performance [14]. Businesses and infrastructure providers have Service Level Agreements (SLA) which stipulates the negotiated quality of service required [11] [19] [33]. Therefore, data center transport protocols must to meet these performance requirements as great portion of the performance depends on it [20]. Data center transport protocols are different from transport protocols in the Internet, as the traffic properties differ from traffic of campus networks.

The end user's quality of experience depends a lot on the performance of the data centers. For instance, when a user requests a new facebook page, the page has to be displayed at least within a couple of seconds. This request is broken down into multiple sub-requests in the data center. In the Partition-Aggregate model (Figure 1.1) used in most data centers, the aggregator assigns these sub-requests to "worker" servers, which are served parallelly. Intra data center traffic is usually characterized in terms of number of flows [29], each of which is a sequence of packets belonging to the same application and end-to-end servers [28]. The aggregator receives flows back from the worker servers. These flows have assigned deadlines to be transferred to the aggregator. Deadlines are generally found in the order of a few milliseconds. Therefore, the protocol used to transfer flows to the aggregator determines the achieved performance. If this transfer protocol is not efficient enough to transfer flows on time, many flows will miss their deadlines.

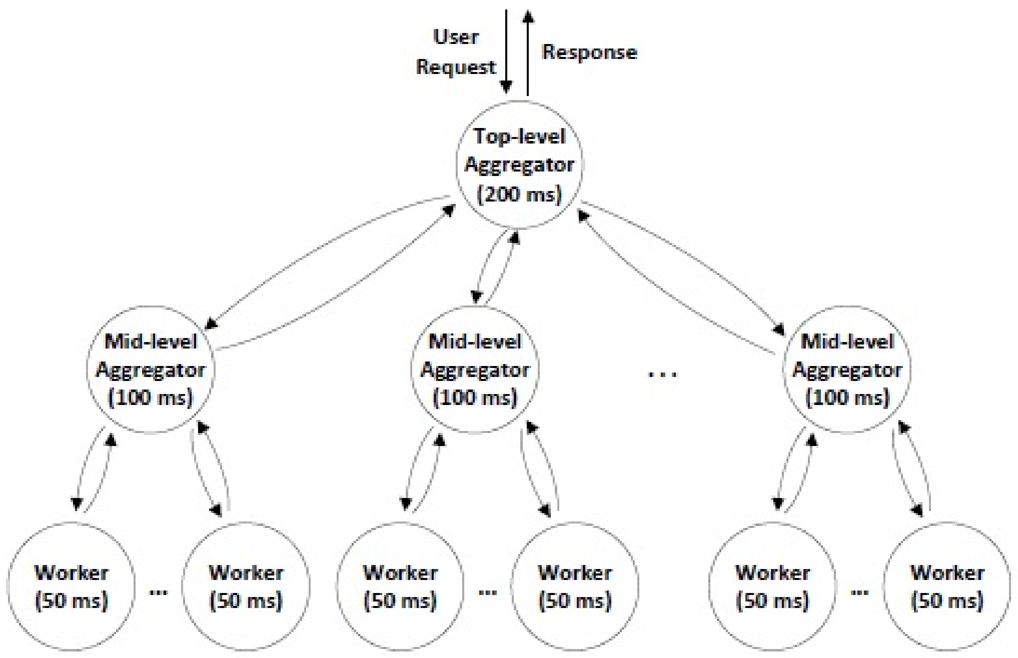

User Request
Response
Top-level Aggregator (200 ms)
Mid-level Aggregator (100 ms)
Mid-level Aggregator (100 ms)
...
Mid-level Aggregator (100 ms)
Worker (50 ms)
...
Worker (50 ms)
Worker (50 ms)
...
Worker (50 ms)
Worker (50 ms)
...
Worker (50 ms)

Figure 1.1 Example of Partition-Aggregate model [35].

There are several DCN architectures recently proposed. They are mainly classified as switch centric, server centric and hybrid structures [17] [40] [7] [34] [8]. In the switch centric architecture, switches are used to forward packets; in Server centric architecture, servers with multiple network interface cards (NICs) are used to forward packets and a hybrid architecture combines both. Examples of these architectures are described in the following section.

1.1 DCN Architectures

1.1.1 Three-Tier

This is one of most commonly used architectures. It consists of three layers: access, aggregation and core layers [18]. Servers are usually connected to the access-layer switches. Each access-layer switch is connected to two aggregation-layer switches. Further, these aggregation-layer switches are connected to two core-layer switches. This architecture is shown in the Figure 1.2. Core switches provide several services like load balancing, firewall, etc.

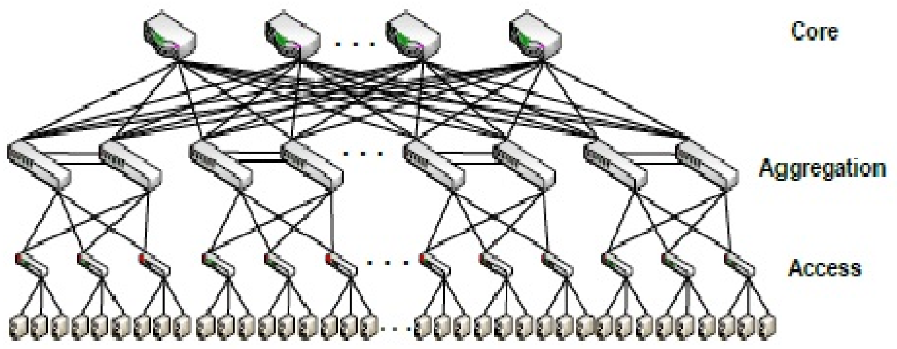


Figure 1.2 Three-tier architecture [29].

1.1.2 Fat Tree

This DCN consists of three layers: edge, aggregation, and core layers. The number of ports is k, which is same for all switches in a DCN. At the edge, there are k pods, or groups of servers, each pod has $k^2/4$ servers. Each edge switch is connected to $k/2$ servers, and the other $k/2$ ports of the switch are connected to $k/2$ aggregation switches. The total number of core switches in the network is $(k/2)^2$. Every core switch has one port connected to each of the k pods. A Fat Tree architecture with k-port switches can accommodate $k^3/4$ servers. Figure 1.3 shows an example of a Fat Tree DCN.

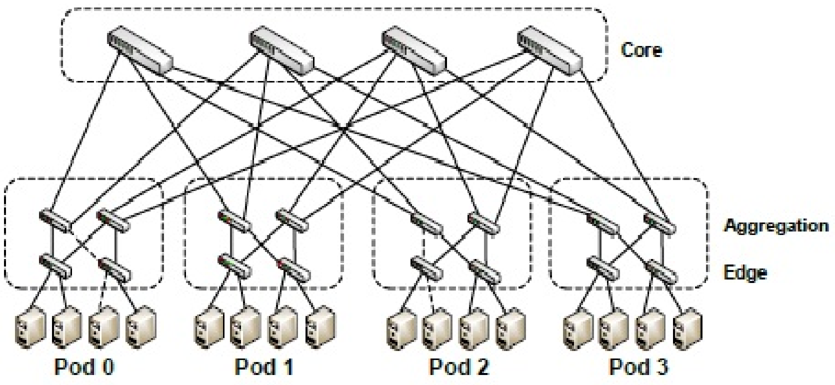


Figure 1.3 Example of a Fat Tree architecture [29].

1.1.3 VL2

This is a hierarchical fat-tree based architecture. The VL2 network uses three different types of switches: intermediate, aggregation, and ToR switches. Intermediate and aggregation switches have different port numbers denoted by Di and Da, respectively. The network uses Da/2 intermediate switches. Di aggregation switches and (Da*Di)/4 ToR switches. This type of network can support up to 20*Da*Di/4 ToR servers. This architecture is shown in Figure 1.4

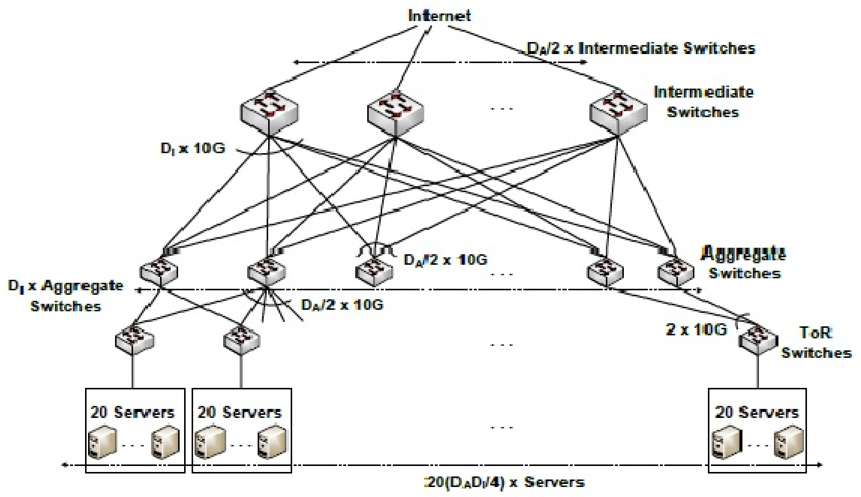


Figure 1.4 Example of a VL2 architecture [29].

1.1.4 CamCube

CamCube is a server centric architecture [10] [2]. It uses a 3D-Torus topology to connect servers directly to each other as shown in Figure 1.5. As it can be seen, there are no switches/routers in this architecture. Instead, CamCube allows applications running on servers to implement routing protocols by providing Application Programming Interfaces (APIs). Servers need to have multiple NICs to be part of this network. This architecture has high application-level performance. Also, because there are no switches/routers, its cost is low.

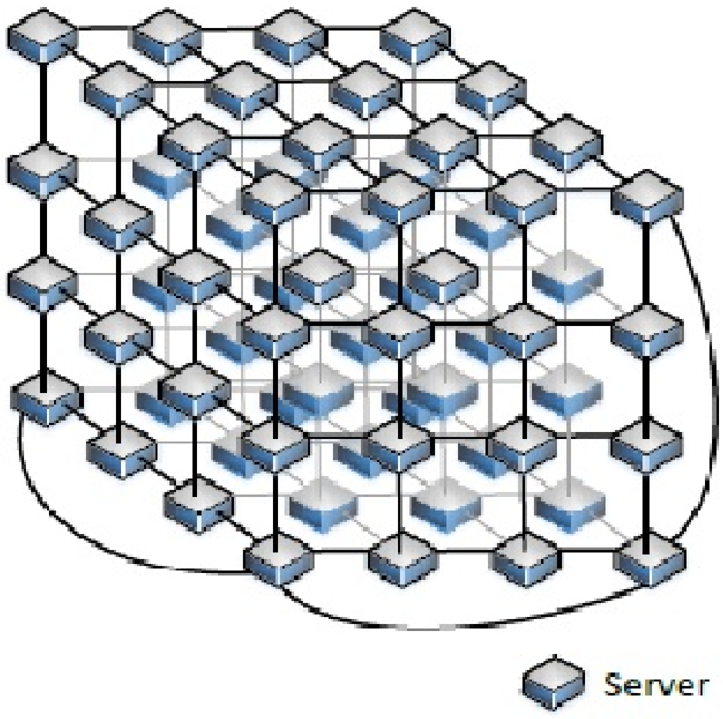


Figure 1.5 Example of a CamCube architecture [29].

1.1.5 DCell

DCell is a hybrid architecture [16]. It uses both switches and servers for packet forwarding. DCell uses a basic building block called DCell0 to construct larger DCells like DCell1, DCell2, etc. In general, DCell k ($k > 0$) is used to denote a level-k DCell that is constructed by combining n+1 DCell k-1s, where n denotes the number of servers in DCell0. DCell0 has n ($n <= 8$) servers and a commodity switch to interconnect them. Moreover, each server in a DCell0 is directly connected to a server in a different DCell0. The interconnection of all DCell0s forms a complete graph if each DCell0 is considered as a large virtual node. Figure 1.6 shows a DCell1, constructed with five DCell0s and 4-port commodity switches.

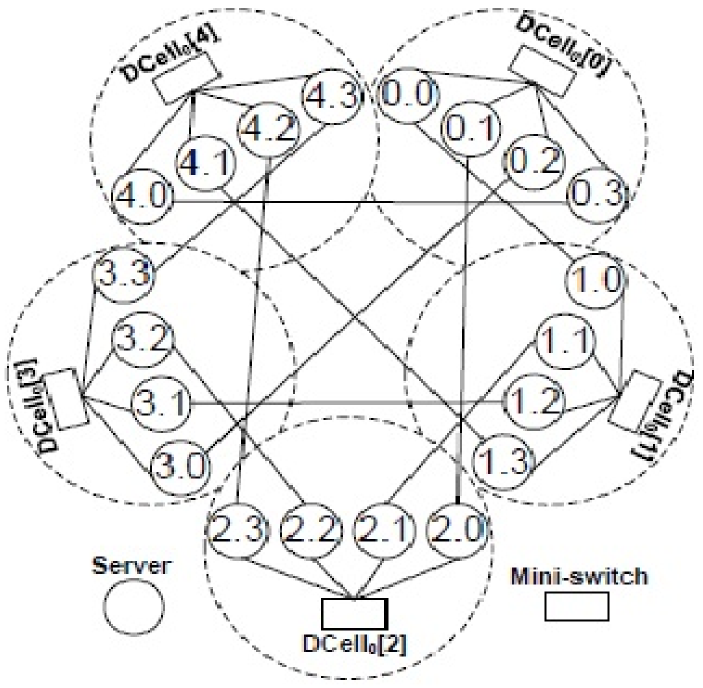


Figure 1.6 Example of a DCell architecture [29].

1.1.6 BCube

BCube is also a hybrid architecture [15]. BCube employs both servers and switches as forwarding elements. The scalability of this architecture is limited compared to Fat Tree, VL2, and DCell. However, BCube provides high bisection bandwidth and a graceful degradation of throughput under equipment failures. A bisection is created by partitioning a network into two equally sized sets of nodes. The bandwidth of a bisection is found by summing all of the link capacities between two partitions and the smallest bandwidth of all those partitions is the bisection bandwidth [13]. Similar to DCell, BCube also uses a basic building block usually named BCube0 to construct larger networks, which simply consists of n servers connected to an n-port switch. In BCube, n BCube0s and n n-port switches build a BCube1 network and so on. Figure 1.7 shows the architecture for n = 4 and Figure 1.8 shows the generic architecture.

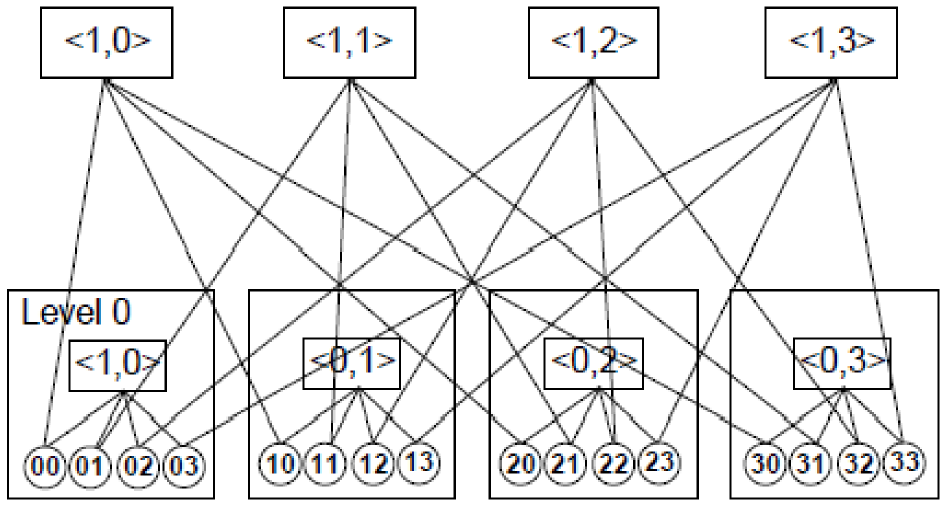


Figure 1.7 Example BCube architecture [29].

1.1.7 Traffic Patterns in DCNs

Traffic in data centers follow different distributions. Here, we discuss those distributions which are relevant for our experiments in this thesis. The first one is the uniform distribution. Uniform distribution means that values from an infinite or finite range can occur with equal probability. Traffic can be distributed among a number of servers or switch ports or in other words, are equally probable. Hence, uniform distribution can be as shown in Figure 1.9. Where p is the probability of occurrence which is same for all values. In our examples, we use uniform distribution to generate flows with random sizes. We assume that all packets are of fixed size that

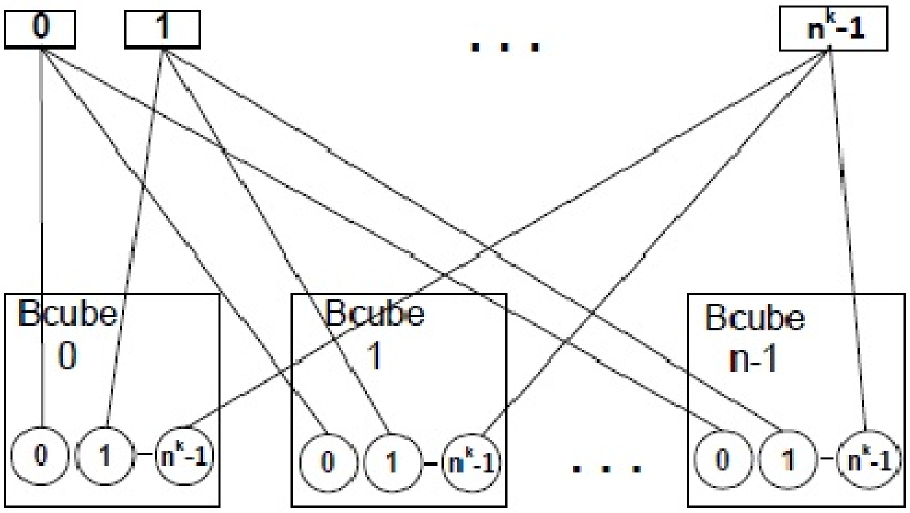


Figure 1.8 Example BCube architecture [29].

of 1KBytes.The flow sizes are between 2 to 198 packets. Generating flows of any size in this range are equally probable.

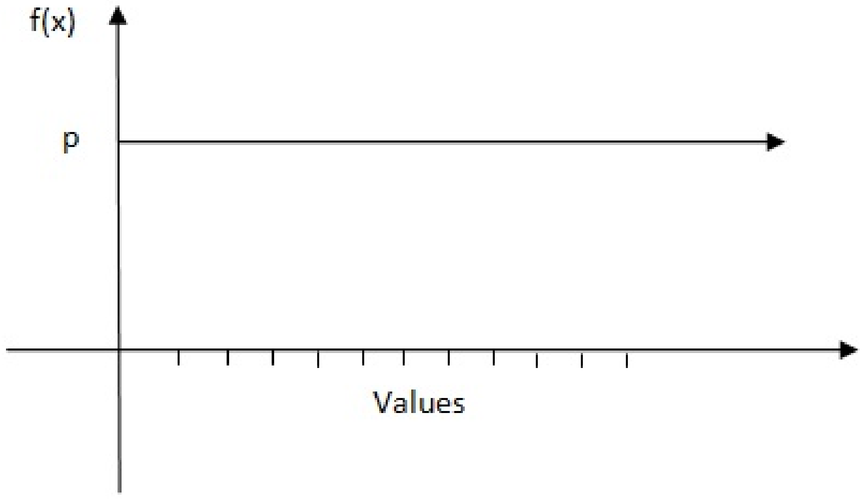


Figure 1.9 A general representation of a uniform distribution.

Unlike uniform distribution, in normal distribution, the probability of occurrences of values are not equal. For a finite range of values, values which are closer to the mean of the value set, including the mean, have higher probability. The probability decreases as we go farther from the mean, in both directions. Normal distribution is often illustrated using a bell curve as shown in Figure 1.10. The probability of occurrence of the mean value is the highest. Of all values, 68 percent are within one standard deviation from the mean (Figure 1.10), 95 percent are within two standard deviations from the mean (Figure 1.11) and 99.7 percent are within three standard deviations from the mean (Figure 1.12). The mean divides the bell curve into two parts with equal area.

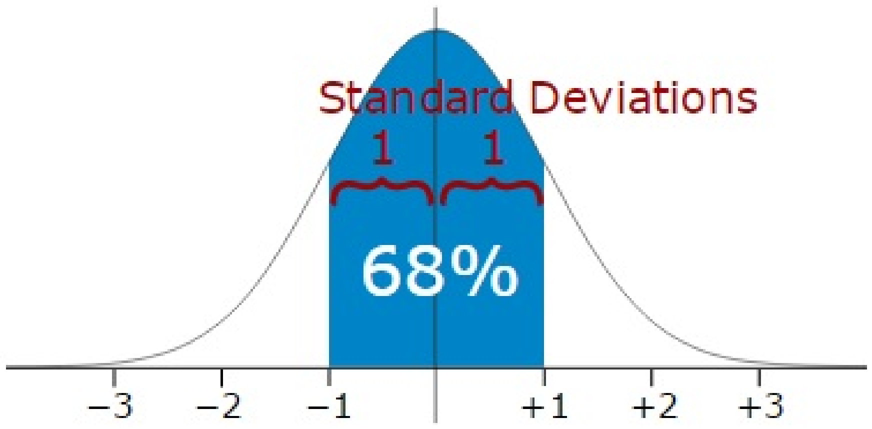


Figure 1.10 One standard deviation in a normal distribution [1].

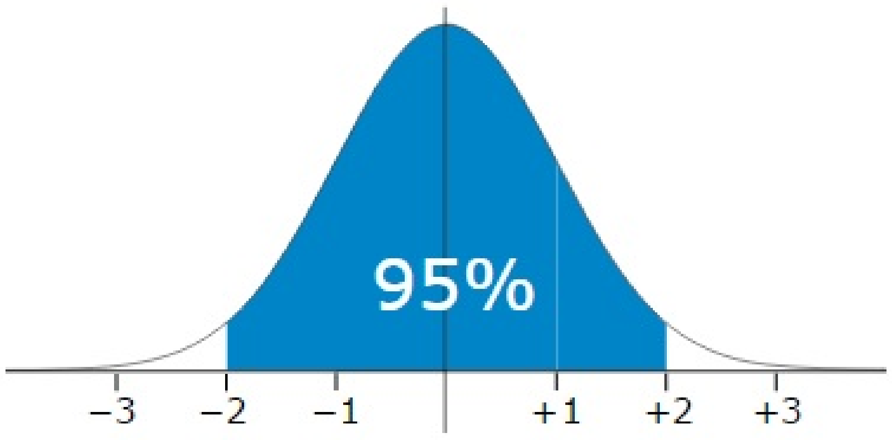


Figure 1.11 Two standard deviations in a normal distribution [1].

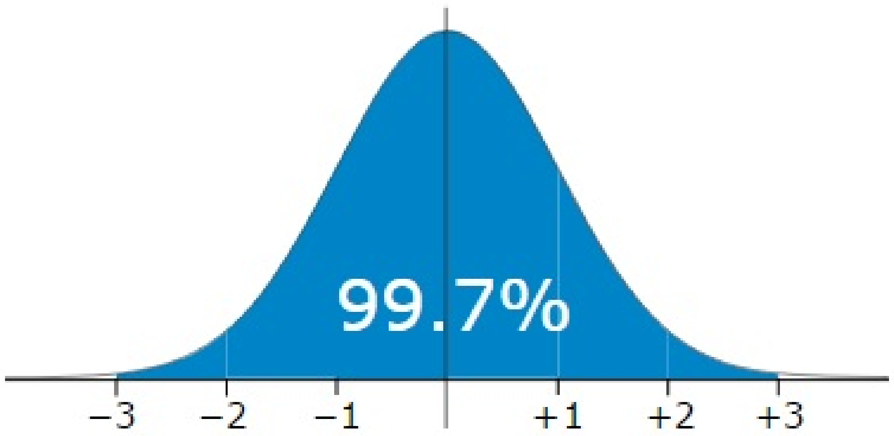


Figure 1.12 Three standard deviations in a normal distribution [1].

For understanding exponential distribution, we need to understand Poisson process. Poisson process is a process where events occur continuously and independently at a constant rate. The probability distribution that describes the time between events in a Poisson process is called exponential distribution. A sample exponential distribution is as shown in Figure 1.13

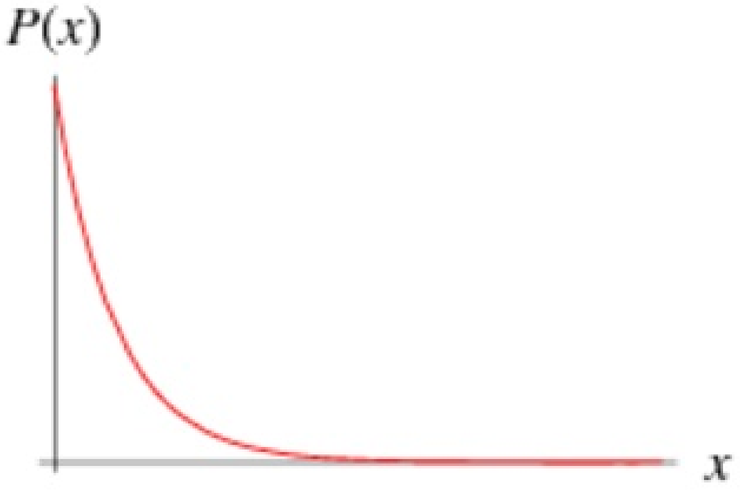


Figure 1.13 Example of exponential distribution.

1.2 Transmission Control Protocol

Transmission Control Protocol (TCP) is one of the main protocols in the Internet Protocol suite. It is the standard transport protocol used in the Internet for reliable data transfer services. TCP and many of its variants customized specifically for data centers have been used in data centers. Some of these protocols are Data Center TCP (DCTCP), Multipath TCP (MCTCP), Deadline Aware DCTCP (D^2TCP).

Before the actual data transfer starts, TCP end points have to establish a connection. TCP uses a three way handshake as connection establishment procedure. The initiator sends a SYN message to which the other end point sends an ACK. The initiator sends an ACK for the received ACK. TCP tracks every byte of data transferred by assigning them a sequence number. The ACK for data transfer carries the sequence indicating the byte number that is expected to receive. If there is any data missing, then the receiver detects it and requests for retransmission using the sequence number of the missing byte in the ACK.

TCP also provides error detection and timeout based retransmission services. Flow control is another feature of TCP where the sender sends data at a rate the receiver can handle. The receiver can inform the sender if the sender is sending at a rate which is overwhelming the receiver. In other words, the receiver can ask the sender to slow down to its own rate, thereby avoiding any packet loss. TCP achieves this by a technique called sliding window. TCP also defines congestion window as the maximum amount of data that can be sent without getting an ACK. If there is no congestion in the network, the congestion window increases enabling the end point to send more data for one ACK. When a congestion is detected, TCP reduces the congestion window exponentially, thereby reducing the number of data bytes sent for one ACK.

1.2.1 The TCP Incast Problem

The use of TCP is not as widespread in data centers as it is in the Internet [31]. One of the many reasons for this is the TCP incast issue [26] [38]. In the Partition-Aggregate model used in most data centers, an aggregator may request data which may be spread across many servers. The servers send responses what the aggregator combines. When the servers send their responses, if the buffer at the switch overflows, there will be packet loss. The TCP Retransmission Time Out (RTO) in the servers, which is about 200ms, will cause the server not to retransmit the lost packet until the RTO occurs. Since the aggregator has not received responses from all the designated servers, it will not issue new requests. As a result, those servers whose responses are received without packet loss will be idle. This causes the throughput at the aggregator side to drastically reduce. This phenomenon is called the TCP incast problem [39]. Figure 1.14 shows the collapse of throughput due to incast as the number of servers increases. This is because as the number of servers increases, the buffers at the switch fill up at a faster rate and the probability of packet loss increases.

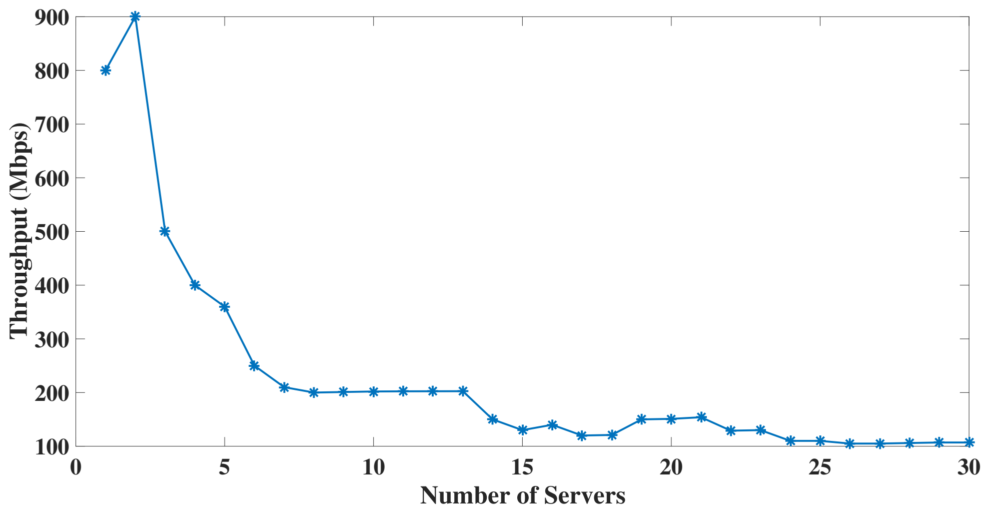


Figure 1.14 Throughput collapse in TCP Incast.

1.3 Scheduling Disciplines

Scheduling disciplines are algorithms used in computing systems and computer network equipment, such as in switches and routers, where incoming packet traffic has to be forwarded but there is no capacity to send all packets at the same time and some have to be sent first and other must wait. Scheduling, in this context is about deciding which among the pending requests have to be served and in what order, in an efficient way and in certain cases, fair [22] [27]. There are many well-known scheduling disciplines. Here, we will introduce some of those disciplines which will be helpful in describing the transport schemes we use in DCNs, and discussed in this thesis.

1.3.1 First Come First Serve (FCFS)

This is one of the basic scheduling schemes. To decide which request has to be served next, this scheme chooses request arrival time as the criteria. The first arrived request is served first. In other words, requests will be served in the same order in which they arrive. This is same as First in First Out (FIFO). FCFS is used in the scheduling scheme Deadline Driven Delivery D^3 which is one of the schemes we use to compare the results of our proposed scheme PEQ.

1.3.2 Earliest Deadline First (EDF)

This scheduling discipline applies to those requests with a time limit on the time they must be served. This time is called the deadline. The request is of no use if it is not served completely before its deadline. Hence, this scheme first schedules the request that has the smallest deadline. This scheme seeks the request that is closest to its deadline as the most urgent request and hence prioritizes it first. PDQ is one of the transport schemes that uses EDF. This scheme is considered an optimal scheme for serving requests with deadlines. However, as we will show in the subsequent sections that EDF is not optimal always. Also, we will show that considering other parameters can outperform EDF.

1.3.3 Shortest Job First (SJF)

This scheduling scheme considers the size of the requests to decide which request is selected to be served next. In the current context, size can be defined as the processing time required. Hence, this scheme selects requests with the least processing time. One of the advantages of this scheme is that it minimizes the average waiting time for all of its requests [30]. Since we deal with flows, we define flow completion time (FCT) as the time required to serve a flow completely. Using SJF a smaller average FCT may be achieved.

1.4 State of the Art

The main objective of most of the data center transport schemes are to either solve the TCP incast problem or to minimize flow latency. Transport schemes for minimizing flow latency are of broadly two types, flow deadline agnostic or flow deadline aware. For the deadline aware schemes, the most important performance parameter is the application throughput. Hence PEQ, which is also a deadline-aware scheme, focuses on improving the application throughput as its most important performance parameter.

There are many schemes focusing on minimizing the FCT [24] [5] [37] [36], like Rate Control Protocol (RCP), Data Center TCP(DCTCP), Router Assisted Capacity Sharing (RACS). RCP is a scheme used to control congestion in DCNs [12]. It is a deadline agnostic scheme. RCP achieves this by emulating Process Sharing (PS) at each router. Process sharing is a service policy where every job receives an equal share of the available service capacity [9]. RCP routers simply assign a single rate to all the flows that pass through them. The router calculates the value of this rate on the basis of the current queue occupancy and the aggregate input traffic rate. This single rate, R(t), is calculated and updated every RTT. Thus, RCP may maintain near-constant queue occupancy.

DCTCP is a scheme that uses Explicit Congestion Notification (ECN) to control congestion [4]. ECN is an extension to the TCP which allows end-to-end notification of network congestion. It uses most of the algorithms in TCP except those for congestion control. DCTCP issues ECN messages indicating the extent of congestion instead of just indicating if there is congestion or not. ECN is typically used with an active queue management technique such as Random Early Detection (RED) at switches/routers. ECN uses a field in the IP header with two bits, called ECN codepoints, to inform the receiver that end hosts are ECN-capable and about the incipient congestion. The ECN codepoint "11" is assigned to indicate congestion and is called the Congestion Experienced (CE) codepoint. Any router along the path between the source and the destination sets the CE codepoint if its average queue length is above a predefined threshold. In this case, the receiver generates an Acknowledgement (ACK) packet marked with an ECN Echo flag (ECE) in the TCP header to reflect the encountered congestion upon receiving the packet with the CE codepoint set. The sender's TCP reacts by halving the congestion window (cnwd) and reducing the value of the slow-start threshold (ssthresh).

The RACS scheme [25] was proposed to emulate the Shortest Remaining Processing time (SRPT) policy, which is similar to the SJF policy discussed previously. SRPT selects the job with the shortest remaining time first. In RACS, every flow is assigned a weight corresponding to either the remaining processing time or the residual flow size. This information is updated periodically so that routers re-allocate the rate for each flow after each update. A RACS switch allocates bandwidth in proportion to the flow's weight.

There are numerous deadline aware schemes for flow scheduling like Deadline Driven Delivery (D^3), Preemptive Distributed Quick (PDQ), Deadline-Aware Datacenter TCP (DDTCP) etc

DDTCP, like DCTCP, controls its congestion window proportional to the extent of congestion in the network [32] [29]. The switch monitors the queue length. When it increases beyond a threshold, it marks CE codepoint and when the receiver receives packets with CE codepoint enabled, it sends a ECN feedback to the sender. The sender then adjusts its congestion window. If no congestion is detected, the sender increases its congestion window.

D^3 uses a greedy approach for serving flows [35]. Each flow passing through one or more aggregator switch requests its sending rate every RTT based on its size and deadline. The switch has fixed capacity. If the remaining capacity at the switch is more than the requested rate, the switch allocates the requested rate. Along with the requested rate allocation, if the remaining capacity had to be equally shared by all the started flows, also known as fair share, is allocated to the current flow. If the switch does not have enough capacity to serve the flow at its requested rate, D^3 allocates all of its remaining capacity to the flow. D^3 picks flows greedily, or on a FCFS basis. D^3 does not maintain any kind of state information about the flows at the switch. The senders calculate the requested rate every RTT. If the flow is not served, the requested rate in the next RTT increases. The main drawback of D^3 is that it selects flows on a FCFS basis, which may not be always effective.

PDQ uses two scheduling policies, EDF and SJF. Here, EDF has a higher priority [21]. Flows with tighter deadlines are considered more critical and are prioritized for serving. In case of a tie, the flow with the smallest size would be selected. Hence, SJF is used as a tie breaker for EDF. The switch maintains the state of the flows currently being served. At the beginning of every time slot, there may be new flows arriving. Out of the newly arriving flows, PDQ selects flows starting with smallest or earliest deadline and so on. If there is a tie, PDQ resolves it by prioritizing the flow with smaller size. By doing so PDQ creates an ordered list according to which PDQ starts serving the flows, starting from the head of the list, as per the available bandwidth. The flows in this list will be updated when new flows arrive or any flow in the list finishes. New flows may preempt existing flows. The flows which are not being served currently will have to wait. Since their deadlines are fixed, some of the waiting flows may become invalid. These invalid flows are removed from the list every time slot. This process is called as Early Termination. EDF may yield a near optimal scheduling in terms of average FCT when deadline-constrained flows are considered. However, PDQ may not produce a near-optimal or optimal flow scheduling if the only criteria to schedule the flows is their deadlines. Figure 1.15(a) shows 4 flows with their sizes and deadlines given in unit time.

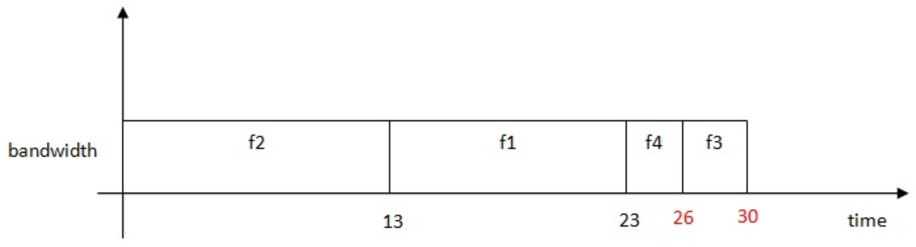
Since PDQ uses EDF, the flows are ordered as shown in Figure 1.15(b). EDF policy schedules the flows with shorter deadlines first without considering their sizes. Therefore, f2 and f1, which have larger sizes than f4 and f3, are served before f4 and f3. By the time f2 and f1 finish, the deadlines of f4 and f3 are violated as indicated in red, in Figure 1.15(b)

If, however, flows, who can afford to wait and be completed before their deadlines, wait, as for f1 in this example, the number of flows completed could be increased from two to three. Consider the following order of flows as in Figure 1.15(c)

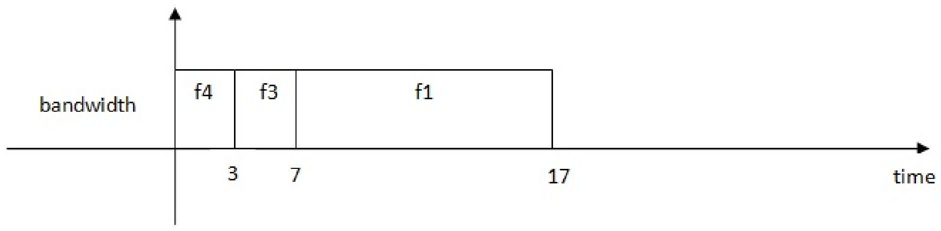
Here, flow f1 has its size as 10 time slots and deadline as 23 time slots. So the latest time slot at what f1 can be started and still be completed successfully is the 13th time slot. If the flow starts after the 13th time slot, it cannot be completed before its deadline expires. Hence, we try to fit in as many small flows as possible within 13 time slots. In the example, we schedule flows f4 and f3. Flows f4 and f3 complete after 7 time slots, hence f1 is also be able to be completed since it starts

Flow ID	Size	Deadline
f1	10	23
f2	13	15
f3	4	26
f4	3	25

(a) Example of arriving flows.



(b) Flows ordered as per EDF.



(c) Improvement over EDF.

before the 13th time slot. If f2 is scheduled anywhere before or in between f4, f3 and f1, the overall throughput decreases, hence f2 is not scheduled. This is a trade off.

CHAPTER 2

PREMPTIVE EFFICIENT QUEUEING

2.1 Protocol Description

In PEQ we define an attribute that we call “cushion.” It is the difference of a flow’s deadline and its transmission time. It indicates how long the flow can afford to wait before being transmitted and without violating its deadline. The bigger the cushion, the larger the number of other flows may be transmitted before the flow is transmitted.

At the beginning of every time slot, there may be new flows arriving. Every flow is assigned an ‘H’ value upon arrival. This is calculated as follows

$$H = \alpha\tau + \beta(\tau/D) \tag{2.1}$$

where τ is remaining flow transmission time in time slots, D is remaining flow deadline in time slots, α a weight factor for the transmission time and is assigned a constant value of 0.6, β the weight factor for the cushion of the flow and is assigned a constant value of 0.4

The first term indicates the flow size. The second term, τ/D, is the relative cushion and it shows how large is the flow transmission time compared to its deadline. The coefficients α and β serve as weights for the first and second term respectively. The values are chosen such that the flow size or transmission time carries more weight than the relative cushion. So smaller flows usually have lower H value and vice versa.

PEQ generates an ordered list of flows to serve in decreasing order of flows’ H value; smaller H values first.

2.2 Handling of Incoming Flows

The following is a step-by-step description of PEQ operation. *SelectPEQFlow* is a function to select a flow with minimum H value among all the other flows at the beginning of a time slot. *PEQEnqueue* implements a function to insert flows Id in the service queue.

- At the beginning of a time slot, all newly arrived and queued flows are considered. PEQ calculates the H value for each flow using the flow's transmission time and deadline as described previously.
- The flow with the minimum H value is picked by *SelectPEQFlow*. This flow is given to *PEQEnqueue* as input.
- Because this is the first flow, it is added to the service queue managed by *PEQEnqueue*.
- *SelectPEQFlow* picks the flow with smallest H value out of the unscheduled flows and passes it to *PEQEnqueue*.
- *PEQEnqueue* may have already created a service queue with one or more flows. The flow(s) in the queue can be seen as a single flow with a transmission time equal to the sum of transmission times of all the flows in the queue. We call this as aggregated transmission time. The cushion of the new flow is checked. If the cushion of the new flow is large enough to accommodate the aggregated transmission time, then the flow is added as the new tail flow of the service queue.
- If the cushion of the new flow is not large enough to accommodate the aggregated transmission time, then check the cushion of the scheduled flows, starting from the tail flow in the service queue each, to the head flow in the service queue. To check any flow in the queue, all the other flows in front

of it and the new flow are seen as a single large flow with transmission time equal to aggregated transmission time until that last flow. If the aggregated transmission time fits in the cushion of the flow being checked, then the new flow is inserted in front of it.

- Check *PEQEnqueue* if the flow's deadline would be expired or not by inserting it in that specific position. If the deadline would not be expired, then the new flow is inserted. Else Step 6 is repeated with the next queued flow in front of the current flow.

- If the flow cannot be inserted in the list (as that may force one or more of the queued flows violate its deadline), then the new flow is discarded. Therefore, cushions of flows closer to the head of the queue are no checked.

- Repeat Step 4 and the following steps until all the newly arrived flows are checked for insertion.

The pseudo code of the description above is as shown in Algorithm 1. PEQ does not maintain the state of the flows at the switch. This feature simplifies the scheme considerably. At the beginning of a new RTT, the flows picked up or/and served in the previous RTT are considered along with newly arrived flows. This property allows new flows to preempt a flow which was previously being served but that it may not longer remain in high priority for service. There is an exception for the flow being currently served, all other flows have to wait. Therefore, their remaining time to deadlines decrease. At some point few flows may become expired (or will expire before been completed). Such flows are checked and removed every RTT.

Also, as the pseudo code shows we compare the new flow starting from the tail of the queue rather than from the head. If the comparison had started with the head flow and if the new flow is inserted at a particular position, then the flows below the newly inserted flow may be delayed and this delay may make the flows to violate their deadlines. To avoid this, we start the comparison with the tail flow, so for any particular position in the queue, we ensure that the insertion of the new flow will not disturb complying with deadlines of the flows below.

Table 2.1 Variables used in the Pseudo Code.

Variable Name	Description
$nFlow$	new flow returned from *SelectPEQFlow*
$nTailFlag$	Flag to indicate the new flow will be the new tail of the queue.
$insertionInd$	Indicates the position where the new flow will be inserted.
$fPtr$	A flow pointer to iterate through the queue.
$qHead$	Pointer to the head of the queue
$qTail$	Pointer to the tail of the queue
τ_a	Aggregated transmission time

```
if queue is empty then
    create a flow entry and populate flow attributes;
    return;
else
    for $fPtr = qTail$ to $qHead$ do
        if $((fPtr{==}qTail)$ and $(nFlow.D$ - $nFlow.\tau \geq fPtr.\tau_a))$ then
            $insertionInd$=NULL;
            $nTailFlag$=1;
            break;
        else if $(fPtr.D$ - $fPtr.\tau_a \geq nFlow.\tau)$ then
            if $(fPtr.\tau_a$ - $fPtr.\tau) + nFlow.\tau \leq nFlow.D$ then
                $insertionInd = fPtr$; //Remember this position
                continue;
            else
                continue;
            end
        else
            break;
    end
end
if ((insertionInd == NULL) and ($nTailFlag$)) then
    insert the nFlow as the new tail flow of the queue;
else if $insertionInd \neq NULL$ then
    insert the nFlow at index pointed by insertionIndex;
else
    discard nFlow;
```

Algorithm 1: PEQEnqueue: PEQ handling of new incoming flows.

CHAPTER 3

EVALUATION

Typically there are two types of flows in data centers. Deadline unconstrained long flows and deadline constrained short flows [23] [6]. Here we consider only deadline constrained short flows. Hence, our main performance parameter is application throughput. We also evaluate FCT.

3.1 Formal Properties and Assumptions

We use the typical partition-aggregate topology as our test environment.

We define a flow by a set of parameters: flow size, transmission time, deadline, rate and H value. The units for all the flow parameters are time slots. We assume a fixed packet size of 1Kb. Also, one packet or data equivalent to 1Kb can be transferred in one time slot. Hence, we consider the size of short flows to be in the range of 2 to 198 timeslots. Flow sizes follow a uniform distribution. Flow deadlines follow an exponential distribution. Rate is fixed to 1 packet per timeslot. We assume we have long queues in the switch and the servers so that the system is lossless. In our evaluation, RTT=1 time slot.

3.2 Implementation

We have developed all the simulators listed below in C language, on Linux operating system:Ubuntu.

- PEQ
- PDQ
- D^3: Two different versions of D^3 are implemented. One version where the switch running D^3 does not maintain any flow state information. This D^3

mode is as per the specifications mentioned in [35]. In the other version of D^3, the switch maintains flow state information. Here the switch maintains the list of flows currently being served. Every RTT, these flows are served first and new flows are added as the serving flows are being completed. If available capacity allows additional flows, only then new flows are included. We call this scheme Statefull D^3 and it is our own proposed modification to D^3 to make it a more fair comparison to PEQ as Statefull D^3 is expected to achieve the maximum performance of D^3 original working principle [35]. We will see later that effectively this enhancement significantly improves the performance of D^3.

- Optimal Scheme: A scheme which performs optimal flow scheduling by evaluating all possible combinations for a set if incoming flows.

Before presenting the various evaluation scenarios, we elaborate on the example in presented in Section 1.3. If larger flows have tighter deadlines while small flows have longer deadlines, PDQ's performance deteriorates significantly. As PDQ uses EDF, it starts serving the bigger flows and *early termination* eliminates the short flows gradually. We observed the impact of number of such flows on application throughput. We generate flows once, at the beginning of the simulation and the schemes schedule and serve those flows. We varied the number of flows while measuring the application throughput. Figure 3.1 shows the results. It can be seen that the application throughput of PEQ is considerably larger than that of PDQ. As the number of flows increases, PEQ's application throughput is larger than twice that of PDQ. For the same test scenario, we also measured the time taken by PEQ and PDQ for completing the flows constituting the achieved application throughputs. It can be seen from the Figure 3.2 that PEQ completes more flows than PDQ while taking almost half the time PDQ takes. This is because PDQ spends time on longer flows which may not count towards its application throughput while PEQ starts with smaller flows and flows with large cushions which are more likely to count towards its application throughput.

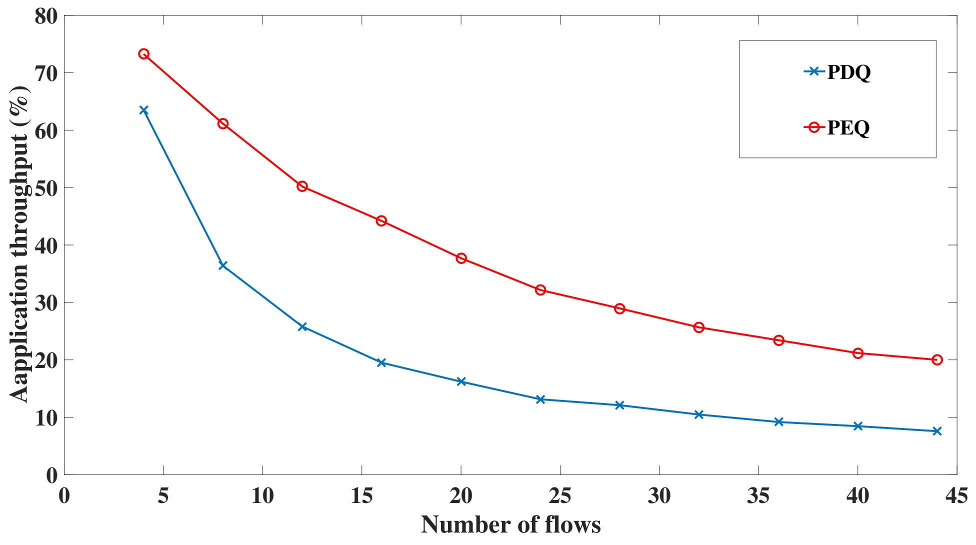


Figure 3.1 Application throughput comparison of PEQ.

We repeated the above experiment by generating new sets of flows after every fixed number of time slots, called flow generation interval. We define flow generation interval as the number of time slots after which new set of flows generated. We varied the flow generation interval from 5 time slots to 30 time slots for 24 servers, each generating one flow. Figure 3.3 shows the results.

It can be seen from Figure 3.3 that as the flow generation increases interval, the percentage application throughput of PEQ increases as compared to that of PDQ. When flow generation interval is very small, due to preemption, more flows are partially served hence decreasing the application throughput. As the flow generation interval becomes larger, the performance of PDQ and PEQ tend towards that of one time flow generation, as shown in Figure 3.1.

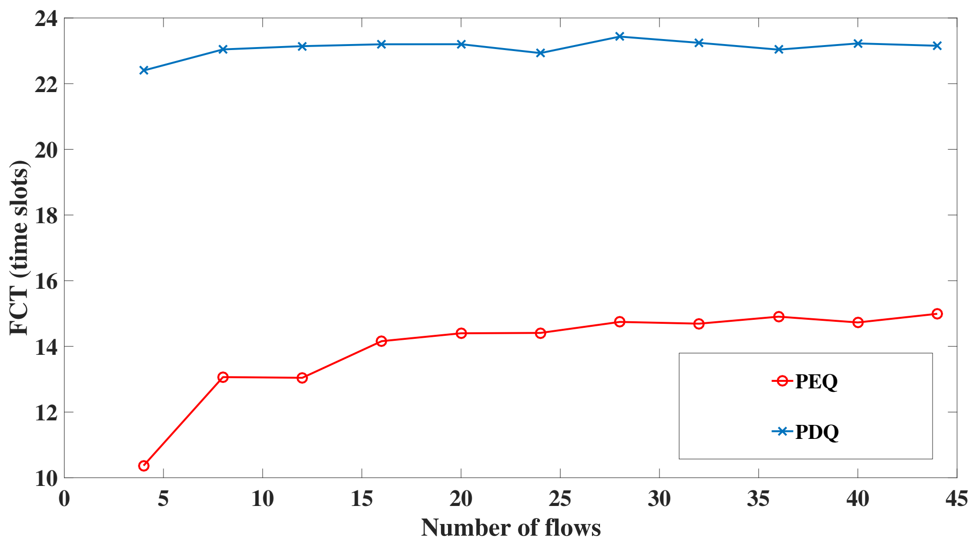


Figure 3.2 Flow completion time comparison of PEQ.

3.3 Comparison with Optimal Flow Scheduling Scheme

We implemented the Optimal Scheduling scheme as a simulation program that calculates all possible combinations of flows and evaluates the throughput for each combination. The program stores the combination which yields the maximum application throughput and the actual application throughput. For a set of flows, this is the best possible scheduling. If some flows fail in this case, it means that it is not possible that that those flows can succeed.

Here, we compare PEQ and PDQ with the Optimal flow scheduling scheme. A set of flows is generated only once at the beginning of the simulation. The optimal scheme, PDQ, and PEQ all schedule the same set of flows and serve them. We assume that there is no preemption in this experiment. The results show that PDQ is very close to the optimal scheme already. But PEQ is even closer to the optimal scheme than PDQ. Because PDQ is close to optimal scheme, the apparent small improvement in application throughput of PEQ over PDQ that we discuss in all the experiments discussed in the next sections, is not trivial.

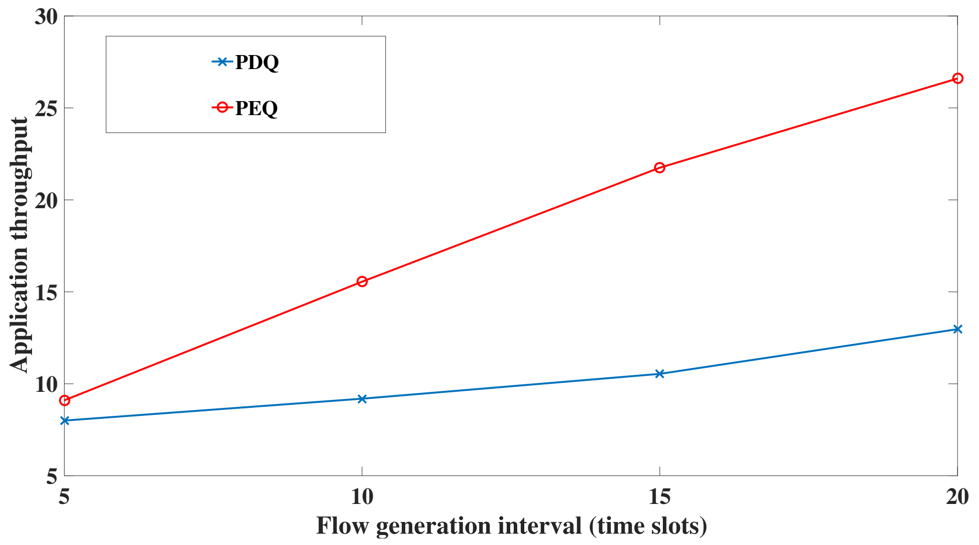


Figure 3.3 Application throughput comparison of PEQ (varying flow generation intervals).

In this experiment, we vary the number of flows generated and observe the application throughput of the Optimal scheme, PDQ, and PEQ. The application throughput of PDQ and PEQ are given in comparison with the that of the Optimal scheme. Hence the application throughput of the Optimal scheme is always 100 percent as we use admissible number of flows, flow sizes, and deadlines. Figure 3.4 shows the results.

3.4 Effect of Deadline on Application Throughput

In this experiment, we observe the impact of flow deadlines on application throughput. The flow sizes are uniformly distributed in the range of 2 to 198 time slots. The deadlines are exponentially distributed with the means in the range of 20 to 120 time slots. A mean value set to 20 causes the flows to have smaller (and tighter) deadlines. As the mean value is increased, the flow deadlines also increase making them longer (with more time for the flow to be completed). We also perform the experiment for different number of servers. The application throughput is evaluated for 40 to 100 servers, with step increases of 20 servers. Figures 3.5, 3.6, 3.7, and 3.8 show this.

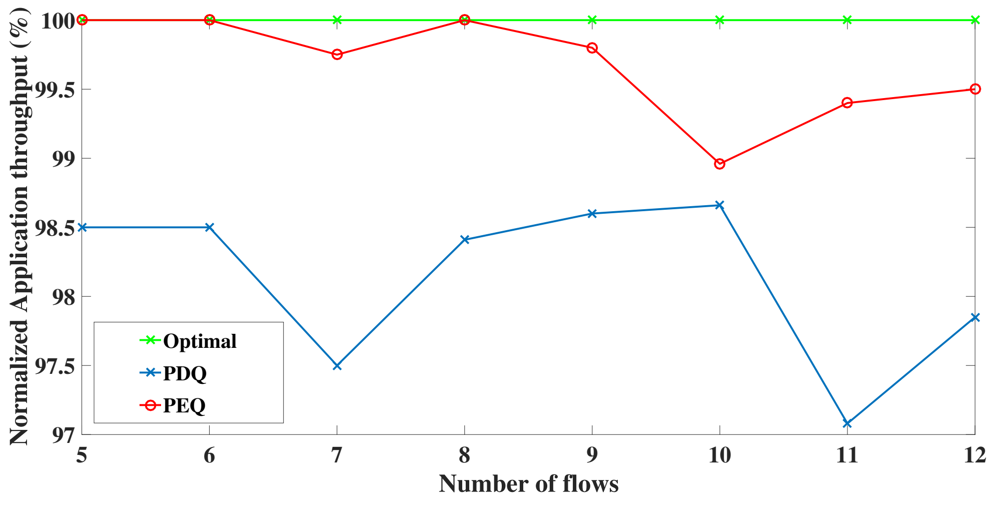


Figure 3.4 Application throughput comparison of Optimal Scheme, PDQ, and PEQ.

It can be seen from the graph that PEQ achieves higher application throughput than other schemes for all number of servers. Increasing the mean value allows all schemes to finish more flows on average and hence the results show the application throughput for all schemes to be non-decreasing

3.5 Effect of Overloading the Switch Output Link on Application Throughput

We consider the system to be overloaded when there are more flows generated than that can be served by the switch throughout the duration of simulation. For all the remaining experiments conducted, the simulation duration is 1 million time slots. Because the flow size is uniformly distributed from 2 to 198 timeslots and by considering the average flow size, we can find out how many such flows can possibly be served in 1 million timeslots. This number of flows when generated makes the system fully loaded. If the number of flows exceed this number, then the system is considered overloaded. We can set the amount of system load by controlling the number of flows generated. One way is to increase the number of servers and the

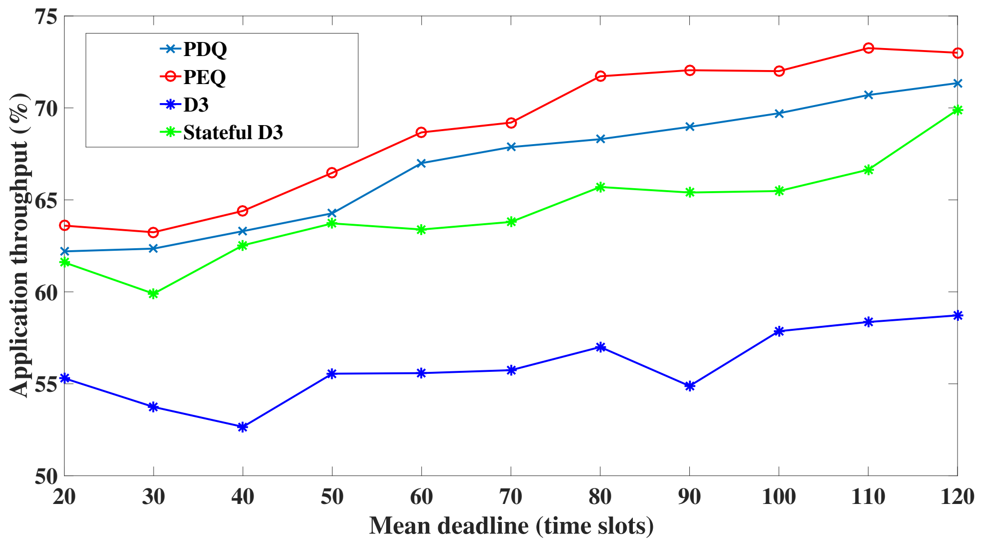


Figure 3.5 Application throughput comparison as a function of flow deadline (40 servers).

other is by generating more flows (increasing the generation rate). As mentioned earlier each server can have utmost one active flow at a time. Only after the current active flow is completed can the server generate a new flow. When the server is idle, it generates a flow with probability p, which we call the flow generation probability. Therefore, for a given p we can increase the number of servers or else, explicitly increasing p. In the following experiments, we overload the system in using these two strategies and observe the impact on the application throughput.

Initially, we overload the system by increasing the number of servers. We adjust the system to be fully loaded for 40 servers and increase the servers from 40 to 120, in steps of 10 servers. Figures 3.9, 3.10 and 3.11 show this. As the number of servers increases, they generate more flows and the load at the aggregator switch increases. We repeat the experiment for three different classes of flow deadlines namely tight, moderate and relaxed. For tight flow deadlines, we set the mean of the exponential distribution as 30 (Figure 3.9). For moderate deadlines, we set the mean as 90 (Figure 3.10), and for relaxed deadlines, we set the mean as 120 (Figure 3.11). It

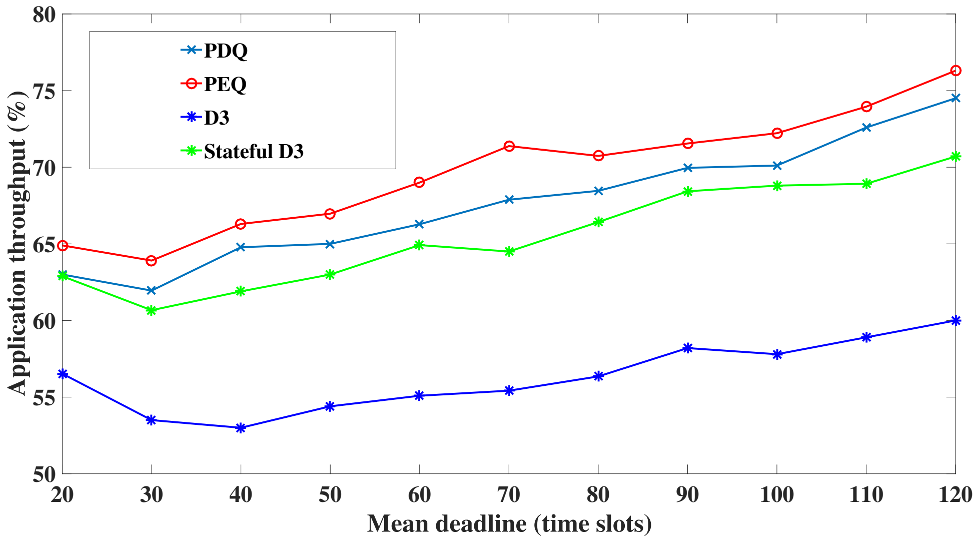


Figure 3.6 Application throughput comparison as a function of flow deadline (60 servers).

can be seen from the graphs that for all three types of flow deadlines, PEQ achieves higher application throughput than PDQ, Statefull D^3, and D^3.

Next, we overload the system by keeping the number of servers constant at 180. The flow deadlines are also made tight by setting the mean for exponential distribution as 30. The probability is gradually increased enabling servers to generate flows more frequently and thereby increase the load at the switch. We observe the impact on the application throughput. As the load increases, the application throughput of all the schemes decreases overall. It can also be seen that PEQ has higher application throughput compared to the other schemes. We repeat this experiment for the three different classes of flow deadlines, namely tight deadlines with a mean for exponential distribution as 30 (Figure 3.12), moderate (mean = 90, Figure 3.13) and relaxed (mean = 120, Figure 3.14).

3.6 Effect of Flow Size on Application Throughput

For most of the testing, we have considered flow sizes to be uniformly distributed in the range 2 to 198 timeslots [21]. In this test, the flow size is varied and its impact

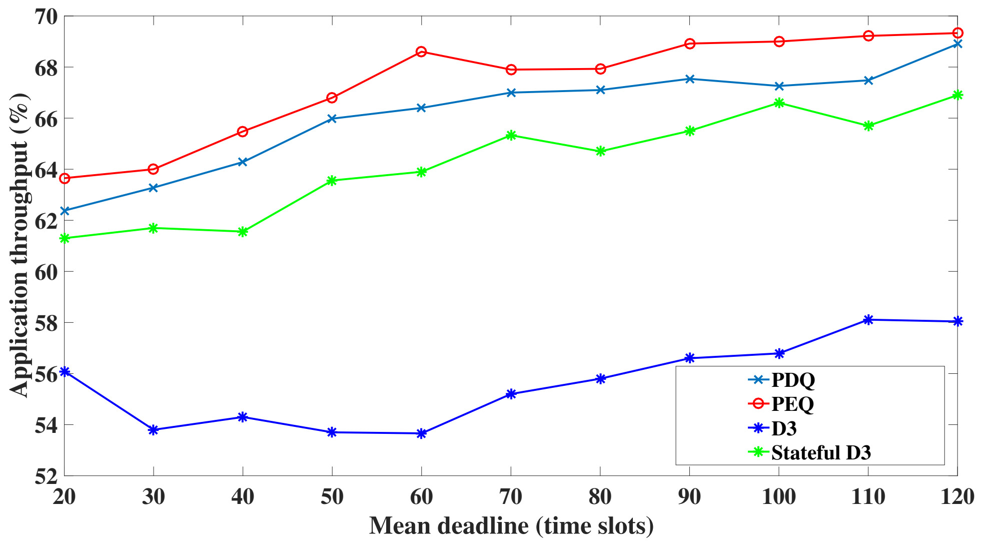


Figure 3.7 Application throughput comparison as a function of flow deadline (80 servers).

on the application throughput is observed. The deadlines are fixed, i.e. they are exponentially distributed with the mean set as 100 times slots. The upper limit of the range of flow sizes is varied from 2 to 50 times slots, until 190 time slots in steps of 20 time slots. For fixed deadlines, when the flow sizes are between 2 and 50 time slots, there is a large number of short flows that can be completed. As flow sizes increase, it takes more time for the flows to be completed and hence the number of flows completed successfully comes down comparatively. Therefore, the average application throughput decreases as the flow size increases. Figure 3.15 shows the results for 40 servers. Initially, the system is optimally loaded for 40 servers. The upper limit for flow size is varied and the application throughput is recorded. This experiment is repeated for 100 servers (Figure 3.16). It can be seen from the results that PEQ achieves a higher application throughput than the other schemes for all flow sizes.

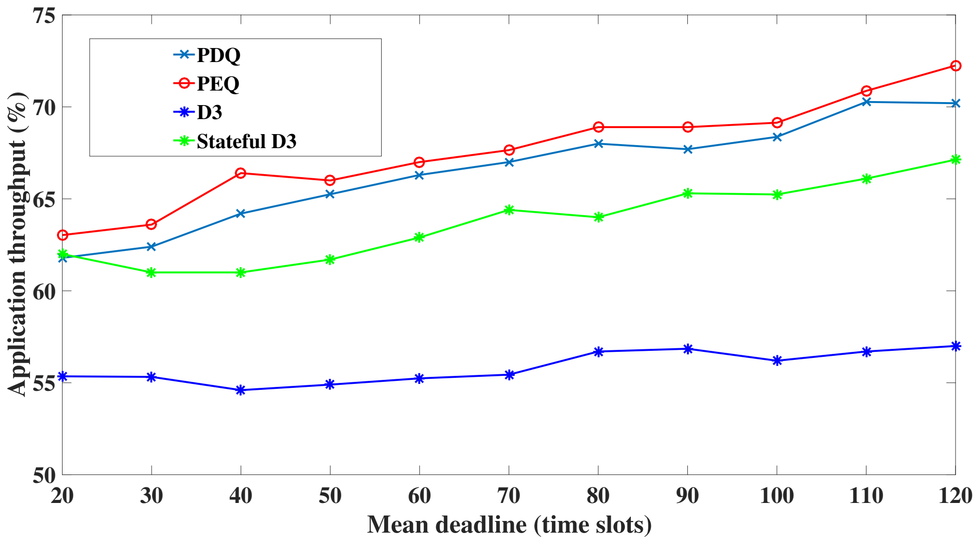


Figure 3.8 Application throughput comparison as a function of flow deadline (100 servers).

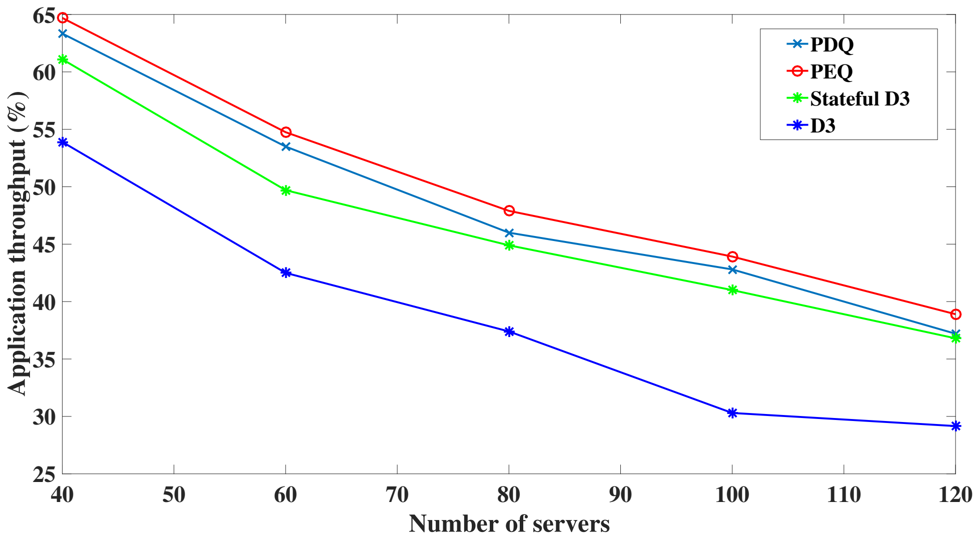


Figure 3.9 Application throughput comparison under overloaded link (tight flows).

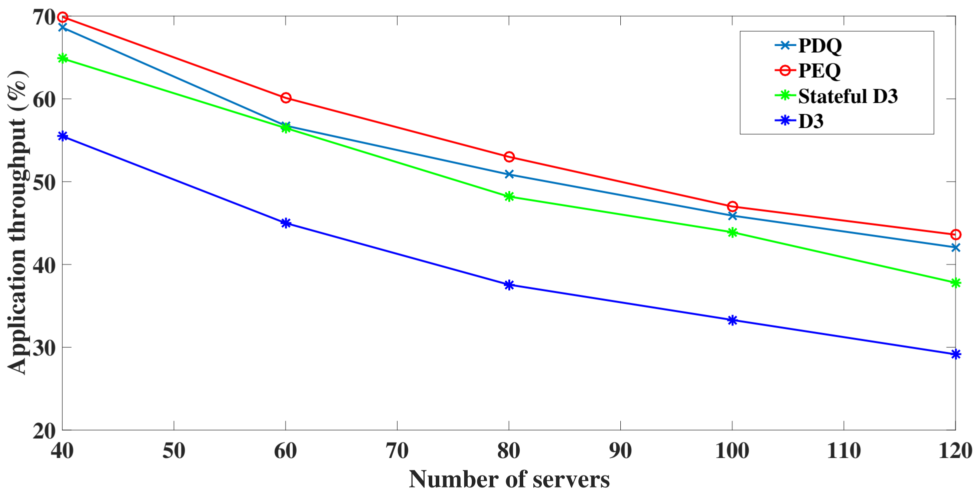


Figure 3.10 Application throughput comparison under overloaded link (moderate flows).

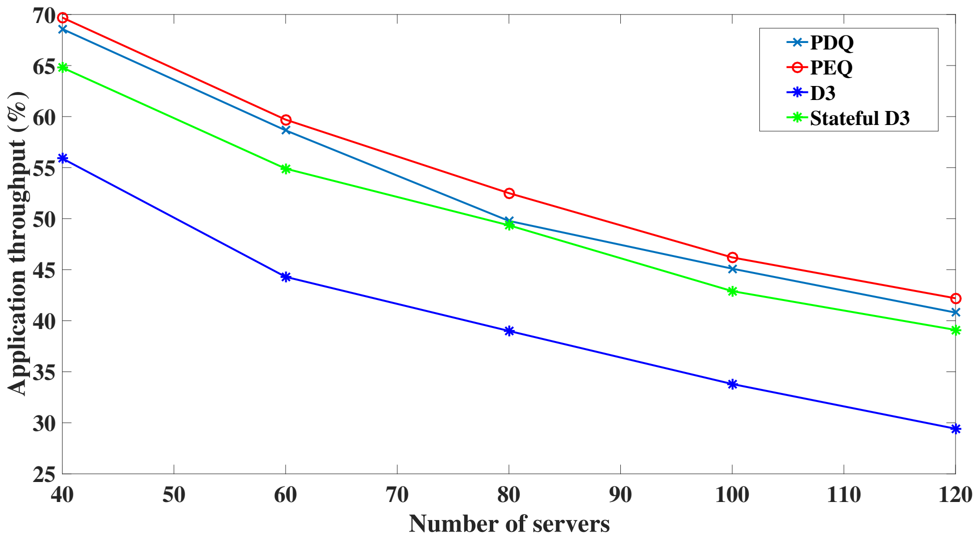


Figure 3.11 Application throughput comparison under overloaded (relaxed flows).

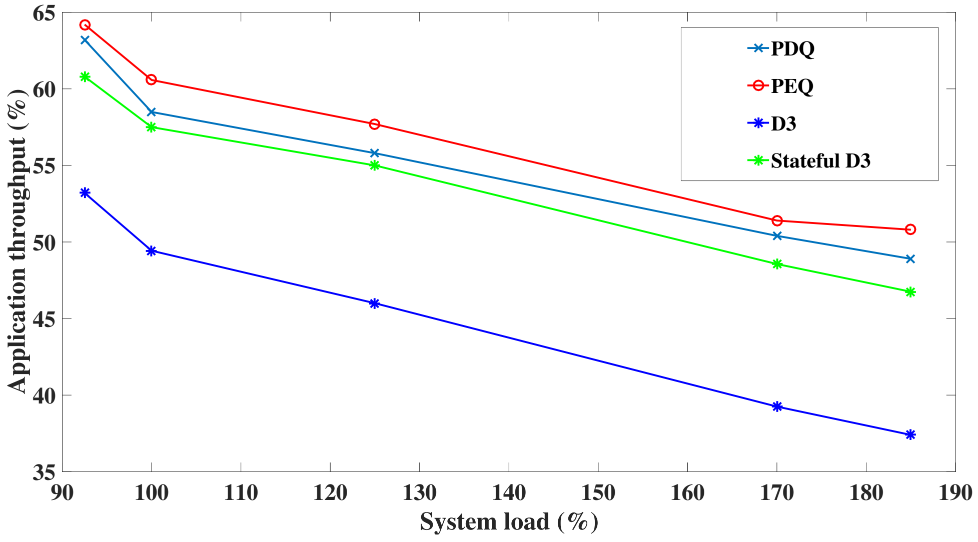


Figure 3.12 Application throughput comparison under overloaded link (tight deadlines).

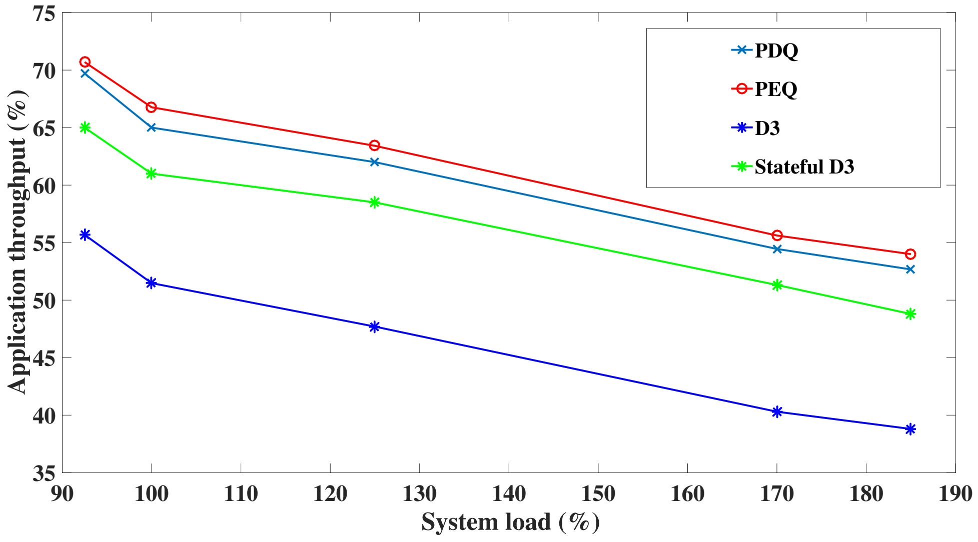


Figure 3.13 Application throughput comparison under overloaded link (moderate deadlines).

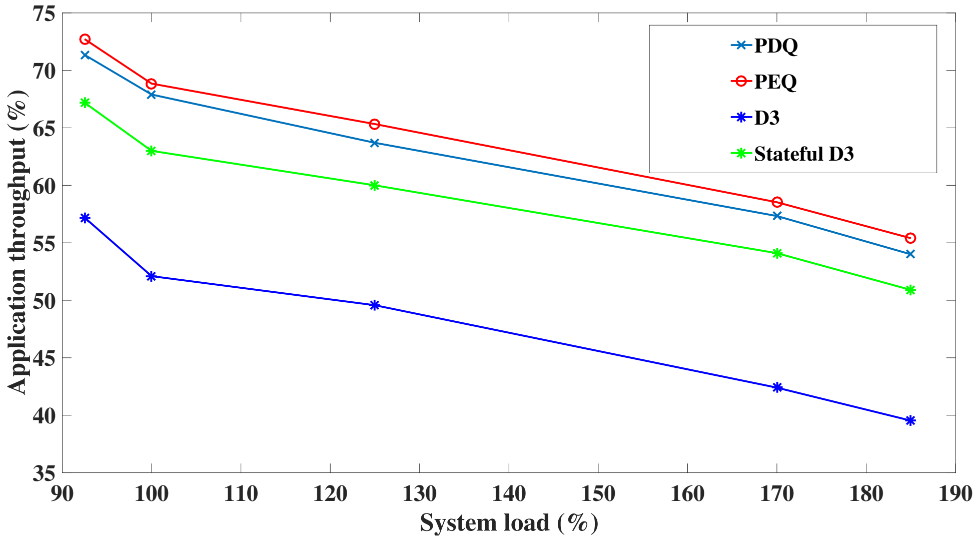


Figure 3.14 Application throughput comparison for overloaded link (relaxed deadlines).

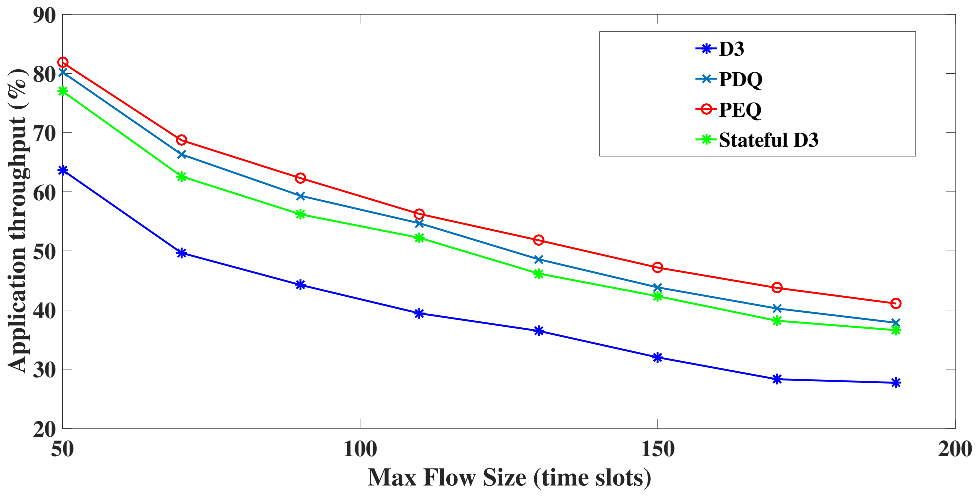


Figure 3.15 Impact of flow size on application throughput (40 servers).

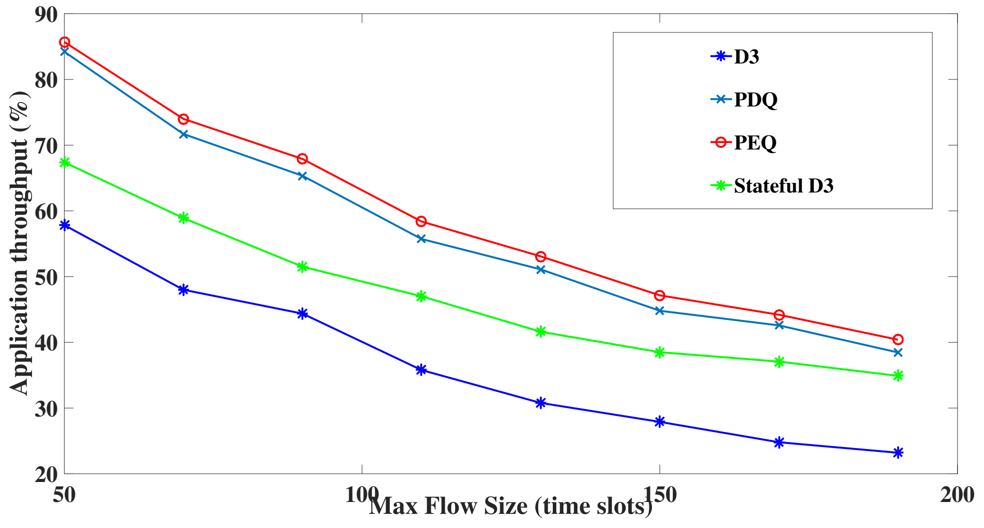


Figure 3.16 Impact of flow size on application throughput (100 servers).

CHAPTER 4

CONCLUSIONS

In this thesis, we introduced a new deadline aware flow scheduling scheme for data centers called PEQ. Unlike other schemes which uses well-known scheduling disciplines like FCFS, EDF, SJF, etc., PEQ uses a new paradigm to schedule flows. This new paradigm is based on a flow's cushion. Cushion is defined as the difference of the flow's remaining deadline and its transmission time or size. A flow's cushion indicates how long a flow can be procrastinated without risking its failure. PEQ procrastinates a flow to accommodate more urgent flows, that is flows with tighter deadlines.

In our simulation the flow sizes are uniformly distributed in the range 2 to 198 times slots and the flow deadlines are exponentially distributed with mean being varied between 20 to 120 time slots depending on the evaluation type. We use the typical partition-aggregate model used in data centers as the test scenario, modeled in software. The number of servers/workers is configurable starting from 20 and up to 180. Each server can have only one active flow in the system at a time. Our main performance parameter is the average application throughput. We show, through various testing conditions, that PEQ outperforms PDQ by about 1 to 2 percent, and D^3 by 15 to 20 percent, on average. We also show that the application throughput of PDQ is about 98.5 percent whereas the application throughput of PEQ is about 99 percent, and therefore approaching to the optimal scheme. We perform various types of testing where we observe the impact of flow deadlines on application throughput and the impact of overloading on application throughput. Our test results show that under these conditions PEQ maintains higher application throughput than other state-of-the-art schemes. Under a specific category of flows, where larger flows have stricter deadlines and smaller flows have bigger deadlines than larger flows, we have shown that PDQ's performance deteriorates while PEQ's application throughput is about twice the application throughput of PDQ. In addition to this, PEQ's flow completion time is smaller than that of PDQ. Our results show that PEQ is able to complete more flows before their deadlines than PDQ, but also it finishes those flows quicker than PDQ.

We have also proposed a comparable and improved version of D^3 and call it Statefull D^3. We also show that Statefull D^3 actually boosts the application throughput of the scheme by about 15 to 20 percent of the original D^3 scheme.

REFERENCES

[1] Normal distribution, 2014. Last Access: 5/2/2017.

[2] Hussam Abu-Libdeh, Paolo Costa, Antony Rowstron, Greg O'Shea, and Austin Donnelly. Symbiotic routing in future data centers. *ACM SIGCOMM Computer Communication Review*, 40(4):51–62, 2010.

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

[4] Mohammad Alizadeh, Albert Greenberg, David A Maltz, Jitendra Padhye, Parveen Patel, Balaji Prabhakar, Sudipta Sengupta, and Murari Sridharan. Data center tcp (dctcp). In *ACM SIGCOMM computer communication review*, volume 40, pages 63–74. ACM, 2010.

[5] Mohammad Alizadeh, Abdul Kabbani, Tom Edsall, Balaji Prabhakar, Amin Vahdat, and Masato Yasuda. Less is more: trading a little bandwidth for ultra-low latency in the data center. In *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*, pages 19–19. USENIX Association, 2012.

[6] Theophilus Benson, Ashok Anand, Aditya Akella, and Ming Zhang. Understanding data center traffic characteristics. *ACM SIGCOMM Computer Communication Review*, 40(1):92–99, 2010.

[7] Kashif Bilal, Samee U Khan, Limin Zhang, Hongxiang Li, Khizar Hayat, Sajjad A Madani, Nasro Min-Allah, Lizhe Wang, Dan Chen, Majid Iqbal, et al. Quantitative comparisons of the state-of-the-art data center architectures. *Concurrency and Computation: Practice and Experience*, 25(12):1771–1783, 2013.

[8] Kashif Bilal, Saif Ur Rehman Malik, Osman Khalid, Abdul Hameed, Enrique Alvarez, Vidura Wijaysekara, Rizwana Irfan, Sarjan Shrestha, Debjyoti Dwivedy, Mazhar Ali, et al. A taxonomy and survey on green data center networks. *Future Generation Computer Systems*, 36:189–208, 2014.

[9] Edward G Coffman Jr, Richard R Muntz, and H Trotter. Waiting time distributions for processor-sharing systems. *Journal of the ACM (JACM)*, 17(1):123–130, 1970.

[10] P Costa, A Donnelly, G O'shea, and A Rowstron. Camcube: a key-based data center. *Technical Report MSR TR-2010-74, Microsoft Research*, 2010.

[11] Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall, and Werner Vogels. Dynamo: amazon's highly available key-value store. *ACM SIGOPS operating systems review*, 41(6):205–220, 2007.

[12] Nandita Dukkipati and Nick McKeown. Why flow-completion time is the right metric for congestion control. *ACM SIGCOMM Computer Communication Review*, 36(1):59–62, 2006.

[13] Nathan Farrington, Erik Rubow, and Amin Vahdat. Data center switch architecture in the age of merchant silicon. In *High Performance Interconnects, 2009. HOTI 2009. 17th IEEE Symposium on*, pages 93–102. IEEE, 2009.

[14] Albert Greenberg, James R Hamilton, Navendu Jain, Srikanth Kandula, Changhoon Kim, Parantap Lahiri, David A Maltz, Parveen Patel, and Sudipta Sengupta. Vl2: a scalable and flexible data center network. In *ACM SIGCOMM computer communication review*, volume 39, pages 51–62. ACM, 2009.

[15] Chuanxiong Guo, Guohan Lu, Dan Li, Haitao Wu, Xuan Zhang, Yunfeng Shi, Chen Tian, Yongguang Zhang, and Songwu Lu. Bcube: a high performance, server-centric network architecture for modular data centers. *ACM SIGCOMM Computer Communication Review*, 39(4):63–74, 2009.

[16] Chuanxiong Guo, Haitao Wu, Kun Tan, Lei Shi, Yongguang Zhang, and Songwu Lu. Dcell: a scalable and fault-tolerant network structure for data centers. In *ACM SIGCOMM Computer Communication Review*, volume 38, pages 75–86. ACM, 2008.

[17] Ali Hammadi and Lotfi Mhamdi. A survey on architectures and energy efficiency in data center networks. *Computer Communications*, 40:1–21, 2014.

[18] Americas Headquarters. Cisco data center infrastructure 2.5 design guide. In *Cisco Validated Design I*. Cisco Systems, Inc, 2007.

[19] Todd Hoff. 10 ebay secrets for planet wide scaling. *High Scalability-*, 2009.

[20] Todd Hoff. Latency is everywhere and it costs you sales-how to crush it. *High Scalability, July*, 25, 2009.

[21] Chi-Yao Hong, Matthew Caesar, and P Godfrey. Finishing flows quickly with preemptive scheduling. *ACM SIGCOMM Computer Communication Review*, 42(4):127–138, 2012.

[22] Carlee Joe-Wong, Soumya Sen, Tian Lan, and Mung Chiang. Multiresource allocation: Fairness-efficiency tradeoffs in a unifying framework. *IEEE/ACM Transactions on Networking (TON)*, 21(6):1785–1798, 2013.

[23] Srikanth Kandula, Sudipta Sengupta, Albert Greenberg, Parveen Patel, and Ronnie Chaiken. The nature of data center traffic: measurements & analysis. In *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference*, pages 202–208. ACM, 2009.

[24] Ali Munir, Ihsan A Qazi, Zartash A Uzmi, Aisha Mushtaq, Saad N Ismail, M Safdar Iqbal, and Basma Khan. Minimizing flow completion times in data centers. In *INFOCOM, 2013 Proceedings IEEE*, pages 2157–2165. IEEE, 2013.

[25] Ali Munir, Ihsan Ayyub Qazi, and Saad Bin Qaisar. On achieving low latency in data centers. In *Communications (ICC), 2013 IEEE International Conference on*, pages 3721–3725. IEEE, 2013.

[26] Amar Phanishayee, Elie Krevat, Vijay Vasudevan, David G Andersen, Gregory R Ganger, Garth A Gibson, and Srinivasan Seshan. Measurement and analysis of tcp throughput collapse in cluster-based storage systems. In *FAST*, volume 8, pages 1–14, 2008.

[27] Lucian Popa, Gautam Kumar, Mosharaf Chowdhury, Arvind Krishnamurthy, Sylvia Ratnasamy, and Ion Stoica. Faircloud: sharing the network in cloud computing. In *Proceedings of the ACM SIGCOMM 2012 conference on Applications, technologies, architectures, and protocols for computer communication*, pages 187–198. ACM, 2012.

[28] Jarno Rajahalme, Shane Amante, Sheng Jiang, and Brian Carpenter. Ipv6 flow label specification. 2011.

[29] Roberto Rojas-Cessa, Yagiz Kaymak, and Ziqian Dong. Schemes for fast transmission of flows in data center networks. *IEEE Communications Surveys & Tutorials*, 17(3):1391–1422, 2015.

[30] Linus Schrage. Letter to the editor—a proof of the optimality of the shortest remaining processing time discipline. *Operations Research*, 16(3):687–690, 1968.

[31] Rohit P Tahiliani, Mohit P Tahiliani, and K Chandra Sekaran. Tcp variants for data center networks: a comparative study. In *Cloud and Services Computing (ISCOS), 2012 International Symposium on*, pages 57–62. IEEE, 2012.

[32] Balajee Vamanan, Jahangir Hasan, and TN Vijaykumar. Deadline-aware datacenter tcp (d2tcp). *ACM SIGCOMM Computer Communication Review*, 42(4):115–126, 2012.

[33] W Vogels. Performance and scalability, 2009.

[34] Ting Wang, Zhiyang Su, Yu Xia, and Mounir Hamdi. Rethinking the data center networking: Architecture, network protocols, and resource sharing. *IEEE access*, 2:1481–1496, 2014.

[35] Christo Wilson, Hitesh Ballani, Thomas Karagiannis, and Ant Rowtron. Better never than late: Meeting deadlines in datacenter networks. In *ACM SIGCOMM Computer Communication Review*, volume 41, pages 50–61. ACM, 2011.

[36] Hong Xu and Baochun Li. Repflow: Minimizing flow completion times with replicated flows in data centers. In *INFOCOM, 2014 Proceedings IEEE*, pages 1581–1589. IEEE, 2014.

[37] David Zats, Tathagata Das, Prashanth Mohan, Dhruba Borthakur, and Randy Katz. Detail: reducing the flow completion time tail in datacenter networks. *ACM SIGCOMM Computer Communication Review*, 42(4):139–150, 2012.

[38] Jiao Zhang, Fengyuan Ren, and Chuang Lin. Modeling and understanding tcp incast in data center networks. In *INFOCOM, 2011 Proceedings IEEE*, pages 1377–1385. IEEE, 2011.

[39] Yan Zhang and Nirwan Ansari. On mitigating tcp incast in data center networks. In *INFOCOM, 2011 Proceedings IEEE*, pages 51–55. IEEE, 2011.

[40] Yan Zhang and Nirwan Ansari. On architecture design, congestion notification, tcp incast and power consumption in data centers. *IEEE Communications Surveys & Tutorials*, 15(1):39–64, 2013.