Information-Agnostic Flow Scheduling for Commodity Data Centers

Wei Bai, Li Chen, Kai Chen,
Dongsu Han, Chen Tian, Hao Wang

NSDI 2015

The paper is from my previous lab: iSINGLab @ HKUST!
The 1st NSDI paper in Hong Kong!

Outline

- **Background (Not in the paper…)**
- PIAS
- Implementation and evaluation
- Review

Why Flow Scheduling Matters

- Long-tail distribution again!
- Many small flows
- Few large flows consume most bandwidth
  - Small flows prefer low latency
  - Large flows require high throughput

Flow Completion Time (FCT)

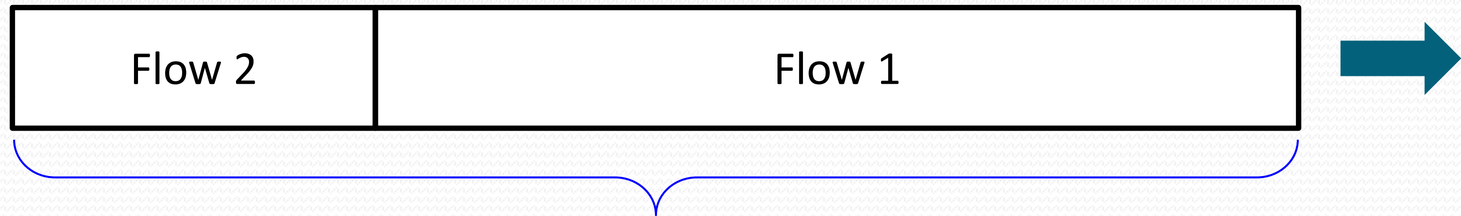
- A flow contains many packets (Same 5-tuple)
  - Src/dest address
  - Src/dest port
  - Protocol
- Large flows have many packets
- Small flows have few packets
- FCT = the arrive time of the last packet – the send time of the first packet

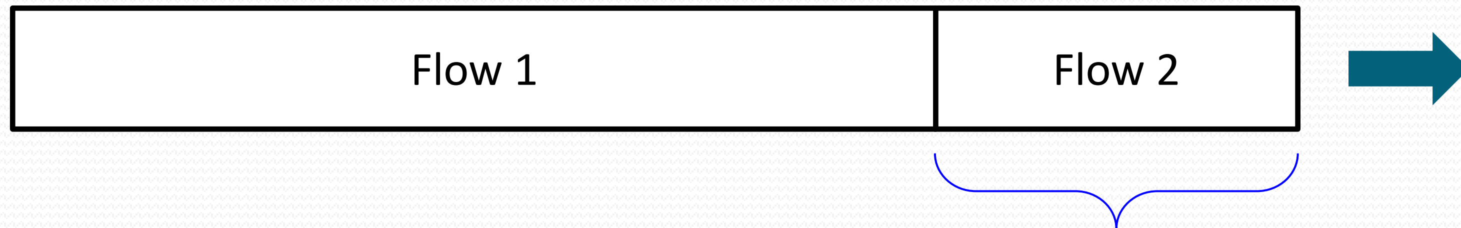
Flow Completion Time (FCT)

- FCT can be used as a uniform metric for both flows
  - Large flows: throughput
  - Small flows: latency

Short Flows Blocked by Long Flows

- An example - **Queue on switch**

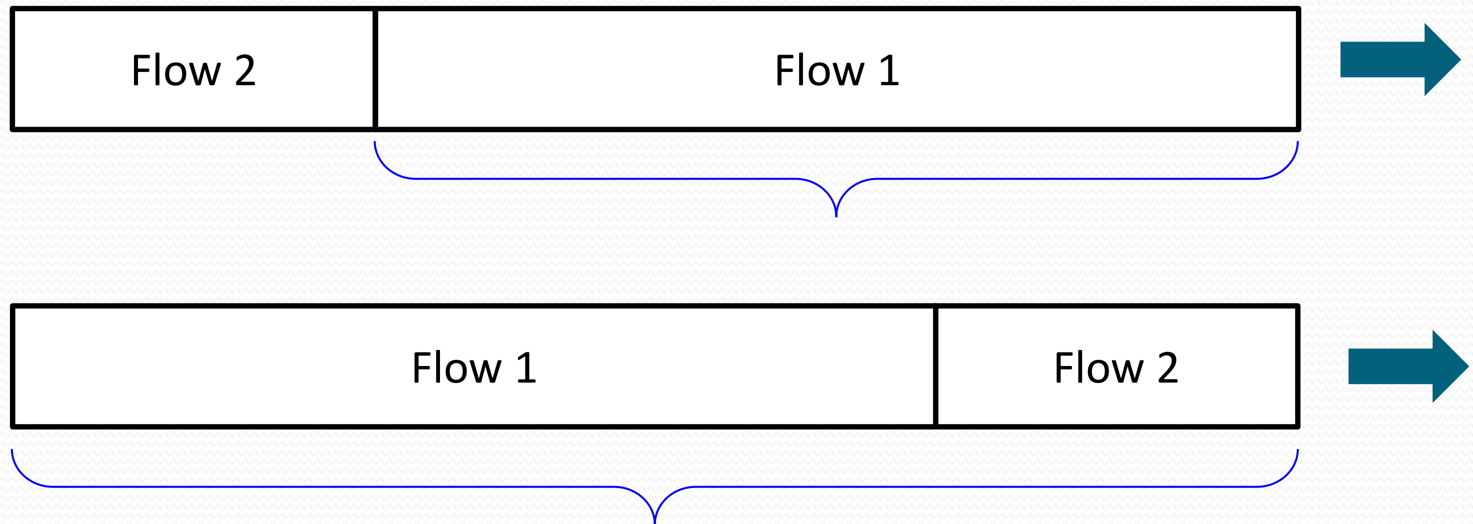




If Flow 2 is sent first, then it will not be blocked by Flow 1

Short Flows Blocked by Long Flows

- An example - **Queue on switch**



Does not impact the performance of large flow 1 as flow 2 is small

Short Flows Blocked by Long Flows

- An example - **Queue on switch**
  - Short flows are blocked by large flows
  - Similar to **Convoy effect** in task scheduling in OS

	OS	Network
Schedule Target	Job/Task	Flow
Information	Job size/duration	Flow size
What to Schedule	CPU time	Link bandwidth
Metric	JCT	FCT

Hey, Let’s First Recap Task Scheduling

Only need to know the arrival order

FCFS

First-Come, First-Served (FCFS) Scheduling

Process	Burst Time
P_1	24
P_2	3
P_3	3

- Suppose that the processes arrive in the order: P_1 , P_2 , P_3
The Gantt Chart for the schedule is:

- Waiting time for P_1 = 0; P_2 = 24; P_3 = 27
- Average waiting time: (0 + 24 + 27)/3 = 17
- **Average turn-around time: (24+27+30)/3 = 27**
- Turn around time = Completion Time−Arrival Time
- **Turn-around time is similar to FCT**

FCFS Scheduling (Cont.)

Suppose that the processes arrive in the order:

$$P_2 , P_3 , P_1$$

The Gantt chart for the schedule is:

Waiting time for $P_1 = 6; P_2 = 0, P_3 = 3$

Average waiting time: (6 + 0 + 3)/3 = 3 (much shorter!)

Average turn-around time: (30+3+6)/3 = **13**

Convoy effect – A short process stuck behind a long process, bad for short jobs, depending purely on the arrival order

- Consider one CPU-bound and many I/O-bound processes, FCFS results also in low I/O device utilization
- Waiting in banks: depositing a check, stuck behind new account opening

Knowledge Map

Only need to know the arrival order

Long tail distribution → **Convoy Effect**

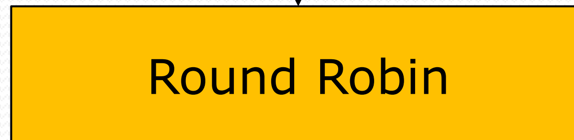
Makes it worse

Knowledge Map

Only need to know the arrival order

Convoy Effect

Intuition: Reduce the time a long process can run



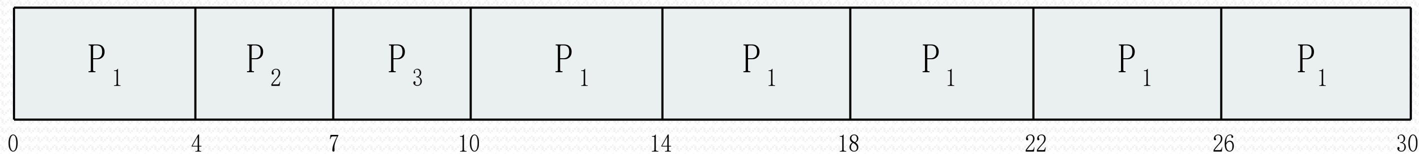
Round Robin (RR)

- The FCFS scheduling algorithm is non-preemptive. Once the CPU core is allocated to a process, the process keeps the CPU until it releases CPU
- The FCFS is thus particularly troublesome for interactive systems, where it is important that each process get a share of the CPU at regular intervals.
- The round-robin (RR) scheduling algorithm is similar to FCFS scheduling, but preemption is added to enable system to switch between processes
- Each process gets a small unit of CPU time (time quantum q), usually 10-100 milliseconds. After this time has elapsed, the process is preempted and added to the end of the ready queue (exactly like FCFS).
- Given n processes, each process gets $1/n$ of the CPU time in chunks of at most q time units at once – context switching time is ignored
  - No process waits more than $(n-1)q$ time units.
- A timer interrupts every quantum to schedule next process, or the process blocks upon completing its current CPU burst time when its CPU burst time or remaining CPU burst time < q

Example of RR with Time Quantum = 4

Process	Burst Time
P_1	24
P_2	3
P_3	3

- The Gantt chart is:



Waiting time for P1=6, P2=4,P3=7

- Average waiting time (6+4+7)/3 = 5.67
- Average turn-around time: (30+7+10)/3 = **15.67**
- **Response time** for P1=4, P2=7, P3=10, average = **7**
- The average waiting time under RR policy can be long, but is inherently more “fair” (FIFO order), usually perform better for short jobs than FCFS, and offers better average response time – important for interactive jobs

Time Quantum and Context Switch Time

The performance of the RR algorithm depends heavily on the size of the time quantum

q large ⇒ FCFS

q small ⇒ interleaved, but **q** must be large with respect to context switch time (usually < 10 usec), otherwise overhead is too high

process time = 10	quantum	context switches
0 – 10	12	0
0 – 6 – 10	6	1
0 1 2 3 4 5 6 7 8 9 10	1	9

Comparisons between FCFS and RR

Assuming zero-cost context-switching time, is RR always better than FCFS?

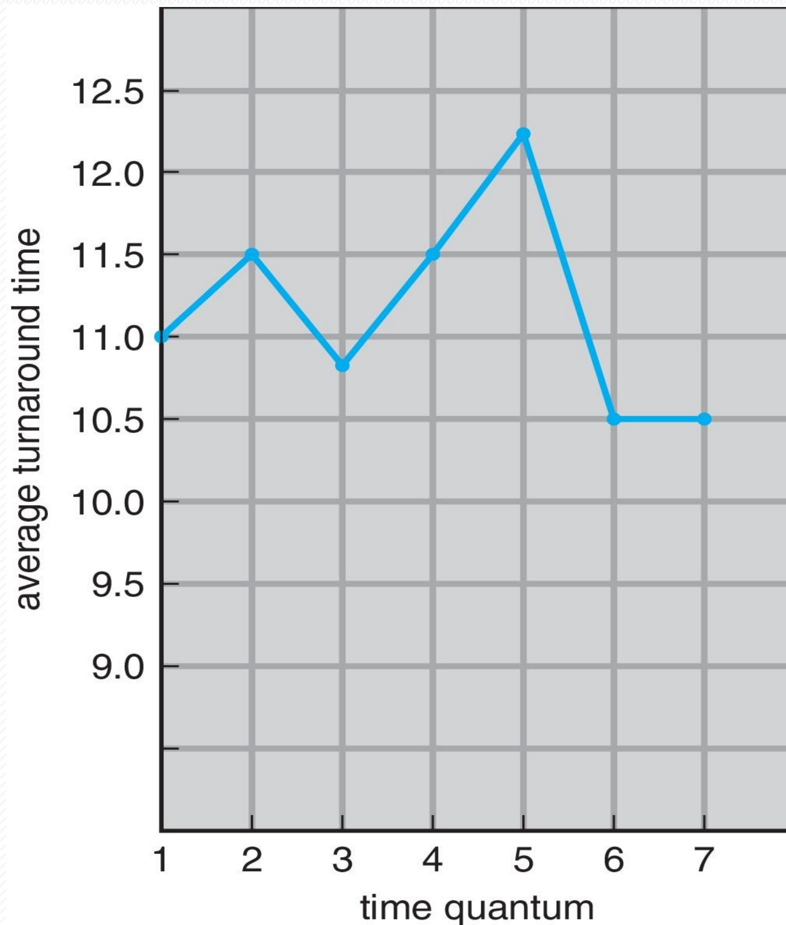
An example: 10 jobs starting at the same time, each taking 100s of CPU time; RR scheduler quantum of 1s;

Job #	FIFO	RR
1	100	991
2	200	992
…	…	…
9	900	999
10	1000	1000

The average job turn-around time is much worse under RR!

- Bad when all jobs have the same length
- The average response time is much better

Turnaround Time Varies With The Time Quantum



process	time
P_1	6
P_2	3
P_3	1
P_4	7

The average turnaround time does not necessarily improve when the quantum increases

In general, the average turnaround time can be improved if most processes finish their `current` CPU bursts within a single quantum

The time quantum can not be too big, in which RR degenerates to an FCFS policy

A rule of thumb: 80% CPU bursts should be shorter than the time quantum **`q`**

Knowledge Map

Only need to know the arrival order

FCFS

A large Q

Convoy Effect

Intuition: Reduce the time a long process can run

Round Robin

Tradeoff between responsiveness and turnaround time
Bad performance when all processes have similar lengths

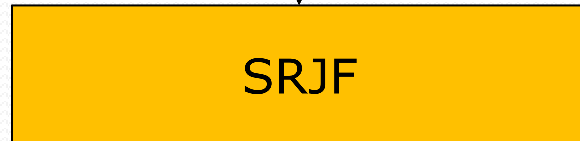
Knowledge Map

Only need to know the arrival order

Know the order and the burst time

The duration of a process

Preemptive



Shortest-Job-First (SJF) Scheduling

Noticing that in FCFS and RR, we do not need to know the next CPU burst time of each process during scheduling, and scheduling is done solely based on the arrival order to the ready queue

What if we knew the future – the next CPU burst time of each process

Associate with each process the length of its next CPU burst

- To schedule the process with the shortest next CPU burst

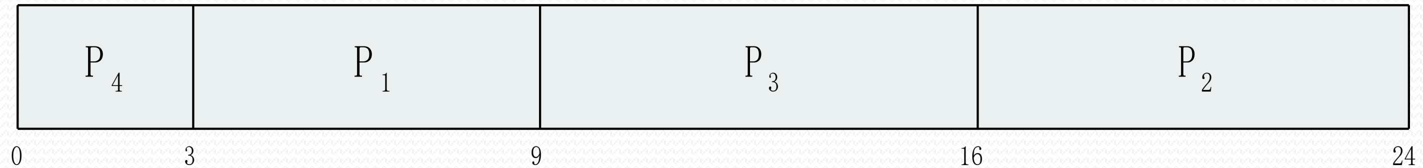
The Shortest Job First or SJF scheduling algorithm is optimal – it produces the minimum average waiting time for a given set of processes

- The difficulty is knowing the length of the next CPU request
- The basic idea is to get the short jobs out of the system sooner
- Big effect on short jobs, relatively small effect on long jobs
- This can be applied to an entire program, or the current CPU burst
- Perhaps a more precise term should be the shortest-next-CPU-burst algorithm, but shortest job first or SJF is commonly used.

Example of SJF

Process	Burst Time
P_1	6
P_2	8
P_3	7
P_4	3

SJF scheduling chart



Average waiting time = (3 + 16 + 9 + 0) / 4 = 7

The “best” FCFS performs the same if arrival order happens to be the same

Sketch Proof: Moving a short process before a long one decreases the waiting time of the short process more than it increases the waiting time of the long process. Consequently, the average waiting time decreases

	T1	T2	

	T2	T1	

Example of Shortest-Remaining-Time-First

The SJF algorithm can be either preemptive or nonpreemptive. The choice arises when a new process arrives at the ready queue while another process is still executing

- Preemptive version called shortest-remaining-time-first（SRTF）

Process	Arrival Time	Burst Time
P_1	0	8
P_2	1	4
P_3	2	9
P_4	3	5

Preemptive SJF Gantt Chart

Average waiting time = [(10-1)+(1-1)+(17-2)+5-3)]/4 = 26/4 = **6.5**

Now scheduling needs to be invoked when there is an arrival to the ready queue (scheduling condition 4)

Comparison of SJF/SRTF and FCFS

- SJF/SRTF are the best we can do towards minimizing the average waiting time. or minimizing the average turnaround time
  - Provably optimal (SJF among non-preemptive, SRTF among preemptive)
  - SRTF is always at least as good as SJF
- SJF/SRTF performs the same as FCFS if all processes have the same CPU burst times
- SJF/SRTF can possibly lead to starvation for long process if there is always shorter process joining the ready queue
  - “fairness” can not be enforced

Knowledge Map

Only need to know the arrival order

Know the order and the burst time

How to know the burst time?

Predict

Approximate

FCFS

Round Robin

SJF

SRJF

Knowledge Map

Only need to know the arrival order

Know the order and the burst time

How to know the burst time?

Predict

Approximate SJF

FCFS

Round Robin

SJF

SRJF

MLFQ

Knowledge Map

Only need to know the arrival order

Know the order and the burst time

How to know the burst time?

Predict

Approximate SJF

FCFS

Round Robin

SJF

SRJF

Preliminary Concepts

Priority

MQ

MLFQ

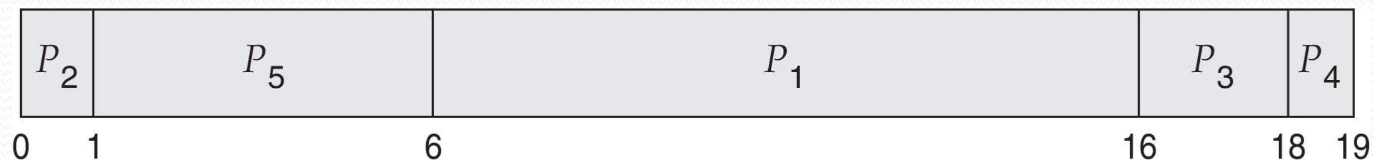
Priority Scheduling

- A priority number (e.g., integer) is associated with each process
- The CPU is allocated to the process with the highest priority (smallest integer ≡ highest priority), it can be
  - Preemptive （upon new arrival of a higher priority process)
  - Nonpreemptive
- Equal-priority processes are scheduled in FCFS order
- SJF is a special case of the general priority-scheduling algorithm, where priority is the inverse of predicted next CPU burst time
- Problem ≡ Starvation – low priority processes may never execute
- Solution ≡ Aging – as time progresses increase the priority of the process

Example of Priority Scheduling

Process	Burst Time	Priority
P_1	10	3
P_2	1	1
P_3	2	4
P_4	1	5
P_5	5	2

- Priority scheduling Gantt Chart

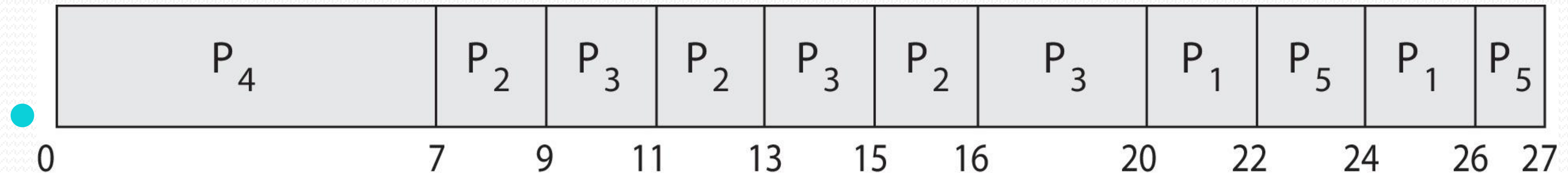


- Average waiting time = 8.2 msec

Priority Scheduling w/ Round-Robin

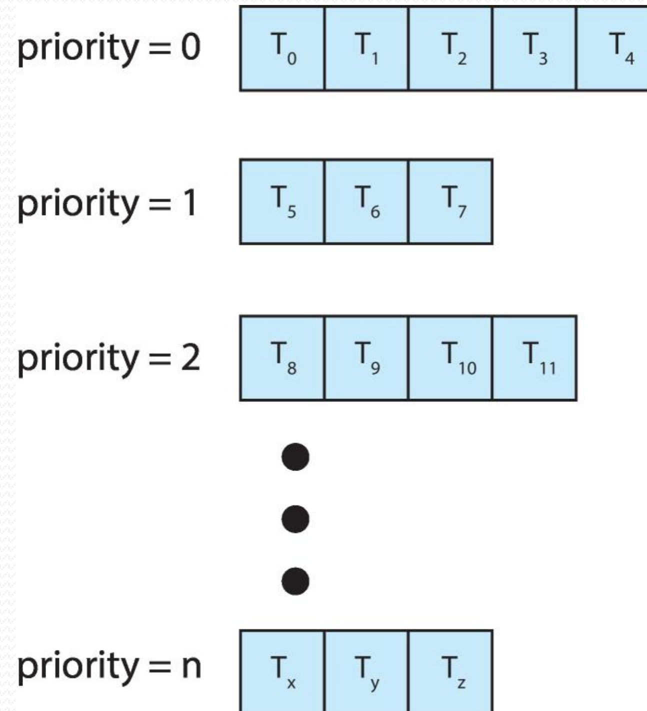
Process	Burst Time	Priority
P_1	4	3
P_2	5	2
P_3	8	2
P_4	7	1
P_5	3	3

- Run the process with the highest priority. Processes with the same priority run round-robin



Multilevel Queue

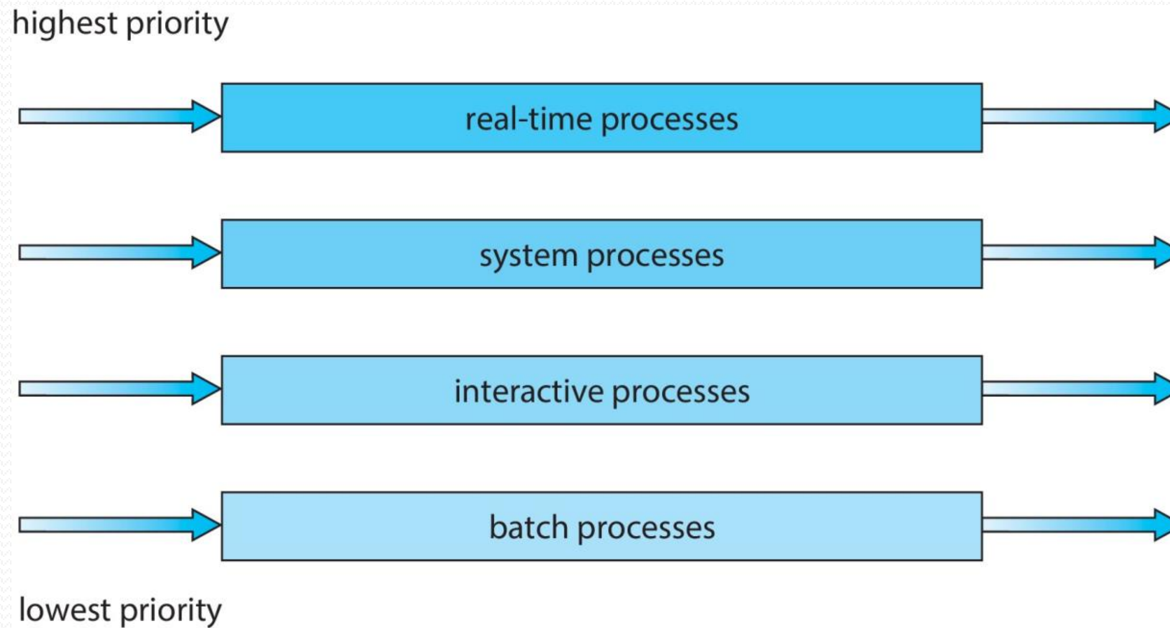
- Multilevel queue scheduling can still be a priority scheduling combined with round-robin
- A priority is assigned statically to each process, and a process remains in the same queue for the duration of its runtime



Multilevel Queue (Cont.)

Partition processes into different queues based on process type

- Each queue can have its own scheduling algorithm based on the needs
- The scheduling among the queues, is commonly implemented as fixed-priority preemptive scheduling or each queue gets certain amount of CPU time – time-slice (for instance 60%, 20%, 10%, 10%)



Multilevel Feedback Queue (MLFQ)

- A process can move between the various queues; aging can be implemented this way. This provides the flexibility
- Multilevel-feedback-queue or MLFQ scheduler defined by the following parameters:
  - The number of queues
  - scheduling algorithms for each queue
  - method used to determine when to upgrade a process
  - method used to determine when to demote a process
  - method used to determine which queue a process will enter when that process needs service

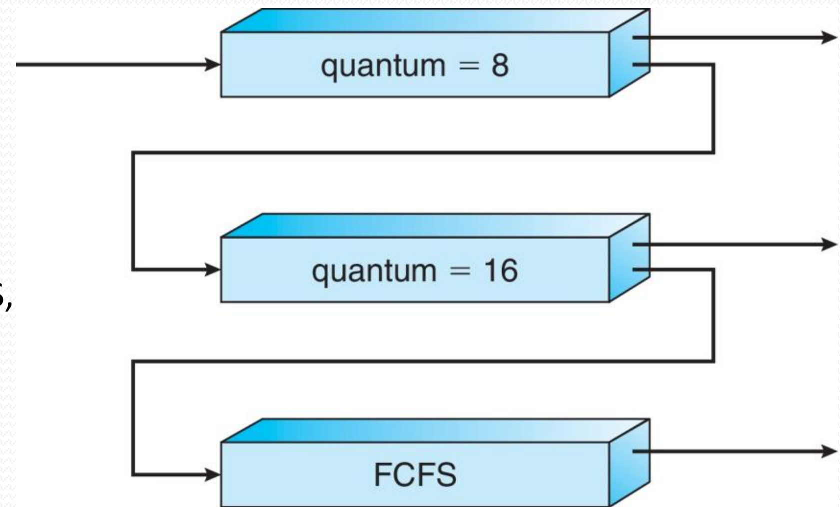
Example of Multilevel Feedback Queue

Three queues:

- Q_0 – RR with time quantum 8 milliseconds
- Q_1 – RR time quantum 16 milliseconds
- Q_2 – FCFS

Scheduling

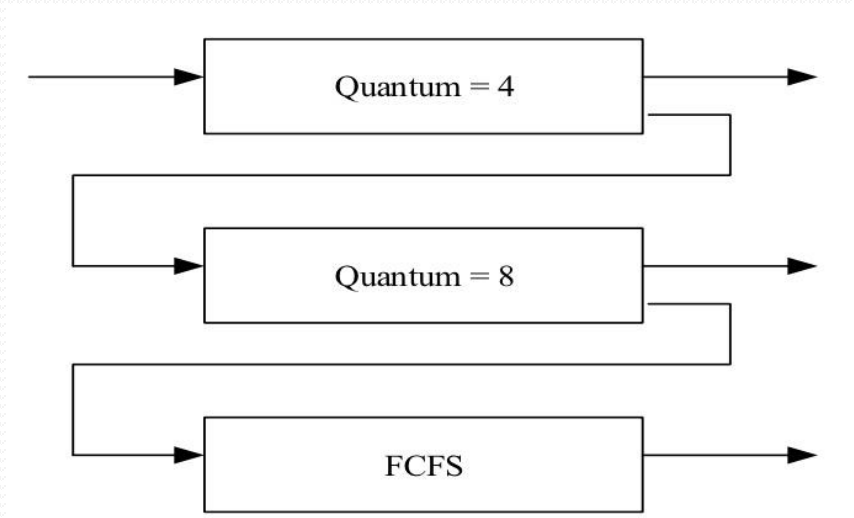
- A new job enters queue Q_0 which is served FCFS, also preempts jobs from Q1 or Q2 if currently running on CPU
  - When it gains CPU, job receives 8 ms
  - If it does not finish in 8 milliseconds, job is moved to the queue Q_1
- At Q_1 job is again served FCFS and receives 16 additional milliseconds
  - If it still does not complete, it is preempted and moved to queue Q_2
- If a job from Q_1 or Q_2 is preempted by a new job from Q_0, it joins the head of the queue Q_1 or Q_2 , respectively



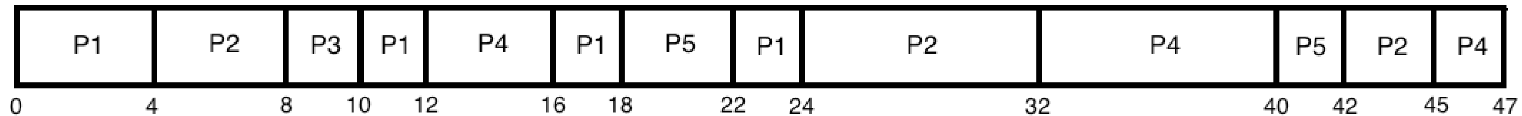
This approximates SRTF:

- CPU bound jobs drop like a rock
- Short-running I/O bound jobs stay near top

MLFQ Example

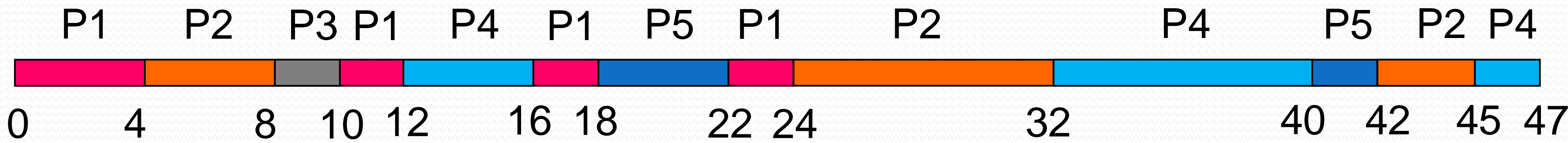


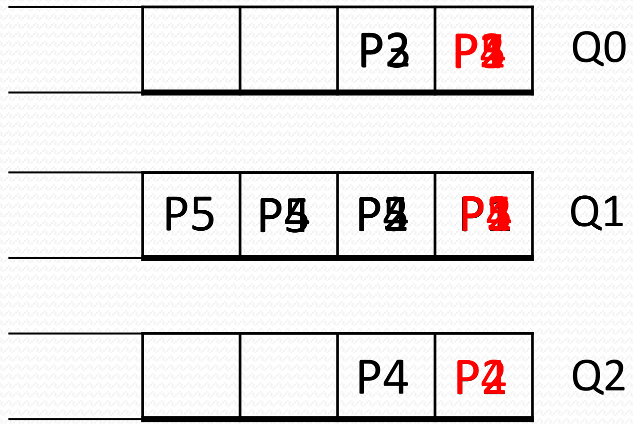
Process	Arrival Time (ms)	Burst Time (ms)
P1	0	10
P2	2	15
P3	5	2
P4	12	14
P5	18	6



MLFQ Scheduling: Example

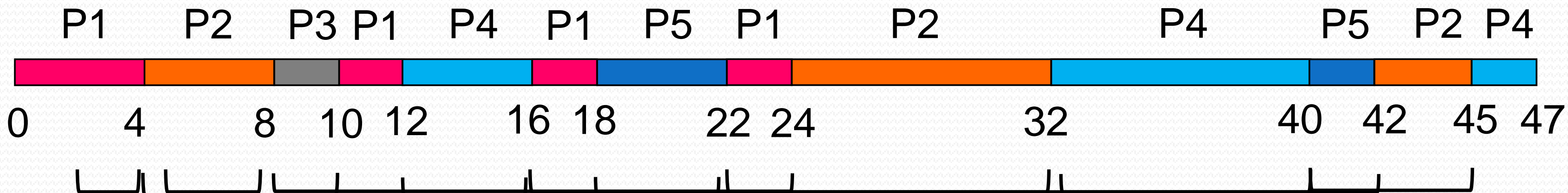
	Process	Burst Time	Arrival Time	Remaining Time
√	P1	10	0	0
√	P2	15	2	0
√	P3	2	5	0
√	P4	14	12	0
√	P5	6	18	0





MLFQ Scheduling: Example

Process	Burst Time	Arrival Time	Remaining Time
P1	10	0	0
P2	15	2	0
P3	2	5	0
P4	14	12	0
P5	6	18	0



2 5

Waiting time for P1=14, P2=28, P3=3, P4=21, P5=18

P1=6+4+4=14

Average waiting time: (14+28+3+21+18)/5=16.8

Knowledge Map

Only need to know the arrival order

Know the order and the burst time

How to know the burst time?

Predict

Approximate SJF

FCFS

Round Robin

SJF

SRJF

Preliminary Concepts

Priority

MQ

MLFQ

Let's Come Back to Advanced Computer Networking

- Can we use FCFS?
  - No! **Convoy effect….**

Let's Come Back to Advanced Computer Networking

- Can we use FCFS?
  - No! **Convoy effect…**
- Can we use RR?
  - No! RR is for **fairness**...
  - RR does not deliver ideal **FCT…**
  - Switch does not allow **flexible time quantum**

Let's Come Back to Advanced Computer Networking

- Can we use SJF/SRJF
  - **Maybe, but it relies on flow size prediction**
  - Mohammad Alizadeh (Big M AGAIN!!!), Shuang Yang, Milad Sharif, Sachin Katti, Nick McKeown, Balaji Prabhakar, Scott Shenker. pFabric: Minimal Near-Optimal Datacenter Transport. SIGCOMM 2013
  - Prediction leads to **inaccuracy** and deployment **complexity**.

Let's Come Back to Advanced Computer Networking

- Can we use MLFQ?
  - **Good idea!!**
  - **But commodity switches do not support…**

What Can a Commodity Switch Offer?

- Multiple Hardware Priority Queues
    - Priority bit in IP Header (DSCP)
    - Host sets the DSCP
    - Switch does the mapping
- Limited Scheduling Support:
  - Strict Priority Scheduling
  - Weighted Round Robin
- No Flow-level Information

Summary for Ideal Flow Scheduling in DCN

- What do we actually need:
  - **FCT minimization.**
  - **Information-agnostic:** do not assume flow sizes are known in advance or predict flow size
  - **Practical:** can be deployed with commodity switches

The answer is **PIAS**: **P**ractical **I**nformation-**A**gnostic flow **S**cheduling

Outline

- Background
- **PIAS**
- Implementation and evaluation
- Review

Overview

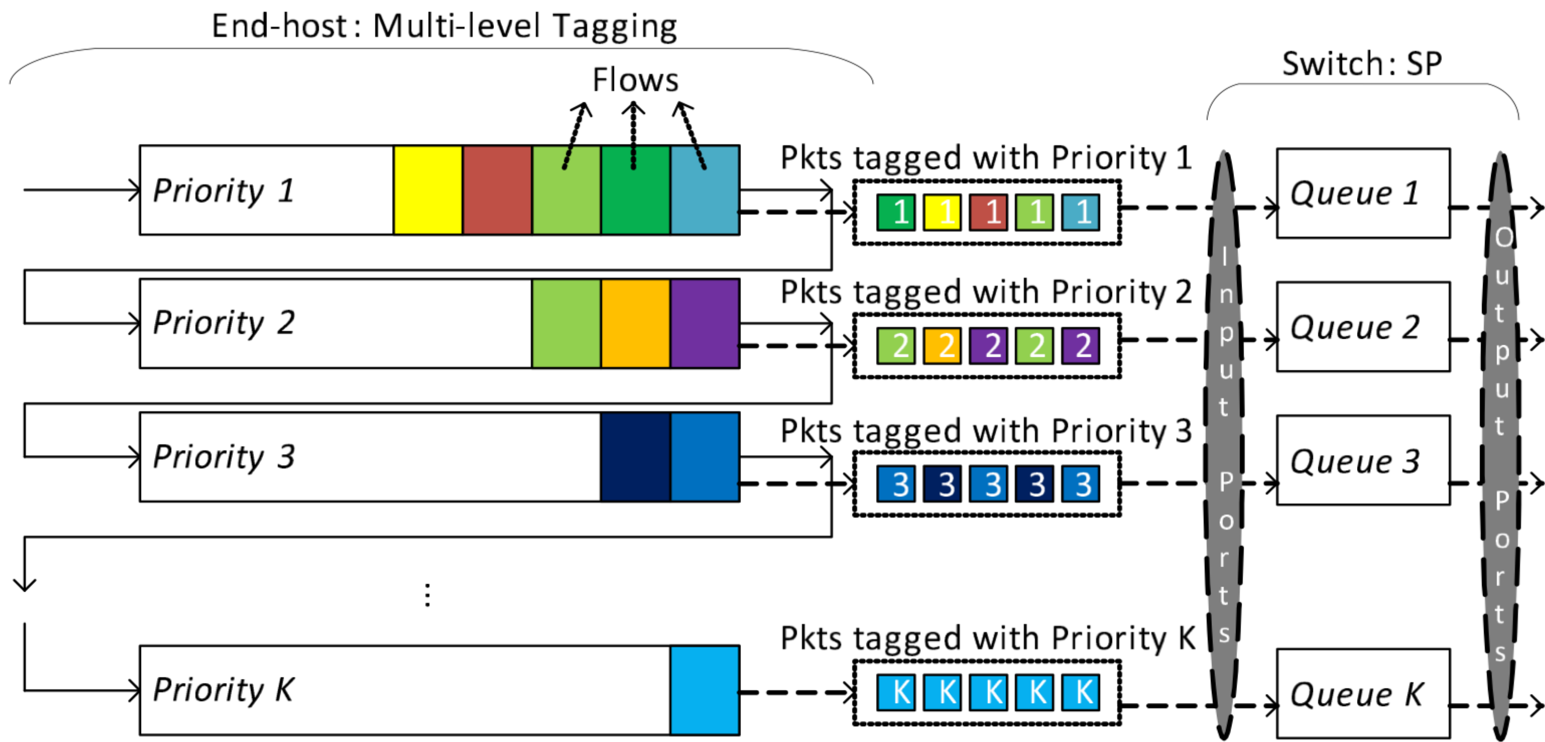


Figure 1: PIAS overview

Core Idea of PIAS

- **Still use MLFQ** (Of course, we spent so much time introducing it)
- But PIAS does not target implementing MLFQ directly on switch
  - Packet tagging at the endhost – implement MLFQ using software
  - Strict priority queue at switch – strict priority based on the priority tags

Simple Example Illustrating PIAS

Congestion

Simple Example Illustrating PIAS

Flow 1 with 10 packets and flow 2 with 2 packets arrive

Priority 1

Priority 2

Priority 3

Priority 4

Simple Example Illustrating PIAS

Flow 1 and 2 transmit simultaneously

Priority 1

Priority 2

Priority 3

Priority 4

Simple Example Illustrating PIAS

Flow 2 finishes while flow 1 is demoted to priority 2

Simple Example Illustrating PIAS

Flow 3 with 2 packets arrives

Priority 1

Priority 2

Priority 3

Priority 4

Simple Example Illustrating PIAS

Flow 3 and 1 transmit simultaneously

Priority 1

Priority 2

Priority 3

Priority 4

Simple Example Illustrating PIAS

Flow 3 finishes while flow 1 is demoted to priority 3

Priority 1

Priority 2

Priority 3

Priority 4

Simple Example Illustrating PIAS

Flow 4 with 2 packets arrives

Priority 1

Priority 2

Priority 3

Priority 4

Simple Example Illustrating PIAS

Flow 4 and 1 transmit simultaneously

Priority 1

Priority 2

Priority 3

Priority 4

Simple Example Illustrating PIAS

Flow 4 finishes while flow 1 is demoted to priority 4

Priority 1

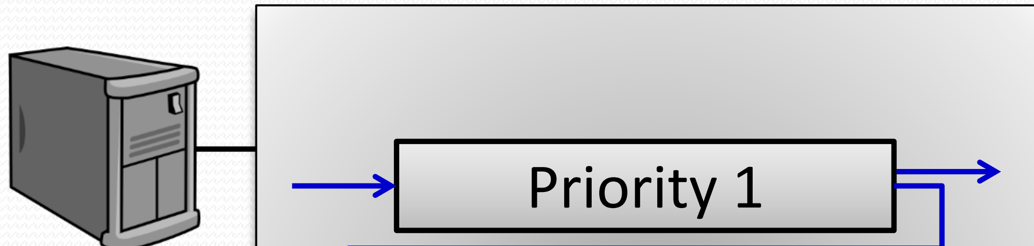
Priority 2

Priority 3

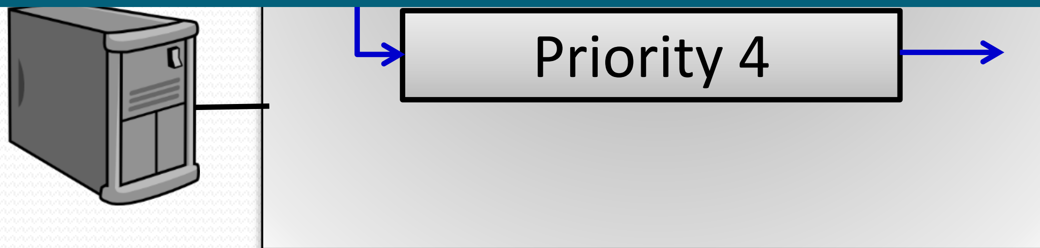
Priority 4

Simple Example Illustrating PIAS

Eventually, flow 1 finishes in priority 4



With MLFQ, PIAS can emulate Shortest Job First without prior knowledge of flow size information



Challenges Exist

- How to determine the demotion threshold for each queue of MLFQ to minimize the FCT?
- As DCN traffic varies across both time and space, how to make PIAS perform efficiently and stably in such dynamic environment?

Determine Thresholds

- Thresholds depend on:
  - Flow size distribution
  - Traffic load
- Solution:
  - Solve a FCT minimization problem to calculate demotion thresholds
- Problem:
  - Traffic is highly dynamic

Traffic variations -> Mismatched thresholds

Impact of Mismatches

10MB

10MB

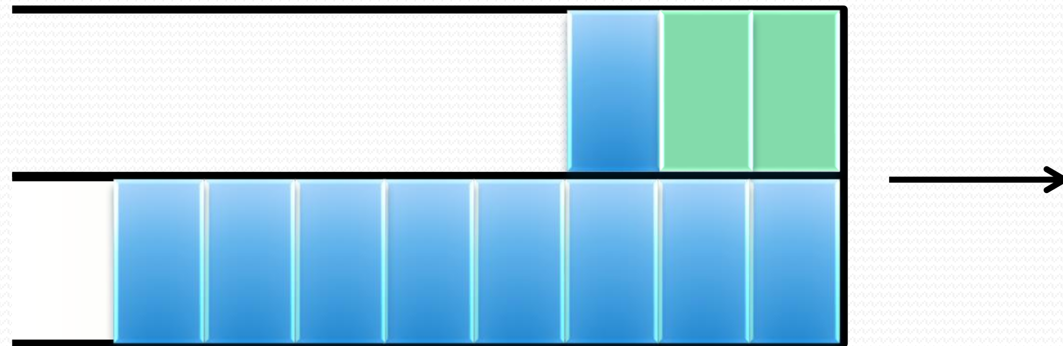
20KB

High

Low

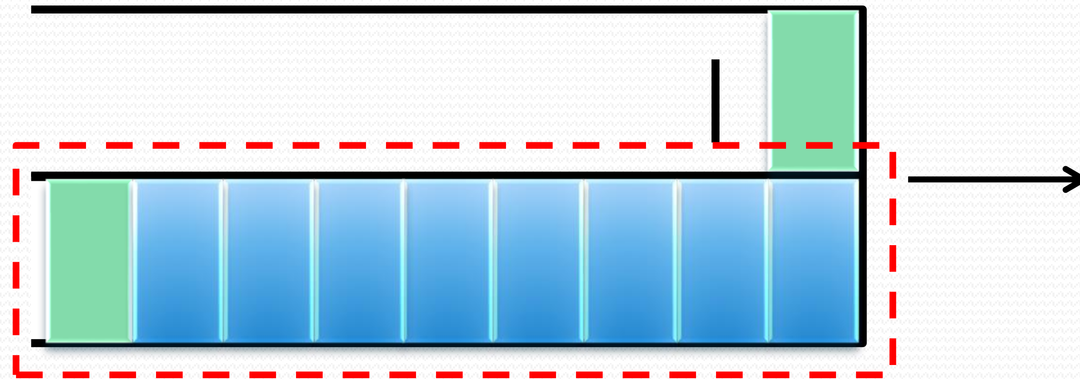
Impact of Mismatches

- When the threshold is perfect (20KB)



Impact of Mismatches

- When the threshold is too small (10KB)



Increased latency for short flows

Impact of Mismatches

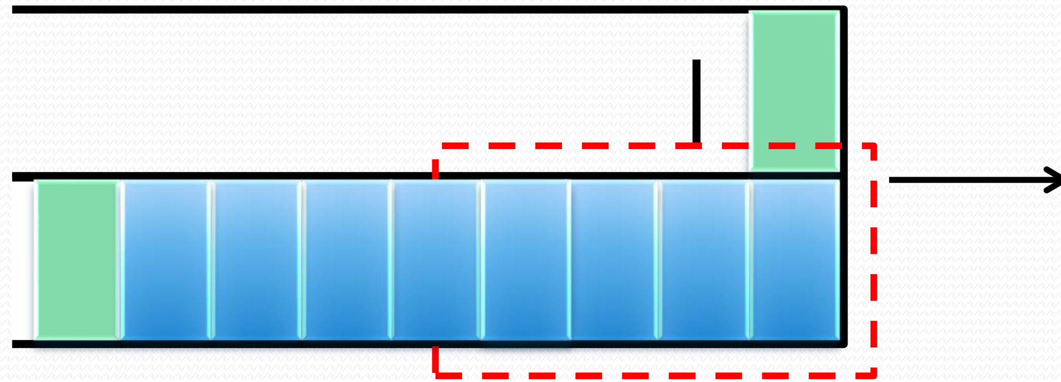
- When the threshold is too large (1MB)

Leverage ECN to keep low buffer occupancy

Increased latency for short flows

Handle Mismatches

- When the threshold is too small (10KB)

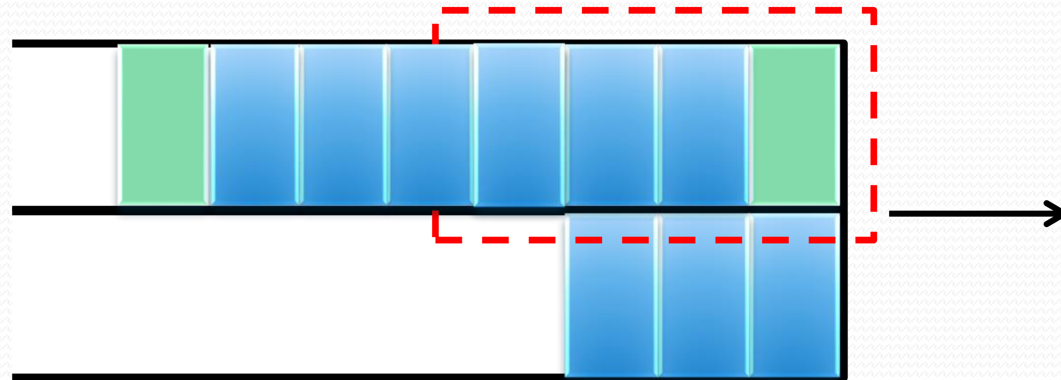


If we enable ECN

ECN can keep low latency

Handle Mismatches

- When the threshold is too large (1MB)



ECN can keep low latency

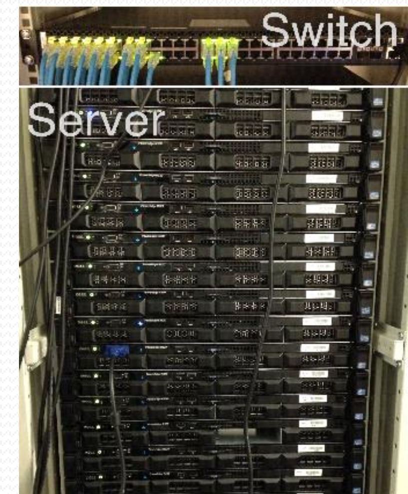
If we enable ECN

Outline

- Background
- PIAS
- **Implementation and evaluation**
- Review

Testbed Experiments

- PIAS prototype
  - https://github.com/HKUST-SING/PIAS-Software
- Testbed Setup
  - A Gigabit Pronto-3295 switch
  - 16 Dell servers
- Benchmarks
  - Web search (DCTCP paper)
  - Data mining (VL2 paper)
  - Memcached

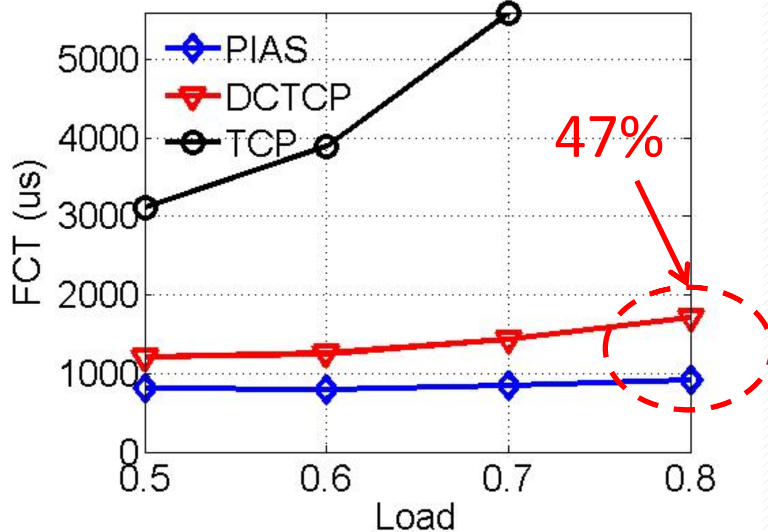


Testbed Experiments

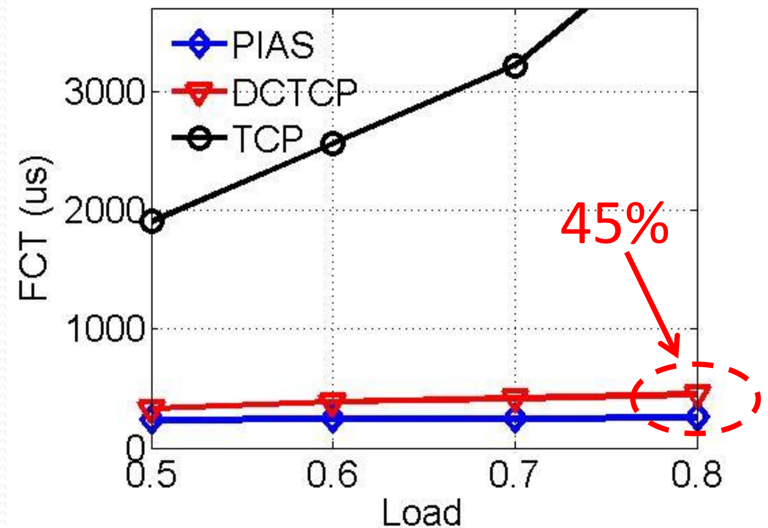
Name	Last commit message	Last commit date
..		
Makefile	Init pias4	10 years ago
flow.c	Fix some indentation	10 years ago
flow.h	Init pias4	10 years ago
jprobe.c	Init pias4	10 years ago
jprobe.h	Init pias4	10 years ago
main.c	Init pias4	10 years ago
netfilter.c	Init pias4	10 years ago
netfilter.h	Init pias4	10 years ago
network.c	Init pias4	10 years ago
network.h	Init pias4	10 years ago
params.c	Fix some indentation	10 years ago
params.h	Init pias4	10 years ago

Linux Kernel Module

Small Flows (<100KB)



Web Search



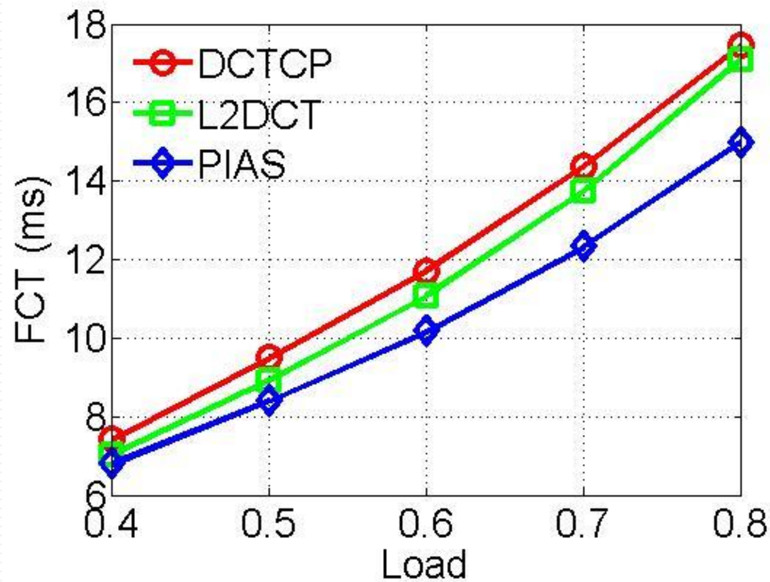
Data Mining

Compared to DCTCP, PIAS reduces average FCT of small flows by up to 47% and 45%

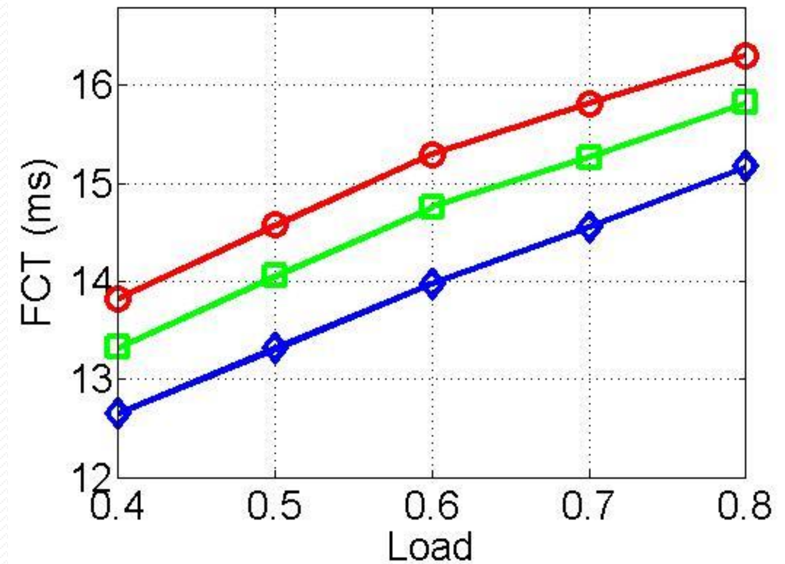
NS2 Simulation Setup

- Topology
  - 144-host leaf-spine fabric with 10G/40G links
- Workloads
  - Web search (DCTCP paper)
  - Data mining (VL2 paper)
- Schemes
  - Information-agnostic: PIAS, DCTCP and L2DCT
  - Information-aware: pFabric

Overall Performance



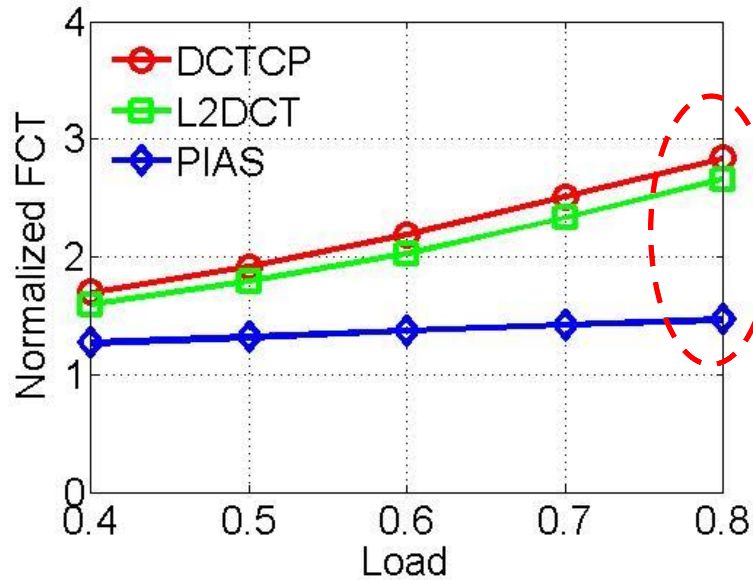
Web Search



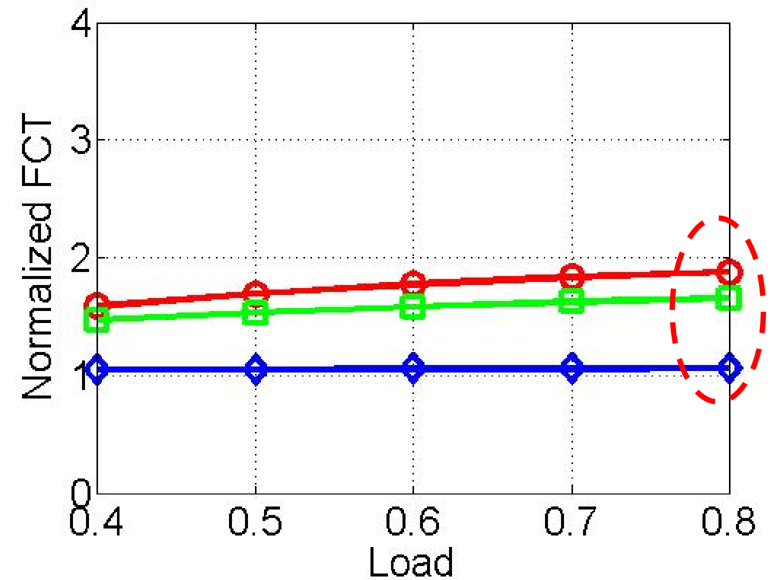
Data Mining

PIAS has an obvious advantage over DCTCP and L2DCT in both workloads.

Small Flows (<100KB)



Web Search

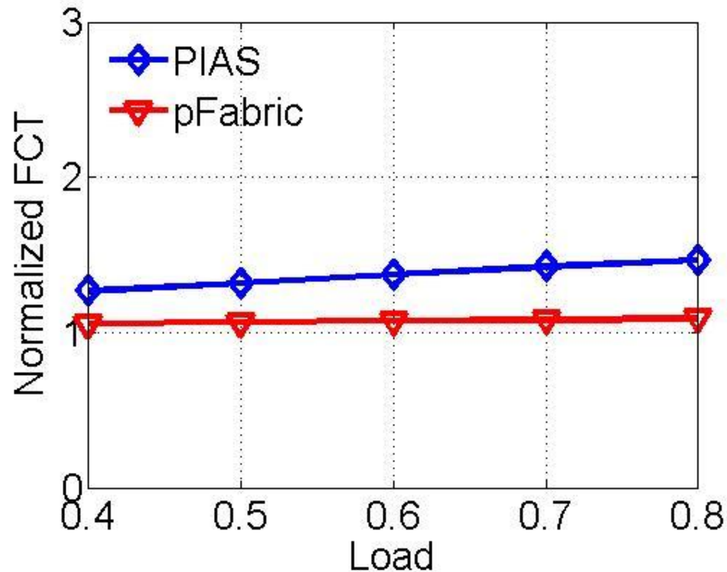


Data Mining

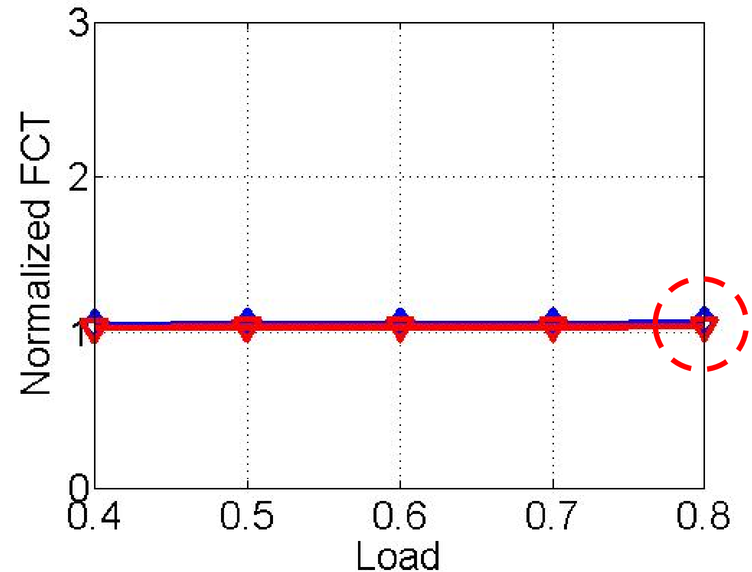
40% - 50% improvement

Simulations confirm testbed experiment results

Comparison with pFabric



Web Search



Data Mining

PIAS only has 4.9% performance gap to pFabric for small flows in data mining workload

Outline

- Background
- PIAS
- Implementation and evaluation
- **Review**

Review

- We introduce scheduling algorithms:
  - FCFS/RR/SJF/SRJF/MLFQ
- We introduce PIAS, which implements MLFQ using commodity switch
  - Packet tagging at endhost using MLFQ
  - Strict priority at switch

One More Thing

- Are flow sizes indeed difficult to predict?
  - Yes ~10 years ago. DCN hosts diverse applications.
  - Maybe **No** nowadays.
    - Flows generated from machine learning training/inference are very regular in terms of size & patterns.
      - E.g., training follows an iterative pattern