A ϕ-Competitive Algorithm for Scheduling Packets with Deadlines

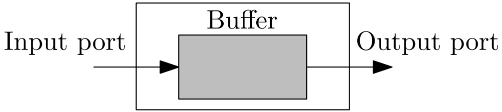
Pavel Veselý
University of Warwick

Joint work with **Marek Chrobak** (UC Riverside),
Łukasz Jeż (Wrocław Univ.), and
Jiří Sgall (Charles Univ., Prague).

SODA'19, January 6

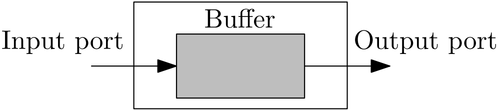
Online Packet Scheduling with Deadlines

Network switch:

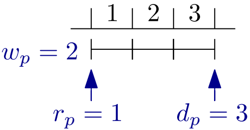


Online Packet Scheduling with Deadlines

Network switch:

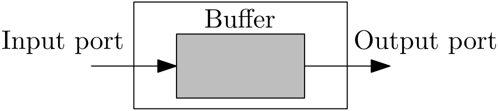


- Packets arrive over time
- Each has a deadline and a weight

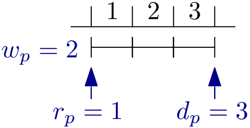


Online Packet Scheduling with Deadlines

Network switch:

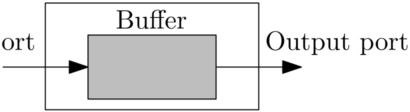


- Packets arrive over time
- Each has a deadline and a weight
- Time discrete, consisting of *slots* or *steps*
- One packet transmitted in each step

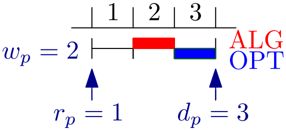


Online Packet Scheduling with Deadlines

Network switch:

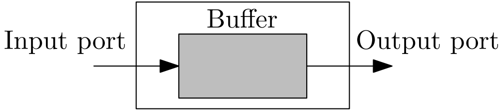


- Packets arrive over time
- Each has a deadline and a weight
- Time discrete, consisting of *slots* or *steps*
- One packet transmitted in each step

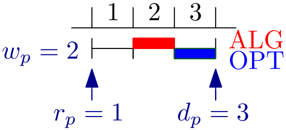


Online Packet Scheduling with Deadlines

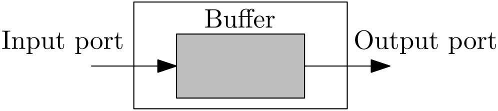
Network switch:



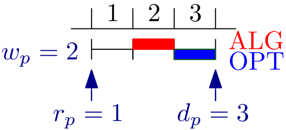
- Packets arrive over time
- Each has a deadline and a weight
- Time discrete, consisting of *slots* or *steps*
- One packet transmitted in each step
- Goal: maximize total weight of scheduled packets

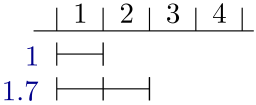


Online Packet Scheduling with Deadlines

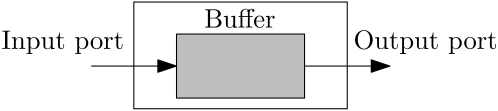
Network switch: 

- Packets arrive over time
- Each has a deadline and a weight
- Time discrete, consisting of *slots* or *steps*
- One packet transmitted in each step
- Goal: maximize total weight of scheduled packets

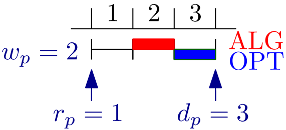


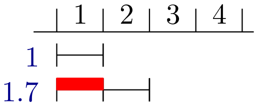


Network switch:

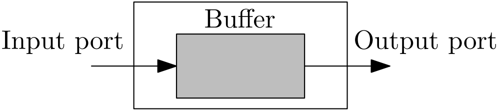


- Packets arrive over time
- Each has a deadline and a weight
- Time discrete, consisting of *slots* or *steps*
- One packet transmitted in each step
- Goal: maximize total weight of scheduled packets

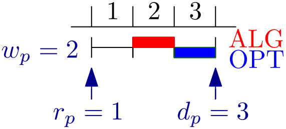


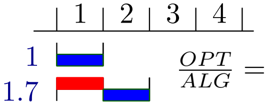


Network switch:



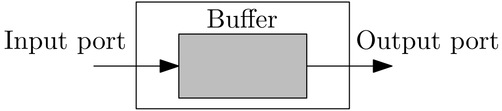
- Packets arrive over time
- Each has a deadline and a weight
- Time discrete, consisting of *slots* or *steps*
- One packet transmitted in each step
- Goal: maximize total weight of scheduled packets



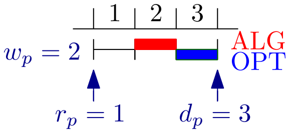


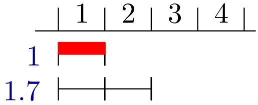
$$\frac{OPT}{ALG} = \frac{1.7+1}{1.7} \approx 1.59$$

Online Packet Scheduling with Deadlines

Network switch: 

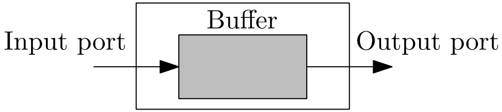
- Packets arrive over time
- Each has a deadline and a weight
- Time discrete, consisting of *slots* or *steps*
- One packet transmitted in each step
- Goal: maximize total weight of scheduled packets



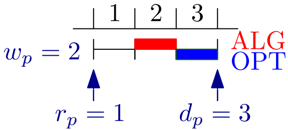


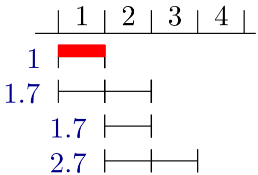
Online Packet Scheduling with Deadlines

Network switch:



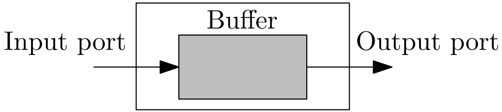
- Packets arrive over time
- Each has a deadline and a weight
- Time discrete, consisting of *slots* or *steps*
- One packet transmitted in each step
- Goal: maximize total weight of scheduled packets



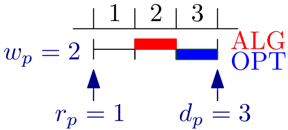


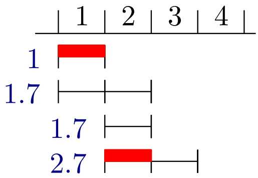
Online Packet Scheduling with Deadlines

Network switch:



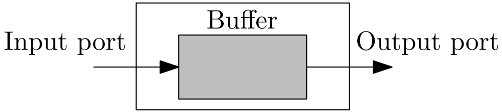
- Packets arrive over time
- Each has a deadline and a weight
- Time discrete, consisting of *slots* or *steps*
- One packet transmitted in each step
- Goal: maximize total weight of scheduled packets



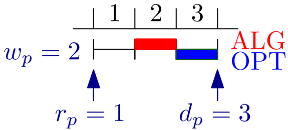


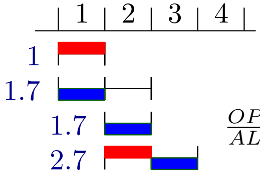
Online Packet Scheduling with Deadlines

Network switch:



- Packets arrive over time
- Each has a deadline and a weight
- Time discrete, consisting of *slots* or *steps*
- One packet transmitted in each step
- Goal: maximize total weight of scheduled packets

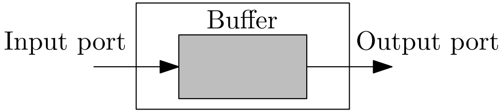




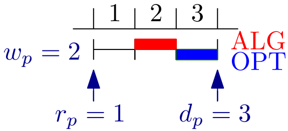
$$\frac{OPT}{ALG} = \frac{2 \cdot 1.7 + 2.7}{1.7 + 2.7} \approx 1.39$$

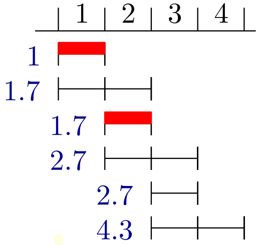
Online Packet Scheduling with Deadlines

Network switch:



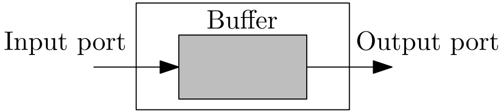
- Packets arrive over time
- Each has a deadline and a weight
- Time discrete, consisting of *slots* or *steps*
- One packet transmitted in each step
- Goal: maximize total weight of scheduled packets



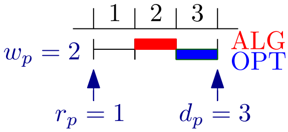


Online Packet Scheduling with Deadlines

Network switch:



- Packets arrive over time
- Each has a deadline and a weight
- Time discrete, consisting of *slots* or *steps*
- One packet transmitted in each step
- Goal: maximize total weight of scheduled packets



Scheduling problem $1|online, r_j, p_j = 1| \sum w_j(1 - U_j)$
A.k.a. Buffer Management in QoS Switches

Competitive ratio of online algorithms

- Algorithm is R-competitive if for any instance I

$$\mathrm{OPT}(I) \leq R \cdot \mathrm{ALG}(I)$$

Competitive ratio of online algorithms

- Algorithm is R-competitive if for any instance I

$$\mathrm{OPT}(I) \leq R \cdot \mathrm{ALG}(I)$$

- Game: the algorithm vs. an adversary 
  - The adversary decides on further input to maximize OPT/ALG

Previous work

- We focus on deterministic algorithms

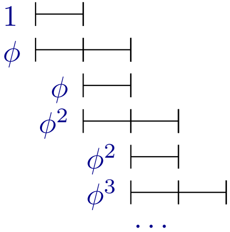
Previous work

- We focus on deterministic algorithms
- Greedy algorithm 2-competitive
  - Schedules always the heaviest pending packet

Previous work

- We focus on deterministic algorithms
- Greedy algorithm 2-competitive
  - Schedules always the heaviest pending packet
- Lower bound of the golden ratio $\phi = \frac{1}{2}(\sqrt{5}+1) \approx 1.618$ [Hájek '01, Andelman *et al.* '03, Chin & Fung '03]

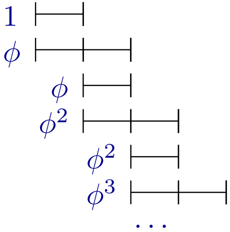
$$\phi + 1 = \phi^2$$



Previous work

- We focus on deterministic algorithms
- Greedy algorithm 2-competitive
  - Schedules always the heaviest pending packet
- Lower bound of the golden ratio $\phi = \frac{1}{2}(\sqrt{5}+1) \approx 1.618$ [Hájek '01, Andelman *et al.* '03, Chin & Fung '03]

$$\phi + 1 = \phi^2$$

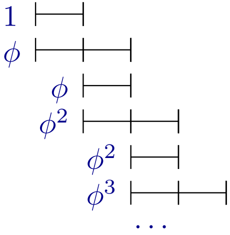


- 1.939-competitive algorithm [Chrobak *et al.*'04]
- 1.854-competitive algorithm [Li *et al.*'07]
- 1.828-competitive algorithm [Englert & Westermann '07]

Previous work

- We focus on deterministic algorithms
- Greedy algorithm 2-competitive
  - Schedules always the heaviest pending packet
- Lower bound of the golden ratio $\phi = \frac{1}{2}(\sqrt{5}+1) \approx 1.618$ [Hájek '01, Andelman *et al.* '03, Chin & Fung '03]

$$\phi + 1 = \phi^2$$

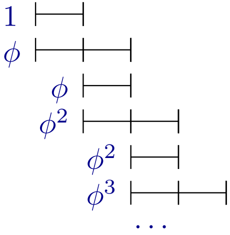


  - 1.939-competitive algorithm [Chrobak *et al.*'04]
  - 1.854-competitive algorithm [Li *et al.*'07]
  - 1.828-competitive algorithm [Englert & Westermann '07]
  - Ratio ϕ for special instances [Kesselman *et al.* '01, Chin *et al.* '04, Li *et al.* '05, Bienkowski *et al.* '13, Böhm *et al.* '16]

Previous work

- We focus on deterministic algorithms
- Greedy algorithm 2-competitive
  - Schedules always the heaviest pending packet
- Lower bound of the golden ratio $\phi = \frac{1}{2}(\sqrt{5}+1) \approx 1.618$ [Hájek '01, Andelman *et al.* '03, Chin & Fung '03]

$$\phi + 1 = \phi^2$$



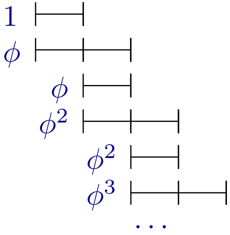
- 1.939-competitive algorithm [Chrobak *et al.*'04]
- 1.854-competitive algorithm [Li *et al.*'07]
- 1.828-competitive algorithm [Englert & Westermann '07]
- Ratio ϕ for special instances [Kesselman *et al.* '01, Chin *et al.* '04, Li *et al.* '05, Bienkowski *et al.* '13, Böhm *et al.* '16]

Is there a ϕ-competitive algorithm?

Previous work

- We focus on deterministic algorithms
- Greedy algorithm 2-competitive
  - Schedules always the heaviest pending packet
- Lower bound of the golden ratio $\phi = \frac{1}{2}(\sqrt{5}+1) \approx 1.618$ [Hájek '01, Andelman *et al.* '03, Chin & Fung '03]

$$\phi + 1 = \phi^2$$



- 1.939-competitive algorithm [Chrobak *et al.*'04]
- 1.854-competitive algorithm [Li *et al.*'07]
- 1.828-competitive algorithm [Englert & Westermann '07]
- Ratio ϕ for special instances [Kesselman *et al.* '01, Chin *et al.* '04, Li *et al.* '05, Bienkowski *et al.* '13, Böhm *et al.* '16]

Is there a ϕ-competitive algorithm?

Yes!

New result

Theorem

There is a ϕ-competitive deterministic algorithm.

New result

Theorem

There is a ϕ-competitive deterministic algorithm.

Key technique: Plan

- Max-weight *feasible* subset of pending packets in step t
  - feasible = can be scheduled in slots $t, t+1, \ldots$

New result

Theorem

There is a ϕ-competitive deterministic algorithm.

Key technique: Plan

- Max-weight *feasible* subset of pending packets in step t
  - feasible $=$ can be scheduled in slots $t, t+1, \ldots$
- Optimal future profit unless new packets arrive

New result

Theorem

There is a ϕ-competitive deterministic algorithm.

Key technique: Plan

- Max-weight *feasible* subset of pending packets in step t
  - feasible = can be scheduled in slots $t, t+1, \ldots$
- Optimal future profit unless new packets arrive
- Scheduled plans (a.k.a. provisional schedules) used already by [Li *et al.* '05, Li *et al.* '07, Englert & Westermann '07]

Algorithm LessGreedy(ϕ)

Plan $\mathcal{P}$

- Max-weight *feasible* subset of pending packets in step t
  - feasible = can be scheduled in slots $t, t+1, \ldots$

Algorithm LessGreedy(ϕ)

Plan $\mathcal{P}$

- Max-weight *feasible* subset of pending packets in step t
  - feasible = can be scheduled in slots $t, t+1, \ldots$

Algorithm LessGreedy(ϕ)

- Schedule packet $p \in \mathcal{P}$ maximizing $\phi \cdot w_p + w(\mathcal{Q}_p)$

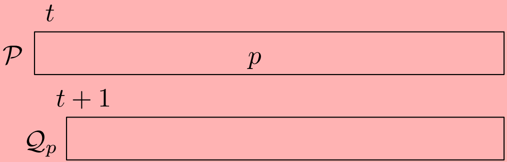
Algorithm LessGreedy(ϕ)

Plan $\mathcal{P}$

- Max-weight *feasible* subset of pending packets in step t
  - feasible = can be scheduled in slots $t, t+1, \dots$

Algorithm LessGreedy(ϕ)

- Schedule packet $p \in \mathcal{P}$ maximizing $\phi \cdot w_p + w(\mathcal{Q}_p)$
  - $\mathcal{Q}_p$ is the plan after p is scheduled and time is incremented ($p \notin \mathcal{Q}_p$)



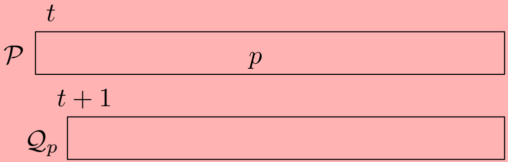
Algorithm LessGreedy(ϕ)

Plan $\mathcal{P}$

- Max-weight *feasible* subset of pending packets in step t
  - feasible = can be scheduled in slots $t, t+1, \ldots$

Algorithm LessGreedy(ϕ)

- Schedule packet $p \in \mathcal{P}$ maximizing $\phi \cdot w_p + w(\mathcal{Q}_p)$
  - $\mathcal{Q}_p$ is the plan after p is scheduled and time is incremented ($p \notin \mathcal{Q}_p$)



  - w_p is the immediate gain
  - $w(\mathcal{Q}_p)$ is the optimal *future* profit unless new packets arrive

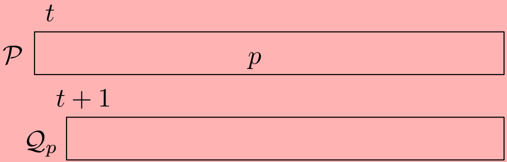
Algorithm LessGreedy(ϕ)

Plan $\mathcal{P}$

- Max-weight *feasible* subset of pending packets in step t
  - feasible $=$ can be scheduled in slots $t, t+1, \ldots$

Algorithm LessGreedy(ϕ)

- Schedule packet $p \in \mathcal{P}$ maximizing $\phi \cdot w_p + w(\mathcal{Q}_p)$
  - $\mathcal{Q}_p$ is the plan after p is scheduled and time is incremented ($p \notin \mathcal{Q}_p$)



  - w_p is the immediate gain
  - $w(\mathcal{Q}_p)$ is the optimal *future* profit unless new packets arrive

- Very elegant algorithm ...

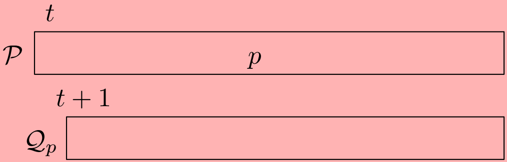
Algorithm LessGreedy(ϕ)

Plan $\mathcal{P}$

- Max-weight *feasible* subset of pending packets in step t
  - feasible = can be scheduled in slots $t, t+1, \ldots$

Algorithm LessGreedy(ϕ)

- Schedule packet $p \in \mathcal{P}$ maximizing $\phi \cdot w_p + w(\mathcal{Q}_p)$
  - $\mathcal{Q}_p$ is the plan after p is scheduled and time is incremented ($p \notin \mathcal{Q}_p$)



  - w_p is the immediate gain
  - $w(\mathcal{Q}_p)$ is the optimal *future* profit unless new packets arrive

- Very elegant algorithm ...
- ... but *not* ϕ-competitive

Plan and its Structure

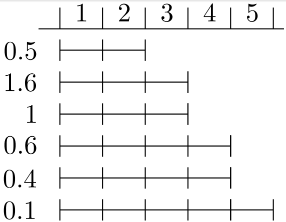
Plan $\mathcal{P}$

- Max-weight *feasible* subset of pending packets in step t
  - feasible = can be scheduled in slots $t, t+1, \ldots$

Plan and its Structure

Plan $\mathcal{P}$

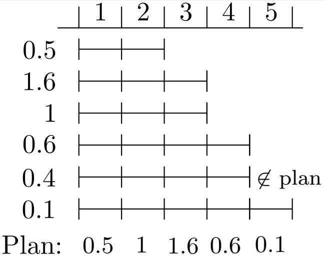
- Max-weight *feasible* subset of pending packets in step t
  - feasible = can be scheduled in slots $t, t+1, \ldots$



Plan and its Structure

Plan $\mathcal{P}$

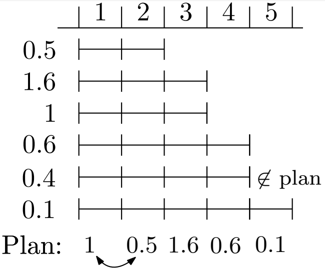
- Max-weight *feasible* subset of pending packets in step t
  - feasible = can be scheduled in slots $t, t+1, \ldots$



Plan and its Structure

Plan $\mathcal{P}$

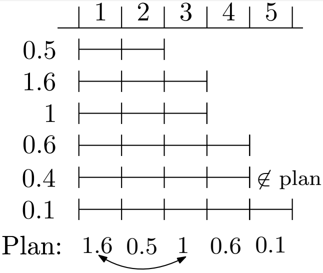
- Max-weight *feasible* subset of pending packets in step t
  - feasible = can be scheduled in slots $t, t+1, \ldots$



Plan and its Structure

Plan $\mathcal{P}$

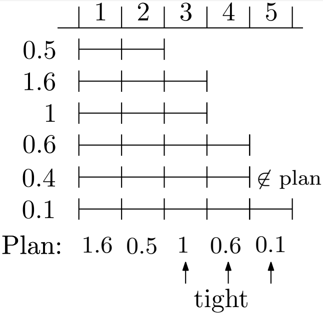
- Max-weight *feasible* subset of pending packets in step t
  - feasible = can be scheduled in slots $t, t+1, \ldots$



Plan and its Structure

Plan $\mathcal{P}$

- Max-weight *feasible* subset of pending packets in step t
  - feasible = can be scheduled in slots $t, t+1, \ldots$



Definition

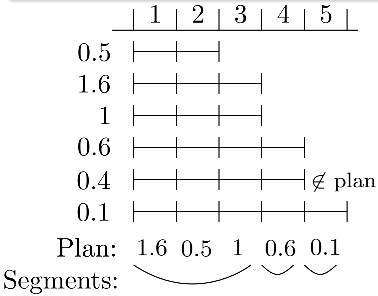
Slot τ is *tight* iff

$\#$ of slots till $\tau = \#$ of packets $j \in \mathcal{P} : d_j \leq \tau$

Plan and its Structure

Plan $\mathcal{P}$

- Max-weight *feasible* subset of pending packets in step t
  - feasible = can be scheduled in slots $t, t+1, \ldots$



Definition

Slot τ is *tight* iff

$\#$ of slots till $\tau = \#$ of packets $j \in \mathcal{P} : d_j \leq \tau$

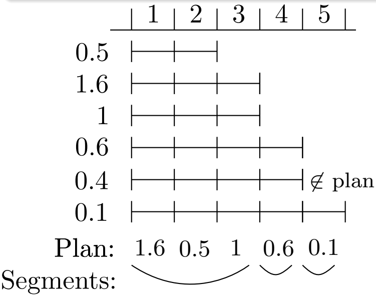
Definition

Segment = interval between tight slots

Plan and its Structure

Plan $\mathcal{P}$

- Max-weight *feasible* subset of pending packets in step t
  - feasible = can be scheduled in slots $t, t+1, \ldots$



Definition

Slot τ is *tight* iff

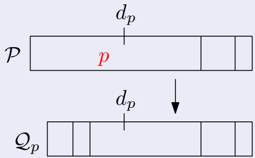
$\#$ of slots till τ = $\#$ of packets $j \in \mathcal{P} : d_j \leq \tau$

Definition

Segment = interval between tight slots

Plan Updates After Packet p is Scheduled

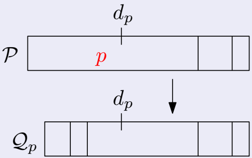
p in the 1st segment



$$\mathcal{Q}_p = \mathcal{P} \setminus \{p\}$$

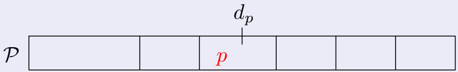
Plan Updates After Packet p is Scheduled

p in the 1st segment



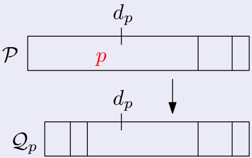
$\mathcal{Q}_p = \mathcal{P} \setminus \{p\}$

p in a later segment



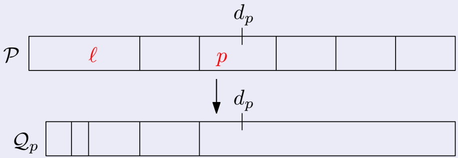
Plan Updates After Packet p is Scheduled

p in the 1st segment



$\mathcal{Q}_p = \mathcal{P} \setminus \{p\}$

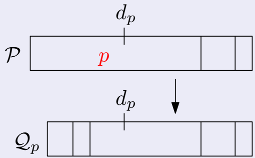
p in a later segment



- ℓ = lightest in the 1st segment

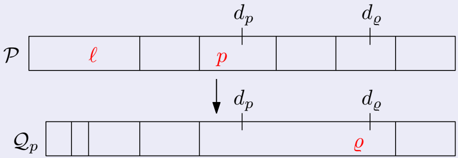
Plan Updates After Packet p is Scheduled

p in the 1st segment



$$\mathcal{Q}_p = \mathcal{P} \setminus \{p\}$$

p in a later segment



$$\mathcal{Q}_p = \mathcal{P} \setminus \{\ell, p\} \cup \{\varrho\}$$

- ℓ = lightest in the 1st segment
- ϱ = heaviest not in $\mathcal{P}$ which can replace p
  - *replacement packet* for p

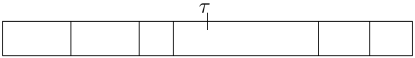
Problem of LessGreedy: Weight Decreases in the Plan

Problem of LessGreedy: Weight Decreases in the Plan

- minwt(τ) = min-weight in $\mathcal{P}$ that can be on a slot up to τ

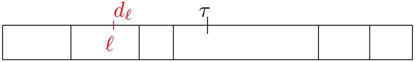
Problem of LessGreedy: Weight Decreases in the Plan

- minwt(τ) $=$ min-weight in $\mathcal{P}$ that can be on a slot up to τ
- $=$ min-weight in $\mathcal{P}$ *till the next tight slot after* τ



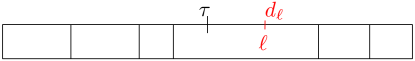
Problem of LessGreedy: Weight Decreases in the Plan

- minwt(τ) = min-weight in $\mathcal{P}$ that can be on a slot up to τ
- = min-weight in $\mathcal{P}$ *till the next tight slot after* τ



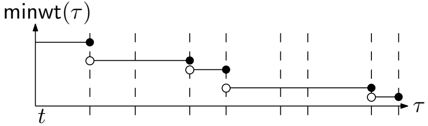
Problem of LessGreedy: Weight Decreases in the Plan

- minwt(τ) $=$ min-weight in $\mathcal{P}$ that can be on a slot up to τ
- $=$ min-weight in $\mathcal{P}$ *till the next tight slot after* τ



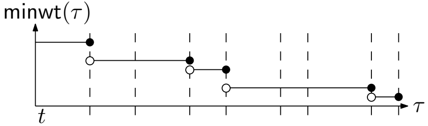
Problem of LessGreedy: Weight Decreases in the Plan

- minwt(τ) $=$ min-weight in $\mathcal{P}$ that can be on a slot up to τ
- $\quad\quad\quad\quad = $ min-weight in $\mathcal{P}$ *till the next tight slot after* τ



Problem of LessGreedy: Weight Decreases in the Plan

- $\text{minwt}(\tau) =$ min-weight in $\mathcal{P}$ that can be on a slot up to τ
- $=$ min-weight in $\mathcal{P}$ *till the next tight slot after* τ

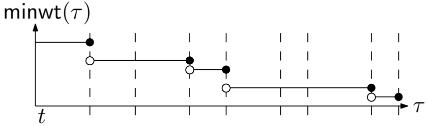


minwt(τ) after plan updates

- for any fixed τ, $\text{minwt}(\tau)$ does not decrease:
  - after arrival of a new packet
  - after scheduling a packet from the 1st segment

Problem of LessGreedy: Weight Decreases in the Plan

- minwt(τ) $=$ min-weight in $\mathcal{P}$ that can be on a slot up to τ
- $=$ min-weight in $\mathcal{P}$ *till the next tight slot after* τ

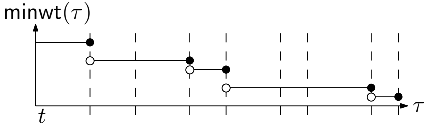


minwt(τ) after plan updates

- for any fixed τ, minwt(τ) does not decrease:
  - after arrival of a new packet
  - after scheduling a packet from the 1st segment
- minwt(τ) decreases for some τ after sch. a packet from later segment

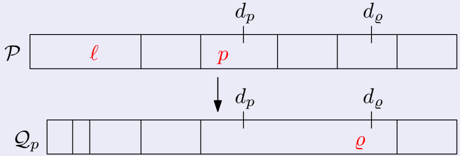
Problem of LessGreedy: Weight Decreases in the Plan

- minwt(τ) = min-weight in $\mathcal{P}$ that can be on a slot up to τ
- = min-weight in $\mathcal{P}$ *till the next tight slot after* τ



minwt(τ) after plan updates

- for any fixed τ, minwt(τ) does not decrease:
  - after arrival of a new packet
  - after scheduling a packet from the 1st segment
- minwt(τ) decreases for some τ after sch. a packet from later segment



The problem:

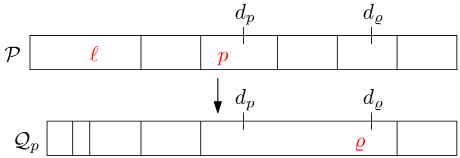
$\varrho \notin \mathcal{P} \Rightarrow w_\varrho < \text{minwt}(d_\varrho)$

Solution: Maintaining Slot-Monotonicity of minwt

- Idea: modify $\textsc{LessGreedy}(\phi)$ so that $\text{minwt}(\tau)$ never decreases

Solution: Maintaining Slot-Monotonicity of minwt

- Idea: modify LessGreedy(ϕ) so that minwt(τ) never decreases

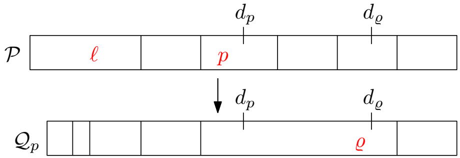


The problem:

$\varrho \notin \mathcal{P} \Rightarrow w_\varrho < \text{minwt}(d_\varrho)$

Solution: Maintaining Slot-Monotonicity of minwt

- Idea: modify LessGreedy(ϕ) so that minwt(τ) never decreases



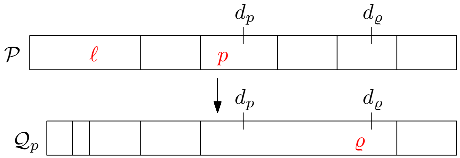
The problem:

$\varrho \notin \mathcal{P} \Rightarrow w_\varrho <$ minwt(d_ϱ)

- $\Rightarrow$ increase the weight of ϱ to minwt(d_ϱ)

Solution: Maintaining Slot-Monotonicity of minwt

- Idea: modify LessGreedy(ϕ) so that minwt(τ) never decreases



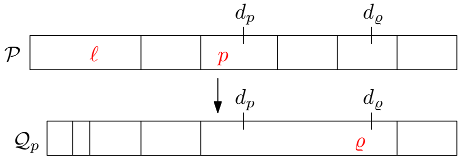
The problem:

$\varrho \notin \mathcal{P} \Rightarrow w_\varrho < \text{minwt}(d_\varrho)$

- $\Rightarrow$ increase the weight of ϱ to minwt(d_ϱ)
- Not enough if segments merge:

Solution: Maintaining Slot-Monotonicity of minwt

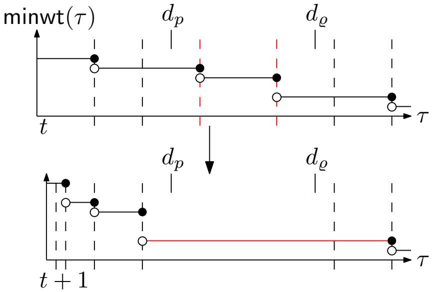
- Idea: modify LessGreedy(ϕ) so that minwt(τ) never decreases



The problem:

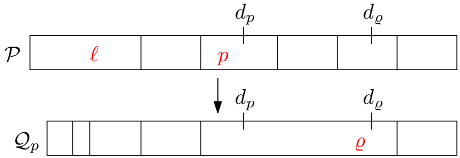
$\varrho \notin \mathcal{P} \Rightarrow w_\varrho < \text{minwt}(d_\varrho)$

- $\Rightarrow$ increase the weight of ϱ to minwt(d_ϱ)
- Not enough if segments merge:



Solution: Maintaining Slot-Monotonicity of minwt

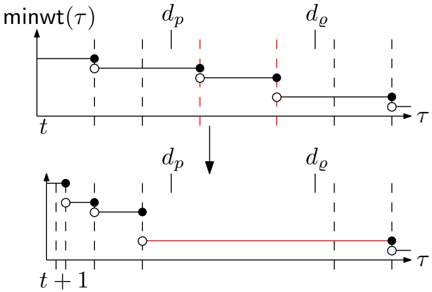
- Idea: modify LESSGREEDY(ϕ) so that minwt(τ) never decreases



The problem:

$\varrho \notin \mathcal{P} \Rightarrow w_\varrho < \text{minwt}(d_\varrho)$

- $\Rightarrow$ increase the weight of ϱ to minwt(d_ϱ)
- Not enough if segments merge:



- $\Rightarrow$ avoid merging segments

Algorithm $\textsc{PlanM}(\phi)$ Maintaining Slot-Monotonicity

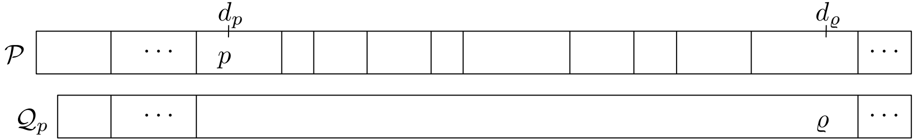
- Schedule packet $p \in \mathcal{P}$ maximizing $\phi \cdot w_p + w(\mathcal{Q}_p)$

Algorithm $\textsc{PlanM}(\phi)$ Maintaining Slot-Monotonicity

- Schedule packet $p \in \mathcal{P}$ maximizing $\phi \cdot w_p + w(\mathcal{Q}_p)$
- If p is in a later segment of $\mathcal{P}$:
  - Increase the weight of ϱ to minwt(d_ϱ)

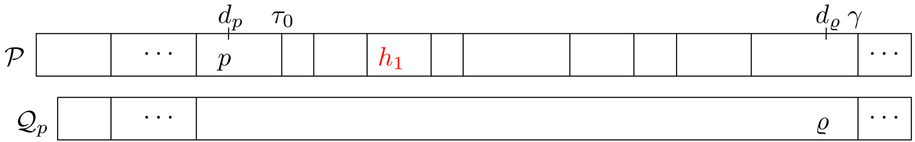
Algorithm $\textsc{PlanM}(\phi)$ Maintaining Slot-Monotonicity

- Schedule packet $p \in \mathcal{P}$ maximizing $\phi \cdot w_p + w(\mathcal{Q}_p)$
- If p is in a later segment of $\mathcal{P}$:
  - Increase the weight of ϱ to minwt(d_ϱ)
  - Avoid merging segments:



Algorithm $\textsc{PlanM}(\phi)$ Maintaining Slot-Monotonicity

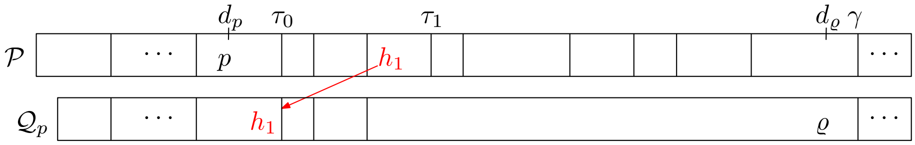
- Schedule packet $p \in \mathcal{P}$ maximizing $\phi \cdot w_p + w(\mathcal{Q}_p)$
- If p is in a later segment of $\mathcal{P}$:
  - Increase the weight of ϱ to minwt(d_ϱ)
  - Avoid merging segments:



  - h_1 = heaviest packet in $(\tau_0, \gamma]$,

Algorithm $\textsc{PlanM}(\phi)$ Maintaining Slot-Monotonicity

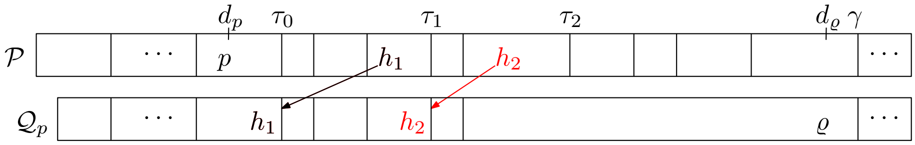
- Schedule packet $p \in \mathcal{P}$ maximizing $\phi \cdot w_p + w(\mathcal{Q}_p)$
- If p is in a later segment of $\mathcal{P}$:
  - Increase the weight of ϱ to minwt(d_ϱ)
  - Avoid merging segments:



  - h_1 = heaviest packet in $(\tau_0, \gamma]$,
  - decrease deadline of h_1 to τ_0

Algorithm PLANM(ϕ) Maintaining Slot-Monotonicity

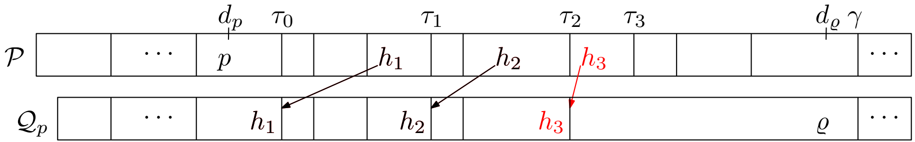
- Schedule packet $p \in \mathcal{P}$ maximizing $\phi \cdot w_p + w(\mathcal{Q}_p)$
- If p is in a later segment of $\mathcal{P}$:
  - Increase the weight of ϱ to minwt(d_ϱ)
  - Avoid merging segments:



  - h_2 = heaviest packet in $(\tau_1, \gamma]$,
  - decrease deadline of h_2 to τ_1

Algorithm PLANM(ϕ) Maintaining Slot-Monotonicity

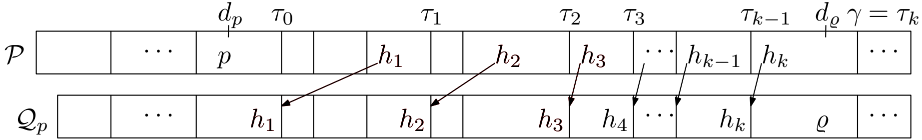
- Schedule packet $p \in \mathcal{P}$ maximizing $\phi \cdot w_p + w(\mathcal{Q}_p)$
- If p is in a later segment of $\mathcal{P}$:
  - Increase the weight of ϱ to minwt(d_ϱ)
  - Avoid merging segments:



  - h_3 = heaviest packet in $(\tau_2, \gamma]$,
  - decrease deadline of h_3 to τ_2

Algorithm $\text{PLANM}(\phi)$ Maintaining Slot-Monotonicity

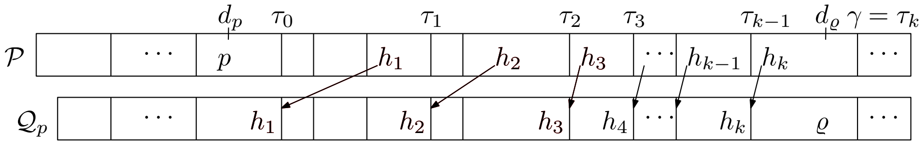
- Schedule packet $p \in \mathcal{P}$ maximizing $\phi \cdot w_p + w(\mathcal{Q}_p)$
- If p is in a later segment of $\mathcal{P}$:
  - Increase the weight of ϱ to minwt(d_ϱ)
  - Avoid merging segments:



  - for $i = 1, 2, \ldots :$ h_i = heaviest packet in $(\tau_{i-1}, \gamma]$,
  - decrease deadline of h_i to τ_{i-1}
  - stop when $\tau_i = \gamma$

Algorithm PlanM(ϕ) Maintaining Slot-Monotonicity

- Schedule packet $p \in \mathcal{P}$ maximizing $\phi \cdot w_p + w(\mathcal{Q}_p)$
- If p is in a later segment of $\mathcal{P}$:
  - Increase the weight of ϱ to minwt(d_ϱ)
  - Avoid merging segments:



  - for $i = 1, 2, \ldots :$ h_i = heaviest packet in $(\tau_{i-1}, \gamma]$,
  - decrease deadline of h_i to τ_{i-1}
  - stop when $\tau_i = \gamma$
  - if $w_{h_i} < \text{minwt}(\tau_{i-1})$, then increase weight of h_i to minwt(τ_{i-1})

Key Ideas of the Analysis

Goal: $\mathrm{OPT}(I) \leq \phi \cdot \mathrm{ALG}(I)$ for any instance I

Key Ideas of the Analysis

Goal: $\mathrm{OPT}(I) \leq \phi \cdot \mathrm{ALG}(I)$ for any instance I

Amortization Techniques

1. Increasing weights
   - Algorithm's future profit *may* get higher

Key Ideas of the Analysis

Goal: $\text{OPT}(I) \leq \phi \cdot \text{ALG}(I)$ for any instance I

Amortization Techniques

1. Increasing weights
   - Algorithm's future profit *may* get higher
     - Decrease algorithm's current profit by weight increase

Key Ideas of the Analysis

Goal: $\mathrm{OPT}(I) \leq \phi \cdot \mathrm{ALG}(I)$ for any instance I

Amortization Techniques

1. Increasing weights
   - Algorithm's future profit *may* get higher
     - Decrease algorithm's current profit by weight increase
2. Modifications of the adversary (optimal) schedule ADV

Key Ideas of the Analysis

Goal: $\text{OPT}(I) \leq \phi \cdot \text{ALG}(I)$ for any instance I

Amortization Techniques

1. Increasing weights
   - Algorithm's future profit *may* get higher
     - Decrease algorithm's current profit by weight increase
2. Modifications of the adversary (optimal) schedule ADV
3. Potential function

Key Ideas of the Analysis

Amortization Techniques

1. Increasing weights
2. Modifications of the adversary (optimal) schedule ADV
3. Potential function

Potential function

Advantage of the algorithm over the adversary:

1. $\mathcal{P} \setminus \text{ADV} =$ packets in the plan that the adversary will not schedule

Key Ideas of the Analysis

Amortization Techniques

1. Increasing weights
2. Modifications of the adversary (optimal) schedule ADV
3. Potential function

Potential function

Advantage of the algorithm over the adversary:

1. $\mathcal{P} \setminus \text{ADV} =$ packets in the plan that the adversary will not schedule
2. $\mathcal{F} =$ subset of pending packets not in plan $\mathcal{P}$
   - Candidates for replacement packets

Key Ideas of the Analysis

Amortization Techniques

1. Increasing weights
2. Modifications of the adversary (optimal) schedule ADV
3. Potential function

Potential function

Advantage of the algorithm over the adversary:

1. $\mathcal{P} \setminus \mathsf{ADV}$ = packets in the plan that the adversary will not schedule
2. $\mathcal{F}$ = subset of pending packets not in plan $\mathcal{P}$
   - Candidates for replacement packets

$$\text{potential} = \frac{1}{\phi} w\left(\mathcal{P} \setminus \mathsf{ADV} \cup \mathcal{F}\right)$$

Key Ideas of the Analysis

Amortization Techniques

1. Increasing weights
2. Modifications of the adversary (optimal) schedule ADV
3. Potential function

Potential function

Advantage of the algorithm over the adversary:

1. $\mathcal{P} \setminus \text{ADV}$ = packets in the plan that the adversary will not schedule
2. $\mathcal{F}$ = subset of pending packets not in plan $\mathcal{P}$
   - Candidates for replacement packets

$$\text{potential} = \frac{1}{\phi} w\left(\mathcal{P} \setminus \text{ADV} \cup \mathcal{F}\right)$$

Invariant

- set $\mathcal{P} \setminus \text{ADV} \cup \mathcal{F}$ is feasible

Further Research Directions

$m \geq 1$ packets sent in each step

- Our algorithm is $\phi \approx 1.618$-competitive for any $m \geq 1$
- Best upper bound tends to $\frac{e}{e-1} \approx 1.58$ [Chin *et al.* '04]

Further Research Directions

$m \geq 1$ packets sent in each step

- Our algorithm is $\phi \approx 1.618$-competitive for any $m \geq 1$
- Best upper bound tends to $\frac{e}{e-1} \approx 1.58$ [Chin *et al.* '04]

Randomized algorithms

- Gap between 1.25 [Chin & Fung '04] and $\frac{e}{e-1} \approx 1.58$ [Chin *et al.* '04]

Further Research Directions

$m \geq 1$ packets sent in each step

- Our algorithm is $\phi \approx 1.618$-competitive for any $m \geq 1$
- Best upper bound tends to $\frac{e}{e-1} \approx 1.58$ [Chin *et al.* '04]

Randomized algorithms

- Gap between 1.25 [Chin & Fung '04] and $\frac{e}{e-1} \approx 1.58$ [Chin *et al.* '04]

Memoryless algorithms

- Is there a lower bound $> \phi$ for memoryless algorithms?
- What is the ratio of $\textsc{LessGreedy}(\alpha)$? (Schedule $p \in \mathcal{P}$ max. $\alpha \cdot w_p + w(\mathcal{Q}_p)$)
  - Ratio ≈ 1.893 for a similar algorithm [Englert & Westermann '07]

Further Research Directions

$m \geq 1$ packets sent in each step

- Our algorithm is $\phi \approx 1.618$-competitive for any $m \geq 1$
- Best upper bound tends to $\frac{e}{e-1} \approx 1.58$ [Chin *et al.* '04]

Randomized algorithms 

- Gap between 1.25 [Chin & Fung '04] and $\frac{e}{e-1} \approx 1.58$ [Chin *et al.* '04]

Memoryless algorithms

- Is there a lower bound $> \phi$ for memoryless algorithms?
- What is the ratio of $\textsc{LessGreedy}(\alpha)$? (Schedule $p \in \mathcal{P}$ max. $\alpha \cdot w_p + w(\mathcal{Q}_p)$)
  - Ratio ≈ 1.893 for a similar algorithm [Englert & Westermann '07]

Thank you!