A ϕ-Competitive Algorithm for Scheduling Packets with Deadlines

Pavel Veselý
University of Warwick

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

DIMAP Seminar, October 2
To appear in SODA '19

Outline

- Introduction to competitive analysis
- Model & result
- Algorithm
- Analysis techniques
- Further research directions

Introduction to competitive analysis

An Example: Cabinetmaker

- Each week you make one cabinet

An Example: Cabinetmaker

- Each week you make one cabinet
- Customers order cabinets, each order has
  - a deadline
  - a reward

An Example: Cabinetmaker

- Each week you make one cabinet
- Customers order cabinets, each order has
  - a deadline
  - a reward
- You have two orders on the table:
  - u: deadline this week, reward 10 000 CZK
  - v: deadline next week, reward 16 180 CZK

An Example: Cabinetmaker

- Each week you make one cabinet
- Customers order cabinets, each order has
  - a deadline
  - a reward

- You have two orders on the table:
  - u: deadline this week, reward 10 000 CZK
  - v: deadline next week, reward 16 180 CZK

1) If you select u, then:

An Example: Cabinetmaker

- Each week you make one cabinet
- Customers order cabinets, each order has
  - a deadline
  - a reward
- You have two orders on the table:
  - u: deadline this week, reward 10 000 CZK
  - v: deadline next week, reward 16 180 CZK

1) If you select u, then:

- new order v' arrives
  - deadline next week, reward 16 180 CZK

An Example: Cabinetmaker

- Each week you make one cabinet
- Customers order cabinets, each order has
  - a deadline
  - a reward
- You have two orders on the table:
  - u: deadline this week, reward 10 000 CZK
  - v: deadline next week, reward 16 180 CZK

1) If you select u, then:

- new order v' arrives
  - deadline next week, reward 16 180 CZK
- only one of v and v' served

An Example: Cabinetmaker

- Each week you make one cabinet
- Customers order cabinets, each order has
  - a deadline
  - a reward

- You have two orders on the table:
  - u: deadline this week, reward 10 000 CZK
  - v: deadline next week, reward 16 180 CZK

1) If you select u, then:

- new order v' arrives
  - deadline next week, reward 16 180 CZK
- only one of v and v' served

2) If you select v, then:

An Example: Cabinetmaker

- Each week you make one cabinet
- Customers order cabinets, each order has
  - a deadline
  - a reward
- You have two orders on the table:
  - u: deadline this week, reward 10 000 CZK
  - v: deadline next week, reward 16 180 CZK

1) If you select u, then:

- new order v' arrives
  - deadline next week, reward 16 180 CZK
- only one of v and v' served

2) If you select v, then:

- no order arrives for next week
- u expires unserved

An Example: Cabinetmaker

- Each week you make one cabinet
- Customers order cabinets, each order has
  - a deadline
  - a reward

- You have two orders on the table:
  - u: deadline this week, reward 10 000 CZK
  - v: deadline next week, reward 16 180 CZK

1) If you select u, then:

- new order v' arrives
  - deadline next week, reward 16 180 CZK
- only one of v and v' served

2) If you select v, then:

- no order arrives for next week
- u expires unserved

These are worst-case scenarios

Online optimization & algorithms

Online computation

-

Offline computation

-

Online optimization & algorithms

Online computation

- Input arriving piece by piece

Offline computation

- Whole input available at the beginning

Online optimization & algorithms

Online computation

- Input arriving piece by piece
- Making decisions without knowing future

Offline computation

- Whole input available at the beginning
- All decisions made at once

Online optimization & algorithms

Online computation

- Input arriving piece by piece
- Making decisions without knowing future
- Decisions irrevocable

Offline computation

- Whole input available at the beginning
- All decisions made at once

Online optimization & algorithms

Online computation

- Input arriving piece by piece
- Making decisions without knowing future
- Decisions irrevocable
- Cannot be optimal (usually)

Offline computation

- Whole input available at the beginning
- All decisions made at once
- Find an optimal solution

Online optimization & algorithms

Online computation

- Input arriving piece by piece
- Making decisions without knowing future
- Decisions irrevocable
- Cannot be optimal (usually)

Offline computation

- Whole input available at the beginning
- All decisions made at once
- Find an optimal solution
- Time/memory efficient algorithms

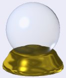
Online optimization & algorithms

Online computation

- Input arriving piece by piece
- Making decisions without knowing future
- Decisions irrevocable
- Cannot be optimal (usually)

Offline computation

- Whole input available at the beginning
- All decisions made at once
- Find an optimal solution
- Time/memory efficient algorithms



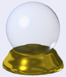
Online optimization & algorithms

Online computation

- Input arriving piece by piece
- Making decisions without knowing future
- Decisions irrevocable
- Cannot be optimal (usually)

Offline computation

- Whole input available at the beginning
- All decisions made at once
- Find an optimal solution
- Time/memory efficient algorithms



Online model

- Sequence of *events* (orders), arrive over time

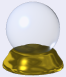
Online optimization & algorithms

Online computation

- Input arriving piece by piece
- Making decisions without knowing future
- Decisions irrevocable
- Cannot be optimal (usually)

Offline computation

- Whole input available at the beginning
- All decisions made at once
- Find an optimal solution
- Time/memory efficient algorithms



Online model

- Sequence of *events* (orders), arrive over time
- Algorithm knows only events that arrived so far

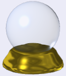
Online optimization & algorithms

Online computation

- Input arriving piece by piece
- Making decisions without knowing future
- Decisions irrevocable
- Cannot be optimal (usually)

Offline computation

- Whole input available at the beginning
- All decisions made at once
- Find an optimal solution
- Time/memory efficient algorithms



Online model

- Sequence of *events* (orders), arrive over time
- Algorithm knows only events that arrived so far
- Some events ask to make *decisions* (Monday mornings)

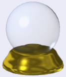
Online optimization & algorithms

Online computation

- Input arriving piece by piece
- Making decisions without knowing future
- Decisions irrevocable
- Cannot be optimal (usually)

Offline computation

- Whole input available at the beginning
- All decisions made at once
- Find an optimal solution
- Time/memory efficient algorithms



Online model

- Sequence of *events* (orders), arrive over time
- Algorithm knows only events that arrived so far
- Some events ask to make *decisions* (Monday mornings)
- Decisions influence the *objective function* (rewards served orders)

Competitive ratio of online algorithms

- Worst-case ratio between
  - value of the optimum solution OPT and
  - value of the algorithm's solution ALG

Competitive ratio of online algorithms

- Worst-case ratio between
  - value of the optimum solution OPT and
  - value of the algorithm's solution ALG

- Algorithm is R-competitive if for any instance I

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

(assuming maximization)

Competitive ratio of online algorithms

- Worst-case ratio between
  - value of the optimum solution OPT and
  - value of the algorithm's solution ALG

- Algorithm is R-competitive if for any instance I

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

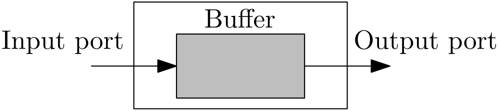
(assuming maximization)

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

Model & Result

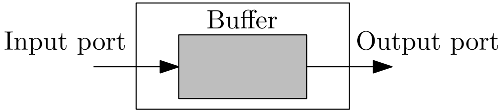
Online Packet Scheduling with Deadlines

Network switch:

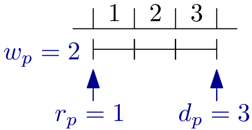


Online Packet Scheduling with Deadlines

Network switch:

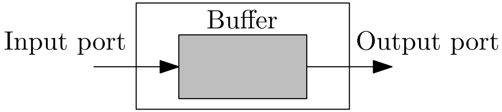


- Packets arrive over time
- Each has a deadline and a weight

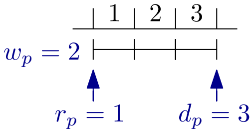


Online Packet Scheduling with Deadlines

Network switch:

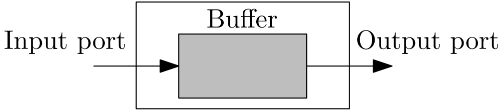


- Packets arrive over time
- Each has a deadline and a weight
- Time discrete, consisting of *slots* or *steps*
- One packet transmitted in each step

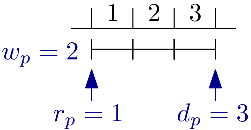


Online Packet Scheduling with Deadlines

Network switch:



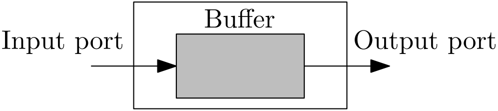
- Packets arrive over time
- Each has a deadline and a weight
- Time discrete, consisting of *slots* or *steps*
- One packet transmitted in each step



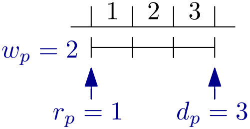
- Goal: maximize total weight of scheduled packets

Online Packet Scheduling with Deadlines

Network switch:

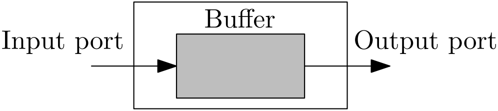


- Packets arrive over time (orders)
- Each has a deadline and a weight (reward)
- Time discrete, consisting of *slots* or *steps*
- One packet transmitted in each step (weeks)

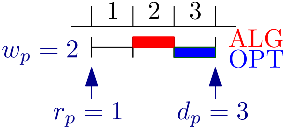


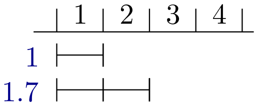
- Goal: maximize total weight of scheduled packets

Online Packet Scheduling with Deadlines

Network switch: 

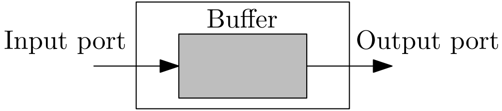
- Packets arrive over time (orders)
- Each has a deadline and a weight (reward)
- Time discrete, consisting of *slots* or *steps*
- One packet transmitted in each step (weeks)
- Goal: maximize total weight of scheduled packets



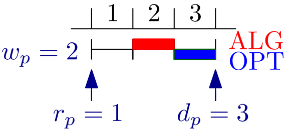


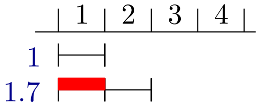
Online Packet Scheduling with Deadlines

Network switch:

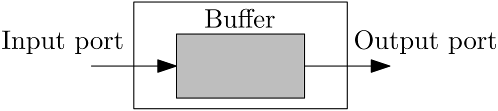


- Packets arrive over time (orders)
- Each has a deadline and a weight (reward)
- Time discrete, consisting of *slots* or *steps*
- One packet transmitted in each step (weeks)
- Goal: maximize total weight of scheduled packets

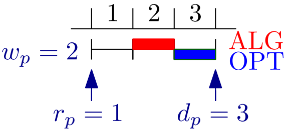


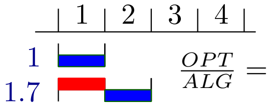


Network switch:



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

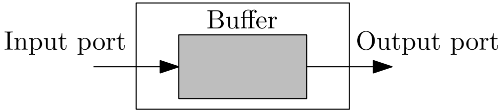




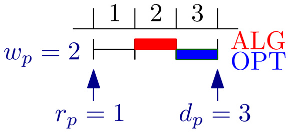
$$\frac{OPT}{ALG} = \frac{1.7+1}{1.7} \approx 1.59$$

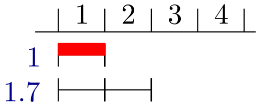
Online Packet Scheduling with Deadlines

Network switch:



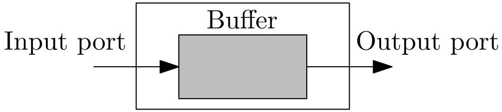
- Packets arrive over time (orders)
- Each has a deadline and a weight (reward)
- Time discrete, consisting of *slots* or *steps*
- One packet transmitted in each step (weeks)
- Goal: maximize total weight of scheduled packets



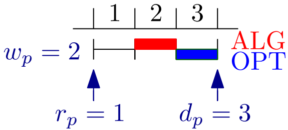


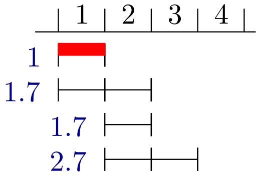
Online Packet Scheduling with Deadlines

Network switch:



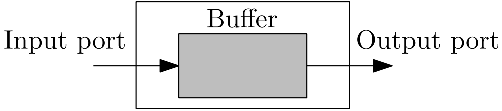
- Packets arrive over time (orders)
- Each has a deadline and a weight (reward)
- Time discrete, consisting of *slots* or *steps*
- One packet transmitted in each step (weeks)
- Goal: maximize total weight of scheduled packets



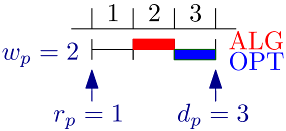


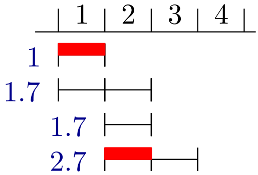
Online Packet Scheduling with Deadlines

Network switch:



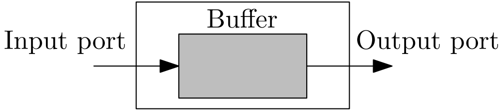
- Packets arrive over time (orders)
- Each has a deadline and a weight (reward)
- Time discrete, consisting of *slots* or *steps*
- One packet transmitted in each step (weeks)
- Goal: maximize total weight of scheduled packets



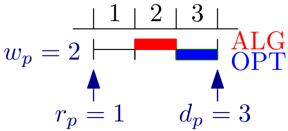


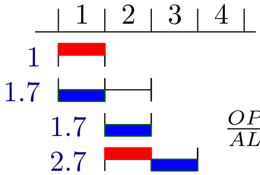
Online Packet Scheduling with Deadlines

Network switch:



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

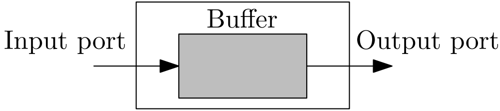




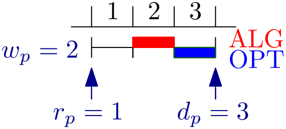
$$\frac{OPT}{ALG} = \frac{2\cdot 1.7 + 2.7}{1.7 + 2.7} \approx 1.39$$

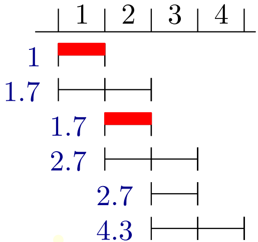
Online Packet Scheduling with Deadlines

Network switch:

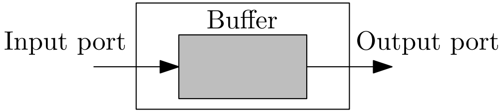


- Packets arrive over time (orders)
- Each has a deadline and a weight (reward)
- Time discrete, consisting of *slots* or *steps*
- One packet transmitted in each step (weeks)
- Goal: maximize total weight of scheduled packets

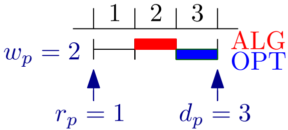




Network switch:



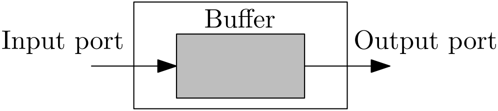
- Packets arrive over time (orders)
- Each has a deadline and a weight (reward)
- Time discrete, consisting of *slots* or *steps*
- One packet transmitted in each step (weeks)
- Goal: maximize total weight of scheduled packets



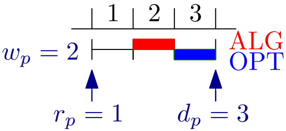
Scheduling problem $1|online, r_j, p_j = 1| \sum w_j(1 - U_j)$

Online Packet Scheduling with Deadlines

Network switch:



- Packets arrive over time (orders)
- Each has a deadline and a weight (reward)
- Time discrete, consisting of *slots* or *steps*
- One packet transmitted in each step (weeks)
- 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 Quality of Service Switches

Previous work

- We focus on deterministic algorithms

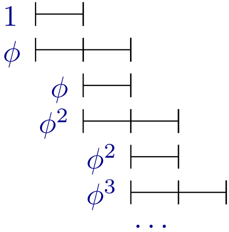
Previous work

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

Previous work

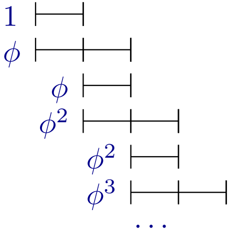
- We focus on deterministic algorithms
- Greedy algorithm 2-competitive
  - Schedules always the heaviest pending packet
- Lower bound of the golden ratio $\phi = \frac{1}{2}(\sqrt{5}+1) \approx 1.618$ [Hájek '01, Andelman *et al.* '03, Chin & Fung '03]

$$1 + \frac{1}{\phi} = \phi$$

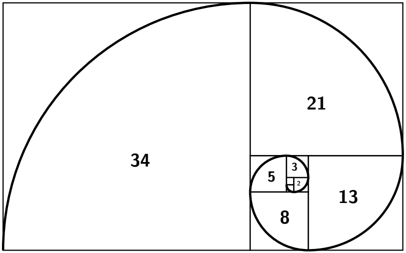


Previous work

- 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]



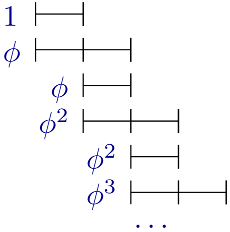
$$1 + \frac{1}{\phi} = \phi$$



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]

$$1 + \frac{1}{\phi} = \phi$$

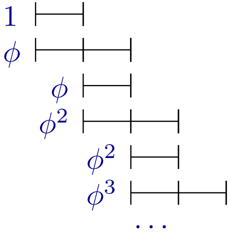


- $2\sqrt{2} - 1 \approx 1.828$-competitive algorithm [Englert & Westermann '07]

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]

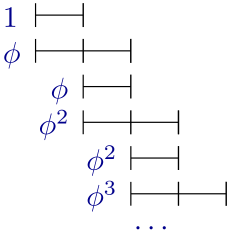
$$1 + \frac{1}{\phi} = \phi$$



  - $2\sqrt{2} - 1 \approx 1.828$-competitive algorithm [Englert & Westermann '07]
  - ϕ-competitive algorithms for some 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]



$$1 + \frac{1}{\phi} = \phi$$

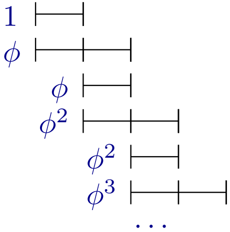
- $2\sqrt{2} - 1 \approx 1.828$-competitive algorithm [Englert & Westermann '07]
- ϕ-competitive algorithms for some special instances [Kesselman *et al.* '01, Chin *et al.* '04, Li *et al.* '05, Bienkowski *et al.* '13, Böhm *et al.* '16]

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]

$$1 + \frac{1}{\phi} = \phi$$



- $2\sqrt{2} - 1 \approx 1.828$-competitive algorithm [Englert & Westermann '07]
- ϕ-competitive algorithms for some 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

Algorithm $\textsc{Plan}(\phi)$

Plan $\mathcal{P}$

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

Algorithm $\textsc{Plan}(\phi)$

Plan $\mathcal{P}$

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

Algorithm $\textsc{Plan}(\phi)$

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

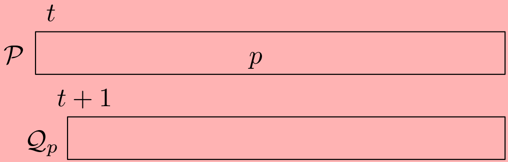
Algorithm $\textsc{Plan}(\phi)$

Plan $\mathcal{P}$

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

Algorithm $\textsc{Plan}(\phi)$

- 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 ($p \notin \mathcal{Q}_p$)



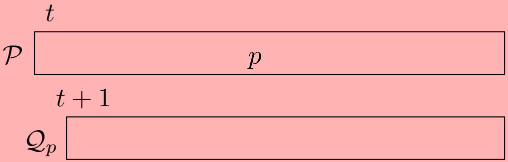
Algorithm $\textsc{Plan}(\phi)$

Plan $\mathcal{P}$

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

Algorithm $\textsc{Plan}(\phi)$

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

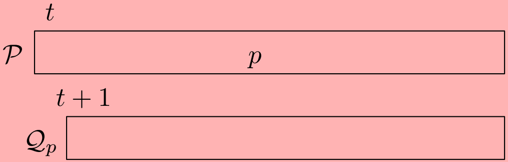
Algorithm $\textsc{Plan}(\phi)$

Plan $\mathcal{P}$

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

Algorithm $\textsc{Plan}(\phi)$

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

- Very elegant algorithm ...

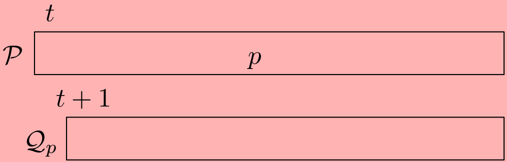
Algorithm $\textsc{Plan}(\phi)$

Plan $\mathcal{P}$

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

Algorithm $\textsc{Plan}(\phi)$

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

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

Plan and its Structure

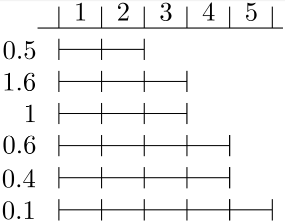
Plan $\mathcal{P}$

- 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 $\mathcal{P}$

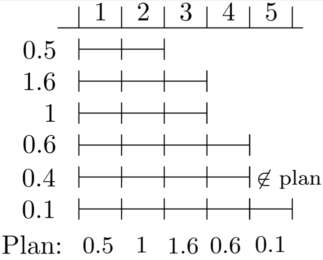
- Max-weight *feasible* subset of pending packets in step t
  - feasible = can be scheduled in slots $t, t+1, \ldots$



Plan and its Structure

Plan $\mathcal{P}$

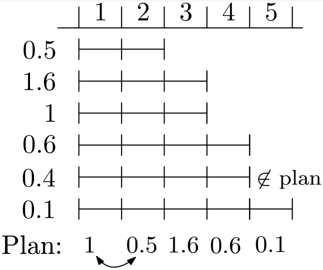
- Max-weight *feasible* subset of pending packets in step t
  - feasible = can be scheduled in slots $t, t+1, \ldots$



Plan and its Structure

Plan $\mathcal{P}$

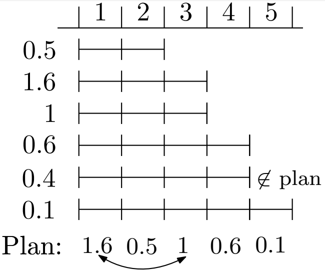
- Max-weight *feasible* subset of pending packets in step t
  - feasible = can be scheduled in slots $t, t+1, \ldots$



Plan and its Structure

Plan $\mathcal{P}$

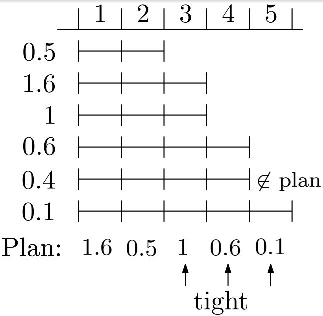
- Max-weight *feasible* subset of pending packets in step t
  - feasible = can be scheduled in slots $t, t+1, \ldots$



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

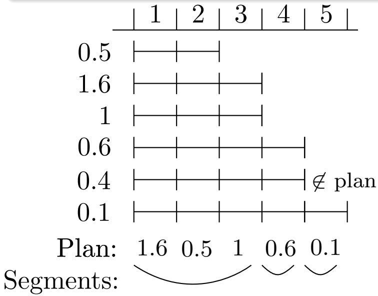
Slot τ is *tight* w.r.t. plan $\mathcal{P}$ iff

# of slots till τ = # 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* w.r.t. plan $\mathcal{P}$ iff

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

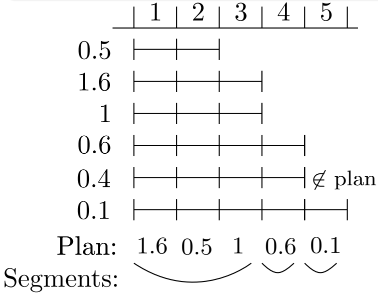
Definition

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* w.r.t. plan $\mathcal{P}$ iff

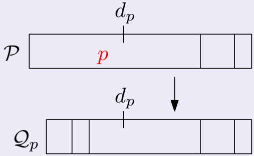
$\#$ of slots till $\tau = \#$ of packets $j \in \mathcal{P} : d_j \leq \tau$

Definition

Segment $=$ interval between tight slots

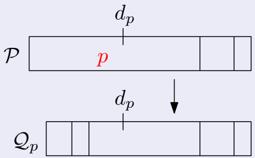
Plan Updates After Packet p is Scheduled

p in the 1st segment ("greedy step")

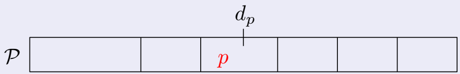


Plan Updates After Packet p is Scheduled

p in the 1st segment ("greedy step")

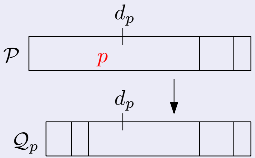


p in a later segment ("leap step")

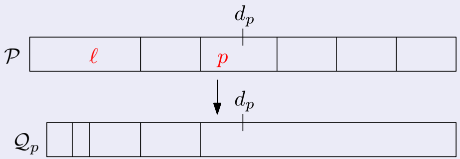


Plan Updates After Packet p is Scheduled

p in the 1st segment ("greedy step")



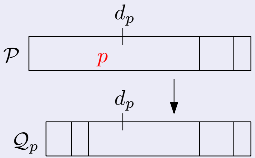
p in a later segment ("leap step")



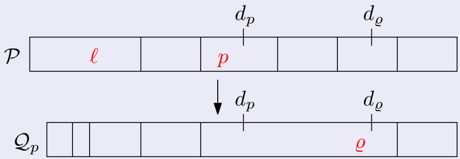
- ℓ = lightest in the 1st segment

Plan Updates After Packet p is Scheduled

p in the 1st segment ("greedy step")



p in a later segment ("leap step")



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

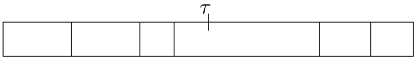
Problem of $\textsc{Plan}(\phi)$: Weight Decreases in the Plan

Problem of $\text{PLAN}(\phi)$: Weight Decreases in the Plan

- $\text{minwt}(\tau) =$ min-weight in $\mathcal{P}$ till the next tight slot after τ

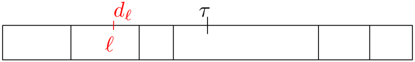
Problem of $\textsc{Plan}(\phi)$: Weight Decreases in the Plan

- minwt(τ) $=$ min-weight in $\mathcal{P}$ till the next tight slot after τ
  - In a schedule of $\mathcal{P}$, any packet can be in the 1st slot of a segment



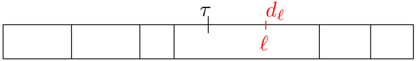
Problem of $\textsc{Plan}(\phi)$: Weight Decreases in the Plan

- $\mathrm{minwt}(\tau)$ = min-weight in $\mathcal{P}$ till the next tight slot after τ
  - In a schedule of $\mathcal{P}$, any packet can be in the 1st slot of a segment



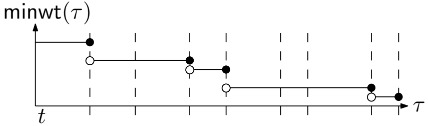
Problem of $\textsc{Plan}(\phi)$: Weight Decreases in the Plan

- minwt(τ) = min-weight in $\mathcal{P}$ till the next tight slot after τ
  - In a schedule of $\mathcal{P}$, any packet can be in the 1st slot of a segment



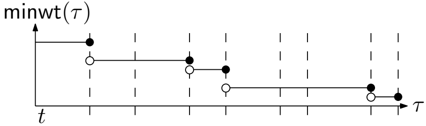
Problem of $\textsc{Plan}(\phi)$: Weight Decreases in the Plan

- minwt(τ) = min-weight in $\mathcal{P}$ till the next tight slot after τ
  - In a schedule of $\mathcal{P}$, any packet can be in the 1st slot of a segment



Problem of $\textsc{Plan}(\phi)$: Weight Decreases in the Plan

- minwt(τ) = min-weight in $\mathcal{P}$ till the next tight slot after τ
  - In a schedule of $\mathcal{P}$, any packet can be in the 1st slot of a segment

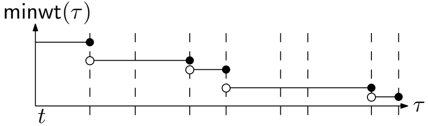


minwt after plan updates

- minwt(τ) does not decrease for any τ:
  - after arrival of a new packet
  - after scheduling a packet from the 1st segment (greedy step)

Problem of PLAN(ϕ): Weight Decreases in the Plan

- minwt(τ) = min-weight in $\mathcal{P}$ till the next tight slot after τ
  - In a schedule of $\mathcal{P}$, any packet can be in the 1st slot of a segment

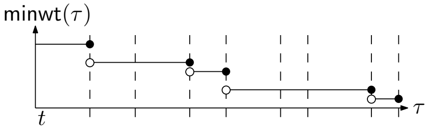


minwt after plan updates

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

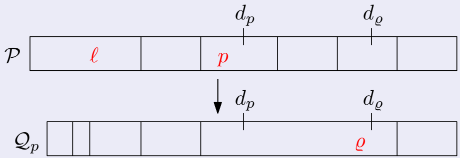
Problem of $\textsc{Plan}(\phi)$: Weight Decreases in the Plan

- minwt(τ) = min-weight in $\mathcal{P}$ till the next tight slot after τ
  - In a schedule of $\mathcal{P}$, any packet can be in the 1st slot of a segment



minwt after plan updates

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



The problem:

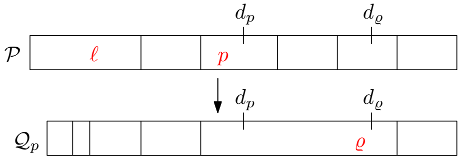
$\varrho \notin \mathcal{P} \Rightarrow w_\varrho < \text{minwt}(d_\varrho)$

Solution: Maintaining Slot-Monotonicity of minwt

- Idea: modify $\textsc{Plan}(\phi)$ so that $\mathrm{minwt}(\tau)$ never decreases for any τ

Solution: Maintaining Slot-Monotonicity of minwt

- Idea: modify $\textsc{Plan}(\phi)$ so that $\mathsf{minwt}(\tau)$ never decreases for any τ

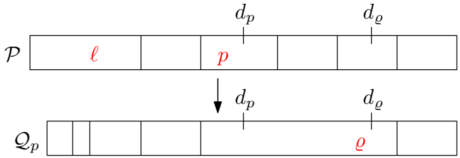


The problem:

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

Solution: Maintaining Slot-Monotonicity of minwt

- Idea: modify $\textsc{Plan}(\phi)$ so that minwt(τ) never decreases for any τ



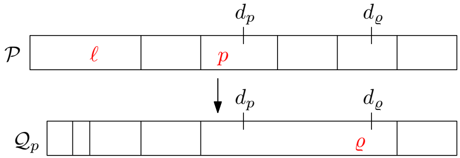
The problem:

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

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

Solution: Maintaining Slot-Monotonicity of minwt

- Idea: modify $\mathrm{P{\scriptsize LAN}}(\phi)$ so that $\mathsf{minwt}(\tau)$ never decreases for any τ



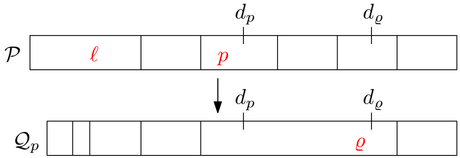
The problem:

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

- $\Rightarrow$ increase the weight of ϱ to $\mathsf{minwt}(d_\varrho)$
- Not enough if segments merge:

Solution: Maintaining Slot-Monotonicity of minwt

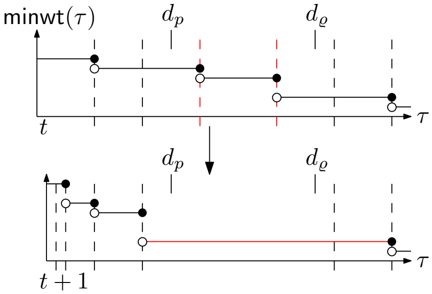
- Idea: modify $\textsc{Plan}(\phi)$ so that $\mathsf{minwt}(\tau)$ never decreases for any τ



The problem:

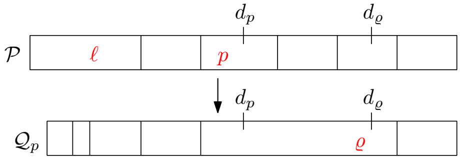
$\varrho \notin \mathcal{P} \Rightarrow w_\varrho < \mathsf{minwt}(d_\varrho)$

- $\Rightarrow$ increase the weight of ϱ to $\mathsf{minwt}(d_\varrho)$
- Not enough if segments merge:



Solution: Maintaining Slot-Monotonicity of minwt

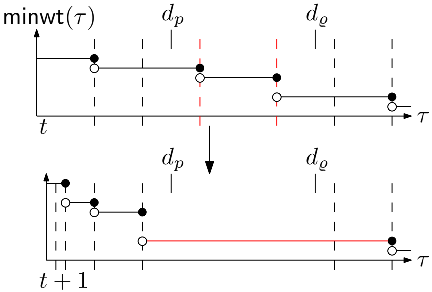
- Idea: modify $\text{Plan}(\phi)$ so that $\mathsf{minwt}(\tau)$ never decreases for any τ



The problem:

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

- $\Rightarrow$ increase the weight of ϱ to $\mathsf{minwt}(d_\varrho)$
- Not enough if segments merge:



- $\Rightarrow$ avoid merging segments

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

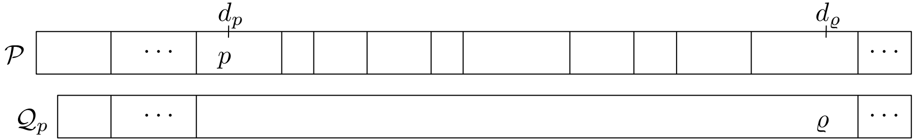
- Schedule packet $p \in \mathcal{P}$ maximizing $\phi \cdot w_p + w(\mathcal{Q}_p)$
  - $\mathcal{Q}_p$ is the plan after p is scheduled and time is incremented ($p \notin \mathcal{Q}_p$)

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

- Schedule packet $p \in \mathcal{P}$ maximizing $\phi \cdot w_p + w(\mathcal{Q}_p)$
  - $\mathcal{Q}_p$ is the plan after p is scheduled and time is incremented ($p \notin \mathcal{Q}_p$)
- If p is not in the 1st segment of $\mathcal{P}$ (leap step):
  - Increase the weight of ϱ to $\mathrm{minwt}(d_\varrho)$

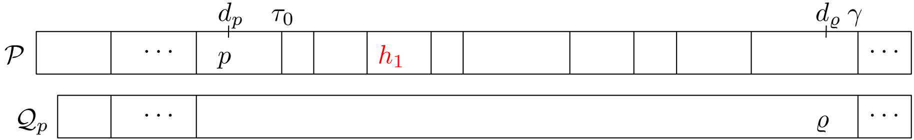
Algorithm PLANM(ϕ) Maintaining Slot-Monotonicity

- 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 ($p \notin \mathcal{Q}_p$)
- If p is not in the 1st segment of $\mathcal{P}$ (leap step):
  - Increase the weight of ϱ to minwt(d_ϱ)
  - Avoid merging segments:



Algorithm PLANM(ϕ) Maintaining Slot-Monotonicity

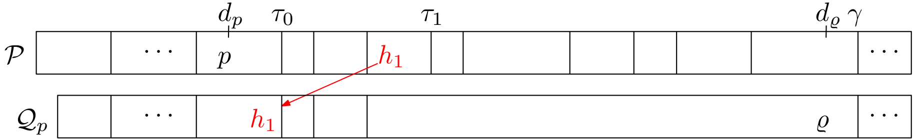
- Schedule packet $p \in \mathcal{P}$ maximizing $\phi \cdot w_p + w(\mathcal{Q}_p)$
  - $\mathcal{Q}_p$ is the plan after p is scheduled and time is incremented ($p \notin \mathcal{Q}_p$)
- If p is not in the 1st segment of $\mathcal{P}$ (leap step):
  - Increase the weight of ϱ to minwt(d_ϱ)
  - Avoid merging segments:



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

Algorithm PlanM(ϕ) Maintaining Slot-Monotonicity

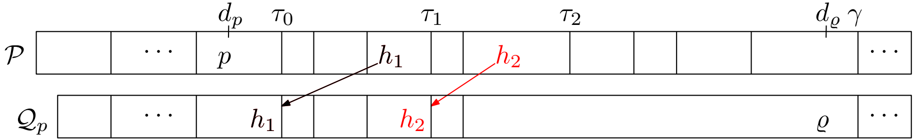
- Schedule packet $p \in \mathcal{P}$ maximizing $\phi \cdot w_p + w(\mathcal{Q}_p)$
  - $\mathcal{Q}_p$ is the plan after p is scheduled and time is incremented ($p \notin \mathcal{Q}_p$)
- If p is not in the 1st segment of $\mathcal{P}$ (leap step):
  - Increase the weight of ϱ to minwt(d_ϱ)
  - Avoid merging segments:



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

Algorithm PLANM(ϕ) Maintaining Slot-Monotonicity

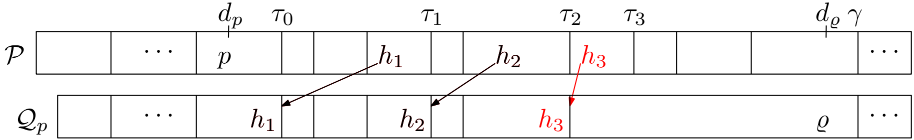
- Schedule packet $p \in \mathcal{P}$ maximizing $\phi \cdot w_p + w(\mathcal{Q}_p)$
  - $\mathcal{Q}_p$ is the plan after p is scheduled and time is incremented ($p \notin \mathcal{Q}_p$)
- If p is not in the 1st segment of $\mathcal{P}$ (leap step):
  - Increase the weight of ϱ to minwt(d_ϱ)
  - Avoid merging segments:



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

Algorithm PLANM(ϕ) Maintaining Slot-Monotonicity

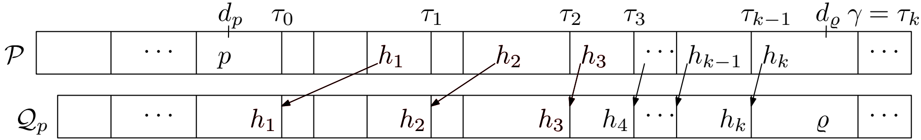
- Schedule packet $p \in \mathcal{P}$ maximizing $\phi \cdot w_p + w(\mathcal{Q}_p)$
  - $\mathcal{Q}_p$ is the plan after p is scheduled and time is incremented ($p \notin \mathcal{Q}_p$)
- If p is not in the 1st segment of $\mathcal{P}$ (leap step):
  - Increase the weight of ϱ to minwt(d_ϱ)
  - Avoid merging segments:



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

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

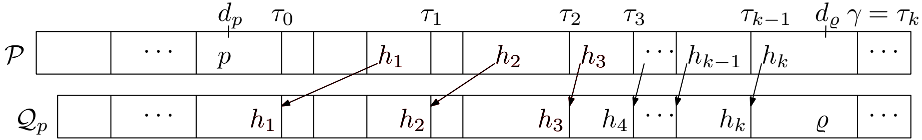
- Schedule packet $p \in \mathcal{P}$ maximizing $\phi \cdot w_p + w(\mathcal{Q}_p)$
  - $\mathcal{Q}_p$ is the plan after p is scheduled and time is incremented ($p \notin \mathcal{Q}_p$)
- If p is not in the 1st segment of $\mathcal{P}$ (leap step):
  - Increase the weight of ϱ to minwt(d_ϱ)
  - Avoid 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 $\phi \cdot w_p + w(\mathcal{Q}_p)$
  - $\mathcal{Q}_p$ is the plan after p is scheduled and time is incremented ($p \notin \mathcal{Q}_p$)
- If p is not in the 1st segment of $\mathcal{P}$ (leap step):
  - 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 $\tau_i = \gamma$
  - ensure: $w_{h_i} \geq$ minwt(τ_{i-1})
    - ★ if $w_{h_i} <$ minwt(τ_{i-1}), then set new weight of h_i to minwt(τ_{i-1})

Algorithm PLANM(ϕ) Maintaining Slot-Monotonicity

- 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 ($p \notin \mathcal{Q}_p$)
- If p is not in the 1st segment of $\mathcal{P}$ (leap step):
  - Increase the weight of ϱ to $\text{minwt}(d_\varrho)$

In a nutshell

Avoid merging segments and minwt decreases in a right way

- Done by decreasing deadlines and increasing weights of certain packets

Analysis

Analysis Overview

- Competitive analysis
  - Goal: $w(\mathrm{OPT}) \leq \phi \cdot w(\mathrm{ALG})$ for any instance

Analysis Overview

- Competitive analysis
  - Goal: $w(\text{OPT}) \leq \phi \cdot w(\text{ALG})$ for any instance
  - Game between algorithm and adversary 
    - Adversary schedules packets from OPT

Analysis Overview

- Competitive analysis
  - Goal: $w(\mathrm{OPT}) \leq \phi \cdot w(\mathrm{ALG})$ for any instance
  - Game between algorithm and adversary 
    - Adversary schedules packets from OPT

Amortization Techniques

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

Analysis Overview

- Competitive analysis
  - Goal: $w(\mathrm{OPT}) \le \phi \cdot w(\mathrm{ALG})$ for any instance
  - Game between algorithm and adversary 
    - Adversary schedules packets from OPT

Amortization Techniques

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

Analysis Overview

- Competitive analysis
  - Goal: $w(\mathrm{OPT}) \leq \phi \cdot w(\mathrm{ALG})$ for any instance
  - Game between algorithm and adversary 
    - Adversary schedules packets from OPT

Amortization Techniques

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

Adversary Schedule ADV

- Consists of already-released packets from OPT in future slots

Adversary Schedule ADV

- Consists of already-released packets from OPT in future slots

	1	2	3	4	5
ADV					
OPT	a	b	c	d	e

Adversary's gain:

Adversary Schedule ADV

- Consists of already-released packets from OPT in future slots

	1	2	3	4	5
ADV		b			
OPT	a	b	c	d	e

Adversary's gain:

Adversary Schedule ADV

- Consists of already-released packets from OPT in future slots

	1	2	3	4	5
ADV		b		d	
OPT	a	b	c	d	e

Adversary's gain:

Adversary Schedule ADV

- Consists of already-released packets from OPT in future slots

	1	2	3	4	5
ADV	a	b		d	
OPT	a	b	c	d	e

Adversary's gain:

Adversary Schedule ADV

- Consists of already-released packets from OPT in future slots

	1	2	3	4	5
ADV		b		d	
OPT	a	b	c	d	e

Adversary's gain: w_a

Adversary Schedule ADV

- Consists of already-released packets from OPT in future slots

	1	2	3	4	5
ADV		b	c	d	e
OPT	a	b	c	d	e

Adversary's gain: w_a

Adversary Schedule ADV

- Consists of already-released packets from OPT in future slots

	1	2	3	4	5
ADV			c	d	e
OPT	a	b	c	d	e

Adversary's gain: $w_a + w_b$

Adversary Schedule ADV

- Consists of already-released packets from OPT in future slots

	1	2	3	4	5
ADV			c	d	e
OPT	a	b	c	d	e

Adversary's gain: $w_a + w_b$

- Packet in ADV can be replaced by $\begin{cases}\text{another } \textit{lighter} \text{ packet,} \\ \end{cases}$

Adversary Schedule ADV

- Consists of already-released packets from OPT in future slots

	1	2	3	4	5
ADV			c	d	e
OPT	a	b	c	d	e

Adversary's gain: $w_a + w_b$

- Packet in ADV can be replaced by $\begin{cases} \text{another } \textit{lighter} \text{ packet,} \\ \text{fictitious ``treasure packets''} \end{cases}$ 

Adversary Schedule ADV

- Consists of already-released packets from OPT in future slots

	1	2	3	4	5
ADV			c	d	e
OPT	a	b	c	d	e

Adversary's gain: $w_a + w_b$

- Packet in ADV can be replaced by $\begin{cases} \text{another } \textit{lighter} \text{ packet,} \\ \text{fictitious "treasure packets"} \end{cases}$

- Adversary's gain increased by total weight decrease in ADV

Adversary Schedule ADV

- Consists of already-released packets from OPT in future slots

	1	2	3	4	5
ADV			c	f	e
OPT	a	b	c	d	e

Adversary's gain: $w_a + w_b + (w_d - w_f)$

- Packet in ADV can be replaced by
  - another *lighter* packet,
  - fictitious "treasure packets"

- Adversary's gain increased by total weight decrease in ADV

Adversary Schedule ADV

- Consists of already-released packets from OPT in future slots

	1	2	3	4	5
ADV			 t_c	f	e
OPT	a	b	c	d	e

Adversary's gain: $w_a + w_b + (w_d - w_f) + (w_c - w_{t_c})$

- Packet in ADV can be replaced by $\begin{cases} \text{another } \textit{lighter} \text{ packet,} \\ \text{fictitious “treasure packets”} \end{cases}$

- Adversary's gain increased by total weight decrease in ADV

Adversary Schedule ADV

- Consists of already-released packets from OPT in future slots

	1	2	3	4	5
ADV				f	e
OPT	a	b	c	d	e

Adversary's gain: $w_a + w_b + (w_d - w_f) + (w_c - w_{t_c}) + w_{t_c}$

- Packet in ADV can be replaced by $\begin{cases}\text{another } \textit{lighter} \text{ packet,} \\ \text{fictitious ``treasure packets''}\end{cases}$

- Adversary's gain increased by total weight decrease in ADV

Adversary Schedule ADV

- Consists of already-released packets from OPT in future slots

	1	2	3	4	5
ADV				f	e
OPT	a	b	c	d	e

Adversary's gain: $w_a + w_b + (w_d - w_f) + (w_c - w_{t_c}) + w_{t_c}$

- Packet in ADV can be replaced by $\begin{cases} \text{another } \textit{lighter} \text{ packet,} \\ \text{fictitious "treasure packets"} \end{cases}$

- Adversary's gain increased by total weight decrease in ADV

Fictitious "treasure packet"

- *Not* pending for the algorithm

Adversary Schedule ADV

- Consists of already-released packets from OPT in future slots

	1	2	3	4	5
ADV				f	e
OPT	a	b	c	d	e

Adversary's gain: $w_a + w_b + (w_d - w_f) + (w_c - w_{t_c}) + w_{t_c}$

- Packet in ADV can be replaced by $\begin{cases} \text{another } \textit{lighter} \text{ packet,} \\ \text{fictitious "treasure packets"} \end{cases}$

- Adversary's gain increased by total weight decrease in ADV

Fictitious "treasure packet"

- *Not* pending for the algorithm
- Tied to a slot τ in ADV, no release time or deadline, never changes in future

Adversary Schedule ADV

- Consists of already-released packets from OPT in future slots

	1	2	3	4	5
ADV				f	e
OPT	a	b	c	d	e

Adversary's gain: $w_a + w_b + (w_d - w_f) + (w_c - w_{t_c}) + w_{t_c}$

- Packet in ADV can be replaced by $\begin{cases} \text{another } \textit{lighter} \text{ packet,} \\ \text{fictitious "treasure packets"} \end{cases}$

- Adversary's gain increased by total weight decrease in ADV

Fictitious "treasure packet"

- *Not* pending for the algorithm
- Tied to a slot τ in ADV, no release time or deadline, never changes in future
- Deposit of profit to be collected by the adversary

Adversary Schedule ADV

- Consists of already-released packets from OPT in future slots

	1	2	3	4	5
ADV				f	e
OPT	a	b	c	d	e

Adversary's gain: $w_a + w_b + (w_d - w_f) + (w_c - w_{t_c}) + w_{t_c}$

- Packet in ADV can be replaced by $\begin{cases} \text{another } \textit{lighter} \text{ packet,} \\ \text{fictitious ``treasure packets''} \end{cases}$

- Adversary's gain increased by total weight decrease in ADV

Fictitious "treasure packet"

- *Not* pending for the algorithm
- Tied to a slot τ in ADV, no release time or deadline, never changes in future
- Deposit of profit to be collected by the adversary
- **Weight bounded by minwt**(τ)

Adversary Schedule ADV

- Consists of already-released packets from OPT in future slots

	1	2	3	4	5
ADV				f	e
OPT	a	b	c	d	e

Adversary's gain: $w_a + w_b + (w_d - w_f) + (w_c - w_{t_c}) + w_{t_c}$

- Packet in ADV can be replaced by $\begin{cases}\text{another } \textit{lighter} \text{ packet,} \\ \text{fictitious "treasure packets"}\end{cases}$

- Adversary's gain increased by total weight decrease in ADV

Fictitious "treasure packet"

- *Not* pending for the algorithm
- Tied to a slot τ in ADV, no release time or deadline, never changes in future
- Deposit of profit to be collected by the adversary
- **Weight bounded by minwt**(τ)
- Slot-monotonicity: minwt(τ) never decrease

Adversary Schedule ADV

- Consists of already-released packets from OPT in future slots
- Packet in ADV can be replaced by
  - another *lighter* packet,
  - fictitious "treasure packets" 
- Adversary's gain increased by total weight decrease in ADV

Fictitious "treasure packet" 

- *Not* pending for the algorithm
- Tied to a slot τ in ADV, no release time or deadline, never changes in future
- Deposit of profit to be collected by the adversary
- **Weight bounded by minwt**(τ)
- Slot-monotonicity: minwt(τ) never decrease

Invariant (A)

ADV consists of two types of packets:
- (real) packets in plan P
- all other packets are treasures 

Potential Function

Relative advantage of the algorithm over the adversary:

Potential Function

Relative advantage of the algorithm over the adversary:

- $\mathcal{P} \setminus \mathsf{ADV}$ = packets in the plan that the adversary will not schedule

Potential Function

Relative advantage of the algorithm over the adversary:

- $\mathcal{P} \setminus \mathrm{ADV}$ = packets in the plan that the adversary will not schedule
- Set $\mathcal{F}$
  - Pending packets forced out of the plan

Potential Function

Relative advantage of the algorithm over the adversary:

- $\mathcal{P} \setminus \mathsf{ADV}$ = packets in the plan that the adversary will not schedule
- Set $\mathcal{F}$
  - Pending packets forced out of the plan
  - Can be used as replacement packets in a leap step

Potential Function

Relative advantage of the algorithm over the adversary:

- $\mathcal{P} \setminus \mathsf{ADV}$ = packets in the plan that the adversary will not schedule
- Set $\mathcal{F}$
  - Pending packets forced out of the plan
  - Can be used as replacement packets in a leap step
- "Backup plan" $R = \mathcal{P} \setminus \mathsf{ADV} \cup R$

Potential Function

Relative advantage of the algorithm over the adversary:

- $\mathcal{P} \setminus \mathsf{ADV}$ = packets in the plan that the adversary will not schedule
- Set $\mathcal{F}$
  - Pending packets forced out of the plan
  - Can be used as replacement packets in a leap step
- "Backup plan" $R = \mathcal{P} \setminus \mathsf{ADV} \cup R$

Invariant

Backup plan R is feasible

R feasible = packets in R can be scheduled in future slots $t, t+1, \ldots$

Potential Function

Relative advantage of the algorithm over the adversary:

- $\mathcal{P} \setminus \mathrm{ADV}$ = packets in the plan that the adversary will not schedule
- Set $\mathcal{F}$
  - Pending packets forced out of the plan
  - Can be used as replacement packets in a leap step
- "Backup plan" $R = \mathcal{P} \setminus \mathrm{ADV} \cup R$

Invariant

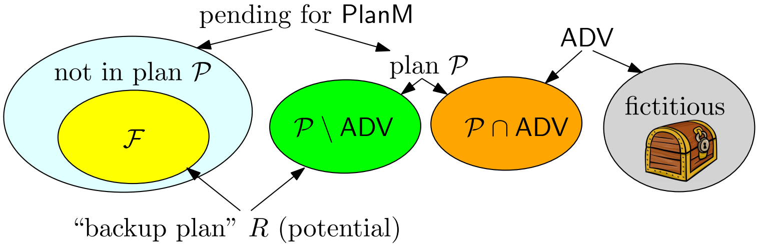
Backup plan R is feasible

R feasible = packets in R can be scheduled in future slots $t, t+1, \ldots$

Potential

$$\Psi := \frac{1}{\phi}\, w(R)$$

Packet Types in the Analysis



Overview of the Analysis

To prove

- Packet arrival: $\Delta\Psi \geq 0$

Overview of the Analysis

To prove

- Packet arrival: $\Delta\Psi \geq 0$
- Scheduling step t
  - $j = \mathrm{ADV}[t]$ scheduled by the adversary (possibly $j \neq \mathrm{OPT}[t]$)
  - $p = \mathrm{ALG}[t]$ scheduled by the algorithm

Overview of the Analysis

To prove

- Packet arrival: $\Delta\Psi \geq 0$
- Scheduling step t
  - $j = \mathsf{ADV}[t]$ scheduled by the adversary (possibly $j \neq \mathsf{OPT}[t]$)
  - $p = \mathsf{ALG}[t]$ scheduled by the algorithm
  - Adversary gain $\mathsf{advgain}^t = w_j^t +$ credit for replacing packets

Overview of the Analysis

To prove

- Packet arrival: $\Delta\Psi \geq 0$
- Scheduling step t
  - $j = \mathrm{ADV}[t]$ scheduled by the adversary (possibly $j \neq \mathrm{OPT}[t]$)
  - $p = \mathrm{ALG}[t]$ scheduled by the algorithm
  - Adversary gain $\mathrm{advgain}^t = w_j^t +$ credit for replacing packets
  - $\Delta^t\mathrm{Weights} =$ amount by which the weights are increased in step t

$$\mathrm{advgain}^t \;\leq\; \phi \cdot (w_p^t - \Delta^t\mathrm{Weights}) + \Delta\Psi$$

Overview of the Analysis

To prove

- Packet arrival: $\Delta\Psi \geq 0$
- Scheduling step t
  - $j = \mathsf{ADV}[t]$ scheduled by the adversary (possibly $j \neq \mathsf{OPT}[t]$)
  - $p = \mathsf{ALG}[t]$ scheduled by the algorithm
  - Adversary gain $\mathsf{advgain}^t = w_j^t +$ credit for replacing packets
  - $\Delta^t\mathsf{Weights} =$ amount by which the weights are increased in step t

$$\mathsf{advgain}^t \;\leq\; \phi \cdot (w_p^t - \Delta^t\mathsf{Weights}) + \Delta\Psi$$

Proof of ϕ-competitiveness

- Potential equal to 0 at the beginning and at the end

Overview of the Analysis

To prove

- Packet arrival: $\Delta\Psi \geq 0$
- Scheduling step t
  - $j = \mathrm{ADV}[t]$ scheduled by the adversary (possibly $j \neq \mathrm{OPT}[t]$)
  - $p = \mathrm{ALG}[t]$ scheduled by the algorithm
  - Adversary gain $\mathrm{advgain}^t = w_j^t +$ credit for replacing packets
  - $\Delta^t\mathrm{Weights} =$ amount by which the weights are increased in step t

$$\mathrm{advgain}^t \;\leq\; \phi \cdot (w_p^t - \Delta^t\mathrm{Weights}) + \Delta\Psi$$

Proof of ϕ-competitiveness

- Potential equal to 0 at the beginning and at the end

$$w^0(\mathrm{OPT}) = \sum_t \mathrm{advgain}^t$$

Overview of the Analysis

To prove

- Packet arrival: $\Delta\Psi \geq 0$
- Scheduling step t
  - $j = \text{ADV}[t]$ scheduled by the adversary (possibly $j \neq \text{OPT}[t]$)
  - $p = \text{ALG}[t]$ scheduled by the algorithm
  - Adversary gain $\text{advgain}^t = w_j^t +$ credit for replacing packets
  - $\Delta^t\text{Weights} =$ amount by which the weights are increased in step t

$$\text{advgain}^t \ \leq \ \phi \cdot (w_p^t - \Delta^t\text{Weights}) + \Delta\Psi$$

Proof of ϕ-competitiveness

- Potential equal to 0 at the beginning and at the end

$$w^0(\text{OPT}) = \sum_t \text{advgain}^t \leq \sum_t \left[\phi \cdot (w^t(\text{ALG}[t]) - \Delta^t\text{Weights})\right]$$

Overview of the Analysis

To prove

- Packet arrival: $\Delta\Psi \geq 0$
- Scheduling step t
  - $j = \text{ADV}[t]$ scheduled by the adversary (possibly $j \neq \text{OPT}[t]$)
  - $p = \text{ALG}[t]$ scheduled by the algorithm
  - Adversary gain $\text{advgain}^t = w_j^t +$ credit for replacing packets
  - $\Delta^t\text{Weights} =$ amount by which the weights are increased in step t

$$\text{advgain}^t \;\leq\; \phi \cdot (w_p^t - \Delta^t\text{Weights}) + \Delta\Psi$$

Proof of ϕ-competitiveness

- Potential equal to 0 at the beginning and at the end

$$w^0(\text{OPT}) = \sum_t \text{advgain}^t \leq \sum_t \left[\phi \cdot (w^t(\text{ALG}[t]) - \Delta^t\text{Weights})\right] \leq \phi \cdot w^0(\text{ALG})$$

Conclusions

Summary

$\phi \approx 1.618$-competitive deterministic algorithm

- 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 ($p \notin \mathcal{Q}_p$)

Summary

$\phi \approx 1.618$-competitive deterministic algorithm

- 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 ($p \notin \mathcal{Q}_p$)
- Maintain slot-monotonicity of minwt
  - Done by increasing weights and decreasing deadlines of certain packets

Summary

$\phi \approx 1.618$-competitive deterministic algorithm

- 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 ($p \notin \mathcal{Q}_p$)
- Maintain slot-monotonicity of minwt
  - Done by increasing weights and decreasing deadlines of certain packets

Analysis

- Potential function
  - Advantage of the algorithm over the adversary in future steps
  - Invariant ensures that this advantage is feasible

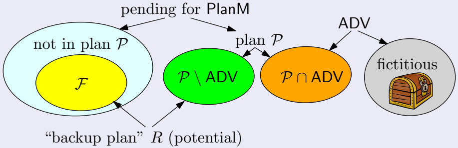
Summary

$\phi \approx 1.618$-competitive deterministic algorithm

- 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 ($p \notin \mathcal{Q}_p$)
- Maintain slot-monotonicity of minwt
  - Done by increasing weights and decreasing deadlines of certain packets

Analysis

- Potential function
  - Advantage of the algorithm over the adversary in future steps
  - Invariant ensures that this advantage is feasible
- Modifications of adversary schedule to maintain certain invariants



Further Research Directions

$m \geq 1$ packets are sent in each step

- Our algorithm is $\phi \approx 1.618$-competitive for any $m \geq 1$
- The best algorithm has ratio $\frac{1}{1-\left(\frac{m}{m+1}\right)^m} \to \frac{e}{e-1} \approx 1.58$ [Chin *et al.* '04]
- Can our algorithm be modified to give a better ratio for $m > 1$?

Further Research Directions

$m \geq 1$ packets are sent in each step

- Our algorithm is $\phi \approx 1.618$-competitive for any $m \geq 1$
- The best algorithm has ratio $\frac{1}{1-\left(\frac{m}{m+1}\right)^m} \rightarrow \frac{e}{e-1} \approx 1.58$ [Chin *et al.* '04]
- Can our algorithm be modified to give a better ratio for $m > 1$?

Randomized algorithms

- Improve randomized algorithms using plans
- 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 are sent in each step

- Our algorithm is $\phi \approx 1.618$-competitive for any $m \geq 1$
- The best algorithm has ratio $\frac{1}{1-\left(\frac{m}{m+1}\right)^m} \to \frac{e}{e-1} \approx 1.58$ [Chin *et al.* '04]
- Can our algorithm be modified to give a better ratio for $m > 1$?

Randomized algorithms

- Improve randomized algorithms using plans
- 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 $\textsc{Plan}(\alpha)$? (Schedule $p \in \mathcal{P}$ max. $\alpha \cdot w_p + w(\mathcal{Q}_p)$)

Further Research Directions

$m \geq 1$ packets are sent in each step

- Our algorithm is $\phi \approx 1.618$-competitive for any $m \geq 1$
- The best algorithm has ratio $\frac{1}{1-\left(\frac{m}{m+1}\right)^m} \to \frac{e}{e-1} \approx 1.58$ [Chin *et al.* '04]
- Can our algorithm be modified to give a better ratio for $m > 1$?

Randomized algorithms 

- Improve randomized algorithms using plans
- 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 $\textsc{Plan}(\alpha)$? (Schedule $p \in \mathcal{P}$ max. $\alpha \cdot w_p + w(\mathcal{Q}_p)$)

Thank you!