Steady-State Scheduling

Frédéric Vivien

October 9, 2013

Overview

1. The context
2. Routing packets with fixed communication routes
3. Resolution of the “fluidified” problem
4. Building a schedule
5. Packet routing without fixed path
6. Bags of sequential applications

Overview

Platform

Platform: heterogeneous and distributed:

- processors with different capabilities;
- communication links of different characteristics.

Applications

Application made of a very (very) large number of tasks, the tasks can be clustered into a finite number of types, all tasks of a same type having the same characteristics.

Bag-of-tasks applications, parameter sweep applications, etc.

Principle

When we have a very large number of identical tasks to execute, we can imagine that, after some initiation phase, we will reach a (long) steady-state, before a termination phase.

If the steady-state is long enough, the initiation and termination phases will be negligible.

Overview

The problem

Problem: sending a set of message flows.

In a communication network, several flow of packets must be dispatched, each packet flow must be sent from a source to a destination, while following a given path linking the source to the destination.

Notations

- (V, A) a directed graph, representing the communication network.
- A set of n_c flows which must be dispatched.
- The k-th flow is denoted (s_k, t_k, P_k, n_k), where
  - s_k is the source of packets;
  - t_k is the destination;
  - P_k is the path to be followed;
    We denote by $a_{k,i}$ the i-th edge in the path P_k.
  - n_k is the number of packets in the flow.

Hypotheses

- A packet goes through an edge A in a unit of time.
- At a given time, a single packet traverses a given edge.

Objective

We must decide which packet must go through a given edge at a given time, in order to minimize the overall execution time.

Lower bound on the duration of schedules

We call **congestion** of edge $a \in A$, and we denote by C_a, the total number of packets which go through edge a:

$$C_a = \sum_{k \mid a \in P_k} n_k \qquad C_{\max} = \max_a C_a$$

Lower bound on the duration of schedules

We call **congestion** of edge $a \in A$, and we denote by C_a, the total number of packets which go through edge a:

$$C_a = \sum_{k \mid a \in P_k} n_k \qquad C_{\max} = \max_a C_a$$

$C_{\max}$ is a lower bound on the execution time of any schedule.

$$C^* \geq C_{\max}$$

Lower bound on the duration of schedules

We call **congestion** of edge $a \in A$, and we denote by C_a, the total number of packets which go through edge a:

$$C_a = \sum_{k \;|\; a \in P_k} n_k \qquad C_{\max} = \max_a C_a$$

$C_{\max}$ is a lower bound on the execution time of any schedule.

$$C^* \geq C_{\max}$$

A "fluid" (fractional) resolution of our problem will give us a solution which executes in a time $C_{\max}$.

Overview

Fluidified (fractional) version: notations

Principle:

- we do not look for an integral solution but for a rational one.

- $n_{k,i}(t)$ (fractional) number of packets waiting at the entrance of the i-th edge of the k-th path, at time t.

- $T_{k,i}(t)$ is the overall time used by the edge $a_{k,i}$ for packets of the k-th flow, during the interval of time $[0; t]$.

Fluidified (fractional) version: writing the equations

1. Initiating the communications

$$n_{k,1}(t) = n_k - T_{k,1}(t), \qquad \text{for each } k$$

Fluidified (fractional) version: writing the equations

1. Initiating the communications

$$n_{k,1}(t) = n_k - T_{k,1}(t), \qquad \text{for each } k$$

2. Conservation law

$$n_{k,i+1}(t) = T_{k,i}(t) - T_{k,i+1}(t), \qquad \text{for each } k$$

Fluidified (fractional) version: writing the equations

1. Initiating the communications

$$n_{k,1}(t) = n_k - T_{k,1}(t), \qquad \text{for each } k$$

2. Conservation law

$$n_{k,i+1}(t) = T_{k,i}(t) - T_{k,i+1}(t), \qquad \text{for each } k$$

3. Resource constraints

$$\sum_{(k,i) \ | \ a_{k,i}=a} T_{k,i}(t_2) - T_{k,i}(t_1) \leq t_2 - t_1, \forall a \in A, \forall t_2 \geq t_1 \geq 0$$

Fluidified (fractional) version: writing the equations

1. Initiating the communications

$$n_{k,1}(t) = n_k - T_{k,1}(t), \qquad \text{for each } k$$

2. Conservation law

$$n_{k,i+1}(t) = T_{k,i}(t) - T_{k,i+1}(t), \qquad \text{for each } k$$

3. Resource constraints

$$\sum_{(k,i) \ | \ a_{k,i}=a} T_{k,i}(t_2) - T_{k,i}(t_1) \leq t_2 - t_1, \forall a \in A, \forall t_2 \geq t_1 \geq 0$$

4. Objective

$$\textsc{Minimize } C_{\text{frac}} = \int_0^\infty \mathbb{1}\left(\sum_{k,i} n_{k,i}(t)\right) dt$$

Lower bound

- $n_{k,1}(t) = n_k - T_{k,1}(t), \qquad$ for each k
- $n_{k,i+1}(t) = T_{k,i}(t) - T_{k,i+1}(t), \qquad$ for each k

Lower bound

- $n_{k,1}(t) = n_k - T_{k,1}(t), \qquad$ for each k
- $n_{k,i+1}(t) = T_{k,i}(t) - T_{k,i+1}(t), \qquad$ for each k
- At any time t, $\sum_{j=1}^{i} n_{k,j}(t) = n_k - T_{k,i}(t)$

Lower bound

- $n_{k,1}(t) = n_k - T_{k,1}(t), \qquad$ for each k
- $n_{k,i+1}(t) = T_{k,i}(t) - T_{k,i+1}(t), \qquad$ for each k
- At any time t, $\sum_{j=1}^{i} n_{k,j}(t) = n_k - T_{k,i}(t)$
- For each edge a:

$$\sum_{(k,i)|a_{k,i}=a} \sum_{j=1}^{i} n_{k,j}(t) = \sum_{(k,i)|a_{k,i}=a} n_k - \sum_{(k,i)|a_{k,i}=a} T_{k,i}(t)$$

Lower bound

- $n_{k,1}(t) = n_k - T_{k,1}(t), \qquad$ for each k
- $n_{k,i+1}(t) = T_{k,i}(t) - T_{k,i+1}(t), \qquad$ for each k
- At any time t, $\sum_{j=1}^{i} n_{k,j}(t) = n_k - T_{k,i}(t)$
- For each edge a:

$$\sum_{(k,i)|a_{k,i}=a} \sum_{j=1}^{i} n_{k,j}(t) = \sum_{(k,i)|a_{k,i}=a} n_k - \sum_{(k,i)|a_{k,i}=a} T_{k,i}(t) \geq C_a - t$$

As long as $t < C_a$, there are packets in the system.

Therefore, $C_{\mathsf{frac}} \geq \max_a C_a = C_{\max}$

A candidate solution

For $t \leq C_{\max}$

A candidate solution

For $t \leq C_{\max}$

- $T_{k,i}(t) = \dfrac{n_k}{C_{\max}} t$, for each k and i.

A candidate solution

For $t \leq C_{\max}$

- $T_{k,i}(t) = \dfrac{n_k}{C_{\max}} t$, for each k and i.
- $n_{k,1}(t) = n_k - T_{k,1}(t) = n_k - \dfrac{n_k}{C_{\max}} t = n_k \left(1 - \dfrac{t}{C_{\max}}\right), \quad \forall k$

A candidate solution

For $t \leq C_{\max}$

- $T_{k,i}(t) = \dfrac{n_k}{C_{\max}} t$, for each k and i.
- $n_{k,1}(t) = n_k - T_{k,1}(t) = n_k - \dfrac{n_k}{C_{\max}} t = n_k \left(1 - \dfrac{t}{C_{\max}}\right), \quad \forall k$
- $n_{k,i}(t) = 0$, for each k and $i \geq 2$.

A candidate solution

For $t \leq C_{\max}$

- $T_{k,i}(t) = \frac{n_k}{C_{\max}} t$, for each k and i.
- $n_{k,1}(t) = n_k - T_{k,1}(t) = n_k - \frac{n_k}{C_{\max}} t = n_k \left(1 - \frac{t}{C_{\max}}\right), \quad \forall k$
- $n_{k,i}(t) = 0$, for each k and $i \geq 2$.

For $t \geq C_{\max}$

A candidate solution

For $t \le C_{\max}$

- $T_{k,i}(t) = \frac{n_k}{C_{\max}} t$, for each k and i.
- $n_{k,1}(t) = n_k - T_{k,1}(t) = n_k - \frac{n_k}{C_{\max}} t = n_k \left(1 - \frac{t}{C_{\max}}\right), \quad \forall k$
- $n_{k,i}(t) = 0$, for each k and $i \ge 2$.

For $t \ge C_{\max}$

- $T_{k,i}(t) = n_k$

A candidate solution

For $t \leq C_{\max}$

- $T_{k,i}(t) = \dfrac{n_k}{C_{\max}} t$, for each k and i.
- $n_{k,1}(t) = n_k - T_{k,1}(t) = n_k - \dfrac{n_k}{C_{\max}} t = n_k \left(1 - \dfrac{t}{C_{\max}}\right), \quad \forall k$
- $n_{k,i}(t) = 0$, for each k and $i \geq 2$.

For $t \geq C_{\max}$

- $T_{k,i}(t) = n_k$
- $n_{k,i}(t) = 0$

A candidate solution

For $t \leq C_{\max}$

- $T_{k,i}(t) = \dfrac{n_k}{C_{\max}} t$, for each k and i.
- $n_{k,1}(t) = n_k - T_{k,1}(t) = n_k - \dfrac{n_k}{C_{\max}} t = n_k \left(1 - \dfrac{t}{C_{\max}}\right), \quad \forall k$
- $n_{k,i}(t) = 0$, for each k and $i \geq 2$.

For $t \geq C_{\max}$

- $T_{k,i}(t) = n_k$
- $n_{k,i}(t) = 0$

This solution is a schedule of makespan $C_{\max}$. We still have to show that it is feasible.

Checking the solution (for $t \leq C_{\max}$)

1. $n_{k,1}(t) = n_k - T_{k,1}(t), \qquad$ for each k
   Satisfied by definition.

Checking the solution (for $t \leq C_{\max}$)

1. $n_{k,1}(t) = n_k - T_{k,1}(t), \qquad$ for each k
   Satisfied by definition.

2. $n_{k,i+1}(t) = T_{k,i}(t) - T_{k,i+1}(t), \qquad$ for each k
   $T_{k,i}(t) - T_{k,i+1}(t) = \frac{n_k}{C_{\max}}t - \frac{n_k}{C_{\max}}t = 0 = n_{k,i+1}(t)$

Checking the solution (for $t \le C_{\max}$)

1. $n_{k,1}(t) = n_k - T_{k,1}(t), \qquad$ for each k
   Satisfied by definition.

2. $n_{k,i+1}(t) = T_{k,i}(t) - T_{k,i+1}(t), \qquad$ for each k
   $T_{k,i}(t) - T_{k,i+1}(t) = \frac{n_k}{C_{\max}}t - \frac{n_k}{C_{\max}}t = 0 = n_{k,i+1}(t)$

3. $$\sum_{(k,i)\ \mid\ a_{k,i}=a} T_{k,i}(t_2) - T_{k,i}(t_1) \le t_2 - t_1, \forall a \in A, \forall t_2 \ge t_1 \ge 0$$
   $$\sum_{(k,i)\ \mid\ a_{k,i}=a} T_{k,i}(t_2) - T_{k,i}(t_1) = \sum_{(k,i)\ \mid\ a_{k,i}=a} \frac{n_k}{C_{\max}}(t_2 - t_1) =$$
   $$\frac{C_a}{C_{\max}}(t_2 - t_1) \le t_2 - t_1$$

Overview

Definition of a round

- $\Omega \approx$ duration of a round (will be defined later).

Definition of a round

- $\Omega \approx$ duration of a round (will be defined later).
- m_k: number of packets of k-th flow distributed in a single round.

$$m_k = \left\lceil \frac{n_k \Omega}{C_{\max}} \right\rceil.$$

Definition of a round

- $\Omega \approx$ duration of a round (will be defined later).
- m_k: number of packets of k-th flow distributed in a single round.

$$m_k = \left\lceil \frac{n_k \Omega}{C_{\max}} \right\rceil .$$

- $D_a = \sum_{(k,i)|a_{k,i}=a} 1 = |\{k|a \in P_k\}|$

$$D_{\max} = \max_a D_a \leq n_c$$

Definition of a round

- $\Omega \approx$ duration of a round (will be defined later).
- m_k: number of packets of k-th flow distributed in a single round.

$$m_k = \left\lceil \frac{n_k \Omega}{C_{\max}} \right\rceil .$$

- $D_a = \sum_{(k,i)|a_{k,i}=a} 1 = |\{k|a \in P_k\}|$

$$D_{\max} = \max_a D_a \leq n_c$$

- Period of the schedule: $\Omega + D_{\max}$.

Schedule

During the time interval $[j(\Omega + D_{\max}); (j+1)(\Omega + D_{\max})]$:

Schedule

During the time interval $[j(\Omega + D_{\max}); (j+1)(\Omega + D_{\max})]$:

The link a forwards m_k packets of the k-th flow if there exists i such that $a_{k,i} = a$.

Schedule

During the time interval $[j(\Omega + D_{\max}); (j+1)(\Omega + D_{\max})]$:

The link a forwards m_k packets of the k-th flow if there exists i such that $a_{k,i} = a$.

The link a remains idle for a duration of:

$$\Omega + D_{\max} - \sum_{(k,i) | a_{k,i} = a} m_k$$

Schedule

During the time interval $[j(\Omega + D_{\max}); (j+1)(\Omega + D_{\max})]$:

The link a forwards m_k packets of the k-th flow if there exists i such that $a_{k,i} = a$.

The link a remains idle for a duration of:

$$\Omega + D_{\max} - \sum_{(k,i) | a_{k,i} = a} m_k$$

(If less than m_k packets are waiting in the entrance of a at time $j(\Omega + D_{\max})$, a forwards what is available and remains idle longer.)

Feasibility of the schedule

$$\begin{aligned}
\sum_{(k,i)|a_{k,i}=a} m_k &= \sum_{(k,i)|a_{k,i}=a} \left\lceil \frac{n_k \Omega}{C_{\max}} \right\rceil \\
&\leq \sum_{(k,i)|a_{k,i}=a} \left(\frac{n_k \Omega}{C_{\max}} + 1 \right) \\
&\leq \frac{C_a}{C_{\max}} \Omega + D_a \\
&\leq \Omega + D_{\max}
\end{aligned}$$

Behavior of the sources

- $N_{k,i}(t)$: number of packets of the k-th flow waiting at the entrance of the i-th edge, at time t.
- $a_{k,1}$ sends m_k packets during $[0, \Omega + D_{\max}]$.
  $N_{k,1}(\Omega + D_{\max}) = n_k - m_k$
- $a_{k,1}$ sends m_k packets during $[\Omega + D_{\max}, 2(\Omega + D_{\max})]$.
  $N_{k,1}(2(\Omega + D_{\max})) = n_k - 2m_k$
- We let $T = \left\lceil \dfrac{C_{\max}}{\Omega} \right\rceil (\Omega + D_{\max})$

$$N_{k,1}(T) \leq n_k - \frac{T}{\Omega + D_{\max}} m_k \leq n_k - \frac{n_k \Omega}{C_{\max}} \frac{C_{\max}}{\Omega} = 0$$

Propagation delay

- $a_{k,1}$ sends m_k packets during $[0, \Omega + D_{\max}]$.
  $N_{k,1}(\Omega + D_{\max}) = n_k - m_k \qquad N_{k,2}(\Omega + D_{\max}) = m_k$
  $N_{k,i\geq 3}(\Omega + D_{\max}) = 0$

- $a_{k,1}$ sends m_k packets during $[\Omega + D_{\max}, 2(\Omega + D_{\max})]$.
  $N_{k,1}(2(\Omega + D_{\max})) = n_k - 2m_k \qquad N_{k,2}(2(\Omega + D_{\max})) = m_k$
  $N_{k,3}(2(\Omega + D_{\max})) = m_k \qquad N_{k,i\geq 4}(2(\Omega + D_{\max})) = 0$

- The delay between the time a packet traverses the first edge of the path P_k and the time it traverses its last edge is, at worst:
  $$(|P_k| - 1)(\Omega + D_{\max})$$
  We let $L = \max_k |P_k|$.

Makespan of the schedule

$$
\begin{aligned}
C_{\text{total}} &\leq T + (L-1)(\Omega + D_{\max}) \\
&= \left\lceil \frac{C_{\max}}{\Omega} \right\rceil (\Omega + D_{\max}) + (L-1)(\Omega + D_{\max}) \\
&\leq \left(\frac{C_{\max}}{\Omega} + 1 \right) (\Omega + D_{\max}) + (L-1)(\Omega + D_{\max}) \\
&= C_{\max} + LD_{\max} + \frac{D_{\max} C_{\max}}{\Omega} + L\Omega
\end{aligned}
$$

The upper bound is minimized by $\Omega = \sqrt{\frac{D_{\max} C_{\max}}{L}}$

$$
C_{\text{total}} \leq C_{\max} + 2\sqrt{C_{\max} D_{\max} L} + D_{\max} L
$$

Asymptotic optimality

$$C_{\max} \leq C^* \leq C_{\textsf{total}} \leq C_{\max} + 2\sqrt{C_{\max} D_{\max} L} + D_{\max} L$$

$$1 \leq \frac{C_{\textsf{total}}}{C_{\max}} \leq 1 + 2\sqrt{\frac{D_{\max} L}{C_{\max}}} + \frac{D_{\max} L}{C_{\max}}$$

$$\text{With } \Omega = \sqrt{\frac{D_{\max} C_{\max}}{L}}$$

Resources needed

$$\sum_{(k,i)|a_{k,i}=a,k\geq 2} m_k \leq \sum_{(k,i)|a_{k,i}=a,k\geq 2} \left(\frac{n_k}{C_{\max}} \sqrt{\frac{D_{\max} C_{\max}}{L}} + 1 \right)$$

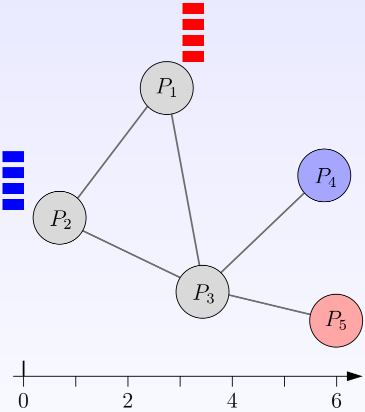
$$\leq \sqrt{\frac{D_{\max} C_{\max}}{L}} + D_{\max}$$

Conclusion

- We forget the initiation and termination phases
- Rational resolution of the steady-state
- Round whose size is the square-root of the solution:
  - Each round "loses" a constant amount of time
  - The sum of the waisted times increases less quickly than the schedule
  - Buffers of size the square-root of the solution

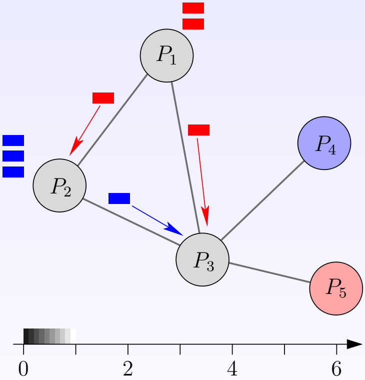
Overview

Packet routing without fixed path



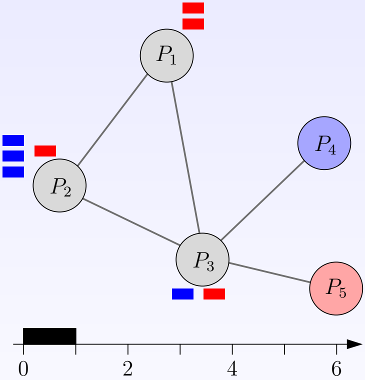
- n_c collections of packets to be routed
- packets of a same collection may follow different paths
- $n^{k,l}$: total number of packets to be routed from k to l

Packet routing without fixed path



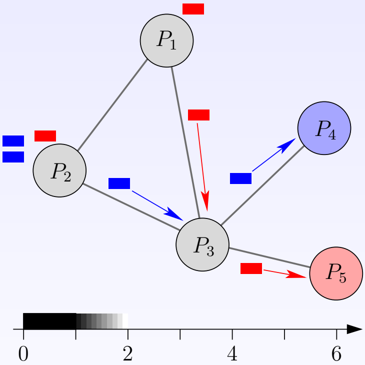
- n_c collections of packets to be routed
- packets of a same collection may follow different paths
- $n^{k,l}$: total number of packets to be routed from k to l
- rule: one edge cannot carry two packets at the same time

Packet routing without fixed path



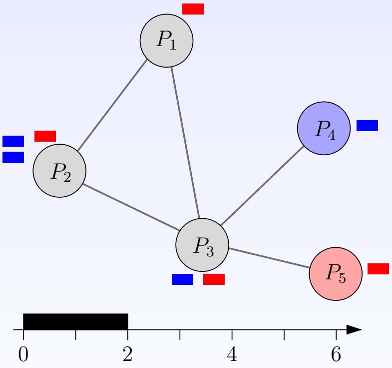
- n_c collections of packets to be routed
- packets of a same collection may follow different paths
- $n^{k,l}$: total number of packets to be routed from k to l
- rule: one edge cannot carry two packets at the same time

Packet routing without fixed path



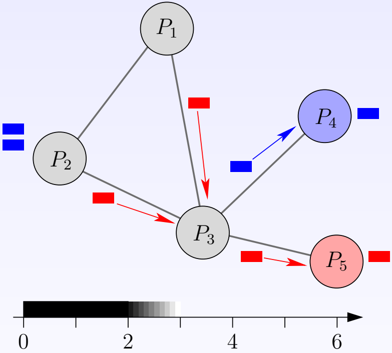
- n_c collections of packets to be routed
- packets of a same collection may follow different paths
- $n^{k,l}$: total number of packets to be routed from k to l
- rule: one edge cannot carry two packets at the same time

Packet routing without fixed path



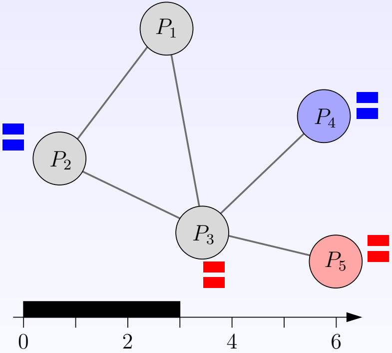
- n_c collections of packets to be routed
- packets of a same collection may follow different paths
- $n^{k,l}$: total number of packets to be routed from k to l
- rule: one edge cannot carry two packets at the same time

Packet routing without fixed path



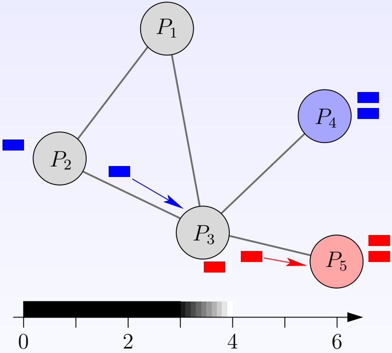
- n_c collections of packets to be routed
- packets of a same collection may follow different paths
- $n^{k,l}$: total number of packets to be routed from k to l
- rule: one edge cannot carry two packets at the same time

Packet routing without fixed path



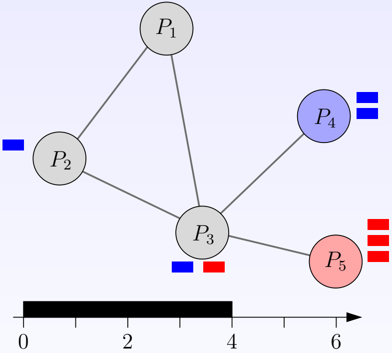
- n_c collections of packets to be routed
- packets of a same collection may follow different paths
- $n^{k,l}$: total number of packets to be routed from k to l
- rule: one edge cannot carry two packets at the same time

Packet routing without fixed path



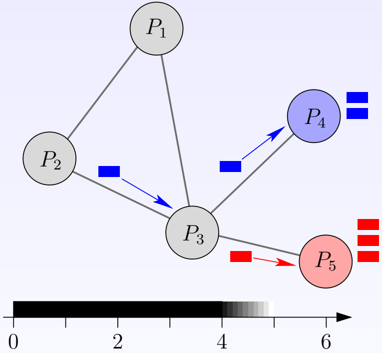
- n_c collections of packets to be routed
- packets of a same collection may follow different paths
- $n^{k,l}$: total number of packets to be routed from k to l
- rule: one edge cannot carry two packets at the same time

Packet routing without fixed path



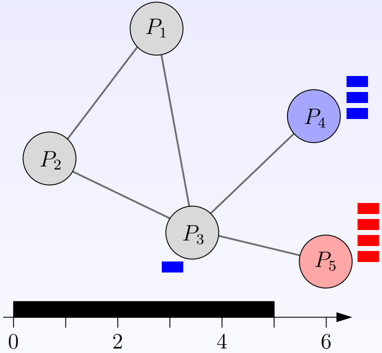
- n_c collections of packets to be routed
- packets of a same collection may follow different paths
- $n^{k,l}$: total number of packets to be routed from k to l
- rule: one edge cannot carry two packets at the same time

Packet routing without fixed path



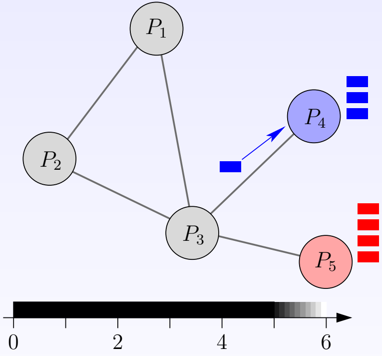
- n_c collections of packets to be routed
- packets of a same collection may follow different paths
- $n^{k,l}$: total number of packets to be routed from k to l
- rule: one edge cannot carry two packets at the same time

Packet routing without fixed path



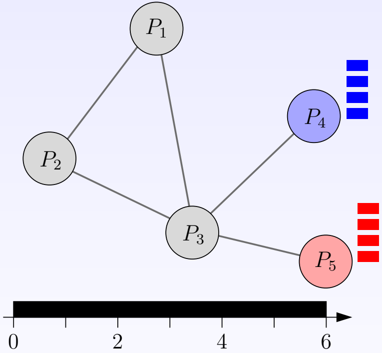
- n_c collections of packets to be routed
- packets of a same collection may follow different paths
- $n^{k,l}$: total number of packets to be routed from k to l
- rule: one edge cannot carry two packets at the same time

Packet routing without fixed path



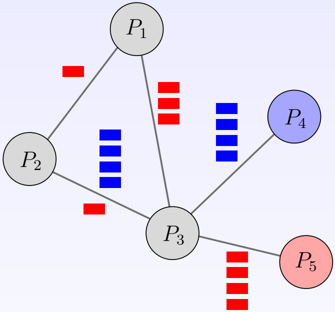
- n_c collections of packets to be routed
- packets of a same collection may follow different paths
- $n^{k,l}$: total number of packets to be routed from k to l
- rule: one edge cannot carry two packets at the same time

Packet routing without fixed path



- n_c collections of packets to be routed
- packets of a same collection may follow different paths
- $n^{k,l}$: total number of packets to be routed from k to l
- rule: one edge cannot carry two packets at the same time

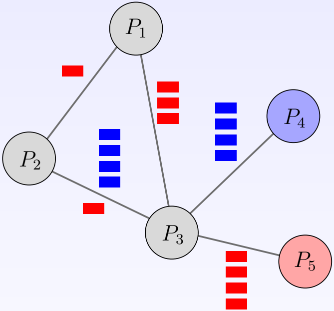
Packet routing without fixed path



- n_c collections of packets to be routed
- packets of a same collection may follow different paths
- $n^{k,l}$: total number of packets to be routed from k to l
- rule: one edge cannot carry two packets at the same time

- $n_{i,j}^{k,l}$: total number of packets routed from k to l and crossing edge (i,j)

Packet routing without fixed path



- n_c collections of packets to be routed
- packets of a same collection may follow different paths
- $n^{k,l}$: total number of packets to be routed from k to l
- rule: one edge cannot carry two packets at the same time

- $n_{i,j}^{k,l}$: total number of packets routed from k to l and crossing edge (i,j)
- Congestion: $C_{i,j} = \sum_{(k,l)|n^{k,l}>0} n_{i,j}^{k,l}; \quad C_{\max} = \max_{i,j} C_{i,j}$

Equations (1/2)

1. Initialization

$$\sum_{j|(k,j)\in A} n_{k,j}^{k,l} = n^{k,l}$$

2. Reception

$$\sum_{i|(i,l)\in A} n_{i,l}^{k,l} = n^{k,l}$$

3. Conservation law

$$\sum_{i|(i,j)\in A} n_{i,j}^{k,l} = \sum_{i|(j,i)\in A} n_{j,i}^{k,l} \quad \forall (k,l), j \neq k, j \neq l$$

Equations (2/2)

4. Congestion

$$C_{i,j} = \sum_{(k,l)|n^{k,l}>0} n_{i,j}^{k,l}$$

5. Objective function

$$C_{\max} \geq C_{i,j}, \qquad \forall i, j$$

Minimize $C_{\max}$

Linear program in rational numbers: polynomial-time solution.

Solution:
number of messages $n_{i,j}^{k,l}$ on each edge to minimize congestion

Routing algorithm

1. Computing optimal solution $C_{\max}$ of previous linear program
2. Consider periods of length Ω (to be defined later)
3. During each time-interval $[p\Omega, (p+1)\Omega]$, follow the optimal solution: edge (i,j) forwards:

   $$m_{i,j}^{k,l} = \left\lfloor \frac{n_{i,j}^{k,l}\Omega}{C_{\max}} \right\rfloor$$

   packets that go from k to l.
   (if available)
4. number of such periods: $\left\lceil \frac{C_{\max}}{\Omega} \right\rceil$
5. After time-step

   $$T \equiv \left\lceil \frac{C_{\max}}{\Omega} \right\rceil \Omega \leq C_{\max} + \Omega$$

   sequentially process M residual packets; this takes no longer than ML time-steps, where L is the maximum length of a simple path in the network

Feasibility

$$\sum_{(k,l)} m_{i,j}^{k,l} \leq \sum_{(k,l)} \frac{n_{i,j}^{k,l}\Omega}{C_{\max}} = \frac{C_{i,j}\Omega}{C_{\max}} \leq \Omega$$

Makespan

- Define Ω as $\Omega = \sqrt{C_{\max} n_c}$.

Makespan

- Define Ω as $\Omega = \sqrt{C_{\max} n_c}$.
- Total number of packets still inside network at time-step T is at most
$$2|A|\sqrt{C_{\max} n_c} + |A|n_c$$

Makespan

- Define Ω as $\Omega = \sqrt{C_{\max} n_c}$.
- Total number of packets still inside network at time-step T is at most
$$2|A|\sqrt{C_{\max} n_c} + |A|n_c$$
- Makespan:
$$C_{\max} \leq C^* \leq C_{\max} + \sqrt{C_{\max} n_c} + 2|A|\sqrt{C_{\max} n_c}|V| + |A|n_c|V|$$
$$C^* = C_{\max} + O(\sqrt{C_{\max}})$$

Steady-state scheduling

Background Approach pioneered by Bertsimas and Gamarnik

Rationale Maximize throughput (total load executed per period)

Simplicity Relaxation of makespan minimization problem

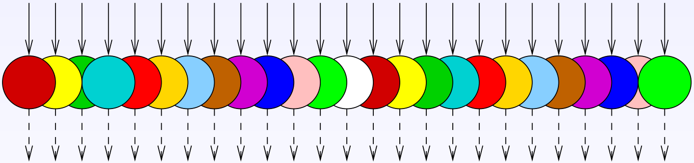
- Ignore initialization and clean-up phases
- Precise ordering/allocation of tasks/messages not needed
- Characterize resource activity during each time-unit:
  - which (rational) fraction of time is spent computing for which application?
  - which (rational) fraction of time is spent receiving or sending to which neighbor?

Efficiency Periodic schedule, described in compact form

Overview

Application graph

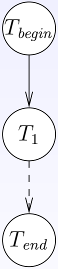
n problem instances $\mathcal{P}^{(1)}, \mathcal{P}^{(2)}, \ldots, \mathcal{P}^{(n)}$, where n is large



Application graph

n problem instances $\mathcal{P}^{(1)}, \mathcal{P}^{(2)}, \ldots, \mathcal{P}^{(n)}$, where n is large

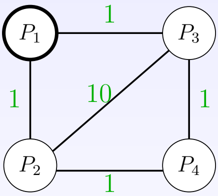
Each problem corresponds to a copy of the same task graph $G_A = (V_A, E_A)$, the application graph



T_{begin} et T_{end} are fictitious tasks, used to model the scattering of input files and the gathering of output files

Platform graph

Target platform represented by platform graph $G_P = (V_P, E_P)$

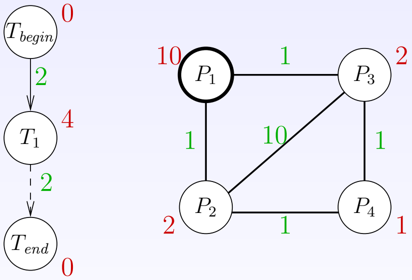


Edge $P_i \rightarrow P_j$ is labeled with $c_{i,j}$: time needed to send a unit-length message from P_i to P_j

Communication model: full overlap, one-port for incoming and outgoing messages

Computations and communications

P_i requires $w_{i,k}$ time-units to process task T_k ($k \in \{begin, 1, end\}$).



Edge $e_{k,l} : T_k \to T_l$ in G_A is labeled with $data_{k,l}$: data volume generated by T_k and used by T_l

Transfer time of a file $e_{k,l}$ from P_i to P_j: $data_{k,l} \times c_{i,j}$

Definitions

Allocation An allocation is a pair of mappings: $\pi : V_A \mapsto V_P$ and $\sigma : E_A \mapsto \{\text{paths in } G_P\}$

Schedule A schedule associated to an allocation (π, σ) is a pair of mappings: $t_\pi : V_A \mapsto \mathbb{R}$ and application $t_\sigma : E_A \times E_P \mapsto \mathbb{R}$, satisfying to:

- precedence constraints
- resource constraints on processors
- resource constraints on network links
- one-port constraints

Activity variables

$cons(P_i, T_k)$: average number of tasks of type T_k processed by P_i every time-unit

$$\forall P_i, \forall T_k \in V_A,\ 0 \leq cons(P_i, T_k) \times w_{i,k} \leq 1$$

$sent(P_i \rightarrow P_j, e_{k,l})$: average number of files of type $e_{k,l}$ sent from P_i to P_j every time-unit

$$\forall P_i, P_j,\ 0 \leq sent(P_i \rightarrow P_j, e_{k,l}) \times (data_{k,l} \times c_{i,j}) \leq 1$$

Steady-state equations

1. One-port for outgoing communications. P_i sends messages to its neighbors sequentially

$$\forall P_i, \sum_{P_i \to P_j} \sum_{e_{k,l} \in E_A} \left(sent(P_i \to P_j, e_{k,l}) \times data_{k,l} \times c_{i,j} \right) \leq 1$$

2. One-port for ingoing communications. P_i receives messages sequentially

$$\forall P_i, \sum_{P_j \to P_i} \sum_{e_{k,l} \in E_A} \left(sent(P_j \to P_i, e_{k,l}) \times data_{k,l} \times c_{j,i} \right) \leq 1$$

3. Overlap. Computations and communications take place simultaneously

$$\forall P_i, \sum_{T_k \in V_A} cons(P_i, T_k) \times w_{i,k} \leq 1$$

Conservation law

Consider a processor P_i and an edge $e_{k,l}$ of the application graph:

Files of type $e_{k,l}$ received: $\displaystyle\sum_{P_j \to P_i} sent(P_j \to P_i, e_{k,l})$

Files of type $e_{k,l}$ generated: $cons(P_i, T_k)$

Files of type $e_{k,l}$ consumed: $cons(P_i, T_l)$

Files of type $e_{k,l}$ sent: $\displaystyle\sum_{P_i \to P_j} sent(P_i \to P_j, e_{k,l})$

In steady state:

$$\forall P_i, \forall e_{k,l} : T_k \to T_l \in E_A,$$

$$\sum_{P_j \to P_i} sent(P_j \to P_i, e_{k,l}) + cons(P_i, T_k) =$$

$$\sum_{P_i \to P_j} sent(P_i \to P_j, e_{k,l}) + cons(P_i, T_l)$$

Upper bound for the throughput

MAXIMIZE $\rho = \sum_{i=1}^{p} cons(P_i, T_{end})$,

UNDER THE CONSTRAINTS

$$
\begin{cases}
\text{(1a)} & \forall P_i, \forall T_k \in V_A,\ 0 \leq cons(P_i, T_k) \times w_{i,k} \leq 1 \\
\text{(1b)} & \forall P_i, P_j,\ 0 \leq sent(P_i \to P_j, e_{k,l}) \times (data_{k,l} \times c_{i,j}) \leq 1 \\
\text{(1c)} & \forall P_i, \displaystyle\sum_{P_i \to P_j} \sum_{e_{k,l} \in E_A} \big(sent(P_i \to P_j, e_{k,l}) \times data_{k,l} \times c_{i,j}\big) \leq 1 \\
\text{(1d)} & \forall P_i, \displaystyle\sum_{P_j \to P_i} \sum_{e_{k,l} \in E_A} \big(sent(P_j \to P_i, e_{k,l}) \times data_{k,l} \times c_{j,i}\big) \leq 1 \\
\text{(1e)} & \forall P_i, \displaystyle\sum_{T_k \in V_A} cons(P_i, T_k) \times w_{i,k} \leq 1 \\
\text{(1f)} & \forall P_i, \forall e_{k,l} \in E_A : T_k \to T_l, \\
& \qquad \displaystyle\sum_{P_j \to P_i} sent(P_j \to P_i, e_{k,l}) + cons(P_i, T_k) = \\
& \qquad\qquad \displaystyle\sum_{P_i \to P_j} sent(P_i \to P_j, e_{k,l}) + cons(P_i, T_l)
\end{cases}
$$

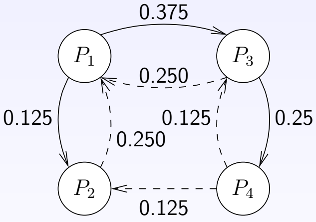
How to design a schedule achieving this throughput?

Back to the example

Computations

	$cons(P_i, T_1)$
P_1	0.025
P_2	0.125
P_3	0.125
P_4	0.250
Total	21 tasks / 40 seconds

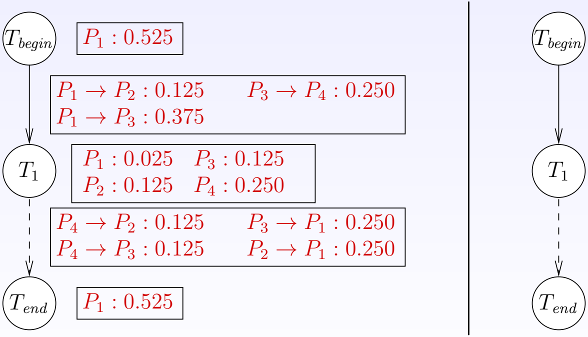
Communications



$$sent(P_i \rightarrow P_j, e_{k,l})$$

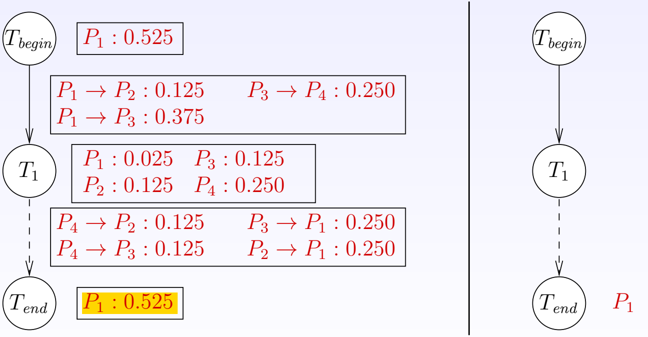
Decomposition into a set of allocations (1/2)

Steady state = superposition of several allocations



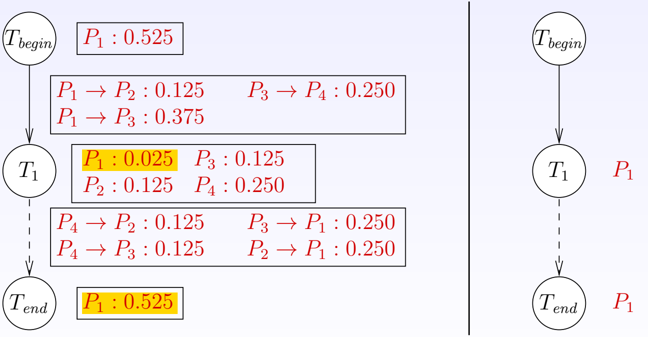
Decomposition into a set of allocations (1/2)

Steady state = superposition of several allocations



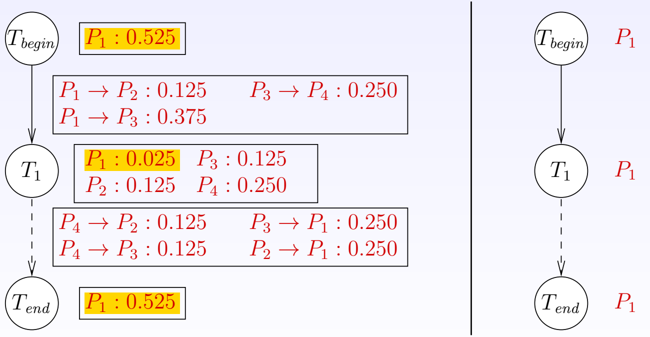
Decomposition into a set of allocations (1/2)

Steady state = superposition of several allocations



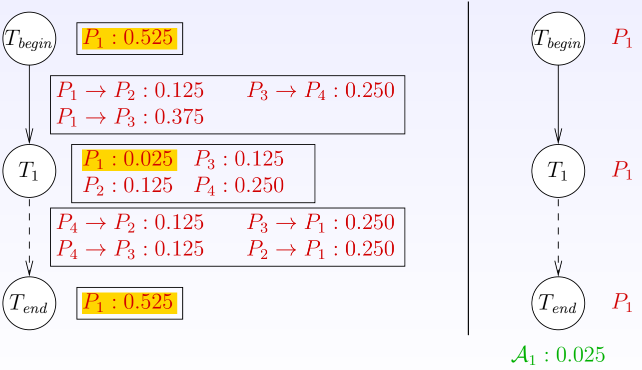
Decomposition into a set of allocations (1/2)

Steady state = superposition of several allocations



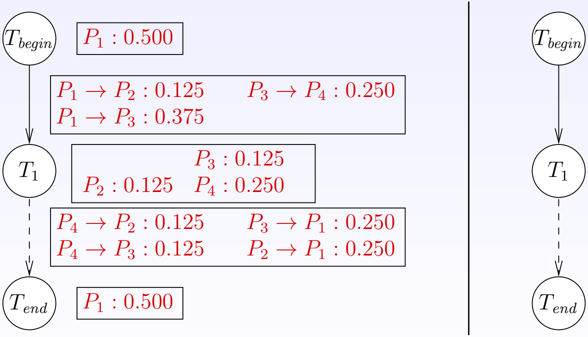
Decomposition into a set of allocations (1/2)

Steady state = superposition of several allocations



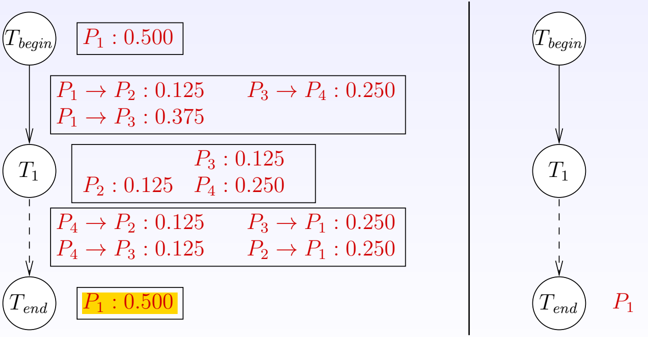
Decomposition into a set of allocations (1/2)

Steady state = superposition of several allocations



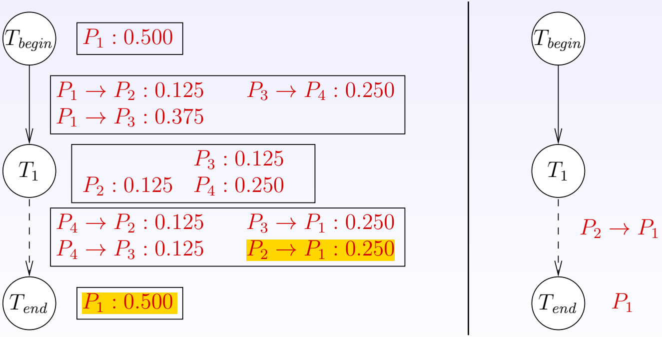
Decomposition into a set of allocations (1/2)

Steady state = superposition of several allocations



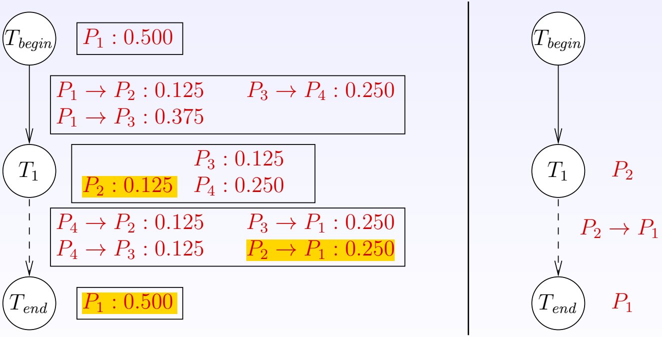
Decomposition into a set of allocations (1/2)

Steady state = superposition of several allocations



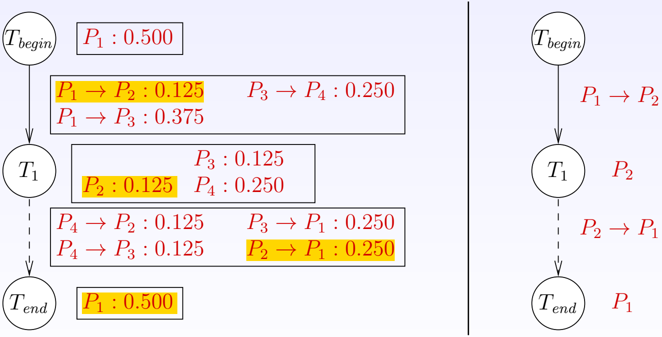
Decomposition into a set of allocations (1/2)

Steady state = superposition of several allocations



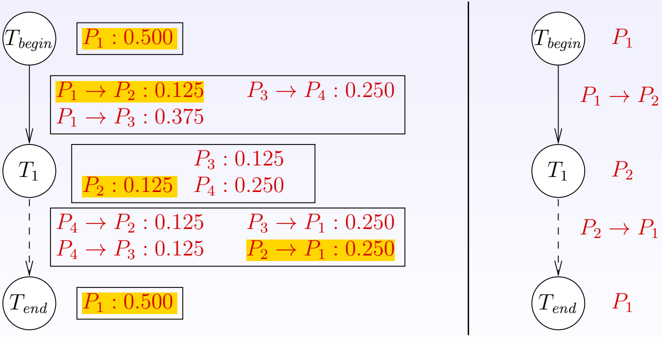
Decomposition into a set of allocations (1/2)

Steady state = superposition of several allocations



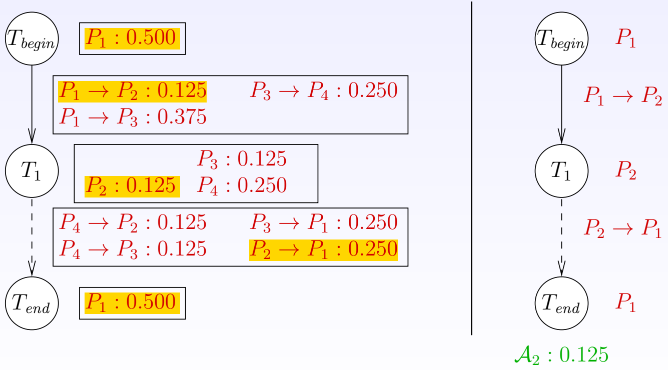
Decomposition into a set of allocations (1/2)

Steady state = superposition of several allocations



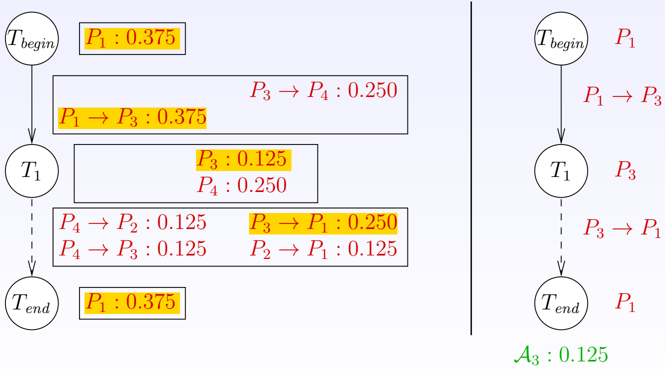
Decomposition into a set of allocations (1/2)

Steady state = superposition of several allocations



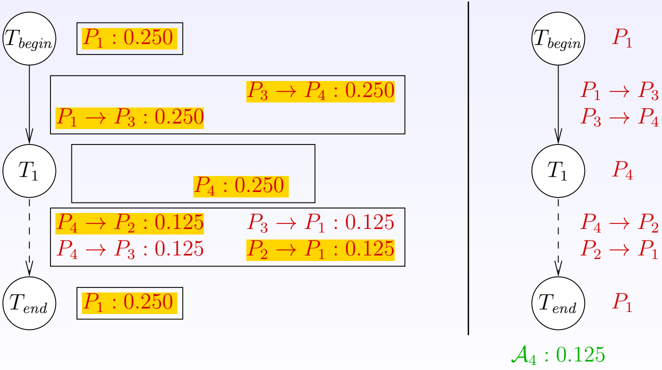
Decomposition into a set of allocations (1/2)

Steady state = superposition of several allocations



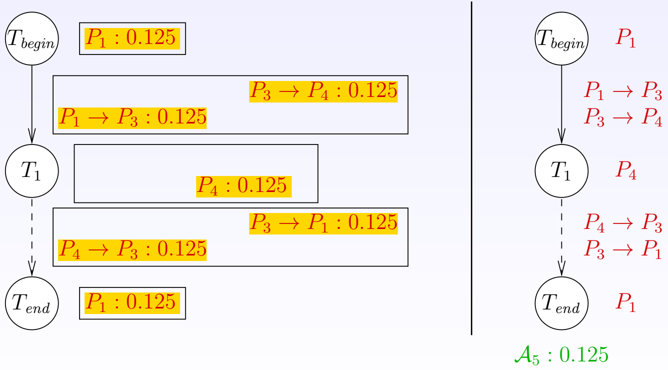
Decomposition into a set of allocations (1/2)

Steady state = superposition of several allocations

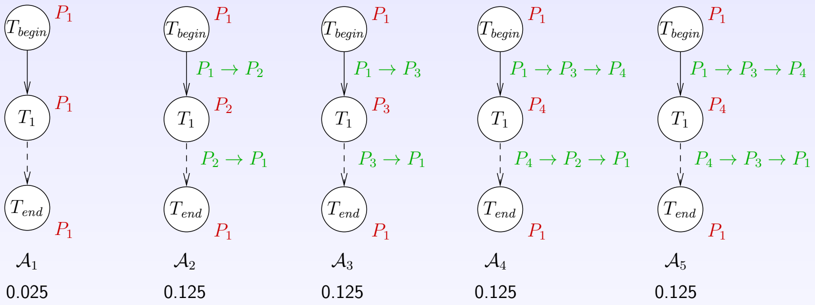


Decomposition into a set of allocations (1/2)

Steady state = superposition of several allocations

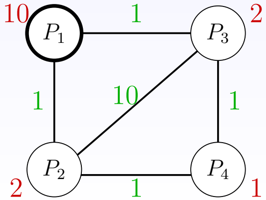


Decomposition into a set of allocations (2/2)

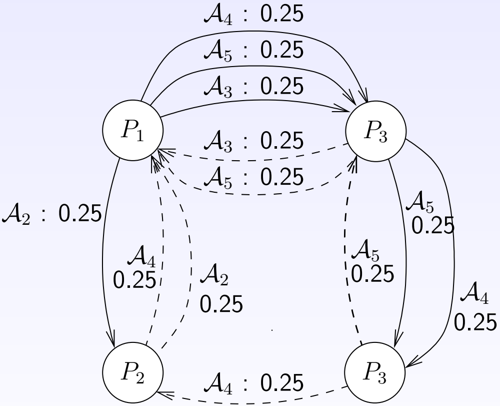


This decomposition is always possible

How to orchestrate these allocations?

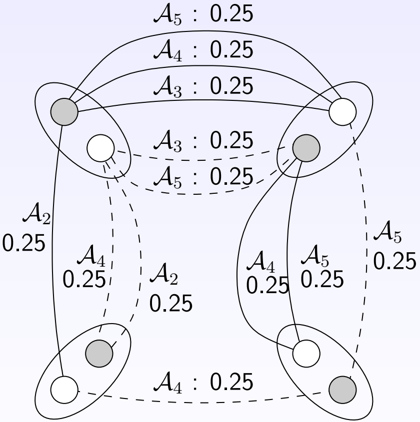


Communication graph

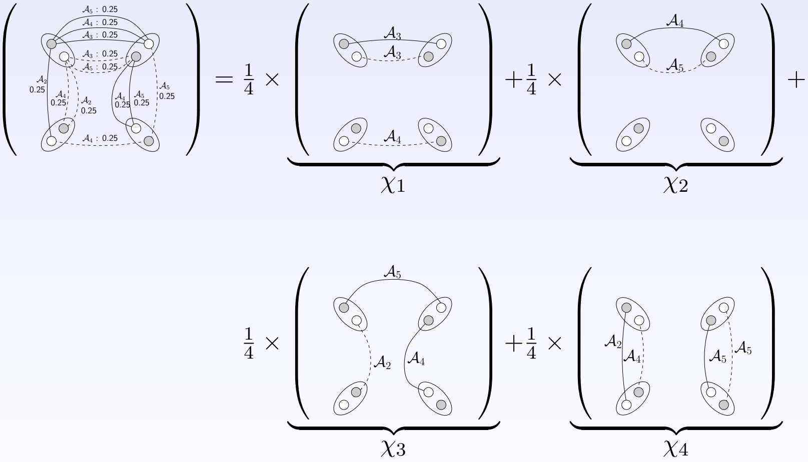


Fraction of time spent transferring some $e_{k,l}$ file from P_i to P_j for a given allocation

One-port constraints = matching

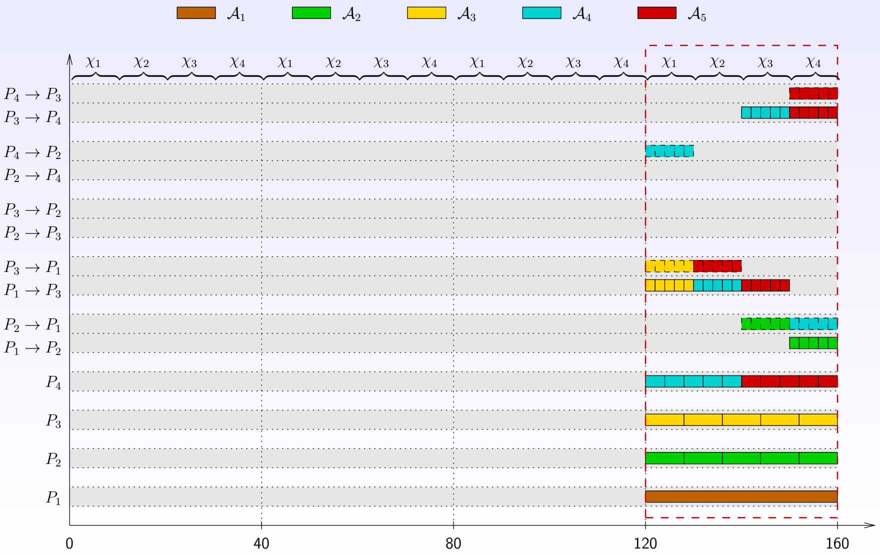


Edge coloring (decomposition into matchings)

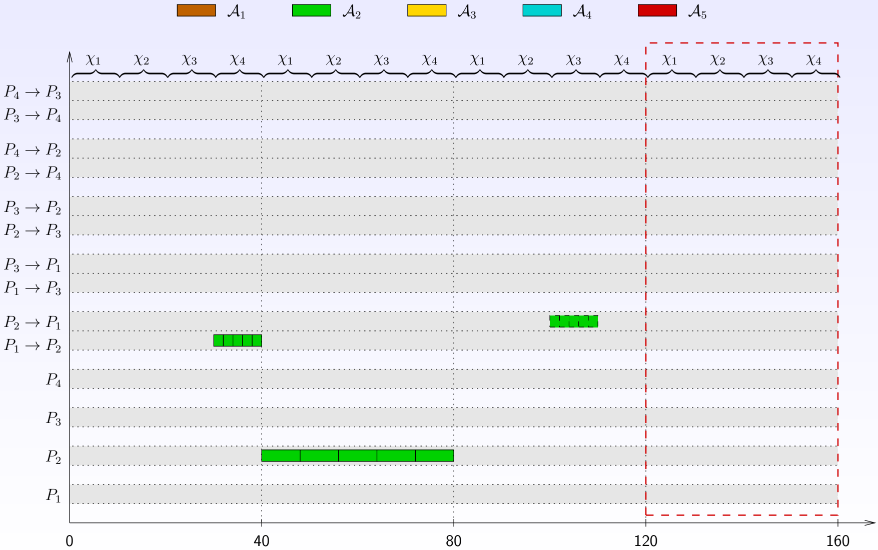


This decomposition is always possible

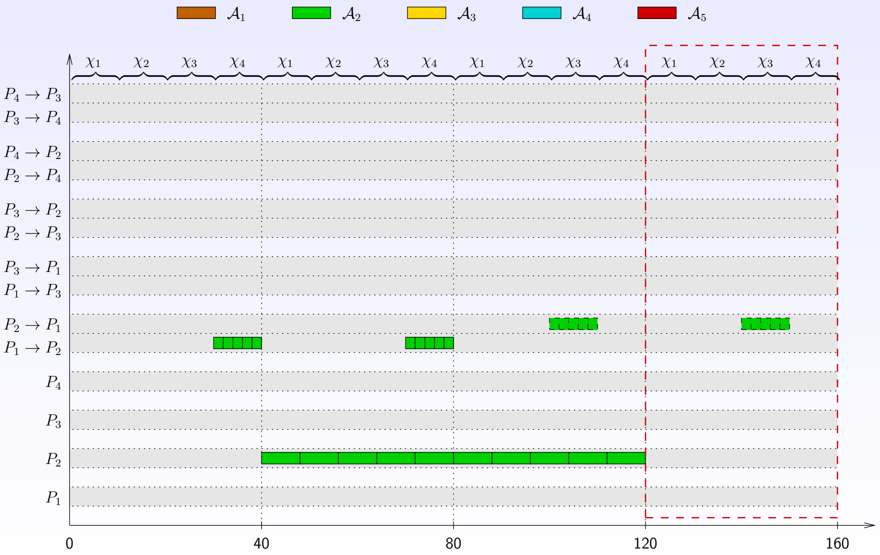
Cyclic scheduling achieving optimal throughput



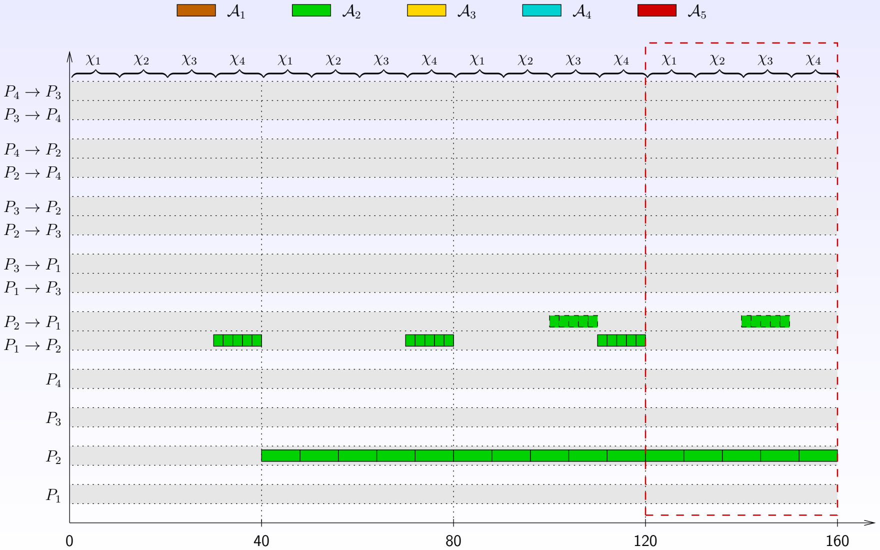
Cyclic scheduling achieving optimal throughput



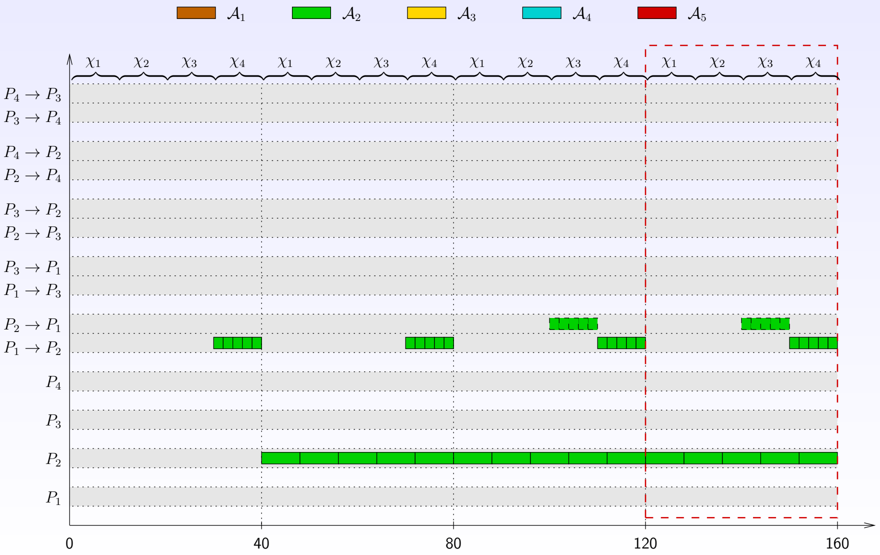
Cyclic scheduling achieving optimal throughput



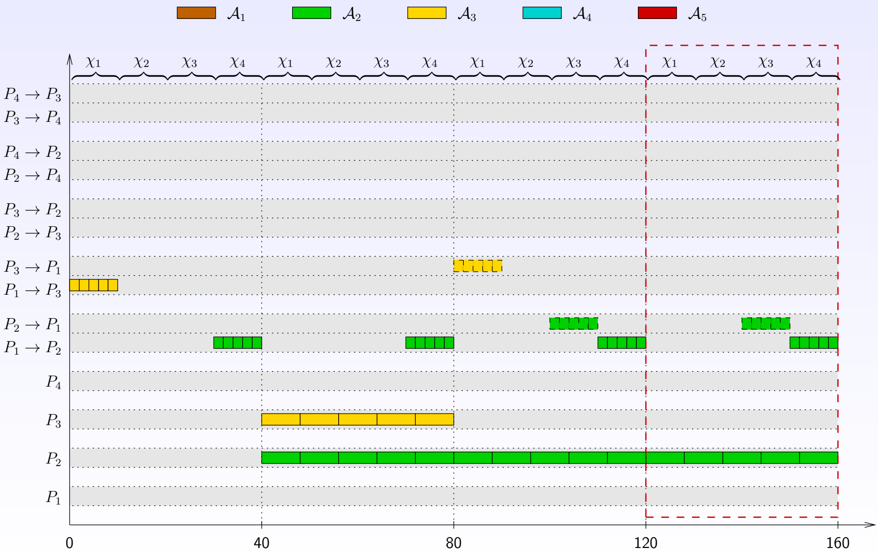
Cyclic scheduling achieving optimal throughput



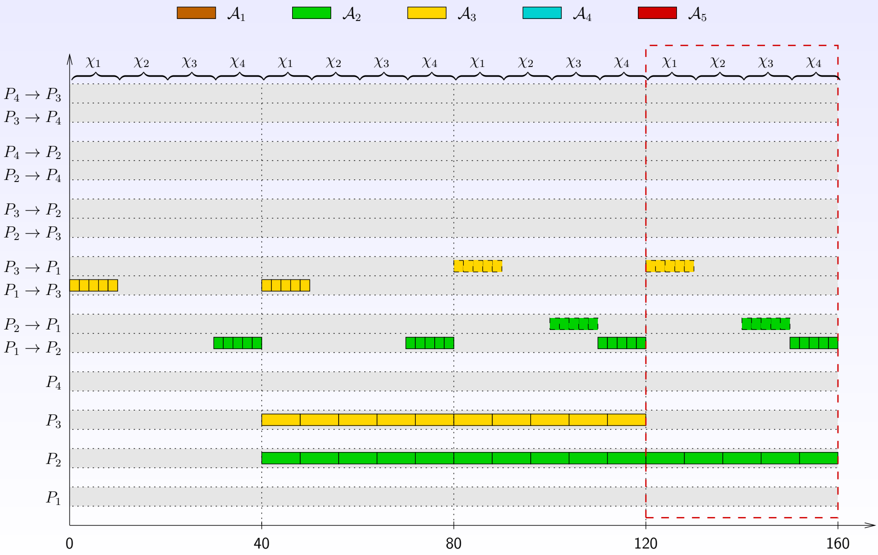
Cyclic scheduling achieving optimal throughput



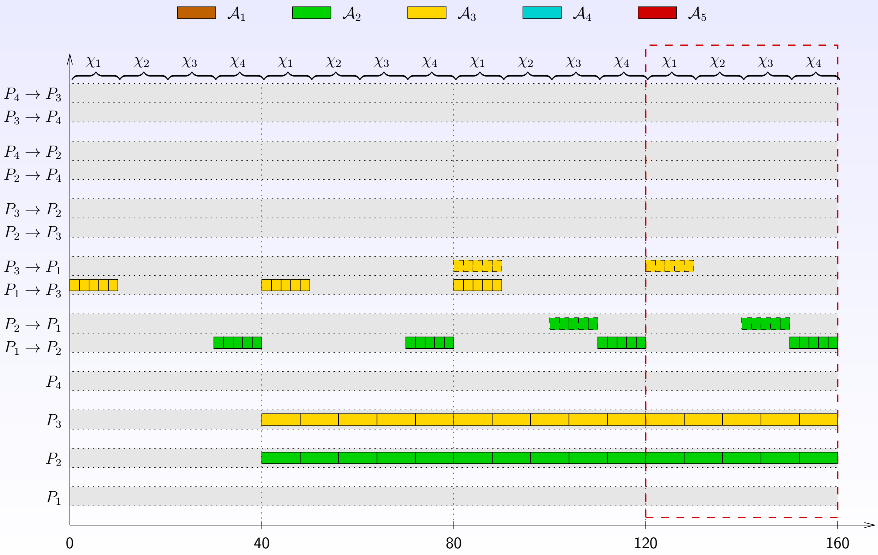
Cyclic scheduling achieving optimal throughput



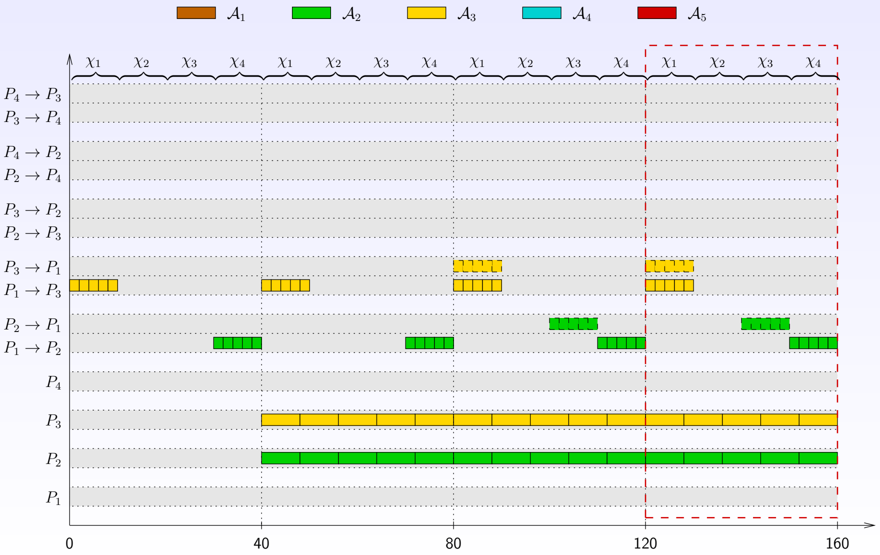
Cyclic scheduling achieving optimal throughput



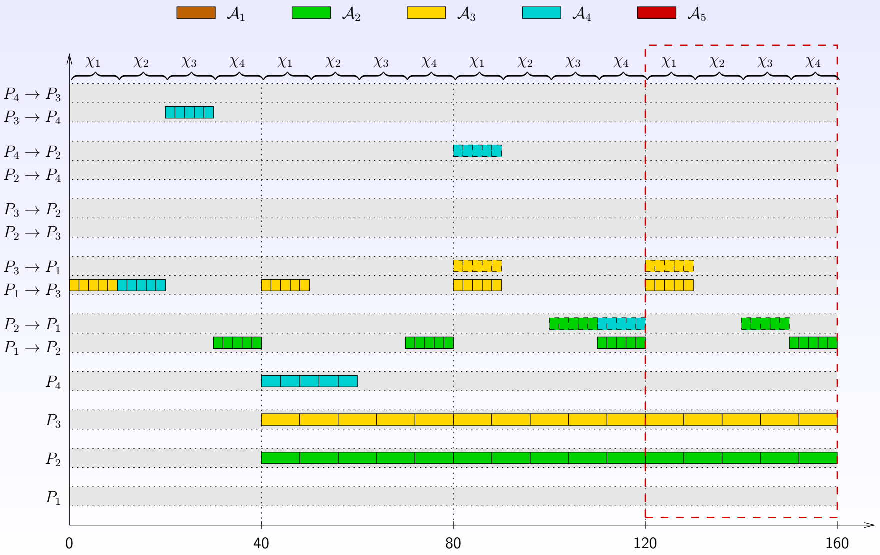
Cyclic scheduling achieving optimal throughput



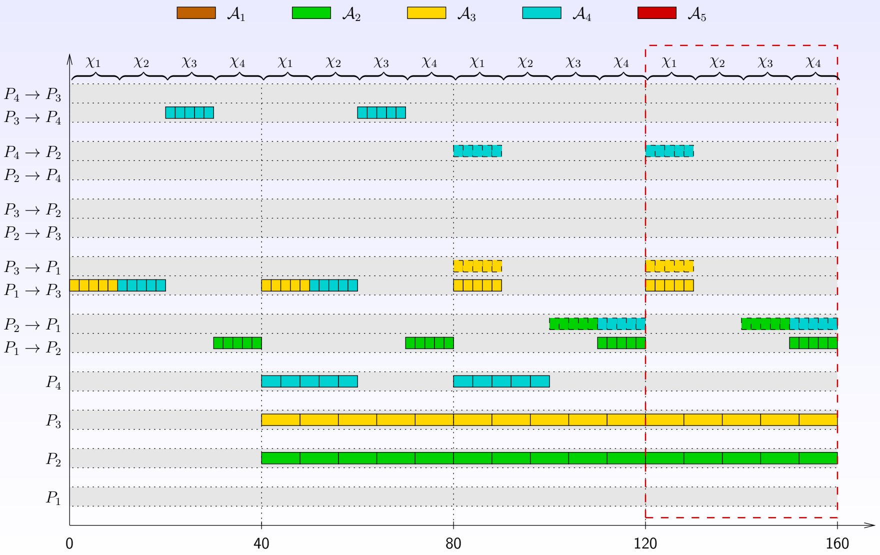
Cyclic scheduling achieving optimal throughput



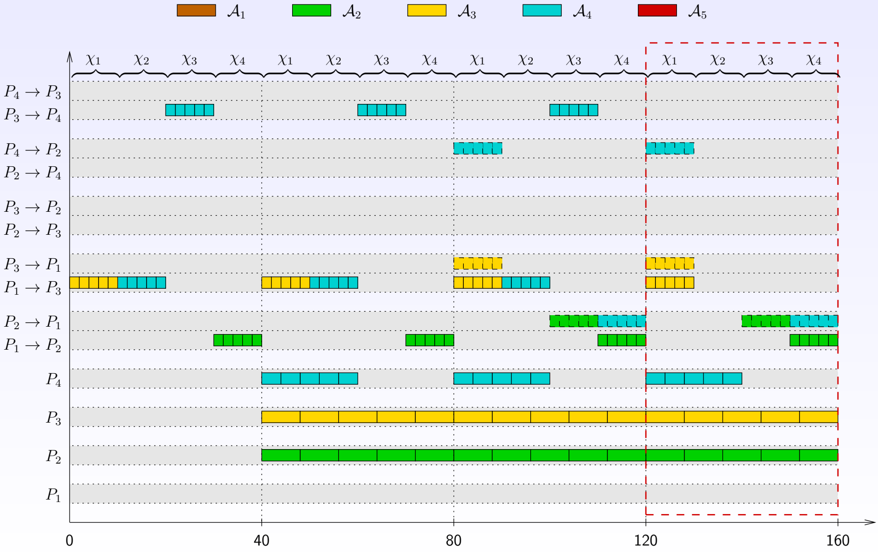
Cyclic scheduling achieving optimal throughput



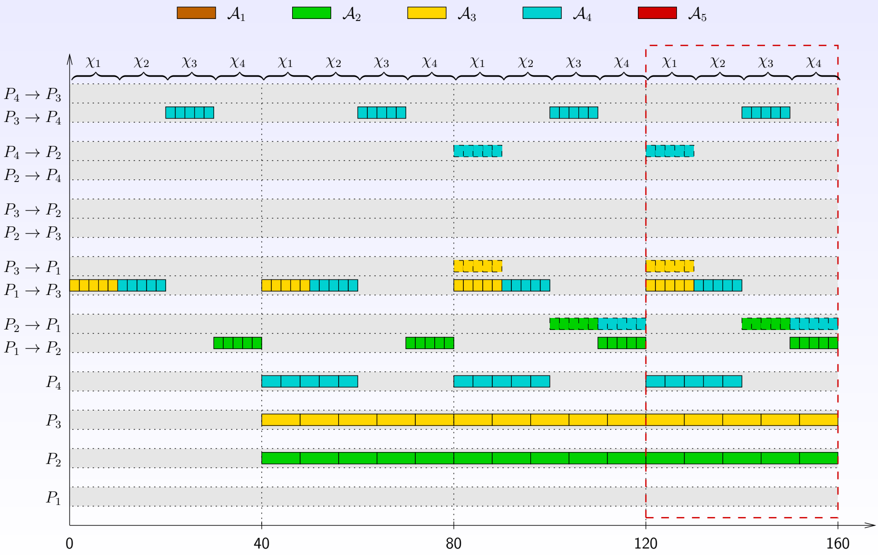
Cyclic scheduling achieving optimal throughput



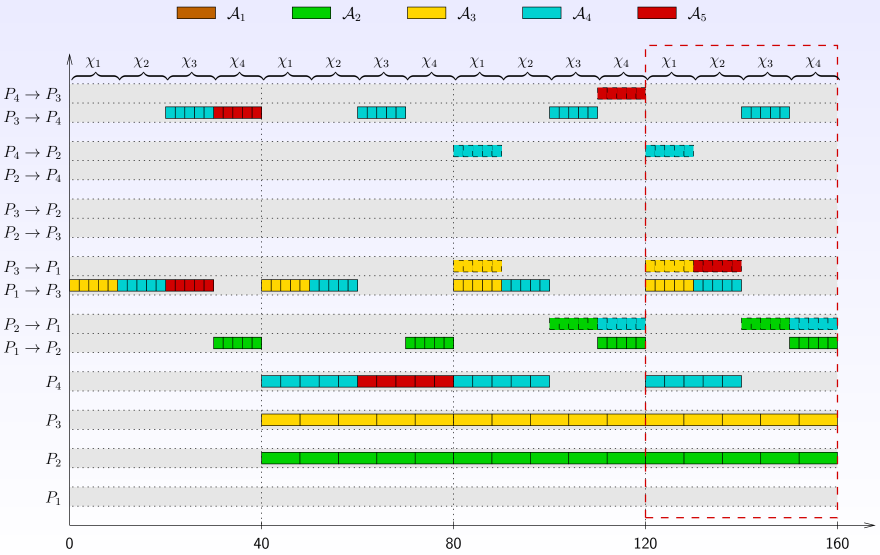
Cyclic scheduling achieving optimal throughput



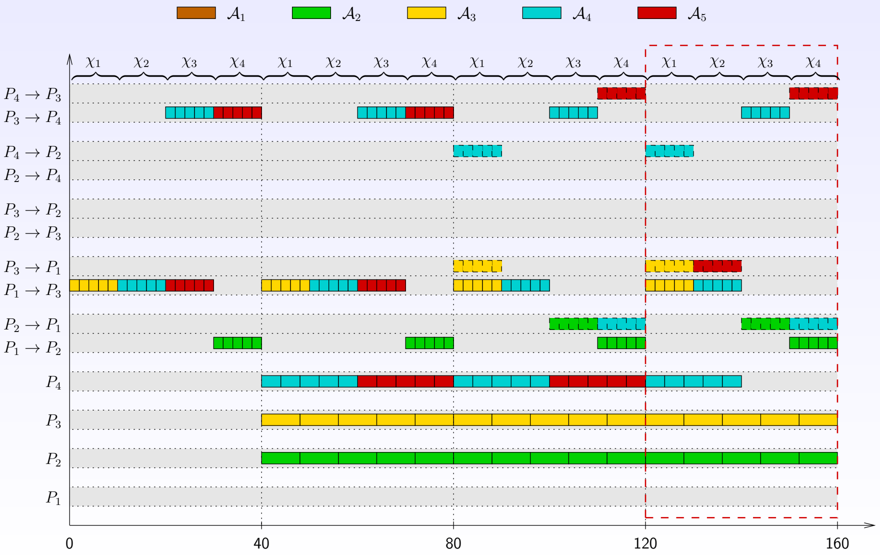
Cyclic scheduling achieving optimal throughput



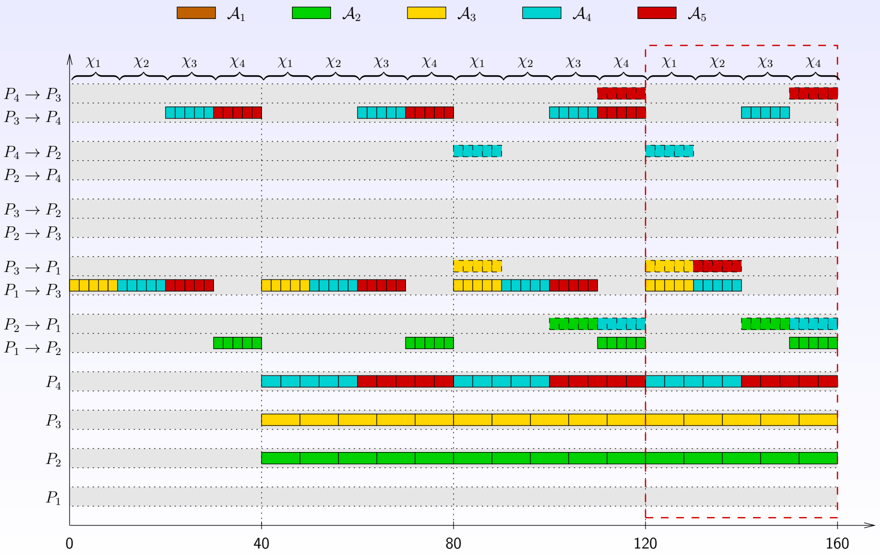
Cyclic scheduling achieving optimal throughput



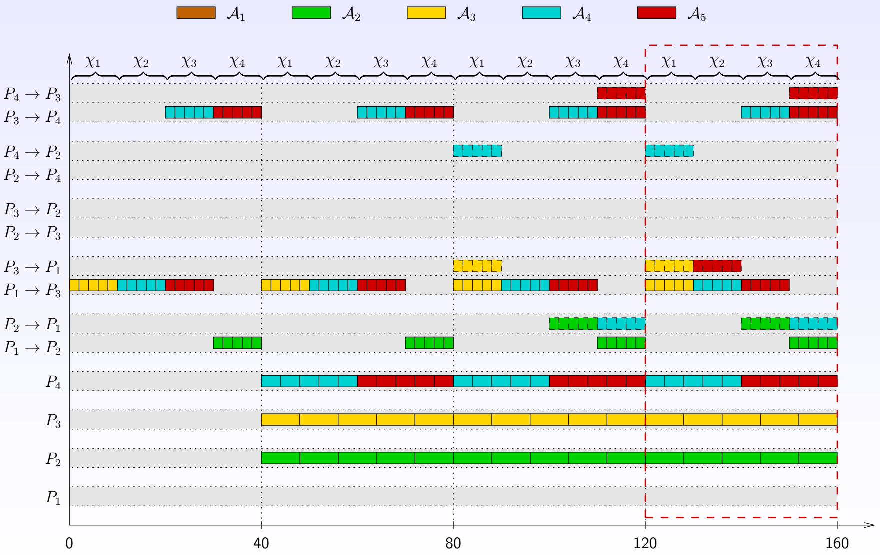
Cyclic scheduling achieving optimal throughput



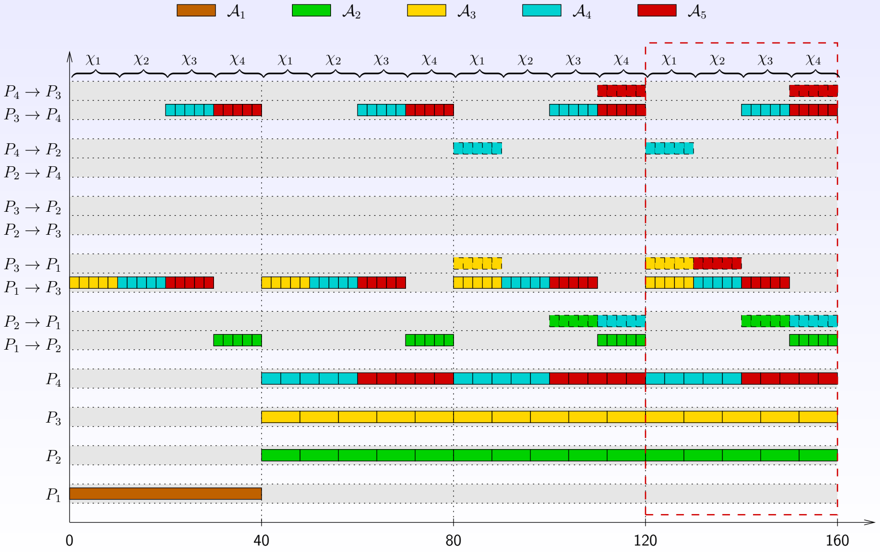
Cyclic scheduling achieving optimal throughput



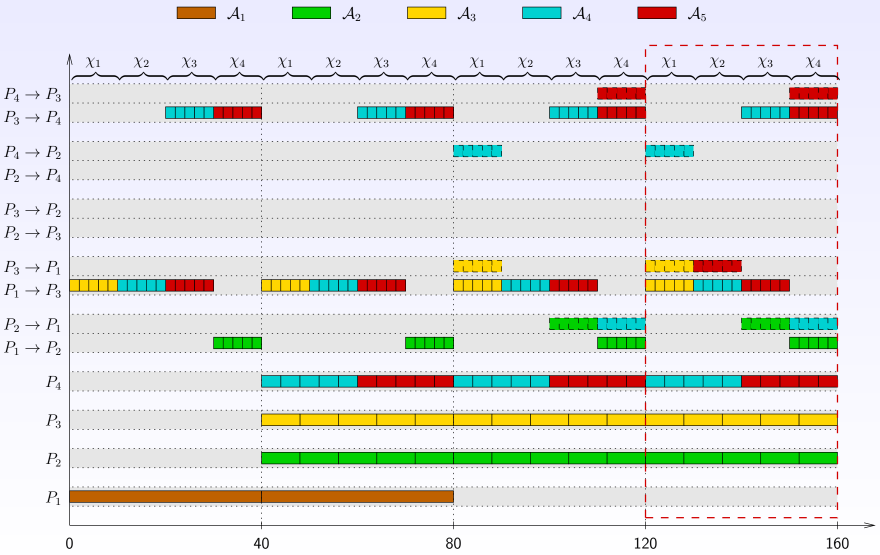
Cyclic scheduling achieving optimal throughput



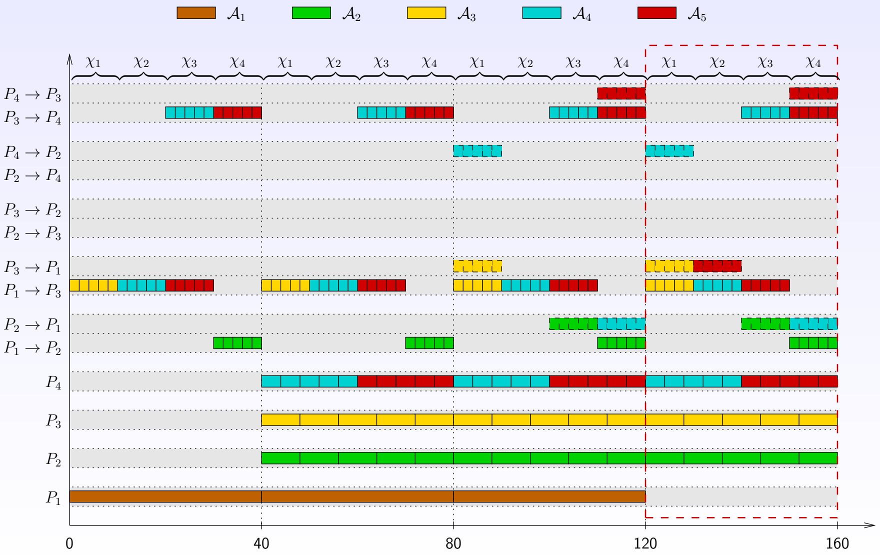
Cyclic scheduling achieving optimal throughput



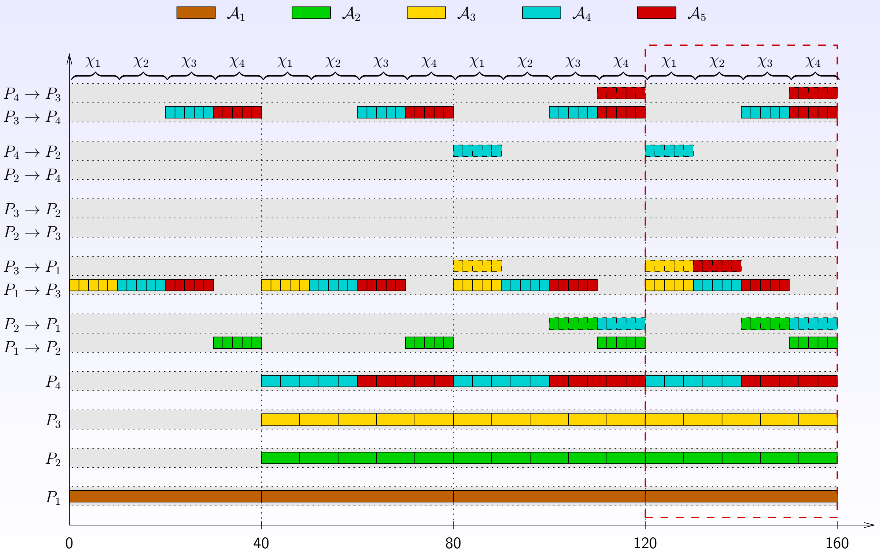
Cyclic scheduling achieving optimal throughput



Cyclic scheduling achieving optimal throughput



Cyclic scheduling achieving optimal throughput



Asymptotically optimal schedule

- The technique used in the example is
  - general
  - polynomial
- The resulting schedule is asymptotically optimal: within T time-steps, it differs from the optimal schedule by a constant number of tasks (independent of T)

Extensions to collections of general task graphs

- More difficult but possible
- Maximizing throughput NP-hard ☹
- Most application DAGs have polynomial number of joins
  $\Rightarrow$ polynomial solution ☺