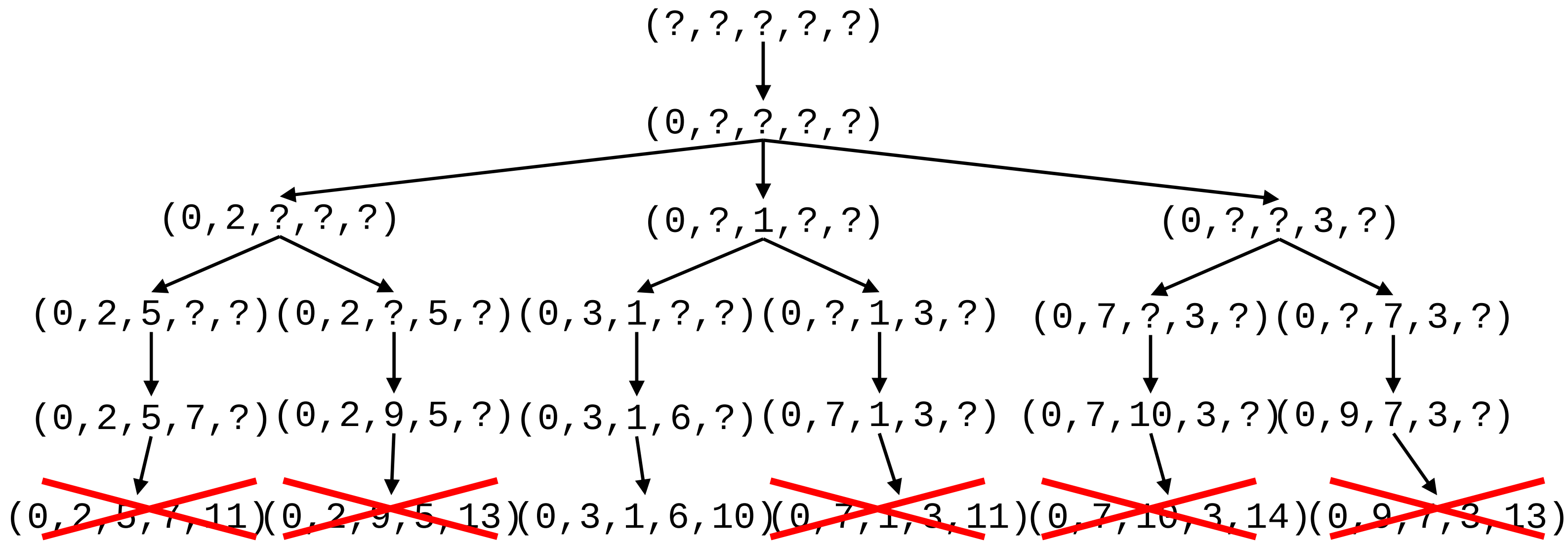
Adaptive Scheduling of Streams in Real-time Applications

Marek Balej, Zdenek Hanzalek
Czech Technical University in Prague

Contents

- Time-triggered Approach and Problem description
- Scheduling with time constraints
- Adaptive scheduling, i.e. adding new messages/nodes into the existing schedule

Search space of real-time problems



Complexity of the **optimal synthesis problem** is comparable the one of the **verification/analysis problem.**

In practice: sub-optimal solution, found in a part of the search tree (e.g. by heuristic algorithm) **has practical value**, but **partial verification** (i.e the one which does not consider all possible behaviors) has none.

It is **easier to synthesize the time-constrained system** than to leave the freedom to the designer and consequently verify its time properties.

Problem Description

- Time-triggered scheduling
- Non-preemptive scheduling
- Routing is given (e.g. tree topology)
- Centralized algorithm
- Respecting time constraints

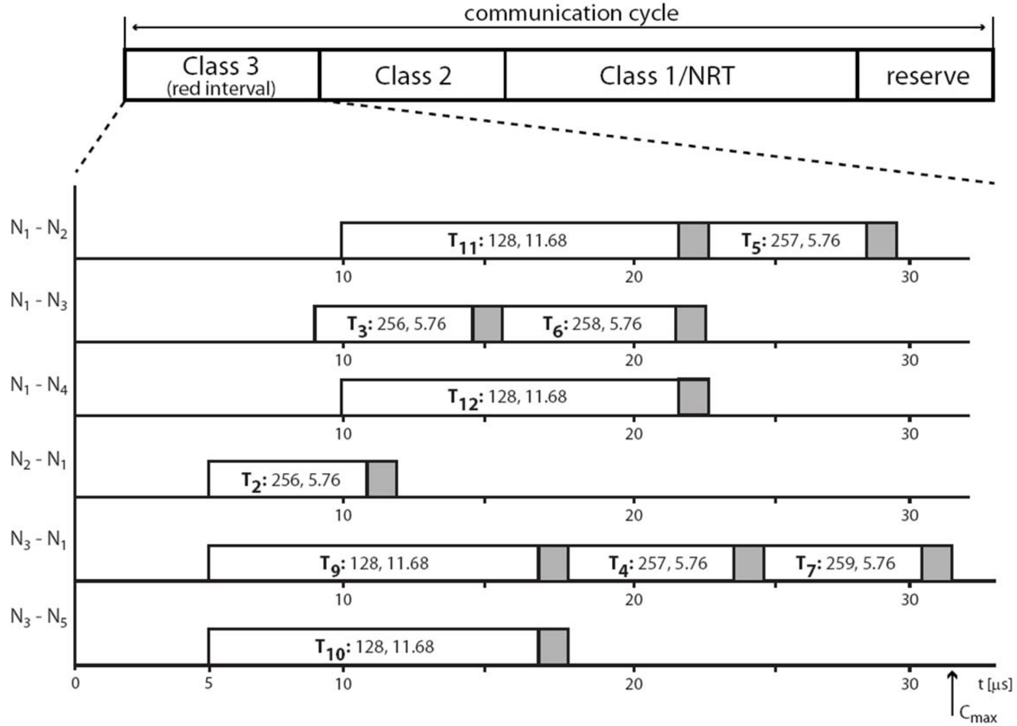
(Non-adaptive) Scheduling

Basic assumptions

- Tree topology
- switch integrated in each node (special HW)
- full duplex

Time-triggered interval

- highest-priority
- strictly isochronous - Precision Transparent Clock Protocol
- data are forwarded according to a static communication schedule



Input Parameters of the Scheduling Problem

List of links

- link delay

link	$N_1 \rightarrow N_3$	$N_1 \rightarrow N_4$	$N_1 \rightarrow N_2$	$N_2 \rightarrow N_1$	$N_3 \rightarrow N_1$	$N_4 \rightarrow N_1$	$N_3 \rightarrow N_5$	$N_5 \rightarrow N_3$
T_{LD} [ns]	4875	5130	5862	3841	4875	4895	4875	4875

$$T_{LD} = T_{TxD} + T_{CD} + T_{RxD} + T_{ad} + T_{BD}$$

List of messages

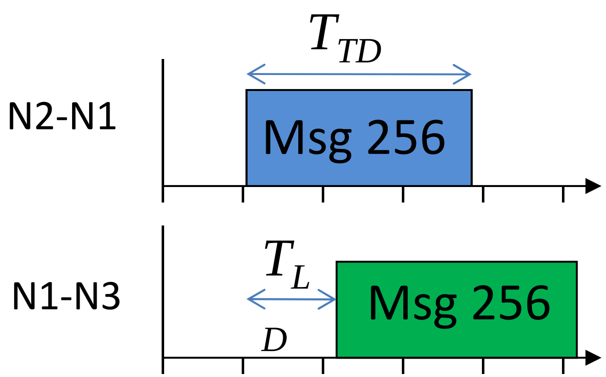
- source
- destination(s)
- transmission delay
- required
  - release date
  - deadline
  - end-to-end delay
- multicast message – used e.g. for synchronization

ID	path	T_{TD} [ns]	$\widetilde{r}$ [ns]	$\widetilde{d}$ [ns]	$\widetilde{e}$ [ns]
256	$N_2 \rightarrow N_3$	5760	5000	20000	11000
257	$N_3 \rightarrow N_2$	5760	15000	40000	15000
258	$N_1 \rightarrow N_3$	5760	15000	–	–
259	$N_3 \rightarrow N_1$	5760	20000	35000	–
128	$N_3 \rightarrow \{N_1,N_2,N_4,N_5\}$	11680	5000	{–,–,–,18000}	{–,17675,17675,15000}

Problem Refinement

The objective is to find the shortest schedule for the TT interval based on a network topology/parameters, message parameters and required position in the schedule

- messages on the same link are separated with a minimum inter-message gap (added to the transmission delay T_{TD})
- as soon as the first bit of a message is received, it may be forwarded to another link, i.e. if $T_{LD} < T_{TD}$, two nodes may process a different part of the same message at the same time



overlapping precedence relation

Solution of TT scheduling

Formulation in terms of the Resource Constrained Project Scheduling with Temporal Constraints minimizing the schedule makespan (denoted PS|temp|C_{max})

Tree topology of nodes

- determines the rooting of messages

Unicast message

- chain of tasks executed on dedicated communication links
- chain starts at the source node and ends at the destination node

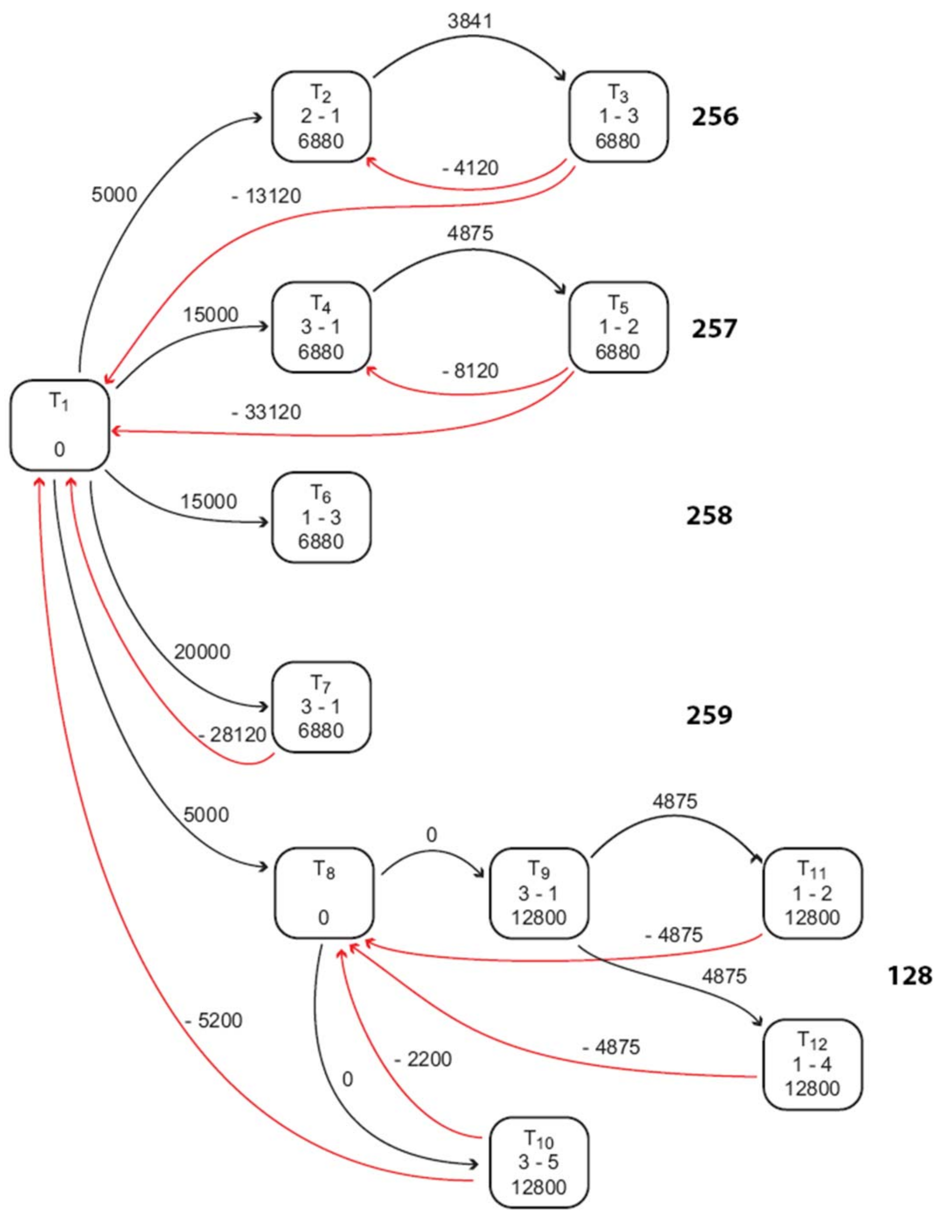
Multicast message

- tree of tasks

Modeling by PS|temp|C_{max}

<u>Task execution</u> corresponds to a <u>transmission of a message on the respective link</u>

- <u>Transmission delay</u> - processing time equal to T_{TD}
- <u>Link delay</u> - edge with positive weight T_{LD}
- <u>Release date</u> – edge with positive weight from dummy task to source task
- <u>Deadline</u> - edge with negative weight from sink task to dummy task
- <u>Required end-to-end delay</u> - edge with negative weight from sink task to source task



ILP formulation of the problem

input:	vector p, processing time of the tasks
	vector a, assignment of tasks to links
	adjacency matrix W
internal variables:	x_{ij} is equal to 1 iff task T_i is followed by task T_j
output:	vector s, start time of the tasks
	makespan C_{max}

$\min C_{max}$

subject to:

$$
\begin{aligned}
s_j - s_i &\geq w_{ij}, && \forall i, j \in 1, \ldots n \mid w_{ij} > -\infty \\
s_i - s_j + UB \cdot x_{ij} &\geq p_j, && \forall i, j \in 1, \ldots, n \mid i > j \wedge a_i = a_j \\
s_j - s_i + UB \cdot (1 - x_{ij}) &\geq p_i, && \forall i, j \in 1, \ldots, n \mid i > j \wedge a_i = a_j \\
s_i + p_i &\leq C_{max}, && \forall i \in 1, \ldots, n
\end{aligned}
$$

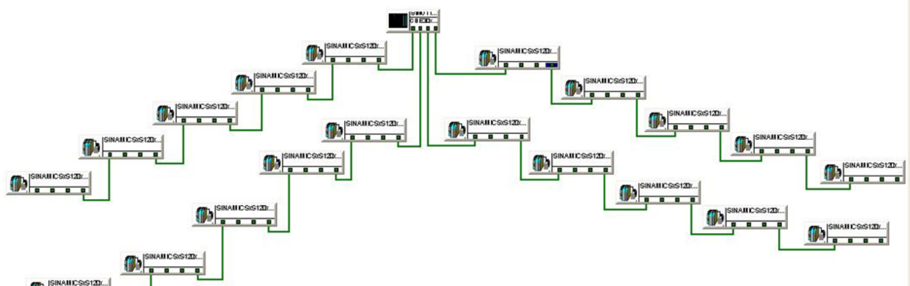
where: $x_{ij} \in \{0, 1\}$; $s_i, C_{max} \in \mathbb{R}_0^+$ and UB is a constant such that $UB > C_{max}$

Scheduling

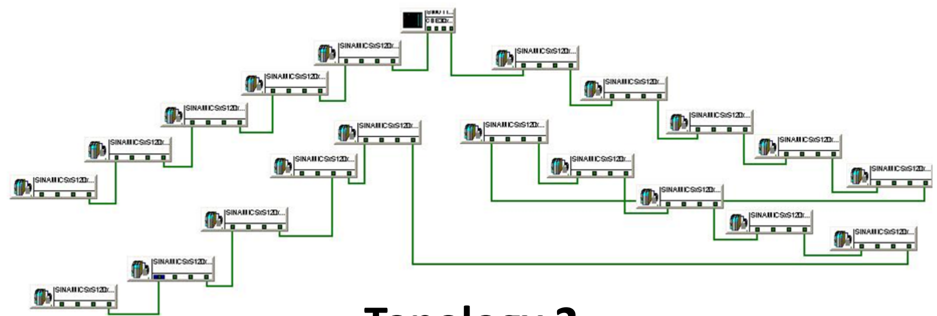
- Results returned by a heuristic using MTS (Most Total Successors) priority rule for the same instance

Topology	n	Cmax (µs)
1	140	47,20
2	290	116,00
3	440	150,40
4	960	219,20
5	510	120,26
6	880	154,66

- Schedules are computed in a fraction of time compared to ILP
- Cmax is close to optimal one, computation times are in the range of 10ms



Topology 1



Topology 2

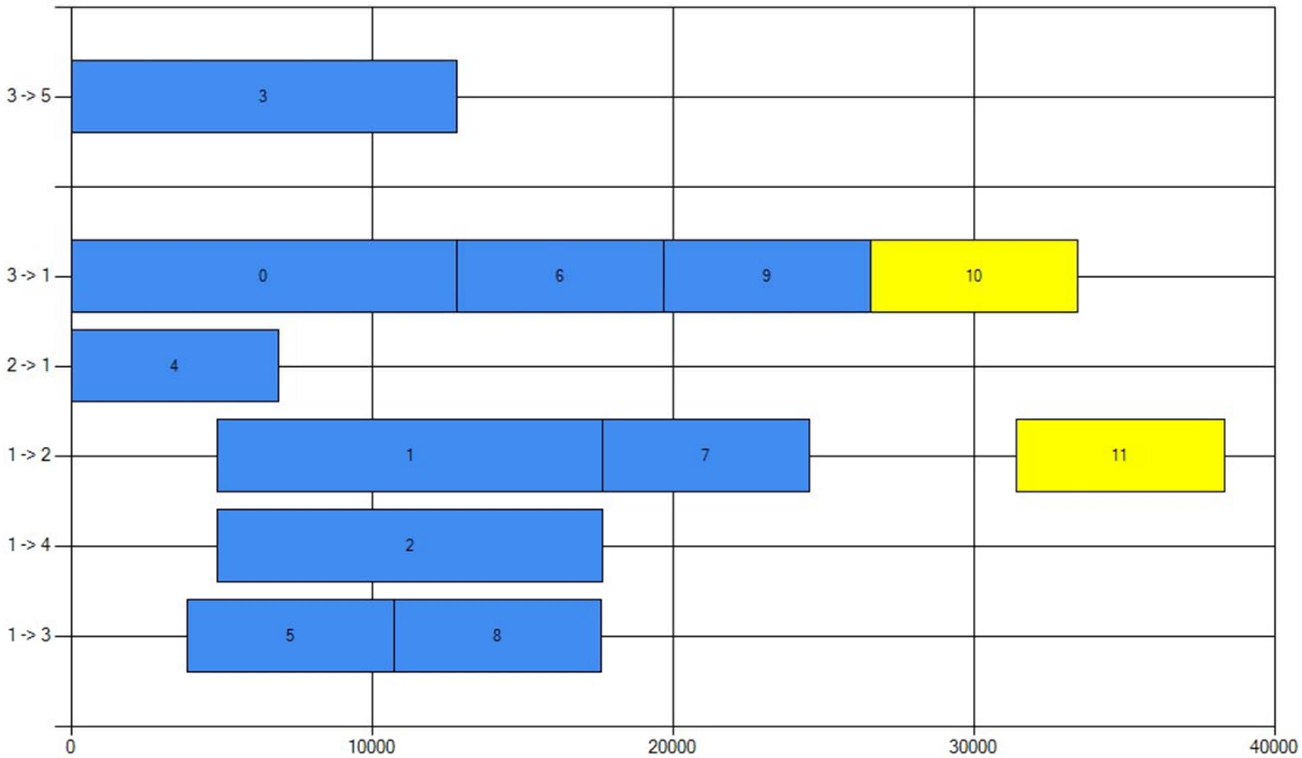
Adaptive Scheduling
(also called Rescheduling)

- What if we need to add a new message/node into the existing schedule?
- **Free rescheduling -** make a new schedule from scratch – finds a new place for all tasks (both new and original ones) - the algorithm is the same
- **Fixed rescheduling -** add new tasks without moving the original ones
- Schedule may “degenerate” (increase of makespan of the schedule) if we fix positions of original tasks, but scheduling process is faster (it schedules only new tasks)
- Degeneration is not very big, depends on the topology and may be eliminated by free rescheduling from time to time

Schedule with new messages and with fixed original tasks (fixed rescheduling)

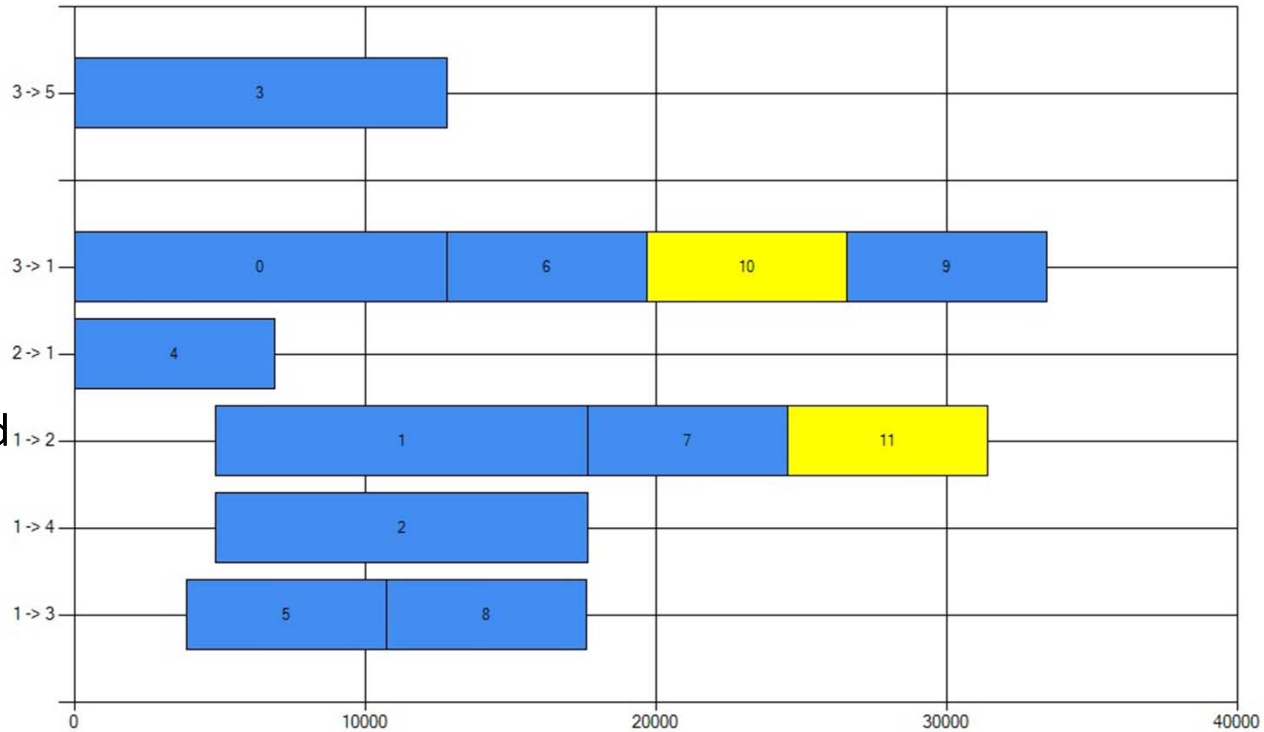
- Another instance for the same topology
- Blue tasks were in the original schedule
- The new message contains new tasks 10 and 11
- Task 11 caused the prolongation (it must wait until execution of his predcessor which cannot be placed earlier in the schedule – this breaks the priority rule)

Message	Tasks
1	4 -> 5
2	6 -> 7
3	8
4	9
5	11 -> 12
multicast	0, 1, 2, 3



Completely new schedule for the same messages (free rescheduling)

- The same instance as on the previous slide
- If we make a new schedule from all tasks (both original and new), task 10 will be placed earlier because it has higher priority than task 9 and its successor can be executed earlier in the schedule and in this case, the resulting schedule is optimal



Effect of adding new message

- If we cannot move the original tasks, we cannot use the priority rule for the whole schedule (it works only with messages added in current time)
- Adding a task with high priority often causes a prolongation of the schedule
- This **problem occurs mainly on the critical resource**
- The **length of prolongation is dependent on the number of nodes traversed** (number of tasks) starting from the critical resource
- The **prolongations do not sum-up**, i.e. rescheduling can make use of the unused space in the previous schedule

Experiments

- Tests of degeneration of schedule makespans (a comparison of the two previous ways – **free** and **fixed** rescheduling)
- Results are an average of **300 instances**
- Messages have different transmission delays - 6880, 4000, 2560 ns
- Messages were added in cycles. **5 messages were added in each cycle**.
- This schedule was compared with the schedule where all the same tasks were scheduled at once.
- Messages were generated at random
- Using two algorithms for fixed rescheduling:
  - **fixed 1** – we cannot move the tasks that were already scheduled
  - **fixed 2** – we can move the tasks that were already scheduled if they have no successors (this change cannot expand into the whole schedule)
- Tested for 5 priority rules, but there are no big differences between them (the results shown in this presentation apply to MST priority rule)
- Tested for two schemes but parallel scheme is more complex in rescheduling than the serial scheme

Parameters of the tests

Inputs

- **Topology** – there are 7 different topologies
- **Cycles** – number of cycles for adding new messages
- **n** – number of tasks in the original schedule
- $\mathbf{n_{new}}$ – number of added tasks (sum for all cycles)

Outputs

- C_{max}^{free} – average makespan of schedule when all tasks are scheduled at once
- C_{max}^{fixed} – average makespan of schedule when tasks are added in cycles
- ΔC_{max} – percentage difference between two previous makespans
- ΔC_{max}^{worst} – the biggest difference in makespans from all instances (percentage)
- T_C^{free} – average time of scheduling of all tasks at once (this is not directly comparable with the next parameter, because if we use scheduling of all tasks, we must do it in every cycle but T_C^{free} contains only time for scheduling after the last cycle)
- T_C^{fixed} – sum of scheduling all scheduling cycles
- d^{free} – average end to end delay for the first way
- d^{fixed} – same as the previous for fixed original tasks

Results – small instances (10 cycles)

Topology	n	n_{new}	FREE RESCHEDULING C_{max}^{free} (μs)	T_C^{free} (ms)	d^{free} (μs)	FIXED RESCHEDULING 1 C_{max}^{fixed} (μs)	ΔC_{max} (%)	ΔC_{max}^{worst} (%)	T_C^{fixed} (ms)	d^{fixed} (μs)
1	228	244	114,888	5	37,835	128,631	11	26	0	34,744
2	340	373	129,928	11	51,821	163,419	25	53	1	46,688
3	347	370	129,736	11	49,944	164,21	26	48	0	46,599
4	599	526	164,457	24	63,71	224,802	36	58	2	57,56
5	615	526	163,96	25	70,206	226,122	37	61	1	64,192
6	1186	697	196,689	70	88,536	268,092	36	52	6	84,784
7	416	217	163,955	4	59,263	169,318	3	7	0	56,804

- For small instances $\Delta C_{\max}$ is big in some cases
- Big differences in $\Delta C_{\max}$ and $\Delta C_{\max}^{worst}$ between difference topologies
- Fixed rescheduling is much faster
- No big differences between fixed rescheduling 1 and 2

FIXED RESCHEDULING 2 C_{max}^{fixed} (μs)	ΔC_{max} (%)	ΔC_{max}^{worst} (%)	T_C^{fixed} (ms)	d^{fixed} (μs)
121,517	6	22	0	35,768
159,784	23	54	0	47,285
162,464	24	53	0	47,099
221,743	35	55	2	58,386
221,24	37	59	2	64,624
266,311	35	50	9	85,589
167,691	2	5	0	57,275

Results – medium instances (25 cycles)

			FREE RESCHEDULING			FIXED RESCHEDULING 1				
Topology	n	n_{new}	C_{max}^{free} (μs)	T_C^{free} (ms)	d^{free} (μs)	C_{max}^{fixed} (μs)	ΔC_{max} (%)	ΔC_{max}^{worst} (%)	T_C^{fixed} (ms)	d^{fixed} (μs)
1	228	607	178,918	12	51,299	193,666	8	17	2	44,816
2	340	927	214,227	28	72,499	247,833	15	30	4	58,058
3	347	933	216,036	29	70,355	250,534	15	30	6	57,207
4	599	1313	235,024	59	87,282	303,912	29	46	8	69,46
5	615	1312	244,262	60	92,665	307,484	25	44	10	75,912
6	1186	1749	267,356	142	111,457	349,302	30	55	22	97,321
7	416	545	242,189	9	83,153	249,904	3	7	0	76,546

FIXED RESCHEDULING 2				
C_{max}^{fixed} (μs)	ΔC_{max} (%)	ΔC_{max}^{worst} (%)	T_C^{fixed} (ms)	d^{fixed} (μs)
184,234	3	14	0	46,61
241,61	12	31	3	60,149
244,542	13	31	2	58,974
301,311	26	44	7	72,219
304,813	24	38	7	78,06
345,445	30	49	21	97,875
243,139	1	3	0	77,329

- The percentage $\Delta C_{\max}$ is lower than in smaller instances and the absolute value of $\Delta C_{\max}$ is very similar (no increase)

Results – big instances (100 cycles)

			FREE RESCHEDULING			FIXED RESCHEDULING 1				
Topology	n	n_{new}	C_{max}^{free} (μs)	T_C^{free} (ms)	d^{free} (μs)	C_{max}^{fixed} (μs)	ΔC_{max} $(\%)$	ΔC_{max}^{worst} $(\%)$	T_C^{fixed} (ms)	d^{fixed} (μs)
1	228	2432	495,96	81	123,17	513,11	3	9	3	89,95
2	340	3714	642,31	194	206,71	676,50	5	9	9	105,03
3	347	3702	639,24	194	201,26	675,48	5	10	10	102,27
4	599	5262	647,76	401	232,48	714,81	10	17	35	119,66
5	615	5280	659,52	404	235,02	724,43	9	14	37	124,08
6	1186	6981	671,53	795	260,38	756,74	12	20	72	148,60
7	416	2174	636,59	76	201,42	646,78	1	3	4	174,65

FIXED RESCHEDULING 2				
C_{max}^{fixed} (μs)	ΔC_{max} $(\%)$	ΔC_{max}^{worst} $(\%)$	T_C^{fixed} (ms)	d^{fixed} (μs)
498,28	0	4	14	103,731
643,21	1	7	68	129,481
650,00	1	6	72	127,031
695,24	8	14	128	137,564
703,17	7	14	128	142,732
738,28	11	18	205	162,644
636,71	0	0	3	180,409

- Still no increase in absolute value of $\Delta C_{\max}$, so the percentage values are again lower
- For big instances, values of d^{fixed} are much better than values of d^{free}
- Now we can see that the second algorithm for fixed scheduling shows a little better results for $\Delta C_{\max}$, but the computation time is longer

We are looking for collaboration

- API related to application your experience
  - End-to-end dealy
  - Absolute time windows (deadlines, release dates)
  - Time windows relative to some event
  - Synchronization of tasks
  - Parameters for adaptivity
    - fixed messages-nodes …. combination of free and fixead resched.
    - constraints to be respected in mode changes
  - Redundance issues
- Typical topologies/data for benchmarks
- Integration of our algorithms to your products