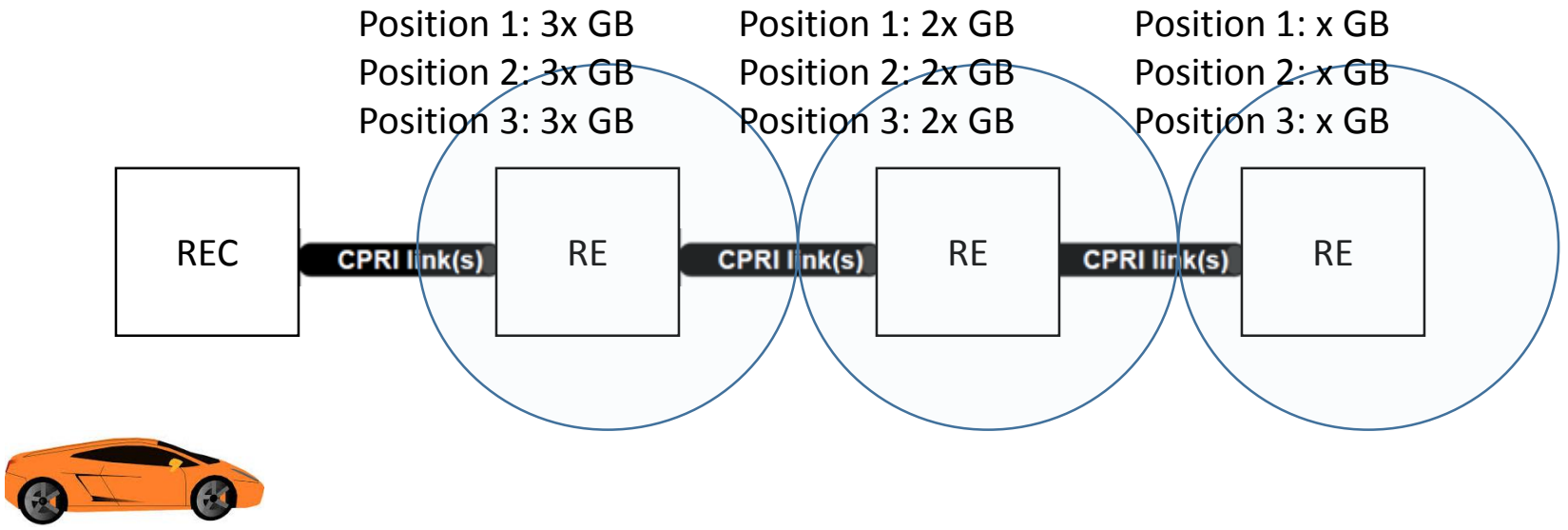
Scalable algorithms for Jitter minimized scheduling in reconfigurable CoE

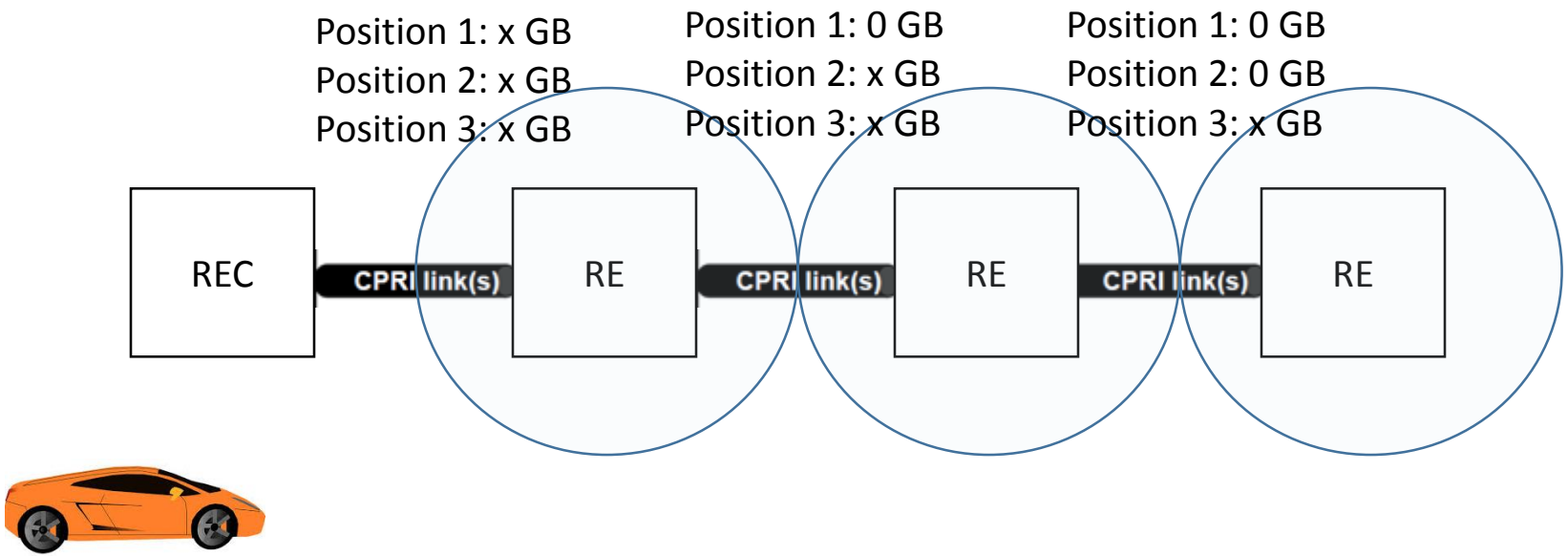
Divya Chitimalla

Reconfiguration in fronthaul

- 5G systems aim to achieve flexibility and reconfigurability in both radio access part and signal processing part
- Classified into bandwidth reconfigurability and network reconfigur*ability*
- *Bandwidth reconfigurability* : flexible on-the-fly bandwidth allocation to fronthaul links depending on need of RE
- Fronthaul can be dimensioned for current traffic rather than peak traffic, saving capacity and network equipment based on traffic profile, antenna capacity, cell size, user level QoE
- *Network reconfigurability* : ability to change fronthaul network topology on-the-fly based on requirements of cells
- Network can change based on co-ordination scenarios (CoMP), energy-efficiency schemes, etc., thus changing fronthaul topology

Bandwidth Reconfiguration



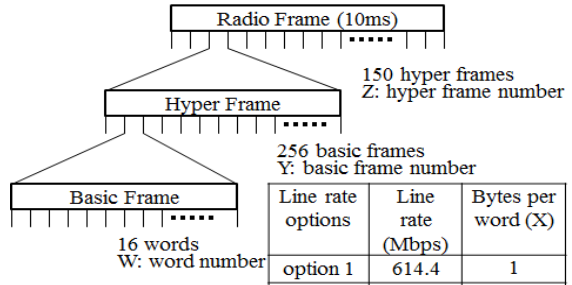


Bandwidth Reconfiguration - Antenna configurations

- CPRI allows rate negotiation to switch to different line rate when antenna configuration changes
- Antenna reconfiguration on/off (sectors, antenna, cell) can happen depending on user traffic requirement
- Power saving, interference reduction, frequency reuse etc. are use cases
- LTE cell breathing – adaptive coding (each sector has different coding)
- Vendor – cell adaptive (change frequency) 3GPP – each sector different modulation – within a sector different modulation (5Mhz to 20Mhz)

Antenna configuration	LTE Bandwidth	
	10 MHz	20 MHz
2x2 MIMO	1.2288 Gbps (IP rate 75Mbps)	2.4576 Gbps (IP rate 150Mbps)
4x2 (4x4) MIMO	2.4576 Gbps (IP rate 150Mbps)	4.9152 Gbps (IP rate 300Mbps)
8x2 (8x4, 8x8) MIMO	4.9152 Gbps (IP rate 300Mbps)	9.8304 Gbps (IP rate 600Mbps)

* Source: CPRI Specification v6.0 (Aug. 30, 2013)



Line rate options	Line rate (Mbps)	Bytes per word (X)
option 1	614.4	1
option 2	1228.8	2
option 3	2457.6	4
option 4	3072.0	5
option 5	4915.2	8
option 6	6144.0	10

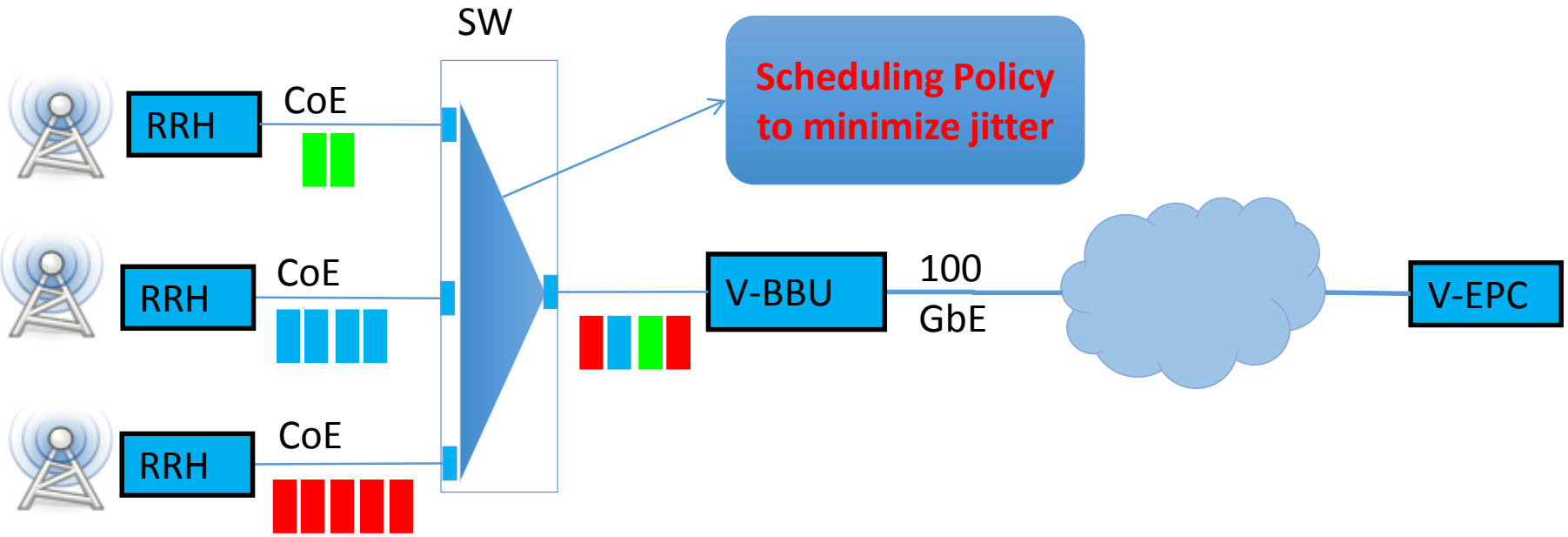
http://www.netmanias.com/en/post/blog/6237/c-ran-cpri-fronthaul/why-wdm-is-essential-in-c-ran-fronthaul-networks-ultra-high-cpri-link-capacity

Antennas for MIMO	Sectors	LTE BW (MHz)	IQ DR	CPRI DR	Line Rate
1/2	1	10/5		614.4	1
1/2/4	1	20/10/5		1228.8	2
1/2	3	10/5		1843.2	3
2/4	1	20/10		2457.6	4
2 + 1	2	20 + 10		3072.0	5
1/2	3 + 3	10/5		3686.4	6
1/2 + 1	3 + 3 +1	10/5 + 10		4300.8	7
4/8		20/10		4915.2	8
4/8 + 1		20/10+10		5529.6	9
				6144	10
				6758.4	11
				7372.8	12
				7987.2	13
				8601.6	14
8/16		20/10		9216	15

Fronthaul stringent requirements and problems

- CPRI switching can be very difficult (as it is very highrate)
- Need to enable statistical multiplexing by evolving from CBR CPRI to packet-based fronthaul
- Fronthaul must be dimensioned for peak traffic rather than current traffic
- Stringent performance requirements imposed by CPRI
  1) 100us of one way delay – previous study shows this is met
  2) 65ns of maximum variation in delay (i.e., jitter) – prev. study showed
  3) up to 10Gbps of throughput per RRH
  4) 10^{-12} of maximum bit error rate.

System Model: Could/Virtualized RAN



- **CPRI over Ethernet (CoE)**

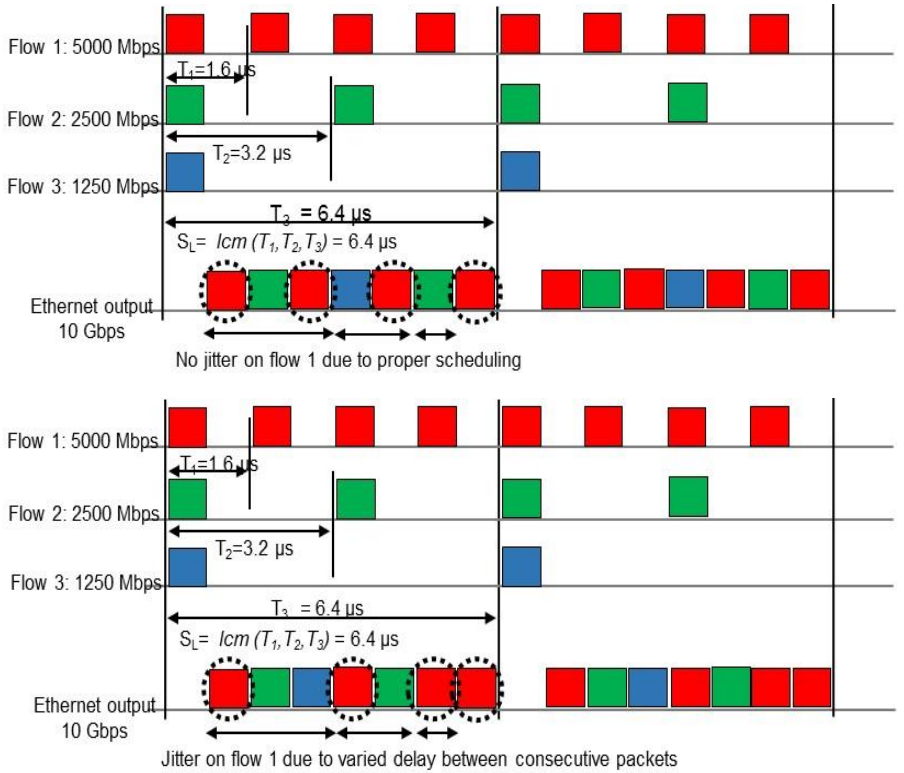
Jitter

Variation in delay : two types

- Intra frame jitter – due to different cycle lengths in frame – more prominent
- Reconfiguration jitter – due to adapted line rate .Change in scheduling – happens in larger timescale- **will be studied now**

- TDM signals are isochronous meaning that time between two consecutive bits is theoretically always same. This time is called unit interval(UI)
- Jitter is conventionally measured in unit interval peak-to-peak (UIpp) that is, difference between maximum and minimum time intervals in units of nominal UI
  - For example, for an E1 signal with a UI of 488 nanoseconds, if maximum interval were 500 nanoseconds and minimum 476, jitter would be(500-476)/488 = 0.05 UIpp

Intra frame jitter



- Flow 1, 2, 3 have different rates as shown , for example, there is jitter on flow 1 since proper scheduling policy is not enforced on the CoE switch, leading to variation in delay between consecutive flow 1 packets as shown above
- For ideal zero jitter, all the flows must be exact spaced according to their inter packet time without Ethernet encapsulation

Exhaustive search Comb fitting algorithm

Slide combs until they perfectly fit among each otherIf they do not perfectly fit, break a tooth and put them in adjacent available slot.

ALGORITHM 2: COMB FITTING

Input: Schedule of CoE flows given by basic offset algorithm
Output: Non-conflicting schedule of CoE packets
Step 1: Form all possible permutations of order of flows from 1 to N_F (N_F! different sequences) denoted by {SE$_i$}

tep 2: **for** each sequence SE$_i$ € *{SE$_i$}*
for j in SE$_i$
initialize: $matcomb$ *as first element in {SE$_i$// matcomb is temporary matrix*
$matcomb$ = ***matcombine(***$comb^j$, $matcomb$***)***
end
Calculate jitter *matcomb*
end
Pick $matcomb$ with least amount of jitter

Not scalable

MATCOMBINE SUBROUTINE

Input: $comb^i$, $comb^j$ (any two schedules)
Output: Combined non-conflicting schedule
Step 1: Initialize: $matcomb$ as a matrix with length as sum lengths of $comb^i$, $comb^j$
Step 2: Take the longest sequence out of $comb^i$, $comb^j$ and add its contents to $matcomb$, call the other matrix $mattemp$
Step 3: Shift $mattemp$ by the multiples of E_{TS} to form a perfect non-conflicting schedule with $matcomb$
Step 4: **if** there is a success in this procedure
Copy $mattemp$ to $matcomb$ and **return** $matcomb$
Step 5: **else**
Copy the non-conflicting packets of $mattemp$ to $matcomb$
for all conflicting packets in $mattemp$
Find the nearest open timeslot which can fit in the packet and update $matcomb$
end
return $matcomb$
end

Explore scalable algorithms to find Jitter minimized schedule in reconfigurable CPRI

- JOCN work had exhaustive search which is not scalable for dynamic large fronthaul which has multiple switches scheduling CoE packets

- Distance constrained scheduling
- Pinwheel algorithm
- SMD algorithm

Distance constrained scheduling

- A common approach to scheduling hard real-time tasks with repetitive requests is *periodic task* model [l], in which each task T_i has a period P_i and an execution time e_i
- T_i must be executed once in each of its periods
- Some real-time tasks must be executed in a (temporal) *distance-constrained* manner, rather than just periodically
- Temporal distance between any two consecutive executions of a task should not be longer than a certain amount of time – within tolerable jitter

Scheduling jobs with temporal distance constraints

- Job scheduling problems for real-time jobs with temporal distance constraints (JSD) are presented
- In JSD, start times of two related jobs must be within a given distance
- General JSD problem is NP-hard
- Define multilevel unit-time JSD (MUJSD) problem for systems with m chains of unit-time jobs in which neighboring jobs in each chain must be scheduled within c time units
- Efficient algortihms exist to solve this - $o(n^2)$ time algorithm, where n is total number of jobs in system, and also an $o(m^2c^2)$-time algorithm

Scheduling problem (single processor)

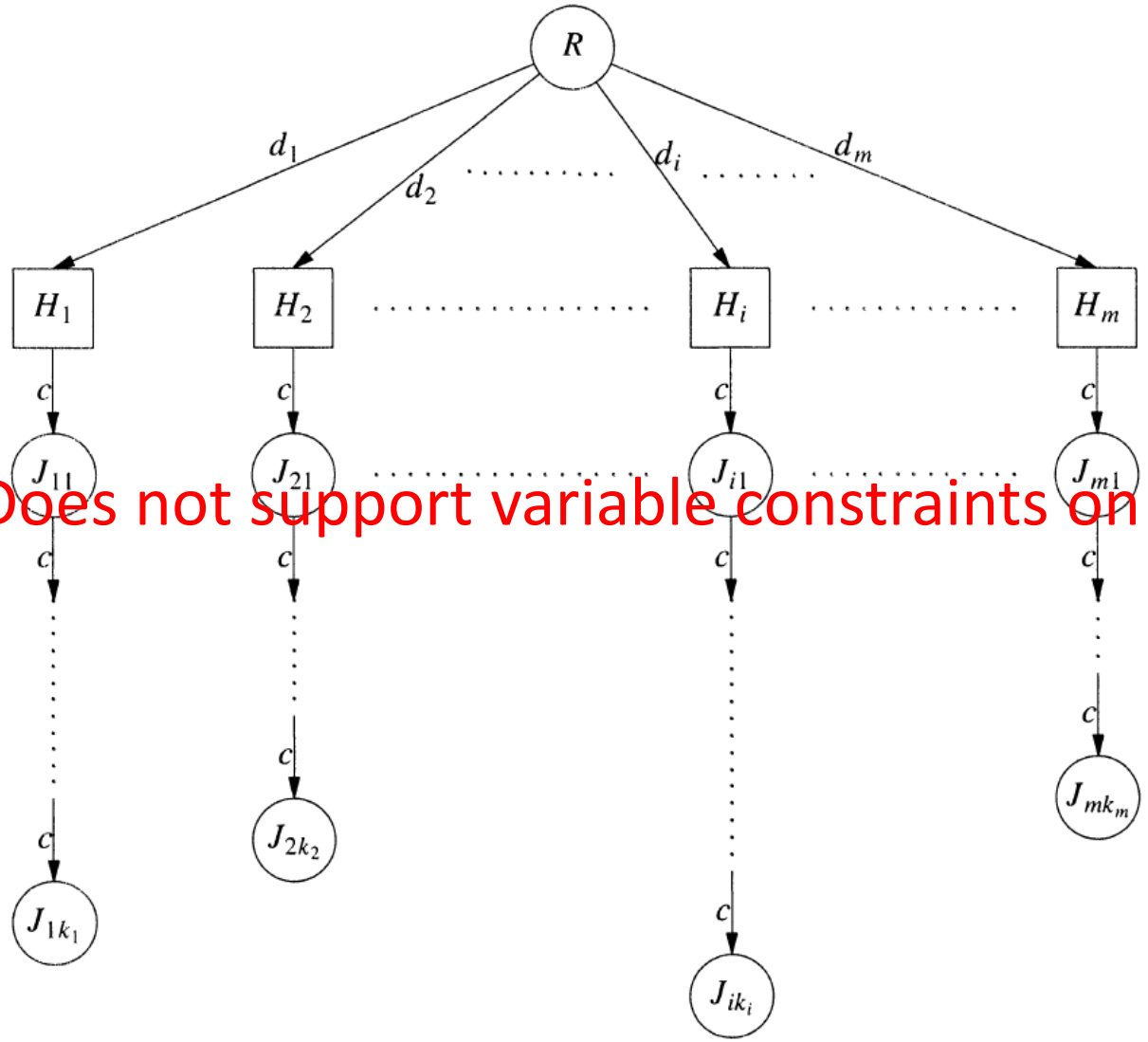
- Given a set of jobs J $\{J_1, J_2 J_n\}$, in which each job J_i has execution time e_i, ready time r_i, and deadline d_i, $1 <= i <= n$, job scheduling with distance constraint (JSD) problem is to find a start time function f such that for $1 <= i, j <= n$, and $i != j$,

  (1) $f(J_i) >= r_i$,

  (2) $f(J_i) + e_i < d_i$, and

  (3) $| f(J_i) - f(J_j)| <= w(J_i, J_j)$ distance constraint between $J_i J_j$

Related problems: linear array problem (LAP), bandwidth minimization problem (BMP)



The m-chain tree structure of the MUJSD problem.

In an MUJSD system, job set is divided into chains of jobsIn each chain, only first job has a deadline

other jobs have a constant distance constraint with their immediate predecessors

Does not support variable constraints on different flows

SMD algorithm

- Among all jobs remaining to be scheduled, we always pick job with largest number of successors to schedule next
- If job is a head job we schedule it at empty slot, if any, closest to and before job's deadline
- If job is a tail job with its predecessor scheduled at slot s, and if there exists any empty slot between time s and time s + c, we can simply schedule job at empty slot closest to time s + c
- Otherwise we schedule tail job at slot s and then reschedule its predecessor
- Lemma: If SMD terminates successfully without reporting "unschedulable," schedule generated by algorithm is a feasible schedule for job set
- Lemma: If an MUJSD system is schedulable, then SMD will find a feasible schedule for it.

```
ALGORITHM SMD
Step 1. Sort the jobs into S_1, S_2, ..., S_n with nonincreasing number of successors.
Step 2. For i from 1 to n do {
            if S_i is a head job H_j then SCHED(i, d_j, 0)
            else { suppose S_i's predecessor is scheduled at slot s;
                   SCHED(i, s + c, 0); }
        }
procedure SCHED(i, t, r);
{
  if t = 0 then output "unschedulable" and stop
  else if slot t is empty then f(S_i) = t - 1   /* schedule S_i at slot t */
       else {
            suppose slot t is now assigned to S_k;
            if (S_k is the predecessor of S_i) or (r = 1 and S_k is reschedulable)
            then { f(S_i) = t - 1;   /* schedule S_i at slot t */


                   SCHED(k, t - 1, 1); }   /* reschedule S_k */
            else SCHED(i, t - 1, r);
          }
}
```

Pinwheel scheduling algorithm

- **Definitions:** *v* is list of *n* positive integers. A *pinwheel schedule* for *v* is a doubly infinite sequence drawn from labels *{1,...,n}* such that each label *i* occurs at least once in each window of v_i consecutive positions.
- If such a schedule exists for *v*, then *v* is *schedulable. v* is nondecreasing

  value $\sum(1/v_i)$ is *density* of *v*, written *d(v)*.
- **Background:** necessary but not sufficient condition for schedulability is having density at most *1.* If each v_i is a power of *2*, then density at most 1 is sufficient.
- (Chan-Chin) If $d(v) \leq 5/6$, then *v* is schedulable.
- **Comments:** earliest posing of problem showed that $d(v) \leq 1/2$ is sufficient to make *v* schedulable. If *v* is schedulable, then there is a periodic pinwheel schedule for *v* with period at most $\prod v_i$
- For general problem, Chan and Chin gave various algorithms that proved sufficiency of $d(v) \leq 2/3$ and $d(v) \leq .65$This was improved to *0.7* in Chan and Chin. Fishburn and Lagarias further improved it to *0.75.*
- Decision problem of schedulability is in PSPACE. For density *1*, problem is in NP but may not be NP-hard. Fast algorithms for generating schedules have also been studied.

Distance-constrained scheduling algorithms

- J_{i1}, J_{i2}, J_{i3}.,.Task T_i, has an execution time e_i and a (temporal) distance constraint c_i.

$$\rho(\mathrm{T}) = \sum_{i=1}^{n} \rho(T_i) = \sum_{i=1}^{n} \frac{e_i}{c_i}.$$

- Density thresholds (schedulability conditions) for guaranteeing a feasible schedule for a pinwheel problem instance have also derived, to be 1/2, 13/20, 2/3, 0.6964, and 0.7, for $\mathbf{S_a}$, $\mathbf{S_x}$, $\mathbf{S_{bc}}$, $\mathbf{S_{by}}$, and $\mathbf{S_{xy}}$
- T_i is transformed into an element, a_i, in pinwheel instance, where a_i = floor(c_i/e_i)
- Every e_i consecutive time slots allocated to i^{th} symbol of pinwheel instance are actually allocated to one job request of task T_i
- Algorithms designed for pinwheel problem are used to solve DCTS problem

Distance-constrained scheduling algorithm based on s_x

- **Sx** first tries to find an integer *x*, $a_i/2 < x <= a_i$, and specializes **A** with respect to ***{X}*** to get specialized multiset **B**
- Starting from $x = a_1$, down to $x = a_1/2 + 1$, **Sx** specializes **A** with respect to *{x)* and chooses an *x* that minimizes p(B), or chooses first *x* which makes **P(B)** <= 1 (or it finds that no such integer exists)
- For example, If **A** = (4, 6, 7, 13, 24, 28, *33}* is specialized with respect to {4}, specialized multiset is **B** = (4,4,4, 8,16,16,32} with a total density of 33/32 > 1, and if **A** is specialized with respect to *{3],* specialized multiset is **B** = *(3,* 6,6,12,24,24,24) with a total density of *7/8.* **Sx** will choose *x* = 3 and get **B** = {3,6,6,12,24,24,24)
- S_R is operation that is used to specialize a general DC task set
- S_R is a generalization of **Sx,** Sr specializes **C** with respect to *{r},* where **r is real number** chosen from range ($c_1/2$, c_1) so that specialized task set has a minimum density increase
- S_R uses polynomial algorithm to find best **r** and then specializes distance constraint multiset C with respect to *{r}.*

Scheduling algorithms

- Scheduler **Sr** can schedule task sets with temporal distance constraints.
- Distance-constrained task set with *n* tasks can be feasibly scheduled by using Scheduler Sr as long as its total density is less than or equal to $n(2^{1/n} - 1)$
- Deterministic guarantee that all tasks will meet their deadlines as long as total density is held within density threshold
- If total density of a DC task set after specialization is less than or equal to 1, DC task set can be feasibly scheduled by Scheduler Sr
- Sr supports scheduling variable flow constraints required for Jitter minimization in fronthaul
- Multiprocessor scheduling will be explored to form schedules for large number of switches in fronthaul

References

- Chan, MY.; Chin, Francis; Schedulers for larger classes of pinwheel instancesAlgorithmica 9 (1993), no5, 425--462
- Mee Yee Chan, Francis YLChin; General schedulers for pinwheel problem based on double-integer reduction, IEEE Transactions on Computers, 41 (1992), 755--768
- Fishburn, PC.; Lagarias, JC.; Pinwheel scheduling: achievable densitiesAlgorithmica 34 (2002), 14--38
- Holte, Robert; Mok, A; Rosier, Louis; Tulchinsky, Igor; Varvel, Donald; pinwheel: A scheduling problem, Proc22nd Hawaii IntlConfSystems Sci., (1989), 693-702
- Holte, Robert; Rosier, Louis; Tulchinsky, Igor; Varvel, Donald; Pinwheel scheduling with two distinct numbersMathematical foundations of computer science 1989 (Por\polhk abka-Kozubnik, 1989), 281--290, Lecture Notes in ComputSci., 379, Springer, Berlin, 1989, and TheoretComputSci100 (1992), no1, 105--135
- Lin, Shun-Shii; Lin, Kwei-Jay; A pinwheel scheduler for three distinct numbers with a tight schedulability boundAlgorithmica 19 (1997), no4, 411--426

/* Input: $\mathbf{T} = \{T_i = (e_i, c_i) \mid 1 \le i \le n\}$, where $\mathbf{T}$ is a DC task set and $c_i \le c_j$ for all $i < j$. */

/* Output: r^*, $\Phi_{\mathbf{T}}(r^*)$, and $\mathbf{T}' = \{T_i' = (e_i, b_i \mid 1 \le i \le n\}$, where $b_i \mid b_j$ for all $i < j$. */

1. **for** $i := 1$ **to** n **do** $l_i = c_i / 2^{\lceil \log(c_i/c_1) \rceil}$;
2. sort $(l_1, l_2, \ldots, l_n)$ into nondecreasing order and remove duplicates;
   let $(k_1, k_2, \ldots, k_u)$ be the resulting sequence;
3. **for** $i := 1$ **to** n **do** put T_i into subset τ_{l_i};
4. **for** $v := 1$ **to** u **do** $\rho(\tau_{k_v}) := \sum_{T_i \in \tau_{k_v}} e_i / c_i$;
5. compute $\Phi_{\mathbf{T}}(k_u)$ according to (4.1);
6. **for** $v := u - 1$ **downto** 1 **do** $\Phi_{\mathbf{T}}(k_v) := \frac{k_{v+1}}{k_v} \Phi_{\mathbf{T}}(k_{v+1}) - \rho(\tau_{k_v})$;
   /* see Lemma 1 */
7. find r^* such that $\Phi_{\mathbf{T}}(r^*) = \min_{r \in \{k_1, k_2, \ldots, k_u\}} \Phi_{\mathbf{T}}(r)$;
8. **for** $i := 1$ **to** n **do** $b_i := r^* \cdot 2^{\lfloor \log(c_i/r^*) \rfloor}$;
9. output r^*, $\Phi_{\mathbf{T}}(r^*)$, and $\mathbf{T}' = \{T_i' = (e_i, b_i) \mid 1 \le i \le n\}$.

Polynomial time algorithm for MUJSD

- PMD algorithm generates a feasible schedule for a schedulable MUJSD system with m job chains and distance constraint c in $O(m^2c^2)$ time
- Sort job chains and re-index them so that chain with a larger tail deadline has a larger index (ties are broken arbitrarily)
- Create a pseudochain 0 which has only one job with a deadline 0 (note that head deadlines of all other job chains are larger than 0)This pseudochain serves as a marker to trigger final cleanup process which will move jobs scheduled before time 0 to empty slots after time 0
- Step 2 sets initial tail positions of v-chains and initializes counter p which points to current job chain being scheduled,

ALGORITHM PMD

Step 1. Sort and reindex the job chains in nondecreasing tail deadline order
(i.e., $d_{ik_i} \le d_{i+1,k_{i+1}}$, for $1 \le i < m$);
Create a pseudochain B_0 with $d_0 = 0$ and $k_0 = 0$;

Step 2. Set $D_1 = d_{mk_m}$;
For $i = 2$ to c do { set $D_i = D_{i-1} - 1$; }
Set $p = m$;

Step 3. Let y be the index of the v-chain with $D_y = \max_{1 \le i \le c} D_i$;
If $d_{pk_p} \ge D_y$
then { /* schedule chain p */
If $p = 0$ goto Step 4;
initialize S-list S_p to be $[(k_p; D_y; c)]$;
reset $D_y = D_y - (k_p + 1)c$;
set $p = p - 1$; }
else { /* reschedule chains */
Among scheduled chains $p + 1$ to m find a chain B_i,
and locate the job J_{ij}, $j \ge 0$, where
(S3.1) J_{ij} is the latest job in B_i scheduled before D_y, and
(S3.2) $d_{ij} \ge D_y$.
If B_i and job J_{ij} exist
then reschedule jobs $J_{i0}, J_{i1}, \ldots, J_{ij}$; /* as in §4.3 */
else reset $D_y = D_y - \lceil (D_y - d_{pk_p})/c \rceil c$;
}
Repeat Step 3.

Step 4. If there is any head job scheduled before time 0
then output "unschedulable";
else output the m S-lists.

MUJSD problem with distinct distance constraints is NP-completeWith distinct distance constraints, even if we restrict graph to a bilevel tree (not a bilevel chain tree) we can show that problem is still NP-complete.