Handling Aperiodic Overload

Brian Kelein Ngwoh Visas
Electronics Engineering
Hamm-Lippstadt University of Applied Sciences
Lippstadt, Germany
brian-kelein-ngwoh.visas@stud.hshl.de

***Abstract*—Handling Aperiodic Overloads is a critical concept when it comes to computing and scheduling tasks in a system. This is because Real-Time Systems play an important role in complex systems, for example, railway switching systems, automotive and avionic systems [1]. In a system, we have tasks, and each job has to be completed within a particular Deadline. But when a system has so many jobs to compute, then there is the possibility that all tasks are not going to meet their deadlines, and when this happens then, a system is unable to perform its functions properly. For example, imagine a rail switching system missing a deadline because of an overload, and then lives are in danger because then accidents are possible. In this, we briefly talk about these Overloads, how they come about, and possible ways to manage these overloads such that almost if not all tasks could meet the deadlines set for them. Also, we look at the FTT-CAN(Flexible Time-Triggered Communication on Control Area Network) thoroughly explained in [2] and [3]. Here we see how message streams are being scheduled based on this FTT-CAN protocol and how this process helps manage the overload situation to produce optimal results.**

***Index Terms*—Handling Overload,Scheduling analysis, FTT-CAN Protocol**

I. Introduction

A. Definitions

The way Systems are designed, most if not all require a modus operandi. In this form of Operation, they all have to complete tasks within a particular time to ensure they function correctly, which is also known as their Deadlines. So the main reason why some systems are called Real-Time systems is that they can meet all the Deadlines without missing out on any. According to [4] therefore the objective usually is to minimize task responsive time, unlike traditional computer systems. The emphasis here is on the fulfillment of timing requirements.

1) An Overloaded System: Generally, a system always has tasks to fulfil, which has to be done within a time frame. When a system cannot meet up with all the deadlines, this reduces its computing power and refers to such a system as an Overloaded System. Also in due to Overload, some tasks are left unattended. When such tasks are taken up and completed after their deadlines, the system is an Under loaded system. From [6] there exists a schedule that will meet every task.

2) Aperiodic Tasks: Aperiodic in this context refers to tasks that have no idea of future instances when scheduling. These tasks are separate from each other and the Offline scheduled tasks. They have the following characteristics, Time, Firm absolute deadline and Value. Most of this is unknown at design time, and aperiodic tasks can also handle non-critical routine activities. [7]

Symbol	Definition
t	system time instant
$\mathcal{T}$	set of real-time tasks
τ_i	real-time task, $\tau_i \in \mathcal{T}$, where i is index of the task
r_i	the request time instant of τ_i
c_i	the required execution time of τ_i
rc_i	the remaining execution time of τ_i
d_i	the deadline of τ_i
$f_{i,j}$	the j-th indivisible fragment of τ_i
$f_{i,e}$	the last indivisible fragment of τ_i
$s_{i,j}$	the start execution time of $f_{i,j}$
$c_{i,j}$	the required execution time of $f_{i,j}$

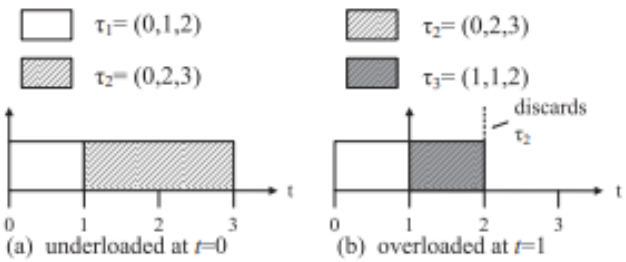

Fig. 1. Example for overloaded and underloaded [5]

3) Value: It is a metric for the system's profit from finishing a task on time. Offline planned tasks are never evaluated for rejection when addressing overload situations; therefore, only aperiodic tasks are connected with values. The values are cumulative, which means that the aggregate of two sets of tasks can be compared. Tasks make a difference in the system by adding value.

4) Typical Scheduling Schemes: Based on the problems that arise due to overloads, Academic enthusiasts have come up with Algorithms whose main aim is to predict and handle these overloads, and in [1] it is divided into three main classes.

a) Best Effort: In this category, Algorithms have no prediction for overload. When a new task arrives, it is taken directly into the ready queue. Here system performance would be controlled adequately through priority assignment, which takes task values into consideration.

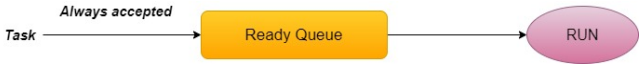

Fig. 2. Best Effort [1]

b) With Acceptance: Here we deal with Algorithms that have an admission control, and a guarantee test is performed at every job activation. Every time there is an incoming task the schedulability of those set of tasks is verified based on worst-case assumptions, and this routine is almost always guaranteed. When a set of task are schedulable then it is accepted into the system. else it is rejected.

Fig. 3. With Acceptance [1]

c) Robust: This category includes those algorithms that separates sets of tasks by timing constraints and importance. In doing this two policies are considered one for task rejection and the other for task acceptance. With this category when a new task enters the system, the schedulability based worst-case assumption is verified. Once found schedulable it is accepted into a ready queue, else rejected.

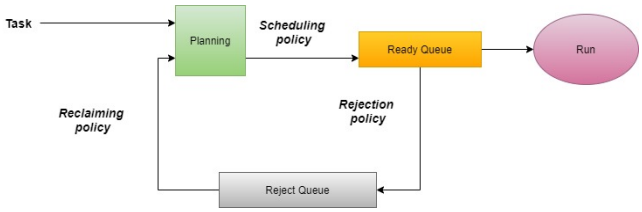

Fig. 4. Robust [1]

B. Some Important Terms

1) Performance metrics: When tasks are activated dynamically, an overload occurs. There are no algorithms that can guarantee a feasible schedule of a set of task. Since one more task will miss their deadlines, late tasks should be the less important ones to achieve graceful degradation. Distinguishing between time constraints and importance for a set of tasks is very crucial for an overloaded system. For a system to emphasize the importance, each task is associated with an additional value parameter. This Value is essential concerning Value such that most times, this Value is set to computation time. Moreover, other times set to an arbitrary integer number or the ratio of an arbitrary number and the task computation time, which can be referred to as Value density as analyzed by Buttazzo in [1]. The system can use this Value to make scheduling decisions.

The graphs above describe Utility functions that can be associated to a task to describe its importance.In [1] When

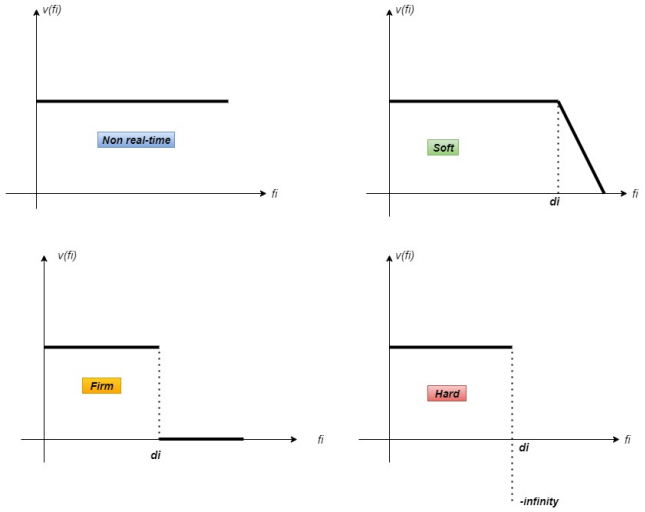

Fig. 5. Utility functions that can be associated to a task to describe its importance [1]

the importance of a task set has been defined, then we can measure the performance of a scheduling algorithm by accumulating the values of a task utility functions computed at their completion time.B is defined as the cumulative value of a scheduling algorithm

$$\Gamma_B = \sum_{i-1}^{n} v(f_i) \tag{1}$$

Below are some analysis of maximum values in [1]

Furthermore a scheduling algorithm is said to be at its Optimum, when it is able to maximize the value achievable on a task set.

2) Competitive factor: A scheduling algorithm's cumulative value for a task set is a measure of that task set's performance. The competitive factor, developed by Baruah et al., is a parameter that assesses a scheduling algorithm's worst-case performance. [6] We can comfortably say a scheduling Algorithm posseses a competitive factor ϕ_A provided it can assure you, you get a cumulative value $\Gamma_A \geq \phi_A \Gamma^*$. Here Γ^* is the cummulative value, gotten by the *optimal clairvoyant scheduler*.From above we can see that the Competitive factor is a Real number $\phi_A \in [0, 1]$.If the overload is endless, no online algorithm can guarantee an over-zero competitive factor. Nevertheless, overload is temporary and usually short; therefore, planning algorithms with a high competitive factor are desirable. The simple EDF algorithm, unfortunately, has a zero competitive factor without any guarantee. To demonstrate this result, an overload situation is sufficient to determine whether the cumulative value obtained by EDF can be arbitrarily small compared to that of the clairvoyant scheduler.

a) Definition:

II. Handling Overloads

A. Overload Handling

1) Problem Formulation: In this section, we talk about some of the real-time system designers' choices. The choice

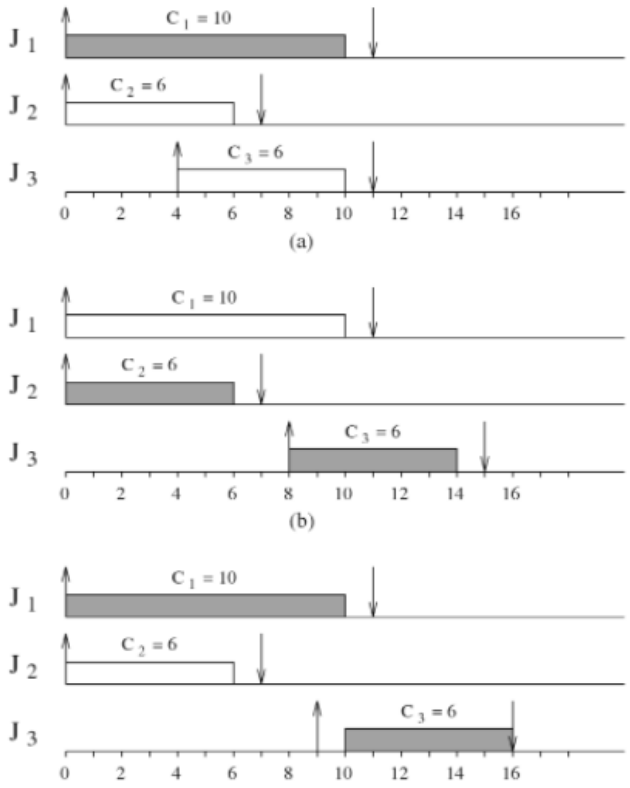

Fig. 6. No optimal online Algorithm exist in overload conditions, since the schedule that maximizes Γ depends on the knowledge of future arrivals $\Gamma_{max} = 10$ in case (a), $\Gamma_{max} = 12$ in case (b) and $\Gamma_{max} = 16$ case (c). [1]

of which algorithm to use depends in part on the system's requirements. One section of the system could require harsh guarantees and complex constraints in real-time, while another has less stringent requirements. Therefore, the designer needs to choose either an off-line or on-line programming algorithm or perhaps both, as stated above. The potential need to foresee dynamic operation, such as aperiodic or sporadic tasks and overload, which has an influence on selecting the scheduling algorithm. An associated problem is that most OSs provide only a tightly integrated fixed scheduling algorithm [8] with the kernel. This setup does not allow designers to choose which algorithm to use, even if the algorithm provided is not the most appropriate.

B. Scheduling schemes with example Algorithms

1) ROBUST Algorithm (Best Effort): Here we have Resistance to Overload By using Slack Time(ROBUST). From the name, we see it is using the slack time to schedule tasks in phases and also analyse its EPU performance, but in this paper, we will look at the main features of this Algorithm, but more can be found in [4]. First, we will look at a case where the Slack factor of all tasks is set at a minimum of 2.0, and a different look at a case where the slack factor could be another value.Here are the overview as explained by [4]

1) The ROBUST algorithm partitions the interval overloaded into an equal number of adjacent phases—Phase-1, Phase2,..... Phase-2n. 2) The length is equal to that of each phase's previous odd (numbered) phase. That is, for all $I1 \leq i \leq n$, the length of Phase-2i equal to the length of phase-$(2i-1)$. 3) During every strange phase, the algorithm does "utile" work, performing tasks to complete. This is achieved by not performing an entire task in each strange phase, thereby ensuring that EPU contributes to the amount of time spent on this task. 4) Following the next even phase, after each odd phase, the most valuable task for the next odd phase is set up. Our most valuable task is the one that can contribute almost immediately to the EPU; this is the most significant task,namely to perform the maximum amount of time between every task currently feasible... However, no task must be completed in the same phases—such completion is a bonus and helps raise the EPU by more than half of their guaranteed value. In order to guarantee their performance, the scheduling algorithm ROBUST requires that the slack factor of all tasks exceed a particular minimum value. The semanticization of specific applications can, in practice, enable a trade-off of various slack factors. Designing the developers of security-critical real-time systems aims to maximize the minimal factor of slackness in a task system.

2) D^{over} Algorithm(With Acceptance): Koren and Shasha [KS92] found the best competitive factor possible by any online algorithm, called the D^{over}, which has been proven to be ideal (that is, 0.25). Unless there is any overload,$D^{over}$is compatible with EDF. When a prepared task reaches Last Start Time (LST), an overload is detected; that is to say, the time at which the task remains the same as until the deadline. Some task needs to be dropped at this time: either the task reaching its LST or another. [1] The ready tasks set in D^{over} is divided into two disjointed sets: privileged and waiting tasks. If a task is anticipated, it is a favourite. Nevertheless, whenever a task is planned due to an LST, all ready tasks (whether preempted or not performed) are waiting for tasks. If the J_z task reaches the LST, an overload is detected, then the value of J_z is compared to the total V_{priv} value of all the tasks (including the value v_{curr} of the currently running task).

$v_z > (1 + \sqrt{k})(v_{curr} + V_{priv})$

(where k is the ratio of the highest value density and the lowest value density task in the system), then Jz is executed; otherwise, it is left out . If J_z is executed, all the privileged tasks become waiting tasks. Task J_z can, in turn be dropped in favour of another task J_x that reaches its LST, but only if;

$v_x > (1 + \sqrt{k})v_z$.

We notice that having the best competitive factor between all online algorithms does not necessarily mean the best possible performance. D^{over}can reject tasks with values above the existing task but not beyond the threshold that ensures optimality to ensure the best competitive factor. This mean D^{over} does not take advantage of random sequences to handle worst-case sequences and may decline more value than necessary.In [1] further analysis is done to compare and see how effective this Algorithm can be compared to others.

3) RED Algorithm (Robust scheme): This is a solid scheduling algorithm that Buttazzo and Stankovic propose [BS93, BS95] for dealing with aperiodic desktop work in overloaded surroundings. RED (Robust Early Deadline) The algorithm combines many features synergistically, including graceful overload degradation, tolerance for deadlines, and resource reclaim. It works with good performance in normal and overload conditions and can predict not only deadline failures but their size, duration, and overall system impact. Each RED task of $J_i(C_i, D_i, M_i, V_i)$ has four parameters: a (C_i) time in the worst case, a relative (D_i) date, a (M_i) tolerance in the deadline, and a (V_i) importance value. [1] The tolerance to a deadline is when a task is allowed to delay, i.e. the amount of time a job can execute and produce a valid result after its deadline. In many real applications like robot and multimedia systems, this parameter can be helpful where the timing semantics are more flexible than the theory generally allows.

4) SMT Based Scheduling: The main job is to formalize the problem with the first-order language to use SMT to solve this problem of satisfaction. To represent the formalized problem, we use a Sat model. This sat model is a series of logic formulae of the first order (in linear format) which express all of the constraints that should be met in the desired optimum schedule. Two types of limitations exist system constraints and target restrictions. System restrictions are based on the system in question. For example, a schedule should ensure there is no overlap in the time domain for uniprocessors with the execution of two tasks. Objective limitations are based on the planning target.Once the model is built, an SMT solver can be inserted (e.g., Z3). The SMT solver returns a solution model. This solution model interprets all of the variables defined in the Sat model, and all logic formulas in the Sat model are assessed as valid according to interpretation. It means the problem of satisfactoriness represented by the sat model is solved, and a desired optimal schedule can be generated based on this interpretation. [5]

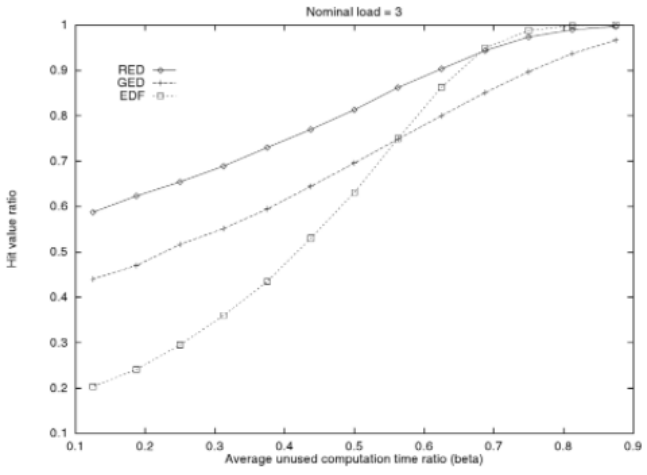

Fig. 7. Performance of various EDF scheduling schemes: Best Effort(EDF), guaranteed (GED), and Robust (RED) [1]

III. Application of Handling Overloads in FTT-CAN (Flexible Time-Triggered Communication on Control Area Network) Protocol

A. FTT-Can Protocol

The CAN bus represents an industry standard for communication in cars, manufacturing plant and medical devices, computer or micro-controller-based applications. The reasons for its success include low costs, standardization, reliability and the possibility of ensuring, through a predictable MAC level Protocol, the worst possible time for transmitting a message. In addition, many scheduling algorithms allow the predictable scheduling of real-time messages on the CAN bus.This CAN Bus is brought to life by some scheduling Algorithm, which can be classified into Asynchronous and Synchronous scheduling.Most of the schemes which are Synchronous posses an advantage over Asynchronous class because, it permits strict separation between allocation IDs.This separation gives them some extra flexibility and eases the development of Applications as discussed in [2]. FTT-CAN is one of the second class's best representatives. This ensures that periodical (time-based) and possible aperiodic (event-based) message streams are scheduled. The scheduling policy is implemented in the primary node, allocating the transmission into the elementary cycle of individual message instances. Thus, FTT-CAN's predictable schedule (i.e. periodic) and sporadically loads are provided by FTT-CAN when message streams can be strictly characterized at design time.On the other hand, ensuring that changes to periodic and average message flows are dynamical loading or handling requests requires an adequate input control and possible overload management capabilities, although supported, that are left unspecified to adapt to specific application classes at a later point.

B. Problem Formulation

The FIT-CAN protocol (for a detailed description, see [3]) divides time into an endless sequence of fixed duration elementary cycles (ECs). Each elemental cycle is further broken down into two phases (windows) in which time and event-driven traffic are temporally isolated. In the first windows, event-oriented or asynchronous messages are transmitted. The second window will transmit time or synchronous streams of messages.

At the start of each EC, all nodes are synced by a trigger message received (TM). Slaves whose aperiodic requests are pending try to transmit immediately throughout the asynchronous phase of each elementary cycle. The master does not know the requests caused by the event. At the end of the EC, the synchronous message window conveys its relative beginning moment. The reason is to decode the EC trigger message and avoid priority inversions between asynchronous messages by giving nodes sufficient CPU time. FIT-CAN reserves certain identifying bits for internal use: as valid message identifiers, only the last 6 bits can be used (out of 11), while the remainder can is used for protocol handling.This is fully explained in [2].

The protocol permits us to define a maximum total length and corresponding maximum bandwidth for the synchronous windows (LSW) for that type of traffic. Therefore, asynchronous traffic can be guaranteed minimum bandwidth. Furthermore, synchronous traffic is protected by preventing transmission starts that cannot be completed within the respective window against any interference in asynchronous requests. This is done by deleting any pending request that cannot be completed within the network controller transmission buffer interval.Therefore, at the end of the asynchronous window, a short quiet period might appear. Rate Monotonic (RM), early deadline (EDF) or possibly another algorithm that permits a projection of periodical requests can be used to determine the guidelines used by the master for scheduling periodic requests. CAN's native MAC protocol programming periodic messages. For analysis, therefore, the fixed priority timetable must be considered. For the schedulability analysis of statements in FIT-CAN, further definitions will be required.

C. Schedulability Analysis

In [3], the authors perform an analysis of schedulability of both periodic and aperiodic messages by using cycle-by-cycle scheduling analysis. This leads to an iterative schedulability test that cannot be used during run-time. The schedulability analysis can be simplified by simply getting a new and faster admission test, assuming that the worse time of the periodic message schedule does not depend on the cycle instance. To this end, we believe that the synchronous LSW(Least Synchronous Window) window with the asynchronous LAW(Least Asynchronous Window) is of constant size. The schedulability analysis of this protocol can be seen in full in [2].

D. Admission Control

In the previous section, we will take another look at the schedulability test. The proposed admission protocol can be outlined as follows.LAW^{gua} and LSW^{gua} are the guaranteed minimum width for the asynchronous and synchronous window, respectively. Additional requests may arrive at run-time for both synchronous and asynchronous streams. If accepted, those requests increase the minimum width of both synchronous and asynchronous windows respectively at LSW^{req} and LAW^{req} (Figure 7). The new requirements may or may not be safely guaranteed. LAW(Least Asynchronous Window) and LSW(Least Synchronous Window).

a) : When a periodic message request arrives, the corresponding new minimum synchronous window LSW^{req} and the available time for synchronous requests LSW^{avail} are computed. If $LSW^{req} > LSW^{avail}$, there is an overload situation on periodic messages;

b) : If a new sporadic message arrives, LAW^{avail} (2) is computed similarly. If$LAW^{req} > LAW^{avail}$, there is an overload situation in progress on periodic messages;

E. Overload management

It involves the possibility of overcharge if new corporate type streams are allowed in the system and when time parameters of guaranteed streams are dynamically changed. Therefore, our dynamic admissions control strategy needs a protocol for overload management. Our choice is to allow slave nodes to manage the possible overload (and possibly recover from it) instead of a centrally-focused and perhaps complex overload handling strategy. The master node can only signalize slaves from an existing overload situation using the message of a trigger (p/o field for regular stream overload, s/o for sporadic overload) A simple accord protocol between slave nodes requires the selection of the stream to be deleted. Again, this is being exploited through the standard CAN arbitration mechanism. Slave nodes perform overload recuperation by removing certain streams from master tables. The control messages are used to signal the master's decision to drop one of the slave streams. Our Algorithm selects the lowest priority stream among all slaves who have firm-type streams, which can restore schedulability once it is removable.

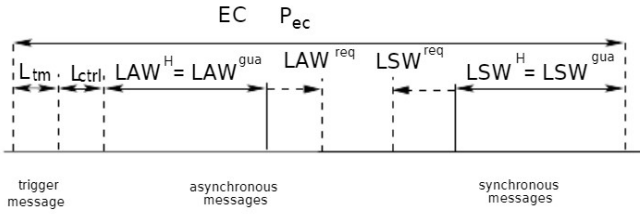

Fig. 8. Extending Asynchronous and the Synchronous windows beyond the guaranteed amount [2]

In [9] we can see here the author performs schedulability analysis of both periodic and aperiodic messages by exploiting time analysis on a cycle by cycle basis. This results in an iterative schedulability test, which can hardly be used at run-time. When the system has an overload on periodic streams, it is possible to recover from overload simply by decreasing the frequent load from the highest identification messages (which is the simplest way for lowering the system load in the case of EDF scheduling). On the other hand, if there is a sporadic overload, then the highest priority message should be calculated, missing its deadline.Since removing intermittent streams with priority lower than the computed threshold, would not affect the schedulability of the stream with priority threshold (periodic messages are scheduled with fixed priorities). The slave nodes must be informed of the intermittent streams' lowest importance, which must be considered for rejection. In this case, the trigger message that signals overload also indicates the lower bound on the priority of the messages considered for rejection (node field). If an overload is detected and signalled in the trigger message, it makes no sense to permit control messages from slave nodes in the same elementary cycle. The master node sends a dummy control message to avoid such scenarios, with a priority higher than any other possible control message from the slaves, right after the trigger. After the recovery cycle, all slaves assume that the overload situation is still in progress and prepare for contention and remove the next message by repeating the procedure.

The following example explains the sequence of actions that take place in a sample recovery procedure. The CAN network consists of four nodes,$S1, S2, S4, S6$. Node $S1$ outputs (periodic) message streams with identifiers $2, 5, 19, 23$; messages with identifiers $4, 15$ are sent by node $S2; 11, 30$by node $S4$ and $20, 21, 25$ by node $S6$. The corresponding lowest priorities are message with $id = 23$ for $S1, id = 15 for S2, id = 30$ for $S4$, and $id = 25 for S6$. If messages are labelled with the corresponding identifiers, then $Pm15 > Pm23 > Pm25 > Pm30$ (lower id means higher priority) [2]. If an overload situation is detected, all nodes try to transmit a control message that corresponds to an attempt at removing the corresponding stream:$msg_i dsubf I(m23) = 9, msg_i dsubf 2(m15) = 17, msg_i dsubf 4(m30) = 2, msg - idsubf 6(m25) = 7$. The situation is represented in Figure , where node $S4$ wins the contention with the control message (characterized by an$msg.Id.subf = 2$) corresponding to the stream with identifier 30 (lowest priority among all firm-type streams). Removal of the message with $id = 30$ is not sufficient for recovery from overload. Hence in the following elementary cycle, [2] the master keeps the overload bit set in the trigger message. The slave nodes send another set of control messages to determine the next message to be removed, with node s6 winning the contention for the message with $id = 25$, which is removed in the following cycle. After this contention round, the overload is cleared. The master signals recovery from overload by clearing the corresponding bit in the following trigger message and sending the dummy control message right after it [2].

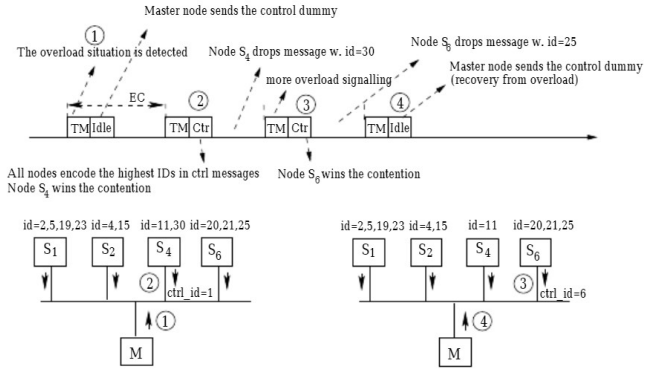

Fig. 9. System Overload Management [2]

F. Handling Overload Conditions

Some experiments have been performed in [2] to show the behaviour of admission protocol in overload conditions. The first case features an application example, where the minimum inter-arrival and deadline attributes for the newly arrived streams, are subject to negotiation with the schedulability manager in the master node. The set loosely inspires the abstract message set (represented in the Table below) in the SAE benchmark. Message stream two and streams with identifiers from $1 to 33$feature alternate possible values for the minimum inter-arrival time and deadline attributes. Our admission control protocol takes advantage of this opportunity by admitting new streams with relaxed timing constraints and missing no deadline. In the experiment (Figure 10), the starting message set includes messages with identifiers from $1 to 9$. The load of the system is progressively increased by letting message stream change their attributes (such as stream two at time 64) or new streams enter the system (message streams $12, 31, 32, 33$ at time 96 and message streams $11, 13, 14, 36$ at time 192).

id	size	mit/dline	alt. dline 1	alt. dline 2
1	135	32/5		
2	135	32/12	16/9	
3,4	135	32/12		
5	135	11/11		
6	135	20/11		
7,8,9	135	24/11		
10,12,13,14				
31,32,33,36 (*)	135	48/9	32/16	24/24

Fig. 10. Message Set Example 1 [2]

The graph in Figure 11 shows time is on the X-axis and the sum of the reciprocal of the deadlines. These are computed over all the messages in the set (an indication of the system load/criticality) is on the Y-axis. Our algorithm negotiates relaxed inter-arrival/deadline constraints or rejects new streams altogether and keeps the system schedulable at all times. Please note how new streams are admitted one at a time, with a delay of (at least) one elementary cycle, because of the need for asking admission with a control message sent to the master node. As a result, the system's load increases until the maximum guaranteed value (dotted line) is obtained. On the contrary, if the streams had been just sent in an FTT-CAN implementation without admission control, 27 deadline miss would have occurred.

id	size	mit/dline	arrival time
1	135	32/5	
2	135	16/9	
3,4	135	32/12	
5	135	11/11	
6,7,11,14	135	24/9	96
20(*)	135	20/11	
21,25,26(*)	135	24/11	
22,30(*)	135	24/11	192
27(*)	135	24/9	192

Fig. 11. Message Set Example 2 [2]

In the second experiment (Second Table and Figure 12), we tested the behaviour of the admission control scheme against the arrival of new streams with fixed timing attributes. Message streams $1 to 5$ and $20, 21, 25, 26$ were assumed to be the baseload and added new message streams at96 and 192.

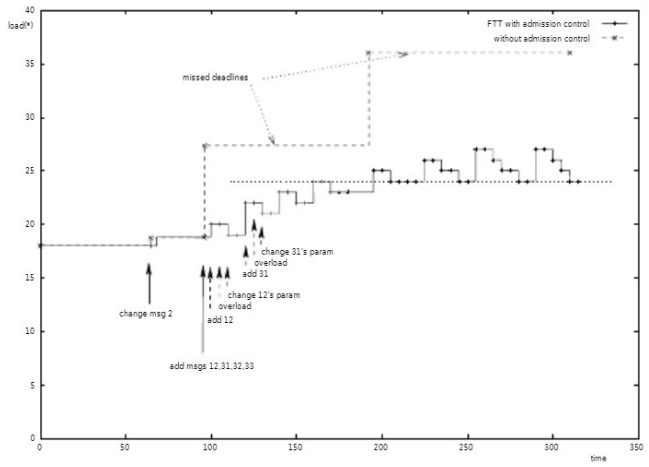

Fig. 12. Handling overloads caused by changes in the timing attributes of messages [2]

In this case, there was no possibility of adjusting the timing parameters. Therefore, it simply dropped rejected streams from the system. The graph shows how the algorithm keeps the system load under the guarantee condition (at the price of some pessimism). Had the messages been sent without an admission test, 15 deadline miss would have occurred after t=96 units.

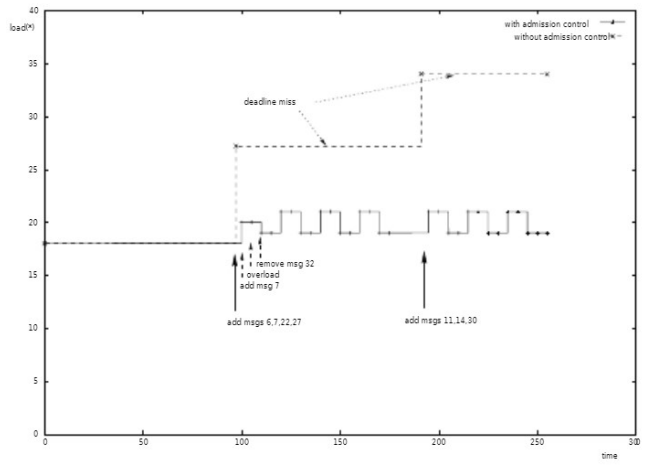

Fig. 13. Handling overloads caused by new message streams [2]

IV. Conclusion

In this paper, we were able to discuss Handling Overloads in complex systems. We defined the general Overload problem, saw some scheduling algorithms proposed by various scholars to aid us in Handling Overload problems. Furthermore, we saw the FIT-CAN Proposal being used. An admission protocol based on a worst-case message guarantee permits a faster implementation of hardware such as micro-controllers and others. Also, it allows a sure bandwidth and bounded response time for admission requests and recovery from a temporal overload of event-based messages. We obtained these results through an experiment which was thoroughly conducted in [2]

References

[1] G. C. Buttazzo, *Hard real-time computing systems: predictable scheduling algorithms and applications*. Springer Science & Business Media, 2011, vol. 24.

[2] F. Bertozzi, M. Di Natale, and L. Almeida, "Admission control and overload handling in ftt-can," in *IEEE International Workshop on Factory Communication Systems, 2004. Proceedings.*, 2004, pp. 175–184.

[3] L. Almeida, P. Pedreiras, and J. Fonseca, "The FTT-CAN protocol: why and how," *IEEE Transactions on Industrial Electronics*, vol. 49, no. 6, pp. 1189–1201, Dec. 2002. [Online]. Available: http://ieeexplore.ieee.org/document/1097741/

[4] S. Baruah and J. Haritsa, "Scheduling for overload in real-time systems," *IEEE Transactions on Computers*, vol. 46, no. 9, pp. 1034–1039, 1997.

[5] Z. Cheng, H. Zhang, Y. Tan, and Y. Lim, "Scheduling overload for real-time systems using smt solver," in *2016 17th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD)*, 2016, pp. 189–194.

[6] S. Baruah, G. Koren, B. Mishra, A. Raghunathan, L. Rosier, and D. Shasha, "On-line scheduling in the presence of overload," in *[1991] Proceedings 32nd Annual Symposium of Foundations of Computer Science*, 1991, pp. 100–110.

[7] J. Carlson, T. Lennvall, and G. Fohler, "Enhancing time triggered scheduling with value based overload handling and task migration," in *Sixth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing, 2003.*, 2003, pp. 121–128.

[8] T. Lennvall, *Handling Aperiodic Tasks and Overload in Distributed Off-line Scheduled Real-Time Systems*. Mälardalen University, 2003.

[9] F. Bertozzi, M. Di Natale, and L. Almeida, "Admission control and overload handling in ftt-can," 10 2004, pp. 175 – 184.