Distributed Real-Time Systems Survey

Lars Cederholm
Mälardalen University
School of Innovation, Design and Engineering
Box 883, Västerås, Sweden
lcm06001@student.mdh.se

Niklas Pettersson
Mälardalen University
School of Innovation, Design and Engineering
Box 883, Västerås, Sweden
npn06002@student.mdh.se

ABSTRACT
In this survey we present two commonly used analysis methods for event-triggered distributed real-time systems, the holistic analysis and the offset based analysis, and compare them to determine which is preferable to use. We also present time-triggered scheduling and how to extend a time-triggered system to work in a distributed environment.

We give an introduction to scheduling algorithms, distributed systems and communication protocols. The scheduling algorithms that are presented are intended for single processor systems but can be extended to work in distributed systems with the help of real-time communication protocols.

This survey was intended to contain the majority of algorithms and protocols associated with distributed real-time systems but comes to the conclusion that the field of distributed real-time systems is too vast and expanding to fast to summarize the whole field in one paper.

Categories and Subject Descriptors
C.2.2 [Computer-Communication Networks]: Network Protocols; C.2.4 [Computer-Communication Networks]: Distributed Systems; C.3 [Special-Purpose and Application-Based Systems]: Real-Time and Embedded systems

General Terms
Algorithms, Design, Theory

Keywords
Real-Time, Distributed Systems, Scheduling, Communication, Networks, System Design

1. INTRODUCTION
Just as the Internet have evolved the personal computer real-time systems are becoming more and more complex and using networks to build a distributed real-time system, where several nodes work together on a common goal. Take for example an airplane control system that is a combination of sensors, actuators and different automated systems all controlled from the cockpit.

This paper is a survey of distributed real-time systems and will present what is needed besides the fundamentals in both distributed and real-time systems. We will first present the fundamentals in distributed and real-time systems and then give an introduction to distributed real-time systems.

1.1 Distributed Systems
With added complexity of computer systems and widespread use of networks, there is a need for computer systems where several systems work together to achieve a common goal, a distributed system. A prime example of a distributed system is the Domain Name System (DNS) used in networks to translate domain names to IP-addresses. The four main goals of a distributed system is considered according to Tanenebaum and Van Steen [18] to be:

- Resource allocation, where several computers share resources due to cost or other reasons, for example: file servers and printers.
- Transparency, there are several transparencies that hide different unneeded information from the user, for example failure transparency hides the failure and subsequent fix of the system from the user and location transparency where it’s not needed for the user to know where the location of the system used is.
- Openness, with a focus on openness the system is easy to extend and have a uniform interface for easier interoperability.
- Scalability, designing a system so that it can easily be made bigger or smaller depending on current needs for the system.

In a distributed system it is important to have a common time reference to ease communication between different nodes, to achieve this we need a time synchronization protocol or a global time reference like internet time. Also just like a normal system with processes competing for the same resource we need a system for deciding who gets access and make sure no one else interferes with the usage of the resource, this can for example be solved with a centralized system deciding who gets to do what or by passing a token to the node who’s turn it is to have access rights.
Consistency in distributed systems details how to guarantee that all nodes see the exactly same data while containing local copies for much used data. Another related aspect is fault tolerance where we assume that errors will happen and making sure that the system will still be operational after the fault occurs. To guarantee this we first need ways of detecting the faults using different detection methods and redundancy to replace faulty parts of the system. During communication we need ways to make sure that all messages reach all nodes with atomic broadcasts and commit algorithms that guarantee that all nodes have the same data after a commit.

When communicating over networks we need to guarantee the security of our system with access control and encrypted traffic.

### 1.2 Real-Time Systems

In a general purpose system the goal is to arrive at correct results, preferably as fast as possible but if the result is delayed it is generally of no great concern. A real-time system however does not only require correct results but also predictability as to when the results are available. For example an airbag in a car should not be activated too soon or too late. If the system works as fast as possible it might inflate the airbag too soon and injure the driver by being fully inflated or in the process of deflating when the driver hits it. If the airbag is inflated too late the driver might have already hit the steering wheel and then be knocked back by the airbag.

Real-time systems does not necessarily equal fast computing, the goal is instead to achieve timely computing which means that all activities in the system have individual timing constraints. A timing constraint is usually in the form of a deadline, which is the latest point in time when an activity must finish.

Depending on the activity a deadline can be either a hard deadline or a soft deadline. If a hard deadline is missed it is typically considered a catastrophe and can result in severe consequences for the environment the task was interacting with (for example airbags and ABS-brakes that fail to function properly). A soft deadline miss does not generally have dire consequences but may cause some slight grievances (for example interruptions in phone calls). Should there be a case where a deadline will be missed it is therefore important to prioritize hard deadlines over soft deadlines.

Some of the key characteristics of a real-time system are:

- Fast and predictable handling of events, the system must be able to schedule all tasks in a fashion that no task is allowed to miss its deadline.
- Timeliness, a task in the system is not allowed to finish prematurely or too late.
- Transient load stability, if the system is overburdened and the result is some missed deadlines. The system still has to guarantee that critical task deadlines are still met.
- Handle multiple tasks, it is a waste of resources to have one system for each existing task, instead one system should be able to handle several tasks at once. To achieve this, the system must divide its resources between the tasks.
- Possibility to prioritize, some tasks are more important than others. The electric windows in a car should not be able to prevent the airbag from triggering. If there are multiple tasks ready to execute the system must allow the most important task to execute first.

### 1.3 Distributed Real-Time Systems

In a distributed system several nodes work together towards a common goal and in real-time systems we need to find a result within a deadline. With a distributed real-time system we combine these and have a system where several nodes work together with a deadline towards a common goal.

A common distributed real-time system is the x-by-wire systems, where we replace mechanical systems like the steering wheel on a car with a distributed system. This is prime example where we can save a lot of money by using a distributed system, since we can use the same system in all cars.

There are two main approaches when creating a distributed real-time system. The first is time-triggered using a time triggered communication protocol like TTP, see the MARS project in [11]. The other one is using an event-triggered scheduler with an event-triggered communication protocol like CAN. There is also the FlexRay communication bus that combines time- and event-triggered communication.

The issue we found when going from a real-time system to a distributed real-time system is how to schedule the system while also taking into account that tasks will communicate between nodes and that tasks are dependent on tasks on other nodes that must have been executed before, this will be the basis of our paper.

In section 2 we will look at the two main paradigms of scheduling, time-triggered where we create a schedule before the system is operational and event-triggered where we use a scheduling algorithm to select which task to execute during runtime. Then in section 3 we will look at communication protocols for time-, event-triggered systems and a protocol combining the two. When we have the basic schedule and communication we can look at how to create a distributed real-time schedule or how to find out if the schedule will meet the deadlines over the whole system, this will be in section 4.

### 2. SCHEDULING

In a general purpose computer system a scheduler is considered good if it is fair and gives execution time to all tasks equally. When scheduling a real-time system what is most important is that all tasks meet their deadlines and are executed so that any task depending on them, meet their deadlines as well.

#### 2.1 Event-Triggered Scheduling
Event-triggered scheduling algorithms determine what task will get access to the processor during runtime. All tasks are assigned priorities, and the task with the highest priority that is ready to execute will be allowed to. There are two approaches to event-triggered algorithms, either the tasks are assigned static priorities which do not change during runtime, or the scheduler assigns priorities to tasks dynamically.

In a distributed event-triggered system the communication protocol will generally need to handle sporadic messages being sent over the communication bus, preferably the protocol should also be event-triggered (for example Controller Area Network) which uses priorities to determine in which order queued messages should be sent. However, it is possible to use time-triggered communication and assign each node a slot where it is allowed to send messages.

### 2.1.1 Rate Monotonic

The rate monotonic (RM) scheduling algorithm was introduced by Liu and Layland in 1973 [14]. This algorithm assigns static priorities to tasks depending on their period. The task with the shortest period is given the highest priority. RM is an optimal scheduling algorithm for systems containing only independent tasks (no shared resources or semaphores) [14].

To determine if a task set is schedulable using rate monotonic we must calculate the total utilization of the set and compare it to a maximum utilization bound which can be determined using the following theorem [16].

**Theorem 1.** A set of n independent tasks scheduled by the rate-monotonic algorithm will always meet their deadlines for all task start times, if

\[
\sum_{i=1}^{n} C_i T_i \leq n(2^{1/n} - 1)
\]

When n grows large the maximum utilization bound will approach 69%, this is very pessimistic because the equation assumes the worst-case task set possible which is unlikely to be encountered. The average maximum utilization is closer to 88% [16], so failing Theorem 1 does not necessarily mean that a task set is unschedulable. Instead we can perform an exact test [16]

**Theorem 2.** For a set of independent tasks, if a task meets its first deadline $D < T$, when all the higher priority tasks are started at the same time, then it meets all its future deadlines with any other task start times

\[
W_n(t) = \sum_{j=1}^{n} C_j \left[ \frac{t}{T_j} \right]
\]

### 2.1.2 Earliest Deadline First

Up until now we have described schedulers that assign static priorities to tasks, now we will present a scheduler that assigns priorities to tasks dynamically during runtime. Earliest Deadline First [9] is the name given to a scheduling algorithm found by Horn in 1974 [7] which is based on an algorithm called Earliest Due Date (EDD) that was published in 1955 by Jackson [9]. Jacksons idea was simple, in order to minimize the maximum lateness a task could have in a system with no preemptions and that allowed a task to only run once, we schedule the tasks in such a way that the task with the shortest absolute deadline is allowed to run first and the task with the longest absolute deadline is run last.

EDF allows a system to be preemptive and have both periodic and aperiodic instances of tasks. EDF determines which task to run by assigning the highest priority to the current instance of a task with the nearest deadline. This means that instead of assigning priorities to tasks themselves, the scheduler assigns priorities to each instance of a task. The scheduler will preempt the currently running task as soon as another task is ready to execute and compare the deadlines to determine which of the instances to run.

In terms of feasibility EDF is an optimal scheduler in the sense that if there exists a feasible schedule for a task set, EDF will find it which was proved by Dertouzos in [2]. The upper bound for utilization using this type of scheduling was calculated by Liu and Layland to be 100%

**Theorem 3.** For a given set of m tasks, the deadline driven scheduling algorithm is feasible if and only if

\[
\sum_{i=1}^{m} \frac{C_i}{T_i} \leq 1
\]

In comparison to rate monotonic we get more utilization and in the general case less overhead because there will be less preemption. In the event of a missed deadline, RM can guarantee that it is not a critical task that misses the deadline but with EDF is hard to predict which task that will miss the deadline since the priorities change from instance to instance. As such RM is preferred over EDF in critical systems because it is more predictable and easier to control the execution by shortening the periods of tasks.

### 2.2 Time-Triggered Scheduling

Time triggered scheduling[10] is done before the start of the system (offline scheduling) and is able to schedule more complex system at the cost of flexibility. The scheduler chooses what task to run based on a table created before the start of the system. The analysis of a time-triggered system is very trivial since the proof is just showing that the system can be scheduled, this is also true during runtime since the only thing the scheduler does is reading the table to find out what task should execute.

To find a schedule for a set of tasks the first step is create precedence graphs[4], this will give us an order to schedule them in. We create one graph for all tasks with the same period, the graphs also have all the useful information of a task like execution time, deadline and release time that is needed to schedule them. The following figure is an example of precedence graphs for a single processor system with 3 tasks. What the graphs tells us is that we need to schedule A1 before A2 and that we can schedule A3 at any time.
Figure 1: Precedence graphs for a single processor system with three tasks.

The second step is to take all precedence graphs and create a joint graph from those. We create a joint graph with a period of the least common multiple (LCM) of all the precedence graphs, so each graph is added \( \frac{\text{LCM}}{\text{Period}} \) times into the joint graph, with release times and deadline changed to match the subsequent instance of the task. The next figure is a joint graph created from the precedence graphs we create earlier, as we can see task A3 is added twice since the LCM of 100 and 50 is 100 and A3 has a period of 50, we also changed the deadline and release time of the second instance of A3 to 100 and 50.

Figure 2: Joint graph with a period of 100 and 4 task instances

3. COMMUNICATION

We are going to have a look at time- and event-triggered communication protocol and the FlexRay protocol which combines both techniques. Time triggered communication divides the time on the bus giving all nodes a specific time to send messages. While event triggered instead uses priorities to decide what node will be allowed to send the message. Time triggered avoids collisions on the bus while event triggered instead must handle the collisions.

3.1 Controller Area Network

Controller Area Network (CAN)[15] was developed for the automotive industry as a communications protocol to use in their distributed real-time systems and is now used in a variety of industries.

The CAN architecture allow nodes in a network to see all messages sent on the bus, this makes CAN a useful protocol to solve the atomic broadcasting problem in distributed systems. The architecture of CAN has a maximum speed of 1mbit/s for a bus size smaller then 50m this can not compare to a modern Ethernet network. On the other hand CAN is a very fault tolerant protocol and provides many techniques to detect and handle potential errors.

The original CAN specification is event-triggered and uses priorities to determine what message will be broadcasted on the bus, by sending the ID of the message bit-wise on the bus, using logical AND to find the highest priority message in queue. This guaranties us that a message with the highest priority \( m \) will be sent as soon as the bus is available for a new message. Which gives \( m \) a response time of 130µs (maximum transmission time of a CAN message) plus transmission time of \( m \) [13].

We can use a response time analysis similar to tasks to determine the response time of a message in CAN, this gives CAN predictability so that we can take communication into account when scheduling a distributed real-time system.

Using equation 1 we can calculate the worst case response time of a CAN message, we must first find the response time
of all messages with higher priority, then we must also take into account that there might be a lower priority message that is blocking the bus.

\[ w_i = B_i \sum_{j \in \mathcal{B}_P(i)} \left[ \frac{w_i + J_i + \tau_{b,c}}{T_j} \right] C_j \]  \hspace{1cm} (1)

Then we take the result from equation 1 and calculate the actual worst case response time of the message using equation 2.

\[ R_i = J_i + w_i + C_i \]  \hspace{1cm} (2)

### 3.1.1 Time Triggered CAN

TTCAN [6] is built on top of the existing CAN protocol using a common time reference to schedule messages at a certain time slot, giving CAN a time triggered schedule. The common time reference is maintained by sending reference messages between all nodes and synchronizing their local time.

An alternative for time triggered CAN is found in [3] where we get a time triggered representation of the CAN messages.

### 3.2 Time Triggered Protocol

Time Triggered Protocol (TTP) [12] is a communication protocol made for time triggered systems. TTP communication uses TDMA\(^1\) to create a slot for each node where it sends messages over the bus, when we guarantee that all nodes send at least one message we can use these messages to synchronize time so that all nodes have a common time reference.

To make a TTP system more fault-tolerant we can add redundant nodes for each node in the cluster and arrange them into Fault-Tolerant Units (FTU), this will prevent single points of failure in the system. Another tool to accomplish fault-tolerance is to use two replicated buses for communication. This further reduces the chance of faulty messages being processed.

There are currently two implementations of TTP these are TTP/C and TTP/A. TTP/C provides the full fault-tolerance offered by a TTP protocol and is for example suitable in fly-by-wire. TTP/A is designed for process control systems and interfacing with sensors and actuators. [5]

### 3.3 FlexRay

FlexRay[17] is a commercial communications protocol developed by FlexRay Consortium for the automotive industry. Like TTP FlexRay uses two busses for communication and TDMA to schedule the message cycles, unlike TTP instead of only having a section for time triggered messages FlexRay divides the TDMA cycle into four sections and uses one of them to schedule event triggered messages, one for time triggered messages and the other two for synchronization and control messages for the system.

Like the combined online-offline schedulers, FlexRay brings a lot of flexibility to the system, since we gain the benefits of both time and event triggered communication protocols. Going from TTP to FlexRay does loose some of the safety functionality in TTP, which have to be implemented or be omitted from the system.

### 4. DISTRIBUTED REAL-TIME SCHEDULING ANALYSIS

Scheduling a distributed real-time system requires one more dimension, scheduling the communication between nodes, so that if we have a Task C on one node sending a message to Task A on another node that is required to execute before Task B on the same node. To find out if Task B will meet its deadline we must take into account not only the response time of Task A but also the message \( C \rightarrow A \) and Task C.

![Timeline for communication between nodes.](image)

We have looked at event triggered and time triggered and now we will have investigate how to find out if the schedules will be correct over a network as well.

### 4.1 Event Triggered

In the previous sections we have looked at scheduling for single node systems, we will present different ways to find a schedule for a distributed system over several nodes. While there is no special extension for EDF or RM to enable scheduling in distributed systems, we will look at two different methods of analyzing a distributed system scheduled with EDF or RM to find out if the current setup will be schedulable over the full system.

#### 4.1.1 Holistic Analysis

The first method is holistic [19] schedulability analysis in which we schedule each node and the communication bus as individual systems and then analyze the system as a whole to determine if the schedule is feasible.

To find an equation to calculate the response time of any task in the system, we first need to find all tasks preceding it on the local node, then find if any of those are preceded by a message from another node. So if we consider a system with two nodes (A, B) with A having two tasks (A1, A2) and B having one task (B1) with these dependencies: B1 requires to send a message to A1 before it can execute and A2 requires A1 to be executed before it can run i.e. \( B1 \rightarrow A1 \rightarrow A2 \). So the response time of A2 will not only be effected by A1 as in a single node system but now requires B1 and the message between B1 and A1 to be added to the response time equation as well.

Find the set of tasks and messages that precedes \( \tau_i \) (see the figure above for an example). For each task in the chain we calculate the local response time using equations 3, 4 and for each message we use an equivalent equation for the

---

\(^1\)Time Division Multiple Access (TDMA) [5] splits the communication bus into time slots where each node is allowed a number of slots for communication.
5. SUMMARY AND CONCLUSIONS

We have looked at the most commonly used scheduling techniques and communication protocols used in distributed real-time systems.

When we started this paper we were looking at doing a survey of everything related to distributed real-time systems, but found that to be a vast and growing field so we narrowed our field to communication protocols and scheduling algorithms used in distributed real-time systems.

During our research for this paper we have been looking at schedulers that are created to work better with the FlexRay communications bus or in a similar fashion. Examples of scheduling algorithms using both time- and event-triggered tasks can be seen in [8] and [1].

We have taken a real interest in distributed real-time systems and are looking forward to seeing the advances in scheduling and communication for distributed real-time systems and maybe one day join the research field our selves.

6. REFERENCES

