DISTRIBUTED REAL-TIME SYSTEM

Dhruvkumar D Patel
Department of Computer Science
University at Albany, SUNY
Albany, NY, USA
dpatel23@albany.edu

***Abstract*—A real-time system is a computer system in which the correctness of the system behavior depends not only on the logical results of the computations but also on the time when the results are produced. A real-time computer system is distributed, it consists of a set of (computer) nodes interconnected by a real-time communication network. This paper is about different aspects of DRTS system, why it is needed, how it works. It discusses the characteristics of Distributed Real-Time Systems and how this system differs from the general operating system. It also discusses the importance of clock synchronization, scheduling of real-time tasks, network management and challenges faced in such a system.**

***Index Terms*—Synchronization, Predictability, Deadlines, Fault Tolerance, Concurrency, Scheduling, Redundancy , Carrier sense with multiple access/collision arbitration (CSMA/CA), Time division multi-access (TDMA), Time Trigger Protocol (TTP).**

I. Introduction

A distributed real-time system (DRTS) consists of autonomous computing nodes connected by a real-time network. Each node in the system has a specific goal that has to be done within a specific time limit. Together they achieve a common goal. Distributed real-time systems are needed for a variety of reasons. First and foremost, for an application, if processing parts are apart then real-time computing is naturally distributed. The second reason for deploying distributed systems is to achieve fault tolerance which is a fundamental requirement from any real-time system. Also, balancing the load across the nodes of a DRTS improves performance.

The computation performed at each node should meet the timing constraints of the tasks. Moreover, the network must provide data processing with bounded message delays. Many real-time applications are distributed in nature where tasks performed by a node at such a system are affected by tasks run at other nodes within the network. Tasks must communicate and synchronize in a DRTS over the real-time networks. A modern car is supplied with such a DRTS where sensing nodes for temperature, speed, water, oil levels, etc. are connected over a real-time network.

Each node of the network is responsible for certain dedicated functions and needs to communicate with other nodes over the real-time network

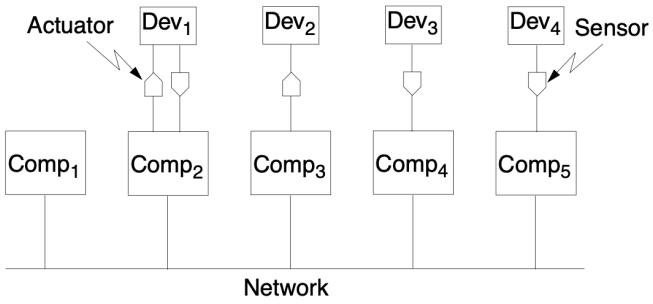


Fig. 1. Data flows in above figure, from sensors and control panels to processors between processors from processors to actuators and displays

to perform its functions and produce the required output

II. Characteristics of Distributed Real-Time Systems

- **Meeting Deadlines**: Deadlines of all hard real-time tasks must be met to prevent loss of lives and property. They are time critical. The failure to meet time constraints can lead to degradation of the service or to catastrophe. A deadline impose on a process can be:

  Hard deadline: It has to be met strictly otherwise it would be catastrophe. Also the peak load of the performance must be predictable and should not violate the predefined deadlines.

  Soft deadline : processes can be finished after its deadline, but not desired. a degraded operation in a rarely occurring peak load can be tolerated
- **Predictability**: It is one of the most important properties of any DRTS. We have predict the worst-case response times and whether the deadlines of all tasks will be met prior to the execution of tasks in the system. We also have to determine worst case communication delays,on the interconnection network and overheads due to operating system (interrupt handling, process management, context switch, etc.). Predictability commonly involves proving theoretically that all deadlines will be met in a hard real-time system.
- **Safety and Reliability**: In Real-time systems operate and control environments such as nuclear plants and process control, where safety and reliability are of utmost concern. Unsafe and unreliable systems are prone to errors which may result in loss and damage to lives and property. Example, [6] In 1999, A disaster investigation board reports that NASA's Mars Climate Orbiter burned up in the Martian atmosphere because the software calculated the force the thrusters needed to exert in pound of force and a separate piece of software took in the data assuming it was in the metric unit: newton.
- **Fault Tolerance**:A fault occur when a component of software or hardware of a computer system fails and the system does not behave according to its specification anymore. A Fault may result in a failure , which may cause loss of lives and property in a real-time system.

A fault may be permanent causing the system to fail permanently or transient, which may disappear after some time. Fault tolerance is the ability of a system to resist faults and continue functioning correctly in the presence of fault.

- **Concurrency**: The real-time computer generally exhibits parallel execution of events in their physical environment.The real-time computer must be able to cope with this parallel operation using concurrent system software capabilities or distributed hardware. hard real-time system.

III. A Distributed Real-Time Operating System Versus A General Operating System

A distributed real-time operating system has most of the functionalities of a general operating system but it also has to provide this operation in a predictable and timely manner.

- **Interrupt Handling**: External devices usually generate interrupts and interrupt service routines (ISRs) handle the routine.An ISR run in high priority mode in a general operating system by preempting the running task. But, in real time system, interrupting a task with a hard deadline can cause missing of the deadlines. So, the ISR should be short and fast because prohibiting of more interrupts far a long time is not desired in a real- time system. A general approach is to have the ISR as another task of high priority to be scheduled by the real-time operating system.
- **Synchronization and Communication**:Functions for task creation, destruction, synchronization, and communication, should be done in a timely manner.
- **Scheduling:**: It is the process of assigning a ready task to the processor. Completing deadlines of a task important concern for real-time scheduler. The context switching time should be kept as minimal as possible which requires efficient ready queue handling procedures. Facilities for periodic, aperiodic, and sporadic tasks scheduling and management should be provided.
- **Memory Management**:The memory space should be managed efficiently in distributed real-time operating system. In order to be predictable, dynamic memory allocation is not preferred in a real-time system to prevent unwanted waiting and uncertainties. At compile time, all the needed memory is allocated, so waiting time foe memory is considerably reduced. However, we still need memory manager to be able to reuse the space when it is freed. For instance, a network driver will take data from network and fill it in already allocated buffer then deliver it to a higher-level protocol. In the end message will be delivered to the application, which will use data in the

buffer and should return buffer to free memory space for future use. If such recycling method to free space is not provided, then system will run out of memory space in a short time.

IV. Time and Event Triggered Distributed Systems

A trigger is an event that initiates a response in a real-time system. The system must respond to external events at predefined instants in time-triggered systems as stated before. For example, measuring the temperature of a liquid at every 5 s is performed in a time- triggered system. Scheduling of the events can be performed offline as the time and duration of the execution are known beforehand with insignificant overheads at run time. Testing and fault tolerance can be performed more simply in a time-triggered DRTS as the characteristics of the failing hardware or software module is known beforehand to replace it with an exact replica. Time-triggered systems are suitable for periodic hard real-time tasks; however, they have very little flexibility when there is a change in the task set and the load experienced by the system.

A time-triggered architecture requires synchronous communication necessitating the use of a common clock. Having such a facility is difficult to realize in distributed real-time systems. However, it is possible to synchronize the freely running clocks of a DRTS at regular intervals using suitable algorithms. The time-triggered architecture (TTA) can be used as a template to implement distributed real-time systems. Each node in this network is allocated a fixed time slot to broadcast any message it has, hence providing a guarantee on message delivery. TTA architecture is implemented in many real-time applications such as automotive applications.

In contrast, the system must react to external events that arrive sporadically in an event- triggered system as we briefly reviewed before. Online, priority-driven scheduling of tasks is needed in such a system. In general, an event-triggered system is more flexible and adaptive to changing system characteristics. However, these systems suffer from significant overheads at run time due to more complex scheduling algorithms than the time-triggered case. Communication in a distributed real-time system can also be classified as event-triggered where communication is started when a send command is received, or time-triggered during which messages are sent periodically. In general, the event-triggered approach is convenient in DRTSs in which unpredictable events from the external world occur frequently. On the other hand, the time-triggered method is suitable for deterministic DRTSs which have known task sets with characteristics of tasks known be- forehand. In the common case, a DRTS will have time-triggered and event-triggered components.

V. Distributed Real-Time Operating Systems and The Middleware

A real-time operating system needs to manage the resources that exist at a node and also has to provide a convenient user interface where applicable. A distributed real-time operating system (DRTOS) has to perform functions like, scheduling of tasks, inter-task communication and synchronization, task management, handling of interrupts, management of input/output, locally at a node and also should provide synchronization and communication among tasks residing at different nodes of the system.

Timeliness is the major requirement from a DRTS which generally consists of a small real-time kernel, commonly called micro-kernel, replicated at each node of the distributed system that also performs the above-listed functions. Moreover, a distributed real-time kernel has to coordinate the relay of messages over the real-time network. The messages must be delivered reliably and in time which necessitates support for fast delivery by the network hardware and protocol. A subset of higher-level tasks required from the real-time operating system can also be realized by an individual node, that is, the functions needed may be partitioned across the nodes. The higher layers of the operating system work along with the micro-kernel per- forming low-level functions close to hardware and the unreplicated components involved in higher-level functions.

A. *The Middleware*

The middleware layer resides between the operating system and the application. the functions provided by it are typically extensions of the functions provided by the operating system as well as management of the network. There is a need for this layer, instead of embedding the provided functions within the operating system since different applications may need different middleware activities but they all need the basic operating functions. Yet, these procedures are general and will be required by many applications to have a general framework on top of the operating system.

Three main middleware functions needed in a distributed real-time system are: Clock synchronization, Scheduling of real-time tasks and Network management.

B. *Clock Synchronization*

Agreement on a common time frame is crucial in a DRTS, Failure to do so may result in erroneous states of the nodes which can lead to the collapse of the whole system. For in- stance, missing deadlines of tasks distributed over the network due to different clock val- ues at nodes becomes inevitable. In a DRTS, the usual requirement is to have the clocks synchronized to an external real clock or a reference clock within allowed margins. The universal coordinated time (UTC) keeps time based on the atomic clock with corrections needed because of the vari-

ations in the rotation of the earth. The UTC clock value can be accessed via satellites which propagate it constantly. The global positioning system (GPS) [7] allows determining the geographical position of an entity. A GPS satellite broadcasts its position with a timestamp allowing accurate reception of time by receivers. Each computing element of a DRTS is equipped with a hardware clock that is fed by a crystal. Although the frequency of every crystal is stable, frequencies of crystals at nodes may have different frequencies which cause the clock values to drift away from one an- other in time. Keeping the time at a node is performed by having a timer that decrements its value at a specific rate per second and when this value reaches zero, the timer interrupt is generated. The timer interrupt handler then increments the clock value of the node to reflect the current time. The clock synchronization algorithms attempt to correct the drifting clock values of nodes in the DRTS. Two cases can be considered for this purpose; a DRTS with one or more nodes that keep their time with GPS and all other nodes syn- chronized to these nodes, or none of the nodes have GPS receivers and they are to be synchronized to each other within allowable limits.

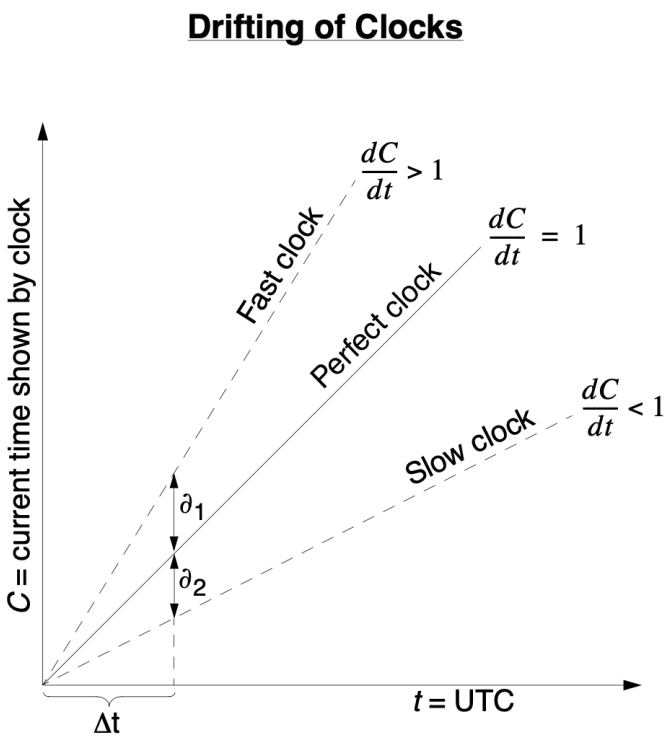
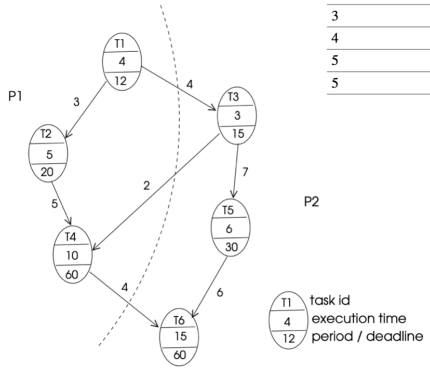


Fig. 2. d2: drift of first clock after Δt, d2: drift of second clock after Δt, d1 + d2: skew between clocks after Δt. In an ideal case: The clock shows UTC: C = t. i.e dC/dt = 1. In reality: The clock shows UTC: C = t. i.e dC/dt = 1 ± p. Where, p is the maximum drift rate, and should be specified by the manufacturer. Two processors with similar clocks could be apart with, S=2Δt. If we have to guarantee a skew less than Smax, the clocks have to be synchronized at ani nterval:= Δt¡Smax/2p

C. *Distributed Scheduling*

An important problem to be handled by the distributed real-time OS is to schedule tasks among the nodes of the distributed system so each task meets its deadline and load is fairly distributed. This so-called distributed scheduling and load balancing isn't a trivial task.

τ_i	C_i	D_i
1	4	12
2	5	20
3	3	15
4	10	60
5	6	30
5	15	60



Task graph example

Static distributed scheduling refers to the schedul-

ing of the tasks before runtime. One way of achieving this is to have a task graph and partition it using some heuristic. Let us consider the hard real-time task set given in the above Table with tasks 1,...,6 having computation times Ci and deadlines Di. Let's further assume the communication among these tasks as shown in the task graph of the above Figure, where an arrow pointing from a task i to j shows i must finish before j can begin running. Scheduling of these six periodic tasks to two processors P1 and P2 can be achieved by partitioning the task graph as shown with a dashed line. While scheduling these tasks, communication between two tasks running in the same processor is considered to have zero cost although inter-processor communication is significant and needs to be taken into account. Possible

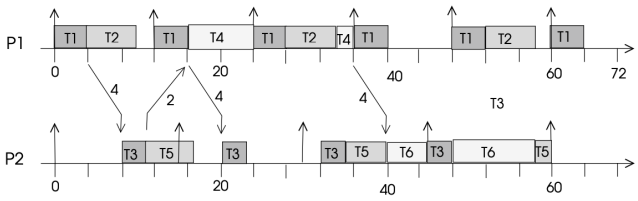


Fig. 3. Scheduling of P1 P2 in two processors.

scheduling of this set to the 2 processors using a Gantt chart which shows task executions against time is depicted in the above Figure. Note that 3 cannot run after 4 units of time of 1 finishing since it has to wait for the message from this task to start executing. Similarly, 4 waits for the message of 3 and 6 for 4. Note that this scheduling may repeat every 60-time units. The distributed scheduling of tasks is NP-hard and heuristics are commonly used to find feasible schedules.

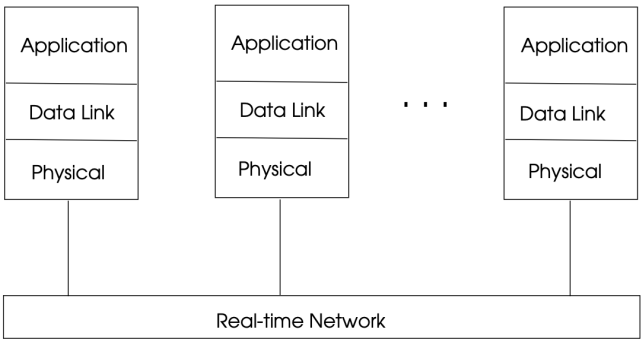
Dynamic Load Balancing

It is possible in a distributed real-time system that some nodes are heavily loaded because of the unpredictable irregularity of task execution, while other nodes may have very few tasks to run. In such a scenario, the migration of tasks from heavily loaded to a lightly loaded node could be considered. Aim of dynamic load balancing methods is to even the processor loads across different nodes. this enables dynamically adding of tasks to the system and allow different types of tasks such as asynchronous and sporadic tasks to be scheduled.

These methods can be centralized or fully distributed or somewhere between these two approaches [8]. In the Centralized Approach, Scheduling decisions are taken by a central node, which might become a bottleneck with a large number of nodes. Also, the state information to be kept at the central node may be large. Moreover, the central node is a single point of failure, deficiency of which can cause the system to halt. However, for systems with a small number of nodes this approach may be practical and simple to apply. In fully distributed, Decision is made at each node using sender-initiated or receiver-initiated approaches. A node that finds its load as heavy searches for a light node in the sender-initiated method while a light node searches for a heavy node to receive tasks in the receiver-initiated method.

D. Network Management

A real-time communication network should provide a certain level of quality of service (QoS) to the real-time application. The main QoS requirements from a real-time network are bound on the network delay, bound on the loss rate experienced by the network, and a small blocking probability. In real-time communication requires the arrival of messages should be on time and reliable. Messages have deadlines within which they must be delivered to the destination application over the real-time network. Commonly, provisions for prioritized message passing is provided in a real-time network such that high-priority messages are delivered before the lower ones within the network. Real-time application do not need Open Systems Interconnect (OSI) seven layer model for communication. Since many real-time networks are developed for a certain application and use a local network. Hence, the presentation and network layers of the OSI model can be discarded in Real Time Networks(RTN). Also, short messages are commonly needed to transfer messages in a RTN, assembly facilities and making the fragmentation is unnecessary. A small RTN, three layer model is shown in figure where data link layer is directly accessible. It is more common to have another layer typically at transport level to provide a network-independent interface to the application. Generally, real-time application commonly consists of processing sites that are within close proximity to each other, hence, the real-time network is typically a local network that has bus, ring, tree, mesh structure.



Collapsed OSI model used in a real-time network

There exist three main types of traffic in real-time networks [9] :

- **Constant Bit Rate (CBR) Traffic** : The network is utilized at a constant rate, for example, when sensors generate data periodically. Commonly, hard real-time tasks produce CBR traffic using fixed length messages.
- **Variable Bit Rate (VBR) Traffic**: Data transmission over the network varies at different times.
- **Sporadic Traffic**: Network data transfer occurs at bursts typically followed by long periods of no transfers. This type of traffic is a special case of the VBR traffic with the main difference that there is a certain time interval between two sporadic data transfers. A typical example is an alarm occurring in a real-time network.

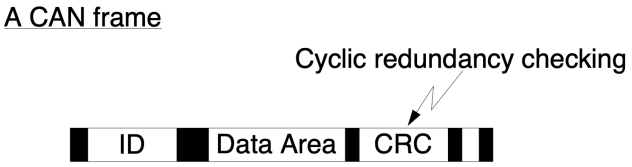
*1) **Medium Access Protocols (Real Time Network)***

The arbitration of the shared communication medium is done by MAC sub-layer. Three main methods of real-time arbitration at MAC layer are :

- **Carrier sense with multiple access/collision arbitration (CSMA/CA):** Instead of CSMA/CD, CSMA/CA is used in RTN, where message priorities are imposed.Whenever a collision occurs, the highest priority message is transmitted which provides the determinism needed.
- **Time division multi-access (TDMA)** A predetermined duration of network usage called a frame, is allocated to each station in the network. This type of protocol is essential for distributed real-time systems because of its deterministic characteristics. Each station is aware of the time it has right to access to the network. TDMA is commonly used in shared bus distributed real-time systems.
- **Clustering** groups data points that are similar according to a given metric. Small datasets can be clustered by manually labeling every instance, but for larger datasets, this might be infeasible, which proves that it needs to automatically label the data.
- **Token-based Communication** Possession of a token provides the right to use the network for transferring a message. A station that has no data to send transfers the token to the next node in the latter in which a token continuously rotates in the network (Token-Ring). A token also circulates the network in the Token Bus archiecture, but the sequence of stations that it can traverse is determined by the soft- ware. Each station has a predecessor and a successor assigned to it. Fiber distributed data transfer has dual counter-rotating tokens. With token-ring maximum bounds on message delay can be established by token hold time (the longest time a node may hold the token) and token rotation time (the longest time needed for a full rotation of the token). Fault tolerance can be a problem for token-based com- munication, if one node fails, whole traffic is disrupted.

*2) **CAN (Control Area Network) protocol [10]***

CAN is a (CSMA/CA) protocol, which provides a higher degree of flexibility in the case of irregular flow. It is widely used in automation systems such as, ships, avionics equipment, and medical equipment. Modern automobiles have several tens of electronic con- trol units (ECUs) to engine, navigation, and control brake. CAN uses two wires for the multi access communication bus resulting in major reduction in the size of wires. The station that want to transmit monitors the bus and starts sending its message when the bus is idle. CAN communication is based on frame. The identifier (ID) field has for two purposes, to distinguish between

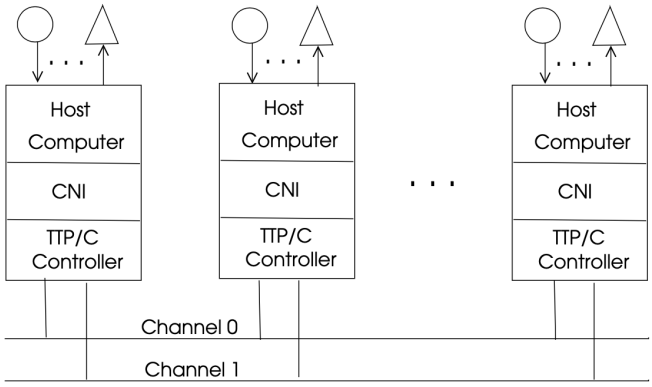


different frames and to assign relative priorities to the frames. The special bus arbitration of the CAN protocol provides efficient handling of the collisions. A CAN controller is attached to each processor in the system. It ensures that, The highest priority frame waiting to be transmitted from the respective processor is entering the arbitration for the bus and The arbitration procedure performed in cooperation by the controllers, guarantees access to the message with highest priority. CAN uses four message types:

- The Data Frame: This is the most common message type to convey data.
- The Remote Frame: This frame is used to solicit transmission data from a remote node.
- A node detecting an error in a message sends this frame causing other nodes to send the error frame. The sender of the message then retransmits the message.
- The Overload Frame: This frame is used by a busy node to delay transfer of messages.

*3) **Time-Triggered Protocol [11]***

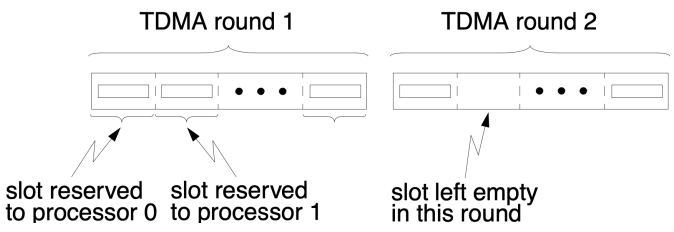
The time-triggered protocol (TTP) is a family of fault-tolerant protocols with small over- head designed by Kopetz and Grunsteidl. TTP has two versions, TTP/C for automotive class C for hard real-time applications and TTP/A for automotive class A for low-cost soft real-time applications. Master/worker-based bus arbitration is used in TTP/A, in which a master node supervises bus access. TTP employs TDMA to access the network and is based on time-triggered distributed real-time architecture in which each event is defined in a common time base.All clock of the node are synchronized with high precision and messages which are mostly periodic are transferred at predetermined time instants. TTP/C used for hard real-time systems



TTP/C network

has a communication network interface (CNI) between the communication controller and the host computer in a network node. The TTP/ C communication controller (CC) is the actual interface between the network and the TTP/C station. Each node is assigned a fixed time slot to access the network. The TTP/C network connects nodes by two replicated channels named Channel 0 and Channel 1. Clock synchronization between the stations is provided in each TDMA round.

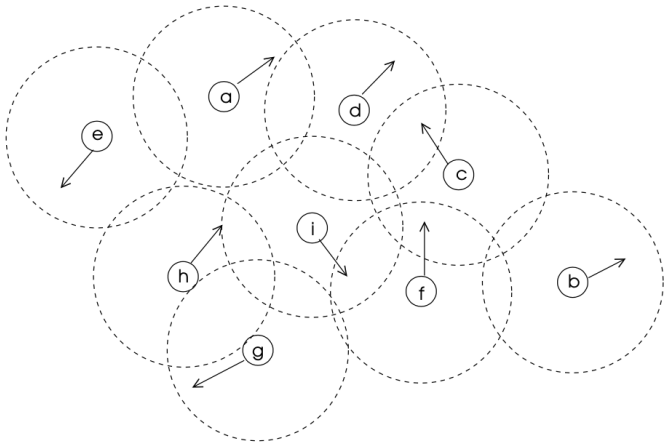
The total channel (bus) capacity is statically divided into several slots. Each slot is as- signed to a node (processor). With a system of N processors, the sequence of N slots is called a TDMA round. One frame is sent by one processor in a TDMA round and this frame is placed into the slot which was allocated to that processor. if the processor sends no frame, then the empty slot will be sent in that round. The duration of one TDMA round is called TDMA period. TDMA practically means a static partitioning of access time to the bus so, Each processor knows in advance when and for how long it is allowed to access the bus. Collisions are avoided as processors know when they have guaranteed exclusive access to the bus. Message passing delay is bounded: a message is split into a certain number of frames that are transmitted in successive slots (one per TDMA round). Not all slots have to be of identical length. However, slot length corresponding to a given node is identical for all rounds. Slot length is determined by the designer, depending on the particularities of the processes running on each node (considering, for example, number and length of messages generated) TDMA



provide high degree of predictability and it well suited to safety critical applications. But it has low level of flexibility(irregular flow) and can lead to poor utilisation of the available bus bandwidth (e.g. empty slots).

VI. A Mobile Wireless Sensor Network[12] Example of Distributed Real-Time Systems

There are many methods to perform ensembling, such as: A wireless sensor network (WSN) consists



of small autonomous nodes with sensors, lim- ited computational capabilities, and wireless communication using radio waves. A mobile WSN (MWSN) is a sensor network with mobile nodes. MWSNs have many applications such as environment monitoring, rescue operations, military surveillance, and healthcare applications. Generall hybrid topologies are preferred, but it may have other topologies (bus, mesh, cluster).

Routing in MWSN uses multi-hop message transfers and has to deal with correct delivery of the messages over a dynamic topology. MWSN above had A to I nodes. A dashed circle around a node

shows its transmission range and we can see each node is within transmission range of at least one other node. Each node is communicating with other node. Nodes move in the direction shown by arrows pointed outside and nodes may be disconnected from the network at the next time instant, for example, node b is such a candidate. Providing a connected network at all times is a fundamental problem in an MWSN. The MWSN nodes can be attached to animal, people and vehicles for monitoring purposes. Monitoring can be performed periodically using time-triggered method or it can be event based where data is generated when an event occurs. One can have a mix- ture of these modes as in the general distributed real-time case.

VII. Challenges in Distributed Real-Time Systems

Designing DRTS faces many challenges.In DRTS, the general task performed by operat- ing system are enhanced in two directions: the operating system is now in real-time and distributed. User envisioned a distributed operating system as a single operating system. More complexity is added to the operating system as we add more functionality to it. Still, we need to provide the basic distributed operating system functions such as intertask communications and synchronization over the network in real time in a predeter- mined manner which is a significant task. The middleware is situated between the DR- TOS and the distributed real-time application to provide a common set of services to the application.So, many diverse real-time applications use it. The synchronization of clocks is a typical middleware service in DRTS.

Implementation of fault tolerance [13] is crucial in a distribution system, whether system is real time or non-real time. In the real time case, the cost of a fault can be greater both in terms of human life and other cost. So, detecting fault such as software malfunction or network failures and recovering from these faults are fundamental function to be performed in DRTS. Common sources of faults in a DRTS are network failures, the hard- ware failure of nodes, distributed coordination problems and transmission delays that exceed the bound.

In the single node real-time system, we need to have the tasks meet their deadlines as the most basic and fundamental task. In the distributed case, realization of this function is more difficult and hence a challenge in a DRTS. Scheduling tasks with known deadlines and execution times can be performed by partitioning the task graph to the computing nodes available. Implementation of graph partitioning brings in the additional requirement that the tasks should meet their deadlines. Yet, another problem encountered in a DRTS is the testing which becomes a challenge on its own to handle issues such as fault tolerance and environment simulation, compared to a non-real-time distributed system.

VIII. Conclusion

DRTS consists of real-time processing nodes connected by a real-time network. The basic requirement from the real-time network is the timely delivery of messages. The operating system should provide the interface to the network with the provision of all real-time functionalities of a single node real-time system, and also provide mechanisms for distributed scheduling of tasks. Meeting deadlines and Predictability is the most important property of a real-time system. Distributed static scheduling refers to assignment of tasks to the nodes of the distributed system prior to run time so that each task meets its deadline. For aperiodic and sporadic tasks that are acti- vated at runtime, various dynamic scheduling policies are commonly employed. A fundamental requirement from a distributed real-time system middleware is the provision of clock synchronization as all of the tasks need to synchronize at certain points in time over a global time base. The network time protocol is used as a standard for time synchronization in the Internet. Protocols like CAN and TDMA are used in many automotive application and provides high predictability and the potential for safety critical applications.

References

[1] http://www.it.uom.gr/teaching/distrubutedSite/dsIdaLiu/lecture/lect11-12.frm

[2] Erciyes K. (2019) The Hardware. In: Distributed Real-Time Systems. Computer Communications and Networks. Springer, Cham.

[3] https://www.seas.upenn.edu/~lee/10cis541/lecs/kopetz_chap_1and2-ilx2.

[4] Hermann Kopetz, Real-Time Systems Design Principles for Distributed Embedded Applications

[5] Kopetz H (1997) Real-time systems-design principles for distributed embedded ap- plications. Kluwer Academic Publishers.

[6] https://www.wired.com/2010/11/1110mars-climate-observer-report

[7] Zogg J-M (2002) GPS basics. Technical Report GPS-X-02007, UBlox, Mar 2002 .

[8] Erciyes K, Ozkasap O, Aktas N (1989) A semi-distributed load balancing model for parallel real-time systems. Informatics 19(1):97–109 (Special Issue: Parallel and Dis- tributed Real-Time Systems).

[9] National Programme on Technology Enhanced Learning. Real-time systems course. Govt. of India.

[10] CAN bus. http://www.can-cia.org/can/protocol/

[11] Kopetz H, Grunsteidl G (1993) TTP—a time-triggered protocol for fault-tolerant real-time systems. In: IEEE CS 23rd international symposium on fault-tolerant comput- ing, FTCS-23, Aug 1993, pp 524–533.

[12] Hayes T, Ali FH (2016) Mobile wireless sensor networks: applications and routing protocols. Handbook of research on next generation mobile communications systems. IGI Global. ISBN 9781466687325, pp 256–292.

[13] Lee PA, Anderson T (1990) Fault tolerance: principles and practice, 2nd edn. Springer.