Integrated Resource Management and Scheduling with Multi-Resource Constraints [*]

Sourav Ghosh[†] Jeffery Hansen[‡] Ragunathan (Raj) Rajkumar[§] John Lehoczky[¶]

Abstract

Dynamic real-time systems such as phased-array radars must manage multiple resources, satisfy energy constraints and make frequent on-line scheduling decisions. These systems are hard to manage because task and system requirements change rapidly (e.g. in radar systems, the targets/tasks in the sky are moving continuously) and must satisfy a multitude of constraints. Their highly dynamic nature and stringent time constraints lead to complex cross-layer interactions in these systems. Therefore, the design of such systems has long been a conservative and/or unpredictable mixture of pre-computed schedules, pessimistic resource allocations, cautious energy usage and operator intuition. In this paper, we present an integrated approach that simultaneously maximizes overall system utility, performs task scheduling and satisfies multi-resource constraints. Using a phased-array radar system, we show that our approach can reconfigure settings of 100 *tracks at every* 0.7 sec *in real-time, and performs within 0.1% of the achievable optimal solution.*

1. Introduction

In this paper, we consider an integrated framework for QoS optimization and scheduling for dynamic real-time systems having multiple resource constraints. A good example of such systems is a phased-array radar system. Unlike traditional radar systems, a phased-array radar can electronically steer energy beams in a desired direction. To effectively track a target, it must be scanned at a frequency that depends upon its distance, velocity and acceleration. The larger the distance to the target, the larger the energy needed to track it. Once a beam transmission is initiated, it cannot be preempted. The goal of the radar system is to utilize its finite energy and time resources to maximize the quality of the target tracks, to search for new targets, and to confirm new tracks.

Radar system software must make two basic decisions. First, it must decide the fraction of its resources (energy and time) to be allocated to each target. For a given resource allocation, it must schedule the radar antenna(s) to transmit the beams and receive the return echoes, both in a non-preemptive fashion. Since the targets move continually, sometimes evasively, the resource allocation and scheduling decisions must be made frequently and in real-time. Due to the multi-dimensional nature of radar resource allocation [12], the problem of optimally determining the resource allocations to maximize total system utility is NP-hard [7]. Given the high computational complexity and the need for real-time decision-making, phased-array radar systems have traditionally employed a combination of conservative heuristics, table lookups, and/or operator intuition to make the required resource allocation and scheduling decisions.

In our approach, we develop an integrated framework that performs near-optimal resource allocation and schedules the radar front-ends dynamically in real-time. We show that such decisions can be made nearly optimally, while maintaining schedulability and satisfying the resource constraints of the system. We concentrate primarily on the radar resources as these are generally more scarce compared with system computing resources. Unlike traditional radar systems, we use two layered components: a QoS optimization component that is concerned with determining task resource allocation, and a scheduling component that is concerned with determining the scheduling of the radar tracking tasks. In short, our radar resource management scheme deals with two primary concerns:

Selection of Operating Points: We use the QoS-based Resource Allocation Model (Q-RAM)[8][11] as the basis of our QoS optimization framework. In Q-RAM, the optimiza-

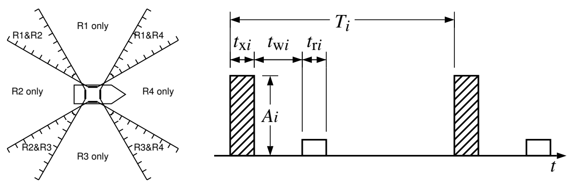
[*] This work was supported by the DoD Multidisciplinary University Research Initiative (MURI) program administered by the Office of Naval Research under Grant N00014-01-1-0576

[†] Carnegie Mellon University, Department of Electrical and Computer Engineering, sourav@cs.cmu.edu

[‡] Carnegie Mellon University, Institute for Complex Engineered Systems, hansen@cmu.edu

[§] Carnegie Mellon University, Department of Electrical and Computer Engineering, raj@ece.cmu.edu

[¶] Carnegie Mellon University, Department of Statistics. jpl@stat.cmu.edu



(a) Radar Angle Ranges

(b) Radar Dwell

Figure 1. Radar System Model

tion goal is to maximize the total utility derived by all tasks in the system while meeting resource constraints. In general, this optimization problem is NP-hard, thus a Q-RAM solution will not be optimal; however, the solutions are close to optimal over a wide range of conditions. Furthermore, the Q-RAM solutions can be found very efficiently. We presented a *scalable* Q-RAM technique for a radar tracking scenario in [2]. In principle, this approach can work with a very large number of targets, target configurations, and cooperative radar systems. It also allows for changing the weights associated with each of the target types. We assume the results presented in [2].

Ensuring Schedulability: In general, Q-RAM assumes the use of simple inequality resource constraints. For example, the total instantaneous usage of any single resource must be less than or equal to 100%. In a phased-array radar system, the situation is much more complicated. It is possible that a Q-RAM-generated resource allocation may not be schedulable, jitter constraints can be violated even when all resource utilizations are less than 100%, and furthermore complex constraints on the use of radar energy over time must be satisfied. Consequently, the QoS allocator (Q-RAM) and the task scheduler need to be tightly integrated. In this paper, we present a solution that integrates the Q-RAM framework with radar schedulability test that can successfully handle even highly dynamic tracking scenarios.

Many recent studies have focused on phased-array radar systems. For example, Kuo et al. in [6] proposed a reservation-based approach for real-time radar scheduling. This approach allows the system to guarantee the performance requirement when the schedulability condition holds. However, Kuo et al. do not consider the energy constraint. Shih et al. in [13, 12] use a template-based scheduling algorithm in which a set of templates is constructed offline, and tasks are fit into the templates at run-time. The templates consider both the timing and power constraints. They consider interleaving dwells to allow beam transmissions (or receptions) on one target to be interleaved with beam transmissions (or receptions) on another. The templates' space requirements limit the number that can be used, and "service classes" designed offline determine how QoS operating points are assigned to discrete sets of task configurations across an expected operating range. Goddard et. al. [3] addressed the real-time scheduling of radar tracking algorithms using a data-flow model.

The rest of this paper is organized as follows. Section 2 presents our model of the radar system, its associated resources and constraints. Section 3 describes our radar dwell interleaving scheme. Section 4 presents our integrated resource management model. In Section 5, we present an evaluation of our experimental results. Finally, in Section 6, we summarize our concluding remarks and provide a brief description of our future work.

2. Radar Model

We assume the same radar model as used in [2]. It consists of a single ship with 4 radar antennas oriented at 90° to each other as shown in Figure 1(a). We also assume that each antenna is capable of tracking targets over a $120°$ arc. This means that there are regions of the sky that are capable of being tracked by only one radar antenna, as well as regions that can be tracked by two antennas. The antennas are assumed to share a large pool of processors used for tracking and signal processing algorithms, and a common power source to supply energy to the antennas.

2.1. Radar Task Model

Each radar task consists of a front-end "dwell" sub-task at the antenna and a back-end signal processing sub-task at the processors. A dwell is defined as a single instance of transmitting a radar beam, waiting for the signal to be reflected, and receiving the reply. A typical radar dwell is illustrated in Figure 1(b). It is characterized in terms of a transmit power A_i, a transmit pulse width $t_{\mathrm{x}i}$, a wait time $t_{\mathrm{w}i}$ and a receive time $t_{\mathrm{r}i}$. Generally, $t_{\mathrm{x}i} = t_{\mathrm{r}i}$ and the wait time is based on the round-trip time of the radar signal (e.g., about 1 *ms* for a target 100 miles away). In practice, the radar transmission is actually composed of a high frequency series of pulses. Higher the number of pulses, typically higher is the tracking precision. Also, while the radar may dissipate some power while receiving, this power is much smaller than the transmit power. For purposes of simplicity, we assume that the receive power is negligible compared to the transmit power.

A radar task is periodic with a strict jitter constraint. For example, for a task with period T_i, the start of each dwell must be exactly[1] T_i milliseconds from the start of the previ-

1 In practice, if two successive dwells are not separated exactly by T_i,

ous dwell. In this paper, we make the seemingly conservative choice of using only harmonic periods for radar tasks, since the harmonics satisfy the stringent periodic jitter constraints in a pipe-line scheduling of radar antenna and CPU (a pin-wheel scheduling problem [9]).

Since the radar is unused during the waiting period of a dwell, a typical utilization of the radar with N periodic tasks (dwells) is given by:

$$U_r = \sum_{i=1}^{N} \frac{t_{\mathrm{x}i} + t_{\mathrm{r}i}}{T_i}. \qquad (1)$$

In order to appropriately track a target, the dwell needs to have a sufficient number of pulses (target illumination time) with a sufficient amount of power (A_i) on the pulses to traverse across air, illuminate the target and return back after reflection. Based on the received pulses, an appropriate signal processing algorithm must be used in order to properly *estimate* the target range, velocity, acceleration, type, etc. Since a target can maneuver to avoid being tracked, the estimates are valid only for the duration of the illumination time. Based on these data, the time-instant of the next dwell for the task must be determined. Therefore, the tracking task needs to be repeated periodically with a smaller period providing better estimates. In the absence of any jitter, the tracking period is equal to the temporal distance between two consecutive dwells. For a large temporal distance, the estimated error can be too high such that the dwell will miss the target. On the other hand, a small temporal distance will require higher resource utilization. Hence, the radar needs to track the targets with higher importance with better tracking precision than the ones with lower importance [2].

2.2. Radar QoS Model

In [2], we developed a Q-RAM model for the radar tracking problem. The model defines the problem in terms of a per-task QoS space (Q_i), an operational space (Φ_i), an environmental space (E_i) and a shared resource space (R). Each of these spaces is defined by a set of dimensions. *QoS dimensions* represent the aspects of tracking quality that are of direct interest to the user such as the tracking error on a target. *Operational dimensions* represent the aspects of the task over which the radar system has control. For example, in the case of a radar tracking task the operational dimensions include the dwell period, the dwell time and the transmit power. While the operational dimensions affect the tracking quality, they are not of direct interest to the user. *Environmental dimensions* are the aspects of the task that we do not control, but that affect the mapping between QoS and resource requirements. Examples of environmental dimensions for a tracking task include the distance to the target, the type of the target and its speed.

For each point in the QoS space Q_i of each task i, there is a utility value u_i representing the benefit to tasks of operating at that quality level. The QoS for a task is a function of the selected point in the operational space Φ_i (also called the set-point) and the current point in the environmental space. The objective of the QoS optimizer is to choose a point in Φ_i for each task i so as to maximize the sum of the utility values over all tasks. The details on the scalable optimization algorithm for radar tracks can be found in [2].

In this paper, we will consider two primary types of tasks: tracking tasks and search tasks. For a tracking task, the tracking error is its only QoS dimension, with task utility increasing as its tracking error decreases (or, more precisely, as the reciprocal of tracking error increases). For search tasks, the number of beams within the angular region of the radar determines the quality of searching. Hence, the number of beams is the QoS dimension for the search task, with a higher number of beams yielding a higher utility. The expressions for tracking error and utilities are given in Appendix A.

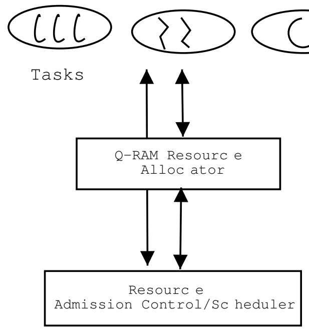


Figure 2. Q-RAM & Scheduler

2.3. QoS and Scheduling

As mentioned earlier, a given resource allocation obtained from Q-RAM may not result in a schedulable task set. If the tasks are not schedulable, Q-RAM must reduce the resource utilization bounds in order to obtain a schedulable task set. The interaction between Q-RAM and the scheduler admission control is shown in Figure 2. We discuss this in more detail in Section 4 for phased-array radars.

The Q-RAM optimization must be performed at regular intervals (known as the reconfiguration rate) as a background process in a dynamic scenario where the task set is not fixed and tasks are continuously arriving and departing from the system [4]. It accepts all newly arrived tasks, per-

lower tracking quality will result. If the jitter is higher than a (small) threshold, an entire track may be lost.

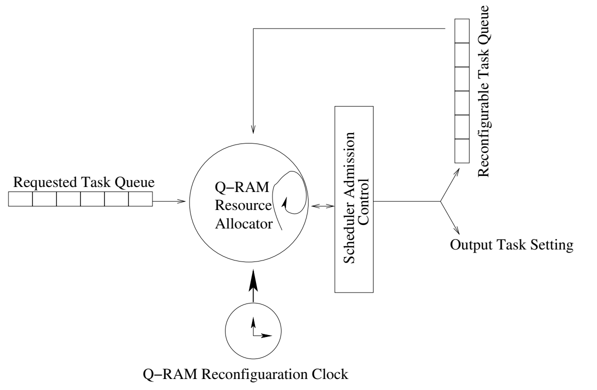


Figure 3. Dynamic Q-RAM Optimization

forms optimization along with the existing tasks, and produces the resource allocation settings for all of them as illustrated by Figure 3. Therefore, the scalability of Q-RAM operation determines how often we can optimize the system in a dynamic environment.

2.4. Radar Power Constraints

In addition to timing constraints, radars also have power constraints. Violating a power constraint can lead to overheating and even permanent damage to the radar. A radar can have both long-term and short-term constraints. For example, there may be a long-term constraint of operating below an average power of 1 kW, and a (less stringent) short-term constraint of operating below an exponentially weighted moving average power of $1.25\ kW$ in any $200\ ms$ window.

2.4.1. Long-Term Power Utilization Bound If $\overline{P}_{\max}$ is the maximum sustained long-term power dissipation for a radar, then we define the long-term power utilization for a set of N tasks to be

$$U_P = \frac{1}{\overline{P}_{\max}} \sum_{i=1}^{N} A_i \frac{t_{\mathrm{x}i}}{T_i}. \tag{2}$$

That is, the long-term power is given by the fraction of time each task is transmitting, multiplied by the transmit power for that task. Dividing by $\overline{P}_{\max}$ gives a utilization value which cannot exceed 1. To handle long-term constraints in Q-RAM, we simply treat power as a resource, and denote the amount of that resource consumed by task i by $\frac{1}{\overline{P}_{\max}} A_i \frac{t_{\mathrm{x}i}}{T_i}$.

2.4.2. Short-Term Power Utilization Bound We will now derive a short-term power utilization bound. Short-term power requirements are defined in terms of an exponential sliding window [1] with time constant τ. With an exponential sliding window, pulses transmitted more recently have a larger impact on the average power value than less recently transmitted pulses. Also, the rate that the average power decreases is proportional to the average power value meaning that immediately after transmitting a pulse, we have a relatively high but steadily decreasing cooling rate. The use of an exponentially weighted average over a sliding window has two benefits: it is memoryless, and it closely models thermal cooling which is often the motivation for the constraint.

In order to define the short-term average power, we first define an instantaneous power dissipation as $p(t)$. This function is 0 when the radar is not transmitting and A_i while pulse i is being transmitted. We then define the average power at time t for a time constant τ by

$$\overline{P}_\tau(t) = \frac{1}{\tau} \int_{-\infty}^{t} p(x) e^{(x-t)/\tau} dx. \tag{3}$$

Figure 4(a) shows an example of the average power value for a set of pulses along with the exponential sliding window at time t_0. The shaded bars represent the transmitted radar energy, and the dotted line represents the sliding window at time t_0. The short-term average power constraint is considered to be satisfied if (3) never exceeds some bound $\overline{P}_{\tau\max}$. This bound is called the *power threshold* over a *look-back period* τ. Alternatively, the expression $\overline{E}_{th} = \overline{P}_{\tau\max}\tau$ is defined as the *Energy threshold* of the system.

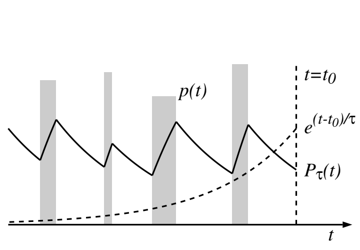
Now, we define a timing parameter called the *cool-down time* $t_{\mathrm{c}i}$ that precedes a dwell of each task i. The cool-down time for a task is the time required for $\overline{P}_\tau(t)$ to fall from $\overline{P}_{\tau\max}$ to a value just low enough that at the end of the transmit phase of a dwell, $\overline{P}_\tau(t)$ will be restored to $\overline{P}_{\tau\max}$ as shown in Figure 4(b). The cool-down time is a function of the transmit time $t_{\mathrm{x}i}$ and the average power A_i of a dwell, the time constant τ and the short-term average power constraint $\overline{P}_{\tau\max}$. This factor allows the power constraints to be converted into simple timing constraints.

We will now derive the cool-down time $t_{\mathrm{c}i}$ for a task i. We will assume that for this task $A_i \geq \overline{P}_{\tau\max}$, else $t_{\mathrm{c}} = 0$. Let $\overline{P}_{\mathrm{s}}$ be the average power at the beginning of the cool-down period, $\overline{P}_{\mathrm{in}}$ be the average power at the end of the cool-down period, and $\overline{P}_{\mathrm{out}}$ be the average power at the end of the transmit. We want $\overline{P}_{\mathrm{s}} = \overline{P}_{\mathrm{out}} = \overline{P}_{\max}$. We can express $\overline{P}_{\mathrm{in}}$ in terms of $\overline{P}_{\mathrm{s}}$ as

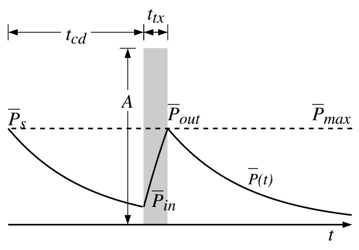
$$\overline{P}_{\mathrm{in}} = \overline{P}_{\mathrm{s}} e^{-t_{\mathrm{c}i}/\tau}, \tag{4}$$

and $\overline{P}_{\mathrm{out}}$ in terms of $\overline{P}_{\mathrm{in}}$ as

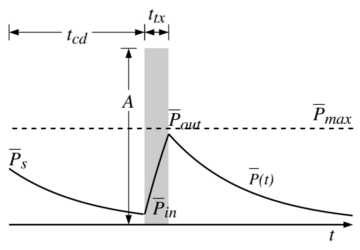
$$\overline{P}_{\mathrm{out}} = \overline{P}_{\mathrm{in}} e^{-t_{\mathrm{x}}/\tau} + A_i(1 - e^{-t_{\mathrm{x}i}/\tau}). \tag{5}$$



(a) Average Power Window



(b) Cool-down time



(c) Non-Optimal Initial Average Power

Figure 4. Transformation of Short-term Power Constraints to Time Domain

Substituting $\overline{P}_{\tau\max}$ for $\overline{P}_{\text{out}}$ into (5) and solving for $\overline{P}_{\text{in}}$, we obtain:

$$\overline{P}_{\text{in}} = \frac{\overline{P}_{\max} - A_i(1 - e^{-t_{\text{x}i}/\tau})}{e^{-t_{\text{x}i}/\tau}}. \tag{6}$$

We can now substitute $\overline{P}_{\tau\max}$ for $\overline{P}_{\text{S}}$ into (4) and set the forward and backward definitions (4) and (6) for $\overline{P}_{\text{in}}$ to be equal and solve for $t_{\text{c}i}$ to yield the expression for the cool-down time,

$$t_{\text{c}i} = -\tau \ln \frac{\overline{P}_{\tau\max} - A_i(1 - e^{-t_{\text{x}i}/\tau})}{\overline{P}_{\tau\max} e^{-t_{\text{x}i}/\tau}}. \tag{7}$$

We now present the following theorem:

Theorem 1. *For any set of N periodic radar tasks that do not violate the short-term average power constraint and satisfy $A_i \geq \overline{P}_{\tau\max}, 1 \leq i \leq N$, the total short-term average power utilization,*

$$U_\tau = \sum_{i=1}^{N} \frac{t_{\text{c}i} + t_{\text{x}i}}{T_i}, \tag{8}$$

must satisfy $U_\tau \leq 1$.

Proof. Assume that we have a set of tasks for which $U_\tau = 1$. From (7), it can be shown that any decrease in $\overline{P}_{\tau\max}$ will cause $t_{\text{c}i}$ to increase, and cause U_τ to exceed 1. If we can show that when $U_\tau = 1$, the optimal schedule must include a point where the average power $P_\tau(t)$ equals $\overline{P}_{\tau\max}$, then this implies that the theorem must hold. Now, assume that we have a schedule $\mathcal{S}$ where tasks are scheduled such that each dwell transmit period $t_{\text{x}i}$ is preceded by an idle time of $t_{\text{c}i}$ with the cool-down time for each dwell beginning exactly at the end of the previous dwell's transmit. Now let $\overline{P}_{\text{S}}$ be the average power at the beginning of the cool-down period preceding a dwell transmit. It can be shown from (4) and (5) that if $\overline{P}_{\text{S}} < \overline{P}_{\tau\max}$, then the $\overline{P}_{\text{out}}$ for that dwell must satisfy $\overline{P}_{\text{S}} < \overline{P}_{\text{out}} < \overline{P}_{\max}$ as shown in Figure 4(c) due to the fact that the cooling rate is proportional to the current average power. This implies that at the end of each transmit period for each successive dwell, the average power will increase until it converges to $\overline{P}_{\tau\max}$. This means that in the steady-state, the average power will be $\overline{P}_{\tau\max}$ at the end of the transmit period for every dwell. The schedule $\mathcal{S}$ must be optimal since moving a dwell any sooner would result in an increase in $\overline{P}_{\text{in}}$ for that dwell and increase $\overline{P}_{\text{out}}$ above $\overline{P}_{\tau\max}$. Moving a dwell any later would trade-off the efficient cooling immediately after the transmit when average power is at $\overline{P}_{\tau\max}$ for less efficient cooling before the transmit resulting in a violation after the next dwell. This shows that the schedule $\mathcal{S}$ must be optimal and that it must have a point where average power is equal to $\overline{P}_{\tau\max}$. □

Based on (8), we model the short-term average power constraint in the Q-RAM optimization framework by treating power as a pseudo-resource with a maximum value of 1 and treating each radar task as if it consumes $\frac{t_{\text{c}i}+t_{\text{x}i}}{T_i}$ units of that resource, with $t_{\text{c}i}$ computed using (7). Hence, we refer to the expression in (8) as the *cool-down utilization U_c* of the system.

It is interesting to note that if we take the limit as $\tau \to \infty$ in (7), it can be shown that:

$$t_{\text{c}i} = (\frac{A_i}{\overline{P}_{\tau\max}} - 1)t_{\text{x}i}. \tag{9}$$

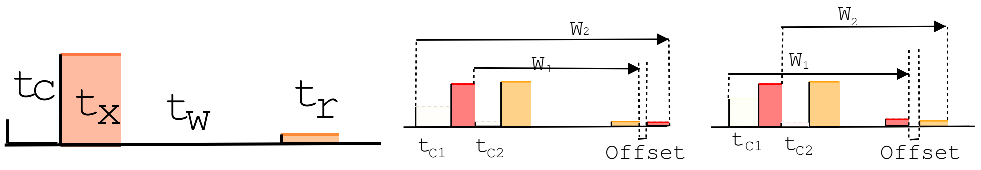
If we then substitute the above into (8), we obtain

$$U_{\tau=\infty} = \frac{1}{\overline{P}_{\tau\max}} \sum_{i=1}^{N} A_i \frac{t_{\text{x}i}}{T_i}. \tag{10}$$

We see that this equation has the same form as the long-term power utilization given in (2).

3. Scheduling Considerations

Our model of each radar dwell task consists of 4 phases: cool-down time(t_{c}), transmission time (t_{x}), waiting time



(a) Dwell with Cool-down Time (b) Proper Nesting (c) Improper Nesting

Figure 5. Interleaving of radar dwells

(t_{W}), and receiving time (t_{r}), as shown in Figure 5(a). The durations t_{X} and t_{r} are non-preemptive, since a radar can only perform a single transmission or a single reception at a time. However, t_{C} of one task can be overlapped with t_{r} or t_{W} of another task, since the radar can cool down during the waiting and the receiving period.

Allowing the entire duration of a dwell (from transmission start to reception end) to be a non-preemptive job wastes resources and decreases the schedulability of the system [13]. Task dwells can, however, be interleaved to improve schedulability. Dwells can be interleaved in two ways: (1) properly nested interleaving and (2) improperly nested interleaving. An optimal construction of interleaved schedules using a branch-and-bound method has been described in [12] and [13]. In this paper, we focus on fast and inexpensive construction of dwell inter-leavings in the presence of dynamically changing task-sets. The inter-leavings that we construct may not necessarily be optimal in the sense of [13], but they will be schedulable.

3.1. Proper Nesting of Dwells

Two dwells are said to be properly nested if one dwell fits inside the waiting time (t_{W}) of another. Figure 5(b) demonstrates this scheme where dwell W_2 fits in the waiting time of dwell W_1. The necessary condition for this interleaving is given by Equation (11).

$$t_{\mathrm{W}w1} \geq (t_{\mathrm{C}w2} + t_{\mathrm{X}w2} + t_{\mathrm{W}w2} + t_{\mathrm{r}w2}). \qquad (11)$$

We define a phase offset for a proper interleaving as given in (12). For instance, we can schedule the cool-down time of the dwell W_2 right after the transmission time of W_1. Thus, the value of the phase offset determines how tightly two nested tasks fit together. Our aim is to minimize this offset,

$$o_p = t_{\mathrm{W}w1} - (t_{\mathrm{C}w2} + t_{\mathrm{X}w2} + t_{\mathrm{W}w2} + t_{\mathrm{r}w2}). \qquad (12)$$

The proper nesting procedure is detailed in Algorithm 1 in Appendix B. The core of the scheme deals with fitting a dwell of the smallest size into a dwell with the smallest feasible waiting time.

3.2. Improper Nesting of Dwells

Two dwells are said to be improperly nested when one dwell only partially overlaps with another (e.g. as illustrated in Figure 5(c)). Suppose that task $W1$ is improperly interleaved with task $W2$, where $W1$ starts first. Task $W1$ is called the *leading task* and task $W2$ is called the *trailing task*. Based on the phasing illustrated in Figure 5(c), the necessary conditions for the interleaving to occur are given in (13) and (14),

$$\begin{aligned} t_{\mathrm{W}w1} &\geq t_{\mathrm{C}w2} + t_{\mathrm{X}w2}, & (13)\\ t_{\mathrm{C}w2} + t_{\mathrm{X}w2} + t_{\mathrm{W}w2} &\geq t_{\mathrm{W}w1} + t_{\mathrm{r}w1}. & (14)\end{aligned}$$

We define a phase offset for this case given by

$$o_i = t_{\mathrm{C}w2} + t_{\mathrm{X}w2} + t_{\mathrm{W}w2} - (t_{\mathrm{W}w1} + t_{\mathrm{r}w1}). \qquad (15)$$

Our improper nesting scheme is given in Algorithm 2 in Appendix B. It starts with the task with the largest waiting time (t_{W}), and attempts to interleave it with the task with the largest possible t_{W} smaller than that of the original task based on the conditions stated in (13) and (14). The algorithm repeats the process until it reaches the task with the smallest t_{W} that can no longer be interleaved, or all tasks are interleaved to form a single virtual task.

3.3. Dwell Scheduler

The responsibilities of the radar dwell scheduler are as follows.

- Obtain period and dwell-time information ($t_{\mathrm{C}}, t_{\mathrm{X}}, t_{\mathrm{W}}, t_{\mathrm{r}}$) from Q-RAM for each task.
- Interleave tasks with the same period using proper and/or improper nesting to create a fewer number of virtual tasks.

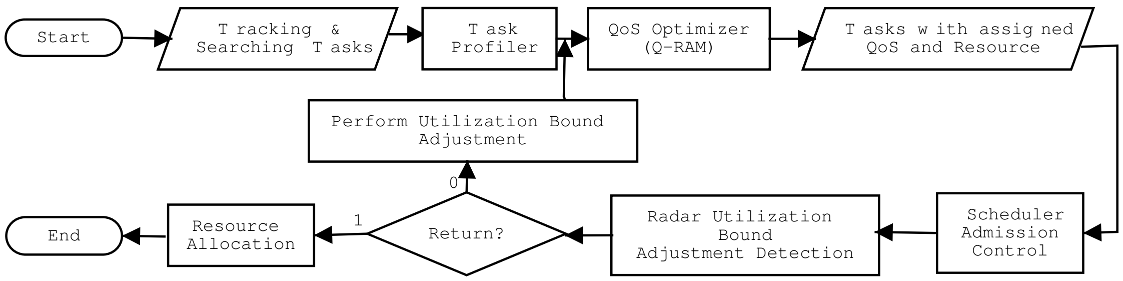

Start
T racking & Searching T asks
T ask Profiler
QoS Optimizer (Q-RAM)
T asks w ith assig ned QoS and Resource
Perform Utilization Bound Adjustment
0
1
Return?
End
Resource Allocation
Radar Utilization Bound Adjustment Detection
Scheduler Admission Control

Figure 6. Resource Management Model of Radar Tracking System

- Perform a non-preemptive harmonic schedulability test for the virtual tasks.

Next, we describe our schedulability test.

3.3.1. Schedulability Test As mentioned earlier, in order to satisfy jitter requirements, only relatively harmonic periods are used for the dwells[2]. We define the following terms:

- N_i = Number of tasks with a period T_i
- C_{i_j} = Total run-time of the j^{th} task under period T_i
- N_T = Total number of periods
- $T_i > T_j, \forall i < j$

The response time t_{R_i} of the tasks for a given period T_i is given by:

$$t_{R_i} = \underbrace{\sum_{j\geq 1}^{j=i-1} \lceil \frac{T_i}{T_j} \rceil \sum_{k=1}^{N_j} C_{j_k}}_{\text{run-time of higher priority tasks}} + \underbrace{\sum_{k=1}^{N_i} C_{i_k}}_{\text{run-time of tasks with period } T_i} + \underbrace{B_i}_{\text{Blocking term}} \quad (16)$$

The blocking term B_i is defined as the maximum run-time C_{m_j} among tasks with lower priority, as defined in (17). As already mentioned, each radar task (virtual or otherwise) is considered non-preemptive under the schedulability test, and

$$B_i = \underbrace{max(C_{m_n}), \forall i < m \leq N_T, 1 \leq n \leq N_m}_{\text{Maximum task size among all tasks of lower priority}} . \quad (17)$$

For a task-set to be schedulable, we must satisfy

$$t_{R_i} \leq T_i, \forall i \in N_T. \quad (18)$$

Recall that using nesting, we combine multiple tasks into a smaller number of virtual dwell tasks within each period. The run time of a task is given by $C_{j_k} = t_{\mathrm{c}_{j_k}} + t_{\mathrm{x}_{j_k}} + t_{\mathrm{w}_{j_k}} + t_{\mathrm{r}_{j_k}}$, where the parameters $t_{\mathrm{c}_{j_k}}$ etc. may be virtual parameters if the dwells are nested.

4. Iterative Resource Allocation

The entire resource allocation process is detailed in Figure 6 in the form of a flow-chart. The tracking and searching tasks obtained from the environment are fed into the QoS optimizer, `Q-RAM`. `Q-RAM` determines the resource settings of the tasks and attempts to maximize global utility. The `Scheduler Admission Control` interleaves the radar tasks as densely as possible, and then runs the efficient schedulability test. If the task set is not schedulable, the `Radar Utilization Limit Adjustment` function repeats the Q-RAM and schedulability tests by varying the Q-RAM radar utilization bound using a binary search. That is, if an interleaved task set is not schedulable, the utilization bounds on the radar antennas are reduced, and vice versa. This iteration is stopped when it reaches a schedulable task-set, and the utility values obtained in two successive iterations differ only by a small value (such as 0.1%).

5. Experimental Results

This section is divided into 3 parts. We first describe our assumed radar configuration in detail. Next, we present a set of experiments to study the impact of using only harmonic periods in QoS optimization. We compare two different harmonic period distributions against a wide choice of periods. Finally, we run the entire resource allocation process as described in Figure 6 using various interleaving schemes for radar scheduling and compare their performances.

5.1. Experimental Configuration

We assume a radar model as pictured in Figure 1(a). Radar tasks are classified into tracking tasks, high priority search tasks, and low-priority search tasks. The ranges that

2 As we show in the next section, our model of radar system does not show significant degradation in the accrued utility due to the restricting use of harmonic periods

Parameter	Type	Range
Distance	All	$[30, 400]$ km
Acceleration	All	$[0.001g, 6g]$
Noise	All	$[kTB_w, 10^3 kTB_w]$ [3]
speed	1 (helicopter)	[60,160] km/hr
	2 (fighter-jet)	[160,960] km/hr
	3 (missile)	$[800, 3200]$ km/hr
Angle	All	$[0°, 360°]$

Table 1. Environmental Dimensions

Tasks	Number of beams	Period (ms)	Power (kw)	Tx Time (ms)
Hi-Priority Search	$[15, 60]$	800	5.0	0.5
Tracking	1	$[100, 1600]$	$[0.001, 16.0]$	$[0.02, 50.0]$
Lo-Priority Search	$[10, 30]$	1600	3.0	0.25

Table 2. Period and Power Distribution

we use for periods, dwell time, dwell power and the number of dwells among them are given in Table 2.

As mentioned earlier, tracking error is assumed to be the only QoS dimension for each tracking task, and the number of beams is the QoS dimension for search tasks. The tracking error, in turn, is assumed to be a function of environmental dimensions (target distance r_i, target speed v_i, target acceleration a_i, target noise n_i) and operational dimensions (dwell period T_i, number of pulses η_i in dwell transmit time C_i, pulse width w, dwell transmit pulse power A_i, tracking filter algorithm) [2]. For a search task, each beam corresponds to a single dwell with certain values of T_i, η_i and A_i.

The assumed ranges of the environmental dimensions for targets are shown in Table 1. The ranges of various operational dimensions for all types of tasks are shown in Table 2. We assume three types of tracking filters, namely Kalman, $\alpha\beta\gamma$ and least-squares, to account for computational resources. Their estimated run-times have been extrapolated to equivalent run-times of a $300MHz$ processor as shown in Table 3 in Appendix A. This is because the radar computing system is assumed to be a distributed system consisting of a large (128) number of $300MHz$ processors. We also assume that the overhead for a search task is assumed to be the same as that of the Kalman filter.

We use a $2.0GHz$ Pentium IV with $256MB$ of memory for all our experiments.

5.2. The Effect of Harmonic Periods

Our first experiment studies the effect of using only harmonic periods on the total utility obtained by the QoS optimization. We consider only tracking tasks for simplicity. We study the impact of harmonic periods across a wide range of system configurations. Specifically, we vary the amount of the two primary resources in the system, namely energy limits and available time. We achieve this by varying two factors:

• **Energy threshold ($\overline{E}_{th}$):** Lowering $\overline{E}_{th}$ increases the cool-down time for each quality set-point of a radar task, and therefore the increases the cool-down utilization requirement. $\overline{E}_{th}$ is defined at the end of Section 2.4.2.

• **Transmission-time tracking constant (`Tx-factor`)**: This factor directly influences the requirement of transmission time. A higher `Tx-factor` raises the transmission time for a particular quality set-point. This, in turn, increases both the radar utilization as well as the cool-down utilization requirements for a given quality of any task. `Tx-factor` is defined in Appendix A.

We use the settings from Tables 1, 2 and 3, randomly generate 512 tasks (tracks) and develop their profiles. We also vary `Tx-factor` from 1 to 16 in a geometric fashion, and $\overline{E}_{th}$ from $20J$ to $670J$ keeping the look-back period τ constant at $200ms$[6]. We then perform QoS optimization under three distributions of available periods between the range $[100, 1600]ms$:

• **A:** Arithmetic distribution in steps of $10ms$ (to approximate a continuous range of available periods to choose from),

• **G2:** Geometric distribution with a common ratio of 2 (100, 200, 400, $\cdots$),

• **G4:** Geometric distribution with a common ratio of 4 (100, 400, 1600).

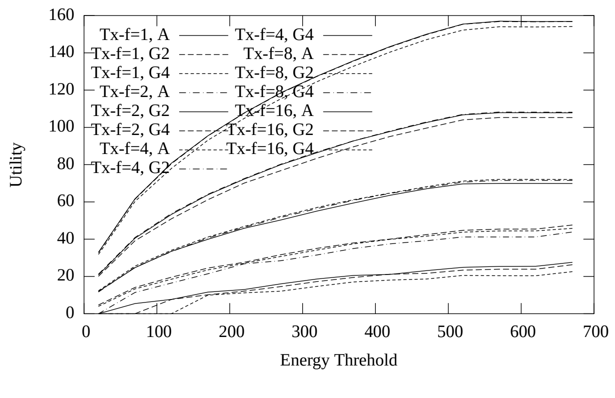


Figure 7. Utility with Energy & `Tx-factor`(X)

Figure 7 shows the plot of utility against $\overline{E}_{th}$ at various values of `Tx-factor` averaged over several randomly generated tasks. As expected, when $\overline{E}_{th}$ increases, cool-down times decrease and higher utility is accrued since higher energy levels are available. As `Tx-factor` increases, higher transmission times are required for achieving the same tracking error, and utility accrued is lowered since the system runs out of time.

In fact, at a `Tx-factor` of 16 and $\overline{E}_{th}$ of around 100, not all tasks are admitted into the optimizer under `G2` or `G4`. That is, some tasks do not even reach their minimum QoS operating points. These conditions represent an *over-constrained* system, and occur for `Tx-factor` values above 16 and $\overline{E}_{th} \leq 100$. Likewise, the system becomes *under-constrained* for $\overline{E}_{th} \geq 500$.

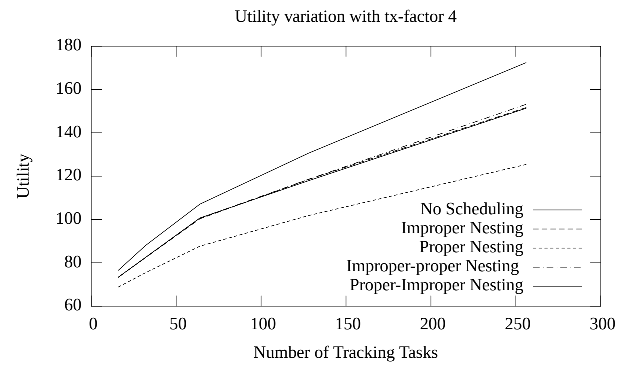


Figure 8. Utility Variation

Let us only consider the general case when all tasks are admitted and can get at least a minimum (non-zero) amount of tracking. Under these conditions, the *maximum* utility drop for `G2` relative to a wide choice of periods (represented by `A`) is 12.35% at an $\overline{E}_{th}$ value of $170J$ and a `Tx-factor` value of 16. Similarly, the maximum drop for `G4` is 24.5% at an $\overline{E}_{th}$ value of $270J$ and a `Tx-factor` value of of 16. The average utility drops for `G2` and `G4` are 2.22% and 6.82% respectively and the corresponding standard deviations are 9.4 and 38.83 respectively across the entire range of `Tx-factor` and $\overline{E}_{th}$. We limit periods to harmonics only to satisfy jitter constraints, and these experiments show that the choice of `G2` satisfies jitter constraints with only small drops in total utility obtained.

Next, we perform the iterative (binary search) process of resource allocation for tasks and analyze the performances of different dwell interleaving schemes.

5.3. QoS Optimization and Scheduling

In this set of experiments, we maintain an $\overline{E}_{th}$ value of $250J$[6] and a `Tx-factor` value of 4 with the aim of keeping the system to be neither under-constrained nor over-constrained. Under these conditions, smart schemes will be better able to exploit available resources to maximize overall utility. We vary the number of tasks from 16 to 256 and perform the whole iterative resource allocation process as shown in Figure 6. The period distribution is limited to `G2` (namely 100, 200, 400, 800 and 1600) based on our earlier experiments. The process of QoS optimization and generation of schedule repeats until we arrive at a schedulable task-set where the radar utilization has a precision of at least 0.1%.

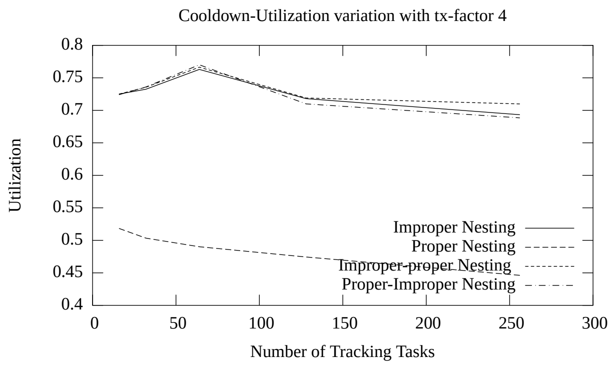


Figure 9. Avg Cool-down Utilization

Our next experiment deals with comparing performances of various interleaving schemes, under the condition that the schedulability test must be satisfied. We use the following 4 different interleaving schemes:

- **Proper scheme:** Perform Proper nesting of tasks alone.
- **Improper scheme:** Perform Improper nesting of tasks alone.

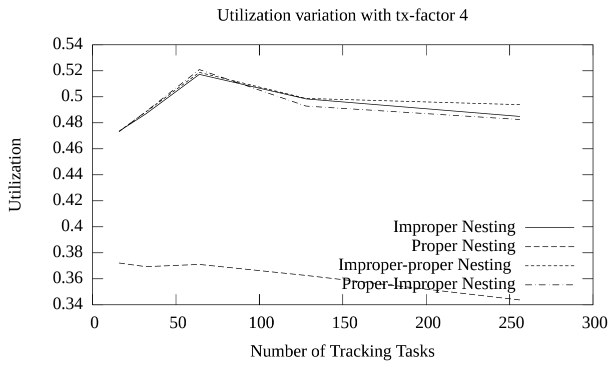


Figure 10. Avg Radar Utilization

• **Improper-proper scheme:** Perform Improper nesting followed by Proper nesting.

• **Proper-improper scheme:** Perform Proper nesting followed by Improper nesting.

For each task configuration and each interleaving scheme, we determine the overall utility accrued, the execution time and the radar utilizations. We finally average the results across several runs.

Figure 8 shows the variation of utility accrued as the number of tracking tasks increases under our interleaving schemes. From the figure, the `Improper-proper` scheme provides the highest utility followed in descending sequence by `Proper-Improper`, `Improper` and `Proper`. The difference in utility between `Improper-proper` and `Proper` is 18.11% at 256 tasks. The drop in utility from Q-RAM with no interleaving to `Improper-proper` is 11.17% for the same number of tasks. In other words, the need to schedule the raw outputs of Q-RAM (operating with only 100% utilization requirements and no scheduling constraints) leads to a drop of 11.17%.

Next, we plot the variation of radar cool-down utilization ((8)) under these schemes in Figure 9. We again observe that the `Improper-proper` scheme provides the best utilization (close to 73.11%) and `Proper` scheme performs the worst (on average 48.64%). We also observe that the radar utilization actually drops for large task sets (e.g. from 128 tasks to 256 tasks). This is because we admit all tasks in the systems at their minimum QoS level before performing any optimization or allocation. This reduces the radar utilization of the system as the tasks are non-preemptive and we admit tasks whose minimum QoS itself requires high radar utilization. Our conclusion that the `Improper-proper` is the best interleaving scheme is further substantiated in the plot of radar utilization((1)) in Figure 10, as it gives higher utilizations than the other schemes. Figure 10 also shows that the cool-down utilization plays a bigger role in determining the utilization of the system than the radar utilization in our model.

Our experiments show that task-sets are best interleaved by improper nesting followed by proper nesting. They also show that improper nesting only performs more efficiently than proper nesting only.

5.4. Interleaving Execution Times

Finally, we measured the execution time consumed by resource allocation as the number of tasks changes and plotted the results in Figure 11. The execution time includes the Q-RAM optimization followed by the schedulability analysis on 4 radars. The execution times presented do not include the task profile generation time (which occurs only once) since it does not depend on the interleaving scheme and the task-set can choose profiles from a set of discrete profiles generated off-line[4]. The plot shows that all the interleaving schemes have rather comparable run-times, with `Improper` being the fastest and `Proper-Improper` being the slowest. As can be seen, with about 256 tasks, dynamic interleaving can be performed in about $2.5\ sec$ - with very little optimization carried out in our code. This shows the *reconfiguration rate* of the optimizer as defined in Section 2.3 can be once in $2.5\ seconds$ for 256 tasks. In other words, the radar system can re-evaluate the entire system and re-optimize every $2.5\ seconds$. In practice, the number of tasks is unlikely to exceed 100 tasks. In that case, the reconfiguration rate is increased to roughly once in $0.7\ seconds$. In addition, our experimental results show that this can be drastically reduced to the order of $100\ ms$ if we reduce the radar utilization precision by 1% or more.

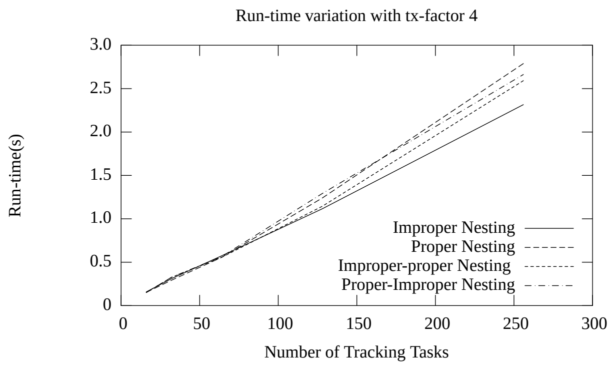


Figure 11. Resource Allocation Run-time

6. Conclusion and Future Work

We developed an integrated QoS resource management and scheduling framework, which can rapidly adapt to dynamic changes in the environment. This framework relaxes the need for maximizing (or minimizing) an objective function only using strict inequalities as resource constraints. We accomplish this by introducing an efficient schedulability test on system resources, and repeating the optimization a small number of times using a binary search technique. We apply our approach to a phased-array radar system, which introduces additional requirements. First, these systems also have stringent long-term and short-term energy constraints. Second, tasks in these systems must satisfy zero-jitter requirements. Third, radar tasks have multiple non-preemptive phases. We transformed the energy constraint into a timing constraint by defining a concept called *cool-down* time. We then restricted ourselves to choosing

4 The analysis of discrete profile generation has been studied in [2].

only harmonic periods in order to satisfy jitter requirements, and showed that such a restriction leads to only small drops in the overall utility of the system. Finally, we interleave the phases of different radar tasks so as to minimize the time utilization. We found that “improper nesting” of tasks followed by “proper nesting” yields the best results. Our future work will involve distributed QoS optimization and scheduling across larger-scale systems such as networked radar systems.

References

[1] R. Baugh. *Computer Control of Modern Radars*. RCA M&SR-Moorestown Library, 1973.

[2] S. Ghosh, J. Hansen, R. Rajkumar, and J. Lehoczky. Adaptive qos optimizations with applications to radar tracking. Technical Report 18-03-04, Institute for Complex Engineering Systems, Carnegie Mellon University, 2004.

[3] S. Goddard and K. Jeffay. Analyzing the real-time properties of a dataflow execution paradigm using a synthetic aperture radar application. In *Proceedings of the IEEE Real-Time and Embedded Technology and Applications Symposium*, June 1997.

[4] J. P. Hansen, J. Lehoczky, and R. Rajkumar. Optimization of quality of service in dynamic systems. In *Proceedings of the 9th International Workshop on Parallel and Distributed Real-Time Systems (WPDRTS)*, April 2001.

[5] M. O. Kolawole. *Radar Systems, Peak Detection and Tracking*. Newnes Press, 2002.

[6] T. W. Kuo, Y. S. Chao, C. F. Kuo, C. Chang, and Y. Su. Real-time dwell scheduling of component-oriented phased array radars. In *IEEE 2002 Radar Conferences*, Apr 2002.

[7] C. Lee. *On Quality of Service Management*. PhD thesis, Carnegie Mellon University, Aug. 1999.

[8] C. Lee, J. Lehoczky, R. Rajkumar, and D. Siewiorek. On quality of service optimization with discrete qos options. In *Proceedings of the IEEE Real-Time Technology and Applications Symposium*. IEEE, June 1998.

[9] K. Lin. Distributed pinwheel scheduling with end-to-end timing constraints. In *IEEE Real-Time Systems Symposium*, Dec 1995.

[10] M. Munu, I. Harrison, D. Wilkin, and M. Woolfson. Target tracking algorithms for phased array radar. *Radar and Signal Processing, IEE Proceedings-F*, 139(5):336–342, October 1992.

[11] R. Rajkumar, C. Lee, J. Lehoczky, and D. Siewiorek. A resource allocation model for qos management. In *IEEE Real-Time Systems Symposium*, December 1997.

[12] C. Shih, S. Gopalakrishnan, P. Ganti, M. Caccamo, and L. Sha. Scheduling real-time dwells using tasks with synthetic periods. In *Proceedings of the IEEE Real-Time Systems Symposium*, December 2003.

[13] C. Shih, S. Gopalakrishnan, P. Ganti, M. Caccamo, and L. Sha. Template-based real-time dwell scheduling with energy constraint. In *Proceedings of the IEEE Real-Time and Embedded Technology and Applications Symposium*, May 2003.

Filter	Computation Time(ms)	K_1	K_2	K_3	K_C
Kalman	0.022	0.60	0.4	1000.0	$[1, 16]$
Least-squares	.00059	0.60	0.4	30.71	$[1, 16]$
$\alpha\beta\gamma$	0.0004	0.80	0.2	0.0	$[1, 16]$

Table 3. Filter Constants

[14] D. Wilkin, I. Harrison, and M. Wooflson. Target tracking algorithms for phased array radar. *Radar and Signal Processing, IEE Proceedings-F*, 138(3):255–262, June 1991.

A. Tracking Error Computation

Type	Number of beams	Utility
Hi-Priority	15	20×0.3
	30	20×0.7
	45	20×0.85
	60	20×0.95
Lo-Priority	10	2×0.3
	20	2×0.7
	30	2×0.9

Table 4. Utility Distribution of Search Tasks

The tracking error is assumed to be a function of the environmental dimensions (target distance r_i, target speed v_i, target acceleration a_i, target noise n_i) and the operational dimensions (dwell period T_i, number of pulses η_i in dwell transmit time C_i, pulse width w, dwell transmit pulse power A_i, tracking filter algorithm) [2]. With the help of [5],[14] and [10], we formulate a general expression of tracking error as presented in (19). We do not claim this to be the precise expression.

$$\epsilon_i = \frac{\{K_1\sigma_r + K_2(\sigma_v T_i + K_3 a_i * T_i^2)\}}{(r_i - d)} \tag{19}$$

$$\sigma_r = \frac{c}{2B_w\sqrt{\frac{A_i(C_i/K_C)/(2T_i)}{n_i}}} \tag{20}$$

$$\sigma_v = \frac{\lambda}{2(C_i/K_C)\sqrt{\frac{A_i(C_i/K_C)/(2T_i)}{n_i}}} \tag{21}$$

$$B_w = \frac{M}{w} \tag{22}$$

$$d = v_i T + \frac{1}{2}a_i T^2 \tag{23}$$

Where
σ_r = standard deviation in distance measurement
σ_v = standard deviation in speed measurement
λ = wavelength of the radar signal
B_w = bandwidth of the radar signal
M = bandwidth amplification factor by modulation
d = estimated displacement of the target in time T
K_1 = position tracking constant
K_2 = period tracking constant
K_3 = acceleration tracking constant
K_C = transmission time tracking constant (`Tx-factor`).

The values of the constants are are presented in Table 3. The utility of this tracking error is given by (24).

$$U(\epsilon_i) = w_i(1 - e^{-\gamma/\epsilon_i}) \tag{24}$$

where γ is a constant to map the tracking error into a utility value and w_i is a weight function of the form:

$$w_i = K_t(\frac{v_i}{r_i + K_r}). \tag{25}$$

providing an estimate of the importance of a particular target. The K_r term represents the importance based on the target type, and the right-most term represents the time-to-intercept (i.e., the time that would be required for a target to reach the ship if flying directly toward it).

The utility distributions relative to the number of beams for search tasks are shown in Table 4.

B. Algorithms for Interleaving Schemes

Algorithm 1 Proper Nesting Algorithm

```
Require: $n > 1$
 1: $n_v \leftarrow n$ {$n$ = Number of inputted tasks, $n_v$ = number of virtual tasks}
 2: Create a sorted list of the tasks in increasing order of $(t_\text{C} + t_\text{X} + t_\text{W} + t_\text{r})$
 3: Create a sorted list of the tasks in increasing order of $t_\text{W}$
 4: loop
 5:   if $n_v > 1$ then
 6:     Choose the task $\tau_a$ with smallest $t_\text{C} + t_\text{X} + t_\text{W} + t_\text{r}$
 7:     Find the task $\tau_w$ with smallest possible $t_\text{W}$ that can properly nest $\tau_a$ in its $t_\text{W}$
 8:     if no task $\tau_w$ is found then
 9:       break from the loop
10:     else
11:       Fit $\tau_a$ inside $\tau_w$ by proper nesting (Figure 3) to form a single virtual task
12:       Remove the original two tasks from the sorted lists and insert the new virtual task into them
13:       $n_v \leftarrow n_v - 1$
14:     end if
15:   else
16:     break from the loop
17:   end if
18: end loop
```

Algorithm 2 Improper Nesting Algorithm

```
Require: $n > 1$
 1: $n_v \leftarrow n$ {$n$ = Number of inputted tasks, $n_v$ = number of virtual tasks}
 2: Create a sorted list of the tasks in increasing order of $t_\text{W}$
 3: loop
 4:   if $n_v > 1$ then
 5:     start with a task $\tau_w$ with biggest $t_\text{W}$
 6:     while a task is found do
 7:       Find a task $\tau_{wn}$ with biggest possible $t_\text{W}$ smaller than that of $\tau_w$ that can be the *leading task* in improper nesting with $\tau_w$
 8:       if $\tau_{wn}$ is found then
 9:         compute the nesting offset as $o_n$
10:       end if
11:       Find a task $\tau_{wi}$ with biggest possible $t_\text{W}$ smaller than that of $\tau_w$ that can be the *trailing task* in improper nesting with $\tau_w$
12:       if $\tau_{wi}$ is found then
13:         compute the nesting offset as $o_i$
14:       end if
15:       if both $\tau_{wn}$ and $\tau_{wi}$ are found then
16:         if $o_n < o_i$ then
17:           Merge $\tau_w$ and $\tau_{wn}$ by improper nesting with $\tau_{wn}$ as the leading task
18:         else
19:           Merge $\tau_w$ and $\tau_{wi}$ by improper nesting with $\tau_{wi}$ as the trailing task
20:         end if
21:         $n_v \leftarrow n_v - 1$
22:         Remove the merged two tasks from the sorted list and insert the new virtual task into it
23:       else if only $\tau_{wn}$ is found then
24:         Merge $\tau_w$ and $\tau_{wn}$ by improper nesting with $\tau_{wn}$ as the leading task
25:         $n_v \leftarrow n_v - 1$
26:         Remove the merged two tasks from the sorted list and insert the new virtual task into it
27:       else if only $\tau_{wi}$ is found then
28:         Merge $\tau_w$ and $\tau_{wi}$ by improper nesting with $\tau_{wi}$ as the trailing task
29:         $n_v \leftarrow n_v - 1$
30:         Remove the merged two tasks from the sorted list and insert the new virtual task into it
31:       else
32:         Go to the task with next lower $t_\text{W}$
33:       end if
34:     end while
35:   else
36:     break from the loop
37:   end if
38: end loop
```