Integrated Air and Missile Defense Resource Management

Matthew R. Smouse, Edwina P. Liu, and James J. Sylvester

ABSTRACT

Integrated air and missile defense (IAMD) resource management can apply to many different commodities within today's modern militaries. This article addresses radar resources, which are radio-frequency energy and time segments used to detect, track, and discriminate targets with a phased-array radar. IAMD radar resources can be managed at both the discrete dwell level and at the macro task level. The first part of this article presents an IAMD radar scheduling algorithm that uses a variation on interval and "earliest-deadline-first" scheduling to efficiently achieve desired search frame times while satisfying fixed task deadlines. The latter portion of the article then discusses the design of a track coordination algorithm for long-duration ballistic missile defense tasks. Both concepts are applicable to multifunction phased-array radars and were designed to improve efficiency while meeting existing performance parameters.

INTRODUCTION

Integrated air and missile defense (IAMD) is the set of capabilities that provides layered defense against aircraft, cruise missiles, and ballistic missiles. IAMD resource management is the set of combat system and system-of-systems capabilities that manage the finite resources available to any IAMD combat system or force. Figure 1 shows an example of the various desired capabilities and functions of a multimission (including IAMD) radar.

Legacy weapon systems are constrained; responses to new threats and complex environments for which these systems were not originally designed have to be addressed with legacy hardware, computers, and networks, increasing energy, time, and system complexity. New IAMD systems have fewer but different constraints on their resources to meet the same challenges. Yet regardless of the modernity of the system, the US Navy will never have enough resources to throw energy and metal into space without accounting for how those resources might be needed in the future and efficiently coordinating their application. This article presents two algorithms for IAMD radar resource management. The first algorithm is a scheduling algorithm for a single radar. The second algorithm is a coordination algorithm for multiple radars.

BACKGROUND

IAMD is fundamentally a multifaceted optimization problem. One goal of an IAMD radar control system is

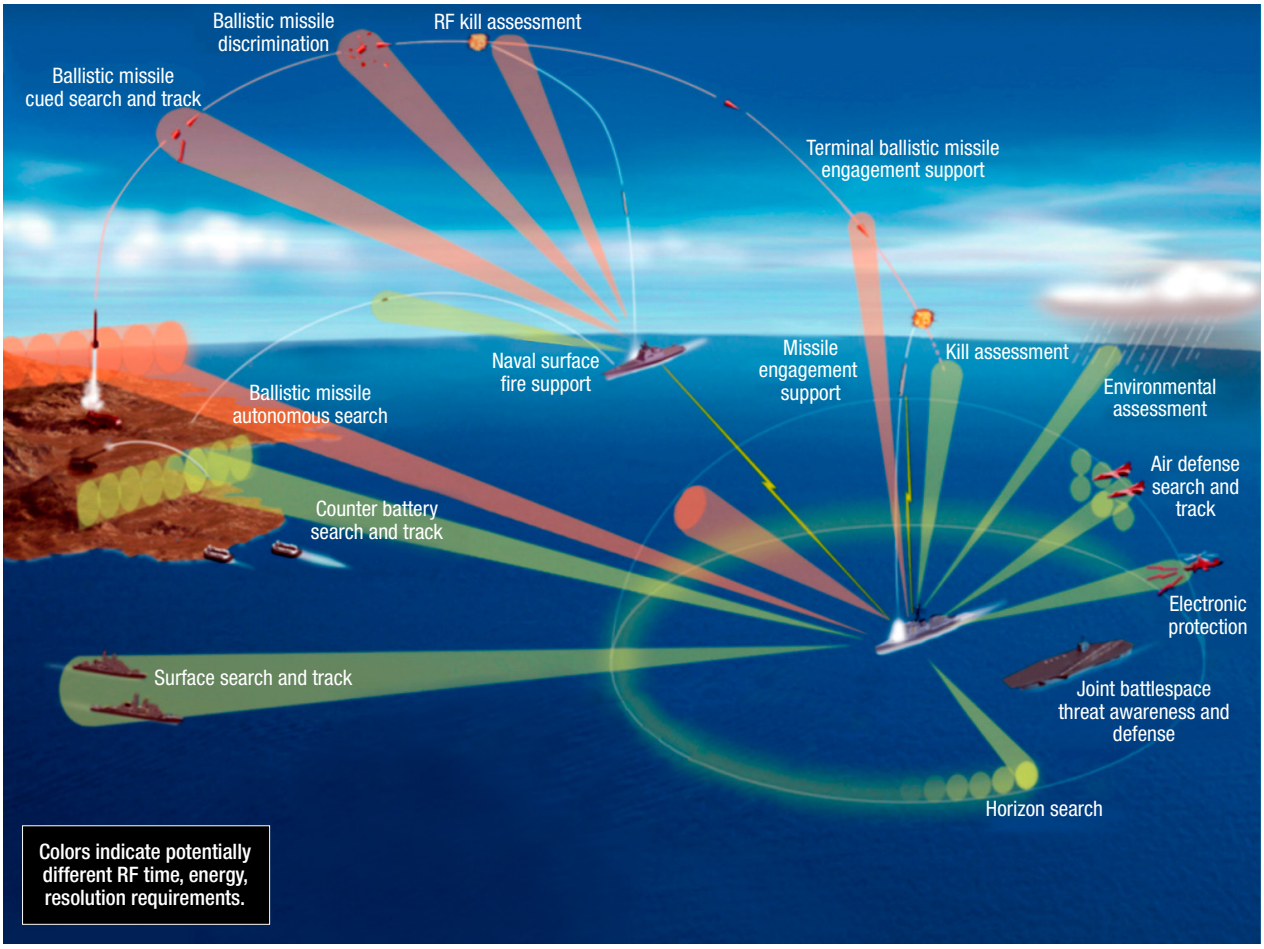

Figure 1. IAMD radar desired capabilities and functions.

to maximize the number of functions, such as search and track, that can be executed within a given period of time while constrained by the hardware capability and other limitations. Considerations include duty factor, peak output power, signal processing throughput, instantaneous bandwidth, receive chain sensitivity, and array architecture. Radar control algorithms schedule radar tasks in a recurring radar scheduling interval (RSI) according to prioritized event queues while adhering to the aforementioned constraints. For each RSI, legacy algorithms would start from the highest-priority queues and only move to lower-priority queues when the high-priority queues were empty and there was still a vacancy in the RSI. This approach may be described as a "greedy scheduler" because the RSI is fixed (not flexible) and resources are applied only to appease priority, not to achieve efficiency.

Efficient IAMD resource management applies techniques based on optimization theory, specifically combinatorial optimization. This particular branch of mathematical optimization treats the sample and solution sets as discretes. Radar tasks or "dwells" may be thought of as discrete events that must fit into a fixed container—the RSI. The goal is to completely fill the RSI with radar events without leaving gaps in the timeline. Combinatorial optimization examples in open-source literature, such as the knapsack problem and interval scheduling, are particularly applicable to radar scheduling. Interval scheduling is a problem in computer science in which the largest set of intervals (tasks) with fixed start and end times must be selected to execute within a given time period. There are variations on the interval scheduling problem that put tasks into groups and establish goals on the number of tasks scheduled from each group or weight the groups based on priority with the goal to maximize the weighted value of tasks scheduled. This final variation closely resembles the IAMD resource management problem as typically posed by system designers. Finally, the central processing unit (CPU) design has provided an example of an efficient dynamic scheduling algorithm, earliest-deadline-first (EDF) scheduling. If each task can be characterized by an arrival time (order in the queue), an execution requirement (duration), and a deadline (request time), then EDF will select tasks whose deadline is closest to the current time.

IAMD SCHEDULING ALGORITHM

A radar scheduler must meet the following criteria:

- Achieve specific search frame times (nominally the time between revisiting each point in space—or, equivalently, each beam position).
- Perform specific track functions while minimizing dropped track update events. (A track update is a radar beam transmitted to a position where a target is anticipated plus the subsequent detection on that target that "updates" its estimated location.)
- Maximize radar occupancy (scheduler efficiency).

Let the tasks that are to be scheduled by the radar be one of two types: fixed or flexible. Fixed tasks have a specific request time for execution. Flexible tasks have no specific request time, but they are considered to be continuing tasks consisting of predetermined sequences with the goal of completing each sequence at a certain minimum rate. Fixed and flexible tasks may have different priorities dependent on the objectives of the system as determined by the overall mission. Tasks are placed in queues based on the time that a request is made.

Fixed tasks have an assigned optimal requested transmission time for a single event (or "dwell") as determined by the fixed task manager. Fixed tasks generally have a small amount of slack, which is the maximum amount of time the scheduled time can deviate (either earlier or later) from the requested time. The slack will vary according to the type of radar task. Examples of fixed tasks for radar include track, discrimination, missile communications, and cued acquisition events.

Flexible tasks have a defined set of events to be executed in a specific order or pattern. Events assigned to each flexible task may have different lengths. Although these events have no specific request times, the pattern as a whole will have minimum and maximum time periods within which it must be executed. Flexible task patterns will be repeated indefinitely by the system until commanded to stop. All flexible task requests associated with specific tasks within a queue will have the same priority. Examples of flexible tasks are volume search, clutter mapping, and essential test functions.

The scheduling algorithm presented is a variation on interval scheduling and EDF. It uses an EDF approach to select fixed tasks (radar events that have a deadline) when composing a dynamic RSI, but then uses a heuristic for setting a dynamic priority on flexible tasks (radar events that have no deadline but have goals on execution for that task queue). The algorithm consists of scheduling fixed tasks at or near their requested times and then filling in the intervals between fixed tasks with flexible task events. Flexible tasks are chosen based not on priority but rather on dynamically recalculated pattern rates. This single-pass algorithm design assumes that all radar events are duty-factor compliant and that fixed tasks will always have priority over flexible tasks. These assumptions do not always hold for US IAMD radars such as AN/SPY-1, -3, -4, and -6 due to radar hardware, algorithm design, and system performance constraints. However, this scheduling algorithm is an approach based on abstract principles that are relevant to all radar scheduling algorithms.

The IAMD scheduling algorithm leverages the attributes of fixed tasks to compose a dynamic RSI (DRSI). Figure 2 depicts fixed task attributes. Fixed tasks must be performed within a certain time period. The requested time is the time originally requested for the start of execution. The length is the period of time (occupancy) that the task will consume (i.e., the period of time between the start of the task and the next possible time that another task or event can be scheduled). The slack is the acceptable scheduling window before and after the requested time for that task. The earliest possible start time is the requested time minus the leading slack. The latest possible start time is the requested time plus the trailing slack. The leading and trailing slacks can be the same duration but are not required to be of the same duration. The DRSI is the interval between the end of one fixed task and the end of the next fixed task. The algorithm will attempt to place flexible task events in the DRSI.

The algorithm also requires that flexible task pattern states be monitored. The last time that a pattern was started is the pattern start time (PST). A frame (cycle of the pattern) is completed if all of the elements in the pattern have been executed once. The sum of all the event lengths in the pattern is the pattern length (PL). The desired time duration in which one frame of the pattern is to be completed is designated the desired frame time (DFT). The PL and DFT are typically not the same (otherwise the flexible task would require full occupancy).

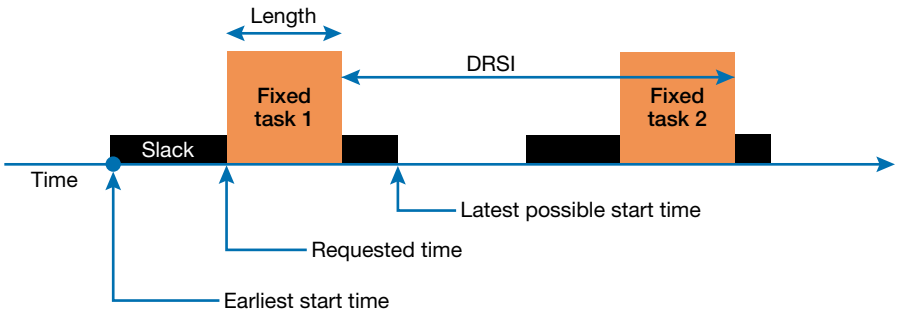

Figure 2. Fixed task attributes.

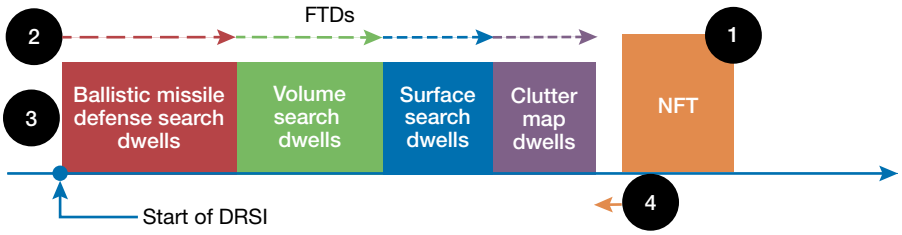

Figure 3. Scheduling algorithm steps.

The desired amount of the pattern scheduled at the current time is the PL divided by the DFT and multiplied by the difference between the current time and the PST. The flexible task deficit (FTD) is the desired amount of the pattern scheduled at the current time minus the sum of the event lengths already scheduled for the current cycle of pattern execution. For example, a flexible task's DFT is 8 s, and the difference between the current time and the PST is 4 s. If the PL for the flexible task is 3.2 s, then the desired amount of the pattern scheduled at the current time is 1.6 s. Suppose that the sum of the event lengths scheduled for the current cycle of the pattern is 1.4 s. The FTD is therefore 0.2 s. The FTD is used in the scheduling algorithm to dynamically prioritize that flexible task queue for the current DRSI.

The algorithm executes in four steps, as shown in Figure 3, with some examples of flexible task types. First, the next fixed task (NFT) is selected to create the DRSI. Second, FTDs are calculated to determine which flexible task events should be scheduled first and how many should be scheduled to fill the DRSI as much as possible and to reduce the largest FTD. Flexible tasks are considered in the order of their FTD, with longer FTDs given higher priority. Third, the flexible task events are scheduled up to the NFT with no gaps between each event. Fourth, the NFT is scheduled either at the earliest start time (in which case there might be a gap) by using up the available slack or immediately following the flexible task events previously scheduled (leaving no gap). The algorithm for selecting the NFT also allows for scheduling fixed tasks with no gap in between. Longer DRSIs are obtained and longer flexible tasks can be scheduled when fixed tasks can be placed adjacent to one another on the timeline. This feature addresses the situation when one of the queues of flexible tasks cannot advance because the DRSI is too short to insert an event from that queue.

The earliest fixed task request from each fixed task queue populates the NFT candidate pool. The selection algorithm first moves each candidate as late (to the right) as permitted by the candidate's slack (Figure 4). For case 1 (the highest-priority task has the earliest deadline), the selection algorithm chooses the highest-priority fixed task as the NFT. The algorithm attempts to move the NFT sooner so that it is adjacent to a previously scheduled fixed task with no gap in between. If the NFT cannot be shifted sooner, then it is kept as late as possible, and the next DRSI is defined as the interval between the current time and the end of that selected NFT. Figure 5 depicts task A creating a DRSI, flexible tasks scheduled prior to task A, followed by task B, which is scheduled with no gap after task A.

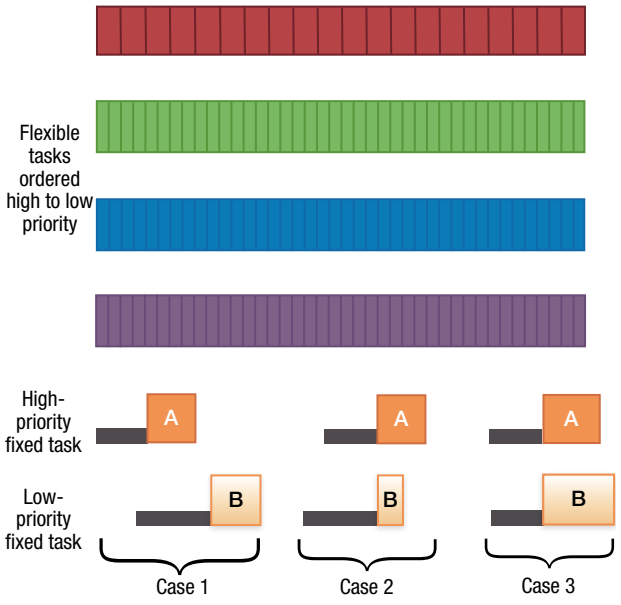

Figure 4. NFT selection.

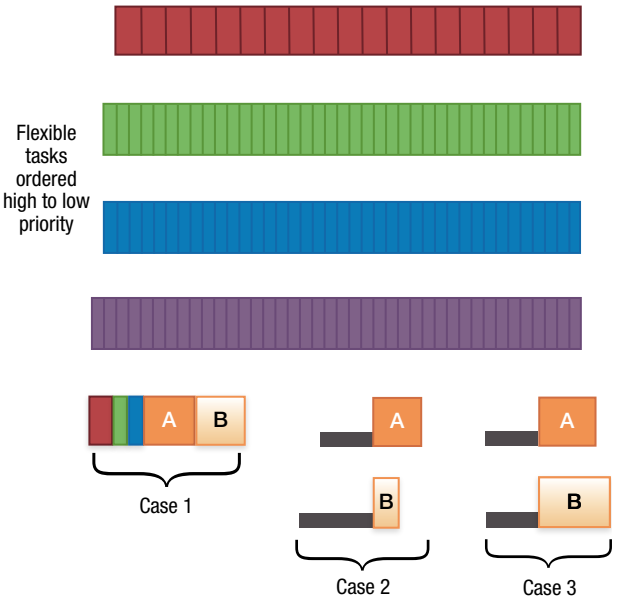

Figure 5. NFT case 1, priority order with no conflicts.

Once all the events have been placed in the DRSI, there is usually a gap that is an unfilled portion of the DRSI between the last scheduled flexible task and the beginning of the fixed task. The fixed task is moved sooner in the timeline until either the gap is closed or the earliest limit of the slack is reached. For case 1, task A was shifted slightly sooner so that there was no gap between it and the preceding flexible task event.

The algorithm selects a lower-priority task as the NFT if it starts earlier than a higher-priority task and there is enough slack so that the lower-priority fixed task can begin and end prior to the highest-priority fixed task. The lower-priority task forms the DRSI, and the higher-priority task is scheduled immediately following the lower-priority task. This is case 2, which is shown in Figure 6.

Case 2 also schedules tasks B and A so that no gap remains in the DRSI. But if the earliest limit of the slack in task B would have been reached such that it was not adjacent to the preceding flexible task event, then the gap could not have been closed. When a gap occurs, the gap length is tabulated for use when calculating the scheduler efficiency. The efficiency of the scheduler is the total time minus the sum of the gaps divided by the total time.

Finally, the algorithm will not schedule a lower-priority fixed task if there is a higher-priority fixed task that must be scheduled and the lower-priority fixed task is either too long or does not have enough slack to be accommodated. This is case 3, which is shown in Figure 7.

The higher-priority fixed task is chosen as the NFT when a conflict arises. In this case, the lower-priority fixed task is returned to the task manager. Notice also

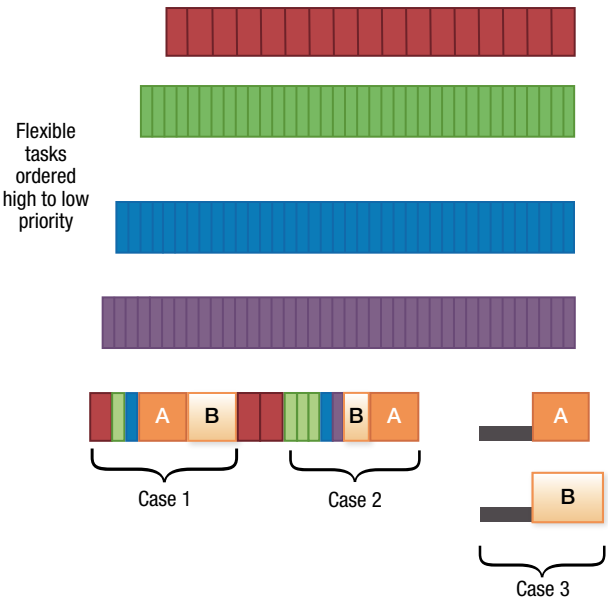

Figure 6. NFT case 2, priority-order inversion due to earliest deadline.

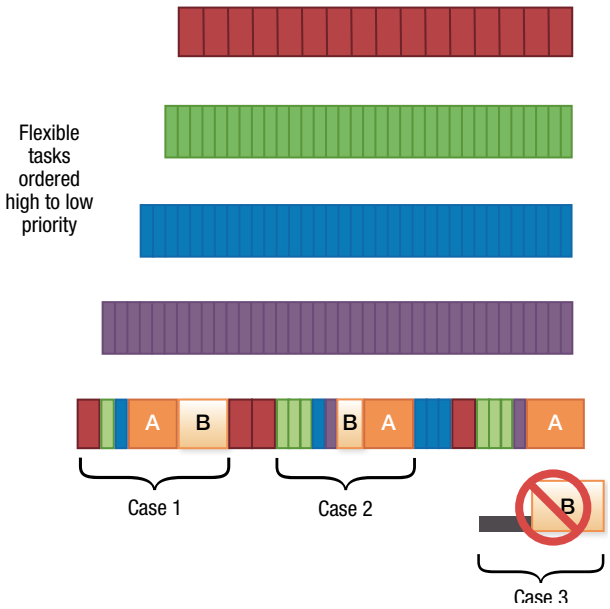

Figure 7. NFT case 3, unresolvable fixed task conflict.

how the flexible tasks have been selected in this particular example. The algorithm calculates queue FTDs each time a DRSI has been established. Because this is a one-pass algorithm, events from the queue with the largest FTD are scheduled first. A "priority" inversion in flexible task scheduling occurs when a lower-priority flexible task has a larger FTD. The algorithm fills the DRSI with as many events from the largest FTD queue as are necessary to reduce the FTD to zero or as can fit within the DRSI. The process repeats in deficit order until the DRSI is filled or all queues have been addressed. Flexible task priority is inherited from the flexible task parameters (DFT and PL).

Two additional, necessary algorithm features have been identified through simulation. The first feature is an adaptation of the NFT selection algorithm that restricts the maximum DRSI. The DRSI must be constrained to a maximum value so that if a new fixed task arrives on a queue it can be scheduled without delay or denial of service due to execution of a long DRSI. If the eligible NFT would create a DRSI that exceeds the maximum DRSI, then the current DRSI is closed with a flexible task rather than a fixed task. The second feature is an interrupt action that will allow new critical fixed tasks to preempt the current DRSI. The characteristics of a critical fixed task are high priority, a request time that is close to the current time, and no slack. Fixed and flexible task events are returned to their queues from the DRSI if a critical fixed task occurs.

The scheduling algorithm uses a four-step, single-pass approach with no look-ahead. The feature of keeping track of FTDs incorporates past performance into goals for the next DRSI and prevents one task from dominating the timeline due to priority alone. This fea-

ture also allows processes to degrade gracefully. The algorithm may sometimes not meet incremental goals but on average over longer time spans, DFTs are generally met, and few fixed tasks are lost (timed out) unless their requested time would have prevented a higher-priority fixed task from being executed. Note that performance of the algorithm still depends on proper fixed and flexible task parameter inputs, such as the amount of slack and DFTs. Requests may also require further prioritization in heavy loading conditions; this prioritization would involve adjustment of flexible task DFTs and potentially the parameters of events within the pattern as well as the overall pattern length.

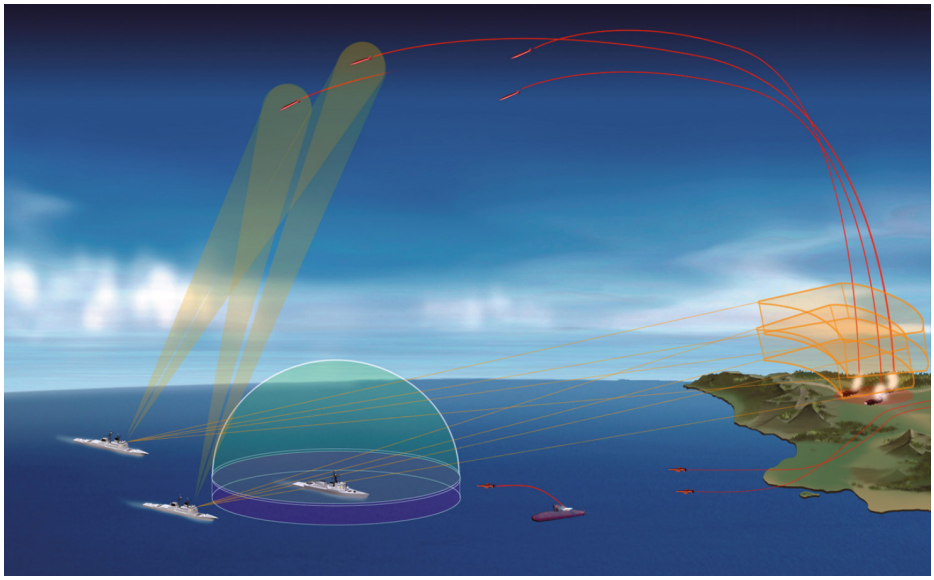

Figure 8. Notional redundant tracking without coordination.

BALLISTIC MISSILE DEFENSE TRACK COORDINATION ALGORITHM

Force-level radar resource management (FLRRM) for IAMD is an Office of Naval Research Future Naval Capabilities project to establish technology that will yield enhanced defensive performance through coordination of radar tasking. Although broad efforts within the project are examining many aspects of radar task management, there is a strong focus on the coordination of ballistic missile defense (BMD) tracking because of the inherent stressing nature of that task.

Coordination of BMD tracking solves limitations with current force planning and execution. Friendly force laydowns with no overlap in tasking (such as a sectored defense design) can limit raid performance because a BMD raid from a single launch area can exceed the single ship capacity. Friendly force laydowns with overlap in tasking also have limitations because overlapping radar search doctrine or cued acquisitions will lead to redundant tracking and possibly over-engagement without some form of intervening coordination. Although manual forms of coordination are possible, raid interval timing renders them ineffectual (Figure 8).

The objectives of FLRRM track coordination (FTC) are to increase probability of raid annihilation by reducing redundant BMD tracking and to preserve ship self-defense capability by coordinating between multiple ships. FTC takes on the form of a generalized assignment problem (GAP) as well as a multi-armed bandit (MAB) problem. The former class of problems inherits from combinatorial optimization. MAB is rendered from probability theory. The crucial difference is that combinatorial optimization requires an assignment for each "opportunity," whereas MAB does not. Because BMD track coordination does not allow abstention, it more naturally follows a GAP approach similar to the multiple knapsack problem.

The GAP formulation is shown in Figure 9. The number of tracks is N, and the number of sensors is M. A sensor-track pair is represented by ij. The profit and weight (cost) of pair ij are p_{ij} and w_{ij}, respectively. The total capacity of sensor i is represented by w_i. The assignment vector is x_{ij}; if x_{ij} is 1, then pairing ij is to be used in the solution (i.e., sensor i is preferred for track j). Typical solutions involve a dynamic program or an approximation algorithm. However, the nature of ballistic missile tracking and the available communication mechanism further constrain the problem. Each time a new ballistic missile is detected by one of the sensors or

$$\textbf{maximize} \quad \sum_{i=1}^{M}\sum_{j=1}^{N} p_{ij}x_{ij} \quad \textbf{subject to} \quad \sum_{j=1}^{N} w_{ij}x_{ij} \le w_i \qquad i = 1,\ldots,M$$

$$\sum_{i=1}^{M} x_{ij} \le 1 \qquad j = 1,\ldots,N$$

$$x_{ij} \in \{0,1\} \qquad i = 1,\ldots,M \qquad j = 1,\ldots,N$$

Figure 9. The GAP.

reported remotely, the coordination problem must be re-solved. But if the sensor-track assignments x_{ij} were to change upon introduction of the new track, then the fire control loop on each system may be irreparably disturbed. In other words, once a ballistic missile is assigned to a system, it must stay there. The algorithm may only coordinate an action on each new ballistic missile track while considering the current state of the known (extant) ballistic missile tracks. In addition, coordination must occur early in the tracking phase where little is known about the type and destination of the ballistic missile. Therefore, a simple yet effective profit function based on easily obtained information from the MIL-STD-6016 message set is desired so that a distributed algorithm can be implemented. The result is a first-in, first-out distributed greedy heuristic algorithm with sensor availability and confirmation messaging.

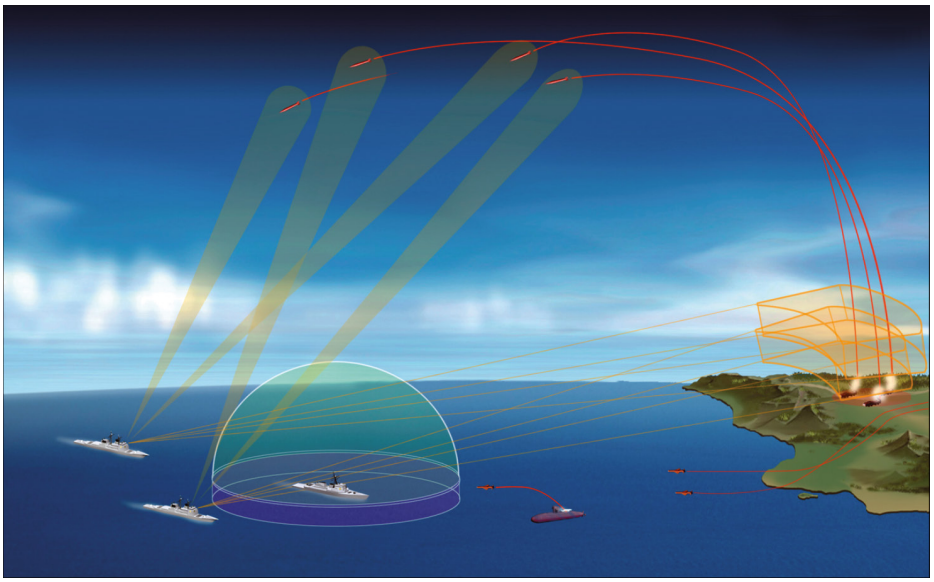

Figure 10. Notional FTC.

Algorithm studies show that FTC dramatically increases the number of engaged targets in various raid environments over an uncoordinated sensor network. Results for tactically relevant scenarios demonstrate an ability to provide high probability of raid annihilation that approaches ideal coordination. The coordination

FTC algorithms provide distributed, controlled BMD track coordination among US Navy IAMD platforms (primarily Aegis cruisers and destroyers). Figure 10 depicts a notional force coordination result with FTC. By avoiding modifications to link messages and the radar, FTC maintains a well-defined scope that is affordable and extensible. On each enabled platform, FTC inputs the space track picture from Link 16 messages conveyed via BMD communications links; performs processing in the command and decision element within the Aegis Weapon System; and then outputs recommendations to the operator. The information exchange requirements are limited to existing MIL-STD-6016 messages and exchanges, recognized by the Aegis BMD 5.1 program, and configuration and supervisory control messages between the operator and weapon system. Figure 11 depicts the modified functions in the Aegis Weapon System.

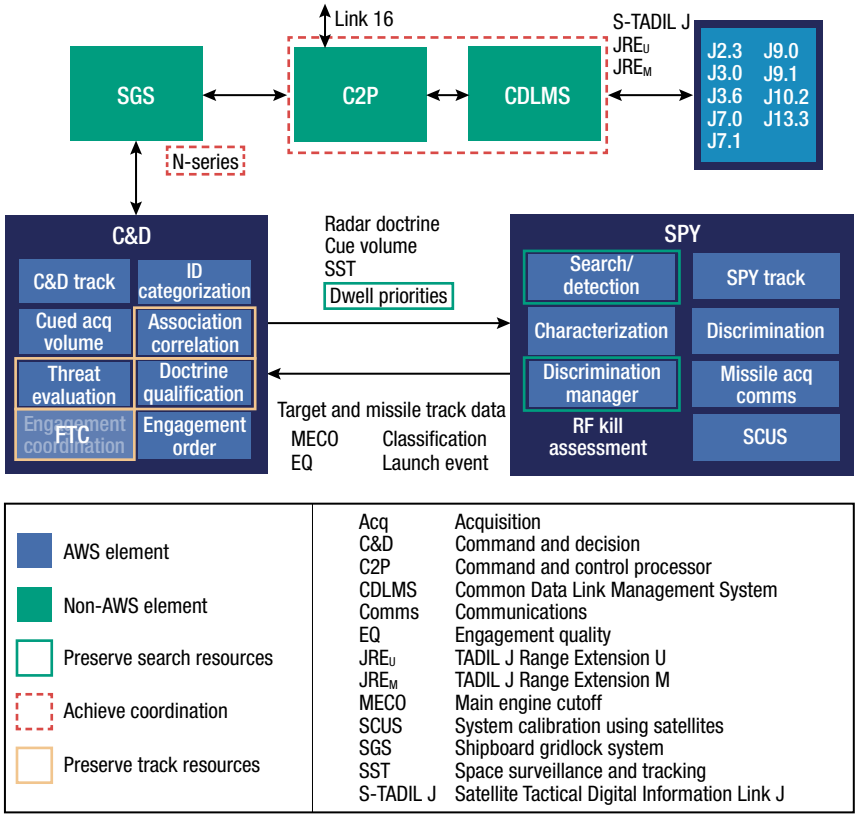

Figure 11. FLRRM FTC modifications to the Aegis Weapon System (AWS).

benefit applies broadly, including instances where only two ships are participating.

FLRRM has been transitioned to the Missile Defense Agency Aegis BMD (MDA/AB) program office and will undergo further enhancement, integration, and critical experiments within a program of record. FTC is expected to be fielded in 2020. Improvements using new sensors and communications links are also being explored.

Matthew R. Smouse, Air and Missile Defense Sector, Johns Hopkins University Applied Physics Laboratory, Laurel, MD

Matthew R. Smouse is a Principal Professional Staff member in APL's Air and Missile Defense Sector. He has a MS in computer engineering from Syracuse University and a BS in electrical engineering from Grove City College. At present, he leads modeling, simulation, and analysis for the Ground-based Midcourse Defense (GMD) technical direction agent. Matt's areas of expertise include digital signal processing; radar tracking; high-fidelity modeling and simulation development, analysis, verification, and validation; systems engineering/requirements analysis; and project management. He has received a number of awards for his work for the Missile Defense Agency. His email address is matthew.smouse@jhuapl.edu.

Edwina P. Liu, Air and Missile Defense Sector, Johns Hopkins University Applied Physics Laboratory, Laurel, MD

Edwina P. Liu is a Senior Professional Staff member and a software engineer in APL's Air and Missile Defense Sector. She has an MS in computer science from California State University, Sacramento, and a BS in chemical engineering from the University of California, Davis. Edwina has extensive experience in design and development of a broad variety of modeling and simulation application architectures, with a background in applying artificial intelligence techniques to process health monitoring and fault diagnostics and prognostics. Her email address is edwina.liu@jhuapl.edu.

James J. Sylvester, Air and Missile Defense Sector, Johns Hopkins University Applied Physics Laboratory, Laurel, MD

James P. Sylvester is a Senior Professional Staff member and a radar and software engineer in APL's Air and Missile Defense Sector. He has a BS in computer science from the University of Maryland, Baltimore County, and a BA in political science from the University of Maryland, College Park. He currently works on AN/SPY-1 radar analysis and model development, particularly with relation to dwell scheduling, and is team leader or co-leader for several modeling and simulation projects involving current and future AN/SPY-1 baselines. His email address is james.sylvester@jhuapl.edu.