Hierarchical Control of Multiple Resources in Distributed Real-time and Embedded Systems*

Nishanth Shankaran, Xenofon D. Koutsoukos,
Douglas C. Schmidt, and Yuan Xue
Department of EECS
Vanderbilt University

Chenyang Lu
Department of Computer Science and Engineering
Washington University, St. Louis

Abstract

Real-time and embedded systems have traditionally been designed for closed environments where operating conditions, input workloads, and resource availability are known a priori, and are subject to little or no change at run-time. There is increasing demand, however, for adaptive capabilities in distributed real-time and embedded (DRE) systems that execute in open environments where system operational conditions, input workload, and resource availability cannot be characterized accurately a priori. A challenging problem faced by researchers and developers of such systems is devising effective adaptive resource management strategies that can meet end-to-end quality of service (QoS) requirements of applications. To address key resource management challenges of open DRE systems, this paper presents the Hierarchical Distributed Resource-management Architecture (HiDRA), which provides adaptive resource management using control techniques that adapt to workload fluctuations and resource availability for both bandwidth and processor utilization simultaneously.

This paper presents three contributions to research in adaptive resource management for DRE systems. First, we describe the structure and functionality of HiDRA. Second, we present an analytical model of HiDRA that formalizes its control-theoretic behavior and presents analytical assurance of system performance. Third, we evaluate the performance of HiDRA via experiments on a representative DRE system that performs real-time distributed target tracking. Our analytical and empirical results indicate that HiDRA yields predictable, stable, and efficient system performance, even in the face of changing workload and resource availability.

1 Introduction

Distributed real-time and embedded (DRE) systems form the core of many mission-critical domains, including autonomous air surveillance [1], total ship computing environments [2], and supervisory control and data acquisition sys-

*This work is supported in part by DARPA, NSF CAREER award (CNS-0448554), Lockheed Martin ATL, BBN Technologies, and Raytheon.

tems [3, 4, 5]. Often, these DRE systems execute in *open* environments where system operating conditions, input workload, and resource availability cannot be characterized accurately *a priori*. These characteristics are beginning to emerge in today's large-scale systems of systems [6], and they will dominate in the next-generation of ultra-large-scale DRE systems [7]. Achieving end-to-end quality of service (QoS) is important and challenging for these types of systems due to their unique characteristics, including (1) constraints in multiple resources (*e.g.*, limited computing power and network bandwidth) and (2) highly fluctuating resource availability and input workload.

Conventional resource management approaches, such as rate monotonic scheduling [8], are designed to manage system resources and providing QoS in *closed* environments where operating conditions, input workloads, and resource availability are known in advance. Since these approaches are insufficient for open DRE systems, there is a need to introduce resource management mechanisms that can *adapt* to dynamic changes in resource availability and requirements. A promising solution is *feedback control scheduling* (FCS) [9, 10, 11], which employs software feedback loops that dynamically control resource allocation to applications in response to changes in input workload and resource availability. These techniques enable adaptive resource management capabilities in DRE systems that can compensate for fluctuations in resource availability and changes in application resource requirements at run-time. When FCS techniques are designed and modeled using rigorous control-theoretic techniques and implemented using QoS-enabled software platforms, they can provide robust and analytically sound QoS assurance.

Although existing FCS algorithms have been shown to be effective in managing a single type of resource, they have not been enhanced to manage multiple types of resources. It is still an open issue, therefore, to extend individual FCS algorithms to work together in a *coordinated* way to manage multiple types of resources, such as managing computational power and network bandwidth simultaneously. To address this issue, we have developed a control-based multi-resource management framework called *Hierarchical Distributed Resource management Architecture* (HiDRA). HiDRA employs a control-theoretic approach featuring two types of feedback controllers that coordinate the utilization of computational power and network bandwidth to prevent over-utilization of system resources. This capability is important because processor overload can cause system failure and network saturation can cause congestion and severe packet loss. HiDRA improves system QoS by modifying appropriate application parameters, subject to the constraints of the desired utilization.

This paper provides contributions to both theoretical and experimental research on FCS. Its theoretical contribution is its use of control theory to formally prove the stability of HiDRA. Its experimental contribution is to evaluate empirically how HiDRA works for a real-time distributed target tracking application built atop *The ACE ORB* (TAO) [12], which is an implementation of Real-time CORBA [13]. Our experimental results validate our theoretical claims and show that HiDRA yields predictable and high-performance resource management and coordination for multiple types of resources.

The remainder of the paper is organized as follows: Section 2 describes the architecture and QoS requirements of our DRE system case study; Section 3 explains the structure and functionality of HiDRA; Section 4 formulates the resource management problem of our DRE system case study described in Section 2 and presents an analysis of HiDRA; Section 5 empirically evaluates the adaptive behavior of HiDRA for our DRE system case study; Section 6 compares our research on HiDRA with related work; and Section 7 presents concluding remarks.

2 Case Study: Target Tracking DRE System

This section describes a real-time distributed target tracking system that we use as a case study to investigate adaptive management of multiple system resources in a representative open DRE system. The tracking system provides emergency response and surveillance capabilities to help communities and relief agencies recover from major disasters, such as floods, hurricanes, or earthquakes. In this system, multiple unmanned air vehicles (UAVs) fly over a pre-designated area (known as an "area of interest") capturing live images. The architecture of this distributed target tracking system, which is similar to other reconnaissance mission systems [14] and target tracking systems [15, 16], is shown in Figure 1.

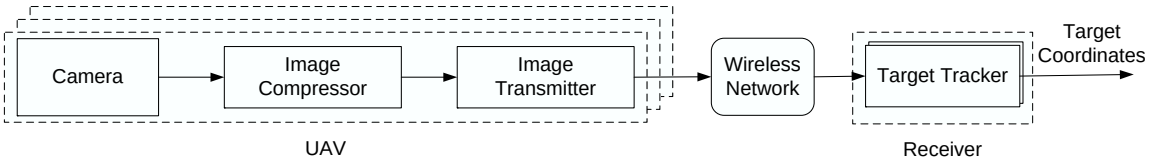


Figure 1: Target Tracking DRE System Architecture

Each UAV serves as a data source, captures live images, compresses them, and transmits them to a receiver over a wireless network. The receiver serves as a data sink, receives the images sent from the UAVs, and performs object detection. If the presence of an object of interest is detected in the received images, the tracking system determines the coordinates of the objects automatically and keeps tracking it. The coordinates of the object is reported to responders who use this information to determine the appropriate course of action, *e.g.* initiate a rescue, airlift supplies, etc. Humans, animals, cars, boats, and aircraft are typical objects of interest in our tracking system.

The QoS of our resource-constrained DRE system is measured as follows:

- *Target-tracking precision*, which is the distance between the computed center of mass of an object and the actual center of mass of the object, and
- *End-to-end delay*, which is the time interval between image capture by the UAV and computation of the coordinates of an object of interest. End-to-end delay includes image processing delay at the UAV, network transmission delay, and processing delay of the object detection and tracking sub-system at the receiver.

Just as any real-time system, end-to-end delay is a crucial QoS in our emergency response system and must be as low as possible. A set of coordinates computed with a lower precision and lower end-to-end delay is preferred over a set of coordinates computed with a higher precision and higher end-to-end delay.

There are two primary types of resources that constrain the QoS of our DRE system: (1) *processors* that provide computational power available at the UAVs and the receiver and (2) the *wireless network bandwidth* that provides communication bandwidth between UAVs and the receiver. To determine the coordinates accurately, images captured by the UAVs must be transmitted at a higher quality when an object is present. This in turn increases the network bandwidth consumption by the UAV. To increase the utility of the system, images are transmitted at a higher rate by the UAVs when objects of interest are

present in the captured images. This in-turn increases the processor utilization at the receiver node, and thus increases the processing delay of the object detection and tracking sub-system. Moreover, transmission of images of higher quality at a higher rate increases the bandwidth consumption by the UAV. If the network bandwidth is over-utilized considerably, the network transmission delay increases, which in-turn increases the end-to-end delay.

The utilization of the system resources (*i.e.*, wireless network bandwidth and computing power at the receiver) are therefore subject to abrupt changes caused by the presence of varying numbers of objects of interest. Moreover, the wireless network bandwidth available to transmit images from the UAVs to the receiver depends on the channel capacity of the wireless network, which in-turn depends on dynamic factors, such as the speed of the UAVs and the relative distance between UAVs and the receiver due to adaptive modulation [17, 18].

The coupling between the utilization of multiple resources, varying resource availability, and fluctuating input workloads motivate the need for adaptive management of multiple resources. To meet this need, the captured images in our system are compressed using JPEG, which supports flexible image quality [19]. Likewise, we choose to use image streams rather than video because video compression algorithms are computationally expensive, the computation power of the on-board processor on the UAVs is limited, and emergency response and surveillance applications and operators do not necessarily need video at 30 frames per sec. However, the computational power of the UAV on-board processor is large enough to compress images of the highest quality and resolution and transmit them to the receiver without overloading the processor.

In JPEG compression, a parameter called the *quality factor* is provided as a user-specified integer in the range 1 to 100. A lower quality factor results in smaller data size of the compressed image. The quality factor of the image compression algorithm can therefore be used as a *control knob* to manage the bandwidth utilization of an UAV. To manage the computational power of the receiver, end-to-end execution rate of applications is used as the control knob.

3 The Hierarchical Distributed Resource-management Architecture (HiDRA)

This section analyzes the adaptive management of multiple resources in open DRE systems using a control-based approach and presents the *Hierarchical Distributed Resource-management Architecture* (HiDRA), which employs a control-theoretic approach to manage processors and network bandwidth simultaneously. Our control framework is shown in Figure 2 and consists of three entities: *monitors*, *controller*, and *effectors*. A monitor is associated with a specific system resource and

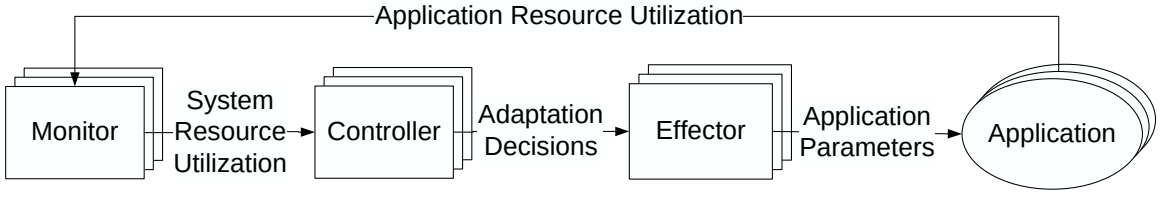


Figure 2: The HiDRA Control Framework

periodically measures the controller with the current resource utilization. The controller implements a particular control algorithm and computes the adaptations decisions for each application (or a set of applications) to achieve the desired system

resource utilization. Each effector is associated with an application and modifies the application parameters to achieve the controller-recommended application adaptation.

We proceed to instantiate the HiDRA control framework for the domain of target tracking described in Section 2. Each application in our DRE system is composed of two subtasks: *image compression* and *target tracking*. To ensure end-to-end QoS, therefore, resource utilization of both subtasks must be controlled. As shown in Figure 3, HiDRA consists of two

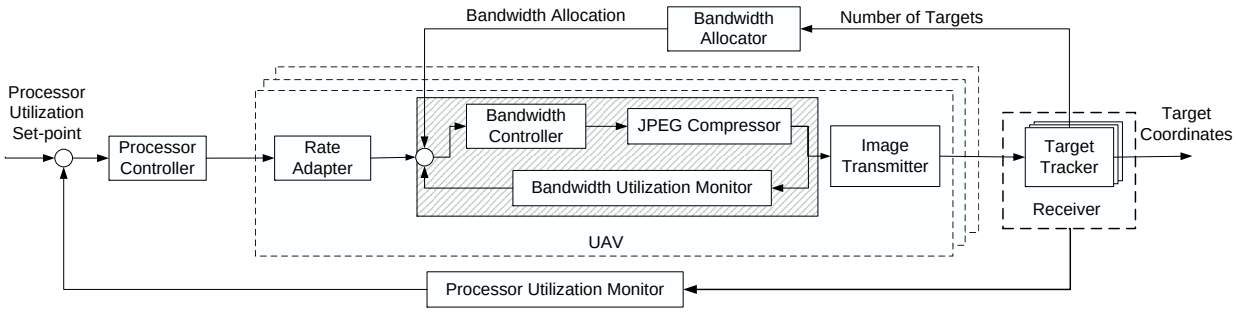


Figure 3: HiDRA's Control Architecture

types of feedback control loops: (1) a processor control loop located at the receiver that manages the processor utilization and (2) a bandwidth control loop located at each UAV that manages the bandwidth utilization. These loops control the utilization of the critical processor and network bandwidth resources and coordinate the execution of the image compression and target tracking subtasks. One approach to manage these system resources is to design *independent* feedback control loops. Unfortunately, this approach does not take into consideration the coupling between the two types of system resources and does not necessarily assure system stability. Therefore, we structure these control loops in a *hierarchical* fashion so that the processor control loop at the receiver is viewed as the *outer* control loop and the bandwidth control loop at each UAV is viewed as the *inner* control loop.

As shown in Figure 4, the processor utilization monitor and processor controller serve as the resource monitor and controller of the processor control loop, respectively. The objective of the processor controller is to ensure that the processor

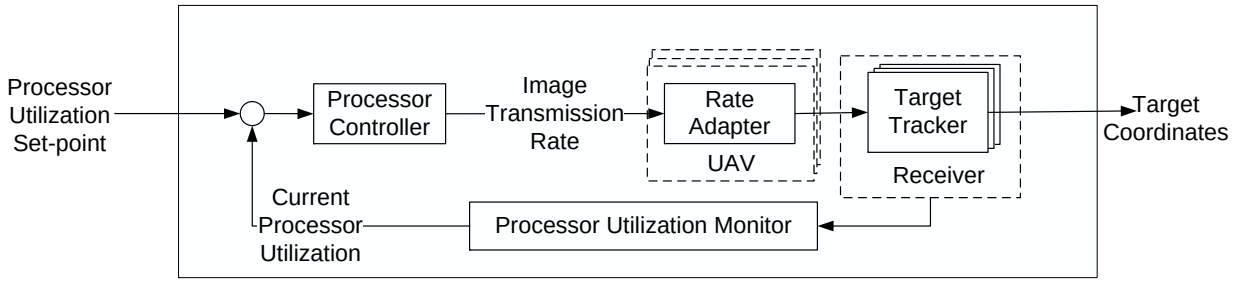


Figure 4: Processor Control Feedback Loop

utilization is maintained at a specified set-point in spite of variations in resource availability and input workload. The utilization set-point of the receiver processor is an input to the processor controller and is specified during system initialization. The

controlled variable for this loop is the processor utilization of the receiver, and the control input from the processor controller to the system is the image transmission rate, which is fed to the rate adapter in each UAV. For the processor control loop, therefore, rate adapters serve as effectors.

The bandwidth allocator shown in Figure 3 is responsible for dynamically computing the bandwidth allocation to each UAV based on (1) presence/absence of objects of interest in the images received from the corresponding UAV and (2) variations in available wireless network bandwidth. The bandwidth controller of each UAV views this allocation as the bandwidth utilization set-point. The bandwidth allocator ensures that the bandwidth requirement of UAVs capturing images of one or more objects of interest is met.

As shown in Figure 5, the bandwidth utilization monitor and the bandwidth controller serve as the monitor and controller of the bandwidth control loop, respectively. The objective of the bandwidth controller is to ensure that the bandwidth utilization

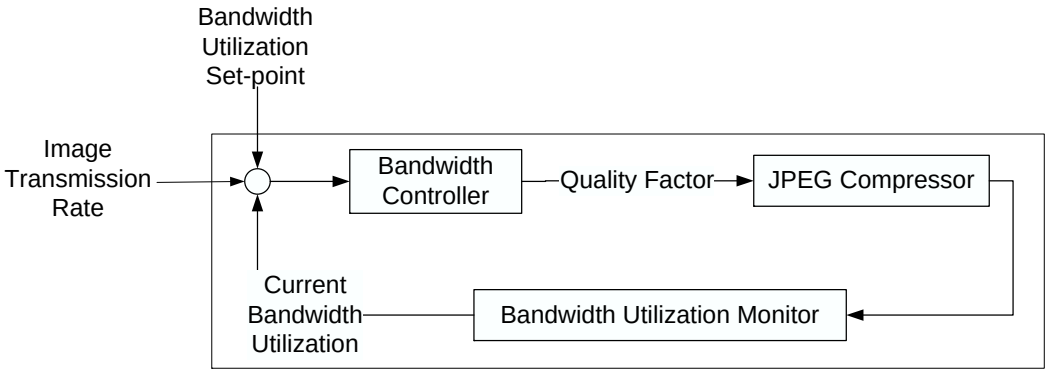


Figure 5: Bandwidth Control Feedback Loop

of the UAV is maintained at the specified set-point despite variations in resource availability and input workload. The image transmission rate and bandwidth utilization set-point are inputs to the bandwidth controller. Based on these inputs, the bandwidth controller computes an appropriate value of the JPEG quality factor to transmit the image of the highest quality, subjected to the specified bandwidth limitation. The controlled variable is the network bandwidth utilization of each UAV and the control input from the bandwidth controller to the system is the quality factor of the JPEG compression algorithm. This input is fed to the implementation of the JPEG compression algorithm, which serves as the effector for this control loop. The coupling between the two types of system resources is captured by using the image transmission rates computed by the processor controller as an input parameters to the bandwidth controllers.

4 Control Design and Analysis

This section first formalizes the resource management problem of our real-time distributed target tracking system. We then map HiDRA to this system to show how it addresses key resource management challenges of our DRE system. Finally, we present analysis that shows how HiDRA ensures the stability of our system. The formalism described below forms the foundations for the design and implementation of HiDRA and also provides analytical assurance about system performance

under fluctuating workload and varying resource availability.

4.1 Problem Formulation

The following formal notations are used throughout the remaining of the paper. The target tracking system consists of n UAVs, and therefore, n end-to-end tasks $\{T_i | 1 \leq i \leq n\}$, each with two subtasks, *i.e.*, image compression subtask executing at the UAV$_i$ and target-tracking subtask executing at the receiver. The sampling period of the processor controller (outer feedback loop) and the bandwidth controller (inner feedback loop) are represented by T_s^{out} and T_s^{in}, respectively. The sampling periods T_s^{out} and T_s^{in} are selected to be larger than the maximum task period. All the entities that make up the bandwidth control loop (such as monitor, controller, and effector) are collocated on each UAV. However, for the processor control feedback loop, the monitor and the controller are collocated on the receiver node, whereas the effectors are located at each UAV node. As a result, in the processor control feedback loop, the communication between the controller and the effectors is over a wireless network. Although there are no theoretical constraints on the sampling periods, for these practical reasons, T_s^{out} is selected to be greater than T_s^{in}. In our model, k^{th} and κ^{th} sampling period represent the k^{th} sampling period of the processor controller and the κ^{th} sampling period of the bandwidth controller, respectively.

Each end-to-end task T_i is invoked periodically at a rate $r_i(k)$ at the k^{th} sampling instant of the processor controller. The rate $r_i(k)$ is assumed to take values within the range $[r_i^{min}, r_i^{max}]$. During the k^{th} sampling instant of the processor controller, images are compressed and transmitted by T_i's data source, UAV$_i$, to the receiver at the rate of $r_i(k)$ images/second.

$C(k)$ represents the channel capacity (available bandwidth) of the wireless network during the k^{th} sampling period. This can be obtained form the wireless network card using operating system tools/commands such as `iwlist`.

4.1.1 Bandwidth Allocator

During each sampling period of the processor controller, the bandwidth allocator computes a desirable bandwidth allocation for each task T_i. The wireless network bandwidth allocation to each task T_i is recomputed by the bandwidth allocator if the presence of an object of interest was detected by any of the target-tracking subtasks or a variation in the available bandwidth was detected during the previous sampling period. For each task, bandwidth is allocated such that the net bandwidth utilization is below the set-point B^s, *i.e.*:

$$\sum_{i=1}^{n} b_i^s(k) \leq B^s C(k) \tag{1}$$

where $b_i^s(k)$ is the bandwidth allocation (utilization set-point) for task T_i during the k^{th} sampling period of the processor controller.

Let $p(k)$ and $p_i(k)$ represent the total number of objects of interest tracked by the system and the number of objects being tracked by T_i during the k^{th} sampling period, respectively. Let b_{min} represent the minimum bandwidth allocation to each task so that images of the lowest quality can be transmitted to the receiver. Bandwidth is thus allocated to each end-to-end task as a function of $p(k)$ and $p_i(k)$ as follows:

$$b_i^s(k) = \begin{cases} B^s C(k)/n & \text{if } p(k) = 0 \\ b_{min} + \frac{(B^s C(k) - n b_{min}) p_i(k)}{p(k)} & \text{if } p(k) > 0 \end{cases}, \forall \quad T_i \quad | \quad 1 \leq i \leq n. \tag{2}$$

If the total number of objects of interest tracked by the system is 0, bandwidth is equally allocated to each task. If the total number of objects of interest tracked by the system is greater than 0, bandwidth allocation to tasks is based on the number of objects currently being tracked by that task. This design ensures that a greater amount of bandwidth is allocated to tasks that are currently tracking objects of interest as compared to the ones that are not.

4.1.2 Processor Utilization Controller

We use the approach in [11] to model processor utilization. Section 4.2 uses the following model in the stability analysis of HiDRA. The target-tracking subtask of each end-to-end task T_i has an *estimated* execution time of c_i known at design time. The estimated processor utilization by the target-tracking subtask of task T_i during the k^{th} sampling period is denoted as $E_i(k)$ and is computed as

$$E_i(k) = c_i r_i(k) \tag{3}$$

where $r_i(k)$ is the invocation rate of end-to-end task T_i during the k^{th} sampling period. The net estimated processor utilization during the k^{th} sampling period is therefore

$$E(k) = \sum_{i=1}^{n} c_i r_i(k). \tag{4}$$

At runtime, however, the *actual* execution times may be different since they depend on the presence (and number) of objects in the images. At runtime, therefore, the actual processor utilization $U(k)$ can be written as

$$U(k) = G_p(k)E(k) \tag{5}$$

where $G_p(k)$ is the processor utilization ratio. Although, $G_p(k)$ is unknown, it is reasonable to assume that the worst case utilization ratio $G_p = \max_k\{G_p(k)\}$ is known. Let the processor utilization set-point of the receiver node be represented as U^s. From (5), the process utilization model can be written as

$$\Delta U(k+1) = \Delta U(k) + G_p v_p(k) \tag{6}$$

where $\Delta U(k) = U(k) - U^s$ and $v_p(k) = E(k+1) - E(k)$. The task of the feedback controller is to compute $v_p(k)$ so that $U(k)$ converges to U^s (or $\Delta U(k) \rightarrow 0$).

We consider a linear proportional controller

$$v_p(k) = K_p \Delta U(k) \tag{7}$$

where K_p is a control gain which will be selected so that the system is stable. The control signal $v_p(k)$ is implemented by the actuators by changing the invocation rate of end-to-end tasks. The closed-loop system is described by

$$\Delta U(k+1) = [1 + K_p G_p]\Delta U(k) \tag{8}$$

The control algorithm is implemented as follows. During each sampling period, the controller compares the current processor utilization $U(k)$ with the utilization set-point U^s, and computes the net estimated utilization $E(k+1)$ for the next sampling period based on the equation $E(k+1) = E(k) + K_p \Delta U(k)$. Since the presence of one or more objects of interest

in the received images increases the execution time the target-tracking subtask, computational power is allocated to target tracking subtasks based on the number of objects of interest that are present in the received images. We therefore have

$$E_i(k+1) = \begin{cases} \frac{E(k+1)}{n} & \text{if } p(k) = 0 \\ E_{min} + \frac{(E(k+1) - nE_{min})p_i(k)}{p(k)} & \text{if } p(k) > 0 \end{cases}, \forall \quad T_i \quad | \quad 1 \leq i \leq n \tag{9}$$

where $p(k)$ represents the total number of objects of interest captured by all the tasks in the system, $p_i(k)$ represents the number of objects of interest being captured by T_i during the k^{th} sampling period, and E_{min} represents the minimum processor allocation to each task so that images can be processed by the receiver at the lowest rate.

If the total number of objects of interest tracked by the system is 0, computational power is equally allocated to each task. If the total number of objects of interest tracked by the system is greater than 0, however, allocation of computational resource to tasks is weighted based on the number of objects currently being tracked by that task. This design ensures that a greater amount of computational power is allocated to tasks that are currently tracking objects of interest as compared to the ones that are not. From equations (3), (7), and (9) we derive the task execution rate as follows:

$$r_i(k+1) = \begin{cases} \frac{E(k) + (U(k) - U^s)K_p/G_p}{nc_i} & \text{if } p(k) = 0 \\ \frac{E_{min}}{c_i} + \frac{p_i(k)(E(k) + (U(k) - U^s)K_p/G_p - nE_{min})}{p(k)c_i} & \text{if } p(k) > 0 \end{cases}, \forall \quad T_i \quad | \quad 1 \leq i \leq n \tag{10}$$

4.1.3 Bandwidth Utilization Controller

We next present the analytical model of the bandwidth controller for each UAV. The following notations are used in this model where the symbols correspond to each UAV and the subscript is omitted for simplicity:

- $b(\kappa)$: Actual bandwidth utilization in the κ^{th} sampling period.
- $b^s(k)$: Desired bandwidth utilization (set-point) computed by the bandwidth allocator in the k^{th} sampling period as shown in equation (2).
- $r(k)$: Task rate computed by the processor controller in the k^{th} sampling period, as shown in equation (10).
- s: Size of an uncompressed image, which is a constant and known at design time.
- $q(\kappa)$: Quality factor of image compression algorithm (JPEG) computed by the bandwidth controller in the κ^{th} sampling period.
- $\phi(q)$: Estimated size of the compressed image compressed with quality factor q.

To simplify our notation, we express $r(k)$ and $b^s(k)$ with respect to the index κ by defining $r(\kappa) = r(k), b^s(\kappa) = b^s(k), k \leq \kappa < k+1$.

The controlled variable of this feedback control loop is the bandwidth utilization, $b(\kappa)$, and the control input from the controller to the UAV is the quality factor of the image compression algorithm, $q(\kappa)$. The controller computes an appropriate value of quality factor, $q(\kappa)$, to ensure that the bandwidth utilization of the UAV, $b(\kappa)$, converges to the set-point, $b^s(\kappa)$, computed by equation (2).

The average size of the compressed image, $\phi(q)$, is related to the quality factor of the image compression algorithm, q, by a non-linear function as shown in Figure 6. For the purpose of our control design, however, we choose q within the

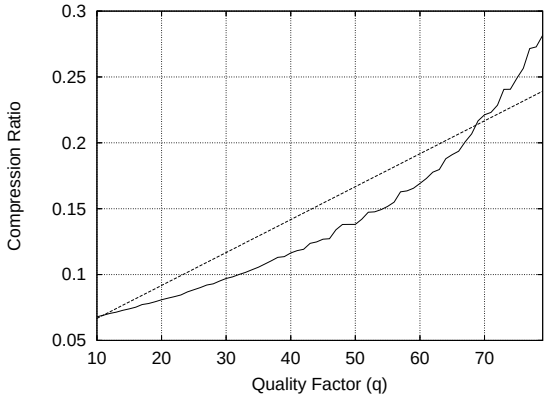


Figure 6: Linearization of $\phi(q)$

range $[10, 70]$ where this function can be approximated by a linear one. A piecewise linear function can also be used. For $10 \leq q \leq 70$, we have

$$\phi(q) = sgq + \omega \tag{11}$$

where g is the slope and ω is the y-intersect of the linear approximation of the function in Figure 6.

The bandwidth utilization monitor measures the bandwidth utilization as the rate at which data is written by the image compression subtask to the underlying network stack, and therefore the resolution of the bandwidth utilization monitor is in the order of the size of the compressed image. Hence, even a small variation in the sampling period and/or image transmission rate will considerably affect the measured bandwidth utilization. To remove the measurement noise in the measured bandwidth utilization, we employ a Kalman filter [20] to estimate the actual bandwidth utilization. It should be noted that the processor utilization monitor obtains the processor utilization directly from the underlying operating system, and therefore is of higher resolution compared to the bandwidth utilization monitor.

The following notations are used in the description of the Kalman filter:

- $\tilde{b}(\kappa)$: Measured bandwidth utilization in the κ^{th} sampling period.
- $\hat{b}^{-}(\kappa)$: Predicted bandwidth utilization in the κ^{th} sampling period.
- $\hat{b}(\kappa)$: Bandwidth utilization in the κ^{th} sampling period estimated using a Kalman filter.
- $P^{-}(\kappa)$: *a priori* estimate of variance in $b(\kappa)$.
- $P(\kappa)$: *a posteriori* estimate of variance in $b(\kappa)$.
- R: Variance in measurement noise in the measured bandwidth utilization.
- $K(\kappa)$: Computed Kalman gain in the κ^{th} sampling period.

Images are compressed with a quality factor q and transmitted at the rate of $r(\kappa)$ images per second from the UAV to the receiver. The following expression gives the predicted bandwidth utilization by the UAV:

$$\begin{aligned} \hat{b}^-(\kappa) &= r(\kappa)\phi(q) \\ &= r(\kappa)sgq(\kappa) + r(\kappa)\omega. \end{aligned} \tag{12}$$

Using a Kalman filter, estimated bandwidth utilization is computed as follows:

$$\begin{aligned} P^-(\kappa) &= P(\kappa - 1) \\ K(\kappa) &= P^-(\kappa)(P^-(\kappa) + R)^{-1} \\ \hat{b}(\kappa) &= \hat{b}^-(\kappa) + K(\kappa)(\tilde{b}(\kappa) - \hat{b}^-(\kappa)) \\ P(\kappa) &= (1 - K(\kappa))P^-(\kappa). \end{aligned} \tag{13}$$

The output of the Kalman filter, $\hat{b}(\kappa)$, is used by the bandwidth controller as the current bandwidth utilization. We therefore get the network utilization model as

$$\Delta\hat{b}(\kappa + 1) = \Delta\hat{b}(\kappa) + r(\kappa)sgv_b(\kappa) \tag{14}$$

where $\Delta\hat{b}(\kappa) = \hat{b}(\kappa) - b^s(\kappa)$ and $v_b(\kappa) = q(\kappa + 1) - q(\kappa)$.

We consider a linear controller

$$r(\kappa)sgv_b(\kappa) = K_b\Delta\hat{b}(\kappa) \tag{15}$$

where K_b is the control gain that will be selected so that the system is stable. The closed-loop system is

$$\Delta\hat{b}(\kappa + 1) = [1 + K_b]\Delta\hat{b}(\kappa). \tag{16}$$

During each sampling period, the controller compares the estimated bandwidth utilization $\hat{b}(\kappa)$ with the utilization set-point $b^s(\kappa)$, and computes the quality factor $q(\kappa + 1)$ by

$$q(\kappa + 1) = q(\kappa) + K_b\Delta\hat{b}(\kappa)/r(\kappa)sg. \tag{17}$$

4.2 Stability Analysis

A control system is said to be stable if and only if the system converges to an equilibrium for any set of initial conditions. In our case study, the initial conditions are used to represent the changes in workload (due to the change of the images' content) and/or resource availability. Our target tracking system is therefore stable if resource utilization of both the system resources (*i.e.*, processor utilization at the receiver and the network bandwidth utilization) converge to their respective utilization set-points in the presence of workload changes and/or resource availability. Although the controller is designed based on a time-invariant model (constant upper bounds on resource utilization), we show that the system is stable even when resource availability and/or utilization changes at runtime, *i.e.*, the system is time-varying.

We can stabilize each of the two types of feedback control loops by selecting the gains K_p and K_b so that the corresponding poles are in the unit circle. Such a design, however, does not necessarily ensure the stability of the hierarchical control

architecture since it does not take into consideration the interaction between the feedback loops (due to the presence of $r(\kappa)$ in equation (17)). We next present an analysis result that allows us to select the control gains so that the overall stability is assured. It should be noted that the introduction of the Kalman filter in the bandwidth utilization control loop does not affect the stability of the system and, therefore, is not considered in our analysis.

Assuming that the input buffer of the receiver is never empty, it is clear that the processor utilization is independent of the bandwidth utilization. If we select K_p so that $-2/G_p < K_p < 0$ then

$$\Delta U(k) = [1 + K_p G_p(k)]^k \Delta U(k_0), k \geq k_0$$

and $\Delta U(k) \rightarrow 0$ since $|1 + K_p G_p(k)| < 1$.

From equation (10), it follows that in the steady state the utilization for each task $U_i(k)$ will be stable (it will converge to a set-point U_i^s that depends on the presence of objects in the image data) and we can write

$$\Delta U_i(k+1) = \alpha_i(k) \Delta U_i(k) \tag{18}$$

where the function $\alpha_i(k)$ satisfies $|\alpha_i(k)| < 1$.

Let r_i^s denote the rate of the i^{th} task at the steady state, then $r_i(k) = r_i^s + \Delta r_i(k)$ where $\Delta r_i(k) \rightarrow 0$. From equation (16), the bandwidth utilization model for the i^{th} UAV is

$$\Delta b_i(\kappa+1) = [1 + (r_i^s + \Delta r_i(\kappa)) s g K_b^i] \Delta b_i(\kappa) \tag{19}$$

The primary challenge of the stability analysis of our framework is the coupling between the processor and bandwidth controllers. As it can be seen in equation (19), the control input from the processor controller to the system, $\Delta r_i(\kappa)$, is used by the bandwidth controller. Our objective is to deduce the stability properties of the system (18-19) by studying the *isolated system*

$$\Delta U_i(k+1) = \alpha_i(k) \Delta U_i(k) \tag{20}$$

$$\Delta b_i(\kappa+1) = [1 + r_i^s s g K_b^i] \Delta b_i(\kappa) \tag{21}$$

where the equations have been decoupled by setting $\Delta r_i(\kappa) = 0$.

Theorem 1. *The system (18-19) is stable if and only if the isolated system (20-21) is stable.*

Proof. Define the norm $|||[x_1, x_2]||| = |||[x_1, x_2]|||_\infty = \max\{|x_1|, |x_2|\}$ and denote $\Delta U_i(k), \Delta b_i(\kappa)$ and $\Delta U_i^I(k), \Delta b_i^I(\kappa)$ the solutions of (18-19) and (20-21) respectively.

"Only-if": If the system (18-19) is stable, then there exists function $\alpha(\kappa)$ with $\alpha(\kappa) \rightarrow 0$ such that

$$|||[\Delta U_i(\kappa), \Delta b_i(\kappa)]^T|| \leq \alpha(\kappa) |||[\Delta U_i(\kappa_0), \Delta b_i(\kappa_0)]^T|| \tag{22}$$

$\forall \kappa \geq \kappa_0$ and for every initial condition $[\Delta U_i(\kappa_0), \Delta b_i(\kappa_0)]^T$ where $\Delta U_i(\kappa) = \Delta U_i(k), k \leq \kappa < k+1$.

In particular, suppose that the initial condition is $[0, \Delta b_i(\kappa_0)]^T$, then by equation (22) $\forall \kappa \geq \kappa_0,\ |\Delta b_i^I(\kappa)| \leq \alpha(\kappa) |\Delta b_i^I(\kappa_0)|$, which shows that the system (20-21) is stable.

"If": It is easy to see that $\Delta U_i(k) = \Delta U_i^I(k)$ so we have to analyze only $\Delta b_i(\kappa)$. Define $\eta_I(\kappa) = 1 + r_i^s g K_b^i$ and $\eta(\kappa, \Delta r_i(\kappa)) = 1 + (r_i^s + \Delta r_i(\kappa)) g K_b^i$. From the stability of (20-21), we have that $|\eta_I(\kappa)| < 1$ and there exists a function $\alpha_2(\kappa)$ with $0 \leq \alpha_2(\kappa) \rightarrow 0$ such that

$$\Delta b_i^2(\kappa)(\eta_I^2(\kappa) - 1) \leq -\alpha_2(\kappa)\Delta b_i^2(\kappa_0)$$

for every $\Delta b_i(\kappa_0)$ and $\kappa \geq \kappa_0$. But we can write

$$\begin{aligned} \Delta b_i^2(\kappa+1) - \Delta b_i^2(\kappa) &= \Delta b_i^2(\kappa)(\eta_I^2(\kappa) - 1) + \Delta b_i^2(\kappa)(\eta^2(\kappa, \Delta r_i(\kappa)) - \eta_I^2(\kappa)) \\ &\leq -\alpha_2(\kappa)\Delta b_i^2(\kappa_0) + \gamma(\kappa) \end{aligned}$$

where $\gamma(\kappa) \rightarrow 0$ since $\Delta r_i(\kappa) \rightarrow 0$. $\Delta b_i(\kappa) \rightarrow 0$ and the system (18-19) is therefore stable. □

Using the above theorem, we can select the control gains so that our hierarchical control architecture is stable. For the processor utilization feedback loop, the gain could be selected to satisfy $-2/G_p < K_p < 0$ that ensures stability [11, 21]. Similarly, for the bandwidth utilization control loop, the gain should be selected so that (21) is stable. Since r_i^s is not known at design time, we can select the gain to satisfy $-2/(r_i^{\max}) < K_b^i < 0$. A reasonable choice for selecting the control gains is to use deadbeat control [22] based on the worst case utilization ratio and maximum task rate respectively, i.e. $K_p = -1/G$ and $K_b^i = -1/r_i^{\max}$. This selection tries to minimize the settling time keeping the overshoot equal to zero. Other criteria for selection of the gain can be found in [11].

5 Performance Results and Analysis

This section first presents the testbed for our target tracking system, which was used to evaluate the performance of HiDRA in the context of a representative open DRE system. We then describe our experiments and analyze the results obtained to evaluate the performance of our DRE system empirically with and without HiDRA under varying wireless bandwidth availability and input workload. The goal of our experiments was to validate our theoretical claims and show that HiDRA yields predictable and high-performance resource management and coordination for multiple types of resources.

5.1 Hardware and Software Testbed

Our experiments were performed on the Emulab [23] testbed at University of Utah (`www.emulab.net`). The hardware configuration consists of three nodes acting as UAVs and one receiver node. Images from the UAVs were transmitted to a receiver via a wireless LAN configured with a maximum channel capacity of 2 Mbps. The hardware configuration of all the nodes was a 3 GHz Intel Pentium IV processor, 1 GB physical memory, 802.11 a/b/g WIFI interface (Atheros 5212 chipset), and 120 GB hard drive. The Redhat 9.0 operating system with wireless support was used for all the nodes.

The following software packages were also used for our experiments: (1) **TAO 1.4.7**, which is our open-source implementation of Real-time CORBA [13] that HiDRA and our DRE system case study are built upon, (2) **Ffmpeg 0.4.9-pre1** with **Fobs-0.4.0** front-end, which is an open-source library that decodes video encoded in MPEG-2, MPEG-4, Real Video,

and many other video formats to yield raw images, and (3) **ImageMagick 6.2.5**, which is an open-source software suite that we used to compress the raw images to JPEG image format.

5.2 Target Tracking DRE System Implementation

The entities in our target tracking DRE system are implemented as CORBA objects and communicate over the *TAO* [12] Real-time CORBA Object Request Broker to achieve desired real-time performance. The end-to-end application consists of pairs of CORBA objects: the UAV data source and the receiver data sink. The UAV data source object that executes on each UAV's on-board processor performs the following actions: (1) extracts raw images from an on-disk video file using Ffmpeg with Fobs front end[1], (2) compress the raw image into JPEG format using ImageMagick, and (3) "pushes" the compressed images to the data sink object via a CORBA oneway method invocation.

A data sink object at the receiver processes the images received from the corresponding UAV. Each data sink object contains two functional modules: one that determines the presence of one or more objects of interest in the received images, and the other tracks the coordinates of objects of interest in the received image, if present. The second functional module is executed only if the presence of one or more objects of interest is detected by the first module.

To perform target tracking, received images are compared with a reference image, that is given during system initialization. To obtain the reference image, a raw image is extracted from a frame in the video that contains the object of interest. This raw image is then compressed using JPEG compression algorithm with a quality factor of 100 and used as the reference image. The received images are converted from color to gray-scale, and the processed image is "subtracted" from the reference image to obtain the difference image. If the average pixel value of the difference image is greater than a threshold (which indicates the presence of one of more objects of interest), the center of mass of the objected is computed. This approach is common and the coordinates of a moving object can be tracked using a Kalman filter [24].

Table 1 summarizes the number of lines of code of various entities in our middleware and DRE multimedia system case study.[2]

Entity	Total Lines of Source Code
HiDRA	12,243
DRE Target Tracking System	19,875
Ffmpeg + Fobs	214,092
ImageMagick	253,270
The ACE ORB (TAO)	907,035

Table 1: Lines of Source Code for Various System Elements

HiDRA is distributed with TAO, which is our open-source implementation of Real-time CORBA, and can be obtained from `http://deuce.doc.wustl.edu/Download.html`. Other open-source software packages used in our DRE

[1] We used pre-recorded video which was made available on each UAV node as our source of "live" video.

[2] Lines of source code was measured using SLOCCount (`http://www.dwheeler.com/sloccount/`).

multimedia system can be obtained from the following locations: Ffmpeg `http://ffmpeg.mplayerhq.hu/`, Fobs `http://fobs.sourceforge.net/`, and ImageMagick `http://www.imagemagick.org`.

5.3 Experiment Configuration

Our experiments consisted of three (emulated) UAVs containing the data source object that (1) decoded the video from a file, (2) extracted the raw images, (3) compressed them using JPEG compression, and (4) transmitted the compressed images to the corresponding data sink object at the receiver node. Wireless network bandwidth was shared between the three data source/data sink object pairs, and the computational power at the receiver node was shared between the three data sink CORBA objects.

We evaluated the adaptive resource management capabilities of HiDRA under the following operational conditions: (1) constant bandwidth availability and constant workload, (2) constant bandwidth availability and varying workload, (3) varying bandwidth availability and constant workload, and (4) varying bandwidth availability and varying workload. These experimental configurations were chosen to evaluate the performance of HiDRA under all possible combinations of fluctuations in bandwidth availability and input workload. In all operating conditions, we monitored the processor utilization at the receiver and wireless network bandwidth utilization between the UAVs and the receiver. Processor utilization at each UAV node was not monitored since the computational power of the UAV on-board processor was sufficiently large to compress images of the highest quality and resolution and transmit them to the receiver without overloading the processor.

Bandwidth consumption by each UAV was measured as the rate at which data was written to the underlying network stack by the UAV data source CORBA object. The bandwidth utilization can also be measured at the receiver node using the techniques described in [25]. Since our measurement of bandwidth consumption by each UAV was noisy, we used a Kalman filter to suppress the disturbances in the measured bandwidth utilization. Processor utilization at the receiver was measured using the data from the `/proc/stat` file. In our experiments, we also measured application QoS properties, such as target-tracking precision and average end-to-end delay.

We defined target-tracking precision as the inverse of *target-tracking error*, which is the distance between the computed center of mass of an object and the actual center of mass of the object. To compute the actual center of mass of the object, we identified an object present in the video as the object of interest, performed target-tracking on the raw images extracted from the video, and used this value as a reference. At the data sink object, the target-tracking results were then compared with this reference value.

End-to-end delay consists of (1) processing delay at the UAV, (2) network transmission delay from the UAV node to the receiver and (3) processing delay at the receiver node. To measure the end-to-end delay, an image was timestamped by the data source object when the raw image was extracted from the pre-recorded video file, before it was compressed and transmitted to the corresponding data sink object. Upon completion of processing of the received image by the data sink object, the timestamp of the image was compared with the current time on the receiver node to obtain the end-to-end delay. To eliminate time skews, physical clocks on all the nodes in our hardware testbed were synchronized using NTP [26].

In all the above listed operational conditions, we compare the performance of our DRE system when it was operated

with and without HiDRA. Comparison of system performance is decomposed into comparison of resource utilization and application QoS. For system resource utilization, we compare (1) wireless network bandwidth utilization and (2) processor utilization of the receiver node. For application QoS, we compare (1) target-tracking precision and (2) average end-to-end delay.

For all our experiments, we chose the sampling period of the processor controller and the bandwidth controller as 10 seconds and 1 second, respectively. The minimum and maximum image transmission rate $[r_{min}, r_{max}]$ was 5 and 15 images/second. The control gains for the processor controller (K_p) and the bandwidth controller (K_b) were -0.5 and -0.05, respectively. The processor utilization set-point was selected to be 0.7, which is slightly lower than rate monotonic scheduling [8] utilization bound for three tasks (0.77). The goal of utilization control is to (1) prevent processor overload (which can cause system instability), and (2) avoid unnecessarily under utilizing the processor (which leads to a low task rate). The choice of 0.7 as the set point achieves the desired trade off between overload protection and high task rate in our system. Since an IEEE 802.11 DCF-based network has a utilization of approximately 0.7 with 20 active nodes [27], the wireless bandwidth utilization set-point was also configured at 0.7. Although for a system with four nodes the achievable channel utilization could be higher than 0.7 (e.g., as high as 0.8), this value varies depending on many other factors such as packet size, channel bit rate, etc. Considering all these factors, we set the bound to 0.7 conservatively.

5.4 Experiment 1 : Constant Bandwidth Availability and Constant Workload

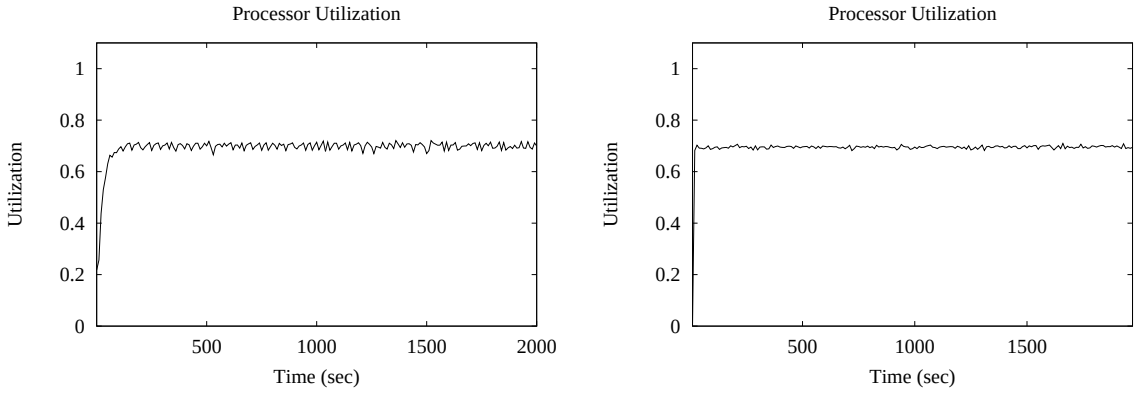
We now present the results obtained from running the experiment under a constant channel capacity of 2 Mbps and a constant 2 objects of interest tracked by the system. This experimental setup provides an operational condition where resource availability and input workload are known *a priori* and not subjected to change during the course of the experiments. Images containing objects of interest were captured by UAVs 1 and 2. This experiment serves as the baseline for all other experiments. It validates that when the tracking system is operated with HiDRA the following behavior occurs: (1) utilization of system resources converge to their respective set-points and (2) application QoS converges to the values that were obtained when the system was operated without HiDRA and application parameters were chosen *a priori*.

UAV	Image Transmission Rate (images/sec)	Quality Factor
UAV 1	10	40
UAV 2	10	40
UAV 3	10	40

Table 2: Application Parameters Chosen in Advance

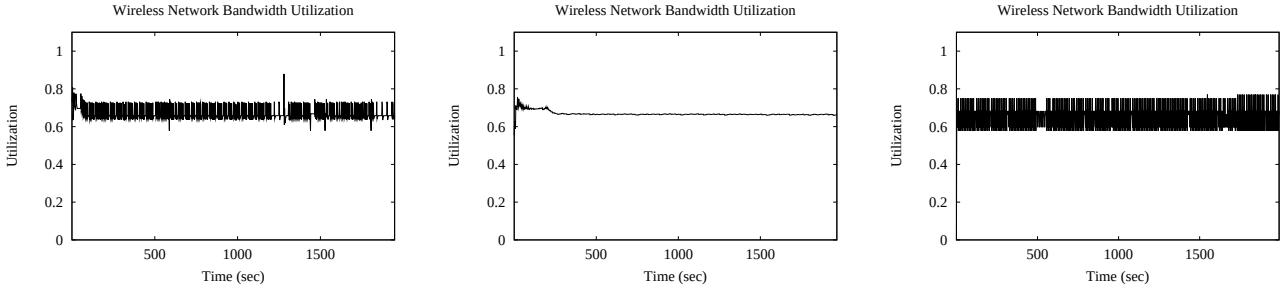
We compare the performance against a static configuration. In the static configuration, application parameters, such as image transmission rates and quality factor of the JPEG image compression algorithm, were chosen *a priori*. Values of these parameters were selected such that (1) both processor utilization of the receiver node and the wireless bandwidth utilization is equal to the set-point of 0.7 and (2) application QoS are maximized. The settings of the static configuration of the system are shown in Table 2.

5.4.1 Comparison of Resource Utilization



(a) Processor Utilization with HiDRA (b) Processor Utilization without HiDRA

Figure 7: Exp 1: Comparison of Processor Utilization



(a) Bandwidth Utilization with HiDRA (b) Bandwidth Utilization with HiDRA (estimates using a Kalman Filter) (c) Bandwidth Utilization without HiDRA

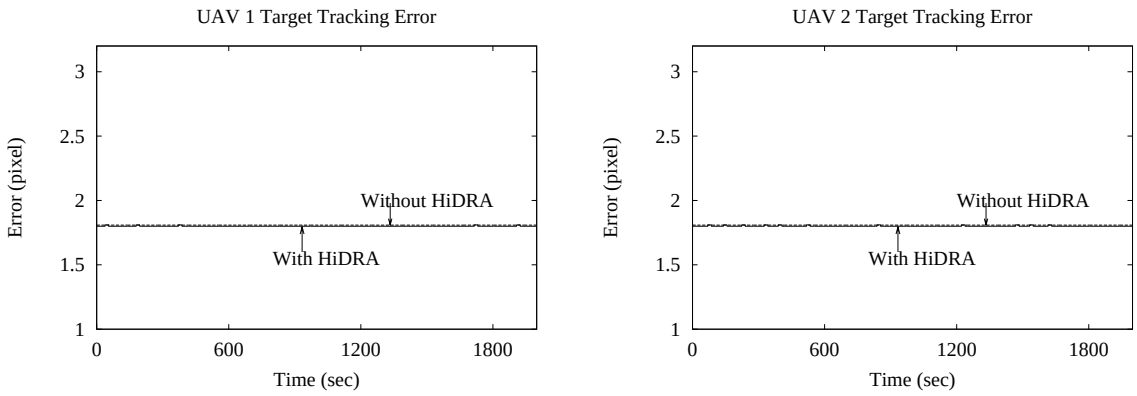
Figure 8: Exp 1: Comparison of Bandwidth Utilization

Figures 7 and 8 compare the processor utilization at the receiver node and the wireless network bandwidth utilization when the system was operated with and without HiDRA. The output of the bandwidth utilization monitor, shown in Figure 8b, was processed with a Kalman filter and used by the bandwidth controller as the current bandwidth utilization.

Figures 7b and 8c show that when the system was operated without HiDRA, resource utilization of both the resources is 0.7 during the course of the experiment. Similarly, Figures 7a, 8a, and 8b show that when the system was operated with HiDRA, resource utilization converges to the set-point of 0.7 and in maintained at 0.7 for the remaining duration of the experiment. These results show that when the system is operated using HiDRA, system resource utilization converges to the respective utilization set-points.

5.4.2 Comparison of QoS

We now compare the application QoS – (1) target-tracking precision, and (2) average end-to-end delay.



(a) UAV-1 (b) UAV-2

Figure 9: Exp 1: Comparison of Target-tracking Error

Figure 9 compares the target-tracking error obtained when the system was operated with and without HiDRA. Figures 9a and 9b show that average target-tracking error—and therefore target-tracking precision—is nearly the same when the system was operated with and without HiDRA.

Table 3, which compares the end-to-end delay when the system was operated with and without HiDRA, shows that average end-to-end delay is the same as when the system was operated with and without HiDRA. Based on these results, we conclude

Number of Objects	End-to-End Delay (msec)	
	With HiDRA	Without HiDRA
2	117	117

Table 3: Exp 1: Comparison of End-to-End Delay

that QoS of applications in our DRE system converges to the values obtained when the system was operated without HiDRA and application parameters were chosen *a priori*.

From our comparison of resource utilization and system QoS, we conclude that when the system is operated with HiDRA (1) utilization of system resources converge to their respective set-points and (2) application QoS converge to the values that were obtained when the system was operated without HiDRA and application parameters were chosen *a priori*.

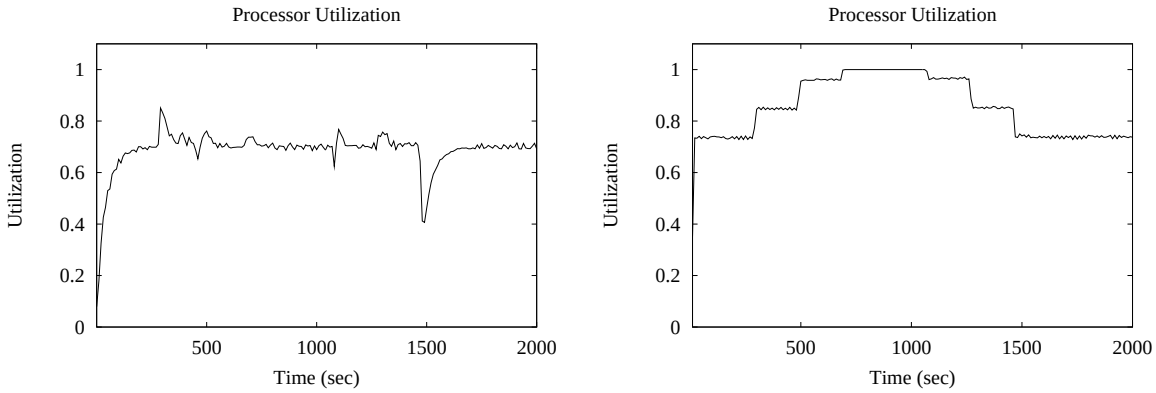
5.5 Experiment 2: Constant Bandwidth Availability and Varying Workload

We next present the results obtained from running the experiment under a constant channel capacity of 2 Mbps and varying number of objects of interest in the system. This experiment demonstrates the adaptive resource management capabilities of HiDRA under constant resource availability and varying input workload. Table 4 summarizes the number of objects of interest that were tracked as a function of time. In this experiment, when the system was operated without HiDRA, the static system configuration shown in Table 2 was used.

Time (sec)	Number of Objects			
	UAV 1	UAV 2	UAV 3	Total
0 - 300	0	0	0	0
300 - 500	1	0	0	1
500 - 700	1	1	0	2
700 - 1,100	1	1	1	3
1,100 - 1,300	0	1	1	2
1,300 - 1,500	0	0	1	1
1,500 - 2,000	0	0	0	0

Table 4: Objects of Interest as a Function of Time

5.5.1 Comparison of Resource Utilization



(a) Processor Utilization with HiDRA

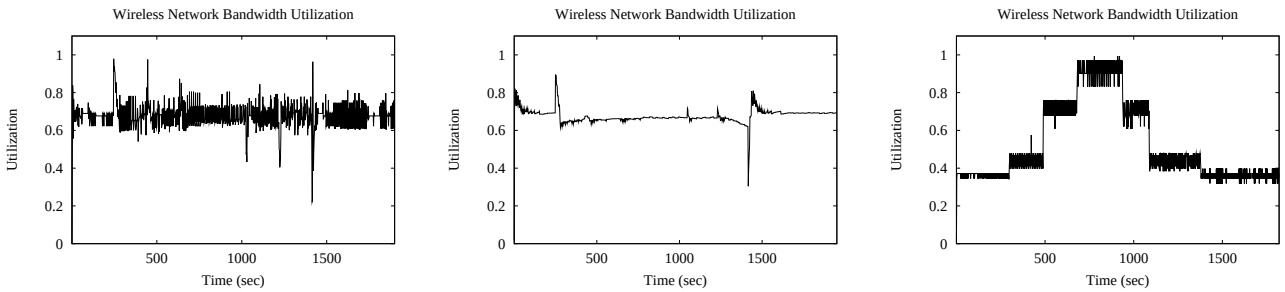
(b) Processor Utilization without HiDRA

Figure 10: Exp 2: Comparison of Processor Utilization

Figures 10 and 11 compare the processor utilization at the receiver node and the wireless network bandwidth utilization when the system was operated with and without HiDRA. The output of the bandwidth utilization monitor, shown in Figure 11b, was processed with a Kalman filter and used by the bandwidth controller as the current bandwidth utilization.

Figures 10a and 10b and Table 4 show that the increase in the processor utilization at $T = 300s$ is due to the presence of the first object of interest. Figure 10a shows that although the processor utilization increased above 0.7, within the next several sampling periods, HiDRA restored the processor utilization to the desired set-point of 0.7. HiDRA achieved this result by reducing the execution rates of data-source/receiver pair(s) deemed less important, *i.e.*, ones that captured images where objects of interest were absent. As shown in Figure 10b, when the system was operated without HiDRA, the processor utilization remained at 0.85, which is significantly higher than the utilization set-point of 0.7.

At $T = 500s$, the presence of the second object of interest was detected. The processor utilization thus increased to 0.9 when the system was operated without HiDRA, as shown in Figure 10b. As Figure 10a shows that the processor utilization quickly re-converges to the set-point after a transient increase when the system was operated with HiDRA.



(a) Bandwidth Utilization with HiDRA

(b) Bandwidth Utilization with HiDRA (processed using a Kalman Filter)

(c) Bandwidth Utilization without HiDRA

Figure 11: Exp 2: Comparison of Bandwidth Utilization

At $T = 700s$, the presence of the third object of interest was detected. As a result, when the system was operated without HiDRA, the processor utilization increased to 1, as shown in Figure 10b. Once again, Figure 10a shows that the processor utilization quickly re-converges to the set-point after a transient increase when the system was operated with HiDRA.

At $T = 1,100s$ the total number of objects being tracked by the system reduced from 3 to 2. Although there was a decrease in the processor utilization, HiDRA restored the processor utilization to the set-point by increasing the execution rate of important data-source/data sink pair(s). Similarly, HiDRA ensured that the processor utilization converges to the desired set-point for the remaining duration of the experiment. Similarly, Figures 11a and 11b shows how HiDRA ensures that the wireless bandwidth utilization converges to the desired set-point of 0.7 within bounded time, even under fluctuating workloads.

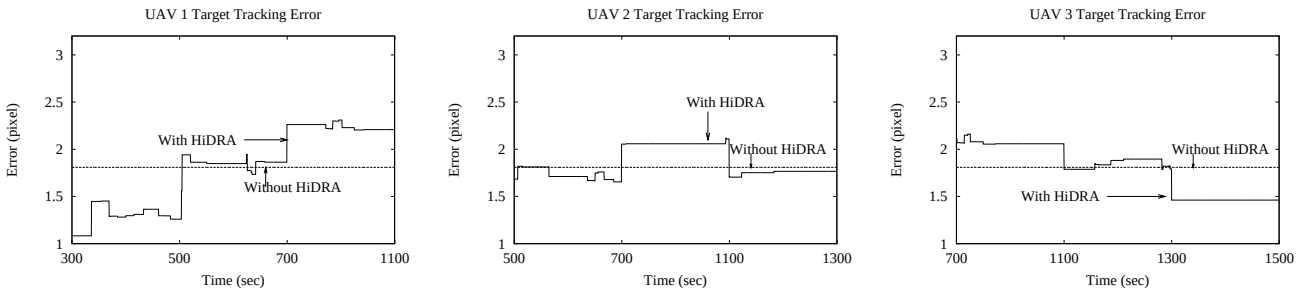
These results show how HiDRA ensures that the processor utilization of the receiver node—as well as the wireless bandwidth of the network—converges to the desired set-point within bounded time, even under fluctuating workloads. We therefore conclude that HiDRA ensures utilization of multiple system resources is maintained within the specified bounds, thereby ensuring system stability.

5.5.2 Comparison of QoS

We now compare the application QoS – (1) target-tracking precision and (2) average end-to-end delay.

Figure 12 compares the target-tracking errors that were obtained when the system was operated with and without HiDRA. Table 4 shows that during $T \in [300s, 500s]$, there was only one object of interest that was tracked by the system, and this object was tracked by UAV 1. When the system was operated without HiDRA, the static configuration of the system (shown in Table 2) assumed that there was a total 2 objects of interests being tracked by the system. As a result, the Figure 12a shows that the target tracking error during $T \in [300s, 500s]$ is lower when the system was operated with HiDRA than without it.

During $T \in [500s, 700s]$, a total of 2 objects of interest that were tracked by the system, and these objects were tracked by UAV 1 and UAV 2. This input workload is the same as the static configuration of the system. As a result, Figures 12a and 12b show that the target tracking error during $T \in [500s, 700s]$ is nearly the same when the system was operated with and without HiDRA.



(a) UAV-1 (b) UAV-2 (c) UAV-3

Figure 12: Exp 2: Comparison of Target-tracking Error

During $T \in [700s, 1100s]$, however, a total of three objects of interest were being tracked by the system, one by each UAV. This input workload is higher than the input workload under which the static configuration of the system was selected. To maintain the bandwidth utilization within specified bounds, therefore, HiDRA lowers the quality factor of the images transmitted by the UAVs to the receiver during $T \in [700s, 1100s]$. As a result, Figures 12a, 12b, and 12c show that the target tracking error during $T \in [700s, 1100s]$ is higher when the system was operated with HiDRA than without it. Similarly, the target tracking precision of the received images for the remaining time intervals can be analyzed.

These results demonstrate that HiDRA effectively maintains utilization of system resource below the specified set-points despite fluctuations in input workload by gracefully adjusting application QoS.

Number of Objects	End-to-End Delay (msec)	
	With HiDRA	Without HiDRA
0	20	20
1	60	60
2	117	117
3	160	250

Table 5: Exp 2: Comparison of End-to-End Delay

Table 5 compares the end-to-end delay when the system was operated with and without HiDRA. This table shows that when the total number of objects of interest tracked by the system was 2 or less, the end-to-end delay was the same when the system was operated with and without HiDRA. This result occurred because the static configuration of the system was selected assuming 2 objects of interest were being tracked by the system. When the number of objects tracked by the system increased to 3, however, system resource were over-utilized considerably when the system was operated without HiDRA, as compared to when the system was operated with it. As a result, when the system was operated without HiDRA, the end-to-end delay is significantly higher than when the system was operated with HiDRA.

HiDRA reacts to fluctuations in input workload by modifying application parameters such as JPEG quality factor. These adaptations ensure that system resources are not over-utilized and thus lowers average end-to-end delay.

5.6 Experiment 3 : Varying Bandwidth Availability and Constant Workload

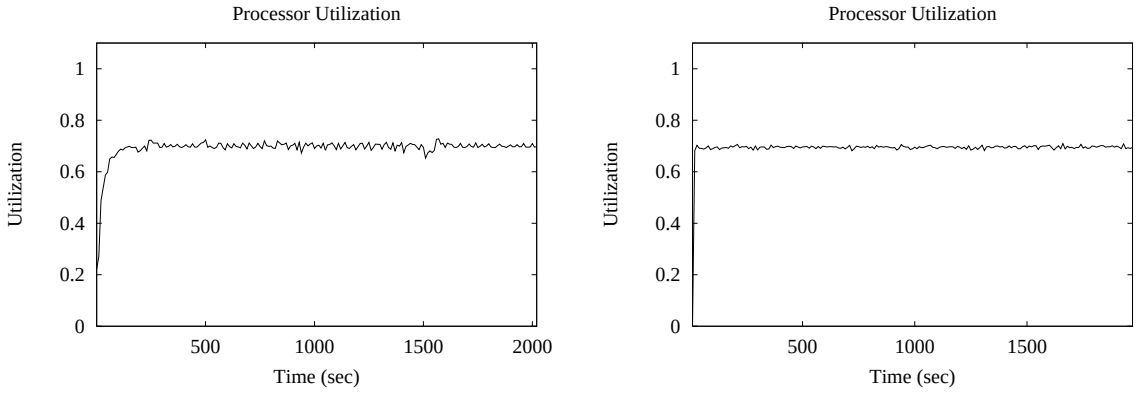
We now present the results obtained from running the experiment under varying channel capacity of the wireless network and a constant 2 number of objects of interest tracked by the system. This experiment demonstrates the adaptive resource management capabilities of HiDRA under varying resource availability and constant input workload. We normalize the channel capacity, bandwidth utilization, and bandwidth utilization set-point to the maximum channel capacity of 2Mbps. Table 6 summarizes the variation in channel capacity and bandwidth utilization set-point as a function of time. As it can be

Time (sec)	Channel Capacity (Mbps)	Bandwidth Utilization Set-Point (Mbps)	Normalized Channel Capacity	Normalized Bandwidth Utilization Set-point
0 - 480	2.0	0.7 * 2.0 = 1.4	2.0 / 2.0 = 1.0	1.4 / 2.0 = 0.7
480 - 1,480	1.0	0.7 * 1.0 = 0.7	1.0 / 2.0 = 0.5	0.7 / 2.0 = 0.35
1,480 - 2,000	2.0	0.7 * 2.0 = 1.4	2.0 / 2.0 = 1.0	1.4 / 2.0 = 0.7

Table 6: Channel Capacity and Bandwidth Utilization Set-Point as a Function of Time

seen in Table 6, the variation in the channel capacity represents a “step function”. A step function is selected because it is one of the most severe form of variation (or disturbance) that a control system can be subjected to. This experiment validates that HiDRA can maintain system stability even under such severe variation in channel capacity. Images containing objects of interests were captured by UAVs 1 and 2. In this experiment, the static configuration of the system shown in Table 2 was used when the system was operated without HiDRA.

5.6.1 Comparison of Resource Utilization

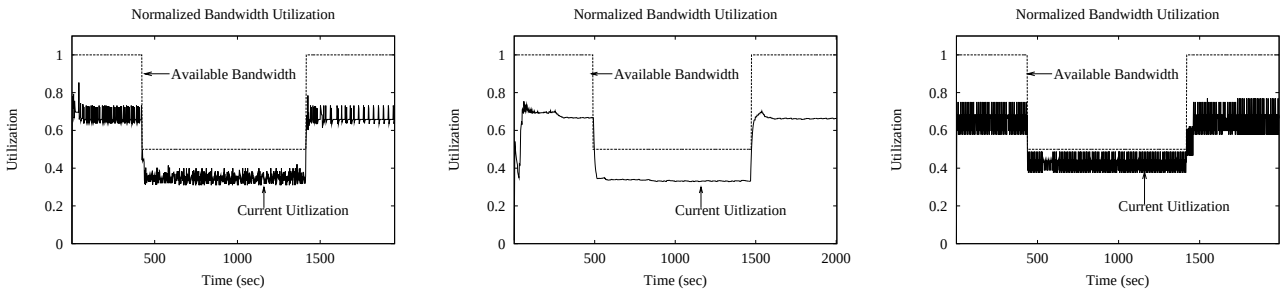


(a) Processor Utilization with HiDRA

(b) Processor Utilization without HiDRA

Figure 13: Exp 4: Comparison of Processor Utilization

Figures 13 and 14 compare the processor utilization at the receiver node and the normalized bandwidth utilization when the system was operated with and without HiDRA. The output of the bandwidth utilization monitor, shown in Figure 14b, was processed with a Kalman filter and used by the bandwidth controller as the current bandwidth utilization. From Figures 13a



(a) Normalized Bandwidth Utilization with HiDRA (processed using a Kalman Filter)

(b) Normalized Bandwidth Utilization with HiDRA

(c) Normalized Bandwidth Utilization without HiDRA

Figure 14: Exp 3: Comparison of Normalized Bandwidth Utilization

and 13b it can be seen that under this experimental scenario, processor utilization is equal to the set-point of 0.7 when the system was operated both with and without HiDRA.

Figure 14c shows that when the system was operated without HiDRA, the normalized bandwidth utilization during $T \in [0s, 480s]$ and $T \in [1480s, 2000s]$ was 0.7, which is equal to the set-point. During $T \in [480s, 1480s]$ the normalized bandwidth utilization was 0.5, which is equal to the normalized channel capacity and significantly greater than the normalized set-point of 0.35. From Figures 14a and 14b, however, it can be seen that when the system was operated with HiDRA, the normalized bandwidth utilization converged to the normalized utilization set-point even under varying channel capacity. HiDRA achieved this behavior by lowering the quality factor of the images in response to fluctuations in network bandwidth.

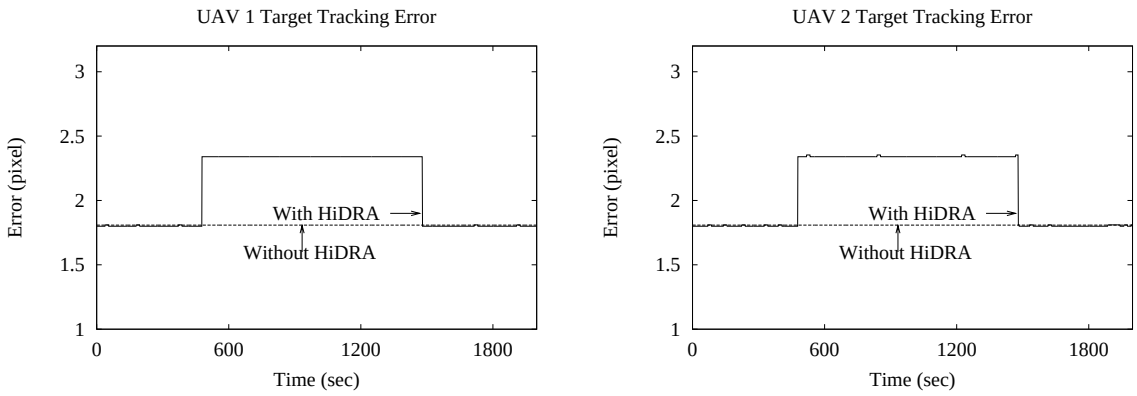
These results show that HiDRA ensures the wireless bandwidth utilization converges to the desired set-point within bounded time, even under varying network bandwidth availability. We therefore conclude that HiDRA ensures system resource utilization is maintained within the specified bounds, thereby ensuring system stability.

5.6.2 Comparison of QoS

We now compare the application QoS, which includes (1) target-tracking precision and (2) average end-to-end delay.

Figure 15 compares the target-tracking error that was obtained when the system operated with and without HiDRA. As shown in Table 6, during $T \in [0s, 480s]$ and $T \in [1, 480s, 2000s]$ the channel capacity of the wireless network was 2 Mbps, which is the resource availability under which the static configuration of the system was selected. As a result, Figures 15a and 15b show that the target tracking error during during $T \in [0s, 480s]$ and $T \in [1, 480s, 2000s]$ is nearly the same when the system was operated with HiDRA and without HiDRA.

During $T \in [480s, 1480s]$, however, the channel capacity of the wireless network was 1 Mbps. Within this time interval, the wireless bandwidth resource availability is half the wireless bandwidth resource availability under which the static configuration of the system was selected. To maintain the bandwidth utilization within specified bounds, HiDRA lowers the quality factor of the images transmitted by the UAVs to the receiver during $T \in [480s, 1480s]$. As a result, Figures 15a and 15b show that the target tracking error during $T \in [480s, 1480s]$ was higher when the system was operated with HiDRA than when the system was operated without it. These results demonstrate that HiDRA effectively maintains utilization of system resource



(a) UAV-1 (b) UAV-2

Figure 15: Exp 3: Comparison of Target-tracking Error

below the specified set-points despite variations in bandwidth resource availability by gracefully adjusting application QoS.

Table 7 compares the end-to-end delay when the system was operated with and without HiDRA. This table shows that

Number of Objects	End-to-End Delay (msec)	
	With HiDRA	Without HiDRA
2	185	276

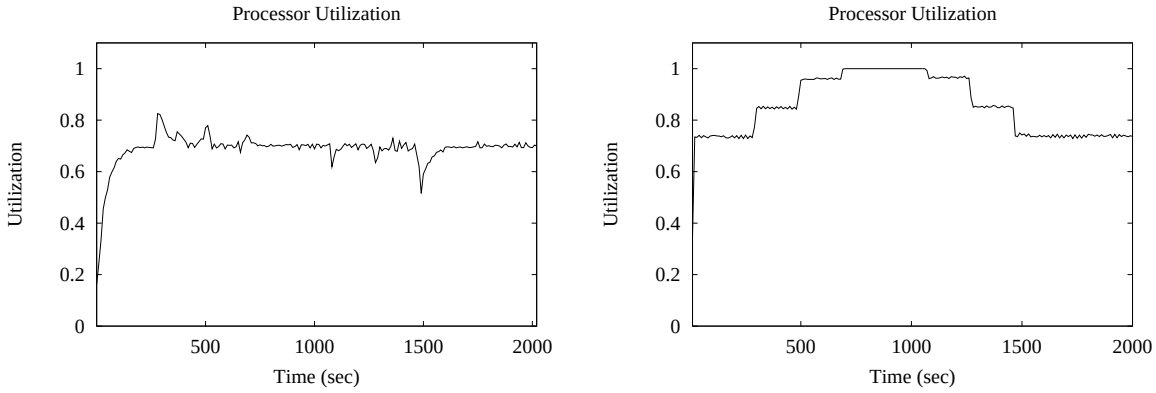
Table 7: Exp 3: Comparison of End-to-End Delay

end-to-end delay was much lower when the system was operated with HiDRA than without it. When the system was operated without HiDRA, during $T \in [480s, 1480s]$, the utilization of the wireless network bandwidth is equal to its channel capacity, which increased packet loss, retransmission delays, and in turn network transmission delay. This behavior accounts for the increase in the average end-to-end delay because the static configuration of the system was selected assuming 2 objects of interest were being tracked by the system and a constant channel capacity of 2 Mbps. When the system was operated with HiDRA, however, HiDRA reacts to variations in channel capacity by modifying application parameters such as JPEG quality factor. These adaptations ensure that system resources are not over-utilized and thus lowers average end-to-end delay.

5.7 Experiment 4: Varying Bandwidth Availability and Varying Workload

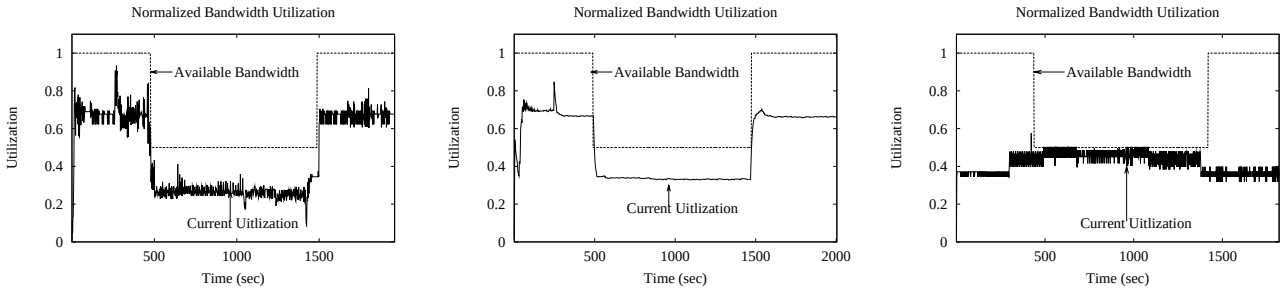
We finally present the results obtained from running the experiment under varying channel capacity of the wireless network, as well as varying number of objects of interest in the system. This experiment demonstrates the adaptive resource management capabilities of HiDRA under varying resource availability and fluctuating input workload. We, once again, normalize the channel capacity, bandwidth utilization, and bandwidth utilization set-point to the maximum channel capacity of 2Mbps. Table 4 summarizes the number of objects of interests that were tracked as a function of time. Table 6 summarizes the variation of channel capacity as a function of time. In this experiment, when the system was operated without HiDRA, the static configuration of the system shown in Table 2 was used.

5.7.1 Comparison of Resource Utilization



(a) Processor Utilization with HiDRA (b) Processor Utilization without HiDRA

Figure 16: Exp 4: Comparison of Processor Utilization



(a) Normalized Bandwidth Utilization with HiDRA (processed using a Kalman Filter) (b) Normalized Bandwidth Utilization with HiDRA (c) Normalized Bandwidth Utilization without HiDRA

Figure 17: Exp 4: Comparison of Normalized Bandwidth Utilization

Figures 16 and 17 compare the processor utilization at the receiver node and the wireless network bandwidth utilization when the system was operated with and without HiDRA. The output of the bandwidth utilization monitor, shown in Figure 17b, was processed with a Kalman filter and used by the bandwidth controller as the current bandwidth utilization.

Figure 16 and Table 4 show that the increase in the processor utilization at $T = 300s$ is due to the presence of the first object of interest. From Figure 16a it can be seen that although the processor utilization increased above 0.7, within the next several sampling periods, HiDRA restored the processor utilization to the desired set-point of 0.7. This behavior was achieved by reducing the execution rates of data-source/receiver pair(s) deemed less important, *i.e.*, ones that captured images where objects of interest were absent. As shown in Figure 16b, when the system was operated without HiDRA, the processor utilization remained at 0.85, which is significantly higher than the utilization set-point of 0.7.

At $T = 500s$, the presence of the second object of interest was detected. As a result, Figure 16b shows that processor utilization increased to 0.95 when the system was operated without HiDRA. As shown in Figure 16a, however, the processor

utilization quickly re-converges to the set-point after a transient increase when the system was operated with HiDRA.

At $T = 700s$ the presence of the third object of interest was detected. When the system was operated without HiDRA, Figure 16b shows how the processor utilization increased to 1. As shown in Figure 16a, however, once again the processor utilization quickly re-converges to the set-point after a transient increase when the system was operated with HiDRA.

At $T = 1100s$ the total number of objects currently being tracked by the system reduced from 3 to 2, although there was a decrease in the processor utilization, HiDRA restored the processor utilization of 0.7 by increasing the execution rate of important data-source/data sink pair(s). Similarly, HiDRA ensured that the processor utilization converges to the desired set-point for the remaining duration of the experiment.

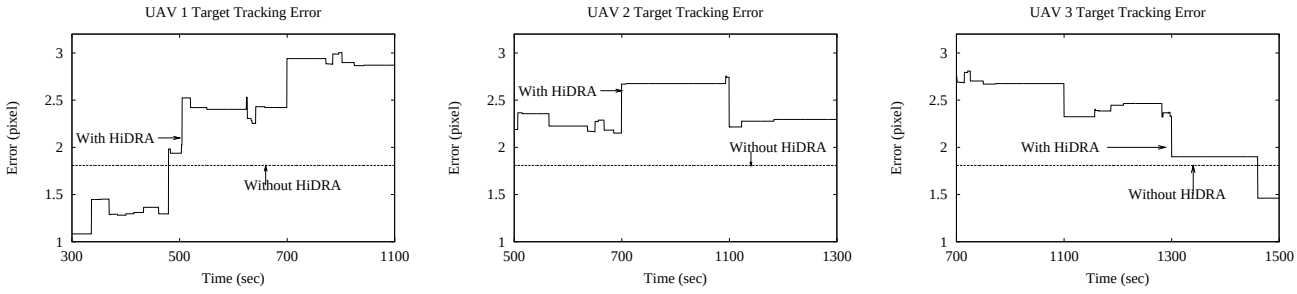
These results show that HiDRA ensures that the processor utilization of the receiver node converges to the desired set-point within bounded time, even under fluctuating workloads.

Figure 17c shows that when the system was operated without HiDRA, the normalized bandwidth utilization during $T \in [0s, 480s]$ and $T \in [1480s, 2000s]$ was below the normalized set-point of 0.7. During $T \in [480s, 1480s]$ the normalized bandwidth utilization was 0.5, which is equal to the channel capacity and significantly greater than the normalized set-point of 0.35. From Figures 17a and 17b, however, it can be seen that when the system operated with HiDRA, the normalized bandwidth utilization converged to the normalized utilization set-point even under varying channel capacity. This behavior was achieved by lowering the quality factor of the images in response to the variations in network bandwidth availability and input workload.

These results show that HiDRA ensures that the wireless bandwidth utilization converges to the desired set-point within bounded time, even under varying channel capacity and input workload. We therefore conclude that HiDRA ensures utilization of system resources is maintained within the specified bounds, even under varying resource availability and input workload, thereby ensuring system stability.

5.7.2 Comparison of QoS

We now compare the application QoS – (1) target-tracking precision, and (2) average end-to-end delay.



(a) UAV-1 (b) UAV-2 (c) UAV-3

Figure 18: Exp 4: Comparison of Target-tracking Error

Figure 18 compares the target-tracking error that were obtained when the system was operated with and without HiDRA. Table 4 shows that during $T \in [300s, 500s]$ there was only one object of interest tracked by the system using UAV 1. When

the system was operated without HiDRA, the static configuration of the system (as shown in Table 2) assumed (1) that there were a total of 2 objects of interests being tracked by the system and (2) a constant channel capacity of 2 Mbps. As a result, Figure 18a shows that the target tracking error during $T \in [300s, 480s]$ is lower when the system was operated with HiDRA than without it.

During $T \in [480s, 1480s]$, however, the channel capacity of the wireless network was 1 Mbps. During Within this time interval, the wireless network bandwidth availability was half the bandwidth availability under which the static configuration of the system was selected. To maintain the bandwidth utilization within specified bounds, therefore, HiDRA lowers the quality factor of the images transmitted by the UAVs to the receiver during $T \in [480s, 1480s]$. As a result, Figures 18a, 18b, and 18c show that the target tracking error during $T \in [480s, 1480s]$ was higher when the system was operated with HiDRA than without it.

These results demonstrate that HiDRA effectively maintains utilization of system resource below the specified set-points despite fluctuations in input workload and variations in bandwidth resource availability by gracefully adjusting application QoS.

Table 8 compares the end-to-end delay when the system was operated with and without HiDRA. This table shows that end-

Number of Objects	End-to-End Delay (msec)	
	With HiDRA	Without HiDRA
0	20	20
1	80	123
2	137	235
3	206	327

Table 8: Exp 4: Comparison of End-to-End Delay

to-end delay is much lower when the system operates with HiDRA than without it. When the system was operated without HiDRA the utilization of the wireless network bandwidth is equal to its channel capacity during $T \in [480s, 1480s]$, which resulted in increased packet loss, retransmission delays, which in turn increased network transmission delay. This behavior accounts for the increase in the average end-to-end delay because the static configuration of the system was selected assuming 2 objects of interest were being tracked by the system and a constant channel capacity of 2 Mbps. When the system was operated with HiDRA, however, it reacts to variations in channel capacity and number of objects by modifying application parameters, such as the JPEG quality factor. These adaptations ensures that system resources are not over-utilized and thus lowers average end-to-end delay.

5.8 Summary

HiDRA responds to fluctuation in input workload and the most severe form of variation in resource availability by periodically monitoring and control of resource utilization. Both our theoretical and empirical analysis assures that the utilization of system resources converge to their specified utilization set-points even if a set-point is specified as a time-varying reference

signal. However, the only assumption is that the variation in the reference signal is slower than the sampling period.

Our results show that when resources utilization increases above the desired set-point, HiDRA lowers the utilization by modifying application parameters, such as execution rates and JPEG quality factor. These adaptations ensure that (1) system resources are not over-utilized and (2) enough resources are available for important applications.

Our analysis of the results described above suggests that applying hierarchical adaptive resource management to our target tracking system helps to (1) maintain system resource utilization within specified bounds and (2) improve overall system QoS. These improvements are achieved largely due to monitoring of system resource utilization, adaptive resource provisioning, and efficient system workload management by means of HiDRA's resource monitors, hierarchical controllers, and effectors, respectively.

6 Related Work

A number of control-theoretic approaches have been applied to real-time system to overcome limitations with traditional scheduling approaches that are not suited to handle dynamic changes in resource availability and result in a rigidly scheduled system that adapts poorly to change. A survey of these techniques is presented in [10]. This section summarizes the relationship of related work on control-theoretic approaches with our research on HiDRA.

Feedback control scheduling (FCS) [11] is designed to address the challenges of applications with stringent end-to-end QoS executing in open DRE systems. These algorithms provide robust and analytical performance assurances despite uncertainties in resource availability and/or demand. FC-U and FC-M [28] and HySUCON [29] to manage the processor utilization. CAMRIT [30] applies control-theoretic approaches to ensure transmission deadlines of images over an unpredictable network link and also presents analytic performance assurance that the transmission deadlines are met.

A hierarchical control scheme that integrates resource reservation mechanisms [9, 31] with application specific QoS adaptation [32] is proposed in [21]. This control scheme features a two-tier hierarchical structure: (1) a global QoS manager that is responsible for allocating computational resources to various applications in the system and (2) application-specific QoS managers/adapters that modify application execution to use the allocated resources efficiently and improves application QoS.

Although the approaches outlined above are similar to HiDRA, these algorithms/mechanisms perform resource management of only one type of system resource, *i.e.*, either computing power *or* network bandwidth. HiDRA's contribution to control-theoretic research, therefore, is its ability to perform resource management of both network and computing resources, which is crucial for many real-world DRE systems.

One approach to manage both computing power and network bandwidth is to manage the network bandwidth utilization with CAMRIT and processor utilization with either the hierarchical control structure proposed in [21], FC-U/FC-M, or HySUCON. Unfortunately, this approach does not take into consideration the coupling between the two types of system resources and does not necessarily assure system stability.

The work of [33] utilizes task control model and fuzzy control model to enhance the QoS adaptation decision of multimedia DRE systems. The control framework established in this work, however, is still confined to single type of resource, *i.e.*, network bandwidth in a distributed visual tracking system.

As described in Section 3, HiDRA uses hierarchical feedback control loops—the processor utilization feedback loop and bandwidth utilization loop—to manage the utilization of system resources, which can be extended to handle more types of resources and end-to-end applications without significant modifications to the existing architecture. Moreover, HiDRA's feedback loops are designed so that the adaptation decisions made by one does not conflict with the decisions made by the other. As shown in Section 4.2 and Section 5, HiDRA's design results in a hierarchical control architecture that ensures system stability.

7 Concluding Remarks

Open DRE systems require end-to-end QoS enforcement from their underlying operating platforms to operate correctly. These systems often run in environments where resource availability is subject to dynamic change. To meet end-to-end QoS in these dynamic environments, DRE systems can benefit from adaptive resource management architectures that monitors system resources, performs efficient application workload management, and enables efficient resource provisioning for executing applications. Resource management mechanisms based on control-theoretic techniques are emerging as a promising solution to handle the challenges of applications with stringent end-to-end QoS executing in DRE systems. These mechanisms enable adaptive resource management capabilities in open DRE systems and adapt gracefully to fluctuation in resource availability and application resource requirement at run-time.

This paper described HiDRA, which is our hierarchical distributed resource management architecture based on control-theoretic techniques that provides adaptive resource management, such as resource monitoring and application adaptation, that are key to supporting open DRE systems. We first presented the theoretical analysis that shows how HiDRA ensures stability in our DRE system. We then evaluated the performance of HiDRA using a representative target tracking DRE system implemented using Real-time CORBA and composed of two types of system resources (*i.e.*, computational power at the receiver and wireless network bandwidth) and three applications (*i.e.*, UAV data sender/receiver pairs). Our theoretical analysis and empirical results show that HiDRA delivers efficient resource utilization by maintaining system resource utilization within specified bounds even under fluctuating work loads, thereby ensuring system stability and delivering effective QoS.

The lessons learned by applying HiDRA to our target tracking system thus far include:

- HiDRA's Control-theoretic approaches yielded in an *adaptive* resource management architecture that can gracefully handle fluctuations in resource availability and/or demand for open DRE systems.
- The formalisms presented in the paper form the foundation for a resource management framework based on control-theoretic principles that can be used to perform system stability analysis and obtain theoretical assurance about system performance.
- Developing applications in which parameters can be fine-tuned to modify the application operation and utilization of system resources helps achieve higher QoS of applications and enables HiDRA to maintain system resource utilization within desired bounds.

DRE systems are increasingly being used for mission-critical applications that operate in hostile environments. Often these systems are subjected to *QoS attacks* that aim at degrading system performance significantly. These attacks often results in loss of system resources, which in turn affects the QoS of mission critical applications operating on these systems. In future work, we will also extend HiDRA to detect such attacks early and prevent them from reducing the QoS of mission-critical applications.

References

[1] P. Sharma, J. Loyall, G. Heineman, R. Schantz, R. Shapiro, G. Duzan, Component-Based Dynamic QoS Adaptations in Distributed Real-time and Embedded Systems, in: Proc. of the Intl. Symp. on Dist. Objects and Applications (DOA'04), Agia Napa, Cyprus, 2004.

[2] D. C. Schmidt, R. Schantz, M. Masters, J. Cross, D. Sharp, L. DiPalma, Towards Adaptive and Reflective Middleware for Network-Centric Combat Systems, CrossTalk - The Journal of Defense Software Engineering.

[3] S. A. Boyer, Supervisory Control and Data Acquisition, ISA, 1993.

[4] J. D. Fernandez, A. E. Fernandez, SCADA Systems: Vulnerabilities and Remediation, J. Comput. Small Coll. 20 (4) (2005) 160–168.

[5] R. Carlson, High-Security SCADA LDRD Final Report, Tech. rep., Advanced Information and Control Systems Department, Sandia National Laboratories, Albuquerque, New Mexico, USA (April 2002).

[6] CORPORATE Computer Science and Telecommunications Board, Keeping the U.S. Computer Industry Competitive: Systems Integration, National Academy Press, Washington, DC, USA, 1992.

[7] S. E. Institute, Ultra-Large-Scale Systems: Software Challenge of the Future, Tech. rep., Pittsburgh, PA, USA (Jun 2006).

[8] J. Lehoczky, L. Sha, Y. Ding, The Rate Monotonic Scheduling Algorithm: Exact Characterization and Average Case Behavior, in: Proceedings of the 10th IEEE Real-time Systems Symposium (RTSS 1989), IEEE Computer Society Press, 1989, pp. 166–171.

[9] T. Cucinotta, L. Palopoli, L. Marzario, G. Lipari, L. Abeni, Adaptive Reservations in a Linux Environment, in: IEEE Real-Time and Embedded Technology and Applications Symposium, 2004, pp. 238–245.

[10] T. F. Abdelzaher, J. Stankovic, C. Lu, R. Zhang, Y. Lu, Feedback Performance Control in Software Services, IEEE: Control Systems 23 (3).

[11] C. Lu, J. A. Stankovic, S. H. Son, G. Tao, Feedback Control Real-Time Scheduling: Framework, Modeling, and Algorithms, Real-Time Syst. 23 (1-2) (2002) 85–126.

[12] D. C. Schmidt, D. L. Levine, S. Mungee, The Design and Performance of Real-time Object Request Brokers, Computer Communications 21 (4) (1998) 294–324.

[13] Object Management Group, Real-time CORBA Specification, OMG Document formal/02-08-02 Edition (Aug. 2002).

[14] J. P. Loyall, R. E. Schantz, D. Corman, J. L. Paunicka, S. Fernandez, A Distributed Real-Time Embedded Application for Surveillance, Detection, and Tracking of Time Critical Targets, in: IEEE Real-Time and Embedded Technology and Applications Symposium, 2005, pp. 88–97.

[15] D. Corman, WSOA-Weapon Systems Open Architecture Demonstration-Using Emerging Open System Architecture Standards to Enable Innovative Techniques for Time Critical Target (TCT) Prosecution, in: Proceedings of the 20th IEEE/AIAA Digital Avionics Systems Conference (DASC), 2001.

[16] D. Corman, J. Gossett, D. Noll, Experiences in a Distributed Real-time Avionics Domain, in: Proceedings of the International Symposium on Object-Oriented Real-time Distributed Computing (ISORC), IEEE/IFIP, Washington, D.C., 2002.

[17] IEEE Std 802.11-1997 Information Technology – Telecommunications and Information Exchange Between Systems – Local and Metropolitan Area Networks – Specific requirements – Part 11: Wireless Lan Medium Access Control (MAC) And Physical Layer (PHY) Specifications, IEEE Computer Society, 345 E. 47th St, New York, NY 10017, USA, 1997.

[18] G. Holland, N. Vaidya, P. Bahl, A Rate-Adaptive MAC Protocol for Multi-Hop Wireless Networks, in: MobiCom '01: Proceedings of the 7th Annual International Conference on Mobile Computing and Networking, ACM Press, New York, NY, USA, 2001, pp. 236–251.

[19] G. K. Wallace, The JPEG Still Image Compression Standard, Communications of the ACM 34 (4) (1991) 30–44.

[20] G. Welch, G. Bishop, An Introduction to the Kalman Filter, Tech. rep., Chapel Hill, NC, USA (1995).

[21] L. Abeni, G. Buttazzo, Hierarchical QoS Management for Time Sensitive Applications, in: RTAS '01: Proceedings of the Seventh Real-Time Technology and Applications Symposium (RTAS '01), IEEE Computer Society, Washington, DC, USA, 2001, p. 63.

[22] G. F. Franklin, J. D. Powell, M. Workman, Digital Control of Dynamic Systems, 3rd edition, Addition-Wesley, 1997.

[23] B. White, J. L. et al, An Integrated Experimental Environment for Distributed Systems and Networks, in: Proceedings of the Fifth Symposium on Operating Systems Design and Implementation, USENIX Association, Boston, MA, 2002, pp. 255–270.

[24] F. Dellaert, C. Thorpe, Robust Car Tracking Using Kalman Filtering and Bayesian Templates, in: Conference on Intelligent Transportation Systems, 1997.

[25] S. H. Shah, K. Chen, K. Nahrstedt, Dynamic Bandwidth Management for Single-hop Ad Hoc Wireless Networks, Mob. Netw. Appl. 10 (1-2) (2005) 199–217.

[26] D. Mills, The Network Time Protocol, in: RFC 1059, Network Working Group, 1988.

[27] G. Bianchi, Performance Analysis of the IEEE 802.11 Distributed Coordination Function, IEEE Journal on Selected Areas in Communications 18 (1-2) (2000) 535–547.

[28] C. Lu, X. Wang, C. Gill, Feedback Control Real-time Scheduling in ORB Middleware, in: Proceedings of the 9th IEEE Real-time and Embedded Technology and Applications Symposium (RTAS), IEEE, Washington, DC, 2003.

[29] X. Koutsoukos, R. Tekumalla, B. Natarajan, C. Lu, Hybrid Supervisory Control of Real-time Systems, in: 11th IEEE Real-time and Embedded Technology and Applications Symposium, San Francisco, California, 2005.

[30] X. Wang, H.-M. Huang, V. Subramonian, C. Lu, C. Gill, CAMRIT: Control-based Adaptive Middleware for Real-time Image Transmission, in: Proc. of the 10th IEEE Real-time and Embedded Tech. and Applications Symp. (RTAS), Toronto, Canada, 2004.

[31] G. Lipari, G. Lamastra, L. Abeni, Task Synchronization in Reservation-Based Real-Time Systems, IEEE Trans. Computers 53 (12) (2004) 1591–1601.

[32] S. Brandt, G. Nutt, T. Berk, J. Mankovich, A Dynamic Quality of Service Middleware Agent for Mediating Application Resource Usage, in: RTSS '98: Proceedings of the IEEE Real-Time Systems Symposium, IEEE Computer Society, Washington, DC, USA, 1998, p. 307.

[33] B. Li, K. Nahrstedt, A Control-based Middleware Framework for QoS Adaptations, IEEE Journal on Selected Areas in Communications 17 (9) (1999) 1632–1650.