CAMRIT: Control-based Adaptive Middleware for Real-time Image Transmission

Xiaorui Wang, Huang-Ming Huang, Venkita Subramonian, Chenyang Lu, Christopher Gill
Department of Computer Science and Engineering
Washington University, St.Louis,MO
{wang,hh1,venkita,lu,cdgill}@cse.wustl.edu

Abstract

Real-time image transmission is crucial to an emerging class of distributed embedded systems operating in open network environments. Examples include avionics mission re-planning over Link-16, security systems based on wireless camera networks, and online collaboration using camera phones. Meeting image transmission deadlines is a key challenge in such systems due to unpredictable network conditions. In this paper, we present CAMRIT, a Control-based Adaptive Middleware framework for Real-time Image Transmission in distributed real-time embedded systems. CAMRIT features a distributed feedback control loop that meets image transmission deadlines by dynamically adjusting the quality of image tiles. We derive an analytic model that captures the dynamics of a distributed middleware architecture. A control theoretic methodology is applied to systematically design a control algorithm with analytic assurance of system stability and performance, despite uncertainties in network bandwidth. Experimental results demonstrate that CAMRIT can provide robust real-time guarantees for a representative application scenario.

1. Introduction

Recent years have seen rapid growth of a new generation of Distributed Real-time Embedded (DRE) systems that integrate digital imaging and wireless networking technology. For example, security systems can perform automatic intruder detection through real-time fusion of images from multiple cameras connected through a wireless network [19]. Similarly, to facilitate avionics mission replanning, personnel on multiple aircraft need to collaborate by exchanging target imagery and display annotations over Link-16 wireless networks [7]. Real-time image transmission is also important in new services on camera-equipped mobile phones (*e.g.*, online collaboration and security monitoring) that rely on "live" image transmission over cellular networks.

These embedded applications are different from traditional imaging applications (*e.g.*, online photo albums) in two ways. First, image transmission in these embedded systems is subject to stringent timing constraints. Second, although higher image quality usually improves system utility, these next-generation embedded applications can tolerate some degree of degradation in image quality. For example, late image delivery can be disastrous in a security system because it may result in a delayed security alarm. On the other hand, distributed event detection algorithms usually can maintain a desired probability of event detection even if input images are not perfect. Similarly, meeting deadlines is much more important in avionics mission replanning than perfect image quality, as long as key target features are still distinguishable.

These emerging embedded applications are also different from traditional embedded systems, such as process control in factories. While traditional embedded systems usually operate over closed and predictable networks, these new types of embedded systems need to perform image transmission across *open* and *unpredictable* networks. For example, Link-16 is widely used for tactical communication between military aircraft, but has very limited effective bandwidth (*e.g.*, roughly 30 to 340 Kbps divided among all aircraft communicating with a common JTIDS terminal [20]). Furthermore, network bandwidth may vary significantly during a mission due to changes in weather, terrain, and communication distance [7]. These bandwidth-constrained and unpredictable networks make real-time image transmission a challenging task.

We have developed *CAMRIT, a Control-based Adaptive Middleware for Real-time Image Transmission*. The CAMRIT project has made three main contributions to the state of the art in performance control for DRE systems.

1. *Adaptive Architecture*: We present a novel middleware architecture for *feedback-based* adaptive management of image transmission. Our architecture features a distributed feedback control loop that supports fine-

grained control over the progress of image transmission by dynamically adjusting the quality factor of image tiles.

2. *Control Modeling*: We derive an analytic model that captures the dynamics of a distributed middleware architecture. Control analysis shows that CAMRIT can assure system stability and transmission latencies under a wide range of available network bandwidth.

3. *Middleware Implementation*: CAMRIT has been implemented as a middleware service based on the TAO [4] real-time CORBA object request broker so it is portable across heterogeneous platforms. Experimental results on a characteristic testbed demonstrate that CAMRIT can provide robust real-time assurance under representative application scenarios.

2. Middleware Architecture

The primary goal of CAMRIT is to complete transmitting an image from a server node to a client node within a user specified deadline. At the same time, CAMRIT aims to maximize image quality because a higher quality image usually has higher utility to the application. This requirement excludes trivial solutions such as always sending an image at the lowest quality.

To achieve both goals despite an unpredictable network, CAMRIT employs a feedback control loop that dynamically adjusts image quality based on performance feedback. CAMRIT exploits existing image compression standards that support flexible image quality. For example, the widely adopted JPEG [23] standard provides a user-specified parameter called the *quality factor* which can be any integer from 1 to 100. Since a lower quality factor leads to a smaller image size after compression, the quality factor parameter provides a knob for controlling the time it takes to transmit an image. However, JPEG only supports a *single* quality factor for a whole image. This is insufficient for our feedback control loop, which needs to adjust the quality factor of an image dynamically during its transmission. To support such fine-grained adaptation, CAMRIT splits each image into tiles, each of which may be compressed with a separate quality factor.

CAMRIT is designed as a *middleware service* for real-time CORBA. All the tasks in CAMRIT are managed and scheduled according to the Rate Monotonic Scheduling (RMS) algorithm [14] using the Kokyu [10] dispatcher within the TAO Real Time Event Channel [11]. We note in passing that the CAMRIT architecture may also be instantiated as individual software or be integrated with other middleware.

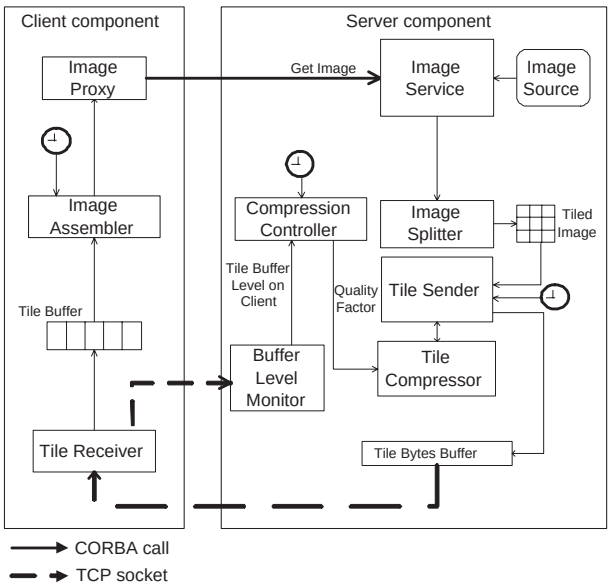

Figure 1. Overview of the CAMRIT Architecture

2.1. Service Interface

An application interacts with CAMRIT's ImageTransmissionService interface, specified in CORBA IDL. The following parameters are passed to the service:

- *image_id*: An identifier (*e.g.*, an image file name) for the requested image.
- *deadline*: The relative deadline for delivery of the image.
- *num_tile*: The number of tiles into which the image is divided. This parameter allows the application to specify the granularity of control of the image quality, with a trade-off of increased overhead for finer granularity.
- *quality_range*: The defined range of acceptable image quality. This parameter allows configuration of application-specific image quality constraints.

The CAMRIT service implementation serves to hide properties of the underlying network from the the application, particularly the variations in available bandwidth over a network, and delivers the image within the specified deadline. Figure 1 shows the major components of the CAMRIT architecture. We first describe the mechanisms responsible for requesting and transmitting an image, and then discuss the feedback loop for controlling transmission latency.

2.2. Image Transmission

The CAMRIT middleware architecture is made up of client and server components, each on a separate endsystem. The *Image Proxy* object in the CAMRIT client component

provides the service interface to the application. When it receives a request for an image, this object makes a CORBA call to the *Image Service* object on the server. This CORBA call has the same parameters as the service interface. A *one-way* CORBA call is used to avoid blocking the client thread that executes the call, because transmitting a large image over a bandwidth-constrained network may take a long time.

The Image Service object is implemented as a CORBA servant in the server component, and is advertised to the outside world. When it receives the CORBA call from the client, the Image Service object retrieves the requested image (*e.g.*, from an image repository or a camera), and calls the *Image Splitter* object to split the retrieved image into a specified number of tiles. Each tile is compressed by the *Tile Compressor* object according to the current quality factor, which is periodically updated by the *Controller* object described in Section 2.4. The *Tile Sender* object then sends each compressed tile, as a byte stream through a TCP socket, to the client component.

The Tile Sender and Tile Compressor are executed by a periodic task. In each invocation, the Tile Sender fills the TCP buffer by sending image tiles to a TCP socket. The sending socket is set to NON_BLOCKING mode so that the kernel will inform the application layer through an EWOULDBLOCK error from the *send* system call if the TCP buffer is full. Note the sender may push a fraction of a tile to fill the TCP buffer. The pseudo-code for this periodic task is shown below. Tile_Bytes_Buffer is a buffer on the server that is used to hold the bytes of a tile (or fraction of a tile) to be sent.

```
Tile_Sender :: handle_timeout () {
  while (1) {
    ret_code = send bytes in Tile_Bytes_Buffer to socket;
    if ( ret_code == EWOULDBLOCK)
        exit the current invocation ;
    Compress next tile with current quality factor ;
    Create a header for the tile ;
    Append the new compressed tile to Tile_Bytes_Buffer ;
  }
}
```

The *Tile Receiver* object on the client reads the byte stream from the socket. The boundaries between tiles are indicated in the tile header that precedes each tile. After it receives a whole tile, the Tile Receiver object enqueues the tile into a buffer that holds received but still compressed tiles.

The *Image Assembler* is executed as a periodic task. The first instance of this task is released when the first tile of the image is inserted into the tile buffer. In every invocation, it dequeues and decompresses a tile from the tile buffer if it is not empty. When all the tiles of an image have been decompressed, it assembles them back into a whole image and notifies the Image Proxy, which then returns a handle (*e.g.*, the memory address) for the decompressed image to the application.

2.3. Selection of Task Periods

The period of the Tile Sender task is chosen such that the TCP buffer never goes empty while an image is being transmitted to the client. Specifically, if B is the TCP buffer size and b_{max} is the maximum bandwidth of the network, the period of the Tile Sender is set to no higher than $\frac{B}{b_{max}}$. This guarantees that the TCP layer in the kernel has enough bytes of data in the TCP buffer to send before the next invocation of the sending task, and hence the network bandwidth is fully utilized during the transmission of an image.

CAMRIT guarantees image deadlines by achieving the following properties. First, the tile buffer on the client always contains at least one tile during the transmission of an image. This is achieved by a feedback control loop described in the next subsection. Second, every invocation of the Image Assembler task is completed before the end of its period. This property is guaranteed by ensuring that the CPU utilization of the client end-system remains below the schedulable utilization bound of the scheduling algorithm used by RT-CORBA. Finally, the period p of the Image Assembler is selected to meet the end-to-end image deadline, as follows. When the first two properties are satisfied, each invocation of the Image Assembler task decompresses one tile by the the end of its period. Suppose the first tile of an image is inserted into the tile buffer t_1 sec after the image request is sent to the server. The first tile is decompressed by $t_1 + p$, and the i^{th} tile is decompressed by $t_1 + ip$. Therefore, the period must satisfy the following condition in order to guarantee the whole image is received and decompressed by the deadline:

$$t_1 + p * num\_tile \leq deadline$$

Hence, the upper bound for the Tile Assembler period is:

$$p \leq \frac{deadline - t_1}{num\_tile} \quad (1)$$

2.4. Feedback Control Loop

As described in the last subsection, CAMRIT must maintain a tile buffer level of at least one tile during the transmission of an image. However, while the Image Assembler dequeues tiles from the tile buffer at a constant rate, the rate at which tiles are inserted into the tile buffer (called the *tile enqueue rate*) depends on the network bandwidth and the size of compressed tiles. To deal with the unpredictable network, we designed a feedback control loop to maintain a specified buffer level (the set point) by periodically adjusting the quality factor of the remaining tiles that are yet to

be transmitted. The feedback control loop is composed of a *Buffer Level Monitor*, a *Controller*, and the Tile Compressor described earlier, which serves as an actuator in the control loop.

Each time the Tile Receiver on the client reads a chunk of data from the socket (*i.e.*, completes a `read()` call), it sends the current tile buffer level to the Buffer Level Monitor on the server. Note that the reported buffer level includes the fraction of the tile that is currently being received by the client. For example, if the tile buffer currently contains 3 tiles, and the Tile Receiver has received the first 2KB of another tile of size 5KB, the current buffer level is $3 + 2/5 = 3.4$. The Buffer Level Monitor makes this information available to the Controller. The use of fractional buffer levels as feedback improves control performance because it gives a more precise representation of the buffer level than would integer values.

The Controller periodically re-computes the quality factor of the remaining tiles based on the current tile buffer level. The new quality factor is then used by the Tile Compressor to compress the remaining tiles that are sent in the following sampling period. Clearly, the Controller is critical to the performance of CAMRIT.

3. Dynamic Model

Modeling the dynamics of the controlled system is crucial for control design. It is also a key challenge in complex distributed middleware systems, whose dynamics are not understood as well as those of many physical control systems. In this section we establish a dynamic model for a characteristic real-time image transmission system controlled by our feedback control loop.

3.1. Controlled System Model

As described in the Section 2, the controlled variable in our feedback control system is the tile buffer level on the client, and the manipulated variable is the quality factor used by the server to compress tiles. We first introduce some essential notation:

- T: the sampling period of the feedback control loop.
- $l(k)$: the tile buffer level at the k^{th} sampling point (kT sec after the system starts). As described in Section 2, $l(k)$ may include a fraction of a tile.
- l_s: the set point, *i.e.*, the desired tile buffer level.
- r: the constant rate (*i.e.*, the frequency) at which tiles are dequeued from the tile buffer by the Image Assembler. It is equal to the inverse of the period of the Image Assembler task, $r = 1/p$.
- $b(k)$: the network bandwidth in the k^{th} sampling period, $[kT,\ (k+1)T)$. The value of $b(k)$ is unknown *a priori* in an unpredictable network environment, but its range $[b_{min}, b_{max}]$ is usually known.
- s: the size of an uncompressed tile. This is known and fixed for a given image and number of tiles.
- $s(q)$: the average size of a tile compressed with a quality factor q.
- $q(k)$: the quality factor computed by the controller at the k^{th} sampling point.

In each sampling period, rT tiles are dequeued from the tile buffer. Supposing $n(k)$ tiles are transmitted and inserted to the tile buffer in the k^{th} sampling period, we then have this equation:

$$l(k+1) = l(k) + n(k) - rT \tag{2}$$

$n(k)$ depends on the size of compressed tiles and the network bandwidth. The size of a compressed tile is a nonlinear function of the quality factor used to compress it. For the purpose of control design, we linearize this function such that

$$s(q) \quad = \quad \frac{sq}{g} \tag{3}$$

where g is a gain that can be estimated through linearization in the steady-state operation region of the system. The details of the linearization are presented in Section 3.2.

In our control design, we assume $b(k) = b$ where b is the nominal bandwidth. Although we design the controller based on b, the controller is tuned such that it remains stable as long as the bandwidth stays within the range $[b_{min}, b_{max}]$.

If we ignore control delay, we get a simple first-order model for the controlled system:

$$l(k+1) = l(k) + \frac{bTg}{sq(k)} - rT \tag{4}$$

Unfortunately, this model is inaccurate because control delay plays a major role in the dynamics of our distributed middleware. This control delay can be modeled as the end-to-end latency from the moment when the Tile Receiver sends out the sampled buffer level from the client, to the moment when this new quality factor starts to have an effect on the client tile buffer. We can divide this control delay into the sampling delay from the client to the server and the actuation delay from the server back to the client. Considering the fact that the communication load from the client to the server is signifcantly lower than the opposite direction during the image transmission, we approximate the control

delay $t_d(k)$ in our system with the actuation delay, the time interval starting from the moment when the controller on the server outputs the new quality factor $q(k)$.

The control delay is due to residual data in the TCP buffer and the Tile Byte Buffer on the server. When the controller outputs a new quality factor, these buffers still contain tiles compressed with the *old* quality factor, $q(k\text{-}1)$. Hence the system will continue to transmit and enqueue those *old* tiles to the tile buffer on the client until all the data in the TCP buffer and the Tile Byte Buffer have been transmitted to the server.

Let $s_t(k)$ and $s_b(k)$ denote the amount of data in the TCP buffer and the Tile Byte Buffer, respectively. The control delay is then

$$t_d(k) = \frac{s_t(k) + s_b(k)}{b} \tag{5}$$

To calculate the control delay, we need to estimate $s_t(k)$ and $s_a(k)$. First, we consider $s_t(k)$. Suppose the TCP buffer size is B, and the period of the Tile Sender task is p_s. The TCP buffer is full (*i.e.*, contains B bits of data) at the end of each invocation of the Tile Sender task. During each period of the Tile Sender, bp_s bits of data are transmitted from the TCP buffer. Therefore, the lower bound for the amount of data that the TCP buffer may hold is $B - bp_s$ bits. Since $s_t(k)$ depends on the specific time when the controller outputs $q(k)$, we approximate $s_t(k)$ with the average of its upper bound and lower bound for our control design:

$$s_t = B - \frac{bp_s}{2} \tag{6}$$

As Section 3.2 describes, the Tile Byte Buffer holds the fraction of a compressed tile that cannot fit into the TCP buffer. On average, this buffer contains half of a tile compressed with quality factor $q(k\text{-}1)$ at the beginning of the k^{th} sampling period. We approximate $s_b(k)$ with its average value, based on (3):

$$s_b(k) = \frac{sq(k-1)}{2g} \tag{7}$$

As Figure 2 illustrates, if we choose a sampling period $T > t_d(k)$, the tiles placed into the tile buffer in the first $t_d(k)$ secconds of the k^{th} sampling period are compressed with quality factor $q(k\text{-}1)$, and the tiles placed there in the remaining part of the sampling period are compressed with quality factor $q(k)$. Therefore, a more accurate model that considers the control delay is

$$l(k+1) = l(k) + \frac{bt_d(k)g}{sq(k-1)} + \frac{b(T - t_d(k))g}{sq(k)} - rT \tag{8}$$

Note that the second to last term in (8) is non-linear because it includes both $q(k)$ and $t_d(k)$, which is a function

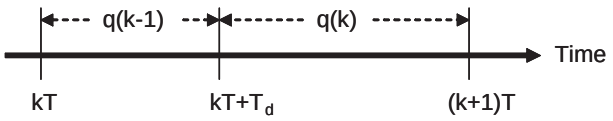

Figure 2. Quality Factors of Tiles Received in the k^{th} Sampling Period

of $q(k\text{-}1)$ (see (5) and (7)). Since the quality factor does not change significantly in a steady state, we can linearize this model by replacing the $q(k\text{-}1)$ in this term with $q(k)$. Finally, let $u(k) = 1/q(k)$ be the control input. We then have an approximate linear model of the controlled system:

$$l(k+1) = l(k) + Au(k) + Cu(k-1) + D \tag{9}$$

where $A = \frac{(bT - s_t)g}{s}$, $C = \frac{s_t g}{s}$ and $D = -rT$.

When control delay is zero, this model is the same as the first-order model in (4). However, when control delay is comparable to the sampling period, the coefficient of the second order term $q(k\text{-}1)$ becomes significant, and the second-order model is needed to capture the system dynamics.

3.2. Tile Size and Quality Factor

We now describe how to estimate the gain g. We first compare the size of the compressed sample image $s(q)$ with each quality factor q, and plot the *inverse of the compression ratio* $a(q) = s/s(q)$ as a function of the inverse of the quality factor $u = 1/q$, which is the control input. For an example aerial image (called Image 0[1] in this paper) its resulting profile of the relationship between those parameters is a non-linear curve. We linearize $a(u)$ in the operational region of the system in steady state, in the following three steps.

1. Given the deadline d for transmission of an image, the rate r of the Image Assembler is calculated using (1). In steady state, tiles are transmitted from the server to the client at the same rate as r, to maintain a constant tile buffer level.

2. We then use the following equation to calculate the range of $a(u)$, $[a_{min}, a_{max}]$, that can satisfy the tile transmission rate r in steady state based on the range of possible network bandwidth $[b_{min}, b_{max}]$.

$$\frac{ba(u)}{s} = r \tag{10}$$

[1] All images used in this paper are available at $http://deuce.doc.wustl.edu/FCS\_nORB/CAMRIT$.

3. Finally, we perform linear regression on the segment of function $a(u)$ where $a_{min} \leq a(u) \leq a_{max}$. The slope of the linear regression is the estimated g.

When an image request is submitted, CAMRIT uses the estimation process above to derive g, based on the specified deadline and the function $a(u)$ from the profiling results for a representative image. While function $a(u)$ may differ for different images, the difference is small for images in a similar application domain (*e.g.*, landscape images taken from airplanes). Furthermore, the feedback control loop can be designed to tolerate a range of variations in g.

As an example, we now show how to estimate g based on hypothetical but plausible system settings, and using the measured profile for Image 0. The key parameters for this example are as follows:

- Image: 640×640 pixels; divided into 64 tiles; each uncompressed tile size s = 18.75 KB.
- Deadline: d = 200 sec.
- Bandwidth: [4 Kbps, 8 Kbps]. The top of this bandwidth range approximates the maximum data rate of a single link at the lowest Link-16 network capacity of 28.8 Kbps [24], with time slots divided among links to 3 aircraft collaborating with a common JTIDS terminal on the Command-and-Control aircraft (C2); we assume a minimum network bandwidth of half the maximum; we use the midpoint of the resulting range, b = 6 Kbps, for our control design.

The rate of the Image Assembler (also the steady-state tile transmission rate) is computed using (1). CAMRIT uses 95% of the actual deadline to give some leeway to the transmission, and t_1 is estimated based on the nominal bandwidth and the tile size with the initial quality factor (68 in this example). The resultant $r = 0.34$ tile/sec. According to (10), in order to allow the bandwidth variation from 4 Kbps to 8 Kbps, the range for the inverse of compression ratio needs to be [6.38, 12.75]. Linearization is then performed in this range for $a(q)$ as shown in Figure 3. The slope of the linear regression is $g = 341.34$. The linear regression fits well (with an $R^2 = 94.87\%$) with the original function in this operation region.

4. Control Design and Analysis

We now apply linear control theory to design the controller based on the controlled system model described in Section 3. The z-transform of the controlled system model (9) is:

$$L(z) = z^{-1}L(z) + Az^{-1}U(z) + Cz^{-2}U(z) + \frac{Dz}{z-1} \tag{11}$$

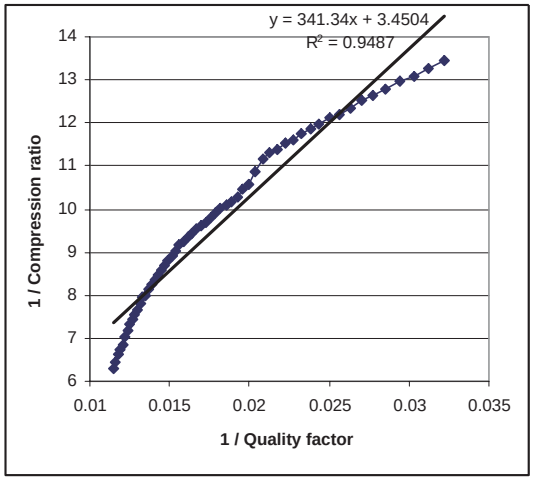

Figure 3. Linearization of $a(u)$

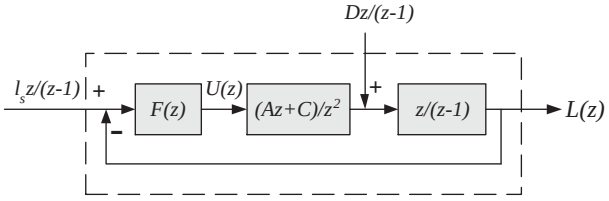

Figure 4. Block Diagram of Closed-Loop System

A block diagram of the closed-loop system is shown in Figure 4. The system has two inputs: the set point of the tile buffer level and a disturbance input $\frac{Dz}{z-1}$ that represents the dequeuing of tiles from the tile buffer by the Image Assembler.

Letting $F(z)$ be the transfer function of the controller, we can derive the closed-loop transfer function in response to the reference input and disturbance, respectively:

$$H_s(z) = \frac{(Az+C)F(z)}{(z-1)z + (Az+C)F(z)}$$
$$H_d(z) = \frac{z^2}{(z-1)z + (Az+C)F(z)} \tag{12}$$

Therefore, the close-loop response to both inputs is

$$L(z) = H_s(z)\frac{z}{z-1}l_s + H_d(z)\frac{z}{z-1}D \tag{13}$$

To achieve stability and zero steady state error, we design a *Proportional-Integral (PI)* controller for our system:

$$F(z) = \frac{K_1(z-K_2)}{z-1} \tag{14}$$

The time-domain form of (14) is:

$$u(k) = u(k-1) + K_1 e(k) - K_1 K_2 e(k-1) \tag{15}$$

where K_1 and K_2 are control parameters that can be analytically tuned to guarantee system stability and zero steady state error using standard control design methods.

We first apply the control design to our example application integrated with the CAMRIT framework. The sampling period is T=10 sec. The TCP buffer size is B = 4 KB. The period of the Tile Sender task is set to 2.67 sec to fully utilize network bandwidth. The other parameters (including g) are the same as for the example given in Section 3.2. From (5), the control delay in the k^{th} sampling period is $T_d = 4 + q(k-1)/27.31$ sec. For example, the control delay is 5.8 sec when $q(k\text{-}1)$=50. Compared to a sampling period of 10 sec, the control delay clearly plays a significant role in the system dynamics. From (9), the parameters of the controlled system model are A=81.922; C=54.614; D=-3.420.

Using the Root-Locus method, we select our control parameters as K_1=0.0068 and K_2=0.9. The corresponding closed-loop poles are $0.278 \pm 0.547i$ and 0.887. Since all the poles are in the unit circle, the system is stable. From the final value theorem [8], we have proved that the closed-loop system achieves zero steady state error. That is, the tile buffer level will achieve the set point in steady state: $\lim_{k\to\infty} l(k) = l_s$. If the set point is set to $l_s \geq 1$, the tile buffer will remain non-empty in steady state, and hence the image transmission deadline will be met. Furthermore, by substituting different bandwidths into the system model, we can prove that the system can maintain stability and zero steady-state error with the same control parameters as long as the network bandwidth remains within the range [4Kbps, 8Kbps]. A detailed analysis is not given here due to space limitations: interested readers are referred to a standard control textbook [8].

In summary, pseudo code for the control algorithm implemented in CAMRIT is as follows:

```
Controller (l_s, K_1, K_2) {
    ℓ = current tile buffer level ;
    e = ℓ_s − ℓ;
    u = u + K_1 ∗ e − K_1 ∗ K_2 ∗ e_prev;
    e_prev = e;
    q = 1/u;
    /∗ enforce constraints on acceptable quality factor ∗/
    /∗ default range is [1,100] ∗/
    if (q < q_min) q = q_min;
    if (q > q_max) q = q_max;
    UpdateQF(q);
    /∗ updated q will be used by the Tile Compressor ∗/
}
```

5. Experimental Evaluation

5.1. WSOA Scenario

The Weapons System Open Architecture (WSOA) [7] program had a primary objective to provide internet-like connectivity, over Link-16, between legacy embedded mission systems in fighter aircraft and off-board Command and Control (C2) systems. This capability was designed to support time-sensitive mission re-planning and redirection of attack nodes, as necessary based on situational events, even if a different mission was already underway.

The following high-level sequence of interactions between the C2 and fighter aircraft constitutes a representative WSOA scenario: 1) The C2 node receives information about a higher priority time critical target and requests a planning session with attack nodes by sending an alert; 2) Upon receiving an alert, a fighter aircraft begins downloading a Virtual Target Folder (VTF). The VTF contains several thumbnail-sized images, each representing a virtual target; 3) Once the fighter receives a folder, the pilot can select a thumbnail image in the folder via a graphical display; 4) A request is then made to the C2 for a larger version of the selected image. The experiments presented in this paper emulate step 4, which is the most time critical part of the application.

5.2. Experimental Platform

Our experimental configuration consists of two machines each running RedHat Linux 9.0 with the 2.4.20 kernel. The C2 aircraft and the fighter were simulated using a 2.53GHz Pentium IV and a 400MHz Pentium II, respectively. The following software was used to perform the experiments:

- **A**CE 5.3.5 + TAO 1.3.5 : TAO is a widely used open-source real-time CORBA standard object request broker [4]. TAO also provides a Real Time Event Channel [18] that is integrated with the Kokyu dispatching and scheduling framework [10]. This integrated middleware framework allow us to (re)schedule rates of invocation of application components, while maintaining deadline-feasible scheduling of critical operations.
- **I**mageMagick++ 5.5.7 : We used this library to compress and decompress images.
- **S**haper 1.3 for Linux : Shaper is a linux script for traffic shaping. It allows us to specify the maximum bandwidth for network connection between two hosts.

We used Shaper to control the bandwidth between the two machines, *i.e.*, to simulate the performance of a Link-16 or other bandwidth-constrainted network over an underlying Ethernet connection. We set the range of bandwidth allowed by the traffic shapers to approximate the effective bandwidth of a plausible Link-16 configuration, *e.g.*, with a maximum network capacity of 28.8 Kbps [24], divided between the client and server. Taking into account the slotted nature of Link-16 communication channels and other Link-16 parameters, and the characteristics of the traffic shaper

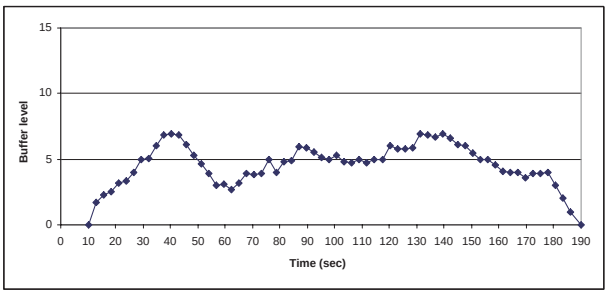

Figure 5. Tile Buffer Levels During Typical Transmission of Image 1

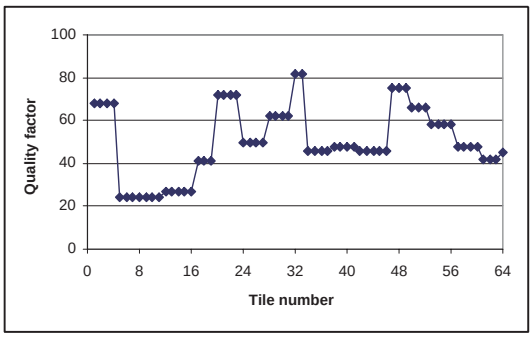

Figure 6. Quality Factors During Typical Transmission of Image 1

we used, we chose a maximum bandwidth of 8 Kbps for our experiments.

5.3. Experimental Parameters

Our experiments used the same parameters as the examples in sections 3.2 and 4. To test CAMRIT's ability to handle different images, our experiments used two other aerial images than Image 0, whose profile was used to tune the control parameters. These two images are called Image 1 and Image 2 respectively. The number of tiles for each image is set to 64 for our experiments, to achieve a reasonable balance between control granularity and overhead.

The set point for the tile buffer level was $l_s = 5$ in our experiments. Note that there is a tradeoff in the choice of the set point. If the set point is too high, the quality factor for tiles transmitted in the first several sampling periods will be unnecessarily low because system has to fill an initially empty buffer with more tiles (with lower quality factors) before it reaches a steady state. On the other hand, if the set point is too low, a fluctuation in the network bandwidth may cause the buffer level drop to zero.

5.4. Experimental Results

CAMRIT uses (10) to calculate $q(0)$ based on its deadline, the nominal bandwidth (6 Kbps), and the profiled image quality function for Image 0. The resulting initial quality factor is $q(0) = 68$ in all of the following experiments. While $q(0)$ provides a reasonable initial value for the control input, that initial value is usually not correct for meeting the deadline because the actual bandwidth may differ from the nominal one.

The tile buffer level and quality factors during a typical transmission of Image 1 over a 6 Kbps network are shown in Figures 5 and 6, respectively. The buffer level is recorded by the Image Assembler before everytime it attempts to dequeue a tile. Time 0 in Figure 5 represents the time instant when the image request is sent to the server. The tile buffer is initially empty until the first tile is inserted at around 11 sec. This 11 sec delay includes the time it takes CAMRIT to send the image request to the server, divide the image into tiles on the server, and transmitting the first tile. Since the buffer level is low initially, CAMRIT reduces the quality factor from 68 to about 20 so that the buffer level rises to 5 tiles (the set point) in about 20 sec. The buffer level remains close to 5 tiles until the last image is transmitted to the client near the end of the run. The transmission of the whole image is completed at time 190 sec. This is consistent with our expectation because 190 sec (95% of the deadline) is used to compute the rate of the Image Assembler. Both tile buffer level and quality factor have some oscillation due to system noise. For example, the sizes of different tiles may be different (corresponding to different g values in our model) even if they are compressed using a same quality factor. However, despite the noise the tile buffer is always above 2.5 throughout the transmission. This is important because CAMRIT can guarantee an image transmission deadline is met as long as the tile buffer always contains at least one tile.

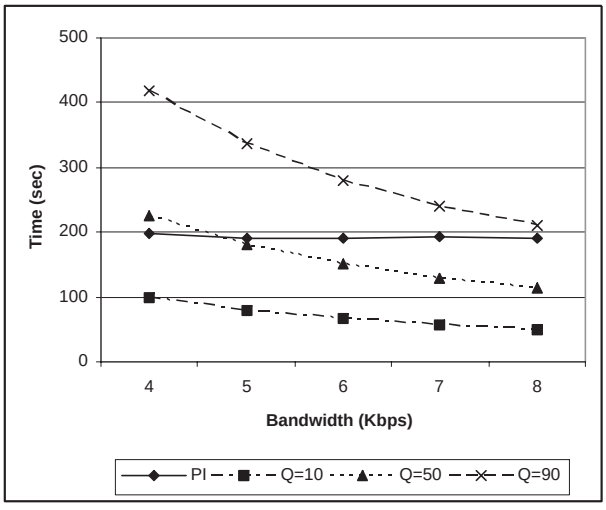

Figure 7. Transmission Delay under Different Network Bandwidth

The primary goal of CAMRIT is to meet image transmission deadlines. Figure 7 shows the transmission delay

of Image 1 under different bandwidths. The transmission delay of CAMRIT (with the feedback loop) is measured through experiments. Each data point of CAMRIT in Figure 7 is the mean of 10 repeated runs. The standard deviation of each data point is within 2.62 sec. The transmission delay results of Image2 are not shown because they are almost identical to those of Image 1. For comparison purposes, we also plot the estimated transmission delays for Image 1 when a fixed quality factor (10, 50, or 90) is used in each run. The transmission delay for an image with a fixed quality factor is estimated by dividing its total (compressed) tile size by the actual network bandwidth[2].

We can see that the transmission delays for images with fixed quality factors vary significantly as the network bandwidth changes. This result confirms the difficulty in selecting a proper quality factor *a priori* when the network bandwidth is unpredictable. A chosen quality factor may be unnecessarily low when transmission completes much earlier than the deadline, or too high causing a deadline miss.

In contrast, the transmission delay under CAMRIT remains close to 190 sec (95% of the original deadline) as the network bandwidth varies from 4 Kbps to 8 Kbps, and every run meets the deadline of 200 sec. The robust real-time performance is attributed to the feedback control loop that effectively maintains the desired buffer level despite the variation in network bandwidth.

The secondary goal of CAMRIT is to improve the image quality. CAMRIT accomplishes this goal by 1) fully utilizing the network bandwidth and 2) completing the transmission of an image close to the deadline (as shown in Figure 7). The combination of both properties means that CAMRIT sends close-to-maximum amounts of data for a requested image, which generally corresponds to a higher image quality.

Figure 8 shows the average quality factors of both images when they are transmitted by CAMRIT under different network bandwidths. Each data point is the mean of 10 repeated runs. The standard deviations are also shown. With CAMRIT the average quality factor improves as more network bandwidth becomes available. This result determines that CAMRIT can automatically adapt to network bandwidth variations by adjusting the quality factor.

6. Related Work

CAMRIT was originally motivated by the WSOA program [7]. The WSOA resource management approach used heuristics for adaptation that do not provide *a priori* performance analysis, but are evaluated empirically [9]. In sharp contrast, CAMRIT was modeled and designed from the start based on a rigorous control theoretic approach. Therefore,

[2]This estimation is slightly lower than the actual delay because it ignores the overhead of protocol headers.

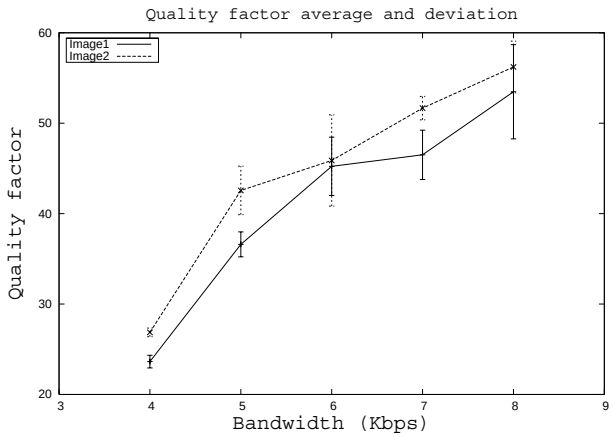

Figure 8. Average Quality Factor under Different Network Bandwidth

CAMRIT can provide the kind of robust and *analytic* performance guarantees that are crucial in mission-critical real-time systems.

Control theoretic approaches have been applied to a number of computing and networking systems. A survey of feedback performance control for software is presented in [1]. A number of feedback-based real-time processor scheduling algorithms (*e.g.*, [2] [15] [22] [5]) have also been presented in the literature. These algorithms only controlled the allocation of the computing resource on a single node, and do not address transmission delays in distributed systems. Although feedback control real-time scheduling has been extended to handle distributed systems [17] [21], communication delays are not the focus of existing algorithms. Control theory has also been applied to design and analyze network routers (*e.g.*, [12] [6]). CAMRIT is different from these network solutions in that it aims to support real-time image transmission over existing networks through adaptation at the endsystems.

Li and Nahrstedt developed Agilos, a distributed visual tracking system based on control-theoretic adaptation [13]. Agilos embodied a feedback loop that achieved desired image transmission rates through several adaptation mechanisms including image compression. However, Agilos did not control the transmission delays of images. Moreover, the effect of control delay on the dynamics of distributed systems was not modeled in that project.

7. Conclusions

In this paper, we have presented the design, modeling, and analysis of CAMRIT based on a control theoretic approach. A key contribution of this work is an analytic model that captures the dynamics of a moderately complex distributed middleware architecture. CAMRIT has been successfully implemented as a CORBA-based middleware ser-

vice atop the TAO real-time ORB. Our experiments on a representative testbed demonstrate that CAMRIT can provide robust feedback control of image transmission delays across a range of available network bandwidth, by automatically adjusting image tile quality factors.

A potential extension to this work is to apply a wider range of adaptive control techniques [3] to further improve the robustness of the system under different degrees and kinds of uncertainty. Another important direction of future work is to integrate CAMRIT with feedback control real-time task scheduling [15] [16] [17] in an end-to-end performance control middleware framework for distributed embedded systems.

Acknowledgements

This research was supported, in part, by DARPA Adaptive and Reflective Middleware Systems (ARMS) program under grant NBCHC030140 and NSF under an ITR grant CCR-0325529. We would also like to thank the reviewers for their detailed feedback.

References

[1] T. Abdelzaher, J. Stankovic, C. Lu, R. Zhang, and Y. Lu. Feedback performance control in sofware services. *IEEE Control Systems*, 23(3), June 2003.

[2] L. Abeni, L. Palopoli, G. Lipari, and J. Walpole. Analysis of a reservation-based feedback scheduler. In *IEEE Real-Time Systems Symposium*, Dec. 2002.

[3] K. Astrom and B. Wittenmark. *Adaptive Control*. Addison-Wesley, 1995.

[4] Center for Distributed Object Computing. The ACE ORB (TAO). www.cs.wustl.edu/∼schmidt/TAO.html, Washington University.

[5] A. Cervin, J. Eker, B. Bernhardsson, and K.-E. Arzen. Feedback-feedforward scheduling of control tasks. *Real-Time Systems*, 23(1):25–53, July 2002.

[6] N. Christin, J. Liebeherr, and T. Abdelzaher. A quantitative assured forwarding service. In *Proceedings of IEEE INFOCOM 2002*, New York, NY, June 2002.

[7] 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)*, Oct. 2001.

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

[9] C. Gill and et al. Integrated Adaptive QoS Management in Middleware: A Case Study. In *Real-time Technology and Application Symposium (RTAS '04), Embedded Applications Track*, Toronto, Canada, May 2004.

[10] C. Gill, D. Schmidt, and R. Cytron. Multi-Paradigm Scheduling for Distributed Real-Time Embedded Computing. *IEEE Proceedings, Special Issue on Modeling and Design of Embedded Software*, 91(1), Jan. 2003.

[11] T. Harrison, D. Levine, and D. Schmidt. The Design and Performance of a Real-time CORBA Event Service. In *Proceedings of OOPSLA '97*, Atlanta, GA, Oct. 1997.

[12] C. Hollot, V. Misra, D. Towsley, , and W. Gong. A control theoretic analysis of RED. In *Proceedings of INFOCOM 2001*, Apr. 2001.

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

[14] C. Liu and J. Layland. Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment. *JACM*, 20(1):46–61, 1973.

[15] C. Lu, J. Stankovic, G. Tao, and S. Son. Feedback Control Real-Time Scheduling: Framework, Modeling, and Algorithms. *Real-Time Systems Journal*, 23(1/2):85–126, July 2002.

[16] C. Lu, X. Wang, and 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)*, Washington, DC, May 2003.

[17] C. Lu, X. Wang, and X. Koutsoukos. End-to-end utilization control in distributed real-time systems. In *International Conference on Distributed Computing Systems ICDCS 2004*, Tokyo, Japan, Mar. 2004.

[18] C. O'Ryan, D. Schmidt, and J. Noseworthy. Patterns and Performance of a CORBA Event Service for Large-scale Distributed Interactive Simulations. *International Journal of Computer Systems Science and Engineering*, 17(2), Mar. 2002.

[19] G. J. Pottie and W. J. Kaiser. Wireless integrated network sensors. *Commun. ACM*, 43(5):51–58, 2000.

[20] R. Collins. JTIDS: Joint Tactical Information Distribution System. www.rockwellcollins.com/ecat/gs/JTIDS.html.

[21] J. A. Stankovic and et al. Feedback Control Scheduling in Distributed Systems. In *The 22nd IEEE Real-Time Systems Symposium (RTSS '01)*, London UK, Dec. 2001.

[22] D. Steere and et al. A feedback-driven proportion allocator for real-rate scheduling. In *Operating Systems Design and Implementation*, pages 145–158, 1999.

[23] G. K. Wallace. The jpeg still image compression standard. *Communications of the ACM*, 34(4):30–44, Apr. 1991.

[24] W. J. Wilson. Applying layering principles to legacy systems: Link 16 as a case study. In *IEEE International Military Communications Conference (MILCOM)*, 2001.