IEEE Fourth International High-Assurance Systems Engineering Symposium (HASE'99),
IEEE Computer Society Press, Los Alamitos, CA, Nov. 17-19, 1999, pp. 207-216.

The Amaranth Framework: Probabilistic, Utility-Based Quality of Service Management for High-Assurance Computing*

Carol L. Hoover, Jeffery Hansen, Philip Koopman, and Sandeep Tamboli
Department of Electrical & Computer Engineering & The Institute for Complex Engineered Systems
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213-1890
{clh, hansen, koopman, stamboli}+@CMU.EDU

Abstract

System resource management for high-assurance applications such as the command and control of a battle group is a complex problem. These applications often require guaranteed computing services that must satisfy both hard and soft deadlines. In addition, their resource demands can vary significantly over time with bursts of high activity amidst periods of inactivity. A traditional solution has been to dedicate resources to critical application tasks and to share resources among noncritical tasks. With the increasing complexity of high-assurance applications and the need to reduce system costs, dedicating resources is not a satisfactory solution. The Amaranth Project at Carnegie Mellon is researching and developing a framework for allocating shared resources to support multiple quality of service (QoS) dimensions and to provide probabilistic assurances of service. This paper is an overview of the Amaranth framework, the current results from applying the framework, and the future research directions for the Amaranth project.

1. Introduction

High-assurance applications such as the command and control of a naval battle group require the timely allocation of resources to enable critical computing on demand. The allocation of resources to support the various mission activities of a battle group is challenging because the necessary processing and data communications of multiple surface ships and aircraft are sporadic with periods of inactivity and bursts of activity. Traditionally, system designers dedicate resources to the highly critical tasks and share other resources among less critical tasks. This solution satisfies high-assurance demands at the cost of potentially inefficient use of resources, a situation which can also result in unnecessary space and maintenance requirements. In addition, the integration of secure data and communications among tasks operating on heterogeneous, distributed platforms can be complex. Insufficient use of resources multiplies the problem.

A more cost efficient solution would be to share a set of heterogeneous resources among the various distributed application and system tasks. Using standardized communication protocols would simplify the design of tasks that can reliably communicate across distributed computing nodes. In particular, it is necessary to multiplex the network bandwidth among soft versus hard real-time tasks and across bursty versus steady-stream communications. System designers seek to configure a set of resources that have the potential to maximize the utility of application tasks while minimizing system costs.

Likewise, application designers want critical tasks to operate with high assurance and less critical tasks to execute at performance levels that match the needs of the intended users. User preferences can vary across QoS features such as timeliness, dependability, security, and application-specific performance. What is needed is an approach for managing resources across multiple QoS dimensions in a way that maximizes their value across user requests while providing some known level of guaranteed service in the event of high resource demands or resource failure.

The Amaranth solution is a user-centric framework for analytically managing resource allocations along multiple QoS dimensions and viewpoints. This paper overviews the Amaranth framework as well as the analytical techniques to support utility-based QoS management with probabilistic guarantees. The authors define the target application as well as the representative application used to validate the Amaranth approach to QoS management. They describe the Amaranth architecture and supporting theoretical foundations. The paper concludes with a discussion of the preliminary research results and future research directions.

*The Amaranth Project at Carnegie Mellon is supported by the Quorum Program of the Defense Advanced Research Projects Agency (DARPA) under agreement number N66001-97-C-8527. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of DARPA or the U.S. Government.

2. Target Application

The goal of the Amaranth research is to develop capabilities to support the underlying distributed communication and computation infrastructure that will become increasingly important in future commercial and military systems. The military units of the twenty-first century will be radically different from their predecessors. Their computer systems will support various types of computing from parallel processing embedded in the twenty-first century surface-combatant (SC-21) computing platform to distributed processing embedded in tanks such as the M2 A1 Bradley which is equipped with a missile subsystem. These systems will run a collection of software agents embedded in a communication and computing infrastructure that provides cognitive support for the interconnection of soldiers, sailors, and commanders through the "tactical internet" of the digital battlefield. As another example, the "soldier phone" applies wireless and multimedia communications to the accomplishment of battlefield objectives [2].

A typical configuration, as shown in Figure 1, would be a group of surface ships whose computing environment consists of a set of heterogeneous computing nodes with software to support mission-critical functions such as planning, communication, and missile tracking and deployment, as well as "housekeeping" operations such as the management of inventory and personnel records. The bandwidth for a ship's local network would be high, while the bandwidth among ships would be limited.

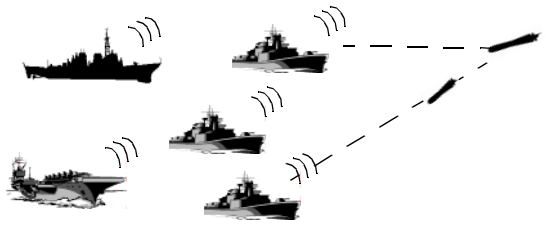

Figure 1: Battle group communication among ships with missile tracking and interception

The mix of tasks utilizing bandwidth would consist of both multimedia data streams and short, intermittent weapons control data. Both types of traffic may experience long periods of relative inactivity (e.g. minutes to hours) with short, intense periods of high activity (e.g. milliseconds or seconds). Though the bandwidth for the active periods of the multimedia stream would most likely be greater than that for the bursty weapons control communication, the control data would be more critical to the mission of the battle group and also less resilient to degradations in QoS. Many multimedia applications can perform effectively with reduced rates of frame update and can employ compression techniques to minimize bandwidth.

System designers cannot afford to size their systems for the worst case expected application mix and behavior to achieve maximum QoS for all task requests. Since many applications can function usefully at a degraded but acceptable QoS for short periods of time, system designers can reduce the system size accordingly. The "right" size system can handle the critical tasks during peak periods of activity with minimal periods of degraded service to the less critical tasks during normal operations. The system resource management policies and mechanisms should be able to adapt to variation in workload due to sudden bursts of new jobs such as incoming mission-critical alarms. They should also be able to provide a high degree of assurance that critical tasks will function properly despite fluctuations in equipment availability due to maintenance downtime and battle damage.

The Amaranth framework is a resource management strategy to enable system designers to scale their systems to minimize cost and to maximize the utility of the system to the anticipated set of users. Because we are not currently able to replicate the exact application mix of the battle group scenario described in the previous paragraphs, we have chosen to simulate representative workloads in our testbed. The representative application mix consists of a video-conference application with periods of varying activity along with workload generators to simulate the burstiness of critical missile activity. Though our approach for maximizing utility scales to multiple resources, we are currently focusing on managing the allocation of network bandwidth.

3. Related Research

Earlier work focused on managing QoS across the layers of the network protocol and on defining the meaning of QoS from source to destination nodes (end-to-end QoS). The advent of multimedia workstations and high-speed networks enabled a new class of applications demanding continuous network bandwidth to support streams of data. Critical distributed control applications requiring reliability and guaranteed bounds on message latency, along with multimedia, have demonstrated the inadequacy of best effort communications and thereby spurred research in the area of QoS management of network resources.

Campbell et al. formally define the concepts of flow and flow management in a paper about the QoS-A architecture, a layered architecture of services and mechanisms for QoS management and control of continuous media flow in multi-service networks [3]. Campbell et al. argue that meeting QoS guarantees in distributed multimedia systems is fundamentally an application-to-application (end-to-end) issue. The architectures that they review focus on QoS in the context of individual layers of a network protocol [4].

QoS management currently resides primarily in the policies and mechanisms to route packets of data. Noteworthy is Nahrstedt and Smith's QoS Broker, a model for specifying application requirements and translating these requirements into negotiated resource allocations. The QoS Broker acts as an intermediary between application processes and the OS/network protocol subsystem to communicate the application needs to the lower level services. The broker orchestrates network resources at the source and destination nodes (end points of a network of computing resources) by coordinating resource management across the communication layer boundaries within an end-point node [13].

More recently, Bashandy et al. have developed a protocol architecture to define distributed multimedia systems from the application as well as the service providers' points of view. Formally defined as finite state machines, their architecture consists of an application layer, a configuration and synchronization layer, and network layers as well as a database and computation backbone. Each layer consists of independent functional units that communicate through a standard framework of messages [1].

Current research also delineates QoS from multiple viewpoints (e.g. user or application, system, and resource) [19]. Sabata et al. outline a taxonomy for classifying QoS parameters from different viewpoints. They specify QoS as a combination of metrics and policies: metrics measure specific quantifiable attributes of the system components, and policies dictate the behavior of the system components [18]. Researchers at SRI have outlined issues in managing resources for complex distributed systems and have developed the Logical Application Stream Model (LASM) for capturing a distributed application's structure, resource requirements, and relevant end-to-end QoS parameters. They have also developed the Benefit Function (BF) model for expressing user QoS preferences and for gracefully degrading an application's QoS under certain conditions [5-6].

The Amaranth research differs from other work by emphasizing (1) QoS management across multiple QoS dimensions defined with respect to the needs of the client applications, (2) QoS contracts with probabilistic assurances that users will receive contracted resources, (3) fault monitoring to detect and predict resource failures in order to help prevent QoS contract violations due to resource downtime, and (4) monitoring of resource usage patterns and reserve capacity to preserve contracted resource allocations in the event of bursts of task requests. The goal of the Amaranth framework is to provide a flexible environment in which system designers can deploy their own QoS management policies.

4. Amaranth Terminology and Functionality

In this section, we present terminology that we used to describe the Amaranth framework and to develop or select mechanisms to communicate between applications and the Amaranth QoS management system.

QoS Dimension:

User point of view - A QoS dimension is a domain-specific feature or application requirement whose performance level can be observed or controlled. These features may be functional such as application quality (e.g. frames per second or resolution for a multimedia application) or type of security (e.g. encryption). They may relate to execution behavior such as timeliness (e.g. latency and allowed lateness) or dependability (e.g. availability and reliability). We model QoS as an n-dimensional space with utility values and resource requirements associated with each point in space.

System point of view - A QoS dimension helps to define the QoS space in which each point corresponds to a specific allocation of required resources and system services.

Application:

User point of view - An application is a type of software that can execute at differing levels of performance. The application designer specifies the levels of service for each QoS dimension. The application user specifies the mapping between the points in the QoS space and their associated utility values. The higher the assigned utility value, the higher the usefulness or importance of the point in the QoS space.

System point of view - An application is a class which, when instantiated and executed, yields utility in return for consumed resources. The application designer with the help of the system determines the resources required to achieve each point in the QoS space. The system then uses the utility values specified by the user for the points in the QoS space to map the utility of each point to the resources required to achieve the associated level of QoS.

Session:

User point of view - A session is an instantiation of an application with a contracted probability of assurance that the system will provide resources sufficient to achieve an agreed-upon level of service for the contracted interval of time. The user associates the usefulness of the session with the contracted QoS level.

System point of view - A session is an entity to which system resources have been committed for a fixed interval of time with a contracted probability of assurance that the resource allocation will not be reduced during the interval.

Session Request:

User point of view - A session request is a request to negotiate a QoS contract and to start an application.

System point of view - A session request involves the negotiation of a QoS contract and a decision of whether or not to provide contracted and guaranteed resource allocations to a session.

Contract:

User point of view - A contract is a guarantee that within a specific session the system will provide the necessary resources to enable the associated application to perform at the contracted QoS level with the contracted probabilistic level of assurance.

System point of view - A contract is a commitment to allocate the resources necessary to enable the application to perform at the contracted QoS level with the contracted probabilistic level of assurance.

QoS Violation:

User point of view - A state in which the application cannot execute at the contracted QoS level because the system has not provided the contracted service.

System point of view - A state in which the system does not provide the resources necessary to support the contracted QoS level of a session.

QoS Availability:

User point of view - The fraction of time over the duration of a session for which the application experienced no QoS violations.

System point of view - The fraction of time over the duration of a session for which the system actions resulted in no QoS violations.[1]

QoS Reliability:

User point of view - The probability $R_{\{Qs\}}(t)$ that the application has experienced no periods of QoS violation during the time interval t beginning with the start of session s.

System point of view - The probability $R_{\{Qs\}}(t)$ that the system actions resulted in no QoS violations during the time interval t starting with the beginning of the session s.[1]

We anticipate the following users to interact with an Amaranth system of resources: software application or component developers, application users or clients, and system architects. Each of these users would specify the QoS parameters which relate to their roles. As discussed below, these users would specify the QoS parameter values necessary to contract a specified level of QoS (point in the QoS space) and level of assurance for the duration of a session.

The *Application* or *Component Developer* would specify application performance levels or QoS points for relevant QoS dimensions such as application-specific, timing, dependability, and security requirements. The following are examples of application-specific and timing requirements.

- Mathematical model of task arrivals.
- Acceptable time delay between task arrivals and completions (used to calculate deadlines).
- Tolerance to lateness or a function which specifies how the value of the task degrades with respect to the time by which a deadline is missed.
- Mapping between the QoS space and resource consumption.
- Probabilistic model of resource consumption for a particular application.

The *Application User* or *Client* would provide the mapping between the QoS space and the utility or "desirability" of each point in the space.

The *System Architect* would specify the weighting or priority of each application or user registered with the system and the system size needed to guarantee minimum acceptable QoS for all critical applications in the event of the worst case load of application requests.

Following the UML use case notation, we model a user's interaction with an Amaranth system as shown in Figure 2 on the next page. For demonstration purposes, the user is a human. In practice, the user may be an application program contracting with the Amaranth system for a specified level of resource allocation.

The major functions of the Amaranth QoS management system are as follows.

1. To receive, process, and admit/reject session requests.
2. To contract, for each session request, levels of QoS and assurance that serve to maximize system goals as specified by the active policies.
3. To allocate resources according to session contracts.
4. To monitor system parameters such as the following:
   - resource usage across sessions
   - current resource usage per session
   - resource usage per session over time
   - system resource capacity instantaneously and over time.
5. To adjust the resource allocations when necessary to satisfy the session contracts while adhering to active system policies and reacting in a timely manner to resource failures and other system disturbances.
6. To monitor and notify the session planner of pending, likely, or actual resource failures and expected downtimes.
7. To enable the human user to visually and tactilely interact with the system to enter session requests and monitor session behavior.
8. To enable the human user to simulate session requests without the use of the actual resources.

In the next section, we will discuss the Amaranth system architecture and the scenario of events and component interactions when a request is made to run an application within a session of contracted resources.

[1] A system action such as degrading the resource allocation of a particular session due to resource shortages could result in a QoS violation.

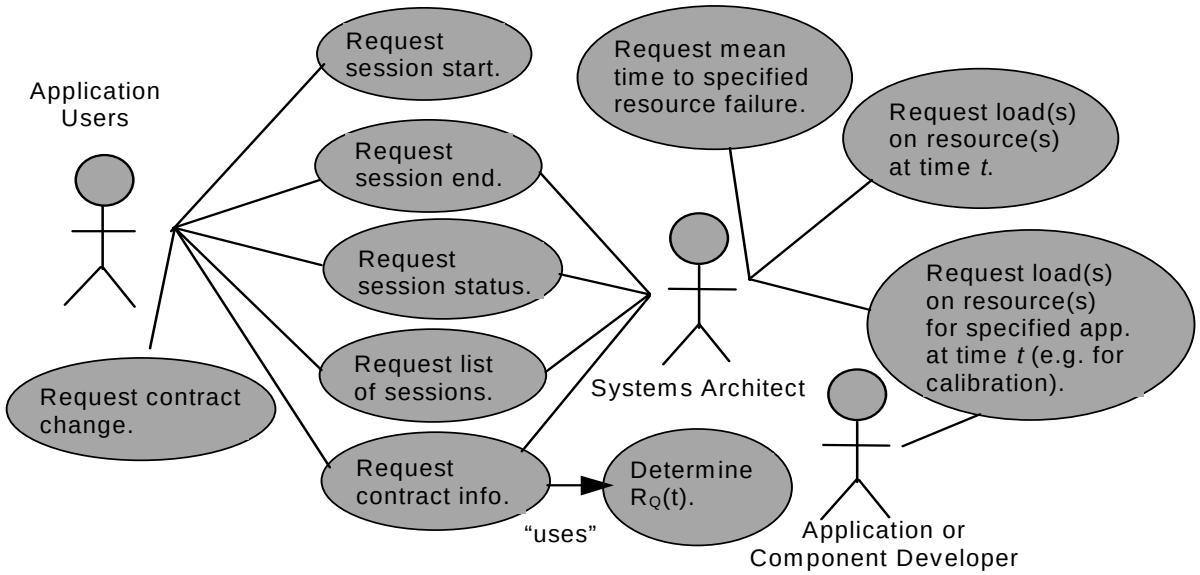

Figure 2: Amaranth use case diagram

5. Architecture and Session Management

The Amaranth architecture consists of the following primary components, each of which contributes to the Amaranth QoS management as described. Figure 3 illustrates the layered interconnections between the components.

- DMOD (Dependability Module) - Uses historical information about past computing node failures to estimate resource availability and to predict node failures.
- FRUM (Forecasted Resource Usage Module) - Uses historical trends to predict future resource usage.
- Q-RAM (QoS Resource Allocation Model) - Determines resource allocations that optimize total system utility across all application requests for resources while satisfying resource constraints.

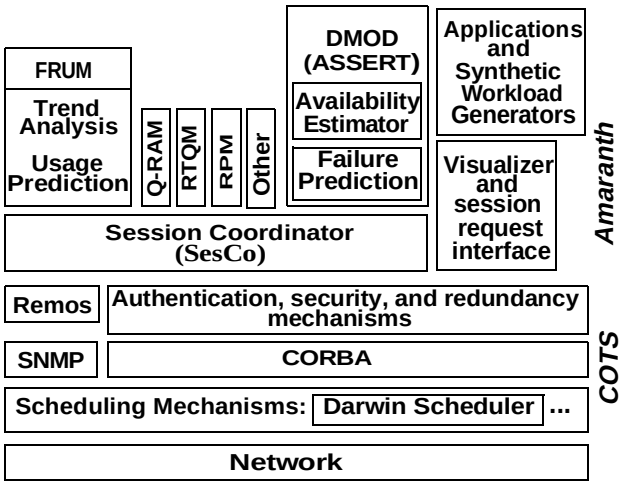

Figure 3: Amaranth architecture

- RTQM (Real-Time Queueing Modeler) - Given arrival and servicing models, estimates the distribution of tasks which would be late due to system resource constraints. Further discussion of real-time queueing appears later.
- RPM (Resource Priority Multiplexing) - Given resource usage profiles, validates assurance levels and computes scheduling parameters.
- RPM Scheduler (Resource Priority Multiplexing Scheduling Mechanism) - Implements resource priority multiplexing.
- SesCo (Session Coordinator) - Receives and processes session requests; queries the monitoring modules DMOD and FRUM as well as the policy advisor modules Q-RAM, RPM, and RTQM to make resource allocation decisions; and controls low-level mechanisms to implement resource allocations.
- Synthetic Workload Generator - Generates artificial session requests and workloads to stress the testbed.
- Visualizer - Assists users in making QoS/resource trade-offs. Visualizer currently displays the network routes utilized by one or more application sessions and enables the user to examine how they interact. In the future, the visualizer will use dynamic aggregation techniques to simplify information about large networks of more than 100 nodes into manageable maps.

The Darwin Scheduler is a resource management mechanism to support application-specific handling of network traffic and sharing of resources between cooperating traffic streams [15]. Amaranth uses Darwin to reserve network resources and to enforce the bandwidth allocations.

Timeliness is an important quality dimension for the types of applications targeted for Amaranth QoS management. For example, the system may need to provide support for real-time control systems in which the control loops have hard timing constraints as well as for voice and video packets which have constraints on latency. Real-time resource management is fairly well understood for the classical periodic task model, but Amaranth must handle tasks with more stochastic behavior and must strive to achieve high levels of resource utilization while providing as high a QoS level as possible to all the applications. The stochastic behavior of tasks makes it difficult to ensure that the system will behave properly with respect to real-time requirements. A new approach to this problem, called real-time queueing theory, provides a set of analysis techniques which offer the ability to combine the system predictability associated with hard real-time scheduling theory and the generality of stochastic task behaviors associated with queueing models [9-12].

Figure 4 demonstrates the scenario of component interactions that enable the session coordinator SesCo to process a session request. SesCo directly controls those components within the dotted line. In the event that the requesting application would like to communicate with another application running on a node outside the set of nodes managed locally, the local SesCo calls a global SesCo Coordinator to communicate with the SesCo that manages the network in which the target node is located.

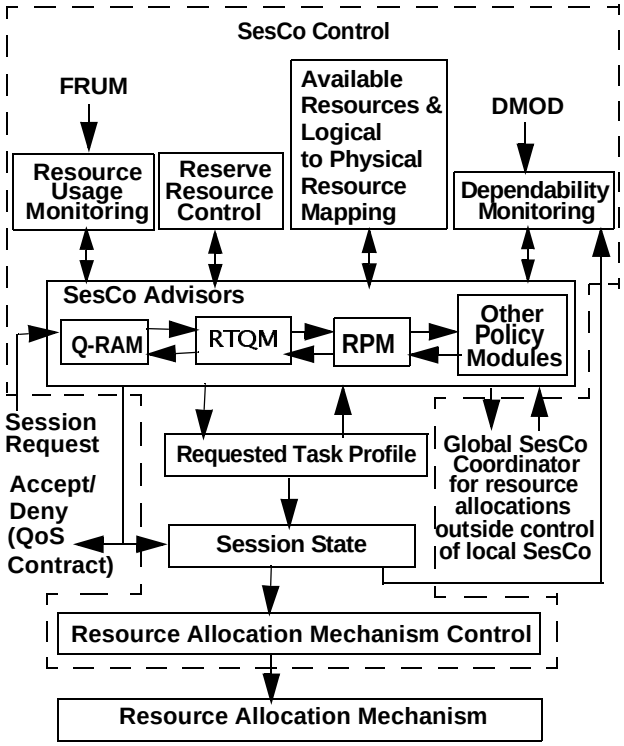

Figure 4: Component interactions to process a session request

The session coordinator component (SesCo) is responsible for receiving and processing session requests to achieve the goals of the active system policies. These policies are embodied in the system components such as Q-RAM and RPM whose respective goals are to maximize system utility across application requests and to validate the probabilistic assurance that a given session will receive the resources that it contracts. System utility is a weighted measure of the user-defined utilities associated with the levels of service (points in the QoS space) of the active applications and of the new requests. RPM analyzes the resources allocated across sessions and uses their probabilistic resource requirements to determine the probability that each session will receive its contracted resources. Using arrival, servicing, and resource availability models, RTQM provides SesCo information about the expected distribution of application tasks which will not be able to meet their deadlines.

In an Amaranth-managed system, FRUM and DMOD execute on each computing node to track usage and failure patterns. SesCo executes along with its policy advisors (currently Q-RAM, RPM, and RTQM) on the network routing nodes. The Darwin Scheduler executes on the routing nodes to enforce network reservations [15]. Visualizer runs on the user nodes, and the workload generators execute on nodes which system experimenters are using to perturb the system. The testbed in the Amaranth laboratory consists of interior routing nodes running FreeBSD (a version of Unix) and exterior user nodes executing Microsoft's Windows NT.

Component interactions across nodes are made via CORBA calls [14], while communications on the same node occur as standard procedure calls. The Visualizer is implemented as a Java application, while the other components are implemented as C++ modules. In the next three sections, we will briefly overview the Amaranth theoretical framework for utility-based resource management, probabilistic guarantees of QoS levels, and the use of reserve capacity to control session admission and resource allocation. We are currently exploring the utility and reserve capacity approaches to resource allocation independently.

6. Utility-Based Resource Management

The goal of the Amaranth QoS management is to allocate the system resources in such a way that user applications can operate effectively according to each application's requirement for assured resource allocations without having to under utilize or dedicate resources. This goal motivates the Q-RAM policy: the maximization of system utility with respect to application utility across multiple performance

levels for each QoS dimension. The feasible combinations of performance levels across the relevant QoS dimensions form an n-dimensional space of QoS points where *n* is the number of QoS dimensions. The user can specify utility or "desirability" values for individual QoS points or define a function that will assign utility values to a domain of points.

The Amaranth advisor Q-RAM determines a near optimal allocation of resources to maximize the system utility across applications without exceeding the available resources. The system allocates to each admitted application task an amount of resources that is within the minimum and maximum amounts requested by the task. The Q-RAM advisor provides polynomial-time algorithms to determine resource allocations which achieve near optimal utility for systems consisting of a single resource and multiple independent QoS dimensions or multiple independent resources and a single QoS dimension [8,16-17].

The Q-RAM advisor also includes a local search technique that provides an approximate solution to the NP-Hard problem of maximizing utility for multiple resources and multiple QoS dimensions. The Q-RAM solution is several orders of magnitude faster than the optimal dynamic programming and mixed integer programming solutions. The approximation algorithm allows the user to trade-off nearness to the optimal solution versus performance [7]. The papers referenced in this section describe in detail the Q-RAM algorithmic solutions for determining near-optimal allocation of resources across QoS dimensions.

7. Contracts and Probabilistic Guarantees

An Amaranth QoS contract is an agreement between the user and the system that the resources necessary to support the given QoS level will be provided for the duration of the session with a given probability. Ideally, the fixed QoS level would map to a fixed resource demand. But this is not the case for many real world applications such as video conferencing. Due to the effects of compression and dependent upon the amount of motion in the scene, the bandwidth requirements for a video stream may vary over time. Bandwidth requirements are highest when the camera is panning or zooming and are lowest when the camera is focused on a low motion scene such as people sitting at a conference table. Network flows having these characteristics are often called Variable Bit Rate (VBR) flows.

In a system based on hard reservations, it is difficult to achieve high resource utilization due to the worst case assumptions that must be made for sessions with VBR flows. Therefore, Amaranth provides probabilistic guarantees for sessions with VBR flows. A session's guarantee is expressed as a lower bound on the expected value, A, of the session's QoS Availability. Hard reservation $A = 1$ and best effort $A = 0$ are special cases of a probabilistic guarantee.

The RPM policy module and RPM kernel-level mechanisms implement the probabilistic guarantees in Amaranth. To enforce the probabilistic guarantees, the RPM kernel mechanisms maintain a set of priority modes. Each priority mode is a strict ordering of the managed sessions. When there is an overflow of packets in a router's queue, the system drops packets of the low priority session associated with the active priority mode. By time multiplexing through the priority modes, the QoS Availability delivered to each of the sessions can be controlled by varying the mode holding times.

The RPM policy module evaluates session requests to determine if sufficient resources are available and computes the set of mode holding time parameters needed by the kernel-level mechanisms to enforce the requested guarantee of service. One can represent the bandwidth resources required by a session as a Markov model with each state characterized by a probability and an instantaneous resource demand. Given the system priority modes and the Markov model for each session, one can form a linear program to determine the mode holding times. If the linear program is feasible, the solution will be the set of mode holding times. If it is not feasible, then the system cannot meet the requested guarantees. The system must then refuse the session request. The mathematical details of the admission algorithm will appear in a future publication.

Probabilistic guarantees can result in significant gains in resource utilization. For example, consider a 100 Mbps link over which we would like to carry a number of video conference flows. Suppose that each flow requires 1.5 Mbps 55% of the time, 2.7 Mbps 37% of the time and 5 Mbps 8% of the time. Figure 5 shows the number of flows that can be admitted on the link as a function of the QoS Unavailability (one minus the QoS Availability) for each of the flows. To provide hard guarantees on the flows, we could admit no more than 20 flows (100 Mbps/5 Mpbs). But with probabilistic guarantees, we can admit nearly double the number of flows.

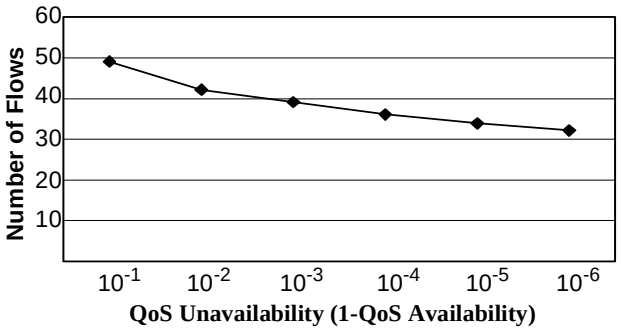

Figure 5: Number of flows with respect to Qos Unavailability

8. Reserve Capacity Approach to Admission Control and Resource Allocation

The Amaranth admission goal is to provide the maximum possible utility consistent with honoring contracted assurance probabilities for QoS violation rates. To accomplish this goal, Amaranth researchers are experimenting with holding resources in reserve to handle random, but statistically likely, bursts of task requests. Using ideas borrowed from control theory, the system attempts to maintain reserve capacity at or near a predetermined value (the *set point*). The system compares the amount of actual reserve capacity against the setpoint and, under the guidance of a control policy, determines the QoS level offered to new task admissions.

The admission control system works in two steps. First, the system admits the session if there are sufficient resources available for the application's minimum QoS requirement. Then the system uses the admission/resource allocation control parameters to determine the actual amount of resources to be allocated to the incoming task. Current experiments deal solely with network bandwidth allocations and with several policies for determining the amount to be allocated to an incoming request. Two simplistic policies are (1) to always allocate the maximum bandwidth possible (a "greedy" strategy that exploits a best effort evaluation scenario) and (2) to always allocate the minimum requested bandwidth (attempting to provide hard reservations in a worst case evaluation scenario). In all cases, the system rejects a task if there is insufficient bandwidth to admit the task even with its lowest acceptable QoS. Additionally, we have considered three common control theory policies: Bang-bang control, Proportional control, and Proportional-Integral-Differential (PID) control.

Figure 6 shows simulation results for the five control policies applied across a range of task inter-arrival times. The task interarrival times (means shown in Figure 6 by dashed lines) as well as the durations (mean of 10 minutes for all experiments) of the simulated tasks are exponentially distributed. The resource demand is uniformly distributed with minimum and maximum bandwidth demands of [1,4) Mbps and (4,7] Mbps, respectively. The reserve capacity set point is 10 Mbps, and the total available bandwidth is 100 Mbps (i.e., the desired goal is to maintain 10% reserve capacity). A mean interarrival time of 20 seconds represents a heavily loaded resource, while 60 seconds represents a lightly loaded resource. While these experimental parameters are merely exemplary values, they do provide an example of a system's response across a range of loading levels.

While many approaches are possible to minimizing QoS violation rates, the simplest approach is to permit no violations for admitted tasks and instead reject new task arrivals (resulting in a conceptually immediate QoS violation for each task that is denied admission). Figure 6 therefore shows the task rejection ratio versus the average satisfaction of admitted tasks. The average task satisfaction is computed, for example purposes, as $AverSat = \frac{(alloc - min)}{(max - min)}$, where *alloc* is the allocated bandwidth, *min* is the minimum acceptable bandwidth, and *max* is the maximum requested bandwidth.

There are five solid curves in Figure 6 corresponding to the performance of various control policies. The dashed lines represent different levels of system load (different mean task inter-arrival rates). For any given contracted assurance level, one can use the graph to indicate viable control policies and their expected average satisfactions (utility values under these simplified assumptions). This is done by considering portions of dashed lines below the task rejection ratio having the value of (1-contracted assurance level). For example, the system can achieve an assurance level of 99% with a mean arrival rate of 40 seconds by using proportional control, PID control, and hard reservations. Proportional control provides the best average satisfaction in this scenario (bang-bang control provides higher satisfaction but has too high a task rejection ratio compared to the requirement of 0.01 for a 99% assurance level).

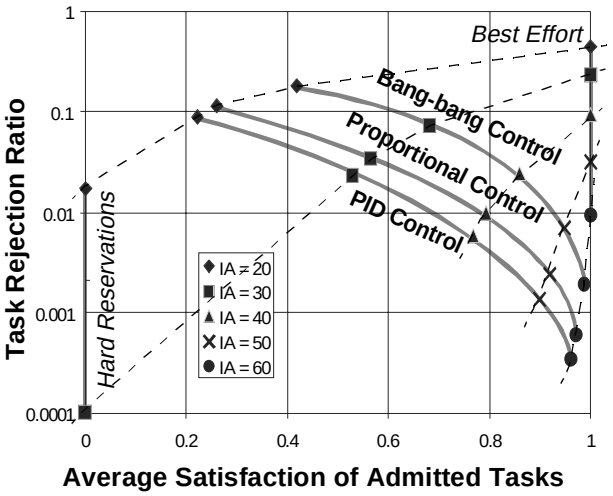

Figure 6: Task rejection ratio vs. average satisfaction of admitted tasks for interarrival times (IA) 20 through 60 seconds and across various algorithms for allocating resources.

In general, the trade-off seen in these results is the expected one of lower rejection levels coming at the price of lower satisfaction levels. However, the use of admission policies based on control theory provides intermediate trade-off points beyond those available with either hard reservations or best effort policies. In particular, the more sophisticated control policies seem able to anticipate future

problems caused by aggressive QoS allocation and to achieve lower rejection ratios for a given task arrival rate than the simplistic bang-bang control policy.

9. Results and Conclusions

Prototypes for most of the components are complete, and we have tested many of the component interactions. Those interactions involving components under development (such as RPM and RTQM) are forthcoming. The Amaranth simulator is complete, and we have used it to simulate the RPM scheduler as well as user sessions and Amaranth-controlled network allocations and routing. D-MOD is able to detect and predict node failures, and FRUM is able to track and record resource usage patterns on nodes and their connected network links. The local SesCo is functional as well as a prototype of the Q-RAM adviser module.

The theoretical models for Q-RAM, RPM, and RTQM have been developed and validated manually or via simulation. For high-assurance computing in which all application task arrivals are guaranteed at least minimum QoS, the system designer can use the application and resource models to size the system to have sufficient resources to satisfy the worst case arrival of tasks with each task receiving at least its minimum required resources. Alternatively the system designer can size the system to handle the worst case arrival of critical tasks. The critical tasks would receive at least minimum QoS, while the noncritical tasks whose total weighted utility maximizes the overall system utility without exceeding the available resources would receive resource allocations. In such high-assurance systems, QoS violations would comprise degrading previously admitted tasks to minimum QoS rather than denying admission to critical tasks.

10. Summary and Future Research

In this paper, we have discussed the Amaranth framework for providing utility-based QoS management with probabilistic guarantees that the QoS contract will be upheld. We have described our target battle group application, which requires a high degree of assurance that necessary system resources will be allocated to critical mission activities while maximizing the utilization of the system resources. We have discussed the required functionality for the Amaranth QoS management system and then presented a diagram of the Amaranth system architecture. We have explained how each system component contributes to the overall system management goals. Lastly, we have overviewed the Amaranth theory with respect to utility-based QoS management, contracts and probabilistic guarantees, and admission control.

In addition to the utility-based QoS management policy, we plan to study alternative policies for managing resources from the user as well as the system viewpoints. Likewise, we are exploring a way to integrate the utility and reserve capacity based approaches to resource allocation. We will implement and validate the RPM method for providing probabilistic guarantees and work on the integration of real-time queueing theory with the determination of optimal utility across current application requests. Later, we will design the Global SesCo Coordinator to scale to interconnected LANs and WANs with local sets of nodes and links managed by local SesCos. The Amaranth architecture will enable the system manager to employ innovative resource management policies while maintaining probabilistic QoS guarantees and dependable communications.

11. Acknowledgments

The authors would like to acknowledge the contributions of the other members of the Amaranth Project as follows: Arjun Cholkar, Lemonte' Green, Pradeep K. Khosla, Chen Lee, John Lehoczky, Don Madden, Ragunathan Rajkumar, Ying Shi, and Daniel P. Siewiorek.

12. References

[1] Bashandy, A. et al., "A Protocol Architecture for Guaranteed Quality of Service in Collaborated Multimedia Applications," In the Proceedings of the *IEEE Symposium on Application-Specific Systems and Software Engineering & Technology (ASSET'99)*, IEEE Computer Society, Los Alamitos, CA, 1999, pp. 120-127.

[2] Bittel, R. et al., "Soldier Phone: An Innovative Approach to Wireless Multimedia Communications," In the Proceedings of the *IEEE Symposium on Application-Specific Systems and Software Engineering & Technology (ASSET'99)*, IEEE Computer Society, Los Alamitos, CA, 1999, pp. 264-268.

[3] Campbell, A., G. Coulson, and D. Hutchison, "A Quality of Service Architecture," *Multimedia Systems*, Vol. 1, No. 6, April 1994, pp. 253-266.

[4] Campbell, A., C. Aurrecoechea, and L. Hauw, "A Review of QoS Architectures," *ACM Multimedia Systems Journal*, 1996.

[5] Chatterjee, S. et al., "Modeling Applications for Adaptive QoS-based Resource Management," In the Proceedings of the *Second IEEE High-Assurance Systems Engineering Workshop (HASE'97)*, 1997, pp. 194-201.

[6] Davis, M.B., and J.J. Sydir, "Resource Management for Complex Distributed Systems," Position paper in the Proceedings of the *Second Workshop on Object-Oriented Real-Time Dependable Systems (WORDS'96)*, IEEE Computer Society, Los Alamitos, CA, 1996, pp. 113-115.

[7] Lee, C., J. Lehoczky, D. Siewiorek, R. Rajkumar, and J. Hansen, "A Scalable Solution to the Multi-Resource QoS Problem," Technical Report: CMU-CS-99-144, Carnegie Mellon University, Pittsburgh, PA, May 1999.

[8] Lee, C., J. Lehoczky, R. Rajkumar, and D. Siewiorek, "On Quality of Service Optimization with Discrete QoS Options," In proceedings of the *IEEE Real-time Technology and Applications Symposium*, June 1999.

[9] Lehoczky, J.P., "Scheduling Communication Networks Carrying Real-Time Traffic," In the Proceedings of the *IEEE Symposium on Real-Time Systems*, December 1998, pp. 470-479.

[10] Lehoczky, J.P., "Real-time Queueing Network Theory," In the Proceedings of the *IEEE Real-Time Systems Symposium*, December 1997, pp. 58-67.

[11] Lehoczky, J.P., "Using Real-time Queueing Theory to Control Lateness in Real-time Systems," *Performance Evaluation Review*, June 25, 1997, pp. 158-168.

[12] Lehoczky, J.P., "Real-time Queueing Theory," In the Proceedings of the *IEEE Real-Time Systems Symposium*, December 1996, pp. 186-195.

[13] Nahrstedt, K. and J. M. Smith, "The QoS Broker," *IEEE Multimedia Magazine*, Vol. 2, No. 1, 1995, pp. 53-61.

[14] Object Management Group, *Common Object Request Broker Architecture Document*, Original version, Object Management Group, Inc., Framingham, MA, 1994.

[15] Chandra, P. et al., "Darwin: Resource Management for Value-Added Customizable Network Service," In the Proceedings of the *Sixth IEEE International Conference on Network Protocols* (ICNP'98), Austin, TX, October 1998.

[16] Rajkumar, R., C. Lee, J. Lehoczky, and D. Siewiorek, "Practical Solutions for QoS-based Resource Allocation Problems," In the *Proceedings of the 19th IEEE Real-Time Systems Symposium*, Madrid, Spain, December 1998, pp. 296-306.

[17] Rajkumar, R., C. Lee, J. Lehoczky, and D. Siewiorek, "A Resource Allocation Model for QoS Management," In the *Proceedings of the 18th IEEE Real-Time Systems Symposium*, San Francisco, CA, December 1997, pp. 298-307.

[18] Sabata, B. et al., "Taxonomy for QoS Specifications," In the Proceedings of the *Third Workshop on Object-Oriented Real-Time Dependable Systems (WORDS'97)*, Newport Beach, CA, December 1997.

[19] Sydir, J., S. Chatterjee, and B. Sabata, "Providing End-to-End QoS Assurances in CORBA-Based Systems," In the Proceedings of the *First IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC'98)*, Kyoto, Japan, April 20-22, 1998.