MODELING, DESIGN AND EVALUATION OF NETWORKING SYSTEMS AND PROTOCOLS THROUGH SIMULATION

by

DANIEL J. LACKS
M.S. University of Central Florida, 2002
B.S. University of Central Florida, 2001

A dissertation submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy
in the School of Electrical Engineering and Computer Science
in the College of Engineering and Computer Science
at the University of Central Florida
Orlando, FL

Fall Term
2007

Major Professor: Taskin Kocak

ABSTRACT

Computer modeling and simulation is a practical way to design and test a system without actually having to build it. Simulation has many benefits which apply to many different domains: it reduces costs creating different prototypes for mechanical engineers, increases the safety of chemical engineers exposed to dangerous chemicals, speeds up the time to model physical reactions, and trains soldiers to prepare for battle.

The motivation behind this work is to build a common software framework that can be used to create new networking simulators on top of an HLA-based federation for distributed simulation. The goals are to model and simulate networking architectures and protocols by developing a common underlying simulation infrastructure and to reduce the time a developer has to learn the semantics of message passing and time management to free more time for experimentation and data collection and reporting.

This is accomplished by evolving the simulation engine through three different applications that model three different types of network protocols. Computer networking is a good candidate for simulation because of the Internet's rapid growth that has spawned off the need for new protocols and algorithms and the desire for a common infrastructure to model these protocols and algorithms. One simulation, the 3DInterconnect simulator, simulates data transmitting through a hardware k-array n-cube network interconnect. Performance results show that k-array n-cube topologies can sustain higher traffic load than the currently used interconnects. The second simulator, Cluster Leader Logic Algorithm Simulator, simulates an ad-hoc wireless routing protocol that uses a data distribution methodology based on the GPS-QHRA routing protocol.

CLL algorithm can realize a maximum of 45% power savings and maximum 25% reduced queuing delay compared to GPS-QHRA. The third simulator simulates a grid resource discovery protocol for helping Virtual Organizations to find resource on a grid network to compute or store data on. Results show that worst-case 99.43% of the discovery messages are able to find a resource provider to use for computation. The simulation engine was then built to perform basic HLA operations. Results show successful HLA functions including creating, joining, and resigning from a federation, time management, and event publication and subscription.

ACKNOWLEDGEMENTS

The author wishes to acknowledge Dr. Taskin Kocak for his guidance and advice as well as commitment when opportunities took him overseas.

The author wishes to acknowledge his wife Aimee for her love, support, and caring during his pursuit of higher education.

The author wishes to acknowledge his parents and family for their support and motivation for seeking higher learning.

The author wishes to acknowledge Dr. Jacob Engel for motivation for seeking his doctoral degree as well as the work done together on the K-Array N-Cube Simulator.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER ONE: INTRODUCTION

This work evolves a custom-built simulation engine through three different simulations; some of this work has already been used to publish conference papers and a journal paper. One simulation, the 3DInterconnect simulator, simulates data transmitting through a hardware k-array n-cube network interconnect (defined on Page 1). The second simulator, Cluster Leader Logic Algorithm Simulator, simulates an ad-hoc wireless routing protocol that uses a data distribution methodology based on the GPS-QHRA routing protocol (defined on Page 3). The third simulator simulates a grid resource discovery protocol (defined on Page 4). The first two simulators have been loosely built with common software but with no compatibility to HLA or DIS or any other standard simulation architecture, however throughout each evolution of the simulation engine, the functionalities are improved and the third simulation has basic HLA operations (defined on Page 4).

Introduction to K-Array N-Cube Networks

There are many candidates in the area of interconnects that can be used to provide a communication link between processors and memories. An interconnect is a conductive connection between two or more circuits on an integrated circuit or between components on a printed circuit board. Networks such as k-array n-cubes include hypercubes, mesh and torus networks. But the uniqueness of the interconnect architecture we seek is contained by the physical constraints characterizing the line card board. Area and I/O pins are limited on the line card. Hence, the number of alternative designs that can physically and functionally fit, given those constraints, is limited. Each embedded chip has fixed and limited number of I/O pins. Therefore, a low-dimensional, packet-switched network may be a good solution.

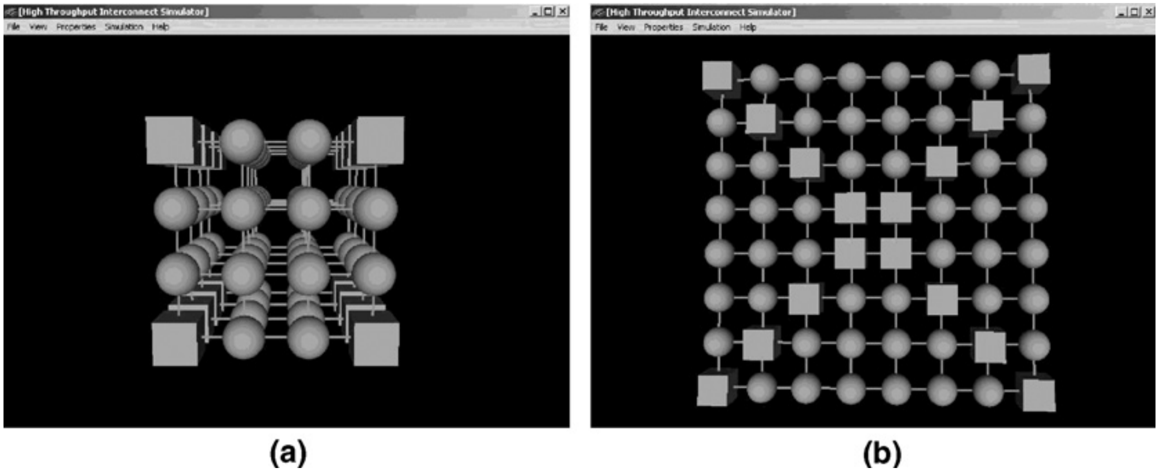

[High Throughput Interconnect Simulator]
File View Properties Simulation Help
(a)
[High Throughput Interconnect Simulator]
File View Properties Simulation Help
(b)

Figure 1 (a) 4 Array 3 Cube Interconnect (b) 8 Array 2 Cube Interconnect

Increasing line rates and deep packet processing operations place heavy strain on the memory bandwidth requirements between the line card network processing elements (PE) and memory modules (M) [25]. In order to support new services, line cards are required to perform multiple functions simultaneously. Moreover, as the network expands, lookup table entries and parameters consume more memory space to store data. As a result, the memory bandwidth requirements, which are greatly limited by the interconnection mechanism used to communicate between PEs and memories, are raised. Although new router architectures and packet processing techniques improve the performance, they still cannot keep up with network capacity growth rates in order to avoid a major traffic bottleneck.

In the heart of every line card there is a network processor unit (NPU) that performs multiple processes in order to analyze the flow of incoming packets. The nature of packet processing requires frequent read/write operations to memories distributed around the NPU. The simulator described in this work replicates the physical and functional environments by imitating different configurations in which the PEs and memories are physically located on the line card. The simulator generates random messages with explicitly random parameters such as source/destination addresses, size of messages, and arrival/departure times from PEs to memory modules and vice versa.

Introduction to Clusterhead Routing

Ad hoc networks, usually characterized as self-creating, self-organizing, and self-administering, consist of wireless devices that communicate with each other directly or indirectly through multiple hops. Such multi-hop networks, also called peer-to-peer networks, play a critical role in places where there are no preexisting infrastructure or not economical to build; such operational aspect is ideal for disorganized or hostile mobile computing environments, law enforcement, and rescue operations.

As various kinds of applications are supported over these networks, there is a need to address the quality of service (QoS) issues. QoS mainly pertains to delay and bandwidth guarantees. In order to improve QoS attributes, one can consider issues related to routing, medium access issues, mobility management, power management, and security [87]. As far as routing is concerned, there are many types of ad hoc routing protocols that have been proposed over the years [48].

A comprehensive survey of routing protocols for ad hoc networks can be found in [88]. Routing protocols have their advantages and disadvantages depending on the network characteristics and the objective of the network. These routing algorithms are distributed in nature; however, a clusterhead-based architecture helps in using some of the well-known centralized concepts that have demonstrated better performance. A clusterhead is one of the mobile nodes that assumes the responsibility of forming a cluster (each consisting of a number of ordinary nodes) and managing the radio resources in that cluster. The fulcrum of cluster based routing protocols is the clusterleader (synonymous with clusterhead). The dynamic and distributed nature of cluster leader election is critical to support the networking hierarchy created by the clusterheads.

Introduction to Grid Computing

Computational grids have been emerging as a new paradigm for solving large complex problems over the recent years [59]. Instead of having one large computer working on a problem using all the data at the same time, grid computers "eat-the-elephant" one bite at a time. The problem space and data set is divided into smaller pieces which are processed in parallel over the grid network and reassembled upon completion.

There are countless examples of how grid technology can be used for research, monitoring, reporting, data storage, modeling and simulation, or other tasks for land, sea, air, and space operations. Examples include weather and oceanographic analysis and/or reporting, networks of real-time sensors, route planning, mission planning, Live Virtual Constructive (LVC) training and simulation, cryptology, and distributed automatic test equipment to name a few.

Introduction to HLA, DIS, and the Simulation Engine

Complex modern software simulation systems, such as constructive simulators used by the military [3][4][5][6][7], share common functionality governed by their infrastructure architecture and protocols. This commonality allows them to pass messages back and forth in formats that the different simulators can interpret and (depending on the infrastructure used) can synchronize time with each other. Most of those simulators are designed to work with a variety of different infrastructures to accommodate different customers whom have funded functionality over time.

For some older or legacy systems [3][7], the infrastructure was built to accommodate a particular customer. As time marches on and new architectures become available or new requirements are imposed, it becomes too costly to change out the underlying infrastructure. So, what is typically done is for the simulation to add a bridge or translator component that allows the old infrastructure to work with the new Figure 2. The bridge acts as a translator between the old and new infrastructure and protocols.

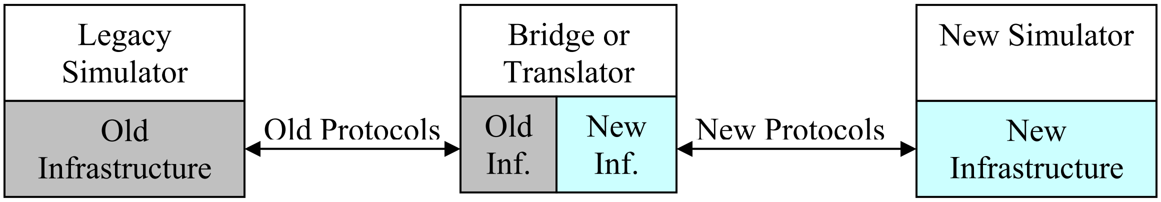


Figure 2 Legacy Simulators Connected to New Simulators via a Bridge

In order to find candidate simulation infrastructure architectures and designs to base a common software infrastructure on, a review of already existing simulation infrastructure was conducted.

Two common simulation architectures, HLA (High Level Architecture) and DIS (Distributed Interactive Simulation), address many of the issues with simulation; however, they do not address all of the issues. DIS has simpler concepts than HLA, however messages are transmitted unreliably resulting in dropped packets and time is managed in real time which means that it may be difficult for a simulator to keep up or it may be difficult or impossible to roll back time to a saved state. HLA is fairly sophisticated and has very advanced data and time management policies; but only one data model (or FOM (Federation Object Model)) can be used per simulation (or federation) and the process of bridging federations can be difficult when two or more FOMs should be shared among federations. FOMs are discussed on page 52.

The motivation behind this work is to build a common software framework that can be used to create new networking simulators for HLA-based federations. The goal is to reduce the time a developer has to learn the semantics of message passing and time management. This is accomplished by evolving the simulation engine through three different applications. The simulation engine developed is a discrete-event event-driven simulation engine [74] meaning that state changes occur at time intervals that can occur at any time. Also, the simulation engine is non-visual (no GUI), though it provides GUI helper functionality, and uses statistical generation.

Computer networking is a good candidate for simulation because of the Internet's rapid growth that has spawned off the need for new protocols and algorithms and the desire for a common simulator to model these protocols and algorithms [73]. The common simulator in [73], VINT, was built on top of ns-2 and nam [23] in a similar fashion how this work is built on HLA. Unlike ns-2 however, this simulation engine will be built to work in a distributed environment.

Main Contributions

There are four main contributions for this work:

- Show results that the k-array n-cube topologies can sustain higher traffic load than the currently used interconnects using wormhole routing.

- Show that the CLL algorithm can realize power savings and reduced queuing delay when compared to GPS-QHRA using cell fans.
- Show results that the grid resource discovery protocol discovery messages are able to find resource providers to use for computation by scoring resource providers.
- Show that the simulation engine evolved through the three simulators above has matured to the point of being HLA compatible.

CHAPTER TWO: BACKGROUND

K-Array N-Cube Interconnect Background

K-Array N-Cube Networks

A k-array n-cube network consists of N = kn nodes, where n represents the dimension of the network and k represents the number of nodes in each dimension. Figure 1 presents 8-ary 2-cube and 4-ary 3-cube networks (as captured from the interconnect simulator introduced later.) Each node in k-ary n-cube interconnect is uniquely labeled and elements of the same plane are connected together. PEs and memories are distributed throughout the interconnect in different configurations and allow each PE to use multiple memories as storage as well as data sharing with other processing elements.

Each node is connected to all of its nearest neighbors via bi-directional channels. The address/location of a node can be represented as a vector consists of two bit-vector fields [28]. Figure 3 represents the 3D-mesh interconnect architecture, which is based on a 2-ary 3-cube network, that is extended in the x-direction. The 3D-mesh interconnect is a packet-based multiple path interconnect that allows network packets to be shared by different processing elements (PE) and memory modules (M) on the network line card. Memories are distributed around processing elements, such as traffic manager, QoS co-processor or classification processor, to allow data sharing among modules and direct processor memory storage. If a link goes down, not only should the fault be limited to the link, the additional links from the intermediate nodes should ensure the connectivity continues.

Communications

Processors and memories communicate by using message-passing mechanisms. Each message is transmitted independently. Each message is partitioned into smaller data segments, also called flits, which contain the maximum amount of data (in bits) that can be transmitted in one cycle from one node to another. Each cycle another flit of the same message is transmitted. Flits of the same message follow one another in a pipeline manner. Therefore, a message is also referred to as a worm since the movement of the message within the interconnect resembles a worm movement. Virtual channels (VCs) allow worms to be stored within a node if all of the output ports of that node are busy transferring other messages. This technique prevents worm transmission failures by holding a worm within a node until one of its ports becomes available.

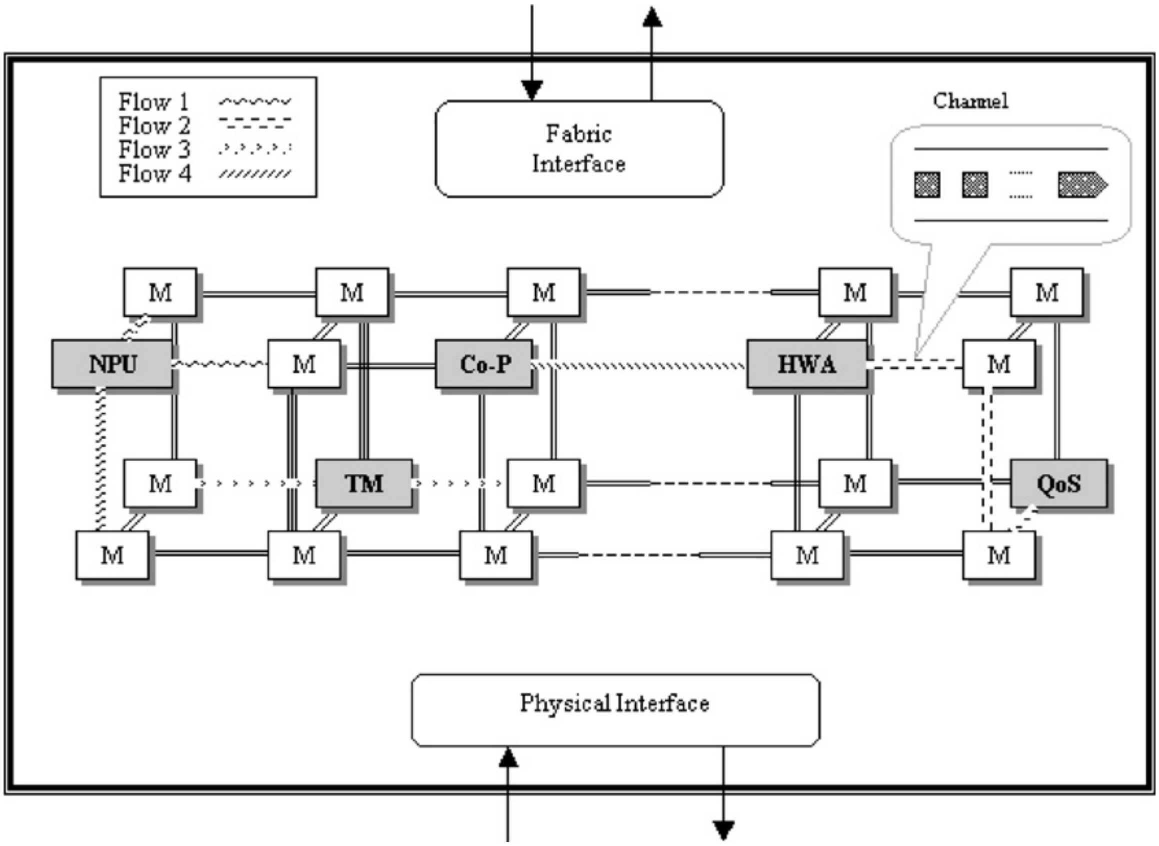


Figure 3 3D Mesh Interconnect Architecture

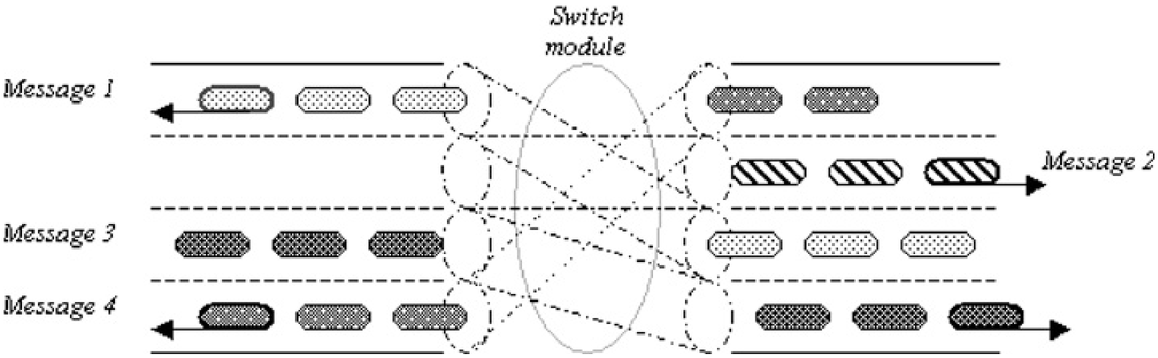


Figure 4 Four Sub-Channels Containing Four Worms Simultaneously

Channels can change their configuration by dividing their width into two or four sub-channels Figure 4. Sub-channeling (SC) permits worms to share the same channel simultaneously. Although per-worm the channel has smaller capacity when sub-divided, it provides worms with extra flexibility in routing through the interconnect instead of being buffered or retransmitted.

The message passing algorithm adaptively routes worms according to three predefined guidelines and by incorporating interconnect traffic conditions. The first guideline ensures that a worm will always attempt to take the shortest path possible to its destination. If the required port is taken by the shortest path rule is occupied as a result of high traffic load, it will test the availability of other ports. The second guideline utilizes past moves to determine the next node that a worm will take towards its destination and avoids certain consecutive moves to inhibit deadlock/livelock situations. The last guideline preserves the worm's relative movement from its source node towards its destination; it will never reverse its direction towards its source.

PEs and memories can be physically located in many different configurations depending on the number of PEs and memories required to complete packet processing tasks. The location and ratio between the number of PEs to memory modules will determine the average distance that a message has to pass in order to reach destination. Average distance has a direct effect on the interconnect performance. Intuitively, as network dimensions increase more configurations can be formed.

One objective, which can be gained by utilizing a simulator, is to find the optimal value of k and n to achieve best performance. The optimal configuration depends on many design constraints as well, such as channel width/density, number of elements connected to the network, and cost. In general, when node delays are neglected and constant bisection width is assumed, a network with lower dimensions has lower latency than higher dimensional networks [29].

Simulating K-Array N-Cube Interconnects

There are several discrete event network simulation and modeling tools available that contain some of the architectural features and functionalities that are incorporated in the model. However, none of these simulation frameworks are capable of delivering the physical and functional attributes required to emulate offchip communications on line cards. Consider three of these simulators, NS-2, Qualnet and OPNET, and the distinction between applications. NS-2, Qualnet and OPNET are well-known network simulators currently used by universities and network design companies [30][31].

NS-2 is an object-oriented, discrete, event-driven network simulator developed at UC Berkeley, written in C++ and OTcl [23]. NS-2 is primarily useful for simulating local and wide area networks; and it supports simulation of TCP, UDP, routing, and multicast protocols over wired and wireless networks [32][33].

The Qualnet is a real-time simulation framework, developed by Scalable Network Technologies (SNT), to emulate the communications of multiple network models [34]. Qualnet includes a rich 3D-visualization interface to provide the user with control over data packets, network topology and performance evaluation. It supports wireless and ad hoc networks as well as parallel and distributed architectures [35]. In addition, it supports multiple routing protocols such as BGP, SIP, RIP, ARP, and BRP. Some related applications that can benefit by using this network simulator include: microwave technologies, high frequency radio communications or satellite communications. OPNET's network modeling and simulation environment delivers a scalable simulation engine that can emulate wireless, point-to-point and multi-point network links. It has the

capability to support routing protocols such as voice, HTTP, TCP, IP, Ethernet, frame relay and more (Wu et al., 2001). Some of the application best
suit for this simulator are mobile, cellular, ad hoc, wireless LAN, and satellite networks. The OPNET simulator allows the user to custom design traffic models since it supports finite state machines and object-oriented modeling (Chang, 1999).

These network simulators are not designed to emulate off-chip communication environment required for our application based on the following differentiations:

- Physical attributes: none of these simulators include specific PCB physical properties which have a great effect on the interconnect performance. Physical properties are crucial to meet the stringent area restrictions on line cards.
- Applications: all three simulators fit better for LAN, AN, mobile and ad hoc communications, not small scale interconnects which require different routing algorithms and flow control mechanisms. The line card simulator must include message flow enhancement features such as virtual channels and sub-channeling.
- Message control: our interconnect simulator provides control of how to deliver messages, perform statistics, gather data, route the packets through the network and run auto test cases. Furthermore, the user has more control of how to save and re-run data using the simulator options menus, rather than learning OTcl or Parsec.
- Participants: while our simulator models communication among PEs and memories, the other simulators include other participants such as PCs, satellite communication, routers or other moving objects.

- Communication medium: most of communication mediums used in these simulators have different signal propagation characteristics and performance. Our off-chip interconnect model is a small scale network in which packets propagate from point-to-point via PCB buses no longer than 1 inch in length.

Cluster Leader Logic Background

Cluster Leader Election

There are three cluster leader election protocols considered for background research to include Control Cluster Head (CCH) [50], Leader Election Algorithm [51], and Least Clusterhead Change (LCC) [52]. CCH and LCC are based on the DMAC (distributed mobility-adaptive clustering) algorithm. DMAC causes clusterheads to change when either of these conditions is met:

1. When two clusterheads come within range of each other.
2. When a node becomes disconnected from the cluster.

DMAC assumes that each node knows its own ID, weight, and role of all its neighbors. In order for this to occur, clusterheads must periodically update their knowledge to other clusterheads. LEA works in a slightly different manner. A new clusterhead is elected when the current clusterhead leaves an area. Clusterheads are organized into a spanning tree; elections can also occur when a clusterhead detects that the spanning tree needs to grow. The spanning tree technique is not used for CLL; however, CLL uses tables similar to LCC.

The background knowledge of the algorithms presents two possible areas of improvement. The first is to reduce or eliminate periodic updates of network statuses to achieve the full picture of the network topology. The second possible improvement would be to eliminate the need for clusterheads to know where each of the other clusterheads is located. These improvements may be realized by using concepts from GPS-QHRA and CLL.

Load Balancing Techniques

Load balancing is an important issue in ad-hoc networks as it translates to end-to-end performance. Among other load balancing techniques, LBAR (Load-Balanced Ad-Hoc Routing) [89] defined a metric called the degree of node activity which represents the load on a node. LBAR sends all the learned routes from the source to a destination node when sending messages. The destination node has the ability to pick the most cost effective route to send messages back. LBAR also uses a path maintenance technique to fix broken links and re-routes packets to other nodes when necessary.

The CLL design for the distributed clustering algorithm is motivated by DMAC and LBAR. The intention is to reduce or eliminate periodic updates (or path maintenance as used in DMAC or LBAR) to maintain a view of the network topology. Also, there is a desire to eliminate the need for clusterheads to know where each of the other clusterheads is located (as used in DMAC). When designing CLL, information is not maintained about network connectivity which is beyond what a particular node needs to know about its immediate surrounding. This reduces the information exchange because routing information does not need to be passed between nodes. Also, when connectivity state is learned by a node, a path maintenance cycle is necessary to maintain and track this information. Depending how far the routes traverse and how fast the wireless nodes may be moving, this overhead could provide little benefit. Aside from the speed of a node, a node may move to an area where the terrain prevents the strongest signal on the least-cost path after the cost is evaluated. CLL tries to emulate wired networking protocols where only the next hop information is known; however due to the more volatile nature of wireless networks, more factors other than just maintaining routing tables are considered and CLL is designed to compensate for these factors.

Related Work on Clustering

Several clustering algorithms and heuristics have been proposed for ad-hoc networks [90], [91], [92], [93]. Many existing solutions take into account various parameters of clusterhead suitability. However the most recognized ones are based on clusterhead selection which rely on random events such as node id assignment (as in the lowest id algorithm) and the degree of connectivity (as in the highest degree algorithm).

The lowest id [94], [95] heuristic assigns a unique id to each node and chooses the node with the minimum id as a clusterhead. Thus, the ids of the neighbors of the clusterhead will be higher than that of the clusterhead.

In highest degree [92], [96], each node broadcasts its id to the nodes that are within its transmission range. A node x is considered to be a neighbor of another node y if x lies within the transmission range of y. The node with maximum number of neighbors (i.e., maximum degree) is chosen as a clusterhead. If there is a tie, it is broken arbitrarily by the nodes’ ids. There are other clustering schemes that consider node and network parameters for deciding the nodes best suited to act as clusterheads.

In the node weight heuristic [97], the nodes are assigned weights based on clusterhead suitability; the neighbor with highest weight wins. This scheme has infrequent node updates but moderate computational overhead. Also, it is not optimized for system throughput and power control.

Uniform leader election [98] is a scheme where a rotated binary tree is used. The non-uniform leader election and the oblivious leader election [99] algorithms are similar in nature; however, based only on a ternary tree and transmit slots respectively. Once again, node suitability is not taken into consideration in neither of the three schemes. The least cluster change (LCC) [100] scheme is based on lowest id or highest connectivity. Re-election is only initiated when a clusterhead moves into another cluster or when a node becomes separated from a cluster. This scheme reduces cluster re-association and increases stability, but is potentially unfair in terms of load distribution.

The mobility-based adaptive clustering scheme is an event driven algorithm based on hybrid routing and node mobility [93]. Two parameters control path availability and effective capacity of path as well as cluster size. It is capable of multi-path transmission to increase capacity; however it has high computational complexity.

In access-based clustering protocol [101] a node receiving a clusterhead declaration from its neighbor prior to declaring itself as a clusterhead becomes a member node. Access to control channel is based on time-division multiplexing with short execution time and incurs low control message overhead. However, clusterhead suitability is not considered. In linked cluster algorithm (LCA) [95], the entire band is divided into M sub-bands (epochs) and the algorithm is performed on each sub-band. The nodes are assumed to have precise synchronized clocks and the number of nodes are known priori.

The max-min D-clustering [102] scheme uses two consecutive broadcasts that are sent in N timeslots to each one-hop neighbor. The scheme is fault tolerant due to availability of multiple paths from gateway nodes; produces fewer clusterheads and is more stable than LCA. The weighted clustering algorithm (WCA) [91] is a weight-based distributed clustering algorithm takes into consideration the ideal degree, transmission power, mobility, and battery power of mobile nodes. A comprehensive comparative performance evaluation of various clustering protocols that help backbone formation in ad-hoc networks can be found in [103].

GPS-QHRA

This work is motivated by GPS-Quorum Hybrid Routing Algorithm (GPS-QHRA) [49]. GPS-QHRA is a routing protocol which uses the clusterhead election process. The routing protocol divides the two-dimensional area into grids and assumes that every mobile node is equipped with GPS capability. A clusterhead, which is also called the Location Database Node (LDN), is identified within a grid. The LDN maintains two routing tables – an inter-zone routing table and an intra-zone routing table.

GPS-QHRA establishes danger zones which give LDNs the ability to change clusterheads if the LDN starts to roam out of a grid. A comparison of proactive (table-driven), reactive (on-demand), and hybrid protocols using geographical zoning and a combination of proactive and reactive routing techniques affirmed that by dividing the GPS-based ad-hoc network into statically allocated hexagonally cellular shaped regions (as opposed to rectangular regions), larger scaled topology networks could be created.

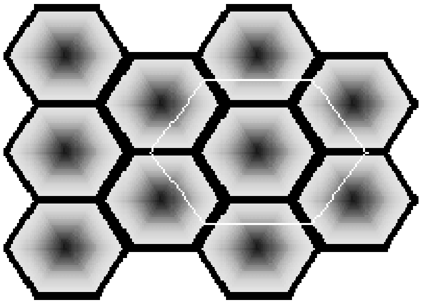


Figure 5 GPS-QHRA Terrain Projected onto 2D Hexagon Cells

The partitioning of such a 2-dimension region is shown in Fig. 1 where the region of interest is divided into fixed sized fixed-location cells. Though nodes and clusterheads move, the cells do not move. The dark regions around the center of the hexagons are the safe zones. The lighter colored regions near the edges of a cell are the danger zones; when a clusterhead is in a danger zone, it may pass (described later) the clusterhead responsibility to another node and change its status to a regular node. These hexagonal regions are an integral part of the algorithm to sort nodes on the topography. The radius of these hexagons is estimated based on the transmission range of the nodes.

Grid Computing Background

State-of-the-art Grid Computing

The grid computing discipline allows for the world’s largest computers to be created [106]. Grids enable resource sharing and aggregation of millions of computational resources over geographically distributed organizations and administrative domains. Grid computing achieves three goals [62]:

1. Resource Aggregation – group computers that are geographically distributed where it appears that there is a single computational system where resources are used as needed.

2. Data Sharing – allow data to be shared between grid resources in a trustworthy and secure fashion.
3. Collaboration – allow different organizations to work together on or integrate projects.

One example of a computational grid problem is a very large problem that can be broken up into pieces where the answers to each piece do not depend on each other. Each piece can be sent out over a network to many computers to be solved. As each piece is solved, it is collected by a server and assembled into a final solution when all pieces arrive.

Consider a hypothetical example for naval military mission planning. Suppose several friendly warships are to engage enemy warships. Two sets of inputs are needed to complete the plan: sensor inputs and platform data. Sensors provide data for friendly and enemy tracking, weather conditions which are needed for weapon systems calculations, oceanographic conditions are necessary for movement calculations, fuel sensors aid in calculating that there is enough fuel to complete the mission and return home, etc. Platform data represents the expected properties of friendly and enemy ships which can include the total number of personnel, the munitions the ship can fire, the quantities of the munitions, the material the hull is made from, etc.

The mission is planned by essentially "rolling the die" for each of these variables with different combinations of quantities or expected behaviors. This type of problem is ideal for a grid because it can be broken into parts where each part represents a roll of the dice; once each set of circumstances is simulated, the results can return to a central location to be compared and reduced to a small set of answers or a single answer. Also, because of the communication medium having little spare bandwidth that the warships use, the ships can only afford to send a limited amount of data to start the planning.

The process by which resources are discovered to plan a mission is unique to this proposal. Typically, resources are logged into a resource broker that is somewhat aware of all of the participants available on the grid. As noted in [60], the resource broker scheme can be a bottleneck because of the amount computational power and network bandwidth needed to maintain a fresh view of the grid. Otherwise, the broker's view of the grid is stale which could produce extra network traffic for work orders to be redirected to different providers. [61] suggests a new concept of placing the load of managing the network on the network itself: inside of the network routing processor (NPU) and memory.

Virtual Organizations

There are several example models that show different configurations where this type of resource discovery would be useful. Before the usage models are introduced, consider the concept of a virtual organization (VO) [62]. Virtual organizations are logical entities, usually with a limited lifetime that are dynamically created to solve a specific problem [106]. VO members negotiate the terms of resource sharing, membership management, security, and access control. For instance, the VO may impose rules for resource sharing that include the amount of time a participant can use the grid, the sharing relationships among the participants, or the sensitivity of the data that participants can process or access. VOs can be organized in many different fashions: for instance a corporation, school, charity, or project can act as a VO.

It is interesting to note revenue possibilities for having a grid infrastructure because membership to an alliance can be billed by a VO Host and/or the VO Host can collect royalties from the transactions delivered and computed on the VO's grid. By being a member of a VO, consumers are aware of the products, security, access, resources available, and protocols run by the VO.

Scheduling

One of the primary grid computing applications is to provision and distribute application codes to specific nodes [106]. One component of the grid computer architecture that performs this functionality is the scheduler. Schedulers can allocate resource for a task and partition the tasks to execute in parallel. A scheduler can be placed on a single machine or distributed throughout the network. The scheduler may schedule resources based on their platform requirements. It may reserve resources in advance, enforce and/or validates service level agreements, enforce resource turn-around policies, monitor job execution status, and reschedule events.

Resource Brokers

The resource broker pairs resources between the resource consumers and resource providers. By knowing various attributes about the grid network, the resource broker can match tasks the best fitting resources. Some factors a resource broker may consider are availability, hardware/software capabilities, bandwidth, and costs. In order for the resource broker to make these types of decisions, it must be aware of job allocation, status management, and data distribution [106]. Middleware exists as part of the

GLOBUS project [65], called GRAM, which allows the resource broker to perform these services: resource allocation, process creation, monitoring, and management services.

Grid Toolkits and Middleware

PlanetLab

PlanetLab [66] provides distributed resources on top of the Internet using the Globus Grid Infrastructure [65, 79, 80]. PlanetLab has two purposes:

- Act as a test bed:
  - Gives researchers access to a large set of geographically distributed machines.
  - This is a realistic network that experiences congestion, failures, and diverse link behaviors (as opposed to just a simulation).
  - There is a potential for real client workloads.
- Act as a deployment platform providing:
  - Researchers with a direct technology transfer path for popular new services.
  - Users with access to those services.

PlanetLab includes a feature called the Virtual Machine Monitor (VMM). One must install the PlanetLab software that downloads a VMM and installs it on the resource node. This is done to add machines to the network and to make them available (which is technically called "slices" of available resources). The VMM specifies the interface to which the services distributed over the testbed are written. The VMM also provides strict security over the amount of memory, disk, bandwidth, and processing power is allowable: with the appropriate password, one can log in as “root”; but even as root some privileges are denied.

UNICORE

UNICORE [9, 64] covers another interesting and applicable area of concern: resource agreements. UNICORE-style resource agreement can be used to form and maintain VO agreements. Using UNICORE as a base, an agreement is made from an agreement template that is converted into an agreement offer that then becomes an agreement instance. This is achieved by an automatic factory service is what provides and allows access rights for the grid consumer. This ideology appears fine, but UNICORE is not very clear on what services are available from the automatic factory service.

Legion

Legion applications use objects to represent processors, data systems, and file systems and construct a shared virtual workspace to collaborate and exchange information [106]. Legion is middleware that resides on the operating system and mediates resources between resource consumers and providers. This allows users to create context spaces to use objects in distributed systems. As objects are defined, they are managed by object metaclasses that have capabilities to create, destroy, activate, or deactivate class instances as well as provide information to client objects.

Condor-G

Condor is a workload management system optimized for high throughput computing where tasks do not need to communicate with each other [106]. It provides task queuing, task scheduling and prioritization, and resource monitoring and managements functions. Condor-G is implemented to work in concert with Globus' GRAM service for inter-domain resource management while using its own software for intra-domain resource management.

Grid Computing Constraints and Issues

Despite the powerful benefits of grid computing as shown with SETI@Home[57] and Einstein@Home [58], the grid has not been formally deployed because of scalability and security concerns. The goal is to design a grid resource discovery protocol to enhance scalability and to develop a simulation to model the grid network using these new developments built on a common software baseline that can be used to create other simulators.

Typically, computational grid resources are logged into a resource broker [66][67] that is aware of the participants available on the grid. The resource broker scheme can be a bottleneck because of the amount computational power and network bandwidth needed to maintain a fresh view of the grid. Otherwise, if the view is not maintained, the broker's view of the grid becomes stale which could produce extra network traffic for work orders to be redirected to different providers. A new concept is suggested of placing the load of managing the network resource discovery on the network itself: inside of the network processor (NPU) that is employed on the line cards in routers.

This imposes changes to the grid computing architecture as well as to the networking infrastructure. The traditional role of the resource broker is greatly simplified. The grid resource discovery protocol finds resources by using a scoring mechanism; the resource broker only needs to determine a desired score of a task. The role of the scheduler is changed as well. The scheduler will less work to do for monitoring resources since the network routers will be doing that work as resource providers update them as they become available or consumed.

Grid Deployment Environments

In order to see how the resource discovery protocol fits in the real world, it helps to understand the environments that grids are deployed in [56]. These environments provide the scenario that the resource discovery protocol can be simulated in. The differences between different environments lie in the application of the scenario, the type of deployment, and the security needed. Five such environments are discussed in [68]: science portals, distributed computing, large-scale data analysis, computer-in-the-loop instrumentation, and collaborative work. Each of these examples is discussed in detail in this section with a brief statement of how the resource discovery protocol can be used in this situation.

Science Portals

Science portals on the web can allow scientists to perform tasks on a grid without having to learn how to install or maintain the grid components necessary to run [68]. This type of deployment for portals is known as thin deployment [69] that allows communication to occur using standard web browsers and HTML and DHTML.

[67] highlights an example science portal called the astrophysical computing. The goal of the resource discovery protocol characterized in this work is for the workload to

be possibly reduced or eliminated for two of the components identified in the astrophysical portal design: resource monitoring and resource management. These components can be moved from the application server to the networking hardware infrastructure if the resource discovery protocol proves to be effective. For this scenario, the discovery routing protocol could work as follows:

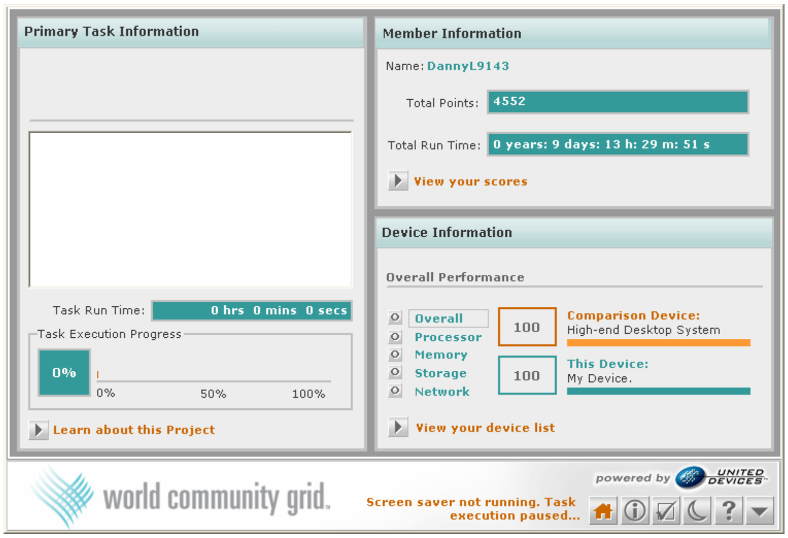
1. A scientist logs onto a science portal and identify the task to be computed.
2. The portal identifies the types of resources needed to perform the computation and sends a request message through the networking infrastructure, which uses the proposed resource discovery routing protocol.
3. If resources are found, each resource sends a message to the portal via reverse path forwarding.
4. The portal negotiates the connection between the resource and the scientist's computer and computation thus begins.

Distributed Computing

Individual PCs can be combined via parallelization to provide substantial computational resources. One such example of distributed computing is FightAIDS@Home that is part of the World Community Grid [70]. Individuals wishing to donate their idle computational clock cycles can have their PC's run molecular analysis to help find drugs to fight HIV and AIDS. To help FightAIDS@Home, one downloads an agent (pictured in Figure 6) which requests for drug molecule representations and models its effects on HIV or AIDS.

The grid resource discovery protocol could help the server which doles out drug molecule models expedite its workload more efficiently. Rather than waiting for pings from available agents, the server could send the work order out over the network and the routers will deliver the drug molecule model to an appropriate resource. Consider a resource running an agent that has available idle computational power and no molecule to model:

1. The agent sends a resource availability message out through the grid network.
2. The routers in the network record the resource availability as the message is forwarded.
3. The server has a new drug molecule to model and sends the request through the network.
4. The request is routed through the network and ends at an available resource.
5. The resource agent contacts the drug molecule server, downloads the molecular model, and begins computation.


Name: DannyL9143

Total Points: 4552

Total Run Time: 0 years: 9 days: 13 h: 29 m: 51 s

Task Run Time: 0 hrs 0 mins 0 secs

Task Execution Progress: 0%

Comparison Device: High-end Desktop System 100

This Device: My Device. 100

Screen saver not running. Task execution paused...

Figure 6 FightAIDS@Home Execution Window

To help FightAIDS@Home, download the agent shown in Figure 6. When your computer becomes idle (for instance when your screen saver is on), the agent will download a drug molecule to model fighting HIV/AIDS and begin modeling it.   The proposed resource discovery protocol can help this situation by allowing the molecule server to expedite requests without having to wait for pings from available agents.

Large-Scale Data Analysis

Computational grids provide the capability of acting as a large storage facility in addition to providing computational powerhouses. Scientific problems exist which require petabytes (1,000,000 gigabytes) of data to be stored and processed throughout a grid network [71]. The grid resource discovery protocol can help with this scenario because it uses storage as one of the determining factors for tracking grid resource providers. The discovery protocol would as follows in this scenario:

1. A grid resource with s megabytes of storage space becomes available to the grid network. A resource availability message is sent from the resource provider to the central archive that indicates the CPU speed, storage space s, and various other parameters.
2. As the message hops from router to router in the archive, the parameters (including s) in the message are recorded in tables within the networking hardware.
3. When the central archive is ready, it sends out a new work order through the grid network containing a tuple of search criteria: CPU speed and storage capacity.
4. As the order hops through the networking hardware, the parameters are compared to the values in the resource tables to ensure that the CPU speed needed is met or exceeded and that the storage capacity needed is less than or equal to s.
6. Eventually, the work order will arrive at a grid resource provider, the data will be downloaded, and the processing can begin.

Computer In-The-Loop Instrumentation

There are scientific instruments that are used to collect streams of data which are archived and processed later to detect things of scientific value [6]. The processing can take a significant amount of time that may result in finding a brief period of information that is very useful to a scientist. It would be more practical, for instance, for automated software to detect when useful information is about to be captured, process that information immediately and then highlight intermediate results to a scientist before the entire data set is collected. The on-demand type of analysis can be conducted using a grid network. The transaction would work like this:

1. The instrumentation detects that an important event is about to occur. A resource discovery message is sent out through the grid network for available resources.
2. When the resources reply back to the instrumentation device, the device immediately sends work orders through the grid networking infrastructure.
3. The data is sent to the grid resources for processing. When the processing is complete, the instrumentation (or another computer) can receive the message and notify a scientist of an important observation as it is being monitored by instrumentation and processed in the grid.

Collaborative Work

When scientific results are collected and analyzed, scientists may want to collaborate to discuss results and offer suggestions. This type of collaboration can be done in real time that demands high bandwidth, fast processing power, and access to stored results [68]. While one group of scientists review simulation results, other scientists may be examining the data or similar data from different runs more closely or they may be running their own simulations to verify the results. The resource discovery protocol proposed in this work suits this scenario as well since the distributed nature of the protocol does not allow for many scientists to simultaneously accessing the same resource broker while the resources are talking to it. The discovery protocol would as follows in this scenario:

1. A scientist wishes to validate a fellow's work by running a similar analysis. A resource availability message is sent from the resource provider to the central server that indicates the task to be run.
2. As the order hops through the networking hardware, the message is routed to an available server.
4. Eventually, the work order will arrive at a grid resource provider, the work order will be downloaded, and the processing can begin.

Simulation Protocol Background

Aggregate Level Simulation Protocol (ALSP)

One example of a legacy simulation protocol is Aggregate Level Simulation Protocol (ALSP) [2] developed in 1992. ALSP is an example of a protocol allowing Advanced Distributed Simulation (ADS): the integration of simulations to support training in a large parallel computing environment called a confederation. This allowed the formal introduction of four important principles that ALSP borrowed from SIMNET [8]: dynamic configurability, geographic distribution, autonomous entities, and communication protocols. ALSP also introduced new concepts, at the time, to include simulation time management, data management, and architecture independence. These features are described in Table 1.

Simulators that participate in an ALSP confederation are called actors. Actor simulation objects, or entities, go through a dynamic lifecycle from creation to removal during a simulation exercise. Each entity has associated attributes or values belonging to it as defined in the confederation object model. This is similar to an object in Object Oriented Design (OOD) [9].

Table 1

ALSP Architectural Features

Architectural Feature	Description
Dynamic Configurability	Allows simulators to arbitrarily join or leave a confederation.
Geographic Distribution	Simulators can exist anywhere around the world, but the terrain used is the same logical terrain.
Autonomous Entities	Each simulation controls its own resources (objects or entities.)
Communication Protocols	Information is passed from simulator to simulator using the same messaging protocol.
Time Management	Constructive simulators can operate outside of the normal wall-clock time experienced: faster or slower than wall-clock time.
Data Management	Maps the internal simulator state representation consistently at the confederation level.
Architectural Independence	By being architecturally independent, ALSP was designed to be non-obtrusive and easy to adapt.

One distinguishing feature of ALSP from OOD is that different actors can own different ALSP attributes within the same entity object. The process of owning an attribute in ALSP is called locking. Objects are locked based on their registration or discovery. An actor registers objects into the ACM by default in the locked state (or optionally in the unlocked state). Another actor’s ACM discovers the object registry and puts the information in its local database. Also, objects that are seen but not owned by other actors are known as ghosts. Interactions are the messages that are passed between actors when there is a change to an object and the ghost must reflect that change.

The ALSP infrastructure is composed of four components:

- The ALSP Common Module (ACM)
    - Performs time synchronization: synchronous (time-stepped) or asynchronous (next-event).
    - Manages objects.
    - Coordinates actors joining and leaving the confederation.
    - Filters out incoming messages that are not needed by the receiver.
    - Allows and enforces attribute ownership transferability.
- The ALSP Broadcast Emulator (ABE) – provides message distribution capabilities in LAN and WAN environments.
- The ALSP Control Terminal (ACT) – used to control confederation wide messages.
- The Confederation Management Tool (CMT) – used to view various confederation parameters or statistics.

Object management introduces the concept of filters. The ACM database is composed of several data sets about object creations, object updates, and other object interests. These can be used in conjunction with filters to prevent the actor from knowing certain interactions while allowing the actor to know other interactions. Filters can be used to discriminate objects, attribute values or ranges, and/or geographic locations of the entities to notify the actor of only relevant data.

Data is passed from actor to actor via a text-based messaging scheme. The semantics of the protocol are confederation dependent; so if a simulator is blindly transferred from one confederation to another, there is no guarantee that it will be able to successfully read or write understandable messages to or from other actors.

Distributed Interactive Simulation (DIS)

Distributed Interaction Simulation (DIS) was designed to be an infrastructure to build distributed simulations on [15]. DIS addresses application protocols, real-time communications, and exercise management and feedback. Even though ALSP [2] and HLA [10] were spawned from Department of Defense interests, DIS is tightly coupled to military exercises where ALSP and HLA are looser and can be applied to other domains. Like ALSP [8], DIS has origins from SIMNET.

DIS, functionally, is designed to achieve seven functional requirements [15]:

1. Entity Information and Interactions. An entity can be a vehicle, person, building, munition, or cloud. All entities are enumerated based on their entity type as defined in the DIS spec [15].
2. Warfare. Warfare involves firing and detonating munitions.
3. Logistics. Logistics messages are composed of supply (or resupply) and repair services to include medical repair.
4. Radio Communications. Sending entities define the details of the communications device and the data communicated; the receiving entity determines if the data can be received.
5. Distributed Emission Generation. Representation of lasers and active electromagnetic and acoustic emissions are essential in certain simulation

exercises. Emitting entities simulate their emitter and output real-time operational parameters. Each receiving entity is responsible for determining if the emission is detectable. [15]

6. Management. DIS management is divided into network management and simulation management. The network manager analyzes performance, monitors load and network nodes and gateways, and helps with error recovery. The simulation manager manages the simulation exercise which includes starting, stopping, and pausing the exercise, removing models from an exercise, and the collection and distribution of data within the exercise.
7. Environment Information. Different factors in the environment (terrain, weather, oceans/water, ambient illumination, engineering objects like bridges and buildings, and atmospheric conditions) make the simulation exercises more realistic.

Application Protocols

The main application protocol mechanism, which distinguishes DIS from HLA and ALSP, is the transfer of Protocol Data Units (PDU) [15][16]. PDUs are data messages sent between simulation applications on a network. Messages are grouped into specialized domains called protocol families. All PDU information is "hard-coded" into the DIS standard that guarantees that, in theory, any DIS application can work with any other DIS application.

Simulations are generally responsible for controlling at least one entity in the simulation. Also as an added responsibility, when the entity modeled performs an observable action, the simulation that controls the entity is responsible to send the appropriate PDUs on the network to the applications. The receiving simulations are responsible for tracking and monitoring these messages. These observable actions or states are known as ground truth data. The receiving simulation may take this ground truth data and change it to what its model thinks it sees (known as perceived truth.) For instance, a radar simulator may be notified of a flying aircraft before it is supposed to display it to the operator (perhaps due to the limitation of the radar fan). So, the operator does not perceive an aircraft until the simulator calculates that it is within range of the radar.

Table 2

An example PDU: Minefield Response NACK PDU

Field Size in Bits	Minefield Response NACK PDU	
96	PDU Header	Protocol Version—8-bit enumeration Exercise ID—8-bit unsigned integer PDU Type—8-bit enumeration Protocol Family—8-bit enumeration Timestamp—32-bit unsigned integer Length—16-bit unsigned integer Padding—16 bits unused
48	Minefield ID	Site—16-bit unsigned integer Application—16-bit unsigned integer Entity—16-bit unsigned integer
48	Requesting Entity ID	Site—16-bit unsigned integer Application—16-bit unsigned integer Entity—16-bit unsigned integer
8	Request ID	8-bit unsigned integer
8	Number of Missing PDUs	8-bit unsigned integer
8n	Missing PDU Sequence Numbers	8-bit unsigned integer

The number of bits, type of data, and format of data is specified.

When entity location PDUs are passed around the simulation, a standard view of the world is used which rotates just as the Earth does. A right-handed geocentric coordinate system is used. Geocentric means the origin of the (x, y, z) axes is that the center of the Earth [17]. The positive x-axis passes through the Prime Meridian at the Equator, the positive y-axis passes through the Equator 90 degrees east of the Prime Meridian, and the positive z-axis passes through the North Pole. One unit of measurement in this system is equal to 1 meter in the simulation. An entity's location is based on its center of its bounded volume and excludes extremities. When firing munitions, the location of the weapon and type of munitions (at a minimum) are communicated.

Real-Time Communications

DIS promotes ad-hoc networking by not requiring any computer to control the simulation [15]. Thus, simulation applications can join or leave the DIS exercise at any time (from a technical perspective). The simulations are responsible for knowing the state of the entities in an exercise.

In an attempt to reduce the amount of data on a DIS network, an algorithm known as dead reckoning [15] is used to limit the amount of positional (or "Here I am!") messages on the wire. One technique of achieving this is to send an entities orientation and speed (or its velocity vector) with its initial location. Receiving simulations can then estimate, or dead reckon, the course the entity would take over time. When the entity changes speed or direction, if the entity moves past a particular threshold, or on occasion, the controlling simulator will send out a new PDU indicating the new location, speed, and orientation of the entity.

There are a couple of caveats worth mentioning about dead reckoning. For most military ground objects, dead reckoning is an appropriate algorithm. However, for "fastmovers" such as airplanes, jets, and especially missiles, the dead reckoning calculation is not as effective as when it is used for ground entities. By the time the next positional update PDU is generated, the missile has most likely hit its target and the airplane or jet has moved so fast that the dead reckoning algorithm may not be of much value. Also, each simulation may use different parameters or formulas when calculating dead reckoning. So, one simulation may show an entity in a particular location where another simulation might show the same entity in a different location. This could produce an issue, for example, where if a bomb goes off in the first location, one simulation may perceive the entity as alive whereas the other simulation may perceive the entity as destroyed.

There are other ways of optimizing communications in DIS. These can include data compression, simulations filtering out data, putting different simulations on different multicast subnets, and sending only changes to PDUs rather than entire PDU updates.

Time Management

DIS communications are real-time (as defined by the Universal Coordinated Time (UTC)) and an exercise can commence during a simulation time. So, the UTC real-time is the present time, but a simulated time could be two years ago. PDUs can be time-stamped to indicate the time when the PDU is valid. Also, DIS has the concept of a heartbeat when all entities are refreshed periodically. This allows DIS simulations that leave and re-enter an exercise the opportunity to catch-up to what has been going on since the simulator left. Also, DIS traffic is unreliable, so if a message was dropped due to network congestion, the heartbeat allows a mechanism to resend this data.

Exercise Management and Feedback

Simulation management functions can be divided into exercise management and data management [15][18]. Both entities and exercises can be initialized, started, or stopped by the simulation manager and entities can be paused, reconstituted, or removed. When entities are created, an acknowledgement message is sent to affirm the creation. A Set Data PDU can be issued to change parameters of an entity.

Entities are allowed to have three states [15]:

- Simulation state - when the entity is being simulated.
- Wait state – when an entity is removed.
- Stopped or Frozen state – when the entity is not simulating and can be started at any time.

Feedback is provided to the simulation management through several mechanisms to include the Event Reporting PDU. Also, data can be requested by using the Data Query PDU. A simulator can monitor this traffic and display it to a simulation manager as appropriate or it can record this information for retrieval or playback at a later time.

High Level Architecture (HLA)

Signed into effect October 2005, the U.S. Department of Defense created their Modeling and Simulation Master Plan [11]. Among other things, the plan calls for all DoD models and simulations to conform to HLA (High Level Architecture.) HLA, as outlined by the plan, serves many purposes:

- Facilitate interoperability.
- Encourage reuse.

- Make no specification about the internal structures of simulation.
- Provide the Runtime Infrastructure (RTI) Services that allow models and simulations to participate in an HLA simulation.
- Use the Object Model Template (OMT) that describes the entities and interactions in an HLA simulation.

Thus, HLA was officially born and work began creating federation rules, an interface specification, and the OMT [10]. The federation rules help to define the proper interactions between simulations and describe each simulation's responsibilities. The interface specification defines the RTI services and identifies callback functions each federate must provide. The OMT provides a common way for simulations to share data by creating the Federation Object Model (FOM), Simulation Object Model (SOM), and Management Object Model (MOM).

Federation Rules

HLA definitely has similarities to ALSP [2]: ALSP has a confederation with actors, a confederation object model, and objects and interactions; HLA has a federation with federates, a federation object model, and objects and interactions. The federation rules differ between ALSP and HLA and HLA is more specific in some instances than ALSP with federation rules. Also, ALSP was an architecture and an implementation where HLA is an architecture and the RTI is the implementation; the two were completely split apart. There are ten basic rules of HLA as defined in [10] and the next section.

Run-Time Infrastructure (RTI)

The Run-Time Infrastructure (RTI) is the implementation of HLA [10]. As outlined in the DoD M&S Master Plan [11], the RTI encourages interoperability and distributed computing. One of the primary concepts behind the RTI is that it separates simulation from communication: the federates simulate, the RTI encapsulates federate-to-federate communications. Main functionalities of the RTI are discussed below: improvements from DIS and ALSP, the lifecycle of a federation, object declaration and management, time management, and sync points and federation commands.

The RTI Software

The RTI software is composed of the RTI Executive Process (RtiExec), the Federation Executive Process (FedExec), and the libRTI library. The RtiExec manages the creation of a FedExec process within a single network. The libRTI library provides the HLA services to the federate. Any model that desires to become a federate must include the RTI header files, call the appropriate functions to act as a federate, and link to the libRTI library.

The RTI can execute on a single computer, on a LAN, or on a distributed complex network. The RtiExec process is started on a computer; when the first federate creates a federation, the RtiExec process forks off a FedExec process on its same computer. The FedExec process manages federates entering and leaving the federation.

When a federate initializes their local instance of the RTI, the libRTI creates the Local RTI Component (LRC). The mechanism by which the LRC knows how to communicate to the RtiExec is through settings in the RID file which indicate the IP address of the computer hosting the RtiExec process. When a connection is successfully established, the federate can start sending and receiving objects and interactions and perform all other HLA functionalities.

Table 3

HLA Federation and Federate Rules

Federation Rules	Federate Rules
Federations shall have an HLA Federation Object Model (FOM), documented in accordance with the HLA OMT.	Federates shall have an HLA Simulation Object Model (SOM), documented in accordance with the HLA OMT.
In a federation, all representation of objects in the FOM shall be in the federates, not in the RTI.	Federates shall be able to update and/or reflect any attributes of objects in their SOM and send and/or receive SOM object interactions externally, as specified in their SOM.
During a federation execution, all exchange of FOM data among federates shall occur via the RTI.	Federates shall be able to transfer and/or accept ownership of an attribute dynamically during a federation execution, as specified in their SOM.
During a federation execution, federates shall interact with the RTI in accordance with the HLA Interface Specification.	Federates shall be able to vary the conditions under which they provide updates of attributes of objects, as specified in their SOM.
During a federation execution, an attribute of an instance of an object shall be owned by only one federate at any given time.	Federates shall be able to manage local time in a way that will allow them to coordinate data exchange with other members of a federation.

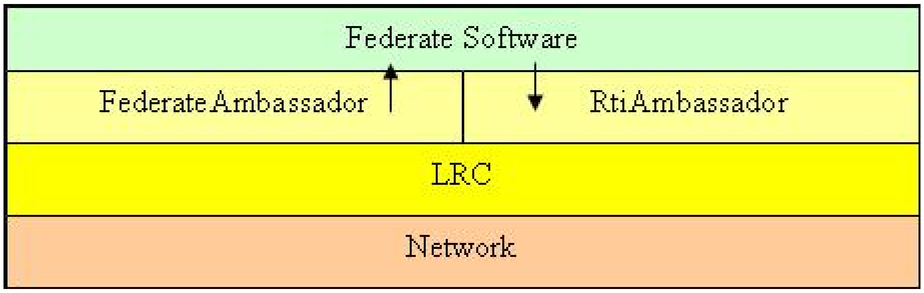


Figure 7 Federate Outbound RTIambassador and Inbound FederateAmbassador Architecture

The libRTI library contains the RTIambassador class which gives access to all of the functions defined to provide HLA services. Federates receive callbacks and information through the FederateAmbassador abstract class either synchronously or asynchronously. Shown inFigure 7, the federate cannot access the LRC or network directly. All calls are made into the RTIambassador by the federate.

Improvements from DIS and ALSP

There are several improvements of the RTI over DIS and ALSP:

- The simulation is separate from the communications. This means that minimal changes are needed to a federate as the RTI changes. In DIS, the communication mechanisms are generally wide open. This also allows for sophisticated communications models that can be shared among different federations.
- The RTI is information independent and the RTI saves no state and message passing is generally consistent from federation to federation. DIS heavily relies on predefined PDUs. ALSP has data formats that differ from confederation to confederation.
- The RTI dynamically handles FOM data as the FOM is read in during federation creation. In DIS, the PDUs are actually part of the IEEE spec. So, changing the default PDUs officially requires an act of IEEE.
- The RTI handles synchronous and asynchronous time models as well as connected and connectionless modes. With the connected mode, synchronous time management is possible as well as creating federations that manage the joining and resigning of federates. Connectionless mode enables ad-hoc joining

and resigning and asynchronous without requiring RtiExec or FedExec processes (thus the RTI has the ability to back-support DIS in an HLA style). There is no realistic way to run an HLA federation using a DIS backbone.

- The RTI introduced the MOM which allows federates to know the internal status of the RTI and the federation at any time. Also, the federation can be controlled through MOM interactions.
- RTI messages are passed as binary data where ALSP passes data as human readable strings. This allows a greater variety of data types and increases their accuracy.

The Lifecycle of a Federation

Each HLA federate maintains a similar lifecycle as pictured in Figure 8. The federate attempts to create a federation and then joins it either if it was created successfully or was already created. Then, the federate declares what objects and interactions it is capable of publishing. Objects are created and registered, and then the federate subscribes to the objects it wishes to know about. A discovery is received for each object in the federation. Messages are sent and received and object updates are received. Optionally, the federate may choose to exchange attribute ownership with other federates. Eventually, some objects will be deleted. When the federate is ready to retire, it resigns from the federation and tries to destroy it. If there are other federates in the federation, the RTI will not allow the FedExec to be destroyed.

Object Declaration and Management

As outlined in Figure 8, federates can publish (send) and subscribe to (receive) object creation and updates and interactions. If a federate does not subscribe to any data, it will not receive any data. Publication and subscription requests can be modified at any time during the simulation. So, for instance, if a federate has a GUI window open which pertains to monitoring vehicle locations, the federate can subscribe to the vehicle location updates. However, if the GUI window is closed, then the federate can unsubscribe from the vehicle updates since they are no longer visible to the user; this could improve the performance of this particular federate and the network traffic.

Objects are the things being simulated; interactions represent the events that happen between these objects. Objects have attributes and federates subscribe and publish the individual attributes of each object. Interactions have parameters and either a federate subscribes to or publishes an entire interaction; the federate cannot just subscribe or publish a particular interaction parameter. Objects persist throughout the game (unless removed) whereas interactions only occur once when sent. Both interactions and attribute updates can be time stamped.

Creating and updating objects are two separate tasks when using the RTI. The procedure of creating an object is called object registration (

Figure 10). Once the object is registered, it can be updated (Figure 11). Interactions, on the other hand, are just sent (Figure 9). Further details regarding the function calls and code examples are in [10].

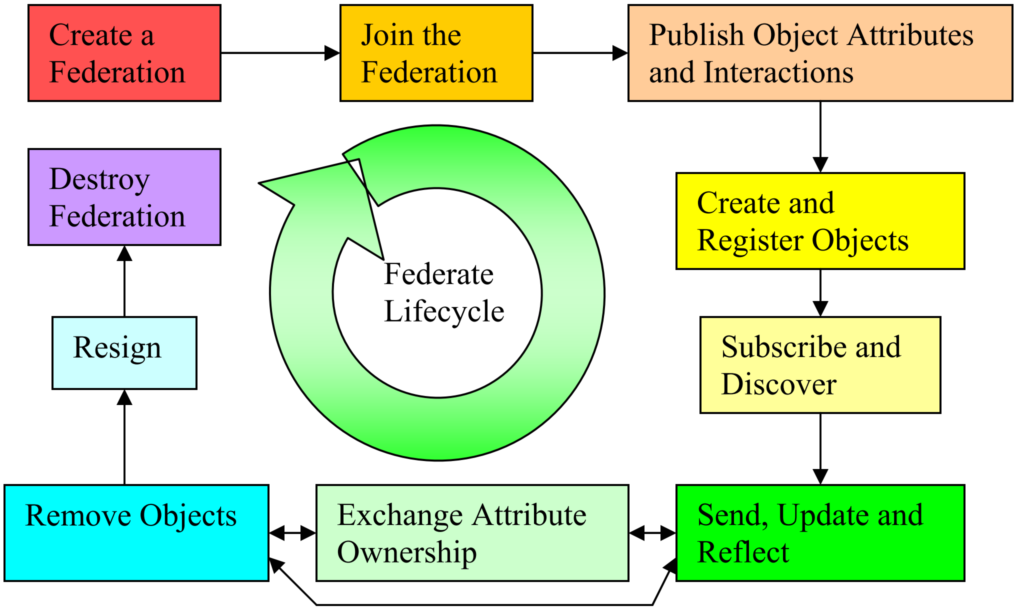


Figure 8 A Typical Federate Lifecycle

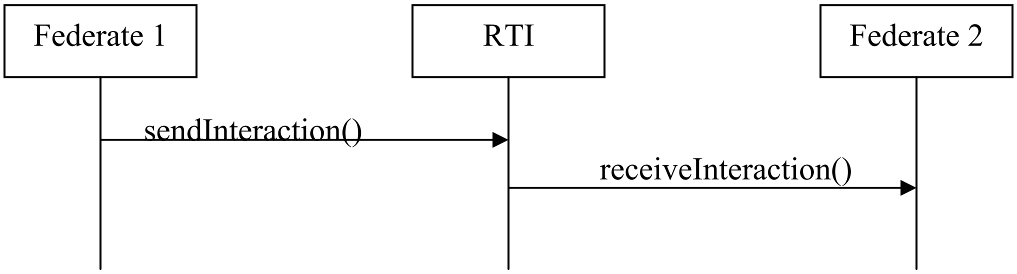


Figure 9 RTI Methods to Send and Receive an Interaction

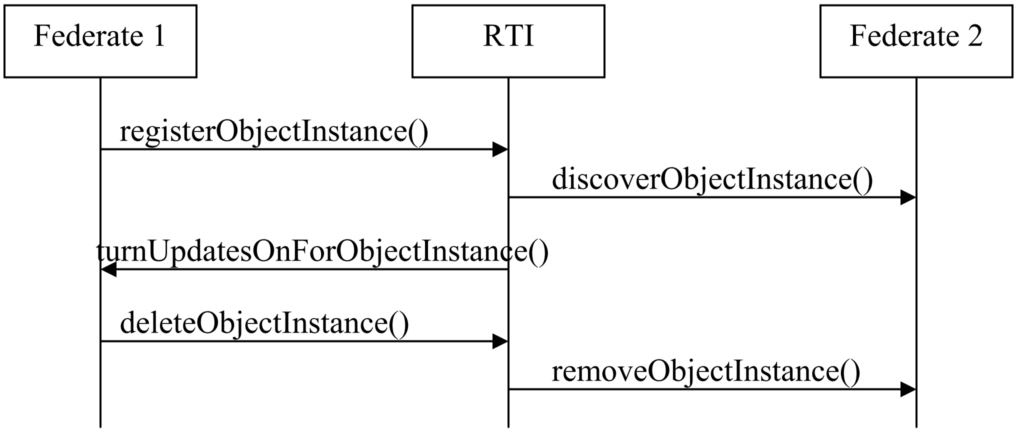


Figure 10 Object Creation and Deletion Sequence Diagram

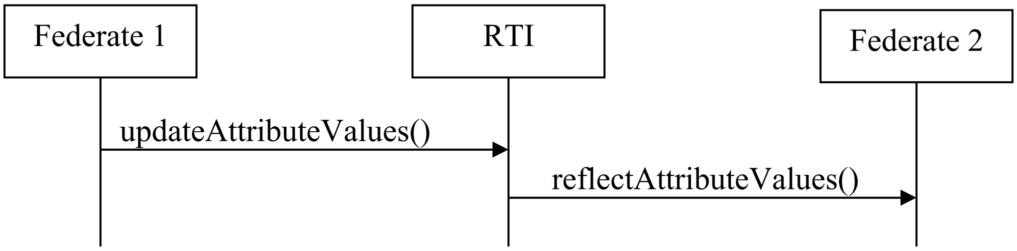


Figure 11 Updating an Object's Attributes

Time Management

There are several time management policies available from the RTI [10] as described in Table 4. Time management with the RTI can work cooperatively with other federates in a simulation or there can be no time management at all. Different federates in the same federation can have different time management policies. By default, the RTI does not have a time policy, but time always moves forward.

Table 4

RTI Federate Four Time Management Options

	Not Time Regulating	Time Regulating
Not Time Constrained	Default setting. The RTI does not manage this federate's time.	This federate can control the advancement of time for federates that are time constrained.
Time Constrained	This federate is controlled by federates that are time regulating.	This federate can control the advancement of time and be affected by other federates that are time regulating.

When time management is enabled, the time advances are designed to make sure that object updates and events are delivered in an ordered fashion. It is possible for different federates to have a different current time. If a federate can hold the clock, then it is a time regulating federate and the appropriate RTI call is made to set the federate as time regulating. When time regulating federates hold or advance time, the RTI can throttle federates to either pause or process when time constraining is enabled by a federate. Note that the status of regulating or constraining can be changed at any time during a federate's lifetime.

To apply a timely delivery of an interaction or object updates, these orders must be time-stamped to alert the RTI that these messages are time sensitive. Time constrained federates receive their events in time-stamp order. Time-stamped messages must be sent from a time regulating federate at a time equal time its current time plus the lookahead value which is greater than or equal to zero (note that zero is a special case).

At the time a federate becomes time regulating, it specifies the lookahead value for the RTI and the federate to use. TSO events do not have to be generated in order; but they must be greater than current time plus lookahead. When the time regulating federate posts time-stamped messages, the messages are placed in a Time-Stamped Ordered Queue (TSO Queue). Time constrained federates receive the TSO events in order; non-time constrained federates receive the event but not in any guaranteed order and absent of the time-stamp information. These events are considered receive-ordered (RO) events and are placed in a FIFO RO queue. The fact at which a message can be placed in a TSO queue is identified in the FED file (discussed in the OMT section) when a message's time management policy is marked as "timestamp" (as opposed to "receive").

The lowest time for a message that a federate can receive is the Lower Bound Time-Stamp (LBTS). The LBTS calculations consider the earliest possible time that any of the federates can send a message [10]. So, this value is continuously being updated as each time regulating federate progresses through time. A federate can never advance its internal clock past the LBTS.

All federates, regardless if they are time constrained or not, ask the RTI (the LRC) for a time grant. Unconstrained federates will immediately receive a time grant. Time constrained federates, however, will wait for the RTI to grant them permission; when permission is granted, the RTI will notify the federate what time to advance to (thus preventing them from exceeding their LBTS). Interestingly, if a federate joins late into a federation with time regulating and constrained federates, the federate will be granted a time where it cannot send events in the past.

Time advancement requests can be one of three ways which can be changed during any time during the execution of a federation: time-step, event-based, or optimistic. Time-step federates process all events within the window of current time plus the time step. When a federate calls timeAdvanceRequest() (TAR) or timeAdvanceRequestAvailable(), the federate is then allowed to receive messages in the RO queue and messages from the TSO queue less than or equal to the time requested from the TSO queue. When all eligible TSO events are received, the federate receives a timeAdvanceGrant() (TAG) callback from the LRC with the time requested from the TAR.

Event-based simulations would call the nextEventRequest() (NER) or nextEventRequestAvailable() function (NERA) similar to the TAR. The reason for using event-based time requests is that the sending of events is dependent on the time of receipt of a previous event. Likewise when using TAR, a TAG is received equal to the minimum event time in the TSO queue or the NER or NERA when all possible TSO messages with time equal to the minimum next event time have been received.

Optimistic federates can actually process events ahead of the LBTS in the future. Thus, the federate wants to receive all events regardless of their time-stamp. Federates enact this by calling the flushQueueRequest() function. Similarly, once all messages flagged for delivery are de-queued, a TAG is given of the time requested from the flushQueueRequest() call. Optimistic messages are received out of order; so the possibility exists for a new event occurring before an event already received could invalidate previous messages. Thus, the invalid message has to be retracted through retraction services provided by the RTI.

An RTI mechanism, rather than an HLA mechanism, of ticking time is required by the RTI in order to receive events. Since the RTI is multi-threaded, the tick() method notifies the RTI that it can do internal processing so the LRC. Failing to tick() the RTI could cause a federation wide deadlock condition. Note that a call to tick() does not advance the federation time, it allows the RTI to process data.

Sync Points and Federation Commands

The RTI also allows for additional functionalities such as sync points and the federation wide saving and restoring of data [10]. Since there are varying time advancement policies, it may be necessary to have the federates synchronize at a particular point in time before continuing on through time. To synchronize a federation, the caller needs to provide a string label to the registerFederationSynchronizationPoint() function call in Figure 12.

The RTI also has the ability for a federation wide save or restore capability. The save feature is requested by a federate, all federates save their local state to local files. Then, the LRC, RtiExec, and FedExec processes save their data as appropriate. Once all saves are complete, then the federation continues its normal processing. Each federation save is essentially a snapshot of the federation at a particular time.

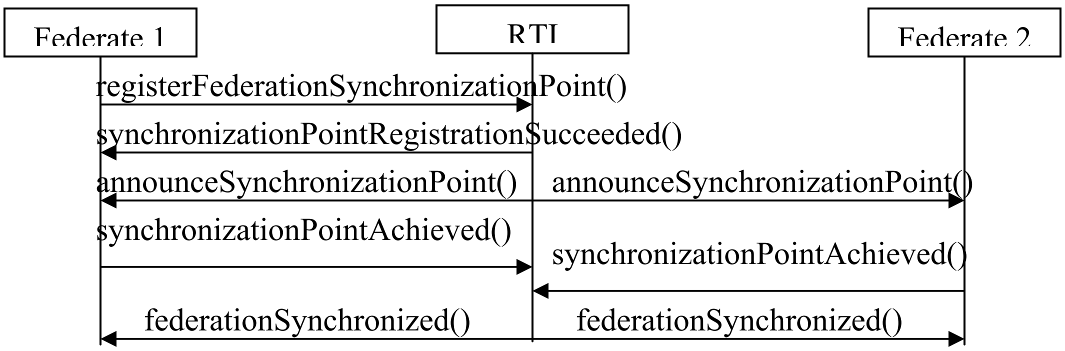

Federate 1
RTI
Federate 2
registerFederationSynchronizationPoint()
synchronizationPointRegistrationSucceeded()
announceSynchronizationPoint()
announceSynchronizationPoint()
synchronizationPointAchieved()
synchronizationPointAchieved()
federationSynchronized()
federationSynchronized()

Figure 12 Announcing and Achieving a Synchronization Point

The RTI supports two types of restore methodologies: cold and warm (or hot) restore. For a cold restore, the federation is brought up in a minimal state, then the federate state is restored from a previous save file. When all federates and the federation have restored, the simulation continues on from the time it left off. A warm restore happens when a simulation is running and a restore occurs when the simulation is not starting from scratch. Thus, each federate must appropriately clean up all of its memory, data, and open file handles and sockets before attempting to restore from a previous save file. When the federates and federation have restored their states, the federation essentially jumps to the time of the saved federation.

Object Model Template (OMT) and the Federation Object Model (FOM)

The Object Model Template (OMT) provides the common framework for object and interaction documentation and interoperability, and encourages reuse of objects [14]. These objects and interactions are described as managed by a federate and what is visible outside of that federate. Data definitions fall into three areas of the OMT: the FOM, the SOM, and the MOM.

The FOM is described in several different files at different levels of detail: the FED file, the omd file, and the omt file. For HLA 1.3 [10] the fed file has a custom format but in the most recent HLA version IEEE 1516 [12], the FOM is in an XML format. The RTI uses the FED (Federation Execution Data) file which is really a subset of the FOM, the other files are products of a tool called OMDT Pro [13]. The omd and omt files contain additional data (such as FOM item descriptions) which some federates may find useful. The SOM is a federate's local copy of the FOM with additional items (if desired) that are included within the federate only and not shared in the federation.

The MOM provides simulation management data by fields specified in the FOM. Though the RTI is technically FOM independent, if the MOM is present in the FOM (which it should always be), then the RTI can provide useful information such as:

- Federates in a federation.
- Current time.
- Federates status of time constraining and time regulating.
- Save and restore features.
- The pacing rate if set and other time and LBTS calculation information.
- The ability to turn advisories on or off.

- The ability to resign a federate.

Distributed Interactive Simulation (DIS) Revisited

Modern adaptations of DIS actually make DIS more like HLA. One such spec, the GRIM RPR (Guidance, Rationale, and Interoperability Modalities Real-time Platform Reference) [1], is based on DIS where the DIS PDUs are placed into a RPR (pronounced *reaper*) FOM. Using the RTI [10], which has connectionless features (unreliable message delivery) and time unconstrained and non-regulation, DIS has an improved networking backbone than the traditional way of sending messages in DIS by broadcasting. Also, depending on the implementation of the RTI, the MOM can still provide useful federation and federate data in the connectionless mode. By using the RTI, this also means that other HLA federates or tools can participate in a DIS exercise.

CHAPTER THREE: METHODOLOGY

To this point, background research has been presented in the following areas:

- Computer Networking
- K-Array N-Cube Interconnects
- Clusterhead Leader Logic Algorithm
- Grid Computing
- Simulation Protocols

Computer Networking

The proposed research covers an understanding of different computer networking systems and protocols. All of the areas of simulation incorporate knowledge from computer networking. For the k-array n-cube interconnects, wormhole routing is used to route packets through the hardware interconnect. The CLL algorithm requires knowledge of wireless ad-hoc networks and the GPS-QHRA protocol. HLA involve applications of networking and an understanding of nuances of distributed computing such as routing and multicasting, and load balancing. Grid computing also requires knowledge of distributed computing, and in the case of the proposed research, the OSI network model and routing protocols.

K-Array N-Cube Interconnect Design

The objective for the k-array n-cube networks work is to find which k-array n-cube based interconnect architecture can be the best candidate to replace existing line card communication mechanisms, such as shared-bus or crossbars. Both shared-bus and the crossbar cannot scale well as the number of modules (PEs or memories) connected to it increases. In addition, the shared-bus requires a distributed arbitration mechanism, as the number of modules connected to it grows, thus, adding latency and space to the overall system. Pin constraints bound the bus size that can be interfaced with the NPU [26].

Hence, only a packet-based network-on-board can provide the required performance improvement between the NPU and off-chip memory modules. The work entails creating a simulation model that includes statistical data such as IP length distribution [27] and physical measures of PCB placement and spacing, as well as network properties such as IP packet size, in order to increase the accuracy of calculations. In addition, true IP network properties such as switching, propagation and routing latencies are applied. The simulator must provide real time performance analysis with detailed metrics on packets processed at each simulation cycle and overall detailed results at the end of each simulation.

The Simulation Architecture

The simulator architecture, shown in Figure 13, depicts the interconnect interaction with the control modules which adjust, collect and modify the interconnect settings, data flow, and performance metrics. These attributes are built in the functionalities of the modules. The simulator configuration manager sets the interconnect type, its properties (wire propagation delay, switching delay or routing delay) and enable/disable enhanced features such as channel width, VC on/off, and bi-directional channel.

The interconnect properties are set by the user interface and are recorded to allow the configuration manager to be updated via the worm manager. The worm manager utilizes interconnect properties and configuration parameters in order to set other modules accordingly in the system that participate in the simulation. The traffic sampler continuously records performance data such as throughput, latency, routing accuracy, interconnect bandwidth utilization and interconnect resources utilization. This information is fed back to the worm manager that adjusts worm generation rate and load balances the traffic. The routing algorithm receives each individual worm location and its destination node from the worm manager. Then, it determines the shortest route possible for each worm by avoiding spots of heavy traffic.

The worm jar is a storage module that contains worms. In the simulator there are two instances of the worm jar: one jar is for worms waiting to enter the interconnect and the other jar contains worms that are processed. The total number of worms during simulation are initially determined by the user. The scheduler is responsible to inject worms into the interconnect taking into account the total network capacity and traffic load. Since the worm manager knows the total number of worms that are modeled throughout the simulation, it must inform the scheduler at the end of the simulation when there are no more worms to model.

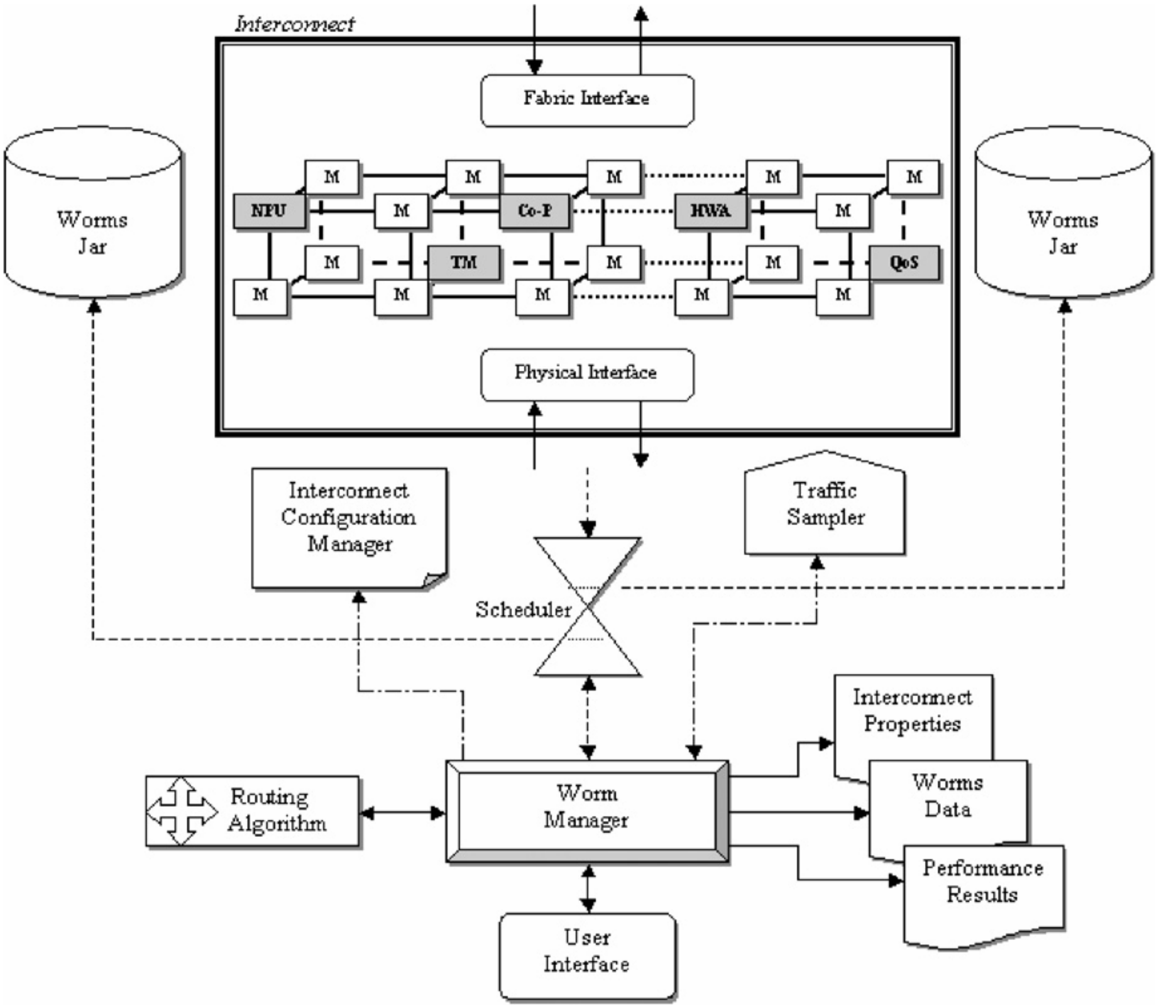


Figure 13 The K-Array N-Cube Simulator Architecture

The simulator accounts for all practical parameters characterizing off-chip interconnect architectures such as switching delays (T_s), routing delays (T_r) and propagation delays (T_w) as well as the complete functionality of each system components (nodes, links, PE/Memory, interfaces, virtual channels, and channel partitioning) [36]. The user has the option to change each of these parameters in case new technology introduces higher standards. Simulation time is based on a unit cycle that equals one clock cycle ($T_w + T_r$). All other delays are calculated as multiples of it; that provides the advantage of having single uniform simulation clock.

Message size in bytes and message generation-time are obtained by using pseudo-random number generator, which is utilized to resemble the randomness of packet transmission by both processors and memories. Each worm is linked to performance-bookkeeping function which records its latency, throughput, simulation cycles, failures, and route-taken from the moment the worm enters the interconnect until it completely reaches its destination. Comprehensive performance results are provided at the end of each simulation in a comma separated value spreadsheet.

The Simulation Modeling Approach

The high-level design of the simulator is comprised of four sets of C++ classes (Figure 14) supporting: the interconnect topology and configuration (Interconnect), the user interface (User Interface), the worm controller and administrator (WormManager), and worm structure and characteristics class (Worm). The worm contains a header field and data payload.

The Interconnect class represents the physical structure and includes all the hardware required to implement it. The properties represent two types of parameters: physical parameters of electrical components comprising the interconnect (such as wire delays, switching delays, routing delays), and parameters of additional features that enhance the interconnect performance (for example, channel partitioning, virtual channels, interconnect configuration). The simulator models the interconnect functionality in order to evaluate and compare different configurations and settings.

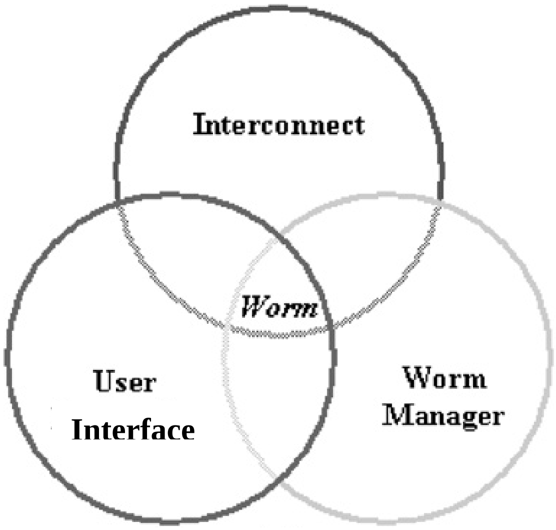


Figure 14 Major Class Relationships with Each Other and the User

Interconnect layout, of VCs and SC for example, affect worm routing flexibility and resources it can use while propagating through the interconnect. The Port class contains VCs and SCs which are modeled as logical topologies on top of the physical network architecture. VCs as well as SCs have a great effect on the worms transmission success/failure rates and deadlock/livelock avoidance. Although VCs improve routing accuracy and reduce worm transmission failure rate, they also increase the worm latency and interconnect implementation costs. The WormManager class records worm data, arrival and departure time stamps of worms, and controls the worm generation rate in order to load balance the number of worms processed simultaneously within the interconnect. The Worm class encapsulates the properties of a worm such as the header with source/destination fields and the route that the worm takes through the interconnect. The worm routes itself through the interconnect while continuously being monitored by the worm manager. The adaptive routing algorithm is used by the worm to determine the best available path that it can take to reach destination. The routing algorithm is derived and based on [37], [38], and [39]. The worm updates its shortest path coordinates with each movement to ensure its optimal path even when it is required to take a detour as a result of hot-spot node. Figure 15 shows a UML class diagram of the interconnect architecture [40][41]. A single type of interconnect is a set of faces which each contain multiple nodes. Within each node there are six ports. A node can be modeled as either a memory or a PE; in this case the node still possess the same structure and functionality as any node, but it reserves one port as an I/O port to the device.

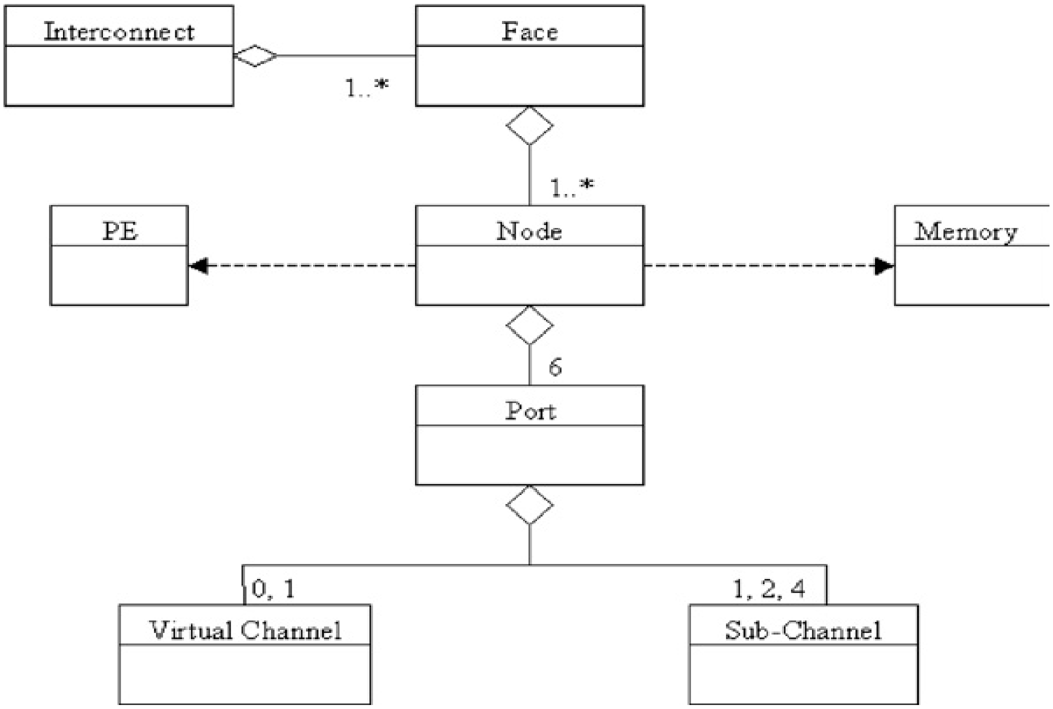


Figure 15 UML Class Diagram of the Interconnect

The simulation setup shown in Figure 16 is an abstract view of the high level system components and their interactions in order to initialize, execute, and complete the simulation. First, the user sets the simulation properties. These properties are crucial for worm generation, timing delays, and other simulation aspects. Then, the messages (worms) are created and are placed in a data structure (the Jar class). Since the interconnect configurations can be changed, PE and memory locations will be changed accordingly. Therefore, source/destination addresses must be correctly set before the worms can be generated.

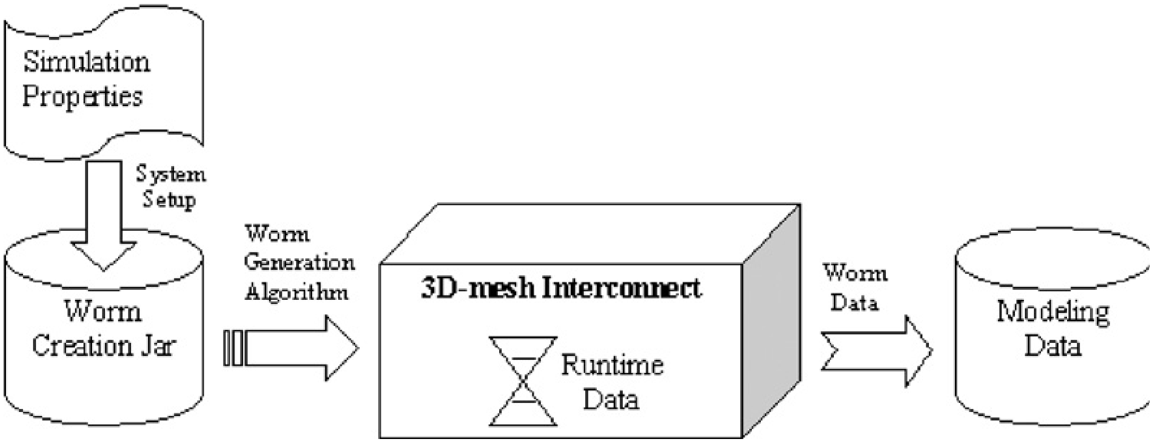


Figure 16 Process for Running the Simulator

The simulation properties are configured and the WormManager creates all of the worms needed and puts them in a jar. Then, when the simulation begins, the worms are picked up from the jar and are placed in the interconnect to route their way through. When the worms are complete, the WormManager places them in the worms modeled jar and then computes the modeling data.

When the user chooses to run the simulation, the properties and the data of the worms in the jar are recorded in separate files. The interconnect receives worms from the jar of generated worms according to a configurable probability called worm generation rate (GR). In addition, the user can determine the maximum number of worms that can occupy the interconnect at any one given time by changing the value of the MAX_WORMS_IN_INTERCONNECT variable (MWII). If no value is set for this variable, the default value is unlimited number of worms. The worms that enter the interconnect are modeled until they reach their destination.

All runtime worm data is collected in a separate output file that provides individual details about each worm. After the complete simulation is modeled, several spreadsheet files are generated recording the performance of the simulation.

Software Algorithms

Figure 17 portrays a dynamic model (action oriented) of the routing algorithm class and its subclasses with interconnect system components and the WormManager class. This model depicts the actions performed by the routing algorithm in order to maneuver each worm within the interconnect with respect to its current position, its destination and traffic conditions [42]. The routing algorithm is coupled with the worm manager since the worm manager controls worms entering and leaving the interconnect while the routing algorithm controls the worms within the interconnect.

First, the routing algorithm analyzes the source node type (where the worm is generated) and the enabled interconnect features such as virtual channels, bi-directional channels and PE–M configuration. Then, it checks the preferred (shortest path) direction in which the worm needs to move. The routing algorithm scans each node's port and dictates the movement of the worm giving priority to ports that are pointing in direction towards its destination. If none of the ports are available, the routing algorithm will check the availability of virtual channels. If enabled, the worm will be queued into one of the virtual channels until one of the ports clears. If virtual channels are not available then the routing algorithm notifies the worm manager of a worm routing failure. This will result in a retransmission of the same worm but statistics are kept to identify the failure.

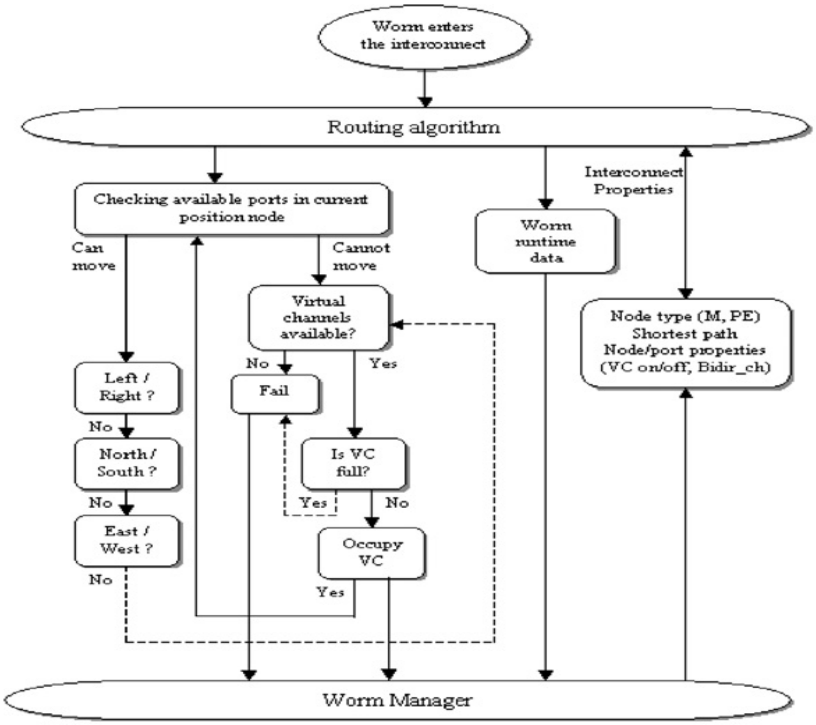


Figure 17 Dynamic Model of the Routing Algorithm Used

Figure 18 depicts a data flow diagram (DFD) of the user interface module. DFD charts assisted in determining what to automate in the simulator design and which data must be inputted exclusively by the user [42][43]. The user has two choices: using default settings or changing settings/properties in order to simulate the interconnect with different configuration. Once the interconnect type and configuration are defined, the user must complete the following steps before the simulation execution:

- Select if new worms will be generated or worms should be restored from an existing file.
- Determine the number of worms to simulate.
- Decide if worms are generated randomly or manually.

- Input the number of sampled throughput points (include the initial sampling point and the number of simulation cycles between samples).
- Select if the newly generated worms will be saved or not.

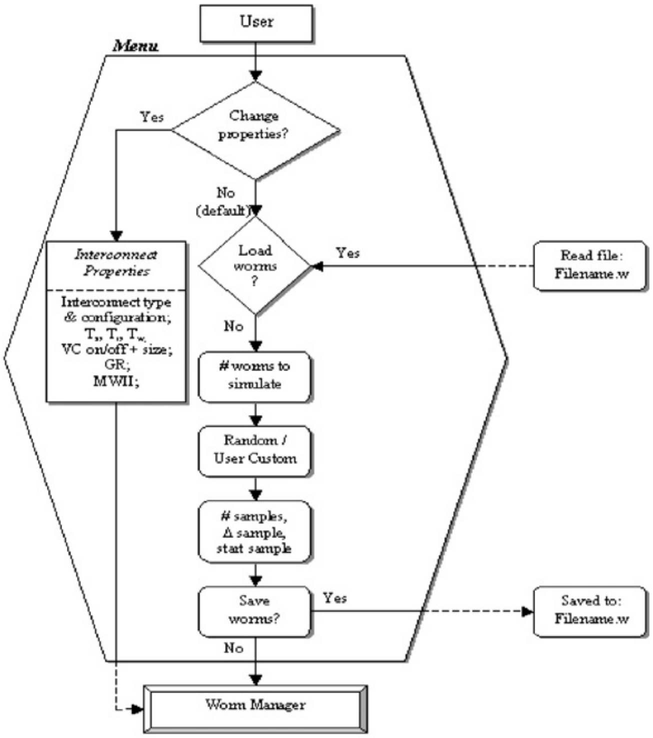


Figure 18 Data Flow Diagram of the Steps the Used to Start the Simulation.

Cluster Leader Logic Algorithm Design

This work proposes a new clustering algorithm for GPS-based mobile ad-hoc networks that takes into consideration the direction of the overall traffic flow in the network. The proposed cluster leader logic (CLL) algorithm is motivated by the GPS quorum hybrid routing algorithm (QHRA) where clusterheads react to changing data flow patterns of the network to provide better load balancing throughout the network using a new concept called cell fanning.

There are several key concepts used in the CLL algorithm which were built from GPS-QHRA which are summarized here:

- Dividing the area into cellular regions
- Establishing danger zones
- Maintaining inter-cell and intra-cell tables
- Assuming that nodes have GPS capabilities

Assumptions

In order for the CLL algorithm to work, some assumptions are made. As mentioned previously, all nodes must have positioning (GPS) capabilities that provide position information and clock synchronization. This is essential for a node to know which hexagonal grid it is located in. Also, this allows the CLL algorithm to measure where and how data traffic is changing. The accuracy of the positioning resolution is not so important for the sake of describing the algorithm; though the accuracy of the resolution affects the performance of the algorithm.

An important distinction from GPS-QHRA is the assumption that the cell sizes are at most one half of the distance of the transmission range between two adjacent nodes minus the width of the danger zone of a cell: ½ * largest_two_adjacent_node_distance – danger_zone_width. See Figure 19. This way, worst case, a clusterhead that is farthest away from the neighboring clusterhead can still communicate with that clusterhead. If cell sizes are smaller than the transmission range, then the algorithm will still work but the performance will degrade. This extra padding will allow for either fast moving nodes or instances when the cell zones are very small.

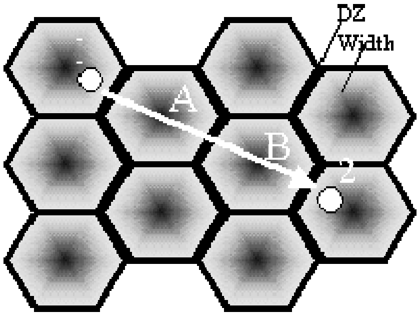


Figure 19 Danger Zone Width and Clusterhead Transmission Range

In Figure 19, Node 1 is the clusterhead for cell A and Node 2 is the clusterhead for cell B. The Danger Zone (DZ) width is shown for reference. This is a worst-case circumstance where the clusterheads are at farthest points in the danger zone - almost touching the next cell's safe zone. The diameter for any give cell should be at least 1/2 * distance$_{12}$ – DZ_width.

In addition to the cell size distinction, it is assumed that the messages transmit from an omni-directional antenna. A directional antenna could possibly provide some improvements [13]; but this work only focuses on free space omni-directional transmissions. An important assumption is that the routing algorithms or transport mechanisms used do not directly affect cluster leader election. For instance, the routing can be IP based, Geocasted [9], or routed from cell to cell. One assumption is that the clusterheads talk to other nodes or clusterheads in a single thread of execution. Also, it is assumed that subordinate nodes talk to clusterheads and clusterheads talk to other clusterheads and subordinate nodes. The final assumption is that all nodes have been pre-initialized to know the cell topology and their node identifiers.

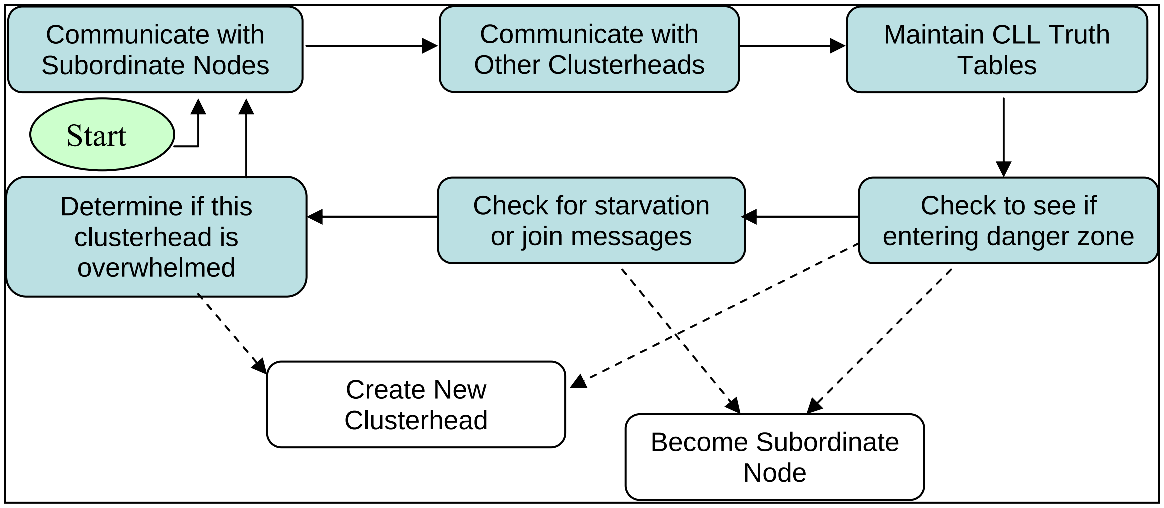
CLL Algorithm High Level Design

Before the details of the algorithm are discussed, it is important to understand the high level workings which surround the algorithm. The following figures give a context for the algorithm and the underlying mechanisms which make the algorithm work. Some aspects are taken for granted and are not covered (like routing needs) because this does not affect CLL.

```
void initialize()
{
  Nodes are turned on or enabled and clocks are synchronized
  Establish static cellular grid regions with danger zones
  All nodes are numbered
  Initial clusterheads are elected // For example using lowest id
  or highest degree of connectivity
  Wait for synchronized start signal // Nodes continuously listen
  when started
}
```

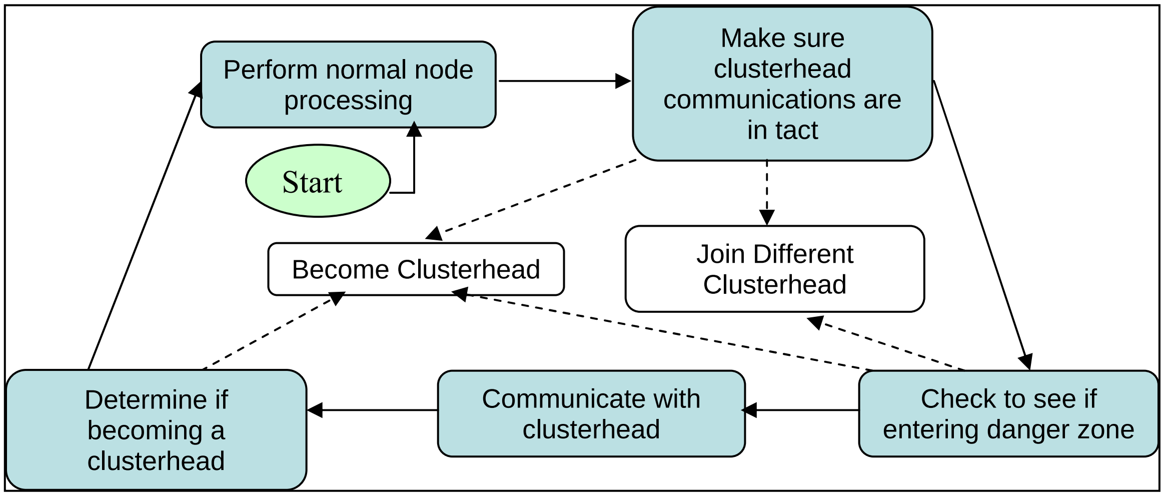
Figure 20 The Cluster Leader Election Algorithm Initialization Sequence

From a high level perspective, a designated master node initializes the network and nodes, loops until the nodes are ready to shutdown, then shuts down the simulator and logs statistics. First, in Figure 20, the initialization pseudo code is executed. If a node is a clusterhead, then the code performs as pictured in Figure 21. Otherwise, if the node is a subordinate node, then the code performs as depicted in Figure 22.



Communicate with Subordinate Nodes
Communicate with Other Clusterheads
Maintain CLL Truth Tables
Start
Determine if this clusterhead is overwhelmed
Check for starvation or join messages
Check to see if entering danger zone
Create New Clusterhead
Become Subordinate Node

Figure 21 The Cluster Leader High Level Design State Diagram



Perform normal node processing
Make sure clusterhead communications are in tact
Start
Become Clusterhead
Join Different Clusterhead
Determine if becoming a clusterhead
Communicate with clusterhead
Check to see if entering danger zone

Figure 22 The Subordinate Node High Level Design State Diagram

As a clusterhead, communication with the subordinate nodes is performed and CLL truth values are gathered (more on this later). The clusterhead checks to see if it is entering a danger zone; if so, then it must hand-off its clusterhead responsibilities, if necessary, and join or form a new cluster in the new cell. If the clusterhead is not in the danger zone, it tries to determine if it is starving for data or if it must acquire a new subordinate node. If it is starving, then it joins another clusterhead and negotiates its subordinate nodes to that clusterhead. Otherwise, if the clusterhead is overwhelmed, the final step is to determine if it should split its duties with a new clusterhead based on the CLL truth values it perceives.

As a subordinate node, the node begins by performing normal tasks. Routinely, clusterhead communications are checked. If communications are bad or if designated, then it can join a new clusterhead or become one. Similar to the clusterhead algorithm, the subordinate node checks to see if it is in a danger zone. If not, then it communicates with the clusterhead. The clusterhead will let its subordinate know if it should become a clusterhead.

Algorithm Detailed Design

Up to this point, the high level simulator design was described to show how the algorithm can fit in the context of clusterhead networks. In order to understand the algorithm detail design, the variables and data structures are first explained and then the algorithm is introduced.

Messages

In order to establish, transfer, or decommission clusterheads, there are several messages which communicate essential parameters which are outlined in Table 5. The ClusterheadElectionAck and ClusterheadJoinAck messages contain an acknowledgement Boolean flag where acknowledge is true and decline is false. The TruthValuesAck message itself is an acknowledgement; so receiving this message constitutes the acknowledgement.

Table 5

Messages Used in the CLL Algorithm

Message Type	Message Description
RequestClustheadChange	A clusterhead sends this message when it determines that either an additional clusterhead is needed in a cell or another node needs to take its place in the cell. The message is sent to a specific node that the clusterhead finds to be a suitable clusterhead candidate.
ClusterheadElectionAck	A node sends this message back to the originating clusterhead when it accepts or rejects becoming a clusterhead leader.
JoinClusterhead	A node sends this message to neighboring clusterheads when it needs to join another clusterhead. This could be from a circumstance when a clusterhead has no subordinate nodes.
ClusterheadJoinAck	Either a clusterhead sends this message to a node indicating that it can or cannot support this node as a subordinate node or a node sends this to a clusterhead acknowledging that it accepts or denies joining its cluster.
TransferTruthValues	A clusterhead send this message to a node to notify it of its truth-telling data traffic behavior.
TruthValuesAck	A node sends this message acknowledging receipt of a TransferTruthValues message.

Variables

There are several static constant variables that are configured prior to initialization of the network for the CLL algorithm. The idea of making these variables static for distributed computing means that each node has a copy of the same values. Also, making a variable constant means that the value of the variable cannot change.

Most important for the CLL algorithm are the variables that represent the truth weights. These variables are neither static nor constant. The CLL algorithm has persistent truth weights, $PTWeight_{Dir}$ and $GTWeight_{Dir}$, and temporary weights, $PTTransmissionFreq_{Dir}$ and $GTTransmissionFreq_{Dir}$.

Table 6

CLL Constants

Simulation Constants	Description
MaxClusterheadsPerCell	Data distribution is based on a divide-and-conquer approach. This variable controls when the CLL algorithm can divide a cell between multiple clusterheads and how many divisions can occur per cell.
ClusterheadDivisionTruthThreshold	Indicates the threshold of the number of effective subordinate nodes a clusterhead can maintain.
PTTimeout	Perceived table entries do not persist forever. This variable controls the limit when a PTWeight value becomes stale and when the weights are updated with the latest traffic information.
GTTimeout	This variable helps control when a GTWeight value is updated with the latest traffic updates.
GTWeighingFactor	Designates how important to make the weighing calculations for determining subordinate node transmission factors.
PTWeighingFactor	Designates how important to make the weighing calculations for determining neighboring cell transmission factors.
StartingWeight	Designates what value the weights should start at.
PurgeWeightsWhenCHSplit	Has a true or false value. When clusterheads (CH) split, this determines if truth weights should be transferred to the new clusterhead or if the weights should be purged instead of being transferred.

Ground truth represents accurate knowledge that a clusterhead has about traffic density in its current cell and the transmissions that start from or end at its cell; perceived truth represents the clusterhead's best guess at what the traffic looks like in cells surrounding it based on transmissions that are hopped through its cell. Knowledge of traffic density is used to weigh whether or not a clusterhead should split its load with a new clusterhead, become a subordinate node to another clusterhead, or maintain its status as a clusterhead. The next section on data flow tables walks through a ground/perceived truth example and explains how the values are used and differ from each other.

Table 7

CLL Simulation Variables

Simulation Variables	Description
$\text{PTWeight}_{\text{Dir}}$	A positive real number on a scale of 1 to 100. This is the perceived truth (PT) weighing factor and it is initialized to StartingWeight. A clusterhead has independent PTWeight variables, one for each neighboring cell. As messages are forwarded from one cell to another, the weight is adjusted using an exponential mean average.
$\text{GTWeight}_{\text{Dir}}$	A positive real number representing a ground truth (GT) weighing factor, from 1 to 100, initialized to StartingWeight. A clusterhead has independent GTWeight variables, one for each neighboring cell and one for its cell. As messages are transmitted to or received from subordinate nodes, the weight is adjusted using an exponential mean average.
$\text{GTTransmissionFreq}_{\text{Dir}}$	The temporary number of ground truth transmissions which sets purged each time an EffectiveNodeCountGT calculation is done. Each node has a transmission frequency for each direction capable of transmitting to.
$\text{GTTransmissionFreq}_{\text{Dir}}$	The temporary number of perceived truth transmissions which gets purged each time an EffectiveNodeCountPT calculation is done. Each node has a transmission frequency for each direction capable of transmitting to.

Note that both the $\text{PTWeight}_{\text{Dir}}$ and $\text{GTWeight}_{\text{Dir}}$ variables are adjusted using an exponential mean average (EMA) [53] shown in Equation 3 (ground truth exponential mean average equation) and Equation 4 (percieved truth exponential mean average equation). Note that Equation 1 has the alpha value used in the GT EMA equation and Equation 2 has the alpha value used in the PT EMA equation. The EMA was chosen because brief spikes in network traffic influence the result as little as possible; the EMA lags behind the actual trend and prevents over-reacting. Also, since the timeouts which clear the tables are constantly occurring, it helps to balance the symmetry between increasing and decreasing the weight values; this also helps for the weights to converge if traffic stabilizes.

$$\alpha_{GT} = \frac{2}{(1 + GTTimeout\ )} \qquad \text{(Equation 1)}$$

$$\alpha_{PT} = \frac{2}{(1 + PTTimeout\ )} \qquad \text{(Equation 2)}$$

$$GT\_EMA_i = (1 - \alpha_{GT})EMA_{i-1} + \alpha_{GT} * Freq \qquad \text{(Equation 3)}$$

$$PT\_EMA_i = (1 - \alpha_{PT})EMA_{i-1} + \alpha_{PT} * Freq \qquad \text{(Equation 4)}$$

Each GTWeight$_{\mathrm{Dir}}$ and PTWeight$_{\mathrm{Dir}}$ has an independent EMA allocated for it. As mentioned earlier, there is one weight for each of the six directions and the weight of the intra-zone messages direction is used as the seventh GTWeight$_{\mathrm{Dir}}$. The frequency of message transmissions or receptions in Equation 5 (ground truth message frequency) is the summation of transmissions in a particular direction based on the CLL tables. For instance, GTFreq$_{\mathrm{UP}}$ = 4 if the ground truth table has four entries for data flowing up. See Equation 5 for calculating ground truth frequencies and Equation 6 (perceived truth message frequency) for calculating perceived truth frequencies. Please note that the frequencies help determine how often messages travel in a particular direction (not to be confused with transmission tuning frequencies).

$$GTFreq_{dir} = \sum_{AllGTTableEntries} TableEntry_{dir}$$
$$dir = \begin{Bmatrix} \text{up, up - right, down - right,} \\ \text{down, down - left, up - left,} \\ \text{intra - cell} \end{Bmatrix} \qquad \text{(Equation 5)}$$

$$PTFreq_{dir} = \sum_{AllPTTableEntries} TableEntry_{dir}$$
$$dir = \begin{Bmatrix} \text{up, up - right, down - right,} \\ \text{down, down - left, up - left} \end{Bmatrix} \qquad \text{(Equation 6)}$$

Values are also based on truth: ground or perceived truth. Thus, α for ground truth will use GTTimeout and α for perceived truth will use PTTimeout. As the timeout value increases, more stress is placed on older values compared to newer values.

Data Flow Tables

Each clusterhead maintains two data flow tables for network characteristics. Both tables have the same column headings, but the tables themselves have two different purposes and are populated and unpopulated using different heuristics. The table format is specified in Table 8.

Table 8

CLL Ground Truth and Perceived Truth Table Format

Time	Traffic Direction	Destination Id
Time of message forwarding or receipt	Up, Down, Up-Left, Up-Right, Down-Left, Down-Right, and Intra-cell (GT Only)	Destination node id

Note that the time recorded is either the time the message is forwarded or when the destination node receives the message.

The first column of Table 8 designates the time either that a message is forwarded or the time the message is received at the last hop. The second column is the direction that the message is traveling. There is no need to record intra-zone transmissions or receptions for perceived truth because perceived truth only applies to messages hopping through a cell. The final column is the destination node id. Please note that the bit-width and ranges of the destination node ids, time, and other variables is implementation dependent. The simulation created used unsigned long values for destination node id and time with a range of [0, UNSIGNED_LONG_MAX].

There are two instances of table maintained for CLL by each node: a ground truth table and a perceived truth table. Both tables count any type of message including messages that are retransmitted due to failure. The ground truth table represents two types of factual transmissions:

- Messages that emanated from a clusterhead's subordinate nodes or the clusterhead.
- Messages received by a cell's clusterhead for either itself or a subordinate node.

The perceived truth table represents communications that are forwarded on behalf of a clusterhead's cell to another cell. The purpose of recording perceived truth data applies to data flowing to or from neighboring cells only. Thus, an individual node can estimate traffic load in other directions, but these estimations are not factual because there could be transmission occuring that a neighboring node may not know about.

The perceived truth tables can vary very differently from the ground truth data flow tables. This is the key for the CLL algorithm: there is no desire to have ground truth global knowledge of the entire network data flow. Each clusterhead only cares about the transmission characteristics through its cell and around its cell. It is hypothesized that having ground truth knowledge of the entire network could actually degrade data flow performance of the CLL algorithm.

Figure 23 shows two simultaneous message transmissions starting at time 1: one from node 1 to node 2, the other from node 3 to node 4. White circles represent subordinate nodes and dark circles represent clusterheads. These nodes are mobile, so the clusterheads are free to move about their cells as long as they stay within the danger zone of their respective cell. Assuming the data flow tables are empty before transmission, the clusterheads in cells {A, D, E} modify their ground truth tables. Clusterheads in cells {B, C} modify their perceived truth tables. Three example table entries are shown in Table 9, Table 10, and Table 11. Since clusterhead 7 is forwarding the message from cell A to cell D, it perceives the terminating cell to be cell C even though the transmission is through cell C to clusterhead 8. As a reminder, the clusterheads may be mobile and they may not be centrally located in a cell. Clusterheads on the periphery fall into the danger zone and will become subordinate nodes if another clusterhead is around for it to join with.

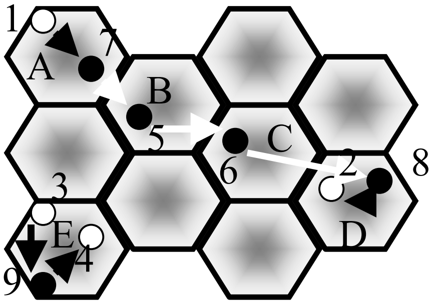


Figure 23 Two Simultaneous Message Transmissions; Nodes are Numbered Circles.

Table 9

Clusterhead 7's Ground Truth Table Entry

Time	Traffic Direction	Destination Id
1	Down-Right	2

Based on the communications in Figure 23.

Table 10

Clusterhead 5's Perceived Truth Table Entry

Time	Traffic Direction	Destination Id
2	Down-Right	2

Based on the communications in Figure 23.

Table 11

Clusterhead 9’s Ground Truth Table Entry

Time	Traffic Direction	Destination Id
1	Intra-Cell	4

Based on the message flow in Figure 23.

Dark circles are clusterheads, white circles are subordinate nodes. Dark arrows represent clusterhead-node transmissions, light arrows represent node-to-node forwarded transmissions. One message travels from cell A to cell D, the other message starts and ends in cell E. Clusterheads in cells {A, D, E} modify ground truth data flow tables. Clusterheads in cells {B, C} modify perceived truth data flow tables. Since the clusterhead is forwarding the message from cell A to cell D, it perceives the terminating cell to be cell C even though the transmission going to node 2 through cell C.

Load Balancing and Algorithm Execution

Now consider how CLL truth data is used to achieve load balancing. The CLL algorithm is composed of three separate functions: initialize(), updateTables(), and process(). The initialize() function Figure 24, called in the high level design initialize function, sets up the internal variables needed for the CLL algorithm and sets timers for the values of GTTimeout and PTTimeout. The updateTables() function (Figure 25) will update the GT or PT tables with a new row of information. The information includes data gathered about whether a clusterhead has transmitted any information and whether the transmission affects perceived or ground truth.

```
void initialize()
{
   setInitialGTTimeout();
   setInitialPTTimeout();
}
```

Figure 24 The CLL initialize() Function Sets the Initial GT and PT Timers

```
void updateTables()
{
   if (isForwardedMessage())
   {
      updatePTTable();
   }
   else
   {
      updateGTTable();
   }
}
```

Figure 25 The CLL updateTables() Function

The process() function takes the updated data and adjusts the weight values for the clusterhead depending on whether a GT or PT timeout has been received. The process() function also purges stale PT data before weights are adjusted. Most important, the process() function calculates effective node counts and splits a clusterhead if necessary.

In order to determine when to create a new clusterhead, the algorithm calculates the EffectiveNodeCount. This counts subordinate nodes that transmit and receive data based on their weights and adds the perceived truth weights based on their weights as well. This value is then compared to the ClusterheadDivisionActivationLevel and a clusterhead will split its load when this value is exceeded.

```
void process()
{
   if (isGTTimeout())
   {
      adjustGTWeights();
   }
   else
   {
      purgeOldPTEntries(current_time – pt_timer_length);
      adjustPTWeights();
   }

   if (getEffectiveNodeCount() > ClusterheadDivisionActivationLevel and
      NumClusterheadsInCell < MaxClusterheadsPerCell)
   {
      createNewClusterhead();
   }

   resetTimer()
}
```

Figure 26 The CLL process() Function is Called when Timeouts Occur

When each weight is adjusted according to Equations 3 and 4, the data flow tables are evaluated by summing the frequencies for each direction as shown in Equations 5 and 6. The EffectiveNodeCount is calculated in Equation 7 that is based on Equation 8 (ground truth EffectiveNodeCount) plus Equation 9 (perceived truth EffectiveNodeCount).

$$\text{EffectiveNodeCount} = \text{EffectiveNodeCount}_{GT} + \text{EffectiveNodeCount}_{PT} \quad \text{(Equation 7)}$$

$$\text{EffectiveNodeCount}_{GT} = \sum_{\text{All Directions}} \begin{pmatrix} \text{WeighingFactor}_{GT} \times \\ 0.01 \times \text{LearnedWeight}_{dir} \times \\ \text{TransmissionFreq}_{dir} \end{pmatrix} \quad \text{(Equation 8)}$$

$$\text{EffectiveNodeCount}_{\text{PT}} = \sum_{\text{All Directions}} \begin{pmatrix} \text{WeighingFactor}_{\text{PT}} \times \\ 0.01 \times \text{LearnedWeight}_{\text{dir}} \times \\ \text{TransmissionFreq}_{\text{dir}} \end{pmatrix} \qquad \text{(Equation 9)}$$

As a reminder, the variables in these formulas are described in Table 6 and Table 7. $\text{GTWeight}_{\text{Dir}}$ and $\text{PTWeight}_{\text{Dir}}$ vary between 1 and 100; they are multiplied by 0.01 to make this value a percentage. The ground truth transmission frequencies in Formula 7 represent all of the frequencies recorded for a particular cell if PurgeWeightsWhenCHSplit is false. If PurgeWeightsWhenCHSplit is true, then each time a clusterhead splits, the ground truth value is erased; this option prevents over-reacting to sudden data flow changes. Also, it contains the growth of the tables to consuming too much memory.

The perceived truth transmission frequencies account for transmission frequencies since the last PTTimeout occurred. If no messages were hopped in this cell, then $\text{EffectiveNodeCount}_{\text{PT}}$ is zero.

Cell Fanning

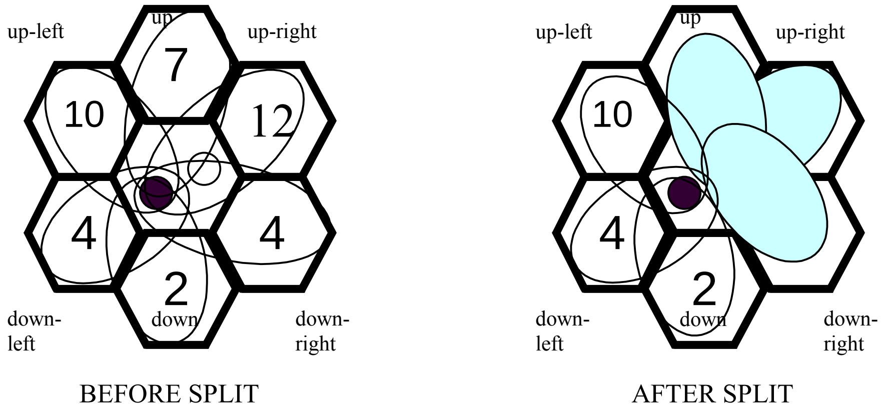
A new concept called cell fanning is introduced for the clusterheads. For traditional cellular architectures, techniques like cell sectoring and cell splitting [54] can be used to transmit signals directionally or limit signal transmission for different frequencies. Since, for this study, no assumption is made about directional antennas or multi-frequency transmission capabilities, cell sectoring and cell splitting techniques are not considered. However, it would be useful to assign directions of responsibility for clusterheads within a cell to share the workload fairly.

Cell fanning allows a clusterhead to split its workload with another clusterhead in its cell which prevents the original clusterhead from becoming overloaded and the new clusterhead becoming starved for data transmissions. This is achieved by designating, via round-robin mechanism, which clusterhead will forward messages to other cells within a fan-out pattern. Collocated clusterheads within one cell can share the transmissions to other cells by transmitting data to their designated set of adjacent cells.

Due to the cell distance constraints, all clusterheads within one cell distance should hear a message that needs to be forwarded. Only one receiving clusterhead in a cell will forward the message however. This receiver-side filtering is determined by the cell fans that the sending clusterhead is assigned. In other words, the receiving clusterhead determines if the message is for its cell and either processes it or drops it if it is meant for another clusterhead.

Considering Figure 27. When there was one clusterhead (dark circle) in the center cell, its cell fan set included cells in all directions. However, when the load was high for one clusterhead, the clusterhead split its duties with another subordinate node. The numbers in the adjacent cells represent the EffectiveNodeCount of those cells. When the clusterhead splits its duties, it fans in a round-robin fashion based on the frequencies of the numbers of transmissions in descending order. The result is shown where one clusterhead will filter data transmission to the bottom left cell fan and the other clusterhead will filter data transmission to the top right cell fan.

For example, if a clusterhead has the cell fan set (‘up’, ‘up-right’, ‘down’) and the clusterhead receives a message which is destined upward according to the routing algorithm used, then this clusterhead will relay the message to the clusterhead in the cell above it because ‘up’ is in the cell fan set. If, however, the same clusterhead receives a message destined for ‘down-right’, then the clusterhead does not relay the message. Note the term not relaying is not the same as a packet being dropped. Not relaying a message implies another node collocated in the same cell should relay the message. Dropping a message means that a message was not able to reach its destination.



BEFORE SPLIT

AFTER SPLIT

Figure 27 Cell Fanning Example – Before (Left) and After (Right)

Even though the clusterhead in the previous example forwarded the message ‘upward’, all clusterheads in all adjacent directions may hear the message if omni-directional antennas are used. However, only the ’upward’ adjacent clusterheads will react to the message; the other clusterheads will filter out the message.

Grid Resource Discovery Protocol Design

The additional focus of this work is to create an ad-hoc grid resource discovery protocol. This protocol will find computing resources on the Internet without the need for dedicated servers to track existing clients on the Internet. In the real world, this can be implemented inside of custom networking hardware or programmable networking hardware by introducing a new protocol layer on the OSI network stack just above the network layer [24]. The hardware would maintain resource tables that can help make efficient use of the grid computing resources. Since it is not feasible to create and deploy this hardware over the Internet to create of grid computing network of thousands of computers, there is a need to build a simulator to model this environment to test the feasibility of the algorithm and to find the best parameters for the algorithm to operate within.

Although it is feasible that this simulation can be built using NS-2 [23], since the K-Array N-Cube simulator has been built from the ground up and the CLL Simulator has shared some parts of that simulator, there is motivation to create a simulation architecture based on the previous work done. Both of the previous simulators also model network traffic. This work can be done in a fashion where a simulation engine can be built which other simulators can be built from in the future. Unlike NS-2, this simulation engine could be more generic to simulate other non-networking related models.

Protocol Design

This section details the protocol design that considers the lifecycle of the resource providers, the events exchanged over the network, the structure of the data tables used inside the routers, and the technique used for scoring resource providers. Also, the responsibilities are reviewed for the resource provider, the router, and the VO host to include a description of how data tables are modified and the conditions needed to send events.

Lifecycle

There are five phases involved in the lifetime of a grid resource provider: subscription, advertisement, transaction, sign-off, and retirement. See Figure 28 for a lifecycle view of the different phases. Before the subscription phase, the resource provider acquired software from the VO (introduced on page 21). During the subscription phase, the resource provider is subscribed to the VO's list of resource providers. During this transaction, an account is setup that includes ways for the VO to track the trustworthiness of the resource provider.

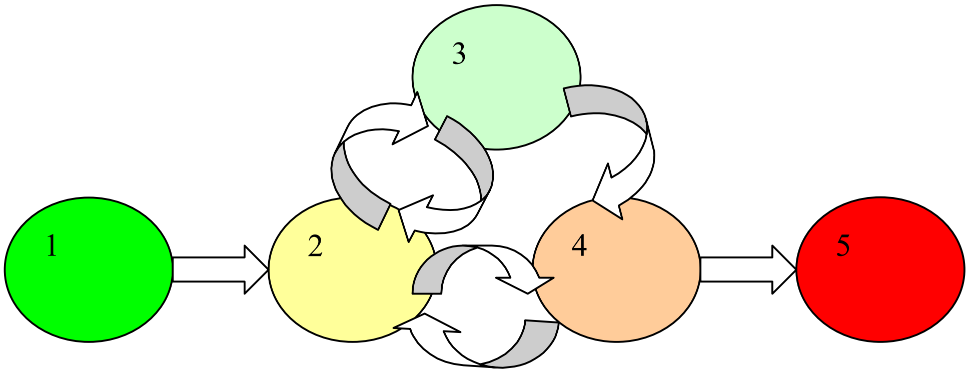


Figure 28 The Lifecycle of a Grid Resource Provider has Five Phases: 1) Subscription, 2) Advertisement, 3) Transaction, 4) Sign-off, and 5) Retirement

The second phase, advertisement, is when the grid resources advertise their availability to the grid. Information sent to the grid includes any statistical information necessary for the grid to facilitate tasks. The information includes the following fields:

- VO Memberships (The VO would track the software packages)
- Number of CPUs
- Available CPU Speed
- Available Memory/Disk
- Network Connection Speed

The transaction, the third phase, is when the actual task is delivered to, computed on, and published from the resource providers. For the sign-off phase, the grid is notified that the resource is unavailable for an unknown period of time. This differs from the retirement phase because when the resource retires, it may never rejoin the network again. Resources can either retire because they want or need to base on their own assessment or they can retire because their trustworthiness rating is poor and the VO kicks them off of the network.

Event Header

Table 12

Event Header Data Variables

Variable Name	Variable Data Type	Size in Bytes
path	List of IP Addresses	0…n (multiple of 4 bytes)
path_index	Unsigned Byte	1
path_size/score	Unsigned Byte	1
start_address	IP Address	4
end_address/score	IP Address	4/1 (*score* used when *routing_type* = DISCOVERY)
event_type	Unsigned Nibble	½
routing_type	Unsigned Nibble	½
original_time	Unsigned Long Long	8
event_id	Unsigned Long	4

IPv4 addressing is assumed when noting the IP address sizes.

The grid resource discovery protocol sends various types of events through the network that are introduced in Table 12. Each event has a common event header. The *path* variable is a variable length list of IP addresses of *path_size* length and the *path_index* is used to point to the next destination IP address in the *path*. Two other IP addresses, the *start_address* and *end_address* are populated when possible to designate the origin and destination of the event. The *event_type* identifies the type of event represented by the data and the *routing_type* indicates how to route the event to the grid protocol software handling the event. When the *routing_type* is set to DISCOVERY, the *end_address* is used as a *score* variable. The *score* represents the score of the resource provider being sought. Two other fields, *original_time* and *event_id* are populated from the originating device for use with tracking the event at its destination.

Routing Techniques

There are several different routing techniques used by the grid protocol design. The routing techniques describe where the routers should direct each event based on the event type and are outlined in Table 13.

Table 13

Routing Techniques

Routing Technique	Description
STANDARD	The events travel through the network the same way they would in a normal TCP/IP environment. Each hop IP address is stored in the event path storage field.
FORWARD PATH	Events are passed through the network according the path stored in the event.
REVERSE PATH	Events are passed through the network according the reverse order of the path stored in the event.
DISCOVERY	Events hop between routers based on a scoring scheme. Each hop IP address is stored in the event path storage field.

The STANDARD routing type applies to events that are directed through the network using TCP/IP routing. The first entry in the path is populated at the origin device of the event and the *path_index* variable is set to 1. When a grid protocol event arrives at a router capable of handling grid protocol events, the event is passed to the hardware which handles the protocol. The router then places the current IP address of the router inside the event's path structure, then increments the *path_index* variable by 1. When the event traverses the entire route, the final device adds its IP address to the path and the entire path is available to other events which will be constructed from this event.

The *path* variable plays an important role for this protocol. The reason why the path is cached is because the IP routing protocol does not guarantee that the path used to send a event one time to a destination will be the same path used to send the event again to the same destination. Also, IP routing does not guarantee that the path the event takes to the destination will be the same path the event will take on the way back. The path allows the protocol to update specific resource tables within the network along the same path each time. Otherwise, an event that arrives at the wrong router may not know how to direct a event or it may drop the event if it is not authorized.

The FORWARD PATH routing scheme uses the path learned from the STANDARD routing scheme to move an event from the IP address in the beginning of the event to the IP address at the end of the event. The event is sent from the origin (the first entry in the *path*) to the second entry in the *path* and the *path_index* is set to 2 (or 1 for a zero based array). Each time the event arrives at a hop recorded in the *path*, the *path_index* is incremented by one and the event is sent to the next hop in the path. The event arrives at its destination when the *path_index* equals the index of the final element in the *path*.

The REVERSE PATH routing scheme is similar to the FORWARD PATH routing scheme except that the event travels from the final destination in the *path* to the original destination. Also, the *path_index* initially points to the last address in the *path* and is decremented to the origin address in the *path*.

The DISCOVERY routing scheme attempts to find a resource in the network by using a one-byte *score* variable. The *score* variable is part of the event header when the routing type is set to DISCOVERY. The *score* variable replaces the *end_address* because the event is attempting to discover the end address. Similar to STANDARD routing, the path is learned for the DISCOVERY scheme as well. The *score* value may change between hops, so the path taken by the event may not be the same path that the STANDARD event took when traveling between the resource provider and the VO host computer.

Events

Over the course of the lifecycle, many events are exchanged over the network to advertise resource availability, update router data tables, and maintain the security of the network. The events sent during this protocol map to a lifecycle phase as shown in Figure 29. If a router or VO host receives an event out of order, it either drops or forwards the event and notes an entry in the blacklist. The following events are associated with designated lifecycle stage as shown in Table 14.

Table 14

Events Used by the Grid Resource Discovery Protocol

Lifecycle Phase	Event	Description
Subscription	SIGNUP	Signup a resource provider to the grid.
	ACCEPT	VO host accepts the resource provider.
Advertisement	ADVERTISE	The resource provider advertises its availability.
Transaction	TASK	The VO host wants to discover a resource provider.
	TASK COMPLETE	The resource provider finished completing a task.
	CONFIRM DELIVERY	VO host acknowledges the deliver of the task data.
	CONFIRM TRANSACTION	Resource provider acknowledges receipt of the data by the VO host.
	TASK UNSATISFIED	The TASK event could not discover a resource with the score sought.
Sign-off	GOODBYE	A resource provider is not available to the grid.
Retirement	UNSUBSCRIBE	The resource provider wished to leave the VO's grid network.

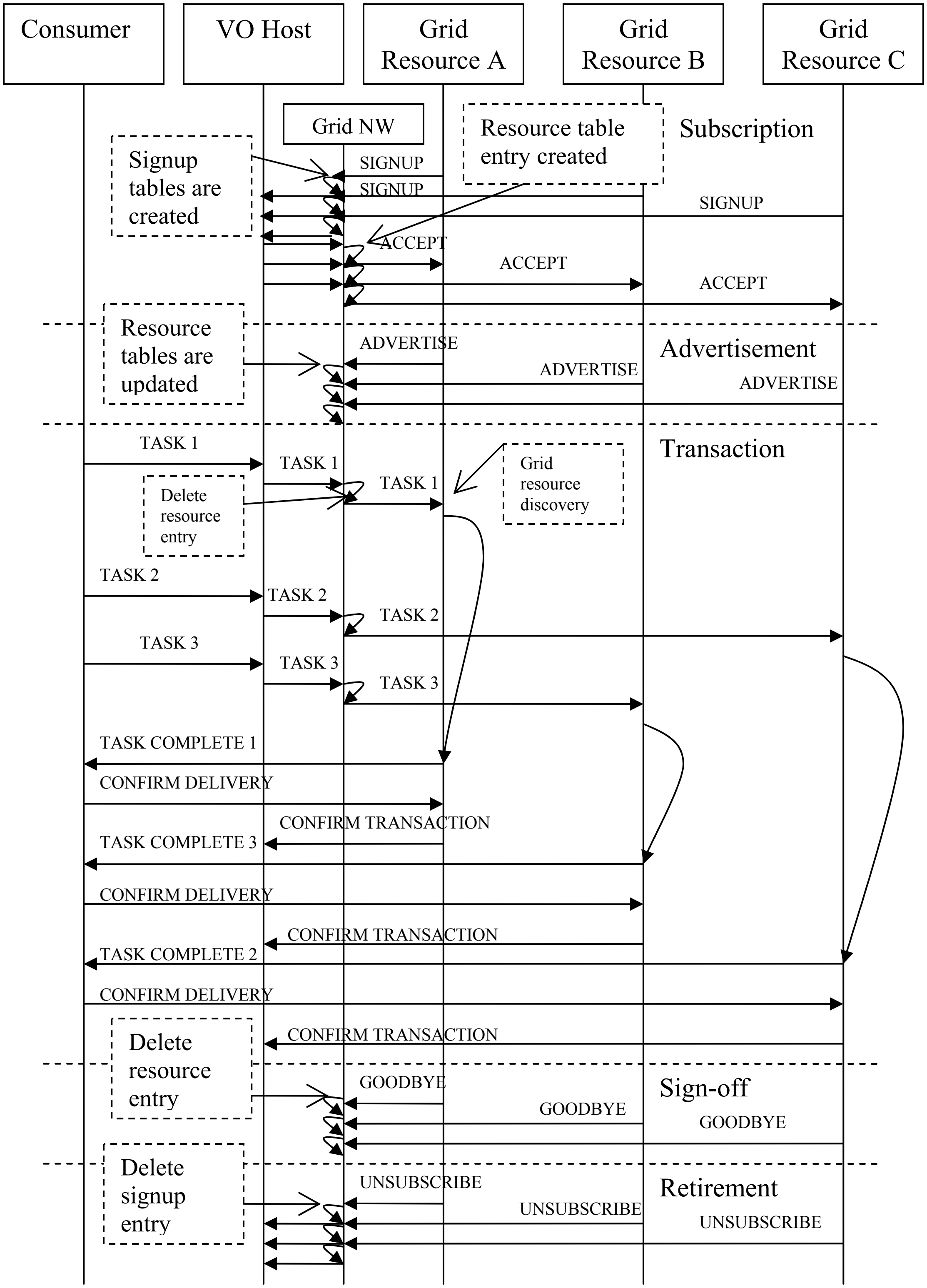


Figure 29 Events Exchanged over the Network

SIGNUP Event

The SIGNUP event allows a resource provider to sign up to the VO and provide its resource to the grid. The event is sent the first time a resource participates on the network and periodically every 24 hours to update the SIGNUP table. It uses the STANDARD routing technique populated with a unique *event_id*, traveling from a resource provider to a VO host, and requires a *score* variable in its payload in addition to the variables in the event header. When a SIGNUP event arrives at a router, it records information in its SIGNUP and BLACKLIST tables. A router may reject a SIGNUP event if the resource has been blacklisted, but generally the signup event is recorded and the resource is awaiting acceptance from the VO.

An example SIGNUP event transaction is shown in Figure 30. The event originates in the switch in the bottom of the diagram and makes its way through the network until it reaches its VO host destination. Note that the simulator aggregates resource providers into a switch (which is why the resource is not shown).

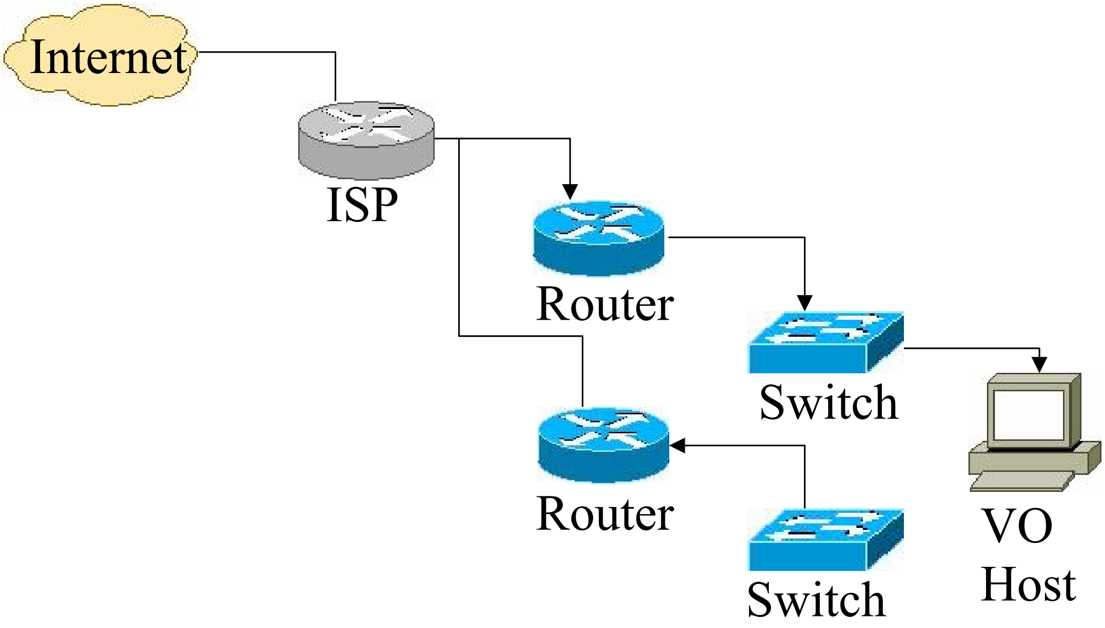


Figure 30 SIGNUP Event Standard Routing Example

ACCEPT Event

The ACCEPT event notifies the routers to accept the resource in its resource tables if it receives an ADVERTISE event. It uses the REVERSE PATH routing technique with the path learned from the SIGNUP event populated with the same *event_id* as used in the SIGNUP event, traveling from the VO to the resource provider, and requires an *accepted* Boolean flag in its payload designating when the event is accepted or not. The SIGNUP table marks the event as accepted, and the event is erased from the BLACKLIST table. As shown in Figure 31, the VO sends the ACCEPT event back to the resource provider (located in the switch).

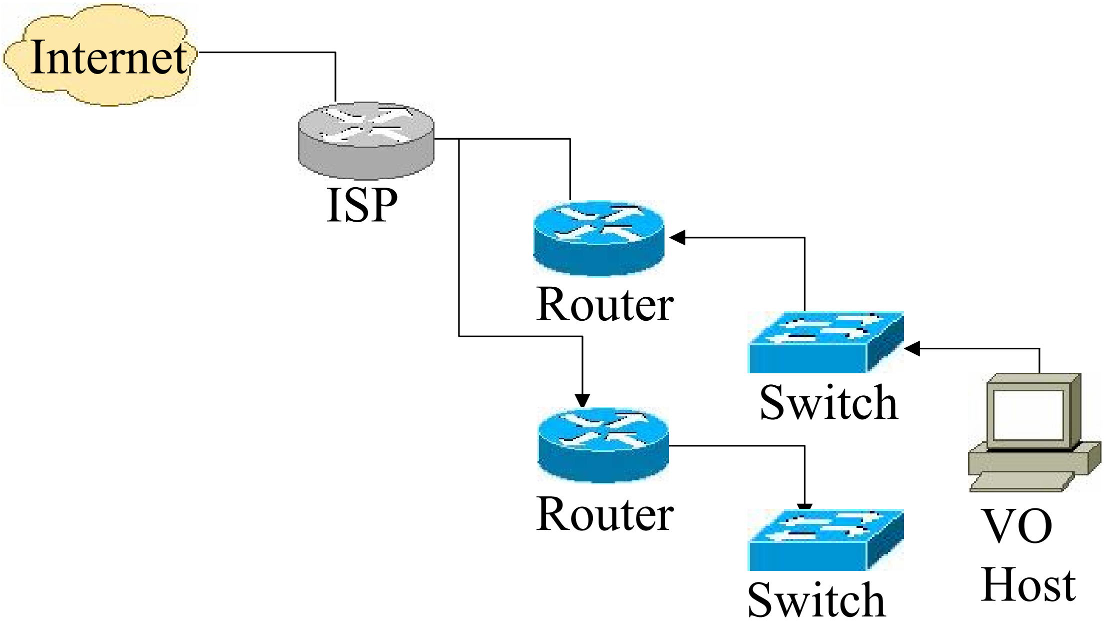


Figure 31 ACCEPT Event Reverse Path Routing Example

ADVERTISE Event

The ADVERTISE event advertises the availability of a resource and allows TASK events to discover the resource. It uses the FORWARD PATH routing technique with the path learned from the SIGNUP event populated with a unique *event_id*, traveling from the resource provider to the VO. The ADVERTISE event does not have any additional data in its payload. If the resource provider is signed up and accepted in the router, the ADVERTISE event signals the router to populate the RESOURCE TABLE. As shown in Figure 32, the resource provider in the switch advertises its resources to the router closest to the VO host. The VO host is not notified of the advertisement.

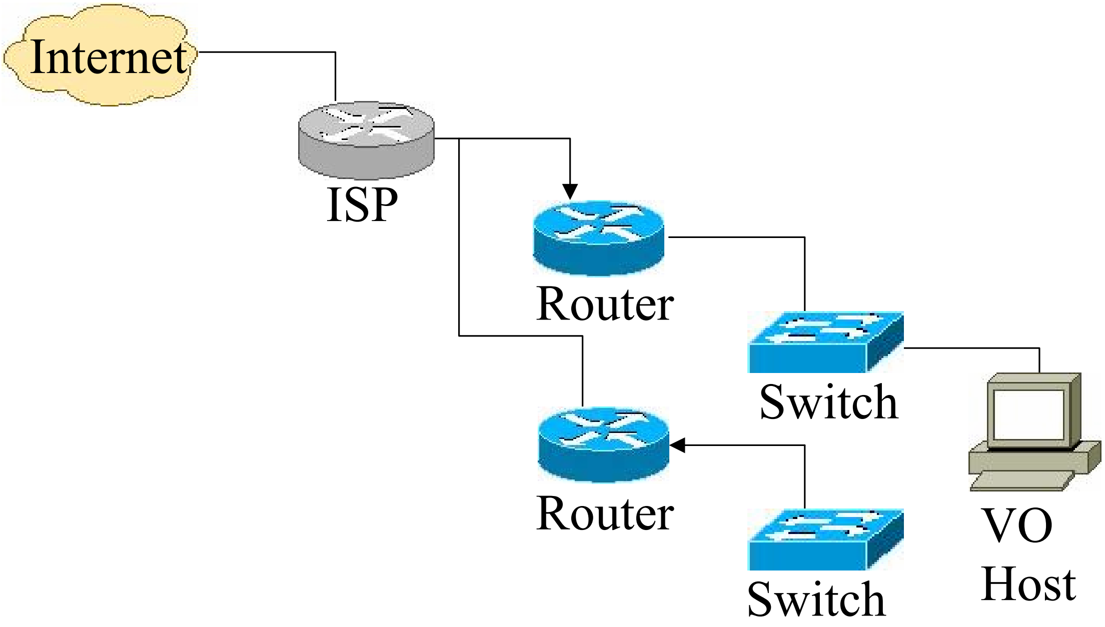


Figure 32 ADVERTISE Event Forward Path Routing Example

TASK Event

The purpose of the TASK event is to discover a resource available on the network based on a score devised by the VO host. It uses the DISCOVERY routing technique populated with the same *event_id* as the ADVERTISE event, traveling from a VO host to a resource provider, and uses a *next_address* populated in its payload. As the event travels from router to router, the *next_address* field is populated from the RESOURCE tables. When a resource is found in the table based on the *score*, the resource entry is removed from the RESOURCE table. If the *score* is not satisfied, then a TASK UNSATISFIED event is sent back to the VO host indicating that the task request was unfulfilled. Otherwise, the TASK event will eventually end up at a resource provider. As shown in Figure 33, the VO host sends a TASK event which finds its way over to the resource provider aggregated in the switch on the bottom of the diagram.

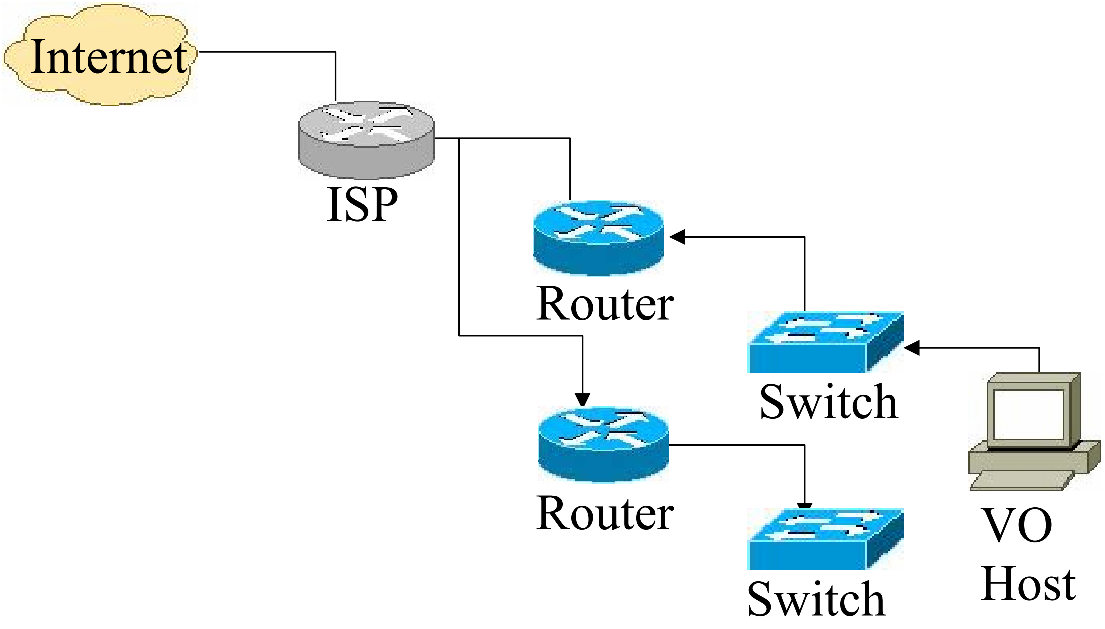


Figure 33 TASK Event Discovery Routing Example

TASK COMPLETE Event

The TASK COMPLETE event signals that a resource provider has finished computing its tasks. It uses the STANDARD routing technique with the same *event_id* as the ADVERTISE event, traveling from the resource provider to the VO host. No data table entries are modified during the transmission though the possibility is available for future use. The event includes a Boolean indicator to indicate if the event was *complete*. When the resource provider has finished its task, it sends a TASK COMPLETE event back to the VO host to indicate that results are ready to be transferred (Figure 34).

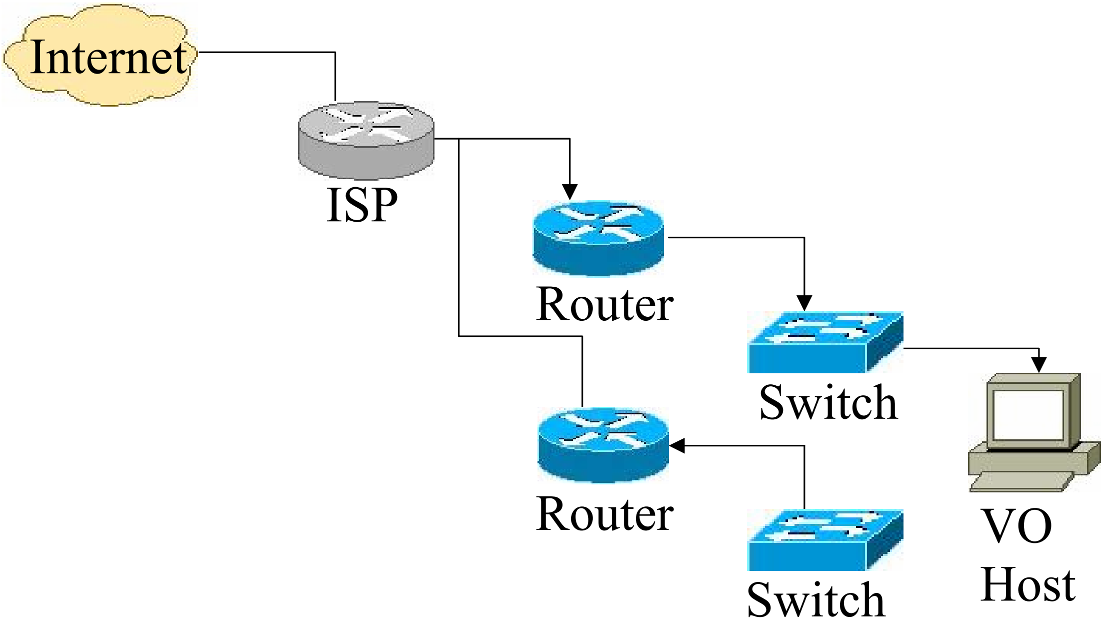


Figure 34 TASK COMPLETE Event Standard Routing Example

TASK UNSATISFIED Event

The TASK UNSATISFIED event is sent from a router or resource provider to a VO host if a resource cannot be found with the score requested. It uses the STANDARD routing technique since the score entries were erased from the RESOURCE tables along the TASK event's path. The event is populated with the same *event_id* as the ADVERTISE event. Also, as the event hops between routers, no data tables are modified. When the TASK UNSATISFIED event arrives at a VO host, it can decide to resend the TASK event at a later time. As shown in Figure 35, the top-most router cannot find a score that matches the VO host request. A TASK UNSATISFIED message is sent back to the VO host to indicate that the TASK event did not find a resource provider.

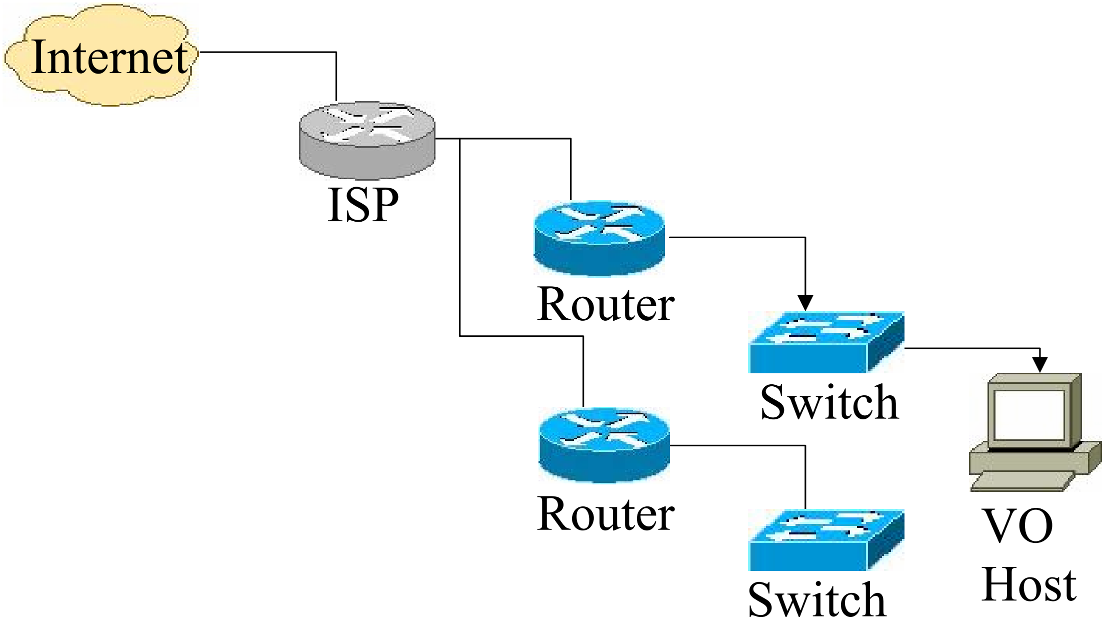


Figure 35 TASK UNSATISFIED Event Reverse Path Routing Example

CONFIRM DELIVERY Event

The CONFIRM DELIVERY event signals that a VO has received completed task results from the resource provider. It uses the REVERSE PATH routing technique from the TASK COMPLETE event with the same *event_id* as the ADVETISE event, traveling from the VO host to the resource provider. No data table entries are modified during the transmission though the possibility is available for future use. As shown in Figure 36, the CONFIRM DELIVERY message is sent from the VO host to the resource provider when the results have been successfully uploaded to the VO.

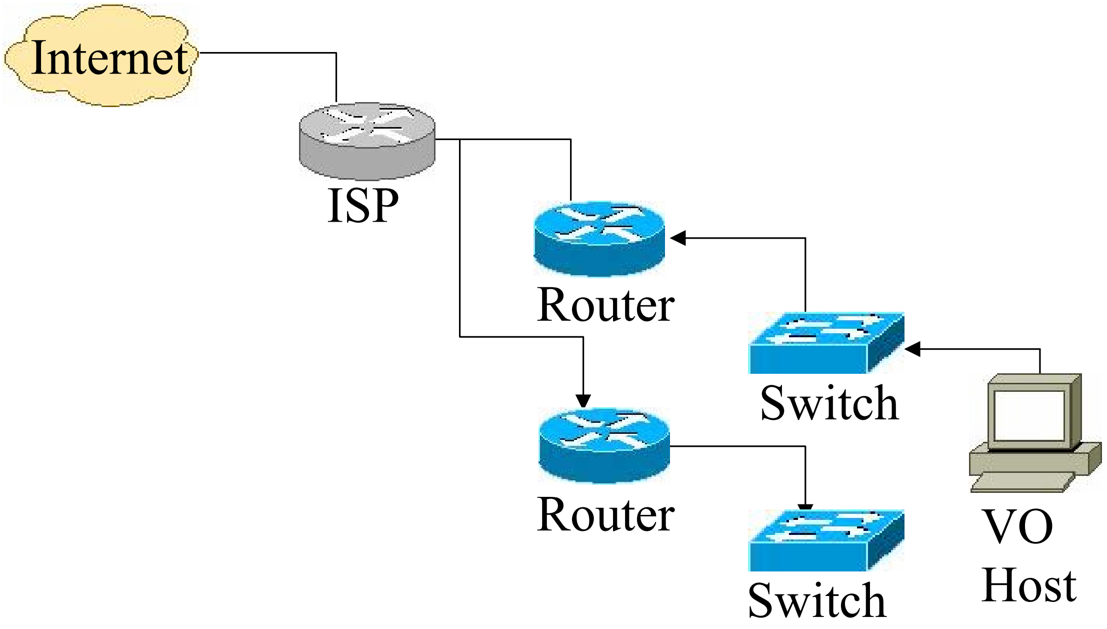


Figure 36 CONFIRM DELIVERY Event Reverse Path Routing Example

CONFIRM TRANSACTION Event

The CONFIRM TRANSACTION event signals that a VO acknowledged receiving completed task results from the resource provider. It uses the FORWARD PATH routing technique from the CONFIRM DELIVERY event with the same *event_id* as the ADVETISE event, traveling from the resource provider to the VO host. No data table entries are modified during the transmission though the possibility is available for future use. As shown in Figure 37, the CONFIRM TRANSACTION event is sent from the resource provider to the VO host indicating that it is aware that the VO has received the task results and does not need to attempt to send them again.

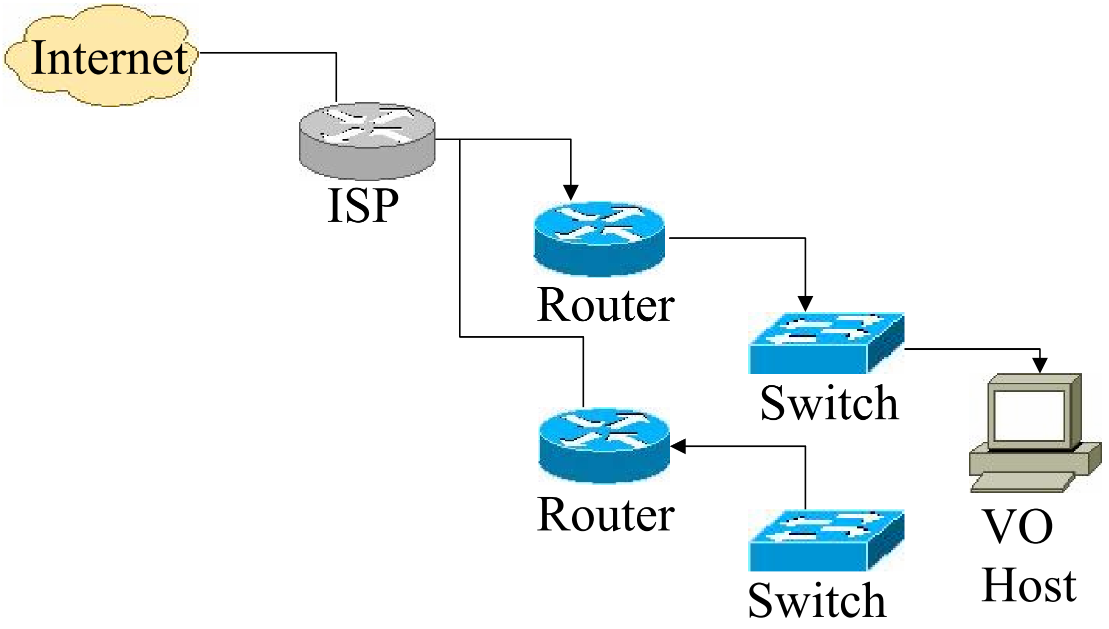


Figure 37 CNFIRM TRANSACTION Event Forward Path Routing Example

GOODBYE Event

The GOODBYE event signals that a resource provider wishes to leave the grid network. It uses the FORWARD PATH routing technique and the same *event_id* as the SIGNUP event, traveling from the resource provider to the VO host. The GOODBYE event removes the resource provider's entries in the RESOURCE tables of the routers along the path. The GOODBYE event is sent from the resource provider to the VO (as shown in Figure 38).

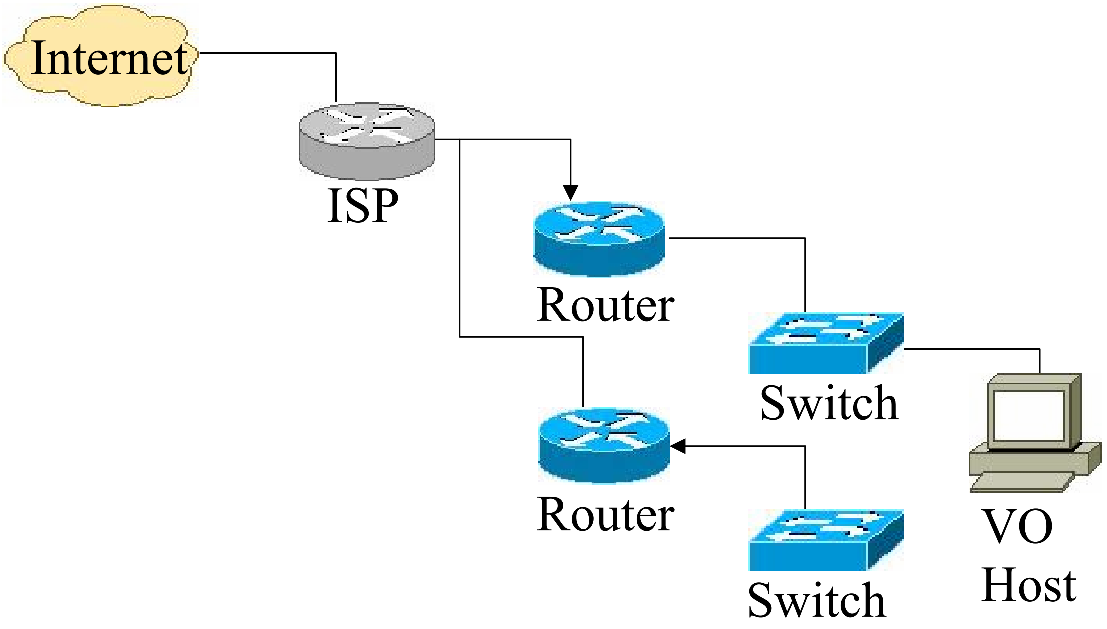


Figure 38 GOODBYE Event Forward Path Routing Example

UNSUBSCRIBE Event

The UNSUBSCRIBE event signals that a resource provider wishes to permanently leave the grid network. It uses the FORWARD PATH routing technique and the same *event_id* as the SIGNUP event, traveling from the resource provider to the VO host. The UNSUBSCRIBE event removes the resource provider's entries in the SIGNUP tables of the routers along the path. The event also includes a *permanent* Boolean to indicate if the un-subscription is permanent or temporary. A temporary un-subscription will not remove resource provider information from the VO host, whereas the permanent un-subscription will. The UNSUBSCRIBE event is sent from the resource provider to the VO host as shown in Figure 39.

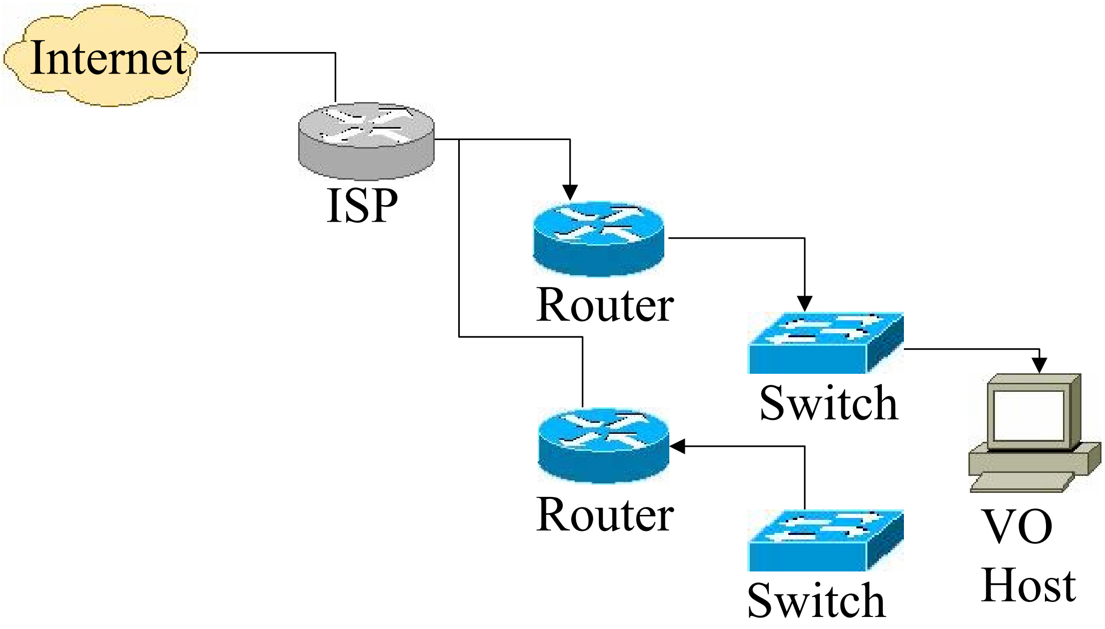


Figure 39 UNSUBSCRIBE Event Forward Path Routing Example

Resource Providers' Responsibilities

Looking at Figure 29, the resource providers are involved with the sending and processing of several interactions. When sending a SIGNUP event, the resource provider must track what VO it sent the event to until it unsubscribes from the network. When sending the ADVERTISE event, it can only send the event if the ACCEPT event was received. Also, the resource provider cannot send another ADVERTISE event until a TASK message arrives at the router or if the resource provider sends a GOODBYE message. The TASK COMPLETE event depends on the TASK event reception, the CONFIRM TRANSACTION event depends on the CONFIRM DELIVERY event. The GOODBYE message can be sent when an ADVERTISE event was sent and the resource provider was not tasked via a TASK message. The UNSUBSCRIBE event can be sent if the router was accepted with the ACCEPT event.

Router Responsibilities and Usage of Data Tables

There are three data tables used by the grid protocol in the routers along an event path: the SIGNUP, RESOURCE, and BLACKLIST tables. Each table serves a different purpose and follows a set of guidelines when data should be added or removed from the tables. The tables are presented in this section. Note that the tables have optimized implementations that are specified in the simulation section.

SIGNUP Table Usage

The SIGNUP table's purpose is to record when a resource is signed up and allowed to participate on the grid network. In addition to maintaining the resource and VO host IP addresses, the signup table keeps track of a timeout value, the resource score, and whether or not the resource is accepted on the network. Table 17 is composed of Table 16, which is composed of Table 15; this allows the implementation to save memory when storing the data structures. Each router has a SIGNUP table.

Table 15

SIGNUP TABLE ENTRY Data Structure

SIGNUP TABLE ENTRY

Timeout	Accepted	Score

Table 16

SIGNUP TABLE HELPER Data Structure

SIGNUP TABLE HELPER

VO IP Address	SIGNUP TABLE ENTRY

Table 17

SIGNUP TABLE Data Structure

SIGNUP TABLE	
Resource IP Address	SIGNUP TABLE HELPER

The initial entry into the SIGNUP table occurs when the SIGNUP event arrives. If the resource provider is not blacklisted, a 10-byte SIGNUP TABLE ENTRY is created which contains the timeout value, accepted Boolean value, and the resource provider's score. Initially, the accepted Boolean is set to false, the timeout is set to 120 seconds from the current time, and the other values are populated from the SIGNUP event. The SIGNUP TABLE HELPER allows fast lookup of SIGNUP TABLE ENTRIES and helps to save memory.

When the ACCEPT event arrives at the router, the SIGNUP TABLE ENTRY is retrieved and the accepted value is set to TRUE if the ACCEPT event's *accepted* value is TRUE and the timeout value is set to 24 hours from the current time. If the ACCEPT event's *accepted* value is FALSE, the event is blacklisted and the entry is removed from the SIGNUP table. The ACCEPT event is always sent to the next hop in the *path* because each router must know the state of acceptance.

Other events may access the SIGNUP table, but the only other event which modifies the SIGNUP table is the UNSUBSCRIBE event. When the UNSUBSCRIBE event arrives at a router, the SIGNUP TABLE ENTRY is removed from the router's SIGNUP table. From the router's perspective, a resource can signup if a SIGNUP TABLE ENTRY does not exist, if the 24 hour wait period expired, or even if the resource retired from the VO's network. The router does not track retired resource providers.

RESOURCE Table Usage

The RESOURCE table is used to track which resource providers are available. A resource becomes available when it advertises its availability via the ADVERTISE event. When a router receives the ADVERTISE event, it populates the RESOURCE TABLE ENTRY with data from the ADVERTISE event and the SIGNUP table. The RESOURCE TABLE ENTRY is removed when a TASK reserves a resource or when a GOODBYE event is received by a router.

Table 18

RESOURCE TABLE ENTRY Data Structure

RESOURCE TABLE ENTRY	
Next Hop IP	Number of Devices

Table 19

RESOURCE TABLE HELPER Data Structure

RESOURCE TABLE HELPER	
Score	RESOURCE TABLE ENTRY

Table 20

RESOURCE TABLE Data Structure

RESOURCE TABLE	
VO Host Hash Key	RESOURCE TABLE HELPER

Table 21

VO Host Hash Key Data Structure

VO Host Hash Key Data Structure (16 bits)		
VO Host IP Last 4 Bits of Field 3 IP Address	VO Host IP Field 4 (8 bits)	VO Product Id (4 bits)

Just as memory is saved for the signup tables, each router has a resource table like Table 20 composed of Table 19, which is composed of Table 18. The RESOURCE TABLE uses a VO Host Hash Key (shown in Table 21) to lookup RESOURCE TABLE HELPER tables; the helper table has a score key to enable fast lookup to access the RESOURCE TABLE ENTRY. The RESOURCE TABLE HELPER has a key to lookup the entry by the one-byte score. The RESOURCE TABLE ENTRY holds the one-byte number of devices available for the particular resource and the four-byte IP of the previous hop address. If the number of devices decrements to zero, then the entry is erased. Likewise, if the entry is removed then the helper entry is erased for that score value.

BLACKLIST Table Usage

The BLACKLIST table (Table 22) is used to prevent unauthorized access or data transmissions between members on grid network. When a resource provider sends a SIGNUP event, a BLACKLIST table entry is created with a one-byte count of 1. If the VO sends an ACCEPT event with an *accepted* value of TRUE, then the BLACKLIST entry is erased. If the *accepted* value is FALSE, then the BLACKLIST entry is set to three indicating that the resource is considered blacklisted and the resource provider is not allowed to participate in the grid network.

Table 22

BLACKLIST TABLE Data Structure

BLACKLIST TABLE	
Resource Provider IP	Count

VO Host Responsibilities

The VO host primarily serves as a resource provider and consumer authenticator and authorizer. When the VO host receives the SIGNUP message, it sends an ACCEPT message to the resource provider and optionally to the consumer (not shown in Figure 29). TASK messages sent from the consumer are sent to the VO Host (since the consumer must be authenticated and authorized), and then to the grid network routers. The VO host also receives CONFIRM TRANSACTION events for tracking purposes and security reasons (like for allowing SIGNUP events to be accepted). When it receives an UNSUBSCRIBE event, it allows resource providers to retire from VO membership when the retiring flag is set.

Scoring

Each resource provider participating in a grid network has attributes that define the resource: number of CPUs, CPU speed, amount of RAM, available hard drive space, and the speed of their bandwidth connection are shown in Table 23. The grid protocol scores these devices based on their attributes using a one-byte unsigned character.

Table 23

Score Data Structure

Score Data Structure (8 bits)

CPU	Memory	Hard Drive	Bandwidth

Score data structure has four two-bit fields representing CPU type and count, memory, hard drive, bandwidth.

The scores represent ranges of resource attributes from 0-3. The enumerations of each range are specified for a VO. For example, VO #1 may designate a CPU class of 0 to represent 1 CPU machine up to 2 GHz, a class of 1 to represent 1 CPU machine over 2 Ghz, a class of 2 to represent a 2 CPU machine under 2 GHz, and a class of 3 to represent a 2 CPU machine over 2 GHz. Likewise, a different VO # 2 may designate a workstation class CPU an enumeration of 0, a server class CPU an enumeration of 1, a multiprocessor device with an enumeration of 2, and a cluster computer or higher with an enumeration of 3.

A sample scoring table is provided in Table 24. A score of 205, for example, can be represented as 1000 1101 (binary) or 0x8D (hexadecimal) which decomposes into a CPU score of 2, a memory score of 0, a hard drive score of 3, and a bandwidth score of 1. Using Table 24, this translates into a 4 CPU machine with 512 megs of memory or less available, a hard drive capacity over 120 gigs available, and a bandwidth connection speed of 128 K.

Table 24

Example Scoring Table

VO #3 Scoring Table Example

	Score of 0	Score of 1	Score of 2	Score of 3
CPU	1 CPU	2 CPUs	4 CPUs	8 CPUs
Memory	<= 512 Meg	.5 – 1 Gig	1 – 2 Gig	> 2 Gig
Hard Drive	<= 10 Gig	10-80 Gig	80-120 Gig	> 120 Gig
Bandwidth	<= 56 K	128 K	256 K	> 256 K

One other factor to consider for score is how the TASK events generate scores to seek. The five deployment schemes and their score-seeking techniques are defined in Table 25. The “don’t cares” indicate that the particular VO does not care about the particular resource attribute when creating events.

When a TASK event arrives at a router, the score is looked up in the RESOURCE TABLE. First the router checks to see if the score is matched perfectly. If it is not, it finds the next highest score. If it cannot find a score, then a TASK UNSATISFIED event is sent back to the VO host.

Table 25

Deployment Environment Don’t Cares

Deployment Environment	Don’t Cares
Science Portal	Bandwidth
Distributed Computing	None
Computer-in-the-Loop Instrumentation	Hard Drive
Large-Scale Data Analysis	CPU, Memory, Bandwidth
Collaborative Work	Memory

```
If score matches perfectly
        Decrement the score in the RESOURCE TABLE
        Forward message to appropriate router
Else if a higher score is available
        Find a higher score that is as close to the desired score
        Change the desired score in the TASK event
        Decrement the score in the RESOURCE TABLE
        Forward the message to the appropriate router
Else
        Send TASK UNSATISFIED event to VO host
        Drop the TASK event
```

Figure 40 Router Search Algorithm for Finding a Score in the RESOURCE TABLE

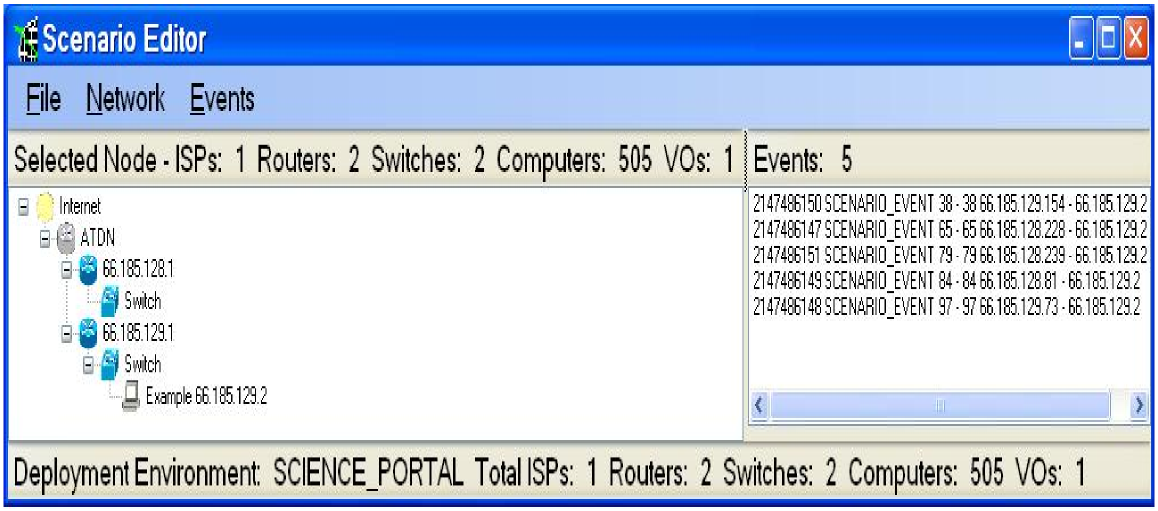
Grid Topology Scenarios

Since grid deployment environments [56], resource agreements, VMMs, and VOs have been discussed in the Grid Computing Background section, consider the network topology of the deployment environments. Each of these models will use the decentralized concept of the resource discovery proposed. The differences lie in the application of the model’s scenario and the way that the routers will use the scoring mechanism to find resource providers.

The network topology can contain the following network devices: the root network identifier node (usually the Internet), ISPs, routers, switches, and VO hosts. The switches can aggregate up to 253 resource providers. The following rules apply when building a network:

- The root network node can only have ISP children.
- The ISP nodes can only have router children.
- The router nodes can have ISPs, routers, switches, or VO hosts.
- The switch nodes can have routers, resource providers, or VO hosts.
- VO hosts cannot have children nodes.

Figure 41 shows an example of a network topology. The root node, labeled "Internet," has one ISP labeled "ATDN." ATDN has two routers with the IP addresses 66.185.128.1 and 66.185.129.1. The switch under 66.185.128.1 has 253 available devices with an IP range from 66.185.128.2-66.185.128.254 (not shown). The switch under 66.185.129.1 has a VO host names "Example" with an IP of 66.185.129.2 and 252 resource providers with an IP range between 66.185.129.3-66.185.129.254 (not shown). A minimum network topology is shown in Figure 42 which shows some resource providers.



Selected Node - ISPs: 1 Routers: 2 Switches: 2 Computers: 505 VOs: 1

```
Internet
  ATDN
    66.185.128.1
      Switch
    66.185.129.1
      Switch
        Example 66.185.129.2
```

Events: 5

```
2147486150 SCENARIO_EVENT 38 - 38 66.185.129.154 - 66.185.129.2
2147486147 SCENARIO_EVENT 65 - 65 66.185.128.228 - 66.185.129.2
2147486151 SCENARIO_EVENT 79 - 79 66.185.128.239 - 66.185.129.2
2147486149 SCENARIO_EVENT 84 - 84 66.185.128.81 - 66.185.129.2
2147486148 SCENARIO_EVENT 97 - 97 66.185.129.73 - 66.185.129.2
```

Deployment Environment: SCIENCE_PORTAL Total ISPs: 1 Routers: 2 Switches: 2 Computers: 505 VOs: 1

Figure 41 Scenario Editor Network Topology Example

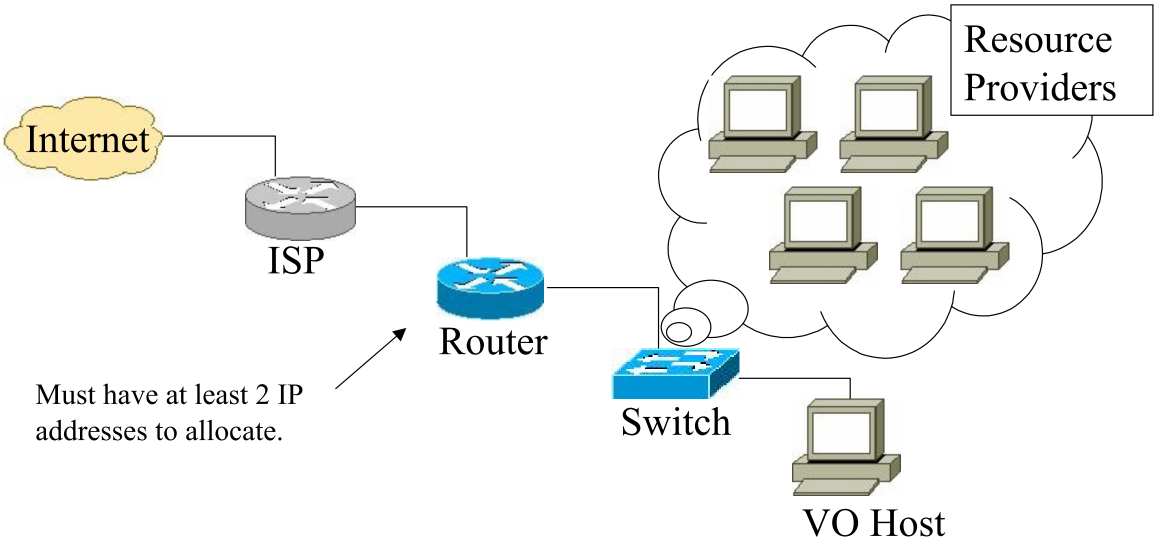


Figure 42 Minimum Network Layout

Science Portals

In the Science Portal deployment environment, a scientist would log onto a VO host computer via a web-based thin client connection. The scientist sends a work order to the VO web portal host computer, the VO host computer divides the tasks into manageable pieces, and then the host computer sends the appropriate TASK events throughout the network to find resource providers. With the science portal scenario type, the scientists do not care about bandwidth, so the bandwidth value in the score is set to a "don't care" value of zero. As shown in Figure 43, scientists would send work orders to the VO web portal that would send the work out over the grid network. If the grid network cloud were expanded out, it would look similar to the network in Figure 41 where the example VO would be the VO web portal.

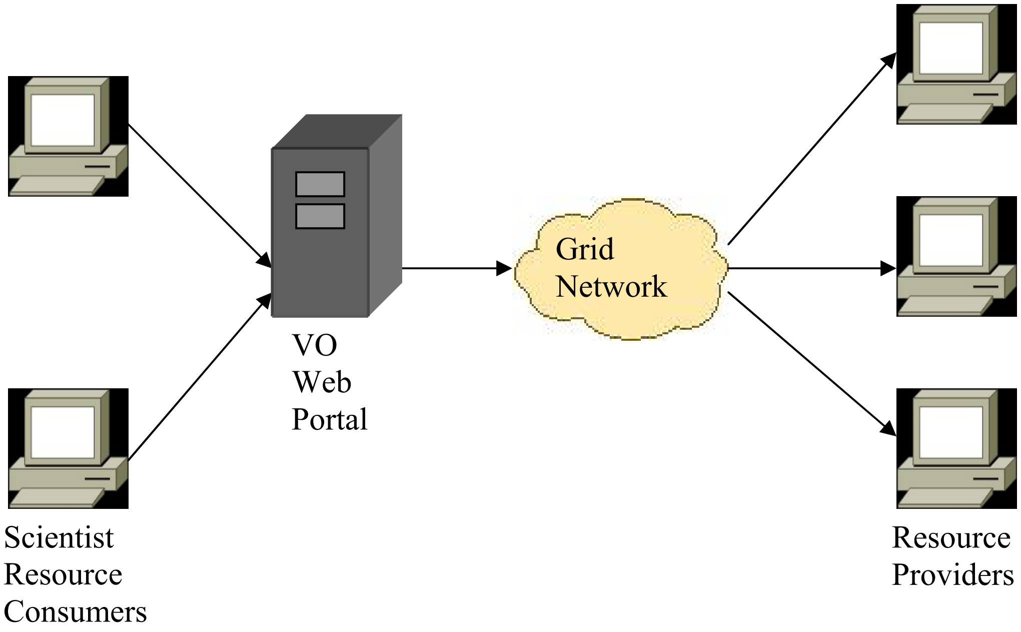


Figure 43 Sending TASK Events in the Science Portal Scenario.

Distributed Computing

In the Distributed Computing deployment environment, the scenario allows individual PCs to be combined via parallelization to provide substantial computational resources. The VO host may have a very long list of TASK events to process. The VO host can send these events when it receives a SIGNUP or CONFIRM TRANSACTION event from a resource provider or whenever it chooses to. With the distributed computing scenario type, all attribute values of the score are considered (none are set to "don't care" values). As shown in Figure 44, work orders are sent from the VO host to work out over the grid network.

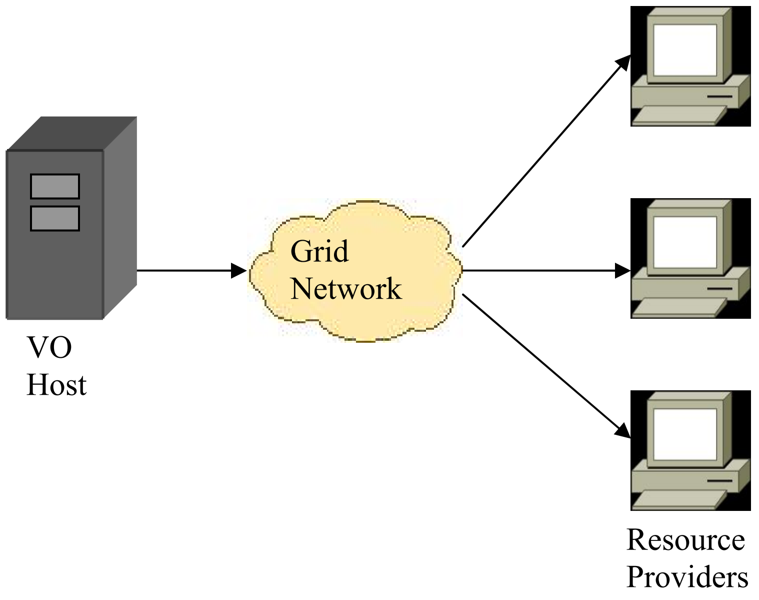


Figure 44 Sending TASK Events in the Distributed Computing Scenario

Large-Scale Data Analysis

In the Large-Scale Data Analysis deployment environment, computational grids provide the capability of acting as a large storage facility in addition to providing computational powerhouses. The VO host, for example, could try to periodically send out TASK events requesting a particular sized hard drive. With the large-scale data analysis scenario type, the hard drive space matters most and the other fields are marked as "don't cares." As shown in Figure 45, researchers would send a request to store and analyze a large amount of data to the VO host. The VO host would divide the request up into multiple TASK events that would be sent out over the grid network.

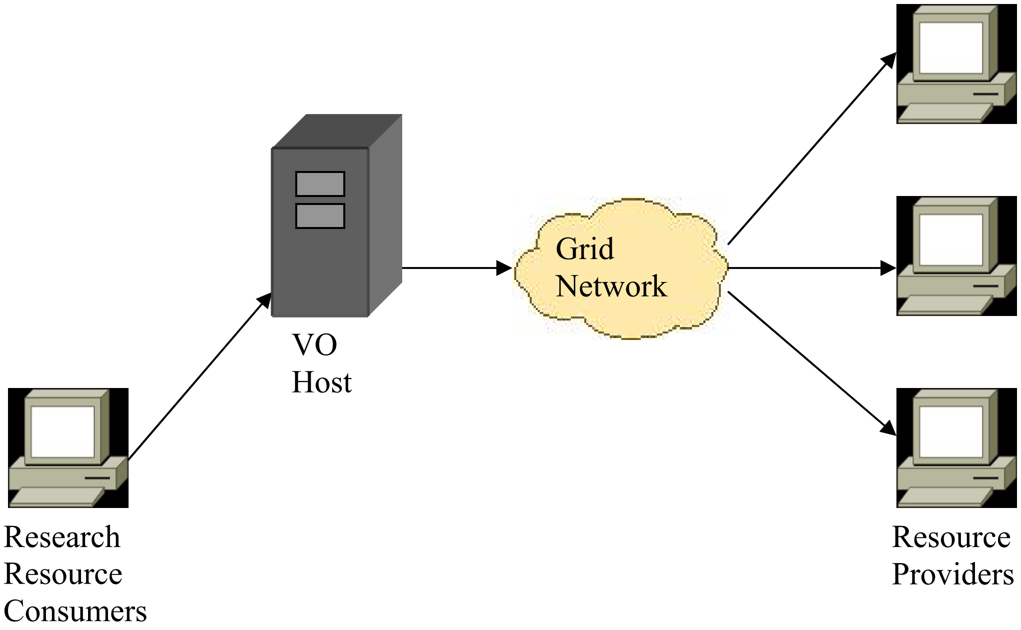


Figure 45 Sending TASK Events in the Large-Scale Data Analysis Scenario

Computer in-the-loop Instrumentation

In the Computer-in-the-loop Instrumentation deployment environment, scientific instruments are used to collect streams of data which are archived and processed later to detect things of scientific value. The VO host, for example, could try to periodically send out TASK events requesting a particular CPU, bandwidth speed, and block of memory to receive streaming data. With the computer in-the-loop instrumentation scenario type, the hard drive space available is marked as a "don't care" assuming the VO requires a large enough amount of free space when the resource subscribes to the VO. As shown in Figure 46, scientific instruments constantly stream data to a VO host. The VO host issues TASK events over the grid network.

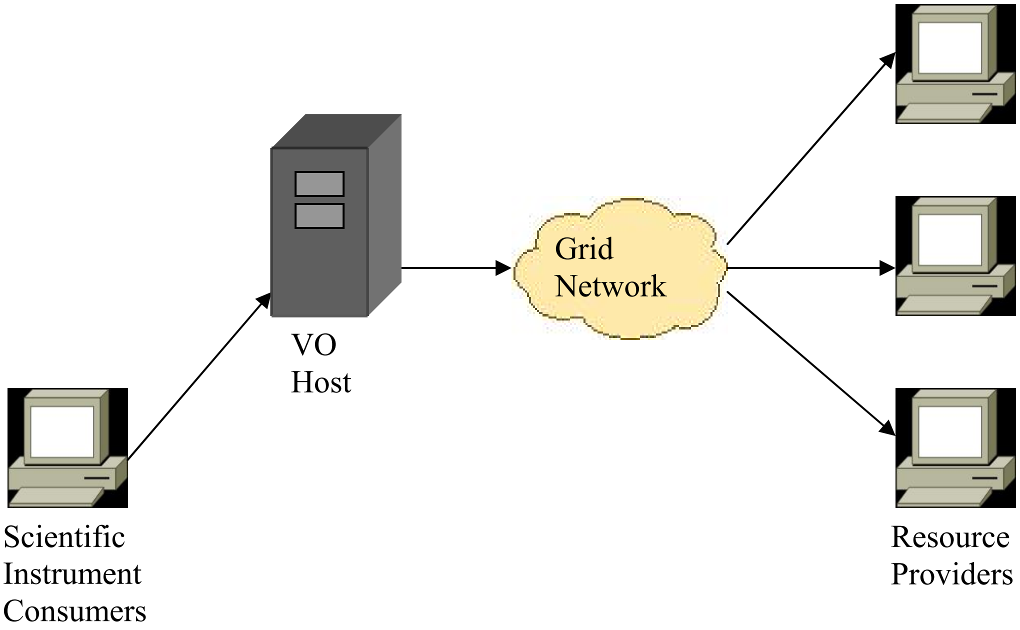


Figure 46 Sending TASK Events in the Computer in-the-Loop Scenario

Collaborative Work

In the Collaborative Work deployment environment, scientists may want to collaborate to discuss results and offer suggestions. The VO host, for example, could try to periodically send out TASK events requesting a particular CPU, bandwidth speed, and hard drive space to accommodate collaboration. With the collaborative work scenario type, the memory available is marked as a “don’t care.” As shown Figure 47, Scientific instruments constantly stream data to a VO host. The VO host issues TASK events over the grid network.

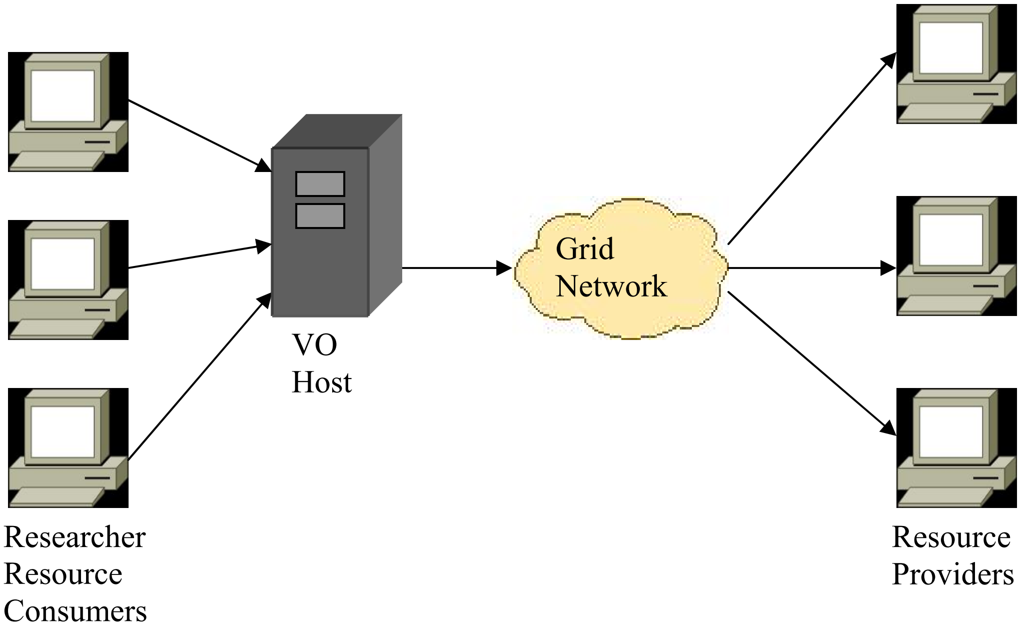


Figure 47 Sending TASK Events in the Collaborative Work Scenario

Grid Security

"Grid systems and applications may require any or all of the standard security functions, including authentication, access control, integrity, privacy, and non-repudiation," [72]. Regarding security, the VOs act as a trust domain (as defined in [72]). The VO host can handle grid resource provider and consumer authentication and protection of credentials. VO hosts can also act as proxies to other VO hosts; they can use criteria to judge another entity based on its VO sponsorship. Access control is granted through authentication and use of the VOs API on the grid resources: the API will only have functionality programmed into it which allows access to devices specified by the security policy of the proxy or the VO which provides the API. Integrity of data can be monitored by the grid routing protocol. For all successful transmissions, an integrity counter can be incremented on the routers and at the proxy to indicate that a successful transaction has occurred.

Privacy can be controlled somewhat by the use of encryption, but as [72] points out, not all countries agree on similar types of encryption (assuming there is a world wide distributed grid network). Also, if remnants of computer usage (i.e. temp files or source code from the trusted consumer model) are not deleted, then privacy can be compromised.

The new security risk that these models introduce has to do with data tables being stored on routers. If someone could hack into a router, this person could alter credentials or BLACKLIST tables and redirect more traffic to his or her own network to steal information or to make more money. One way to discourage this behavior is for the proxy to watch for a fair distribution of the grid resources. Based on the resources available, if a resource appears to be a hog by not allowing other grid resources to get their shares of the workload, the VO host temporarily suspends authentication for that grid resource provider thus forcing work orders to go to different accounts.

Another security risk for the router integrity is for the resource providers to send repetitive SIGNUP or ADVERTISE events to inflate the amount of available resources. Routers track the frequency of SIGNUP and ADVERTISE events; if too many events arrive in too short of a time or without any satisfactions over a long period of time, the router can disable any TASK events from going to that resource provider.

One other security risk for the protocol is that any component can be an imposter component: that is a component which looks and acts like a trusted component but is really designed for malicious purposes. Proxies can be made to steal names, passwords, or other security credentials. Resource providers can steal data or produce bogus results. Grid consumers can be falsely identified so a different customer is billed for activity the customer did not use. These situations can also be monitored reactively through the use of integrity counters as described above.

HLA Simulation Protocol and the Simulation Engine

Rather than building a simulation architecture from the ground up, after reviewing three popular simulation architectures (ALSP, DIS, and HLA), the decision is made to design a simulation engine that performs basic HLA operations. There are two reasons for this decision: the first is because ALSP is a legacy product. It was designed by many of the same people and the same organization (MITRE) that designed HLA [22], so the shortcomings of ALSP were addressed in HLA [2][8][10]. The second reason is even though DIS is considered legacy, it is still used in the industry today [19][20][21] and it has been adapted to work in concert with HLA [1]. The architectural approach to achieving this is to create a software simulation layer in-between the simulation code and the RTI interface as shown in Figure 48. Also, this simulator can be built from core software from the k-array n-cube and CLL simulators though it will introduce new code.

Another requirement for the simulation engine is for it to be able to operate without the RTI as shown in Figure 49. Thus, time management, object and event management, scenario parsing, and other features provided by the RTI will be provided by the simulation engine. This requirement is imposed because not all simulations may require distributed simulation or perhaps the simulation programmer desires a simplistic testing environment.

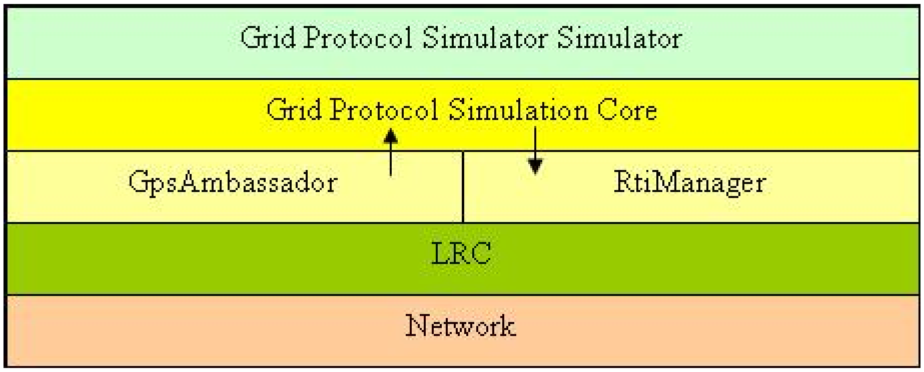

Grid Protocol Simulator Simulator
Grid Protocol Simulation Core
GpsAmbassador
RtiManager
LRC
Network

Figure 48 The Simulation Core is placed Between the Simulation Software Application and the RTI

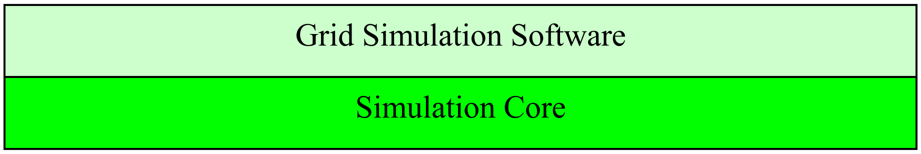


Figure 49 The Simulation Engine Supports a Mode Where RTI Services are not used

Simulation Core

The simulation core software component is responsible for keeping the simulation running by managing simulation time, sending and receiving events, understanding the FOM, and managing the network infrastructure. In the software, these classes are packaged in the GPSC namespace (Grid Protocol Simulation Core). The core software supports two modes of operation: with and without the RTI. The software components are similar when running in either mode, but the RTI mode adds a few extra classes. As shown in Figure 48, the GpsAmbassador receives messages from the RTI and the RtiManager class sends messages over the network. The GpsAmbassador class inherits from the RTI's FederateAmbassador class as prescribed in the RTI spec [10].

The other core classes include: EventManager, Event, NetworkTree, NetworkNodeBaseClass, StateMachine, and TimeManager. These classes are described in the upcoming sections.

EventManager Class

The EventManager is responsible for scheduling and delivering simulation events, maintaining an event queue, and remembering event statistics. The events are stored in an ArrayList structure provided by the CLR framework. The structure is not sorted, but the list is manually sorted each time advance. When events are sent, they are added to the end of the event queue and sorted to the proper position when time is ready to advance forward. Events can only be sent in the future (current time plus one or more), not at the present or in the past.

The EventManager also tracks event statistics in a data structure. Each event that is sent is counted. The event is only counted once because the event id is stored as a unique key. When the simulation ends, the EventManager is asked to give statistics for the all events passed through the simulation.

Event Class

The Event class is the base class for any event propagated or represented in the simulation. Each event has the capability to track its path through the network, starting and ending IP addresses, event starting time, time of next delivery, routing method, and the event type. The path can be populated or used in forward or reverse based on the routing method. Some events may not use parameters; like the TASK message that does not know its destination because it has to be discovered.

NetworkTree Class

The NetworkTree class is the container that holds all of the network devices. The network tree is a tree structure with functions to assist in the routing of messages. In some cases, particularly for routers, the network device will have to route the events. The device is given the first chance to route an event. If the device routes the event, then the network tree will not route the message; otherwise it will.

NetworkNodeBaseClass Class

The NetworkNodeBaseClass is the base class for all network devices contained in the network tree (i.e. the routers, switches, and VO hosts). Any device inheriting from the NetworkNodeBaseClass will have a name, a device type enumeration, and a reference to its parent node in the tree.

StateMachine Class

The StateMachine class is responsible for maintaining the current state of the simulation. The simulation states are: STOPPED, INITIALIZING, RUNNING, SHUTTING_DOWN, and PAUSED. When the simulation is started, it transitions from the STOPPED state to INITIALIZING and eventually to RUNNING. When the simulation is complete, the simulation enters SHUTTING_DOWN state followed by the STOPPED state. When the simulation is in RUNNING state, the simulation can transition to PAUSED and then back to RUNNING.

During the STOPPED and PAUSED states, no simulation activity is occurring. The INITIALIZING state signals the simulation to read in the scenario and populate the

NetworkTree and EventManager event queue. The RUNNING state starts the simulation clock and event transactions. When the simulation is in the SHUTTING_DOWN state, the event statistics are calculated and the Excel spreadsheets are generated.

TimeManager Class

The TimeManager class is the container for the simulation's current time while running. The starting, advancing, and stopping of the clock is done from this class by interfacing with the GpsGui class' background worker thread that runs the simulation. The TimeManager also provides mutex services for pausing the simulation and synchronizing with RTI synchronization points.

Simulation Engine Common Library

In addition to the classes mentioned above, the simulation core includes an additional namespace called SECL (Simulation Engine Common Library). The distinction between the classes in the common library and the core is that common library classes can only call standard C++, C++/CLR, and SECL classes. Thus, these classes are designed to be the most reusable parts of the simulation engine. Examples include math classes (such as Random), error display (such as GuiUtilities), and logging (such as Logger).

Simulation Architecture

When using the RTI, this means that the simulated network event traffic can be distributed to different computers running the simulation. Distributing the workload means that the simulations run faster because each simulation event queue has to process fewer events and the network topology is smaller. The network is portioned out based on the tier 1 ISPs that fall under the root network node. Consider an example where a two-ISP network scenario is simulated on CPU 1. All of the simulation is done on this CPU as shown in Figure 50; the thought cloud shows the CPU is computing messages through two ISPs. The workload can be distributed over another CPU since there are two ISPs in this particular scenario. This is done by CPU 1 loading the first ISP and CPU 2 loading the second ISP as shown in the thought clouds in Figure 51. CPU 1 also will run the RTIExec program that is responsible for creating and managing the federation. The computers are connected over a LAN and events are exchanged as appropriate.

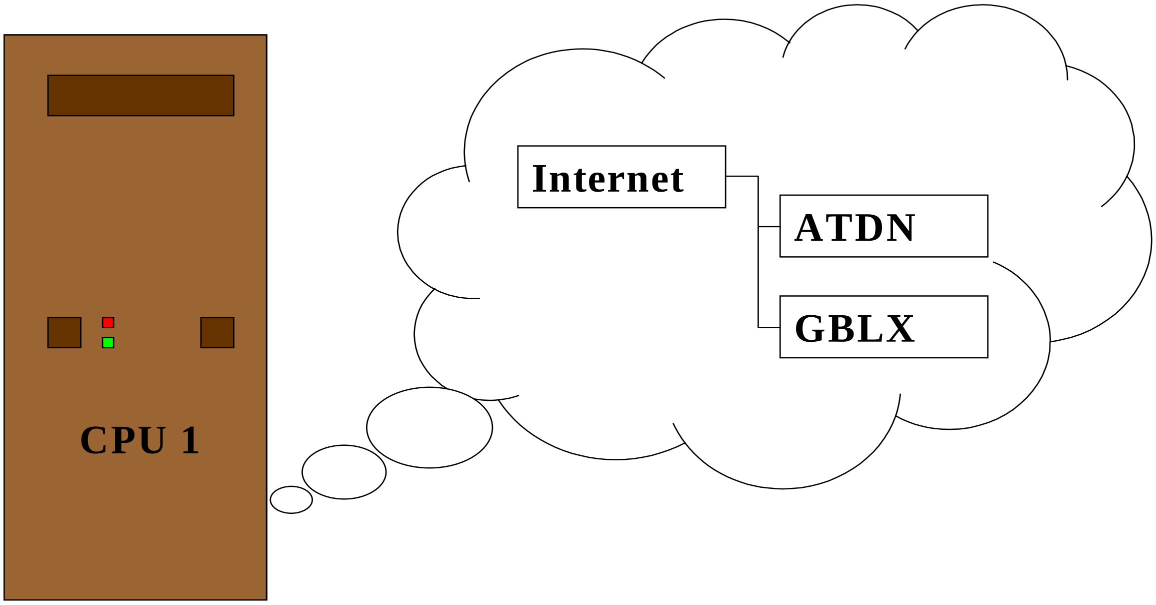


Figure 50 Simulation without the RTI

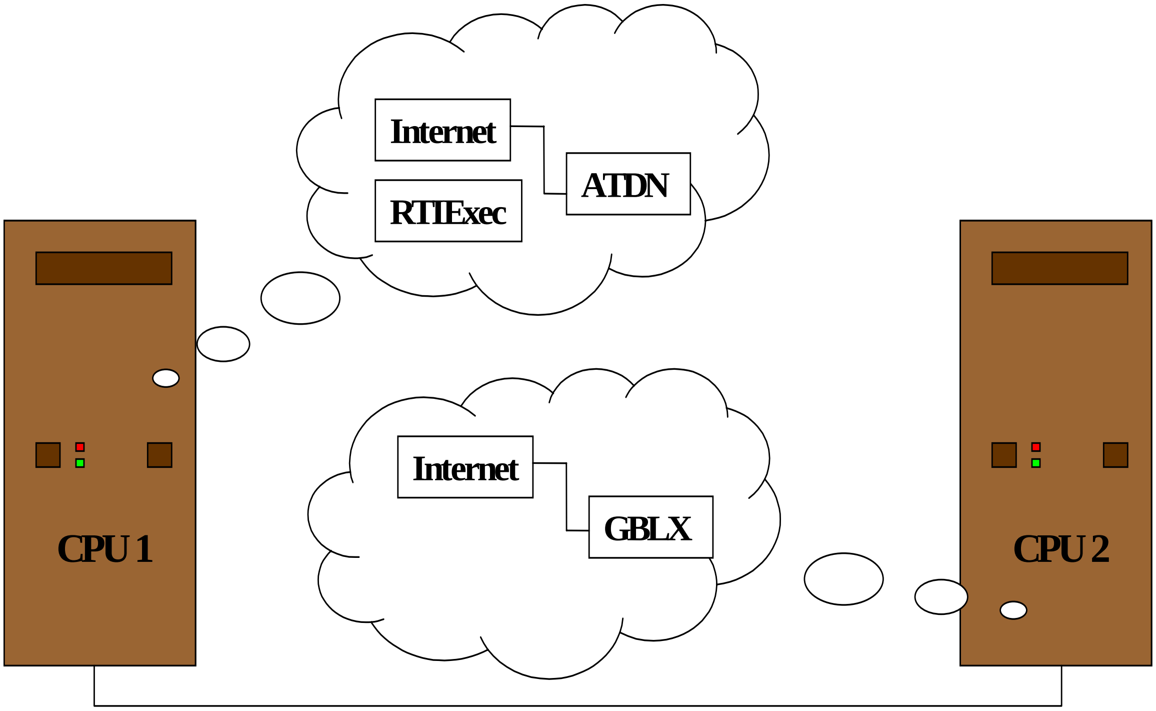


Figure 51 Simulation with the RTI

Scenarios contain network and event information. When the simulation not running the RTI loads the scenario, it loads the entire network and event queue and simulates the grid discovery protocol behavior. However, when using the RTI, CPU 1 only loads the first ISP and CPU 2 only loads the second ISP. Likewise, scenario events that pertain to the other CPU are dropped based on the IP address of the resource provider. For example, if a scenario event has a resource provider with an IP address of 92.168.123.123 and if that resource provider exists in the GBLX ISP, then CPU 1 will drop the message and CPU 2 will process the message.

The RTI connection is used when an event has to cross from one ISP to the other; thus the RTI acts as the Internet backbone between Tier 1 routers. For example, consider a scenario where resource IP address 92.168.123.123 resides in GBLX and VO Host IP address 93.168.123.123 exists in ATDN. When the first SIGNUP event has to travel from GBLX to ATDN, a corresponding SIGNUP event is created (based on the FOM) and the message is passed over the RTI.

Time regulating and constraining settings are disabled when using the RTI. This allows the simulator to control time rather than having the RTI control time. When events are sent over the network, they call the sendInteraction() function which does not take a time parameter. This does not timestamp messages the cross between the CPUs or federates; messages are placed in the Receive Order (RO) queue rather than the Timestamp Order (TSO) queue. Messages that arrive in the receive order queue may arrive out of order. Considering the architecture of the simulation, messages can be received out of order since each message is independent of the other.

The benefit of using the RTI is to save time simulating the scenarios, but there are two drawbacks. The first drawback is the usability factor where the user will have to take additional steps to run the simulation with the RTI. This includes starting the RTIExec process, the RTI license manager, and setting up the simulator to run with the RTI (enabling an RTI checkbox, setting the federate name, etc.) The added complexity leaves more room for human error. The second drawback is that the simulation results will reflect the results per each federate. So, when the scenario simulation is completed, the user must combine the results across the federates to see the big picture of the simulation. The simulation design allows this to happen because the events carry their statistics internally (i.e. the number of hops, start time, etc.)

CHAPTER FOUR: FINDINGS

K-Array N-Cube Evaluation and Results

Simulation Implementation and Techniques

Object-oriented software implementation strategies such as use of the STL, singleton classes and pure virtual functions have been employed to provide a flexible, extensible, and robust means to establish network hardware structures. These implementation strategies are vital implementation methodologies to the network benchmark model in order to obtain higher modularity and lower integration complexity. A systematic usage of these functionalities throughout the simulator design lead to a better model that improves system performance and supports future upgrades such as additional types of networks, protocols and/or flow control mechanisms. A brief review of the implementation techniques is provided in the next few sections.

The Singleton Class

The singleton classes [44], such as WormManager and Interconnect shown in Figure 52, guarantee that only one class instantiation is created. Figure 52 shows all the objects and functions (public and private) included in each of these singleton classes. The single instance is held as a static variable as a private member of the class. These singleton classes are not automatically initialized when the program loads. Instead, initialization occurs the first time that singleton class' create method is called by the client. The create method also allows the callers to access methods of that singleton class since it returns a pointer to the class. In a similar manner the Singleton class can release the object from memory by calling destroy. The Interconnect is a singleton class, that is only one interconnect is created per simulation. The WormManager creates a new interconnect at the start of each simulation and destroys it when done. The reason for this is that there might be different configurations which require construction of the object in different ways within the WormManager class.

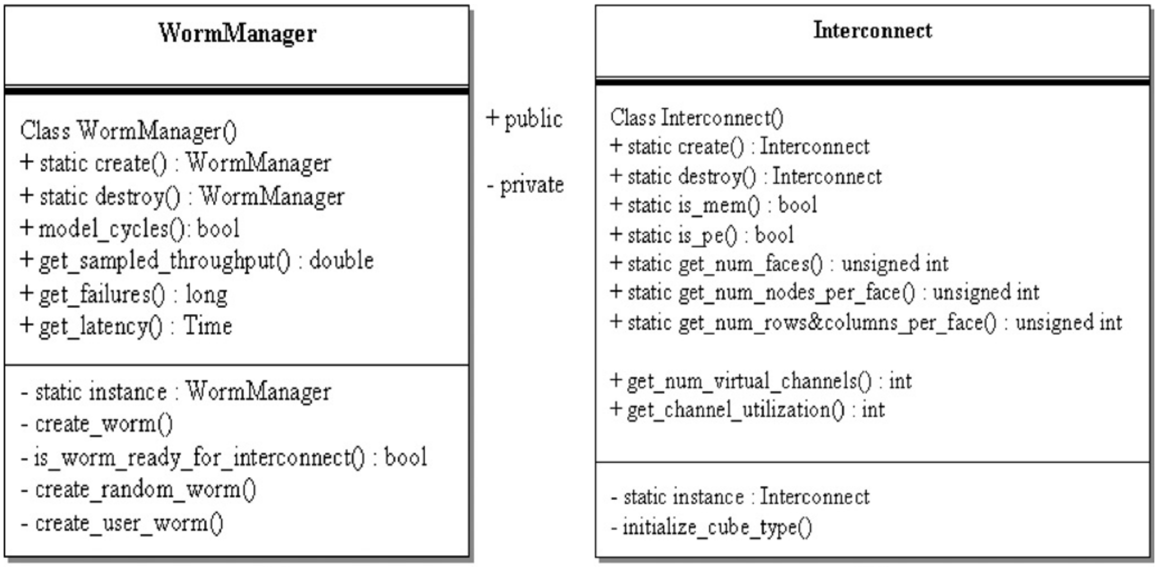

WormManager
Class WormManager()
+ static create() : WormManager
+ static destroy() : WormManager
+ model_cycles(): bool
+ get_sampled_throughput() : double
+ get_failures() : long
+ get_latency() : Time
- static instance : WormManager
- create_worm()
- is_worm_ready_for_interconnect() : bool
- create_random_worm()
- create_user_worm()
+ public
- private
Interconnect
Class Interconnect()
+ static create() : Interconnect
+ static destroy() : Interconnect
+ static is_mem() : bool
+ static is_pe() : bool
+ static get_num_faces() : unsigned int
+ static get_num_nodes_per_face() : unsigned int
+ static get_num_rows&columns_per_face() : unsigned int
+ get_num_virtual_channels() : int
+ get_channel_utilization() : int
- static instance : Interconnect
- initialize_cube_type()

Figure 52 Two Singleton Class Examples: WormManager and Interconnect

Pure Virtual Functions

The SaveRestoreInterface class provides save and restore functions that are pure virtual functions which forces derived classes to override the functions [45]. By having classes with only pure virtual functions, these classes can be declared as interfaces. This means that classes can call the save() or restore() methods without having to know what class it is saving. The following is an example of pure virtual function signatures:

```
class SaveRestoreInterface {
public:
virtual File & save(File & file) = 0;
virtual File & restore(File & file) = 0};
```

In the save/restore functionality of the inheriting class, a sentinel acts as a safeguard to assure that the correct version of code is used. The sentinel is recorded in the saved file. Upon restore, it is verified that the saved file matches the current software version.

System Design with the Standard Template Library (STL) Functions

The interconnect is modeled using a map data structure from the Standard Template Library (STL). The STL is a general purpose library of algorithms and data structures. The STL enables generic programming where reusable functions, data structures and algorithms are available for the programmer [46][47]. The interconnect is constructed of three main components: a face, a node, and a port Figure 53. For the 3D-mesh interconnect, each face has four nodes at the corners. Each node has six ports (some of which can point to nowhere). Therefore, a map is created for each component to organize the connectivity and construct the interconnect structure. The map is accessed based on the location of the face, node, or port desired to access. These locations are predefined.

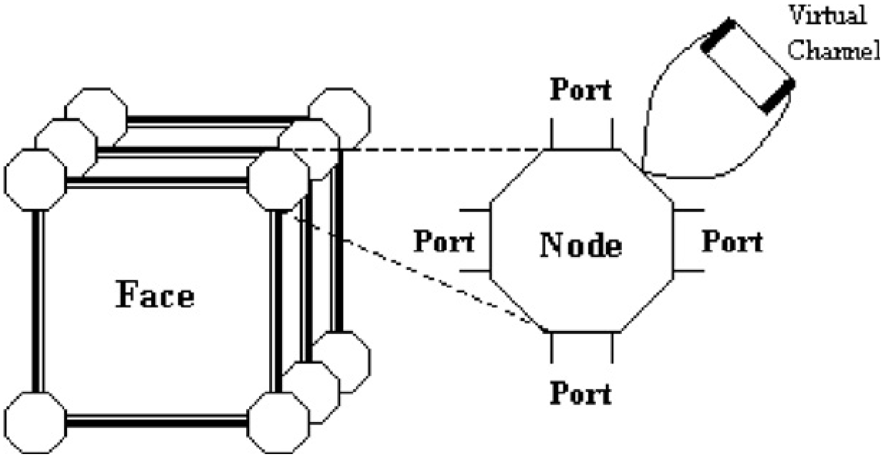


Figure 53 Layout of the Interconnect

Face #ID to face map: `typedef std::map<int, Face> FaceMap;`
Node #ID to node map: `typedef std::map<int, Node> NodeMap;`
Port #ID to port map: `typedef std::map<int, Port> PortMap;`
VC #ID to VC map: `typedef std::map<int, VirtualChannel> MemoryManager;`

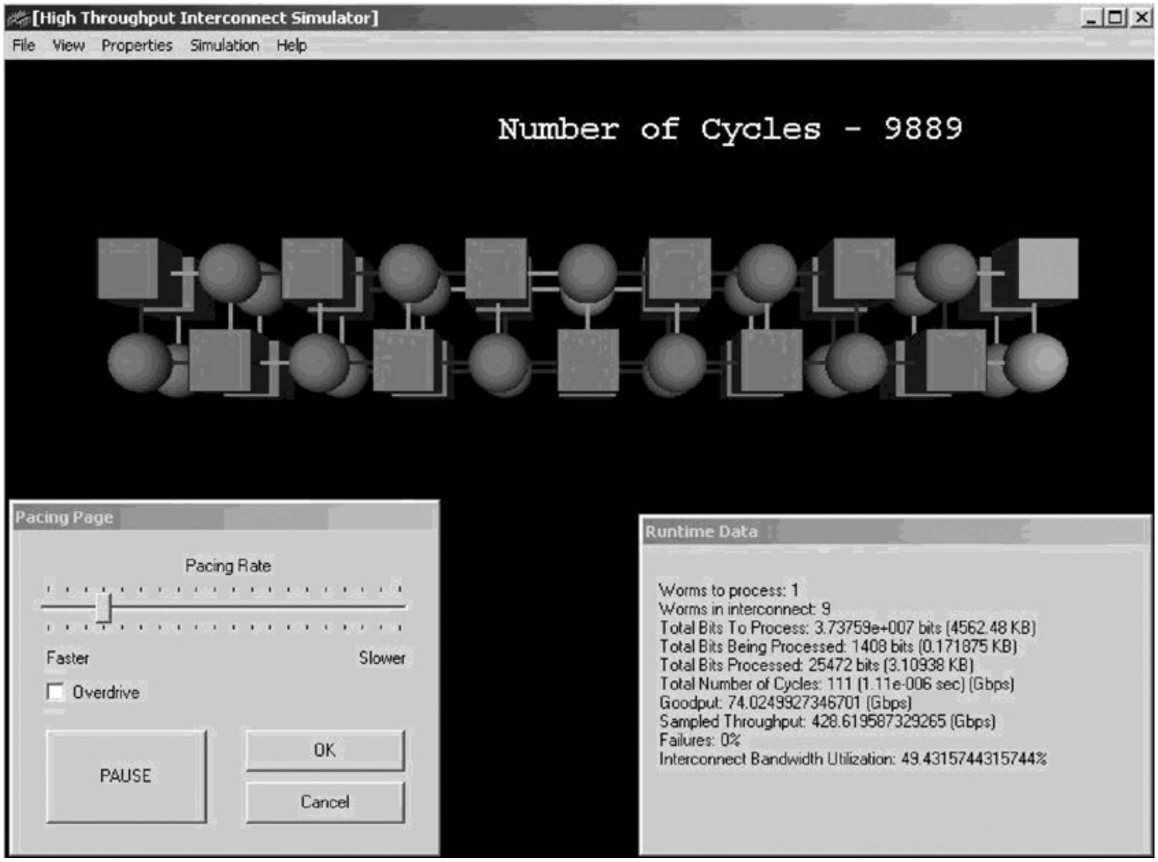
Figure 54 STL Map Declarations for the Faces, Nodes, Ports, and Virtual Channels

Simulation Data and Observations

During execution, the network simulator provides two windows to control the pacing of simulation time and the collection simulation data. The runtime data window (bottom right side of Figure 55) shows performance metrics updated on-the-fly. In addition, runtime data is also recorded in the output spreadsheet files. The pacing window (on the bottom left side of Figure 55) allows the user to control the pace of simulation that can pause it completely if desired.

Latency and throughput analysis

Latency represents the time it takes for a worm to reach its destination. Depending on the worm movement, latency sums wire transfer, switching and routing delays at each cycle. The resulting latency is an average of latencies collected from all worms modeled at the end of the simulation.


[High Throughput Interconnect Simulator]
File View Properties Simulation Help
Pacing Page
Pacing Rate
Faster
Slower
□ Overdrive
PAUSE
OK
Cancel
Runtime Data

Number of Cycles - 9889

```
Worms to process: 1
Worms in interconnect: 9
Total Bits To Process: 3.73759e+007 bits (4562.48 KB)
Total Bits Being Processed: 1408 bits (0.171875 KB)
Total Bits Processed: 25472 bits (3.10938 KB)
Total Number of Cycles: 111 (1.11e-006 sec) (Gbps)
Goodput: 74.0249927346701 (Gbps)
Sampled Throughput: 428.619587329265 (Gbps)
Failures: 0%
Interconnect Bandwidth Utilization: 49.4315744315744%
```

Figure 55 Simulation Graphical Modes with the Pacing and Runtime Data Windows

Three representative k-array n-cube interconnects were chosen for the simulations: 8-array 2-cube, 4-array 3-cube and 3D-mesh (all three interconnects have 64 nodes). Figure 56 shows a comparison among all three interconnects with VC and channel partitioning enabled. The results shown are an average of 10 different simulations with both short (128 B–1 KB) and long (1 KB–8 KB) worms and identical interconnect settings. The lowest latency was recorded for the 3D-mesh, while the 4-ary 3-cube network has slightly higher latency than the 3D-mesh. Throughput is measured by taking samples of the total bits processed within the interconnect at each cycle.

Throughput significantly increases when VCs are enabled since they allow more worms to occupy the interconnect without transmission failures. The highest throughput was reached by the 3D-mesh interconnect for both short and long messages.

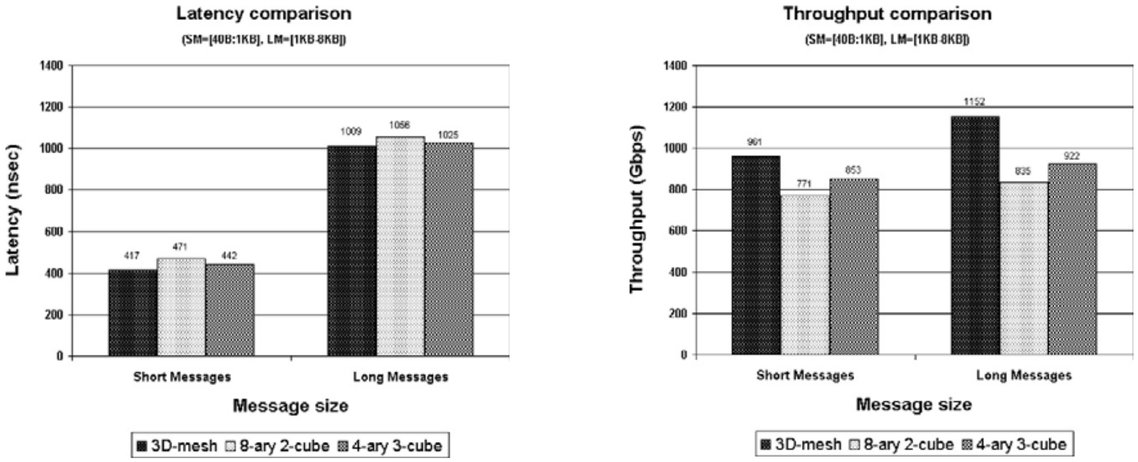


Figure 56 Latency (Left) and Throughput (Right) Comparisons Between 3D Mesh, 8-Array 2-Cube and 4-Array 3-Cube

Worm Allocation and Distribution

Worm allocation and distribution measurements, depicted in Figure 57, show three groups of worms: worms that are currently propagating in the interconnect, worms that are waiting in jar to be modeled and worms that are finished and reached their destinations. The figures show that the number of currently modeled worms (worms in the interconnect) increases as the number of worms waiting in the jar and the number of already modeled worms (finished) decreases.

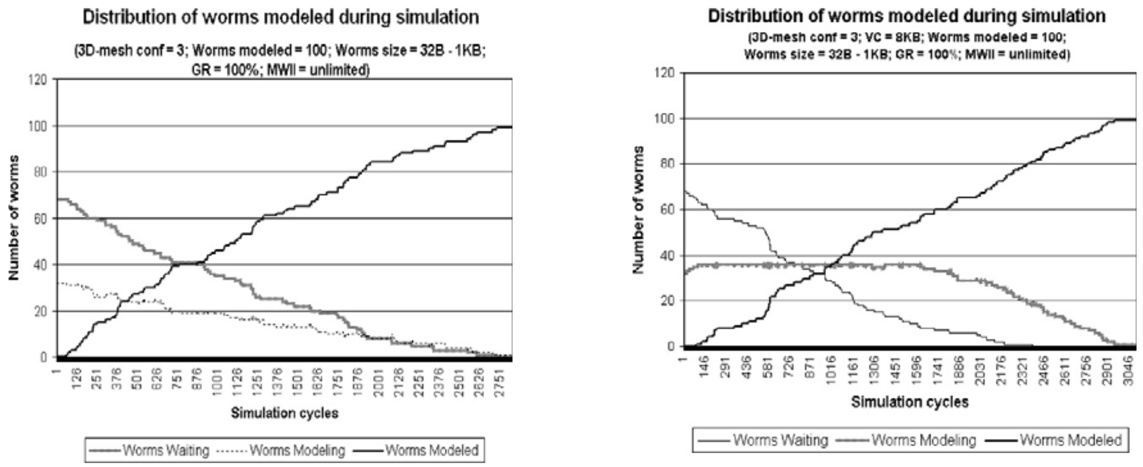


Figure 57 Worm Allocation and Distribution with (Right) and without (Left) Virtual Channels

When VCs are enabled, more worms occupy the interconnect at a faster rate than without VCs. This shows that as more worms are modeled, the number of worms waiting to be modeled diminishes. It is also noticeable that when VCs are enabled more simulation cycles are required.

Routing Accuracy

Routing accuracy measures how close the actual path of each worm is to its shortest path. Routing accuracy is calculated by taking the ratio between the shortest path possible to the actual path taken; this signifies the worm's deviation from its shortest path. Figure 58 shows a simulation of 100 worms using 3D-mesh interconnect with VCs disabled and no sub-channeling. At the top of the figure, the top-most line portrays the percentage of deviation from the shortest path. The top line shows, for example, a triangular point for a certain worm is at 100, that means the worm has taken the shortest path possible. If the value of the line is equal to 20, the worm deviated from its shortest path by 80% (and has taken more channel links).

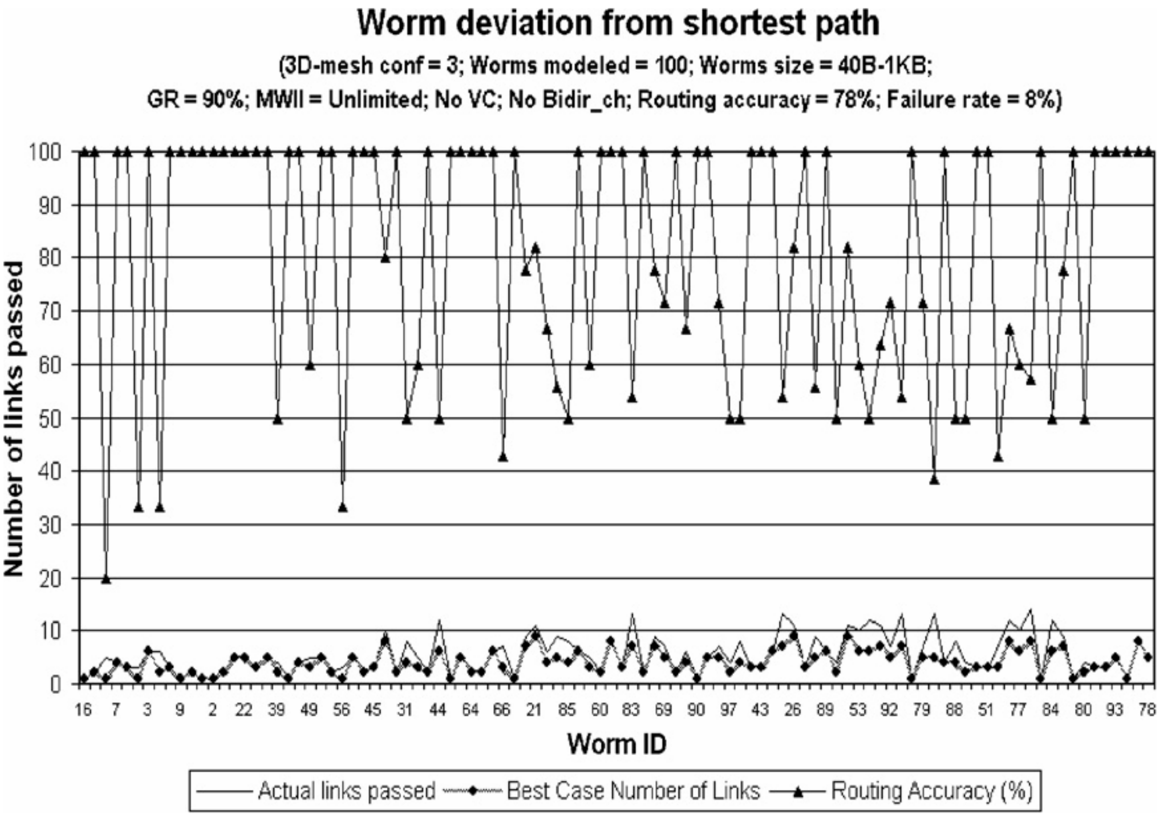


Figure 58 3D Mesh Worm Deviation from its Shortest Path

On the left-hand side, Figure 58 shows the number of links passed for each worm modeled using 3D-mesh interconnect. On the bottom part, the deviation of each worm (top line) from its shortest path (bottom line) is shown. Therefore, when both lines completely overlap each other for a certain worm, that worm has taken the shortest path. For example, worm 44 took a path passing 12 nodes to get to its destination, but it should have taken 7. As the number of channel links passed increases with respect to the shortest path possible, the thin line becomes further apart from the thick line. It turns out, the path the worm takes depends on the traffic load at certain nodes of the interconnect. As the load increases, most worms deviate from their shortest path and adaptively propagate to their destination avoiding areas of hot-spots [39].

Interconnect and Bandwidth Utilization

Interconnect bandwidth utilization measures the number of occupied channels (or sub-channels) with respect to the total number of channels available in the interconnect. Figure 59 portrays that the highest bandwidth utilization is achieved by using the 4-array 3-cube network, while the 8-array 2-cube has the lowest utilization rate. Sub-channeling improves bandwidth utilization as the channel is partitioned into more sub-channels. The combination of VCs and SCs brings all interconnects close to their full capacity.

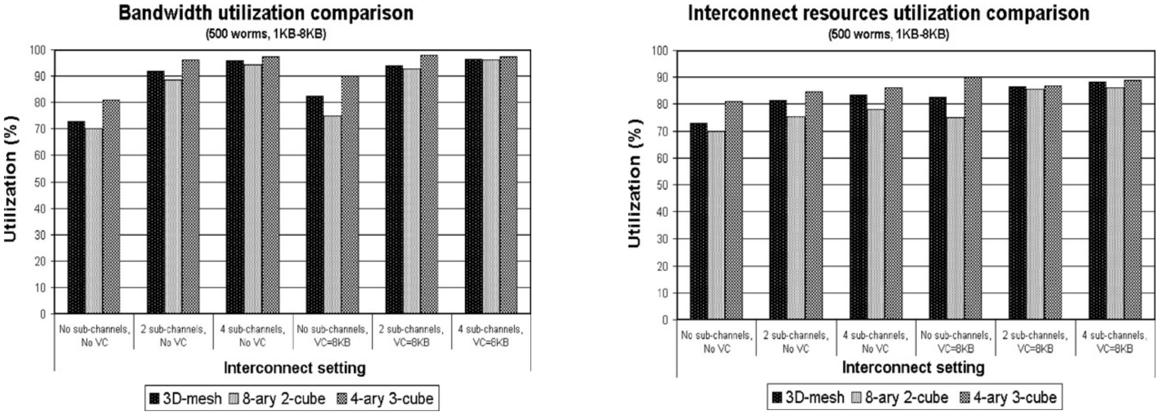


Figure 59 Bandwidth (Left) and Interconnect (Right) Utilization

Interconnect utilization counts the number of busy ports within each traffic controller per simulation cycle. At the end of the simulation it provides the average number of ports that were set to busy status out of the total number of ports available in the interconnect throughout simulation. The results of interconnect utilization show very close relationship to bandwidth utilization. Again, 4-array 3-cube ports are set to busy status more often than the 3D-mesh or 8-array 2-cube. Although interconnect utilization seems an equivalent measure to bandwidth utilization, it is a little different since the port status is not directly related to the channel usage. An output port can stay in the not-busy state if a worm that intends to use it is buffered into virtual channels. Since each traffic controller has a minimum of four ports, a worm entering from a different direction can utilize the channel connected to the non-busy port.

Failure Rate

Failure rate is a measure of the number of worms, out of the total number of worms generated that were retransmitted during simulation. Retransmission takes place when a worm is blocked and it cannot obtain the resources it requires to maintain an active status within the interconnect. For example, when VCs are disabled, then a worm will require retransmission if it cannot be routed to any output port within a certain node for more than one simulation cycle. Figure 60 depicts a failure rate comparison for all interconnect types with VC switched to enabled/disabled. This figure shows that using VCs significantly reduces failure rate. Moreover, the size of the VC has a major effect on failure rate as well. As the size of the VC increases more worms can be buffered for longer periods of time within each node instead of failing and being retransmitted [38].

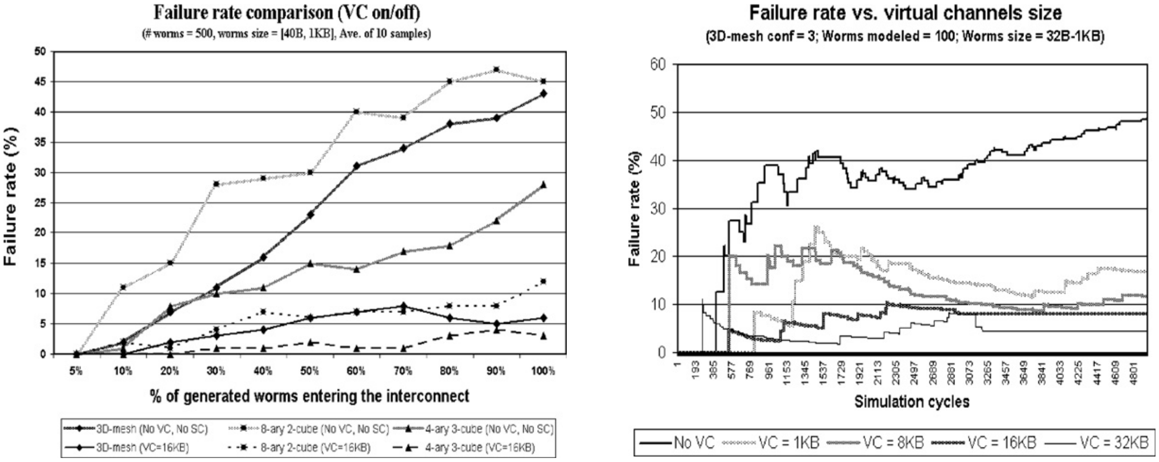


Figure 60 Worm Failure Rate Comparisons with and without Virtual Channels (Left) and with Different Virtual Channel Sizes (Right)

Routing Accuracy vs. Hot-Spot Nodes

In this simulation, the paths taken by all worms using 3D-mesh, 8-array 2-cube and 4-array 3-cube interconnects were recorded. Then, the paths were analyzed to collect the nodes which were most frequently used and as a result caused other worms to deviate from their shortest path to avoid transmission failure.

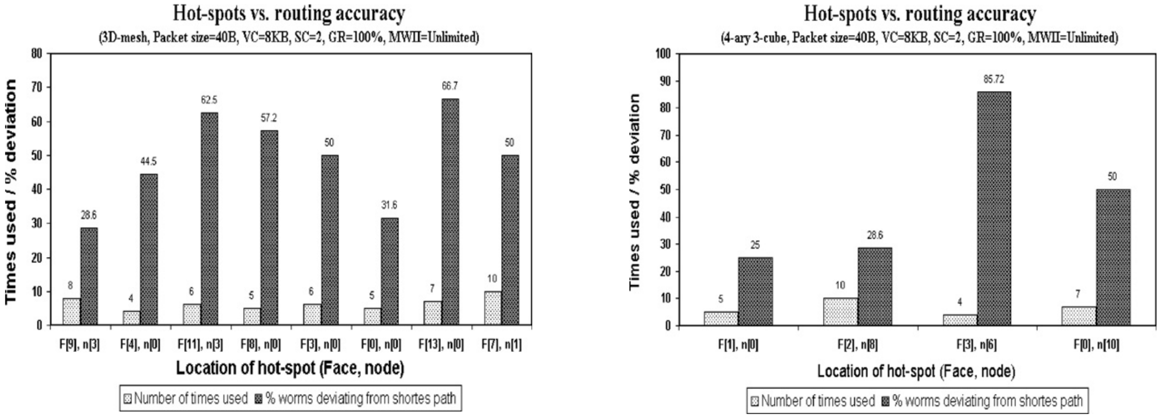


Figure 61 Hot Spots Versus Routing Accuracy

Results given in Figure 61 show that some hot-spot nodes caused approaching worms to deviate from their shortest path by 50–60% more channel links than the shortest path available. For example, the hot-spot in face 11 node 3 (F[11], n[3]) caused six approaching worms to deviate from their shortest path by 62.5%. Traffic is randomly generated with random message lengths and from random nodes. Since the adaptive routing algorithm changes the path the worms take in each simulation, every simulation creates hot-spots in different locations and in different frequencies. The right diagram in Figure 61 shows a hot-spot which occurred in face 3 node 6 (F[3], n[6]) that caused approaching worms to deviate from their shortest path by an average of 85%. Although only few hot-spots occur per simulation, their effects on performance were significant. As the rate of hot-spot increases (a function of traffic load), worms tend to deviate from their shortest path more frequently and, as a result, the overall interconnect latency increases.

K-Array N-Cube Interconnect Performance Comparison with Common Interconnects

In this section, 3D-mesh, 8-ary 2-cube, and 4-ary 3-cube interconnects are compared with other currently used high-performance interconnect technologies such as Hypertransport (HyperTransport Consortium, 2005), Infiniband (Infiniband Trade Association, 2000) and PCI-Express (PCI Special Interest Group, 2003; Sassone, 2003).

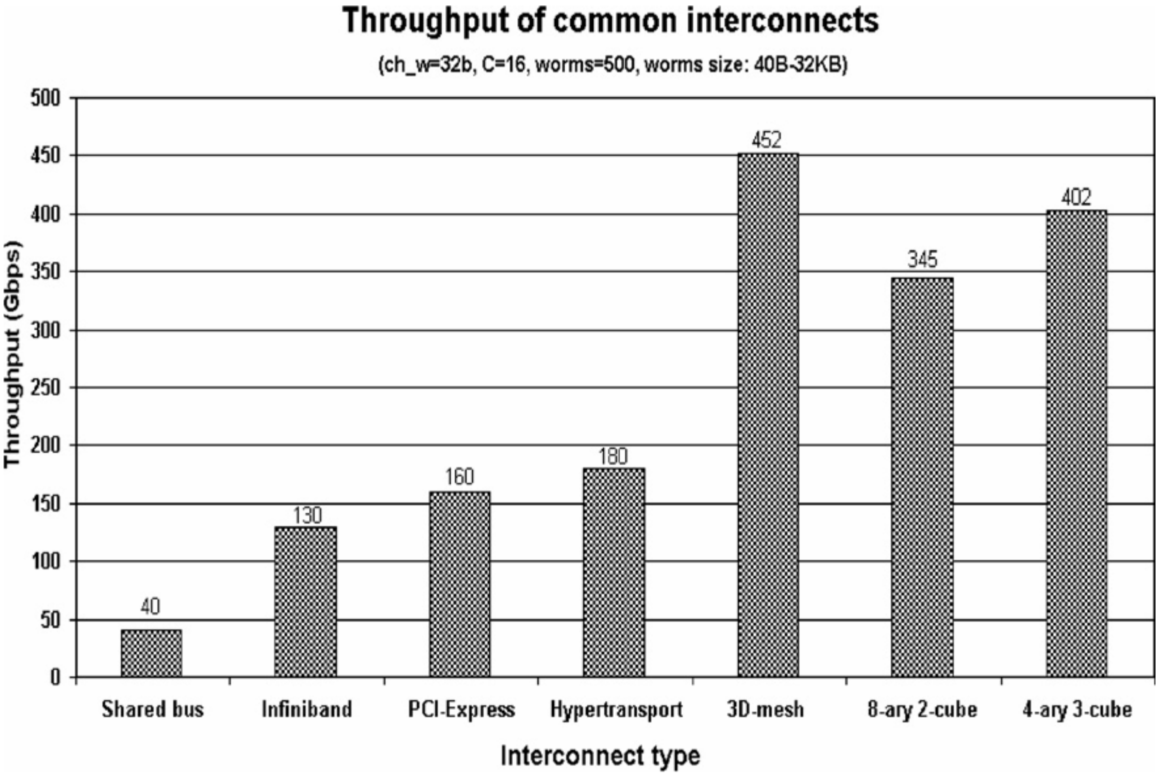


Figure 62 Comparison of Different Interconnects

Reported results provided by each individual vendor were used to compare with the results from this simulation. In addition, the performance properties of these technologies take into account a constant channel size of 32-bits and a single communication link. For the 3D-mesh interconnect the settings are: channel width is 32 bits, interconnect size is 16 cubes, number of worms generated is 10, each worm is 1KB in size.

Virtual channels as well as channel partitions were enabled. The throughput comparison results are shown in Figure 62. The throughput values of the 3D-mesh, 8-ary 2-cube and 4-ary 3-cube interconnects represent the average throughput of each interconnect. 3D-mesh shows superior results compared to all of its competitors reaching a peak throughput of 452 Gbps (about twice the throughput of the best interconnect available not including the other types of k-array n-cubes tested).

Cluster Leader Logic Evaluation and Results

Simulation experiments are conducted with enforced directional traffic patterns. Two important results are presented in this section: power consumption per clusterhead and average queuing delay for each clusterhead. Results in terms of message overheads, number of clusterheads, power consumption, and queuing delay reveal that system performance is enhanced when clusterheads are chosen considering the direction of the traffic flow.

The CLL Simulator

In order to test the feasibility of the proposed CLL algorithm, a simulator was created to validate the architecture and find the expected performance. NS-2 [55] was evalutated, but it did not have native GPS-QHRA support. Also, it was important to neglect conventional cluster-based routing algorithm shortcomings for dropping messages because it would be difficult to figure out if messages are dropped from the CLL algorithm or the routing algorithm choices.

Thus, a custom simulator was written to create an omniscient routing protocol which would not drop messages. The simulator console application is written in C++, is object oriented, and implements advanced concepts such as templates and generics, and is built from some of the simulation infrastructure as the simulator used for the k-array n-cube simulator [25]. The simulator is composed of two executables: a scenario generator and the CLL simulator. A configuration file was created to allow the tester to configure the static constant variables defined above. The simulator is event-based and scenario file driven.

The benefits of having scenario files include the ability to tweak test cases without having to recompile code, the abilities for a human to read and edit the file, and the capability to trace each test case to a scenario which can be re-run to double-check a concept. The scenario format allows the tester to place nodes in cells, send time-stamped messages between nodes, time-stamp node movement, and add comments to the scenario file as appropriate.

The implementation of the simulator follows the CLL algorithm very closely; the simulator varies from the real world because it is a single threaded single process and does not have true simultaneous multithreaded communication. The benefits of having simultaneous communications would not directly prove or disprove the CLL algorithm; it would affect the performance of the algorithm since collisions would occur and message would be dropped and re-transmitted more frequently.

When the simulator is executed, the simulator reads the scenario specified, populates each node with its respective messages and movements, executes the simulation by stepping through simulation time, and shuts down the program and logs statistics when complete. Validation scenarios were created with hand-calculated results to test different aspects of the simulation to expose bugs with both the implementation and the algorithm design and then later were used to fix the bugs. Once the validation scenarios passed testing, scenarios were created to compare native GPS-QHRA to GPS-QHRA with CLL.

Scenario Design

Once the simulator functionality stabilized and results matched hand-calculated results, several larger scenarios were created to prove the concepts of the CLL algorithm. The scenario set is divided into two classes: the slash scenario and the random scenario.

The slash scenario set organizes 76 nodes into a slash (a diagonal formation from the top-left to the bottom-right) formation within a 128 cell region where only 37 cells are occupied. There are several reasons for picking a slash pattern:

- The pattern represents a two-lane road with network traffic traveling one way against the top part of the slash and the opposite way against the bottom part of the slash.
- Cell fanning could be double-checked against expectations performed in hand-calculations.
- A bottleneck is created which will force clusterheads to split.

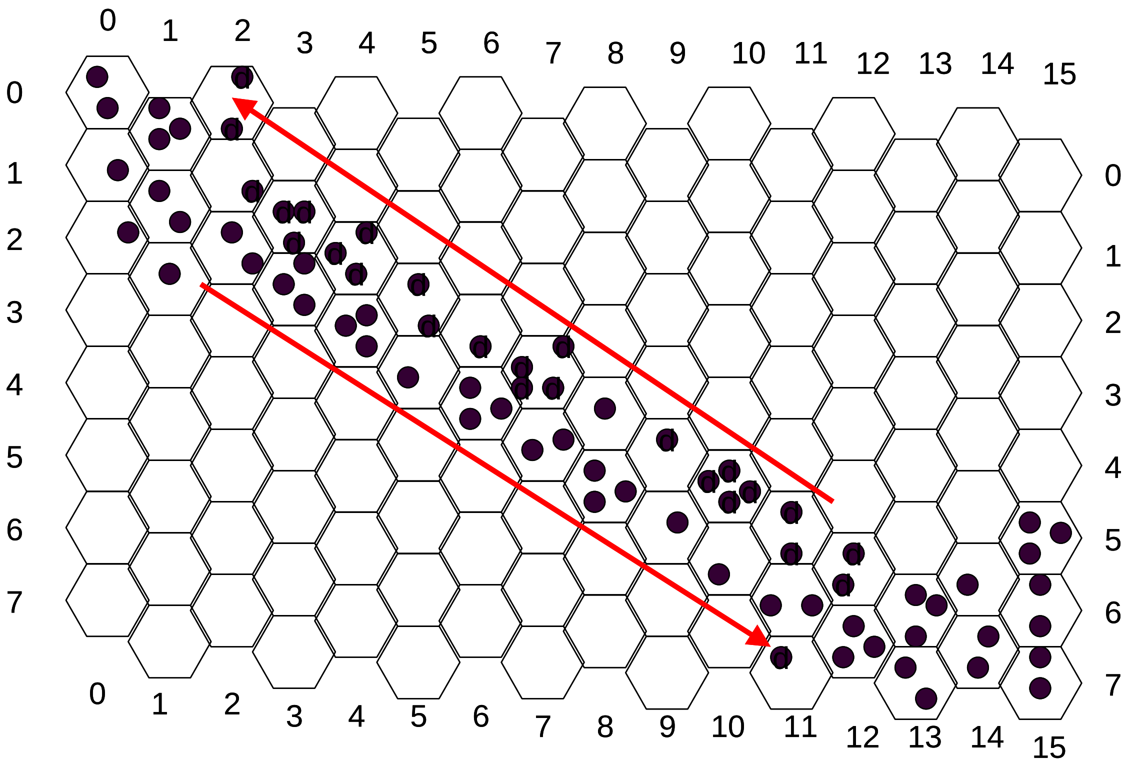


Figure 63 The Slash Scenario before Any Node Movement

There are two versions of the slash scenario: with and without node movement Figure 63. The arrow represents the direction of movement when the nodes start moving. The algorithm is designed to not care if nodes are stationary or moving. The affects of sending a message and then moving could cause a dropped packed: this is why the test cases are differentiated. These are some other constraints imposed on this scenario set.

- There are no holes in connectivity that would cause the routing algorithm to drop packets.
- There are at most 4 nodes in a cell.
- Messages originate in the bottom right and move up-left or messages originate on the top and move down-right.

The slash-movement scenario is the same as the first except that nodes move at almost random times. The movements were designed not to break connectivity, so they could not be truly random movements. But, the movements create different situations where clusterheads would be forced to split, join, or do nothing based on the movements.

The second scenario set, the random scenarios, were also created with 250 randomly distributed nodes within an 8 x 16 play-box. Scenarios were created to inject 2000, 4000, 6000, 8000, and 10,000 messages over a 200 second time period where the message origination and destinations were random but did not start and end in the same cell. Within this 200 second time period, 10 nodes moved in a manner to cross the boundaries of their cells to cause a state change from clusterhead to subordinate node or vice versa or the clusterhead kept its state; at a minimum one example of each situation was tested. The simulation should expect between 10-50 messages per second to be generated. This translates approximately into each node sending a message between 5 to 25 seconds. These scenarios were run over 150 times each with variations to the configuration files producing over 750 different results for this vignette. The quantity of the variations were intended to find the best clusterhead configurations for each situation (one, two, or four clusterheads) so these results could be compared and contrasted.

Results

The results are intended to prove or disprove the CLL algorithm concept that includes the concept of cell fans. The proof of the concepts is achieved when enough test cases are run with different parameters to see that in each case the clusterhead overloads converge to a low value when parameters are altered. A clusterhead overload occurs when the cluster leader cannot create a new cluster leader to share its load. In order to prove the concepts, 15 variables were examined to help identify trends and ways to improve the algorithm and scenario design. 8 of those variables represent the configurable variables. The 15 variables monitored are shown in Table 26.

One way the algorithm design was improved was to create the PurgeWeightsWhenCHSplits variable. Before this variable existed, clusterheads always handed all learned data to newly created clusterheads.

It was found through experimentation that this caused the clusterhead splitting to be too aggressive for newly created clusterheads. By creating this variable and setting it to true, the clusterhead gives a chance to observe its busy cell fans data flow for itself. In all cases, the number of clusterhead overloads increased and the clusterhead stability decreased significantly when the value is false. In addition to the observances above, for moving node scenarios, additional clusterheads were created when the value is set to false.

Scenario design was improved as well. A special test case scenario was designed based on these parameters. Certain test cases with moving nodes had dropped packets that should not have dropped packets. A scenario was created to test nodes moving and communicating at the same time. The movements included clusterheads with and without subordinate nodes. The communications included transmitting, receiving, and hopping messages. This situation ended up being the most complex to fix since movement of nodes can occur anywhere in the execution of the algorithm; but the fixes applied increased the accuracy of the results of the simulator significantly.

The x-axis in Figure 64 represents different configurations for the same slash scenario run for these tests: one through four clusterheads allowable per cell. When one clusterhead is present, this case reflects the native GPS-QHRA protocol. The y-axis of the left diagram represents a count for each time a clusterhead is overloaded and has to queue a message because it cannot share its workload with other nodes in its cell. The y-axis of the right diagram represents the final clusterhead count.

Table 26

Simulation Variables Monitored

Variable Name	Definition
InitialClusterheads	Before the simulation starts, this is the count of clusterheads selected based on lowest id.
FinalClusterheads	When the simulation ends, this is the result of all present clusterheads.
ClusterheadSplits	The number of times any clusterhead splits.
ClusterheadJoins	The number of times when a node joins a different clusterhead.
ClusterheadStability	This number is incremented each time getEffectiveNodeCount() <= activation level.
ClusterheadPotentialOverload	This number is incremented each time getEffectiveNodeCount() > activation level.
ClusterheadOverload	Equal to ClusterheadPotentialOverload – ClusterheadSplits.
C2CRelay	Incremented each time a clusterhead sends a message to another clusterhead.
C2SRelay	Incremented each time a clusterhead sends a message to a subordinate node.
S2CRelay	Incremented each time a subordinate node sends a message to a clusterhead.
NotRelayed	Incremented when the cell fans determine that a clusterhead should not relay a message.
Messages Delayed	Incremented when messages are delayed because of queuing delays.
Power Consumption	Calculates the amount of power used for message transmission.
Dropped	Number of messages not received by the intended recipient.
Total Simulation Runtime Cycles	Total amount of time taken to run the scenario.

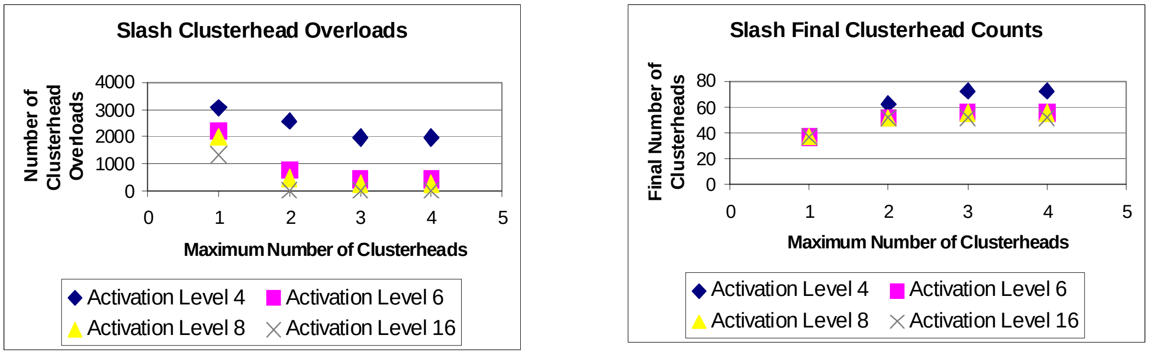


Figure 64 The Slash Scenario Results with No Node Movement – Clusterhead Overloads (Left) and Clusterhead Counts (Right)

As mentioned earlier, one of the main performance metrics is the clusterhead overload value. The number of overloads is affected by the ClusterheadDivisionActivationLevel. As the EffectiveNodeCounts are calculated, they are compared to the ClusterheadDivisionActivationLevel which is a constant value. If the EffectiveNodeCount values are consistently below the activation level for a long period of time, the clusterhead will try to become a subordiante to another clusterhead in its cell (if one is available) by joining its cell fan with the other clusterhead and switching its state machine to a subordinate node. When the EMA exceeds the activation level, the clusterhead attempts to split its cell fan with another subordinate node (if available) and switch to the clusterhead state machine. If in that case no subordinate node is available, then the clusterhead is overloaded especially in the case in Figure 64 when the maximum number of allowable clusterheads is one. Ideally, as the activation level increases, the number of clusterhead overloads should decrease. Higher activation levels make the algorithm less aggressive since the clusterheads split less often and allow more data to flow through them.

The next important metric to measure is the final number of clusterheads. The initial number of clusterheads may differ than the final count of clusterheads since there will be splitting and joining throughout the simulation. The converged value would determine the optimal amount of clusterheads this scenario could have. The results in Figure 64 show the clusterhead overload value stabilizes as expected and achieves zero clusterhead overloads in these test cases when the activation level is 16. As more clusterheads are allowed, fewer overloads occur (left diagram). Higher activation levels cause fewer clusterheads to be created (right diagram).

As shown in Figure 65, there are fewer clusterhead overloads with fewer clusterheads existing in the end of the simulation when the nodes are moving. As more clusterheads are allowed, less overloads occur (left diagram). Higher activation levels cause fewer clusterheads to be created (right diagram). These numbers appear to converge at about 52 for the stationary scenario and about 50 clusterheads for the motion scenario. These results are proof that the concept of the CLL algorithm converges to a meaningful value. These are meaningful values because 37 cells are occupied meaning that about 74% of the cells have one clusterhead and about 26% have multiple clusterheads. The algorithm does not appear too agressive.

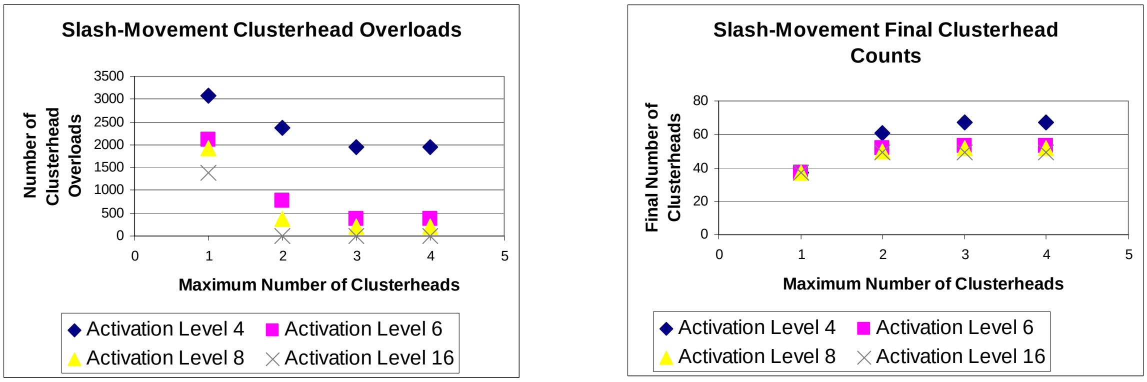


Figure 65 The Slash-Movement Scenario Results

Once convincing results were obtained, the performance of GPS-QHRA and CLL was measured and compared. As mentioned in the background section, GPS-QHRA is similar to LCC; a comparison to Leader Election Algorithm was not performed because the experimentation is not geared to measuring the performance difference between having table or tree data structures.

The performance of GPS-QHRA and CLL was measured and compared. The five randomly distributed scenarios described earlier were created and run over 750 different ways. This includes five scenarios times three configurations (one, two, or four clusterhead maximum) times 50 different values for activation level that are tweaked by experimentation to produce a level playing field between the test cases.

Two important results are presented in this work: power consumption per clusterhead (Figure 66) and average queuing delay for each clusterhead (Figure 67). The power consumption compares between GPS-QHRA (1 clusterhead) and CLL with 2 or 4 maximum clusterheads in a cell. Depending on the amount of messages sent in the same amount of time, the CLL algorithm can realize a maximum of 45% power savings. The queuing delay also compares between GPS-QHRA (1 clusterhead) versus CLL (2 or 4 clusterheads maximum per cell). There are noticeable improvements (25% maximum) between GPS-QHRA vs. 2 CH CLL. However, differences between 2CH and 4CH are less than 1%.

Both of these results were run with one, two, and four maximum allowable clusterheads for all of the scenarios. The one clusterhead maximum runs are meant to mimic native GPS-QHRA. All allowable configurations for the maximum number of clusterhead were initially run (one through seven clusterheads because there are at most seven different directions). However, eventually, only the one, two, and four maximum clusterheads were reported because other allowances did not show any meaningfully different results. It is hypothesized that more nodes and/or messages might have shown more of a significant distribution between having varying maximum amounts of clusterheads.

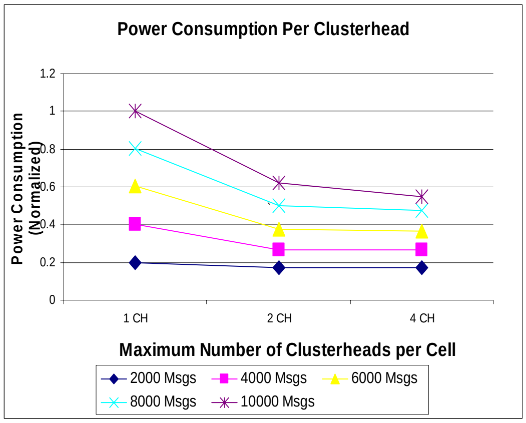


Figure 66 Power Consumption Comparisons Between GPS-QHRA and CLL

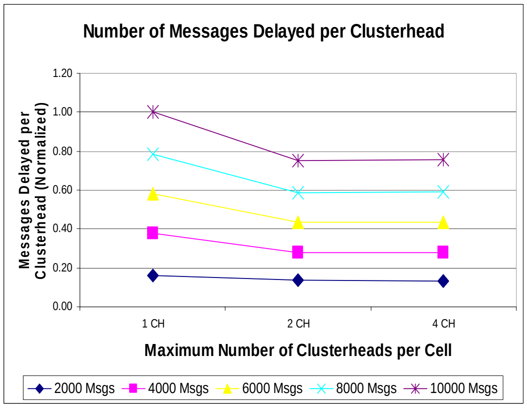


Figure 67 Queuing Delay Comparisons Between GPS-QHRA and CLL

The results for power consumption show up to 45% power savings when using CLL over GPS-QHRA. Power is conserved because clusterheads distribute the messages that they need to transmit because of cell fanning. So, for instance, two clusterheads transmitting one message each use half the amount of transmission power of one clusterhead transmitting two messages. The best power savings is realized when more messages are sent with more allowable clusterheads per cell than with GPS-QHRA.

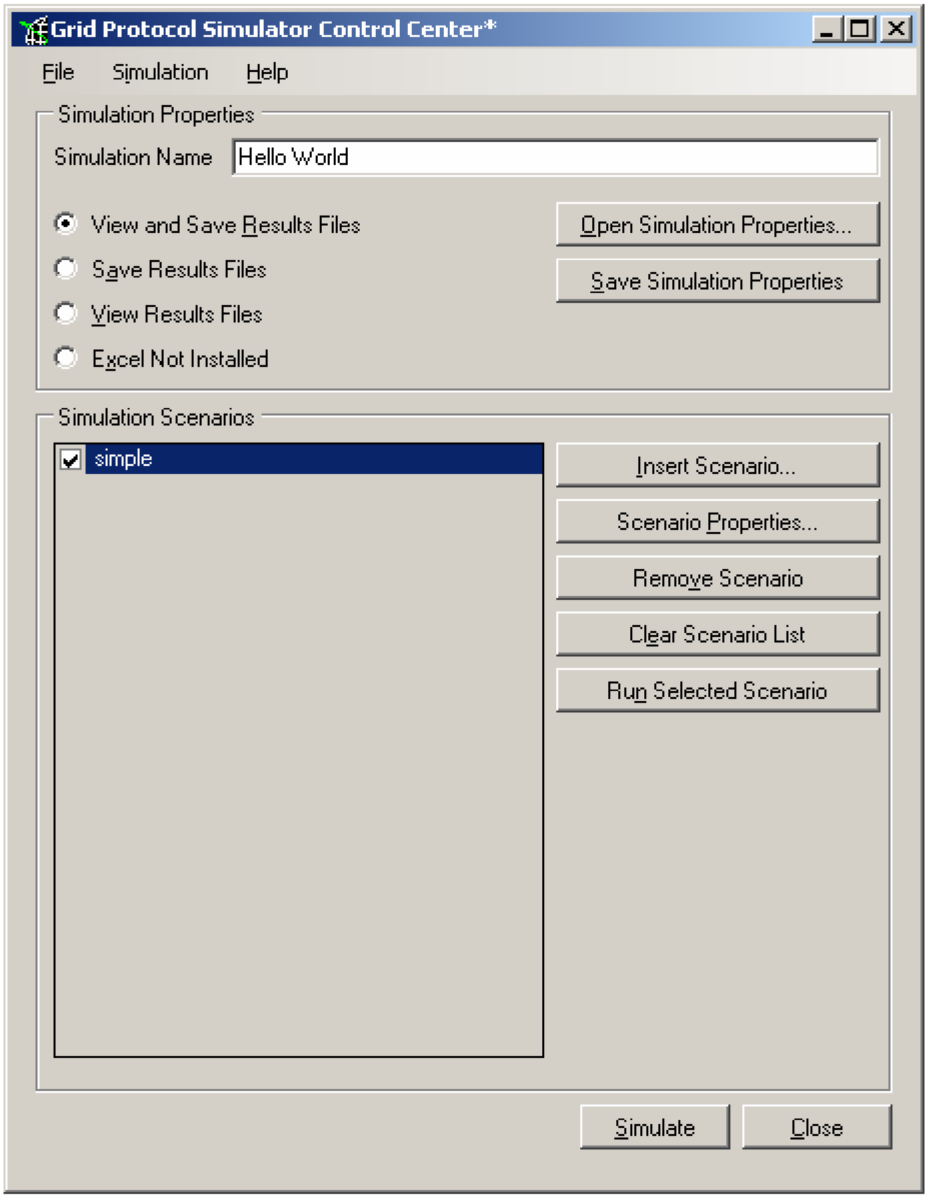
Queuing delays are also improved when CLL is used over GPS-QHRA up to 25%. The effects of CLL versus GPS-QHRA are noticeable; this is most likely because of the receiver side filtering available from cell fanning which is done before queuing takes place. However, unlike the power savings results, the differences between having two or four maximum clusterheads per cell is negligible.

The Grid Protocol Simulator Evaluation and Results

The grid protocol was simulated using the Grid Protocol Simulator software suite. The suite is composed of three major components: the Control Center, the Scenario Editor, and the Grid Protocol Simulator. These three software applications are discussed within this section along with a general discussion about the design and implementation of the software.

The Control Center shown in Figure 68 is the entry point of the program and enables the user to start the Scenario Editor and the Grid Protocol Simulator. The interface allows the user to schedule multiple runs to happen sequentially after each other which automates the testing and execution of the simulation. The user can also configure runtime parameters such as logging, suppressing error messages, and creating situations when events are blacklisted. Another useful feature is that the Control Center configuration can be saved in and restored from “gsp” files. The gsp files allow you to run the same experiment again or to restore the experiment, add or remove tests, and then run the experiment.

The Scenario Editor shown in Figure 73 allows users to create and edit scenarios to run in the simulation. The main outputs from scenario generation are the network tree, the selection of which deployment environment (Figure 69) to simulate, and the event list of events to run through the network. The Scenario Editor allows the user to use the “Generate Network” feature (Figure 70) to automatically populate routers and switches within a specified IP address range. The user can also manually add and remove nodes from the network one at a time. Once the network is laid out, the user can either select where to place VO host devices or use the "Generate VOs" feature (Figure 71) to automatically place where VOs are located in the network tree. This will allow you to create events either manually or automatically. When using the "Event Generator" (Figure 72), the user can choose what times and VO hosts to send the SIGNUP and ADVERTISE events to.

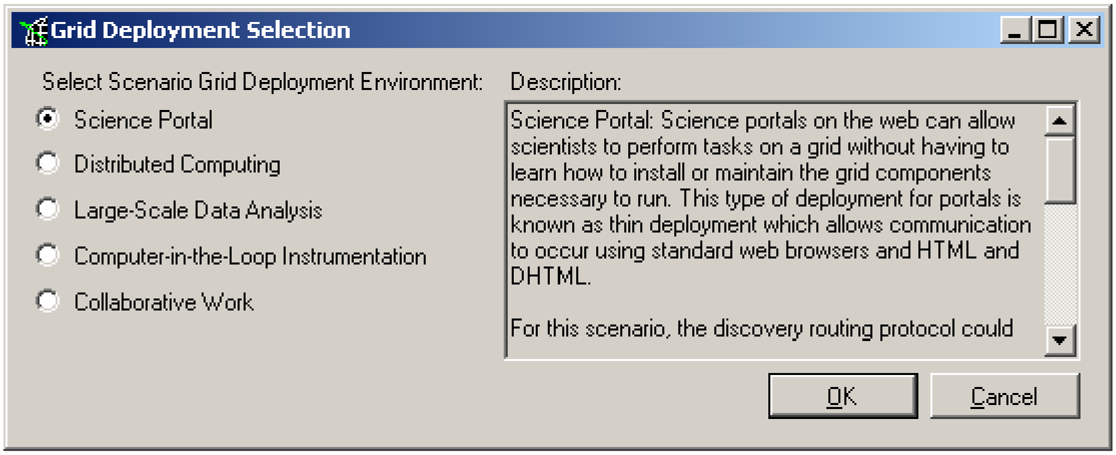


Simulation Name: Hello World

● View and Save Results Files
○ Save Results Files
○ View Results Files
○ Excel Not Installed

☑ simple

Figure 68 The Grid Protocol Simulator Control Center


Grid Deployment Selection

Select Scenario Grid Deployment Environment:

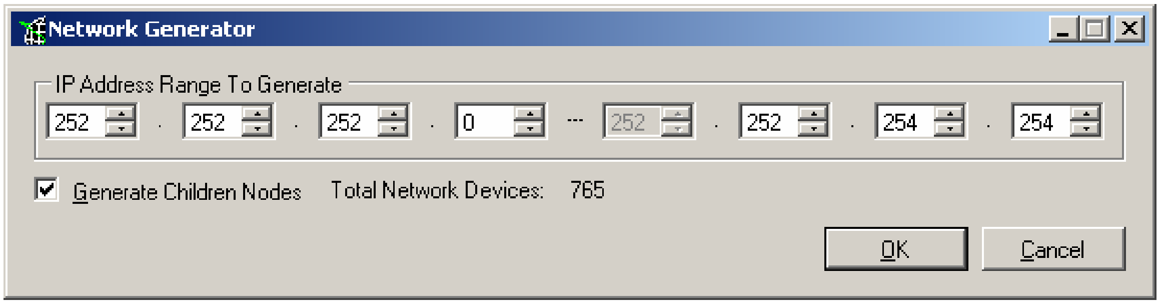
● Science Portal
○ Distributed Computing
○ Large-Scale Data Analysis
○ Computer-in-the-Loop Instrumentation
○ Collaborative Work

Description:

Science Portal: Science portals on the web can allow scientists to perform tasks on a grid without having to learn how to install or maintain the grid components necessary to run. This type of deployment for portals is known as thin deployment which allows communication to occur using standard web browsers and HTML and DHTML.

For this scenario, the discovery routing protocol could

Figure 69 Grid Deployment Selection Form

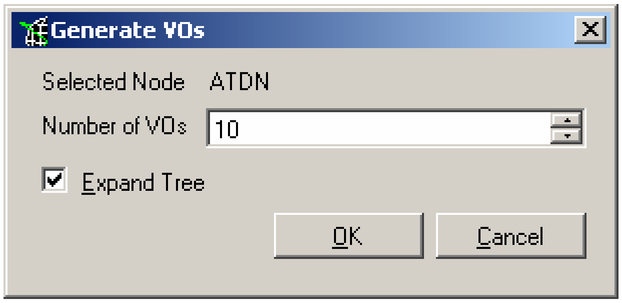

Network Generator

IP Address Range To Generate

252 . 252 . 252 . 0 --- 252 . 252 . 254 . 254

☑ Generate Children Nodes   Total Network Devices: 765

Figure 70 The Network Generator Form

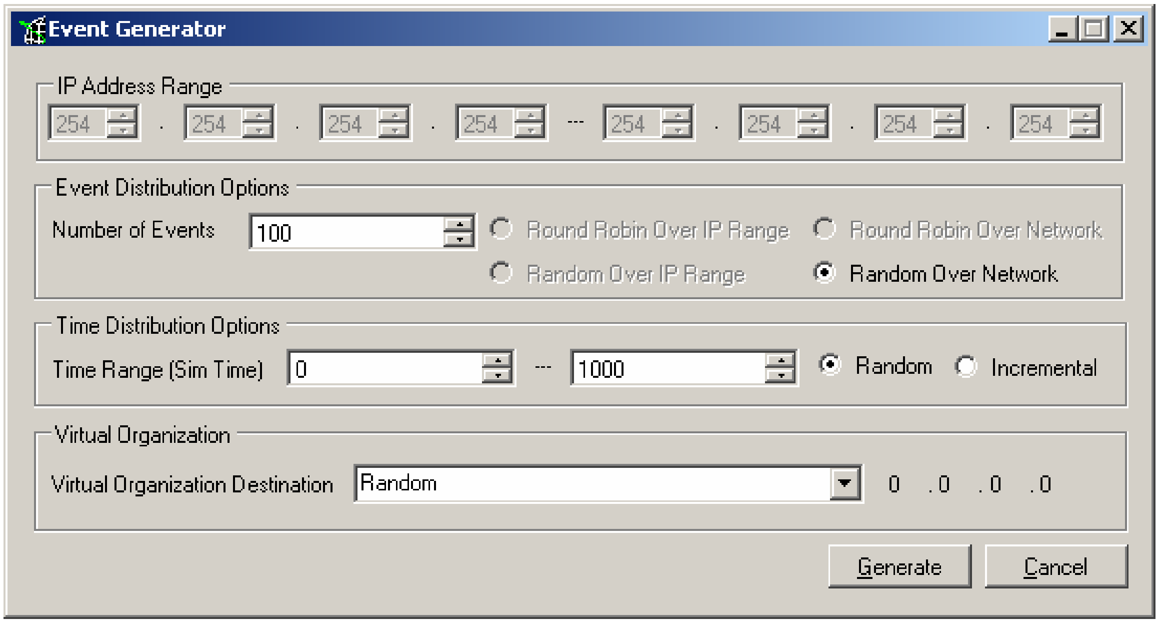

Generate VOs

Selected Node ATDN

Number of VOs 10

☑ Expand Tree

Figure 71 The Generate VO Hosts Form



IP Address Range

254 . 254 . 254 . 254 --- 254 . 254 . 254 . 254

Event Distribution Options

Number of Events: 100

○ Round Robin Over IP Range ○ Round Robin Over Network
○ Random Over IP Range ● Random Over Network

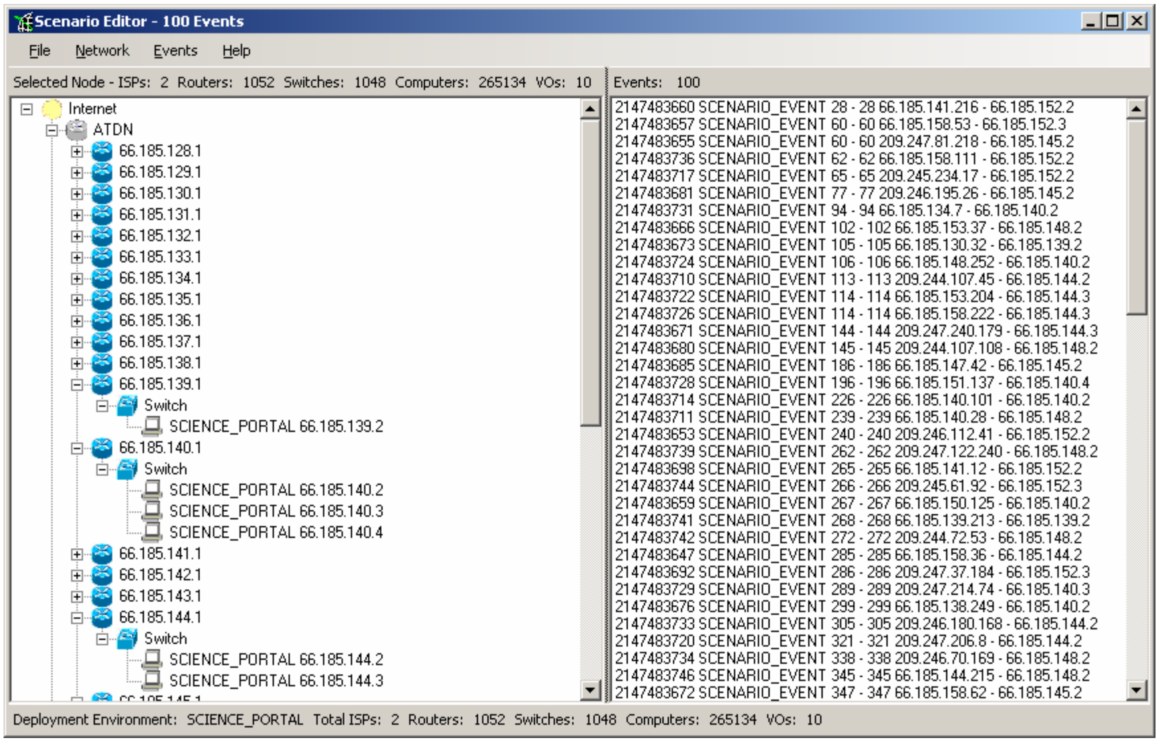
Time Distribution Options

Time Range (Sim Time): 0 --- 1000 ● Random ○ Incremental

Virtual Organization

Virtual Organization Destination: Random 0 .0 .0 .0

Figure 72 The Event Generator Form



Selected Node - ISPs: 2 Routers: 1052 Switches: 1048 Computers: 265134 VOs: 10

```
⊟ Internet
  ⊟ ATDN
    ⊞ 66.185.128.1
    ⊞ 66.185.129.1
    ⊞ 66.185.130.1
    ⊞ 66.185.131.1
    ⊞ 66.185.132.1
    ⊞ 66.185.133.1
    ⊞ 66.185.134.1
    ⊞ 66.185.135.1
    ⊞ 66.185.136.1
    ⊞ 66.185.137.1
    ⊞ 66.185.138.1
    ⊟ 66.185.139.1
      ⊟ Switch
          SCIENCE_PORTAL 66.185.139.2
    ⊟ 66.185.140.1
      ⊟ Switch
          SCIENCE_PORTAL 66.185.140.2
          SCIENCE_PORTAL 66.185.140.3
          SCIENCE_PORTAL 66.185.140.4
    ⊞ 66.185.141.1
    ⊞ 66.185.142.1
    ⊞ 66.185.143.1
    ⊟ 66.185.144.1
      ⊟ Switch
          SCIENCE_PORTAL 66.185.144.2
          SCIENCE_PORTAL 66.185.144.3
    ⊞ 66.185.145.1
```

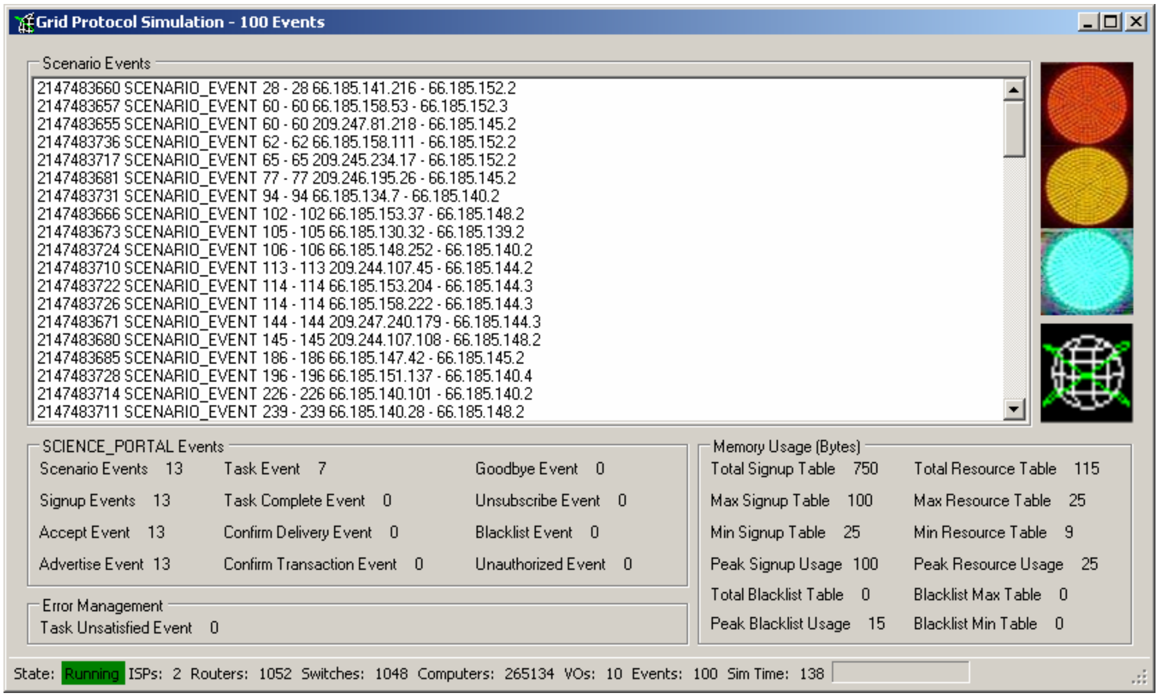
Events: 100

```
2147483660 SCENARIO_EVENT 28 - 28 66.185.141.216 - 66.185.152.2
2147483657 SCENARIO_EVENT 60 - 60 66.185.158.53 - 66.185.152.3
2147483655 SCENARIO_EVENT 60 - 60 209.247.81.218 - 66.185.145.2
2147483736 SCENARIO_EVENT 62 - 62 66.185.158.111 - 66.185.152.2
2147483717 SCENARIO_EVENT 65 - 65 209.245.234.17 - 66.185.152.2
2147483681 SCENARIO_EVENT 77 - 77 209.246.195.26 - 66.185.145.2
2147483731 SCENARIO_EVENT 94 - 94 66.185.134.7 - 66.185.140.2
2147483666 SCENARIO_EVENT 102 - 102 66.185.153.37 - 66.185.148.2
2147483673 SCENARIO_EVENT 105 - 105 66.185.130.32 - 66.185.139.2
2147483724 SCENARIO_EVENT 106 - 106 66.185.148.252 - 66.185.140.2
2147483710 SCENARIO_EVENT 113 - 113 209.244.107.45 - 66.185.144.2
2147483722 SCENARIO_EVENT 114 - 114 66.185.153.204 - 66.185.144.3
2147483726 SCENARIO_EVENT 114 - 114 66.185.158.222 - 66.185.144.3
2147483671 SCENARIO_EVENT 144 - 144 209.247.240.179 - 66.185.144.3
2147483680 SCENARIO_EVENT 145 - 145 209.244.107.108 - 66.185.148.2
2147483685 SCENARIO_EVENT 186 - 186 66.185.147.42 - 66.185.145.2
2147483728 SCENARIO_EVENT 196 - 196 66.185.151.137 - 66.185.140.4
2147483714 SCENARIO_EVENT 226 - 226 66.185.140.101 - 66.185.140.2
2147483711 SCENARIO_EVENT 239 - 239 66.185.140.28 - 66.185.148.2
2147483653 SCENARIO_EVENT 240 - 240 209.246.112.41 - 66.185.152.2
2147483739 SCENARIO_EVENT 262 - 262 209.247.122.240 - 66.185.148.2
2147483698 SCENARIO_EVENT 265 - 265 66.185.141.12 - 66.185.152.2
2147483744 SCENARIO_EVENT 266 - 266 209.245.61.92 - 66.185.152.3
2147483659 SCENARIO_EVENT 267 - 267 66.185.150.125 - 66.185.140.2
2147483741 SCENARIO_EVENT 268 - 268 66.185.139.213 - 66.185.139.2
2147483742 SCENARIO_EVENT 272 - 272 209.244.72.53 - 66.185.148.2
2147483647 SCENARIO_EVENT 285 - 285 66.185.158.36 - 66.185.144.2
2147483692 SCENARIO_EVENT 286 - 286 209.247.37.184 - 66.185.152.3
2147483729 SCENARIO_EVENT 289 - 289 209.247.214.74 - 66.185.140.3
2147483676 SCENARIO_EVENT 299 - 299 66.185.138.249 - 66.185.140.2
2147483733 SCENARIO_EVENT 305 - 305 209.246.180.168 - 66.185.144.2
2147483720 SCENARIO_EVENT 321 - 321 209.247.206.8 - 66.185.144.2
2147483734 SCENARIO_EVENT 338 - 338 209.246.70.169 - 66.185.148.2
2147483746 SCENARIO_EVENT 345 - 345 66.185.144.215 - 66.185.148.2
2147483672 SCENARIO_EVENT 347 - 347 66.185.158.62 - 66.185.145.2
```

Deployment Environment: SCIENCE_PORTAL Total ISPs: 2 Routers: 1052 Switches: 1048 Computers: 265134 VOs: 10

Figure 73 The Scenario Editor

Once the scenario is generated, it can be run in the Grid Protocol Simulator shown in Figure 74. The simulator loads the scenario, builds the network and event queues, starts the clock, runs the simulation, stops the simulation when the event queue is empty, and creates an Excel file output.



Grid Protocol Simulation - 100 Events

Scenario Events

```
2147483660 SCENARIO_EVENT 28 - 28 66.185.141.216 - 66.185.152.2
2147483657 SCENARIO_EVENT 60 - 60 66.185.158.53 - 66.185.152.3
2147483655 SCENARIO_EVENT 60 - 60 209.247.81.218 - 66.185.145.2
2147483736 SCENARIO_EVENT 62 - 62 66.185.158.111 - 66.185.152.2
2147483717 SCENARIO_EVENT 65 - 65 209.245.234.17 - 66.185.152.2
2147483681 SCENARIO_EVENT 77 - 77 209.246.195.26 - 66.185.145.2
2147483731 SCENARIO_EVENT 94 - 94 66.185.134.7 - 66.185.140.2
2147483666 SCENARIO_EVENT 102 - 102 66.185.153.37 - 66.185.148.2
2147483673 SCENARIO_EVENT 105 - 105 66.185.130.32 - 66.185.139.2
2147483724 SCENARIO_EVENT 106 - 106 66.185.148.252 - 66.185.140.2
2147483710 SCENARIO_EVENT 113 - 113 209.244.107.45 - 66.185.144.2
2147483722 SCENARIO_EVENT 114 - 114 66.185.153.204 - 66.185.144.3
2147483726 SCENARIO_EVENT 114 - 114 66.185.158.222 - 66.185.144.3
2147483671 SCENARIO_EVENT 144 - 144 209.247.240.179 - 66.185.144.3
2147483680 SCENARIO_EVENT 145 - 145 209.244.107.108 - 66.185.148.2
2147483685 SCENARIO_EVENT 186 - 186 66.185.147.42 - 66.185.145.2
2147483728 SCENARIO_EVENT 196 - 196 66.185.151.137 - 66.185.140.4
2147483714 SCENARIO_EVENT 226 - 226 66.185.140.101 - 66.185.140.2
2147483711 SCENARIO_EVENT 239 - 239 66.185.140.28 - 66.185.148.2
```

SCIENCE_PORTAL Events

Scenario Events	13	Task Event	7	Goodbye Event	0
Signup Events	13	Task Complete Event	0	Unsubscribe Event	0
Accept Event	13	Confirm Delivery Event	0	Blacklist Event	0
Advertise Event	13	Confirm Transaction Event	0	Unauthorized Event	0

Error Management

Task Unsatisfied Event 0

Memory Usage (Bytes)

Total Signup Table	750	Total Resource Table	115
Max Signup Table	100	Max Resource Table	25
Min Signup Table	25	Min Resource Table	9
Peak Signup Usage	100	Peak Resource Usage	25
Total Blacklist Table	0	Blacklist Max Table	0
Peak Blacklist Usage	15	Blacklist Min Table	0

State: Running ISPs: 2 Routers: 1052 Switches: 1048 Computers: 265134 VOs: 10 Events: 100 Sim Time: 138

Figure 74 The Grid Protocol Simulator

Software Design and Implementation

The simulation suite is implemented in C++/CLR (Common Language Runtime) which uses new features which are part of the CLS (Common Language Specification) [77]. The major driving factor to use C++/CLR is the ability to use the latest .NET forms and controls (a. k. a. widgets) and to interface directly with Microsoft Excel to create spreadsheets through the software using Visual Studio Tools for Office (VSTO). The CLR allows a common execution environment for Microsoft platforms (Windows XP, Windows CE, etc.) Microsoft is in the process of making the CLI (Common Language Interface) an IEEE standard.

As the dissertation work progressed from simulator to simulator, there was a drive to reuse common components from the previous simulator software when building the next. C++/CLR is similar to C++, but there have been major changes [81]. While some of the software was reused from the CLL and wormhole routing simulators, there was a minor conversion effort to make the software classes work with the new language.

C++/CLR

There are three main distinguishing features between C++ and C++/CLR that are relevant to this work: garbage collection, pointers, and new keywords.

Garbage Collection

Garbage collection allows for an automated way for developers to write code without having to worry about the details of memory management and cleanup [75]. When the developer allocates a block of memory, it is registered with the garbage collector. The allocation of the memory returns a handle to the memory in a managed heap of memory. If the handle is copied, the garbage collector keeps track of the copies. If all copies of the handle fall out of scope in the software or are marked with the *nullptr* keyword, then the memory is ready for deletion from the heap.

Memory is usually not automatically deleted from the heap when it needs to be. Memory allocations and de-allocations are typically temporally expensive operations with unmanaged memory and they are faster when they are grouped together in one large block with managed memory. When managed memory should be deleted, it is assigned to an older generation of memory. When a generation of memory reaches a particular size, it may be deleted or given an older generation. As the generations get older, they are deleted when resources are running low or when the application is closed.

The CLR garbage collector has two heaps: a managed heap and an unmanaged heap. The unmanaged heap contains the memory used for regular C++ data types which are allocated to dynamic memory. The managed heap contains the memory which is allocated from the new C++/CLR managed objects. The drawback for using both heaps is that memory is typically duplicated between heaps. The duplication not only wastes memory, but there is additional overhead to copy, delete, and track both heaps.

To address this issue, the simulator was compiled in a managed mode which means that the regular C++ keywords and operators no longer work and have been replaced by the new C++/CLR keywords and operators. One tradeoff of doing this is regular C++ variables are now *boxed* [76] meaning that they have been wrapped inside of a C++ managed class (which adds a small amount of extra memory consumption and processing time).

Using regular C++, dynamic memory is manually de-allocated using the *delete* keyword. The *delete* keyword still exists in managed C++, but the use of it is different. In regular C++, if you call the *delete* operator on dynamically allocated memory, the destructor is called for the class and the memory is de-allocated. In managed C++, if you call the *delete* operator, the destructor is called but the memory is not de-allocated. As mentioned before, the timing of the memory release is up to the garbage collector. The garbage collection method can be called, but it is not guaranteed to collect all freed memory.

In addition to implementing a destructor, the option exists to implement a *Finalize* method [75]. The *Finalize* method is called from a special thread right before the memory is de-allocated for that class. The developer cannot call the *Finalize* method manually except when a child class calls a parent class' *Finalize* method.

This poses an interesting dilemma. Sometimes, a developer may implement a destructor to close a network or socket connection, file handle, or database connection. Since the timing of the de-allocation is non-deterministic, the destructor may not be called at a logical time. This results in open connections that probably should be closed or a deadlock situation. Also, there is a possibility that a destructor can be called more than once, so the closure of the connection must be guarded to prevent an exception from being thrown or some other error condition. It may be a better option to implement a *Finalize* method if the timing of the closure does not matter.

C++/CLR Pointers

Another new feature for using the CLR garbage collection is the way that allocation and de-allocation strategies and procedures of memory occur. When allocating managed classes with the managed mode compiler option, the regular C++ pointer (*) does not work and has been replaced with the hat operator (^). Also, the C++ *new* operator has been replaced with the *gcnew* operator. The "gc" indicates and reminds the developer that memory is being managed by the garbage collector.

For example, `int *x = new int(3);` now becomes `System::Integer ^x = gcnew System::Integer(3);` with the new language. The new integer class is a boxed implementation of the old integer data type. The hat handle operator replaces the star pointer operator, and the *gcnew* operator replaces the *new* operator. The star operator is still used to deference a handle. However, the C++ reference operator (&) has been replaced with the C++/CLR handle reference operator (%).

C++/CLR Keywords

In order to allocate a managed class, a class is marked as a managed class by using the new *ref* keyword. For example, a developer would use `public ref class A` in managed C++ rather than `class A` in C++ when defining a class. There are several other new keywords that impact the implementation of the simulator such as: *sealed*, *for each*, and *abstract*.

The *sealed* keyword allows a developer to seal a base class or base class method from being over-ridden or overloaded in a child class. The *for each* operator allows a developer to iterate through a Collection (which implements the IEnumerable interface) with fewer lines of code [84]. The *abstract* keyword allows a developer to mark a parent class as non-instantiatable class meaning that a class must inhert the class if the developer wants to declare an object of that type.

Visual Studio Forms and Controls

Visual Studio provides a simplified way to create GUIs by allowing the developer to drag-and-drop graphical objects into windows [78]. The windows and containers are referred to as forms and the graphical objects the user interactions with are known as controls. The .NET library contains a large library of controls including drop-down combo boxes, spinners (or up-down numeric counters), text boxes, and check boxes.

The Visual Studio 2005 Professional Edition allows a developer to use a more "modernized" approach for working with Windows controls than previous versions of Visual Studio (like version 6.0 used for the wormhole routing simulator) [82]. The improvements have to do with the way that many of the detailed handling of Windows events has been encapsulated inside of the forms classes. Also, the technique for declaring event handlers using delegates simplifies the way to receive callbacks when significant Windows events (like pressing a key or moving a mouse) occur.

Another improvement is the way that background threads can be spawned using the *BackgroundWorker* class [83]. The *BackgroundWorker* was used several times in the Grid Protocol Simulator to allow the GUI to function while performing lengthy tasks. Examples of this are loading or saving a scenario file while showing the progress indicator window and running the simulator while displaying the simulation GUI and updating the simulation statistics on the fly.

Visual Studio Tools for Office

One of the main motivations of using Visual Studio is the ability to create spreadsheets using the Excel API provided by Visual Studio Tools for Office (VSTO) [85]. VSTO adds support for Word, Excel, Outlook, and Infopath and the 2005 version of Visual Studio integrates the support into .NET. It allows developers to use the Office System to display, format, chart, calculate and analyze data in Excel. For instance, simulation data is recorded in an Excel workbook with several worksheets that include a simulation summary, and VO, event, and memory statistics.

```
Excel::Application
    ^app = gcnew Excel::ApplicationClass();
Excel::Workbook
    ^wb = app->Workbooks->Add(Type::Missing);
Excel::Worksheet
      ^ws = safe_cast<Excel::Worksheet ^>(wb->ActiveSheet);
```

Figure 75 Basic Steps to Create an Excel Workbook and Worksheet Using VSTO

The basic steps to create an Excel workbook and worksheet are shown in Figure 75. Creating the Excel application will spawn an Excel process. Note that calling the quit method can kill the application. If the developer's program crashes the process may have to be manually killed using the *Task Manager*. Once the Excel application is started, a workbook is added. By default, the workbook has three worksheets. The first worksheet is active by default and can be accessed by the *ActiveSheet* data member.

```
ws->Name = "Simulation Summary";
ws->Range["C1", Type::Missing]->Value = "Simulation Summary";
ws->Range["C1", Type::Missing]->Font->Bold = true;
ws->Range["D1", Type::Missing]->Value = scenario_name;
```

Figure 76 Basic Worksheet Operations Using VSTO

```
Excel::ChartObjects
    ^chart_objects = safe_cast<Excel::ChartObjects ^>(
        ws->ChartObjects(Type::Missing));
Excel::ChartObject
    ^chart_object = chart_objects->Add(300, 0, 1200, 300);
Excel::Chart
    ^chart = chart_object->Chart;

chart->ChartWizard(
    ws->Range["B3:B" + row.ToString() +
        ",C3:C" + row.ToString(),Type::Missing],
    Excel::XlChartType::xl3DColumn,
    Type::Missing,
    Excel::XlRowCol::xlColumns,
    1, 1, false,
    "Number of Resource per VO",
    "VO IP",
    "Number of Resources",
    Type::Missing);
```

Figure 77 Creating a Chart in Excel Using VSTO

In order to populate the worksheet, the developer specifies the range of cells to edit. In the example in Figure 76, the first cell C1 is updated to show the text, “Simulation Summary,” then on the next line of code the text is marked as bold. There are many features available to the program such as writing formulas, auto-fitting the cells around the text, and sorting data.

Another useful method allows the developer to chart data. Figure 77 shows an example for creating a chart by instantiating a *ChartObject* in the worksheet. The chart is moved to a specific location in the worksheet, then it is populated with data. In this example, the data used for this 3D bar chart comes from columns B and C.

Software Design

There were several major design decisions made when implementing the simulator. The first design topic introduced has to do with the layout of the network for the scenario generator and the simulator. Both applications represent the network the same way, but the differences lie in the way they are used.

Originally it was conceived that the network tree would be displayed in the simulator and scenario generator. When the tree is displayed in the scenario generator, it allows the user to add, remove, or modify network devices in the tree to configure the network for scenario generation. Showing the network tree in the simulator would have allowed the user to visually see the network traffic traveling through the network in real time as the simulator was running.

It turns out that the *TreeView* form does not appropriately handle the large network trees required for a grid network. The Microsoft online documentation [86] recommends not exceeding 32,767 *TreeNode*s in the tree because the tree structure may lose references to nodes at that point. Also, the tree structure uses a very large amount of memory and the expanding, inserting, and removing of nodes in the tree becomes extremely slow when the tree is large. Another issue is that the tree uses a hash map to find nodes. This means that it is possible to lose nodes in the tree if a duplicate hash value is generated. Fortunately, since the hash values are four bytes and the IP addresses used are four bytes, the IP address was used as the hash value that prevents duplication.

Since the network tree was necessary for the scenario generator, it has been optimized to aggregate resource provider devices in one switch if they belong to that subnet of IP addresses. However, the network tree was not used in the simulation GUI because updating the tree was too slow and provided minimal value to the user when comparing the performance tradeoff to the graphical depiction. This resulted in divergent and repetitive implementations. The scenario editor version of the network tree inherits from *TreeView* while the simulation version of the network tree does not inherit from a Windows Form or Control.

Another major design decision involves how the messages are delivered through the network. The original grid protocol spec declares four routing methodologies: STANDARD, FORWARD_PATH, REVERSE_PATH, and DISCOVERY. Because of the way events are managed on the event queue, all of the scenario events sit on the queue when the simulator starts. So, if a node is supposed to receive an event at a particular time, the only way to route the message was through one of those four techniques (of which only STANDARD routing would apply). The downside of using STANDARD routing is that the event is delayed one simulation second each time it would travel from the network tree root to the destination node.

Looking up the destination node not only circumvents the routing system, but also incurs a delay of looking up the destination node in the network tree. So, since the destination has to be looked up no matter what, a new routing technique called DIRECT was created to allow a message to travel from the network tree root node to the destination node outside of simulation time. This routing technique is a by-product of the simulator implementation and is not included in the grid protocol spec. Also, it is not "cheating" because these messages are supposed to occur at the appropriate time and there is no other mechanism for doing that in the simulator and because the events that use the DIRECT routing technique are logged and graphed in the simulation output files.

Scenario Design

The scenarios used to run in the grid protocol simulator are based on the five deployment environments. Each deployment environment has a suite of scenarios with the same basic layout; so there are five scenario suites. Each suite has five scenarios with the same network topology but varying amounts of traffic. The scenarios vary based on the number of messages: 25, 250, 2500, 10,000, and 25,000. The basis of this design is to see whether or not the routers can hold enough information in their routing tables and to see how many discovery messages are successful when comparing the deployment environments.

The scenario files themselves are XML text files which can be displayed using any XML text reader. The Scenario Editor automatically generates the files based on the user's depiction of the network and events. There are two major XML blocks: the network and the events. The network has a name (usually "Internet"), and devices that fall under it. The network can have Internet service providers (ISPs). Under the ISPs, the user can place routers, switches, and VO hosts. By default, the user will have two ISPs pre-constructed with 1,052 routers and 1,048 switches representing 265,144 computers. Each computer is capable of sending one event in a scenario, this means there is a maximum of 265,144 events that can be created.

The event block contains all of the scenario events. These events include information about what resource would like to signup to a grid network and what VO host the signup will go to. The resource score is determined runtime, thus it is not in the scenario file. The score is generated randomly to allow different results to be achieved with the same scenario run multiple times. An advantage of doing this prevents from having to write many scenarios. A disadvantage is that it could be hard to reproduce errors or special conditions.

The scenario network topology construction is laid out in Table 27. The networks are intergrids [79] meaning that VOs do not communicate with each other. The Scenario Editor randomly generates scenario topologies. A basic Internet topology is provided with two ISPs, 1,052 routers (2 deep), and 1,048 switches representing 265,144 computers. From that, the user can extend the depth of the network; the scenarios tested have an extended depth of 5. This means that the total depth of the network will not exceed 6 routers deep for the first ISP or 7 routers deep for the second ISP giving a diameter of 13 possible router hops a message can travel.

Notice there are four less switches than routers. This has to do with the way that the basic Internet topology is represented. Each router has a switch except for four high-level routers that host the maximum number of routers they can support. Also, even though the same setup parameters are specified, this does not mean that each scenario will have the same number of devices. The algorithm in Figure 78 starts at the root network node and creates the basic Internet. The algorithm goes to the first leaf node. Then, if the network depth is not exceeded, the algorithm draws a random number between zero and one. If the number is less than or equal to 0.5, then the algorithm creates a router and a switch. It repeats this process for every branch until the entire tree is traversed. This results in branches with varying depths.

Table 27

Scenario Network Topologies

Deployment Environment	Routers	Switches	Computers
Science Portal	199781	199777	662162
Distributed Computing	388258	388254	1040539
Computer-in-the-Loop Instrumentation	446260	446256	1155893
Large-Scale Data Analysis	890437	890433	2044676
Collaborative Work	371683	371679	1007067

```
// Generate network tree
//
Generate basic network tree from flat file
Start at first leaf node
Loop until tree traversed
        If network depth is not exceeded
                If randomly extend tree
                        Create router and switch
        Advance to the next leaf node
End loop
```

Figure 78 Network Tree Generation Algorithm

Simulated Virtual Organization Scoring

This section explains the methodology used for scoring used in the simulator. Each virtual organization uses the same scoring policy in the simulator. The scores assigned to resource providers in the simulation cannot be discrete random variables between 0-255. In other words, the simulator cannot simply pick an arbitrary number between 0-255. This would yield computer configurations that most likely are not implemented in the real world. One example is a very fast multi-CPU machine with 64 MB of memory and a 320 MB hard drive. Likewise, older machines typically cannot support large amounts of RAM or disk storage.

The grid resource discovery protocol allows for the VO to define a 32-bit score variable and a VO product id. The score must have 8-bit chunks for CPU, memory, hard drive, and bandwidth scores (in that order). The VO product id can correspond to any numbering scheme the VO wants to use. For this simulator's virtual organizations, the 4-bit product id is divided into a 2-bit CPU type and a 2-bit OS type. There are four CPU types {PC_486, PC_586, APPLE_G4, SUN_SPARC} and four OS types {WINDOWS, LINUX, OS_X, SOLARIS}. All of this information is stored in the ResourceSpecs class.

Table 28

Possible Scoring Combinations Based on CPU Type

CPU Type	OS Type	CPU Speed	Memory Size	HD Size	Bandwidth
PC_486	WINDOWS LINUX	400-800 MHz	256-512 MB	10-200 GB	MODEM CABLE
PC_586	WINDOWS LINUX	.8-4 GHz	256-4096 MB	10-2000 GB	MODEM CABLE DSL T1 T3
APPLE_G4	OS_X	1.6-3.2 GHz	256-2048	10-2000 GB	CABLE DSL T1 T3
SUN_SPARC	SOLARIS	400-800 MHz	256-8192 MB	10-2000 GB	T1 T3

Table 28 shows possible scoring combinations based on the CPU type. For example in this hypothetical VO scheme, a 486 PC computer can run Windows or Linux, must have a speed of at least 400 MHz, RAM of at last 256 MB (free), hard drive of at least 10 MB (free space), and must have at least a MODEM connection to the network. The ranges (like 400-800 MHz for CPU speed) are there to give the range of score values for the VO. If a CPU speed greater than 800 exists, the VO still assigns it a score as if it has an 800 MHz processor.

When generating a random score for a resource provider, a discrete random variable is found between 0-100. If the random variable is less than 2, the CPU type is set to PC_486, when between 2 and 80 it is set to PC_586, when between 81 and 98 it is set to APPLE_G4, and any number greater than 98 sets the CPU type to SUN_SPARC. Once the random value is drawn for the CPU type, the other score attributes are randomized based on the ranges in Table 29 through Table 32.

Table 29

Simulation PC_486 Scoring Table

	Score of 0	Score of 1	Score of 2	Score of 3
CPU	400-499 MHz	500-599	600-699	>= 700
Memory	<= 256	> 256	N/A	N/A
Hard Drive	< 50 Gig	50-99 Gig	100-149 Gig	>= 150 Gig
Bandwidth	MODEM	CABLE/DSL	T1	>= T3

Table 30

Simulation PC_586 Scoring Table

	Score of 0	Score of 1	Score of 2	Score of 3
CPU	< 1600 GHz	1600-2399	2400-3199	>= 3200
Memory	<= 1024	1025-2048	2049-3072	>= 3073
Hard Drive	< 500 Gig	500-999 Gig	1000-1499	>= 1500
Bandwidth	MODEM	CABLE/DSL	T1	>= T3

Table 31

Simulation APPLE_G4 Scoring Table

	Score of 0	Score of 1	Score of 2	Score of 3
CPU	< 2000 GHz	2000-2399	2400-2799	>= 2800
Memory	<= 512 MB	512-1024	1024-1536	> 1537

Hard Drive	< 500 Gig	500-999 Gig	1000-1499	>= 1500
Bandwidth	MODEM	CABLE/DSL	T1	>= T3

Table 32

Simulation SUN_SPARC Scoring Table

	Score of 0	Score of 1	Score of 2	Score of 3
CPU	500-599 MHz	600-699	700-799	>= 800
Memory	<= 256	> 256	N/A	N/A
Hard Drive	< 500 Gig	500-999 Gig	1000-1499	>= 1500
Bandwidth	MODEM	CABLE/DSL	T1	>= T3

Results

Results are presented for each of the deployment environments. The methodology for presenting the results mainly come from [104], [107], and [108], but some methods of reporting results for this work are new since the type of work is different than traditional grid resource discovery protocols. Some new results reported for this work are for signup, resource, and blacklist table usage as well as score deviations. [105] presents resource usage of a single resource. Resource usage of a single resource does not apply to this research because there are thousands of resources modeled; reporting one does not aid in presentation of results. On the other hand, [104] reports the amount of events dropped, average number of hops, and the distribution of events that are reported for this work.

The work in [107] identifies four attributes: resource discovery speed, system efficiency, load balancing, and discovery success rate. The resource discovery speed is not considered in this work as a significant result because the time to discover a resource is significantly less than the time to process a task. System efficiency, the balance between resource advertisement and discovery, is defined by the scenarios and simulation configuration and is a 1:1 relationship for all of the scenarios presented in this work. The user predefines load balancing when creating scenarios. In the case of this work, the load balancing is randomly distributed as is shown in the event distribution bar charts below. The discovery success rate is the opposite statistic of the amount of events dropped from [104] which is presented in the results below.

[108] is based on a discovery protocol for sensor networks. One unique result tracked is the amount of memory consumption in a sensor node based on the number of nodes in the network. This is another important result to track for this work because the memory consumption of entries in the routing tables must be implemented in hardware. The grid resource discovery protocol has three different routing tables that are populated and unpopulated at different times in the lifecycle of a message.

Science Portal

The science portal simulation results are presented in this section. The event distribution diagrammed in Figure 79 shows that the distribution of traffic between each of the VOs is roughly the same. The VO hosts and resource providers are distributed throughout the network and the average number of hops is around 14.45 as shown in Figure 80, and the discovery event hops are presented for each scenario in Figure 81 thru Figure 85. A hop is considered movement from one network device to another.

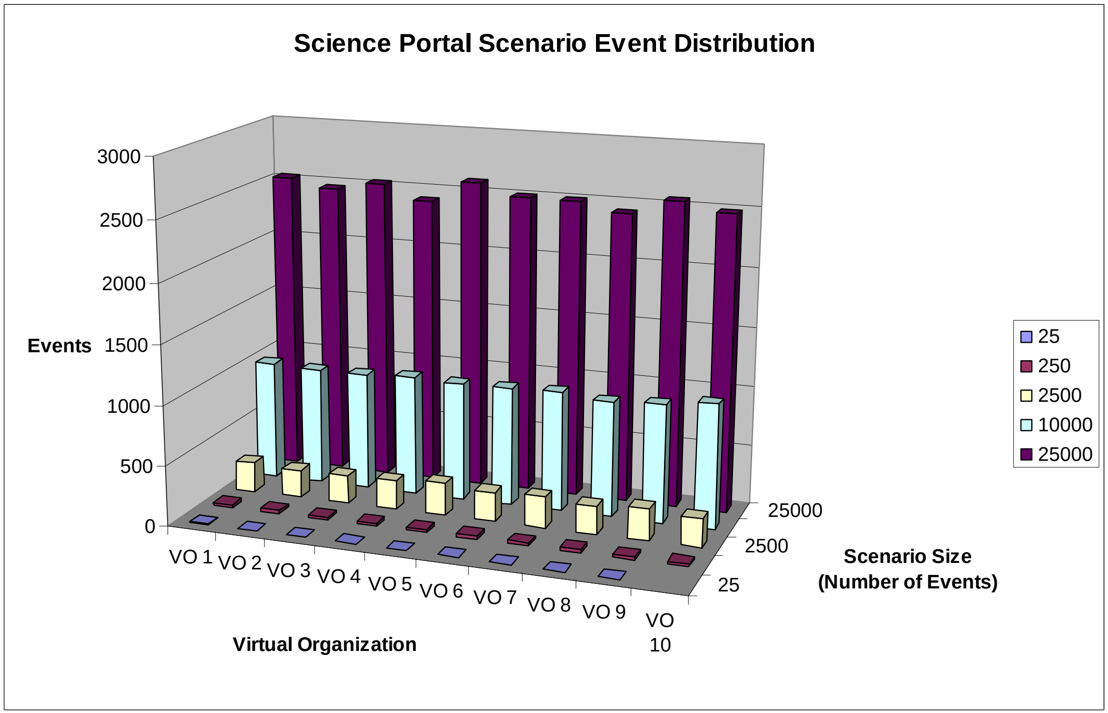


Figure 79 Science Portal Scenario Event Distribution

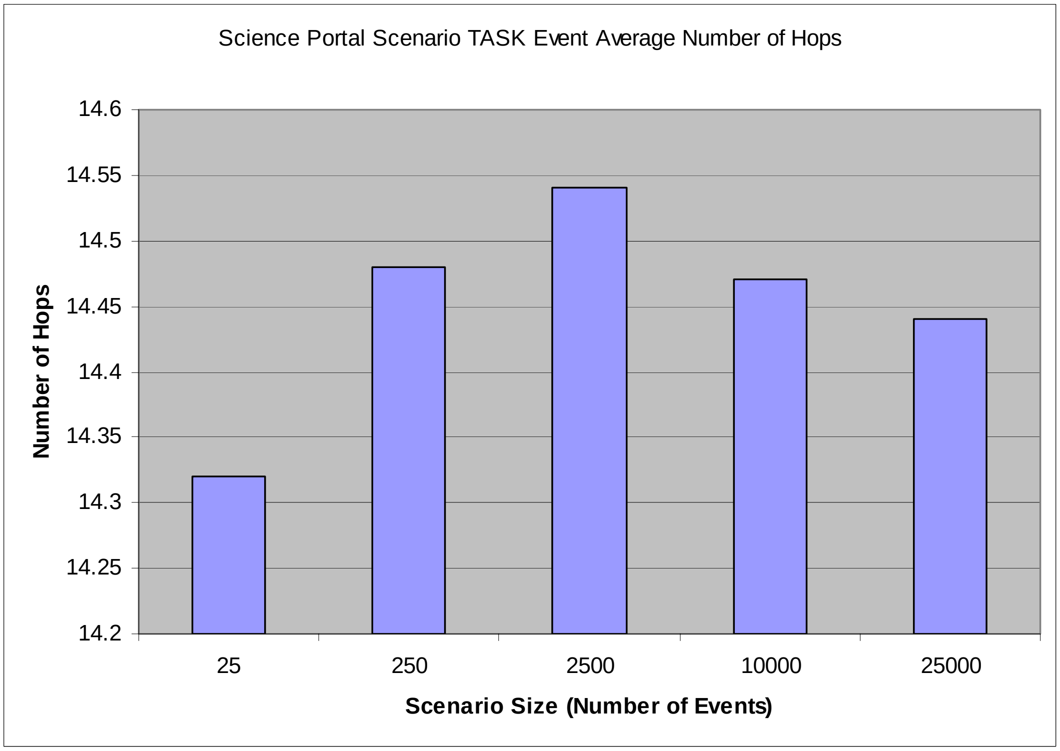


Figure 80 Science Portal Scenario Average Number of Hops

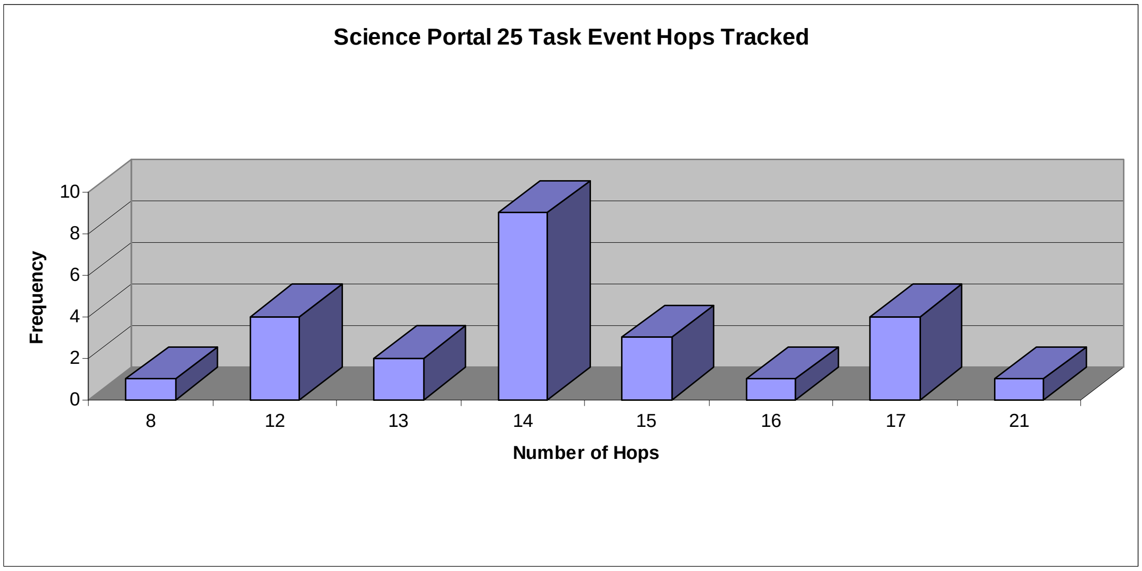


Figure 81 Science Portal Scenario Number of Hops for 25 Event Scenario

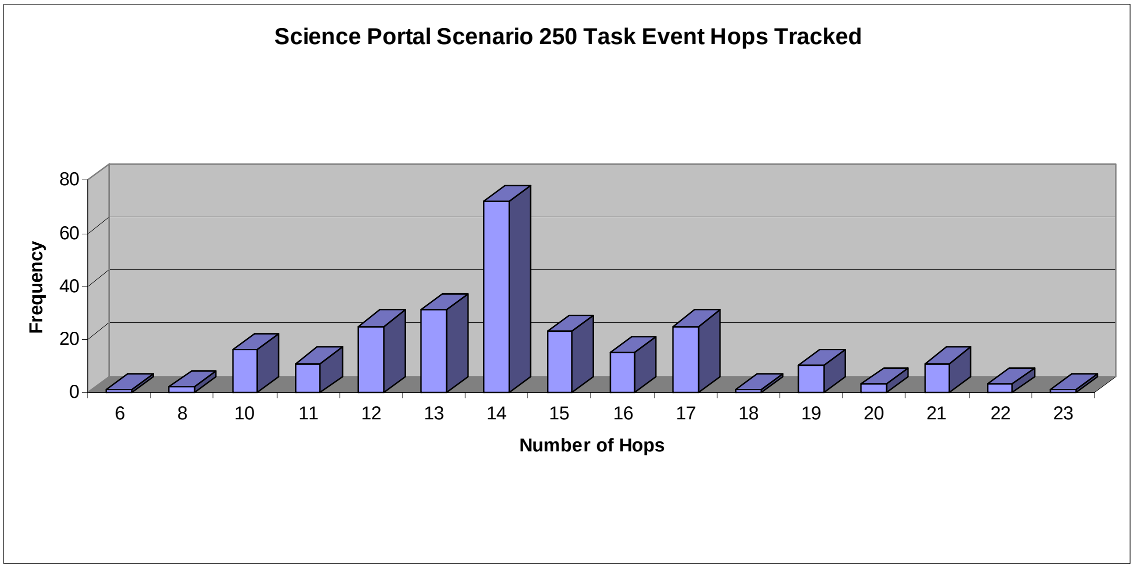


Figure 82 Science Portal Scenario Number of Hops for 250 Event Scenario

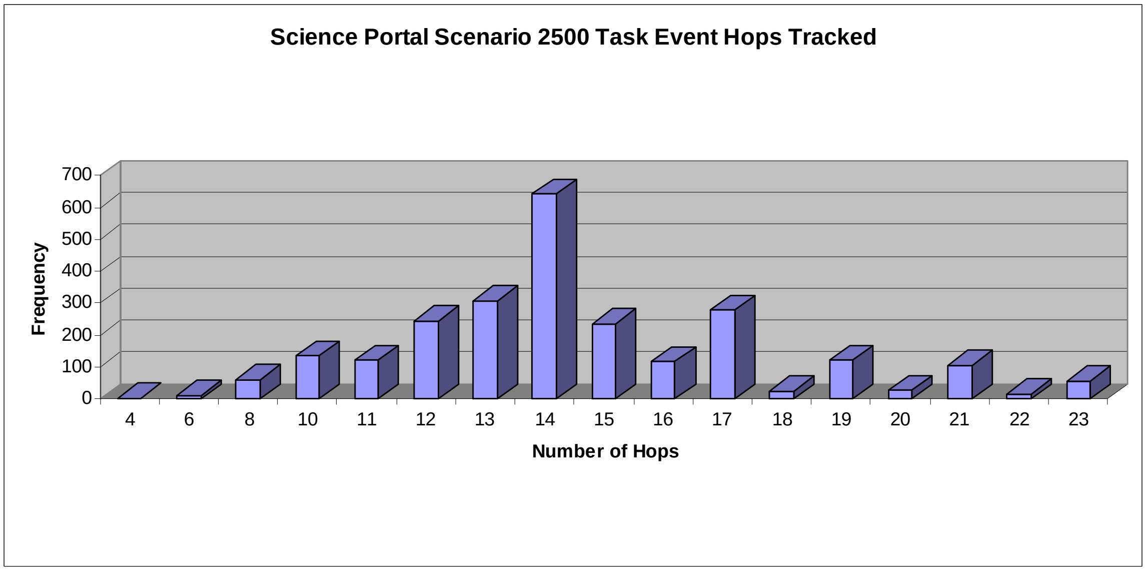


Figure 83 Science Portal Scenario Number of Hops for 2500 Event Scenario

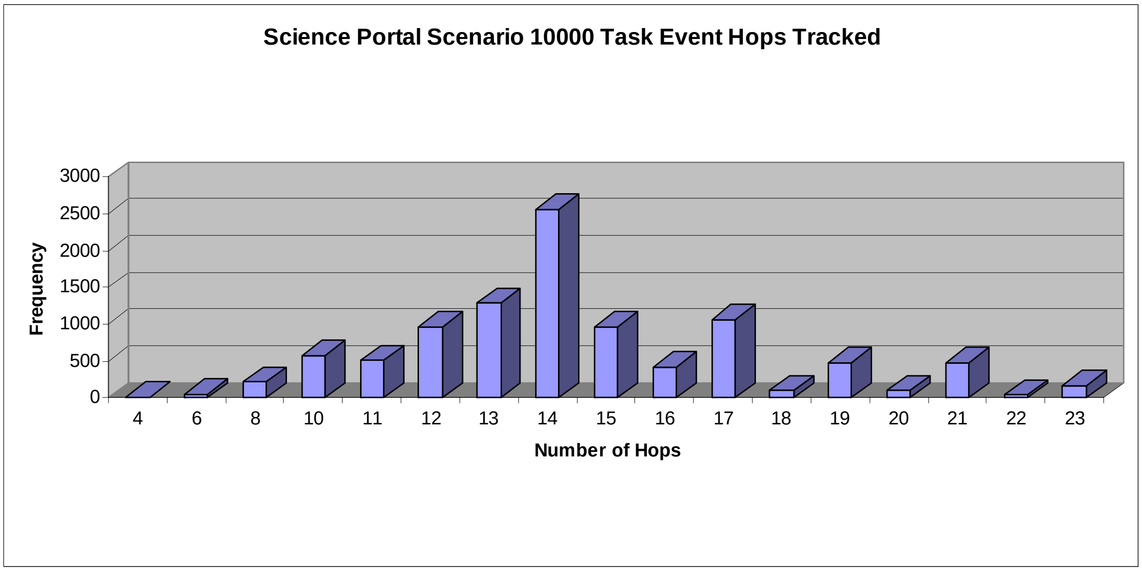


Figure 84 Science Portal Scenario Number of Hops for 10000 Event Scenario

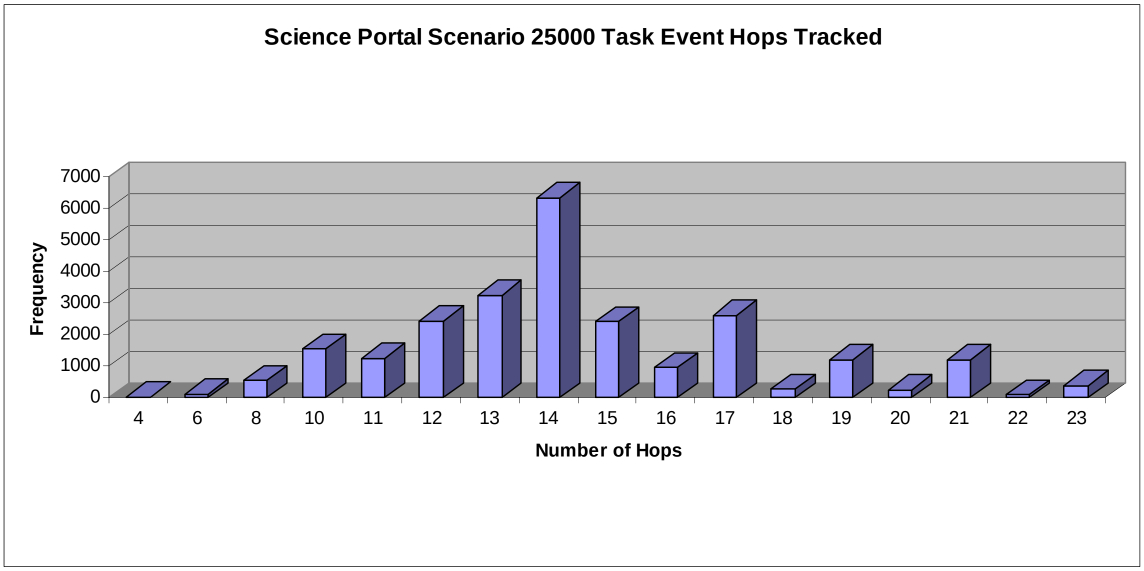


Figure 85 Science Portal Scenario Number of Hops for 25000 Event Scenario

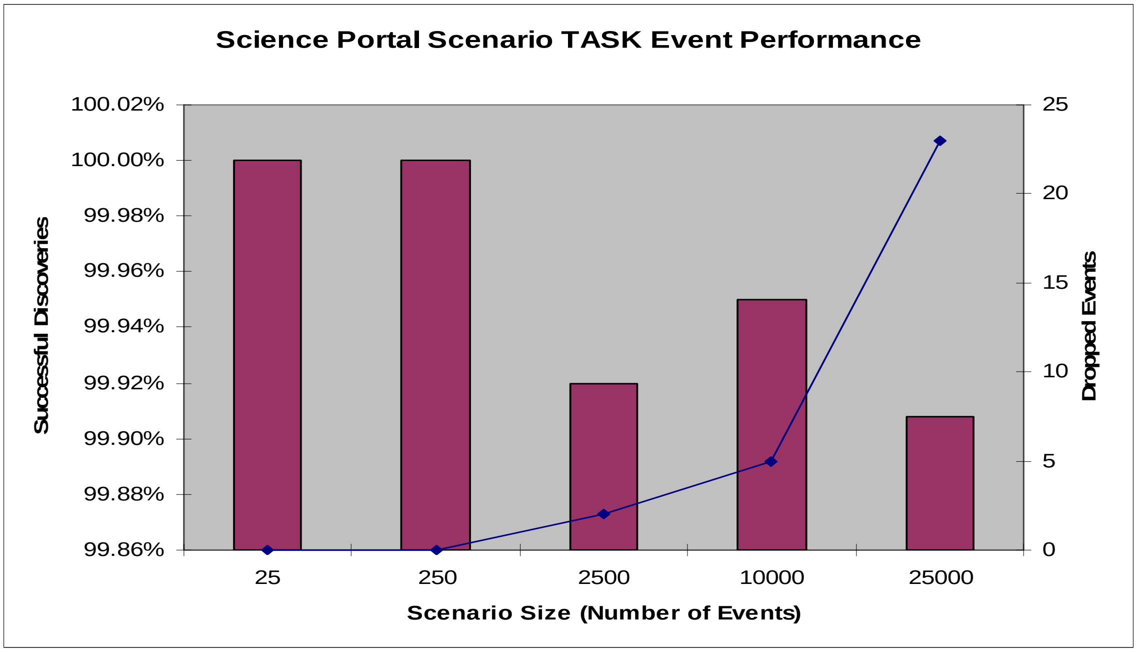


Figure 86 Science Portal Scenario Successful TASK Events

One important metric measures how successful the discovery approach was at finding a resource. Figure 86 shows the success rates of the TASK events finding an available resource. The values range between 99.91%-100% successful discoveries or 0-

23 dropped packets. In the case of the unsuccessful TASK event not finding a resource, in the real world the VO host would simply try until it finds a resource. But, the simulator does not model this for the purposes of finding the success rates.

One new statistic provided in this research has to do with tracking how scores deviate from a perfect score.   A perfect score does not deviate from the score of the resource, that score deviation value would be zero. With the case of science portal scnenario, the bandwidth field is a "don't care." This means that the 8 bit score composed of CPU, memory, hard drive, and bandwidth would have a mask of 0xFC. This yields scores in the ranges of {0-3} with a deviation of zero. Considering the bit positions, one would expect scores to deviate around 0, 1, 2, and 3 depending when the resource score has a 0 or 1 in the spot of the don't care.

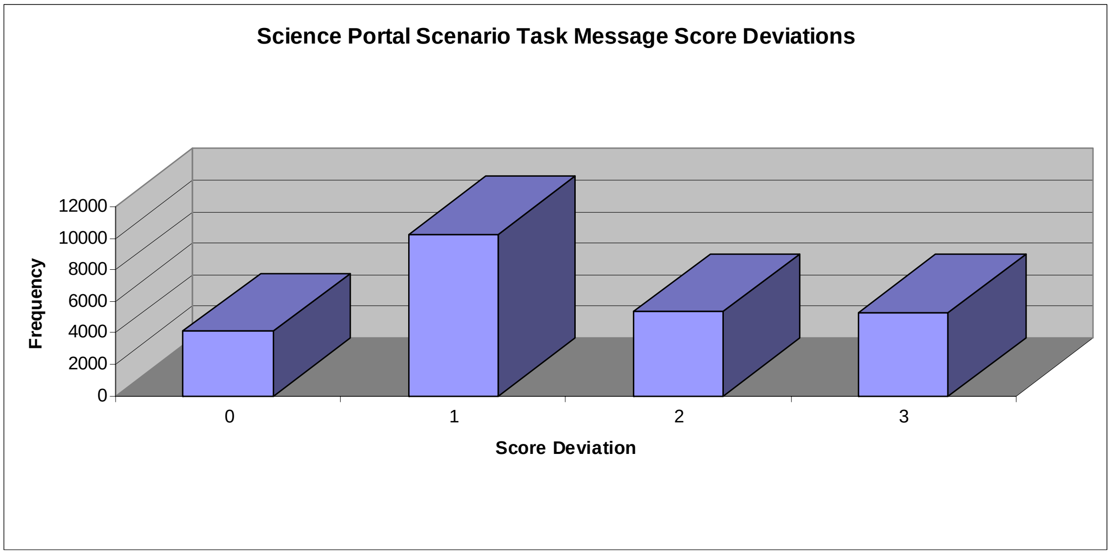


Figure 87 Science Portal Scenario Score Deviation for the 25,000 Event Scenario

Looking at Figure 87, the scores tend to deviate in that fashion. This figure represents the score deviations in the 25,000 event scenario. 100% of the scores fall exactly on 0, 1, 2, and 3. When numbers deviate from the desired score, they deviate by an average of 1.47.

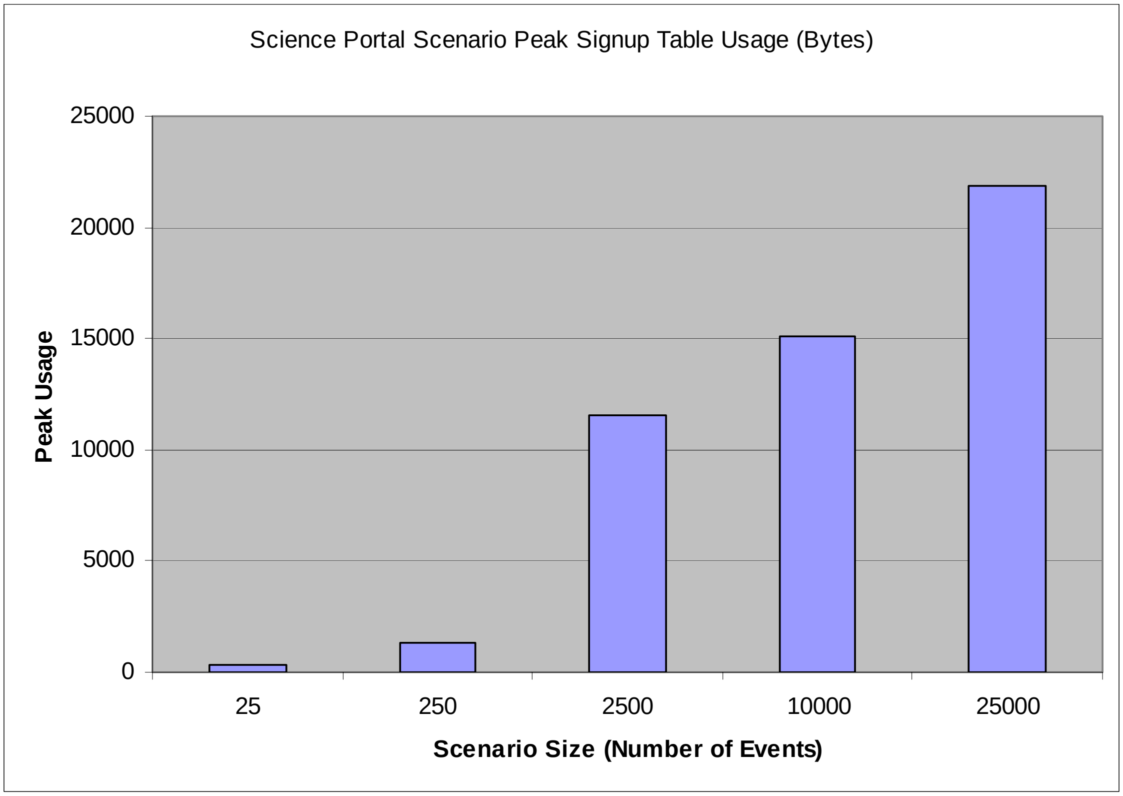


Figure 88 Science Portal Scenario Peak Signup Table Usage

The signup table peak memory usage is shown in Figure 88. As the number of events is increased, the memory usage caps at 21876 bytes. This happens because devices are un-subscribing from the network as time is advancing which reduces the size of the signup table usage. By default, the resource providers unsubscribe from the VO in 200 simulation seconds after the CONFIRM DELIVERY event is sent. The signup table worst-case peak usage can be estimated if UNSUBSCRIBE are not sent. Since there are 25,000 events (worst case) from 25,000 different resource providers with a 4 byte address, 10 VOs with each with a 4 byte VO Host IP address, and 10 bytes per SIGNUP TABLE ENTRY, then the worst case peak memory consumption for any particular router is 450 KB or 25KB * (4 + 4 + 10). As a reminder, signup entries are removed every 24 hours to prevent uncontrolled growth of these tables.

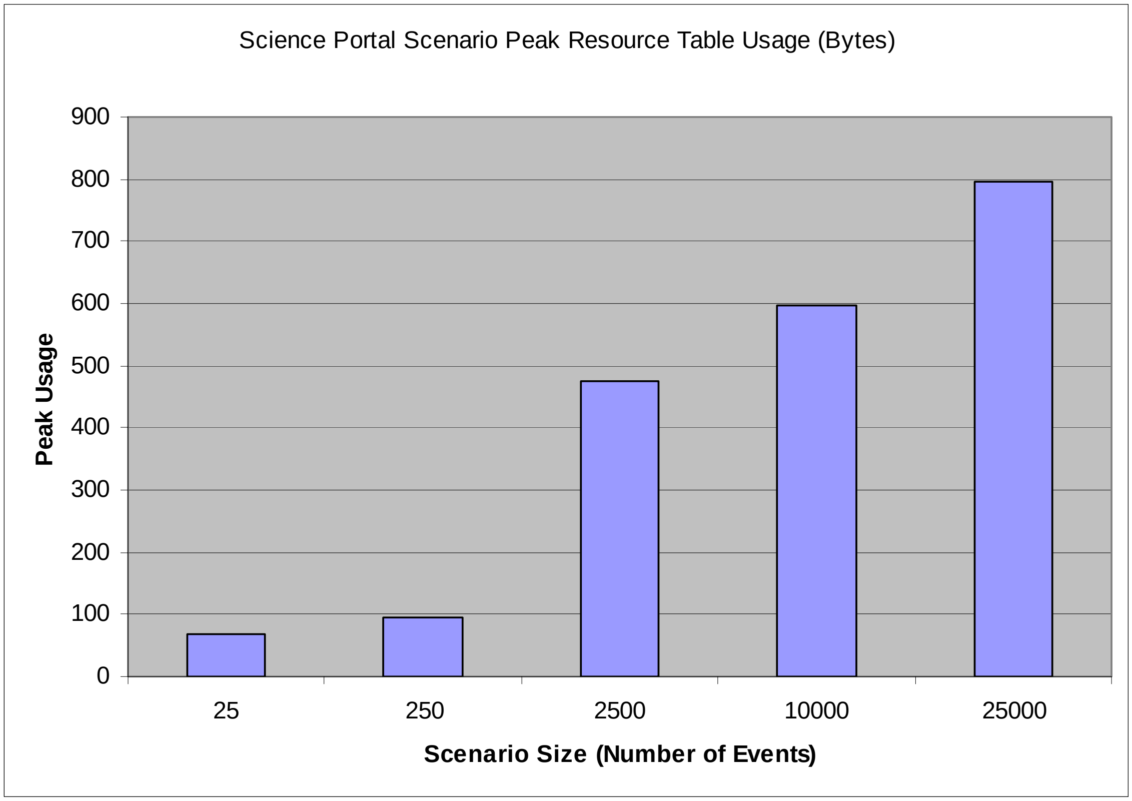


Figure 89 Science Portal Scenario Peak Resource Table Usage

Next, the peak resource table usage is examined in Figure 89. The usage caps at 796 bytes as the number of TASK events grow. This happens for different reasons than the signup table previously presented. The resource table has a smaller sized hash key and uses a one-byte score to lookup data. As the scenarios grow larger, once there are more than 256 resource providers, the scores will definitely overlap. The resource table is optimized to aggregate and count the number of devices with a particular score rather than to list individual resource providers. Also in this case, the simulation is greedy in discovering resources. Because resources are discovered in a greedy fashion, the table size does not grow very large because resources are consumed very quickly. Again the worst case tables size could be estimated by considering 25,000 resource IP addresses with 2 byte hash key, the score of one byte, the next hop IP of 4 bytes, and the count of one byte totaling 200KB or 25KB * (2 + 1 + 4 + 1).

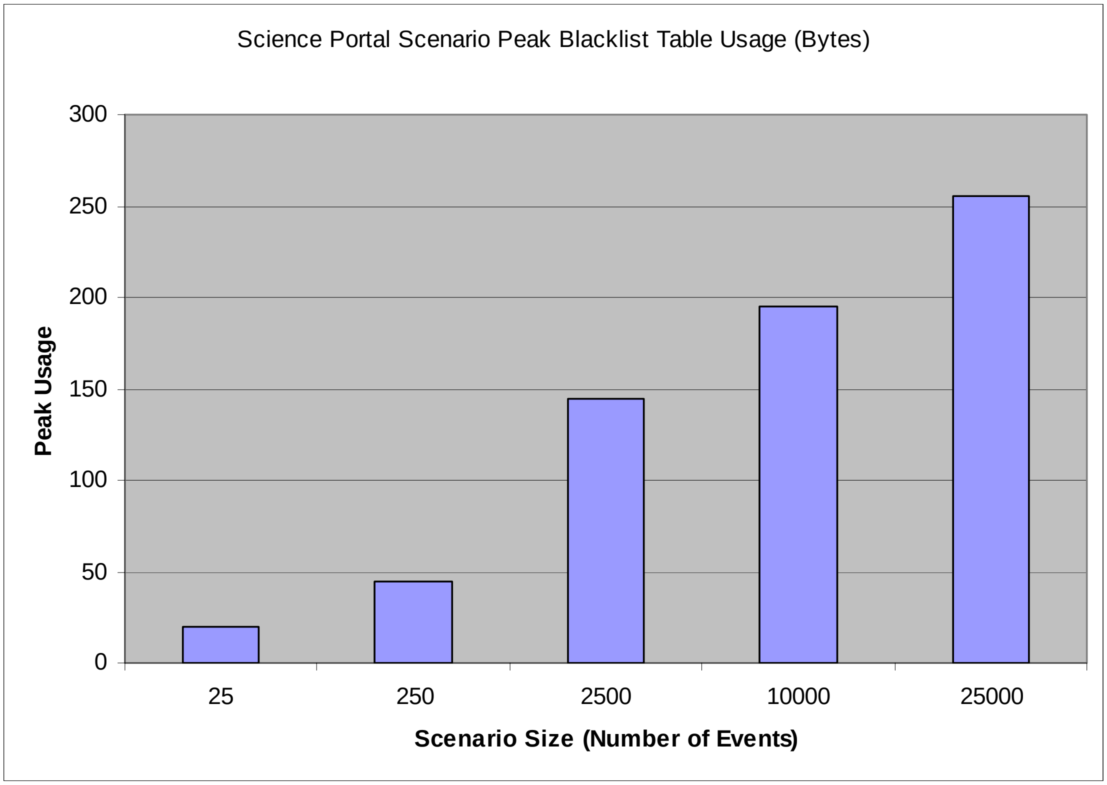


Figure 90 Science Portal Scenario Peak Blacklist Table Usage

The final results examined for the science portal scenarios are the blacklist tables. Since the blacklist tables are populated when the SIGNUP event is sent and unpopulated with the ACCEPT event is sent, the tables are much smaller than the others because the SIGNUP and ACCEPT events happen very close to each other in time. As shown in Figure 90, the memory usage caps at a value similarly to the other tables; this time around a value of 255 bytes. Doing the math, the BLACKLIST TABLE contains a four-byte IP address and a one-byte count totaling 5 bytes. Dividing 255 by 5 means that each router kept no more than 51 entries in its BLACKLIST TABLE at a given time. The worst case blacklist size can be calculated as well by multiplying 25,000 resource providers times 5 bytes totaling 125KB. This would imply that all 25,000 resource providers send their SIGNUP events at the same time through the same router.

Distributed Computing

The distributed computing scenario simulation results are presented in this section. The event distribution diagrammed in Figure 91 shows that the distribution of traffic between each of the VOs is roughly the same. The VO hosts and resource providers are distributed throughout the network and the average number of hops is around 15.09 as shown in Figure 92, and the discovery event hops are presented for each scenario in Figure 93 thru Figure 97. A hop is considered movement from one network device to another.

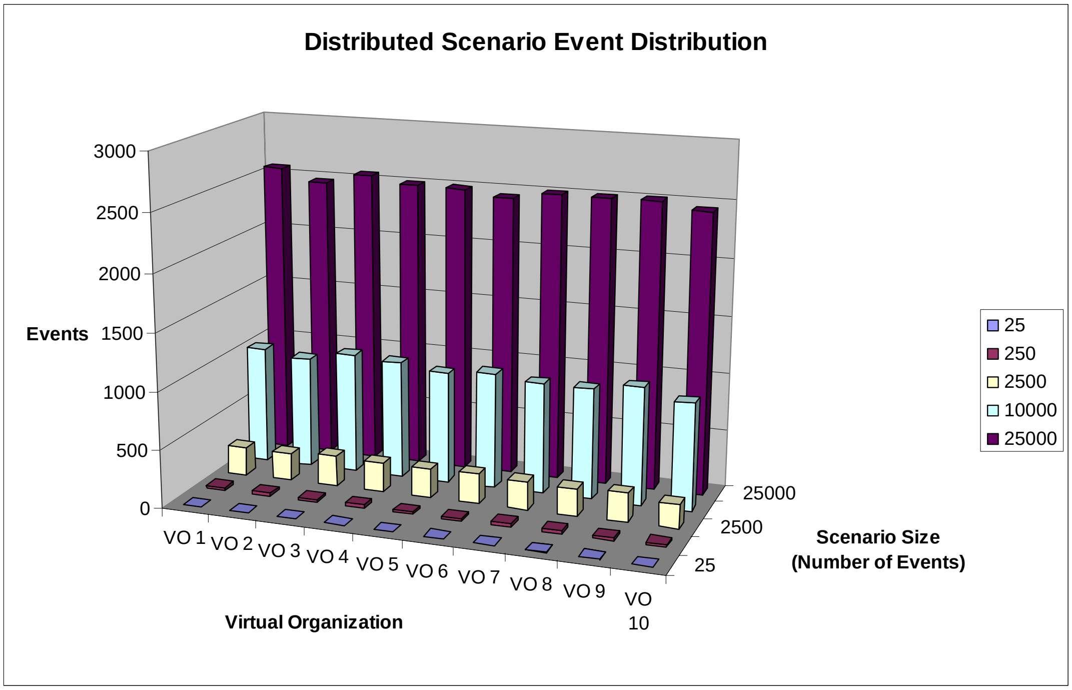


Figure 91 Distributed Computing Scenario Event Distribution

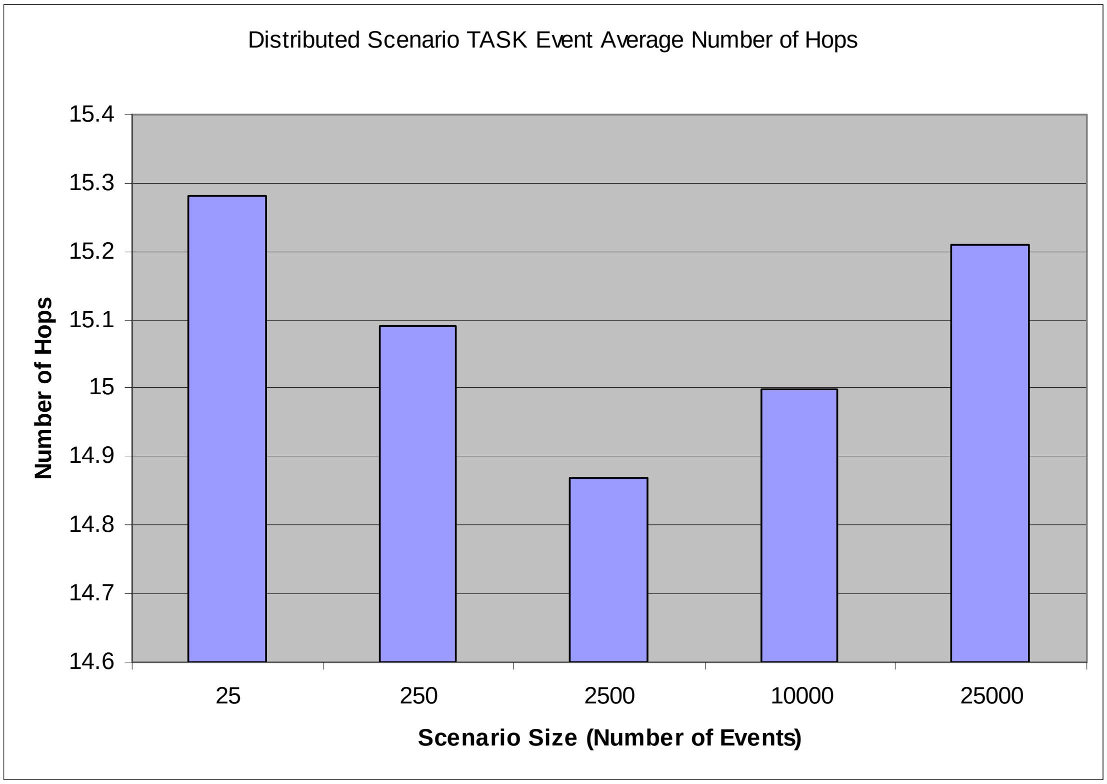


Figure 92 Distributed Computing Scenario Average Number of Hops

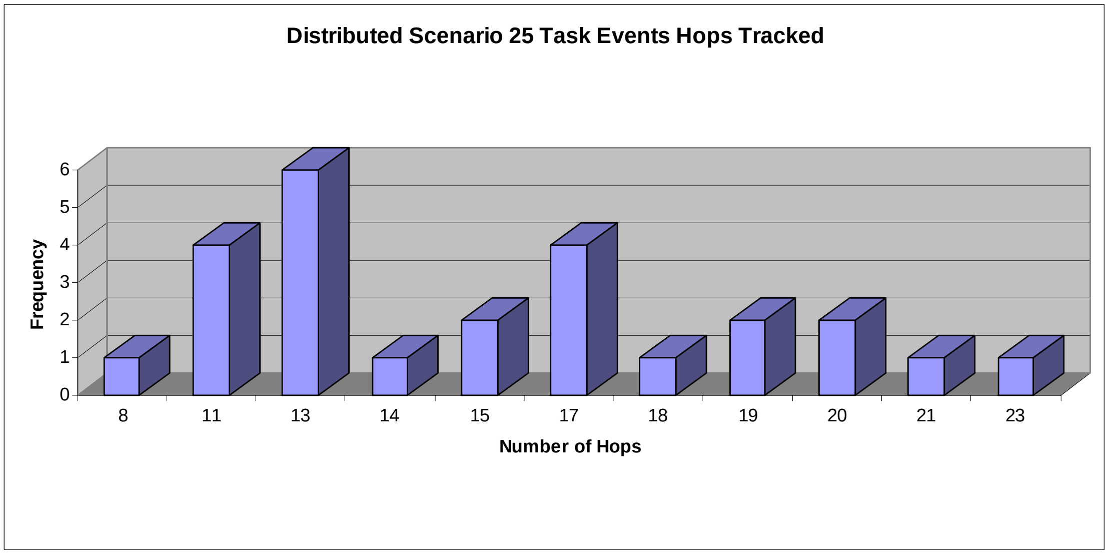


Figure 93 Distributed Computing 25 Event Scenario Number of Hops

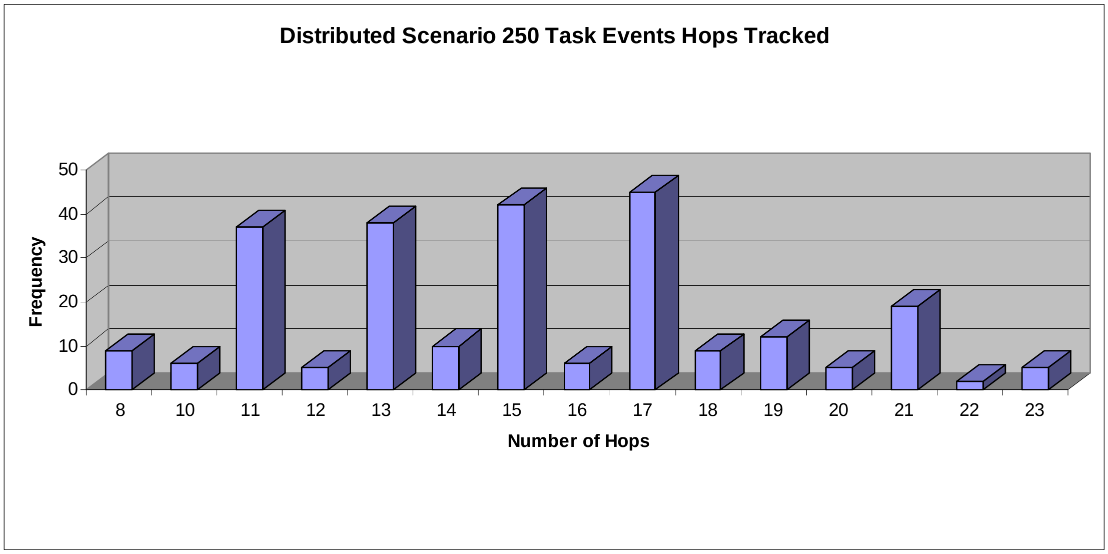


Figure 94 Distributed Computing 250 Event Scenario Number of Hops

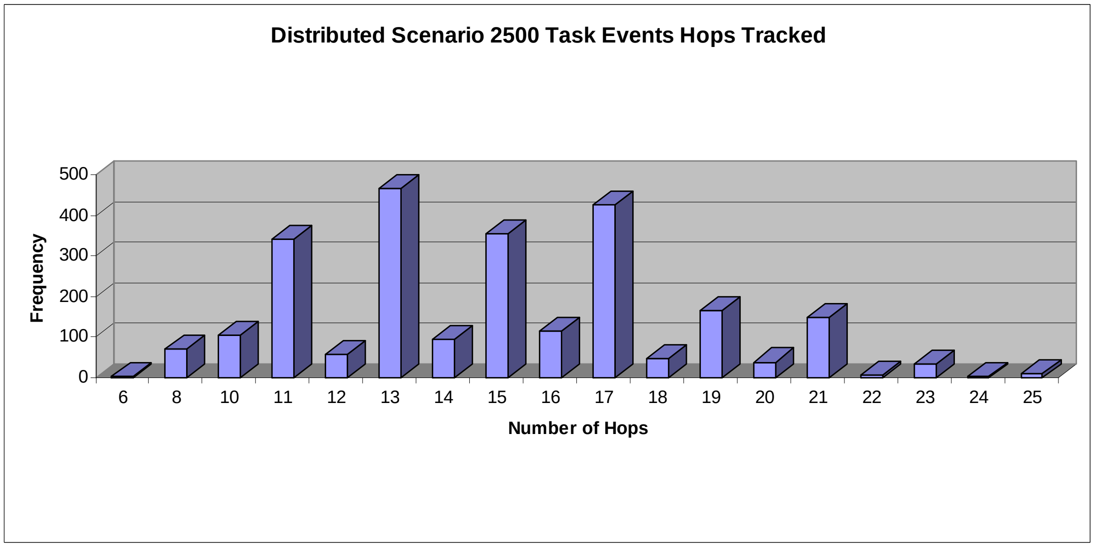


Figure 95 Distributed Computing 2500 Event Scenario Number of Hops

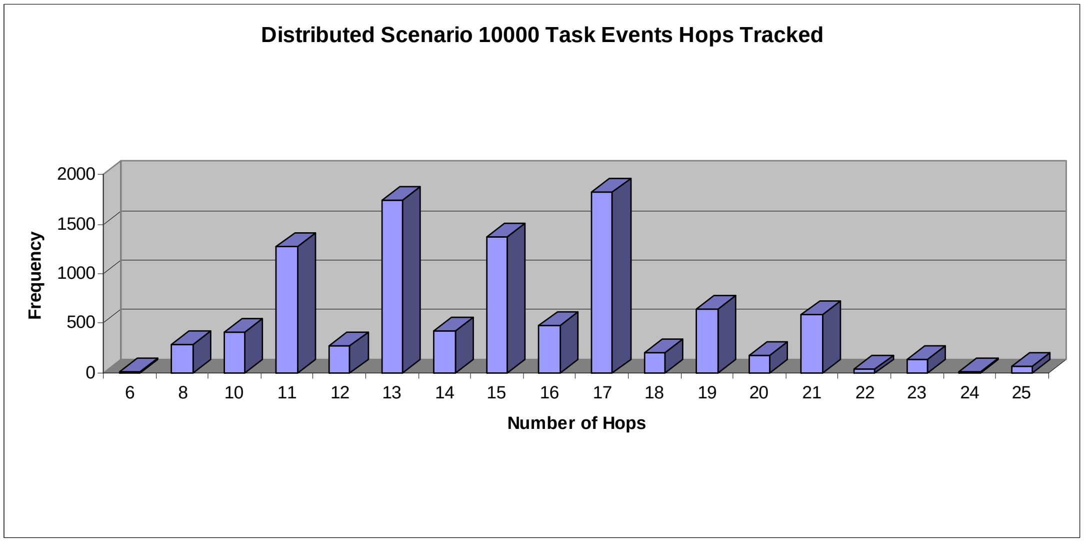


Figure 96 Distributed Computing 10000 Event Scenario Number of Hops

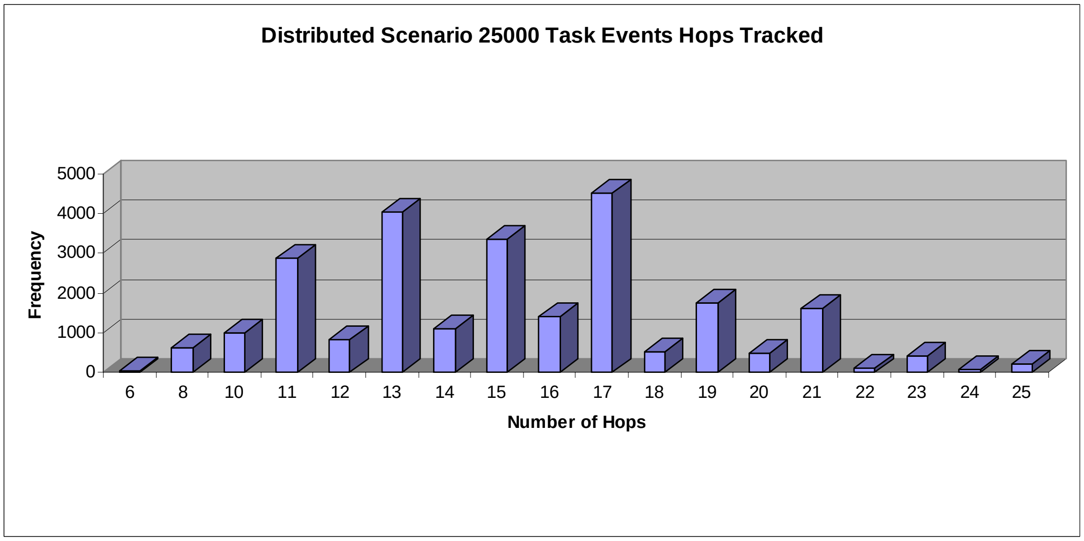


Figure 97 Distributed Computing 25000 Event Scenario Number of Hops

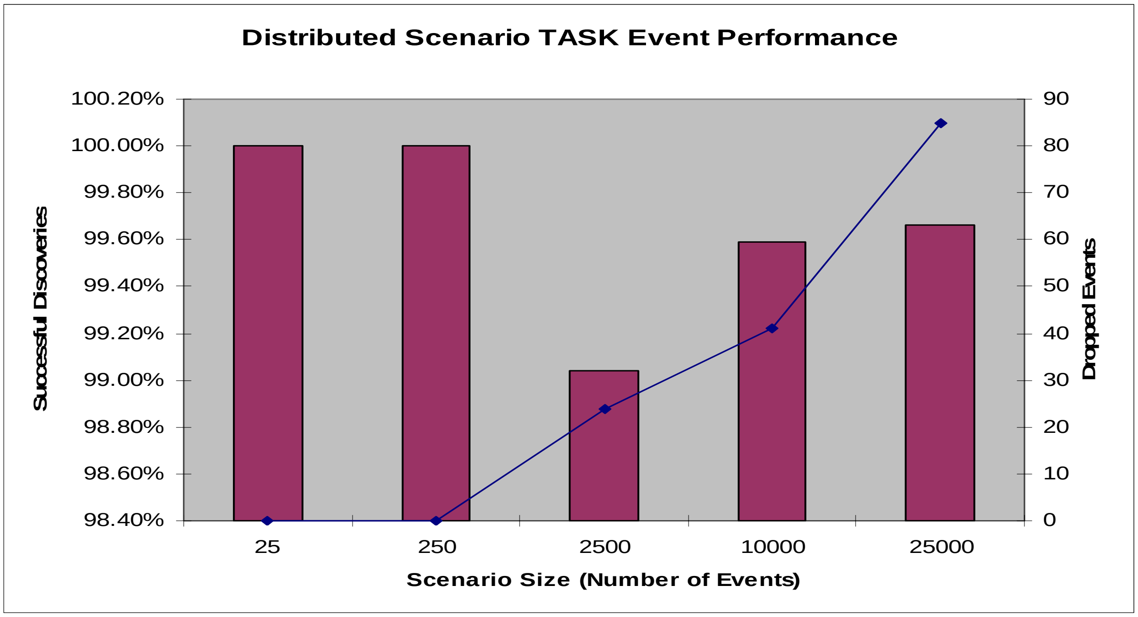


Figure 98 Distributed Computing Scenario Successful TASK Events

One important metric measures how successful the discovery approach was at finding a resource. Figure 98 shows the success rates of the TASK events finding an available resource. The values range between 99.04%-100% successful discoveries or 0-85 dropped packets. In the case of the unsuccessful TASK event not finding a resource, in the real world the VO host would simply try until it finds a resource. But, the simulator does not model this for the purposes of finding the success rates.

One new statistic provided in this research has to do with tracking how scores deviate from a perfect score. A perfect score does not deviate from the score of the resource, that score deviation value would be zero. With the case of the distributed computing scenario, no fields are marked as "don't cares." This means that the 8 bit score composed of CPU, memory, hard drive, and bandwidth would have a mask of 0xFF. This yields scores in the ranges of {0} with a deviation of zero. Considering the bit positions, one would expect scores to deviate around 0 since deviations are not expected in this scenario.

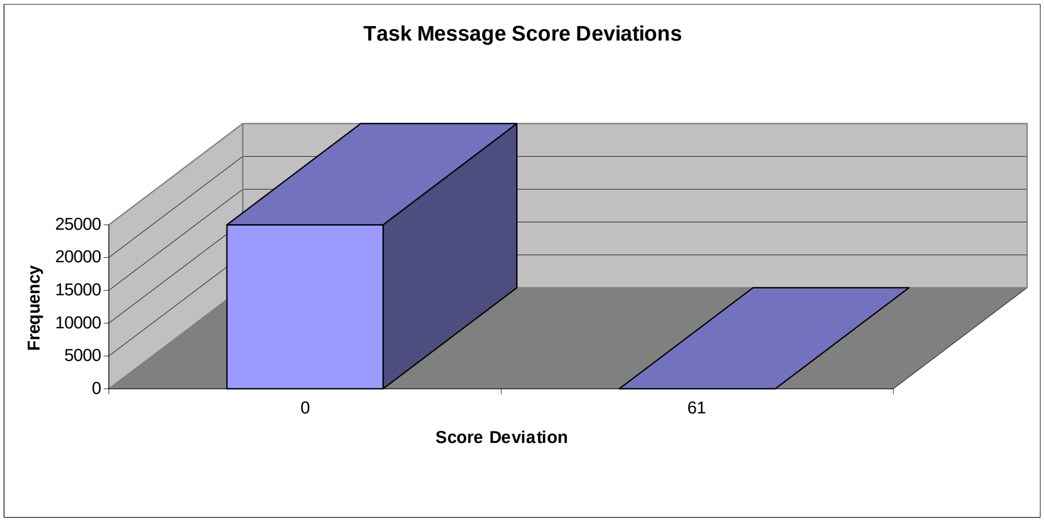


Figure 99 Distributed Computing 25,000 Event Scenario Score Deviation

Looking at Figure 99, almost all of the scores (except one) have the expected score. This figure represents the score deviations in the worst-case 25,000 event scenario. Approximately 100% of the scores fall exactly on 0; the deviation was about 0.2%. Investigating the log file, the one message deviated because another SINGUP message coming from the same resource provider was already in the routing table with a score of 173. Since SIGNUP tables are unpopulated with UNSUBSCRIBE messages, this means the UNSUBSCRIBE message did not arrive at the router yet.

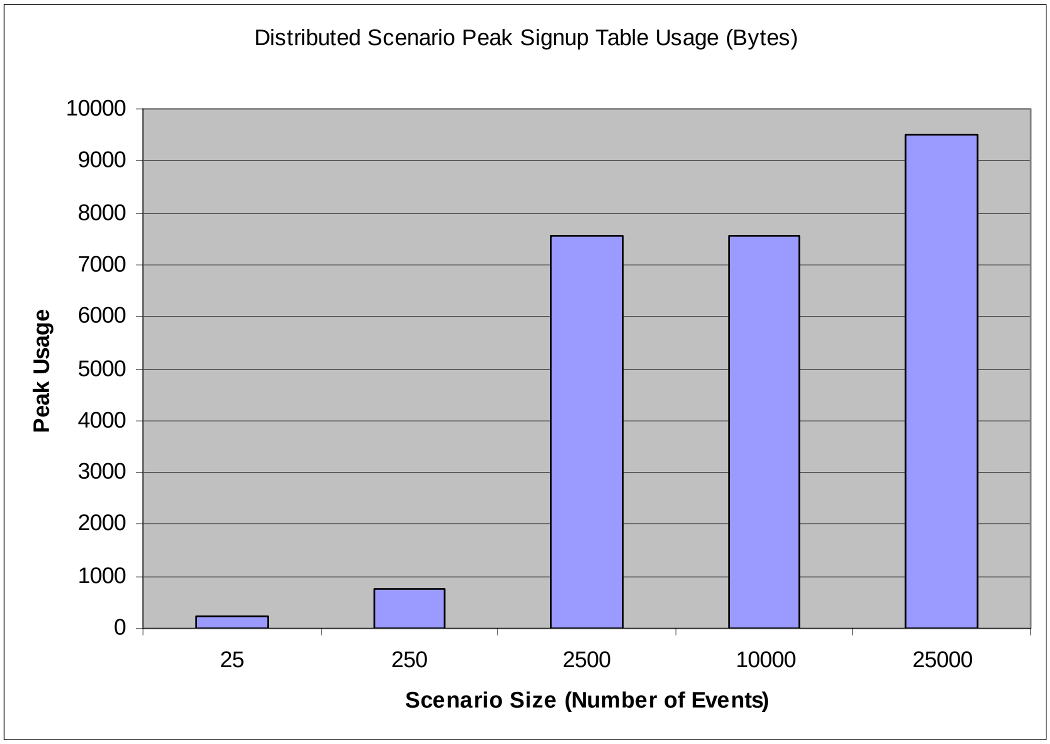


Figure 100 Distributed Computing Scenario Peak Signup Table Usage

The signup table peak memory usage is shown in Figure 100. As the number of events is increased, the memory usage caps at about 9519 bytes. This happens because devices are un-subscribing from the network as time is advancing which reduces the size of the signup table usage. By default, the resource providers unsubscribe from the VO in 200 simulation seconds after the CONFIRM DELIVERY event is sent. The signup table worst-case peak usage can be estimated if UNSUBSCRIBE are not sent. Since there are 25,000 events (worst case) from 25,000 different resource providers with a 4 byte address, 10 VOs with each with a 4 byte VO Host IP address, and 10 bytes per SIGNUP TABLE ENTRY, then the worst case peak memory consumption for any particular router is 450 KB or 25KB * (4 + 4 + 10). As a reminder, signup entries are removed every 24 hours to prevent uncontrolled growth of these tables.

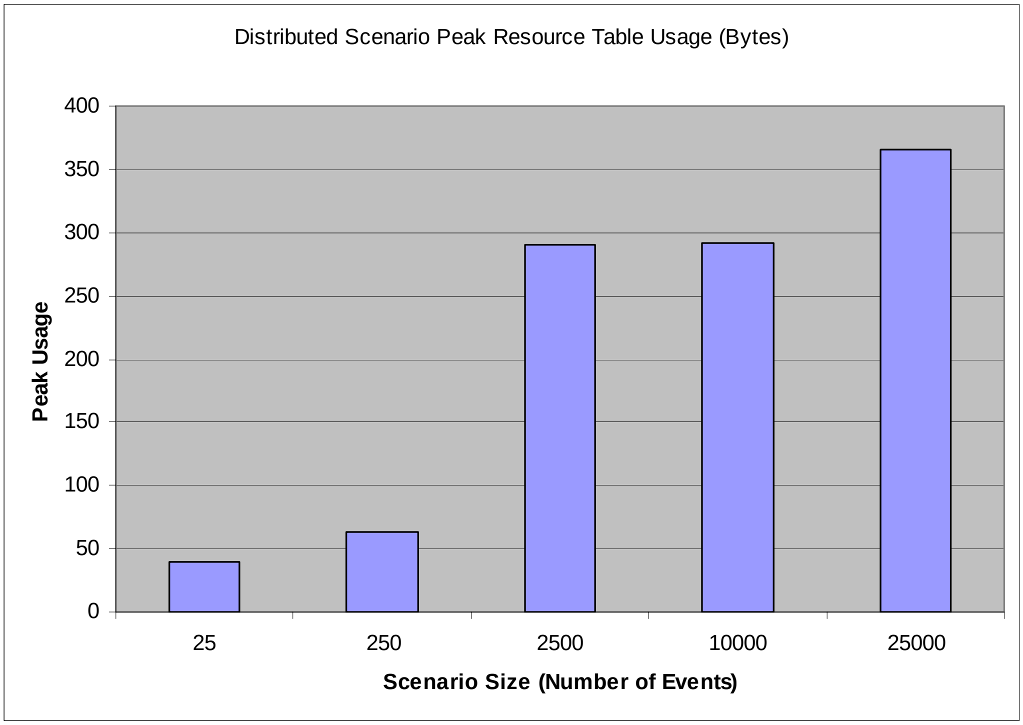


Figure 101 Distributed Computing Scenario Peak Resource Table Usage

Next, the peak resource table usage is examined in Figure 101. The usage caps at 366 bytes as the number of TASK events grow. This happens for different reasons than the signup table previously presented. The main reason is that the resource table has a smaller sized hash key and uses a one-byte score to lookup data. As the scenarios grow larger, once there are more than 256 resource providers, the scores will definitely overlap. The resource table is optimized to aggregate and count the number of devices with a particular score rather than to list individual resource providers. Also in this case, the simulation is greedy in discovering resources. Because resources are discovered in a greedy fashion, the table size does not grow very large because resources are consumed very quickly. Again the worst case tables size could be estimated by considering 25,000 resource IP addresses with 2 byte hash key, the score of one byte, the next hop IP of 4 bytes, and the count of one byte totaling 200KB or 25KB * (2 + 1 + 4 + 1).

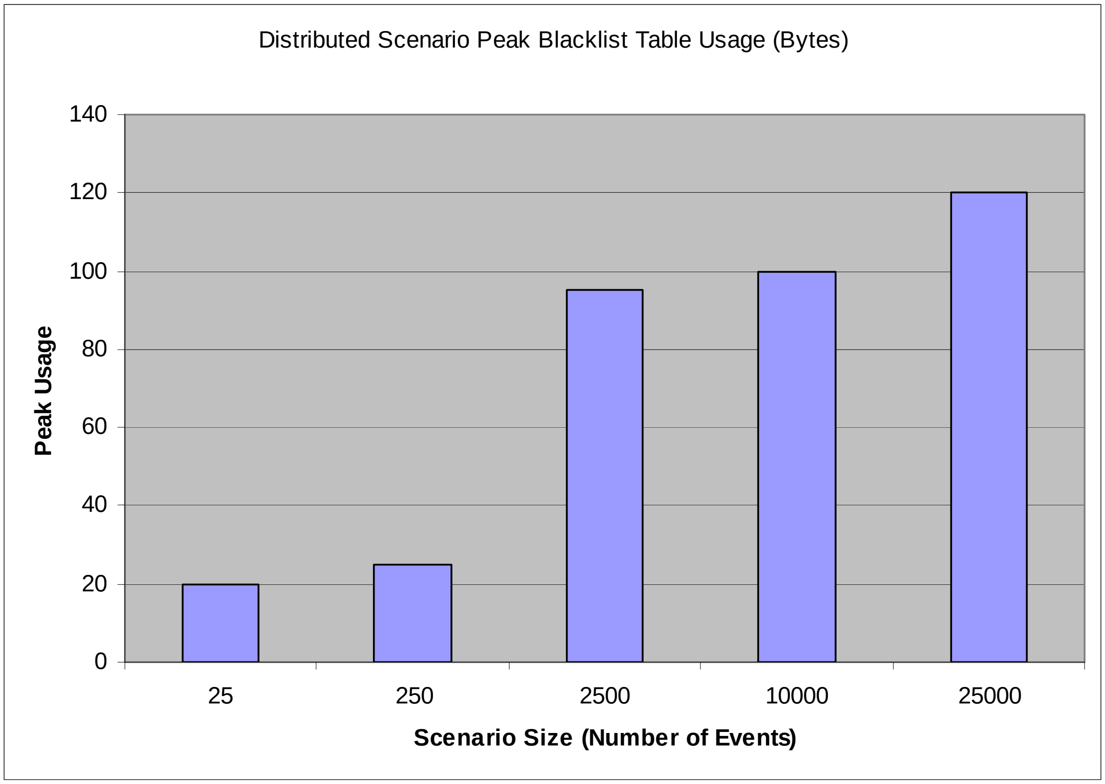


Figure 102 Distributed Computing Scenario Peak Blacklist Table Usage

The final results examined for the distributed computing scenarios are the blacklist tables. Since the blacklist tables are populated when the SIGNUP event is sent and unpopulated with the ACCEPT event is sent, the tables are much smaller than the others because the SIGNUP and ACCEPT events happen very close to each other in time. As shown in Figure 102, the memory usage caps at a value similarly to the other tables; this time at a value of 120 bytes. Doing the math, the BLACKLIST TABLE contains a four-byte IP address and a one-byte count totaling 5 bytes. Dividing 120 by 5 means that each router kept no more than 24 entries in its BLACKLIST TABLE at a given time. The worst case blacklist size can be calculated as well by multiplying 25,000 resource providers times 5 bytes totaling 125KB. This would imply that all 25,000 resource providers send their SIGNUP events at the same time through the same router.

Computer-in-the-Loop Instrumentation

The computer-in-the-loop simulation results are presented in this section. The event distribution diagrammed in Figure 103 shows that the distribution of traffic between each of the VOs is roughly the same. The VO hosts and resource providers are distributed throughout the network and the average number of hops is around 14.1 as shown in Figure 104, and the discovery event hops are presented for each scenario in Figure 105 thru Figure 109. A hop is considered movement from one network device to another.

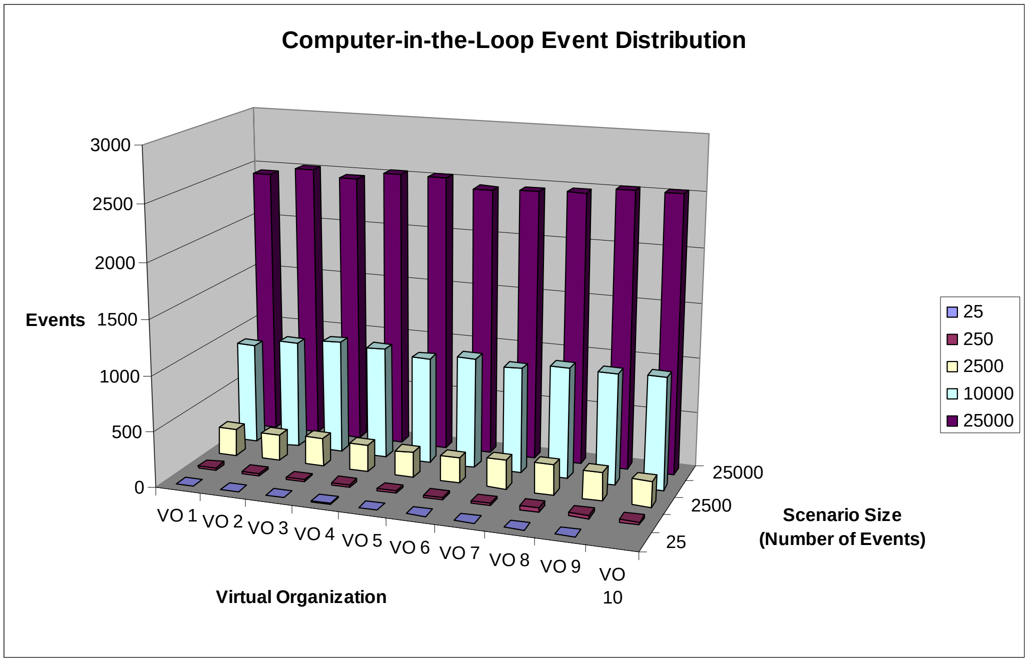


Figure 103 Computer-in-the-Loop Scenario Event Distribution

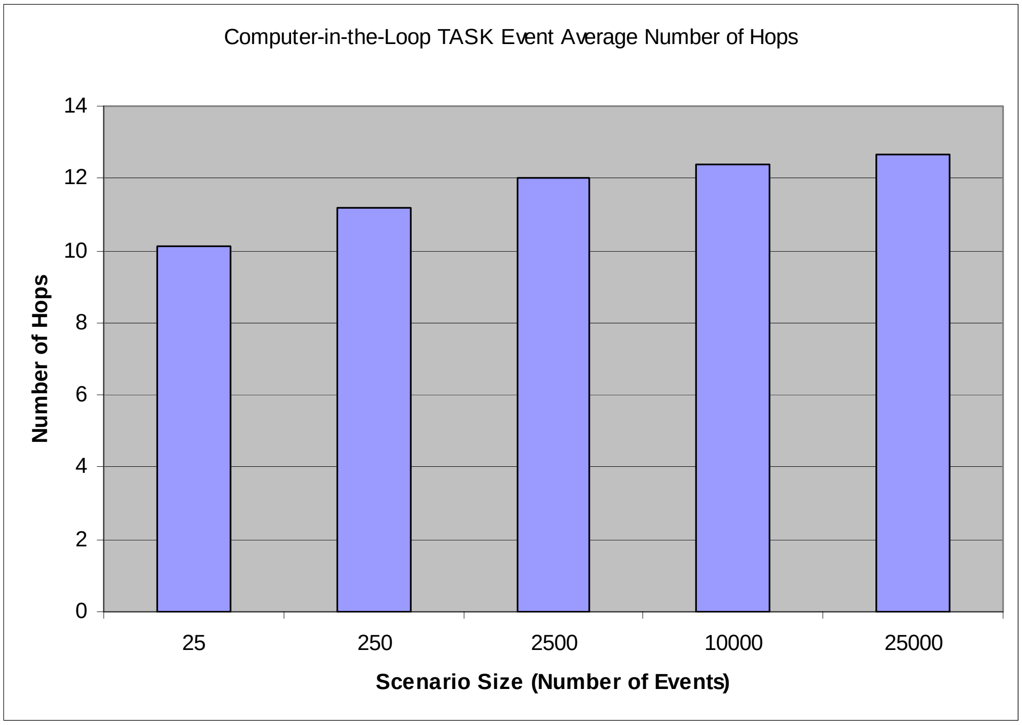


Figure 104 Computer-in-the-Loop Scenario Average Number of Hops

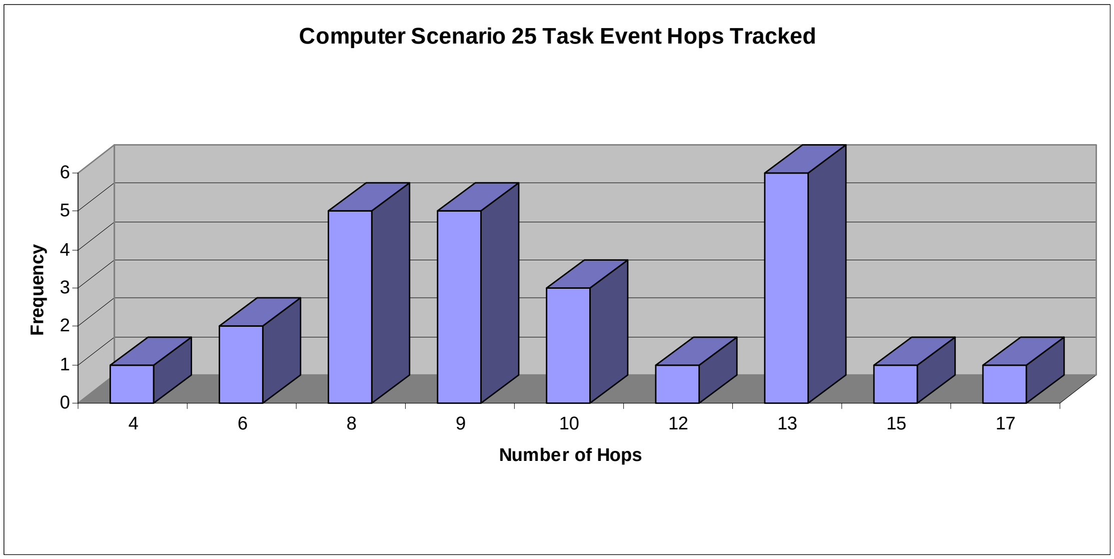


Figure 105 Computer-in-the-Loop 25 Event Scenario Number of Hops

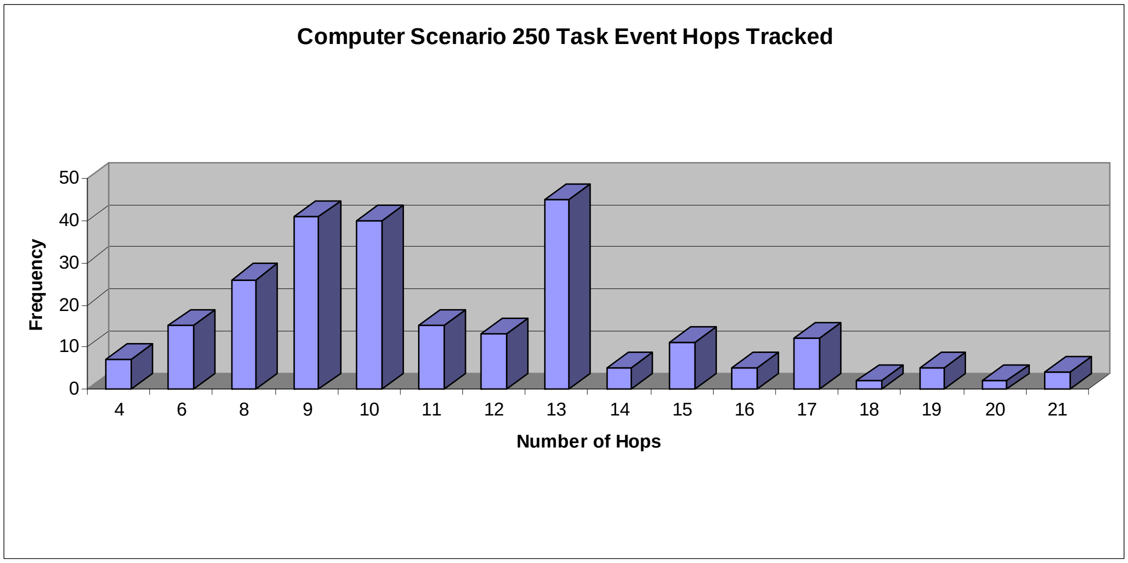


Figure 106 Computer-in-the-Loop 250 Event Scenario Number of Hops

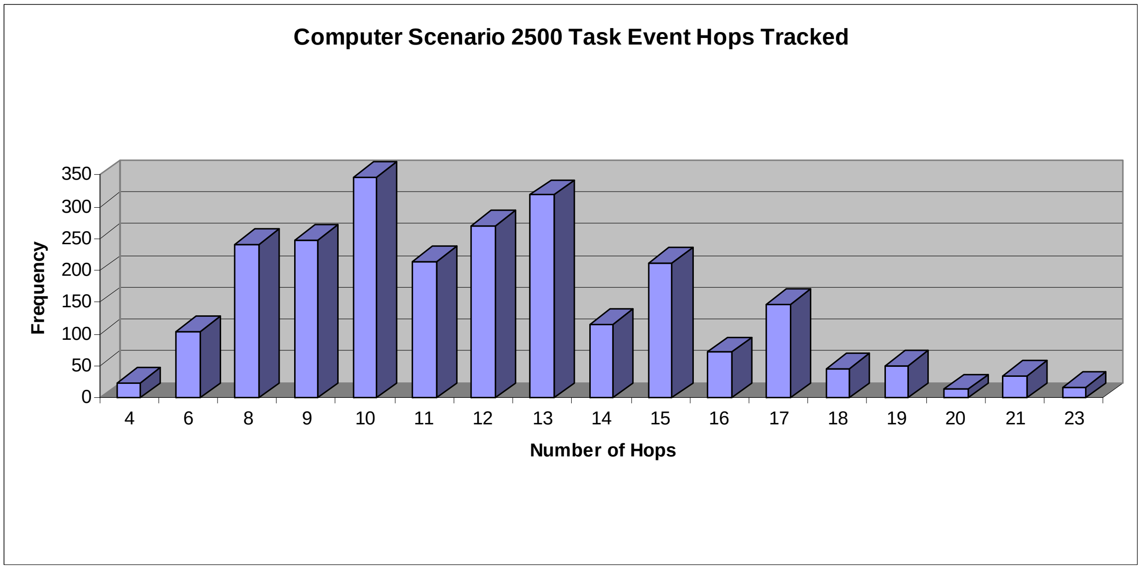


Figure 107 Computer-in-the-Loop 2500 Event Scenario Number of Hops

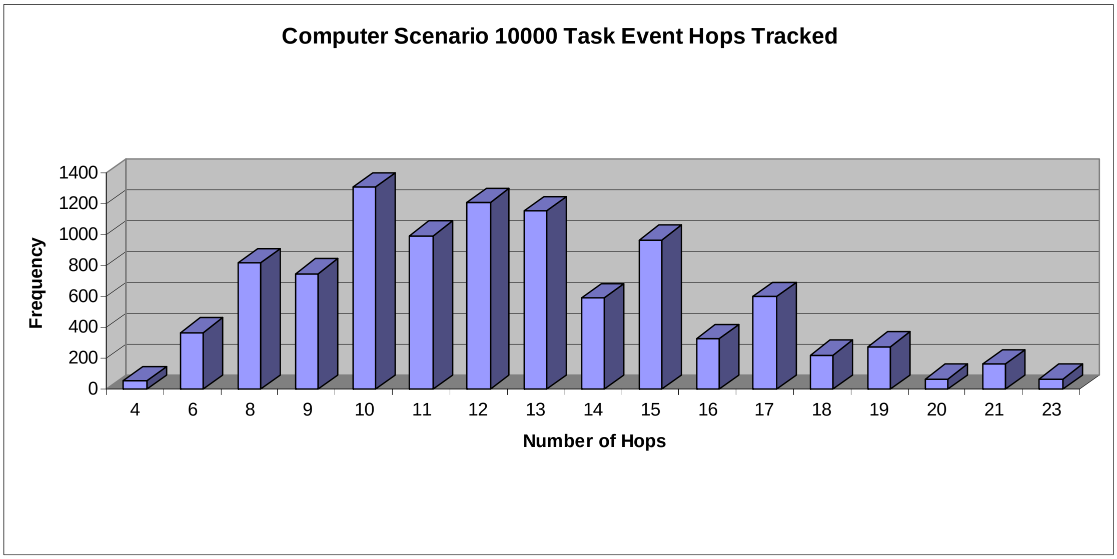


Figure 108 Computer-in-the-Loop 10000 Event Scenario Number of Hops

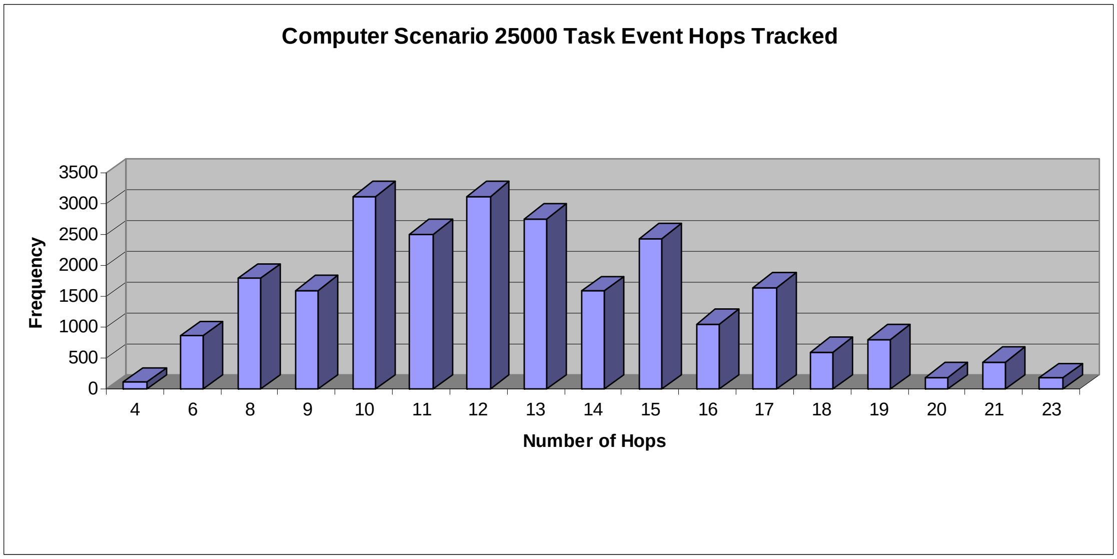


Figure 109 Computer-in-the-Loop 25000 Event Scenario Number of Hops

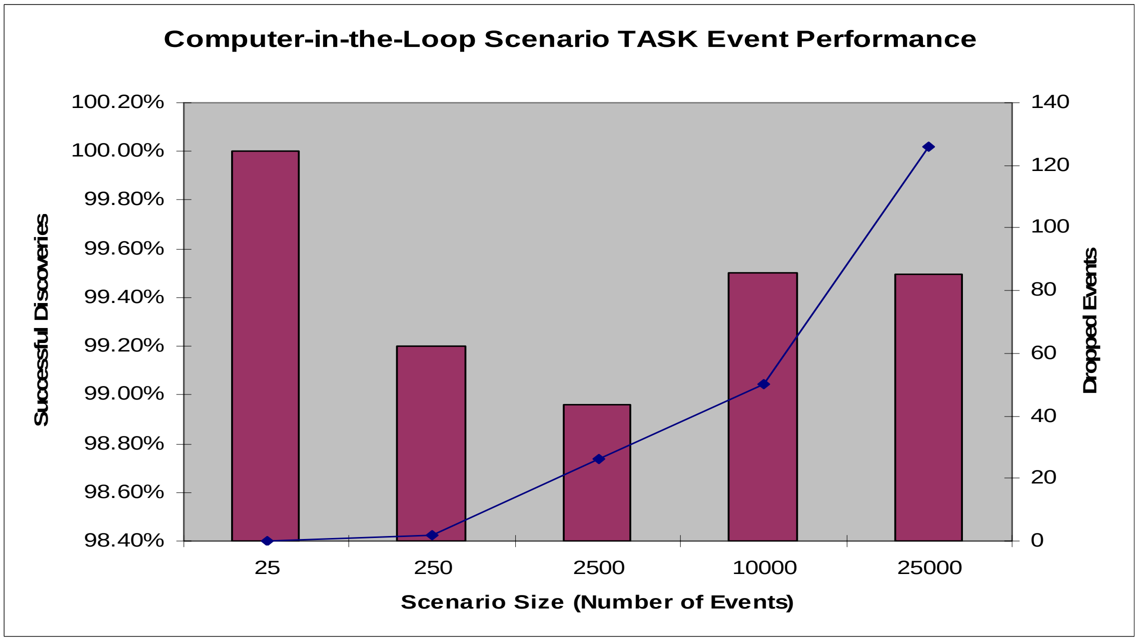


Figure 110 Computer-in-the-Loop Scenario Successful TASK Events

One important metric measures how successful the discovery approach was at finding a resource. Figure 110 shows the success rates of the TASK events finding an available resource. The values range between 98.96%-100% successful discoveries or 0-126 dropped packets. In the case of the unsuccessful TASK event not finding a resource, in the real world the VO host would simply try until it finds a resource. But, the simulator does not model this for the purposes of finding the success rates.

One new statistic provided in this research has to do with tracking how scores deviate from a perfect score. A perfect score does not deviate from the score of the resource, that score deviation value would be zero. With the case of the computer-in-the-loop scenario, the hard drive field is marked as a "don't care." This means that the 8 bit score composed of CPU, memory, hard drive, and bandwidth would have a mask of 0xF3. This yields scores in the set of {0, 4, 8, 12} with a deviation of zero. Considering the bit positions, one would expect scores to deviate around spots of the don't cares.

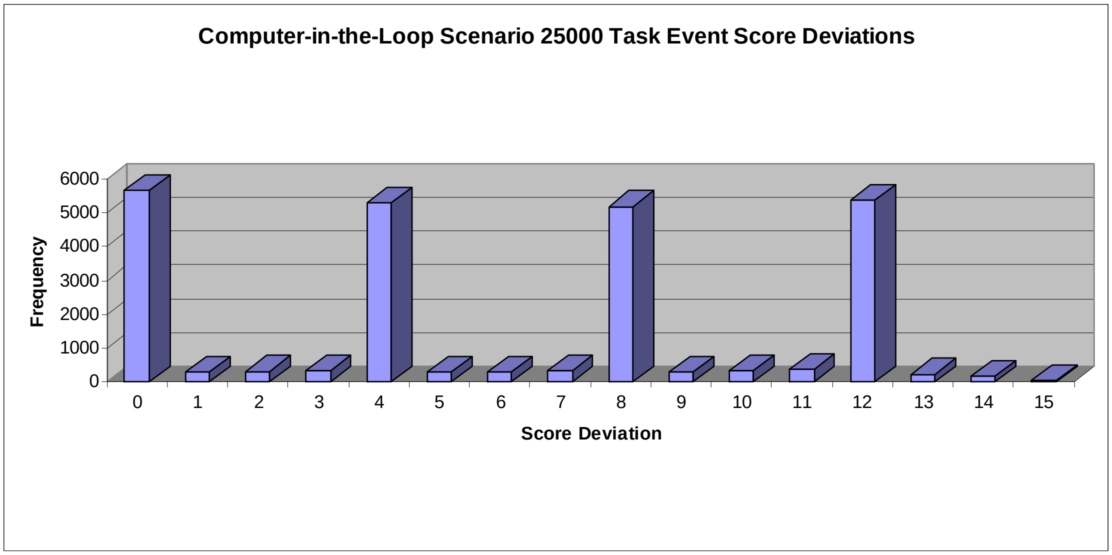


Figure 111 Computer-in-the-Loop Scenario Score Deviation for the 25,000 Event Scenario

Looking at Figure 111, the scores tend to deviate in that fashion. This figure represents the score deviations in the 25,000 event scenario. Approximately 86% of the scores have a deviation of zero from the intended score. When numbers deviate from the desired score, they deviate by an average of 6.07.

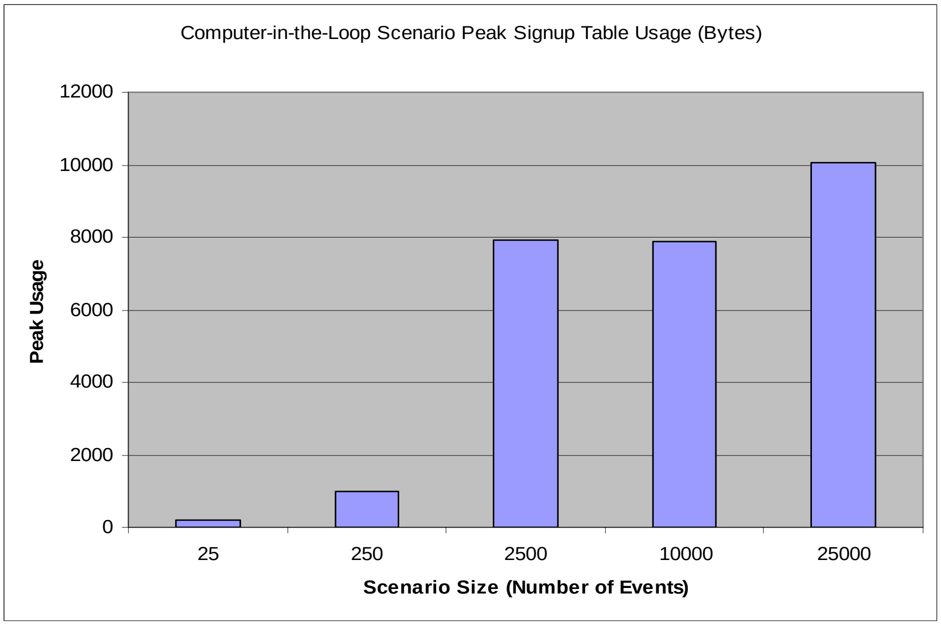


Figure 112 Computer-in-the-Loop Scenario Peak Signup Table Usage

The signup table peak memory usage is shown in Figure 112. As the number of events is increased, the memory usage caps at about 10043 bytes. This happens because devices are un-subscribing from the network as time is advancing which reduces the size of the signup table usage. By default, the resource providers unsubscribe from the VO in 200 simulation seconds after the CONFIRM DELIVERY event is sent. The signup table worst-case peak usage can be estimated if UNSUBSCRIBE are not sent. Since there are 25,000 events (worst case) from 25,000 different resource providers with a 4 byte address, 10 VOs with each with a 4 byte VO Host IP address, and 10 bytes per SIGNUP TABLE ENTRY, then the worst case peak memory consumption for any particular router is 450 KB or 25KB * (4 + 4 + 10). As a reminder, signup entries are removed every 24 hours to prevent uncontrolled growth of these tables.

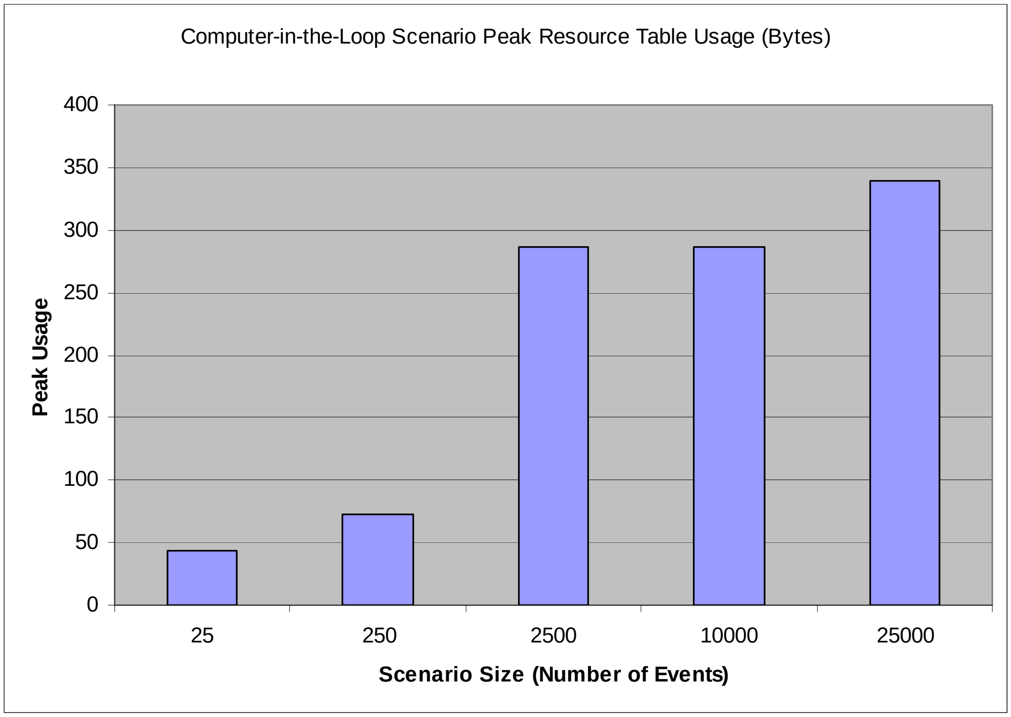


Figure 113 Computer-in-the-Loop Scenario Peak Resource Table Usage

Next, the peak resource table usage is examined in Figure 113. The usage caps at 339 bytes as the number of TASK events grow. This happens for different reasons than the signup table previously presented. The resource table has a smaller sized hash key and uses a one-byte score to lookup data. As the scenarios grow larger, once there are more than 256 resource providers, the scores will definitely overlap. The resource table is optimized to aggregate and count the number of devices with a particular score rather than to list individual resource providers. Also in this case, the simulation is greedy in discovering resources. Because resources are discovered in a greedy fashion, the table size does not grow very large because resources are consumed very quickly. Again the worst case tables size could be estimated by considering 25,000 resource IP addresses with 2 byte hash key, the score of one byte, the next hop IP of 4 bytes, and the count of one byte totaling 200KB or 25KB * (2 + 1 + 4 + 1).

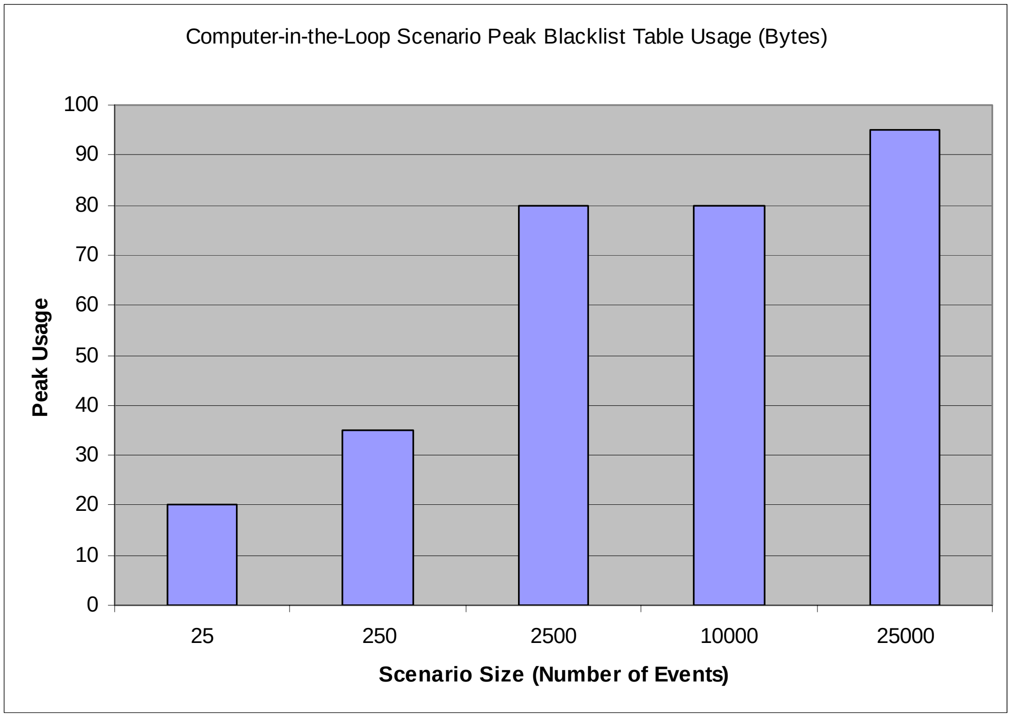


Figure 114 Computer-in-the-Loop Scenario Peak Blacklist Table Usage

The final results examined for the computer-in-the-loop scenarios are the blacklist tables. Since the blacklist tables are populated when the SIGNUP event is sent and unpopulated with the ACCEPT event is sent, the tables are much smaller than the others because the SIGNUP and ACCEPT events happen very close to each other in time. As shown in Figure 114, the memory usage caps at a value similarly to the other tables; this time at value of 95 bytes. Doing the math, the BLACKLIST TABLE contains a four-byte IP address and a one-byte count totaling 5 bytes. Dividing 95 by 5 means that each router kept no more than 19 entries in its BLACKLIST TABLE at a given time. The worst case blacklist size can be calculated as well by multiplying 25,000 resource providers times 5 bytes totaling 125KB. This would imply that all 25,000 resource providers send their SIGNUP events at the same time through the same router.

Large-Scale Data Analysis

The large-scale data analysis simulation results are presented in this section. The event distribution diagrammed in Figure 115 shows that the distribution of traffic between each of the VOs is roughly the same. The VO hosts and resource providers are distributed throughout the network and the average number of hops is around 14.43 as shown in Figure 116, and the discovery event hops are presented for each scenario in Figure 117 thru Figure 121. A hop is considered movement from one network device to another.

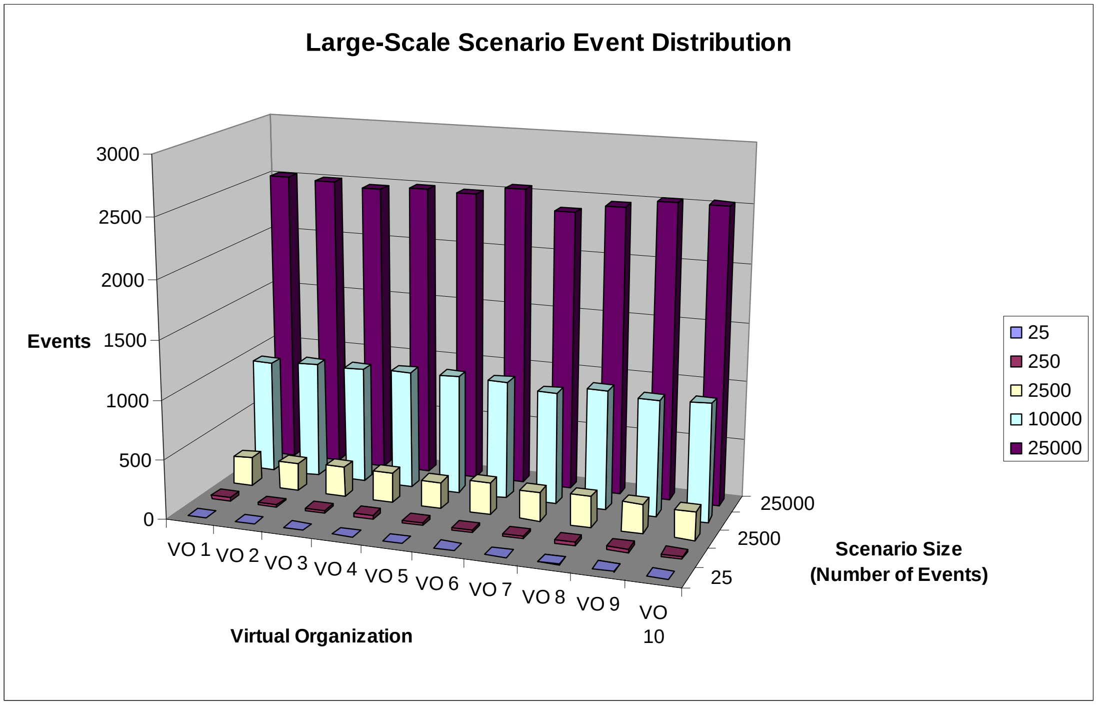


Figure 115 Large-Scale Scenario Event Distribution

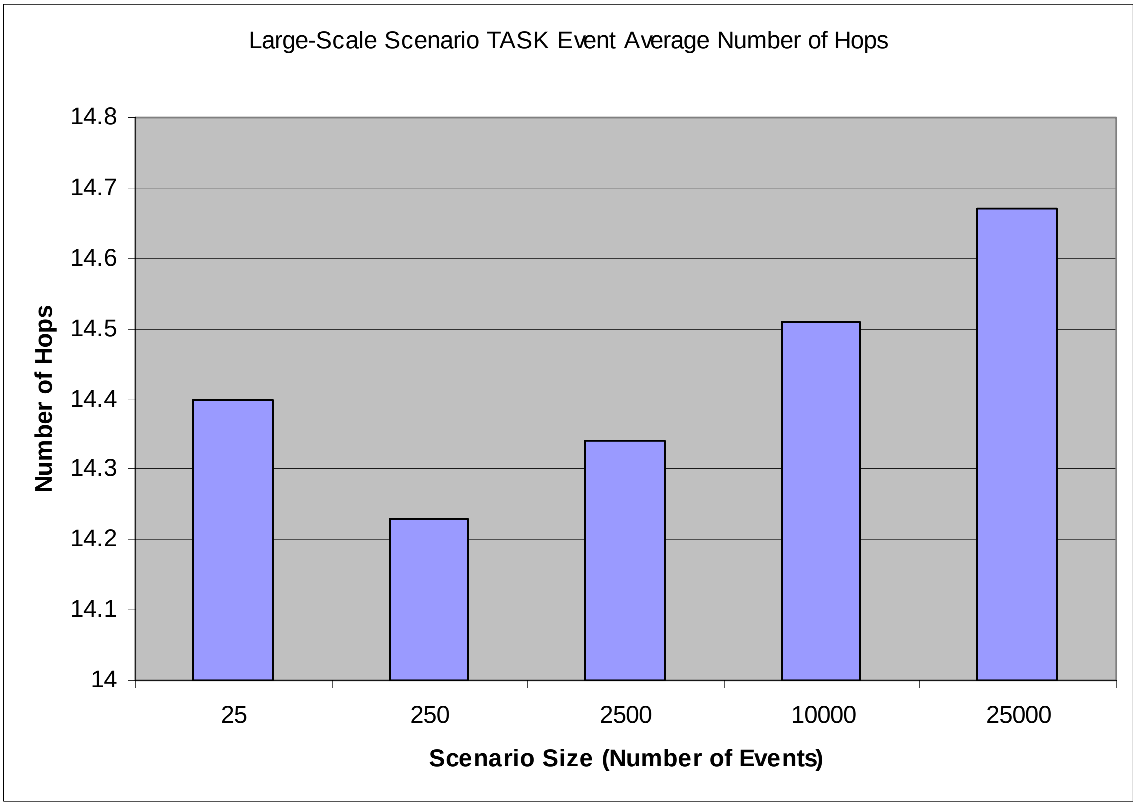


Figure 116 Large-Scale Scenario Average Number of Hops

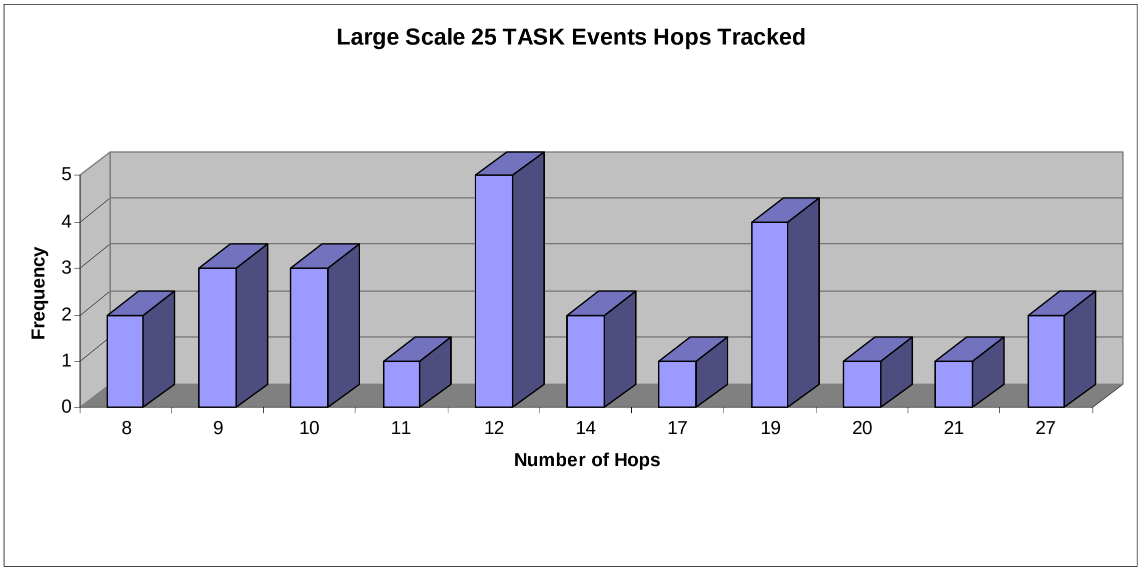


Figure 117 Large-Scale Scenario Number of Hops for 25 Event Scenario

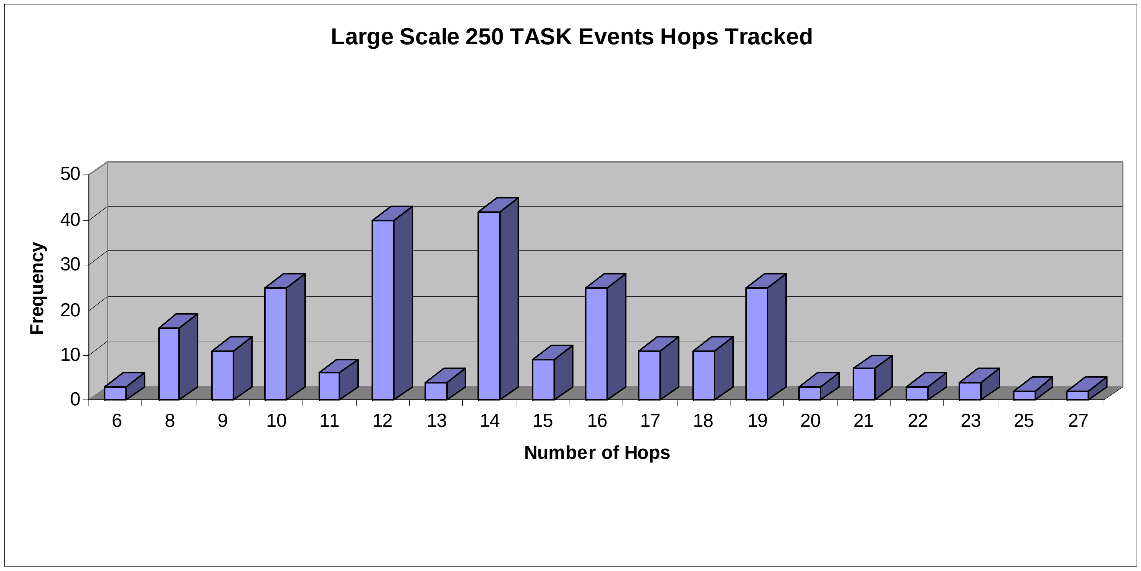


Figure 118 Large-Scale Scenario Number of Hops for 250 Event Scenario

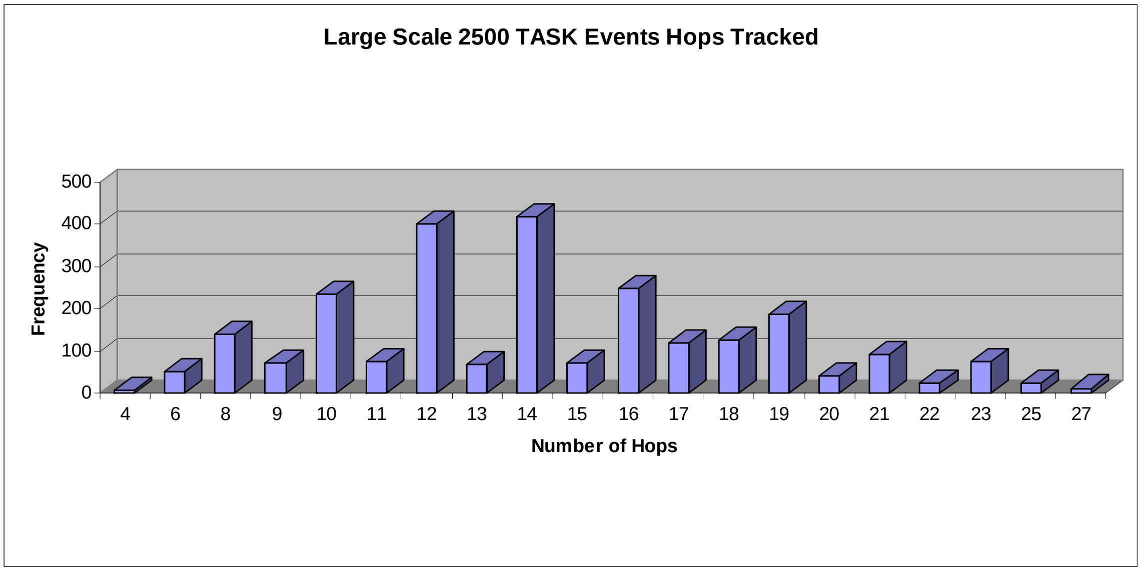


Figure 119 Large-Scale Scenario Number of Hops for 2500 Event Scenario

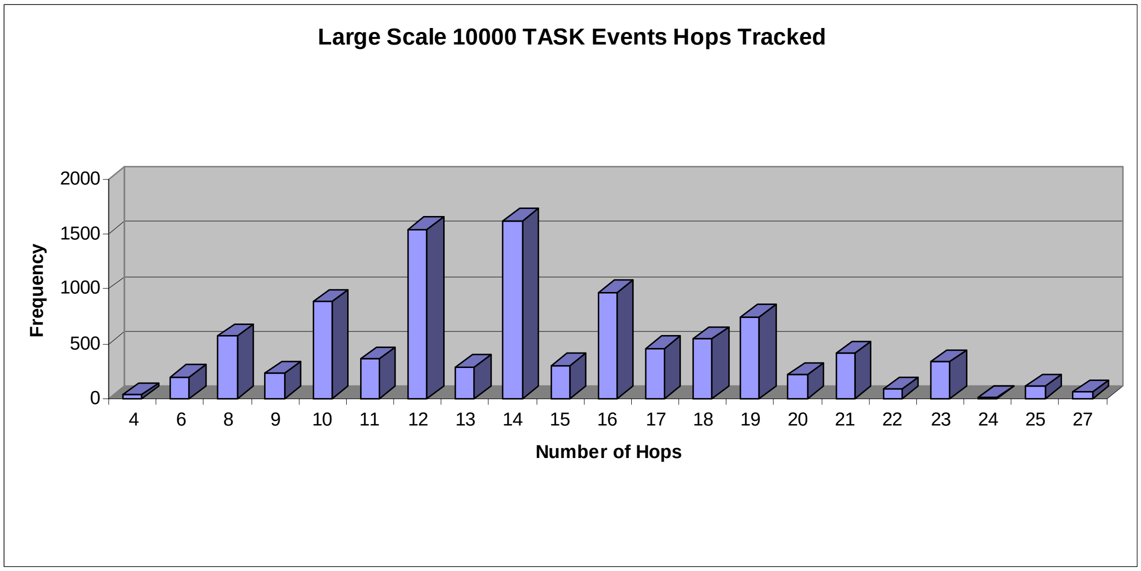


Figure 120 Large-Scale Scenario Number of Hops for 10000 Event Scenario

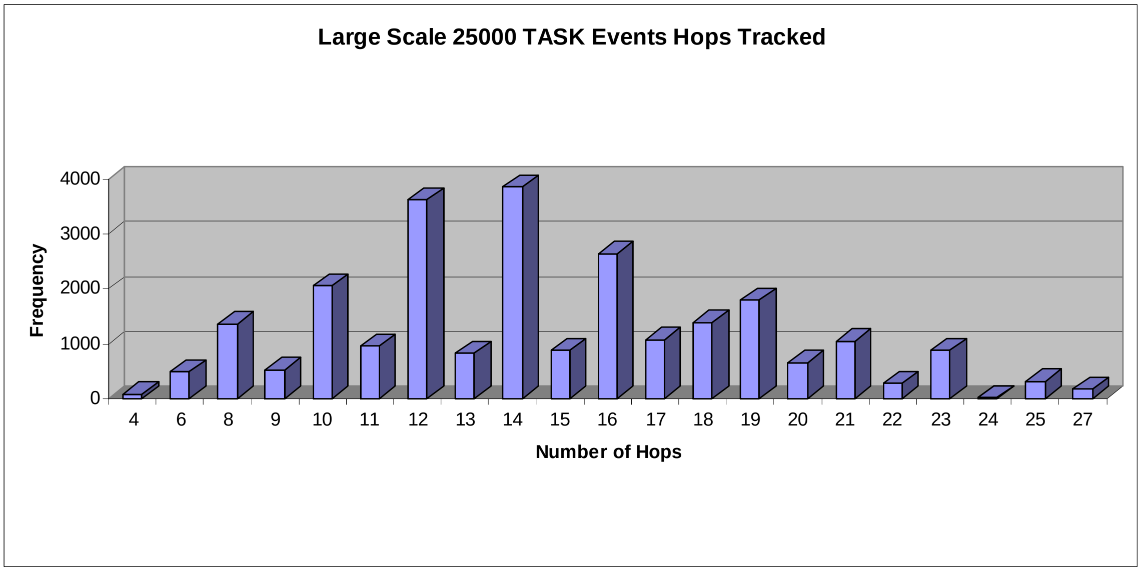


Figure 121 Large-Scale Scenario Number of Hops for 25000 Event Scenario

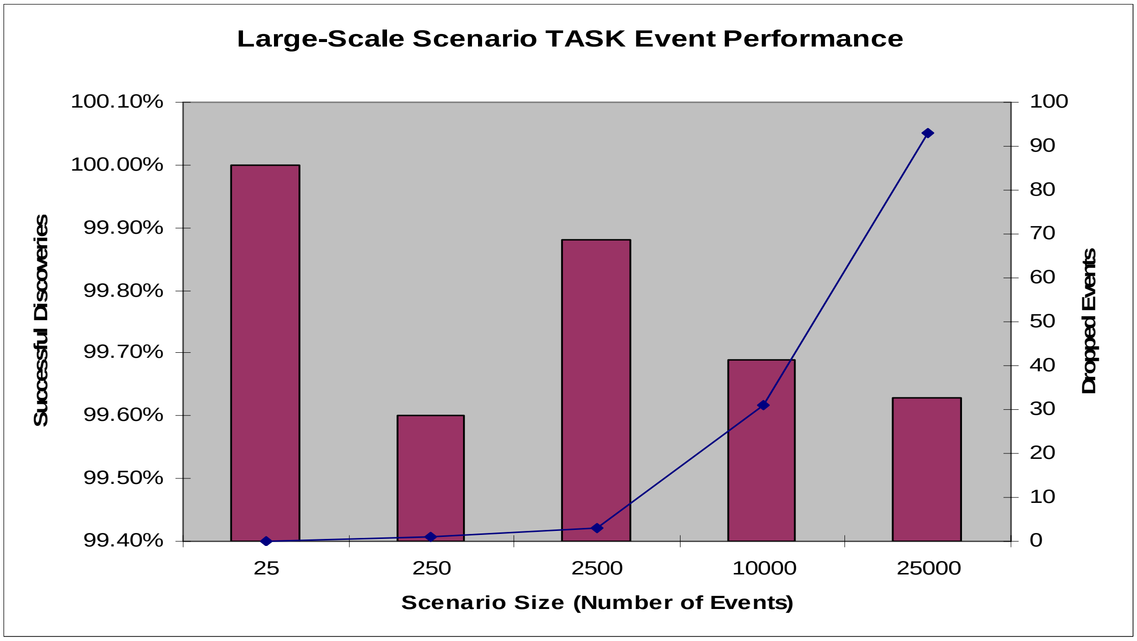


Figure 122 Large-Scale Scenario Successful TASK Events

One important metric measures how successful the discovery approach was at finding a resource. Figure 122 shows the success rates of the TASK events finding an available resource. The values range between 99.6%-100% successful discoveries or 0-93 dropped packets. In the case of the unsuccessful TASK event not finding a resource, in the real world the VO host would simply try until it finds a resource. But, the simulator does not model this for the purposes of finding the success rates.

One new statistic provided in this research has to do with tracking how scores deviate from a perfect score. A perfect score does not deviate from the score of the resource, that score deviation value would be zero. With the case of the large-scale scenario, the CPU, memory, and bandwidth fields are “don’t cares.” This means that the 8 bit score composed of CPU, memory, hard drive, and bandwidth would have a mask of 0x0C. This yields scores in the set of {0-3, 16-19, 32-35, …, 240-243} with a deviation of zero. Considering the bit positions, one would expect scores to deviate around spots of the don’t cares.

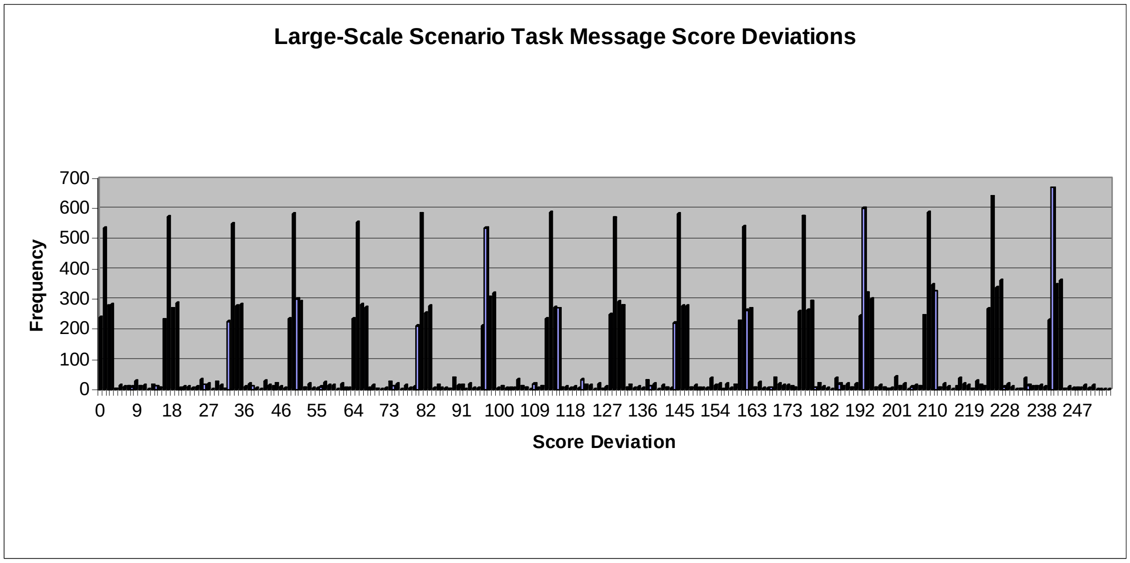


Figure 123 Large-Scale Scenario Score Deviation for the 25,000 Event Scenario

Looking at Figure 123, the scores tend to deviate in that fashion. This figure represents the score deviations in the 25,000 event scenario. Approximately 89.9% of the scores have a deviation of zero from the intended score. When numbers deviate from the desired score, they deviate by an average of 125.42.

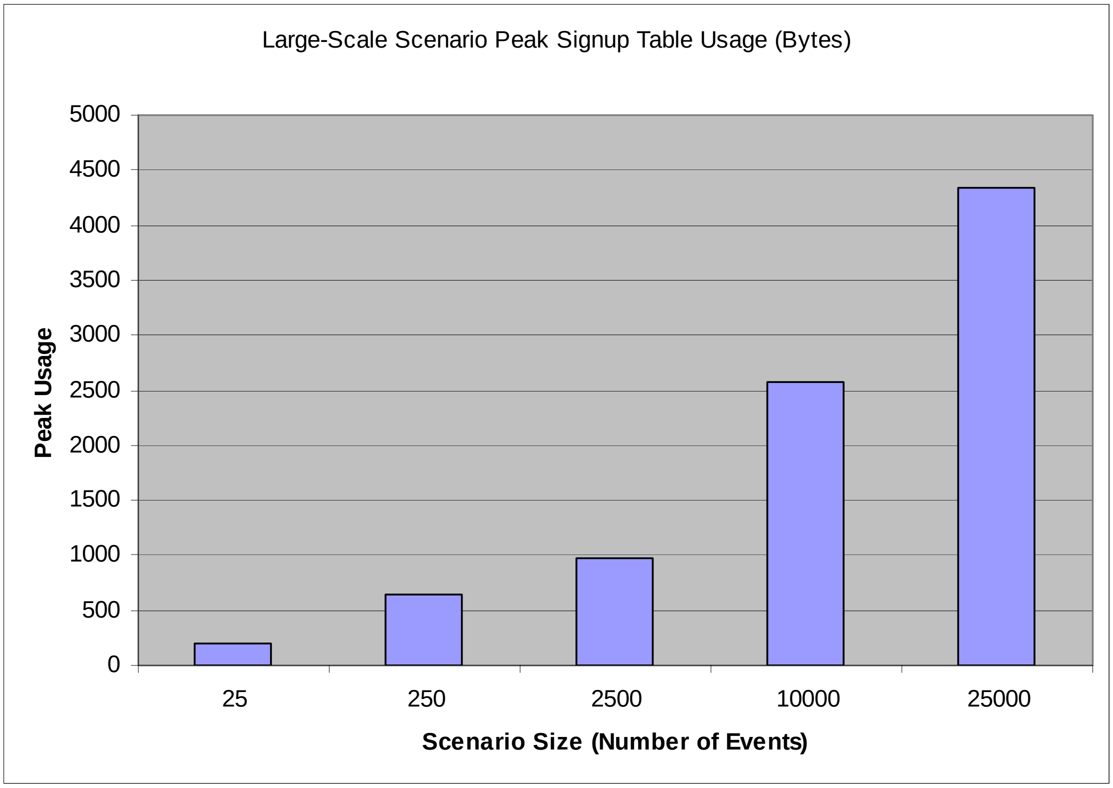


Figure 124 Large-Scale Scenario Peak Signup Table Usage

The signup table peak memory usage is shown in Figure 124. As the number of events is increased, the memory usage caps at about 4300 bytes. This happens because devices are un-subscribing from the network as time is advancing which reduces the size of the signup table usage. By default, the resource providers unsubscribe from the VO in 200 simulation seconds after the CONFIRM DELIVERY event is sent. The signup table worst-case peak usage can be estimated if UNSUBSCRIBE are not sent. Since there are 25,000 events (worst case) from 25,000 different resource providers with a 4 byte address, 10 VOs with each with a 4 byte VO Host IP address, and 10 bytes per SIGNUP TABLE ENTRY, then the worst case peak memory consumption for any particular router is 450 KB or 25KB * (4 + 4 + 10). As a reminder, signup entries are removed every 24 hours to prevent uncontrolled growth of these tables.

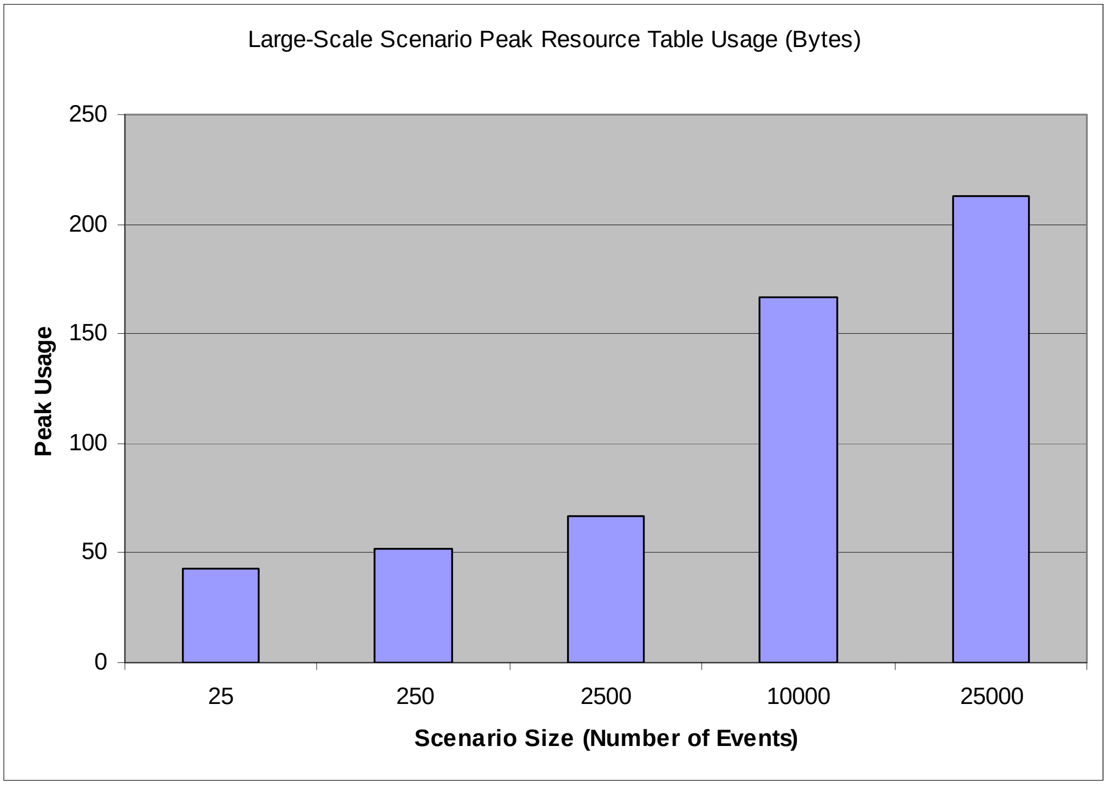


Figure 125 Large-Scale Scenario Peak Resource Table Usage

Next, the peak resource table usage is examined in Figure 125. The usage caps at 213 bytes as the number of TASK events grow. This happens for different reasons than the signup table previously presented. The resource table has a smaller sized hash key and uses a one-byte score to lookup data. As the scenarios grow larger, once there are more than 256 resource providers, the scores will definitely overlap. The resource table is optimized to aggregate and count the number of devices with a particular score rather than to list individual resource providers. Also in this case, the simulation is greedy in discovering resources. Because resources are discovered in a greedy fashion, the table size does not grow very large because resources are consumed very quickly. Again the worst case tables size could be estimated by considering 25,000 resource IP addresses with 2 byte hash key, the score of one byte, the next hop IP of 4 bytes, and the count of one byte totaling 200KB or 25KB * (2 + 1 + 4 + 1).

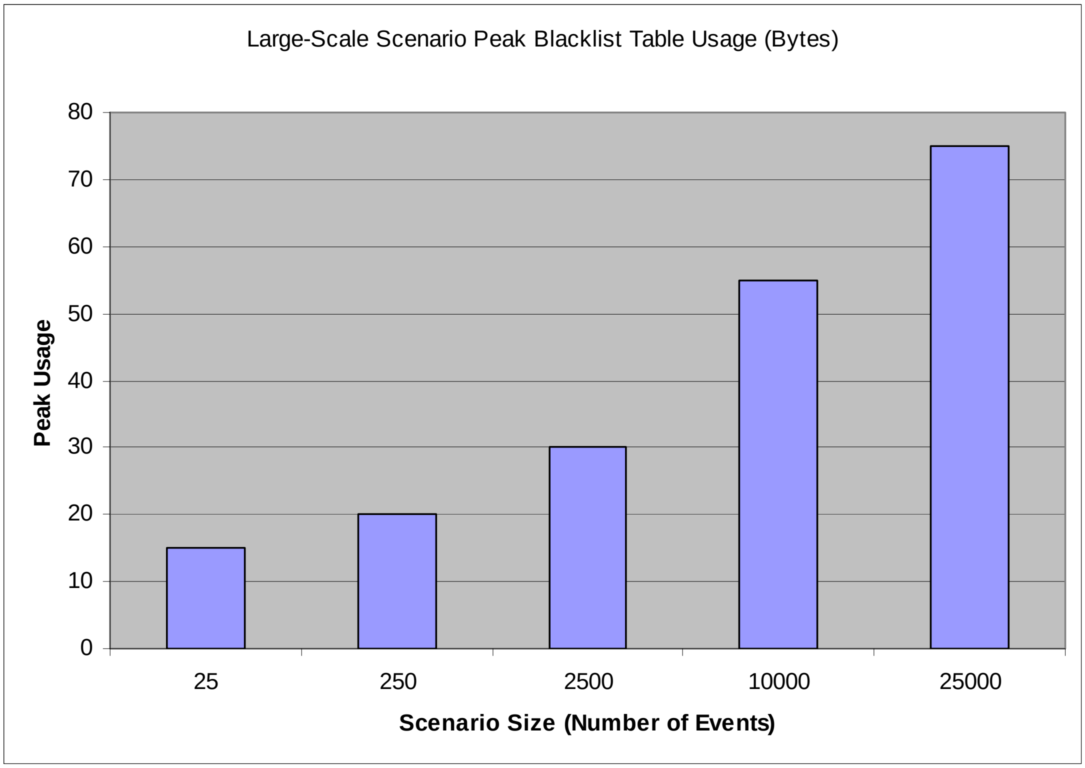


Figure 126 Large-Scale Scenario Peak Blacklist Table Usage

The final results examined for the large-scale scenarios are the blacklist tables. Since the blacklist tables are populated when the SIGNUP event is sent and unpopulated with the ACCEPT event is sent, the tables are much smaller than the others because the SIGNUP and ACCEPT events happen very close to each other in time. As shown in Figure 126, the memory usage caps at a value similarly to the other tables; this time at value of 75 bytes. Doing the math, the BLACKLIST TABLE contains a four-byte IP address and a one-byte count totaling 5 bytes. Dividing 75 by 5 means that each router kept no more than 15 entries in its BLACKLIST TABLE at a given time. The worst case blacklist size can be calculated as well by multiplying 25,000 resource providers times 5 bytes totaling 125KB. This would imply that all 25,000 resource providers send their SIGNUP events at the same time through the same router.

Collaborative Work

The collaborative work simulation results are presented in this section. The event distribution diagrammed in Figure 127 shows that the distribution of traffic between each of the VOs is roughly the same. The VO hosts and resource providers are distributed throughout the network and the average number of hops is around 15 as shown in Figure 128, and the discovery event hops are presented for each scenario in Figure 129 thru Figure 133. A hop is considered movement from one network device to another.

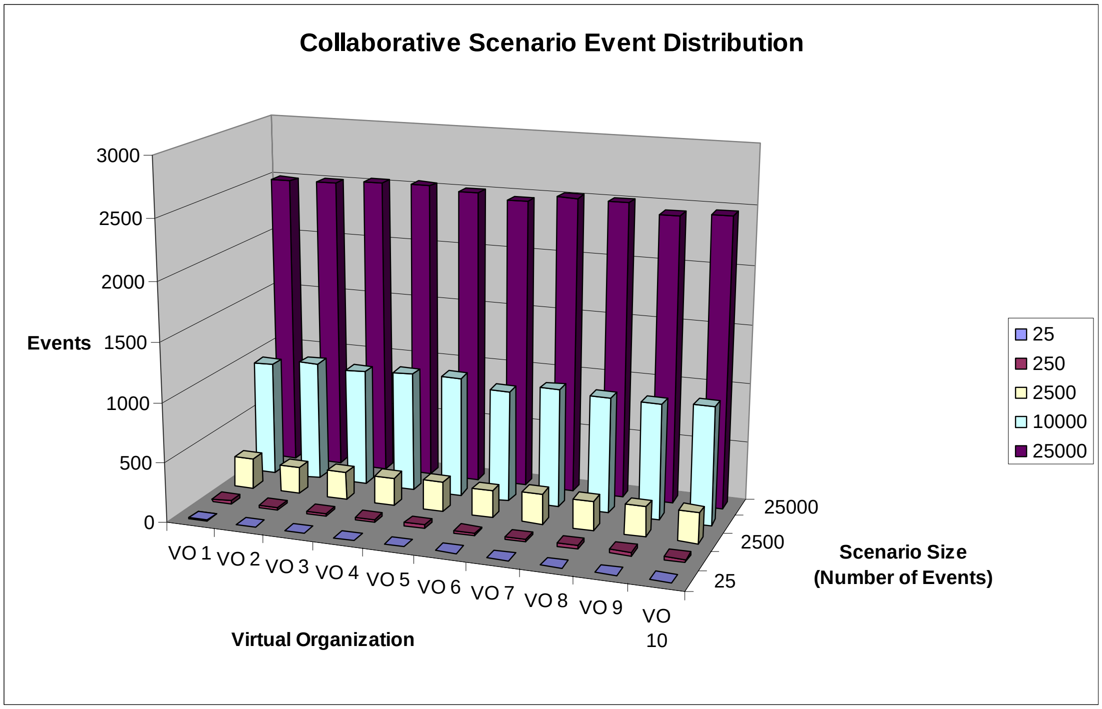


Figure 127 Collaborative Work Scenario Event Distribution

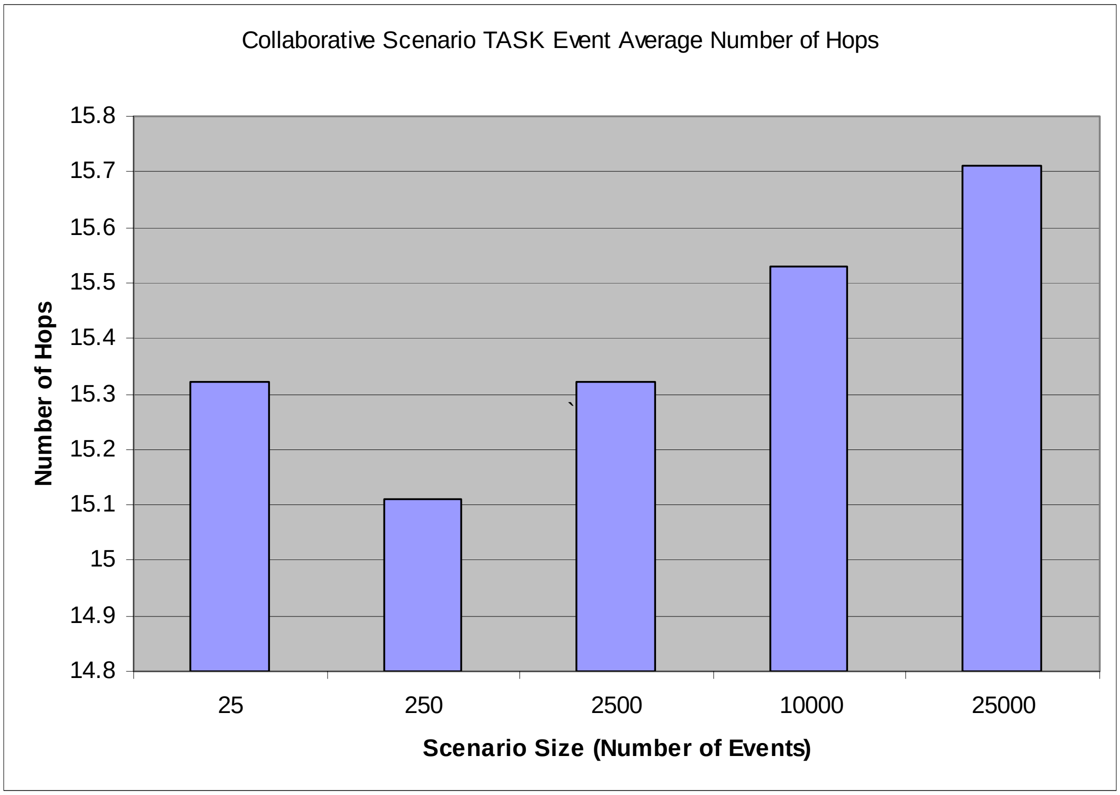


Figure 128 Collaborative Work Scenario Average Number of Hops

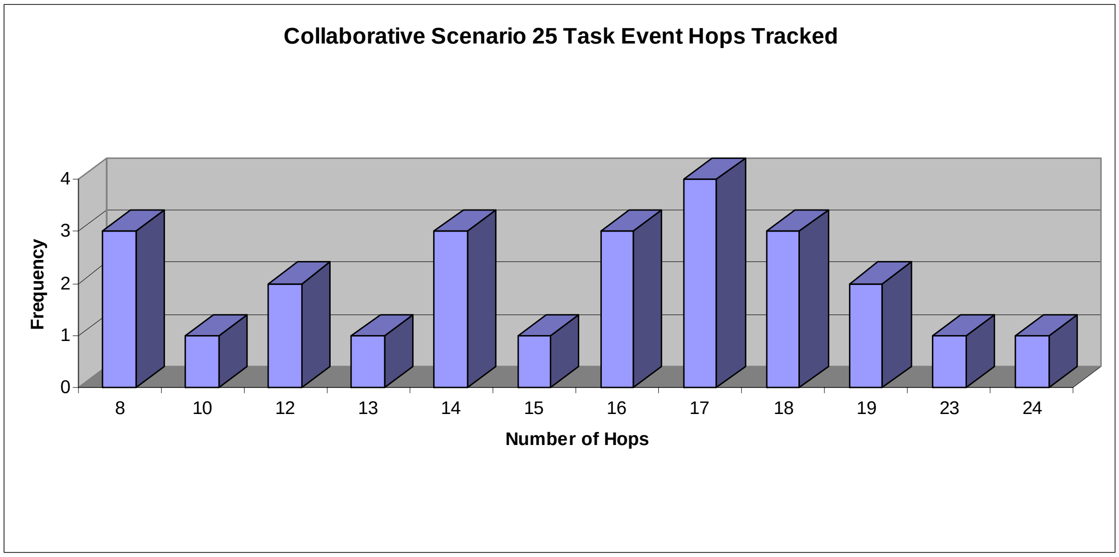


Figure 129 Collaborative Work Scenario Number of Hops for 25 Event Scenario

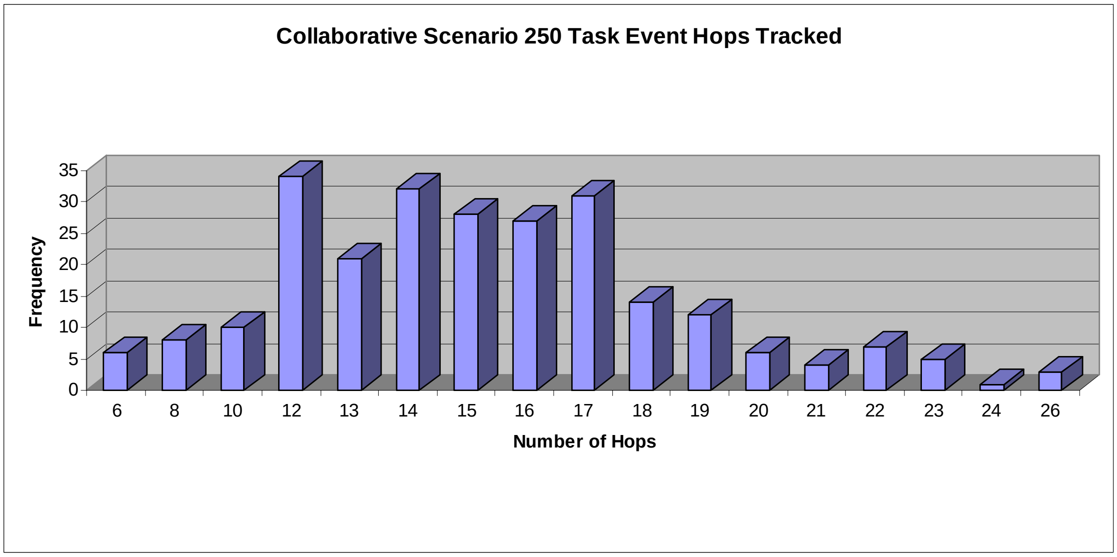


Figure 130 Collaborative Work Scenario Number of Hops for 250 Event Scenario

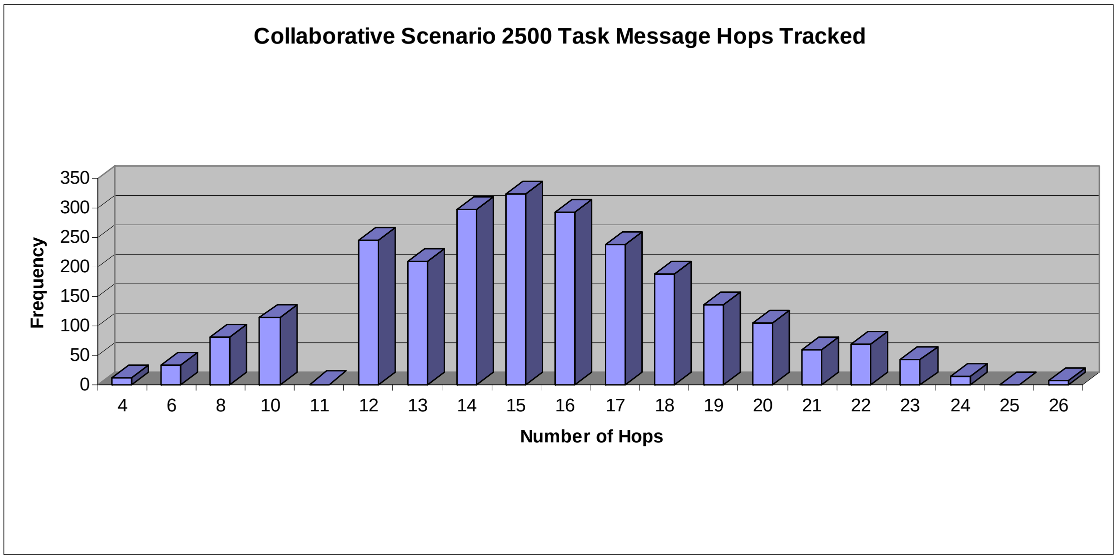


Figure 131 Collaborative Work Scenario Number of Hops for 2500 Event Scenario

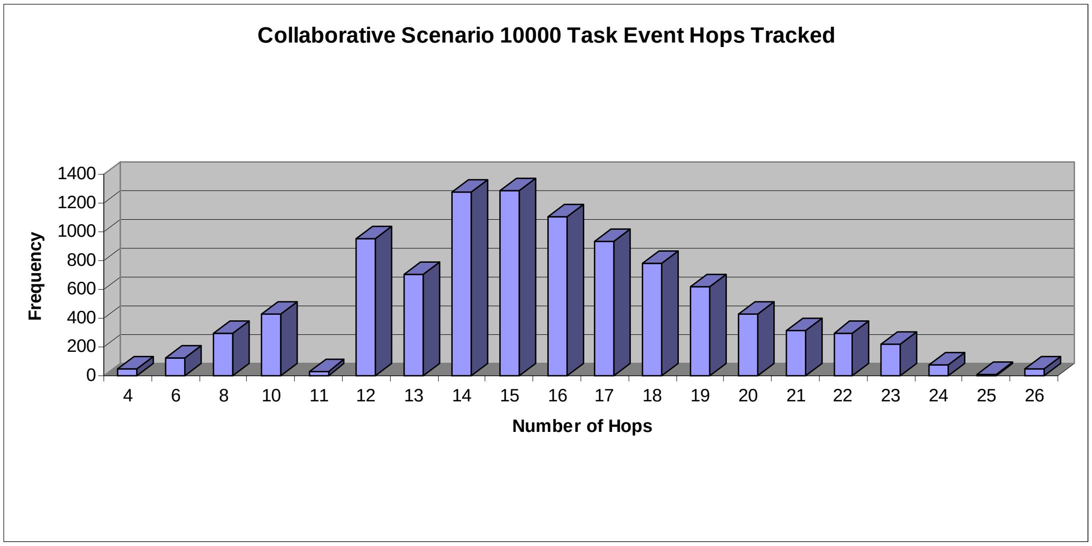


Figure 132 Collaborative Work Scenario Number of Hops for 10000 Event Scenario

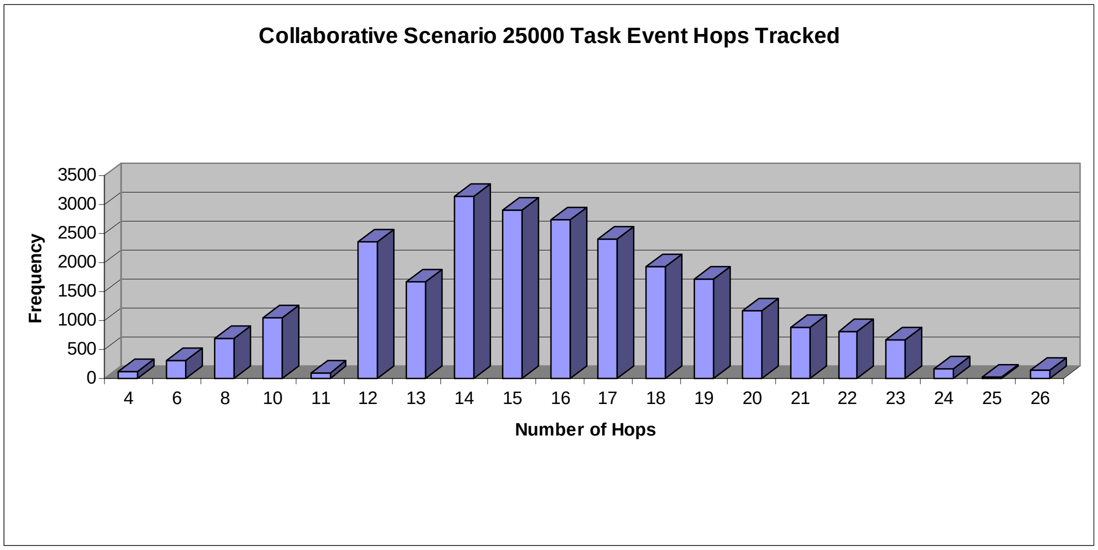


Figure 133 Collaborative Work Scenario Number of Hops for 25000 Event Scenario

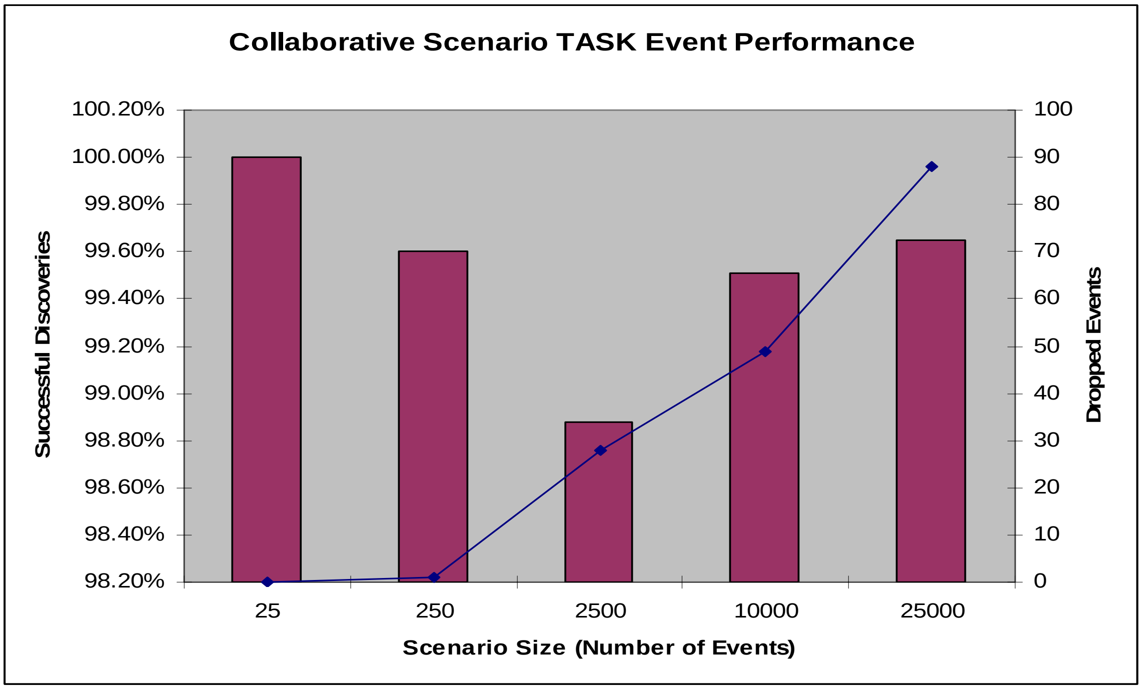


Figure 134 Collaborative Work Scenario Successful TASK Events

One important metric measures how successful the discovery approach was at finding a resource. Figure 134 shows the success rates of the TASK events finding an available resource. The values range between 98.88%-100% successful discoveries or 0-88 dropped packets. In the case of the unsuccessful TASK event not finding a resource, in the real world the VO host would simply try until it finds a resource. But, the simulator does not model this for the purposes of finding the success rates.

One new statistic provided in this research has to do with tracking how scores deviate from a perfect score. A perfect score does not deviate from the score of the resource, that score deviation value would be zero. With the case of collaborative work, the memory field is a “don’t care.” This means that the 8 bit score composed of CPU, memory, hard drive, and bandwidth would have a mask of 0xCF. This yields scores in the ranges of {0-15, 64-79, 128-143, 192-207} with a deviation of zero. Thus, since there are gaps of 48 between the score values, most of the score deviations should be between 0 and 48. Also considering the bit positions, one would expect scores to deviate around 0, 16, 32, and 48 depending when the resource score has a 0 or 1 in the spot of the don't care.

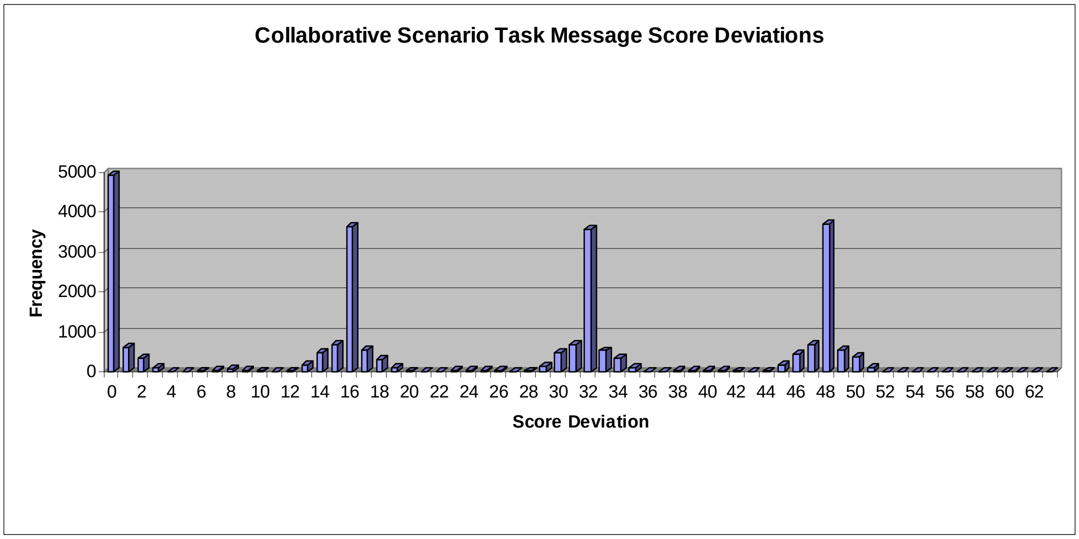


Figure 135 Collaborative Work Score Deviation for the 25,000 Event Scenario

Looking at Figure 135, the scores tend to deviate in that fashion. This figure represents the score deviations in the 25,000 event scenario. Approximately 63% of the scores fall exactly on 0, 16, 32, and 48 with the other scores tending to be very close to those numbers. When numbers deviate from the desired score, they deviate by an average of 24.11. Also, approximately 31% of the scores were less than 16 which explains why the 0 value is larger than the other three spikes.

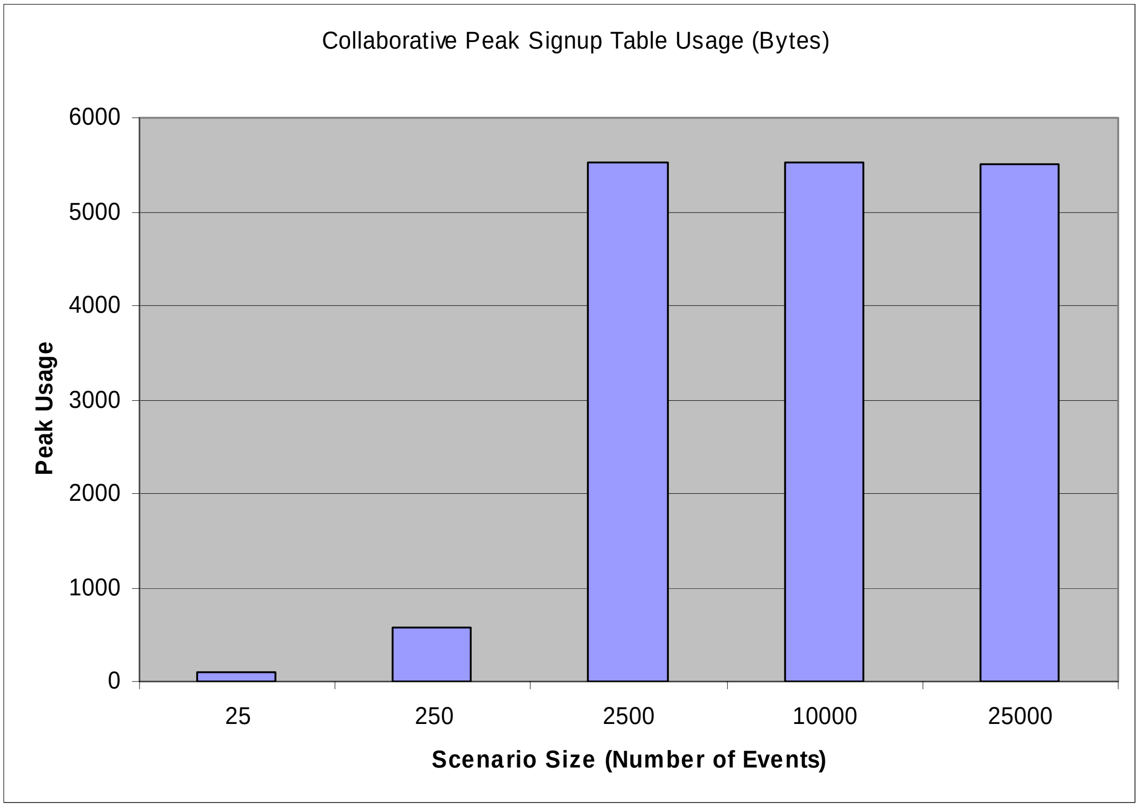


Figure 136 Collaborative Work Scenario Peak Signup Table Usage

The signup table peak memory usage is shown in Figure 136. As the number of events is increased, the memory usage caps at about 5500 bytes. This happens because devices are un-subscribing from the network as time is advancing which reduces the size of the signup table usage. By default, the resource providers unsubscribe from the VO in 200 simulation seconds after the CONFIRM DELIVERY event is sent. The signup table worst-case peak usage can be estimated if UNSUBSCRIBE are not sent. Since there are 25,000 events (worst case) from 25,000 different resource providers with a 4 byte address, 10 VOs with each with a 4 byte VO Host IP address, and 10 bytes per SIGNUP TABLE ENTRY, then the worst case peak memory consumption for any particular router is 450 KB or 25KB * (4 + 4 + 10). As a reminder, signup entries are removed every 24 hours to prevent uncontrolled growth of these tables.

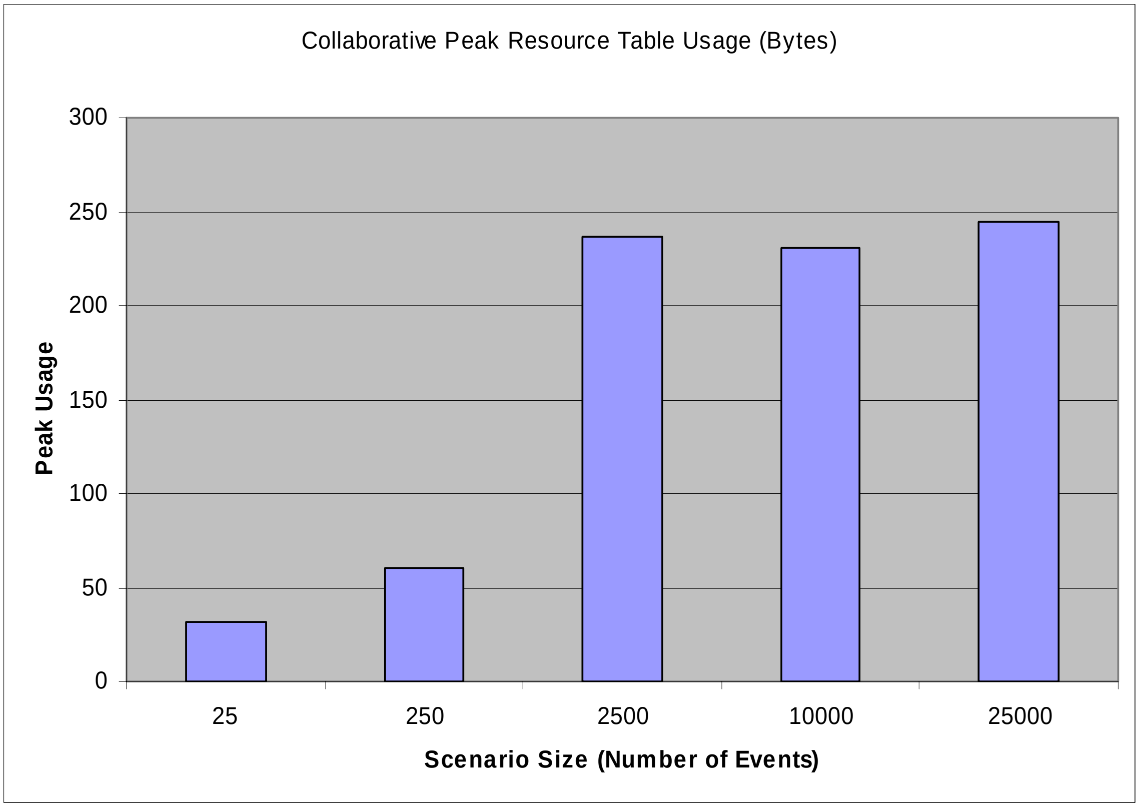


Figure 137 Collaborative Work Scenario Peak Resource Table Usage

Next, the peak resource table usage is examined in Figure 137. Notice that the usage caps at 245 bytes as the number of TASK events grows. This happens for different reasons than the signup table previously presented. The main reason is that the resource table has a smaller sized hash key and uses a one-byte score to lookup data. As the scenarios grow larger, once there are more than 256 resource providers, the scores will definitely overlap. The resource table is optimized to aggregate and count the number of devices with a particular score rather than to list individual resource providers. Also in this case, the simulation is greedy in discovering resources. Because resources are discovered in a greedy fashion, the table size does not grow very large because resources are consumed very quickly. Again the worst case tables size could be estimated by considering 25,000 resource IP addresses with 2 byte hash key, the score of one byte, the next hop IP of 4 bytes, and the count of one byte totaling 200KB or 25KB * (2 + 1 + 4 + 1).

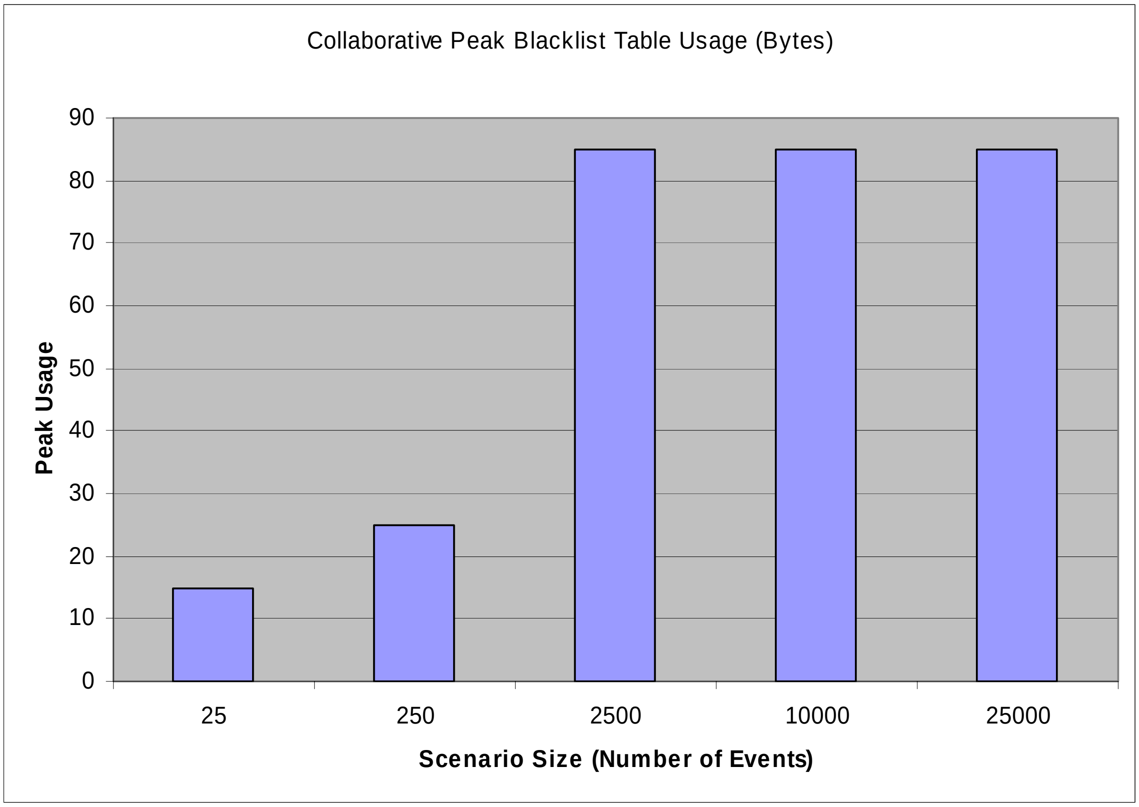


Figure 138 Collaborative Work Scenario Peak Blacklist Table Usage

The final results examined for the collaborative work scenarios are the blacklist tables. Since the blacklist tables are populated when the SIGNUP event is sent and unpopulated with the ACCEPT event is sent, the tables are much smaller than the others because the SIGNUP and ACCEPT events happen very close to each other in time. As shown in Figure 138, the memory usage caps at a value similarly to the other tables; this time around a value of 85 bytes. Doing the math, the BLACKLIST TABLE contains a four-byte IP address and a one-byte count totaling 5 bytes. Dividing 85 by 5 means that each router kept no more than 17 entries in its BLACKLIST TABLE at a given time.

The worst case blacklist size can be calculated as well by multiplying 25,000 resource providers times 5 bytes totaling 125KB. This would imply that all 25,000 resource providers send their SIGNUP events at the same time through the same router.

Deployment Environment Summary

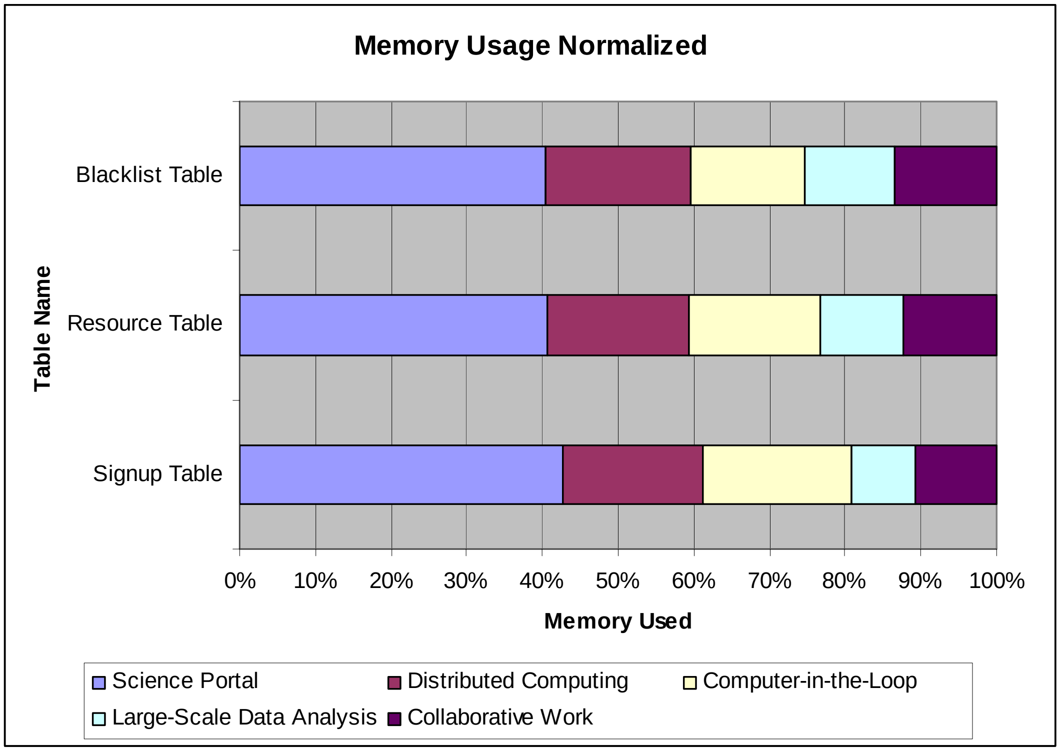


Figure 139 Memory Used Normalized

Five different deployment environments were modeled with 25, 250, 2500, 10,000, and 25,000 TASK messages sent from 10 VOs to many resource providers. Figure 139 shows the amount of memory used in each table for each of the five scenarios. The science portal scenario uses the most amount of memory per router where the large-scale scenario uses the least. Memory usage depends on the timing of the messages being sent, the length of time each task takes to process, and the overall size of the network. As evident in Figure 140, the science portal scenario had the least amount of computers where the large-scale analysis had the most. Since the each of the deployment scenarios was allowed to expand to the same maximum number of hops (network tree depth), this meant that the science portal had the thinnest tree (network tree width) whereas the large-scale data analysis had the widest tree. The wider the tree, the less of a chance that a router will have to store data in its tables. Note that the memory usage does not appear to be impacted by the average number of hops as shown in Figure 141.

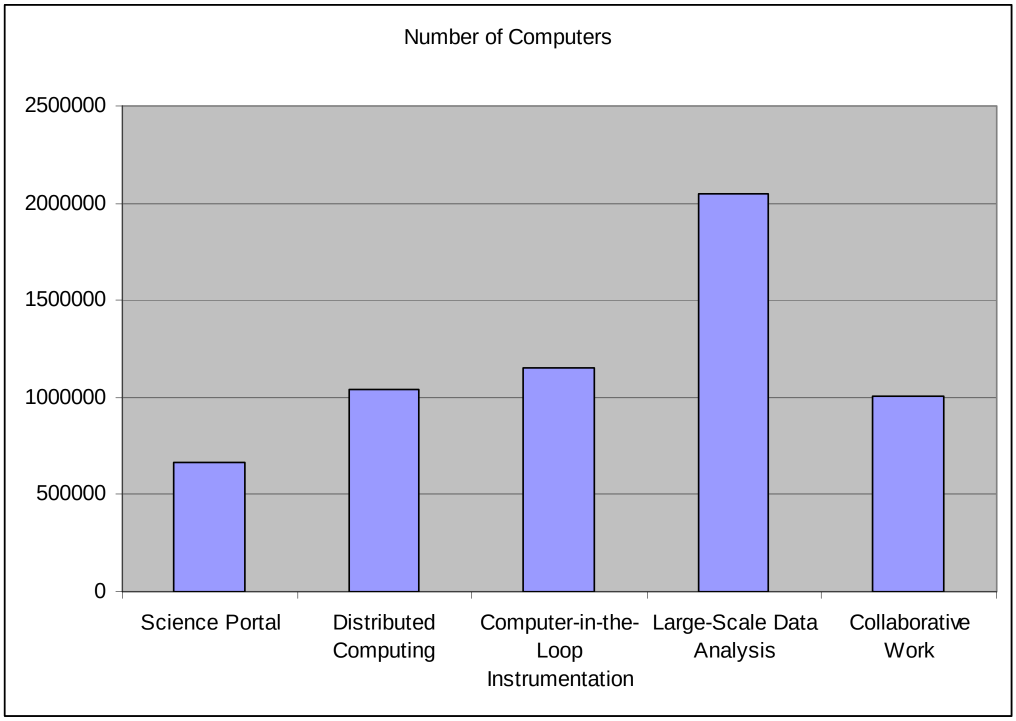


Figure 140 Number of Computers

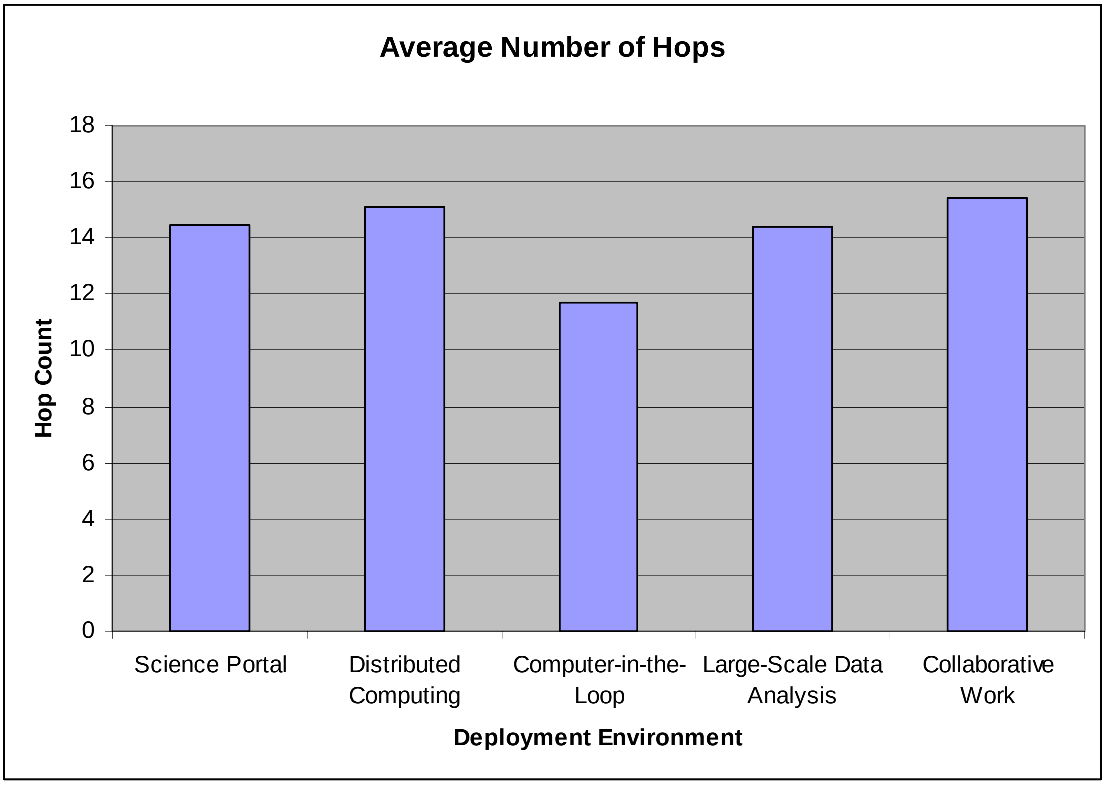


Figure 141 Average Number of Hops

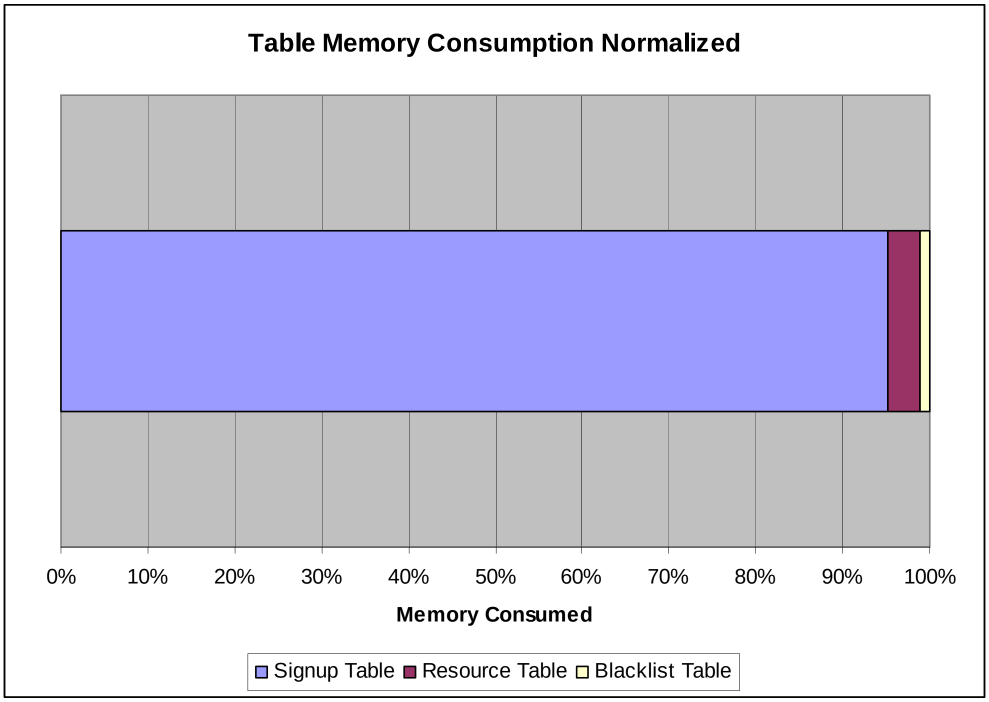


Figure 142 Table Memory Consumption Normalized

Clearly, the signup table consumes the most amount of memory in the simulation Figure 142. This happens because the signup table has more persistent entries lasting longer in the table than the other tables. The resource table is reduced quickly because VO hosts are aggressive when finding resource providers. The blacklist table is small because the entry size is much smaller than the other tables and the blacklist table is cleared quickly as well.

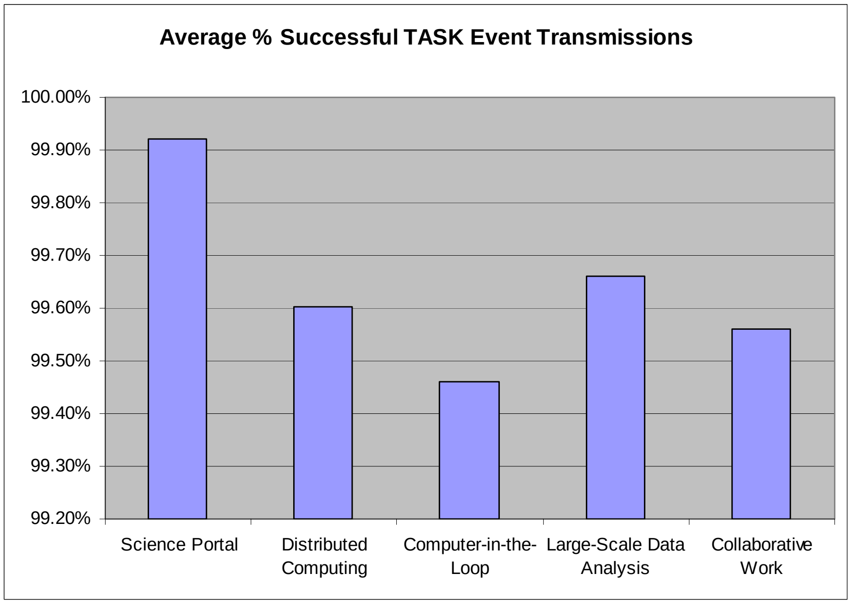


Figure 143 Average Successful TASK Event Transmissions

The averaged percentage of successful TASK (discovery) event transmissions is shown in Figure 143. The best performing scenario is the science portal (99.92%) where as the worst performing scenario is the computer-in-the-loop scenario (99.46%). The scoring does not appear to impact the performance of the discovery algorithm. This is suspected because the distributed computing scenarios are designed not to have a score deviation; and the distributed computing scenario ranks in the middle of the range. This is shown in Figure 144 because the distributed computing scenario's score deviation barely registers in the chart. The values do not correspond to the number of computers (Figure 140), the average number of hops (Figure 141), or the memory usage statistics (Figure 142) either. This means that the differences lie with the simulation, the scenario generation process, the timing of the messages relative to each other, and the distribution of the messages. Thus, the discovery process does not appear to be impacted by the network size, memory consumption, score deviation, or number of hops each message travels. The results indicate that the grid resource discovery algorithm will produce satisfactory results when deployed.

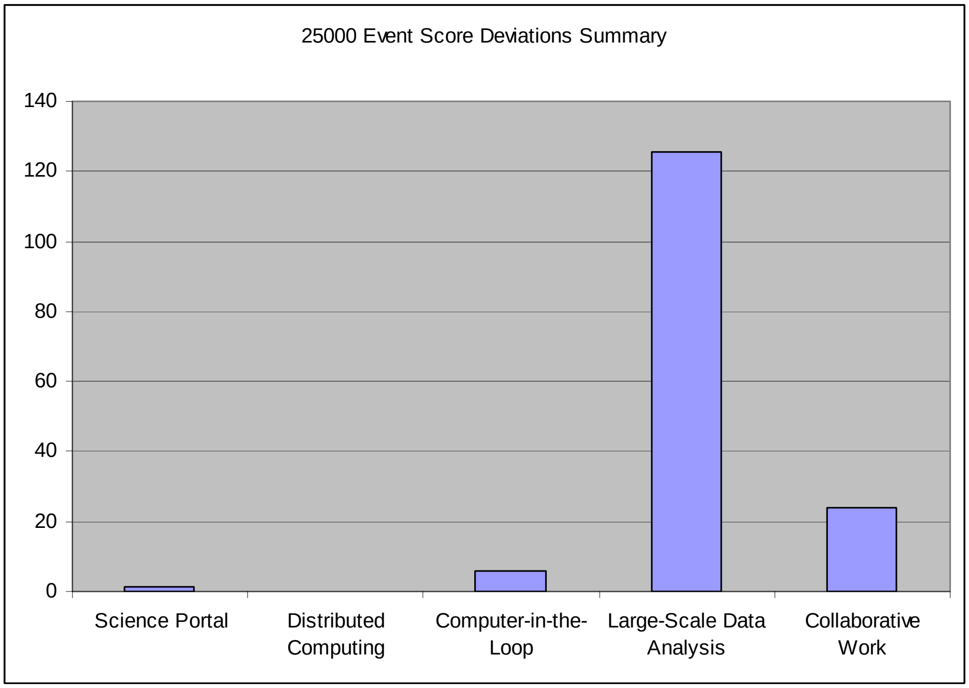


Figure 144 25000 Event Score Deviations Summary

HLA/RTI Evaluation

The final purpose of this work is to make the simulation engine perform basic HLA operations. By making the simulator HLA compatible, the workload can be divided between different federates to model the grid resource discovery protocol. All tests performed in this section had two federates: one federate managed the network traffic for one ISP (ATDN), the other federate managed the network traffic for the other ISP (GBLX). If an event needed to travel from one ISP to the other, it had to go through the RTI. This section summarizes the work done and how the results were verified.

RTI and Experimentation Hardware Information

There are several RTI implementations available on the market. The experimentation done for this work used RTI-NG Pro Version 3.0.2.3 available from Raytheon VTC. This version of the RTI implements HLA Version 1.3. The license management and configuration of the laptop used only allows operations from a 2 GHz single core laptop computer with 512 MB of memory. Under normal circumstances, the license server can be accessed from any remote machine. However this particular laptop configuration locked down the ability to do that.

Due to the limitations of the laptop hardware, large simulation executions could not be performed on this platform since a minimum of four processes were needed to include the RTIExec, two federates, and the license manager. Though the license manager and RTIExec are lightweight processes, the federate software is not. The execution time on this platform was slower than if the work could be distributed either on a multi-core machine or between different computers. Also, there were memory constraints as the scenario and network size grew larger. This limited the ability to test large and complex scenarios; thus the RTI experiments provide a proof of principle instead.

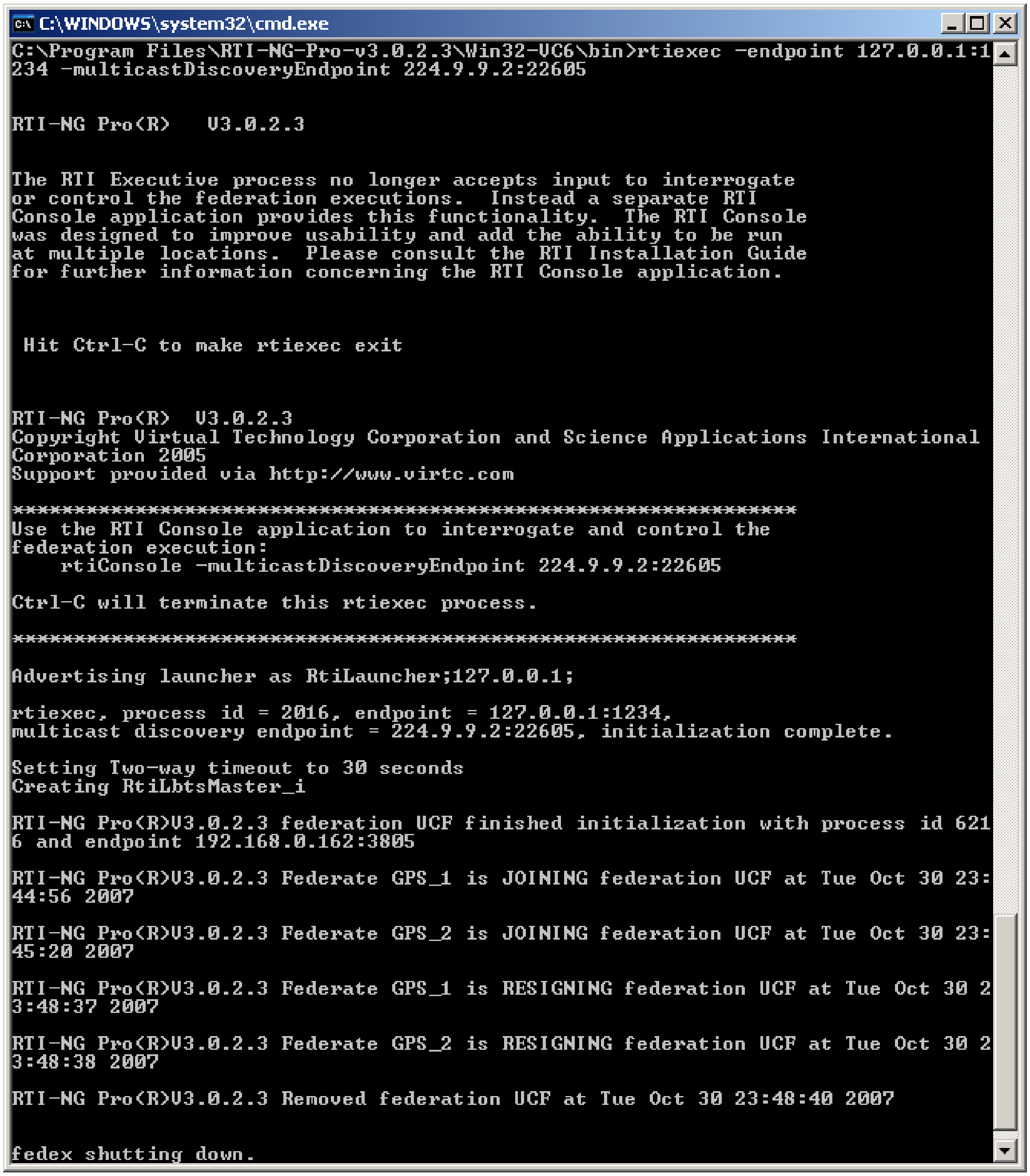
Results

Basic Federation and Federate Operation

The most important and basic operations for participating in an HLA federation are creating a federation, joining a federate to the federation, resigning the federate from the federation, and destroying the federation executable. The results of these actions can be verified by looking at the RTIExec screen which prints this basic information to the screen. In Figure 145, the federation name was “UCF” and the two federates were named “GPS_1” and “GPS_2”. The federation is created and the FOM format is verified when UCF was finished initializing about half way through the output screen. Next GPS_1 and GPS_2 have joined the federation. Time stepping is not shown on the display. When the federates were done modeling, they resigned from the federation. Finally, the federation was destroyed when the fedex was shutdown.

Event Management

There are several aspects of the simulation that have to do with event management. The first is declaring the ability to publish and subscribe to events. While doing this process, the software caches the event and parameter RTI handles needed for sending the events later. This functionality cannot be verified on the RTIExec console window. The RTI usually uses negative acknowledgements to let the user know something has gone wrong (rather than indicating something has gone right). The RTI does this by throwing exceptions. By examining the log outputs, there are no errors related to publications or subscriptions or for invalid FOM class name lookups.



```
C:\Program Files\RTI-NG-Pro-v3.0.2.3\Win32-VC6\bin>rtiexec -endpoint 127.0.0.1:1
234 -multicastDiscoveryEndpoint 224.9.9.2:22605


RTI-NG Pro(R)   V3.0.2.3


The RTI Executive process no longer accepts input to interrogate
or control the federation executions.  Instead a separate RTI
Console application provides this functionality.  The RTI Console
was designed to improve usability and add the ability to be run
at multiple locations.  Please consult the RTI Installation Guide
for further information concerning the RTI Console application.



 Hit Ctrl-C to make rtiexec exit



RTI-NG Pro(R)  V3.0.2.3
Copyright Virtual Technology Corporation and Science Applications International
Corporation 2005
Support provided via http://www.virtc.com

****************************************************************
Use the RTI Console application to interrogate and control the
federation execution:
    rtiConsole -multicastDiscoveryEndpoint 224.9.9.2:22605

Ctrl-C will terminate this rtiexec process.

****************************************************************

Advertising launcher as RtiLauncher;127.0.0.1;

rtiexec, process id = 2016, endpoint = 127.0.0.1:1234,
multicast discovery endpoint = 224.9.9.2:22605, initialization complete.

Setting Two-way timeout to 30 seconds
Creating RtiLbtsMaster_i

RTI-NG Pro(R)V3.0.2.3 federation UCF finished initialization with process id 621
6 and endpoint 192.168.0.162:3805

RTI-NG Pro(R)V3.0.2.3 Federate GPS_1 is JOINING federation UCF at Tue Oct 30 23:
44:56 2007

RTI-NG Pro(R)V3.0.2.3 Federate GPS_2 is JOINING federation UCF at Tue Oct 30 23:
45:20 2007

RTI-NG Pro(R)V3.0.2.3 Federate GPS_1 is RESIGNING federation UCF at Tue Oct 30 2
3:48:37 2007

RTI-NG Pro(R)V3.0.2.3 Federate GPS_2 is RESIGNING federation UCF at Tue Oct 30 2
3:48:38 2007

RTI-NG Pro(R)V3.0.2.3 Removed federation UCF at Tue Oct 30 23:48:40 2007


fedex shutting down.
```

Figure 145 RTIExec Output Window

Another aspect of event management is actually publishing or receiving an RTI event. This is verified by examining the output of the simulation on the GUI screen and in the log files. The GUI screen shows that event counts are incrementing. The log files indicate that some messages have been dropped on the local machine and transferred to the other federate. This can be traced by looking up the event id for that event. The event converted to text on the sending federate must match the event on the receiving federate when converted to text. 10 events out of 100 were manually checked to cross from one federate to the other and no problems were found. Also, since both federates are sending and receiving events, this also helps verify the publication and subscription task.

Synchronization Points

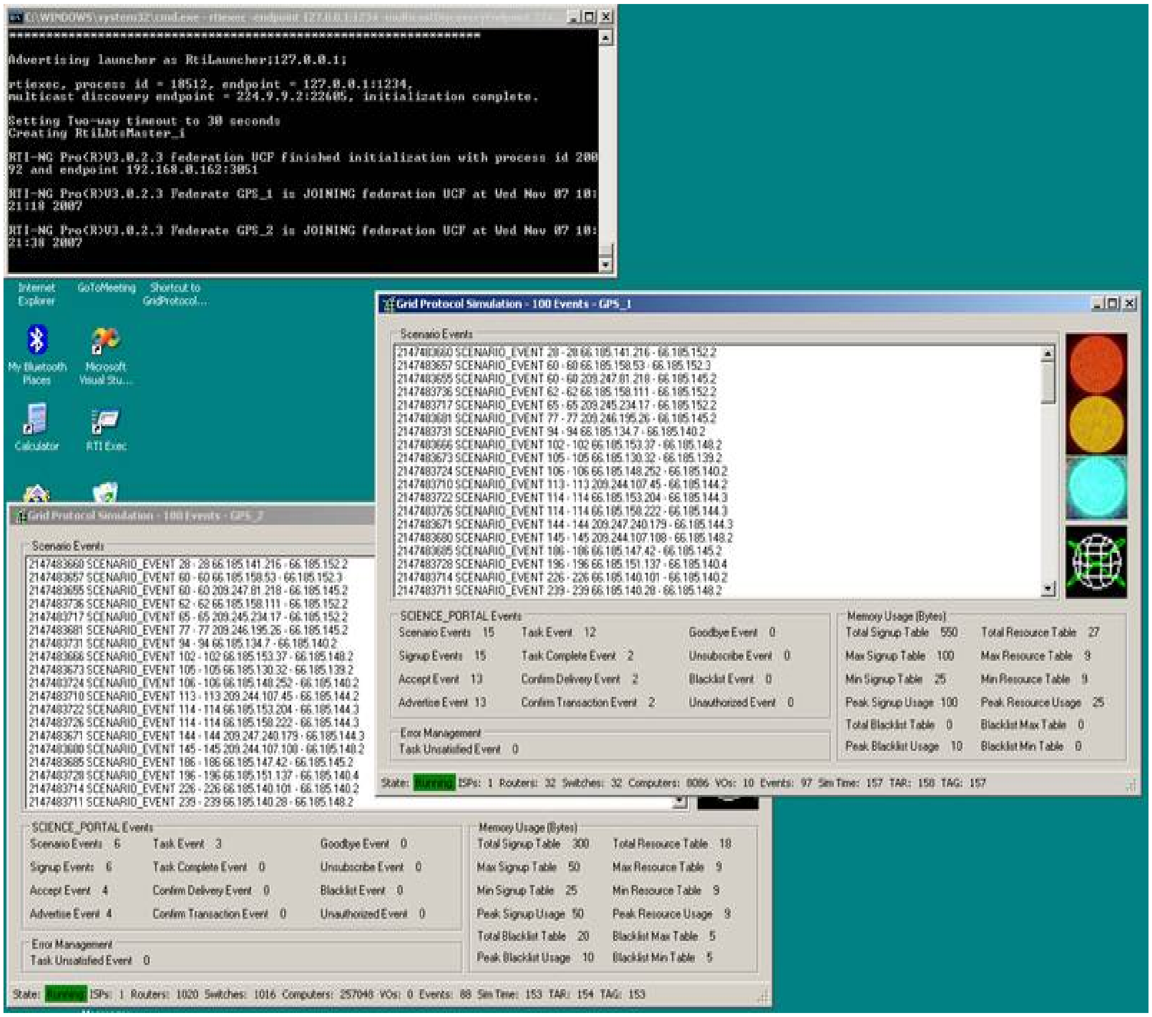
The grid protocol simulator uses two different synchronization points: a start synch point and a stop synch point. The start synch point is used to hold federates from starting the clock until the last federate joins. This is achieved by having the final federate register a synchronization point with the RTI, then having each of the federates accept the synch point announcement. Once the two federates accept the synch points, the RTI notifies the federates that the federation is synchronized. Upon receipt of this notification, the simulation clock is officially started.

Thus, the start synch point was verified in two ways. The first was the first federate to join the RTI sat and did not advance the clock until the second federate joined. The federate is actually sitting on a mutex that does not release until the federation is synchronized. The second verification came when the second federate joined, but federate clocks began advancing (as was evident on the GUI screen).

The stop synch point has a similar implementation, but for a different purpose. Even through the two federates are running concurrently, it is important that each federate stays in the federation until federation execution is complete. Thus, the federate will keep advancing time until the synch point stop is received. Once received, the federate resigns from the federation. This is verified by seeing that both federates resign from the federation at approximately the same time. If one federate resigns when its event queue is empty, this will be premature. In that case, one federate will resign and the other will continue to advance its clock until its event queue is emptied. During validation, both federates resigned at the same time.

Time Management

As mentioned above, time was advancing when the start and stop synch points were achieved. There is another way to verify proper time management. This simulation is time constrained and time regulating. It was hypothesized and observed that the federates try to catch up to each other's clocks. Thus, by watching the clocks on the two federates GUI screens, the racing was observed and verified.



```
C:\WINDOWS\system32\cmd.exe - rtiexec -endpoint 127.0.0.1:1234
************************************************************************

Advertising launcher as RtiLauncher:127.0.0.1;

rtiexec, process id = 18512, endpoint = 127.0.0.1:1234,
multicast discovery endpoint = 224.9.9.2:22605, initialization complete.

Setting Two-way timeout to 30 seconds
Creating RtiLbtsMaster_i

RTI-NG Pro(R)V3.0.2.3 federation UCF finished initialization with process id 200
92 and endpoint 192.168.0.162:3051

RTI-NG Pro(R)V3.0.2.3 Federate GPS_1 is JOINING federation UCF at Wed Nov 07 10:
21:18 2007

RTI-NG Pro(R)V3.0.2.3 Federate GPS_2 is JOINING federation UCF at Wed Nov 07 10:
21:38 2007
```

Grid Protocol Simulation - 100 Events - GPS_1

Scenario Events

```
2147483660 SCENARIO_EVENT 28 - 28 66.185.141.216 - 66.185.152.2
2147483657 SCENARIO_EVENT 60 - 60 66.185.158.53 - 66.185.152.3
2147483655 SCENARIO_EVENT 60 - 60 209.247.81.218 - 66.185.145.2
2147483736 SCENARIO_EVENT 62 - 62 66.185.158.111 - 66.185.152.2
2147483717 SCENARIO_EVENT 65 - 65 209.245.234.17 - 66.185.152.2
2147483681 SCENARIO_EVENT 77 - 77 209.246.195.26 - 66.185.145.2
2147483731 SCENARIO_EVENT 94 - 94 66.185.134.7 - 66.185.140.2
2147483666 SCENARIO_EVENT 102 - 102 66.185.153.37 - 66.185.148.2
2147483673 SCENARIO_EVENT 105 - 105 66.185.130.32 - 66.185.139.2
2147483724 SCENARIO_EVENT 106 - 106 66.185.148.252 - 66.185.140.2
2147483710 SCENARIO_EVENT 113 - 113 209.244.107.45 - 66.185.144.2
2147483722 SCENARIO_EVENT 114 - 114 66.185.153.204 - 66.185.144.3
2147483726 SCENARIO_EVENT 114 - 114 66.185.158.222 - 66.185.144.3
2147483671 SCENARIO_EVENT 144 - 144 209.247.240.179 - 66.185.144.3
2147483680 SCENARIO_EVENT 145 - 145 209.244.107.108 - 66.185.148.2
2147483685 SCENARIO_EVENT 186 - 186 66.185.147.42 - 66.185.145.2
2147483728 SCENARIO_EVENT 196 - 196 66.185.151.137 - 66.185.140.4
2147483714 SCENARIO_EVENT 226 - 226 66.185.140.101 - 66.185.140.2
2147483711 SCENARIO_EVENT 239 - 239 66.185.140.28 - 66.185.148.2
```

SCIENCE_PORTAL Events

Scenario Events	15	Task Event	12	Goodbye Event	0
Signup Events	15	Task Complete Event	2	Unsubscribe Event	0
Accept Event	13	Confirm Delivery Event	2	Blacklist Event	0
Advertise Event	13	Confirm Transaction Event	2	Unauthorized Event	0

Error Management

Task Unsatisfied Event 0

Memory Usage (Bytes)

Total Signup Table	550	Total Resource Table	27
Max Signup Table	100	Max Resource Table	9
Min Signup Table	25	Min Resource Table	9
Peak Signup Usage	100	Peak Resource Usage	25
Total Blacklist Table	0	Blacklist Max Table	0
Peak Blacklist Usage	10	Blacklist Min Table	0

State: Running ISPs: 1 Routers: 32 Switches: 32 Computers: 8086 VOs: 10 Events: 97 Sim Time: 157 TAR: 158 TAG: 157

Grid Protocol Simulation - 100 Events - GPS_2

Scenario Events

```
2147483660 SCENARIO_EVENT 28 - 28 66.185.141.216 - 66.185.152.2
2147483657 SCENARIO_EVENT 60 - 60 66.185.158.53 - 66.185.152.3
2147483655 SCENARIO_EVENT 60 - 60 209.247.81.218 - 66.185.145.2
2147483736 SCENARIO_EVENT 62 - 62 66.185.158.111 - 66.185.152.2
2147483717 SCENARIO_EVENT 65 - 65 209.245.234.17 - 66.185.152.2
2147483681 SCENARIO_EVENT 77 - 77 209.246.195.26 - 66.185.145.2
2147483731 SCENARIO_EVENT 94 - 94 66.185.134.7 - 66.185.140.2
2147483666 SCENARIO_EVENT 102 - 102 66.185.153.37 - 66.185.148.2
2147483673 SCENARIO_EVENT 105 - 105 66.185.130.32 - 66.185.139.2
2147483724 SCENARIO_EVENT 106 - 106 66.185.148.252 - 66.185.140.2
2147483710 SCENARIO_EVENT 113 - 113 209.244.107.45 - 66.185.144.2
2147483722 SCENARIO_EVENT 114 - 114 66.185.153.204 - 66.185.144.3
2147483726 SCENARIO_EVENT 114 - 114 66.185.158.222 - 66.185.144.3
2147483671 SCENARIO_EVENT 144 - 144 209.247.240.179 - 66.185.144.3
2147483680 SCENARIO_EVENT 145 - 145 209.244.107.108 - 66.185.148.2
2147483685 SCENARIO_EVENT 186 - 186 66.185.147.42 - 66.185.145.2
2147483728 SCENARIO_EVENT 196 - 196 66.185.151.137 - 66.185.140.4
2147483714 SCENARIO_EVENT 226 - 226 66.185.140.101 - 66.185.140.2
2147483711 SCENARIO_EVENT 239 - 239 66.185.140.28 - 66.185.148.2
```

SCIENCE_PORTAL Events

Scenario Events	6	Task Event	3	Goodbye Event	0
Signup Events	6	Task Complete Event	0	Unsubscribe Event	0
Accept Event	4	Confirm Delivery Event	0	Blacklist Event	0
Advertise Event	4	Confirm Transaction Event	0	Unauthorized Event	0

Error Management

Task Unsatisfied Event 0

Memory Usage (Bytes)

Total Signup Table	300	Total Resource Table	18
Max Signup Table	50	Max Resource Table	9
Min Signup Table	25	Min Resource Table	9
Peak Signup Usage	50	Peak Resource Usage	9
Total Blacklist Table	20	Blacklist Max Table	5
Peak Blacklist Usage	10	Blacklist Min Table	5

State: Running ISPs: 1 Routers: 1020 Switches: 1016 Computers: 257040 VOs: 0 Events: 88 Sim Time: 153 TAR: 154 TAG: 153

Figure 146 Two Federates Running a Scenario with the RTI

CHAPTER FIVE: CONCLUSIONS

K-Array N-Cube Design Conclusion

An event-driven, custom-designed interconnect simulation environment was created to evaluate the performance of off-chip k-array n-cube interconnect architectures for line cards. The interconnects were examined using the network simulator in order to find which of the interconnects can provide the highest performance and memory bandwidth to replace the existing shared-bus systems.

The simulator provides the user with a flexible and robust tool that can emulate multiple interconnect architectures under non-uniform traffic patterns. The simulator offers the user with extensive control over network parameters, performance enhancing features and simulation time frames that make the platform as close as possible to the physical line card features.

Performance results show that k-array n-cube topologies can sustain higher traffic load than the currently used interconnects. Flow control mechanisms such as virtual channels (VC) and sub-channeling (SC) have an important impact on the interconnect performance. VC and SC mechanisms, together, reduce the transmission failure rate significantly by 75% and increase the interconnect bandwidth utilization in the range of 15–25% depending on the topology. A variation of 2-array 3-cube, called 3D-mesh, was introduced that provides a better processor-memory distribution under non-uniform traffic. The combination of the 3D-mesh interconnect and the adaptive routing algorithm facilitate to reach the highest throughput of 452 Gbps; this is better than twice the throughput of the leading solution in the marketplace. 3D-mesh meets both the stringent performance requirements and the physical constraints on the line card while enabling future scalability to adopt higher line rates.

CLL Algorithm Conclusion

A new cluster leader election algorithm called the cluster leader logic (CLL) algorithm was proposed and simulated. GPS-QHRA is based on the presence of a GPS device with the networking node. The cluster leaders react to data flow patterns of the network by providing better load balancing throughout the wireless GPS-based ad-hoc network by sharing their load. Based on the geographical direction of the net traffic flow, the clusterheads are selected in such a manner that there are more clusterheads at locations where there is more traffic activity.

Thus the clusterheads are able to share the load for forwarding packets. At locations of lower or no traffic flow, there are less numbers of clusterheads since clusterhead overloading is not a problem. The clusterheads can filter data sent based on the ground and perceived truth knowledge of the network and by introducing a new concept called cell fanning. Cell fanning allows a clusterhead to split into two clusterheads preventing the original clusterhead from becoming overloaded and the new clusterhead becoming starved for data transmissions.

Extensive simulation experiments were conducted to demonstrate that the system performance is enhanced when the proposed algorithm chooses clusterheads. The simulator was built on top of the simulation infrastructure used in the k-array n-cube simulator. The results show up to 45% power savings and up to 25% improvement in queuing delays when CLL is compared to GPS-QHRA.

Grid Resource Protocol Conclusion

The Grid Protocol Simulator, the third simulator in this work, simulated five different deployment environments with 25, 250, 2500, 10,000, and 25,000 TASK messages sent from 10 VOs to many resource providers. The five different environments varied the application of the scoring mechanism used to route the TASK messages through the network. Hop counts, memory usage, message distribution, discovery message successes, and score deviation statistics were collected and presented in this work.

The science portal scenario uses the most amount of memory per router where the large-scale scenario uses the least. Memory usage depends on the timing of the messages being sent, the length of time each task takes to process, and the overall size of the network. The science portal scenario had the least amount of computers where the large-scale analysis had the most.

The signup table consumes the most amount of memory in the simulation. This happens because the signup table has more persistent entries lasting longer in the table than the other tables. The resource table is reduced quickly because VO hosts are aggressive when finding resource providers. The blacklist table is small because the entry size is much smaller than the other tables and the blacklist table is cleared quickly as well. Also, worst-case memory consumption was calculated in the results section. The signup table worst-case memory consumption per router is 450KB, the resource table is 200KB, and the blacklist is 125KB totaling 775 KB per router for 25,000 resource providers mapped to 10 VO Hosts.

The best performing scenario with respect to successful discovery message transmissions is the science portal scenario (99.96%) where as the worst performing scenario is the computer-in-the-loop scenario (99.43%). The scoring does not impact the performance of the discovery algorithm. The discovery process does not appear to be impacted by the network size, memory consumption, score deviation, or number of hops each message travels.

Simulation Engine Conclusion

The main purpose of this work is to model and simulate networking architectures and protocols by developing a common underlying simulation infrastructure. All three simulators kept the same overall architecture: creating scenarios, feeding them into an event-driven simulation, and getting results at the end. The scenario generation process evolved into the generation of XML-based text files to represent networks and event. The simulator evolved to support HLA/RTI which is a primary simulation architecture in the present time. The results generation has evolved into the software automatically producing multi-worksheet spreadsheets with sorted and formatted data, formulas, charts, and graphs. In conclusion, the simulation engine supplies reusable modules at a minimum if not an entire infrastructure that can be built from or expanded.

The sim engine allows the developer to perform basic HLA functions such as time-constrained time-regulating time management, the creation, sending, and receiving of RTI events, and synch point management. The simulation engine is configured through the use of a GUI control form and the results are stored in the RtiManager class.

The developer can create and join a federation, subscribe to interactions, and designate which FED FOM file to use.

In addition to performing those functions with the RTI, the sim engine supports a mode where the RTI is not present. This is configurable at run time rather than compile time thus allowing the developer to support one executable software delivery. The developer inherits base classes to perform the duties required. The sim engine also allows the developer to reuse the capability to represent a network by reading in XML scenario files. The sim engine also provides auxiliary functionalities such as logging and error reporting, an IP V4 address container, and random number generator. The sim engine also provides graphical interfaces for asking the user questions or displaying an error message GUI.

Future Directions for this Work

Even though a considerable amount of work was done to conclude this work, there are still enhancements and improvements that can be made which are beyond the scope of this work. For the k-array n-cube wormhole routing protocol, a good continuation would be to attempt to emulate the protocol in hardware. Results can be gathered to compare the simulated results to the emulated results.

For the CLL algorithm, it would be beneficial to find more scenarios to simulate; similar to researching and representing the five grid deployment environments done for the grid discovery protocol. Possible places to look for deployment environments are military live training ranges such as 29 Palms [110]. Once these ad-hoc wireless deployments are identified, scenarios can be generated to represent the terrain, situation, and node characteristics and then simulated.

As for the grid discovery protocol, a very useful experiment would be to implement the algorithm either in programmable routers (such as [111]), hardware, or computers simulating routers by directing network traffic like the protocol would. A lab would be needed with enough devices to represent a reasonably sized network to test on. A different continuation of work would be to study the GLOBUS architecture [65][79][80] to see how the grid discovery protocol can fit into it. This would require a possible replacement of the GLOBUS broker services, GIS, MDS, GRAM, and scheduler.

The simulation engine can be evolved further to increase the HLA capabilities. One improvement would be for the simulation engine to support any type of time management (various combinations of time regulating and time constraining). The event interface can be cleaned up to encapsulate the ability for directly calling the RTI functions. For instance, the Event class requires the developer to create RTI handle value pairs and call the sendInteraction() function. A more elegant design would be for the developer to serialize the data in FOM order into memory and hand the block of memory to a class that would perform the responsibilities of converting the memory into RTI data and function calls. Another goal would be to implement HLA objects and save/restore functionality. The grid protocol simulator does not own any objects and the functionality to create them or to have a save/restore capability was never needed or developed.

LIST OF REFERENCES

[1] Reilly, Sean and Keith Briggs. "Guidance, Rationale, and Interoperability Modalities for the Real-time Platform Reference Federation Object Model (RPR FOM)." *Simulation Interoperability Standards Organization*, September 1999.

[2] Weatherly, Richard M., Annette L. Wilson, Bradford S. Canova, Ernest H. Page, Anita A. Zabek. "Advanced Distributed Simulation through the Aggregate Level Simulation Protocol." *Proceedings of the 29th International Conference on System Sciences*. Vol. 1, pp. 407-415, Wailea, Hawaii, 3-6 January 1996, [*Online*]. Available WWW: http://www.thesimguy.com/ernie/papers/hicss-29/camera.html.

[3] Corps Battle System. [*Online*]. Available WWW: http://www.peostri.army.mil/PRODUCTS/CBS/.

[4] Intelligence Electronic Warfare Tactical Proficiency Trainer. [*Online*]. Available WWW: http://www.peostri.army.mil/PRODUCTS/IEWTPT/.

[5] One Semi-Automated Forces. [*Online*]. Available WWW: http://www.peostri.army.mil/PRODUCTS/ONESAF/.

[6] Tactical Simulation. [*Online*]. Available WWW: http://www.peostri.army.mil/PRODUCTS/TACSIM/.

[7] Warfighters' Simulation. [*Online*]. Available WWW: http://www.peostri.army.mil/PRODUCTS/WARSIM/.

[8] Seidel, D. (1993). ``Aggregate Level Simulation Protocol (ALSP) Program Status and History," *The MITRE Corporation*, McLean, VA, 22102, March.

[9] Deitel, Harvey and Paul Deitel. "C++ How to Program." Prentice Hall, 5th Edition January 5, 2005.

[10] "RTI NG Pro Programmer's Guide." *Virtual Technology Corporation*, version 3.0, Jume 2005.

[11] "MODELING AND SIMULATION (M&S) MASTER PLAN." *United States of America Department of Defense*, DoD 5000.59-P, October 2005. [*Online*]. Available WWW: https://www.dmso.mil/public/library/policy/guidance/500059p.pdf.

[12] "IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) - Object Model Template (OMT) Specification." *IEEE* 1516.2-2000.

[13] OMDT Pro. [*Online*]. Available WWW: http://www.aegistg.com/labcut/lwProducts/productsheets/OMDTPro101.pdf.

[14] "High-Level Architecture Object Model Template Specification Version 1.3." *U.S. Department of Defense*, 5 February 1998.

[15] "IEEE Standard for Distributed Interactive Simulation - Application Protocols," *IEEE* Std 1278.1-1995.

[16] "IEEE Standard for Distributed Interactive Simulation - Application Protocols," *IEEE* Std 1278.1A-1998.

[17] Geocentric Coordinate Systems. [*Online*]. Available WWW: http://en.wikipedia.org/wiki/Geocentric_coordinates.

[18] "Draft, Recommended Practice for Distributed Interactive Simulation Exercise Management and Feedback," *IEEE* Std P1278.3.

[19] Extended Air Defense Simulation. [*Online*]. Available WWW: http://www.eadsim.com/.

[20] FireSim XXI. [*Online*]. Available WWW: http://sill-www.army.mil/blab/sims/FireSimXXI.htm.

[21] Tactical Simulation Interface Unit. [*Online*]. Available WWW: http://www.aegistg.com/dbst/index.htm.

[22] High Level Architecture. [*Online*]. Available WWW: http://www.mitre.org/news/the_edge/january_98/fourth.html.

[23] The Network Simulator 2. [*Online*]. Available WWW: http://www.isi.edu/nsnam/ns/.

[24] OSI Model. [*Online*]. Available WWW: http://en.wikipedia.org/wiki/OSI_model.

[25] Engel, Jacob, Daniel Lacks, Taskin Kocak. "Modelling and simulation of off-chip communication architectures for high-speed packet processors." *The Journal of Systems and Software*, Volume 79, May 6, 2006: 1701-1714.

[26] Kumar, R., Zyuban, V., Tullsen, D.M., 2005. Interconnections in multicore architectures: understanding mechanisms, overheads and scaling. *Proceedings of the IEEE 32nd International Symposium on Computer Architecture*, June 2005.

[27] Test Procedures, March 5, 2001. [*Online*]. Available WWW: http://www.lightreading.com

[28] Dally, W.J., 1990. Performance analysis of k-ary n-cube interconnection networks. *IEEE Transactions on Computers* 39 (6), 775–785.

[29] Agarwal, A., 1991. Limits on interconnection network performance. *IEEE Transactions on Parallel and Distributed Systems,* 2 (4), 398–412.

[30] One Tactical Engagement Simulation System. [*Online*]. Available WWW: http://www.peostri.army.mil/PRODUCTS/ONETESS/.

[31] Ogier, Richard. "A Simulation Comparison of TBRPF and AODV." *SRI International*, December, 2001. [*Online*]. Available WWW: http://www.sri.com/esd/projects/tbrpf/docs/NS2-Sim_Com.ppt.

[32] Jung, J.W., Mudumbai, R., Montgomery, D., Kahng, H.K., 2003. "Performance evaluation of two layered mobility management using mobile IP and session initiation protocol." *Proceedings of the Global Telecommunications Conference*, Vol. 3, pp. 1190–1194.

[33] Kornblit, R., Schwartzmann, E. "Multicast Protocols Evaluation in Wireless Domains, Project report." *Technion*, Israel, 2004.

[34] Qualnet. [*Online*]. Available WWW: http://www.scalable-networks.com/products/developer/new_in_40.php.

[35] Hsu, J., Bhatia, S., Takai, M., Bagrodia, R., Acriche, M.J., 2003. "Performance of mobile ad hoc networking routing protocols in realistic scenarios." *Proceedings of the Military Communications Conference*, vol. 2, pp. 1268–1273.

[36] Zhang, Y., 2003. "Microstrip-multilayer delay line on printed-circuit board." *Technical Report, University of Nebraska, Lincoln*, April 2003.

[37] Chiu, G.M., 2000. "The odd–even turn model for adaptive routing." *IEEE Transactions on Parallel and Distributed Systems* 11 (7), 729–738.

[38] Dally, W.J., 1992. "Virtual-channel flow control." *IEEE Transactions on Parallel and Distributed Systems* 3 (2), 194–199.

[39] Lysne, O., 1999. "Deadlock avoidance for switches based on wormhole networks." *Proceedings of the Annual International Conference of Parallel Processing*, pp. 68–74.

[40] Cope, M.C., 2005. "Object Oriented Analysis and Design Using UML." White paper, *Ratio group*. [*Online*]. Available WWW: http://www.ratio.co.uk/white.html.

[41] Nakata, T., Kuwamura, S., Zhu, Q., Matsuda, A., Shoji, M., 2002. "An object-oriented design process for system-on-chip using UML." *Proceedings of the 15th international symposium on System Synthesis*, pp. 249–254.

[42] Schach, S.R., 1996. "Classical and Object-Oriented Software Engineering." Third ed., Irwin group.

[43] Fritz, D.G., Sargent, R.G., 1995. "An overview of hierarchical control flow graph models." *Proceedings of the IEEE Simulation Conference*, pp. 1347–1355.

[44] Townsend, M., 2002. "The Singleton Design Pattern." Microsoft Corporation, *MSDN library*, February 2002. [*Online*]. Available WWW: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnbda/html/singletondespatt.asp.

[45] "Pure Virtual Functions and Abstract Classes." Microsoft Corporation, *MSDN library*, 2005. [*Online*]. Available WWW: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/vccore98/html/_langref_pure_virtual_functions_and_abstract_classes.asp.)

[46] "Introduction to the Standard Template Library." *SGI*, white paper, 2003. [*Online*]. Available WWW: http://www.sgi.com/tech/stl/stl_introduction.html.

[47] Meyers, S. “Effective STL: 50 Specific Ways to Improve Your Use of the Standard Template Library.” Addison-Wesley, Boston, Mass. 2001.

[48] E. M. Royer and C.-K. Toh. ``A Review of Current Routing Protocols for Ad-Hoc Mobile Wireless Networks." {\it IEEE Personal Communications Magazine}, April 1999, pp. 46-55.

[49] I. Hwang, C. Chien, and C. Wang, ``A Novel GPS-Based Quorum Hybrid Routing Algorithm (GPS-QHRA) for Cellular-Based Ad Hoc Wireless Networks", {\em Journal of Information and Science Engineering}, Vol. 21, pp. 1-21, 2005.

[50] Lee, Keun-Ho, Han, Sang-Bum, Suh, Heyi-Sook, Lee, SangKeun, Hwang, Chong-Sun. “Authentication based on multilayer clustering in ad hoc networks.” EURASIP Journal on Wireless Communications and Networking, Dec 15, 2005 i5 p731(12).

[51] Vasudevan, Sudarshan, Kurose, Jim, Towsley, Don. “Design and Analysis of a Leader Election Algorithm for Mobile Ad Hoc Networks.” UMass Computer Science Techincal Report 03-20.

[52] Elizabeth M. Royer and C.-K. Toh. "A Review of Current Routing Protocols for Ad-Hoc Mobile Wireless Networks." IEEE Personal Communications Magazine, April 1999, pp. 46-55.

[53] Burgstahler, Lars, Neubauer, Martin. “New modifications of the exponential moving average algorithm for bandwidth estimation.” Proceedings of the 15th ITC Specialist Seminar on Internet Traffic Engineering and Traffic Management, Würzburg, 2002, pp. 210-219. [*Online*]. Available WWW: http://www.ikr.uni-stuttgart.de/en/Content/Publications/

[54] L. Harte, R. Levine, R. Kikta. ``3G Wireless Demystified." McGraw-Hill, 2002, pp. 39-40.

[55] K. Fall and K. Varadhan. ``The ns Manual." January 20, 2007. *UC Berkeley, LBL, USC/ISI, and Xerox PARC.* [*Online*]. Available WWW: http://www.isi.edu/nsnam/ns/doc/ns_doc.pdf.

[56] Lacks, Daniel, and Taskin Kocak. "A Distributed Grid Resource Discovery and Management Protocol and Its Deployment Environments." GCA'06 - The 2006 International Conference on Grid Computing and Applications, June 2006, pp. 11-17.

[57] SETI@Home. [*Online*]. Available WWW: http://setiathome.ssl.berkeley.edu/.

[58] Einstein@Home. [*Online*]. Available WWW: http://einstein.phys.uwm.edu/.

[59] Buyya, Rajkumar. "Economic-based Distributed ResourceManagement and Scheduling for Grid Computing." *School of Computer Science and Software Engineering of Monash University*, 2002.

[60] Bernatz, John C. COL (Ret.), Shockley, John. "A Funny Thing Happened on the Way to an LVC Integration: Great Training." *SRI International*, 2004. [*Online*]. Available WWW: http://www.jtepforguard.com/pubs/04E-SIW-065.pdf.

[61] T. Kocak and L. Boloni, "Highly distributed resource discovery and allocation in the grid", Proc. of the 47th IEEE Midwest Symp. on Circuits and Systems, Hiroshima, Japan, July 2004.

[62] Joseph, J., Ernest, M., Fellenstein, C. "Evolution of grid computing architecture and grid adoption models." *IBM Systems Journal*, Dec 2004 v43.

[63] Navy Organization. [*Online*]. Available WWW: http://www.navy.mil/navydata/organization/org-over.asp.

[64] M. Riedel, V. Sander, P. Wieder, J. Shan. "Web Services Agreement-based Resource Negotiation in UNICORE." *Central Institute of Applied Mathematics*, 2005.

[65] GLOBUS. [*Online*]. Available WWW: http://www.globus.org.

[66] L. Peterson, T. Anderson, D. Culler, and T. Roscoem "A blueprint for introducing disruptive technology into the Internet." *Proceedings of the First ACM Workshop on Hot Topics in Networks (HotNets-I)*, Princeton, NJ, October 2002. [*Online*]. Available WWW: http://www.cs.princeton.edu/courses/archive/fall03/cs597B/handouts/pdn02-001.pdf.

[67] M. Riedel, V. Sander, P. Wieder, J. Shan, "Web services agreement-based resource negotiation in UNICORE." *Proc. of the 2005 International Conference on Parallel and Distributed Processing Techniques and Applications*, Las Vegas, NV, June 2005.

[68] I. Foster, "The Grid: A new infrastructure for 21st century science." *Physics Today*, February 2002.

[69] G. von Laszewski, E. Blau, Eric, M. Bletzinger, J. Gawor, P. Lane, S. Martin, and M. Russell. "Software, component, and service deployment in computational grids." *Lecture Notes in Computer Science*, vol. 2370, pp. 244-256, Springer, 2002.

[70] FightAIDS@Home. [*Online*]. Available WWW: http://www.worldcommunitygrid.org/projects_showcase/viewFaahResearch.do.

[71] LHC Grid Project. [*Online*]. Available WWW: http://lcg.web.cern.ch/LCG/overview.html.

[72] Ian Foster, Carl Kesselman , Gene Tsudik , Steven Tuecke. "A security architecture for computational grids." *Proceedings of the 5th ACM conference on Computer and communications security*, November 1998. [*Online*]. Available WWW: http://www.ee.princeton.edu/~rblee/ELE572Papers/p83-foster.pdf.

[73] Breslau, Lee, Deborah Estrin, Kevin Fall, Sally Floyd, John Heidemann, Ahmed Helmy, Polly Huang, Steven McCanne, Kannan Varadhan, Ya Xu, and Haobu Yu. "Advances in Network Simulation." IEEE, May 2000.

[74] Sulistio, Anthony, Chee Shin Yeo and Rajkumar Buyya. "A taxonomy of computer-based simulations and its mapping to parallel and distributed systems simulation tools." Software – Practice and Experience, Vol. 34, 2004: pp. 653-673.

[75] Löwy, Juval. "Programming .NET Components, Second Edition." O'Reilly Media, July 2005.

[76] Templeman, Julian, and Andy Olsen. "Microsoft® Visual C++® .NET: Step by Step." Microsoft Press, March 26, 2003.

[77] Shepherd, George and David Kruglinski. "Programming with Microsoft Visual C++ .NET." Microsoft Press, October 25, 2002.

[78] Gregory, Kate. "Microsoft® Visual C++® .NET 2003 Kick Start." Sams, December 4, 2003.

[79] Ferreira, Luis, Viktors Berstis, and Jonathan Armstrong. “Introduction to Grid Computing with Globus.” IBM, October 1, 2003.

[80] Jacob, Bart, Luis Ferreira, and Norbert Bieberstein. “Enabling Applications for Grid Computing with Globus.” IBM, June 18, 2003.

[81] Powers, Lars and Mike Snell. “Microsoft Visual Studio 2005 Unleashed.” Sams, 2007.

[82] Hurwitz, Dan and Jesse Liberty. “Programming .NET Windows Applications.” O’Reilly, October 2003.

[83] Sells, Chris and Michael Weinhardt. “Windows Forms 2.0 Programming.” Addison Wesley Professional, May 16, 2006.

[84] Marshall, Donis. “Programming Microsoft® Visual C#® 2005: The Language.” Microsoft Press, December 20, 2005.

[85] Carter, Eric and Eric Lippert. “Visual Studio Tools for Office: Using Visual Basic 2005 with Excel, Word, Outlook, and InfoPath.” Addison Wesley Professional, April 26, 2006.

[86] BUG: TreeView Nodes Count Property Limited to 32767. [*Online*]. Available WWW: http://support.microsoft.com/kb/182231.

[87] S. Chakrabarti, A. Mishra, “QoS issues in ad hoc wireless networks”, IEEE Communications Magazine, Feb 2001, Vol. 39 Issue 2, pp. 142-148.

[88] A. Boukerche, M.Z. Ahmad, B. Turgut, and D. Turgut, “A Survey on Routing Protocols in Ad hoc Networks”, Handbook on Algorithms and Protocols for Wireless Ad hoc and Sensor Networks, Eds. Boukerche, Wiley, 2007.

[89] H. Hassanein and A. Zhou. “Routing with load balancing in wireless Ad hoc networks.” Proceedings of the 4th ACM International Workshop on Modeling, Analysis and Simulation of Wireless and Mobile Systems, 2001, pp. 89-96.

[90] A. Amis and R. Prakash, “Load-balancing clusters in wireless ad hoc networks”, Proc. of ASSET, Richardson, TX, March 2000, pp. 25-32.

[91] M. Chatterjee, S.K. Das and D. Turgut, “WCA: A Weighted Clustering Algorithm for Mobile Ad hoc Networks”, Journal of Cluster Computing (Special Issue on Mobile Ad hoc Networks), Vol. 5, No. 2, April 2002, pp. 193-204.

[92] M. Gerla and J.T.C. Tsai, “Multicluster, mobile, multimedia radio network”, Wireless Networks, Vol. 1, No. 3, 1995, pp. 255-265.

[93] A.B. McDonald and T.F. Znati, “A mobility-based framework for adaptive clustering in wireless ad hoc networks”, IEEE Journal on Selected Areas in Communications, Vol. 17, No. 8, 1999, pp. 1466-1487.

[94] D.J. Baker and A. Ephremides, “A distributed algorithm for organizing mobile radio telecommunication networks”, Proc. of the 2nd Int. Conference on Distributed Computer Systems, April 1981, pp. 476-483.

[95] D.J. Baker and A. Ephremides, “The architectural organization of a mobile radio network via a distributed algorithm,” IEEE Transactions on Communications COM-29 11 (1981), pp. 1694-1701.

[96] A.K. Parekh, “Selecting routers in ad-hoc wireless networks”, Proceedings of the SBT/IEEE International Telecommunications Symposium, August 1994.

[97] S. Basagni, “Distributed and mobility-adaptive clustering for multimedia support in multi-hop wireless networks”, Proc. of Vehicular Technology Conference, VTC, Vol. 2, 1999-Fall, pp. 889-893.

[98] K. Nikano, S. Olariu, “Uniform Leader Election Protocols for Radio Networks”, IEEE Transactions on Parallel and Distributed Systems, Vol. 13, No. 5, May 2002, pp. 516-526.

[99] K. Nikano, S. Olariu, “Randomized Leader Election Protocols in Radio Networks with No Collision Detection”, Proc. of the 11th International Conference on Algorithms and Computation, 2000, pp. 362-373.

[100] C.C. Chiang, H.K. Wu, W. Liu, and M. Gerla, “Routing in clustered multihop, mobile wireless networks with fading channel”, Proc. of IEEE SICON, 1997, pp. 197-211.

[101] T.C. Hou and T.J. Tsai, “An access-based clustering protocol for multihop wireless ad hoc networks,” IEEE Journal on Selected Areas in Communications, vol. 19, no. 7, July 2001.

[102] A. D. Amis, R. Prakash, D. Huynh, and T. Vuong, “Max-Min D-cluster formation in wireless ad hoc networks”, Proc. Of IEEE INFOCOM, 2002, pp. 32-41.

[103] S. Basagni, M. Mastrogiovanni, A. Panconesi, and C. Petrioli, “Localized Protocols for Ad Hoc Clustering and Backbone Formation: A Performance Comparison”, IEEE Transactions on Parallel and Distributed Systems, Volume 17, Issue 4, April 2006, pp. 292-306.

[104] Iamnitchi, A. and I. Foster. "On Fully Decentralized Resource Discovery in Grid Environments." International Workshop on Grid Computing, Denver, CO, November 2001.

[105] Bradley, Alan, Kevin Curran and Gerard Parr. "Discovering Resource in Computational GRID Environments." The Journal of Supercomputing, Volume 35, 2006, pp. 27-49.

[106] Joseph, Joshy, and Craig Fellenstein. "Grid Computing." IBM Press, Upper Saddle River, N. J., December 30, 2003.

[107] Leong, P., Chunyan Miao, and Bu-Sung Lee. "Agent oriented software engineering for grid computing." Parallel Computing in Electrical Engineering, 2006.

[108] Bucur, Doina. "Resource Discovery in Activity-Based Sensor Networks." Mobile Networks and Applications, Vol. 12 Issue 2/3, pp. 129-142.

[109] Dubhashi, Devdatt, Olle Häggström, Gabriele Mambrini, Alessandro Panconesi, and Chiara Petrioli. "Blue pleiades, a new solution for device discovery and scatternet formation in multi-hop bluetooth networks." Wireless Networks, Volume 13 Issue 1, pp. 107-125.

[110] Joint Training Experimentation Program. [*Online*]. Available WWW: http://www.jtepforguard.com/29Palms405.html

[111] Possio's PX 30 Hackable Wireless Router. [*Online*]. Available WWW: http://linuxdevices.com/articles/AT7459336271.html