# Distributed Computer Architectures for Combat Systems

*Mark E. Schmid and Douglas G. Crowe*

Advances in computer hardware, networks, and support software have brought distributed computing to the forefront of today's Navy combat systems. An effort known as the High Performance Distributed Computing (HiPer-D) Program was established in 1991 to pursue advantageous applications of this technology to the Navy's premier Aegis cruiser and destroyer combatants. This article provides an overview of the program accomplishments and presents a brief discussion of some of the intercomputer communications issues that were paramount to the program's success. Most significant was the credibility built for distributed computing within the Aegis community. This was accomplished through many demonstrations of combat system functions that exhibited capacity, timeliness, and reliability attributes necessary to meet stringent Aegis system requirements.

## INTRODUCTION

In the 1950s, the Navy embarked on its course of developing standardized computers for operational use. The AN/UYK-7, AN/UYK-20, AN/UYK-43, and AN/UYK-44 are still the primary combat system computers deployed today (Fig. 1). They function at the heart of the Navy's premier surface ships, merging information from sensor systems, providing displays and warnings to the crew, and controlling the use of weapons. These computers are wholly Navy developed. The processing units, interfaces, operating system, and support software (e.g., compilers) were all built by the Navy and its contractors. The AN/UYK-43 is the last in the line of Navy standard computers (oddly enough going into production *after* its smaller sibling, the AN/UYK-44). Described in 1981 as "the new large scale computer . . . expected to be in service for the next 25 years,"[1] it will actually see a service life beyond that.

The Navy Standard Computer Program was successful at reducing logistics, maintenance, and a host of other costs related to the proliferation of computer variants; however, in the 1980s, the Navy faced a substantial increase in the cost of developing its own machines. Pushed by leaps in the commercial computing industry, everything associated with Navy Standard Computer development—from instruction set design to programming language capabilities—was becoming more complex *and* more expensive.

By 1990, it was clear that a fundamental shift was occurring. The shrinking cost and dramatically growing performance of computers were diminishing the significance of the computer itself. The transformation of centralized mainframe computing to more local and responsive minicomputers was bringing about the proliferation of even lower-cost desktop computers. Microprocessors

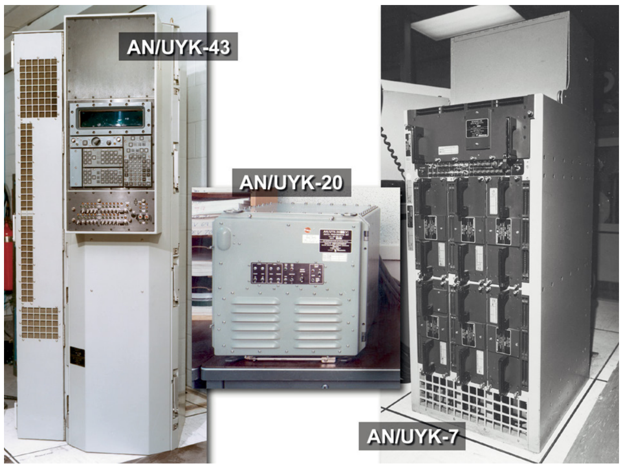


**Figure 1.** Navy family computers.

had indeed achieved the performance of their mainframe predecessors just 10 years earlier, but this was only *half* of the shift! Spurred by new processing capabilities, system designers were freed to tackle increasingly complex problems with correspondingly more intricate solutions. As a new generation of complex software took shape, the balance of computer hardware and software investments in Navy systems cascaded rapidly to the software side.

Harnessing the collective power of inexpensive mainstream commercial computers for large, complex systems was clearly becoming prominent in Navy combat system design. The highly successful demonstration in 1990 of the Cooperative Engagement Capability (CEC) prototype, built with over 20 Motorola commercial microprocessors, was a milestone in distributed computing for Navy combat system applications. Developments in both Aegis and non-Aegis combat systems focused thereafter on applying the capabilities of distributed computing to virtually all of the Navy's prototype efforts. Distributed computing offered the potential for providing substantially more powerful and extensible computing, the ability to replicate critical functions for reliability, and the simplification of software development and testing through the isolation of functions in their own computing environments. Indeed, when the Navy's Aegis Baseline 7 and Ship Self-Defense System (SSDS) Mk 2 Mod 1 are delivered (expected in 2002), a commercial distributed computing infrastructure will have been established for all new builds and upgrades of the Navy's primary surface combatants (carriers, cruisers, and destroyers).

Distributed computing for combat systems, however, is not without its difficulties. Requirements for systems such as Aegis are stringent (response times, capacities, and reliability) and voluminous (dozens of independent sensor systems, weapon systems, operator interfaces, and networks to be managed), and the prior large body of design knowledge has been focused on Navy Standard Computer implementation. (As an example, the Aegis Combat System had a small number of AN/UYK-7 [later AN/UYK-43] computers. Each represented an Aegis "element" such as the Weapon Control System, Radar Control System, or Command and Decision System. Within an element, interactions among subfunctions were carried out primarily via shared memory segments. These do not map well to a distributed computing environment because the distributed nodes do not share any physical memory.)

Among the most fundamental issues facing this new generation of distributed computing design was the pervasive need for "track" data to be available "everywhere"—at all the computing nodes. Tracks comprise the combat system's best assessment of the locations, movements, and characteristics of the objects visible to its sensors. They are the focal point of both automated and manual decision processes as well as the subject of interactions with other members of the battle group's networks. Ship sensors provide new information on each track periodically, sometimes at high rate. While the reported information is relatively small (e.g., a new position and velocity), the frequency of the reports and the large number of objects can combine to yield a demanding load. The number of track data consumers can easily amount to dozens and place the overall system communication burden in the region of 50 to 100,000 messages per second. (Figure 2 shows a scan with many tracks feeding messages forward.)

Although commercial machines and networks have advanced admirably in their ability to communicate data across networks, the combat system applications have some unusual characteristics that make their needs substantially different from the commercial mainstream. The track information that comprises the bulk of the combat system communications consists of many small (a few hundred bytes) messages. These messages can be independently significant to the downstream consumers, with varying requirements to arrive "quickly" at particular destinations. Commercial systems are not tuned to handle large numbers of small messages; they focus on handling the needs of the mass market, which are typically less frequent and larger sets of data.

## HIGH PERFORMANCE DISTRIBUTED COMPUTING

The HiPer-D Program originated as a joint Defense Advanced Research Projects Agency (DARPA)/Aegis (PMS-400) endeavor, with three participating technical organizations: APL, Naval Surface Warfare Center Dahlgren Division (NSWCDD), and Lockheed Martin Corporation. It was inspired by a future shipboard computing vision[2] and the potential for synergy between it and a wave of emerging DARPA research products in distributed computing. The program commenced in

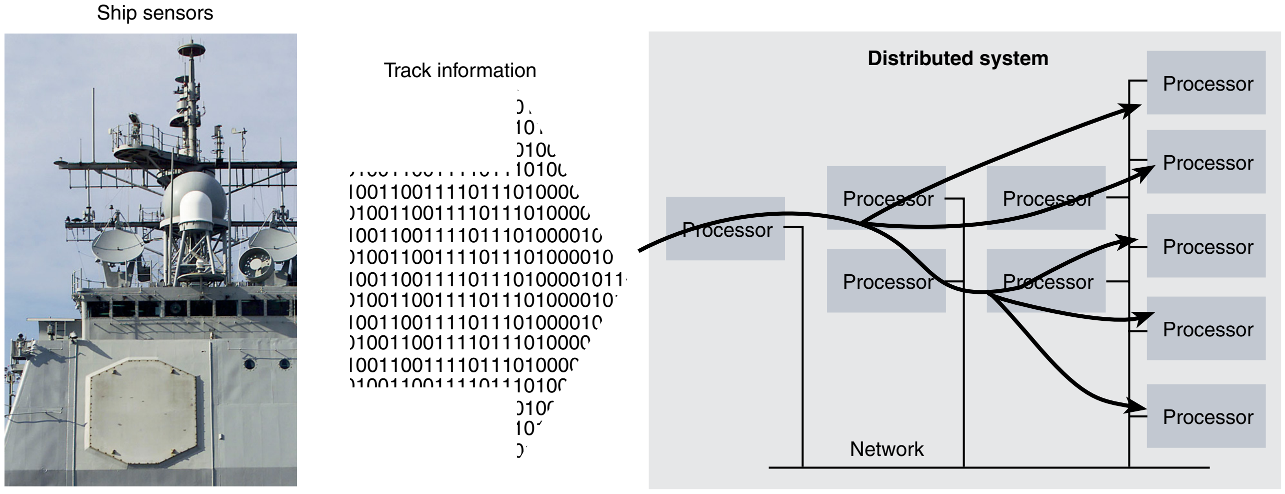


**Figure 2.** Information originates from sensor views of each object. Each report is small, but the quantity can be very large (because of high sensor report rates and potentially large numbers of objects). The reports flow throughout to be integrated and modified, and to enable rapid reaction to change.

June 1991 and has been an ongoing research and development effort designed to reduce risk due to the introduction of distributed computing technologies, methods, and tools into the Aegis Weapon System.

Many of the significant developments of HiPer-D have been showcased in demonstrations. Since 1994, when the first major HiPer-D demonstration was held, there have been seven demonstrations that provided exposure of concepts and techniques to Aegis and other interested communities. The first HiPer-D integrated demonstration (Fig. 3) focused primarily on the potential for applying massively parallel processor (MPP) technology to combat system applications. It brought together a collection of over 50 programs running on 34 processors (including 16 processors on the Intel Paragon MPP) as a cooperative effort of APL and NSWCDD. The functionality included a complete sensor-to-weapon path through the Aegis Weapon System, with doctrine-controlled automatic and semi-automatic system responses and an operator interface that provided essential tactical displays and decision interfaces. The major anomaly relative to the actual Aegis system was the use of core elements from CEC (rather than Aegis) to provide the primary sensor integration and tracking.

By many measures, this first demonstration was a significant success. Major portions of weapon system functionality had been repartitioned to execute on a large collection of processors, with many functions constructed to take advantage of replication for reliability and scaling. The Paragon MPP, however, suffered severe performance problems connecting with computers outside its MPP communication mesh. Key external network functions were on the order of 20 times slower than on Sun workstations of the same era. The resulting impact on track capacity (less than 100) drove later exploration away from the MPP approach.

The next-generation demonstration had two significant differences: (1) it focused on the use of networked workstations to achieve the desired computing capacity, and (2) it strove for a stronger grounding in the existing Aegis requirements set. The primary impact of these differences was to switch from a CEC-based ship tracking approach to an Aegis-derived tracking approach. Additional fidelity was added to the doctrine processing, and

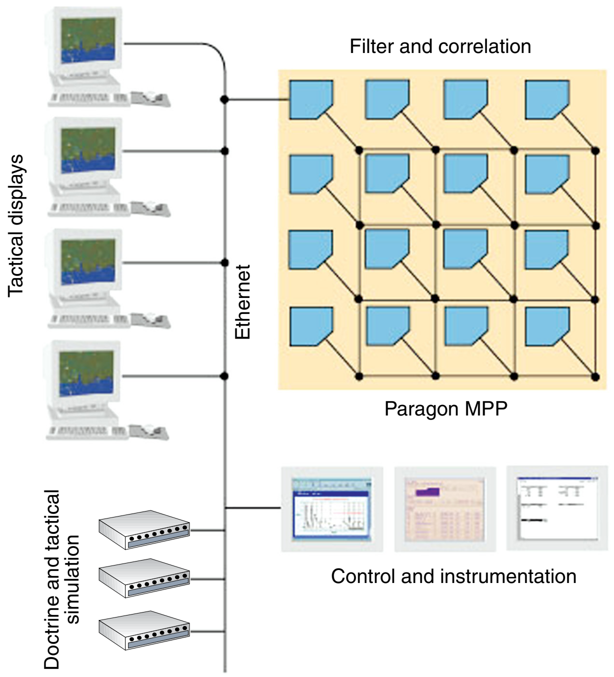


**Figure 3.** First HiPer-D demonstration physical/block diagram.

the rapid self-defense response capability referred to as "auto-special" was added to provide a context in which to evaluate stringent end-to-end timing requirements. The 1995 HiPer-D demonstration[3–5] represented a significant milestone in assessing the feasibility of moving commercial distributed computing capabilities into the Aegis mission-critical systems' domain. Track capacity was improved an order of magnitude over the original demonstration; a variety of replication techniques providing fault tolerance and scalability were exhibited; and critical Aegis end-to-end timelines were met.

Subsequent demonstrations in 1996 to 1998 built upon the success of the 1995 demonstration, adding increased functionality; extending system capacity; exploring new network technologies, processors, and operating systems; and providing an environment in which to examine design approaches that better matched the new distributed paradigm. Indeed, a fairly complete software architecture named "Amalthea"[4] was created to explore the feasibility of application-transparent scalable, fault-tolerant distributed systems. It was inspired by the Delta-4 architecture done in the late 1980s by ESPRIT (European Strategic Programme for Research and Development in Information Technology)[5] and was used to construct a set of experimental, reliable applications. In addition to architectural exploration, the application domain itself was addressed. Even though the top-level system specification continued to convey a valid view of system performance requirements, the decomposition of those top-level requirements into the Aegis element program performance specifications had been crafted 20 years earlier with a computing capacity and architecture vastly different from the current distributed environment.

The command and decision element, from which much of the HiPer-D functionality originated, seemed like a particularly fertile area for significant change. An example of this was the tendency of requirements to be specified in terms of a periodic review. In the case of doctrine-based review for automatic actions, the real need was to ensure that a track qualifying for action was identified within a certain amount of time, not necessarily that it be done periodically. In the HiPer-D architecture, it is simple to establish a client (receiver of track data) that will examine track reports for the doctrine-specified action *upon data arrival*. From a responsiveness standpoint, this far exceeds the performance of the periodic review specified in the element requirements and yet is a very straightforward approach for the high-capacity and independent computing environment of the distributed system.

The 1999 and 2000 demonstrations[6,7] continued the pattern, bringing in new functionality and a significant new paradigm for scalable fault-tolerant servers that use new network switch technology (see next section). This phase of development also addressed a need for instrumenting the distributed system which had been identified early in the program but met with a less-than-satisfactory solution.

Distributed systems apply the concept of "work in parallel," where each working unit has its own independent resources (processor and memory). Although this eases the contention between functions running in parallel, it elevates the complexity of test and analysis to a new level. Collection and integration of the information required to verify performance or diagnose problems require sophisticated data gathering daemons to be resident on each processor, and analysis capabilities that can combine the individual processor-extracted data streams must be created. Early in the HiPer-D effort, a German research product called "JEWEL"[8] was found to provide such capabilities with displays that could be viewed in real time. The real-time monitoring/analysis capability of HiPer-D was not only one of its most prominent demonstration features, but some would claim also one of the most significant reasons that the annual integration of the complex systems was consistently completed on time. JEWEL, however, never achieved status as a supported commercial product and was somewhat cumbersome from the perspective of large system development. For example, it had an inflexible interface for reporting data, allowing only a limited number of integers to be reported on each call. Data other than integers could be reported, but only by convention with the ultimate data interpreter (graphing program).

Beginning in 1999, a real-time instrumentation tool kit called Java Enhanced Distributed System Instrumentation (JEDSI) was developed and employed as a replacement for JEWEL. It followed the JEWEL model of providing very low overhead extraction of data from the processes of interest but eliminated the interface issues that made it awkward for large-scale systems. It also capitalized on the use of commercially available Java-based graphics packages and communications support to provide an extensive library of real-time analysis and performance visualization tools.

Current and planned initiatives in HiPer-D are moving toward multi-unit operations. Functionality for integrating data from Navy tactical data links has been added to the system, and the integration of real CEC components is planned for late 2001. Such a capability will provide the necessary foundation to extend the exploration of distributed capabilities beyond the individual unit.

## EVOLUTION OF COMMUNICATIONS IN HiPer-D

The first phase of HiPer-D, culminating in the system shown in Fig. 3, had a target computing environment that consisted of an MPP as the primary computing resource, surrounded by networked computers that modeled the sensor inputs and workstations that provided graphical operator interfaces. The MPP was an Intel Paragon (Fig. 4). It had a novel internal interconnect and an operating system (Mach) that had message passing as an integral function. Mach was the first widely used (and, for a brief period, commercially supported) operating system that was built as a "distributed operating system"—intended from the outset to efficiently support the communication and coordination requirements of distributed computing.

Despite these impressive capabilities, an important principle had been established within the DARPA community that rendered even these capabilities incomplete for the objective of replication for fault tolerance and scalability. Consider the computing arrangement in Fig. 5. A simple service—the allocation of track numbers to clients who need to uniquely label tracks—is replicated for reliability. The replica listens to the requests and performs the same simple assignment algorithm, but does not reply to the requester as the primary server does. This simple arrangement can be problematic if the communications do not behave in specific ways. If one of the clients' requests is not seen by the backup (i.e., the message is not delivered), the backup copy will be offset 1 from the primary. If the order of multiple clients' requests is interchanged between the primary and backup servers, the backup server will have an erroneous record of numbers assigned to clients. Furthermore, at such time as the primary copy is declared to have failed, the backup will not know which client requests have been serviced and which have not.

**Figure 4.** Paragon.

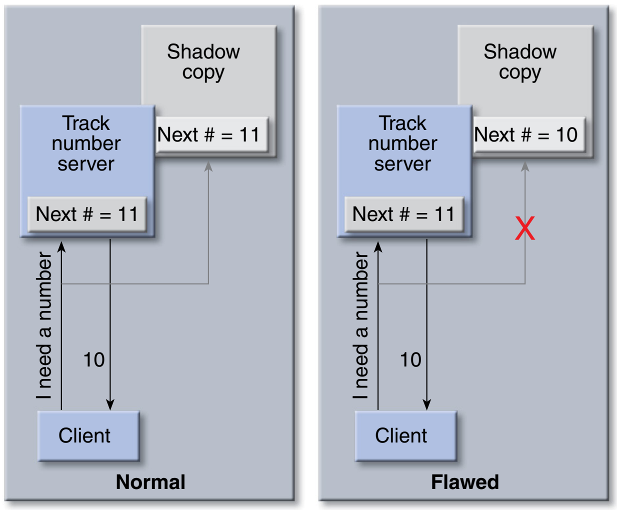


**Figure 5.** Simple replicated track number service operations.

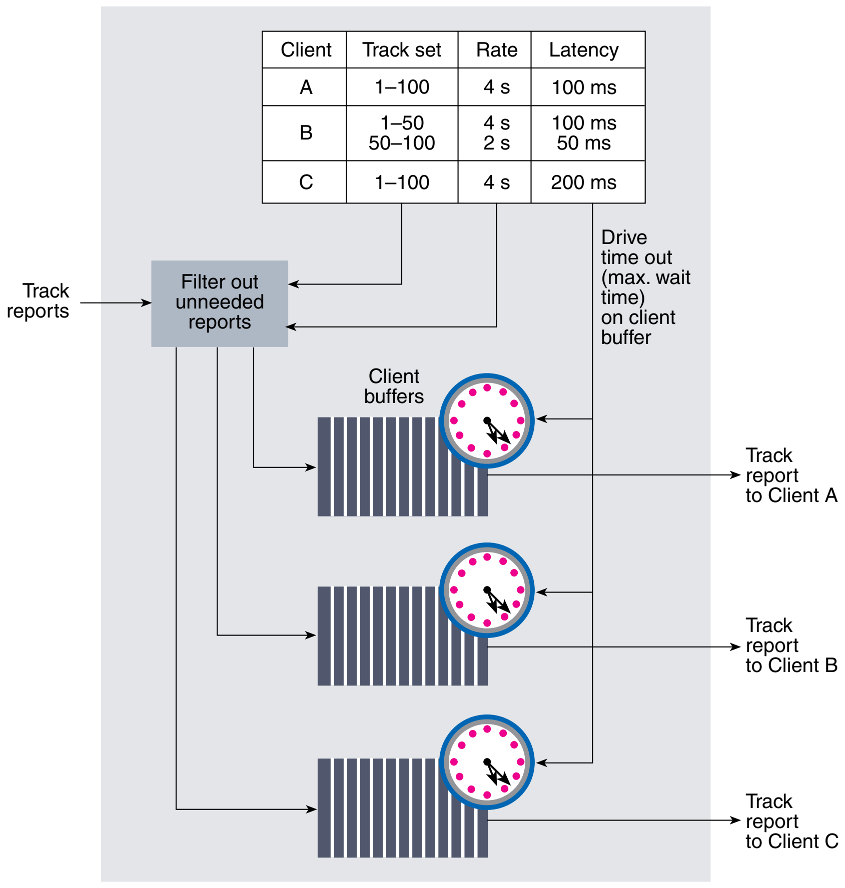
Communication attributes required to achieve the desired behavior in this server example include reliable delivery, ordered delivery, and atomic membership change. Reliable delivery assures that a message gets to all destinations (or none at all) and addresses the problem of the servers getting "out of synch." Ordered delivery means that, within a defined group (specifying both members and messages), all messages defined as part of the group are received in the same order. This eliminates the problem that can occur when the primary and replica see messages in a different order. Atomic membership change means that, within a defined group, any member entering or leaving the group (including failure) is seen to do so at the same point of the message stream among all group members. In our simple server example above, this allows a new replica to be established with a proper understanding of how much "history" of previous primary activity must be supplied, and how much can be picked up from the current client request stream.

A convenient model for providing these communication attributes is "process group communications." In this model, the attributes above are guaranteed, i.e., all messages are delivered to all members or none, all messages are delivered to all group members in the same order, and any members that leave or join the group are seen to do so at the same point within the group's message stream. Early HiPer-D work used a toolkit called the Isis Distributed Toolkit[9] that served as the foundation for process group communications. Initially developed at Cornell University, it was introduced as a commercial product by a small company named Isis Distributed Systems.

The cornerstone element of the HiPer-D communications chain was an application called the Radar Track Server (RTS). Its task was to service a large set of (potentially replicated) clients, providing them with track data at their desired rate and staleness. Fed from the stream of track data emerging from the primary radar sensor and its subsequent track filter, the RTS allows each consumer of the track information to specify desired update rates for the tracks and further specify latency or staleness associated with the delivery of track information. This was a significant, new architectural strategy. In prior designs, track reports were either delivered en masse at the rate produced by the sensor or delivered on a periodic basis. (Usually, the same period was applied to all tracks and all users within the system, which meant a compromise of some sort.) These new server capabilities brought the ability to selectively apply varying performance characteristics (update rate and latency) based on each client's critical evaluation of each individual track.

The RTS matches a client's requested rate for a track by filtering out unnecessary reports that occur during the target report intervals (Fig. 6). For instance, if the sensor provides an update every second but a client requests updates every 2 s, the RTS will send only alternate updates. Of course, this is client- and track-specific. In the previous example, should a track of more extreme interest appear, the client could direct the RTS to report that specific track at the full 1-s report rate. (There was also successful experimentation in use of the RTS to feed back consumer needs to the sensor to enable optimization of sensor resources.) If a consumer's desired report rate is not an integral multiple of the sensor rate, the RTS maintains an "achieved report rate" and decides whether to send an update based on whether its delivery will make the achieved report rate closer or further away from the requested report rate.

One lesson learned very early in the HiPer-D effort was that, in network communications, the capacity to deliver messages benefits immensely from buffering and delivery of collections of messages rather than individual transmission of each message. Of course, this introduces latency to the delivery of the messages that are forced to wait for others to fill up the buffer. The RTS uses a combination of buffering and timing mechanisms that allows the buffer to fill for a period not to exceed the client's latency specification. Upon reaching a full buffer condition or the specified maximum latency, the buffer—even if it is only a single message—is then delivered to the client.



| Client | Track set | Rate | Latency |
|---|---|---|---|
| A | 1–100 | 4 s | 100 ms |
| B | 1–50<br>50–100 | 4 s<br>2 s | 100 ms<br>50 ms |
| C | 1–100 | 4 s | 200 ms |

**Figure 6.** Radar track server.

The criticality of the RTS to the system and the anticipated load from the large number of clients led to its immediate identification as a prime candidate for applying replication to ensure reliability and scalability to high load levels. All these factors made the RTS a rich subject for exploration of process group communications. Figure 7 depicts the basic set of process group communications–based interactions among a set of cooperating servers providing track data to a set of clients. The process group communications delivery properties (combined with the particular server design) allowed the server function to be replicated for both improved capacity and reliability. It also allowed clients to be replicated for either or both purposes.

The first Paragon-centered demonstration included the use of the process group communications paradigm and the RTS. Even though the internal Paragon communications mesh proved to be very capable, the system as a whole suffered from a severe bottleneck in moving track data into and out of the

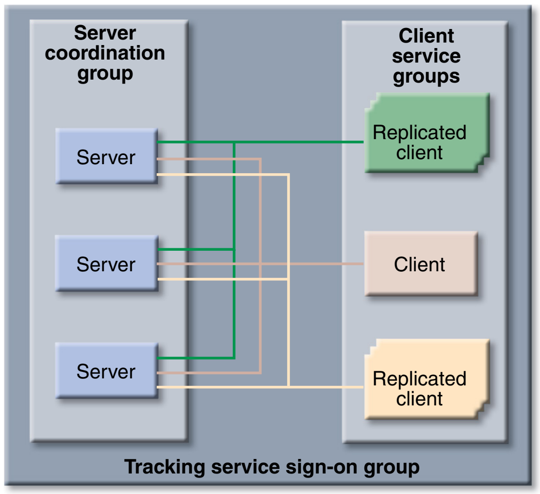


**Figure 7.** Basic RTS operations (different groups and message content).

Paragon. The 16-node Paragon mesh was shown capable of 8 pairs of simultaneous communications streams of 1000 messages a second (with no buffering!). Compared to the roughly 300 messages per second being achieved on desktops of that period, the combination of Mach messaging features and Paragon mesh was impressive. However, as described earlier, the Paragon's Ethernet-based external connectivity was flawed. Apparently, in the niche of Paragon customers, Ethernet interface performance was not of substantial concern. Serving the "supercomputer market," the Paragon was more attuned to high-performance parallel input devices (primarily disk) and high-end graphics engines that had HiPPi's (High-Performance Parallel Interfaces).

The migration from Paragon MPP hardware, with its mesh-based communications, to the network-based architecture of subsequent demonstrations proved to be smooth for process group communications. The elimination of poor-performance Paragon Ethernet input/output allowed the network-based architecture to achieve a realistic capacity of 600 tracks while meeting subsecond timelines for high-profile, time-critical functions. The RTS scalable design was able to accommodate 9 distinct clients (with different track interests and report characteristics), some of them replicated, for a 14-client total.

Although the network architecture and process group communications were major successes, the Isis Distributed Toolkit mentioned earlier was encountering real-world business practices. When Isis Distributed Systems was acquired by Stratus Computer, the event was initially hailed as a positive maturation of the technology into a solid product that could be even better marketed and supported. However, in 1997, Stratus pulled Isis from the market and terminated support.

This was unsettling. Even though there was no need to panic (the existing Isis capabilities and licenses would continue to serve the immediate needs), the demise of Isis signaled the indifference of the commercial world to the type of distributed computing that Isis supported (and was found to be so useful). APL eventually employed another research product, "Spread," developed by Dr. Yair Amir of The Johns Hopkins University.[10] It provides much of the same process group communications support present in Isis, is still used by HiPer-D today, and has just recently been "productized" by Spread Concepts LLC, Bethesda, MD. The absence of any large market for commercial process group communications, however, is an important reminder that, although systems that maximize the use of commercial products are pursued, the requirements of the Navy's deployed combat systems and those of the commercial world will not always coincide.

In 1998, an interesting new capability in network switches, combined with some ingenuity, provided a new approach for developing fault-tolerant and scalable server applications. Recent advances in network switch technology allowed for fast, application-aware switching (OSI Level 4). With an ability to sense whether a server application was "alive," a smart switch could intelligently balance incoming client connections among multiple servers and route new connections away from failed ones.

The RTS was chosen as a proving ground for this new technology. Because of its complexity, the RTS was the one remaining HiPer-D function that employed Isis. It also had the potential to benefit substantially (performance-wise) by eliminating the burden of inter-RTS coordination. The inter-RTS coordination used features of Isis to provide a mechanism for exchange of state information (e.g., clients and their requested track characteristics) between server replicas. This coordination added complexity to the RTS and the underlying communications and consumed processing time, network bandwidth, and memory resources.

The switch-based RTS continues to support the major functionality of the original RTS but does so with a much simpler design and smaller source code base. The guiding principle in the design of the new RTS was simplicity. Not only would this prove easier to implement, it would also provide fewer places to introduce latency, thus improving performance. Each RTS replica operates independently of all others. In fact, the servers do not share *any* state information among them and are oblivious to the existence of any other servers. The implementation of a design that does not require knowledge of other servers was a dramatic change from the original RTS design, which had required extensive coordination among the servers to ensure that all clients were being properly served, particularly under the aberrant conditions of a client or server failure.

Clients of the switch-based RTS server set view it as a single virtual entity (Fig. 8). Each time a new client requests service from the "virtual RTS," the Level 4

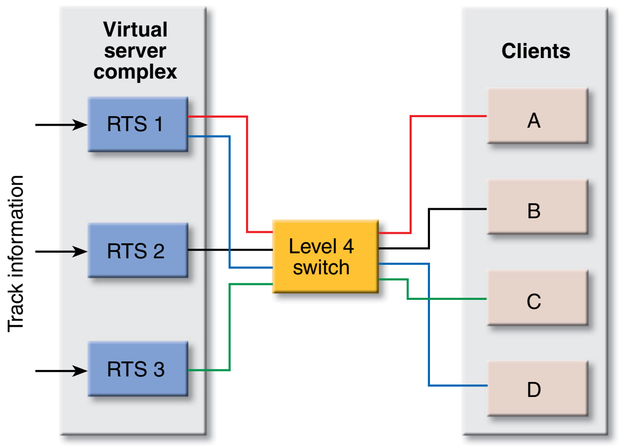


**Figure 8.** Switch-based RTS block diagram.

switch selects one of the RTS servers to handle the request. The selection is currently based on a rotating assignment policy but could be based on other loading measures that are fed to the switch. Once a client is established with a particular server, that server handles all its needs. In the event of server failure, a client-side library function establishes connection with a new server and restores all the previously established services. In addition to replication of the server, the Level 4 switch itself can be replicated to provide redundant paths to the servers. This accommodates failure of the switch itself as well as severance of a network cable.

Figure 9a is a time plot of a typical end-to-end response for the combat system demonstration application that uses the RTS. From sensor origination through track filtering and RTS delivery to the client, average latencies are approximately 80 ms. The 80-ms latency is a nominal level of performance that is intended for tracks and processing purposes where low latency is not of high concern. By tuning the processing functions and clients, latencies in the 20- to 30-ms range can be achieved but at the expense of capacity. This is a clear engineering trade-off situation, where the proper balance can be adjusted for a particular application (using command line parameters).

A server software failure was forced in the center of the run shown in Fig. 9a, with little perceptible impact. Figure 9b shows the spikes in maximum latency that occur as a result of physical disconnection of the server from the switch. Physical reconfigurations, however, take 2 to 4 s to complete.

The switch-based RTS is a significant simplification, reducing the source code from 12,000 to 5,000 lines; however, two of its performance attributes are of potential concern to clients:

1. Delivery of all data to a client is not strictly guaranteed. When a server fails, messages that transpire between the server failure and the client's reestablishment with a new server are not eligible for delivery to that client. However, when the client reconnects, it will immediately receive the freshly arriving track updates. In the case of periodically reported radar data, this is sufficient for most needs.
2. Failures of the switch itself, the cabling, or server hardware incur a relatively large delay to recovery. This is a more serious potential problem for clients because the reestablishment of service takes multiple seconds. Initially, much of this time was used to detect the fault. A "heartbeat" message was added to the client-server protocol to support subsecond fault detection, but the mechanisms in the Level 4 switch for dismantling the failed connection and establishing a new one could not be accelerated without greater vendor assistance than has been available. This seemed to be a classic case of not representing a strong enough market to capture the vendor's attention. Had this been a higher-profile effort, with more potential sales on the line, it is our perception that this problem could have been addressed more satisfactorily.

The use of Level 4 network switch technology provides two major advantages. First, server implementation is much simpler. The new RTS architecture is similar in design to a replicated commercial Internet server. The complex server-to-server coordination protocols required to manage server and client entries exist, and failures have been eliminated. Second, the communications layer

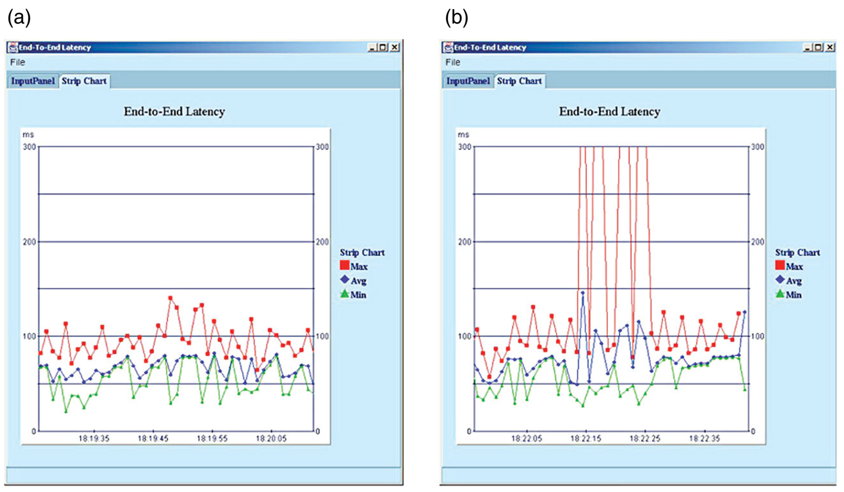


**Figure 9.** Nominal performance with recovery (a) and performance with cable pull diagram (b).

used between the server and its clients is the same "socket" service used by major Internet components like browsers and Web servers. The use of standard commercial communications greatly reduces the dependency of the software system on specific products. Therefore, porting the software to new (both processor and network) environments will be straightforward, lowering the life-cycle and hardware upgrade costs.

## CONCLUSION

Foremost among the accomplishments of the HiPer-D Program is the long list of proof-of-concept demonstrations to the Aegis sponsor and design agent, Lockheed Martin:

- The sufficiency of distributed computing for Aegis requirements
- The ability to replicate services for reliability and scalability
- The ability to employ heterogeneous computing environments (e.g., Sun and PC)
- The feasibility and advantages of a track data server
- A single track numbering scheme and server for allocating/maintaining track numbers
- The feasibility of standard network approaches like Ethernet, FDDI (Fiber-Distributed Data Interface) and ATM (asynchronous transfer mode)
- Strategies for control of the complex distributed system and reconfiguration for reliability or scaling
- The value of real-time instrumentation

The relationship between these accomplishments and the production Aegis baselines is primarily indirect. While it would be improper to claim "responsibility" for any of the advances that the Lockheed Martin engineering team has been able to infuse into ongoing Aegis developments, it is clear that the HiPer-D efforts have emboldened the program to advance rapidly. Currently, three major Aegis baselines are under development, each with increasingly higher reliance on distributed computing and technology. The most recent, referred to as Aegis Open Architecture, will fully capitalize on its distributed architecture, involving a complete re-architecting of the system for a commercial-off-the-shelf distributed computing infrastructure.

In the specific area of communications, HiPer-D has continued to follow the advance of technology in its pursuit of services that can effectively meet Aegis tactical requirements. Robust solutions with satisfyingly high levels of performance have been demonstrated. The problem remains, however, that the life of commercial products is somewhat fragile. The bottom-line need for profit in commercial offerings constrains the set of research products that reach the commercial market and puts their longevity in question. The commercial world, particularly the computing industry, is quite dynamic. Significant products with apparently promising futures and industry support can arrive and disappear in a very short time. An interesting example of this is FDDI network technology. Arriving in the early 1990s as an alternative to Ethernet that elevated capacity to the "next level" (100 Mbit), and garnering the support of virtually all computer manufacturers, it has all but disappeared. However, it is this same fast-paced change that is enabling new system concepts and architectural approaches to be realized.

The challenge, well recognized by Lockheed Martin and integrated into its strategy for Aegis Open Architecture, is to build the complex application system on a structured foundation that isolates the application from the dynamic computing and communication environment.

## REFERENCES

[1]Wander, K. R., and Sleight, T. P., "Definition of New Navy Large Scale Computer," in *JHU/APL Developments in Science and Technology*, DST-9, Laurel, MD (1981).

[2]Zitzman, L. H., Falatko, S. M., and Papach, J. L., "Combat System Architecture Concepts for Future Combat Systems," *Naval Eng. J.* **102**(3), 43–62 (May 1990).

[3]*HiPer-D Testbed 1 Experiment Demonstration*, F2D-95-3-27, JHU/APL, Laurel, MD (16 May 1995).

[4]*Amalthea Software Architecture*, F2D-93-3-064, JHU/APL, Laurel, MD (10 Nov 1993).

[5]Powell, D. (ed.), *Delta-4: A Generic Architecture for Dependable Distributed Computing*, ESPRIT Project 818/2252, Springer-Verlag (1991).

[6]*High Performance Distributed Computing Program 1999 Demonstration Report*, ADS-01-007, JHU/APL, Laurel, MD (Mar 2001).

[7]*High Performance Distributed Computing Program (HiPer-D) 2000 Demonstration Report*, ADS-01-020, JHU/APL, Laurel, MD (Feb 2001).

[8]Gergeleit, M., Klemm, L., et al., *Jewel CDRS (Data Collection and Reduction System) Architectural Design*, Arbeitspapiere der GMD, No. 707 (Nov 1992).

[9]Birman, K. P., and Van Renesse, R. (eds.), *Reliable Distributed Computing with the Isis Toolkit*, IEEE Computer Society Press, Los Alamitos, CA (Jun 1994).

[10]Stanton, J. R., *A Users Guide to Spread Version 0,1*, Center for Networking and Distributed Systems, The Johns Hopkins University, Baltimore, MD (Nov 2000).

## THE AUTHORS

MARK E. SCHMID received a B.S. from the University of Rochester in 1978 and an M.S. from the University of Maryland in 1984, both in electrical engineering. He joined APL in 1978 in ADSD's Advanced Systems Development Group, where he is currently an Assistant Group Supervisor. Mr. Schmid's technical pursuits have focused on the application of emergent computer hardware and software technology to the complex environment of surface Navy real-time systems. In addition to the HiPer-D effort, his work includes pursuit of solutions to Naval and Joint Force system interoperability problems. His e-mail address is mark.schmid@jhuapl.edu.

DOUGLAS G. CROWE is a member of the APL Senior Professional Staff and Supervisor of the A2D-003 Section in ADSD's Advanced Systems Development Group. He has B.S. degrees in computer science and electrical engineering, both from The Pennsylvania State University, as well as an M.A.S. from The Johns Hopkins University. Mr. Crowe has been the lead for groups working on the development of automatic test equipment for airborne avionics, training simulators, and air defense C3I systems. He joined APL in 1996 and is currently the lead for APL's HiPer-D efforts. His e-mail address is douglas.crowe@jhuapl.edu.