AFRL-IF-RS-TR-2006-148
Final Technical Report
April 2006

PROMETHEUS: ENHANCING THE QUALITY OF SERVICE OF THE JOINT BATTLESPACE INFOSPHERE

Vanderbilt University

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE
ROME RESEARCH SITE
ROME, NEW YORK

STINFO FINAL REPORT

This report has been reviewed by the Air Force Research Laboratory, Information Directorate, Public Affairs Office (IFOIPA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

AFRL-IF-RS-TR-2006-148 has been reviewed and is approved for publication.

APPROVED: /s/

DALE W. RICHARDS
Project Engineer

FOR THE DIRECTOR: /s/

JAMES W. CUSACK
Chief, Information Systems Division
Information Directorate

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 074-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	APRIL 2006	Final Mar 2005 – Jan 2006

4. TITLE AND SUBTITLE
PROMETHEUS: ENHANCING THE QUALITY OF SERVICE OF THE JOINT BATTLESPACE INFOSPHERE

5. FUNDING NUMBERS
C - FA8750-05-2-0128
PE - 62702F
PR - 558J
TA - BA
WU - P2

6. AUTHOR(S)
DOUGLAS C. SCHMIDT, KEN BIRMAN, MIKE REITER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Vanderbilt University
Station B, Box 7749,
110 21st Avenue, South
Nashville Tennessee 37235

8. PERFORMING ORGANIZATION REPORT NUMBER
N/A

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Air Force Research Laboratory/IFSE
525 Brooks Road
Rome New York 13441-4505

10. SPONSORING / MONITORING AGENCY REPORT NUMBER
AFRL-IF-RS-TR-2006-148

11. SUPPLEMENTARY NOTES

AFRL Project Engineer: Dale W. Richards/IFSE/ Dale.Richards@rl.af.mil

12a. DISTRIBUTION / AVAILABILITY STATEMENT
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 Words)*
This project investigated performance, scalability and associated quality of service (QoS) needs of the Global Information Grid (GIG) and Network-Centric Enterprise Systems (NCES) in the context of large-scale DoD C2 and C4ISR systems-of-systems (SoS) architectures, with additional attention to security considerations. The objectives of this study were to (1) Explore and classify near-term (lower risk) requirements and opportunities; (2) Develop prototypical proofs-of-concepts and evaluations aimed at establishing a technology baseline in the area of QoS-enabled JBI; (3) Produce a guide to future R&D activities, (4) Develop an empirically-based understanding of the extent to which QoS and other properties can be addressed in limited ways by these COTS platforms; and (5) Identify open questions on which fundamental research may be required to close the gap between capabilities of existing SOA platforms, including anticipated platform extensions, and requirements that arise in DoD contexts.

14. SUBJECT TERMS
Quality of Service (QoS), Global Information Grid (GIG), Network-Centric Enterprise Systems (NCES), Joint Battlespace Infosphere (JBI), Systems of Systems (SoS), Service-Oriented Architectures (SOA)

15. NUMBER OF PAGES
78

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

Table of Contents

List of Figures

Executive Summary

This final report provides technical details concerning the results of activities performed for the ***Prometheus*** project, which was a 9-month project that studied the performance, scalability and associated quality of service (QoS) needs of the Global Information Grid (GIG) and Network-Centric Enterprise Systems (NCES) in the context of large-scale DoD C2 and C4ISR systems-of-systems (SoS) architectures, while also maintaining attention to security considerations. The task objectives of this study were to

1. Explore and classify near term (lower risk) requirements and opportunities for addressing GIG performance, scalability, and security,
2. Develop prototypical proofs-of-concepts and evaluation aimed at establishing a baseline for technology in the area of QoS-enabled JBI, and
3. Produce annotated brief-outs and a final report to guide future R&D activities.

Our team evaluated the technical options and challenges associated with addressing DoD QoS requirements in the context of GIG and NCES architectures – notably the Air Force Joint Battlespace Infosphere (JBI) – on software platforms based on more conventional architectures such as the most widely used object-oriented/component-based architectures (CORBA, J2EE and .NET) and the emerging Web Services and Publish/Subscribe (Pub/Sub) architectures, which we collectively refer to as Service-Oriented Architectures (SOAs). The results of our investigations found that while the match between GIG/NCES systems and standard commercial-off-the-shelf (COTS) SOA platforms holds much promise for moving the DoD away from traditional proprietary, stove-piped systems, SOA technologies have significant shortcomings with respect to their QoS support that can impede attempts to implement a GIG or NCES system directly on today's commercial SOA platforms.

The most serious deficiencies with SOAs are associated with QoS issues arising in systems that perform time-critical event reporting, and that must offer robustness to failure and attack, self-management, scalability, high-availability and advanced security features. Jointly, these

limitations suggest that the development of GIG/NCES systems over COTS SOA platforms could be a daunting task. Yet the potential benefits of doing so would be enormous since this would make it possible for DoD system integrators to leverage the millions of lines of commercial code-base expected in the SOA area and to work with well-supported commercial products, thereby reducing total ownership costs for DoD Armed Services.

Our Prometheus investigation arrived at two types of recommendations for the Air Force and the DoD:

1. A description of technical objectives that can be satisfied by working directly with SOA platform features that either already exist in COTS products, or can reasonably be anticipated in the two-to-three year time horizon, as well as an empirically-based understanding of the extent to which QoS and other properties can be addressed in limited ways by these COTS platforms.
2. Identification of open questions on which fundamental research may be required to close the gap between capabilities of existing SOA platforms (even including anticipated platform extensions) and requirements that arise in DoD contexts.

As a result of our work on Prometheus, we believe there are compelling reasons to look further at SOA platforms, identify specific deficiencies, and then conduct additional R&D to determine how existing technologies (or ones that might feasibly be developed within a few years) could be used to overcome today's limitations. Such a process would lead to a QoS-enabled GIG implemented as much as possible on SOA platforms, but extended in ways that respond to stringent DoD requirements.

Overview of the Prometheus Project

1. Motivation and Approach

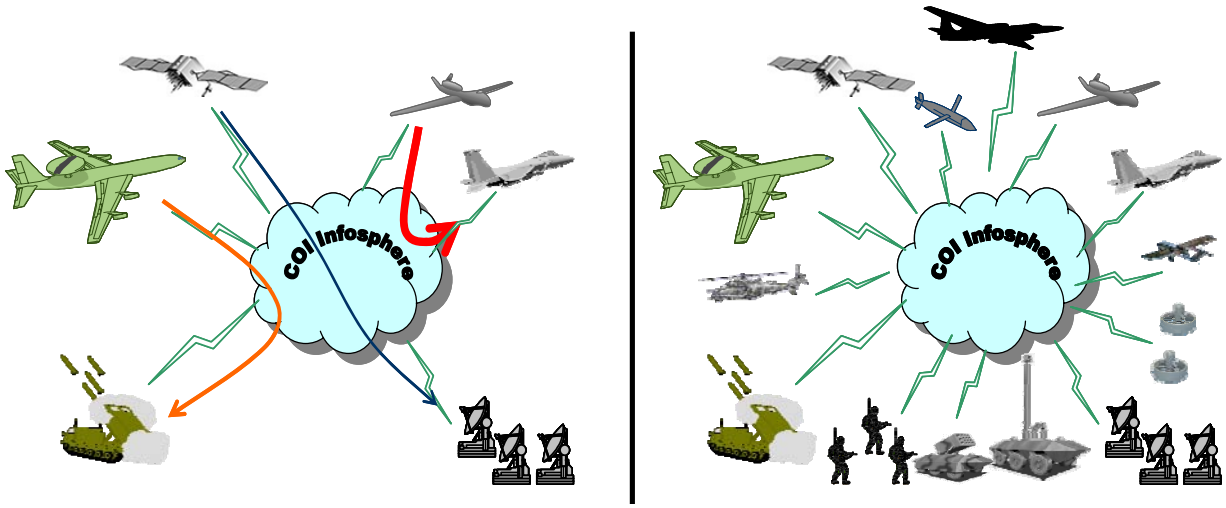
Future DoD missions will run on *system of systems* (SoS) characterized by thousands of platforms, sensors, decision nodes, weapons, and warfighters connected through heterogeneous wireline and wireless networks to exploit information superiority and achieve strategic and tactical objectives. The networks, operating systems, middleware, and applications that populate SoSs offer a combinatoric number of configuration points for adjusting their resource requirements and the quality of service (QoS) they deliver. The Global Information Grid (GIG) is an emerging DoD SoS intended to organize and coordinate this large application and technology space to manage information effectively and provide DoD planners and warfighters with the right information to the right place at the right time across a range of enterprise business systems and battlefield tactical systems.

This is an ambitious and intrinsically difficult goal. The technology base that inspires the SoS vision is a largely commercial product line created by vendors in support of commercial computing applications. Such applications run with only modest security, in networks that are relatively stable, and where few applications face demanding quality of service requirements, such as the need to deliver media with good real-time properties. Even the very largest commercial systems (data centers operated by companies such as Amazon or Google) are relatively "calm" environments when compared with the DoD networking infrastructure during some future crisis. Yet the economic and productivity benefits of working with a COTS technology base, modified in only limited ways, compel us to imagine building the future GG SoS platform with precisely this set of components.

To successfully support enterprise and tactical information management needs, the emerging GIG SoS technologies must provide (1) universal – yet secure – access to information from a wide variety of sources running over a wide variety of hardware/software platforms and networks, (2) an orchestrated information environment that aggregates, filters, and prioritizes the delivery of this information to work effectively in the face of transient and enduring resource constraints, (3) continuous adaptation to changes in the operating environment, such as dynamic

network topologies, publisher/subscriber membership changes, and intermittent connectivity, and (4) tailorable, actionable information that can be distributed in a timely manner in the appropriate form and level of detail to users at all echelons.

As the Air Force and AFRL work to implement and build upon the GIG architecture and vision, they will confront significant challenges in the area of tactical information management. In particular, the commercial Web Services technologies and standards on which the GIG is currently based cannot address the real-time quality of service (QoS) requirements of tactical information management, several of which are shown in Figure 1, as described below. Conventional Web Service technologies are well matched to enterprise environments, such as centralized data centers whose computers are connect via wireline networks, but are poorly suited for tactical environments due to their lack of QoS policies and real-time operating system integration, as well as high time/space overhead.



(a) Differentiated QoS Flows Through the GIG (b) Large Number of Clients & Resources

Figure 1: QoS Needs in the GIG

The Prometheus project was a 9-month study of the performance and scalability challenges currently confronting the GIG from a perspective informed by state-of-the-art insight in real-time, security, and reliable computing technologies. Our goal was to explore Air Force needs against the context of these COTS solutions and offer recommendations to the Air Force where gaps could be identified. We believe that the Air Force has gained a deep and subtle

understanding of the challenges surrounding information management in GIG settings. There remain, however, the following unique challenges that must be overcome to fully realize the promise of the GIG in large-scale DoD C2 and C4ISR systems-of-systems deployments:

- ***DoD applications generate high data rates, often with time-sensitivity or other QoS requirements***. The GIG will need the capability to offer these types of guarantees, i.e., such that the right data will reach the right consumers where and when required and that critical QoS properties will be assured even under attack or other stresses. Initial work on the GIG deferred consideration of these properties, but the time is right to tackle them now. Figure 1(a) illustrates the need for soft real-time quality of service (QoS) properties, such as priorities and differentiated flows of information through the GIG.
- ***The GIG by its very nature must scale to some very ambitious*** *configurations* [14][15][12]. For example, even early deployments will have hundreds of users producing and consuming data, and longer term goals will require scaling to thousands or even hundreds of thousands of users, as shown in Figure 1(b). When the GIG technology base is initially delivered and as later improvements are made, it will be necessary to evaluate the scalability properties of the technology platforms against the requirements seen in DoD scenarios to identify gaps. Where these are found (and we anticipate many of them), Air Force must ask how to integrate state-of-the-art insights and capabilities into scalable, reliable and self-managed systems, both in the context of GIG solutions from NSA and DISA, and in the context of major Air Force technologies such as the JBI architecture and its Gemini reference implementations.

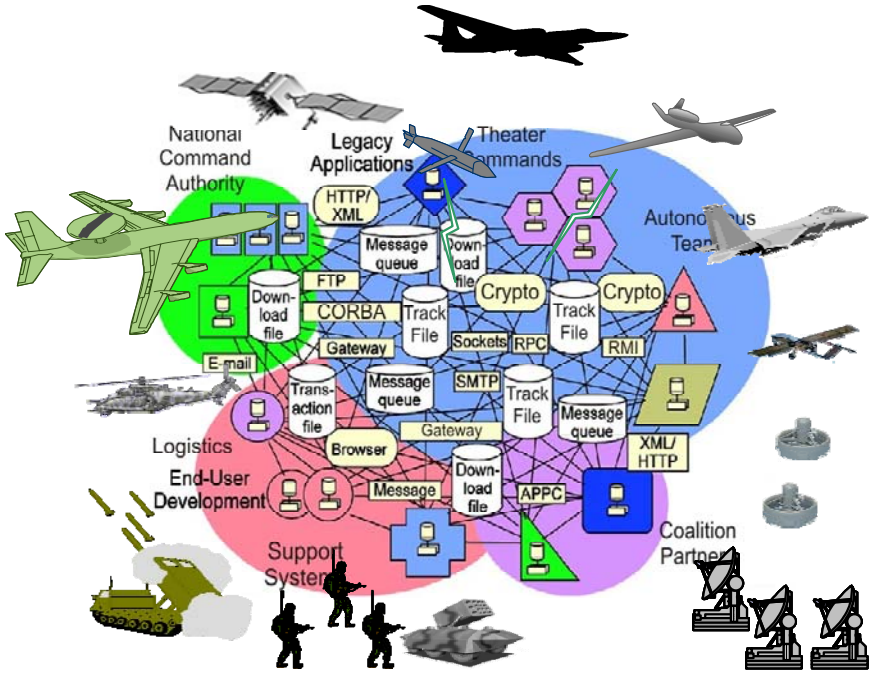


Figure 2: Heterogeneity and Compartmentalization in the GIG

- ***All aspects of the GIG architecture must be viewed in light of the DoD's unique security requirements and operational environments***, whose heterogeneity and multiple levels of security and compartmentalization far exceed anything seen in commercial platforms and systems, as shown in Figure 2. We need to learn how to "ruggedize" DoD net-centric systems-of-systems, in general, and the GIG, in particular, to provide QoS and other performance guarantees even in light of denial of service attacks or platform penetration by insiders. This is an unfortunate conclusion because it is generally recognized that security must be designed in from the outset if one is to hope for a genuinely secure solution. In the case of the GIG, it seems clear that the off-the-shelf components and legacy content of the platform will preclude a first-principles approach to security.

In the Prometheus study, these questions were posed in the light of broader industry trends [12][13][14][15] to consolidate distributed object computing architectures (such as J2EE and CORBA), Pub/Sub platforms (such as JMS and DDS), and Web Services environments (such as .NET and SOAP) within QoS-enabled Service-Oriented Architecture (SOA) platforms and standards. We therefore focused on what QoS (i.e., performance guarantees, scalability, and

security) should mean for the GIG, as well as how to implement these properties in a GIG built using QoS-enabled SOA platforms. We believe this to be a particularly timely issue: the SOA community hopes to tackle such questions but has its hands full with concerns of a more commercially immediate nature, and the major vendors seem not to even have such questions on their radar screens. Demonstrated success stories in the GIG context could let the military exert leadership within the relevant standards organizations, such as the World-Wide Web Consortium (W3C) and the OMG. Moreover, since military requirements often go well beyond commercial requirements, there may be an opportunity to push the SOA standards community further than it might go under the pressure of commercial market incentives.

To illustrate our general approach, consider the following example. SOA platforms offer reliability mechanisms, and reliability is clearly a requirement for many anticipated GIG applications. But this superficial match does not really answer the key question: are SOA reliability mechanisms adequate to satisfy the full range of GIG QoS needs? If one looks closely at the emerging WS_RELIABILITY standard [12], for example, one can contrast the intuitive expectation given the name of the standard and the reality of what the standard provides. Based on its name, one would assume that WS_RELIABILITY is a general purpose reliability specification for web services, addressing such aspects as service availability and recovery from faults, status of client requests interrupted by a crash, and so forth. But the actual standard is quite a different matter. First, we find that it is limited to a rather narrow style of application (i.e., one in which a client is trying to send a request to a server for "out of band" processing), and focused on a specific scenario (i.e., one in which a queuing mechanism has been interposed between the client and the server) [13][14][15]. Admittedly, this is an important case, but it is hardly a very general one. At any rate, in this specific situation, WS_RELIABILITY defines the guarantees available to a client, namely that the request will be processed at least once, at most once, or exactly once, and details the associated mechanisms, which are increasingly visible to the application designer and costly as we seek stronger guarantees. The standard says nothing at all about any other form of reliability – it simply permits the client to specify desired behavior from a queuing intermediary in a pipelined processing setting. Moreover, it says nothing about how an application should implement the requested properties. Thus a client could request

"exactly once" semantics with no way of knowing how this kind of reliability will be provided, what the cost implications might be, or indeed, whether or not the property is even achievable.

Given this example, we can now undertake the style of analysis outlined above. For some purposes, WS_RELIABILITY should be a good match. For example, a time-critical QoS system might have deadlines measured in the hours, and for such uses WS_RELIABILITY may be adequate. The Prometheus study explored some of these good-news stories, and we undertook some preliminary experimental work to understand the associated costs and performance implications. We also discovered, however, that WS_RELIABILITY was inadequate for other purposes, e.g., where QoS translates to tight real-time bounds, softer time-critical event delivery constraints, continuous or high availability, etc. In these cases, we considered how the stronger requirement could be addressed. Indeed, we can identify a great variety of plausible reliability needs and we find that the web services specifications address almost none of them.

Security is another fundamental need in DoD GIG applications, but the topic is extremely broad and a serious treatment of SOA security was beyond the scope of our effort. Accordingly, we focused on exploring security issues specific to the QoS mechanisms on which we focus. Moreover, we viewed the term "security" in a broad sense, i.e., a secure system for us will be one that maintains its QoS properties while (1) limiting unauthorized data access (e.g., through cryptographic means) and (2) tolerating intrusions into portions of the system (e.g., so-called "Byzantine" failures). In this broad sense, the ability of a SoS to diagnose problems is a security property, as are its repair and management characteristics. In effect, we were interested in SoS that are secure enough to only do what they are expected to do, and that can also be counted upon to automatically handle minor forms of damage.

2. Summary of Results

It would be far too easy for a study such as ours to content itself with a laundry list of criticisms. More challenging is to prioritize and identify low-hanging fruit: opportunities for targeted investment that might have a big impact on the capabilities of existing platforms at modest cost to the Air Force. Accordingly, the task objectives of the Prometheus study were to explore and

classify near-term (lower risk) opportunities for addressing the primary challenges in the context of the following four GIG focus areas:

- QoS-enabled Publish/Subscribe Technologies for Tactical Information Management
- Scalable Fault- and Intrusion-Tolerance for Critical GIG Services
- Scalable Enterprise Service-Oriented Architectures
- A Framework for Demonstrating Access Control Policy Compliance

As described below, our results included prototypes, experiments, and analyses;, as well as a recommend roadmap for next steps by the AFRL development and research team and their academic and industrial partners.

2.1. Focus Area 1: QoS-enabled Pub/Sub Technologies for Tactical Information Management

Tactical information management systems communicate by publishing the information they have and subscribing to the information they need in the face of interactive and/or autonomous adaptation to fluid environments. To realize the promise of tactical information management for the GIG in large-scale DoD C2 and C4ISR SoS deployments, the GIG will need the capability to offer certain types of assurance, e.g., the right information will reach the right consumers where and when required and that critical QoS properties will be assured to an appropriate degree even under attack or other resource constraints. Initial work on the enterprise portions of the GIG deferred consideration of these properties, but recent advances in QoS-enabled platform technology and tools warrants revisiting solutions to these challenges.

QoS can mean many things. Since the Prometheus effort focuses on tactical information management in the GIG, we emphasize publish/subscribe event notifications whose latency can be assured, even in the presence of some degree of disruption, online reconfiguration, network congestion, failures, or even limited attacks. Since the GIG is inherently a SoS architecture this implies that cross-system overheads and protocols could have an impact on QoS, hence we focused on identifying and evaluating such issues.

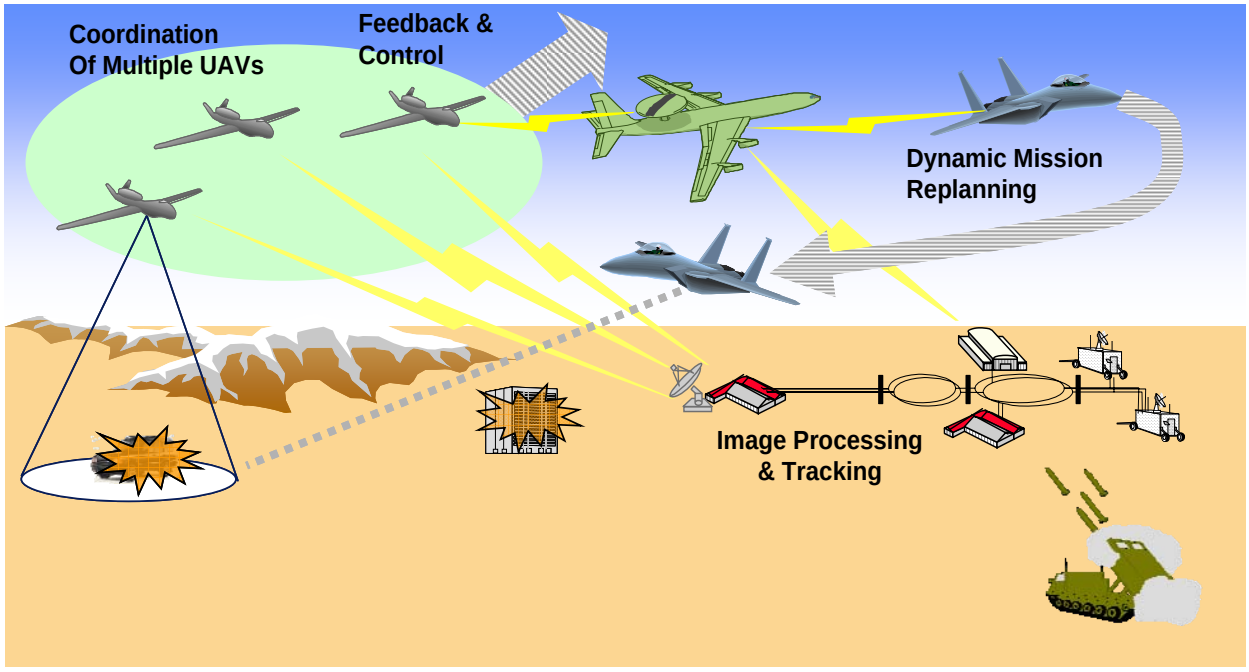


Figure 3: PCES Capstone Demo Scenario

Portions of our work on the Prometheus project were motivated by prior work in the DARPA *Program Composition of Embedded Systems* (PCES) program, which the principal investigator for this Prometheus effort, Dr. Doug Schmidt, ran during his stint as a Program Manager at DARPA's Information eXploitation Office (IXO) from 2000 to 2003. Figure 3 illustrates the interactions between assets used in the PCES Capstone demonstration at White Sands Missile Range (WSMR) on April 14, 2005. This demonstration prototyped multiple types of time-critical targeting services using standards-based enabling technologies and evaluated their ability to meet the QoS needs of network-centric systems distributed over a wide area, and networked together over 100 miles. The following live equipment was used in the demo:

- Two ScanEagle Unmanned Air Vehicles (UAVs)
- A High Mobility Artillery Rocket System (HIMARS) wheeled variant of the Multi-Launch Rocket System (MLRS) and
- An F-16 strike fighter.

The UAVs in the PCES Capstone demo communicated over Link-16, which is a tactical military network with insufficient bandwidth to provide streaming video for all assets. The demonstration scenario started with multiple UAVs in the air doing reconnaissance, followed by the appearance of multiple pop-up targets that were prosecuted by the PCES Operations Center Commander (OCC) who has the ability to task UAVs and designate targets for tracking and engagement. When a stationary target was identified as harboring a meeting of hostile combatants and authorized for prosecution, the OCC tasked the HIMARS MLRS to destroy the target. Battle damage assessment conducted in real-time using images from the UAVs indicated that some hostile combatants escaped in a truck. Using information obtained from a UAV assigned to track the fleeing truck, the OCC then retargeted an F-16 strike fighter, which deployed a Joint Directed Attack Munition (JDAM) to destroy the mobile target.

The PCES Capstone demo evaluated the use of standards-based COTS middleware technologies. For example, real-time information to the cockpit of the strike fighter used Real-time CORBA and the Real-time CORBA Event Service to communicate over a Link-16 tactical network. Likewise, the C2 & C4ISR information management used to exchange track information within the operations center was based on the Joint Battlespace Infosphere (JBI), which uses Java 2 Enterprise Edition (J2EE) and the Java Messaging Service (JMS). Figure 4 illustrates the layered architectures used to provide real-time information to the cockpit and track processing in the PCES Capstone demonstration.

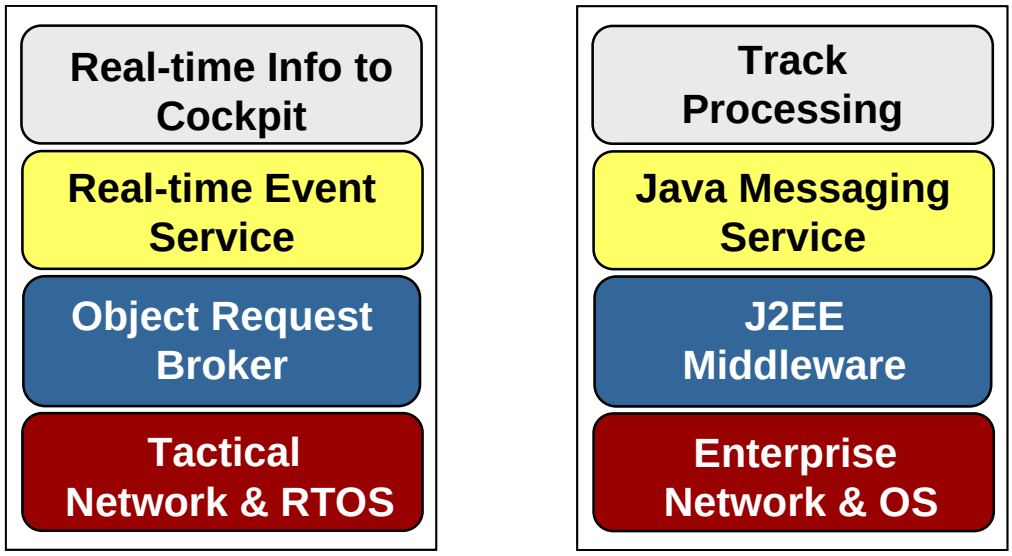


Figure 4: Technology Layers in the PCES Capstone Demo

Although the technologies shown in Figure 4 represent advanced middleware technologies in the context of tactical information management in the DoD, our experiences on the PCES Capstone demonstration revealed the following limitations:

- Real-time CORBA and the Real-time CORBA Event Service are platform-centric technologies that are well-suited for point-to-point and static pub/sub command processing over wireline networks [34][49][28]. They were poorly suited, however, for dynamic pub/sub info dissemination to multiple nodes in mobile networks due to too many layers, excessive time/space overhead, inflexible QoS policies and pub/sub model, and inadequate support for surviving cyber attacks.
- The J2EE and JMS technologies are well-suited for operational enterprise environments, such as large data centers connect via wireline networks. They were poorly suited, however, for tactical environments due to the lack of QoS policies, lack of real-time operating system integration, extremely high time/space overhead, and inadequate support to survive cyber attacks.

Our PCES experience therefore indicated the need for information management technologies that can function properly in tactical SoS where communication bandwidth is limited/variable,

connectivity is intermittent, connections are noisy, processing and storage capacity is limited, power and weight limits affect usage patterns, unanticipated workflows are common, and dynamic network topology and membership changes are frequent. Since connectivity cannot be guaranteed at all times in such environments, a realistic goal is to provide "better than best effort" QoS support, subject to network and platform resource constraints. Ultimately, such technology solutions should also align with – and influence – COTS standards and integrate with heterogeneous legacy systems.

As part of the Prometheus project, we evaluated the Object Management Group (OMG)'s Data Distribution Service (DDS), which is a standards-based and COTS Pub/Sub infrastructure available from multiple suppliers. Figure 5 illustrates the architecture of DDS and shows how it is designed to support

- *Location independence*, via anonymous pub/sub protocols that enable communication between publishers and subscribers,
- *Scalability*, via supporting any number of topics, data readers, and data writers, and
- *Platform portability and protocol interoperability*, via standard APIs and transport formats.

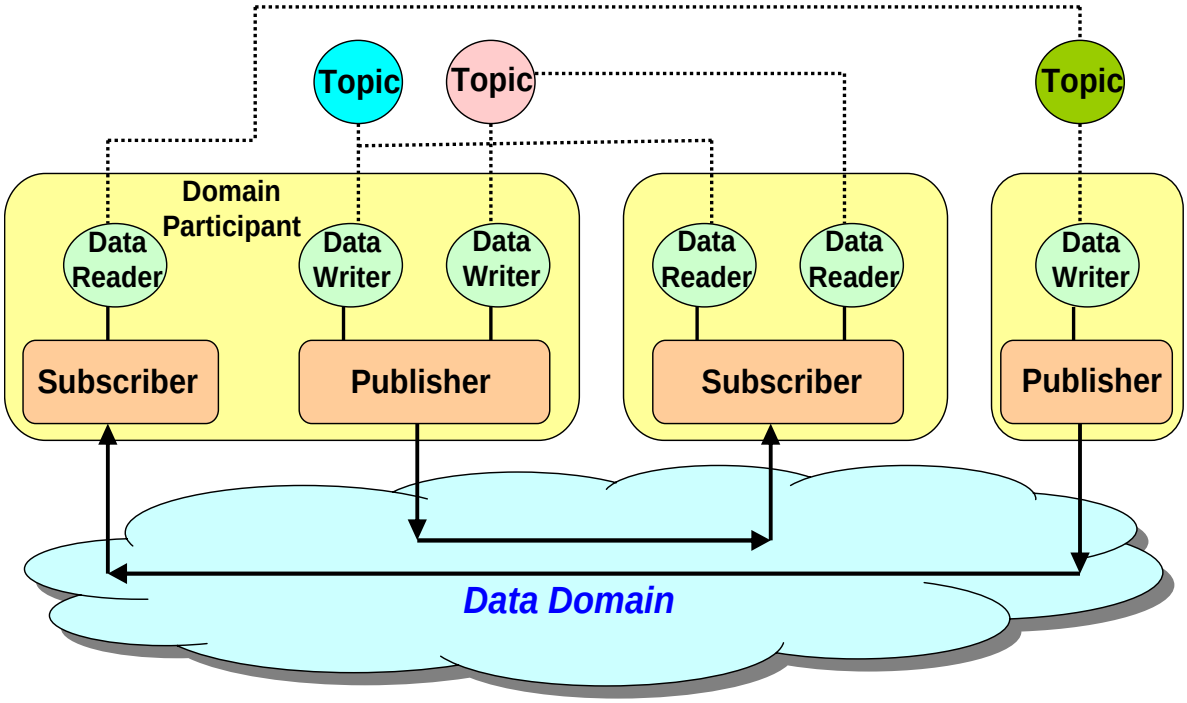


Figure 5: OMG DDS Architecture

The OMG DDS specification has been adopted in the past several years based on decades of experience with proprietary Pub/Sub communication platforms, such as NDDS from Real-Time Innovations and SPLICE from Thales. As a result, the technologies underlying the DDS specification have had a long time to mature in the context of production distributed, real-time, and embedded systems. In particular, DDS implementations are used in a wide range of DoD systems including DD(X) Ship Self Defense System (SSDS), Littoral Combat Ship (LCS), Spy OA, Sea Slice, LPD 17, the Joint Strike Fighter (JSF), and the Future Combat Systems (FCS) program.

There are several DDS capabilities that make it better suited as the basis of tactical information management than the standard COTS middleware showcased in the PCES Capstone demo. Figure 6 shows that DDS has *fewer layers* in its architecture than the CORBA and J2EE technologies shown in Figure 4. In particular, the DDS Pub/Sub service is implemented in the core of the middleware, rather than being implemented as a layered service, as is the case with CORBA and the CORBA Event Service. This reduction in layers significantly reduces the

latency/jitter and increases the scalability of DDS, as shown by the empirical results shown in Figures 12, 13 and 14 later in this report.

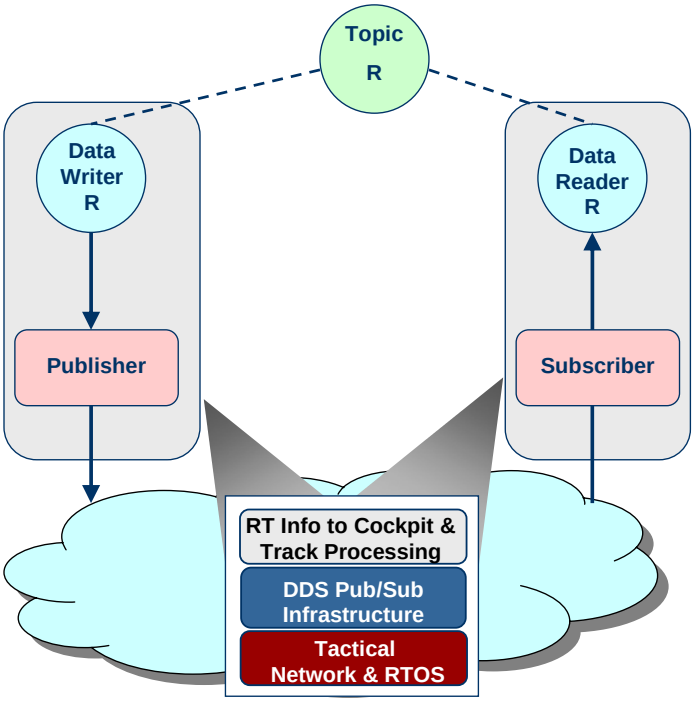


Figure 6: Fewer Layers in the DDS Architecture

Figure 7 shows that DDS supports many *QoS policies* that can be tailored to meet the data distribution requirements of tactical information systems. These QoS policies can be configured at multiple layers (e.g., middleware, OS, and network) and DDS entities (e.g., topics, data readers, and data writers). Some relevant QoS policies include:

- DEADLINE, which establishes a contract regarding the rate at which periodic data is refreshed
- LATENCY_BUDGET, which establishes guidelines for acceptable end-to-end delays
- TIME_BASED_FILTER, which mediates exchanges between slow consumers and fast producers
- RESOURCE_LIMITS, which controls memory and CPU resources utilized by DDS entities
- RELIABILITY, which enables a trade-off between best effort and reliable data transport
- HISTORY, which controls which (of multiple) data values are delivered

- DURABILITY, which determines if data outlives time when they are written. DDS can automatically check the compatibility of QoS policy requests and offers.

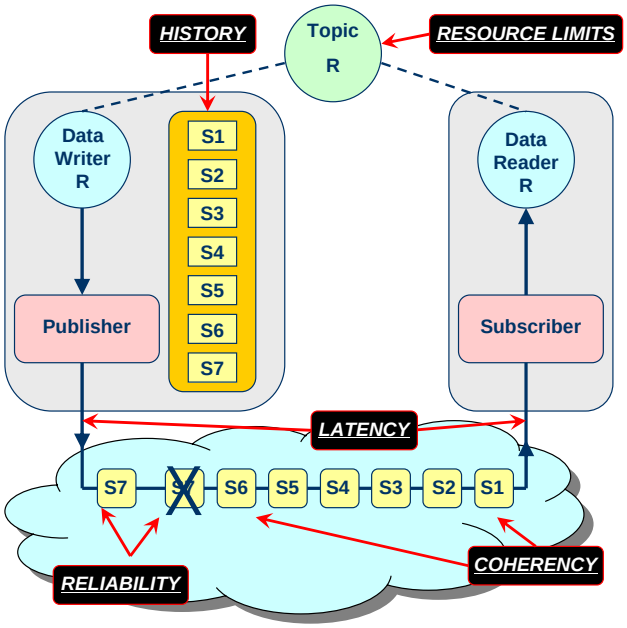


Figure 7: DDS QoS Policies

Figure 8 shows how DDS's sophisticated filtering capabilities can *move processing close to the data*, which reduces network bandwidth in resource constrained tactical links. Techniques for negotiation and placement of filtering functionality at optimum positions within infrastructure and data flows to deliver information only (1) when it is needed, (2) if it is needed, or (3) at requested frequencies.

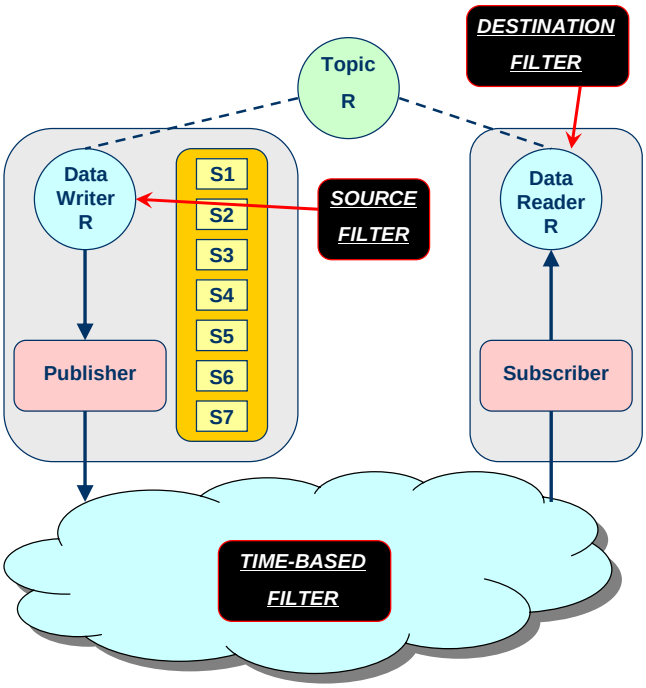


Figure 8: Moving Processing Close to the Data

Figure 9 shows how DDS enables clients to *subscribe to meta-events* that they can use to detect dynamic changes in network topology, membership, and QoS levels.

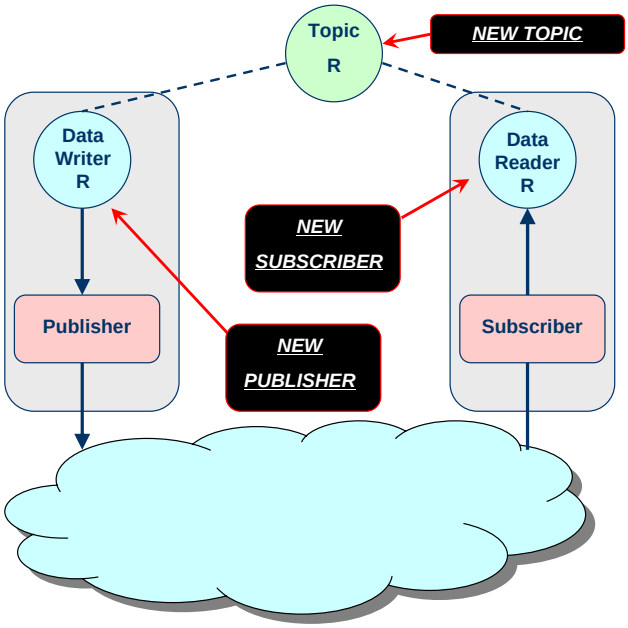


Figure 9: Subscribe to Meta-Events

During the Prometheus project we conducted many experiments to evaluate the pros and cons of DDS relative to other communication models using the ISISlab testbed shown in Figure 10.

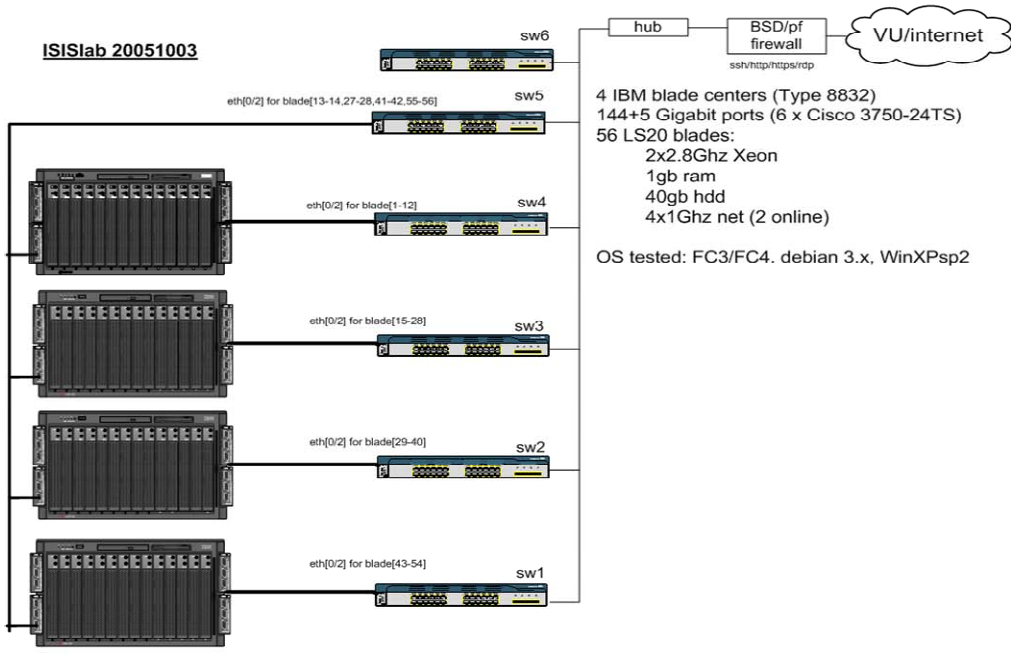


Figure 10: ISISlab Testbed

The ISISlab testbed contains 56 dual-CPU Intel(R) Xeon(TM) CPUs running at 2.80GHz with 1GB ram. These machines ran Linux version 2.6.14-1.1637_FC4smp and used g++ (GCC) 3.2.3 20030502. We used this testbed to conduct experiments using three COTS implementations of DDS (NDDS, SPLICE and TAO DDS) and compared their performance relative to other popular platforms, such as the Java Messaging Service (JMS), CORBA Notification Service, and gSOAP. The goal of these experiments was to evaluate the importance of tactical information management solutions in light of prevailing technology trends and emerging standards and COTS platforms and tools. Our goal was to ensure that the military can leverage the trends, and also to ensure that if the SOA and Pub/Sub technologies are overlooking questions of importance to the military, adequate energy can be directed towards those questions early so that solutions will be in hand if and when needed. Since military requirements often go well beyond commercial requirements, it will be necessary to push the COTS standards communities further than they might go under the pressure of conventional commercial market incentives.

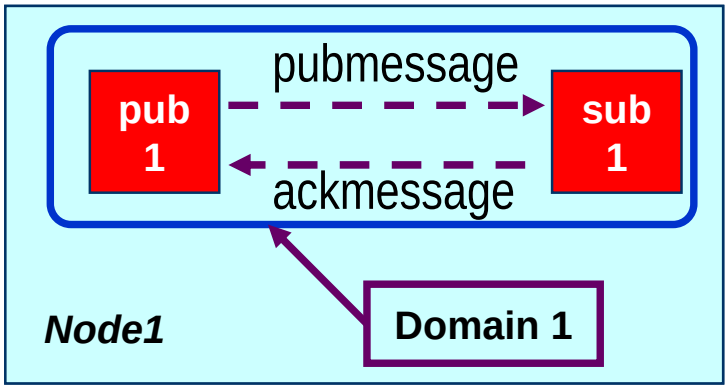


Figure 11: Pub/Sub Application Design

The first round of experiments we ran were organized as shown in Figure 11. Two processes in a single DDS domain perform interprocess communication (IPC), in which a client initiates a request to transmit a number of bytes (e.g., approximating a message containing XML-encoded data) to the server along with a seq_num (pubmessage), and the server simply replies with the same seq_num (ackmessage). The invocation is essentially a two-way call, i.e., the client/server waits for the request to be completed. The client and server processes are collocated on the same node since our goal in these experiments was to evaluate the overhead incurred by the various communication mechanisms. DDS and JMS provides topic-based pub/sub model, the TAO Notification Service uses push model, and SOAP uses P2P schema-based model.

Figure 12 presents results that compare round-trip latency for all the communication mechanisms. The differences are compressed by log scale, e.g., latency for the TAO Notification Service is about 4 times that for DDS2 at small message sizes, and likewise JMS is 3 times that of the Notification Service. To eliminate 'cold start' effects we ran 10,000 iterations at each message size, with 100 iterations 'warmup' before measuring the results. The figure shows that DDS has significantly lower latency than other Pub/Sub services. Although there is variation in the latency of the DDS implementations, they are all at least twice as fast (and in some cases dozens of times faster) than other Pub/Sub platforms as the amount of data increases. As a result, the delay before the application learns critical information is very high with conventional Pub/Sub mechanisms, whereas with DDS latency is low across the board.

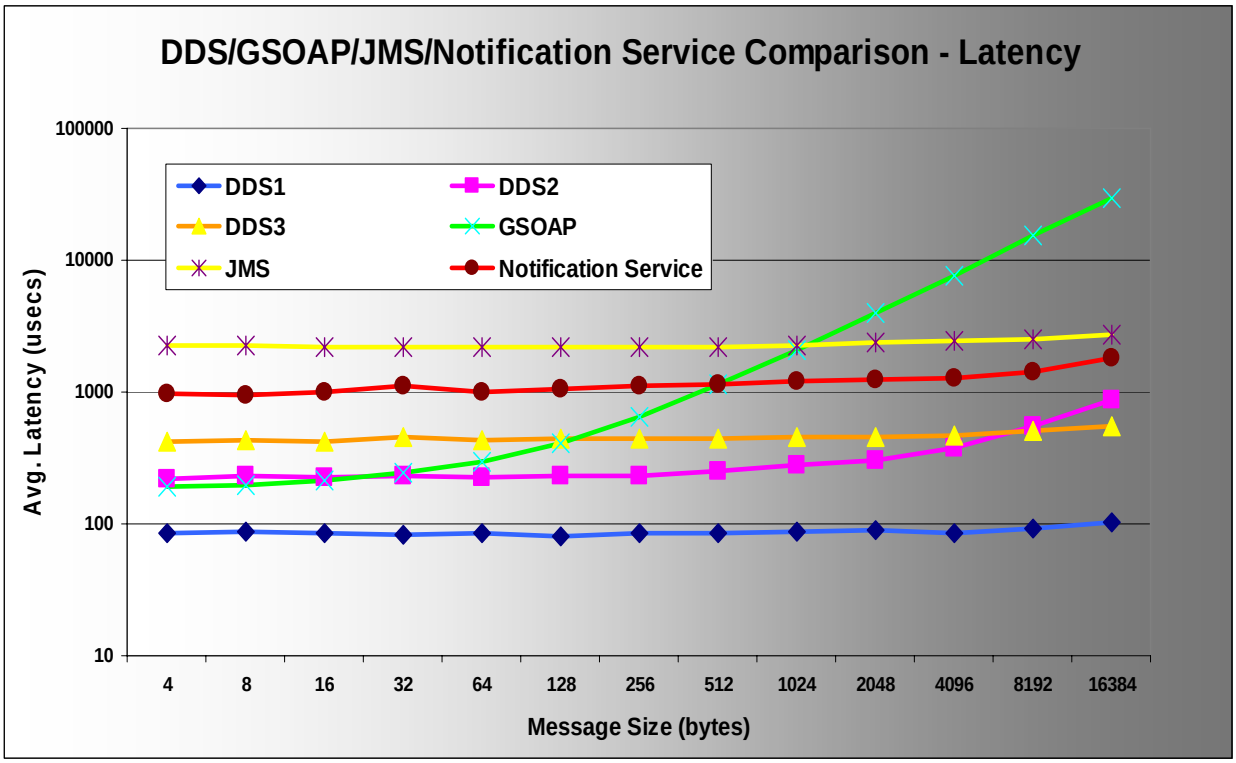


Figure 12: Comparing Latency for Simple Data Types on Middleware Platforms

Figure 13 presents results that compare round-trip jitter (i.e., variation in the latency) for all the communication mechanisms. Although there is variation in the jitter of the DDS implementations, they are considerably more predictable than other Pub/Sub platforms as the amount of data increases. As a result of the high jitter in conventional Pub/Sub mechanisms they are not well suited for tactical information management systems.

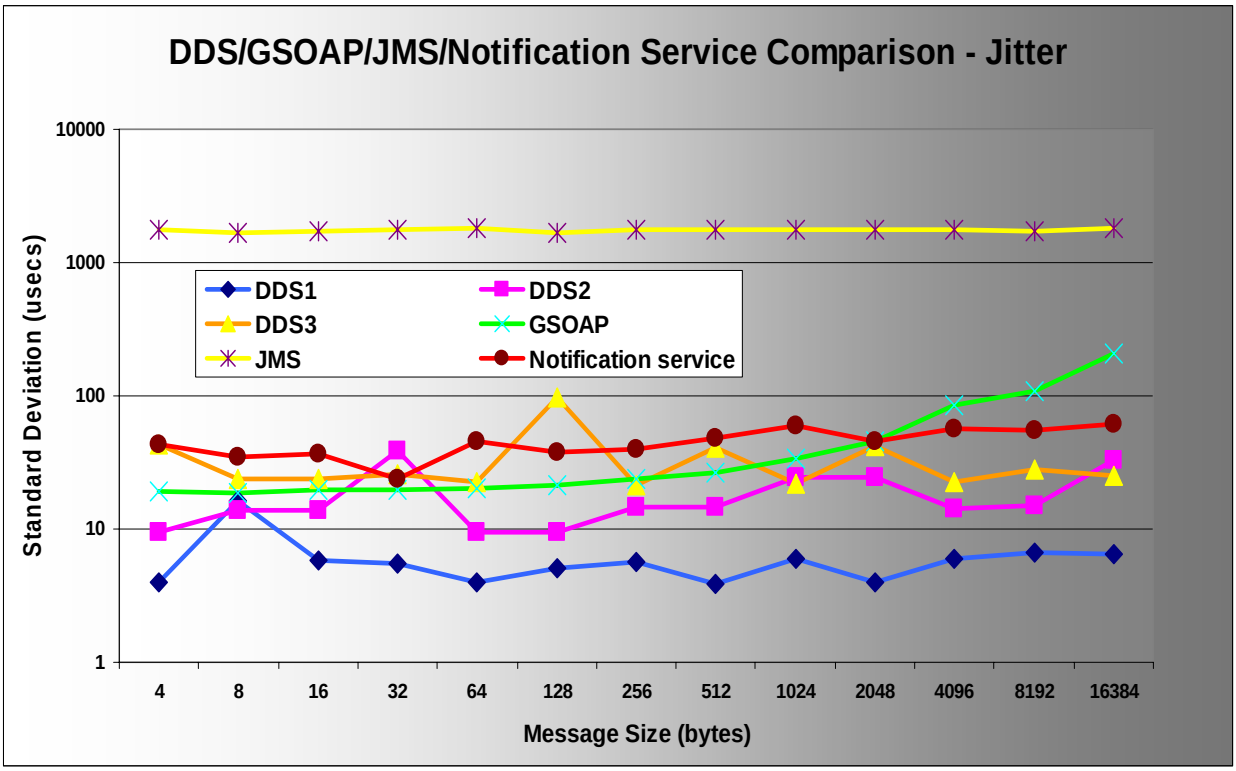


Figure 13: Comparing Jitter for Simple Data Types on Middleware Platforms

We also ran experiments using more complex data types, such as the following:

```
// Complex Sequence Type
struct Inner {
   string info;
   long index;
};
typedef sequence<Inner> InnerSeq;
struct Outer {
   long length;
   InnerSeq nested_member;
};
typedef sequence<Outer> ComplexSeq;
```

This data type is significantly more complicated than the simple sequence of bytes we used in the previous tests. As a result, it exercises the data marshaling and de-marshaling capabilities of the

various middleware platforms more thoroughly. Figure 14 shows the latency results of this data type for DDS and GSOAP middleware. These results illustrate that even for complex data types DDS is more efficient than SOAP.

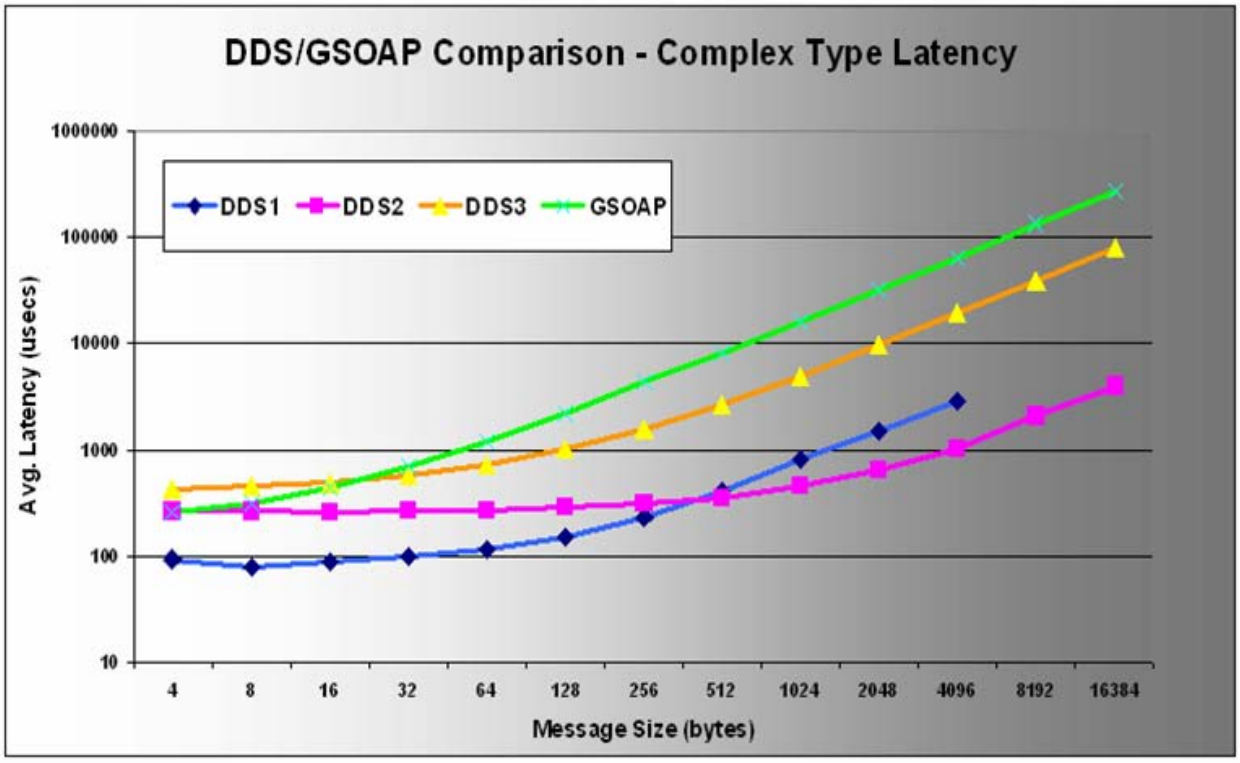


Figure 14: Comparing Latency of Complex Data Types on Middleware Platforms

Although these experiments just example a few of the many important QoS issues in the GIG, their results underscore the importance of evaluating tactical information management solutions in light of prevailing technology trends and emerging standards and COTS platforms and tools. The goal of Prometheus was to help ensure that the military can leverage the trends, and also to ensure that if the SOA and Pub/Sub technologies are overlooking questions of importance to the military, adequate energy can be directed towards those questions early so that solutions will be in hand if and when needed. Since military requirements often go well beyond commercial requirements, it will clearly be necessary to push the COTS standards communities further in the future than they might go under the pressure of conventional commercial market incentives.

Addressing the challenges described above is the underlying theme of the recommended R&D activities presented Section B.3.

For example, existing implementations of the DDS standard differ widely in their handling of one-to-many communication patterns (e.g. situations where many receivers are monitoring a shared data feed). In such situations, the real-time delivery properties of a platform can easily be lost, for example if a single source must send data to a dozen separate receivers. This is an example of a challenge that arises not so much at the level of the standard per-se, but rather in the implementation of the standard. Needed, in this case, are scalable multicast technologies with extremely low delivery latency (delay); such technology could then be layered beneath a DDS front-end to offer the benefits seen in the experiments just reported, but without this kind of scalability concern. Among the challenges posed in B.3 is the one just summarized; we see it as a good example of a near-term opportunity that on the one hand demands some basic scientific research, but on the other should be readily solved with modest investment of resource, could easily be demonstrated, and hence that could easily transition into the vendor platforms most popular within the services once solved.

2.2. Focus Area 2: Scalable Fault- and Intrusion-Tolerance for Critical GIG Services

The second focus area of the Prometheus project is "intrusion tolerance," which is essentially fault-tolerance for an extreme form of fault in which a faulty component can behave arbitrarily, and even in coordination with other faulty components in an effort to violate the goals of the system. This type of behavior can model an application that has been compromised (e.g., via a software vulnerability or even physical capture), but is also a good way to model the possibility of data corruption or bugs, which can result in unpredictable behavior: If a system is provably capable of tolerating even deliberate attack, it should also be resilient to these kinds of mundane problems (which, nonetheless, fall outside of what many "fault-tolerant" systems are able to handle!) The clear reality is that mission-critical GIG services must survive both these forms of failures and also the possibility of outright attack, most likely just when reliability is most important.

Figure 15 illustrates how the GIG is vulnerable to failures due to faults and intrusions, where components are likely to come under forms of attack such as physical capture that are atypical in commercial systems. What is desired, of course, is an architecture in which correct behavior can be assured even if some percentage of applications or service representatives have been compromised or are experiencing arbitrary failures.

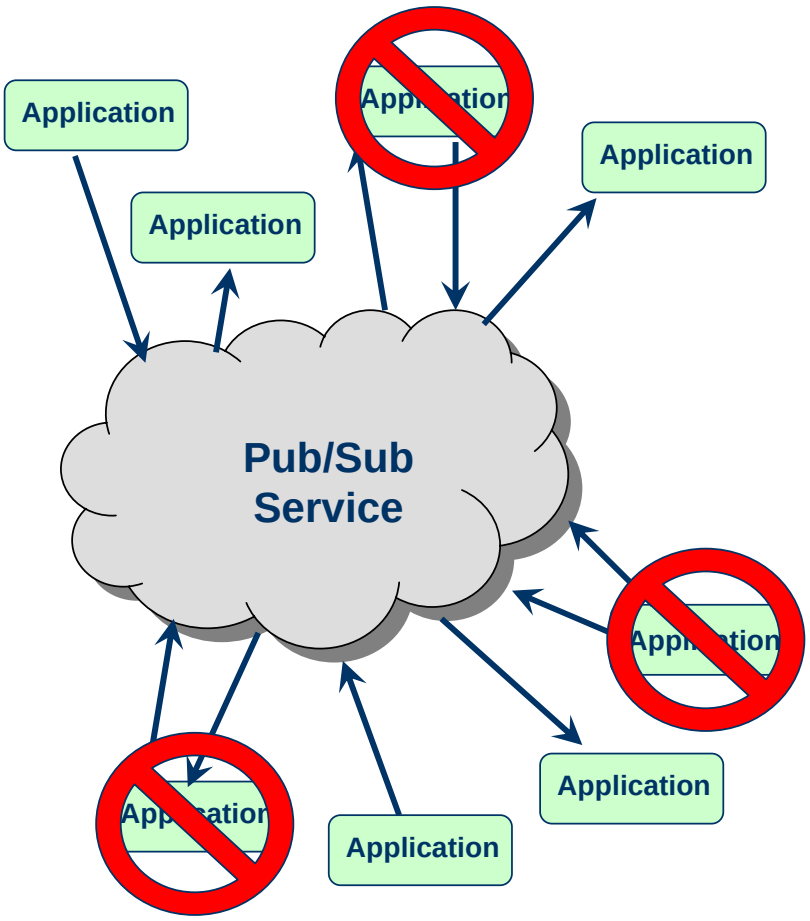
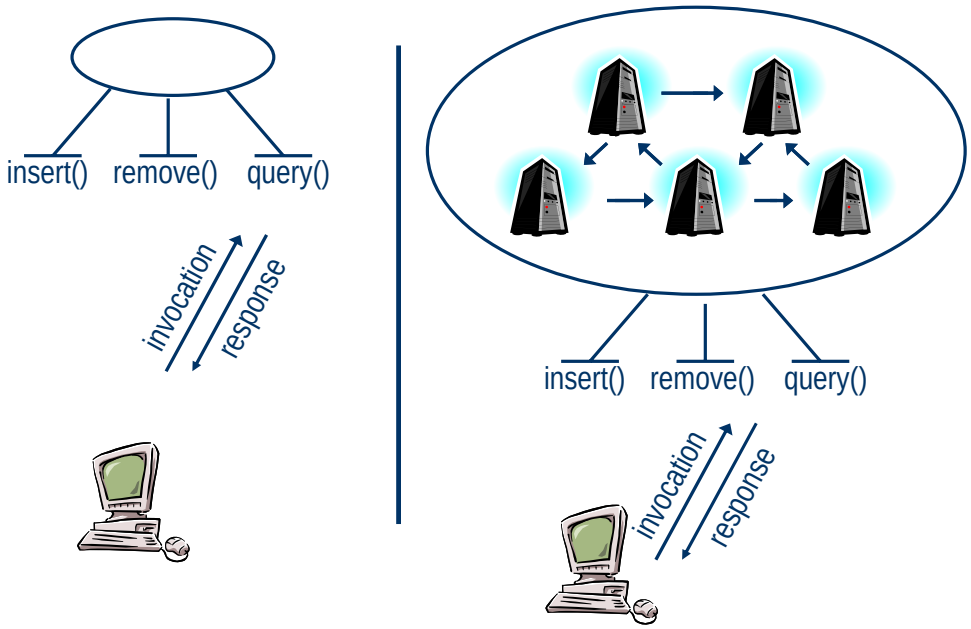


Figure 15: Protecting GIG Services from Failures due to Faults and Intrusions

Note that our wording emphasizes the need to *operate through* an intrusion. Traditional DoD SoS operate with barrier security: they erect a firewall to keep intruders out, but take no additional steps; if an intruder circumvents the firewall, his actions might be arbitrarily malicious. In our view, it is far more prudent to defend in depth: employ firewalls, but also

anticipate the event that components of a system are compromised by an attacker *despite* the firewall. In the "intrusion tolerance" mindset, we model this eventuality and build the system so as to withstand it. An *intrusion-tolerant* service is thus one that continues to operate correctly despite the corruption of some of its components [15][16][1][2][3][5][12][14][21][22].

Though an appealing goal, one trouble with intrusion-tolerant architectures in the past is that they were very costly to operate, and in fact performance would degrade dramatically as systems scaled up in distributed environments. Figure 16 gives some insight into the cause of this degradation, showing the distinction between the abstraction of a service and its distributed implementation.



(A) Top-level Service Abstraction (B) Implemented via Distributed Services

Figure 16: Distributed Services based on the SOA Model

As pictured on the left of Figure 16, the client need not even be aware that the service is implemented in a distributed way. "Under the covers" (pictured on the right), however, the distributed implementation is explicit, and in fact the coordination protocols that are involved in the implementation of an intrusion-tolerant service are typically complex and costly. The performance degradation of these protocols as the system scaled up helped justify eschewing intrusion-tolerance technologies, leaving the core system undefended under the argument that the

defense would have an intolerable cost. Our research in intrusion-tolerance during Prometheus therefore focused primarily on building systems whose services were more fault-scalable than prior approaches but without these sorts of unacceptable costs and overheads.

Historically, the most thoroughly studied approach to building intrusion-tolerant systems is state machine replication, shown in Figure 17. In this approach, client invocations are submitted to the servers using an *atomic multicast* protocol that ensures the delivery of all invocations to all (correct) servers in the same order. Provided that the servers are initialized to the same state, if they process invocations in the order received (and are deterministic), the correct servers will execute identically. In this way, each will provide the same response to the client, and the client can use these identical responses to "out-vote" the faulty servers, i.e., by accepting the response that occurs in the majority.

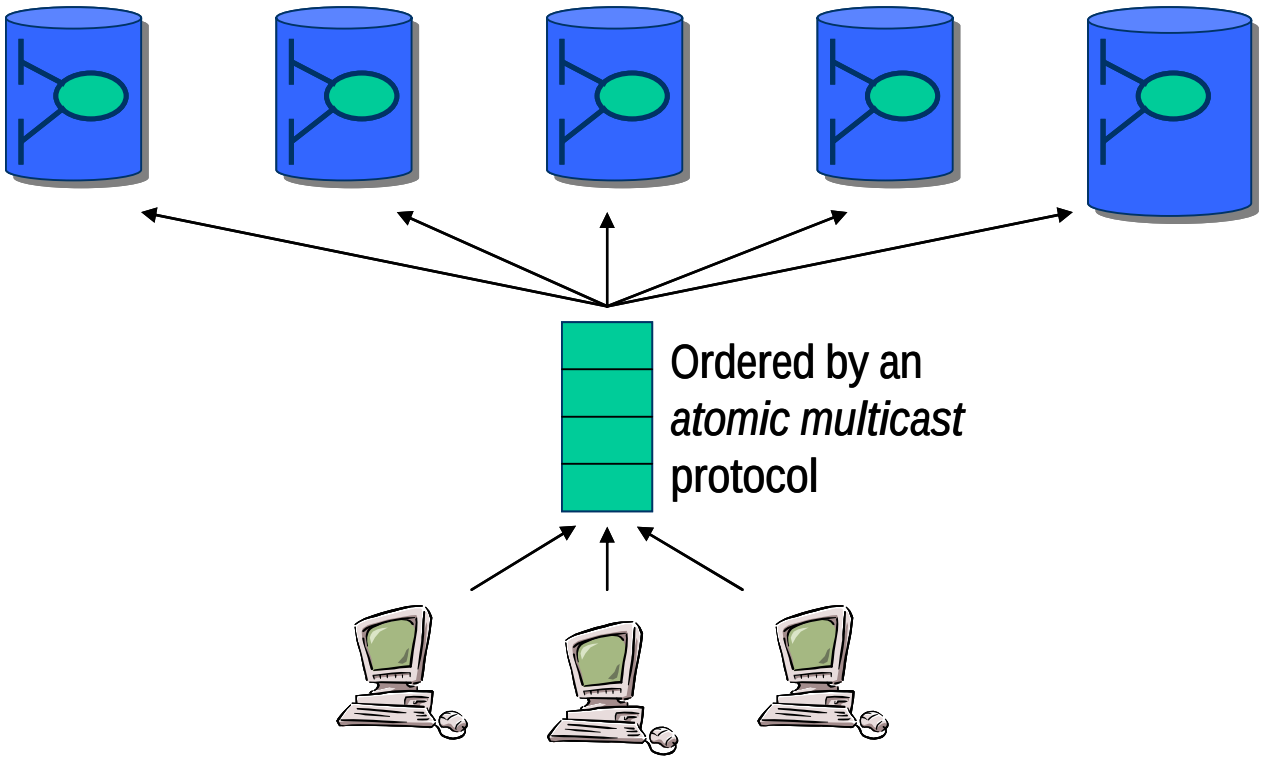


Figure 17: State Machine Replication

While effective for small service implementations, state machine replication does not scale well. In particular, since every request must be reliably delivered to and processed by every server,

adding servers to the service implementation does not improve its throughput. In fact, due to the additional messages involved in ordering each request among more servers, adding servers can significantly decay the performance of this approach.

To address these limitations, during Prometheus we studied alternative designs to state machine replication for building intrusion-tolerant applications. One such design is the "Query/Update" (Q/U) protocol, which offers a different set of tradeoffs than state machine replication. In the Q/U protocol, accesses are made to *quorums* of servers that can be much smaller than the full set of servers; this enables load to be dispersed across different quorums. In addition, the protocol is optimistic, incurring multiple rounds of communication only in uncommon cases. This optimism is enabled through versioning: updates to a server are not destructive, but rather create new versions at the server. A subsequent operation can then access multiple versions to determine which one is, in fact, the latest complete update. Finally, the protocol uses very lightweight cryptographic primitives to enhance performance.

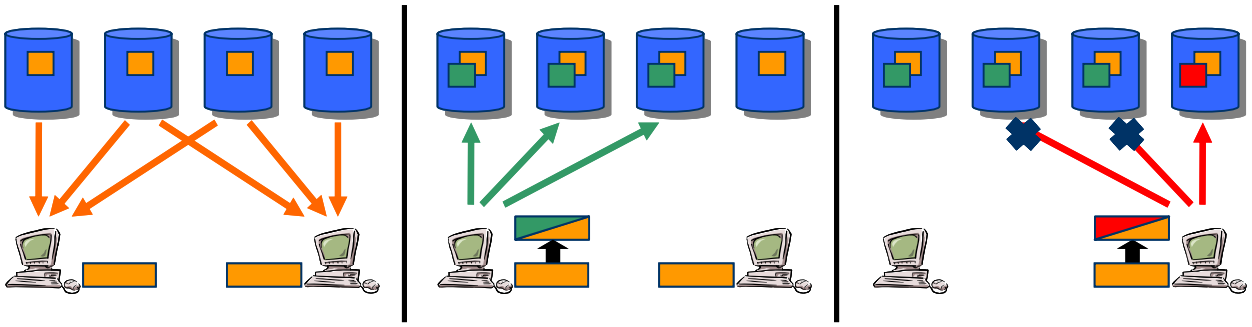


Figure 18: The Query/Update (Q/U) Protocol

Figure 18 illustrates a fault-scalable approach based on the Q/U protocol. The Q/U design is best understood by considering a related protocol from which it is built, called "Read/Conditional-Write." As its name suggests, the protocol enables a client to perform a read operation on the service, or to perform a conditional write. In a conditional write, the client attempts to write a new value to the object, conditioned on the current value of the object not having changed since this client last read the object. So, for example, if both clients in this slide read the object (at a quorum) simultaneously (first frame of Figure 18) and then concurrently perform conditional writes conditioned on the orange value being current, at most one of them—in this example, the

left client's (the green value, see second frame)—succeeds. The right client's write attempt (the red value, third frame) fails, since it was conditioned on the orange value but the green value is current.

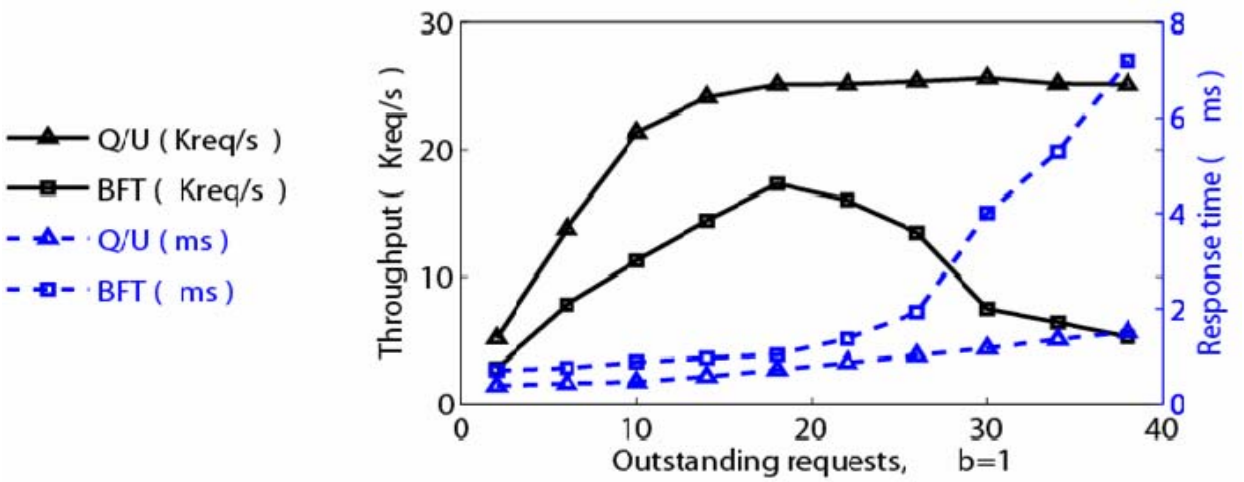


Figure 19: Performance Comparison of Q/U and BFT Under Load

Figure 19 presents a graph that gives an example of the performance of Q/U, in comparison to that of the "BFT" implementation of state machine replication by Castro and Liskov [21][22]. BFT is widely considered to be the most efficient publicly available implementation of state machine replication. This graph shows that as the load on the system grows, Q/U retains low latency and high throughput, while BFT decays in these circumstances. These tests were conducted on a cluster of 2.8 GHz Pentium 4s with 1GB RAM running over a 1 Gb switched Ethernet with no background traffic.

2.3.Focus Area 3: Scalable Enterprise Service-Oriented Architectures

In addition to evaluating the real-time QoS support of conventional SOA platforms, we also evaluated the scalability of these platforms in focus area 3. As described below, our investigation made it clear that today's SOA platforms are weak in aspects concerned with scalable management of security or appropriate-use policy. For example, consider a typical Air Force application that operates in a war zone and communicates with secured servers back in Washington, as shown in Figure 20. There may be up to 40 "hops" over a set of interconnected networks each with its own firewalls, resource allocation policies, routing policies, etc. The

regional commander might wish to make a policy statement prioritizing warfighting systems over less critical applications, but how would we implement such a policy?

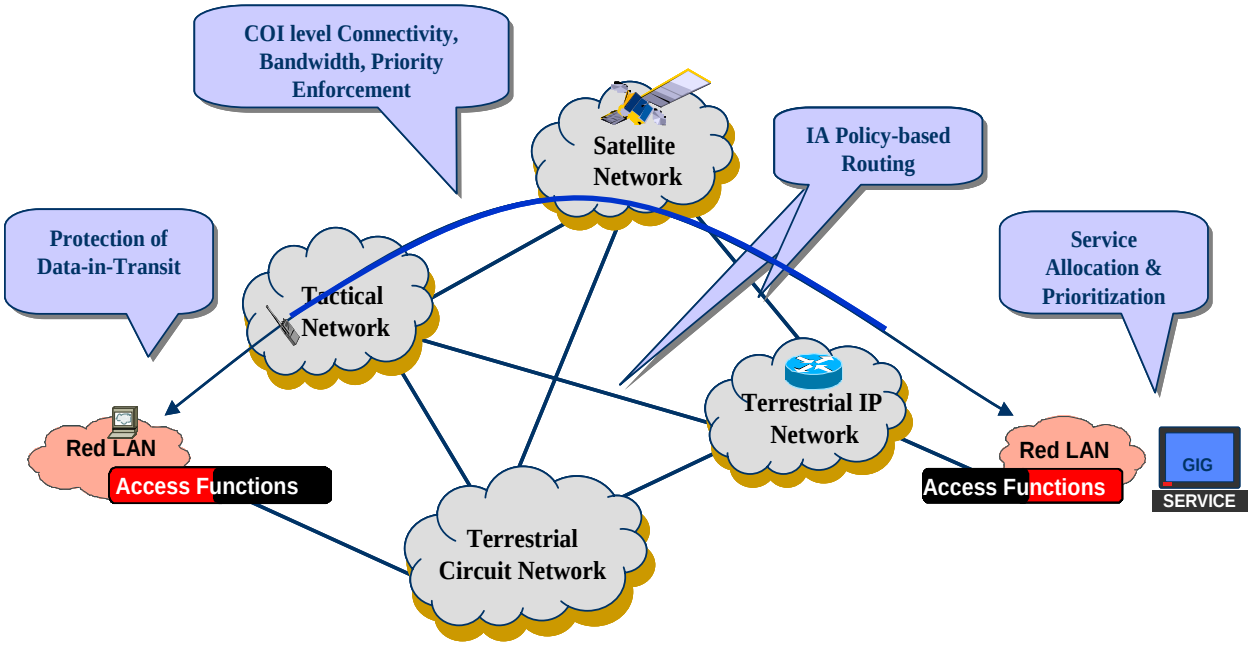
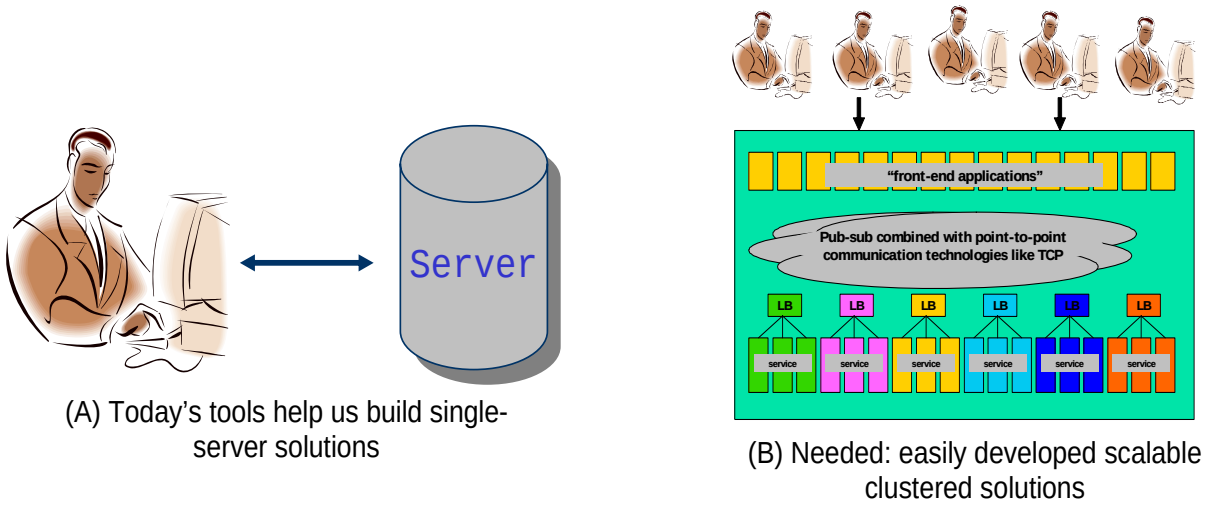


Figure 20: The Scope of a Typical SOA-based GIG Application

Even managing the policy and configuration data for a large set of firewalls spread over a world-wide deployment is a potentially hard problem. Incrementally updating a policy once a firewall has been widely deployed represents a very important, practical, and yet largely ignored problem. Relatively little is known about disseminating the associated databases of policy rules, ensuring consistency when rules are updated during times when parts of the network may be reconfiguring or disabled, detecting errors and inconsistencies, etc. Moreover, as a war is fought, needs and loads constantly change, and equipment damage will occur. Operating such a complex system robustly in the face of these kinds of threats is a tough challenge. Our Prometheus study identified a great number of such questions: questions that would arise immediately in any practical deployment and yet that are largely ignored by existing platforms. Our study did not find definitive answers to these problems, but we did identify a number of key R&D topics that have direct, measurable impact on the ability of the Air Force to operate critical networks, including scalable QoS-enabled clusters, server/Service discovery, and enterprise publish/subscribe architectures. We discuss each of these topics below.

2.3.1. Scalable QoS-enabled Clusters

As shown in Figure 21(A), COTS GIG/NCES technologies facilitate building *single server* platforms. In many development tools, suffices to click a button and designate interfaces to import/export. Unfortunately, today's solutions do not scale to support large numbers of clients, do not manage themselves, and cannot assure any form of time-critical responsiveness. Figure 21(B) shows what is needed, namely policy mechanisms, for matching information consumers to the appropriate services running on particular servers in large configurations with many data sources.



(A) Today's tools help us build single-server solutions

(B) Needed: easily developed scalable clustered solutions

Figure 21: Reality vs. Needs for Scaleable QoS-enabled Cluster

Commodity clusters are the cheapest way to put computing cycles at the disposal of popular applications [12][14][15]. For example, suppose that a map server maintains map information for a remote area of North Korea. Normally, there would be little need to handle high volumes of queries and that server would run on small numbers of nodes. But in the event of a confrontation, the load could suddenly spike and become huge. We would not want to port the application to a supercomputer "on demand" so it makes more sense to design it to scale out and ensure that if demand surges, it can simply run on more nodes while continuing to satisfy QoS properties like time-critical responsiveness.

This is perhaps an obvious observation: when load increases, we would like a way to automatically allocate resources to handle the surge. Yet we found that industry has been largely inattentive to the issue, with one notable exception: databases and transactional systems often include load-surge handling software that scales well. Thus the problem is recognized by industry, but in a narrow way, and this narrow recognition omits important classes of problems for which system-level solutions are required. By this we mean that there is an identifiable need for platform technologies directed towards typical developers with normal skill sets, designed to assist them in arriving at scalable solutions that will work well under a wide range of loads, stresses, and styles of use.

As part of the Prometheus effort, we evaluated platform technologies and systems that have these properties. For example, *Astrolabe* [59][60] is a scalable monitoring infrastructure developed by Cornell researchers several years ago using a novel fusion of peer-to-peer gossip protocols [7][24][17][32][33][39][40][48][53][54][55][59][58]0 and higher speed, more standard, multicast protocols[16][64][62]. It permits the user to diagnose complex problems, data mine or even dynamically reconfigure even a massive system. Amazon is now using this solution in their largest web centers. *Bimodal Multicast* [17] is a technology for disseminating notifications and other urgent events to very large numbers of receivers that can scale seemingly without limit and can tolerate even the most severe disruptions. *Kelips* [33] is a powerful indexing technology that can index items widely distributed in a LAN and offers client systems low-cost lookups, again in a manner that will resist even extreme disruption or reconfigurations.

Looking to the future, one project we evaluated is called Tempest, which is a time-critical cluster computing platform, still under development, that will slash the difficulty of building services that scale well on clusters. The basic vision is of a drag and drop solution: the developer builds a non-distributed web service solution using the Tempest methodology, then drags it onto the cluster, and after a quick dialog about desired properties, Tempest automatically clones the system and deploys it to the cluster, dynamically healing damage in the event of a crash or other failure [43], rescaling as the need arises for more or fewer nodes, and maintaining the good timing properties of the original application as it does these things. The key innovation is to fuse a new kind of gossip-based epidemic communication protocol [24][23][14][32][33] with a more

standard way of doing replication and distributed consistency tracking. The goal is that the developer of that time-critical map server might just run it under Tempest and in this way benefit from sophisticated scaling solutions without ever needing to implement them by hand. Underlying Tempest is Ricochet [8], a time-critical multicast update protocol that can be accessed through a DDS interface of the sort discussed earlier in this report.

In general, our findings suggest a new approach to scalable enterprise SOA platforms that focuses on the following approach:

- **New protocols:** Scalable systems demand scalable components. For example, in our research we invented Slingshot [9], a new form of forward-error correction (FEC) protocol that integrates with multicast to give high reliability, excellent scalability, and outstanding real-time properties, and Ricochet [8], an extension of Slingshot that can be used in publish-subscribe and event notification environments. Given these components, we were able to develop a scalable systems architecture [43] around them.
- **Use of standards.** We favor the development of platforms designed to fit elegantly into a Web Services setting. For example, the Tempest system will take unmodified Web Services servers, pretty much "right off the truck," and turn them into scalable load-balanced real-time clusters at the touch of a button. (In fact applications do need to be re-linked against our libraries, but this isn't a hard thing to do).
- **Rigorous techniques.** Our investigation concluded that the use of epidemic gossip often permits the developer to arrive at scalable solutions backed by rigorous mathematical formalism [7][24][32] and to reason in a rigorous way about solutions even as they are being designed and implemented. This results in a powerful mixture of real technology with "deep" understanding in how that technology will work in a target setting
- **Emphasis on real-world needs.** We could have recommended the development of more and more components along the lines of Astrolabe. But the Air Force needs to see how these can fit into realistic data center and Pub/Sub scenarios. We see Tempest as a pre-production solution. Cornell, which is developing Tempest, will make it available for the public, but the

main goal is to show potential AF vendors and platform sources precisely how it can be done! We recommend this as a general strategy for Air Force studies in this overall area.

This ongoing work at Cornell is yielding a variety of empirical results and has resulted in a series of conference submissions. For example, Ricochet, our time-critical replication protocol [8], is a tremendously powerful way of doing data transport in real-time applications. There has been tremendous interest in this work, for example from Raytheon (for use in designing hardware platforms capable of nearly instantaneous recovery from damage), Yahoo! (for use in financial trading systems), Ericcson (for use in next generation telecommunications switches), and many others. Right now, we're in the process of releasing Ricochet for general use even as we design the detailed architecture of the framework around it – Tempest – and the experiments we'll use to show that for realistic applications we can actually automate the mapping from a server that some high-school trained individual coded on Visual Studio or a similar tool into a clustered, real-time, scalable solution.

Not every server will work – Tempest will be limited to applications with weak consistency requirements right now. Our hope is to build out from that as a starting point. But many servers work with noisy data and we do offer a meaningful form of consistency – not virtual synchrony, but not best effort. So we aren't tossing out the whole question; we're just starting with low-hanging fruit.

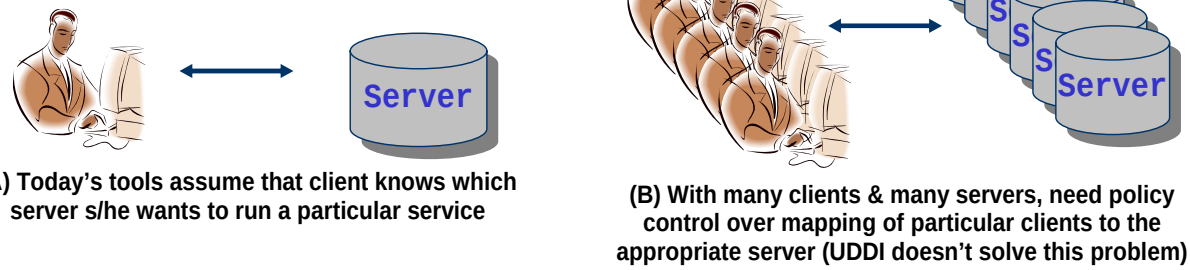
We think that being able to empower an ordinary application developer to build a scalable cluster that manages itself and has good real-time properties would be an amazing contribution even if some kinds of services need a different technology to support them, e.g., we won't tackle transactional services.

2.3.2. Scalable Server/Service Discovery

Another example of a challenging research problem on which much work is needed involves what is commonly called "service discovery," where clients seek information, e.g., maps, intelligence, locations of friendly or enemy units [10]. Services exist throughout GIG servers. Some servers have been scaled onto clusters and replicated at multiple locations, whereas some are operated by other military branches, or even coalition partners. A key challenge is to

implement *policy* to control routing of client requests to service on most appropriate server, where the policy subject to security constraints, load-balancing, locality, specific client needs, and other specific server properties.

Web services and SOA systems are designed right now under the assumption that, much like for the web, one can easily find the right server and hence the standards focus on a single pair: a single client-server "dialog." As shown in Figure 22(A). But suppose that massive numbers of clients suddenly start to search for servers in a huge pool of candidate services, as shown in Figure 22(B). These clients have individual preferences, e.g., one may want "current maps" for a region of North Korea, while another wants "annual satellite data" showing evidence of military construction or other activities over a period of time. Clients care about performance: given several sources for the identical data, they will want to communicate with a lightly loaded source. Some clients want very high quality data, or specially annotated data that has been approved for their use by intelligence professionals. We can turn these goals around and obtain a similar set of server-side, policy objectives, such as security policies, load-balancing policies, etc.



(A) Today's tools assume that client knows which server s/he wants to run a particular service

(B) With many clients & many servers, need policy control over mapping of particular clients to the appropriate server (UDDI doesn't solve this problem)

Figure 22: Scalable Service/Server Discovery

How should this problem be solved? Right now, the suggestion in the web services framework is to build a single, centralized database using UDDI to describe services and then for clients to query this database [12]. But it is easy to see that many of the questions just articulated can't be solved in that manner – and moreover, it makes little sense to take the intrinsically decentralized system we started with and to somehow impose a single centralized component. Who would run

that component, and set the standards, the Air Force, or the Army, the NSA, etc.? We therefore need a decentralized approach that can be shown to work well in the many dimensions articulated and that works with standards as much as possible but also goes beyond them as needed. Research issues here include domain-specific languages for specifying properties and policies, approaches to solving the secure query problem that scale well, too, and even "trust" as it arises in such a setting, i.e., suppose that a client is directed to such and such a service: what kinds of trust is the client implicitly accepting when using that service? Expressed differently, how can we make it hard for an adversary to confuse the client and convince it to use a low-quality data source?

Our Prometheus study found other relevant prior work and we feel that AFRL will need to engage some of the domain experts who are doing so: Balakrishnan and Shenker's team at MIT and Berkeley [10], for example, and Gun Sirer through the IAI that links AFRL and Cornell.

Subsequent to completing the Prometheus study, we helped organize a PRET proposal to AFOSR (AF-TRUST-GNC) that includes a sub-effort focused on this problem space.

2.3.3. Scalable Publish-Subscribe Architectures

Pub/Sub [45] is another area that needs to be revised in light of the GIG/NCES trends. Web services use Pub/Sub to announce events, transmit updates, etc [12]. This arises both within data centers (where the Tempest system is an example of how one might tackle the technology need) but also in scalable setting with remote clients. Unfortunately, in addition to the real-time QoS problems described in Section 2.1, existing commercial products do not scale well enough for their envisioned use in enterprise GIG and NCES systems. Air Force applications are more demanding in terms of numbers of clients, mobility, data sizes and rates, reliability and QoS needs, etc than anything one sees in the commercial sector. And while there are solution aimed at sub-problems (like DDS, a Pub/Sub solution for real-time applications but where the number of subscribers for any particular topic is typically small), we find nothing with the spectrum of needs and scalability properties required.

AFRL should invest to develop and demonstrate solutions that can later transition into major products or at least plug and play into major web services platforms in a clean and easily used

manner. At Cornell, a project called Quicksilver is tackling some of these demanding problems, again using a mixture of classical protocol mechanisms (such as virtual synchrony, the model used in our work on Isis and Horus [12][16][18][25][50][51][52][60][61][62][64] and adopted by the New York and Swiss Stock Exchanges, the French ATC, the Navy for the AEGIS warship, etc) and scalable mechanisms such as the gossip/epidemic recovery protocols demonstrated in Bimodal Multicast [17]. Quicksilver is a scalable Pub/Sub architecture with greatly improved security, reliability guarantees, stability even under distributed denial of service (DDoS) attacks and major performance overloads, designed for stable behavior in massive deployments.

The key idea of Quicksilver is to map Pub/Sub to group communication but to do so using a radically new scalable architecture that scales both in numbers of groups and in numbers of users 0. Previous approaches were limited to tiny numbers of groups. In Quicksilver each group has its own protocol stack 0. We expect to support best effort protocols similar to DDS but also much stronger reliability/security mixes, such as secured virtual synchrony. We're using the Web Services eventing model and are focused on Windows as our primary development platform. QuickSilver is just reaching an experimental evaluation stage but should become useable sometime in 2006. The eGrid community is eager to include this into their "tool set." This work is at a preliminary stage but is promising and could eventually offer solutions of the type the Air Force will need.

2.4. Focus Area 4: A Framework for Demonstrating Access Control Policy Compliance

The fourth focus area of our work during the Prometheus effort involved the development of a unified framework for demonstrating policy compliance. The particular type of policy that we studied in this effort was *access control policy*, which defines the circumstances under which requests to access a resource should be granted or denied. Clearly access control will be a requirement of a security-sensitive system like the GIG, and the complexity of the GIG demands that an access control framework be put in place that can express a wide range of policies.

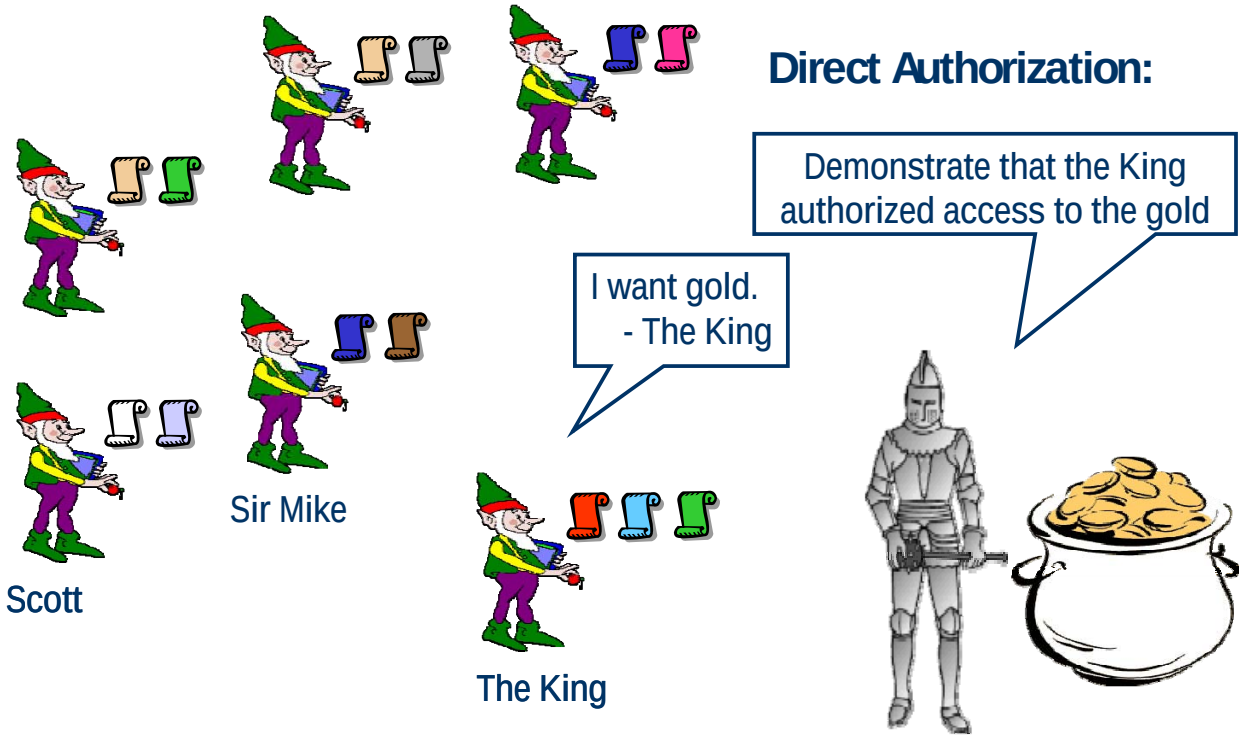


Figure 23: Direct Authorization

Access control policy is also an instructive example because policies can be quite complex, and demonstrating compliance can require collecting facts from many distributed locations. To demonstrate policy compliance with high assurance, the framework with which we experimented requires a proof in a formal logic that a policy is satisfied. Less formal approaches have also been considered in the literature. Figures 23–25 give a somewhat whimsical (but relevant) analogy of this process in which the knight represents the resource monitor, guarding a pot of gold (the resource). The knight enforces the policy that any access to the gold must be approved by the king. Authorization is conveyed from one party to another in the form of scrolls, i.e., credentials, that contain a statement and a signature. Authorization can occur in two forms: direct and indirect. An example of direct authorization is shown in Figure 23 where the King himself would like to access the gold. To do this, he signs a credential stating that he would like to access the gold.

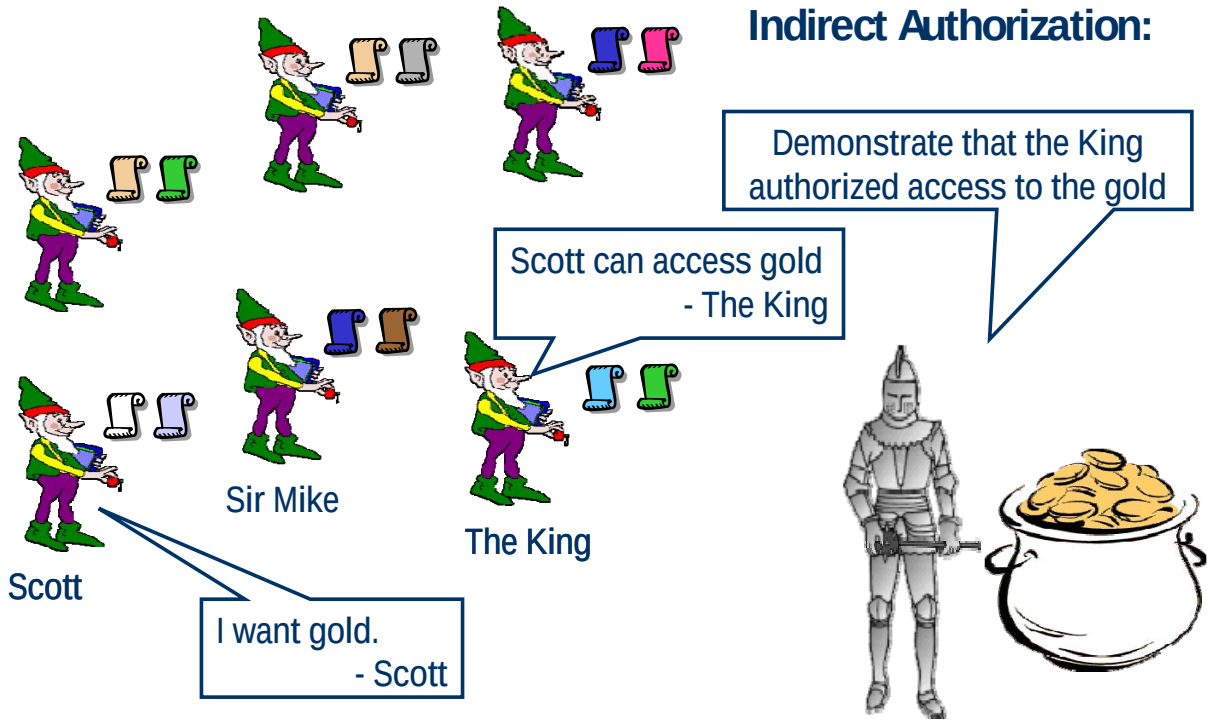


Figure 24: Indirect Authorization

However, the king may want others to conduct financial affairs on his behalf. An example of an indirect authorization is shown in Figure 24, where the king indicates that another party may access the gold. In this case, Scott requires two credentials to access the gold – one from the king stating that Scott is allowed to access the gold, and another from Scott stating that he would like to access the gold. The scenario in Figure 24 can be extended to include longer chains of authorization. For example, Figure 25 shows an example where the King has delegated control over the gold to Sir Mike, who then authorizes Scott to access the gold.

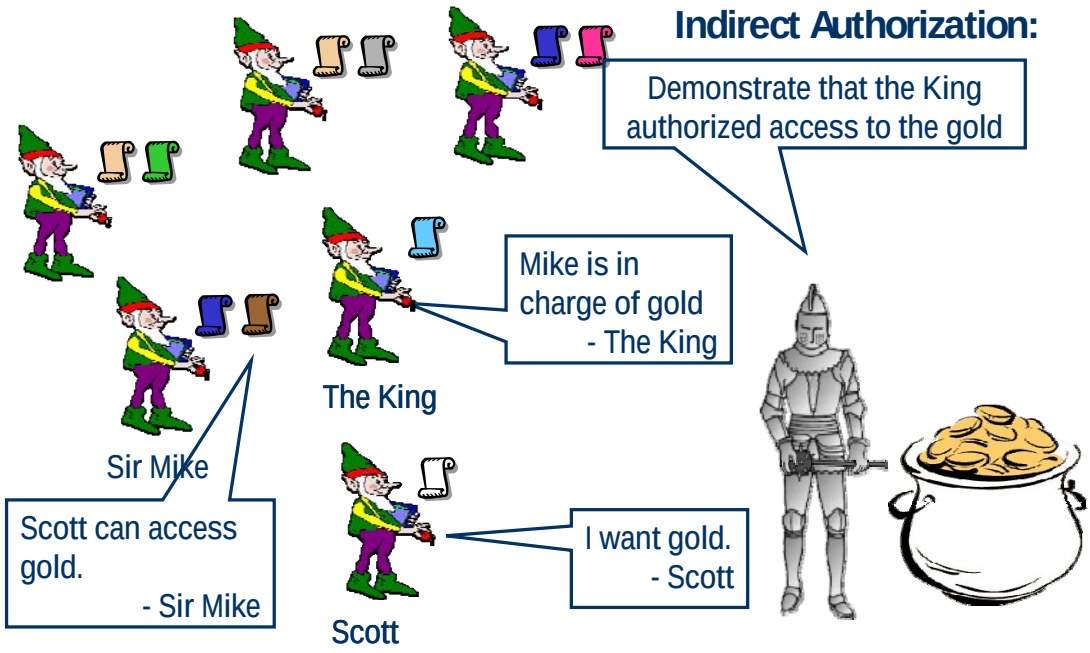


Figure 25: Delegation Chains

In all three scenarios, the responsibility of the knight (a.k.a, resource monitor) is only to verify that each credential is valid, and that the collection of credentials imply that access should be granted. This approach has several advantages:

- The demonstration of authority may take the form of a logical proof. Algorithms exist to allow the knight to check these proofs efficiently, while the use of logic gives us a greater assurance of correctness.
- Authority can potentially be demonstrated in many ways. Flexible delegation, roles, and groups allow the system to more closely resemble the relationships that arise in practice. Furthermore, the knight need not iterate all possible combinations beforehand, he must simply check that a particular demonstration is valid.
- The policy is distributed, which allows the system to better model scenarios in which there is no central authority, such as a partnership between two businesses.

However, the approach described above has several drawbacks as well. Since there are many ways to derive authority, anyone constructing a proof must potentially consider each of these

ways. Moreover, the credentials are distributed among the signers, and may be created dynamically in response to a request. For privacy reasons, you may not wish to reveal all of your credentials to other parties, which makes it difficult to identify and retrieve the set of credentials that will demonstrate access. During the Prometheus project we therefore undertook an initial study of this problem, i.e., to develop a more efficient algorithm for identifying and retrieving this set of credentials.

Figure 26 shows some preliminary results for a new credential-collection and policy-demonstration algorithm that we developed. The approach we developed yielded a significant (i.e., four-fold) improvement in the number of requests needed to locate the proper credentials and create proofs over prior approaches in the literature. One of our optimizations is called "automatic tactic generation" (ATG), which "generalizes" approaches to collecting the appropriate credentials from a single proof for one access. Since resources in an environment typically are governed by common policies, how one resource was accessed can give insight into how another resource might be accessed. This insight is exploited in ATG to more efficiently construct proofs of policy compliance once a single proof has been constructed.

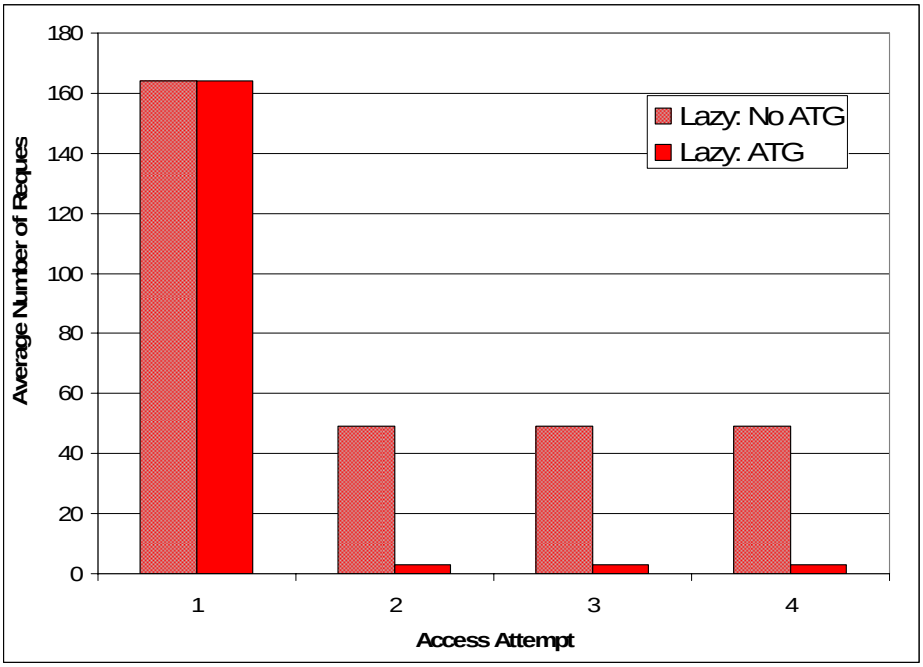


Figure 26: Performance of New Credential-Collection and Policy-Demonstration Algorithm

During the Prometheus project we deployed our access control policy framework in a testbed at CMU for further experimentation. Our results showed performance gains via novel proof construction optimizations, but there is much left to do. We plan to conduct future research on this topic in the context of a unified framework for QoS policy management in the Pollux effort, as described in Section B.2.

2.5. Summary of Results

Section A has described our activities and results in the Prometheus project that focused on (1) assessing the challenges of applying existing SOA methods, platforms, and tools to develop SoS architectures for tactical and operational information management in the GIG settings and (2) developing initial requirements and examples for the key technical innovations that will constitute the QoS-enabled GIG. As part of these activities, we designed of prototypical proofs-of-concepts and conducted several feasibility studies aimed at establishing a baseline for technology in the area of QoS-enabled GIG, developed recommendations concerning the architecture of a QoS-enabled GIG platform, and identified metrics where progress is needed.

The activities in Section B below involved the evaluation and refinement of prototypes and experiments described above. The goal was to produce a technology roadmap for future GIG R&D activities that is based on solid experimental and analytical results.

Study Recommendations for Future R&D Efforts

3. Overview

Our 9-month Prometheus study revealed that some of the most promising technologies for meeting the challenges in the GIG described in Section A involved (1) enabling users (e.g., commanders and administrators) and developers (e.g., software and systems engineers) to specify various real-time and security QoS policies that convey their intent at a domain-centric level and (2) flowing these domain-centric policies down to configure and deploy software and hardware components in networks, operating systems, middleware, and applications that enforce these policies layer-to-layer and end-to-end across potentially autonomous domains.

To explore these technologies further, the Prometheus study produced recommendations related to developing, analyzing, empirically evaluating, and optimizing QoS-enabled SOA and Pub/Sub middleware platforms and tools that will provide the following capabilities:

- A Unified Framework for QoS Policy Management
- QoS-enabled Pub/Sub for Tactical Information Management
- Scalable Fault- and Intrusion-Tolerant Services
- Technology for Scalable Clustered Service Architectures
- Scalable Service/Server Discovery Architectures

The remainder of this section describes future R&D efforts that can provide these capabilities. We plan to explore many of these topics in follow-on efforts to Prometheus, including the Pollux and Castor projects.

4. A Unified Framework for QoS Policy Management

The goal of this set of R&D activities is to build upon the Prometheus activities described in Sections A.2.1 and A.2.4 to define, validate, and optimize a unified framework for QoS policy management that enables the predictability and resource control required by tactical information management systems, while preserving the modularity, scalability, and robustness that’s the hallmark of the Pub/Sub model. In platforms that support tactical information management for

the GIG, QoS policies will enable users and developers to control many aspects of middleware, operating systems, and network mechanisms. QoS policies will often be implemented as contracts between publishers and subscribers, where publishers offer and subscribers request various levels of service. Examples of QoS policies available to users and developers of tactical information management applications in the GIG include:

- The lifetime of each data instance, i.e., whether it will be (1) destroyed after it is sent, (2) kept available for the sender's lifetime, or (3) kept available for a specified duration.
- The degree and scope of coherency for information updates, i.e., whether a group of updates can be received (1) as a unit and (2) in the order in which they were sent, and to how large a subset of all updates (1) and (2) may be applied.
- The frequency of information updates, i.e., the rate at which updated values are sent, or the maximum rate at which they may be received.
- The maximum latency of data delivery, i.e., the maximum acceptable interval between the time data is sent and the time it is delivered to the receiver.
- The priority of data delivery, i.e., the priority used by the underlying transport when delivering the data.
- The reliability of data delivery, i.e., whether missed deliveries will be retried, and for how long the sender may block while waiting for a successful retry.
- How simultaneous modifications to shared data can be arbitrated, i.e., how a selection can be made among the owners of the shared data to determine which owner's modification will be the one that is actually applied.
- Mechanisms and parameters to determine liveliness, i.e., the frequency with which liveliness is asserted, whether or not assertion is automatic (handled by the infrastructure) or manual, and, if manual, who is responsible for making the assertion.
- Parameters for filtering by data receivers, i.e., predicates which, when bound to the values of individual data instances, determine whether instances are accepted or rejected.

- The duration of a data instance's validity, i.e., the specification of an expiration time for a data instance, so that the delivery of stale data can be avoided.
- The depth of the 'history' included in updates, i.e., how many prior updates will be available at any time, for example 'only the most recent update' or 'the last *n* updates' or 'all prior updates'.
- The ability of publishers and subscribers to access data and resources, i.e., particularly so for access-control policies that adversaries will attempt to circumvent through clever means.

Specifying and enforcing all of these QoS policies in tactical information management systems today is an enormously tedious and error-prone task. In particular, users and developers need to configure and tune all these policies at multiple levels, including lower-level networking mechanisms (such as policies for classifying and prioritizing packets and flows), operating system mechanisms (such as policies for prioritizing access to CPUs, memory, and I/O devices), middleware services (such as event notification, scheduling, fault tolerance, security, and load balancing), and services in the applications themselves (i.e., the actual "business logic" of the applications). This problem is exacerbated by the fact that not all combinations of options are semantically compatible.

During the past decade, researchers have increasingly moved toward formal approaches to specifying, enforcing, and demonstrating conformance to QoS policies. Early work focused on modeling QoS mechanisms using formal logic (e.g., [42]) and constraint checking systems [37][63] to justify their correctness before system deployment. More recently, there has been progress in using formal logic within the system at run-time to construct proofs that policies can be satisfied before resources are allocated and/or access is granted [19]0. In this approach, a proof is constructed to demonstrate that the policies can be satisfied. To ensure security, this proof is built using a formal logic from digitally-signed credentials issued by various parties in the system. Using formal logic in the QoS enforcement implementation limits the opportunities for implementation errors at run-time, in contrast to merely modeling an abstraction of the system in formal logic at design-time.

In tactical information management systems of the scope, complexity, and flexibility of the GIG, we believe that such formal approaches offer the best chance for minimizing costly (indeed, deadly) errors in policy enforcement. These formal logic approaches have not, however, been demonstrated in large-scale systems, much less systems-of-systems, so the current state-of-the-art leaves open many questions regarding their use in real systems. Within the context of future effort, therefore, research is needed to understand the degree to which these approaches are suitable for use in real systems, and also develop new distributed algorithms, tools, and middleware platforms to make them more suitable. Specific challenges and approach that future research should address involve developing a unified framework for QoS policy management that can:

- Significantly ease the task of creating new tactical information management applications for the GIG and integrating them with existing artifacts in new/larger contexts/constraints. Key R&D challenges involve converting commander's intent, along with static/dynamic environment, into QoS policies. To address these challenges, we recommend the creation of domain-specific languages and a suite of tools that can be used to precisely specify a wide variety of QoS policies. These tools should present choices to users in an intuitive way that guides them to meet warfighter objectives, not confuse them with barrage of obscure options. These tools should also be make it possible to analyze, validate, and certify QoS policies for various static and dynamic properties, such as semantic conflicts, time/space overhead/performance, pinpointing workflow bottlenecks.
- Build technologies that (1) enable the decentralized creation of access control policies for distributed resources and (2) exercise that authority efficiently when resources need to be accessed. The goal is to construct a logical framework that can be used in tactical information management systems to formally validate access-control decisions as compliant with policies. In addition, this framework should be extensible to adapt to changing needs, so that new policies can be created on-the-fly. A rigorous and flexible access-control infrastructure is a step toward mechanisms to enforce other types of policies as well, such as the QoS policies that will also be explored in this proposed effort. Reconfiguring a system to ensure QoS properties is itself an activity that should be subject to access control, since doing so

might be at the expense of some other property or party in the system. To prevent the abuse of this flexibility, it is appropriate to limit the ability to reconfigure the system in such a way that would impose on others' activities. For example, a Colonel's attempts to reconfigure the system to support her QoS needs should perhaps be permitted only if this does not interfere with her General's QoS needs. The unified framework for QoS policy management should support the specification of rich access control policies such as these.

Figure 27 illustrates the general approach for a unified framework for QoS policy management. To achieve such a framework will require the development of tools to capture commander's intent and/or application-level resource management priorities and map them to the underlying platform QoS policies when (1) entities contending for QoS resources are operating in disjoint contexts, i.e., no higher level arbitration may be feasible in time-frames of interest, (2) QoS requirements change dynamically as mission needs evolve, (3) critical information needed to establish urgency is transient and only available locally, and (4) local entities are motivated to inflate their urgency. In conjunction with the QoS-enabled platform technologies described in Section B.3 below, this unified framework for QoS policy management should support dynamic/consistent end-to-end system monitoring, adaptation, and optimization at multiple layers in tactical information management systems.

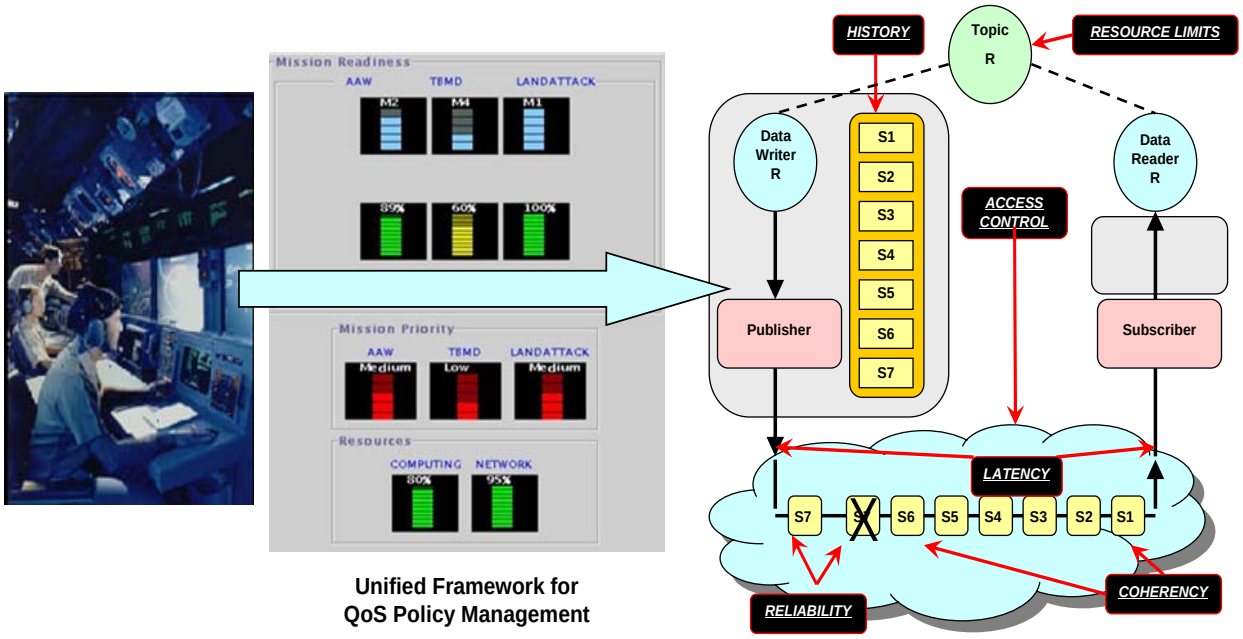


Figure 27: Configuring QoS Policies via Unified Framework

5. QoS-enabled Pub/Sub for Tactical Information Management

Some of the most promising technologies for meeting the challenges in the GIG described in Section A involve (1) enabling users (e.g., commanders and administrators) and developers (e.g., software and systems engineers) to specify various real-time and security QoS policies that convey their intent at a domain-centric level and (2) flowing these domain-centric policies down to configure and deploy software and hardware components in networks, operating systems, middleware, and applications that enforce these policies layer-to-layer and end-to-end across potentially autonomous domains. To explore these technologies further we recommend developing, analyzing, empirically evaluating, and optimizing QoS-enabled Pub/Sub middleware platforms and tools that will provide:

1. Universal – yet secure – access to information from a wide variety of sources running over a wide variety of hardware/software platforms and networks.
2. An orchestrated information environment that aggregates, filters, and prioritizes the delivery of this information to work effectively in the face of transient and enduring resource constraints.

3. Continuous adaptation to changes in the operating environment, such as dynamic network topologies, publisher/subscriber membership changes, and intermittent connectivity.
4. Tailorable, actionable information that can be distributed in a timely manner in the appropriate form and level of detail to users at all echelons.

The goal of this set of recommended R&D activities is to integrate the unified framework for QoS policy management described in Section B.2 with QoS-enabled Pub/Sub technologies so that tactical information management applications will function properly in environments where communication bandwidth is limited/variable, connectivity is intermittent, connections are noisy, processing and storage capacity are limited, power and weight limits affect usage patterns, unanticipated workflows are common, and dynamic network topology and membership changes are frequent. Although the results in Section A.2.1 show that standards-based COTS middleware like OMG DDS provides a promising foundation for tactical information management in the GIG, existing specifications and implementations are not yet sufficient to support the stringent real-time and security QoS requirements. In particular, key R&D challenges associated in the QoS-enabled platforms portion of this effort involve: (1) specifying and enforcing integrated QoS policies at multiple layers (e.g., application, middleware, OS, transport, and network) to support communities of interest within multiple domains, (2) managing resources in the face of intermittent communication connectivity, e.g., power, mission, environments, silence/chatter, and (3) compensating for limited resources in tactical environments, e.g., bandwidth, compute cycles, primary/secondary storage.

To address these challenges, we recommend that future R&D activities focus on the following topics pertaining to platforms for tactical information management in the GIG:

- Develop centralized and/or decentralized services for automatically discovering publishers and subscribers for topics and establish data flows between them as permitted by the settings of the contracts defined using the unified framework for QoS policy management described in Section B.2. These services will leverage and integrate mechanisms in the GIG infrastructure that enable control over QoS on per-data-flow basis, where each pub/sub pair can establish independent QoS contracts.

Figure 28 illustrates the general approach, where different flows can specify their relative importance and resource needs, which the GIG infrastructure is responsible for enforcing given the operating conditions and competing requirements from other clients in the system. To achieve these capabilities will require developing and evaluating mechanisms for enabling QoS policies to be set (or overridden) at the service, topic, or publish/subscribe pair level and enforced at multiple layers of the GIG infrastructure, e.g., the middleware, OS, and network layers. It will also require developing and evaluating online and offline techniques for determining whether the QoS offered by publishers/subscribers – and the resources available from the infrastructure – can satisfy client requests, establishing the communication paths on success or indicating an error on failure. Careful attention should be paid to QoS techniques that run over IPv6 and use its DiffServ packet classification and IntServ flow reservation mechanisms since IPv6 is slated to become the standard networking infrastructure for the DoD.

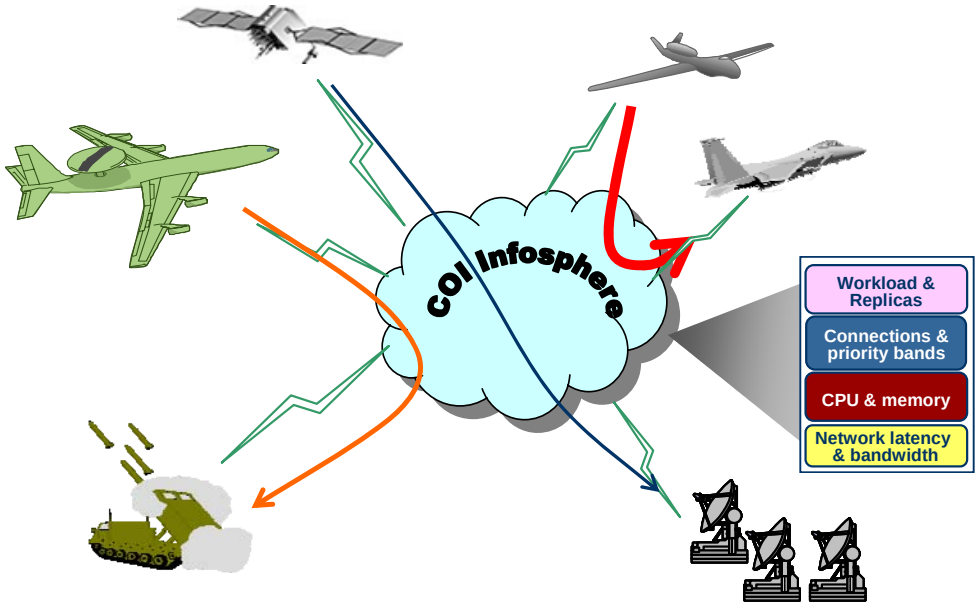


Figure 28: Establishing QoS Data Flows Through the GIG

- Our experience benchmarking various Pub/Sub mechanisms in the Prometheus study to produce the results shown in Figures 12, 13, and 14 underscored the fact that developing, optimizing, and evolving Pub/Sub applications *manually* is tedious and error-prone. Key

R&D challenges involve (1) quantitatively evaluating and exploring complex and dynamic QoS problem/solution spaces to evolve effective solutions and (2) assuring QoS in face of interactive and/or autonomous adaptation to fluid environments. We therefore recommend that future R&D activities focus on creating and evaluating Model-Driven Development (MDD) tools [38] that can significantly ease the task of creating new QoS-enabled information management applications integrating them with existing artifacts in new and larger contexts.

(A) System Design Tools | (B) QA & Optimization Tools | (C) Resource Management Tools

Figure 29: Design and Quality Assurance Tools

Figure 29 illustrates examples of MDD tools developed in prior efforts in the context of the Real-time CORBA and Real-time CORBA Event Services shown in Figure 4. Figure 29(A) illustrates design tools that specify QoS policies for a particular domain, system representation, and/or software architecture [11]. Figure 29(B) illustrates tools for quality assurance (QA), such as distributed continuous quality assurance [44]. Figure 29(C) illustrates tools for resource management and visualization of resource usage, such as system execution modeling [47]. We recommend that future R&D activities enhance these tools so that the work with the new generation of QoS-enabled Pub/Sub platforms, such as OMG DDS.

We also recommend that future R&D efforts benchmark standards- and COTS-based technologies to compare different Pub/Sub platforms, as well as to pinpoint use cases and environments in which one technology is better suited than others for particular missions and tasks.

6. Scalable Fault- and Intrusion-Tolerant Services

There is much research left to be done on the topic of scalable fault- and intrusion-tolerant services. One example is their use in "multi-tier" applications in which one intrusion-tolerant service becomes a client of others. Due to the distribution inherent in intrusion-tolerant services, when such a service is acting as a client to another, access requests from individual client replicas must be managed collectively, lest a request be performed multiple times or without sufficient corroboration from correct client replicas. A second example is scalable approaches to dynamically modify the server membership underlying an intrusion-tolerant service, e.g., to seamlessly grow its capacity after it has been deployed. Finally, little is known about how to position servers in a large-scale network to provide low-latency and low-congestion access for clients. In fact, for quorum-based services, many such questions are NP-hard, and so heuristic or approximations are needed.

We have identified several promising approaches to address challenges of scalable fault- and intrusion-tolerant services.

6.1. Multi-tier Fault- and Intrusion-Tolerant Services

Most fault-/intrusion-tolerant services have investigated only single-tier services. Multi-tier services, in which one service acts as a client to another, introduces additional complexities, such as promoting a corrupted component of one service into a client of the other. Such corrupted clients could potentially corrupt the service irreparably. We therefore need to investigate approaches to implement multi-tier fault-/intrusion-tolerant services scalably.

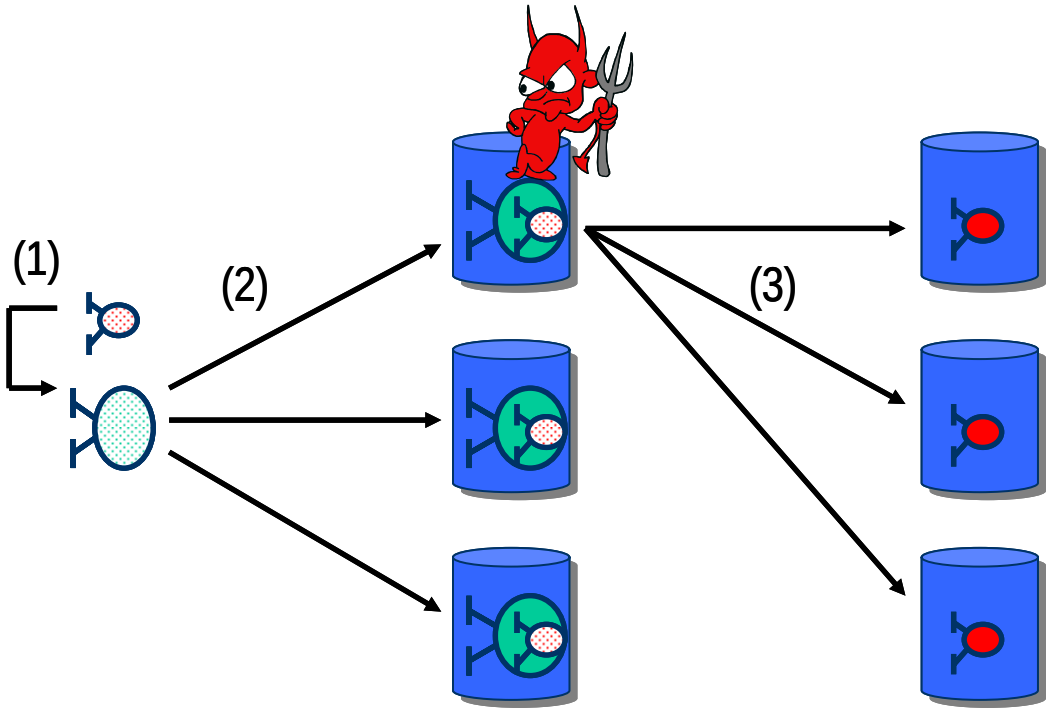


Figure 30: Multi-tier Support for Fault- and Intrusion-Tolerant Services

Figure 30 illustrates a key challenge of multi-tier intrusion-tolerant services. In this particular example, there are three replicas of two services (a red one and a green one). The client at the left holds a handle (or stub) for accessing each service, but in this particular example, the client passes the handle of the red service into the handle of the green service (step 1), thereby propagating that handle down to the replicas of the green service (step 2). Now, an adversary who corrupts a green replica obtains a handle for invoking the red service, essentially promoting the adversary to be an authorized client of the red service, as well (step 3). Further research is needed to understand how to most efficiently address this problem.

6.2. Dynamically Adjusting Membership in Fault- and Intrusion-Tolerant Services

Dynamic service membership has not been adequately examined in intrusion-tolerant systems, in that services are typically defined over a static collection of servers. Dynamic membership might enable the use of intrusion-tolerant access protocols in peer-to-peer environments, where services are implemented over transient peers. To date there has been little investigation of this approach, especially for quorum-based implementations. Yet the problem is fundamental: when we talk

about discovery of a service the steps include first matching the client to some sort of service description (for example, matching a client seeking “maps of Faluja” with such-and-such a database), but then resolving the service identifier or name into a set of representatives that might vary over time as nodes crash or recover and as loads vary, and then vectoring the request to a specific service representative that will handle it, perhaps with further constraints, such as “this request should go to the same representative that handled my last inquiry, if possible, to let it exploit cached authentication certificates and speed up the access.” Thus we face a multi-layer, secure, dynamic membership tracking and lookup requirement.

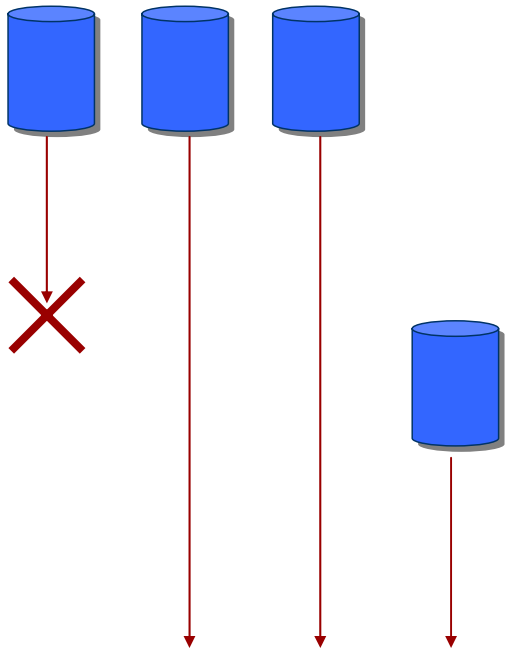


Figure 31: Dynamically Adjusting Membership in Fault- and Intrusion-Tolerant Services

Figure 31 describes the need to adjust the membership of a fault- and intrusion-tolerant service dynamically. In this figure, time is running downward. The “X” denotes a crash or departure, and the new (rightmost) server comes in to pick up the slack. The question is, what is the right algorithm to seamlessly integrate the new server into the access protocol? This specific problem has been studied in the context of gossip-based services such as the ones discussed earlier in this report, and in the context of replication with strong properties such as virtual synchrony or one-copy-serializability (the database replication model). Yet there has been surprisingly little work on this question in the context of quorum-based services, particularly intrusion-tolerant ones.

6.3. Deploying Quorum-based Services in Large-scale Systems

Deployment of scalable fault- and intrusion-tolerant services for GIG "systems-of-systems" environments requires careful placement of servers. Placement should seek to minimize network-centric costs (e.g., delay and congestion) for accessing the service from various parts of the network. Applicability to GIG "systems-of-systems" demands attention to this problem, which is currently very poorly understood.

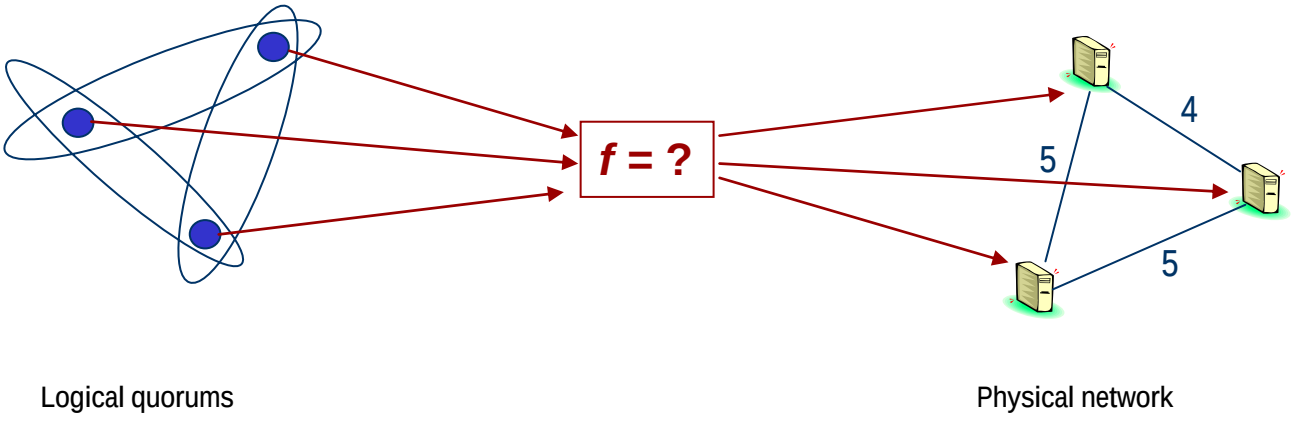


Figure 32: Deploying Quorum-based Services in Large-scale Systems

Figure 32 shows the problem face when attempting to deploy a quorum-based service in a large-scale system. A quorum-based protocol is expressed with accesses to logical quorums, i.e., accesses to logical subsets of servers. To deploy such a protocol in a real system, however, the quorum elements must be mapped to physical machines. But what should this mapping be? Ideally each quorum element would be placed to minimize some network-centric measures, e.g., the expected latency for, or congestion due to, client accesses. Additional research is needed to address these questions.

7. Technology for Scalable Clustered Service Architectures

To address the challenges of developing scalable clustered services described in Section A.2.3.1, we believe that the standard application-development tools and technologies will need to be extended with add-ons targeting military applications and scenarios. For example, we are urging that tools be created that would start with a standard service implemented in Web Services built

using traditional tools with a WSDL file. These tools would analyze the WSDL file, solicit additional input, & build a "cluster-enabled" version of the application. Data would then be replicated within clusters of server nodes for scalability and load-balancing using component composition tools and protocols with QoS assurances to automate the creation of a scalable, self-managed solution. This is just one concrete example of the broader question and opportunity. Tools of the nature we envision would lower the entry barrier for technology development in the GIG-SoS arena and hence open the door to a wave of powerful solutions for major Air Force and military needs.

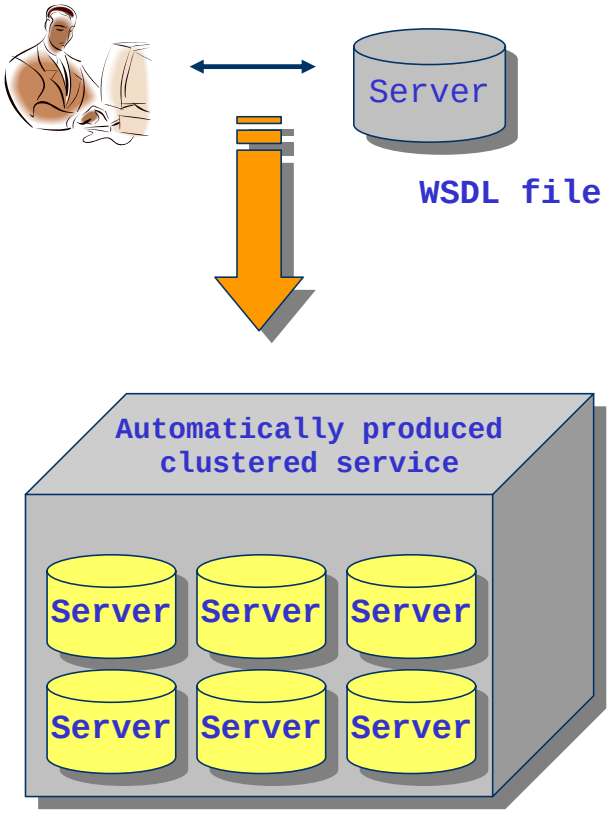


Figure 33: Automating Service Clustering

Cornell's work on the Tempest system can be understood as an initial foray into this important and substantial part of the GIG/SoS space. Figure 33 revisits the Tempest scenario, which was discussed in some detail earlier, but now with the aim of identifying opportunities beyond the work already underway. On the top we see a picture of Tempest and how it might look to the programmer: he or she develops the application in a standard way on a non-distributed platform.

The role of Tempest is to take that initial solution and automatically transform it into a clustered solution.

Earlier, we explained how Tempest actually works: it clones the user's service and uses Ricochet [8] as the core of a replication and data consistency protocol that ensures extremely rapid response to incoming updates. With Ricochet, one arrives at a cluster with nice real-time properties. But in fact one could also explore many other options, for example ones focused mostly on consistency and with lower overheads but less good real-time properties and we may go in the extreme reliability direction too using virtual synchrony as our "model". Cornell's Tempest system will ultimately host a family of solutions and the first one we're implementing (with Ricochet as the protocol) emphasizes real-time; later we may explore other QoS requirements within the same framework. We recommend that the Air Force consider launching additional research efforts in this part of the design space, because we believe the payoffs to the military could be enormous, and because we see these problems as solidly matched to near term and rather specific needs arising in the Air Force and not seen in many other settings. For example, Air Force's requirements for time-critical response go well beyond those seen in most military environments and arise on a much broader range of applications and services.

Development of powerful end-user oriented tools (GUIs) is an important research area in its own right. An important insight one can gain from the SOA world is that developers have come to expect and demand these kinds of tools. We can invent better technology and yet fail by offering it in unpalatable forms. We need to learn to view the presentation of the solution as part of the research!

8. Scalable Service/Server Discovery Architectures

To address the challenges of developing scalable service/server discovery architectures described in Section A.2.3.2, we recommend the creation of an architecture and a supporting platform for finding services in a massive system with huge numbers of clients, services, & policy governing discovery. The recommended approach would be to describe services with UDDI, extended to "service description ontology" and build a global-scale repository for maintaining these for later search. Likewise, we recommend the development of a client query framework and also a

managerial policy language for finding services, along with a scalable repository to implement architecture and search techniques.

After this infrastructure is created, we recommend next considering the routing issues that arise if a service exists in many places and the application has policy requirements governing the binding of clients to service instances. In addition to routing, we need to explore the co-existence of this new service/server discovery architecture in the context of Internet routing policies and Internet DNS name resolution options.

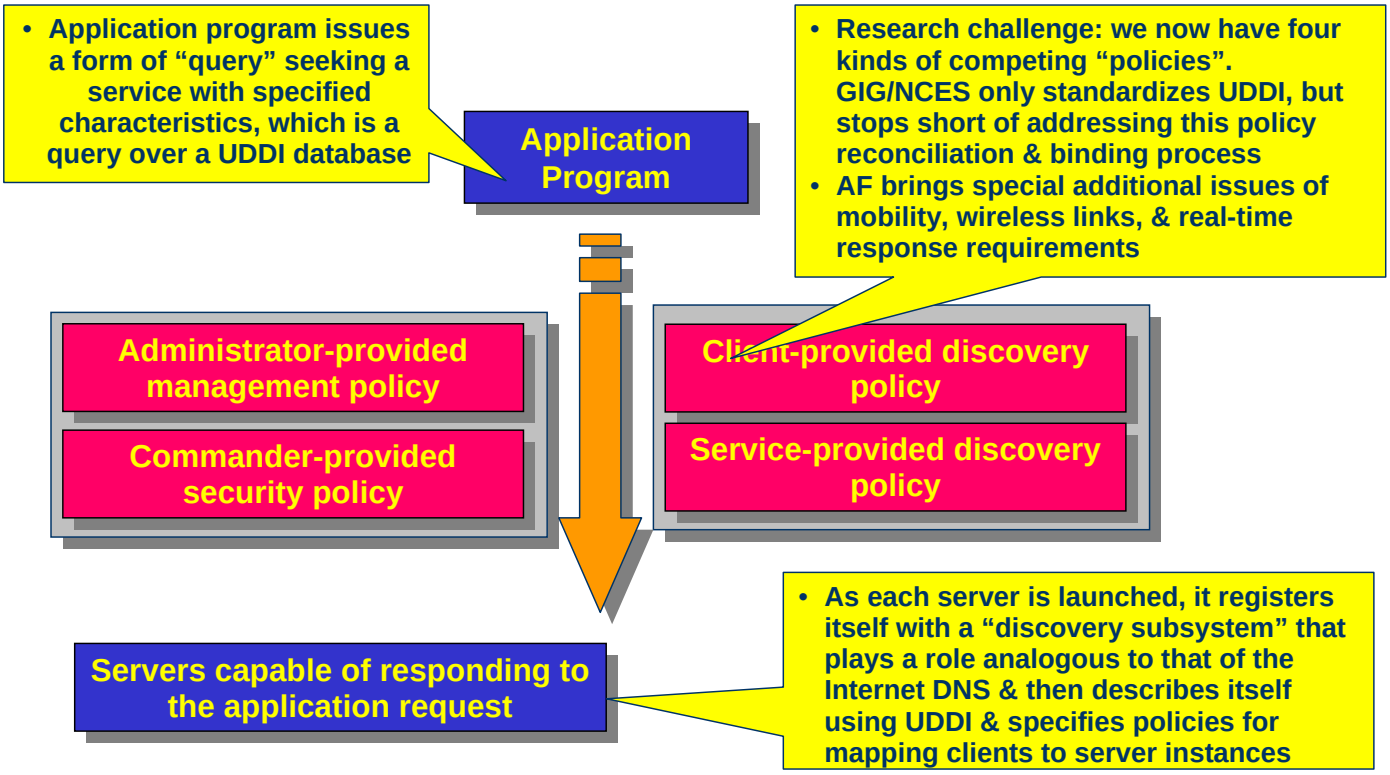


Figure 34: Challenges of Reconciling Discovery Policy Rules

Figure 34 illustrates the research challenges associated with reconciling "discovery policy" rules that originate from many competing sources, some of which do not trust one-another! Researchers have explored similar questions in other settings, for example in Andrew Myers' work on JIF, a framework for writing policy and trust rules down and for compiling them into distributed solutions. JIF could be a starting point for solutions in this critical application space.

9. Summary of Recommendations

To summarize, our Prometheus study has explored a very substantial technology space. The space is so broad, indeed, that it can be hard to discuss in any constrained way, since it spans (in effect) all of modern distributed computing, quality of service, autonomic response and security. We're recommended what can be appreciated as a prioritized attack on problems that arise in this environment:

1. The first priority should be to understand the manner in which GIG/NCES and SoS technologies will be employed in future military systems such as the Air Force.
2. From this understanding, one can identify short-term practical questions that need to be explored with demonstration and early prototyping efforts that seek to improve understanding within the service research arms while also clarifying technology gaps.
3. We are already involved in many such efforts and reviewed representative examples here, including work on scalability, real-time systems and clustered time-critical services, security in depth, and other related topics. These are broad and significant topic areas but represent just a narrow subset of the overall "space".
4. From our own experiences and an analysis of the GIG/NCES technology trends we are able to identify significant puzzles: areas in which relatively little is known and yet where Air Force systems will need to offer superior solutions in the near term. We pointed to a number of these and gave examples of why they are important.
5. Air Force work on the Joint Battlespace Infosphere (JBI) turns out to inform our inquiry in important ways. This prior JBI research has elucidated such key questions as how one might architect an information structure to ensure maximum compatibility between applications developed independently but sharing a communications environment. Other JBI insights include perspective into the central role of publish-subscribe in systems of the near future, and an appreciation of how hard it will be scale solutions of this nature up to the necessary size and complexity while also assuring that security will be maintained through pro-active policy specification and enforcement, in-depth defense against attack or disruption, and so forth.

6. Against this set of steps, we illustrated the manner whereby a near-term and mid-term research agenda can be extracted that is likely to have maximum short-range impact on problems of the greatest possible importance to the Air Force. While we do not view the research topics articulated here as exhaustive, we do think they are illustrative and that they indeed capture a wide range of vital questions that must be answered if the Air Force is to be assured of success in its own mission.

10. Concluding Remarks

The challenges addressed by the Prometheus study revealed a very substantial research area. Our Prometheus Team conducted a study, produced reports and annotated briefings, and laid out a roadmap aimed at helping AFRL take the GIG down the right path. The material presented in this final report was presented to the AFRL Science Advisory Board in November, 2005.

A key to the success of this and future R&D efforts on the GIG is a recognition of the vital importance of picking paths that can be pursued in the right time frame and can be transitioned to DoD systems integrators and COTS/GOTS suppliers in the relatively near term. Prometheus identified a wide range of problems and challenges, but also performed triage on various promising solution technologies (such as QoS-enabled QOS and Pub/Sub). Our efforts focused on understanding what could already be done today using state-of-the-art technologies in best-of-breed ways. This final report laid out a set of concrete objectives based on experimental proof-of-concepts and feasibility studies that (1) represent moderate risk, high payoff steps for the GIG and (2) benefit the Air Force GIG strategy in measurable and tangible ways.

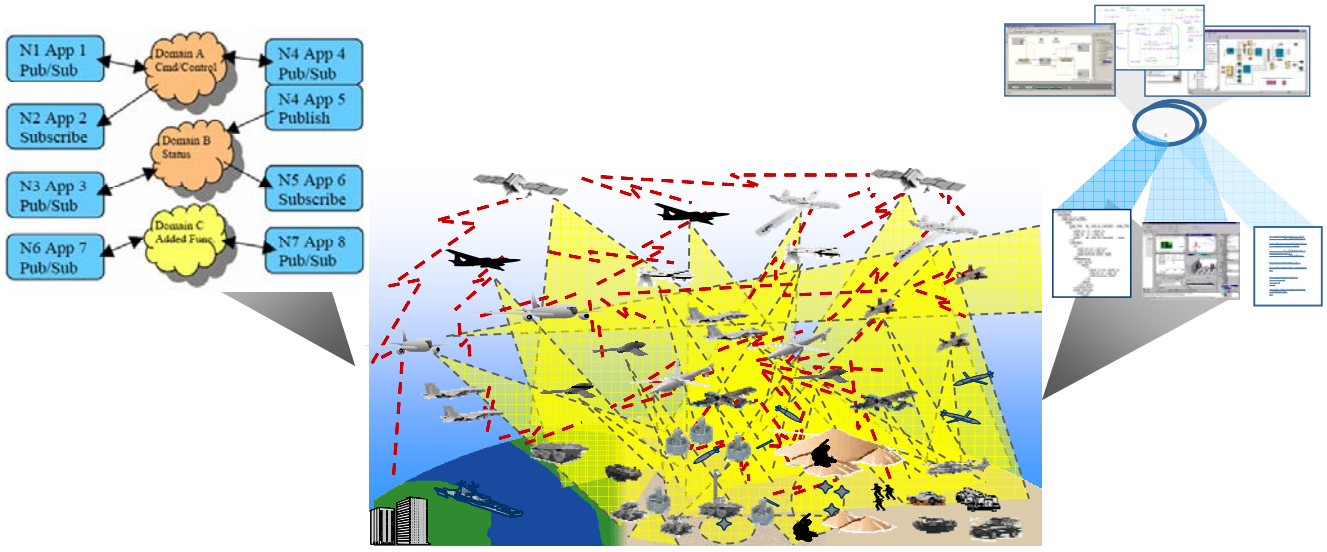


Figure 35: Future GIG QoS Needs Require Innovations in Both Platforms and Tools

Technology transitioning may be an issue that needs attention. Groups like ours at Cornell, Vanderbilt and CMU typically limit themselves to technology prototypes and even when they are useful and we make them available, they aren't products. People who download them for free

will need to expect to get what they pay for (namely limited documentation, limited support…). Thus academic research is needed, but is unlikely to press beyond the stage of showing how to break through technology barriers and how the solutions can still look natural. Even when such work is successful, unless the development team takes the step of starting a company to commercialize something (and we do, but not always) the work most often becomes commercial through partnerships with major vendors like Microsoft or with integrators like Telcordia, Raytheon, Boeing, and so forth. We feel that the Prometheus study needs to emphasize that sometimes, incredibly desirable technologies just need a sort of maturation process and pipeline and that University research is only one stage in that pipeline. AFRL will need to be attentive to this issue and explore options for creating such pipelines, with the University efforts partnering with vendors.

References

[1] Abraham, I., D. Malkhi. Probabilistic Quorums for Dynamic Systems. In the *17th International Symposium on Distributed Computing (DISC 2003)*, Sorento, Italy, October 2003.

[2] Aguilera, M.K., Strom, R.E., Sturman, D.C., Astley, M., Chandra, T.D.. "Matching Events in a Content-based Subscription System". *18th ACM Symposium on Principles of Distributed Computing (PODC)*, Atlanta, GA. 1999.

[3] Alvisi, L., D. Malkhi, E. Pierce, M. Reiter. Fault Detection for Byzantine Quorum Systems. *IEEE Transactions on Parallel and Distributed Systems* 12(9):996—1007, 2001.

[4] Amir, Y., C. Nita-Rotaru, J. Stanton, G. Tsudik. Scaling Secure Group Communication: Beyond Peer-to-Peer *Proceedings of DISCEX3 Washington DC*, April 22-24, 2003.

[5] Anceaume, E., B. Charron-Bost, P. Minet, and S. Toueg. "On the Formal Specification of Group Membership Services." Technical Report 95-1534. Department of Computer Science, Cornell University, August 1995.

[6] Andersen, D., H. Balakrishnan, M. F. Kaashoek, R. Morris. Resilient Overlay Networks. *Proceedings of the Symposium on Operating Systems Principles* 17, 131-145, Vancouver, CA, Oct. 2001.

[7] Bailey, N. *The Mathematical Theory of Epidemic Diseases*, 2d ed. London: Charles Griffen and Company, 1975.

[8] Balakrishnan, Birman, Phanishayee, and Pleisch. Ricochet: Low-Latency Multicast for Scalable Time-Critical Services. In Submission. 2005

[9] Balakrishnan, Pleisch, Birman. "Slingshot: Time-Critical Multicast for Clustered Applications," IEEE Network Computing and Applications 2005 (NCA 05). Boston, MA

[10] Balazinska, Balakrishnan, and Karger. INS/Twine: A Scalable Peer-to-Peer Architect ure for Intentional Resource Discovery Pervasive 2002 - I–ternational Conference on Pervasive Computing, Zurich, Switzerland, August 2002. © Springer-Verlag.

[11] Balasubramanian, Krishnakumar, Jaiganesh Balasubramanian, Jeff Parsons, Aniruddha Gokhale, Douglas C. Schmidt, "A Platform-Independent Component Modeling Language for Distributed Real-time and Embedded Systems," Proceedings of the 11th IEEE Real-Time and Embedded Technology and Applications Symposium, San Francisco, CA, March 2005.

[12] Birman, K.P. *Reliable Distributed Systems Technologies, Web Services, and Applications.*. 2005, XXXVI, 668 p. 145 illus., Hardcover ISBN: 0-387-21509-3

[13] Birman, K.P.. The Untrustworthy Services Revolution. To appear, February 2006, *IEEE Computer*.

[14] Birman, K.P. Can Web Services Scale Up? Ken Birman. *IEEE Computer*. Volume 38. Number 10. Pgs.107-110. October 2005

[15] Birman, K.P., Hillman, R., Pleisch, S. Building network-centric military applications over service oriented architectures. SPIE Defense and Security Symposium 2005. March 29-31, 2005. Orlando, Florida.

[16] Birman, K.P. "A Review of Experiences with Reliable Multicast." Software Practice and Experience Vol. 29, No. 9, pp, 741-774, July 1999.

[17] Birman, K.P., Mark Hayden, Oznur Ozkasap, Zhen Xiao, Mihai Budiu and Yaron Minsky. "Bimodal Multicast." *ACM Transactions on Computer Systems,* Vol. 17, No. 2, pp 41-88, May, 1999.

[18] Birman, K. P., and T. A. Joseph. "Exploiting Virtual Synchrony in Distributed Systems." *Proceedings of the Eleventh Symposium on Operating Systems Principles* (Austin, November 1987). New York: ACM Press, 123-138.

[19] Bauer, L., S. Garriss and M. K. Reiter, "Distributed Proving in Access-Control Systems," In Proceedings of the 2005 IEEE Symposium on Security and Privacy, pages 81-95, May 2005.

[20] Bauer, L, S. Garriss, J. McCune, M. K. Reiter, J. Rouse and P. Rutenbar, "Device-enabled Authorization in the Grey System," In Information Security: 8th International Conference, ISC 2005 (Lecture Notes in Computer Science 3650), pages 431?446, Springer-Verlag, September 2005.

[21] Castro, M., B. Liskov: Practical Byzantine Fault Tolerance and Proactive Recovery. *ACM Transactions on Computer Systems (TOCS)* 20(4): 398-461 (2002)

[22] Castro, M., R. Rodrigues, B. Liskov: BASE: Using abstraction to improve fault tolerance. TOCS 21(3): 236-269 (2003)

[23] Chandra, T., and S. Toueg. "Unreliable Failure Detectors for Asynchronous Systems." *Journal of the ACM*, in press. Previous version in *ACM Symposium on Principles of Distributed Computing* (Montreal, 1991), 325-340.

[24] Demers, A. et al. "Epidemic Algorithms for Replicated Data Management." *Proceedings of the Sixth Symposium on Principles of Distributed Computing*, (Vancouver, August 1987): 1-12. Also *Operating Systems Review* 22:1 (January 1988): 8-32.

[25] Dolev, D., and D. Malkhi. "The Transis Approach to High Availability Cluster Communication." *Communications of the ACM* 39:4 (April 1996): 64-70.

[26] Dolev, D., D. Malkhi, and R. Strong. "A Framework for Partitionable Membership Service." Technical Report TR 95-4. Institute of Computer Science, Hebrew University of Jerusalem, March 1995.

[27] Ghemawat, S; Gobioff, H; and Leung, S.T. "File and storage systems: The Google file system". *19th ACM Symposium on Operating Systems Principles (SOSP*), Bolton Landing, NY. October 2003.

[28] Gill, Christopher D., Douglas C. Schmidt, and Ron Cytron, "Multi-Paradigm Scheduling for Distributed Real-time Embedded Computing," IEEE Proceedings, October 2002.

[29] Gray, J., J. Bartlett, and R. Horst. "Fault Tolerance in Tandem Computer Systems." *The Evolution of Fault-Tolerant Computing*, A. Avizienis, H. Kopetz, and J. C. Laprie, eds. Springer-Verlag, 1987.

[30] Gray, J., Helland, P., and Shasha, D. "Dangers of Replication and a Solution". *ACM SIGMOD International Conference on Management of Data*. Montreal, Quebec, Canada. June, 1996.

[31] Gray, J., and A. Reuter. *Transaction Processing: Concepts and Techniques*. San Mateo, CA: Morgan Kaufmann, 1993.

[32] Gupta, I., A.M. Kermarrec, A.J. Ganesh. "Efficient Epidemic-Style Protocols for Reliable and Scalable Multicast." Proceedings of the 21st Symposium on Reliable Distributed Systems (SRDS 02), Osaka, Japan. October 2002. pp. 180-189.

[33] Gupta, I., K.P. Birman, P. Linga, A. Demers, R. Van Renesse. "Kelips: Building an Efficient and Stable P2P DHT Through Increased Memory and Background Overhead.". *Proc. 2nd International Workshop on Peer-to-Peer Systems (IPTPS '03)*, Oakland CA, 2003.

[34] Harrison, Timothy H., David L. Levine, and Douglas C. Schmidt, "The Design and Performance of a Real-time CORBA Event Service," Proceedings of OOPSLA '97, Atlanta, GA, October 1997.

[35] Holliday, J., D. Agrawal, A. El Abbadi: The Performance of Database Replication with Group Multicast. FTCS 1999: 158-165

[36] Holliday, J., D. Agrawal, A. El Abbadi: "Partial Database Replication using Epidemic Communication." ICDCS 2002: 485.

[37] Krishna, Arvind S., Emre Turkay, Aniruddha Gokhale, and Douglas C. Schmidt, "Model-Driven Techniques for Evaluating the QoS of Middleware Configurations for DRE Systems," Proceedings of the 11th IEEE Real-Time and Embedded Technology and Applications Symposium, San Francisco, CA, March 2005.

[38] Krishna, Arvind S., Cemal Yilmaz, Atif Memon, Adam Porter, Douglas C. Schmidt, Aniruddha Gokhale, and Balachandran Natarajan, "A Model-Driven Approach to Ensuring Continuity of Critical Properties in Distributed Systems," *IEEE Software* special issue on the Persistent Software Attributes, Nov/Dec 2004.

[39] Kubiatowicz, J. *et. Al.*: OceanStore: An Architecture for Global-Scale Persistent Storage. *Proceedings of Architectural Support for Progamming Languages and Systems (ASPLOS) 2000*, 190-201

[40] Kubiatowicz, J., D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, B. Zhao. "OceanStore: An Architecture for Global-Scale Persistent Storage." Proceedings of ACM ASPLOS, 2000.

[41] Lampson, B. "Hints for Computer System Design." *Proceedings of the Ninth Symposium on Operating Systems Principles* (Bretton Woods, NH, October 1993), 33-48.

[42] Lampson, B., Abadi, M., Burrows, M. and Wobber, T. Authentication in distributed systems: Theory and practice. *ACM Transactions on Computer Systems* 10(4), November 1992.

[43] Marian, T., Birman K, and van Renesse, R. A Scalable Services Architecture. In submission.

[44] Memon, Atif, Adam Porter, Cemal Yilmaz, Adithya Nagarajan, Douglas C. Schmidt, and Bala Natarajan, Skoll: Distributed Continuous Quality Assurance, Proceedings of the 26th IEEE/ACM International Conference on Software Engineering, Edinburgh, Scotland, May 2004.

[45] Oki, B., M. Pfluegl, A. Siegel, and D. Skeen. "The Information Bus—An Architecture for Extensible Distributed Systems." *Proceedings of the Thirteenth ACM Symposium on Operating Systems Principles* (Asheville, NC, December 1993). New York: ACM Press, 58-68.

[46] Ostrowski, Birman, and Phanishayee. The Power of Indirection: Achieving Multicast Scalability by Mapping Groups to Regional Underlays. In submission.

[47] Paunov, Stoyan, James Hill, Douglas C. Schmidt, John Slaby, and Steve Baker, "Domain-Specific Modeling Languages for Configuring and Evaluating Enterprise DRE System Quality of Service," Proceedings of the 13th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems (ECBS '06), Potsdam, Germany, March 2006.

[48] Pittel, B. "On Spreading of a Rumor." *SIAM Journal of Applied Mathematics* 47:1 (1987): 213-223.

[49] Pyarali, Irfan, Douglas C. Schmidt, and Ron Cytron, "Techniques for Enhancing Real-time CORBA Quality of Service," the IEEE Proceedings Special Issue on Real-time Systems, May 2003, co-edited by Yann-Hang Lee and C. M. Krishna.

[50] Reiter, M. K., and K. P. Birman. "How to Securely Replicate Services." *ACM Transactions on Programming Languages and Systems* 16:3 (May 1994): 986-1009.

[51] Reiter, M. K., K. P. Birman, and L. Gong. "Integrating Security in a Group-Oriented Distributed System." *Proceedings of the IEEE Symposium on Research in Security and Privacy* (Oakland, May 1992). New York: IEEE Computer Society Press, 18-32.

[52] Reiter, M. K., K. P. Birman, and R. van Renesse. "A Security Architecture for Fault-Tolerant Systems." *ACM Transactions on Computing Systems*, May 1995.

[53] Rowstron, A. and P. Druschel, "*Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems*". IFIP/ACM International Conference on Distributed Systems Platforms (Middleware), Heidelberg, Germany, pages 329-350, November, 2001.

[54] Rowstron, A., and Druschel, P.. "Storage management and caching in PAST, a large scale, persistent peer-to-peer storage utility". *18th ACM Symposium on Operating Systems Principles (SOSP)*, Banff, Canada. October 2001

[55] Rowstron, A., A.M. Kermarrec, M. Castro, P. Druschel: SCRIBE: The Design of a Large-Scale Event Notification Infrastructure. *Second International Workshop on Networked Group Communication*, Oakland CA, 2001: 30-43

[56] Rowstron, A., Kermarrec, A.M., Druschel, P., and Castro, M.. "SCRIBE: A large-scale and decentralized application-level multicast infrastructure". *IEEE Journal on Selected Areas in communications (JSAC)*, 2002.

[57] Rowstron, A. and P. Druschel. "*Storage management and caching in PAST, a large-scale, persistent peer-to-peer storage utility*", ACM Symposium on Operating Systems Principles (SOSP'01), Banff, Canada, October 2001.

[58] Van Renesse, R. and Johansen, H. "Fireflies: Scalable Support for Intrusion-Tolerant Overlay Networks," To appear, Operating Systems Design and Implementation (OSDI) 2006

[59] van Renesse, R., K.P. Birman and W. Vogels. "Astrolabe: A Robust and Scalable Technology for Distributed System Monitoring, Management, and Data Mining." *ACM Transactions on Computer Systems*, May 2003, Vol.21, No. 2, pp 164-206

[60] van Renesse, R., K.P Birman, D. Dumitriu, and W. Vogels. "Scalable Management and Data Mining Using Astrolabe." *Proceedings of the First International Workshop on Peer-to-Peer Systems (IPTPS).* Cambridge, Massachusetts. March 2002.

[61] van Renesse, R., K. P. Birman, R. Friedman, M. Hayden, and D. Karr. "A Framework for Protocol Composition in Horus." *Proceedings of the Fourteenth Symposium on the Principles of Distributed Computing* (Ottawa, August 1995). New York: ACM Press, 80-89.

[62] van Renesse, R., K. P. Birman, and S. Maffeis. "Horus: A Flexible Group Communication System." *Communications of the ACM* 39:4 (April 1996): 76-83.

[63] Schantz, Richard, Franklin Webber, Partha Pal, Joseph Loyall, and Douglas C. Schmidt, "Protecting Applications Against Malice with Adaptive Middleware," Certification and Security in E-Services stream of the 17th IFIP World Computer Congress, Montreal, Canada, August 25-30, 2002.

[64] W. Vogels, Birman, K., R. Van Renesse. "Six Misconceptions about Reliable Distributed Computing." Proceedings of the Eighth ACM SIGOPS European Workshop. Sintra, Portugal, September 1998.

[65] Zhao, B., Y. Duan, L. Huang, A. D. Joseph, J. Kubiatowicz. Brocade: Landmark Routing on Overlay Networks. *IPTPS 2002*, 34-44

[66] Zhao, B., L. Huang, J. Stribling, S. C. Rhea, A. D. Joseph, and J. Kubiatowicz. "Tapestry: A Resilient Global-scale Overlay for Service Deployment", *IEEE Journal on Selected Areas in Communication.*

[67] Zhao, B. Y. Duan, L. Huang, A.D. Joseph and J.D. Kubiatowicz. "Brocade: landmark routing on overlay networks", *First International Workshop on Peer-to-Peer Systems (IPTPS)*, Cambridge, MA. March 2002.

[68]

Acronyms

AAW	Anti-Air Warfare
AF	Air Force
AFOSR	Air Force Office of Scientific Research
AFRL	Air Force Research Laboratory
API	Application Programming Interface
APPC	Advanced Program to Program Communications
ATG	Automatic Tactic Generation
BFT	Byzantine Fault Tolerance
C2	Command and Control
C4ISR	Command, Control, Communications, Computers, Intelligence, Reconnaissance and Surveillance
CMU	Carnegie Mellon University
COI	Community of Interest
CORBA	Common Object Request Broker Architecture
COTS	Commercial, Off-the-Shelf
CPU	Central Processing Unit
DARPA	Defense Advanced Research Project Agency
DDoS	Distribued Denial of Service
DDS	Data Distribution Service
DISA	Defense Information Systems Agency
DNS	Domain Name Server
DoD	Department of Defense
FCS	Future Combat System
FEC	Forward Error Correction
FTP	File Transfer Protocol
GCC	GNU Compiler Collection
GIG	Global Information Grid
GOTS	Government Off-the-Shelf
GUI	Graphical User Interface
HIMARS	High Mobility Artillery Rocket System
HTTP	HyperText Transfer Protocol
IPC	Interprocess Communication
IAI	Information Assurance Institute
I/O	Input/Output
IP	Internet Protocol
J2EE	Java 2 Enterprise Edition
JBI	Joint Battlespace Infosphere
JDAM	Joint Directed Attack Munition
JIF	Java + Information Flow
JMS	Java Messaging Service
JSF	Joint Strike Fighter

LAN	Local Area Network
LCS	Littoral Combat Ship
MDD	Model-Driven Development
MLRS	Multi-Launch Rocket System
NCES	Network-Centric Enterprise Systems
NDDS	Network Data Distribution Service
NP	Non-Polynomial
NSA	National Security Agency
OCC	Operations Center Commander
OMG	Object Management Group
OS	Operating System
P2P	Peer-to-Peer
PCES	Program Composition of Embedded Systems
PRET	Partnerships for Research Excellence and Transition
QA	Quality Assurance
QoS	Quality of Service
Q/U	Query/Update
R&D	Research and Development
RMI	Remote Method Invocation
RPC	Remote Procedure Call
RT	Real Time
RTOS	Real Time Operating System
SOA	Service-Oriented Architectures
SOAP	Simple Object Access Protocol
SoS	Systems of Systems
SSDS	Ship Self Defense System
TBMD	Theatre Ballistic Missile Defense
UAV	Unmanned Aerial Vehicle
UDDI	Universal Description, Discovery and Integration
WSDL	Web Services Description Language
WSMR	White Sands Missile Range
XML	eXtensible Markup Language