Yalta: A Secure Collaborative Space for Dynamic Coalitions

Gregory T. Byrd, Fengmin Gong, Chandramouli Sargor Timothy J. Smith,

Abstract—

Dynamic coalitions are the means through which a group of entities with common interest collaborate to achieve significant mission objectives. The coalition participants are often distributed across geographic areas, have different levels of technological capability, and have language and cultural barriers to effective communications. In the information age, the execution of all the coalition functions will be heavily mediated through information technologies. The purpose of our effort is to develop, demonstrate and deliver a scalable, reliable, and survivable application platform that enables efficient formation, operation, and management of secure dynamic coalitions suitable for the military environment.

Our approach consists of four key innovative ideas: (1) Create a Secure Collaborative Space from shared space technologies developed in the distributed computing research community. (2) Build a certification authority (CA) service based on threshold cryptography, integrated into the collaborative space. Intrusion-tolerance and scalability will be achieved through multiple groups of shared CA servers. The CA servers will be integrated with external authentication and policy mechanisms. (3) Implement a certificate revocation notification (CRN) service based on event notification mechanism in shared space. Both prompt notification by subscription/publication and pulling of certificate revocation list (CRL) will be supported. (4) Support use of the collaborative space as a coordination channel for setting up other special communications. This mediated communications can accommodate special needs such as machine translation, very high-bandwidth data exchange or encrypted communication using non-standard algorithms.

I. Introduction

U.S. military operations increasingly involve multinational alliances, or coalitions. [1] Coalition partners are geographically distributed and often have technology, language, and cultural barriers to effective communication. Also, the membership of the coalition may change over time, at both the individual member and the participating country levels.

Existing technologies do not provide adequate support for efficient creation and management of coalitions, nor for secure collaboration within dynamically established mission-specific coalitions. The *Yalta* project is dedicated to developing technologies for the efficient support of successful coalition operations. In particular, we address the following challenges: (1) building a distributed computing platform that will foster ubiquitous development and deployment of coalition applications, (2) efficiently providing some of the critical infrastructure services for information assurance within dynamic coalitions, e.g. basic confidentiality and authentication services, and public key infrastructure, and (3) making this platform scalable, survivable, and extensible.

Our approach consists of four key innovative ideas:

- Create a Secure Collaborative Space from shared space technologies developed in the distributed computing research community.
- Build a certification authority (CA) service based on threshold cryptography, integrated into the collaborative space. Intrusion-tolerance and scalability will be achieved through multiple groups of shared CA servers.
- Implement a certificate revocation notification (CRN) service based on event notification mechanism in shared space. Both prompt notification by subscription and pulling of certificate revocation list (CRL) will be supported.
- Support use of the collaborative space as a coordination channel for setting up other special communications. This can accommodate special needs such as very high-bandwidth data exchange or encrypted communication using non-standard algorithms.

A prototype system, based on the JavaSpaces [2] shared space platform, is being built incrementally, from the basic platform to public key infrastructure services. The prototype will be evaluated in a testbed with at least a dozen machines and a hundred coalition members and with a collaborative application.

II. Coalition Requirements

Effective, dynamic coalition application platform has the following properties:

- Collaborative platform that is secure, scalable, survivable, and extensible.
- Platform must provide authentication, privacy, non-repudiation, integrity and access control services through standards-based credential and policy management interfaces.
- Dynamically manage users and their attributes, such as access authorizations.
- Dynamically manage services with discovery, negotiation, and delivery mechanisms.

III. Project Overview

A. Objectives

The primary objective of the Yalta project is to develop a scalable infrastructure to support secure dynamic coalitions. This infrastructure consists of two main features: first, a secure means of sharing information among coalition

G.T. Byrd: North Carolina State University, Raleigh, NC.
F. Gong: Intruvert Networks, Inc., Santa Clara, CA.
C. Sargor: CoSine Communications, Inc., Redwood City, CA.
T.J. Smith: MCNC, Research Triangle Park, NC.

This work is funded by the DARPA Information Technology Office (ITO), under contract F30602-00-C-0068.

members; second, scalable and responsive support for authentication, allowing coalition membership to be granted, verified, or revoked in a secure and timely manner.

In Yalta, coalition members share information through the use of a shared tuplespace, described in Section III-C. First developed in the Linda coordination language [3], a tuplespace is a content-addressable shared memory. Like a traditional shared memory, the space allows producers and consumers of data to be decoupled in time and space. Like traditional message passing, tuples are passed by value, not by reference, and may be freely copied. We must add a security overlay to the tuplespace, so that only authenticated members of the coalition have access to the contents, and so that copies of tuples are encrypted as they are transferred to and from the space. The security overlay will be largely transparent to applications, so that existing space-based applications can easily be adapted to serve the needs of the coalition.

The nature of a dynamic coalition demands that security services be both scalable and responsive. Coalitions may consist of hundreds or thousands of end users, as well as processes acting on behalf of those users. Also, the organizations participating in the coalition may only partially trust each other, and membership in the coalition may change quickly. The authorizations for entire groups of users may be revoked, and the sensitivity of the shared information means that such a revocation must be disseminated as widely and as quickly as possible.

Yalta provides scalable security services in two ways. First, we use threshold cryptography to implement a scalable and intrusion-tolerant certification service, as described in Section III-D. Second, we provide a Certificate Revocation Notification (CRN) service, described in Section III-E, to make sure that end users are notified quickly in the event that particular users are no longer to be trusted.

In the following sections, we describe the overall system architecture, details of the tuplespace, and the scalable certification and revocation services.

B. System Architecture

The basic Yalta architecture is shown in Figure 1. At the top level is a *coordination space*, used to coordinate access to a group of coalitions. Below the coordination space is a collection of coalition spaces. Each coalition has a dedicated space for communication among its own members.

The coordination space is available to the public (or to all potential members of coalitions). A potential member can look for or solicit members of a new coalition. It may look for existing coalitions that may be appropriate for its mission. It may look for coalitions of which it is already a member, in case it has forgotten how to attach to those coalition spaces. Even though the coordination space is considered a public space, there are still authentication and privacy concerns, particularly with respect to operations that change the contents of the space.

In addition to the coordination space, each active coalition has its own space, used for coordinating the tasks of the coalition members. The use of a separate space for each coalition limits the scope of matching for coalition members. It also provides some access control through partitioning, since only authenticated coalition members know how to access the space. The coalition space must be protected as much as possible from malicious coalition members. Admission to the coalition must be policy-driven; for example, the admissions policy may require the agreement of some threshold number of current coalition members.

Coalition space services have to deal with members who leave the coalition. The keys that protect sensitive information should be changed quickly, so that the ex-member may not have access to new data or to old data that had not been retrieved earlier. It is also desirable that the space itself change identity, so that the ex-member may no longer attempt to access or modify objects.

C. Secure Collaborative Tuplespace

The tuplespaces in the Yalta prototype are implemented using JavaSpaces [2], which combines the concept of a tuplespace with the object-oriented Java language. The tuplespace services are built from the distributed programming facilities of Jini [4], a Java-based package for network-enabled devices. The tuples in JavaSpaces are objects which implement the Entry interface. There are no special methods or state variables associated with an Entry; it is merely the base class of objects that are handled by a JavaSpace object. All Entry-based objects (tuples) must have public member variables, so that any object may inspect the parts of the tuple during a match. Also, member variables cannot be native Java types, such as integer. They must be classes, so that a null object can be used as a wildcard during a match.

The JavaSpace object implements the operations that control the contents of the tuplespace. `Write` places an object into the space. `Read` retrieves a copy of a matching object. `Take` removes a matching object from the space. The non-blocking forms of `read` and `take` are called `readIfExists` and `takeIfExists`. In addition to these basic operations, JavaSpaces provides three other mechanisms for space-based coordination: leases, transactions, and distributed events.

A lease is a time limit associated with a tuple. The lease determines how long the tuple remains in the space. If the lease expires without being renewed, then the tuple is removed from the space. The lease can be used to insure that information retrieved from the space is timely. Also, it allows for graceful recovery if tuples do not get removed as expected.

For dynamic coalitions, the lease operation can be useful in maintaining certificates and revocation lists. A certificate authority (CA) could be required to renew its lease on a CRL, for instance, to make sure that other entities retrieve only the latest version.

A transaction is a collection of operations that are performed atomically. Either all of the operations succeed, or none of them does. If all operations occur successfully, the

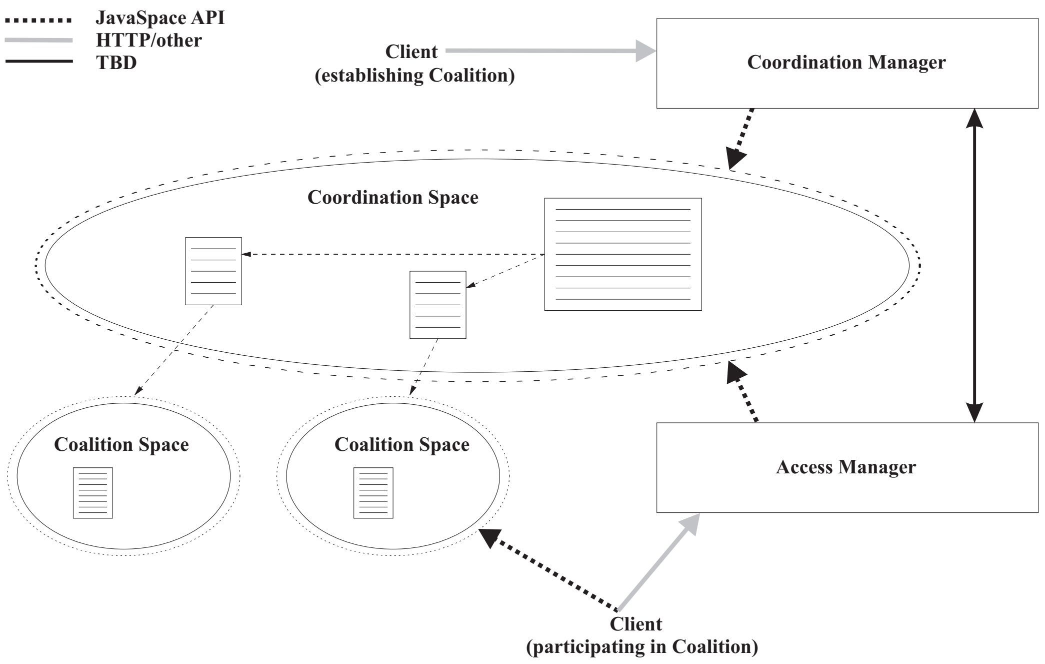

Fig. 1. Space-Based Architecture for Dynamic Coalitions

transaction is said to be committed. If it is not committed, then the transaction is aborted, either explicitly by an object that detects an error or automatically if the transaction does not commit or abort before its lease expires. In the context of dynamic coalitions, consider an operation that requires coalition membership to be verified. A transaction could be used to check membership, perform the operation, then re-verify membership. If a member leaves the coalition while the operation is in progress, then it will be aborted; any state changes related to the aborted operation will not be seen by the other coalition members. The distributed event mechanism allows an object to be notified if a particular type of tuple is added to the space. The consumer object issues a notify request, with a tuple template, to the space. If an object is added that matches the template, a notification is sent by the space to the consumer object. The request, of course, may have a lease associated with it, so that the consumer may express interest in the event for only a limited time.

Notification can be used to quickly convey revocation information to interested parties. For example, when one member begins a transaction or conversation with another, it may request notification if the other's certificate becomes revoked. Upon revocation, the CA can insert a tuple into the space, notifying interested parties even before the CRL is published.

The JavaSpaces specification does not address security, so we must create a security overlay to address those concerns. The overlay must provide access control and authentication, so that only authentic members of the coalition may access the space, as well as encryption, so that data flowing to and from the space cannot be captured. Objects within the space are not encrypted, since that would hinder the lookup process within the space. (Applications may choose to encrypt certain data fields, of course, before placing objects into the space.) Our initial approach will be to use SSL [5] between the client and the space, using persistent or resumable connections to reduce overhead. Other approaches will also be investigated.

The following sections describe how specific services—certification and revocation—can be implemented in a scalable and reliable manner using a space-based computing environment. The examples will be described in terms of JavaSpaces, and each example will note features that must be added to the space to support secure collaboration.

D. Survivable and Scalable Certification Service

Issuing a certificate for a person, organization or entity implies placing a certain amount of trust in the binding between the information contained in the certificate (name, address, credentials) and the associated public key. Typically, the organization issuing the certificate would require valid proof of such information before it issues the certificate. The actual task of generating the certificate, once all information has been verified, is performed by a certificate server which signs the information contained in the certificate with the issuing organization's private key. The certificate may then be added to a certificate database for widespread distribution. Depending on the validity period of the certificates, the certificate database can grow considerably over a period of time. The certificate server is usually a single host with adequate resources (CPU, memory, etc.).

This model has several shortcomings. Storing the sensitive private key on a single host (certificate server) makes it a single point of failure, both from the perspective of equipment failure and security breach. The private key can be revealed if the certificate server is compromised.

Once the certificate server is compromised, all certificates issued previously would have to be invalidated and new certificates issued. As the certificate database can contain many entries, a large number of certificates would have to be reissued in a relatively short time. While a single host certificate server architecture is designed to handle normal certificate requests, it is typically not capable of handling such large scale re-certification demands. Although it is possible to configure multiple machines with the same private key to aid in large scale certificate generation, this introduces additional problems of securing multiple machines and administering them in a secure fashion. Furthermore, compromise of any one of these machines would render the private key invalid.

One promising solution to the above problems is to use techniques based on threshold cryptography for sharing a private key among multiple hosts such that the compromise of a few of these hosts will reveal no information about the private key. [6]

An RSA private key consists of a modulus N and a secret exponent d. The modulus N is publicly known and is a product of two large primes that are secret. The exponent d is a positive integer less than N. Both encryption and decryption using the private key require an exponentiation to the power of d. For a given message M, the ciphertext is generated by computing $M^d \text{mod} N$. The key to sharing the private key among multiple hosts is to pick k random integers $d[i], i = 1...k$, in the range $[-N, N]$, such that their sum equals d. The individual shares $d[i]$ are computed in a distributed fashion by the k machines such that each machine learns nothing about the shares of others.

To encrypt a message M, the client sends M to each of the k machines which compute $M^{d[i]} \text{mod} N$ and send the result back to the client. The client then multiplies all of these individual results to obtain the actual ciphertext. (This is true because $(M^{d[1]} \text{mod} N)(M^{d[2]} \text{mod} N) \cdots (M^{d[k]} \text{mod} N) = M^{(d[1]+d[2]+\ldots d[k])} \text{mod} N = M^d \text{mod} N$.)

The threshold approach builds intrusion tolerance into the system, since the private key is never reconstructed on a single machine. To compromise a private key, all of the k hosts must be compromised. Moreover, one can introduce diversity by using hosts with different hardware platforms, operating systems, and so forth, to make it more difficult to compromise multiple machines. Given a set of n machines, it is possible to share the private key among k of them ($k < n$) such that any k of the n machines may be used for private key based operations [6]. Since $k < n$, this implies that there exists more than one such grouping of k machines. A single such grouping is identified by means of a unique group ID. Unlike the single server case, survivability is built into this approach; even if $n - k$ machines are offline, it is possible for the remaining k machines to continue to function. A client specifies a unique grouping of k machines by including the group ID as part of the request. By selecting appropriate values of k and n, it is possible to achieve improvements in throughput over the single server case. However, latency—the time to compute a single signature—will be higher due to increased communication costs.

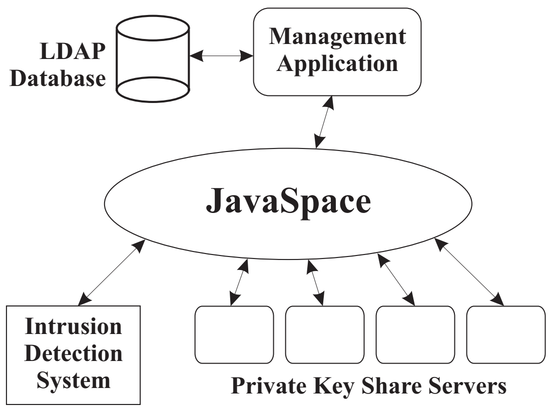

Fig. 2. JavaSpaces Certificate Authority Infrastructure

One of the key factors to improving throughput is to balance the load among the n servers. The choice of servers is based on the group ID specified by the client. This means that the client(s) must pick the group ID's in such a way that a busy server is less likely to be included than an idle one. One could pick the group ID's in a random fashion. While this reduces the probability that a busy server is included, it however, does not guarantee it. It is easy to see that with the client specifying the group ID, it is not possible to achieve optimal throughput, as the client has no way of knowing which machines are busy and which are idle at a given instant of time. A more distributed approach is required that permits the share servers themselves to load balance among one another to maximize throughput and minimize latency.

We propose to implement a JavaSpaces based CA infrastructure using shared private keys to address the limitations of the approaches discussed above. Figure 2 depicts the JavaSpaces CA architecture, consisting of different entities connected to one another via a common JavaSpace. The CA is comprised of a number of share servers sharing the CA's private key; a subset of these hosts is required to perform the certificate signing. A management entity is responsible for placing objects into the space to be signed and reading/removing it from the space after it has been signed. The management entity may choose to use the newly signed certificates to update an LDAP certificate database so that the certificate may also be accessed via traditional means (outside of the space). Finally, an Intrusion Detection System (IDS) monitors the share servers for compromise by periodically requesting messages to be signed and verifying the signature. Algorithms have been published that allow the IDS to identify exactly which share server is corrupt.

When a certificate needs to be signed, the information is simply written into the space. When not actively signing certificates, the share servers are blocked on a `take` operation waiting to retrieve objects that need to be signed. The `take` operation succeeds for one of the share servers, which then determines which group of servers will be used to sign the object, based on a distributed load balancing algorithm. Each of the share servers in the specified group then retrieves the object and signs it using its partial key share. Note that the order in which the share servers sign the object is not important; therefore, the signing can happen in parallel. Since the responsibility of the application using the JavaSpace CA infrastructure is merely to place objects to be signed into the space and retrieve it once it has been signed, the application is no longer burdened with having to decide which set of share servers will be enlisted to perform the signature. This decoupling of information to be signed from a specific selection of a group of share servers allows us to develop distributed load balancing algorithms to achieve optimal throughput. These load balancing algorithms will be investigated as part of this proposed research. As mentioned earlier, space based systems decouple the sender and receiver of information by supporting retrieval of objects from the space based on their contents. This means that the sender no longer needs to specify the recipient's address to communicate. This has an important security implication for our JavaSpace CA infrastructure, in that the different share servers that need to communicate with one another during the shared key generation process do not need to know the physical identity (IP address) of the other share servers. Coordination could be achieved through other means such as by using a unique identifier that only distinguishes one share server from another but does not reveal any information about its identity. This leads to the idea of share server anonymity, which can further hinder an attacker from trying to compromise a group of share servers to obtain the private key.

Another interesting subject of research is the potential for dynamically adding share servers from among a set of trusted machines in a coalition to share the burden of authentication, credential signing and other compute intensive tasks. Platform heterogeneity is not a concern for JavaSpace based systems. Java also makes it easier to share code for private key operations (shared key generation, signing etc.) among the servers. As a coalition grows and new trusted members join the coalition, the set of trusted machines would perhaps grow as well. This implies that the space based architecture may be self-scaling if appropriate mechanisms can be provided to dynamically add share servers and coordinate compute intensive operations efficiently through distributed load balancing algorithms. Our assumption is that the CA infrastructure developed would be focused on meeting the needs of dynamic coalitions as opposed to being a complete solution for the planetary PKI infrastructure. To this end, we assume that each dynamic coalition would require a CA infrastructure of its own to support secure operations such as admitting/removing members and granting/revoking credentials. When new members are to be added to a coalition, it would be the responsibility of the joining member to produce a certificate proving his/her identity to the satisfaction of the CA serving that coalition. Upon successful verification, the CA would issue the newly joined member a certificate of its own, valid for the expected duration of the coalition. This certificate would be honored by all existing members of the coalition. We could also require that the new member generate a new public-private key pair for the new certificate. This approach has the added benefit of isolating coalition members from certificate revocations in the planetary PKI infrastructure.

E. Certificate Revocation Notification Service

One of the key responsibilities of the CA infrastructure is to notify users when published certificates have a change in status. One key certificate status change is revocation, which can happen for many reasons: change of information in the user's certificate, compromise of user's private key, or even the compromise of the CA's private key. Another example of certificate status change is renewal, when validity periods are changed for the private or public key.

Traditionally, notification of revocation has been accomplished by means of Certificate Revocation Lists (CRL) containing the serial numbers of revoked certificates. As the list of revoked certificates can grow very large over a period of time, processing the CRL's to search for the revocation status for a particular certificate can be a time consuming operation. Also, the timely delivery of such large CRL's is another problem. To alleviate this problem, delta CRL's have been proposed, in which the CA only includes information about certificates that have been revoked since the previous update. However, even with delta-CRL's, the timeliness of revocation information is not adequate for certain critical transactions that depend on up-to-date certificate status information.

To address this problem, an online certificate status protocol (OCSP) [7] has been developed to allow applications to obtain the status of a certificate just prior to its use. While this addresses the timeliness issue, it can place a huge burden on the CA infrastructure due to frequent status update requests.

A dynamic coalition is usually put together to achieve a specific mission and coalition members are granted certain credentials to accomplish the mission. In this context, it is very important that timely revocation information be made available to coalition members. For instance, while discussing an attack strategy with a coalition member, it is vitally important to know if the member's certificate has been revoked during the course of the conversation because he was found to be an enemy spy. It is certainly impractical to constantly keep checking on the status of the peer's certificate during a conversation or a discussion. It would be more appropriate for the user of a certificate (relying party) to be notified of a certificate revocation by the CA infrastructure.

We propose an alternative to the client polling model found in OCSP, the subscriber/publisher model, that would greatly reduce the burden on CA infrastructure. This could be accomplished by creating an subscription/publication extension to OCSP (OCSP-X). OCSP-X would allow the requester to subscribe to the status of a particular certificate for a certain period of time. The publisher would notify the subscriber when any change had occurred in the certificate status during that time period. As with OCSP, this extension addresses certificate status in general, and not just revocation. To utilize OCSP-X, the requester does have to have a server infrastructure to receive the published information.

We intend to utilize the event notification functionality of Jini and JavaSpaces to develop such a certificate revocation notification service. Traditional CRL-based certificate revocation will also be supported.

The Jini Distributed Event Model incorporated into the JavaSpaces technology allows processes to register their interest in receiving certain event notifications with the space. Processes can solicit notification of the arrival of entries (revoked certificates) into the space that match a given template (serial number). When an entry matching the template is added to the space, an event is generated by the space and sent to the process. As the space handles the event registration and notification, the burden on the CA is significantly reduced. Moreover, as part of the event registration the process can specify a certain time period (lease) during which it wishes to receive such event notification. This allows a process to solicit event notification only while it is actively using a certificate.

The Jini Distributed Event model allows notification of events in one space to be sent to processes in another space or even to another non-space Java Virtual Machine. This is important in the context of dynamic coalitions, since each coalition would most likely be implemented as a separate JavaSpace. However, it is likely that the same individual may be participating in multiple coalitions. Revocation information would therefore need to be distributed to multiple spaces.

The Jini Distributed Event Model does not guarantee the reliable delivery of event notification. Moreover, the source can send the notification more than once if it is not convinced that the receiver has received it. Duplicate notifications are handled by means of a monotonically increasing sequence number. Of equal importance is the authenticity of event notification. The recipient of the revocation event must be convinced that it is the CA that has originated the event notification.

To address reliable delivery concerns, we propose the OCSP-X heartbeat message. The publisher will issue a message containing a timestamp and status information of the subscribed certificates or CRLs to each subscriber. The message could be authenticated with a one-way hash message authentication code (HMAC) to the subscribers on a negotiated interval. The cryptographic overhead of HMAC is orders of magnitude less then an RSA signature verification.

To implement the HMAC function, the client and server must negotiate and agree upon a symmetric key to used in the function. We are investigating appropriate cryptographic key-agreement protocols.

With the proper key-agreement protocol the publisher could batch status information for a particular subscriber reducing the message traffic and cryptographic workload.

No acknowledgement of the HMAC message from the client to the server is necessary. OCSP-X clients who do not receive a heartbeat within a certain period, that receive unverified heartbeat messages, simply degrade the level of trust in the subscribed certificates.

Developing a generalized reliable and authenticated event notification mechanism will be one of the problems we intend to address as part of building a secure JavaSpace.

The Simple Certificate Validation Protocol [8] allows a client to delegate certain PKI related tasks such as monitoring CRL's and OCSP responses to an SCVP server. The exact set of PKI functions performed by the SCVP server depends on the level of trust placed by the client in the server. Use of an SCVP server can decouple the user from the esoteric world of PKI.

Even with the refinement and reduction of workload that OCSP-X brings, generating heartbeat messages for a large number of subscribers would heavily load the OCSP-X responder and generate a high volume of message traffic. To further distribute the load of rapid dissemination of certificate status information would be to have coalition members delegate responsibility for certificate status monitoring to a OCSP-X aware SCVP server.

Each user in a coalition requesting event notification could register with a trusted OCSP-X aware SCVP server instead of directly from the space. The SCVP server could batch a unique set of requests or renewed requests to the OCSP-X responder.

The SCVP would be the single entity informed of which certificates are in use and request event notifications on behalf of the users. When this entity is notified of a revocation, it could directly invoke methods in the user's processes that would notify the user of the certificate revocation.

The use of batch subscriptions or renewals by the SCVP server and the use of batch OCSP-X heartbeat messages would greatly reduce and distribute message flow volume and cryptographic overhead of verification. This concept fits especially well within the Jini model and uses the Jini Lease holder, LeaseManager and Landlord pattern.

F. Experimentation

We will perform several basic experiments over our testbeds in order to fully evaluate the solution. Our basic plan of attack is to create two test labs, one located at NCSU and the other at MCNC. Each test lab will be comprised of a cluster of 4–6 compute nodes interconnected with 100 Mbps Ethernet LANs. Coalition members will be simulated by software processes called member agents. Each PC will capable of generating from 1 to 25 member agent process, each with a unique name and resource/need attributes. The labs will each connect indirectly to the NCNI GigaPOP network. Three main type of experiments are planned.

Experiment1: basic functionality testing. This experiment will utilize mock applications which exercise the lifecycle of a coalition: creation, membership changes, and termination. Secure coordination and communication services will be tested by having several mock applications invoking such services with each other. This experiment will be conducted independently on both the testbeds at MCNC and NCSU.

Experiment2: robustness testing. This experiment will also use mock applications. The emphasis for this experiment is on repeated invocation of various space services, and on continuous operation of the space without reset. Our aim is not at exhaustive testing of error conditions but rather to get a reasonable level of comfort with the robustness of the implementation. This experiment will also be performed independently on the two testbeds at MCNC and NCSU.

Experiment3: quantitative measure of performance. This experiment is designed to measure the real performance of the space. We will implement a real application in the secure collaborative space (multi- party videoconferencing is one candidate). These tests will be performed using both testbeds, connected via the NCNI GigaPOP. Possible measures to be taken include:

- Latencies for some of the basic coalition operations;
- Changes of these latencies with increasing number of active coalition members;
- Scaling property of the certification signing throughput with respect to the number of signing servers;
- Notification latency for certificate revocation notification service;
- Scaling property of notification latency with respect to coalition size or subscriber population.

It should be noted that some of these measures will be highly dependent on the testing conditions (e.g. system load). We will document as accurately as possible the testing environment to facilitate accurate interpretation of the results.

IV. Future Work

A. Integration with other efforts

There are several areas of IA&S programs with which we expect to find opportunities for integration. First, we are concentrating on the infrastructure services sub-area of the Dynamic Coalitions program. We clearly recognize the need for a trust and policy management framework and we expect to have hooks in our secure collaborative space for working with this framework.

Second, our secure collaborative space is an infrastructure technology for dynamic coalitions. The coalition applications can be autonomous agents in an Autonomic Information Assurance system, or multiple analysis agents in a Cyberspace Command and Control hierarchy. All these agents can achieve secure and effective coordination through a secure collaborative space. Furthermore, our space is in essence a very flexible platform for coalition applications. Any specific mission logic or policy can be implemented by developing a new coalition application. We expect that domain specific knowledge will be required to develop these applications. Such integration effort will be best conducted by technical personnel from those domains. However, we will provide any technical assistance to such parties should the need arise.

B. Service Broker

In real military or business dynamic coalitions, we foresee a strong desire to have efficient brokerage services, which introduce parties with complementary needs and resources. In a military setting, such services can be used for selecting candidate countries to form a coalition for peace-keeping missions. In a business setting, it can be used to find joint-venture partners, or just to find new recruits with special skills.

We believe that the space-based environment provides an excellent opportunity for supporting these brokerage services. Buyers and sellers can describe their needs and resources, placing these descriptions as objects in the top-level coordination space. A service broker process can identify potential matches, and the space can offer mechanisms for negotiating the formation of a new coalition.

C. Scalable Tuplespaces

The Yalta prototype will be based on an implementation of JavaSpaces that uses a centralized server. This is obviously not a solution that promotes high scalability or reliability, since the single server can easily become a bottleneck and a single point of attack or failure. Our design, however, does not rely on a centralized server, and we plan to incorporate new and existing research on distributed tuplespaces [9] into future implementations.

V. Conclusion

Tuplespace environments are a natural medium for collaborative applications. Producers and consumers of data are decoupled in space and time. Content-based addressing and the capability to exchange platform-independent executable code allow applications to adapt to changing requirements and conditions.

By integrating security with the space, Yalta provides a means for sharing information among trusted coalition partners. The space is used for member collaboration, as well as to implement the security services required for coalition applications.

Our CA and CRN services together constitute a survivable and scalable public key infrastructure for the collaborative space, which is the basis for information assurance for all the dynamic coalitions operating in the space. The collaborative space also provides a basic service for creation of new spaces. Therefore, support for multiple coalitions with multiple overlapping- participants is a native capability with the collaborative space. Last but not least, many basic security services will be provided as part of the common platform. These services will deal with many security threats including information theft and corruption, message replay, participant impersonation, and cyberspace denial-of-service.

References

[1] T. Gibson, "An architecture for flexible multi-security domain networks," in *Network and Distributed System Security Symposium (NDSS'01)*, Feb. 2001, pp. 63–72.

[2] E. Freeman, S. Hupfer, and K. Arnold, *JavaSpaces: Principles, Patterns, and Practice*, Addison-Wesley, Menlo Park, CA, 1999.

[3] D. Gelernter, "Generative communication in Linda," *ACM Transactions on Programming Languages and Systems*, vol. 7, no. 1, pp. 80–112, Jan. 1985.

[4] K. Arnold, B. O'Sullivan, R. Scheifler, J. Waldo, and A. Wollrath, *The Jini Specification*, Addison-Wesley, Menlo Park, CA, 1999.

[5] A. O. Freier, P. Karlton, and P. C. Kocher, "The SSL protocol, version 3.0," Internet Draft, Netscape Communications Corporation, Mar. 1996.

[6] M. Malkin, T. Wu, and D. Boneh, "Experimenting with shared generation of RSA keys," in *Symposium on Network and Distributed System Security*. Internet Society, Feb. 1999, pp. 43–56.

[7] M. Myers, R. Ankey, A. Malpani, S. Galpering, and C. Adams, "X.509 internet public key infrastructure Online Certificate Status Protocol (OCSP)," RFC 2560, IETF, June 1999.

[8] A. Malpani, P. Hoffman, and R. Housley, "Simple Certificate Validation Protocol (SCVP)," IETF Internet Draft, work in progress, Nov. 2000.

[9] J. B. Fenwick, Jr. and L. Pollock, "Issues and experiences in implementing a distributed tuplespace," *Software: Practice and Experience*, vol. 27, no. 10, pp. 1199–1232, Oct. 1997.