Information Technology Journal 6 (3): 380-389, 2007
ISSN 1812-5638

Reliability and Security Enhancements for a User-Controlled Lightpath Provisioning System

Jing Wu, Michel Savoie, Hanxi Zhang and Scott Campbell
Communications Research Centre Canada, 3701 Carling Avenue, Ottawa, Ontario, Canada, K2H 8S2

Abstract: User owned and managed optical networks offer new benefits compared to carrier networks. A user-controlled lightpath provisioning system is designed to address the network management challenges. The prototyped management software has a service-oriented architecture and uses the Jini and JavaSpaces technologies. Since the new management system is a distributed system, it has special requirements on system reliability and security. System reliability is improved by using lease-based resource and service sharing, transaction-based resource reservation, concurrent access and persistent storage of resource information and federation clustering. With these system reliability enhancements, the system is able to maintain its integrity even in the event of partial system failures. Since the new management system may be deployed over a public Internet infrastructure, secure access to the management modules is required. The application of existing system security technologies to the new management system is analyzed.

Key words: Network management software, software reliability, distributed software security, user-controlled lightpath provisioning

INTRODUCTION

There are basically two types of user owned and managed optical networks: metro dark fibre networks and long-haul wavelength networks (Wu *et al*., 2003; St. Arnaud *et al*., 2004). Schools, hospitals and government departments are acquiring their own dark fibres in metropolitan areas. They participate in so-called condominium dark fibre networks to better manage their connectivity and bandwidth. They light up the fibres with their own equipment and interconnect their fibres to other institutions, commercial service providers or Internet Exchanges. In the long-haul area, many providers are selling or leasing point-to-point wavelength channels. Some providers are offering condominium wavelength solutions, where a number of users share the capital costs of deploying long-haul optical networks. In return, each user in the condominium consortium owns a set of wavelength channels.

User owned and managed optical networks offer new benefits compared to carrier networks. In user-owned optical networks, the cost of bandwidth is substantially reduced, as it now largely becomes a capital cost rather than an ongoing service charge. The user is able to optimize the overall resource utilization. The user purchases dark fibres and/or wavelength channels from a number of independent suppliers and participates in condominium wavelength networks. Therefore, the user has more flexibility in network planning and deployment and is able to negotiate better deals from different suppliers. The user may fine-tune the usage of each resource. User-managed networks reduce Internet costs via remote peering and transit. Since the user directly owns and manages an optical network, the bandwidth and quality of service are guaranteed. User-controlled lightpath provisioning is a traffic engineering mechanism for inter-domain applications. The primary application for this technology is high-end scientific research in high-energy physics, astronomy, bio-informatics, etc.

User-controlled optical networks face new technical challenges in resource management, operation co-ordination and information storage management. Managing networks with resources from different suppliers is a new issue that has not been fully addressed by existing network management techniques (Wu *et al*., 2003). Now, only the customer has complete visibility of its own network and no provider can see all the network elements. This is in contrast to the traditional centrally managed networking technologies, e.g., Generalized Multi-Protocol Label Switching (GMPLS) and Automatic Switched Optical/Transport Network (ASON/ASTN), which assume that the provider manages network elements and a common management system is used. Rather than any provider, the user is in a better position to decide the optimal solution for protection and restoration. The protection and restoration need to be coordinated among multiple providers. The collaboration among multiple independent users is critical for end-to-

Corresponding Author: Jing Wu, Communications Research Centre Canada, 3701 Carling Avenue, Ottawa, Ontario, Canada, K2H 8S2 Tel: +1 613 9982474 Fax: +1 613 9908382

end connection provisioning. User-managed networks adopt the peer-to-peer architecture, in which users peer with each other. Each user not only receives transport services from other users but also contributes new transport services. During the establishment of an end-to-end connection, each connection segment is set up on a peer-to-peer basis. Central guiding intelligence and arbitration of conflicts may be necessary, but day-to-day management and per connection control are decentralized. How to search and take control of partner's shared resources has to be addressed. Policy enforcement, authorization and authentication have to be applied.

The research on building a user-controlled network was initiated by CANARIE, Inc., Canada's advanced Internet development organization. The Internet inter-domain routing protocol, Border Gateway Protocol (BGP), was extended to facilitate lightpath routing. OBGP takes the BGP routing table at the source node and queries along Autonomous System (AS) paths to check if there are available lightpaths (Blanchet *et al*., 2001). Later on, it was realized that network management systems offer more flexibility in providing new functions and features than a distributed control plane. In 2003, CANARIE launched a directed research program to design and prototype network management systems enabling user-controlled lightpath provisioning (St. Arnaud *et al*., 2003). A Web-service based management system was designed primarily focusing on the partitioning and concatenation of inter-domain channel resources and transferring ownership among users (Boutaba *et al*., 2004), assuming a central database provides a global view of the inter-domain topology and resource availability on all inter-domain links. A policy based admission control was proposed in another prototype system (Truong *et al*., 2004) to regulate the user's request to the connection provisioning. The nature of inter-domain management was addressed by using a global directory to provide links to the management modules for every domain.

We are motivated to design and prototype a management system directly confronting the challenges of user-centered network management. To provision a connection covering multiple domains, the management modules for the involved domains need to collaborate in a peer-to-peer manner. It is critical to avoid any central resource database or directory. Since optical networks are dominant transport networks, we prototyped our system for optical networks such as wavelength-routed Wavelength Division Multiplexing (WDM) networks, Synchronous Optical Networks (SONET) or Synchronous Digital Hierarchy (SDH) networks. Therefore, we call our system a User-Controlled Lightpath Provisioning (UCLP) system. However, the design is generally applicable to any type of inter-domain circuit-switched or bandwidth guaranteed packet-switched connection provisioning. Recently, our research partner at Technical University of Catalonia, Spain, extended the UCLP system to manage Multi-Protocol Label Switching tunnels.

Since the UCLP system has a distributed architecture, the system reliability and security are of particular significance. The whole system is required to operate even if service modules are temporarily unavailable. For example, when a communication failure between two management modules occurs, the management system should not run into an unstable state. The management function should be available after the failed communication is recovered. The access of the management modules should be protected by security features. In the present research we discuss the reliability and security enhancements for the UCLP system.

JINI TECHNOLOGY

Building on top of the Java technology, Jini provides an infrastructure for defining, advertising and searching services in a network (Li *et al*., 2000; Edwards and Rodden, 2001). The Java Virtual Machine (JVM) provides a hardware-independent software execution platform. The Jini infrastructure spans multiple JVMs that may run on different devices or computers connected via a network. Unless we want to emphasize a module is a user or provider of a function, we generally called a Jini module as a Jini service, because a module may provide services to other modules and meanwhile the same module may be a client of other modules utilizing other modules' services. Since Jini services may be spread throughout a network, Jini technology needs to define, advertise and search services and then enable services to communicate to each other over a network.

Service lookup and discovery: The Jini Lookup Service (JLS) is central to the Jini technology. The primary function of the JLS is to let a service know the existence of other services. On the other hand, a service announces its presence by registering to a JLS. Fault-tolerance may be provided by using multiple JLS's for the same group of Jini services. A JLS itself is a Jini service and can be registered to other JLS's and therefore can be searched for.

A Jini service locates a JLS by using service discovery. Two types of service discovery are supported: multicast and unicast service discovery. In a multicast service discovery, a Jini service sends out a multicast

message and then listens to possible replies from active JLS's. Multicast service discovery does not require a Jini service know the location of a JLS. But, it depends on the IP multicast. A JLS may not be located within the IP multicast range of the Jini service. In a unicast service discovery, a Jini service specifies the Universal Resource Locator (URL) of a JLS. In addition to the two service discovery mechanisms, a JLS may actively announce its presence so that other services within the same multicast domain are notified.

A Jini service registers its proxy to JLS's, so that available Jini services may be searched for and the registered proxies may be downloaded to where the Jini services are to be called. A proxy of a Jini service is a stub that contains an interface to the Jini service. Several proxy downloads or uploads are involved in a Jini system. First, after a Jini service discovers a JLS, a proxy for the JLS (called a registrar) is downloaded to the Jini service. Second, the Jini service uses the registrar to register to the JLS by uploading the Jini service's proxy. Third, when other services search for a Jini service in the JLS by describing an interface type (written in the Java Programming Language) and possibly, other attributes. The Jini service's proxy may be downloaded to other clients when a match is found. Other services use the downloaded proxy to invoke the Jini service.

Remote method invocation and dynamic code downloading: The Java Remote Method Invocation (RMI) supports distributed Jini services to communicate to each other (Edwards and Rodden, 2001). Jini services that are implemented as Java objects may be distributed on different JVMs and be hosted by different networked machines. RMI supports a Java object to utilize another remote object's service. The two objects running on different JVMs are remote to each other. The service is implemented as methods in the object. The service user and provider have a common understanding about what functions the service offers and what are the input parameters and return values by using the same group of Java interfaces to describe the service. Two objects, i.e., stub and skeleton, can be generated for a service object using a compiler. The stub, i.e., the service proxy, runs on the client JVM, taking care of networking details, such as underlying communication protocols, packing and unpacking input parameters to the service and return values from the service. The skeleton runs on the service JVM providing similar functions as the stub (Fig. 1). The operations of stub and skeleton are transparent to the client and service objects, as if the client and service objects were communicating directly.

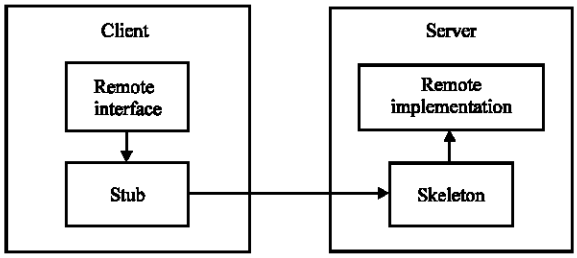

Fig. 1: Functions of a stub and a skeleton in RMI

The stub and skeleton use the serialization mechanism to convert the content of the corresponding object into a byte stream that can be sent over a network. The input parameters and return values may be of primitive types (such as integer, character, boolean, etc.), or objects. Primitive-type input parameters and return values are marked by the source end, sent over a network and recognized by the destination. Representing the content of a particular instance of a class needs special treatments, because there is no standard approach to mark and recognize objects as for primitive types. The serialization mechanism is used to package up the member data within an object. A serialized object may be written to a file, or stored as an intermediate object of a special Java class type, called marshaled object, before it is recovered (i.e., unmarshalled) to a functional object that can run on a JVM. The JLS stores and forwards the Jini service proxy as a marshaled object.

The class file corresponding to a serialized object is dynamically downloaded into the JVM where the object is to be reconstructed. The class file is required to create a functional object, because the serialization only packages up the states of all the member variables within the object, not the bytecodes for the object itself. To interpret and translate the received serialized object, the class file for the object needs to be downloaded into the JVM where the reconstruction of the object is to take place. The place from where the class file is downloaded is called the codebase of the class and is packaged into the serialized object. Tagging the codebase to the serialized data stream is called annotating the stream with the codebase, because the stream depends on the codebase class to be interpreted. After receiving the byte stream representing the serialized object, the receiver discovers the codebase of the class file, downloads the class file and then reconstructs an object identical to the object that is serialized at the sender. The codebase is specified as a URL. For example, when a client downloads a Jini service proxy from a lookup service, the client also needs to download the stub class file from the Jini service

proxy's codebase, in most cases, from a directory or a Java ARchive (JAR) file that the service made available for downloading.

Jini federation: Jini services for the same management domain create a Jini federation, in which a Jini service may access other Jini services in the same Jini community via service proxies dynamically downloaded from a local JLS. Jini services in the same federation collectively offer a group of functions under the same administration. Therefore, basic notions of trust, identification and policy are presumably agreed upon.

The integration of a JLS with other types of naming or directory services provides a means of hierarchical lookup. One option is to include other lookup services as service objects in a JLS. The other option is to place a reference to a JLS in other naming or directory services, providing a means for clients of those services to access a Jini system.

A MANAGEMENT TOOL DESIGN BASED ON JINI AND JAVASPACES

System architecture: Figure 2 shows the distributed deployment of two copies of the UCLP system, one for each independently managed domain (i.e., a federation). Within one management system for a federation, there are six key components: a JLS, an instance of JavaSpaces for storage of Light Path Objects (LPOs), a Jini Service Access Point (SAP), an LPO service, an instance of switch communication service for each switch in the transport layer and a Grid SAP.

Lightpath management services: Figure 3 shows the system architecture in more detail and indicates the main service methods provided by the component interface. The management services provided by our system are classified into two groups: those only available to administrative users and those available to all users. The

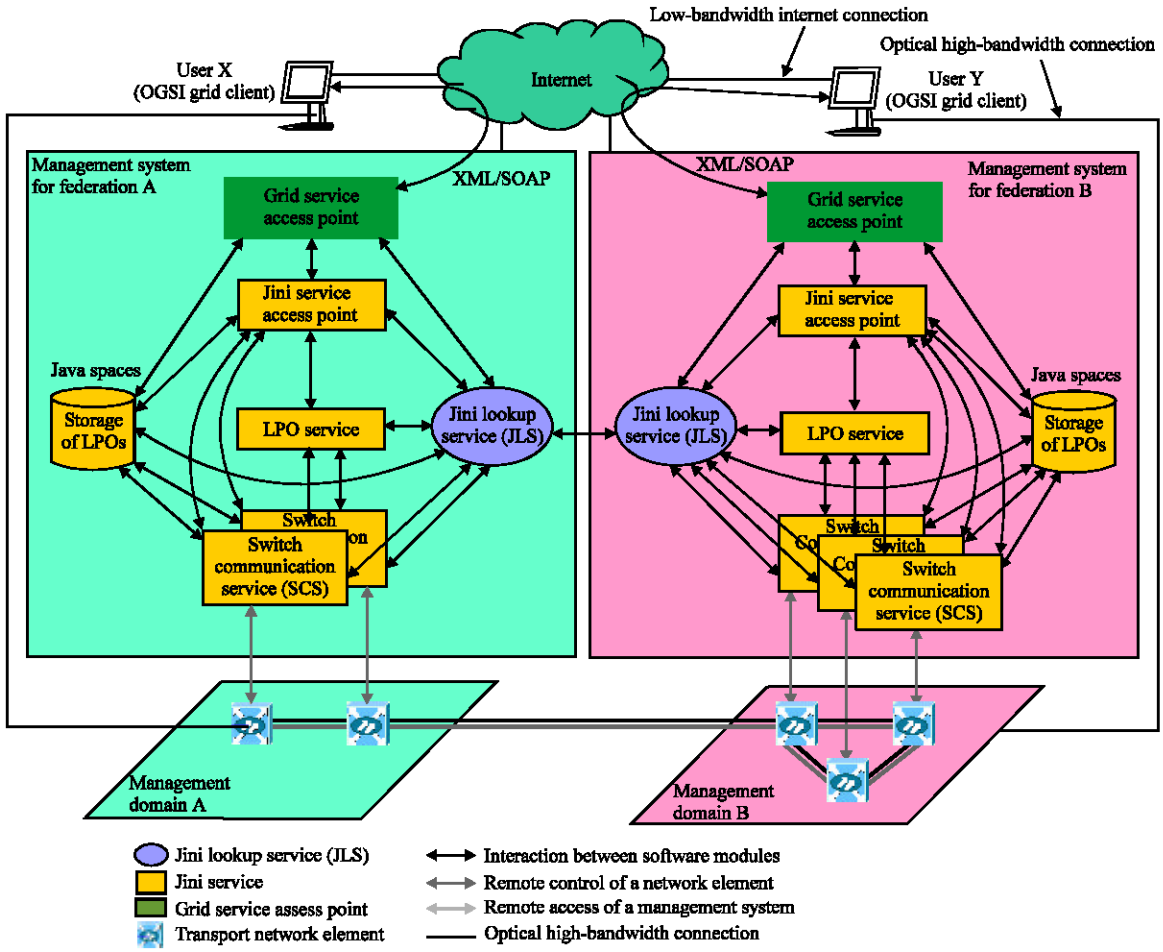

Fig. 2: Distributed deployment of the UCLP system

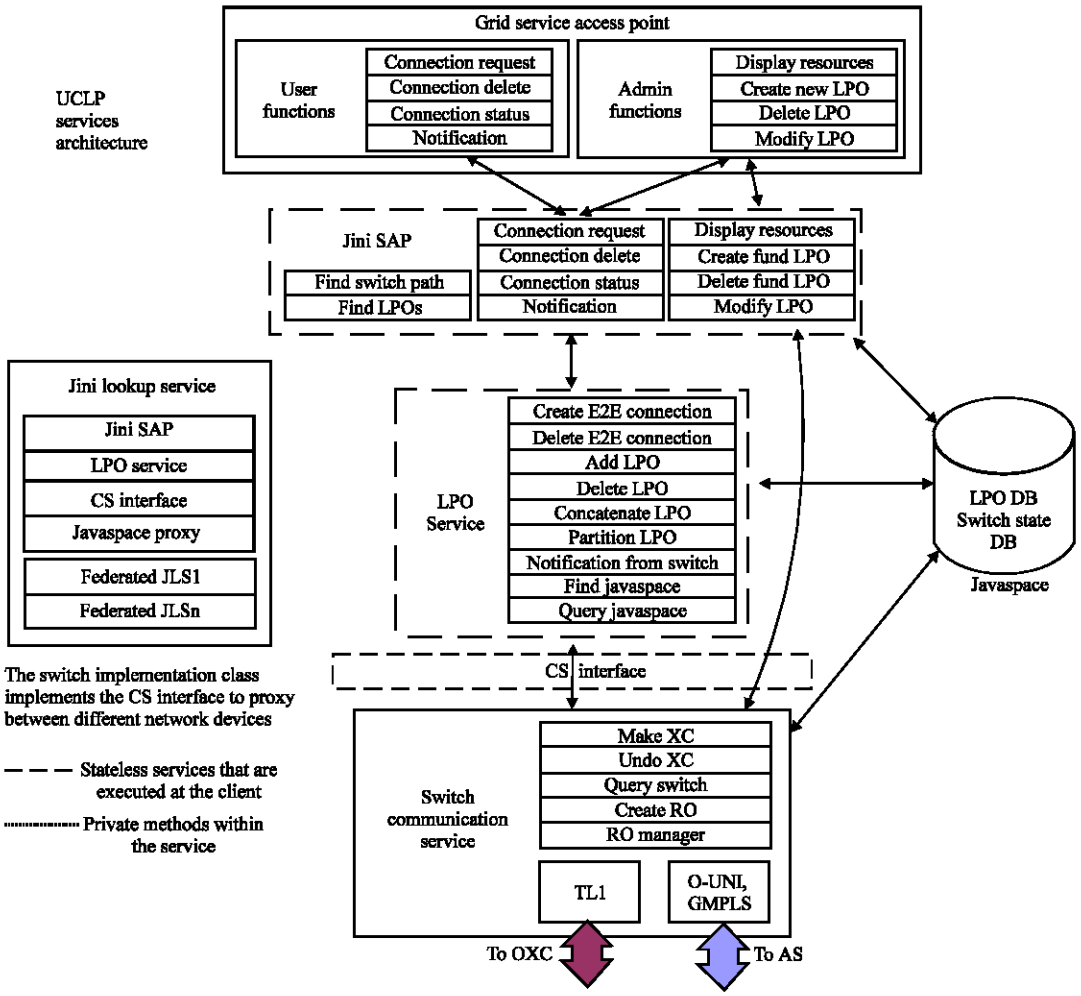

Fig. 3: Detailed UCLP services architecture

latter include in particular ConnectionRequest by which a user can request the establishment of an end-to-end connection from a given entry port of a given switch to a given exit port on another given switch, possibly belonging to a different federation. One of the functions reserved to administrative customers is the addition of new physical links to the available optical network.

In our design, LPOs are objects stored in JavaSpaces. An LPO is an abstraction of a lightspan. It is associated with a set of attributes and methods that enable possible peering to other LPOs at a switch to create an end-to-end connection or a longer lightspan. Supported customer operations on LPOs include: concatenating two LPOs, partitioning one LPO into many LPOs sharing common start and end points but with smaller bandwidth allocations and reserving/using/releasing LPOs. The administrative operations include adding new LPOs and deleting LPOs corresponding to changes in the physical layer and the allocation of new resources.

For the execution of the ConnectionRequest, the Jini SAP uses the internal methods FindSwitchPath and FindLPOs. The latter searches through the pertinent JavaSpaces to look for LPOs with attributes suitable for the end-to-end connection to be built. It also uses the functions provided by the methods of the LPO service.

RELIABILITY AND SECURITY ENHANCEMENTS

Lease based resource and service sharing: A leasing mechanism is used to ensure the integrity of the UCLP system. A lease is a grant of access to a resource or a service over a time period. The provider of a lease tries its best to make the promised resource or service available for the lease time. Therefore, the user of a lease may make the reasonable assumption that to a certain extent, the leased resource or service can be used. However, the availability of the leased resource or service is only based on the provider's best-effort and is not absolutely

guaranteed, since the availability of the leased resource or service may also depend on some factors that are beyond the provider's control. Frequent check-up and renewal of a lease may improve the lease user's confidence on the availability of the leased resource or service and reduce the negative impact of their unavailability. A trade-off is decided between frequent lease renewals and the processing load of the UCLP system.

The registry of a Jini service is based on a lease. A UCLP module registers to a JLS with a lease time. The UCLP module renews its registry to confirm its validity (i.e., the service definition attributes and interface are unchanged) and its availability (i.e., at least the service is still connected to the network). When a UCLP module fails to renew its registry before the expiration time, the uploaded proxy in the JLS expires and is automatically removed, so that no future download of the proxy is possible and no new client of the service is to be accepted. With a notification mechanism, an expiration of a UCLP module's proxy in the JLS may trigger a notification to clients who have downloaded the proxy. Upon receiving such a notification, the clients who use an affected service are made aware of the unavailability of the service.

Each LPO has a lifetime in the UCLP system, i.e., each LPO is associated with a lease time. When a root LPO is initially configured using the UCLP system, the integrity of resources corresponding to the root LPO is verified. Over time the UCLP system may lose track of the integrity of resources, especially when the notifications are inadequate. The resource from a provider may be based on a lease contract which requires the resource being returned to the provider after the lease expires. So the lease based root LPO inventory management provides administrators with a tool to clean up obsolete root LPOs.

The constituent LPOs for an inter-domain lightpath are leased from the related domains. The lifetime of the lightpath cannot exceed the lease time of any of its constituent LPOs. To extend the lifetime of the lightpath, negotiation needs to be conducted to renew the constituent LPOs' lease time. When a lightpath expires, the lightpath is split into the root LPOs that were concatenated to create the lightpath and the LPOs are returned to their original owners. If a constituent LPO's lease time is not expired yet, the lease for that LPO is to be explicitly cancelled. Just as creating a lightpath involves taking the leased LPOs out of their database and adding an LPO for the created lightpath into a database, splitting a lightpath into its constituent LPOs involves the reverse operations.

Transaction-based resource reservation: The primary purpose of using a distributed transaction based resource reservation in the UCLP system is to ensure an atomic and consistent resource usage. The creation or deletion of a lightpath involves distributed services, possibly distributed in different domains. The more LPOs are used and the more domains are involved, there is an increased possibility of having undesired events happen. The number of event combinations may become large, where some service calls are successfully finished but others fail. Compared to the design of a processing logic to handle each scenario, the transaction based resource reservation is much easier to handle and the processing logic is much more straightforward and clean. All the service calls involved in the creation or deletion of a connection should be organized in a transaction, such that either all service calls are committed (i.e., successfully finished), or none of them is committed as if nothing had happened. All the LPOs for a lightpath creation should be successfully reserved, or no LPO is reserved at all. Inconsistent partial LPO reservations should never happen, otherwise this would result in partially reserved LPOs become missing or unaccounted for in the UCLP system and unusable to any users.

Jini supports a framework, called a two-phase commit protocol, to coordinate a distributed transaction. A transaction manager is a Jini service providing synchronization among distributed services, so that all services in a transaction come together at certain time, in known and consistent system states. Jini provides a distributed transaction manager, called mahalo.

Figure 4 illustrates the two-phase commit protocol. First, a transaction session is created and transaction participants join the transaction session. Upon the request from the transaction manager to prepare for the commit, participants prepare the required work. It is important that the required work is able to roll back if necessary. If any participant fails to prepare, all other participants are asked to roll back to the state before the preparation, as shown in the lower part of Fig. 4. The failure of a participant may be explicitly reported to the transaction manager by an abort-message or implied by the transaction manager when a time-out for a response occurs.

Concurrent access and persistent storage of resource information: The JavaSpaces service is used to store LPOs and concurrently access LPOs. The JavaSpaces service is a core Jini service that provides a high-level means of creating collaborative and distributed applications (Freeman and Hupfer, 1999). We use the term

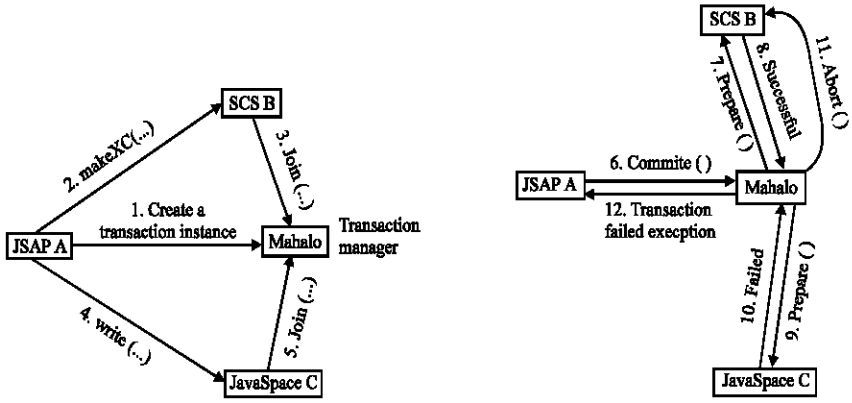

Fig. 4: Jini two-phase commit protocol for a distributed transaction

space to refer to a JavaSpaces service implementation, i.e., an instance of the JavaSpaces service. Supporting concurrent access of LPOs stored in a space is key to the UCLP system. Since a space may serve more than one user, the requests for using the same LPO may cause a contention. The JavaSpaces service has a built-in function that only allows one of multiple requests to successfully acquire the LPO while all others are rejected. That is, a space guarantees that the same LPO is never allocated to more than one request at any time.

Because the JavaSpaces service supports leasing, transaction and distributed event notification, as other Jini services, it is convenient to use the JavaSpaces as an object storage in the UCLP system. The basic operations on a space include write (i.e., place an LPO into a space), read (i.e., obtain a copy of an LPO matching a given template) and take (i.e., remove an LPO matching a given template and return it to a requestor). The basic and the derived (such as the conditional read or take) operations on a space can be controlled by a Jini transaction manager. When an LPO is written into a space, a lease object is created for the LPO. The time period of the LPO to exist in the space is governed by the lease, which can be renewed or cancelled by a holder of the lease object. The distributed event notification is used for a client to monitor the state changes of a space, e.g., the appearance or disappearance of a certain type of LPOs.

The UCLP system builds an LPO service to access a space, so that the access policy control can be enforced. The JavaSpaces service does not provide access control. When a client obtains a proxy for a space, the client acquires complete control over all LPOs in the space. Thus, the access control needs to be done by the service that utilizes the JavaSpaces service. Direct access to a space should be strictly controlled to protect the usage of network resources. The LPO service verifies the access rights to use a given LPO before any operation on the LPO.

JavaSpaces also offers persistent storage of LPOs. Once an LPO is written into a space, even if the space restarts or reboots, the LPO remains in the space. The persistent storage is based on logging the LPO information in a file on a computer hard disk drive. As long as the file is preserved, the space's persistent storage is offered.

Federation clustering: A federation manager is introduced to facilitate the clustering of Jini federations. To link two federations together, their JLS's need to register with each other. So partner federations need to discover or search for each other's JLS's. JLS discovery can be realized as Jini multicast discovery, unicast discovery and multicast announcement. The multicast discovery and multicast announcement are limited by the IP multicast range and reliability. The unicast discovery requires the priori knowledge of the IP address and port number of each partner federation's JLS. Thus, the unicast discovery does not scale well. With the federation hierarchy, a higher level federation facilitates the discovery of Jini lookup services for the lower level federations. A high level federation uses a federation manager, which is implemented as a space containing entries of the IP address and port number of all lower level federations below it. A lower level federation dynamically registers its lookup service IP address and port number with the federation manager (Fig. 5). A federation manager also notifies its lower level federations the appearance or disappearance of a JLS.

System security

Secure socket layer (ssl): To securely transfer objects across a network, SSL is used to encrypt RMI data streams and thus data streams between Jini services. SSL provides data stream security at the session layer on top of TCP/IP. SSL offers four functions: server authentication, data encryption, message integrity and

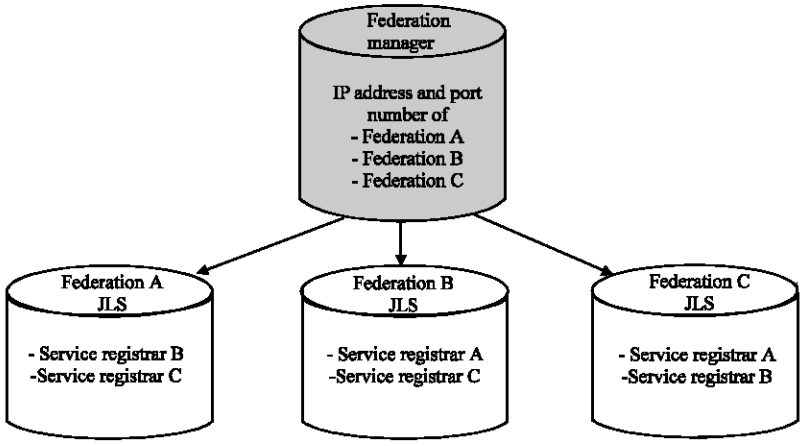

Fig. 5: Jini federation manager in the federation clustering

optional client authentication (Lail, 2002). Compared to encrypting IP flows using the IPsec protocol or encrypting RMI data streams using public and private keys, SSL demands less computation by using the same session key to encrypt and decrypt RMI data streams. In the session establishment phase, a session key is derived from a randomly generated number. The secure distribution of the session key between the authenticated server and client is achieved by the client's encryption of the session key using the server's public key and the server's decryption of the session key using the server's private key. Practically, only with the server's private key, can the encrypted session key be decrypted and only the server knows its own private key, so nobody else can intercept the session key. The encrypted RMI data streams can only be decrypted by using the right session key, which is solely known to the involved client and server.

The server and client authentication in SSL has limitations in authenticating dynamically discovered services. In SSL, the server authentication assumes the client knows the identity of the server; the client authentication assumes the server maintains a list of authorized clients who are allowed to access the service. The server authentication only proves to the client that the server is the real one that the client intends to connect to. It is the client's responsibility to make sure which server to trust and then to connect to. Similarly, the client authentication only verifies that the client is whom the client claims to be. The server needs to know which clients are allowed to use its service before any clients connect to the server. However, in Jini, clients and servers dynamically discover each other. It is not required that they know each other's identity in advance. A trade-off needs to be made between the effectiveness of SSL security and the dynamics and spontaneousness of the Jini networking environment.

There is a security gap between the data stream security provided by the SSL and the system security observed by a Jini client, because the client's communication to a remote service uses the RMI mechanism, which uses dynamically downloaded service proxies (i.e., service stubs) at the client (Hasselmeyer *et al*., 2000). The client needs a mechanism to verify the trust associated with the downloaded proxy. By attaching an authentication authority's signature to the proxy and using a public key based decryption, the client may build a trust for the signed proxy. A different approach is proposed in Jini version 2.0 to verify proxy trusts without using public/private key based infrastructures.

Java security model: To securely execute a dynamically downloaded Java class, Jini adopts the Java security model. A security manager provides authorization to dynamically download classes by specifying which dangerous operations are allowed for each class. The security manager offers flexible, fine-grained security policy control. Based on the codebase and the optional signatures associated with c the class file, the security manager looks up the security policy file whether a permission on a specific operation is granted. Examples of such operations include executing other application programs, shutting down the JVM, accessing other application processes, accessing system resources (such as print queues, event queues, system properties and windows), file system and network operations, etc. By attaching a third-party's signature to the bytecode, a bytecode is signed to guarantee the bytecode's origination and the integrity of the bytecode.

This bytecode signing procedure is similar to the certificate signing in the SSL and also public-key based decryption is used. Multiple third-parties may sign the same bytecode, resulting in potential matching multiple policy entries in the security policy. In such situations, permissions are granted in an additive fashion.

Jini adds customerized permissions (Li *et al*., 2000). One such extension is the discovery permission. A Jini client or service may be granted permissions to join a specific group of JLS's. This restricts the JLS's that a Jini client or service can discover in a multicast discovery. The risk is reduced for a Jini client or service running into an unknown or undesired JLS. In the UCLP system, the domain that a Jini service (including a JLS) serves for is defined. Therefore, the multicast discovery permission can be properly set up. The location of the JLS's remains floating, so it is flexible to use different JLS's (e.g., backup JLS's) at various locations.

The Java security model is static and operates at the per class granularity. The security permissions are pre-configured in policy files. Pre-configured policies cannot be modified during the execution of an object. Although the bytecode verifier examines downloaded bytecodes based on basic Java design rules, it is advantageous to delay granting security permissions until an object is going to be executed. Before an advanced proxy verification finishes, granting security permissions is risky.

Jini Extensible Remote Invocation (JERI) security features: To securely use a dynamically downloaded proxy to communicate to a remote service, JERI is implemented in Jini version 2.0 (Venners, 2002). JERI supports security features such as invocation constraints, remote method control and the trust verification model. JERI is an extension of the Java RMI model.

Proxy trust is a new security issue in Jini, because a Jini client relies on a dynamically downloaded proxy to communicate to a Jini service. JERI makes a good trade-off between the flexibility and the security of the Jini architecture by limiting pre-configurations required for security. Two assumptions are made for a client to verify a downloaded service proxy: i) the client knows the server's identity; and ii) the client has a locally pre-installed small bootstrap proxy to securely connect to a server. The flexibility of using dynamically downloaded service proxies is maintained by not limiting where the proxies are downloaded from and who signs the proxies. JERI avoids using public-key based proxy authentication because of the complexity associated with the authentication authority management, public key distribution to all potential clients and signing large number of proxies that may change over time.

JERI supports dynamic policy for a downloaded Jini proxy in addition to existing static permissions in the Java security model. Dynamic policy allows a Jini client to delay the permission granting until after the client has fully verified the proxy trust. A client may specify constraints on the behaviour of remote invocations through a downloaded proxy. A server may specify its constraints on the service based on the incoming remote requests. The constraints are assigned to individual remote invocations as opposed to the entire proxy as for static permissions in the Java security model. When a JERI proxy wants to invoke a remote method, the client's and the server's constraints are combined for that particular method, taking into consideration the execution context related constraints such as time related constraints. For example, a client may require a server authentication to execute a remote method. A client may require the server to be authenticated as any subset of a list of names. Independently, a server may require a client authentication as well. Both a client and a server may choose to require encryption on the data streams between them, verifying the data is unaltered during transmission, etc.

CONCLUSIONS

The service-oriented architecture is used in the management system for user-controlled lightpath provisioning. The architecture is proven to be modular and easy to maintain. Jini and JavaSpaces are suitable for the management system thanks to their rich functions such as supporting leasing, transaction and event notification. We demonstrated our prototype management system in several events showing its capability in setting up lightpaths directly controlled by users.

The system reliability is a fundamental requirement for the user-controlled lightpath provisioning system. The major system reliability enhancements include lease-based resource and service sharing, transaction-based resource reservation, concurrent access and persistent storage of resource information and federation clustering. The management system's integrity is improved by properly using these features.

Since the management system transfers information over public Internet infrastructures and uses dynamically downloaded management modules or their proxies, system security is required. Secure socket layer is used to securely transfer information over public networks. Java security functions, such as granting execution permissions, help to securely execute a dynamically downloaded Java class. Jini extensible remote invocation offers new security features that improve our system security. These security features are analyzed in the context of our management system.

ACKNOWLEDGMENTS

The research is partially funded by CANARIE's directed research program on UCLP. We thank Bill St. Arnaud at CANARIE, Inc. for his leadership and innovative vision for the UCLP research. We thank Prof. Gregor von Bochmann at the University of Ottawa for his contributions in the discussions and system design and the team he led, including Jun Chen, Wei Zhang and Ling Zou for their contributions to the implementation of a prototype system. We thank Mathieu Lemay (École de Technologie Supérieure, Montréal, Québec); Sergi Figuerola (i2CAT) and Eduard Grasa, Joaquim Recio and Albert López (Technical University of Catalonia, Barcelona, Spain) for their participation in the discussions.

REFERENCES

Blanchet, M., F. Parent and St. B. Arnaud, 2001. Optical BGP (OBGP): Inter AS lightpath provisioning. IETF draft, draft-parent-obgp-01.txt.

Boutaba, R., W. Golab, Y. Iraqi and St. B. Arnaud, 2004. Lightpaths on Demand: A Web-Services-Based Management System. IEEE Commun. Mag., 42: 101-107.

Edwards, W. and T. Rodden, 2001. Jini Example by Example. Prentice Hall PTR. New Jersey, US.

Freeman, E. and S. Hupfer, 1999. Make room for JavaSpaces. Java World.

Hasselmeyer, P., R. Kehr and M. Voß, 2000. Trade-offs in a Secure Jini Service Architecture. In: Proceedings of Trends Towards a Universal Service Market (USM 2000). Munich, Germany.

Lail, B., 2002. Broadband Network and Device Security. Osborne/McGraw-Hill.

Li, S. *et al*., 2000. Professional Jini; Wrox Press Inc.

St. Arnaud, B., J. Wu and B. Kalali, 2003. Customer Controlled and Managed Optical networks. IEEE/OSA J. Lightwave Technol. Special Issue on Optical Networks, 21: 2804-2810.

St. Arnaud, B., A. Bjerring, O. Cherkaoui, R. Boutaba, M. Pott and W. Hong, 2004. Web services architecture for user control and management of optical Internet networks. Proceedings of the IEEE., 92: 1490-1500.

Truong, D., O. Cherkaoui, H. Elbiaze, N. Rico and M. Aboulhamid, 2004. A Policy-Based Approach for User Controlled Lightpath Provisioning. In: Proceedings of IEEE/IFIP Network Operations and Management Symposium, Korea, pp: 859-872.

Venners, B., 2002. www.artima.com/intv/jinisecuP.html

Wu, J., S. Campbell, M. Savoie, H. Zhang, G. Bochmann and B. St. Arnaud, 2003. User-Managed End-To-End Lightpath Provisioning Over CA*net 4. In Proceedings of National Fibre Optic Engineers Conference (NFOEC 2003), Orlando, Florida, USA. Sept 7-11, 2003, 2: 275-282.