Credentialed Secure Communication “Switchboards” (TR2001-821)

Eric Freudenthal, Lawrence Port, Edward Keenan,
Tracy Pesin, and Vijay Karamcheti
Department of Computer Science
Courant Institute of Mathematical Sciences
New York University
{*freudent, lport, woodiek, tracyp, vijayk*}*@cs.nyu.edu*

Abstract

Software development in distributed computation is complicated by the extra overhead of communication between connected, dispersed hosts in dynamically changing, multiple administrative domains. Many disparate technologies exist for trust management, authentication, secure communication channels, and service discovery, but composing all of these elements into a single system can outweigh principal development efforts.

The NYU Disco Switchboard consolidates these connectivity issues into a single convenient, extensible architecture, providing an abstraction for managing secure, host-pair communication with connection monitoring facilities. Switchboard extends the secure authenticated communication channel abstraction provided by standard interfaces such as SSL/TLS with mechanisms to support trust management, key sharing, service discovery, and connection liveness and monitoring.

We present an extensible architecture which is particularly useful in dynamically changing, distributed coalition environments. Applications that utilize Switchboard benefit from the availability of authentication, trust management, cryptography, and discovery, while retaining the simplicity of a common interface.

1 Introduction

The nature of distributed computation poses developers non-trivial connectivity challenges which considerably add to software development efforts. Applications needs may extend beyond socket pair manipulation, requiring support for authentication, authorization, message digests, and ciphers. Some distributed environments may require services to reside on dispersed hosts, necessitating service discovery and retrieval.

Increasingly, applications must operate in coalition environments, i.e. distributed systems involving multiple administrative domains featuring asymmetric and dynamic trust relationships. In this scenario, distributed devices are connected under separate administrative domains with dynamically changing sets of users. These participants must be authenticated and authorized by multiple domains, and a user’s proof of authorization must be monitored to ensure validity. Cellular phone users, for example, may roam through networks administrated by different mobile communications providers, and must have some way of being recognized and authenticated by each of these domains. The providers must be able to recognize the user as she enters the network, and if authorized, provide service. A user’s authorization may dynamically change (if the she were to cancel a plan, or neglect payment), in which case authorization must be dynamically revoked.

It is with an eye towards the increasing need to develop applications for distributed coalition environments that we present Switchboard. The Switchboard abstraction unites functionality for managing secure and continuously monitored connections, authentication, and trust management into a single substrate, presenting an API that extends the model of the BSD socket interface for TCP connections. Independent systems for these connective support technologies are all extant, but the composition of a system containing all of these elements presents significant challenges that can dwarf principal development efforts.

Switchboard runs as an inherently trusted component on a host, as does an operating system. Once a client asks Switchboard for a connection to a service, our infrastructure locates the service, establishes a connection to the appropriate host with the required level of security, and authenticates and authorizes all entities involved in the transaction. Support for continuous monitoring of participants' trust relationships is provided through a provided trust management interface.

Switchboard achieves this functionality via an extensible, modular architecture, structured around three main components. The Discovery Module is responsible for locating a desired service or finding a replacement host should a connection fail. The Registry Module maintains tables that track a local host's connections with remote hosts, as well as advertises available services on the local host. The Connection Module is responsible for initiating and maintaining inter-host connectivity, and establishing secure communication channels. In addition to these three components, Switchboard implements replaceable modules that implement cryptographic functionality and trust management.

We have implemented and evaluated Switchboard in the context of the DisCo project of the Distributed Sanctuaries effort at New York University. DisCo is investigating techniques for secure decomposable service instantiation in networked coalition environments.

The need for Switchboard's abstractions arose directly from our own efforts, and we envision a much broader application of this technology for distributed computation. The initial overhead of developing the Switchboard architecture has enabled us to concentrate our efforts on our larger research goals. We have found that development has been accelerated and facilitated by the abstraction, and have discovered several optimizations for inter-host communication.

This paper is organized as follows: Section 2 explores Switchboard functionality. Section 3 presents the Switchboard API and Architecture, followed by Switchboard Implementation in Section 4. Section 5 is an evaluation of our implementation, and Section 6 discusses related work.

2 The Switchboard Abstraction

2.1 Terminology: Services and Users

Throughout this paper we will use the term "services" to refer to objects desired by an entity on a remote machine. In this paper we restrict our attention to services as data feeds with unique identities, but in a larger sense they represent a desired resource, such as a communication channel, data feed, remote procedure, objects, or actual programs.

The term "user" in this paper refers to programs, agents, or other controlling entities which can make requests to the Switchboard. "Correspondent" refers to the corresponding entity to which a local entity is connected.

Below we refer to "trust relationship", a condition that is satisfied when corresponding entities may communicate based on valid authentication and authorization.

2.2 Summary of Abstraction

From a high-level perspective, the lifecycle of a Switchboard connection begins when a service advertises on its local Switchboard, allowing a client to make a request for it. A monitored connection is then established between client and service provider ensuring a valid trust relationship (refer to Figure 1).

Below we explore in greater detail the lifecycle of a switchboard request.

1. **Service Registration**

   Switchboard provides a distributed substrate that allows server agents to provide advertised services to client agents. Registration includes the service provider's identity (with proof), the service's generic identity, evidence that it has the right to provide the service, a trust management agent to authorize clients and their hosts, and an encryption agent that enumerates the set of acceptable ciphers for each connection.

2. **Client Request and Authorization**

   A client explicitly requesting a service provides its identity (with proof), the name of the requested service, a trust management agent to authorize a service provider and its host, and credentials that prove the client's eligibility to receive the service. Switchboard identifies a host qualified to provide the requested service. If the provider similarly accepts the client's credentials, a connection is generated that extends the semantics of TCP.

3. **Connection Monitoring**

   Once a trust relationship is established and a connection between client and service provider is made, Switchboard will monitor the authorization of all entities' access rights. The set of credentials that initially authorize a connection may require renewal throughout a connection's lifetime. To support this, interfaces are provided for both correspondents to transmit credential updates to their partner's trust managers. If a change in access rights violates any required trust relationship, the channel is closed.

2.3 Example Use of Switchboard

Consider the following scenario in a distributed coalition environment:

One machine, Camera 1, is responsible for capturing video from a connected camera and transmitting the video to authorized users in the distributed environment.

Two other hosts in the network, Host A and Host B, are authorized to view video. Consider the issues that arise when users on Host A and Host B wish to connect to a video server to receive Camera 1's video feed. We need some way of:

1. authenticating the identities of Host A, Host B, and their respective users.

2. validating that Host A, Host B, and their respective users are authorized to receive the video feed.

3. locating a host that can serve Camera 1's video feed - this may be Camera 1 itself or an authorized proxy server (potentially with its own identity) that distributes the broadcast.

4. setting up communication using security requirements for Host A, Host B, and their respective users.

5. ensuring that all parties involved in the connection maintain a valid trust relationship (by monitoring their access rights) so that the connection may continue.

6. terminating the connection should a trust relationship become invalidated or if either party voluntarily disconnects.

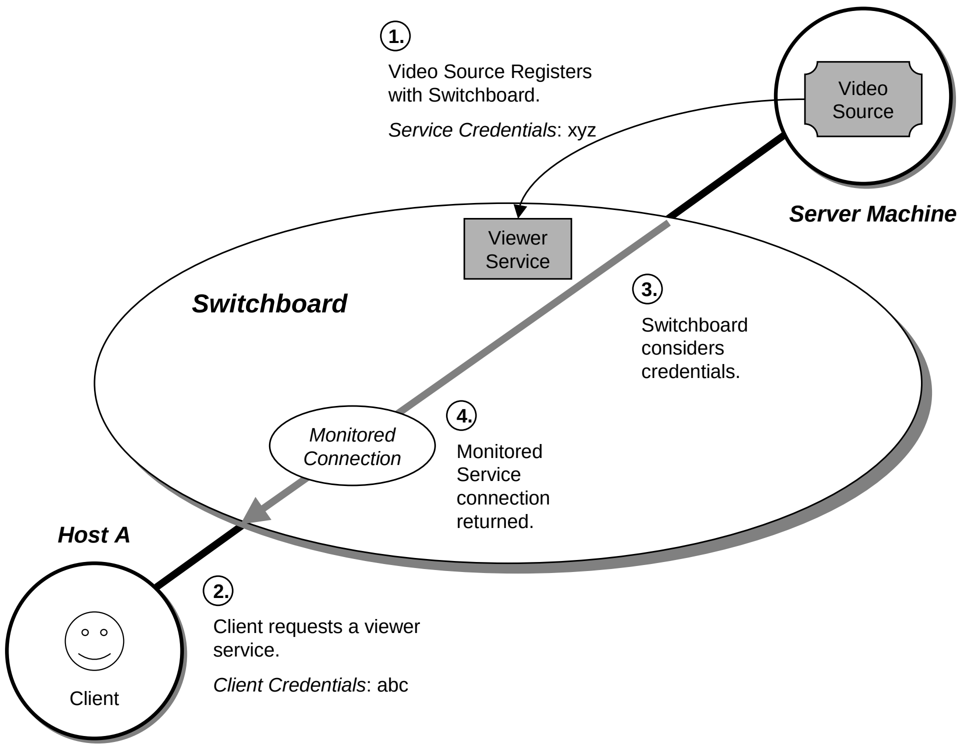
1.
Video Source Registers with Switchboard.
Service Credentials: xyz
Video Source
Server Machine
Viewer Service
Switchboard
3.
Switchboard considers credentials.
4.
Monitored Service connection returned.
Monitored Connection
Host A
2.
Client requests a viewer service.
Client Credentials: abc
Client

Figure 1: Four basic stages of a Switchboard connection. 1. A viewer service is registered with Switchboard with a set of credentials for authorization. 2. A client makes a request for a viewer service, providing its own set of credentials. 3. The Switchboard evaluates the credentials of both entities and their hosts. 4. If all parties are authorized, a monitored connection is established.

Consider extending this model to a system with hundreds of cameras and thousands of hosts administered by multiple trust authorities. Solutions appropriate for small centralized systems with few users (such as ACLs) are inappropriate for their significantly larger cousins. Switchboard scales with the problem by implementing trust management as an extensible component.

Finally, consider our example in a dynamic coalition environment. A coalition environment is characterized by numerous hosts under multiple administrative domains with changing trusts relationships. An example of such an environment is a military coalition, in which infantrymen from one country must cooperate with their counterparts from a different country for the duration of one mission. Clearly the trust management infrastructure of a communication substrate must be able to adapt to these dynamically changing relationships.

Coalitions necessitate multiple trust hierarchies, as each party in the coalition will introduce its own hierarchy and set of authorizations into the environment. Extending our example, imagine Host A and Host B are credentialed by different trust hierarchies as infantrymen from two different countries joined together for a single mission. For the duration of the mission, all soldiers involved must be able access reconnaissance images provided by Camera 1. As soon as the mission is over, all soldiers from Host B’s country are no longer permitted to view the video feed.

Coalition-relevant trust relationships must extend rather than replace already extant trust semantics within each organization, and must be able to be authorized or revoked by appropriate parties.

A trust management system that supports transitive trust relationships such as those that implement role-based access control are natural solutions to this problem. Our research has included the development of dRBAC [9], a distributed trust management system that supports remote credential discovery and the continuous monitoring of sustained trust relationships required for Switchboard connections.

3 Switchboard Architecture

3.1 The Switchboard API

Switchboard provides a clear API for developers so they may be able to effectively harness its functionality without delving into implementation details of socket connections and cryptographic libraries. The full API is listed in Figure 2.

Switchboard Fields

`ServiceName` - a name for a given service.

`Credentials` - provides proof of authorization of some entity (as evidenced by its public identity) to act in some role. `ClientCredentials` logically represent the credentials of a client, likewise `ServiceCredentials` represent the credentials of a service.

`MyIdentity` – Contains the public identity (i.e.the corresponding public key) of a given object and an object that can respond to an identity challenge.

`TrustMonitor` - an object which monitors and authorizes a trust relationship. `ClientMonitors` are `TrustMonitors` that monitor the credentials of a client, likewise `ServiceMonitors` monitor the credentials of a service. These objects contain the following functionality:

- **`initialize(connectionObject)`** - begin monitoring a connection's trust. Public identities of remote party and their credentials are available from the `ConnectionObject`. Returns true if connection is authorized.
- **`updateCredentials()`** - updates the credentials being evaluated by the `TrustMonitor`

`ServiceHandle` - a interface to a given service, which can be used to make calls on the service. Contains its own methods:

- **`accept()`** - returns a `ConnectionObject`.
- **`updateCredentials()`** - updates the credentials for this `ServiceHandle` and all open connections for this service.

`ConnectionObject` - an object representing a connection between two entities on a system. Relevant fields and methods include:

- **`ConnectedEntity`** - the public identity of the counterpart agent in a connection.
- **`ConnectedHost`** - the public identity of the host where the connection's counterpart executes.
- **`read()`** - functionality to read from the connection.
- **`write()`** - functionality to write from the connection.
- **`updateCredentials(CredentialObject)`** – transmits credential updates to the `TrustMonitor` for the correspondent. (Calls `updateCredentials(CredentialObject)` in the correspondent's `TrustMonitor`)
- **`start()`** - indicates preparedness to communicate with correspondent
- **`close()`** - terminates this connection.

`CipherSpec(interfaceSecurity)` - a object that enumerates acceptable ciphers for a particular interface.

Switchboard Methods

`registerService(ServiceName, ServerIdentity, ServerCredentials, ClientMonitorFactory, CipherSpec)` - Register an agent as providing a service. Returns a `ServiceHandle` on success, else raises an exception.

`connectService(ServiceName, ClientIdentity, ClientCredentials, ServiceMonitor)` - called by a client, returns a ConnectionObject on success, else it raises an exception.

Figure 2: The Switchboard API.

3.1.1 Switchboard Semantics

The following section describes the semantics of the Switchboard API.

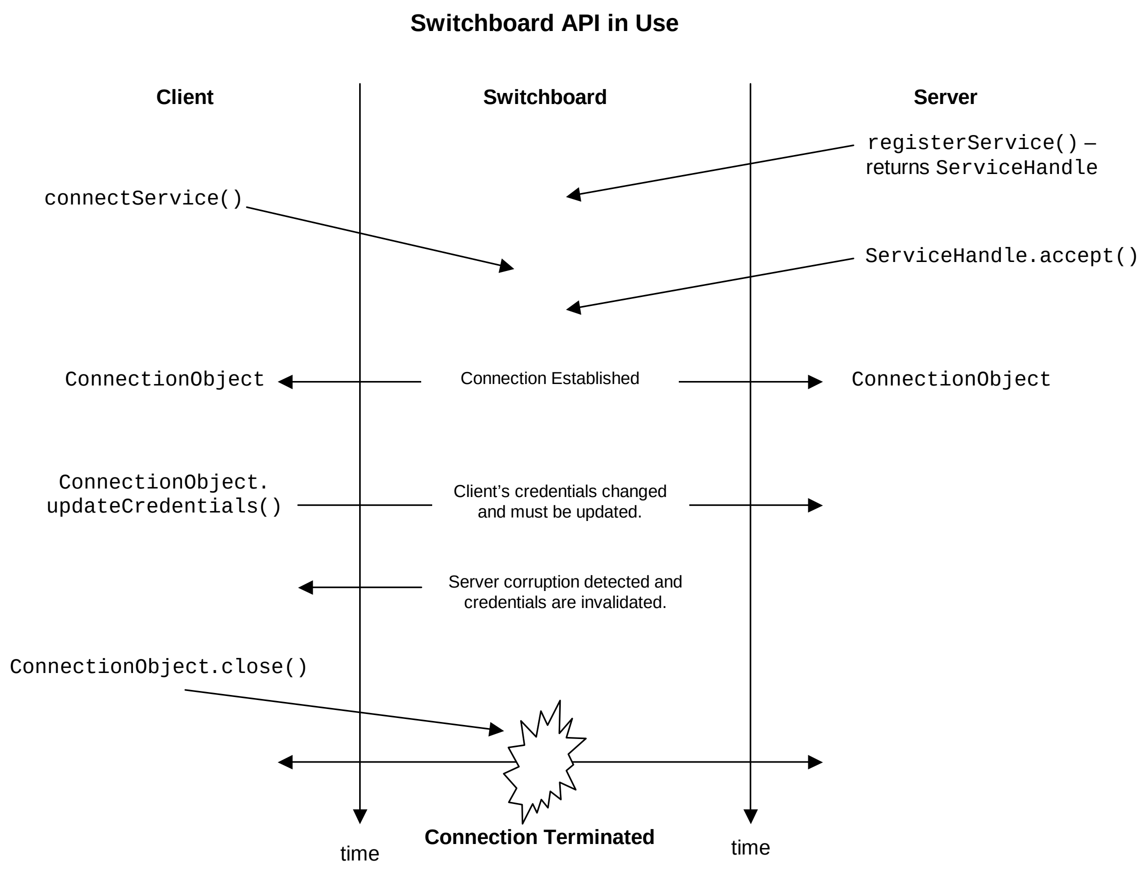

Figure 3: The Switchboard API used over time.

Registration of Service Providers An agent authorized to provide a service using Switchboard registers their intent with via its host's `Switchboard.RegisterService` method. This agent specifies:

- its public identity (with proof)
- an identifier for the service being provided (for discovery and connection matching)
- proof of its right to provide the service (as interpreted by a potential client)
- a factory for potential client trust evaluators

`RegisterService` returns a `ServiceHandle`. Following the model of Berkeley sockets, an agent awaiting a connection blocks on the `ServiceHandle.accept` method.

Client Connection Again taking our lead from the BSD socket interface, an agent requests a connection from a registered service via the `Switchboard.connect` method. The agent must provide its identity, the desired service's identifier, credentials demonstrating its authorization to access the service and its

`CipherSpec`. Switchboard will choose a registered service provider on some host using an appropriate cipher; the identities of both corresponding entities and hosts are verified, and both `TrustMonitors` are queried to determine if both agents should trust each other. If so, a `ConnectionObject` is returned to both sides.

Interaction Once a connection is established, `ConnectionObjects` have similar semantics to a TCP port. Data written using the `write` method of one `ConnectionObject` can be `read` from its correspondent's `ConnectionObject`. The trust relationship between the correspondents is continuously monitored on both sides by their respective `TrustMonitor` objects.

At any time, a connected agent can update credentials it has registered with its correspondent's `TrustMonitor` by calling its `ConnectionObject's updateCredentials(Object)` method. The object provided to this message will be communicated to its correspondent's `ConnectionObject`, and in turn passed as an argument to the corresponding `TrustMonitor.updateCredentials` method.

Connection Shutdown Either correspondent can terminate a connection by calling their respective connection's `close` method. In addition, while monitoring the authority of a correspondent's connection, a `TrustMonitor` object can also terminate connection using the `close` method.

3.2 Internal Architecture

Switchboard architecture consists of three major components. The Connection Module contains the code responsible for listening to incoming requests and maintaining connections between hosts. The Registry Module is a local store of activity, maintaining tables which track connections, services, and other needed entities. The Discovery Module provides functionality for querying the environment and locating available services.

The internal architecture also relies on independent libraries for authorization and security. Several ciphers are available, additional ones that conform to the JCE interfaces can be inserted. Likewise, authorization infrastructures other than the supplied ACL and dRBAC systems (such as KeyNote [1], SPKI [6]) can be provided.

3.2.1 Connection Module

- **Listener**

  The Switchboard Listener extends the semantics of Java RMI and RPC's portmap interface. This object listens for incoming connection requests on a reserved port number. When a new Listener connection is established, host identities are exchanged and authenticated using PKI. Like SSH [23], SSL [8]/TLS [5], a cipher is selected and appropriate keys are generated. The Listener then manufactures a Host Manager, which will assume responsibility for all communication within this host-pair using this particular cipher.

- **Host Manager** The Host Manager maintains a connection between a host pair using the particular (potentially null) cipher selected by the Listener. Like SSH, a Host Manager contains a single connection through which one or more multiplexed channels of communication can be piped, and its available mux channels are a resource available for future connections that require the same cipher. Each individual connection will be represented in each participating Switchboard by a `ConnectionObject`. If subsequent connections require a different cipher, an additional Host Manager will be established. Finally, Host Managers monitor connectivity by utilizing a replay attack resistant heartbeat mechanism on multiplexer channel zero.

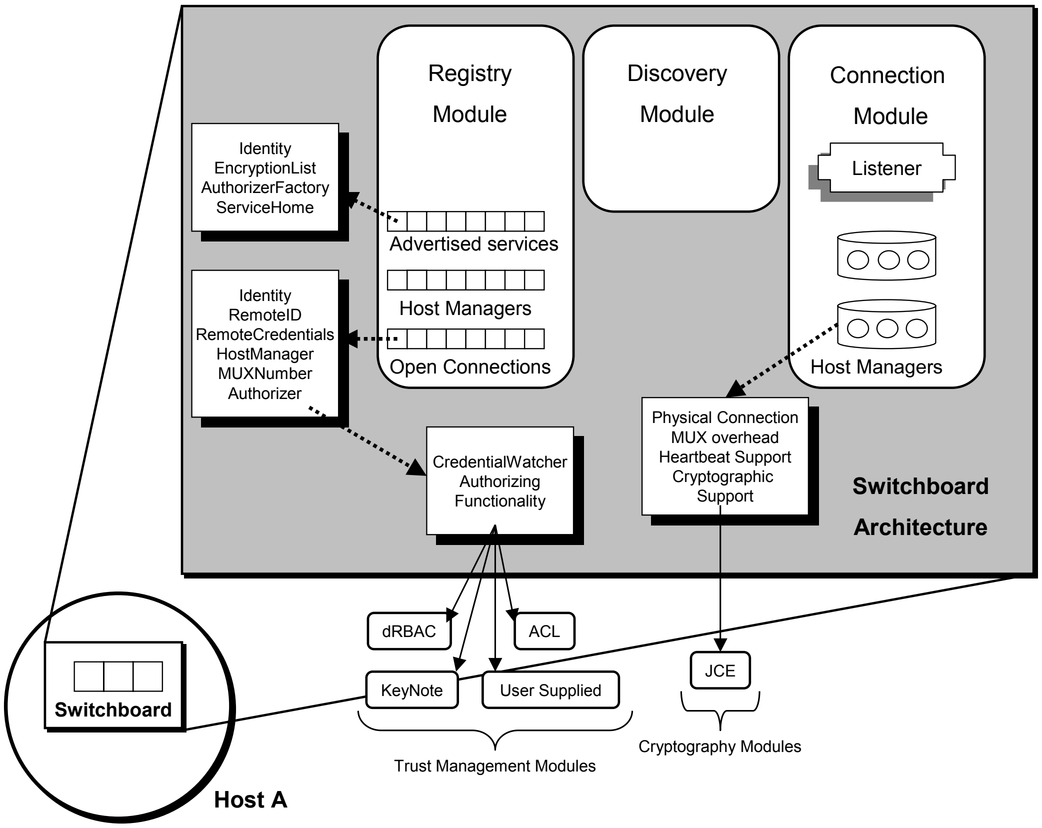

Figure 4: A graphical representation of the three main modules of the Switchboard abstraction: Connection, Discovery, and Registry. Extensible support libraries are depicted outside of the Switchboard box.

3.2.2 Registry Module

The Registry module is the central lookup structure of a Switchboard. It maintains tables of Advertised Services, Open Connections, and Host Managers, as described below.

- **Advertised Services**

  Every service that is installed on a host must register itself with the Advertised Services Registry. The Registry also advertises available services to other switchboards.

- **Open Connections**

  Open connections (as represented by `ConnectionObjects`) are tracked via the Switchboard's local registry. Each `ConnectionObject` maintains an `Identity` field, which holds the correspondent's identity. Likewise the `RemoteIdentity` and `RemoteCredential` fields maintain the `Identity` and `Credentials` of the entity to whom it is connected. Each connection is multiplexed through a Host Manager, so the `ConnectionObject` must reference the Host Manager handling its connection and corresponding mulitplexer channel.

The `ConnectionObject` also maintains an `Authorizer`, an object that maintains a `TrustMonitor` and specifies the trust management mechanism to be used with this connection. The `Authorizer` is responsible for establishing whether an entity must be authenticated using dRBAC, ACL's, or other user supplied package. These fields are initialized when a `ConnectionObject` is constructed.

- **Host Manager List**

  The Registry provides the Host Manager List to map host-cipher pairs to Host Managers so that unused mux channels from extant Host Managers can be made available for future connections.

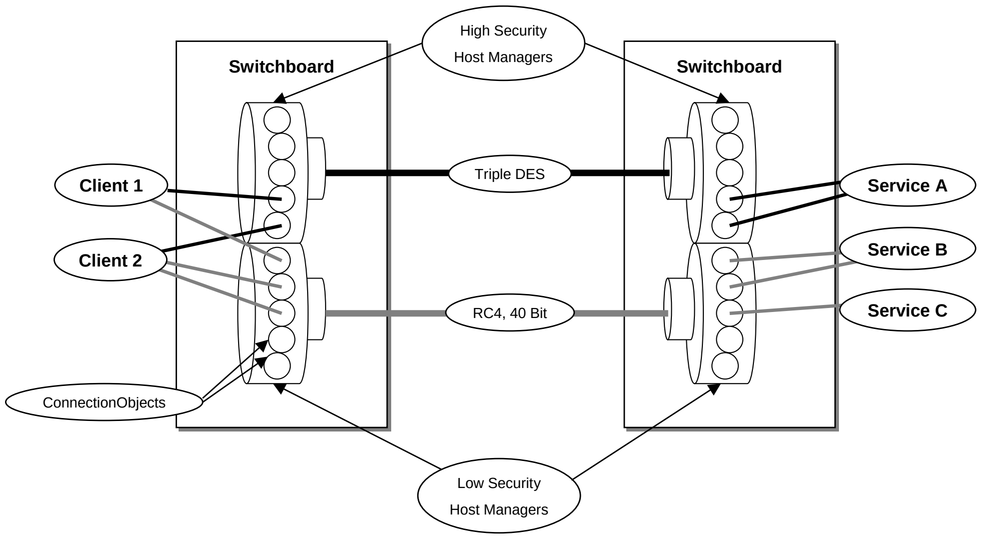

Figure 5: Host Managers multiplexing connections between hosts.

3.2.3 Discovery Module

The Discovery Module handles lookup of services in the environment. Our initial implementation, like JINI, floods the local network with advertisements discovery queries. If no provider is found, the service's home host is contacted (presently incorporated in a service identifier) for either a connection or referral. Our API is independent of the underlying service discovery mechanism, and we plan on extending our system to incorporate more ambitious schemes such as those of Plaxton et. al. [14].

3.2.4 Cryptographic Libraries

Each Switchboard, charged with the responsibility of maintaining secure connections, must have access to an adequate library of cryptographic libraries. Support for encryption, hashing, digital signatures, and message digests are essential for any secure environment. We include standard ciphers found in SSL [8] and SSH [23], including Triple-DES, DES, RC4, RC2, and RSA, as well as provide standard message authentication schemes SHA-1 and RipeMD-160.

3.2.5 Authorization Libraries

Switchboards are responsible for providing identity authentication and authorization. Trust management libraries supporting ACL and dRBAC algorithms are provided, and may be extended based on need.

4 Implementation Issues

This section presents our current Java-based implementation of Switchboard. We have developed the architecture with the Java 1.3 SDK and taken advantage of the language's extensive libraries. Object Oriented Design philosophy guided the construction of our architecture and proved useful for the creation of the extensible environment we were seeking. Key entities in our system, such as services and connections, are represented as objects. Java Remote Method Invocation (RMI) served as a basis for exploring inter-host connectivity requests, and the Java Cryptographic Extension (JCE) provided a framework for encryption, PKI, and data integrity.

4.1 Identity Authentication and Authorization

The nature of a distributed environment ensures that any given host will have to interact with other hosts or entities in the system, and may need to authenticate the identities of such objects.

Our solution to this problem is to assign every Switchboard compatible object in our system an identity, which is implemented as a public key. These identities are authenticated using standard PKI [13] challenge techniques of the JCE and JAAS.

Any trust management algorithm compatible with public keys as identities can be used with Switchboard such as PGP's web of trust, SPKI/SDSI [6, 16], PolicyMaker [2] or KeyNote [1]. We provide trust management libraries implementing access control lists (ACL) and role based authorization using our dRBAC [9] system.

4.2 Registering Switchboard Compatible Services

A service must implement certain features to be compatible with Switchboard. Each service must have a public key identity to be advertised in the Registry's list of available services. A service contains a `ServiceCredentials` object, a credential verifying that it may interact with specific clients. Each service must specify a set of acceptable encryption algorithms (its `CipherSpec`) and must provide a `ClientMonitor` object, which is used to monitor the authorization of requested connections.

4.3 Requesting and returning a service

When a user on a host desires a service, it queries its Switchboard, which either will locate the requested service in its Registry or will have to look elsewhere in the environment for the request. If the service is available locally, Switchboard must evaluate whether the access requirements of the desired resource are met by the access privileges of the client. It accomplishes authorization using the techniques described above, by analyzing the identity of the client and verifying it with the authentication system specified by the `ConnectionObject's` authorizing functionality. If access is granted, then a handle to the service is returned to the client. In our current implementation, service handles are primarily interfaces to data feeds and remote procedure calls.

We are investigating extending this model to implement secure object delivery including support for on-demand delivery of code.

If the desired service does not reside on the local machine, work is then passed to the Discovery Module. If no known provider is found among locally-accessible hosts, the request is forwarded to the service's home, which will either accept the request or suggest another host capable of providing the required service.

Once a Switchboard is located on which the desired service is registered, it connects to the host (if not connected already), and authorizes the client's ability to obtain the resource. If access is granted, then the client may utilize the service.

4.4 Establishing a connection

Initial Switchboard connection requests are first directed to the server host's Listener on the designated port. This commences an inter-Switchboard handshake protocol similar to that of SSH and SSL, in which host identities are verified via public key cryptography and a suitable encryption algorithm is negotiated between the host pair.

Once an encrypted channel has been established, each side of the host pair manufactures a Host Manager object, which will provide control for the physical connection between two hosts and maintain multiplexers for individual service connections. A Host Manager will provide a static security level, established by the requirements of the connection request that necessitated the Host Manager's instantiation.

The case may be that a client issues a request for a service advertised by an already connected host. With an already established connection, the client examines what security requirements are necessary to interface with the service and selects an appropriate Host Manager to handle the request. If no Host Manager with an acceptable cipher exists, Switchboard must manufacture one and establish a counterpart on the corresponding host.

At this point the local Switchboard will choose an unreserved, unallocated multiplex number in the appropriate Host Manager and send the request for the desired service to the remote Switchboard. The request is signed by the client private key for verification, and includes the client's identity, the service's identity and the client's credentials.

The server analyzes the authenticity and validity of the request via the client's signature, and ensures that the client has required access rights and can support encryption mandated by the server side. If the server accepts the client's request, the server returns its idenity and authorizing credentials (stating that the server may provide the service) in a message signed with the server's private key.

The client, upon successfully authenticating the server and validating its response, may connect to the service. If any of the authentication or authorization steps fail, the service connection is denied.

4.4.1 Connection Failures

A connection can fail due to an unsatisfied trust requirement or disconnection of a Host Manager from its remote correspondent. In either case, all affected connections are immediately closed.

Applications utilizing Switchboard can detect disconnects and can decide to initiate another appropriate connection. Switchboard, however, provides no reconnect functionality on its own.

4.5 Monitoring connections

Each `ConnectionObject`, charged with the responsibility of maintaining a specific connection, includes an `Authorizer` which keeps watch over the credentials of its counterpart on the remote host (this functionality in turn is exercised by a `TrustMonitor`). The `Authorizer` also has a prescribed certifying functionality, such as dRBAC or an ACL library, which can determine the validity of the credentials it monitors. Should the credentials of the connected entity change in such a way as to revoke access privileges, the connection is terminated.

4.5.1 dRBAC and Monitoring of Connections

We have developed a distributed role-based access control system dRBAC [9] that implements a novel mechanism to monitor sustained trust relationships based on revocable credentials. The dRBAC `TrustMonitor` plug-in accepts credential updates and will inform the switchboard when the required trust relationship is no longer valid.

5 Evaluation of Switchboard Abstraction/Architecture

The development of Switchboard was motivated by needs of the DisCo effort of the NYU Distributed Sanctuaries project. DisCo is developing application-neutral middleware infrastructure to support secure decomposable service delivery in semi-trusted coalition environments.

Applications developed for DisCo require infrastructure for establishing secure communication channels between agents executing on multiple hosts administered by disjoint trust authorities. DisCo utilizes Switchboard's facilities for secure object delivery, RPC, and code distribution.

To evaluate Switchboard, we compare both the "code bloat" and increased execution and communication overhead involved in Switchboard functionality to the overhead involved in performing the same functionality independently. Our conclusion is that Switchboard incurs negligible additional overhead beyond other available schemes (such as SSL libraries) and often benefits from key reuse. In addition, the Switchboard API is no more difficult to use than the available alternative mechanisms that offer substantially reduced functionality.

5.1 Discovery

Discovering the whereabouts of a needed replicated resource in a distributed environment requires a system-wide locator utility. Switchboard enabled hosts in an environment provide this functionality automatically via their Discovery and Registry Modules. Clients requiring inter-host communication with discovery, if not using our proposed architecture, must procure some independent means of maintaining system-wide resource tracking, which build on relatively low-level devices such as DNS.

5.2 Port Usage

As previously discussed, initial host connectivity is established using a designated port number. All service requests can be made based on the connections that are established after this first connection is made. When new services are introduced into a host's environment, there is no need beyond registering with the local Switchboard to notify the host system.

In contrast, a normal environment would require a system administrator to register a port number with a given service, as is the case with portmap and traditional TCP/IP ports. This port number must then be made known to all clients who might desire this service.

5.3 Comparison of Opening New Connections

A host pair setting up an unaided connection typically relies on the BSD socket system calls listen(), accept(), and connect(). Switchboard connection functions map directly to traditional system connection calls - establishing the connection in terms of lines of code is equal. Switchboard does incur the extra overhead of establishing Host Managers and their included multiplexer. However the cost of establishing a cipher in Switchboard is equivalent to that of SSL. Furthermore, the cost of establishing a Host Manager pair and related cipher can be amortized over the many connections that may exist over their lifetimes. The cost of authentication and authorization for a single connection is equivalent for Switchboard and other systems.

5.4 Cost of Multiplexing of Connections

Multiplexed connections can result in a reduction of the number of signaling messages passed between hosts when compared to approaches that utilize multiple TCP ports. A single acknowledgment or heartbeat message applies to all connections sharing the same mux. Minimum message size requirements of many networks and ciphers reduce the efficiency of small messages; a multiplexed channel can aggregate small messages from multiple senders into a single larger message, resulting in higher efficiency.

5.5 Comparison of Credentialed Connections

Credentialing is most commonly seen in secure applications such as SSH and SSL, both of which maintain access control lists and private key signatures to ensure identities. A client needing to incorporate a credentialing scheme into network communication would most likely sandwich SSL between the application and network layers.

5.6 Advantages of Switchboard Abstraction

The primary advantage of the Switchboard abstraction is that much functionality is contained within a relatively small API with similar performance to the disparate competing technologies. The switchboard provides a powerful and easy to use interface that would otherwise require considerable development time.

6 Related Work

The switchboard architecture combines several mechanisms that are presently available only as partially compatible sub-systems into a single, extensible API. Our work extends the following developed technologies.

Secure Connections Many standard and open-source encryption libraries are available for development purposes. SSL [8]/TLS [5] authentication protocols provide a widely used toolset for establishing authenticated connections where connection authorization decisions are made using access control lists. IPSEC is available for lower-level, network-layer security.

Switchboard features an extensible architecture which enables the developer to utilize the most appropriate ciphers for secure connections.

Monitoring of Connections Switchboard enables dynamic tracking of access rights by monitoring the connections of identities in the environment. Monitored connections are rooted in Publish/Subscribe systems, such as IBM's Gryphon project [19].

Authentication Various authentication schemes exists and are implemented in a variety of technologies. Java Authentication and Authorization Service (JAAS) is a Java package which allows resources to authenticate and enforce access control among users. Considerable work has recently been done in the area of trust management (RBAC) [17, 18], which allows for more dynamic trust relationships than ACLs and accommodates our decentralized trust needs (see SPKI/SDSI [6, 16], KeyNote [1], PolicyMaker [2], and Li and Winsborough's work on the RT0 trust model [11]). Other authentication models are based on the endorsements of certificate authorities, as in X509 certificates.

Switchboard allows for pluggable authentication modules, permitting the developer to select the right authentication scheme for the desired application.

Discovery Much recent work has taken place in the field of resource discovery, notably by Sun Microsystem's Jini technology [22] and peer-to-peer lookup schemes such as Chord [4] and work by Plaxton et. al. [14]. We will be looking at these as well as more established discovery mechanisms, such as DNS and the Java Naming and Directory Interface (JNDI) for our work with Switchboard's Discovery Module.

Unified Technologies

- **Peer-to-Peer Infrastructures** Peer-to-peer infrastructures such as JXTA [15] provide a set of generalized protocols including dynamic discovery that allow any connected device in a network to communicate. JXTA is in its early stages of development and presently lacks sufficient functionality for what we are trying to accomplish. Most interesting for Switchboard are JXTA Pipes, which are generalized asynchronous, unidirectional channels for inter-host communication. JXTA pipes can transfer binary code, data strings, Java technology-based objects, and/or applets. JXTA pipes resemble Switchboard connections, but the underlying trust model [3] is not extensible. Furthermore, there is no support for monitoring connections, a significant drawback in developing dynamically changing trust relationships in coalition environments.

- **SSH** SSH [23], combines public key infrastructure, session key sharing, and ACLs to provide a user interface for secure communications. Like Switchboard, SSH incorporates a connection multiplexer utilized to forward multiple TCP connections between ports on different hosts through a single encrypted TCP connection. The most common use of this feature is to forward incoming X11 window connections through outgoing-only firewalls. While SSH extends standard Xauth [7] security measures (through copying of environment variables), additional inter-host port mappings can be established that provide alternate connection paths between hosts. However, no mechanism is provided to authenticate trust relationships between processes that connect via these forwarded ports.

- **Component Frameworks** Component frameworks such as J2EE [20], CORBA [10], and .NET [12] provide infrastructures that combine support for remote secure communication, authentication and authorization, and naming and discovery. To consider J2EE as an example, it builds on other Java technologies such as RMI, JAAS [21], and JNDI to provide a "container" environment for server applications. In addition to allowing services to be written in a modular fashion, the containers offload the process of managing client-server connectivity across untrusted environments in much the same way as the Switchboard abstraction. However, unlike the relatively stylized interfaces required of J2EE components, Switchboard defines a relatively simple and familiar API, enabling its use with low cost by a larger number of applications. More importantly, a fundamental part of the Switchboard abstraction is its support for connection monitoring. The latter is not explicitly supported in any of the component infrastructures.

7 Conclusion

In creating Switchboard, we have established an abstraction which decouples application development from the high overhead cost of programming complicated inter-host communication protocols in distributed computation environments. Developers are assisted by a consolidated API including trust management, service discovery and delivery, and secure communication provided by Switchboard, and direct more effort towards their ultimate project goals.

The Switchboard architecture is a highly modular system containing separate modules for connectivity, discovery, and registration. Existing libraries for trust management and cryptography may be added to Switchboards extensible structure for more flexible usage.

Our creation of Switchboard is a development of our Disco project, in which we are exploring service distribution in an environment with dynamic trust relationships. The distributed nature of our efforts has steered us in the direction of formulating a system-wide communication substrate.

8 Acknowledgements

Oliver Kennedy, Vladimir Vanyukov, and Jordan Applebaum implemented several prototypes of switchboard components.

This research was sponsored by DARPA agreements F30602-99-1-0157, N66001-00-1-8920, and N66001-01-1-8929; by NSF grants CAREER:CCR-9876128 and CCR-9988176; and Microsoft. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright annotation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as representing the official policies or endorsements, either expressed or implied, of DARPA, Rome Labs, SPAWAR SYSCEN, or the U.S. Government.

References

[1] Matt Blaze, Joan Feigenbaum, and Angelos D. Keromytis. KeyNote: Trust management for public-key infrastructures. In *Proceedings of the 1998 Security Protocols International Workshop, Springer LNCS vol. 1550*, pages 59–63, 1998.

[2] Matt Blaze, Joan Feigenbaum, and Jack Lacy. Decentralized trust management. In *Proceedings of the IEEE Conference on Privacy and Security*, 1996.

[3] Rita Chen and William Yeager. Poblano: A Distributed Trust Model for Peer-to-Peer Networks. Sun Microsystems, Inc. White Paper, Available at `http://www.jxta.org/project/www/docs/trust.pdf`, 2001.

[4] Frank Dabek, Emma Brunskill, M. Frans Kaashoek, David Karger, Robert Morris, Ion Stoica, and Hari Balakrishnan. Building Peer-to-Peer Systems with Chord, a Distributed Lookup Service. In *Proceedings of ACM SIGCOMM*, 2001.

[5] T. Dierks and C. Allen. The TLS Protocol, Version 1.0. IETF Request for Comments 2246, Available at `http://www.ietf.org/rfc/rfc2246.txt`, 1999.

[6] Carl M. Ellison, Bill Frantz, Butler Lampson, Ron Rivest, Brian M. Thomas, and Tatu Ylonen. SPKI Certificate Theory. IETF Request for Comments 2693, Available at `http://www.ietf.org/rfc/rfc2693.txt`, 1998.

[7] Jeremy Epstein and Jeffrey Picciotto. Trusting X: Issues in Building Trusted X Window Systems -or- What's Not Trusted About X? In *Proceedings of the 14th Annual National Security Conference*, 1991.

[8] A. Freier, P. Karlton, and P. Kocher. The SSL Protocol, Version 3.0. Internet Draft <draft-freier-ssl-version3-02.txt>, Available at `http://www.netscape.com/eng/ssl3`, 1996.

[9] Eric Freudenthal, Tracy Pesin, Lawrence Port, Edward Keenan, and Vijay Karamcheti. dRBAC: Distributed Role-Based Access Control for Dynamic Coalition Environments (TR2001-819). Technical report, Department of Computer Science, New York University, 2001.

[10] Object Management Group. Common Object Request Broker Architecture (CORBA) Specification Version 2.5. Available at `http://www.omg.org/technology/documents/formal/corba_iiop.htm`, 2001.

[11] Ninghui Li, William H. Winsborough, and John C. Mitchell. Distributed credential chain discovery in trust management. In *Proceedings of the 8th ACM Conference on Computer and Communications Security*, 2001.

[12] Microsoft Corporation. Microsoft .NET Framework SDK Beta 2. Available at `http://www.microsoft.com/net`, 2001.

[13] R. Perlman. An overview of PKI trust models. *IEEE Network*, 13(6):38–43, 1999.

[14] C. Greg Plaxton, Rajmohan Rajaraman, and Andrea W. Richa. Accessing nearby copies of replicated objects in a distributed environment. In *ACM Symposium on Parallel Algorithms and Architectures*, pages 311–320, 1997.

[15] Project JXTA. JXTA Version 1.0 Protocols Specification. Available at `http://spec.jxta.org`, 2001.

[16] Ronald L. Rivest and Butler Lampson. SDSI – A simple distributed security infrastructure. In *Proceedings of CRYPTO'96*, 1996.

[17] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman. Role-based access control: A multi-dimensional view. In *10th Annual Computer Security Applications Conference,* pages 54–62, 1994.

[18] Ravi S. Sandhu, Edward J. Coyne, Hal L. Feinstein, and Charles E. Youman. Role-based access control models. *IEEE Computer*, 20(2):38–47, 1996.

[19] Robert Strom, Gurudth Banovar, Tushar Chandra, Marc Kaplan, Kevan Miller, Bodhi Mukherji, Daniel Sturman, and Michael Ward. Gryphon: An Information Flow Based Approach to Message Brokering. In *Proceedings of the International Symposium on Software Reliability Engineering*, 1998.

[20] Sun Microsystems, Inc. Java$^{\mathrm{TM}}$ 2 Platform, Enterprise Edition Specification, Version 1.3. Available at `http://java.sun.com/j2ee/docs.html`, July 2001.

[21] Sun Microsystems, Inc. Java$^{\mathrm{TM}}$ Authentication and Authorization Service, Version 1.0 (JAAS Specification). Available at `http://java.sun.com/products/jaas`, 2001.

[22] Jim Waldo. The Jini architecture for network-centric computing. *Communications of the ACM*, 42(7):76–82, 1999.

[23] T. Ylonen, T. Kivinen, M. Saarinen, T. Rinne, and S. Lehtinen. SSH Protocol Architecture. Internet Draft <draft-ietf-secsh-architecture-09.txt>, Available at `http://www.ssh.com/tech`, 2001.