Carnegie Mellon University
Information Networking Institute

THESIS

SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

Master of Science in Information Networking

A Service Brokerage Deployment Architecture

Presented by Carlos Olguin

Accepted by the Information Networking Institute.

Thesis Advisor	Peter Steenkiste	_____ ___/___/___
Reader	Srini Seshan	_____ ___/___/___
INI Academic Advisor	Richard Stern	_____ ___/___/___

Carnegie Mellon University
Information Networking Institute

A Service Brokerage Deployment Architecture

A THESIS SUBMITTED TO THE INFORMATION NETWORKING INSTITUTE IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE

Master of Science in Information Networking

By

Carlos Olguin

Pittsburgh, Pennsylvania
April, 2001

ABSTRACT

As more and different services appear over the Internet, there is a need to have a brokerage architecture that abstracts complexity from the user and is scalable enough to be internet-wide deployed.

We present an architecture for the deployment of service brokers over the Internet. We have done this by merging SLP and AS1 and adding our own design elements. The design considerations for the work presented were (1) high variability of usage, (2) high diversity of services and resources, (3) no single point of failure, and (4) network awareness. In addition, we decoupled the notion of agent, service, and resource in three separate logical components and we illustrate how SLP and AS1 behave fundamentally different in their perception of them.

In the proposed architecture, a service broker optimizes resource consumption by deciding to reproduce or aggregate based on the demand of their services and the consequent load they experience. A service broker can also "reincarnate" the functionality of another service broker who ceased to function.

We implemented Joxer, a proof of concept prototype. To measure its effectiveness, we defined a benefit/cost metric called Effective Resource Usage Metric (ERUM). We defined ERUM as the inverse of bandwidth utilization times average CPU load consumed by active Service Brokers.

Introduction

Along with the explosive growth of the Internet, there has been an explosion of architectures intended to give some kind of coherence to such chaotic growth. Recently, one particular area of study has gained more interest within the research and industry communities, service brokerage (SB). Service brokerage is a mechanism that allows a user to request and access a specific service. Service brokerage can be done in different contexts. It may be as simple as redirecting a client to the first service that matches a request or it may involve more intelligent mechanisms in order to decide which service to choose.

Another important aspect of a SB is the set of discovery algorithms necessary for building and updating a repository of services currently available in a network. Many solutions have been proposed to improve such algorithms [1,2,4]. Closely related to this issue but much less studied is the question of how service brokerage mechanisms, seen as another type of service, could be deployed in a scalable way, adapting to the demand they experience. That is, extend existing service brokerage architectures to include the service broker deployment.

As more and different services appear over the Internet, there is a need to have a brokerage architecture that abstracts complexity from the user and is scalable enough to be internet-wide deployed. The solution should meet the following general criteria:

- Scalability
- Robustness
- Security

1.1 Objectives

This document presents an architecture and proof of concept implementation for the deployment of service brokers over the Internet. As we will see in Chapter 4, we will expand these general criteria to a more specific set of measures in the context of service brokerage.

We believe that an important factor for a successful deployment of service brokers is the degree of network awareness available in the system. As part of the architecture that we propose in Chapter 5, we describe how service brokers decide to reproduce or aggregate based on the demand of their services and the consequent load they experience. We believe that such decisions can be more accurately taken by considering in addition,

conditions external to the service broker, such as link speed, bottlenecks, latency, and hop proximity between users and services.

Finally, this work has been done in adherence to existing frameworks that, from our perspective, provide partially a deployment architecture for service brokers. In particular, we will use two existing architectures, Service Location Protocol (SLP) [1] and Active Services (AS1) [3]. We argue that each of them separately provides interesting characteristics for a service brokerage architecture over the Internet. Thus, we will use them as a starting point for our own design.

1.2 Limitations

This document does not address the decision-making processes for identifying the services other than the service broker. Also, although we consider security a critical factor for the effective deployment of any service brokerage architecture over the Internet, we focus in this document only on scalability and robustness. We believe that our proposal can be complementary to the security components of existing service discovery efforts such as [13] and therefore take advantage of them as they grow.

1.3 Remainder of this Document

In Chapter 2 we outline relevant aspects of SLP with respect to service brokerage, including its implementation over wide area networks. Similarly, in Chapter 3 we outline AS1. Chapter 4 compares these two frameworks. In Chapter 5 we propose our own solution. In Chapter 6 we present experimentation results for a subset of the SB problem space. Chapter 7 presents a final summary and future work.

2 Service Location Protocol

SLP provides a framework for the discovery and selection of network services. The key idea behind this protocol is to allow a device to transparently make use of network services with little or no static configuration. This is done by using a lightweight session approach in a network under "cooperative administrative control". The use of technologies such as multicast IP prevents SLP from being scalable over the entire Internet. As we will see later on, there are other reasons why SLP, in its current state, cannot be deployed over the Internet.

2.1 SLP Basic Mechanisms

We first define the basic elements that compose SLP:

- User Agents (UA) who act on behalf of a client application to contact a service.
- Service Agents (SA) who act on behalf of one or more services to advertise them.
- Directory Agents (DA) who collect information about services advertised.

For an example of the basic elements of SLP, see Figure 2-1 below.

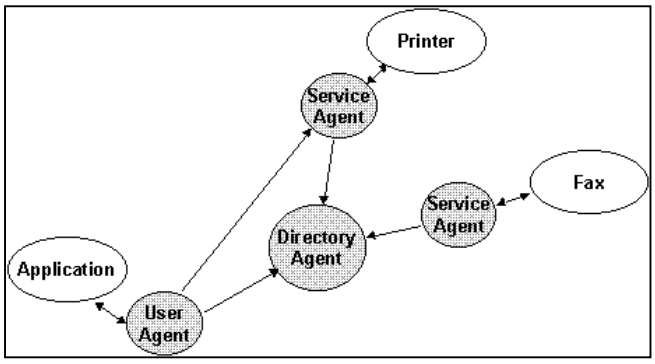


Figure 2-1. SLP Elements

Within this context, a UA issues a service request specifying the characteristics of the service that the client application requests. There are two methods to issue such request, depending on whether there is a known DA or not. If there is a known DA that can service the request, unicast is used. The UA will receive a service reply from the DA specifying the location of all the services in the network that satisfy the request. If there is no known DA, multicast is used to contact all possible SAs. Each SA that matches the service request will individually issue a service reply containing only its own information.

Moreover, whenever there is a DA, SAs will periodically register (refresh) their services with it. DAs can be assigned to a SA statically or dynamically (DHCP). More interestingly, SAs and UAs can discover the existence of a DA in at least two ways. First, when SAs and UAs startup, they will multicast a directory service request. Second, the DA will send an unsolicited advertisement infrequently. In either case UAs and SAs will receive an advertisement from the DA.

2.2 Wide Area SLP

Wide Area SLP extends [9, 10] SLP version 1 beyond the local area network space by adding the following components:

- Broker Agents who selectively collect information about services offered in other SLP domains (SLPD)

- Advertisements Agents who selectively advertise information about services offered within a specific SLPDs

The exchange of information between different SLPD's is based solely in using multicast techniques. In addition, there is no hierarchy within the domain distribution, which potentially means a large bandwidth overhead and emergence of bottlenecks in information transfer. These two factors limit Wide Area SLP from being used Internet-wide.

2.3 SLP version 2

SLP v2 redefines the use of scopes to very much resemble the Wide Area SLP Domains mentioned in Section 2.2. More formally, a scope groups together services under some specific criteria (administrative, geographic, service type, etc.). In SLPv2, SAs and DAs are always assigned to a scope. An agent, in general, communicates only with other agents sharing the same scope. These and other changes make SLPv2 more scalable. In addition, security improvements made in this version make the deployment in an open network more feasible. However, the reasons exposed in Section 2.2 hold true in SLPv2 preventing its Internet-wide applicability.

3 AS1

Similar to how active networks inject user-defined computation into the network, allowing certain type of applications to improve their performance, AS1 targets a subset of problems to be solved by active networks by restricting its design space to the application layer.[1]

Furthermore, in a more aggressive way than SLP, AS1 is also deployed and maintained by using a lightweight-session approach, where ideally all announce-listen communications are based on multicast IP, and state is preserved by using soft state tables in every entity. As explained in the next section, AS1 presents what we call an "implicit service brokerage". In this way, by using a highly distributed and loosely coupled approach, AS1 achieves great robustness against failures with the price of a large bandwidth overhead and limited intelligence for choosing the best service for a specific request.

3.1 AS1 Basic Mechanisms

AS1's most important components can be classified in the following way:

- *Clients,* who similarly to UAs in SLP, announce their service request to a "well-know point-of-contact" (normally a multicast address).

- *Host Managers* (HM) who rendezvous with clients and service their requests by deploying the appropriate servents.

- *Servents* who are launched by a *host manager* on behalf of the *client* who requested their services.

When a client makes a service request, a pool of HMs will respond to this request using a technique called multicast damping. That is, every HM on their own will setup a random time-out period to wait until it deploys a servent to service the request. At the same time, every HM will also listen for announcements of a servent deployed by some other HM who timed out earlier. When this happens, the HM will avoid deploying any servent at all, and instead register in a soft state table that this request has been serviced already. In this way, every future announcement made by the client will be ignored as long as the servent keeps announcing its existence.

[1] The reason exposed by the authors of AS1 [3] behind this decision is to preserve compatibility and to facilitate its incremental deployment.

This situation is nonexistent in SLP because even in the case of not having a DA, no matter how many service replies a UA may receive, these are only notifications of the existence of a service, and not the provision of the service itself. Therefore, in SLP the final decision for choosing a service remains always in the user side, avoiding a possible contention between more than one SAs contending to provide the same service.

Figure 3.1 shows a multimedia transcoding application of AS1, the Media Gateway (MeGa). In this simplified MeGa example, a servent is casted as a video gateway. This gateway performs transcoding services for a video client that, due to its own limitations (it cannot support a specific video format) or because of network limitations (link bandwidth too slow), requires some kind of manipulation for the video streams it wants to receive.

In addition, under AS1 a pool of HMs can achieve some degree of load balance by having each HM accept or ignore a service request, considering also its own current CPU load. This mechanism provides an implicit load balancing functionality[2]. In the case of SLP such functionality does not exist. However, as we will see in Chapter 5, it could be easily integrated on top of a DA, given its coordination role.

Another important element within the AS1 architecture is the soft state gateway (SSG) providing compatibility for networks with no multicast support. SSG joins a multicast session on behalf of a client with no multicast support. In this way, all communication between SSG and client is unicast while the SSG will convert to multicast when necessary and forward the client packets. As a reminder for the case of SLP, UAs and SAs initially find a DA by one of three possible ways: static pre-configuration, DHCP, or multicast. After that, unicast communication will be used. In AS1, multicast is always used as there is no specific centralizing entity. In either case, when there is no multicast support, a static configuration (or even DHCP in the case of SLP) must be used, which limits its scalability.

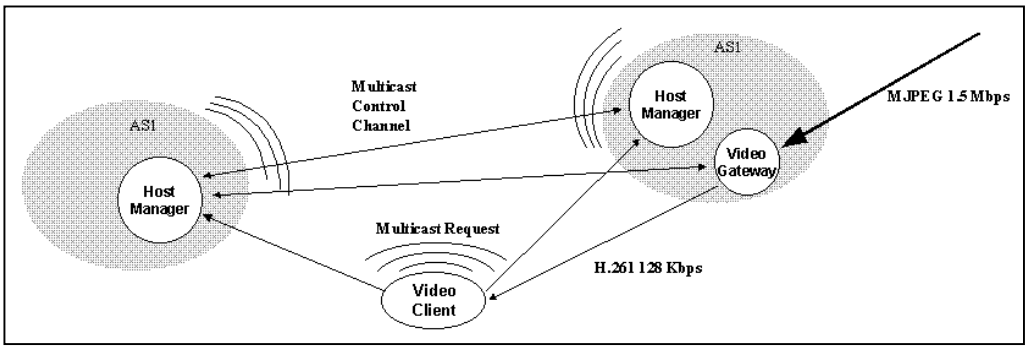


Figure 3-1 Mega Service in AS1

[2] Unfortunately, at least in mash version 5.01b this functionality was non-existent or disabled.

Finally, AS1 is an implicit service brokerage mechanism because brokerage of services is achieved without the need of an explicit broker – each member of a pool of host managers decides internally what request it should service.

3.2 Agent/ Service/ Resource Decoupling

Similar to the way that java applets are launched on a virtual machine to perform a specific action,[3] AS1 provides a metaphor where the type of service provided to a user is decoupled from the actual resource providing it. That is, the resource can be running multiple types of services as long as it can access and execute the code for such services. For instance, a pc managed by an HM running different transcoding servents. This is fundamentally different from SLP where there is no notion of an HM that could potentially launch SAs in reply to UA's demand. Instead, as services represented under SLP are usually stationary – such as printers and faxes – and their creation on demand is not evident (at least not today!), the type of services that a SA represents is always static. On the other hand, in AS1 there is no distinction between the service agent and the service itself (not the resource). That is, once an HM launches a *servent* to perform a service, the *servent* also becomes its own agent. Figure 3-2 outlines these relationships in SLP and AS1, respectively.

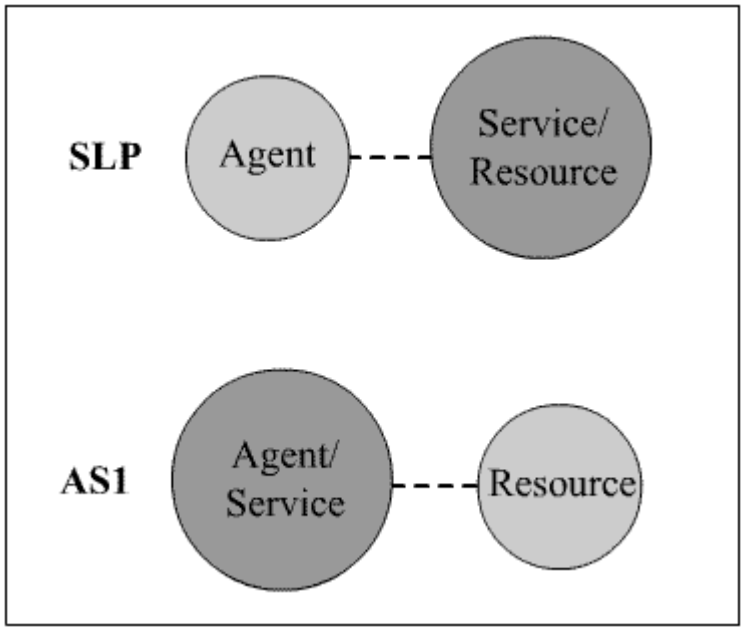


Figure 3-2 Agent/Service/Resource Decoupling in SLP and AS1

[3] A more specific example, which is specially related to the design domain of AS1, could be the use of delegates within CMU's Network Resource Management project, Darwin.

4 Extending the Design Criteria for a Service Brokerage Architecture

In our goal of finding the best deployment architecture for service brokerage we now turn to the task of taking the AS1 and SLP frameworks and examine them in terms of how well each approach meets the criteria of scalability and robustness. Furthermore, we decompose scalability and robustness into the following design considerations:

- High variability of usage
- High diversity of services and resources
- No single point of failure
- Network awareness

4.1 High Variability of Usage

Any service brokerage architecture intended to be used Internet-wide should obviously be able to service a very large number of users. However, provisioning for a large demand of users should not include wasting resources when demand is low, or manually configuring the decrease or increase of such resources available.

As described in Chapter 3, AS1 provides a highly distributed and loosely coupled architecture. However, compared to a centralized approach there is an overhead penalty that can be significantly large as the number of entities increase or as they become more "separated" from each other. Figure 4.1 illustrates the problem where, every node multicasts its information to every other node, resulting in an increase in bandwidth consumption that is in theory *n* times larger than in the centralized approach, where *n* is the number of participants in the session (HMs, servents, clients).

We must mention that the evaluation of these sub-criteria is subjective and qualitative. Because SLP and AS1 are quite different in their approaches, it would not be possible or useful to carry out a full quantitative comparison. Thus, we used these criteria to give our analysis a common focus implementing specific key testing scenarios, rather than to do a complete side-by-side objective comparison.

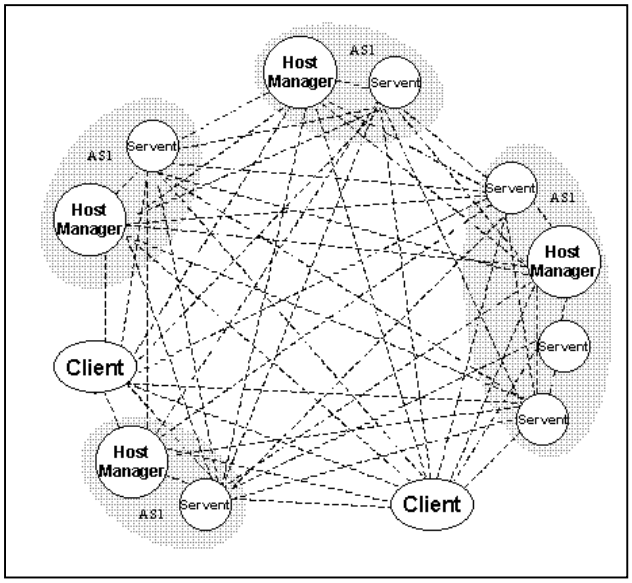


Figure 4-1 Communication Overhead in AS1

Similar to other protocols such as SAP [11] and RTP [12], AS1 regulates the control bandwidth by causing the refreshment period to increase as the number of entities increases. For instance, if the main control channel is set to 20 kbps, and there is a total of 10 host managers and servents, each of these entities will transmit their control information at 2Kbps. The danger of this approach is that if the refresh is too infrequent, the information used by HMs and servents could be stale and provoke undesired behaviors (e.g. having a HM to launch a duplicate gateway because it didn't know that one was already lunched). This imposes an inherent limit in the number of session members (around 300 according to AS1 authors).

While SLP can be configured to work in a similar way to AS1 using solely multicast communication (no DA), we have already seen that this could only be feasible within small service environments. Larger environments under this many-to-many control communication become less and less efficient. This is where the notion of a DA becomes more useful, where service requests and service advertisements are efficiently centralized through a single entity. Furthermore, by introducing the notion of scopes, multiples DA can exist (one for each scope), scaling even more. However, another problem arises as the service environment keeps growing and scopes multiply without any structure, DAs may become soon a bottleneck of information. Other mechanisms are needed to scale the number of users to Internet proportions.

4.2 High Diversity of Services and Resources

As the number of services and resources (service sites) increase, it is impractical to think that a single service broker will be able to maintain state over all the services available over the Internet. A service brokerage architecture should be able to rearrange itself according to some kind of organization (e.g. type of service, region). This rearrangement could be done in a hierarchical fashion.

In the case of AS1 there is no explicit reference in the literature to hierarchical organization of HMs. The closest thing we found is a reference to service composition. Although there is no specific design description, the hierarchical chain of servents referred to could possibly be used for a hierarchical setup of services. We can also imagine having different pools of HMs specialized according to some defined category (e.g. each category could be a different multicast address). However, unless there are other mechanisms besides those exposed in the AS1 architecture, such approach would still provide a flat arrangement of service categories that cannot scale beyond a small number.

SLP provides two orthogonal means to deal with service diversity. The first of them is the definition of the service itself within a service advertisement. Services belong to a unique service type. Moreover, service types can themselves belong to an abstract service type. Conversely, one service can have attributes that differentiates it from other similar services. The second mean is the concept of scope that allows to group services under a common administration domain. DAs can be configured to selectively listen to service advertisements; however, there is a common default scope that every DA listens to. Furthermore, although scopes provide some degree of scalability, they are not hierarchical which is one reason why SLP could not be deployed over Internet where potentially hundreds or thousands of scopes would be available.

4.3 No Single Point of Failure

Service brokers should not represent a single point of failure to a client requesting a service. Moreover, as explained in 4.1, to avoid a large overhead, the number of service brokers available to a client should depend on the current demand experienced. Statically allocating service brokers is not a scalable solution.

SLP avoids a single point of failure. In case of a DA failure, user agents and service agents can potentially multicast among each other to communicate. Similarly, AS1 avoids a single point of failure by always staying in this multicast-based communication scheme. However, we have already seen the inconvenience of this approach in large service environments.

In addition, when a service broker reincorporates after a failure (or just starts functioning for the first time), it should do it in a seamless way with respect to the other entities it communicates (e.g clients, services, other brokers). A commonly used technique to achieve such functionality is the lightweight session model.

Both SLP and AS1, use the lightweight session model where each entity maintains a soft state of other existing entities. Although adjustable, the maximum timeout periods of the refreshments are normally set quite large under SLP to avoid excessive overhead. However, as the service request rate increases, stale information can become a problem. AS1 timeouts are also adjustable, however the MeGa implementation maintains a very small refreshment period with the subsequent penalty in bandwidth overhead.

Using soft state becomes critical in a very large loosely coupled architecture (such as the one we propose in Chapter 5), where the system as a whole is trying to adapt quickly to the demand of services requested and service brokers are born or die incessantly.

4.4 Network Awareness

The degree of network awareness within a SB architecture can drastically improve the efficiency in resource utilization, making the SB deployment more scalable. Specifically, decisions of when to perform SB reproduction or SB aggregation can be more accurately taken by considering aspects such as:

- Link speed
- Bandwidth bottlenecks
- Latency
- Hop proximity with users and services

Moreover, while we focus only in the deployment of services brokers, the planning and decision layer which sits on top of our proposed architecture -and out of the scope of this document- can greatly benefit too from being network aware. For instance, tradeoffs between bandwidth and CPU cycles, or bandwidth and delay, could be considered, and provide a more accurate quality of service that includes a more efficient use of resources.

Neither SLP nor AS1 involve network-aware elements in their architectures. One exception is the auto-regulation mechanism for the control bandwidth available in both, SLP and AS1, –as well as in RTP/RTCP– where there is a limit in the overall control bandwidth induced by all entities within a service environment. The limit for the control bandwidth is represented as a percentage of the total bandwidth assigned. However, unless there is an external mechanism such as a network administrator who manually configures this percentage according to the total available bandwidth, bandwidth assignments are done arbitrarily as there is normally no way to detect even the immediate link speed[4].

[4] Some applications estimate the total bandwidth available between two points from the round trip time.

Network awareness should be applied in all layers. In fact, its effective inclusion in any application environment not only involves applications that are prepared to "suck" information from the network but also networks that are capable of providing such information. Today we are not at this point yet. However, we believe that along with other research and commercial efforts, network awareness will play a more important role to provide richer services over the Internet in the future.

Based on the evaluation of our criteria we see that the search for a scalable and robust solution has serious limitations and compromises that should be considered.

5 Proposed Solution

We are now ready to propose an architecture for the deployment of services brokers over the Internet by merging SLP and AS1 and adding our own design elements.
Of the two frameworks studied so far we will take SLP and build from there on with design concepts from AS1 and our own. The justification for the use of SLP is twofold: (i) it is a known Internet standard, (ii) it provides a basic service brokerage functionality that we can extend more logically to our design needs.

5.1 Design Goals

As is clearly stated in SLP version 2,

> SLP has been designed to serve enterprise networks with shared services, and it may not necessarily scale for wide-area service discovery throughout the global Internet, or in networks where there are hundreds of thousands of clients or tens of thousands of services.

Table 5.1 shows our criteria defined in Chapter 4 compared against SLP.

	CURRENT STATE OF SLP
Adapt to High Usage Variability	No
Adapt to High Service Diversity	No
No Single Point of Failure	Yes
Network Awareness	No

Table 5-1 Design Criteria and SLP

If we could extend SLP by addressing each of the criteria shown above, we will be closer point to an ideal solution. As described earlier, a service brokerage architecture is composed of two layers: service discovery and service selection. The design we will propose is along the lines of the former. However, we extend the concept of service discovery to the brokerage service itself by proposing a deployment architecture of service brokers over an open network environment. In addition, similar to AS1, we extend SLP's notion of services by decoupling them from the actual resources or service sites that provide them.

Accordingly, we first cast a Directory Agent under SLP as a raw service broker and we then define three services: reproduction of service brokers, aggregation of service brokers, and reincarnation of service brokers.

5.2 Reproduction of Service Brokers

Reproduction of service brokers occurs when the current SB population cannot fulfill the demand for services. One way to achieve this is to make an SB aware that a bottleneck is emerging. Bottlenecks could be measured in different ways, each of them requiring different levels of knowledge about the environment surrounding the SB. Simple measures could be the CPU load of the SB hosting machine. Other measures could be added such as some degree of network awareness (e.g. adjacent link bandwidth consumption).

Once an SB decides to reproduce we imagine two approaches for how to make this reproduction to happen:

- Increasing the level of service categorization

- Increasing the level of locality

5.2.1 Increasing the level of service categorization

We propose to subcategorize the type of services a SB brokers. Once an SB decides to reproduce, the services it was brokering before reproduction will now be distributed among itself and the new SBs. This means that certain users will be redirected to a different SB.

In SLP, there is the notion of abstract service type, service type, service, and service attributes. Under the approach we propose, this classification could be repurposed in order to categorize services and redistribute them among the SBs resulting after reproduction. However, a more scalable solution would inevitably involve redefining the current service naming in SLP allowing for hierarchies of services to exist and grow or shrink on demand. Such new naming structure would have no fundamental need for a priori knowledge of the service categorization. Instead, the actual information taken at runtime from the service requests is used to categorize. This means that as the number of users increases, the hierarchy depth could increase too. For instance, an SB for an video streaming service could then specialize on video on-demand.

As shown in Figure 5.1 a typical SB deployment scenario could require a user to maintain a relationship with different service brokers. Resource owners (represented by SAs in SLP) could also subscribe their services with different SBs, however, the decision for reproducing should consider actual demand. In other words, we do not want to utilize service brokerage resources just because more services are being subscribed when there are no users demanding their services. Similarly, in order to avoid underutilized SBs, reproduction should not necessarily be symmetrical. That is, reproducing could mean having one very specialized SB (where demand has increased) and one more general SB.

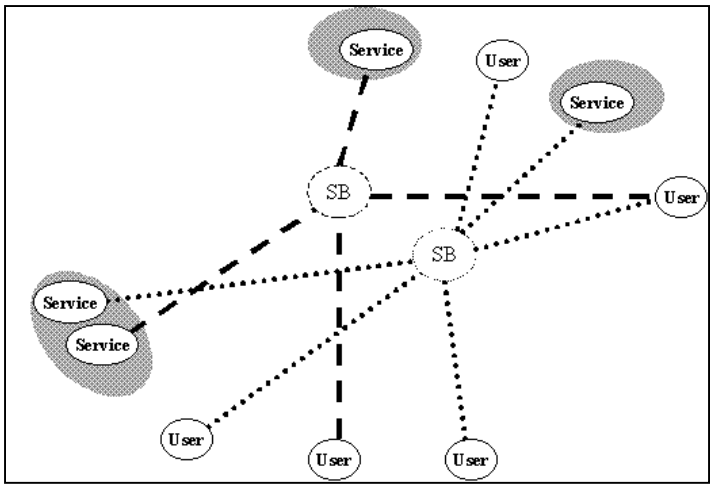


Figure 5-1 User Maintaining a Relationship with Two SBs

5.2.2 Increasing the level of locality

Another criterion to achieve SB reproduction considers the administrative differences among users. In this way, SBs would reproduce by increasing the level of locality within the service environment they are deployed. Figure 5.2 illustrates reproduction of SB by locality. Again, network awareness can play an important factor in achieving the most efficient way to reproduce an SB on a localization basis.

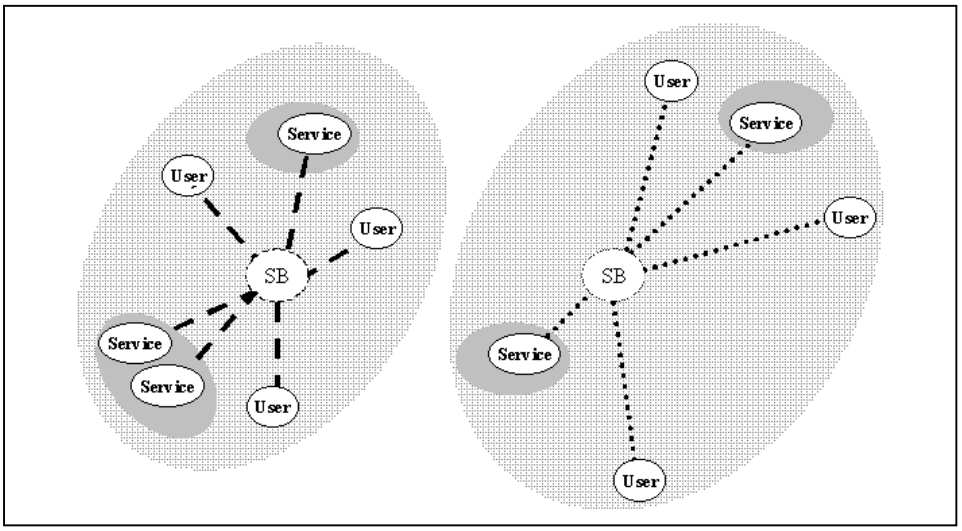


Figure 5-2 SB Reproduction by Locality

5.3 Aggregation of Service Brokers

Similar to how an SB can detect an emerging bottleneck, awareness of underutilized SB resources could be detected to enable an aggregation process. Once an SB has decided to combine with another, the first aggregation candidate to consider should be its other half SB (after reproduction). If there is no recollection of the other half, or it is unreachable, a SB discovery process can take place similar to how UAs and SAs discover DAs. Such discovery process should be done incrementally by looking up in the service hierarchy. The actual look up mechanisms depends on whether the current stage of discovery is using a service type categorization or locality criteria. In either case, the discovery process continues until another SB is found or until the point where the SB would find itself being the unique general broker within the service environment perceived. This mechanism is not only useful when adapting to network partitions, more importantly, it is a key concept for being able to deploy a self-regulated network of service brokers that always will tend to broker every possible type of service.

In the case that another SB has been found, aggregation can be attempted according to the following steps:

1. Exchange of resource usage information
2. If at least one of the SBs can handle the other's load continue, otherwise abort
3. Migration of state may occur to accelerate aggregation, however it is not necessary as future refreshments will eventually build the original soft state
4. Redirection of users and services to new aggregate SB

The last step is different for each group of users and services. Those entities who's SB will no longer exist should be redirected with the address and new scope of the aggregate SB. Entities who were already communicating to the aggregate SB should only be notified of its more general service scope.

5.4 Reincarnation of Service Brokers

Assuming that the entities the service broker communicates with (other SBs, services themselves, and users) can detect its failure, these other entities could potentially "reincarnate" the service broker functionality. To avoid confusion among all the potential reincarnating entities, a well understood set of steps should follow to define who should reincarnate the dead SB. We call this set of steps a *reincarnation protocol*. This protocol can be quiet simple, again using existing techniques such as soft state and multicast damping. In particular, we propose that every entity should maintain state about every other entity which it can potentially reincarnate. Moreover, as this state would be kept in a soft state table, time-outs would be proportional to the level of proximity between the

two entities. Ways to measure proximity include whether the two entities share the same service type, belong to the same administrative domain, or have the same role (SB, service provider, customer). Thus, when a SB dies, "closer" entities will timeout first and begin reincarnation before others. Still, entities within the same proximity group could try to reincarnate simultaneously. Similar to how AS1 tries to avoid multiple launchings of the same servent, the reincarnation behavior could include a mechanism similar to multicast damping[5]. Therefore, every entity will introduce a random factor for when to actually timeout and begin reincarnating the dead SB. Finally, whenever an entity begins reincarnation, it will announce this information so that all other entities listening will "damp" themselves from beginning an additional reincarnation process. In the case of simultaneous launchings, a conflict resolution mechanism must be introduced.

The mechanism above described has a potential restriction about how entities communicate among each other. That is, in order to have the reincarnated SB make the other entities aware of its existence, there must be a well-known point of contact that the reincarnated SB can utilize. We could imagine this happening in at least two ways: (1) through a multicast address that all entities knew before as the point of contact for SB announcements, or (2) by having the reincarnated SB to be aware of the entities that the dead SB was communicating with. The former should be preferred whenever multicast is available, as the latter can have a huge impact on the amount of information that every entity has to maintain. Although we believe that multicast is a necessary technology, the complexity it involves affects the chances it has of being fully deployed over the Internet in the near future. In Section 5.7, we will elaborate the mechanisms for whenever multicast is only partially available.

5.5 Other Considerations

In a real world environment we expect to have more complex scenarios than the ones we have described. For instance, we can easily imagine that different ISPs could impose limits to the deployment of SBs within their domains. In addition, there are some services such as printing where their service environment is inherently local and there would be no apparent reason to change that. Conversely, some other services will tend to have a more remote service environment, e.g. media compression services in remote congested links. More formally, we can classify a service with respect to the entity issuing the request in the following way:

- Local
- Remote
- Ubiquitous

The category where each service falls plus the administrative boundaries imposed by ISPs will determine how a SB for such service should be reproduced or aggregated.

[5] This mechanism was first introduced in Multicast IP, although under a different context

Figure 5.3 illustrates a more complete scenario under such conditions. Notice how the SB in the center is servicing users that are in other administrative domains. This could be possible as long as the SB has the appropriate security requirements, which is outside of the scope of this document. Still, crossing administrative regions is a factor to consider in the decision making process. For instance, we can imagine a service broker giving a higher weight to a service provider in its own administrative region.

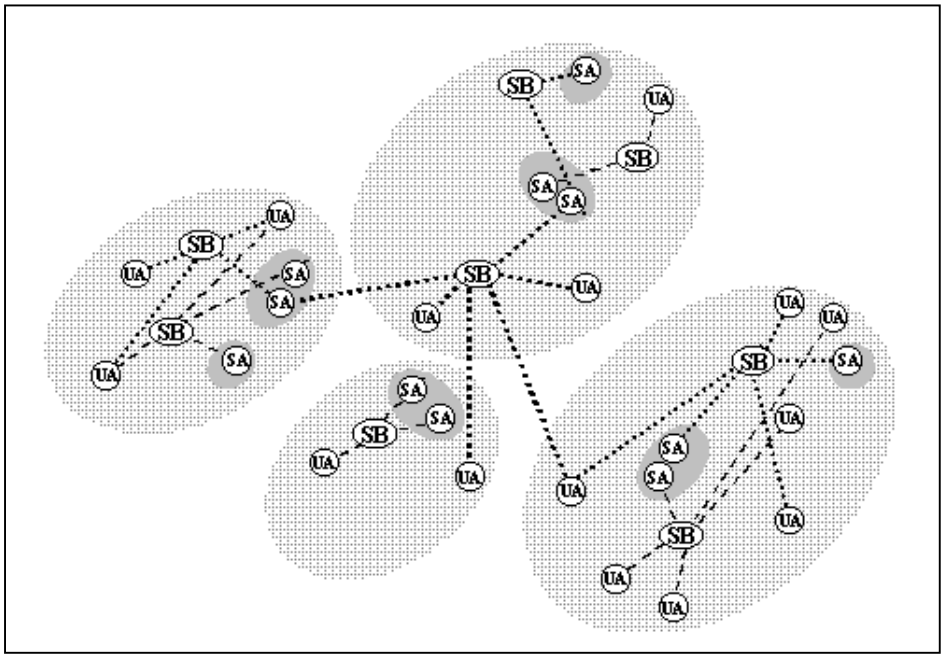


Figure 5-3 Administrative Boundaries

5.6 Implications of Centralizing Information into a Single Broker

We have previously referred to the overhead saved by having a single broker centralizing information within a service environment, compared to a decentralized scheme where all entities communicate among each other. Another advantage of a centralized approach is the degree of intelligence that an SB can potentially manifest. By centralizing information from the service environment where it is deployed, an SB could make more sophisticated decisions concerning how to map service requests to services. Even in the case of having a decentralized architecture that could achieve the same degree of intelligence, it may imply replicating such intelligence in each service entity with the consequent impact in resource utilization.

In reality, replicating SB intelligence across services entities does not necessarily consume *n* times more computations cycles compared to a central one, where *n* is the number of entities. Decentralized architectures are typically more CPU efficient - they do consume more CPU in aggregate but usually not n times more. However, they have

access to n times more CPU resources - which makes them faster. This is a driving force for why the desire to parallelize things in general. Conversely, a central scheme can potentially achieve a much greater level of SB intelligence with much less overall resource consumption but with the potential penalty in speed (e.g. response time).

5.7 The Effective Resource Usage Metric

To measure the effectiveness of our designs, we define a benefit/cost metric called Effective Resource Usage Metric (ERUM). An appropriate metric for the benefit would be the responsiveness of the system, i.e. how quickly can the SB satisfy a request. However, since we did not measure the response time, we will approximate the response time by the inverse of the average CPU load on the service brokers[6]. Since a higher load will translate into a slower response time, that should be a reasonable approximation. For the cost, we will use the bandwidth consumed by each SB. We could also include the number of SBs used in the cost function, but since a larger number servers results in a higher network load, this effect is already captured indirectly. We end up with the following definition of ERUM:

$$ERUM(\overline{CPULoad}, bandwidth) = \frac{1}{\overline{CPULoad} \times bandwidth}$$

Equation 5-1

More generally, assuming there are other resources in the system that significantly affect response time, and need to be minimized, we can extend the notion of resources —to include other aspects such as data storage. Equation 5.2 represents a large number of resources.

$$ERUM \quad (r_1, r_2, \ldots r_n) = \frac{Benefit}{r_1 \times r_2 \times \ldots \times r_n}$$

where $r_1, r_2, \ldots r_n$ are resources

Equation 5-2

[6] Another approximation could be to use the distance to the SB, which in some cases could be more reflective of response time than BW or CPU.

ERUM is merely reflecting the benefit/cost tradeoffs in a SB deployment architecture for the distribution of resources. In Chapter 6 we use ERUM to measure the effectiveness of different architectures, however, ERUM could also be used within one particular architecture as a tuning aid to find the number of SBs that will yield the optimal allocation of resources (highest benefit). To further develop this idea is food for future work.

5.8 Multicast Availability

Although we believe that multicast technologies in the network layer are crucial for the scalability of any service broker architecture in general, we are aware of the complexity involved in its deployment and further maintenance. Taken from the AS1 framework, we propose the use of soft state gateways that participate in a multicast session on behalf of the unicast-connected entity. We recognize however that such scheme works only over networks that are at least partially multicast enabled.

5.9 Regulation of Service Resources Population

Similarly to SB reproduction and SB aggregation, we can imagine mechanisms that would allow an SB to grow or shrink a service population according to the demand of users. AS1 already provides an equivalent mechanism although the target population number is static[7] and launching new host managers occurs only when one of them fails or when the initial population is below the target.

[7] As of mash version 5.01b

6 Experimental Design

To evaluate the design, an experiment is performed where a subset of the proposed architecture is implemented and tested. We created a prototype called *Joxer*. In particular, we focus on the adaptive characteristics that allow an SB to reproduce or aggregate on demand. The success criterion used is ERUM, which has been defined in Chapter 5.

6.1 Implementation

The *Joxer* prototype runs in a network testbed consisting of four service brokers and a load generator. The service brokers and load generator are PCs running FreeBSD.

To develop the service broker code, we modified the original highly decentralized MeGa service –built on top of the mash toolkit. We were not interested in the MeGa transcoding service per se, but instead in the AS1 mechanisms, which we extended to fit our design needs. Therefore, we constrained our modifications to the active services control protocol (ASCP), leaving the other protocols involved in the MeGa service (RTP/RTCP, SCUBA, etc) intact.

6.2 Experimental Setup

We propose the following 3 service brokerage scenarios to compare under an identical set of service requests across time:

- Centralized SB
- Statically distributed SBs
- Dynamically distributed SBs

In all three scenarios, we inject into the system an equal number of service requests for a period of 61 seconds. The service requests were evenly distributed during that time period. In addition, every SB periodically sends two types of multicast updates. The first is an announcement of its services to potential customers. The second type of update is a report sent to other SBs about the services offered and the customers being attended at that moment. The reports are used by the SBs to maintain soft state tables as explained in Chapter 5. Service requests vary in the level of specialization required to broker them. When an SB receives a service request it must be specialized to broker it. If the service broker doesn't have the appropriate specialization it will need to acquire it. In the Joxer testbed, we assume that specialization is obtained from a well-known point-of-contact. Moreover, on each specialization acquired by a SB, CPU load and bandwidth is affected.

Suppose a service request is constructed in the following way:

/generic/audio/mp3/

Each of these subdivisions implies a level of specialization required in the SB. That is, in order to have an SB to broker this request, it must have obtained the generic specialization (e.g. bandwidth, cycles), the audio specialization (e.g. knowledge of compression algorithms, tradeoffs among them), and the mp3 specialization (e.g. copyrights permissions). In our experiment, each specialization builds on top of the ones below to assure a full end-to-end quality of service. However, as explained in Chapter 5, we imagine that each level of specialization by itself can provide a brokerage service to certain type of applications. There will be applications asking only for bandwidth and cycles. Furthermore, certain service may require more than one path of specialization below them, for instance, videoconferencing may require both video and audio specialization. In this way a broker can specialize in several areas at the same time.

6.3 Centralized SB

In the case of a centralized SB, there is only one service broker that is trying to service all incoming requests, see Figure 6.1. Multicast announcements are sent by the SB to announce its presence to potential customers. Multicast reports to other SBs are sent too, despite there is only one SB in the system.

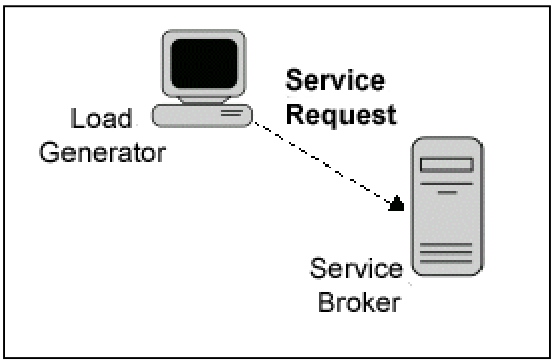


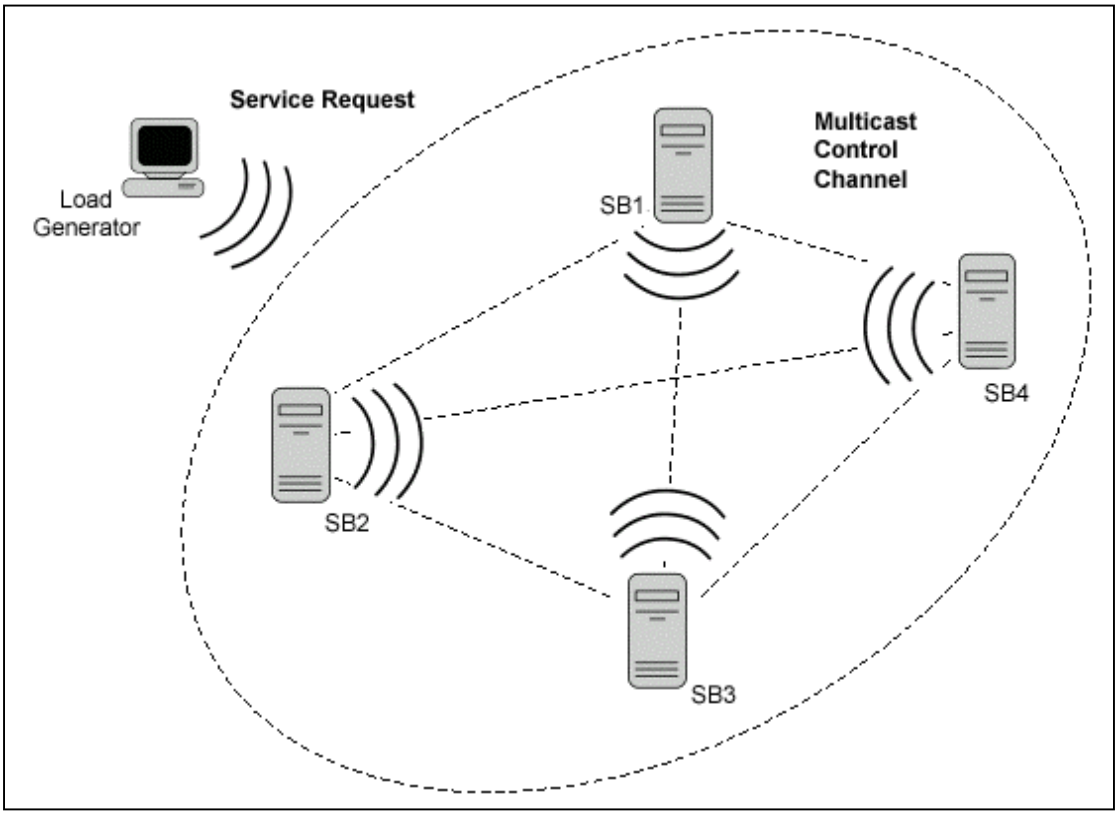
Figure 6-1 Centralized SB

6.4 Statically distributed SBs

In this scenario, the population of service brokers is maintained static across time. Each SB specializes differently according to the requests it services, see Figure 6-2. Moreover, assuming we maintain all other variables equal, if two service brokers receive a service request (/generic/video/videoconferencing/) and one has no specialization and the other is

specialized in /generic/video/videoconferencing/, the later should service this request – being the "expert". In the case that both SBs have the same specialization, we chose the CPU load as the selection criterion. However, if the CPU loads of both SBs is above an arbitrary defined threshold of 0.7, a third SB, with a lighter CPU load, will service the request.

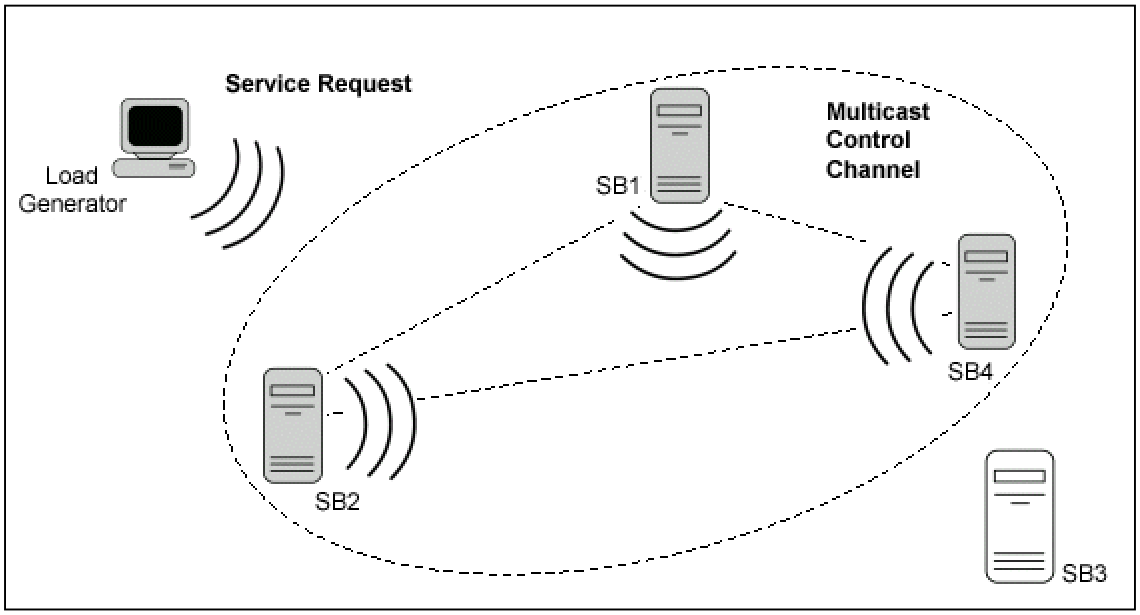
Figure 6-2 Statically Distributed SBs



6.5 Dynamically distributed SBs

Starting as a centralized service brokerage (see Section 6.1), SB reproduction –and subsequent aggregation– takes place according to the demand of service requests. We apply the criteria described in the statically distributed scenario to determine which SB services a request when two or more are present. In addition, if a SB has a CPU load above the load threshold (0.7) and doesn't detect any other SB with a CPU load below the threshold, the reproduction process will be triggered, spawning a new SB, see Figure 6-3.

Figure 6-3 Dynamically Distributed SBs



6.6 Measures

In the experiments only CPU load and bandwidth are considered. Each time an event occurs, we log it. For the sake of simplicity, we assign certain values for CPU and bandwidth consumption for each type of event (service hit, update report, and specialization)**.** Each event is assigned a duration of 1.0 second. See Table 6–1. CPU load assigned to an event is given as a fraction of the total system CPU cycles (used and idle). In addition, if an SB gets a service hit and decides to service it, additional CPU cycles are assigned for processing the request. The additional increase in CPU load will remain effective for a time dependent on the type of service being brokered. The more complex the service is, the more the time we assign to broker the request. In addition, a factor is introduced to represent load-dependent differences in the duration of the request being processed. Table 6-2 shows the list of hypothetical service requests used in our experiment along with the values for CPU load we assigned them.

CPU (fraction of total cycles)		BW (Kb/s)	
Receiving a service hit	0.01	Service hit	1
Sending an update report	0.01	Update report	1
Receiving an update report	0.01	Specialization	5
Specializing	0.05		

Table 6-1 Resource Consumption per Type of Event

Service	CPU (fraction of total cycles)
/generic/films/starwars/episode3/	0.2
/generic/disted/	0.1
/generic/disted/unext/mba/finance/	0.2
/generic/news/cnn/	0.2
/generic/films/independent/happytexas/	0.2
/generic/banwidth/	0.2
/generic/videoconferencing/highquality/	0.3
/generic/films/starwars/episode3/spanish/	0.3
/generic/advertisement/poland/class1	0.2
/generic/sports/worldcup/1986/final/	0.1
/generic/news/tf1/	0.1
/generic/cpu/500mhz/	0.1
/generic/bandwidth/t1/	0.2
/generic/videoconferencing/highquality/	0.3

Table 6-2. Hypothetical Service Requests Used

7 Measurements

It is important to remark that the experiments focused on measuring how the dynamically distributed scenario, implemented upon our proposed service brokerage framework (Chapter 5) compares against scenarios 1 (centralized) and 2 (statically distributed). Our comparisons measures are overhead bandwidth and incurred CPU load levels.As shown in Figure 7.1, a centralized scenario (dotted line) will consume the least amount of bandwidth because overhead bandwidth due to multicast announcements and reports is minimum, as there is only one SB.

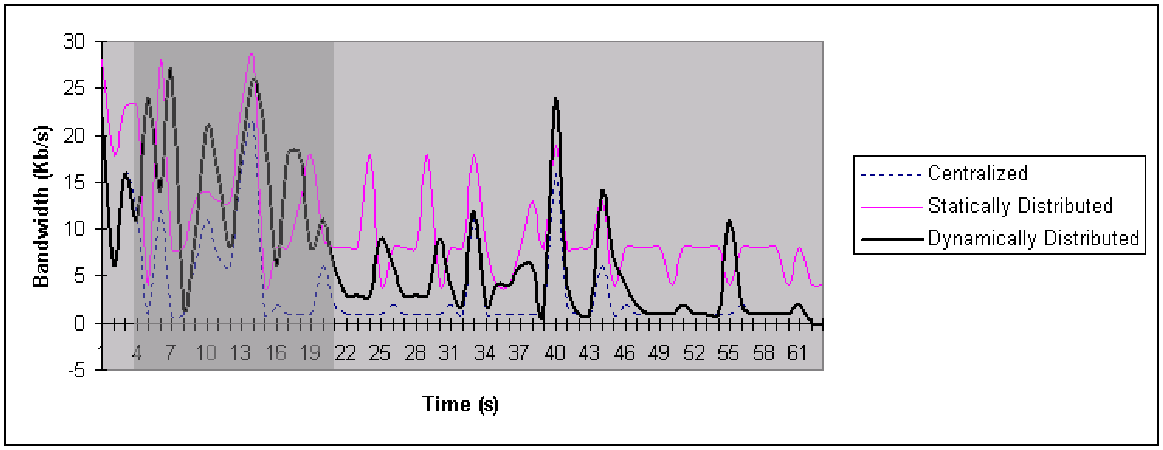


Figure 7-1 Bandwidth Consumption

Also, because there is only one SB in the centralized scenario, the bandwidth penalty incurred when acquiring a specialization happens only once and further requests for the same service don't involve any new intelligence to be acquired, which is depicted by the a heavier bandwidth consumption in the beginning. Scenario 2 and 3 present no great difference when SB population is similar (shadowed section), otherwise the statically distributed scenario is more bandwidth intensive as the SB population doesn't shrink and state maintenance becomes more expensive to maintain as service requests decrease.

Conversely in Figure 7.2, as there is only one SB who is receiving the entire load, the centralized scenario presents the highest consumption in CPU cycles. The distributed scenarios present a much lower average CPU consumption as there are more SBs sharing the load. Of the two distributed scenarios, the static one presents the least CPU consumption per SB due to the fact that there are always four SB active–as opposed to starting with one SB in the dynamic scenario and reproducing and aggregating SB as necessary.

As explained Section 5.7, we did not include response time in our measurements, however, as the response time is mostly a function of precisely CPU load and bandwidth

utilization, we argue that minimizing response time is addressed by minimizing cpu load and bandwidth. For instance, as described in Section 6.4 and Section 6.5, in the two distributed systems (static and dynamic), an SB that has reached 0.7 in CPU load will first let other SBs service a request (even if it has all the necessary specialization to service it). Therefore one could argue that when compared to the centralized scenario, these two scenarios are preventing an SB from incurring large response times. In Figure 7-2, the high CPU load in the centralized scenario suggests a long response time relative to the other two scenarios .

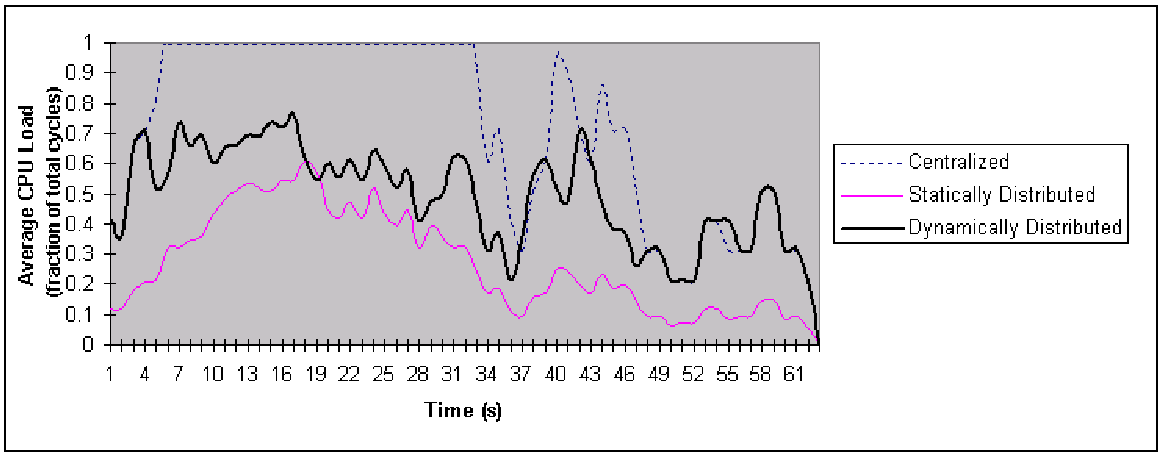


Figure 7-2 CPU Load Consumption

Finally, using ERUM, the benefit/cost metric we defined in Section 5.7, we can see that the dynamically distributed scenario behaves similar to other two scenarios, which is exactly what we wanted, see Figure 7-3. This is because both, centralized and statically distributed, are ideal scenarios respectively when CPU cycles and bandwidth are abundant. However, they do not scale to large numbers of service requests -centralized- or large numbers of SBs -statically distributed- as cost becomes excessively high if not impossible to maintain. By dynamically reproducing and aggregating SBs we can preserve a similar value of *ERUM* in an organic fashion.

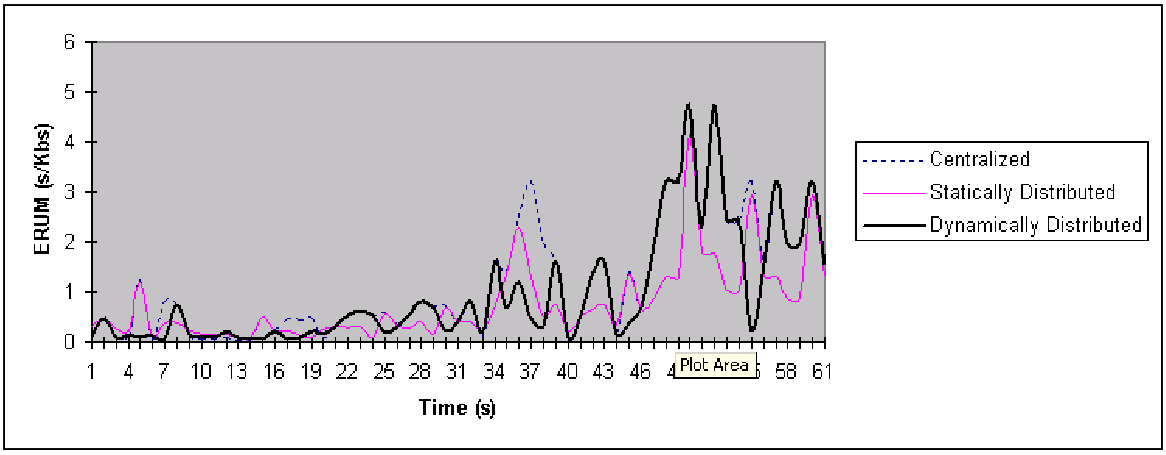


Figure 7-3 ERUM

8 Summary and Future Work

As more and different services appear over the Internet, there is a need to have a brokerage architecture that abstracts complexity from the user and is scalable enough to be internet-wide deployed.

We have presented an architecture and proof of concept for the deployment of service brokers over the Internet. We have done this by merging SLP and AS1 and adding our own design elements. The design considerations for the work presented were (1) high variability of usage, (2) high diversity of services and resources, (3) no single point of failure, and (4) network awareness.

In addition, we decoupled the notion of agent, service, and resource in three separate logical components and we illustrate how SLP and AS1 behave fundamentally different in their perception of them.

In the architecture proposed, a service broker optimizes resource consumption by deciding to reproduce or aggregate based on the demand of their services and the consequent load they experience. A service broker can also "reincarnate" the functionality of another service broker who ceased to function.

To measure the effectiveness of our designs, we defined a performance metric called Effective Resource Usage Metric (ERUM). We defined ERUM as the inverse of bandwidth utilization times average CPU load, consumed by active SBs

8.1 Future work

We have provided arguments to validate what we consider a promising architecture; however, future work should focus on testing different service loads as well as measuring different resources besides CPU load and bandwidth consumption, like storage or response time. In addition, there are a number of areas related to the deployment of a service broker where we can extend the work here presented. We explain briefly three of them: weighted ERUM, intelligent multicasting, and specialization of service brokers.

8.1.1 Weighted ERUM

An area of future research is to extend ERUM to represent the impact that each resource has in the response time by assigning weights to them. Weights can be used for representing network conditions, which convert one resource more significant than another one. Equation 8-1 represents a weighted ERUM for a potentially infinite number of resources:

$$ERUM \quad (r_1, r_2, \ldots r_n) = \frac{Benefit}{r_1^{e_1} \times r_2^{e_2} \times \ldots \times r_n^{e_n}}$$

Equation 8-1

Where e_n is the weight assigned to r_n. Thus, the more significant a resource is –higher e_n–, the less it should be consumed in order to maintain a similar value of ERUM.

Equation 9-1 has the limitation that r_n must be always be greater or equal than 1. If r_n is smaller than 1, the effect of e_n will be inversed. Another option would be:

$$ERUM \quad (r_1, r_2, \ldots r_n) = \frac{Benefit}{(1 + r_1)^{e_1} \times (1 + r_2)^{e_2} \times \ldots \times (1 + r_n)^{e_n}}$$

Equation 8-2

However, in this case, small values of r_n would be overwritten by the 1 in the equation. Other options need to be considered.

8.1.2 Considerations on Service Broker Specialization

Another open issue is the case when an SB reaches its CPU load threshold and a service request arrives for which the SB is the only one with the appropriate specialization to broker the request. Normally, one of two things could happen, a less loaded –but less expert- SB will service the request or, if all SBs are at their maximum load, a reproduction process will take place spawning a new SB. In either case, there is a cost for the specialization of the new SB that needs to be considered, which could lead the specialized but overloaded SB to service the request, despite the potential delay. Besides CPU load and bandwidth consumed, this consideration has implications in two variables, storage of the SB specialization and response time to service the request, which are food for future work.

8.1.3 Intelligent Multicasting

As mentioned in Chapter 5, maintaining state among SBs is achieved by multicast technologies when available. Our experiments used a multicast address where all SBs subscribe and share information among each other. When a service request is submitted into this shared space, every SB will know who should be the most appropriate to broker such request (the most expert, the less loaded, etc). Even under a dynamic distribution of SBs, this solution imposes a heavy overhead on each request hitting every SB and every SB communicating a potentially large amount of information to every other SB. Another solution could be to have different multicast addresses for different specializations areas. This implies having a large number of multicast addresses. However, the overhead is still heavy because specialized brokers will now subscribe to many multicast addresses. In this subsection we explore the possibility of using what we call intelligent multicasting.

Intelligent multicasting would require only one multicast address, but that address would carry hierarchical information in it. The idea is that if a service broker could send its degree of specialization with the request its subscription to a multicast address, service requests could potentially be routed based not only in the address but in the specialization too. This is not that same as having multiple multicast addresses, each representing a degree of specialization. Instead, it is a hierarchical array of virtual multicast addresses contained within one physical multicast address. In this way, a service request would initially touch only those SBs who are prepared to respond, reducing the bandwidth and CPU overhead For instance, consider a service request that arrives at the root of a multicast tree,

/generic/audio/mp3/

Assuming a generic multicast protocol for session management, a router would normally forward a multicast packet based on the existence of subscribers to that multicast address. In this case, a router will have some degree of intelligence about the SBs down the multicast tree in order to reduce the number of interfaces to which to forward the service request.

Having multicast routers to be able to understand beyond the multicast address implies introducing more intelligence in a router who's primary objective is to move packets around fast. Still, we believe that a service brokerage network such as the one proposed herein should be supported even in the routers, seen as a meta service that is general enough to enclose any type of service and therefore, its importance should be perceived by a router in terms of the amount of resources that would be used otherwise, including resources in the same routers (maintaining several multicast address, forwarding more packets).

This technique doesn't intend to find a unique service broker, but instead, to reduce the overhead bandwidth by reducing the number of services brokers that will listen to a service request. Using an expanding ring search technique, as the same request arrives for the second time to a router, the router can relax the criteria to forward the service request (e.g. by neglecting the deepest level of expertise). Therefore, the request will now be forwarded to a broader audience of service brokers.

In our experimentation we didn't consider an intelligent multicast scenario such as the one here described, however, figure 8-1 extrapolates our simulation to a scenario where intelligent multicasting is deployed. The extrapolation criteria were very basic, as it assumed that all requests were serviced in the first transmission and only one SB would receive the request. However, despite the rather simplistic assumptions, we strongly believe the results represent an ideal case that is meaningful enough to be presented. For instance, both scenarios with intelligent multicasting present in general a higher *ERUM*. It is interesting to notice that during the last period of the experimentation, the statically distributed scenario with intelligence multicasting achieves by far the highest value of *ERUM*. One reason for this is because during the last period the frequency of service requests is still constant but demand for specialization is considerably lower. Therefore,

the bandwidth penalty for having every active SB to receive a request has been reduced by sending the service request to one SB always. Figure 8-1 is not definitive because the original bandwidth penalty has not disappeared but instead transformed into a CPU load on every multicast router, which is not represented here. Still, the significantly high value of *ERUM* gives us an important argument for doing more research.

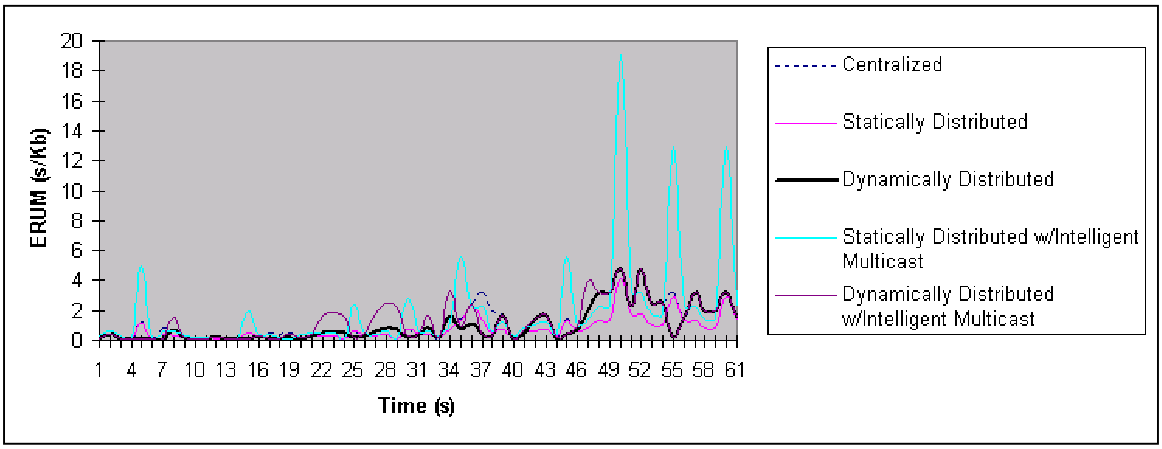


Figure 8-1 Performance of ERUM using Intelligent Multicast

References

[1]
Service Location Protocol for Enterprise Networks
Kempf J., St. Pierre P.
John Wiley & Sons, Inc. 1999

[2]
Toward a Common Infrastructure for Multimedia-Networking Middleware.
McCanne S. et al. In Proc. 7th Intl. Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV '97), St. Louis, Missouri, May 1997.

[3]
An Active Service Framework and its Application to Real-time Multimedia Transcoding.
Amir E., McCanne S., Katz R. Proceedings of ACM SIGCOMM '98, Vancouver, British Columbia, September 1998.

[4]
Chandra P., Fisher A., Kosak C., Eugene Ng T.S., Steenkiste P., Takahashi E., Zhang H., *Darwin: Resource Management for Value-Added Customizable Network Service*, Sixth IEEE International Conference on Network Protocols (ICNP'98), Austin, October 1998

[5]
Chandra P., Fisher A., Steenkiste P., *Beagle: A Resource Allocation Protocol for an Advanced Services Internet*, technical report CMU-CS-98-150, August 1998

[6]
Chandra P., Fisher A., Kosak C., Steenkiste P., *Network Support for Application-Oriented Quality of Service*, Sixth IEEE/IFIP International Workshop on Quality of Service, Napa, May 98.

[7]
Stoica I., Zhang H., Eugene Ng T. S., *A Hierarchical Fair Service Curve Algorithm for Link-Sharing, Real-Time and Priority Service* To appear in Proceedings of SIGCOMM'97.

[8]
Davie B., Casner S., Iturralde C., Oran D., Wroclawski J., Integrated Services in the Presence of Compressible Flows, RFC 3006, November 2000.

[9]
J. Rosenberg, H., Schulzrinne, and B. Suter. Wide area network service location. Internet Draft, work in progress, IETF, November 1997.
http://www.bell-labs.com/mailing-lists/wasrv/

[10]
Perkins, C. Wide Area Service Location Protocol, work in progress, March 1998. http://www.bell-labs.com/mailing-lists/wasrv

[11]
Handley, M. *SAP: Session Announcement Protocol*, Mar. 1997. Internet Draft.

[12]
Schulzrinne H., Casner S., Frederick R., and Jacobson V. *RTP: a transport protocol for real-time applications*. RFC 1889, Internet Engineering Task Force, Jan. 1996.

[13]
Czerwinski, S. et al. An Architecture for a Secure Service Discovery Service. In Fifth Annual ACM/IEEE International Conference on Mobile Computing and Networking, pages 24 -- 35, Seattle, WA USA, August 1999.