AFRL-IF-RS-TR-2004-42
Final Technical Report
February 2004

SCALABLE ADVANCED NETWORK SERVICES BASED ON COORDINATED ACTIVE COMPONENTS

Carnegie Mellon University

Sponsored by
Defense Advanced Research Projects Agency
DARPA Order No. H489

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

APPROVED: /s/
SCOTT S. SHYNE
Project Engineer

FOR THE DIRECTOR: /s/
WARREN H. DEBANY, JR.
Technical Advisor
Information Grid Division
Information Directorate

REPORT DOCUMENTATION PAGE	*Form Approved OMB No. 074-0188*

1. AGENCY USE ONLY (Leave blank)	**2. REPORT DATE** FEBRUARY 2004	**3. REPORT TYPE AND DATES COVERED** **FINAL Jun 99 – Sep 03**

4. TITLE AND SUBTITLE SCALABLE ADVANCED NETWORK SERVICES BASED ON COORDINATED ACTIVE COMPONENTS	**5. FUNDING NUMBERS** G - F30602-99-1-0518 PE - 62301E PR - H489 TA - 00 WU - 01
6. AUTHOR(S) Peter Steenkiste Srini Seshan Hui Zhang	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Carnegie Mellon University 5000 Forbes Avenue Pittsburgh PA 15213-3890	**8. PERFORMING ORGANIZATION REPORT NUMBER** N/A
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) Defense Advanced Research Projects Agency 3701 North Fairfax Drive Arlington VA 22203-1714 — AFRL/IFGA 525 Brooks Road Rome NY 13441-4505	**10. SPONSORING / MONITORING AGENCY REPORT NUMBER** AFRL-IF-RS-TR-2004-42

11. SUPPLEMENTARY NOTES

AFRL Project Engineer: Scott S. Shyne/IFGA/(315) 330-4819 Scott.Shyne@rl.af.mil

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

13. ABSTRACT *(Maximum 200 Words)*

The Internet has evolved from a simple best-effort communication platform to an infrastructure that delivers a wide range of services to end users. However, deploying services that are both scalable and provide rich functionality remains a major challenge. The CMU Libra project developed a set of technologies that allows new services to be defined through composition of service components. Service components either take the form of highly scalable communication layers that are self-organizing or of higher level building blocks that offer computation or storage functionality. Active networking technology is used both to customize generic service components so they better support specific user requirement, and to deployment services instances where they are needed. A shared service infrastructure implements basic operations such as network measurements and universal connectivity. These technologies were demonstrated using a set of example services that include multicast, distributed games, video conferencing, and broadcasting.

14. SUBJECT TERMS active networking, services, network, multicase	**15. NUMBER OF PAGES** 36
	16. PRICE CODE

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

Table of Contents

1 Summary 1
2 Introduction 2
3 Methods, Assumptions, and Procedures 4
 3.1 The Players 4
 3.2 Service Composition Process 4
 3.3 Research Approach 5
4 Results 5
 4.1 An Example: Video Conferencing 5
 4.2 Service Deployment Architecture 7
 4.3 Service synthesis 8
 4.4 Network-Sensitive Service Discovery (NSSD) 11
  4.4.1 NSSD API 11
  4.4.2 Query processing 12
  4.4.3 Alternative approaches 13
 4.5 ESM 13
  4.5.1 Architecture 14
  4.5.2 The Case for End System Multicast 15
  4.5.3 Light-Weight Measurement-Based Optimization 15
  4.5.4 Using End System Multicast in the Internet 16
 4.6 Measurement Infrastructure 17
  4.6.1 Global Network Positioning 17
  4.6.2 Packet Train Rate Probing 17
  4.6.3 Address Virtualization Enabling Service (AVES) for Incrementally Scaling the Internet Address Space 18
 4.7 Publish-subscribe communication services 18
  4.7.1 Mercury 19
  4.7.2 Camel 20
 4.8 Service Customization 21
  4.8.1 Classes of EEs 21
  4.8.2 Customization 22
 4.9 Congestion control 24
5 Discussion and Conclusions 25
References 27

List of Figures

Figure 1: Entities in the service model 3
Figure 2: A video conferencing scenario 6
Figure 3: Hierarchical service composition example: distance learning 7
Figure 4: Synthesizer steps for the video conferencing scenario 8
Figure 5: A service recipe for the video conferencing service 9
Figure 6: A service recipe for the end system multicast (ESM) service 10
Figure 7: Handling an NSSD query 12
Figure 8: Classes of Execution Environments (EE): (a) overlay EE and (b) control EE 22
Figure 9: Active Service Customization for (a) overlay EEs and (b) control EEs 23

1 Summary

The Internet is trying to evolve from a best-effort network that moves data files to a sophisticated infrastructure that delivers value-added services such as video conferencing, virtual reality games, and distributed simulation. However, providing such services in a scalable fashion turns out to be very challenging. Today's solution is to deploy *vertically-integrated* services, i.e. the service provider decides on the required service functionality and develops a tightly integrated system that delivers the service to users. This approach has many disadvantages: services are hard to modify and extend, it does not easily allow code reuse, and it is difficult to exploit new network features or to meet unique requirements of specific customers.

As an alternative to vertically-integrated services, Libra supports the development of new services through composition of existing, primitive services. For example, a distance learning service could build on existing video conferencing and content distribution services. This approach is more flexible since it allows services to be modified and extended by replacing some components. It also automatically supports code reuse and it allows advanced network features to be used by developing components that are optimized to exploit the new technologies. However, this approach comes with its own set of challenges. First, how should we manage the diverse resources that are needed to support the computational and communication requirements of the service components. Second, how can we meet user specific requirements given that components only offer generic service functionality.

We address the resource management problem by having components use resource management mechanisms that match their functionality. Specifically, Libra uses two classes of components. The first class consists of low-level communication services that are generic and are shared by many high level services. As a result, scalability is a key concern. Such services will in general rely on distributed self-organization to optimize resource use and communication performance. We explore self organization in the context of three example communication services: the Narada End-System Multicast services, and the Mercury and Camel publish-subscribe systems. Our results show that self-organization is an attractive way of managing this class of components, assuming the system adapts to resource availability and can be extended to meet user-specific requirements.

A second class of services provides more specialized computational and storage functionality. For such services, support for user-specific optimizations is more important. This suggests the use of a centralized approach to service composition and resource allocation. We developed the *service synthesizer* to support this class of services. It creates optimized service instances based on an understanding of the user requirements and the conditions in the network. We developed a number of video gateway and transcoding components to evaluate the use of service synthesis. We believe a synthesizer-based approach is promising although our experience suggests that it is again important that the synthesizer can adapt to resource availability and user requirements.

Both the service synthesizer and self-organizing service components need runtime support help in resource management and service deployment. Examples of support functions are network measurements that help in the placement of service components and connectivity support that helps services deal with the presence of NATs and firewalls. The Libra project developed the

Global Network Positioning (GNP) service for measuring network latencies, the Packet Train Rate (PTR) algorithm for estimating available bandwidth, and Address Virtualization Enabling Service (AVES) tool for providing universal connectivity.

The Libra project relies heavily on active networking technology to meet its goals. Active networking is used in two crucial ways. First, we need the ability to deploy service components in the network as needed. New service instances may for example need to be deployed to satisfy increased load. More importantly, new instances of service components often need to be placed "close" to the user to meet end-to-end latency or bandwidth requirements. Second, we use active networking technology to customize generic service components so they better support the needs of higher level services or end-users. The idea is that base service functions as an execution environment that can host active applications that implement user or domain-specific functionality, e.g. prioritization of traffic or matching operations in publish-subscribe systems. Customization is especially important for generic low-level services.

2 Introduction

The Internet is continuing to grow at a very rapid pace, both in terms of the number of users and network throughput. At the same time, it is trying to evolve from a best-effort network that moves data files to a sophisticated infrastructure that delivers value-added services such as video conferencing, virtual reality games, and distributed simulation. These two concurrent trends are creating significant challenges in both the infrastructure and electronic services areas. A first question is how to develop a network infrastructure that is not only highly scalable, but that at the same time has sufficient functionalities such as QoS and multicast to support the envisioned services. For electronic services, the main challenge is complexity: value-added services are large distributed programs that have to execute in an unpredictable runtime environment (the Internet), raising questions such as how to meet end-to-end service properties.

To deal with these challenges, providers currently deploy *vertically-integrated* services, i.e. the service provider decides on the required service functionality and develops a tightly integrated system that delivers the service to users. In many cases the service is based on a client-server architecture, since it is easy to set up and manage. For example, a "video conferencing service" would be set up as a set of servers running a server application that includes all the service functionality: video multicasting, transcoding, stream thinning, and translating between session standards (e.g. SIP [27] and H.323 [33]). While there are some exceptions to this design (most notably the Web and peer-to-peer services such as Gnutella), they tend to be fairly specialized infrastructures that focus on a specific task (e.g. file sharing) and their design does not generalize to other services.

Such a vertically-integrated service model has a number of disadvantages. First of all, it is not very flexible. For example, adding new functionality to satisfy a new user will require a non-trivial extension of the integrated service code, or the creation of specialized versions of the service for that user. Secondly, reusability or interoperability are difficult since the service is developed as a stand alone system. As a result, it is typically not possible to use the service as a component of a richer value-added service, and service providers often have to "reinvent the wheel" instead of reusing existing service functionality. Third, it is difficult to use advanced network features such as multicast and Quality of Service (QoS). The reason is that these features are not universally available or come in different forms, and it is difficult for a vertically-integrated service to handle such diversity. Being restricted to using the smallest common denominator network feature hampers the service provider's ability to develop sophisticated services quickly.

The Libra[1] project explored a fundamentally different way of developing and deploying network services. Instead of vertically-integrated services, we propose to support the development of new services through composition of existing, primitive services. For example, a distance learning service could build on existing video conferencing and content distribution services. This approach is very flexible since it allows new services to be defined in a variety of ways and allows existing services to be modified and extended by replacing some components. Second, it automatically supports code reuse. Finally, advanced network features can be used by using components that are optimized to exploit specific technologies that may be available in some parts of the network.

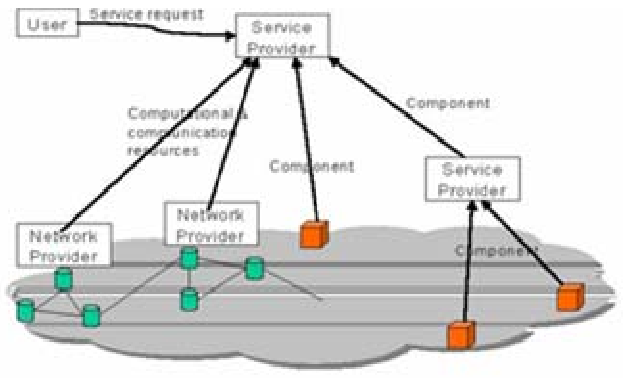


Figure 1: Entities in the service model

While a component-based approach has many advantages, it has the drawback that the resulting services can be very generic since that are based on generic components. The active networking research community has been developing technology that allows the rapid deployment of new functionality, such as data processing or control protocols, by dynamically inserting mobile code segments into the network. The Libra project combines active networking technology with a component based approach to deliver services that are optimized to meet specific user requirements. Active networking is used in two ways. First, we need the ability to deploy service components in the network as needed; new instances of service components may fore example need to be placed "close" to the user to meet end-to-end latency requirements. Second, we use active networking technology to customize generic service components so they better support the needs of higher level services or end-users.

The Libra project developed a set of mechanisms and tools to manage a set of rich composable network services in a scalable fashion. These components were developed in the context of a set of example services including video conferencing, broadcasting, and distributed games. In the process, we had to address a number of fundamental questions. For example, should services be

[1] The name "Libra" refers to the fact that we try balance rich functionality on one hand and scalability on the other hand.

managed in centralized, distributed, or in some hybrid fashion? To what degree can the service infrastructure be shared by multiple services or in other words, how much of the services infrastructure must be service specific? Finally, how can we strike a balance between the convenience of using generic components versus the benefits of using components that have been specialized for specific users.

3 Methods, Assumptions, and Procedures

We describe the context in which we expect services to be deployed and the service composition process. We also present the research approach used in Libra.

3.1 The Players

We assume a network service model in which "providers" fall in two classes (Figure 1). *Network providers* provide the resources necessary to deliver services, i.e. bandwidth on network links, computing cycles and memory on network nodes, and possibly specialized devices. *Service providers* (or value-added service providers in [12]) deliver more advanced services such as distance learning or backup services to end-users. Such value-added services are built by combining a set of service components and by executing them on a set of resources that is leased from an network provider. Service components can be implemented internally by the service provider or can be provided by other service providers.

This model assumes that network services will be developed and delivered in a competitive market. Service providers get paid by its customers (end-users and higher-level service providers) for the value-added services they deliver; network providers get revenue from the users of their infrastructure (service providers and possibly end-users). As in any competitive market, all the providers will want to be able to differentiate their product (resources/services) from that of their competitors', and they will want to bring services to the market faster. Note that in practice the distinction between network providers and service providers may not be that clear. Service providers may own some communication and computational resources, and similarly, network providers may deliver some value-added services.

3.2 Service Composition Process

Hierarchically-synthesized services go through three stages in their lifetime. The first phase is the *design and implementation phase*. During this phase, the service provider determines the specification of the service and what components and resources may be needed to meet the different users' requirements. Some components may have to be implemented by the service provider, and other components/resources will be provided by other service providers and network providers. The outcome of this phase is a service recipe describing, for a specific type of user requests, what components/resources are needed and how they fit together.

The second phase is the *deployment phase*. During this phase, the service provider needs to make sure that the necessary components and resources will be available at runtime. The service provider may decide specifically what suppliers will be used for the different components / resources (note that some may have multiple suppliers), or the service provider can maintain a directory of available suppliers for components and resources so that such information can be used at execution time for service composition. The service provider can also arrange a provider that provides a "service discovery" service to provide such information at execution time.

The third phase is the *execution phase*. During this phase, customers submit requests to the service provider, and the synthesizer use the service recipe and information on components and resources availability to decide how to best satisfy the requests. One of the advantages of the hierarchically-synthesized approach over the vertically-integrated approach is that more decisions are made at runtime, allowing the service to be more adaptive and flexible, e.g. adapt to network load and address specific customer requirements. The execution phase is the focus of the Libra project.

3.3 Research Approach

Our approach to tackle the challenges associated with the deployment and execution of composable, active services was to develop a set of sample service components that provide a context for the development of tools and runtime support. The service components fall in two categories. First, we built a small number of high functionality components, e.g. a video gateway and transcoders, that can be used to create end user services. Second, we developed a set of scalable communication services, e.g. end-system multicast and publish-subscribe services, that support different communication paradigms between components.

These components provide a context to explore a number of mechanisms and tools in support of the management of large scale composable services:

- *Self-organization* as a way of managing large scale communication services in a purely distributed fashion.

- A *service synthesizer* that can compose optimized service instances for clients using a combination of components.

- *Service customization*, which is a particular application of active networking, as a means of customizing both high functionality and scalable communication components to meet the needs of specific services.

- A *service runtime infrastructure* that provides support for network measurements and for universal service access. This infrastructure is shared by self-organizing services and service synthesizers.

The project also included a second task on the application of active networking to congestion control. This task was headed by Srini Seshan.

4 Results

In this section we start with a motivating example and a description of our service architecture. We then summarize our results.

4.1 An Example: Video Conferencing

Let us use a video conferencing service as an example to illustrate how *service composition* can be used to provide flexible and sophisticated network services more easily than using a vertically-integrated approach. Consider the following scenario (Figure 2): *Suppose Alice, who is at W University, wants to hold a video conference that includes two participants at W University, two participants at X Corporation, two participants at Y Corporation, and one participant at Z Airport. The participants at W and X all use the vic/SDR applications, but W and X are not in the same IP multicast zone (i.e. participants at W cannot reach participants at X using IP multicast). The participants at Y are using the NetMeeting application, and the participant at Z is using a receive-only handheld device (which does not do protocol negotiation).*

Supporting this scenario is difficult because of the heterogeneity of the systems involved. For example, different participants are using different video conferencing tools (NetMeeting [40] versus the MBone tools vic [55] and SDR [47]) that use different protocols (H.323 [33] versus SIP [27]). Similarly at the network layer, IP multicast is not universally supported, and while application-level multicast alternatives exist, they are more difficult to set up. For this reason, video conferencing services today typically require users to use specific software. For example, the service may set up an H.323 server and will require all participants to use, for example, Net-Meeting. Alternatively, it may require that all participants use the MBone tools and are connected to MBone [39]. Obviously, this solution is not very convenient.

The challenge in supporting this scenario is not that the software does not exist. We developed a prototype service that supports the scenario of Figure 2 using the following existing software packages: (1) a SIP/H.323-translation gateway [28], which helps vic/SDR and NetMeeting applications establish a joint session by translating the protocol negotiations (and also handles

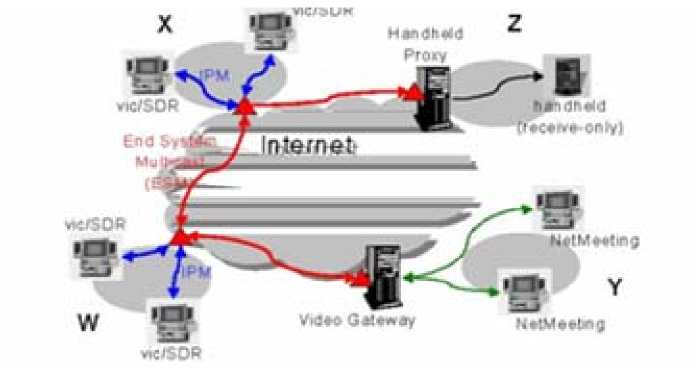


Figure 2: A video conferencing scenario

the video demultiplexing for Net Meeting users); (2) End System Multicast (ESM) [16,15] proxies, which implement multicast functionality using an overlay over a unicast- only network; and (3) a hand held proxy that performs protocol negotiation on behalf of the hand held device and forwards the video streams, optionally performing transcoding to reduce bandwidth consumption. We also make use of vic/SDR and NetMeeting clients and IP multicast. This video conferencing service, using many of the technologies described in the rest of this section, was demonstrated at the 2002 DARPA Active Networks Conference and Exposition (DANCE'02) meeting, which was held in May 29-30, 2002 in San Francisco. The challenges are instead in putting the existing software together: (1) how can we get these separately- developed packages to work together to deliver a higher- level service, (2) how can we manage the deployment of the components in the infrastructure so that the video conferencing service can be efficient and of high quality (to the user), and (3) how can we automate the service composition process so user request can be handled efficiently without manual intervention. In the remainder of this report, we will describe how an architecture based on hierarchical composition of active service components addresses these issues.

4.2 Service Deployment Architecture

The problem of deploying can be broken up in two pieces. First, how do we provide rich service functionality, i.e. how do we manage the service code so we can best meet the diverse requirements of end users. Second, how do we manage the server and network resources that execute the service code. Developing and deploying software using reusable software components is a widely used technique for building large complex software systems. The simplest example is the use of libraries in sequential programs while Corba [6] applies this idea to the distributed context by organizing software systems as a set of distributed self-contained objects. When applying this idea to Internet scalenet work services, hierarchical service composition is a very natural solution. Figure 3 shows for example how a distance learning application could be composed of more primitive services: each service combines its own functionality with that of one or more child services to support its clients (parents). The second problem corresponds to the traditional problem of distributed resource management. Services need both "point" resources, e.g. CPUs and storage, to execute service components and communication resources that connect the point resources with paths with appropriate properties,

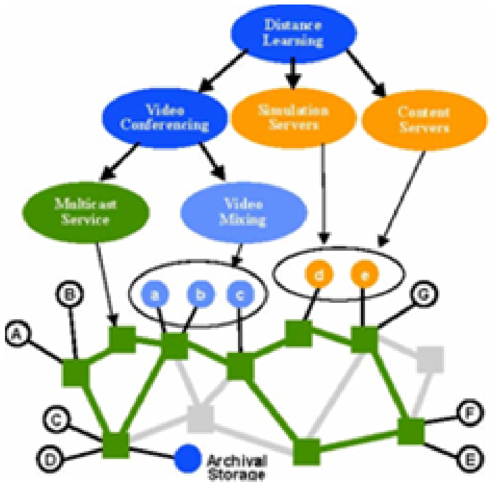


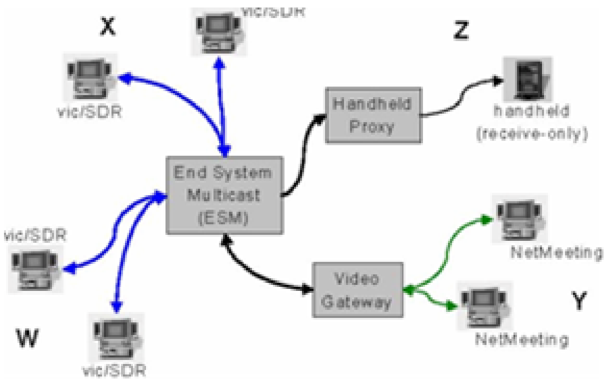
Figure 3: Hierarchical service composition example: distance learning

e.g. bandwidth and latency. Centralized solutions have the advantage that they support aggressive optimization of both individual service requests and overall resource use, but they do not scale. Distributed solutions are much more scalable, but they limit the type of optimizations that can be performed. This conflict between scalability and richness of the optimization is fundamental and traditional heuristic solutions tend to compromise one or the other property.

The hierarchical service composition model offers an attractive way of achieving both scalability and rich optimization by using a hybrid resource management solution. The idea is that each service component can independently select its resource management strategy. We will use the video conferencing service as an example to illustrate the hybrid resource management strategy using Libra:

- High-level services will in general be fairly specialized, lending themselves to user-specific optimizations. This suggests the use of a centralized approach to service composition and resource allocation. We developed the *service synthesizer* to support this class of services. It creates optimized service instances based on an understanding of the user requirements and the conditions in the network. Active networking technology is used to deploy service components on-the-fly in the right part of the network, e.g. deploying a video transcoder near the clients.
- Low-level services will in general, be more generic and will be shared by many high level services. As a result, scalability is a key concern. Such services will in general rely on distributed self-organization. To explore this service class, we developed a set of scalable communication services. Active networking technology is used both to deploy the service and to customize its operation.

In the remainder of this section, we summarize our work in the following areas: service synthesis, network-sensitive service discovery, scalable multicast and publish-subscribe communication services, the service support infrastructure, service customization, and congestion control.



(a) Abstract solution

Figure 4: Synthesizer steps for the video conferencing scenario

4.3 Service synthesis

The service synthesizer creates optimized service instances for incoming user requests. More specifically, it performs the following tasks at runtime: (1) The synthesizer receives the user's service request, which specifies the desired service. For example, for the video conferencing service, the user request will specify the conference participants, the conferencing applications used by the participants, and so on. (2) The synthesizer generates one or more *abstract solutions* for the user according to the service provider's service recipe (Figure 4(a)). An abstract solution describes one possible combination of components and resources and how they should be put together so that the user's requirements can be satisfied. (3) The synthesizer then needs to "realize" the abstract solutions by finding the actual components in the network, i.e. it "binds" each of the abstract components to an actual component in the network (Figure 4(b)). For example, if the abstract solution specifies "a video transcoder is needed", then the synthesizer needs to locate a physical machine that is running the transcoding software. An actual component can be an existing service owned by the service provider, a service provided by another service provider, or a newly instantiated service on an available computation node (possibly provided by a network provider). In addition, there will typically be many possible realizations of the abstract solutions possible, corresponding to a set of *feasible solutions*. The synthesizer has to use some optimization criteria to pick the "best" binding, considering both the service quality for the user and the efficient use of the infrastructure (cost). (4) Finally, the synthesizer needs to configure the components to actually start the service instance. How to configure the components is specified in the service recipe. For example, in the video conferencing service, the synthesizer needs to instruct the transcoding service to establish connections to appropriate video source and sink.

We can see that the functionalities needed to perform task (3) are fairly *generic*, for example, a component discovery infrastructure can be shared by many synthesizers. Therefore, in our synthesizer architecture, we implement these generic functionalities as libraries that can be used by different service providers to implement their synthesizers. As a result, service providers can concentrate on developing their service recipes, which specifies how to perform the service-specific tasks (2) and (4). Note that other similarly observed the benefits of service composition, e.g. Panda [45] and Ninja [25]. However, these efforts focused on a path-based service model while our approach can be applied to more general multi-point services.

The service recipe can be implemented in a number of ways. One option is realize the service recipe as service-specific code. This gives the service provider a lot of flexibility, but can be

Ingredient:

- Handheld proxy: if some participants use receive-only handheld devices, use a handheld proxy for each of them.
- SIP/H.323-translation gateway
- End system multicast (ESM) service: ask the ESM service to establish an ESM session among the vic/SDR endpoints, the gateway, and the handheld proxies. (The ESM service returns the data path entry points for those in the ESM session.)

Instruction:

1. Give the gateway the list of NetMeeting endpoints and the list of vic/SDR endpoints and handheld proxies (along with their data path entry points) so that the gateway can call the endpoints to establish the conference session.

Optimization:

1. Use the generic optimization support

Figure 5: A service recipe for the video conferencing service

Ingredient:

- ESM proxy: figure out whether the participants are in the same IP multicast zone, and use an ESM proxy for each participant of the ESM session. (All participants in the same IP multicast zone can share the same proxy.)

Instruction:

1. For each ESM proxy, choose a multicast address and a port which will be used by the ESM proxy to communicate with its associated participant (or participants in the associated zone). This is the data path entry point for the participant(s).

2. Set up the ESM session among the proxies.

3. Return the list of data path entry points to the entity that requests for this ESM session.

Optimization:

1. Use the generic optimization support

Figure 6: A service recipe for the end system multicast (ESM) service

complex. An alternative is to use a declarative language; this may simplify the definition of new services but may limit flexibility. In either case, the recipe will have to provide three types of information: the service components (*ingredients*) that are used to generate the abstract solutions (task 2), a set of *instructions* to configures the components (task 4), and the *optimization* directives.

To illustrate what a service recipe looks like, Figure 5 sketches (in English) a service recipe for the video conferencing service (the actual service recipe is the video conferencing synthesizer code). This recipe also illustrates that a component of a service can in fact be a service provided by another service provider. In this recipe, the end system multicast (ESM) component is a service provided by an ESM service provider. Therefore, the ESM service provider will implement an ESM synthesizer using the ESM service recipe in Figure 6, for example.

Of the three parts of the recipe, the optimization section is the least well understood. The reason is that the optimization problem is a difficult problem: it involves solving a hard problem (NP-complete) over a potentially very large number of services and resources. We decided to use a heuristic that breaks up the problem in two parts. The first step is to identify for each service component a limited number of candidate servers that can provide the required functionality and that are likely to have a right network connectivity properties; this step is performed by the network-sensitive service discovery (NSSD) service, which is described in the next section. The second step takes the candidates identified for each component by NSSD and the optimization criteria provided by the service provider and solves a global optimization problem. Algorithm of choice depends on the problem size: for small requests, exhaustive search works best, while large problems can be solved using commercial packages such as Cplex [32].

4.4 Network-Sensitive Service Discovery (NSSD)

We consider the problem of network-sensitive service selection (NSSS): finding services that match a particular set of functional and network properties. Current solutions handle this problem using a two-step process. First, a user obtains a list of candidates through service discovery. Then, the user applies a network-sensitive server selection technique to find the best service. Such approaches require each application to implement its own selection technique, which may not be practical. Moreover, the overhead of service discovery and network-sensitive selection may be high, since each user tries to solve the NSSS problem independently.

The Libra project developed a simple alternative, namely network-sensitive service discovery (NSSD) [32]. By combining network-sensitive server selection with the service discovery process, NSSD allows users who are looking for services to specify both the desired functional and network properties at the same time. Users and providers can exploit the benefits of network-awareness without implementing their own selection techniques, and NSSD can solve the NSSS problem more efficiently by amortizing overheads over many users.

4.4.1 NSSD API

Before we can define our API for NSSD, we need to determine what functionalities NSSD should provide, and what should be left to users. The server selection problem can be formulated as an optimization problem: finding a solution (e.g., a game server) that optimizes a certain metric (e.g., the maximum latency) for a set of targets (e.g., the players). A solution may be a single server (as in the game example) or a set of servers (ESM example). Moreover, in some scenarios, solving a global optimization problem (e.g., find a streaming server and a transcoder such that the total bandwidth usage is minimized) is better than combining the solutions of local optimization problems (e.g., find the streaming server closest to the user and then find the transcoder closest to the selected streaming server).

Since it is not feasible to support all optimizations imaginable, our goal in defining the API is the following: users should be able to easily specify simple and common optimizations using the API, and the API should provide enough flexibility so that it is possible for users to perform their own service-specific and possibly complex optimizations if necessary. The goals led to the following design decisions:

- *Local optimization only*: Our API only allows users to specify local optimization problems, i.e., a user can only query for one service at a time. The reason for this decision is that service-specific global optimizations may be arbitrarily complex. Therefore, it may be very difficult to specify such problems. Also, solving global optimization problems in NSSD may be infeasible due to computational complexity.

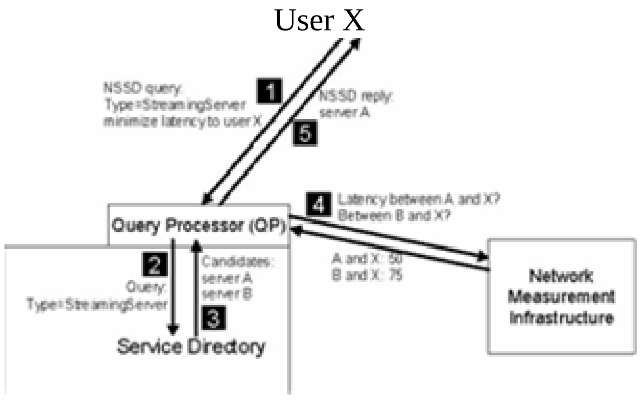


Figure 7: Handling an NSSD query

- *Provide a set of standard metrics*: Our API allows a user to specify constraints and preferences on a set of standard metrics: maximum and average latency, minimum and average bandwidth, and server load. However, we do not allow, for example, user-defined utility functions, since they can make the API too complex.

- *Best-n-solutions*: The first two decisions define a set of standard network-sensitive selection techniques provided by NSSD. While we believe this set is sufficient for many users and services, it cannot cover all possible needs. Since we want to let users perform their own more complicated optimizations, we make our API flexible by allowing a user to ask for the best solutions. Therefore, for example, a user can ask for all streaming servers (and the closest transcoder to each of them) to compute the globally optimal solution if absolute optimality is required. Furthermore, this allows users to control the trade-off between optimality and complexity (for example, for another user it may be good enough to find the best solution among the closest three streaming servers).

The detailed API is presented elsewhere [32]. An evaluation based on Planetlab measurement and simulation shows that API allows providers to implement service-specific optimization efficiently. The reason is that the main challenge (dealing with large numbers of resources) is handled by NSSD.

4.4.2 Query processing

Given the NSSD API described above, NSSD queries can be resolved in many different ways. In this section, we first describe a simple approach that heavily leverages earlier work in service discovery and network measurement; this is also the design used in our prototype. We mention alternative approaches in the next section.

As shown in Figure 7, a simple NSSD query processor (QP) can be built on top of a service discovery infrastructure (e.g., the Service Location Protocol [26]) and a network measurement infrastructure that can estimate the network properties (e.g., latency) between nodes. One possibility is to integrate the functionality of QP into the service discovery infrastructure, i.e., it is a module that extends the functionality of a traditional service directory.

Let us describe how NSSD queries are handled. When the QP module receives an NSSD query (step 1 in Figure 7), it forwards the functional part of the query to the service directory (step 2). The directory returns a list of candidates that match the functional properties specified in the query (step 3). Then the QP module retrieves the necessary network information (e.g., the latency between each of the candidate and user X) from the network measurement infrastructure (step 4). Finally, the QP module computes the best solution as described below and returns it to the user (step 5).

One benefit of integrating service discovery and server selection in a single service is that caching can be used to improve performance and scalability. When NSSD gets requests from many users, the cost of collecting network status information or server load information can be amortized. Furthermore, network nodes that are close to each other should have similar network properties. Therefore, nodes can be aggregated to reduce the amount of information required (for example, use the same latency for all users in an address prefix). This should improve the effectiveness of caching.

4.4.3 Alternative approaches

Now let us look at some alternative approaches that can potentially be used to handle NSSD queries. [21] Describes how a hash-based overlay network mechanism (such as Chord [52]) can be used to discover content described as a set of attribute-value pairs (see also Section 4.7). Such a system can form the basis of a service discovery infrastructure by describing each service as a set of attributes that describe its functionality and network properties. Network awareness is introduced in the query resolution phase of the system so that the returned matches satisfy certain network properties specified in the query.

Another alternative is application-layer any casting [56], in which each service is represented by any cast domain name (ADN). A user submits an ADN along with a server selection filter (which specifies the selection criteria) to any cast resolver, which resolves the ADN list of IP addresses and selects one (or more) from the list using the filter. Potentially, the ADN and resolvers can be extended to allow users to specify the desired service attributes, and the filter can be generalized to support more general metrics.

Finally, distributed routing algorithms are highly scalable, and they can, for example, be used to find a path that satisfies certain network properties and also includes a server with certain available computational resources [31]. A generalization of this approach can be combined with a service discovery mechanism to handle NSSD queries.

4.5 ESM

Traditional network architectures distinguish between two types of entities: end systems (hosts) and the network (switches and routers). One of the most important architectural decisions is then the division of functionality between end systems and networks. In the Internet architecture, the internetworking layer, or IP, implements a minimal communication unicast and multicast service, while end systems implement all other important functionality such as error, congestion, and flow control. While this minimalist approach to communication services is probably the single most important technical reason for the Internet’s growth, it places severe constraints on the richness of the service. The Libra project explored an alternative approach to multicast.

4.5.1 Architecture

In his seminal work in 1989 [18], Deering argues that multicast should be implemented at the IP layer. This view has been widely accepted and most routers today implement IP Multicast. However, IP Multicast has several drawbacks that have so far prevented the service from being widely deployed. First, IP Multicast requires routers to maintain per group state, which not only violates the “stateless” architectural principle of the original design, but also introduces high complexity and serious scaling constraints at the IP layer. Second, the current IP Multicast model allows for an arbitrary source to send data to an arbitrary group. This makes the network vulnerable to flooding attacks by malicious sources, and complicates network management and provisioning. Third, IP Multicast requires every group to dynamically obtain a globally unique address from the multicast address space and it is difficult to ensure this in a scalable, distributed and consistent fashion. Finally, IP Multicast is a best effort service. Providing higher level features such as reliability, congestion control, flow control, and security has been shown to be more difficult than in the unicast case.

All these drawbacks are a direct result of the fact that key features of the multicast protocol are fixed, yet even these simple fixed features stress the capabilities of traditional routers. In light of these issues, the Libra Project studied a fundamentally different approach to implement multicast functionality. In particular, we consider a model in which multicast related features, such as group membership, multicast routing and packet duplication, are implemented at end systems, assuming only *unicast* IP service. We call the scheme End System Multicast (ESM). End-systems can be either desk-top machines or programmable network nodes that support multicast as a high-level service. Here, end systems participating in the multicast group communicate via an overlay structure that is based on unicast paths. The overlay is self-organizing, i.e. it selects in a distributed fashion what set of resources (communication links and ESM nodes) will best satisfy the clients.

We believe that End System Multicast can address most problems associated with IP Multicast. Since all packets are transmitted as unicast packets, network provisioning is not affected and de-ployment may be accelerated. End System Multicast maintains the stateless nature of the network by requiring end systems, which subscribe only to a small number of groups, to perform additional complex processing for any given group. In addition, we believe that solutions for supporting higher layer features such as error, flow, and congestion control can be significantly simplified by leveraging the capabilities of the ESM end systems. Finally, an end system based architecture no longer requires global consistency in naming of groups and allows for application specific naming. The last two features directly exploit the programmability of the ESM nodes, which allows application specific solutions to be implemented.

To understand the fundamental technical challenges in the ESM architecture and devise effective solutions for these problems, we have conducted several studies. First, we have designed basic overlay network construction techniques for ESM and demonstrated the basic viability of the ESM approach. Second, we have demonstrated the suitability of ESM for multimedia conferencing applications in the dynamic and heterogeneous commercial Internet. Third, we have designed light-weight measurement-based algorithms for optimizing ESM's performance. Finally, we have implemented and deployed an Inter live broadcasting toolkit based on the End System Multicast architecture. The system is seamlessly integrated with commercial audio and video software (e.g. Apple Quicktime Broadcaster and Player). The system was part of the Libra demo presented at the *2002 DARPA Active Networks Conference and Exposition* meeting and it has also been used to live broadcast numerous CMU distinguished lectures and university events, and the ACM SIGCOMM 2002 and 2003 conferences. Overall, the system has already been used by over 2000 clients.

In the following, we summarize the contributions of the ESM research.

4.5.2 The Case for End System Multicast

A key challenge in the ESM approach to multicast is the distributed resource management problem. The topology of the overlay must be chosen so it uses resource efficiently (e.g. we should minimize duplicate packets on physical links) while also optimizing user performance (e.g. minimize end-to-end delay). We have studied these questions in the context of a protocol that we have developed called *Narada*. Narada constructs an overlay structure among participating end systems in a *self-organizing* and *fully distributed* manner. Narada is robust to the failure of end systems and to dynamic changes in group membership. End systems begin with no knowledge of the underlying physical topology, and they determine latencies to other end systems by probing them in a controlled fashion. Narada continually refines the overlay structure as more probe information is available. Narada may be distinguished from many other self-organizing protocols in that it does not require a native multicast medium.

We evaluate the performance penalty of the overlay Narada produces using simulations. In a group of 128 members, the delay between at least 90% of pairs of members increases by a factor of at most 4 compared to the unicast delay between them. Further, no physical link carries more than 9 identical copies of a given packet. We have also implemented Narada and conducted preliminary Internet experiments. For a group of 13 members, the delay between at least 90% of pairs of members increases by a factor of at most 1.5 compared to the unicast delay between them. These results show that the performance penalty of implementing multicast functionality in end systems is not very high and thus ESM is a very viable approach.

4.5.3 Light-Weight Measurement-Based Optimization

One of the challenges associated with distributed resource management dynamic adaptivity to changing conditions. Narada uses a measurement-based communication-peer selection strategy to improve the performance of bandwidth-demanding overlay systems. Traditionally, such peer selection is based on basic light-weight measurement-based techniques such as RTT probing, 10KB TCP probing, and bottleneck bandwidth probing may work in practice in the peer-to-peer environment. By conducting trace-based analyses on over 10,000 end hosts in the Internet, we find that the basic techniques can help achieve 40 to 50% optimal performance. To deepen our understanding, we analyze some of the intrinsic properties of these techniques. Our analyses reveal the inherent difficulty of the peer selection problem due to the extreme heterogeneity in the peer-to-peer environment, and that the basic techniques are limited because their primary strength lies in eliminating the low-performance peers rather than reliably identifying the best-performing one.

However, our analyses also reveal two key insights that can potentially be exploited by applications. First, for adaptive applications that can continuously change communication peers, the basic techniques are highly effective in guiding the adaptation process. In our experiments, typically an 80% optimal peer can be found by trying less than 5 candidates. Secondly, we find that the basic techniques are highly complementary and can potentially be combined to better identify a high-performance peer, thus even applications that cannot adapt may benefit. Using media file sharing and End System Multicast streaming as case studies, we have systematically experimented with several simple combined peer selection techniques. Our results show that for the non-adaptive media file sharing application, a simple combined technique can boost performance to 60% optimal. In contrast, for the continuously adaptive End System Multicast application, we find that a basic technique with even low-fidelity network information is sufficient to ensure good performance. We believe our findings will help guide the future designs of high-performance peer-to-peer systems.

4.5.4 Using End System Multicast in the Internet

While our preliminary simulation results conducted in static environments show that ESM is very promising, we have yet to consider the challenging performance requirements of real world applications in a dynamic and heterogeneous Internet environment. To address this concern, we have explored how Internet environments and application requirements can influence End System Multicast design. We explore these issues in the context of audio and video conferencing: an important class of applications with stringent real-time performance requirements. We have conducted an extensive evaluation study of schemes for constructing overlay networks on a wide-area test-bed of about twenty hosts distributed around the Internet.

Our results demonstrate that in order to meet the performance requirements of audio and video conferencing, it is necessary for self-organizing protocols to adapt to both latency and bandwidth metrics. We have devised techniques by which such protocols can adapt to dynamic metrics like available bandwidth and latency, and yet remain resilient to network noise and inaccuracies inherent in the measurement of these quantities. We demonstrate our ideas by incorporating them into the Narada self-organization protocol.

We have evaluated our techniques by testing the redesigned Narada protocol on a wide-area test-bed. Our test-bed comprises twenty machines that are distributed around North America, Asia and Europe. Our results demonstrate that our techniques can provide good performance, both from the application perspective and from the network perspective. With our scheme, the end-to-end bandwidth and latency attained by each receiver along the overlay is comparable to the bandwidth and latency of the unicast path from the source to that receiver. Further, when our techniques are incorporated into Narada, applications can see improvements of over 30–40% in both throughput, and latency. Finally, the costs of our approach can be restricted to 10–15% for groups of up to twenty members. Our results indicate that End System Multicast is a promising architecture for enabling performance-demanding conferencing applications in a dynamic and heterogeneous Internet environment.

4.6 Measurement Infrastructure

Managing and deploying services requires information about the performance properties of the network. Moreover, services typically assume universal connectivity, which does not exist in practice because of the wide spread use of NATs. The Libra project developed two network measurement tools, one estimating latency (GNP) and one estimating available bandwidth (PTR), and the AVES tool for providing connectivity across address spaces.

4.6.1 Global Network Positioning

Achieving high performance is a key challenge in building large-scale globally-distributed network services and applications such as distributed content hosting services, overlay network multicast, content addressable overlay networks, and peer-to-peer file distribution. We believe a promising approach to achieve high performance is to use network distance (i.e., round-trip propagation and transmission delay, a relatively stable characteristic) between hosts as a first-order metric to optimize application performance. Global Network Positioning (GNP) is a solution designed to predict network distance accurately with little network measurements [42]. The key idea is to transform the complex structure of the Internet into a simple geometric space (e.g. an N-dimensional Euclidean space) representation based on a small amount of network measurements to several Landmark hosts in the Internet. In this representation, each host in the Internet is characterized by its position in the geometric space with a set of geometric coordinates. If the representation is accurate, then the easily computable geometric distances between hosts in this geometric space can accurately approximate the Internet network distances, and no actual network measurements are required.

In extensive Internet experiments, we have found that by using a 7-dimensional Euclidean space, in 90 among a globally distributed set of hosts with less than 50 which is the best among all known solutions. Key distinguishing properties of GNP are (1) peer-to-peer architecture friendly, (2) highest prediction accuracy among known algorithms, (3) extremely fast, (4) highly scalable, (5) geometric representation directly applicable in many applications.

The GNP measurement infrastructure has been used by a number of systems, including ESM and NSSD.

4.6.2 Packet Train Rate Probing

Many applications are more sensitive to bandwidth than latency, so it is important to have bandwidth estimates when placing services in the network. Until recently, the tools for estimating bandwidth were very crude. Typically, applications use a TCP session of about 5-15 seconds to get an estimate on the available bandwidth between two nodes. This is clearly an expensive solution.

We developed a more light-weight solution, called the Packet Train Rate (PTR) method [29]. It is based on the observation that when we send a packet train of back to back packets over an Internet path to a destination, then competing traffic on bottleneck router will separate the packets, so the inter-packet gaps measured at the destination will be larger than at the source. However, when we send a packet train with large inter-packet gaps, the packets will travel through the network independently, and the inter-packet gap does not change (on average). The smallest inter-packet gap for which a packet train can travel along a path without having its inter-packet gap increased by competing traffic is called the turning point and it has the property that consumes all the unused bandwidth on the bottleneck link.

The PTR method uses a sequence of packet trains with different inter-packet gaps to experimentally identify the turning point. It then uses the rate of the train at the turning point as an estimate of the available bandwidth. PTR turns out to be quite effective. A set of Internet experiments shows that on most network paths, PTR can get a good estimate of the available bandwidth in about 6 roundtrip times using 16-packet trains [29]. This is significantly faster than other tools such as Pathload [34].

PTR is a very promising probing technique that can be used in many applications. For example, we used PTR to improve the Slow Start mechanism that TCP uses to probe for available bandwidth. The resulting algorithm, called Paced Start, is faster and has much lower packet loss than Slow Start [30].

4.6.3 Address Virtualization Enabling Service (AVES) for Incrementally Scaling the Internet Address Space

We have conducted research in providing connectivity across Internet networks of heterogeneous address spaces. The rapid growth of the Internet has made IPv4 addresses a scarce resource. Today we witness two major trends to get around this problem. The first is to upgrade and deploy networks using IPv6; the second is to deploy networks using reusable-IPv4 addresses. As a result, the Internet is rapidly evolving into a collection of networks of heterogeneous address spaces. Such development jeopardizes the universal connectivity property of the Internet.

To solve this problem, we propose a distributed waypoint service called AVES [41]. The key idea is to use IPv4 waypoints (3rd-party network agents) as relays to connect any IPv4 host to an arbitrary set of IPv6 or reusable-IPv4 hosts. As a result, the internetworking heterogeneity is handled by the waypoints and hidden from the existing IPv4 infrastructure, making non-intrusive deployment of AVES possible. Once AVES is deployed, the service is instantaneously accessible transparently by any IPv4 host in the Internet via host name resolution. This approach is unique because it solves the connectivity problem without changing the networking layer of existing systems. We are currently deploying an AVES prototype at CMU to service the community members who have, for example, created home networks with reusable-IPv4 addresses and need universal connectivity for their home computers.

4.7 Publish-subscribe communication services

Publish-subscribe (pubsub) systems are an alternative to traditional communication in which unicast or multicast addresses are used to deliver messages. There are two important aspects to a generic publish-subscribe system: Data Naming (how publications are described), and Subscription Language (how subscribers describe their interests). A key design goal for publish-subscribe systems is to support names and subscriptions that are flexible enough to support a wide variety of applications. However, there are often fundamental tradeoffs that may make such flexibility impractical.

There have been two important types of publish-subscribe systems: subject/channel-based and content-based. In a channel-based system, every publication gets implicitly associated with a channel and a subscription explicitly names the channel the receiver is interested in. A closely-related variant of such systems is when publications and subscriptions are forced to contain specific, a priori chosen subjects. There have been a number of highly-scalable implementations of such systems including systems using IP Multicast [19] and more recent systems using distributed hash tables (DHTs) [46, 9]. Unfortunately, while these approaches are scalable, the relatively restrictive naming and subscription language limit the number of applications that such systems can support.

Content-based publish-subscribe systems support subscriptions that specify any predicate over the entire content of the message. Such content-based pubsub systems provide an application flexibility in framing its queries depending on its needs. However, previously known approaches to implementing these systems suffer scalability problems. Existing systems use one of two designs: 1) Subscriptions and publications are sent to and matched at a centralized server [48], and 2) Subscriptions or publications are flooded to all participants in the network [11, 5]. Each of these implementation styles suffer from a number of obvious scaling flaws.

Based on previous system designs, it appears that there is a fundamental trade-off between the richness of the subscription language and the scalability of the system. The Libra project developed two pubsub systems that explore intermediate points in the richness-scalability spectrum. Both systems use names that are a list attribute-value pairs and they support rich queries, e.g. subset matching and range searches. The first system, Mercury, uses a content-based approach while the second system, Camel, is based on DHTs.

4.7.1 Mercury

Mercury's basic approach is to create a routing hub for each attribute used in subscription queries. Subscriptions are passed to exactly one of the hubs corresponding to the attributes that are queried, while publications are routed to all hubs for which the publication has an associated attribute. Given this structure, the publish-subscribe data delivery can be accomplished by: 1) routing subscriptions and publications to rendezvous points within each hub, 2) matching subscription and publications at these rendezvous points, and 3) delivering the publication from the rendezvous to interested subscribers. The key challenges in this design are routing subscriptions and publications to rendezvous points quickly and efficiently, and distributing load uniformly throughout the network.

Within each routing hub, nodes are organized in a circular fashion much like the Chord DHT [52]. However, there are some key differences from Chord. First, each node in the hub is responsible for a contiguous range of attribute values -unlike Chord, no cryptographic hashes are used to map the data values to keys. This allows for a simple mapping of the range predicates in subscriptions to a collection of contiguous nodes in a hub. Second, since cryptographic hashes are not used, the distribution of values used in the routing is no longer uniform. This forces the Mercury system to: 1) have mechanisms other than finger pointers for providing efficient routing, and 2) have mechanisms for balancing load among the participating nodes. The key to Mercury's design is a novel sampling technique that allows each node to estimate the current distribution of data values.

Each node having an approximation of the data distribution enables a number of interesting techniques within Mercury. First, Mercury uses the data distribution to construct appropriately spaced routing pointers to ensure logarithmic bounds on the number of routing hops. Second, the approximate information allows Mercury to estimate the selectivity of subscriptions among the different hubs. This allows subscriptions to be routed to the most selective hub, minimizing the number of nodes to which the subscription is routed. Our results show that this reduces the average number of nodes involved in a subscription (for our test workloads) by 25-30 random hub selection. Finally, the distribution estimate allows Mercury to identify portions of a hub that are receiving excessive load. Mercury achieves load-balancing by biasing the addition of new nodes towards these regions of the hub and by having lightly loaded nodes periodically leave and rejoin the system. Our results show that Mercury can quickly (within 100 rounds of leaving and rejoining) load-balance nodes despite a heavily skewed workload.

Applications can utilize this publish-subscribe infrastructure using a simple API exported by the Mercury substrate. As a demonstrative application, we have implemented a simple 2-dimensional distributed-interactive simulation-like (DIS) multiplayer game using Mercury. The design of the Mercury system was presented at the Netgames 2002 workshop [3].

Mercury is a useful service that can be implemented on an active networking infrastructure. Active routers in key locations could act as Mercury forwarding agents.

4.7.2 Camel

The Camel Content Discovery System (CDS) [22, 21] uses names that consist of a list of arbitrary attribute-value pairs. Camel is built on top of a distributed hash table, such as Chord [52] or Tapestry [8]. When registering a name, Camel will hash each attribute-value pair individually and register the name with the nodes corresponding to each hash value. This means that a name with N AV-pairs will be registered in N nodes. Queries can be forwarded to the DHT node corresponding to any of the AV-pairs in the query and that node will then do a local lookup on its database. Camel uses a query optimization whereby its sends the query to the node that has the lowest load, i.e. the node corresponding the least popular AV-pair.

By using Rendezvous Points (RP), network-wide message flooding is avoided at both registration and query times. However, in practice, some AV-pairs may be much more common or popular in MFDs than in others. It has for example been observed that the popularity of keywords in Gnutella follows a Zipf-like distribution [35]. Such a distribution will cause a few peers being overloaded by registrations and queries, while the majority of the peers in the system stay underutilized. To improve the system's throughput under skewed load, the CDS deploys a distributed dynamic load balancing mechanism, where multiple peers are used as RP points to share the heavy load incurred by popular AV-pairs. When an AV-pair appears in many registrations, instead of sending all the registrations to one peer, the system partitions them among multiple peers. Similarly, when there are many queries for the same AV-pair, the system allows the original peer who is responsible for this pair to replicate its database to other peers. The partitions and replicas corresponding to one AV-pair are organized in a logical matrix called the Load Balancing Matrix (LBM). This matrix automatically expands and shrinks based on the pairs query and registration load. Note that because of the query optimization mentioned above, an LBM should never have both a large number of partitions and replicas.

This load balancing mechanism allows the system to maintain very high throughput, even under very skewed distributions [21, 22]. An evaluation of the system in the context of a content-based music retrieval system confirmed these results [24, 54].

Active networking plays a key role in Camel. While it is built on top of a generic DHT, the processing performed on the Rendezvous Points is specific to Camel. This code should be downloaded dynamically, as is needed.

4.8 Service Customization

In this section we take a closer look at the different types of Execution Environments (EEs) that have been developed. We then introduce customizable service EEs as a new class of EEs.

4.8.1 Classes of EEs

While there is general EE architecture that defines what support is needed to host an Active Application (AA), there are several ways of deploying an EE in a network. For the purpose of our discussion, we will distinguish between two classes of EEs.

The first class of EEs is characterized by the fact that its primary purpose is to process the packets that flow through AAs hosted by the EE. Examples include ANTs and the PLAN/Switchlets. Per-packet processing ranges from relative simple local operations such as compression or error correction (e.g. [38]) all the way to complex AAs that in effect implement virtual routers or bridges [2]. One property of such EEs is that the AAs that they host have minimal interaction with the rest of the router infrastructure. Their primary interaction with the rest of the router is the exchange of packets, although they may occasionally also collect status information or negotiate with the Node OS for resources. We will call such EEs *overlay EEs* since they typically add network functionality that is quite separate from that of the hosting network infrastructure.

The second class of EEs is characterized by the fact that the AAs they host modify the behavior of the router indirectly. For example, they may update routing tables or change the parameters of packet schedulers. Any packets handled directly by such AAs are typically control packets that may trigger actions by the AA. Examples of such EEs include the Darwin QoS delegates runtime environment [23, 53] and ASP [7]. A key design parameter for such EEs is the API they offer to their AAs [44, 53]. It determines what actions the AAs can take. We will call such EEs *control EEs* since their AAs typically control router functionality.

Figure 8 illustrates the difference between overlay and control EEs. The thick arrows represent the primary dataflow while the thin dashed arrows indicate control operations. APIs are represented by horizontal dashed lines.

Not surprisingly, control EEs will typically be implemented in the control plane of the router. Overlay EEs, on the other hand, are logically part of the data plane, although how they are executed depends on how expensive the AA processing is and on the architecture of the router. On high-end active routers, overlay EEs could execute on dedicated processing resources on the router port cards, e.g. plugins [17], while on low end platforms, slow path data processing may share resources with the control plane.

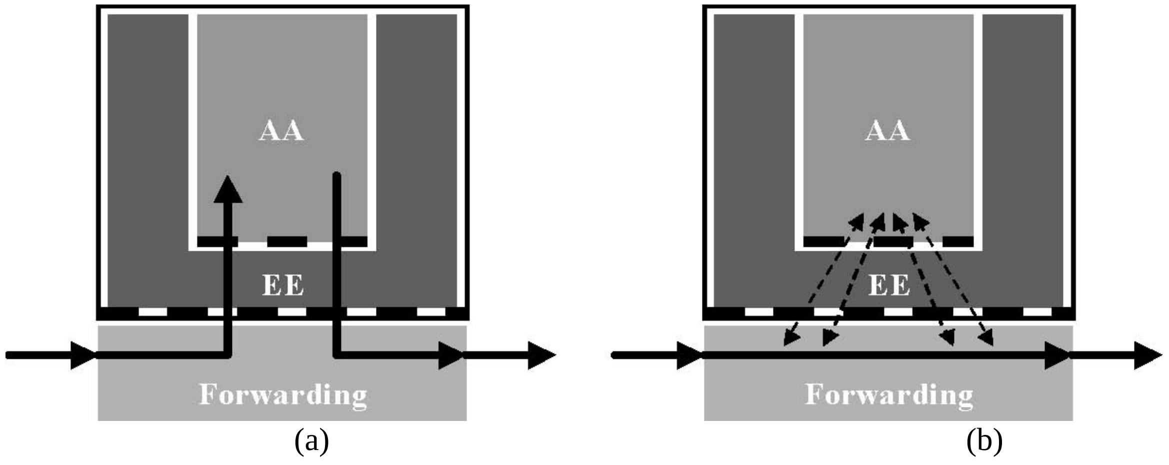


Figure 8: Classes of Execution Environments (EE): (a) overlay EE and (b) control EE

4.8.2 Customization

While building a set of service components for the Libra project, we observed that many components share the following property. While most of the service functionality is generic and is required by all users, some service features can be supported in many different ways. Users want to choose how these features are supported, since it has a big impact on the end-to-end service properties. Let us look at some examples:

- QoS: Many services can benefit from QoS support, for example so they can deliver more predictable services to end-users. However, the details of how, for example, reserved bandwidth should be managed will be different.

- Transcoding: Many users need to be able to translate video format or reduce video resolution. However, the precise formats needed or the required video resolution will differ and may change over time.

- Multicast: Basic multicast support, e.g. delivering a packet to many receivers, is useful for many applications. However, properties such as required reliability of data delivery and practical methods for doing congestion control will be different.

While it is of course possible to implement the service component as a series of AAs, each implementing a slightly different version of the service, this is inefficient since EEs may have to run many copies of very similar code. A more elegant solution is to break the service into two parts: a base service that implements shared functionality, and a customization code module that allows users to “fine tune” the service. The customization module is often very simple, since it only has to extend or modify the existing base functionality. This is a natural solution and this approach is commonly used on end-nodes. For example, many end-user applications such as text editors or spreadsheets provide ways of extending or customizing the capabilities of the application through programs or macros.

The overall structure of a customizable service component is very similar to the EE/AA architecture. The base service combined with a small runtime environment can be viewed as an EE and the customization code is the AA. There are also some significant differences. First, in a customizable service component, most of the functionality is provided by the service EE. In

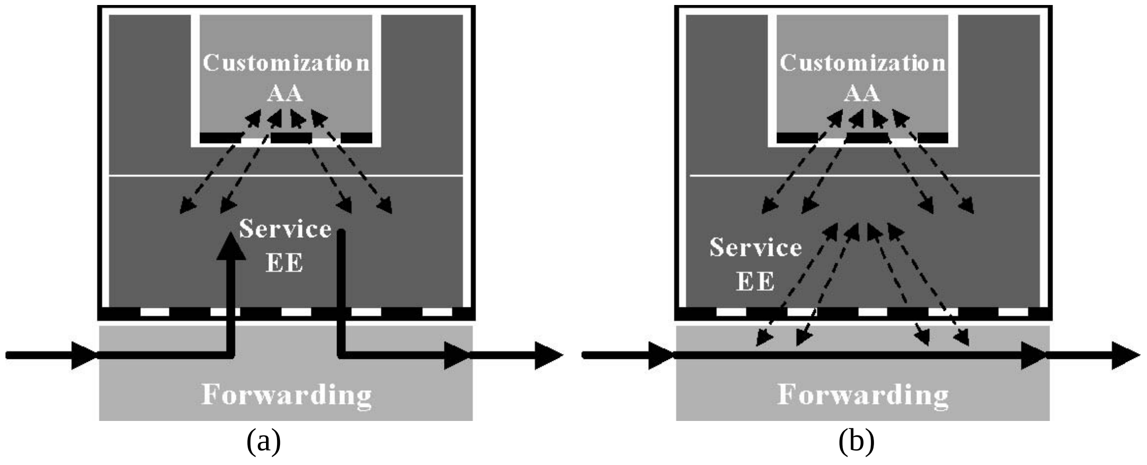


Figure 9: Active Service Customization for (a) overlay EEs and (b) control EEs

contrast, most traditional EEs primarily provide support for AA execution, e.g. language support, downloading and possibly caching of AA code modules, and installing of AAs. Another difference is that the function performed by the customization code is very focused and SPECIFIC to the service being customized. This second difference provides the motivation for including the base service functionality as part of the EE. Viewing just the runtime environment as the EE makes little sense since it has no useful function without the presence of the base functionality of the service. Note that customization applies to both overlay and control EEs, as is illustrated in Figure 9.

A customizable EE has two APIs (dashed lines in Figure 9). First, it has the API that it implements for its AAs. As we discussed above, we expect the API to be fairly narrow and highly service specific, and we provide some examples later in the paper. The second API DEFINES how the EE interacts with the rest of the router. This API will typically correspond to the Node OS API and it is much more flexible and powerful than the first API. This difference in APIs suggests that installing a new customization AA will typically be a very lightweight operation, while installing a new customizable service is more heavy weight. This is not unlike the difference between installing an AA and an EE in, for example, the ABone. Installing a new EE requires special privileges, while any user can install an AA.

While we have introduced the use of customization EEs as a pragmatic solution to the problem of supporting services that differ only in certain features, this approach turns out to have another significant advantage. The API that the customization EE offers to its AAs is typically very focused and simple. This simplifies the security challenges that come with the flexibility of active networking. Specifically, checking the correctness of AA calls is typically simple and very lightweight. We will illustrate this point when discussing examples later in this paper.

Using our definition of customization EEs, a number of other active networking projects have developed EEs that are very similar to our customization design. An example is the Concast effort [10]: the base service is incast (the reverse of multicast), while the specific way of merging messages from the leaves to the root can be customized.

A number of examples of service customization are discussed in detail in [51, 53, 49, 20, 36, 13, 37].

4.9 Congestion control

The objective of our congestion control work was to develop a suite of algorithms that would allow a collection of hosts to collaboratively implement QoS-like bandwidth allocation in the network. Our plan was to use/adapt the Congestion Manager system to implement congestion control algorithms at the endpoints. Such a design requires endpoints to behave in a cooperative fashion. This cooperation can be created either by securing end-points and ensuring that they run the appropriate code (e.g. through secure network adapters or trusted computing techniques or by creating the appropriate incentives to ensure that the end-points behave correctly). Based on these goals, our work explored three topic areas: 1) enhancing the Congestion Manager to make it a practical system for our use, 2) determining the correct end-point congestion control algorithms to use, and 3) developing an understanding of how network designs (and incentives) affect end-point congestion control decisions.

Congestion Manager. The Congestion Manager [4] is one of several recent proposals that have been made for sharing congestion information across concurrent flows between end-systems. In these proposals, the granularity for sharing has ranged from all flows to a common host, to all hosts on a shared LAN. A significant problem with all past designs is that two or more flows sharing congestion state may in fact not share the same bottleneck. We have categorized the origins of this false sharing into two distinct cases: 1) networks with QoS enhancements such as differentiated services, where a flow classifier segregates flows into different queues, and 2) networks with path diversity where different flows to the same destination address are routed differently.

Our results have shown that persistent overload can be avoided with window-based congestion control even for extreme situations of false sharing, but higher bandwidth flows run at a slower rate. We found that delay and reordering statistics can be used to develop robust detectors of false sharing and are superior to those based on loss patterns. We also found that it is markedly easier to detect and react to false sharing than it is to start by isolating flows and merge their congestion state afterwards. The combination of these observations led to the development of effective fixes for the Congestion Manager system. These results will appear at ICNP 2003 [50].

End-point Algorithms. From the early days of modern congestion control, ushered in by the development of TCP's and DECbit's congestion control algorithm and by the pioneering theoretical analysis of Chiu and Jain [14], there has been widespread agreement that linear

Additive-Increase Multiplicative-Decrease (AIMD) control algorithms should be used. However, the early congestion control design decisions were made in a context of fairly primitive loss recovery (e.g., TCP Reno) and router queuing behavior (e.g., FIFO drop-tail). In subsequent years, there has been substantial improvement in TCP's loss recovery schemes (e.g., SACK) and router queuing mechanisms (e.g., RED, ECN). These mechanisms are significantly better at tolerating bursty behavior.

Our work explored the following fundamental questions: Is AIMD the sole choice for congestion control even in these modern settings? If not, can other scheme(s) provide better performance?

We evaluated the four linear congestion control styles -AIMD, AIAD, MIMD, MIAD -in the context of these various loss recovery and router algorithms. Our results showed that while AIMD is an unambiguous choice for the traditional setting of Reno-style loss recovery and FIFO drop-tail routers, it fails to provide the best goodput performance in the more modern settings. Where AIMD fails, we found that AIAD proves to be a reasonable choice.

End-point incentives. For years, the conventional wisdom has been that the continued stability of the Internet depends on the widespread deployment of "socially responsible" congestion control. However, the perception has been that network end-points have little or no incentive to really perform socially and that left to their own desires, end-points would be overly aggressive.

In our work, we explored how the issues such as loss recovery and router behavior affects each flow's attempts to maximize the throughput it achieves by modifying its congestion control behavior. Using a combination of analysis and simulation, we have explored the Nash Equilibrium of this game.

We found that in more traditional environments -where end-points use TCP Reno-style loss recovery and routers use drop-tail queues -the Nash Equilibriums are reasonably efficient. However, when endpoints use more recent variations of TCP (e.g., SACK) and routers employ either RED or drop-tail queues, we found that the Nash Equilibriums are very inefficient. This suggests that the Internet of the past could remain stable in the face of greedy end-user behavior, but the Internet of today is vulnerable to such behavior. Second, we found that restoring the efficiency of the Nash Equilibriums in these settings does not require heavy-weight packet scheduling techniques (e.g., Fair Queuing) but instead can be done with a very simple stateless mechanism based on CHOKe [43]. These results were presented at Sigcomm 2002 [1].

This style of congestion control extends the concept of active networking to the network interfaces of end-hosts. The deployment of our techniques would require both the control and programming of these network interfaces by a bandwidth allocation system.

5 Discussion and Conclusions

The starting point for the Libra project was the tension between providing Internet services that are both scalable and offer rich functionality to end-users. Our thesis was that a component-based approach to service creation and deployment combined with the use of active networking can support rich, scalable services. The main challenges in making this approach work were in the areas of resource management and service specialization. We believe our results in these areas demonstrate the practicality and value of our approach.

The class of services we studied in Libra, primarily interactive services with strict response time requirements, typically use two types of components. A fairly generic communication component (e.g. multicast, publish-subscribe) that provides communication services to a set of high functionality components that provide primarily computational and storage services. The different nature of these components suggests that they can and should be managed differently. For low-level communication-oriented components, scalability is the biggest challenge and a distributed resource management solution using self-organization is attractive. In contrast, for user-level services that offer rich functionality, addressing precise user requirements is more important, so the use of service synthesizers that have application knowledge is more attractive. Both the self-organizing components and service synthesizers can share a common runtime infrastructure that provides, for example, measurement and connectivity support.

Active networking technology has played a key role in making a component-based work. Active networking is used in two crucial ways. First, we need the ability to deploy service components in the network as needed. New service instances may for example need to be deployed to satisfy increased load. More importantly, new instances of service components often need to be placed "close" to the user to meet end-to-end latency or bandwidth requirements. Second, we use active networking technology to customize generic service components so they better support the needs of higher level services or end-users. The idea is that base service functions as an execution environment that can host active applications that implement user or domain-specific functionality, e.g. prioritization of traffic or matching operations in publish-subscribe systems. Customization is especially important for generic low-level services.

While the Libra approach to service deployment looks very promising, a number of open problems remain. A first question is to what degree service synthesizers can be generic. While we have been able to show that it is possible to develop new services using a declarative service definitions that are interpreted by a generic synthesizer, the performance tradeoffs associated with this approach are not well understood. A second question is how we control the deployment of active applications. Clearly, the number of customization or active extensions that a network node supports should not grow linearly with the number of network users. Clearly, some discipline is needed. The answer is probably that we should develop active applications that support applications or application domains, but we have limited experience with this. Finally, it is an open question how the presence of many entities that do active adaptive resource management will affect network behavior and performance. In general, this is likely not to be a problem (today's network user are already adaptive). However, it is possible that the behavior of a small number of large services, e.g. large organizing overlay networks, could result in oscillations and poor performance for other users. More research is needed in this area.

References

[1] A. Akella, R. Karp, C. Papadimitrou, S. Seshan, and S. Shenker. Selfish Behavior and Stability of the Internet: A Game-Theoretic Analysis of TCP. In *Proceedings of the SIGCOMM '02 Symposium on Communications Architectures and Protocols*, Pittsburgh, PA, August 2002.

[2] Scott Alexander, Marianne Shaw, Scott Nettles, and Jonathan Smith. Active Bridging. In *Proceedings of the SIGCOMM '97 Symposium on Communications Architectures and Protocols*, pages 101–111. ACM, September 1997.

[3] Sanjay G. Rao Ashwin Bharambe and Srinivasan Seshan. Mercury: A Scalable Publish-Subscribe System for Internet Games. In *Proc. First Workshop on Network and System Support for Games (NetGames 2002)*, Braunschweig, Germany, April 2002.

[4] H. Balakrishnan, H. S. Rahul, and S. Seshan. An Integrated Congestion Management Architecture for Internet Hosts. In *Proceedings of the SIGCOMM '99 Symposium on Communications Architectures and Protocols*, pages 175–187, Cambridge, MA, September 1999.

[5] G. Banavar, T.D. Chandra, B. Mukherjee, J. Nagarajarao, R.E. Strom, and D.C. Sturman. An efficient multicast protocol for content-based publish-subscribe systems. *The 19th IEEE International Conference on Distributed Computing Systems (ICDCS '99)*, pages 262–272, May 1999.

[6] Ron Ben-Natan. *CORBA -A Guide to Common Object Request Broker Architecture*. McGraw-Hill, 1995.

[7] Steve Berson, Robert Braden, Ted Faber, and Bob Lindell. The ASP EE: An Active Network Execution Environment. In *Paper Collection DARPA Active Networks Conference and Exposition*. IEEE CS Press, 2002.

[8] B.Y. Zhao, J.D. Kubiatowicz, and A.D. Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Technical report, U.C. Berkeley, April 2000. UCB//CSD-01-1141.

[9] Luis Felipe Cabrera, Michael B. Jones, and Marvin Theimer. Herald: Achieving a Global Event Notification Service. In *Proceedings of the 8th IEEE Workshop on Hot Topics in Operating Systems*, Elmau, Germany, May 2001.

[10] Kenneth L. Calvert, James Griffioen, Billy Mullins, Amit Sehgal, and Su Wen. Concast: Design and Implementation of an Active Network Service. *IEEE Journal on Selected Areas in Communications*, 19(3):–, March 2001.

[11] Antonio Carzaniga, David S. Rosenblum, and Alexander L. Wolf. Design and Evaluation of a Wide-Area Event Notification Service. *ACM Transactions on Computer Systems*, 19(3):332– 383, August 2001.

[12] Prashant Chandra, Yang-Hua Chu, Allan Fisher, Jun Gao, Corey Kosak, T.S. Eugene Ng, Peter Steenkiste, Eduardo Takahashi, and Hui Zhang. Darwin: Customizable Resource Management for Value-Added Network Services. *IEEE Network*, 15(1), January 2001.

[13] Prashant Chandra, Allan Fisher, Corey Kosak, T. S. Eugene Ng, Peter Steenkiste, Eduardo Takahashi, and Hui Zhang. Darwin: Customizable Resource Management for Value-Added Network Services. In *Sixth International Conference on Network Protocols*, pages 177–188, Austin, October 1998. IEEE Computer Society.

[14] D-M. Chiu and R. Jain. Analysis of the Increase and Decrease Algorithms for Congestion Avoidance in Computer Networks. *Computer Networks and ISDN Systems*, 17:1–14, 1989.

[15] Y. Chu, S.G. Rao, S. Seshan, and H. Zhang. A Case for End System Multicast. *IEEE Journal on Selected Areas in Communication, Special Issue on Networking Support for Multicast*, 20(8), 2002.

[16] Y. Chu, S.G. Rao, and H. Zhang. A Case for End System Multicast. In *Proceedings of ACM Sigmetrics*, June 2000.

[17] Dan Decasper, Zubin Dittia, Guru Parulkar, and Bernard Plattner. Router Plugins: A Software Architecture for Next Generation Routers. In *Proceedings of the ACM SIGCOMM '98 conference*, pages 229–253. ACM, August/September 1998.

[18] S. Deering. Multicast routing in internetworks and extended lans. In *Proceedings of the ACM SIGCOMM 88*, pages 55–64, Stanford,CA, August 1988.

[19] S. Deering and D. Cheriton. Multicast routing in datagram internet works and extended LANs. *ACM Transactions on Computer Systems*, pages 85–111, May 1990.

[20] Jun Gao and Peter Steenkiste. An Access Control Architecture for Programmable Routers. In *2001 IEEE Open Architectures and Network Programming (OPENARCH'99)*, pages 15–24, Anchorage, April 2001. IEEE.

[21] Jun Gao and Peter Steenkiste. Rendezvous Points-Based Scalable Content Discovery with Load Balancing. In *Fourth International Workshop on Networked Group Communication)*. ACM, October 2002.

[22] Jun Gao and Peter Steenkiste. Design and Evaluation of a Distributed Scalable Content Discovery System. *IEEE Journal on Seclected Areas in Communications*, 2003. Special Issue on Recent Advances in Service Overlay Networks. To appear.

[23] Jun Gao, Peter Steenkiste, Eduardo Takahashi, and Allan Fisher. A Programmable Router Architecture Supporting Control Plane Extensibility. *IEEE Communications Magazine, special issue on active and programmable networks*, pages 152–159, March 2000.

[24] Jun Gao, George Tzanetakis, and Peter Steenkiste. Content-Based Music Retrieval of Music in Scalable Peer-to-Peer Networks. In *Proceedings of the 2003 IEEE International Conference on Multimedia & Expo(ICME'03)*, pages 309–312. ACM, July 2003.

[25] Steven D. Gribble, Matt Welsh, Rob von Behren, Eric A. Brewer, David Culler, N. Borisov, S. Czerwinski, R. Gummadi, J. Hill, A. Joseph, R.H. Katz, Z.M. Mao, S. Ross, and B. Zhao. The Ninja Architecture for Robust Internet-Scale Systems and Services. *IEEE Computer Networks, Special Issue on Pervasive Computing*, 35(4), March 2001.

[26] E. Guttman, C. Perkins, J. Veizades, and M. Day. Service Location Protocol, Version 2. RFC 2608, June 1999.

[27] M. Handley, H. Schulzrinne, E. Schooler, and J. Rosenberg. SIP: Session Initiation Protocol. RFC 2543, March 1999.

[28] Jia-Cheng Hu and Jiann-Min Ho. A Conference Gateway Supporting Interoperability Between SIP and H.323 Clients. Master's thesis, Carnegie Mellon University, March 2000.

[29] Ningning Hu and Peter Steenkiste. Evaluation and Characterization of Available Bandwidth Probing Techniques. *IEEE Journal on Seclected Areas in Communications*, 2003. Special issue on Internet and WWW measurement, mapping, and modeling. To appear.

[30] Ningning Hu and Peter Steenkiste. Improving TCP Startup Performance using Active Measurements: Algorithm and Evaluation. In *11th IEEE International Conference on Network Protocols (ICNP 2003)*. IEEE, November 2003.

[31] An-Cheng Huang and Peter Steenkiste. Distributed Load-Sensitive Routing for Computationally-Constrained Flows. In *Proceedings of ICC 2003 (to appear)*, May 2003.

[32] An-Cheng Huang and Peter Steenkiste. Network-Sensitive Service Discovery. In *4th USENIX Symposium on Internet Technologies and Systems (USITS 2003)*. Usenix, March 2003.

[33] ITU-T Recommendation H.323. Packet-based Multimedia Communications Systems, November 2000.

[34] Manish Jain and Constantinos Dovrolis. End-to-end Available Bandwidth: Measurement Methodology, Dynamics, and Relation with TCP Throughput. In *SIGCOMM 2002*, Pittsburgh, PA, August 2002.

[35] K. Sripanidkulchai. The popularity of gnutella queries and its implications on scalability. http://www.cs.cmu.edu/ ~ kunwadee/research/p2p/gnutella.html.

[36] Keng Lim. A Network Architecture for Virtual Private Networks with Quality of Service, March 2000. Master's Thesis.

[37] L. Keng Lim, Jun Gao, T.S. Eugene Ng, Prashant Chandra, Peter Steenkiste, and Hui Zhang. Customizable Virtual Private Network Service with QoS. *Computer Networks*, 36(2-3):137– 151, July 2001.

[38] W. Marcus, I Hadzic, A. McAuley, and J. Smith. Protocol Boosters: Applying Programmability to Network Infrastructures. *IEEE Communications Magazine*, 36(10):79–83, October 1998.

[39] Introduction to the MBone. http://www-itg.lbl.gov/mbone/.

[40] Microsoft Windows NetMeeting. http://www.microsoft.com/windows/netmeeting/.

[41] T. S. E. Ng and H. Zhang. A waypoint service approach to connect heterogeneous internet address spaces. In *Proceedings of USENIX Annual Technical Conference*, 2001.

[42] T. S. Eugene Ng and Hui Zhang. Predicting Internet Network Distance with Coordinates-Based Approaches. In *Proceedings of IEEE INFOCOM 2002*, June 2002.

[43] Rong Pan, Balaji Prabhakar, and Konstantinos Psounis. CHOKE, a stateless active queue management scheme for approximating fair bandwidth allocation. In *Proceedings of IEEE INFOCOM 2000*, Tel Aviv, Israel, March 2000.

[44] G. Phillips, B. Braden, J. Kann, and B. Lindell. ASP PPI: An Active Execution Environment's Protocol Programming Interface, May 1999. Available at URL http://www.isi.edu/active-signal/ARP/DOCUMENTS/PPI.ps.

[45] P. Reiher, R. Guy, M. Yarvis, and A. Rudenko. Automated Planning for Open Architectures. In *Proceedings for OPENARCH 2000 – Short Paper Session*, pages 17–20, March 2000.

[46] A. Rowstron, A-M. Kermarrec, P. Druschel, and M. Castro. Scribe: The design of a large-scale event notification infrastructure. http://www.cs.rice.edu/CS/Systems/Pastry/SCRIBE.html.

[47] Session Directory. http://www-mice.cs.ucl.ac.uk/multimedia/software/sdr/

[48] B. Segall and D. Arnold. Elvin has left the building: A publish/subscribe notification service with quenching. In *Proceedings of AUUG '97*, Brisbane, Australia, September 1997.

[49] Umair Shah and Peter Steenkiste. Customizable Cooperative Metering for Multi-ingress Service Level Agreements in Differentiated Network Services. In *Proceedings Sixth IEEE/IFIP International Workshop on Quality of Service*, pages 325–341. Springer-Verlag, June 2001.

[50] Srinivasan Seshan Srinivasa Aditya Akella and Hari Balakrishnan. The Impact of False Sharing on Shared Congestion Management. In *Eleventh International Conference on Network Protocols*, November 2003.

[51] Peter Steenkiste, Prashant Chandra, Jun Gao, and Umair Shah. An Active Networking Approach to Service Customization. In *DARPA Active Networks Conference and Exposition (DANCE),* May 2002. Published by IEEE Computer Science Press.

[52] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications. In *Proceedings of ACM SIGCOMM 2001*, August 2001.

[53] Eduardo Takahashi, Peter Steenkiste, Jun Gao, and Allan Fisher. A Programming Interface for Network Resource Management. In *1999 IEEE Open Architectures and Network Programming (OPENARCH'99)*, pages 34–44, New York, March 1999. IEEE.

[54] George Tzanetakis, Jun Gao, and Peter Steenkiste. A Scalable Peer-to-Peer System for Music Content and Information Retrieval. In *4th International Conference on Music Information Retrieval*, October 2003.

[55] vic -Video Conferencing Tool. http://www-nrg.ee.lbl.gov/vic/.

[56] Ellen W. Zegura, Mostafa H. Ammar, Zongming Fei, and Samrat Bhattacharjee. Application-Layer Anycasting: A Server Selection Architecture and Use in a Replicated Web Service. *IEEE/ACM Trans. on Networking*, 8(4):455–466, August 2000.