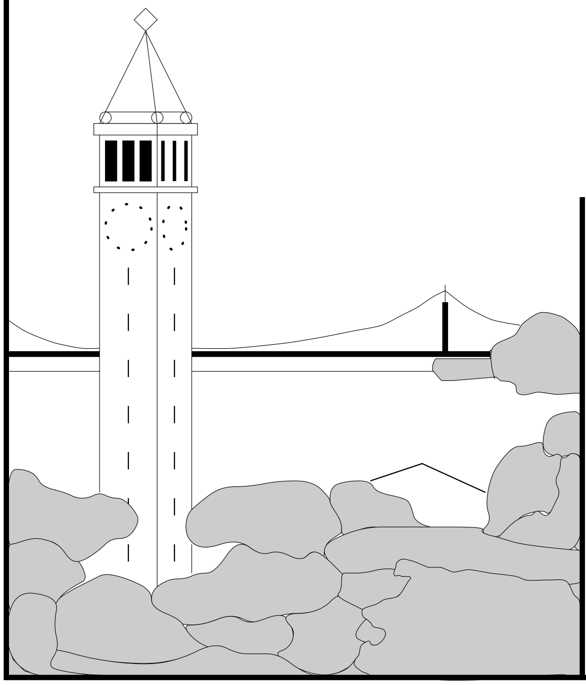
Fault-tolerant, Scalable, Wide-Area Internet Service Composition

Zhuoqing Morley Mao Eric A. Brewer Randy H. Katz
{*zmao,brewer,randy*}*@eecs.berkeley.edu*
CS Division, EECS Department, U.C.Berkeley



Report No. UCB/CSD-1-1129

January 2001

Computer Science Division (EECS)
University of California
Berkeley, California 94720

This technical report is supported by grant number DABT98-C-0038

Fault-tolerant, Scalable, Wide-Area Internet Service Composition

Zhuoqing Morley Mao Eric A. Brewer Randy H. Katz
{zmao,brewer,randy}@eecs.berkeley.edu
CS Division, EECS Department, U.C.Berkeley

January 2001

Abstract

As the Internet rapidly evolves, a growing number of Web services begins to emerge, and the demand for accessing them through extremely diverse end devices increases. Furthermore, a need to create novel functionality (e.g., personalized speech-enabled mailbox) by composing existing services is growing. This presents an emerging opportunity for a middleware service that, given end-point specifications and desired QoS metrics, **automatically** *composes services in a fashion similar to plugging together tinkertoy-like elements to allow service access through heterogeneous devices and networks.* **Services** *in our context are any strongly-typed, network-, and programmatically accessible applications.*

In this report, we present the architectural design of such a system, the implementation and performance evaluation of a prototype. We identify the key requirements of a service composition platform: automation, scalability, and fault-tolerance. We achieve these goals through use of cluster computing platforms, identification of common patterns of ad-hoc, application-specific compositions, classification of services into a strongly typed system, redundant control mechanisms for monitoring and recovery, and a continuous optimization process with feedback. One important contribution of our work is to enable creation of new functionality from existing services with minimal effort. In addition to service reuse, our system also achieves good resource utilization in the wide area by strategically placing and locating services and dynamically adapting to resource variations. We prove its usability by demonstrating the ease with which novel functionality results from existing services and good scaling performance of the system.

1 Introduction and Motivation

New services are appearing every day in the Internet. Popular examples include various search services for people and maps, entertainment services for watching movies or listening to radio or even "attending" live concerts, and E-commerce services such as stock-trading and online-shopping. To enable ubiquitous access to these services, service providers must accommodate heterogeneous end-user devices, access networks, and protocols. For instance, end devices can have vastly differing capabilities ranging from simple ones such as a Palm Pilot with limited computational power and memory due to size and power constraints to a powerful desktop machine with a high-resolution color display, plenty of memory and CPU processing power, as well as access to high network bandwidth. The challenge is to overcome these variations to allow any user to have access to services from any end device.

Similarly, users with different access networks (GSM, pager, PSTN, ATM, and gigabit Ethernet) require the necessary transcoding be done on the data before the service is readily available. Transcoding is needed here because of the differences in these network characteristics. Wired and wireless networks, for instance, have drastically different properties in terms of data format, packet error rate, loss rate, jitter, and bandwidth that must be considered when disseminating service data. In terms of protocol differences, one prominent example is diverse security protocols. A Palm Pilot cannot support heavy-weight security operations involved in SSL [12] due to the lack of CPU power and memory space; therefore, a simplified security protocol such as shared key encryption is used. Sensitive information from the service output consequently needs to be filtered out before delivered to the end-user. Furthermore, heterogeneities exist at the levels of trust. Important data crossing different trust domains before reaching the end-user need to encrypted to prevent compromising security.

In addition to the demand for service access from arbitrary end points, another natural consequence of the explosive increase of Internet services is that novel and interesting functionality can be easily created by composing existing services in a UNIX pipeline fashion rather than building from scratch a complex one. For instance, to access a Web map service using a cell phone, the service output should go through in sequence first a content-extraction service to extract the driving directions from the HTML response. Then it would flow through an optional language translation service to adapt to end-user's language preference. Finally it would pass through a text-to-speech translation service before being finally delivered to the user. In this particular example, four services are composed to achieve the desired functionality.

All the above concerns demand a general software *infrastructure service*[1] that enables service access from diverse clients, *automated* composition of services, data flow optimization, and data path adaptivity. We define **services** to be any strongly-typed, network-, and programmatically accessible applications. **Service Composition** means that a service's output data feeds as input to another service, which in turn may get input from more than one service's output data flow. As a result, the functionality of the services are **composed**. We identify the requirements of a service composition platform to be providing guarantees of scalability, fault-tolerance, and

[1] Infrastructure service: usually a service within the network providing specific functionality for simplification of service constructions. Examples are service discovery services and service composition services such as APC.

automation. In addition, it makes service placement decisions for resource optimization. In contrast, existing Web services have until now accommodated heterogeneity by adding new functionality for each novel device, resulting in monolithic, complex, and unmanageable systems. This application-specific, "vertical" approach, unfortunately, does not scale well with the rapidly increasing variety of devices and requires unnecessary duplication of existing functionality. To guarantee extensibility, it forces services to adapt their content and access protocols for not only all current but also future devices. It is unreasonable to assume that future devices will adopt the standards provided by existing services. One such example is the WAP [8] protocol-specific access of services from wireless devices. Existing approaches do not allow quick and easy deployment of new services when rapid service development is crucial.

Furthermore, vertical approaches generally have worse fault-tolerant properties due to the extra complexity. Rather than having simple, manageable software modules with a specialized functionality and cleanly defined interface for interaction with other components, the traditional approach is much more prone to failures and more difficult to guarantee high availability.

Therefore, a framework is needed for quick and flexible service deployment through "horizontal" or modular composition of existing and new services. Our solution uses the powerful path concept to achieve this goal. A **path** is a flow of Application Data Units (ADUs[2]) [4] through multiple services and transformational operators across the wide area. The need for the latter is to adapt the data into the acceptable format or protocol specifications expected by the subsequent service or device along the compositional chain. Automatic composition is possible by changing services when needed to export strongly-typed, programmatically accessible interfaces, as opposed to untyped, unstructured user interfaces common to many existing services. Legacy services are adapted to this framework through the classification of their input/output format or the construction of simple wrapper services.

We clearly separate computation from transportation for ease of composition by introducing **operators** and **connectors**. An operator or an Internet service instance has a clearly defined input and output type. It is only responsible for computation and is mostly agnostic of the actual data transport mechanism used by a connector for delivery to the user. Thus, one single operator can use different connectors as appropriate for the network conditions and requirements of the end-user. This separation of data computation from delivery allows failures due to network transport to be handled cleanly, which can be independent of computational errors (e.g., wrong input data). Another benefit is that optimization decisions are cleanly separated. For instance, load-balancing decisions only relate to operators and are closely associated with their placement. Connectors, on the other hand, deal with the communication protocol and adapt to the network characteristics to achieve the level of reliability appropriate for the application semantics.

To demonstrate the usefulness and applicability of our solution, we have deployed it in the following contexts. In the ICEBERG [33] project, with the goal of personalized integration of

[2] ADU: the smallest application specific data that can be operated on independently.

communication devices across heterogeneous networks to enable any-to-any communication, our mechanism, **Automatic Path Creation service (APC)**, plays an essential role by seamlessly supporting any new communication device in the infrastructure through merely introducing the appropriate data format conversion services [37]. It takes less than 100 lines of Java code to write such a service using off-the-shelf software. In the Interactive Voice Room Control application [19], APC enables the control of a room's A/V resources such as light and camera via multiple *modes* such as a microphone attached to a desktop, a cell phone, a Palm Pilot or a text window. Currently, we plan to use APC in the post-PC security proxy infrastructure [28], where secure multimodal access to Internet services requires various data transformations, content filtering, security protocol translation services. In addition to successful usage experience of APC within our own research group, we also received good feedback from outside people who downloaded the first release of our software. Those feedback came from a research group at Technical University at Berlin and the Networks and Infrastructure Research group at Motorola Labs in Schaumburg, Illinois. Outside users were able to effortlessly install and use our software with our sample applications as well as extend APC with new kinds of operators for different applications.

The rest of the report is organized as follows. We begin with related work to highlight contributions of our work. Section 3 illustrates the architecture overview of APC. We then describe the local area path design (Section 4) to lay the foundation for subsequent discussion of wide area design of paths. Implementation details of our prototype are covered in Section 7. We present the performance evaluation in Section 8. The remaining section summarizes and discusses future work.

2 Related Work

We briefly examine related efforts focusing on seamlessly interconnecting Internet services and resource-constrained devices. The main distinction is that our architecture provides fault-tolerant, wide-area, and scalable composition automation of both legacy and novel services with dynamic optimization of resource utilization. The resources we consider include computational, memory, and network resources. The optimization criteria (e.g., latency, jitter, data throughput) are either defined by the service authors or deduced from the type of the service. Our emphasis is on reusing existing services and enabling a quick and easy way to obtain new service functionality from existing ones rather than building a very complex and difficult-to-evolve service accommodating a fixed set of protocols and devices. Existing work addresses only specific aspects of the problem space. Additionally, one of our contributions is to provide a well-defined framework for Internet service composition and a taxonomy of computation paradigms to provide differentiated quality of service (QoS).

Our work is influenced by flexible middleware systems supporting distributed computing across heterogeneous resources. For example, Corba [32] provides platform-independent, object-based network communication. DCOM [7] is an equivalent of Corba for the Windows platform. However, neither system directly supports optimal placement of computations. Jini [22] is a Java-centric view exploiting bytecode mobility to deliver stub code implementing a private communication protocol between client and service. Nevertheless, it is mainly designed for use on a much smaller scale than wide area, e.g., workgroup. HP's e-Speak [18] does target for wide area operation but does not address service scalability and fault-tolerance; nor does it provide access from simple devices that are incapable of running java-based communication protocol.

Our work is heavily influenced by projects (e.g., [1, 20, 29, 30, 31, 41, 40]) that transcode to adapt service content to better suit impoverished small devices. However, these approaches are vertically integrated. They do not use composition as a way to easily extend system functionality. Furthermore, quite a number of works have tried to develop a single interface for large classes of devices (e.g., WAP protocol [8]). The success of our work, however, does not depend on the adoption of a single standard. We provide bridging of multiple standards by providing translational elements across them by designing an extensible architecture to adapt to future standards.

On the surface, our design goals are similar to active networks [38], which allow code injection into network routers to deploy new network protocols, implement traffic shaping, and perform packet filtering. Our architecture allows placing computation within the network with the goal of performance and resource usage optimization. The distinction is that for active networks, data processing occurs at transport or packet level rather than using application semantics. In our work, operators are application-oriented and process application data rather than router-level packets.

The idea of path or composition, similar to UNIX pipeline-like chaining of different commands, existed in many previous works. For example, Scout [23] uses the path as an explicit communication-oriented abstraction in operating system design. In Scout, path facilitates OS specialization by enabling configurability and exposing global context that optimization techniques can exploit. Path also assists resource allocation and management by being a single unit of scheduling entity. Infospheres project [16], as another example, focused on building compositional systems from interacting components. MPLS (Multiprotocol Label Switching) [13] defines the abstraction of a label-switched path to provide QoS in IP networks. Here, we extend this idea of composition to wide-area, independent Internet services. The extension includes automatic path formulation as well as runtime path maintenance.

Our work can be considered as an extension to the TACC programming model [11] with additional design of wide area service placement, continuous resource optimization through feedback, and generalized load balancing. TACC model provides composition of stateless data transformation and content aggregation with uniform caching of data. The composition model is static and inflexible. We are exploring a completely automated composition model and programmable compilation of composition chain. Furthermore, services considered in our framework are quite general, including continuous latency-sensitive stream services such as live audio and video as well as support for mobile wireless clients. These are not addressed by TACC.

3 Automatic Path Creation Service Architecture Overview

In this section, we introduce the overall architecture of the APC service by first illustrating the design goals, benefits of the APC, and then describing the individual components of the service architecture and finally the path construction process.

3.1 Design Goals

The main design goal of Automatic Path Creation is to simplify service authorship by making it easy to compose services and guarantee good performance of the resulting composed service entity. There are three key properties that the APC must have to achieve this goal. We illustrate how these goals are achieved in subsequent sections.

Automation. To facilitate service composition, the APC service should attempt to automate as many parts of the path creation process as possible to reduce unnecessary user involvement. Our design automates discovering paths between system components, fine tuning and optimizing the performance of the data flow dynamically during execution, and handling error conditions. The details of these mechanisms are described in the implementation Section 7.

Fault-Tolerance. Whenever possible, the APC service should protect users from the failure of individual path components or communication links between them. It must provide the illusion that user is accessing a single robust service entity providing the composed functionality.

Scalability. The APC service should be able to handle large numbers of concurrent users. In addition, the path construction algorithms must scale well with the increase of services. This is particularly difficult, as the number of possible paths grows exponentially as components are added.

3.2 Benefits of the APC

The benefits of a sophisticated path creation and maintenance mechanism are far reaching. In addition to enabling component reuse and simplifying service creation, the APC service also provides interesting new points for dynamic performance optimization. These benefits are described below. The complexity of path construction mechanism does not necessarily imply high overhead, because our search algorithms are efficient and we use caching for speedup. We argue that intelligent path maintenance and optimization are indispensable to provide high availability and guaranteed quality of service of the resulting composed service.

Service Access from Diverse Clients. As discussed in Section 1 as motivations for the APC service, such a facility detects when to insert operators to transform the content of the service to make it suitable for the end clients. It is similar to the system implemented in [10]. For example, image distillation operators and HTML-to-text transformation operators are automatically inserted to display the Web content on a Palm Pilot.

Composition of Services. The APC service can be used to create new services by combining functionality from existing Web services. In addition, the dynamic nature of the APC effectively allows the creation of services on-the-fly, without requiring explicit service setup by a programmer or a system administrator. For example, given an MP3 jukebox service and a GSM transcoding service, the APC service could immediately create a service allowing users to listen to MP3 songs on their cell phones.

Data Flow Optimization. The APC service examines many potential paths before deciding on a particular one. During the course of this examination, it can weigh the costs of the various paths, and choose one that optimizes for quality of service as specified by the user, resource consumption, or some other desired metric.

Data Path Adaptivity. By allowing the optimization process to continue throughout the lifetime of a given path, the APC service can dynamically adapt the path to the changing characteristics of the execution environment. For example, if a network link becomes overloaded while data is flowing through the path, the APC service could redirect this flow through a different channel, to improve the quality of service. If a network link suddenly becomes lossy, then FEC operators can be dynamically inserted to increase data goodput.

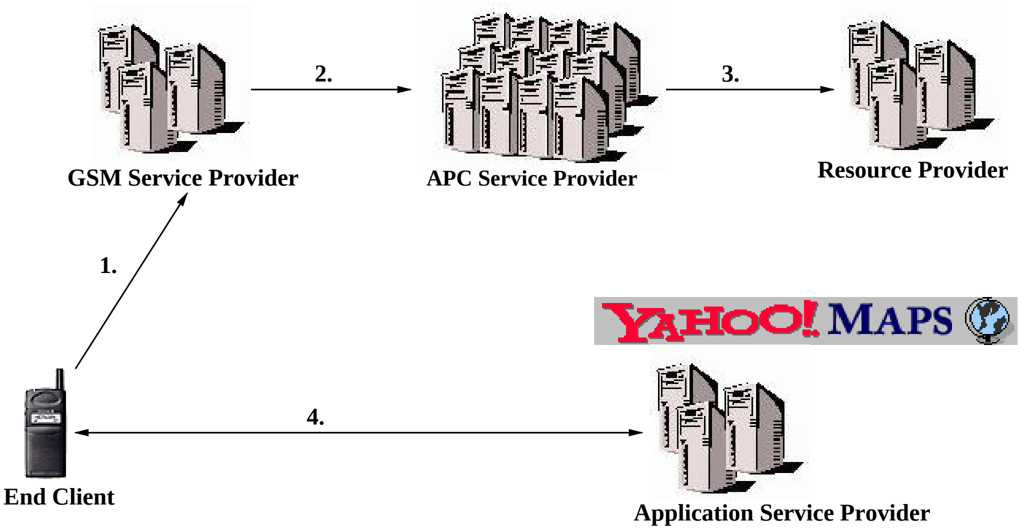


Figure 1: **Example path request scenario:** This figure shows the sequence of events that occur for a cell phone user to access a Yahoo Map Service.

3.3 Example Path Scenario

We revisit our motivating example in Section 1 for service composition. The example is using a cell phone to access a Web map service, providing HTML-formatted driving directions in response to a user's query. The following sequence of events occur for path creation as indicated by the arrows and numbers in the Figure 1.

1. First, the End Client sends a request for accessing the Yahoo Map Service to its Network Service Provider by dialing a number or selecting the menu on the screen. In this case, the End Client sends that request to his GSM Service Provider.
2. Then, the GSM Service Provider sends a data path creation request to the APC Service Provider in the network to request for the path to be built between the Map Service and the client.
3. The APC, given end point information, determines the data path and its physical locations using its path creation algorithms (Section 7.4). It subsequently contacts the corresponding Resource Provider of these physical locations to place the computation in the network. If the operator is already running, the APC sends a request to it.
4. The Resource Provider creates the necessary computations in the network. Thus, a data flow is established between the End Client – cell phone user and the Application Service Provider – Yahoo Map Service.

We now describe the actual paths created by the APC in this example scenario. Given user's specifications of the desired service and access device, the APC creates the following two paths as shown in Figure 2. The first one going from the cell phone to the map service is used to send the request to the service using voice command. The request transcoder converts phone input into an HTML form submission acceptable to the map service. The second path in the reverse direction returns the map answer to the user. The map service response in HTML format first goes through a content extractor that extracts driving directions. Then it goes through a text-to-speech translator to convert it to audio data. Finally, the speech output needs to be changed to GSM format before it is returned to user's cell phone. This example illustrates the composition of the map and a content extractor service on the Internet. It also shows how transformational operators for protocol or data format conversion (e.g., text-to-speech converter) can be dynamically created as needed.

3.4 Operators and Connectors

Now we formally define the key concepts of path. A **path** is comprised of a sequence of **operators** that perform computation on data joined by **connectors** that provide data transport

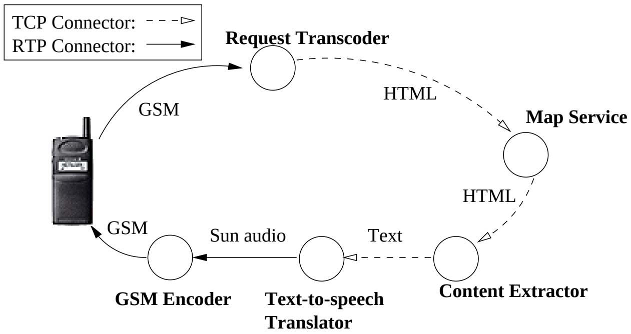


Figure 2: **Example path scenario:** This figure illustrates the two paths created for requesting map directions using a cell phone from a Web map service. Each circle denotes an operator which can be either a long-lived service instance (e.g., Map Service) or a dynamically created one (e.g., GSM Encoder). Each arrow in the figure indicates a connector and the direction of data flow.

between operators. The goal of creating these two distinct abstractions is to enable easy-to-use and semantically correct composition of tinkertoy-like elements that result in new service functionality.

A connector is an abstraction of the ADU transport mechanism between two operators. This abstraction encapsulates various properties of the transport such as reliability, order of delivery, and drop policy. Application Level Framing (ALF) [4] can be easily supported to allow service authors to assert application-specific control instead of simply choosing between TCP and UDP transport. The key advantage of this abstraction is that a connector hides the potential differences in network protocols from the operators and allows them to communicate as long as the output data type of the downstream operator matches the input data type of the upstream operator. Each connector is characterized by a specific transport protocol. In the above example, there is an RTP-based audio connector from the cell phone to the request transcoder and a TCP-based reliable stream connector to the map service.

Operators perform the actual computation tasks required by the path. Their input and output are strongly typed. Strong typing not only enables dynamic composition and automatic optimal path creation, but also reduces runtime errors. Operators also have various attributes, such as communication protocols they support, computational requirement, special external input data (e.g., a remote database) and other application specific ones. In addition, operators have associated cost metrics, which describe the run-time performance of the operators. They are used as optimization criteria during path selection. The type and attributes (Section 3.7) for each operator are combined to form an XML description of the operator. Operator XML descriptions conform to a Document Type Definition (DTD) defining the general set of operator properties. This description determines which combinations of operators could make a valid computation path and which is the optimal one.

The APC supports both long-lived and dynamically created operators. The former includes standard Web services. These can be registered with and located through a Service Discovery Service [5]. Dynamically created ones are light-weight, short-lived transformation elements created by the APC as required. They have only soft-state, and hence can be restarted transparent to the user in the case of a node or process failure. If a long-lived service instance fails, the APC automatically finds another instance and redirects the data flow.

While the reliability of both kinds of operators helps guarantee that a path is reconstructed when a failure occurs, this does not safeguard against the loss of data that was already in the path when the failure occurred. Thus, application-level retransmission using acknowledgment in addition to connectors with reliable transport is needed for guaranteed data delivery.

3.5 Operational Model of the APC architecture

Before describing the detailed steps of the path construction process (Section 3.6), we first introduce the operational model of the APC and answer questions such as who constructs a path and how the APC fits in the existing model of the Internet. In this model, five classes of entities exist: APC Service Providers, Network Service Provider, Application Service Providers, Resource Providers, and End Clients. **APC Service Providers** are responsible for running APC service instances. They accept path creation requests, construct paths, maintain paths, and tear down paths. **Network Service Providers** are the ISPs for the different networks, providing services to the clients. For example, Sonera is a GSM Service Provider in Finland. **Application Service Providers** maintain application services such as Yahoo! Map Service, various E-entertainment sites and E-commerce sites. Network Service Providers typically request to have paths built on behalf of their **End Clients**. The request can be triggered by a number of events, e.g., data format mismatch between the service and the end client, the need for new functionality. After the paths have been created, APC Service Providers maintain the paths until they receive a tear-down request from Application Service Providers. The APC service itself is advertised at a well-know IP address, which is the virtual IP address of a Web switch for server load balancing (e.g., [9]), behind which a large number of the APC service instances exist for purposes of scalability, fault-tolerance, and availability. **Resource Providers** offer locations in the network and computational resources for running services. For example, data hosting centers in the current Internet are a class of Resource Providers. Application Service Providers can use these resources to host their applications. APC Service Providers can also utilize such resources to create short-lived operators on demand. They can also potentially use these resources to run their own APC service instances. Before a physical machine can be used to run operators, a **Connection Manager** service needs to be spawned by the APC service. It is responsible for starting and maintaining the operators on the physical node. The final class of entities – **End Clients** are average consumers who use the service provided by Application Service Providers. They do not have knowledge of the existence of APC Service Providers which operate transparently to the End Clients.

Operators have the following operational model. They are located using a wide area Service Discovery Service (SDS) (e.g., [5]), which maintains a searchable database of all the XML descriptions its corresponding operators. SDS is advertised at a well-known IP address. It also takes care of operator registration, discovery, and querying. Only trusted Application Service Providers are allowed to register new operators with SDS for security reasons. After the APC determines where to place an operator, it contacts the Connection Manager Service running on that machine to place the computation and establish the network connections to neighboring operators.

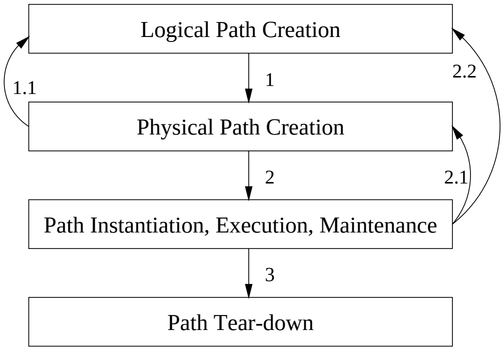


Figure 3: **Path construction process:** Path execution is an iterative process of optimization. The APC service guarantees the availability and fault-tolerance of a constructed path by rebuilding its physical or logical path when components fail. The following information is passed between each stages corresponding to the arrows in time sequence: 1: Logical path description and user QoS metric, 1.1: Operators that do not satisfy the QoS metric. 2: Physical path description. 2.1 and 2.2: Failed operators or connectors. 3: Physical path description and running status.

3.6 Path Construction Process

To construct a path, the Network Service Provider for the service whose content is to be composed sent a request to the APC along with the information pertaining to the endpoints of the required path, any specific operators that must be included in the path, the optimization metric, and an acceptable range of costs for the path. Both the metric and the cost are application specific and can be one of the following: data latency, data throughput, and output data characteristics (e.g., audio/video quality, image resolution). This information is needed to construct an optimal path.

The path construction process consists of four steps. As shown in Figure 3, it is an iterative process of continuous feedback and optimization.

Step 1: Logical Path Creation. A **logical path** consists of an ordered sequence of operators joined by connectors. During the logical path creation, the APC searches through the XML descriptions of the operators available to find valid sequences that could perform the computation requested by the Application Service Provider. The result is a list of possible operator sequences ordered by decreasing cost on the user's input parameters for optimization (e.g., latency, data throughput, voice quality). The search is performed using shortest path search on the graph of operator space. Optimization criteria are application specific. One goal of the APC is to adapt to application requirements to optimize resource utilization.

Note that since some operators may be commutative (e.g., image format transcoders), the space of all possible logical paths can be huge given a large number of services. Hence, as a heuristic, only a small number of logical paths are generated initially. As indicated in Figure 3, additional logical paths can be produced as needed if the physical paths for the first set of selections are not optimal (i.e. cannot satisfy user's specified QoS metric or do not have acceptable performance). Thus the tradeoff for better response time does not compromise the degree of optimization for quality of service.

Step 2: Physical Path Creation: A **physical path** is a logical path, along with a choice of actual nodes (i.e. physical machines) on which to run the operators. Nodes for long-lived operators are chosen from the known service instances' locations that provide the desired functionality based on application dependent criteria such as response time, data throughput, and image resolution. Nodes for dynamic operators are selected according to the computational capabilities, the cost of using that node in the path, and various other criteria. Some of these operator placement decisions include operator computational requirement, software/hardware requirement, output/input properties (e.g., data location, data volume, delay-sensitivity, degradation properties), network characteristics (e.g., bandwidth, delay, packet loss rate, and jitter characteristics). The APC constructs a physical path from a logical path by finding the lowest cost nodes that meet the requestor's requirements. Please refer to Section 7 for detailed algorithms of both logical and physical path search.

During physical path creation, optimization operators such as FEC and compression operators are inserted for performance enhancement. FEC operators are added between a wireless link to reduce packet loss rate. Compression and decompression operators are added between links with heavy data throughput to avoid overloading the network.

Step 3: Path Instantiation, Execution, Maintenance, and Querying: Once the physical path has been determined, the APC starts any required dynamic operators and sets up appropriate connectors between communicating operators. After all the nodes in the path are set up, the data flow is started. In addition, a **control path** is established between the operator nodes and the APC. It is used for both reporting of error conditions and performance information.

During the lifetime of the path, the APC actively monitors the operator nodes to make sure that they are functional. Any operator can also report problems to the APC about its neighboring operators, so that the path can be repaired when necessary. The control path also plays an important role in enabling operator repair, deletion, and insertion. It is used for exception handling, controlling parameters of path components, monitoring and analyzing path performance. Therefore, it needs to be highly robust and unaffected by the data path's failures. However, a control path can overlap the data path: each path operator can have a handle to its two neighboring operators. The APC thus monitors the performance of the path and reroutes the data path if new conditions make the original path suboptimal by going back to the physical or logical path creation stages. Please refer to Section 6 for details of path adaptivity.

Step 4: Path Tear-Down: When a path is no longer needed, the Application Service Provider informs the APC. The APC service then stops the data flow, removes connectors, shuts down any dynamic operators, and frees other relevant resources. As a performance optimization, the APC caches the logical and physical information of commonly used paths for reuse at a later time.

3.7 Operator Properties

To efficiently accommodate diverse application requirements, the APC needs to consider operator properties in the following dimensions. Based on this taxonomy, we also discuss affected design decisions in path and operator construction, operator fail-over mechanism.

- **Latency Sensitivity:** real-time, interactive (e.g., PCM-to-GSM encoder) vs. off-line, bulk-data operation (e.g., Postscript-to-PDF converter). The former must be light-weight with low startup overhead. The operator author should reuse computation results to speed up processing. Similarly, as an optimization, path search results such as logical or physical paths can be reused. Admission control and continuous monitoring of the physical machines should be used to guarantee high throughput. Furthermore, fast failure recovery is desired to minimize disturbance to the end-user. Buffering is implemented by the operators to hide failures and minimize fault recovery overhead. Paths consisting of real-time operators must be highly aware of changes in network characteristics (e.g., jitter, loss rate, delay, and throughput). Flow control mechanisms are built into the connectors and operators (Section 7). Dynamic insertion of error correction codes (e.g., FEC operators) to adapt network changes are desirable.

  Off-line operators on bulk data are much simpler to construct. No timing constraints exist in paths consisting entirely of this type of operators. In case of faults, the entire path can be restarted if necessary.

- **Computation Requirement:** compute-intensive (e.g., MPEG Encoder) vs. light-weight (e.g., sound format converter). The first type of operator is usually long-lived and shared among different path instances. It is thus always reused rather than created on demand. The second type must have low startup overhead, since it is usually created on the fly. If the startup latency becomes unacceptable, then such operators are reused or "prefetched" by creating in advance.

- **Location Dependence:** location affinity vs. location-independent. Location dependence can be due to the need of special hardware/software or data input locations (e.g., a remote database). During physical path creation, locations of this kind of operators need to be predetermined to satisfy these constraints.
- **Data Output Size:** large operator data output vs. small. The former kind has large bandwidth requirement; therefore, placing it closer to its subsequent operators is preferred to avoid overloading the network when network resources are scarce. If that is not possible in practice, a *compression* operator can be inserted after it to reduce the data size.
- **Operator State:** stateless vs. stateful. The former kind can be restarted as needed for failure recovery. The latter kind needs special mechanisms such as checkpointing for fault-tolerance and robustness.

3.8 Operator Functional Classification

To automate the logical path construction process, the APC needs to be aware of all supported operators. It is thus useful to have a meaningful categorization of operators in their functionality to aid the operator selection process. Both the functional classification and the operator property in the previous section are included in the operator XML description. We classify operators into the following four categories.

- **Data Format Translation Operators.** This type of operator performs conversions between different data formats; e.g., GIF to JPEG, PCM to GSM, powerpoint to HTML.
- **Protocol Conversion Operators.** These include translation between different security protocols. For example, an operator that converts between the heavy-weight security handshake protocol such as SSL [12] to simple shared-key encryption and decryption falls into this category. Another example would be UI interface conversion. Obviously, the graphical user interface of an application running on the desktop would be completely different from that running on a Palm Pilot. Necessary conversion needs to be done.
- **Content Transformation Operators.** This type of operator filters or aggregates the information produced by the previous operator based on user preference, e.g., filtering out action movies. Another example would be a language translation operator from English to Chinese. or an operator giving a summary of a paragraph.

- **Optimization Operators.** This category of operators does not change the semantic content of the data. Instead, they modify the properties such as data size and security features. These can be lossy or lossless compression or decompression operators to optimize bandwidth usage, encryption or decryption operators, and FEC operators to lower data error rate. The tradeoff of adding these operators is improved resource utilization but more operation complexity and potentially increased latency.

4 Local Area Path Construction

The APC service is designed to accommodate Internet service composition across the wide area. A result of this path abstraction is that a path becomes a single unit of resource allocation, scheduling and error-handling. To effectively tackle this problem, we first examine our design for local area service compositions and then generalize it to the wide area case in the next section.

4.1 Cluster Computing Platform

Within the local area network, the APC's execution environment is a cluster of workstations. We leverage several desirable properties of a cluster computing platform: incremental scalability, fault-tolerance, high availability through redundancy, and high network bandwidth [3]. A cluster allows incremental scalability, as new machines are added to increase computational and storage capacity. By creating multiple service instances when existing ones are overloaded, the APC service can easily scale with increasing number of operators and clients. Each APC service instance can handle any path creation requests. Furthermore, each instance represents an independent failure boundary; there is no single point of failure in the service architecture. When one instance fails, another instance takes over its task due to the distribution of computation and persistent service state. To achieve strict consistency of state information, we use a distributed data structure (DDS) [15], a self managing storage layer designed to run on a cluster of workstations and to handle heavy Internet service workloads. Given that there are multiple services instances running within the cluster, the probability that none of them being available is rather low. Therefore, the overall availability of the APC service is greatly increased. Furthermore, a cluster can usually provide networks with greater than 1 Gb/s throughput with 10 to 100 μs latency [3].

Resources within a cluster are much more easily managed due to high network bandwidth, low response time, and low probability of network partition. Therefore, a *cluster resource monitoring service* for reporting dynamic information of network and computational resources is not difficult to build. Such a monitoring service would keep track of running services, machine load, memory usage, machine computational capacity, network bandwidth and various other resource information helpful to make load-balancing and operator placement decisions (see Section 4.3). Given a specified set of constraints in terms of software, hardware, processing speed, memory requirement, the resource monitoring service should find the available set of machines satisfying them. The monitoring service is a distributed network service with the scope of the distributed clusters such as that Millennium project [34].

In addition, a cluster provides the functionality of service location or discovery, service querying or identifications [5], and service listing to the APC facility.

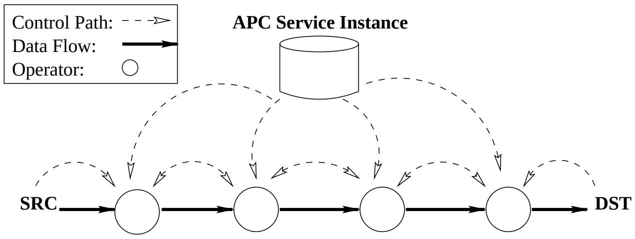


Figure 4: **Redundant control paths:** To ensure a fast and robust fault-recovery model, multiple control paths are built into the system to guarantee their robustness. Each APC service instance is responsible for a subset of paths created in the cluster.

4.2 Fault Recovery Model

Despite inherent redundancy and low probability for network partition within a cluster, failures can still occur at both the process and machine level due to hardware problems or software bugs. If paths span multiple clusters, network partition can also cause path failures. It is critical for a path to gracefully and quickly recover from failures. The assumption is that the APC itself is highly available and fault-tolerant due to redundancy in the cluster. New APC service instances are created in response to failures or high load. At any given time, the cluster guarantees that there are sufficient number of service instances to handle the load. The number of service instances grow dynamically in response to increasing load. Overload is detected by the cluster computing platform when the task queue size of each service instance exceeds a limit. If any APC service instance fails, the cluster automatically starts a new one. Additionally, any component within the constructed path can also malfunction due to process or node failure.

However, the cluster fault recovery mechanism alone is not enough, since existing running paths will not recover even when new services instances (i.e. operators) are created to replace the failed ones. Connectors need to be reestablished and lost state may need to be recovered. The APC provides two redundant sets of control paths for detecting and handling path component failures as illustrated in Figure 4.

APC Monitoring– The APC periodically sends UDP heartbeat beaconing messages to each operator within the path to make sure they are functional. Upon timeout, the APC assumes either process or node failure, and thus attempts to restart the operator. First it tries to reuse the existing node provided it is not overloaded or unreachable. If that fails, APC locates a new node through the cluster. If that is still unsuccessful, reconstruction of logical paths is attempted.

Peer Monitoring– Operators are network I/O intensive and continuously receive data from and send data to neighboring operators. Upon catching an I/O exception when reading or writing data, an operator immediately notifies the APC so that failure recovery can be quickly initiated.

4.3 Placement of Operators

Given the classification of operators (Section 3.7 and 3.8), we now examine what considerations affect operator placement decisions in step 2 of the path construction process. We will revisit this topic in the context of wide area path design (Section 5).

The goal of this process is to optimize the path requestor's specified QoS metric by optimally utilizing resources within the infrastructure. Resources encompass not only CPU, but also memory, storage, and network capacity. Placement decisions first must satisfy potential resource requirements of the operators such as special hardware or software needs, processing power, and storage capacity. This limits the set of physical nodes as the candidates for running a given operator. APC uses a cluster resource monitoring service to match this set of constraints with the set of physical machines. Based on user's optimization metric, the best placement solution is then found using shortest path graph search (e.g. Dijkstra's algorithm) by expressing various dependencies among resources and optimization goals as a directed graph. Nodes of the graph represent potential computational sites. The edge itself exists if the two nodes are adjacent nodes in the logical path. The cost of the edge is dependent on the properties of the network link between the nodes as well as the characteristics of the nodes. Usually the two end points of the paths are fixed given the location of the End Client and the service of the requesting Application Service Provider.

5 Wide-Area Path Construction

We now discuss our design of constructing wide area paths consisting of local area paths linked together. There are several major differences from the local area case. First of all, wide-area latency is much higher. There is a higher probability of failures due to network partition. As a result, cluster-level monitoring is necessary to detect any cluster failure. Additionally, there are more security concerns as data travels across different trust domains, since it must then be encrypted. Accurate resource monitoring is difficult due to high statistical variation and high latency. Finally, there are more administrative and policy issues to be addressed when searching for logical and physical paths.

5.1 Wide-Area Path Selection Decision

We first address when to construct a wide area path as opposed to a local one. If a path requires special operators not present in the local area, it is inevitable to build a wide area path. Furthermore, if the two end points of the path span across the wide area, and the operators are latency-sensitive, it becomes desirable to place certain operators closer to either end point rather than gathering all operators on a single cluster in the middle of the network. In addition, it is impossible to place all services within the core of the network due to high network delay and difficulties to support applications such as real-time video conferencing which demand high quality of service. It is thus vital to distribute services to the edge of the network where there is large amount of bandwidth, low packet loss rate and jitter. However, certain services still need to remain in the core network for ease of accounting and state maintenance. Consequently, there is a need for wide area path connecting between end users at the edge to the core of the network.

5.2 Wide-Area Path Construction

Some of the lessons learned from building local area paths are clearly applicable here, given that a wide area path is made of local area paths strung together. Since resources in the wide-area are more unpredictable, less controlled, and access is complex to obtain, adaptivity to resource changes becomes even more important. Intelligent decisions in operator placement, insertion, and deletion are critical to obtain good performance and scalability. The wide area path construction is built upon or protocol for the local area presented in Section 3.6. Given a path construction request consisting of relevant information (e.g., data format, location descriptions, any special constraints) and quality of service specifications in terms of application-specific metrics, an optimal path is determined through the following protocol.

1. First, the set of available clusters are determined, along with their properties such as supported format, operators, available resources. This information can be obtained through a soft-state-based database on each cluster updated through periodic beacons, whose frequency is adaptive to resource changes.

2. Given this information, we determine which clusters to use for the path based on the network distance between the path endpoints, the network connectivity within and between clusters, and the available computational, memory, and storage resources of each cluster. We select clusters that are close to the path end points (e.g., cluster A, C in Figure 5), have high network bandwidth between each other, and with sufficient resources.

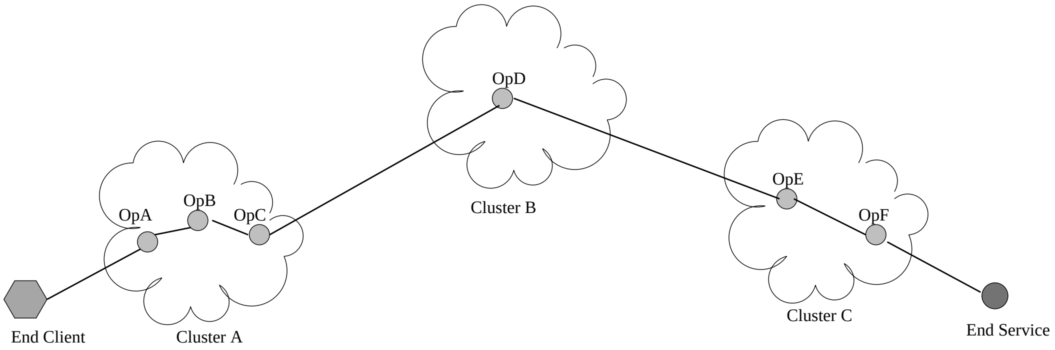


Figure 5: **An example wide area path:** This figure illustrates an example wide area path. Each circle denotes an operator. Each connecting line between the operators indicates a connector. The path is requested by the end service.

3. We then determine the set of intermediate data formats between clusters. This is achieved through a negotiation protocol. Each pair of adjacent clusters in the path exchange common data format information. In Figure 5, the negotiation occurs between Cluster A and B, Cluster B and C. The two end clusters (A and C) need to guarantee that a path exists between the common format and the path end point.

4. After the common format has been determined, each individual cluster performs its own shortest path search for locating the optimal logical local area path, based on the input and output formats.

5. The results of individual clusters are aggregated to verify that user constraints and input QoS requirements are satisfied. The process may be repeated until we find a set optimal logical paths satisfying the input constraints.

6. Subsequently, each local area physical path can be determined in individual clusters with special resource optimization considerations discussed in the Section below.

If data crosses different trust boundaries, encryption operators need to be inserted. Note, the logical path creation process may be repeated if no physical path satisfying the input QoS requirement is found.

Placement of Operators– For a path that spans the wide-area, network resources are less controlled. It therefore becomes critical to exploit network topology awareness to optimally place operators. It is desirable to have an up-to-date repository of network resource information (e.g., [39]) to determine which clusters to use for constructing the path and how to strategically place operators. Optimizations such as FEC and compression operators become very important to achieve high QoS.

Cluster Failure Recovery– Across the wide area, a network partition can occur causing an entire cluster to be unreachable. It is therefore important to have cluster-level failure detection mechanisms in addition to that of process and node discussed in Section 4.2. APC designates a few nodes in each cluster to be *monitoring nodes* responsible for periodically sending beacon messages to other clusters in the wide-area path. By appropriately adjusting the timeout value based on the tradeoff between bandwidth utilization and response time to fault, APC can quickly detect cluster failure or network partition. APC services on the remaining clusters of the path subsequently locate a new cluster with similar locality and operator functionality to rebuild the path on the failed cluster by searching through its softstate-based database containing wide area information. Connections between this local path and other path components are re-established. If no available cluster is found, the logical path is rebuilt by the APC service instances on the remaining paths. Path rebuilding and recovery is transparent to both the End Client and the Application Service Provider.

6 Paths Adaptation and Personalization

Before diving into implementation details of the APC in the next Section, here we focus on how paths adapt to resource variations and end user mobility to maintain high quality of service. We also briefly discuss how the APC achieves personalization of services.

6.1 Enabling Personal Mobility

In addition to format conversion, the APC also provides personal mobility support needed by mobile clients. A roaming client switching from wired network to wireless access may change his IP address but would still like to continue the ongoing service session without any interruption. A session service redirection proxy is provided by the mobile operator created in the data path to dynamically detect user's mobility pattern change and redirect the service data to the moved clients. Such a redirection proxy also caches application-specific data during temporary network disconnectivity to provide the illusion of a continuous session.

6.2 Dynamic Adaptation to Resource Variations

Frequently end users may experience degraded performance of the service due to dynamic changes in network conditions, e.g., sudden drop in bandwidth due to network congestion, or other resources such as computational cycles and memory space. There is a need for applications to adapt to dynamic changes in available resources to optimize user's perceived quality of service. We define three ways to adapt to changes in resources. These adaptations are performed only after the effect has been observed to persist beyond a threshold amount of time, since reacting to transient resource changes result in unjustified overhead and unstability of the system.

- **Application-intelligent adaptation**

  The application is powerful enough to do its own adaptation to resource changes. For instance, RealAudio combines multiple streams encoded for different bit rates into a single clip, and RealVideo uses a single codec to encode data for all bandwidths. During runtime, the audio and video streams dynamically adapt to changes in bandwidths [26]. In this case, the APC should directly take advantage the application adaptation mechanism in the composed path.

- **Application-specific adaptation**

  The application provides mechanisms for dynamic adjustment, but does not do so automatically. For instance, for bandwidth adaptation, there are different instances of the same codec intended for different bit rate. A codec may be error-resilient, but needs to be notified of the current error rate through a control channel. In this case, the APC is responsible for monitoring the resource changes and providing the feedback to the applications to enable dynamic adaptation.

- **Application-independent adaptation**

  If there is a lack of knowledge of the underlying implementation of the application, the APC treats it as a black box and does application-independent adaptation. For instance, to adapt to high packet loss rate, forward error correction (FEC) and compression operators are inserted for better data throughput. FEC can also vary the amount of redundancy based on the packet loss rate.

Combinations of the above approaches can be used if needed. Given the path requestor's optimization criteria, the APC strives to create service compositions that best utilize network and computational resources to achieve the optimal desired QoS. Optimized resource utilization and differentiated QoS are enabled by our iterative data path construction process with continuous feedback (see Section 7) and clear specification of optimization metrics.

6.3 Enabling Service Personalization

Not only does ICEBERG provide service mobility support, it also stresses the concept of having the person instead of the device as the communication endpoint. This level of personal mobility is possible by creating a single identity for an individual providing a level of indirection to the desired endpoint for communication. A service thus can be transparently accessed by the user regardless of his endpoint communication device. Furthermore, services are customized using the client's preference specification depending on the user's current activities. The *Personal*

Activity Coordinator (PAC) service [17] keeps detailed account of the current ongoing activity of the user to make services properly customized according to user's location and activity. The APC service queries the database of user's preference and constructs the appropriate data path.

6.4 Localization of Services

Depending on end user's current location, services can be *localized* by incorporating useful local information. For instance, a user sitting in a traffic jam moving slowly will automatically receive updates on alternative routes to his destinations as part of the service data. A student going into a Computer Science building automatically gets information about ongoing seminars. This is possible because the APC is continuously keeping track of user's current location by having the very last operator in the composition chain reporting to the APC and getting updates through PAC about user's current activity and customizes the service output accordingly. Thus, services can become more context- and location-aware.

7 Implementation

Here we describe our experience in implementing two prototypes of the APC, and using them to compose services. We discuss how we achieved our design goals: Automation, fault-tolerance, and scalability. The detailed performance analysis for the local area is found in Section 8.

7.1 The APC Service

Our implementation encompasses the full range of path creation steps described previously supporting both long-lived and dynamic service instances. Our APC implementation is a cluster-based Ninja infrastructure service [35] providing fault-tolerance and high availability of the composed service.

7.2 iSpace-based Implementation

The first prototype is based on Ninja iSpace, which is thread-based and uses blocking RMI calls for interprocess communication. We quickly ran into scaling problems after the number of clients using the service exceeds a certain number. This is due to thread context switching overhead as well as the memory requirement for storing thread state. Some other disadvantages of thread-based programming model include lack of graceful degradation and lack of explicit scheduling exposed to the users.

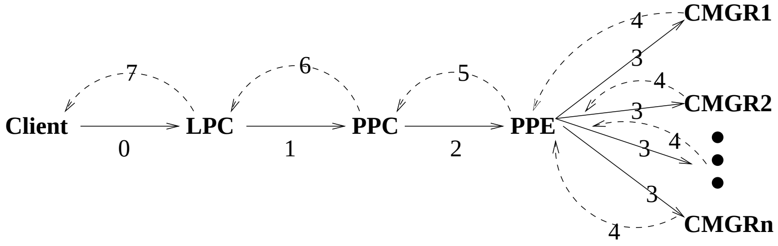


Figure 6: **vSpace-based path implementation:** This diagram shows the sequence of task dispatching (indicated by solid arrows) and completion returning (indicted by dashed arrows) between different workers and the end client to execute a path request.

7.3 vSpace-based Implementation

The second prototype uses Ninja vSpace, an event-driven cluster service programming platform. vSpace workers or service instances are asynchronous event handlers that accept tasks (i.e. units of work to be processed) as well as completions (i.e. return values). Interactions with the distributed data structures and other modules is achieved through non-blocking, split-phase RPC calls, taking an upcall as a parameter and returning immediately. At a later time, when the actual result is received, the upcall handler is invoked. We perceive a great improvement in scalability as a result, due to reduced overhead in context-switching and thread state.

The APC service is structured in the manner illustrated in Figure 6. It consists of four types of workers – Logical Path Creation Worker (LPC), Physical Path Creation Worker (PPC), Physical Path Execution Worker (PPE), and Connection Manager Worker (CMGR). LPC workers create logical paths provided with a path request (arrow 0) containing the two end point descriptions of data format, IP address, port number, input arguments to the service, QoS metric, any required operators in the data path. Then they dispatch a task (arrow 1) to the PPC workers which determine the physical locations to run individual operators. This task consists of the original path request as well as the logical path description of the sequence of operators joined by connectors. Note, the path may not be linear.

Subsequently PPE workers receive a task (arrow 2) to instantiate paths from PPC workers. This task contains the physical path description (i.e. the operator name as well as their running location). PPE then contacts CMGR worker (arrow 3) which runs on each physical node of the path to create operator instance (if operators are to be created on the fly) and the connector between operators. This task (arrow 3) contains the name of the operator as well as its input arguments and the path identifier. There are two stages in path execution: First, a CMGR worker instantiates the operators and creates the necessary connectors. Once the entire path has been instantiated, a CMGR worker starts the operator, which begins computation and starts to receive from and send data through its connectors.

The path requestor will finally receive a completion event indicating the success or failure of the path request after a successive sequence completion events flowing from CMGR workers back to the client (arrows 4-7). If any worker fails, the underlying vSpace cluster will take care of resubmission of the task to another worker. After the retry has exceeded a specified number, a failed completion event is propagated back to the client indicating the failure and its reason. This decomposition of tasks within the APC Service maximizes the concurrency and minimizes blocking.

Each path has associated with it a unique identifier. Users of paths can send requests to APC to make changes to paths (e.g., change data output location). If any operator or connector fails, PPE will initiate repair by noticing the failure itself, or receiving a message from the corresponding CMGR worker, or receiving a repair request from CMGR workers running on neighboring operators of the failed operator. This is due to the redundant control paths mechanism (Section 4.2).

7.4 Path Search Algorithms

7.4.1 Logical Path Search

Given application-specific optimization metrics, to find an optimal logical path, APC conducts the shortest path search on a graph modeling the space of operators. Any vertex of the graph denotes a data or protocol format. Each edge represents an operator performing the translation between the two formats with a value denoting the cost of operation. Using Dijkstra's shortest path search algorithm, the running time is $O(E \log V)$, where V is the number of vertices in the graph, and E is the number of edges.

7.4.2 Physical Path Search

Subsequently, the optimal physical path can be located also using the shortest path search on a graph constructed by mapping physical machine nodes to vertices and path connectors to edges. First, the set of potential physical machines for running each operator is determined. An edge exists between two vertices representing two physical machines if they are adjacent operators in the logical path. The cost of each edge is determined by properties of the two connected operators as well as the network characteristics between the two physical nodes. To make the path search algorithm scale with the number of operators, we propose to make the search space domain specific (e.g., audio format translation, language conversion) to limit the size of the graph. Most graphs are relatively static and can be reused. Furthermore, popular search results are also cached for reuse.

7.5 Operators

For our prototype, dynamic operators are multi-threaded, inheriting from an abstract class to have desired behavior of initialization, creation, execution, maintenance, and termination. When they fail, restarting them will not cause loss of important state information. We have also built a special class of long-lived operators to wrap existing services to make use of legacy services that do not communicate directly with our connectors. Using off-the-shelf software when necessary, we have implemented at least one operator in each of the categories described in Section 3.8, shown in Table 1. Each implementation is simple and has less than 100 lines of Java code.

Each operator is strongly typed and has a description of its attributes written in XML. We choose XML since it allows encoding of arbitrary structures of hierarchically named values. In addition, there are existing well-supported software (e.g., [2]) for validating XML descriptions against well-defined schemas (Document Type Definitions). The DTD contains the following information about each operator: unique identifier, code location, functional operator classification (Section 3.8), operator categorization (according to operator properties in Section 3.7), number of inputs, number of outputs, each input's type, each output's type, acceptable connector for each input and output, cost, and special requirement (e.g., hardware or software).

Multiple operators can run on a single physical processor. To manage the resources of a single node, we establish a connection manager service on each node where operators can execute. The job of a connection manager, acting as an operator registry, also includes allowing querying of operators and connectors, uploading an operator, creating connections for an operator, and notifying failures of operators or connectors to the APC service for path repair.

Functional Classification:	Operators implemented:
Data format conversion operators:	PCM to GSM, GSM to PCM, MPEG3 to PCM, MPEG decoder, REAL encoder, Speech recognition, Speech synthesizer, G.723 to PCM
Protocol conversion operators:	SSL to simple encryption
Content transformation operators:	text summarization operator, language translation operator
Optimization operators:	FEC, compression, encryption, decryption

Table 1: **Implemented operators:** Using the current prototype of APC, we have experimented with the above set of operators.

7.6 Connectors

Each operator has a reference to an output and input connector. All connectors implement a common Java interface. To interact with previous and subsequent operators in the operator chain, each operator invokes these read and write methods of the interface to receive its input data and send its output data. TCP, UDP, and RTP connectors are supported in the current prototype. We are exploring various connector properties such as delivery order, latency, etc. Our initial taxonomy categorizes a connector in the following dimensions.

- Reliability guarantees: reliable vs. unreliable.
- In order delivery guarantees: TCP-like in order delivery vs. no ordering guarantees like UDP.
- Flow control: whether flow control mechanism exists to prevent overloading the receiver.
- Duplicate delivery: whether duplicate packets are ever delivered.
- Security levels: whether packets are encrypted.
- Blocking: whether the interface is blocking (e.g., RMI like) or nonblocking and asynchronous.

7.7 APC Example Applications

We have implemented these applications using the APC: accessing Jukebox service [14] (MP3 format) using a cell phone, accessing email through Vat [36] (an audio tool), communication between a PSTN phone and a GSM cell phone, and voice-enabled interactive room control. Most of the above applications are developed for the ICEBERG architecture [37] with the main goal of enabling any-to-any communication independent of end devices. The functionality of path allows seamless integration of any new device into the communication infrastructure [25]. It only requires the addition of an operator that converts between a supported format to the new device's data format.

Our experience shows that service composition is greatly simplified by the APC and the QoS of the resulting composed entity is guaranteed through the path search process and path runtime adaptivity. Moreover, we find the classification of operators based on the properties in Section 3.7 quite helpful during operator and path creation. Now, we briefly describe some the applications developed using the APC.

7.7.1 Listening to Jukebox MP3 Songs using Cell Phone

Ninja Jukebox is a distributed, collaborative music repository that delivers digital music in MP3 format to Internet clients in real-time. A GSM cell phone user interested in using the service can take advantage of the APC's data transformation functionality to convert the music into the right data format. Furthermore, the data path established also hides the network jitter and bandwidth fluctuations by buffering and downsampling the data.

This data path is built from existing Unix utility tools – *mpg123* and *sox* programs, and a GSM lossy speech compression codec [6]. This demonstrates that the APC can easily integrate legacy code as operators to construct data paths.

7.7.2 Universal Inbox

Universal Inbox is an infrastructure service in ICEBERG that provides customizable redirection of incoming communication based on user preference profiles as well as user's end devices. The inbox is universal because it accepts all types of communication, e.g., voice mail, paging message, email, news feeds from the Web. Depending on user's current end device, the incoming message is automatically transformed to the proper format before delivery. For example, an HTML-email message goes through a speech synthesizer and a PCM encoder before reaching a user on a PSTN phone. For Universal Inbox, the APC is the key to extensibility and service portability.

7.7.3 Interactive Voice Room Control

In this era of ubiquitous computing, it becomes important to control various appliances in smart spaces using various modes, e.g., speech, text, gesture, etc. There is such a smart space in our lab, consisting of various audio and visual appliances that can be controlled over the network. Using the APC service, we can control A/V equipment (e.g., move cameras, turn on lights, program VCR) using a variety of input devices. Paths are automatically constructed from input source to the room control application and vice versa. For instance, speech input is first converted to PCM audio, then speech-to-text conversion is performed on the output, which goes through natural language processing to the text format. The text subsequently is changed into commands accepted by the room application. Responses from the application goes through the inverse transformation: first to text, then PCM speech, and finally GSM audio if the end user is using a cell phone.

7.7.4 Real-time Streaming Video for Wireless Mobile Clients

One of the path applications we have recently developed is a video streaming service for mobile wireless clients. As a future extension, we plan to support real-time video conferencing between wired and wireless clients using a variety of end-devices. Depending on the access network bandwidth and client's end device capability, the APC automatically generates the proper sequence of transformation operators and content adaptors to generate desired data format at the proper data rate and quality. Furthermore, the constructed data path is intelligent enough to redirect data when user roams. We have built a path (shown in Figure 7) transcoding from MPEG streams to REAL format [27].

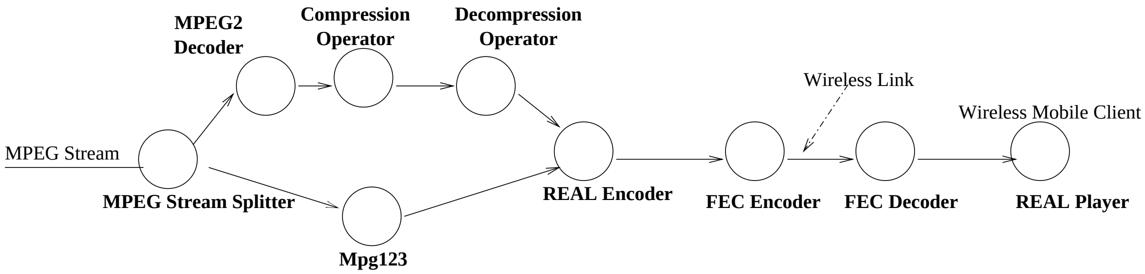


Figure 7: **Video transcoding path:** This figure shows the path transcoding from MPEG video streams to Real format delivered to a wireless mobile client running a REAL Player. This nonlinear path demonstrates the use of compression operators to reduce data size, FEC operators to reduce packet loss rate on the wireless link.

7.8 Fault Recovery Implementation: Partial Path Repair

We now discuss our implementation of path fault recovery using redundant control paths (Section 4.2). To actively detect operator failures, the APC periodically contacts all operators in running paths. This active monitoring will detect failures at both the process and machine level. In addition to active polling, passive monitoring is also used by taking advantage of the continuous data flow during execution. Any interruption of data flow can be an indication of a potential failure. Each operator can time out depending on the application-specific expected latency of receiving data from the previous operator in the chain. Furthermore, upon catching an I/O exception or getting an error message when an operator attempts to read or write, it can deduce that its connectors or neighboring operators have failed.

To achieve fast fault recovery, the following protocol we developed aims at minimizing the amount of down time of the path and its impact on the end-users. Thus, *partial path repair* is always attempted before restarting the complete path. There is usually never a need to tear down the entire path to rebuild it as long as the control path is resilient to failures.

The following discussion of path repair applies to both single and multiple operator failure. First, if failure occurred only at the process level, the failed operator is restarted on the original node. If that is unsuccessful, a new physical path is constructed to relocate the failed operator to a different processor. We always try to reuse existing running operators to avoid loading and initialization overhead. If none are found, the physical path construction process is repeated to find an optimized location to restart the failed operator. The connections between the failed operator and its neighboring operators are reestablished to resume the data flow. To minimize the amount of data lost, as soon as the path component failure is detected, the data flow is halted (i.e. buffered) when possible to avoid unnecessary loss of data. To achieve full reliability, lost data sent need to be retransmitted by the application. If the newly found physical path fails again, the logical path is rebuilt.

7.9 Path Flow Control

For real-time paths, flow control is needed to compensate for the differences in the rate at which operators produce data. This property belongs to the operator's XML description. Data is buffered for operators that generate data at a higher speed, so that slower ones will not drop packets due to buffer overflow. Furthermore, connectors exert back pressure to slow down faster operators. For operators that are relatively slow, output data should not be buffered.

8 Performance Evaluation and Analysis

In this section, we discuss performance evaluation of our prototype in the local area. We first focus on the evaluation of the APC's functionality. Thus, as example service instances, we choose null-operators which perform no operation other than copying data[3]. For all our measurements, we run the APC service and operators on a local area cluster of 400MHz Pentium-II machines each with 256MB of main memory and 512KB of processor cache with 100 megabit Ethernet connection. All the Java programs use IBM's JDK v1.1.8.

Table 3 shows good performance of a single APC service instance for a path made of 6 null-operators on 2 nodes with 200 paths constantly being created and torn down in the background. *Path creation latency* is the amount of time between path request and actual data flow. *Latency through path* denotes the time it takes for data to travel from the first to the last operator. *Fault recovery latency* in this table applies to a single operator failure. It is the time elapsed between operator failure and the restart of path data flow.

	mean	std dev.
Single-Node APC throughput	15 creations/sec	0.4
Path Creation Latency	66.3 ms	2.5
Latency through Path	5.7 ms	0.2
Fault Recovery Latency	77 ms	2.3

Table 2: **Performance** of a single APC service instance for operating on 6-nullop-paths on 2 nodes. There are 200 paths continuously being created and destroyed in the background. Results are averaged over 50 measurements.

[3] Each null operator is implemented as a process in an independent JVM. Therefore, APC path creation throughput is low due to the overhead of running these heavy-weight processes requiring large amount of memory. The APC does however support sharing of a service through multiple threads.

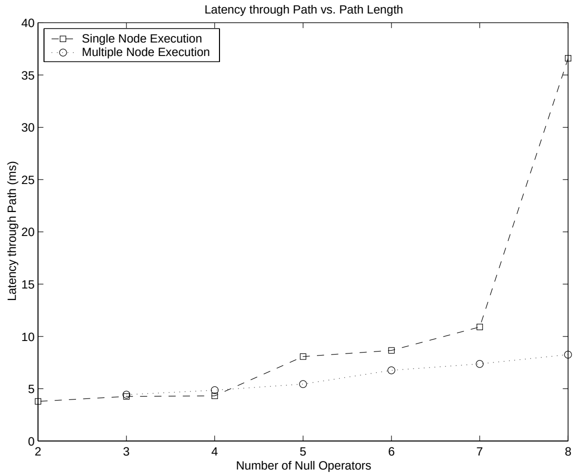


Figure 8: **Latency through Path:** This graph shows as the path length increases, lower path latency is achieved by distributing operators across different machines.

8.1 Latency through Path

It is beneficial to distribute computation across multiple physical machines for ease of scalability, as the number of operator increases. The benefit nevertheless needs to offset the increased communication overhead between processors. As Figure 8 shows, for paths of simple null operators in a cluster, after the number of operators exceeds 4, it is more beneficial to place operators on separate machines. The bottleneck here is memory space. As the number of operators on a single machine exceeds 7, thrashing starts. If we use realistic operators that actually perform computation on data, the crossing point would even be smaller due to processing power requirement. We now extrapolate the scenario to the wide area, where communication latency can be much higher. We expect the curve to be slightly shifted to the right. To justify multiple machine execution, we need longer path length for low path latency. We can conclude from these measurements that except for very short paths placing operators on separate nodes is beneficial for load balancing.

8.2 Scalability

It is important that our path construction process scales well with respect to increasing path length given fixed amount of physical resources and also with increasing number of machines for a given path. The two plots of Figure 9 confirm this claim. As the number of processors available for running operators increases, the total path creation latency decreases quickly due to parallelizing of the tasks, increase in memory and processing power. On the other hand, given fixed number of nodes, path creation latency increases only *linearly* with respect to the path

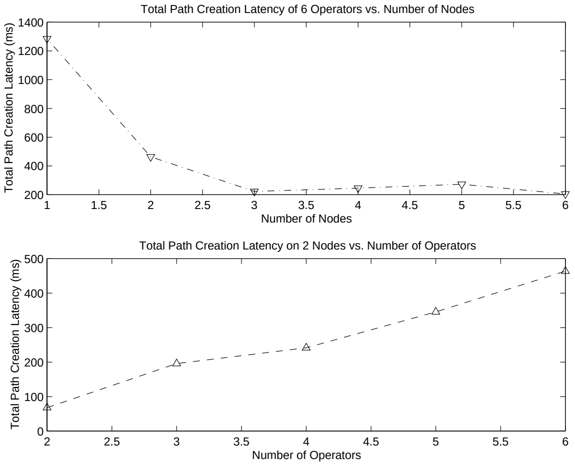


Figure 9: **Path Creation Latency of a 6-nullop-path:** As the number of available machines to run operators increases, the overhead is significantly reduced. The slight increase from 3 to 5 nodes is an artifact due to better cache locality when more than one null operator runs on the same node.
Path Creation Latency of on Two Nodes: it increases linearly with path length provided that there are enough memory and processing capacity.

length.

The path construction process can be broken down into the following steps. (See Figure 10 and Figure 11). Figure 10 examines for a fixed path length of 6 operators, how various path creation components scale with respect to the number of physical machines available to execute the path. Figure 11 studies the path creation latency breakdown on two physical machines as the number of operators in the path varies.

1. **Logical Path Creation Latency** (Figure 10-A, 11-A) is relatively constant due to fixed search space of operators in this case. In general, as the number of operators (V) increases, the search is $O(E \log V)$.

2. **Physical Path Creation Latency** (Figure 10-B, 11-B) is also roughly constant due to fixed amount of physical resources. Using the shortest path graph search, the running time is $O(E \log V)$ (V: number of physical machines, E: lengths of path). Both latencies can be reduced by reusing search results for logical and physical paths.

3. **Operator Instantiation Latency** (Figure 10-C, 11-C) increases linearly with the number of operators before saturation. This is because more operators in the path imply more processing is necessary. Due to cache locality behavior, operator instantiation latency is very small for 3 node case when the number of operators running on each node is maximized to 2. This is an artifact because the same operator is used in the path. However, in general,

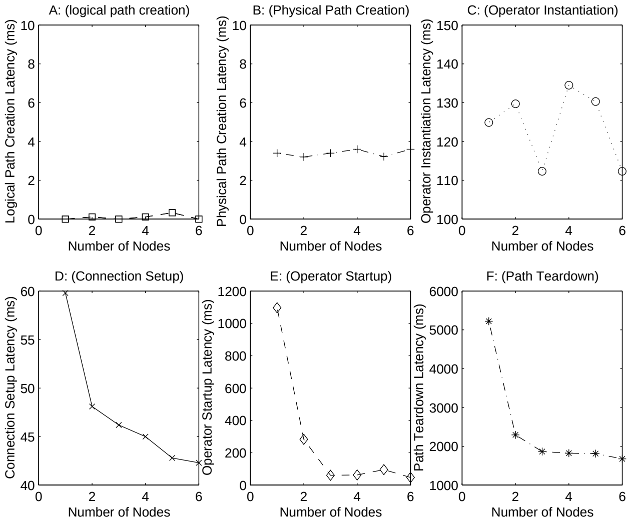


Figure 10: **Path Creation Latency Breakdown of a 6-Operator Path:** The first five plots show how each detailed step of path creation process behaves with increasing physical resources. The last plot shows path tear down latency decreases with increasing number of nodes.

for a fixed path length of different operators, operator instantiation latency is independent of the number of machines as long as there are enough memory and processing capacity. This is because operators in a path are instantiated in parallel.

4. **Connection Setup Latency** (Figure 10-D, 11-D) decreases significantly with increasing number of nodes. This is because creating multiple socket connections on a single machine is expensive due to processing-bound kernel operations. Distributing connection creations across multiple machines reduces the load on each individual machine. The connection setup latency increases linearly with number of operators due to the need to setup more connections.

5. **Operator Startup Execution Latency** (Figure 10-E, 11-E) has similar behavior as connection setup latency. Operator startup is also a processing-bound operation; thus, distributing it across different machines speeds it up. As path length increases, the number of operators that need to start increases as well. Consequently, the total operator startup latency increases with path length given fixed physical resources. *Path teardown latency* shown in Figure 10-F and Figure11-F has similar behavior.

We can conclude from this set of measurements that to guarantee good performance, it is crucial to determine whether the computation performed is CPU-, memory- or I/O-bound. We must load balance accordingly by either distributing the computation across different machines or collocating neighboring operators to reduce network bandwidth requirement. The set of bench-

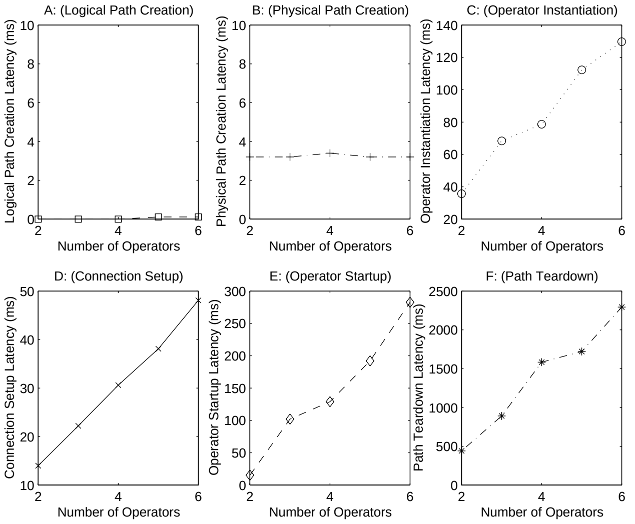


Figure 11: **Path Creation Latency Breakdown of on Two Nodes:** The first five plots show how each detailed step of path creation process behaves with increasing path length. The last plot shows path tear down latency increases with path length as expected.

marks presented above are useful for service authors to characterize the relative performance characteristics of their operators.

8.3 End-to-end performance numbers

To demonstrate the usability of the APC service, we now present some end-to-end performance measurements for one path application (Section 7.7.1)– accessing a MP3 streaming Jukebox service using a cell phone. Table 3 shows good performance of a single APC service instance for a path consisting of 4 operators on 2 nodes with 200 paths continuously being created and torn down in the background. The performance shown is acceptable because the response time for a path creation is less than 500ms. Users typically do not care about how long it takes for the service session to terminate. In this case, it takes less than 300ms. Recovery takes slightly longer (i.e. 400ms); however, if buffering is used, the user can hardly notice any gap in the output audio. The scalability of path is also reasonable – 16 paths per machine, 15 creations per second. This means that to handle 1000 service creations per second, we need 67 machines, while 200,000 service sessions are in progress.

In the context of two-way telephone calls, statistics [21] show that during busy hours, the average call arrival rate $R = 2.8\ calls/hour/user \times N$ (N is the number of users in the system), with the call duration $t = 2.6\ minutes$. From our measurements, we know that the rate of path creation is 15 $paths/sec$ with 32 paths running in the background. The call arrival rate a two-node APC can handle is therefore given by $32/(2.8 * 60) = .19\ call/sec$. Thus, the system can handle $N = .19/(2.8\ calls/hour) = 244$. Therefore, a two-node APC service can easily handle over 200 users for this type of transcoding operators: sound conversion operations and GSM to PCM codecs.

Logical and physical path creation time:	264ms
Path instantiation time:	215ms
Path teardown time:	289ms
Path recovery from one failed operator:	402ms
Data throughput:	64kbps
Path construction latency:	479ms
Path scalability:	32 simultaneous paths
Single-Node APC throughput	15 creations/sec

Table 3: **Performance** of a single APC service instance for operating on 4-operator-paths on 2 nodes. There are 200 paths continuously being created and destroyed in the background on other nodes. Results are averaged over 50 measurements.

9 Future Work and Conclusions

So far we have only deployed our prototype in the local area. We are yet to measure the performance of paths that span across multiple clusters, although our implementation has built-in mechanisms for cluster level failure detection and recovery. We plan to use remote sites from our Swedish collaborators to test out wide area paths and also use the emulation testbed provided by University of Utah [24]. Additionally, we plan to build more path applications to make APC service general for all types of Internet service composition. We would like to explore dynamic path adaptation and performance tuning, and other optimizations such as caching and reusing path results.

In conclusion, we presented the design and implementation of a service composition platform that automates compositions of both legacy and new Internet services across the wide area. Automation is enabled through a strong typing system and the encoding of service attributes through flexible XML descriptions. By providing features of fault-tolerance, scalability, and optimized adaptive resource utilization, we allow service authors to focus on the specific content of their services rather than how the service will be accessed by different devices and how it will interoperate with other services. We achieve fault-tolerance through redundant control paths responsible for fast fault-recovery. Scalability is leveraged from using a cluster computing platform. Optimized resource utilization and differentiated QoS are obtained through the iterative path construction process with continuous feedback and a clear specification of the user's optimization criteria. Moreover, we identify two key abstractions (i.e., operator and connector) useful for putting together services in a tinkertoy-like fashion and for service component reuse. Another contribution of our work is to present the key set of properties of operators or Internet services that need to be considered to achieve high application-specific QoS in the resulting service composition entity.

We validated our claim of easy service compositions by presenting numerous applications built using our prototype APC service. In the context of the ICEBERG project, we demonstrated that the APC provides extreme ease in integrating any new devices into the communication infrastructure enabling true any-to-any communication paradigm. Finally, we justified our design goals by demonstrating good scaling performance of the APC facility in the local area.

10 Acknowledgement

I thank Professor Randy Katz, my advisor for his guidance and encouragement since I joined the ICEBERG research group. I also thank Professor Brewer and Professor Culler for their insightful comments and ideas. I am especially grateful for Professor Brewer who first initiated the path concept in the Ninja project. I owe a great deal to members of the Ninja and ICEBERG project for their input.

References

[1] Charles Brooks ad Murray S. Mazer, Scott Meeks, and Jim Miller. Application-Specific Proxy Servers as HTTP Stream Transducers. In *Proceedings of the Fourth International World Wide Web Conference*, December 1995.

[2] IBM AlphaWorks. Xml parser for java. `http://alphaworks.ibm.com/tech`, September 2000.

[3] T. E. Anderson, D. E. Culler, and D. Patterson. A Case for NOW (Network of Workstations). *IEEE Micro*, February 1995.

[4] D. Clark and D. Tennenhouse. Architectural Consideration for a New Generation of Protocols. In *Proceedings of SIGCOMM '90*, Philadelphia, PA, 1990.

[5] S. Czerwinski, B. Zhao, T. Hodes, A. Joseph, and R. Katz. An Architecture for a Secure Service Discovery Service. In *Fifth Annual International Conference on Mobile Computing and Networks (MobiCOM '99)*, August 1999.

[6] Jutta Degener. Gsm 06.10 lossy speech compression. `http://kbs.cs.tu-berlin.de/~jutta/toast.html`.

[7] G. Eddon and H. Eddon. *Inside Distribued COM.* Microsoft Press, Redmond, WA.

[8] WAP Forum. *Wireless Application Protocol (WAP) Forum.* `http://www.wapforum.org`.

[9] `http://www.foundrynet.com/appNotes/traffic.html/`.

[10] Armando Fox, Steven D. Gribble, Eric A. Brewer, and Elan Amir. Adapting to client variability via on-demand dynamic distillation. In *Proceedings of the ACM Seventh International Conference on Architectural support for Programming Languages and Operating Systems*, October 1996.

[11] Armando Fox, Steven D. Gribble, Yatin Chawathe, and Eric A. Brewer. Extensible Cluster-Based Scalable Network Services. In *Proceedings of the 16th ACM Symposium on Operating Systems Principles (SOSP-16), St. Malo, France, October 1997.*, 1997.

[12] A. Frier, P. Karlton, and P. Kocher. *The SSL 3.0 Protocol.* `http://www.netscape.com/eng/ssl3/ssl-toc.html`, March 1996.

[13] Vijay Srinivasan George Swallow. Multiprotocol label switching (mpls). `http://www.ietf.org/html.charters/mpls-charter.html`, Nov. 17th, 2000.

[14] Ian Goldberg, Steven D. Gribble, David Wagner, and Eric A. Brewer. The Ninja Jukebox. In *Proceedings of the 2nd USENIX Symposium on Internet Technologies and Systems, Boulder, CO.*, October 1999.

[15] Steven D. Gribble, Eric A. Brewer, Joseph M. Hellerstein, and David Culler. Scalable, Distributed Data Structures for Internet Service Construction. In *To Appear in OSDI 2000*, 2000.

[16] Caltech Infospheres group. Caltech Infospheres Project. Joint W3C/OMG Workshop on Distributed Objects and Mobile Code, June 1996.

[17] Xia Hong. Personal acitivity coordinator: A coordination layer for independent services. Master's thesis, U.C. Berkeley, December 1999.

[18] Hewlett Packard Inc. *eSpeak: The Universal Language of E-Serices.* `http://www.e-speak.net/`.

[19] A. D. Joseph, B. Hohlt, R. H. Katz, and E. Kiciman. System support for multimodal information access and device control. Workshop on Mobile Computing Systems and Applications (WMCSA), 1999.

[20] M. Liljeberg and et al. Enhanced Services for World Wide Web in Mobile WAN Environment. Technical Report C-1996-28, University of Helsinki CS Department, April 1996.

[21] C. N. Lo, R. S. Wolff, and R. C. Bernhardt. An Estimate of Network Database Transaction Volume to Support Universal Personal Communications Services. In *First International Conference on Universal Personal Communications (ICUPC '92)*, 92.

[22] Sun Microsystems. *Jini Connection Technology.* http://www.sun.com/jini/.

[23] D. Mosberger and L. Peterson. Making Paths Explicit in the Scout Operating System. In *Proceeds of OSDI'96*, 1996.

[24] University of Utah. Utah network testbed operations. http://www.emulab.net/.

[25] Bhaskaran Raman, Randy H. Katz, and Anthony D. Joseph. Universal Inbox: Providing Extensible Personal Mobility and Service Mobility in an Integrated Communication Network. WMCSA 2000.

[26] Real.com, http://service.real.com/help/library/blueprints/8codecs/producer8codecs%.html. *Working with RealProducer 8 codecs*, June 28, 2000.

[27] http://www.realplayer.com/.

[28] Steven J. Ross. A proxy for secure multi-modal access to internet services from post-pc devices. Master's thesis, U.C. Berkeley, 2000.

[29] Y. Sato. DeleGate Server. http://wall.etl.go.jp/delegate/, March 1994.

[30] M.A. Schickler, M.S. Mazer, and C. Brooks. Pan-Browser Support for Annotations and Other Meta-Information on the World Wide Web. In *Fifth International World Wide Web Conference (WWW-5)*, May 1996.

[31] B. Schilit and T. Bickmore. Digestor: Device-Independent Access to the World Wide Web. In *Sixth International World Wide Web Conference (WWW-6)*, Santa Clara, CA, April 1997.

[32] The Common Object Request Broker Architecture. *The Object Management Group (OMG).* http://www.corba.org.

[33] http://iceberg.cs.berkeley.edu/. The iceberg project.

[34] http://millennium.berkeley.edu/. The millennium project.

[35] http://ninja.cs.berkeley.edu/. The ninja project.

[36] http://www-nrg.ee.lbl.gov/vat/. *VAT Mbone Audio Conferencing Software.*

[37] Helen J. Wang, Bhaskaran Raman, Chen nee Chuah, and et al. Iceberg: An internet-core network architecture for integrated communications. *IEEE Personal Communications*, August 2000.

[38] David J. Wetherall, John Guttag, and David L. Tennenhouse. ANTS: A Toolkit for Building and Dynamically Deploying Network Protocols. In *Proceedings of IEEE OPENARCH'98*, San Francisco, CA, April 1998.

[39] Rich Wolski, Neil T. Spring, and Jim Hayes. The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing. *Journal of Future Generation Computing Systems*, 1999.

[40] Ka-Ping Yee. Shoduoka Mediator Service. `http://www.shoduoka.com`, 1995.

[41] Bruce Zenel and Dan Duchamp. A General Purpose Proxy Filtering Mechanism Applied to the Mobile Environment. In *Thrid Annual ACM/IEEE Conference on Mobile Computing and Networking (Mobicom'97)*, New York, NY, 1997. ACM.