A Framework for Application-Specific Customization of Network Services

Sriram Ramabhadran and Joseph Pasquale

Department of Computer Science and Engineering
University of California, San Diego

{sriram , pasquale}@cs.ucsd.edu

Abstract

We propose a network service framework where common network functions such as routing and multicast can be customized on a per-application basis. Network service customization is achieved through service plug-in modules that can be dynamically loaded by applications. In addition to their customization, services can also be composed to form complex aggregate services. Finally, our framework is deployed using an overlay network infrastructure.

1. Introduction

The increasing heterogeneity of the Internet in terms of networking technologies and client devices raises significant engineering challenges for the development of wide-area applications. The process of building robust and adaptive Internet applications is highly non-trivial, and requires careful design and implementation. In this paper, we propose a middleware–based system that facilitates this process. Specifically, we propose a framework for application–specific customization of network services.

Two distinct aspects of the development of Internet applications motivate the framework. First, many network applications use certain common network-based functions like data transport (unicast and multicast), caching, lookup and others. Our framework tries to abstract out these common functions in the form of services that can be used by application developers as basic building blocks to construct more sophisticated services. Although such services are generic enough to be of utility to several applications, there is still benefit in modifying these services in application-dependant ways. The increasing diversity of Internet applications with vastly different requirements demands that such services be customizable.

The second factor is the necessity of applications to adapt to changes in network properties such as latency, bandwidth and loss rate, which may exhibit both spatial and temporal variation. There has been considerable research [1,3,4,5] in application adaptation. In this context, embedding application-aware computation in the network has been proposed as key mechanism to enable adaptation. Our framework extends work in application adaptation by providing the applications the ability to customize network functionality. Effective adaptation demands exposing the network to applications instead of isolating it purely as a communication substrate.

One approach to customizing network services for applications is to make them parameterized, with the application instantiating a specific version of the service by supplying the relevant parameters. We believe this approach, though feasible, is not expressive enough because it limits the range of application customizations possible. What is required is a sufficiently expressive, yet simple, way to customize network services.

In this paper, we propose a service model in which services can be customized by dynamically loading policy modules called *service plug-ins*. In contrast to the limited nature of parameter-based customization, server plug-ins can be used to specify richer forms of customizations. Our model also permits applications to compose services to form an aggregate service. We also propose an overlay network infrastructure to deploy these services.

The remainder of the paper is organized as follows. Section 2 motivates the need for application-specific customization of network functionality through illustrative examples. Section 3 describes a framework that enables the development and deployment of these services. Section 4 describes multicast as an example of an application-specific network service. Section 5 surveys related work and Section 6 concludes.

Proceedings of the Fourth Annual International Workshop on Active Middleware Services (AMS'02)
0-7695-1721-8/02 $17.00 © 2002 IEEE

2. Examples of Network Services

Consider routing of application data as a basic service provided by the network. Routing in the Internet is often based on hop count within a routing domain, and on administrative policy between routing domains. This can result in end-to-end data paths being significantly sub-optimal, as applications have little or no control over data paths established between application endpoints. An application-specific routing service would provide applications the ability to establish paths based on metrics important to the application. A real-time application such as Internet telephony would conceivably choose paths optimized for low network latency. A bulk data transfer application such as a large FTP, on the other hand, would conceivably choose paths optimized for high bandwidth. Other applications may wish to implement a highly specific routing policy. An example of such a policy is onion routing [11], which makes anonymous the communication between endpoints by routing through a series of intermediaries.

As another example, consider multicast as a network service. For various reasons, IP multicast, the Internet's native multicast infrastructure, lacks widespread deployment especially in the wide area. Application-level multicasting using overlays has been proposed as an alternative [10,13]. Although multicast as a service is potentially useful to several applications, the exact service requirements can differ sharply. For example, it is possible to construct the multicast distribution tree based on different metrics. Applications differ on the extent of loss that can be tolerated, and hence make different tradeoffs between the required level of reliability and the cost of implementing it.

Many applications could benefit from the ability to customize network functions like routing and multicast. The same is true of application layer services such as caching, lookup, and network storage, that are not core network functions, but are network–based functions useful to many applications. A more flexible framework in which applications can easily customize these services is required. In the next section, we describe such a framework.

3. Framework

In this section, we describe an architectural framework for application-specific customization of network services. First we describe the service model for the creation and customization of services, and then the infrastructure for the deployment of these services.

3.1 Service Model

In our framework, a network service is an abstraction for some specific network-based function that applications require. As previously mentioned, examples include routing, caching, multicast, lookup, and network storage. Services may also include transformational operations on application data inside the network, such as image filtering and compression, and other application adaptations.

Each service is associated with certain semantics, which are closely tied to its function. However within the semantics of the service, there may be certain operations that applications would require to be implemented in a customized manner. For example, a routing service determines, given a destination node, the next hop node in the network. But the procedure to determine the next hop node could be different for different applications. As another example, a caching service stores application objects for future retrieval. However, applications could implement different replacement policies and different procedures to determine when a particular object becomes stale in the cache. In each case, the semantics of a service are well defined, but there are certain operations in the service that may vary from application to application. This is the basis of our customization architecture for services.

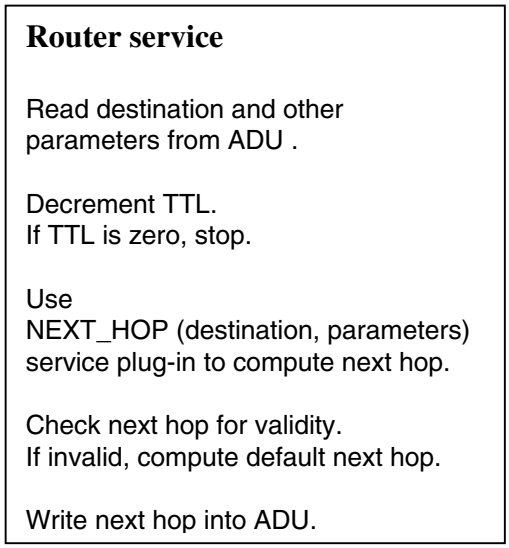

Figure 1. Service definition

Each service is associated with a service definition, which contains the code that implements the service. This code is not fully specified, but uses abstract components called *service plug-ins*. A service is instantiated by

binding the service code with specific implementations of each server plug-in used by the service. Applications customize services by providing appropriate implementations of the server plug-ins for a service, as specified by its definition. Figure 1 shows pseudo–code for a routing service, which uses a single plug-in to determine the next hop in the network. These server plug-ins are strongly-typed entities that are accessed through a well–defined programmatic interface. This enables them to be dynamically bound to the service code during service instantiation. This interface is also specified in the service definition. Service plug-ins can also be viewed as policy modules that express arbitrary application-specific policies. For example, in the case of the routing service, the plug-in can not only implement routing based on different metrics, but also a more sophisticated routing strategy such as onion routing.

This service model has several advantages. It provides applications with the ability to use standard services, and yet customize them with minimal effort. It encourages modularity and reuse. Our service model also has advantages in terms of protection from malicious applications. Services are meant to be higher privileged entities than service plug-ins. Therefore services may have access to system resources such as permanent storage and the underlying network, whereas service plug-ins may not. This limits the amount of damage that can be done by a malicious or buggy service plug-in. In addition, the service code can enforce safety checks on the behavior of a server plug-in using reasonable defaults if the output is determined to be erroneous. For example, a malicious service plug-in for a routing service can cause data to circulate in the network forever, thereby consuming network bandwidth. This can be prevented by the routing service by using a TTL counter that is decremented at each hop independent of the plug-in.

One potential drawback of our service model is that by fixing the semantics of a service a priori in terms of service code, we limit the range of customizations possible to those that can be expressed through service plug-ins. However our focus is on specific network services like routing and caching, which have relatively well–defined semantics. Therefore we do not expect that this is a serious limitation of our approach.

In addition to customization of services, our architecture also permits composition of multiple services. The output of one service may be used as input for another service, thereby forming a *service pipeline*. Figure 2 shows a schematic diagram of a service pipeline. This composition is made possible by a standard interface for all services. Every service takes an application data unit as input and returns as output possibly multiple application data units. Service composition provides applications the ability to construct sophisticated services from relatively simple services. We provide an example of this in Section 4.

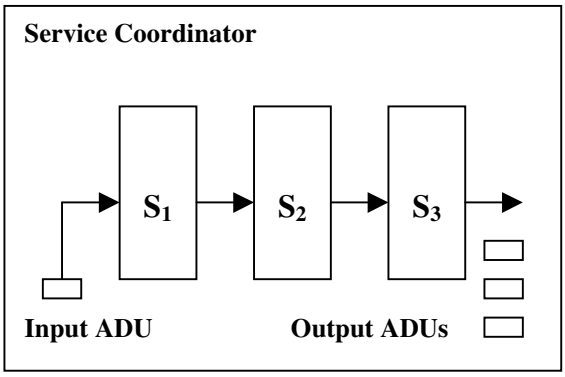

Figure 2. Service composition

3.2 Network Infrastructure

The network infrastructure to deploy these services consists of a system of servers placed at various points in the network. These servers form an overlay network, a virtual topology implemented by application-level routing over the actual IP network of the Internet. Overlay networks have been used to provide increased robustness [6], to support wide–area multicast [10,13], and as a testbed for active networks. We use an overlay-based infrastructure for deploying application-specific network services. Applications connect to one of the servers in the infrastructure through a client library. All communication between application endpoints is routed through the overlay network.

The overlay-based approach has a number of advantages. The servers in the overlay network provide a natural location for hosting application-specific services. Unlike the active network architecture, the framework operates at the application layer, and avoids any change to the Internet routing infrastructure. This makes the framework safe and practical, and thereby mitigates a formidable barrier to deployment. Overlay networking also provides a natural mechanism to establish application-specific data paths in the network, in contrast to routing in the Internet where data paths are often beyond the control of applications.

3.3 Service Execution Environment

The execution environment at each server in the system consists of a Service Coordinator and a Network Coordinator. The Service Coordinator implements the service model described earlier, while the Network

Coordinator provides the network interface to executing services. Figure 3 show a schematic diagram of the execution environment at a server.

The Service Coordinator is responsible for instantiating services, customizing them and composing them into pipelines. Each application data unit contains a *service manifest*, which is a list of services that are to be composed together for processing that application data unit. The service manifest also names service plug-ins and parameters for each service listed. For each service listed, the Service Coordinator instantiates the service and dynamically binds it to any service plug-ins specified. The service manifest may either contain code for a service plug-in or may simply name its location. In the latter case, the Service Coordinator must first obtain the code. Service plug-ins are cached for further use. The Service Coordinator then composes the services to form a pipeline.

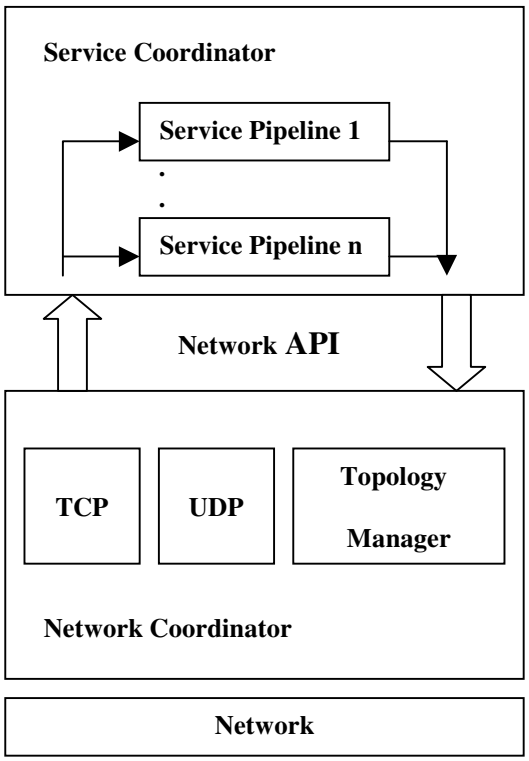

Figure 3. Service execution environment

The Network Coordinator provides network access with standard interfaces to common transport protocols such as TCP and UDP. All network communication by the Service Coordinator and services takes place through the Network Coordinator. In addition, it also implements a Topology Manager, which is responsible for maintaining the overlay network. Servers are organized into a mesh, in which each server maintains virtual links to at least k other servers. The parameter k is chosen to keep the network connected and resilient to failures. The Topology Manager maintains state about the latency, bandwidth, and loss rates of virtual links in the network, and monitors the other servers for failure. This information is available to services through a well-defined programmatic interface.

Application Level Framing [2] is used to structure application communication into application data units. An Application Data Unit (ADU) is an abstract unit of application communication. For Web applications, this would be a HTTP requests/responses, while for a video application, this would be an MPEG frame. By operating at the application layer, the framework admits a richer set of services such as caching and data type-specific distillation [15].

4. Application Specific Multicast

Recent research [10,13] has suggested overlay networking as a mechanism for implementing multicast as an application-level service. Nodes placed at strategic locations in the network organize themselves into a multicast distribution tree, in which data is forwarded between nodes using unicast channels. In this section, we describe *application-specific multicast*, a way in which applications can construct a multicast service suited to their requirements. In particular, we show how service customization and service composition can be used for this purpose.

To illustrate how different applications can construct a multicast service in different ways, consider the following two applications. The first application is a real-time multimedia multicasting application such as videoconferencing. The second application is a bulk data distribution application in which large volumes of data are to be reliably transmitted to multiple destinations. The first application is delay sensitive, but is willing to tolerate loss of data to some degree. The second application is bandwidth intensive and requires reliable transmission of the data.

An overlay multicast service has the following semantics. On receiving an ADU, the service must determine which of it neighbors to forward it to. For each such neighbor, it generates a duplicate copy of the ADU. Just like the router service used a plug-in to determine the next hop, the multicast service uses a plug-in for this purpose. The first application would specify a plug-in that optimized the construction of the multicast tree using delay as the primary metric. Since occasional loss can be tolerated, the plug-in can constantly improve the multicast

tree in response to variations in link delays. The unicast transport protocol used for forwarding could be UDP as reliable delivery is not required. The second application would specify a plug-in that optimized the construction of the multicast tree using available bandwidth as the primary metric. Since loss is not tolerated, the plug-in would not change the multicast tree unless the topology of the overlay network changes. The unicast transport protocol used for forwarding would be TCP as reliable delivery is required. Thus plug-in-based customization of services helps two very different applications implement specialized versions of the same service.

In addition, service composition can be used to further enhance the operation of the multicast service. By composing the multicast service with a multimedia filtering service, the videoconferencing application can construct a multicast service that filters the video based on the bandwidth available on each output link. Since the second application requires reliable delivery of data, the multicast service can be composed with a caching service that stores that data. If loss due to change in topology occurs, data from the cache can be retransmitted. The service pipelines for the two applications are shown in Figure 4. The transport protocols are not part of the service pipeline but are shown in the figure as an example of how the same service can be used with different transport protocols by different applications.

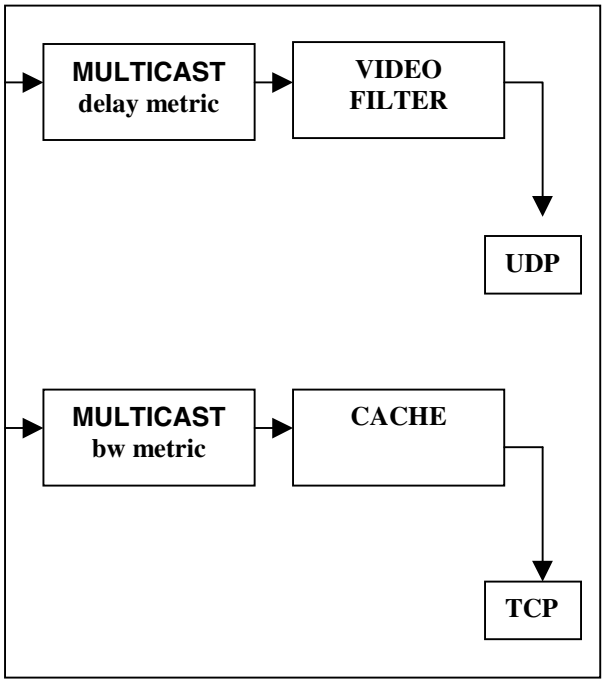

Figure 4. Aggregate multicast services

5. Related Work

The Active Network initiative [1,14] has similar goals of enabling the dynamic customization of network protocols. It involves presenting the network as a fully programmable computational environment in which arbitrary application defined code can be run on the routers within the network. The concept of slots proposed in [14] is very similar to that of a service plug-in. However, our work differs from that of Active Networks in two significant respects. First, we an overlay approach in which application-specific computation runs on end systems as opposed to routers. Therefore it involves no fundamental change in the architecture of the Internet. Second, our infrastructure operates at the application layer where the unit of processing is the application data unit and not at the network layer where the unit of processing is the IP packet.

Active services [3] and application-level active networking [4] are research initiatives that seek to limit user computation in the network to the application layer. These projects focus on application adaptation by embedding application-specific computation in the network. Conductor [5] is a network infrastructure for distributed adaptation. Conductor has a proxy node architecture, similar to that of our overlay network of servers, over which adaptor modules are deployed. Our work is more focused on customizing a predefined set of services in application-specific ways. In that sense, our work is complementary to these projects in that we focus on a different kind of application adaptation.

Several other projects like CANS [7], APC [8] and Ninja [9], propose architectures for composable Internet services. These projects are mainly concerned with a composition model for constructing application-level services.

RON [6] deals with overlay routing and mentions routing based on application-specific metrics as one of its goals. Our work takes this one step further in using an overlay infrastructure as a general framework for application-specific customization of other network services as well.

6. Conclusion

We argued for the need for application-specific customization of network functionality to better support the increasing diversity of demands of Internet-based applications. We described a framework for the development and deployment of services that embody common types of network functionality, and how they can be customized using service plug-in modules. Beyond the customization of basic services, more complex aggregated

services are easily defined by composing existing ones. An overlay network approach is adopted for practical and safe deployment of our framework.

We are currently developing a Java-based prototype of the framework, with routing and multicast as the initial target services. We plan to subsequently report on our experiences with the prototype, and to extend the prototype to other network-based services such as caching, lookup, content distribution, and network storage.

Acknowledgements

This research was supported by research grants from AFOSR, DARPA, and a NSF Research Infrastructure grant as part of the UCSD Active Web project.

References

[1] David Tennenhouse and David Wetherall, "Towards an Active Network Architecture," *Computer Communication Review*, Vol. 26, No. 2, April 1996.

[2] David Clark and David Tennenhouse, "Architectural Considerations for a New Generation of Protocols," *Proceedings of ACM SIGCOMM*, Philadelphia, USA, 1990.

[3] Elan Amir, Steven McCanne and Randy Katz, "An Active Service Framework and its Application to Real-time Multimedia Transcoding," *Proceedings of ACM SIGCOMM*, Vancouver, British Columbia, 1998.

[4] Michael Fry and Atanu Ghosh, "Application Level Active Networking," *Computer Networks*, Vol. 31, No. 7, 1999.

[5] Mark Yarvis, Peter Reiher, Kevin Eustice and Gerald Popek, "Conductor: Enabling Distributed Adaptation," *UCLA Tech. Report CSD-T010025*, June 2001.

[6] David Anderson, Hari Balakrishnan, Frans Kaashoek, and Robert Morris, "Resilient Overlay Networks," *Proceedings of 18th ACM SOSP*, Banff, Canada, October 2001.

[7] X. Fu, W. Shi, A. Akkerman and V.Karemcheti, "CANS: Composable, Adaptive Network Services Infrastructure," *Proceedings of USENIX Symposium on Internet Technologies and Systems (USITS*), March 2001.

[8] Zhuoqing Morley Mao, Eric Brewer and Randy Katz, "Fault-tolerant, Scalable, Wide-area Internet Service Composition," *UCB Tech. Report CSD-1-1129*, January 2001.

[9] Steven Gribble, Matt Welsh, Rob von Behren, Eric Brewer, David Culler, N. Borisov, S. Czerwinski, R. Gummadi, J. Hill, Randy Katz, Z. M. Mao, S. Ross, and B. Zhao, "The Ninja Architecture for Robust Internet-Scale Systems and Service," *to appear in a Special Issue of Computer Networks on Pervasive Computing*.

[10] John Jannotti, David K. Gifford, Kirk L. Johnson, M. Frans Kaashoek, and James W. O'Toole, Jr., "Overcast: Reliable Multicasting with an Overlay Network," *Proceedings of OSDI*, San Diego, October 2000.

[11] Michael G. Reed, Paul F. Syverson, and David M. Goldschlag, "Anonymous Connections and Onion Routing," *IEEE Journal on Selected Areas in Communication: Special Issue on Copyright and Privacy Protection*, 1998.

[12] Bruce Zenel and Dan Duchamp, "A General Purpose Proxy Filtering Mechanism Applied to the Mobile Environment," *Proceedings of ACM Mobicom*, New York, 1997.

[13] Yang-hua Chu, Sanjay G. Rao, Srinivasan Seshan and Hui Zhang, "Enabling Conferencing Applications on the Internet using an Overlay Multicast Architecture," *Proceedings of ACM SIGCOMM*, San Diego, August 2001.

[14] S. Merugu, S. Bhattacharjee, Y. Chae, M. Sanders, K. Calvert and E. Zegura, "Bowman and CANEs: Implementation of an Active Network," *Proceedings of the 37th Annual Allerton Conference*, Monticello, IL, Sept.1999.

[15] Armando Fox and Eric A. Brewer, "Reducing WWW Latency and Bandwidth Requirements by Real-Time Distillation," *Proceedings of the Fifth International World Wide Web Conference*, Paris, May 1996.