Service Clouds: Distributed Infrastructure for Adaptive Communication Services

Farshad A. Samimi, Philip K. McKinley, S. Masoud Sadjadi, Chiping Tang, Jonathan K. Shapiro, and Zhinan Zhou

***Abstract*— This paper describes *Service Clouds*, a distributed infrastructure designed to facilitate rapid prototyping and deployment of adaptive communication services. The infrastructure combines adaptive middleware functionality with an overlay network substrate in order to support dynamic instantiation and reconfiguration of services. The Service Clouds architecture includes a collection of low-level facilities that can be invoked directly by applications or used to compose more complex services. After describing the Service Clouds architecture, we present results of experimental case studies conducted on the PlanetLab Internet testbed alone and a mobile computing testbed.**

***Index Terms*— Adaptive communication, self-managing system, overlay network, service composition, mobile computing, autonomic computing, quality of service.**

I. Introduction

COMPUTER applications play an increasing role in managing their own execution environment. This trend is due in part to the emergence of autonomic computing [1], [2], where systems are designed to adapt dynamically to changes in the environment with only limited human guidance. One area in which adaptive behavior can be particularly helpful is communication-related software: self-adaptation can be used to improve quality of service (QoS), support fault tolerance, and enhance security in the presence of changing network conditions. This paper presents the design and evaluation of an extensible, distributed infrastructure to support the development and deployment of enhanced communication services.

Realizing adaptive behavior typically requires cooperation of multiple software components. This cooperation may be *vertical*, involving different system layers, or *horizontal*, involving software on different platforms. Many approaches to vertical cooperation are based on adaptive middleware [3]–[6]. Middleware is an ideal location to implement many types of self-monitoring and adaptive behavior. Examples of horizontal cooperation include dynamic composition of services among multiple nodes [7]–[9] and transcoding of data at intermediate nodes [10]. Many approaches involve the use of *overlay networks* [11], in which end hosts form a virtual network atop a physical network. The presence of *hosts* along the paths between endpoints enables intermediate processing of data streams, without modifying the underlying network protocols or router software.

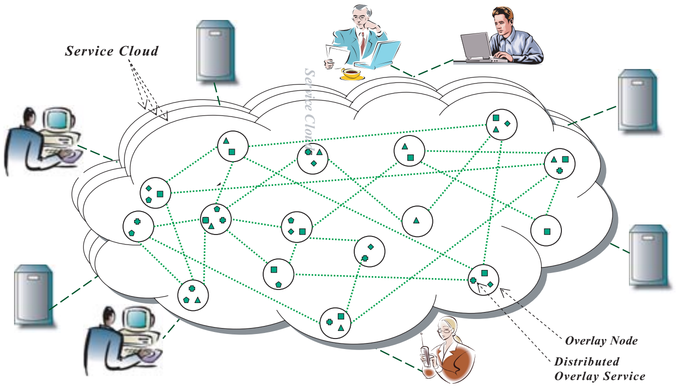


Fig. 1. Conceptual view of the Service Clouds infrastructure.

Designing systems involving both vertical and horizontal cooperation is challenging due to the dynamic nature of adaptive software, uncertainty in the execution environment, and heterogeneity among software modules and hardware platforms. We argue that a general, extensible infrastructure to support such interactions can be very useful to the development and deployment of autonomic communication services. In this paper, we describe Service Clouds, an architecture for organizing constituent service components and using them to compose complex communication services. Service Clouds is part of RAPIDware, an ONR-sponsored project investigating the design of high-assurance adaptable middleware [12].

Figure 1 shows a conceptual view of Service Clouds. Effectively, the nodes in the overlay network provide a “blank computational canvas” on which services can be instantiated as needed by user applications, and later reconfigured in response to changing conditions. Individual nodes in the cloud use adaptive middleware and cross-layer collaboration to support self-configuring behavior; an overlay network among these nodes serves as a vehicle to support cross-platform adaptation. The main contribution of this work is to propose a model for building such an infrastructure, including identification of the main components and their interaction. In addition, we describe an extension of the Service Clouds model to support

Manuscript received August 31, 2006; revised February 1, 2007; accepted August 1, 2007. The associate editors coordinating the review of this paper and approving it for publication were A. Keller and J. P. Martin-Flatin.

F. Samimi and P. McKinley are with the Department of Computer Science and Engineering, Michigan State University (e-mail: {farshad, mckinley}@cse.msu.edu).

S. M. Sadjadi is with the School of Computing and Information Sciences, Florida International University (e-mail: sadjadi@cs.fiu.edu).

C. Tang is with Microsoft Corporation (e-mail: tangcp28@gmail.com).

J. Shapiro is with Adverplex (e-mail: jshapiro@cse.msu.edu).

Z. Zhou is with Samsung Corporation (e-mail: z.zhou@samsung.com).

This research was conducted while C. Tang, J. Shapiro, and Z. Zhou were at Michigan State University.

Digital Object Identifier 10.1109/TNSM.2007.070901.

robust mobile computing at the wireless edge of the Internet. Finally, we have implemented a prototype of Service Clouds and used it to compose autonomic communication services atop both the PlanetLab Internet testbed [13] and a local mobile computing testbed.

This paper is organized as follows. After discussing related work in Section II, we describe the Service Clouds architecture and the prototype implementation in Section III. In the remaining sections, we present three case studies where we used the Service Clouds infrastructure to develop new communication services. The first is a TCP-Relay service, in which a node is dynamically selected and configured to serve as a relay for a data stream. Experiments demonstrate that in many cases this service can yield significantly better performance than using a direct TCP/IP connection. The second is MCON, a service for constructing robust connections for multimedia streaming. When a new connection is being established, MCON exploits physical network topology information to dynamically find and establish a high-quality secondary path, which is used as shadow connection to the primary path. The third is supporting dynamic proxy services at the wireless edge. Experiments demonstrate autonomic instantiation and reconfiguration of proxies to keep a roaming user connected to the network. Preliminary results of this work have been presented in conference papers [14], [15]; herein we provide a more complete description of the project and additional experimental results.

II. Related Work

The Service Clouds concept integrates and extends results from three areas of research. First, adaptive middleware and distributed programming frameworks and models [16]–[19] enable dynamic reconfiguration of software in response to changing conditions. Research in this area has been extensive (see [20] for a survey) and has provided a better understanding of several key concepts relevant to autonomic computing, including reflection, separation of concerns, component-based design, and transparent interception of communication streams. Second, cross-layer cooperation mechanisms [21], [22] enable coordinated adaptation of the system as a whole, and in ways not possible within a single layer. The Service Clouds architecture supports cross-layer cooperation and incorporates low-level network status information in the establishment and configuration of high-level communication services. Third, overlay networks [11] provide an adaptable and responsive chassis on which to implement communication services for many distributed applications [10], [23]–[25].

Recently, researchers have investigated several ways to combine these technologies to support dynamic overlay-based services: iOverlay [26] for lightweight message switching and overlay monitoring; Accord [18] and GridKit [27] for programming support in developing autonomic distributed services; SpiderNet [7] and CANS [28] for dynamic creation and reconfiguration of services along a data path; DSMI [29] for resource-aware stream management; and MACEDON [25] for providing a language-based approach to describing overlay services and automatically generating code for their implementations. The pluggable nature of Service Clouds enables incorporation of existing frameworks such as iOverlay. In addition, developers could take advantage of programming frameworks such as MACEDON, Accord, and GridKit in designing new components for integration into the Service Clouds infrastructure. The Service Clouds model integrates all three major parts (middleware, cross-layer adaptation, and overlay networks) in a single framework. Our case studies demonstrate that such an integrated infrastructure can greatly facilitate the development and deployment of autonomic communication services. Moreover, we view Service Clouds as complementary to SpiderNet, CANS, and DSMI, focusing on the dynamic instantiation and reconfiguration of services on "blank" nodes and the establishment and maintenance of service paths among them.

Service Clouds is also related to research in network management and service provisioning [30], [31], monitoring services [32], and multi-domain network operation [31], [33] in which management and monitoring extend across different administrative domains. Martin-Flatin [34] has proposed a self-managing organizational model in which lower-level managers perform event correlation and self-management rather than having all agents report to higher-level managers. The novelty of this approach is self-managing systems that can communicate with higher-level managers, across multiple domains, when necessary. Such approaches can be used to support efficient execution of autonomic overlay frameworks such as Service Clouds. Strassner and Dupler [35] describe a model-driven approach to implementing network services and argue that universal representation of network functionality is necessary to enable automatic management. As an overlay-based infrastructure, Service Clouds can benefit from autonomic services in the network *underlay*, especially in realizing cross-layer cooperation and distributed management. Moreover, we are currently using model-based approaches in designing the next generation of Service Clouds.

Finally, we emphasize that while our initial investigations focus primarily on functionality and application performance, issues such as software assurance and security of the infrastructure are also being investigated in our laboratory. Examples of software assurance include how to specify, analyze, and verify that a system will always exhibit safety, liveness, and quality-of-service properties when it is deployed. A key issue in dynamically adaptive systems is to ensure that a given adaptive action does not put the system into an inconsistent state [36]. In terms of security, the core of any service infrastructure must be extremely well protected from attackers by the usual battery of security tools (e.g strong encryption to ensure the confidentiality and authenticity of communication). Beyond this, in an ongoing project we are addressing *covert* monitoring of the infrastructure. Specifically, we are investigating how to hide changes in software monitoring from would-be intruders to prevent them from impeding or corrupting the monitoring process. Artifacts and techniques produced by these studies are planned for integration into the next generation of Service Clouds.

III. Architecture and Prototype

The design of Service Clouds has its roots in an earlier study [37], where we designed and constructed the Kernel-Middleware eXchange (KMX), a set of interfaces and services

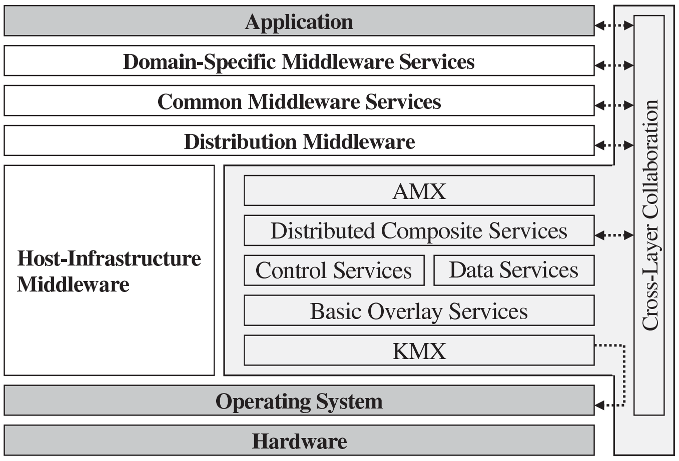

Application
Domain-Specific Middleware Services
Common Middleware Services
Distribution Middleware
Host-Infrastructure Middleware
AMX
Distributed Composite Services
Control Services
Data Services
Basic Overlay Services
KMX
Operating System
Hardware
Cross-Layer Collaboration

Fig. 2. Relationship of Service Clouds to other system layers.

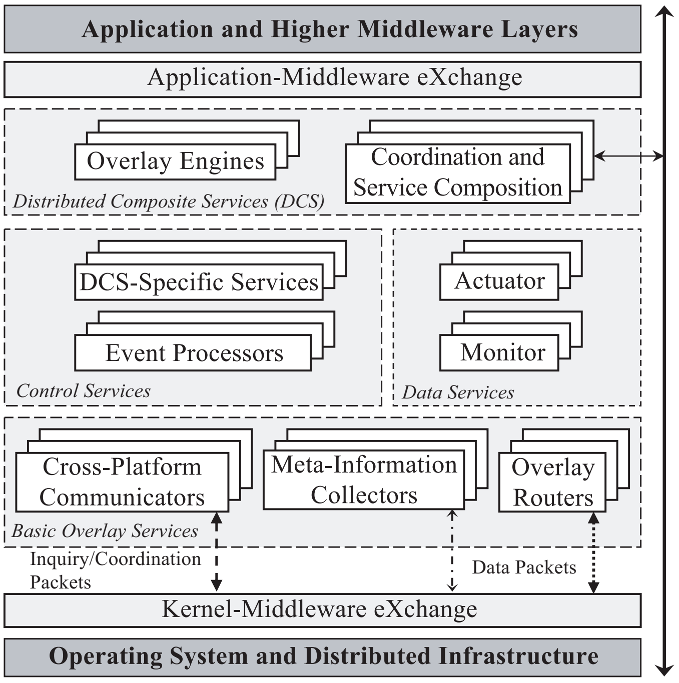

Application and Higher Middleware Layers
Application-Middleware eXchange
Overlay Engines
Coordination and Service Composition
Distributed Composite Services (DCS)
DCS-Specific Services
Event Processors
Control Services
Actuator
Monitor
Data Services
Cross-Platform Communicators
Meta-Information Collectors
Overlay Routers
Basic Overlay Services
Inquiry/Coordination Packets
Data Packets
Kernel-Middleware eXchange
Operating System and Distributed Infrastructure

Fig. 3. Service Clouds architecture.

to facilitate collaboration between middleware and the operating system. We used KMX to improve QoS in video streams transmitted across wireless networks. That study yielded the concept of a *transient proxy*, whereby a service is instantiated on one or more hosts in the network, as needed, in response to changing conditions. We view Service Clouds as a generalization of this concept. The overlay network provides processing and communication resources on which transient *services* can be created as needed to assist distributed applications.

Model. Figure 2 shows a high-level view of the Service Clouds software organization and its relationship to Schmidt's model of middleware layers [38]. Most of the Service Clouds infrastructure can be considered as *host-infrastructure* middleware, as it can be invoked directly by either the application itself or by another middleware layer. An *Application-Middleware eXchange (AMX)* provides interfaces for that purpose, and encapsulates high-level logic to drive various overlay services. The architecture is designed with other developers in mind. Distributed composite services are created by plugging in new algorithms and integrating them with lower-level control and data services, as well as with mechanisms for cross-layer collaboration.

Architecture. Figure 3 provides a more detailed view of the Service Clouds architecture. The architecture comprises four main groups of services, sandwiched between the AMX and KMX layers. The *Distributed Composite Services* layer coordinates interactions among the layers of a single platform and activities across platforms. Each composite service is decomposed into *Data Services* (e.g., transcoding a video stream for transmission on a low bandwidth link) and *Control Services* (e.g., coordinating the corresponding encoding/decoding actions at the sender and receiver of the stream). *Basic Overlay Services* provide generic facilities for establishing an overlay topology, exchanging status information and distributing control packets among overlay hosts. The current prototype framework provides the "glue code" and a repository of component services to be used as a toolkit for developing and deploying higher-level services. Source templates are provided for each service type, facilitating extension of the framework. In addition, we are currently applying design patterns and model-driven development to construct a new prototype of Service Clouds, with a goal of enabling developers to automatically generate new service elements from corresponding models. Next, we discuss each group of services, starting at the bottom and working upward.

Basic Overlay Services include three main types of components: *meta-information collectors*, *cross-platform communicators*, and *overlay routers*. Meta-information collectors at a given node gather system status information, such as current packet loss rate and available bandwidth on each overlay link connected to the node. Cross-platform communicators send and receive inquiry packets, which carry meta-information across platforms in a distributed system. An overlay router component forwards data packets among overlay nodes, and also supports their interception for intermediate processing.

Control Services include both *event processors* and *DCS-specific services*. An event processor handles control events and inquiry packets, for example, extracting information useful to higher-level services (such as failure of a component at another node). An event processor can also perform intermediate processing, for example, constructing statistical information on network conditions. In contrast, a DCS-specific service implements functionality tied to a particular high-level service, for example, an application-specific probing service.

Data Services are used to process data streams as they traverse a node. *Monitors* carry out measurements on data streams. The metrics can be generic in nature (e.g., packet delay and loss rate) or domain-specific (e.g., jitter in a video stream). *Actuators* are used to modify data streams, based on the information gathered by monitors or by explicit requests from higher-level services. We differentiate two types of actuators: *local adaptors* and *transient proxies*. A local adaptor operates on a data stream at the source and/or the destination. For example, a local adaptor operating on a mobile node may handle intermittent disconnections so that applications

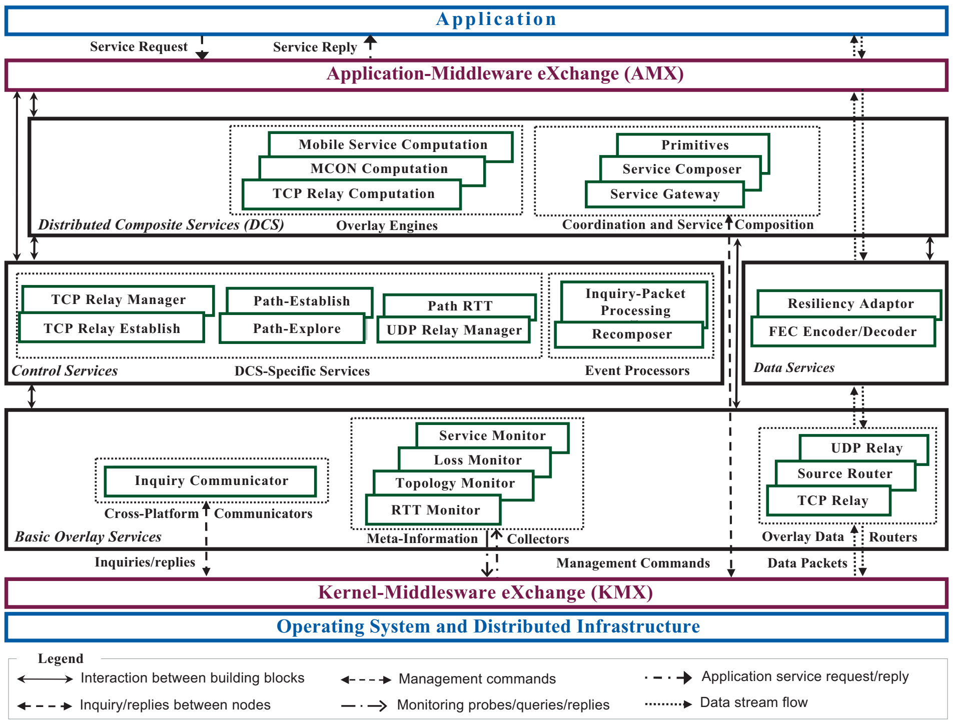


Fig. 4. Instantiation of the general Service Clouds model.

running on the node do not hang or crash. Transient proxies are adaptors that manipulate data streams at intermediate nodes. For example, a transient proxy at the wireless edge of the Internet can intercept a stream carrying sensitive data and re-encrypt it before it traverses a (potentially less secure) wireless channel.

The Distributed Composite Services unit includes two types of components: *overlay engines* and *coordination and service composition*. An overlay engine executes a distributed algorithm across overlay nodes. Examples include building and maintaining a multicast tree for content distribution, establishing redundant overlay connections between nodes, and identifying and configuring relay nodes to improve throughput in bulk data transfers. Coordination and service composition refers to overseeing of several supporting activities needed to realize vertical and horizontal cooperation. For example, a coordinator can determine to use a particular overlay algorithm, set the update intervals in meta-information collectors for gathering information (which affects the overhead and accuracy of the gathered data), and configure an event processing component to perform pre-processing of the received pieces of information.

Prototype. Figure 4 shows the components that comprise the current Service Clouds prototype. The prototype combines service-oriented building blocks in a framework to interact with each other. The layered architecture enables higher-level components to use and share lower ones. The functionality of individual components will be described later with the corresponding case study.

The Service Clouds prototype is written primarily in Java, but components can incorporate modules written in other languages, by using the Java Native Interface. The prototype software has a main driver, which implements service composition and coordination. It reads configuration files containing the IP addresses of overlay nodes and the initial overlay topology, instantiates a basic set of components as threads, and configures them according to the default or specified parameter values. Example parameters include the interval between probes for monitoring packet loss, the maximum number of hops an inquiry packet can travel, and an assortment of debugging options. To manage the distributed infrastructure, we use Java Message Service (JMS). The AMX interface to the infrastructure is implemented using local TCP sockets. Components such as meta-information collectors implement the *singleton pattern*, which means that different overlay services refer to and use the same instantiation of a monitoring service. This design facilitates sharing of component services among more complex services.

The format of inquiry packets exchanged among nodes is XML, and we use Document Object Model (DOM) to enable Service Clouds components to access and manipulate their content. Although DOM may not be as fast as the other alternatives, such as the Simple API for XML (SAX), it provides a convenient interface for inspecting and modifying XML fields, and benchmarking of our prototype indicates its performance is satisfactory. Further, the emergence of efficient XML technologies, such as binary XML format, also addresses concerns about bandwidth consumption and the speed of processing XML documents.

We have used the Service Clouds prototype to conduct three case studies, presented in the following sections. All three studies make use of PlanetLab [13], an Internet research testbed comprising hundreds of Linux-based Internet nodes distributed all over the world. The third case study combines

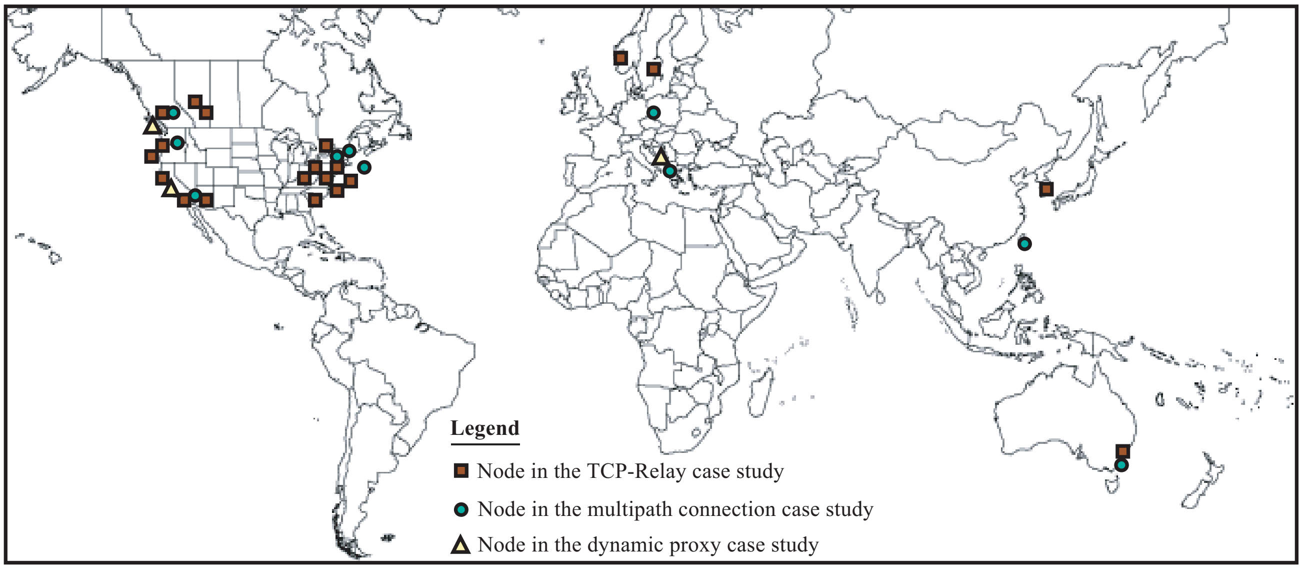

Legend
■ Node in the TCP-Relay case study
● Node in the multipath connection case study
△ Node in the dynamic proxy case study

Fig. 5. PlanetLab nodes used in the case studies.

PlanetLab nodes with those on a mobile computing testbed. Figure 5 shows the PlanetLab nodes used in the case studies.

IV. TCP-Relay Case Study

The first case study investigates a means to improve throughput for bulk data transfers. Recent studies indicate that application-level relays in an overlay network can actually improve TCP throughput for long-distance bulk transfers [39]. Specifically, due to the dependence of TCP throughput on round-trip time, splitting a connection into two (or more) shorter segments can increase throughput, depending on the location of the relay nodes and the overhead of intercepting and relaying the data. To develop a practical TCP relay service, key issues to be addressed include identification of promising relay nodes for individual data transfers, and the *dynamic instantiation* of the relay software. In this case study, we use the Service Clouds infrastructure to construct such a service, and we evaluate the resulting performance. We note that TCP relays can be more easily deployed than some other approaches to improving TCP throughput, such as using advanced congestion control protocols [40], which require either router support or kernel modifications. Alternatively, TCP relays can be used in tandem with such techniques.

Basic Operation. Figure 6 illustrates the use of a TCP relay for a data transfer from a source s to a destination d. Let R_{sd} and p_{sd} denote the round-trip time (RTT) and loss rate, respectively, along the default Internet path connecting s and d. The performance of a large TCP transfer from s to d is mainly determined by TCP's long-run average transmission rate, denoted T_{sd}, which can be approximated using the following formula [41]:

$$T_{sd} \approx \frac{1.5\,M}{R_{sd}\sqrt{p_{sd}}},$$

where M is the maximum size of a TCP segment. Since T_{sd} is inversely proportional to RTT, TCP flows with long propagation delays tend to suffer when competing for bottleneck bandwidth against flows with low RTT values. On the other

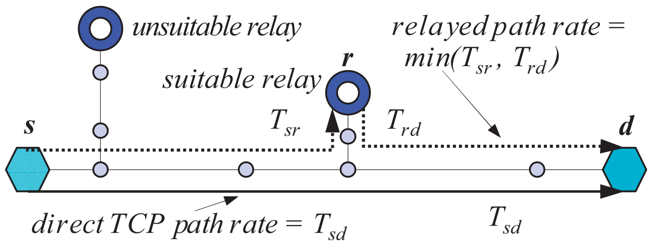

unsuitable relay
relayed path rate = min(T_{sr}, T_{rd})
r
suitable relay
s
T_{sr}
T_{rd}
d
direct TCP path rate = T_{sd}
T_{sd}

Fig. 6. An example of a TCP relay.

hand, if the same transfer is implemented with two separate TCP connections through node r, then the approximate average throughput T_{srd} will be the minimum throughput on the two hops, $T_{srd} = \min(T_{sr}, T_{rd})$.

If more than one suitable relay can be chosen, then a routing decision is required to determine which of the possible relays will yield the best performance. Typically, a well chosen relay will be roughly equidistant between s and d and satisfy:

$$R_{sr} + R_{rd} < \gamma\, R_{sd},$$

where $\gamma > 1$ is a small stretch factor. In this case study, we limit consideration to a single relay. However, routing a connection through a sequence of relays can further improve performance, with marginally diminishing improvement as the number of relays increases.

A TCP relay service must satisfy several requirements. First, the path properties such as RTT and loss rate used to predict TCP throughput must be measured continuously and unobtrusively. Since no overlay node can directly measure all paths, measurement results must be distributed across the overlay. Second, the initiator of the transfer must use these measurements to predict the TCP throughput for all possible relays and select the best option (which may be the direct Internet path if a good relay does not exist). Third, the initiator must signal the chosen relay node to set up and tear down a linked pair of TCP connections. Finally, the relay must forward data efficiently, with as little overhead as possible.

TABLE I
LIST OF NODES IN THE TCP-RELAY EXPERIMENT.

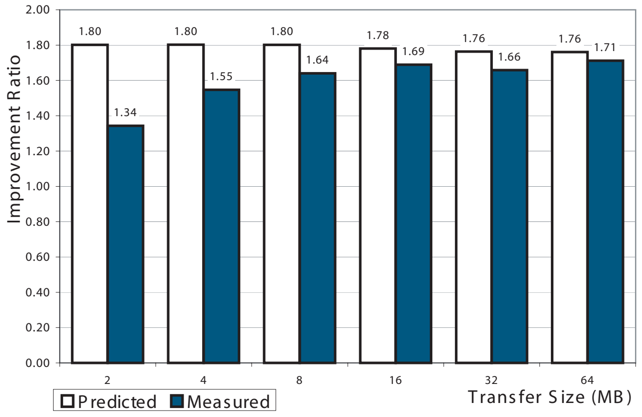
ID	Node	Location
0	planet-lab-1.csse.monash.edu.au	Australia
1	plab1.cs.ust.hk	Hong Kong (China)
2	planetlab1.cs.cornell	USA-NY
3	planet1.cs.ucsb.edu	USA-CA
4	freedom.ri.uni-tuebingen.de	Germany
5	planet1.pittsburgh.intel-research.net	USA-PA
6	planetlab01.cs.washington.edu	USA-WA
7	planetlab3.uvic.ca	Canada
8	planetslug1.cse.ucsc.edu	USA-CA
9	planetlab2.polito.it	Italy

As we will see below, these requirements map neatly to the separation of concerns among particular component types in the Service Clouds architecture.

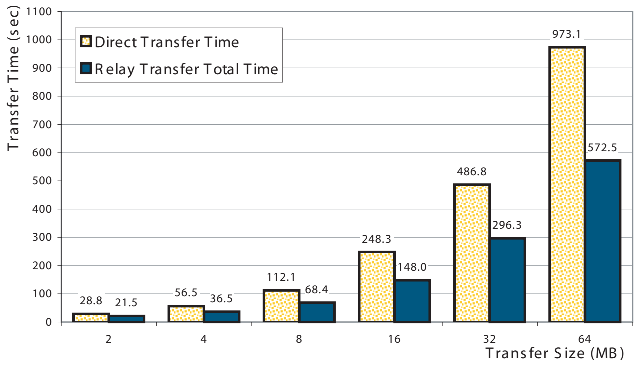
Implementation Details. Figure 4 includes components that were used in TCP-Relay. First, the *RTT Monitor* component encapsulates the collection and distribution of delay measurements. Second, the *TCP Relay Computation* component encapsulates the computation required to predict TCP throughput and find optimal relays. It exposes an interface for relay selection and relies on the RTT Monitor to provide input for its computation. Third, the *Relay Establish* and *Relay Manager* components collectively implement the signaling protocol to setup and control a relay. Finally, the *TCP Relay* component implements data forwarding at the relay node, executing in a transient thread instantiated by the Relay Manager for the duration of a particular transfer.

Experimental Results. To evaluate the TCP-Relay service for connections with long RTT, we constructed an overlay network comprising 10 geographically diverse PlanetLab nodes, as illustrated in Figure 5, and conducted an exhaustive set of data transfers between all possible sender-receiver pairs. Table I lists the nodes used in this case study. In these tests, all transfers originate and terminate at overlay nodes, possibly using other overlay nodes as relays. To evaluate the performance with respect to a particular pair, we performed two back-to-back data transfers of equal size—one via a direct TCP connection and another using the TCP relay infrastructure—recording the throughput achieved by the relayed transfer divided by that of the direct transfer. We refer to this quantity as the *improvement ratio*. While Service Clouds is not always able to find a promising relay (in which case the native TCP/IP connection is used) our results show it can enhance performance for many situations.

Since our focus in this paper is on the architecture and use of Service Clouds, we present only a sample of the results. Figure 7 shows results for a particular sender-receiver pair in our experiment. The sender (ID 0 in Table I) is in Australia, and the receiver (ID 1) is in Hong Kong. For



(a) average improvement ratio



(b) average transfer time

Fig. 7. TCP-Relay results for a selected pair.

this pair, our prototype service consistently selected a relay on the East Coast of the United States (ID 2). Although the location of the relay is counterintuitive, an examination of paths discovered by "traceroute" offers a straightforward explanation. The default Internet route from the sender to the receiver makes a trans-Pacific hop to the United States, then traverses a high-performance backbone network (Abilene) to an inter-ISP exchange point in Chicago. A few hops after Chicago is a second trans-Pacific hop to China. Each trans-Pacific hop adds roughly 200 msec to the total RTT, placing the Chicago exchange point very close to the midpoint of the route. The chosen relay turns out to be an excellent choice—adding only a 20 msec detour through Abilene before rejoining the direct route in Chicago.

Figure 7(a) shows the predicted and observed improvement ratio for relayed transfers of various sizes using the Australia-to-Hong Kong pair. Each observed value is an average of at least 10 measurements. The plot shows that the measured improvements are close to the theoretical prediction. We also observe the amortization of startup latency at larger transfer sizes, which can be seen in comparisons of the durations of direct and relayed transfers in Figure 7(b). The largest transfer in our tests, 64MB, constitutes a relatively small bulk transfer. Yet even at this scale, we see a significant reduction in transfer time compared to using native TCP/IP—from about 15 to about 10 minutes.

V. Multipath Connection Case Study

Establishing multiple connections (e.g., a primary path and one or more backup paths) between a source and destination node can improve communication QoS by mitigating packet loss due to network failures and transient delays. This capability is especially important to applications such as high-reliability conferencing, where uninterrupted communication is essential. However, in overlay networks, selection of high-quality primary and backup paths is a challenging problem due to the sharing of physical links among overlay paths. Such sharing affects many routing metrics, including joint failure probability and link congestion. In an earlier work, we described TAROM [42], a distributed algorithm for disseminating physical topology information to establish multipath connections. In this case study, we use TAROM in the construction of MCON (Multipath CONnection), a distributed service that *automatically* finds, establishes, and uses a high-quality secondary path whenever a primary overlay path is established.

Basic Operation. MCON combines traditional distributed multipath computation with topology-aware overlay routing and overlay multipath computation. A topology-aware overlay approach [43] considers shared physical links between two paths as a factor when determining high-quality backup paths. Since sharing of physical links among overlay paths reduces their joint reliability, this method leads to finding more robust backup paths.

The basic operation of MCON connection establishment is depicted in Figure 8. To establish a multipath connection, the source node sends a request packet to the destination node along the primary path (path-establish procedure). This packet collects path composition (physical topology) and quality information (packet loss in this prototype) along the way. Upon reception, the destination node forwards this information to its neighbors in *path-explore* packets, each of which attempts to find its way to the source. When a node receives a path-explore packet, it uses both the received information and its local state to calculate the joint quality of the primary path and the partial path from itself to the destination. If the joint quality-cost ratio is above a specified threshold, it forwards this information to its neighbors. By adjusting the value of the threshold, the application can control the scope and overhead of this path-explore process. If a node receives path-explore packets from two neighbors, it calculates the joint quality of the two partial paths and forwards the information of the primary and the better partial path to its neighbors. The process terminates when the remaining branches reach the source, providing it with a set of high-quality backup paths. Details of the protocol can be found in [42].

Implementation Details. In the Service Clouds implementation, two resiliency strategies are supported: topology-unaware (TUA) and topology-aware using inference (TA-i). The former attempts to find a high-quality secondary path with respect to the overlay topology, but without considering underlying path sharing in the physical network. The latter uses physical topology information, some of it measured directly and the rest obtained using inference techniques that have previously been shown to be highly accurate [43].

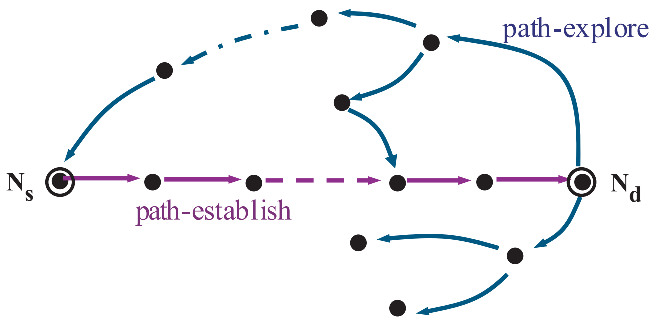


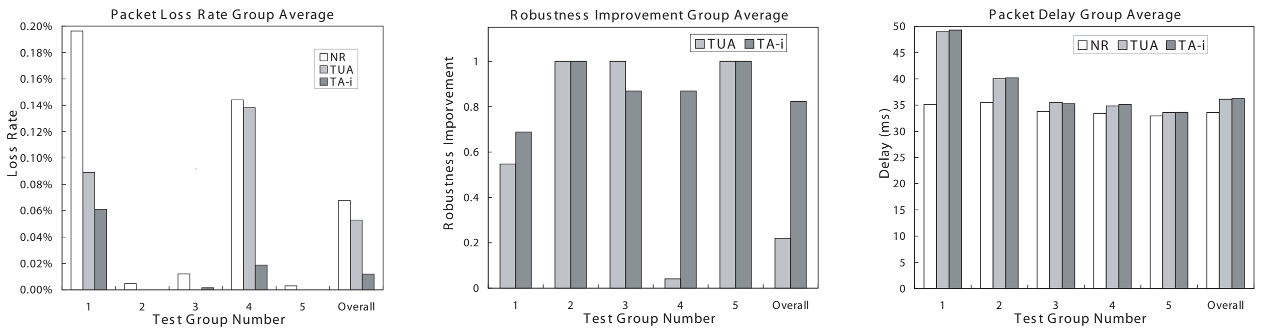
Fig. 8. Basic operation of MCON.

As described earlier, the logic for a distributed service is encapsulated in an overlay engine. In the case of MCON, we had available to us a C++ implementation of the TAROM algorithm, which had been used as part of a simulation study. With only minor modifications, we were able to plug the existing C++ code into the Service Clouds infrastructure as an MCON overlay engine, and access it through the Java Native Interface.

MCON-specific control services include components to transmit and forward *Path-Establish* and *Path-Explore* packets among nodes. In terms of data services, MCON requires a *Resiliency Adaptor* (an instance of a local adaptor) at both the source and destination. At the source, the adaptor creates a duplicate of the data stream and sends it to the destination over the discovered secondary path. At the destination, the adaptor delivers the first copy of each data packet received and discards redundant copies.

Experimental Results. In this set of experiments, we selected 20 Planetlab nodes, listed in Table II (also illustrated on the geographical map in Figure 5), and established an overlay network among them. We chose five sender-receiver pairs from the overlay network and transmitted a UDP stream from each sender to its corresponding receiver using each of three different services: one without any resiliency services (NR), one with topology-unaware multipath service (TUA), and one with topology-aware multipath service (TA-i). For the experiments presented here, the application generates data packets of length 1KB and sends them with 30 msec intervals for 3 minutes, which generates 6000 UDP packets for each sample stream. The results presented are the average of seven separate experimental runs.

We compare the performance of TUA and TA-i with NR in terms of packet loss rate reduction, robustness improvement, and overhead delay. The robustness improvement I for a multipath data transmission is defined as the percentage of fatal failure probability (the rate of simultaneous packet losses on all paths) reduction due to the usage of the secondary path, or, $I = (F_s - F_d)/F_s$, where F_s is the single path failure probability of the primary path, and F_d is the double path failure probability of both the primary and the secondary path. Figure 9 shows a sample of the results for different test groups (aggregated according to sender-receiver pairs) and the overall results. Figure 9(a) demonstrates that compared with NR, both TUA and TA-i can substantially reduce the average packet loss rate. Figure 9(b) shows TA-i exhibits higher robustness improvement than TUA, except in group 3. For group 4, the



(a) average packet loss rate (b) average robustness improvement (c) average packet delay

Fig. 9. MCON empirical results on PlanetLab.

TABLE II
LIST OF NODES IN THE MCON EXPERIMENTS.

ID	Node Name	Location	Sender/ Receiver	Pair #
0	planetlab1.enel.ucalgary.ca	Canada	Sender	1
1	plab2.ee.ucla.edu	US – CA	Receiver	
2	planetlab-3.cs.princeton.edu	US – NJ	Sender	2
3	planet-lab-2.csse.monash.edu.au	Australia	Receiver	
4	planetlab2.sics.se	Sweden	Sender	3
5	planetlab2.arizona-gigapop.net	US – AZ	Receiver	
6	planetlab2.win.trlabs.ca	Canada	Sender	4
7	planetlab1.cs.uit.no	Norway	Receiver	
8	csplanetlab2.kaist.ac.kr	South Korea	Sender	5
9	blast.cs.uwaterloo.ca	Canada	Receiver	
10	plil-pa-4.hp1.hp.com	US – CA		
11	planetlab1.nbgisp.com	US – OR		
12	planetlab3.comet.columbia.edu	US – NY		
13	planetlab-2.eecs.cwru.edu	US – OH		
14	planet03.csc.ncsu.edu	US – NC		
15	planetlabone.ccs.neu.edu	US – MA		
16	planetlab1.een.orst.edu	US – OR		
17	planetlab1.rutgers.edu	US – NJ		
18	planetlab1.netlab.uky.edu	US – KY		
19	planetlab3.uvic.ca	Canada		

average robustness improvement for TA-i is over 20 times greater than that of TUA. Finally, while the use of the Service Clouds infrastructure introduces additional computational and communication overhead, Figure 9(c) shows that except for group 1, the effect on data packet delay is relatively small, when compared to NR.

These results demonstrate that the MCON service implemented within the Service Clouds prototype, can improve communication reliability in real Internet environments. Moreover, we emphasize that this improvement exists, even though the implementation is not optimized. We simply plugged an existing algorithm and rudimentary monitoring routines into the prototype. While some anomalies appear in the results, due to the dynamics of the Internet testbed, the overall performance demonstrates the benefits of using these services. Further, we note that the experiments described here were conducted under "normal" network conditions; the most important benefit of the MCON service is likely to be in handling situations where serious failures occur along the primary path, and a high-quality secondary path is needed to deliver the data stream to the destination.

VI. DYNAMIC PROXY CASE STUDY

In the third case study, we extend the Service Clouds concept to support adaptive behavior in mobile computing. In this model, depicted in Figure 10, *mobile* service clouds (MSC) comprise collections of hosts that implement services close to the wireless edge (e.g., a collection of nodes on a university intranet or used by an ISP to enhance QoS at wireless hotspots), while *deep* service clouds perform services using an Internet overlay network (such as the PlanetLab wired hosts). In a manner reminiscent of Domain Name Service, the clouds in this *federation* cooperate to meet the needs of applications. As a mobile user interacts with different service clouds while moving about the wireless edge, services are instantiated and reconfigured dynamically to meet changing needs.

Basic Operation. Let us consider scenarios where a mobile node on a wireless link wants to receive continuous data streams (e.g., in an interactive video conference or in a live video broadcast), despite changes in network connections and failures of hardware and software components. In this case, MSC needs to fulfill the following requirements. First, the quality of the received stream must remain acceptable as the wireless link experiences packet loss. Second, the video stream must be transcoded to satisfy resource restrictions such as wireless bandwidth and processing power at the mobile device. Third, stream delivery should not be interrupted as conditions on the service path change (e.g., when user movement causes a wireless network domain change, or when a service node fails).

At the wireless edge, services are often deployed on *proxy* nodes, which operate on behalf of mobile hosts. Extensive research has been conducted in the design of proxy services

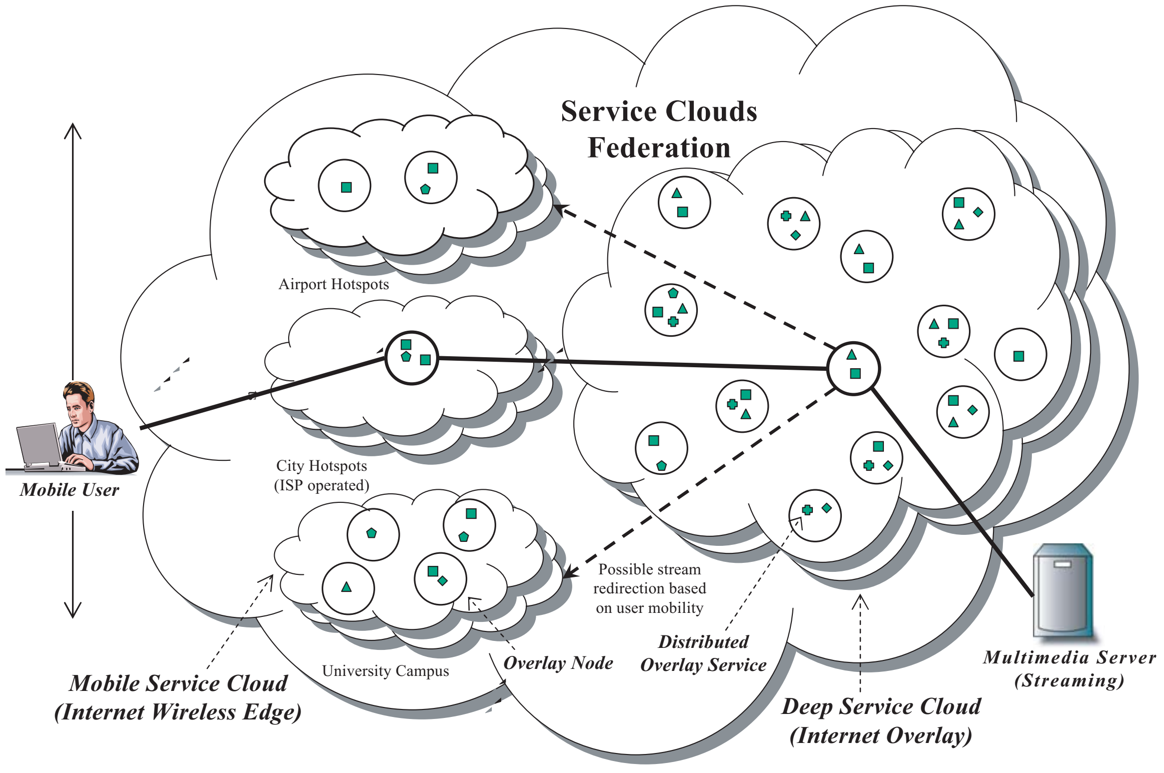

Service Clouds Federation
Airport Hotspots
City Hotspots (ISP operated)
University Campus
Mobile User
Mobile Service Cloud (Internet Wireless Edge)
Overlay Node
Distributed Overlay Service
Possible stream redirection based on user mobility
Deep Service Cloud (Internet Overlay)
Multimedia Server (Streaming)

Fig. 10. Example scenario involving Mobile Service Clouds.

to support data transcoding, handle frequent disconnections, and enhance the quality of wireless connections through error-correction techniques [44]–[47]. Service Clouds provides the infrastructure needed to quickly compose such services and, at run time, to support their dynamic instantiation and reconfiguration in response to changing conditions.

Implementation Details. For the prototype implementation of Mobile Service Clouds, we introduced new components but also reused several others. The components are depicted in Figure 4. First, the "Coordination and Service Composition" manages the interaction between a mobile host and a service cloud federation. Tasks include selection of a node in a deep service cloud—using the *Service Path Computation* overlay engine—called the *primary proxy*, which coordinates composition and maintenance of the service path between two end nodes. The Service Path Computation engine also finds a suitable node in a mobile service cloud on which to deploy the transient proxy services (FEC in our study). We also required lower-level services to help identify potential proxies, including an implementation of the RTT monitor that measures round-trip time to an arbitrary node on demand, and the *Path RTT* component that measures end-to-end RTT between two nodes, whose communication is required to pass through an intermediate node.

The *Service Gateway* component implements a simple protocol to accept and reply to service requests. Upon receiving a request, it invokes the overlay engine to find a suitable primary proxy. The *Service Composer* component implements mechanisms for composing a service path. It uses the *Relay Manager* to instantiate and configure a UDP relay on the primary proxy and the transient proxy. The UDP relay on the transient proxy enables the infrastructure to intercept the stream and augment it with FEC encoding. Accordingly, as soon as the FEC proxy service is instantiated, the *Service Monitor* on the transient proxy begins sending heartbeat messages through a TCP channel toward the service monitor on the primary proxy. The *Recomposer* component on the primary proxy tracks activity of the service monitors. Upon detecting a failure, it starts a self-healing operation that recomposes the service path and restores communication. Additionally, a monitor on the client middleware notifies the primary proxy of changes, such as low-battery status or change of IP address, and the recomposer reconfigures the service as necessary.

Experiments in Seamless Mobility. The particular scenario we consider is a multicast-capable proxy that also supports mobility. Here, a set of clients wish to receive a multimedia stream (e.g, in video-conferencing or live video broadcast), and the infrastructure fulfills the following requirements: first, avoiding stream interruptions as a user relocates and connects to a new network domain, gaining a new IP address; second, high-quality reception regardless of the network connection.

Figure 11 shows the testbed, with 3 PlanetLab nodes in a deep service cloud, two workstations in a mobile service cloud on our university intranet, and two laptops with RTP-based video players (listed in Table III). Subnet A is a wired LAN and subnet B is wireless. The middleware software on a client connects to a Service Gateway ($N3$) and requests the video. The Primary Proxy ($N1$) creates a service path and coordinates monitoring and automatic reconfiguration during the communication. In this scenario, proxies at the wireless edge deploy two functionalities: multicasting and forward error correction (FEC). Since multicasting is not commonly

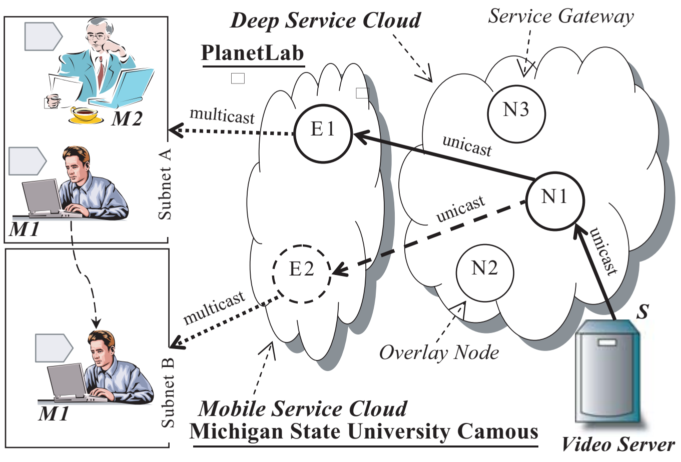


Fig. 11. The experimental setup for seamless mobility.

TABLE III
LIST OF NODES IN THE DYNAMIC PROXY EXPERIMENT.

M1	senslap10.cse.msu.edu	User 1 (mobile)
M2	senslap12.cse.msu.edu	User 2
S	copland.cse.msu.edu	Video Streaming Server
N1	planetlab1.cs.unibo.it	Service Gateway
N2	planetslug1.cse.ucsc.edu	Candidate Primary Proxy
N3	planetlab3.uvic.ca	Candidate Primary Proxy (chosen at run time)
E1	countbasey.cse.msu.edu	Wireless edge node
E2	senslap11.egr.msu.edu	Wireless edge node

available on the Internet, the stream is unicasted toward the wireless edge, where the proxy multicasts it toward the clients. To maintain the quality of the video, the proxy applies FEC on the stream when a client detects high packet loss rate. The infrastructure supports continuous streaming by dynamic instantiation of proxies, while users roam along different subnets.

Figure 11 depicts the following scenario: *(1)* User $M1$ on wired subnet A requests to watch the video from the server; accordingly, the primary proxy creates a service path comprising streaming relays and a unicast-to-multicast proxy on $E1$. *(2)* User $M2$ requests the same video on wired subnet A; since the video is already being multicasted in subnet A, no extra configuration is necessary, except registering user $M2$ as a service receiver and configuring $M2$ to receive the video. *(3)* User $M1$ switches from wired connection on subnet A to wireless connection on subnet B; the middleware on the client detects change of IP address and notifies the primary proxy. At this point, the primary proxy extends the service path to make the stream available in subnet B via a proxy on $E2$. Moreover, since the connection to $E2$ is wireless, if packet loss rate becomes intolerable, the proxy performs FEC encoding.

In this implementation, the stream comprises audio and

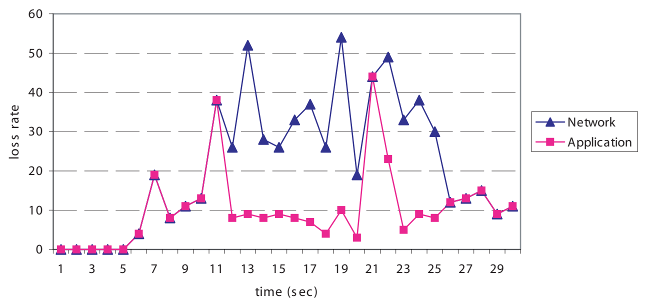


Fig. 12. Audio packet loss rate at mobile node M1.

video packets sent over separate UDP sockets. For test purposes we stream a prerecorded 30 frame-per-second motion-JPEG video. Whenever the client in the wireless subnet detects intolerable loss rate (20%), the proxy in the mobile cloud FEC encodes the stream by breaking each packet to four packets and sending them along with four extra parity packets. Figure 12 plots the packet loss rates for audio at $M1$. We have deployed FEC encoding only on the audio stream to balance QoS and bandwidth consumption. High-quality wireless streaming requires more complicated adaptation techniques to transcode the video, which is beyond the scope of this study. As the plot shows, at second 5 the user switches from wired subnet to the wireless one and the network loss rate raises significantly. Accordingly, the system enables or disables FEC encoding, based on the feedback from the client middleware, and effectively mitigates the packet loss rate observed by the application.

In [14], we describe the use of Service Clouds to construct self-healing proxy services. The tests described above demonstrate that the Service Clouds infrastructure can also be used to construct adaptive service paths at the wireless edge. The infrastructure provides self-management in the dynamic instantiation and migration of proxy services. The resulting system can adapt to dynamic changes in the environment of mobile nodes, including changes in connectivity and channel conditions, while maintaining a high quality of service on the stream delivered to the user.

VII. CONCLUSIONS

In this paper, we described Service Clouds, an overlay-based infrastructure for composing adaptive communication services. We described three experimental case studies using a Service Clouds prototype on the PlanetLab testbed: dynamic selection, creation and use of TCP relays for bulk data transfer; dynamic construction and use of secondary paths for highly resilient streaming; and dynamic instantiation and reconfiguration of proxies at the wireless edge to support mobile computing. These case studies demonstrate the usefulness of the Service Clouds infrastructure in deploying new services, and the performance results reveal the benefits of the services themselves. The Service Clouds model facilitates the engineering and deployment of complex and robust services without requiring changes or reconfiguration to the underlying networks. Specifically, by providing a rich and extensible overlay-based software infrastructure, Service Clouds enables developers to quickly and add new capabilities to the Internet, at very low cost.

Acknowledgments. This work was supported in part by the U.S. Department of the Navy, Office of Naval Research under Grant No. N00014-01-1-0744, National Science Foundation grants EIA-0000433, EIA-0130724, and ITR-0313142, and a Quality Fund Concept grant from Michigan State University.

Further Information. Related publications on the RAPID-ware project, as well as a download of the Service Clouds prototype, can be found at the following website: `http://www.cse.msu.edu/rapidware`.

REFERENCES

[1] J. O. Kephart and D. M. Chess, "The vision of autonomic computing," *IEEE Computer*, vol. 36, pp. 41–50, Jan. 2003.

[2] R. Sterritt, M. Parashar, H. Tianfield, and R. Unland, "A concise introduction to autonomic computing.," *Advanced Engineering Informatics*, vol. 19, no. 3, pp. 181–187, 2005.

[3] G. Blair, G. Coulson, and N. Davies, "Adaptive middleware for mobile multimedia applications," in *Proc. 8th International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV)*, pp. 259–273, 1997.

[4] D. C. Schmidt, D. L. Levine, and S. Mungee, "The design of the TAO real-time object request broker," *Computer Commun.*, vol. 21, pp. 294–324, April 1998.

[5] F. Kon, M. Román, P. Liu, J. Mao, T. Yamane, L. C. Magalhães, and R. H. Campbell, "Monitoring, security, and dynamic configuration with the dynamicTAO reflective ORB," in *Proc. IFIP/ACM International Conference on Distributed Systems Platforms and Open Distributed Processing*, pp. 121–143, April 2000.

[6] S. M. Sadjadi and P. K. McKinley, "ACT: An adaptive CORBA template to support unanticipated adaptation," in *Proc. 24th IEEE International Conference on Distributed Computing Systems (ICDCS'04)*, March 2004.

[7] X. Gu, K. Nahrstedt, and B. Yu, "SpiderNet: An integrated peer-to-peer service composition framework," in *Proc. IEEE International Symposium on High-Performance Distributed Computing (HPDC-13)*, pp. 110–119, June 2004.

[8] X. Fu, W. Shi, A. Akkerman, and V. Karamcheti, "CANS: Composable and adaptive network services infrastructure," in *Proc. 3rd USENIX Symposium on Internet Technology and Systems*, March 2001.

[9] D. Xu and X. Jiang, "Towards an integrated multimedia service hosting overlay," in *Proc. ACM Multimedia 2004*, pp. 96–103, ACM Press, Oct. 2004.

[10] S. D. Gribble, M. Welsh, J. R. von Behren, E. A. Brewer, D. E. Culler, N. Borisov, S. E. Czerwinski, R. Gummadi, J. R. Hill, A. D. Joseph, R. H. Katz, Z. M. Mao, S. Ross, and B. Y. Zhao, "The Ninja architecture for robust Internet-scale systems and services," *Computer Networks*, vol. 35, no. 4, pp. 473–497, 2001.

[11] D. G. Andersen, H. Balakrishnan, M. F. Kaashoek, and R. Morris, "Resilient overlay networks," in *Proc. 18th ACM Symposium on Operating Systems Principles (SOSP 2001)*, pp. 131–145, Oct. 2001.

[12] "The RAPIDware Project." `http://www.cse.msu.edu/rapidware`. Software Engineering and Network Systems Laboratory, Michigan State University - Department of Computer Science and Engineering, East Lansing, MI, USA.

[13] L. Peterson, T. Anderson, D. Culler, and T. Roscoe, "A blueprint for introducing disruptive technology into the Internet," in *Proc. ACM Workshop on Hot Topics in Networks (HotNets–I)*, pp. 59–64, Oct. 2002. `http://www.planet-lab.org/`.

[14] F. A. Samimi, P. K. McKinley, and S. M. Sadjadi, "Mobile Service Clouds: a self-managing infrastructure for autonomic mobile computing services," in *Proc. Second International Workshop on Self-Managed Networks, Systems & Services (SelfMan 2006)*, vol. 3996 of *LNCS*, pp. 130–141, Springer-Verlag, June 2006.

[15] P. K. McKinley, F. A. Samimi, J. K. Shapiro, and C. Tang, "Service Clouds: a distributed infrastructure for constructing autonomic communication services," in *Proc. 2nd IEEE International Symposium on Dependable, Autonomic and Secure Computing (DASC'06)*, (Indianapolis, Indiana, USA), pp. 341–348, IEEE Computer Society, September 2006.

[16] J. A. Zinky, D. E. Bakken, and R. E. Schantz, "Architectural support for quality of service for CORBA objects," *Theory and Practice of Object Systems*, vol. 3, no. 1, pp. 1–20, 1997.

[17] B. Redmond and V. Cahill, "Supporting unanticipated dynamic adaptation of application behaviour," in *Proc. 16th European Conference on Object-Oriented Programming*, vol. 2374 of *LNCS*, pp. 205–230, June 2002.

[18] H. Liu, M. Parashar, and S. Hariri, "A component-based programming model for autonomic applications," in *Proc. 1st International Conference on Autonomic Computing*, pp. 10–17, IEEE Computer Society, May 2004.

[19] K. Arnold, ed., *The Jini Specifications*. Addison-Wesley Professional, second ed., 2000.

[20] P. K. McKinley, S. M. Sadjadi, E. P. Kasten, and B. H. C. Cheng, "Composing adaptive software," *IEEE Computer*, vol. 37, no. 7, pp. 56–64, 2004.

[21] B. D. Noble, M. Satyanarayanan, D. Narayanan, J. E. Tilton, J. Flinn, and K. R. Walker, "Agile application-aware adaptation for mobility," in *Proc. 16th ACM Symposium on Operating Systems Principles*, pp. 276–287, Oct. 1997.

[22] J. Kong and K. Schwan, "KStreams: kernel support for efficient data streaming in proxy servers," in *Proc. 15th International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV)*, pp. 159–164, ACM, 2005.

[23] J. W. Byers, J. Considine, M. Mitzenmacher, and S. Rost, "Informed content delivery across adaptive overlay networks," *IEEE/ACM Trans. Networking*, vol. 12, pp. 767–780, Oct. 2004.

[24] B. Li, D. Xu, and K. Nahrstedt, "An integrated runtime QoS-aware middleware framework for distributed multimedia applications," *Multimedia Systems*, vol. 8, no. 5, pp. 420–430, 2002.

[25] A. Rodriguez, C. Killian, S. Bhat, D. Kostic, and A. Vahdat, "Macedon: methodology for automatically creating, evaluating, and designing overlay networks," in *Proc. USENIX/ACM First Symposium on Networked Systems Design and Implementation (NSDI 2004)*, pp. 267–280, March 2004.

[26] B. Li, J. Guo, and M. Wang, "iOverlay: a lightweight middleware infrastructure for overlay application implementations," in *Proc. Fifth ACM/IFIP/USENIX International Middleware Conference*, vol. 3231 of *LNCS*, pp. 135–154, Oct. 2004.

[27] P. Grace, G. Coulson, G. Blair, L. Mathy, D. Duce, C. Cooper, W. K. Yeung, and W. Cai, "GRIDKIT: Pluggable overlay networks for Grid computing," in *Proc. International Symposium on Distributed Objects and Applications (DOA)*, pp. 1463–1481, Oct. 2004.

[28] X. Fu and V. Karamcheti, "Automatic creation and reconfiguration of network-aware service access paths.," *Computer Commun.*, vol. 28, no. 6, pp. 591–608, 2005.

[29] V. Kumar, B. F. Cooper, Z. Cai, G. Eisenhauer, and K. Schwan, "Resource-aware distributed stream management using dynamic overlays," in *Proc. 25th International Conference on Distributed Computing Systems (ICDCS 2005)*, pp. 783–792, IEEE Computer Society, June 2005.

[30] P. Martin-Flatin, P.-A. Doffoel, and M. Jeckle, "Web services for integrated management: a case study," in *Proc. 2004 European Conference on Web Services (ECOWS 2004)*, vol. 3250 of *LNCS*, pp. 239–253, Springer-Verlag, Sept. 2004.

[31] J. Xiao and R. Boutaba, "QoS-aware service composition and adaptation in autonomic communication," *IEEE J. Sel. Areas Commun.*, vol. 23, pp. 2344–2360, Dec. 2005.

[32] A. Jain, J. M. Hellerstein, S. Ratnasamy, and D. Wetherall, "A wakeup call for internet monitoring systems: The case for distributed triggers," in *Proc. 3rd ACM SIGCOMM Workshop on Hot Topics in Networks (HotNets-III)*, Nov. 2004.

[33] A. Hanemann, J. W. Boote, E. L. Boyd, J. Durand, L. Kudarimoti, R. Aapacz, D. M. Swany, S. Trocha, and J. Zurawski, "PerfSONAR: a service oriented architecture for multidomain network monitoring," in *Proc. Third International Conference on Service Oriented Computing (ICSOC)*, vol. 3826 of *LNCS*, pp. 241–254, Springer-Verlag, Dec. 2005.

[34] J.-P. Martin-Flatin, "Distributed event correlation and self-managed systems," in *Proc. First International Workshop on Self-* Properties in Complex Information Systems (Self-Star 2004)*, pp. 61–64, June 2004.

[35] J. Strassner and C. Dupler, "LANS: a model-driven environment for implementing location-aware network services," *Computer Networks*, vol. 46, no. 5, pp. 581–603, 2004.

[36] J. Zhang, B. H. C. Cheng, Z. Yang, and P. K. McKinley, "Enabling safe dynamic component-based software adaptation," in *Architecting Dependable Systems III, Springer Lecture Notes for Computer Science* (A. R. Rogerio de Lemos, Cristina Gacek, ed.), Springer-Verlag, 2005.

[37] F. A. Samimi, P. K. McKinley, S. M. Sadjadi, and P. Ge, "Kernel-middleware interaction to support adaptation in pervasive computing environments," in *Proc. 2nd Workshop on Middleware for Pervasive and Ad-Hoc Computing*, pp. 140–145, ACM Press, Oct. 2004.
[38] D. C. Schmidt, "Middleware for real-time and embedded systems," *Communications of the ACM*, vol. 45, pp. 43–48, June 2002.
[39] Y. Liu, Y. Gu, H. Zhang, W. Gong, and D. Towsley, "Application level relay for high-bandwidth data transport," in *Proc. First Workshop on Networks for Grid Applications (GridNets)*, Oct. 2004.
[40] C. Jin, D. X. Wei, and S. H. Low, "Fast TCP: motivation, architecture, algorithms, performance," in *Proc. IEEE INFOCOM*, pp. 2490–2501, March 2004.
[41] M. Mathis, J. Semke, J. Madhavi, and T. Ott, "The macroscopic behavior of the TCP congestion avoidance algorithm," *ACM Computer Commun. Review*, vol. 27, no. 3, pp. 67–82, 1997.
[42] C. Tang and P. K. McKinley, "Improving multipath reliability in topology-aware overlay networks," in *Proc. Fourth International Workshop on Assurance in Distributed Systems and Networks (ADSN), held in conjunction with the 25th IEEE International Conference on Distributed Computing Systems*, June 2005.
[43] C. Tang and P. K. McKinley, "On the cost-quality tradeoff in topology-aware overlay path probing," in *Proc. 11th IEEE International Conference on Network Protocols (ICNP)*, pp. 268–279, November 2003.
[44] J. Kristiansson and P. Parnes, "An application-layer approach to seamless mobile multimedia communication," *IEEE eTrans. Network Service Management (eTNSM)*, vol. 3, no. 1, pp. 33–42, 2006.
[45] A. Fox, S. D. Gribble, Y. Chawathe, and E. A. Brewer, "Adapting to network and client variation using active proxies: Lessons and perspectives," *IEEE Personal Commun.*, vol. 5, pp. 10–19, Aug. 1998.
[46] B. Zenel, "A general purpose proxy filtering mechanism applied to the mobile environment," *Wireless Networks*, vol. 5, pp. 391–409, 1999.
[47] M. Roussopoulos, P. Maniatis, E. Swierk, K. Lai, G. Appenzeller, and M. Bake, "Person-level routing in the mobile people architecture," in *Proc. 1999 USENIX Symposium on Internet Technologies and Systems*, pp. 165–176, October 1999.

Farshad A. Samimi is a Ph.D. candidate in the Department of Computer Science and Engineering at Michigan State University. He received the B.S. degree in computer engineering from Shiraz University, Iran. His research interests include autonomic computing, networking and communication services, adaptive systems and middleware, and pervasive mobile computing. Contact him at farshad@cse.msu.edu.

Philip K. McKinley received the Ph.D. degree in computer science from the University of Illinois at Urbana-Champaign in 1989. Dr. McKinley is currently a Professor of Computer Science and Engineering at Michigan State University. He has served as an Associate Editor for IEEE Transactions on Parallel and Distributed Systems and was co-chair of the program committee for the 2003 IEEE International Conference on Distributed Computing Systems. His research interests include self-adaptive software, autonomic computing and digital evolution. Contact him at mckinley@cse.msu.edu.

S. Masoud Sadjadi received the Ph.D. degree in computer science from Michigan State University in 2004, the M.S. degree in software engineering from Azad University of Tehran in 1999, and the a B.S. degree in hardware engineering from University of Tehran in 1995. He is currently an assistant professor of Computer Science and co-director of the Autonomic Computing Research Laboratory at Florida International University. His research interests include autonomic computing and dynamic software adaptation in pervasive computing and grid systems. Contact him at sadjadi@cs.fiu.edu.

Chiping Tang received the Ph.D. and M.S. degrees from Michigan State University in 2005 and 2002, respectively, and the B.S. degree from Peking University, all in computer science. He is currently with Microsoft Corporation. His research interests include Internet computing and mobile/wireless systems. Contact him at tangcp28@gmail.com.

Jonathan K. Shapiro received the Ph.D. and M.S. degrees in computer science from the University of Massachusetts at Amherst in 2003 and 2000, respectively, and the B.A. degree in physics from Columbia University in 1990. He was previously on the faculty at Michigan State University and is currently with Adverplex. His research interests include applications of economics to problems in networks and distributed systems, privacy and anonymous communication, and peer-to-peer systems. Contact him at jshapiro@adverplex.com.

Zhinan Zhou received the Ph.D. degree in Computer Science in 2006 from Michigan State University, and the M.S. and B.E. degrees from Tsinghua University, Beijing, China, in 2001 and 1999, respectively. He is currently with Samsung Corporation. His research interests include multimedia streaming, adaptive middleware and mobile computing. Contact him at z.zhou@samsung.com.