Technical Report MSU-CSE-05-31, November 2005.

Service Clouds: A Distributed Infrastructure for Composing Autonomic Communication Services *

Philip K. McKinley, Farshad A. Samimi, Jonathan K. Shapiro, and Chiping Tang†

Software Engineering and Network Systems Laboratory
Department of Computer Science and Engineering
Michigan State University
East Lansing, Michigan 48824
{mckinley, farshad, jshapiro, tangchip}@cse.msu.edu

Abstract

This paper proposes *Service Clouds*, a distributed infrastructure designed to facilitate rapid prototyping and deployment of autonomic communication services. The Service Clouds infrastructure combines dynamic software configuration methods with overlay network services in order to support both cross-layer and cross-platform cooperation. The Service Clouds architecture includes a collection of low-level facilities that can be either invoked directly by applications or used to compose more complex services. The architecture is designed for extension: developers can plug in new modules, at different layers of the architecture, and use them in constructing additional services. We have implemented a prototype of Service Clouds atop the PlanetLab Internet testbed. After describing the Service Clouds architecture, we present results of two experimental case studies conducted on PlanetLab. In the first we use Service Clouds to implement a service in which an overlay node is dynamically selected and configured as a TCP relay for bulk data transfers; experiments demonstrate that the Service Clouds approach often produces better performance than using native IP routes. In the second we use Service Clouds to deploy and evaluate a multipath protocol that dynamically establishes a high-quality secondary path between a source and destination. The secondary path supports a "shadow" transmission of the data stream in order to mitigate transient delays and failures on the primary path.

Keywords: autonomic computing, overlay network, service composition, cross-layer cooperation, cross-platform cooperation, quality of service, TCP relay, multipath connection.

1 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 technologies [1], where systems are designed to respond dynamically to changes in the environment with only limited human guidance. One area in which

*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, and in part by National Science Foundation grants EIA-0000433, EIA-0130724, and ITR-0313142.

†Currently with Microsoft Corporation. Part of this research was conducted while the author was a graduate student at Michigan State University.

autonomic behavior can be particularly helpful is communication-related software; autonomic computing can be used to support fault tolerance, enhance security and improve quality-of-service in the presence of dynamic network conditions. This paper investigates the design of an extensible distributed infrastructure to support the development and deployment of autonomic communication services.

Realizing autonomic behavior involves cooperation of multiple software components. In the case of autonomic communication services, this interaction often requires both cross-layer (vertical) and cross-platform (horizontal) cooperation. Figure 1 illustrates an example in which a data stream is routed from a source to a destination, traversing three intermediate hosts. Such a situation can occur in environments, such as mobile ad hoc networks and overlay networks, where hosts are involved in routing of data. Let us assume in this example that packets are routed via the operating system kernel, but that a problem (for example, a sudden increase in packet loss rate) has occurred on the link connecting nodes A and B. A monitoring service detects this problem and triggers node A to intercept the data stream in the kernel and redirect it to a middleware layer (vertical cooperation), where it is "hardened" by encoding it with a stronger error control method, such as forward error correction. Node A informs node B (horizontal cooperation) that it also should intercept the data stream. The modified stream is forwarded to node B, which does as directed and passes the stream to middleware for decoding.

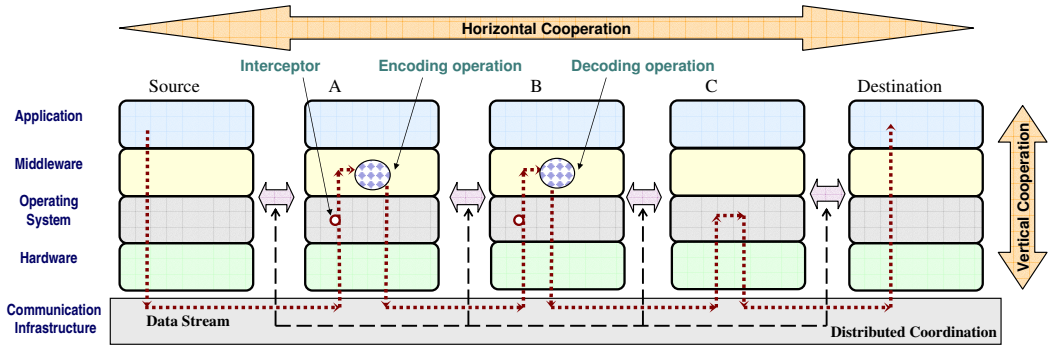


Figure 1: Vertical and horizontal cooperation.

Although this is a simple example, the need for adaptation has led to a variety of methods for implementing vertical cooperation [2–4]. Many involve adaptive middleware interacting with other system layers. While the traditional role of middleware is to hide resource distribution and platform heterogeneity from the business logic of applications, middleware is also an ideal location to implement many types of

self-monitoring and adaptive behavior [5–9]. Moreover, since middleware executes within the context of the application, it can exploit application-specific information in these tasks, but without requiring modification of the application code itself. In addition, numerous examples of horizontal cooperation have been studied, including dynamic composition of services among multiple nodes [10–14], and transcoding of data at intermediate nodes [15–17]. Many approaches involve the use of *overlay networks* [18], in which end hosts form a virtual network atop a physical network. The presence of *hosts* along the paths between nodes enables intermediate processing of data streams, without modifying the underlying routing protocols or router software.

Designing a system involving both vertical and horizontal cooperation is a challenging task, 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 for supporting such interactions can be very useful to the development and deployment of autonomic communication services. In this paper, we report on an investigation into the design and operation of such an infrastructure. The main result of this study is Service Clouds, a proposed architecture for how to organize the constituent services and use them to compose autonomic distributed applications. Figure 2 shows a conceptual view of Service Clouds. Individual nodes in the cloud use adaptive middleware and cross-layer collaboration to support autonomic behavior; an overlay network among these nodes serves as a vehicle to support cross-platform adaptation. Effectively, the nodes in the overlay network provide a "blank computational canvas" on which services can be instantiated as needed, and later reconfigured in response to changing conditions. The Service Clouds infrastructure is designed to be extensible: a suite of low-level services for local and remote interactions can be used to construct higher-level autonomic services.

We have implemented a prototype of Service Clouds[1] and experimented with it atop the PlanetLab Internet testbed [19]. Building and testing the prototype has revealed several key practical issues that must be addressed in distributed service-oriented frameworks. In this paper, we describe two case studies in which we used the Service Clouds prototype to construct new autonomic communication services that can be used to enhance a variety of distributed applications. The first is a TCP-Relay service, in which a node is selected and configured dynamically to serve as a relay for a data stream. Experiments demonstrate that our implementation, which is not optimized, can in many cases produce significantly better performance

[1] A Beta version of Service Clouds is available for download at http://www.cse.msu.edu/rapidware.

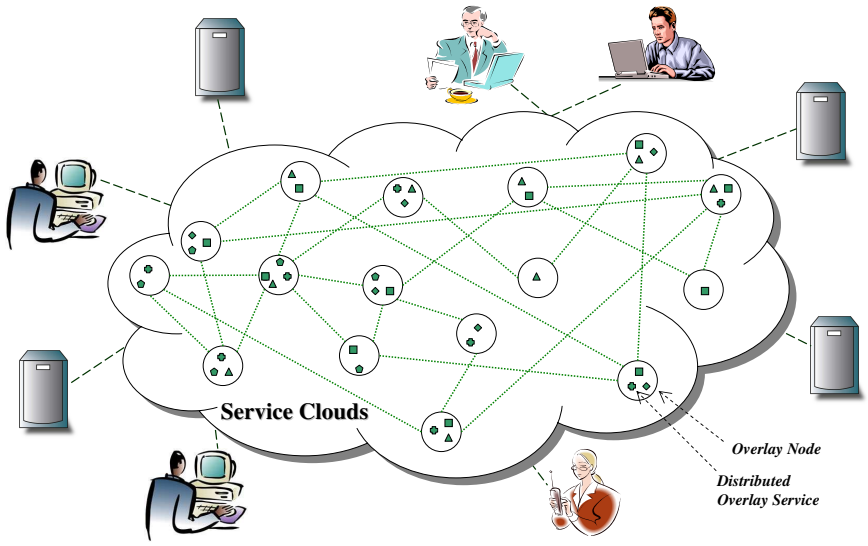


Figure 2: Conceptual view of the Service Clouds infrastructure.

than using a direct 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 in order to dynamically find and establish a high-quality secondary path, which is used as shadow connection to the primary path.

The remainder of this paper is organized as follows. Section 2 discusses related work. Section 3 describes the proposed architecture for Service Clouds. In Section 4 we describe the prototype implementation, present the two case studies and discuss the experimental results. Finally, in Section 5, we summarize the paper and discuss future directions.

2 Related Work

The Service Clouds infrastructure integrates and extends results from three primary areas of study in distributed systems. First, *adaptive middleware* [5–7, 20–23] enables distributed software to detect and respond to changing conditions in a manner that is orthogonal to, and in many cases independent of, the business logic of the application. Research in this area over the past several years has been extensive (see [24] for a recent 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. Second, cross-layer cooperation mechanisms [2–4, 25–27] enable the system to adapt in a coordinated

manner that is optimal for the system as a whole, and in ways not possible within a single layer. As described later, 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 [18] provide an adaptable and responsive chassis on which to implement communication services for many distributed applications [11, 13, 14, 17, 28–30]. Recently, researchers have investigated several ways to use overlay networks to support dynamically configurable communication services [10, 31–33]. In the remainder of this section, we briefly review several projects that are related to Service Clouds: Accord [34], iOverlay [31], CANS [11], GridKit [32], DSMI [35], and SpiderNet [10].

iOverlay [31] is a lightweight middleware that facilitates construction of overlay applications by providing a message switching engine and primitives for monitoring and reporting on system status (e.g., node/link failures, link delay, and channel throughput). In contrast, Service Clouds is intended to support the construction of higher-level services. Developers can plug algorithms for new services into the default Service Clouds infrastructure and easily customize the underlying components, if needed. As such, a developer could take advantage of iOverlay services by plugging them into the basic overlay services module of the Service Clouds architecture, discussed later.

Accord [34] is a component-based programming framework for constructing autonomic distributed applications. The main building block is the "autonomic component," which in addition to defining specified interfaces and dependencies, also equips the component with self-management rules and mechanisms. Self-management is realized by deploying agents that inject rules in autonomic components, thereby dynamically reconfiguring their behavior and interactions. While the Service Clouds architecture is also intended to facilitate development of autonomic distributed services, we focus primarily on the communication infrastructure, rather than on the programming framework. We have identified constituent services and how they can be composed to construct more complex communication services. Although our current prototype is written directly in Java, a programming framework such as Accord offers an appealing method to deploy autonomic service components within the Service Clouds architecture.

CANS [36] is an infrastructure for dynamic creation and reconfiguration of services along a data path, with a primary focus on connections between Internet hosts and mobile clients. Since mobile devices have limited resources, data may need to be suitably adapted as it approaches the wireless edge of the network. The CANS infrastructure is used to compose a path service that adapts a data stream by inserting, removing,

and reconfiguring components at intermediate nodes and end nodes. Service Clouds also can support adaptive path services, but uses an overlay network as its primary means of distributed coordination. A possible future project could integrate CANS software adaptation methods with the Service Clouds communication infrastructure.

GridKit [32] supports the notion of "plug-in" overlay networks by promoting interaction types (e.g., various flavors of request-reply, pub-sub, tuple space, P2P) to first class services. These overlay services can be used to provide service binding, resource discovery, resource management, and security. In the Service Clouds architecture and our initial PlanetLab implementation, we address distributed service management using overlays, focusing primarily on how to integrate vertical and horizontal adaptation with a single framework. The component interface abstractions proposed in GridKit to enable reconfiguration could be very useful to Service Clouds component interaction, and we intend to explore this line of study in future research.

DSMI [35] is a scalable resource-aware approach to distributed stream management. Specifically, this project addresses distributed and adaptive aggregation of streams at overlay nodes. An underlying layer performs dynamic network partitioning based on the availability of resources, producing a data-flow graph for efficient processing of the streams. DSMI uses a middleware framework to reconfigure overlay service nodes and processing components according to the data-flow graph. At the application layer, DSMI supports high-level descriptions of application data flows and processing operations. In supporting underlying communication services, Service Clouds could be used to deploy a management system such as DSMI.

SpiderNet [10] is a service composition framework for establishing high-quality fault-tolerant service paths among peer-to-peer systems. A key feature is a quality-aware bounded probing protocol for service composition. To compose a service, the probing mechanism tries to find a sequence of nodes that provide required functions with the specified quality (depending on the availability of resources). Then, the system composes a primary service path and maintains a number of possible service paths as backups. The proactive method for failure recovery, in soft realtime streaming, monitors the condition of backup paths with low-overhead probing and replaces a failed service path by a backup path. Recently, the SpiderNet model has been used in a hybrid approach that combines distributed probing with coarse-grained global state management for optimal component composition in streaming applications [37]. We view Service Clouds and SpiderNet as complementary, with the former focusing on dynamic instantiation and reconfiguration of services on "blank" nodes, and the latter focusing on composition of services already established

in the network.

3 Service Clouds Architecture

The architecture of Service Clouds has its roots in an earlier study [38], where we designed and constructed the Kernel-Middleware eXchange (KMX), a set of interfaces and services to facilitate collaboration between middleware and the operating system. We used the KMX to improve quality-of-service in video streams transmitted across wireless networks. Our primary focus in the KMX study was on vertical cooperation; the prototype was intended as a proof of concept, and the scenario described above involves very simple horizontal cooperation among nodes. However, the study yielded the concept of a *transient proxy* [38], whereby a service is instantiated on one or more hosts in the network, as needed, in response to changing conditions. An example is the processing on nodes A and B in Figure 1. Indeed, 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.

In the design of Service Clouds, we have incorporated lessons learned from the KMX study and have produced a more complete architecture to support vertical and horizontal cooperation. The architecture is designed with other developers in mind. Figure 3 shows a high-level view of the Service Clouds software organization and its relationship to Schmidt's model of middleware layers [39]. 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. 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.

Figure 4 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* (for example, transcoding a video stream for transmission on a low bandwidth link) and *Control Services* (for example, coordinating the corresponding encoding/decoding actions at the sender and receiver of the stream). *Basic Overlay Services* provide generic

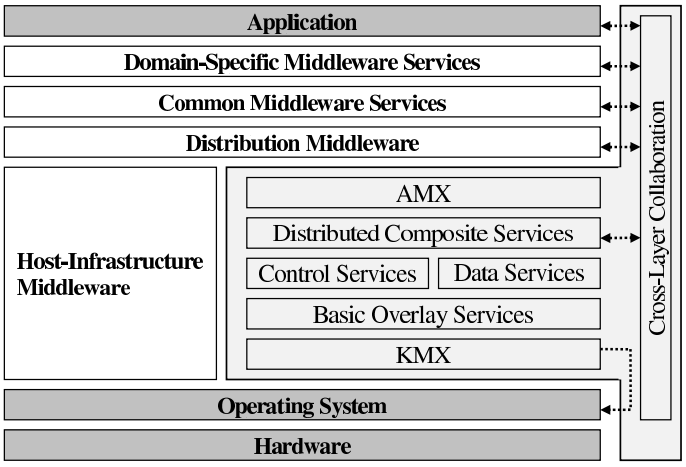


Figure 3: Relationship of Service Clouds to other system layers.

facilities for establishing an overlay topology, exchanging status information and distributing control packets among overlay hosts. Next, we discuss each group of services in turn, starting at the bottom and working upward.

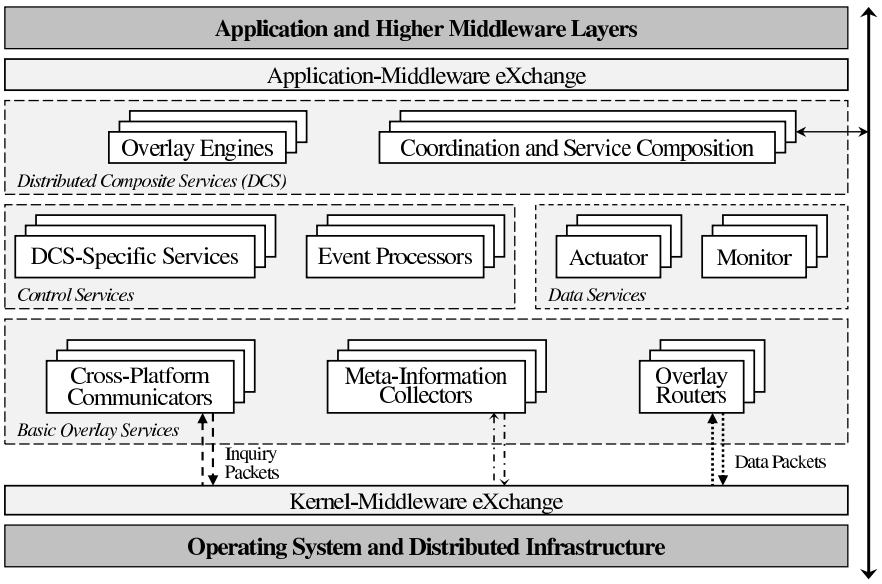


Figure 4: Service Clouds architecture.

Basic Overlay Services include three main types of components: *cross-platform communicators, meta-information collectors,* and *overlay routers*. Cross-platform communicators send and receive inquiry packets. Inquiry packets carry meta-information across platforms in a distributed system. The information provides information such as status of resources and availability of services on a node. 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. An overlay routing component not only forwards data packets among overlay nodes, but also supports their interception for intermediate processing.

Control Services include both *event processors* and *DCS-specific services*. An event processor handles specific control events and inquiry packets. For example, such a service might receive certain types of inquiry packets and extract information useful to multiple higher-level services. An event processor can also perform intermediate processing, for example, to construct statistical information on network conditions. On the other hand, 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* are used to 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; they have only limited decision making capabilities. Considering the example from Figure 1, an actuator can insert an FEC filter on a data stream according to a predefined set of rules and the current conditions. 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 running on the node do not crash. Transient proxies are adaptors that manipulate data streams on intermediate nodes on their route. 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 enabling mechanism for data adaptation is interception, which is usually transparent to the application.

The Distributed Composite Services unit includes two types of components: *overlay engines* and *co-ordination and service composition*. An overlay engine executes a particular distributed algorithm across overlay nodes. Examples include building and maintaining a multicast tree for content distribution, establishing redundant overlay connections between a source and destination, and identifying and configuring relay nodes to improve throughput in bulk data transfers. Coordination and service composition refers to oversight of several supporting activities needed to vertical and horizontal cooperation. For example, a co-ordinator can determine to use a suitable 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 some pre-processing of received pieces of information and discard useless ones early to avoid unnecessary resource consumption.

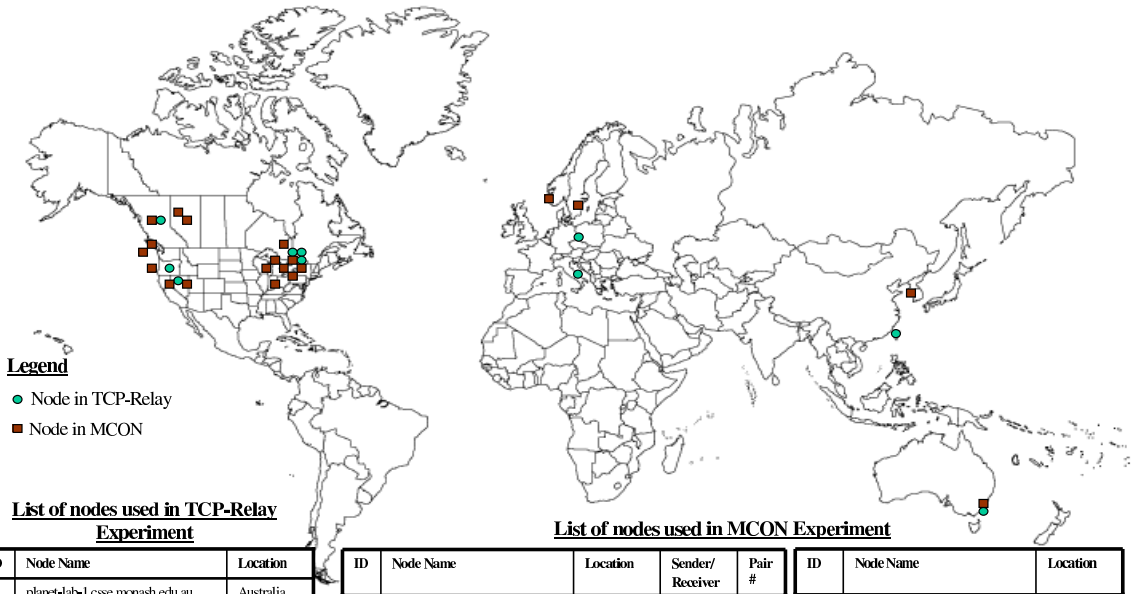
Finally, as we mentioned in the previous section, the Service Clouds architecture includes two bounding strata: the KMX and the AMX. The KMX [38] provides a platform-independent interface to OS- and network-specific services and information. For example, KMX facilities provide information on local system resource availability, report the status of connections to other nodes, and (if the operating system provides such services) intercept data streams at the OS level and redirect them to transient proxies. The AMX layer provides an interface that enables applications and higher-level middleware layers to communicate with the Service Clouds infrastructure.

4 PlanetLab Case Studies

We have implemented a prototype of the Service Clouds architecture and used it to conduct two case studies on PlanetLab [19], an Internet research testbed comprising hundreds of Linux-based Internet nodes distributed all over the world. The first case study involves automatic (and transparent) deployment of TCP relays, whereby data streams are redirected through a selected overlay node. Experimental results show that for bulk data transfers, use of a TCP relay can reduce total delay substantially, relative to the native TCP/IP connection. The second case study investigates the deployment of MCON, a service for creating resilient network connections by dynamically establishing one or more high-quality backup paths between a source and destination node. Figure 5 shows the PlanetLab nodes used in the case studies. We begin by describing details of the Service Clouds implementation. Then, for each case study, we describe the basic operation of the service, its details of the implementation, and the experimental results.

4.1 Prototype Implementation

Figure 6 depicts the Service Clouds prototype, including specific components used to construct the TCP-Relay and MCON services. Components labeled with an *R* are used in the TCP-Relay implementation, while those labeled with an *M* are used in the MCON implementation. The Service Clouds prototype is written primarily in Java, but components can incorporate modules written in other languages, using the



List of nodes used in TCP-Relay Experiment

ID	Node Name	Location
0	planet-lab-1.csse.monash.edu.au	Australia
1	plab1.cs.ust.hk	Hong Kong (China)
2	planetlab1.cs.cornell.edu	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

List of nodes used in MCON Experiment

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	S. Korea	Sender	5
9	blast.cs.uwaterloo.ca	Canada	Receiver	

ID	Node Name	Location
10	pli1-pa-4.hpl.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

Figure 5: PlanetLab nodes used in the experiments.

Java Native Interface (JNI) to interact with them.

The prototype software has a main driver, which deploys coordination and service composition. It reads configuration files containing the IP addresses of overlay nodes and the overlay topology, instantiates a basic set of (composite) components as threads, and configures the components according to the default or specified parameter values. Examples of these 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.

The AMX interface to the infrastructure is implemented using local TCP sockets. Components such as meta-information collectors implement the *singleton pattern* [40], 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. Some service primitives are transient, in that they are instantiated as needed to compose a particular service. An example is the data router used to realize a *TCP*

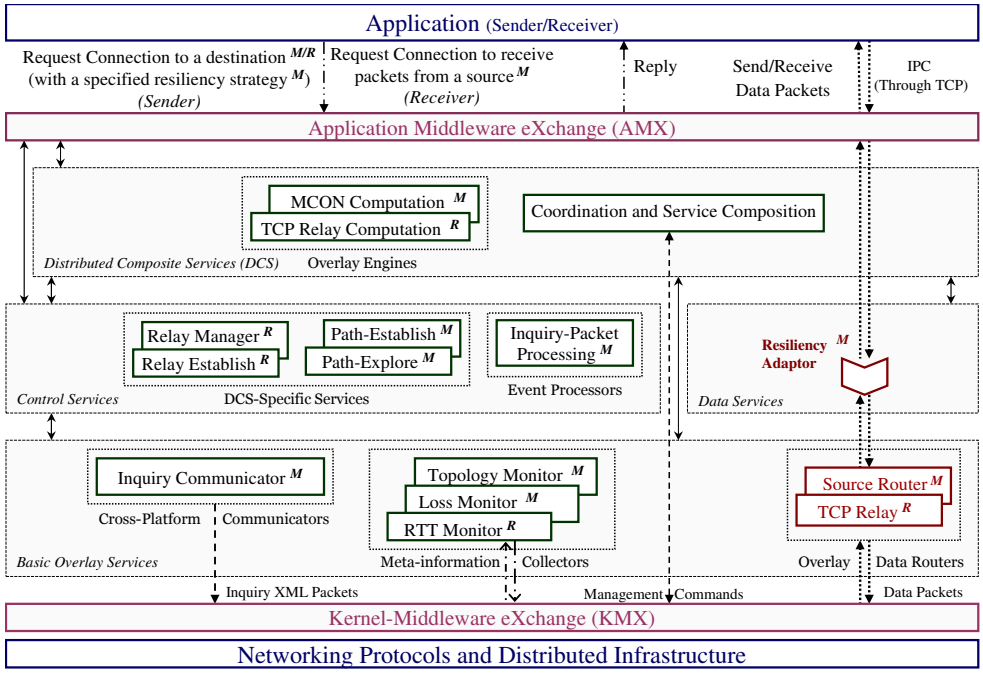


Figure 6: The Service Clouds prototype.

relay, which is dynamically created at a selected overlay node. In this prototype, such remote transient services are created by sending an appropriate request to the configuration unit on the remote node, which spawns the necessary threads. However, other software packages such as H20 [41], could also be integrated into Service Clouds for this purpose.

The format of inquiry packets exchanged among nodes is XML, and we use DOM [42] to enable Service Clouds components to access and manipulate their content. Although DOM may not be as fast as the other alternatives, such as SAX [43], 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 [44], also address concerns about bandwidth consumption and the speed of processing XML documents.

Finally, the prototype includes a collection of scripts to support execution on PlanetLab. These scripts are used to configure nodes, update code on the nodes, launch the infrastructure, setup and run test scenarios, and collect results. To manage the infrastructure distributed on several nodes, we have used Java Message Service (JMS) [45]. We have implemented a simple set of remote management commands that enable gathering of statistics (for example, obtaining number of inquiry packets received at each node), to change

framework parameters at run time, and to shut down the distributed infrastructure. Now, let us turn to the case studies.

4.2 TCP Relays for High Performance Bulk Data Transfer

Early overlay networks provided UDP-based routing and data forwarding services, essentially using application-layer entities to emulate network-layer functionality. However, a set of recent studies indicate that application-level relays in an overlay network can actually improve TCP throughput for long-distance bulk transfers [46–48]. Specifically, due to the dependence of TCP throughput on round trip time (RTT), 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 [49–51], which require either router support or kernel modifications. Alternatively, TCP relays can be used in tandem with such techniques.

Basic Operation. Figure 7 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 derived using the following formula [52]:

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

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 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

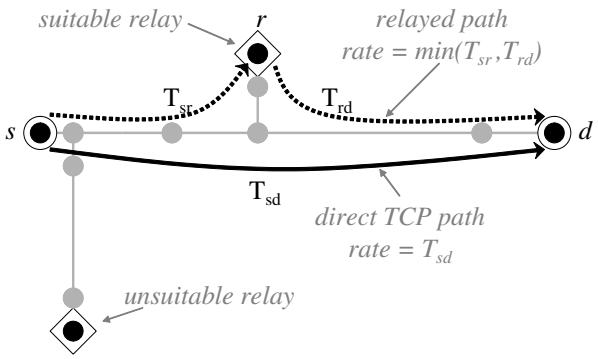


Figure 7: An example of a TCP relay.

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 as needed. Second, the initiator of the transfer must use the measurement results to predict the TCP throughput for all possible choices of relay and select the best option (which may be the direct Internet path if no good relay exists). 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. As we will see below, these requirements map neatly to the separation of concerns among particular component types in the Service Clouds architecture. Furthermore, the natural modularization provided by the Service Clouds architecture provides clear implementation points for future performance improvements and functional enhancements.

Implementation Details. Figure 6 shows the architectural relationships among various Service Clouds components. Those used in the TCP-Relay service are labeled with an R. Four components collaborate to address the four requirements described above. First, the *RTT Monitor* component encapsulates the collection and distribution of delay measurements. It directly measures RTT between the local node and all neighbors by periodically sending "ping" messages in a low priority background thread. Another back-

ground thread implements a simple UDP-based protocol for requesting the RTT between any two remote nodes, to maintain a local view of delay conditions across the entire overlay. Other local components query RTT Monitor to obtain RTT information. Although the current prototype does not measure loss rates, an analogous *Loss Monitor* component would occupy a similar position in the architecture as another type of Meta-information Collector. 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 Meta-Information Collectors such as the RTT Monitor to provide input for its computation. Third, the *Relay Establish* and *Relay Manager* components collectively implement the signalling protocol to setup and control a relay. Relay Establish exposes an interface by which a local application (via AMX) can initiate the setup process. The Relay Manager operates at the relay node to allocate resources in response to requests for relay service. Finally, the *TCP Relay* component implements data forwarding at the relay node, running in a transient thread instantiated by the Relay Manager for the duration of a particular transfer. This component currently performs data forwarding in user space by transferring data between two sockets. However, a more advanced implementation might make use of KMX to invoke kernel-level support for more efficient data forwarding.

Figure 8 illustrates an example of the distributed operation of the TCP-Relay prototype, in which an application sender (on the left) uses the service to transfer data via a relay (on the right) to a receiver (not shown). The interactions among the four Service Clouds components are numbered to indicate the sequence of steps required to complete the transfer, which we now describe.

The sender issues a request to AMX for a connection to a particular receiver address (step 1). Upon receiving the application request, the AMX layer invokes the TCP Relay Computation component to find a suitable relay (step 2). The TCP Relay Computation engine queries the RTT Monitor component to obtain the RTTs between the node and all of its neighbors as well as the RTTs between its neighbors and the intended receiver (step 3), using this information to select a relay node. The address of the chosen relay is returned to AMX (step 4).

Having been informed of the relay, AMX invokes the Relay Establish component to request service (step 5), causing Relay Establish to signal the Relay Manager at the relay node (step 6). The Relay Manager spawns a thread in which to run a new instance of the TCP Relay component (step 7). Using socket system calls, the TCP Relay opens a TCP connection to the receiver and also begins listening for an incoming con-

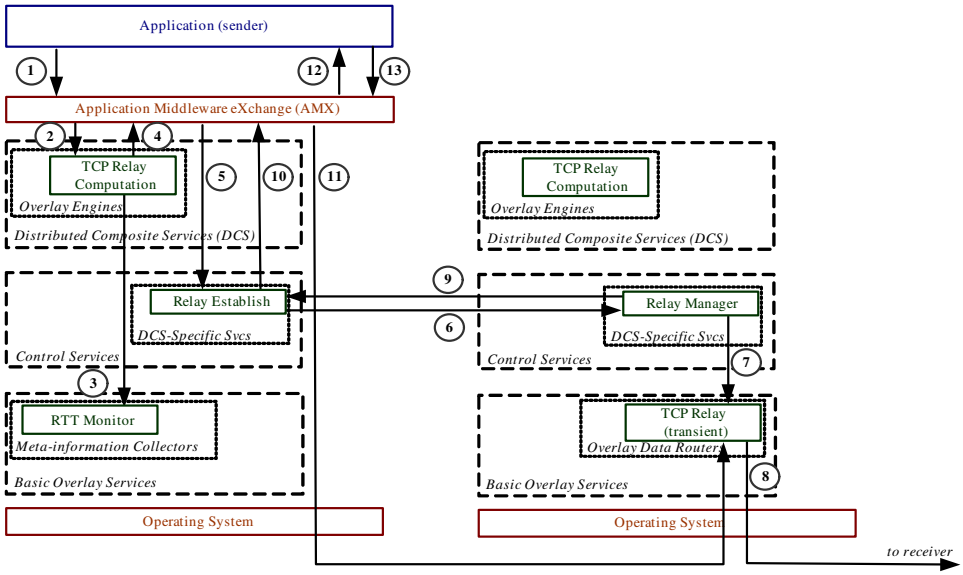


Figure 8: An example of TCP-Relay run-time operation.

nection from the sender (step 8). The Relay Manager remains blocked until the connection to the receiver is established; once unblocked, it signals Relay Establish at the sender, indicating the port on which the newly created TCP Relay can be contacted (step 9). AMX receives control along with the relay port number (step 10) and opens a TCP connection to the relay (step 11). AMX also creates an inter-process communication (IPC) port on which it will accept data and returns control to the application (step 12).

With the infrastructure now in place to support the transfer, the application connects to the IPC port and begins to send data (step 13). AMX forwards the data over the TCP connection created in step 11. The TCP Relay component reads this data and forwards it over the connection created in step 8. When the data transfer is complete, the application closes the IPC socket, triggering AMX to close its connection to the relay, which in turn closes its connection to the receiver. When all connections are closed, TCP Relay terminates its thread, thereby removing any transient state associated with the transfer.

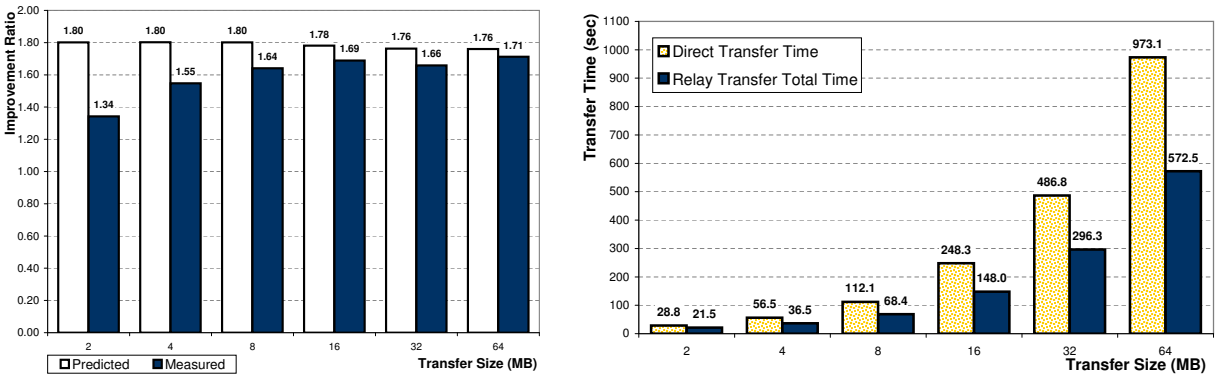
Experimental Results. To evaluate the TCP-Relay service, we constructed a small overlay network comprising 10 PlanetLab nodes. In these tests, all transfers originate and terminate at overlay nodes, possibly using other overlay nodes as relays. As we expect the TCP-Relay service to be particularly useful for long distance transfers, we selected a set of geographically dispersed nodes across four continents, as illustrated in Figure 5.

To understand the network-wide behavior of the TCP-Relay service, we conducted an exhaustive set of data transfers between all possible sender-receiver pairs. Since any PlanetLab node can reach any other over the Internet, the overlay network for our experiment can be represented as a complete directed graph with ten vertices and edges representing the 90 possible sender-receiver pairs. For each sender-receiver pair, any one of the eight remaining overlay nodes is a candidate relay. To evaluate the performance of the prototype 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*.

For transfer sizes ranging from 2 to 64MB, we repeatedly cycle through all possible sender-receiver pairs collecting one measurement of improvement ratio per pair per cycle. We ran this experiment continuously for a period of roughly two weeks. Note that there is no guarantee that the same relay will be selected across all measurements for the same sender-receiver pair. While we observed some changes in the selected relay, we also observed several sender-receiver pairs with highly stable relays that provided a significant improvement in performance.

Since our focus in this paper is on the architecture and use of Service Clouds, we present only a sample of the results. A more complete report on the performance of the TCP relay service is in preparation. Figure 9 shows results for a particular sender-receiver pair in our experiment. The sender (ID 0 in Figure 5) is in Australia, and the receiver (ID 1 in Figure 5) is in Hong Kong. For this pair, our prototype service consistently selected a relay on the East Coast of the United States (ID 2 in Figure 5). 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 (Abeline) 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 Abeline before rejoining the direct route in Chicago.

Figure 9(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.



(a) average improvement ratio

(b) average transfer time

Figure 9: TCP-Relay results for a selected pair.

The plot on the left shows that the measured improvements are close to the theoretical prediction. It is worth emphasizing that this performance improvement is entirely attributable to the relay's reduction of TCP feedback delay. We also observe the amortization of startup latency at larger transfer sizes, which can also be seen in comparisons of the durations of direct and relayed transfers in Figure 9(b). At 64MB, the largest transfer in our experiment still constitutes a relatively small bulk transfer. Yet even at this scale, we see a significant reduction in transfer time – from about fifteen to about ten minutes.

4.3 MCON for Highly Resilient Connectivity

Establishing multiple connections (for example, 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 distributed applications such as high-reliability conferencing, where uninterrupted communication is essential. In overlay networks, selection of high quality primary and backup paths is a challenging problem, however, due to 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 [53], a distributed algorithm for establishing 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 [54] 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 10. 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.

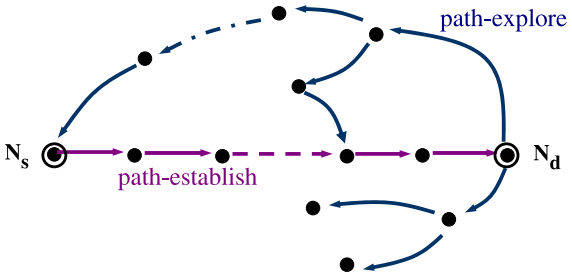


Figure 10: Basic operation of MCON.

Implementation Details. Now, let us describe the implementation of MCON in terms of the Service Clouds infrastructure (the components labeled with an M in Figure 6). The *AMX* interface requires the caller to simply specify the destination node and, optionally, a resiliency strategy for finding a secondary path. In the current 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 [54].

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 the MCON Computation 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, as described earlier. 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.

Basic overlay services used in MCON are as follows. The *Inquiry Communicator* processes arriving path-establish and path-explore packets, extracting information required by the overlay engine. The *Topology Monitor* and *Loss Monitor* components provide topology and loss rate information between a node and its neighbors. To gather topology information, this implementation invokes the Linux/Unix "tracepath" tool. The packet loss rate to the neighbors is measured by sending and receiving beacons and acknowledgements. The *Source Router* implements a source routing mechanism to route data packets through the overlay network along a predetermined path. Specifically, after finding a (primary or secondary) path to the destination, the overlay service on the source uses this mechanism to insert an application-level header in each outgoing data packet, indicating which overlay nodes should be traversed on the path toward the destination.

Experimental Results. In this set of experiments, we selected 20 Planetlab nodes (as shown 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

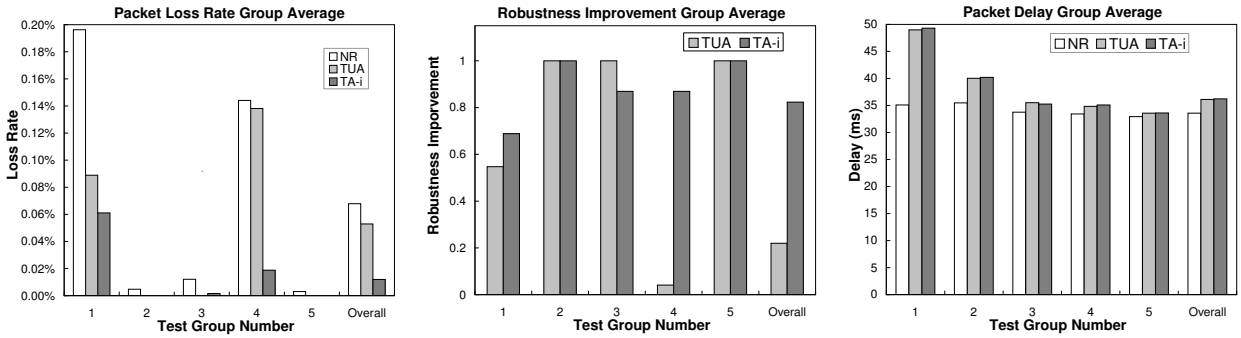
separate experimental runs.

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,

$$I = \frac{F_s - F_d}{F_s},$$

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 11 shows the results for different test groups (aggregated according to sender-receiver pairs) and the overall results. Figure 11(a) demonstrates that compared with NR, both TUA and TA-i can substantially reduce the average packet loss rate. Figure 11(b) shows TA-i exhibits higher robustness improvement than TUA, except in group 3. Finally, while the use of the Service Clouds infrastructure introduces additional computational and communication overhead, Figure 11(c) shows that except for group 1, the effect on data packet delay is relatively small, when compared to NR.



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

Figure 11: MCON empirical results on PlanetLab.

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 Service Clouds infrastructure.) Further, we note that the experiments described here were conducted under "normal" network conditions. The true value of the

MCON service is to be demonstrated in situations where serious failures occur along the primary path, and a high-quality secondary path continues to deliver the data stream to the destination.

5 Conclusions and Future Work

In this paper, we introduced the Service Clouds infrastructure for composing autonomic communication services. Service Clouds combines dynamic software configuration methods with overlay network services in order to support both cross-layer and cross-platform cooperation. We described two experimental case studies using a Service Clouds prototype on the PlanetLab testbed. The first is dynamic selection, creation and use of TCP relays for bulk data transfer, and the second is dynamic construction of secondary paths for highly resilient streaming. These services demonstrate the usefulness of the Service Clouds infrastructure from a development stand point, and the experimental performance results indicate that these services may be helpful to many distributed applications.

Our ongoing research addresses several issues. First, we plan to conduct additional experiments with TCP-Relay (using link loss status in selecting relay nodes, experiments with the use of multiple relays, and dynamic migration of relays) and MCON (response time to handle emulated failures of the primary path, use of more than two paths). Second, we plan to deploy Service Clouds in two additional environments: the wireless edge of the Internet, where Service Clouds can be used to dynamically configure and migrate proxies for wireless nodes, and mobile ad hoc networks, where Service Clouds can be used to enhance QoS of streams transmitted across the network. We have studied these domains previously [55, 56], but the Service Clouds prototype will enable us to focus on software design and evolution issues, in addition to performance. Third, we will conduct a more complete study of the extension of Service Clouds into the OS kernel, by way of the KMX. We have done some of this work using a local testbed [38], but we need to conduct experiments on wide area network other than PlanetLab, which does not allow access to the kernel. Finally, we plan to extend our recent work on machine learning for autonomic decision making [57] to the Service Clouds prototype.

Further Information. Related publications on the RAPIDware 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, January 2003.

[2] B. D. Noble, M. Satyanarayanan, D. Narayanan, J. E. Tilton, J. Flinn, and K. R. Walker, "Agile application-aware adaptation for mobility," in *Proceedings of the Sixteen ACM Symposium on Operating Systems Principles*, pp. 276–287, October 1997.

[3] S. V. Adve, A. F. Harris, C. J. Hughes, D. L. Jones, R. H. Kravets, K. Nahrstedt, D. G. Sachs, R. Sasanka, J. Srinivasan, and W. Yuan, "The Illinois GRACE Project: Global Resource Adaptation through CoopERation," in *Proceedings of the Workshop on Self-Healing, Adaptive, and self-MANaged Systems (SHAMAN)*, June 2002.

[4] C. Poellabauer, H. Abbasi, and K. Schwan, "Cooperative run-time management of adaptive applications and distributed resources," in *Proceedings of the 10th ACM Multimedia Conference*, (France), pp. 402–411, December 2002.

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

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

[7] 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 *Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms and Open Distributed Processing*, pp. 121–143, April 2000.

[8] T. Ledoux, "OpenCorba: A reflective open broker," *Lecture Notes in Computer Science*, vol. 1616, pp. 197–214, July 1999.

[9] S. M. Sadjadi and P. K. McKinley, "ACT: An adaptive CORBA template to support unanticipated adaptation," in *Proceedings of the 24th IEEE International Conference on Distributed Computing Systems (ICDCS)*, (Tokyo, Japan), March 2004.

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

[11] X. Fu, W. Shi, A. Akkerman, and V. Karamcheti, "CANS: composable and adaptive network services infrastructure," in *The 3rd USENIX Symposium on Internet Technology and Systems*, (San Franscisco, California), March 2001.

[12] D. Xu and X. Jiang, "Towards an integrated multimedia service hosting overlay," in *Proceedings of ACM Multimedia 2004*, (New York, NY), October 2004.

[13] X. Gu, K. Nahrstedt, R. N. Chang, and C. Ward, "QoS-assured service composition in managed service overlay networks," in *Proceedings of the 23rd IEEE International Conference on Distributed Computing Systems (ICDCS 2003)*, (Providence, Rhode Island), May 2003.

[14] B. Raman, S. Agarwal, Y. Chen, M. Caesar, W. Cui, P. Johansson, K. Lai, T. Lavian, S. Machiraju, Z. M. Mao, G. Porter, T. Roscoe, and Mukund, "The SAHARA model for service composition across multiple providers," in *Proceedings of the First International Conference on Pervasive Computing, also Lecture Notes In Computer Science; Vol. 2414*, pp. 1–14, Springer-Verlag, August 2002.

[15] M. Yarvis, P. L. Reiher, and G. J. Popek, "Conductor: A framework for distributed adaptation," in *Proceedings of the Seventh Workshop on Hot Topics in Operating Systems*, (Rio Rico, Arizona), pp. 44–49, March 1999.

[16] A. Nakao, L. L. Peterson, and A. C. Bavier, "Constructing end-to-end paths for playing media objects.," *Computer Networks*, vol. 38, no. 3, pp. 373–389, 2002.

[17] 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.

[18] D. Andersen, H. Balakrishnan, F. Kaashoek, and R. Morris, "Resilient Overlay Networks," in *Proceedings of 18th ACM Symposium on Operating Systems Principles (SOSP 2001)*, 2001.

[19] L. Peterson, T. Anderson, D. Culler, and T. Roscoe, "A Blueprint for Introducing Disruptive Technology into the Internet," in *Proceedings of HotNets–I*, (Princeton, New Jersey), October 2002.

[20] 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.

[21] G. S. Blair, G. Coulson, A. Andersen, L. Blair, F. C. Michael Clarke, H. Duran-Limon, T. Fitzpatrick, L. Johnston, R. Moreira, N. Parlavantzas, and K. Saikosk, "The Design and Implementation of Open ORB 2," *IEEE DS Online, Special Issue on Reflective Middleware*, vol. 2, no. 6, 2001.

[22] R. Baldoni, C. Marchetti, A. Termini, "Active software replication through a three-tier approach," in *Proceedings of the 22th IEEE International Symposium on Reliable Distributed Systems (SRDS02)*, (Osaka, Japan), pp. 109–118, October 2002.

[23] B. Redmond and V. Cahill, "Supporting unanticipated dynamic adaptation of application behaviour," in *Proceedings of the 16th European Conference on Object-Oriented Programming*, (Malaga, Spain), Springer-Verlag, June 2002. volume 2374 of Lecture Notes in Computer Science.

[24] P. K. McKinley, S. M. Sadjadi, E. P. Kasten, and B. H. C. Cheng, "A taxonomy of compositional adaptation," Tech. Rep. MSU-CSE-04-17, Computer Science and Engineering, Michigan State University, East Lansing, Michigan, May 2004.

[25] C. Ellis, "The case for higher level power management," in *Proceedings of the Seventh Workshop on Hot Topics in Operating Systems*, p. 162, IEEE Computer Society, March 1999.

[26] H. Zeng, C. Ellis, A. Lebeck, , and A. Vahdat, "ECOSystem: Managing Energy as a First Class Operating System Resource," in *Proceedings of the 10th International Conference on Architectural Support for Programming Languages and Operating Systems*, (San Jose, California), pp. 123–132, October 2002.

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

[28] K. N. Jingwen Jin, "QoS service routing in one-to-one and one-to-many scenarios in next-generation service-oriented networks," in *Proceedings of The 23rd IEEE International Performance Computing and Communications Conference (IPCCC2004)*, (Phoenix, Arizona), April 2004.

[29] J. W. Byers, J. Considine, M. Mitzenmacher, and S. Rost, "Informed content delivery across adaptive overlay networks," *IEEE/ACM Transactions on Networking (TON)*, vol. 12, pp. 767–780, October 2004.

[30] 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.

[31] B. Li, J. Guo, and M. Wang, "iOverlay: A lightweight middleware infrastructure for overlay application implementations," in *Proceedings of the Fifth ACM/IFIP/USENIX International Middleware Conference, also Lecture Notes in Computer Science*, vol. 3231, (Toronto, Canada), pp. 135–154, October 2004.

[32] 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 *Proceedings of International Symposium on Distributed Objects and Applications(DOA)*, (Larnaca, Cyprus), pp. 1463–1481, October 2004.

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

[34] H. Liu, M. Parashar, and S. Hariri, "A component-based programming model for autonomic applications.," in *Proceedings of the 1st International Conference on Autonomic Computing*, (New York, NY, USA), pp. 10–17, IEEE Computer Society, May 2004.

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

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

[37] X. Gu, P. S. Yu, and K. Nahrstedt, "Optimal component composition for scalable stream processing," in *Proceeedings of the 25th International Conference on Distributed Computing Systems (ICDCS 2005)*, (Columbus, OH, USA), pp. 773–782, IEEE Computer Society, June 2005.

[38] F. A. Samimi, P. K. McKinley, S. M. Sadjadi, and P. Ge, "Kernel-middleware interaction to support adaptation in pervasive computing environments," in *Proceedings of the 2nd Workshop on Middleware for Pervasive and Ad-Hoc Computing*, (Toronto, Ontario, Canada), pp. 140–145, ACM Press, October 2004.

[39] D. C. Schmidt, "Middleware for real-time and embedded systems," *Communications of the ACM*, vol. 45, pp. 43–48, June 2002.
[40] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements od Reusable Object-Oriented Software*. Addison-Wesley Professional Computing Series, New York, NY: Addison-Wesley Publishing Company, 1995.
[41] D. Kurzyniec, T. Wrzosek, D. Drzewiecki, and V. S. Sunderam, "Towards self-organizing distributed computing frameworks: The H2O approach," *Parallel Processing Letters*, vol. 13, no. 2, pp. 273–290, 2003.
[42] W3C, *Document Object Model (DOM) Level 2 Core Specification*, version 1.0 ed., November 2000.
[43] SAX, "Simple API for XML." `http://www.saxproject.org/`.
[44] W3C, "Xml binary characterization working group public page." `http://www.w3.org/XML/Binary/`.
[45] Sun Microsystems, Inc., *JavaTM Message Service Specification*, version 1.1 ed., April 2002.
[46] M. Swany, "Improving throughput for grid applications with network logistics," in *Proceedings of IEEE/ACM Conference on High Performance Computing and Networking*, November 2004.
[47] Y. Liu, Y. Gu, H. Zhang, W. Gong, and D. Towsley, "Application level relay for high-bandwidth data transport," in *Proceedings of the First Workshop on Networks for Grid Applications (GridNets)*, (San Jose), October 2004.
[48] H. Pucha and Y. C. Hu, "Overlay TCP: Multi-hop overlay transport for high throughput transfers in the Internet," Tech. Rep. TR-05-08, Dept. Electrical and Computer Engineering, Purdue University, March 2005.
[49] D. Katabi, M. Handley, and C. Rohrs, "Congestion control for high bandwidth-delay product networks," in *Proceedings of the ACM SIGCOMM*, 2002.
[50] S. Floyd, "Highspeed TCP for large congestion windows," Tech. Rep. RFC 3649, IETF, 2003.
[51] C. Jin, D. X. Wei, and S. H. Low, "Fast TCP: Motivation, architecture, algorithms, performance," in *Proceedings of the IEEE INFOCOM*, March 2004.
[52] M. Mathis, J. Semke, and J. Madhavi, "The macroscopic behavior of the TCP congestion avoidance algorithm," *ACM Computer Communications Review*, vol. 27, no. 3, pp. 67–82, 1997.
[53] C. Tang and P. K. McKinley, "Improving mutipath reliability in topology-aware overlay networks," in *Proceedings of the Fourth International Workshop on Assurance in Distributed Systems and Networks (ADSN), held in conjunction with the 25th IEEE International Conference on Distributed Computing Systems*, (Columbus, Ohio), June 2005.
[54] C. Tang and P. K. McKinley, "On the cost-quality tradeoff in topology-aware overlay path probing," in *Proceedings of International Conference on Network Protocols (ICNP)*, pp. 268–279, November 2003.
[55] P. K. McKinley, U. I. Padmanabhan, N. Ancha, and S. M. Sadjadi, "Composable proxy services to support collaboration on the mobile internet," *IEEE Transactions on Computers (Special Issue on Wireless Internet)*, pp. 713–726, June 2003.
[56] C. Tang and P. K. McKinley, "Energy optimization under informed mobility," *IEEE Transactions on Parallel and Distributed Systems*, 2006. in press.
[57] E. P. Kasten and P. K. McKinley, "Meso: Perceptual memory to support online learning in adaptive software," in *Proceedings of the 3rd International Conference on Development and Learning (ICDL'04)*, (La Jolla, California), October 2004.