A Framework for Highly-Available Composed Real-Time Internet Services

Bhaskaran Raman
Qualifying Examination Proposal
CS Division, EECS Department, U.C.Berkeley, CA 94720, USA
bhaskar@cs.berkeley.edu
Phone: +1-510-642-8284, Fax: +1-510-642-5775
Committee: Profs. Anthony Joseph (Chair), Randy Katz, Ion Stoica, David Brillinger

Abstract

Application services for the end-user are all important in today's communication networks, and could dictate the success or failure of technology or service providers [39]. It is important to develop and deploy application functionality quickly [18]. The ability to *compose* services from independent providers provides a flexible way to quickly build new end-to-end functionality. Such composition of services across the network and across different service providers becomes especially important in the context of growing popularity and heterogeneity in 3G+ access networks and devices [25].

In this work, we provide a framework for constructing such composed services on the Internet.

Robustness and high-availability are crucial for Internet services. While cluster-based models for resilience to failures have been built for web-servers [27] as well as proxy services [11, 4], these are inadequate in the context of composed services. This is especially so when the application session is *long-lived*, and failures have to be handled *during* a session.

In the context of composed services, we address the important and challenging issues of resilience to failures, and adapting to changes in overall performance during long-lived sessions.

Our framework is based on a connection-oriented overlay network of compute-clusters on the Internet. The overlay network provides the context for composing services over the wide-area, and monitoring for liveness and performance of a session. We have performed initial analyses of the feasibility of network failure detection over the wide-area Internet. And we have a preliminary evaluation of the overhead associated with such an overlay network. We present our plans for further evaluation and refinement of the architecture; and for examining issues related to the creation of the overlay topology.

1 Introduction

Application functionality for the end-user is the driving force behind development of technology and its adoption. It is important to be able to develop and deploy new functionality quickly [18]. Composition of existing services to achieve new functionality enables such quick development through reuse of already deployed components. Consider for example, the scenario shown in Figure 1.

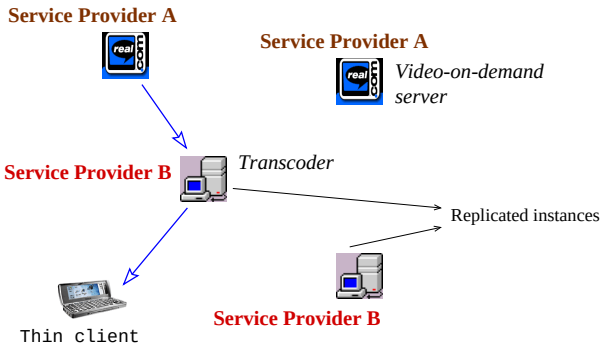


Figure 1. Service composition: an example

A video-on-demand service is deployed by provider A. We wish to enable this functionality in a novel handheld in a wireless access network. Since the device capabilities are limited, we may need to transcode, reduce the frame size and/or rate, or in any other way adapt to the device capabilities. The transformation agent necessary for this is deployed by service provider B. The wireless service provider "composes" these two services to reach the video to the end-user.

Such composition enables quick development of functionality since the wireless service provider need not deploy the other services. Not only is the time to code development reduced because of reuse of existing functionality, but also, the wireless provider does not have the hassle of deploying and maintaining the other services.

We have the notion of a *service-level path* of composed or cascaded *services*. In the rest of the paper, unless qualified otherwise, we shall use the term "service" as in this context.

Service composition is especially important since the Internet is fast growing to be a middleware service platform [38]. Third generation mobile and cellular systems and beyond are currently being standardized and will soon be deployed by service providers [31, 28]. The number and diversity of mobile devices and access networks are exhibiting a growing trend [25]. Several architectures have been proposed for the integration of services across such heterogeneity [38, 22, 37, 12].

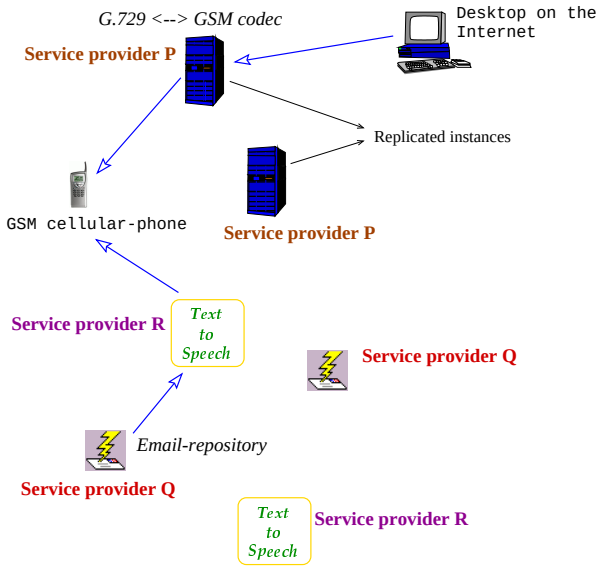


Figure 2. Application scenarios with heterogeneous networks

Two other examples of service composition in the context of such integration are shown in Figure 2. In the example, the middleware platform for service integration consists of different transformation agents. In the first example, an Internet host using a G.729 audio coding communicates with a GSM cellular phone. The audio stream goes through an appropriate codec deployed by an independent provider P. (This can be considered a simple case of composition – with only one intermediate service). In the second example, the email service of provider Q is composed with a text-to-speech conversion agent deployed by provider R to enable the user to listen to emails over cellular-phone.

Services may need to be composed across the network since different service providers may deploy their services at different locations. Furthermore, some services may have specific hardware implementations for performance reasons. In spite of the strong motivation for such composed services, there exists no framework today to compose services across the network, across service providers.

In this work, we provide a framework for constructing such composed services on the Internet.

Robustness and high-availability to the end-user are crucial for Internet services. The availability requirements for real-time or interactive communication services are especially stringent [24]. Furthermore, in many cases service-level paths could persist for a long time. That is, there is a *long-lived session*: for a few minutes, or may be even hours. It becomes important to keep track of the liveness and performance of a session as long as it lives.

In the context of composed services, we address the important and challenging issues of resilience to failures, and adapting to changes in overall performance during long-lived sessions.

1.1 Requirements and Open Challenges

A framework for robust composition of services should include at least the following:

1. A design for how services are deployed and replicated by the different service providers; and a design for how services are located, who composes services and how a service-level path is created.

2. Importantly, we need mechanisms for resilience to different kinds of failures. We need to detect failures and recover from them.

3. When there are multiple replicas of each service, at different points in the network, we need to be able to choose between the many service-level paths that are possible. That is, we need to optimize the service-level path we construct according to some performance metric; and we need to have this performance information available.

The first issue of composition of services across service providers and across the network is a challenging problem. The design needs to allow enough flexibility for new service providers to add their services; but there should also be right level of coupling between services to allow composition. That is, service cannot be completely ignorant of one another. The TACC model for proxy services [11] addresses this, but does so within a single service-provider cluster. The mechanism to compose services across the network has not been addressed adequately so far.

The second issue of robustness of Internet services has been an important topic of research. Clusters have been used as a platform for construction and management of robust Internet services [11, 4, 27]. While clusters provide a natural platform for redundancy, the mechanisms are inadequate in the context of a service-level path that extends

across the network. Furthermore, while clusters handle *process/machine level failures*, they provide little support when it comes to a network partition between itself and the client – when there is a *network failure*.

Service-level paths could stretch across the wide-area, and an end-to-end composed session could be *long-lived*: last for several minutes to probably a few hours. Now, network partitions and routing flaps on the Internet occur often, and could persist for a long time [23]. Hence we are faced with the challenge of providing continued service to the end-user in the presence of such failures. This is especially important if the session is real-time or interactive. We need to detect and recover from failures *quickly*. Achieving this given the vagaries of wide-area Internet traffic, congestion, and losses is a challenging problem.

Session-recovery in the context of a service-level path could involve a *session-transfer* to an alternate service. For instance, in Figure 1, when there is a problem with the original service-level path, we may decide to use an alternate instance of the transcoder service.

The third issue of making an appropriate choice between distributed replicas based on performance measurements has been well studied in the context of web-mirrors [36, 9]. But the problem is even more challenging in our case just because of the fact that we have multiple composed services in a single session. There are different legs of the path, and multiple choices for each intermediate service. It is a challenging problem to design a mechanism to collect performance information for the different legs at a single place to make the optimal choice for the service-level path.

Finally, a challenge concerning all the three issues is that of **scalability**. We intend our framework to work in the wide-area, and scale to a large number of client sessions. Hence the mechanisms we design for composing services, failure detection and recovery, and gathering performance information for optimizing service-level paths should all be scalable.

1.2 Overall Approach

Our framework for fault-tolerant composed services to address the above challenges consists of the following components (Figure 6 in Section 4):

- An overlay of service clusters deployed at different locations on the Internet. These clusters form the platform for service providers to run their services.
- Peering between pairs of service clusters for:
  - Cascading services across these clusters.
  - Aggregated, active monitoring of the network path between them. Such monitoring is for the purpose of detecting failures in the network path, as well as for measuring other metrics such as latency and available bandwidth.

We have made the design choice of having *active* monitoring for purposes of failure detection. This is appropriate since our goal is to detect failures quickly.

In our service infrastructure, clusters are the unit of construction. This has two advantages. Firstly, we get to leverage cluster redundancy for handling process/machine level failures. That is, we have two-levels of monitoring: one within the cluster, for handling process/machine failures, and one across clusters, for handling network-path failures. We term this *hierarchical monitoring*.

The second advantage of using clusters is that the cost of active network-path monitoring between clusters is now amortized over all sessions that go through a pair of peering clusters.

We address the challenges presented in the previous section by defining a logical overlay network of the service clusters. The logical links in this overlay network are the peering relationships. Service level paths are formed as "paths" in the overlay network. Service providers could either deploy their own service clusters, or deploy their services in a 3rd party provider's cluster.

The overlay network is also used for exchanging different kinds of information. A key idea is to use different levels of information exchange in this overlay network. For instance, network-path liveness information is monitored very closely, but has very little overhead. And metrics such as latency or bandwidth are measured much less frequently, but have higher overhead.

Furthermore, we make the observation that the overlay is a connection-oriented network: service-level paths are constructed on this overlay in a connection-oriented fashion. This allows us exploit ideas such as backup paths or dynamic re-routing for quick recovery.

1.3 Problem scope

The Ninja project defines the notions of a *path* and *operators* (the intermediate services in a path) in a more generic setting. There are several semantic issues in the creation of service-level paths [19]. We do not focus on this aspect of the problem. Also, we assume that the decision of which services to compose for a given functionality is somewhat static. We do not decide this on a per-session, dynamic fashion as in [21].

In our context, we assume that the intermediate services do not have "hard" state in them. That is, sessions can be transferred from one service instance to another without having to worry about the application-level state that was built up at the original instance. We assume that the state can either be discarded altogether, or that it can be

reconstructed from the original source. For example, a 3 second video buffer could probably be discarded without affecting the rest of the session, and a piece of text that is being converted to speech at an intermediate service can be re-retrieved from the original source. We believe that this assumption is valid for a large class of useful applications.

We work under the assumption that there is no end-to-end resource reservation done for the service-level path. This is the way services on the Internet work today – applications are adaptive to congestion. Note that even when applications are adaptive, it is important to address recovery from failures, and optimization of service-level paths.

Finally, in our network-path failure-detection mechanisms, we are concerned with the wide-area Internet, over which the end hosts have little control. Recovery within a local area network, or within an autonomous system is better addressed by solutions such as MPLS [34].

1.4 Overview

The rest of this paper is organized as follows. We present a summary of prior work related to ours in Section 2. We then begin with the question of the feasibility of quick network-path failure-detection over the wide-area in Section 3. We present the design of our framework for cascaded services and session recovery in Section 4. We focus on the problem of routing in the overlay network in Section 4.3. We discuss issues related to the creation of the overlay topology in Section 4.4. Section 5 presents a preliminary evaluation of our architecture. Our plans for the further evaluation and refinement of the architecture are discussed in Section 6. We then summarize our contributions in the last section.

2 Related Work

Research efforts related to our work fall into three main categories: architectures for highly available Internet services, overlay topologies on top of the Internet, and mechanisms for routing around failures in different kinds of networks. We discuss these in order below.

2.1 Fault-tolerant Internet Services

Failure detection and fail-over mechanisms have been considered in several related contexts in the domain of Internet services. Two research efforts closely related to our work are the TACC model and the AS1 model.

The TACC model presents a framework for high-availability of a transcoding proxy service [11]. The solution is based on management within a cluster – service nodes are monitored using keep-alive heartbeats and reinstantiated by the cluster-manager on failure. The cluster-manager itself is monitored by a front-end machine. While elegant in design and capable of handling composed services as well, the model is limited in that it cannot compose services or handle failures outside of the cluster. If the proxy cluster is cut-off from the client, TACC cannot handle the failure. This is not so much of a concern for TACC since the application space considered does not involve long-running client sessions – if the session is of short-duration, network path failures in the wide-area *during* a session are very unlikely.

The Active Services model (AS1) [4] does consider long-running client sessions such as a video transcoding proxy. The AS1 framework also provides a solution for fault-tolerance within a cluster. This is achieved by means of a multicast keep-alive heartbeat from the client to the service cluster. Such a mechanism imposes a restriction on the placement of the service cluster – it has to be close to the client (without this restriction, other issues of the use of multicast address, and wide-area multicast would crop up). Also, like with the TACC framework, AS1 also does not have a mechanism for composition of services across service clusters.

Cluster-based solutions have also been studied for web-server fault-tolerance. For instance, the LARD project [27] explores mechanisms for appropriate choice of web-server machine within the cluster. The SPAND project [36] addresses the problem of making such a choice in the wide-area – appropriate web-mirror selection. However, these mechanisms are appropriate only for short-lived client sessions (like most web-transfers). They do not include mechanisms for detecting failures and effecting recovery *during* a long-lived session.

2.2 Overlay topologies

Routing on overlay networks on the Internet has been considered by several other projects in related contexts. The Tapestry architecture [45] is based on the Plaxton data-structure [29] for locating named objects in the wide-area. Nodes in the overlay network contain objects (services) and also perform the function of routing from one point to another. The Content Addressable Network research project [32] also proposes a very similar mechanism for routing and locating objects in the wide-area. The main advantages of the mechanism are the simple routing data-structures (for scalability) and the fault-tolerance achieved over the overlay network.

The Intentional Naming System [2] also provides routing based on a generic service description (as opposed to an IP-address). It differs from Tapestry in that it uses a more conventional routing protocol (e.g. RIP). The trade-off involved here is that the overhead of routing (in terms of bandwidth and latency) is reduced, but the routing mecha-

nism is less scalable. Also, the INS routing protocols do not focus on quick recovery from failures.

The RON (Resilient Overlay Networks) project proposes an overlay network with a small number of nodes for routing in a fault-tolerant fashion. The routing protocol can use application specific metrics and there could be an overlay network per application.

What all these projects share in common with each other is the idea that an overlay network can be better in providing routing around failures than Internet routing (although none of them have shown this conclusively). We share this idea with these projects.

Our architecture differs in three important aspects from these work. Firstly, our framework includes a mechanism for creation, maintenance, and optimization of composed or cascaded services. This is a first-order goal of our work. Secondly, our overlay network is connection-oriented. Connection-oriented networks are fundamentally better equipped at handling failures quickly since state can be setup in the network in terms of a backup path. This is a crucial observation and an especially important one in the context of real-time interactive services. Finally, the nodes in our overlay network are clusters. This allows us to leverage well-known mechanisms for fault-tolerance and load-balancing within the cluster. We have two levels of monitoring: one within the cluster, for process/machine failures; and one across clusters, for network path failures.

Other overlay topologies worthy of mention are those currently deployed by content providers such as Akamai [40]. These overlay nodes on the Internet help in reaching content (web content, or streaming media) close to the end-user. While little is known about the nature of these overlay configurations, we can say for sure our work differs from these in that we provide a mechanism for cascaded services; while these do not.

The IDMaps project [17] uses an overlay network of measurement nodes on the Internet for the purposes of estimating the "distance" between Internet hosts. While the motivation and application space is quite different from our work, we believe that we can borrow some of their ideas with regard to placement of the overlay nodes on the Internet. For instance, in both cases, it is beneficial to have an overlay node close to the backbone of the Internet.

2.3 Routing and Failure Recovery

We first compare routing in our overlay topology with BGP routing since both operate in the wide-area Internet. Routing in our overlay network is different in that we can use a connection-oriented approach. And even the routing protocol we have for routing the connection-setup messages is different from BGP in that we only need to exchange connectivity information of the overlay nodes. In contrast, BGP has to exchange connectivity information for the end-hosts as well. In our case, we assume a level of indirection for knowing the nearest overlay node for a given end-host (more on this in Section 4.3).

There has been a lot of work in fault-tolerant routing in ATM networks as well as MPLS [34]. Several flavors of fault-tolerant routing have been proposed: on-demand route recovery versus having a pre-established path; end-to-end route recovery versus local recovery, etc [35]. A lot of the work in ATM networks has focussed on the issue of optimal resource allocation with backup virtual paths [20]. While such approaches are in general applicable in our context, there are important differences. Firstly, we do not have a tight control over the topology as many of the related approaches suggest [43]. Secondly, our overlay network has end-to-end paths that may have to go through one or more intermediate services. And finally, because of the effects of aggregation of client sessions across clusters, we can employ several heuristics such as path caching, and incremental path optimization.

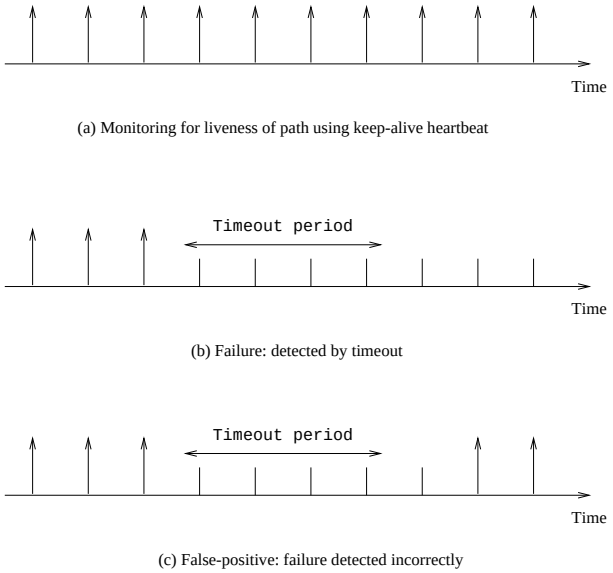
3 Feasibility of Wide-Area Failure Detection: Analysis

Before we design a framework for robust service-level paths in the wide-area, we need to ask ourselves whether at all it is possible to detect failures over the wide-area. The feasibility of quick detection of network-path failures in the wide-area is an important question. Failure detection and recovery in STM networks can be of the order of a few tens of milliseconds [13]. Interactive applications demand such quick recovery. Internet protocols do not have such support for quick recovery. We wish to quickly detect failures at a level above IP, by using an active-monitoring mechanism between two points on the Internet. In this section, we present our study of how quickly and reliably failures can be detected by such a mechanism.

In an active monitoring system, failure detection is done by having a periodic keep-alive heartbeat. The receiving end of the keep-alive messages concludes failure by means of a timeout. This is illustrated in Figure 3. Such a mechanism is natural for a system like the Internet; several other protocols such as RIP [15], BGP [33], SLP [14] use such a mechanism.

With such a timeout mechanism, there is a notion of a *false-positive* in failure detection. This is shown in Figure 3(c). The heartbeat receiver concludes "too soon" that there has been a failure.

In a distributed system, there is a fundamental trade-off between the time-to-detection of failures and the occurrences of false-positives. If the timeout is too small, there could be a large number of false-positives. If the timeout is too huge, there is a long time to detect failure when there



(a) Monitoring for liveness of path using keep-alive heartbeat

(b) Failure: detected by timeout

(c) False-positive: failure detected incorrectly

Figure 3. Monitoring using Keep-Alive Heartbeats: timeouts and false-positives

is actually one. We seek to understand this trade-off in the context of active monitoring in terms of a periodic heartbeat between two hosts on the Internet: how quickly and reliably can network path failures be detected given the vagaries of Internet traffic, losses, and jitter in the latency.

False-positives in a keep-alive stream could be due to (a) simultaneous losses, or (b) sudden increase in RTT (the two or somewhat equivalent from the point of view of the end receiving the heartbeats).

We first look at previous studies that indicate the nature of sudden increases in RTT. In their study of Internet RTT, Acharya and Saltz observe that RTT spikes are isolated in a ping stream [1]: such spikes are undone within a couple of seconds. Along similar lines, Allman and Paxson observe in their study of TCP RTO in [3] that significant RTT increases are quite transient: 86% of bad TCP timeouts are due to one or two elevated RTTs.

We also studied the nature of simultaneous losses in repeated ping measurements between geographically distributed hosts, using ping servers. We counted the number of loss runs with a count of over four. The rate of occurrence of such loss runs was an average of less than once an hour for most host pairs, and of the order of once a day for many pairs of hosts. We were encouraged by these results to perform more detailed experiments. We do not present the details of the ping experiments themselves since the UDP-based heartbeat experiments below represent a keep-alive heartbeat more realistically.

UDP-based Heartbeats

In this set of experiments, we choose geographically distributed hosts in which we had login accounts. We selected pairs of hosts among these and executed a program at either end to send and receive UDP-based keep-alive heartbeats every 300ms. We chose this value for the heart-beat since it represents a relatively low bandwidth usage, and since we expect the jitter in RTT to be 50-100 milliseconds anyway (typical Internet audio tools like VAT [16] use a playout buffer of 80ms).

We measured the gaps between the receipt of successive heartbeats and studied the cumulative distribution of these gaps. We make the following observations from the data. The full set of results is in Appendix A.

- There are a significant number of failures that persist for over 30 seconds. Such failures happen of the order of about once a day. This is significant disruption of service compared to the availability requirements of communication networks [24]. Hence it is important to deal with such failures.
- Although there is a good amount of variability across different host-pairs, in many of the cases, the cumulative distribution has a knee point around 0.9-1.5 seconds. That is, losses that last for 1.5 seconds actually extend to longer loss periods in most cases. This means that we can conclude a “failure” with a timeout of around 1.5 seconds. Specifically, we studied the case where we concluded failure (a loss period of over 30 seconds) after a timeout of just 2 seconds. We observed that the false-positive rate could be as low as 50% between well connected pairs of hosts. If we have a mechanism for session-transfer based on such a failure-detection scheme, we can potentially workaround failures in that much time. Note that there need not be any additional end-to-end loss in the data stream because of a false-positive.

  While this value of two seconds may not be good enough for interactive applications such as two-way telephony, it is at least an order of magnitude better than what is possible today with Internet route recovery – which could take anywhere from 30 seconds to more than ten minutes [23]. And this value of 2 seconds is definitely tolerable for buffered on-demand streaming applications that have buffer-data – typically these have 5-10 seconds worth of buffered data.

4 Design

Encouraged by the results from our analysis above, we now design the rest of the framework for deciding who monitors which portion of the network path, how service-level paths are constructed, and how recovery is effected.

Before we present our architecture, we briefly discuss the alternatives we considered, and rejected for high availability of long-lived sessions.

4.1 Design alternatives

Given the good trade-off between time-to-detection and occurrence of false-positives in failure detection, one might be tempted to take an approach of end-to-end monitoring for robustness. This is shown in Figure 4. This is only approach feasible today and does not require any infrastructure support. On detecting a failure, one can imagine the client choosing an alternate source for the stream, possibly from a pre-determined list of such servers.

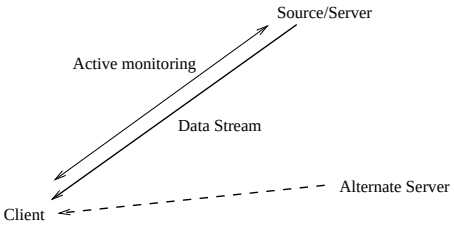


Figure 4. An End-to-End Approach to Monitoring

There are several problems with this approach. What is immediately obvious is that with such end-to-end monitoring, the overhead of active monitoring could be significant. This is especially so when the data stream is low bit-rate. For instance, the G.723 codec can operate at 5.3kbps; the effective rate is half of this if there is only one person talking during a two-way conversation.

The problem of overhead itself may not be enough reason for choosing an alternative approach. For instance, one can use in-band monitoring in the data stream. But there are several other issues with such a naive approach. Firstly, there is no framework for service composition and the only way services can be composed is hop-by-hop. The overall service-level path could be very sub-optimal. Secondly, when a network path failure is detected with the original server, the alternate server itself could be cut-off from the client, probably due to the same failure. Hence it becomes necessary to keep track of which service replicas are unreachable. Finally, there is a problem when the source end-point is fixed – this is the case with IP-telephony, or other live multimedia streams. There is no notion of a service replica in this case.

We also considered an approach where monitoring is aggregated at the client side, much like in SPAND [36], but with active monitoring for network path liveness. This is depicted in Figure 5. However, this approach addresses only one of the issues mentioned above. That is, it solves the problem of excessive overhead of monitoring. The other problems persist: there is still no framework for service composition. The alternative server could be unreachable, and many alternate instances may have to be monitored. And there is no mechanism to handle cases where the source end-point of the data stream is fixed.

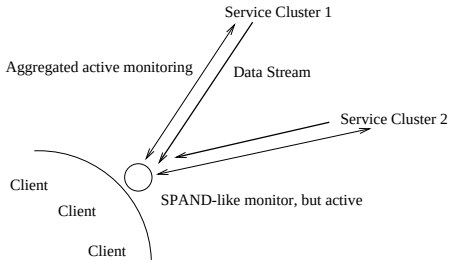


Figure 5. A SPAND-like approach: client-side aggregation

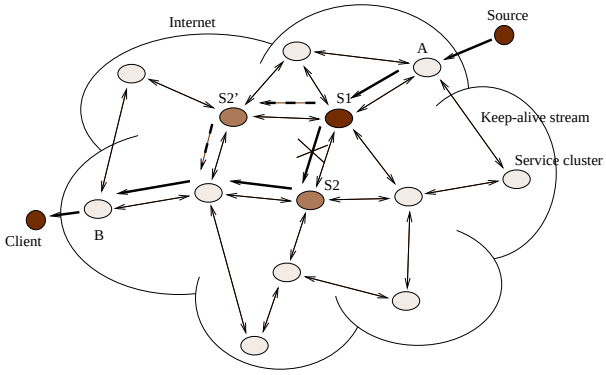


Figure 6. Architecture: Overlay Network of Service clusters

4.2 Our Architecture

Motivated by the notion of a service-level path, we think in terms of a service-level overlay network. Nodes in this network implement services, and a service-level path is constructed as a "path" in this overlay network. This is shown in Figure 6. At an intuitive level, the overlay network nodes exchange information to enable composition of services, optimization of such composed service-level paths, and recovery on detecting failures (dotted lines in the figure).

The overlay network constitutes a middleware service infrastructure on the Internet. An important feature of the overlay is that the nodes are service clusters. They form the compute platforms on which services are deployed. In

this model, pairs of service clusters *peer* with each other. This is shown in terms of arrows between the nodes in Figure 6. Such a peering arrangement implies a keep-alive, active monitoring stream between the clusters for detecting network path failures between them. This peering constitutes an edge, or a link, in the overlay network.

A service-level path between such a source and destination is formed by going through a set of nodes in the overlay network. The figure shows a service-level path consisting of two services (S1 and S2) between a source and a client. In general, there may be "null" services in the service-level path – these represent simple forwarding of the data stream. In the figure, the nodes other than S1 and S2 on the path of the data stream represent null services.

There are several advantages of the architecture that address our overall goals. The use of clusters as nodes in the overlay has many advantages. Firstly, the monitoring overhead gets amortized across multiple end-to-end client sessions along the common legs of the overlay. Although there may be several end-to-end sessions, there are only as many keep-alive streams as edges in the overlay network. Such amortization of overhead allows us the freedom of aggressive monitoring without worrying about the overhead.

Secondly, the use of clusters for nodes in the overlay network means that machine or process level failures of services can be handled within a cluster. This allows us to think of the overlay nodes as resilient to failures. That is, we can have two levels of monitoring: one within the cluster, and one across clusters. With such a *hierarchical monitoring* approach, machine/process failures are handled within a cluster, and network path failures are detected and handled by the monitoring mechanism across clusters.

Conceptually, an end-to-end path is routed along monitored portions of the network, the legs of the service-level path are automatically monitored. However the portion of the network between the source and the *entry point (A)*, or the *exit point (B)* and the destination are not monitored. The intention is to deploy overlay nodes in such a fashion that these legs are short. For instance, we could have an overlay node "close" to the Address Prefix (AP) to which the end-hosts belong. We return to this problem of overlay node placement later in Section 4.4.

The main feature of the architecture is the context it provides for addressing the issues of optimal construction of service-level paths, and the mechanism for quick session-recovery on failure detection – we address both these issues as routing in the overlay network. We turn to discuss this now.

4.3 Routing on the Overlay Network

Consider the example in Figure 6 where a data stream is routed from source to destination via the overlay. To provide a complete solution, we need to address the following issues:

- **I1** Find an entry point into the overlay network, and an exit point (marked A and B in Figure 6).
- **I2** Find a route in the overlay from the entry point to the exit point through links that are currently active; going through intermediate services, if any. Such a service-level path should be optimal.
- **I3** Provide a mechanism for recovery from failure during a session.

A slight variation of this case is when the source is not "fixed". That is, there could be several replicas of the source. For instance, an on-demand server could have several mirrors. These replicas could be in our overlay service clusters, or could be outside of it, in which case there are multiple points of entry. In either case, the creation and optimization of the path is handled like in other cases (I2). For the discussion below, we assume that the source is fixed.

We address each of the issues I1, I2, and I3 now.

4.3.1 Finding a point of entry and exit

We simplify this problem by making the following assumption: the choice of nearest overlay node is relatively static, compared to the dynamicity of routes within the topology. This is reasonable for the following reasons:

- Since the overlay nodes are clusters, there is no question of the overlay node failing (or is at least very improbable).
- We could place overlay nodes such that there is an obvious choice of the closest overlay node. As mentioned earlier, we could place the overlay node close to the connection point of the end-point's Address Prefix to the rest of the Internet. In such a case, if the network between the end-host and the overlay node fails, with high probability, the end-host is also cut-off from any other overlay node. Thus the choice of closest overlay node is more or less static.

The end-host (source or destination) could learn of the closest overlay node before-hand. This could just be preconfigured or could be learned using an expanding scope search.

4.3.2 Routing from entry to exit

This issue is one of deciding which intermediate nodes to use for the service-level path. When there are many choices for the intermediate services, we need to make an optimal choice.

We have a routing algorithm on the overlay network – this was one of the reasons for defining the overlay network. A routing algorithm in general exchanges information about two things: it conveys reachability information; and it optimizes paths or routes according to some metric. (There could also be routing policies, which we ignore for the moment). We are faced with a challenging problem since we may have to route through intermediate services, each of which may have several replicas. We need to make an optimal choice based on criteria such as network path liveness, metrics such as latency/bandwidth, and service location information.

Our key idea to address this is as follows. We have different levels of information exchange on the overlay:

- **L1** Network path liveness information
- **L2** Metric information such as latency or bandwidth
- **L3** Information about location of services at the different service clusters

Network path liveness information is exchanged between peering nodes – this is the keep-alive stream. This is at a very fine granularity (once every 300ms in our experiments in Section 3). But it is of very low bandwidth. Also, this information is propagated to the rest of the network only when there is a change of liveness status.

Latency or bandwidth information is measured on a much coarser time granularity, of the order of once in several minutes. We do not need to track changes in these metrics very closely, especially if the end application is adaptive (Real-Audio can tolerate 10% packet losses quite easily, and sometimes even upto 50% [26]). Also, this information does not change very frequently. In the studies in [5], it is observed that significant changes in available bandwidth occur of the order of once in several minutes. And in the RTT studies in [1], it is noted that typically significant changes in RTT occur of the order of once an hour.

Information exchanged in the third level L3 is at an even larger granularity – only when services are deployed or taken out of service. This information exchange is for the purpose of knowing which services are at which service clusters. This information could be bulky, and is exchanged at the granularity of once in a few weeks or even months.

We could also make use of an external wide-area service discovery mechanism such as [7], if one exists. We leave this choice open at this stage, and for the rest of the discussion assume that there is a way of knowing the location of the replicas of a given service.

Before we describe how all of these work together, we make another important observation. The overlay network is a connection-oriented network. That is, there is an explicit creation phase for the service-level path; and the intermediate nodes can, and do have "switching" state per session. This has important implications on the session-recovery – we come back to this in Section 4.3.3.

Having observed that what we have is a connection-oriented network, we now describe how connection setup is done. That is, we describe how the connection setup messages are routed. Note that this routing of connection setup messages itself has to be connection-less.

From the exchange of reachability information in the routing protocol, each node has information about the overlay topology. We use a simple link-state algorithm for arriving at the overlay topology at each node. This topology information also includes approximate metric information of the links in the overlay topology. We have also assumed that each node has the information on where the different service replicas are, or can get it on-demand.

Now, the problem of finding an optimal service-level path through the required intermediate services can be solved locally, within a node. The entry node finds such a service-level path. It then sends the connection-setup messages in a source-routed fashion to the exit node, to setup the session.

Path caching and Dynamic Path Optimization

In the description of the session setup process above, we mentioned that the entry node finds the optimal path. Although the problem is greatly reduced because we have all the information at a single node, this could involve significant computation, especially if each of the intermediate services have several replicas – we have to search through all the possible service-level paths.

Once again, we leverage the aggregation properties of the cluster nodes. We can cache paths so that future computations are more efficient. This is similar to the idea of using past information to choose a web-mirror, in SPAND [36].

Another idea that we can use to reduce the computations to find an optimal path is that of dynamic optimization. Since session-transfer is a first-order feature of our framework, the initial path we choose need not be optimal. We can transfer the session to an alternate path of better performance after session setup. Note that this process, unlike session-recovery during a failure, need not involve losses in the data stream – since we can setup the alternate path before we perform the session transfer.

4.3.3 Session recovery on failure

We now turn to addressing the issue of session recovery. In connection-less networks, an end-to-end path recovers when the failure information propagates through the network. Connection-oriented networks are inherently better at handling failures since the failure information need not propagate to all the nodes before the end-to-end path is rerouted. Appropriate switching state can be setup along an alternate path.

Connection oriented networks have different flavors of recovery from failures [35]. Broadly, these fall under:

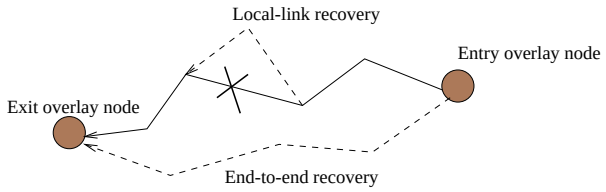


Figure 7. Recovery Mechanisms in Connection-Oriented Networks

- Edge-to-edge recovery versus local-link recovery (See Figure 7).
- Pre-established backup versus on-demand recovery.

Local-link recovery typically implies on-demand recovery. While we have not fully explored the trade-offs between these approaches, we are inclined towards a local-link recovery since the overlay links are what we monitor closely. Edge-to-edge recovery means that the failure information has to propagate to one end of the path – this could involve multiple wide-area hops.

With a local-link recovery scheme, the upstream node of the failed link tries to find an alternate route through the overlay to the node downstream of the failed link (the dotted lines on the top in Figure 7). We evaluate the additional latency overhead with such an approach in Section 5.

4.4 The Overlay Topology

So far, we have not addressed the issue of how the overlay topology is formed and how it maps onto the physical topology. We discuss these issues now.

In our framework, we need to address the questions of:

- How many overlay nodes are deployed
- Where they are deployed
- How they peer with each other

These are somewhat long-term decisions compared to the routing and recovery mechanisms discussed earlier. There are several factors affecting these:

- Since the portion of the network path between the source and the entry point, or between the exit point and the destination is not monitored actively in our framework, we need to have overlay nodes "close" to end-hosts.
- In an overlay topology, the logical links may share the same physical links. We would like such sharing to be minimal. This is because such sharing means bandwidth overhead on the physical link – when a path is routed twice on the same physical link. This means that in general, it is good to have the service clusters placed close to the backbone of the Internet – where paths to different other parts of the Internet are less likely to share physical links.

  This factor of physical link sharing would also go into deciding the peers of a service cluster.
- Another factor deciding placement and peering between clusters is the nature of connectivity between them. If they are too far apart over the wide-area, there may be too many false-positives and failures (refer the bad cases in Table 1).

The issue of the number of overlay nodes is very important and has direct implications on the scalability and stability of the information exchanged on the overlay. It is clear that the overlay is a much smaller network than the Internet. Since we are concerned with failures in the wide-area, intuitively, it is enough to place overlay nodes at points near the Internet backbone, to take advantage of the redundancy in the Internet's topology. However, we also need to have overlay nodes "close" to end-hosts. The granularity of this "closeness" is an open question which we plan to address in our work.

5 Evaluation

The main advantage of our framework is the provision for quickly detecting and routing around failures, for a service-level path. In Section 3, we presented a study of how quickly failures can be detected based on measurements in the wide-area. In this section, we evaluate some of the other aspects of the architecture. We present the main results from our evaluation below. The details of the experiments and the complete results are in Appendix B.

Routing overhead in the overlay: One of the concerns with an overlay network is the additional overhead it introduces in terms of end-to-end latency. To get an idea of this overhead, we model the network using generated topologies (TIERS, Transit-Stub) as well as real ones (AS-Jan2000, MBone). The graphs have between 4000 and 6500 nodes. We randomly choose a set of nodes in the graph to be the overlay nodes. We form peering relationships between overlay nodes, giving preference to pairs that are closer to one another. In this process, we impose the constraint that no physical link is shared by two overlay links. Because of this, sometimes we end up with a disconnected overlay graph.

Once we generate the overlay topology, we choose 1000 random pairs of nodes in the original graph. We select their nearest overlay nodes, and find a route through the overlay links from entry to exit. We compare the overhead of such routing over direct routing between the two nodes.

We observe that the percentage of end-host pairs for which the routing overhead is above 50% is very less in all the cases. For the Transit-Stub graph, only 2.1% of the end-host pairs have over 5% routing overhead.

Routing overhead after local-link recovery: We now study the additional overhead when we use a local-link recovery strategy. With such a strategy, on detecting a failure, the upstream overlay node of the failed link finds an alternate path to the downstream node. This was illustrated in Figure 7. To simulate a link failure, we simply remove a link from the overlay graph, and recompute the path between the two-ends of the failed link.

The overhead of such re-routing is definitely higher than just routing on the overlay network. For instance, in the case of Transit-Stub topology, 6.1% of the end-host pairs experience an overhead of over 25%.

Estimate of recovery time: In the case of local-link recovery, as shown in Figure 7, the recovery involves sending connection setup messages to setup appropriate state on the alternate path between the ends of the failed link. The network delay cost of this process can be estimated as the propagation delay along this alternate path. We measure this as a fraction of longest distance between any two overlay nodes in the graph (diameter of the overlay graph).

The 95th percentile of this fraction for the Transit-Stub graph is 0.5. That is, in 95% of the cases, the latency to recovery is less than half of the diameter of the graph.

We should note that this is only a rough estimate and does not account for variability in Internet link propagation times [10] or for behavior under load.

Effect of the size of the overlay: We compare the routing overhead with different numbers of overlay nodes: 50, 100, and 200. We observe that, the number of overlay nodes has a definite effect on the routing overhead. The denser the overlay topology, the lesser the routing overhead. For example, for the Transit-Stub graph, with 100 overlay nodes, 57.5% of the node-pairs had a routing overhead of over 5%, in comparison with just 2.1% node-pairs for 200 overlay nodes.

Time to establish "connection" state: We study this for our implementation of the $GSM \leftrightarrow PCM$ codec service. This is a codec that has widespread hardware based implementations deployed at the Inter-Working Function between the PSTN and the GSM networks. This is important for Internet-based integration of services between these networks [38]. We measure the time to setup a new session at this service, as a function of the number of existing sessions. We see that the session setup latency is under 50ms independent of the load – this particular codec is not very CPU intensive.

6 Research Methodology and Plan

There are several metrics that we intend to use to evaluate the architecture:

- **Overhead:** There are two overheads associated with the architecture: the addition to end-to-end latency because of routing over the overlay network, and the bandwidth overhead of the information exchanged in the overlay network.
- **Latency to recovery:** This is a measure of the effectiveness of the architecture.
- **Use of composability:** This refers to the use of cascaded services in building application functionality. This is a measure of effectiveness at the application level.
- **Scalability:** Since our architecture is for the wide-area Internet, it is important to study its scalability to a huge number of client sessions.
- **Stability:** We make decisions on the optimal service-level path based on information exchanged in the overlay network. The stability of this information needs to be analyzed. Furthermore, we should also ensure that dynamic optimization does not lead to oscillations.

We plan to study these metrics through a combination of simulations, trace-measurements in the wide-area, and real implementation. For initial estimates of the overhead, a simulation-based approach is appropriate (Section 5). To study metrics such as the bandwidth overhead of the architecture, and its stability, it is appropriate to use a combination of traces collected over the wide-area, and simulation. This would allow us to capture a wide range of the Internet's heterogeneity. Such a method is also appropriate for studying the issues related to the topology of the overlay.

To analyze scalability, it is more appropriate to use a real testbed implementation. We are currently collaborating with TU-Berlin and UNSW for a wide-area testbed. We are also exploring possibilities for extending this to Uppsala University, Sweden, Stanford University, and some industrial locations as well. This testbed will be maintained in the context of the ICEBERG project [38].

We also intend to use real services to examine the end-to-end effects of session-transfer and session-recovery. We have already developed a number of such composable services in the context of the ICEBERG project's Universal Inbox [30] for communication between heterogeneous devices. A canonical example of this the composition of an

MP3 streaming service, an MP3 to PCM transcoder, and a PCM to GSM transcoder to enable music on a GSM cellular phone. We have this functionality working in our ICEBERG testbed at Berkeley [30]. Implementation of real services would provide us the necessary feedback to refine the architecture.

Overall, our approach would be based on a cycle of **analysis**, **design**, and **evaluation**. In our work so far, we have completed the initial analysis, design, and preliminary evaluation (Figure 8). We now summarize our future plans in three phases of 6 months each.

Phase I (0-6 months)

- Detailed analysis of
  - latency and bandwidth overhead
  - latency to recovery
- Use traces of latency/bandwidth over wide-area
- Develop real implementation in parallel
  - this is already in progress (we have implemented some of the mechanisms for session state setup and transfer)
  - this will give feedback for the analysis above

Phase II (6-12 months)

- Use the implementation from Phase I
  - deploy real services on the wide-area testbed
  - analyze end-to-end effects of session-recovery
  - examine scalability
- Use traces from Phase I to analyze stability of optimality decisions
  - collect more traces of latency/bandwidth

Phase III (12-18 months)

- Use feedback from deployment of real services to refine the architecture
- Analyze placement strategies
  - use wide-area measurements and traces from phases I and II

Appropriate conferences and workshops for presenting our work include NOSSDAV, ACM MultiMedia, SOSP, INFOCOM, and SIGCOMM. We will plan on submission to these and other conferences depending on the submission deadlines and the status of our work.

7 Summary and Conclusions

We started with the goal of providing a framework for quick recovery from failures in the context of composed services. As a preliminary evaluation of the feasibility, we performed experiments on the wide-area Internet to understand the trade-off involved in quick failure detection. We have designed an architecture based on a connection-oriented overlay network of service clusters. Quick recovery is achieved by means of providing dynamic or pre-constructed backup paths in the overlay network. We have a preliminary evaluation of the architecture. We plan to further evaluate and refine our architecture based on simulations, trace collection, and a real implementation of composed services on a wide-area testbed.

Our main contribution is the framework for composing services across service providers and across the wide-area Internet. This allows rapid development and deployment of new functionality on 3G+ devices.

We have presented the notion of a connection-oriented service network on top of a connection-less network. Connection-oriented networks are better in terms of management of end-to-end sessions – they allow construction of backup paths, dynamic or pre-constructed, for fast failover. While this has been explored in detail at the network and link layers, this is yet unexplored in the service or middleware layer.

We have intentionally chosen an application space that is rather forgiving in terms of the ease of session transfer, and in terms of the end-to-end behavior during such a transfer. Although this does not include all possible applications, it covers a good range of important and useful applications.

References

[1] A. Acharya and J. Saltz. A Study of Internet Round-Trip Delay. Technical Report CS-TR 3736, UMIACS-TR 96-97, University of Maryland, College Park, 1996-97.

[2] W. Adjie-Winoto and et.al. The Design and Implementation of an Intentional Naming System. In *Proc. 17th ACM SOSP*, Jan 2000.

[3] M. Allman and V. Paxson. On Estimating End-to-End Network Path Properties. In *ACM SIGCOMM'99*, Oct 1999.

[4] E. Amir. *An Agent Based Approach to Real-Time Multimedia Transmission over Heterogeneous Environments*. PhD thesis, U.C.Berkeley, 1998.

[5] H. Balakrishnan and et.al. Analyzing Stability in Wide-Area Network Performance. In *ACM SIGMETRICS Conference on Measurement & Modeling of Computer Systems*, Jun 1997.

[6] K. L. Calvert, M. B. Doar, and E. W. Zegura. Modeling Internet Topology. *IEEE Personal Communication Magazine*, Jun 1997.

[7] S. Czerwinski and et.al. An Architecture for a Secure Service Discovery Service. In *MobiCom*, Aug 1999.

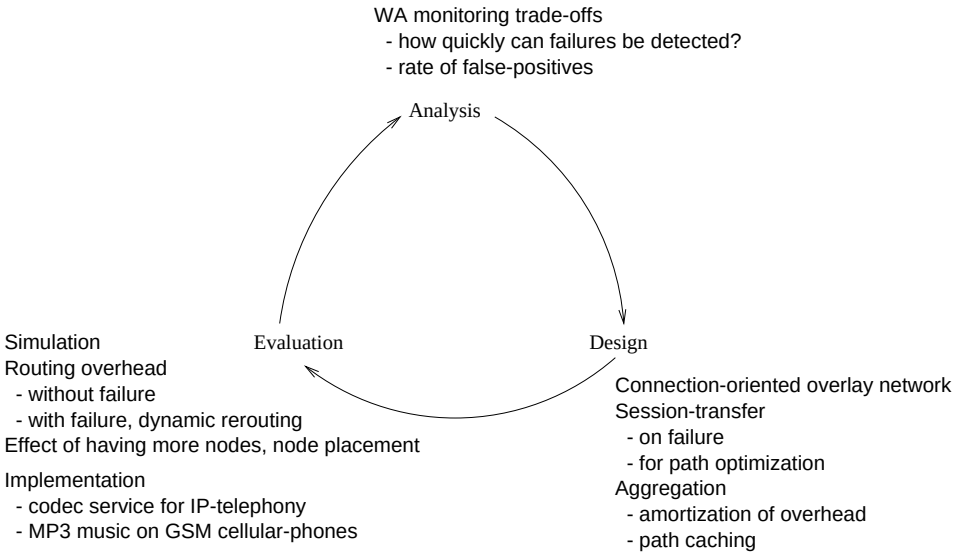


Figure 8. Research methodology (and current status)

[8] M. B. Doar. A Better Model for Generating Test Networks. In *Globecom'96*, Nov 1996.

[9] S. G. Dykes, C. L. Jeffery, and K. A. Robbins. An Empirical Evaluation of Client-side Server Selection Algorithms. In *IEEE Infocom*, Mar 2000.

[10] M. Faloutsos, P. Faloutsos, and C. Faloutsos. On Power-Law Relationships of the Internet Topology. In *ACM SIGCOMM'99*, Aug 1999.

[11] A. Fox. *A Framework for Separating Server Scalability and Availability from Internet Application Functionality*. PhD thesis, U.C.Berkeley, 1998.

[12] C. Gbaguidi and et.al. A Programmable Architecture for the Provision of Hybrid Services. *IEEE Communications Magazine*, Jul 1999.

[13] W. D. Grover and et.al. Performance studies of a selfhealing network protocol in Telecom Canada long haul networks. *GLOBECOM '90*, 1990.

[14] E. Guttman and et.al. *Service Location Protocol, Version 2, Request for Comments: 2608*, June 1999.

[15] C. Hedrick. *Routing Information Protocol, Request for Comments: 1058*, Jun 1998.

[16] V. Jacobson and S. McCanne. VAT Mbone Audio Conferencing Software. ftp://ftp.ee.lbl.gov/conferencing/vat.

[17] S. Jamin and et.al. On the Placement of Internet Instrumentation. In *IEEE INFOCOM'00*, Mar 2000.

[18] A. Joseph, B. R. Badrinath, and R. H. Katz. A Case for Services over Cascaded Networks. In *First ACM/IEEE International Conference on Wireless and Mobile Multimedia*, Oct 1998.

[19] A. D. Joseph and et.al. System Support for Multi-Modal Information Access and Device Control. In *Work-in-Progress, WMCSA'99*, Feb 1999.

[20] R. Kawamura and I. Tokizawa. Self-Healing ATM Networks Based on Virtual Path Concept. *IEEE Journal on Selected Areas in Communication*, Jan 1994.

[21] E. Kiciman and A. Fox. Using Dynamic Mediation to Integrate COTS Entities in a Ubiquitous Computing Environment. In *Second International Symposium on Handheld and Ubiquitous Computing*, 2000.

[22] B. Kreller and et.al. UMTS: A Middleware Architecture and Mobile API Approach. *IEEE Personal Communications Magazine*, Apr 1998.

[23] C. Labovitz and et.al. An Experimental Study of Delayed Internet Routing Convergence. In *Computer Communication Review, ACM SIGCOMM'00*, Aug/Sep 2000.

[24] A. R. Modarressi and R. A. Skoog. Signaling System No. 7: A Tutorial. *IEEE Personal Communications Magazine*, Jul 1990.

[25] W. Mohr and W. Konhauser. Access Network Evolution Beyond Third Generation Mobile Communications. *IEEE Communications Magazine*, Dec 2000.

[26] C. Overton. Berkeley Multimedia, Interfaces, and Graphics Seminar, and Personal Communication, Jan 2001.

[27] V. S. Pai and et.al. Locality-Aware Request Distribution in Cluster-Based Network Servers. In *Eighth Symposium on Architectural Support for Programming Languages and Operating Systems*, Oct 1998.

[28] R. Pandya and et.al. IMT-2000 standards: network aspects. *IEEE Personal Communications Magazine*, Aug 1997.

[29] C. Plaxton, R. Rajaraman, and A. W. Richa. Accessing Nearby Copies of Replicated Objects in a Distributed Environment. In *Ninth Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA)*, Jun 1997.

[30] B. Raman, R. H. Katz, and A. D. Joseph. Universal Inbox: Providing Extensible Personal Mobility and Service Mobility in an Integrated Communication Network. In *Workshop on Mobile Computing Systems and Applications (WMSCA'00)*, Dec 2000.

[31] J. Rapeli. UMTS: targets, system concept, and standardization in a global framework. *IEEE Personal Communications Magazine*, Feb 1995.

[32] S. Ratnasamy and et.al. A Scalable Content Addressable Network. Technical Report tr-00-010, International Computer Science Institute, Oct 2000.
[33] Y. Rekhter and T. Li. *A Border Gateway Protocol 4 (BGP-4), Request for Comments: 1771*, Mar 1995.
[34] E. Rosen, A. Viswanathan, and R. Callon. *Multiprotocol Label Switching, Internet Draft draft-ietf-mpls-arch-07.txt*, Jul 2000.
[35] V. Sharma and et.al. *Framework for MPLS-based Recovery, Internet Draft draft-ietf-mpls-recovery-frmwrk-01.txt*, Nov 2000.
[36] M. Stemm. SPAND: Shared Passive Network Performance Discovery. In *1st Usenix Symposium on Internet Technologies and Systems (USITS '97)*, Dec 1997.
[37] G. Vanecek. Enabling Hybrid Services in Emerging Data Networks. *IEEE Communications Magazine*, Jul 1999.
[38] H. J. Wang, B. Raman, and et.al. ICEBERG: An Internet-core Network Architecture for Integrated Communications. *IEEE Personal Communications Magazine*, Aug 2000.
[39] J. Williams. The Hard Road Ahead for WAP. *IT Professional Magazine*, Sep/Oct 2000.
[40] WWW. Akamai: Delivering a Better Internet. http://www.akamai.com/.
[41] WWW. Modeling Topology of Large Internetworks. http://www.cc.gatech.edu/fac/Ellen.Zegura/graphs.html.
[42] WWW. National Laboratory for Applied Network Research. http://www.nlanr.net/.
[43] Y. Xiong. Optimal Design of Restorable ATM Mesh Networks. In *IEEE ATM Workshop*, 1998.
[44] E. Zegura, K. Calvert, and S. Bhattacharjee. How to Model and Internetwork. In *IEEE INFOCOM'96*, Apr 1996.
[45] B. Zhao. Tapestry: The Oceanstore Regenerative Wide-area Location Mechanism. http://www.cs.berkeley.edu/ ravenben/research/plaxton6_2000/, 2000.

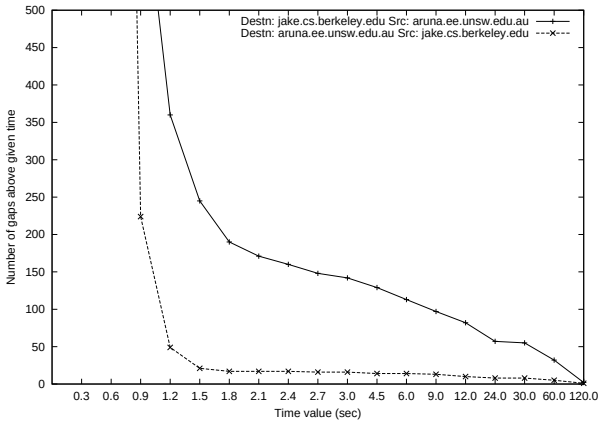


Figure 9. Gap distribution: Berkeley-UNSW

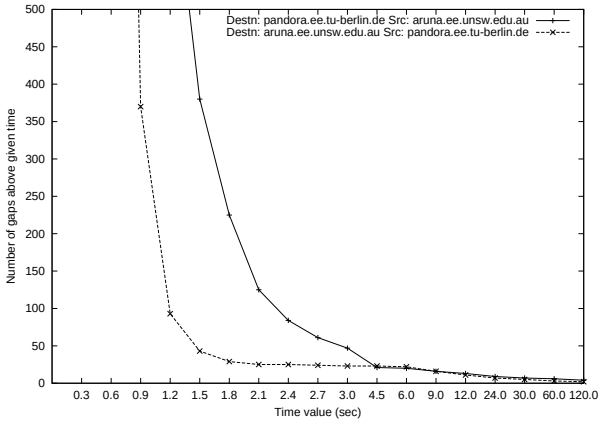


Figure 10. Gap distribution: UNSW-TUBerlin

A UDP-based heartbeats: Detailed results

In this section, we present the details of our UDP-based heartbeat experiments. For each pair of hosts, we measured the gaps between the receipt of successive heartbeats. Figures 9-14 show the cumulative distribution of these gaps. We have shown six separate graphs for clarity. Each of the graphs have two lines – representing behavior in either direction for a pair of hosts. For each time-value on the x-axis, we show the number of gaps in the keep-alive stream above that value in the y-axis. This y-axis value represents the number of timeouts that would have happened if the given time-value in the x-axis were chosen as the timeout for concluding a failure. Note that the x-axis is not linear.

Table 1 shows a hypothetical case where we define the notion of failure to be when we do not receive any heartbeat for 30 seconds. And we have a timeout period of 2 seconds for concluding failure. Hence, we define a false-positive to have occurred when we see no heartbeat for 2 seconds (and hence conclude failure), but see one before 30 seconds (an actual failure had not occurred).

In the table, the first and second columns give the destination and source of the heartbeats, respectively. The third column gives the total amount of time for which we ran the experiment. The fourth and fifth columns give the count of false-positives and the actual number of failures, according to our definition above.

The first six rows represent very widely separated host-pairs – with at least one trans-oceanic link between them. The last six rows represent host-pairs within the US.

In all of the graphs, we see a huge number of gaps that are over 600ms. This means that there are a lot of single and double packet losses in the heartbeat stream. While this is not surprising, it is interesting to note that there is a knee point around 0.9-1.5 seconds for many of the graphs after which the graph more or less flattens. Such a flat region means that when there is a loss period of about 1.5 seconds, we can conclude with high probability that the loss period is going to persist for a long time – there has been a failure. Table 1 shows this in more concrete terms with the number of false-positives and the actual number of failures. When

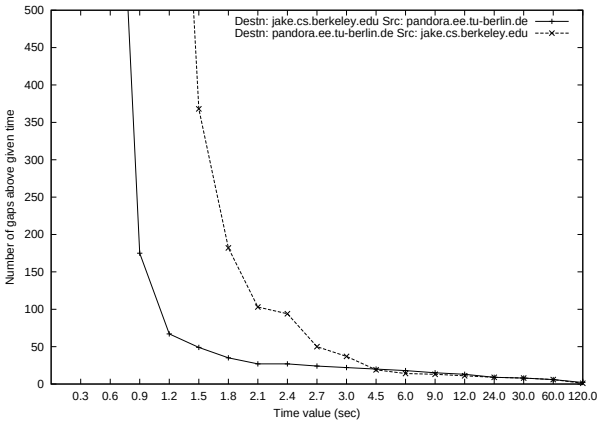


Figure 11. Gap distribution: TUBerlin-Berkeley

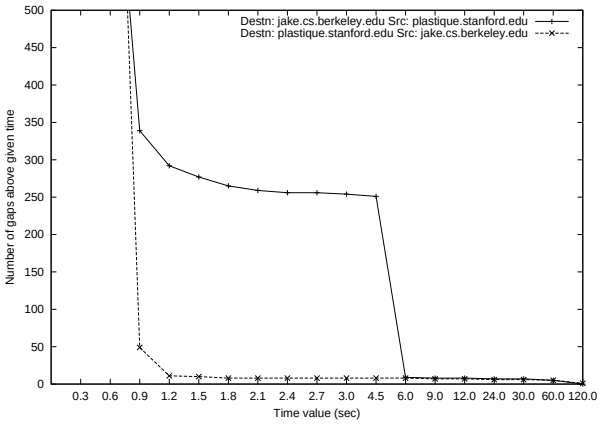


Figure 12. Gap distribution: Berkeley-Stanford

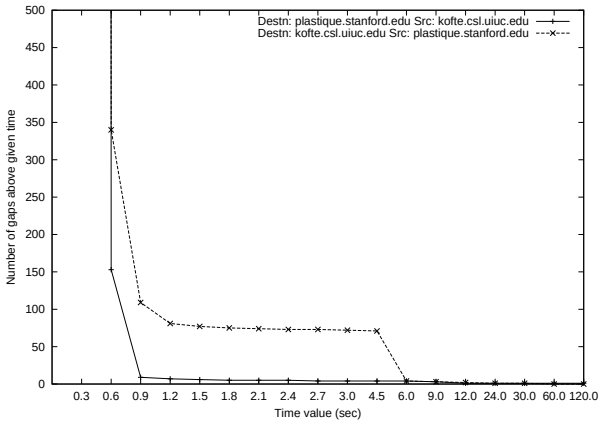


Figure 13. Gap distribution: Stanford-UIUC

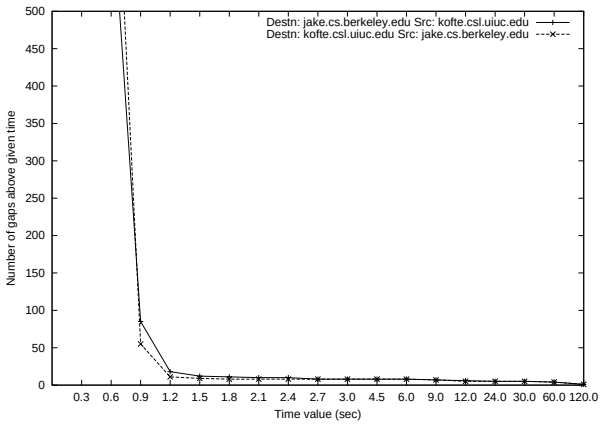


Figure 14. Gap distribution: UIUC-Berkeley

the number of false positives is comparable to the number of actual failures, and both are low, this represents a good case. This is the case for the four rows with (Destn←Src): Stanford←Berkeley, Stanford←UIUC, Berkeley←UIUC, and UIUC←Berkeley. Even for the really wide-area cases UNSW←Berkeley, and Berkeley←UIUC, the numbers represent quite good cases.

The other pairs separated over trans-oceanic links represent very bad cases – which is understandable. For the two cases where Stanford was the origin of the heartbeats, there are a lot of false-positives. Closer examination of the graphs in Figures 12 and 13 shows that there is a huge drop in the number of timeouts as we go from 4.5 seconds to 6 seconds. That is, there are a lot of failures that persist for 4.5-6 seconds. We have not examined this any further, but it looks like a problem with a router or link along the network paths in these two cases. We consider this to be an extraneous case.

B Evaluation: Detailed results

In this section, we present the details of the results we summarized in Section 5.

B.1 Routing overhead in the overlay

To get an idea of the overhead on the end-to-end latency because of routing through the overlay network, we perform the following measurements. To model the network, we use two generated topologies (TIERS, Transit-Stub), and two real ones (AS-Jan2000, MBone). The artificially generated topologies roughly model the Internet's hierarchical structure [6]. The TIERS topology has 5000 nodes, and was generated using the TIERS generator [8]. The Transit-Stub topology had a total of 6510 nodes, with 14 transit ASs, each with 15 nodes, 10 stub-ASs per transit-node, and 3

HB destn	HB src	Total time	Num. False Positives	Num. Failures
Berkeley	UNSW	130:48:45	135	55
UNSW	Berkeley	130:51:45	9	8
Berkeley	TU-Berlin	130:49:46	27	8
TU-Berlin	Berkeley	130:50:11	174	8
TU-Berlin	UNSW	130:48:11	218	7
UNSW	TU-Berlin	130:46:38	24	5
Berkeley	Stanford	124:21:55	258	7
Stanford	Berkeley	124:21:19	2	6
Stanford	UIUC	89:53:17	4	1
UIUC	Stanford	76:39:10	74	1
Berkeley	UIUC	89:54:11	6	5
UIUC	Berkeley	76:39:40	3	5

Table 1. UDP-based heartbeats

nodes per stub-AS [44]. This topology was generated using the GT-ITM package [41]. The AS-Jan2000 topology models the connectivity between ASs, and has 6474 nodes. It was generated by the National Laboratory for Applied Network Research [42] using BGP tables. The MBone graph was collected the SCAN project at USC/ISI in 1999, and each node represents an MBone router, with a total of 4179 nodes.

We make the overhead estimates when there are no intermediate services – so that the effect of placement of service replicas is excluded. We also exclude the effects of policy-based routing on the Internet and simply assume shortest distance routing. We randomly choose a set of nodes in the graph to be the overlay nodes. We examine pairs of overlay nodes in the order of their closeness and decide to form peering relations between these. In this process, we impose the constraint that no physical link is shared by two overlay links. Because of this, sometimes we end up with a disconnected overlay graph.

Once we generate the overlay topology, we choose 1000 random pairs of nodes in the original graph. We select their nearest overlay nodes, and find a route through the overlay links from entry to exit. We compare the overhead of such routing over direct routing between the two nodes. Figure 15 shows the cumulative distribution of this overhead for the different graphs. In each of the cases, we used 200 overlay nodes.

In all of the graphs, the cumulative percentage for the routing overhead factor of 1.0 is slightly less than 100% – this is because some end-host pairs are not reachable from one another through the overlay network (since the overlay network ends up being disconnected, as mentioned earlier).

We see that the percentage of end-host pairs for which the routing overhead is above 50% is very less in all the cases. For the Transit-Stub graph, only 2.1% of the end-host pairs have over 5% routing overhead.

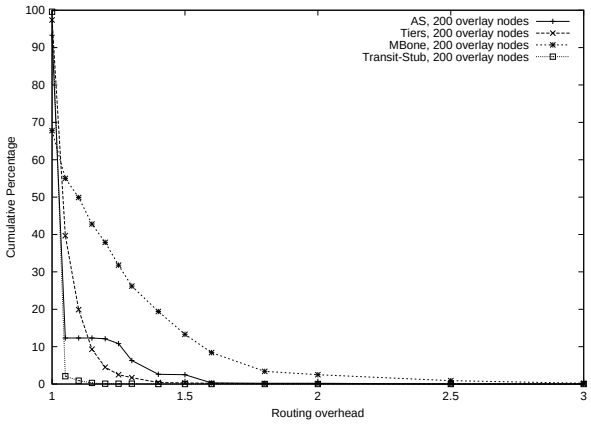


Figure 15. Overhead of routing on the overlay network

B.2 Routing overhead after local-link recovery

We now study the additional overhead when we use a local-link recovery strategy. With such a strategy, on detecting a failure, the upstream overlay node of the failed link finds an alternate path to the downstream node. This was illustrated in Figure 7. To simulate a link failure, we simply remove a link from the overlay graph, and recompute the path between the two-ends of the failed link.

Figure 16 shows the cumulative distribution of the routing overhead after such a recovery is effected. And Figure 17 compares the overhead before and after recovery for the Transit-Stub topology. In these cases also, we use 200 overlay nodes for all the graphs.

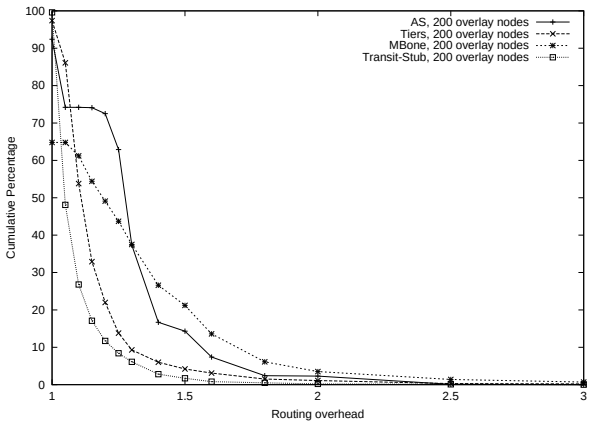


Figure 16. Overhead of routing on the overlay network, after local-link recovery

The distributions are similar to the previous cases, except

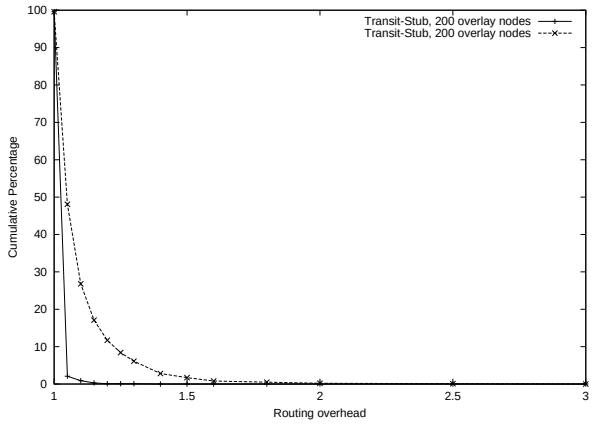


Figure 17. Overhead of routing, before and after local-link recovery

that the overhead after re-routing is higher. In the case of Transit-Stub topology however, only 6.1% of the end-host pairs experience an overhead of over 25%.

B.3 Effect of the size of the overlay

Figure 18 shows the routing overhead for the Transit-Stub topology for different sizes of the overlay graph (50, 100, and 200 nodes). We see that at least for this case, the number of overlay nodes has a definite effect on the routing overhead. The denser the overlay topology, the lesser the routing overhead.

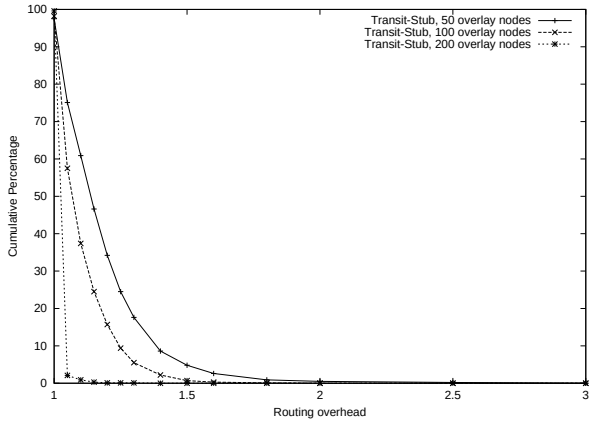


Figure 18. Effect of the size of the overlay

B.4 Time to establish "connection" state

We now present numbers from an implementation of the $GSM \leftrightarrow PCM$ codec. This is a codec that has widespread hardware based implementations deployed at the Inter-Working Function between the PSTN and the GSM networks. This is important for Internet-based integration of services between these networks [38].

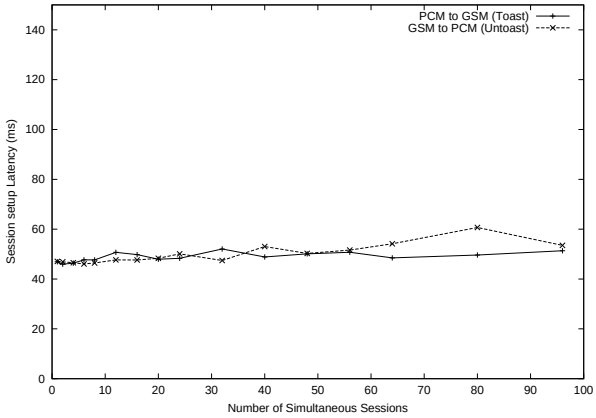


Figure 19. Latency to setup a session on a machine within a cluster

We reproduce numbers from [30]. Although the measurements were done in a slightly different context, these numbers show the latency involved in instantiating a new session with this particular codec. Figure 19 shows the session setup latency (to create a new intermediate service) for the coder and decoder for GSM audio. This is shown as a function of the number of simultaneous sessions already present (which is a measure of the load in the system). The measurements were done on a 500MHz Pentium-III 2-way multiprocessor machine. We see that the session setup latency is under 50ms independent of the load – this particular codec is not very CPU intensive.