A Conceptual Framework for Network and Client Adaptation

B. Badrinath, Armando Fox, Leonard Kleinrock, Gerald Popek,

Peter Reiher, M. Satyanarayanan

Abstract

Modern networks are extremely complex, varying both statically and dynamically. This complexity and dynamism are greatly increased when the network contains mobile elements. A number of researchers have proposed solutions to these problems based on dynamic adaptation to changing network conditions and application requirements. This paper summarizes the results of several such projects and extracts several important general lessons learned about adapting data flows over difficult network conditions. These lessons are then formulated into a conceptual framework that demonstrates how a few simple and powerful ideas can describe a wide variety of different software adaptation systems. This paper describes an Adaptation Framework in the context of the several successful adaptation systems and suggests how the framework can help researchers think about the problems of adaptivity in networks.

1. Introduction

Computer networks are becoming increasingly complex and variable, with mobility exacerbating the problem dramatically. Several researchers in the field of networking and distributed systems recognized this problem in the recent past, and started designing solutions to the problems of complex variability. Many of these researchers addressed the problem through different forms of software-supported adaptivity. Recently, systems embodying their ideas have been built, tested, validated, and, in some cases, deployed for production use, demonstrating the real power of software-supported adaptivity.

The authors examined the characteristics of the adaptive software systems they built and discovered that although the systems were independently designed and built, they shared three kinds of commonality:

1. The systems shared certain fundamental characteristics that could be described in fairly simple architectural terms.
2. The designers made similar design choices across the different systems.
3. Similar lessons were learned in the design and implementation of the different systems.

The framework presented in this paper captures these commonalities, clarifies several issues surrounding the structure and design of software that adapts to difficult network conditions, and suggests key issues that require further investigation in this field. The framework can also help other researchers characterize their own adaptive software and understand how it relates to other systems.

In section 2, we discuss in more detail the characteristics of modern networks that motivate the need for adaptivity, especially in the mobile computing arena. Section 3 briefly describes some of the systems that provided inspiration for the framework. Section 4 describes the framework. Section 5 presents how each of the sample systems from section 3 fits into the framework. Section 6 suggests ways in which the framework may help other researchers think about the structure of their own adaptive systems. Section 7 concludes with open issues that the framework exposes and suggests areas of future work.

2. The Need for Network Adaptation

Many of the characteristics of modern networks vary dramatically. Bandwidths currently provided by networking hardware in daily use range from a few tens of kilobits per second up to thousands of megabits per second. Similarly, bit error rates of commonly used network devices span orders of magnitude. Latencies can range from nanoseconds to large fractions of a second. Networks that contain mobile elements tend to experience a wide range of these characteristics, often with rapid changes.

The scale of today's and tomorrow's networks adds great complexity. High growth rates are expected for the future, even leaving aside the additional scaling potential of "smart spaces", where many billions of tiny embedded devices worldwide will have some networking capabilities. Such scale makes any form of static planning or optimization of network operations impossible.

We also demand far more of our networks than ever before. Not only is the total volume of traffic increasing at an alarming rate, but also new applications put new kinds of demands on the network. Web browsing, video conferencing, and Internet telephony have very different network requirements than such old Internet staple applications like electronic mail and file transfer.

Mobility greatly exacerbates the problem. Many of the computers being sold today are either portables or handheld devices. In the smart spaces world of the future envisioned by some, extremely small embedded devices will travel everywhere, be embedded in everything from walls to automobiles to shoes, all the while communicating, processing, controlling, actuating, capturing data, etc. A bewildering array of wireless networks is being deployed to serve such mobile devices.

The mobile environment also introduces another complication: heterogeneity in the communicating devices. Cell phones, personal digital assistants, palmtop computers, digital pagers, digital cameras and portable computers all have different capabilities and different requirements. Part of the difficulty of adaptation in the mobile environment is not just to deliver data over challenging network conditions, but to deliver it in formats suitable for the devices that need it.

Other issues, such as security and economic questions, also complicate the problem. Generally, adding the need for security to any computing question complicates it. The existing networking infrastructure that we have inherited was not designed with commercial use in mind; as a result, performing efficient, safe business transactions over that network infrastructure is challenging.

Moreover, the existing network protocols that have enabled the Internet revolution are not perfectly suited to the environment they themselves have created. TCP, for example, does not work well on noisy links (e.g., many wireless links), and often behaves poorly over satellite links due to long latencies. Researchers have changed some protocols to handle such problems, but our understanding of networks is insufficient to allow us to design protocols that behave well in the face of all probable network conditions. Even if we could develop such protocols, we would face the challenge of converting the enormous installed base of today's network infrastructure. The Internet is distributed, decentralized and vast, and the simple solution of complete replacement of that existing infrastructure is daunting.

But it is important to realize that even if we could successfully deploy new protocols quickly, problems would still remain. The real goal of adaptive networking is to provide good end-to-end service, where the end points are located in applications. Without considering the needs of applications and their users, no adaptive solution at the network level alone can solve the entire problem.

These trends suggest that we must deal with larger, more variable, more complex, rapidly growing networks that must meet ever increasing demands, yet rely largely on existing networks and protocols. One general class of solutions to solving this problem is to allow various forms of adaptation of network traffic. Such solutions allow hardware or software to alter the protocols or the data content being transmitted to provide a better quality of service to users.

Data flows over networks can be usefully adapted in many ways:

- The underlying protocol can be altered to handle difficult conditions. The Berkeley snoop protocol improves TCP over high error rate links [BSAK95]; an adaptation mechanism can slip the snoop protocol into place when such links are established [AHKO97].

- The data can be altered in a lossless way. Various systems allow data compression or encryption across links with poor connectivity, without any application involvement.

- Lossy adaptations can be used to obtain better compression of data over limited links by dropping inessential portions of the information, or sending a lower-fidelity version. TranSend improved performance by an order of magnitude or better using lossy compression [FGCB98].
- Data can be automatically converted to formats better suited to the end systems or the intermediate networks. The Top Gun Wingman browser [FGG+98] converts Web images into 2-bit grayscale bitmap displays before sending them to Palm Pilots. Mowgli [LHKR96] converts GIF images to more compact JPEG before sending them over wireless links. Although adaptation to client heterogeneity is an important area in which extensive work has been done (see [FGCB98] for an overview and pointers to related work), in this paper we focus on adapting to *network* variability, remarking that the architecture we describe has been successfully used to address client adaptation as well.

Adaptive solutions to network problems embrace many interesting variations: the various proxies built at Berkeley [FGCB98], the Odyssey system [NSN+97], transformer tunnels [SB98], active networks [TW96], and intelligent agents [TK96]. While these systems have some very significant differences, all offer methods of changing the contents of the transmitted data or the methods used to send that data. All adapt to changing conditions specific to the data transmission requested, or to prevailing network conditions, or to needs of the users. This body of research has many successes, but none claim to solve the complete problem or even to suggest a framework for thinking about the problem and its solution. This paper's goal is to propose such a framework.

3. Some Characteristic Adaptive Systems

Although at first glance there may appear to be little commonality across the wide variety of approaches to network adaptation, significant commonality is revealed by closer examination of the decisions made by independent researchers taking different approaches to the problem. We present below several independently designed, operational systems developed by one or more of the authors. While the chosen systems certainly do not cover all work done in the field (or even all work in the field by the authors), they illustrate the wide variety of possibilities in adaptive network software solutions. Each system's designers started from the assumption that adaptivity was required to solve some set of problems, but otherwise the design assumptions varied radically. Examples of differences include the following:

- Application-transparent vs. application-aware adaptation: is the application informed that adaptation is occurring and perhaps expected to provide an application-level response (as in Odyssey), or does the system attempt to completely shield the application from this fact (as in Conductor)?
- General vs. application-specific adaptation: does the system provide general machinery to support a collection of unrelated applications (as in disconnected file systems such as Coda), or does it support a specific application or narrowly-defined class of applications (as is the case for TranSend)?
- Does the adaptation machinery reside in the client, in the server, in one or more intermediate proxies, or all of these?

Despite such differing goals and assumptions, some key common ideas and themes emerged. We now examine these example systems, which on the surface appear extremely different. Closer examination of their conceptual architectures, however, reveals strong similarities, which we tie together with the framework we describe in Section 4.

3.1 UC Berkeley TranSend

UC Berkeley's *TranSend* Web accelerator proxy [FBA96] was one of the earliest projects to explore adaptation proxies aggressively. TranSend intercepts HTTP requests from standard Web clients and applies datatype-specific lossy compression when possible; for example, images can be scaled down or downsampled in the frequency domain, long HTML pages can be broken up into a series of short pages, etc. TranSend's primary goal was to provide network adaptation for users of slow links, such as UC Berkeley's modems or the Metricom Ricochet service [Met94], which is popular in the Bay Area.

TranSend supports a wireless vertical handoff mechanism [SK97]. When a client equipped with multiple wireless interfaces switches between wireless networks, the client-side vertical handoff software (which is completely independent of TranSend) generates a notification packet containing some essential characteristics (e.g., estimated expected throughput) of the new network. This packet would be sent to a special UDP port on TranSend where the notification would be processed and stored in a per-client profile. TranSend would then process future requests from that client in accordance with the new network type; for example, very aggressive image downsampling was performed for clients connecting over Ricochet with an expected throughput of 15-25 Kb/s, whereas compression was much less aggressive (and in some cases disabled) for WaveLAN clients connecting at about 1 Mb/s.

Because HTTP is a "stackable" protocol (i.e. it is possible to have several HTTP "hops" in a request chain), TranSend-based adaptations are naturally composable, allowing a multilevel system with some "baseline" compression performed far upstream, and additional compression performed near the clients.

TranSend evolved into a general system for deploying scalable, fault-tolerant adaptive applications [FGCB98]. Top Gun Wingman [FGG+98], for example, allows users of thin clients such as the USR PalmPilot handheld device to browse the Web. Although similar in spirit to TranSend, Wingman provides an additional service, a *network adapter*. TranSend uses HTTP to communicate with clients and servers, but the PalmPilot's modest capabilities suggested a simpler protocol. A simple datagram-based client-to-adapter protocol that also encapsulates security and encryption was crafted for Wingman. Wingman's proxy-side adapter translates between this protocol and HTTP, giving Wingman the ability to access existing Web servers. When Wingman was evolved into a PalmPilot implementation of the shared whiteboard [CFMB98], the network adapter was augmented to tunnel multicast to the PalmPilot over a unicast TCP connection, to compensate for the PalmPilot's inability to handle multicast directly; this is another example of network adaptation.

3.2 CMU Odyssey

Odyssey is a system built at Carnegie Mellon University to support challenging network applications on portable computers [NSN+97]. Odyssey particularly focuses on resource management for multiple applications running on the same machine. Odyssey was designed primarily to run in wireless environments characterized by changing and frequently limited bandwidth, but the model is sufficiently general to handle many other kinds of challenging resource management issues, such as battery power or cache space. The goal of the system is to provide all applications on the portable machine with the best quality of service consistent with available resources and the needs of other applications.

Odyssey is an application-aware approach to adaptation intended primarily to assist client/server interactions. The Odyssey system consists of a *viceroy*, an operating system entity in charge of managing the limited resources for multiple processes; a set of data type-specific *wardens* that handle the intercommunications between clients and servers; and applications that negotiate with Odyssey to receive the best level of service available. Applications request the resources they need from Odyssey, specifying a window of tolerance required to operate in a desired manner. If resources within that window are currently available, the request is granted and the client application is connected to its server through the appropriate warden for the data type to be transmitted. Wardens can handle issues like caching or pre-fetching in manners specific to their data type to make best use of the available resource. If resources within the requested window are not available, then the application is notified and can request a lower window of tolerance and corresponding level of service. As conditions change and previously satisfied requests can no longer be met (or, more happily, conditions improve dramatically), the viceroy uses upcalls registered by the applications to notify them that they must operate in a different window of tolerance, possibly causing them to alter their behavior.

3.3 UCLA Conductor

The UCLA Conductor system allows deployment of cooperating adaptive agents at specially enabled nodes throughout a network [YRP99]. Conductor is an application-transparent adaptation mechanism. Applications can benefit from Conductor without being recoded or explicitly requesting its services. Instead, the underlying system is configured to indicate what kinds of data flows Conductor is capable of assisting and the Conductor system automatically traps and adapts those data flows.

Conductor also handles issues of composing adaptations in support of a single flow at multiple nodes. Conductor determines the characteristics of the data path from source to destination and determines if the path will meet the needs of the applications using it. If not, Conductor will automatically deploy adapters at one or several of the available nodes along the path to adapt the data flow to network conditions, allowing better application-visible network behavior. Conductor plans the cooperative behavior of the agents and handles problems of transient or long-term failure of particular adapter nodes.

Conductor is designed to handle general-purpose adaptations, including both lossy and lossless adaptations. Combining lossy adaptations and reliability is especially challenging, since a lossy adapter may drop part of the data or may transform several data packets into fewer packets. If an adapter or its node fails, some of the adapted packets could be delivered while others were not. Without the lossy adapter's state to determine which original packets were dropped or coalesced, the system may find it difficult to resume transmission without either duplicating already received information or failing to deliver required information. Unaware applications are generally unprepared for either problem, so Conductor must hide these problems from such applications. Conductor attaches numbers to pieces of semantic content that do not vary when adapted. For example, if every other packet is dropped, the undropped packets are renumbered to include the dropped packets. The system is thus able to determine which information has and has not been delivered despite failures.

3.4 UCLA Smiley

Smiley is an intelligent agent real-time program developed at UCLA to augment Web browsers [JK99]. It has two components: (i) a dynamic Graphical User Interface that informs users of the nature of the links on a Web page, and (ii) a transparent agent that prefetches carefully selected links. The GUI provides users a measure of the quality of connectivity available between themselves and the servers they contact to obtain Web pages [JK99], and of the nature of the data residing behind that link. It was designed to handle both the kinds of limited links common in mobile computing and general connectivity and bandwidth problems in the overall network. Smiley's GUI provides user feedback, in the form of augmentations to the links shown on a Web page, allowing the user to predict the likely effect of clicking on a particular link. This feature allows a user to avoid requesting a page that is unavailable or will take a long time to retrieve. Smiley prefetches web pages intelligently to allow users to browse more effectively over limited and variable links. A prefetch threshold algorithm is used to decide when to prefetch a web page the user hasn't yet asked for. Smiley includes models that consider different users associated with different time and bandwidth costs, trying to minimize the average cost for each request in the entire system.

3.5 CMU Coda

Coda is an optimistic file replication system developed for the mobile computing environment that uses client/server optimistic replication to maintain replicas of files required by disconnected or poorly connected clients [KS92]. Optimistic replication permits any replica of a file to be updated freely (as allowed by normal file system access permissions), without regard to the status of other replicas. Optimistic replication provides great performance and availability advantages over other replication alternatives, at the cost of occasionally permitting concurrent updates. Experience with and measurements of Coda [KS92] and other optimistic replication systems [RHR+94] shows that concurrent updates are uncommon in practice, and many of them can be resolved without human intervention.

Coda's server copy is kept on a well-connected machine that the portable computers contact when possible. Updates performed by the portable computer during disconnection are saved in a log, which is replayed to the server when possible. The server detects any concurrent updates and rejects them, requiring the client to use automated conflict resolution mechanisms to resolve any problems resulting from such concurrency [KS93, KS95]. The client portable also requests new updates from the server.

Adapting to network conditions was not the primary goal of Coda, but experience with its operation in the mobile environment caused the Coda designers to extend it to do so [MES95]. Coda performs trickle reintegration when only limited bandwidth is available for communicating updates to the server. This method of reintegrating updates from the mobile computer to the server allows effective, adaptive use of the available bandwidth between the two machines.

3.6 Rutgers Environment Aware API

Application adaptivity implies that applications must be structured to receive notifications about any important changes in the environmental state and to react appropriately. Since the network state is complex, the applications must interact with many environmental conditions, sources, and possible reactions. The Rutgers Environment Aware API addresses this problem. This API is based on a flexible mechanism for asynchronous event delivery. Environmental changes are modeled as asynchronous events that are delivered to mobile computing applications over an entity called an Event Channel [WEBA98]. This entity implements the event delivery mechanism. The events are organized as an extensible type hierarchy, and the architecture itself can be configured and extended. This extensibility enables support for a new condition to be easily incorporated into an existing system. A novel feature of the API is the ability to utilize event type information not only to filter out uninteresting events, but also to handle an event at an appropriate level of abstraction. An application that chooses to be environmentally aware creates a handler for that event type. The application specific response to the new situation is encoded in this handler and is invoked when the appropriate event is delivered.

4. A Conceptual Framework for Network Adaptation: The Adaptation Framework

Careful thought about these and other network adaptive systems reveals important common themes. We now present a conceptual framework that encapsulates those themes. Each of the systems presented above maps well into this framework, despite their many different details.

The framework had to display certain characteristics:

- it should encompass all reasonable alternatives to major design questions
- it should be as simple as possible (but, to quote Einstein, no simpler)
- it should consider issues of incremental deployment of different technologies, interoperation with legacy systems, and other practical issues
- it should make interoperation between different adaptation technologies easier
- it should distill the extensive knowledge, experience, and real systems produced for adaptation
- it should provide a starting point and common vocabulary for describing future work in the important area of adaptive architectures
- it should not preclude future innovations that provide alternative approaches to adaptive networks

Data flowing across an arbitrarily large and complex network of varying characteristics should be delivered to its destination in the best manner possible, given a variety of constraints. Some of these constraints relate to physical and technological limitations, such as the speed of light or the capacity of a link on the path. Others relate to systems concerns, such as the need to share a link or the costs of providing reliable delivery. Given the wide variety of possible conditions that could be present in the network, many different adaptations to the data flow could prove beneficial.

The essence of the problem is illustrated in Figure 1. A process on a source node sends data to a process on a destination node. The data flows across various links and nodes in the network. The thickness of the connecting lines is meant to suggest relative capabilities of the links involved in the data flow.

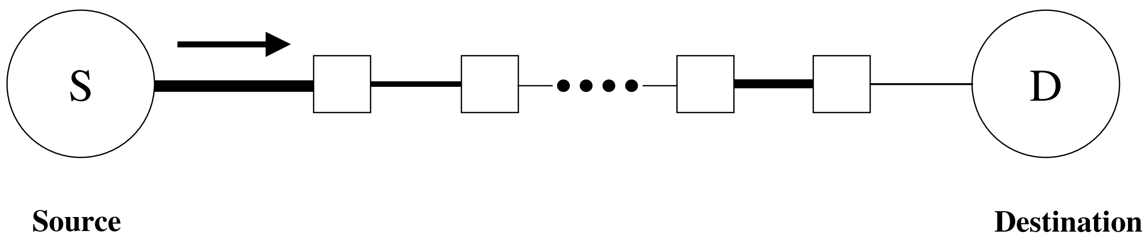

Figure 1: A data flow in a variable network

To some extent, this figure is a simplification of the general problem. It shows a simple data flow with a single source (S) and destination (D), and it does not illustrate problems such as delivery deadlines or security concerns, nor does it suggest the level of complexity possible in even a single network data flow. But the figure captures the heart of the problem. A stream of data flows from a source to a destination across a network, using links of varying capabilities. At some or all points in the network, altering the data flow in various ways could lead to better overall results, from the point of view of the sender, the receiver, the administrator of the network, or the complete population of network users. Without some mechanism to apply such adaptations, however, no improvements can be made.

Figure 2 shows how the introduction of adapters alters the situation. Now, the data can be altered in various ways, allowing for better results. *Adaptation Agencies* (labeled AA in the figure) represent many different kinds of adaptation mechanisms, from adaptive protocols to heavyweight code executed on behalf of the data flow. Note that all adaptive components in this diagram are optional, and that any single AA can be replaced with multiple AA's arranged in complex ways. The degenerate case where all are omitted is a simple client-server or peer system with no adaptivity support.

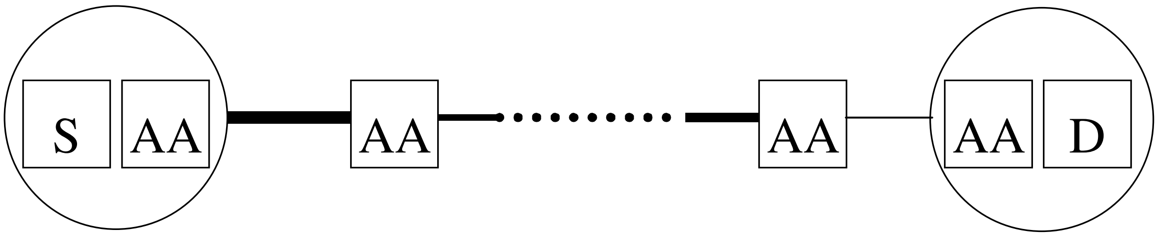

Figure 2: Adapters assist the data flow

Figure 3 shows how the Adaptation Framework fills in the details of Adaptation Agencies. An AA consists of several parts:

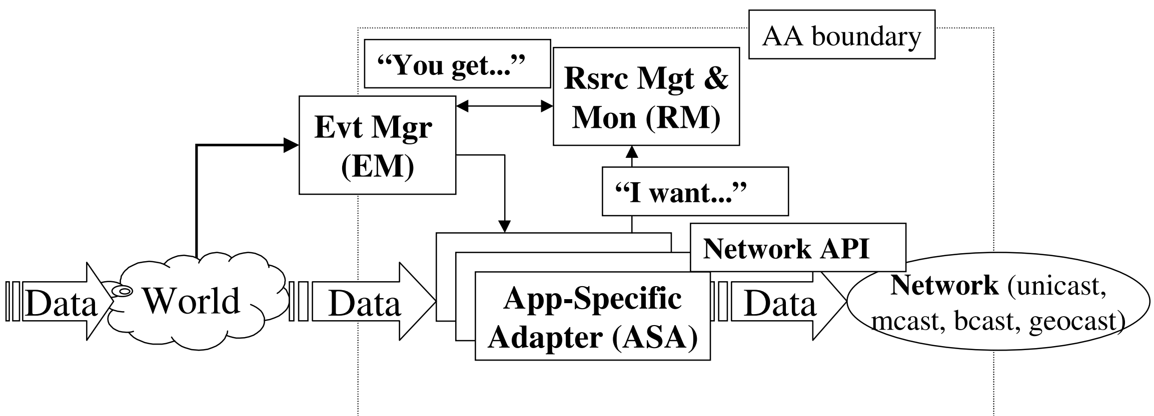

Figure 3: An Adaptation Agency

- The *Event Manager* (EM) monitors the AA's environment. The components of that environment are defined broadly, for generality, but are likely to include things like traffic and error conditions on network links, available CPU cycles on a local processor, or security threats that have been detected. The event manager can receive control messages that will alter the behavior of the AA. These messages can originate from other AA's, from local operating system services, or from applications.
- The *Resource Management and Monitor* (RM) component handles resources under direct control of the AA. If the AA has been allocated a certain percentage of a data link's bandwidth, the RM determines how to best use that bandwidth to meet the needs of all data flows under its control.

- Each AA may contain zero or more *Application Specific Adapters* (ASA's). These modules are capable of performing some particular adaptation on a data stream. Each ASA requires certain resources to perform its adaptation properly.

An Adaptation Agency accepts data from some source and delivers possibly adapted data to some other destination. The source may be one network link and the destination another network link, but source and destination might also be other AA's. If a particular AA is working directly with a network, however, it will have some knowledge of the specifics of that network, such as whether the network supports broadcast or not. The AA can use this knowledge when performing adaptations.

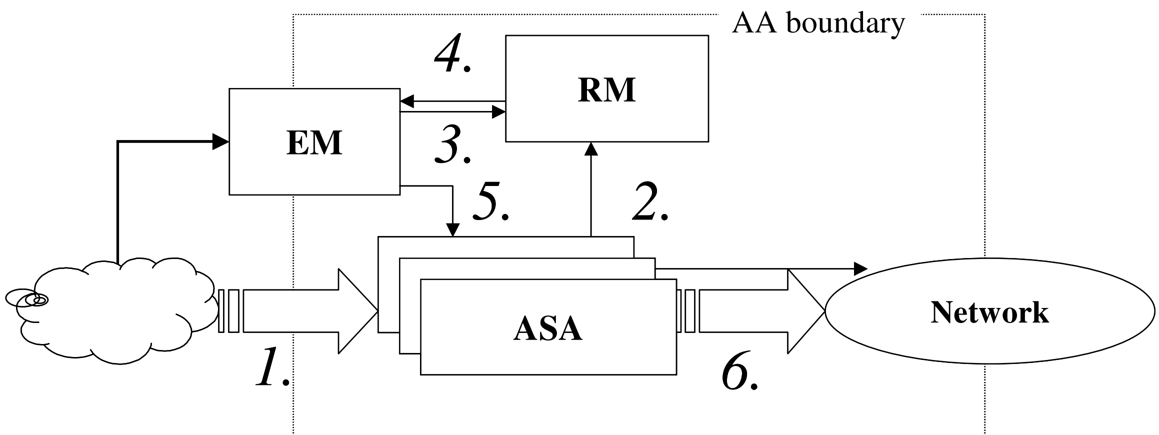

Figure 4: Data Flow Through an AA

The connection and interaction of AA components is also important. (See figure 4.) Generally, data comes into an AA and is delivered to one of its ASA's (1), which decides whether to adapt the data. If resources are required for an adaptation, the ASA requests them from the RM (2). The RM can accept or reject such a request, based on what resources are available and its resource allocation algorithms. The RM obtains the availability information from the EM (3), which sends the RM updates whenever significant events occur. When the RM has decided on how to handle a request from an ASA, it informs the EM of the new resources that have been made available to the ASA (4). The EM can then alter its view of local conditions, and can also deliver the response to the ASA (5). The EM will also signal the ASA when other situations lead to changes in conditions relevant to ASA operations. After adaptation, the ASA passes the adapted data into the network for delivery to the destination or the next ASA (6).

AA's can be organized hierarchically, with one AA controlling a group of other AA's, allowing the framework to specify that one entity control a shared resource for several other entities. Figure 5 demonstrates this concept. Two disjoint data flows pass through a single physical entity, which could be a gateway machine, a network link, or an entire local area network. The data flows must in some way share the physical entity's resources. The adaptation framework handles this issue by permitting a higher level AA to assume control of all of the physical entity's shared resource. It then communicates with the Event Managers of the AA's actually supporting the two data flows to tell them how much of the shared resource is available to them. These lower level AA's in turn communicate internally with the ASA modules chosen to use for adaptation of each data flow. The hierarchy can continue to higher levels, if necessary, allowing one set of AA's to handle data flows, a higher level set to mediate shared use of a switch or gateway, and an even higher level AA to coordinate overall network activity through its instructions to the middle level AA's.

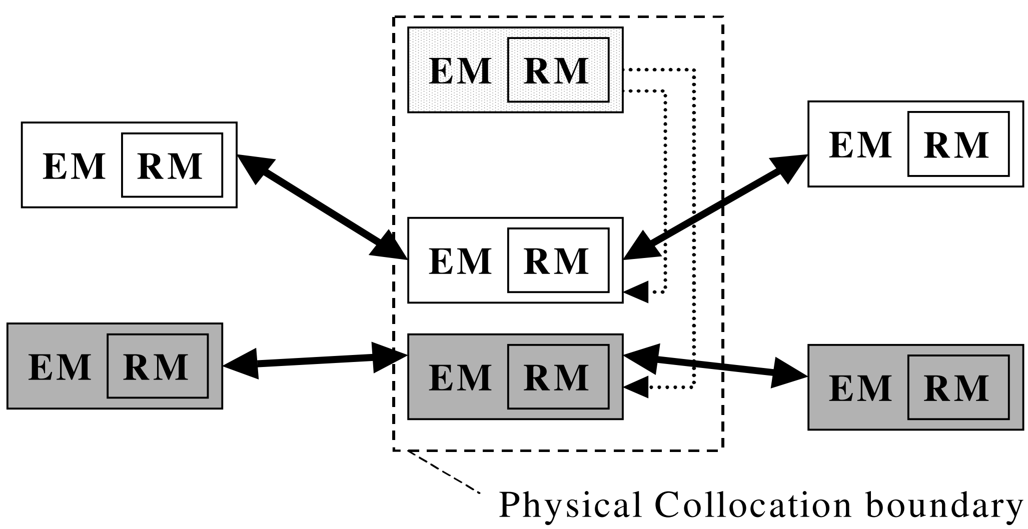

Figure 5: Adaptation Agencies using a shared resource

5. Mapping Real Systems Into the Adaptation Framework

The Adaptation Framework is intended to encompass a wide variety of adaptation mechanisms. Here we describe how the systems described in section 3 can be fit into this framework. For each system, the accompanying diagram shows as shaded the sections of a single ASA (or, in some cases, multiple ASA's) that are provided by that system.

5.1 TranSend

TranSend can be thought of as a complete Adaptation Agency (AA) that initially ran on a single workstation but was later extended to run on a cluster. The entire cluster can be regarded as a single AA that serves extremely large communities of users [FGC+97]. Within the AA, TranSend contains a separate ASA for each MIME type (GIF, JPEG, HTML, etc.) Incoming data is either passed to the appropriate ASA by type, or passed directly through the AA to the client if no appropriate ASA exists. The ASA then performs datatype-specific lossy compression before forwarding the data.

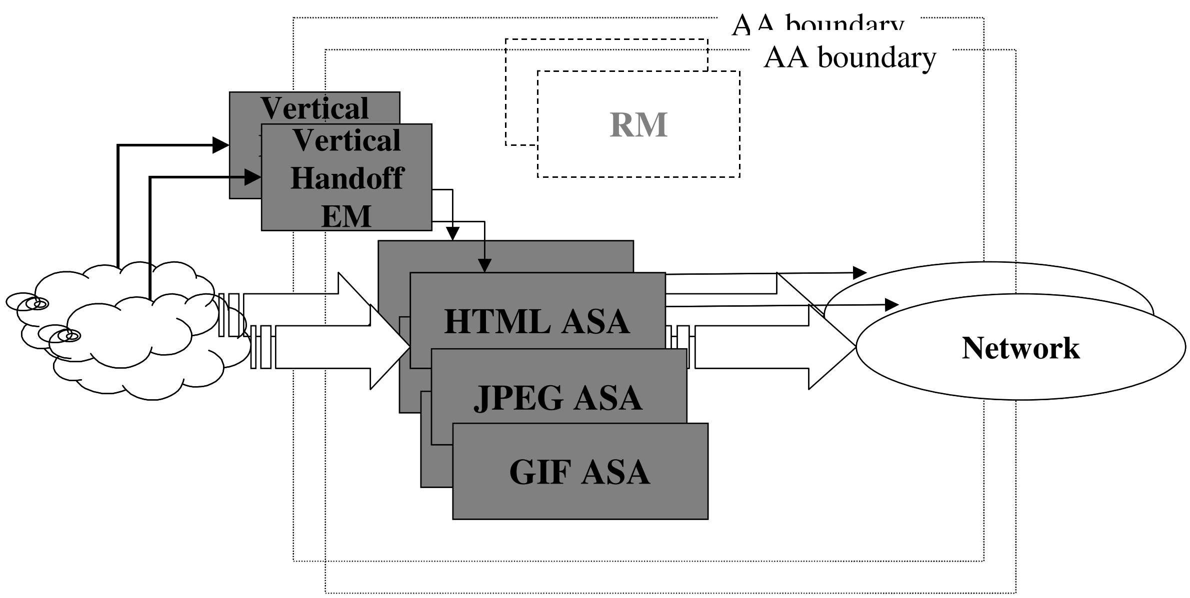

Figure 6: Mapping TranSend to the Adaptation Framework

Transend's vertical handoff mechanism worked with a simple Event Manager (EM) to determine when handoff was necessary. Because TranSend was designed under the assumption that it would have use of all the workstation's resources, no RM was designed into it. However, external RM schemes such as SRI's Resource Management framework should be able to interoperate with TranSend.

5.2 Odyssey

Odyssey fits well into the adaptation framework. Odyssey on a portable node is a single AA. The viceroy is a combination of resource manager and event manager. The wardens are ASA's specific to individual data types. One Odyssey AA can host several warden ASA's.

One interesting aspect of Odyssey with regard to the adaptation framework is that much of the adaptation in this model is actually done by the applications, which interact with Odyssey. For example, Odyssey itself doesn't decide that color video frames should be converted to black-and-white, but rather instructs the application that some action is required. The application itself decides how adaptation should occur, and typically instructs its server to make the adjustment. Alternately, the application can request even higher-level control, such as requesting user advice on the kinds of adaptations that should be applied when conditions change. This aspect highlights the architecture's inclusion of the possibility of control traffic between applications and AA's.

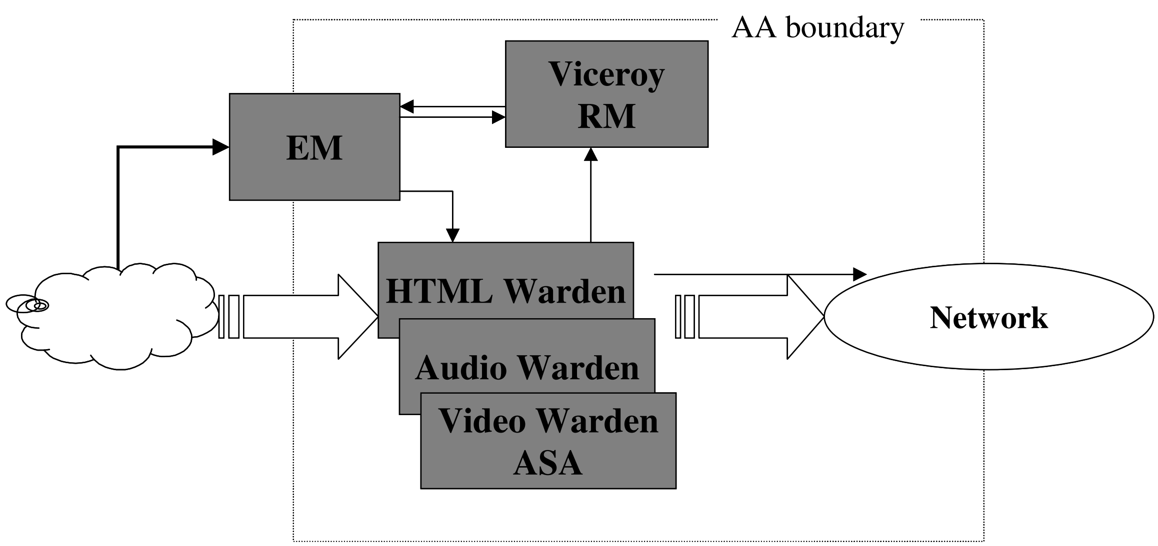

Figure 7: Mapping Odyssey to the Adaptation Framework

5.3 Conductor

Conductor can be regarded as a set of complete AA's that cooperate to plan and regulate the overall behavior of a connection. Each Conductor node hosts an AA that will allow adaptation of multiple flows through that node. The Conductor AA contains a RM that allocates the resources the node makes available to Conductor between the different flows the local Conductor AA controls. It has an EM that captures new data transmissions coming in or originating at the node, monitors the progress of data flows, and watches for control information sent by other Conductor AA's. Multiple ASA's can be run at a given Conductor node, either composed for the benefit of a single data flow, or separate for the benefit of multiple independent data flows. The Conductor architecture also permits independent data flows to share an ASA, such as a caching or prefetching adapter.

Conductor sends information between its AA's to assist in planning the deployment of agents and to watch for failures. This information is processed in a distributed fashion. Essentially, the AA's cooperate to create a plan at the start of a data flow. This plan indicates which ASA's should be located at given nodes, and may suggest how each ASA should behave. If connections fail, the nodes involved in a flow on either side of the failure can replan to handle the failure. They can choose to shut down the flow, re-route the flow (requiring, in general, a new plan and new ASA's), or perform some local actions in anticipation that the failure will be fixed shortly. An example of the latter would be prefetching data from the source while waiting for a transient connection to reappear.

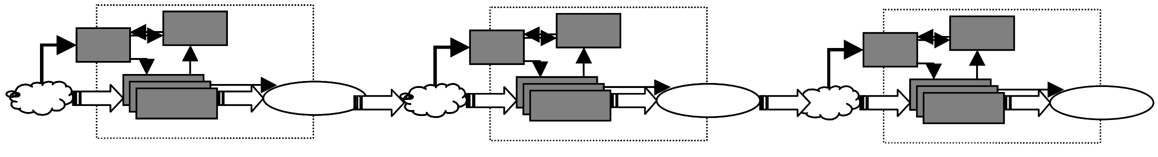

Figure 8: Mapping Conductor to the Adaptation Framework

5.4 Smiley

Smiley can be regarded as a special purpose AA that resides on a mobile node, supporting a single adaptation. It contains an RM that worries about the available link bandwidth, an EM that sends out probes to the network to determine connectivity and latency information, and prefetching and page rewriting ASA's. Smiley is an example of an AA that maps tightly to a particular application.

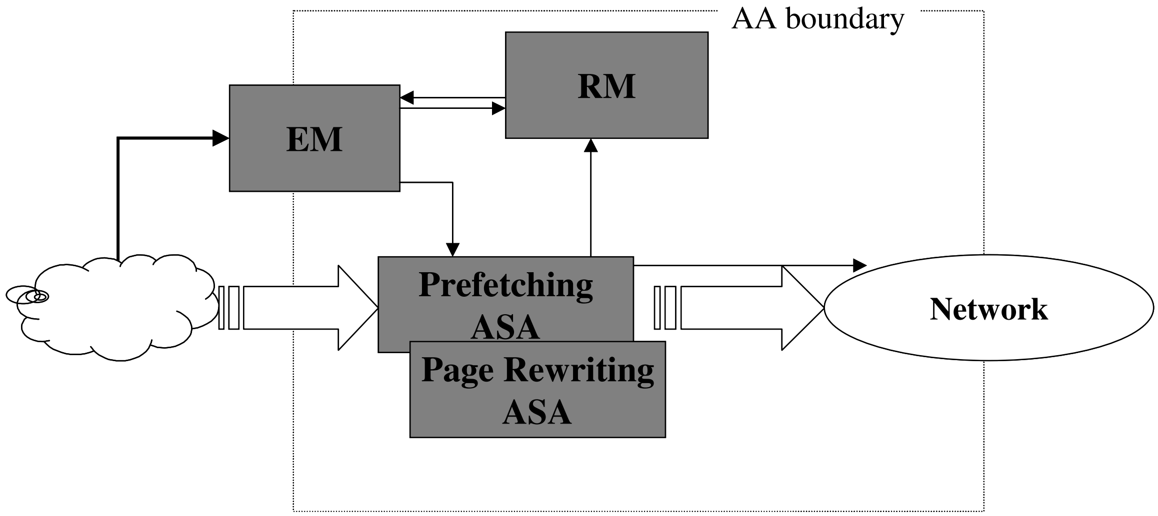

Figure 9: Mapping Smiley to the Adaptation Framework

5.5 Coda

Coda shows how the framework can incorporate application and system software with adaptive components. Coda's trickle reintegration suggests an AA at the client side that uses an event manager to monitor the available bandwidth. The Coda cache manager, Venus, combines the ASA, EM, and RM functions. Venus acts like an ASA to select updates to reintegrate with the server replica and feed them across the limited bandwidth link. Venus also performs EM functions to watch the link and RM functions to handle usage of the link.

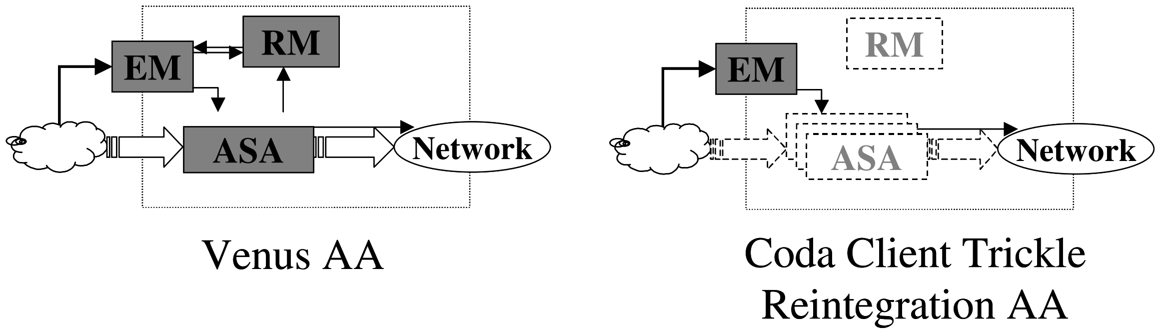

Figure 10: Mapping Coda to the Adaptation Framework

5.6 The Rutgers Environment Aware API

The monitoring and delivery of events over the event channel in the Rutgers approach is an example of the EM in the architecture. The EM monitors the environment and also delivers events of interest according to a system-defined policy. The event handler also provides a framework for implementing an ASA. The application can install separate ASA's for each interesting event type. When the EM delivers a notification, the appropriate ASA is invoked. The ASA responds to the new situation appropriately for its application. For example, when a new network is detected, the characteristics of the network such as expected bandwidth are encapsulated in the event. The ASA can use this information in its response by changing the transmission from rich data to summary data and vice versa.

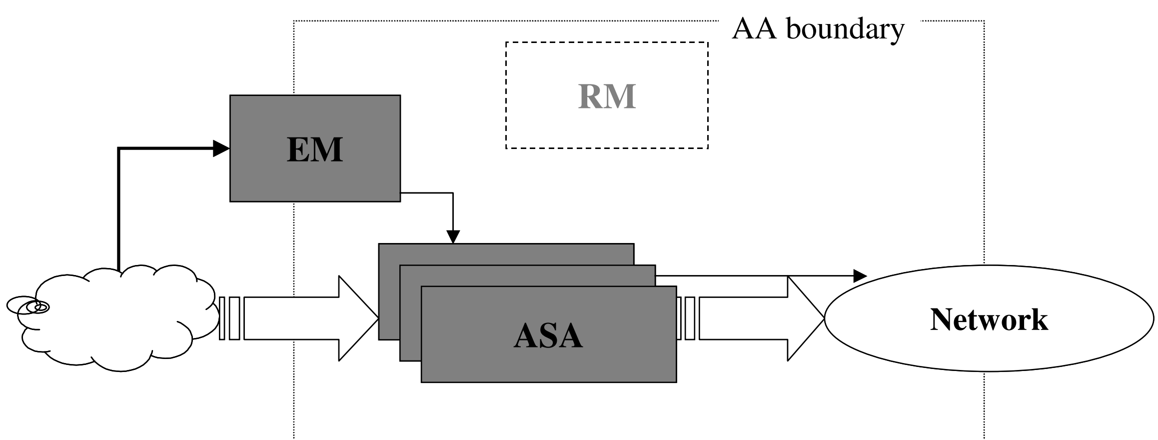

Figure 11: Mapping the Rutgers Environment Aware API to the Adaptation Framework

5.7 Commercial Systems

The research projects discussed in this article have already influenced commercial efforts. The network adaptation ideas pioneered in TranSend have appeared in various commercial products including Intel QuickWeb. The more aggressive adaptation pioneered in the Top Gun Wingman handheld Web browser has been commercialized by ProxiNet, Inc. (now a division of Puma Technology). Emerging mobile-computing standards from the W3C (World Wide Web consortium), including XML (Extensible Markup Language) and XHTML (Extensible HTML), incorporate mechanisms for "hinting" to intermediate Adaptation Agents to help them adapt content delivery to a range of networks and devices.

The WAP (Wireless Application Protocol) suite is a stack of protocols designed specifically for delivering data and interactive services to the "smart cellular phone" class of mobile devices [WAP97]. Although the application-level markup and scripting languages (WML and WMLscript) include features motivated by the limited capabilities of the intended client devices, the languages do not appear to provide any functionality that directly facilitates the application-level adaptation we motivate in this retrospective. It will be interesting to see whether the evolution of the WAP protocols will follow the pattern of HTML, where application-level adaptation machinery has to be retrofitted after the protocols have become entrenched.

5.8 Summary

The following table summarizes how each of the example systems fits the Adaptation Framework.

System	Description	ASA	EM	RM	Network
TranSend	Web acceleration through datatype-specific lossy compression	MIME-type-specific compressors	Wireless vertical handoff notification affects compression aggressiveness	N/A	Composable/stackable using HTTP
Odyssey	Application-aware adaptation by multiple applications using diverse data types	Wardens (device driver-like OS extensions)	Log of passive bandwidth observations	Viceroy	Runs on arbitrary networks. API expressive enough to handle resources such as power and cache space.
Conductor	Combines adaptations across a network for a data flow	Supports arbitrary adaptations	Monitors network conditions and notifies RM of changes.	Handles planning and re-planning of adapter deployment	Uses TCP from point-to-point, provides end-to-end reliability itself.
Smiley	Web prefetching matched to network conditions	Prefetching and web page rewriting.	Probes remote Web servers to determine current network conditions	Handles local link and cache	Runs on arbitrary networks
Coda	Trickle re-integration matches log replay to channel characteristics	Decides what and how much to send and keeps track of incomplete transmissions	Observes available link bandwidth to server	N/A	Specifics other than bandwidth and latency are transparent to Coda
Environmen Aware API	Management of environmental change and application level reaction	Event handler which encodes the response of the application	Framework for monitoring the status if the environment and delivering the induced change to applications	N/A	Not specific to network attributes Supports other considerations such as power.

6. The Adaptation Framework and the Structure of Adaptive Applications

Our framework distinguishes the functionality of specific components of an adaptive application, and we have argued that this decomposition captures a broad class of adaptive applications. This decomposition also provides the ability to decouple the various adaptation-related entities from each other. Certainly in some cases tight coupling between entities can lead to a more efficient implementation; for example, responding to an event by invoking a registered upcall is fast and efficient and may not even cross an address-space protection boundary. However, in cases where loose coupling provides acceptable performance and sufficiently small overhead, it offers some important benefits:

1. It allows applications to be designed to function in either adaptive or non-adaptive environments, depending on whether environmental monitoring information is available. This simplifies application development by avoiding the need for "hardwiring" the monitoring machinery directly into the application.

2. It allows the components to be designed as separate autonomous subsystems that are loosely coupled and operate essentially independently. For example, using multicast, an environment-monitoring subsystem can be in a continuous-monitoring mode in which interested parties subscribe to specific types of environment-change events and react to them, rather than using a tighter coupling (such as the registering of upcalls) to support dynamic adaptation.

3. The third advantage derives directly from the first two: systems composed of autonomous, loosely-coupled modules are more robust, generally less susceptible to cascading failure (because of the inherent fault isolation afforded by module autonomy), easier to maintain, and often easier to deploy incrementally than their more tightly coupled counterparts.

For example, some applications in our framework do not require the presence or functionality of the EM; they function correctly without it, but display better adaptive behavior when it is functioning. The

TranSend application goes a step further by decoupling the mechanisms used for communication between the EM and the ASA's: in TranSend, the EM is a separate process that multicasts network-change events on a well-known multicast channel. The EM can function without TranSend (it doesn't matter that no one is listening to a multicast transmission) and vice versa (if no events are received, TranSend continues to function with its current settings). Such techniques contributed to the "infrastructure-level" degree of robustness achieved in the scalable cluster-based server that hosted the second-generation TranSend prototype [FGC+97]. The decoupling made possible by a careful implementation of our framework may be a worthwhile starting point for the design of future adaptive applications. We consider it a strength of our framework that it accommodates both loosely-coupled and tightly-coupled implementations, as circumstances and needs may require.

7. Open Issues and Conclusions

This framework is merely a starting point for thinking about the general characteristics of software that supports network adaptivity. Many important issues are clarified, but not solved, by this framework.

7.1 The Adaptation Framework and Active Networks

Active Networks [TW96] defines a very general model for programming the network. In its full generality, potentially every network packet can carry code, and every network entity (including routers and general computation nodes) can execute that code and maintain state. In addition to network adaptation such as we have described, ANs attempt to address a wide range of other tasks involving computation in the network, such as packet filtering, encryption, and incremental protocol deployment.

Currently, the exact definition and architecture of Active Networks are topics of research. In many cases, however, ongoing Active Network projects are producing software that is likely to fit well into the framework. As the research community defines the architectural components of an Active Network more precisely, we anticipate that architecture will map comfortably into the framework outlined here.

7.2 Interactions between adaptations at different levels.

Adaptation can occur at multiple levels, as it does in the sample systems discussed. Some adaptations relate to altering the behavior of an underlying protocol, some to altering the behavior of an operating system, some to altering the behavior of an application. In some cases, different adaptations might be applied at different levels of the overall system. How such adaptations would interact is far from clear.

Similarly, the framework points out the possibilities of composing adaptations, even those at the same level. Some systems, such as TranSend and Conductor, already support some forms of composition, but the framework points out many possible methods of composing adaptations. However, the methods used to determine that composed adaptations produce the desired behavior, particularly when they are being deployed and composed automatically, are unknown.

7.3 Breadth of applicability.

While this framework more than adequately describes the systems developed by the authors, and other systems with which they are familiar, the model is new, and has followed the development of these adaptive systems rather than preceded them. Whether the framework contains sufficient generality and features to properly describe all worthwhile adaptive software systems remains to be seen. Further examination of the alternative methods being used throughout the research community and deeper thought may further refine the framework.

The framework, as it stands, is not an architecture. No APIs have been defined that describe how data and control information flows into and out of AAs and their components. While the individual systems discussed above all map neatly into the framework, none of them could be seamlessly and effortlessly connected, as presumably they could be if they conformed to a single architecture. Conversion of the framework into a true architecture would require tight specification of the APIs between its components and validation by re-writing several adaptive systems to conform to these specifications.

The OSI seven-layer reference model provides a useful analogy to our framework. Like our model, the OSI reference model is not an architecture but a framework. The OSI model proposed a tremendously useful way to think about networking protocols. It allowed the community to discuss key issues and to define specific architectures. It provided a decomposition and layered structure that accelerated implementation considerably. Many systems violate the OSI model, but those very violations are all the more understandable and valuable precisely because we can place them in the context of a framework. We believe that the adaptation framework outlined here can serve a similar role in the increasingly important field of network adaptation.

The framework outlined in this paper is primarily intended to distill the common lessons learned from several successful network adaptation systems. The authors hope it will lead to more general discussion and study of the properties of network adaptation systems and the important features of such systems.

References

[AHKO97] Mark Allman, Chris Hayes, Hans Kruse, Shawn Ostermann, "TCP Performance Over Satellite Links," 5th International Conference on Telecommunications Systems, 1997.

[BHAK95] Hari Balakrishnan, Srinivasan Seshan, Elan Amir, and Randy Katz, "Improving TCP/IP Performance Over Wireless Networks," Mobicom 95, Nov. 1995

[BMMM95] Charles Brooks, Murray S. Mazer, Scott Meeks, and Jim Miller, "Application-Specific Proxy Servers as HTTP Stream Transducers," Fourth International World Wide Web Conference, November 1995.

[CFMB98] Yatin Chawathe, Steve Fink, Steven McCanne, and Eric A. Brewer. *A proxy architecture for reliable multicast in heterogeneous environments.* Proc. IFIP Middleware 98, Lake District, UK, Sept. 1998.

[FBA96] Armando Fox, Steven D. Gribble, Eric A. Brewer and Elan Amir. Adapting to Network and Client Variability via On-Demand Dynamic Distillation. Proc. Seventh Intl. Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-VII), Cambridge, MA, Oct. 1996.

[FGCB98] Armando Fox, Steven D. Gribble, Yatin Chawathe and Eric A. Brewer. *Adapting to Network and Client Variation Using Active Proxies: Lessons and Perspectives.* IEEE Personal Communications (invited submission), August 1998.

[FGC+97] Armando Fox, Steve D. Gribble, Yatin Chawathe, Eric A. Brewer, and Paul Gauthier. *Cluster-based scalable network services.* Proceedings of the Sixteenth Intl. Symposium on Operating Systems Principles (SOSP-16), St.-Malo, France, October 1997.

[FGG+98] Armando Fox, Ian Goldberg, Steven D. Gribble, David C. Lee, Anthony Polito, and Eric A. Brewer. *Experience With Top Gun Wingman, A Proxy-Based Graphical Web Browser for the USR PalmPilot.* Proc. IFIP Middleware 98, Lake District, UK, Sept. 1998.

[JK99] Zhimei Jiang and Leonard Kleinrock, “An Adaptive Pre-fetching Scheme,” to appear in Journal of Selected Areas in Communications, 1999.

[KS95] Puneet Kumar and M. Satyanarayanan, “Flexible and Safe Resolution of File Conflicts,” Proceedings of the 1995 Winter Usenix Conference, January 1995.

[KS92] Jay Kistler and M. Satyanarayanan, “Disconnected Operation in the Coda File System,” ACM Transactions on Computers, Vol. 10, No. 1, Feb. 1992.

[KS93] P. Kumar and M. Satyanarayanan, “Supporting Application-Specific Resolution in an Optimistically Replicated File System,” Proceedings of the Fourth Workshop on Workstation Operating Systems, Napa, CA, October 1993

[LHKR96] Mika Liljeberg, Heikki Helin, Markku Kojo, Kimmo Raatikainen, “Enhanced Services for World-Wide Web in Mobile WAN Environment,” University of Helsinki Computer Science Department Technical Report C-1996-28.

[MCS98] A. Mallet, J.D. Chung, J. M. Smith, “Operating System Support for Protocol Boosters,” HIPPARCH Workshop, June 1997.

[MES95] Lily Mummert, Maria Ebling, M. Satyanarayanan, "Exploiting Weak Connectivity for Mobile File Access," Symposium on Operating System Principles, December 1995.

[Met94] Metricom Inc. Ricochet wireless modem service. *http://www.ricochet.net*

[NSN+97] Brian Noble, M. Satyanarayanan, D. Narayanan, James Tilton, Jason Flinn, Kevin Walker, "Agile Application-Aware Adaptation for Mobility," Symposium on Operating System Principles, Nov. 1997.

[RHR+97] Peter Reiher, John Heidemann, David Ratner, Gregory Skinner, and Gerald Popek, "Resolving File Conflicts in the Ficus File System," Proceedings of the 1994 Summer Usenix Conference, June 1994.

[SB98] Pradeep Sudame, B. R. Badrinath, "Transformer Tunnels: A Framework for Providing Route-Specific Adaptations," Usenix Annual Technical Conference, June 1998.

[SKK+90] M. Satyanarayanan, James Kistler, Puneet Kumar, Maria Okasaki, Ellen Siegel, and David Steere, "Coda: A Highly Available File System for a Distributed Workstation Environment," *IEEE Transactions on Computers*, Vol. 39, No. 4, April 1990.

[SK97] Mark Stemm and Randy H. Katz. *Vertical handoffs in wireless overlay networks.* ACM Mobile Networking (MONET) Special Issue on Mobile Networking in the Internet, Fall 1997.

[TK96] B. Tung and Leonard Kleinrock, "Using Finite State Automata to Produce Self-Optimization and Self Control," IEEE Transactions on Parallel and Distributed Systems, Vol. 7, No. 4, April 1996.

[TW96] David Tennenhouse and David Wetherall, "Towards an Active Network Architecture," Computer Communications Review, Vol. 26, No. 2, April 1996.

[WAP97] Wireless Applications Forum home page and standards documents. http://www.wapforum.org

[YRP99] Mark Yarvis, Peter Reiher, and Gerald Popek, "Conductor: A Framework for Distributed Adaptation," Proc. Seventh Workshop on Hot Topics in Operating Systems (HotOS-VII), March 1999.

Biographies

B. Badrinath received his Ph.D from University of Massachusetts Amherst in 1989. He has since been on the faculty in the Computer Science Department at Rutgers University, where he is an Associate Professor. His research interests are in mobile and wireless computing, particularly network design issues for supporting large scale mobile and wireless nodes(e.g., sensor networks, smart spaces, dataspaces).

Armando Fox joined the Stanford faculty as an Assistant Professor in January 1999, after getting his Ph.D. from UC Berkeley as a researcher in the Daedalus wireless and mobile computing project. His research interests include the design of robust Internet-scale software infrastructure, particularly as it relates to the support of mobile and ubiquitous computing, and user interface issues related to mobile and ubiquitous computing. In previous lives, Armando received a BSEE from M.I.T. and an MSEE from the University of Illinois, and worked as a CPU architect at Intel Corp. He can be reached at fox@cs.stanford.edu. He is also an ACM member and a founder of ProxiNet, Inc. (now a division of Puma Technology), which is commercializing thin client mobile computing technology developed at UC Berkeley.

Leonard Kleinrock is known as the inventor of Internet technology, having created the basic principles of packet switching (the technology underpinning the Internet) while a graduate student at MIT. Dr. Kleinrock received his Ph.D. from MIT in 1963 and has served as a professor of computer science at the University of California, Los Angeles, since then. He received his BEE degree from CCNY in 1957 (and an Honorary Doctor of Science from CCNY in 1997). He is a co-founder of Linkabit, and also founder and chairman of Nomadix, Inc. and of Technology Transfer Institute, both hi-tech firms located in Santa Monica, California. Dr. Kleinrock has published more than 200 papers and authored six books on a wide array of subjects including packet switching networks, packet radio networks, local area networks, broadband networks and gigabit networks.

Dr. Kleinrock is a member of the National Academy of Engineering, an IEEE fellow and a founding member of the Computer Science and Telecommunications Board of the National Research Council. Among his many honors, he is the recipient of the CCNY Townsend Harris Medal, the CCNY Electrical

Engineering Award, the Marconi Award, the L.M. Ericsson Prize, the UCLA Outstanding Teacher Award, the Lanchester Prize, the ACM SIGCOMM Award, the Sigma Xi Monie Ferst Award, and the IEEE Harry Goode Award.

Gerald Popek received his Ph.D. from Harvard University in 1973. He has been a member of the faculty of the UCLA Computer Science Department since 1973. He founded the Locus Computing Corporation in 1983, and served as first as its President and CEO, later as its Chairman, until it was acquired by Platinum technologies, inc. in 1995. He served as Platinum's CTO until 1999. He is now CTO of Carsdirect.com. Dr. Popek's research has concerned security, operating systems, distributed systems, and databases. He is a member of the ACM.

Peter Reiher received his Ph.D. from UCLA in 1987. He worked at JPL for five years as principal designer of the Time Warp Operating System. He returned to UCLA in 1993, where he is an Adjunct Associate Professor. Dr. Reiher's research interests include distributed systems, adaptive technologies for networking, security, data replication, and parallel discrete event simulation. He is a member of the ACM.

M. Satyanarayanan is an experimental computer scientist who has pioneered research in the field of mobile information access. An outcome of this work is the ***Coda File System***, which provides application-transparent support for disconnected and weakly-connected operation. Key ideas from Coda have been incorporated by Microsoft into a forthcoming release of the Windows NT file system. More recently, Satyanarayanan and his research group have been working on application-aware adaptation, a more general approach to mobile information access. This concept is being explored in the context of a new platform, ***Odyssey***. Prior to his work on Coda and Odyssey, Satyanarayanan was a principal architect and implementer of the ***Andrew File System***, a location-transparent distributed Unix file system that addressed issues of scale and security. Later versions of this system have been commercialized and incorporated into the Open Software Foundation's DCE offering.

Satyanarayanan is the ***Carnegie Group Professor of Computer Science*** at Carnegie Mellon University. He received the PhD in Computer Science from Carnegie Mellon, after Bachelor's and Master's degrees from the Indian Institute of Technology, Madras. He has been a consultant and advisor to many industrial and governmental organizations.