Conductor: Enabling Distributed Adaptation[†]

Mark Yarvis, Peter Reiher, Kevin Eustice, Gerald J. Popek
{yarvis, reiher, kfe, popek}@fmg.cs.ucla.edu

University of California, Los Angeles
Tech Report: CSD-TR-010025

Abstract—**Varying and suboptimal network conditions require applications to degrade gracefully to meet a user's needs. Existing technologies usually overcome a single network deficiency with one adaptation, often near the last mile of the network. These solutions are insufficient in the increasingly heterogeneous Internet, where applications will benefit from coordinated distributed adaptation at various points in the network. We describe the challenges of distributed adaptation and the Conductor adaptation service that meets these challenges, providing potentially dramatic improvements.**

1 Introduction

Advancing network technologies increase the interconnectivity and heterogeneity of computer networks. The average bandwidth between any two points in the Internet is going up, but so is the range of experienced bandwidth. Applications that cannot adapt to varying network conditions will become decreasingly useful, particularly to mobile users. One solution is to deploy a distributed adaptive service into nodes in the network. These nodes can provide processing and/or storage capabilities and operate on higher protocol stack layers.

Historically, transport (and higher) protocol layers have been provided only at endpoints. Building knowledge of higher-layer protocols into routers has several disadvantages, including performance and reliability. While this paper does not prove that the advantages outweigh the disadvantages, it provides evidence from two perspectives to support raising the level of network services. First, we explore common cases in which application-level adaptation within the network helps applications adapt their services under suboptimal conditions. Second, we describe technologies that allow a network operating system to provide sensible and reliable adaptive services.

Successfully enabling distributed adaptation requires several important services. If adaptations may add, delete, or modify data stream elements, the typical model of exactly-once delivery is no longer suitable. A new model for reliable delivery of adapted data must be developed. In addition, a security model is required that balances the needs for secrecy and adaptation. These and other challenges must be met. Doing so may require changes to the operating system or support from operating system services. Further, many of the problems encountered are similar to certain problems in distributed operating systems.

We have constructed an adaptation framework called Conductor to explore distributed adaptation. Conductor provides an application-transparent adaptation service within the network. Adaptor code modules, dynamically deployed as needed at Conductor-enabled points within the network, adapt data streams as required. Conductor allows multiple coordinated adaptations to be deployed without compromising reliability. Conductor's techniques enable applications to be adaptable in heterogeneous networks.

2 Distributed Adaptation

While the Internet's overall capacity is increasing, high performance networking is far from ubiquitous. Congestion, failures, mobility, and the "last mile" create orders of magnitude differences in bandwidth, latency, jitter, security, cost, and reliability. At the same time, general-purpose software, multimedia applications, thin-client software, PDAs, and other Internet appliances increasingly demand connectivity.

Internet applications typically require network characteristics to meet a minimum threshold for adequate performance. When not met, an application's cost in time, security, or money may exceed its value, perhaps providing no useful service. The service provided by an application should degrade gracefully to match the network's capabilities.

Graceful degradation can be provided outside an application by adapting its communication stream. Common examples are link-level compression in PPP and ssh tunneling to add encryption. Proxy nodes within the network can perform many services. Application-specific adaptations, such as dynamically reducing the color depth or resolution of images, can greatly reduce costs with an acceptable loss of quality [6]. Performing such adaptations externally to applications frees developers from having to predict all possible combinations of network conditions an application will face and may allow the user to choose from a wider range of adaptations.

Research on adaptation within a network has primarily focused on individual adaptations for individual problems. Placing proxy nodes adjacent to a last-mile link, for instance, allows adaptive performance over that link. Several simultaneous link deficiencies, however, require careful coordination between multiple adaptations. For example, securing a connection by tunneling through ssh across a modem link renders PPP compression ineffective.

Heterogeneous networks require system support for *distributed adaptation*—multiple adaptations, potentially distributed within the network, that are carefully coordinated for

[†] This work was partially supported by DARPA under contracts numbered DABT63-94-C-00080 and N66001-98-C-8512.

proper end-to-end adaptation. The following sections describe three case studies that require distributed adaptation.

2.1 Secure Communication From a Home LAN

The home of the near future may contain several Internet-capable devices (multiple workstations, TV, radio, refrigerator, etc.), possibly connected by a wireless LAN for internal communication with a broadband router providing Internet access for all devices on the LAN (Figure 1).

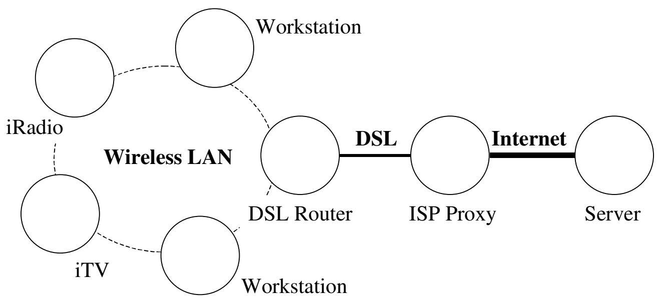

Figure 1: A home network supporting multiple Internet clients.

Web access to a user's bank account in this environment will pose several concerns: the insecure Internet and wireless LAN and insufficient bandwidth in the shared DSL link. Data could be adapted at several points: the client, the server, the DSL router, and a "proxy" node provided by the ISP.

A client can improve the performance of web fetches (several of which typically occur at once) by prioritizing data transfer across the DSL link. Priority might be established by size, allowing smaller images and text pages to be received first and preventing slowdowns caused by large software and document downloads. Or priority might be established by type; text data might have priority over images. Prioritization should be deployed at the ISP proxy, allowing HTTP sessions with various servers to be prioritized together.

All of the data from the bank's web pages should be encrypted, reducing the chance of sensitive information being transmitted in the clear. A software download, which may be occurring simultaneously, might also require encryption. End-to-end encryption of the data stream, while simple, would obscure the data, preventing prioritization of the interactive traffic over the software download at the ISP proxy.

Two solutions are possible. An end-to-end encryption adaptation that tags each session with its length (or type) could replace a generic encryption adaptor. Or encryption could be performed twice, once server-to-proxy and again proxy-to-client, with prioritization based on the temporarily decrypted data. In each solution, available bandwidth has changed the manner in which encryption should be applied.

2.2 Power Savings for Mobile Users

The network in Figure 1 is similar to a mobile-deployed wireless network that might be used by an emergency rescue team or an archeology team in the field. Simply replace the DSL link with a point-to-point wireless link and the DSL router with a wireless access point. While the bandwidth of the Internet connection will still be low (perhaps even lower), and security may still be of concern, a new concern is the limited operating power of the devices on the wireless LAN.

If a client on the wireless LAN performs a high-latency database query, it will expend power listening for an asynchronous response. A useful adaptation would allow the client to power down its wireless interface while the query is outstanding. For each request, the client could compute the expected time for completion and notify the access router that it will be offline for that length of time. The access router would cache any results that are returned in the meantime, transmitting them after the specified time has expired.

A compressor deployed either at the server or the ISP's proxy can reduce the transfer time across the low-bandwidth link. Decompression is best done at the mobile access point, reducing the workload on the power-limited mobile device.

2.3 A Multimedia Session Between Wireless Users

Mobile users often have poor last-mile links. Increasingly, mobile users will want to communicate with each other, leading to multiple poor links. For instance, a mobile-to-mobile video-conference has two low-quality last-mile links.

Consider two mobile users using WaveLAN packet radios that provide around 6 Mb/s of bandwidth (Figure 2). The wireless link has insufficient bandwidth to carry the video transmission from Client 1 to Client 2. The data could be compressed before transmission over the WaveLAN links (at Client 1 and Proxy 2) and decompressed upon receipt (at Proxy 1 and Client 2). To avoid inefficient repeated compression and decompression, we can adapt from end to end, compressing at Client 1 and decompressing at Client 2.

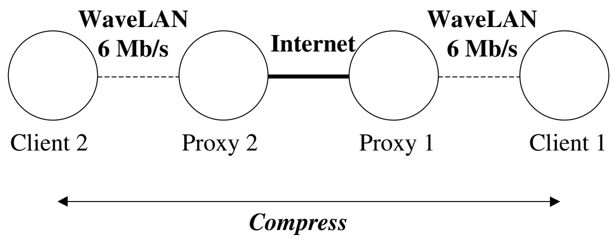

Figure 2: Two mobile users with WaveLAN connectivity.

If Client 2 is, instead, connected via a Metricom Ricochet-2 wireless modem [19], with 128 Kb/s bandwidth, compression is insufficient (Figure 3). Instead, a significant number of frames must be dropped, preferably at Client 1. The resulting stream will be small enough to allow transmission over both links without requiring further compression.

The correct adaptation changes again if a video clip is transmitted instead. Perhaps a cache is present somewhere in the Internet that could service other requests for the same clip without requiring further data transmission over Client 1's WaveLAN network. If the previous adaptation scheme were used, the cache would receive a reduced-fidelity version of the video, which might not meet the needs of other clients. In this case, it may be preferable to perform lossless compres-

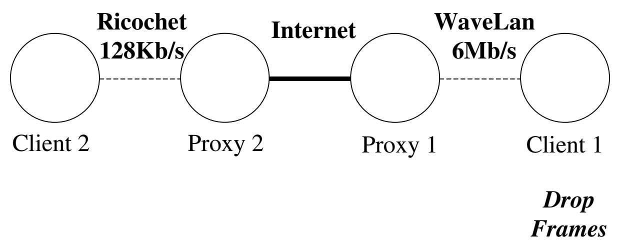

Figure 3: One mobile user connected by WaveLAN and one connected by Ricochet.

sion between Client 1 and Proxy 1 and drop frames at Proxy 2, delivering the full-fidelity data stream to the cache.

2.4 Discussion of Case Studies

The above case studies are real-world situations in which multiple network deficiencies exist at various points in the network. These cases depict two important characteristics:

• Adaptation may be required at several points in the network. In the web prioritization case, prioritization must occur at the ISP proxy (to prioritize all streams crossing the DSL link), while encryption is required at the client, server, and perhaps at an intermediate point. In the mobile power-saving example, result caching must occur at the wireless access point, control of the mobile device's wireless interface must occur on that device, and compression must occur at or before the edge of the high-bandwidth network.

• Adaptations to handle independent network problems may require coordination. In the web prioritization example, application-level end-to-end encryption precludes prioritization. In the user-to-user multimedia example, independent adaptation can lead to redundant and wasteful operations.

These examples are not isolated. Wireless home networks, personal-area networks (e.g., Bluetooth [10]), and multi-hop wireless networks [19] are extending the last mile. At the other extreme, even the most robust and powerful servers have succumbed to congestion, distributed denial-of-service attacks, and unexpected load peaks (e.g., the "Slashdot effect" [1]). Heterogeneity constrains how adaptations can be deployed. If a client adjacent to a low bandwidth link needs prefetching to handle the high server latency, prefetching from the client would overload the low bandwidth link. On the other hand, while an end-to-end compression adaptation might handle several bandwidth constraints, a given server may have insufficient CPU cycles to provide such an adaptation to all clients, requiring other possible points of adaptation to balance the load.

These cases demonstrate that end-to-end network heterogeneity may require a coordinated set of adaptations at multiple points in the network. Such a service requires more complexity than independent application-, link-, or proxy-level adaptation.

3 Providing Distributed Adaptation

Applications communicating across heterogeneous networks, including the Internet, may require multiple coordinated and potentially distributed adaptations, something not well supported by existing adaptation frameworks. We have developed a systems service called Conductor to explore the challenges and rewards of distributed adaptation.

3.1 The Conductor Approach

Like several existing systems, Conductor enables adaptation of reliable connection-oriented streams from within the network. However, Conductor allows adaptation to be distributed into the network dynamically, applying several coordinated adaptations to a single connection at various points along that connection path.

Conductor is incrementally deployable. It can be deployed on a subset of nodes within the network—ideally at clients, servers, and gateways between networks of differing characteristics. The additional resources required by adaptation (notably processing and storage) can be deployed as needed. The more resources present, the greater the service provided. Conductor-enabled nodes might be provided by an organization's network infrastructure, by an ISP (for use by subscribers), or by a third party on a fee-for-service basis [24].

Conductor nodes support the deployment of adaptor code modules that implement particular adaptations. These adaptors operate on application-level protocols, arbitrarily modifying the data stream. Conductor includes a planning infrastructure that selects the appropriate adaptors for the given conditions. The planning process is user-controllable via a set of user preferences.

Conductor ensures that the data stream delivered to the destination application is in a usable form, allowing application-transparent adaptation. However, the data delivered may be different from the data transmitted. Conductor is application-transparent, but not user-transparent. For instance, an adaptor may reduce the bandwidth requirement of a video transmission by dropping some frames, effectively reducing the frame rate. If necessary, another adaptor can restore the original frame rate before delivery by duplicating frames, filling the gaps. The application will accept the video stream without noticing any changes, but the user will see a qualitative difference. Since Conductor is fundamentally application-transparent, it can support both legacy and closed-source applications. However, an API could also be added, allowing aware applications to provide additional input and control over planning and adaptation.

3.2 Key Challenges

Conductor's approach to adaptation introduces several key challenges, including resilience to failures, the ability to choose an appropriate set of adaptors, and protection from attack and misuse.

The new components that distributed adaptation introduces into the network must not reduce the reliability of data connections. Node, link, and adaptor failures should not cause connection failures. Today, reliability is typically provided end-to-end and a given connection's reliability does not depend on any particular nodes within the network. This reliability model is insufficient to support distributed adaptation. First, distributed adaptation introduces unique and potentially stateful nodes in the middle of the network upon which a connection may depend. Second, adaptation may modify the data stream in transit, potentially confusing an end-to-end reliability service. Third, this reliability model offers no protection against changes in adaptation, particularly unexpected changes, which must not result in the delivery of unintelligible data to the user. For example, an adaptation that reduces bandwidth requirements by dropping video frames provides intentional data loss, which should not be considered a failure. Also, adaptor failure should not break the data semantics by delivering half of a frame. Conductor requires a new model of reliability that is compatible with adaptation

For each connection that may need adaptation, a distributed adaptation service must determine which, if any, adaptors are required and where to deploy them. Planning should be automatic, requiring as little user and application participation as possible. If data flows before adaptors are deployed, resources could be wasted on low-priority data, or sensitive data could be transmitted unprotected over insecure links. So planning must be quick. This point favors decentralized planning, in which each node locally selects and deploys the adaptors it believes are necessary. But the case studies in Section 2 show that the right choice of adaptation for one link may be influenced by other links and other adaptors desired for the connection. A global view, and therefore inter-node communication, is required to make the correct global decisions. Since some links may be slow, we must minimize the communication cost of obtaining a global view. However, mistakes in planning are expensive, so Conductor must balance plan quality against startup latency.

Since end-to-end encryption is incompatible with many desirable adaptations, and link-level encryption assumes that all routers are trusted, Conductor must provide an intermediate approach that adapts only at trusted nodes. The planning process itself must be protected from subversion. Input to the planning process must be limited to trusted nodes, and adaptor deployment must be triggered only by an authentic plan. Finally, each adapting host must be protected from the adaptations it allows, including overuse or misuse of resources.

3.3 Potential Pitfalls of Distributed Adaptation

Historically, packet processing at network routers has been limited to network-level protocols. Usually, higher-level functionality is best provided as close to the application endpoints as possible [25], though special systems with visibility into higher-level protocols have been constructed, both for adaptive purposes [3] and in firewalls for security. The advantages of application-level adaptation must be balanced against packet overhead and transparency.

Building application-level adaptation into network nodes increases packet processing costs. Fortunately, adaptation is not required at all network nodes, nor by all packets. For packets requiring adaptation, the processing costs should be offset by the benefits obtained. For instance, the time it takes to reduce the color depth of a video frame is made up by lower transmission cost. Many dynamic application-level adaptations are feasible today [6]. Since processing speed is improving faster than network speed, the set of feasible adaptations is growing. Also, specialized hardware may enable efficient active processing at network nodes [11].

Adaptation removes the transparency of the transmission channel. Unaware adaptation might not always be appropriate. In the Conductor model, the user is in control of adaptation and can therefore select the desired application behavior.

4 The Conductor Architecture

Conductor is composed of two main elements: adaptors and the framework for deploying those adaptors (Figure 4).

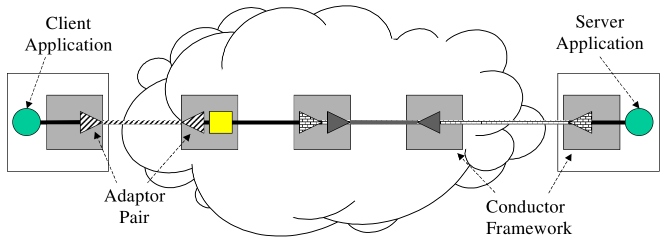

Figure 4: Conductor intercepts client-server communication channels and deploys distributed adaptors.

Conductor adaptors are self-contained pieces of Java code. Many adaptors are type-specific, expecting a specific protocol or data type. Most adaptors are paired. A pair of adaptors typically converts from an input protocol to a protocol better suited for transmission over a network with particular characteristics, and back to the original protocol. Adaptors can arbitrarily modify the data stream, allowing any desired type of adaptation. Adaptors can be lossy; the data delivered to the application may be different from the data transmitted.

Newly developed adaptors can be added to the available suite as new protocols, new network technologies, and new user requirements are developed. Adaptors are dynamically deployed for new connections as the need arises, limited only by the availability of node resources. Adaptors can be composed and combined sequentially with other adaptors, limited only by the input and output protocols expected by each adaptor. Adaptors are self-descriptive, specifying the required input protocol, the resulting output protocol, and the resources required from the node.

The Conductor framework is primarily a user space Java program that provides a runtime environment for adaptors (Figure 5). This runtime environment intercepts connections from application clients to application servers, forming a data path through Conductor-enabled nodes between the client and server, and deploying adaptors at appropriate nodes along that path. The protocols used by Conductor to enable distributed adaptation are described in the following sections.

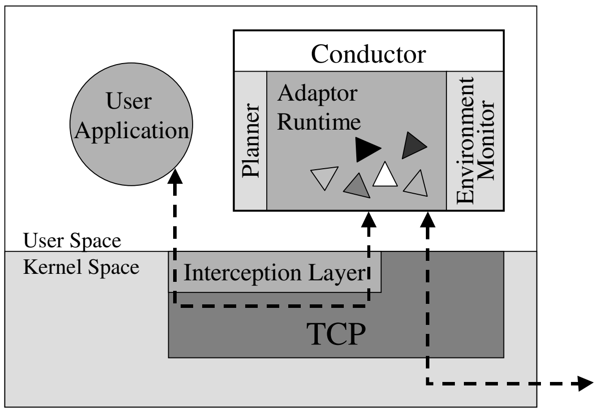

Figure 5: The architecture of a Conductor-enabled node.

4.1 Stream Interception and Handling

Before it can adapt a data stream, Conductor must intercept that stream and determine if the data is a type that Conductor can adapt. If so, Conductor must find Conductor-enabled nodes between the client and the server, and provide each with sustained access to the data stream.

When an application starts a new data connection on a Conductor node, Conductor's interception layer traps the opening of the socket, connecting it to Conductor instead of the opposite endpoint. Since Conductor is connection-oriented, only TCP connections are intercepted. Under the Linux 2.2 kernel, the interception layer is a small kernel modification that traps connections destined for particular remote ports specified by Conductor. Under Linux 2.4 and modern FreeBSD kernels, the built-in firewall can perform this function. In other systems, extensibility mechanisms can allow trapping of data flows without kernel modifications [18]. If the client node is not Conductor-enabled, Conductor could also trap connections from a node within the network using Linux's transparent proxy facility [29].

To properly adapt a data stream, Conductor must correctly identify protocols and data types. Such identification is possible because client and server applications must already provide sufficient clues to communicate with one another. For example, a well-known port number is used to contact a particular service. Some protocols, like HTTP, may use several port numbers or may have several protocol versions. In these cases, an initial handshake that identifies a protocol version or enabled options is common. The type of data being transmitted via a transport protocol like FTP, SMTP, or HTTP is frequently identified explicitly (using the "Content-Type:" header in HTTP), but magic numbers can also be used. Since Conductor only intercepts those protocols that it knows can be effectively adapted, we reduce the overhead for non-adapted streams. Currently, interception is based entirely on remote port number.

Once a connection is intercepted, Conductor can identify (from the interception layer or transparent proxy facility) the connection's original destination. The intercepting node must identify other Conductor-enabled nodes along the path from the client to the server. Currently, Conductor-enabled nodes are identified by sending probe packets (which follow the typical network route) toward the server. Conductor-enabled nodes along this route capture this packet and participate in planning for this connection. Once the last node is reached, a connection is created to the intended server, and planning occurs to select a set of adaptation nodes and a set of adaptors to deploy. Before data flows, TCP connections are created between adjacent pairs of Conductor nodes at which adaptation is desired. The result is an end-to-end path made up of multiple split-TCP connections.

4.2 Automated Planning

Conductor includes a planning facility to automatically select an appropriate set of adaptors and decide where to deploy them. Automatic planning is essential because both the user and the application writer typically lack the networking expertise and forethought required to understand either the characteristics and requirements of the network or the capability and interoperability of available adaptors. Planning occurs on a per-connection basis and consists of three main elements: information gathering, plan formulation, and adaptor deployment.

The planning process considers user preferences, node resources, and link characteristics. User preferences describe the resources (time, money, battery power, etc.) and the qualities of the data most important to the user's current task. For instance, when downloading a map over a limited bandwidth link, an impatient user may prefer a high-resolution black-and-white image, allowing street names to be read, or he may prefer a low-resolution full-color image, allowing the parks to be easily located. Node resources include the set of available adaptors and the resources available to those adaptors, such as CPU cycles and storage space. Nodes may also wish to express security constraints that would prevent them from executing particular adaptors. Link characteristics include bandwidth, latency, security level, jitter, and reliability.

Determining the inputs to planning requires some facility for environmental monitoring at each node. Conductor provides a pluggable architecture for monitoring local node and link conditions. An environmental monitor should allow current and historical conditions to be queried and should notify Conductor of drastic changes that might require a revised plan. Currently, Conductor uses a very simple mechanism for measuring node resources and link characteristics. Eventually, Conductor could leverage more sophisticated technologies developed by other researchers [26] [34].

Conductor balances global optimality against speed by using a single-round-trip centralized planning architecture. When planning begins, all nodes along the data path forward their local planning information to a single planner node (Figure 6). Currently, this node is the Conductor node closest to the server. The planner node uses this information to formulate a plan, which is transmitted from the planner node toward the client endpoint, passing through each Conductor-enabled node along the way. Once the adaptors are deployed

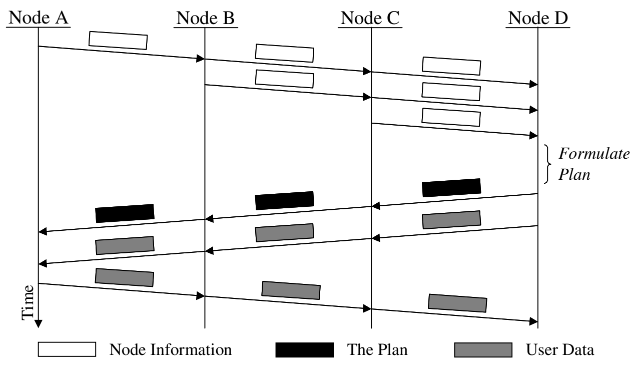

Figure 6: Conductor's planning protocol in action.

on a given node, data may flow in that direction. As a result, once the plan reaches the client node, the first bytes of application data from the server will reach the client. The plan now flows in reverse, deploying adaptors for the other direction of flow. Once the plan reaches the server node, the first bytes of application data from the client will reach the server. Thus, one extra round trip has been added before application data begins to flow in both directions.

The Conductor architecture can plug in many plan formulation algorithms; the current algorithm is template-based. A template is a pre-formulated description of a hypothetical deployment of adaptors. Templates are frequently (but not always) protocol-specific and describe a hypothetical set of nodes, the adaptors to be deployed on those nodes, the resources required on each node, the effect of adaptation on the links between adjacent nodes, and any qualitative change in the delivered data. For instance, a template might specify that using a frame-dropping adaptor on one node and a frame-duplicating adaptor on another doubles the effective bandwidth of the links between them, but reduces video quality. Since the number of Conductor-enabled nodes in the data path is typically greater than the number of nodes specified in the template, a given template can be applied to a network in many ways (by matching each template node to each real node). The best application of all available templates is chosen based on the effective network characteristics desired by the user and the preferred resulting data characteristics.

Planning occurs at connection start-up time based on the current network conditions. Dynamic network conditions may lead to replanning. Adapting to minor variations in bandwidth, delay, etc., is the job of the individual adaptors. If, however, the variations are too large for the adaptors to handle, or there is an actual failure, one Conductor node will signal to the others, initiating a distributed reevaluation of the deployed adaptors. Conductor may try to find a new data path or alter the set of deployed adaptors on the old path.

4.3 Reliability

TCP provides a convenient model for application writers: exactly-once, in-order delivery of a byte stream. However, this model is incompatible with adaptation. Since adaptation seeks to deliver some version of the transmitted data that is cost-effective, the bytes delivered may bear no resemblance to the bytes transmitted. To maintain the semantics expected by the user while enabling adaptation, Conductor provides a new model of reliability that instead provides exactly-once, in-order delivery of *adapted* data.

4.3.1 Previously Proposed Approaches

Three approaches to reliability have been incorporated into previous adaptive services. The first is to restrict the types of operations that can be performed on the data stream. The Snoop Protocol [3] avoids violating TCP semantics, while Protocol Boosters [5] allows only additive operations, transmitting the additional information independently from TCP streams. In either case, adaptation failure only reduces or removes the adaptation benefits. This approach cannot support arbitrary adaptations, however.

A second approach is to increase the reliability of adapting nodes. The Berkeley Proxy [7] allows arbitrary adaptations by splitting the TCP channel into two, from the server to a proxy and from the proxy to the client. The potential for failure is mitigated through the use of redundant hardware and software in the proxy node. While this approach works for a single proxy near the last mile, deploying redundant hardware throughout the network is not feasible. Also, this approach does not address software failures in adaptation modules.

A third approach is to bypass the end-to-end reliability mechanism (using split-TCP or by adapting TCP itself [15]) and accept some failures. This approach is particularly appealing when only one path exists from the client to the rest of the network, as is commonly the case for last-mile adaptation. If the point of adaptation is along this path, then failure of an adapting node implies packets cannot be forwarded. Since some degree of failure would occur anyway, the presence of adaptation does not decrease the level of reliability. However, this argument does not address the issue of adaptor failure or more transient node failures. Also, when adaptation is distributed, alternative routes may very well exist, allowing data transmission to continue despite node failures.

4.3.2 A New Model for Reliability

Most reliability models assume that data is immutable during transmission. Each byte transmitted is received, unchanged, at the destination. The measure of reliability in such a system is exactly-once, in-order delivery of bytes. This type of reliability can be guaranteed by the endpoints, the failure of which will cause the connection to fail.

Adaptation violates this model's assumption of data immutability. When a failure occurs, the system must ensure that the resulting adapted data stream conforms to the protocol expected by the application. Maintaining data integrity is not simple, however. The data already output by an adaptor may be incompatible with a switch to the original stream. For instance, consider an adaptor that adds a *lowsrc* attribute to an *image* tag in an HTML data stream (Figure 7). If a failure occurs in the middle of the tag, a byte-count retransmission

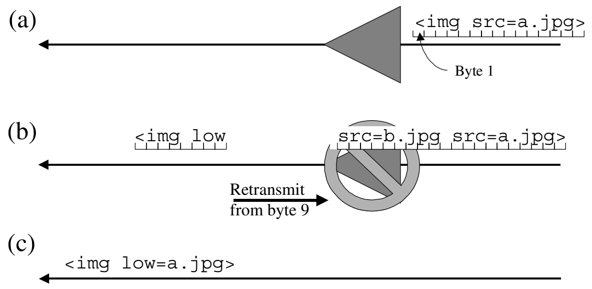

Figure 7: Failure recovery using a byte-count scheme – (a) data arrives at adaptor, (b) failure and retransmission occur, (c) retransmission produces an undesirable result.

scheme may result in data that is neither the adapted data nor the original data, and may not even be syntactically correct.

Even if an adaptor fails at an appropriate point in the stream, retransmission of lost data is still difficult. For instance, if an adaptor converts color video frames into black-and-white, the adapted frames will constitute a shorter byte stream. If a failure occurs immediately after the last byte of frame 100, a simple byte-count retransmission scheme might begin retransmission after frame 50, duplicating data the user has already received. The system must correlate the data received downstream with the data transmitted to determine an appropriate point of retransmission. If this correlation is lost with the adaptor, transmission cannot continue.

We propose that in the face of adaptation, a reliable system should preserve two properties of a data stream:

1. Each semantically meaningful element in the transmitted data stream is delivered exactly once and in order.
2. Delivered data conforms to the expected protocol.

The first property requires that the data stream be divided into segments that are semantically meaningful to the protocol being transmitted before adaptation can occur. For instance, a video stream might be divided into frames, or an HTML stream into tags and text. Each segment must be delivered exactly once, in some acceptable form (original, adapted, or deleted entirely) in the proper order.

The second property restricts the content of the segments at the time of final delivery. If halving the frame rate is acceptable to the application, then delivering empty segments in the place of every other frame might be allowed. Segments must be chosen so that the failure of any adaptor will not result in delivery of data that confuses the destination application, thereby ensuring that some viable version of the data produced at the source will arrive at the destination.

4.3.3 Attaining Reliability

Conductor uses a TCP connection between adjacent Conductor nodes along the data path, providing reliable delivery between adaptor modules on different nodes. Since adaptations occur only at Conductor nodes, TCP's model works well between them. Unless an adaptor, a link, or a node fails, end-to-end transmission of adapted data proceeds reliably and in order. If one of these events occurs, Conductor's data recovery mechanism protects against data loss. Once the data path is restored, Conductor must determine which data has already been received downstream and request retransmission from this point. As previously described, this mechanism must be compatible with adaptation, providing exactly-once delivery of semantic meaning.

4.3.4 Semantic Segmentation

Semantic segmentation allows data recovery despite the presence of adaptation. A semantic segment is a dynamically adjustable unit of retransmission for data recovery. Semantic segments also preserve the correspondence between an adaptor's input and output data streams. Adaptors, which have an understanding of the format of the data stream and the operations they will perform on that stream, have a responsibility to maintain appropriate segmentation.

The initial data stream consists of bytes being transmitted by the application and intercepted by a Conductor module on the source node. Conductor considers these bytes to be logically divided into one-byte segments, which are numbered sequentially. It is not necessary, at this stage, to track segment boundaries or segment numbers. For efficiency, the bytes can be transmitted with very little overhead; simply counting the bytes can identify individual one-byte segments.

Adaptors form larger segments by combining smaller segments. When segments are combined, the new segment receives the segment ID of the last combined segment. When operating on the data stream, adaptors must perform segment combination in two situations:

1. When modifying a semantic element in the data stream that crosses a segment boundary
2. When adaptor failure between segments could otherwise violate the expected protocol

Consider an adaptor that compresses video frames. Before compressing a frame, the segments making up that frame are combined into one segment. If the stream consisted of 100-byte frames, each frame would initially be represented in the stream as 100 1-byte segments. Before reducing each frame to 50 bytes, the adaptor would combine these 100 segments. The first 100-byte frame would be in segment 100, and the second would be in segment 200. Each segment would then be adapted, producing a 50-byte segment numbered 100 and another numbered 200. Each resulting 50-byte segment contains the same semantic content as the 100-byte segment and the 100 1-byte segments; only the format has changed.

Subsequent adaptors may further combine segments, and segments may grow to arbitrary length. In general, to support lossy adaptation, combined segments can never be taken apart. At the destination node, the Conductor module simply removes any segment markers and delivers the resulting data to the application (with one restriction, described below).

4.3.5 The Recovery Protocol

Conductor uses the semantic segment as the unit of retransmission. To allow retransmission, data transmitted from each endpoint is cached at the Conductor node closest to the source. If the source node is Conductor-enabled, we depend on it not to fail, just as TCP does. To improve the speed of recovery, caching can also be added at other points along the data stream.

Recovery is initiated downstream of the failure. Any segment that has been partially received is cancelled and discarded. Note that if the application is unaware of the adaptation system, the possibility of cancellation of partial segments requires that the adaptation system not deliver any segment to the application until that segment is complete. Once cancellation is complete, the ID of the previous completely received segment will be known.

A retransmission request containing the ID of the last segment received travels upstream until it can be serviced, either by a cache or by an adaptor (perhaps from an internal cache). Nodes that cannot satisfy the retransmission request will forward the request upstream and discard any subsequent segments until retransmission begins, preserving in-order delivery and ensuring that any resulting change in adaptation is observed in all subsequent data. The mandatory cache at the source node provides a fallback source if retransmission does not occur prior to this point. Once a source for the requested segment is found, transmission begins with that segment and proceeds in order with the following segments. Note that the possibility of retransmission requires adaptors to accept a rollback to a previous point in the data stream, or fail. Since semantic segmentation ensures semantic equivalency of data, retransmission can occur with any version of the desired segment, including the original version. The data can then be re-adapted in the same way, or perhaps differently.

Continuing the video example from the previous section, consider the case where the first frame (in segment 100) and part of the second frame (in segment 200) are received at a point immediately downstream from the decompression adaptor (Figure 8). If both the compression and decompression adaptors fail, Conductor will discard the part of segment 200 that was partially received downstream and request retransmission starting at segment 101. Retransmission will thus begin with the second frame, as desired. If only one of the

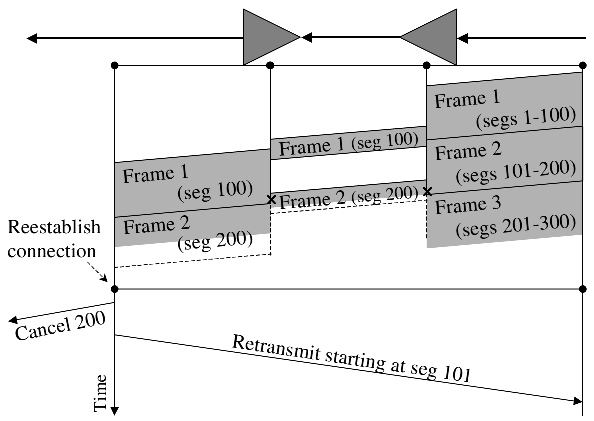

Figure 8: Recovery from the failure of an adaptor that compresses frames in a video stream.

two adaptors fail, Conductor will automatically remove the other to preserve adaptation symmetry.

This scheme provides failure-based recovery. Like any system based on negative acknowledgements, Conductor also needs to limit cache growth and allow adaptors to free any accumulated internal state. For this reason, the destination node generates acknowledgements whenever a segment is completely received. Acknowledgements are cumulative, allowing all composed segments to be acknowledged when a segment is finally received at the destination.

More details of Conductor's reliability mechanisms, including the ability to restore proper pairing of adaptors after a failure, are available in [32].

4.4 Adaptor API

An adaptor operates on one direction of data flow. Each adaptor has its own thread of execution, a window into the data stream, and access to an inter-adaptor communication facility.

Each adaptor accesses the data stream through a *window*, which defines a portion of the data stream to which it has exclusive access. The flow of data is controlled by moving the window boundaries along the stream. An adaptor can read new data into the window using an `expand()` operation. The `expand()` operation blocks until the requested data is available. An adaptor can write data out of the window using the `contract()` operation.

An adaptor is also able to view and modify the data stream by other window operations, controlling segmentation in the process. An adaptor can freely read the bytes within the window. Segmentation will not be affected. To modify the data stream, an adaptor can use the `replace()` operation, which replaces a portion of the data in the window with a new set of bytes. The data being replaced may belong to several adjacent segments, which will be automatically combined into one segment and labeled appropriately. Once contained within a single segment, the old data can be removed and replaced with the new data.

A malfunctioning adaptor can provide incorrect data to the user, but it cannot violate the rules of segmentation. However, an adaptor must ensure that segments are semantically meaningful and delineates appropriate places for adaptation to cease, or a failure may result in a meaningless data stream.

Additional API elements allow adaptors to leverage existing Java code. An adaptor can read a byte stream from the window using an `InputStream` interface and write a byte stream to the window using an `OuputStream` interface. Existing components that implement standard algorithms, such as encryption or compression, can alter the stream using this standard API. Since the operations performed in such a filter are hidden from Conductor, however, the resulting stream will be one large semantic segment.

Finally, adaptors have access to inter-adaptor communication facilities, one for communication between adaptors operating on the same stream, the other for communication between any adaptors on the same node.

4.5 Securing Distributed Adaptation

Secrecy and integrity of transmitted data is of concern in any distributed system. Providing distributed adaptation introduces two additional concerns. The first concern, protecting nodes from malicious adaptor code, is well known and can be addressed by techniques such as sandboxing, signed code, or the results of other ongoing research [2] [8]. Conductor's security design instead focuses on the relatively new issue of protecting the user from undesired adaptation.

Conductor provides a security architecture that, for a given connection, discovers a set of nodes that can be trusted, ensures that only those nodes participate in the selection of adaptations, and (when necessary) provides encryption over the "virtual links" between trusted nodes.

4.5.1 Establishing Trust

Distributed adaptation can require one or more nodes in the network to act on the user's behalf. Conductor must determine which nodes the user trusts. A user can have different levels of trust toward a given network node. One node might be trusted to adapt the plaintext of a data stream. Another node might only be trusted to adapt encrypted data. A third node may not be trusted at all.

We assume that the endpoints of each connection are fully trusted by the user (to access a given stream), as is typically the case. When this is not the case, the application can employ end-to-end encryption. One or both of the trusted endpoints must determine which other nodes can also be trusted. Establishing trust requires two elements: an authentication mechanism to establish the identity of participating nodes, and a model for determining which nodes are trustworthy.

Conductor currently uses a static trust model. The user can specify a list of trustworthy nodes or networks. Future versions of Conductor may include more flexible and dynamic models of trust, perhaps leveraging an automated trust management system such as Keynote [4]. Conductor selects the set of trusted nodes on the server endpoint as part of the planning process. While the client endpoint could also participate, performing selection on the server reduces the number of round trips required.

4.5.2 Authentication

Authentication prevents an untrusted node from masquerading as a trusted one. Since there is no ubiquitous Internet infrastructure for authentication, and because different applications need different strengths of authentication, a pluggable authentication architecture is needed, allowing the user to choose a suitable authentication mechanism for each stream.

Conductor currently provides several authentication modules, which assume different amounts of network infrastructure and provide different levels of security. A *null*-authentication module is used when no authentication, and therefore no security, is required. The *tree* module assumes that a hierarchy of certificate authorities (CAs) provides authenticatable public keys. A third module, dubbed *chain*, is also based on public keys, but assumes that administrative

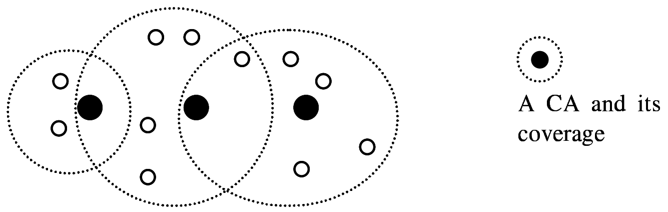

Figure 9: A network with loosely related certificate authorities (suitable for forming a chain of trust).

domains have independent, rather than hierarchical, CAs (Figure 9). Assuming that each local CA can certify some of the "neighbor" domain CAs (and perhaps some distant nodes as well), then a chain of trust may be formed.

The public and private keys managed in the chain and tree modules will be used to sign and encrypt planning messages (as described in the following sections). Conductor allows other authentication modules to be constructed.

Figure 10 depicts the secured planning protocol. Authentication begins at connection startup, before planning begins. The client node chooses an authentication scheme and sends a message from node to node to the server.

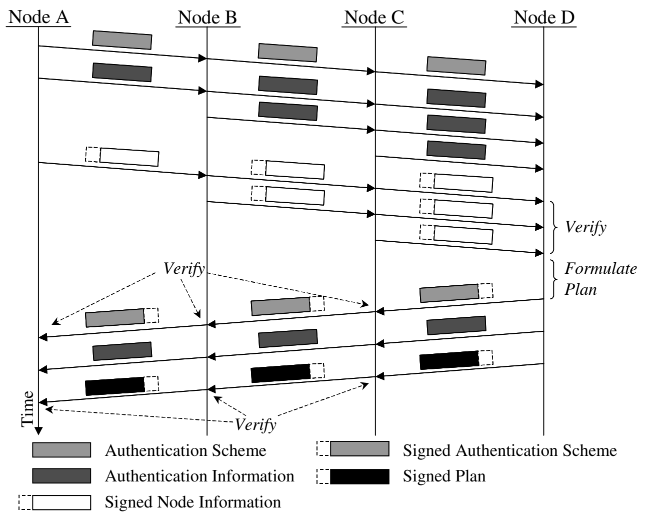

Figure 10: Conductor's secured planning protocol.

Since the server node determines which nodes will be trusted, each node must authenticate itself to the server node. To this end, each node sends an authentication message, which is carried from node to node, to the server. The contents of this message depend on the type of authentication requested. For instance, if the tree module is used, one public key certificate from each level of the hierarchy is included in each authentication message. Given the public key of the root server, which is known by all nodes, an authenticated public key for the node in question can be obtained by the planner.

Independent authentication between each node and the server is insufficient, unlike point-to-point authentication services like IPsec [14]. The chain module, for example, will create an authentication message containing a certificate for the local node from the local CA. However, it will also examine all authentication messages received, before forwarding them toward the server. The chain module will attempt to obtain certificates for each node and each CA mentioned in the message from its local CA. The authentication message received by the server may include a chain of certificates starting with the certificate for the public key of the node in question and ending with a certificate signed by the local CA.

Some authentication mechanisms will also require the server to provide authenticating information to the client. For instance, in both of the public key authentication modules, each node will also need to obtain the public key of the server. In this case, a single authentication message is also sent in the reverse direction, from the server through each node to the client.

Since multiple authentication mechanisms are supported, Conductor must ensure that each node uses the same mechanism to authenticate other nodes. The resulting chicken-and-egg problem of what mechanism to use to establish the common authentication mechanism must also be solved. To this end, a message identifying the type of authentication actually used to authenticate nodes is signed by the server and sent to the client. If the method selected by the client was not used by the planner, the client will kill the connection. Note that in this and all other signed messages, a globally unique connection identifier is also signed to prevent replay attacks.

4.5.3 Protecting Adaptor Selection

Conductor uses information such as link characteristics, user preferences, and available node resources to decide which adaptations to deploy and where to deploy them. Attackers could force unnecessary or even undesirable adaptations by falsifying information about conditions or illicitly altering the resulting plan. Information gathering, planning, and plan distribution must therefore be protected.

Plans are formulated on the server node, which is trusted. However, the input to that process, the planning information from each node, must be shown to be authentic and from a trusted node before it can be used as an input to the plan.

The authentication information received prior to planning can be leveraged to form a verifiable digital signature for each planning information message. For example, public key encryption can be used in the typical manner. Each node would sign its planning information with its private key before transmission. On receipt, the planner would test the signature using the verified public key for that node.

Once the plan has been formulated, it is signed by the planner, allowing each node to verify its authenticity before deploying the selected adaptors. Note that in the case of public key encryption, the reverse authentication process allows each node to obtain the public key of the server.

4.5.4 Support for Virtual Link Encryption

The plan generated by Conductor must include provisions for protecting sensitive data from untrusted nodes. A simple way to accomplish this effect is to deploy encryption and decryption adaptors on all trusted nodes that need to adapt plaintext data. If each trusted node encrypts the stream for transmission to the next trusted node (where it can be decrypted), secure virtual links will be created between adjacent trusted nodes, and a secure channel will be formed.

While wholesale encryption of the stream is an obvious form of security adaptation, a variety of others can provide various degrees of protection. Different types of encryption may be used. Particular portions of the stream can be encrypted, perhaps just plaintext passwords or credit card numbers. Or, encryption might be avoided by removing sensitive portions of the text.

Encryption requires secure key distribution. Since each adaptor can use a unique encryption algorithm with a unique key type, Conductor does not generate keys. Instead, each adaptor specifies the type of key it requires and provides a key generation function for that key type. After planning completes, Conductor examines the plan on the server node, determines what key types are required, and requests that those keys be generated. Typically, for a given connection, one key of each type can be used for each virtual link. For example, if DES encryption adaptors are to be deployed on several trusted nodes, a single DES key can be generated on the planner node and distributed to each trusted node.

A copy of each key is packaged specifically for each node requiring that key. For public key authentication, keys are encrypted with the receiving node’s public key and signed with the planner node’s private key. Thus, each packaged key can only be decrypted using the private key of the intended recipient, and the authenticity of the key can be ascertained using the server's public key (obtained during reverse authentication). The authentication scheme should be at least as cryptographically strong as the encryption algorithm.

Conductor's security framework has been shown to have very low overhead [20]. The overhead of individual authentication schemes and encryption adaptors depends heavily on the cost of communication with certificate authorities or key servers and the cost of the required cryptographic operations. Since Conductor allows selection of security modules of different strengths, the user can choose an appropriate method.

5 Experiences with Conductor

Conductor’s effectiveness can be measured by its ability to improve the user’s experience through useful adaptations. Thus we have obtained both experiences in real use as well as empirical evidence of Conductor’s effectiveness.

Conductor has been deployed for a modest sized group of mobile users to improve the behavior of their applications over a variety of networks, including within an office LAN, across residential links, and over various wireless networks. Other users have also written and deployed their own adaptors for a variety of protocols, including POP and HTTP.

5.1 Experimental Results

We have measured Conductor’s ability to adapt in three scenarios similar to those described in Section 2. In one of these experiments, previously reported in [33], we created an environment similar to that of a field archeologist from the case study in Section 2.2. By powering down the wireless interface while waiting for slow queries and compressing transmitted data, Conductor was able to reduce the power consumption of the user's PDA by a factor of 10 and simultaneously multiply the throughput by a factor of 3.2.

Additional experiments were conducted to measure Conductor's performance in scenarios similar to the two additional case studies and to measure Conductor's overheads. In these experiments, each node consisted of an HP OmniBook 4150 laptop with a 500 Mhz Pentium III processor and 192 MB of memory running RedHat Linux 7 with kernel version 2.2.16. Conductor executed under IBM JDK 1.3. Apache 1.3.14 was used to serve the HTTP protocol. A custom HTTP client was constructed in Java to allow accurate measurements. The client was designed to mimic the actions of a typical browser and supported up to three simultaneous download streams. 100 Mbps PCMCIA Ethernet cards were used for all non-bandwidth constrained links. A serial link and PPP running at 56 Kbps (with no compression) were used for low-bandwidth links. Frequency-hopping 802.11 PCMCIA wireless network cards were used for wireless links. All results are reported with 95% confidence intervals.

We constructed a user-to-user network, like that of the case study in Section 2.3 (Figure 2), in which two poorly connected users are trying to share images captured by a digital camera. For simplicity, the HTTP protocol was used for transport, and three web pages were constructed with three different sets of photographs: small (31 thumbnail JPEG images with a total size of 155 KB), medium (4 medium-sized JPEG images with a total size of 867 KB), and large (1 large JPEG image with a total size of 895 KB). For each dataset, page download time was measured in two cases: with and without Conductor. In the Conductor case, an adaptor to convert the JPEG images to ASCII-art (using libaa [12]) was deployed on the server and compression and decompression adaptors were deployed on the server and client respectively.[1] While not desirable in all cases, this adaptation represents one of several that might be chosen, depending on user requirements and end-to-end network conditions. Results are shown in Figure 11. In the medium and large photo cases, Conductor dramatically reduced the transfer time by reducing the quality of the images, cutting the number of bytes transmitted over the modem link by a factor of 7:1. In the small photo case, the cost of this adaptation was only slightly less than the gain in transmission speed. In this case, the number of bytes transmitted was halved.

Connection startup time (defined as the time from the client's `connect()` call to the time data begins to flow to the server) in the above hardware configuration with Conductor,

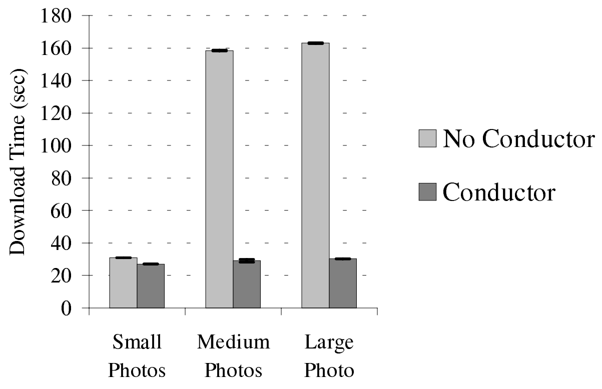

Figure 11: User to user download times of digital camera photo sets with and without Conductor fidelity reduction.

including planning and adaptor deployment, was 556±23 ms. Startup time without Conductor was 109±1 ms.

We measured Conductor's ability to support secure web browsing over a limited channel with a network approximating that of the case study in Section 2.1 (Figure 1). Three datasets were measured: the Apple home page[2] (41 files with a total size of 92 KB), a Slashdot article[3] (10 files with a total size of 187 KB), and a photo gallery similar to those in the previous experiment (10 images totaling 550 KB). Conductor provided encryption at the server and the DSL router, decryption at the ISP proxy and at the client, and prioritization at the ISP proxy. For each web page, we took three measurements. We measured the download time without Conductor or background traffic to obtain the best possible result that prioritization could achieve. The download time was also measured without Conductor but with background traffic (generated by an additional download of a large file from the server, to simulate a large software download). Finally, the download time was measured with Conductor providing encryption and prioritization. The results appear in Figure 12. In all cases, Conductor reduced the download time of the web page at the expense of the software download. In two of the three cases, Conductor achieved nearly the theoretical limit despite additional encryption costs.

Connection startup time for the above hardware configuration with Conductor was 3580±299 ms. The startup time without Conductor was 60±1 ms. The slow startup time in the Conductor case is primarily due to an interaction between Conductor and the wireless hardware used in our experiments. Successful tests with other wireless and Ethernet devices indicate that it is not a fundamental problem with Conductor. Preliminary measurements with other wireless hardware indicate that connection setup, including secure key distribution, can be accomplished in approximately 560 ms.

Conductor's overheads were measured using four nodes connected by a 100 Mbps Ethernet. Table 1 contains the throughput observed while transferring 5 MB without adapta-

[1] This adaptor configuration assumes that the user does not have a graphical display, delivering an ASCII-art formatted image. For users with a graphical display, an ASCII-art browser plug-in could be constructed that could render these images at roughly the same speed as any image or text page.

[2] http://www.apple/com, March 10, 2001.

[3] http://www.slashdot.org/articles/01/03/13/020208.shtml, March 13, 2001.

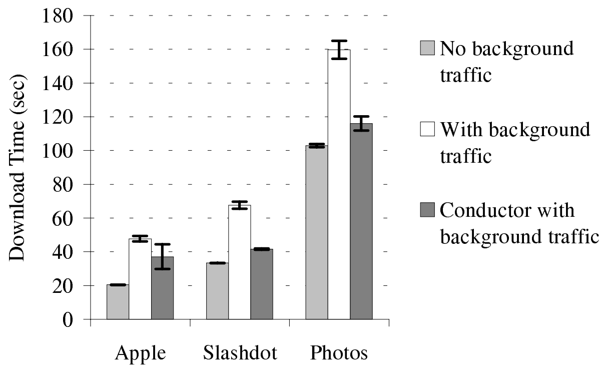

Figure 12: Improvement in web download times using Conductor's prioritization and encryption adaptors.

tion through the four nodes, while the number of nodes on which Conductor was deployed was varied. The data was delivered to the network at a limited rate of no more than 5 Mbps to avoid swamping the PCMCIA bus bandwidth of our hardware. A moderate overhead was observed in deploying two Conductor nodes, compared with the non-Conductor (0 node) case. However, the incremental cost of additional Conductor nodes is very low. Table 2 contains the throughput through the same set of nodes, all running Conductor, while the number of null adaptors (an adaptor that efficiently forwards unadapted data) deployed on one of the nodes is varied. Note that the incremental cost of additional null adaptors is also very low.

Connection startup time in the above hardware configuration with Conductor was 80.7±30.2 ms. The startup time without Conductor was 1.01±0.01 ms.

# of Conductor Nodes	0	1	2	3	4
Throughput (Kbps)	4254 ±6	▨	4171 ±3	4173 ±3	4173 ±3

Table 1: Throughput with varying numbers of Conductor nodes.

# of Null Adaptors	0	1	2	3	4
Throughput (Kbps)	4173 ±3	4173 ±3	4173 ±3	4175 ±3	4174 ±3

Table 2: Throughput with varying numbers of null adaptors.

5.2 Discussion of Results

These experiments have shown that deployment of the right combination of adaptors at the right points in the network can greatly improve the user's experience. In the secure web-browsing example, careful coordination of adaptations enabled the deployment of both encryption and prioritization. In the user-to-user picture-sharing example, the correct adaptation choice depended on end-to-end network characteristics. In addition, dramatically reduced download times were possible because an extreme adaptation was available and acceptable to the user.

While these adaptations were designed to perform well in networks similar to our experimental setup, they would not always be appropriate. For example, while preliminary evidence suggests that prioritization would benefit our datasets over links up to 112Kb/s, this adaptation would clearly be counterproductive in a high-bandwidth network. Similarly, the ASCII-art adaptation is not beneficial when image processing time (a few hundred milliseconds to a small number of seconds on our experimental hardware, depending on image size) is close to or greater than the transmission time of the unadapted image. However, Conductor would correctly handle these cases by not selecting the ASCII-art adaptor when no network bottleneck exists. The adaptor could also be modified to pass small images without alteration (unless the receiving devices cannot display graphics).

The additional startup costs imposed by Conductor are acceptable for most applications, including interactive sessions and connections with a lifespan of more than a few seconds.

In our experiments, the 56Kb/s serial channel's PPP compression was turned off, for several reasons. First, the experimental results are easier to interpret with an underlying channel that provides equal bandwidth to all transmitted data. Second, using the PPP compression rather than a compression adaptor would not have given a clear presentation of the performance of Conductor adaptors. Third, such compression is not always available and not always sufficient, either because the data is incompressible or because the resulting performance is insufficiently improved. Conductor can offer a greater variety of adaptations, including some with very drastic and special-purpose results. Finally, adaptations performed at lower layers can be incompatible with adaptations performed at higher layers. In the secure web-browsing example, had the modem link been considered insecure, PPP compression would have been rendered ineffective by encryption. Conductor, on the other hand, would arrange for a proper ordering of encryption and compression adaptors.

6 Other Approaches

There are at least three approaches to making applications degrade gracefully: situation-specific applications, adaptable applications, or an adaptive service within the network.

6.1 Situation-Specific Applications

Users of substandard networks may choose applications specially designed for their situation. The PalmVII wireless device replaces a general-purpose web browser with "clipping" applications [22] that provide a unique interface to specific web pages. Because each custom-built clipping application requires a proxy host to filter responses from the associated web server, this solution has scaling problems.

As another example, many users who access the Internet through a slow modem or wireless network use a text-based browser rather than a graphical browser to avoid the delays of downloading images. Mobile users who split time between wired and wireless access must manually switch between interfaces depending on their network connectivity, or accept imageless pages even when wired.

6.2 Adaptable Applications

Applications can automatically identify the characteristics of the network and tailor their behavior accordingly. For example, RealPlayer [23] can select a version of a video or audio stream most appropriate for the connectivity available to the client (as specified by the user). Programming or OS support can aid in building adaptable applications.

Both Rover [13] and Odyssey [21] provide tools that help developers build applications that automatically adapt to changing network conditions. Odyssey provides a system-level service that monitors network conditions, informing applications of changes and helping them adapt transmitted data to prevailing conditions. Odyssey illustrates the value of cooperation between an application and the operating system, as well as cooperation between applications. Rover is an application development toolkit that provides higher-level networking abstractions for data migration and disconnected operation, simplifying the design of adaptable applications.

Researchers have also suggested methodologies for partitioning an application to adapt its communication pattern to network conditions. For example, partitioning can allow aggregating functions to occur close to a data source(s) when bandwidth is low or for lookup functions to be performed close to the client when connectivity is intermittent. Several partitioning methodologies have been proposed [16] [30].

Application development tools such as these can help build adaptable applications without programmer knowledge of networking details. However, substantial effort is required to design new applications or retrofit existing applications. Also, these tools are designed primarily to deal with communications between a single mobile client and a fixed server across one bad link, not heterogeneous networks.

6.3 Adaptability as a Network Service

A powerful way to allow applications to be more adaptable is to alter their communication protocols and data from within the network. Selected nodes within the network can monitor and modify packets generated by applications as they flow through the network.

The Snoop protocol [3] improves TCP performance over wireless links by providing caching and quick retransmission of packets from a gateway between the wired and wireless networks. The Protocol Boosters adaptation framework [5], an application-level analog of Snoop, allows pairs of adaptation modules to be transparently deployed, adding new features to existing protocols, such as forward error correction. Protocol Boosters provide lossless adaptation, since the system has no support for reliable delivery if some packets are dropped or permanently altered. Protocol Boosters are composable, but the system does not provide support for determining if a given set of boosters will perform well together.

Transformer Tunnels [27] use IP tunneling to alter the behavior of a protocol over a troublesome link. Transformer Tunnels usually provide protocol-independent adaptations, such as consolidation of packets, scheduling of transmissions to preserve battery power, encryption, lossless compression, and buffering. Transformer Tunnels are transparent to applications, but do not support composition of adaptations.

One of the most advanced proxy solutions is the Berkeley proxy [7]. This system can provide a wide variety of application-level adaptations to a large number of mobile PDA users. A high degree of reliability and scalability has been achieved by leveraging cluster-computing technology. The Berkeley researchers have also investigated methods of composing adaptations on a single machine [9].

Mowgli [17] improves web performance across low-bandwidth, high-latency GSM channels by breaking the channel into two segments at a proxy. Between the client node and the proxy, custom protocols replace both TCP and HTTP, improving link utilization and supporting disconnected operation. Columbia University [35] provides a general-purpose framework that allows mobile clients to dynamically load and control adaptive "filters" on a proxy node at the edge of the wired network. While both of these systems improve the user's experience, neither addresses multiple coordinated adaptations, reliability, or security.

Active networks add generalized adaptability into the network infrastructure [28] [31]. Each packet potentially executes special code at each visited router to determine its proper handling. Key design issues in active networks remain unsolved, including security, cost, and proper architectures. Active network researchers are only beginning to look at composability of adaptations and reliability. In the long term, active networks may provide a superior adaptation infrastructure, but their success is not yet certain.

Research into adaptation within the network has shown that worthwhile adaptation can be accomplished outside of the application itself. Further, adapting application-level protocols, rather than providing generic adaptations at the transport level, can often obtain the best results. Research in this area, however, has focused on providing a single adaptation for a single suboptimal link, usually the last mile.

7 Conclusions

New technologies will increase network heterogeneity over the coming years. To be useful in this environment, applications will have to gracefully adjust to prevailing network conditions. Distributed adaptation can play an important role in assisting applications to meet this challenge.

Conductor demonstrates that systems-level support for gracefully degrading application services is both desirable and achievable. Previous solutions that support adaptation at the endpoints or at individual proxy nodes are frequently insufficient when network heterogeneity extends beyond the last mile. Instead, our experiences with Conductor have shown that greater benefits can often be achieved through coordinated, multi-site adaptation.

Distributed adaptation cannot be achieved by deploying multiple instances of individual adaptation services, but neither does it require a fully programmable network. Instead, Conductor enables distributed adaptation in an incrementally deployable fashion by providing three key facilities: reliability, distributed planning, and security. Conductor includes a reliability model based on semantic segmentation that preserves exactly-once, in-order delivery of semantic meaning, allowing each data stream to be reliably adapted. Conductor provides a planning infrastructure that allows a fully coordinated set of adaptors to be selected based on end-to-end network conditions. The planning process can be secured (to a desired level), ensuring that only desired adaptations are performed and only at trusted nodes. Experimental evidence shows that these facilities can be provided at a sufficiently low cost.

By focusing on adapting protocols and not applications, Conductor is able to provide these services in an application-transparent manner. By remaining fundamentally application-transparent, Conductor allows adaptation of existing and closed-source applications as well as allowing application developers to focus on their applications, rather than the intricacies of current and future network technologies. At the same time, Conductor does not rule out the possibility of greater application-level control of adaptation through a future application-aware API.

Given the Conductor framework, building coordinated and distributed adaptation is relatively straightforward. An adaptation developer needs only focus on understanding application-level protocols, the desired adaptation, and the Conductor API. Conductor provides for the selection and secure deployment of such adaptors, according to prevailing conditions and the desires of the user, and ensures that coherent data is delivered, despite failures.

References

[1] Stephen Adler, "The Slashdot Effect: An Analysis of Three Internet Publications," 1999. Available at http://ssadler.phy.bnl.gov/adler/SDE/SlashDotEffect.html.

[2] D. S. Alexander, W. A. Arbaugh, A. D. Keromytis, and J. M. Smith, "A Secure Active Network Environment Architecture," *IEEE Network*, 12(3):37-45, May/June 1998.

[3] H. Balakrishnan, S. Seshan, E. Amir, and R. Katz, "Improving TCP/IP Performance Over Wireless Networks," *Proceedings of the 1st ACM International Conference on Mobile Computing and Networking (MobiCom '95)*, November 1995.

[4] M. Blaze, J. Ioannidis, and A. Keromytis, "Trust Management for IPsec," *Proceedings of the 2001 Symposium on Network and Distributed Systems Security (NDSS '01)*, San Diego, February 2001.

[5] D. C. Feldmeier, A. J. McAuley, J. M. Smith, D. S. Bakin, W. S. Marcus, and T. M. Raleigh, "Protocol Boosters," *IEEE Journal on Selected Areas in Communications*, April 1999, 16(3):437-444.

[6] A. Fox, S. Gribble, E. Brewer, E. Amir, "Adapting to Network and Client Variability via On-Demand Dynamic Distillation," *Proceedings of the Seventh International Conference on Architectural Support for Programming Languages and Operating Systems*, October 1996.

[7] A. Fox, S. Gribble, Y. Chawathe, E. Brewer, and P. Gauthier, "Cluster-Based Scaleable Network Services," *Proceedings of the 16th ACM Symposium on Operating System Principles (SOSP '97)*, October 1997.

[8] Li Gong, Marianne Mueller, Hemma Prafullchandra, and Roland Schemers, "Going Beyond the Sandbox: An Overview of the New Security Architecture in the Java Development Kit 1.2," *Proceedings of the USENIX Symposium on Internet Technologies and Systems (USITS '97)*, Monterey, California, December 1997.

[9] S. D. Gribble, M. Welsh, E. A. Brewer, and D. Culler, "The MultiSpace: an Evolutionary Platform for Infrastructural Services," *Proceedings of the 1999 USENIX Annual Technical Conference*, Monterey, CA, June 1999.

[10] Jaap Haartsen, Mahamoud Naghshineh, Joh Inouye, Oalf J. Joeressen, and Warren Allen, "Bluetooth: Vision, Goals, and Architecture," *Mobile Computing and Communications Review*, Oct. 1998, 2(4):38-45.

[11] I. Hadzic, J.M.Smith, W.S. Marcus, "On-the-fly Programmable Hardware for Networks," *Proceedings of IEEE Global Communications Conference (GLOBECOM '98)*, November 1998.

[12] Jan Hubicka and Kamil Toman, the ASCII-art library (libaa). Originally at: http://www.ta.jcv.cz/aa. Currently available at ftp://ftp.freshmeat.net/pub/rpms/aalib.

[13] A. Joseph, A. deLespinasse, J. Tauber, D. Gifford, and F. Kaashoek, "Rover: A Toolkit for Mobile Information Access," *Proceedings of the 15th ACM Symposium on Operating System Principles (SOSP '95)*, December 1995.

[14] S. Kent and R. Atkinson, "Security Architecture for the Internet Protocol," RFC 2401, November 1998.

[15] David Kidston, J. P. Black, and Thomas Kunz, "Transparent Communication Management in Wireless Networks," *Proceedings of the Seventh Workshop on Hot Topics in Operating Systems (HotOS-VII)*, Rio Rico, AZ, March 1999.

[16] Dietmar Kottmann and Christian Sommer, "Stublets: A Notion for Mobility-Aware Application Adaption," *Proceedings of the ECOOP'96 Workshop on Mobility and Replication*, Linz, Austria, July 1996.

[17] Mika Liljebert, Heikki Helin, Markku Kojo, and Kimmo Raatikainen, "Mowgli WWW Software: Improved Usability of WWW in Mobile WAN Environments," *Proceedings of IEEE Global Internet 1996*, London, England, November 1996.

[18] D. Mosberger and L. Peterson, "Making Paths Explicit in the Scout Operating System," *Proceedings of the 2nd Symposium on Operating Systems Design and Implementation (OSDI '96)*, October 1996, pp 153-168.

[19] Metricom, Inc., *The Ricochet Modem Reference Guide*. Available at http://www.metricom.com/downloads/modem.pdf.

[20] Jun Li, Mark Yarvis, and Peter Reiher, "Securing Distributed Adaptation," To appear in *Proceedings of the Fourth IEEE Conference on Open Architectures and Network Programming (OPENARCH '01)*, April 2001

[21] B. Noble, M. Satyanarayanan, D. Narayanan, J. Tilton, J. Flinn, and K. Walker, "Agile Application-Aware Adaptation for Mobility," *Proceedings of the 16th ACM Symposium on Operating System Principles (SOSP '97)*, October 1997.

[22] Palm, Inc., "Web Clipping Applications Web Developer's Tutorial." Available at http://www.palmos.com/dev/tech/webclipping/resources.html

[23] RealNetworks Inc., RealPlayer, Software available for download at http://www.real.com.

[24] Dickon Reed, Ian Pratt, Paul Menage, Stephen Earlyk and Neil Stratford, "Xenoservers: An Accountable Execution of Untrusted Programs," *Proceedings of the Seventh Workshop on Hot Topics in Operating Systems (HotOS-VII)*, Rio Rico, AZ, March 1999.

[25] J. H. Saltzer, D. P. Reed, D. D. Clark, "End-to-end Arguments in System Design," ACM Transactions on Computer Sytems, November 1984, 2(4):277-288.

[26] Pradeep Sudame and B. R. Badrinath, "On Providing Support for Protocol Adaptation in Mobile Wireless Networks," Rutgers University, Department of Computer Science Technical Report, DCS-TR-333, July 1997.

[27] P. Sudame and B. Badrinath, "Transformer Tunnels: A Framework for Providing Route-Specific Adaptations," *Proceedings of the USENIX Technical Conference*, June 1998.

[28] D. Tennenhouse and D. Wetherall, "Towards an Active Network Architecture," *Computer Communications Review*, April 1996.

[29] Jos Vos and Willy Konijnenberg, "Linux Firewall Facilities for Kernel-level Packet Screening," November 1996. Available at http://www.xos.nl/linux/ipfwadm/paper/.

[30] Teri Watson, "Effective Wireless Communication Through Application Partitioning," *Proceedings of the Fifth Workshop on Hot Topics in Operating Systems (HotOS-V)*, May 1995.

[31] D. Wetherall, J. Guttag, and D. Tennenhouse, "ANTS: A Toolkit for Building and Dynamically Deploying Network Protocols," *Proceedings of the First IEEE Conference on Open Architectures and Network Programming (OPENARCH '98)*, San Francisco, CA, April 1998.

[32] Mark Yarvis, Peter Reiher, and Gerald J. Popek, "A Reliability Model for Distributed Adapttion," *Proceedings of the Third IEEE Conference on Open Architectures and Network Programming (OPENARCH '00)*, Tel-Aviv, Isreal, March 2000.

[33] M. Yarvis, A. Wang, A. Rudenko, P. Reiher, and G. Popek, "Conductor: Distributed Adaptation for Complex Networks," UCLA Tech Report, CSD-TR-990042, August 1999.

[34] Y. Yemini, A. V. Konstantinou, and D. Florissi, "NESTOR: An architecture for self-management and organization," *IEEE Journal on Selected Areas in Communications (JSAC)*, 18(5), May 2000.

[35] Bruce Zenel and Dan Duchamp, "A General Purpose Proxy Filtering Mechanism Applied to the Mobile Environment," *Proceedings of the Thrid Annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom '97)*, Budapest, Hungary, September 1997.