A Reliability Model for Distributed Adaptation

Mark Yarvis, Peter Reiher, and Gerald J. Popek
University of California, Los Angeles

***Abstract*—End-to-end connectivity is growing increasingly diverse, with orders of magnitude differences in characteristics throughout the network. At the same time, most applications assume a level of network characteristics below which they either provide no service, or service at a cost higher than the user is willing to pay. Open architecture networks can help applications degrade gracefully when network conditions are poor by pushing adaptation technology into the network. Unfortunately, since adaptation seeks to change the data stream during transmission, it is incompatible with the traditional model for reliable data streams.**

A new model of reliability is required to allow general adaptation of reliable data streams. We propose one possible model that provides reliable delivery of the semantic meaning in the data stream despite adaptation. We present this model in the context of *Conductor*, a framework for distributed adaptation. By using a reliability model that is compatible with adaptation, Conductor allows arbitrary, distributed components to operate on a data stream without reducing end-to-end reliability. While Conductor is one possible adaptation service, it is also an example of the type of reliability model required in open architecture networks.

***Index Terms*—adaptation, reliable transport, semantic segmentation, proxy.**

I. Introduction

Network applications often have major problems when network characteristics change during use. Applications must adapt to these changes to provide satisfactory service. An application's communications may cross landlines, satellite links, dial-up service, wireless relays, and asymmetric links, often without user or application awareness. These various links differ in many ways, and even a single type of link can vary substantially, causing severe problems. Early attempts at Internet telephony were largely rejected due to frequent unexplained segments of garbled content. Some users abandoned the Web because of unpredictable and intolerable delays in accessing graphically dense but information-light, content. There are many other examples.

The behavior of network applications can be adapted to these changing conditions in many ways, while still delivering the necessary services: reducing the fidelity of video, graphics and audio; switching to text only; adding application-level redundancy for critical values; encrypting sensitive content; etc. The open architecture approach is a very flexible way to adapt an application's network usage by pushing adaptive services into the network. In many cases, adaptation can only be effective when deployed into the network. It is important, however, that the addition of adaptive services does not decrease the overall reliability of the system.

This work was partially supported by the Defense Advanced Research Projects Agency under contract DABT63-94-C-0080.

Authors may be contacted at: {yarvis, reiher, popek}@fmg.cs.ucla.edu.

Unfortunately, the most popular model for reliability — exactly-once, in-order delivery of bytes — generally assumes that the stream will not be modified in transit. This model may be neither practical nor desirable. Adaptive techniques seek to deliver a more easily transmittable version of a data stream, generally by modifying the data in transit. Consider an adaptation that reduces the color depth of an image. Although the user prefers the adapted version, the number of bytes transmitted will no longer correspond with the number of bytes received, confusing the reliability mechanism.

Some adaptive systems have chosen to limit the allowable adaptations to protect the reliable transport from possible confusion [2, 9]. Generally, these systems do not allow any modification of the original byte stream. Other systems allow more general adaptation, providing a support structure that hides the presence of adaptation from the reliable transport [1, 5, 8]. For instance, an end-to-end TCP connection can be split in two, allowing an adaptation to occur in-between. Unfortunately, hiding the adaptation from the reliable transport introduces a new point of failure to the connection. If the adaptation system were to fail, the end-to-end connection will also fail.

We propose a new model of reliability that is compatible with both an application's need for a reliable transport and the user's desire to receive gracefully degraded service through the use of adaptation. This model provides exactly-once, in-order delivery of *adapted* data. While we present this model in the context of the Conductor distributed adaptation service, the model could be applied to any open architecture network.

II. Distributed Adaptation

Adaptation allows applications to gracefully degrade their services to the dynamically changing levels of service potentially present in the network. Applications place an increasing level of expectation on the network, as is evident in the latest breed of multimedia, Web-based, and thin-client software. When the network does not conform to the expected level of bandwidth, latency, jitter, security, packet loss, and monetary cost, many applications either provide no useful service, or provide a service at a cost greater than the benefit to the user.

This situation is not likely to change. Although new technologies such as high-speed backbones and DSL continue to increase the standard of connectivity, there will always be cases in which the user experiences a substandard quality of service. In particular, mobile-enabling technologies like metropolitan-area wireless services [10] and personal-area networks [7] typically lag in performance and security. In addition, many network links that have sufficient nominal capacity simply become overloaded (e.g., the "Slashdot effect"[1]). Thus, the encountered range of network characteristics is increasing.

One way in which researchers propose to promote graceful degradation of applications is to push adaptive technologies into the network. Users of a metropolitan-area wireless network can browse the Web more quickly by allowing fetched images to be reduced in quality by a proxy node, immediately prior to transmission across the wireless network [6].

When the network is truly heterogeneous, consisting of several links of varying characteristics, use of several points of adaptation may be beneficial. Consider a businessman in a carpool using a PDA. The PDA might connect, via a personal-area network, to the car's cellular phone. The cellphone is periodically connected to the office network, which is in turn connected to the Internet via an ISDN router. A variety of adaptations might be desired. Encryption might be employed on the wireless, cellular, and Internet links. However, encryption cannot be provided end-to-end, or it would interfere with other desirable adaptations. For instance, an adaptor deployed on the cellular phone could control the connection status. When the phone connection is offline, only high-priority traffic should be allowed to establish the costly connection. However, when the connection is already established, any traffic may pass, since the line's use is virtually free. This adaptation must be placed on the phone itself, since only that device is aware of the status of the connection. Other adaptations might also be useful in this example: data distillation for low-bandwidth links, caching of data shared by car-poolers or office staff, or prefetching of stock quotes when the link is active to serve later requests.

A few key characteristics tend to drive the need for distributed adaptation. Some adaptations must be deployed on a particular device to gain access to and control over local device status. Devices such as cellular phones, ISDN routers, and wireless MAN interfaces may require local management of connect-time cost, highly variable bandwidth, or power consumption. Deployment location may also determine the number of users that will benefit from an adaptation. For instance, a cache is most useful if it is accessible by a group of clients. Other adaptors may not be compatible with certain types of network links. While prefetching is useful over high-latency links, it is counterproductive for low bandwidth links. Thus, when a high latency link is adjacent to a low bandwidth link, the location of the adaptor is restricted. Finally, adding security concerns to any of the above situations typically requires link-level adaptation, rather than end-to-end adaptation, thereby increasing the number of points of adaptation. For a mobile user, it is not uncommon for several of these cases to arise, all requiring distributed adaptation.

Reliability is a key issue in distributed adaptation. When only a single point of adaptation is present, it may be acceptable for adaptor failure to lead to end-to-end failure. However, when adaptation is distributed in the network, each new point of adaptation is another potential point of failure. To support distributed adaptation, the system must remain resilient to the failure of adaptor modules and the nodes upon which they execute. The adaptation framework must therefore provide a reliability model compatible with adaptation.

III. The Conductor Adaptation Framework

We have constructed an adaptation framework called Conductor to demonstrate the value and feasibility of distributed adaptation. While we use Conductor as a basis for the discussion of a new model for reliability, the concepts we introduce could be implemented in other contexts as well. Conductor is a stream-oriented distributed adaptation service that is transparent to applications. Conductor consists of two main pieces: adaptors, and the framework for deploying those adaptors.

A. Adaptors

Conductor adaptors are self-contained pieces of code that perform some particular adaptation, often only for a particular type of data stream. The set of Conductor adaptors is expandable. Each Conductor node might have a different set of adaptors available for local use. Adaptors are frequently (although not necessarily) paired, converting from a given protocol to a protocol better suited to the transmission medium, and back to the given protocol. As a simple example, a pair of adaptors might be used to compress and subsequently decompress a data stream. Several adaptations can also be combined serially or composed together. For example, an encryption/decryption pair might be composed within a compression/decompression pair, encrypting the compressed data and allowing the transmitted stream to be both smaller and more secure.

Adaptor pairing allows the protocol expected by the application to be delivered at the endpoint. By conforming to the expected protocol, Conductor is able to provide an applica-

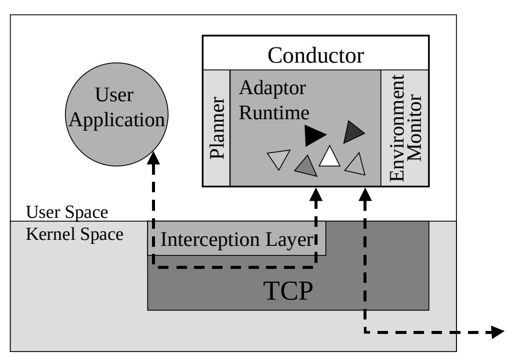

Figure 1: The Conductor architecture deployed on a node.

[1] Named for a popular source of technical news (http://www.slashdot.org), the "Slashdot effect" refers to a sharp increase in traffic experienced at a WWW server that has been referenced at a popular Web page, causing unusually high latency.

tion-transparent service. However, paired adaptors need not regenerate the original data flow, nor are adaptations necessarily user-transparent. Adaptors may deliver *any* data to the application, so long as it conforms to the expected protocol. For instance, an adaptor may cause a color image to be transformed to a black-and-white image, or frames to be dropped from a video stream. These adaptations will clearly affect the user's experience, but must still conform to the protocol expected by the application. Thus, Conductor is application-transparent, but not user-transparent.

B. Framework

Conductor's runtime framework supports adaptor selection, deployment and execution. Figure 1 shows the architecture of Conductor on a single node, consisting primarily of a user-space module that handles monitoring of data flows, delivery of data streams to local adaptors, transmission of data streams between Conductor nodes, planning for new data flows, and recovery and reliability. In addition, in most systems Conductor requires a small kernel modification to trap data flows, allowing Conductor to examine them for possible adaptation. In some systems, existing extensibility mechanisms may allow trapping of data flows without kernel modifications [11].

C. Data Path Setup

Conductor's initial goal for each new connection is to gain access to the data stream on the client and server nodes and at various points along the route from the client to the server. To accomplish this, Conductor should be present on participating client and server nodes. Preferably, Conductor would also be deployed on nodes that are at or near gateways between networks of differing characteristics, allowing adaptation modules to be deployed at these points.

When a new TCP connection is started by an application (which is unaware of the presence of Conductor and of the prevailing network conditions), Conductor traps the opening of its socket, effectively kidnapping the TCP stream. Conductor can then provide the illusion of end-to-end TCP, when actually Conductor is handling the reliable end-to-end delivery of data.

Once Conductor has chosen to intercept a connection, it must form a path over which data will flow. Presumably this path will contain both Conductor-enabled and non-enabled nodes. Conductor employs standard IP routing and the transparent proxy capability built into the Linux kernel [12] to identify Conductor-enabled nodes along the normal data path. The Conductor framework on the client node will attempt to contact the server node. If a Conductor-enabled node is present along the path to the server node, it will intercept the connection and repeat the process.

As this path of potential adaptation sites is formed, each discovered node forwards, along the path, information about its local network conditions and node capabilities Once the path is formed, the information required to generate a plan has been collected at the destination node. This information is used to generate a plan specifying adaptors to deploy. This plan is then delivered back to the participating nodes in one round-trip message, causing a data path to be created with the appropriate adaptors inserted.

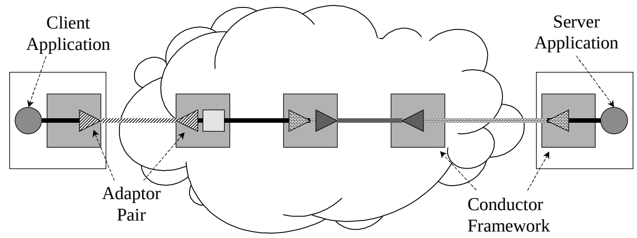

Figure 2: Conductor intercepts client-server communication channels and deploys distributed adaptors.

Once the path is set up, Conductor forwards the user's data stream down the path. Figure 2 gives a simple view of Conductor in use. At each Conductor node, an adaptation might be applied to the data. Some adaptations do not change the data, but many do. Potentially, the bits that arrive at the destination may be very different than the bits that were sent. However, if Conductor's planner has done its job properly, the arriving bits are the most suitable, semantically meaningful version of the data that was possible to deliver in the face of prevailing network conditions. For example, while transmitting a video stream, dropping color in the face of limited bandwidth yields black-and-white frames that are semantically related to the color frames that were sent, but the overall sets of bits are very different.

D. Split-TCP

Conductor uses TCP for reliable data transmission between Conductor nodes. Effectively, Conductor partitions the end-to-end TCP connection into a series of individual, node-to-node TCP connections. Use of split-TCP allows each Conductor node unrestricted access to the data stream without interfering with the underlying transport mechanism. The primary advantage of this approach is ease of implementation. We could have instead built a new transport layer (or modified TCP) to both provide reliability and interoperate with adaptation. By using the existing TCP transport, we avoid duplicating the effort of designing a reliable transport. Also, the TCP interface is convenient in that it provides Conductor transparent interoperation with a large number of existing applications. Finally, previous research has suggested that splitting an end-to-end TCP connection can provide a significant performance gain in heterogeneous networks [3].

Unfortunately, split-TCP does not maintain end-to-end reliability semantics. Data transmitted from one endpoint is acknowledged before it is received at the opposite endpoint. Thus, if the intermediate node fails, data loss can occur. To improve end-to-end reliability, Conductor includes an additional mechanism that is compatible with adaptation. Conductor's reliability scheme is described in detail in Sections V and VI.

A prototype of Conductor has been developed and used to successfully demonstrate that distributed adaptation can provide a real benefit to users [14]. To succeed, however, the additional points of failure that Conductor adds must not reduce the overall reliability of the system.

IV. Reliability in Distributed Adaptation

TCP guarantees exactly-once, in-order delivery of a byte-stream and provides a convenient model for application writers. However, this model is entirely at odds with adaptation. Adaptation seeks to deliver a version of the transmitted data that is cost-effective, so the bytes delivered may bear no resemble to the bytes transmitted. A new model of reliability is required that instead provides exactly-once, in-order delivery of *adapted* data.

At the same time, adaptation adds new components to a connection: adaptor modules, nodes upon which adaptation is occurring, and the links between those nodes. To avoid reducing the overall reliability of a connection, the end-to-end reliability mechanism must protect against the failure of these new components. Notice, however, that nodes play a supporting role along the data path. To recover from a node failure, we need only consider the resulting failure of one or more adaptors and links.

When a failure does occur, the remaining nodes can re-establish a data path with the same mechanism used to establish the original data path. Any adaptors no longer in the data path are then assumed to have failed. Once the data path is restored, recovery must be provided, not only for lost data, but also for failed or lost adaptors.

A. Complexities Introduced by Distributed Adaptation

Protocols that provide transport-level reliability (e.g., TCP) are typically implemented at the endpoints. Packets may flow along any path and do not depend on particular intermediate nodes. Network-level protocols mask the failure of links and routers along the transmission path.

The introduction of adaptation into the network complicates this model. Adaptation modules execute on particular nodes in the network. In general, this constrains the data path and adds additional points of failure, both in terms of the software adaptors and the hardware on which they are deployed. Conductor splits the TCP connection that would normally be created from client to server into several independent TCP connections, one between each adjacent pair of Conductor nodes. In doing so, there is a danger of violating the normal, end-to-end reliability model, whereby no data should be acknowledged before it is delivered. Fundamentally, however, Conductor's use of split-TCP will be acceptable if the end-to-end reliability of the system does not depend on the reliability of the intermediate nodes.

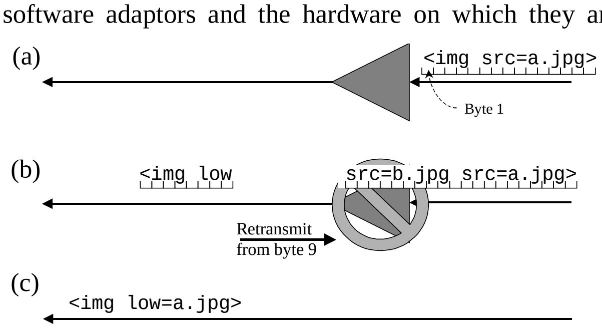

Figure 3: Failure recovery using a byte-count – (a) data arrives at adaptor; (b) adaptor fails while adding *lowsrc* attribute and retransmission is requested; (c) retransmission produces an undesirable result.

When a failure occurs, recovery is complicated by the fact that adaptors are constantly operating on the stream. The system must ensure that the resulting data stream conforms to the protocol expected by the application. Maintaining data integrity is not simple, however. An adaptor failure does not necessarily occur at a point appropriate for switching back to the original stream. For example, consider an adaptor that adds a *lowsrc* attribute to an *image* tag in an HTML data stream (see Figure 3). If a failure occurs in the middle of the tag, a byte-count retransmission scheme will not necessarily retransmit from the beginning of the tag. The resulting stream may be neither the adapted data nor the original data, and may not even be syntactically correct.

Even if an adaptor fails at an appropriate point (for switching adaptors) in the stream, it is still difficult to effect retransmission of lost data. For instance, if an adaptor converts color video frames into black-and-white, the adapted frames will consist of a much shorter byte stream. If a failure occurs after frame 100, a simple byte-count retransmission scheme might begin retransmission after frame 50, duplicating data the user has already received. The information that correlates the data received downstream with the data transmitted is crucial to determining an appropriate point of retransmission. If this information is lost with the adaptor, correct transmission can not continue.

Any reliability scheme must also preserve the properties of the adaptation activities. In Conductor, adaptors are frequently deployed in pairs and may also be composed together. Each paired adaptor expects its counterpart to be present. For example, an encryption adaptor should not be allowed to operate without its corresponding decryptor. Similarly, the inner pair of two composed pairs may expect the outer pair to be present. Consider an adaptor pair that takes a motion-JPEG stream, drops every other frame for transmission, and then restores the original frame rate by duplicating each frame for delivery. To deploy such an adaptor in an MPEG stream, a pair of adaptors that converts MPEG to motion-JPEG and back must surround the frame-dropping pair (see Figure 4). Should one of the format-converting adaptors

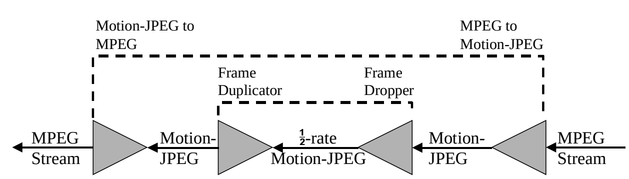

Figure 4: A sample composed adaptation.

fail, the repair action must also involve the frame-dropping pair.

We do not believe that these problems can be adequately addressed by the applications themselves. Most of the problems described above require adaptation-specific knowledge for proper failure recovery, which application-writers may not have. While simple schemes, like restarting a connection from the beginning upon failure, are possible, this is not a desirable programming model. Instead, we have designed a reliability scheme that is controlled by the adaptation system and requires adaptor participation rather than application participation.

B. Redefining Reliability

Most reliability models assume that data is immutable during transmission. Each transmitted byte travels through the network and 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. Notice that the property of immutability is required by the notion of exactly-once delivery.

When adaptation occurs within the network, the bytes transmitted may be very different from the bytes that travel through the network. The bytes seen at various points in the network may, in turn, be different from the bytes that are eventually delivered. For example, if an adaptor is used to reduce the resolution of a video stream, it is acceptable that some bytes are received, while other bytes are not. Exactly-once delivery of bytes is, therefore, not a desirable model. In this example, it is exactly-once delivery of the video frames that is important.

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 in order for adaptation to occur, the data stream must be carved up into segments that are semantically meaningful within the protocol being transmitted. For instance, a video stream might be broken into frames, while an HTML stream may be divided into tags and text. Each segment must be delivered exactly once, in some form. The data may be original, or adapted, or deleted entirely, but the segment must arrive exactly once, and in order.

The second property restricts the content of the segments at the time of final delivery. If halving the frame rate is within the constraints of the protocol expected by the application, then delivering empty segments in the place of every other frame might be acceptable. The second property also requires that segments have an appropriate scope so that an adaptor failure will not violate the expected protocol. These properties ensure that some viable version of the data produced at the source will arrive at the destination.

C. Attaining Reliability

Conductor combines three mechanisms to provide reliability in the face of adaptation. Conductor employs a TCP connection between adjacent Conductor nodes along the data path, providing reliable delivery between adaptor modules on different nodes. When Conductor nodes, adaptors, and the links between them do not fail, end-to-end transmission of adapted data proceeds reliably and in order. When one of these elements fails, data loss can result. Conductor must, therefore, detect link, node, and adaptor failures. Since Conductor provides the run-time environment for adaptors, it can easily detect their failure. For simplicity, the failure of nodes and links is assumed to cause one or more TCP failures that can be detected at the adjacent Conductor nodes. Once a failure is detected, Conductor provides two additional reliability mechanisms to prevent end-to-end connection failure.

First, Conductor provides a data recovery mechanism to protect against data loss on the stream. Once the data path is restored, Conductor must determine what 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.

Second, Conductor provides an adaptor recovery mechanism. Conductor must determine which components were lost and ensure that proper adaptor composition is preserved. For efficiency, this mechanism should not require global coordination across all nodes in the data path.

Since node and adaptor failures result in the loss of information crucial to recovery, the key is to maintain enough information at surviving nodes to guarantee recovery from loss. The data recovery and adaptor recovery mechanisms are described in detail in the following two sections.

V. Data Recovery

The goal of data recovery is to prevent data loss if a portion of the data path fails. Each semantic element of the data stream should be delivered exactly once. Furthermore, adaptation of those semantic elements must not be incomplete. In other words, a particular adaptor should adapt an entire semantic element, or it should adapt none of it; partial adaptation should be disallowed.

A. Semantic Segmentation

These goals are accomplished using a technique called *semantic segmentation*. A semantic segment is the 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 the responsibility for maintaining appropriate segmentation.

The initial data stream consists of bytes being transmitted by the application and intercepted by a Conductor module on the source node. These bytes are considered by Conductor to be logically segmented into one-byte segments, which are numbered sequentially. Note that it is not necessary, at this stage, to track segment boundaries, or segment numbers. The data 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 either of 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 the example of an adaptor that compresses video frames. Before each frame can be compressed, the segments making up that frame would also be combined into one segment. If, hypothetically, the stream consisted of 100-byte frames, each frame would initially be represented in the stream as 100 1-byte segments. If the first frame began with segment 1, the second would begin with segment 101. Before reducing each frame to 50 bytes, the adaptor would combine the 1-byte segments for a given frame into a single segment. 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 labeled 100 and another labeled 200. The resulting 50-byte segments contain the same semantic content as the 100-byte segments and the 100 1-byte segments; only the format has changed.

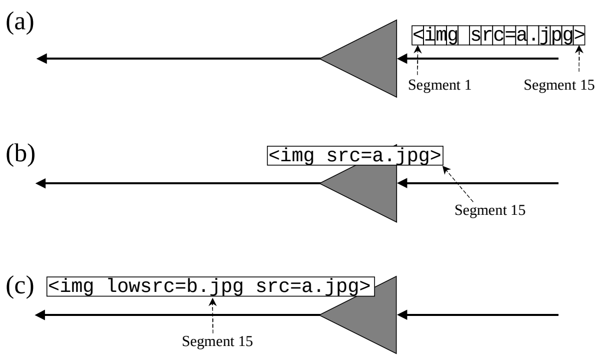

Figure 5: Proper segmentation to adapt an HTML stream – (a) the stream initially has 1-byte segments, (b) the adaptor combines these segments, and (c) performs the adaptation.

In the HTML example from Section IV, when adding a *lowsrc* attribute to an *image* tag, the adaptor would create a segment that contains the entire tag (see Figure 5). Because the entire tag is contained in a segment, the *lowsrc* attribute can be added without disturbing overall segment numbering. In both of these examples, the segments chosen correlate the semantic meaning of the data in the stream, before and after adaptation, and provide an appropriate granularity for stopping the given adaptation.

Subsequent adaptors may cause further segment combination, and segments may grow to arbitrary length. It is not, in general, possible to correlate individual bytes in a segment with their original source segments. As a result, a segment cannot be broken down without violating the rules of segmentation. Once 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, given in Section C).

B. The Adaptor API

Semantic segmentation is implemented as part of the adaptor API. Each adaptor is given a window into the data stream. An adaptor is able to view and modify the data stream through this window. To read new data into the window, an adaptor performs the `expand()` operation. To write data out of the window, the adaptor performs the `contract()` operation. Thus, the adaptor controls the flow of data by moving the window boundaries along the data stream.

The operations that adaptors can perform on the data stream enforce the rules of semantic segmentation. 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. These segments must first be combined into one large segment and labeled appropriately. Once contained within a single segment, the old data can be removed and replaced with the new data. Facilities are also provided for more complex operations such as deletion and insertion and for multiple operations that constitute one semantic modification to the data stream.

Notice that the adaptor API controls segmentation of the data stream. Therefore, while a malfunctioning adaptor can provide incorrect data to the user, it cannot violate the rules of segmentation.

C. The Recovery Protocol

When a node, adaptor, or TCP link fails, some portion of the data stream may be lost. Conductor uses the semantic segment as the unit of retransmission.

To allow retransmission, Conductor caches segments internally at the source node. Since we can depend on the source node not to fail (as is the case with traditional reliability mechanisms), this cache provides a durable point of failure recovery. To improve the speed of recovery, caching can also be added at other points along the data stream.

Recovery is initiated at the point immediately downstream of the failure. First, the data path is spliced across the failure. In the case of a failed node or link, a new TCP connection will be initiated, replacing those that have failed. Next, any segment that has been partially received is discarded. If the beginning of a partial segment has already been passed downstream, a cancellation message will be sent downstream. Note that for applications unaware of the adaptation system, the possibile cancellation of partial segments requires that the adaptation system not deliver any segment to the application until that segment is complete.

The node downstream of the failure then sends a retransmission request containing the ID of the last segment received. The retransmission request travels upstream until it can be serviced, either by a cache or by an adaptor (perhaps from an internal cache). To preserve in-order delivery, all data transmission to nodes awaiting retransmission is suspended (incoming segments are discarded) until retransmission begins. 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 segment. The data can then be re-adapted in the same way, or perhaps differently.

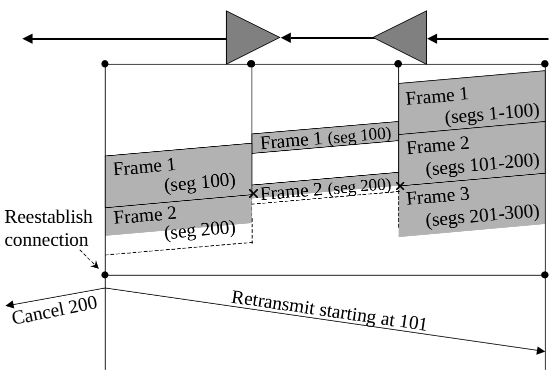

Figure 6: Recovery from the failure of an adaptor pair that compresses frames in a video stream.

Continuing the video example from the previous section, consider the case in which 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 (see Figure 6). 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, however, only one of the two adaptors fails, although the lost data can be recovered, retransmission through the remaining adaptor will not produce the correct results. Additional steps are required to ensure that a correct set of adaptation modules is present when the data flow resumes. This problem will be addressed in Section VI.

The above recovery scheme provides failure-based recovery. Since a retransmission request occurs when there is a failure and indicates exactly what data is required for retransmission, acknowledgements (beyond what is already provided by TCP in the underlying connection) are not required. However, in order to limit the size of cache growth, and to allow adaptors to free any accumulated internal state, the destination node generates acknowledgements whenever a segment is completely received. Acknowledgements are cumulative, allowing all combined segments to be acknowledged when a segment is finally received at the destination.

D. Discussion

Semantic segmentation is effective because it allows enough state to be maintained outside of the adaptor modules to properly control retransmission.

Provided that adaptors follow the rules of segmentation, retransmission after a failure will always provide correct results. A segment lost downstream can always be replaced by one or more segments from upstream that have equivalent semantic meaning. Retransmission will also occur from a point at which adaptor changes (i.e., the possible failure of an adaptor) will not affect the correctness of the resulting data stream.

Moreover, retransmission will always be possible. Data will always be available because a cache at the source node is mandatory. A segment boundary downstream will always correspond to a segment boundary upstream, since segments, once combined, are never divided.

Finally, proper end-to-end reliability semantics are preserved. Although the individual TCP connections provide acknowledgements before data is received at the destination, Conductor monitors the data received at each node (including the destination node), correlates that data with the stream transmitted from the source, and performs retransmissions in response to failure.

VI. Component Recovery

Frequently, several adaptors are deployed together, operating on the same data stream, but potentially distributed throughout the network. A given adaptor may depend on the presence of other adaptors for correct operation. Recall that Conductor allows adaptors to be paired to convert from one protocol to another protocol and back. For example, an adaptor pair that compresses and decompresses an HTTP stream maintains a compressed-HTTP protocol between the pair of adaptors. A correct data stream can only be presented to the application if either both adaptors are present or neither adaptor is present.

Two pairs of adaptors can also be composed together. Composition of adaptors generates a hierarchy of protocols. For example, recall the data path in Figure 4. An MPEG stream flows through a pair of adaptors that converts the stream to motion-JPEG. Within that pair is another pair of adaptors: one of which drops every other JPEG frame, and the other which duplicates the frames, halving the frame rate for transmission and then restoring it again. Between this pair, therefore, the data conforms to the half-rate-motion-JPEG protocol.

When adaptor composition is present, the inner pair expects as input and generates as output the protocol found within the outer pair. Thus, the inner pair is dependent on the outer pair. Conductor must ensure that dependencies between adaptor pairs are also restored after a failure.

A. Recovery Strategies

Two actions are possible when an adaptor fails. A failed adaptor could be re-instantiated. The replacement adaptor could be created on the same node, or on a different node (in the case of node failure), provided its logical position in the data stream remains the same. Unfortunately, because some adaptors maintain state, this is not always possible. For instance, many compression algorithms produce their dictionaries on the fly. A replacement decompression adaptor would be out of sync with an existing compression adaptor.

The other alternative is not to replace the failed adaptor, but to instead remove any adaptors that depend on the failed adaptor. Adaptor removal is always safe because the data recovery algorithms can always restore lost data, as in the case of a failure. There will, however, be the performance penalty of data retransmission. Once the required adaptors are removed and a stable set of adaptors remain, the adaptors that were removed can typically be replaced. By instantiating related adaptors together, synchronization is maintained.

B. Tracking the Protocol Hierarchy

The key to restoring a balanced set of adaptors after a failure is determining which of the remaining adaptors also need to be removed. The solution is complicated by the fact that a given Conductor node may not know which adaptors have been lost. In addition, to speed recovery, it would be best if recovery could be accomplished using only local information, as is the case for the data recovery algorithm.

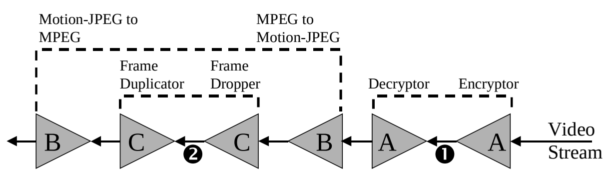

Figure 7: A sample adaptation hierarchy.

To allow component recovery, additional information is recorded at the time that adaptors are deployed. For each logical link between adaptors, Conductor keeps track of the hierarchy of composed adaptors that surround that link. In Figure 7, the hierarchy recorded for the link labeled 1 is A (for the encryption adaptation). Similarly, the hierarchy recorded for the link labeled 2 is B/C (for the MPEG-Motion-JPEG and the frame rate reduction adaptations).

When an adaptation is installed, the new hierarchy is trivial to obtain as a function of the existing adjacent hierarchy. For instance, before the frame reduction adaptation is deployed, the type hierarchy at point 2 is B, so it is trivial to determine that the new hierarchy on the link created between the pair is B/C.

C. The Recovery Algorithm

The component recovery algorithm is executed during data recovery, prior to data retransmission. Recall from Section V that data path recovery begins by splicing together the data path. The two spliced ends will not necessarily have the same adaptation hierarchy. Comparing these two hierarchies, however, will reveal the target hierarchy for this link, once the pairs to failed adaptors are removed, allowing corrective action to be taken.

Recovery occurs as follows: The two hierarchies present at the splice point are compared (either upstream or downstream from the failure) to obtain the greatest common ancestry, which is the correct hierarchy for this link. Two messages are generated, each containing the identity of this hierarchy. One is sent upstream, the other downstream. Each message will proceed, causing the removal of any adaptors that are present, until a link with the correct type hierarchy is discovered. Once the recovery message reaches a link with the desired hierarchy, the recovery message is discarded.

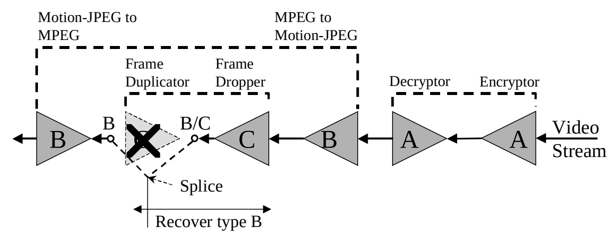

Figure 8: Recovery of appropriate pairing after failure of the frame duplicator.

In the previous example, if the frame duplication adaptor were to fail, as indicated in Figure 8, the two hierarchies present at the splice point would be B and B/C. The greatest common ancestry is simply B. A message containing this information is sent in each direction. In the downstream direction, B is already present, so no action is taken. In the upstream direction, the matching pair to the failed adaptor is removed, after which the proper hierarchy is achieved.

Recovery messages also affect any data caches maintained by Conductor. These caches store data that has already been adapted. Since the goal of a recovery message is to change the type of data flowing at a particular point in the data stream to conform to a new type, any cache encountered by a recovery message must be cleared. Otherwise, a retransmission request could be serviced by data of the wrong type. Since caches in the middle of the data stream only enhance performance, clearing a cache will not affect correctness. Note that it will never be necessary to clear the cache at the source node, since it is located ahead of all adaptors.

For the example shown in Figure 8, any cache present between the frame dropping and frame duplication adaptors would be cleared. Notice that in some cases, including this example, an adaptor may not change all segments, and unchanged segments do not have to be removed from the cache. However, we believe that the additional information required to allow a cache to determine which data to keep is not worth the benefit.

D. *Discussion*

This algorithm works because enough information is kept locally to determine which adaptors to remove upon any possible failure. Comparison of the hierarchy at any two points will obtain the correct hierarchy that should be present if all adaptors between those points were to fail. All that remains is to remove the now inappropriate adaptors.

Furthermore, by placing the recovery messages into the data flow, surviving data can be processed before recovery is attempted. For example, if an encrypting adaptor fails, data that survived the failure and is in transit to the decryptor can be processed by the decryptor before it is removed. The recovery message will follow this data and trigger removal of the decryptor after the data is processed. On the other hand, if the decryptor fails, the component recovery algorithm must only ensure that the encryption adaptor is destroyed before the data recovery algorithm performs retransmission of the segment following the last decrypted segment.

VII. RELATED WORK

Many adaptive systems have preceded Conductor. Some have been tremendously successful, demonstrating that adaptation can greatly improve the user's experience. Adaptation systems that operate on reliable streams can be divided into two categories: those that preserve end-to-end reliability semantics, and those that do not.

Indirect-TCP [1] provides one form of adaptation that seeks to improve the performance of TCP in the face of drastically different networks. TCP does not react well to the high error rates and hand-off behavior of wireless links. Rather than change TCP on both wired and wireless hosts to better support wireless users, Indirect-TCP seeks to provide a gateway between the wired and wireless networks. Standard TCP is used between the gateway and hosts on the wired net, while a version of TCP, optimized and extended for wireless networks, is used between the wireless host and the gateway.

A similar approach can also be taken with application-level protocols via the use of application-layer proxies. One of the most advanced proxy solutions is the Berkeley proxy [5]. This system uses cluster computing technology to provide a shared proxy service for a wide variety of PDAs used at UC Berkeley. The proxy is capable of providing many important services, including transformation (changing the data from one format to another), aggregation (combining several pieces of data into one), caching, and customization (typically converting a data format into one suitable for a particular PDA). The Berkeley researchers have also examined how to use a clustered proxy service to provide highly reliable, scaleable services to a large number of customers.

Both Indirect-TCP and the Berkeley proxy design use a split-TCP approach, which inherently breaks end-to-end reliability semantics and adds a proxy node that is an additional point of failure. The Berkeley researchers have partially mitigated this issue by increasing the reliability of the proxy node. However, their reliability solution cannot be extended to fully distributed adaptation, where adaptation might be required at many points in the network.

Another approach, described in [8], also allows application-level adaptation at a proxy node. However, rather than splitting the TCP connection, the proxy node translates all TCP meta-data traveling between the two endpoints according to the adaptation that was performed, tricking TCP into believing that it is still providing exactly-once delivery of bytes. Although this approach avoids splitting the TCP connection, it still fails to provide end-to-end reliability semantics. If the proxy node fails, the illusion provided to the TCP connection will vanish, at best causing failure of the TCP connection.

Other researchers have proposed solutions that do preserve end-to-end reliability semantics. Like Indirect-TCP, the Snoop protocol [2] was designed to improve TCP performance across wireless links. The Snoop protocol provides a gateway at the border of the wired network that caches all unacknowledged packets delivered to the wireless network. If the gateway believes that some of these packets might have been lost, it retransmits them proactively. This scheme improves TCP's ability to react to packet loss, which occurs far more frequently on a wireless network, while carefully avoiding violations of the basic TCP protocol. Therefore, failure of the gateway (or a handoff to a new gateway) leaves a functional TCP connection.

One application-level analog of Snoop is the Protocol Boosters [9] adaptation framework. Protocol Boosters allows pairs of adaptation modules to be added transparently to the protocol graph. These adaptation modules can add new features, such as forward error correction, to existing protocols. Provided that the adaptation is limited to adding additional information to the data stream without disturbing the original data, end-to-end reliability semantics are preserved. When adaptor failure occurs, only the benefit provided by the adaptation is lost.

Snoop and Protocol Boosters preserve end-to-end reliability semantics by cleverly limiting the set of allowed adaptations. Among previously existing systems, the attempt to provide both adaptation and a traditional reliability model has led to a choice between generalized adaptation and uncompromised reliability.

The semantic segmentation mechanisms presented in this paper are somewhat similar to the synchronization facility specified in the OSI Session Layer [4]. In the OSI model, applications can specify synchronization points, allowing later rollback of the communication channel. However this facility is insufficient for providing reliability in the face of adaptation. Endpoint-specified synchronization can not insulate the transport layer from midstream adaptation nor provide proper recovery from adaptor failure.

VIII. Status and Future Work

Our belief that distributed adaptation is an important facility, as the complexity of networks increases, was originally published in [13]. Since that time, we have completed our initial implementation of the Conductor framework. This implementation allows selection and distributed deployment of adaptor modules into a network. These adaptor modules can then perform arbitrary operations on a data stream.

We also constructed a sample set of adaptations that were used to measure the potential benefit of distributed adaptation in a sample real-world scenario [14]. We measured the performance of a sample application deployed in a mobile network environment and were able to show that the use of a few carefully placed adaptors could improve response time by 69% and reduce battery power consumption by a factor of 10. Thus, we believe that distributed adaptation services will provide significant improvements in the cost/benefit ratio experienced by network-challenged users.

Implementation of the reliability model described in this paper is nearly complete. Semantic segmentation and the adaptor API are functional, but the recovery algorithms are under development. However, the previously reported performance measurements have included all of the overheads involved in this model in the absence of failures. With the implementation of the recovery protocols, Conductor will allow both arbitrary adaptation and end-to-end reliability.

Once implementation of the reliability algorithms is complete, a few extensions will also be possible. One drawback of the current system is the use of ever-growing segments. The permanence of segment combination is troublesome when applications are unaware of adaptation, since data can only be delivered to applications from completely received segments. Further research is required to determine if this is an important limitation and whether the recovery algorithms can be extended to allow breaking down of segments.

Another area of further research is caching adaptations. Currently, segments are only created at an endpoint and may not gain any additional semantic meaning in transit. As a result, any adaptor that generates responses to queries from an internal cache (e.g., a Web cache) is disallowed. We are currently completing the design for an extension to the existing segmentation model that includes a "reliable round trip" for queries and responses.

IX. Conclusions

Systems that push new features into the network have received a great deal of attention in recent years. Meanwhile, little, if any, research has focused on ways to provide an appropriate level of reliability for such systems. Since reliable streams are currently the dominant transport mechanism in the Internet, it is important to consider how services within the network affect the reliability experienced by the user and the application software.

Historically, two stances have been taken. Some researchers have chosen to consider particular intermediate nodes as having special status. Failure of these nodes is considered an acceptable risk, which is minimized as much as possible. Other researchers have chosen to restrict allowable operations to those that do not jeopardize reliability.

Conductor takes a third stance by allowing arbitrary operations on the data stream while protecting against all reasonable failures. Conductor integrates reliability with the ability to adapt the data stream, providing exactly-once, in-order delivery of *adapted* data and enforcing the preservation of semantic meaning from end to end. Conductor also protects the semantics of the adaptation system itself, ensuring that a coherent set of operations occur.

Conductor is an example of one choice in the spectrum of reliability, protecting against all failures. Further research may indicate that by accepting some additional failure modes, the cost of reliability can be reduced. It is clear that for open architecture networks to succeed, some new model of reliability is required. We believe that Conductor's model is an important first step.

References

[1] Ajay V. Bakre and B. R. Badrinath, "Implementation and Performance Evaluation or Indirect TCP," *IEEE Transactions on Computers*, 46(3):260-278.

[2] 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)*, Nov. 1995.

[3] R. Cohen and S. Ramanathan, "Using Proxies to Enhance TCP Performance over Hybrid Fiber Coaxial Networks," Hewlett-Packard Laboratories Tech Report #HPL-97-81, 1997. Available at: http://www.hpl.hp.com/techreports/97/HPL-97-81.html.

[4] Willard Emmons and A. S. Chandler, "OSI Session Layer: Services and Protocols," *Proceedings of the IEEE*, Dec. 1983, 71(12):1397-1400.

[5] A. Fox, S. Gribble, Y. Chawathe, E. Brewer, and P. Gauthier, "Cluster-Based Scalable Network Services," *Proceedings of the 16th ACM Symposium on Operating System Principles*, Oct. 1997.

[6] A. Fox, I. Goldberg, S. D. Gribble, D. C. Lee, A. Polito, and E. Brewer, "Experience with TopGunWingman: A Proxy-Based Graphical Web Browser for the 3Com PalmPilot," *Proceedings of the IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing (Middleware '98)*, Lake District, UK, Sept. 1998.

[7] 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.

[8] 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*, Rio Rico, AZ, March 1999.

[9] A. Mallet, J. Chung, and J. Smith, "Operating System Support for Protocol Boosters," HIPPARCH Workshop, June 1997.

[10] Metricom Corp., "Ricochet Wireless Modem," http://www.ricochet.net.

[11] D. Mosberger and L. Peterson, "Making Paths Explicit in the Scout Operating System," *Proceedings of OSDI '96*, Oct. 1996, pp 153-168.

[12] Paul Russell, "Linux IPCHAINS-HOWTO," March 1999. Available at: http://www.rustcorp.com/linux/ipchains/HOWTO.html

[13] Mark Yarvis, Peter Reiher, and Gerald J. Popek, "Conductor: A Framework for Distributed Adaptation," *Proceedings of the Seventh Workshop on Hot Topics in Operating Systems*, Rio Rico, AZ, March 1999.

[14] Mark Yarvis, An-I A. Wang, Alexey Rudenko, Peter Reiher, and Gerald J. Popek, "Conductor: Distributed Adaptation for Complex Networks," UCLA Tech Report, CSD-TR-990042, 1999. Available at: http://fmg-www.cs.ucla.edu/Conductor/CSD-TR-990042.ps.