Distributed Adaptation: A Research Proposal

Mark Yarvis

University of California, Los Angeles
Technical Report: CSD-TR-200018

***Abstract*—Under suboptimal network conditions, applications must degrade gracefully in order to provide service that is valuable to users. The research proposed in this paper will explore the notion that distributed adaptation—the coordination of several different adaptations at various points in the network—is beneficial in heterogeneous networks. Existing technologies tend to provide one adaptation to overcome a single network deficiency, often focusing on the "last mile" of the network. These solutions are insufficient in the increasingly heterogeneous Internet. This paper describes the challenges of distributed adaptation and proposes a research course to develop the technology that will allow applications to adapt to the networks of the future.**

1 Introduction

Advances in wired and wireless technologies simultaneously increase the world's interconnectivity and the heterogeneity of computer networks. The average bandwidth between any two points in the Internet is going up, but the range of experienced bandwidth is also increasing. Applications that are unable to adapt to suboptimal network conditions will become decreasingly useful, particularly to mobile users.

One approach to handling this challenge is to add adaptive technology to network nodes. These nodes could provide processing and/or storage capabilities and be allowed to operate on higher protocol stack layers. This idea is controversial. Historically, transport (and higher) protocol layers have been provided at endpoints only. Building knowledge of higher layer protocols into routers has several disadvantages, including performance and reliability. The research proposed in this paper will not decisively resolve this controversy. Instead, it will provide evidence in favor of raising the level of services within the network from two perspectives. First, this research will demonstrate several cases in which awareness of application-level protocols at nodes within the network can help applications adapt their services when network conditions are suboptimal. Second, it will develop technologies that allow the network to provide adaptive services to applications in a sensible and reliable manner.

Successfully enabling distributed adaptation requires several new technologies. In the presence of adaptations that may arbitrarily delete, add, or modify elements in the data stream, the typical model of exactly-once delivery no longer makes sense. Instead, a new model for the reliable delivery of adapted data must be developed. Without protection, a service for distributed adaptation would introduce new security vulnerabilities to networks. Yet, the user's data and control over adaptation itself must be protected across administrative domains and without the presence of a ubiquitous infrastructure for authentication. These challenges, and others, will be met by this research.

To this end, we will construct an adaptation framework called Conductor to explore the advantages of distributed adaptation. Conductor will provide an application-transparent adaptation service within the network. Adaptor code modules, dynamically deployed as needed at Conductor-enabled points within the network, will adapt data streams as required. Conductor will allow multiple adaptations to be deployed in a coordinated manner, efficiently adapting to the characteristics of heterogeneous networks. Conductor will support arbitrary adaptations for application-level protocols without compromising reliability. The techniques developed for Conductor will enable applications to be adaptable in tomorrow's heterogeneous networks.

Section 2 describes the need for gracefully degrading applications and some typical approaches to this problem. Section 3 describes a new approach to this problem and several common cases in which this approach is superior. Section 4 describes the challenges and requirements of successful distributed adaptation. Section 5 describes the status of the Conductor prototype and its key technologies. The remaining sections provide a schedule for completion of the research, a basis on which its success will be measured, a discussion of future work, and concluding remarks.

2 Adaptation

While new technologies are improving overall network capacity, the Internet is becoming increasingly heterogeneous. A given link has particular levels of bandwidth, latency, jitter, security, cost, and reliability. These characteristics may vary by several orders of magnitude between different link technologies. In addition, transient conditions, such as congestion and noise, may alter a given link over time. While LANs may offer up to Gigabit bandwidths, significantly lower bandwidth connections to the home will continue to be common. Wireless users, who trade bandwidth, cost, security, and reliability for mobility, will probably never experience network characteristics comparable to wired networks.

At the same time, applications are increasingly dependent on network connectivity. While in the past only a few specialized applications made use of network services, now nearly all do. Word processors, spreadsheets, tax preparation software, audio/video players, and games provide added value from the Internet. Thin client software uses a network, not for communication itself, but to reduce the storage requirements of client hardware and to reduce system administration. Internet appliances like the Kerbango Internet radio[1] and the Aplio Internet telephone[2] are available today. Other appliances like televisions and refrigerators may also leverage the Internet in the future.

Application designers cannot anticipate all possible network conditions their code will encounter. There are too many possible combinations of link characteristics, and new technologies continue to appear. Thus, an application is typically designed to perform adequately when the network's characteristics meet an assumed minimum. When latency is too high, bandwidth is too low, or too many packets are lost, an application's cost in time, security, or money may exceed its value.

This cost/benefit imbalance is not uncommon: participate in a video-conference over a modem, risk credit card theft when an Internet vendor has neglected to provide a secure web form, or play Backgammon at Yahoo when the server latency is high. Increasingly, heterogeneous networks require an application's behavior to be automatically adaptable, gracefully degrading service to match the current network capabilities.

2.1 Graceful Degradation

When the network cannot support a high quality of service, an application that degrades gracefully can provide a lower quality of service rather than no useful service at all. An application's use of the network can be adapted to the network's characteristics in many ways. When bandwidth is an issue, simple lossless compression may help. In more extreme cases, more drastic application-level solutions may be useful. For instance, on-demand distillation (reduction of color depth or resolution) of images can improve the performance of a web browser with an acceptable loss of quality [4]. Bandwidth use can also be prioritized, delaying some transactions in favor of others, or eliminating them entirely by using cached data.

Adaptations can handle other link deficiencies as well. High network latency can be addressed by prefetching data ahead of the user or application's actual need. Selective acknowledgements may be helpful when error rates are high. For insecure links, an entire data stream can be encrypted, or key elements can be obfuscated or filtered.

Application-specific adaptations can often be more effective than generic adaptors. By allowing some data loss, an adaptation that reduces color depth in an image, for example, can reduce transmission cost significantly more than Lempel-Zev compression. Each possible adaptation represents a different tradeoff for the user. Different users and uses of an application may require different notions of proper service degradation.

2.2 Network Heterogeneity and Distributed Adaptation

Until recently, research in adaptive networking has primarily focused on the "last mile" of the network, connectivity to the home or mobile user. It is increasingly common, however, to find deficient links throughout the network. For instance, homes that once had a single computer connected to the network via a modem now have a LAN with multiple computers and other devices. Such a LAN is typically connected to the Internet through a single access point, pushing the deficient link one hop away from the client node. Similarly, in the future mobile users may carry multiple devices that share a single wireless access point via a personal-area-network. Multi-hop wireless networks (like Metricom's Ricochet service [14]) also extend the "last mile," introducing a series of wireless links in the "last mile."

User-to-user services also increase network heterogeneity. Services like IP telephony, instant messaging, multi-user games, and home video-conferencing introduce a pattern of direct communication between users in their homes. Rather than one "last mile," each scenario contains two "last miles."

Finally, although typical commercial servers are well connected, network and server congestion can decrease bandwidth and increase latency. Even the most robust and powerful servers have succumbed to distributed denial-of-service attacks and "The Slashdot Effect." [1]

Because of heterogeneity in the network, providing adaptation at the endpoints of a connection, or at a proxy node, is not always sufficient. Various factors constrain the location for proper adaptation. Some adaptations need to be adjacent to one particular problem link. For instance, an adaptor that drops video frames when a link is congested gains agility by closely monitoring the conditions on that link. Similarly, an adaptor that deactivates a wireless receiver (to save power) when no data is expected must be collocated with the hardware it controls. The presence of several dissimilar problem links may also constrain the manner in which adaptations can be deployed. For instance, if prefetching is required to overcome high latency near the server, and compression is required to overcome low bandwidth near the client, prefetching must not be performed from the client. Finally, deployment of adaptations can be constrained by limited node resources. For instance, while end-to-end compression may provide the most effective use of overall bandwidth, a server may not have sufficient CPU cycles to provide such an adaptation to all clients with deficient links. Other possible points of adaptation can provide the needed load balancing.

As network heterogeneity becomes more common, more sophisticated adaptation deployment will be required. Heterogeneous networks frequently require multiple adaptations to solve multiple simultaneous problems and may introduce constraints on how those adaptations can be successfully deployed. Distributed adaptation allows both detection of end-to-end network characteristics and the deployment of a coordinated solution for multiple simultaneous problems.

[1] A product of Kerbango, Inc. Available at http://www.kerbango.com.
[2] A product of Aplio, Inc. Available at http://www.aplioinc.com.

2.3 Approaches

There are at least three ways to make applications degrade gracefully: employ situation-specific applications, build adaptable applications, or provide an adaptive service within the network.

2.3.1 Situation-Specific Applications

Users of substandard networks may choose applications specially designed for their situation. The PalmVII wireless device, rather than using a general web browser, uses special "clipping" applications [17] that provide a unique interface to specific web pages. Each web page accessible by this device has a proxy server, a host on the wired network that filters responses from the associated web server, and a specially built application that interfaces with this proxy. This solution has obvious 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.

2.3.2 Adaptable Applications

Applications can automatically identify the characteristics of the network and tailor their behavior accordingly. A simple example is the RealPlayer [18], which 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 [8] and Odyssey [16] 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 to adapt transmitted data to prevailing conditions. Odyssey pays particular attention to the issue of supporting multiple networking applications on a single mobile node simultaneously, and to the value of cooperation between the application and the operating system. Rover is an application development toolkit that provides higher-level networking abstractions, simplifying the design of adaptable applications. Rover's communication abstraction reduces network traffic by helping applications move data and code closer to where they are needed, and provides infrastructure for operations while disconnected.

Researchers have also suggested methodologies for partitioning an application to adapt its communication pattern to network conditions. When latency is high, a function that aggregates many individual pieces of data should be placed closer to the data source. When connectivity is intermittent, the required data and the functions that operate on that data should be replicated near the data consumer. Several partitioning methodologies have been proposed [10] [26] for partitioning the communication functions from other application functions. Data and functions involved in communication can then be dynamically migrated as required by network conditions.

Application development tools like those described above 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.

2.3.3 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. Special 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. An application-level analog of Snoop is the Protocol Boosters [13] adaptation framework. Protocol Boosters allow pairs of adaptation modules to be transparently deployed, adding new features to existing protocols, such as forward error correction. Generally, Protocol Boosters are assumed to provide lossless adaptation, since the system provides no support for ensuring reliable deliver 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 [23] use IP tunneling to alter the behavior of a protocol over a troublesome link. Generally, Transformer Tunnels are used to 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 provide support for composition of adaptations.

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. The proxy can provide a variety of application-level adaptations, 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 for use by a particular PDA). The Berkeley researchers have investigated methods of composing adaptations on a single machine [7]. They have also examined how to use a clustered proxy service to provide highly reliable, scalable services to a large number of customers.

Similar to the Berkeley proxy, the Mowgli WWW service [11] improves WWW performance across low-bandwidth, high-latency GSM channels by breaking the communication channel into two segments at a proxy. Between the proxy and the server, standard TCP and HTTP are maintained. Between the client node and the proxy, however, custom protocols replace both TCP and HTTP. By removing unnecessary protocol overheads, compressing and prioritizing data, and filtering unnecessary requests, Mowgli is able to reduce channel demand, smooth traffic to improve utilization, and support disconnected operation. While Mowgli provides a useful service, it is not designed for the extensible addition of new adaptations nor does it make any provisions for reliability in the case of proxy failure.

Another proxy mechanism from researchers at Columbia University [29] provides a general-purpose framework to support mobile clients. A proxy node is placed at the edge of the wired network. A mobile host can dynamically load, execute, and control adaptive “filters” on this proxy. Filtering is allowed at the application layer as well as lower protocol layers. While much more dynamic than other proxy solutions, this system does not address issues of multiple coordinated adaptations, reliability, or security.

Active networks are an attempt to add generalized adaptability into the network infrastructure [24] [27]. Potentially, each packet would execute special code at each visited router to determine its proper handling. In some active network models, packets can exercise only a small set of useful options. In others, an arbitrary action is permitted within security and resource limitations imposed by the network infrastructure. Active networks can provide very general adaptation, but require substantial deployment costs. Key design issues 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, the best results can often be obtained by adapting application-level protocols, rather than providing generic adaptations at the transport level. Research in this area, however, has focused on providing a single adaptation for a single suboptimal link, usually the "last mile."

3 Distributed Adaptation

Applications communicating across heterogeneous networks, including the Internet, may require multiple coordinated and potentially distributed adaptations. This manner of adaptation is not well supported by existing adaptation frameworks. We are developing an adaptation service called Conductor to explore the challenges and rewards of distributed adaptation.

3.1 The Conductor Approach

Like several existing systems, Conductor provides an adaptation service from within the network. However, Conductor allows adaptation to be distributed into the network dynamically, applying several 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 clients, servers, and gateways between networks of differing characteristics. The more Conductor-enabled nodes along the network path between a client and server, the more effective adaptation can be.

Conductor nodes support the deployment of adaptor code modules, which implement particular adaptations. These adaptors operate on application-level protocols, arbitrarily modifying the data stream. 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 than 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.

Conductor allows multiple adaptations to be dynamically deployed throughout the network, where they are needed. 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 allows adaptation of reliable connection-oriented streams. Conductor includes a new model of reliability that is compatible with adaptation, protecting applications from the failure of (potentially stateful) adaptors, Conductor nodes, and links.

These facilities allow Conductor to provide a coordinated and distributed set of adaptations to a data stream. The growing heterogeneity and complexity of networks require such a distributed adaptation infrastructure.

3.2 Case Studies in Distributed Adaptation

Since suboptimal characteristics can be found at many different locations in a network, a given end-to-end connection may contain several links that require adaptation. The following subsections examine in more detail three cases in which multiple adaptations may be required by one connection.

3.2.1 Secure Communication From a Home LAN

In the near future, the average home will likely contain several Internet-capable devices (multiple workstations, TV, radio, refrigerator, etc), possibly connected by a wireless

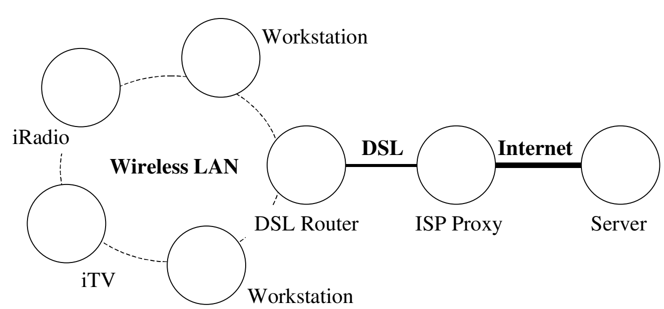

Figure 1: A home network supporting multiple Internet clients.

LAN for internal communication and a DSL router providing Internet access for all devices on the LAN (Figure 1).

If a user of one of the workstations accessed his bank balance, the network would present several concerns. First, the Internet and perhaps also the wireless LAN are insecure. Second, the DSL line might not provide sufficient bandwidth, particularly since several devices are sharing it. Adaptation can be employed to overcome these problems. Several points of adaptation may be useful in this situation: the client, the server, the DSL router, and a "proxy" node provided by the ISP.

Using secure sockets for end-to-end encryption of the data would typically solve the security problem. In this case, however, since the bandwidth of the DSL link is also an issue, a different solution is required.

To improve the performance of web fetches (several of which typically occur at once), the client may wish to prioritize 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. Such prioritization requires access to the data, but end-to-end encryption makes stream access problematic.

To apply the shortest-job-first algorithm to the banking example, the adaptation would need to be deployed immediately prior to the DSL link, allowing HTTP sessions with various servers to be prioritized together. However, all of the data from the bank's web page are likely to be encrypted (both text and images) to reduce the chance of sensitive information being transmitted in the clear. Since this adaptation requires access to the data stream, it cannot be performed on encrypted data.

Two solutions are possible. An end-to-end encryption adaptation that tags each session with its length 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 unencrypted data in between. In each solution, available bandwidth has changed the manner in which encryption should be applied.

Encryption interferes with many other desirable adaptations as well. For example, a user participating in a videoconference may want it encrypted for privacy. If the DSL channel is barely sufficient to support this traffic, the user may be willing to drop some frames when other traffic is present. Doing so requires either individual encryption of each frame, or that the stream be decrypted before the adaptation and reencrypted after. Again, the choice of one adaptation is altered by the presence of another.

3.2.2 A Multimedia Session Between Wireless Users

Mobile users frequently have very low-quality "last mile" links. As mobile network access gains popularity, it will be more common for mobile users to communicate directly with other mobile users. For instance, a mobile-to-mobile videoconference has two low-quality "last mile" links.

Consider two mobile users, both connected via WaveLAN packet radios which provide around 6 Mb/s of bandwidth

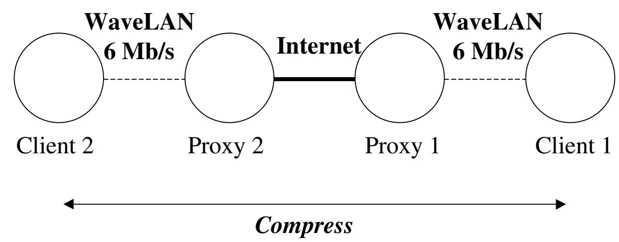

Figure 2: Two mobile users with WaveLAN connectivity.

(Figure 2). The wireless link has insufficient bandwidth to carry the video transmission. To decrease the capacity requirements, 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). Since repeated compression and decompression is inefficient, it is better to adapt from end to end, compressing once at Client 1 and decompressing at Client 2.

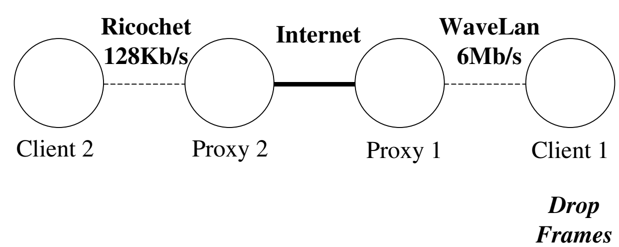

Figure 3: One mobile connected by WaveLAN and one connected by Ricochet.

If Client 2 is, instead, connected via a Metricom Ricochet 2 wireless modem, 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, in place of the video-conference, a video clip were transmitted. Perhaps a cache is present somewhere in the Internet that would service other clients' requests for the same video 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 compression between Client 1 and Proxy 1 and drop frames at Proxy 2, delivering the full fidelity data stream to the cache.

3.2.3 Database Access From the Field

An archaeologist has built an image database of artifacts, stored on high-powered servers. He can take a picture of a new artifact with a digital camera attached to his computer and use it to query the database for similar items. This system works well in the office environment.

The archeologist, like many of his colleagues, is interested in using wireless technologies in the field [2]. Using a PDA with a digital camera, a local area wireless network, and a modem link from the base camp to the Internet, a field researcher can gain access to the same databases available in the lab. The archeologist could generate queries on the PDA, which would traverse the wireless network and modem line to the server. Unfortunately, this setup won't work as well as in the office environment. The large queries and responses transmit slowly over the modem. While waiting for the response to be delivered, the PDA is wasting battery power, particularly to maintain the (potentially unused) wireless link.

Two adaptations are possible to help this situation. First, the data transmitted over the modem can be compressed, reducing transmission time. Doing so requires adaptation at a point on either side of the modem link. To reduce power consumption on the PDA, the base-station should be one of these points. The other point could be either the server across the Internet or a proxy at the ISP.

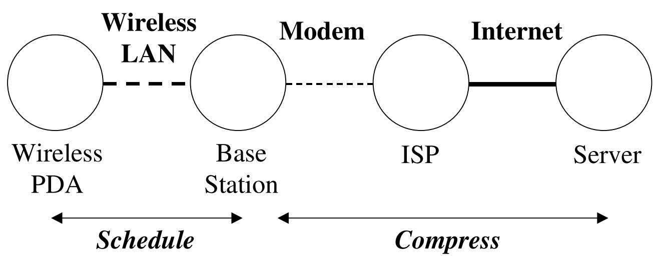

Figure 4: Two disjoint adaptations deployed for one data stream.

Another useful adaptation would be to schedule the use of the wireless link. When a query is generated, the base-station can guess the response time of the server based on the historical response time of similar queries. The base-station can instruct the PDA to shut down its wireless adaptor for that period. Meanwhile, the query is transmitted and the response is received. If the response comes back earlier than predicted, it can be buffered at the base station until the agreed-upon period of silence expires. Then the PDA will reactivate its wireless link and await a response from the server. Note that part of this adaptation must occur on the PDA and part must occur on the base-station, to be as close to the PDA as possible. Thus, at least three adaptation points are required (Figure 4).

The archeologist may also wish to protect his research from competitors, so encryption may be required for transmission over the Internet. Encryption could be deployed between the base-station (or a proxy at the ISP) and the server. If encryption were performed end-to-end instead, it makes the other adaptations difficult. The scheduler may not be able to discern the start and end of each query and response. The compressor will be unable to compress the now randomized data stream.

This scenario is not unique to the field of archeology. Other professionals, such as firefighters, architects, and salesmen, frequently work at off-site locations and require mobile data access. When connectivity is available at all, such users commonly face dynamic and heterogeneous network conditions.

3.3 Advantages of Distributed Adaptation

Heterogeneous network environments, like those explored in the previous section, require more complex support for graceful degradation. Under many circumstances, an adaptation service must provide coordination among multiple points of adaptation within the network to be successful.

If adaptation were only necessary at the endpoints, there would be no need to add complexity to the network. In the example of secure web browsing, however, prioritization of web traffic must occur at the point immediately prior to the slow link. Otherwise, data flowing from different servers cannot be prioritized together. Similarly, in the archeology example, part of the link scheduling adaptation must be present at the base-station, immediately prior to the wireless link, to cache query responses when the wireless link is inactive. If this function were instead deployed on the server, the wireless link would have to be active during data transmission across the modem, allowing no power savings.

In addition, heterogeneous networks sometimes require multiple adaptations. Care must be taken to coordinate multiple adaptations into an integrated solution. In the example of secure web browsing, the desire to adapt to the low-bandwidth link changed the proper choice of encryption deployment. Had security been deployed end-to-end, the desired bandwidth-saving adaptation would not have been possible. On the other hand, while link-by-link encryption would always allow other adaptations to be deployed, this solution is less efficient and requires trust in all intermediate nodes. Similarly, in the example of multimedia for mobile users, if the two wireless links were considered independently, redundant effort would be spent on lossless compression and decompression. If filtering is required by one link, compression and decompression for another link is pure overhead.

Although distributed problems within the network could be addressed by deploying multiple instances of the Berkeley proxy, Protocol Boosters, or Transformer Tunnel systems, the resulting adaptations would be entirely independent. As previously discussed, independently chosen adaptations can conflict, making some adaptations impossible or redundant. Even active networks, which allow nearly arbitrary action to be taken at each point within the network, have no inherent ability to coordinate actions that might be performed at each node.

3.4 Potential Pitfalls of Distributed Adaptation

The advantages of distributed adaptation do not come without cost. 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 end-points as possible [21]. While proxy nodes with visibility into higher-level protocols have been proposed both for adaptive purposes [4] and in firewalls for security, they are not the norm. The previous examples have shown that access to and adaptation of application-level protocols within the network has advantages, but such a service impacts packet overhead, reliability, and transparency.

Building application-level adaptation into network nodes increases packet processing costs. However, adaptation is not required at all network nodes, nor by all packets. For packets requiring adaptation, the processing costs are offset by the benefits obtained. For instance, the time it takes to reduce the color depth of a video frame is made up for in its transmission cost. Since processing speed is increasing faster than network speed, the set of feasible adaptations is growing.

Adaptation causes the network to cease to be a transparent transmission channel. Unaware adaptation might not always be appropriate. In the Conductor model, the user is in control of adaptation. If adaptation affects application behavior unacceptably, a different adaptation can be used.

Finally, deployment into the established Internet can be a major impediment for any new network technology. The Conductor model does not require total network deployment. Instead, Conductor can be incrementally deployed onto nodes that provide the most benefit, primarily those adjacent to deficient links. The additional resources required by adaptation (notably processing and storage) can be deployed as needed to serve the user community. The more resources that are 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 [19].

4 Challenges in Distributed Adaptation

Our goal is to build an adaptation service that helps applications deal with heterogeneous network conditions by supporting distributed adaptation. The service should also be application transparent, supporting both legacy and new applications without requiring a sophisticated understanding of the network. Finally, the system should allow an extensible set of arbitrary adaptations to be employed.

The adaptation service will support reliable, stream-oriented communications, which account for a large portion of common network protocols.

To succeed, such a system will require the following capabilities: the ability to obtain sustained access to target data streams at the points in the network where adaptation is required, the ability to select and deploy a set of adaptors appropriate to prevailing network conditions, and the ability to protect itself from component failures and influence from unauthorized users.

4.1 Stream Handling

Before any adaptation can occur, an adaptation service must be able to identify data streams that are candidates for adaptation. The adaptation service must determine which protocols and data types are being used by a particular connection. Stream identification is possible via the same mechanisms that allow clients and servers to communicate without any prior arrangement: well known-port numbers, protocol identifiers, and magic numbers. Many clients contact a remote service using a well-known port number. The protocol for such a service is known in advance. If a well-known port is not used, or if multiple versions of a protocol exist, the desired protocol will typically be identified or negotiated in an initial handshake between the client and server.

For example, the HTTP protocol is typically, but not always, located on well-known port 80. In either case, the first line of communication from the server to the client contains a string identifying the exact version of HTTP that is supported.

Some application-level protocols (e.g., HTTP, SMTP, FTP) act as transports for a variety of data types. Frequently, the type of the data being transported is identified by the transport (e.g., HTTP's Content-Type header). Even when the data type is not explicitly identified by the transport, most file formats include a magic number in the first few bytes that forms a unique identifier for that file type. For instance, GIF89a files begin with the unsurprising string "GIF89a."

If the adaptation service determines that it can handle a particular data connection, it must identify adaptation-enabled nodes between the client and the server at which adaptation might occur. Perhaps it is best to identify adaptation-enabled nodes along the default routing path (usually the shortest path). On the other hand, other paths may provide more desirable network characteristics or more nodes capable of providing adaptation.

Finally, to deploy adaptation into the network, it must be possible to sustain access to the data stream at the desired points of adaptation. If an adaptation is deployed, but not all of the data flows through that node, the results will be unpredictable. Some mechanism is needed to force the data stream to follow the selected path.

4.2 Planning

For each connection that may need adaptation, a distributed adaptation service must determine which, if any, adaptors are required and where they should be deployed. Planning should be automatic, requiring as little participation from the user and the application as possible. 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.

There are several inputs to the planning process: 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 golf courses 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. Current, as well as historical, conditions may be of interest to the planning process. The adaptation service must be able to query the environmental monitor at planning time and also be notified when drastic changes occur.

The speed of planning and the quality of the resulting plan are key design factors. Planning should occur quickly, so the adaptors can be selected and deployed before data flows. 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. This argument tends to favor decentralized planning, in which each node locally selects and deploys the adaptors it believes to be necessary. Recall however, from the examples in Section 3.2, 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. Due to the potential presence of low-quality links, the communication cost of obtaining a global view should be minimized.

At the same time, since mistakes in planning are expensive, a quality plan must be produced. The desire for plan quality must be balanced against the amount of startup latency that is tolerable by the user.

4.3 Reliability

A distributed adaptation service should not reduce the reliability of a data connection. In particular, connection failure should be prevented in the face of node, link, and adaptor failure. Typically, reliability is provided on an end-to-end basis and the reliability of a given connection 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. Distributed adaptation requires a new model for reliability.

Three approaches to reliability have been incorporated into previous adaptive services. The first approach is to restrict the types of operations that can be performed on the data stream. Both Snoop and Protocol Boosters take this approach. Snoop's actions are designed not to violate TCP semantics, while Protocol Boosters allows only additive operations, transmitting the additional information independently from TCP streams. In either case, failure of an adaptation only reduces or removes the benefit of adaptation. If arbitrary adaptations are to be supported, however, a different approach is required.

A second approach is to increase the reliability of adapting nodes. The Berkeley Proxy allows arbitrary adaptations by splitting the TCP channel into two, from the server to a proxy and from the proxy to the client. While arbitrary adaptations are possible without confusing the reliable transport, failure of the proxy node will result in failure of the connection. This potential for failure is mitigated through the use of redundant hardware and software. While this approach is adequate for a single proxy situated near the last mile, deploying redundant hardware throughout the network is not feasible. In addition, this approach does not address the possibility of 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 [9]) 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 will also cause packet forwarding to cease. 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. In addition, when adaptation is distributed, alternative routes may very well exist, allowing data transmission to continue despite node failures.

When an adaptor fails, the reliability mechanism must ensure that the semantics of the data stream are not violated. For instance, if an adaptor is saving bandwidth by dropping every other frame in a video stream, each frame must either be dropped or delivered. If an adaptor fails in the middle of a frame, it should not be the case that half of the frame is delivered to the application.

Arbitrary distributed adaptation requires a new model for reliability. In particular, the reliability mechanism must be aware that adaptation is occurring. Since arbitrary adaptation has the potential to delete, add, or modify the data stream, the typical notion of exactly-once delivery of data is no longer correct. A new model is required in which intentional data loss (e.g., the non-delivery of a video frame that has been dropped by an adaptor) is not considered to be a failure.

Where possible, implementation of the reliability model should leverage existing technology. A service that exists above or below an existing reliable transport, like TCP, is preferred to building a new transport.

4.4 Security

While a distributed adaptation service aids users in adapting a data stream to meet their needs, this same mechanism might be exploited by an attacker. An unauthorized user might subvert the planning facility, causing it to direct data to flow through a convenient node for interception. Or, the attacker might instruct the planner to deploy adaptations that lower the quality of server, when no adaptation is actually needed. The planning process must therefore be protected from untrusted entities.

In addition, when insecure links are present, the secrecy and integrity of the application data stream may require protection. Encryption adaptations can be employed where needed, but support is required to determine when and where encryption is needed and to aid in key distribution. Clear text and encryption keys should only be available to trusted nodes.

Implicit trust can be given to the end nodes of any connection, since they already have full access to the data stream. One of these nodes should control the planning process. Moreover, any input to the planning process provided by participating nodes must be authenticated. If all inputs to the planning process are digitally signed, then the planner need only determine which inputs are authentic and which nodes are trusted. The results of planning must also be protected in a manner similar to the planning information, signed by the planner and authenticated by each receiving node.

Unfortunately, there is no widely accepted authentication mechanism in the Internet. Even if there were, such a mechanism would not be strong enough for all applications. The adaptation service should, therefore, allow for secure agreement upon what authentication mechanism to use.

Finally, if dynamic deployment of adaptors is permitted, the adaptation service must ensure that (1) the executed code modules are exactly those specified in the plan and (2) the host node is protected from the actions of malicious or buggy code. Protection of the local node should also include enforcement of any local resource constraints (e.g., CPU cycles, storage) imposed by the node during the planning process.

5 Conductor: Proof of Concept

The Conductor adaptation service has been constructed to demonstrate the advantages and feasibility of distributed adaptation. While not yet fully functional, Conductor includes basic facilities for stream interception and routing, formulation of plans, deployment of adaptors, security, and reliability. Using the existing prototype, several useful adaptations have been constructed and deployed, including image compression and distillation, encryption, and web download prioritization. The following sections describe Conductor's current and future capabilities and how it addresses the challenges of distributed adaptation.

5.1 Conductor Architecture

Conductor is composed of two main elements: adaptors and the framework for deploying those adaptors (Figure 5).

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.

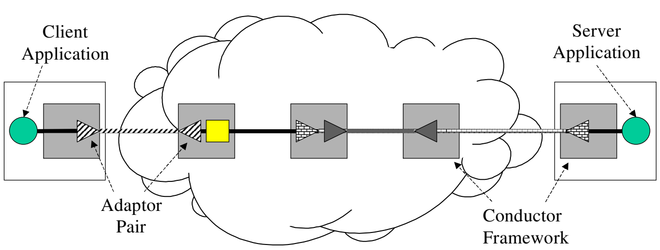

Figure 5: Conductor intercepts client-server communication channels and deploys distributed adaptors.

New adaptors can be developed and 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. The ability to combine adaptors is 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.

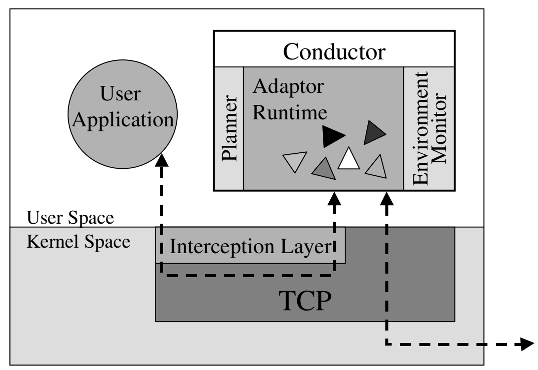

Figure 6: Architecture of a Conductor-enabled node.

The Conductor runtime framework is primarily a Java program running in user space that provides a runtime environment for adaptors (Figure 6). 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.

Each Conductor node includes a planning element that participates in a distributed planning algorithm. The planning architecture is discussed in more detail in Section 5.3. The planning process directs each node to deploy an adaptor along a particular data connection. The Conductor node implements an API for adaptors, which gives the required resources to each adaptor and allows it access to the data stream. The adaptor API is described in Section 5.5. Each Conductor node is equipped to handle failures along the data path, as described in Section 5.4.

5.2 Stream Interception and Handling

Before Conductor can adapt a data stream, it must intercept that stream and determine if it 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 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. In the Linux environment, the interception layer consists of a small kernel modification that traps connections destined for a particular remote port number. In some systems, existing extensibility mechanisms allow trapping of data flows without kernel modifications [12]. 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 [25]. Currently, interception is based entirely on remote port number.

Once 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 toward the server, which follow the typical network route. Conductor-enabled nodes along this route are able to capture this packet and participate in planning for this connection. Once the last node is identified, a connection is created to the intended server, and planning for adaptation occurs. The planning process will 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.

5.3 Automated Planning

Conductor includes a planning facility to automatically select an appropriate set of adaptors and decide where to deploy them. Planning occurs on a per-connection basis. The planning process consists of three main elements: information gathering, plan formulation, and adaptor deployment.

Much information must be obtained from the nodes between the client and server to formulate a useful plan. Conductor provides a pluggable architecture for monitoring local node and link conditions. 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 [22].

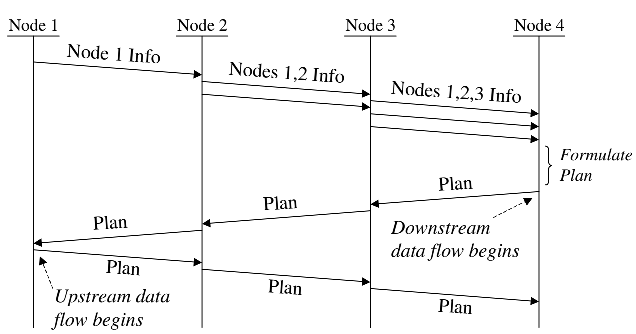

Figure 7: Conductor's planning protocol in action.

Conductor balances the need for a global view with the desire for quick planning by employing 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 7). Currently, this node is the Conductor node closest to the server. The planner node uses this information to formulate a plan. Once formulated, the plan must be delivered to all nodes, allowing adaptors to be deployed. The plan is transmitted from the planner node toward the client endpoint, passing through each Conductor-enabled node along the way. Once the adaptors are deployed on a given node, data may flow in that direction. As a result, once the plan reaches the client node (assuming it is Conductor-enabled), the first bytes of application data from the server will reach the client. The plan will now flow in the reverse direction, 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.

Conductor requires an automated plan formulation service to generate a correct plan from the information gathered. In addition to link characteristics, this service must consider available node resources, the capabilities of available adaptors, and adaptor compatibility in order to generate an appropriate plan. One possible algorithm would treat the set of possible plans as a search space, applying constraints and an evaluation function to find the best feasible plan. Alternately, a set of predetermined templates could be tested against the current node and link conditions to establish a match. Conductor's centralized information-gathering infrastructure allows practically any algorithm to be plugged in. Unfortunately, the number of nodes, possible link characteristics, and number of adaptors lead to a very large search space. Constraints on adaptor combination and nodes that can host them cause the complexity of the problem to grow quickly. Finding an appropriate set of adaptors within an acceptable time is a great challenge [20]. For the purposes of this research, an algorithm that generates acceptable plans for the small variety of cases in our limited deployment environment will be sufficient.

Planning occurs at connection start-up time based on the current network conditions. Since network conditions can be dynamic, replanning may also be necessary. Due to the moderate cost of altering the set of deployed adaptors, Conductor is primarily interested in failures and extreme variations in available resources. 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.

Conductor's information-gathering protocol is fully functional; however, the plan formulation facilities are somewhat primitive. Additional research is required to obtain an appropriate algorithm for automatic planning.

5.4 Reliability

TCP guarantees exactly-once, in-order delivery of a byte-stream, a convenient model for application writers. However, this model is incompatible with adaptation. Adaptation seeks to deliver some version of the transmitted data that is cost-effective, so the bytes delivered may bear no resemblance to the bytes transmitted. Conductor requires a new model of reliability that instead provides exactly-once, in-order delivery of *adapted* data.

5.4.1 Redefining Reliability

Most reliability models assume that data is immutable during transmission. Each byte transmitted 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.

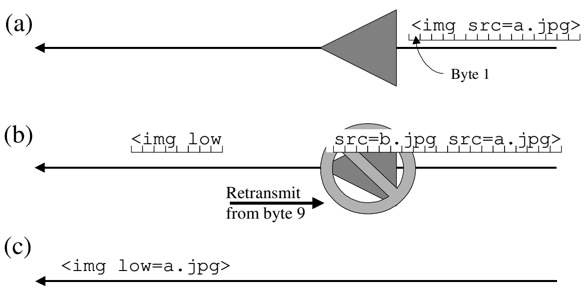

Figure 8: Failure recovery using a byte-count – (a) data arrives at adaptor, (b) failure and retransmission occur, (c) retransmission produces an undesirable result.

Adaptation violates this model's implicit assumption of data immutability. When a failure occurs, the system must ensure that the resulting data stream conforms to the adapted protocol expected by the application. Maintaining data integrity is not simple, however. An adaptor may fail at a point in its output stream that is inappropriate for switching back to the original stream. For instance, consider an adaptor that adds a *lowsrc* attribute to an *image* tag in an HTML data stream (Figure 8). 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 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 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 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, 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 before adaptation can 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 some acceptable representation of 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. Segments must be chosen so that the failure of any adaptor will not result in delivery of data that confuses the destination application. These properties ensure that some viable version of the data produced at the source will arrive at the destination.

5.4.2 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 provides a data recovery mechanism to protect 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.

5.4.3 Semantic Segmentation

A central and unique contribution of this research is the notion of *semantic segmentation*, which allows data recovery despite the presence of adaptation. 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. Conductor considers these bytes to be logically segmented 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 the example of an adaptor that compresses video frames. Before each frame can be compressed, the segments making up that frame would 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. 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 numbered 100 and another numbered 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.

Subsequent adaptors may cause further segment combination, and segments may grow to arbitrary length. 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 below).

5.4.4 *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. 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.

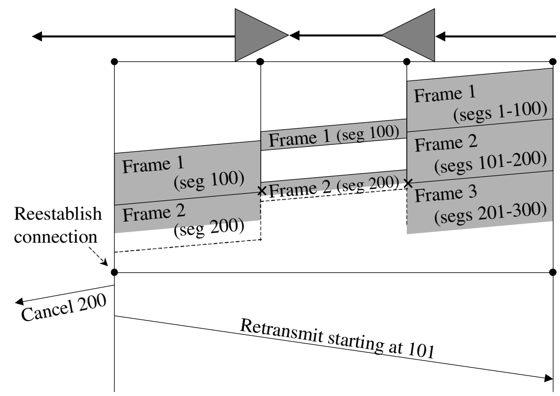

Figure 9: Recovery from the failure of an adaptor that compresses frames in a video stream.

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) were received at a point immediately downstream from the decompression adaptor (Figure 9). If both the compression and decompression adaptors fail, Conductor would discard the part of segment 200 that was partially received downstream and request retransmission starting at segment 101. Retransmission would thus begin with the second frame, as desired. If only one of the two adaptors fail, Conductor would automatically remove the other to preserve adaptation symmetry.

The above recovery scheme provides failure-based recovery. Since a retransmission request occurs when there is a failure and indicates exactly which data must be retransmitted, acknowledgements (beyond what is already provided by TCP in the underlying connection) are not required. However, to limit cache growth and allow adaptors to free any accumulated internal state, 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 [28].

5.4.5 *Implementation Status*

Conductor's reliability mechanism is partially implemented. Conductor is able to create and properly maintain semantic segments. Data is cached at the end nodes should retransmission be required. Implementation of the recovery protocol and an API for requesting adaptor rollback is in progress.

5.5 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 has exclusive access to a portion of the data stream defined by a window. Access to portions of the stream outside the window is prohibited. 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. Thus, the adaptor controls the flow of data by moving the window boundaries along the data stream.

An adaptor is also able to view and modify the data stream by other window operations. These operations implement and help enforce 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, 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.

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. However, an adaptor must still ensure that segments are semantically meaningful and delineate appropriate places for adaptation to cease, or a failure may result in a meaningless data stream.

Finally, adaptors have access to two 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. These facilities allow an adaptor to store any object, tagged with an identifier. Another adaptor can then obtain that object using the given identifier. Synchronization is provided, allowing an adaptor to block until a desired object is present. Adaptors should treat this cache as soft-state that may be purged if the cache becomes full.

5.6 Security

Conductor provides an extensible architecture for securing both the planning process and the user's data. The planning process is protected using a "security box." All planning-related messages sent and received by a Conductor node must pass through the security box. Many security box implementations are possible, each providing a particular level of message authentication, protection from replay, and possibly secrecy. The level of protection provided depends entirely on the particular security box implementation.

We have constructed security boxes that use public key encryption to digitally sign and authenticate each message. One such security box assumes the presence of a hierarchy of certificate servers, allowing any node to provide a certifiable copy of its public key to any other node. A different security box assumes no such hierarchy exists and attempts to form a "chain of trust" from which a certified copy of a node's public key can be obtained. Other boxes could be constructed based on entirely different authentication mechanisms, such as Kerberos [15]. A security box that provides no security at all has also been implemented.

Since there is no ubiquitous authentication mechanism and because each connection may require a different level of protection, Conductor simultaneously supports many different security box implementations. For a given connection, the client node selects the desired security box and each Conductor node will use that scheme for all planning messages (or not participate).

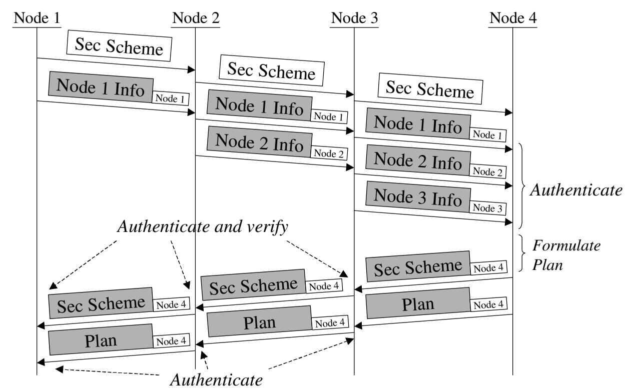

Figure 10: The secured planning process. Messages are signed with a dynamically determined authentication scheme.

Conductor provides a secure mechanism that uses the selected security box itself to ensure that every node has used the same security box throughout the planning process. When planning begins, a message indicating the selected security scheme is sent unprotected from the client node along the path (through all participating nodes) to the server node (Figure 10). The planner uses this scheme to authenticate messages from all nodes, discarding any information that cannot be authenticated. Once the plan has been generated, it is delivered, along with an indication of the security scheme used, to each node for execution. Both the plan and the security scheme identifier are protected by the security scheme. Each node can therefore securely determine what mechanism the planner used and ensure that the expected scheme was used. If a node is tricked by an attacker into using a different security scheme, that node will not be trusted by the planner. If the planner is tricked into using a different security mechanism, the client node will notice that the requested security mechanism was not used. Additional mechanisms are also provided to protect against replay attacks.

Conductor is also able to deploy encryption adaptors whenever the secrecy or integrity of the data stream must be protected. The planner itself is responsible for deciding where any encryption and matching decryption adaptors are required. Successful deployment of these adaptors, however, typically requires some form of key distribution to establish a shared secret. Keys can be generated on the planning node, which is automatically trusted, but must be securely distributed to the nodes performing encryption and decryption. Typically the security box can aid in key distribution using the same mechanism used for authentication. For instance, if public key cryptography was used for authentication, the public key for the target node can be used to encrypt the key, ensuring that only that node can obtain the key. A separate copy of the key is encrypted for each node that requires it.

Conductor will not directly address the issues of security for mobile code. The currently accepted solutions to these problems, digitally signed code and sandboxed execution, could be employed and are available in the Java security model [6].

6 Measurement of Success

To be successful, Conductor must (1) provide effective adaptation in the face of deficient networks, (2) introduce little or no overhead when network conditions are good, (3) provide adequate services in support of transparent, distributed adaptation, and (4) demonstrate overall usability.

Conductor's ability to provide effective adaptation will be demonstrated by constructing several of the examples of heterogeneous networks requiring distributed adaptation described in Section 3.2. Various user-visible performance characteristics will be measured in trials with and without Conductor-enabled adaptation. These experiments will determine the level of end-user benefit gained by using Conductor.

While Conductor overheads will already be factored into the above experiment (reducing the measured benefit), Conductor must also have an acceptable level of overhead when adequate network capabilities exist and no adaptations are required. Conductor's impact on network protocols when no adaptation is required will be measured. In addition, the minimum overhead of introducing additional Conductor-enabled nodes and additional adaptors will also be measured.

Conductor provides dynamic planning and reliability facilities that allow it to cope with changing network conditions. We will show through demonstration that Conductor can (1) dynamically select an appropriate set of adaptors under a variety of network conditions, (2) cope with a drastic change in network conditions by performing reselection and redeployment of adaptors in the middle of a network connection, and (3) provide resilience to component failure by recovering from the loss of a node providing adaptation.

The overall usability of Conductor will be demonstrated through deployment in an office environment. We will construct an environment that requires distributed adaptation and use it on an everyday basis. By developing and deploying a variety of useful adaptations in this environment, all of Conductor's capabilities will be exercised. Successful office use will demonstrate that Conductor possesses the core set of capabilities required for distributed adaptation

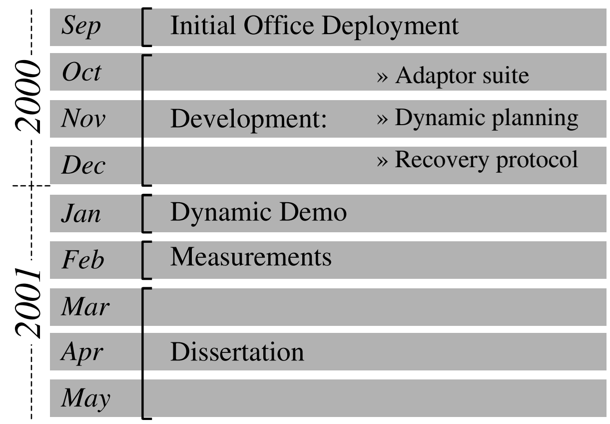

Figure 11: Completion schedule.

7 Completion Schedule

A great deal of Conductor has already been developed, and it is nearly ready for initial office deployment. The remaining work is expected to proceed according to the schedule in Figure 11.

Once initially deployed in a limited capacity, work will continue on increasing Conductor's capabilities. A suite of new adaptors is needed to make use of Conductor by office members more compelling. A more sophisticated planning capability is needed to generate plans dynamically based on a user's current connectivity characteristics, rather than using statically defined plans. Finally, implementation of the recovery algorithm must be completed.

Once the bulk of development is done, a demonstration of Conductor's ability to deploy adaptations dynamically, adjust as conditions change, and recover from component failure will be developed. Finally measurements of Conductor's ability to improve user-perceived performance and Conductor's overheads will be measured. After the research phase is completed, a dissertation will be written to report on Conductor's technologies and the results of the above experiments and experiences.

8 Future Work

While we plan to construct a planning algorithm that can automatically select an appropriate set of adaptors for various conditions that can occur in our test environment, the more general planning problem is much harder to solve [20]. Additional research is needed to find a planning algorithm that can search the very large planning space without an excessive increase in startup latency.

Conductor permits user-input into the planning process to help determine which adaptors to select based on which data characteristics are most important to the current user task. Conductor's current user control interface is limited, and human factors are important in this case. A richer interface is not necessarily a better one: the impact of dropping B-frames in a video transmission is probably little known to the average user. An effective user interface for control over adaptation is an open issue. At the same time, although Conductor is designed to transparently select appropriate adaptations to improve the behavior of an application, knowledgeable programs could probably give Conductor better advice than its planner could deduce on its own. A "Conductor-aware" application interface is needed to allow a more seamless interface to the user.

In general, determining whether two adaptors are compatible is hard. A given adaptor may unknowingly perform operations that render another adaptor ineffective. The problem can be lessened by limiting the possible set of adaptors or what adaptors can do, but ideally possibilities of improving data flows should not be discarded because they may have unforeseen side effects. We are investigating methods of allowing rich adaptors that can be properly and automatically combined.

Conductor allows individual nodes to allocate a limited portion of their local resources to a given connection. In general, greater control of resource allocation may be desirable. For instance, we may want to limit the total resources consumed by a given connection across all nodes. Or we may want to limit the amount of network resources available to a given user across all connections. Achieving these goals is difficult due to the potential overheads of distributed load balancing and user authentication and tracking. These issues are important in the context of active networks, and research into accounting for many small, distributed costs is ongoing. In the future, it should be possible to leverage these results and provide similar facilities in Conductor.

9 Conclusions

The availability of new network technologies will almost certainly increase the heterogeneity of networks over the coming years. To provide useful service in this environment, applications will have to allow their services to degrade gracefully with prevailing network conditions. Distributed adaptation can play an important role in enabling graceful degradation of applications in heterogeneous environments. Conductor provides an important step in this direction by exploring the technologies required for success.

When complete, Conductor will provide a technological step forward in several respects. Conductor will be the first design and the first implementation of a transparent distributed adaptation service. Conductor will include all of the necessary services to coordinate the resources of points of adaptation throughout the network. Conductor will introduce a unique model for reliable delivery, enabled by *semantic segmentation*, that can accommodate the presence of adaptation. Finally, Conductor will include a new mechanism for ensuring trusted coordination between a disjoint set of nodes. These unique features will allow the introduction of distributed adaptation as a reliable and secure network service.

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] M. Ancona, G. Dodero, C. Fierro, V. Gianuzzi, V. Tine, A. Traverso, "Mobile Computing for Real Time Support in Archaeological Excavations," *Proceedings of Computer Applications in Archaeology*, University of Birmingham, UK, April 1997.

[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] 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, September 1998.

[5] 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.

[6] 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.

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

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] A. Mallet, J. Chung, and J. Smith, "Operating Systems Support for Protocol Boosters," *Proceedings of the 3rd International Workshop on High Performance Protocol Architectures (HIPPARCH '97)*, June 1997.

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

[15] B. Clifford Neuman and Theodore Ts'o, "Kerberos: An Authentication Service for Computer Networks," *IEEE Communications Magazine*, 32(9):33-38, September 1994.

[16] 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.

[17] Palm, Inc., "Web Clipping Applications Web Developer's Tutorial." Available at http://www.palm.com/devzone/webclipping/tutorials/turorial_web.html

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

[19] 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.

[20] Peter Reiher, Richard Guy, Mark Yarvis, and Alexey Rudenko, "Automated Planning for Open Architectures," Short paper presented at *The Third IEEE Conference on Open Architectures and Network Programming (OPENARCH '00)*, Tel-Aviv, Isreal, March 2000.

[21] 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.

[22] 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.

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

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

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

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

[27] 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.

[28] 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.

[29] 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.