Why Path-Based Adaptation? Performance Implications of Different Adaptation Mechanisms for Network Content Delivery

(NYU-CS TR2003-843)

Xiaodong Fu and Vijay Karamcheti
Department of Computer Science
Courant Institute of Mathematical Sciences
New York University
{*xiaodong,vijayk*}*@cs.nyu.edu*

Abstract

Because of heterogeneous and dynamic changing network environments, content delivery across the network requires system support for coping with different network conditions in order to provide satisfactory user experiences. Despite the existence of many adaptation frameworks, the question that which adaptation approach performs the best under what network configurations still remains unanswered. The performance implication of different adaptation approaches (end-point, proxy-based and path-based approaches) has not been studied yet. This paper aims to address this shortcoming by conducting a series simulation-based experiments to compare performance among these adaptation approaches under different network configurations. In order to make a fair comparison, in this paper approach-neutral strategies are proposed for constructing communication paths and managing network resources. The experiment results show that there are well-defined network environments under which each of these approaches delivers its best performance; and among them, the path-based approach, which uses the whole communication path to do adaptation, provides the best and the most robust performance under different network configurations, and for different types of servers and clients.

1 Introduction

Recent advances in network technology have made it possible to deliver real-time rich media content across the Internet. However, despite increasing bandwidth, the Internet still remains a best effort platform, and exhibits considerable heterogeneity and highly dynamic resource availability for individual network paths. This is especially the case when we consider ubiquitously used mobile devices and a variety of wireless networking options ranging from Bluetooth [5] to Wireless 3G [11]. For example, a typical network path between a mobile user and the visited Internet server may include several different types of links with very different bandwidth, delay, and error characteristics, ranging from high-bandwidth WAN links to wireless connections. Similar heterogeneity can also be observed on end devices, which may possess very different computation/storage capabilities. These differences, combined with the fact that the load on each of these network resources can change constantly, is a major cause for the unsatisfactory performance perceived by end users.

To cope with the above problems, a widely used approach in current-day network services is to provide differentiated service for different kinds of (last-hop) network links and end-devices. For example, most media sites usually supply several *versions* of content for different client groups that use different connection options to access the Internet. These versions usually have different requirements on bandwidth and client computation capacity. Although this approach can improve user experience to some extent, the view that clients can be placed into a few fixed classes with constant characteristics is incompatible with the fact that availability of network resources can change over small timescales. For example, the bandwidth to a mobile device can change continually and dramatically due to the mobility of users or the dynamic load of the shared network environment.

More promising approaches to address this problem rely on a general framework that provides system support enabling applications to *adapt* to different network conditions. Examples of such systems include Rover [6], Odyssey [9], ActiveProxy [2], Conductor [12], Active Names [10], Ninja [4], CANS [3] and Scout [8], to name a few. A common property of these systems is that the data in transmission can be processed on the fly with various operators (in some cases, routed via) to match different network conditions. Some of the systems, such as CANS [3], also provide built-in support for dynamic modification of these operations. Compared with the static nature of the differentiated services solution, these systems because of their flexibility can deliver better user experience. Additionally, because of their general structure, such solutions can be applied to a wide range of applications.

Despite these common benefits, the general frameworks described above are quite different from each other. An important difference has to do with the location in the network where adaptation functionality is introduced. With regards to this issue, the systems above can be grouped into three categories: *end-point* approaches (e.g., Rover [6], Odyssey [9]) where only client and server nodes are involved in adaptation; *proxy-based* approaches (e.g., Ninja [4], Active Service [1]) where a proxy node, usually close to the client side, is used to process data, possibly with the cooperation of client nodes; and *path-based* approaches (e.g., Active Names [10], Conductor [12], CANS [3], and Scout [8]), where adaptation can happen at arbitrary intermediate locations in the network in addition to the server, client, and proxy sites as above.

A natural question to ask is whether these three categories of approaches bring any special advantages or disadvantages. Unfortunately, although previous work has shown that the use of such adaptation systems does result in better performance and user experience as compared to situations when no adaptation is provided, the performance implication of these different approaches has not really been studied in any detail. In particular, the answers to the following questions are not readily available: How is the performance that is achievable constrained by the location of adaptation? Under which network conditions is one kind of approach preferred over another? Is the additional complexity of path-based approaches, which requires control over several locations in the network, really necessary? Answering these questions is important, not only for understanding the performance of existing adaptation frameworks, but also for building appropriate solutions as new devices and networks are deployed.

The study presented in this paper aims to answer these questions. We extensively simulate the behavior of different adaptation approaches in the context of a large scale network topology, characterizing in detail the performance and resulting client behaviors. The study looks at how different approaches fare for different assumptions about server computation capacity, client capacity and connectivity options, and request distribution between clients and servers. A key aspect of our methodology is the use of approach-neutral mechanisms for constructing network paths and managing network resources, which attempt to make a fair comparison by maximizing the performance of each approach given its individual constraints of where network adaptation can happen.

The principal results from the study are summarized below. The key finding is that there are well-defined network environments under which each of the approaches delivers its best performance. More importantly, each approach, with one exception – the path-based approach, also exhibits performance shortcoming in certain environments. Specifically,

- The end-point approach works well in network environments where the server sites have plenty of computation capacity, but performs poorly in situations where server sites have limited resources. From the client perspective, the end-point approach provides considerably better performance for clients using high bandwidth network connections.

- Compared with the end point approach, the proxy-only approach exhibits little bias towards different types of servers or clients. However, restricting the adaptation to take place just before the last hop can result in a significant waste of network resources, bringing down the performance of the entire network under high loads.

- Path-based approaches combine the benefits of both end-point and proxy based approaches, providing the most robust and best performance under different network configurations. This behavior mainly stems from the fact that path-based approaches can take advantage of surplus resources wherever they may be available in the network, something that the other approaches are incapable of doing.

The rest of this paper is organized as follows. In Section 2, we briefly review general adaptation frameworks for content delivery across heterogeneous networks. In Section 3, we describe the simulation scenario used in this study. Section 4 presents path creation and resource management strategies that can be uniformly applied to all three

adaptation approaches to optimize their performance, enabling fair comparison. The results from the simulation study are presented and analyzed in Section 5. We conclude in Section 6.

2 Background: Adaptation Mechanisms for Content Delivery

For content delivery applications, user experience is directly related to the underlying network conditions. While an ideal network for such applications should provide guaranteed QoS, today's Internet still remains a best-effort platform for delivering data packets. Consequently, to provide satisfactory user experience, applications themselves have to cope with various problems caused by heterogeneous and continually changing resource availability. Adaptation is one way to achieve this objective.

In contrast to application-specific mechanisms, general adaptation frameworks provide application independent solutions to cope with different network conditions, typically by transcoding the data using various operators prior to its delivery. Such approaches are attractive because of their applicability to a diverse range of applications. Furthermore, such approaches can also simplify the construction of solutions targeted towards one specific application domain. Despite a great deal of commonality among the proposed approaches, they exhibit significant variation along one dimension: the *network location where adaptation can take place*. It is this dimension that our study focuses on. With respect to adaptation location, general adaptation frameworks can be placed into three groups: end-point approaches, proxy-based approaches and path-based approaches.

End-point approaches (e.g. Odyssey [9], Rover [6], and InfoPyramid [7]), use only client and server nodes in adaptation. For example, Odyssey [9] allows client applications to register their expectations of resource availability to the underlying framework. The framework notifies the application whenever the resource expectation can no longer be met. The client application can then respond to such notifications, say by changing the fidelity of data in transmission. The cooperation protocol to change the data fidelity level is handled by the server and a component on the client side called a Warden.

For *proxy-based approaches* (e.g. Ninja [4], Active Proxies [2], and Active Services [1]), instead of server nodes, shared proxy nodes are used to cope with different network connections and end devices. For example, Active Proxies [2] proposed the use of cluster-based proxies, usually placed close to the client nodes, to perform aggressive computation such as content distillation and transcoding on-the-fly to cope with client-side variation (including network, hardware, and software used by the client). The Active Proxy work also comments that proxy-based approaches offer an advantage over end-point approaches in that using a large central server for performing content transcoding operations is more efficient than a collection of small ones. Our results partly agree with this comment, however, as we shall discuss later, the truth of this statement is closely dependent on assumptions about expected resource utilization in other parts of the network.

Compared with the first two approaches, *path-based* approaches (e.g. Conductor [12], Active Names [10] CANS [3], and Scout [8]) are more general in the sense that any node along a network path can participate in adaptation. An example is our CANS infrastructure, which provides mechanisms for automatic construction and reconfiguration of network aware paths according to high level application performance requirements. CANS paths consist of operators (called *drivers*) that can do various operations such as filtering/transcoding, reconnection, and rerouting. CANS algorithms dynamically control the deployment of drivers to network resources so as to match application performance requirements with underlying network conditions.

Previous studies on these systems have already shown that using these adaptation frameworks does result in better performance, compared with the situation where no adaptation is performed. However, to the best of our knowledge, a comparison of the suitability of these approaches under different network configurations, especially from a performance perspective, has not yet been studied. The work presented in this paper attempts to address this shortcoming to better understand both how constraints on adaptation location affect overall performance and conversely, to determine how best to allocate network resources so as achieve the best performance from a given approach.

Although the focus of this study is on the performance impact of where adaptation happens in the network, we note that the ultimate choice of one adaptation strategy over another is also affected by additional factors such as security and/or management overheads. With regard to these latter considerations, some approaches, such as end-point approaches appear to offer clear advantages over others. However, given that the objective of adaptation in content delivery is to satisfy performance-related quality requirements, one needs to always balance the performance implications of the decision against the other factors. The motivation behind this study is to provide input to allow this

tradeoff to be made in an informed fashion.

3 Methodology and Simulation Scenario

In order to study the performance of different adaptation approaches under different network conditions, we adopt a simulation-based methodology. Using a detailed simulator modeling a typical large-scale network where multiple concurrently-active clients download media content from server sites, we characterize the performance of the three approaches — end-point, proxy-based, and path-based. We provide an overview of our simulation scenario and performance metrics of interest below, deferring a detailed description of the specific parameters to Section 5.

Simulated Network. The network modeled in our simulator is depicted in Figure 1. The network contains multiple ISP regions, each of which is modeled as a centralized gateway/proxy node providing a connection to the Internet backbone. The server and client nodes in the network are attached to one of these ISP nodes using various connectivity options. Available options for client nodes include: wired links with sufficient bandwidth (e.g. T1, T3, or ADSL), wired links with limited bandwidth (e.g. dialup connections), and shared wireless links (e.g. IEEE 802.11b). Server connectivity options include two types of higher bandwidth connections (corresponding to OC-3 and OC-12 links). Server and ISP nodes are allocated portions from a fixed computation budget, which is divided up as discussed in Section 4.2 to individually optimize the performance of each approach.

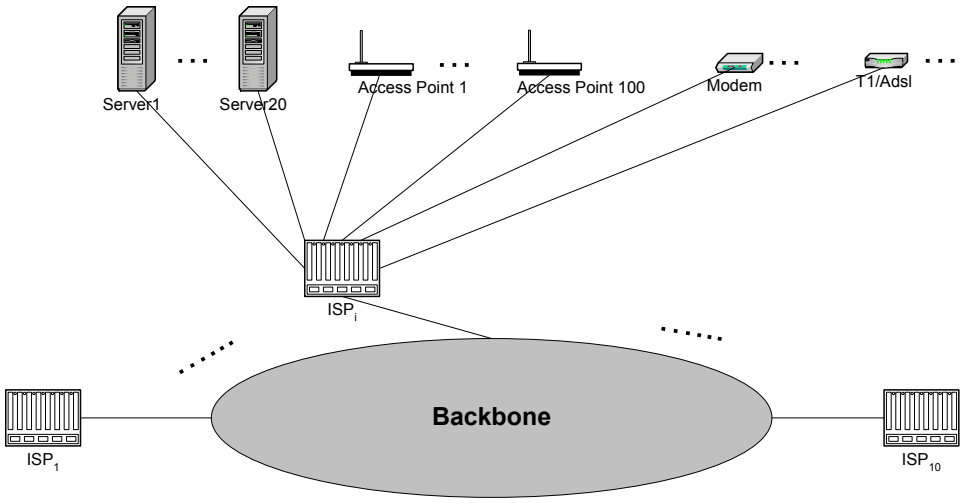


Figure 1: Experiment Network Topology

Application Behavior. The simulation models users connecting to server nodes from client nodes to download and display streaming media content. The connection is released once the download session is completed (which can happen either after the content is completely downloaded, or if the download task is cancelled by the user). To display the received content appropriately, the throughput of a download path is required to be in some specific range (i.e. a certain number of frames per second). When the available bandwidth is not sufficient to meet the requirement, several operators can be used to reduce bandwidth consumption (e.g. by filtering the content).

Performance Metrics. In our experiments, we extensively examined the behaviors of different adaptation approaches by varying different parameters affecting the above scenario: load level in the network, load distribution among server sites, and client connection options. The primary metric used to characterize the performance in each case is the aggregate time (across all sessions) when the throughput of the path is in the required range. A secondary performance metric is the total number of connection failures as a result of an admission control policy that actively rejects new connections when the available resources are insufficient for sustaining additional connections.

4 Approach-Neutral Strategies for Path Creation and Resource Management

Since the focus of this study is on the performance achievable by different adaptation approaches, a key challenge is to ensure that the comparison between approaches is as fair as possible. This entails ruling out factors that are not inherent to the adaptation approach being evaluated, but which can nevertheless impact performance. For the simulation setting outlined in the previous section, there are three such factors:

1. Given an allocation of network resources for a particular connection, how should one select the operators to make up the adapted path?
2. How should shared network resources be allocated among the multiple connections that need to use them?
3. What is the relative computation capacity of server and proxy sites?

Decisions on the first two factors directly affect the performance achievable by an approach (and for reasons that have very little to do with the study's central issue of adaptation location), while the third factor has an indirect affect by influencing where adaptation operators can be placed. [1] What is required in each case are strategies that permit each adaptation approach to achieve its best possible performance.

In the rest of this section, we present strategies that meet the above objective. These strategies have been developed and tested in the context of the CANS system [3], but are equally applicable to the other general adaptation frameworks mentioned earlier. We start by introducing a common model for operators, resources, and data paths, and then describe in turn, algorithms for path creation, resource allocation, and resource distribution.

A Model for Operators and Adapted Connections

An **operator** is an entity that processes data in transmission, sometimes transforming it from one type to another. Each operator is modeled in terms of its *computation load factor* ($\mathrm{load}(o)$), the average per-input byte cost of invoking the operator, and its *bandwidth impact factor* ($\mathrm{bwf}(o)$), the average ratio between input and output data volume. For example, a compression operator that reduces the size of input data by a factor of two has a $\mathrm{bwf} = 2.0$.

Each **node** n_i in the network is modeled in terms of its computation capacity, while a **link** between two adjacent nodes is modeled in terms of its bandwidth capacity. For nodes or links that are shared by multiple paths, each individual path is assigned a share of the total capacity. A **route** is a sequence of nodes and links between the client and server nodes.

A **data path** is a sequence of type-compatible operators. Type compatibility is defined in a **type graph** G_t, whose vertices ($t \in V(G_t)$) represent types, and edges $e = (t_1, t_2) \in E(G_t)$ represent an operator that can transform data from type t_1 to type t_2. Only operators that have compatible types can be connected together. Such a typed view has been used in several systems such as CANS [3], Ninja [4] and Conductor [12].

A **mapping**, $M : D \rightarrow R$, associates operators on a data path D with nodes in the route R. The three adaptation approaches place restrictions on this mapping, by defining which nodes along the path can receive operators.

4.1 Automatic Path Creation Strategy

The path creation strategy answers the first of the questions posed at the start of this section, namely given an allocation of network resources for a particular connection, how should one select the operators to make up the adapted path. The strategy, which can be uniformly applied to end-point, proxy-based, and path-based approaches, simultaneously selects and maps operators to the route so that the performance of the resulting data path is optimized. In the remainder of the paper, we will use the terms *planning* and *plan* to refer to the application of this strategy and the produced output respectively.

[1] This issue is more important for path-based approaches as opposed to end-point or proxy-based ones. For the latter, the resource distribution question is simply resolved in favor of biasing the sites where adaptation takes place.

The input to planning includes a route R, a source data type t_s, a destination data type t_d, and the type graph G_t representing m types and n operators. The plan defines both the data path D that transforms t_s to t_d and its mapping to R while yielding the best performance (e.g. maximal value of throughput or minimal value of latency etc.).

Obtaining an optimal solution to the planning problem is NP-hard. To make the problem tractable, we simplify it in two ways. First, we restrict our attention to mappings that satisfy the restriction that $M(o_i) = n_u, M(o_{i+1}) = n_q \Rightarrow u \leq q$, i.e., components are mapped to nodes in path sequence order. The intuition behind this is sending data back and forth between nodes in a route usually results in poor performance and is wasteful of resources. Second, we view computation capacity as being partitioned into a fixed number of *discrete* load intervals; i.e., capacity is allocated to components only at interval granularity. This practical assumption allows us to define, for a route R with p hosts, the notion of an *available computation resource vector*, $\vec{A}(R) = (r_1, r_2, \ldots, r_p)$, where r_i reflects the available capacity intervals on node n_i (normalized to the interval [0,1]).

Base Dynamic Programming Strategy The above simplifications make the planning problem amenable to a dynamic programming solution. Each step of the algorithm constructs for different amounts of route resources, optimal solutions with $i + 1$ (or fewer) operators, using as input optimal partial solutions involving i (or fewer) operators. The step i solution is representable as $s[t_s, t, \vec{A}, i]$, and denotes the data path that yields the best performance for transforming the source type t_s to type t, using i operators or fewer and requiring no more resources than $\vec{A}$.

The algorithm is described in detail in Appendix A. The main observation is that each step involves enumeration of a fixed number of candidate solutions because of the fixed number of operators, types, and resource vectors. Additionally, because of the mapping restriction above, only resource vectors of the form $(1, ..., 1, r_j \in [0, 1], 0, ..., 0)$ need to be considered. As described in Appendix A, the optimal solution can be identified simply by expressing the overall performance of the partial data path in terms of the characteristics of individual operators and their mappings.

The complexity of this algorithm is $0(n^3 \times p^3)$ as opposed to $O(p^n)$ for an exhaustive enumeration strategy. In most scenarios, p is expected to be a small constant, with overall complexity determined by the number of components.

Planning for Range Metrics

The base algorithm can easily be extended from planning for maximal/minimal values of performance metrics to ensuring that the value of the metric is within a certain *acceptable range*. Only after this range has been met does the application worry about other preferences. For example, most media streaming applications (include the application used in our experiments) require that the data transmission rate fall in some suitable range so that media data can be rendered appropriately at display devices; once this is true, other factors such as data quality become the concern for the application. We use the terms *range metrics* and *performance metrics* to refer to the two types of preferences.

The basic idea behind planning for range metrics is simply stated. Given that the algorithm constructs data paths by incrementally filling in a solution table of $s[t_s, t, \vec{A}, i]$, it is natural to extend this to check that retained solutions satisfy two conditions: (1) values of range metrics achieved on the current partial solution lie within the desired range, and (2) the value of any performance metrics is in fact optimized. Heuristic functions help choose among candidate paths that can all meet the required range.

Additional work is required for some range metrics, such as path latency, where although the current value of the metric may not fall within the desired range, there exists the possibility (through insertion of additional operators) that such partial paths may become a part of the final solution. These candidate solutions are identified by running the planning algorithm in reverse in a process called *complementary planning*. The latter provides information about whether or not there exist operator mappings that would allow the range metrics to reach the desired range using residual resources along a data path. Note that complementary planning needs to be run just once for the whole planning process.

Note that although the algorithm described above strives for optimality, this fact is less crucial for the conclusions drawn from this study. More important is that this approach can be applied to all three adaptation approaches without introducing any bias into the comparisons.

4.2 Resource Sharing among Multiple Paths

We now describe a strategy that answers the second of the questions posed at the start of this section, namely how should shared network resources be allocated among the multiple connections that need to use them. The goal is to ensure that optimized performance is delivered to as many paths as possible.

To achieve this, we first need to understand how an individual network-aware path behaves under a shared network

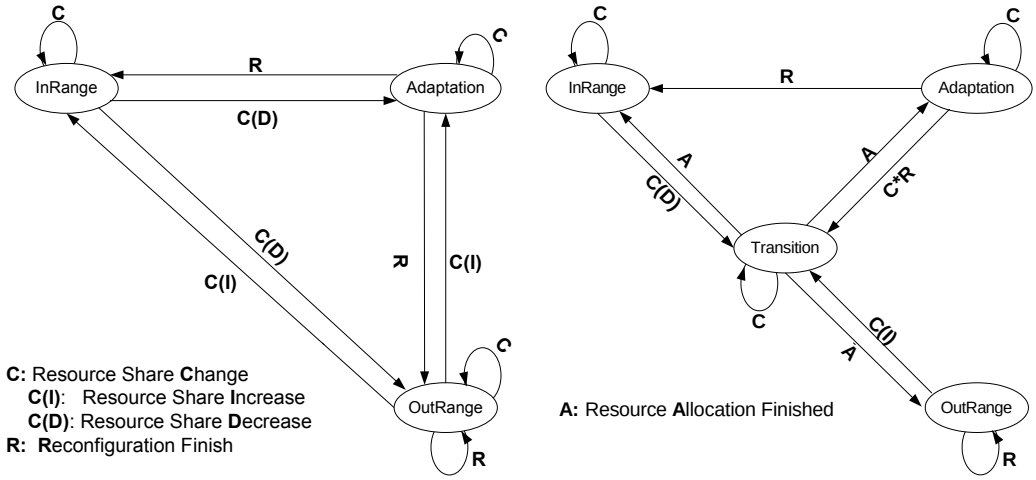


Figure 2: (a) General state transitions for an individual Path (b) In our scheme.

environment. Figure 2(a) shows the state transitions an individual path goes through during its lifetime (the start and finish states are omitted to simplify the presentation). If the resources allocated to the path is sufficient for meeting its performance requirements, a path is deemed to be in *InRange* state, i.e., its performance is in the desired range. When some of its resource shares change, either it continues to meet its performance requirements or not. In the latter case, there are two possibilities depending on whether or not the path can manage to reconfigure itself to go back into the InRange state. If it can, it enters a state called *Adaptation* until reconfiguration is complete. If not, it enters the *OutRange* state, from which it can transition to the other states only when the path's allocated resource shares have been raised. We call these three states *stable* to distinguish them from transition periods between these states.

By examining the state transition in Figure 2(a), one can also observe that there are three different types of resource shares that can be associated with a network-aware path during its lifetime. The first one is the share value used in planning for a new configuration. In general, the greater the value, the better the generated plan will be. The second is the upper bound that the path is allowed to use, i.e. the allocated share. The last one is the actually used by the path at current time t.

Taking these three types of resource shares into consideration, to improve both individual path performance and throughput of the whole network, an ideal scheme for allocating resource shares among multiple paths should be able to 1) maximize the value of the share value for planning purpose to produce good quality paths; 2) minimize the difference between the allocated and the actual used share to avoid resource waste. This is the basic idea of our scheme, which employs the following strategies: 1) when a planning is needed, a path is allowed to ask for an increase in its allocated shares; 2) whenever a path enter a stable state, it is required to release unused resources. The state diagram of an individual path using this scheme is shown in Figure 2(b). The *Transition* state is for a path to send *allocation* requests to all resources involved.

In addition to producing better plans, another benefit of using the *allocation* request is that such a simple approach can effectively balance load across the network. For example, if path A uses nodes n_1 and n_2 of which n_1 is heavily loaded but n_2 has relatively light load. After sending *allocation* requests to both of them, path A will receive a larger partition at n_2. Consequently, most computation required by A will be moved to node n_2. Though a tight cooperation between n_1 and n_2 may also achieve the similar effect, such cooperation usually requires expensive information exchange about dynamic resource availability. In contrast with this, Our approach is much simpler and can scale well for large networks.

To realize this scheme, we need to specify how to (1) calculate the share that a network resource allocates in response to a path's request for resources; and (2) adjust shares for existing paths, triggered either by a need to satisfy allocation requests from other paths or by the release of some resources.

Allocation. The allocation step maximizes the likelihood that an appropriate plan for a path can in fact be constructed. For this reason, the strategy allocates either a maximum share of the resource (when the resource is undercommitted)

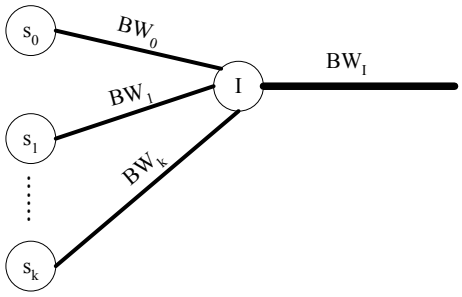


Figure 3: Hierarchical arrangement of servers and ISP nodes.

or returns the largest share it can by repartitioning the resource among available paths. In the latter case, the strategy described in detail in Appendix B attempts to minimize frequent reconfiguration and cascading adjustments.

The above allocation, referred to as the *planning share*, is made with the understanding that a path will release the unused portion of resources for use by other paths.

Adjustment. The adjustment step is required in two situations: (1) when there are insufficient resources available to satisfy requests for a new share or increases to already allocated shares, and (2) when an allocated share is released.

For the first situation, a set of existing paths need to be selected and their shares will be reduced in order to accumulate a large-enough share for the requester. The allocation step above is responsible for determining how large this share needs to be; the adjustment step decides which paths to take away resources from. Several different heuristic schemes can be employed to guide the latter process. Our scheme picks victims in decreasing order of the allocated shares, affecting paths that have a larger share of the resource. This basic scheme can be extended to restrict attention only to paths in the InRange state. The intuition here is twofold. First, such paths are more amenable to reconfiguration for staying in the desired range, as opposed to the paths in the Transition or OutRange states. Second, if resources are overcommitted, it is usually more acceptable to reject new connections than push existing paths into the OutRange state. Note that, as described in Appendix B, employing this extension may end up reducing the overall share that can be allocated to the requester.

The adjustment in the second situation is simpler: when a share of the resource is released, it is used to increase the allocation of paths in the OutRange state up to the preconfigured maximum. The intuition here is the same as in the allocation step: providing paths with the maximal opportunity of reentering the InRange state.

4.3 Resource Distribution Between Server and ISP Nodes

The third question raised at the start of this section focused on the relative computation capacity of server and ISP-level proxy sites. This question is of interest primarily for path-based approaches, where different distributions of a same amount of total computation capacity among the network nodes can result in very different performance. Given the study's objective, we need to distribute the computation resource in a way that path-based approaches can perform their best. In the rest of this section, we describe our approach for achieving this.

Our strategy is motivated by the observation that although path-based approaches can in general deploy operators on any node along a network path, usable nodes in practice are most likely a small set of strategic nodes such as ISP and gateway nodes. These kinds of nodes are also typically subject to administrative agreements between a higher-level network domain (e.g., the ISP) and a lower-level one (e.g., the server). Therefore, one can view the computation distribution problem as one of rearranging computation resources in a hierarchical network graph. Specifically, given a fixed *computation budget*, initially assumed allocated to nodes of a lower-level domain (servers in our case), the problem becomes one of moving a portion of the budget to nodes in a higher-level domain (ISP nodes in our case) to provide better overall performance by providing resource sharing across the network: overloaded servers see improved performance by taking advantage of spare resources at underloaded servers aggregated into a shared resource pool at a higher-level node in the network graph. Our strategy achieves this *without compromising* the performance of the domain from which computation resources are moved out.

The reassignment problem is illustrated in Figure 3. Each server s_i has a computation budget C_i (in operations per second), and is connected to the ISP node (I) via a link with bandwidth BW_i. The ISP node in turn is connected to the

Internet backbone with a link of bandwidth BW_I. We use the terms *server link* and *ISP link* to distinguish between the two types of links above. The problem is one of determining what portion of C_i ought to be moved from s_i to I.

We only present the main intuition behind our algorithm here, directing the reader to Appendix C for the details. Given a load distribution for the various servers and a model of the computation and bandwidth resource utilization of client connections, the algorithm compares the aggregate number of connections that can be sustained on the servers assuming the server link becomes the bottleneck with that sustainable when the ISP link becomes the bottleneck. Note that because typically $\sum_i \mathrm{BW}_i \geq \mathrm{BW}_I$ (as otherwise the bandwidth in the higher-level network domain would remain underutilized), there must exist some servers for which the ISP link becomes a bottleneck prior to the corresponding server link. For these servers, it is possible to move the unused portion of the computation budget to the ISP node. Our algorithm is used to identify these servers and calculate how many resources can be moved out without compromising their performance.

Our description above considered a two level hierarchy (i.e., between servers and ISPs). However, this strategy can be easily extended to a hierarchically organized network domain with multiple levels. The basic idea is as follows: when moving resources to a n^{th} level node, consider the $(n-1)^{\mathrm{th}}$ level node as aggregating the resources of all lower levels for purposes of the bottleneck comparison above. Note that only resources from the $(n-1)^{\mathrm{th}}$ level node would actually get transferred.

A practical note: the algorithm sketched above assumes prior knowledge of the load distribution among low level network domains. Since the load distribution varies over time, the redistribution process can be made more conservative by capping the maximal amount of resources that can be moved.

5 Performance Implications of Adaptation Approaches

To study the performance implications of end-point, proxy-based, and path-based adaptation approaches, we simulate their behaviors on a large scale network in the context of a representative workload, which models clients downloading streaming real-time media content from server sites. Our simulation study investigates how these approaches perform under different loads, for different types of servers or clients, and for different client connectivity options. We start by describing the simulation settings, and then present the detailed results of these investigations.

5.1 Simulation Settings

Application Performance Requirements In our simulation, every client downloads continuous JPEG image frames (with an average size of 4K bytes) from a server site. In order to display the received content appropriately, the throughput of a download path is required to be in the range of 10 to 16 frames per seconds,[2] and within this range higher data quality is preferred.

Possible operators that can be used with these paths include an *image-filter* and an *image-resizer*, which reduce bandwidth consumption by degrading image quality or reducing image size respectively. Both operators support ten different configurations; in each case the n^{th} configuration reducing image quality or size by a factor of $n/10$. Details about the `load` and `bwf` values for each operator configuration are omitted here for brevity.

Network Characteristics The topology of our simulated network was shown earlier in Figure 1. For the results reported here, the network is assumed to comprise ten ISPs. Each ISP is connected to the Internet backbone via an OC48 (2.488Gbps) link and includes 20 media servers, 100 public IEEE802.11b (6.0Mbps[3]) access points, and a number of client sites.

Connectivity options for clients include T3 (44.73Mbps), T1 (1.544Mbps), ADSL (1.5Mbps), Dialup (56Kbps), and IEEE802.11b connections (via the public access points). The T3, T1 and ADSL links are assumed to have sufficient bandwidth for the media application, while Dialup connections are incapable of meeting throughput requirements without the use of compression operators. For wireless connections, available bandwidth is dependent on the load of the access point and may sometimes necessitate compression operators along the path.

At each ISP, we model the arrival of clients as a Poisson process; the arrival rate of clients is a parameter that can be

[2]This means the media player must display at least 10 frames per second for a satisfactory user experience, but can not handle more than 16 frames in one second.

[3]We assume a 60% bandwidth utilization of an IEEE802.11b network.

adjusted to achieve different load levels. Once initiated, the duration of a download session is assumed exponentially distributed with an average of 1 minute.

Media servers within each ISP fall into one of two configurations. One fourth of the servers are categorized as *large sites*, with high-bandwidth connections to the ISP node (via an OC12 link operating at 622Mbps) and a computation budget uniformly distributed between 100 to 200 units.[4] The remainder three fourth of the servers are categorized as *small sites*, with lower-bandwidth connections to the ISP node (an OC3 link operating at 155Mbps) and a smaller computation budget uniformly distributed between 10 and 100 units.

Adaptation Approaches Our studies considered five different adaptation approaches: the end-point approach, the proxy approach, an approach that use servers in addition to proxies (labeled as *server+proxy*), the path-based approach, and a path-based approach without reconfiguration support (labeled as *path-reconfig*). The last approach clarifies the benefits of dynamic adaptation; communication paths in this approach can adapt to different network conditions only at path-creation time. As mentioned earlier, the first four approaches represent different constraints on where adaptation is allowed. For the *end-point* approach, only the server node and the client node of a communication path can be involved in adaptation. The *proxy* approach is allowed to use client nodes and client-side ISP nodes. The *server+proxy* approach represents an intermediate point, which, in addition to nodes used by the proxy approach, can also use server nodes for adaptation. Finally, *path* approach can use all four nodes along a communication path: the server node, the server-side ISP node, the client-side ISP node, and the client node.

To make a fair comparison between these approaches, our studies used the same total computation resource budget in each case.[5] In the end-point approach, all resources reside on server sites. For the proxy approach, all resources on server sites are aggregated on the ISP nodes they attach to. For the server+proxy approach and the path approach, a portion of the computation budget of every server site is moved to its ISP node using the strategy described in Section 4.3. The redistribution assumes that requests from clients are uniformly distributed among all server sites. Our study also examines situations where this assumption does not hold, providing insights into how performance is affected by inaccuracies in client traffic models.

Performance Metrics. Our simulations characterize two major performance metrics. The first measures the aggregate time of all paths when the throughput of a path is in the desired range. We refer to this as the `InRange` time, i.e., the time where paths stay in the *InRange* state of the state diagram shown in Figure 2. Another reasonable metric is the aggregate InRange time that is further weighted by data quality of the communication paths. Since we observed similar trends between the weighted and unweighted metrics in our study, we report only on results for the unweighted metric below.

The second performance metric is the total number of connection failures due to insufficient resources. Connection failures result from admission control, which actively rejects any incoming connection request if the initial planning cannot produce a communication path that meets the performance requirements.

In addition to aggregate data for the whole network, we also collected data for different types of servers and clients to further examine how different adaptation approaches perform from the perspective of individual servers or clients. In particular, we report on data for server sites that have the maximum or minimal computation budget, and for clients that use different connectivity options.

Reconfiguration Overheads Path reconfiguration in our study is modeled after a general protocol proposed in the context of the CANS infrastructure [3]. This protocol, which can be uniformly applied to end-point, proxy, and path-based approaches, preserves the semantic continuity of data transmission by employing the following six steps:

- Detection of changes in resource availability. As described in Section 4.2, our resource allocation strategy relies on network resources (nodes and links) being responsible for allocating partitions for individual paths. Therefore, the delay of detecting a change of resource availability is basically the time for delivering notifications. Since a notification is a small message that can be embedded in the regular data stream,[6] we model the delivery time as the total network link latency between the resource and the receiver.
- Planning. In general, the time for calculating a new path is highly dependent on the planning algorithm, but can be significantly reduced by employing a path cache of previously generated solutions. Given that attributes of most paths in our study (content type, client connectivity, resource availability) are likely to be clustered

[4] One unit is normalized as a computer with a Pentium III 1GHZ processor and 256MByte 800MHz RDRAM.

[5] The computation budget refers only to resources available for path transcoding and compression operations. Sufficient resources are assumed available on the server and proxy nodes for data retrieval from disk and forwarding through the protocol stacks.

[6] For example, the outbound data mechanism in TCP can be used for delivering such notifications.

in a small range, we expect a good hit rate from such a cache. Consequently, we assume that planning incurs negligible overhead, modeling the situation where new plans are almost always obtained directly from the cache.

- Distribution of the new plan. New plan partitions need to be distributed to every node, which needs to be reconfigured, along the communication path. This is done by sending, in parallel to all these nodes, a data packet containing the plan partition of the receiving node. The packet itself has a size that is plan-dependent, and incurs latency dictated by the available bandwidth on the network resources being utilized.
- Flushing data in transmission. The protocol ensures semantic continuity of data transmission by flushing any incomplete data segments in transmission or internal state built up in operators. Additional details about this process can be found in [3]. We model the overhead of this step in the simulation as the time required for transmitting the required segments.
- Deployment of new operators. Because operators are reusable and contain only soft state, the time for replacing old components with new operators on a node is usually a constant. In our study we use a value of 100 milliseconds, which is consistent with that observed in experimentation with the CANS infrastructure.
- Resumption of data transmission. The final step resumes data transmission through the new path. In the simulation, this step is assumed to incur negligible overhead.

In the rest of this section, we first report on the performance achieved by different adaptation approaches with client traffic uniformly distributed among the various server sites for a particular client connectivity profile. We then separately examine how performance is affected by non-uniform traffic distribution (where "hotspot" servers receive a larger share of the connection requests), and when the client connectivity profile is changed (with different fractions of clients using high-bandwidth and low-bandwidth links). In each case, we simulate the network for 4 minutes, recording data only for sessions that are started within the last 2 minutes, i.e. after the network reaches a stable state (recall the average length of a session is 1 minute). The measurement ends at the 4 minute mark.

5.2 Performance under Uniform Load Distribution

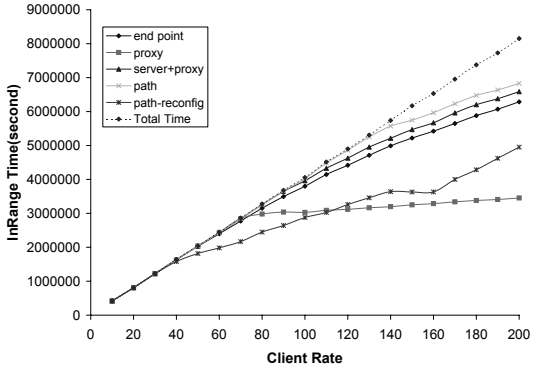
This configuration uniformly distributes client requests among all server sites, varying client arrival rates at each ISP from 10 to 200 clients per second. These rates correspond to 6000 to 120,000 active paths simultaneously existing in the network. The client connectivity profile is fixed as follows: 25% use links with sufficient bandwidth (T1/T3/ADSL), 25% use Dialup, and the remaining 50% use wireless connections. We examine the impact of changes from this profile later in Section 5.4.

The performance achieved by different adaptation approaches for this configuration is shown in Figure 4. Plots (a) and (b) respectively show values for the InRange time and number of rejected connections, aggregated over all servers. Plots (c),(d) and (e),(f) show the corresponding plots for the server with the maximum computation budget (199 units), and for that with the minimum budget (10 units). Figures 4(g) and (h) show the InRange time (normalized with respect to the total session time) seen by clients who use T1/T3/ADSL links and those that use weaker dialup/wireless connections.

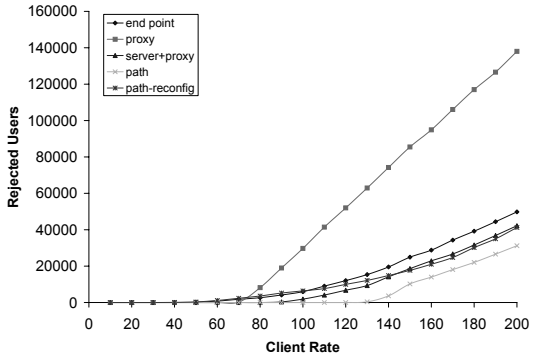
5.2.1 Analysis of Aggregate Performance

From Figures 4(a) and (b), it can be observed that all four adaptation approaches that include reconfiguration support perform very well when the network is lightly loaded. However, after the load increases to some level (client rate=80 in Figure 4(a)), the performance of the proxy approach is the first to reach saturation. This is explainable by the following: since adaptation can only occur on the node before the last hop, all paths end up consuming considerable bandwidth in the network core, consequently saturating this portion of the network much faster than other approaches. Once the network gets saturated, further increases in InRange time are still possible, albeit at a much smaller rate, because of local loops (a client downloads contents from a server that is attached to the same ISP).

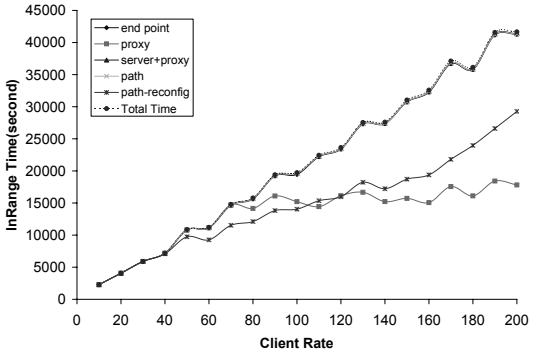
Compared with the proxy approach, the end-point approach performs better (with higher InRange Time and fewer connection failures), especially after the "saturation" point of the proxy approach. This is expected because the end-point approach uses server sites to do image filtering or resizing, and does not waste bandwidth on the network links. However, it can also be observed from the Figure 4(b) that the end-point approach starts to reject connections early, even when the network is lightly loaded. These rejections mainly come from clients that use weaker links such as Dialup to access small sites with limited computation capacity.



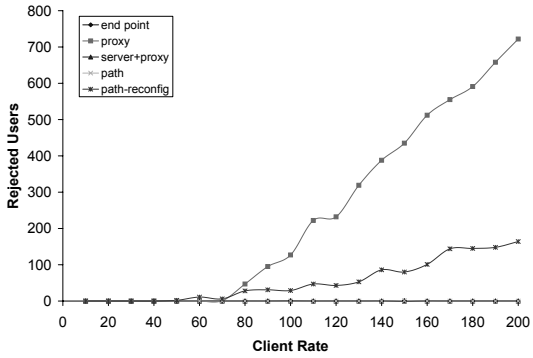
(a) Aggregate InRange Time



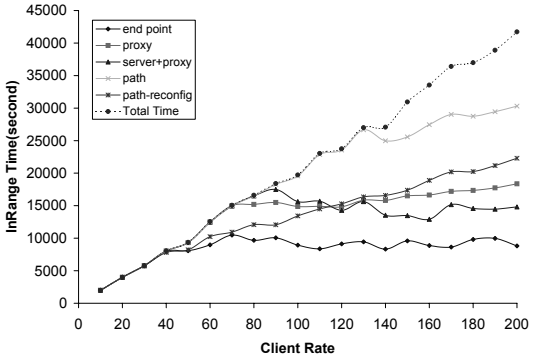
(b) Aggregate Connection Failures



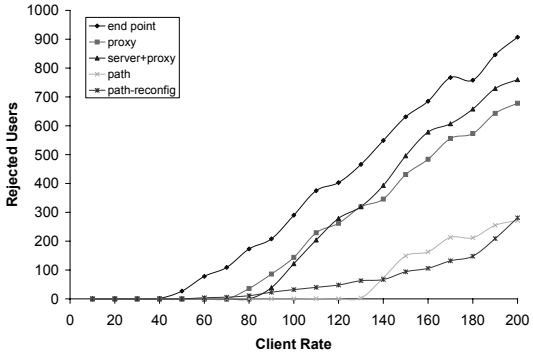
(c) InRange time for Server with Max. Budget



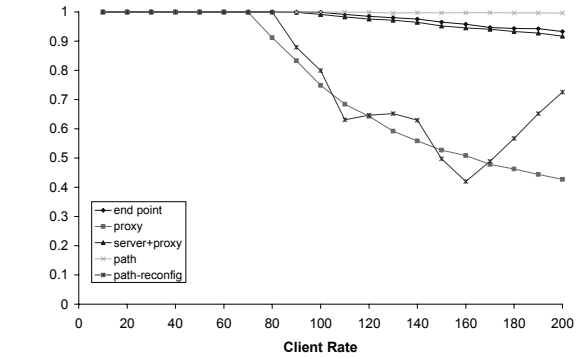
(d) Conn. Failures for Server with Max. Budget



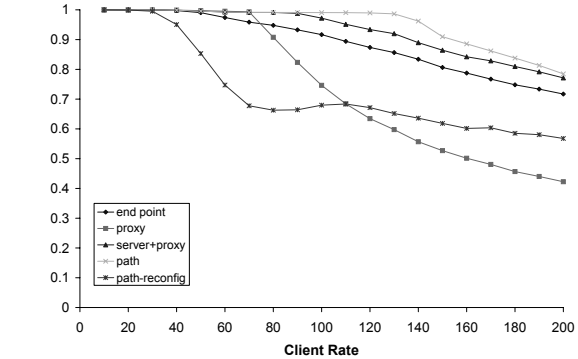
(e) InRange time for Server with Min. Budget



(f) Conn. Failures for Server with Min. Budget



(g) Normalized InRange Time for T3/T1/ADSL Clients



(h) Normalized InRange Time for Dialup/Wireless Clients

Figure 4: Performance under Uniform Load Distribution.

Figures 4(a) and (b) also show that the path-based approach provides the best performance at all load levels. The InRange time of the path-based approach is up to 12% and 97% higher than that of the end-point approach and the proxy approach respectively. The number of connection failures of the path-based approach is also much lower. For example, for a client rate of 200 connections/second, the end-point approach rejects 59% more connections and the proxy approach rejects about 343% more connections than the path approach. The reason for this behavior is because the path-based approach combines the advantages of both proxy and end-point approaches. On one hand, similar to the end-point approach, the path-based approach can utilize upstream nodes along a communication path to ensure that network bandwidth is not wasted; and on the other, similar to the proxy approach, the path-based approach can set up shared resource pools across the network, permitting overloaded servers to benefit from spare computation resources elsewhere.

The performance of the server+proxy approach falls between the path-based approach and the end-point approach, which verifies that allowing adaptation to happen on even one more node in the middle of the communication path can improve overall performance.

5.2.2 Performance of Different Server Sites

Comparing between Figures 4(c)–(f) allows us to draw conclusions about how the different adaptation approaches perform from the perspective of connections targeting servers with higher or lower computation budgets. The results indicate that the end-point approach shows a distinct bias, performing much better with the largest server than with the smallest one. The proxy approach performs uniformly with both servers, primarily because all computation resources are aggregated at proxy sites. The path-based approach performs as well as the end-point approach for the largest server, and performs the best for the smallest server. This can again be explained by the flexibility brought by resource sharing and being able to use upstream nodes to do adaptation.

Another point deserving mention is the performance decrease of the server+proxy approach in Figure 4(e) for client arrival rates higher than 90 connections/second. This can be explained as follows: after load increases to the point where the smallest server runs out of computation resources, other server nodes continue to support filtering or resizing operators because they have additional computation capacity. Since compressed connections (with operators) consume less bandwidth than uncompressed ones, accepting more compressed connections for these servers can in turn decrease the number of uncompressed connections to the smallest server because the size of resource shares in the core network links shrinks as more compressed connections join in. Consequently, for the smallest server, the InRange time drops and the number of connection failures increases as load increases. Note that the path-based approach avoids this situation by exploiting resource pooling at server-side proxies.

5.2.3 Performance of Different Clients

Figures 4(g) and (h) show the performance of the adaptation approaches from the perspective of different client classes, i.e., clients connected to the network with sufficient bandwidth versus those that use weaker connections. We can observe that while the proxy approach exhibits more or less uniform behavior, the end-point approach demonstrates considerable preference for clients with better connectivities over others. The path approach, in addition to providing the best performance, uniformly supports different classes of clients until one runs out of computation resources beyond a certain load level. At this point, all approaches end up rejecting more clients with weak connectivity because they require image filtering or resizing operations along the paths.

5.2.4 Performance Impact of Dynamic Reconfiguration

The plots in Figure 4 also show that there is a considerable performance penalty incurred for disallowing reconfiguration after the path has been created. This validates the need for a *reactive* mechanism to cope with dynamic changes. In general, different types of paths may have different requirements on network resources (e.g. some of them may require more bandwidth while others may need more computation). As load changes, it is necessary to adjust allocated shares of existing paths in order to accept more connections.[7] Without reconfiguration, adjustments for one path may end up pushing other paths out of the required range, and thereby negatively impact overall performance.

[7] One can argue that using reservations may eliminate the need for dynamic adjustments, but such approaches usually have poor throughput as load dynamically changes

Another detail that should be mentioned about the path-reconfig approach is the ramp-up at the end of Figure 4(g). This can be explained as follows: as the number of client connections increase, the number of partitions of network resources grows while decreasing the size of each partition. Eventually, it becomes difficult for clients who use weak connections to successfully connect to servers because the partition of computation resources is too small to perform the required image filtering or resizing operations. As a result, a large number of such connections end up getting rejected. On the other hand, connection requests from clients with higher bandwidth links continue getting accepted. Moreover, because more compressed paths are rejected, the likelihood that an uncompressed path will get pushed out of the required range decreases. This results in increased normalized InRange time for clients who use T3/T1/ADSL links.

5.3 Performance under Non-Uniform Load Distribution

This configuration examines how different adaptation approaches perform when connection requests from clients are directly non-uniformly towards servers. Similar to load patterns observed on the Internet, we assume a "hot-spot" model, where a small number of servers (the hot-spots) receive most of the requests from clients. Specifically, 20% of the servers receive 80% of the total requests. We further ensure that the average load of large sites (i.e., sites with an OC12 link with computation budget uniformly distributed in the range [100,200)) is about 4 times the average load of small sites (i.e., sites with an OC3 link with computation budget uniformly distributed in the range of [10,100)).

Figures 5(a)–(h) show the performance achieved by the different approaches. The organization of the plots is similar to that seen earlier in Figure 4. There are several observations that one can make here. First, focusing on aggregate performance, we see that the overall ranking of performance among these adaptation approaches remains the same as in the uniform distribution case. However, the total InRange time is noticeably lower than the values we saw in Section 5.2. This is expected because the overloaded hot-spot servers cause increased connection failures. Second, the relative performance of the path-reconfig approach is worse than seen earlier. This verifies our intuition that such an approach performs poorly when some portions of the network get overloaded; due to the absence of reconfiguration, existing paths cannot be adjusted so they take advantage of surplus resources in lightly loaded regions of the network.

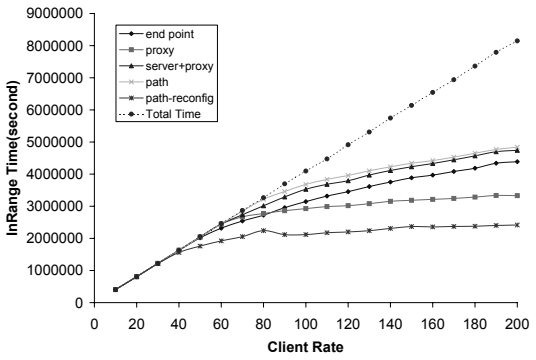
Looking at the performance from the perspective of servers with the maximum and minimum computation budgets (Figure 5(c)–(f)), it can be observed that the path-based approach outperforms all other approaches. This again verifies the benefit of resource sharing in the network: overloaded sites can always take advantage of spare computation resources elsewhere. This is true even for sites that have plenty of computation resources, because there will be a load level that causes these sites to become overloaded. The end-point approach performs poorly on sites with smaller computation budgets. The proxy approach exhibits the same behavior, independent of computation budget, as in the uniform distribution case. However, as before, the problem of bandwidth waste results in the network core becoming an early bottleneck as load increases.

Looking at performance seen by clients with different connectivity options, the overall trends mirror those seen for uniform traffic. Figure 5(g) is a little different from the corresponding plot in Figure 4 in that the end-point approach gets the highest normalized in-range time for clients using T3/T1/ADSL connections. This value comes at the cost of more connection failures for clients with weak connections (recall that 75% of all clients use dialup/wireless connections). The aggregated InRange time of the path-based approach is still the best among the five approaches.

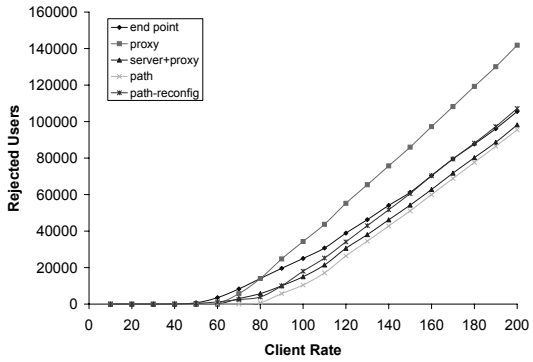
A more interesting point with this set of results is that they correspond to the same resource distribution between server and ISP nodes as in Section 5.2, namely one that *assumes a uniform load distribution*. This is relevant because load distributions at run-time are likely to be different from that considered when deciding about how to provision resources in the network. Our results show that the path-based approach still performs very well even with an inaccurate knowledge of load distribution. This robustness mainly comes from the shared resource pools across the whole network that act like "buffers", absorbing any negative impact because of the unexpected load.

5.4 Performance under Different Client Connectivity Profiles

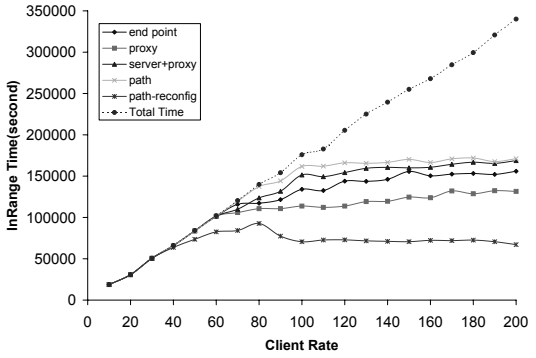
In this configuration, we examined how the different adaptation approaches perform when different fractions of clients use different connectivity options. The simulations run with the same settings as in Section 5.3 with only two differences: the client arrival rate was fixed at 100 users per second, and we varied the percentage of clients that use weak connections (dialup or wireless) from 0 to 100 percent (the ratio between numbers of clients that use dialup and



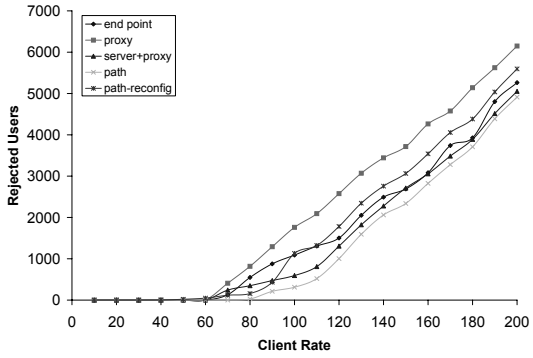
(a) Aggregate InRange Time



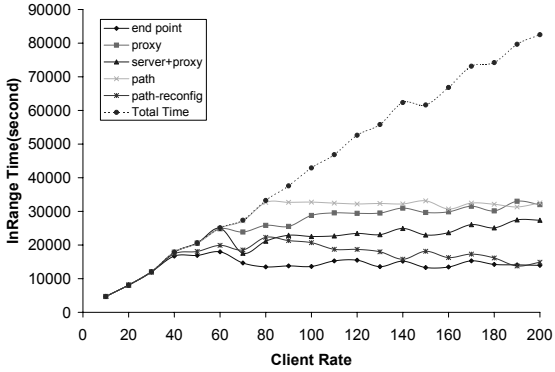
(b) Aggregate Connection Failures



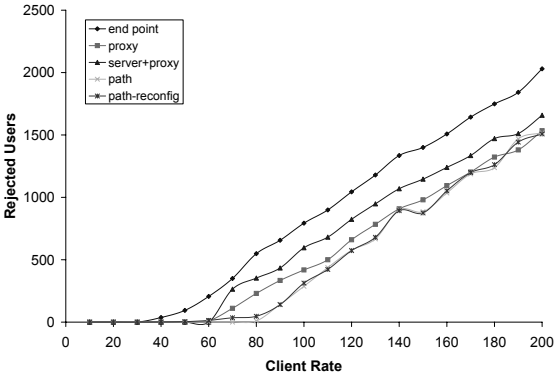
(c) InRange time for Server with Max. Budget



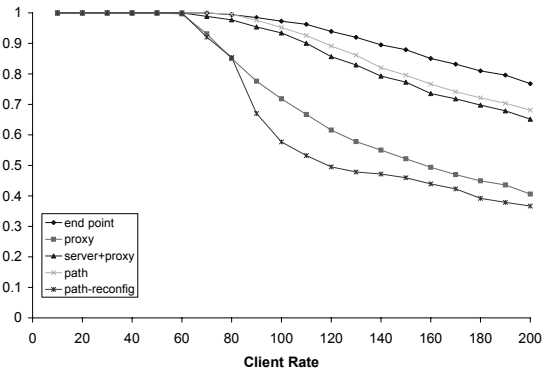
(d) Conn. Failures for Server with Max. Budget



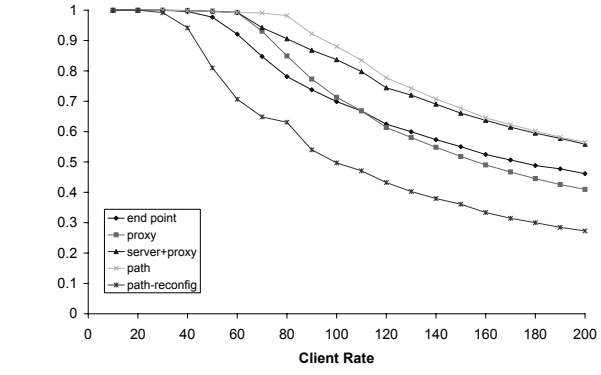
(e) InRange time for Server with Min. Budget



(f) Conn. Failures for Server with Min. Budget

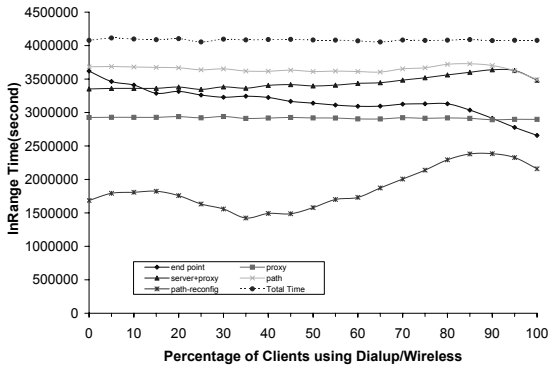


(g) Normalized InRange Time for T3/T1/ADSL Clients

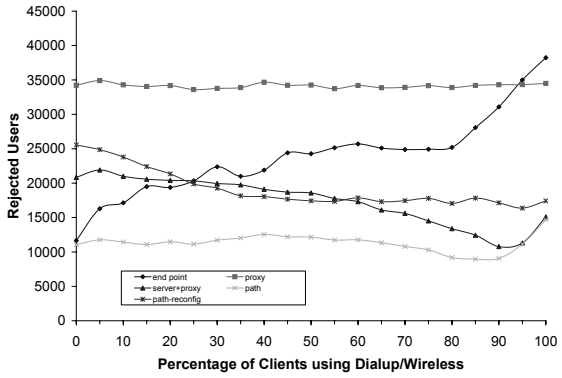


(h) Normalized InRange Time for Dialup/Wireless Clients

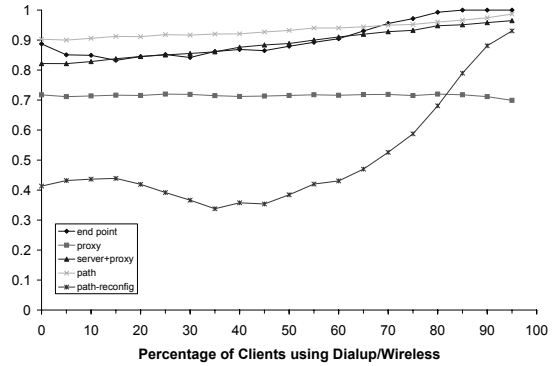
Figure 5: Performance under Non-Uniform Load Distribution.



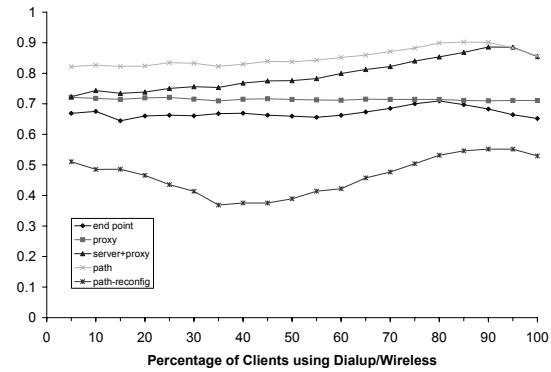
(a) Aggregate InRange Time



(b) Aggregate Connection Failures



(c) Normalized InRange Time for T3/T1/ADSL Clients



(d) Normalized InRange Time for Dialup/Wireless Clients

Figure 6: Performance under Different Client Connectivity Profiles.

wireless connections was maintained at 1:2).

Figure 6(a)–(d) shows the performance results. One can observe that among the four approaches with reconfiguration support, the end-point approach is the only one that exhibits decreasing performance as more clients use weak connections while the other three approaches achieve relatively stable performance across different configurations. Because the end-point approach does not support resource sharing, smaller sites or overloaded sites end up rejecting many connection requests once they run out of computation resources.

It can also be seen that the path-reconfig approach performs better when the client connectivity profile is more uniform. This can be explained as follows: as more paths exhibit similar behavior (i.e., have similar resource requirements), there is lower likelihood that an existing path will get pushed out of its required performance range by the arrival of a new connection. Stated differently, the more heterogeneous the environment, the larger the need for dynamic reconfiguration.

Some clarification is needed for the increasing InRange time achieved in Figure 6(a) by the server+proxy approach as more clients use weak connections. While this may appear counter-intuitive, the following explains this behavior. Consider what happens when clients use connections that have sufficient bandwidth. As load increases, initially modest compression (filtering/resizing) will be introduced into the paths and executed on the server sites. As the number of connections further increases, the size of partition on the server sites will eventually become too small to do the required compression. Consequently, after this point, the network core starts become a bottleneck and once it does, new connections end up getting rejected. Note however that when this happens, the proxy sites close to clients remain underutilized because they are ineffective for reducing bandwidth requirements in the network core.

On the other hand, the situation is different when most of the clients are using weak connections. Due to the limited bandwidth of weak connections, strong compression will be required at the server sites from the beginning.

The strong compression results in considerable saving in bandwidth in the network core. Therefore, as load increases, some of the new connections can take advantage of the saved bandwidth in the network core and do compression at the client side proxy sites. As a result, the utilization of the proxy sites is high and more connections are accepted.

The above behavior also provides further evidence for the benefits from using additional nodes in the data path to perform adaptation operations.

5.5 Summary of Simulation Results

The main results from our study are summarized below:

1. Support for dynamic reconfiguration is important for the performance of both individual paths and the whole network.
2. The end-point approach usually works well with server sites that have a large amount of computation resources and for clients that connect to the network with relatively high bandwidth links. However, servers that have limited computation capacity or clients that use weak connections may suffer from poor performance using such an approach.
3. The proxy approach usually does not exhibit bias towards different types of servers or clients. The shared resource pool at proxy sites can bring better performance for small server sites or clients that have weak connectivity. However, constraining the adaptation to only occur before the last hop can cause considerable resource wastage in the network, in turn leading to early saturation as load increases.
4. The path-based approach has all the benefits of both end-point and proxy approaches. Adaptation can be conducted on upstream nodes without being limited to the node before the last hop. More importantly, the approach sets up shared resource pools across the whole network, providing the most flexibility for overloaded servers to benefit from spare computation resources elsewhere. In summary, with effective resource management strategies, this approach provides the best and the most robust performance under different network configurations.

6 Conclusions

In this paper, we have investigated the performance implication of different adaptation approaches that have different constraints on adaptation location: the end-point approach, the proxy approach and the path-based approach. By conducting a series of simulation-based experiments using different network configurations we have shown that there are well-defined network environments under which each of the adaptation approaches delivers its best performance. More importantly, each approach, with one exception – the path-based approach, also exhibits performance shortcomings in certain environments. The path-based approach ends up delivering the best and most robust performance under different network configurations, and for different types of servers and clients because of its ability to carry out adaptation along the entire path and pool resources across the while network. Thus, despite their somewhat increased complexity, path-based approaches appear the most promising for delivering high performance in environments comprising diverse network resources and where additionally the characteristics of these resources are subject to constant change.

References

[1] E. Amir, S. McCanne, and R. Katz. An Active Service Framework and its Application to Real-time Multimedia Transcoding. In *Proc. of the SIGCOMM'98*, August 1998.

[2] A. Fox, S. Gribble, Y. Chawathe, and E. A. Brewer. Adapting to Network and Client Variation Using Infrastructural Proxies: Lessons and Prespectives. *IEEE Personal Communication*, August 1998.

[3] X. Fu, W. Shi, A. Akkerman, and V. Karamcheti. CANS:Composable, Adaptive Network Services Infrastructure. In *Proc. of the 3rd USENIX Symposium on Internet Technologies and Systems (USITS)*, March 2001.

[4] S. D. Gribble and et al. The Ninja Architecture for Robust Internet-Scale Systems and Services. *Special Issue of IEEE Computer Networks on Pervasive Computing*, 2000.

[5] J. Haartsen. BLUETOOTH– The universal radio interface for ad hoc, wireless connectivitity. *Ericsson Review*, 1998.

[6] A. D. Joseph, J. A. Tauber, and M. F. Kasshoek. Mobile Computing with the Rover Toolkit. *IEEE Transaction on Computers:Special Issue on Mobile Computing*, 46(3), March 1997.

[7] R. Mohan, J. R. Simth, and C.S. Li. Adapting Multimedia Internet Content for Universal Access. *IEEE Transactions on Multimedia*, 1(1):104–114, March 1999.

[8] A. Nakao, L. Peterson, and A. Bavier. Constructing End-to-End Paths for Playing Media Objects. In *Proc. of the OpenArch'2001*, March 2001.

[9] Brian D. Noble. *Mobile Data Access*. PhD thesis, School of Computer Science, Carnegie Mellon University, 1998.

[10] A. Vahdat, M. Dahlin, T. Anderson, and A. Aggarwal. Active names: Flexible location and transport of wide-area resources. In *Proceedings of the USENIX Symposium on Internet Technologies and Systems (USITS)*, 1999.

[11] U. Varshney and R. Vetter. Emerging Mobile and Wireless Networks. *Communications of the ACM*, pages 73–81, June 2000.

[12] M. Yavis, A. Wang, A. Rudenko, P. Reiher, and G. J. Popek. Conductor: Distributed Adaptation for complex Networks. In *Proc. of the Seventh Workshop on Hot Topics in Operating Systems*, March 1999.

A Path Creation Algorithm

The algorithm corresponding to the base dynamic programming strategy described in Section 4.1 is shown in Figure 7. The global metric being optimized is throughput, the number of data units produced per unit time at the destination.

The algorithm fills up a table of partial optimal solutions, $s[t_s, t, \vec{A}, i]$, in the order $i = 0, 1, 2, \ldots$. Each solution corresponds to the data path that yields the best performance for transforming the source type t_s to type t using i operators or fewer requiring no more resources that $\vec{A}$. As mentioned in Section 4.1, only resource vectors of the form $(1, ..., 1, r_j \in [0, 1], 0, ..., 0)$ need to be considered. This set of resource vectors is designated RA.

Algorithm *Plan*
Input: t_s,t_d, G_t, R
Output: The data path that yields maximal throughput from type t_s to t_d on route R

1. ($*$ Step 1: Initialization for partial plans with zero components $*$)
2. **for** all $t, \vec{A} \in \mathrm{RA}$
3. **do** calculate $s[t_s, t, \vec{A}, 0]$
4. ($*$ Step 2: Incrementally building partial solutions $*$)
5. **for** $i \leftarrow 1$ **to** $p \times n$
6. **do for** all $t \in V(G_t), \vec{A} \in \mathrm{RA}$
7. **do** $s[t_s, t, \vec{A}, i] \leftarrow s[t_s, t, \vec{A}, i-1]$
8. **for** all $d = (t', t) \in E(G_t)$
9. **do for** all n_j that $\vec{A}[n_j] > 0$
10. **do** $M(d) \leftarrow n_j$
11. $\vec{A}' \leftarrow (\vec{A}[0], \ldots, \vec{A}[n_j - 1], \vec{A}[n_j] - \mathrm{load}(d), 0, \ldots)$
12. **if** $\mathrm{throughput}(\mathrm{append}(s[t_s, t', \vec{A}', i-1], d, \vec{A})) > s[t_s, t, \vec{A}, i]$
13. **then** $s[t_s, t, \vec{A}, i] \leftarrow \mathrm{throughput}(\mathrm{append}(s[t_s, t', \vec{A}', i-1], d, \vec{A}))$
14. **return** $s[t_s, t_d, \vec{A} = [1, 1, ..., 1], p \times n]$

Figure 7: Path Creation Algorithm

Line 3 of the algorithm handles the base case: only the case $t = t_s$ achieves non-zero throughput. Lines 8–13 represent the induction step, examining different drivers to extend the current partial solution for each specific intermediate type t and resource vector $\vec{A}$. Lines 12 and 13 ensure that the driver achieving the maximum throughput defines the next-level partial solution.

The throughput for a particular mapping can be computed given the node throughput and link bandwidth properties. Node n_i's throughput itself is decided by the incoming throughput, its computation capacity $\mathrm{comp}(n_i)$, and the load and bwf properties of components mapped to the node. Link bandwidth has the effect of saturating throughput, in case the bandwidth requirements exceed what can be sent over the link. Only lines 12 and 13 of the algorithm need change if a different metric is being optimized (such as latency).

The algorithm terminates at Step $p \times n$ with the final solution in $s[t_s, t_d, (1, ..., 1), p \times n]$. This follows from the observation that there is no performance benefit from mapping multiple copies of the same component to a node. The complexity of this algorithm is $O(n^2 \times m \times p^3) = O(n^3 \times p^3)$[8] as opposed to $O(p^n)$ for an exhaustive enumeration strategy. As stated earlier, in most scenarios, p is expected to be a small constant, with overall complexity determined by the number of components.

B Calculation of Resource Share for Allocation Requests

In Section 4.2, we described how network resources are shared amongst multiple paths that use them. Here, we present the details of the algorithm used to determine the share that is granted upon receipt of an allocation request. As mentioned earlier, the goals of the algorithm are to provide the largest allocation (up to a maximum value, MAX) to ensure success during planning, while at the same time avoiding frequent reconfiguration and cascading adjustment. The algorithm listed in Figure 8 reflects these ideas.

Algorithm *Allocation*
Input: Path
Output: Allocated Share for the path
1. **if** $\mathrm{available} > \mathrm{MAX}$
2. **then return** MAX
3. ($* p$: number of paths, n: increase in p within the last time unit $*$)
4. $r \leftarrow \max(1, n)$
5. ($* p_r = \max(\lceil p/r \rceil \times r, p + c *$)
6. **if** (Path is a New Path)
7. **then return** $1/p_r$
8. **else return** $\max(\mathrm{current\_share}, \min(\mathrm{available}, 1/p_r))$

Figure 8: Calculation of the Allocated Share for an Individual Path

When the resource is underutilized, allocation requests result in a predefined MAX amount of resources being allocated. Information about this amount can either be provided by the path or specified by the resource. Note that since paths return unused resources, allocating a large share for planning purposes does not negatively impact resource availability for future paths.

The case where the resource is oversubscribed (i.e., fewer than MAX resources are available) is more interesting. Intuitively, the algorithm implements a fair policy: the resource is equally partitioned among all active paths. However, this base policy needs to be refined to meet our original goals, namely to avoid frequent reconfiguration and cascading adjustment.

A situation where frequent reconfiguration happens with the base policy is when new paths are continually entering the system, making allocation requests. If paths were allocated a share of $1/p$ (where p is the number of paths) the arrival of each new path would force an adjustment of the shares granted to all existing paths, resulting in an undesirable user experience. Consequently, the algorithm "damps" the effect of path arrivals by instead allocating a smaller share $1/p_r$, where the quantity p_r is computed in terms of two parameters n and c as shown in Figure 8. As computed in lines 5–7, p_r takes on a new value only once for each time period (using n, a prediction for the expected increase of active connections over the period). This has the benefit that each existing path would need to be adjusted at most once over a time period.

Line 7 also shows how the parameter c is used to bound the minimum value of p_r. By observing that each adjustment of a path returns a share equal to $(1/p - 1/p_r)$, it follows that the fraction of paths that will need to be adjusted to grant an allocation request is $1/(p_r - p)$. The value of this fraction is at most $1/c$, thereby limiting the amount of work that will need to be done in the worst case. c would typically be a different predefined constant for each individual resource. In our experiments, we choose c to be 5% of the maximum number of paths that can be supported on the resource.

The other refinement over the base scheme is shown in lines 8–10 of the algorithm, where different shares are returned depending on whether the allocation request is made by an existing path or a new one. For existing paths,

[8] It is safe to assume that m, the number of types is smaller than n, the number of components.

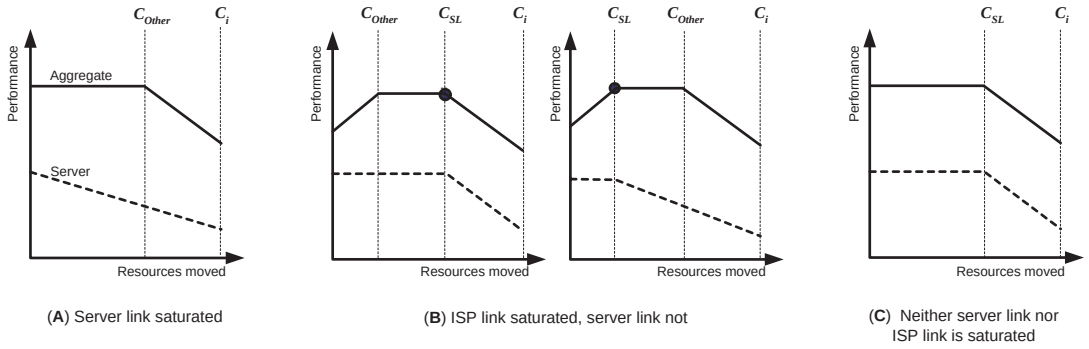


Figure 9: Performance impact of incrementally transferring computation resources from a single server node to the ISP node for a fixed load level. The three cases correspond to different saturation situations for the server and ISP links. C_{Other} denotes the maximum resource level that can be utilized for improving the performance of other servers. C_{SL} denotes the resource level at which the server link gets saturated.

the algorithm ensures that any increases in share allocation are constrained from above by the amount of available resources (i.e., those resources that can be granted without adjusting share allocations for other paths). To understand this policy, consider what would happen in its absence for a path A, which shares the resource r_1 with path B. If A requests an increase in its allocated share, the share of path B may need to be reduced. To maintain required performance, path B may in turn need to issue allocation requests to increase its shares on other resources along the path. These requests from B may affect path C in a similar fashion if B and C happen to use a resource r_2. The same thing can happen for path D if C and D share another resource r_3, and so on. The situation would be even worse if D is actually A and r_3 is actually r_1, in which A initially tries to increase its shares on r_1 but end up with a decreased share on that resource. In short, such propagation may cause the whole network to oscillate with overwhelming allocation requests and adjustments. The constraint in line 10 avoids this problem.

The algorithm in Figure 8 treats all paths uniformly for resource allocation purposes. Note that it is straightforward to extend this scheme to handle cases where some paths have higher priority than others by associating weights with paths.

C Distribution of Computation Budget for Path Based Approaches

As described in Section 4.3, our scheme moves unused resources from lower-level network domains to higher-level ones, where these resources can be used by other servers and thereby improve overall system performance. We limit our discussion below to the case of two domains involving multiple server nodes in the lower level, and a single ISP node at the higher level (see Figure 3). In this context, the question that our strategy answers is what fraction of the computational resources from which servers can be transferred to the ISP node *without compromising* the performance of the contributing servers, namely leaving unchanged the number of connections that they can serve.

To describe the strategy, we need to introduce a model for client connection requests. We assume that client communication paths require a throughput of TH data units per second, and that these paths are of two kinds: *uncompressed* and *compressed*. The latter involve transcoding and/or compression operators to reduce bandwidth requirement of the path. Each compressed path requires an average computation of c operations per data unit,[9] and reduces bandwidth requirements by the fraction D. We further assume that the fraction $p_i (0 \leq p_i \leq 1)$ of all requests via ISP I are for accessing contents on server s_i, which has a computation budget of C_i. As in Figure 3, we refer to the bandwidth on the link connecting server s_i to I (the server link) as BW_i and that on the link connecting I to the Internet (the ISP link) as BW_I. Our strategy computes C'_i, the computation resource left at each server s_i. Note that $C'_i \leq C_i$.

Assuming the above client traffic distribution, the strategy identifies servers for whom the corresponding server

[9]Here we only consider the computation capacity required for manipulating data; the overhead of reading content from disk and passing it through a protocol stack are not counted since these overheads are always present.

$n_I = \mathrm{BW}_I/\mathrm{TH} + (1-D) \times \sum_i \frac{C_i}{\mathrm{TH} \times c}$ $\left[\begin{array}{c} n_I = n_c(\mathrm{compressed}) + n_{\mathrm{uc}}(\mathrm{uncompressed}) \\ n_c = \sum_i \frac{C_i}{\mathrm{TH} \times c} \\ n_{\mathrm{uc}} \times \mathrm{TH} + n_c \times D \times TH = \mathrm{BW}_I \end{array} \right]$	Maximum number of connections sustainable over the ISP link
$n_{\mathrm{sl},i} = \mathrm{BW}_i/\mathrm{TH} + (1-D) \times \frac{C'_i}{\mathrm{TH} \times c}$	Maximum number of connections can be sustained at the server link of s_i after moving some of its computation resources to the ISP node.

Table 1: Expressions for the number of connections that can be sustained on ISP and server links.

link has unused capacity for load levels where the ISP link is operating at capacity. The rationale for this choice can be seen by examining Figure 9, which depicts, for a given load level, the impact on overall performance as resources are incrementally transferred from a particular server, s_i, to the ISP node. Depending on whether the ISP link is saturated or not, and whether a particular server link is saturated or not, one can distinguish three classes of server behavior: (A) when the server link is saturated; (B) when the server link is unsaturated while the ISP link is saturated; and (C) when neither the server link or the ISP link are saturated. For each class, Figure 9 shows the impact on aggregate system performance (solid line) and individual server performance (dashed line) as resources are incrementally moved out of the server.

When the server link gets saturated (case (A)), any movement of computation resources out of s_i will decrease the number of connections sustainable at the server. This decrease is offset at the aggregate system level until C_{Other} resources have been transferred, by other servers benefiting from the pooled resources.

When the ISP link gets saturated before the server link (case (B)), there are two situations. Both situations start off by seeing an increase in aggregate performance because of additional compressed connections being served on other servers. Meanwhile, moving computation resources out of s_i increases the bandwidth consumption at its server link but does not affect its performance. This situation continues until we reach a point where either there is no further benefit from additional ISP resources (the left figure), or the server link gets saturated (the right figure). In the first situation, aggregate performance levels off until the C_{SL} level is reached at which point both server and aggregate performance start decreasing. In the situation where the server link gets saturated first, server performance starts decreasing immediately but its impact on aggregate performance is offset as in case (A) above until the C_{Other} level is reached. The points marked by black circles in the case (B) figures represent the maximal amount of computation resources that can be moved out of s_i without degrading its performance.

In Figure 9(C), neither the ISP link nor the server link is saturated, so moving computation resources from s_i does not increase the aggregate performance unlike in case (B). This is understandable because at this time the number of connections that can be sustained at a server is unaffected by the amount of computation resources at the ISP node. Only after the server link gets saturated does both the aggregate and server performance decrease. In principle it is possible to move resources out of servers that fall into category (C) above without degrading their performance; however, this requires knowledge not only of the traffic distribution but the actual load seen by each server. In most cases, the latter information is not readily available. Changes in load levels can convert a case (C) server into either case (A) or (B) depending on whether the server link or the ISP link gets saturated first.

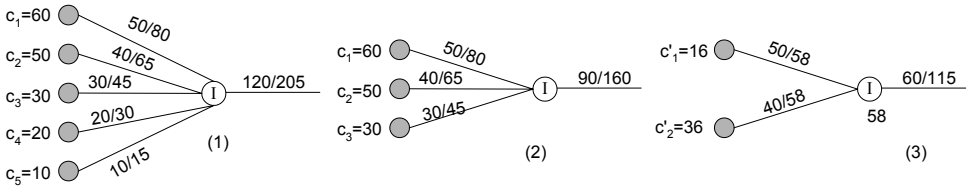
In light of the above analysis, our strategy restricts itself to identifying servers that would fall into category (B) above. For such servers, it is safe to move resources up to the point marked by the black circles in Figure 9(B) irrespective of the encountered load levels.

Servers whose server links remain unsaturated when the ISP link is saturated can be identified by comparing $p_i \times n_I$, the maximum number of server connections that can be sustained assuming that the ISP link becomes the bottleneck (p_i is the fraction of connections directed towards s_i) with $n_{\mathrm{sl},i}$, the maximum number of server connections that can be sustained assuming the server link becomes the bottleneck. Table 1 shows how these parameters are computed by considering the number of compressed and uncompressed connections that can be supported by a given amount of computation and bandwidth resources. The $n_{\mathrm{sl},i}$ expression assumes that C'_i resources are left behind at the server. Servers in case (B) must have $n_{\mathrm{sl},i} \geq p_i \times n_I$, i.e. satisfy the following equation:

$$\begin{cases} 0 \leq C'_i \leq C_i \\ \mathrm{BW_i}/\mathrm{TH} + (1-D) \times \frac{C'_i}{\mathrm{TH} \times c} \geq p_i \times n_I \end{cases} \tag{1}$$

Algorithm *Distribute*
Input: Server Set S, BW_I
Output: Distribution of computation between servers and the ISP node
1. $S' \leftarrow S$
2. $\mathrm{BW}'_I \leftarrow \mathrm{BW}_I$
3. **for** all $s_i \in S'$
4. **do if** for s_i equation (1) has a solution
5. **then** set C'_i
6. **else** $C'_i \leftarrow C_i$
7. $\mathrm{BW}'_I \leftarrow \mathrm{BW}'_I - BW_i$
8. $S' \leftarrow S' - s_i$
9. **if** $S' \neq S$
10. **then** Adjust load distribution for all $s \in S'$
11. **Call** *Distribute*(S',BW'_I);
12.

Figure 10: Distribution of Computation Resources between ISP and Server Nodes



TH=1 D=0.5 X/Y: bandwidth = X, Number of connections can be sustained=Y

Figure 11: Example showing recursive calculation of the computation budget transferred to the ISP node.

It is easy to prove, by contradiction, that there must be at least one category (B) server. Let us assume that no server has a valid solution for Equation 1. This implies that (summing up over all servers)

$$\sum_i \frac{\mathrm{BW_i}}{\mathrm{TH}} + (1-D) \times \sum_i \frac{C_i}{\mathrm{TH} \times c} < \sum p_i \times n_I$$

The right hand side of the above inequality is just n_I, which in turn can be substituted by the corresponding expression from Table 1. Thus, our assumption leads us to the inequality

$$\sum_i \frac{\mathrm{BW_i}}{\mathrm{TH}} + (1-D) \times \sum_i \frac{C_i}{\mathrm{TH} \times c} < \frac{\mathrm{BW}_I}{\mathrm{TH}} + (1-D) \times \sum_i \frac{C_i}{\mathrm{TH} \times c}$$

This requires $\sum_i BW_i < BW_I$, which is in contradiction with our previous assumption of $\sum_i BW_i \geq BW_I$. Therefore, there must be at least one category (B) server.

The recursive algorithm employed by our strategy is shown in Figure 10. Lines 3–8 check, for a given load distribution, whether a server has a valid solution for Equation (1). If not, it is excluded from further consideration, with the available ISP link bandwidth adjusted as shown in Line 7. To understand this, note that the bandwidth contribution of such servers on the ISP link cannot exceed BW_i, because no additional connections (compressed or uncompressed) for this server can be supported once the server link is saturated. The recursive call uses this reduced value of available ISP link bandwidth and adjusted load distribution values (based on the relative contributions from remaining servers). Note those two invariants about load distribution (after adjustment, i.e. $\sum_i p_i = 1$) and bandwidth (i.e. $\sum_i \mathrm{BW}_i \geq \mathrm{BW}_I$) hold for each call upon a reduced graph. The algorithm terminates when all servers in S' have valid solutions for Equation (1). It is only these servers that can contribute a portion of their computation budget to the ISP node. The amount that can be transferred is easily determined by picking the minimum value C'_i for each such server that still results in Equation (1) being satisfied.

Figure 11 illustrates this algorithm using an example system consisting of 5 servers with computation budgets 10, 20, 30, 50, and 60 units; with client connection paths requiring $\mathrm{TH} = 1$, $c = 1$, and $D = 0.5$. Client requests are

uniformly distributed amongst these servers. The first call to the *Distribute* routine results in servers 4 and 5 being removed from S' because their server links cannot sustain $\frac{205}{5}$ connections. The second call removes server 3 because it cannot sustain $\frac{160}{3}$ connections. The algorithm terminates on the third call when both servers 1 and 2 can sustain $115/2$ connections, while contribute the computation resource amounts shown to the ISP node.