Towards a Theory of Optimal Communication Pipelines*

Randolph Y. Wang† Arvind Krishnamurthy† Richard P. Martin†
Thomas E. Anderson‡ David E. Culler†

Abstract

In this paper, we study how to minimize the latency of a message through a network that consists of a number of store-and-forward stages. This research is especially relevant for today's low overhead communication subsystems that employ dedicated processing elements for protocol processing. We develop an abstract pipeline model that reveals a crucial performance tradeoff. We subsequently exploit this tradeoff and present a series of fragmentation algorithms designed to minimize message latency. We provide an experimental methodology that enables the construction of customized pipeline algorithms that can adapt to the specific pipeline characteristics and application workloads. By applying this methodology to the Myrinet-GAM system, we have improved its latency by up to 51%. We also study the effectiveness of this technique for other realistic cases.

1 Introduction

The goal of this research is to answer a simple question: how do we minimize the latency of a message through a network that consists of a number of store-and-forward stages?

This question arose during our effort to improve the performance of a distributed file system [2] on a high-speed local area network [3]. Two important characteristics of the communication pattern distinguish the file system from the supercomputer applications which traditionally run on these networks. The first is the synchronous nature of the communication. While many supercomputer applications tend to communicate asynchronously to mask communication latency [13], a read miss in the file system cache, for example, blocks the application until the entire file block arrives. The second distinguishing characteristic is message size. Traditional high-speed network research has focused on minimizing the latency of *small* messages (of a few words) and obtaining saturating bandwidth for *bulk* messages. The file system, on the other hand, reads and writes file blocks (4KB or 8KB *medium* messages), whose performance characteristics, as we will see, are governed by a model that is not well understood. These communication characteristics are not only common in other high performance distributed file systems [12, 19], they are also shared by distributed shared memory systems [10, 18, 6, 9] and database applications.

Although traditional communication layers such as TCP/IP have examined the use of fragmentation in the context of managing congestion, buffer overflow, and errors in the wide area [15, 16], they have not systematically addressed the issue of optimizing latency by properly fragmenting these medium messages as the high host overhead in these systems has made fine-grained fragmentation infeasible.

Recent developments in high performance local area network technology have necessitated revisiting the message fragmentation issues. As network speed continues to increase, although the network fabric often supports cut-through routing, the processing elements in the network interface introduce store-and-forward delays. Choosing the appropriate fragment size to exploit the inherent parallelism in this communication pipeline becomes a key issue. New communication software [21, 20] has significantly reduced host processing overhead, which in turn has made it possible to use finer-grained fragmentation to increase the parallelism in the communication subsystem. Among such communication subsystems, however, different approaches exist. For example, Active Messages (AM2) [4] do not fragment medium messages, while Fast Messages (FM) [14] use 128 byte fragments. In order to evaluate the soundness of these design choices, one needs a systematic approach. The lack of an analytical model and the heavy reliance on simulation and empirical experience in some previous research efforts such as [8] have made it hard to generalize their results to systems other than their own.

In this paper, we develop a framework that may lead to a complete theory of optimal communication pipelines. We demonstrate several important optimality criteria. We show that an optimal fragmentation strategy depends on the user packet size, and minimizing latency requires carefully considering the tradeoff

*This work was supported in part by the Defense Advanced Research Projects Agency (N00600-93-C-2481, F30602-95-C-0014), the National Science Foundation (CDA 9401156), California MICRO, the AT&T Foundation, Digital Equipment Corporation, Sun Microsystems, Hewlett Packard, IBM, Intel, Microsoft, Mitsubishi, Siemens, and Xerox Corporation. Anderson was also supported by a National Science Foundation Presidential Faculty Fellowship.

†Computer Science Division, University of California, Berkeley, {rywang,arvindk,rmartin,culler}@cs.berkeley.edu

‡Department of Computer Science and Engineering, University of Washington, Seattle, {tom}@cs.washington.edu

involving the effects of the overhead of the bottleneck stage and the bandwidth of the remaining stages.

We provide a methodology that systematically uncovers communication pipeline parameters and constructs customized pipeline algorithms. We present empirical studies comparing theory to practice. In one example, we show that the discrepancy between the model prediction and the implementation measurement on Myrinet-GAM [13] averages 5.9%. By applying the fixed-sized fragmentation to this example system, we have achieved a performance improvement of up to 51%. We extend the study to important hypothetical pipelines including disks.

The remainder of the paper is organized as follows. Section 2 defines the abstract problem and lays the foundation for the pipeline model. Section 3 and Section 4 present a series of fragmentation algorithms. Section 5 presents the results of the case studies. Section 6 describes some of the related work. Section 7 concludes.

2 Problem Statement

We represent the network as a sequence of store-and-forward pipeline stages characterized by the following parameters:

- $\boldsymbol{n}$: the number of pipeline stages.
- $\boldsymbol{g}_i$: the fixed per-fragment *overhead* for stage i.
- $\boldsymbol{G}_i$: the per-byte cost (*inverse bandwidth*) for stage i.

Given these network characteristics, the challenge is to find an optimal fragmentation strategy of a user packet so that its latency is minimized. Note that fragmentation is only useful in minimizing the latency of a finite-sized packet; for steady-state data streams, the bandwidth increases monotonically with fragment size and asymptotically approaches the bottleneck bandwidth. Also, we observe that if none of the stages have overhead, the best fragmentation policy is trivial – parallelism is maximized at the maximum degree of fragmentation possible. Similarly, if none of the stages have bandwidth limitations, then the best policy is the opposite – we send the entire packet in a single fragment. For a general pipeline that has both overhead and bottleneck limitations, we now develop a model for calculating packet latency using the following variables:

- $\boldsymbol{B}$: the size of the entire packet.
- $\boldsymbol{k}$: the number of fragments.
- $\boldsymbol{x}_i$: the size of the ith fragment ($\sum_{i=0}^{k-1} x_i = B$).
- $\boldsymbol{t}_{i,j}$: the time the ith fragment spends in the jth stage, which is $t_{i,j} = g_j + x_i \cdot G_j$.
- $\boldsymbol{\tau}_{i,j}$: the time at which the ith fragment exits the jth stage. The clock starts when the first fragment enters the first stage.
- $\boldsymbol{T}$: the latency of the entire packet, equivalent to $\tau_{k-1,n-1}$.

Figure 1 shows an example three-stage store-and-forward pipeline. In Figure 1(a), the user packet is transmitted as a single fragment and its latency through the pipeline is longer than that shown in Figure 1(b) where the same packet is fragmented into three pieces. This is because when the packet size is sufficiently large, the benefit achieved from increased parallelism in the latter case is larger than the overhead introduced by the extra fragments.

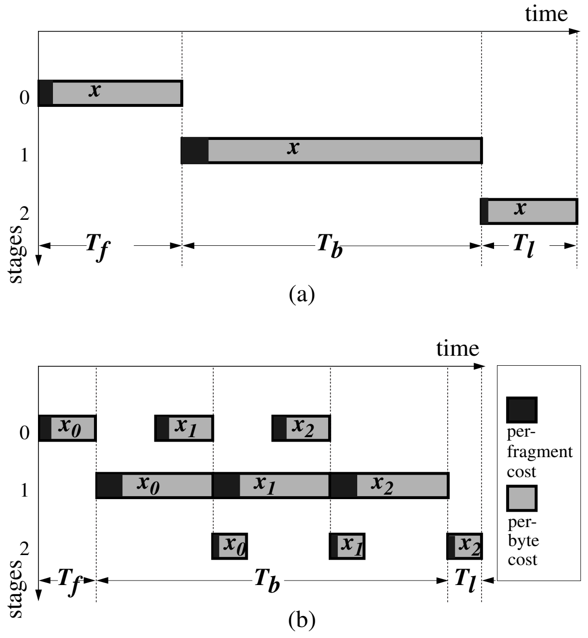

Figure 1: An example pipeline. Stage 1 is the bottleneck. In (a), the user packet is not fragmented. In (b), the same user packet (drawn to the same scale) has been fragmented into three pieces and experiences a lower latency. Minimizing the sum of the three components of the total latency (T_f, T_b, and T_l) is our goal.

Figure 1 also shows the constraints imposed by a store-and-forward pipeline. A fragment can enter the next stage as soon as it exits the current stage in its entirety. Also, a fragment can not enter a stage before the previous fragment exits the same stage. These constraints can be translated into the following system of linear inequalities:

$$\tau_{0,j} = \sum_{h=0}^{j}(g_h + x_0 \cdot G_h) \qquad (1)$$

$$\tau_{i,l} \geq \tau_{i,l-1} + (g_l + x_i \cdot G_l) \qquad (2)$$

$$\tau_{m,j} \geq \tau_{m-1,j} + (g_j + x_m \cdot G_j) \qquad (3)$$

where $0 \leq i < k$, $0 \leq j < n$, $1 \leq l < n$, and $1 \leq m < k$. Given the number of fragments k, we can find the optimal fragmentation strategy by solving this linear program where minimizing $\tau_{k-1,n-1}$ is the objective. By repeating this process for all possible values of k, we can find the sequence of optimal fragment sizes.

This exhaustive search, however, has a number of disadvantages. It does not provide much insight into the characteristics of an optimal fragmentation strategy; neither does it show how changes in the pipeline parameters affect the end-to-end latency. This linear system with its limitation of a known number of fragments does not model reassembly or refragmentation, which might be necessary for pipeline stages with high overheads. The linear system also requires full knowledge of all pipeline parameters, which might not always be readily available.

In contrast, the pipeline models that we develop in the rest of this paper explicitly illustrate the tradeoffs that determine the optimal fragmentation strategy. The models can also serve as a guideline for communication subsystem designers by quantifying how the speeds of individual stages impact the overall performance. Some of our models also work with more limited information of the pipeline than that required by the linear system.

The fragmentation algorithms we develop share a number of common themes, which we will explore in greater detail in the following sections. In our models, the contribution of the "slowest" pipeline stage to the total packet latency is a key issue. We define the *bottleneck* stage to be the stage whose transfer time for a fragment is the greatest. We denote the bottleneck as the bth stage, and its characteristics by $(\boldsymbol{g}_b, \boldsymbol{G}_b)$. Figure 1 illustrates some of the properties of the bottleneck stage. First, the bottleneck stage (stage 1) in the figure is kept busy (except at the beginning and towards the end). We will explain why this is necessary for minimizing end-to-end latency. Second, if we assume the bottleneck is kept busy, then the total packet latency can be represented as:

$$T = T_f + T_b + T_l \tag{4}$$

where

- $\boldsymbol{T_f}$ is the time the *first fragment* takes to reach the bottleneck,
- $\boldsymbol{T_b}$ is the time the *entire packet* spends in the bottleneck, and
- $\boldsymbol{T_l}$ is the time the *last fragment* takes to exit the pipeline after leaving the bottleneck.

How to trade off one of these components against the others is a crucial question that we shall explore in all the models that we develop in the subsequent sections. The third observation is a weak lower bound of the total latency:

$$T > B \cdot G_b + \sum_{j=0}^{n-1} g_j \tag{5}$$

In this lower bound, the sizes of the leading and trailing fragments are both zero, and the packet travels through the bottleneck stage as a single fragment. Our goal in the following sections is to devise fragmentation models that are as close to this lower bound as possible.

3 Fixed-sized Fragmentation

We first explore fragmenting a packet into fragments of equal size. Section 4 considers the more general case in which fragment sizes can vary. We derive the optimal fixed fragment size assuming the existence of a stage that is the bottleneck for all possible fragment sizes. Then we generalize our approach for pipelines whose slowest stage depends on the fragment size.

3.1 Optimal Fragment Size

Restricting fragments to a uniform size implies that once the bottleneck stage starts operating, it never idles until the last fragment leaves it. Therefore Equation (4) holds. To minimize T, we must minimize $(T_f + T_b + T_l)$. Minimizing $(T_f + T_l)$ requires small fragments, because as we decrease the size of the fragments, the amount of time the first fragment takes to reach the bottleneck (T_f) decreases; so does the amount of time the last fragment takes to leave the bottleneck (T_l). On the other hand, minimizing T_b requires large fragments, because fewer fragments incur less overhead in the bottleneck. This is a fundamental tradeoff.

In practice, the fragment size x_i and the number of fragments k must be positive integers. To simplify the discussion, we develop the subsequent theorems assuming a definition of the packet latency T as a *continuous* function of x_i or k.

Theorem 1 (Fixed-sized Theorem) *The fragment size x_i that minimizes the latency function T is:*

$$\sqrt{\frac{B \cdot g_b}{\sum_{j \neq b} G_j}} \tag{6}$$

Proof. We express T_f as a sum of the times the lead fragment spends in the stages leading to the bottleneck:

$$\begin{aligned} T_f &= \sum_{j=0}^{b-1} t_{0,j} \\ &= \sum_{j=0}^{b-1} (g_j + x_0 \cdot G_j) \end{aligned} \tag{7}$$

Similarly, T_l is the sum of the times the last fragment spends in the stages after the bottleneck:

$$T_l = \sum_{j=b+1}^{n-1} (g_j + x_{k-1} \cdot G_j) \tag{8}$$

Each fragment spends an equal amount of time in the bottleneck and there are k fragments:

$$\begin{aligned} T_b &= k \cdot t_{0,b} \\ &= k \cdot (g_b + x_0 \cdot G_b) \end{aligned} \tag{9}$$

We sum equations (7), (8), and (9) to obtain the total latency T, and substitute x_0 with B/k:

$$\begin{aligned} T &= T_f + T_l + T_b \\ &= \sum_{j \neq b} (g_j + x_0 \cdot G_j) + k \cdot (g_b + x_0 \cdot G_b) \\ &= \sum_{j \neq b} (g_j + \frac{B}{k} \cdot G_j) + (k \cdot g_b + B \cdot G_b) \end{aligned} \tag{10}$$

To obtain the optimal number of fragments, we differentiate (10) with respect to k, and set the result to 0:

$$\frac{dT}{dk} = -\frac{B \cdot \sum_{j \neq b} G_j}{k^2} + g_b = 0 \qquad (11)$$

Thus the optimal number of fragments is:

$$k = \sqrt{\frac{B \cdot \sum_{j \neq b} G_j}{g_b}} \qquad (12)$$

The result in (6) follows by dividing (12) into B. □

In practice, when we apply the fragmentation model summarized by (12), k must be an integer in $[1, B]$. Due to the shape of the latency function $T(k)$ (as shown in Figure 2(a)), we only need to apply the floor and ceiling functions to (12) and choose the integer solution that minimizes T.

We make several observations of this model. First, minimizing the latency of a packet requires the fragmentation strategy to adapt to the packet size (B). A static algorithm (such as the one used in FM) can achieve the same result for a single packet size and a single pipeline, but it becomes suboptimal as we move away from that design point. Second, we must balance the effects of the overhead of the bottleneck (g_b) and the bandwidth of the remaining stages ($\sum_{j \neq b} G_j$). In particular, if the overhead of any stage becomes large relative to inverse bandwidth characteristics of other stages, fragmentation becomes unnecessary ($k = 1$ and $x_i = B$). Third, interestingly, factors such as the overheads of the non-bottleneck stages and bandwidth of the bottleneck stage do not affect the fragmentation strategy.

3.2 Generalizing Bottleneck Definition

The derivation of the Fixed-sized Theorem assumes the existence of a bottleneck which is *always* the slowest stage. A sufficient condition is when a single stage has both the worst overhead and bandwidth. In this section we generalize the definition of a bottleneck – a stage (g_b, G_b) is the bottleneck when its latency $(g_b + G_b \cdot x)$ is the greatest for *some* range of fragment sizes (x). We generalize the fixed-sized fragmentation model for the new definition.

Equation (10) can also be expressed as:

$$T = k \cdot g_b + \frac{B}{k} \cdot \sum_{j \neq b} G_j + (\sum_{j \neq b} g_j + B \cdot G_b) \qquad (13)$$

If the location of the bottleneck never changes, the latency of a packet that is fragmented into k fragments can be expressed as a function of the form $c_1 \cdot k + c_2/k + c_3$, where c_1, c_2, and c_3 are constants. Figure 2(a) shows the typical shape of this function. We can regard this curve as the "signature" curve of the pipeline and, in particular, of its bottleneck. If the location of the bottleneck depends on the fragment size, then the latency function becomes a concatenation of segments from different bottleneck signature curves, as shown by the solid curve in Figure 2(b). In this example, stage p (whose signature is the thin curve) is initially the bottleneck. As we increase the degree of fragmentation beyond point A, stage q (whose signature is the thick curve) becomes the bottleneck instead.

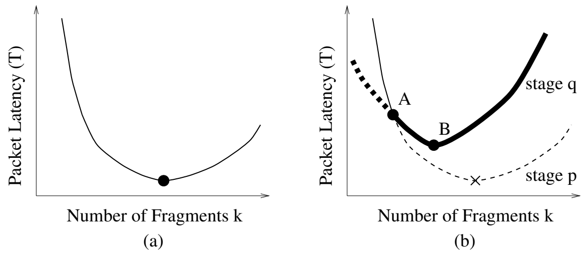

Figure 2: Pipeline latency curves. In (a), one stage remains the bottleneck for all fragment sizes. In (b), two stages can be bottlenecks for different ranges of fragment sizes. The solid curve is the latency curve. The circles mark the candidate points for the global minimum. The cross marks a false local minimum.

Finding the optimal fragmentation size requires finding the global minimum on the latency curve. For cases such as the one shown in Figure 2(b), we must 1) locate the transition points for the latency function (such as point A) by applying a methodology detailed in Section 5.3.2, 2) find the local minimums of all signature curves that fall on the latency curve (such as point B) by repeated application of the Fixed-sized Theorem, and 3) take the minimum of all these candidate points.

4 Variable-sized Fragmentation

In this section, we explore algorithms that allow packets to be fragmented into variable-sized fragments. The motivation is to have smaller leading and trailing fragments to minimize $(T_f + T_l)$ of Equation (4), while the fragments in the middle are larger to minimize the overhead component of T_b. We will study two-stage and three-stage pipelines to gain more insights before we examine more general cases.

4.1 Two-stage Pipelines

In this section, we develop an optimal fragmentation strategy for an arbitrary two-stage pipeline.

4.1.1 Reversing a Pipeline

While considering the solutions of a two-stage pipeline and its reversed counterpart whose stages are arranged in the opposite order, we saw that the reversal of the optimal solution of one pipeline was the optimal solution of the other. Unlike the other theorems for two-stage pipelines in this section, this observation applies to more general pipelines as well:

Theorem 2 (Reversibility Theorem)
If $(x_0, x_1, \ldots, x_{k-1})$ is an optimal fragmentation of B

bytes for a pipeline characterized by (g_0, G_0), (g_1, G_1), ..., *and* (g_{n-1}, G_{n-1}), *then* $(x_{k-1}, x_{k-2}, \ldots, x_0)$ *is an optimal fragmentation of* B *bytes for a pipeline characterized by* (g_{n-1}, G_{n-1}), (g_{n-2}, G_{n-2}), ..., *and* (g_0, G_0).

The Reversibility Theorem is an intuitively simple outcome, but it is a powerful tool for understanding pipeline solutions that are symmetrical to known ones.

4.1.2 A "Bubble-free" Pipeline

Theorem 3 (No-stall Theorem) *For a two-stage pipeline, a necessary condition of an optimal fragmentation solution is* $t_{i+1,0} = t_{i,1}$*, where* $t_{i,j}$ *is the transfer time for fragment* i *through stage* j.

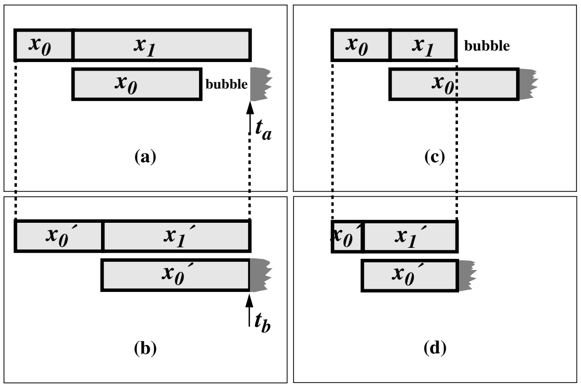

Figure 3: Bubble-free pipelines. A bubble in the second stage in (a) is eliminated by increasing x_0 in (b). A bubble in the first stage in (c) is eliminated by decreasing x_0 in (d). Note that according to our definition, the first stage of pipeline (c) is considered to contain a bubble regardless of whether fragment x_1 is the last.

Proof. More informally, the theorem states that the first stage completes the transfer of a fragment exactly when the second stage completes the transfer of the previous fragment. We sketch the proof using Figures 3(a) and (c), which are examples that violate this condition. Figures 3(b) and (d) show how a better fragmentation can be achieved by matching the transfer times of the two stages by resizing the fragments while keeping the sum of the two fragments constant. At time $t_b = t_a$, the pipeline in Figure 3(b) has achieved a better state than that of (a) because its second stage has transferred more bytes. A similar argument applies if the bubble occurs anywhere in the second stage and we resize all the fragments leading to the bubble to eliminate it. And finally, because a bubble in the first stage is equivalent to a bubble in the second stage of the reversed pipeline, it follows from the Reversibility Theorem that a pipeline which contains bubbles in its first stage is also suboptimal. □

4.1.3 "Ramp-up" Algorithm

Theorem 4 (Ramp-up Theorem) *For a two-stage pipeline, the optimal fragment sizes* x_i *follow the recurrent relationship:*

$$x_{i+1} = \frac{g_1 - g_0}{G_0} + x_i \cdot \frac{G_1}{G_0}$$

Given any number of fragments k*, there is always a unique initial fragment size* x_0 *that leads to a bubble-free solution.*

Proof. Using the No-Stall Theorem:

$$\begin{aligned} t_{i+1,0} &= t_{i,1} \\ g_0 + x_{i+1} \cdot G_0 &= g_1 + x_i \cdot G_1 \end{aligned}$$

Thus,

$$x_{i+1} = \frac{g_1 - g_0}{G_0} + x_i \cdot \frac{G_1}{G_0} \tag{14}$$

To simplify the notations, we define:

$$a = \frac{G_1}{G_0} \tag{15}$$

$$b = \frac{g_1 - g_0}{G_0} \tag{16}$$

so that we can rewrite (14) as a function f of x_0, i, a, and b:

$$\begin{aligned} x_{i+1} &= a \cdot x_i + b \\ &= \begin{cases} x_0 \cdot a^{i+1} + b \cdot \frac{a^{i+1}-1}{a-1} & \text{if } a \neq 1 \\ x_0 + (i+1)b & \text{if } a = 1 \end{cases} \\ &= f(x_0, i+1, a, b) \end{aligned} \tag{17}$$

B can be expressed as another function h of the same variables:

$$\begin{aligned} B &= \sum_{i=0}^{k-1} x_i \\ &= \begin{cases} x_0 \frac{a^k-1}{a-1} + \frac{b}{a-1}\left(\frac{a^k-a}{a-1} - k + 1\right) & \text{if } a \neq 1 \\ kx_0 + \frac{k(k-1)b}{2} & \text{if } a = 1 \end{cases} \\ &= h(x_0, k, a, b) \end{aligned} \tag{18}$$

From (18), the initial fragment size x_0 can be solved given k. □

According to Theorem 4, if the second stage is slower, then the fragment size gradually increases; hence the name "Ramp-up Theorem". If the second stage is faster, then the fragment size gradually decreases. In general, the Ramp-up Theorem dictates that the fragment size is a monotonic function even if the bottleneck stage depends on the fragment size.

The Ramp-up Theorem leads to a practical way of finding an optimal fragmentation for two-stage pipelines. We can express the total latency T in terms of k and follow the same approach as that of the Fixed-sized Theorem to find integer solutions of k that minimizes T.

To understand intuitively why ramp-up fragmentation is an improvement over fixed-sized fragmentation, we revisit the tradeoff articulated by Equation (4). Notice that the No-stall Theorem guarantees that the bottleneck stage never idles once it begins its operation. Therefore, if we compare a ramp-up fragmentation against a fixed-size fragmentation which has the same number of fragments k, because the fragments in the new algorithm successively increase in size, the initial fragment x_0 is smaller, and therefore it has a smaller "lead" time T_f. On the other hand, if we compare a ramp-up fragmentation against a fixed-sized fragmentation which has the same initial fragment x_0, again because the fragment sizes in the new algorithm increase, the new algorithm has fewer fragments, and therefore the packet spends a smaller amount of time in the bottleneck (T_b). By decreasing T_f or T_b or both, the ramp-up algorithm is able to out perform the fixed-sized algorithm.

4.2 A "Fast-slow-fast" Pipeline

In this section, we extend our results to a three-stage pipeline that consists of a fast stage, a slow stage, and another fast stage. A fundamental characteristic of the optimal fragmentation strategy is formalized in the following theorem and illustrated by Figure 4.

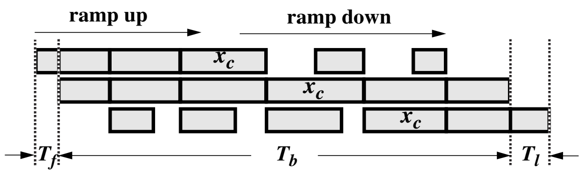

Figure 4: "Ramp-up-and-down" algorithm. Fragment size gradually increases to keep the first two stages busy until it reaches x_c; then it decreases to keep the last two stages busy.

Theorem 5 (Ramp-up-and-down Theorem) *Suppose $(x_0, x_1, \ldots, x_{k-1})$ is a fragmentation solution for a "fast-slow-fast" pipeline, a necessary condition for this solution to be optimal is that there exists a fragment x_c, $0 \le c < k$, such that $t_{i+1,0} = t_{i,1}$ for $0 \le i < c$ and $t_{i+1,1} = t_{i,2}$ for $c \le i < k$.*

Informally, the theorem states that fragments $(x_0, x_1, \ldots, x_c)$ monotonically increase in a manner that keeps the first two stages busy, while fragments $(x_c, x_{c+1}, \ldots, x_{k-1})$ monotonically decrease so as to keep the last two stages busy. Due to space limitations, we only provide a sketch of the proof. The proof consists of three steps. First, we can show that if the bottleneck stage is not kept busy at all times, the fragments can be resized (as in Figures 3(a) and (b) of the No-stall Theorem) to obtain a better fragmentation. Next we can show that there is no bubble in the first stage for fragments $(x_0, x_1, \ldots, x_c)$ using an argument similar to the one shown by Figure 3(c) and (d). Lastly we show that there is no bubble in the last stage for the fragments $(x_c, x_{c+1}, \ldots, x_{k-1})$. To accomplish this step, we show that a bubble in the last stage after fragment x_c can be transformed into a bubble in the bottleneck stage, which we have already shown to be non-optimal.

We can utilize the necessary condition of Theorem 5 to develop an optimal fragmentation strategy. For convenience, we will use a pair of integers k_1 and k_2, where $k_1 + k_2 = k$, k_1 is the number of monotonically increasing fragments, and k_2 is the number of monotonically decreasing fragments. Suppose the sum of the monotonically increasing fragments is B_1 and the sum of the monotonically decreasing fragments is B_2:

$$B_1 + B_2 = B + x_c \tag{19}$$

Because $(x_0, x_1, \ldots, x_c)$ is a bubble-free solution for pipelining B_1 bytes through the first two stages, Equation (18) of the proof of the Ramp-up Theorem applies:

$$B_1 = h(x_0, k_1, a, b) \tag{20}$$

where a and b are given by Equations (15) and (16). If we similarly define c and d for the last two stages of the three-stage pipeline, we have:

$$B_2 = h(x_c, k_2, c, d) \tag{21}$$

But x_c is not only the first fragment of the "ramp-down" part of Equation (21), it is also the last fragment of the "ramp-up" part, therefore its relationship with x_0 can be deduced from Equation (14):

$$x_c = f(x_0, k_1 - 1, a, b) \tag{22}$$

Substitute (20), (21), and (22) into (19) and we have an equation in which the only free variable is x_0. We can solve for the unique x_0 which determines a unique solution that satisfies the necessary condition given by Theorem 5. We then express the total latency T as a function of k_1 and k_2, derive the partial derivatives, and solve $\partial T/\partial k_1 = \partial T/\partial k_2 = 0$ to yield the optimal fragmentation factors k_1 and k_2. The function $T(k_1, k_2)$, however, can be complicated; so in practice we simply search all feasible values of k_1 and k_2.

Note that the Ramp-up-and-down Theorem subsumes the simpler Ramp-up Theorem by setting either k_1 or k_2 to one. However, unlike the Ramp-up Theorem, the Ramp-up-and-down Theorem has the limitation that it assumes the bottleneck stage to be the slowest for all fragment sizes.

4.3 A "Fast-slow-slower" Pipeline

In this section, we study a three-stage pipeline that consists of a fast stage, a slow stage, and a third stage that is slower yet. We show how recursive applications of the algorithms discussed in the previous sections can naturally lead to reassembly when necessary. We then generalize the approach for other pipelines.

4.3.1 Hierarchical Fragmentation

In the algorithms that we have discussed so far, a fragment always travels through all stages without being refragmented into smaller *sub-fragments* or being reassembled into larger ones. Fragmentation strategies with this restriction are no longer optimal when the third stage (the bottleneck) has a large transfer cost either due to high overhead or low bandwidth.

If the third stage has a high overhead cost, in order to amortize the large overhead and minimize T_b (of Equation (4)), we must start with a large initial fragment x_0. This large initial fragment, unfortunately, may not be appropriate for the first two faster pipeline stages. In order to minimize the time it takes to get x_0 through these two stages (T_f), we may need to break down x_0 into even smaller sub-fragments and treat the first two stages as a sub-pipeline.

Next consider a third stage whose bandwidth is much lower than those of the first two stages but has a comparable overhead. Although the initial fragment x_0 for the bottleneck may no longer need reassembly, we now must prepare a second fragment x_1 for the bottleneck while it is busy processing x_0 at a low bandwidth. In order to minimize the number of fragments required for the bottleneck to keep T_b low, we must find the largest possible x_1. This again may require treating the first two stages as a sub-pipeline so that we can choose optimal sub-fragment sizes in order to assemble the largest possible x_1 for the bottleneck. We now see that reassembly may be necessary to accommodate high overhead, low bandwidth, or both of the bottleneck stage.

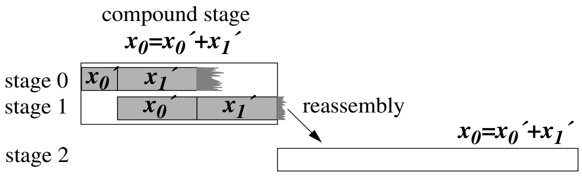

Figure 5: Hierarchical fragmentation. The first two stages of the three-stage pipeline are treated as a *compound stage*. Small fragments exiting from the compound stage are reassembled into larger ones for the bottleneck stage.

Figure 5 illustrates this hierarchical approach. In this example, we reduce a three-stage pipeline to a two-stage *compound pipeline*, whose first stage is a *compound stage*, which in turn consists of two *internal stages*. The number of fragments used for the compound pipeline uniquely determines the sequence of fragment sizes for the compound pipeline, which in turn determines how each of these fragments is transmitted through the first two stages, potentially at even finer granularity.

We note that this hierarchical approach is orthogonal to the base fragmentation strategies used. In other words, we can use the fixed-sized fragmentation (of Section 3), or any of the variable-sized fragmentation algorithms (of Section 4), or a mixture of these algorithms for the different levels of the hierarchy.

Unfortunately, the mathematics involved in the precise modeling of this hierarchical approach becomes rather complicated. To simplify the model, we approximate a compound stage by the model of a simple stage. We model its latency characteristics with a per-fragment overhead and a per-byte cost. Although this is not strictly accurate, empirical experiences suggest that this is an effective approximation[1]. We also assume that a compound stage becomes free only when the last internal stage becomes free. Again, this is not strictly accurate because the earlier internal stages in a compound stage become free earlier. If the base fragmentation strategy is variable-sized, a precise model must consider how the earlier internal stages should utilize this extra time. But as we shall see in Section 5.5, hierarchical fragmentation is only useful for pipelines in which the bottleneck latency is sufficiently large that the the slight loss of free time towards the end of the (non-bottleneck) compound stage is not significant.

With these simplifications, modeling hierarchical fragmentation becomes straightforward. First, we apply either the fixed-sized or variable-sized fragmentation model to the internal stages of the compound stage. Next we approximate the compound stage with a simple pipeline stage by finding its overhead and bandwidth. Finally we apply the fixed-sized or variable-sized fragmentation model to the compound pipeline using the approximation. We can improve the accuracy of the model by iteratively narrowing the range of the fragment sizes used for the approximation in the second step.

4.3.2 Generalizations

The Reversibility Theorem of Section 4.1 allows us to generalize our fragmentation strategy to other pipelines. First, reversing the first two stages of the "fast-slow-slower" configuration does not change the characteristics of the compound stage of the compound pipeline. Therefore, the fragmentation strategy for a "slow-fast-slower" compound pipeline remains the same, but the internal fragmentation of the compound stage is the reversal of that of the original pipeline. Second, if we reverse the entire pipeline and reach a "slowest-slow-fast" configuration, it immediately follows from the Reversibility Theorem that the solution of the new pipeline is the reversal of that of the original one. In this case, instead of reassembly for the bottleneck, we have refragmentation after leaving the bottleneck.

4.4 Discussion

Here we briefly review the fragmentation strategies that we have studied in the past sections:

- **Fixed-Sized fragmentation** works well for pipelines whose stages have comparable speeds.

[1] In one typical experiment of Section 5.5, the standard deviation of errors of the approximation was $65\mu s$, for a mean latency of $5118\mu s$.

- **Ramp-up fragmentation and ramp-up-and-down fragmentation** are provably optimal for two-stage and "fast-slow-fast" pipelines respectively. They allow us to evaluate the effectiveness of approximate algorithms by providing an upper bound on performance. They also provide insight into an optimal algorithm for arbitrary pipelines.
- **Hierarchical fragmentation** works well when one pipeline stage is significantly slower than all other stages.

A fragmentation algorithm for a general pipeline combines these strategies. We identify the bottleneck stage. Then we apply the fragmentation algorithm recursively to construct models for the compound stage before the bottleneck and the stage after. We then apply the ramp-up-and-down algorithm to the top level compound pipeline. In the next section, we see that a simple combination of fixed-sized fragmentation and hierarchical fragmentation, in which we apply fixed-sized fragmentation to each level of the hierarchical fragmentation, is an effective approximation.

5 Experimental Results

In this section, we apply the analytical models to realistic case studies. We present a systematic methodology for constructing pipeline algorithms that adapt to network characteristics and user packet sizes. We evaluate the accuracy and effectiveness of the models with a combination of implementation and simulation studies.

5.1 Methodology

To construct a fragmentation algorithm for a specific pipeline, we first need sufficient details of the pipeline characteristics. We obtain these parameters through either "clear-box" instrumentation, where we instrument the pipeline stages to obtain their overhead and bandwidth characteristics, or "black-box" measurement, where we deduce the combined effects of the individual stages by non-intrusively observing how the network responds to different workloads. The customized pipeline algorithm adapts the fragmentation at run time to different user packet sizes. We can also monitor changes of the pipeline parameters, such as those due to contention, and fine-tune the fragmentation at run time resulting in a customized fragmentation algorithm that adapts to user packet sizes and network characteristics.

5.2 Experimental Platform

Our main experimental platform is the Berkeley NOW [1, 5], which is a cluster of UltraSPARC Model 170 workstations running at 167 MHz. The host operating system is Solaris 2.5. The workstations are connected by Myrinet [3]. Each workstation is equipped with a Myricom M2F network interface card on the SBUS. Figure 6 shows the block diagram of the interface card. It contains a host DMA engine, which moves data from host main memory to the SRAM on the network interface, a network DMA engine, which moves data from the SRAM into the network, and a 37.5 MHz LANai processor, which executes the messaging protocol and is responsible for coordinating the actions of the DMA engines and interactions with the host. The presence of these processing elements is the source of the parallelism which we shall exploit in the pipelining algorithms. We perform our experiments on two workstations connected via a Myricom M2F switch, which supports a link bandwidth of 160 MB/s.

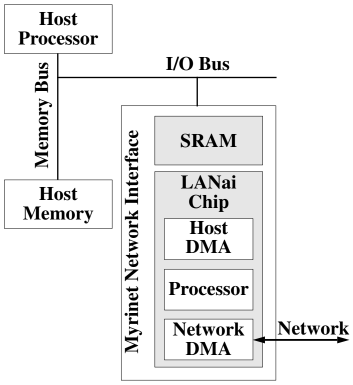

Figure 6: Myrinet network interface. To send a medium message, the host DMA moves the data from host memory into the on-board SRAM, then the network DMA moves the data from the SRAM into the network.

The base communication subsystem we use in the study is Generic Active Messages (GAM) [13], a version of Active Messages [20] enhanced to support cluster applications. One notable feature of this communication layer is its low overhead, which has enabled fine-grained fragmentation that would be infeasible on systems with higher overhead.

In order to experiment with pipeline parameters that are different from the Berkeley NOW, we also supplement our study with a pipeline simulator.

5.3 GAM Pipeline Parameters

The first step towards constructing a customized pipeline algorithm is determining the pipeline parameters. In this section, we present the direct instrumentation approach, which provides exhaustive information but requires source code access, and the indirect observation, which provides just enough information for the fixed-sized fragmentation strategy and is less intrusive.

5.3.1 Direct Instrumentation

We instrument the source code that runs at each stage of the pipeline. We then send packets of various sizes through the data path *without* pipelining to obtain a timing vector. From this vector, we obtain the overhead and bandwidth characteristics of each stage by a simple linear regression.

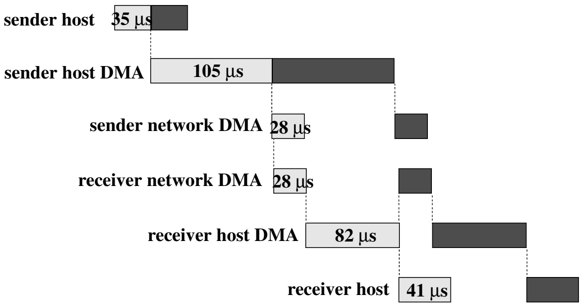

Figure 7: The timing of two 4KB fragments in the Myrinet GAM pipeline.

Before we transform these measurements into an abstract pipeline model, we examine the data path of our experimental platform more closely. Figure 7 shows the path of two 4KB fragments sent back-to-back through the network. The steps labeled "sender host" and "receiver host" correspond to data copying on the two hosts. The step labeled "sender host DMA", which occurs over the SBUS, is the slowest. The steps labeled "sender network DMA" and "receiver network DMA" almost completely overlap each other because the network fabric is cut-through, whereas the other components at the network interface are store-and-forward. The step labeled "receiver host DMA" is faster than "sender host DMA" because of the asymmetry of the SBUS. In fact, since the sum of the time spent in the receiver network DMA and the receiver host DMA roughly equals that of the sender host DMA, a conscious design decision was made in GAM not to pipeline the receiver network and host DMA's; they are implemented as one sequential step. As a result, the third, fourth, and fifth steps in figure 7 are combined into a *single* abstract pipeline stage. Table 1 summarizes the pipeline parameters.

stage i	g_i (μs)	G_i (μs/KB)
0	7.2	7.2
1	5.2	24.9
2	7.5	24.9
3	7.4	7.9

Table 1: Myrinet GAM pipeline parameters. Stages 0 and 3 are the copies on the end hosts. Stage 1 is the sender host DMA. Stage 2 includes the two network DMA's and the receiver host DMA and is the bottleneck.

5.3.2 Indirect Measurement

Direct instrumentation, while providing much insight, requires access to and careful analysis of the source code of the communication subsystem, which is not always feasible. In this section, we demonstrate how to indirectly deduce the pipeline parameters by observing the performance of the network at the interface level.

In the Fixed-sized Theorem of Section 3.1, we had shown that it is not necessary to have the parameters of *each* stage to construct the desired algorithm. Instead, we only need two parameters: the overhead of the bottleneck (g_b) and the sum of the inverse bandwidths of the remaining stages ($\sum_{j\neq b} G_j$).

We take a two-step approach to indirectly obtain these parameters. We first send packets of various sizes through the network *without* pipelining them and measure their latencies. The network behaves as a single stage pipeline whose parameters are ($\sum g_i$, $\sum G_i$). By performing a linear regression on the packet sizes and the corresponding latencies, we obtain these sums: ($\sum g_i$, $\sum G_i$). In the second step, we send packets into the network as quickly as we can, and measure the frequency at which they exit from the pipeline. The inter-arrival time of the packets at the receiver is the time the packet spends in the bottleneck stage. By varying the packet sizes and running another linear regression, we obtain the bottleneck parameters: (g_b, G_b). Subtracting G_b from $\sum G_i$, which was obtained in the first step, we can compute $\sum_{j\neq b} G_j$. Table 2 compares the results of the indirect instrumentation with the direct measurements.

method	g_b (μs)	$\sum_{j\neq b} G_j$ (μs/KB)
direct	7.5	40.0
indirect	7.6	37.8

Table 2: Parameters for fixed-sized fragmentation.

If the bottleneck varies with fragment size, the fixed-sized fragmentation algorithm requires us to identify the range of fragment sizes for which a pipeline stage is the bottleneck. To find these ranges, we first apply the methodology above to find the bottleneck parameters for a fragment size that is the lower bound of an allowable user packet size. We then repeat the process for the upper bound of the allowable user packet size. If the two bottleneck stages match, then we know the bottleneck does not change location. If these two bottlenecks differ, a third measurement is required in the middle of the previous two fragment sizes. We repeat the process for each of the two half ranges until we locate all the transition points.

5.4 Evaluation of Fixed-sized Fragmentation

We use the GAM parameters obtained in the previous section to implement the fixed-sized fragmentation. We first validate the model presented in Equation (10) of Section 3 by comparing it against the measurements of the implementation. Figure 8 shows the result. In this example, we show how the 4KB-packet latency varies as a result of varying the fragmentation granularity. The

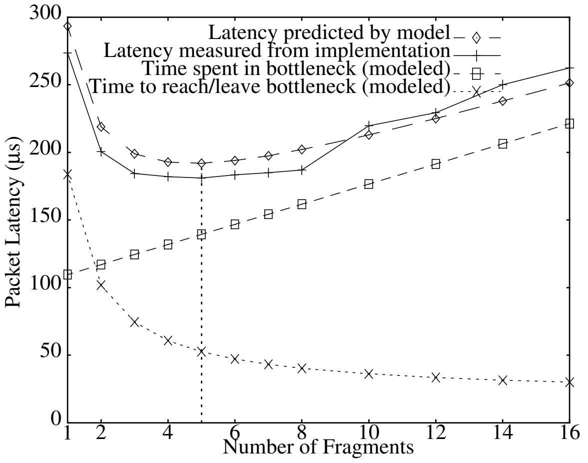

Figure 8: Validation of the fixed-sized fragmentation model for 4KB packets.

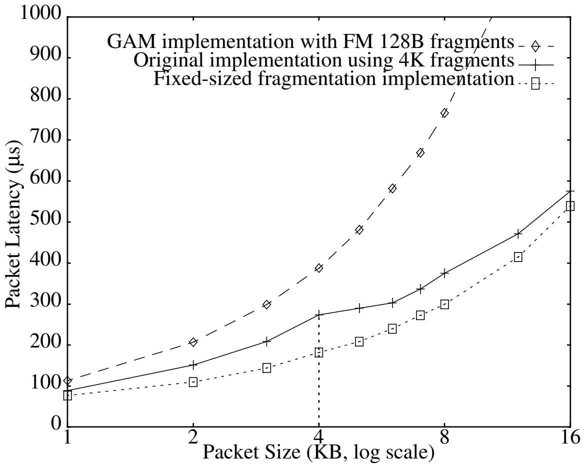

Figure 9: Latency comparison of fixed-sized fragmentation versus static fragmentation.

model prediction has an error that averages 5.9%. The model of Equation (12) predicts that the best latency is achieved when the number of fragments is five. This prediction is confirmed by the implementation measurement.

Figure 8 also illustrates the tradeoff presented in Equation (4). As the number of fragments increases, the time the packet spends in the bottleneck increases due to the overhead introduced by each additional fragment. At the same time, however, as the fragment size decreases, the time the first fragment takes to reach the bottleneck stage decreases; so does the amount of time the last fragment takes to leave the pipeline after the bottleneck.

We compare the performance of our fixed-sized fragmentation algorithm with that of the original GAM implementation and that of FM [14], both of which always use a static fragmentation strategy (128B for FM and 4KB for the original GAM). Figure 9 shows the result. FM suffers from the high overhead incurred by the small fragments for large packets. With respect to GAM, the fixed-sized fragmentation achieves the largest performance gain of 51% for a packet size of 4KB. The performance gain diminishes for large packets as the bandwidth of the bottleneck stage becomes the more dominant limiting factor. The improvement is also small for smaller packets due to the inherent overhead in the pipeline, which prevents fine-grained fragmentation. We also notice that the latency curve for GAM is jagged beyond 4KB as the fragments of a packet are not of equal size unless the packet size is evenly divisible by 4KB. The curve for the fixed-sized fragmentation implementation, on the other hand, is smooth as it dynamically adapts to the user packet size and the fragments of a single packet are always equal in size.

5.5 Evaluation of Variable-sized Fragmentation

In this section, we study the utility of the variable-sized fragmentation strategies. When the pipeline stages have similar speeds (as in the GAM pipeline), fixed-sized fragmentation is sufficient. At the other extreme, when the pipeline is dominated by one or more very slow stages, fragmentation provides little improvement. It is in between these extremes that the variable-sized fragmentation is useful.

We study the performance of a simulated system that has a disk attached to a conventional network. The network speed is one fifth of that of Myrinet-GAM; it has five times the overhead of Myrinet-GAM and one fifth of its bandwidth. These parameters are comparable to that of conventional communication subsystems such as TCP running on a 100 Mb/s ethernet. As we are interested in the impact of the *relative* speed of the network to the rest of the system, we vary the overhead and bandwidth characteristics of the disk.

We study three algorithms. Algorithm A uses a fixed-sized fragmentation that treats the four internal stages of a network stage as peers to the disk stage. Algorithm B organizes the pipeline stages into hierarchies and applies the hierarchical fragmentation. Fixed-sized small fragments that are used for the internal stages of the network stage are reassembled into fixed-sized large fragments for the disk stage. Algorithm C further improves algorithm B by varying the fragment sizes for the compound pipeline. It applies the Ramp-up Theorem to gradually increase the fragment size to keep both the network and disk busy.

We vary the overhead of the disk stage from 0.5 ms to 32 ms, and its bandwidth from 1 MB/s to 20 MB/s. Figure 10 shows the improvement of hierarchical fragmentation (algorithm B) over the simple fixed-sized fragmentation (algorithm A) for 64KB packets. The performance results can be explained using the fixed-sized theorem and the tradeoff expressed in Equation (4). We

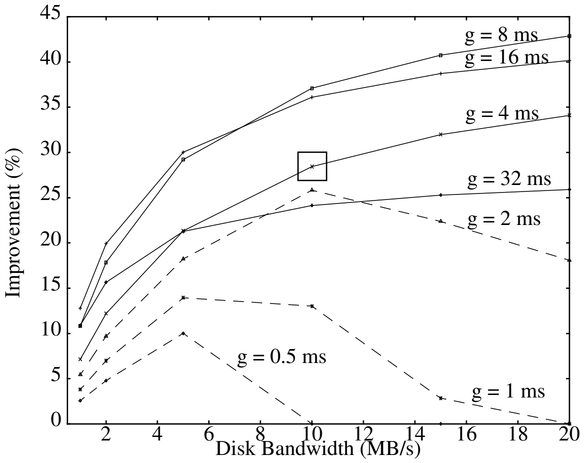

Figure 10: **Performance improvement of hierarchical fragmentation over fixed-sized fragmentation as a function of disk parameters for 64KB packets.** The point marked by the square represents the parameters of a typical modern disk.

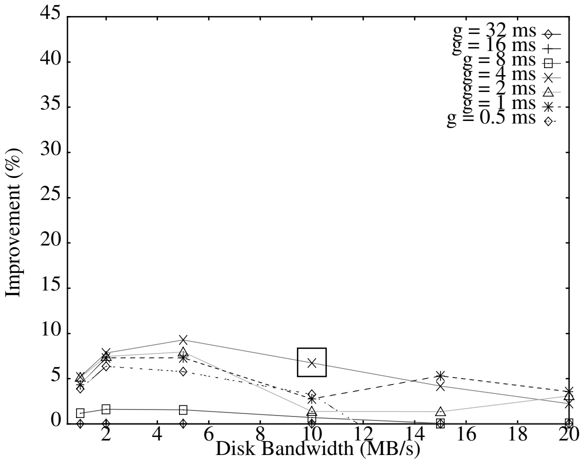

Figure 11: **Performance improvement of ramp-up fragmentation over fixed-sized fragmentation as a function of disk parameters for 64KB packets.** The point marked by the square represents the parameters of a typical modern disk.

observe that for both hierarchical and non-hierarchical fixed-sized fragmentation strategies, the fragment size does not depend on the bandwidth of the bottleneck stage. Consequently, as long as the disk remains the bottleneck, the improvement in packet latency in absolute terms for a given disk overhead is independent of the disk bandwidth. However, the relative improvement of the hierarchical strategy increases as the overall packet latency is lowered with better disk bandwidth. For most disks, the performance benefits reach an asymptote as the packet latency itself reaches a plateau. However, for disks with low overheads ($g \leq 2$ ms), the disk is not the bottleneck when operated at a high bandwidth and the performance benefits of hierarchical fragmentation taper off. Also, for a given disk bandwidth, the improvement in packet latency in absolute terms increases with higher disk overheads as hierarchical fragmentation minimizes the disk start-up costs. However, the relative performance improvement is lower for very large disk overheads ($g = 16$ ms, or $g = 32$ ms) as the packet latency itself becomes large. Overall, we see that hierarchical fragmentation provides excellent performance improvement when the overhead of the bottleneck stage is roughly an order of magnitude greater than the remaining stages and its bandwidth is no worse than an order of magnitude smaller than the other stages.

Figure 11 shows the result of repeating the above experiment for algorithm C, which adds the ramp-up algorithm to the hierarchical pipeline of algorithm B. We see that its additional improvement over fixed-sized hierarchical fragmentation is never greater than 10%.

In the next experiment, we keep the disk bandwidth constant (5 MB/s), and vary the user packet size (from 4KB to 4MB) and the disk overhead (from 0.5 ms to 32 ms). Figure 12 shows the improvement of algorithm B over algorithm A. Hierarchical fragmentation provides

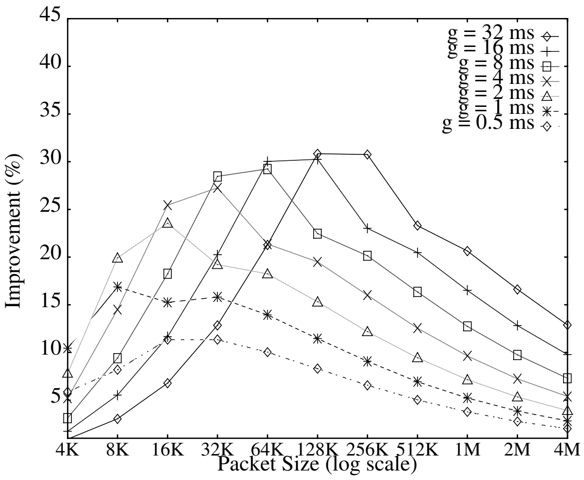

Figure 12: **Performance improvement of hierarchical fragmentation over fixed-sized fragmentation as a function of packet sizes.**

excellent performance improvement for "medium-sized" packets. As expected, the benefits of hierarchical fragmentation is less for small packets and for disks with low overhead. Also, when the packet size is very large, the packet approximates a continuous stream and a non-hierarchical strategy performs almost as well. Figure 13 shows the result of repeating the above experiment for algorithm C. The curve shows a similar trend but the additional improvement is again never greater than 11%.

Figure 14 shows further insight into the working of the variable-sized fragmentation algorithms by analyzing one data point from the above experiments. The packet size is 64KB. The disk stage has a 4 ms overhead and a 5 MB/s bandwidth. The first three bars model a pipeline that consists of a network stage and a

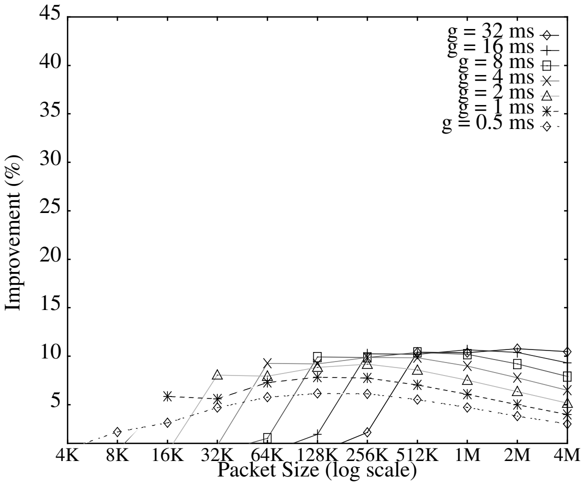

Figure 13: Performance improvement of ramp-up fragmentation over fixed-sized fragmentation as a function of packet sizes.

disk stage. The next three bars model a longer pipeline that consists of a network stage, a disk stage, and another identical network stage. The x-axis is labeled by the algorithms as defined above. In addition, we have introduced Algorithm C' (the ramp-up-and-down fragmentation), which starts and ends with small fragments but has larger fragments in the middle.

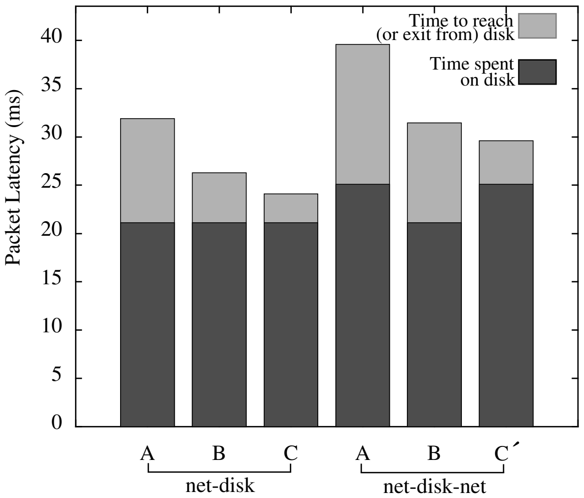

Figure 14: Performance improvements of hierarchical fragmentation and variable-sized fragmentation compared to fixed-sized fragmentation for 64KB packets.

We notice that the primary effect of the more complex algorithms is to decrease the time spent by the leading and trailing fragments in the network stage(s) ($T_f + T_l$). For the first pipeline, we see that the amount of time contributed by the disk is constant for the three different algorithms as the number of fragments used for the disk stage does not change. Algorithm B's hierarchical fragmentation reduces the time of the lead fragment in the network (T_f) by recursively fragmenting it resulting in a 21.3% overall reduction of latency. By gradually ramping up the fragment size, algorithm C uses a smaller lead fragment size, which further decreases T_f and reduces the overall latency by another 9.2%.

Algorithm A spends more time on the disk for the second pipeline as it uses more fragments to overlap the additional network stage. In other words, we must increase the time the packet spends on the disk, T_b, to keep $T_f + T_l$ low. This increase in the number of fragments, however, is unnecessary for algorithm B since the hierarchical fragmentation keeps the network time contributed by the leading and trailing fragments small. Overall, algorithm B out-performs algorithm A by 25.8%. Algorithm C' finds a more optimal solution as it derives more benefit from an increase in the number of fragments so it can have small leading and trailing fragments. However, the performance benefit of algorithm C' over algorithm B is only 6.4%.

From the experiments in this section, we conclude that the optimal algorithms are unlikely to obtain significant improvements over the simple combination of fixed-sized fragmentation and hierarchical fragmentation.

6 Related Work

Internet protocols have long used fragmentation to manage packet buffering, congestion control, and packet losses [15, 16]. Due to the high overheads of these protocols, it is generally better to use large packets to maximize bandwidth. However, most datagram networks impose a maximum fragment size. Higher level protocols (TCP/UDP) which are unaware of this limit may generate packets that cause extraneous fragments at IP level, which can significantly degrade performance due to high overheads and due to the fact that reassembly is not performed until the IP fragments reach the destination. Kent and Mogul discussed this problem in [11] and argued that the higher level protocol must make an effort to use packets whose size matches the minimum of the maximum fragment allowed on the route. This is a compromise for the legacy systems on the internet and is not optimal. However, their proposed changes to the internet architecture allow the application of our technique to the IP internet: the use of *transparent fragmentation* where each hop performs reassembly and the recording of path information in each packet allow high level protocols to intelligently choose fragment sizes.

In pathchar [7], Jacobson discusses a technique of measuring the latency and bandwidth characteristics of the individual hops on an internet path. By gradually increasing the "time-to-live" field of an IP packet, pathchar isolates the latency contributed by each additional hop and uncovers its characteristics. The inclusion of a diagnostic mechanism similar to the IP "time-to-live" field in a communication pipeline can allow the black-box measurement of detailed pipeline parameters.

GMS relies on simulation to find the optimal fragment size for sending an 8KB message through a pipeline that consists of an AN2 network and DEC Alpha workstations [8]. Using the GMS pipeline parameters derived from that work (Table 3), we were able to conclude that the optimal number of fragments is three or four, a result confirmed by the original GMS experiments.

stage i	g_i (μs)	G_i (μs/KB)
Srv-DMA	2.1	25.6
Wire	4.0	60.1
Req-DMA	2.1	25.6
Req-CPU	92.8	26.2

Table 3: GMS pipeline parameters. The bottleneck shifts from the "Req-CPU" stage to the "Wire" stage as the fragment size increases.

The Myrinet-BIP system [17] is the only other system that we are aware of that systematically adapts the fragment size to the user packet size. The model used in this system, while achieving a similar goal as that of the Fixed-sized Theorem, is more complex and its use requires the full knowledge of the pipeline parameters. The Fixed-sized Theorem only relies on as few as two parameters and they can be easily obtained from unintrusive observation of the network. An advantage of the BIP model is that it allows for adjustments to pipeline parameters for the different fragments.

7 Conclusion

In this paper, we present a number of fragmentation algorithms designed to minimize the latency of a message through a network of store-and-forward pipeline stages. The models provide insight into a crucial performance tradeoff that requires the careful balance of the different effects of the bottleneck stage and the remaining stages. We also present an experimental methodology that allows one to construct a customized pipeline algorithm that can adapt to the network characteristics and user packet sizes. By applying this methodology, we have not only achieved significant performance improvement on the Myrinet-GAM system, but we have also seen that a combination of fixed-sized fragmentation and hierarchical fragmentation can achieve performance results close to the theoretical optimum.

Acknowledgements

We would like to thank Susan Owicki and John Zahorjan for their suggestions that improved the proofs, Anna Karlin for the linear programming solution, Geoffrey Voelker and Michael Feeley for the GMS parameters, and Bernard Tourancheau for an interesting discussion on the Myrinet-BIP system, hiking in the French Alps, and local cuisine of Lyon. We would also like to thank Steve Lumetta and David Bacon for their comments on earlier drafts.

References

[1] Anderson, T., Culler, D., Patterson, D., and the NOW team. A Case for NOW (Networks of Workstations). *IEEE Micro* (Feb. 1995), 54–64.

[2] Anderson, T., Dahlin, M., Neefe, J., Patterson, D., Roselli, D., and Wang, R. Serverless Network File Systems. *ACM Transactions on Computer Systems 14*, 1 (Feb. 1996), 41–79.

[3] Boden, N., Cohen, D., Felderman, R., Kulawik, A., Seitz, C., Seizovic, J., and Su, W. Myrinet – A Gigabit-per-Second Local-Area Network. *IEEE MICRO* (Feb. 1995), 29–36.

[4] Chun, B., Mainwaring, A., and Culler, D. Virtual Network Transport Protocols for Myrinet. In *Proc. of 1997 Hot Interconnects V* (August 1997).

[5] Culler, D. E., Arpaci-Dusseau, A., Chun, B., Lumetta, S., Mainwaring, A., Martin, R., Yoshikawa, C., and Wong, F. Parallel Computing on the Berkeley NOW. In *9th Joint Symposium on Parallel Processing* (1997).

[6] Feeley, M. J., Morgan, W. E., Pighin, F. P., Karlin, A. R., Levy, H. M., and Thekkath, C. A. Implementing Global Memory Management in a Workstation Cluster. In *Proc. of the 15th ACM Symposium on Operating Systems Principles* (December 1995), pp. 201–212.

[7] Jacobson, V. pathchar – A Tool to Infer Characteristics of Internet Paths. http://www.msri.org/sched/empennage-/jacobson.html, 1997.

[8] Jamrozik, H. A., Feeley, M. J., Voelker, G. M., II, J. E., Karlin, A. R., Levy, H. M., and Vernon, M. K. Reducing Network Latency Using Subpages in a Global Memory Environment. In *Proceedings of the Sixth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-VII)* (Oct. 1996), pp. 258–267.

[9] Johnson, K. L., Kaashoek, M. F., and Wallach, D. A. CRL: High Performance All-Software Distributed Shared Memory. In *Proc. of the 15th ACM Symposium on Operating Systems Principles* (December 1995), pp. 213–228.

[10] Keleher, P., Cox, A. L., Dwarkadas, S., and Zwaenepoel, W. TreadMarks: Distributed Shared Memory on Standard Workstations and Operating Systems. In *Proc. of the 1994 Winter Usenix Conference* (January 1994), pp. 115–132.

[11] Kent, C. A., and Mogul, J. C. Fragmentation considered harmful. In *Proc. of Frontiers in Computer Communications Technology, ACM SIGCOMM* (August 1987).

[12] Lee, E. K., and Thekkath, C. E. Petal: Distributed Virtual Disks. In *Seventh International Conference on Architectural Support for Programming Languages and Operating Systems* (October 1996), pp. 84–92.

[13] Martin, R. P., Vahdat, A. M., Culler, D. E., and Anderson, T. E. Effects of Communication Latency, Overhead, and Bandwidth in a Cluster Architecture. In *Proceedings of the Twenty-Fourth International Symposium on Computer Architecture* (May 1997), pp. 85–97.

[14] Pakin, S., Lauria, M., and Chien, A. High Performance Messaging on Workstations: Illinois Fast Messages (FM) for Myrinet. In *Proc. of Supercomputing '95* (November 1995).

[15] Postel, J. Internet protocol. Request for Comments 791, Information Sciences Institute, Sept. 1981.

[16] Postel, J. Transmission control protocol. Request for Comments 793, Information Sciences Institute, Sept. 1981.

[17] PRYLLI, L., AND TOURANCHEAU, B. New protocol design for high performance networking. Tech. Rep. 97-22, LIP-ENS Lyon, 69364 Lyon, France, 1997.

[18] SCALES, D. J., AND LAM, M. S. The Design and Evaluation of a Shared Object System for Distributed Memory Machines. In *Proc. of the First Symposium on Operating Systems Design and Implementation* (November 1994), pp. 101–114.

[19] THEKKATH, C. A., MANN, T., AND LEE, E. K. Frangipani: A Scalable Distributed File System. In *Proceedings of the ACM Sixteenth Symposium on Operating Systems Principles* (Oct. 1997).

[20] VON EICKEN, T., BASU, A., BUCH, V., AND VOGELS, W. U-Net: A User-Level Network Interface for Parallel and Distributed Computing. In *Proc. of the 15th ACM Symposium on Operating Systems Principles* (December 1995), pp. 40–53.

[21] VON EICKEN, T., CULLER, D., GOLDSTEIN, S., AND SCHAUSER, K. E. Active Messages: A Mechanism for Integrated Communication and Computation. In *Proceedings of the Fifth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-V)* (May 1992), pp. 256–266.