Modeling Communication Pipeline Latency[*]

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 systems that employ dedicated processing elements for protocol processing. We develop an abstract pipeline model that reveals a crucial performance tradeoff involving the effects of the overhead of the bottleneck stage and the bandwidth of the remaining stages. We exploit this tradeoff to develop a suite of fragmentation algorithms designed to minimize message latency. We also provide an experimental methodology that enables the construction of customized pipeline algorithms that can adapt to the specific system characteristics and application workloads. By applying this methodology to the Myrinet-GAM system, we have improved its latency by up to 51%. Our theoretical framework is also applicable to pipelined systems beyond the context of high speed networks.

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 workload from the parallel applications which traditionally run on these networks. The first is the synchronous nature of the communication. While many parallel applications tend to communicate asynchronously to mask communication latency [9], file system operations, such as a read miss in the file system cache, stall the application until the entire operation is performed. 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 generating 4KB or 8KB *medium* messages, whose performance, as we will see, is governed by a model that is not well understood. These communication requirements are not only common to other high performance distributed file systems [14], but are also shared by distributed shared memory systems [7] and database applications.

While traditional communication layers such as TCP/IP have examined packet fragmentation for managing congestion, buffer overflow, and errors in the wide area context [11, 12], the high initiation overhead in these systems has made fine-grained fragmentation infeasible. Consequently, they have not systematically addressed the issue of developing an optimal fragmentation strategy for medium messages.

Recent developments in high performance local area network technology have necessitated revisiting the message fragmentation issues. Although the network fabric often supports cut-through routing, the processing elements in the network interface introduce store-and-forward delays. As network speed continues to increase, choosing the appropriate fragment size to exploit the inherent parallelism in this communication pipeline becomes a key issue. New communication software [16, 15] 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. However, these communication subsystems employ rigid policies for fragmentation. For example, the Active Message layer (AM2) [4] does not fragment medium messages, while Fast Messages (FM) [10] uses 128 byte fragments. In order to evaluate the soundness of these design choices, one needs a systematic approach. While previous research efforts such as [6] have employed simulation and empirical techniques for studying fragmentation, the lack of an analytical model prevents a generalization of their results to other systems.

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 the fragmentation strategy depends on the user packet size, and minimizing latency requires careful examination of the tradeoff between the effects of the overhead of the bottleneck stage and the bandwidth of the remaining stages.

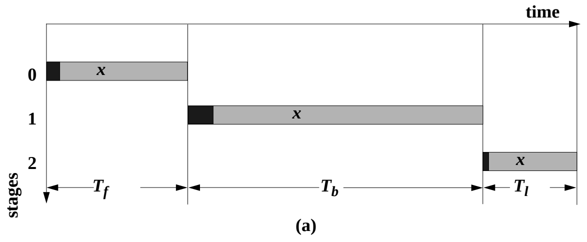
We provide a methodology that systematically uncovers the communication pipeline parameters and constructs customized pipeline algorithms. We present empirical studies comparing theory to practice. In one example, we show that the discrepancy between the model prediction and the implementation measurement on Myrinet-GAM [9] averages 5.9%. By applying the fixed-sized fragmentation to the system, we achieve a performance improvement of up to 51%. We also study the effectiveness of our algorithms for simulated systems with high overhead components such as disks.

The remainder of the paper is organized as follows. Section 2 defines the abstract problem and lays the foundation

[*]This work was supported in part by the Defense Advanced Research Projects Agency (N00600-93-C-2481, F30602-95-C-0014, F30602-95-C-0136), the National Science Foundation (CDA 9401156), Army Research Office (DAAH04-96-1-0079), U.S. Department of Energy (W-7405-ENG-48), 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



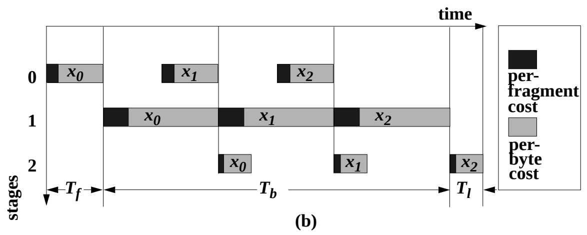


Figure 1: An example pipeline. Stage 1 is the bottleneck. The user packet is not fragmented in (a). When the same user packet has been fragmented into three pieces, it achieves a lower latency. Our goal is to minimize the latency, which is expressed as the sum of T_f, T_b, and T_l.

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

In this section, we formalize the pipeline model. The goal is to find the fragmentation strategy for a packet to minimize its latency. We represent the network as a sequence of store-and-forward pipeline stages characterized by the following parameters:

- n: the number of pipeline stages.
- g_i: the fixed per-fragment *overhead* for stage i.
- G_i: the per-byte cost (*inverse bandwidth*) for stage i.

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 is asymptotically limited by the stage with the minimum bandwidth. If the stages have no overhead costs, the lowest latency is achieved by maximizing parallelism using the finest possible fragmentation. Similarly, if the transfer costs are insignificant compared to the overhead costs, the best policy is to send the entire packet as a single fragment. For a general pipeline with both overhead and bandwidth limitations, we optimize for latency using the following variables:

- B: the size of the entire packet.
- k: the number of fragments.
- x_i: the size of the ith fragment. $\sum_{i=0}^{k-1} x_i = B$.
- $t_{i,j}$: the time the ith fragment spends in the jth stage. $t_{i,j} = g_j + x_i \cdot G_j$.
- $\tau_{i,j}$: the time when the ith fragment exits the jth stage. $\tau_{0,0} = 0$ by definition.
- T: the latency of the entire packet. $T = \tau_{k-1,n-1}$.

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 is longer than that shown in Figure 1(b) where the same packet is fragmented into three pieces. 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 also shows the constraints imposed by a store-and-forward pipeline. A fragment can enter the next stage only after it exits the current stage in its entirety. Also, a fragment cannot 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 integer 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 optimal fragmentation.

Such an 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 does not model refragmentation/reassembly. The linear system also requires complete knowledge of all pipeline parameters, which might not always be available.

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

Our fragmentation algorithms share a number of common themes. 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 (g_b, G_b). Figure 1 illustrates some of the properties of the bottleneck. First, the bottleneck (stage 1) in the figure is 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:

$$T = T_f + T_b + T_l \qquad (4)$$

where

- T_f is the time the *first fragment* takes to reach the bottleneck,
- T_b is the time the *entire packet* spends in the bottleneck, and
- 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 our models. The third observation is a weak lower bound on the total latency:

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

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 strategies 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 with the assumption that a particular stage 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; as we decrease the size of the fragments, we decrease the time the first fragment takes to reach the bottleneck (T_f) and the time the last fragment takes to leave the bottleneck (T_l). On the other hand, minimizing T_b requires large fragments, because the total overhead experienced at the bottleneck is lowered with fewer fragments. This is a fundamental tradeoff.

We now develop an optimality criteria to compute the optimal fragment size. 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 the time 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 time the last fragment spends in the stages after the bottleneck:

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

As there are k fragments with each fragment spending the same amount of time in the bottleneck,

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

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:

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

Thus the optimal number of fragments is:

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

The result in (6) follows as $x_i = B/k$. $\square$

As we observed earlier, 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 about this result. 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 an optimal latency 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 ineffective ($k = 1$ 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 that is *always* the slowest stage irrespective of fragment size. A sufficient condition is when a single stage has both the worst overhead and bandwidth. In this section we generalize our result to situations where a stage (g_b, G_b) is the bottleneck only for *some* range of fragment sizes (x).

Equation (10) can also be expressed as:

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

If 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

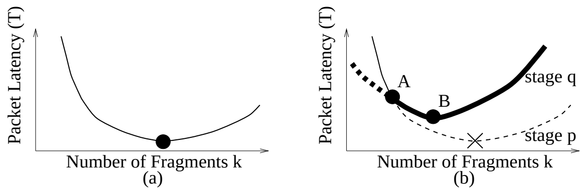


Figure 2: Pipeline latency curves. In (a), one stage remains the bottleneck for all fragment sizes. In (b), two different stages can be bottlenecks for different ranges of fragment sizes. The solid curve is the latency curve. The circles mark the candidate points for the global minimum. The cross marks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

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.

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) and larger fragments in the middle to minimize the overhead component of T_b. We study two-stage and three-stage pipelines 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 modeling a two-stage pipeline and its reversed counterpart (with stages arranged in the opposite order), we noticed that the optimal solution of one was obtained by reversing 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) *Let the sequence $(x_0, x_1, \ldots, x_{k-1})$ be 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 the reversed 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 pipeline with two stages, a necessary condition for 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.*

Proof. Informally, the theorem states that the first stage completes the transfer of a fragment exactly when the second stage completes processing 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 is achieved with matching transfer times for 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) since 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. Finally, as 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 fragmentation that contains bubbles in its first stage is also suboptimal. $\square$

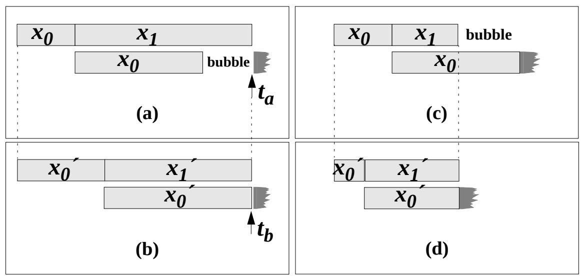


Figure 3: Bubble-free pipelines. A bubble in the second stage in (a) is eliminated by increasing x_0 to obtain (b). A bubble in the first stage in (c) is eliminated by decreasing x_0 to obtain (d). Note that according to our definition, the first stage of pipeline (c) is considered to contain a bubble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

4.1.3 "Ramp-up" Algorithm

Theorem 4 (Ramp-up Theorem) *The optimal fragment sizes for a two-stage pipeline follow the recurrence relation:*

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

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:

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

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

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 now be expressed as another function h of the same variables:

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

The initial fragment size x_0 can be obtained given k. □

According to Theorem 4, if the second stage is slower, 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 requires the fragment size to be 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 why ramp-up fragmentation improves on 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 with the same number of fragments k, as the fragments in the new algorithm successively increase in size, the initial fragment x_0 is smaller resulting in a smaller "lead" time T_f. On the other hand, if we compare a ramp-up fragmentation against a fixed-sized fragmentation with the same initial fragment x_0, again since the fragment sizes in the new algorithm increase, the new algorithm has fewer fragments resulting in the packet spending less time in the bottleneck (T_b). By decreasing T_f or T_b or both, the ramp-up algorithm performs better than 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. The fundamental characteristic of the optimal fragmentation strategy for such a pipeline is formalized in the following theorem and illustrated by Figure 4.

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, $0 \leq c < k$, *such that* $t_{i+1,0} = t_{i,1}$ *for* $0 \leq i < c$ *and* $t_{i+1,1} = t_{i,2}$ *for* $c \leq i < k$.

The theorem states that $(x_0, x_1, \ldots, x_c)$ monotonically increase so as to keep the first two stages busy, while $(x_c, x_{c+1}, \ldots, x_{k-1})$ monotonically decrease in order 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.

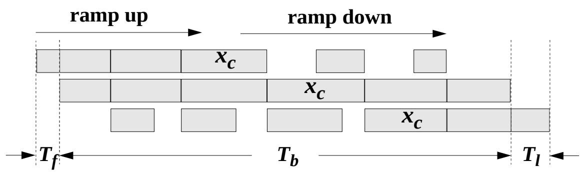


Figure 4: "Ramp-up-and-down" algorithm. Fragments gradually increase in size to keep the first two stages busy until x_c; they then decrease in size to keep the last two stages busy.

We can use 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 aggregate size of the first k_1 fragments is B_1 and the aggregate size of the remaining k_2 fragments is B_2:

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

Since $(x_0, x_1, \ldots, x_c)$ is a bubble-free solution for pipelining B_1 bytes through the first two stages, Equation (18) from 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". It is therefore related to x_0 by Equation (14):

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

We substitute (20), (21), and (22) into (19) to obtain an equation where the only free variable is x_0. We can solve for the unique x_0 to obtain a solution that satisfies 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 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 that the bottleneck stage does not change based on fragment size.

4.3 A "Fast-slow-slower" Pipeline

We will now 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 a recursive application of our algorithms 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 not optimal when the bottleneck stage has either very high overhead or very low bandwidth.

If the third stage has a high overhead, in order to 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 to transmit x_0 through these two stages (T_f), we may need to refragment x_0 and treat the first two stages as a sub-pipeline.

If the third stage has a very low bandwidth, 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 to assemble the largest possible x_1 for the bottleneck. We now see that reassembly may be necessary to accommodate the high overhead, low bandwidth, or both of the bottleneck stage.

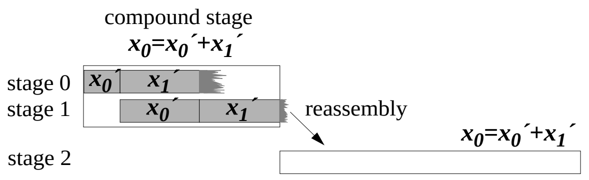


Figure 5: Hierarchical fragmentation. The first two stages of the three-stage pipeline are treated as a *compound stage*. Fragments exiting from the compound stage are reassembled into larger ones for the bottleneck stage.

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), the variable-sized fragmentation algorithms (of Section 4), or a mixture of these algorithms for the different levels of the hierarchy.

Unfortunately, the theory required for the precise modeling of this hierarchical approach becomes rather complicated. To simplify the model, we approximate a compound stage by a simple stage using linear regression analysis. Although this is not strictly accurate, empirical experience suggests 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 slight loss of free time towards the end of the (non-bottleneck) compound stage is not significant.

With these simplifications, modeling hierarchical fragmentation becomes tractable. 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.

4.3.2 Generalizations

The Reversibility Theorem of Section 4.1 allows us to generalize our fragmentation strategies. 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 to obtain a "slowest-slow-fast" configuration, a simple application of the Reversibility Theorem guarantees that the solution for the new pipeline is the reversal of that of the original one. In this case, instead of reassembly at 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.
- Ramp-up and Ramp-up-and-down fragmentation are optimal for two-stage and "fast-slow-fast" pipelines respectively.
- 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. After identifying the bottleneck stage, we apply the fragmentation algorithm recursively to construct approximations for the compound stages before and after the bottleneck stage. 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 several case studies. We present a systematic methodology for constructing pipeline algorithms that adapt to network charac-

[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 $5118\mu s$.

teristics and user packet sizes. We evaluate the accuracy and effectiveness of the models using both direct implementation and simulation.

5.1 Methodology

To construct a fragmentation algorithm for a specific pipeline, we first need sufficient details of the pipeline. 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 effect of the individual stages by non-intrusively observing how the system responds to different workloads. The pipeline algorithms adapt the fragmentation at run time to different packet sizes. We can also monitor changes in the pipeline parameters, due to factors such as contention, to fine-tune the fragmentation at run time.

5.2 Experimental Platform

Our main experimental platform is the Berkeley NOW [1], which is a cluster of UltraSPARC Model 170 workstations running Solaris 2.6. Each workstation is equipped with a Myricom M2F network interface card [3] on the SBUS. Figure 6 shows the details 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 LANai processor, which executes the messaging protocol and is responsible for both coordinating the actions of the DMA engines and interfacing with the host. These processing elements are the source of the parallelism that we exploit in the pipelining algorithms. The workstations are connected using Myricom M2F switches with link bandwidths of 160 MB/s.

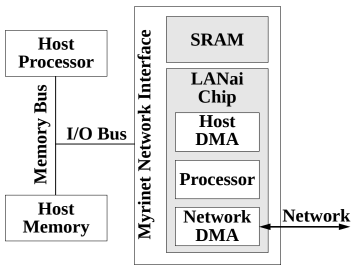


Figure 6: Myrinet network interface. To send a medium message, the host DMA moves the data from host memory into the on-board SRAM, and the network DMA moves the data from the SRAM into the network.

The base communication system that we use in the study is Generic Active Messages (GAM) [9], a version of Active Messages [15]. One notable feature of this communication layer is its low overhead, which enables fine-grained fragmentation. In order to experiment with system parameters that are different from that of the Berkeley NOW, we supplement our study with results from a pipeline simulator.

5.3 GAM Pipeline Parameters

The first step towards constructing a customized algorithm is determining the pipeline parameters. In this section, we present the direct instrumentation approach, which provides complete information using source code modifications, and the non-intrusive indirect observation, which provides just enough information for fixed-sized fragmentation.

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 simple linear regression.

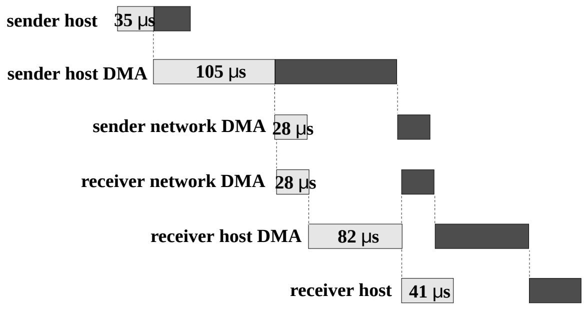


Figure 7: Timing measurements of two 4KB fragments flowing through a Myrinet GAM pipeline.

Before we use these measurements to develop an abstract model, we analyze our platform by studying the flow of two 4KB fragments sent back-to-back through the network (Figure 7). The steps labeled "sender host" and "receiver host" correspond to data copying on the two hosts. The "sender host DMA" step, 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 are store-and-forward. The "receiver host DMA" step is faster than "sender host DMA" due to 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; 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 communication layers, which is not always feasible. In this section, we demonstrate how to indirectly deduce the pipeline parameters by observing the performance of the system at the application level.

In the Fixed-sized Theorem of Section 3.1, we had shown that we need only two parameters to construct the desired algorithm: 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 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 whose parameters are ($\sum g_i$, $\sum G_i$). By performing a linear regression on the packet sizes and the corresponding latencies, we obtain: ($\sum g_i$, $\sum G_i$). In the second step, we send packets into the network as quickly as possible and measure the frequency at which they exit from the pipeline. The inter-arrival time is the time a packet spends in the bottleneck stage. By varying the packet size 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 observation 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 instrument the pipeline to find the bottleneck parameters for a fragment size that is the lower bound on the packet size. We then repeat the process for the largest 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 halves 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 modeled and measured values of fixed size fragmentation for 4KB packets. The model prediction has an error that averages 5.9%. The model predicts that the best latency is achieved when the number of fragments is five, a result confirmed by the implementation. Also shown are the results of an exhaustive search for an optimal fragmentation using the linear equations presented in Section 2. Fixed-sized fragmentation performs only 9% worse than the fragmentation obtained by exhaustive search.

Figure 8 also illustrates the tradeoff from Equation (4). As the number of fragments increases, the time the packet spends in the bottleneck increases due to the overhead introduced by the additional fragments. However, as the fragment size decreases, it takes less time for the first fragment to reach the bottleneck stage and for the last fragment to drain from the pipeline.

We compare our fixed-sized fragmentation algorithm with the original GAM implementation and FM [10], both of which use a static fragmentation strategy (128B for FM and 4KB for the original GAM). Figure 9 shows the comparison. FM suffers from the high overhead incurred by the small fragments for large packets. The fixed-sized fragmentation achieves the largest performance gain of 51% over GAM for a packet size of 4KB. The performance gains diminish for large packets as the bandwidth of the bottleneck stage becomes the dominant limiting factor. The improvement is also small for smaller packets due to the inherent overhead in the pipeline, which prevents very fine-grained fragmentation.

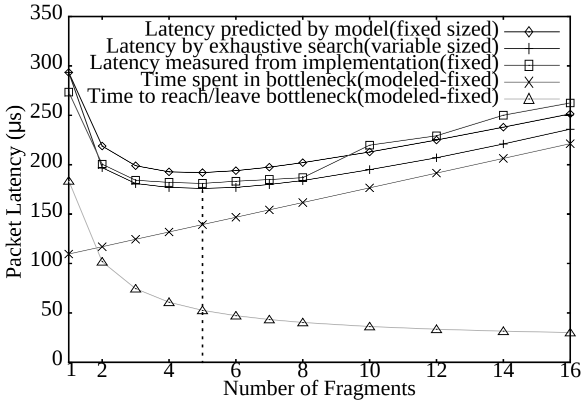


Figure 8: Fixed-sized fragmentation for 4KB packets.

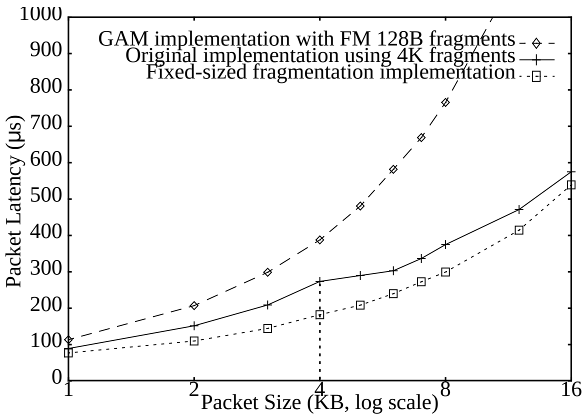


Figure 9: Latency comparison of fixed-sized fragmentation versus static fragmentation.

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 Myrinet-GAM), 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 with a disk attached to a conventional network. The network speed is one fifth of that of Myrinet-GAM; it has five times its overhead and one fifth of its bandwidth. These parameters are comparable to that of conventional systems such as TCP running on a 100 Mb/s ethernet. To understand the impact of the *relative* speed of the network to the rest of the system, we vary the disk's overhead and bandwidth characteristics.

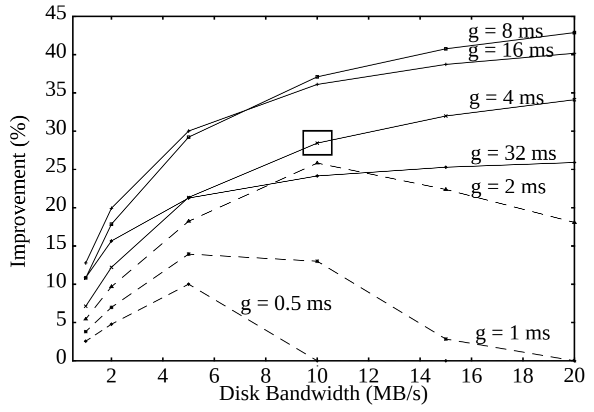


Figure 10: Performance improvement of hierarchical fragmentation over simple fixed-sized fragmentation as a function of disk parameters for 64KB packets. The square represents the parameters of a typical modern disk.

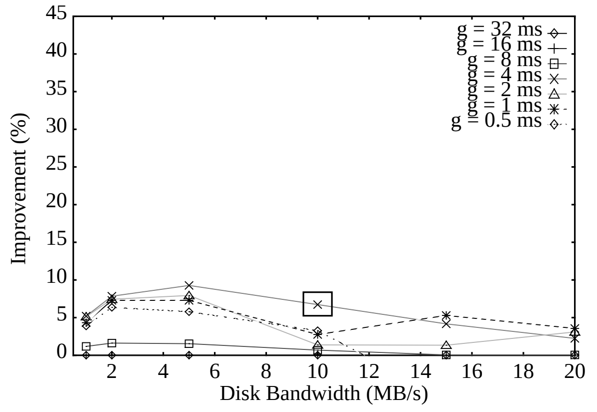


Figure 11: Performance improvement of ramp-up fragmentation as a function of disk parameters for 64KB packets. The square represents the parameters of a typical modern disk.

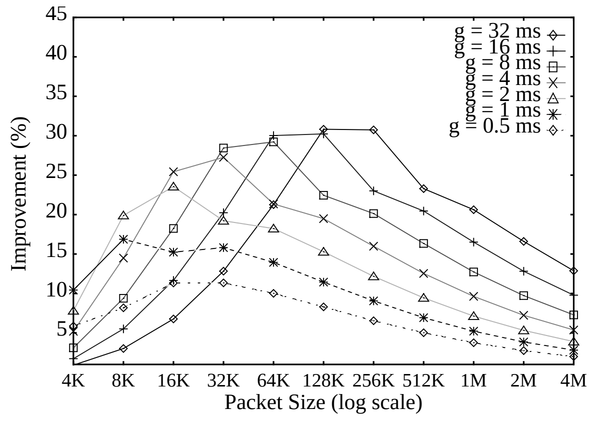


Figure 12: Performance improvement of hierarchical fragmentation over fixed-sized fragmentation.

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 a hierarchy and applies 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 gradually ramps-up the fragment size to keep both the network and disk busy.

We vary the overhead of the disk stage (g) from 0.5 ms to 32 ms and its bandwidth ($1/G$) from 1 MB/s to 20 MB/s. Figure 10 shows the improvement of hierarchical fragmentation (algorithm B) over fixed-sized fragmentation (algorithm A) for 64KB packets. The performance results can be explained using the fixed-sized theorem and the trade-off expressed in Equation (4). Note that for both hierarchical and simple 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, hierarchical fragmentation provides performance benefits when the overhead of the bottleneck 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 results from repeating the above experiment for algorithm C, which uses the ramp-up algorithm on the hierarchical pipeline. We see that its improvement over algorithm B 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 excellent performance 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 results from repeating the same experiment for algorithm C. The curve shows a similar trend but the additional improvement is again never greater than 11%.

Figure 14 provides further insight into the working of variable-sized fragmentation 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 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.

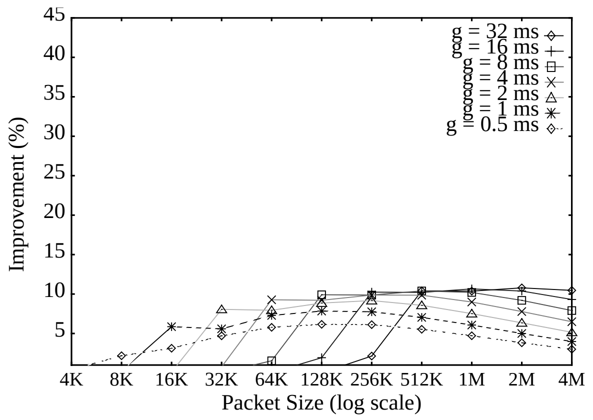


Figure 13: Performance improvement of ramp-up fragmentation as a function of packet sizes.

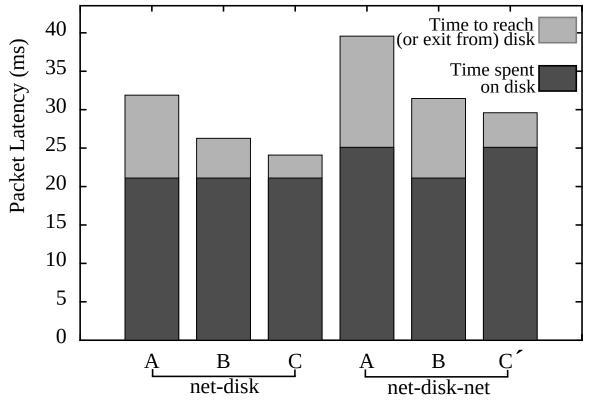


Figure 14: Performance of hierarchical fragmentation and variable-sized fragmentation compared with fixed-sized fragmentation for 64KB packets.

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%.

For the second pipeline, algorithm A increases the time spent on the disk as more fragments are used to overlap the additional network stage. In other words, to keep $T_f + T_l$ low, it increases the number of fragments resulting in a higher T_b. This increase, however, is unnecessary for algorithm B since 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', which is provably optimal for 3-stage pipelines, finds the optimal solution. 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 [11, 12]. 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 both high overheads and a delayed reassembly (which happens only when the IP fragments reach the destination). Kent and Mogul discussed this problem in [8] and argued that the higher level protocol must use packets whose size matches the minimum of the maximum fragment allowed on the route. This strategy is a compromise to suit legacy systems 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 [5], Jacobson discusses a technique for measuring the latency and bandwidth 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. The inclusion of a diagnostic mechanism similar to the IP "time-to-live" field in a communication pipeline can facilitate 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 [6]. Using the GMS pipeline parameters derived from that work (Table 3), we were able to confirm its optimal fragment size by applying the Fixed-sized Theorem. In Trapeze, the network interface implements pipelining in a manner that is transparent to the host [17]. Although successful for the specific packet size and configurations studied, these systems do not provide a general solution.

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 [13] is the only other system that we are aware of that systematically adapts the fragment size to the user packet size. This system uses fixed-sized fragmentation, but its pipeline model is more complex and requires exhaustive information about the pipeline. Our Fixed-sized Theorem only relies on as few as two parameters, which can be easily obtained from non-intrusive observation of the network. An advantage of the BIP model is that it allows the transmission characteristics to vary based on the relative position of a fragment within a packet.

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 effects of the bottleneck's overhead costs and the bandwidth of the other stages. We also present an experimental methodology that facilitates the construction of a customized pipeline algorithm, which can adapt to the network characteristics and user packet sizes. By applying this methodology, we have not only achieved significant performance benefits 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] Jacobson, V. pathchar – A Tool to Infer Characteristics of Internet Paths. http://www.msri.org/sched/empennage-/jacobson.html, 1997.

[6] 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.

[7] 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.

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

[9] 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.

[10] 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).

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

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

[13] Prylli, L., and Tourancheau, B. New protocol design for high performance networking. Tech. Rep. 97-22, LIP-ENS Lyon, 69364 Lyon, France, 1997.

[14] 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).

[15] 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.

[16] 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.

[17] Yocum, K. G., Chase, J. S., Gallatin, A. J., and Lebeck, A. R. Cut-through delivery in trapeze: An exercise in low-latency messaging. In *Proc. of the Sixth IEEE International Symposium on High Performance Distributed Computing* (August 1997).