Modelling and Improving Group Communication in Server Operating Systems

Michael Kwok, Tim Brecht, Martin Karsten, Jialin Song
David R. Cheriton School of Computer Science, University of Waterloo
Waterloo, Ontario, Canada
email: {kfkwok,brecht,mkarsten,j8song}@cs.uwaterloo.ca

Abstract

In recent years, applications that provide a distributed virtual environment (DVE) have become increasingly popular. Many DVE implementations use a client-server architecture that requires the server to send the same data to all members of a collaborating or interacting group. This type of group communication operation is often implemented by sending data from the server to each recipient in a unicast fashion. The problem with this approach is that the cost of communication at the server does not scale very well with the number of participants because the application requires significant interaction with the operating system, network stack and drivers for each individual send. In this paper, we first propose a general analytic framework for predicting how group communication performance impacts DVE server capacity. We then conduct an experimental evaluation to determine the extent to which using a kernel-based group communication mechanism reduces the cost of group send operations. Lastly, we use the measurements obtained from these experiments to demonstrate how to apply the analytic framework by determining the extent to which the kernel-based group communication mechanism permits example applications to scale to more users.

1 Introduction

A distributed virtual environment (DVE) is a shared virtual environment where multiple human users interact with each other. Such an environment is often considered as an interactive, immersive, and multi-sensory environment. In recent years, DVEs have rapidly gained popularity among Internet users. This is substantiated by the increased use of DVEs such as multi-player on-line games (MOGs) [1, 3], and computer supported collaborative work (CSCW) applications [8]. In fact, even relatively conventional systems such as audio and video conferencing and chat room applications can be classified as DVEs. The key attribute of these applications is that they all support a number of participants simultaneously interacting with each other using a network such as the Internet. In many cases, they may involve a large user population that is geographically distributed.

Typically, DVE applications are implemented using a client-server architecture. They use UDP to transmit voice, video, or other real-time data. Since these applications are highly interactive, such as conferencing or on-line gaming, they usually transmit data in small messages to achieve a low packetization delay. They do not necessarily consume a large amount of transmission bandwidth and often, the servers are the system bottlenecks, rather than the network.

Perhaps the most important requirement of a DVE system is to provide the participants with a consistent view of the virtual environment in a timely manner. Any changes in a user's state (e.g., position and orientation) must be distributed to affected participants. In order to provide good service, the server must be able to communicate to all or a group of participants as quickly as possible [10]. Therefore, of prime importance is the efficiency and scalability of the communication mechanism. In the absence of IP-level multicast, the only solution is a unicast-based distribution mechanism. With or without the help of any application-layer multicast (ALM) infrastructure, this requires repeated send operations of the same data to multiple recipients.

In previous preliminary work [13] we have introduced the notion of *kernel-level group unicast*, or *kernel-groupcast* for short, to support efficient group communication using UDP. UDP is used because of its widespread adoption for DVE systems and other group communication applications in the Internet. Initial experimental results show that this technique significantly improves the performance of group send operations, when compared with the traditional approach that requires one send system call and one data copy for each member of the group.

The focus of this paper is to examine how efficient group communication mechanisms can be used to increase the capacity of DVE servers. By capacity, we mean the maximum number of users that a server can support while maintaining an acceptable level of performance. Our contributions are:

- We develop an analytic framework to examine process-

ing and communication requirements in two different types of DVE servers in order to predict the size of the user population that can be supported by different communication mechanisms.

- We illustrate how to apply our analytic framework by demonstrating the potential benefits that could be obtained by using our kernel-groupcast mechanism in a DVE server.
- We derive bounds on the capacity of the application server, that can be obtained through improved group communication mechanisms.

The remainder of this paper is organized as follows. Section 2 describes previous work related to our study. Section 3 introduces an abstraction of DVE server models, and Section 4 presents an analytic framework for assessing the execution efficiency of different group communication mechanisms. Then, Section 5 provides an overview of our kernel-based group unicast mechanism and presents some measurement results. Using these results as inputs, Section 6 demonstrates how our analytic framework can be applied to examine the efficiency of our kernel-groupcast technique at a DVE server. We also derive a general bound for the potential benefits of using any improved groupcast techniques. Finally, Section 7 contains concluding remarks and some directions for future work.

2 Related Work

A common communication pattern required by a DVE server is to send the same data to a group of users. Information delivery systems and audio and video conferencing applications also have similar requirements. Traditionally, group communication in DVEs has been implemented simply by performing repeated unicast sends to all recipients. With an increasing number of users, the use of network-layer multicast has been proposed for many DVE systems [10, 14]. Multicast solves part of the scalability problem by allowing the users or the server to send data only once for the entire group rather than having to repeatedly transfer the data to all the recipients using a unicast send.

IP multicast, as originally proposed by Deering [9], has long been regarded as the right mechanism for multicast. However, its deployment is still very limited due to a variety of technical and non-technical reasons [11]. For instance, IP multicast requires intermediate routers to be upgraded and to support additional tasks such as maintaining group state and routing information. As a result, there have been several proposals for implementing multicast in the application layer rather than the network layer. This approach is known as application layer multicast (ALM). Much research has focused on the construction and maintenance of efficient ALM networks. However, little attention has been paid to analyzing and improving the efficiency of operating system mechanisms for the required underlying group unicast send operations. Our kernel-groupcast mechanism proposed in [13] is designed to reduce communication costs when one host is communicating directly with several hosts (recipients). As a result, this mechanism could be used to reduce the cost of sending data to downstream nodes in ALM networks. Likewise, kernel-groupcast could be used to reduce communication costs in general group communication systems, such as Horus [16] and Spread [7].

Existing work on the performance analysis of DVE systems has been conducted mainly by means of prototyping and measurement. For example, *MiMaze* is a prototype of an MOG employing multicast and has been evaluated with 25 players located in various places in France [10]. On the other hand, the performance of *QuakeWorld* has been measured in terms of throughput, network bandwidth requirements, and a breakdown of the processing time [5]. In a follow-up to this work, the authors have attempted to improve server performance by parallelizing the game server [4]. To our knowledge, these papers are the only source of raw performance data that sheds some light on the costs of network stack processing related to sending data to groups of recipients in a DVE application. However, when comparing the numbers reported in [5] and [4], it is unclear how to interpret the data in each paper. For example, while [5] reports a relative send overhead of 50%, [4] uses a different metric and reports a 5% total overhead for all network-related system calls. The actual fraction of time a server spends performing group send operations depends on several factors which include the server hardware architecture, operating system, NICs, drivers, network environment, application implementation and nature of the required group communication. As a consequence, our work in this paper treats the fraction of time a server spends performing send operations as an input parameter to a model that can be used to determine how many users can be supported and how that number of users could be increased if group communication efficiency would be improved.

In contrast to the above work, our work considers the impact that a group communication mechanism has on overall server performance and develops a model to support explicit performance predictions. There is only limited work on analytic modelling of DVE systems. Muller *et al.* [15] develop an analytic model called a "game scalability model" (GSM) to evaluate scalability of *Rokkatan*, an on-line real-time strategy game. This model divides the processing at the game server into different tasks, e.g., updating the database and sending state updates to the server's local users. For each of these tasks, GSM characterizes the average processing time required, the average amount of data received, and the average amount of data sent. Summing each measure over these tasks and comparing the sums to pre-defined

maximum values, GSM predicts the maximum number of players that can be supported. However, mathematical definitions and detailed characterizations of the above measures are not provided. Also, GSM does not take into account different server execution models. Our model not only addresses these issues, but also provides insight into DVE server performance under a wide range of scenarios.

Other work on analytic modelling of a DVE server can be found in [17]. In this work, the server performance is analyzed based on resource utilization rather than processing time. Their server model does not distinguish different operations performed by the server; therefore, it cannot be applied to investigate the impact of a group communication mechanism on the overall server performance.

Because the time required to send data to all recipients constitutes one portion of the overall execution cost of a DVE server, Amdahl's law [6] could be applied to predict the performance improvement as a result of decreasing the time required to send data. While Amdahl's law provides a general formulation of the resulting speedup, we use a more detailed model that allows us to capture and analyze more specific characteristics of different DVE server models.

In previous work [13] we have described the core idea behind kernel-groupcast, described the system call interface and provided a high-level overview of the implementation. Our work here differs significantly in that the focus of this paper is to produce a detailed analytic model to interpret new micro-benchmark results on modern server architectures in the context of application performance.

3 Server Models

In a DVE any changes in a user's state (e.g., position, velocity, orientation) must be distributed to the workstations of those users who are affected by these changes in real time. The transmission of a state update is done using group communication, by mapping affected users into a group and sending the state update to all users within the group [18].

Many DVE systems are implemented using a client-server architecture. With this architecture, all state updates are first sent to a central server where they are processed and sent to all affected users. This architecture is widely deployed because of its ease of implementation. The server maintains an authoritative copy of the virtual environment, which makes it easy to guarantee consistency among all users and prevent cheating. We therefore focus on the client-server architecture in this paper.

A DVE server can be implemented using one of two execution models. In one model updates are sent synchronously immediately after client messages arrive. In the other model updates are sent asynchronously, according to some time interval which is related to the latency that clients can tolerate. To avoid using the overloaded terms "synchronous" and "asynchronous", we call these two models the *immediate* and the *periodic* send models, respectively.

In the immediate send model, the server receives an update from a client, processes the update, and immediately sends the resulting change to all affected users. On the other hand, in the periodic send model, the server processes all incoming updates immediately, but only sends updates to the affected users periodically. Pseudo-code for both alternatives is shown in Figures 1 and 2, respectively.

```
while (1) {
    receive_client_msg()
    process_client_msg()
    send_update_to_group()
}
```

Figure 1. DVE server - immediate send model

```
set_alarm_handler(send_update_to_group,
     period);
while (1) {
     receive_client_msg()
     process_client_msg()
}
```

Figure 2. DVE server - periodic send model

The immediate send model is simple and often yields the shortest delay in terms of receiving the most up-to-date changes of the virtual environment. However, it may result in more send operations than the periodic send model. There are several scenarios where the immediate send model is the appropriate choice. For example, it may be used to model the server side of CSCW applications. Also, multi-user dungeon (MUD) games, which are text-based MOGs, operate on a coarser time scale, but still provide real-time interaction. One example of MUD games that use the immediate send model is RockyMud [2].

In environments where the server sends real-time audio or video streams to the clients, the frame rate of the audio or video stream typically determines the desired frequency of updates from the server to the clients. Generally, such applications are best described by the periodic send model. Updates at a higher frequency will not improve the user experience and are thus unnecessary. However, if the frame rate frequency cannot be sustained, the perceived quality will be degraded. The periodic send model also applies to streaming audio and video servers where multiple audio or video streams are being sent by the server to multiple clients.

The same design choice between performing immediate and periodic sends also exists for the client program. Nonetheless, the work presented in this paper is independent of the detailed send behavior of the clients.

4 Analytic Framework

In this section, we present an analytic framework which allows us to describe the potential benefits that can be obtained by reducing the time spent in performing group send operations at a DVE server.

In this framework, we consider the two server models introduced in Section 3. For simplicity, we assume all users are in the same group. We also assume that every update needs to be distributed to all the users. This in fact corresponds to the worst case scenario, and such a scenario is of interest when one consider the issue of performance.

Of interest is the `send_update_to_group()` (or *send* for short) operation, and the reduction in the time spent performing this operation. For the `receive_client_msg()` and `process_client_msg()` operations, we simply combine them into a single component (referred to as *other*). Let $\mathcal{S}$ and $\mathcal{H}$ be the average processing times for the *send* and the *other* operations, respectively.

The performance constraint of the DVE server is modelled as a maximum average period between `send_update_to_group()` operations. In the case of the immediate send model, T denotes the maximum average delay from when an update is received to when the corresponding updates are sent to the entire group. In the case of the periodic send model, T is simply the pre-defined period used to determine the frequency of *send* operations.

Given the above performance constraint, we now present our framework for each server model. For the immediate send model, since each incoming message results in an update being sent immediately to the group, we have:

$$\lambda N(\mathcal{H} + \mathcal{S}) \leq T \tag{1}$$

where λ denotes the average number of messages sent from a user per time period T, and N denotes the number of users (in other words, the size of the group). For the periodic send model, updates are generated at the end of the time period T; we have:

$$\lambda N\mathcal{H} + \mathcal{S} \leq T. \tag{2}$$

Note that the basic formulation of Amdahl's law [6] can be directly applied only if the fraction of send times relative to the total processing time is constant for varying numbers of users. This is not always the case with the different server models and the different models of $\mathcal{H}$ considered in this paper (e.g., in the case of an immediate send model with a function of $\mathcal{H}$ that is constant). Therefore, rather than using Amdahl's law, we design a new framework to permit the analysis of and comparison between the two server models in more general scenarios.

Consider now the characterization of $\mathcal{S}$ and $\mathcal{H}$. Typically, in a DVE server, the average time for a *send* operation $\mathcal{S}$ may depend on the number of users involved N, and the size of the update data packets being sent, b. $\mathcal{S}$ can be written as:

$$\mathcal{S} := S(N, b). \tag{3}$$

For the average time required per *other* operation $\mathcal{H}$, our framework provides support for the possibility that $\mathcal{H}$ may also depend on N. Specifically, $\mathcal{H}$ can be written as:

$$\mathcal{H} := H(N). \tag{4}$$

In order to examine the increase in the number of users that can be supported by improving the efficiency of the group communication mechanism, we introduce two performance measures, namely the speedup factor r and the relative capacity improvement ratio k. Suppose we have an original groupcast technique and a new groupcast technique (which is supposed to reduce the "send" time). Let $S_{orig}(N, b)$ and $S_{new}(N, b)$ be the amount of time taken to send packets of size b bytes to N users by the original and the new groupcast techniques, respectively.

The speedup factor r, which is a measure of efficiency of the new technique with respect to the original technique, is defined as:

$$r = \frac{S_{orig}(N, b)}{S_{new}(N, b)}. \tag{5}$$

Values of r greater than one indicate that the new groupcast technique is more efficient, and larger values indicate larger improvements.

The relative capacity improvement ratio k is used to evaluate the potential benefit an application may gain as a result of using the new groupcast technique. Given the characterizations of $\mathcal{H}$ and $\mathcal{S}$, and knowing the send model being used (Equation 1 or 2), we can obtain the maximum number of users the server could support with the original and new groupcast techniques (denoted by N_{orig}^{max} and N_{new}^{max}, respectively). Then, k is defined as:

$$k = \frac{N_{new}^{max}}{N_{orig}^{max}}. \tag{6}$$

Generally speaking, this analytic framework applies to any group communication mechanism. It not only allows us to compute the number of users that can be supported by a particular group communication mechanism at a DVE server, but it also provides a methodology for comparing the efficiency of two different mechanisms and their impact on the number of users the server can support. For illustration, we apply the framework to our proposed groupcast technique, namely kernel-groupcast [13]. In the next section, we will first describe our kernel-groupcast technique and present measurement results which in Section 6 will be used as inputs to the framework.

5 Kernel-Level Group Unicast

In this section, we first briefly describe the design of our kernel-level group unicast implementation from [13]. We then describe the results obtained by running some micro-benchmarks on two different hardware platforms, comparing the performance of user-level group unicast (referred to as *user-groupcast*) with the kernel-level group unicast (referred to as *kernel-groupcast*). These results are used in Section 6 to populate mathematical models for the analysis of application performance.

5.1 General Design

The goal of our operating system extension is to enhance the functionality of sockets and the `send` system call to facilitate sending to a group of recipients with a single system call. To use kernel-groupcast, an application first creates a socket that will be used to refer to a group. A group of recipients is associated with the socket by calling `setsockopt` with the file descriptor of the socket, the `SETSENDGRP` parameter, and an array containing the addresses and ports (`struct sockaddr_in`) used to reach each member of the group. To send the same data to all members of the group, the application simply uses the `send` system call with the file descriptor associated with the group. The kernel then sends the data to each member of the group. In previous work [13] we have shown that the system-call cost for completely changing the entire group between each send operation is negligible, and we have provided a more detailed description of the API and an example use case.

Figure 3 gives an overview of the components of an operating system kernel relevant to sending UDP data. In general, carrying out a send operation consists of the following steps. After initial processing in the socket layer, the payload data is copied into a packet buffer. The UDP layer, in cooperation with the socket layer, prepends the UDP header and retrieves IP addressing information, which is then used in the IP layer to produce the complete IP packet. The packet buffer is submitted for link layer processing and eventually DMA-transferred to the NIC. The basic functionality of kernel-groupcast is located in the UDP processing component. A list of recipient endpoints (address & port) is stored with the socket data structure. If this group address list is not empty, instead of just creating a single IP packet, the UDP output function loops through the list of addresses and creates multiple instances of the packet buffer. Multiple instances of the packet buffer are necessary, since lower level processing is not invoked synchronously, but rather, packet buffers may subsequently be queued in the system. Because different processing steps throughout the network stack are executed asynchronously, naively working with a single copy of the packet buffer would result in overwriting header data, while an outgoing packet buffer is still queued for service. Ideally, packet buffer instances can be formed without copying the payload part of the packet buffer, since only header fields change between successive loop iterations. If the operating system supports appropriate packet buffers, multiple packets can be formed as multiple logical instances of the same packet, each instance comprised of private protocol headers and a shared payload. However, this depends on the detailed design of the packet buffer data structure and other considerations, such as the fragmentation strategy.

We have implemented kernel-groupcast on both FreeBSD and Linux. The FreeBSD version is discussed in [13]. It turns out that these two implementations require different approaches to avoid copying the payload, resulting from different design philosophies for the networking stack. In particular, the Linux networking stack does not separate packet header and packet data for larger payloads, as is the case in BSD-based systems. Therefore, it is not possible to completely avoid copying the payload when creating multiple packets to be sent to each recipient. We have designed and implemented an on-demand *lazy-copy* scheme to avoid making copies until absolutely necessary. Essentially, after a packet is processed by the NIC, its allocated data structure is appended to a shadow queue and available for reuse. This is illustrated in Figure 3.

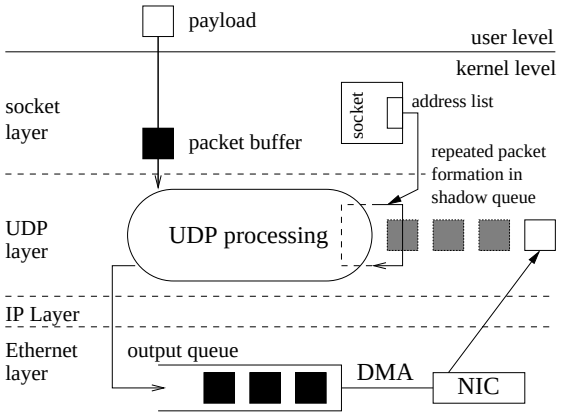

Figure 3. Lazy-copy packet replication

5.2 Experimental Results

We report the results of a number of micro-benchmark experiments, which are designed to help understand the performance improvements of kernel-groupcast over user-groupcast. Specifically, we measure the time required by a single server to send data to a number of recipients. The results are obtained using version 2.6.8 of the Linux kernel. The corresponding FreeBSD results are fundamentally similar [13]. The two hardware platforms under study are

a 2.8 GHz Intel Xeon server (referred to as *i386*) and a 900 MHz Intel Itanium2 server (referred to as *ia64*). Each server uses up to four, 1 Gbps Ethernet interfaces to send out data with its single CPU being the bottleneck. In fact, the experiments revealed that because of the efficiency of our kernel-groupcast implementation, with increasing packet sizes we were able to generate very nearly 4 Gbps of network traffic. We therefore limited the packet sizes to ensure that the CPU is still the bottleneck.

The primary performance metric for our experiments is the average time taken by the sender to send a single message to all members in a group of recipients (average send time). The graphs in this section show the average send time while varying the group size for a set of packet sizes. A reduction in the average send time indicates a corresponding increase in performance with a lower slope indicating better scaling. All experiments consist of at least 100 send requests and are run 10 times each. We then compute 99.9% confidence intervals using the t-distribution. However, the experiments are fairly deterministic, so the confidence intervals turn out to be extremely small and would be barely visible in the graphs. Therefore, we simply omit them.

Figure 4 illustrates the performance gains of kernel-groupcast over user-groupcast on the Linux/i386 server. Kernel-groupcast executes much faster than user-groupcast and as expected the performance gap widens for larger group sizes. Somewhat surprisingly, the packet size does not significantly affect group unicast cost, especially for the kernel-groupcast performance.

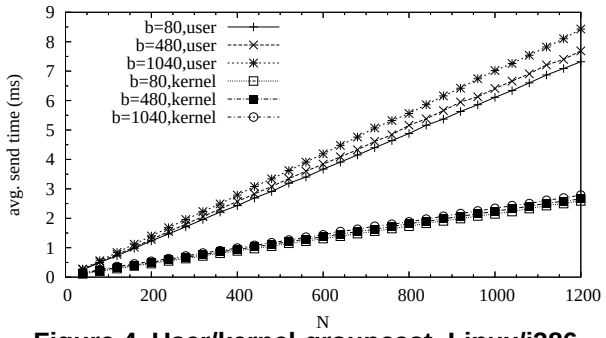

Figure 4. User/kernel-groupcast, Linux/i386

Figure 5 shows the same results obtained on the 64-bit Intel Itanium2 processor running Linux. On this platform user groupcast executes faster than on the i386 architecture (despite the discrepancy in processor speeds), while kernel-groupcast is slower than on the i386. Nonetheless, our results still demonstrate the significant performance advantages offered by using kernel-groupcast. We next examine how these improvements might be used by servers to increase the number of users it can support.

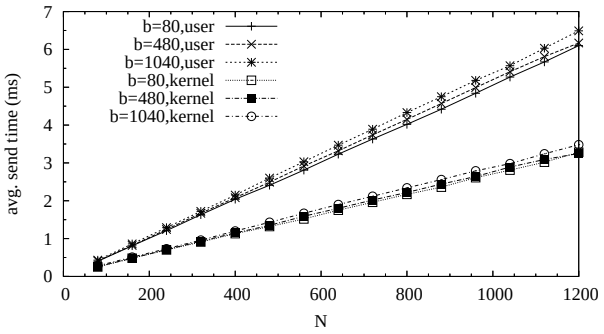

Figure 5. User/kernel-groupcast, Linux/ia64

6 Performance Prediction

In this section, we provide an illustration of how to apply the framework presented in Section 4. We use the experimental results obtained in Section 5.2 to determine the extent to which our proposed kernel-groupcast technique improves the server's capacity to service more users.

6.1 Send Time

The average time required for *send* operations using user-groupcast and kernel-groupcast (shown in Section 5.2) can be characterized using a regression model. Let $S_u(N, b)$ and $S_k(N, b)$ be the average send times with user-groupcast and kernel-groupcast, respectively. These send times can be described using an equation of the form:

$$xN + yNb \tag{7}$$

where x and y are constants. For example, on the Linux/i386 platform we have (in ms):

$$S_u(N, b) = (6.02\text{E-}3)N + (8.69\text{E-}7)Nb \tag{8}$$

and

$$S_k(N, b) = (2.16\text{E-}3)N + (2.12\text{E-}7)Nb. \tag{9}$$

Similarly, but not shown here, we determine equations for the ia64 platform. As an example, Figures 6 and 7 show how accurately these equations approximate the raw data obtained on these two platforms for $b = 80$.

To compare the efficiency of the two groupcast techniques, we compute the speedup factor r in Equation 5. We observe that r is independent of N but not b. For instance, the speedup factor on the Linux/i386 platform is:

$$r = \frac{S_u(N, b)}{S_k(N, b)} = \frac{(6.02\text{E-}3) + (8.69\text{E-}7)b}{(2.16\text{E-}3) + (2.12\text{E-}7)b} \tag{10}$$

where $S_u(N, b)$ and $S_k(N, b)$ are given by Equations 8 and 9, respectively.

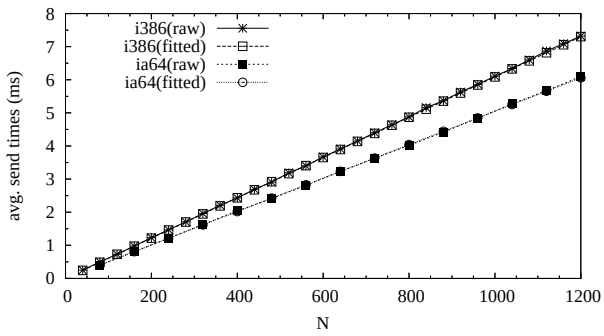

Figure 6. User-groupcast

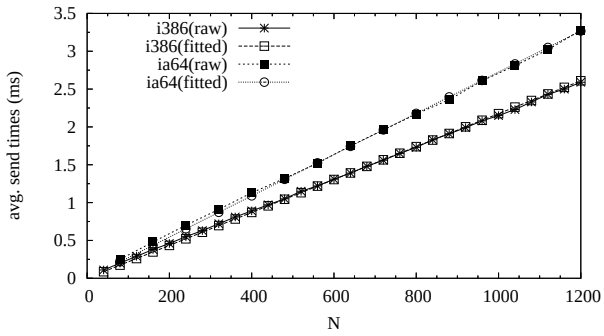

Figure 7. Kernel-groupcast

6.2 Other Processing Time

In addition to characterizing the send time, we also need to characterize the average processing time $\mathcal{H}$ for the *other* operations performed by the server. Recall that this includes the time spent receiving messages from users and the processing time required before being able to send updates to the clients. Clearly, $\mathcal{H}$ depends on the design and implementation of an application. Although there are numerous possibilities for $\mathcal{H}$, we consider two simple cases which we believe will cover a variety of applications. In the first case the server requires a constant amount of time per message:

$$\mathcal{H} := H_1 = c_1, \qquad c_1 \text{ is a constant,} \tag{11}$$

whereas in the second case the time spent per message is a linear function of N:

$$\mathcal{H} := H_2(N) = c_2 N, \qquad c_2 \text{ is a constant.} \tag{12}$$

In some cases one will know or will have measured how the processing time for the *other* operations changes with N. In these cases one can determine c_1 or c_2 directly. For those cases where these values cannot be determined directly, we now describe how they can be calculated.

Suppose user-groupcast is used. We define f as the fraction of time a server spends in distributing updates to the users. So $f \in [0, 1]$ and

$$f = \frac{\text{total send time}}{\text{total processing time}}. \tag{13}$$

If the server is using an immediate send model, one can derive $\mathcal{H}$ by conducting a simple experiment to determine f (as defined above) and t (the average time between sending updates), and then using their values to solve the following set of equations (derived from Equations 1 and 13):

$$\begin{aligned} f &= \frac{M\mathcal{S}}{M\mathcal{S} + M\mathcal{H}} \\ t &= M\mathcal{S} + M\mathcal{H} \end{aligned} \tag{14}$$

where M is the average number of messages received from the users during t, which can easily be obtained in the experiment.

If the server is using a periodic send model, one can again determine f and t, and use the following set of equations to derive $\mathcal{H}$ (derived from Equations 2 and 13):

$$\begin{aligned} f &= \frac{\mathcal{S}}{\mathcal{S} + M\mathcal{H}} \\ t &= \mathcal{S} + M\mathcal{H}. \end{aligned} \tag{15}$$

Substituting $S_u(N, b)$ into $\mathcal{S}$ and H_1 (or $H_2(N)$) from Equations 11 (or 12) into $\mathcal{H}$, we can solve for c_1 (or c_2). Then, using this result together with Equation 1 or 2, we can solve for N_u^{max} and N_k^{max}.

An alternative way to determine the values of c_1 and c_2 is to perform a stress test on the server in such a way that it is 100% utilized and the average update processing time does not exceed T. In this test, one measures the number of users that can be supported, which is actually N_u^{max}, and the fraction of time spent in *send* operations f. Then, we can solve Equation 13 for c_1 or c_2, depending on the type of function H being considered.

6.3 Numerical Examples

We now illustrate how our framework can be used to predict the performance improvement of a DVE server when using kernel-groupcast instead of user-groupcast.

In order to achieve smooth video and interactivity in video-based applications, the desired frame rate is 30 frames per second. In our examples, we therefore assume that $T = 33.3$ ms. We also assume that $\lambda = 1$, meaning that on average each user submits one message per time period T. Recall that T can be interpreted as the average delay a user observes between receiving successive updates from the server. Thus, it is reasonable for the server to expect to receive one message per time period T. We choose a packet size of 80 bytes (i.e., $b = 80$), which is about the mean packet size observed in some MOG systems [12].

With $b = 80$, the average send times on the Linux/i386 platform in Equations 8 and 9 become:

$$S_u(N) = (6.07\text{E-}3)N \tag{16}$$

and

$$S_k(N) = (2.17\text{E-}3)N, \quad (17)$$

respectively.

Suppose we measure f and t for a certain DVE server. Without loss of generality, we assume that $t = T$. Using the methodology presented in Section 6.2, we can determine the corresponding function H by Equation 16 or 17, and consequently derive the relative capacity improvement ratio k. The results of such computations are shown in Figures 8 and 9. They show how k changes with different values of f for the two different functions H.

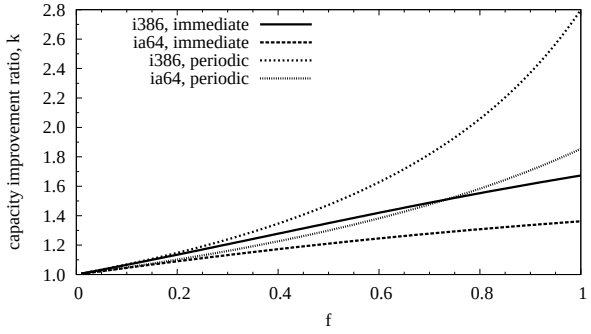

Figure 8. Capacity for f with constant H

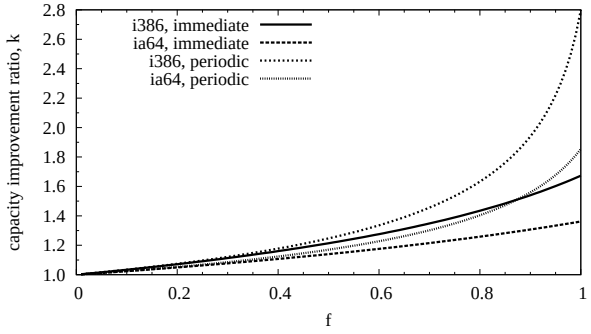

Figure 9. Capacity for f with linear H

These figures show how using kernel-groupcast can improve the overall capacity of some example server applications. The degree of improvement differs between the two platforms. In particular, the relative benefit of kernel-groupcast is higher on the Linux/i368 platform than on the ia64 platform (because of the larger differences in the reduction in send times due to kernel-groupcast on that platform). These results also show that as the *send* operation comprises a larger portion of the total processing time (i.e., as f increases), the capacity improvement increases. More specifically, for small values of f, the immediate send model benefits more from kernel-groupcast than the periodic send model. However, as f increases, the periodic send model yields a better relative capacity improvement ratio.

To better understand how the capacity of various servers can be improved (i.e., how k changes), we provide another perspective on the relationship between the costs of *send* and *other* operations. In particular, we define

$$\alpha = \frac{\text{time for sending one update to one user}}{\text{time for one } other \text{ operation}}. \quad (18)$$

The time for sending one update to one user is given by $S_u(N, b)/N$. Sending an update to a user and performing one round of *other* operations is considered to be the basic operation that one would observe at the server. The value of α can easily be measured by conducting a simple experiment or it may already be known. The relationship between α and f can be shown to be:

Case 1: periodic send model with constant H and immediate send model with linear H

$$\alpha = \frac{f}{1-f};$$

Case 2: immediate send model with constant H and periodic send model with linear H

$$\alpha = \frac{N_u^{max} f}{1-f}.$$

In the latter case, f depends on the number of users being supported by the server.

The results for the relative capacity improvement ratio (k) are shown again in Figures 10 and 11 this time using a range of values for α (from 0.1 to 10). In Case 1, $\alpha = 0.1$ and $\alpha = 10$ correspond to $f = 0.0909$ and $f = 0.9090$, respectively. In other words, it covers a range of servers that spend between roughly 10% and 90% of their time sending updates. In Case 2, the intuition behind $\alpha = 0.1$ is that it corresponds to an environment in which the cost of an individual *send* operation is one-tenth the cost of an *other* operation. At the other end of the spectrum shown in the graphs, $\alpha = 10$ corresponds to a scenario where the cost of an individual *send* operation is ten times the cost of an *other* operation.

In Figures 8 and 9 (as well as in Figures 10 and 11), we observe that the relative capacity improvement ratio k grows more quickly for small values of f and α in the immediate send model than in the periodic send model. This is because the immediate send model performs more *send* operations than the periodic send model. When the cost of performing the *other* operations is small relative to the cost of the *send* operations (i.e., f and α are small), kernel-groupcast provides greater benefits in the immediate send model. However, in the periodic model, if the cost of *send* operations is very high relative to the cost of *other* operations, larger improvements are theoretically possible, which will be discussed in the next section.

Intuitively, and as can be seen in Figures 8 and 9, as f approaches 1, the benefits that can be obtained from more

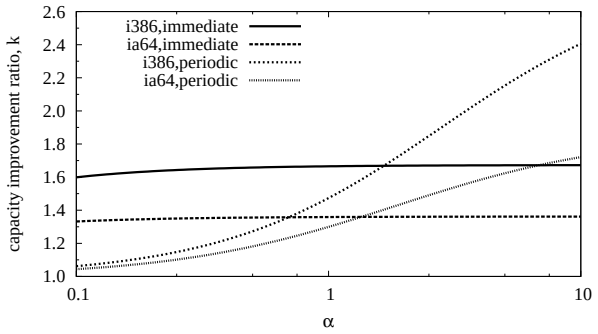

Figure 10. Capacity for α with constant H

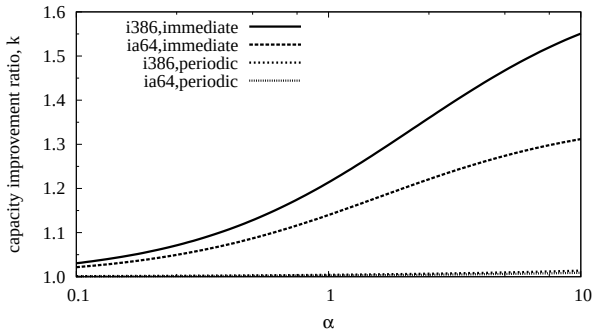

Figure 11. Capacity for α with linear H

efficient group communication is limited. Figures 10 and 11 also show that there is likely an upper limit on the benefits of improved group communication. In the next section, we will also derive such upper bounds.

6.4 Bounds on Capacity Improvements

In this section we show that there exist theoretical upper bounds on the capacity improvements, more specifically for the immediate send model

$$k \leq \sqrt{r} \tag{19}$$

and for the periodic send model

$$k \leq r. \tag{20}$$

We present a proof of the upper bound for the immediate send model; the result for the periodic send model follows from that result.

In the extreme case the *send* operation comprises the total processing time at the server (i.e., $f = 1$). This implies that the processing time for the *other* operation $\mathcal{H}$ becomes negligible. Consider the immediate send model. By Equation 1, we have:

$$\lambda N_u^{max} S_u(N_u^{max}, b) = T$$

and

$$\lambda N_k^{max} S_k(N_k^{max}, b) = T$$

for user-groupcast and kernel-groupcast, respectively. Combining these two equations,

$$N_u^{max} S_u(N_u^{max}, b) = N_k^{max} S_k(N_k^{max}, b). \tag{21}$$

Since our experimental results show that S_u and S_k can be expressed in the form $xN + yNb$ (Equation 7). Assume $S_u(N, b) = x_1 N + y_1 Nb$ and $S_k(N, b) = x_2 N + y_2 Nb$. We rewrite Equation 21 as follows:

$$N_u^{max}(x_1 N_u^{max} + y_1 N_u^{max} b) = N_k^{max}(x_2 N_k^{max} + y_2 N_k^{max} b).$$

After rearranging this equation, we obtain:

$$\left(\frac{N_k^{max}}{N_u^{max}}\right)^2 = \frac{x_1 + y_1 b}{x_2 + y_2 b}$$
$$k = \sqrt{r} \tag{22}$$

by the definition of k and Equation 10. As long as the speedup factor r is bounded (and in our case r is a constant), the relative capacity improvement k will be bounded. A similar approach can be used to derive the upper bound, $k \leq r$, for the periodic send model.

We note that the upper bounds are in fact independent of the characterization of $\mathcal{H}$. This means that these bounds are universally true, regardless of what the function H looks like. Also, the arrival rate of messages from a user λ does not affect the upper bounds.

The intuitive explanation is as follows. In the periodic send model, the total time required to perform *send* operations is linear in the number of users. If the cost of the *send* operations accounts for all of the total execution cost, improving the efficiency of each *send* operation by a constant factor r can lead to the same improvement in the overall server performance. In the immediate send model, the total time spent on the *send* operations, however, is quadratic in the number of users. Therefore, an improvement in the send cost can at most lead to a square-root improvement in overall server capacity.

As further explanation consider now a simple example. Recall that within T units of time, the immediate and the periodic send models require N^2 and N send updates, respectively. (See Equations 1 and 2, and the definition of $\mathcal{S}$.) Again, let λ be 1. Consider an example application server that can support only one user using user-groupcast ($N_u^{max} = 1$). Because the bound is obtained when the entire time period T is spent in sending updates, this implies that the cost of sending an update to a user is T. Now, suppose kernel-groupcast is four times as fast as user-groupcast

(i.e., r = 4). This reduces the sending time to $T/4$. Since the periodic send model requires N send updates, kernel-groupcast can now be used to support 4 users. That is, $N_k^{max} = 4$ and $k = 4 = r$. During the same period of time T, the immediate send model requires N^2 send updates. Although 4 *send* operations can be performed using kernel-groupcast, in this case the server can only support 2 users, yielding $N_k^{max} = 2$ and $k = 2 = \sqrt{r}$.

The difference between these two models is in the number of users being updated during the period T. While the same number of *send* operations is supported in both cases, because the immediate model requires N^2 updates to be sent (where the periodic model only requires N updates), the maximum gain in the relative capacity improvement ratio is limited to $\sqrt{r}$ for the immediate send model.

7 Summary

In this paper, we present an analytic framework to evaluate the potential benefits that would be obtained from using more efficient group communication mechanisms in a server-based DVE application. The bounds on capacity improvement, derived based on our framework, advance our understanding of these potential benefits on different server models. While the application focus is DVEs, our framework and results can also be applied to other Internet servers, e.g., streaming servers and conference bridges.

We use a prototype implementation of our kernel-groupcast to conduct measurement experiments on two modern server platforms. These measurements are then used together with the analytic framework to determine how the reductions in group send times would be translated into increased server capacity for a spectrum of applications.

Unfortunately, as discussed in Section 2, very little work has been done in characterizing the behavior of DVE servers, that can be used in our study. This makes it difficult to directly apply and comment on the capacity improvements that can be obtained in real applications by utilizing kernel-groupcast. We believe that this and other research could benefit greatly from a detailed characterization of client and server behavior in DVE systems. As part of our future work, we plan to conduct a range of experiments to obtain the necessary data. These experiments will also allow us to directly verify the predicted performance improvements on different platforms.

8 Acknowledgments

We gratefully acknowledge the Canada Foundation for Innovation, Hewlett-Packard, the National Sciences and Engineering Research Council of Canada, the Ontario Research Fund, and the Ontario Research and Development Challenge Fund for supporting portions of this work.

References

[1] Quake. http://www.idsoftware.com/games/quake/quake/.

[2] Rockymud. http://www.rockymud.net/.

[3] World of WarCraft. http://www.worldofwarcraft.com/.

[4] A. Abdelkhalek and A. Bilas. Parallelization and performance of interactive multiplayer game servers. In *Proceedings of the IPDPS 2004*, Santa Fe, New Mexico, April 2004.

[5] A. Abdelkhalek, A. Bilas, and A. Moshovos. Behavior and performance of interactive multi-player game servers. *Cluster Computing*, 6(4):355–366, 2003.

[6] G. Amdahl. Validity of the single processor approach to achieving large-scale computing capabilities. In *AFIPS Conference Proceedings*, volume 30, pages 483–485, 1967.

[7] Y. Amir, C. Danilov, and J. R. Stanton. A low latency, loss tolerant architecture and protocol for wide area group communication. In *Proceedings of the 2000 International Conference on Dependable Systems and Networks*, pages 327–336, Washington, DC, 2000.

[8] U. Borghoff and J. Schlichter. *Computer-supported cooperative work: Introduction to distributed applications*. Springer, 2000.

[9] S. Deering. Multicast routing in Internetworks and extended LANs. In *Proceedings of the ACM SIGCOMM*, pages 55–64, August 1988.

[10] C. Diot and L. Gautier. A distributed architecture for multiplayer interactive applications on the Internet. *IEEE Network*, 13(4):6–15, August 1999.

[11] C. Diot, B. Levine, B. Lyles, H. Kassem, and D. Balensiefen. Deployment issues for the IP multicast service and architecture. *IEEE Network*, 14(1):78–88, January 2000.

[12] J. Farber. Network game traffi c modelling. In *Proceedings of the 1st ACM Workshop on Network and System Support for Games*, pages 53–57, Bruanschweig, Germany, April 2002.

[13] M. Karsten, J. Song, M. Kwok, and T. Brecht. Effi cient operating system support for group unicast. In *Proceedings of the NOSSDAV'05*, pages 153–158, June 2005.

[14] M. Macedonia, M. Zyda, D. Pratt, P. Barham, and S. Zeswitz. NPSNET: A network software architecture for large scale virtual environments. *Presence*, 3(4):265–287, 1994.

[15] J. Muller, J. Metzen, and A. Ploss. Rokkatan: Scaling an RTS game design to the massively multiplayer realm. In *Proceedings of the ACM ACE 2005*, pages 125–132, Valencia, Spain, June 2005.

[16] R. van Renesse, K. P. Birman, and S. Maffeis. Horus: a flexible group communication system. *Communications of the ACM*, 39(4):76–83, 1996.

[17] M. Ye and L. Cheng. System-performance modeling for massively multiplayer online role-player games. *IBM Systems Journal*, 45(1):45–50, 2006.

[18] L. Zou, M. Ammar, and C. Diot. An evaluation of grouping techniques for state dissemination in networked multi-user games. In *Proceedings of the MASCOTS 2001*, pages 33–40, Cincinnati, Ohio, August 2001.