NORTHWESTERN
UNIVERSITY

Electrical Engineering and Computer Science Department

Technical Report
NWU-EECS-06-08
August 8, 2006

Prospects For Speculative Remote Display

Sam Rossoff **Peter Dinda**

Abstract

We propose an approach to remote display systems in which the client predicts the screen update events that the server will send and applies them to the screen immediately, thus eliminating the network round-trip time and making the system much more responsive in a wide-area environment. Incorrectly predicted events are undone when the actual events arrive from the server. The predictability of the events is core to the feasibility of this approach. Surprisingly, even a very naive predictor is able to correctly predict the next event 25-45% of the time. This suggests the prospects for speculative remote display are quite good.

Effort sponsored by the National Science Foundation under Grants ANI-0093221, ANI-0301108, and EIA-0224449. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author and do not necessarily reflect the views of the National Science Foundation (NSF).

Keywords: Remote display, windowing systems, user interfaces, prediction

Prospects For Speculative Remote Display

Sam Rossoff Peter Dinda
{s-rossoff,pdinda}@northwestern.edu
Department of Electrical Engineering and Computer Science, Northwestern University

Abstract

We propose an approach to remote display systems in which the client predicts the screen update events that the server will send and applies them to the screen immediately, thus eliminating the network round-trip time and making the system much more responsive in a wide-area environment. Incorrectly predicted events are undone when the actual events arrive from the server. The predictability of the events is core to the feasibility of this approach. Surprisingly, even a very naive predictor is able to correctly predict the next event 25-45% of the time. This suggests the prospects for speculative remote display are quite good.

1 Introduction

Remote display systems allow a distant user to control a computer or application with a graphical user interface. While this technology dates back to the 1980s and the X Window System [9], it has only recently become widely deployed through the success of VNC [8] and Microsoft's inclusion of the Remote Desktop Protocol (RDP) in the mainstream release of Windows. Remote display systems are also key components in thin-client computing [5, 10], which is seeing a resurgence because of the advent of virtual machines [2, 11].

Increasingly, remote display systems are being used over wide-area networks where round-trip latencies are inherently much higher than those in local-area networks, and have far greater variance [7, 4]. These higher latencies dramatically reduce the utility of remote display systems for end-users, making the remote computer seem choppy, slow, and hard to control.

The client and server in a remote display system communicate through two independent event streams. User events (keystrokes, mouse movements and clicks, etc) flow from the client to the server, while screen events (graphics primitives) flow from the server to the client. While neither VNC nor RDP require that the two event streams be synchronized, they are in fact synchronized through the user who will frequently wait for the effects of his actions to be shown in his display. The user is thus subject to the round-trip time of the network path and perceives the high mean latency and variance of the path as slowness and jitter in the display.

We propose the entirely new concept of *speculative remote display* as a way of eliminating or alleviating this problem. The key idea is for the client to predict future screen events from the history of past screen and user events and execute these screen updates immediately. As the actual server-supplied screen events arrive, they are compared against the predicted events that have already been speculatively executed. If there is a difference, the client rolls back (undos) the effects of the erroneously predicted events. In effect, the client is always executing a nested transaction on the frame buffer, and the server provides commit and abort messages. Provided that (1) the event streams are sufficiently predictable, and (2) users are sufficiently tolerant of rollbacks, the idea of speculative remote display has the potential to make wide-area remote display much more usable.

Figure 1(a) illustrates the structure of an RDP system, while Figure 1(b) illustrates the design of our speculative RDP system. Notice that in this design, only client changes are made.

In this initial report, we examine the prospects for speculative remote display by conducting a user study

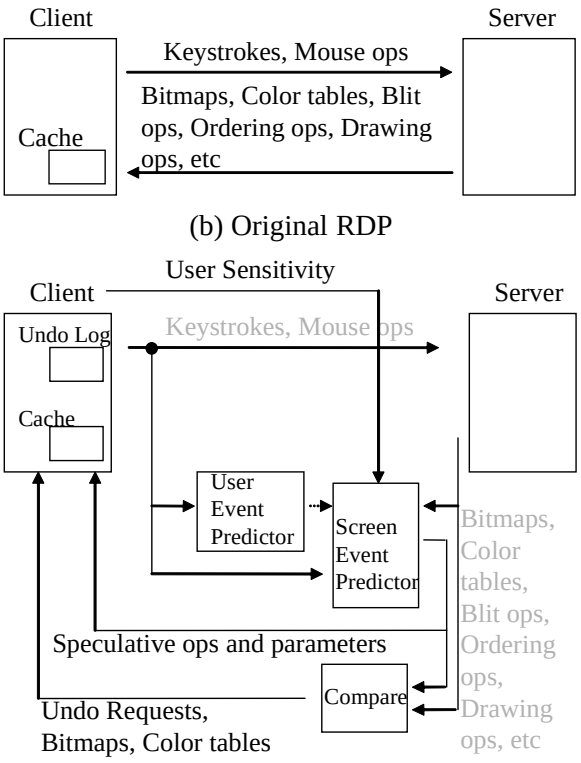


Figure 1. System design.

using an instrumented client for Microsoft's RDP. We focus on (1), the question of how predictable the event streams are. Surprisingly, even an extremely naive predictor has great success in predicting the next event from previous events, especially for the screen events. We are working on completing our full speculative client. With it, we will address question (2) by measuring user tolerance with increasingly aggressive speculation and comparing it to user tolerance without speculation, both on high latency, high jitter networks. For that study, we will use the techniques that have been previously developed for measuring user tolerance with resource borrowing [3]. We expect that the degree of aggressiveness will best be directly controlled by the end-user, a technique that has been previously used in CPU scheduling [6].

2 Prediction study

A key requirement for speculative remote display is that there be, in practice, a considerable degree of predictability in the user and screen event streams. We want to map from all the user and screen events in the past to the predicted next screen event and the probability that it will be correct. In the following, we demonstrate that even a naive predictor performs surprisingly well at predicting the next user event from the history of previous user events, and the next screen event from the history of previous screen events.

2.1 Predictor

Our predictor is a k-th order Markov model. The symbols that the Markov model operates on are the user or system events as supplied as human-readable strings. A typical event contains the type of event (e.g., mouse movement) and its parameters (e.g., (x, y) coordinates). A state is defined as the simple concatenation of the last k symbols. Of course, given this simple scheme, the bound on the state space size of a model is $O(n^{k+1})$, where n is the number of distinct input symbols. Furthermore, because we include parameters in the symbols, n can potentially be astronomical. For this reason, our implementation can constrain the number of states to be between an upper and lower limit, keeping the most visited states and garbage collecting the rest when the upper limit is reached.

Our implementation supports continuous model fitting and prediction. That is, it can operate on a stream of symbols, updating the model on each new symbol as well as supplying a prediction of the symbol that is most likely to occur next. If there is insufficient information (e.g., we are in a state which currently has no outgoing arcs), then the predictor does not attempt to predict the next state.

Initially, our intent was to use Markov models to predict the type of the next event and discrete parameters with a small range, such as bitmap IDs, relying on much more compact linear time series models to capture positional parameters. We were stunned, however, to find that simply using Markov models as described above worked surprisingly well. We are continuing to explore different prediction approaches.

2.2 Traces

To collect trace data to evaluate our predictor, we instrumented the rdesktop [1] open source RDP client so

that it will non-intrusively record all user and screen events to files. We then created an experimental testbed consisting of 2 PCs (P4, 2 GHz, 128 MB, 20" LCD display, Windows XP) connected via a private 100 mbit network. Notice that this is an ideal remote desktop configuration—network latency and jitter are minimized. To collect a trace, the user sat at one PC and used rdesktop (at 1024x768 resolution and 24 bit color) to use the other PC. The user performed the following tasks:

- Acclimatization. (5 minutes)
- Word processing with Microsoft Word 2003. The user spent 15 minutes recreating a supplied document.
- Presentation creation with Microsoft Powerpoint 2003. The user spent 15 minutes recreating a supplied document with considerable drawing required.
- Web browsing using Microsoft Internet Explorer. For 15 minutes, the user visited a news web site, read an article, and then conducted web searches on its topic in another window. (15 minutes).
- First person shooter game playing for 15 minutes. The user played Quake II.

Four users participated. They included undergraduates and graduate students in the computer science department at Northwestern. We are currently working to extend the range of users. The user traces contain 68 to 112 thousand events, while the screen traces contain 2.5 to 4.4 million events.

2.3 Results

We ran the screen and user traces generated by each user through our online Markov predictor, varying the order of the model and the upper and lower limits on the number of states in the model. For each combination of trace, order, and limits, the predictor started with no information at the beginning of the trace and formed its model progressively as it saw the input symbols. This is identical to how a predictor in a speculative remote display system would operate.

The graphs in Figure 2 show the percentage of prediction attempts that are successful, as a function of the order of the model. (a)–(c) are for the user traces, with progressive limitation on the number of states permitted, while (d)–(f) are the screen traces with the same progressive limitation. Each curve corresponds to a particular user. The key point is that with an imminently practical 1000–2000 state model ((b) and (e)), virtually all attempts to predict the next screen event are successful, while, with a sufficiently high order model, $\sim 90\%$ of all attempts to predict the next user event are successful. If we have seen any transitions out of a state before (i.e., if we have seen the state before), we almost always predict the next state correctly.

The graphs of Figure 3 correspond exactly to those of the previous figure with the exception being that we are plotting the percentage of all events that are successfully predicted. That is, it includes those cases where we have not seen the state before as erroneous predictions. Remarkably, with our 1000–2000 state model, we are still able to correctly predict 25-45% of screen events and 12-14% of the events of most of our users.

The upshot of Figures 2 and 3 is that we have found that both screen and user events are surprisingly predictable with even an extremely naive predictor. As predictability of these events is a core requirement for speculative remote display to work, the prospects for the idea seem bright.

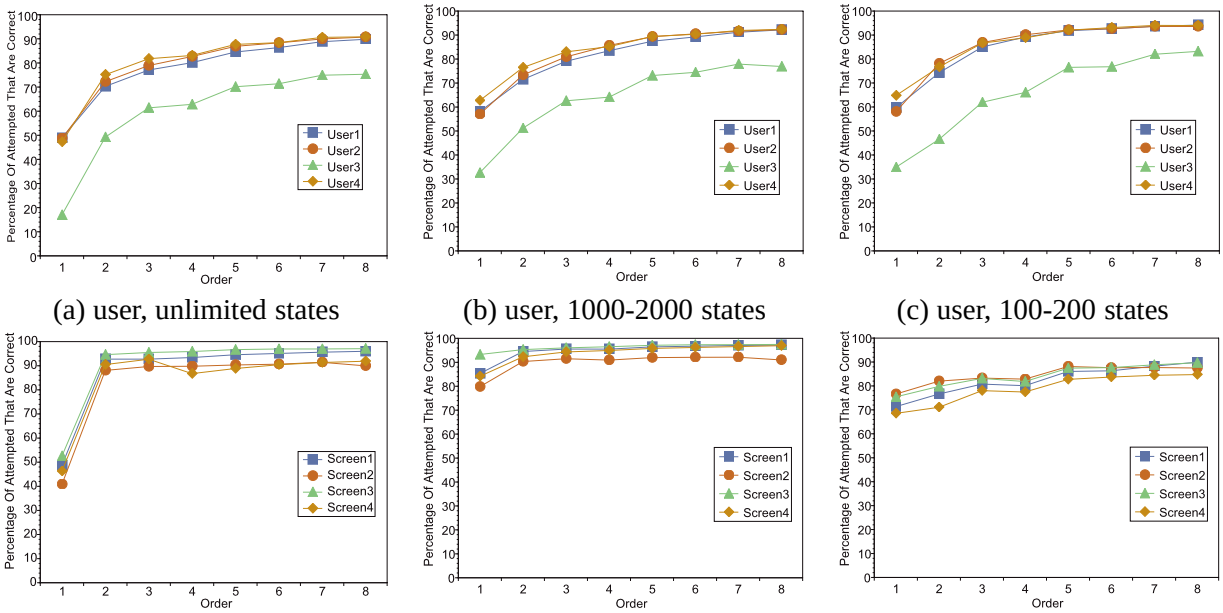
3 Conclusions

We have introduced the idea of a remote display system that predicts and speculatively executes screen update events in order to ameliorate the high latencies seen on wide-area networks. The prospects for this idea are critically dependent on the predictability of user and (particularly) screen events. Surprisingly, even a very naive predictor is able to correctly predict screen events very well. Having found the prospects for speculative remote display to be very good, we are now working to implement the system.

References

[1] CHAPMAN, M. rdesktop: A remote desktop protocol client for accessing windwos nt terminal server. http://www.rdesktop.org.

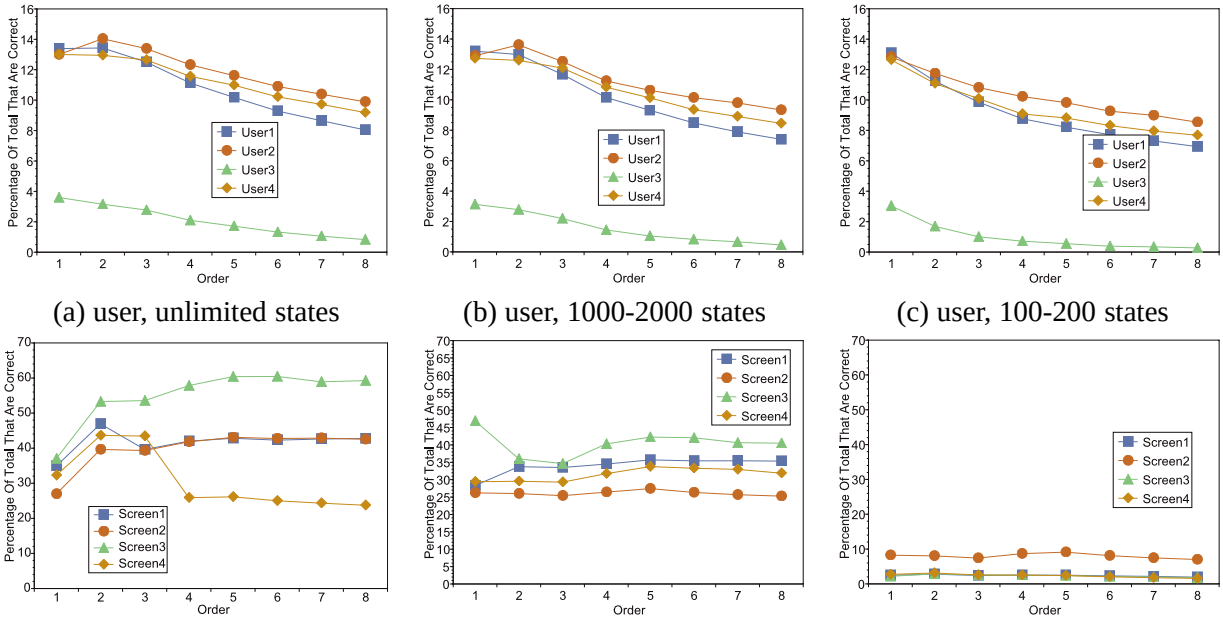
[2] GARFINKEL, T., PFAFF, B., CHOW, J., ROSENBLUM, M., AND BONEH, D. Terra: A virtual



(a) user, unlimited states (b) user, 1000-2000 states (c) user, 100-200 states

(d) screen, unlimited states (e) screen, 1000-2000 states (f) screen, 100-200 states

Figure 2. Percentage of prediction attempts that are correct.



(a) user, unlimited states (b) user, 1000-2000 states (c) user, 100-200 states

(d) screen, unlimited states (e) screen, 1000-2000 states (f) screen, 100-200 states

Figure 3. Percentage of all events that are predicted correctly.

machine-based platform for trusted computing. In *Proceedings of the 19th ACM symposium on Operating systems principles SOSP 2003* (2003), pp. 193–206.

[3] GUPTA, A., LIN, B., AND DINDA, P. A. Measuring and understanding user comfort with resource borrowing. In *Proceedings of the 13th IEEE International Symposium on High Performance Distributed Computing (HPDC 2004)* (June 2004).

[4] KARAM, M., AND TOBAGI, F. A. Analysis of the delay and jitter of voice traffic over the internet. In *Proceedings of IEEE INFOCOM* (2001), pp. 824–833.

[5] LAI, A., AND NIEH, J. Limits of wide-area thin-client computing. In *Proceedings of the ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems* (2002).

[6] LIN, B., AND DINDA, P. Vsched: Mixing batch and interactive virtual machines using periodic real-time scheduling. In *Proceedings of ACM/IEEE SC 2005 (Supercomputing)* (November 2005).

[7] PAXSON, V. End-to-end routing behavior in the Internet. In *Proceedings of the ACM SIGCOMM* (New York, August 1996), ACM Press, pp. 25–38.

[8] RICHARDSON, T., STAFFORD-FRASER, Q., WOOD, K., AND HOPPER, A. Virtual network computing. *IEEE Internet Computing 2*, 1 (January/February 1998).

[9] SCHEIFLER, R. W., AND GETTYS, J. The x window system. *ACM Transactions on Graphics 5*, 2 (April 1986).

[10] SCHMIDT, B., LAM, M., AND NORTHCUTT, J. The interactive performance of slim: A stateless thin client architecture. In *Proceedigns of the 17th ACM Symposium on Operating Systems Principles (SOSP 1999)* (December 1999), pp. 32–47.

[11] SUNDARARAJ, A., GUPTA, A., , AND DINDA, P. Increasing application performance in virtual environments through run-time inference and adaptation. In *Proceedings of the 14th IEEE International Symposium on High Performance Distributed Computing (HPDC)* (July 2005), pp. 47–58.