Dynamic Load Adaptation in Internet Real-Time Applications

Kevin Fall*
Lawrence Berkeley National Laboratory
Berkeley, CA
kfall@ee.lbl.gov

This work explores the use of dynamic load adaptation with respect to both operating system and network in developing distributed perceptual-time collaborative tools with a focus on end system issues. Real-time distributed applications are generally characterized by placing a time constraint or deadline on the completion time of some end-to-end unit of work. Such applications are subdivided based on the rigor of the timing constraint, yielding the classes of "hard" and "soft" real-time, where soft real-time generally suggests that useful work can be accomplished even if the timing constraint is not met. A special case of soft real-time involves the time needed by a human to process information, known as perceptual-time [1]. Approaches to satisfying real-time constraints may include pre-allocation, dynamic adaptation, or some combination of the two. We focus on constructing perceptual-time applications in environments lacking resource pre-allocation, as most current operating systems and networks lack resource reservation. Furthermore, future network services beyond best-effort delivery are likely to be charged at premium rates.

Our approach to constructing real-time applications in environments lacking pre-allocation of resources depends primarily on dynamic adaptation to observed delay, bandwidth, and loss characteristics. Assuming the case of real-time multicast message distribution using current operating systems and the Internet, receivers may smooth network-induced jitter by dynamically adjusting receive buffer sizes and attempt to avoid network congestion by limiting bandwidth use during periods of high loss. Current Internet routers generally discard traffic when FIFO queues overflow due to persistent or transient congestion; end systems discard traffic when the time required to process incoming network messages exceeds the message interarrival time. Both cases lead to burst losses, often causing quality degradation exceeding that experienced when applications can schedule or limit their own losses.

Two mechanisms are used to reduce loss detected by the application. The first, Receiver-driven Layered Multicast (RLM) [2] decreases network bandwidth use as loss is observed. The second, Early Discard Load Shedding (EDLS) [3], reduces end system CPU utilization as loss is observed. These two mechanisms may be implemented separately, but should be combined for best performance. In RLM, sources are constructed such that they encode data across multiple channels in quality "layers." For example, a video source may use a form of layered video coding, with each layer carried in a separate logical network channel using IP multicast. Receivers subscribing to all groups receive the highest quality but incur the highest network load; those observing loss infer the network is congested and react by unsubscribing from the higher-quality layers. Unsubscribing

[0]This work was supported by the Director, Office of Energy Research, Scientific Computing Staff, of the U.S. Department of Energy under Contract No. DE-AC03-76SF00098.

from a group (layer) causes the IP multicast machinery to cease delivering the deselected group, provided no additional viewers are present. Should the congestion abate, the application subscribes to additional layers, thereby improving its quality. Additional details of RLM are given in [2]; the remaining discussion explores improvements to the original EDLS design discussed in [3].

EDLS aims at reducing CPU load (thus limiting the average time required to process arriving network messages) by selectively discarding packets early in the processing pipeline. As with RLM, the procedure is generally invoked when loss is detected, but may also be used to limit an application's overall CPU utilization when executing several real-time cooperating processes on the same end station. Usually, observed loss should first trigger EDLS if overall CPU utilization is high (suggesting packets are being discarded at the local system). High loss coupled with low CPU utilization suggests the network is in overload, implying bandwidth consumption should be reduced.

With EDLS, an application is provided with an input parameter representing the maximum percentage of CPU time it is allowed to execute, subject ultimately to the local operating system scheduler. The application is structured to subdivide this percentage among its own tasks, such that the application never exceeds the bound provided. For example, in a prototype video playback application, EDLS is implemented with two algorithms. The maximum time is provided as the input to the first algorithm which approximates fair-share allocation among the currently active video sources and runs periodically. Each (per-source) decompress/display processing path is controlled by a token bucket regulator with rate with rate R and size B (fixed at 1.0). The regulator controls the processing duty cycle over one-second intervals for each decompress/display path. R is computed based on an exponentially smoothed estimate of the current frame rate for each path and a linear prediction of the amount of time required to process an average-sized frame. During normal operation, a received frame causes an increase in the contents of the token bucket proportional to 1/R, and the time to decompress and display it is subtracted. The periodic execution (on the order of once per three seconds) of the first algorithm provides re-allocation of CPU time among various video sources should some sources leave or others join the session.

The EDLS mechanism and a layered video codec for use with RLM is currently implemented in an experimental version of the *vic* [4] video application. Future work will include the combination of the EDLS and RLM, further refinement of the loss detection machinery, and performance measurements of the combined system. Multiple cooperating real-time processes and their interactions in a non-real-time scheduled operating system environment is also an area of planned further study.

References

[1] Lindblad, Christopher and Tennenhouse, David, "The VuSystem: A Programming System for Compute-Intensive Multimedia", IEEE Journal on Selected Areas in Communications, 1996, to appear

[2] McCanne, Steven and Jacobson, Van and Vetterli, Martin, "Receiver-driven Layered Multicast", Proc. SIGCOMM '96, Aug 1996, p.117-130

[3] Fall, Kevin and Pasquale, Joseph and McCanne, Steven, "Workstation Video Playback Performance with Competitive Process Load", Proc. NOSSDAV '95, Apr 1995, p. 179-182

[4] McCanne Steven and Jacobson, Van, "vic: A Flexible framework for packet video", Proc. Multimedia '95, Nov 1995, p. 511-522