A Comparative Study of the Realization of Rate-Based Computing Services in General Purpose Operating Systems

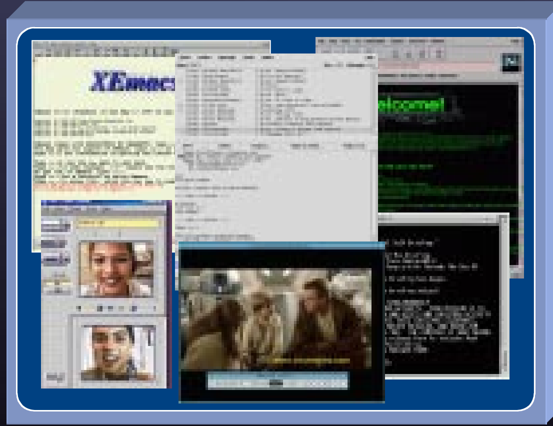
Kevin Jeffay
Department of Computer Science
University of North Carolina
at Chapel Hill
jeffay@cs.unc.edu

Gerardo Lamastra
ReTiS Lab
Scuola Superiore di Studi
Universitari e Perfezionamento
S.Anna, Pisa, Italy

http://www.cs.unc.edu/Research/Dirt/

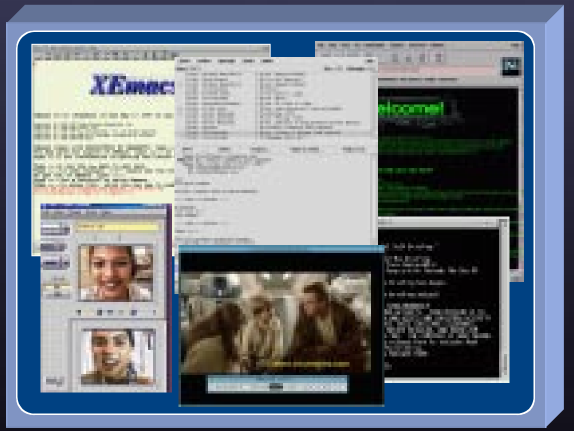
Rate-Based Resource Allocation
A technology for real-time computing on the desktop

- The Problem:
  - How to provide integrated real-time computation and communication services in a time-shared operating system?



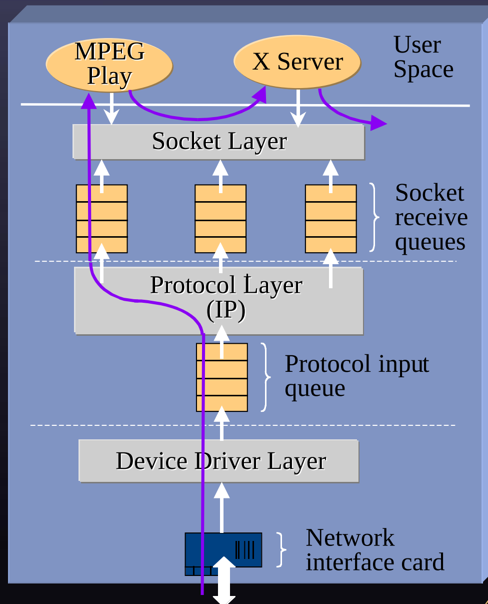
Rate-Based Resource Allocation
A technology for real-time computing on the desktop

- The Problem
- The Solution:
  - Rate-based resource allocation
    - » Proportional share
    - » Server algorithms
    - » "*x* out of *y*" algorithms
- Which solution works best?



Rate-Based Resource Allocation
Integrated real-time resource allocation example

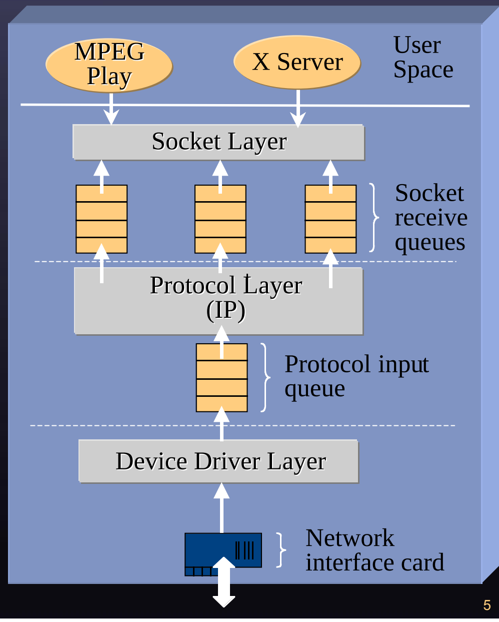
- Data arrives for a video conference over the network
- It is processed by the operating system and delivered to the application
- The application further processes and sends to the window system
- The window system paints the screen



Rate-Based Resource Allocation

Integrated real-time resource allocation example

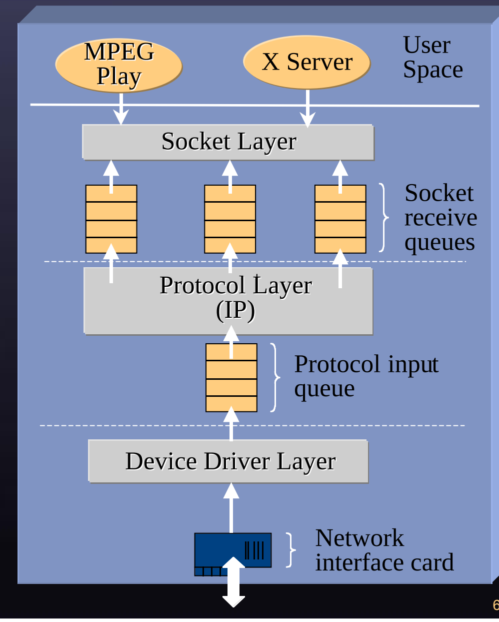
- Technical challenges:
  - Device scheduling and protocol processing
  - Application and system call scheduling
- Candidate technologies
  - Proportional share scheduling (EEVDF)
  - Constant Bandwidth Servers (CBS)
  - Rate-Based extensions to Liu and Layland (RBE)



Rate-Based Resource Allocation

Integrated real-time resource allocation example

- Our study:
  - Compare the performance of applications of rate-based scheduling technology at various levels in the kernel
  - For various characterizations of real-time processing work-loads
    - » Well-behaved periodic job/task arrivals
    - » Bursty job/task arrivals
    - » "Misbehaved" job/task arrivals



Rate-based resource allocation schemes

Proportional share resource allocation

- Processes are allocated a *share* of the processor's capacity
  - Process i is assigned a *weight* w_i
  - Process i's *share* of the CPU at time t is

$$f_i(t) = \frac{w_i}{\sum_{j \in A(t)} w_j}$$

- If processes' weights remain constant in $[t_1, t_2]$ then process i receives

$$S_i(t_1,t_2) = \int_{t_1}^{t_2} f_i(t)\,dt = \frac{w_i}{\sum_j w_j}(t_2 - t_1)$$

units of execution time in $[t_1, t_2]$

Rate-based resource allocation schemes

Constant bandwidth server

- A *server* process executes tasks every T_S time units
- When a request arrives it is serviced with a deadline of

$$d_k = \text{MAX}(t_k, d_{k-1}) + c_k / U_S$$

where

- t_k is the arrival time of the k^{th} task request
- c_k is the execution cost of the k^{th} task request
- $U_S = C_S/T_S$ is the capacity (utilization) of the server
- d_{k-1} is the deadline of the previous task request

Rate-based resource allocation schemes
Rate-based Liu & Layland scheduling

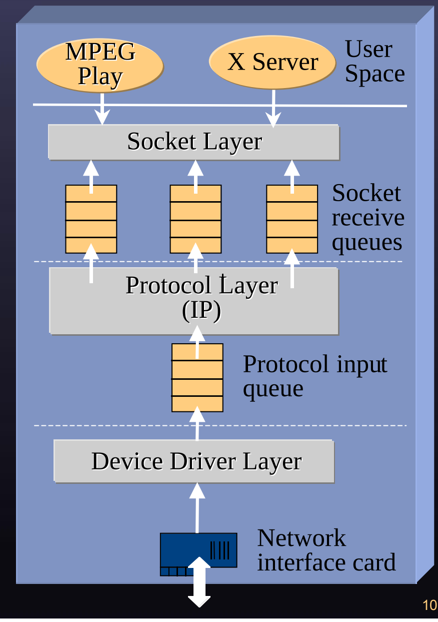
- Processes make progress at the rate of processing *x* events every *y* time units and each event is processed within *d* time units
- For task *i* with rate specification (x_i, y_i, d_i), the j^{th} event for task *i*, arriving at time $t_{i,j}$, will be processed by time

$$D(i,j) = \begin{cases} t_{i,j} + d_i & \text{if } 1 \le j \le x_i \\ \mathrm{MAX}(t_{i,j} + d_i,\ \ D(i, j-x_i)+y_i) & \text{if } j > x_i \end{cases}$$

  - Deadlines occur at least *d* time units after a job is released
  - Deadlines separated by at least *y* time units

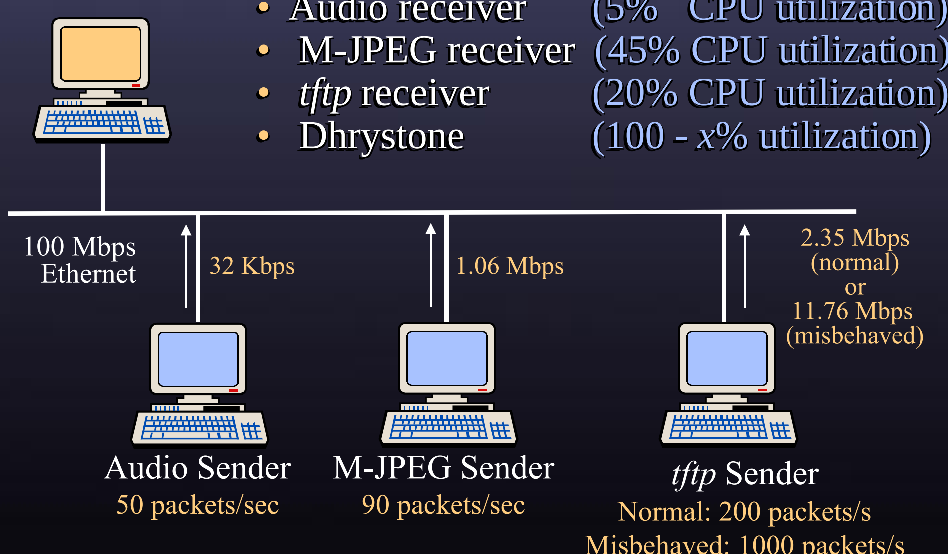
Empirical comparisons
Experimental setup

- Modify FreeBSD UNIX to support rate-based scheduling in the “top” and “bottom” halves of the kernel
- Consider the performance of each rate-based scheme in isolation and in combinations
  - Consider the performance across a variety of multimedia workloads



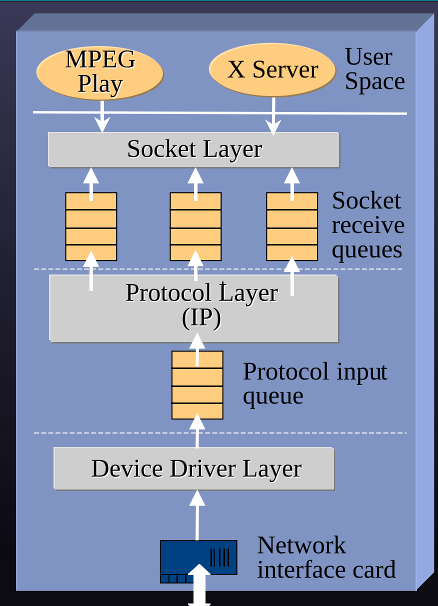
Experimental Setup
Workload generation

- Audio receiver (5% CPU utilization)
- M-JPEG receiver (45% CPU utilization)
- *tftp* receiver (20% CPU utilization)
- Dhrystone (100 - *x*% utilization)



Empirical Comparisons
Performance metrics setup

- Packets dropped at the IP layer
- Packets dropped at the socket layer
- Packets delivered to the application
- Dhrystone performance
- NIC to application response time
- Deadline miss percentage

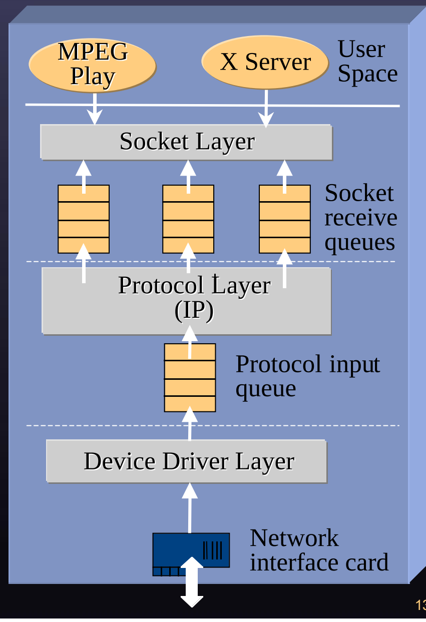


Empirical Comparisons
Experimental plan

- First consider using only
  - Proportional share,
  - CBS, and
  - RBE

  scheduling for all resource allocation problems

- Then attempt to match algorithms to the specific allocation problems where they are best suited



Experimental Results Summary
Well-behaved, periodic packet arrivals

	Prop Share			CBS			RBE		
Phone	0	0	2,993	0	0	2,977	0	0	3,000
ftp	0	0	11,961	2	0	11,914	0	0	11,944
M-JPEG	0	0	5,346	0	0	5,388	0	0	5,443

IP Drops — Socket Drops — Packets Delivered

- In isolation, all rate-based schemes give “perfect” (or very good) performance
  - No packets are dropped
- Liu & Layland rate-based scheduling (RBE) provides the best response times
  - (Not surprising)

Experimental Results Summary
Bursty (pareto) packet arrivals

	Prop Share			CBS			RBE		
Phone	1,585	0	1,312	0	0	2,938	0	0	3,027
ftp	5,315	0	5,408	5	0	10,760	0	0	10,778
M-JPEG	2,705	0	2,498	0	0	3,192	0	0	5,287

IP Drops — Socket Drops — Packets Delivered

- Proportional share scheduling degrades the performance of all applications uniformly
  - A (bad) artifact of quantum-based allocation
- CBS and RBE smooth the arrival process
  - Event driven scheduling works well here
  - Pure event-driven scheduling (RBE) gives lowest response times

Experimental Results Summary
“Misbehaved” *ftp* packet arrivals

	Prop Share			CBS			RBE		
Phone	5	0	2,997	0	0	2,978	0	0	2,998
ftp	17,999	0	11,902	17,880	0	12,120	0	9,052	20,794
M-JPEG	56	0	5,390	0	0	5,391	0	0	5,444

IP Drops — Socket Drops — Packets Delivered

- Proportional share and CBS provide excellent protection/isolation for well-behaved tasks
  - *ftp* packets dropped at the IP layer
- RBE scheduling drops *ftp* packets at the socket layer
  - Pure event-driven scheduling provides no isolation
  - Dhrystone performance suffers drastically

First-Round Experiments Summary
So what?

- When workload is well-behaved all schemes perform well
- Pure-event driven scheduling and quantum allocation don’t work well for “bottom-half” kernel processing
- Server-based allocation doesn’t work well for application-level processing

Combine the scheduling schemes to better match the processing requirements at each level in the system

Combining Allocation Policies
Getting the best of all worlds

- CBS+Proportional Share scheduling

	Constant Rate			Bursty			Misbehaved		
Phone	0	0	2,869	0	0	2,998	0	0	2,797
ftp	0	0	11,722	0	0	10,340	17,898	0	11,545
M-JPEG	0	0	5,343	0	0	4,951	0	0	5,398

- RBE+Proportional Share scheduling

	Constant Rate			Bursty			Misbehaved		
Phone	0	0	2,873	0	0	2,954	0	0	2,789
ftp	0	0	11,802	0	0	10,437	17,872	0	11,647
M-JPEG	0	0	5,324	0	0	4,956	0	0	5,393

IP Drops — Socket Drops — Packets Delivered

Rate-Based Resource Allocation
Conclusions

- “One size does not fit all” (unless the external environment is (perfectly) well-behaved)
  - Quantum allocation within the kernel leads to coarse-grained control
  - Server-based allocation impractical for applications
  - Pure event scheduling doesn’t provide isolation
- Different scheduling algorithms work best at different levels of the kernel
  - Event scheduling best at the device layer
  - Server/quantum scheduling best at the application/ system call layer