*The* UNIVERSITY *of* NORTH CAROLINA
*at* CHAPEL HILL

# Rate-Based Resource Allocation Models for Multimedia Computing and Embedded Systems

*Kevin Jeffay*
Department of Computer Science
University of North Carolina
at Chapel Hill

*Steve Goddard*
Computer Science & Engineering
University of Nebraska – Lincoln

April 2004

***http://www.cs.unc.edu/Research/dirt***

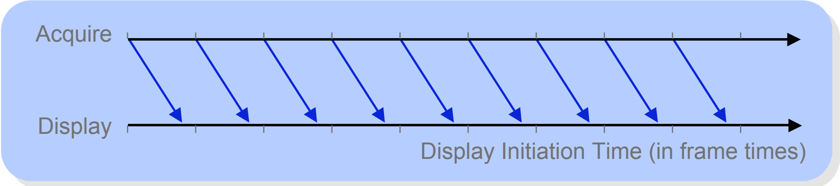
## Rate-Based Resource Allocation
### The case against static priority scheduling

- Static priority scheduling in general, and Rate Monotonic scheduling in particular, dominates in the real-time systems literature
  - VxWorks, VRTX, QNX, pSOSystems, LynxOS all support static priority scheduling
- Does one size fit all?
  - “When you have a hammer, everything looks like a nail”
- Problems with static priority scheduling
  - Feasibility is dependent on a predictable environment and well-behaved tasks.

## Rate-Based Resource Allocation
### Overview

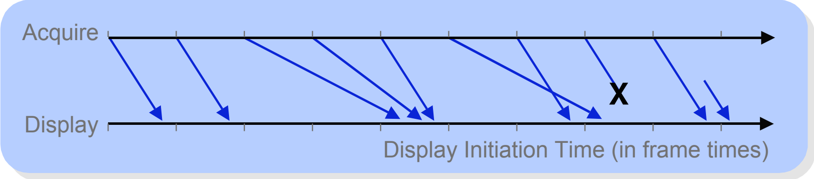
- The problem:
  - How to allocate resources in an environment wherein…
    - Work arrives at well-defined but highly variable rates
    - Tasks may exceed their execution time estimates
  - … and still guarantee adherence to deadlines
- The thesis:
  - Static priority scheduling is the wrong tool for the job (existing task models are too simplistic)
  - Rate-based scheduling abstractions can simplify the design and implementation of many real-time systems and improve performance and resource utilization

## The Case Against Priority Scheduling
### Example: Display-side multimedia processing



- The problem: Receive frames from the network and deliver to a display application so as to ensure...
  - Continuous playout
  - Minimal playout latency
- The theory: Multimedia is easy — it’s periodic!
  - Apply existing theory of periodic or sporadic tasks

# Display-side Media Processing
## The practice

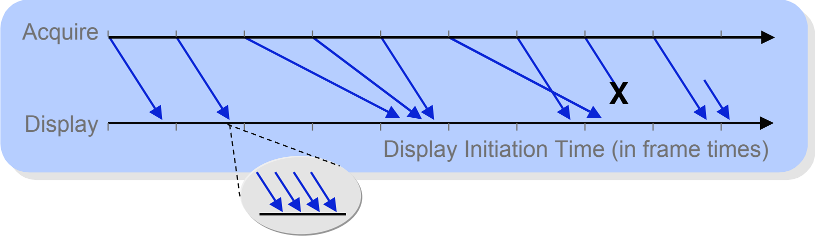

Acquire
Display
X
Display Initiation Time (in frame times)

- Nothing is periodic in a distributed system!

- The effects of distributed systems pathology:
  - Variable message transmission times
  - Out-of-order message arrivals
  - Lost & duplicate messages

---

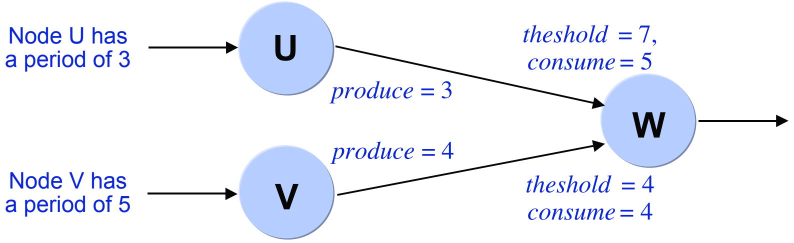
# Display-side Media Processing
## Managing the Network Interface

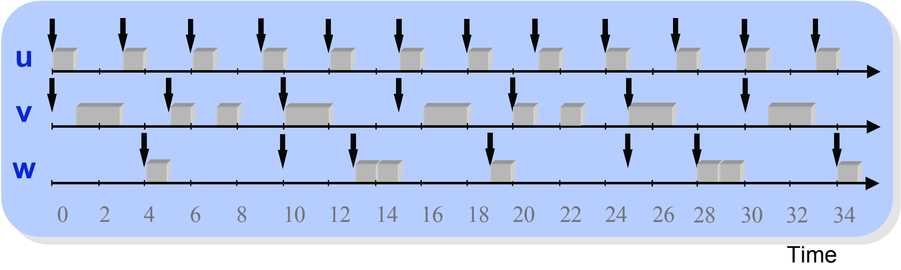

Acquire
Display
X
Display Initiation Time (in frame times)

- Packets fragmented in the network must be reassembled
  - *Messages* have deadlines, *packets* do not
  - Applications know about messages, operating systems do not

---

# The Case Against Priority Scheduling
## Example: Signal processing data flow graphs


Node U has a period of 3
U
produce = 3
theshold = 7,
consume = 5
W
produce = 4
Node V has a period of 5
V
theshold = 4
consume = 4



---

# Rate-Based Computing
## Approaches

- Extend the Liu and Layland model of real-time tasks to allow the expression of real-time rates
  - Hierarchical “server-based” scheduling — Create a “server” process that is scheduled as a periodic task and internally schedules the processing of aperiodic events
  - Event-based scheduling — Process aperiodic events as if they were generated by a virtual “well behaved” periodic process

- Adapt “fluid-flow” models of resource allocation developed in the networking community for bandwidth allocation to CPU scheduling
  - Provide a “virtual processor” abstraction wherein each task logically executes on a dedicated processor with $1/f(n)$ the capacity of the physical processor

# An Event-Based Rate Model

## The Rate-Based Execution (*RBE*) model

- Tasks make progress at the rate of processing $x$ events every $y$ time units and each event is processed within $d$ time units (in the best case)
- 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

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

  - $D(i,j)$ gives the earliest possible deadline for the $j^{th}$ instance of task $i$ ($\geq t_{i,j} + d_i$)

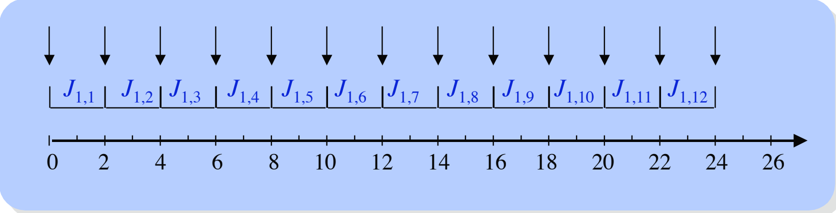
# The RBE Task Model

## Example: Periodic arrivals, periodic service

- Task with rate specification ($x = 1$, $y = 2$, $d = 2$)

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

  - Deadlines separated by at least $y = d = 2$ time units
  - Deadlines occur at least 2 time units after a job is released


$J_{1,1}$
$J_{1,2}$
$J_{1,3}$
$J_{1,4}$
$J_{1,5}$
$J_{1,6}$
$J_{1,7}$
$J_{1,8}$
$J_{1,9}$
$J_{1,10}$
$J_{1,11}$
$J_{1,12}$
0
2
4
6
8
10
12
14
16
18
20
22
24
26

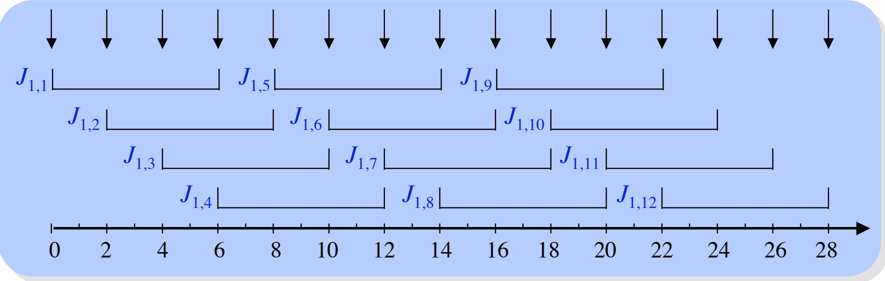
# The RBE Task Model

## Example: Periodic arrivals, *deadline ≠ period*

- Task with rate specification ($x = 1$, $y = 2$, $d = 6$)

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

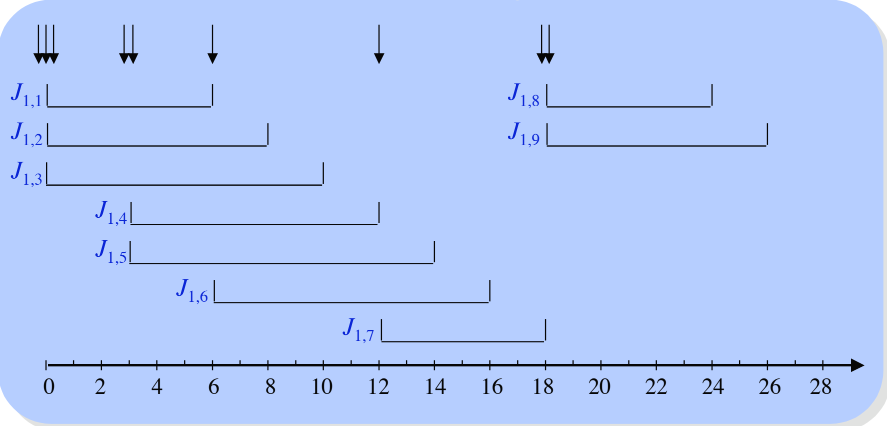
  - Deadlines separated by at least $y = 2$ time units and occur at least $d = 6$ time units after a job is released


$J_{1,1}$
$J_{1,5}$
$J_{1,9}$
$J_{1,2}$
$J_{1,6}$
$J_{1,10}$
$J_{1,3}$
$J_{1,7}$
$J_{1,11}$
$J_{1,4}$
$J_{1,8}$
$J_{1,12}$
0
2
4
6
8
10
12
14
16
18
20
22
24
26
28

# The RBE Task Model

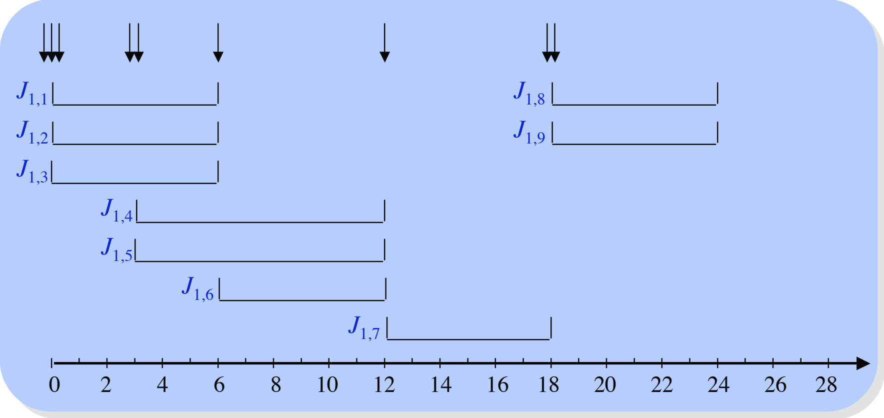
## Bursty arrivals

- Task with rate specification ($x = 1$, $y = 2$, $d = 6$)
  - Deadlines separated by at least $y = 2$ time units and occur at least $d = 6$ time units after a job is released


$J_{1,1}$
$J_{1,8}$
$J_{1,2}$
$J_{1,9}$
$J_{1,3}$
$J_{1,4}$
$J_{1,5}$
$J_{1,6}$
$J_{1,7}$
0
2
4
6
8
10
12
14
16
18
20
22
24
26
28

# The RBE Task Model
## Bursty arrivals

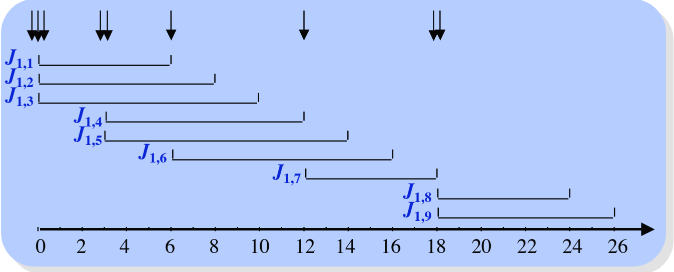
- Task with rate specification ($x = 3$, $y = 6$, $d = 6$)
  - Deadlines separated by at least $y = 6$ time units and occur at least $d = 6$ time units after a job is released


$J_{1,1}$
$J_{1,2}$
$J_{1,3}$
$J_{1,4}$
$J_{1,5}$
$J_{1,6}$
$J_{1,7}$
$J_{1,8}$
$J_{1,9}$
0 2 4 6 8 10 12 14 16 18 20 22 24 26 28

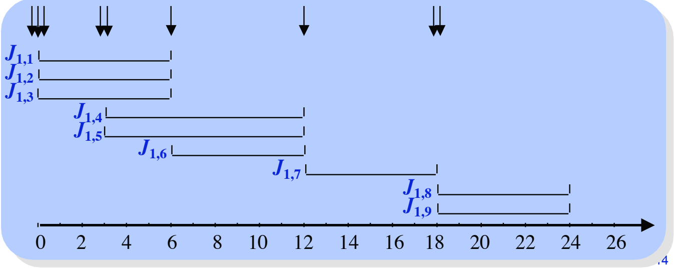
---

# The RBE Task Model
## Comparison of rate specifications

Rate specification ($x$=1, $y$=2, $d$=6)


$J_{1,1}$
$J_{1,2}$
$J_{1,3}$
$J_{1,4}$
$J_{1,5}$
$J_{1,6}$
$J_{1,7}$
$J_{1,8}$
$J_{1,9}$
0 2 4 6 8 10 12 14 16 18 20 22 24 26

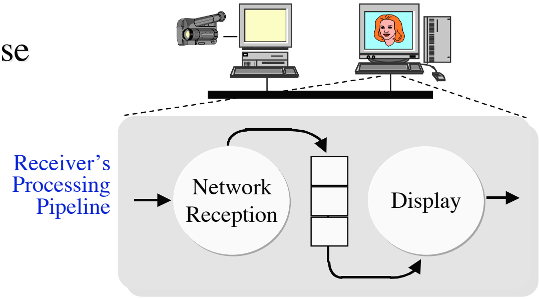
Rate specification ($x$=3, $y$=6, $d$=6)


$J_{1,1}$
$J_{1,2}$
$J_{1,3}$
$J_{1,4}$
$J_{1,5}$
$J_{1,6}$
$J_{1,7}$
$J_{1,8}$
$J_{1,9}$
0 2 4 6 8 10 12 14 16 18 20 22 24 26

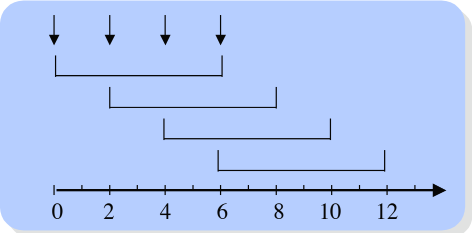
---

# The RBE Task Model
## RBE features/properties

- Provides better response time for non-real-time activities by integrating application-level buffering with the system run queue


Receiver's Processing Pipeline
Network Reception
Display

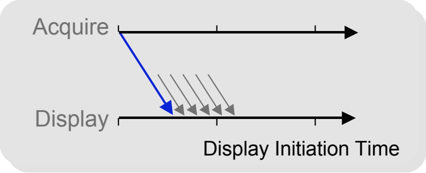
Rate specification ($x = 1$, $y = 2$, $d = 6$)



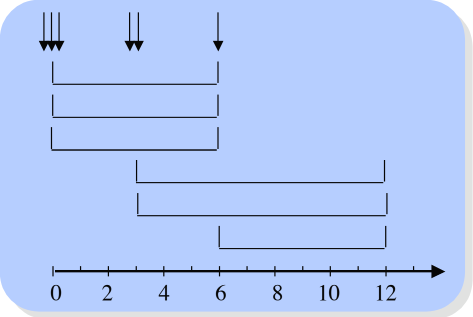
---

# The RBE Task Model
## RBE features/properties

- Provides a more natural way of modeling inbound packet processing of fragmented messages

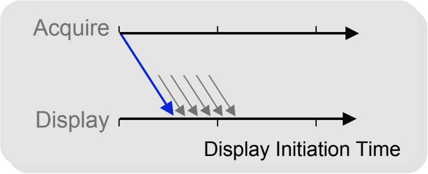

Acquire
Display
Display Initiation Time

Rate specification ($x = 3$, $y = 6$, $d = 6$)

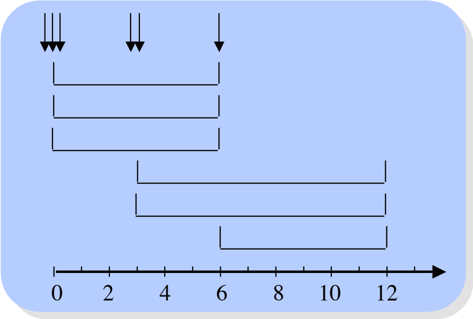


# The RBE Task Model
## RBE features/properties

- Provides isolation from arrival rates that exceed the rate specification
  - (But does not provide isolation from tasks exceeding their stated execution time)



Rate specification
($x = 3$, $y = 6$, $d = 6$)



---

# Fluid Flow Resource Allocation
## Proportional share resource allocation

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

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

- If tasks' weights remain constant in $[t_1, t_2]$ then task *i* receives

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

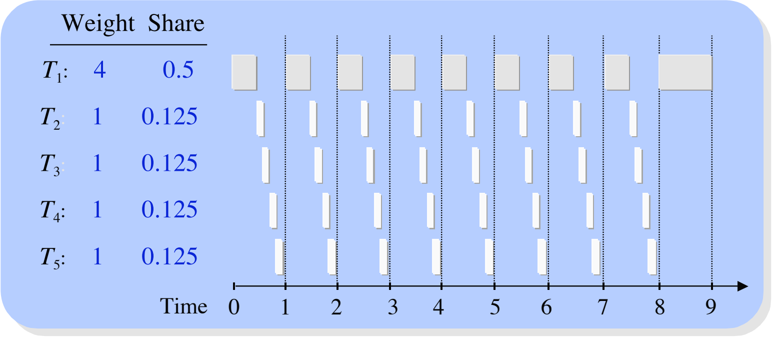
units of execution time in $[t_1, t_2]$

---

# Proportional Share Resource Allocation
## Fluid scheduling example

- Weighted round robin scheduling with an infinitesimally small quantum
- In $[t_1, t_2]$ (if total weight doesn't change) $T_i$ receives

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

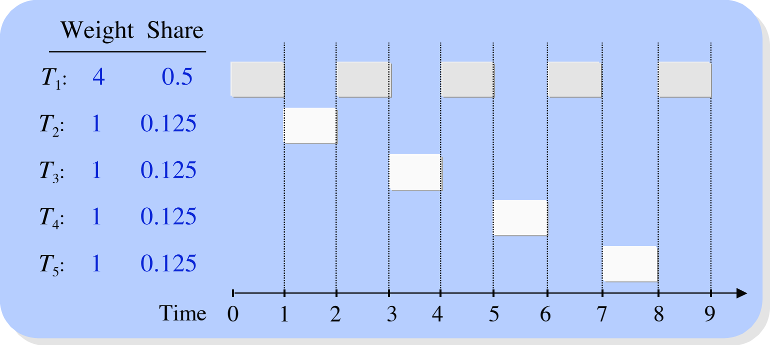


| | Weight | Share |
|---|---|---|
| $T_1$: | 4 | 0.5 |
| $T_2$ | 1 | 0.125 |
| $T_3$ | 1 | 0.125 |
| $T_4$: | 1 | 0.125 |
| $T_5$: | 1 | 0.125 |

---

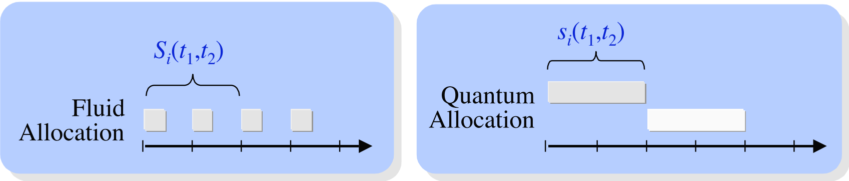
# Proportional Share Resource Allocation
## Quantum scheduling example

- Weighted round robin scheduling with integer quanta
  - $q = 1$
- The quantum system doesn't proportionally allocate the resource over all time intervals



| | Weight | Share |
|---|---|---|
| $T_1$: | 4 | 0.5 |
| $T_2$: | 1 | 0.125 |
| $T_3$: | 1 | 0.125 |
| $T_4$: | 1 | 0.125 |
| $T_5$: | 1 | 0.125 |

# Proportional Share Resource Allocation
## Task scheduling metrics & goals


$S_i(t_1,t_2)$
Fluid Allocation
$s_i(t_1,t_2)$
Quantum Allocation

- Schedule tasks so that their performance is as close as possible to that in the *fluid* system
- Why is fluid allocation important?
  - What about real-time allocation?!

---

# Approximating Fluid Allocation
## Why is this so important?

- Fluid allocation implies real-time progress
- Weights are used to allocate a *relative* fraction of the CPU's capacity to a task

$$f_i(t) = \frac{w_i}{\Sigma_j w_j}$$

- Real-time progress requires a *constant* fraction of the CPU's capacity

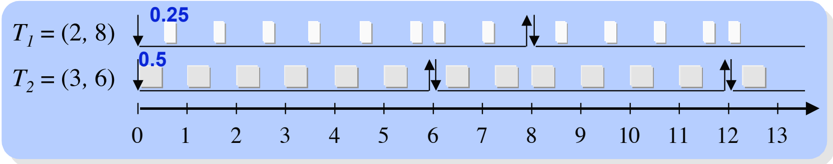
$$\forall t,\ f_i(t) = \textit{execution cost}_i \times \textit{execution frequency}_i$$

  - If a task must execute for 16 *ms* every 33 *ms* then allocating $f = 0.5$ ensures real-time execution
- Thus real-time performance can be achieved by adjusting weights dynamically so that the share remains constant

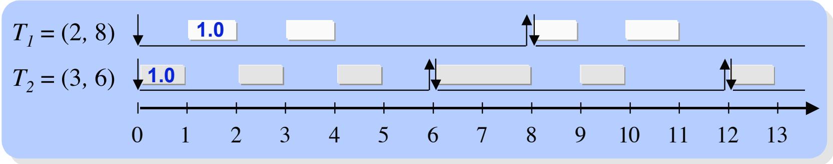
---

# Proportional Share Resource Allocation
## Real-time scheduling example

- Periodic tasks allocated a share equal to their processor utilization
  - Round-robin scheduling with infinitesimally small quantum

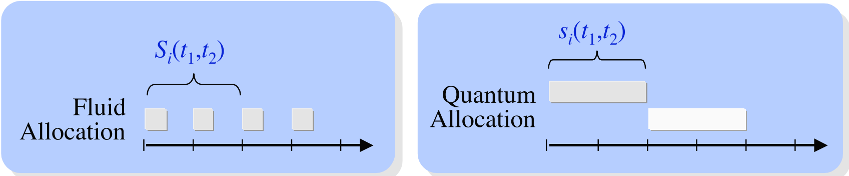

$T_1 = (2, 8)$
0.25
$T_2 = (3, 6)$
0.5
0
1
2
3
4
5
6
7
8
9
10
11
12
13

  - With unit-sized quantum


$T_1 = (2, 8)$
1.0
$T_2 = (3, 6)$
1.0
0
1
2
3
4
5
6
7
8
9
10
11
12
13

---

# Proportional Share Resource Allocation
## Task scheduling metrics & goals


$S_i(t_1,t_2)$
Fluid Allocation
$s_i(t_1,t_2)$
Quantum Allocation

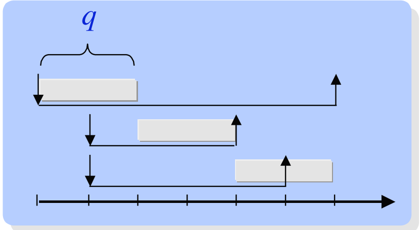
- Goal: Schedule tasks so that their performance is as close as possible to that in the *fluid* system
- Define the allocation error for task *i* at time *t* as

$$lag_i(t) = \left(\begin{array}{c}\textit{allocation the task would have}\\ \textit{received in the fluid system}\end{array}\right) - \left(\begin{array}{c}\textit{allocation the task has received}\\ \textit{in the quantum system}\end{array}\right)$$
$$= S_i(t_i,t) - s_i(t_i,t)$$

- Schedule tasks so that the lag is bounded for all tasks over all time intervals
  - What is the least upper bound on lag?

# Proportional Share Resource Allocation

## Timing analysis



- Is a task guaranteed to complete before its deadline?
  - How late can a task be?
- Theorem: *Let c be the size of the current request of task T. Task T's lag is bounded by*

$$-q < lag_T(t) < q$$

---

# Rate-Based Resource Allocation

## FreeBSD implementation

- We've implemented RBE and proportional share scheduling in FreeBSD
- Goal: Provide integrated real-time computation and communication services in a time-shared operating system
- Technical challenge: Scheduling OS services

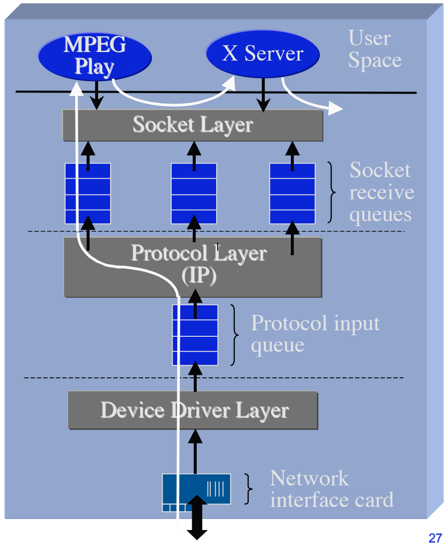
XEmacs
elcome!
WWW.STARWARS.COM

---

# 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

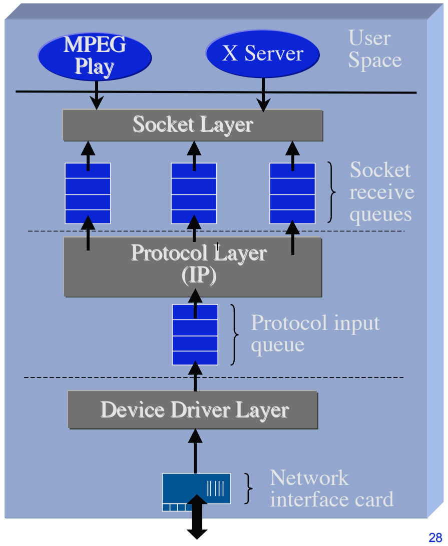

MPEG Play
X Server
User Space
Socket Layer
Socket receive queues
Protocol Layer (IP)
Protocol input queue
Device Driver Layer
Network interface card

---

# Rate-Based Resource Allocation

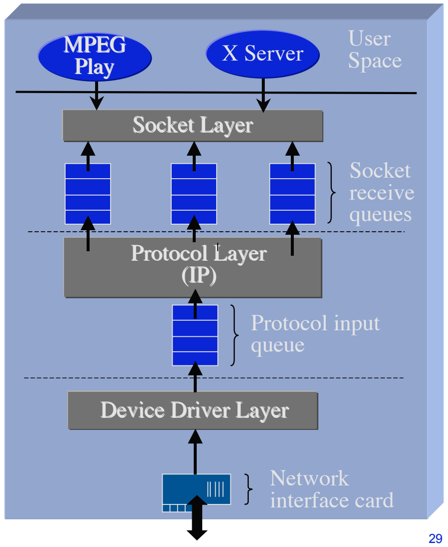
## Integrated real-time resource allocation example

- 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)


MPEG Play
X Server
User Space
Socket Layer
Socket receive queues
Protocol Layer (IP)
Protocol input queue
Device Driver Layer
Network interface card

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

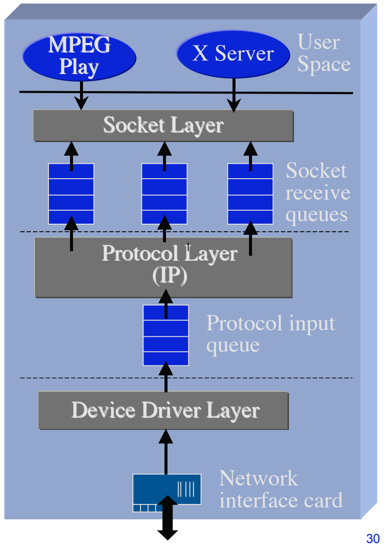
- Our study:
  - Compare the performance of applications of rate-based scheduling technology at various levels in the kernel
  - For various characterizations of real-time processing workloads
    - » Well-behaved periodic job/task arrivals
    - » Bursty job/task arrivals
    - » “Misbehaved” job/task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


MPEG Play
X Server
User Space
Socket Layer
Socket receive queues
Protocol Layer (IP)
Protocol input queue
Device Driver Layer
Network interface card

---

# Empirical Comparisons
## Experimental setup

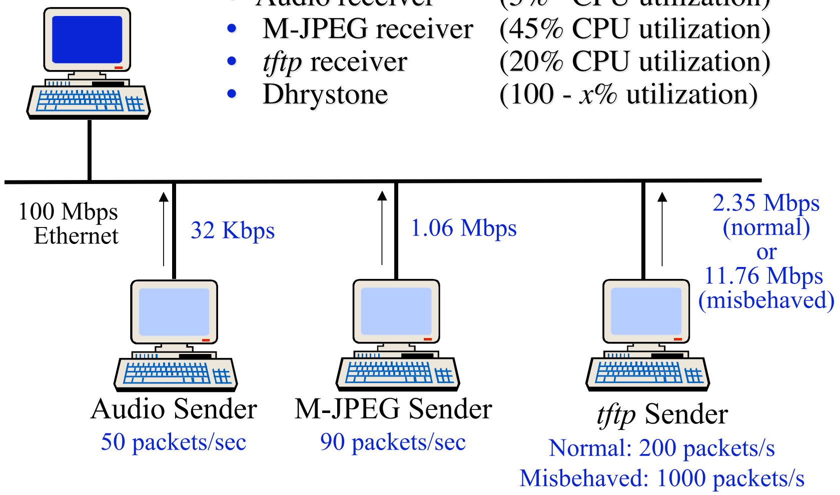
- Modify FreeBSD UNIX to support rate-based scheduling in the “top” and “bottom” halves of the kernel
- Consider the performance of each rate-based scheme in isolation and in combinations
  - Consider the performance across a variety of multimedia workloads


MPEG Play
X Server
User Space
Socket Layer
Socket receive queues
Protocol Layer (IP)
Protocol input queue
Device Driver Layer
Network interface card

---

# Experimental Setup
## Workload generation

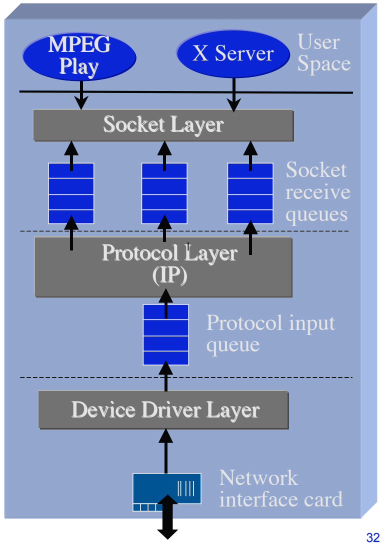
- Audio receiver (5% CPU utilization)
- M-JPEG receiver (45% CPU utilization)
- *tftp* receiver (20% CPU utilization)
- Dhrystone (100 - *x*% utilization)


100 Mbps Ethernet
32 Kbps
1.06 Mbps
2.35 Mbps (normal) or 11.76 Mbps (misbehaved)
Audio Sender
50 packets/sec
M-JPEG Sender
90 packets/sec
*tftp* Sender
Normal: 200 packets/s
Misbehaved: 1000 packets/s

---

# Empirical Comparisons
## Performance metrics setup

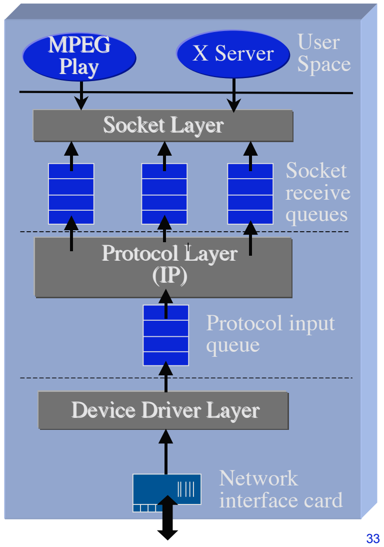
- Packets dropped at the IP layer
- Packets dropped at the socket layer
- Packets delivered to the application
- Dhrystone performance
- NIC to application response time
- Deadline miss percentage


MPEG Play
X Server
User Space
Socket Layer
Socket receive queues
Protocol Layer (IP)
Protocol input queue
Device Driver Layer
Network interface card

# 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

# Initial 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

---

# Rate-Based Resource Allocation
## Summary

- There’s life beyond rate monotonic scheduling
- Rate-based resource allocation simplifies systems wherein
  - Work is generated at non-periodic but structured rates
  - Tasks may “misbehave”
- Liu and Layland extensions
  - Rate models demonstrate a fundamental distinction between static priority and deadline scheduling methods
- Fluid flow models
  - Real-time ±*quantum*
  - No fundamental distinction between real-time and non-real-time tasks
  - Provide strict isolation between tasks