Adaptive Real-Time Resource Management

Richard West

Boston University

Computer Science Department

Outline of Talk

- Problem Statement.
  - How to guarantee QoS to applications?
  - Variable resource demands / availability.
- Approach.
  - System mechanisms.
    - Dionisys.
  - System policies.
    - Dynamic Window-Constrained Scheduling.
- Conclusions.

Problem Statement

- Distributed, real-time (RT) applications:
  - e.g., VE, RT multimedia, tele-medicine, ATR.
  - Require QoS guarantees on end-to-end transfer of information.
- How do we guarantee QoS?
- Need system support to **maintain / maximize QoS**:
  - Policies & mechanisms.
  - **Adaptive** / **coordinated** resource management.

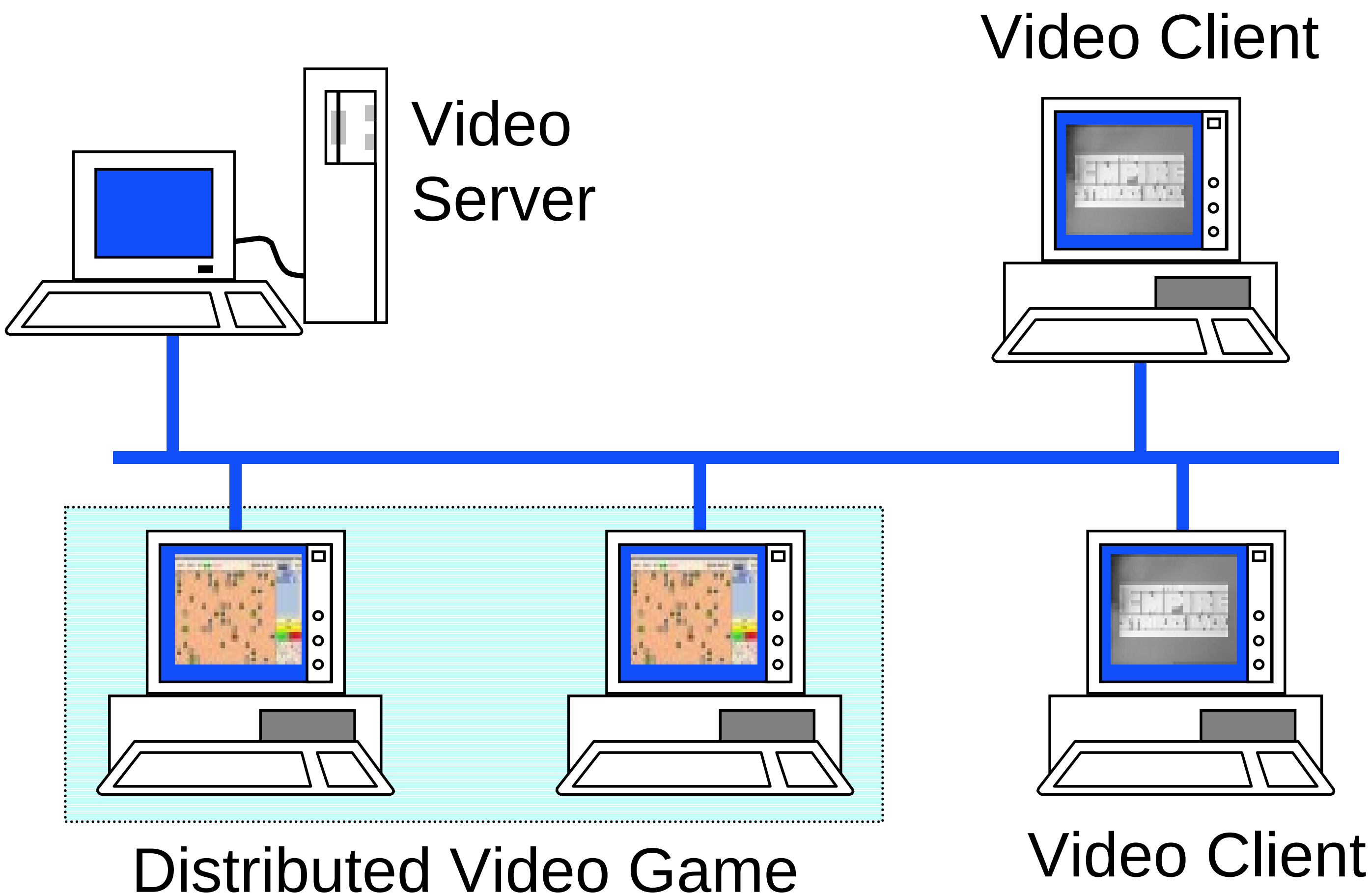
Application Characteristics

- Dynamic exchanges between processes.
  - The information (content & type) to be exchanged changes with time.
- Variable rates (bursts) of exchanges.
- Variable resource demands.
  - Bandwidth, CPU cycles, memory.
- Variable **QoS requirements** on information exchanged.

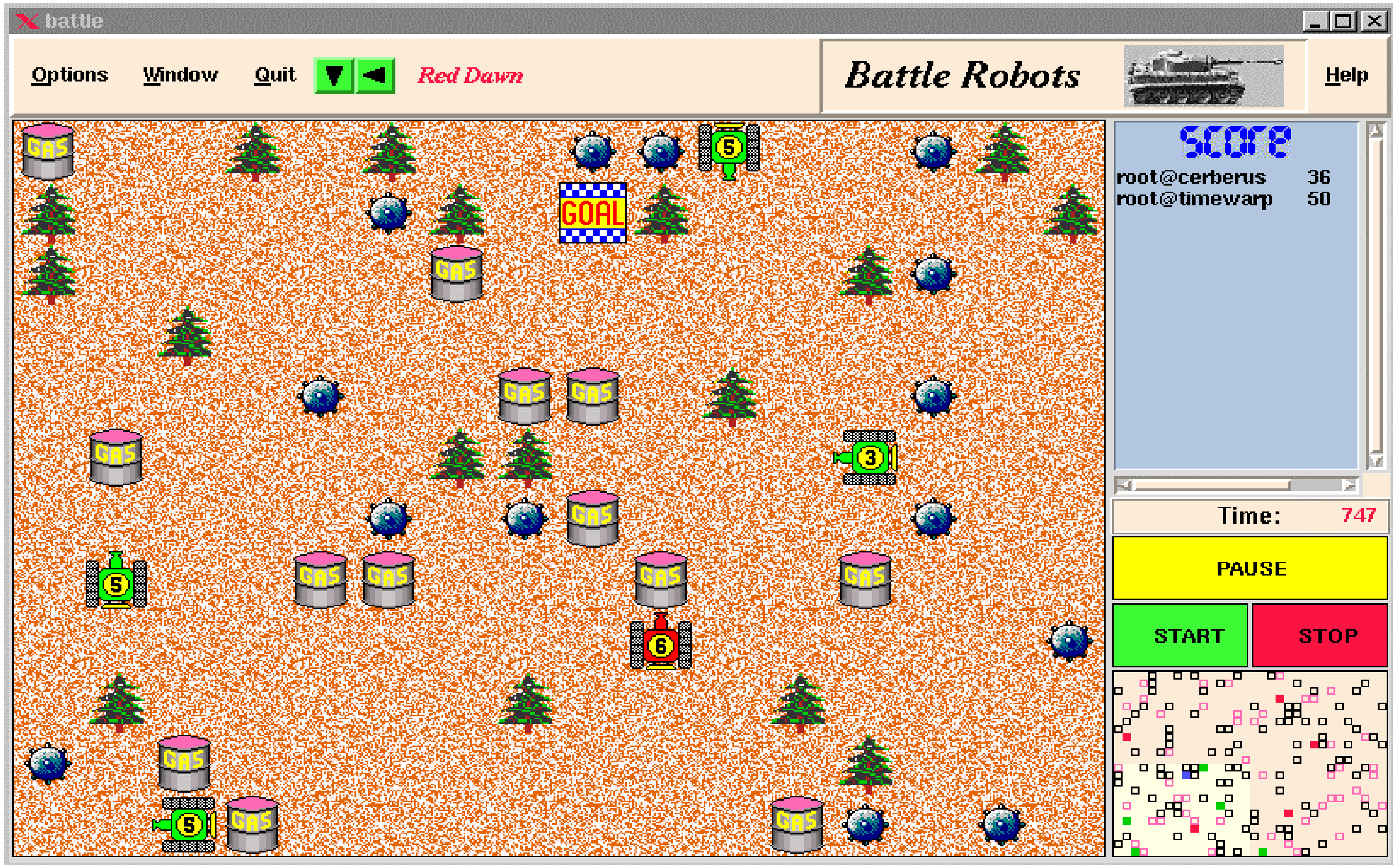
QoS Requirements

- **Delay**: e.g., max end-to-end delay, delay variation.
- **Loss-tolerance, fidelity, resolution**:
  - Minimum degree of detail.
- **Throughput, rate**:
  - e.g., 30 fps video.
  - e.g., min/max updates per second to shared data.
- **Consistency constraints**:
  - *When*, *with whom* semantics.

Example Scenario



Example: Distributed Video Game



BOSTON UNIVERSITY Rich West (2001)

Distributed Video Game

- Requires consistency of shared (tank) objects.
- Here **QoS (and, hence, resource) requirements vary** with time based on current state of application.

- Application-level **spatial** & **temporal** semantics.
  - Exchange state info only when two objects less than distance *d* apart.
  - Exchange position, orientation and (varying amounts of) graphical info about shared objects based on their distance apart.

Example: Video Server

- QoS requirements: Loss-tolerance and frame rate.
- Suppose a client requires at least 15fps playback rate but prefers 30fps.
- If network bandwidth is limited:
  - **Adapt CPU service.**
    - e.g. allocate more CPU cycles to compress video info.
  - **Adapt network service.**
    - e.g. allow 1 frame in 2 to be dropped.

Video Server (continued)

- If CPU cycles are limited:
  - **Adapt CPU service**.
    - If possible, reduce frame generation rate.
  - **Adapt network service**.
    - e.g. ensure no frames are now dropped.
- If CPU and network resources are limited:
  - Adapt to new QoS region / requirements if possible! Re-negotiation?

Summary of Problem

- Need to maintain / maximize QoS on end-to-end transfer of information.
- **Varying** resource requirements & availability.
- **Static** resource allocation too expensive.
  - Poor resource utilization & scalability.

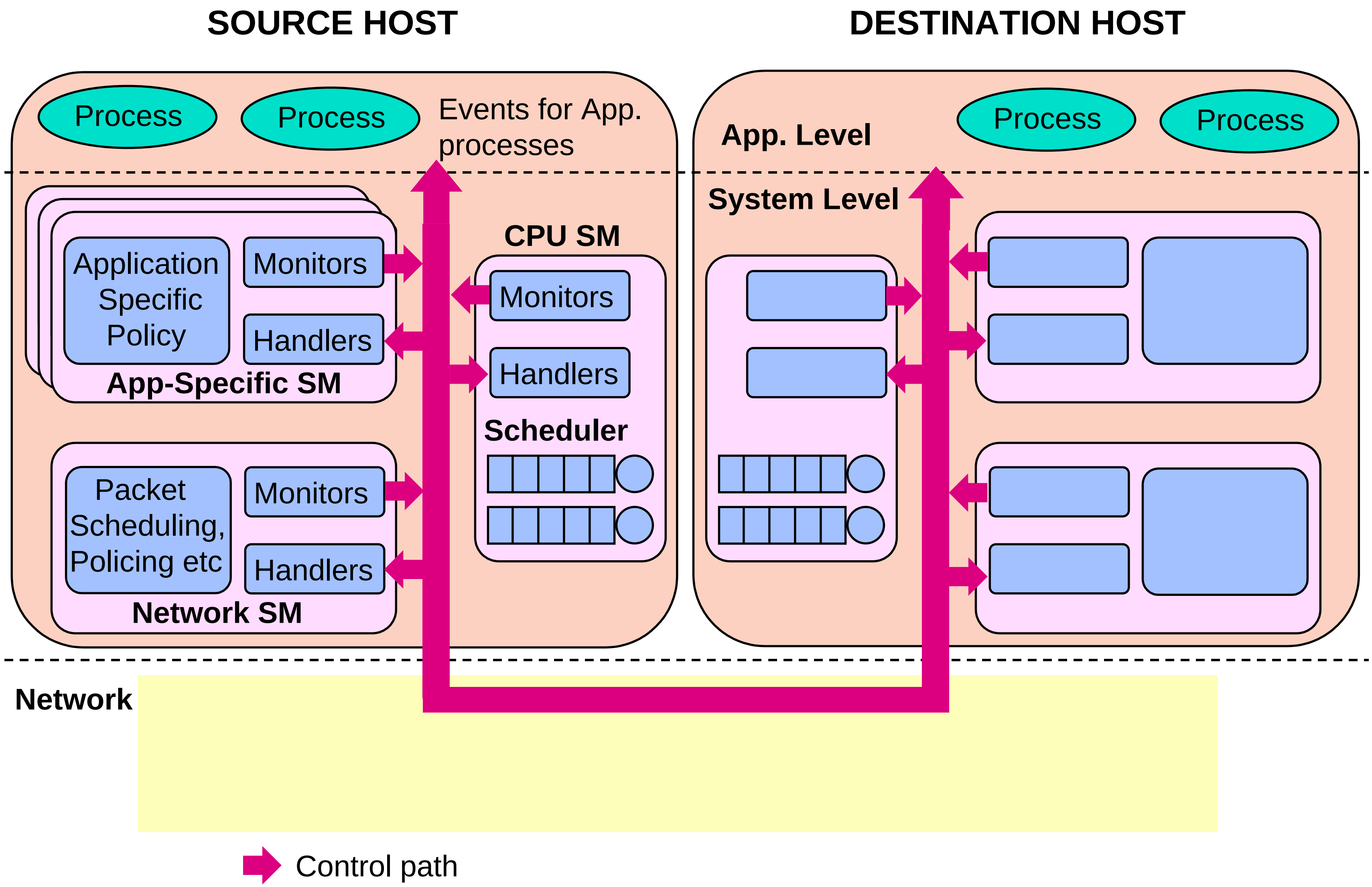
- Suppose enough resources are reserved to meet the minimum needs of all applications.
  - How can we do better?

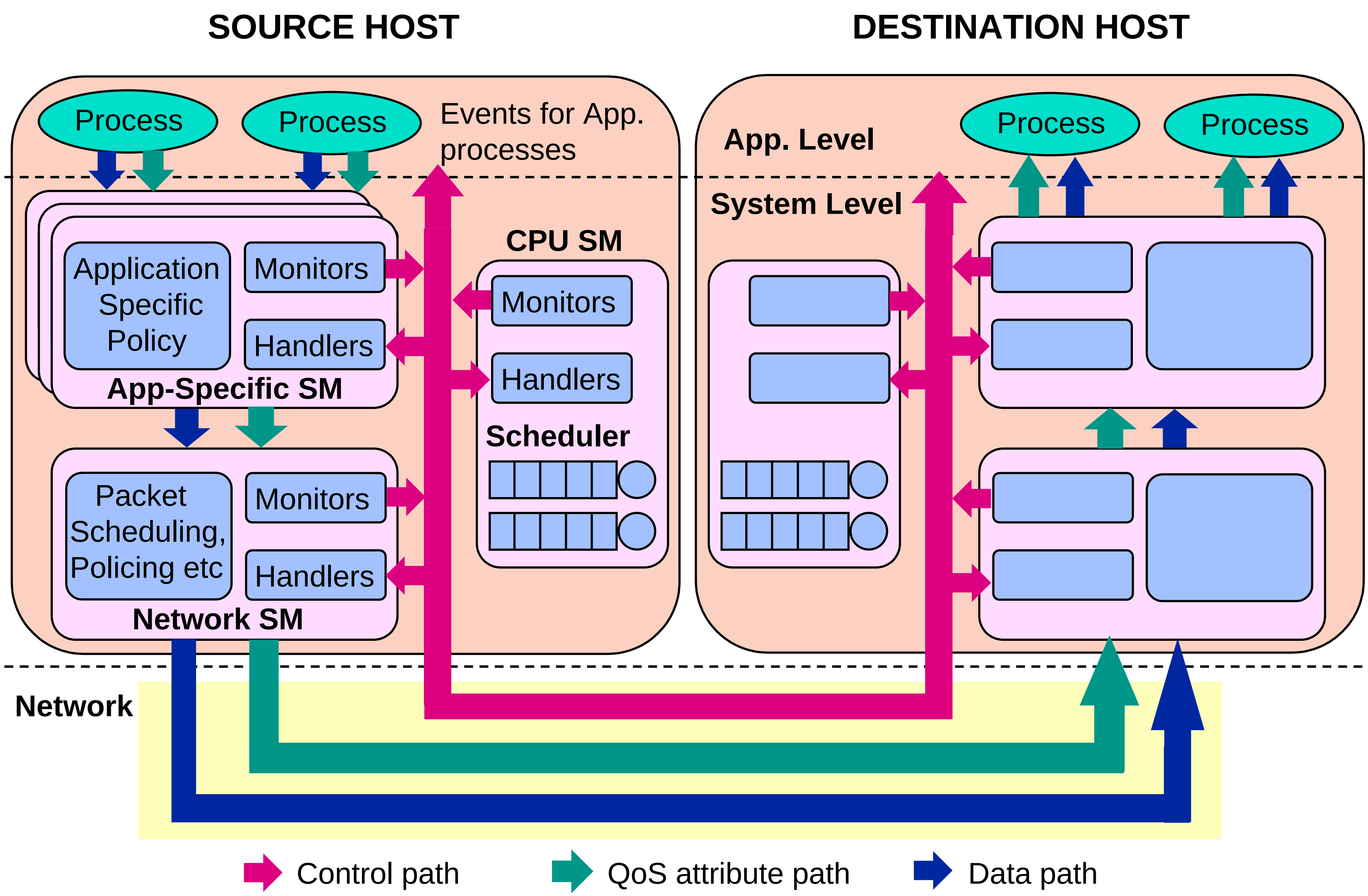
Approach

- **Dionisys** QoS mechanisms.
  - Allow real-time applications to specify:
    - How actual service should be adapted to meet required / improved QoS.
    - When and where adaptations should occur.
- **Coordinated** CPU and network management.
  - **Dynamic Window-Constrained Scheduling.**

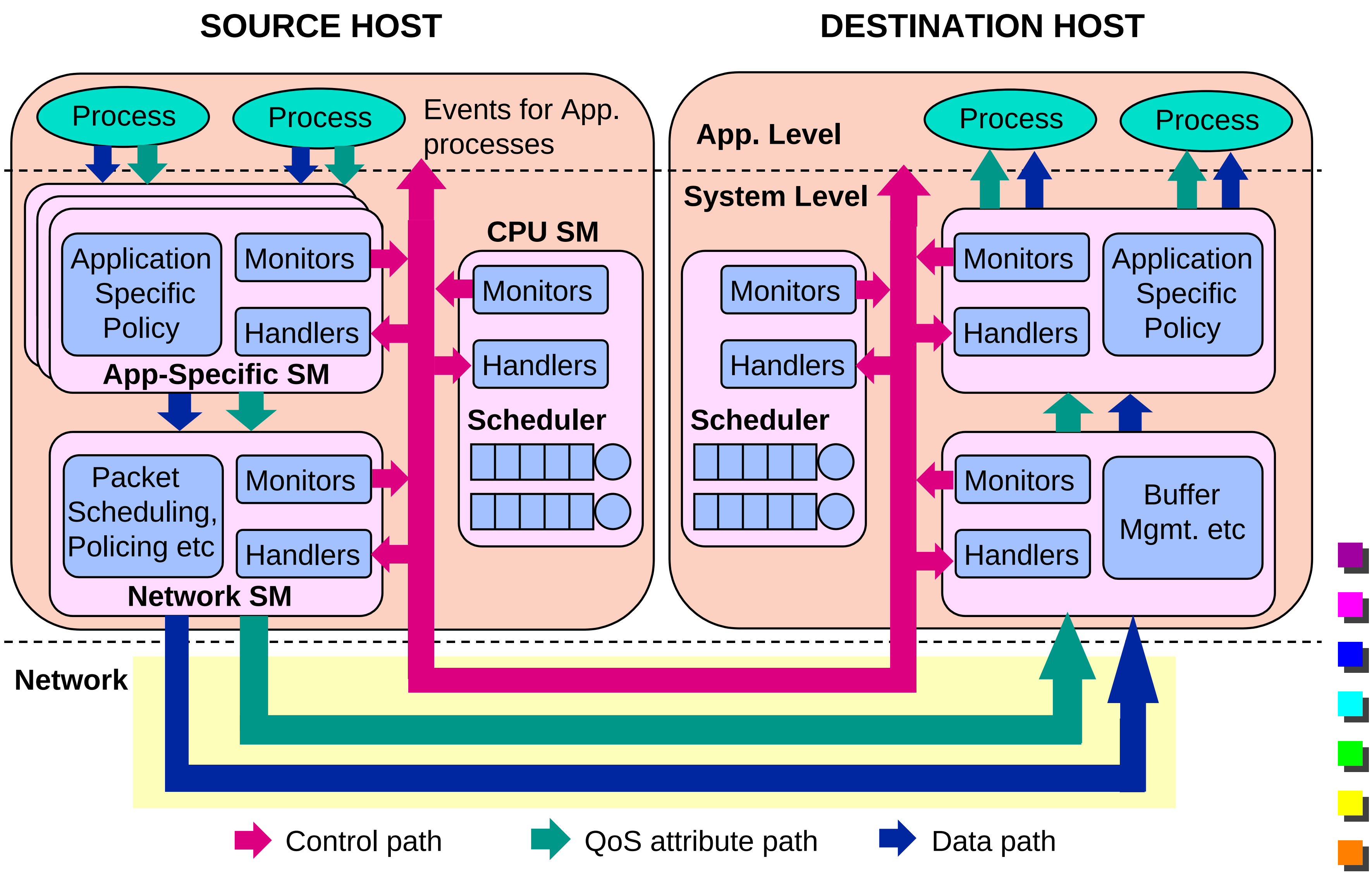
Dionisys

- Key components:
  - Service managers (SMs).
  - Monitors - influence <u>when</u> to adapt.
  - Handlers - influence <u>how</u> to adapt.
  - Events.
    - Delivered to SMs, <u>where</u> adaptation is needed.
  - Event channels.


SOURCE HOST
DESTINATION HOST
Process
Process
Events for App. processes
Application Specific Policy
Monitors
Handlers
App-Specific SM
CPU SM
Monitors
Handlers
Scheduler
Packet Scheduling, Policing etc
Monitors
Handlers
Network SM
App. Level
Process
Process
System Level
Network
Control path



Rich West (2001)



Rich West (2001)

Dionisys Key

- Process — Application process.
- Event channel.
- QoS attribute channel (shared memory on a single host).
- Data channel.
- Service Manager (SM) e.g., CPU SM.
- SM functions: App-specific monitors, handlers and service policy.
- Host machine.

Service Managers

- Responsible for:
  - Monitoring application-specific service.
  - Handling events for service adaptation.
  - Providing service to applications.
    - Resource allocation.
- Kernel level threads.

Monitors

- Functions that monitor a specific service.
- Influence when to adapt service provided to an application.
  - e.g., QoS below desired level, or unacceptable.
- Compiled into objects.
  - Dynamically-linked into target SM address-space.

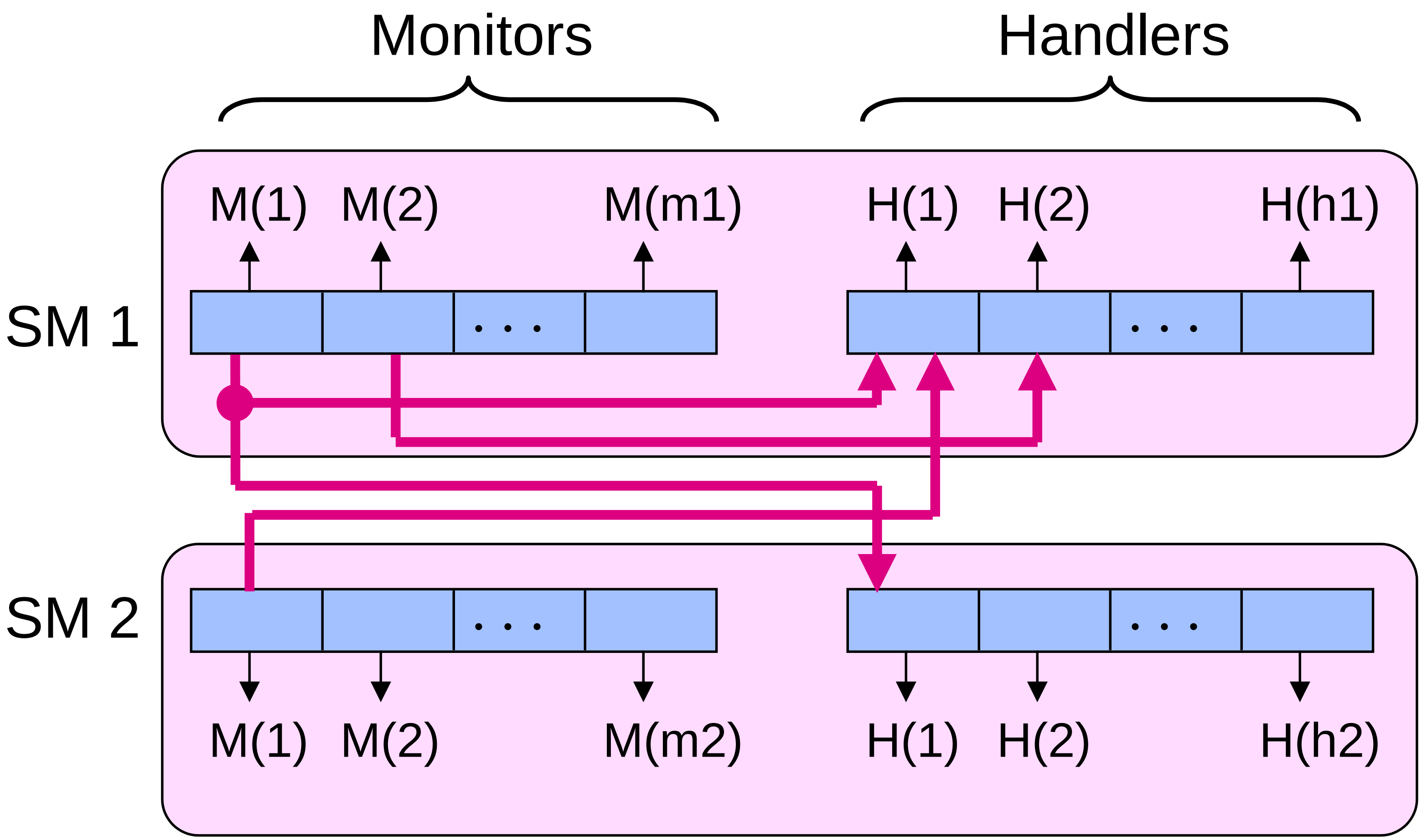
Handlers

- Functions executed in SMs to decide how to adapt service provided to an application.
  - e.g., increase / decrease CPU cycles, or network bandwidth.
- Compiled into objects.
  - Dynamically-linked into target SM address-space.

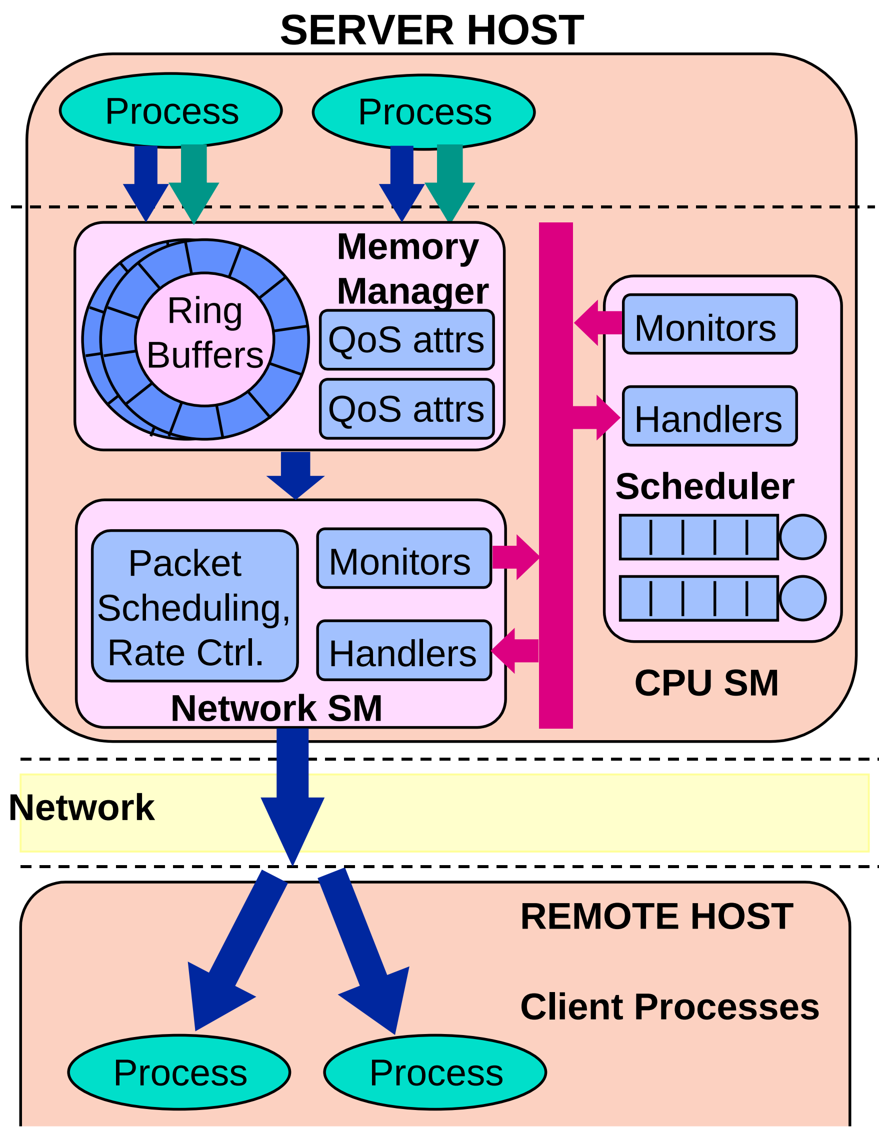
Events

- Generated when service adaptation is necessary.
- Delivered to handlers where service needs adapting.
- Have attributes that influence extent to which service is adapted.
  - “Quality Events”.

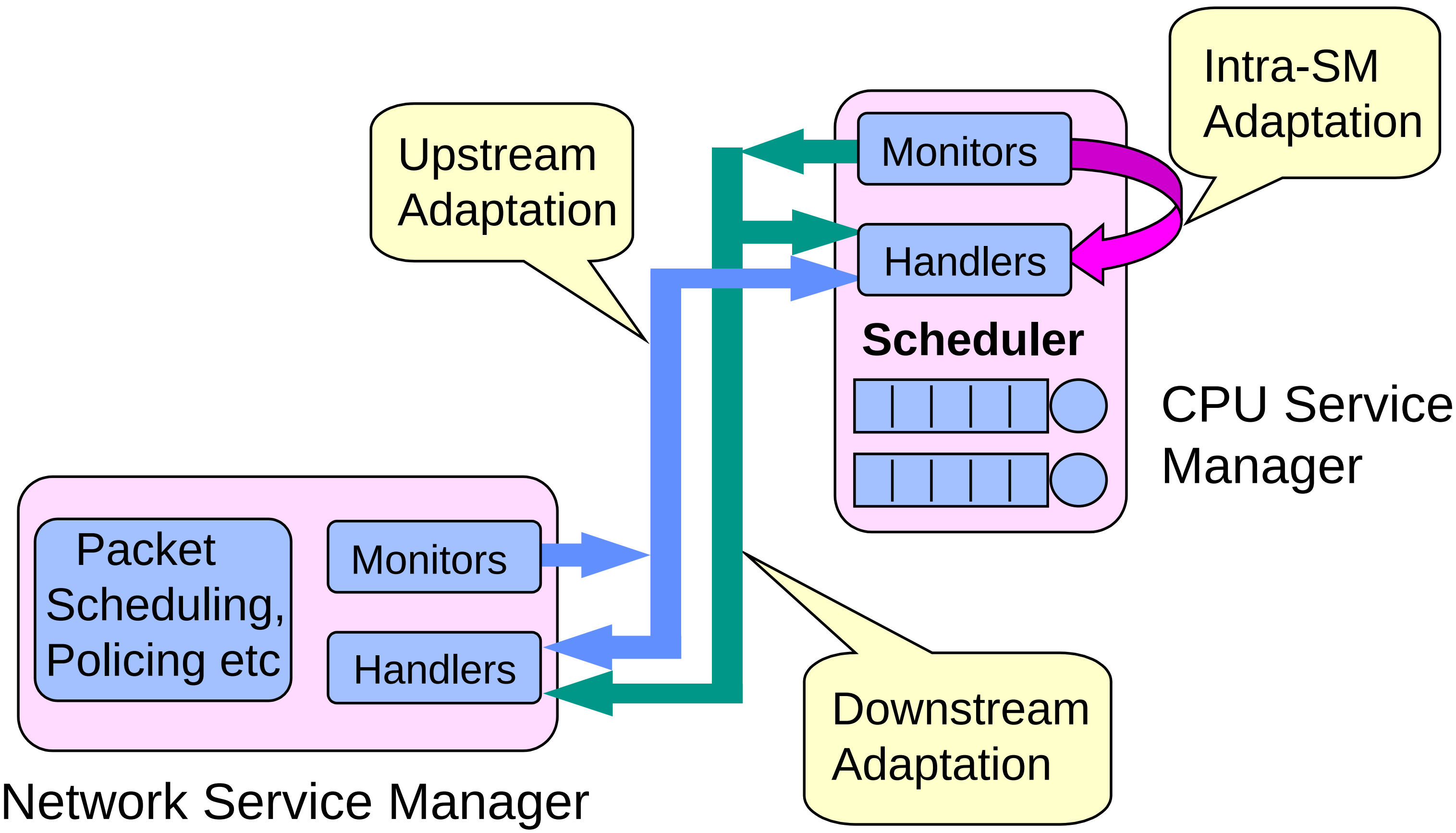
Event Channels



Example: Video Server



Example Adaptation Strategies



Adaptation Strategies (continued)

- **Upstream adaptation**:
  - Applied in direction opposing flow of data.
    - e.g. feedback congestion control.
- **Downstream adaptation**:
  - Applied in direction corresponding to flow of data.
    - e.g. forward error correction.
- **Intra-SM adaptation**:
  - Applied to current service manager.
  - Lacks coordination between SMs.

Adaptation Example: Video Server

- QoS requirements: Loss-tolerance and frame rate.
- If network bandwidth is limited:
  - Apply **upstream adaptation** to increase CPU cycles to e.g. compress video information.
  - Apply **intra-SM adaptation** in the network SM to increase loss-tolerance.

Adaptation Example (continued)

- If CPU cycles are limited:
  - Apply **intra-SM adaptation** in the CPU-SM to reduce, for example, frame (generation) rate.
  - Apply **downstream adaptation** to reduce loss-tolerance.

Experimental Scenario - Part 1

- Server-side processes (one per stream):
  - Generate data for streaming to remote clients.
    - Stream of MPEG-1 I-frames (160x120 pixels) per generator process.
    - Data placed in circular queues in shared memory.
- QoS attributes associated with each data stream:
  - Min / Max / Target frame rate.
- "Quality" event channels between Network and CPU service managers.

Experimental Scenario - Part 2

- Client-side processes (one per stream):
  - Decode and playback incoming frames.
- SparcStation Ultra-2 170Mhz dual processor server, running Solaris 2.6 connected via switched 100Mbps Ethernet to one client (w/ UDP connection).
- 3 Streams:
  - Stream 1: Target 30fps +/- 10% (3000 frames)
  - Stream 2: Target 20fps +/- 10% (2000 frames)
  - Stream 3: Target 10fps +/- 20% (1000 frames)
  - 3 second exponential idle time every 1000 frames.

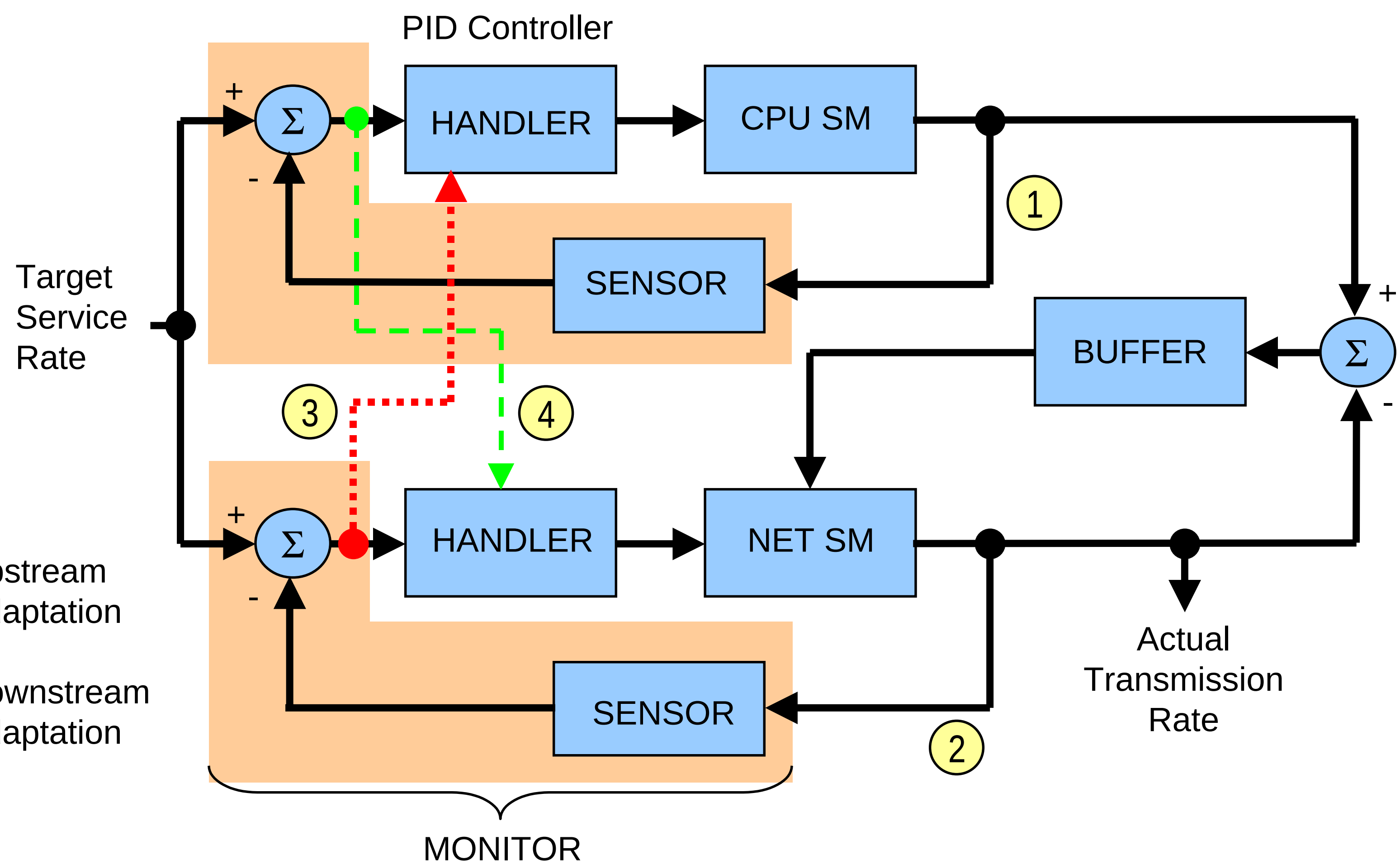
Adaptation in Video Server

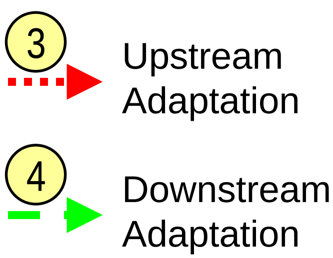
- (**Downstream**) CPU SM monitors frame generation rate.
- (**Upstream**) Net SM monitors frame transmission rate.
- Apply adaptation if (monitored rate != target rate).
- All monitors / SMs run at 10mS intervals.

Adaptation Handlers

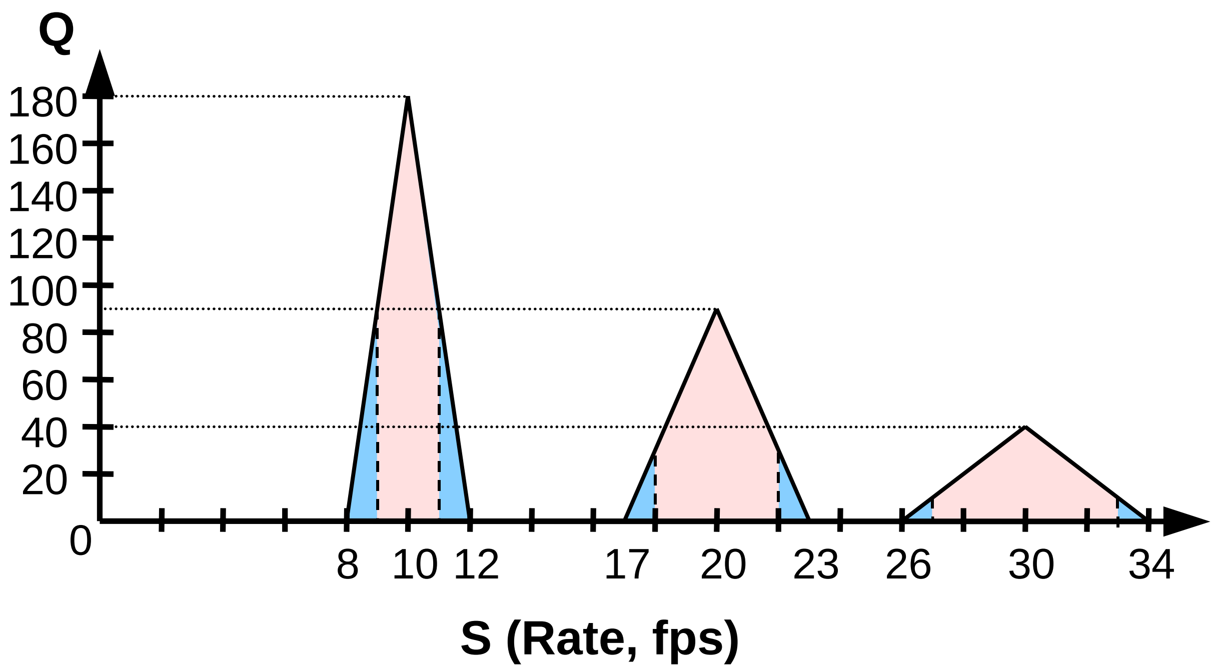
- CPU-Level:
  - Adjust priorities & time-slices of generator processes by a function of target and monitored service rates.
- Network-Level:
  - Invoke rate control if monitored rate exceeds maximum rate.
  - Raise priority of packet stream $\mathbf{S_i}$ if its service falls below minimum service rate.
    - i.e., alter bandwidth allocation $\mathbf{(y_i - x_i) / y_i}$.

Adaptive Rate Control Block Diagram



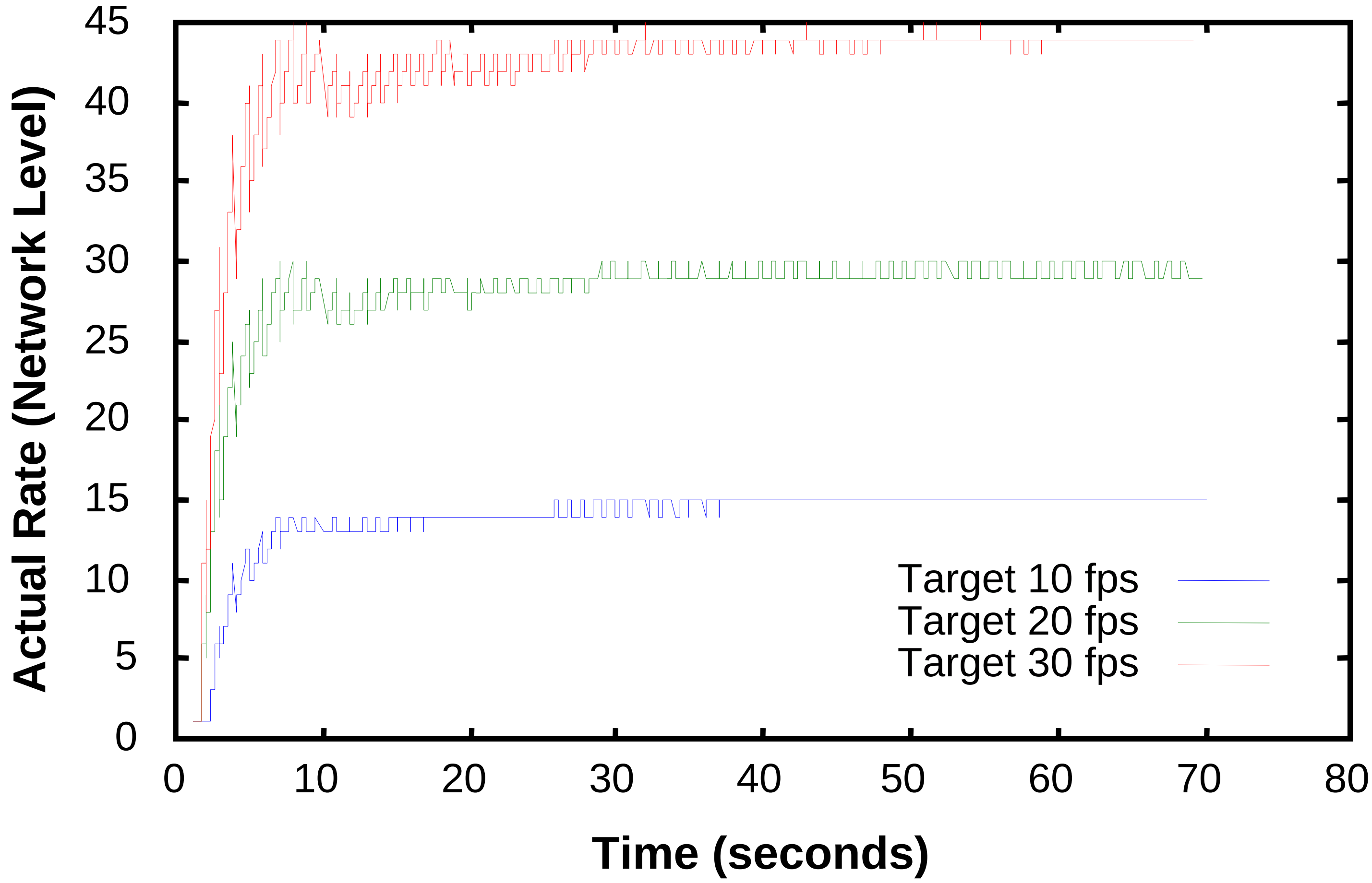


Quality Functions - Example

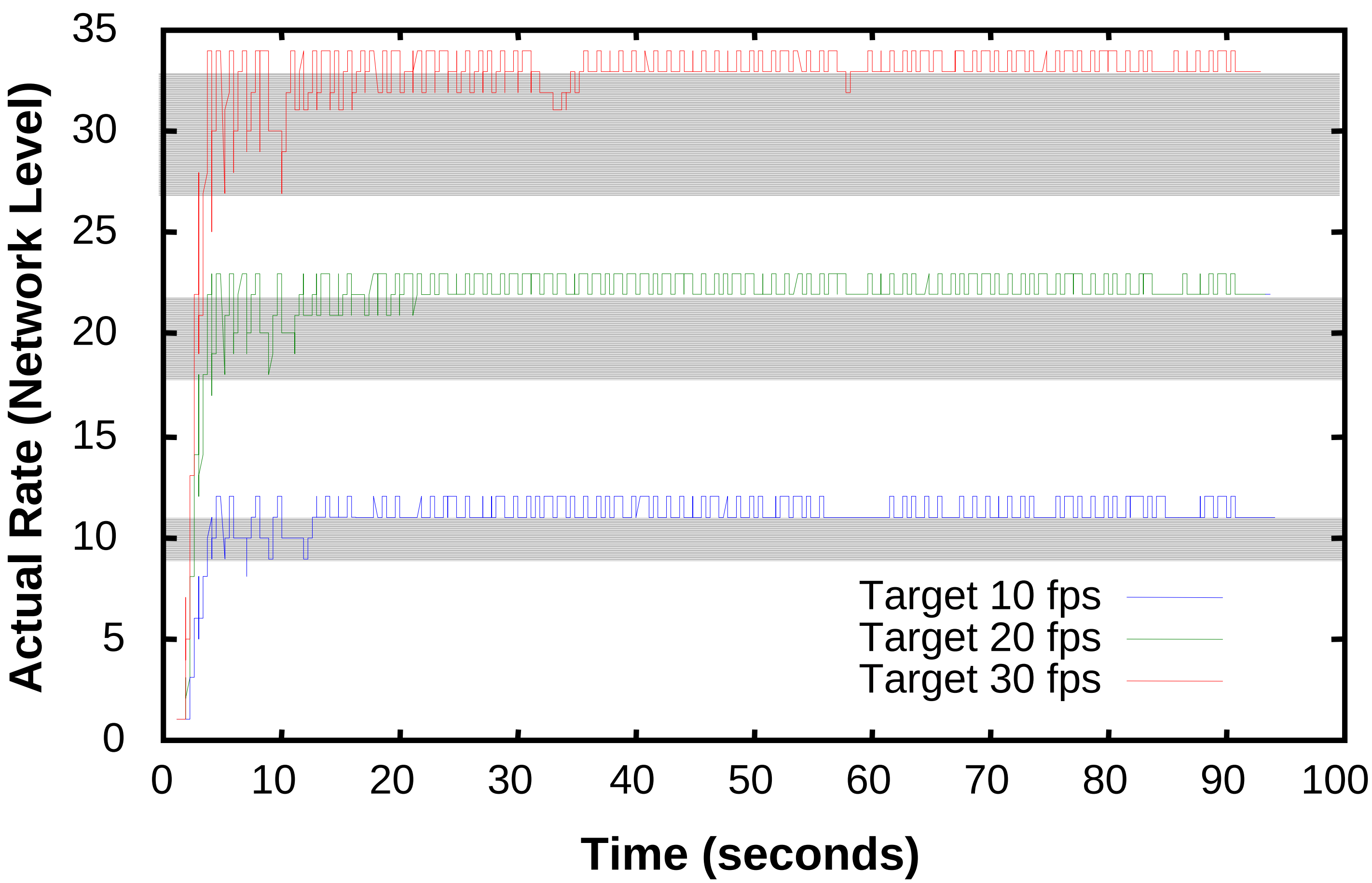


- Can embed quality functions into handlers.
- Service adaptation is a function of actual and required service of all applications.

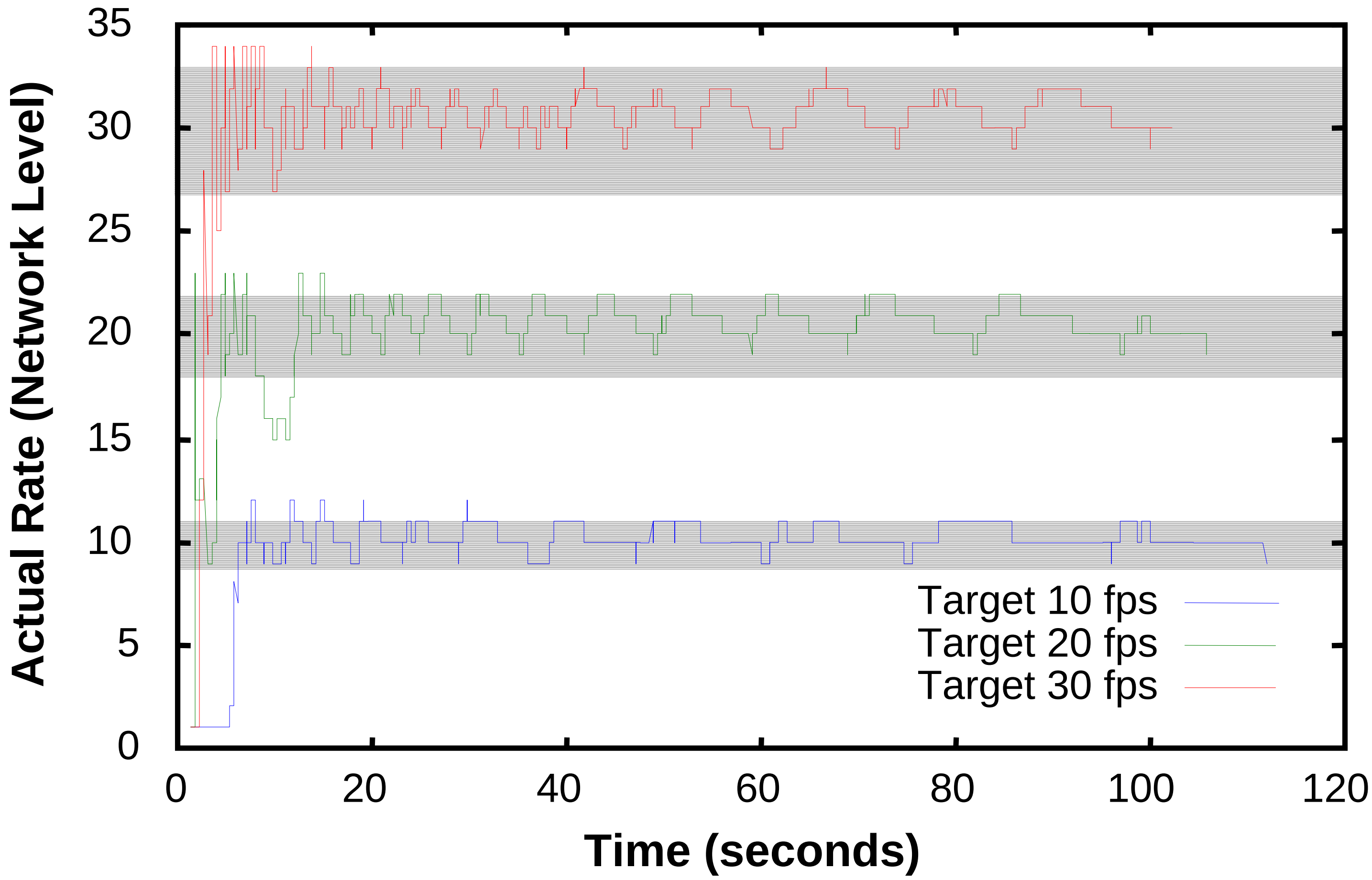
Non-Adaptive Rate Allocating Service



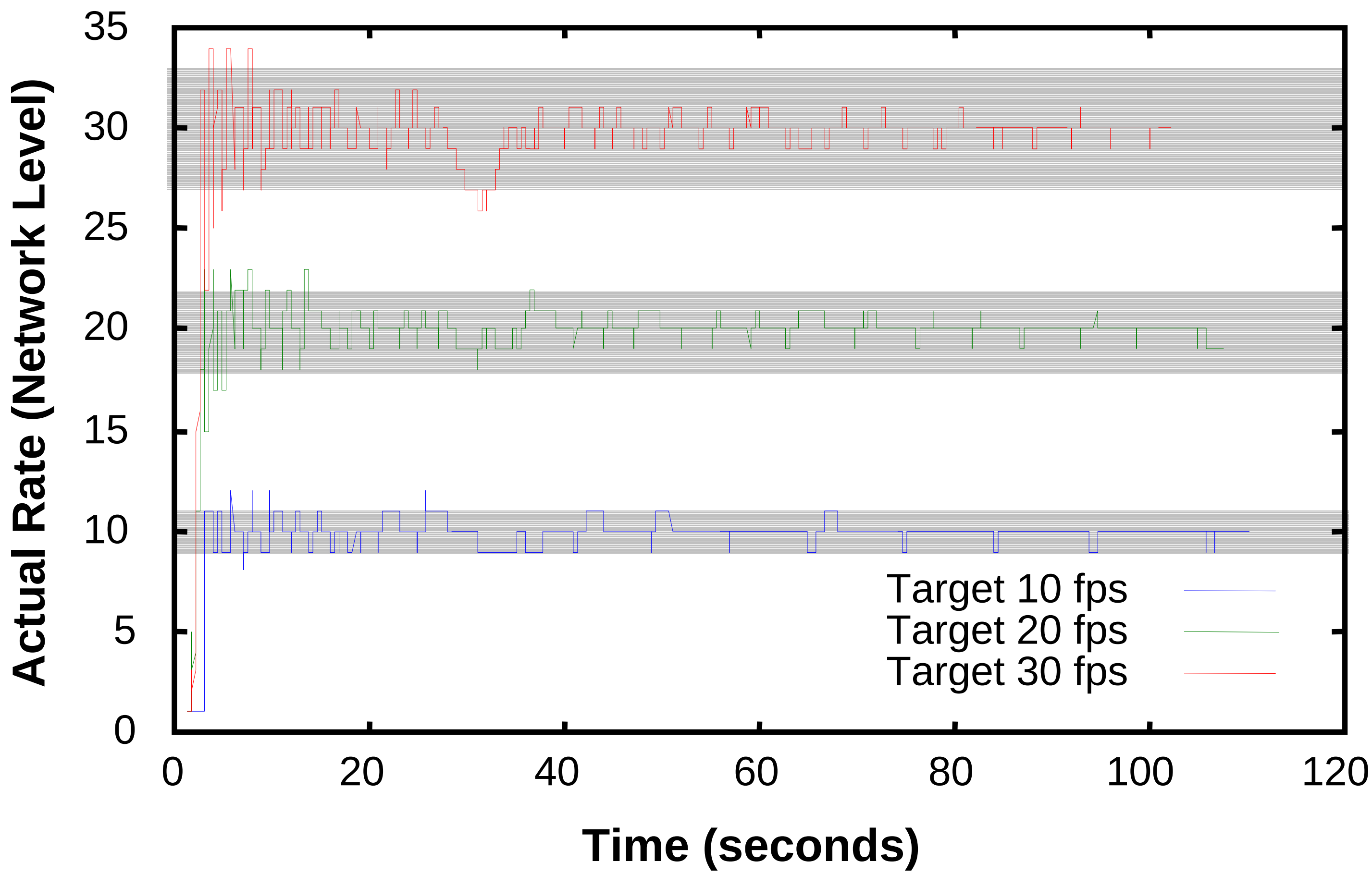
Non-Adaptive Rate Controlled Service



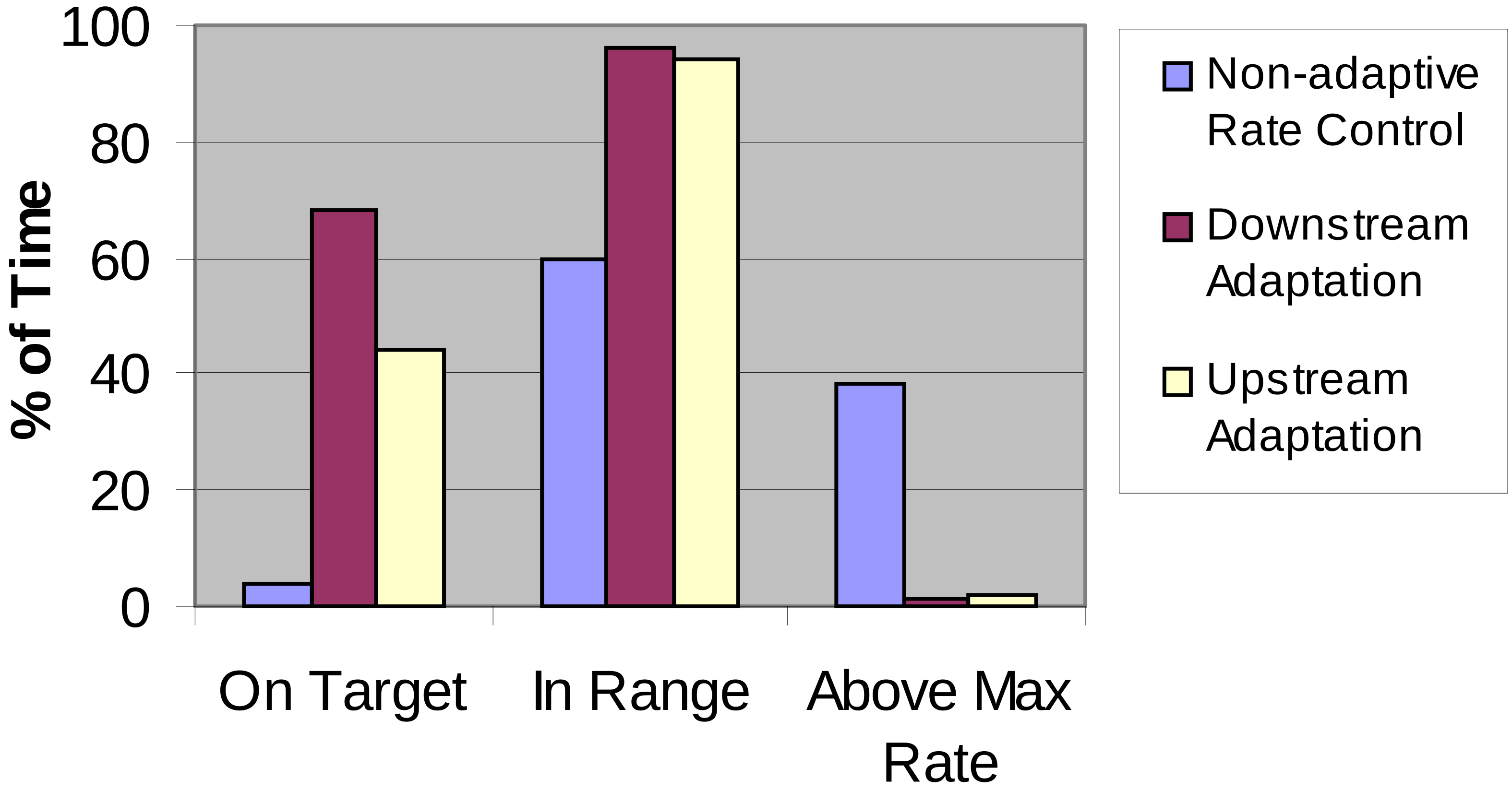
Network Rate - Upstream Adaptation



Network Rate - Downstream Adaptation



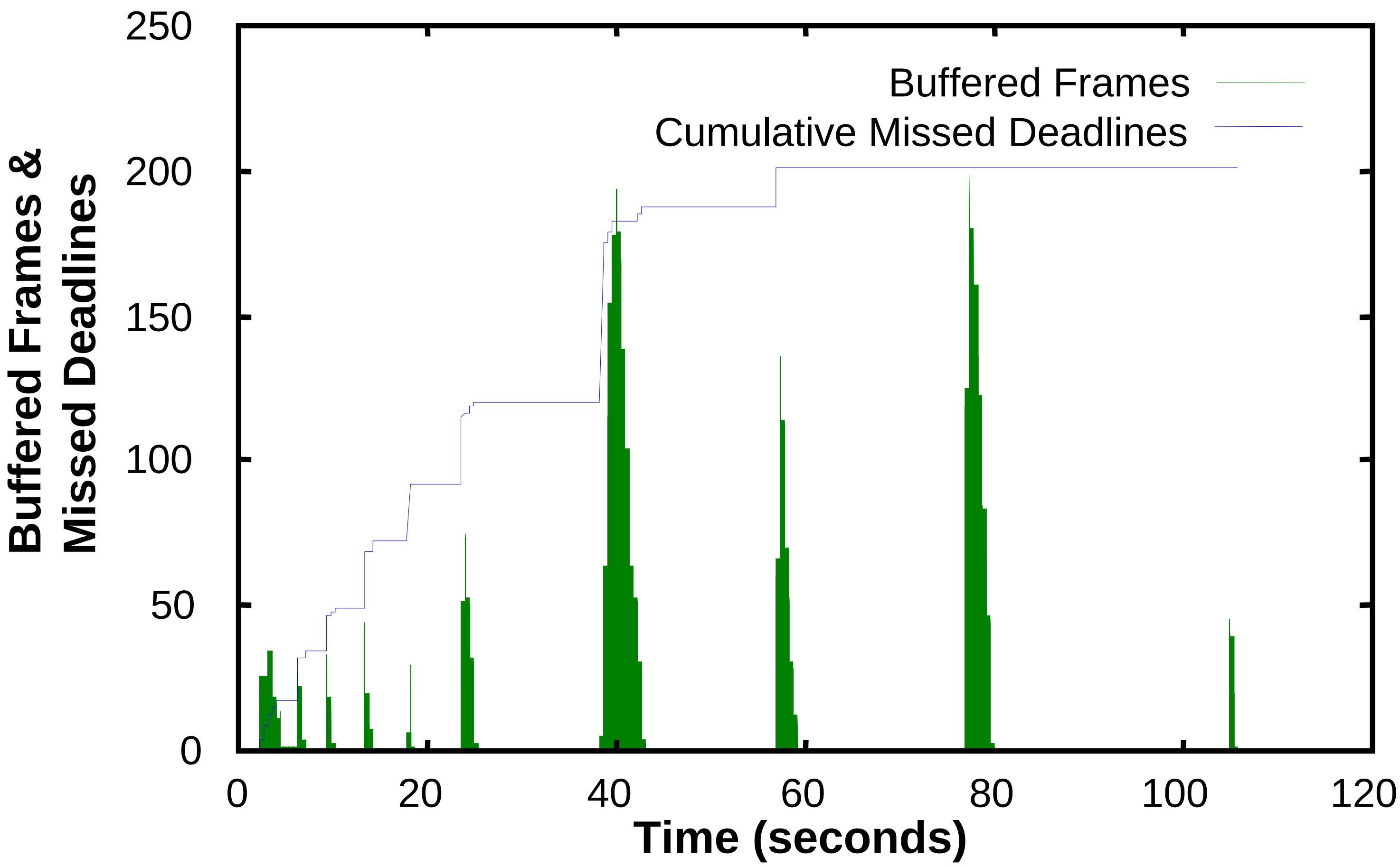
Comparison of Rate Control Methods



Rate Control

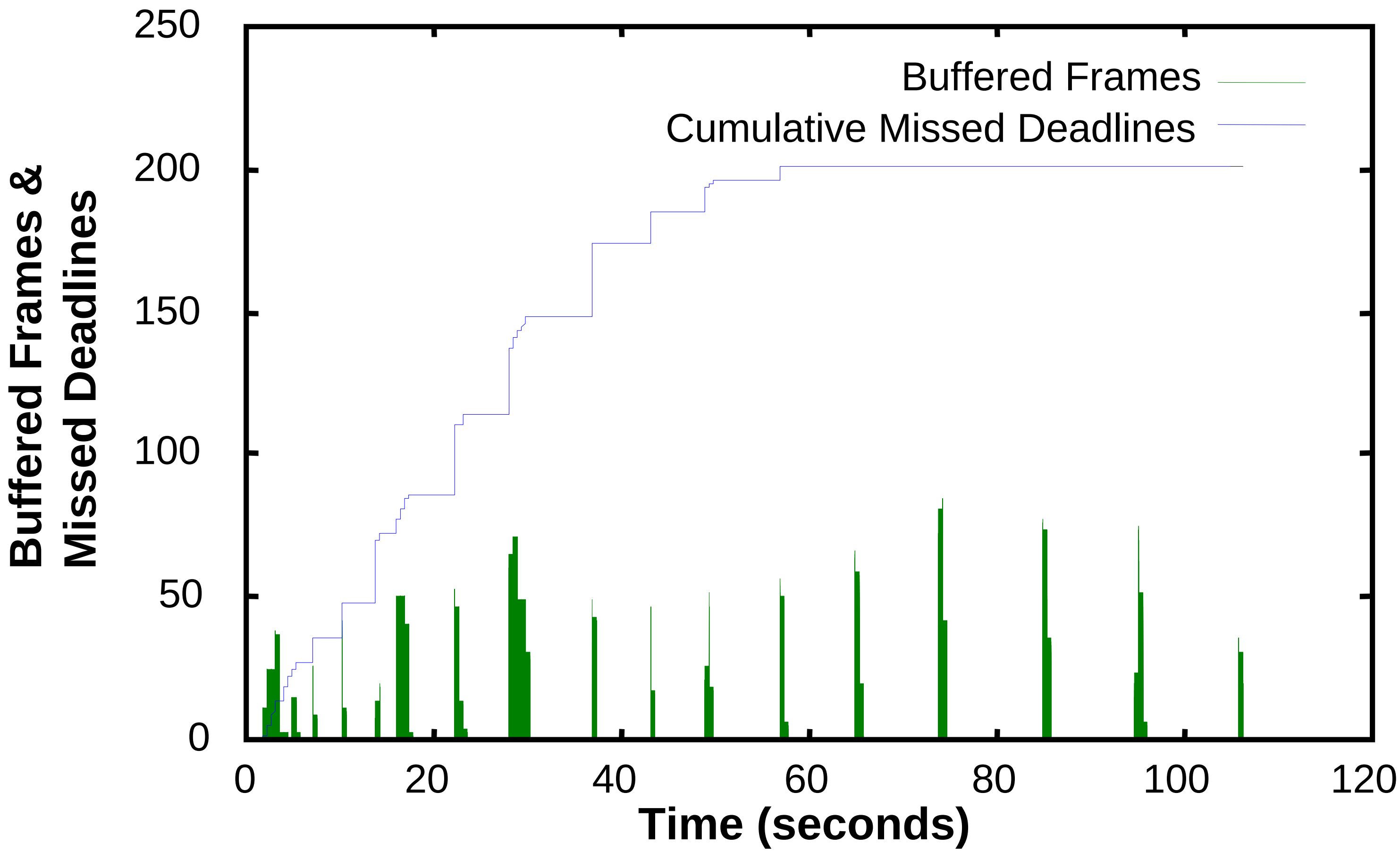
- Upstream adaptation leads to poorer rate control.
  - Longer time to reach steady state.
  - More prominent “sawtooth” effect as target rate is tracked.
  - Larger fluctuations of actual rate from target.
    - Better tracking of target rate for more quality critical streams.

Upstream Adaptation - 10fps



Rich West (2000)

Downstream Adaptation - 10fps



Rich West (2000)

Buffering

- Upstream adaptation leads to greater variance in buffer usage, compared to downstream / intra SM adaptation.
  - Network monitor triggers “request” for generation of frames “too late”. That is, after buffer has emptied.
  - Effect of an event being raised not seen until the next “phase” of monitoring and handling.

Missed Deadlines

- Higher buffering variance and, consequently, higher queueing delays, imply potentially higher consecutive numbers ("bursts") of missed deadlines.
- Downstream adaptation can reduce the number of consecutive deadlines missed at any time by:
  - Providing more accurate (responsive) service.
  - By effecting changes "more quickly" (in the current event/monitoring cycle) at the network-level to compensate for inadequacies in service at the CPU-level.

Summary

- **Dionisys** QoS mechanisms allow real-time applications to specify:
  - How actual service should be adapted to meet required / improved QoS.
  - When and where adaptations should occur.
- **Flexible** approach to run-time service adaptation.

What About Service Policies?

- Certain applications can tolerate lost / late information.
- Restrictions on:
  - when losses of info can occur.
  - when info must be generated.
- Need real-time scheduling of:
  - threads / processes (info generators).
  - packets (info carriers).

DWCS

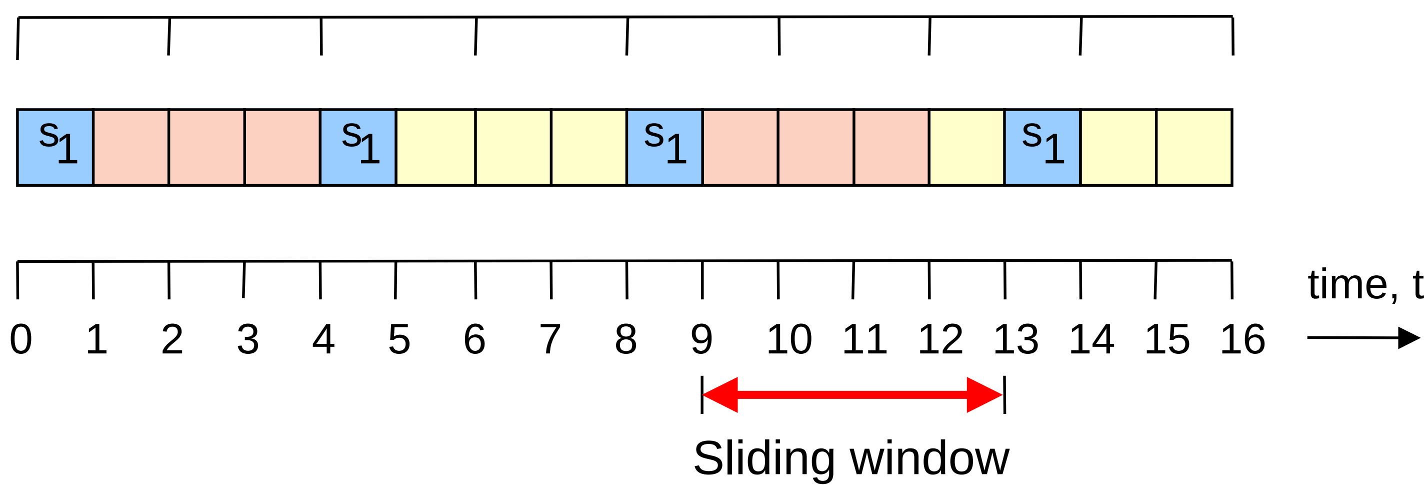
- Dynamic Window-Constrained Scheduling of:
  - Threads
    - “Guarantee” **minimum** quantum of service every fixed window of service time.
  - Packets
    - “Guarantee” at most **x** late / lost packets every window of **y** packets.

DWCS Packet Scheduling

- Two attributes per packet stream, $\mathbf{S_i}$:
  - Request period, $\mathbf{T_i}$.
    - Defines interval between deadlines of consecutive pairs of packets in $\mathbf{S_i}$.
  - Window-constraint, $\mathbf{W_i} = \mathbf{x_i/y_i}$.
    - Essentially, a "loss-tolerance".

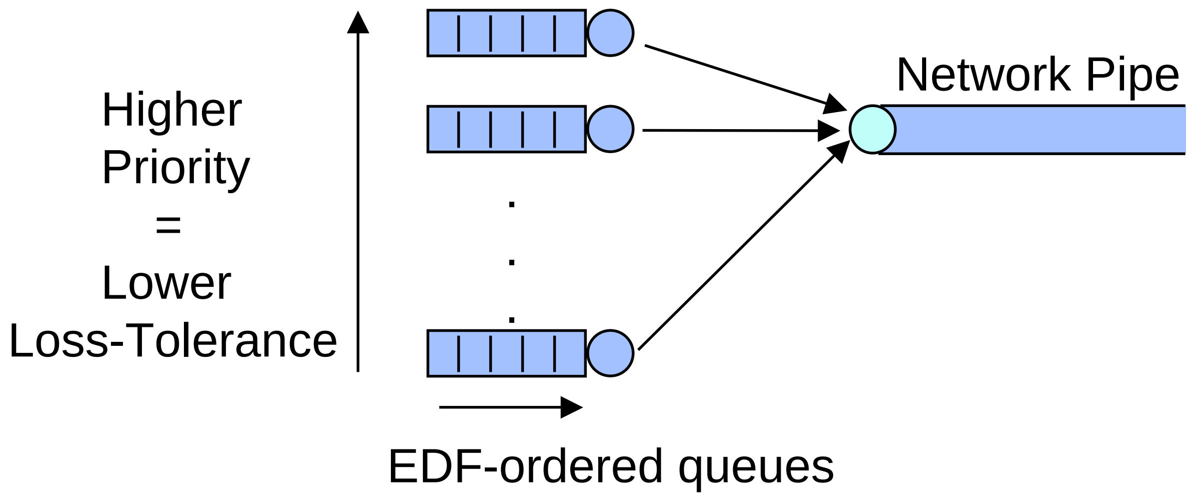
“x out of y” Guarantees

- e.g., Stream S_1 with C_1=1, T_1=2 and W_1=1/2



- Feasible schedule if “x out of y” guarantees are met.

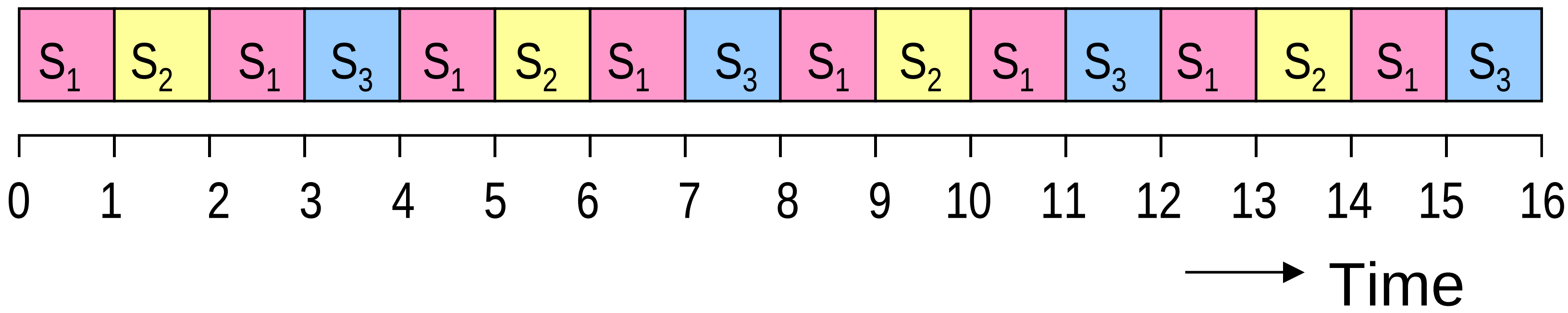
DWCS - Original Conceptual View



(x,y)-firm DWCS: Pairwise Packet Ordering Table

Precedence amongst pairs of packets
• Lowest window-constraint first
• Same non-zero window-constraints, order EDF
• Same non-zero window-constraints & deadlines, order lowest window-numerator first
• Zero window-constraints and denominators, order EDF
• Zero window-constraints, order highest window-denominator first
• All other cases: first-come-first-serve

Example: “Fair” Scheduling

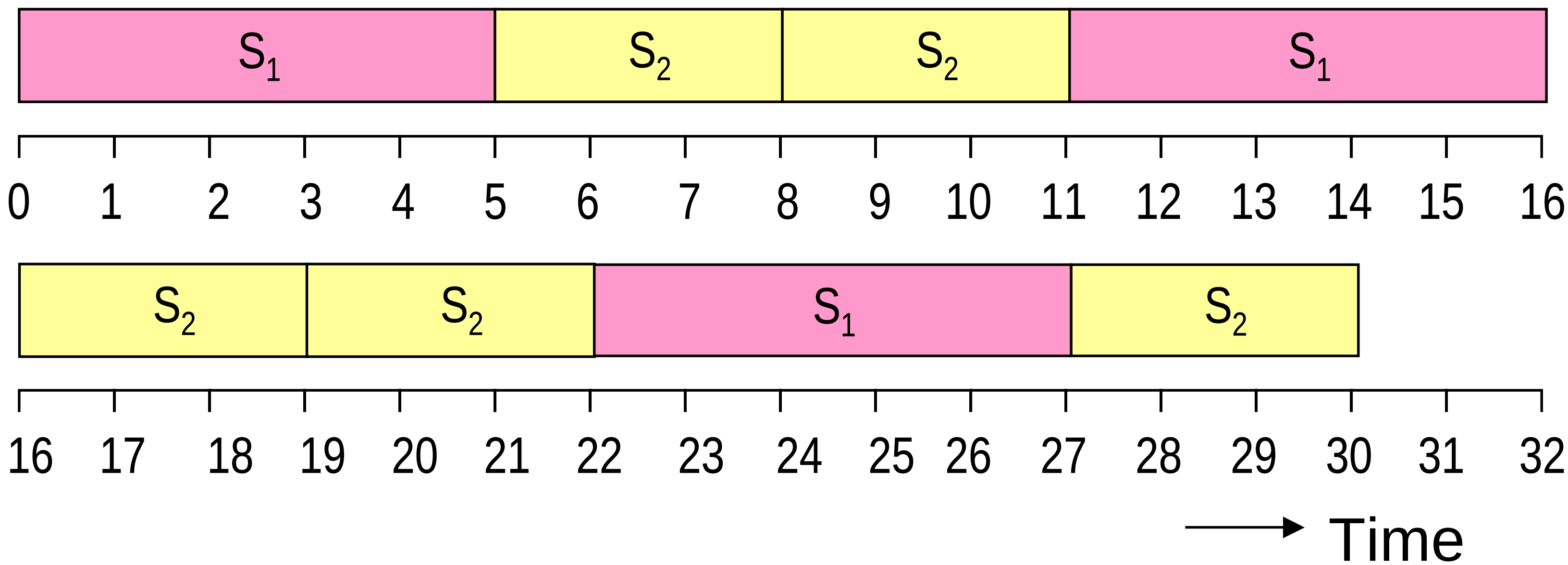


S_1 1/2(0) 1/1(1) 1/2(2) 1/1(3) 1/2(4)...

S_2 3/4(0) 2/3(1) 2/2(2) 1/1(3) 3/4(4)...

S_3 6/8(0) 5/7(1) 4/6(2) 3/5(3) 3/4(4) 2/3(5) 1/2(6) 0/1(7) 6/8(8)...

Example: Variable Length Packets



S_1 1/2(0) 1/1(5) 1/2(10) 0/1(15) 1/2(20) 0/1(25) 1/2(30)...

S_2 1/2(0) 0/1(3) 1/4(6) 1/3(9) 1/2(12) 0/1(15) 1/4(18) 1/3(21) 1/2(24) 0/1(27) 1/2(30)...

Window-Constraint Adjustment (A)

- For stream $\mathbf{S_i}$ whose head packet is serviced **before** its deadline:
  - if ($\mathbf{y_i}$' > $\mathbf{x_i}$') then $\mathbf{y_i}$'=$\mathbf{y_i}$'-1;
  - else if ($\mathbf{y_i}$' = $\mathbf{x_i}$') **and** ($\mathbf{x_i}$' > 0) then
    - $\mathbf{x_i}$'=$\mathbf{x_i}$'-1; $\mathbf{y_i}$'=$\mathbf{y_i}$'-1;
  - if ($\mathbf{x_i}$'=$\mathbf{y_i}$'=0) **or** ($\mathbf{S_i}$ **is tagged**) then
    - $\mathbf{x_i}$'=$\mathbf{x_i}$; $\mathbf{y_i}$'=$\mathbf{y_i}$;
  - if ($\mathbf{S_i}$ **is tagged**) then **reset tag**;

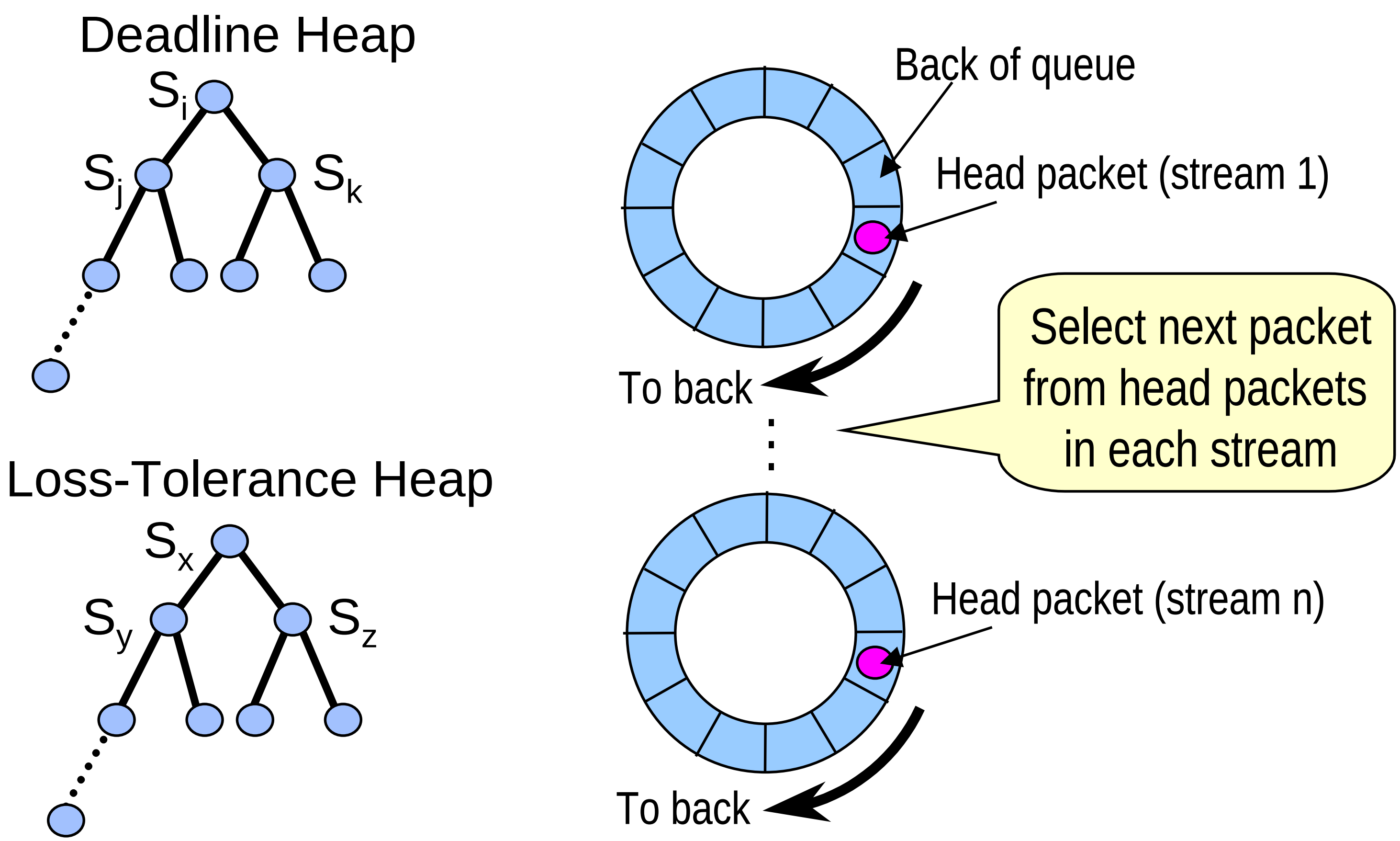
Window-Constraint Adjustment (B)

- For stream $\mathbf{S_j}$ whose head packet **misses** its deadline:
  - if ($\mathbf{x_j'}$ > 0) then
    - $\mathbf{x_j'}$=$\mathbf{x_j'}$-1; $\mathbf{y_j'}$=$\mathbf{y_j'}$-1;
    - if ($\mathbf{x_j'}$=$\mathbf{y_j'}$=0) then $\mathbf{x_j'}$=$\mathbf{x_j}$; $\mathbf{y_j'}$=$\mathbf{y_j}$;
  - else if ($\mathbf{x_j'}$=0) **and** ($\mathbf{y_j}$ > 0) then
    - violation! One solution…
    - $\mathbf{y_j'}$=$\mathbf{y_j'}$+ε;
    - **Tag** $\mathbf{S_j}$ with a violation;

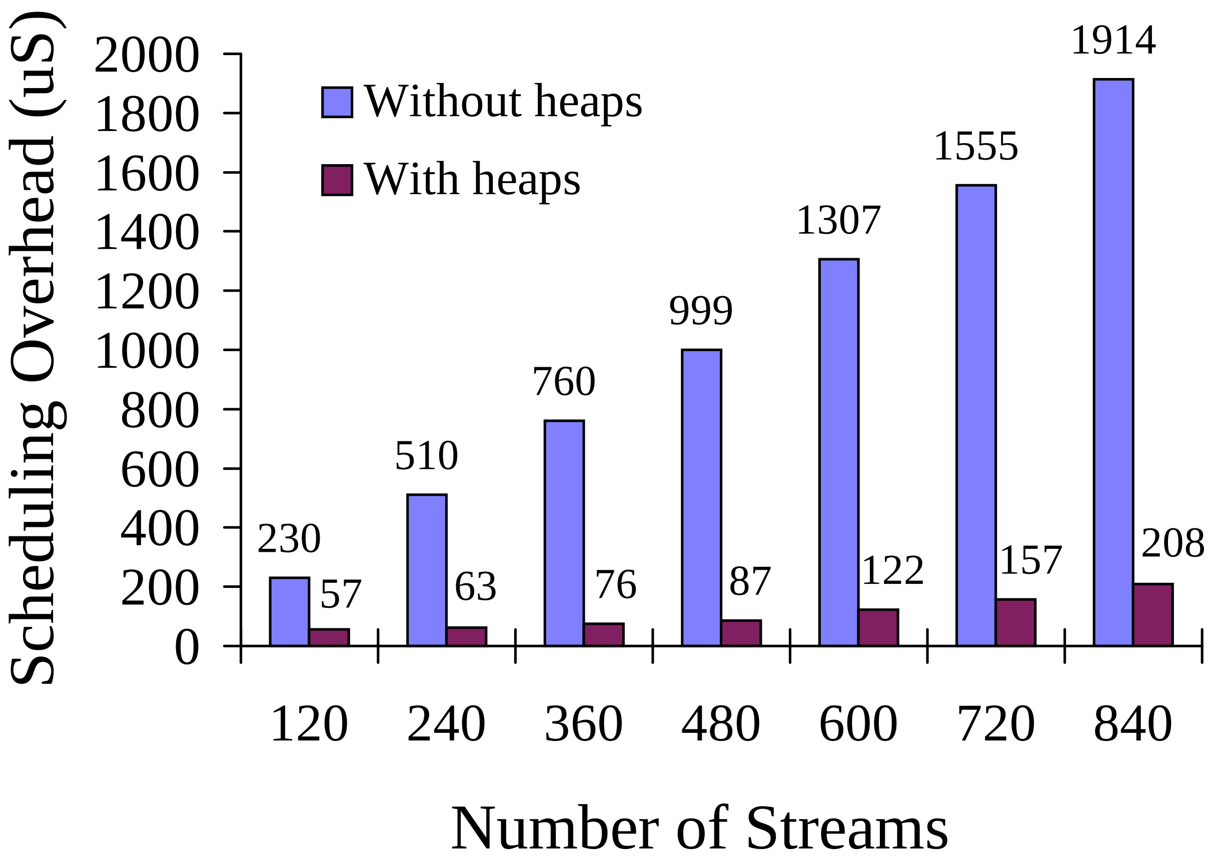
DWCS Algorithm Outline

- Find stream $\mathbf{S_i}$ with highest priority **(see Table)**
- Service head packet of stream $\mathbf{S_i}$
- Adjust $\mathbf{W_i}$**'** according to **(A)**
- $\mathbf{Deadline_i} = \mathbf{Deadline_i} + \mathbf{T_i}$
- For each stream $\mathbf{S_j}$ missing its deadline:
  - While deadline is missed:
    - Adjust $\mathbf{W_j}$**'** according to **(B)**
    - Drop head packet of stream $\mathbf{S_j}$ if droppable
    - $\mathbf{Deadline_j} = \mathbf{Deadline_j} + \mathbf{T_j}$

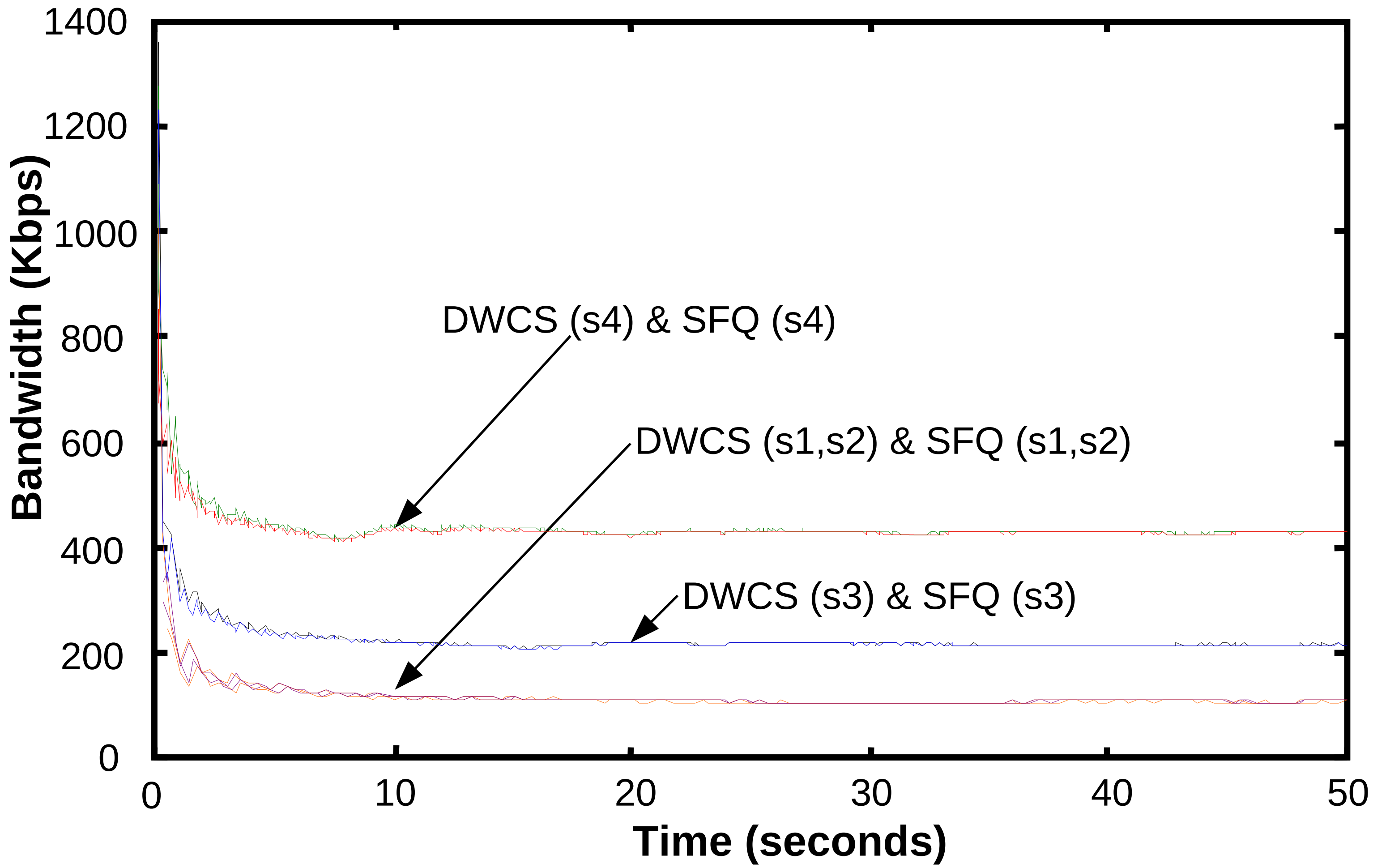
DWCS Implementation



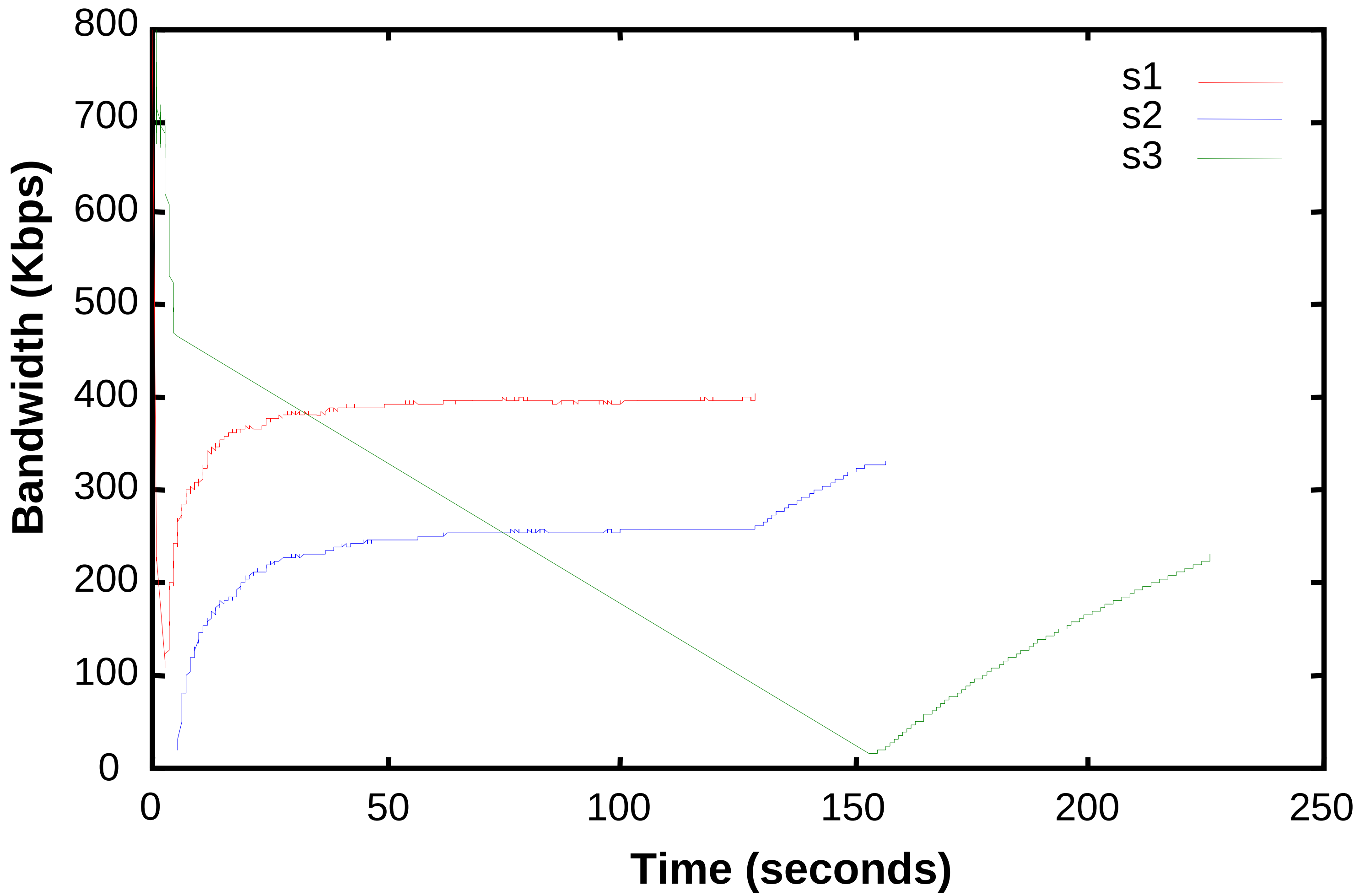
Scheduling Overhead



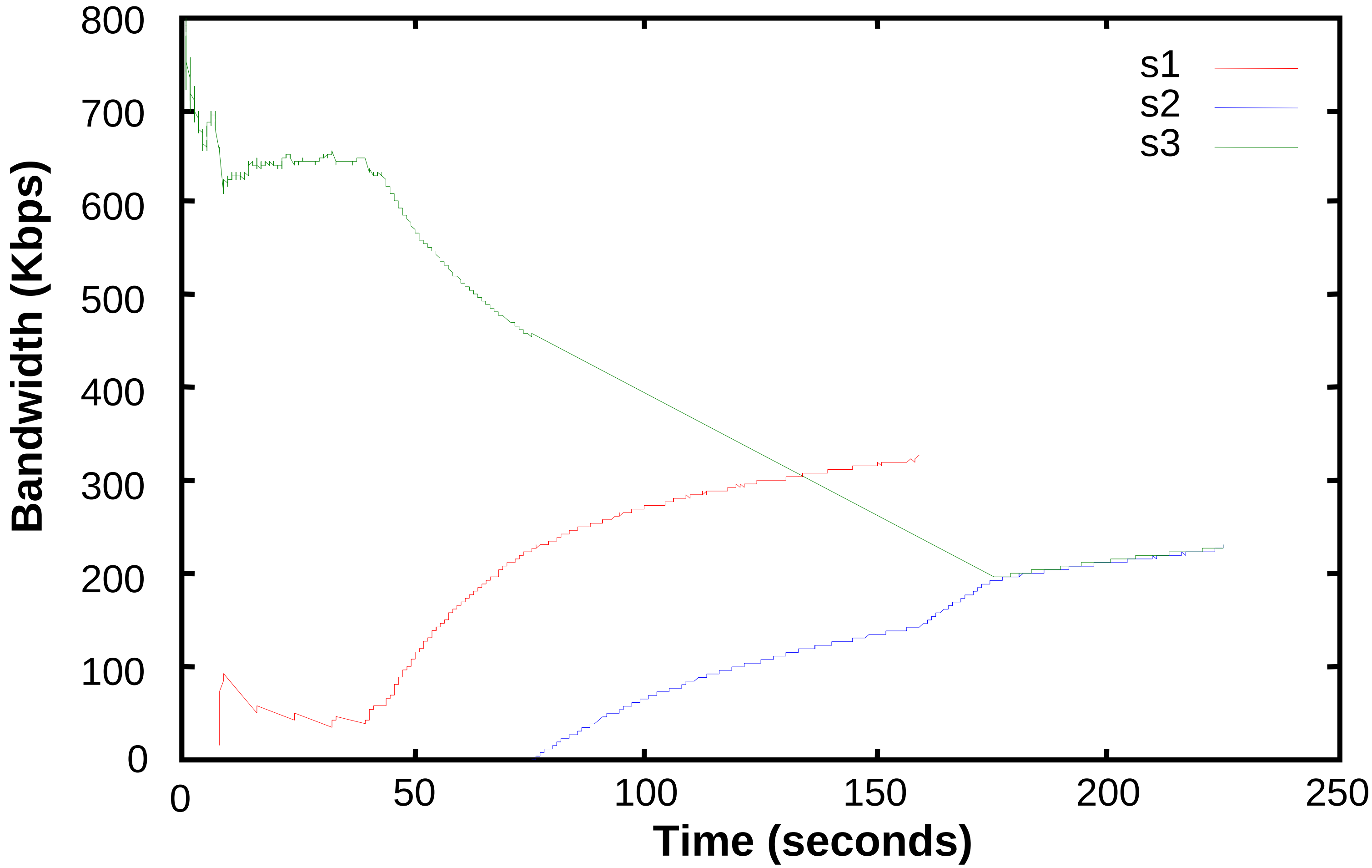
Fair Scheduling: b/w ratios:1,1,2,4 W's=7/8,14/16,6/8,4/8



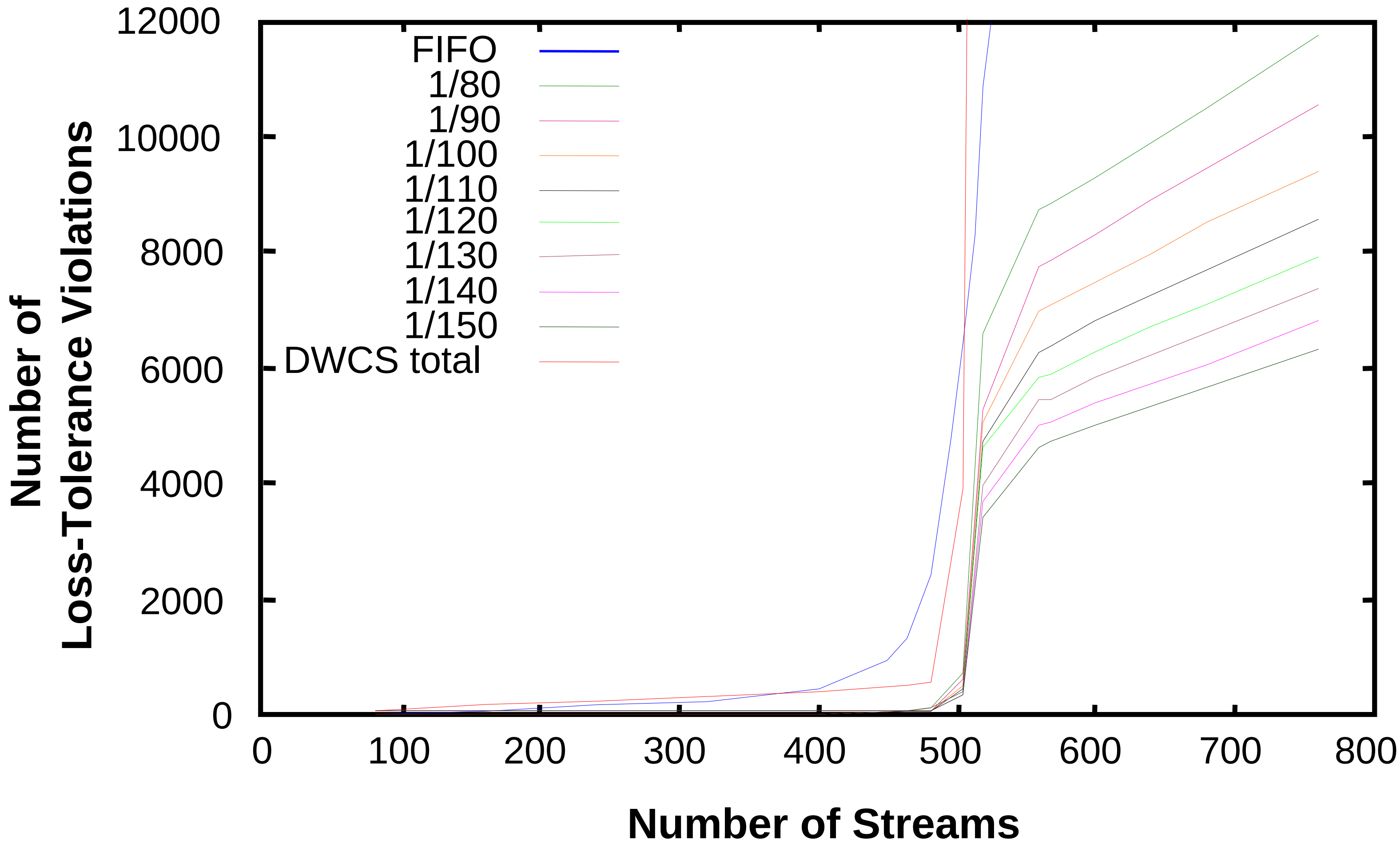
Mixed Traffic: W1=1/3,W2=2/3, W3=0/100,T1=1,T2=1,T3=∞



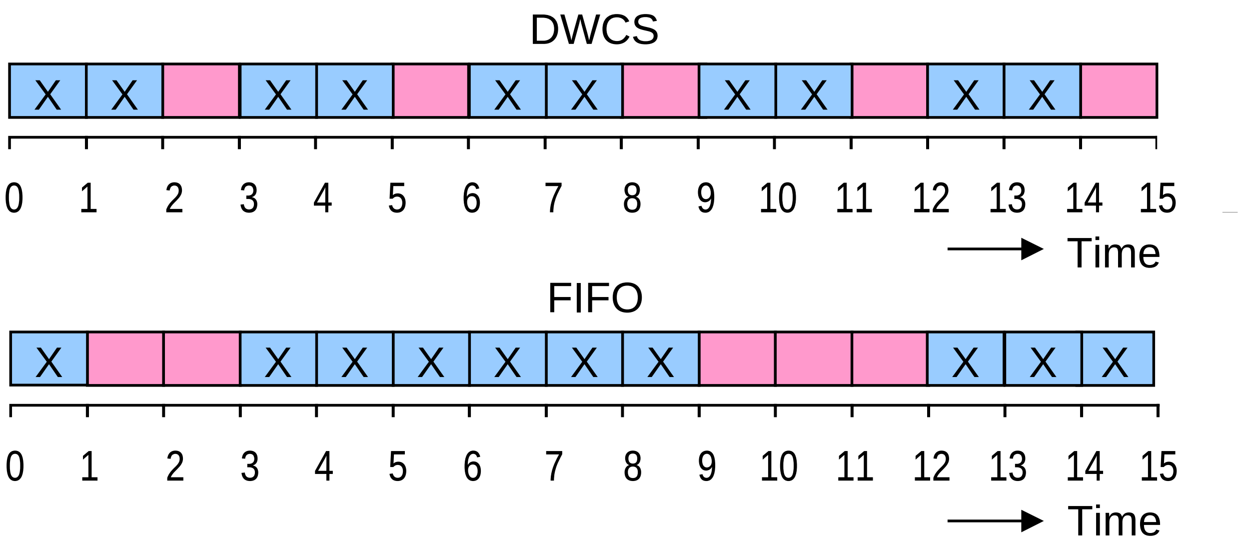
Mixed Traffic: W1=1/3,W2=2/3, W3=0/1500,T1=1,T2=1,T3=∞



Loss-Tolerance Violations (T=500, C=1)

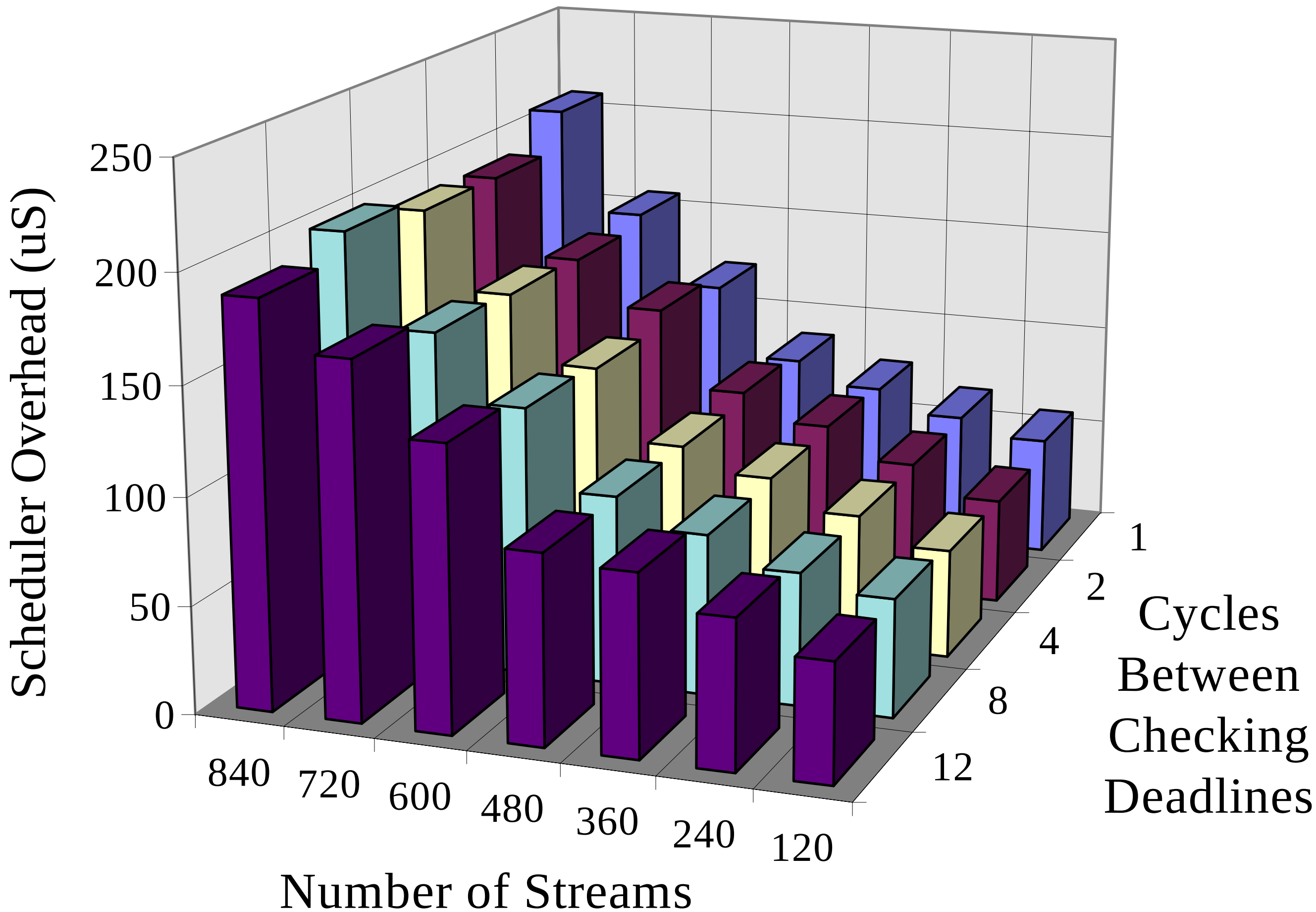


DWCS Spreads Losses

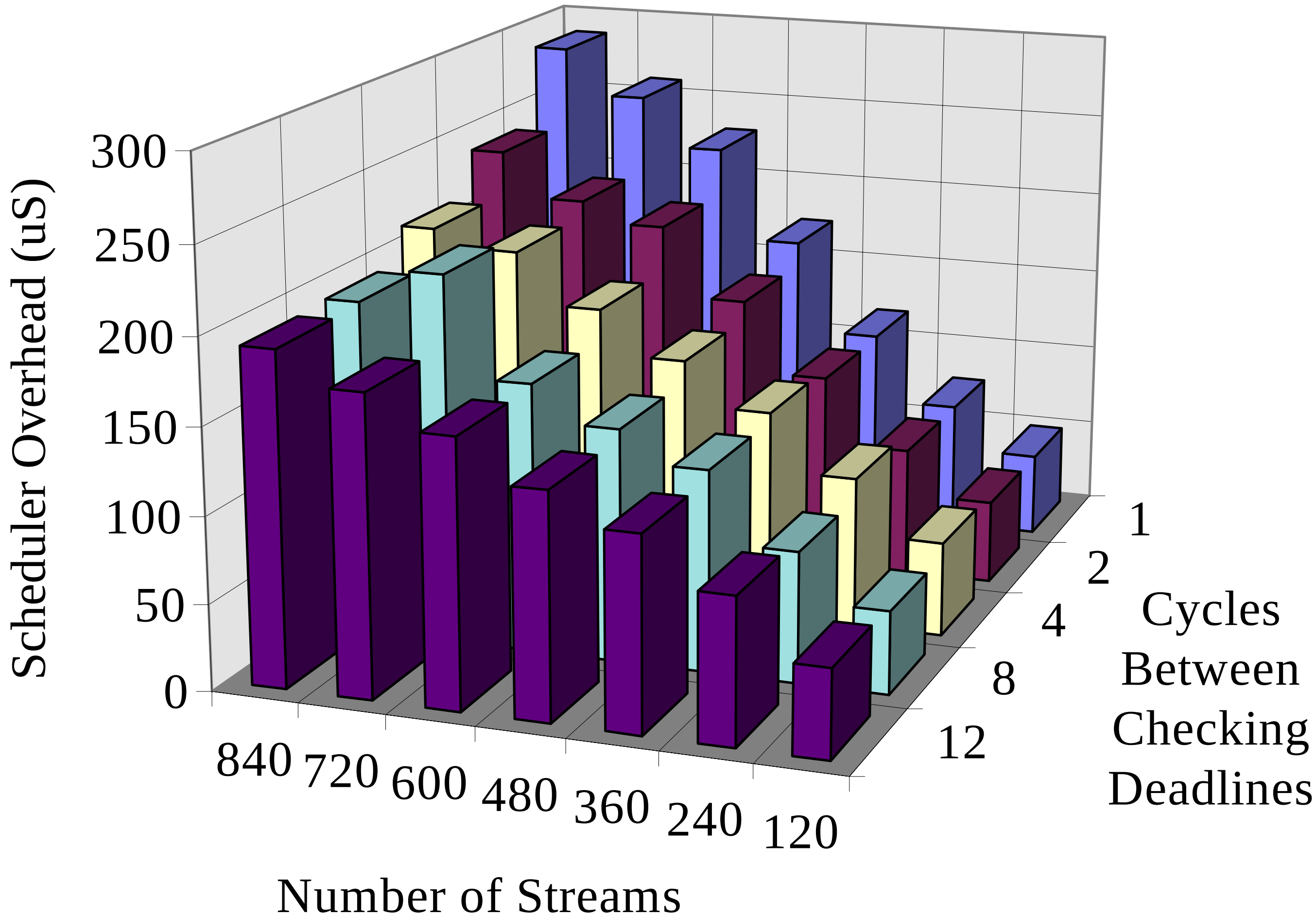


- Here, loss tolerance of 1/3 is violated more times with DWCS than FIFO, but losses are spread evenly.

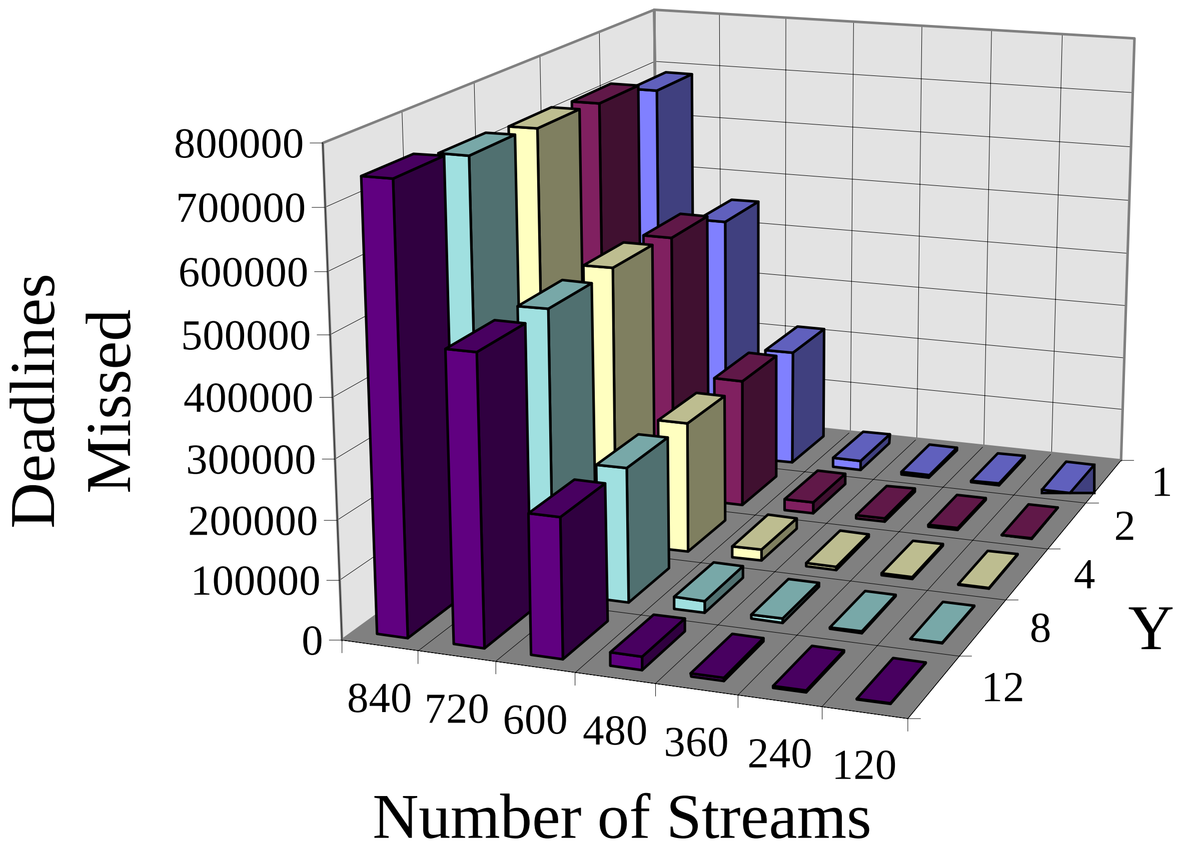
Approximation Overheads (T=500)



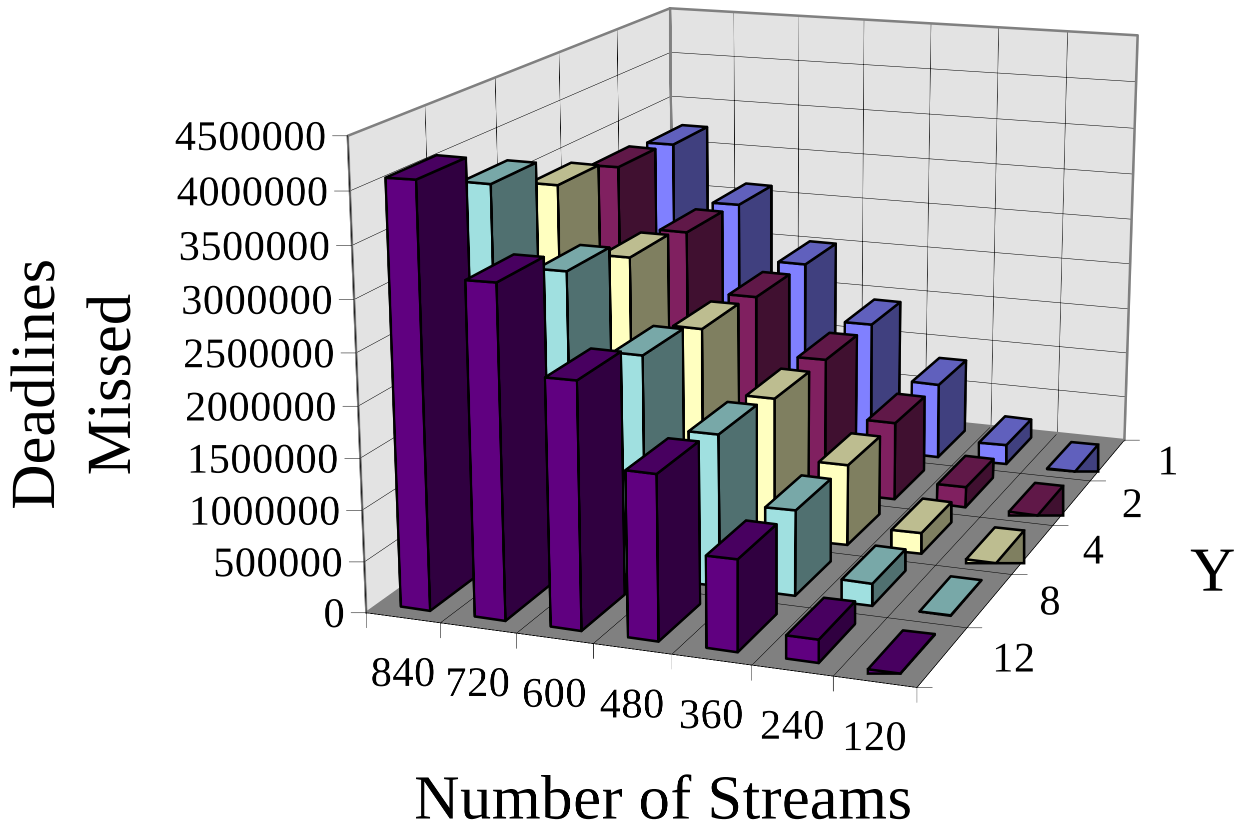
Approximation Overheads (T=200)



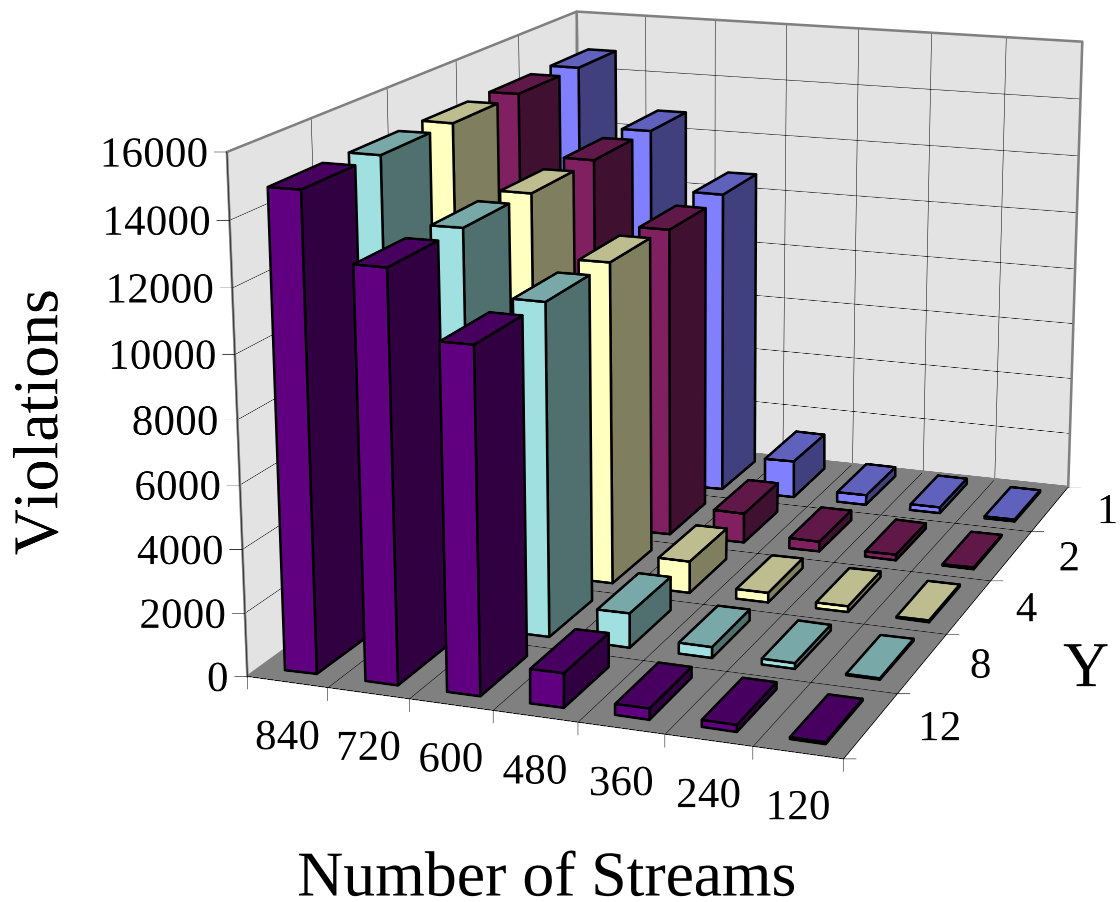
Deadlines Missed (T=500)



Deadlines Missed (T=200)

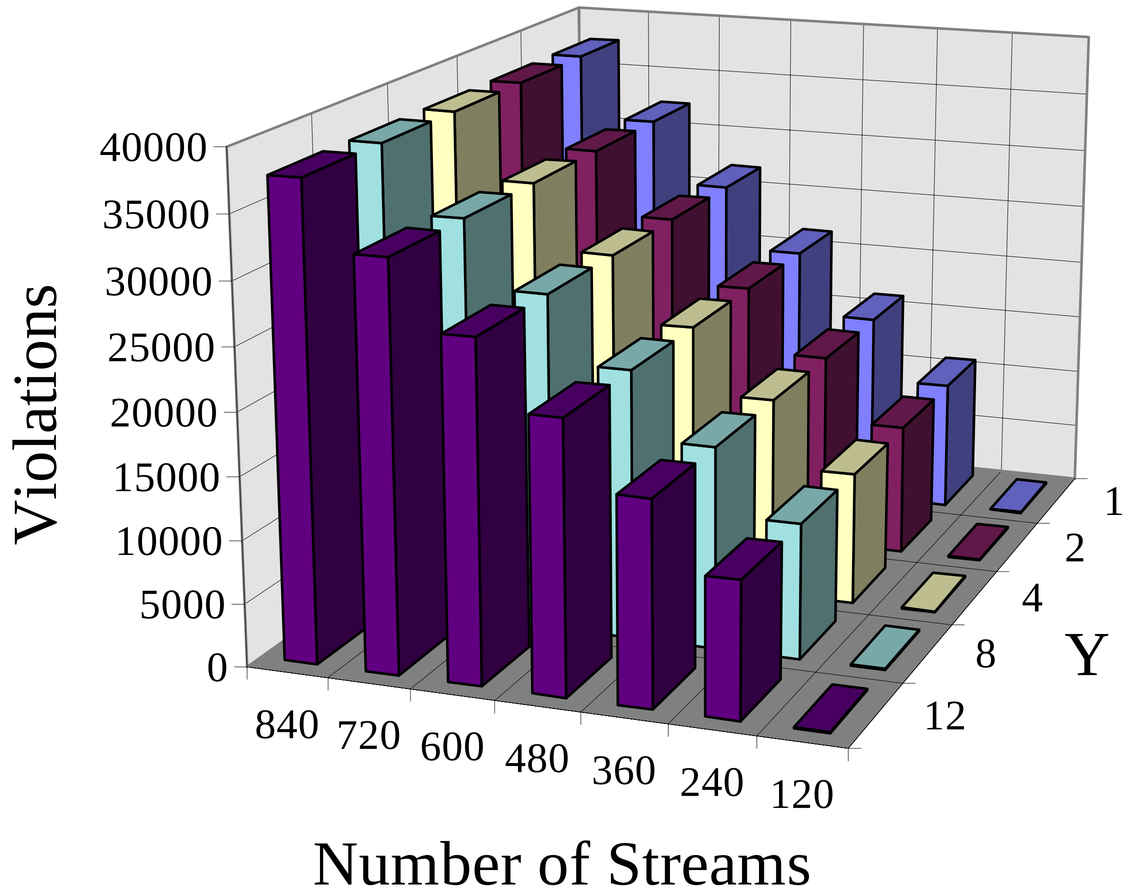


Loss-Tolerance Violations (T=500)



BOSTON UNIVERSITY Rich West (2001)

Loss-Tolerance Violations (T=200)



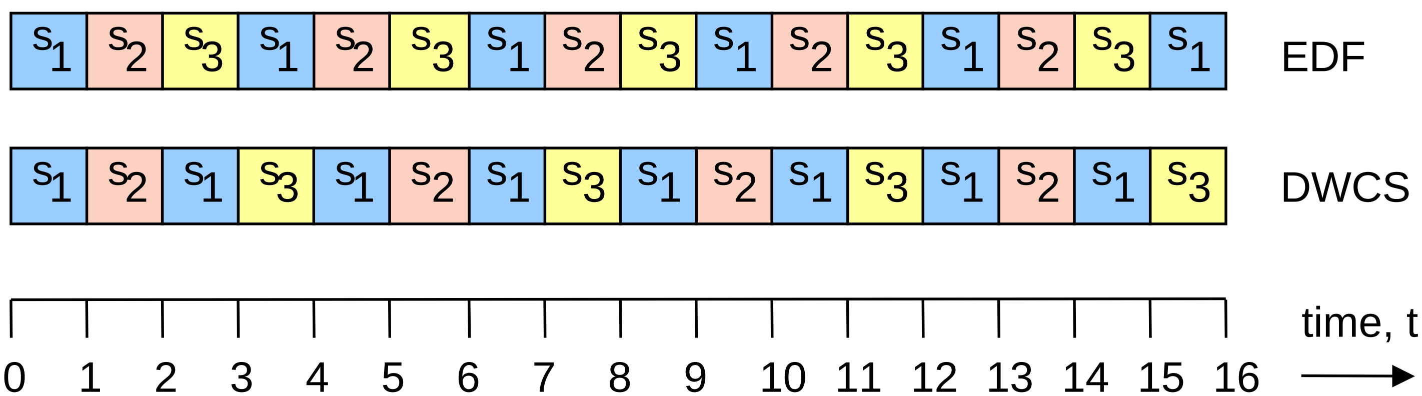
DWCS - Recent Developments

- Support for **(x,y)-hard** deadlines as opposed to **(x,y)-firm** deadlines.
  - Bounded service delay.
  - Guaranteed service in a finite window of time.
  - Optimal (100%) utilization bound for fixed-length packets or (variable-length preemptive) threads.
- Replacement CPU scheduler in Linux kernel.
  - **www.cc.gatech.edu/~west/dwcs.html**

(x,y)-Hard DWCS: Pairwise Packet Ordering Table

Precedence amongst pairs of packets
• Earliest deadline first (EDF)
• Same deadlines, order lowest window-constraint first
• Equal deadlines and zero window-constraints, order highest window-denominator first
• Equal deadlines and equal non-zero window-constraints, order lowest window-numerator first
• All other cases: first-come-first-serve

EDF versus DWCS



s_1 1/2(1),1/1(2),1/2(3),1/1(4),1/2(5)...

s_2 3/4(1),2/3(2),2/2(3),1/1(4),3/4(5),2/3(6),2/2(7),1/1(8),3/4(9)...

s_3 6/8(1),5/7(2),4/6(3),3/5(4),3/4(5),2/3(6),1/2(7),0/1(8),6/8(9)...

DWCS Delay Characteristics

- If feasible schedule, max delay of service to $\mathbf{S_i}$ is:
  - $(\mathbf{x_i} + 1)\mathbf{T_i} - \mathbf{C_i}$
  - Note: Every time $\mathbf{S_i}$ is not serviced for $\mathbf{T_i}$ time units $\mathbf{x_i}$**'** is decremented by 1 until it reaches 0.

- If no feasible schedule, max delay of service to $\mathbf{S_i}$ is still bounded.
- Function of time to have:
  - Earliest deadline, lowest window-constraint, highest window-denominator.

Bandwidth Utilization

- Minimum utilization factor of stream $\mathbf{S_i}$ is:

$$U_i = \frac{(y_i - x_i)C_i}{y_i T_i}$$

  - i.e., min req'rd fraction of bandwidth.
- **Least upper bound** on utilization is min of utilization factors for all streams that fully utilize bandwidth.
  - i.e., guarantees a feasible schedule.
  - L.U.B. is 100% in a slotted-time system.

Scheduling Test

- If:

$$\sum_{i=1}^{n} \frac{\left(1-\frac{x_i}{y_i}\right).C_i}{T_i} \le 1.0$$

  and C_i=**K**, T_i=**qK** for all **i**, where **q** is 1,2,…etc, then a feasible schedule exists.
- For variable length packets:
  - let C_i<=**K** for all **i** or fragment/combine packets & translate service constraints.
    - e.g., ATM SAR layer.

Simulation Scenario

- 8 classes of packet streams:

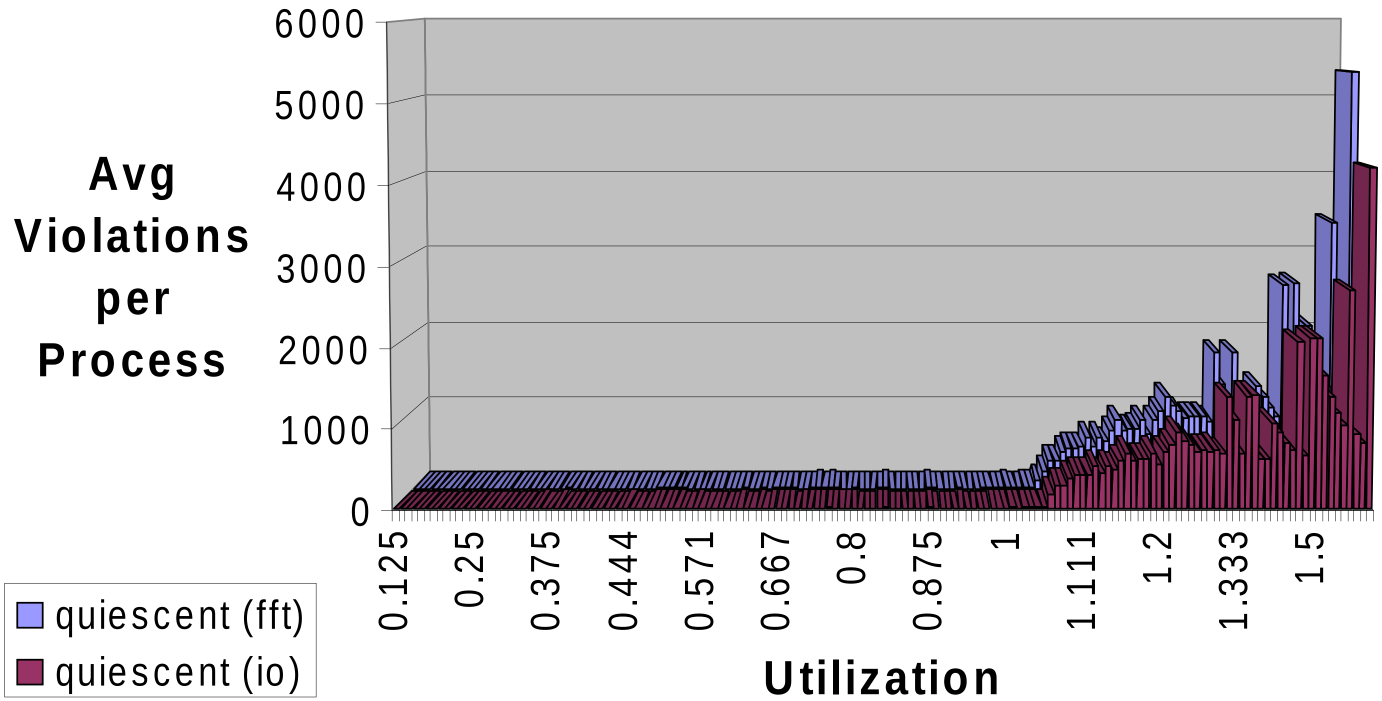
W_i	1/10	1/20	1/30	1/40	1/50	1/60	1/70	1/80
T_i	400	400	480	480	560	560	640	640

- Varied number of streams **n**, uniformly distributed amongst traffic classes.
- Total of a million packets serviced.

Bandwidth Utilization Results

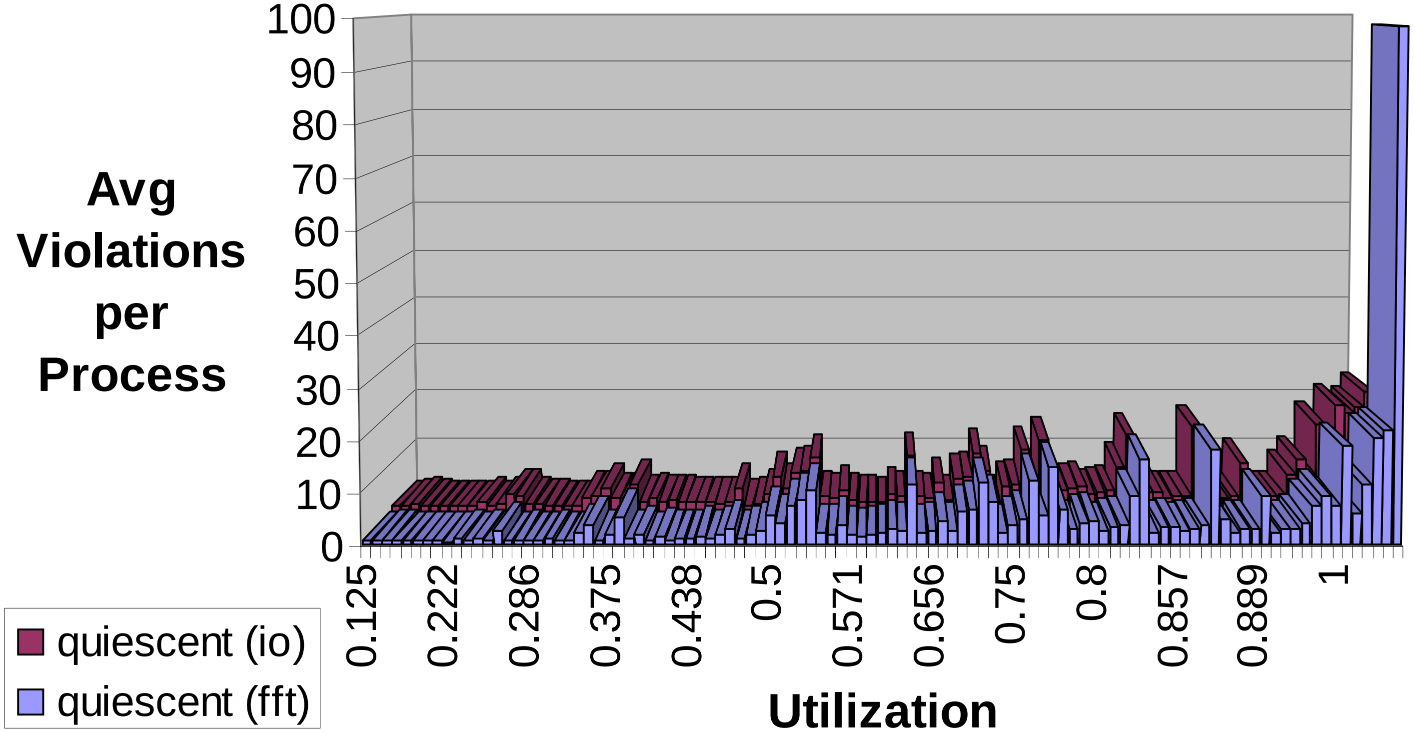
n	D	V	U	$\frac{n}{8} \cdot \sum_{i=1}^{8} \frac{C_i}{T_i}$
480	0	0	0.9156	0.9518
496	0	0	0.9461	0.9835
504	0	0	0.9613	0.9994
512	15152	0	0.9766	1.0152
520	30990	0	0.9919	1.0311
528	46828	7038	1.0071	1.047
544	78528	31873	1.0376	1.0787
560	110240	53455	1.0681	1.1104
640	268800	148143	1.2207	1.269

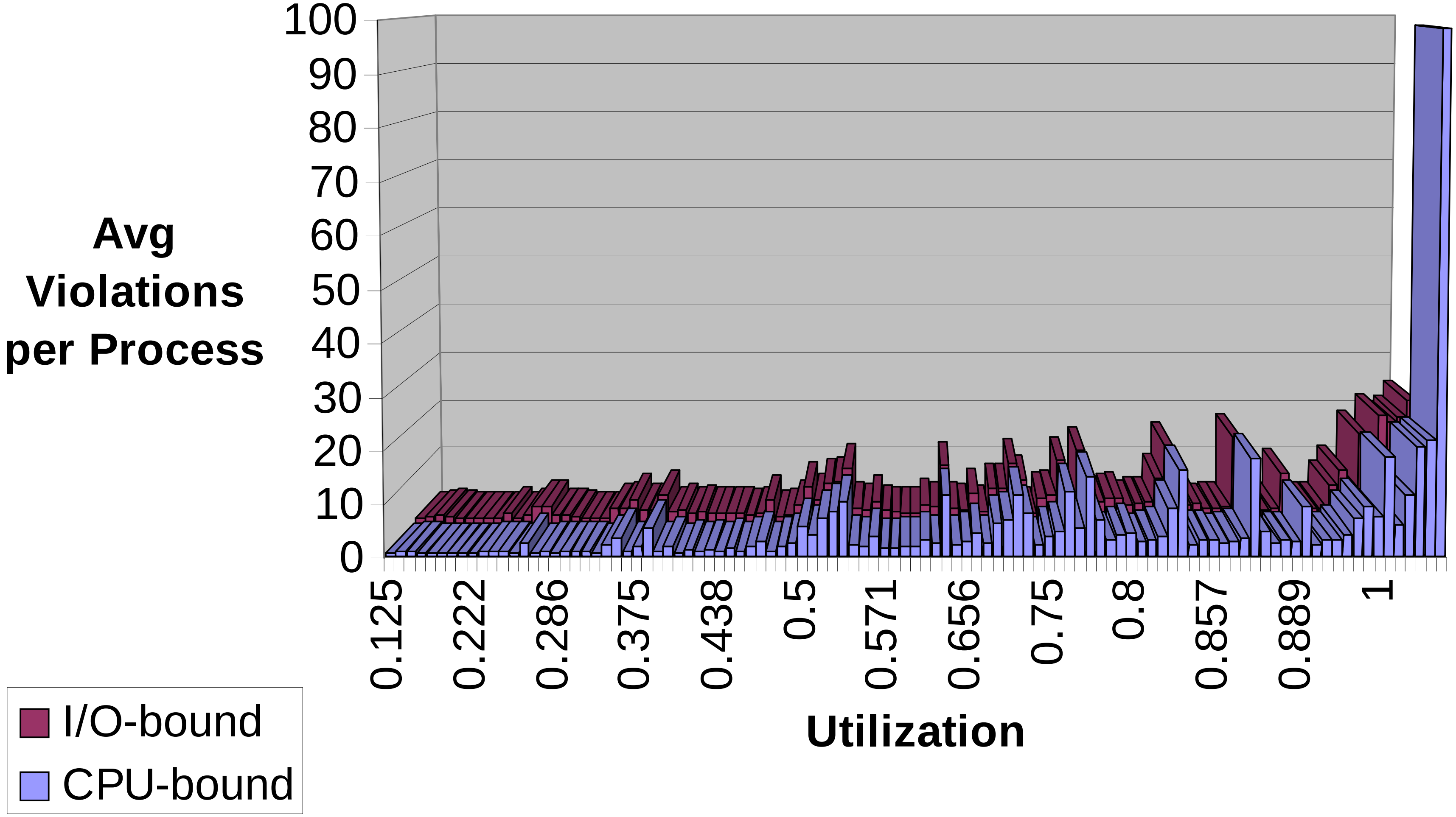
(x,y)-hard Linux CPU DWCS: Average Violations per Process



Rich West (2001)

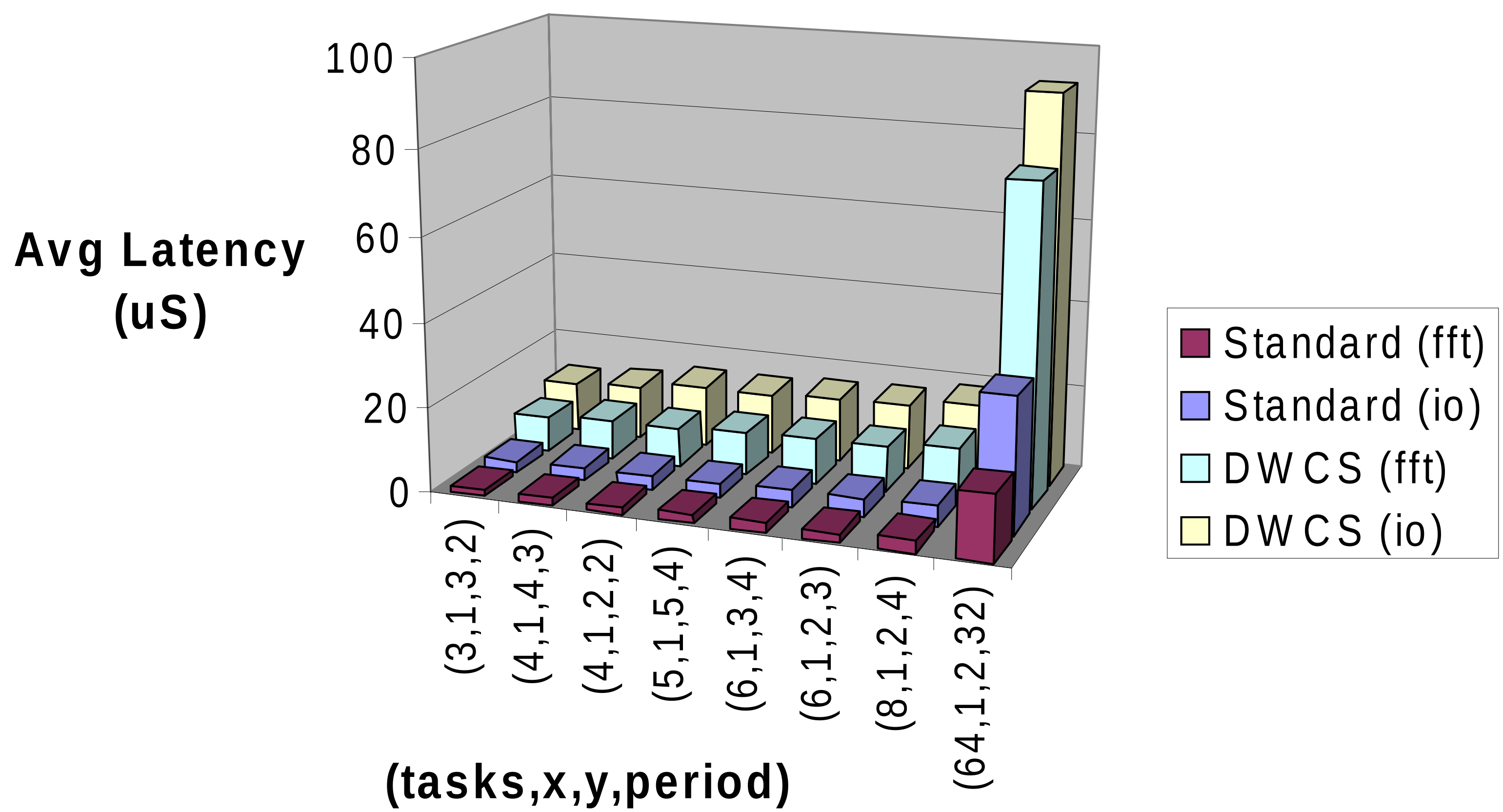
(x,y)-hard Linux CPU DWCS: Average Violations per Process



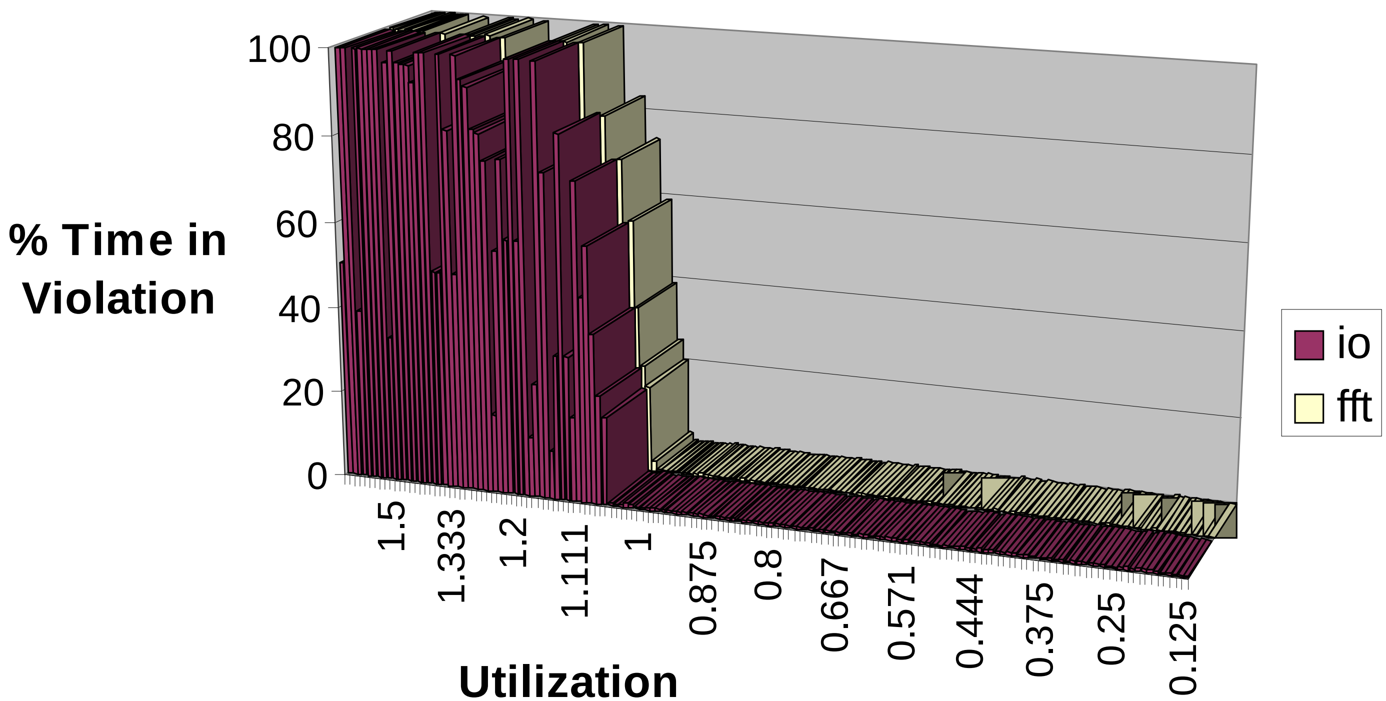


Avg Violations per Process
100
90
80
70
60
50
40
30
20
10
0
0.125
0.222
0.286
0.375
0.438
0.5
0.571
0.656
0.75
0.8
0.857
0.889
1
Utilization
I/O-bound
CPU-bound

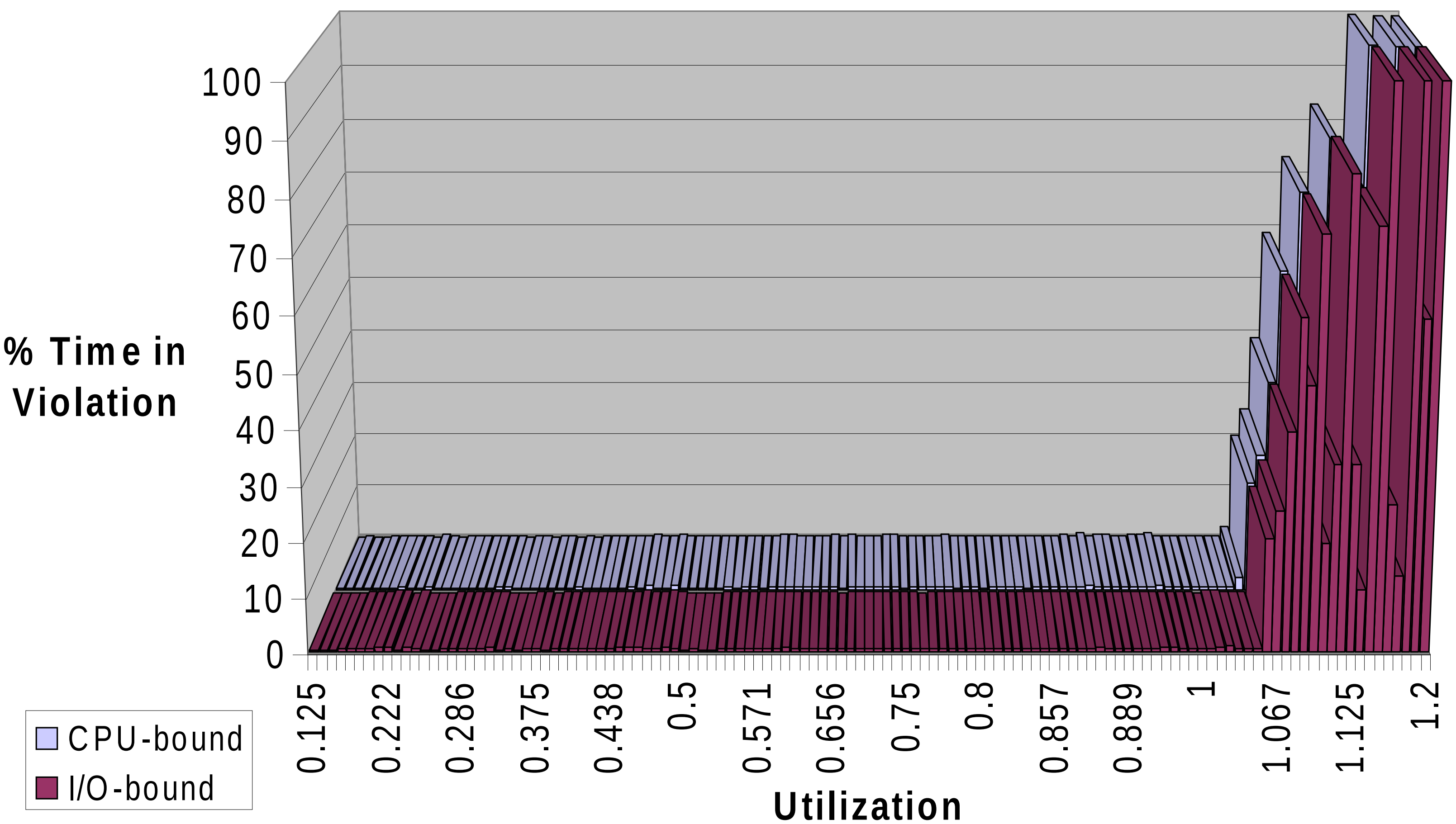
(x,y)-hard Linux CPU DWCS: Scheduling Latency



(x,y)-hard Linux CPU DWCS: % Execution Time in Violation



Rich West (2001)



% Time in Violation
100
90
80
70
60
50
40
30
20
10
0
0.125
0.222
0.286
0.375
0.438
0.5
0.571
0.656
0.75
0.8
0.857
0.889
1
1.067
1.125
1.2
Utilization
CPU-bound
I/O-bound

Conclusions

- **Flexible** approach to run-time service adaptation.
  - When, where and how to adapt.
- **Coordinated** resource management.
  - Dionisys "quality events", monitors, handlers etc.
- **DWCS** guarantees explicit loss and delay constraints for real-time / multimedia applications.

Current & Future Work

- Linux kernel-level implementation of Dionisys mechanisms.
  - Cluster-wide coordination of resources.
  - Language support for “QoS safety”.
    - Stability analysis.
  - Real-time “batched” events in Linux – “Ecalls”.
- Switch / co-processor implementation of DWCS.
  - Scheduling variable-length packets.

Related Work

- **QoS Architectures**: QoS-A (Campbell), Washington Univ. (Gopalakrishna & Parulkar), QoS Broker (Nahrstedt et al), U. Michigan (Abdelzaher, Shin), QuO (BBN) + more…
- **QoS Specification/Translation**: Tenet (Ferrari), EPIQ (Illinois).
- **QoS Evaluation**: Rewards (Abdelzaher), Value fns (Jensen), Payoffs (Kravets).
- **System Service Extensions**: SPIN (U. Washington), Exokernel (MIT).

Scheduling Related Work

- **Fair Scheduling**: WFQ/WF^2Q (Shenker, Keshav, Bennett, Zhang etc), SFQ (Goyal et al), EEVDF/Proportional Share (Stoica, Jeffay et al).
- **(m,k) Deadline Scheduling**: Distance-Based Priority (Hamdaoui & Ramanathan), Dual-Priority Scheduling (Bernat & Burns), Skip-Over (Koren & Shasha).
- **Pinwheel Scheduling**: Holte, Baruah etc.
- **Other multimedia scheduling**: SMART (Nieh and Lam).

Related Research Papers

- **Quality Events: A Flexible Mechanism for Quality of Service Management**, *RTAS 2001.*
- **Analysis of a Window-Constrained Scheduler for Real-Time and Best-Effort Traffic Streams**, *RTSS 2000.*
- **Dynamic Window-Constrained Scheduling for Multimedia Applications**, *ICMCS'99*.
- **Scalable Scheduling Support for Loss and Delay-Constrained Media Streams**, *RTAS'99*.
- **Exploiting Temporal and Spatial Constraints on Distributed Shared Objects**, *ICDCS'97*.