A Framework for Quality-Adaptive Media Streaming

Jonathan Walpole

Department of Computer Science & Engineering
OGI/OHSU

CSE515 Distributed Computing Systems

Ubiquitous Distributed Video: Motivation

Enabling Trends

- Inexpensive computer and consumer electronic devices can handle digital video
- Networks are improving and growing rapidly

Effects reported

- Video traffic in the Internet is on the rise
- In some studies video data has passed audio as the largest component of traffic mix [Sariou02]

Distributed Video Applications

Video on demand (news, entertainment, etc.)

Environmental monitoring

Surveillance

Video-phone, conferencing

Games

Remote control, surgery etc

Why is it hard to deploy video applications on the Internet?

The Facts Today

Systems and networks lack support for real-time video delivery

Network bandwidth is variable

- Dynamic variation due to sharing and heavy load
- Static variation due to widening gap between high and low-end communication, compute, and display capabilities
- No bandwidth guarantees (not really a good solution anyway)
- Increase in wireless networking

Video requirements are also highly variable over time

So why is this a problem?

What can you do about it?

State of the Art

Microsoft Windows Media; Real Networks; and Apple Quicktime

- Select from a canned set of video rates (versions)
- Automatically shift between versions

This coarse-gained adaptation has two main flaws

- Streaming failure if selected rate is too high
- Very low quality if selected rate is too low
  - *Probability of both increases with length of stream*

Conventional Wisdom Today

Video data is brittle

- Random loss of just a few percent will usually break it
- Therefore, you need reservations to stream it

TCP is unsuitable for video streaming on shared networks

- Due to rate variations and retransmission delays

Non-TCP video threatens existing network (>90% TCP)

Multicast helps, but it is inherently not TCP friendly

  - Multiple clients can't drive a single congestion window
  - Group membership changes don't support fine-grain rate adaptation

Talk Overview

Part 1: Non-brittle, *streaming-friendly*, video

- Priority drop, spatially scalable MPEG (SPEG)
- Tailorable quality adaptation, specification and mapping

Part 2: Video streaming over TCP

- Priority-Progress Streaming (PPS)

Part 3: TCP-friendly multi-rate video multicast overlay

- Priority-Progress Multicast (PPM)

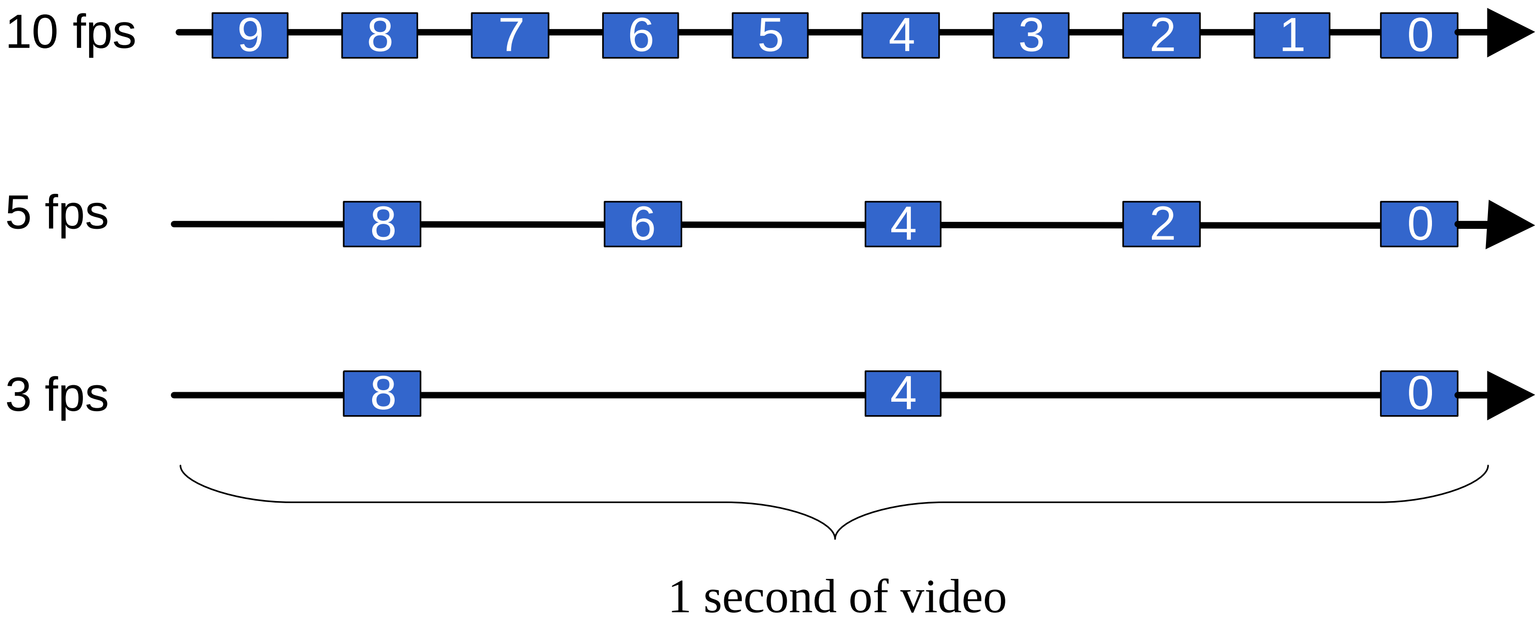
Vision: “*encode once, stream anywhere*”

Part I: Streaming-Friendly Video

Fine Grain, Rate-Adaptive Video

Compressed video need not be so brittle

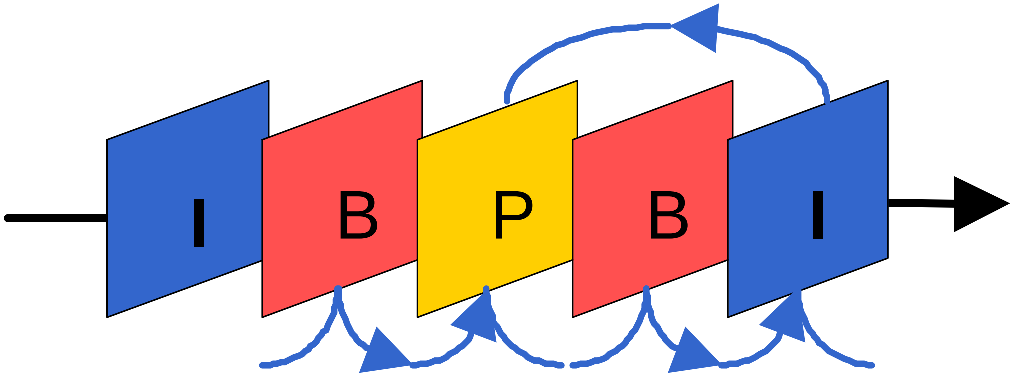
- Frame dropping is a well known technique for quality-rate adaptation



Priority Frame Dropping

Frame dropping is not quite as easy as it sounds

- Inter-frame dependencies constrain valid priority assignments

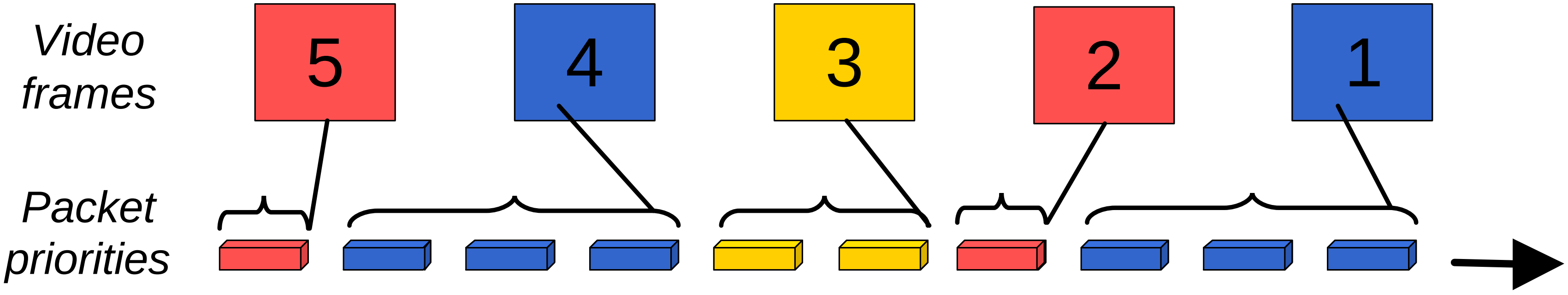


Inter-frame dependencies

🟦 *High priority frame*
🟨 *Medium priority frame*
🟥 *Low priority frame*

Priority Packet Dropping

With application level framing, priority-frame dropping can be implemented via priority packet-dropping

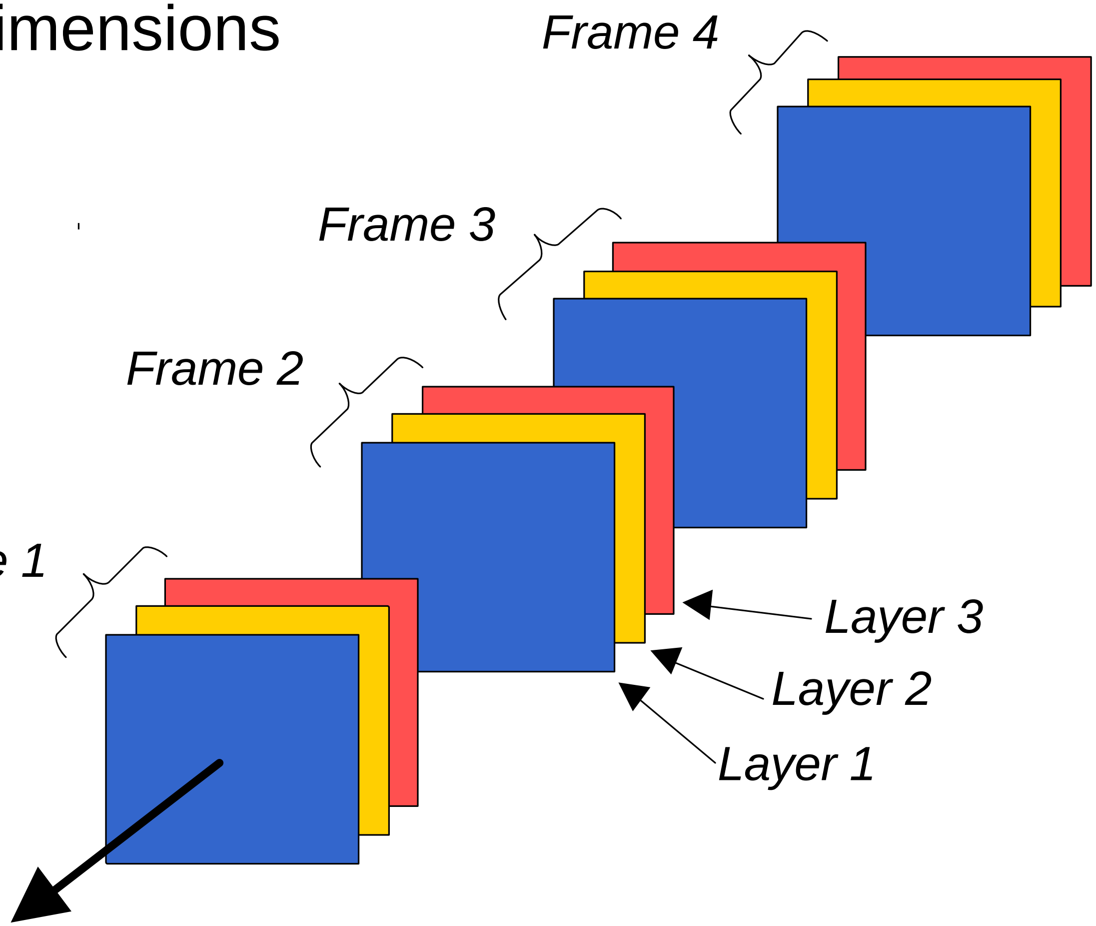


🟦 High priority packet
🟨 Medium priority packet
🟥 Low priority packet

Dropping in other Quality Dimensions

Priority dropping can be extended to other dimensions

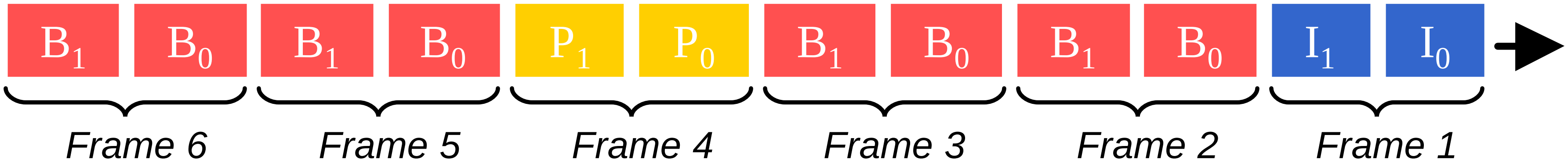
- Spatial detail (SNR)
- Spatial size
- Color
- ....



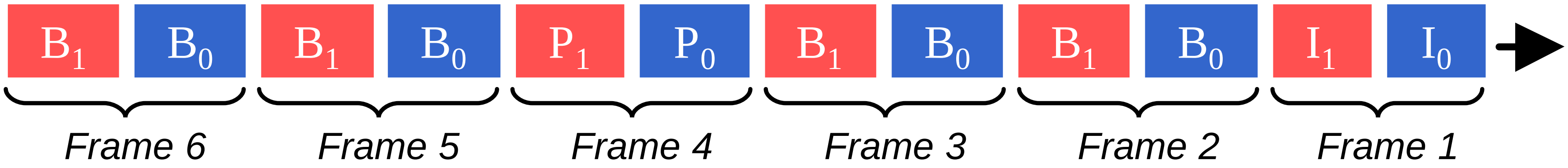
Tailorable Multi-Dimensional Scalability

🟦 High priority packet
🟨 Medium priority packet
🟥 Low priority packet

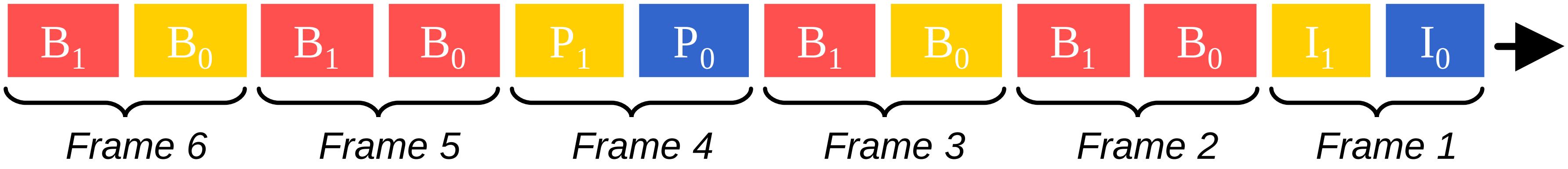
Frame dropping via priority packet dropping:



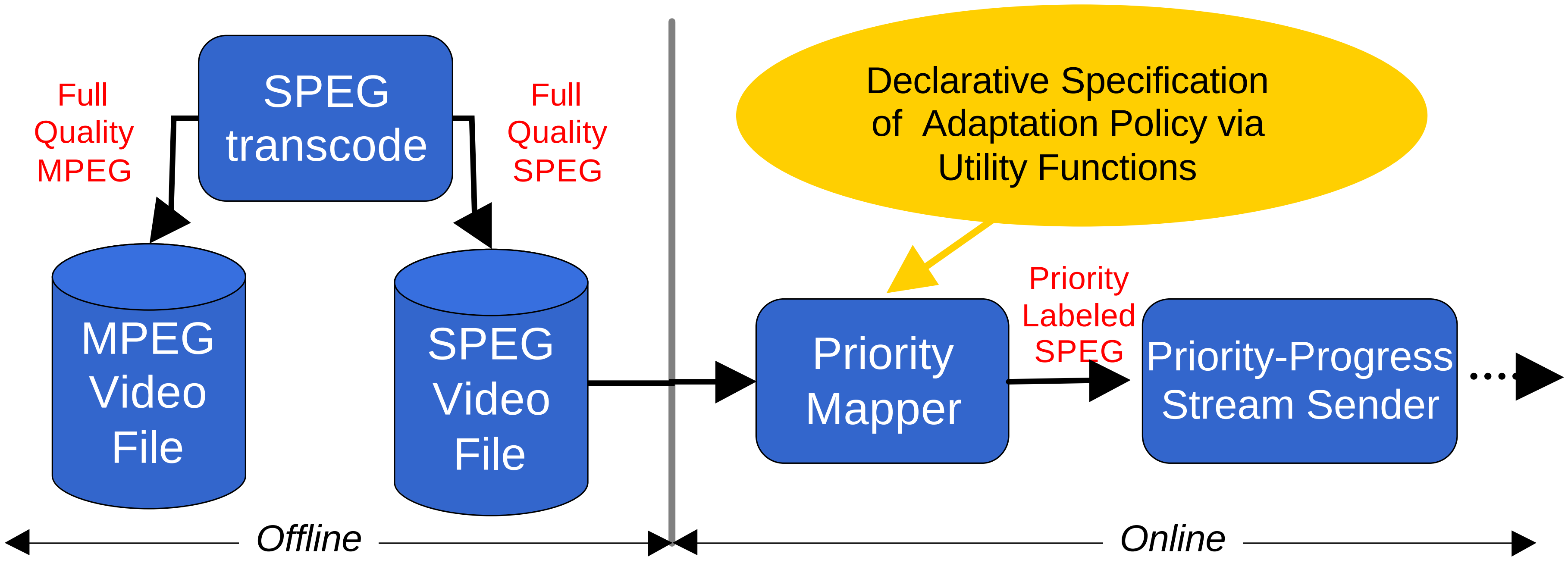
SNR dropping via priority packet dropping:



Mixed frame and SNR dropping with priority packet dropping:



QStream (server side)

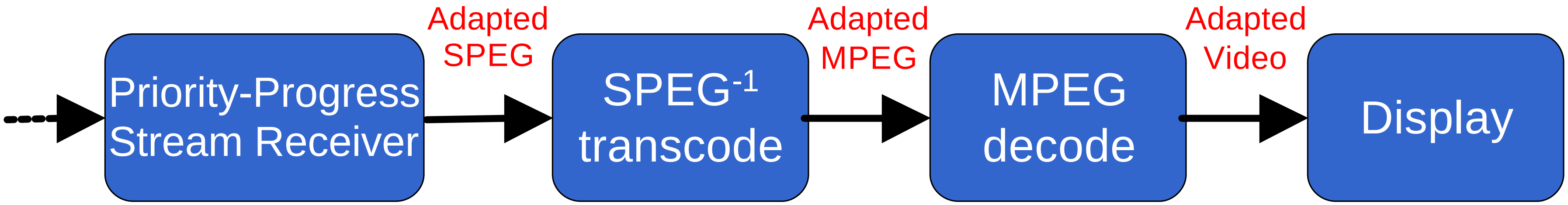


SPEG exposes spatial (SNR) and frame-rate scalability

QoS specifications/preferences define adaptation policies

Mapper translates policies into packet priority assignment

QStream (client side)



SPEG^{-1} transcode reconstructs valid MPEG from priority-packet-dropped SPEG

SPEG (Scalable MPEG)

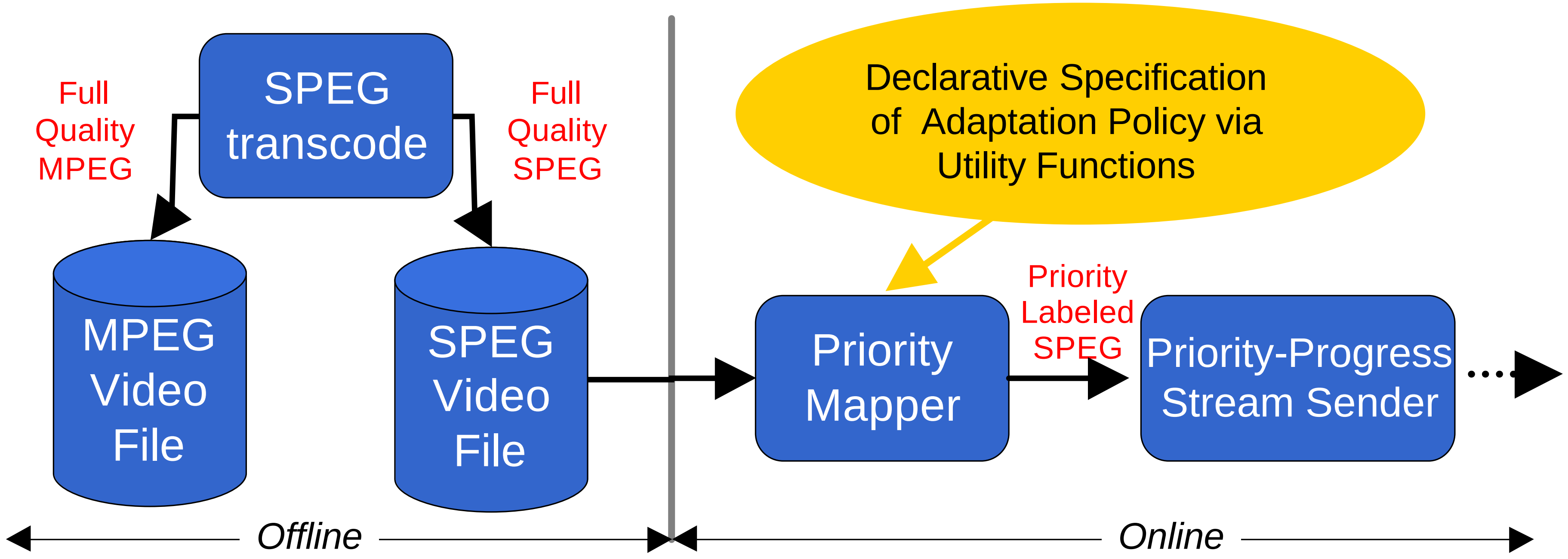
SPEG factors each picture into four (progressive) layers of spatial detail

Each layer corresponds to roughly 25% of the data rate

Compression overhead of SPEG vs MPEG is between 7 and 25%

Our approach to adaptation is a natural fit for MPEG-4 FGS

QStream (server side)



SPEG exposes spatial (SNR) and frame-rate scalability

QoS specifications are adaptation policies

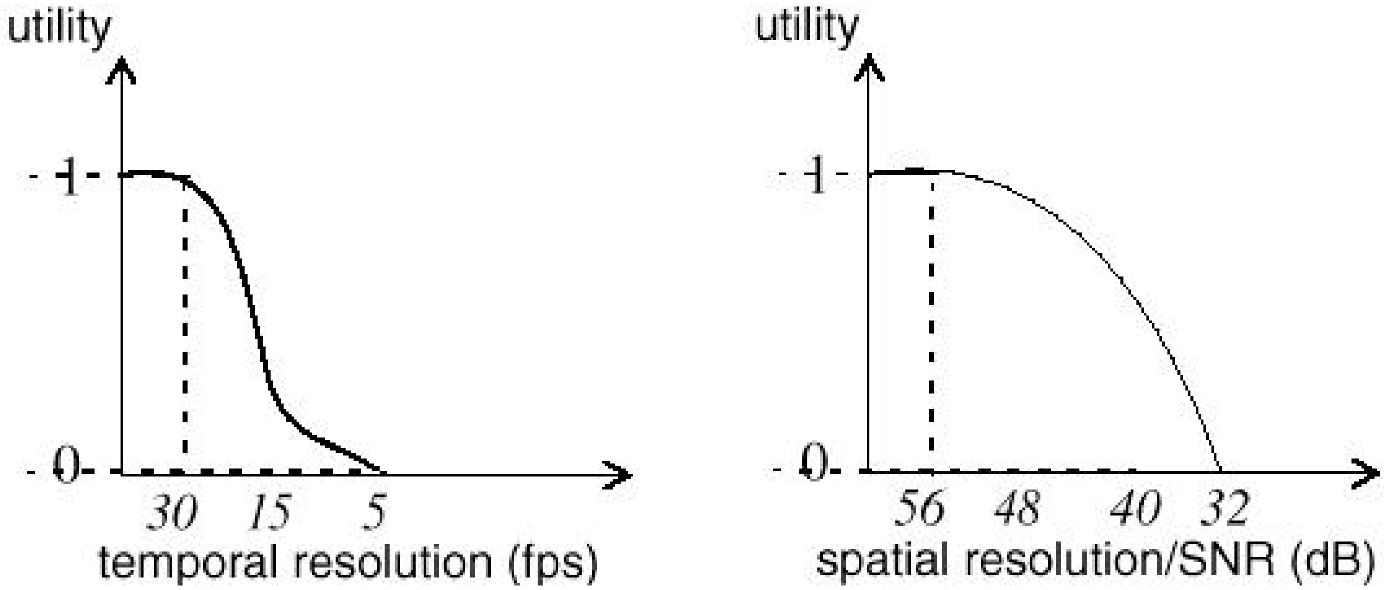
Mapper translates policies into priority assignment

Specifying Adaptation Policies

Utility Functions:

- a *declarative* approach to policy specification [Staehli95, Rajkumar97,Kravits99]
- specify preferences instead of actions
- one utility function per quality dimension
- the adaptation policy is derived automatically from the set of utility functions

Example SPEG Utility Functions



Priority Mapper

Assigns priority to video packets so that priority order dropping results in graceful degradation

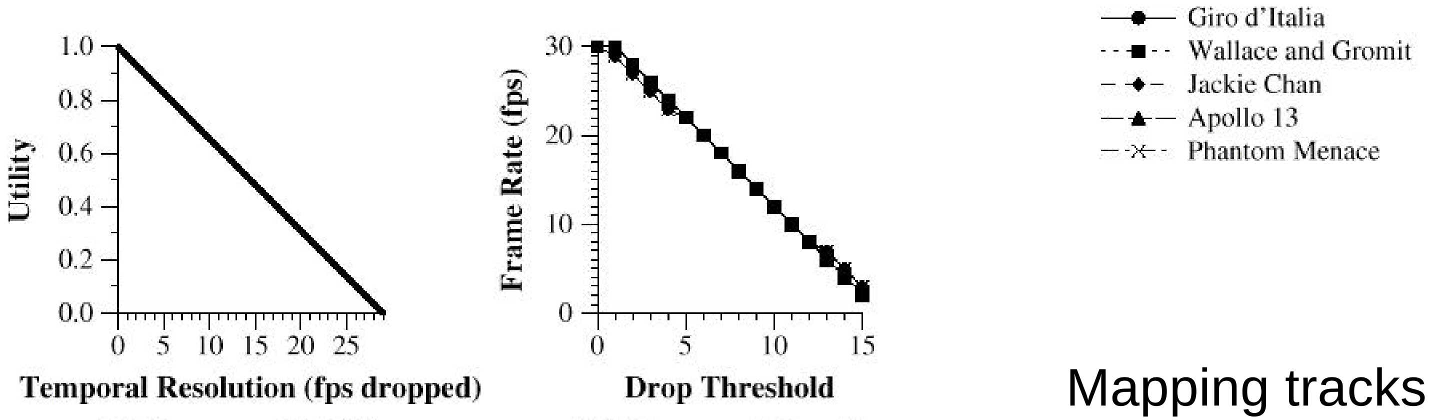
- “Graceful” is defined explicitly via the utility functions
- and the dependencies inherent in the video encoding format

Utility functions can be changed dynamically

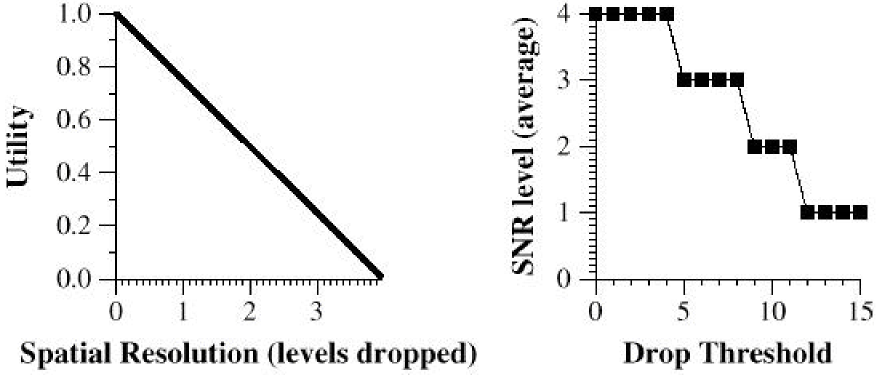
The priority mapper is efficient enough to run online

- window-based algorithm that exhaustively evaluates impact on utility of dropping each packet in the window

From Utilities to Priority



(a) Temporal Utility (b) Temporal Quality

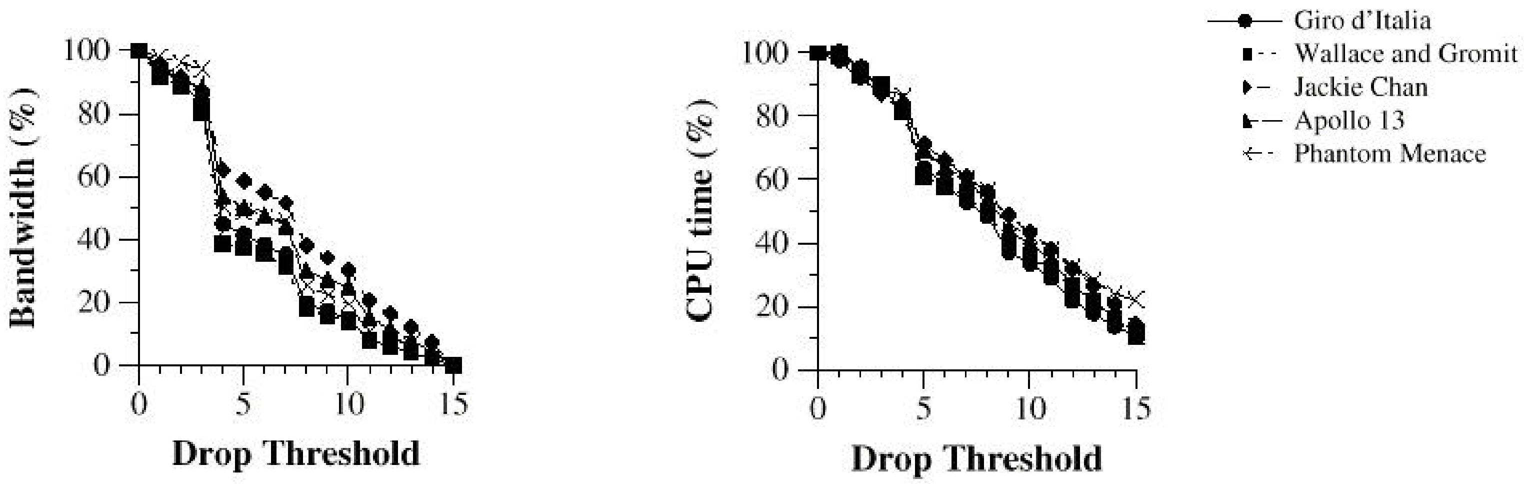


(c) Spatial Utility (d) Spatial Quality

Mapping tracks specification across movies with different encoding characteristics

Adaptation in spatial dimension is coarser than in the temporal dimension

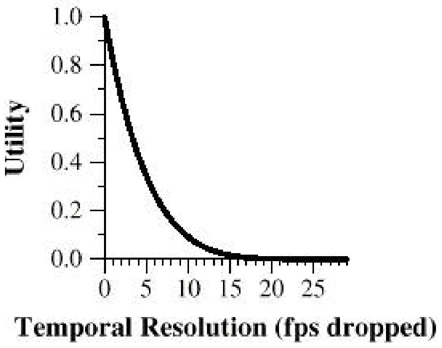
Resource Requirements vs Priority

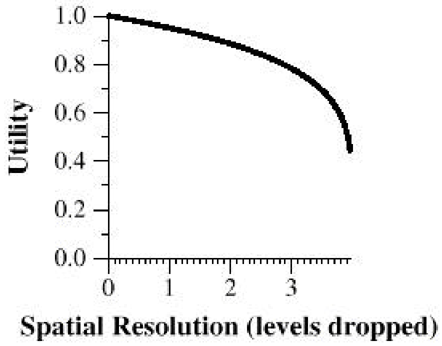


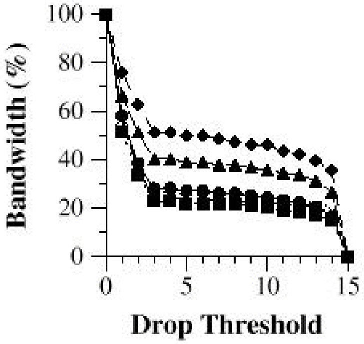
Measured network bandwidth and CPU time required for each priority level

Adaptation range is wide and smooth for both resources

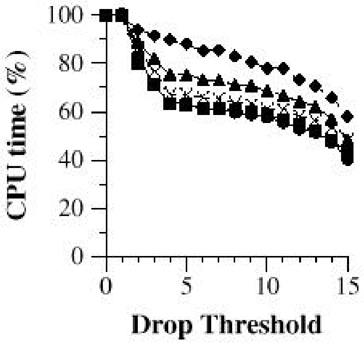
Alternate Policy Example







(c) Bitrate



(d) CPU Consumption

Preferences chosen to give extreme bias to temporal QoS

- Frame dropping is more effective for CPU reduction than SNR dropping!
- Choice of utility functions has important effects on range of adaptation

Summary of Part 1

Informed dropping enables video to support a *wide range* of operating points with *fine-granularity*

Quality is multi-dimensional and the best mix of adaptations is content, task, user or device specific

- Adaptation should be *tailorable*

Our contributions so far:

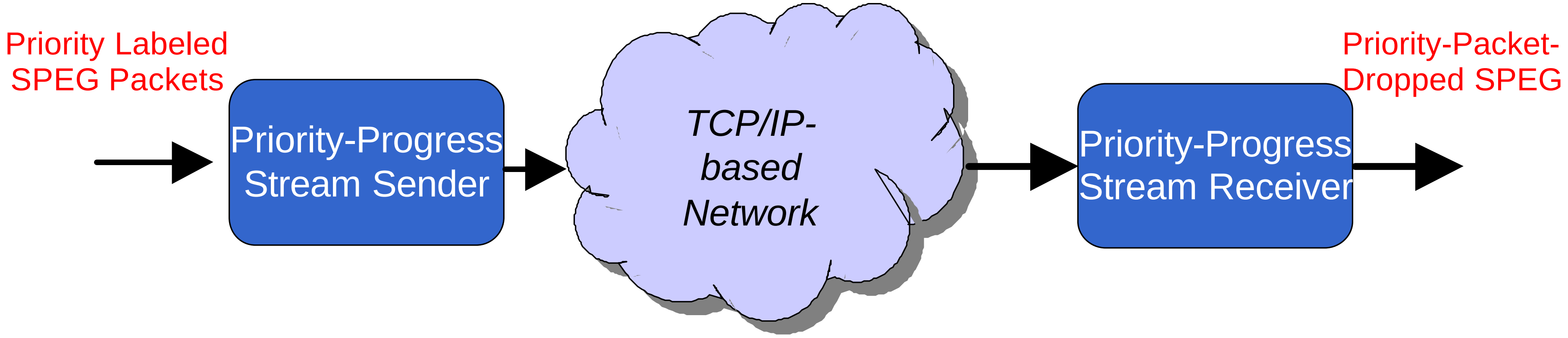
- A scalable video encoding
- A priority-mapper that supports efficient, effective, and tailorable adaptation [WCDS'99]

Part 2: Video Streaming over TCP

Priority-Progress Streaming (PPS)

Goals

- Match video rate to available bandwidth (TCP's sending rate!)
- Combine priority and timing information to decide what video data to send, when to send it, what to drop and when to drop it

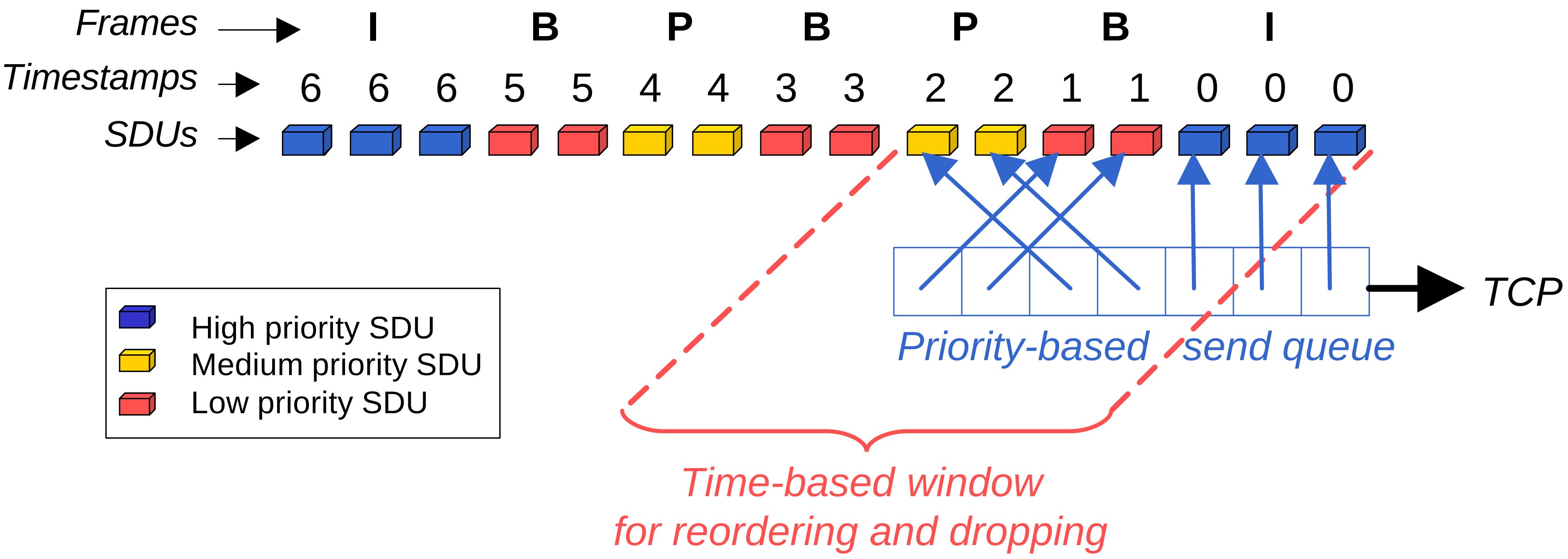


Priority-Progress Sender

Priority-Progress Streaming is a window and clock-based algorithm

Stream Data Units (SDUs) within a time window are sent in priority order

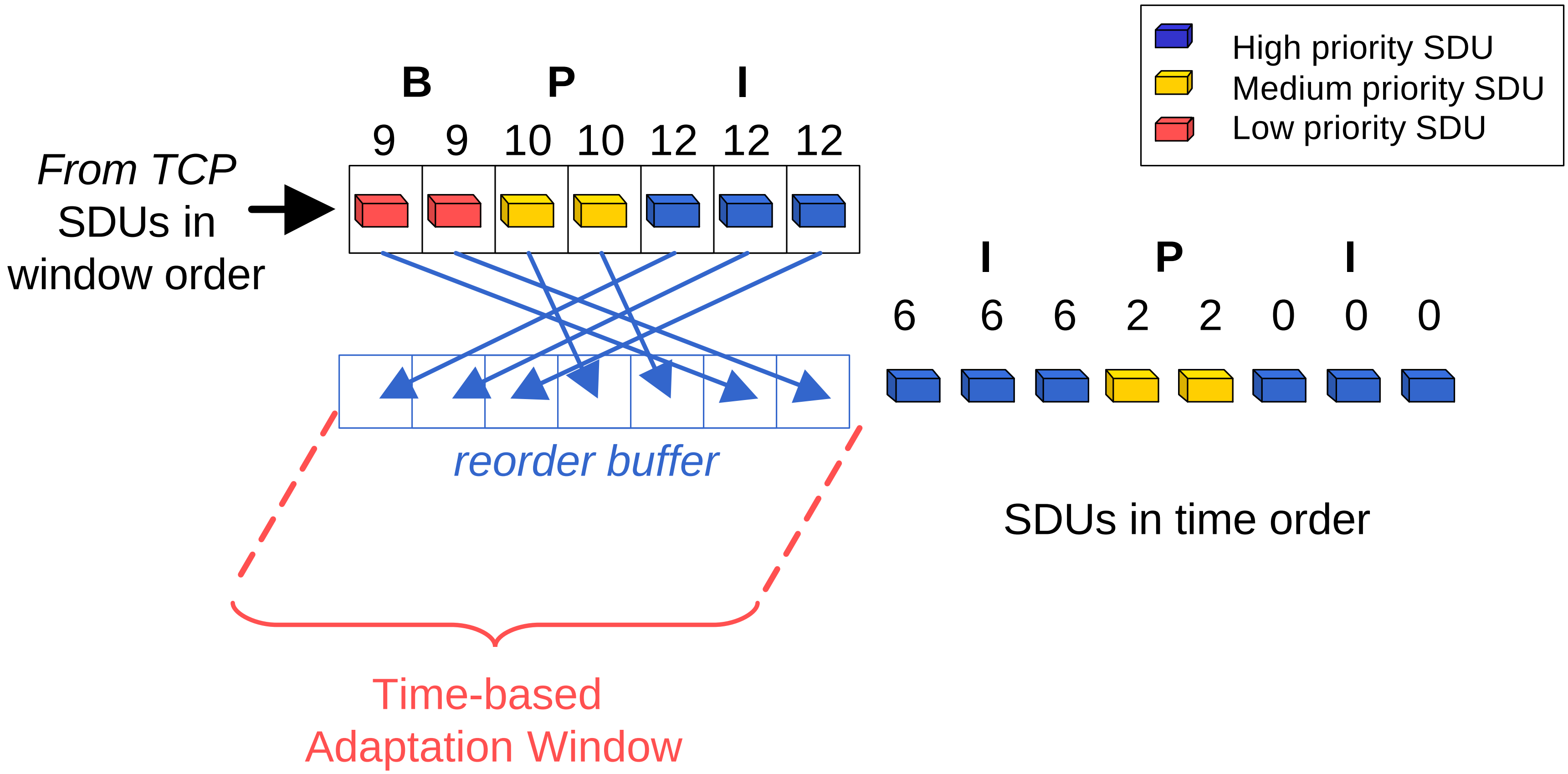
Unsent SDUs dropped as window advances based on clock



Priority-Progress Receiver

Reestablish time order for SDUs received

End of window marker commits each window for display



From TCP
SDUs in
window order

- 🟦 High priority SDU
- 🟨 Medium priority SDU
- 🟥 Low priority SDU

reorder buffer

SDUs in time order

Time-based
Adaptation Window

Performance?

The length of PPS adaptation windows determines

- Responsiveness: how quickly does video react to user input?
- Smoothness of quality: how often does quality change?

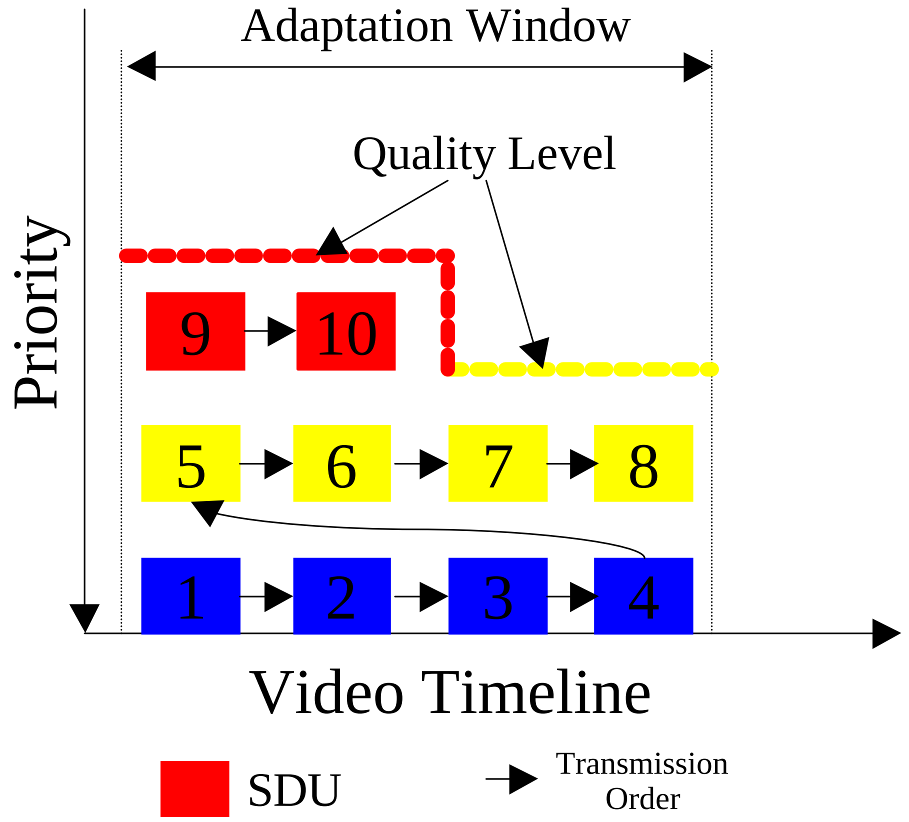
Short windows allow high responsiveness

- Window transmission and display are sequential in PPS
- End-to-end delay is approximately twice the window length

Large windows allow more consistent quality

- The number of quality changes is directly bounded by the number of windows (max of 2 quality changes per window)

Impact of Window Size on Quality Variation



Number of quality levels is at most twice the number of windows, and is independent of network bandwidth variations!

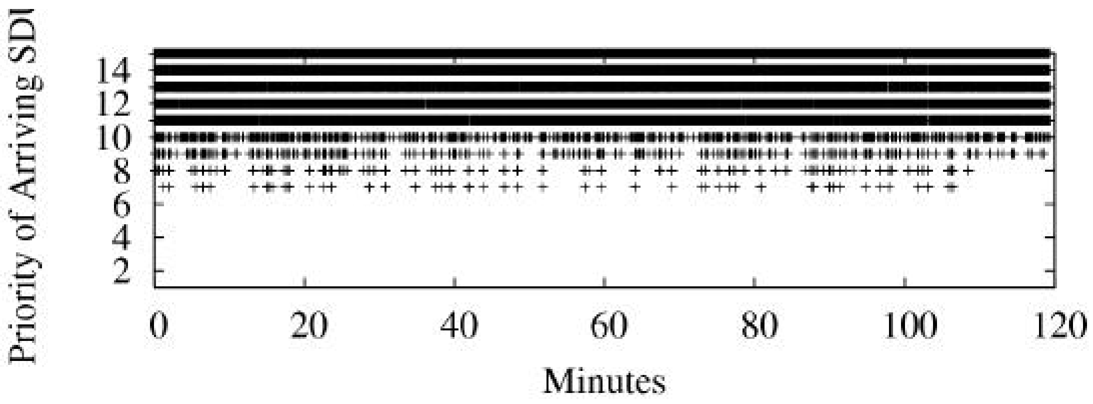
Adaptive Window Scaling

For stored video, end to end latency only matters at startup

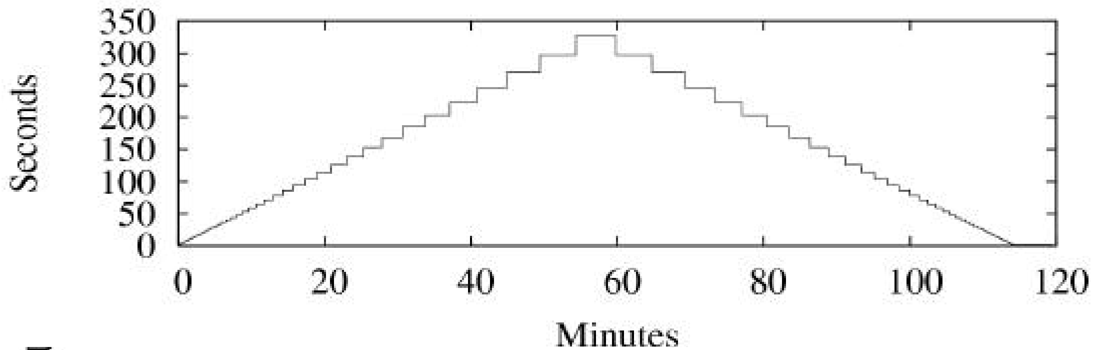
Bandwidth skimming allows a small startup window to grow into a large normal playback window

- Modest skimming (<10%) is very effective
- Dramatic improvements in quality smoothness

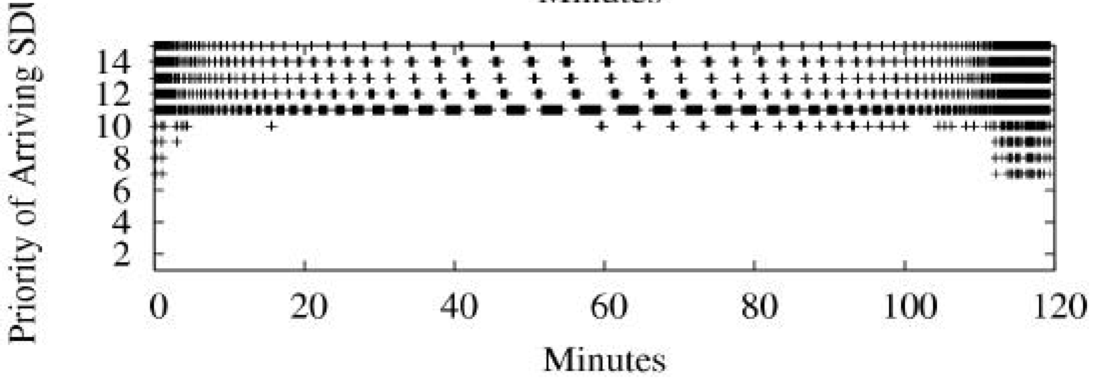
Effect of Fixed vs Adaptive Windows



- SDU arrivals by priority, frontier sets final quality level for each window



- Window size with a 10% skim rate



- SDU arrivals with 10% skim rate, far fewer changes in middle

Priority-Progress Experiments

Experiments run on a network test bed in the OGI/SySL lab

- 12 x 1U Servers (Pentium IV Xeon)
- CISCO 4000 Gigabit Switch

MxTraf traffic generator saturates link with mix of traffic flows

- Elephants (infinite greedy TCP flows)
- Mice (periodic short TCP flows)
- Dinosaurs (non-responsive background UDP)

NISTNet used to emulate a wide area path:

- Add delay and bandwidth limitations

Experiment Parameters

NISTNet:

- 50ms rtt with 25Mbit/sec rate
- tail-drop queue with limit set to bandwidth-delay product

MxTraf background traffic mix:

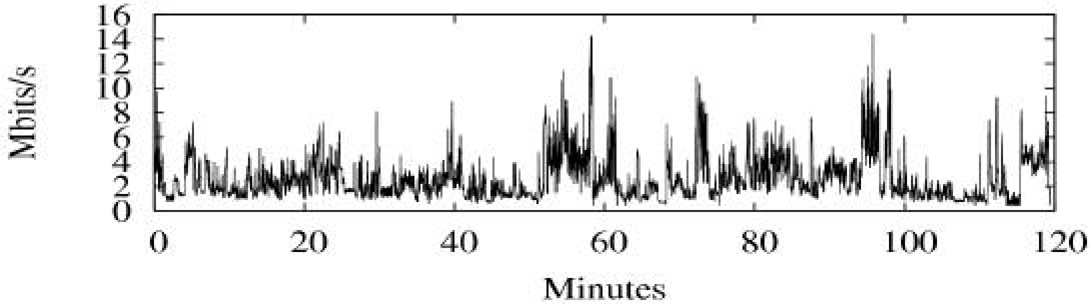
- 10 % UDP, 60 % Mice, 30 % elephants

Baselines: CMT and Feng Streaming Algorithms

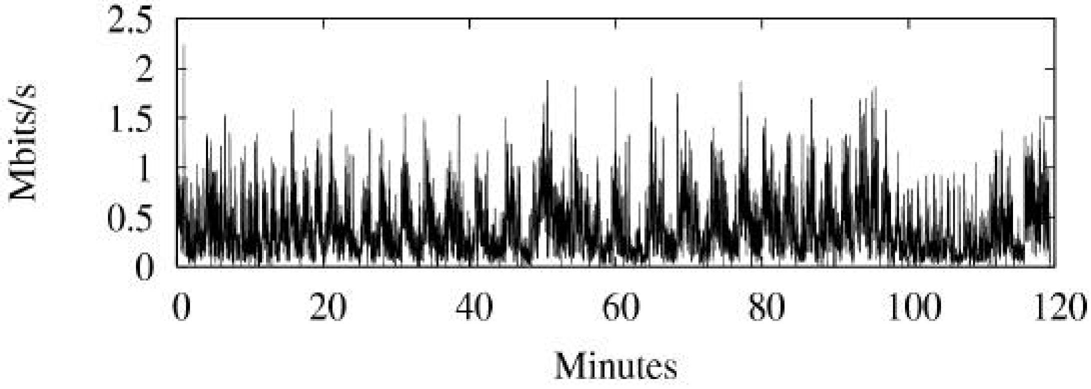
QStream

- 2 hour SPEG movie (Crouching Tiger Hidden Dragon)
- Balanced adaptation policy, fixed and adaptive window size

Video and Network Rates

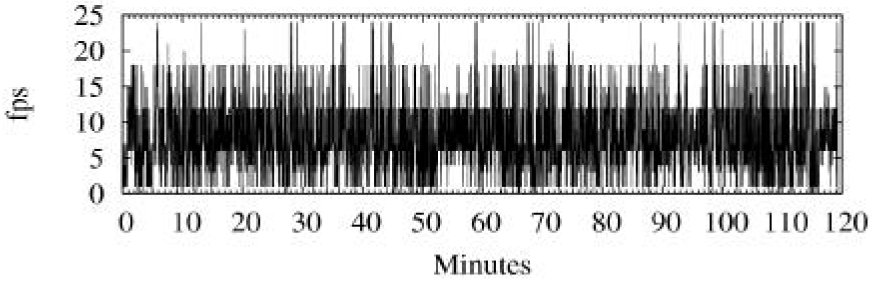


Maximum Video Rate (smoothed to 1s intervals)

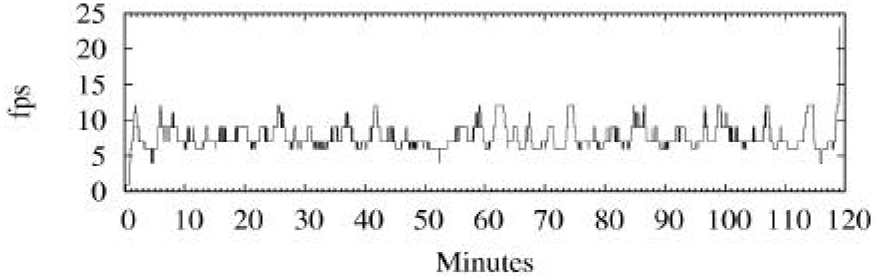


TCP Transmission Rate (smoothed to 1s intervals)

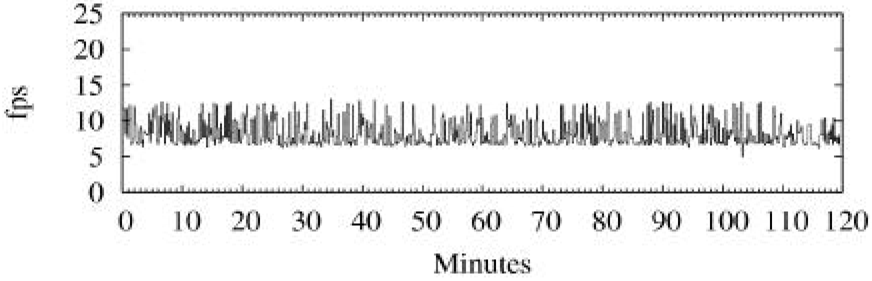
Temporal Quality



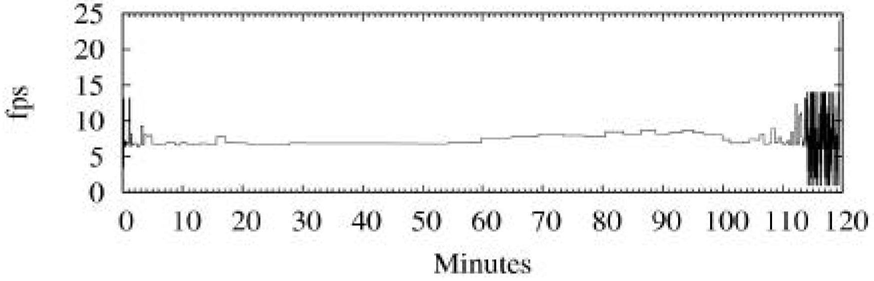
CMT (2s buffer)



Feng Priority Window (60s window)

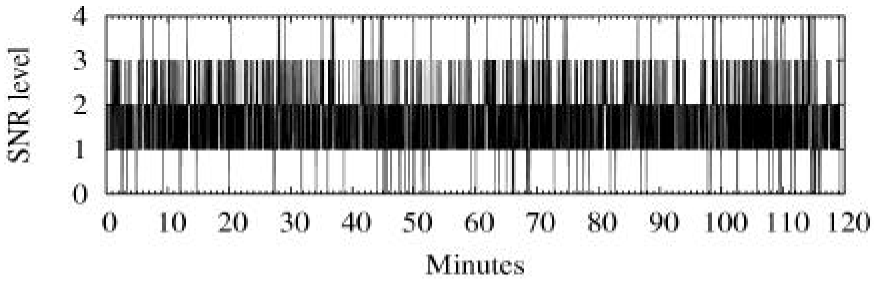


PPS (10s fixed window)

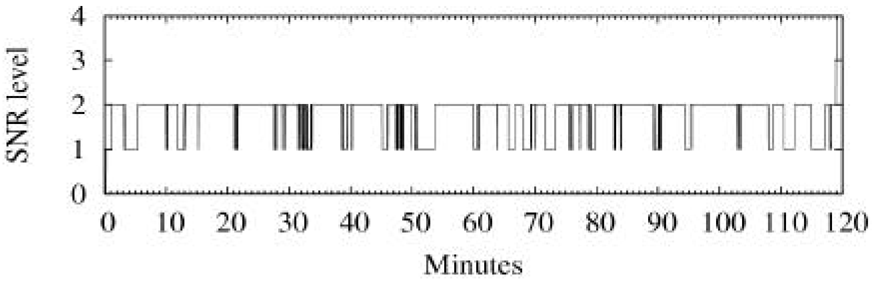


PPS (10% window scaling)

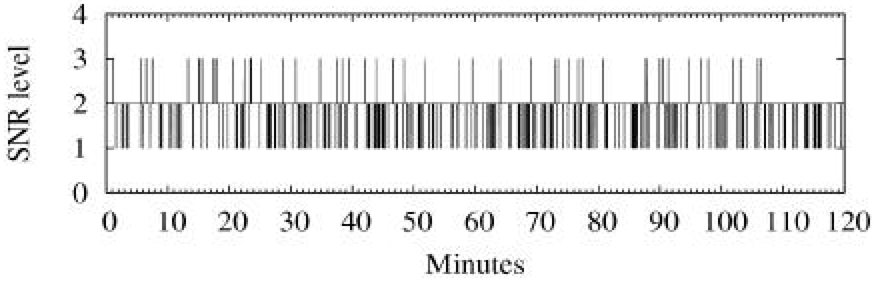
Spatial Quality



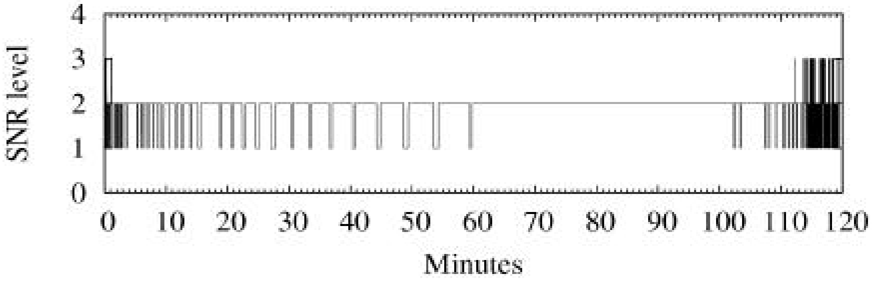
CMT (2s Buffer)



Feng Priority Window (60s window)



PPS (10s fixed window)



PPS (10 % window scaling)

Summary of Part 2

Unicast streaming solution [NOSSDAV 2003]

- TCP-friendly by actually using TCP!
- Could easily use other TCP-Friendly transports too
- Rapid and fine-grain response to bandwidth variations
  - Fully utilizes fair share of bandwidth
- Balance between responsive startup and consistent quality
  - The longer the video, the more consistent quality will become

Part 3 TCP-Friendly Multicast Video Streaming

Multicast Video Streaming: Goals

Ubiquitous access to continuous media streams from a wide range of devices over a wide range of link capacities

Efficient use of bandwidth

- Emulate broadcast where synchronized delivery enables sharing

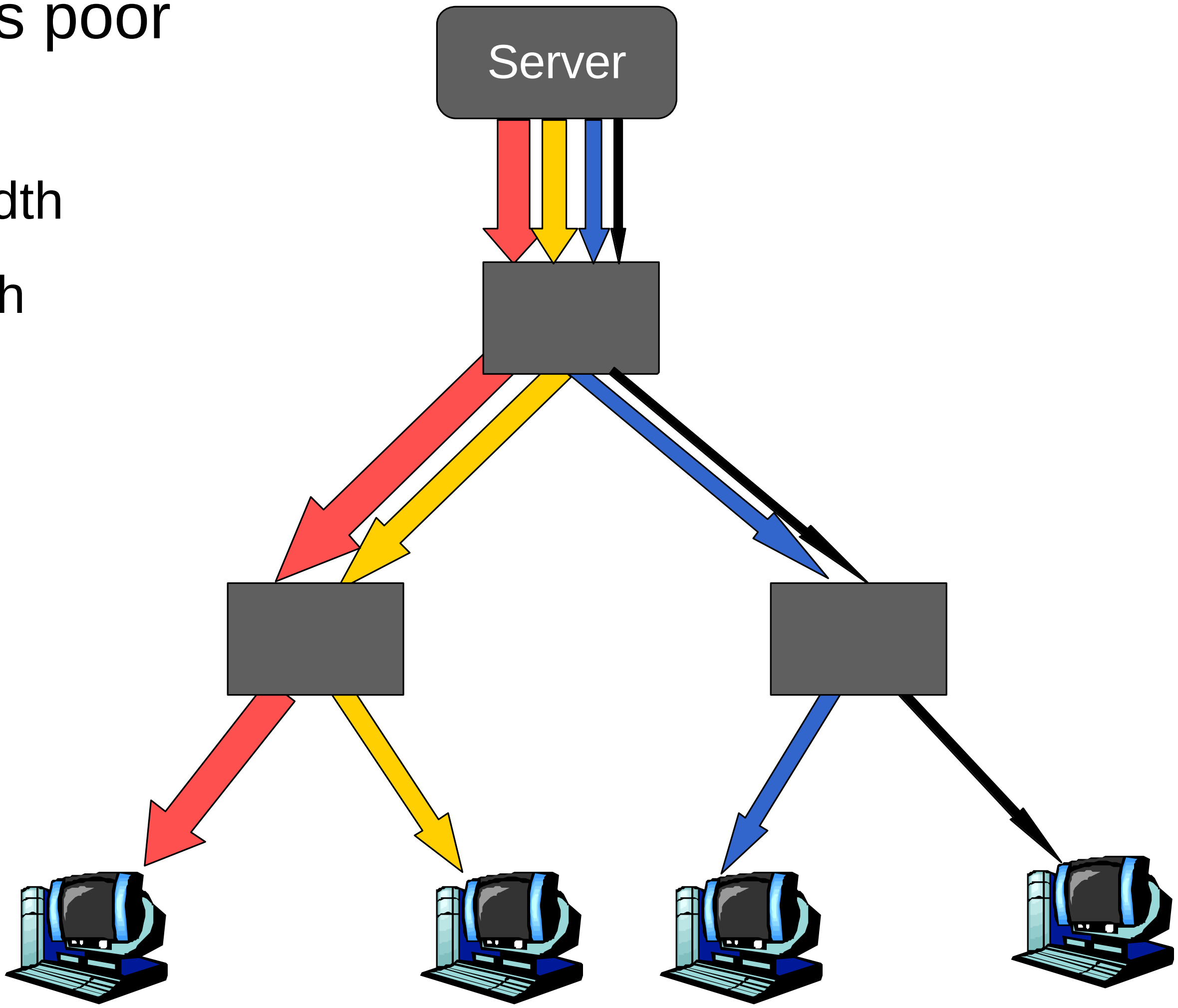
TCP-friendliness

Graceful quality adaptation

Multicast Video Streaming Problem

Unicast delivery has poor scalability

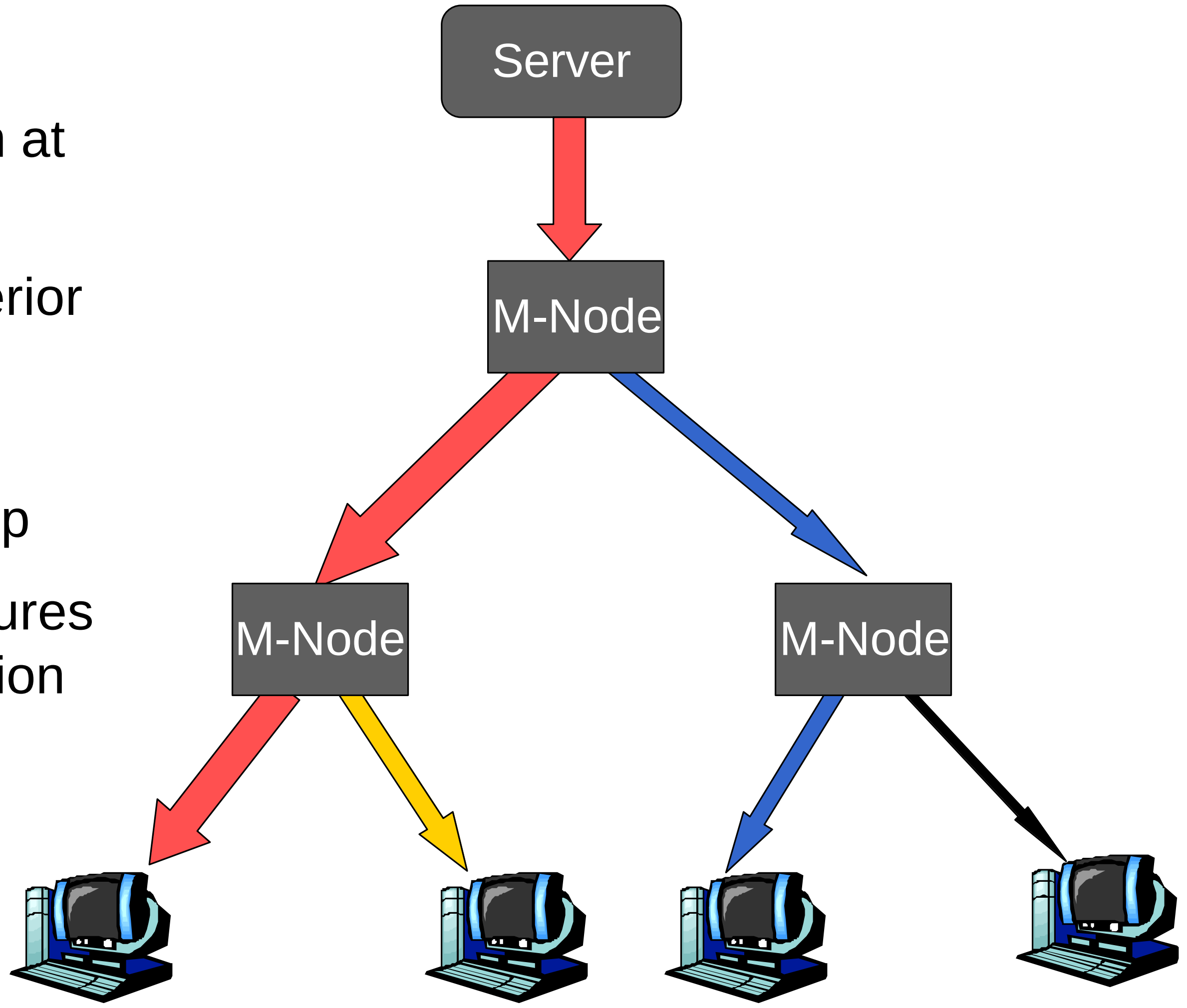
- Network bandwidth
- Server bandwidth
- Server storage
- Administration



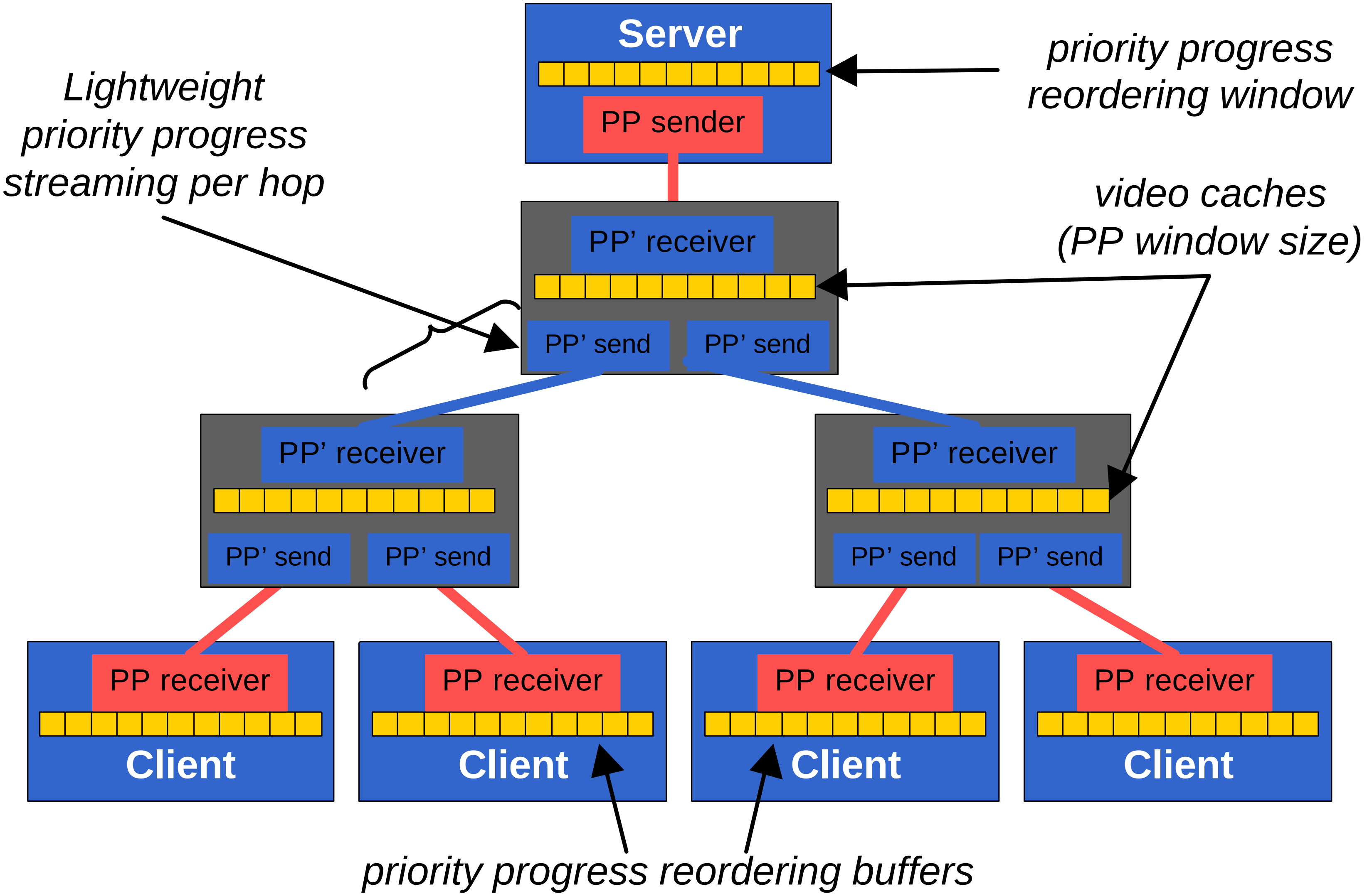
Multicast Video Streaming Goal

Adaptive multicast

- One high quality stream at server
- Duplicate stream at interior nodes
- Match rate to “fair” bandwidth share per hop
- PPS on each edge ensures graceful quality adaptation



Priority-Progress Multicast (PPM) Overlay Network



Lightweight priority progress streaming per hop

priority progress reordering window

video caches (PP window size)

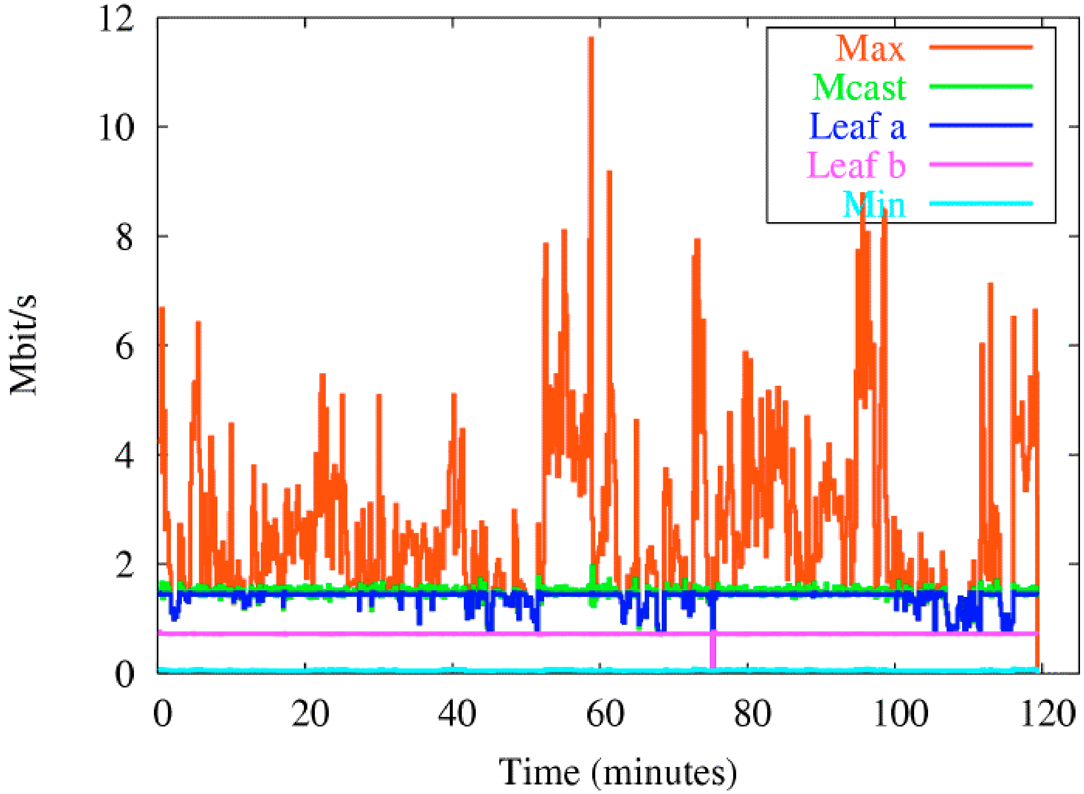
priority progress reordering buffers

Priority-Progress Multicast Node

Multicast nodes receive and forward SDUs

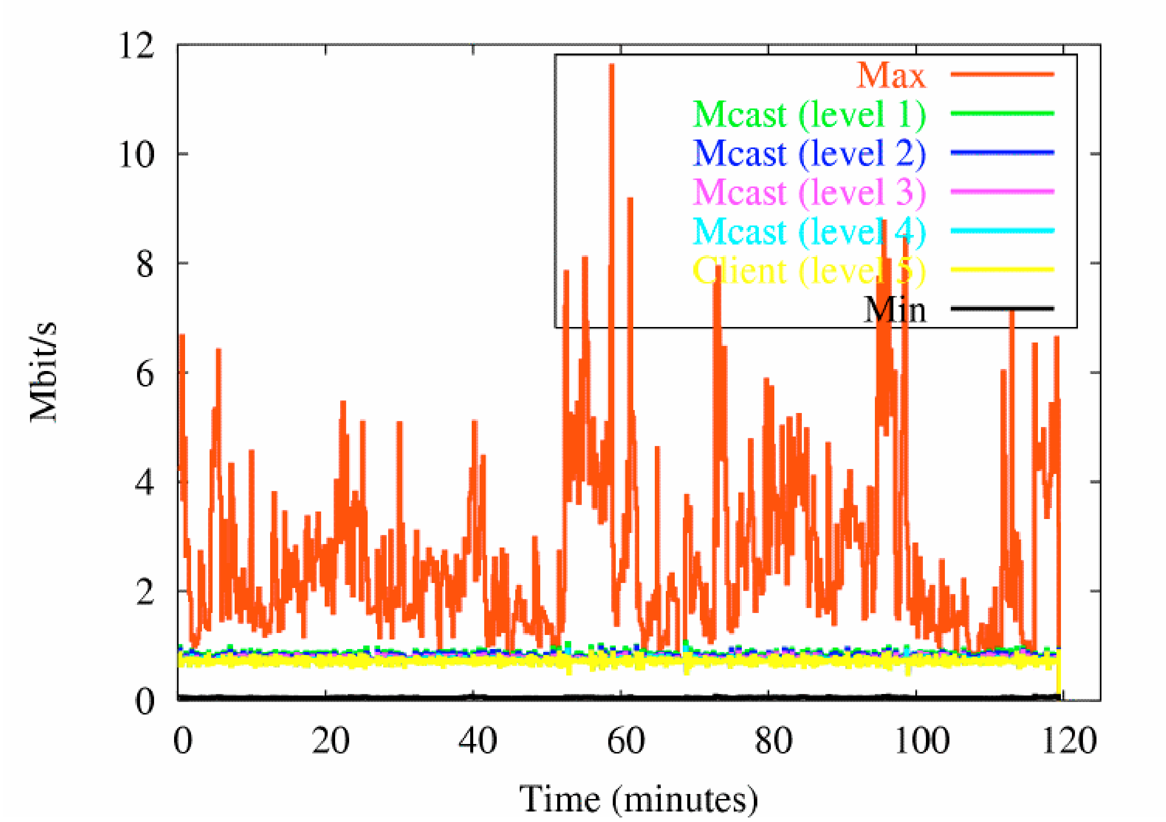
- Video cache size = PPS adaptation window
  - Arrival of start of new window triggers dropping of unsent SDUs (cache flush)
  - SDUs forwarded in FIFO order (priority)
- Sending rate on each outgoing branch regulated by congestion control
  - SDU dropping matches video rates to available bandwidth per downstream branch
- Receive rate on upstream edge regulated by PPM flow control
  - Goal is to match upstream to maximum downstream rate

Experiment 1: Basic Adaptation



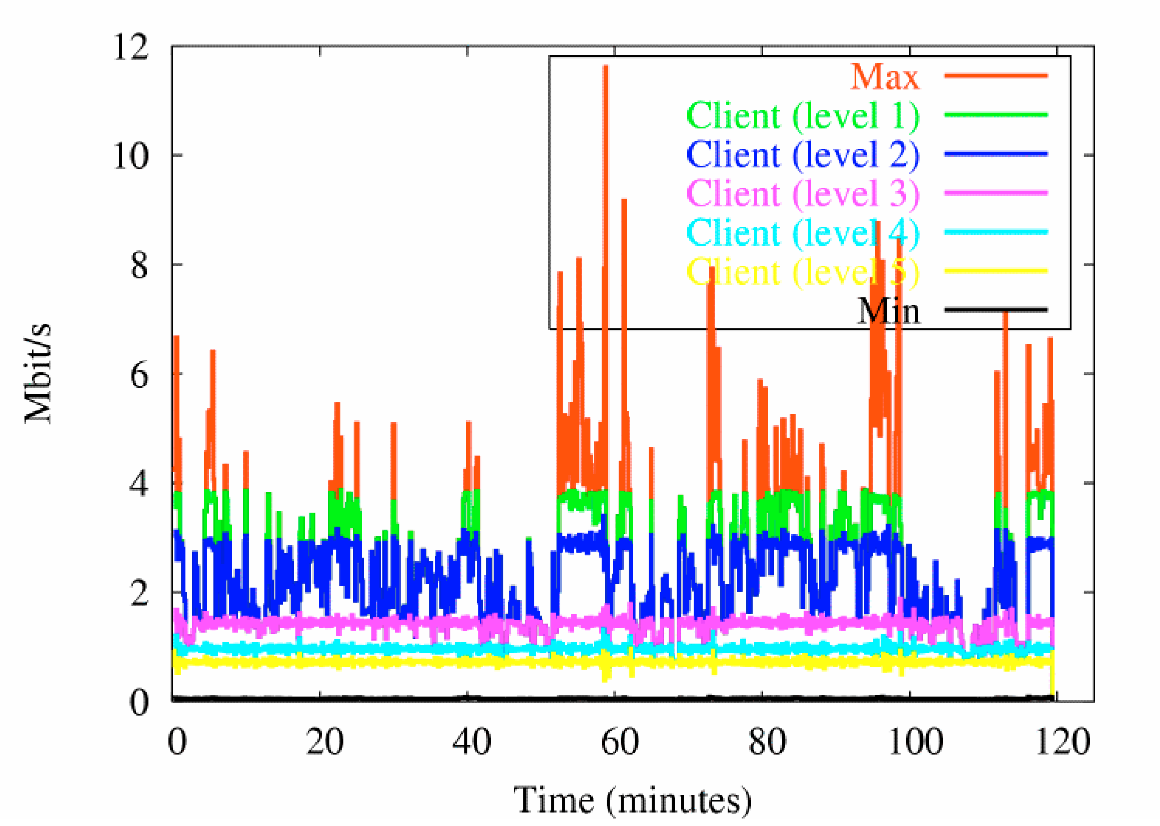
- Two children with different link capacities (0.75Mbs, 1.5Mbs)

Experiment 2: Flow-control Leakage



- Last link (level 5) is the bottleneck
- Flow control effective at limiting usage on upstream links

Experiment 3: Utilization



- Each level has a client that drains at the full rate of the upstream link (4Mbps, 3Mbps, 1.5Mbs, 1Mbs, 0.75Mbs...)
- PPS adaptation achieves full utilization and upstream bandwidth conservation

The Costs

State per multicast session required at interior nodes

- Still much better than state per client though

Data buffering at each hop, but with fixed upper bound

- Buffer size = PPS adaptation window size
- Buffer size aggregates over upstream rates of active sessions
  - 1 Gb group requires same cache space as 1000 1Mb groups
- Tunable, but probably quite large buffers
  - on the order of seconds
  - ~128 MB per second with 1Gb aggregate video rate

End to end latency

- Determined by window size

The Benefits

Single file and single stream at server

- Enables highly scalable servers

Tailorable video quality adaptation

- client-specific data rate for every multicast client
- content-specific policy (dynamically adaptable)

Find-grain, wide-spectrum adaptation

- full link utilisation, optimal quality, TCP-friendly multicast

Highly scalable lightweight forwarding algorithm

- Gigabit rates on modern commodity hardware (Intel IXP)

Conclusions and Future Work

Qstream: TCP-friendly, multicast streaming for VoD applications

- “*Encode once, stream anywhere*”
- Built it, tried it, tested it, … and it really works!

On-going and future work:

- Live video sources [PV 2003]
- Low latency applications [AVSS 2003]
- Alternate transports [IWQoS 2002, IDMS 2001]
- Power-aware video capture and distribution [ACM Multimedia 2004]
- Peer to peer video streaming
- Region of interest adaptation
- Virtual pan/tilt/zoom for interactive surveillance applications

Other Work

Time Sensitive Linux
- [OSDI 2002, RTAS 2002]

Low-Latency Streaming with TCP
- [IWQoS 2002, IDMS 2001]

Infopipes - Streaming Middleware
- [MM Systems 2002, SP&E 2003]

SWIFT - Feedback Control Models
- [RTSS 2002]

Environmental Observation Systems
- CORIE [ISEIS'2003]

Specialization of Systems Software
- [TOCS 2001]

Tools:

Gscope - gscope.sourceforge.net
- [USENIX/FREENIX 2002]

MxTraf - mxtraf.sourceforge.net

LibDV - libdv.sourceforge.net
- 75000+ direct downloads

Demos

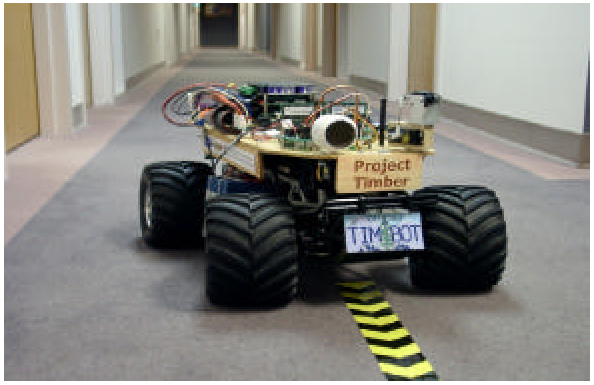
Extra Slides

Pipeline Latency

Future Work

Language based support for real-time development

- Project Timber at OGI



Applications

- Robotics
- Sensor networks for environmental observation
- Tele-presence for distance medicine

Related Work

Quality Adaptive Streaming

- Feng, Rejaie, Feamster,...

QoS for multimedia

- RTP, RSVP, DiffServ

Media friendly transports

- TFRC, TEAR, RAP, etc

Related Work (con't)

Fine Granularity Scalability

- MPEG-4 FGS, PFGS

Adaptive multicast

- RLM, FIDL-DL

Multicast Overlays

- End-system Multicast

SPEG Encoder Structure