網路串流技術 (II)

鄭憲宗

資訊工程學系

國立成功大學
Outline

3. Streaming Techniques:
   3.1. Flow Control and Error Recovery
   3.2. Buffer Management

4. Server & Client-side Issues:
   4.1. 影音同步議題
   4.2. 多媒體串流技術: 可調式壓縮 (scalable coding)
   4.3. flash streaming
Flow Control and Error Recovery
Outlines

– Introduction
– Flow and Congestion Control
– Error Control
Introduction

• Video and audio streaming over the Internet becomes popular
  – As last mile network bandwidth increases (ADSL, Cable modems, Satellite), multimedia traffic will constitute a large portion of Internet traffic.
  – The VoD market will grow accordingly.

• There is no quality of service (Qos) guarantee for video transmission over the current Internet.
Challenging QoS issues

• Bandwidth
  – Current Internet does not provide bandwidth reservation

• Delay
  – Real-time video requires bounded end-to-end delay. Internet does not offer delay guarantee

• Packet Loss
  – Packet loss ratio required to be kept below a threshold. Internet does not provide any loss guarantee
General Approaches

• Network-centric
  – Routers/switches in the network are required to provide QoS support.

• End system based
  – Guarantee QoS without imposing any requirements on the network.
Flow and Congestion Control

• How can we make the best use of the (time varying) bandwidth that is available to our streams?
  – How can we determine what this bandwidth is?
  – How can we track how it changes over time?

• Rate control
• Rate-adaptive and video encoding
• Rate shaping
Rate Control

• Windows-based
  - 類似TCP，傳送端之資料速率不超過接收端之接收能力，傳輸速率由擁塞窗框(congestion window)所控制
  - 以封包遺失或逾時當作網路擁塞的指標
  - 資料傳輸中若有封包遺失或逾時，TCP就會啟動擁塞控制機制快速降低資料傳輸速率

• Rate-based
  - Source-based
  - Receiver-based
  - Hybrid rate control
Windows-Based Rate Control

- Slow Start (CWND < Threshold)
  - 探測目前網路可承載的頻寬
  - 當connection建立以後，CWND大小以指數的速度成長，直至超過Threshold或封包遺失產生為止
Source-Based Rate Control (1)

• The sender is responsible for adapting the transmission rate of the video stream.
• It can minimize the amount of packet loss by matching the rate of the video stream to the available network bandwidth.
• Feedback is employed to convey the changing status of the Internet.
Source-Based Rate Control(2)

- Probed-based approach
  - Additive Increase and Multiplicative Decrease (AIMD)

\[
\begin{align*}
\text{if } (p \leq P_{th}) & \\
r & := \min\{(r + \text{AIR}), \text{MaxR}\} \\
\text{else} & \\
r & := \max\{\alpha \times r, \text{MinR}\}
\end{align*}
\]

- \(p\): packet loss ratio
- \(P_{th}\): threshold for the packet loss ratio
- \(\text{AIR}\): additive increase rate
- \(r\): sending rate at the source
- \(\alpha\): multiplicative decrease factor
Source-Based Rate Control(3)

- Source behavior under the AIMD rate control

![Graph showing rate vs time]
Source-Based Rate Control(4)

• Model-based approach
  – Also called TCP friendly rate control

\[ \lambda = \frac{1.22 \times MTU}{RTT \times \sqrt{p}} \]

- \( \lambda \): throughput of a TCP connection
- MTU: maximum transit unit
- RTT: round trip time
- \( p \): packet loss ratio
Receiver-Based Rate Control(1)

- Target at solving heterogeneity problem
- Layered multicast video
- Probe-based approach and model-based approach
Receiver-Based Rate Control (2)

- Layered Multicast Video
  - Raw video sequence is compressed into multi layers
  - A base layer and one or more enhancement layers
Receiver-Based Rate Control(3)

• Probe-Based Approach
  – When no congestion
    • A receiver probes for the available bandwidth by joining a layer, which leads to an increase of its receiving rate.
  – When congestion is detected
    • The receiver drops a layer, resulting in reduction of its receiving rate.
$\gamma_i$: transmission rate of layer $i$
Hybrid Rate Control

• Receiver regulate the receiving rate of video streams by adding/dropping channels while sender also adjusts the transmission rate of each channel based on feedback information from the receiver.
Rate-Adaptive and Video Encoding

• A compression approach
  – Video conference with H.261, H.263, MPEG-1, MPEG-2, and MPEG-4
  – Maximize the perceptual quality under a given encoding rate.
Rate Shaping(1)

• A rate shaper is a filter between the encoder and the network, with which the encoder’s output rate can be matched to the available network bandwidth.
• It is applicable to any video coding scheme and both live and stored video
Introduction

• Video and audio streaming over the Internet becomes popular
  – As last mile network bandwidth increases (ADSL, Cable modems, Satellite), multimedia traffic will constitute a large portion of Internet traffic.
  – The VoD market will grow accordingly (e.g., AOL + TW).
Rate Shaping(2)

- Transport perspective
  - Server selective frame discard
- Two advantage
  - Taking the network bandwidth and client buffer constraint into account
  - Take advantage of application-specific information such as regions of interest and group of pictures structure, in its decision in discarding frames.
Rate Shaping(3)

• Compression perspective
  – Based on the Rate-Distortion theory, the dynamic rate shaper selectively discards the discrete cosine transform (DCT) coefficients of the high frequency so that the target rate can be achieved.
Error Control

• Forward error correction (FEC)
• Retransmission
• Error resilience
• Error concealment
FEC

- Add extra information to a compressed video bit stream
  - Channel coding
  - Source coding-based FEC
Channel Coding FEC

- **Hierarchical FEC**
  - The FEC stream is used for recovery of a different video layer
  - More flexibility and bandwidth efficiency
- **Unequal error protection**
  - In MPEG  $I$-frame > $P$-frame > $B$-frame
- **Disadvantages**
  - It increases the transmission rate and delay
  - It is not adaptive to varying loss characteristics and works best only when the packet loss rate is stable.
Source Coding-Based FEC

- The redundant information added by SFEC is more compressed versions of the raw data
- SFEC recovers the video with reduced quality
- Advantage
  - Lower delay
- Disadvantage
  - It increases the transmission rate and is inflexible to varying loss character
Receiver-Based Retransmission

Fig. 15. Timing diagram for receiver-based control.

When the receiver detects the loss of packet $N$:

\[
\text{if } (T_r + \text{RTT} + D_s < T_d(N))
\]

send the request for retransmission of packet $N$ to the sender.
Sender-Based Retransmission

Fig. 16. Timing diagram for sender-based control.

When the sender receives a request for retransmission of packet $N$:

$$\text{if } (T_c + T_o + D_s < T_d'(N))$$

retransmit packet $N$ to the receiver.
Error Resilience

• A compression approach
  – Prevent error propagation or limit the scope if the damage
  – Optimal mode selection
  – Multiple description coding
    • A raw video sequence is compressed into multiple streams
    • Advantages
      – Robustness to loss: even if a receiver gets only one description, it can still reconstruct video with acceptable quality
      – Enhance quality
Error Concealment

- A compression approach
  - Human eyes can tolerate a certain degree of distortion in video signals, error concealment is a viable technique to handle packet loss.
  - Spatial and temporal interpolation
- The receiver replaces the whole frame with the previous reconstructed frame
- The receiver replaces a corrupted block with the block at the same location from the previous frame
- The receiver replaces the corrupted block with the block from the previous frame pointed by a motion vector
References

Buffer Management
Layered Code in Multimedia Systems

<table>
<thead>
<tr>
<th>SENDER</th>
<th>RECEIVER</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sampling/Encoding</td>
<td>Decoding/Playout</td>
</tr>
<tr>
<td>Streaming Application</td>
<td>Streaming Application</td>
</tr>
<tr>
<td>Real-time Transport Protocol</td>
<td>Real-time Transport Protocol (De-Packetization)</td>
</tr>
<tr>
<td>(RTP)</td>
<td>UDP</td>
</tr>
<tr>
<td>(Packetization)</td>
<td>IP</td>
</tr>
<tr>
<td>UDP</td>
<td>Ethernet</td>
</tr>
<tr>
<td>IP</td>
<td></td>
</tr>
<tr>
<td>Ethernet</td>
<td></td>
</tr>
</tbody>
</table>

Department of Computer Science and Information Engineering
Wireless and Mobile Communications Lab.
End-to-end Processing and Transmission of Digital Media Signals

- Sampling A/D
- Encoder
- Packetization
- Network
- Decoder
- De-packetization

Analog Media signal

$ps(t)$ -> $e(t)$

$ps(t)$ -> $b_s(t)$

$pr(t)$ -> $dr(t)$

$pr(t)$ -> $br(t)$

Department of Computer Science and Information Engineering
Wireless and Mobile Communications Lab.
Example: Sender and Receiver Curves for transmission of CBR signal (digital speech with 64 kbps over a circuit-switched network)
Sender/Receiver Curves for Transmission of Voice over Packet-switched network

- $p_s(t)$
- $b_s(t)$
- $r(t)$
- $p_r(t)$

Late loss

Department of Computer Science and Information Engineering
Wireless and Mobile Communications Lab.
Outline

• Protocol Requirements on Buffer Management
• Buffer Management
  – Data Copying
  – Offset Management
  – Scatter/Gather System
• Buffering Strategies
  – Minbuf
  – Maxbuf
Buffer Management

- Buffers can be views as **spatial representation** of time
- Buffer plays very important role in **smoothing** traffic
- Network protocols buffer their **service data units (SDUs)** and use **data copying** when going from one protocol layer to another
- Moving data using data copying is very **expensive**
# Layered Code in Multimedia Systems

## Sampling/Encoding
- Streaming Application
- Real-time Transport Protocol (Packetization/And Segmentation)
  - UDP
  - IP
  - Ethernet

## Decoding/Playout
- Streaming Application
- Real-time Transport Protocol (De-Packetization And re-assembly)
  - UDP
  - IP
  - Ethernet

---

Department of Computer Science and Information Engineering
Wireless and Mobile Communications Lab.
Protocol Requirements on Buffer Management and Segmentation

Protocol Requirements

Service Data Unit (SDU)

Example:
Application Data Unit – Video I frame That comes out of encoder

Buffer Management:
Keep identification (PCI) To which application Data unit the segment belong

Example:
I frame gets split into two RTP datagrams

Department of Computer Science and Information Engineering
Wireless and Mobile Communications Lab.
Reassembly and Retransmission
Requirements on Buffer Management

Buffer Management: support linking of memory to form one buffer

REASSEMBLY

PCI SDU1 ► PCI SDU2

RETRANSMISSION

Buffer Management: logical copy of buffer must exist to store SDUs for Possible retransmission
Multi-cast and Multi-target requirements on Buffer Management

**Multicast**

- Buffer Management: Keep only One buffer for all recipients
- Use Multicast Group address

**Multi-Target**

- Buffer Management: Keep one memory segment Common to all buffers

---

Department of Computer Science and Information Engineering Wireless and Mobile Communications Lab.
Buffer Management Techniques

DATA COPYING

Problems:???

OFFSET MANAGEMENT

Buffer Management: Assign as large buffer as data + all headers of the protocols require

Problems:???
Buffer Management Techniques (Scatter-Gather)

Data | PTR1
--- | ---
PCI1 | PTR2
PCI2 | PTR3

Scatter/Gather Table Structure

DATA
PCI1
PCI2

Application Buffer Space
RTP Buffer Space
UDP Buffer Kernel Space

Problems: ????

Department of Computer Science and Information Engineering
Wireless and Mobile Communications Lab.
## Comparison

<table>
<thead>
<tr>
<th></th>
<th>Data Copying</th>
<th>Offset</th>
<th>Scatter/Gather</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory BW</td>
<td>High</td>
<td>Low</td>
<td>Low</td>
</tr>
<tr>
<td>CPU BW</td>
<td>High</td>
<td>Low</td>
<td>Low</td>
</tr>
<tr>
<td>Memory Usage per Layer</td>
<td>Optimal for individual protocol layer because exact amount of space will be allocated</td>
<td>High for application protocol because it must allocate space more than it needs</td>
<td>Compromise, segments are sized depending on requirements</td>
</tr>
<tr>
<td>Are Protocol Requirements without copying satisfied?</td>
<td>NA</td>
<td>No, segmentation needs copying</td>
<td>Mostly yes (one copy and segmentation must be done when data leaves node)</td>
</tr>
</tbody>
</table>
Buffering Strategies in Client-Server Systems

- Read encoded frames from VOD Disk
- Packetization Protocol Stack Processing
- VOD Server
- De-Packetization Protocol Stack Processing
- VOD Client
- Decoder
- playout

Department of Computer Science and Information Engineering
Wireless and Mobile Communications Lab.
VOD Retrieval Transmission and Playout Curves

- Retrieval curve from disk
- Sending curve from VOD server
- Receiving curve at VOD client
- Buffers
- Playout curve at VOD client

Bytes vs. time
Buffer Management Strategies

- Clients need **buffers** for VOD (Video-on-Demand) application to smooth out traffic jitters
- Buffer management strategies balance **bits in transit** (buffer size and bits in channel)
- **Fixed Strategy** (non-adaptive) and **dynamic** (adaptive)
Buffer Management Strategies

• **Fixed buffer strategy**
  – Static buffer allocation during multimedia call setup phase
  – Static buffer allocation does not change during run-time

• **Dynamic buffer strategy**
  – Elastic buffer allocation, i.e., allocate buffers during multimedia call setup, but change them during run-time
Buffering Strategies at VOD Client

• **Minbuf** - minimum buffering strategy
  – Minbuf minimizes buffering requirements at VOD client, but makes more demands on network (throughput and delay guarantees)

• **Maxbuf** – maximum buffering strategy
  – Maxbuf buffers more than one unit of information and eases QoS guarantees demands on network
  – Buffering only up to a limit ($Buf_{max}$)
Buffering Model

- $Z_{Oi}(t)$ – amount of bits in multimedia object (e.g., Video frame) $Oi$ displayed at time $t$
- $C_r(t)$ – amount of bits received at receiver at time $t$
- Two buffer states in each buffering strategy:
  - **Starvation** if $C_r(t) \leq Z_{Oi}(t)$
    - If $C_r(t) > Z_{Oi}(t)$, no starvation at VOD client
  - **Overflow** if $C_r(t) \geq Z_{Oi}(t) + Buf_{max}$
Client Buffer Constraints

(Received amount of data $C_r(t)$ should always be between $Z_{O_i}(t)$ and $Z_{O_i}(t) + \text{Buf}_{\text{max}}$)
Implications of Minbuf Strategy

- With minbuf strategy – delivery time of a unit of information is the display time of previous unit
- Minimum Throughput (B) required is

\[ B_{\text{min}}^{\text{buf}} \geq \frac{Z_{Oi}}{t_i - t_{i-1}} \]

- Delivery time instant of the first unit before start of display:

\[ t_x = \frac{Z_{Oi}}{B_{\text{min}}^{\text{buf}}} \]
Implication of Maxbuf Strategy

• Delivery schedule of VOD server and network may cause that $Buf_{max}$ bits will be delivered every $K$ seconds, where
  
  $K = \frac{Buf_{max}}{B}$, $B$ – throughput of network

• Minimum Throughput required
  
  $\sum Z_{Oi}$ – total size in bits of all objects that will be presented in stream:

\[
B_{min}^{maxbuf} \geq \frac{\sum_{i=1}^{n} Z_{Oi} - Buf_{max}}{StreamDisplayDuration}
\]
Implementation Issues

- Buffers for Uncompressed Periodic Streams
  - Use circular buffers as prefetch buffers with maxbuf strategy
- Buffers for compressed periodic streams
  - Use circular buffers, but carefully consider the size of buffer unit (depending on MPJEG or MPEG) with maxbuf strategy
  - For MPEG may consider dynamic buffer allocation
- Buffers for control information
  - Use priority queues or FIFO
- Buffers for Non-RT data
  - Use maxbuf strategy with static buffer allocation
Memory Management

- Virtual memory versus real memory paradigm
- In VM – *paging* introduces unpredictable delays
- Multimedia timing requirements suggest that *no paging* is desired
- Multimedia applications may want to pin pages into memory which include their time-sensitive code
- On-Chip Caching is desired
  - Intel Pentium Processor with MMX technology (SIMD Instruction Set) has on-chip cache of size 32KB for video processing only
Conclusion

• Need buffering at VOD client side
• Some buffering needed also at VOD Server side
  – Can use FIFO techniques
• Reservation memory schemes are possible
  – Implemented in system, called RK (CMU)
• Will talk about VOD server in next lectures
Audio-video synchronization
Outline

• Synchronization
• Intra-medium Synchronization
• Inter-media Synchronization
• Distributed Multimedia Information Systems
• Synchronization issue for processing user interaction
Synchronization

- Synchronization of continuous media streams
- The most prominent example for continuous media streams that need to be synchronized is the combination of audio and video
- lip-synchronization problem
Synchronization

• Eliminate all delays and variation incurred between media stream transmission and presentation.
  – Packetization delay
  – Network access delay
  – Transmission delay
  – Protocol processing delay
  – Presentation delay
Multi-step Synchronization

- Synchronization during object acquisition
  - during digitizing video
- Synchronization of retrieval
  - access frame in a stored video
- Synchronization during delivery of Logical Data Unit to network
- Synchronization during the transport
  - protocol, router in the network
- Synchronization at the sink
  - delivery to the output devices
- Synchronization within the output devices.
Intra-media Synchronization

- Recovery of the time base for an ordered list of medium samples

- Time-related issues a single medium stream has to deal with
  - Network jitter
  - End-system jitter
  - Codec clock drifts
Intra-media Synchronization

• Recovery of the time base for an ordered list of medium samples
• Synchronization Policy
  – Blocking
  – Nonblocking
Blocking

- When a necessary data unit cannot reach the client within the playback time, the client's application will continue to wait until the data unit is received before continuing playback.
- Often used in audio media.

Diagram:
- Normal playout situation: 1 2 3 4 5 6 7 ....
- Physical playout situation: 1 2 3 4 silent 5 6 7 ....
Nonblocking

• 當所需要的資料單元未能在播放時間內到達用戶端時，則用戶端的應用程式會重複上一個資料單元的播放，直到收到該資料單元後再恢復正常的播放

• 多用在視訊媒體的播放控制上
Inter-media Synchronization

• Inter-media synchronization deals with maintaining the requirements of temporal relationships among media streams

• Synchronization between video and audio
Lip synchronization Problem

- Match lip movements with voice
- Different Media (voice, video, subtitle)
  - Different transmission network channel
  - Different intra-media synchronization policy
  - Different arrival time
Synchronization Policy

• Re-synchronization
  - 延長或減短某些資料單元的播放時間
  - 放棄某些資料單元的播放
Concept Model Description of Media Synchronization

• To satisfy temporal precedence relationships

SIU (Synchronization information unit)
  – Divide each media object into a sequence of subjects with its own sync interval
  – Transmission of an object consists of a stream of SIU
  – SIU’s sync interval number as packet header information
  – ex. lip-sync are divided by the same duration
Concept Model Description of Media Synchronization

• OCPN(object composition Petri net)
  – Defined by the tuple\([T,P,A,M,D,R]\)
• Modeling intra-media synchronization with OCPN
Concept Model Description of Media Synchronization

• OCPN (object composition Petri net)
  – Inter-media: describe a temporal presentation scenario
• Temporal relationship of multimedia
• Modeling inter-media synchronization with OCPN
Synchronization issue for processing user interaction

• User interaction
  – reverse
  – skip
  – freeze-restart
  – scale

• The display of media streams may not be synchronous when a user’s interaction input is received and each medium stream’s presentation is interrupted and paused temporarily
Synchronization issue for processing user interaction

• 系統在使用者互動需求的處理過程中，必須包括再同步控制，如此可以使得在系統暫停播放，處理完同步功能並恢復播放時，可以有平順且同步的開始

• Interrupt point
  • 使用者發出互動需求時，由於video, audio及slide的播放狀況不同而有不同的中斷點

• Resume point
  • 為了讓系統在處理完該項互動需求後，能夠平順地恢復播放，我們必須求出一個相同的恢復播放點
Synchronization issue for processing user interaction

• 主控媒體
  – 系統從各種媒體資料中選擇一個媒體流做為「主控媒體」，各個媒體的同步便由此主控媒體主導

• 主控媒體的選擇
  – 人類感官的敏感度
    • 人類對聲音變化較為敏感，因此聲音常被設定為主控媒體
  – 資訊的重要性
    • 有時聲音並不是最重要的資訊，例如背景音樂，此時可以選擇影像為主控媒體
Re-synchronization

- When the presentation is interrupted and paused temporarily
回饋式的同步控制

- 伺服器端和用戶端間建立一條特殊的訊息回送通道
- 用戶端的播放程式可以送回回饋訊息給伺服器端，告知目前的播放狀況，而伺服器端的傳送程式可依此調整傳送的速率
- 藉由回饋訊息的傳送，傳送與接收的雙方可以取得一個平衡點，使得資料的傳送較有效率，播出也更為平順
Distributed Multimedia Information Systems

- Multiple streams of data can be supplied by different sources
- Various streams need to be coordinated both spatially and temporally for presentation
- Temporal composition of these streams
Architecture of multimedia synchronization system

- In a distributed multi-stream network environment, the server and client each has a Synchronizer and multiple Actors for synchronization and user interaction processing.
  - Actor
    - Intra-medium synchronization
    - User interaction processing
  - Synchronizer
    - Intra-medium synchronization
    - User interaction processing
The Architecture of multimedia synchronization system
參考資料

- http://www2.ndap.org.tw/eBook08/showContent.php?PK=203
- http://cs.uccs.edu/~cs525/synmm/synmm.htm
- Anna Haj Hać and Cindy X. Xue “Synchronization in multimedia data retrieval” International Journal of Network Management
Scalable Video Coding
Motivation

- Provide a universal media access
  - One encoding/multiple decoding
  - Seamless & dynamic adaptation to a diversity of networks/terminals/formats
Architecture

- One base layer, many enhancement layers
Features

• Most components of H.264/MPEG-AVC are used
  – Motion-compensated
  – Intra-prediction
  – Transform and entropy coding
  – Deblocking as well as the NAL unit packetization
  – Base layer of a SVC bit-stream is coded
    in compliance with H.264/MPEG4-AVC
Features

• Capable of reconstructing lower resolution or lower quality signal from partial bit stream
  – Adaptation to network and terminal capabilities

• Different modalities of scalability
  – Spatial, temporal, SNR, rate, computation, fine granularity

• Efficiency v.s. drift problem

• Breakthrough
  – Lifting scheme
  – Motion compensated temporal filtering (MCTF)
  – Combined with DCT-coded base layer (scalable extension to MPEG-4 AVC/H.264)
One encoding

Department of Computer Science and Information Engineering Wireless and Mobile Communications Lab.
## Multiple decoding

<table>
<thead>
<tr>
<th>High resolution</th>
<th>Medium resolution</th>
</tr>
</thead>
<tbody>
<tr>
<td>High frame rate</td>
<td>Medium frame rate</td>
</tr>
<tr>
<td>High quality</td>
<td>Medium quality</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>High resolution</th>
<th>Low resolution</th>
</tr>
</thead>
<tbody>
<tr>
<td>High frame rate</td>
<td>Medium frame rate</td>
</tr>
<tr>
<td>Low quality</td>
<td>High quality</td>
</tr>
</tbody>
</table>
Encoder structure

- 3 spatio-temporal resolutions, each containing 1 low quality layer, 1 SNR enhancement layer
What does scalability mean?

- Encode Once, Decode/Display/Stream Anywhere

- Current digital video applications require at least three types of scalability features:
  - Quality scalability
  - Spatial resolution scalability
  - Temporal (frame rate) scalability
Quality Scalability

• Coarse-grain quality scalability (CGS)
  – A special case of spatial scalability
    • Identical sizes for base and enhancement layers
  – Smaller quantization step sizes of for higher enhancement residual layers
  – Designed for only several selected bit-rate points
    • Supported bit-rate points = Number of layers
  – Switch can only occur at IDR access units
Quality Scalability

• Medium-grain quality scalability (MGS)
  – More enhancement layers are supported
    • Refinement quality layers of residual
  – Key pictures
    • Drift control
  – Switch can occur at any access units
  – CGS + key pictures + refinement quality layers
Spatial Scalability

• Layered structure
  – Over sampled pyramid decomposition
  – MC prediction structures of all layers are aligned
  – Efficient inter-layer prediction (switchable) from low to high layers
Spatial Scalability

- Based layer – low resolution and frame rate
- Spatial Prediction – macro-block, motion vector, residual
- Enhancement layer – high resolution and frame rate
Temporal Scalability

• P/B hierarchical pictures (already supported by H.264/AVC)
  – (not restricted to) dyadic temporal decomposition
  – Very good compression efficiency (better than common GOP structures)
  – Can be combined with multiple reference pictures
Temporal Scalability
Quality Scalability with Temporal Scalability

Group of Picture (GoP) = 8

Display order
0 1 2 3 4 5 6 7 8

Coding order
0 5 3 6 2 7 4 8 1
Network Abstraction Layer

- NAL Unit Header format

<table>
<thead>
<tr>
<th>Slice Type</th>
<th>TL</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>I, P</td>
<td>000</td>
<td>3</td>
</tr>
<tr>
<td>B1</td>
<td>001</td>
<td>2</td>
</tr>
<tr>
<td>B2</td>
<td>010</td>
<td></td>
</tr>
<tr>
<td>B3</td>
<td>011</td>
<td></td>
</tr>
<tr>
<td>PR</td>
<td>000</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>001</td>
<td></td>
</tr>
<tr>
<td></td>
<td>010</td>
<td></td>
</tr>
<tr>
<td></td>
<td>011</td>
<td></td>
</tr>
</tbody>
</table>
Inter-layer prediction

- Enhancement layer MB prediction can apply on
  - Motion data
    - MB partitioning & motion derived from base layer MBs
  - Intra texture
    - Up-sampled base layer texture (intra-coded MB)
    - Smoothed reference high layer texture + base layer residual
  - Inter texture(residual)
    - Up-sampled base layer residual
Single-loop decoding

- Inter-layer intra prediction is restricted to base layer MB that are coded in intra mode
- Single motion compensation loop (including deblocking) is sufficient at decoder side
- Only pictures of highest layer are stored in the decoded picture buffer
- Encoder should be operated in multiple-loop mode
- Additionally required complexity for supporting spatial scalability is believed to be smaller than for MPEG-2, H.263, MPEG-4

- Impact on Coding Efficiency
  - Minor impact (0 – 0.5 dB)
  - New inter-layer prediction concepts (motion & residual) in comparison to MPEG-2, H.263, MPEG-4
Extended Spatial Scalability

- Extended spatial scalability (ESS)
  - Cropping window with any inter-layer ratio

- Interface / Progressive scalability
  - Supports any configuration, e.g. CIF -> SDi, 1080i -> 1080p
Fidelity Scalability

• **Coarse/Medium Grain Scalability (CGS/MGS)**
  – resolution ratio is equal to 1 (no cropping)
  – no up-sampling (motion, texture) required; single-loop decoding
  – Key pictures concept allowing drift control
• **High layer encoded using lower QP value than lower layer**

![Waterfall images](image1.png)  ![Waterfall images](image2.png)  ![Waterfall images](image3.png)
Potential applications/products

• Applications/products
  – Broadband/Mobile video distribution
  – Video in Home Network
  – Professional video manipulation
• Issues
  – Heterogeneous network conditions
  – Multiple devices resolutions: mobile QVGA / VGA / QCIF / CIF – TV SD/HD
  – Adapt content to different resolutions (content repurposing)
  – Use of low resolution proxy for browsing / editing
  – Management of multiple output formats (4K/2K/1K, HD/SD-DVD, VHS, Internet)
• SVC assets
  – Smooth bandwidth adaptation
  – Content Repurposing / Devices adaptation
  – Error resilience / storage erosion
    • Simplified equipments
    • Workflow simplification
    • Content management simplification
Applications of SVC

Department of Computer Science and Information Engineering
Wireless and Mobile Communications Lab.
Summary

• SVC extension of AVC makes efficient scalability of video happen
  – Hierarchical P/B pictures allow temporal scalability and provide best drift-reduced performance
  – Other modalities of scalability achieved by layered representation of pictures
  – Typically, performance better than simulcast, worse than single layer

• SNR scalability
  – Residual coded differentially with almost standard H.264 / AVC
  – Key pictures concept allowing drift control

• Spatial scalability
  – Over-sampled pyramid and independent temporal decomposition in each spatial layer
  – Inter-layer prediction: Intra, motion, residuals
  – Only one MC loop necessary when restricting
Demo Video

• H.264 v.s. SVC
References


Flash Streaming
Outline

• Flash Streaming
• Flash Media Server
• Real-Time Messaging Protocol
• Red5
Flash Streaming

• The Adobe® Flash® Media Server family of products has become the industry-leading solution for streaming video and real-time communication.

• The ubiquity of the Adobe Flash Platform provides a rich viewing experience across virtually all operating systems and screens through integration with the Adobe Flash Player runtime, adopted on 98% of computer screens worldwide.
Flash Streaming (con’t)

- Encode
  - Flash Media Encoding Server
- Protect
  - Flash Access
- Stream
  - Flash Media Interactive Server / Flash Media Streaming Server
- Playback
  - Adobe Flash Player 10 / Adobe Flash Lite® 3 / The Adobe AIR®
Flash Streaming (con’t)

- Encode
  - FFmpeg & x264
- Protect
  - unknown
- Stream
  - Red5
- Playback
  - Adobe Flash Player 10 / Adobe Flash Lite® 3 / The Adobe AIR®

Department of Computer Science and Information Engineering
Wireless and Mobile Communications Lab.
Flash Media Server

- Flash Media Server solutions have both a server-side and a client-side architecture. The client experience is deployed as a SWF or AIR file, created in either Flash or Flex. Clients run within a web browser (Flash Player), mobile device (Flash Lite 3), or as a desktop application (Adobe AIR).
Flash Media Server (con’t)
Flash Media Server (con’t)

• Flash Media Server communicates with its clients using RTMP over TCP, which manages a two-way connection, allowing the server to send and receive video, audio, and data between client and server.

• There are five configurations of RTMP with Adobe Flash Media Server 3.5:
Flash Media Server (con’t)

- RTMP - Standard, unencrypted RTMP. The default port is 1935. If a port is not specified, the client attempts to connect to ports in the following order: 1935, 443, and then via RTMPT on port 80.

- RTMPT - RTMP “tunneled” over HTTP. The RTMP data is encapsulated as valid HTTP data.

- RTMPS - RTMP sent over an SSL. SSL enables secure TCP/IP connections. Flash Media Server natively supports both incoming and outgoing SSL connections. The default port is 443.

The Above 3 are supported in Red5.
Flash Media Server (con’t)

- RTMPE - Enhanced and encrypted version of RTMP. RTMPE is faster than SSL, and does not require certificate management as SSL does. If you specify RTMPE without explicitly specifying a port, the Flash Player scans ports, just as it does with standard RTMP.

- RTMPTE—Encrypts the communication channel, tunneling over HTTP. The default port is 80. The key benefits over SSL (RTMPS) are performance, ease of implementation, and limited impact on server capacity.

The Above 2 are NOT supported in Red5.
Real-Time Messaging Protocol – Introduction

• The Real-Time Messaging Protocol (RTMP) was designed for high-performance transmission of audio, video, and data between Adobe Flash Platform technologies, including Adobe Flash Player and Adobe AIR.

• RTMP is now available as an open specification to create products and technology that enable delivery of video, audio, and data in the open AMF, SWF, FLV, and F4V formats using a binary TCP connection or polling HTTP tunnel.
Real-Time Messaging Protocol – Introduction (con’t)

• A single connection is capable of multiplexing many net streams using different channels. Within these channels packets are split up into fixed size body chunks.

• An RTMP connection uses TCP/IP port 1935. It is also possible to tunnel RTMP over an HTTP connection using port 80.
RTMP – Handshake

• All communications are initiated by the client.
RTMP – Handshake (con’t)

- **Uninitialized**  Both the client and server are uninitialized. The client sends the protocol version in packet C0. If the server supports the version, it sends S0 and S1 in response. If not, the server responds by taking the appropriate action terminating the connection.

- **Version Sent** The client is waiting for the packet S1 and the server is waiting for the packet C1. On receiving the awaited packets, the client sends the packet C2 and the server sends the packet S2. The state then becomes Ack Sent.
RTMP – Handshake (con’t)

• **Ack Sent**  
  The client and the server wait for S2 and C2, respectively.

• **Handshake Done**  
  The client and the server exchange messages.
RTMP – Connect

Client

<table>
<thead>
<tr>
<th>Handshaking done</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command Message(connect) →</td>
</tr>
<tr>
<td>Window Acknowledgement Size ←</td>
</tr>
<tr>
<td>Set Peer Bandwidth ←</td>
</tr>
<tr>
<td>Window Acknowledgement Size →</td>
</tr>
<tr>
<td>User Control Message(StreamBegin) ←</td>
</tr>
<tr>
<td>Command Message(connect response) ←</td>
</tr>
</tbody>
</table>

Server
RTMP – Connect (con’t)

- Client sends the connect command to the server to request to connect with the server application instance.

- After receiving the connect command, the server sends the protocol message ‘Window Acknowledgement Size’ to the client. The server also connects to the application mentioned in the connect command.
RTMP – Connect (con’t)

• Then the server sends the protocol message ‘Set Peer Bandwidth’ to the client.

• The client sends the protocol message ‘Window Acknowledgement Size’ to the server after processing the protocol message ‘Set Peer Bandwidth’.
RTMP – Connect (con’t)

• The server sends an another protocol message of type User Control Message(StreamBegin) to the client.

• The server sends the result command message informing the client of the connection status (success/fail). The command specifies the transaction ID (always equal to 1 for the connect command). The message also specifies the properties, such as Flash Media Server version (string), capabilities (number) etc.
RTMP – Play a stream

Handshaking and Application connect done

Create Stream

Command Message(createStream)

Command Message(createStream response)

Command Message(play)

SetChunkSize

User Control(StreamIsRecorded)

User Control(StreamBegin)

Command Message(onStatus-play reset)

Command Message(onStatus-play start)

Audio Message

Video Message

Keep receiving audio and video stream till finishes

Department of Computer Science and Information Engineering Wireless and Mobile Communications Lab.
RTMP – Play a stream (con’t)

• The client sends the play command after receiving result of the `createStream` command as success from the server.

• On receiving the play command, the server sends a protocol message to set the chunk size.
RTMP – Play a stream (con’t)

• The server sends another protocol message (User Control) specifying the event ‘StreamIsRecorded’ and the stream ID in that message. The message carries the event type in the first 2 bytes and the stream ID in the last 4 bytes.

• The server sends another protocol message (User Control) specifying the event ‘StreamBegin’, to indicate beginning of the streaming to the client.
RTMP – Play a stream (con’t)

• The server sends OnStatus command messages NetStream.Play.Start & NetStream.Play.Reset if the play command sent by the client is successful.

• NetStream.Play.Reset is sent by the server only if the play command sent by the client has set the reset flag. If the stream to be played is not found, the Server sends the onStatus message NetStream.Play.StreamNotFound.
RTMP – Play a stream (con’t)

• After this, the server sends audio and video data, which the client plays.
RTMP – Publish a stream

Handshaking and Application connect done

Publisher Client

Create Stream

Command Message(createStream)

Command Message(createStream response)

Command Message(publish)

User Control(StreamBegin)

Data Message(Metadata)

Audio Data

SetChunkSize

Video Data

Publishing Content

Until the stream is complete

Server

Command Message(publish result)
RTMP – Publish a stream (con’t)

• The client sends the publish command after receiving result of the createStream command as success from the server.

• On receiving the publish command, the server sends protocol message (User Control) specifying the event ‘StreamBegin’, to indicate beginning of the stream publishing.
RTMP – Publish a stream (con’t)

• The client sends data message (Metadata) about the stream data (audio, video etc.) like creation time, duration, and theme to the server.

• The client sends the Audio Data to the server. After that, the client sends a protocol message to set the chunk size.
RTMP – Publish a stream (con’t)

• The server sends the result command message informing the client of the publishing status (success/fail). If the result of the publishing status is success, the client sends the Video Data to the server.

• A publisher can publish a stream and then stream the video to the server. Other clients can subscribe to this published stream and play the video.
RTMP – Shared Object

Handshaking and Application connect done

Client

Create and Connect Share Object

Shared Object Event(Use)

Shared Object Event(UseSuccess, Clear)

Shared Object Event(RequestChange)

Shared Object Event(Success)

Shared Object Event(SendMessage)

Server

Shared Object Event(SendMessage)

Shared Object Event(SendMessage)

Shared Object Set Property

Shared Object Set Property

Department of Computer Science and Information Engineering Wireless and Mobile Communications Lab.
RTMP – Shared Object (con’t)

• The example illustrates the messages that are exchanged during the creation and changing of shared object.

• It also illustrates the process of shared object message broadcasting.
Red5 – Introduction

Red5 is an open source project dedicated towards the interaction between the Flash Player and a Free Connection Oriented Server using RTMP.
Red5 – Introduction (con’t)

• It is written in Java and supports:
  – Streaming Audio/Video (FLV and MP3)
  – Recording Client Streams (FLV only)
  – Shared Objects
  – Live Stream Publishing (live h264 supported now)
  – Remoting
Red5 – Introduction (con’t)

• Red5 is released under Free Software Licenses (GNU Lesser General Public License).

• The project is currently at 0.9.0 RC1. It is started from 2005. Here is the changelog.
Red5 – Legal Stuff

• Red5 is a Open Source, Reverse Engineered Flash Media Server.
  – http://osflash.org/red5/fud
  – http://osflash.org/red5/red5_legal_reasonings
# Red5 – Supported Encoding Standards

<table>
<thead>
<tr>
<th>Video</th>
<th>Audio</th>
</tr>
</thead>
<tbody>
<tr>
<td>On2 VP6</td>
<td>MPEG-1 Audio Layer 3</td>
</tr>
<tr>
<td>Sorenson H.263</td>
<td>Speex</td>
</tr>
<tr>
<td>H.264</td>
<td>AAC</td>
</tr>
</tbody>
</table>
Red5 - Buffering
FMS Buffering Strategy

- Subscribing a pre-recorded FMS stream, it's possible to define a subscribing buffer with the method netStream.setBufferTime(N). The buffer behavior is indeed very simple and similar to that of other media streaming servers.

- The Flash Player receives the stream and archives it until the buffer is full, at that moment, the stream starts to play and the Flash Player tries to keep the buffer full to its nominal size. If the bandwidth is insufficient, the buffer slowly decreases; when the buffer is empty, the playing is stopped until the buffer reaches again the desired length.
FMS Buffering Strategy (con’t)

• At the beginning, the buffer starts to fill linearly because the playing is not yet started and the bandwidth is constant. In time T1 the buffer is full, the stream start to play and the buffer is kept full by a bandwidth greater than the required (Available bandwidth > 100% of the required by stream).
FMS Buffering Strategy (con’t)

• In T2 the available bandwidth becomes insufficient and the buffer starts to empty. In T3 the buffer is definitely empty, the stream is paused and therefore the buffer restart to fill almost linearly because of the almost constant bandwidth (notice that the refill time is more than doubled because of the halved bandwidth).
FMS Buffering Strategy (con’t)

• Reached the full state, the stream exits from pause and the buffer size is the result of the buffer flushing (due to the playing) and the buffer filling (due to the available bandwidth). In time T5 the available bandwidth reach the 100% minimum required value and the buffer becomes to fill again up to the full status (T6).
FMS Buffering Strategy (con’t)

• We have seen that with this standard, static buffering method, if bandwidth decreases under the required value, the buffer may be insufficient to compensate the bandwidth void and one or more rebuffering may be necessary to get over. Obviously with a larger buffer, this could not happen.

• To prevent rebuffering, the buffer must be large. A larger buffer can compensate a longer time lapse with insufficient bandwidth. But a large buffer means an higher buffering time and probably a worst viewer experience.
FMS Buffering Strategy (con’t)

- Using the onStatus event of our netStream object, we are able to recognize when the buffer is full or empty. So, we can set a starting buffer size and then, reached the buffer full status, we can set it to an higher value to exploit the bandwidth eventually in excess. If the buffer goes empty, we can lower the buffer size again to the starting value.
FMS Buffering Strategy (con’t)

- Compared to the previous behavior, when the Starting Buffer is full, the buffer size is enlarged to exploit the amount of bandwidth beyond the required. Until the time T2, when available bandwidth decreases under the 100%, the buffer continues to fill. With this "supply" of video, the lack of bandwidth between time T2 and T3 is handled without video interruptions. At time T3 bandwidth returns over the 100% and the buffer grows again.
FMS Buffering Strategy (con’t)

- The dynamic buffer can guarantee short starting time and at the same time an arbitrarily high resilience to bandwidth fluctuation. Dynamic buffering is more useful when average bandwidth is higher than the required (a very common scenario). When average bandwidth is equal to the required, it is still useful but starting buffer must be large.

Source Code Tracing

• **PlayEngine.java**

• The bandwidth controllable is registered in the bandwidth controller which provides the three token buckets used for bandwidth control.

```java
94 /**
95 * Service that controls bandwidth
96 */
97 private IBWControlService bwController;
```

• The bandwidth controller manages the token buckets assigned to the bandwidth controllable and distributes the tokens to the buckets in an implementation-specific way.
Source Code Tracing (con’t)

```java
private RTMPMessage pendingMessage; // 排隊等待被送出的 message
private boolean waitingForToken = false;
private boolean checkBandwidth = true;

/**
 * Interval in ms to check for buffer underruns in VOD streams.
 */
private int bufferCheckInterval = 0;

/**
 * Number of pending messages at which a <code>NetStream.Play.InsufficientBW</code>
 * message is generated for VOD streams.
 */
private int underrunTrigger = 10;

/**
 * Timestamp when buffer should be checked for underruns next.
 */
private long nextCheckBufferUnderrun;
```

為避免緩衝不足錯誤所定義的檢查時間間隔
為避免緩衝不足錯誤所定義的下一次檢查時間

排隊等待被送出的 message
 bandwidth

當所有排隊等待被送出的 message 總數超過這個值後，代表 bandwidth 不足

Timestamp when buffer should be checked for underruns next.
Source Code Tracing (con’t)

```java
public void setBandwidthController(IBWControlService bwController,
                                     IBWControlContext bwContext) {
    this.bwController = bwController;
    this.bwContext = bwContext;
}

public void setBufferCheckInterval(int bufferCheckInterval) {
    this.bufferCheckInterval = bufferCheckInterval;
}

public void setUnderrunTrigger(int underrunTrigger) {
    this.underrunTrigger = underrunTrigger;
}
```

設定上述變數的 API
Source Code Tracing (con’t)

```java
/**
 * Periodically triggered by executor to send messages to the client.
 */
private class PullAndPushRunnable implements Runnable {

    /**
     * Trigger sending of messages.
     */
    public void run() {
        try {
            pullAndPush();
        } catch (IOException e) {
            // We couldn't get more data, stop stream.
            log.error("Error while getting message", e);
            PlayEngine.this.stop();
        }
    }
}
```
protected synchronized void pullAndPush() throws IOException {
    if (playlistSubscriberStream == State.PLAYING && pullMode && !waitingForToken) {
        if (pendingMessage != null) {
            RTMPEvent body = pendingMessage.getBody();
            if (okayToSendMessage(body)) {
                return;
            }
            sendMessage(pendingMessage);
            releasePendingMessage();
        } else {
            while (true) {
                IMessage msg = msgIn.pullMessage();
                if (msg == null) {
                    // No more packets to send
                    stop();
                    break;
                } else if (msg instanceof RTMPPacket) {
                    RTMPPacket rtmpMessage = (RTMPPacket) msg;
                    RTMPEvent body = rtmpMessage.getBody();
                    if (!receiveAudio && body instanceof AudioData) {
                        // The user doesn't want to get audio packets
                        ((IStreamData) body).getData().free();
                        if (sendBlankAudio) {
                            // Send reset audio packet
                            sendBlankAudio = false;
                        }
                    }
                }
            }
        }
    }
}
Source Code Tracing (con’t)

```java
/**
 * Check if it's okay to send the client more data. This takes the configured
 * bandwidth as well as the requested client buffer into account.
 * @param message the message
 * @return true if okay to send, false otherwise
 */
private boolean okToSendMessage(RTHFEEvent message) {
    if (!(message instanceof IStreamData)) {
        String itemName = "Undefined";
        // if current item exists get the name to help debug this issue
        if (currentItem != null) {
            itemName = currentItem.getName();
        }
        Object[] errorItems = new Object[] {message.getClassName(), message.getDataType(), itemName};
        throw new RuntimeException(String.format("Expected IStreamData but got %s (type %s) for %s", errorItems));
    }
    return okToSendMessage;
}
```
Source Code Tracing (con’t)

```
final long now = System.currentTimeMillis();

// check client buffer length when we've already sent some messages
if (lastMessage != null) {
    // Duration the stream is playing / playback duration
    final long delta = now - playbackStart;
    // Buffer size as requested by the client
    final long buffer = playlistSubscriberStream.getClientBufferSize();
    // Expected amount of data present in client buffer
    final long buffered = lastMessage.getTimestamp() - delta;

    log.trace("okayToSendMessage: timestamp {} delta {} buffered {} buffer {}", new Object[]{lastMessage.getTimestamp(), delta, buffered, buffer});

    // Fix for SN-122, this sends double the size of the client buffer
    if (buffer > 0 && buffered > (buffer * 2)) {
        // Client is likely to have enough data in the buffer
        return false;
    }
}
```

Start time of stream playback. It's not a time when the stream is being played but the time when the stream should be played if it's played from the very beginning.

Eg. A stream is played at timestamp 5s on 1:00:05. The playbackStart is 1:00:00.
所有等待被送出的 message 總數

```java
long pending = pendingMessages();
if (bufferCheckInterval > 0 && now >= nextCheckBufferUnderrun) {
    if (pending > underrunTrigger) {
        // Client is playing behind speed, notify him
        sendInsufficientBandwidthStatus(currentItem);
    }
    nextCheckBufferUnderrun = now + bufferCheckInterval;
}

if (pending > underrunTrigger) {
    // Too many messages already queued on the connection
    return false;
}
```

當系統時間大於或等於為避免緩衝不足錯誤所定義的下一次檢查時間時會進行檢查，而當所有等待被送出的 message 總數超過 underrunTrigger 後，執行 sendInsufficientBandwidthStatus()

下一次檢查時間 = 現在時間 + 檢查時間間隔
Source Code Tracing (con't)

1183   * Insufficient bandwidth notification
1184   * @param item Playlist item
1185   */

private void sendInsufficientBandwidthStatus(IPlayItem item) {
    Status insufficientBW = new Status(StatusCodes.NS_PLAY_INSUFFICIENT_BW);
    insufficientBW.setClientid(streamId);
    insufficientBW.setLevel(Status.WARNING);
    insufficientBW.setDetails(item.getName);
    insufficientBW.setDescription("Data is playing behind the normal speed.");

    doPushMessage(insufficientBW);
}

送出 bandwidth 不足的 message
Source Code Tracing (con’t)

```java
804  IoBuffer ioBuffer = ((StreamData) message).getData();
805  if (ioBuffer != null) {
806      final int size = ioBuffer.limit();
807      if (message instanceof VideoData) {
808          if (checkEbandwidth
809              && !videoBucket.acquireTokenNonblocking(size, this)) {
810              waitingForToken = true;
811              return false;
812          }
813          } else if (message instanceof AudioData) {
814              if (checkEbandwidth
815                  && !audioBucket.acquireTokenNonblocking(size, this)) {
816                  waitingForToken = true;
817                  return false;
818              }
819          }
820  }
821  return true;
822  }
```

Basically token bucket is used to control the bandwidth used by a stream. There’s a background thread that distributes tokens to the buckets in the system according to the configuration of the bucket. The configuration includes how fast the tokens are distributed.

When a stream needs to send out a packet, the packet's byte count is calculated and each byte corresponds to a token in the bucket. The stream is assigned a bucket and the tokens in the bucket are acquired before the packet can be sent out. So if the speed (or bandwidth) in configuration is low, the stream can't send out packets fast.
public double getEstimatedBufferFill() {
    final IRtmpEvent msg = engine.getLastMessage();
    if (msg == null) {
        // Nothing has been sent yet
        return 0.0;  
    }
    // Buffer size as requested by the client
    final long buffer = getClientBufferDuration();
    if (buffer == 0) {
        return 100.0;
    }
    // Duration the stream is playing
    final long delta = System.currentTimeMillis() - engine.getPlaybackStart();
    // Expected amount of data present in client buffer
    final long buffered = msg.getTimestamp() - delta;
    return (buffered * 100.0) / buffer;
}