

# Investigations Into Efficient Temporal Video Scaling

Michael Kropfberger  
mailto:michael.kropfberger@gmx.net

Department of Information Technology, Klagenfurt

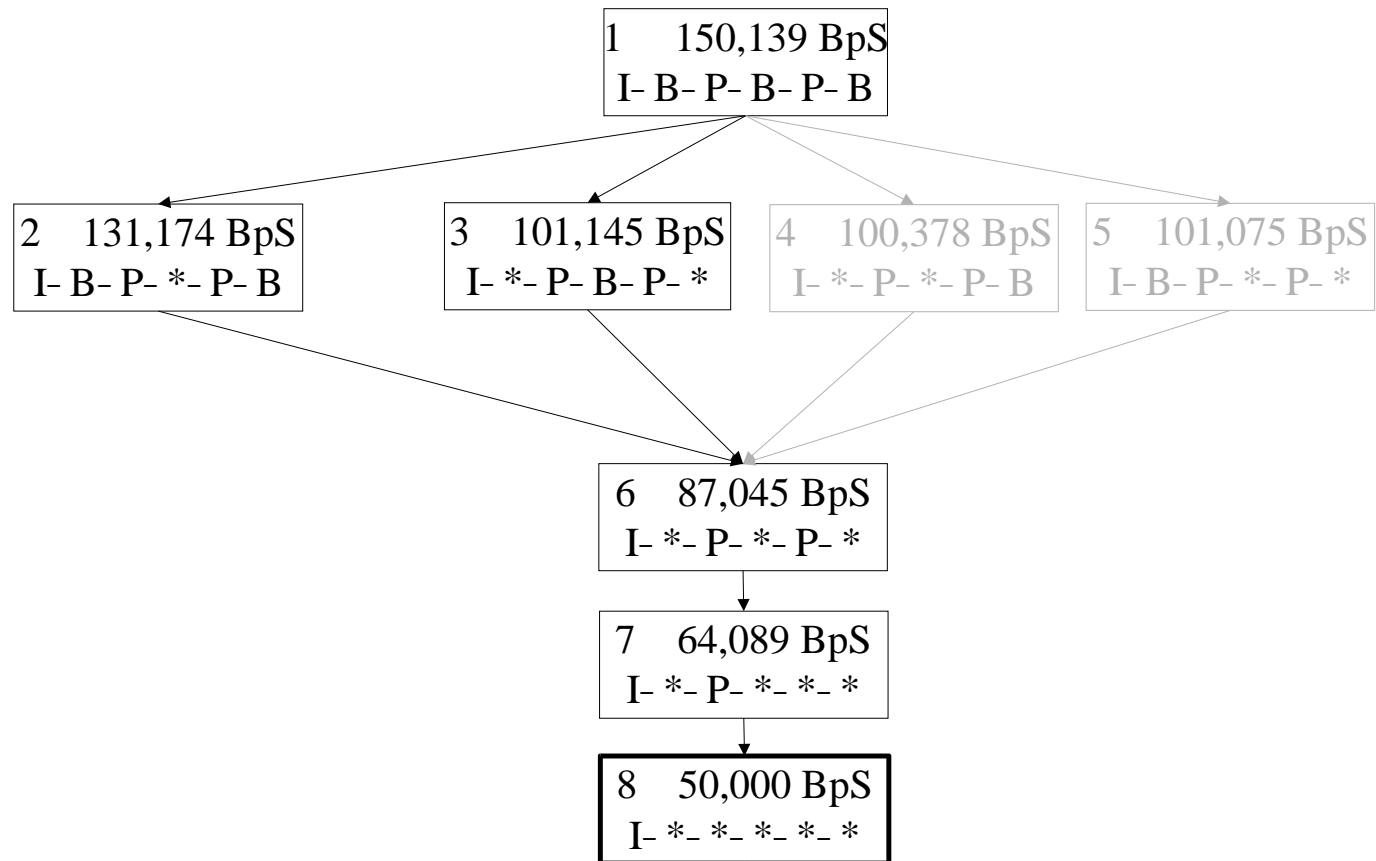
# Adaption Methods

- Non-Realtime Scalability
  - reducing the resolution
  - grayscale
  - changing the colordepth
- Realtime Scalability
  - Temporal Scalability
  - Spatial Scalability
  - SNR Scaling (Signal to Noise Ratio)
  - FGS (Fine Grained Scalability)

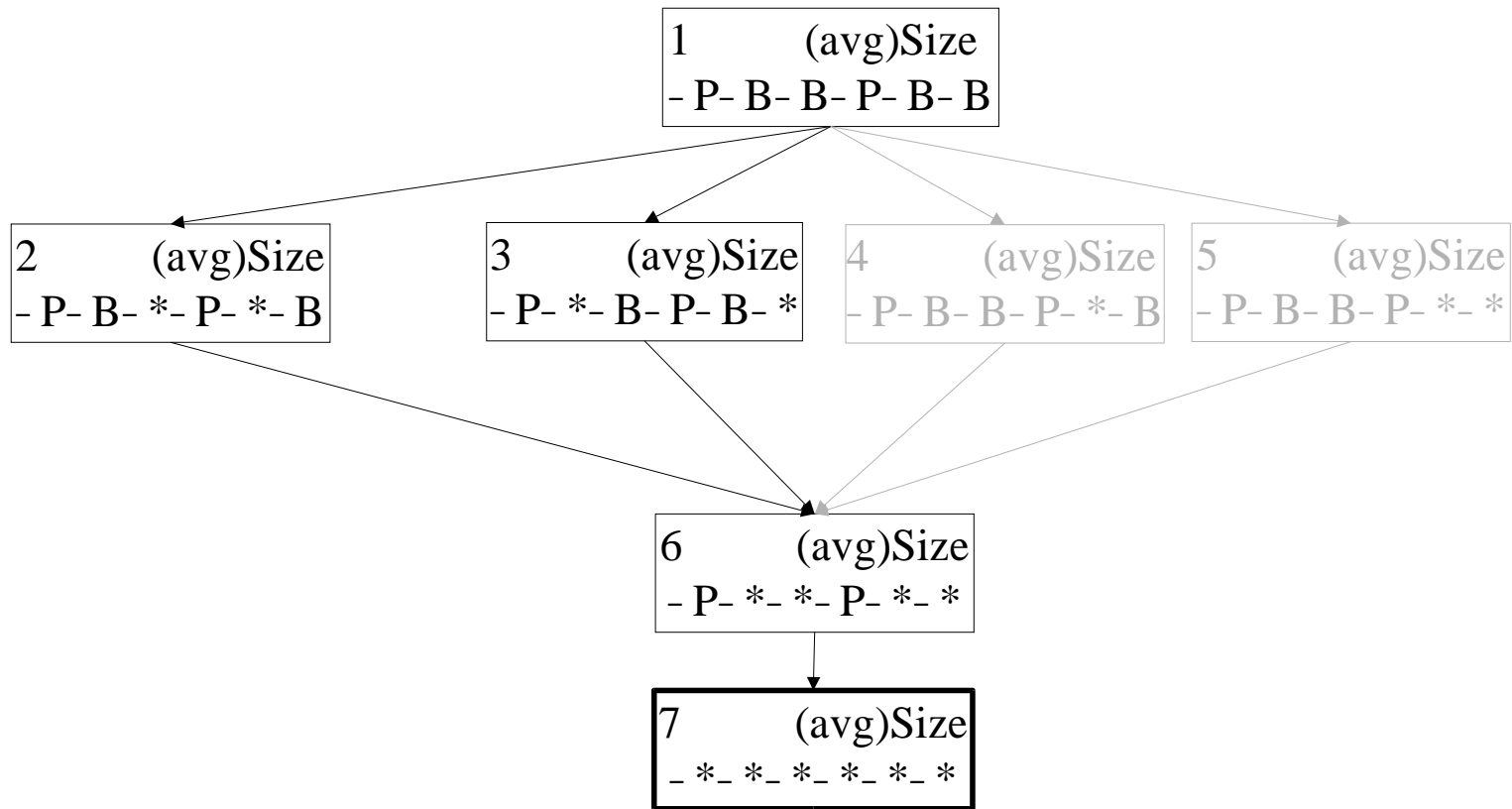
## Pattern Modification Generation

- MPEG-4 Video Elementary Stream: Patterns like IBPBPB
- Temporal Scaling means Dropping Frames
  - B-Frames can be dropped at will
  - P-Frames
    - Problem: IPBBPBBPBB(P)BBPBB I  $\rightarrow$  IPBBPBBP(BBPBBPBB) I
  - I-Frames
    - lose all following P and B-Frames until next I-Frame
- *base layer* (I-B-P-B-P-B-), *enhancement layer* (-P-B-B-P-B-B)

# Tree of Possible Base Layer Pattern Modifications



# Tree of Possible Enhancement Layer Pattern Modifications



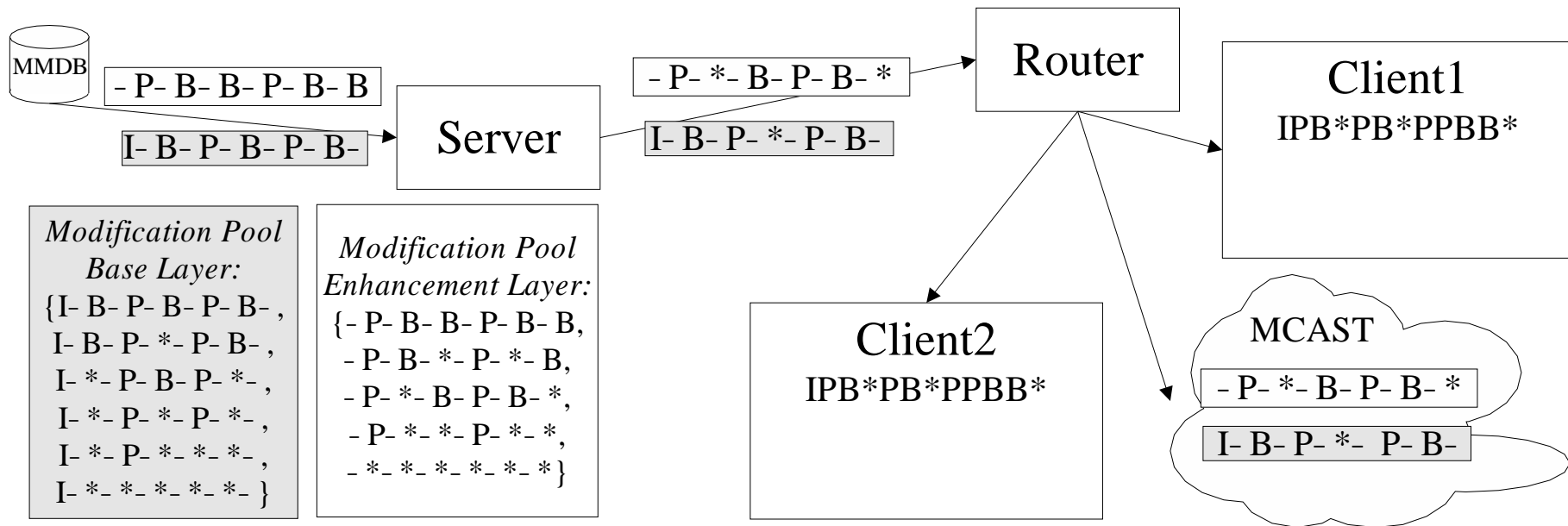
## Heuristics for Pattern Modifications

- Importance ( $I > P > B$ )
- Timely balanced distribution
  - Pattern I-\*P-\*P- better than I-B-P-\*-\*-
- averaged Signal to Noise Ratio (SNR) for modified pattern
- Tree size vs. fine grained scalability

## Tree Chopping

- Tree chopping with respect to heuristics like timely balanced distribution
- delete similarly sized nodes based on threshold values possible decision rules are:
  - preferring higher frame rates
  - preferring patterns with higher-quality frames  
(I-\*-P-\*-P- better than I-B-\*-B-\*-)
  - preferring better SNR averages

# Realtime Stream Adaptation



Two clients requesting same quality

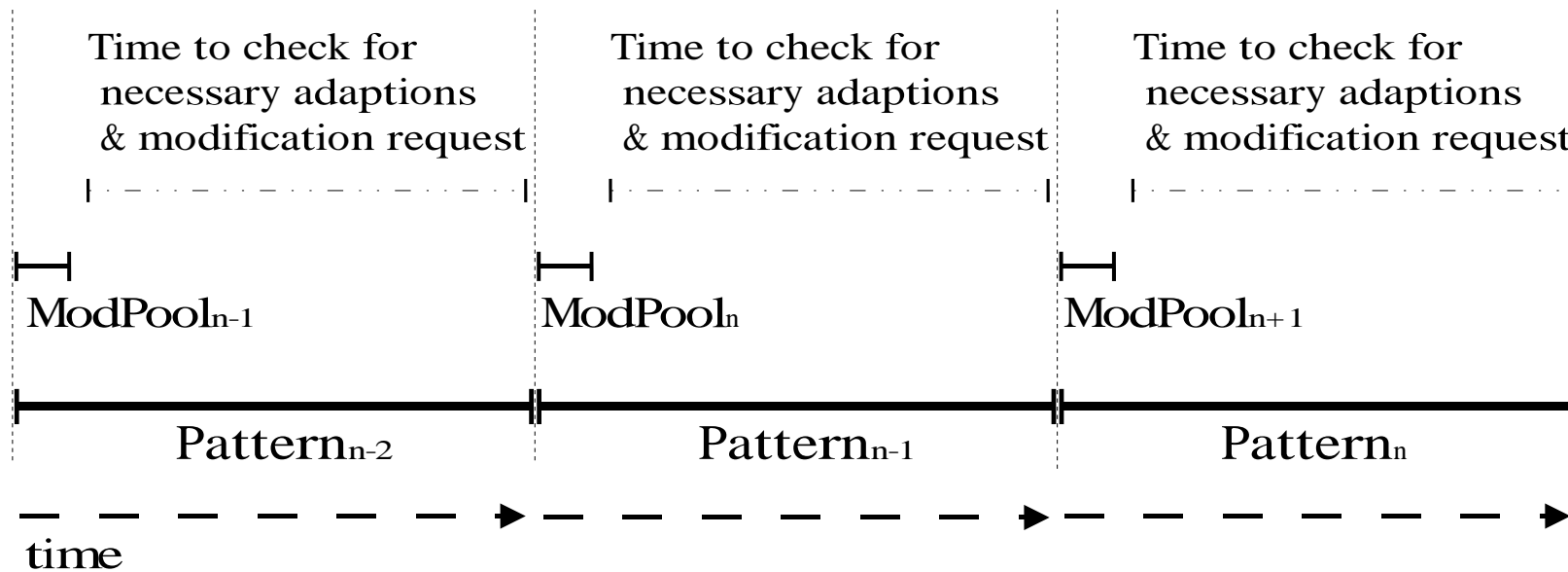
base and enhancement layer arrive via multicast



# Communication Protocol in an Adaptive Environment

1. A new *Client* sends initial specification of video name, resolution, environment and more to the *Router*
2. The *Router* requests the initial video information like resolution, color-depth, average bandwidth and average bandwidth reached by maximum adaptivity
3. *RouterSrv* sends first *modification pool* (in any efficient representation format) with bandwidth requirements respective to the different modifications
4. The *Router* checks available bandwidth to *Client(s)*

5. *Router* requests necessary maximum frame pattern to serve all connected *Clients*
6. *RouterSrv* sends out the requested frame pattern and the next *modification pool*



Modification Pool  $MP_n$  is sent with Pattern  $P_{n-1}$ , so there is enough

time to react to changing bandwidth requirements

7. *Router* sends out the shared base and enhancement layer frames via multicast
8. *Router* generates the specific base and enhancement layers and sends them to the connected *Clients* via unicast
9. if no new clients arrive, go back to Step 4

## BSDL (XML) Example for Video Adaptation: Pre-Sent Info

```
<Init>  
  <Resolution>  
    <X>352</X>  
    <Y>288</Y>  
  </Resolution>  
  <FrameRate>15</FrameRate>  
  <ColorDepth>12</ColorDepth>  
  
  <BaseLayer>  
    <RTPid>514</RTPid>  
    <BpS>150000</BpS>
```

```

<Modifications>                                <!-- Possible modifications
  <Choice>
    <Feasibility>RealTime</Feasibility>
    <ModID>0</ModID>                            <!-- Modification ID -->
    <ScaleType>Temporal</ScaleType>           <!-- Type of modification -->
  </Choice>
  <Choice>
    <Feasibility>NonRealTime</Feasibility>
    <ModID>1</ModID>                            <!-- Modification ID -->
    <ScaleType>GrayScale</ScaleType>         <!-- Type of modification -->
    <BpS>67000</BpS>
  </Choice>
</Modifications>
</BaseLayer>

```

```

<EnhanceLayer>
  <RTPid>516</RTPid>
  <BpS>200000</BpS>

  <Modifications>                                <!-- Possible modifications -->
    <Choice>
      <Feasibility>RealTime</Feasibility>
      <ModID>0</ModID>                            <!-- Modification ID -->
      <ScaleType>Temporal</ScaleType>           <!-- Type of modification -->
    </Choice>
    <Choice>
      <Feasibility>NonRealTime</Feasibility>
      <ModID>1</ModID>                            <!-- Modification ID -->
      <ScaleType>GrayScale</ScaleType>         <!-- Type of modification -->
      <BpS>113000</BpS>
    </Choice>

```

```
    </Modifications>  
  </EnhanceLayer>  
</Init>
```

## BSDL (XML) Example for Video Adaptation: periodical Info

```
<Pattern>
  <BaseLayer>
    <RTPid>514</RTPid>
    <NumFrames>30</NumFrames> <!-- no of frames in the following patten
    <RTPseqStart>3456</RTPseqStart>
    <Modifications>
      <Feasibility>RealTime</Feasibility>
      <Choice>      <!-- No Modification -->
        <ModID>0</ModID>
        <ScaleType>None</ScaleType>
        <BpS>150000</BpS>
```

```
    <TotalSize>300000</TotalSize>
</Choice>
<Choice>      <!-- drop every 8th frame -->
  <ModID>1</ModID>
  <ScaleType>Temporal</ScaleType>
  <BpS>100000</BpS>
  <TotalSize>200000</TotalSize>
  <Drop>
    <Frame>8</Frame>
    <Frame>18</Frame>
    <Frame>28</Frame>
  </Drop>
</Choice>
<Choice>      <!-- drop every 5th frame, temporally distributed -->
  <ModID>2</ModID>
  <ScaleType>Temporal</ScaleType>
```

```
<BpS>80000</BpS>
<TotalSize>160000</TotalSize>
<Drop>
  <Frame>3</Frame>
  <Frame>8</Frame>
  <Frame>13</Frame>
  <Frame>18</Frame>
  <Frame>23</Frame>
  <Frame>28</Frame>
</Drop>
</Choice>
</Modifications>
</BaseLayer>

<EnhanceLayer>
  <RTPid>516</RTPid>
```

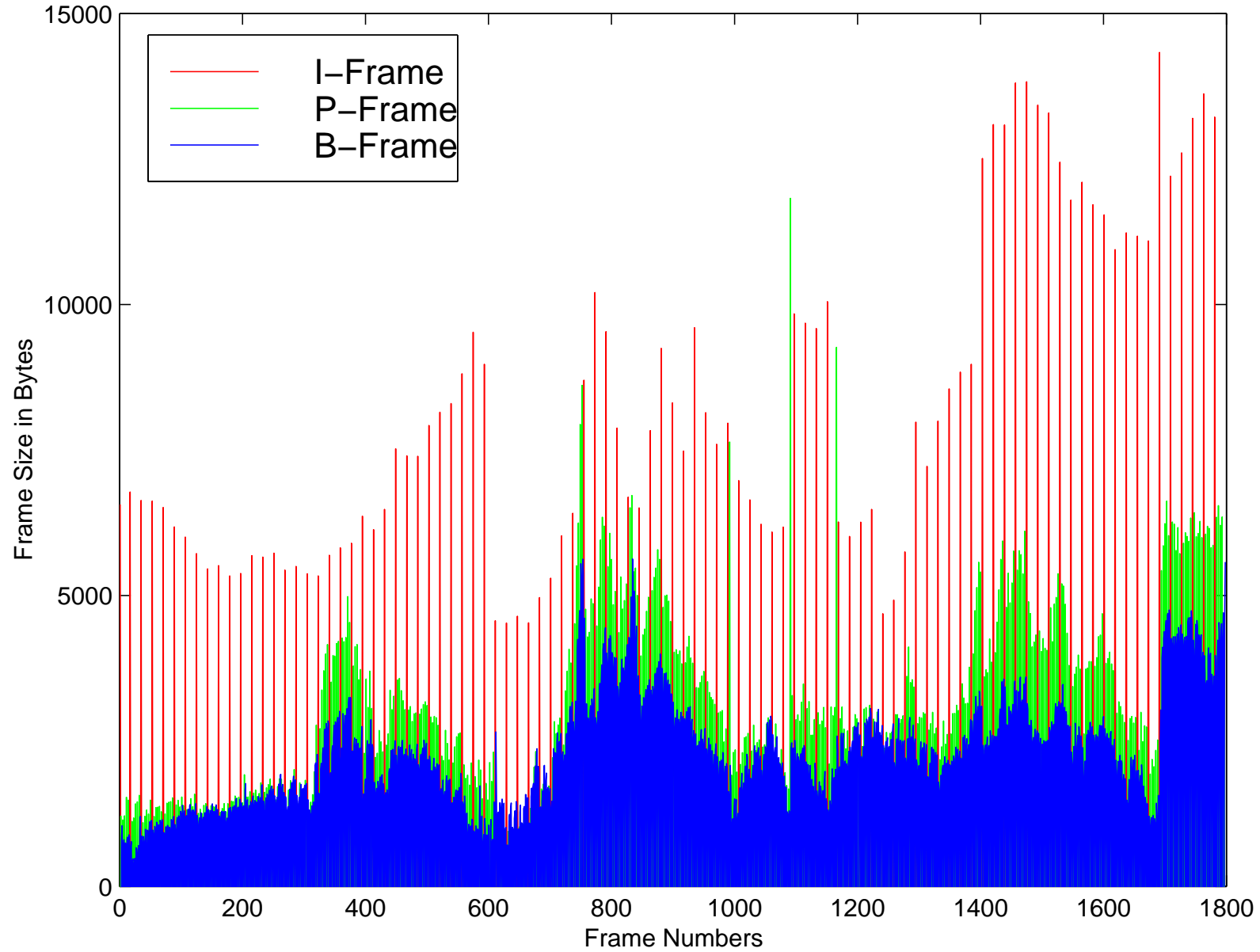
```
<Modifications>  
  ...  
  ...  
  ...  
</Modifications>  
</EnhanceLayer>  
</Pattern>
```



## Ideas on Optimization

- needed preconditions
  - a video uses the same pattern repeatedly over the whole lifetime (or at least for a couple of iterations),
  - the average needed bandwidth per pattern is very similar,
  - and the average framesizes between and within the patterns are similar,
- IDEA: then merge multiple patterns into a *pattern group*
  - problem: too high variations already on a frame-to-frame basis
  - solution: fixed bitrate encoding
    - \* not yet fully supported by any ISO encoder
    - \* drawbacks: massive quality loss or bandwidth overhead

# Frame Sizes



## Available Software

Total input bytes = 5196951

Number of VOPs = 1798

Frame	I-Frame(avg deviation)	P-Frame(avg deviation)	B-Frame(avg deviation)
0	6567		
1		1211	
2			780
3			*** HIGH DEVIATION *** 1052( 1052  136 15%)
4		1142( 1176  -34  3%)	
5			*** HIGH DEVIATION *** 678( 678  -187 22%)
6			801( 739  62  8%)

7		1215( 1195  20  2%)		
8			751( 745  6  1%)	
9			737( 741  -4  1%)	
10		*** HIGH DEVIATION ***		
		1535( 1535  170 12%)		
11			807( 774  33  4%)	
12			849( 811  38  5%)	
13		1485( 1510  -25  2%)		
14			887( 849  38  4%)	
15			863( 856  7  1%)	
16	6779( 6673  106  2%)			
17			*** HIGH DEVIATION ***	
			687( 687  -84 11%)	
18			769( 728  41  6%)	
19		*** HIGH DEVIATION ***		
		1120( 1120  -195 15%)		
20			*** HIGH DEVIATION ***	
			430( 430  -149 26%)	

21			475( 452  23  5%)
22	1180( 1150  30  3%)		
23			404( 428  -24  6%)
24			483( 455  28  6%)
25	*** HIGH DEVIATION ***		
	1419( 1419  135 11%)		
26			*** HIGH DEVIATION ***
			668( 668  107 19%)
27			645( 656  -11  2%)
28	1455( 1437  18  1%)		
29			574( 615  -41  7%)
30			704( 659  45  7%)
31	1566( 1501  65  4%)		
32			*** HIGH DEVIATION ***
			994( 994  168 20%)
33			1205( 1099  106 10%)
.			
.			

.					
.					
.					
1780	13221(13282	-61	0%)		
1781				3969(	3781  188  5%)
1782				3663(	3722  -59  2%)
1783		6346(	6134  212  3%)		
1784				3836(	3779  57  2%)
1785				4234(	4006  228  6%)
1786		6543(	6338  205  3%)		
1787				4534(	4270  264  6%)
1788				4280(	4275  5  0%)
1789		6207(	6272  -65  1%)		
1790				4214(	4244  -30  1%)
1791				4517(	4380  137  3%)
1792		6348(	6310  38  1%)		
1793				4259(	4319  -60  1%)
1794				4701(	4510  191  4%)

```

1795          *** HIGH DEVIATION ***
              4621( 4621| -844|15%)
1796          4520( 4515|    5| 0%)
1797          *** HIGH DEVIATION ***
              5575( 5575|  530|11%)

```

	TNo	TSize	TAvg
I-Frames	100	815961	8159
P-Frames	500	1611609	3223
B-Frames	1198	2769348	2311

```

  1(16frm): tsize:  21360 ( 0%) avg 1335: IPBBPBBPBBPBBPBB
##### ***END OF PATTERN GROUP (1 pats)*** avgTsize 21360 avgFrmSize  1335

```

```

  2(18frm): tsize:  21557 ( 0%) avg 1197: IBBPBBPBBPBBPBBPBB
  3(18frm): tsize:  24178 (-12%) avg 1343: IBBPBBPBBPBBPBBPBB
  4(18frm): tsize:  25941 (-20%) avg 1441: IBBPBBPBBPBBPBBPBB
  5(18frm): tsize:  26183 (-21%) avg 1454: IBBPBBPBBPBBPBBPBB
  6(18frm): tsize:  27913 (-29%) avg 1550: IBBPBBPBBPBBPBBPBB

```

##### \*\*END OF PATTERN GROUP (5 pats)\*\* avgTsize 25154 avgFrmSize 1397

7(18frm):	tsize:	28759 ( 0%)	avg 1597:	IBBPBBPBBPBBPBBPBB
8(18frm):	tsize:	26972 ( 6%)	avg 1498:	IBBPBBPBBPBBPBBPBB
9(18frm):	tsize:	27984 ( 3%)	avg 1554:	IBBPBBPBBPBBPBBPBB
10(18frm):	tsize:	27179 ( 5%)	avg 1509:	IBBPBBPBBPBBPBBPBB
11(18frm):	tsize:	29253 (-2%)	avg 1625:	IBBPBBPBBPBBPBBPBB
12(18frm):	tsize:	30943 (-8%)	avg 1719:	IBBPBBPBBPBBPBBPBB
13(18frm):	tsize:	32198 (-12%)	avg 1788:	IBBPBBPBBPBBPBBPBB
14(18frm):	tsize:	33079 (-15%)	avg 1837:	IBBPBBPBBPBBPBBPBB
15(18frm):	tsize:	34221 (-19%)	avg 1901:	IBBPBBPBBPBBPBBPBB
16(18frm):	tsize:	33450 (-16%)	avg 1858:	IBBPBBPBBPBBPBBPBB
17(18frm):	tsize:	33496 (-16%)	avg 1860:	IBBPBBPBBPBBPBBPBB
18(18frm):	tsize:	34054 (-18%)	avg 1891:	IBBPBBPBBPBBPBBPBB

##### \*\*END OF PATTERN GROUP (12 pats)\*\* avgTsize 30965 avgFrmSize 1720

.  
. .  
.