Network monitoring for Video Quality over IP

Amy R. Reibman, Subhabrata Sen, and Jacobus Van der Merwe
AT&T Labs – Research
Florham Park, NJ

Abstract—**In this paper, we consider the problem of predicting application-level video quality based on monitoring video packets within the network. Our goal is to detect impairments (either server- or network-induced) that will affect the quality of the video displayed to the viewer. We present a framework for in-network video quality degradation monitoring, and describe the information that can be extracted by processing the observed packets at different depths. For the popular Windows Media streaming format, we present techniques to estimate video quality degradations based on examining information at the IP, TCP and application-layers. We demonstrate the potential of our methods using IP traffic observed at a broadband head-end over a one-week period.**

I. Introduction

As more and more homes and offices obtain broadband access to the Internet, more people can view video over the Internet. This trend brings increased availability of more applications that use the Internet for video transmission: distance education, e-commerce, corporate presentations, sports, entertainment, and news. The wider audience brings greater potential revenue – for content providers, hosting centers, and network service providers.

However, the best-effort service model and shared infrastructure of traditional IP networks, coupled with the lack of end-to-end quality-of-service mechanisms, means network impairments will continue to hamper the viewing experience. Viewers of streaming video over the Internet are familiar with the different types of impairments that large and variable delays, jitter, congestion, and loss may introduce. First, the client buffer may starve, resulting in either video that appears jerky or long rebuffering intervals. Second, the system may choose to intentionally reduce the bit-rate transmitted from server to client, resulting in either reduced quality video or a slide-show appearance. Third, video packets sent by the server may never be received by the client, causing artifacts or frozen frames. In addition to the network, the server itself may induce impairments, for example, switch to a lower quality video stream in order to handle more connections under heavy loads. Whether server or network induced, such impairments may have a profound impact on the overall video quality perceived by the user.

Thus, there is increasing interest in monitoring and managing networks and applications that send video over IP, to ensure all end users obtain acceptable video quality. An on-line performance monitoring capability can create a real-time understanding of a particular application, which can aid in monitoring compliance of service level agreements (SLAs) between Internet Service Providers (ISPs), hosting centers, and content providers, alert operators to potential performance problems, and help in root-cause analysis and debugging.

It is not enough to monitor average network performance, because the Internet is complex, heterogeneous and dynamic – different portions of an end-to-end path may traverse different ISPs, have different bandwidth capacities, and undergo different time-varying loads. Furthermore, monitoring using only network-level measurements may be insufficient, since substantially different application-level performance can be caused by seemingly identical network impairments. For example, a single packet loss may be highly visible or completely invisible, depending on its location in the video bitstream [1].

The choice of *where* application-level monitoring should take place depends on a number of competing requirements. Measurements taken at the end-user (for example, by the end-user application) may seem to be the most accurate; however, for reasons of trust, privacy, feasibility, accessibility, and scalability of administration and management, monitoring at the end-user may be inappropriate. For example, a typical ISP or large corporate network can be very large and complex, composed of several hundred routers and even more end-points. In such a situation, in-network monitoring can be the only feasible practicable solution. In-network monitoring is also essential for corporate networks managed by a third-party, where the network operator does not have any access to the end-systems. Further, if an ISP is interested in monitoring the service provided within its network for a video stream traversing multiple ISPs, in-network monitoring is

the only solution. Several obvious choices of location are to monitor between the Local Area Network (LAN) and the Wide Area Network (WAN), or at access (or peering) points where a video stream enters/leaves a particular ISP.

In this paper, we consider the problem of predicting application-level video quality by monitoring video packets within the network. Our goal is to obtain accurate, real-time, end-to-end estimates of per-stream quality degradation for all videos observed from our in-network monitoring location. One particular focus is to be able to identify when the encoded bitstream has been modified (by either server or network) enough to impair the displayed quality. We denote all such cases as "bitstream impairments".

In section II, we describe factors that affect the quality of video over IP. In section III, we describe information that can be extracted from the video packets by examining them at different depths. In section IV, we describe algorithms for analyzing video quality based on information obtained at one particular depth. Section V gives some examples of how application-level quality can be assessed using packets monitored at a cable head-end.

II. Received Video Quality for IP networks

We are interested in performance characterization that accurately reflects the application-level end-user quality experience. How the human end-user perceives the received video quality depends on a number of factors:

- The bits received by the client. These depend on the actual video content (i.e., the amount of motion and texture), the encoding parameters (i.e., overall bit-rate, compression algorithm, spatial and temporal resolution), and any subsequent modifications of the bitstream by either the server (intentional rate changes) or the network (large delays, jitter, reduced throughput, or packet losses).
- The decisions made by the client in response to the received bits, including loss-concealment and client-buffer strategies. Even in response to a bitstream received with no network impairments, the operating system may impact temporal smoothness.
- The viewer, including viewer eyesight, expectations, and the choice of one of a number of possible display resolutions (including full-screen mode).
- The environmental viewing conditions, including the viewing distance and background lighting.

Clearly, a network-based monitor will be unable to observe any factors in the last two categories. Further, many factors in the second category cannot be known exactly. For instance, in the case of a large jitter, clients typically have a proprietary response that may be unknown to a network monitor: the actual buffer delay used to prevent against starvation is unknown and may change with time due to adaptive playback techniques [2]. Finally, because the observations are taken inside the network and not at the end-client, the exact bits received by the client as well as their exact arrival time may not be known precisely.

However, as we will see, network-based monitoring can still provide significant understanding of the video displayed to the viewer. For instance, we can detect when the server adjusts its rate and what mechanism it uses to do so, and we can detect when the packets arrive such that the display times intended by the encoder can not be satisfied. These are all important factors that impact the video quality displayed at the client.

III. Processing depths for monitoring video in the network

We assume we have recorded packets in the network; for each packet, we have the entire content and the time at which it was observed. We now consider the different depths at which we can examine these packets to extract meaningful measures of the video quality at the client.

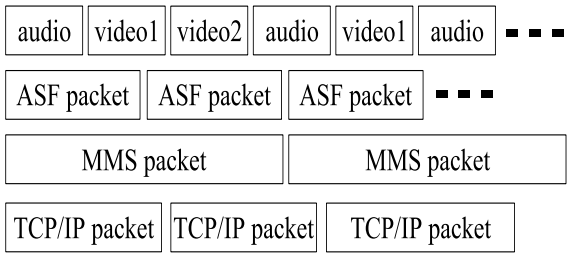

Fig. 1. Audio and video packetization for Windows Media

The different depths correspond to different levels in the protocol stack. Figure 1 illustrates a typical encapsulation of the different protocol layers in a packetized video stream. This example, using Windows Media, is just one example of a protocol stack for supporting video over IP. Another commonly used approach is to send MPEG-4 video using RTP, with UDP over IP. While the details of any video quality monitor would necessarily be different, the basic principles remain.

The innermost (top) layer in Figure 1 consists of an interleaved sequence of compressed digital video frames and associated audio segments. This information is encapsulated in the Advanced Systems Format (ASF) [3] protocol layer, into fixed-size ASF packets that also contain ASF-specific information. The next level of encapsulation involves adding Microsoft Media Streaming (MMS) [4] protocol-specific information to the ASF data stream to

create MMS packets. One or more ASF packets are typically encapsulated in the payload of a single MMS data packet. The outermost two layers correspond to the Internet transport and network protocols. The MMS stream is packetized into TCP packets, with added TCP header information. This is followed by the outermost (or lowest) IP-level packetization.

From the outermost layer to the innermost, the deeper we can look into the contents of a packet, the more precise information we can obtain about the state of the video stream and the more accurately we can predict the resultant video quality. We discuss next the information associated with video quality that can be extracted from the different levels in the protocol stack.

Estimating video quality using either IP-level information or TCP-level information is very difficult. At the IP level, we can extract only information about packet arrival times (at the monitor) and packet length. At the TCP level, we can additionally identify duplicate data transmissions and out-of-order packets. An algorithm designed to estimate video quality or to detect bitstream impairments using only information obtained from the IP- and TCP-levels might use the average transmitted rate over a time interval, the number or rate of duplicate or out-of-order packets, or look for temporal gaps in the transmission.

More accurate information regarding bitstream impairments can be obtained by examining each packet up to and including the application-specific protocols (MMS and ASF in Figure 1). Such an examination requires an understanding of the streaming protocols, but does not examine the media content itself. Thus, we do not need specific knowledge of the audio or video compression algorithms to examine at this depth.

The MMS protocol includes both command and data packets [4]. MMS command packets contain controls signals between server and client, including start, stop, and requests and acknowledgments for rate changes.

ASF packets contain sequence header information or fixed-size data packets [3] in which variable-length media objects are multiplexed. By examining packets at the ASF level, we can obtain summary information about audio or video frames, including bytes per frame and the intended times to send and present (or display) each frame. We describe in detail in section IV how ASF-level monitoring provides a wealth of information for identifying bitstream impairments. In particular, server rate changes can be detected by observing media object ID information, and client buffer behavior can be estimated by using the presentation times included at the ASF level for each frame along with each packet's monitoring time.

Deeper examination requires delving into the video bitstream itself. There are different degrees to which we can examine the video bitstream. We can completely parse the bitstream without decoding to obtain video pixels, using a technique called FullParse (FP) [5]. With FP, we can obtain information regarding macroblock motion, quantization parameters, and the DCT coefficient values. Alternatively, we can also completely decode the video to obtain YUV pixels for every stream, using a full decoding. Once we have the decoded pixels, we can apply existing No-Reference Pixel-based (NR-P) methods to estimate the visual quality. Typical NR-P methods measure blocking [6] or blurring [7] impairments in a single frame. The full decoding requires an inverse Discrete Cosine Transformation (IDCT) to obtain the video pixels.

Clearly, the deeper we examine, the more information can be obtained to estimate quality. However, deeper examination requires more complex processing, which may not be appropriate for in-network monitoring. For example, since NR-P methods requires a video decoder for each stream being monitored, it is likely to be prohibitive inside the network, especially as link speeds continue to increase leading to potentially more concurrent streams on a link. Further, both a full decoding and FP require knowledge of the video syntax; therefore, they may not be applicable for proprietary algorithms.

To obtain the most information regarding video quality, we should gather information at *every* stage of the processing. For example, while the NR-P quality metrics have access to the individual pixels, they typically do not use the motion or quantization information that is available to FP. Further, they would not have information regarding the *intended* time to display a frame, only information regarding the *monitored* time a frame would be displayed; therefore, they may miss key information if not part of a comprehensive, hierarchical quality monitoring system.

IV. Video Quality Analysis

In this section, we describe a video monitoring tool that makes use of information extracted from ASF packets observed in the network. We begin by describing the information that can be extracted from the ASF packets, and then show how these values can be used to identify server rate adaptations and identify sub-optimal client-buffer behavior for each stream.

A. ASF packet observations

Each ASF transmission, or file, starts with a header describing file information (including the file GUID, overall file duration, and overall file bit-rate), the number of available audio and video streams within the file, and summary

information for each audio (sample rate, etc.) and video stream (spatial resolution, compression algorithm) [3].

We can also obtain the following information regarding the media objects (or frames) multiplexed within each ASF data packet: $i, j, b_{ij}, k_{ij}, s_{ij}, p_{ij}, m_{ij}, a_{ij}$. Media object j from stream number i has a total of b_{ij} bytes. If $k_{ij} = 1$, then this media object is a key-frame (or I-frame). The remaining four variables are timing information associated with this media object. The ASF packet contains the target send time of the packet, which is the time the server is supposed to transmit this ASF packet. We denote s_{ij} as the send time of the last ASF packet which contains information about media object j from stream i. The target presentation time, p_{ij}, for media object j in stream i is the time intended by the encoder for the client to present, or display this media object. The time at which the monitor observes the last received ASF packet containing information about this media object is m_{ij}. The arrival time at which the client itself receives all information about this media object, a_{ij}, may not be exactly observable, depending on the location of the monitor within the network. Here, we assume there is no extra jitter on the path between the monitoring point and the client.

B. Detecting server rate adaptation

Using the above information, it is a simple matter to detect when the server has intentionally reduced the transmitted bit-rate, either in response to a client-initiated rate-change request or because of server overload.

Two types of rate adaptation are typically seen. In the first, the ASF file contains multiple video streams. When the server receives a request to reduce the transmitted rate, it switches from one video stream to a lower-rate video stream at the next key-frame. In this case, the new rate is nearly constant with time. In the second type of rate adaptation, the server reduces the rate by suppressing the transmission of non-key frames. At the client, the audio still plays, but the video display becomes a series of still frames, often several seconds apart. Now, because key-frames can appear with arbitrary frequency in the video bitstream, the transmitted rate is bursty. In cases of severe network congestion, the server can also stop sending any video frames at all. In this case, the last received video frame is displayed while audio is played.

It is a simple matter to observe the stream IDs of media objects to determine when rate adaptation via stream-switching has occurred. To detect the case where non-key frames have been discarded, we look for gaps in the sequence of j, the media object ID for video stream i. Gaps caused by a lost packet are typically followed by a non-key frame, while gaps caused by intentional server rate adaptation are followed by a key-frame. The visual impact of both are similar: a freeze-frame.

Gaps in the audio streams can be found the same way, indicating audio glitches at the client.

C. Understanding client-buffer behavior

To analyze the client-buffer behavior for a particular stream, it is necessary to know the sequence of both the arrival times and presentation times for each frame. Once this information is obtained by parsing the ASF packets, it is straightforward to emulate any specific client-buffer strategy (see for example those in [2]) to evaluate its performance. However, inside the network, we can never know the specific strategy used by the client to manage its buffer. Therefore, we do not focus here on specific buffer strategies.

Instead, we focus on detecting when the arrival times, in relationship to the intended presentation times, force the client to show frames at times other than those intended by the encoder. Specifically, we assume the difference between two consecutive presentation times included by the encoder in the ASF packets indicate the best possible amount of time to display each frame. If the client must display a frame for a different amount of time, the temporal rendition of the video will deteriorate.

In this subsection we consider a single video stream only, so we drop the subscript for the stream, j. Thus, a_i is the time when the i-th frame has completely arrived at the client. We assume $a_1 = 0$ and $p_1 = 0$. We define the current delay in the client buffer to be Δ_c.

The client will be forced to alter the display durations if any frame i arrives such that $a_i > p_i + \Delta_c$. Thus, if the client starts displaying frames with an initial delay Δ_c, and if $\Delta_c < \max_i(a_i - p_i)$, then there will be a moment in time when the client can no longer preserve the intended display durations. The last time at which this is no longer possible is defined to be the onset of a buffer-induced display violation for delay Δ_c. The only way to ensure the client never has a buffer-induced display violation is to set the initial delay equal to $\Delta_{max} = \max_i(a_i - p_i)$. Unfortunately, Δ_{max} cannot be known until the entire stream has been received. In this paper, we characterize the sequence of the arrival times for each frame, relative to the presentation times, by Δ_{max}.

V. Examples

We have developed a tool to investigate the quality of streams at a network monitoring point. In this section we present several examples of our tool applied to video data

monitored over a one-week period at the cable head-end of a network service provider. Over the observation interval, we collected data for 30399 Windows Media streams that used TCP transport; 10951 of these contained video.

First, we explore one particular stream using different processing depths, to indicate (i) the types of rate reduction that take place, and (ii) the improved understanding we gain by processing deeper into the payload. The stream we consider contains two types of rate changes: switching between alternate streams with different bit-rates, and temporarily suppressing transmission of non-key frames.

Figure 2 shows the effective rate of the sample stream using two different monitoring depths: TCP and ASF. First, the curve labeled "arrival time" shows the aggregate stream rate using the arrival time of packets at the monitoring point; this rate is calculated using the total number of bytes in the payload of each packet and the arrival time of those packets. The averaging interval is 5 seconds. This rate varies dramatically over the observation period, and there are clearly some times during which this rate is reduced. However, it is impossible to tell what type of rate changes took place. For the second curve, labeled "presentation time", we parse deeper into the payload and extract the presentation time from the ASF packets to calculate the rate. In this case, there are periods where the average rate is nearly constant, which clearly suggests that at different times, streams encoded at different rates are sent.

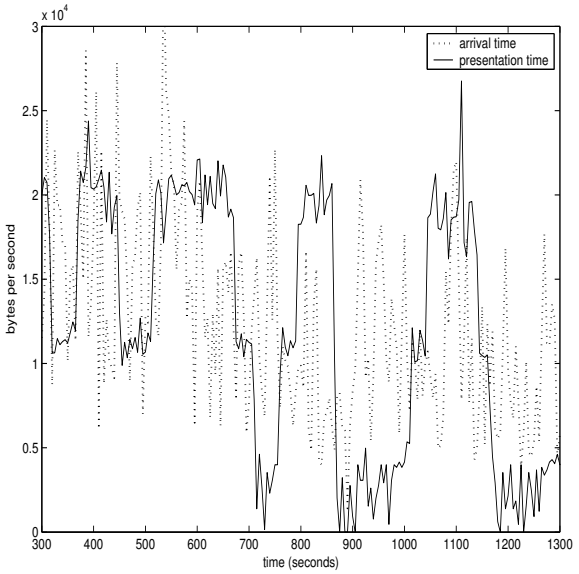

Fig. 2. Effective rate of sample stream based on arrival time and presentation time

Figure 3 confirms this observation, by depicting the detailed contents of the ASF packets. This stream contained three different media objects during the course of the transmission: an audio stream and two video streams at different rates. The figure clearly shows the transition times when the transmission switches from one video stream to another. As can be expected, the rate of the audio stream is fairly constant, but at three times (between 700 and 800 seconds, around 900 seconds and around 1200 seconds) the rate of the audio stream drops, suggesting the audio is also impaired. Note also that during these three time periods, the video rate drops to zero with very occasional short bursts of a few hundred bytes per second. A closer examination of the packets at the ASF-level indicates that during these times, the video in these intervals only contains key-frames.

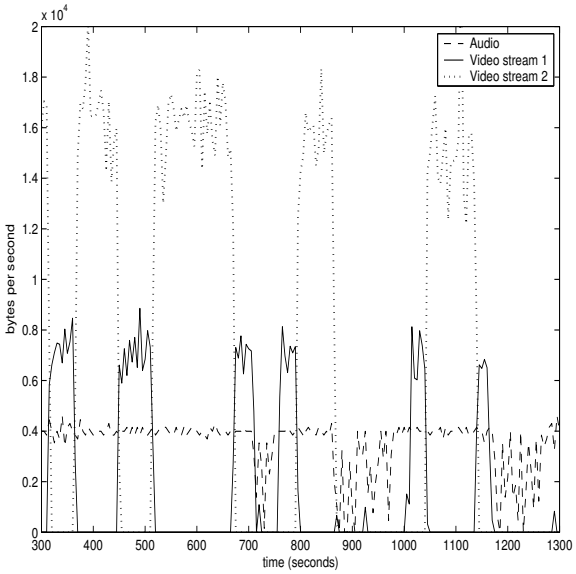

Fig. 3. Effective rate of media components in sample stream

Next, we examine the client buffer behavior for all video streams in our data set. In Figure 4 we show the CDF of Δ_{max}, the minimum time that the client could have waited to fill its initial buffer without being forced to alter the presentation times. While Δ_{max} cannot be known a-priori since it depends on the network conditions, it is instructive in our case to examine it after the fact to understand typical Internet streams. From Figure 4, 50% of the streams had $\Delta_{max} \leq 1.2$ seconds, and 90% had $\Delta_{max} \leq 4.1$ seconds. Focusing on the tail of the distribution, 5% of streams required $\Delta_{max} \geq 8.4$ seconds and 1% required Δ_{max} greater than 45 seconds.

A different way of examining the arrival process relative to the presentation time is to consider the amount of time it takes the media player to receive a certain amount of content. We examine the cumulative distribution functions of the time to receive X seconds worth of video for $X = 5, 10, 15$ and 30 seconds. For small X, this gives

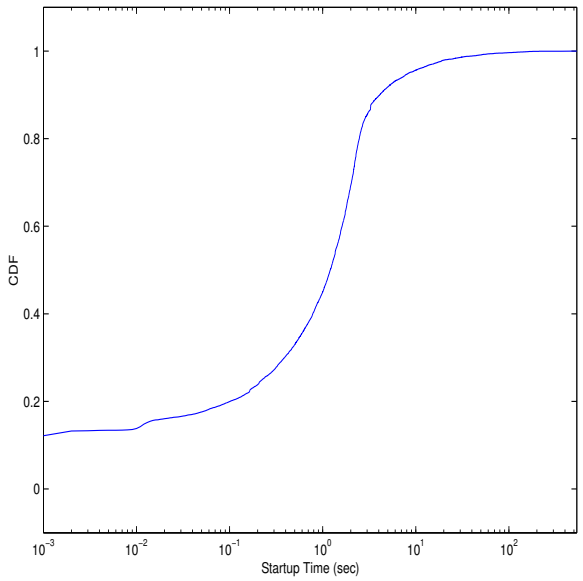

Fig. 4. CDF of minimum playout delay, Δ_{max}.

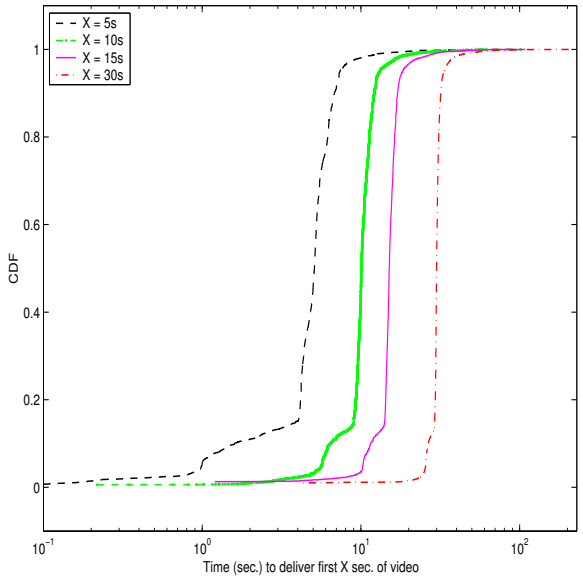

Fig. 5. CDF of the time it takes to receive X seconds worth of content

an indication of the initial transmitted bit-rate relative to the content bit-rate, which lets us examine the case where many servers deliver content at a faster rate during the initial startup period. For larger X, this gives an indication of the network's sustained ability to deliver the content.

Figure 5 shows the results of this analysis, where we consider only those streams with at least X seconds of content. As can be expected these distributions have a very steep incline around the time in question; it typically takes about X seconds to stream X seconds worth of content. All four distributions level of fairly quickly. A small percentage of streams take more than 1.5 times X to receive X seconds of video. For example, less than 4.7% of the streams took more than 7.5 seconds to stream 5 seconds worth of data. Corresponding numbers (i.e., more than 1.5 times X) for the other plots are 3.2% for 10 seconds, 2.1% for 15 seconds and 1% for 30 seconds respectively.

In all four cases roughly half the streams received X seconds worth of content in X seconds or less. However, as is most visible in the 5-second plot, some streams receive the indicated amount of content in *less* than X seconds. We believe this is due to a higher-rate initial startup period, indicating that about 15% of streams benefit from this approach.

VI. Conclusions

We present a framework for monitoring video quality within the network. We presented some initial results that show the advantages of this approach. We can identify when the server changes the transmission rate, and understand the algorithm it uses to do so. Also, we can detect when packets arrive such that the display times intended by the encoder cannot be satisfied.

In our ongoing work, we use the framework and tools for a more extensive study of the quality of video streamed over the Internet.

Acknowledgements

We thank Oliver Spatscheck for collecting the streaming data using the Gigascope infrastructure [8]. We also thank John Apostolopoulos for his helpful comments.

References

[1] A. Reibman, S. Kanumuri, V. Vaishampayan, and P. Cosman, "Visibility of packet losses in MPEG-2 video," in *IEEE International Conference on Image Processing*, October 2004.

[2] M. Kalman, E. Steinbach, and B. Girod, "Adaptive media playout for low-delay video streaming over error-prone channels," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 14, pp. 841–851, June 2004.

[3] Microsoft Corporation, "Advanced systems format (ASF) specification, revision 1.20.02." ASF_Specification.doc available from: `http://download.microsoft.com`, June 2004.

[4] Streaming Download Project, "MMS streaming protocol." Available from: `http://sdp.ppona.com/`.

[5] A. R. Reibman, V. Vaishampayan, and Y. Sermadevi, "Quality monitoring of video over a packet network," *IEEE Transactions on Multimedia*, vol. 6, pp. 327–334, April 2004.

[6] S. Liu and A. C. Bovik, "Efficient DCT-domain blind measurement and reduction of blocking artifacts," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 12, pp. 1139–1149, December 2002.

[7] X. Marichal, W.-Y. Ma, and H.-J. Zhang, "Blur determination in the compressed domain using DCT information," in *IEEE International Conference on Image Processing*, September 1999.

[8] C. Cranor, T. Johnson, O. Spatscheck, and V. Shkapenyuk, "Gigascope: A stream database for network applications," in *Proc ACM SIGMOD*, June 2003.