December 06, 2017

Control of jitter buffer size using machine learning

Ivo Creusen

Oliver Walter

Henrik Lundin

Control of jitter buffer size using machine learning

ABSTRACT

Determining the size of a jitter buffer for calls conducted over the network involves a trade-off between the probability of buffer under-runs and user-perceived delay, which impedes communication. Techniques of this disclosure perform data-driven optimization of the jitter buffer of audio and/or video calls conducted over the internet. Event logs that contain data about network conditions experienced during completed calls are used to calculate optimal jitter buffer size using an optimizer. The output of the optimizer is used as a target for offline training of a machine-learning model to calculate optimal jitter buffer size. The trained model is employed to control jitter buffer size during a call. Control of jitter buffer size using the techniques can reduce buffer under-runs without increasing average delay. Alternatively, depending on the weights in the optimization criterion, the techniques of this disclosure can reduce average delay with the same chance of buffer underrun.

KEYWORDS

- jitter buffer
- real-time transport
- buffer under-run
- VoIP

BACKGROUND

Communication equipment or software that enables audio and/or video calls over the internet uses a jitter buffer to buffer incoming packets. The jitter buffer ensures that there is always enough media available for playback, e.g., to send to the sound card, even when there is jitter in network delay, e.g., under conditions of changing network delay.

Too small a jitter-buffer size results in buffer under-run, e.g., no media available to send to the sound card. Too large a jitter-buffer size results in an increase in user-perceived delay, impeding communication. Thus, having an optimally-sized jitter buffer throughout a call is important to call quality. For example, when an ongoing call over Wi-Fi is switched to cellular data, e.g., LTE, there is a significant change in network conditions. Such significant change warrants a change in jitter-buffer size. In general, a poor network (with high jitter) requires a larger jitter buffer while a good network (low jitter) requires a smaller jitter buffer.

DESCRIPTION

This disclosure presents techniques to predict an optimal size of the jitter buffer based on network conditions, and to adjust the size of the jitter buffer dynamically during a call using such prediction. The optimal size of the buffer is predicted using a trained machine learning model. The size of the jitter buffer can be changed, for example, by adjusting speed of playback to a slightly faster or slower speed, which has the effect respectively of decreasing or increasing the buffer size.

Training data to train the machine-learning models can be obtained from actual VoIP calls, e.g., audio and/or video calls over a packet-switched network such as the internet, upon permission from the user for use of log data. Such calls generate logs that are descriptive of traffic or network conditions during the call. The logs do not include any user data or identifiable information. For example, the logs include real time communications event logs (RTC event logs), and comprise, e.g., timing information of incoming real time transport protocol (RTP) packets, RTP headers, timing information of the sound card requesting media, etc. Packet jitter, which is the difference in timing between packet arrival and request by sound card, is computed for each packet in a call using the event log.

Per techniques of this disclosure, RTC event logs of a completed call are provided as input to an optimizer. The optimizer produces as output an optimal jitter-buffer size for each point in time during the call. In determining the buffer size at any point in a call, the optimizer has access to past, present, and future jitter information from the event logs. The optimizer calculates the buffer size based on an objective cost function based on at least three simultaneous criteria, e.g.,

- reduction in jitter-buffer size by penalizing growth in jitter-buffer size;
- avoidance of buffer under-run by imposing a penalty upon the event of a buffer under-run;
- reduction in the number of changes to jitter-buffer size during a call by penalizing changes in speed of playback; etc.

Programmatically, the optimizer constructs a two-dimensional graph with time steps on the X-axis and buffer levels on the Y-axis. For each possible buffer level at a point in time, there are transitions possible depending on actions taken, e.g., play fast, play slow, play normal, etc., and penalties associated with each transition. By finding the shortest, e.g., lowest penalty, path from the left side to the right side of the graph, penalties, as indicated by objective function, are minimized. In this manner, the optimizer generates optimal buffer size value at different time points for the completed call by use of available events logs of future events in the completed call by optimizing an objective function over the entire call. When the optimizer is applied to a set of log files corresponding to multiple completed calls, it calculates an optimal buffer size for multiple time points throughout the calls.

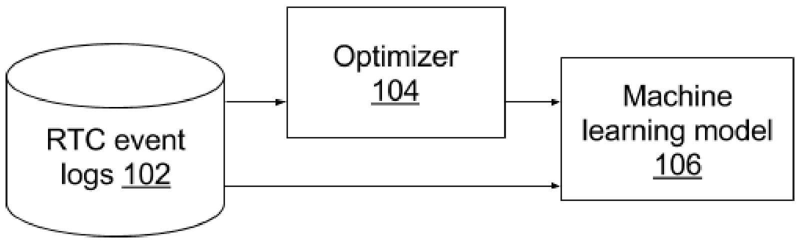

Fig. 1: Training the machine learning model

Fig. 1 illustrates the training of a machine learning model used to predict jitter-buffer sizes. RTC event logs (102) provided to an optimizer (104) which determines optimal jitter-buffer size throughout the call. The jitter-buffer size is provided as a target for machine learning model (106), along with the RTC event logs. The input features for the model include jitter data, e.g., inter-arrival times between incoming RTP packets from the RTC event log. The machine learned model is trained, with the target as the jitter-buffer size calculated by the optimizer. By providing event logs and optimal buffer size from a large number of completed calls, the machine-learning model is trained to predict the optimal buffer size.

In implementation, the trained model can produce less buffer underruns compared to other techniques to manage the buffer size, with a similar average buffer level. Further, the approach is completely data-driven since the model is trained on actual data of network conditions commonly experienced during calls. The less buffer underruns are achieved without an increase in the average delay.

The machine-learning model can be implemented using a long short-term memory (LSTM) neural network. Other types of models, e.g., recurrent neural networks, convolutional neural networks, support vector machines, random forests, boosted decision trees, etc., can also be used. Further, reinforcement learning can be used to train a neural network, e.g., without the use of the optimizer. In this case, only the objective function is used to provide feedback to the machine-learning model during the training.

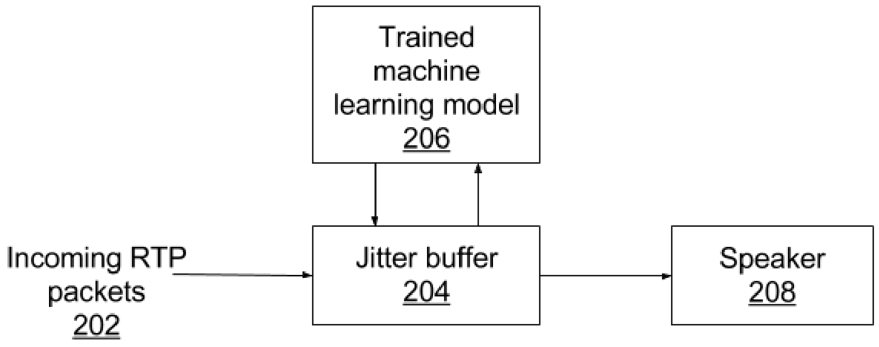

Fig. 2: Using a trained machine learning model to adjust jitter-buffer size in real time

Fig. 2 illustrates the use of a machine learning model, trained per techniques of this disclosure, to set jitter-buffer size. Incoming RTP packets (202) enter jitter buffer (204). Trained machine learning model (206) adjusts jitter-buffer size based on, for example, real-time information about the timing of incoming packets and outgoing data, e.g., to a sound card, for playback. The techniques of this disclosure can be used for calls conducted over the internet, e.g., in audio and/or video conferencing or calling software.

CONCLUSION

Techniques of this disclosure perform data-driven optimization of the jitter buffer of audio and/or video calls conducted over the internet. Event logs that contain data about network conditions experienced during completed calls are used to calculate optimal jitter buffer size using an optimizer. The output of the optimizer is used as a target for offline training of a machine-learning model to calculate optimal jitter buffer size. The trained model is employed to control jitter buffer size during a call. Control of jitter buffer size using the techniques can reduce buffer under-runs without increasing average delay. Alternatively, depending on the weights in the optimization criterion, the techniques of this disclosure can reduce average delay with the same chance of buffer underrun.