Design Doc: Dynamic Screen Capture in Chromium

This Document is Public

Authors: chfremer@chromium.org

One-page overview

Summary

WebRTC is in the process of adding support for accepting partial updates to video frames to be encoded, see Dynamic Screen Capture in WebRTC. This design doc is for changes to be implemented in Chromium in order to be able to feed WebRTC with such partial update video frames.

Demo video

Platforms

All

Teams Involved

WebRTC-MTV team (point of contact: chfremer@chromium.org)
WebRTC-STO team (point of contact: ilnik@webrtc.org)

Bug

Not yet created

Code affected

Video capture stack all the way from capturers to consumers

Design Considerations

The goal is to make screen sharing more efficient by taking into account what parts, if any, of the video content are actually changing. This is assuming that this knowledge is available at the capturer. Possible gains in efficiency can be due to the following:

- Video encoding in WebRTC has to do less work when knowing that either nothing or only a sub-region of a video frame is different from the previous frame.
- The transport of small (possibly even empty) update regions from capturer to encoder through the video capture stack may be cheaper than the transport of full video frames.
- The capturer may have less work to do when only having to capture sub-regions that have changed.

There are two separate aspects to this work that can be handled in sequence:

1. Wiring the information about what sub-region has changed through from the capturer to the encoder while still sending full frames.
2. Only capture and send the updated sub-region.

In Chromium there are capturers for capturing content from the whole screen, individual application windows, as well as tabs of the Chromium browser itself. At first we are going to consider only the capturer for “tab capture”, since in there the information about what has changed is known to be readily available. We may consider the other capturers in a future step.

Here is a diagram with links to code illustrating the path of video frames from the tab capturer to the WebRTC video encoder (as well as other consumers).

Wiring info about updated sub-region from capturer to encoder

On the way from capturer to encoder, the information about the frame is converted several times between different representations. In order to add information about an updated sub-region, we have to include it in each of these representations and update the code where the conversions are done.

Location in video capture stack	How is meta info of video frame represented
viz::FrameSinkVideoCapturerImpl::MaybeCaptureFrame()	media::VideoFrame
viz::FrameSinkVideoCapturerImpl::MaybeDeliverFrame()	media::mojom::VideoFrameInfo

content::VideoCaptureImpl::OnBufferReady()	media::VideoFrame
WebRtcVideoTrackSource::OnFrameCaptured()	webrtc::VideoFrame

Additionally, care has to be taken when the frame may be rotated, scaled, cropped, or letterboxed on its way to the encoder. Such operations may require adjusting the information about the updated region.
Q: Can we guarantee that this does not happen for tab capture?
TODO: Identify a list of code sites on the path of video frames where such operations can happen.
O: Integration testing of this will be very difficult, possible impractically difficult, because for most of the code sites where modification of the frame may happen conditionally, we do not have the ability to explicitly trigger and control these conditions in integration tests.

Care also has to be taken, when frames are dropped on the way to the encoder. If a frame is dropped, the update_rect of the subsequent frame has to be merged with the update_rect of the dropped frame.
TODO: Identify a list of code sites on the path of video frames where frame drops can happen.
Possible strategies for handling frame drops:

ID	**Strategy Description**	**Comment**
S1	Accumulating |update_rect| when frame is dropped. Such code would have to be added to each code site that may drop frames.	There are a lot of such code sites. We might be able to determine that not all of them can be reached for tab capture, but updating only a subset of them would be confusing and likely cause unexpected bugs in the future.
S2	Add code for setting |update_rect| to the full frame in the first frame after frame drops have occured. This also has to be done at each code site that may drop frames.	Simplifies the logic compared to S1. Paid for by having a potentially larger than needed |update_rect| for the first frame after frames drops.
S3	Pass a frame counter with each frame. At consumption sites, use the frame counter to determine if any frame has been dropped since the last received frame. If any frame has been dropped, ignore |update_rect| and treat the frame as full update.	Shifts the responsibility of tracking frame drops from the frame drop sites to the consumption sites interested in |update_rect|. This makes sense if the number of such consumption sites is smaller than the number of frame drop sites. This currently seems to be the case.

		A disadvantage of this strategy is that it may not be clear to every reader that |update_rect| cannot be trusted without checking the frame counter for frame drops first.

Integration testing of S1 or S2 would be very difficult, possible impractically difficult, because for most of the code sites where dropping of the frame may happen conditionally, we do not have the ability to explicitly trigger and control these conditions in integration tests.

Capture and send only the updated sub-region

The main logic for the tab capturer is located in class viz::FrameSinkVideoCapturerImpl. It appears that if no content has changed and the previous frame has already finished being consumed, it directly resends the buffer for the previous frame to the consumer without doing any capturing or copying.

If there is any change to the content, it reads a full frame from a buffer internal to the graphics framework-specific to a pool-managed shared memory buffer for transport over IPC. For example this might be a readback from an OpenGL texture. For the transport of this buffer to consumers, the video capture stack does not perform any additional conversions or copies of the data.
O: This is assuming no scaling or cropping happens. If it does, it would be near the encoder.

With this, the only savings we could expect to get for only sending the updated sub-region down the stack would come from:

- Reduced size of the capture operation into the internal buffer.
- Reduced size of the copy/conversion/readback from the internal buffer to the shared memory buffer.

Whether or not such savings can be expected to be significant enough to be worth the effort depends on the implementation details.
Q: What savings would we expect?
A: Here are some measurements.
Q: How much effort would it be to make the implementation only capture and copy-convert the updated region?

If changing a capturer to only send the updated sub-region instead of full frames, care has to be taken in case that other consumers might be connected that do not support sub-region-only frames. Implementing this might require having some component in the stack that knows whether or not all consumers support it and communicates to the capturer whether or not to enable producing sub-region-only frames.

If we only send the updated sub-region down the stack, we can run into issues when frames are modified or dropped along the way. For example, if cropping is applied, the cropping operation will have to work differently for full frames vs. partial update frames. If partial update frames are dropped, subsequent partial updates are generally no longer valid. To handle this, we would have to send back a message from the frame drop site all the way to the capturer to request a full frame and then drop all partial update frames until a full frame is received. This logic would have to be added to every code site where frame drops can occur.

Integration testing of this would be very difficult, possible impractically difficult, because for most of the code sites where modification of the frame data or dropping of the frame may happen conditionally, we do not have the ability to explicitly trigger and control these conditions in integration tests.

Other potential optimizations

FrameSinkVideoCaptureDevice currently shares and retires a new buffer handle for each frame. This requires that for each frame:

- A shared memory buffer handle is duplicated by the capturer and sent over IPC.
- The consumer maps the shared memory.
- The consumer unmaps the shared memory after having finished consuming.

These operations could be saved by keeping a set of buffers shared and only sending OnNewBufferReady() events.

Measurements

The following measurements were performed on chfremer@'s Linux workstation (HP Z840). The measurements were taken using the Chromium tracing tool with the code instrumented using this CL for Chromium + this diff for third_party/webrtc to add trace events.

The exercise performed during tracing was to move the mouse on top of a focused Chromium tab that was being captured via the Desktop Capture Example extension that comes with the Chromium sources. This would trigger the capturer to capture a full frame and that frame arrived at rtc::VideoStreamEncoder::OnFrame() in its full resolution. To compare the effect of different size rectangles being captured and sent from capturer to encoder, three runs were performed with the Chromium window having different size each time.

```
size 1920 x 1102
157 frames
```

```
CopyFromCapturerToIpcBuffer:     Average CPU Duration = 1.251ms
TransportFromCapturerToEncoder: Average Wall Duration = 1.542ms
  Map shared memory:             Average CPU Duration = 0.045ms
  Unmap shared memory:           Average CPU Duration = 0.013ms

size 388 x 238
187 frames
CopyFromCapturerToIpcBuffer:     Average CPU Duration = 0.094ms
TransportFromCapturerToEncoder: Average Wall Duration = 1.379ms
  Map shared memory:             Average CPU Duration = 0.036ms
  Unmap shared memory:           Average CPU Duration = 0.013ms

size 388 x 52
298 frames
CopyFromCapturerToIpcBuffer:     Average CPU Duration = 0.041ms
TransportFromCapturerToEncoder: Average Wall Duration = 1.382ms
  Map shared memory:             Average CPU Duration = 0.036ms
 Unmap shared memory:        Average CPU Duration = 0.012ms
```

Design Decisions

Based on the design considerations and measurements above, the added complexity and effort of capturing and sending only the updated sub-region does not seem to be worth it. **=> Go with wiring the information about what sub-region has changed through from the capturer to the encoder while still sending full frames.**

Code sites along the stack that may want to modify the frame content, e.g. rotation, cropping, letter-boxing, or scaling, will have to be dealt with by updating the sub-region information.

=> For handling frame drops use strategy S3, i.e. introduce a frame counter.

Implementation Plan

CL Title/Link	CL Description	Status
[Video Capture] Add optional update_rect to video frames and	Changes to structs used for video frame transport: * In media::VideoFrame: Add optional fields |frame_counter| and |update_rect|. * In video_capture::mojom::VideoFrameInfo: Add the	Landed in M74

fill it for tab capture	equivalent fields. Interface changes: * In viz::mojom::FrameSinkVideoConsumer::OnFrameCaptured(), use the update_rect| field that is now part of video_capture::mojom::VideoFrameInfo instead of a separate parameter. * In class viz::FrameSinkVideoCapturerImpl, pass in a media::VideoCaptureOracle into the constructor instead of hard composition in order to allow instrumenting it in tests. Functional changes: * In class viz::FrameSinkVideoCapturerImpl: Instead of always using the full frame dimensions as |update_rect|, pass the the rect that has actually changed since the last frame to the consumer. This information is already available for each captured frame in member |dirty_rect_|. However, since captured frames be dropped in method MaybeDeliverFrame(), this requires accumulating the per-captured-frame |update_rect|s into per-delivered-frame |update_rect|s. Added test cases to get coverage for the functional changes: * FrameSinkVideoCapturerTest.DeliversUpdateRectAndFrameCounter * FrameSinkVideoCapturerTest.AccumulatesUpdateRectWhenFramesAreDropped	
[Video Capture] Wire |update_rect| through to AdaptedVideoTrackSource::OnFrame()	Functional changes in WebRTCVideoTrackSource: * Keep track of CAPTURE_COUNTER and and CAPTURE_UPDATE_RECT to determine when valid |update_rect| information is available. * Accumulate |update_rect| information when dropping frames before delivery to WebRTC * Mark whole frame as changed if cropping or scaling has changed since the last frame delivery. * Translate |update_rect| from being relative to the entire coded frame data (as delivered) to being relative to the visible_rect of the frame (as expected by WebRTC) * When hard-applying cropping and scaling into a new buffer, transform the |update_rect| information accordingly.	Landed in M74

[Tab Capture] Ensure CAPTURE_UPDATE_RECT aligns with I420 subsample boundaries	When outputting frames in I420 format, the CAPTURE_UPDATE_RECT set by tab capture must align with subsample boundaries. Before this CL the CAPTURE_UPDATE_RECT was computed with single pixel precision. This CL expands the CAPTURE_UPDATE_RECT such that it aligns with the subsample size of 2.	Landed in M75. TODO: Merge to M74
TODO	Let WebrtcVideoFrameAdapter report I420 format if no conversion is needed.	
TODO	TODO: Check whether a full update also has to be triggered after black frames in https://cs.chromium.org/chromium/src/third_party/webrtc/media/base/video_broadcaster.cc?g=0&l=73	

Testing Plan

- TODO: Confirm in manual end-to-end testing that things are working correctly around and after window resizes (both with static and animating/video content), since this edge case is difficult to get full coverage with automated tests on.