Why so aggressive? On the support for Background Transfers in WebRTC

Riccardo Reale
Peerialism AB / KTH - Royal Institute of Technology
riccardo@peerialism.com

Anton Blomberg
Stockholm University
anbl4171@student.su.se

Roberto Roverso
Peerialism AB
roberto@peerialism.com

ABSTRACT

The WebRTC framework enables direct browser-to-browser communication for VoIP, video-conferencing but also generic javascript-based peer-to-peer (P2P) applications. In this paper, we address a major deficiency of the framework, that is the absence of a background transfer protocol. In doing so, we enable the development of un-intrusive data-intensive P2P applications, such as file-sharing and content delivery, on top of WebRTC. This work constitutes the first step in the process of studying and improving WebRTC's network stack in the context of a real use-case, that is a commercial distributed content delivery application.

Categories and Subject Descriptors

C.2.2 [**Computer-Communication Networks**]: Network Protocols; C.2.4 [**Computer-Communication Networks**]: Distributed Systems

General Terms

Experimentation, Performance, Measurement

Keywords

LEDBAT, WebRTC, Peer-to-peer

1. INTRODUCTION

Recently, WebRTC has been steadily gaining popularity, fuelled by large scale deployment in most of the browsers on the market, such as Firefox, Chrome and Opera. The WebRTC standard mandates the multimedia and peer-to-peer connectivity stacks for real-time services, once provided by external plugins, to be built directly into the browser. As a consequence, the features of those stacks, such as video acquisition/encoding, encryption and NAT traversal, are made available to developers through HTML5 APIs. WebRTC was mainly designed as a framework to facilitate video and audio conferencing but it does include support for building other types of peer-to-peer applications, such as CDN accelerators [4] and video streaming platforms [6].

Although WebRTC currently provides many features, it lacks support for background transfers. That is an essential requirement for P2P data-intensive applications, e.g. BitTorrent [5] and streaming platforms [8], to avoid disrupting other traffic in the network while transferring large amount of data. Support for background transfers is usually implemented by a custom transport protocol layered over UDP that includes a special delay-based congestion control, such as for instance the Low Extra Delay Background Transport (LEDBAT). LEDBAT has two main features: first, it completely yields to TCP, and second, it introduces a low and constant amount of delay on a link. The former feature allows LEDBAT to back off to all TCP-based traffic and therefore leave all available bandwidth to, for instance, HTTP web browsing traffic and large file transfers through FTP. The latter feature instead prevents large LEDBAT transfers to disrupt delay- and jitter-sensitive applications, such as voice over ip and gaming. This is in direct contrast to the behaviour of TCP, which introduces an arbitrary high delay on a link with a significant amount of jitter.

In this paper, we present what is, to the best of our knowledge, the first implementation of a background transfer protocol for the WebRTC framework. By providing support for LEDBAT, we enable the development of data-intensive P2P applications on top of WebRTC. In our specific case however, the interest in LEDBAT is motivated by the the needs of a commercial peer-assisted javascript-based video distribution application called Hive [7]. Hive utilizes low priority traffic to prefetch data from other peers ahead of the playback deadline without disrupting other traffic in the network.

Besides including LEDBAT in WebRTC, our greater goal is to provide a comprehensive study of the functionality and performance of WebRTC's transport protocols. This, in order to understand and address possible shortcomings that would prevent wide adoption of this technology. For that, as the next step, we intend to provide a detailed analysis of the behaviour of WebRTC's transport protocols in a real use-case, that is our video streaming platform.

In order to promote adoption of our augmented version of WebRTC, which contains background transfer support and other improvements to the WebRTC stack, we make our code available as open-source at [1].

2. LEDBAT IN WEBRTC

In WebRTC, transfer of arbitrary data over P2P channels is a feature provided by the DataChannel APIs.

The DataChannel implementation currently makes use of the Stream Control Transfer Protocol (SCTP)[1] in order to manage generic data transfers. SCTP is message-oriented and provides a variety of options on data transfer to accomodate the needs of different types of applications, such as in- or out-of-order delivery and configurable reliability. From the congestion control point of view, SCTP is very similar to TCP [2], however it is designed to deliver better throughput in networks with high delay but large amount of bandwidth. Therefore, in general, SCTP is expected to equally share the bandwidth available with other TCP and SCTP transfers and, as TCP, to introduce arbitrary delay on a link. The current version of WebRTC's DataChannel incorporates a user-level SCTP implementation[2] written in *C* and built on top of UDP. SCTP interfaces with a security layer, Datagram Transport Layer Security (DTLS), which adds encryption and integrity to all traffic sent through a DataChannel session. All data sent through that session is carried over connections that have been established using the NAT Traversal protocol.

In order to implement LEDBAT support in WebRTC, we opted to integrate the uTP library [3] in the WebRTC protocol stack as runtime alternative to SCTP. uTP is the reference open-source implementation of LEDBAT and is provided by BitTorrent. Similarly to SCTP, uTP provides reliability, in-order delivery and is layered over UDP. We were able to encapsulate uTP in WebRTC by creating a new data channel type, the LedbatDataMediaChannel. This facility interfaces uTP with the other WebRTC components such as those dedicated to security (DTLS) and connection establishment (ICE protocol for NAT traversal). A LedbatDataMediaChannel instance is created and managed by a LedbatDataEngine during is lifetime. LedbatDataEngine also registers all callbacks from the uTP sockets and handles the control messages coming from the network for the setup and tear down of transfer channels. Moreover, since the DataChannel abstraction handles messages rather than data streams, the LedbatDataEngine also acts as a translation layer between the generic DataChannel APIs and the uTP library. The translation is performed by encapsulating outgoing messages using a header consisting in a Payload protocol identifier (PPID) and message length, which later allows to correctly reconstruct incoming packets into messages at the receiver.

3. PRELIMINARY RESULTS

We conducted a preliminary evaluation of our solution in a controlled network environment consisting of two host machines, a sender and a receiver, running Ubuntu Linux with kernel 3.11.0-12 connected through a 1 Mbps link with 100 ms base delay and 60KB of queue buffer to simulate a typical last-mile link.

Both hosts run a Chromium browser built with the modified WebRTC module, which can be configured to use either a STCP DataChannel, or a LEDBAT DataChannel. We then use an HTML5 application to generate DataChannel transfers from the sender to the receiver host, and a separate TCP traffic generator to obtain traffic competing on the same network resources.

As metrics, we utilize the throughput of the DataChannel LEDBAT transfers when competing against TCP traffic, and the Round Trip Time (RTT) measured on the link. With these two metrics, we verify that LEDBAT transfers indeed yield to TCP and, at the same time, we evaluate the relative increase on the one-way delay, which directly affects latency and, therefore, web browsing responsiveness.

[1]http://www.ietf.org/rfc/rfc2960.txt
[2]http://sctp.fh-muenster.de/sctp-user-land-stack.html

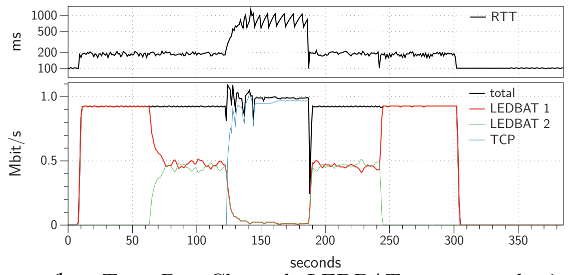

Figure 1: Two DataChannel LEDBAT streams sharing a 1Mbit/s bottleneck with a TCP stream

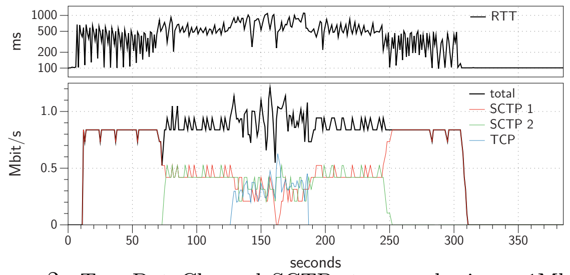

Figure 2: Two DataChannel SCTP streams sharing a 1Mbit/s bottleneck with a TCP stream

Figure 1 shows the throughput evolution of two DataChannel LEDBAT streams and a single TCP stream, starting at 60 seconds from each other. The corresponding Round Trip Time is also shown above. The two LEDBAT flows, as expected, utilize all available bandwidth, until they quickly and completely release the bottleneck in favour of the higher priority TCP flow. The increase in the RTT due to the presence of LEDBAT flows is limited to 100 millisecond, which is the default target delay configuration in uTP. Note that the number of LEDBAT streams doesn't affect the amount of extra delay introduced.

For direct comparison and validation that the normal behaviour of SCTP was not affected, Figure 2 shows the same experiment performed configuring our test application to use DataChannel SCTP streams. Besides fairly sharing the link with the TCP transfer as expected, each of the SCTP streams introduces a large amount of extra-delay on the link as TCP does.

4. REFERENCES

[1] https://github.com/Peerialism/webrtc-ledbat.
[2] http://www.hamilton.ie/net/htcp.htm.
[3] https://github.com/bittorrent/libutp.
[4] Peercdn. https://peercdn.com/.
[5] Pouwelse et al. The bittorrent p2p file-sharing system: Measurements and analysis. In *Peer-to-Peer Systems IV*. Springer, 2005.
[6] Nurminen et al. P2P media streaming with HTML5 and WebRTC (Demo). In *INFOCOM*, 2013.
[7] R. Roverso et al. Hive.js: Browser-Based Distributed Caching for Adaptive Video Streaming. 2014.
[8] R. Roverso, S. El-Ansary, and S. Haridi. Smoothcache: Http-live streaming goes peer-to-peer. In *IFIP NETWORKING 2012*.