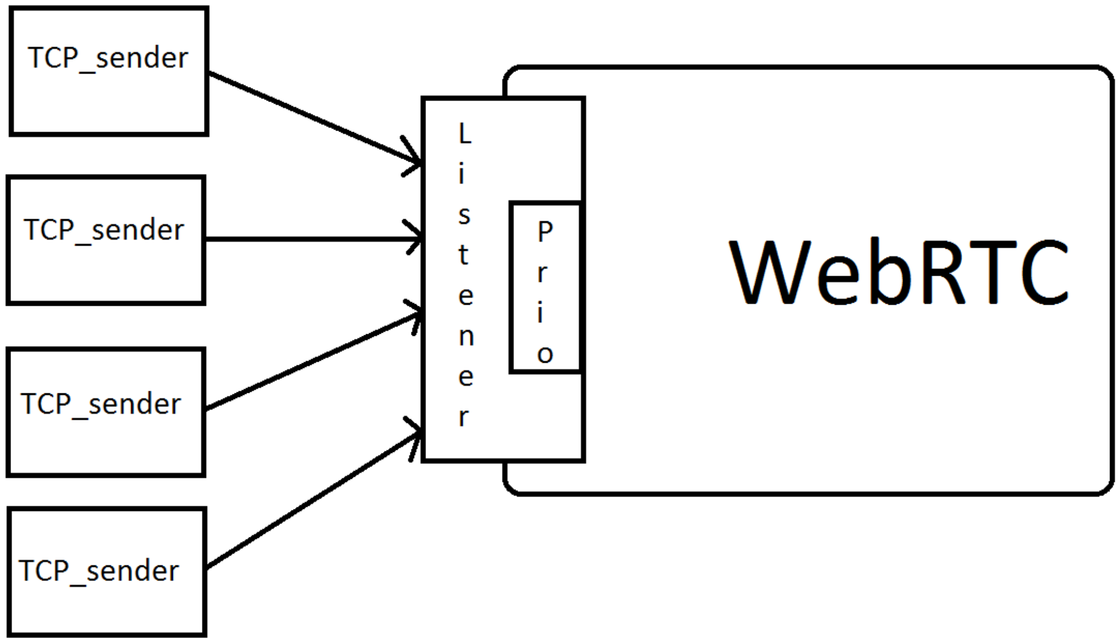
Kommunikation D0020E 2014/2015 WebRTC.

Summary

We are a group of four people that have done research on WebRTC and the possibility to use it as the communication foundation to remote control a wheel loader. This document should be a quick course on how far we have come, with the reaserch and should work maybe as a foundation for future projects.

Where we are today

We use today an example that creates a datachannel, between a natively application and a webbrowser, the protocoll that is used in this connection is SCTP.
We have modified this example, so that the “example” in WebRTC is listening on a port, that then forwards all the data that is sent to that port, in the format of a string.
We have also created a simple program that sends the data trough a TCP connection, to the given port that our application is listening to.
We have also begun on building the foundation on a prioritizing system, that uses multiple FIFO queues.



Versions

Operativ system	Linux Ubuntu, 14.04 LTS
WebRTC	a7384a1126cda7ce726f73b023bad997627fc138
Ninja	
gcc/g++	4.8.2

GitHub repository for files.

https://github.com/pottzer/ProjectKommunikationD0020E

How to get WebRTC:

Due to WebRTC changing dramatically during our work with it, and as such the version above is the highest we know work with our programs. Below are instructions to obtain this code.

1. Follow the instructions laid out at http://www.webrtc.org/native-code/development
2. When you get to Getting the Code, instead of running
   ```
   $ git checkout master
   ```
   run
   ```
   $ git checkout <hash>
   ```
   where <hash> is the hash is the one noted above beside WebRTC.
3. Follow the rest of the instructions. However, DO NOT run
   ```
   $ git pull
   ```
   as it will update the repository to the latest version instead of the one from December 2.

TCP_sender

Files

TCP_sender.c
TCP_sender_test.c (Unit test)
Server.c

Format

```
$ ./TCP_sender "ip<int>" "port<int>" "data<string>"
$ ./TCP_sender_test "ip<int>" "port<int>" "number<int>(number of tests to run)"
```

How it works.
We send the data in the format of a string to the given port that our “listener” is listening to.
Our listener is built in the example that we used as our base.
Build with gcc.

To do
As of today we can only send using this application. We also need to be able to recive data on the same computer for it to be usefull.
We have also included the file Server.c which is the listener. There is a possibility to use that as the foundation for the application to recieve data.

Prioritybuffer.cc

Files
prioritybuffer.cc
pbuffer_unittest.cc (unittest)

Format
```
$ ./pbuffer_unittest
```

How it works
The prioritybuffer is in its essence a vector of queues, in which messages of various messages are stored for later sending. Higher index of the vector means lower priority. The prioritybuffer file itself is not runnable, but is meant to be imported and run.

The pbuffer_unittest is a unittest of the prioritybuffer. It tests various aspects of the prioritybuffer, and is run as-is with no arguments. It is also a useful example of prioritybuffer usage.

Build with g++.

To do
The prioritybuffer would need a .h file for easy usage.
It would also benefit from being integrated into the sending/recieving part, so that input can be buffered.
For further modifiability, consider implementing a custom FIFO queue instead of the default C++ queue.

webrtc_cmd.cc

Files

webrtc_cmd.cc
ListenerSocket.cc
ListenerSocket.h
WebRtcConnectionManager.cc
WebRtcConnectionManager.h

Format

```
$ ./webrtc_cmd
```

How it works

The application creates a connection to the browser application. A SCTP-datachannel is created between the native application and the browser application. The browser is able to chat with the native application and the native application forwards data from tcp_sender.

Tutorial - Adding webrtc_cmd to ninja build list

1) Place the "`sendData2013-2014`" file in "`webrtc/src/talk/examples/peerconnection`".
2) Replace the "`webrtc/src/talk/libjingle_examples.gyp`" file with the new `libjingle_examples.gyp` file in our directory.
3) Run
   ```
   $ gclient sync --force
   ```
   from "`webrtc/src`"
4) Compile webrtc using ninja as instructed in the WebRTC instructions. The executable file `webrtc_cmd` should now appear in `webrtc/src/out/Debug`.

Tutorial - Opening and sending through tcp and webrtc datachannel:

(This assumes you've built webrtc_cmd via ninja)

1. From the `serverless-webrtc` directory, run
   ```
   $ python -m SimpleHTTPServer 8001
   ```
2. Execute "
   ```
   $ ./webrtc_cmd
   ```
   inside `webrtc/src/out/Debug`
3. Enter
   ```
   print
   ```
   into the prompt and copy the "offer link" presented to you.
4. Browse to http://localhost:8001/ in a web browser, open "serverless-webrtc.html" and click Join. Paste the "offer link" from the previous point into the textbox and click Okay. You'll get a reply string back.
5. Paste the reply from the browser into the `webrtc_cmd` terminal and hit Enter.

6. Click "Okay, I sent it!" in the browser. There should now be a connection between the browser and the terminal.
7. The browser is now able to send text to the native application. The client cannot reply by itself.
8. Unless you've already done so, build `tcp_sender.c` using
   `$ gcc tcp_sender.c -o tcp_sender`
9. Run the `tcp_sender` program, either on the same machine or another with
   `$ ./tcp_sender <string>ipaddress <int>port <string>message`
   Use the IP of the machine running webrtc_cmd (can be `localhost`), and default port is 2222.
10. This will pass the message string to the `webrtc_cmd` program, which then passes it on via a WebRTC SCTP datachannel to the browser.

The basics are taken from
https://groups.google.com/forum/#!searchin/discuss-webrtc/c$2B$2B$20data|sort:date/discuss-webrtc/bNUgF7BK09Q/klTIjipq4VEJ

To do

The browser side is to be replaced with a native application.
Establish a connection should be done by a communication server, like in the peerconnection example.
The tcp-port listener in webrtc_cmd.cc should be placed in its own thread to add simultanous read and write from the priority queues.

Contact information:

Pontus Ridderström	: ponrid-2@student.ltu.se
David Boman	: davbom-2@student.ltu.se
Fredrik Grönlund	: fregrn-2@student.ltu.se
Jesper Gladh	: jesgla-2@student.ltu.se