Adaptive video streaming with Ice and GStreamer

Using ICE middleware with GStreamer to implement real-time QoS-aware video streaming for remotely controlled vehicle.

Andrey Nechypurenko
andreynech@gmail.com
Maksym Parkachov
lazy.gopher@gmail.com

GStreamer Conference, 2010

Outline

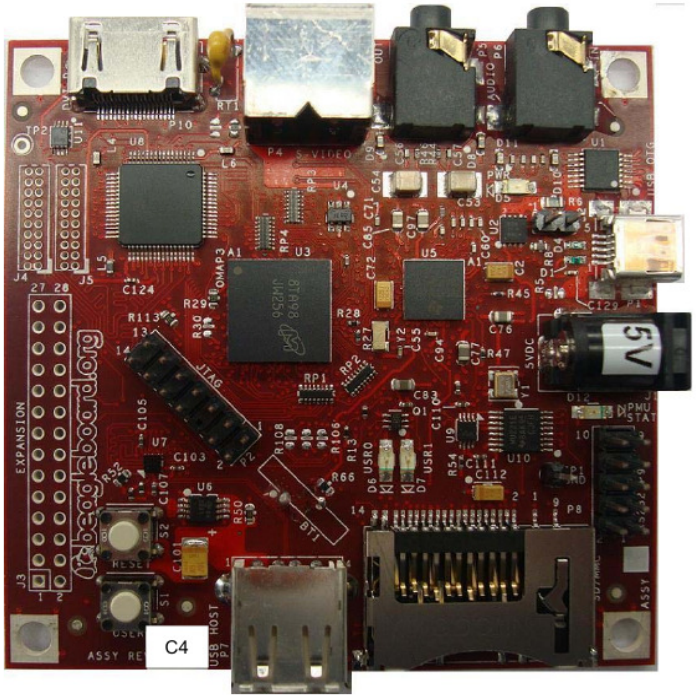
Let's build the vehicle and control it over the Internet

As a hobby project, we start developing small vehicle equipped with on-board computer connected to WLan adapter and web-camera. The idea was to control the car over Internet.

Main challenges

- Find or build mechanical platform
- Build electronic which can:
  - Capture video from camera
  - Compress live video stream to h264 format in real-time
  - Support W-WLan connectivity
  - Have enough IO channels to control motors
- Develop software which can:
  - Deliver video stream and sensor data to the remote driver
  - Display live video stream to the driver
  - Receive user input such as steering and acceleration
  - Deliver control commands from the driver to the vehicle software and drive actuators
  - Support client and server NAT and firewall traversal without the need to configure firewall on the client side
  - Provide high quality video under variable network conditions

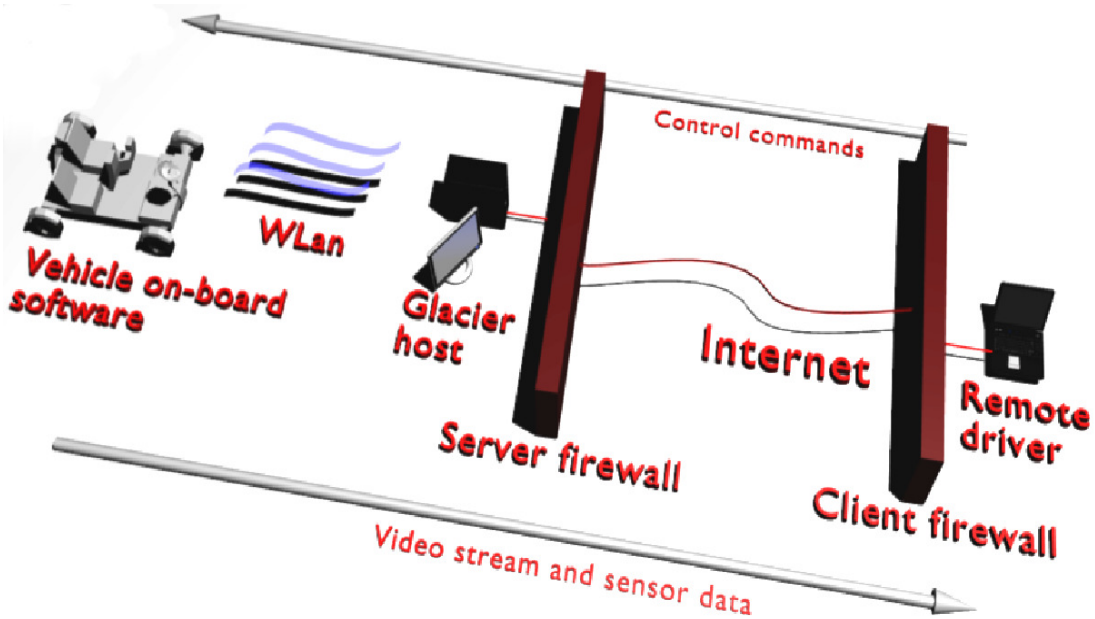
Hardware solutions



Hardware used

- BeagleBoard C4
- Logitech 9000Pro camera
- USB HUB
- D-Link USB WLan (rt73usb)
- SparkFun logic level converter
- SparkFun DC/DC converter breakout
- Reely buggy

High level system overview



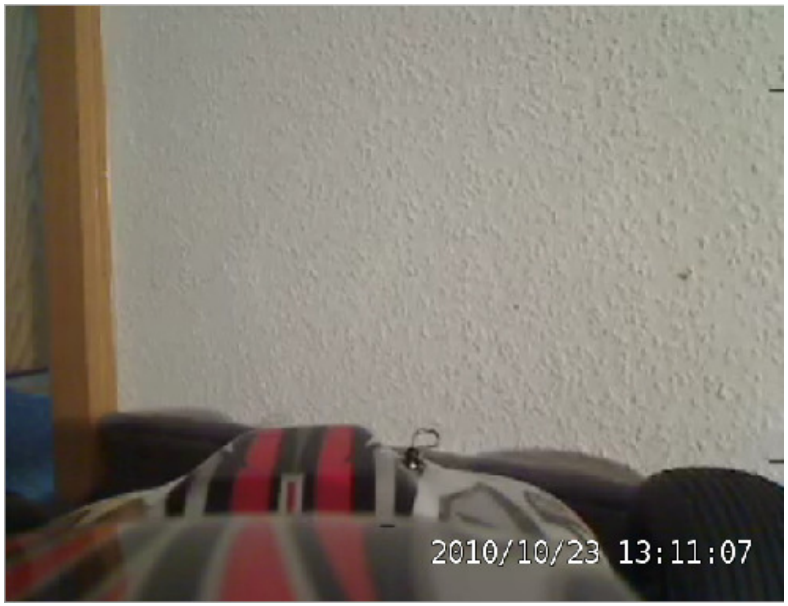
The whole system software has two main tasks:

- Deliver video stream and sensor data from the vehicle to the remote driver and
- Deliver control commands from the driver to the vehicle software and drive vehicle actuators.

Communication happens over the Internet where it is typical to have two firewalls (and/or NATs) on the client and server side.

Need for adaptive video streaming

It is important to provide constant frame rate with predictable latency to precisely control the vehicle. Otherwise, the wall may suddenly appears ahead of the car :-) .



Negative effect of the changing network conditions leads to:

1. Delays in video stream (picture freezes).
2. Corrupted frames (because of dropped frames, etc.).
3. “Fast forward” effect when the next chunk of data arrives.

Solution - QoS aware streaming

The adaptive Quality of Service (QoS) aware streaming implementation is required to let the driver precisely control remote vehicle.

To solve the problems mentioned above:

- Use TCP instead of UDP to get better feedback about packet delivery status.
- Permanently monitor the size of output queue with compressed frames to deduce the current QoS conditions.
- Define the set of states characterized by maximum and minimum queue size to transition to the better or worth state. In addition, the time-based weight is introduced which is calculated based on the overall time spent in certain state. It prevents the system from permanently jumping from one state to another.
- React on the state changes by dynamically adjusting frame size and codec target bitrate (if it is supported by codec).

The core adaptation logic resides in the vehicle on-board application.

Main software modules

To provide this functionality two main modules are required:

- **Driver application** which will be further reffered as **cockpit**.
- **Vehicle on-board application** which will be reffered as **vehicle**.

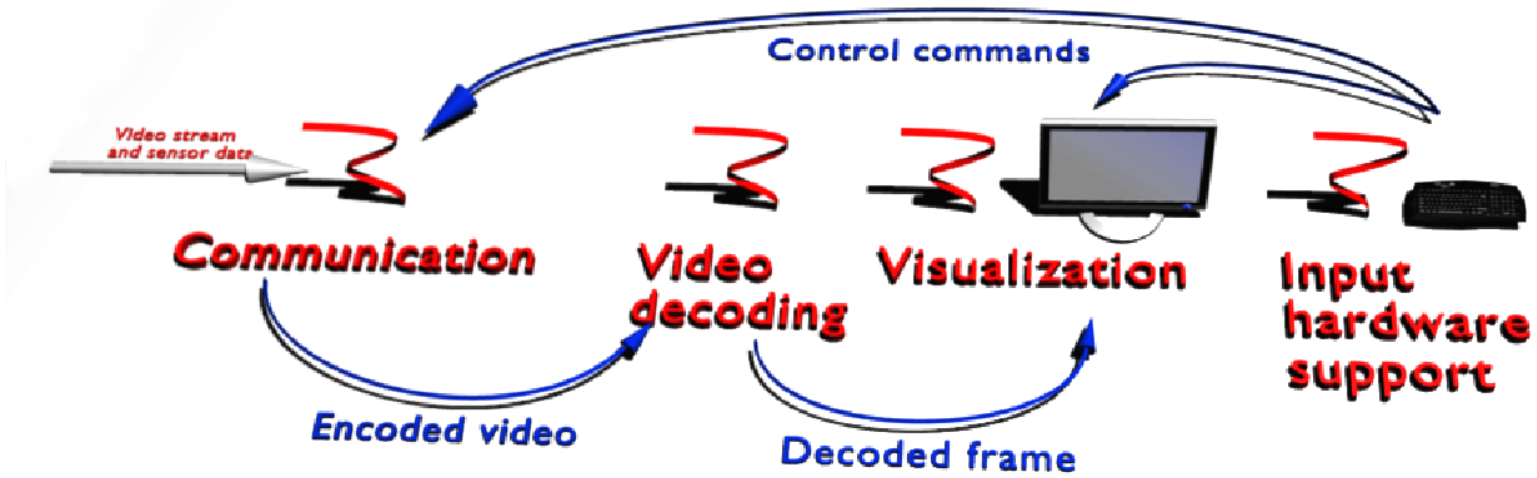
In addition, to perform firewall traversal in the secure and efficient way, additional application is required on the server side. In this project, ZeroC Ice open-source middleware is used for all communication needs.

Implementation strategy

The vehicle on-board application has the following responsibilities:

- Receive control commands (such as steering and acceleration) from remote driver
- control connected accelerators based on the received commands. In particular, send motor control commands over I2C interface.
- Capture video from camera and compress it in real-time.
- Send captured video to the cockpit application.
- **Collect statistic** about network bandwidth to perform adaptation in case of changed network conditions. In particular, the frame size could be reduced, compression rate could be increased or even frame rate could be reduced if the bandwidth is not enough to deliver the video on time.

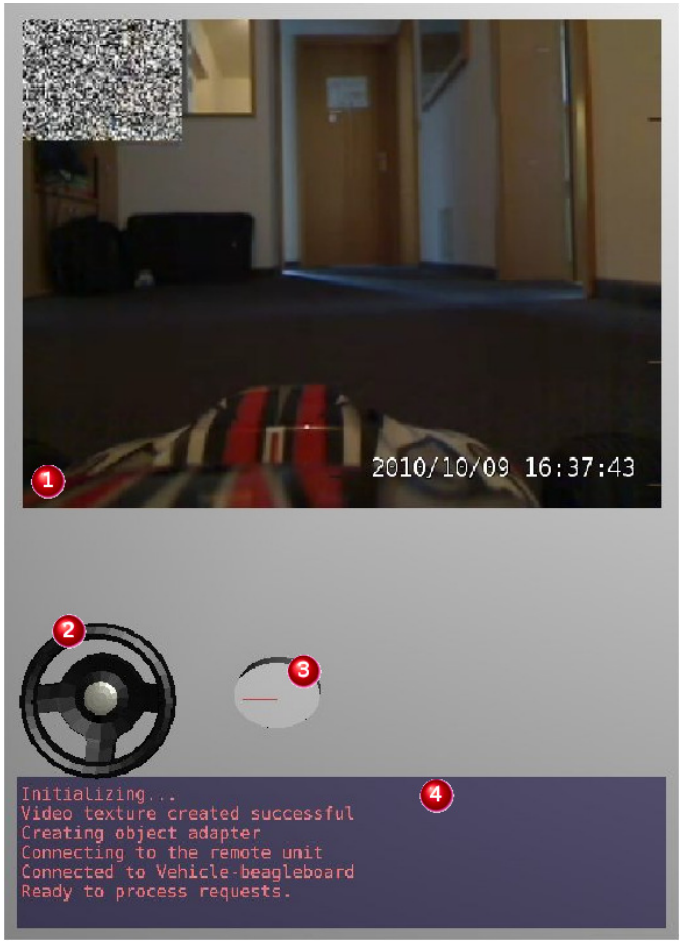
Cockpit application.

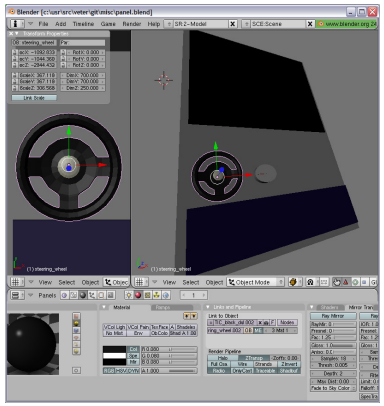


Cockpit application

- Cockpit application is responsible for receiving video stream, decoding, and visualizing it.
- In addition cockpit application receives control signals from input hardware and transmit commands to the vehicle.
- Implementation is heavily multithreaded and uses graphics hardware acceleration.

Cockpit application.



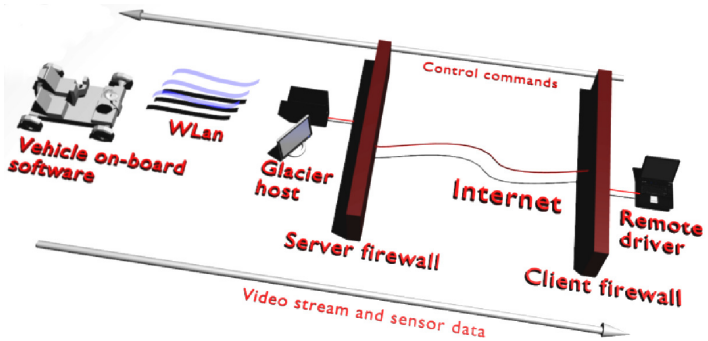


Cockpit application is written mainly with OpenGL. The whole interface is designed as a 3D model in Blender. There are four main elements:

1. Video plane - surface where each video frame is placed as a texture.
2. Steering wheel - to provide visual feedback to the steering actions of the driver.
3. Tachometer - to provide visual feedback to the acceleration actions.
4. Message area - surface where messages will appear in form of 3D text slightly shifted towards the viewer.

There is a desperate need for 3D artist :-) .

Alternatives for streaming implementations



Two implementation options were considered:

- Using GStreamer RTSP server.
- Custom transport protocol for RTP payload using Ice middleware.

To make final decision, both variants were implemented and compared.

Reasons why Ice was considered as communication solution:

- **Reduce complexity** when implementing complex bidirectional communication.
- **Transparently handles cross-platform issues** such as endianess.
- **There are two versions of Ice.** IceE which has reduced footprint and easier to cross-compile. The complete version provides the full set of functionality.
- **Very easy to change the communication protocol.** Changing between UDP, TCP or SSL is the matter of change endpoint description in the configuration file.
- There is service application Glacier which **solves firewall/NAT related problems.**
- Asynchronous Method Invocation (AMI) which makes possible to **collect more information about bufferization and transmission performance**.

Test setup

The following pipeline is currently used on the BeagleBoard for video capturing and encoding. There is a placeholder (videotestsrc) for rear-view camera which is not yet mounted on the current model (but tested, and it works).

Capturing and encoding pipeline:

```
videotestsrc pattern="snow" ! video/x-raw-yuv, framerate=30/1,
width=80, height=60 ! videomixer name=mix ! videoscale
name=qos-scaler ! capsfilter name=qos-caps
caps=video/x-raw-yuv, width=320, height=240 ! ffmpegcolorspace
! video/x-raw-yuv, format=(fourcc)UYVY ! TIVidenc1
codecName=h264enc engineName=codecServer bitRate=320000
encodingPreset=2 genTimeStamps=TRUE byteStream=TRUE ! rtph264pay
pt=96 ! appsink name=icesink v4l2src always-copy=FALSE !
video/x-raw-yuv, width=320, height=240, framerate=30/1 ! mix.
```

Based on the observed QoS conditions and derived system state, caps of the qos-caps capsfilter are dynamically adjusted to control frame size.

Test setup

The following pipeline is currently used by the cockpit application.

Decoding pipeline:

```
appsrc ! application/x-rtp, encoding-name=(string)H264,
payload=(int)96 ! gstrtpjitterbuffer latency=10 ! rtph264depay
! video/x-h264, framerate=30/1 ! decodebin2 ! videoscale !
video/x-raw-yuv, width=640, height=480, framerate=30/1 !
clockoverlay halign=right valign=bottom time-format="%Y/%m/%d
%H:%M:%S" ! ffmpegcolorspace ! video/x-raw-rgb, bpp=24,
depth=24 ! fakesink sync=1
```

Frame hand-off mechanism provided by `fakesink` element is used to get access to raw decoded frames. Cockpit application then uses received frame to create and update OpenGL texture which is mapped to the placeholder plane defined by the 3D interface model.

Test setup

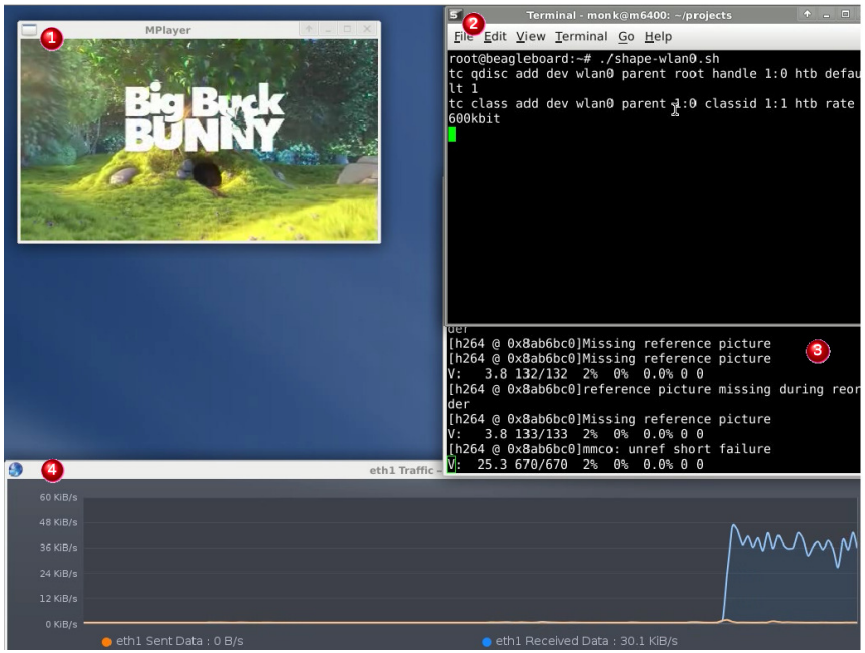
For test purposes, slightly modified encoding pipeline is used to feed the prerecorded video. Original file is scaled down on the fly to make the test comparable with live streaming from web-camera.
Pipeline for cockpit application was not changed.

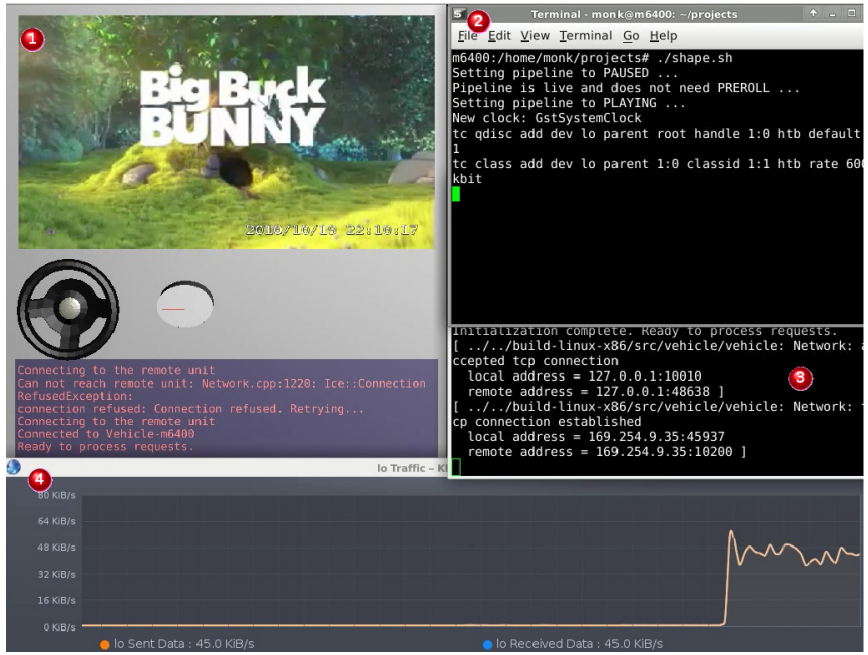
Encoding pipeline for the vehicle application test

```
filesrc location=big_buck_bunny_480p_h264.mov ! qtdemux !
ffdec_h264 ! videoscale name=qos-scaler ! capsfilter
name=qos-caps caps=video/x-raw-yuv,width=320,height=240 ! tee
name=t ! queue2 ! x264enc name=encoder byte-stream=true
bitrate=300 speed-preset=superfast subme=6 ! rtph264pay pt=96 !
appsink name=icesink t. ! queue2 ! ffmpegcolorspace !
autovideosink
```

Test setup

Linux kernel network traffic shaping capabilities are used to restrict the available bandwidth and see how both streaming implementations would react on it.





1. MPlayer video output window.
2. Terminal window where traffic shaping commands are issued.
3. MPlayer console output.
4. KNemo network monitor window.

1. **Cockpit application** window.
2. Terminal window where traffic shaping commands are issued.
3. **Vehicle application** console output.
4. KNemo network monitor window.

Adaptation performance test for GStreamer RTSP server

SCREEN-CAST is provided as a separate download.

screencast-rtsp.ogv - Theora video

Link: https://docs.google.com/leaf?id=0BzV4szKbuvKwNDBhZTMyMTItMGM5Zi00NGFmLThlOGMtMjAzYjlmYWZjMDQx&hl=en

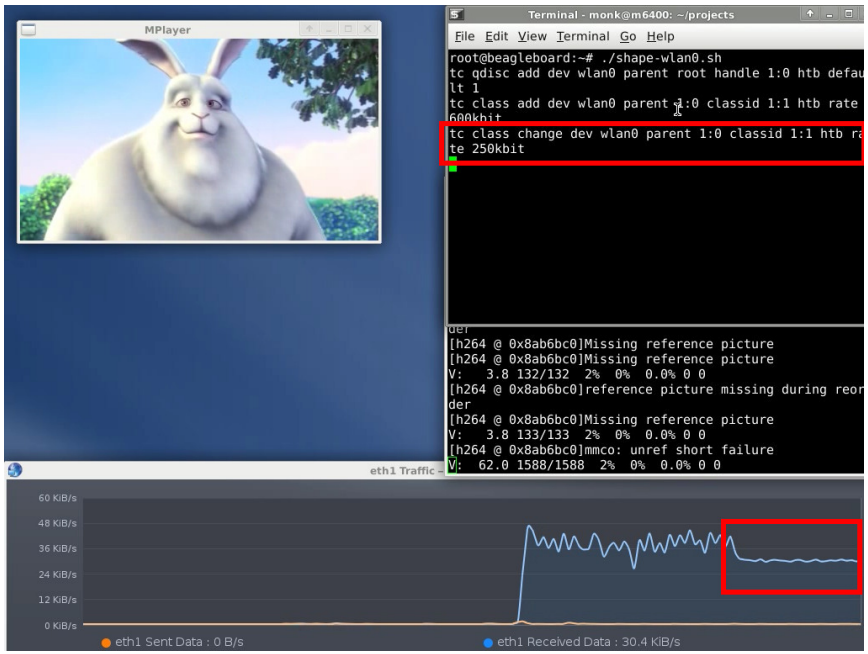
screencast-rtsp.avi - Xvid mpeg4

Link: https://docs.google.com/leaf?id=0BzV4szKbuvKwN2E2MmFlMDctM2MxYy00YjJhLWIyMjItOTUwNjRmZjhmODBk&hl=en

GSTC10_Nechypurenko_Parkachov.zip

There is also a bundle with mpeg4 (avi) files, short readme and PDF presentation with embedded video available. **Link:** https://docs.google.com/leaf?id=0BzV4szKbuvKwNTEwYTQwOGUtNzAzNi00NjQ5LTk2MjEtMWIzYTQzNjVkZTRh&hl=en

Important observations for RTSP server



When available bandwidth decreased below the level required to transmit the video with current encoding parameters (resolution, frame rate, quality, etc.):

- Communication channel is saturated.
- Displayed frame rate is significantly reduced.
- No adaptation is occurred.
- “Old” frames might be also dropped which leads to lowered framerate compared to the original video.

Conclusion:

Such behavior is unacceptable for real-time streaming required to control remote vehicle.

Adaptation performance test for VETER infrastructure

SCREEN-CAST is provided as a separate download.

screencast-veter.ogv - Theora video

Link: https://docs.google.com/leaf?id=0BzV4szKbuvKwODc5NmI2OGYtMzNiZi00YzkxLWEwNGYtM2JhY2JmYTBjZjIz&hl=en

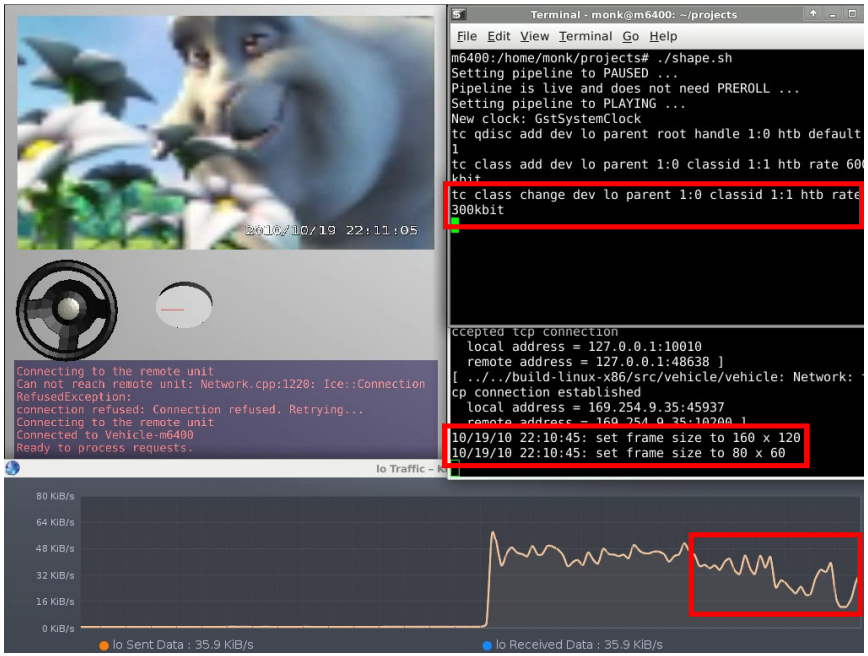
screencast-veter.avi - Xvid mpeg4

Link: https://docs.google.com/leaf?id=0BzV4szKbuvKwMTcxNjg2ODQtYmMwNi00OTQ0LWE5MmItMTlhMDU3OWI5ZGY2&hl=en

GSTC10_Nechypurenko_Parkachov.zip

There is also a bundle with mpeg4 (avi) files, short readme and PDF presentation with embedded video available. **Link:** https://docs.google.com/leaf?id=0BzV4szKbuvKwNTEwYTQwOGUtNzAzNi00NjQ5LTk2MjEtMWIzYTQzNjVkZTRh&hl=en

Important observations for VETER infrastructure



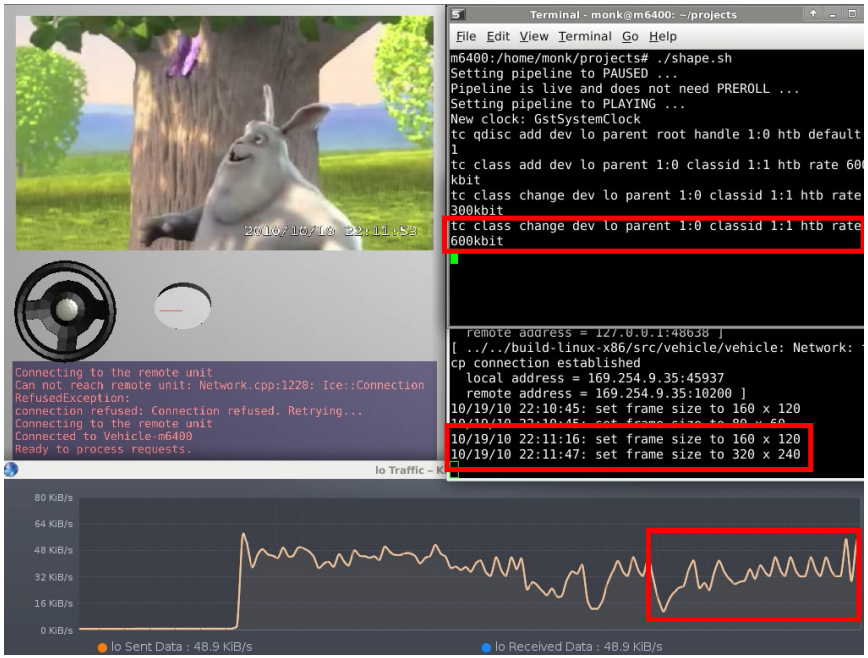
When available bandwidth decreased:

- Vehicle application detects changed networking condition.
- Reacts by reducing frame size (current implementation divides the original size by the power of two for each state).
- Keeps the same frame rate as original video.
- As a result, used bandwidth is below current limit and is not saturated.

Conclusion:

Quality of the video is reduced but frame rate remains the same, which is important for real-time remote control.

Important observations for VETER infrastructure



- After preconfigured amount of time, attempts to return to the original video quality (frame size).
- If required bandwidth available, continues to increase frame size.

Conclusion:

Improved bandwidth usage and as a result best possible quality with constant frame rate is presented to the drivers.

Summary

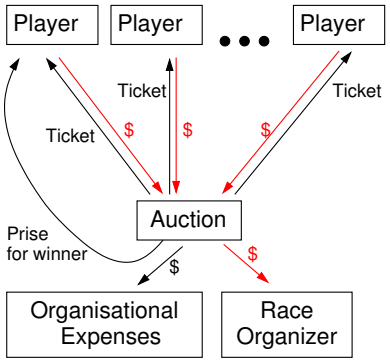
Conclusions:

- Currently there is no out of the box solution for adaptive real-time video streaming suited for embedded system.
- Using GStreamer and Ice it is possible to build such solution.
- To react on network QoS changes, frame size could be changed on the fly using videoscale or appropriate hardware accelerated elements like for example TIVidScale.

Future work:

- **Transform the car to remotely controlled quad-copter.**
- Investigate if it is possible to change encoder's target bitrate on the fly. If yes, which encoders does really support it. In particular if it is supported by DSP-based encoders.
- Experiment with two cameras and stereo vision.
- Write more documentation (in particular on how to cross-compile the project with OE/Angstrom) to attract more contributors.
- Earn the first million with it. See the nexst slide to get to know how... ;-)

Robot races



“Driver” can buy “ticket” (SSL certificate) on auction such as eBay and participate in the race from all over the world. Collected money are to pay prize for winners, cover organizational expenses and make profit.

This is a good example how the race track can looks like. With good Internet connection it is perfect place to conduct races.

We are looking for business partner

We are **completely prepared from the technical point of view**. However, we have no marketing, financial, administrative, etc. experience and **looking for the partner(s) who can help us from the business side**.

Please feel free to contact us at: andreynech@gmail.com

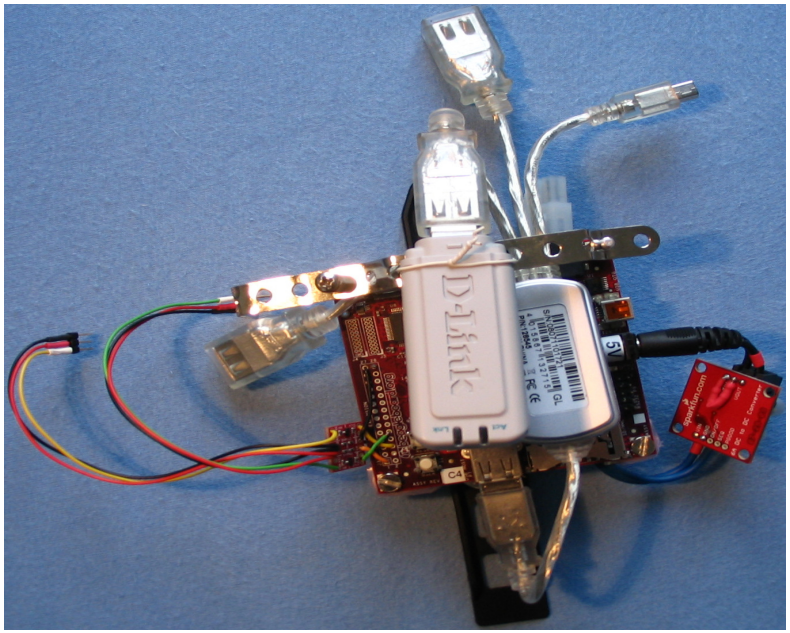
Source code and information availability I

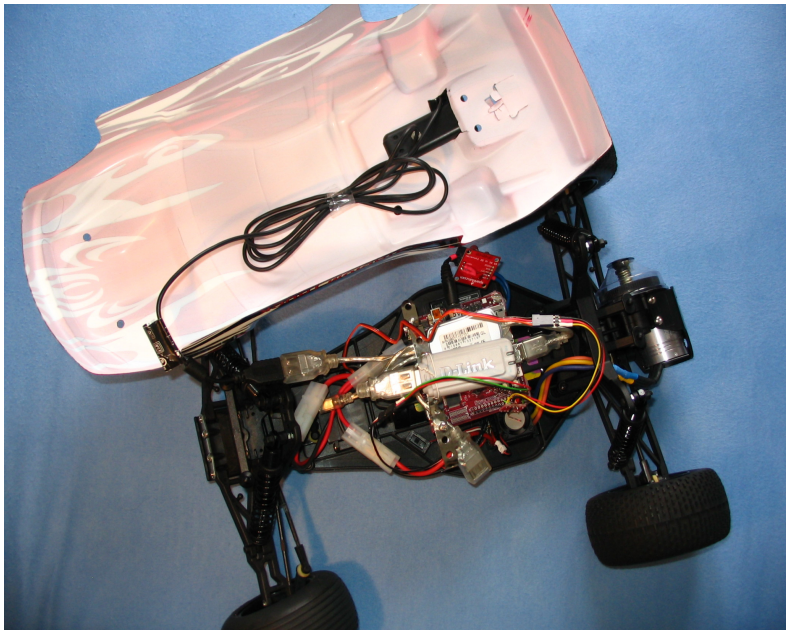
Blog:
http://veter-project.blogspot.com
VETER-project blog with regular updates about the project.

Gitorious:
http://www.gitorious.org/veter
Complete source code and documentation repostiroy (including this presentation).

Wiki:
http://www.gitorious.org/veter/pages/Home
General information about project and further relevant links could be found here.

Main parts assembled together





Wiring

On the picture at the left hand side there are two connectors for the servo-controllers (steering and acceleration). They are connected to the BeagleBoard's +1.8V GPIOs through level converter to provide +5V required for servos. The board is powered through DC-DC converter to provide stable +5V power from the battery.
The right hand side picture illustrates how electronic components are mounted on the car. There is also camera fixed on the plastic cover.

Remote interface definition

```
interface StreamReceiver {
  ["ami"] void nextChunk(ByteSeq chunk);
  idempotent QoSReport getQoSReport();
};
interface MotorControl {
  idempotent void setDuties(MotorDutySeq duties);
};
interface RemoteVehicle {
  . . .
  idempotent MotorControl* motorControlInterface();
  idempotent void addStreamReceiver(StreamReceiver *callback);
};
```

- `RemoteVehicle` and `MotorControl` interfaces are implemented within vehicle application.
- `StreamReceiver` is the callback interface to receive compressed video and sensor data (if any). It is implemented by the cockpit application.
- Invocations on `StreamReceiver` interface are performed using Ice Asynchronous Method Invocation (AMI) which makes possible to **collect more information about bufferization and transmission performance**. This information is used to to derive QoS state and **trigger adaptation actions**.