How to evaluate MPEG video transmission using the NS2 simulator?

I extended Jirka klaue's work, EvalVid - A Framework for Video Transmission and Quality Evaluation. I wrote some codes to integrate Evalvid into NS2. Therefore, researchers can easily experiment with MPEG video transmission using the NS2 simulator. One can download the file from here. But, remember to send me an email and add the following paper in your reference if you want to use this tool.

Chih-Heng Ke, Ce-Kuen Shieh, Wen-Shyang Hwang, Artur Ziviani, "An Evaluation Framework for More Realistic Simulations of MPEG Video Transmission",
Journal of Information Science and Engineering (accepted) (SCI, EI)

In the beginning, I describe how to integrate Evalvid into NS2. These steps include some modifications of original ns2 files and addition of my files. One can find what I have done to original ns2 files by searching the keywords "Henry or smallko" that are my nicknames.

p.s. If you want to use the codec provided by this tool-set, you must use windows system. I have no source code for the NCTU codec. If you want to run the simulation under Linux platform, I suggest that you use my newer tool-set at http://140.116.164.80/~smallko/ns2/myevalvid2.htm.

(Video Demo: how to insert those codes into NS2)
Method 1:

1. ***Put a frametype_ and sendtime_ field in the hdr_cmn header. The frametype_ field is to indicate which frame type the packet belongs to. I frame type is defined to 1, P is defined to 2, and B is defined to 3. The sendtime_ field is to record the packet sending time. It can be used to measure end-to-end delay.***

   Modify the file packet.h in the common folder

```
struct hdr_cmn {
  enum dir_t { DOWN= -1, NONE= 0, UP= 1 };
  packet_t ptype_;      // packet type (see above)
  int     size_;              // simulated packet size
  int     uid_;        // unique id
  int     error_;             // error flag
  int     errbitcnt_;     // # of corrupted bits jahn
  int     fecsize_;
  double      ts_;            // timestamp: for q-delay measurement
  int     iface_;             // receiving interface (label)
  dir_t direction_;       // direction: 0=none, 1=up, -1=down
  // source routing
        char src_rt_valid;
  double ts_arr_; // Required by Marker of JOBS
 //add the following three lines
  int frametype_;             // frame type for MPEG video transmission (Henry)
  double sendtime_;  // send time (Henry)
  unsigned long int frame_pkt_id_;
```

2. ***Modify the file agent.h in the common folder***

```
class Agent : public Connector {
 public:
  Agent(packet_t pktType);
  virtual ~Agent();
  void recv(Packet*, Handler*);


. . . . . .
  inline packet_t get_pkttype() { return type_; }
  // add the following two lines
  inline void set_frametype(int type) { frametype_ = type; } // (Henry)
  inline void set_prio(int prio) { prio_ = prio; }  // (Henry)

 protected:
  int command(int argc, const char*const* argv);


. . . . . .
  int defttl_;                      // default ttl for outgoing pkts
 // add the following line
  int frametype_;             // frame type for MPEG video transmission

. . . . . .
 private:
  void flushAVar(TracedVar *v);
};
```

3. ***Modify the file agent.cc in the common folder***

```
Agent::Agent(packet_t pkttype) :
  size_(0), type_(pkttype), frametype_(0),
  channel_(0), traceName_(NULL),
  oldValueList_(NULL), app_(0), et_(0)
{
}

. . . . . .
Agent::initpkt(Packet* p) const
{
  hdr_cmn* ch = hdr_cmn::access(p);
  ch->uid() = uidcnt_++;
  ch->ptype() = type_;
  ch->size() = size_;
  ch->timestamp() = Scheduler::instance().clock();
  ch->iface() = UNKN_IFACE.value(); // from packet.h (agent is local)
  ch->direction() = hdr_cmn::NONE;

  ch->error() = 0;        /* pkt not corrupt to start with */
 // add the following line
  ch->frametype_= frametype_;
. . . . . .
```

4. **Create a "mympeg" folder under ns-2.27 and put *myudp.cc*, *myudp.h*, *myudpsink2.cc*, *myudpsink2.h*, and *mytraffictrace2.cc* in it.**

5. ***modify the file tcl/lib/ns-default.tcl***
   Add the following two lines

```
Agent/myUDP set packetSize_ 1000
Tracefile set debug_ 0
```

6. ***Modify the Makefile***
   Put **mympeg/myudp.o**, **mympeg/myudpsink2.o** and **mympeg/mytraffictrace2.o** in the **OBJ_CC** list

7. ***Recomplie NS2***

```
cd ns-allinone-2.27/ns-2.27
make clean; make
```

Method 2:
Thanks for Guohan Lu's help.

1. Download the patch file. (here)

2. Just put the file mympeg-patch-to-ns2.28.patch in ns-2.28/ and run the following command

```
patch -p1 < mympeg-patch-to-ns2.28.patch
```

3. Modify the Makefile
   Put **mympeg/myudp.o**, **mympeg/myudpsink2.o** and **mympeg/mytraffictrace2.o** in the **OBJ_CC** list

4. Recomplie NS2

```
cd ns-allinone-2.27/ns-2.27
make clean; make
```

[**Example**]
This example only runs on cygwin environment. If one wants to run on different environment. One can recompile the codes. All codes are coded in C.

1. Encoding a yuv sequence into MPEG4 data format (Encoder / Decoder are from http://megaera.ee.nctu.edu.tw/mpeg/) In this example, I use foreman_qcif.yuv as an example. This sequence has 400 frames.

```
mpeg4encoder.exe example.par
```

2. One can get the compressed MPEG4 file **foreman_qcif.cmp** in cmp/one folder
3. Use MP4.exe (from Evalvid) to record the tracefile sender. Each frame will be fragmented into 1000 bytes for transmission. (Maximun packet length will be 1028 bytes, including IP header (20bytes) and UDP header (8bytes).)

```
MP4.exe –send 224.1.2.3 5555 1000 foreman_qcif.cmp > st
```

4. First, I use a simple topology to test the MPEG4 video delivery over a best-effort service network. The simulation environment is shown below. S1 will use the data in the file st to transmit packets to D1. The script file can refer to **be.tcl**. One thing has to be mentioned here. In the tcl script file, the video frame is snet every 33.33 ms for 30 frames/sec video. I don't use the time recorded in the file st.

S1 ----> R1 --- >R2 ----> D1

```
ns2 be.tcl
```

5. After simulation, ns2 will create two files, **sd_be** and **rd_be**. The file sd_be is to record the sending time of each packet while the file rd_be is used to record the received time of each packet.

6. Using the tool et.exe (from Evalvid) to generate the received video (err.cmp).

```
et.exe sd_be rd_be st foreman_qcif.cmp err_be.cmp 1
```

7. Decode the received video to yuv format. I also put the status of decoding into a file df_be.

```
mpeg4decoder.exe err_be.cmp err_be  176  144  >  df_be
```

8. I think there is something wrong with fixyuv.exe. Therefore I write a new program myfixyuv.exe to fix the decoded yuv sequence.

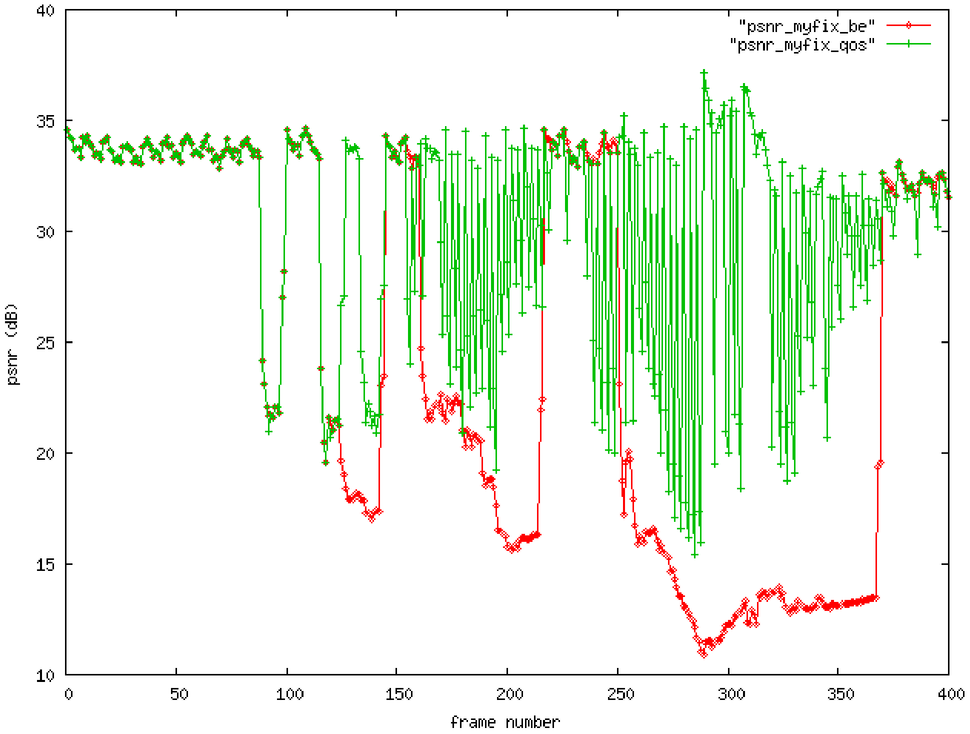
```
myfixyuv.exe  df_be  qcif  400  err_be.yuv  myfix_be.yuv
```

9. Compute the PSNR. Use the psnr.exe (from Evalvid). But I modify the original psnr.c to add frame number id before the psnr of each frame.

```
psnr.exe  176  144  420  foreman_qcif.yuv  myfix_be.yuv > psnr_myfix_be
```

10. Use the same topology above. But I frame packet is pre-marked with lowest drop probability in the application layer at source. P frame packet is pre-marked with middle drop probability and B frame packet is pre-marked with highest drop probability. R1 is implemented WRED. (See qos.tcl)

11. Use the same steps above. The psnr graph is shown below.



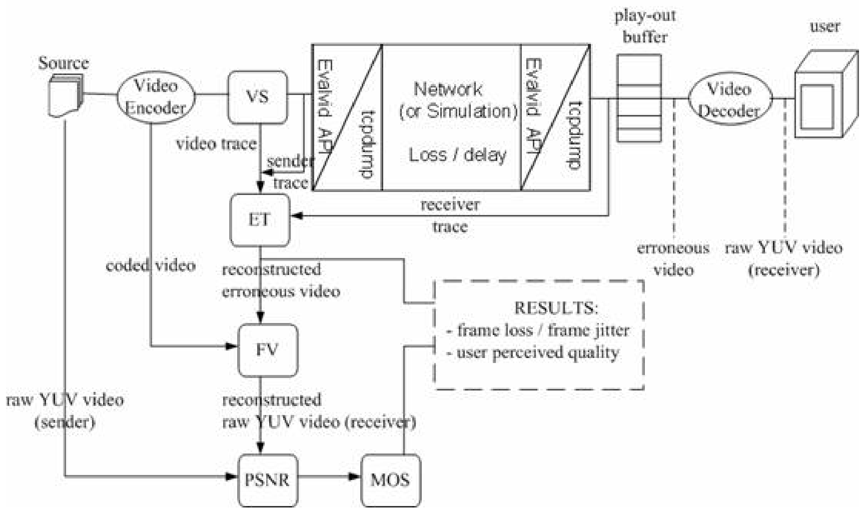
12. One also can use **yuvviewer.exe** to see the video sequences.

BE275	BE276	BE277	BE278	BE279
QoS275	QoS276	QoS277	QoS278	QoS279

- An example for video transmission over wireless network

Overview of EvalVid

The structure of the EvalVid framework is shown as follows.



The main components of the evaluation framework are described as follows:

- **Source**: The video source can be either in the YUV QCIF (176 x 144) or in the YUV CIF (352 x 288) formats.
- **Video Encoder and Video Decoder**: Currently, EvalVid supports two MPEG4 codecs, namely the NCTU codec and ffmpeg. In the present investigation, I arbitrarily choose the NCTU codec for video coding purposes.
- **VS (Video Sender)**: The VS component reads the compressed video file from the output of the video encoder, fragments each large video frame into smaller segments, and then transmits these segments via UDP packets over a real or simulated network. For each transmitted UDP packet, the framework records the timestamp, the packet id, and the packet payload size in the sender trace file with the aid of third-party tools, such as tcp-dump or win-dump, if the network is a real link. Nevertheless, if the network is simulated, the sender trace file is provided by the sender entity of the simulation. The VS component also generates a video trace file that contains information about every frame in the real video file. The video trace file and the sender trace file are later used for subsequent video quality evaluation.
- **ET (Evaluate Trace)**: Once the video transmission is over, the evaluation task begins. The evaluation takes place at the sender side. Therefore, the information about the timestamp, the packet id, and the packet payload size available at the receiver has to be transported back to the sender. Based on the original encoded video file, the video trace file, the sender trace file, and the receiver trace file, the ET component creates a frame/packet loss and frame/packet jitter report and generates a *reconstructed* video file, which corresponds to the possibly corrupted video found at the receiver side as it would be reproduced to an end user. In principle, the generation of the possibly corrupted video can be regarded as a process of copying the original video trace file frame by frame, omitting frames indicated as lost or corrupted at the receiver side. Nevertheless, the generation of the possibly corrupted video is trickier than this and the process is further explained in more details later. Furthermore, the current version of the ET component implements the cumulative inter-frame jitter algorithm for play-out buffer. If a frame arrives later than its defined playback time, the frame is counted as a lost frame. This is an optional function. The size of the play-out buffer must also be set, otherwise it is assumed to be of infinite size.
- **FV (Fix Video)**: Digital video quality assessment is performed frame by frame. Therefore, the total number of video frames at the receiver side, including the erroneous ones, must be the same as that of the original video at the sender side. If the codec cannot handle missing frames, the FV component is used to tackle this problem by inserting the last successfully decoded frame in the place of each lost frame as an error concealment technique.
- **PSNR (Peak Signal Noise Ratio)**: PSNR is one of the most widespread objective metrics to assess the application-level QoS of video transmissions. The following equation shows the definition of the PSNR between the luminance component Y of source image S and destination image D:

$$\mathrm{PSNR}(n)_{\mathrm{dB}} = 20\log_{10}\left(\frac{V_{peak}}{\sqrt{\frac{1}{N_{col}N_{row}}\sum_{i=0}^{N_{col}}\sum_{j=0}^{N_{row}}[Y_S(n,i,j)-Y_D(n,i,j)]^2}}\right),$$

where $V_{peak} = 2^k$-1 and k = number of bits per pixel (lumin tween a reconstructed image and the original one. Prior to transmission, one may then compute a reference PSNR value sequence on the reconstruction of the encoded video as compared to the original raw video. After transmission, the PSNR is computed at the receiver for the reconstructed video of the possibly corrupted video sequence received. The individual PSNR values at the source or receiver do not mean much, but the difference between the quality of the encoded video at the source and the received one can be used as an objective QoS metric to assess the transmission impact on video quality at the application level.

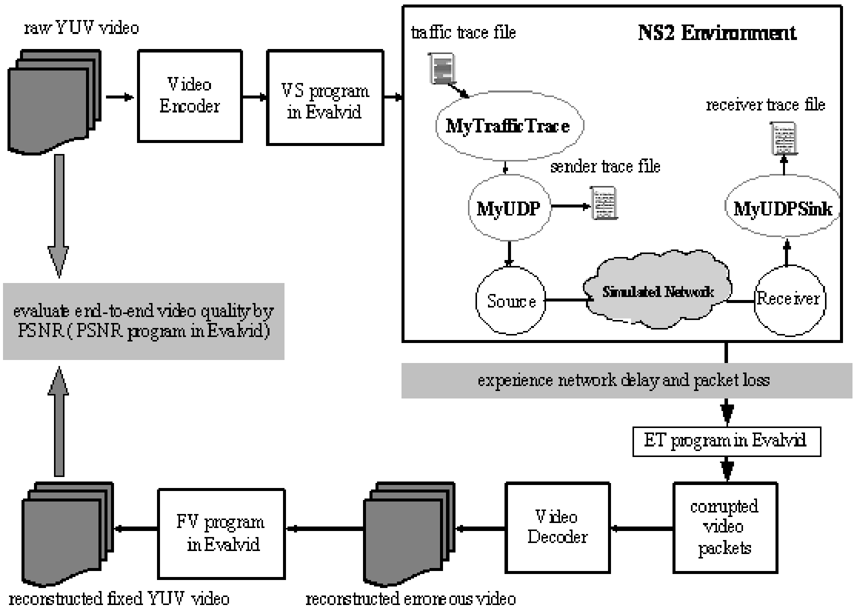
- **MOS (Mean Opinion Score)**: MOS is a subjective metric to measure digital video quality at the application level. This metric of the human quality impression is usually given on a scale that ranges from 1 (worst) to 5 (best). In this framework, the PSNR of every single frame can be approximated_to the MOS scale using the mapping shown as follows.

Possible PSNR to MOS conversion.

PSNR[dB]	MOS
>37	5 (Excellent)
31-37	4 (Good)
25-31	3 (Fair)
20-25	2 (Poor)
<20	1 (Bad)

New network simulation agents

The following figure illustrates the QoS assessment framework for video traffic enabled by the new tool-set that combines EvalVid and NS2. Three connecting simulation agents, namely **MyTrafficTrace, MyUDP,** and **MyUDPSink,** are implemented between NS2 and EvalVid. These interfaces are designed either to read the video trace file or to generate the data required to evaluate the video delivered quality.



MyTrafficTrace

The MyTrafficTrace agent is employed to extract the frame type and the frame size of the video trace file generated from the output of the VS component of EvalVid. Furthermore, this agent fragments the video frames into smaller segments and sends these segments to the lower UDP layer at the appropriate time according to the user settings specified in the simulation script file.

MyUDP

Essentially, MyUDP is an extension of the UDP agent. This new agent allows users to specify the output file name of the sender trace file and it records the timestamp of each transmitted packet, the packet id, and the packet payload size. The task of the MyUDP agent corresponds to the task that tools such as tcp-dump or win-dump do in a real network environment.

MyUDPSink

MyUDPSink is the receiving agent for the fragmented video frame packets sent by MyUDP. This agent also records the timestamp, packet ID, and payload size of each received packet in the user specified file.

Problem of the original YUV program

When the video transmission is over, the receiver trace file has to be sent back to the sender side for the video quality evaluation. Based on the video trace file, the sender trace file, and the receiver trace file, the lost frames can be indicated. If a frame is lost due to packet loss, the ET component sets the vop_coded bit of this video object plane (VOP) header in the original compressed video file to 0. The setting of this bit to 0 indicates that no subsequent data exists for this VOP. This type of frame is referred to as a vop-not-coded frame. While a frame is completely received and the vop_coded bit is set to 1, this type of frame is referred to as a decodable frame. After setting the vop_coded bit to 0 for all the lost frames, the processed file is then used to represent the compressed video file received by the receiver side.

Currently, no standard exists to define an appropriate treatment of vop-not-coded frames. Some decoders with an error concealment mechanism simply replace the vop-not-coded frames by the last successfully decoded frame. In these cases, the FV component is not required. Other decoders, however, without error concealment, such as ffmpeg, decode all frames other than the vop-not-coded frames. In these cases, the FV component can handle these vop-not-coded frames without difficulty by simply replacing them with the last successfully decoded frames. Other decoders, such as Xvid or the NCTU codec, additionally fail to decode the subsequent frames in some cases. For example, when decoding a subsequent frame that is a decodable frame, this frame may fail to be decoded if the frame it depends on is a vop-not-coded frame because there is no enough information to decode it. This type of frame is referred to as a undecodable frame. In this case, the original FV component fails since it does not take this possibility into consideration.

Based on these considerations, a requirement exists to design a new algorithm capable of solving the problem of undecodable frames. In this study, we develop an algorithm that uses the decoder output to fix the decoding results, *i.e.* reconstructed erroneous video sequence. If a frame is decodable, the improved FV component copies this decoded YUV frame data from the reconstructed erroneous raw video file into a temporary file and keeps it in a buffer as the last successfully decoded frame data. If a frame is vop-not-coded, the improved FV component reads this frame data from the reconstructed erroneous raw video file, but it does not copy the data into the temporary file. This is because the data read is useless and the file pointer needs to be moved to the next frame. The improved FV component copies the data from the buffer into the temporary file instead. If a frame is missing or considered undecodable, the improved FV component simply copies the last successfully decoded YUV frame data in the buffer into the temporary file. After processing all the frames in the reconstructed and possibly corrupted video sequence, the resulting temporary file is the reconstructed fixed video sequence. Afterwards, the frame-by-frame PSNR can be evaluated in the usual manner.

[**The projects or papers based on this toolset**]

- **Chih-Heng Ke**, Ce-Kuen Shieh, Wen-Shyang Hwang, Artur Ziviani, "A Two Markers System for Improved MPEG Video Delivery in a DiffServ Network", IEEE Communications Letters, IEEE Press, ISSN: 1089-7798, vol. 9, no. 4, pp. 381-383, April 2005
- J. Naoum-Sawaya, B. Ghaddar, S. Khawam, H. Safa, H. Artail, and Z. Dawy, "Adaptive Approach for QoS Support in IEEE 802.11e Wireless LAN," in IEEE International Conference on Wireless and Mobile Computing , Networking and Communications (WiMob 2005), Montreal, Canada, August 2005
- H. Huang, J. Ou, and D. Zhang, **"**Efficient Multimedia Transmission in Mobile Network by using PR-SCTP", Communications and Computer Networks (CCN 2005)**,** 10/24/2005 - 10/26/2005**,** Marina del Rey, USA
- A. Lo, G. Heijenk, I. Niemegeers, "Performance Evaluation of MPEG-4 Video Streaming over UMTS Networks using an Integrated Tool Environment", Proceedings SPECTS 2005, 2005 International Symposium on Performance Evaluation of Computer and Telecommunication Systems, Philadelphia, PA, USA, July 24-28, 2005.
- Vasos Vassiliou, Pavlos Antoniou, Iraklis Giannakou, and Andreas Pitsillides 'Requirements for the Transmission of Streaming Video in Mobile Wireless Networks", International Conference on Artificial Neural Networks (ICANN), Athens, Greece, September 10-14, 2006
- The statistics for people wrote me emails about this toolset

Country / Company	Research Group & Research Interests
America	1. Stanford CS grad student 2. PhD candidate at Univ. of Houston
Australia	1. Lecture at La Trobe University
Canada	1. Carleton University: MPEG4 transmission evaluation over Resilient Packet Ring Networks
China	
Egypt	1. search assistant at Computer division Faculty of Engineering, Suez Canal University
France	1. Doctorant student at Paris 13 university
Germany	1. TKN
Iceland	1. master student at University of Iceland: Modelling the Icelandic Health Network
India	1. Asst.Prof. at Engg. College:Wireless MANET and multimedia transmission 2. PG student: Accelerating peer to peer video streaming on Multipoint to point communication
Indonesia	1. collage students at STTTelkom Bandung: video streaming over umts network
Israel	1. Ph.D : sumulation of the behaviour of the traffic manager in the MPLS-diffserv environment.
Japan	1. PhD student: real-time P2P video streaming
Korea	1. student at Kunsan National University: streaming video over the Internet
Lebanon	1. using Evalvid to investigate the performance of 802.11 and 802.11e
Mexico	1. IATM
Norway	1. PhD student at NTNU:
Singapore	1. PhD student at National University of Singapore 2. student at Nanyang Technological University: video transmission over MANETs
Sweden	1. PhD student at KTH: cross-layer issues in wireless multimedia
Switzerland	1. PhD student at EPFL
Taiwan	1. Master students at National Yulin of University of Science & Technology 2. Master student at National Taiwan University 3. Master student at National Chung Hsing University
Tunisia	1. researcher in the tunisian supcom university: I have optimized QoS on ad hoc network and implement it in ns-2 and i want to experiment it with MPEG video and other multimedia applications
U.K.	1. PhD student at University of London 2. master student: in the area of QoS 3. postgraduate student from University of Hertfordshire: video evaluation over Mobile Ad hoc Network 4. B(Eng) Computer Systems student at Brunel University: video in ad hoc network 5. Research student at Sheffield Hallam University
Company	1. Ericsson: test the wireless channel and the channels error-probability that at what extent it affects a video transmission. 2. Motorola: 3. Nokia: video transmission project (in China)
Unknown	No information left in the email

[**Note**]
1. 2005/10/28 Add "method 2" to install the module into NS2
2. 2005/10/28 Add the projects based on this toolset
3. 2005/10/29 If someone wants to use this toolset over 802.11e, please contact me. Some C++ codes need to be modified.
4. 2006/02/03 Add my paper (SUTC2006) for reference.
5. 2006/05/31 Thanks for **Julien Chaumond's** notification.

When you run the qos.tcl, please add the following line into the tcl file.

```
$qr1r2 meanPktSize 1000
```

Last modified: 2006/05/31

Author : Chih-Heng, Ke
Website: http://140.116.164.80/~smallko
Email: smallko@ee.ncku.edu.tw
Phd candidate, EE Department, NCKU, Taiwan