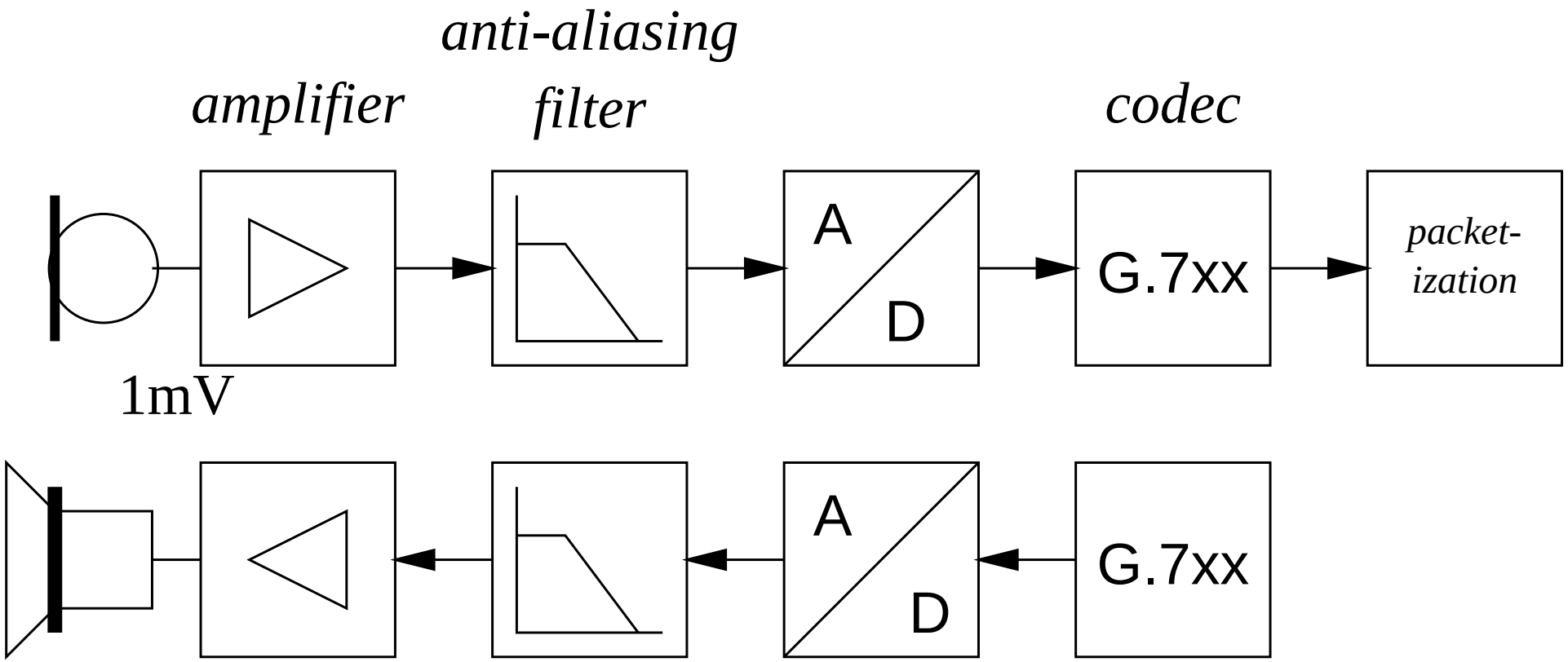
Audio and Speech

Digital sound



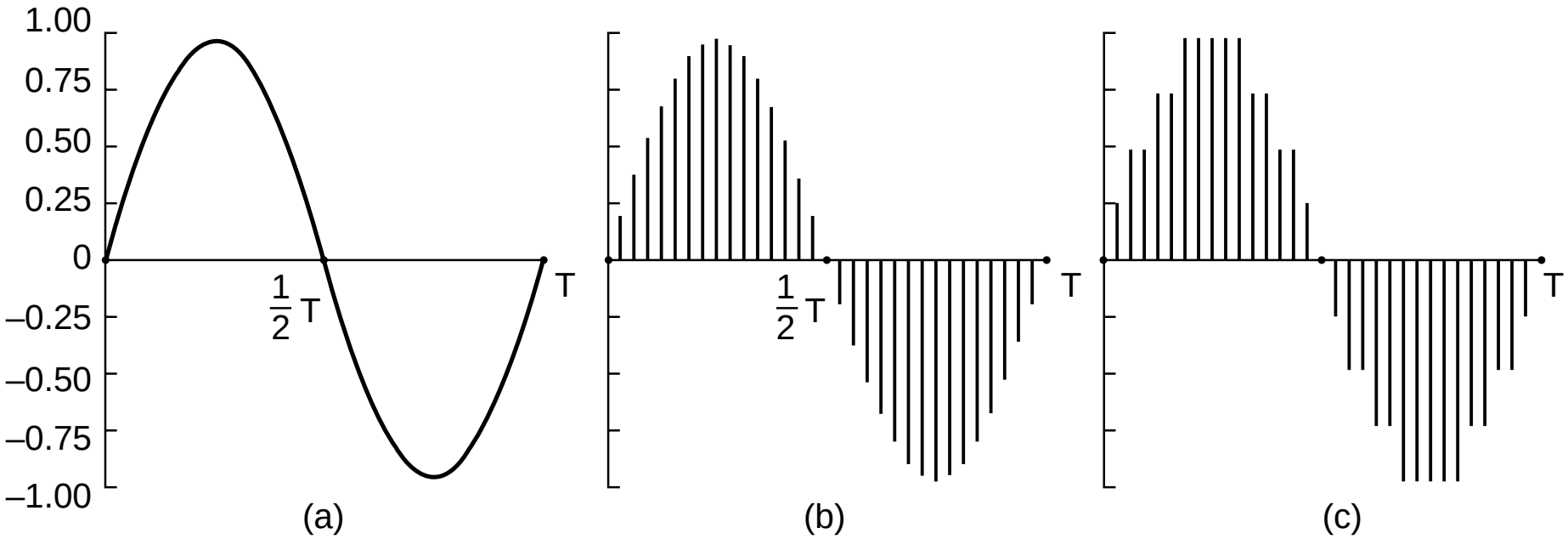
Digital audio

- sample each audio channel and quantize ➠ pulse-code modulation (PCM)
- Nyquist bound: need to sample at twice (+ ϵ) the maximum signal frequency
- analog telephony: 300 Hz – 3400 Hz ➠ 8 kHz sampling $\longrightarrow$ 8 bits/sample, 64 kb/s
- FM radio: 15 kHz
- audio CD: 44,100 Hz sampling, 16 bits/sample (based on video equipment used for early recordings)
- more bits ➠ more dynamic range, lower distortion
- audio highly redundant ➠ compression
- almost all codecs fixed rate

Audio coding

application	frequency	sampling	AD/DA bits	application
telephone	300-3400 Hz	8 kHz	12–13	PSTN
wide band	50-7000 Hz	16 kHz	14–15	conferencing
high-quality	30-15000 Hz	32 kHz	16	FM, TV
	20-20000 Hz	44.1 kHz	16	CD
	10-22000 Hz	48 kHz	≤ 24	pro-audio

Digital audio: sampling



distortion: signal-to-(quantization) noise ratio

Digital audio: compression

Alternatives for compression:

- companding: non-linear quantization ➠ μ-law (G.711)
- waveform: exploit statistical correlation between samples
- model: model voice, extract parameters (e.g., pitch)
- subband: split signal into bands (e.g., 32) and code individually ➠ MPEG audio coding

Newer codings: make use of *masking properties* of human ear

Judging a codec

- bitrate
- quality
- delay: algorithmic delay, processing
- robustness to loss
- complexity: MIPS, floating vs. fixed point, encode vs. decode
- tandem performance
- can the codec be *embedded*?
- non-speech performance: music, voiceband data, fax, tones, . . .

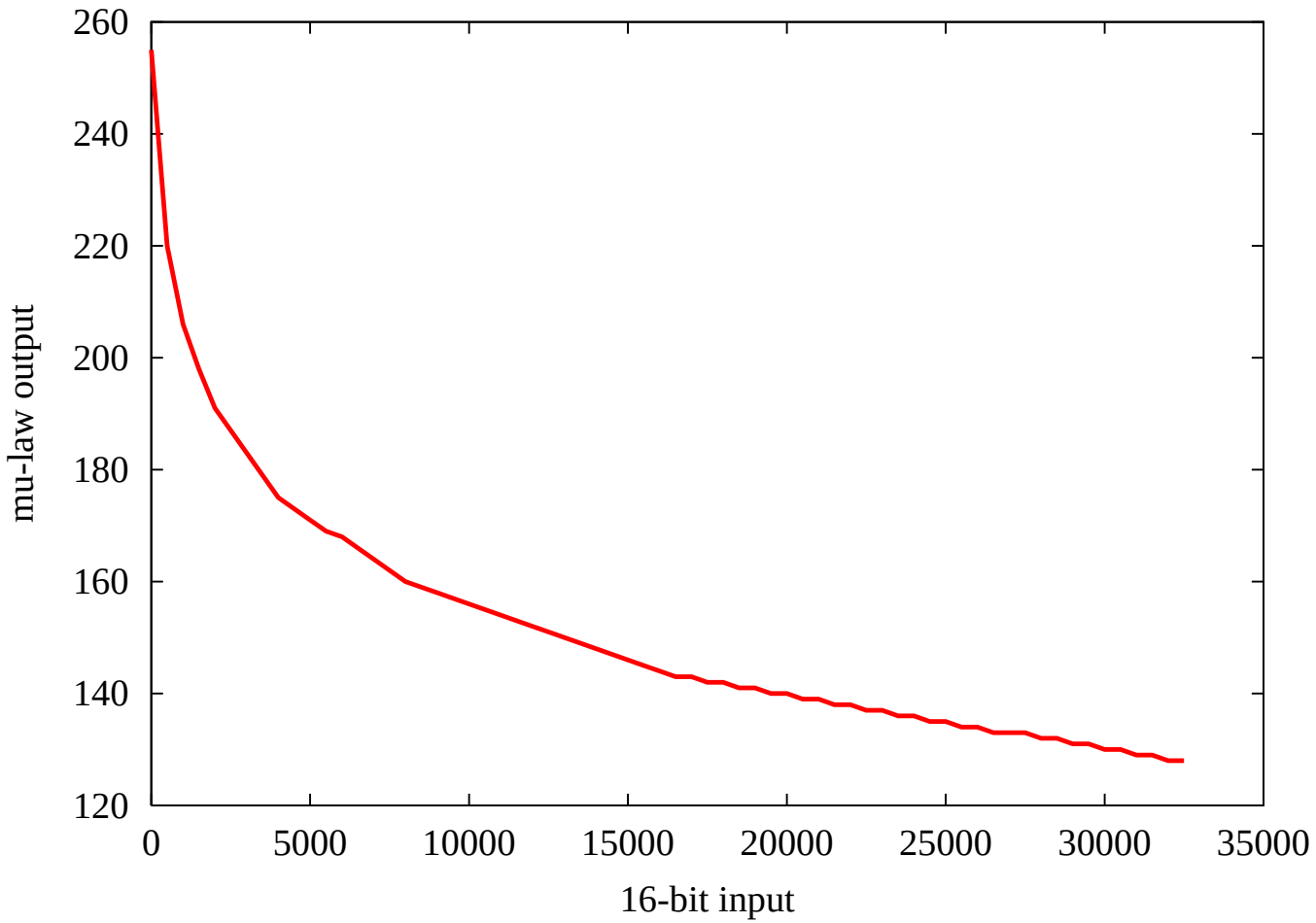
Quality metrics

- speech vs. music
- communications vs. *toll quality*
- mean opinion score (MOS) and degradation MOS

score	MOS	DMOS	
5	excellent	inaudible	no effort required
4	good, toll quality	audible, but not annoying	no appreciable effort
3	fair	slightly annoying	moderate effort
2	poor	annoying	considerable effort
1	bad	very annoying	no meaning

- diagnostic rhyme test (DRT) for low-rate codecs (96 pairs like “dune” vs. “tune”)
  – 90% = toll quality

Companding: μ-law for G.711 ("PCMU")



Also: A-law in Europe

Silence detection (VAD)

- avoid transmitting silence during sentence pauses and/or other person talking
- detect silence based on energy, sound
- hangover – unvoiced segments at end of words
- conferencing!
- comfort noise – white noise, shaped noise with periodic updates
- transmit update (4 byte) when things change

Audio silence detection

- needed in conferences to avoid drowning in fan noise
- also reduces data rate
- in use in transoceanic telephony since 1950's (TASI: time-assigned speech interpolation)
- use energy estimate (μ-law already close) or spectral properties (difficult)
- difficulty: background noise, levels vary
- ➠ vary noise threshold: threshold = running average + hysteresis
- if above threshold, increase running average by one for each block
- if below threshold, update running average
- speech has soft (unvoiced) beginnings and endings ➠ *hang-over*, pre-talkspurt burst

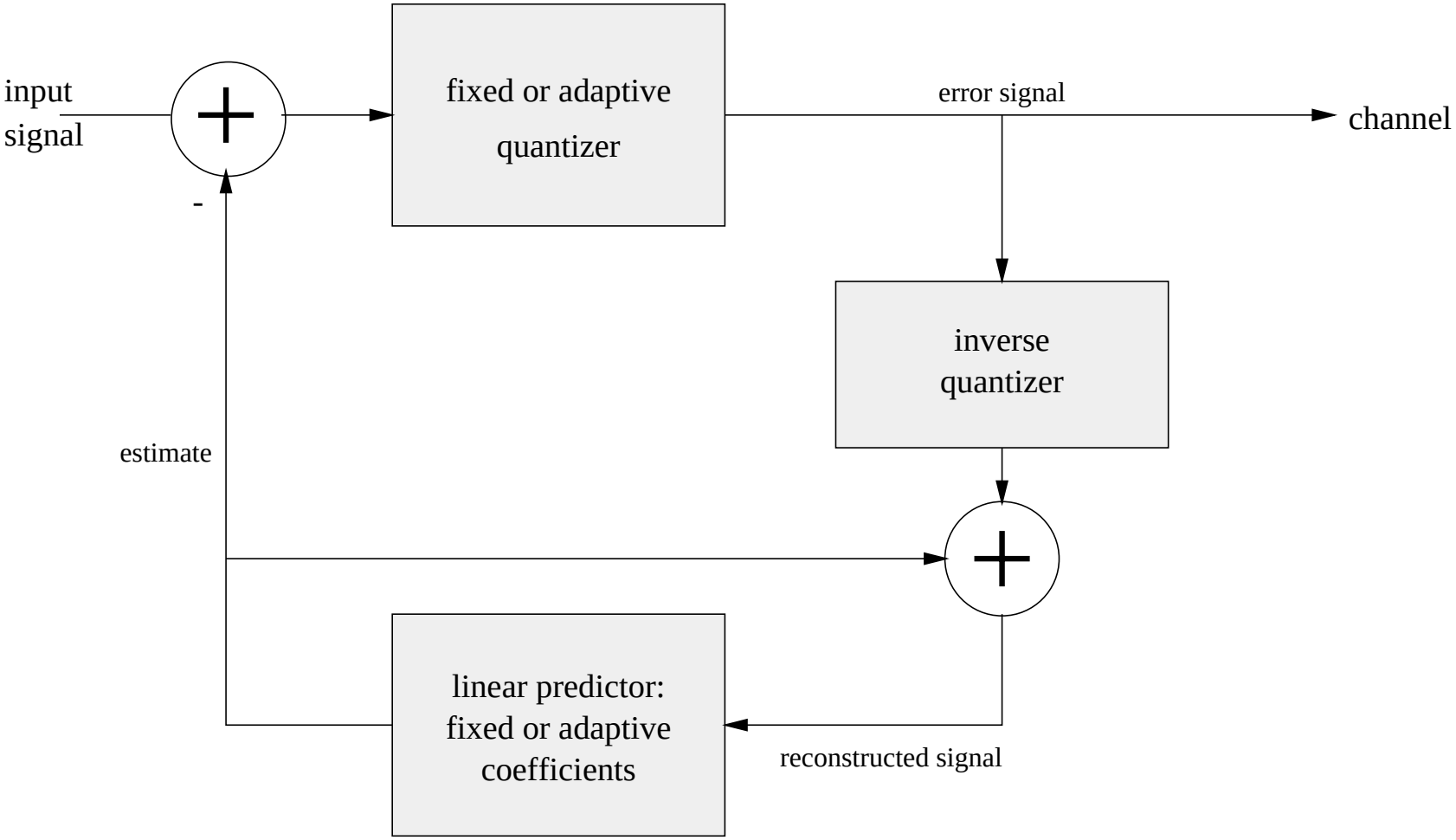
Speech codecs

- *waveform codecs exploit sample correlation: 24-32 kb/s*
- *linear predictive (vocoder) on* frames *of 10–30 ms (stationary): remove correlation* $\longrightarrow$ *error is white noise*
- *vector quantization*
- *hybrid, analysis-by-synthesis*
- *entropy coding: frequent values have shorter codes*
- *runlength coding*

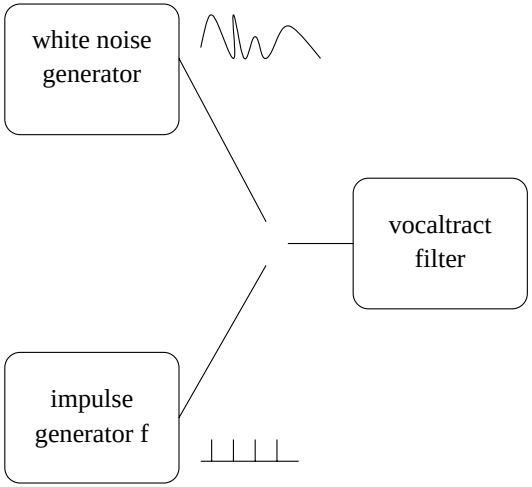
Digital audio: compression

coding	kb/s	MOS	use
LPC-10	2.4	2.3	robotic, secure telephone
G.723.1	5.3/6.3	3.8	videotelephony (room for video)
GSM HR	5.6	3.5	GSM 2.5G networks
IS 641	7.4	4.0	TDMA (N. America) mobile (new)
IS 54/136	7.95	3.5	TDMA (N. America) mobile (old)
G.729	8.0	4.0	mobile telephony
GSM EFR	12.2	4.0	GSM 2.5G
GSM	13.0	3.5	European mobile phone
G.728	16.0	4.0	low-delay
G.726	16-40		low-complexity (ADPCM)
G.726	32	4.1	low-complexity (ADPCM)
DVI	32.0		toll-quality (Intel, Microsoft)
G.722	64.0		7 kHz codec (subband)
G.711	64.0	4.5	telephone (μ-law, A-law)
MPEG L3	56-128.0	N/A	CD stereo
16 bit/44.1 kHz	1411	compact disc	

(Adaptive) Differential Pulse Code Modulation



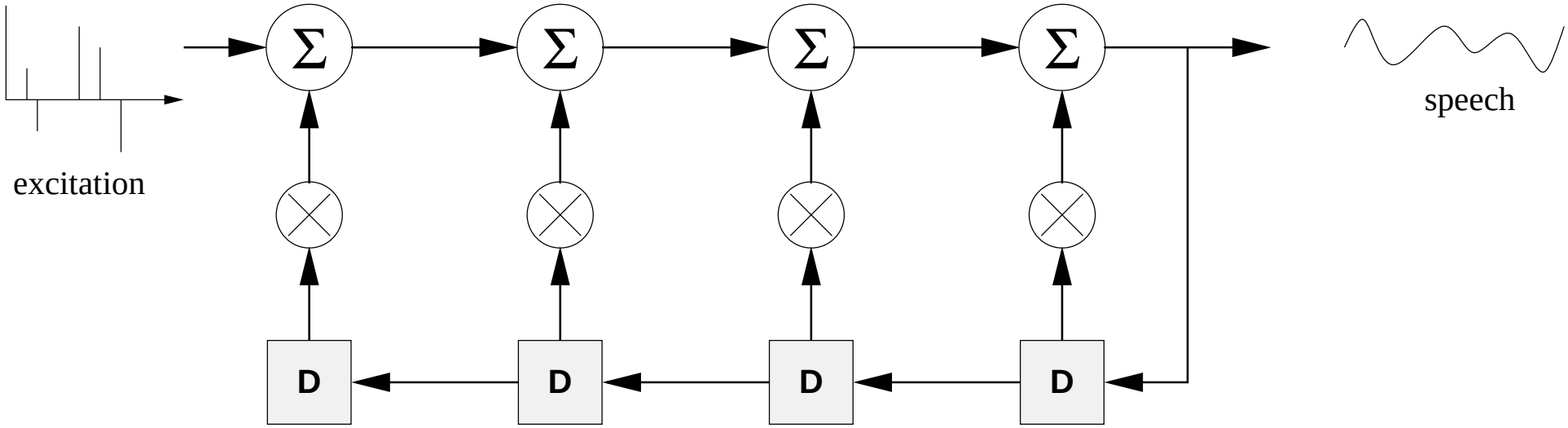
Linear predictive codec



E.g., LPC 10 at 2.4 kb/s:

Sampling rate	8 kHz
Frame length	180 samples = 22.5 ms
Linear predictive filter	10 coefficients = 42 bits
Pitch and voicing	7 bits
Gain information	5 bits

Linear predictive codec



linear (IIR) filter

infinite impulse response (IIR) vs. finite impulse response

G.723 audio codec

- analysis-by-synthesis codec
- 5.3 or 6.3 kb/s bit rate
- 30 ms frames with 7.5 ms look-ahead
- MOS of 3.98
- requires 40% of 100 MHz Pentium or 22 DSP MIPS, 16 kB code
- popular for videoconferencing with modems

G.729 audio codec

- 8 kb/s bit rate
- 10 ms frames with 5 ms look-ahead
- LD-CELP (low-delay code-excited linear prediction): filter coefficients, code book for excitations
- requires 25% of 100 MHz Pentium
- MOS of 3.7/3.2/2.8 with 0/3/5% packet loss
- deals with *frame erasure*
- Annex A: low-complexity; Annex B: VAD; Annex D: 6.4 kb/s; Annex E: 11.8 kb/s

Audio perceptual encoding

- ear = 24-26 overlapping bandpass filters (100 Hz to 5,000 Hz)
- maximum sensitivity between 1,000 and 5,000 Hz
- masked by stronger signal in close frequency proximity
- ISO MPEG-1 Layer I, II, III
- analysis filter bank
- AAC: 64 kb/s for mono for almost CD-quality

Distortion: signal-to-noise ratio

- error (noise) $r(n) = x(n) - y(n)$
- variances $\sigma_x^2, \sigma_y^2, \sigma_r^2$
- power for signal with pdf $p(x)$ and range $-V \ldots + V$:

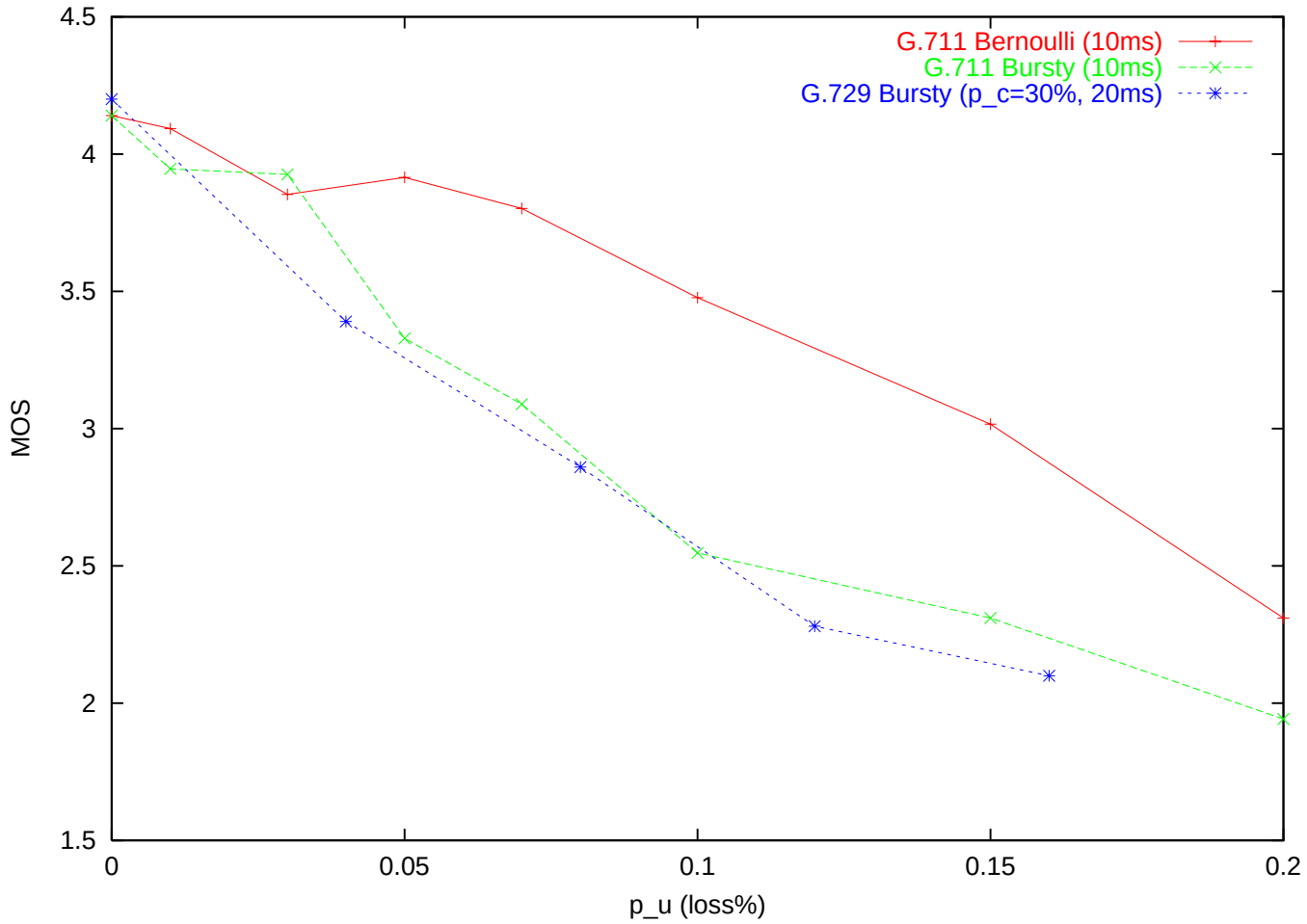
$$\sigma_x^2 = \int_{-V}^{+V} (x - \bar{x})^2 p(x) dx$$

- $\sigma_u^2 = \frac{1}{M} \sum_{n=1}^{M} u^2(n)$
- SNR = $6.02N - 1.73$ for uniform quantizer with N bits

Distortion measures

- SNR *not* a good measure of perceptual quality
- ➠ segmental SNR: time-averaged blocks (say, 16 ms)
- frequency weighting
- subjective measures:
  - A-B preference
  - subjective SNR: comparison with additive noise
  - MOS (mean opinion score of 1-5), DRT, DAM, . . .

MOS vs. packet loss



Objective speech quality measurements

- approximate human perception of noise and other distortions
- distortion due to encoding and packet loss (gaps, interpolation of decoder)
- examples: PSQM (P.861), PESQ (P.862), MNB, EMBSD – compare reference signal to distorted signal
- either generate MOS scores or distance metrics
- much cheaper than subjective tests
- only for telephone-quality audio so far

Objectice quality measures

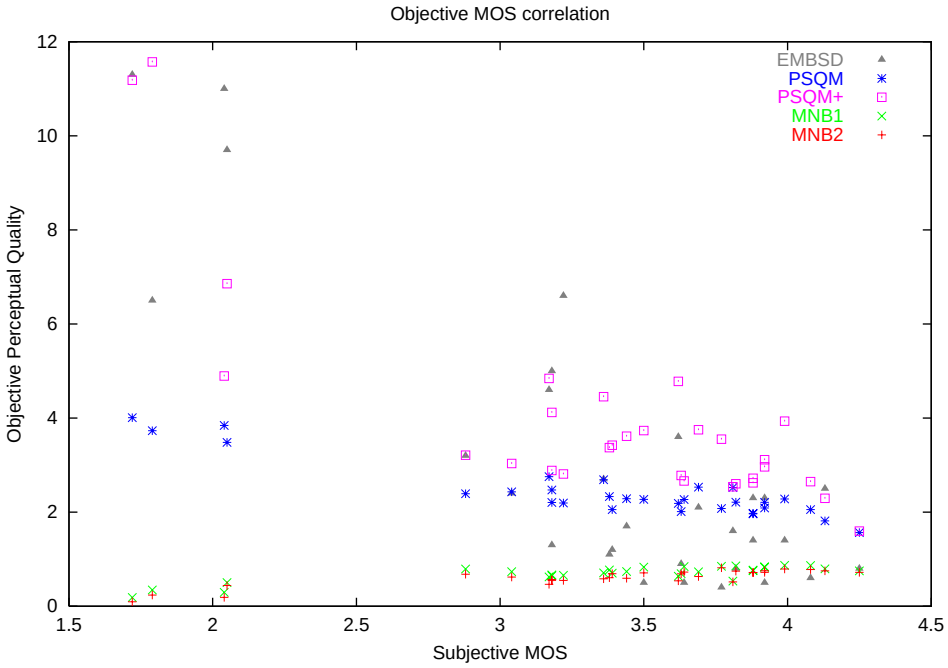
PSQM: perceptual distance; can't handle delay offset

PESQ: MOS scores; automatically detects and compensates for time-varying delay offsets between reference and degraded signal

- time-frequency mapping (FFT)
- frequency warping from Hertz scale to critical band domain (Bark spectrum)
- calculate noise disturbance as the difference of compressed loudness (Sone) intensity in each band between the two signals, with threshold masking
- asymmetry modeling (addition of an unrelated frequency component is worse than omission of a component of the reference signal)

Objective vs. Subjective MOS

Objective MOS tools don’t always handle loss impairments correctly:



Audio traffic models

talkspurt: constant bit rate: one packet every 20...100 ms ➠ mean: 1.67 s

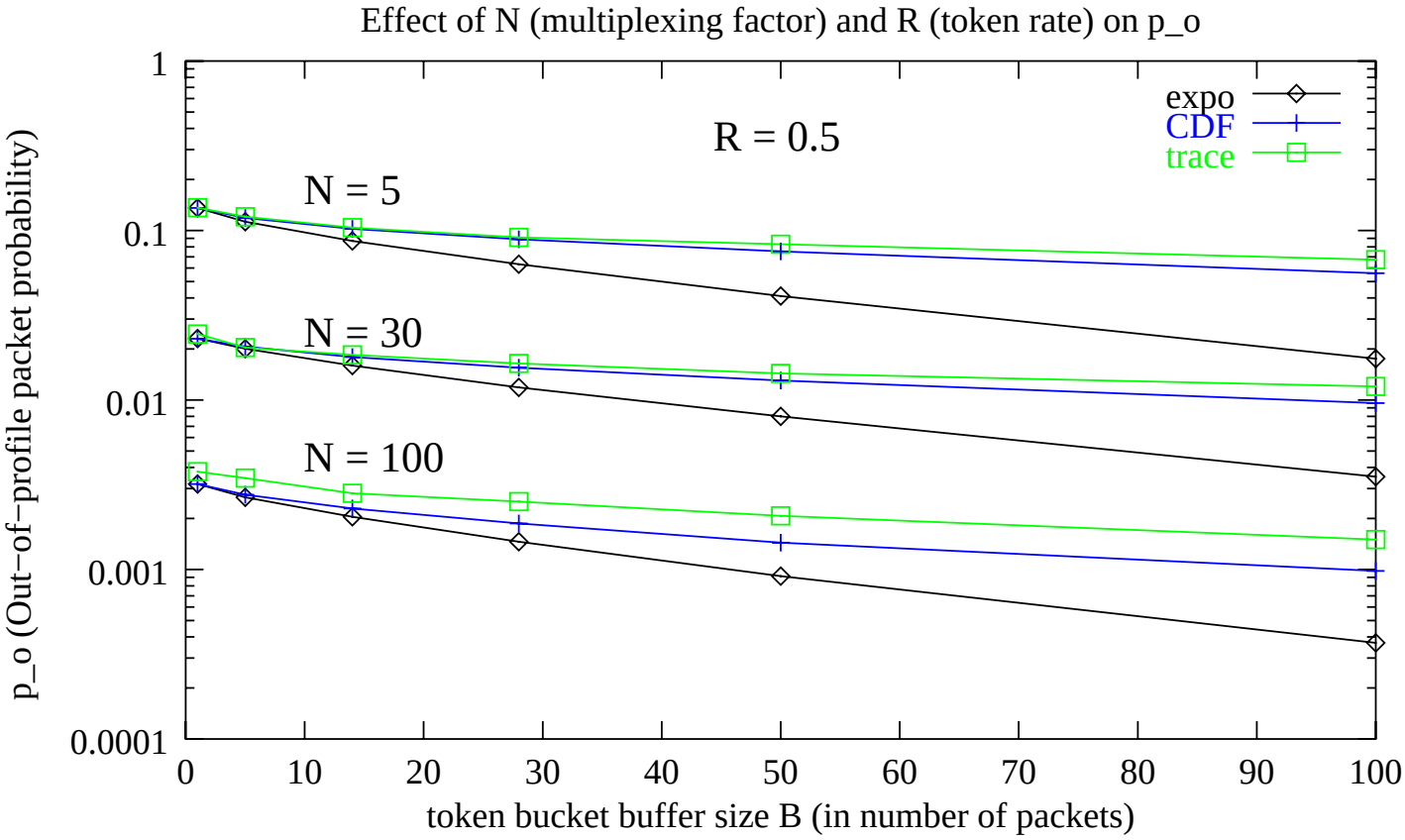
silence period: usually none (maybe transmit background noise value) ➠ 1.34 s

➠ for telephone conversation, both roughly exponentially distributed

- double talk for “hand-off”
- may vary between conversations... ➠ only in aggregate

Multiplexing traffic

In a diff-serv buffer, with $R = 0.5 =$ reserved/peak:



G.729B: about 42-43% silence

Audio on Solaris

- `cat sample.au > /dev/audio` works; see samples in `/usr/demo/SOUND/sounds/`
- `audiocontrol` to change ports and volume
- `audiotool` for recording and playback
- `audioplay` for playing back sound files

Audio on Solaris

```
#include <sys/audioio.h>
ac = open("/dev/audioctl", O_RDWR);
au = open("/dev/audio",    O_RDWR);
AUDIO_INITINFO(&ai);
ai.record.port = AUDIO_MICROPHONE;
ai.play.port   = AUDIO_SPEAKER;
ioctl(ac, AUDIO_SETINFO, &ai);

bytes = read(au, buffer, bytes);

write(au, buffer, bytes);
```

Careful with opening audio input - keep bit bucket handy!

Audio on Solaris

- read() blocks until audio read
- set device to non-blocking if only currently available audio needed
- write() returns when copied to device, but playout may last longer
- typical loop for world's most expensive microphone amplifier:

```
while (1) {
  b = read(au, buffer, bytes);
  /* audio processing */
  write(au, buffer, b);
}
```

- `ioctl(au, AUDIO_DRAIN, 0);` blocks until audio played out
- use select() to handle both network and audio input

Event-based programs

- `read()` is blocking ➠ server only works with single socket ↔ audio, network input
- need I/O multiplexing ➠ event-based programming
- also need to handle time-outs, connection requests
- all events (mouse clicks, windows, etc.) handled by event loop
- **do**
  wait for event(s)
  handle event (hopefully short)
  forever
- harder to maintain state, recursion
- alternative 1: “interrupts” (signals) ➠ called at any time
- alternaitve 2: threads (separate scheduling, same address space)

Multiplexing with `select()`

```
int select(int nfds, fd_set *readfds, fd_set *writefds,
  fd_set *exceptfds, struct timeval *timeout)
```

- block until $>=$ 1 file descriptors have something to be read, written, or an exception, or timeout
- set bit mask for descriptors to watch using `FD_SET`
- returns with bits for ready descriptors set ➠ check with `FD_ISSET`
- cannot specify amount of data ready

Audio timing

Need to write block of audio to speaker every t ms ($t = 20 \ldots 100$ ms) ➠

1. timer ➠
   - OS overhead
   - may not be accurate
   - error accumulation (time between timers)
   - clock may differ from audio sampling clock
2. use audio input: for every block read, write one audio block ➠ stay in sync
3. but: doesn't work for half-duplex audio cards

Audio on Linux

- `/dev/audio` for μ-law device
- `/dev/dsp` for general samples
- `aumix` allows to configure mixer
- use `fuser -v /dev/dsp` to find out who's using the device
- watch for endianness - Linux supports both LE and BE
- guide at http://www.4front-tech.com/pguide/audio.html

Audio on Linux: example

```
#include <ioctl.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/soundcard.h>
#define BUF_SIZE    4096
int format = AFMT_S16_LE, stereo = 1, speed = 11025;
if ((audio_fd = open("/dev/dsp", open_mode, 0)) == -1) {
  /* error */
}
if (ioctl(audio_fd, SNDCTL_DSP_SETFMT, &format) == -1) {
}
if (ioctl(audio_fd, SNDCTL_DSP_STEREO, &stereo) == -1 {
}
if (ioctl(audio_fd, SNDCTL_DSP_SPEED, &speed) == -1) {
}
/* set to full duplex */
if (ioctl(audio_fd, SNDCTL_DSP_SETDUPLEX, 0) == -1) {
}
```

Java sound interface

- Java Sound API (java.sun.com/products/java-media/sound/)
- higher layer: Java Media Framework (JMF), includes RTP
- javax.sound.sampled.spi as *service provider*

Java sound API: example

```
TargetDataLine line;
DataLine.Info info = new DataLine.Info(TargetDataLine.class,
  format); // format is an AudioFormat object
byte[] data = new byte[line.getBufferSize()/5];

if (!AudioSystem.isLineSupported(info)) {
  // Handle the error ...
}
// Obtain and open the line.
try {
  line = (TargetDataLine) AudioSystem.getLine(info);
  line.open(format, bufferSize);
} catch (LineUnavailableException ex) {
  // Handle the error ...
}
// Begin audio capture.
line.start();
numBytesRead = line.read(data, offset, data.length);
```

Audio mixing

- for conferences: several concurrent talkers (+ open mikes) – at least temporarily
- adding two voices $\neq$ twice as loud (+3 dB)
- mixing audio ➠ adding linear samples
- for waveform-encoded (e.g., μ-law) samples: use lookup table

References

- J. Bellamy, *Digital Telephony*, 2nd ed., Wiley, 1991.
- N. S. Jayant and P. Noll, *Digital Coding of Waveforms*, Prentice Hall.
- R. Steinmetz and K. Nahrstedt, *Multimedia: Computing, Communications and Applications*. Upper Saddle River, New Jersey: Prentice-Hall, 1995.
- O. Hersent, D. Gurle and J.P. Petit, *IP Telephony*, Addison-Wesley, 2000.
- L.R. Rabiner and R.W. Schafer, *Digital Processing of Speech Signals*, Prentice-Hall, 1978.

See also http://www.cs.columbia.edu/~hgs/audio