

# How valuable is noisy or limited feedback?

Anant Sahai

based in part on joint work with:

Stark Draper    Tunc Simsek

Wireless Foundations

Department of Electrical Engineering and Computer Sciences

University of California at Berkeley

**Major Support from NSF ITR**

March 29th 2008

# Information theory tells us:

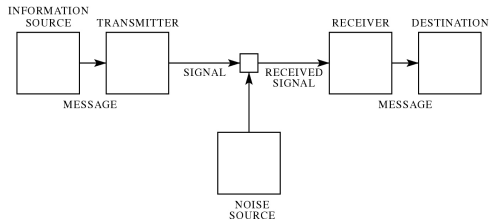


Fig. 1—Schematic diagram of a general communication system.

- Communication is idealized as a *one-way* flow of information.

# Information theory tells us:

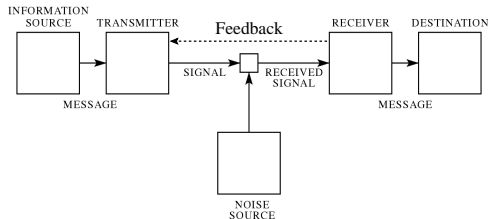


Fig. 1—Schematic diagram of a general communication system.

- Communication is idealized as a *one-way* flow of information.
- But the medium usually supports a path in reverse.
- How should it be used?

# Information theory tells us:

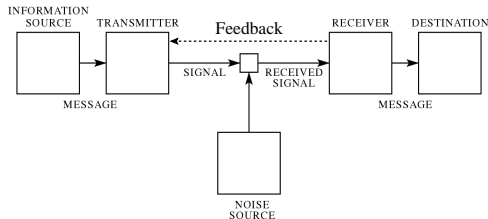


Fig. 1—Schematic diagram of a general communication system.

- Communication is idealized as a *one-way* flow of information.
- But the medium usually supports a path in reverse.
- How should it be used?
  - ▶ Left alone for other users.

# Information theory tells us:

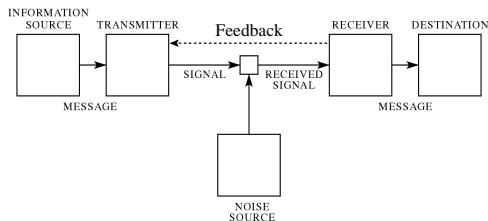


Fig. 1—Schematic diagram of a general communication system.

- Communication is idealized as a *one-way* flow of information.
- But the medium usually supports a path in reverse.
- How should it be used?
  - ▶ Left alone for other users.
  - ▶ For learning/adaptation/universality.

# Information theory tells us:

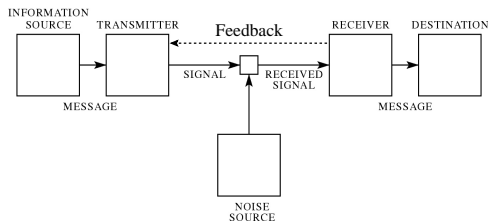


Fig. 1—Schematic diagram of a general communication system.

- Communication is idealized as a *one-way* flow of information.
- But the medium usually supports a path in reverse.
- How should it be used?
  - ▶ Left alone for other users.
  - ▶ For learning/adaptation/universality.
  - ▶ Simplify implementation.

# Information theory tells us:

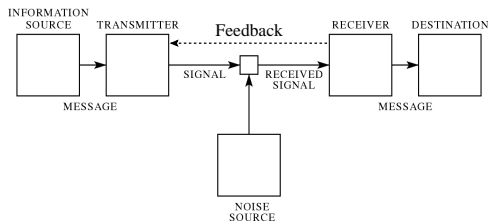


Fig. 1—Schematic diagram of a general communication system.

- Communication is idealized as a *one-way* flow of information.
- But the medium usually supports a path in reverse.
- How should it be used?
  - ▶ Left alone for other users.
  - ▶ For learning/adaptation/universality.
  - ▶ Simplify implementation.
  - ▶ **Improve QoS: delay,  $P_e$ , rate.**

## Flashback and Warning: 1973 (Bob Lucky)

*“Feedback communications was an area of intense activity in 1968... A number of authors had shown constructive, even simple, schemes using noiseless feedback to achieve Shannon-like behavior... The situation in 1973 is dramatically different... **The subject itself seems to be a burned out case...***

*In extending the simple noiseless feedback model to allow for more realistic situations, such as noisy feedback channels, bandlimited channels, and peak power constraints, theorists discovered a certain **“brittleness”** or sensitivity in their previous results... ”*

## Flashback and Warning: 1973 (Bob Lucky)

*“Feedback communications was an area of intense activity in 1968... A number of authors had shown constructive, even simple, schemes using noiseless feedback to achieve Shannon-like behavior... The situation in 1973 is dramatically different... **The subject itself seems to be a burned out case...***

*In extending the simple noiseless feedback model to allow for more realistic situations, such as noisy feedback channels, bandlimited channels, and peak power constraints, theorists discovered a certain **“brittleness”** or sensitivity in their previous results... ”*

**The goal: show **“robustness”** of gains due to feedback.**

# Shannon's Prophecy:

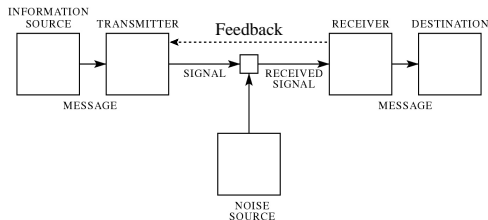
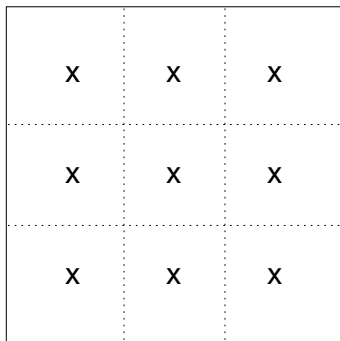


Fig. 1—Schematic diagram of a general communication system.

- Delay is the most basic price of reliability

*“[The duality between source and channel coding] can be pursued further and is related to a duality between past and future and the notions of control and knowledge. Thus we may have knowledge of the past and cannot control it; we may control the future but have no knowledge of it.” — Claude Shannon '59*

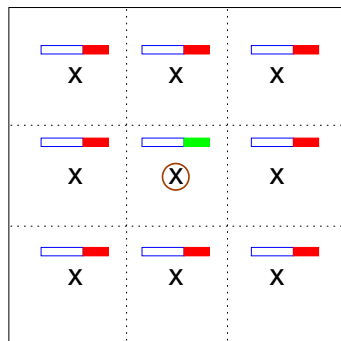
## Review: Fixed blocks



- Hard decision regions cover space

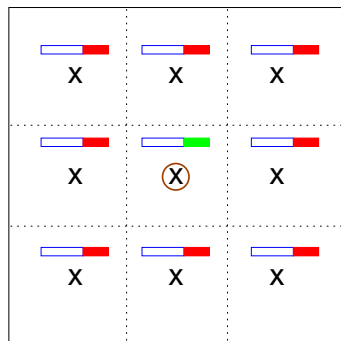
- Feedback is pointless at high rates.
- Dobrushin showed this if channel is symmetric.
- Conjectured for all channels.

## Review: Fixed blocks, Soft deadlines



- Hard decisions regions but check hash signatures

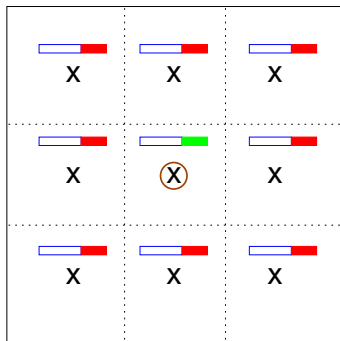
## Review: Fixed blocks, Soft deadlines



- 1 bit feedback can request retransmissions
- Can interpret as expected block-length

- Hard decisions regions but check hash signatures

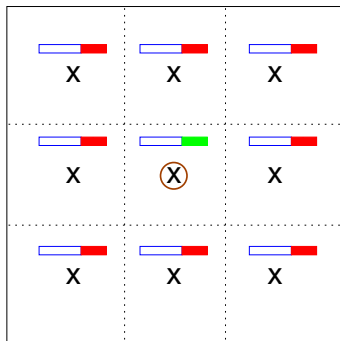
## Review: Fixed blocks, Soft deadlines



- 1 bit feedback can request retransmissions
- Can interpret as expected block-length
- Run close to capacity
- Use  $\approx n(C - R)$  bits for signatures

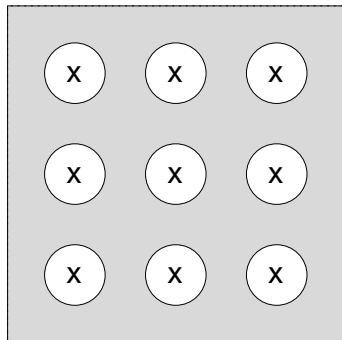
- Hard decisions regions but check hash signatures

## Review: Fixed blocks, Soft deadlines



- 1 bit feedback can request retransmissions
  - Can interpret as expected block-length
  - Run close to capacity
  - Use  $\approx n(C - R)$  bits for signatures
  - Linear slope  $-1$  for error exponent
- 
- Hard decisions regions but check hash signatures

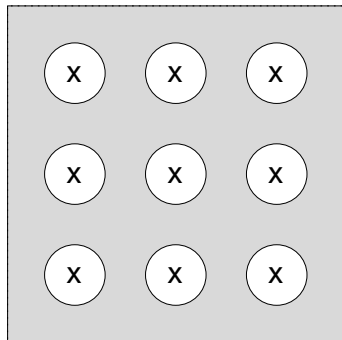
## Review: Fixed blocks, Soft deadlines: Forney-68



- 1 bit feedback can request retransmissions
- Can interpret as expected block-length

- Refuse to decide when ambiguous

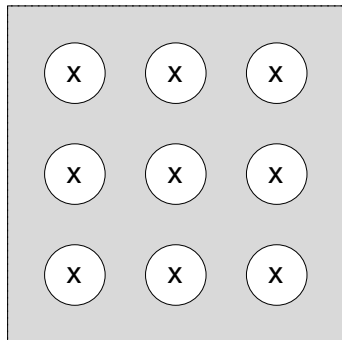
## Review: Fixed blocks, Soft deadlines: Forney-68



- 1 bit feedback can request retransmissions
- Can interpret as expected block-length
- Decision regions catch the typical sets only

- Refuse to decide when ambiguous

## Review: Fixed blocks, Soft deadlines: Forney-68



- 1 bit feedback can request retransmissions
  - Can interpret as expected block-length
  - Decision regions catch the typical sets only
  - Better error exponents at lower rates
- 
- Refuse to decide when ambiguous

## Review: Fixed blocks, Soft deadlines: Burnashev-76

- Is more feedback helpful?
- Burnashev said yes:

# Review: Fixed blocks, Soft deadlines: Burnashev-76

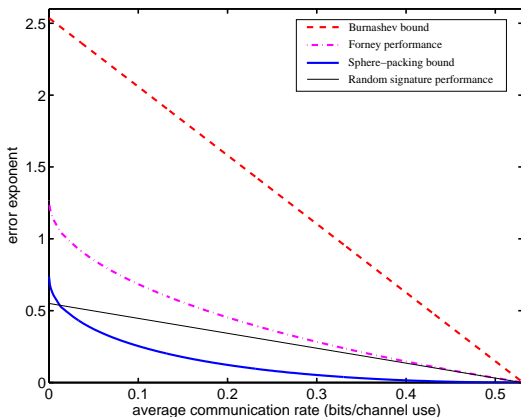
- Is more feedback helpful?
- Burnashev said yes:
  - ▶ Considered expected stopping time and used Martingale arguments.

## Review: Fixed blocks, Soft deadlines: Burnashev-76

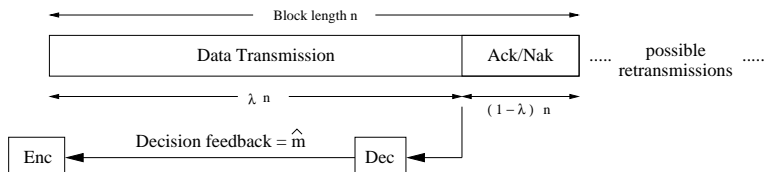
- Is more feedback helpful?
- Burnashev said yes:
  - ▶ Considered expected stopping time and used Martingale arguments.
  - ▶ Showed  $C_1(1 - \frac{R}{C})$  was a bound where  $C_1 = \max_{i,j} D(p_i || p_j)$

# Review: Fixed blocks, Soft deadlines: Burnashev-76

- Is more feedback helpful?
- Burnashev said yes:
  - ▶ Considered expected stopping time and used Martingale arguments.
  - ▶ Showed  $C_1(1 - \frac{R}{C})$  was a bound where  $C_1 = \max_{i,j} D(p_i || p_j)$

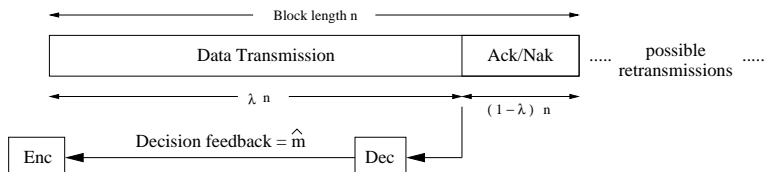


# Review: Yamamoto-Itoh-79 strategy attains the Burnashev bound



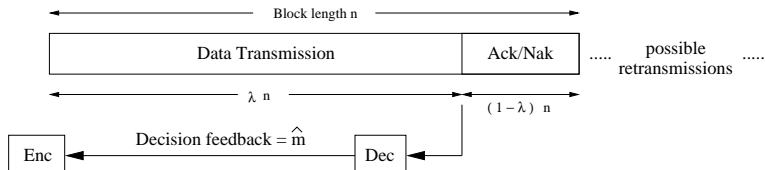
- **Data Transmission:**  $\lambda n$  channel uses for block code at  $R \simeq C$

# Review: Yamamoto-Itoh-79 strategy attains the Burnashev bound



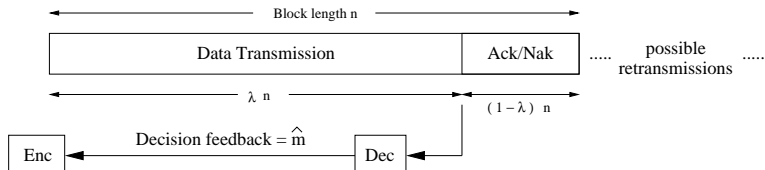
- **Data Transmission:**  $\lambda n$  channel uses for block code at  $R \simeq C$
- **Decision Feedback:**  $\hat{m}$  sent back

# Review: Yamamoto-Itoh-79 strategy attains the Burnashev bound



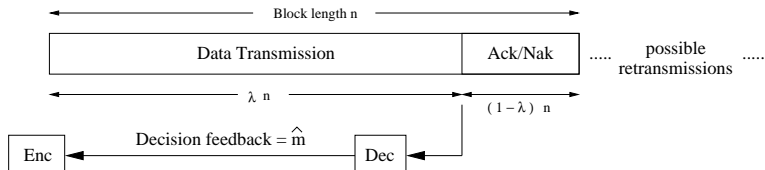
- **Data Transmission:**  $\lambda n$  channel uses for block code at  $R \simeq C$
- **Decision Feedback:**  $\hat{m}$  sent back
- **Confirm/Deny:**  $(1 - \lambda)n$  channel uses to ACK or NAK

# Review: Yamamoto-Itoh-79 strategy attains the Burnashev bound



- **Data Transmission:**  $\lambda n$  channel uses for block code at  $R \simeq C$
- **Decision Feedback:**  $\hat{m}$  sent back
- **Confirm/Deny:**  $(1 - \lambda)n$  channel uses to ACK or NAK
- If confirmed, decode to  $\hat{m}$  otherwise erase.

# Review: Yamamoto-Itoh-79 strategy attains the Burnashev bound

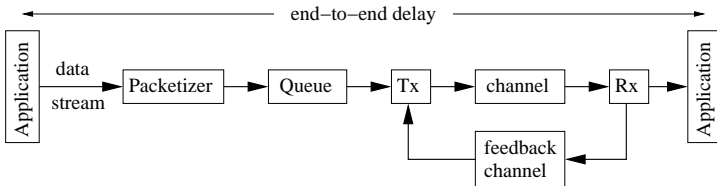


- **Data Transmission:**  $\lambda n$  channel uses for block code at  $R \simeq C$
- **Decision Feedback:**  $\hat{m}$  sent back
- **Confirm/Deny:**  $(1-\lambda)n$  channel uses to ACK or NAK
- If confirmed, decode to  $\hat{m}$  otherwise erase.
- $\Pr[\text{err}] = \Pr[\text{NAK} \rightarrow \text{ACK}] = 2^{-(1-\lambda)nC_1} \simeq 2^{-nC_1(1-\frac{R}{C})}$

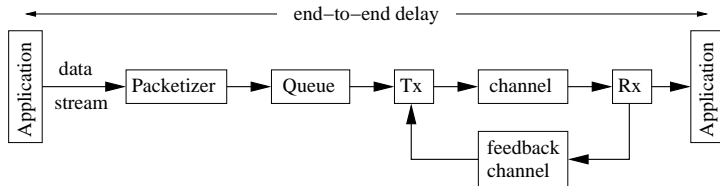
# Outline

- 1 Motivation and background
- 2 **General bidirectional channels: soft blocks**
- 3 General bidirectional channels: soft streaming
- 4 Bidirectional erasure channels: hard streaming
  - ▶ Separate forward and feedback
  - ▶ Shared forward and feedback

# A perspective on soft deadlines

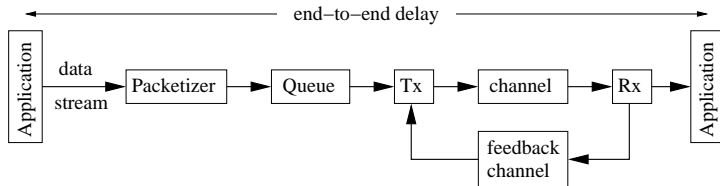


# A perspective on soft deadlines



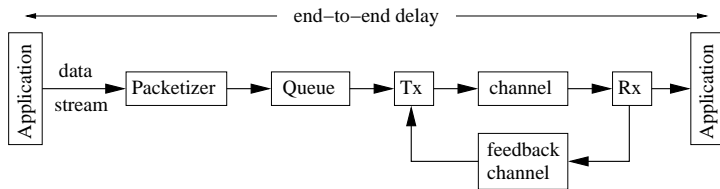
- Queue is optional. Needed if retransmissions are required

# A perspective on soft deadlines

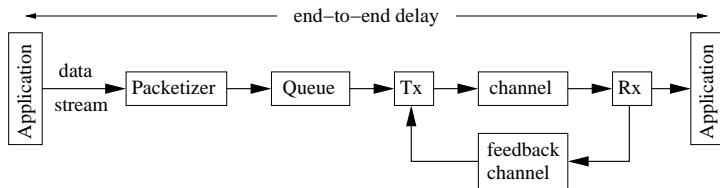


- Queue is optional. Needed if retransmissions are required
- If retransmissions are rare, then **expected** end-to-end delay is dominated by the Tx to Rx delay.

# Two distinct issues

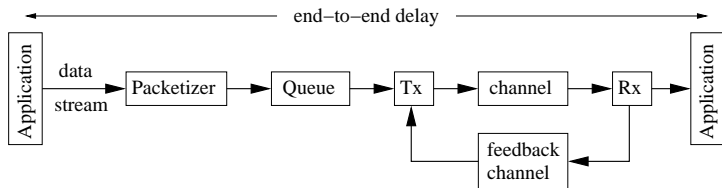


# Two distinct issues



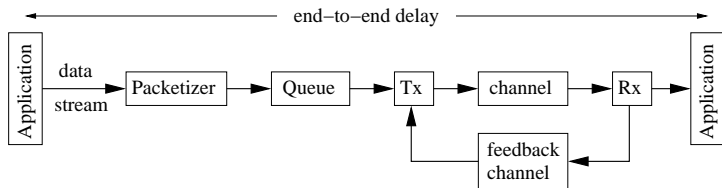
- Retransmission control: maintaining synchronization
- How to NAK?

# Two distinct issues



- Retransmission control: maintaining synchronization
  - ▶ Make sure that the encoder knows what message the decoder is expecting.
- How to NAK?

# Two distinct issues



- **Retransmission control: maintaining synchronization**
  - ▶ Make sure that the encoder knows what message the decoder is expecting.
  - ▶ Can model the state of the receiver as a random walk with drift.
  - ▶ Unstable process, so how can it be reliably tracked?
- **How to NAK?**

# Tracking unstable processes

*“It is worth stressing that we have proved only a source coding theorem for the Wiener process, not an information transmission theorem. If uncorrected channel errors were to occur, even in extremely rare instances, the user would eventually lose track of the Wiener process completely. It appears (although it has never been proved) that, even if a noisy feedback link were provided, it still would not be possible to achieve a finite [mean squared error] per letter as  $t \rightarrow \infty$ .” — Berger ’71*

# Tracking unstable processes

*“It is worth stressing that we have proved only a source coding theorem for the Wiener process, not an information transmission theorem. If uncorrected channel errors were to occur, even in extremely rare instances, the user would eventually lose track of the Wiener process completely. It appears (although it has never been proved) that, even if a noisy feedback link were provided, it still would not be possible to achieve a finite [mean squared error] per letter as  $t \rightarrow \infty$ .” — Berger ’71*

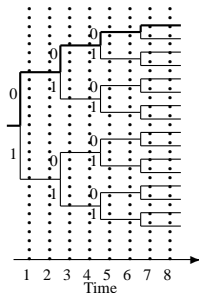
- It is possible to track an unstable process.

# Tracking unstable processes

*“It is worth stressing that we have proved only a source coding theorem for the Wiener process, not an information transmission theorem. If uncorrected channel errors were to occur, even in extremely rare instances, the user would eventually lose track of the Wiener process completely. It appears (although it has never been proved) that, even if a noisy feedback link were provided, it still would not be possible to achieve a finite [mean squared error] per letter as  $t \rightarrow \infty$ .” — Berger ’71*

- It is possible to track an unstable process.
- Key Ingredient: anytime code

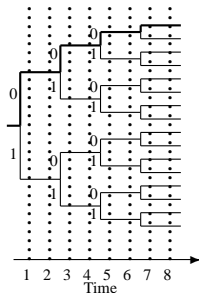
# Anytime codes without feedback



Infinite binary tree, with iid random labels:

- Choose a path through the tree based on data bits
- Transmit the path labels through the channel

# Anytime codes without feedback

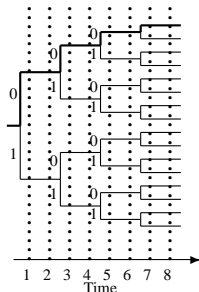


- ML decoding
  - ▶ Disjoint paths are pairwise independent of the true path.
  - ▶  $E_r(R)$  analysis applies: *future events dominate.*

Infinite binary tree, with iid random labels:

- Choose a path through the tree based on data bits
- Transmit the path labels through the channel

# Anytime codes without feedback

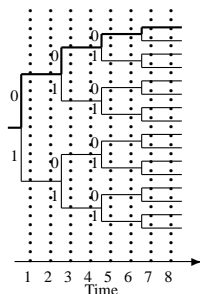


Infinite binary tree, with iid random labels:

- Choose a path through the tree based on data bits
- Transmit the path labels through the channel

- ML decoding
  - ▶ Disjoint paths are pairwise independent of the true path.
  - ▶  $E_r(R)$  analysis applies: *future events dominate.*
- Can implement with time-varying random convolutional code.

# Anytime codes without feedback



Infinite binary tree, with iid random labels:

- Choose a path through the tree based on data bits
- Transmit the path labels through the channel

- ML decoding
  - ▶ Disjoint paths are pairwise independent of the true path.
  - ▶  $E_r(R)$  analysis applies: *future events dominate.*
- Can implement with time-varying random convolutional code.
- **Achieves**  
 $P_e(d) \leq K \exp(-E_r(R)d)$   
**for every  $d$  for all  $R < C$**

# How to NAK over noisy channels

- First key idea: use an identification code
  
  
  
  
  
  
  
  
  
  
- Second key idea: Do not postpone the NAK too long.

# How to NAK over noisy channels

- First key idea: use an identification code
  - ▶ Source knows what message it sent.
  
- Second key idea: Do not postpone the NAK too long.

# How to NAK over noisy channels

- First key idea: use an identification code
  - ▶ Source knows what message it sent.
  - ▶ Use a random hash of the entire packet.
  
- Second key idea: Do not postpone the NAK too long.

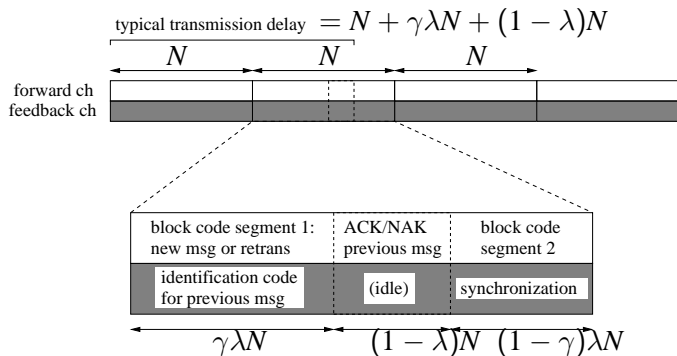
# How to NAK over noisy channels

- First key idea: use an identification code
  - ▶ Source knows what message it sent.
  - ▶ Use a random hash of the entire packet.
  - ▶ Interpret mismatch as NAK.
- Second key idea: Do not postpone the NAK too long.

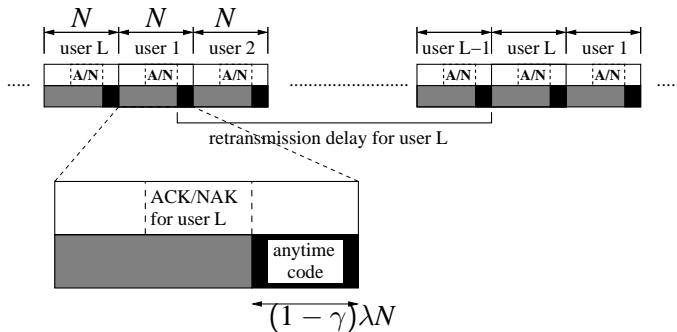
# How to NAK over noisy channels

- First key idea: use an identification code
  - ▶ Source knows what message it sent.
  - ▶ Use a random hash of the entire packet.
  - ▶ Interpret mismatch as NAK.
  - ▶ Probability of hash collision is  $\frac{1}{2^{LC/b}}$ .
- Second key idea: Do not postpone the NAK too long.

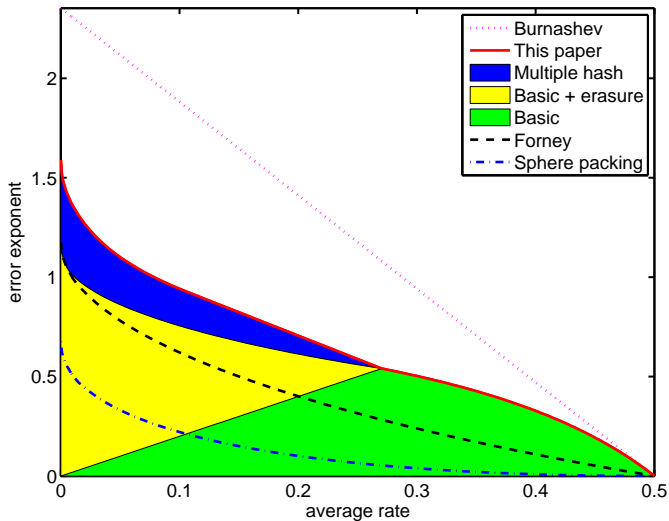
# Frame splitting



# Give the anytime code time to converge



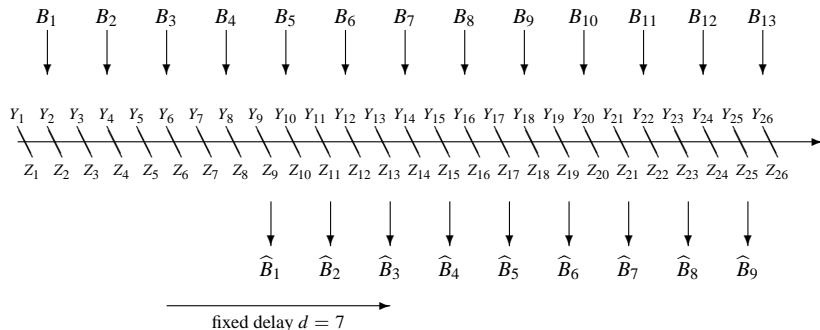
# Final performance



# Outline

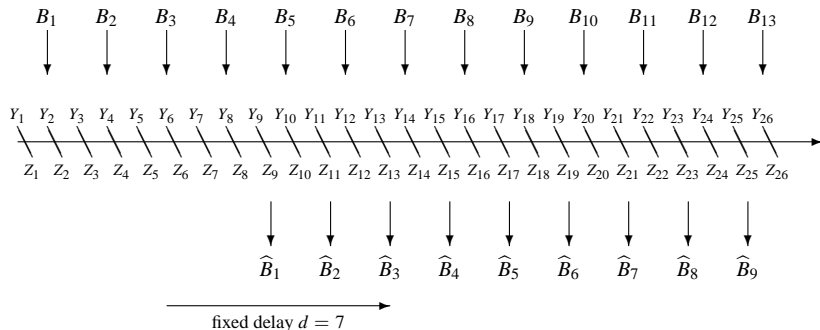
- 1 Motivation and background
- 2 General bidirectional channels: soft blocks
- 3 **General bidirectional channels: soft streaming**
- 4 Bidirectional erasure channels: hard streaming
  - ▶ Separate forward and feedback
  - ▶ Shared forward and feedback

# Communication with a latency requirement



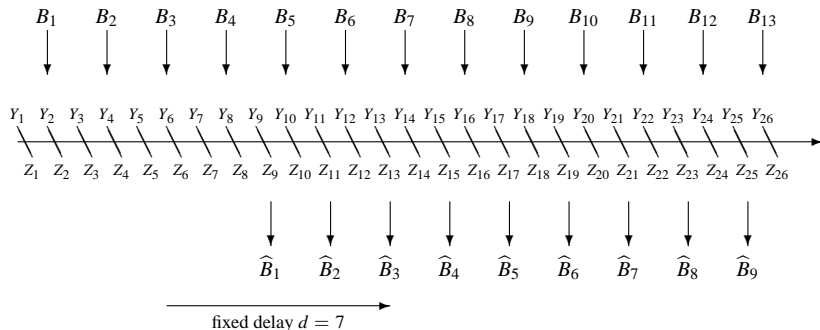
- Bits/packets arrive steadily at rate  $R$  bits per channel use.

# Communication with a latency requirement



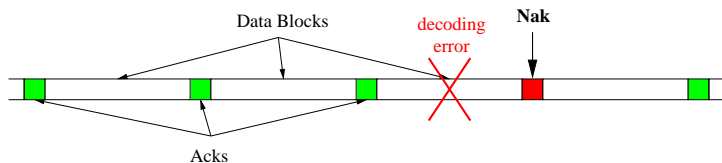
- Bits/packets arrive steadily at rate  $R$  bits per channel use.
- End-to-end latency requirement of  $d$

# Communication with a latency requirement



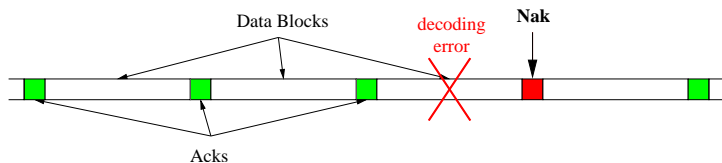
- Bits/packets arrive steadily at rate  $R$  bits per channel use.
- End-to-end latency requirement of  $d$ 
  - ▶ Hard: Declared or undetected errors are equally bad.
  - ▶ Soft: Undeclared errors are very bad, but declared errors should be infrequent.

# Streaming: an opportunity presents itself



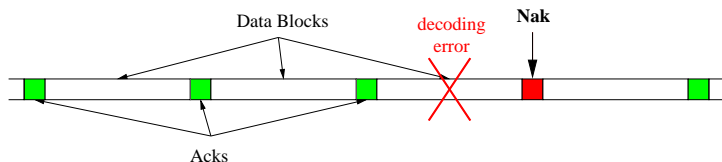
- Erasures are rare so most messages are confirmed.

# Streaming: an opportunity presents itself



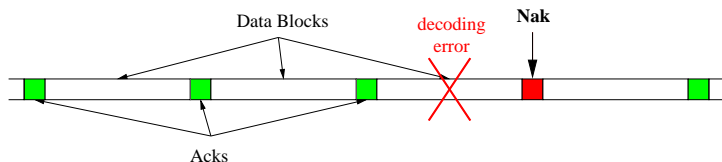
- Erasures are rare so most messages are confirmed.
- We are wasting channel uses.

# Streaming: an opportunity presents itself



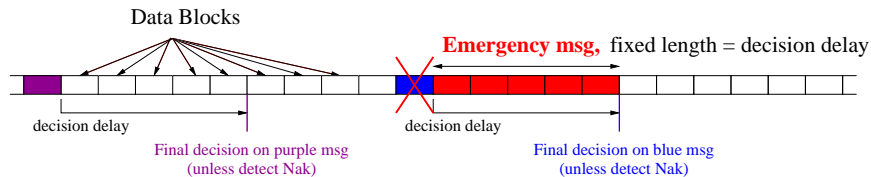
- Erasures are rare so most messages are confirmed.
- We are wasting channel uses.
- What if we only sent NAKs when needed?

# Streaming: an opportunity presents itself



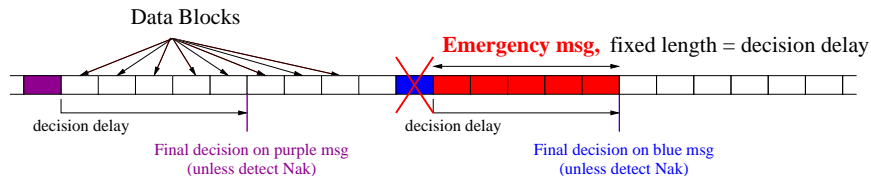
- Erasures are rare so most messages are confirmed.
- We are wasting channel uses.
- What if we only sent NAKs when needed?
- Have a special message for this purpose.

# Sliding blocks with collective punishment only (Kudryashov-79)



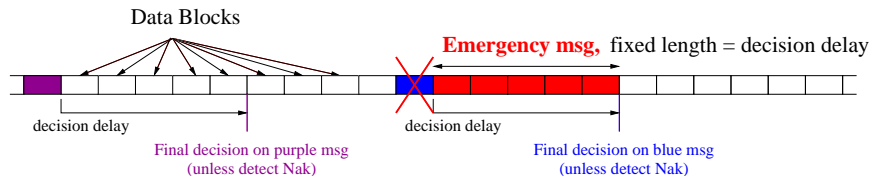
- Make packet size  $n$  much smaller than soft deadline  $d$ .

# Sliding blocks with collective punishment only (Kudryashov-79)



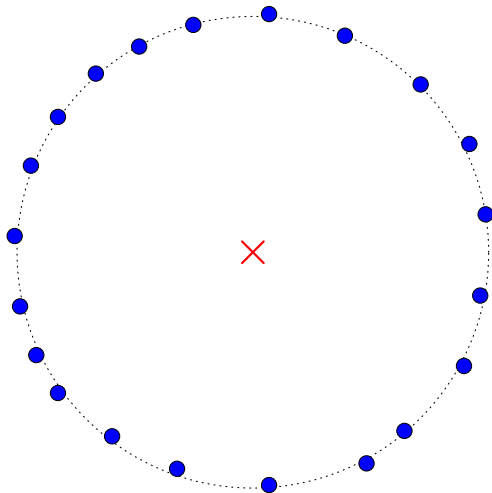
- Make packet size  $n$  much smaller than soft deadline  $d$ .
- A NAK collectively denies the past  $\frac{d}{n} - 1$  packets

# Sliding blocks with collective punishment only (Kudryashov-79)

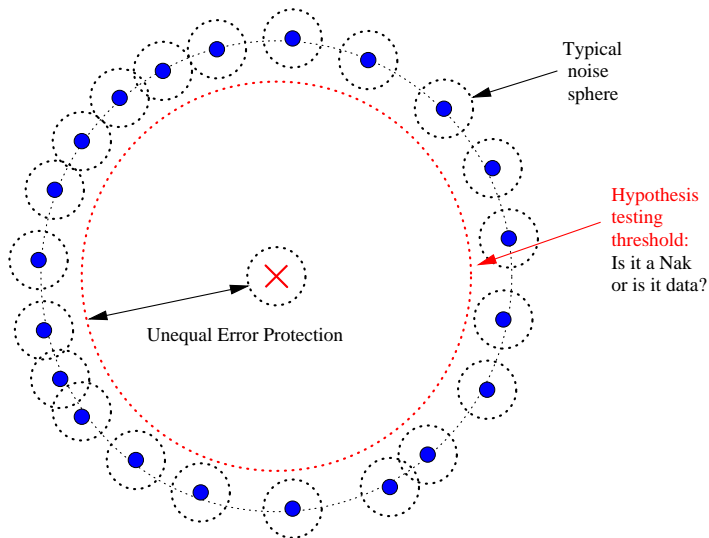


- Make packet size  $n$  much smaller than soft deadline  $d$ .
- A NAK collectively denies the past  $\frac{d}{n} - 1$  packets
- Error only if  $\frac{d}{n} - 1$  NAKs are all missed

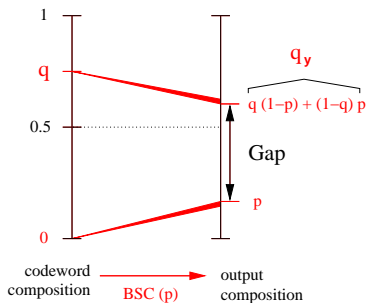
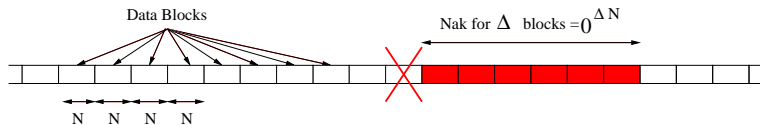
# Unequal error protection required in codebook



# Unequal error protection required in codebook

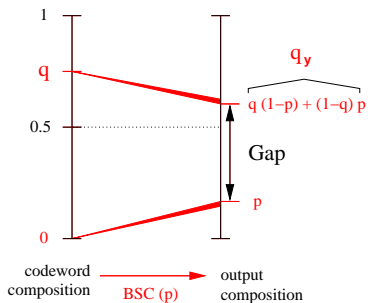
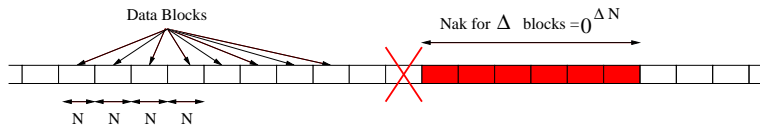


# Specialize to BSC case



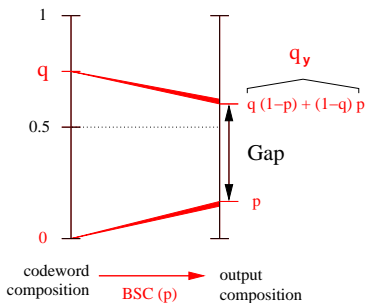
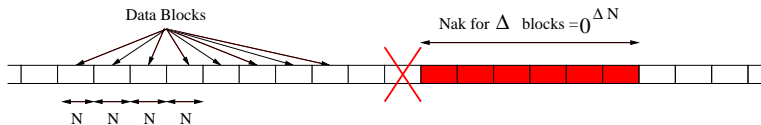
- Use all zero for NAK

# Specialize to BSC case



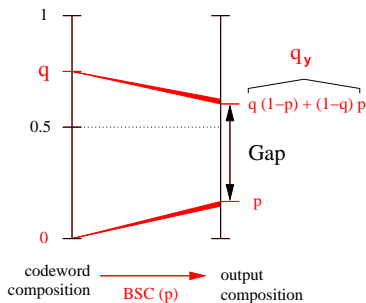
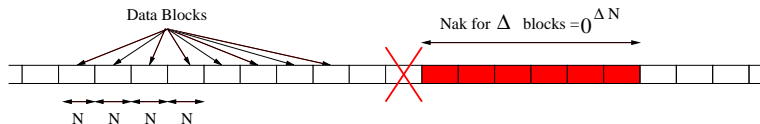
- Use all zero for NAK
- Use composition  $q$  code for data:  $R < H(q) - H(p)$

# Specialize to BSC case



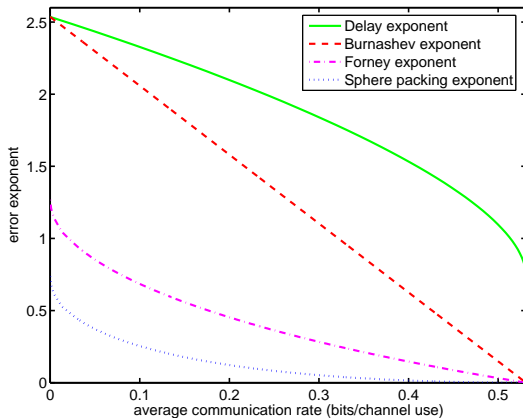
- Use all zero for NAK
- Use composition  $q$  code for data:  $R < H(q) - H(p)$
- Probability of missed NAK is  $2^{-ND(q_y||p)}$

# Specialize to BSC case

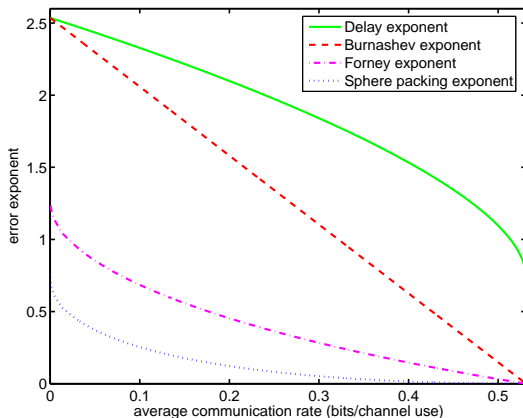


- Use all zero for NAK
- Use composition  $q$  code for data:  $R < H(q) - H(p)$
- Probability of missed NAK is  $2^{-ND(q_y||p)}$
- Get  $\Delta$  chances:  $2^{-\Delta ND(q_y||p)}$

# Resulting exponents

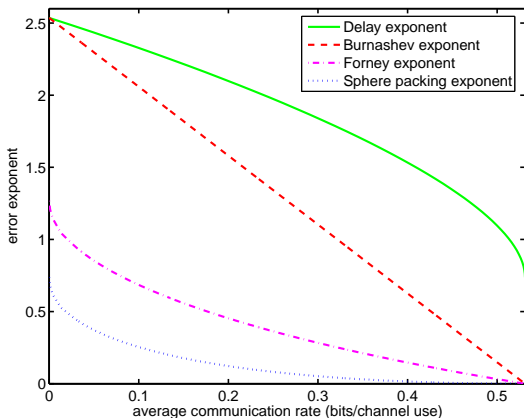


# Resulting exponents



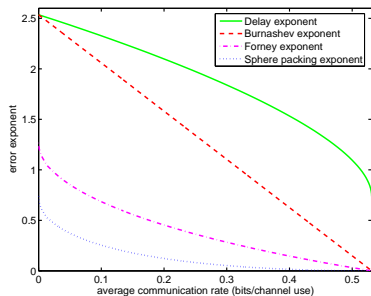
- Positive error exponent  $D(\frac{1}{2}||p)$  even at capacity!

# Resulting exponents



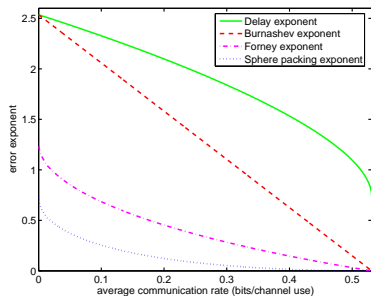
- Positive error exponent  $D(\frac{1}{2}||p)$  even at capacity!
- Matches Horstein's exponent.

# Matches the “Hallucination Bound”



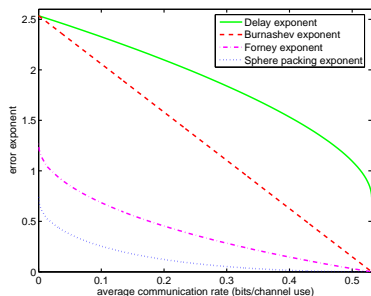
- No converse for Forney, unlike Burnashev and Sphere-packing.

# Matches the “Hallucination Bound”



- No converse for Forney, unlike Burnashev and Sphere-packing.
- What about here?

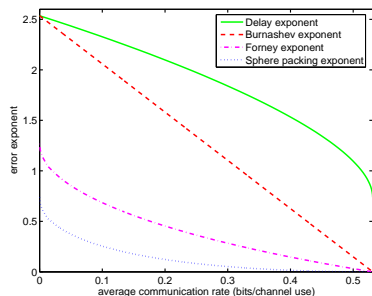
# Matches the “Hallucination Bound”



- Decoding correctly requires a certain “typical” volume of output sequences

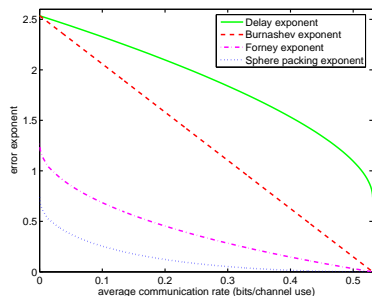
- No converse for Forney, unlike Burnashev and Sphere-packing.
- What about here?

# Matches the “Hallucination Bound”



- Decoding correctly requires a certain “typical” volume of output sequences
  - If the channel forces you to hallucinate for  $d$  time-steps, you are doomed.
- 
- No converse for Forney, unlike Burnashev and Sphere-packing.
  - What about here?

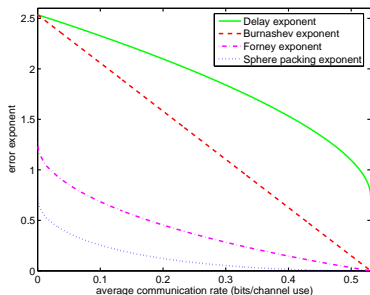
# Matches the “Hallucination Bound”



- No converse for Forney, unlike Burnashev and Sphere-packing.
- What about here?

- Decoding correctly requires a certain “typical” volume of output sequences
- If the channel forces you to hallucinate for  $d$  time-steps, you are doomed.
- With feedback, you can choose the channel input to minimize this probability.

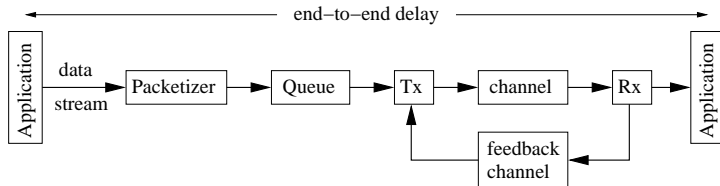
# Matches the “Hallucination Bound”



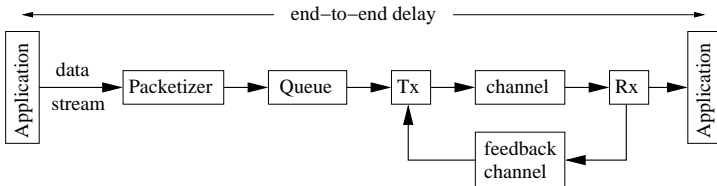
- No converse for Forney, unlike Burnashev and Sphere-packing.
- What about here?

- Decoding correctly requires a certain “typical” volume of output sequences
- If the channel forces you to hallucinate for  $d$  time-steps, you are doomed.
- With feedback, you can choose the channel input to minimize this probability.
- Matches achievability perfectly.

# The two issues revisited

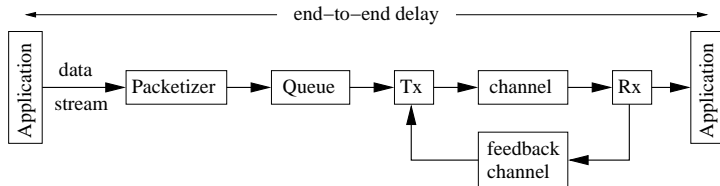


# The two issues revisited



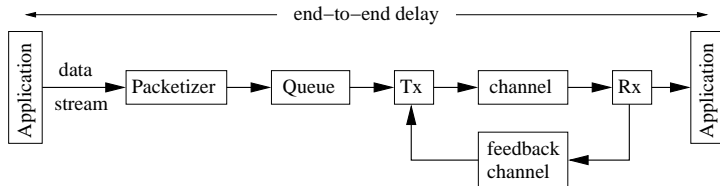
- Retransmission control: maintaining synchronization

# The two issues revisited



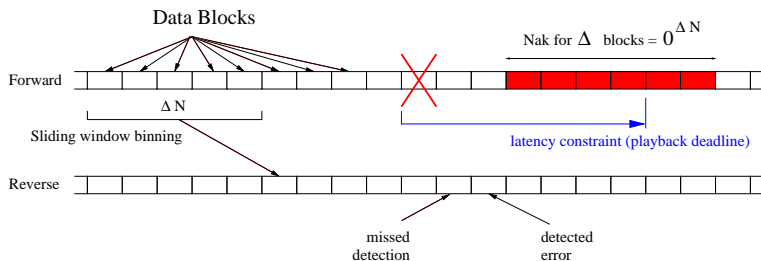
- Retransmission control: maintaining synchronization
  - ▶ As before, it can be tracked using an *anytime* code.
  - ▶ Synchronization rate is negligible — less than 1 bit per packet.

# The two issues revisited



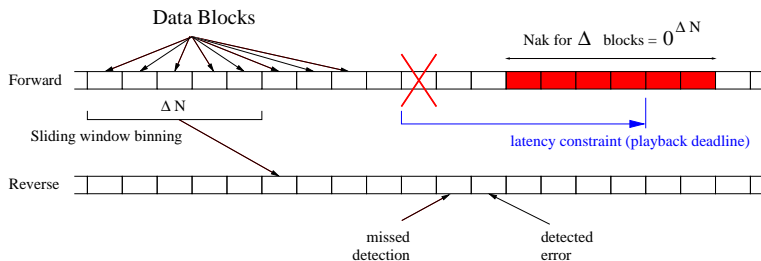
- Retransmission control: maintaining synchronization
  - ▶ As before, it can be tracked using an *anytime* code.
  - ▶ Synchronization rate is negligible — less than 1 bit per packet.
- How to NAK?

# What is needed to NAK?



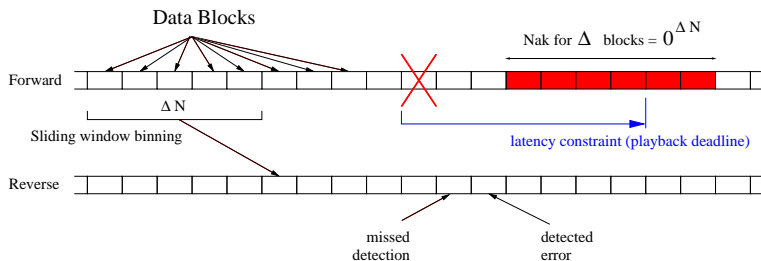
- Still an identification problem.

# What is needed to NAK?



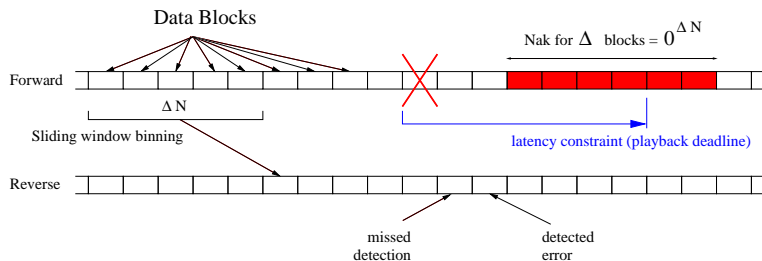
- Still an identification problem.
- Use a random hash of *entire sliding window*.

# What is needed to NAK?



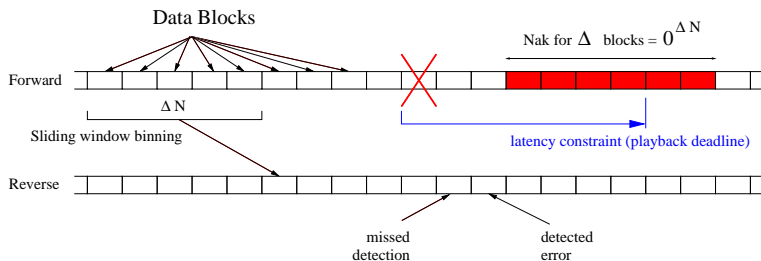
- Still an identification problem.
- Use a random hash of *entire sliding window*.
- Compare  $\Pr(\text{hash collision})$  with  $\Pr(\text{missed NAK})$

# What is needed to NAK?



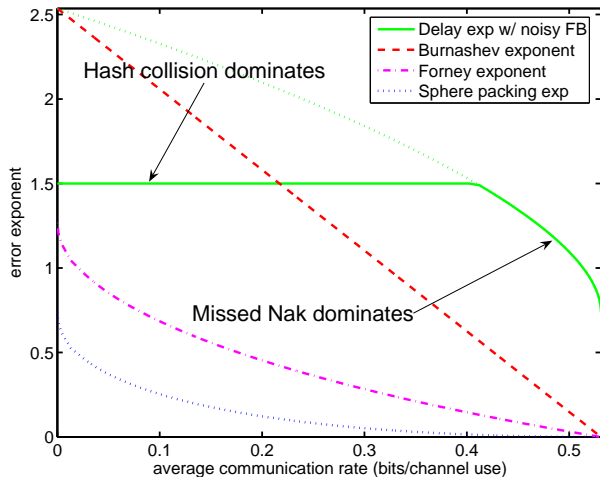
- Still an identification problem.
- Use a random hash of *entire sliding window*.
- Compare  $\Pr(\text{hash collision})$  with  $\Pr(\text{missed NAK})$
- Frame-splitting irrelevant since window is already long.

# What is needed to NAK?



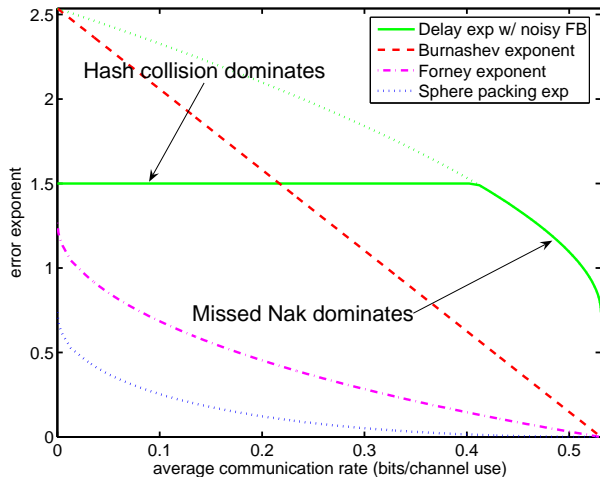
- Still an identification problem.
- Use a random hash of *entire sliding window*.
- Compare  $\Pr(\text{hash collision})$  with  $\Pr(\text{missed NAK})$
- Frame-splitting irrelevant since window is already long.
- If  $C_{fb} > D(q_y || p)$ , no loss in net exponent!

# Feedback capacity acts as a ceiling to reliability



- **Feedback reliability gains are robust to noisy feedback.**

# Feedback capacity acts as a ceiling to reliability



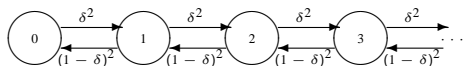
- **Feedback reliability gains are robust to noisy feedback.**
- Open problem: does this also hold in the general hard deadline case?

# Outline

- 1 Motivation and background
- 2 General bidirectional channels: soft blocks
- 3 General bidirectional channels: soft streaming
- 4 **Bidirectional erasure channels: hard streaming**
  - ▶ Separate forward and feedback
  - ▶ Shared forward and feedback

# BEC with feedback and fixed *delay*

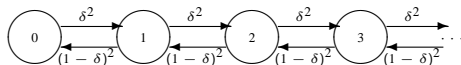
- $R = \frac{1}{2}$  example:



- Birth-death chain: positive recurrent if  $\delta < \frac{1}{2}$

## BEC with feedback and fixed *delay*

- $R = \frac{1}{2}$  example:

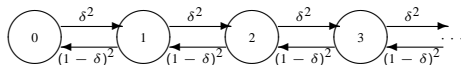


- Birth-death chain: positive recurrent if  $\delta < \frac{1}{2}$
- Delay exponent easy to see:

$$P(D \geq d) = P(L > \frac{d}{2}) = K \left( \frac{\delta}{1-\delta} \right)^d$$

## BEC with feedback and fixed *delay*

- $R = \frac{1}{2}$  example:



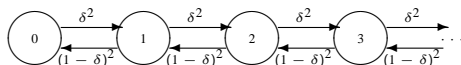
- Birth-death chain: positive recurrent if  $\delta < \frac{1}{2}$
- Delay exponent easy to see:

$$P(D \geq d) = P(L > \frac{d}{2}) = K \left( \frac{\delta}{1-\delta} \right)^d$$

- $\approx 0.584$  vs  $0.0294$  for block-coding with  $\delta = 0.4$

## BEC with feedback and fixed *delay*

- $R = \frac{1}{2}$  example:



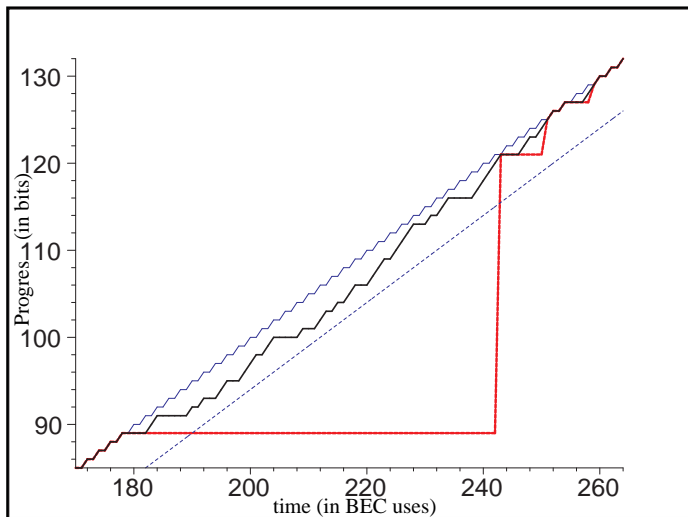
- Birth-death chain: positive recurrent if  $\delta < \frac{1}{2}$
- Delay exponent easy to see:

$$P(D \geq d) = P(L > \frac{d}{2}) = K \left( \frac{\delta}{1-\delta} \right)^d$$

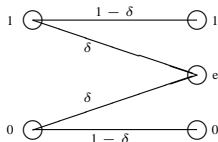
- $\approx 0.584$  vs  $0.0294$  for block-coding with  $\delta = 0.4$

**Block-coding is misleading!**

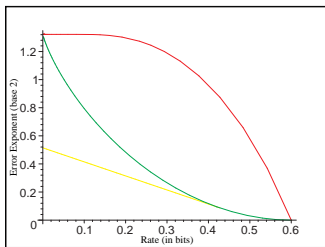
## So where is this boost coming from?



# Erasure channels with perfect feedback



- Simple capacity  $1 - \delta$  bits per channel use
- With perfect non-delayed feedback, simple to achieve: retransmit until it gets through



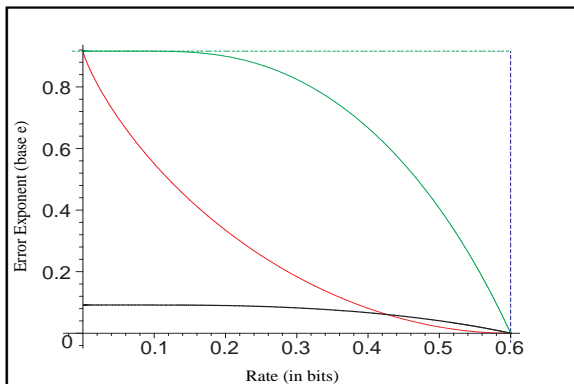
- Hard Deadline bounds
  - ▶ Without Feedback: Sphere-packing bound  $D(1 - R || \delta)$
  - ▶ With Feedback: Uncertainty-focusing bound  $(\frac{E_0(\rho)}{\rho}, E_0(\rho))$

## $k$ -delayed feedback packet erasure case

*First approach: treat as  $k$  parallel unit-delay channels*

## $k$ -delayed feedback packet erasure case

*First approach: treat as  $k$  parallel unit-delay channels*



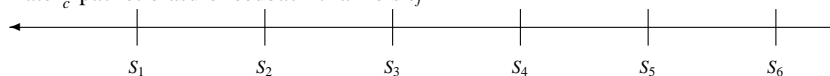
Serious penalty to waiting  $k$  steps between retransmissions

# Unreliable feedback picture

Forward packet-erasure channels  $\delta$



Rate  $\frac{1}{c}$  packet-erasure feedback channels  $\delta_f$



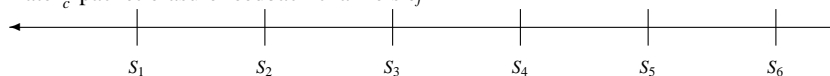
- *Both* channels drop packets randomly.

# Unreliable feedback picture

Forward packet-erasure channels  $\delta$



Rate  $\frac{1}{c}$  packet-erasure feedback channels  $\delta_f$



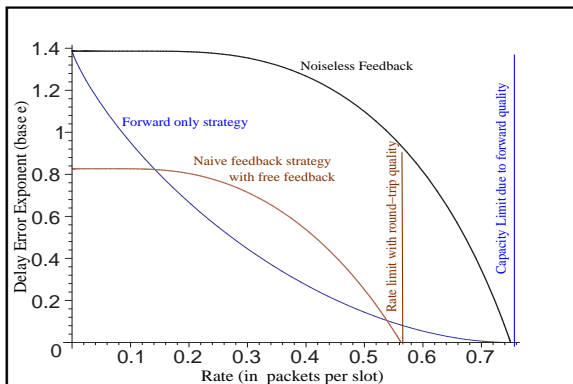
- *Both* channels drop packets randomly.
- Assume target latency  $d \gg c, k$ .

## Example: unreliable feedback

*First approach: consider a lost feedback as negative feedback*

# Example: unreliable feedback

*First approach: consider a lost feedback as negative feedback*



Capacity penalty for treating both losses symmetrically.

# Packetization: first try

- Use block-code
  - ▶ Group bits into packets with length  $nR'$
  - ▶ Transmit using a rate  $R' > R$  random block-code of length  $n \gg k$

# Packetization: first try

- Use block-code
  - ▶ Group bits into packets with length  $nR'$
  - ▶ Transmit using a rate  $R' > R$  random block-code of length  $n \gg k$
  - ▶ Use feedback to ACK/NAK only (low rate)
  - ▶ Retransmit block if unsuccessful

# Packetization: first try

- Use block-code
  - ▶ Group bits into packets with length  $nR'$
  - ▶ Transmit using a rate  $R' > R$  random block-code of length  $n \gg k$
  - ▶ Use feedback to ACK/NAK only (low rate)
  - ▶ Retransmit block if unsuccessful
- Effective block-length  $n + k \approx n$

# Packetization: first try

- Use block-code
  - ▶ Group bits into packets with length  $nR'$
  - ▶ Transmit using a rate  $R' > R$  random block-code of length  $n \gg k$
  - ▶ Use feedback to ACK/NAK only (low rate)
  - ▶ Retransmit block if unsuccessful
- Effective block-length  $n + k \approx n$
- Performance
  - ▶  $E_r(R) < E_0(\rho)$  governs probability of block retransmission

# Packetization: first try

- Use block-code
  - ▶ Group bits into packets with length  $nR'$
  - ▶ Transmit using a rate  $R' > R$  random block-code of length  $n \gg k$
  - ▶ Use feedback to ACK/NAK only (low rate)
  - ▶ Retransmit block if unsuccessful
- Effective block-length  $n + k \approx n$
- Performance
  - ▶  $E_r(R) < E_0(\rho)$  governs probability of block retransmission
  - ▶ **Bad, even without accounting for queuing delay**

# Adapting Hybrid ARQ

- Softer retransmission:

# Adapting Hybrid ARQ

- Softer retransmission:
  - 1 Group bits into blocks of size  $nR$ . ( $c \ll n \ll d$ )

# Adapting Hybrid ARQ

- Softer retransmission:
  - 1 Group bits into blocks of size  $nR$ . ( $c \ll n \ll d$ )
  - 2 Hold blocks in a FIFO queue

# Adapting Hybrid ARQ

- Softer retransmission:
  - 1 Group bits into blocks of size  $nR$ . ( $c \ll n \ll d$ )
  - 2 Hold blocks in a FIFO queue
  - 3 Service using an  $\infty$ -length random codebook. (rateless code)

# Adapting Hybrid ARQ

- Softer retransmission:
  - 1 Group bits into blocks of size  $nR$ . ( $c \ll n \ll d$ )
  - 2 Hold blocks in a FIFO queue
  - 3 Service using an  $\infty$ -length random codebook. (rateless code)
  - 4 Feedback “ACK” when we can decode.

# Adapting Hybrid ARQ

- Softer retransmission:
  - 1 Group bits into blocks of size  $nR$ . ( $c \ll n \ll d$ )
  - 2 Hold blocks in a FIFO queue
  - 3 Service using an  $\infty$ -length random codebook. (rateless code)
  - 4 Feedback “ACK” when we can decode.
- Three delays:

# Adapting Hybrid ARQ

- Softer retransmission:
  - 1 Group bits into blocks of size  $nR$ . ( $c \ll n \ll d$ )
  - 2 Hold blocks in a FIFO queue
  - 3 Service using an  $\infty$ -length random codebook. (rateless code)
  - 4 Feedback “ACK” when we can decode.
- Three delays:
  - ▶ Assembly:  $n$  insignificant relative to  $d$

# Adapting Hybrid ARQ

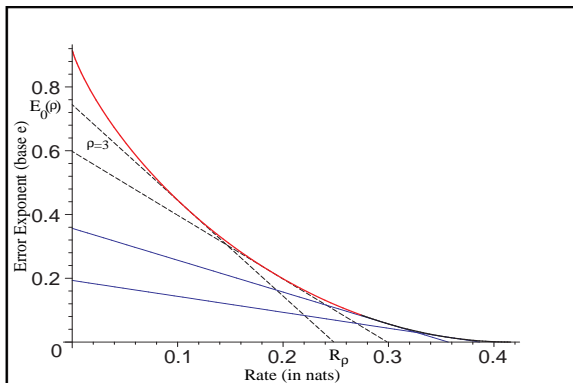
- Softer retransmission:
  - 1 Group bits into blocks of size  $nR$ . ( $c \ll n \ll d$ )
  - 2 Hold blocks in a FIFO queue
  - 3 Service using an  $\infty$ -length random codebook. (rateless code)
  - 4 Feedback “ACK” when we can decode.
- Three delays:
  - ▶ Assembly:  $n$  insignificant relative to  $d$
  - ▶ Queuing: Wait before servicing starts

# Adapting Hybrid ARQ

- Softer retransmission:
  - 1 Group bits into blocks of size  $nR$ . ( $c \ll n \ll d$ )
  - 2 Hold blocks in a FIFO queue
  - 3 Service using an  $\infty$ -length random codebook. (rateless code)
  - 4 Feedback “ACK” when we can decode.
- Three delays:
  - ▶ Assembly:  $n$  insignificant relative to  $d$
  - ▶ Queuing: Wait before servicing starts
  - ▶ Transmission: Service-time distribution

# Transmission-delay: operational interpretation of $E_0(\rho)$

*Block transmission time  $T$  can be bounded by a constant plus a geometric random variable.*



**Need to do list-decoding at low rates.**

# Queuing delay: the point message view

- Consider the queue at the level of blocks:
  - ▶ Arrive every  $n$  channel uses

# Queuing delay: the point message view

- Consider the queue at the level of blocks:
  - ▶ Arrive every  $n$  channel uses
  - ▶ When serviced, immediately consume a constant  $n \frac{R}{R_\rho}$  channel uses

# Queuing delay: the point message view

- Consider the queue at the level of blocks:
  - ▶ Arrive every  $n$  channel uses
  - ▶ When serviced, immediately consume a constant  $n \frac{R}{R_\rho}$  channel uses
  - ▶ After that, geometric service-time with  $1 - \delta_\rho = 1 - \exp(-E_0(\rho))$

## Queuing delay: the point message view

- Consider the queue at the level of blocks:
  - ▶ Arrive every  $n$  channel uses
  - ▶ When serviced, immediately consume a constant  $n \frac{R}{R_\rho}$  channel uses
  - ▶ After that, geometric service-time with  $1 - \delta_\rho = 1 - \exp(-E_0(\rho))$
- Delay  $> d$  implies  $\frac{d}{n}$  messages waiting in the queue

## Queuing delay: the point message view

- Consider the queue at the level of blocks:
  - ▶ Arrive every  $n$  channel uses
  - ▶ When serviced, immediately consume a constant  $n \frac{R}{R_\rho}$  channel uses
  - ▶ After that, geometric service-time with  $1 - \delta_\rho = 1 - \exp(-E_0(\rho))$
- Delay  $> d$  implies  $\frac{d}{n}$  messages waiting in the queue
- Suppose the queue last renewed  $r$  messages ago
  - ▶  $rn$  channel uses since then

## Queuing delay: the point message view

- Consider the queue at the level of blocks:
  - ▶ Arrive every  $n$  channel uses
  - ▶ When serviced, immediately consume a constant  $n \frac{R}{R_\rho}$  channel uses
  - ▶ After that, geometric service-time with  $1 - \delta_\rho = 1 - \exp(-E_0(\rho))$
- Delay  $> d$  implies  $\frac{d}{n}$  messages waiting in the queue
- Suppose the queue last renewed  $r$  messages ago
  - ▶  $rn$  channel uses since then
  - ▶ At most  $q = r - \frac{d}{n}$  blocks serviced

# Queuing delay: the point message view

- Consider the queue at the level of blocks:
  - ▶ Arrive every  $n$  channel uses
  - ▶ When serviced, immediately consume a constant  $n \frac{R}{R_\rho}$  channel uses
  - ▶ After that, geometric service-time with  $1 - \delta_\rho = 1 - \exp(-E_0(\rho))$
- Delay  $> d$  implies  $\frac{d}{n}$  messages waiting in the queue
- Suppose the queue last renewed  $r$  messages ago
  - ▶  $rn$  channel uses since then
  - ▶ At most  $q = r - \frac{d}{n}$  blocks serviced
  - ▶ So remove  $qn \frac{R}{R_\rho}$  channel uses

## Queuing delay: the point message view

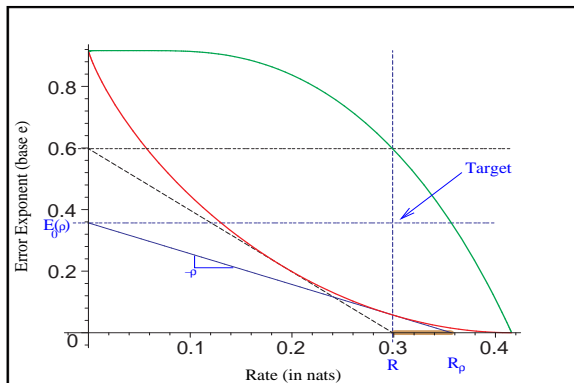
- Consider the queue at the level of blocks:
  - ▶ Arrive every  $n$  channel uses
  - ▶ When serviced, immediately consume a constant  $n \frac{R}{R_\rho}$  channel uses
  - ▶ After that, geometric service-time with  $1 - \delta_\rho = 1 - \exp(-E_0(\rho))$
- Delay  $> d$  implies  $\frac{d}{n}$  messages waiting in the queue
- Suppose the queue last renewed  $r$  messages ago
  - ▶  $rn$  channel uses since then
  - ▶ At most  $q = r - \frac{d}{n}$  blocks serviced
  - ▶ So remove  $qn \frac{R}{R_\rho}$  channel uses
  - ▶ Leaving  $d + qn(1 - \frac{R}{R_\rho})$  “slack” uses of which only  $q$  were successful.

## Queuing delay: the point message view

- Consider the queue at the level of blocks:
  - ▶ Arrive every  $n$  channel uses
  - ▶ When serviced, immediately consume a constant  $n \frac{R}{R_\rho}$  channel uses
  - ▶ After that, geometric service-time with  $1 - \delta_\rho = 1 - \exp(-E_0(\rho))$
- Delay  $> d$  implies  $\frac{d}{n}$  messages waiting in the queue
- Suppose the queue last renewed  $r$  messages ago
  - ▶  $rn$  channel uses since then
  - ▶ At most  $q = r - \frac{d}{n}$  blocks serviced
  - ▶ So remove  $qn \frac{R}{R_\rho}$  channel uses
  - ▶ Leaving  $d + qn(1 - \frac{R}{R_\rho})$  “slack” uses of which only  $q$  were successful.
- This is exactly like a rate  $\frac{1}{n(1 - \frac{R}{R_\rho})}$  code on a perfect-feedback BEC with erasure probability  $\delta_\rho$ .

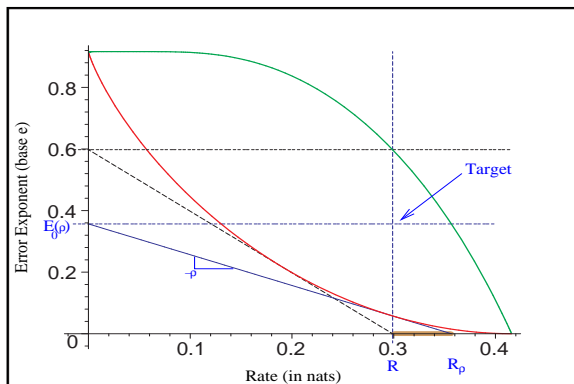
# Queuing delay: reduction to the low-rate erasure case

Pick  $R < R_\rho < C$  and aim for  $E_a^+(R_\rho) = E_0(\rho)$  exponent.



## Queuing delay: reduction to the low-rate erasure case

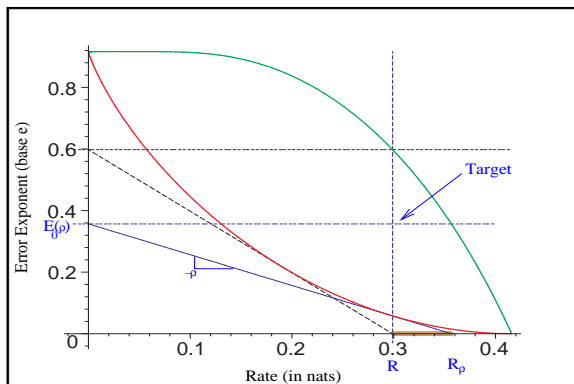
Pick  $R < R_\rho < C$  and aim for  $E_a^+(R_\rho) = E_0(\rho)$  exponent.



If  $n$  large, effective point-message rate  $(n(1 - \frac{R}{R'}))^{-1}$  is small.

## Queuing delay: reduction to the low-rate erasure case

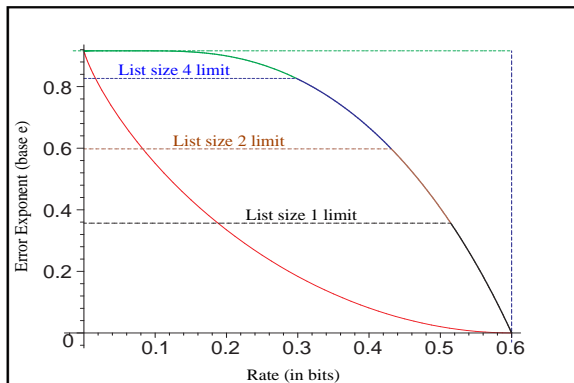
Pick  $R < R_\rho < C$  and aim for  $E_a^+(R_\rho) = E_0(\rho)$  exponent.



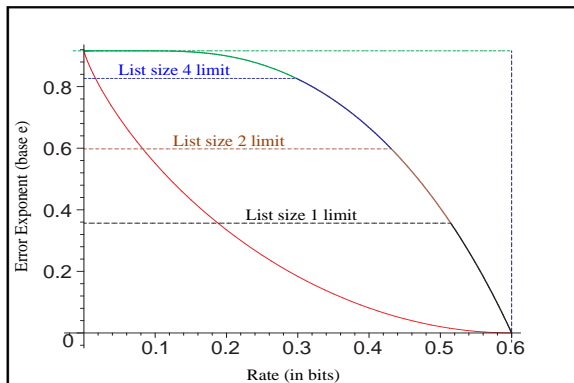
If  $n$  large, effective point-message rate  $(n(1 - \frac{R}{R'}))^{-1}$  is small.

Erasures focusing bound is approximately flat at low rates so queuing delay exponent  $\approx E_0(\rho)$ .

# Performance

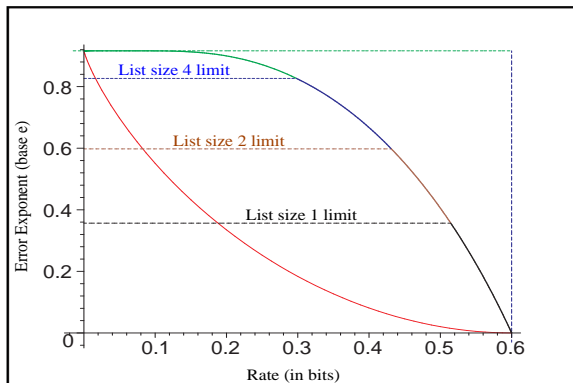


# Performance



Q1: How does the decoder know that the encoder received the ACK?

# Performance



Q2: What is the price to get list-decoding gains?



# Labeling blocks

- High rates, list-size= 1.
  - 1 Re-randomize code for each block.
  - 2 Decoder ACKs when there is only one possible message.
  
- Low rates, list-size  $\geq 2$ .

# Labeling blocks

- High rates, list-size= 1.
  - 1 Re-randomize code for each block.
  - 2 Decoder ACKs when there is only one possible message.
  - 3 Decoder advances counter when the received packet is incompatible with this one possible message.
  
- Low rates, list-size  $\geq 2$ .

# Labeling blocks

- High rates, list-size= 1.
  - 1 Re-randomize code for each block.
  - 2 Decoder ACKs when there is only one possible message.
  - 3 Decoder advances counter when the received packet is incompatible with this one possible message.
    - ★ Probability of same =  $\delta + (1 - \delta)\frac{1}{2} = \frac{1+\delta}{2}$  for BEC.
  
- Low rates, list-size  $\geq 2$ .

# Labeling blocks

- High rates, list-size= 1.
  - 1 Re-randomize code for each block.
  - 2 Decoder ACKs when there is only one possible message.
  - 3 Decoder advances counter when the received packet is incompatible with this one possible message.
    - ★ Probability of same =  $\delta + (1 - \delta)\frac{1}{2} = \frac{1+\delta}{2}$  for BEC.
    - ★ In general, it is  $\exp(E_0(1))$ .
- Low rates, list-size  $\geq 2$ .

# Labeling blocks

- High rates, list-size= 1.
  - 1 Re-randomize code for each block.
  - 2 Decoder ACKs when there is only one possible message.
  - 3 Decoder advances counter when the received packet is incompatible with this one possible message.
    - ★ Probability of same =  $\delta + (1 - \delta)\frac{1}{2} = \frac{1+\delta}{2}$  for BEC.
    - ★ In general, it is  $\exp(E_0(1))$ .
    - ★ Behaves like a second point message.
- Low rates, list-size  $\geq 2$ .

# Labeling blocks

- High rates, list-size= 1.
  - 1 Re-randomize code for each block.
  - 2 Decoder ACKs when there is only one possible message.
  - 3 Decoder advances counter when the received packet is incompatible with this one possible message.
    - ★ Probability of same =  $\delta + (1 - \delta)\frac{1}{2} = \frac{1+\delta}{2}$  for BEC.
    - ★ In general, it is  $\exp(E_0(1))$ .
    - ★ Behaves like a second point message.
- Low rates, list-size  $\geq 2$ .
  - 1 Multiple messages per block.

# Labeling blocks

- High rates, list-size= 1.
  - 1 Re-randomize code for each block.
  - 2 Decoder ACKs when there is only one possible message.
  - 3 Decoder advances counter when the received packet is incompatible with this one possible message.
    - ★ Probability of same =  $\delta + (1 - \delta)\frac{1}{2} = \frac{1+\delta}{2}$  for BEC.
    - ★ In general, it is  $\exp(E_0(1))$ .
    - ★ Behaves like a second point message.
- Low rates, list-size  $\geq 2$ .
  - 1 Multiple messages per block.
  - 2 Add a single “header” bit to forward packets (Massey)
    - ★ 0: first message sub-block
    - ★ 1: second message sub-block
    - ★ 0: third message sub-block
    - ⋮

# How to use list-decoding

- 1 Group bits into blocks of size  $nR$ . ( $k \ll n \ll d$ )
- 2 Hold blocks in a FIFO queue

# How to use list-decoding

- 1 Group bits into blocks of size  $nR$ . ( $k \ll n \ll d$ )
- 2 Hold blocks in a FIFO queue
- 3 Transmit using an  $\infty$ -length random codebook. (rateless code)

# How to use list-decoding

- 1 Group bits into blocks of size  $nR$ . ( $k \ll n \ll d$ )
- 2 Hold blocks in a FIFO queue
- 3 Transmit using an  $\infty$ -length random codebook. (rateless code)
- 4 Feedback “ACK” when we can **almost** decode.

# How to use list-decoding

- 1 Group bits into blocks of size  $nR$ . ( $k \ll n \ll d$ )
- 2 Hold blocks in a FIFO queue
- 3 Transmit using an  $\infty$ -length random codebook. (rateless code)
- 4 Feedback “ACK” when we can **almost** decode.
- 5 Send an  $O((L - 1) \log(nR))$  message requesting clarification of specific bits within the block

# How to use list-decoding

- 1 Group bits into blocks of size  $nR$ . ( $k \ll n \ll d$ )
- 2 Hold blocks in a FIFO queue
- 3 Transmit using an  $\infty$ -length random codebook. (rateless code)
- 4 Feedback “ACK” when we can **almost** decode.
- 5 Send an  $O((L - 1) \log(nR))$  message requesting clarification of specific bits within the block
- 6 Use repetition codes to request and to clarify

# How to use list-decoding

- 1 Group bits into blocks of size  $nR$ . ( $k \ll n \ll d$ )
- 2 Hold blocks in a FIFO queue
- 3 Transmit using an  $\infty$ -length random codebook. (rateless code)
- 4 Feedback “ACK” when we can **almost** decode.
- 5 Send an  $O((L - 1) \log(nR))$  message requesting clarification of specific bits within the block
- 6 Use repetition codes to request and to clarify
- 7 ACK the requests and clarifications

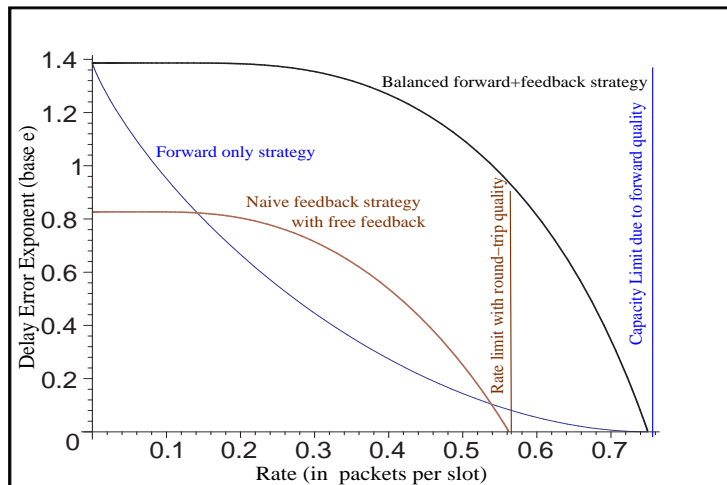
# How to use list-decoding

- 1 Group bits into blocks of size  $nR$ . ( $k \ll n \ll d$ )
  - 2 Hold blocks in a FIFO queue
  - 3 Transmit using an  $\infty$ -length random codebook. (rateless code)
  - 4 Feedback “ACK” when we can **almost** decode.
  - 5 Send an  $O((L - 1) \log(nR))$  message requesting clarification of specific bits within the block
  - 6 Use repetition codes to request and to clarify
  - 7 ACK the requests and clarifications
- A block corresponds to  $O(L \log n)$  point messages — low rate relative to  $O(n)$  slack.

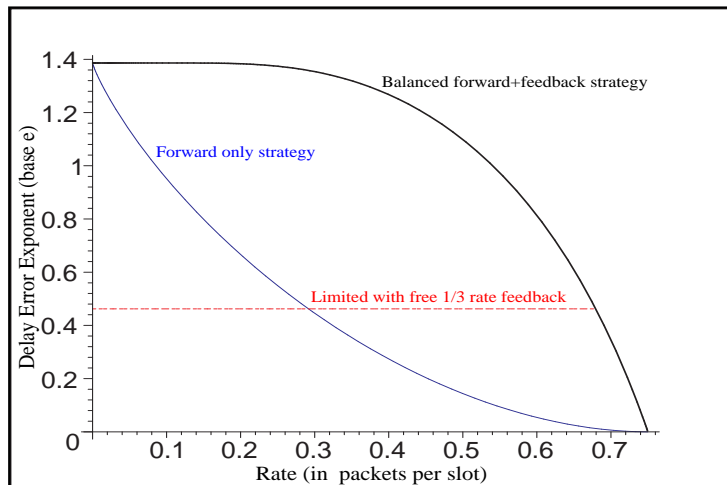
# How to use list-decoding

- 1 Group bits into blocks of size  $nR$ . ( $k \ll n \ll d$ )
  - 2 Hold blocks in a FIFO queue
  - 3 Transmit using an  $\infty$ -length random codebook. (rateless code)
  - 4 Feedback “ACK” when we can **almost** decode.
  - 5 Send an  $O((L - 1) \log(nR))$  message requesting clarification of specific bits within the block
  - 6 Use repetition codes to request and to clarify
  - 7 ACK the requests and clarifications
- A block corresponds to  $O(L \log n)$  point messages — low rate relative to  $O(n)$  slack.
  - Approaches the focusing bound with a rate penalty of the header bits.

# Performance with unreliable feedback channel



# Performance with unreliable feedback channel

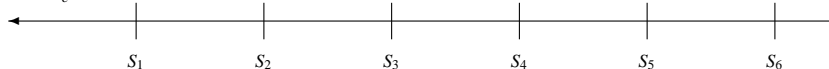


# Shared channel picture

$\frac{c-1}{c}$  forward packet-erasure channels  $\delta$

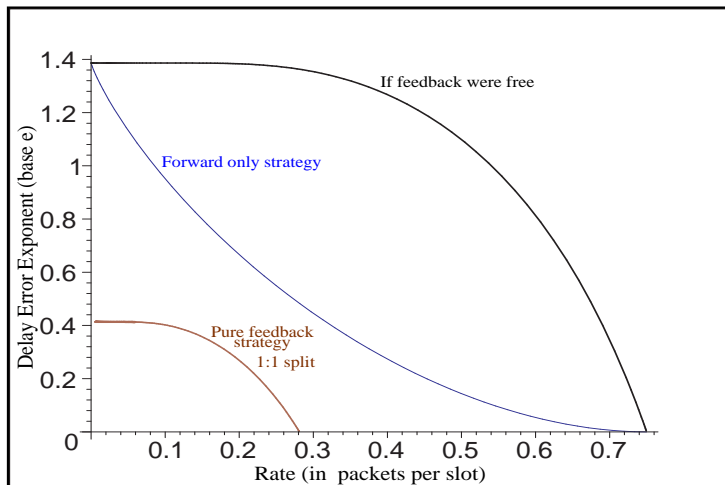


Rate  $\frac{1}{c}$  packet-erasure feedback channels  $\delta_f$

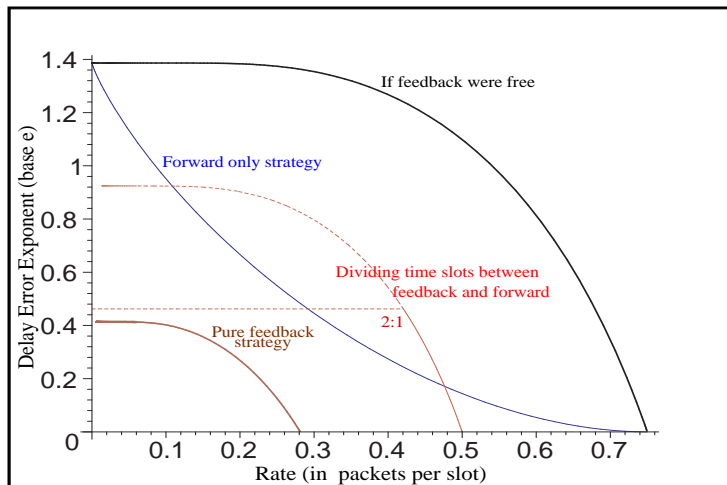


- *Both* users share a single physical channel
- Half-duplex constraint: only one can use at a time
- Assume we must schedule them in advance
- Same erasure probability in both directions

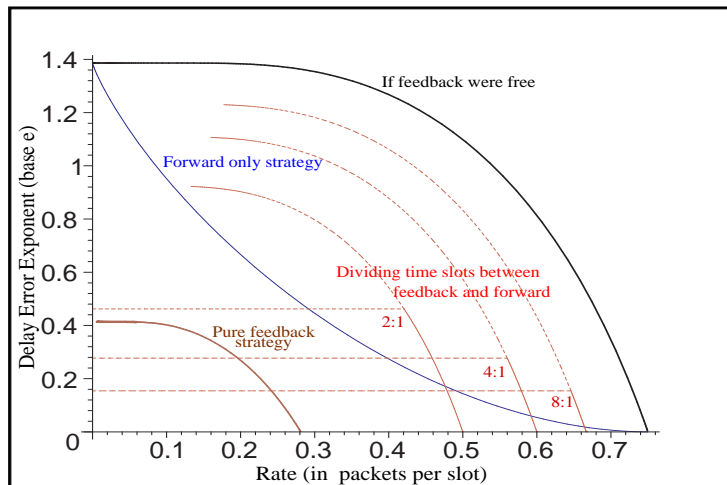
# Shared unreliable channel used for feedback



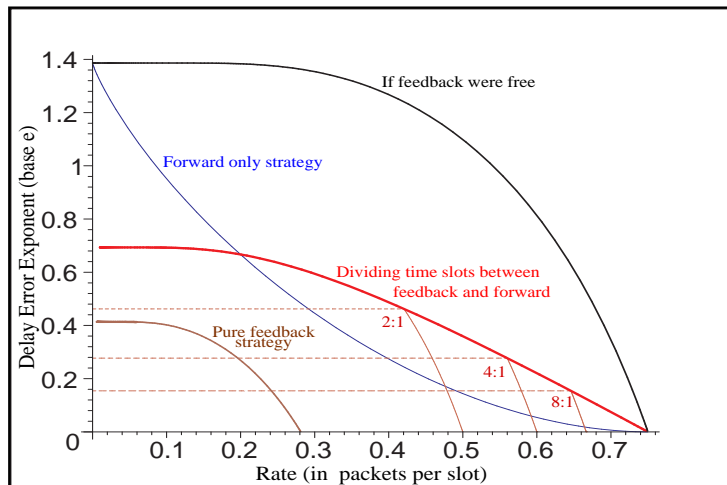
# Shared unreliable channel used for feedback



# Shared unreliable channel used for feedback



# Shared unreliable channel used for feedback



Like focusing bound using  $E'_0(\rho) = (-\ln(\delta_f))^{-1} + E_0^{-1}(\rho))^{-1}$

## Summary of results:

- Perfect feedback performance as long as  $\frac{-1}{c} \log \delta_f$  is high enough.

## Summary of results:

- Perfect feedback performance as long as  $\frac{-1}{c} \log \delta_f$  is high enough.
- It is worth allocating slots for feedback *even if this means taking them away from the forward channel.*

# Final comments on BEC

- Perfect feedback performance with arbitrarily low feedback rate.

# Final comments on BEC

- Perfect feedback performance with arbitrarily low feedback rate.
- For  $L = 1$ , random linear block coding is good enough.

# Final comments on BEC

- Perfect feedback performance with arbitrarily low feedback rate.
- For  $L = 1$ , random linear block coding is good enough.
- For  $L > 1$ , random polynomial block coding suffices.

# Final comments on BEC

- Perfect feedback performance with arbitrarily low feedback rate.
- For  $L = 1$ , random linear block coding is good enough.
- For  $L > 1$ , random polynomial block coding suffices.
- The code is “anytime” in that it is delay universal — application can pick what latency is needed.

# Final comments on BEC

- Perfect feedback performance with arbitrarily low feedback rate.
- For  $L = 1$ , random linear block coding is good enough.
- For  $L > 1$ , random polynomial block coding suffices.
- The code is “anytime” in that it is delay universal — application can pick what latency is needed.
- Computation depends only on  $n$ , not on delay.