Distributed Computing with Channel Noise

Jared Saia
University of New Mexico

Joint with Abhinav Aggarwal, Varsha Dani,
and Tom Hayes

Error Correction

Shannon

- Information Theory & Entropy
- Random Errors

Hamming

- Hamming Bound (aka Sphere-Packing Bound)
- Adversarial Errors

Richard Hamming

"And so I said, 'Damn it, if the machine can detect an error, why can't it locate the position of the error and correct it?'"

Our Problem

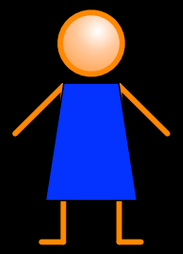
How can we **compute** over a noisy channel?

Texting with Noise

Alice, Bob & Carol have personal preferences for what they want to eat

The run a protocol to find a place that is mutually desirable

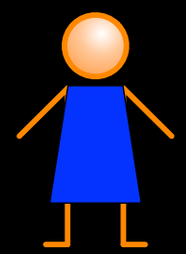
Today they have a noisy connection



Sushi?

Smoothie?

Alice wants liquid kale



Alice wants liquid ale

“Swan in Rushes” Pub?

“Swan in Rushes” Pub?

“Swan in Rushes” Pub?

No. Sushi!

No. Sushi!

No. Sushi!

k LOL

k LOL

k LOL

The Problem

Given a protocol π

π sends L bits over noise-free channels

Create a new protocol π’

π’ sends L’ bits over noisy channels

Goal: L’ is small function of L

Time

Local computation is instantaneous

Time is divided into (channel) steps

In each step, can send one bit over a channel

Goal: Minimize steps

Why Hard?

Assume

Alice and Bob just send 1 bit back and forth in π

Then

Naive use of ECC requires advance knowledge of noise rate for each transmission

Schulman ‘96

π takes L steps over noise-free channel

Only two players

An adversary flips < ε (1/240) fraction of bits

Channels are public

Result: Can create protocol π’ so L’ = θ (L)

Subsequent work: n-players

Much prior work assumes either:

There is a coordinating node connected to all players; or

Network topology is known to all players.

This talk: focus on case where the network topology is arbitrary and unknown in advance

Additionally assume that π is asynchronous

[Censor-Hillel, Gelles, Haeupler ’17]

L and T are known in advance

All channels are public

π is asynchronous

π’ is asynchronous; adversary can’t insert/delete messages

Tolerates O(1/n) fraction corruptions, sends $O(L\, n \log^2 n)$ bits

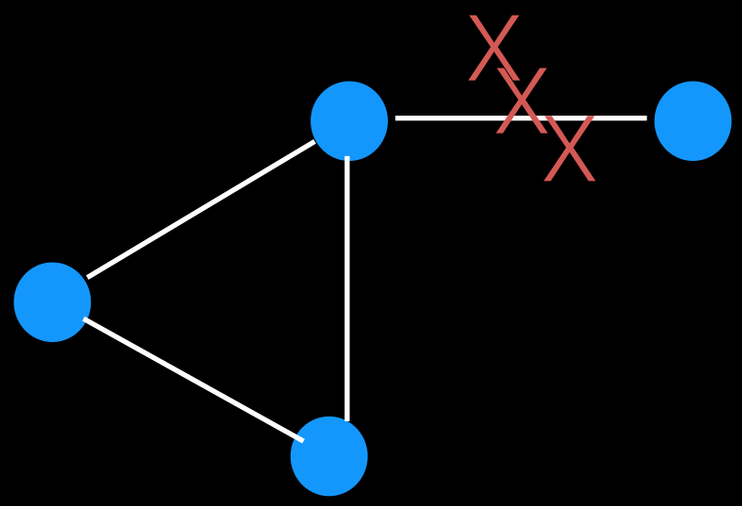
O(1/n) noise tolerance is **optimal** with public channels

Asynchronous π

Messages delivered intermittently

Players have no clocks

Why necessary?



Our Model

Adversary knows π, and our encoding algorithm

Adversary doesn't know local random bits, or bits sent on channel i.e. **private channels**

Adversary selects location of bits to flip

π is asynchronous; π' is synchronous

Can also handle π' asynchronous & adversary can't insert/delete messages

Why Private Channels?

If adversary knows bits, then

Adversary flips bits from Alice to Bob

Convinces Bob that protocol π_1 is being run rather than protocol π_2

Succeeds since can flip any number of bits

Our Result

L and T are **unknown** in advance

All channels are private

Succeeds with probability 1 - δ, for δ>0

Sends O(L log (n(L+T)/δ) + T) bits

α = average message size in π

Total bits sent: O(L (1 +1/α log (n(L+T)/δ)) + T)

Algorithm Overview

Problem: Message Corruption

Solution: Algebraic Manipulation Detection (AMD) Codes

[Cramer et al. ’08]

AMD Codes

Encode a message m into a message m’

Any bit flipping of m’ is detected with probability 1 - δ

|m’| = |m| + O(log 1/δ)

N.B.: Works only with private channels!

Problem 2: Bit Blowup

Problem: Adversary flips one bit, we must resend log bits

Solution: Make the adversary corrupt constant fraction of bits to trigger a resend

Using ECCs, can require adversary to corrupt 1/3 of the bits to trigger resend

Other Problems

Problem: Don't know L and T

Solution: Increase encoding strength over time

Problem: Differentiating message resend and repeated message

Solution: Use a message number parity bit

Problem: Adversary installs messages during channel silence

Solution: Use session keys

Algorithm

For every pair of users (say Alice and Bob):

If Alice has a message m for Bob:

ECC, AMD encoded. Appended with random bits.

Alice asks Bob for a key, and sends her key k1 along.

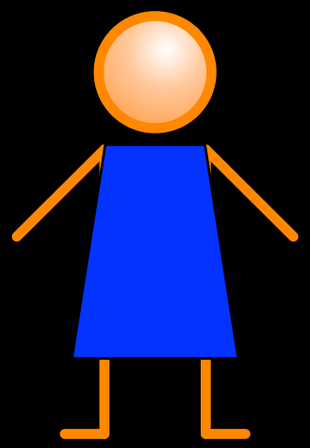
Bob replies with his key k2 along with k1'.

If k1'=k1, Alice sends ((m,b),k2) to Bob. Else, she starts over.

If received correctly, Bob is done and remains silent. Interprets m as a resend if b is unchanged, else a fresh message.

Else he sends noise.

If noise, Alice starts over. Else, she is done.



“My key is k1.
What’s yours?”

(k2,k1)

(m,k2)

…

…

Conclusion

An Application

Alice wants to send L bit message to Bob

Connected via a private channel

Neither knows T, Bob does not know L

Result: Succeeds with error probability 1/poly(L), sends expected

L + O(T + min((T+1)log L, L log L)

Only L + O(log L) bits sent when T = O(1)

Future Work

Can we match the L + O(T) lower bound?

Asynchronous π’ with adversarial insertions?

Can we unify node faults and edge faults into one grand model?

Questions?

Algorithm Overview

First Algorithm

For every pair of users (say Alice and Bob):

If Alice has a message m for Bob:

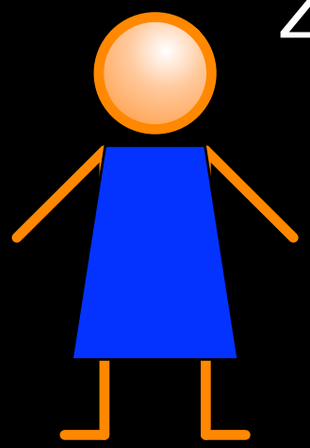
Round

Alice sends m to Bob. ECC,AMD encoded

If received correctly, Bob is done and remains silent. Else he sends noise.

If noise, Alice starts over. Else, she is done.

Problem: Sleeping Players



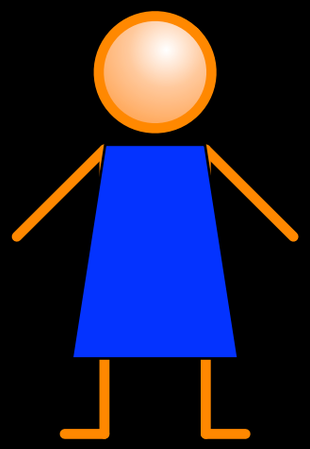
m

…

Problem: Adversary installs a message while Alice sleeps

…

Solution: Keys



“Key Please”

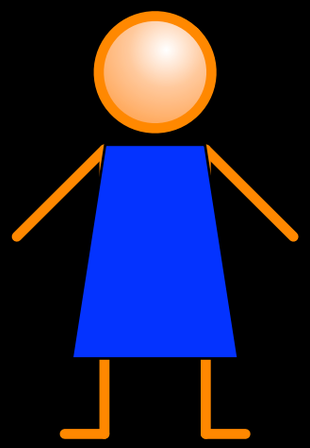
k

(m,k)

…

…

Problem: Sleeping Bob



"Key Please"

k

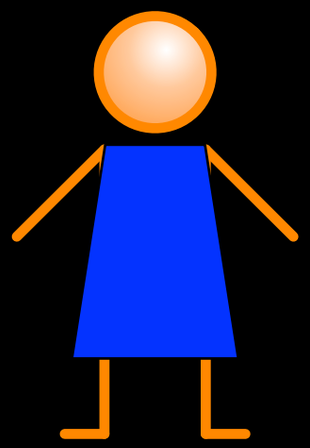
(m,k)

…

Problem: Alice thinks Bob received the message, but he didn't

…

Solution: Keys for Everyone



“My key is k1.
What’s yours?”

(k2,k1)

(m,k2)

…

…

Algorithm with Keys

For every pair of users (say Alice and Bob):

If Alice has a message m for Bob:

ECC, AMD encoded

Alice asks Bob for a key, and sends her key k1 along.

Bob replies with his key k2, along with k1'.

If k1'=k1, Alice sends (m,k2) to Bob. Else, she starts over.

If received correctly, Bob is done and remains silent. Else he sends noise.

If noise, Alice starts over. Else, she is done.

How long should keys be?

If key length is logarithmic in L, T and n, then adversary can never guess a key

Cost?

Say m is always 1 bit

Then get logarithmic blowup

Problem: Don’t know L and T!

Solution: Increase key size slowly over time

Probability of correct guess decreases exponentially with key length

Thus, can keep key bits small

Same holds for AMD encoding strength!

Problem 1: Forging Silence

Adversary “silences out” communication

Solution: Make silence hard to forge

Silence

A b-bit string is **silence** if it has less than b/3 bit alternations

Append logarithmic number of random bits to each word

Pr (“silencing” these random bits) is exponentially small

Modified Algorithm - Add random bits

For every pair of users (say Alice and Bob):

If Alice has a message m for Bob:

ECC, AMD encoded. Appended with random bits.

Alice asks Bob for a key, and sends her key k1 along.

Bob replies with his key k2 along with k1'.

If k1'=k1, Alice sends (m,k2) to Bob. Else, she starts over.

If received correctly, Bob is done and remains silent. Else he sends noise.

If noise, Alice starts over. Else, she is done.

Problem 2: Duplicate messages

Bob thinks that a resend is a fresh message

Solution: Message parity bit

Append j-th message from Alice to Bob with (j mod 2)

This bit remains same throughout resends

Final Algorithm

For every pair of users (say Alice and Bob):

If Alice has a message m for Bob:

ECC, AMD encoded. Appended with random bits.

Alice asks Bob for a key, and sends her key k1 along.

Bob replies with his key k2 along with k1'.

If k1'=k1, Alice sends ((m,b),k2) to Bob. Else, she starts over.

If received correctly, Bob is done and remains silent. Interprets m as a resend if b is unchanged, else a fresh message.

Else he sends noise.

If noise, Alice starts over. Else, she is done.

Analysis Sketch

Hamming

“Shortly before the first field test … a man asked me to check some arithmetic he had done, and I agreed, thinking to fob it off on some subordinate. When I asked what it was, he said, ‘It is the probability that the test bomb will ignite the whole atmosphere.’ I decided I would check it myself!”

Correctness of Simulation

Alice initiates send of m to Bob in π':

(1) She finishes sending m in some round

(2) Either Bob receives m, or he is in a termination state in π

For every message m that Bob receives, Alice must have sent m

Probability of success

Round Sizes

Each round is 4 words long

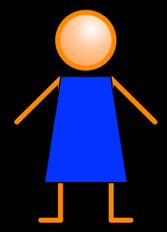
In round r,

Words are of length O(log (nr/δ))

=> Round size is O(log (nr/δ))

Bob updates his round size similarly

Failure Events



Failure of AMD Codes Sushi? ——————> Smoothie?

Forging Silence Sushi? ——————> …

Impersonation … ——————> Smoothie?

Failure Events can happen:

1) On any of the 4 words in a given round

2) On any of the channels between n^2 pairs of users in a given round

3) In any round before termination

Failure of AMD Codes

In round r, set AMD encoding strength to $(n \pi r)^{-2} \delta/2$

Then,

$$\Pr(F) \leq \sum_{1 \leq i,j \leq n} \sum_{1 \leq k \leq 4} \sum_{r \geq 1} (n \pi r)^{-2} \delta/2 \leq \delta/3$$

Forging Silence

In round r, number of random bits in each word is 38 log ($2 \pi n r / \sqrt{\delta}$)

Probability of converting to silence $\leq (n \pi r)^{-2} \delta/4$

Thus,

$\Pr(F) \leq \sum_{1 \leq i,j \leq n} \sum_{1 \leq k \leq 4} \sum_{r \geq 1} (n \pi r)^{-2} \delta/4 \leq \delta/6 \leq \delta/3$

Impersonation

In round r, keys in each word have length
$2 \log (4 \pi n r / \sqrt{\delta})$

So probability of guessing a key $\leq (n \pi r)^{-2} \delta/16$

Thus,

$\Pr (F) \leq \sum_{1 \leq i,j \leq n} \sum_{1 \leq k \leq 4} \sum_{r \geq 1} (n \pi r)^{-2} \delta/16 \leq \delta/24 \leq \delta/3$

Failure probability

Union bound over the three failure events

Total failure probability
$\leq \delta/3 + \delta/3 + \delta/3 = \delta$

Number of bits sent

Recall :

To corrupt a message of size ℓ, adversary spends $\Theta(\ell)$.

We consider only the rounds that are successful

Additional cost will be O(T)

Number of bits sent

In round $r \geq 1$, four codewords each of length $O(\log (nr/\delta))$

$O(\sum \log (nr/\delta))$ bits sent over all rounds

How many rounds?

Number of rounds

Let t(r) be the time at which round r begins.

$t(r+1) = t(r) + O(\log (nr/\delta))$

which gives $t(r) = O(r \log (nr/\delta))$

Number of rounds

Number of rounds = number of successful rounds (L) + number of corrupted rounds (say x)

Round size increases with index and adversary pays Θ of what we pay, so $t(x) \leq T$.

This gives $x = O(T/ \log (nT/\delta))$.

So, total number of rounds = $L + O(T/ \log (nT/\delta))$

Number of bits sent

Recall $O(\Sigma \log (nr/d))$ bits sent over all rounds

$O(\Sigma \log (nr/d))$ for $r = 1$ to $L + O(T/\log (nT/\delta))$

So total bits sent is $O(L \log (n(L+T)/\delta) + T)$

Backup slides

Larger messages in π

Let α be the average message size in π

We show L’ = O(L/α log (n(L+T)/δ) + T)

Probability of success is at least 1 - δ

If α is large, then L’ = O(L+T)

Constant above optimal!

Idea: Break longer message into smaller “chunks”

Question: What is a good chunk size?

Solution: Chunk size in round r

$= O(\log (nr/\delta))$

Constant blowup in word size.

Our Algorithm - Arbitrary message size in π

For every pair of users (say Alice and Bob):

If Alice has a message m for Bob:

While m is not completely sent :

m' <—- next O(log (nr/δ)) bits of m

ECC, AMD encoded. Appended with random bits.

Alice asks Bob for a key, and sends her key k1 along.

Bob replies with his key k2 along with k1'.

If k1'=k1, Alice sends ((m',b),k2) to Bob. Else, she starts over.

If received correctly, Bob is done and remains silent. Interprets m as a resend if b is unchanged, else a fresh message.

Else he sends noise.

If noise, Alice sends m' again. Else, she is done.

Since the union bound in probability of failure remains unchanged, the probability of failure is still at most δ.

$$\Pr(F) \leq \sum_{1 \leq i,j \leq n} \sum_{1 \leq k \leq 4} \sum_{r \geq 1} \Pr(F_{i,j,k,r})$$

Number of bits sent

Step 1 : Compute maximum word size in π’

Step 2 : Compute average per-bit blowup

Step 3 : Compute average number of bits sent

Maximum word size in π’

Index of last round = L + O(T / log (nT/δ))

Word size in round r = O(log (nr/δ))

Thus, maximum word size = O(log n(L+T)/δ)

Average per-bit blowup

w_{max} = max word length = O(log n(L+T)/δ)

Maximum number of uncoded bits of π in a word = log (nr/δ)

Let c = w_{max} / log (nr/δ) be a constant.

Assume π has μ messages of lengths L_1, …, Lμ.

Since $\sum_{1 \le i \le \mu} L_i = L$, it follows that μ = L/α

Average per-bit blowup

Let $b_{i,t}$ = t-th bit of L_i

Blowup associated with $b_{i,t} = 4\ w_{max} / L_i$

If $L_i > \log (n(L+T)/\delta)$, then this blowup is at most 8c.

Thus, blowup for $b_{i,t} = \max (8c, 4\ w_{max} / L_i)$

Average per-bit blowup

We now bound the expected value of max (8c, $4w_{max} / L_i$) over choosing i with probability proportional to L_i.

Note : max (8c, $4w_{max} / L_i$) is convex in L_i.

Thus, expectation is maximized when all but one of the L_i's is 1.

Average per-bit blowup

Thus, we have μ -1 single bit messages. Hence, size of biggest message = L(α -1)/α + 1

Thus, expected blowup per bit = O(Blowup associated with $b_{i,t}$ = O(w_{max} / α)

which is the same as O(1/α log n(L+T)/δ)

Expected number of bits

Expected per-bit blowup in π' = $O(1/\alpha \log n(L+T)/\delta)$

Maximum number of bits in π = L

Thus, expected number of bits in π' = $O(L/\alpha \log n(L+T)/\delta)$ + (cost of corrupted rounds = $O(T)$)

Final result

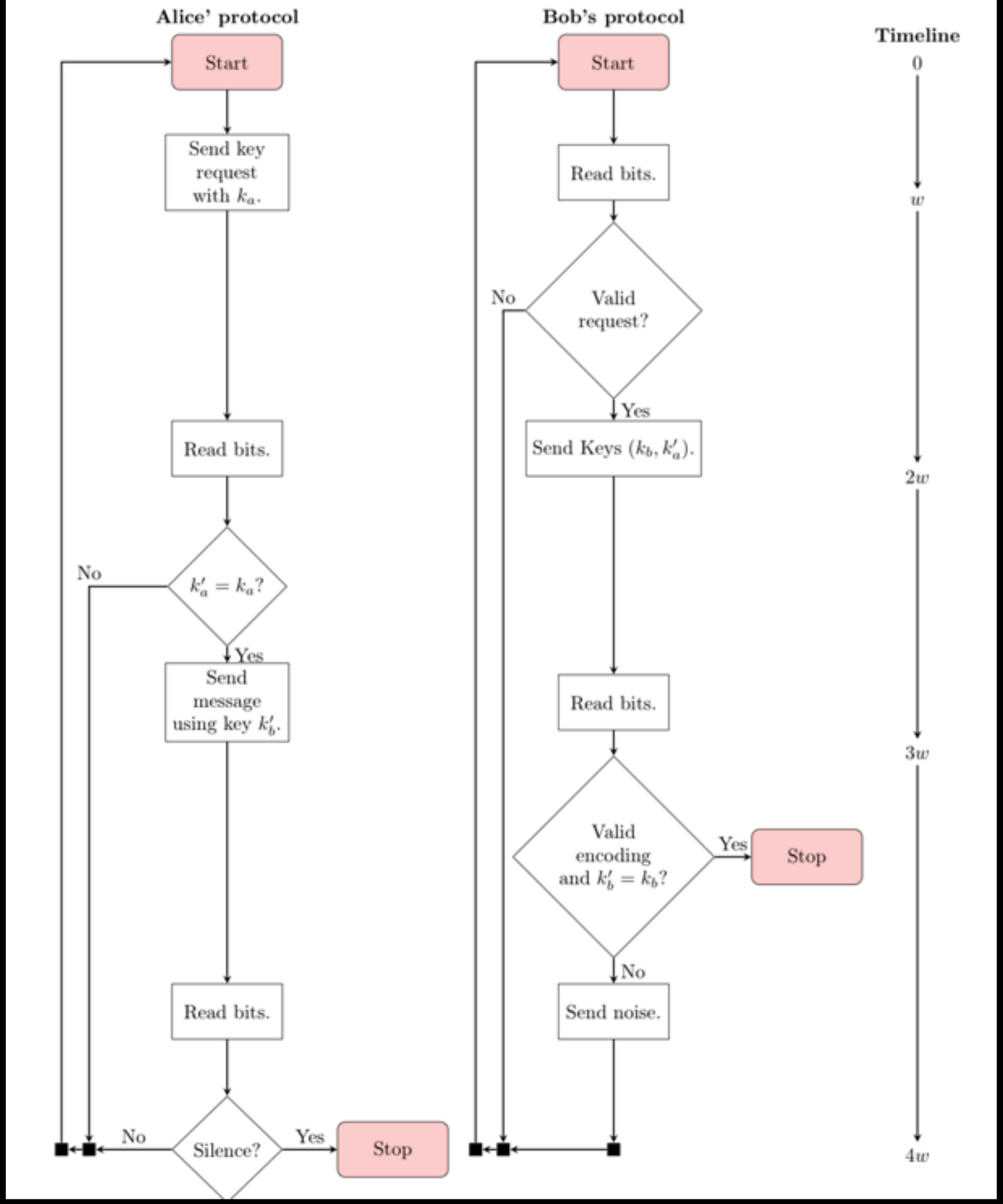
Let α be the average message size in π. We do not assume α is known apriori.

Then, for any given δ ∈ (0,1)

$$L' = O(L/\alpha \log (n(L+T)/\delta) + T)$$

Probability of success is at least 1 - δ

If α is large, then L’ = O(L+T) <—— Constant above optimal!


Alice' protocol
Start
Send key request with k_a.
Read bits.
No
$k'_a = k_a$?
Yes
Send message using key k'_b.
Read bits.
No
Silence?
Yes
Stop
Bob's protocol
Start
Read bits.
No
Valid request?
Yes
Send Keys (k_b, k'_a).
Read bits.
Valid encoding and $k'_b = k_b$?
Yes
Stop
No
Send noise.
Timeline
0
w
$2w$
$3w$
$4w$

Algorithm 1 Message exchange algorithm for the sender.

```
1:  procedure SEND-MESSAGE(m, b, v)      ▷ b is the parity bit for the content m to be sent to user v.
2:      while true do
3:          w_t ← CODEWORD-LENGTH(t)                          ▷ t is the current timestep.
4:          Generate random key k_a.
5:          Send E_t(KEY?, k_a).
6:          M_1 ← w_t bits on the channel from v.
7:          if IsSilence(M_1) then                            ▷ Assume v has already terminated.
8:              Stay silent for 2w_t time steps.
9:              return
10:         else
11:             if D_t(M_1) ≠ (*, k_a) then
12:                 Stay silent for 2w_t time steps.
13:             else
14:                 (k_b, k_a) ← D_t(M_1)
15:                 Send E_t((m, b), k_b).
16:                 M_2 ← w_t bits on the channel from v.
17:                 if IsSilence(M_2) then return
```

Algorithm 2 Message exchange algorithm for the receiver.

procedure RECEIVE-MESSAGE(v) ▷ Run on the channel from user v.
  $w_t \leftarrow$ CODEWORD-LENGTH(t) ▷ t is the current timestep.
  $M_1' \leftarrow w_t$ bits on the channel from v.
  if `IsSilence` (M_1') **then**
    Stay silent for $3w_t$ time steps.
  else
    if $\mathscr{D}_t(M_1') \neq (\texttt{KEY?}, *)$ **then**
      Send noise for w_t time steps.
      Stay silent for $2w_t$ time steps.
    else
      $(\texttt{KEY?}, k_a) \leftarrow \mathscr{D}_t(M_1')$.
      Generate random key k_b.
      Send $\mathscr{E}_t(k_b, k_a)$.
      $M_2' \leftarrow w_t$ bits on the channel from v.
      if $\mathscr{D}_t(M_1') \neq (*, k_b)$ **then**
        Send noise for w_t time steps.
      else
        $((m, b), k_b) \leftarrow \mathscr{D}_t(M_1')$
        if b different from last message **then**
          Record the message m from v.
        Stay silent for w_t time steps.

Algorithm 3 Protocol π'.

1: **for** each user u, in parallel **do**
2: Initialize π_u.
3: **while** π_u is not done, in parallel **do**
4: For each neighbor v, run RECEIVE-MESSAGE(v) on the channel from v.
5: **if** u has a message m for user v in π_u **then**
6: $b \leftarrow j \pmod 2$ where message m is the j^{th} message sent from u to v.
7: Run SEND-MESSAGE(m, b, v) on the channel to v.
8: **return**

Dani et al. ‘15

2 players

Private Channel

An adversary flips T bits

T is unknown in advance

$L' = L + O(\sqrt{(L(T+1)\log L)} + T)$

Succeeds with high probability in L