Truth, Lies and Randomness

Jared Saia

Joint with Valerie King
Byzantine Agreement

Each processor starts with a bit

Goal: 1) all good procs output the same bit; and 2) this bit equals an input bit of a good proc

$\text{t} = \# \text{ bad procs controlled by an adversary}$
Group Decisions

Periodically, components unite in a decision

Idea: components vote. Problem: Who counts the votes?
Model

• Asynchronous

• Full Information

• Adaptive

Adversary takes over processors at any time, up to \( t \) total

Adversary knows all

Adversary controls message ordering
Applications

• Peer-to-peer networks
  “These replicas cooperate with one another in a Byzantine agreement protocol to choose the final commit order for updates.” [KBCCEGGRWWWZ ’00]

• Rule Enforcement
  “... requiring the manager set to perform a Byzantine agreement protocol” [NWD ’03]

• Game Theory (Mediators)
  “deep connections between implementing mediators and various agreement problems, such as Byzantine agreement” [ADH ’08]
Applications

• Peer-to-peer networks
  “These replicas cooperate with one another in a Byzantine agreement protocol to choose the final commit order for updates.” [KBCCEGGRWWWZ ’00]

• Rule Enforcement
  “… requiring the manager set to perform a Byzantine agreement protocol” [NWD ’03]

• Game Theory (Mediators)
  “deep connections between implementing mediators and various agreement problems, such as Byzantine agreement” [ADH ’08]

Also: Databases, State Machine Replication, Secure Multiparty Computation, Sensor Networks, Cloud Computing, Control systems, etc.
Previous Work

• **Time**: maximum length of any chain of messages

• [Ben-Or ’83] gave first randomized algorithm to solve BA in this model

• [FLP ’85] showed BA impossible for deterministic algorithms even when \( t=1 \)

• Ben-Or’s algorithm is exponential expected time
Our Result

- Our algorithm: expected polynomial time; and expected polynomial computation

Theorem 1. Let $n$ be the number of processors, then there is a $t = \Theta(n)$ such that the following holds in the asynchronous message passing model with an adaptive, full-information adversary that controls up to $t$ processors. Byzantine Agreement can be solved in expected $O(n^3)$ communication time, expected polynomial computation time per processor, and expected polynomial bits of communication.
Ben-Or’s algorithm

- Ben-Or’s algorithm consists of repeated iterations.
- It uses private random bits which create a fair global coin with probability $1/(2^n)$ in each iteration.
- For each iteration there is a correct direction so that if there is a global coin and it is in this direction, agreement is reached.
Ben-Or’s algorithm

• Ben-Or’s algorithm consists of repeated iterations

• It uses private random bits which create a fair global coin with probability $1/(2^n)$ in each iteration.

• For each iteration there is a correct direction so that if there is a global coin and it is in this direction, agreement is reached.

**Our goal:** Create a routine that results in a fair global coin after no more than polynomial number of iterations using private random bits.
Ben-Or’s Algorithm Overview
Ben-Or with Global Coin, GC

Set your vote to input bit

Repeat clogn times:

Send your vote to everyone

Let $fraction$ be fraction of votes received for majority bit

If $fraction \geq 2/3$, set vote to majority bit; else set vote to GC

Output your vote
fraction >= 2/3. I’m voting for 0.
fraction >= 2/3. I’m voting for 0.
fraction \geq \frac{2}{3}. I'm voting for 0.

fraction < \frac{2}{3}. I'm checking the coin.
Note: The procs with $\text{fraction} \geq 2/3$ will all change vote to same value

fraction < 2/3. I’m checking the coin.

fraction $\geq 2/3$. I’m voting for 0.
fraction >= 2/3. I'm voting for 0.

fraction < 2/3. I'm checking the coin.
All-to-all fraction $\geq 2/3$. I'm voting for 0.

fraction $< 2/3$. I'm checking the coin.

fraction $\geq 2/3$. I'm voting for 0.

Probability $1/2$ that both groups change vote to the same value
All-to-all fraction $\geq 2/3$. I'm voting for 0.

fraction $< 2/3$. I'm checking the coin.

fraction $\geq 2/3$. I'm voting for 0.

Probability $1/2$ that both groups change vote to the same value.

Once this happens, all votes of good procs will be equal evermore.
Probability $1/2$ that both groups change vote to the same value

Once this happens, all votes of good procs will be equal evermore

Prob of failure $= (1/2)^{clog n}$
Probability 1/2 that both groups change vote to the same value

Once this happens, all votes of good procs will be equal evermore

\[
\text{Prob of failure} = \left(\frac{1}{2}\right)^{\log_2 n} = \frac{1}{n^c}
\]
Probability 1/2 that both groups change vote to the same value

Once this happens, all votes of good procs will be equal evermore

\[ \text{Prob of failure} = \left(\frac{1}{2}\right)^{c \log n} = \frac{1}{n^c} \]

\[ \text{Prob of success} = 1 - \frac{1}{n^c} \]
Probability $1/2$ that both groups change vote to the same value

Once this happens, all votes of good procs will be equal evermore

\[
\text{Prob of failure} = (1/2)^{c \log n} = 1/n^c
\]

\[
\text{Prob of success} = 1 - 1/n^c
\]
Asynchronous Agreement w/ Fail-stop faults (e.g. [AC ’08])

• Flip $n^2$ coins. Let heads be +1 and tails be -1

• Let $\text{sum}$ be the sum of all coins

• **Problem:** Asynchronicity can alter sum value perceived by a processor by $\sim n$

• **Idea:** With constant probability, $|\text{sum}| \geq 2n$

• If $|\text{dev}| \geq 2n$: 1) direction of sum gives a fair global coin; 2) direction of sum is robust to asynchronicity
Adapting to Byzantine

- **New Problem:** Bad procs may lie about their coinflips
- Idea: Blacklist lying procs
- Q: Can we determine which procs lie by studying deviation of coinflips?
Key Idea

• With constant probability, the direction of good procs will be in the correct direction and large enough for Ben-Or’s alg to succeed

• Bad procs need to generate bad dev. in the opposite direction of equal magnitude to foil this good event

• Problem (for bad): There are fewer bad
Deviation Probabilities

![Graph showing deviation probabilities with two curves. The y-axis represents probability and the x-axis represents the range of deviations. The graph includes a label for 'probability' and 'sum'.]
Deviation Probabilities

probability

t procs

n-t procs

sum
Deviation Probabilities

sum

probability

n-t procs

t procs

prob n-t procs have dev $\geq$ kn

sum
Deviation Probabilities

- prob \( n-t \) procs have dev \( \geq kn \)
- prob \( t \) good procs have dev \( \leq -kn \)
- sum
Deviation Probabilities

sum

probability

observed prob for t bad procs

prob t good procs have dev ≤ -kn

prob n-t procs have dev ≥ kn

t procs

n-t procs

sum
List of Issues

1. Equivocation: Bad processors can send different coinflip outcomes to different processors

2. Missing coins: Adversary can delay messages. Implies different processors can receive (somewhat) different sets of coinflips

3. Bias: Bad processors can bias their coinflips
List of Issues

1. Equivocation: Bad processors can send different coinflip outcomes to different processors

2. Missing coins: Adversary can delay messages. Implies different processors can receive (somewhat) different sets of coinflips

• Bracha’s Reliable Broadcast

• Ensures: If a good processor receives a message from a bad proc, q, all other good processors that receive a message from q will receive the same message
List of Issues

1. Equivocation: Bad processors can send different coinflip outcomes to different processors

2. Missing coins: Adversary can delay messages. Implies different processors can receive (somewhat) different sets of coinflips

• We show: At least $n(n-2t)$ common coins

• No more than $2t$ coins from good processors, no more than 2 per processor that are not common

• The common coins are known to $n-4t$ good processors
List of Issues

Ignore in this talk. See paper for details

1. Equivocation: Bad processors can send different coinflip outcomes to different processors

2. Missing coins: Adversary can delay messages. Implies different processors can receive (somewhat) different sets of coinflips

3. Bias: Bad processors can bias their coinflips

Focus of this talk
Deviation

• We assume all coinflips are either +1 or -1

• The **deviation** of \( p \) in an iteration is the absolute value of the sum of \( p \)'s coinflips

• The **direction** of \( p \) in an iteration is the sign of the sum of \( p \)'s coinflips

• **idev**\((S,i)\): absolute value of all coinflips sent by procs in \( S \) in iteration \( i \)
Iterations and Epochs

- In each iteration, we run modified Ben-Or
- There are $cn$ iterations in an epoch
- In each epoch, we expect a constant fraction of iterations to be good i.e. dev. of good procs is $\geq B$ in right direction ($B = c'n$ for a fixed $c'$)
Iterations and Epochs

- In each iteration, we run modified Ben-Or
- There are $cn$ iterations in an epoch
- In each epoch, we expect a constant fraction of iterations to be good i.e. dev. of good procs is $\geq B$ in right direction ($B = c'n$ for a fixed $c'$)
- In a good iteration, bad procs must have dev. $\geq B/2$
- (Remaining “good” deviation undone by scheduler)
Key fact

In every non-terminating epoch \( e \), there is a set of \( c_2 n \) iterations \( I_e \) and a set of \( \leq t \) processors \( B_e \), such that for all \( i \) in \( I_e \):

\[
\text{idev}(B_e, i) \geq B/2
\]
Perspective

• In reality, different processors may receive different sets of coinflips

• In our paper, we have subscripts for terms like $i_{dev}$ to indicate the $n$ different perspectives

• In this talk, we omit subscripts and assume perspective of a processor that receives all coinflips
Spectral Blacklisting
Bipartite Graph
Bipartite Graph

edge between each proc p and each iter i with weight = dvtn of p in iter i
Bipartite Graph

|R| = cn

\(|L| = \frac{n}{2}\) weight

\(B_e\)

\(I_e\)

edge between each proc \(p\) and each iter \(i\) with weight = \(dvtn\) of \(p\) in iter \(i\)
Bipartite Graph

\[ |R| = cn \]
\[ c^2 n \]
\[ \frac{n}{2} \text{ weight} \]

\[ |L| = n \text{ processors iterations} \]

- \( B_e \) (edge between each proc \( p \) and each iter \( i \) with weight = dvtn of \( p \) in iter \( i \))

\[ \frac{n}{10} \]
Matrix Approach

- $M$ is a $m$ by $n$ matrix for this graph
- $M(i,j) =$ devtn in iteration $i$ of processor $j$
(D)etector/(N)eutralizer Game
(D)etector/(N)eutralizer
Game

1. N claims columns, provided total claimed over game $\leq t$
(D)etector/(N)eutralizer Game

1. N claims columns, provided total claimed over game $\leq t$

2. Entries in unclaimed columns set to sum of n indep coinflips
(D)etector/(N)eutralizer Game

1. N claims columns, provided total claimed over game $\leq t$

2. Entries in unclaimed columns set to sum of n indep coinflips

3. Each row selected indep. with prob. 1/2
(D)eveloper/(N)eutralizer Game

1. N claims columns, provided total claimed over game $\leq t$

2. Entries in unclaimed columns set to sum of n indep coinflips

3. Each row selected indep. with prob. 1/2

4. N sets all entries in its columns
(D)etector/(N)eutralizer Game

1. N claims columns, provided total claimed over game $\leq t$

2. Entries in unclaimed columns set to sum of n indep coinflips

3. Each row selected indep. with prob. 1/2

4. N sets all entries in its columns

5. D sees matrix & may remove columns provided total removed over game $\leq 2t$
(D)etector/(N)eutralizer Game

1. N claims columns, provided total claimed over game \( \leq t \)
2. Entries in unclaimed columns set to sum of \( n \) indep coinflips
3. Each row selected indep. with prob. \( 1/2 \)
4. N sets all entries in its columns
5. D sees matrix & may remove columns provided total removed over game \( \leq 2t \)

N’s goal: Deviation of all “selected” rows \( \leq 2n \)
D wins if N fails in its goal
(D)etector/(N)eutralizer
Game

1. N claims columns, provided total claimed over game $\leq t$
2. Entries in unclaimed columns set to sum of n indep coinflips
3. Each row selected indep. with prob. 1/2
4. N sets all entries in its columns
5. D sees matrix & may remove columns provided total removed over game $\leq 2t$

N’s goal: Deviation of all “selected” rows $\leq 2n$
D wins if N fails in its goal

Our result: Win for D in expected $O(n)$ iterations
Related Work (Spectral)

- Page Rank
- Eigentrust
- Hidden Clique
Page Rank [PBMW ’99]

- Google’s $300 billion “secret sauce”
- \( M \) is a stochastic matrix, representing a random walk over the web link graph
- \( r \) is top right eigenvector of \( M \) (and stationary distribution of \( M \)’s walk)
- For a web page, \( i \), \( r[i] \) = “authority” of \( i \)
Eigentrust [KSG '03]

- $M$ is a matrix s.t. $M(i,j)$ represents amount which party $i$ trusts party $j$
- $r$ is top right eigenvector of $M$
- For a party, $i$, $r[i] = \text{“trustworthiness” of } i$
- Party $i$ is trustworthy if it is trusted by parties that are themselves trustworthy
Differences

• Eigentrust and PageRank: Want to identify good players based on **feedback from other players**

• D/N Game: Want to identify bad players based on **deviation from random coinflips**
Hidden Clique
Hidden Clique

- The problem
  - A random $G(n, 1/2)$ graph is chosen
  - A $k$-clique is randomly placed in $G$
- [AKS ’98] give an algorithm for $k = \sqrt{n}$
  1. $v$ is second eigenvector of adj. matrix of $G$
  2. $W$ is top $k$ vertices sorted by abs. value in $v$
  3. Returns all nodes with $3k/4$ neighbors in $W$
Differences

• Hidden Clique: Matrix entries are 0 and 1; Want to find submatrix that is all 1’s

• D/N Game: Matrix entries in [-n,+n]. Want to find submatrix where sum of each row has high absolute value
Our Approach
• Each proc has a “suspicion” value, initially 0
• Use $r$, the top right singular vector of $M$
• Processor i’s “suspicion” value increases by $(r[i])^2$ in each epoch
• Remove processor i once its suspicion value reaches 1
Matrix

• $M$ is a $m$ by $n$ matrix for this graph
• $M(i,j) =$ devtn in iteration $i$ of processor $j$
• $M_g$ is good columns of $M$
• $M_b$ is bad columns of $M$
• Assume $M = [M_b \, M_g]$
$\text{Mg}$
Fact 1: Whp, $|M_g| \leq 5(n(m+n))^{1/2}$

- $M_g$ is a random matrix
- Each entry is an independent r.v. with expectation 0; s.d. = $\sqrt{n}$; and range $[-k,k]$ where $k \sim n^{1/2} \log n$
- Fact 1 follows from Theorem 3 in [AS '07]
$|M_b|$
|\mathbf{M}_b|$

**Fact 2:** $|\mathbf{M}_b| \geq (mn)^{1/2} / (2c_1)$ (where $t = c_1 n$)
\[ |M_b| \]

**Fact 2:** \( |M_b| \geq (mn)^{1/2} / (2c_1) \) (where \( t = c_1 n \))

- \( \mathbf{x} \) is a unit vector with all values \( 1/t^{1/2} \)
\[ |M_b| \]

**Fact 2:** \( |M_b| \geq (mn)^{1/2} / (2c_1) \) (where \( t = c_1 n \))

- \( \mathbf{x} \) is a unit vector with all values \( 1/t^{1/2} \)
- \( \mathbf{y} \) is a unit vector with entries \( \pm 1/(m/10)^{1/2} \) for the \( m/10 \) good iterations and 0 everywhere else (sign of non-zero entries is direction of bad deviation)
\[ |M_b| \]

**Fact 2:** \[ |M_b| \geq (mn)^{1/2} / (2c_1) \] (where \( t = c_1 n \))

- \( \mathbf{x} \) is a unit vector with all values \( 1/t^{1/2} \)
- \( \mathbf{y} \) is a unit vector with entries \( \pm 1/(m/10)^{1/2} \) for the \( m/10 \) good iterations and 0 everywhere else (sign of non-zero entries is direction of bad deviation)
- Then \( \mathbf{y}^t M_b \mathbf{x} \geq (mn/20)/(mt/10)^{1/2} \geq (mn)^{1/2} / (2c_1) \)
\[ |M_b| \geq C |M_g| \]

- **Fact 3**: For any constant C, for t chosen sufficiently small, whp, \( |M_b| \geq C |M_g| \)
  
  - \( m+n \leq 4m \) (can ensure for m sufficiently large)
  
  - Thus, whp, \( |M_g| \leq 20 (mn)^{1/2} \)
  
  - By Fact 2, \( |M_b| \geq (mn)^{1/2} / (2c_1) \)
  
  - Letting \( t = c_1 n \) gives the result
Let \( \mathbf{r} \) be the top right singular vector of \( \mathbf{M} \).

Let \( \mathbf{r}_b \) be the vector such that \( \mathbf{r}_b[i] = \mathbf{r}[i] \) for \( 1 \leq i \leq t \) and all other entries are 0.

Let \( \mathbf{r}_g \) be the vector such that \( \mathbf{r}_g[i] = \mathbf{r}[i] \) for \( t+1 \leq i \leq n \) and all other entries are 0.

**Lemma 1:** Whp, \( |\mathbf{r}_g|^2 < |\mathbf{r}_b|^2 / 2 \).
Lemma 1

Lemma 1: Whp, $|r_g|^2 < |r_b|^2 / 2$

Proof: Assume not. Then $|r_b|^2 \leq 2/3$, so $|r_b| \leq \sqrt{(2/3)}$
Lemma 1

Lemma 1: Whp, $|r_g|^2 < |r_b|^2 / 2$

Proof: Assume not. Then $|r_b|^2 \leq 2/3$, so $|r_b| \leq \sqrt{(2/3)}$

\[
\begin{align*}
|M_B| &\leq |M| \\
&= \ell^T (M r) \\
&\leq |\ell| |M r| \\
&\leq |M_B| |r_b| + |M_G| |r_g|
\end{align*}
\]
Lemma 1

**Lemma 1:** Whp, $|r_g|^2 < |r_b|^2 / 2$

**Proof:** Assume not. Then $|r_b|^2 \leq 2/3$, so $|r_b| \leq \sqrt{(2/3)}$

\[
|MB| \leq |M| \\
= \ell^T (Mr) \\
\leq |\ell||Mr| \\
\leq |MB||r_b| + |MG||r_g| \\
\leq |MB|(|r_b| + (1/C)|r_g|) \\
\leq |MB|(\sqrt{2/3} + 1/C) \\
< |MB|
\]
Lemma 1

**Lemma 1:** Whp, $|r_g|^2 < |r_b|^2 / 2$

**Proof:** Assume not. Then $|r_b|^2 \leq 2/3$, so $|r_b| \leq \sqrt{(2/3)}$

\[
|M_B| \leq |M| \\
= \ell^T (Mr) \\
\leq |\ell| |Mr| \\
\leq |M_B||r_b| + |M_G||r_g| \\
\leq |M_B|(|r_b| + (1/C)|r_g|) \\
\leq |M_B|(|r_b| + (\sqrt{2/3} + 1/C) \\
< |M_B|
\]

where the last line holds if $C \geq 5.45$, or $t \leq .004n$
Algorithm Sketch
Algorithm Sketch

- Repeat until reaching agreement
Algorithm Sketch

• Repeat until reaching agreement
  1. Run an epoch. Let $M$ be the devtn matrix for that epoch
Algorithm Sketch

- Repeat until reaching agreement
  1. Run an epoch. Let $M$ be the devtn matrix for that epoch
  2. Compute the right singular vector, $r$, of $M$
Algorithm Sketch

- Repeat until reaching agreement
  1. Run an epoch. Let $M$ be the deviation matrix for that epoch
  2. Compute the right singular vector, $\mathbf{r}$, of $M$
  3. Increase $\text{cumdev}$ value of each proc $i$ by $\mathbf{r}[i]^2$
Algorithm Sketch

• Repeat until reaching agreement

1. Run an epoch. Let $M$ be the devtn matrix for that epoch

2. Compute the right singular vector, $r$, of $M$

3. Increase \texttt{cumdev} value of each proc $i$ by $r[i]^2$

4. Blacklist a proc when its \texttt{cumdev} value is 1 or more
Caveat

• Problem: Some good processors may not receive the (biased) coinflips of the bad processors in a given epoch
Caveat

- Problem: Some good processors may not receive the (biased) coinflips of the bad processors in a given epoch

- If $|M| \leq \frac{(mn)^{1/2}}{(2c_1)}$ then don’t do cumdev updates

- We show: if there is no agreement, a linear number of good procs will have $|M| \leq \frac{(mn)^{1/2}}{(2c_1)}$
Conclusion

• First expected fully polynomial time algorithm for traditional Byzantine agreement

• Previous best algorithm (Ben-or’s) was expected exponential time

• New technique: design of algorithms that force attackers into statistically deviant behavior that is detectable
Open Problems

• Can we improve latency, resilience, and bandwidth
• Can we improve other randomized algorithms by forcing bad procs into detectably deviant behavior?
• Can we design a general spectral approach for these types of problems?
Questions
Reliable Broadcast (Bracha)

• All coinflip values sent using reliable broadcast

• Ensures if a message is “received” by a good proc, same message is eventually “received” by all procs

• Prevents equivocation

• Doesn’t solve BA
Common Coins

- There are at least $n(n-2t)$ common coins and no more than $2t$ coins from good processors, no more than 2 per processor that are not common.

- The common coins are known to $n-4t$ good processors.
Technique: Brute Force
cumdev(p)

- \textbf{cumdev(p)} starts at 0
- In each epoch, for every proc p in \(B_e\), \textbf{cumdev(p) += \``deviation of each processor p in direction of B_e, summed over all iterations I_e\''}
- We blacklist any proc p when \textbf{cumdev(p)} exceeds \(n^{1.5} \ln n\)
GLOBAL-COIN

• Each processor sends out \( n \) coinflips

• Each processor lets \textbf{dev}\ be the sum of coinflips of all processors that have not been blacklisted

• Each processor sets its value for the global coin to this value
MODIFIED BEN-OR

- Run Ben-Or’s algorithm using GLOBAL-COIN to determine a global coinflip
POLY-EXP-TIME-BA

• Run an epoch (cn iterations of MODIFIED BEN-OR)

• If the epoch fails, find sets $B_e$ and $I_e$

• Increase $\text{cumdev}$ scores for every proc in $B_e$ by amount they contributed to dev in each iter in $I_e$

• Blacklist any proc $p$ when $\text{cumdev}(p)$ exceeds $n^{1.5}(\ln n)$
POLY-EXP-TIME-BA

• Run an epoch (cn iterations of MODIFIED BEN-OR)

• If the epoch fails, find sets $B_e$ and $I_e$

• Increase $\text{cumdev}$ scores for every proc in $B_e$ by amount they contributed to dev in each iter in $I_e$

• Blacklist any proc $p$ when $\text{cumdev}(p)$ exceeds $n^{1.5}(\ln n)$

• (We also blacklist any proc which has highly unlikely dev in any iteration ($> c' n^{0.5} \log n$))
cumdev facts

• Let $X$ be the sum of $\text{cumdev}(p)$ for all procs $p$

• Fact 1: $X$ is upper bounded by $\sim n^{2.5}(\ln n)$

• Fact 2: $X$ increases by $(B/2)c_2n \sim n^2$ in every epoch

  Thus there are $\sim n^{5}(\ln n)$ epochs
Lemma 12: Assume the number of blacklisted procs is \( \leq t \). Then in every non-terminating epoch \( e \), there is a set of \( c_2n \) iterations \( I_e \) and a set of \( \leq t \) processors \( B_e \), such that for all \( i \) in \( I_e \):

\[
\text{idev}(B_e,i) \geq B/2
\]
Lemma 14

- **Lemma 14:** The number of blacklisted good procs is no more than $t$ (whp).
Sketch of Lemma 14

- Let GOOD be the set of good procs, BAD be set of bad procs; \( \text{cumdev}(S,e) = \) amount added to \( \text{cumdev} \) for all procs in
Sketch of Lemma 14

- Let GOOD be the set of good procs, BAD be set of bad procs; \( \text{cumdev}(S, e) \) = amount added to \( \text{cumdev} \) for all procs in

- **Lemma 13:** Whp, for any epoch e, \( \text{cumdev}(\text{GOOD}, e) \leq (B/5)c_2n \)
Sketch of Lemma 14

- Let GOOD be the set of good procs, BAD be set of bad procs; $\text{cumdev}(S,e) =$ amount added to $\text{cumdev}$ for all procs in

- **Lemma 13:** Whp, for any epoch $e$, $\text{cumdev}(\text{GOOD}, e) \leq (B/5)c_2n$

- Fact 2: $X$ increases by $(B/2)c_2n$ in every epoch

- Thus: $\text{cumdev}(\text{BAD} \cap B_e, e) \geq (3 B/10)c_2n$
Sketch of Lemma 14

- $\text{cumdev}(\text{GOOD}, e)$ increases by $\leq \left(\frac{2B}{10}\right)c_2n$

- $\text{cumdev}(\text{BAD} \cap B_e, e)$ increases by $\geq \left(\frac{3B}{10}\right)c_2n$

- You can see where this is going!
Proof of Lemma 13

**Lemma 13:** Whp, for any epoch \( e \),
\[
\text{cumdev}(\text{GOOD}, e) \leq \left( \frac{B}{5} \right) c_2 n
\]

- Let \( G = \text{GOOD} \cap B_e \)
- Fix a set \( G \), a set \( I_e \) and a mapping, \( d \), from iterations in \( I_e \) to \( \{-1, +1\} \)
- Let \( Y = \) sum of coins generated by \( G \) in iterations \( I_e \) in directions given by \( d \)
Proof of Lemma 13

Lemma 13: Whp, for any epoch $e$, $\text{cumdev}(\text{GOOD}, e) \leq \left(\frac{B}{5}\right)c_2n$

- Let $G = \text{GOOD} \cap B_e$
- Fix a set $G$, a set $I_e$ and a mapping, $d$, from iterations in $I_e$ to $\{-1, +1\}$
- Let $Y = \text{sum of coins generated by } G \text{ in iterations } I_e \text{ in directions given by } d$
  We use $Y$ to bound amount added to $\text{cumdev}(G)$ in epoch $e$
Bounding $Y$

$$Pr(Y \geq (\beta/6)(c_2n)) \leq e^{-(c_2n\beta/6)^2/2(|G|c_2n^2)}$$

$$\leq e^{-0.026c_2n^2/t}$$
Bounding $Y$

$$Pr(Y \geq (\beta/6)(c_2n)) \leq e^{-(c_2n\beta/6)^2/2(|G|c_2n^2)} \leq e^{-0.026c_2n^2/t}$$

Let $\xi$ be event that $Y \geq (\beta/6)(c_2n)$ for any $G$, $I_e$, and mapping $d$

- $\binom{cn}{c_2n} \leq (ce/c_2)^{c_2n}$ ways to pick the iterations $I_e$

- $\sum_{i=1}^{t} \binom{n}{i} \leq 2^n$ ways to pick the set $G$

- $2^{c_2n}$ ways to pick the mapping $d$
Bounding $Y$

\[
Pr(Y \geq (\beta/6)(c_2n)) \leq e^{-((c_2n\beta/6)^2)/2(|G|c_2n^2)} \\
\leq e^{-0.026c_2n^2/t}
\]

Let $\xi$ be event that $Y \geq (\beta/6)(c_2n)$ for any $G$, $I_e$, and mapping $d$

- \( \binom{cn}{c_2n} \leq \left(\frac{ce}{c_2}\right)^{c_2n} \) ways to pick the iterations $I_e$

- \( \sum_{i=1}^{t} \binom{n}{i} \leq 2^n \) ways to pick the set $G$

- \( 2^{c_2n} \) ways to pick the mapping $d$
Union Bound

\[ Pr(\xi) \leq \left( \frac{ce}{c_2} \right)^{c_2n} 2^n 2^{c_2n} e^{-0.026c_2n^2/t} \]
\[ \leq e^{11c_2n - (0.026c_2n^2/t)} \]
\[ \leq e^{-\Omega(n)} \quad \text{Setting } n/t \geq 500 \]
Union Bound

\[ Pr(\xi) \leq (ce/c_2)^{c_2 n} 2^{2c_2 n} e^{-0.026c_2 n^2/t} \]
\[ \leq e^{11c_2 n - (0.026c_2 n^2/t)} \]
\[ \leq e^{-\Omega(n)} \quad \text{Setting } n/t \geq 500 \]

Another union bound over the polynomial number of epochs and (views of) all good procs completes the proof.