Practical Verified Computation with Streaming Interactive Proofs

Justin Thaler, Harvard University
Joint work with:
Amit Chakrabarti (Dartmouth),
Graham Cormode (AT&T Labs – Research),
Andrew McGregor (UMass Amherst),
Michael Mitzenmacher (Harvard),
and Ke Yi (HKUST)
Outsourcing

• Many applications require outsourcing computation to untrusted service providers.
  - Main motivation: Commercial cloud computing services.
  - Also, weak peripheral devices; fast but faulty co-processors.
  - Volunteer Computing (SETI@home, World Community Grid, etc.)

• User requires a guarantee that the service provider performed the computation correctly.

• One solution: require provider to prove correctness of answer.
Interactive Proofs

• Two Parties: Prover $P$ and Verifier $V$.

• Think of $P$ and powerful, $V$ as weak. $P$ solves a problem, tells $V$ the answer.
  • Then $P$ and $V$ have a conversation.
  • $P$’s goal: convince $V$ the answer is correct.

• Requirements:
  • 1. Completeness: An honest $P$ can convince $V$ she’s telling the truth.
  • 2. Soundness: $V$ will catch a lying $P$ with high probability no matter what $P$ says to try to convince $V$. 
Interactive Proofs

• IPs have revolutionized Complexity Theory in the last 25 years.
  • IP=PSPACE [Shamir 90].
  • PCP Theorem e.g. [AS 98]. Hardness of approximation.
  • Zero Knowledge Proofs.

• But IPs have had very little impact in real delegation scenarios.
  • Why?
  • Not due to lack of applications!
Interactive Proofs

- Old Answer: Most results on IPs dealt with hard problems, needed $P$ to be too powerful.
- But recent constructions focus on “easy” problems (e.g. “Interactive Proofs for Muggles” [GKR 08]).
- Allow $V$ to run very quickly, use small space, so outsourcing is useful even though problems are “easy”.
- $P$ does not need much more time to prove correctness than she does to solve the problem in the first place!

- Shouldn’t these results be useful and exciting to practitioners?
This Talk: New Application of IPs

- To streaming problems: hard because $V$ has to read input in one-pass streaming manner, but (might be) easy if $V$ could store the whole input. [CCM 09/CCMT 12], [CMT 10], [CTY 12], [CMT 12].

- Fits cloud computing well: streaming pass by $V$ can occur while uploading data to cloud.

- $V$ never needs to store entirety of data!
Data Streaming Model

- Stream: m elements from universe of size n
  - e.g., $S = <x_1, x_2, \ldots, x_m> = 3, 5, 3, 7, 5, 4, 8, 7, 5, 4, 8, 6, 3, 2, \ldots$

- Goal: Compute a function of stream, e.g., median, number of distinct elements, frequency moments, heavy hitters.

- Challenge:
  1. Limited working memory, i.e., sublinear(n,m).
  2. Sequential access to adversarially ordered data.
  3. Process each update quickly.

Slide derived from [McGregor 10]
Graph Streams

- \( S = \langle x_1, x_2, \ldots, x_m \rangle; x_i \in [n] \times [n] \)

- \( S \) defines a graph \( G \) on \( n \) vertices.

- Goal: compute properties of \( G \).

- Challenge: subject to usual streaming constraints.
Bad News

- Many graph problems are impossible in standard streaming model (require linear space or many passes over data, even to approximate).

- E.g. Counting triangles, diameter, perfect matching, shortest $s$-$t$ path.

- What to do?
This Talk: Models

• Two models:

  1. One message (Non-interactive) [CCM 09/CCMT 12]: After both observe stream, $P$ sends $V$ an email with the answer, and a proof attached.

  2. Multiple rounds of interaction [CTY 12]: $P$ and $V$ have a conversation after both observe stream.

• Earlier streaming models for verifiable outsourced computation: [TMDSOJ 05], [LYHK 07], [YLHKS 08], [DLN 09], etc.
Costs in Our Models

- Two main costs: words of communication $h$ and $V$’s working memory $v$.
- We refer to $(h, v)$-protocols.
- Other costs: running time, number of messages.
Comparison of Two Models

- Pros of multi-round model:
  1. Exponentially reduces space and communication cost. Often (polylog $n$, polylog $n$) compared to ($\sqrt{n}$, $\sqrt{n}$).
  2. $P$ often much faster than in single-round case.

- Cons of multi-round model:
  1. $P$ must do significant computation after each message. Requires maintaining state between messages; possibly overhead in setting up each computation.
  2. More coordination needed; network latency might be an issue.

- Pros of single round model:
  1. Space and communication still reasonable (< 1 MB).
  2. $P$ can do all computation at once, just send an email with proof attached.
Non-interactive Protocols: A Sampling
Second Frequency Moment

- The second frequency moment of a stream is defined as follows:
  - Let $X$ be the frequency vector of the stream ($X_i$ is number of occurrences of $i$ in the stream)
  - $F_2(X) = \sum_i X_i^2$

- [CCM 09/CCMT 12] $(\sqrt{n}, \sqrt{n})$-protocol for $F_2$.

- This is optimal. There is a lower bound that says for $(h, v)$-protocol for $F_2$, $hv=\Omega(n)$ lower bound.

- Notice $(1, n)$ and $(n, 1)$ protocols are trivial. What is non-obvious is how to trade off between $h$ and $v$. 
Self-Join Size Protocol

- [CCM 09, CCMT 12] $(\sqrt{n}, \sqrt{n})$-protocol for $F_2$.
- Recall: $F_2(X) = \sum_i X_i^2$
- View universe $[n]$ as $[\sqrt{n}] \times [\sqrt{n}]$.

Slide derived from [McGregor 10]
First idea: Have $P$ send the answer “in pieces”:
- $F_2$(row 1). $F_2$(row 2). And so on. Requires $\sqrt{n}$ communication.
- $V$ exactly tracks a row at random (denoted in yellow) so if $P$ lies about any piece, $V$ has a chance of catching her. Requires space $\sqrt{n}$.

![Frequency Square](image)

$P$ sends
- $20 = 2^2 + 4^2$
- $18 = 3^2 + 3^2$
- $4 = 2^2$

Slide derived from [McGregor 10]
Problem: If $P$ lies in only one place, $V$ has small chance of catching her.

We would like the following to hold: if $P$ lies about even one piece, she will have to lie about many.

Solution: Have $P$ commit (succinctly) to second frequency moment of rows of an error-corrected encoding of the input.

Need $V$ to evaluate any row of the encoding in a streaming fashion. Can do this for “low-degree extension” code. Note: this code is systematic, meaning the first $n$ symbols are just the input itself.
Error-corrected Encoding of Frequency Square $X$

Input is embedded in encoding (low-degree extension)

These values will all lie on low-degree polynomial $s(X)$

H sends

\[ 20 = 2^2 + 4^2 \]
\[ 18 = 3^2 + 3^2 \]
\[ 4 = 2^2 \]
\[ 26 = (-1)^2 + (-5)^2 \]
\[ 180 = (-6)^2 + (-12)^2 \]
\[ 610 = (-13)^2 + (-21)^2 \]
Formal Protocol

• Pick a finite field $\mathbb{F}_p$. $X$ implies a two-variate polynomial $f(x, y)$ over $\mathbb{F}_p$ such that $f(i,j) = X_{(i,j)}$ for all $(i,j) \in [\sqrt{n}] \times [\sqrt{n}]$.

• Let $s(X) = \sum_{y \in [\sqrt{n}]} f^2(X, y)$.

• $V$ picks a random $r \in \mathbb{F}_p$ and computes $s(r) = \sum_{y \in [\sqrt{n}]} f^2(r, y)$ while observing stream (takes $\sqrt{n}$ space).

• $P$ sends a univariate polynomial $g(X)$ claimed to be $s(X)$. Note $g$ has degree $2(\sqrt{n} - 1)$ and so can be represented in $2\sqrt{n}$ words.

• $V$ checks that $g(r) = s(r)$. If so, $V$ outputs $\sum_{x \in [\sqrt{n}]} s(x)$. If $P$ lies (i.e. $g(X) \neq s(X)$), then w.h.p. $s(r) \neq g(r)$ and thus $P$ will be caught.
A general technique

- Arithmetization: Given function $f'$ defined on a small domain, extend domain of $f'$ to a large field and replace $f'$ with its low-degree extension (LDE) $f$ as a polynomial over the field.

- Can view $f$ as an error-corrected encoding of $f'$. The error correcting properties of $f$ give $V$ considerable power over $P$.

- If two (boolean) functions differ in one location, their LDE’s will differ in almost all locations.
Why study $F_2$?

- Optimal Bipartite Perfect Matching Protocol
- Optimal Matrix-Vector Multiplication Protocol
- Optimal Subset Protocol

Omitted: Connectivity, Counting Triangles, Shortest $s$-$t$ path.
Matrix-Vector Multiplication [CMT10]

- Goal: Given $n \times n$ integer matrix $A$ and vector $x$, make $P$ provide the vector $Ax$ and proof of correctness.

- We will get optimal $(n^{2-\alpha}, n^{\alpha})$ protocol for any $\alpha \in [0,1]$. Lower bound: $hv=\Omega(n^2)$. 

Matrix-Vector Multiplication [CMT10]

- Fact: inner-product protocol follows from F2 protocol.
Matrix-Vector Multiplication [CMT10]

- First idea: Treat as $n$ separate inner-product queries, one for each row of $A$.
- Worse than “naïve” solution.
- Multiplies both $h$ and $v$ by $n$, as compared to a single inner-product query.
Matrix-Vector Multiplication [CMT10]

- First idea: Treat as $n$ separate inner-product queries, one for each row of $A$.
  - Worse than “naïve” solution.
  - Multiplies both $h$ and $v$ by $n$, as compared to a single inner-product query.

- Key observation: one vector, $x$, in each inner-product query is constant.
  - Can exploit this structure and linear hashing techniques to only multiply $h$ by $n$.
  - $v$ will be the same as for a single inner product query.
Subset Protocol [CCMT12]

- Goal: Given a list of elements in set $\mathbf{Y}$ followed by a list of elements in set $\mathbf{X}$, determine if $\mathbf{X} \subseteq \mathbf{Y}$.

- Result: $(n^\alpha, n^{1-\alpha})$ protocol for any $\alpha \in [0,1]$. This is optimal.

- Solution: Represent $\mathbf{X}$ and $\mathbf{Y}$ by indicator vectors $\mathbf{x}$ and $\mathbf{y}$. Then $\mathbf{X} \subseteq \mathbf{Y}$ if and only if $F_2(\mathbf{y}-\mathbf{x}) = F_2(\mathbf{y}) - F_2(\mathbf{x})$. So we can just run three copies of the $F_2$ protocol.
Bipartite Perfect Matching [CCMT12]

- Goal: Given a list of edges $e_1 \ldots e_m$ defining a bipartite graph $G=(V, E)$, determine whether $G$ contains a perfect matching (a set $M \subseteq E$ such that each vertex appears in exactly one edge in $M$).

- Result: $(n^{2-\alpha}, n^\alpha)$ protocol for any $\alpha \in [0,1]$. This is optimal.
Bipartite Perfect Matching [CCMT12]

- Solution: If $G=(V, E)$ contains a perfect matching $M$, $P$ lists the edges in $M$. Requires communication $O(n)$.

- $V$ must check the matching is feasible! That is,
  1. $M$ is a matching (i.e. each node appears in exactly one edge in $M$). Can be done with standard hashing techniques.
  2. $M \subseteq E$ (i.e. $P$ isn’t making up edges). Use the Subset protocol.
Bipartite Perfect Matching [CCMT12]

- Solution: If $G=(V, E)$ does not contain a perfect matching, Hall’s Matching Theorem provides a witness to this fact.

1. The witness is a subset $S \subseteq V$ that satisfies a certain property (specifically, $S$ has a small neighborhood). Requires communication $O(n)$.
2. $V$ can check witness validity using matrix-vector multiplication protocol.
3. The matrix $A$ is $G$’s adjacency matrix, and the vector $x$ is the indicator vector of $S$.
4. $V$ can extract the size of $S$’s neighborhood from $Ax$. 
Bipartite Perfect Matching [CCMT12]

- Hall’s Matching Theorem: $G=\langle V, E \rangle$ does not contain a perfect matching if and only if there is a subset $S$ of the nodes in the left partite set such that $|N(S)| < |S|$, where $N(S)$ is neighborhood of $S$.

1. $|N(S)|$ equals the Hamming Norm of $Ax$, where $A$ is adjacency matrix of $G$ and $x$ is indicator vector of $S$.

2. If $P$ sends $S$, $V$ can check $|N(S)| < |S|$ using matrix-vector multiplication protocol applied to $Ax$. 
Interactive Protocols: A Sampling
Interactive Model: A General Result

- Powerful constructions from IP literature can work with streaming verifier. Includes “Interactive Proofs for Muggles” [GKR 08] and a construction of Kilian [Kilian 92].

- Therefore: (polylog \( n \), polylog \( n \)) computationally sound protocols for NP. Efficient protocols even for problems hard in non-streaming setting (i.e. for NP-complete problems) if we are willing to settle for computational soundness.

- (polylog \( n \), polylog \( n \)) statistically sound protocols for all of log-space uniform NC (includes e.g. matrix problems like determinant, and graph problems like MST, shortest paths).
Some comments

- Despite powerful generality, [GKR 08] is not optimal for many low-complexity functions of high interest in streaming and database processing.

- [CTY 12] give improved protocols for these problems.
  - And argues that they are practical.
\( F_2 \) protocol

- Result: \((\log n, \log n)\)-protocol requiring \(\log n\) rounds.
  
  \([GKR 08]\) yields \((\log^2 n, \log^2 n)\) protocol requiring \(\log^2 n\) rounds.

- Moreover, can make \(P\) run in linear time.

- In our implementation, \(P\) handles \(\sim 21\) million updates/second. More experimental results later.
Omitted

• Extensions to *Frequency-Based Functions* of the form $\sum_i g(a_i)$ for arbitrary $g$. Includes DISTINCT, Higher Frequency Moments, $F_{\text{max}}$, etc.

• Heavy hitters.

• Reporting queries. Such as:
  1. **INDEX:** Given $i$, determine $a_i$.
  2. **RANGE-SUM:** Given $[l, u]$, compute $\sum_{i=l}^{u} a_i$
  3. **RANGE QUERY:** Given $[l, u]$ determine the frequency of all items between $l$ and $u$, inclusive.
Experimental Results: A Sampling
[CMT 12]
Two-Pronged Approach [CMT 12]

- Ideal: General purpose implementation allowing to verify arbitrary computation.
  - Implemented general-purpose “Interactive Proofs for Muggles” construction [GKR 08]. Encouraging results.
  - Plus extensions to help move it from theory to practice.

- Also revisit specialized protocols of [CCM 09/CCMT 12], [CMT 12], [CTY 12].
  - Speed up prover for these too.
  - Full implementation, with encouraging experimental results.
First Prong: General Purpose Construction
In “Muggles”, $P$ and $V$ first agree on a layered arithmetic circuit $C$ computing the function of interest. Then $P$ gives $V$ the output of $C$ and proves that the output is correct.

- Runs a *sum-check protocol* for each layer in turn, in order to check that all outputs of each layer are computed correctly.

- A naïve implementation of $P$ would require $\Omega(S^3)$ time, where $S$ is size of circuit $C$, because each message $P$ sends involves a sum over $S^3$ terms.

- We engineer $P$’s time down to $O(S \log S)$, under mild conditions on the circuit.
The $i$th sum-check protocol makes use of the following functions.

- Let $v = \log n$. jth gate at layer $i$ is associated with $v$-bit representation of $j$.
- Functions add-$i$, mult-$i$ (3$v$ bits to 1 bit)
  - $\text{add-}i(a,b,c) = 1$ if gate $a$ at layer $i$ is sum of gates $b$, $c$ from layer $i-1$
  - Extend to multilinear low-degree extension (this is a **SPECIAL** low-degree extension)

- Key point: If you use the **multilinear extension** of add-$i$ and mult-$i$, the sum defining P’s message simplifies nicely.
  - Each gate in layer $i$ only contributes to a single term in the sum, and this contribution can be computed in $O(1)$ time per gate.
  - So each message can be computed with a single pass over the gates at layer $i$. So $O(S \log S)$ time is required over all rounds.
Speeding up P in “Muggles” [GKR08]

- Complication: V needs to evaluate the multilinear extensions at a random point (rest of V’s computation is extremely light-weight).
  - V can do this in log space for any log-space uniform circuit. Sufficient for many streaming applications.
  - Moreover, this computation depends only on the circuit, not the input, so it can occur in an offline pre-processing phase. V will be both space- and time-efficient in online phase.
Speeding up \( \mathbf{P} \) in “Muggles” [GKR08]

- No offline phase needed if multilinear extensions of wiring predicate can be evaluated quickly.
- This holds for a class of "reasonably regular" circuits, including:
  1. FFT
  2. Frequency moments
  3. Pattern matching
  4. Matrix multiplication
  5. Circuits from [GKR08] used for simulating space-bounded Turing Machines.
Protocol Engineering: More Types of Gates

- Also extended [GKR08] to allow for more general kinds of gates in C.
- In particular, “exponentiation” and “sum with big fan-in” gates – reduces the depth of the computation.
  - Another use: computing $x^{p-1} \mod p$ gives 1 when $x$ is not zero mod $p$ and 0 otherwise by Fermat’s Little Theorem. This is a common primitive e.g. it lets you test vectors for equality.
  - Many protocols terminate by computing a big sum of values, even just once at the end.
- Smaller depth means fewer rounds of communication, but larger communication size per round. Optimize for the sweet spot in practice.
• We work over $\mathbb{F}_p$ for $p = 2^{61} - 1$.
• Keeps values in a single 64-bit data type.
• Miniscule probability of error for practical purposes.
  • Errors proportional to $1/p$.
• Reducing modulo $p$ can be done with a bit shift and a bit-wise AND operation.
  • About an order of magnitude faster than standard mod operation.
## “Muggles” Implementation

<table>
<thead>
<tr>
<th>Problem</th>
<th>Gates</th>
<th>Size (gates)</th>
<th>P time</th>
<th>V time</th>
<th>Rounds</th>
<th>Comm</th>
<th>Unverified Alg Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>$F_2$ (n=2^{20})</td>
<td>+, $\times$, $\oplus$</td>
<td>2.1M</td>
<td>29.8 s</td>
<td>.19 s</td>
<td>118</td>
<td>2.5 KB</td>
<td>0.01 s</td>
</tr>
<tr>
<td>MatMult (256 x 256)</td>
<td>+, $\times$, $\oplus$</td>
<td>42M</td>
<td>14.05 Minutes</td>
<td>.03 s</td>
<td>3910</td>
<td>91.6 KB</td>
<td>0.32 s</td>
</tr>
</tbody>
</table>
Follow-up Work [TRMP 12]

• Both $P$ and $V$’s computations in our implementation are massively parallel.

• Implemented on GPU (massively parallel hardware) and obtained robust speedups (40x-100x for $P$, 100x for $V$) relative to the sequential implementation just described.

• Parallel $P$ is roughly 100-1000 times slower than unverifiable sequential algorithms for benchmark problems like matrix multiplication and pattern matching.

• Also achieve 50x speedups for special-purpose protocols using GPU processing, relative to sequential implementation.
## “Muggles” Implementation

<table>
<thead>
<tr>
<th>Problem</th>
<th>Gates</th>
<th>Size (gates)</th>
<th>$P$ time</th>
<th>$V$ time</th>
<th>Rounds</th>
<th>Comm</th>
<th>Unverified Alg Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>$F_2$ ($n=2^{20}$)</td>
<td>$+, \times, \oplus$</td>
<td>2.1M</td>
<td>29.8 s</td>
<td>.19 s</td>
<td>118</td>
<td>2.5 KB</td>
<td>0.01 s</td>
</tr>
<tr>
<td>MatMult (256 x 256)</td>
<td>$+, \times, \oplus$</td>
<td>42M</td>
<td>14.05 Minutes</td>
<td>.03 s</td>
<td>3910</td>
<td>91.6 KB</td>
<td>0.32 s</td>
</tr>
</tbody>
</table>

$F_2 P$ time on GPU: 0.36 s  
MatMult $P$ time on GPU: 39.6 s
Second Prong: Specialized Protocols
Protocol Engineering: Smart FFTs

- Non-interactive $F_2$ protocol requires $P$ to evaluate the low-degree extension of the input at many points.
- Naively computing each point independently requires $\Omega(n^{3/2})$ time, doesn’t scale to large streams.
- Using FFT techniques, we can reduce this to $O(n \log n)$ time.
- Cooley-Tukey DFT algorithm lets us do this over the complex numbers, with the right amount of bit precision.
- But this is slow, requires care in handling precision issues.
- Well over 64 bits precision needed.
- Only works for power-of-two sized inputs; a lot of padding may be needed.
Protocol Engineering: Smart FFTs

- Used Prime Factor Algorithm instead. This works well over certain finite fields $F_p (p=2^{61}-1$ in particular) and allows us to avoid precision issues entirely.

- Achieved an implementation of P processing 250,000-750,000 updates per second across all stream lengths.
F$_2$ Experiments

Multi-round $P$ from [CTY11] vs. Non-interactive $P$ with and without FFT techniques
### Matrix-Vector Multiplication Experiments

<table>
<thead>
<tr>
<th>$\alpha$</th>
<th>V Space</th>
<th>Comm</th>
<th>V Time</th>
<th>P Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>78 KB</td>
<td>78 KB</td>
<td>4.3 s</td>
<td>1.6 s</td>
</tr>
<tr>
<td>.85</td>
<td>20 KB</td>
<td>469 KB</td>
<td>3.0 s</td>
<td>33.9 s</td>
</tr>
<tr>
<td>.80</td>
<td>12 KB</td>
<td>938 KB</td>
<td>2.8 s</td>
<td>58.9 s</td>
</tr>
<tr>
<td>.75</td>
<td>8 KB</td>
<td>1.5 MB</td>
<td>2.6 s</td>
<td>61.0 s</td>
</tr>
</tbody>
</table>

Results for single-round matrix-vector multiplication protocol from [CMT10] with FFT techniques on 10,000 x 10,000 matrix (768 MBs of data).
Open Questions

• Non-interactive protocols:
  1. Proving any specific problem requires $\max(h, v) > \sqrt{n}$ requires novel communication complexity techniques.
  2. Improved protocols for sparse graphs?

• Interactive/General Purpose protocols:
  1. Ultimate goal is a general-purpose compiler that takes as input any computer program and outputs a specification of a protocol for running the program verifiably.
    • Existing compilers (e.g. Fairplay) construct a boolean circuit operating on individual bits, and arithmetize it to get an arithmetic one. But this will lead to impractically large circuits, even for “algebraic” problems which possess small arithmetic circuits.
  2. Improved protocols for “non-algebraic” problems, which may have no small arithmetic circuits.
Thank you!
Parallelizability

- P’s computation in all our single-round protocols are easily parallelizable.

- Using an 8-core machine, and 3 OpenMP statements, achieved close to 8-fold speedup.

- $F_2$ and Matrix-Vector Multiplication also possess simple two-round MapReduce protocols (have not implemented).
Parallelizability

Single-round $F_2$ protocol on single core, $n=225$ million (uncontrolled environment)

<table>
<thead>
<tr>
<th>CPU time</th>
<th>Real time</th>
<th>Updates/s in Real Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>558.0 s</td>
<td>568.9 s</td>
<td>396,919</td>
</tr>
</tbody>
</table>

Single-round $F_2$ protocol on seven cores, $n=225$ million (uncontrolled environment)

<table>
<thead>
<tr>
<th>CPU time</th>
<th>Real time</th>
<th>Updates/s in Real Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>570.7 s</td>
<td>91.0 s</td>
<td>2.47 million</td>
</tr>
</tbody>
</table>
Non-interactive Protocol for Shortest s-t Path

- Goal: Given two nodes \( s \) and \( t \) in a weighted directed graph, compute the length of the shortest path between them.

- We achieve a \((dh, v)\) protocol for any \( hv = n^2 \), \( n \leq h \), where \( d = \max_{v \text{ reachable from } s} d(s, v) \).

- This is optimal for graphs with small diameter! Even in the unweighted, undirected case.
A Tool We Need

- Recall our non-interactive protocol for $\mathbb{F}_2$ that allowed us to compute $\sum_i X_i^2$ for any vector $X$.

- A generalization allows us to compute $\sum_i g(X_i)$ for any vector $X$ and any low-degree polynomial $g$. Call this the polynomial agreement protocol.

- Cost of this protocol depends on degree of $g$. 
The Protocol for Shortest s-t Path

- Primal LP for shortest s-t path:
  \[
  \text{Minimize } \sum_{ij} w_{ij} x_{ij} \text{ subject to }
  \sum_j x_{ij} - \sum_j x_{ji} = 0 \text{ if } i \neq s, t
  \sum_j x_{sj} - \sum_j x_{js} = 1
  \sum_j x_{tj} - \sum_j x_{jt} = -1
  0 \leq x
  \]

- Dual LP:
  \[
  \text{maximize } y_t - y_s \text{ subject to: for all } (i,j) \text{ in } E, \ y_j - y_i \leq w_{ij}.
  \]

- By total unimodularity, both primal and dual LPs have integral optima.
The Protocol for Shortest s-t Path

- First step: \( P \) provides a (claimed) primal-optimal solution i.e. a path between \( s \) and \( t \) claimed to be the shortest.
  - Requires communication \( O(n) \).

- \( V \) must check that the path is feasible i.e. it is indeed a path, and \( P \) isn’t making up edges or lying about their weight.

- Can do this using a generalization of the Subset Protocol.

- For this part of the protocol, we can achieve communication \( O(h) \) and space \( O(v) \) for any \( hv=n^2 \) and \( n \leq h \).
The Protocol for Shortest s-t Path

- Recall Dual LP:
  \[
  \text{maximize } y_t - y_s \text{ subject to: for all } (i,j) \in E, \ y_j - y_i \leq w_{ij}.
  \]
- Second Step: P proves optimality of the primal solution by giving a dual solution \( y \) of the same value.
  - There are only \( n \) dual variables so specifying a dual solution takes \( O(n) \) communication.
- V must check \( y \) is dual feasible.
The Protocol for Shortest s-t Path

- Recall Dual LP:
  \[
  \text{maximize } y_t - y_s \text{ subject to for all } (i,j) \text{ in } E, \ y_j - y_i \leq w_{ij}.
  \]

- Let \( Z \) be the length-\( n^2 \) vector given by \( Z_{ij} = w_{ij} + y_i - y_j \) for all edges \((i,j)\) and \( Z_{ij} = 0 \) otherwise.

- Let \( d \) be an upper bound on \( |y|_\infty \), the largest absolute value of any variable in the dual solution. Then all entries of \( Z \) are between \(-2d\) and \( |w|_\infty + 2d \).

- \( y \) is dual-feasible if and only if \( \sum_i g(Z_i) = 0 \), where \( g \) is a polynomial such that \( g(x) = 0 \) for \( 0 \leq x \leq |w|_\infty + 2d \). \( g \) can have low degree if \( d \) and \( |w|_\infty \) are small.

- Can run polynomial agreement protocol to check dual feasibility!
The Protocol for Shortest s-t Path

- Details brushed under the rug:
  - Need to show V can derive a stream defining the vector Z from the actual observed graph stream.
  - Need to prove that there always exists a dual-optimal solution $y$ with $|y|_\infty \leq \max_{v \text{ reachable from } s} d(s, v)$.

- Final result: $(dh, v)$ protocol for any $hv=n^2$, $n \leq h$, where $d=\max_{v \text{ reachable from } s} d(s, v)$.
Extension to Frequency-Based Functions

• Frequency based function $F(a)$ is of the form $F(a) = \sum_i g(a_i)$ for some $g: \mathbb{N}_0 \rightarrow \mathbb{N}_0$.

• e.g. $F_k$, $F_0$ (DISTINCT), “How many items have frequency at most $i$?”, verifying $F_{\text{max}}$ (highest-frequency).
Extension to Frequency-Based Functions

- First idea: extend $g$ to a polynomial $g$ over $\mathbb{F}_p$ and apply a sum-check protocol to the polynomial $g \circ f$.
- Streaming $V$ can evaluate $g \circ f(r)$ by computing $f(r)$ and then $g(f(r))$.
- Problem: $g$ might have degree $n$. Resulting communication cost is $dn$, worse than trivial protocol.
Extension to Frequency-Based Functions

- First idea: extend $g$ to a polynomial $g$ over $\mathbb{F}_p$ and apply a sum-check protocol to the polynomial $g \circ f$.
  - Streaming $V$ can evaluate $g \circ f(r)$ by computing $f(r)$ and then $g(f(r))$.
  - Problem: $g$ might have degree $n$. Resulting communication cost is $dn$, worse than trivial protocol.

- Solution: We give a $(1/\phi \log u, 1/\phi \log u)$ protocol to identify all items of frequency at least $\phi m$ (the “$\phi$-heavy hitters”). Use this protocol to “remove” the heavy items, which allows to control degree of $g$. 
Extension to Frequency-Based Functions

- Result: a ($\sqrt{u \log u}$, $\log u$)-protocol for any frequency-based function that takes $\log u$ rounds.

- [GKR 08] yields ($\log^2 u$, $\log^2 u$) protocol.

- For 1 TB of data, $\sqrt{u}$ is on the order of 1 MB, $\log^2 u$ is on the order of thousands, $\log u \approx 40$.

- Might prefer to communicate 1 MB of data over 40 rounds than 1 KB over thousands of rounds due to network latency.
Reporting Queries

• Sub-vector query: Given $q_L$ and $q_R$, determine the non-zero entries of $(a_{q_L}, \ldots, a_{q_R})$.

• We give a $(k + \log u, \log u)$-protocol for Sub-vector requiring $\log u$ rounds, where $k$ is number of non-zero entries in $(a_{q_L}, \ldots, a_{q_R})$.

• Protocol is reminiscent of Merkle trees, but we achieve statistical soundness.
Lower Bound: $hv=\Omega(n)$

- Suppose protocol “A” has parameters $(h, v)$, error $\delta=1/3$. 

Slide derived from [McGregor 10]
Lower Bound: $hv = \Omega(n)$

- Suppose protocol “A” has parameters $(h, v)$, error $\delta = 1/3$.
- Define “B”:
  - Protocol “B” has parameters $(h, hv)$, error $(1/3)^2 - h$ (i.e. each $a$ is “bad” for only $(1/3)^2 - h$-fraction of V’s possible coin flips).

1. V runs $t = \Theta(h)$ copies of her part of “A” in parallel
2. Use H’s annotation $a$ for all copies
3. Output majority answer if it exists, else reject.
Lower Bound: $hv = \Omega(n)$

- Suppose protocol “A” has parameters $(h, v)$, error $\delta = 1/3$.
- Define “B”:
  
  Protocol “B”
  
  1. V runs $t = \Theta(h)$ copies of her part of “A” in parallel
  2. Use H’s annotation $a$ for all copies
  3. Output majority answer if it exists, else reject.

- Protocol “B” has parameters $(h, hv)$, error $(1/3)2^{-h}$ (i.e. each $a$ is “bad” for only $(1/3)2^{-h}$-fraction of V’s possible coin flips).

- Define “C”:
  
  V ignores annotation, tries all $2^h$ possible annotations and accepts if *any one* of them causes “B” to accept.
  
  Ensures error at most $1/3$, no helper needed.

- Algorithm “C” solves $F_2$ in $hv$ space. But we know $F_2$ requires $\Omega(n)$ space (randomized).

Slide derived from [McGregor 10]