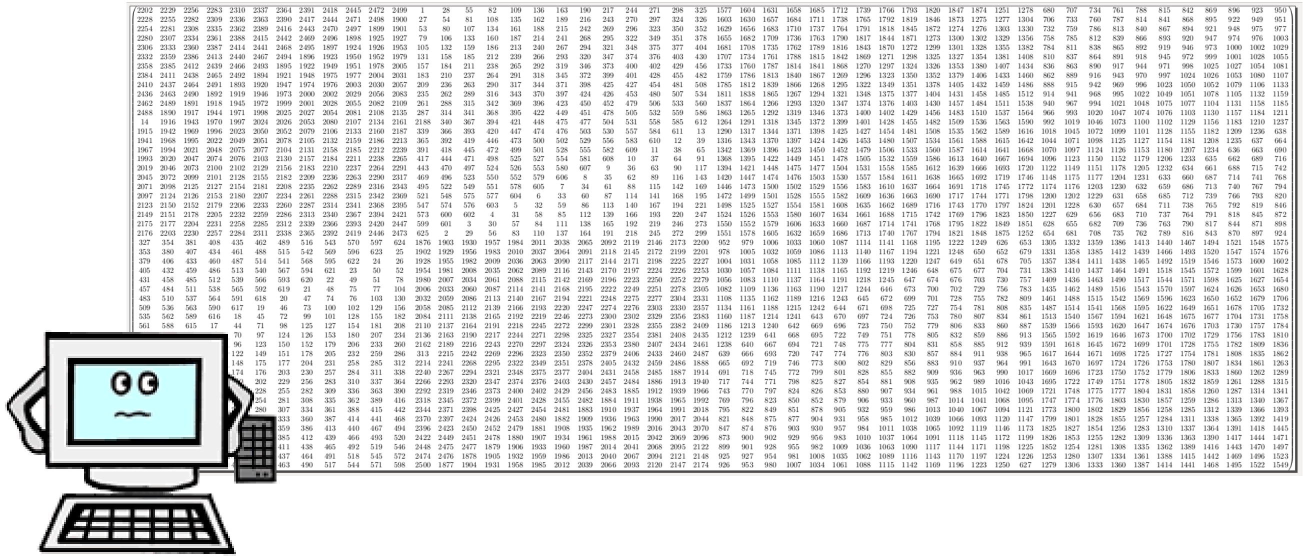
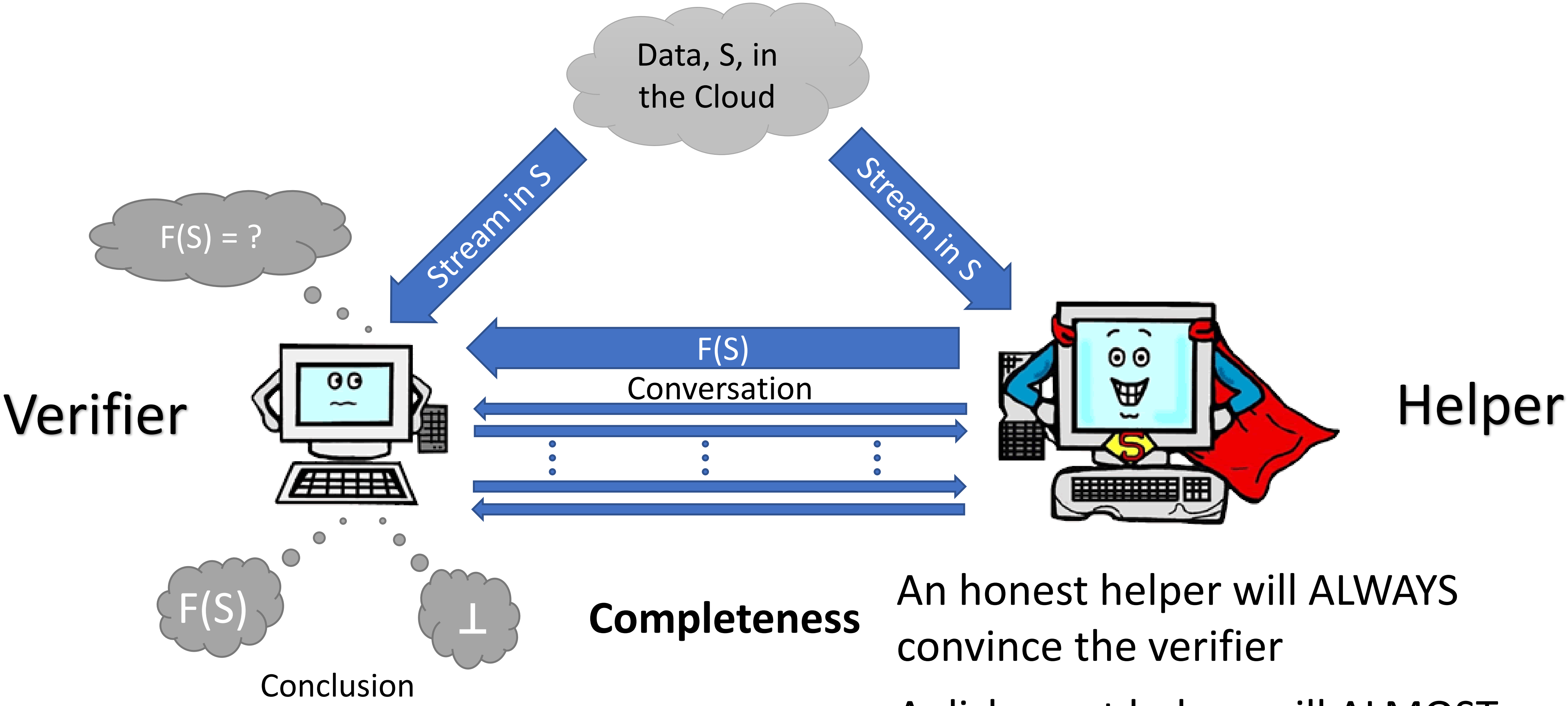
Efficient Interactive Proofs for Linear Algebra

Christopher Hickey Graham Cormode

University of Warwick



Our Model – Streaming Interactive Proofs

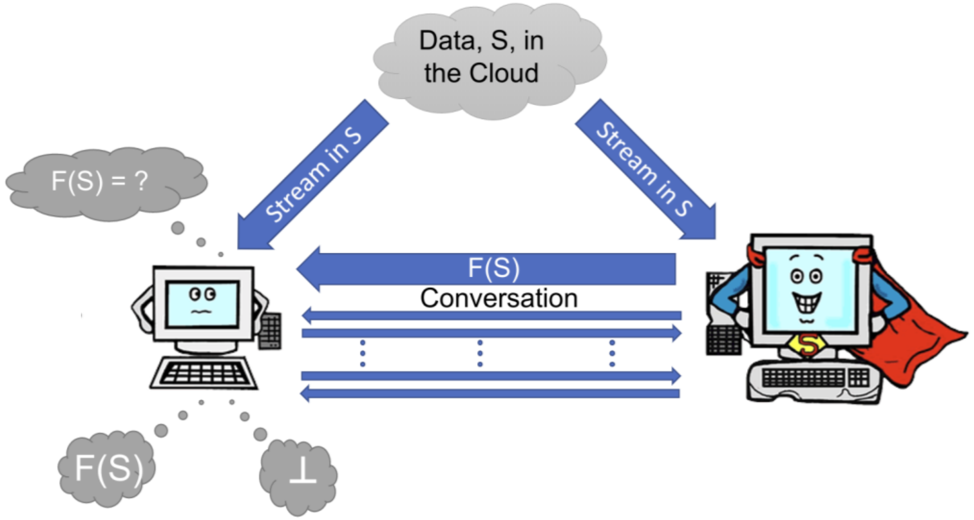


Completeness An honest helper will ALWAYS convince the verifier

Soundness A dishonest helper will ALMOST NEVER trick the verifier

Costs in SIPs

Interactivity	Number of rounds
Verifier Memory	Working memory of the verifier
Communication	Total communication sent in both directions
Verifier’s Streaming Cost	Computational complexity of streaming in S
Verifier’s Checking Cost	Computational complexity of streaming the conversation
Helper Overhead	Additional work required by the helper beyond solving the problem



What Costs to Trade Off

“Non-interactive” costs
Verifier Memory
Verifier's Streaming Cost
Helper Overhead

“Interactive” costs
Communication
Interactivity
Verifier's Checking Cost

Rule of thumb: Decreasing a non-interactive cost usually increases some interactive cost, and vice-versa.

Our work attempts to see which cost is best to relax in order to minimize the **total time** of the protocol.

We focus on linear algebra, as this is a primitive for many problems, and yields interesting examples.

Warm-up: Inner Product

For two vectors of length n, ignoring constant factors.

	Method	This Work	[CTY11] Binary Sum-Check	[CMT12] FFT and LDEs
"Non-interactive" costs	Total Communication	$dn^{1/d}$	$\log n$	$\sqrt{n}$
	Verifier Checking Cost	$dn^{1/d}$	$\log n$	$\sqrt{n}$
	Rounds	$d-1$	$\log n$	1
"interactive" costs	Helper Overhead	$n \log n$	$n \log n$	$n \log n$
	Verifier Streaming Cost	$dn^{1+1/d}$	$n \log n$	$n\sqrt{n}$
	Verifier Memory	$d + n^{1/d}$	$\log n$	$\sqrt{n}$

d is a variable parameter from 1 to $\log n$ determining the number of rounds.

Note that if we set $d = 2$, we get [CMT12], and if we set $d = \log n$ we get [CTY11].

$$n^{\frac{1}{\log n}} = 2$$

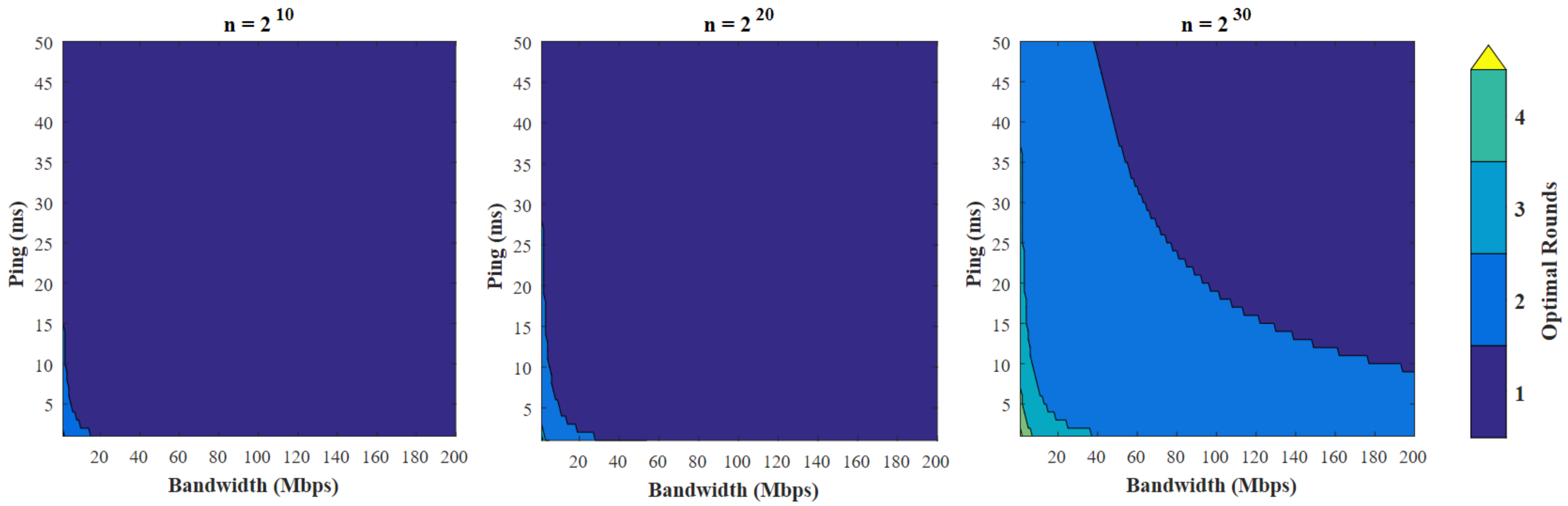
Matrix Multiplication

For two matrices of size $n \times n$, ignoring constant factors.

	Method	This Work	[Thaler13] Binary Sum Check	[CH18] Fingerprints
"Non-interactive" costs	Total Communication	$dn^{2/d}$	$\log n$	n^2
	Verifier Checking Cost	$n^2 + dn^{2/d}$	$n^2 + \log n$	n^2
	Rounds	d	$1 + \log n$	1
"interactive" costs	Helper Overhead	$n^2 \log n$	$n^2 \log n$	1
	Verifier Streaming Cost	$dn^{2+2/d}$	$n^2 \log n$	$n^2 \log n$
	Verifier Memory	$dn^{2/d}$	$\log n$	1

d is a variable parameter from 1 to $\log n$ determining the number of rounds.

Motivation: Minimizing Total Time Taken



Number of rounds considering ***only*** communication for Matrix Multiplication that decreases the total time to send all the data over all the rounds.

Less interactivity, even with more communication reduces overall time!

The question is now how much does this affect the other overheads?

Problem: Given streaming access to two data sets, how can we check they're the same (with high probability)?

Solution: Low Degree Extensions!

Consider a polynomial which passes through each data point (i, v_i).

We index the data via a hypercube $[l]^d$ and create the **unique** polynomial of degree l in d variables that passes through each data point.

We can evaluate this LDE at a random point in $\mathbb{F}^d$ as we stream the data!

LDEs share many useful properties,

- The probability of two different vectors having the same LDE evaluation at a random point is very small
- LDEs have linearity
- They can be constructed in $O(nld)$ time

LDEs can be used with the powerful sum-check protocol [LFKN92] to sum a function of the elements in a data set.

LDEs are very useful for making efficient protocols for inner product and matrix multiplication that use $d = \log n$ and $l = 2$.

Problem: Given $u, v \in \mathbb{F}^n$, how can we check the inner product $u^T v$? [CTY11]

[CTY11] uses LDEs with $n = l^d$, we represent the d-variate LDE of u by $\tilde{u}$ and v by $\tilde{v}$. We want to find

$$u^T v = \sum_{i=1}^{n} u_i v_i = \sum_{k_1=0}^{l-1} \cdots \sum_{k_d=0}^{l-1} \tilde{u}(k_1, \dots, k_d)\, \tilde{v}(k_1, \dots, k_d)$$

They use a well-known protocol called 'sum-check' [LFKN92], a d-round protocol in which the prover allows the verifier to check the following sum against a 'secret' constructed in the streaming phase $\tilde{u}(r_1, \dots, r_d)\tilde{v}(r_1, \dots, r_d)$.

The messages the prover sends are degree $2l$ polynomials, which the prover can create in time $O(nld)$.

Problem: How were LDEs used to solve inner product? [CTY11]

The protocol uses sum-check, this is a d-round protocol involving d messages of $2l$ field elements.

Classification	Cost (ignoring constant factors)	Explanation
Interactivity	d	d rounds
Verifier Memory	$l + d$	Needs to store r, and l evaluations of g_j
Communication	ld	d messages of $2l$ field elements
Verifier's Streaming Cost	$dn^{1+1/d}$	Evaluating $\tilde{u}(r_1, \ldots, r_d)\tilde{v}(r_1, \ldots, r_d)$
Verifier's Checking Cost	ld	l evaluations of g_j, d times
Helper Overhead	nld	Forming g_j for j in $[1, d]$

[CTY11] note that using $l = 2$ and $d = \log n$ minimizes many costs, but with the cost of maximum interactivity.

Problem: How can we make [CTY11] variable-round without sacrificing Helper Overhead?

[CMT12] introduced a non-interactive protocol that massively reduced the helper overhead to $n \log n$ where the prover uses convolutions and fast fourier transforms.

We generalize this result to variable round protocols, as well as implementing a 'stop-short' reduction in sum-check to allow the protocol to run in $d - 1$ rounds.

Note that even with this adaptation, the memory efficient method is to use $d = \log n$. We aim to show experimentally that in practice, it's often most time efficient to use as much memory as you have available.

However, the main motivation behind this protocol is how we can use it as a primitive for other protocols.

Problem: Vector-Matrix-Vector Multiplication

A first example of how to use this primitive is a nifty algebraic trick for multiplying two vectors $u, v \in \mathbb{F}^n$ and $A \in \mathbb{F}^{n \times n}$ we can verify $u^T A v$ by considering

$$u^T A v = \sum_{i=1}^{n} \sum_{j=1}^{n} u_i A_{ij} v_j = (uv^T)_{vec} \cdot A_{vec}$$

Where the subscript $_{vec}$ refers to a canonical transformation from a matrix to a vector.

Using the inner product protocol on the LDEs of A and uv^T gives us a protocol with communication and space costs $\mathrm{O}(l^2 d)$ and d rounds.

Note we can use the inner product protocol as we can construct $\widetilde{uv^T}(r_1, r_2)$ using $\tilde{u}(r_1)\tilde{v}(r_2)$.

Problem: Matrix Multiplication

For matrices $\mathrm{A}, \mathrm{B} \in \mathbb{F}^{n \times n}$ we will have to verify that a sent **matrix** is correct, not just a scalar.

[Thaler13] uses LDEs for verification, and uses $\log n$ rounds and the inner product definition of matrix multiplication.

We use fingerprints in conjunction with our inner product protocol, however implement the outer-product definition of matrix multiplication.

For a vector $\mathrm{v} \in \mathbb{F}^n$, the fingerprint of v with respect to $x \in_R \mathbb{F}$ is:

$$f_x(v) = \sum_{i=0}^{n-1} v_i x^i$$

Fingerprints have the property $f_x(u^T v) = f_{x^n}(u) f_x(v)$ [CH18].

We define fingerprints for matrices analogously.

Problem: Matrix Multiplication

For matrices $\mathrm{A}, \mathrm{B} \in \mathbb{F}^{n \times n}$ we will have to verify that a sent matrix is correct, not just a scalar.

Fingerprints are useful with the following identity

$$f_x(AB) = \sum_{i=1}^{n} f_{x^n}\left(A_i^{\downarrow}\right) f_x(B_i^{\rightarrow}) = \begin{pmatrix} f_{x^n}\left(A_1^{\downarrow}\right) \\ \vdots \\ f_{x^n}\left(A_n^{\downarrow}\right) \end{pmatrix} \cdot \begin{pmatrix} f_x(B_1^{\rightarrow}) & \cdots & f_x(B_n^{\rightarrow}) \end{pmatrix}$$

To use our inner product protocol, the verifier simply needs to be able to find the LDE of these two vectors at a random point, which it can using the linearity of fingerprints and LDEs.

Practical Analysis – Matrix Multiplication

Classification	Costs	How we'll time it
Interactivity	d	The latency between each machine × number of rounds × 2
Verifier Memory	$dn^{2/d}$	n/a
Communication	$dn^{2/d}$	The bandwidth to send all the messages × communication
Verifier's Streaming Cost	$dn^{2+2/d}$	We will not time this, as it happens concurrently to seeing the data, which can happen at any point prior to the protocol starting.
Verifier's Checking Cost	$n^2 + dn^{2/d}$	The time to fingerprint the matrix The time taken to run the interactive protocol
Helper Overhead	$n^2 \log n$	The cost of producing the sum-check polynomials.

Practical Analysis – Matrix Multiplication

Using bandwidth of 100Mbps And Latency of 20ms			Interactivity	Communication	Verifier's Checking cost	Verifier's Checking cost	Helper overhead
Matrix Size ($n = l^d$)	l	d	Latency (ms)	Bandwidth (ms)	Fingerprinting AB (ms)	Interactive Stage (ms)	Forming messages (ms)
2^{12}	2	12	440	0.014	150	0.009	0.23
	8	4	120	0.015	150	0.035	0.10
	64	2	40	0.041	150	0.043	0.11
2^{16}	2	16	600	0.019	40000	0.006	3.50
	16	4	120	0.031	40000	0.046	1.60
	256	2	40	0.163	40000	1.700	1.80
2^{18}	2	18	680	0.022	600000	0.006	14.10
	8	6	200	0.026	600000	0.030	6.30
	512	2	40	0.328	600000	6.400	7.80

Practical Analysis – Matrix Multiplication

Using bandwidth of 100Mbps And Latency of 20ms			Interactivity	Communication	Verifier's Checking cost	Verifier's Checking cost	Helper overhead
Matrix Size ($n = l^d$)	l	d	Latency (ms)	Bandwidth (ms)	Fingerprinting AB (ms)	Interactive Stage (ms)	Forming messages (ms)
2^{12}	2	12	440	0.014	150	0.009	0.23
	8	4	120	0.015	150	0.035	0.10
	64	2	40	0.041	150	0.043	0.11
2^{16}	2	16	600	0.019	40000	0.006	3.50
	16	4	120	0.031	40000	0.046	1.60
	256	2	40	0.163	40000	1.700	1.80
2^{18}	2	18	680	0.022	600000	0.006	14.10
	8	6	200	0.026	600000	0.030	6.30
	512	2	40	0.328	600000	6.400	7.80

This is independent of interactivity!

Practical Analysis – Matrix Multiplication

Using bandwidth of 100Mbps And Latency of 20ms			Interactivity	Communication	Verifier's Checking cost	Verifier's Checking cost	Helper overhead
Matrix Size ($n = l^d$)	l	d	Latency (ms)	Bandwidth (ms)	Fingerprinting AB (ms)	Interactive Stage (ms)	Forming messages (ms)
2^{12}	2	12	440	0.014	150	0.009	0.23
	8	4	120	0.015	150	0.035	0.10
	64	2	40	0.041	150	0.043	0.11
2^{16}	2	16	600	0.019	40000	0.006	3.50
	16	4	120	0.031	40000	0.046	1.60
	256	2	40	0.163	40000	1.700	1.80
2^{18}	2	18	680	0.022	600000	0.006	14.10
	8	6	200	0.026	600000	0.030	6.30
	512	2	40	0.328	600000	6.400	7.80

The latency dominates the other costs significantly, and this would still be the case even with a latency of 5ms.

This clearly demonstrates the location of the **time** bottle-neck in this protocol.

Interactivity and verifier memory

- The time bottleneck is the latency between the verifier and the prover, dominating the other costs that decrease with increased interactivity.
- This leads us to want to reduce the interactivity as much as the verifier's memory ($O(l^2 d)$) will let us.
- For example, for $n = 2^{18}$, optimality will likely be with a 6 round protocol.

For a matrix of size $n = 2^{18}$		
$\boldsymbol{l}$	$\boldsymbol{d}$	$\boldsymbol{l^2 d}$
2	18	72
4	9	144
8	6	384
64	3	12288
512	2	524288

Closing Thoughts

- For our applications, where the problem is highly structured, the interactive protocols are very efficient.
- By adapting [CMT12]'s FFT protocol for binary sum-check to arbitrary sum-check, we achieve faster protocols than previously possible.
- We demonstrate how using certain applications are better with LDEs and some with fingerprints, and show some useful algebraic tricks to apply.
- Latency is the dominant time bottleneck.
- The most efficient protocol for the verifier will be to use as much memory as possible, even though the asymptotics say more interactivity is better.
- A large cost for the verifier is the initial streaming phase. Additional work could be done to uncover efficiency tricks to find the secret.

Any Questions? Email C.Hickey@warwick.ac.uk