CacheCash: A Cryptocurrency-based Decentralized Content Delivery Network

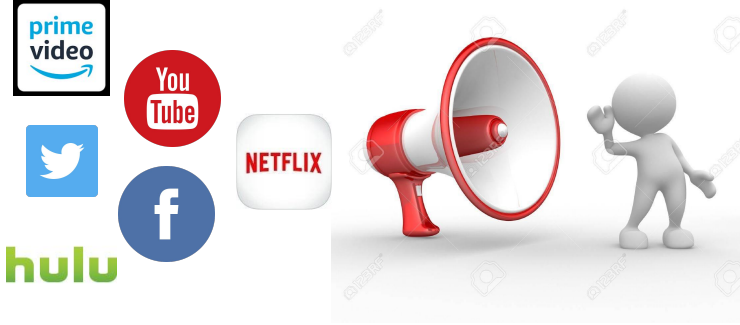
Ghada Almashaqbeh
Columbia University

Ph.D Thesis Defense
May 2019

Outline

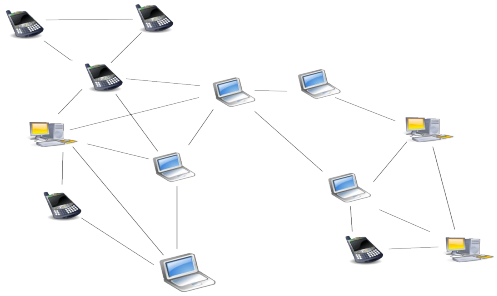
- Background, Motivation, and Problem Statement.
- Building blocks.
  - ABC.
  - CAPnet.
  - MicroCash.
- Putting it all together: CacheCash.
- Security and performance analysis.
- Future directions.
- Conclusion.

Online Content Distribution



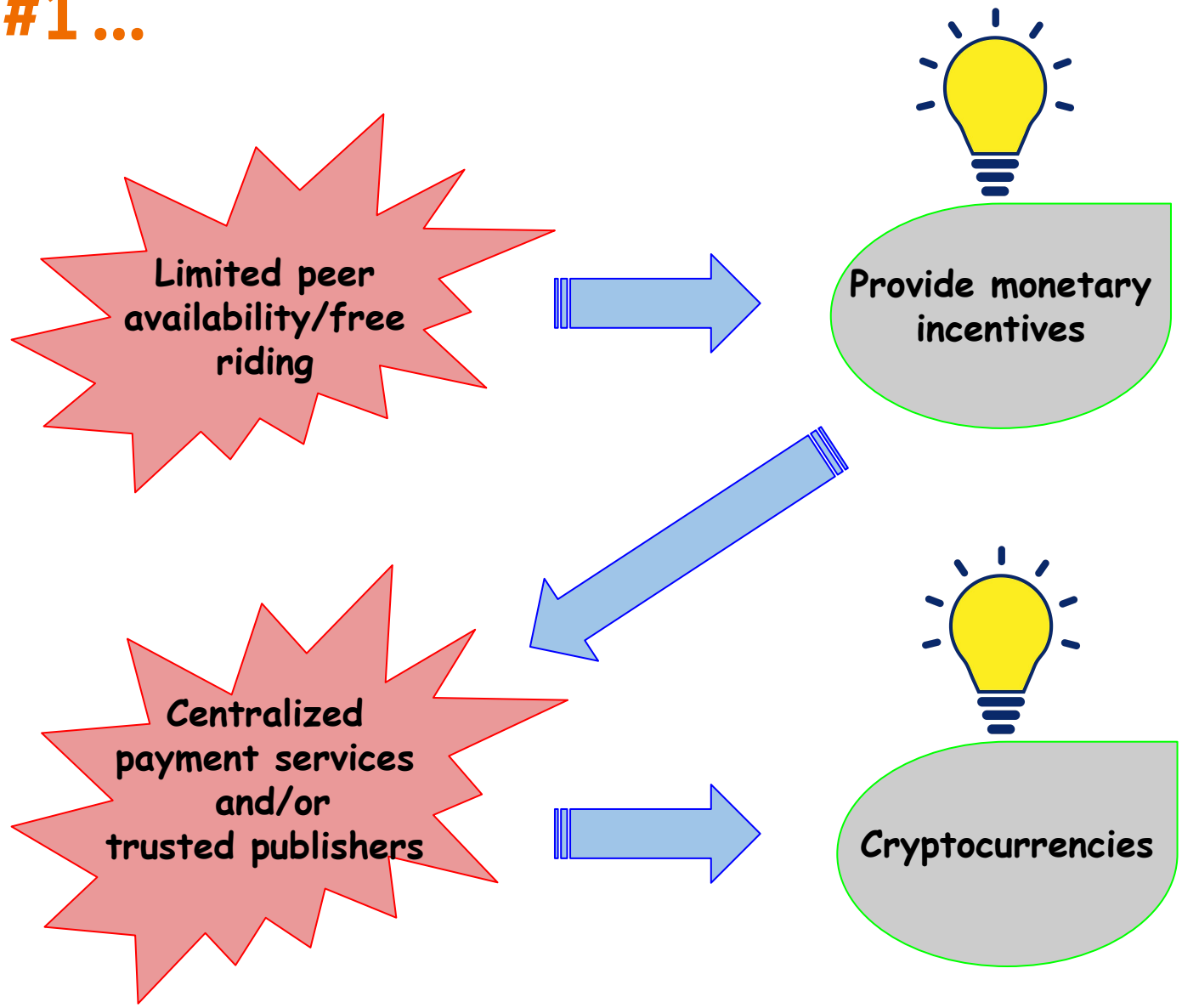
- Dramatic growth over the past decade.
  - Video streaming accounts for ~60% of today's Internet traffic, projected to exceed 80% by 2022.
- Usually, infrastructure-based content delivery networks (CDNs) are used to distribute the load.
  - Through CDN providers, e.g., Akamai.
- **Drawbacks:**
  - Impose costly and complex business relationships.
  - Require overprovisioning bandwidth to handle peak demands.
  - Issues related to reachability, delays to set up new service, etc.

Decentralized CDNs



- Utilize P2P data transfers to build dynamic CDNs.
- Allow anyone to join as a cache and distribute content to others.
- **Advantages:**
  - Offer a lower service cost.
  - Create robust and flexible CDN service.
  - Extend network coverage of traditional CDNs.
  - Scale easier with demand.
  - When managed carefully, provide a good quality of service.

But #1 …


Limited peer availability/free riding
Provide monetary incentives
Centralized payment services and/or trusted publishers
Cryptocurrencies

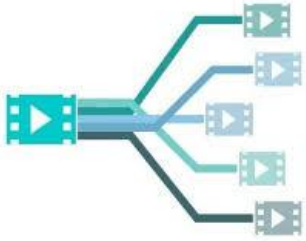
Cryptocurrencies and Blockchain Technology

- An emerging economic force that received a huge interest.
- Started with Bitcoin in 2009.
  - Currently there are ***2149 cryptocurrencies****.
  - Total capital market exceeding $180 billion.
- Early systems focused on providing a virtual currency exchange medium.
  - Distributed.
  - Publicly verifiable.
  - No need for real identities.

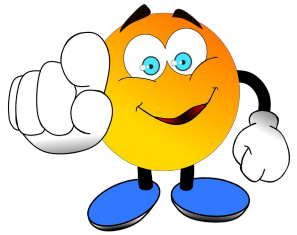
*https://coinmarketcap.com/

Cryptocurrency-Based Distributed Services

- Provide a service on top of the currency exchange medium.
  - E.g., computation outsourcing (Golem), File storage (Filecoin).
- Create an open access, distributed market.
  - Any party can join to serve others in order to collect cryptocurrency tokens.
- The mining itself could be tied to the amount of service put into the system.
- Several economic aspects:
  - Lower service cost than centralized approaches.
  - A step forward on the “useful mining” path.
  - Utility tokens vs. store of value tokens.

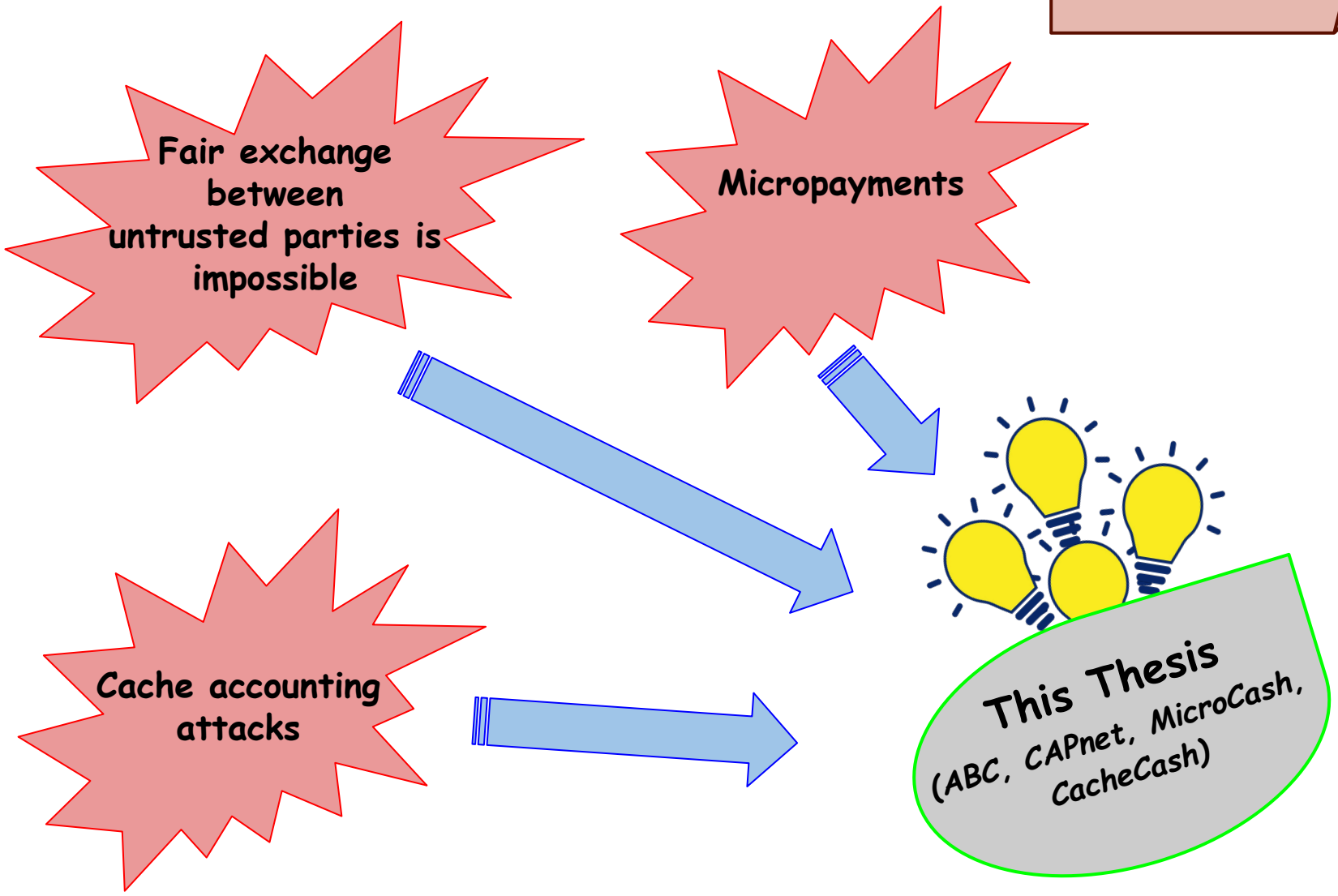


Let's build a distributed bandwidth market then!

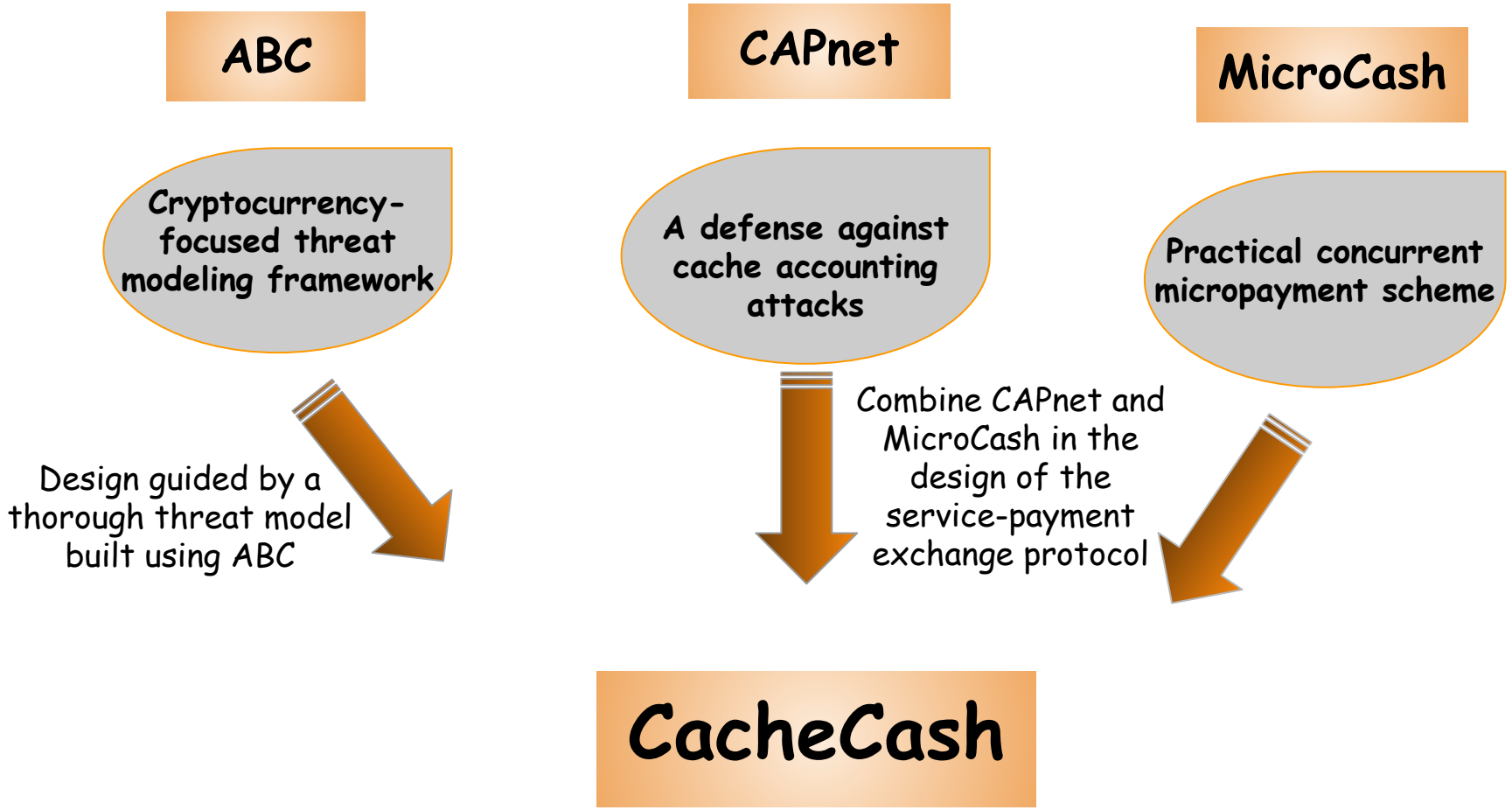


But #2 …

Non-goal:
**Digital rights management.
**Preserving user's privacy.



Roadmap


ABC
CAPnet
MicroCash
Cryptocurrency-focused threat modeling framework
A defense against cache accounting attacks
Practical concurrent micropayment scheme
Design guided by a thorough threat model built using ABC
Combine CAPnet and MicroCash in the design of the service-payment exchange protocol
CacheCash

ABC: Asset-Based Cryptocurrency-focused Threat Modeling Framework

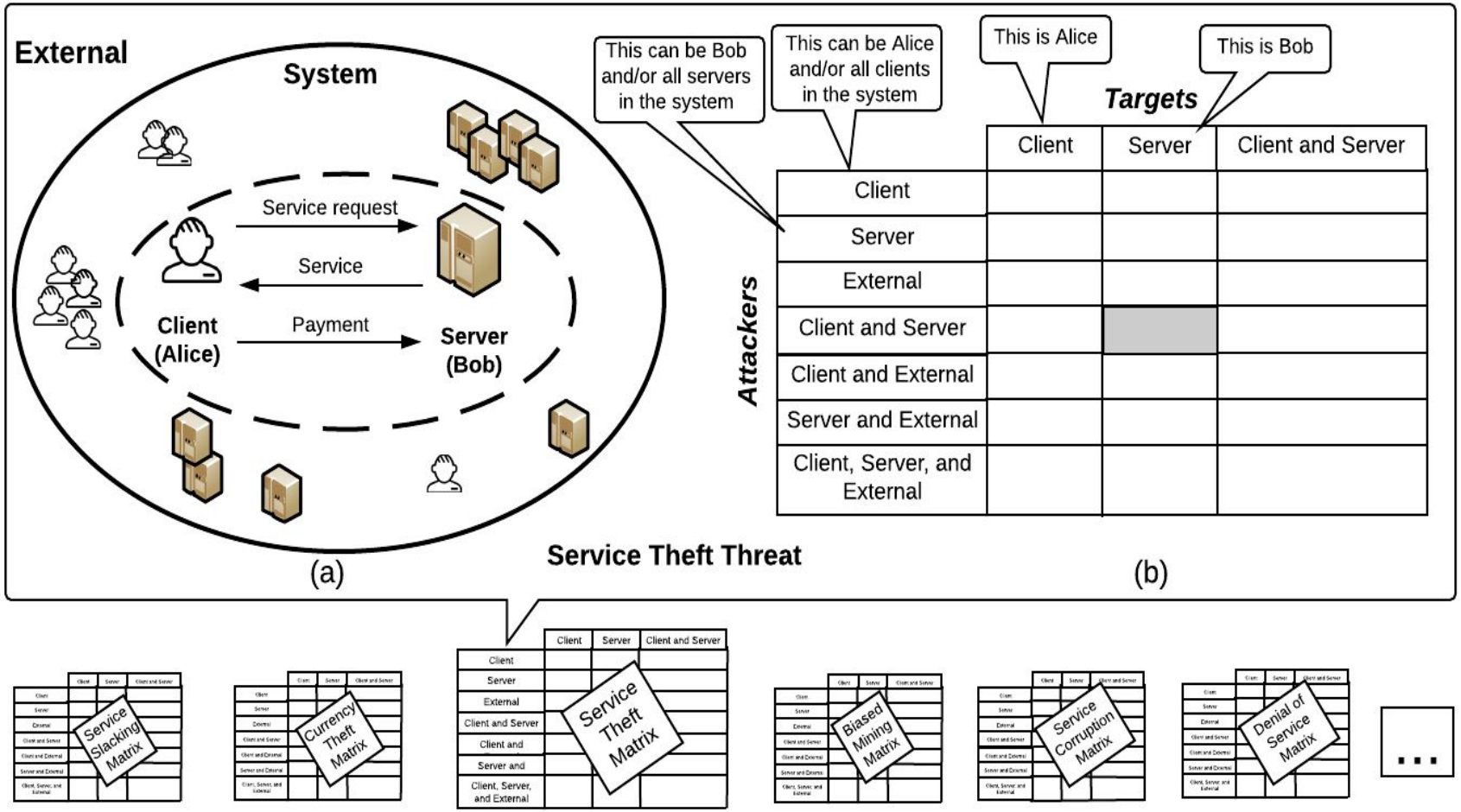
Background

- Threat modeling is an essential step in secure systems design.
  - Explore threat space and identify potential attack scenarios.
  - Helps in both guiding the system design and evaluating security in the after-design stages.
- Traditional approaches do not fit cryptocurrency-based systems.
  - Do not scale.
  - Do not explicitly account for attackers' financial motivation or collusion between these attackers.
  - Do not explicitly consider the new threat types cryptocurrencies introduce.

What is ABC?

- A systematic threat modeling framework geared toward cryptocurrency-based systems.
  - Its tools are useful for any distributed system.
- Helps system designers to consider:
  - Financial motivation of attackers.
  - New asset types in cryptocurrencies.
  - System-specific threat categories.
  - Collusion.
    - Using a new tool called a collusion matrix that also manages the complexity of the threat space.
- Acknowledges that financial incentives can play a major role in other steps in the design process.

A Collusion Matrix Example



What is ABC?

- A systematic threat modeling framework geared toward cryptocurrency-based systems.
  - Its tools are useful for any distributed system.
- Helps system designers to consider:
  - Financial motivation of attackers.
  - New asset types in cryptocurrencies.
  - System-specific threat categories.
  - Collusion.
    - Using a new tool called a collusion matrix that also manages the complexity of the threat space.
- Acknowledges that financial incentives can play a major role in other steps in the design process.

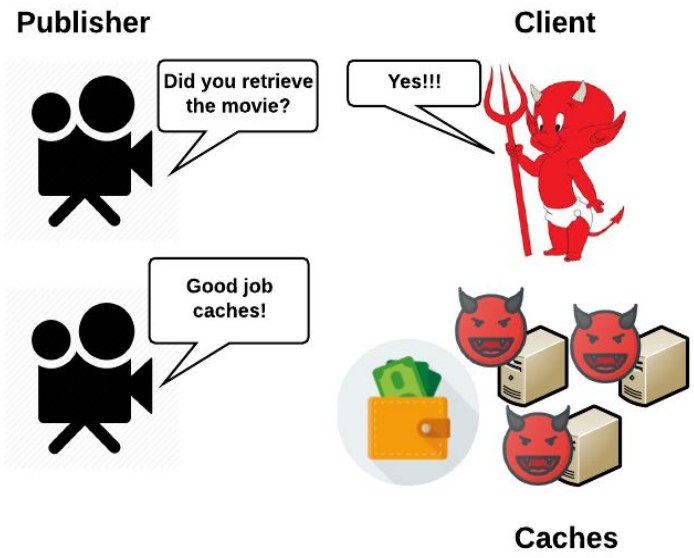
ABC and CacheCash

- Used to build a thorough threat model covering MicroCash, Bitcoin, and the service module of CacheCash.
- Total of 15 collusion matrices.
  - 651 threat cases reduced to 32 distilled cases.
- These threat models guided the design of both MicroCash and CacheCash.
  - Among them, revealed the case of ***cache accounting attacks***, for which we designed ***CAPnet*** as a mitigation.

CAPnet: A Defense Against Cache Accounting Attacks

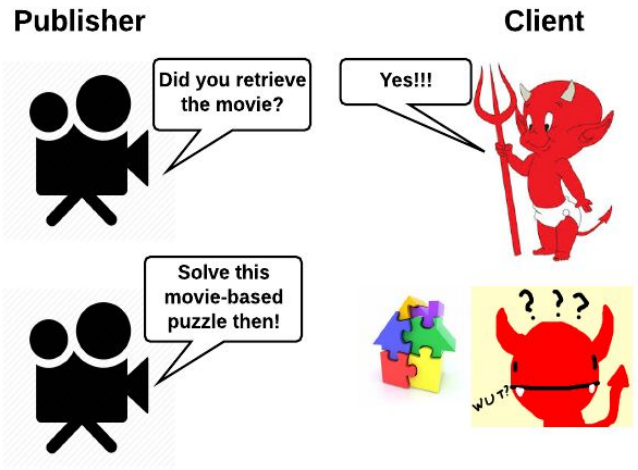
Background

- **Cache accounting attacks:** Clients collude with caches pretending to be served.
  - Caches can collect rewards without doing any actual work.
    - Confirmed by an empirical study on the Maze file system and Akamai Netsession.
- **Previous solutions:** Do not work in typical P2P networks.
  - Either rely on activity reports from the peers themselves.
  - Or assume the knowledge of peer computational power and link delay.

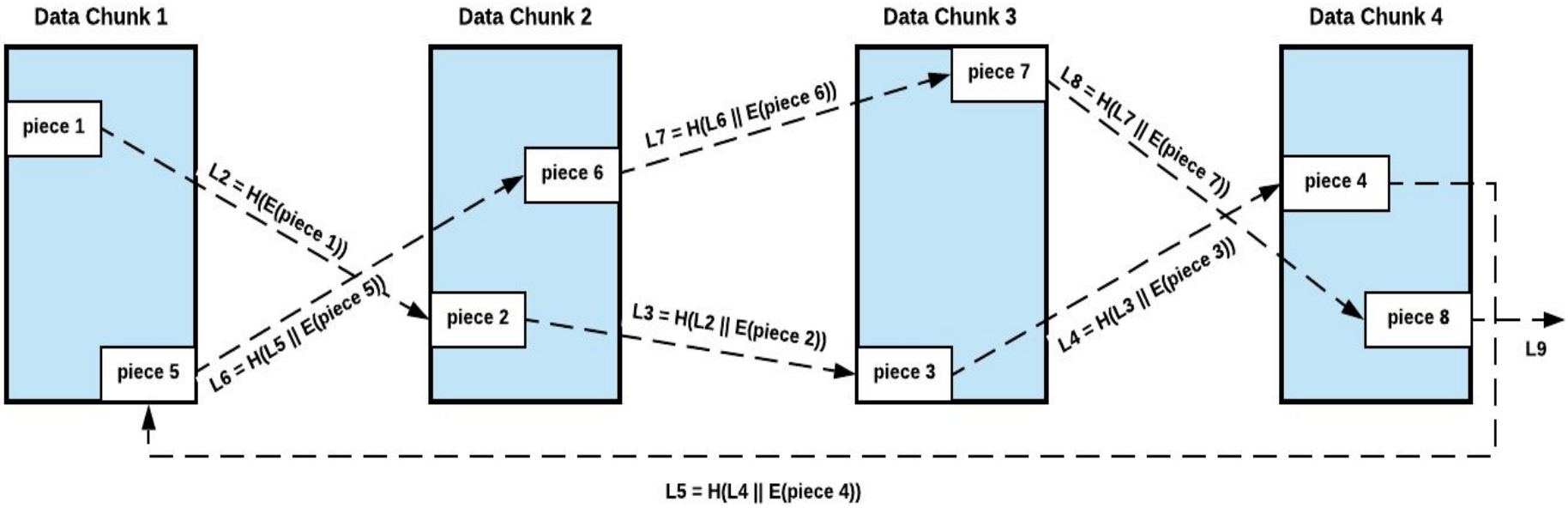


CAPnet

- Lets untrusted caches join peer-assisted CDNs.
- Introduces a novel lightweight cache accountability puzzle that must be solved using the retrieved content.
- Allows a publisher to set a bound on the amount of bandwidth an attacker must expend when solving the puzzle.



Cache Accountability Puzzle Design



Puzzle challenge = $H(L_9)$ Puzzle solution = L_9

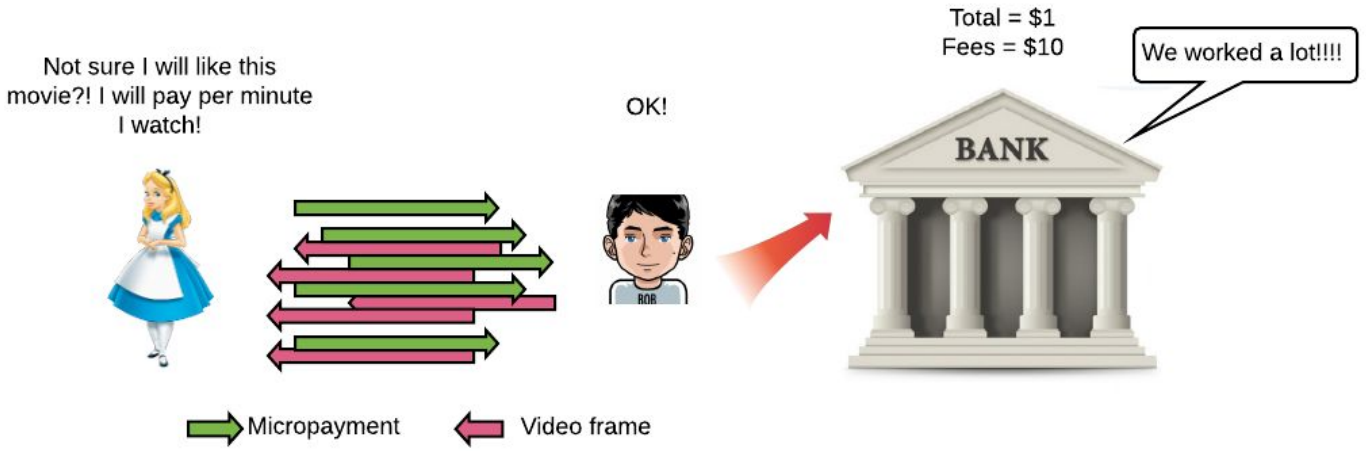
CAPnet ensures that caches perform the requested work, but how to reward them for this work?

MicroCash: Practical Concurrent Processing of Micropayments

Micropayments

*"Micropayments are back, at least in theory, thanks to P2P" **

- Payments of micro values (pennies or fractions of pennies).
- Several potential applications.
  - Ad-free web surfing, online gaming, and rewarding peers in peer-assisted services.
  - In **CacheCash**, a cache is paid per data chunk it serves.
- ***Drawbacks***; high transaction fees and large log size.

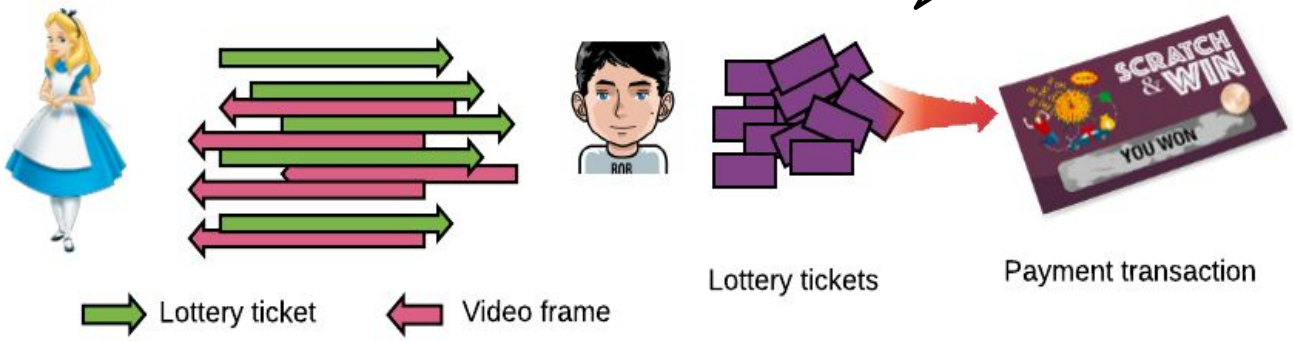


*Clay Shirky, The Case Against Micropayments, http://www.openp2p.com/pub/a/p2p/2000/12/19/micropayments.html

Probabilistic Micropayments

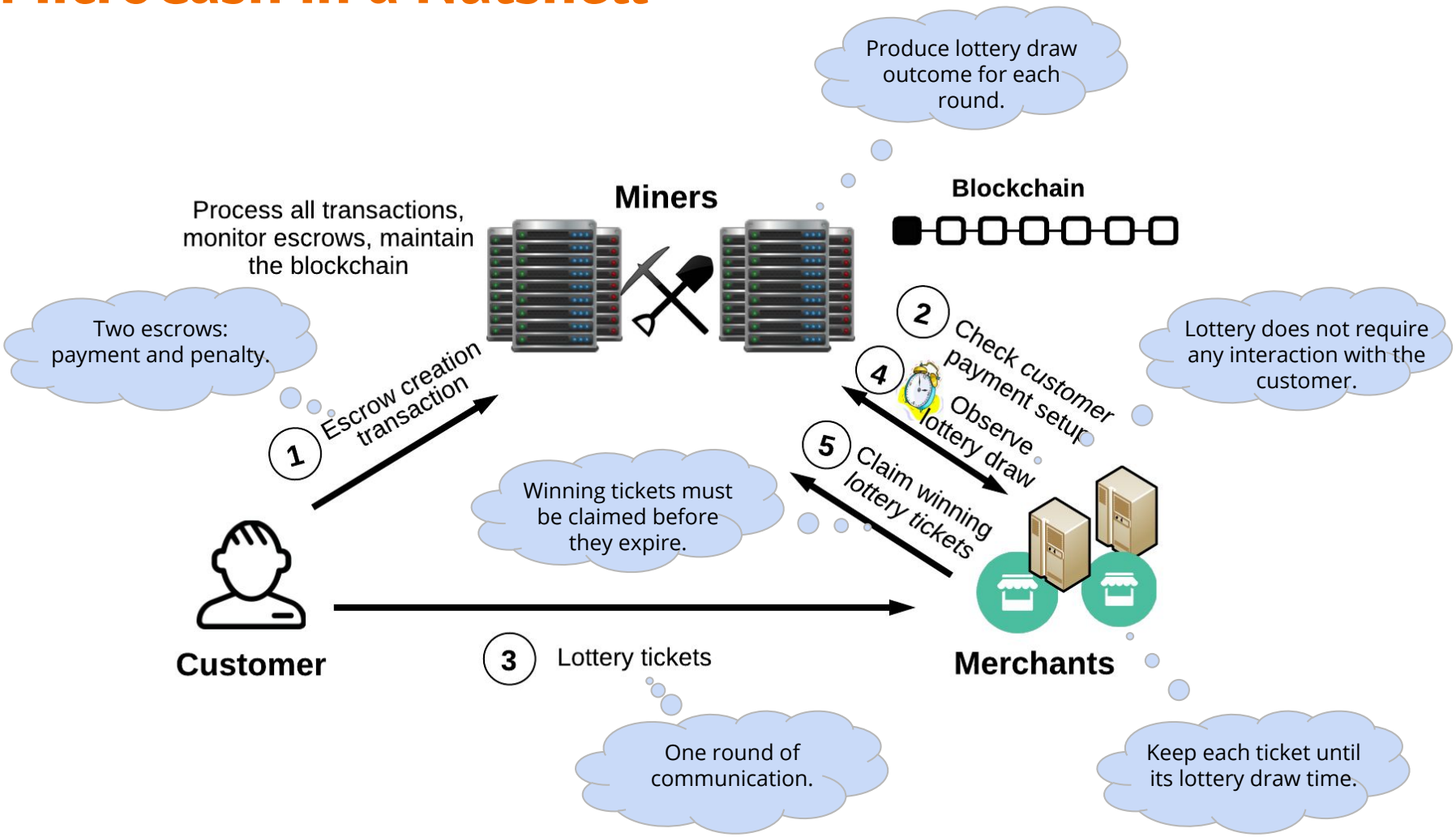
MicroCash addresses these limitations!!

- A solution to aggregate tiny payments.
- Dated back to Wheeler [W96] and Rivest [R97].



- Cryptocurrencies are utilized to build decentralized probabilistic micropayment schemes.
- Prior work: **MICROPAY** [PS15] and **DAM** [CGL+17]
  - Sequential, interactive lottery protocol, computationally-heavy.

MicroCash in a Nutshell



Escrows

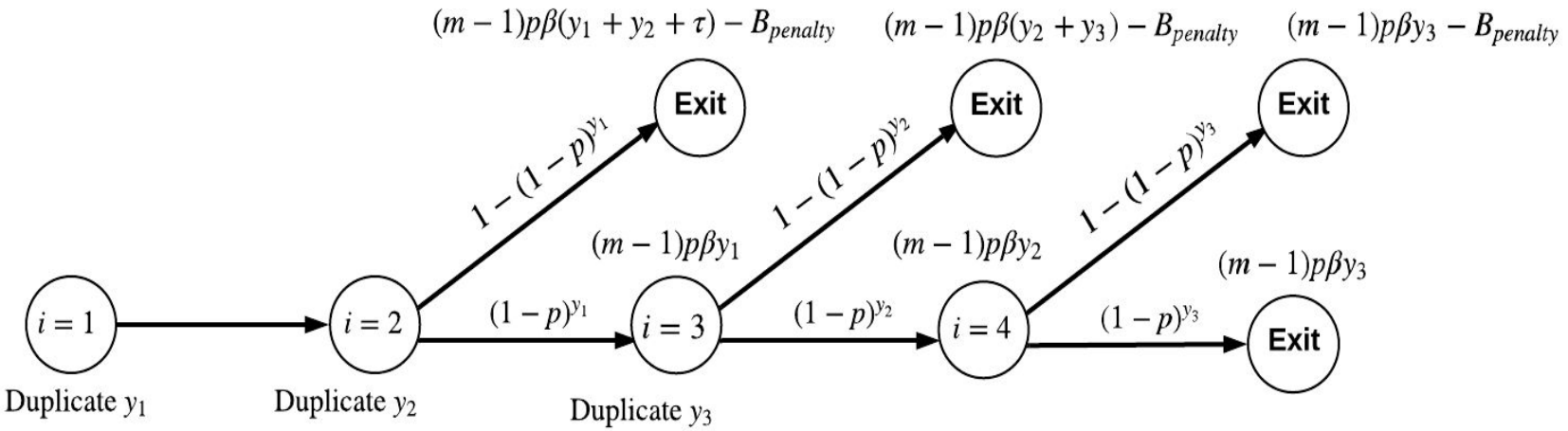
- **Payment escrow:**
  - To allow payment concurrency, the payment escrow balance must cover all winning tickets with high probability (1- ε).
- **Penalty escrow:**
  - Equals at least the additional utility gain a malicious customer obtains over an honest.
  - Intuitively, it is the expected amount of payments a customer would pay for (*m*-1) merchants (at max ticket issuance rate) during the cheating detection period.
- Cannot be withdrawn by the owner customer.
  - Escrows can be spent using a restricted set of transactions.

Payment Escrow

- Ticket winning events are independent.
- Number of winning tickets can be modeled as a binomial random variable.

$$B_{escrow} = \beta F^{-1}(p, l\tau, 1 - \epsilon)$$

Penalty Escrow I



$$\mathbb{E}_l[u(\widehat{C})] = \left(1-(1-p)^{y_1}\right)\left((m-1)p\beta\sum_{i=1}^{d} y_i + (m-1)p\beta r\tau - B_{penalty}\right) +$$

$$(1-p)^{y_1}\left((m-1)p\beta y_1 + \mathbb{E}_{l-1}[u(\widehat{C})]\right)$$

Penalty Escrow II

But $\mathbb{E}_{l-1}[u(\widehat{C})] \leq 0$ and $\mathbb{E}_{l}[u(\widehat{C})] \leq 0$, hence:

$$B_{penalty}(y_1, \ldots, y_d) \geq (m-1)p\beta\left(\frac{y_1}{1-(1-p)^{y_1}} + \sum_{i=2}^{d} y_i + r\tau \right)$$

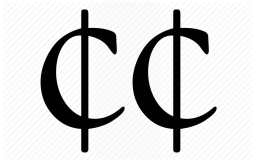
The above is maximized when $y_i = \tau, i \in \{1, \ldots, d\}$, thus:

$$B_{penalty} \geq (m-1)p\beta\tau\left(\frac{1}{1-(1-p)^{\tau}} + d + r - 1 \right)$$

- Putting It All Together -

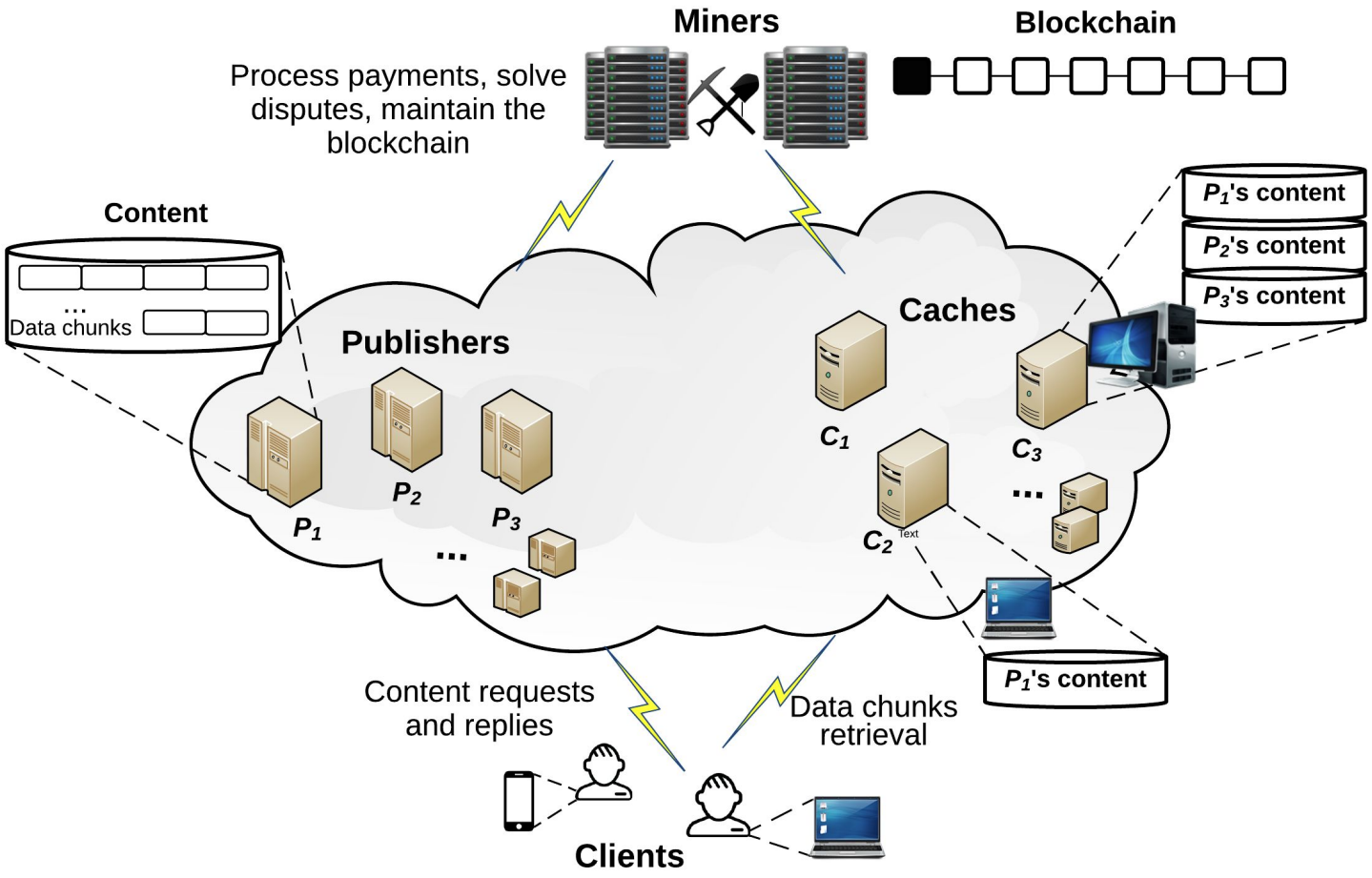
CacheCash

CacheCash



- A decentralized CDN system designed to address many of the limitations of previous solutions.
  - Creates a distributed, trustless bandwidth market.
  - Devises a novel service-payment exchange protocol that reduces the risks caused by malicious actors.
    - Pays a cache two lottery tickets per chunk instead of one, and ties and the currency value of these tickets together.
- Secure.
  - Employs cryptographic and financial approaches.
- Efficient.
  - Introduces several performance optimization techniques, and utilizes computationally-light primitives and protocols.

CacheCash Pictorially

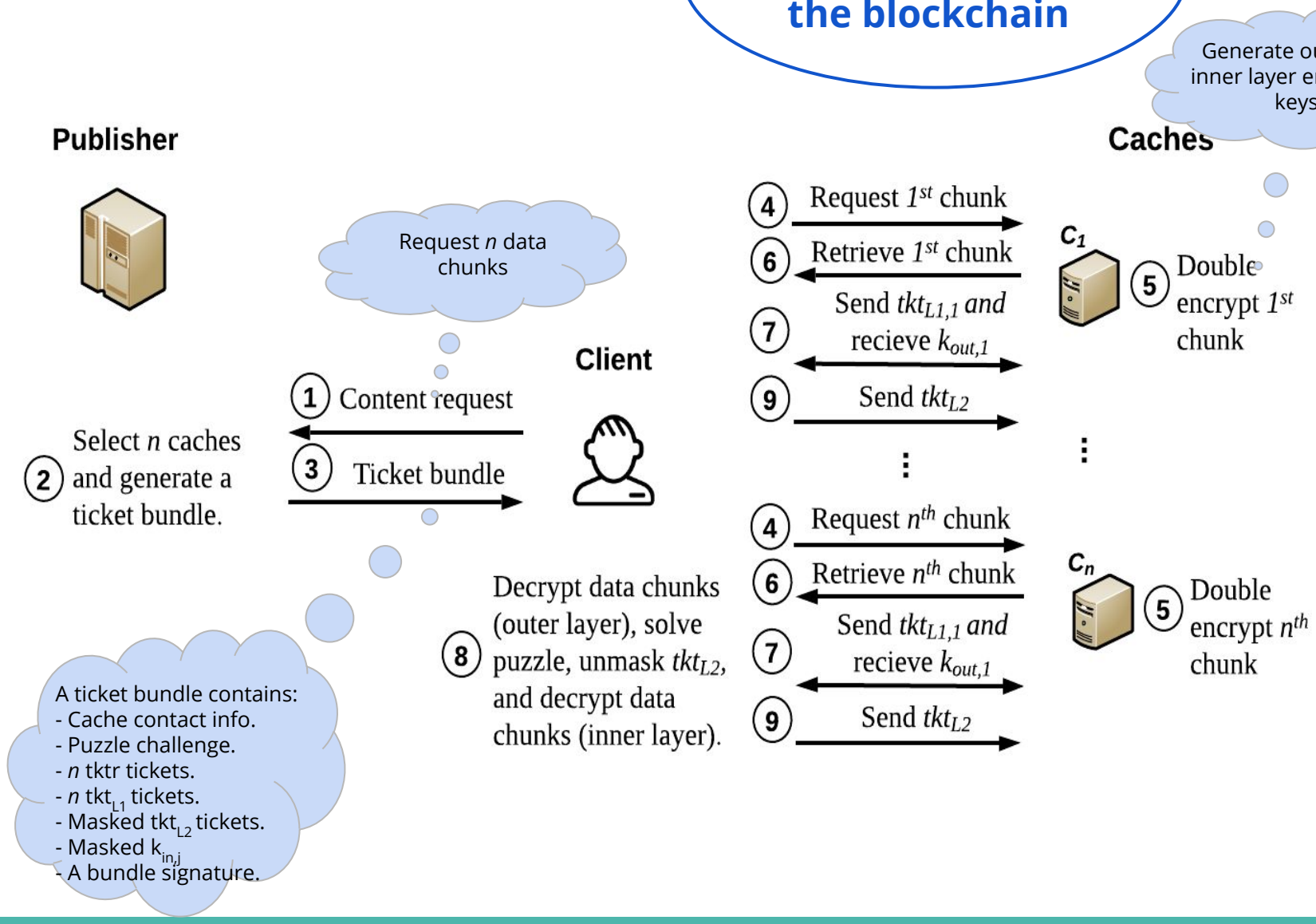


Service Setup

- **Publisher side:**
  - Advertises to recruit caches.
  - Creates payment and penalty escrow that list a set of beneficiary caches and payment setup parameters.
- **Cache side:**
  - Contacts a publisher to join its network.
  - Retrieves a copy of the content.
  - Shares a master secret key with the publisher.
- **Miner side:**
  - Processes the escrow creation transaction.
  - Creates a state for the new escrow to track its status.

Content Distribution

Starts as soon as the escrow is confirmed on the blockchain



Payment Processing

- A cache keeps each lottery ticket until the lottery draw time of that ticket.
- It observes the lottery draw outcome to determine whether this is a winning ticket (same as in MicroCash).
- The two-ticket model changes how payment value is computed:
  - A winning tkt_{L2} increases the (cumulative) counter a cache owns by 1.
  - A winning tkt_{L1} allows a cache to claim currency from the publisher's escrow, with value $f(z) = \alpha z$
- It also requires deriving new bounds for the balances of the payment and penalty escrows.

Payment Escrow

- Ticket winning events are independent.
- Number of winning tickets can be modeled as a binomial random variable.
- We apply a Chernoff bound and round slicing to reduce the required amount of funds while achieving the 1- ε coverage rule.

$$\omega = F^{-1}(p, tkt_{rate2}, 1 - \frac{\epsilon}{2l_{esc}})$$

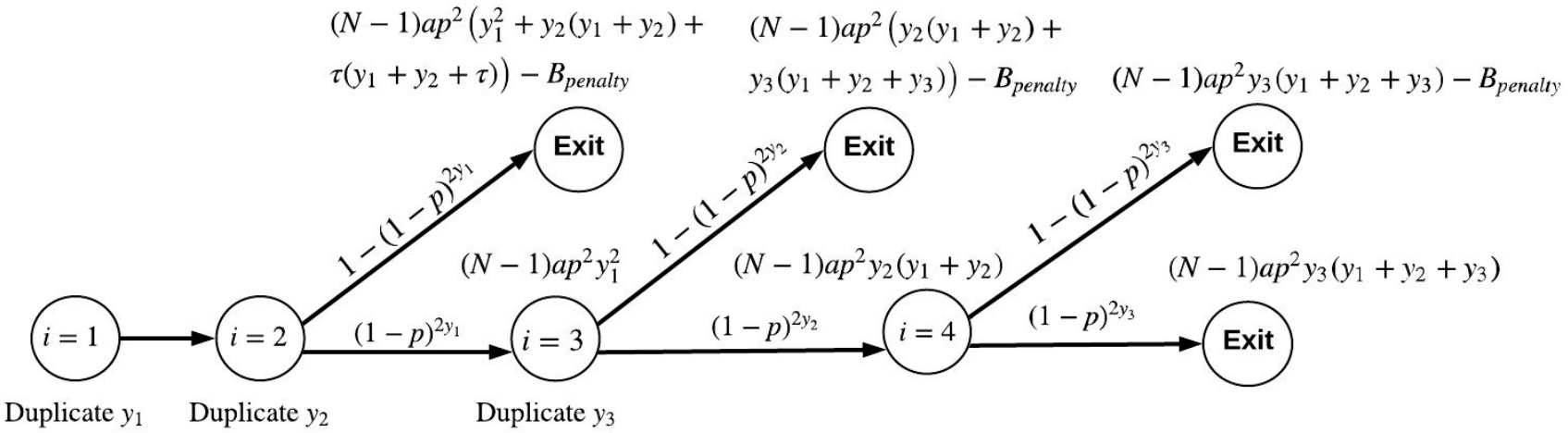
$$i^* \geq \frac{-3\ln(1-0.5\epsilon)}{p\,\epsilon^2\,tkt_{rate1}}$$

$$B_{escrow} = a\omega\left(\sum_{i=1}^{i^*-1} F_i^{-1}(p, i\,tkt_{rate1}, 1 - \frac{\epsilon}{2l_{esc}}) + \sum_{i=i^*}^{l_{esc}} p\,i\,tkt_{rate1}\right)$$

$$= a\omega \sum_{i=1}^{i^*-1} F_i^{-1}(p, i\,tkt_{rate1}, 1 - \frac{\epsilon}{2l_{esc}}) \quad +$$

$$0.5\,a\,\omega\,p\,tkt_{rate1}\left(l_{esc}(l_{esc}+1) - i^*(i^*-1)\right)$$

Penalty Escrow I



$$\mathbb{E}_{l,\mathbf{c}}[u(\hat{P})] = \left(1-(1-p)^{2y_1}\right)\phi_{\mathbf{c}} + (1-p)^{2y_1}\left(apy_1(c+(N-1)py_1) + \mathbb{E}_{l-1,\mathbf{c}'}[u(\hat{P})]\right)$$

$$\phi_{\mathbf{c}} = ap\left(\sum_{i=1}^{d} y_i\left(c+(N-1)\sum_{j=1}^{i} py_j\right)\right) +$$

$$ap\left(\sum_{i=d+1}^{d+r} \tau\left(c+(N-1)\sum_{j=1}^{d} py_j + (N-1)(i-d)p\tau\right)\right) - B_{penalty}$$

Penalty Escrow II

But $\mathbb{E}_{l-1}[u(\hat{\mathcal{P}})] \leq 0$ and $\mathbb{E}_{l}[u(\hat{\mathcal{P}})] \leq 0$, hence:

$$B_{penalty}(c, y_1, \ldots, y_d) \geq \frac{apy_1\left(c+py_1(N-1)\right)}{1-(1-p)^{2y_1}} + ap\sum_{i=2}^{d} y_i\left(c + (N-1)\sum_{j=1}^{i} py_j\right) + ap\sum_{i=d+1}^{d+z}\tau\left(c + (N-1)\sum_{j=1}^{d} py_j + (i-d)(N-1)p\tau\right)$$

The above is maximized when $c = p\tau(l-d-r), y_i = \tau, i \in \{1, \ldots, d\}$, thus:

$$B_{penalty} \geq ap^2\tau^2\left(\frac{l-d-r+N-1}{1-(1-p)^{2\tau}} + (l-d-r)(d+r-1) + 0.5(N-1)(d+r)(d+r+1) - (N-1)\right)$$

CacheCash Security Properties

- Addresses **service corruption**.
  - Done financially, serving invalid chunks prevents unmasking tkt_{L2}.
- Defends against **cache accounting attacks** and other **payment related threats**.
  - By the security of CAPnet and MicroCash.
  - The case of a cache collecting only tkt_{L1} is handled financially.
- Handles **service theft attacks**.
  - Devise financial techniques to deal with ticket duplication, issuing invalid payments, and not paying out tkt_{L2}.

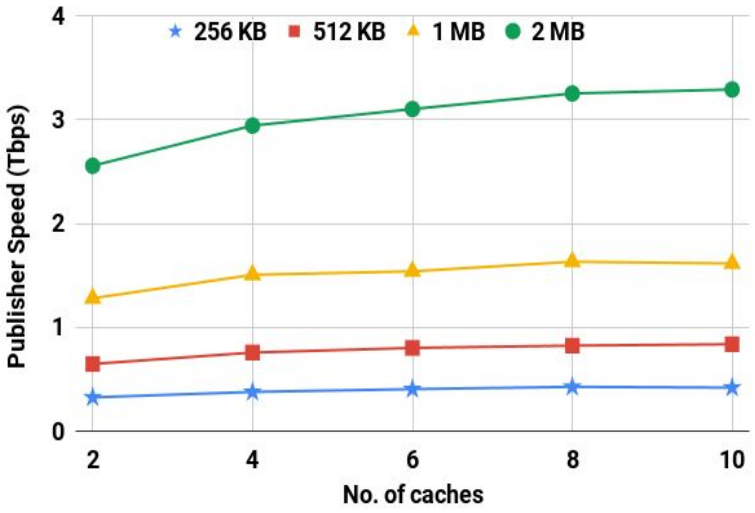
Financial Defenses

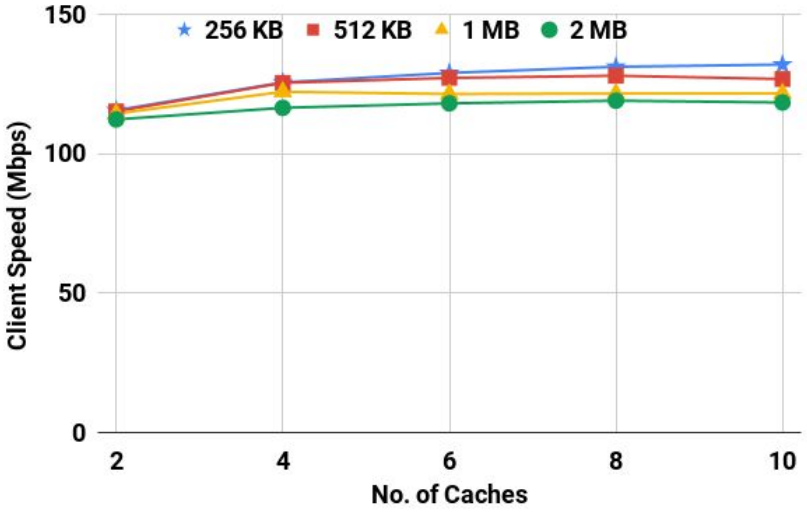
- Models the service as a repeated game between the same set of parties.
- Analyzes publisher collusion case and caches collusion case.
  - Compare the utility gain of a malicious publisher (cache) to an honest one.
- **Cache Collusion**, devise a suitable payment function, *f(z)* = *αz* such that:

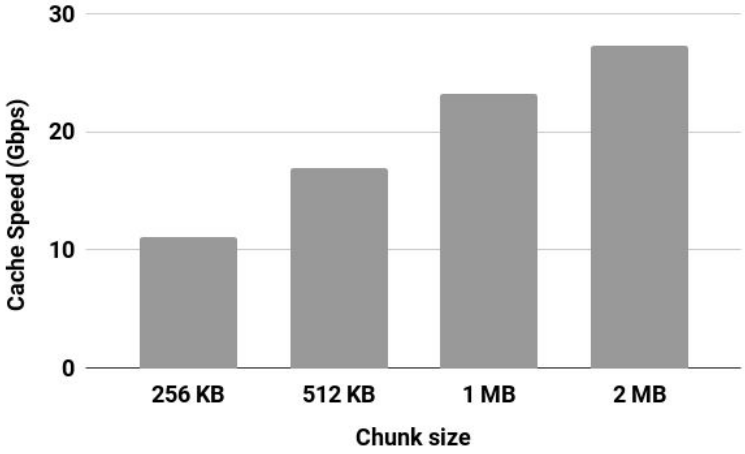
$$a \geq \max_{\substack{j \in \{1,\ldots,N\} \\ i \in \{1,\ldots,l_{esc}\}}} \frac{D_{cost}}{p^2 S_i^j}$$

- **Publisher Collusion**, caches give higher priority to an honest publisher clients than a malicious one.
  - Eventually leave its network.

CacheCash Efficiency







** A modest publisher can handle requests at a rate sufficient to serve **315,780** clients watching the same 1080p video concurrently.

** A modest client is able to retrieve content at a rate of **122 Mbps** (watch **24** 1080p videos concurrently).

** Bandwidth overhead is less than **0.1%**

Last Stop

Future Directions

- Optimize service price and collateral cost.
  - Devise a payment function with a slower growth rate.
  - Modify the lottery protocol in a way that reduces the needed payment escrow balance.
- Preserve user privacy.
- Fault tolerance and cache selection.
- Full system implementation and deployment.

Conclusion

- Cryptocurrencies have provided new templates for reshaping large-scale distributed systems and services.
- CacheCash targets online content distribution.
  - Creates a distributed bandwidth market.
  - Though this idea has been around for a long time, all prior solutions suffered from various drawbacks that restrained practicality.
- CacheCash addresses these drawbacks by introducing several building blocks.
  - ABC, CAPnet, MicroCash, financial defenses, etc.
- Benchmark results show that modest machines can serve/retrieve content at a high bitrate with minimal bandwidth overhead.


THANK YOU

References

[R97] Ronald Rivest.1997. Electronic lottery tickets as micropayments. In International Conference on Financial Cryptography. Springer, 307–314.

[W96] David Wheeler. 1996. Transactions using bets. In International Workshop on Security Protocols. Springer, 89–92.

[PS15] Pass, Rafael, Abhi Shelat. "Micropayments for decentralized currencies." In Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, pp. 207-218. ACM, 2015.

[CGL+17] Chiesa, Alessandro, Matthew Green, Jingcheng Liu, Peihan Miao, Ian Miers, and Pratyush Mishra. "Decentralized Anonymous Micropayments." In Annual International Conference on the Theory and Applications of Cryptographic Techniques, pp. 609-642. Springer, Cham, 2017.

Limitations of Existing Solutions

System	CDN-specific	Decentralized	Trustless Publishers	Incentive Type	Sponsoring content retrieval for clients	Address cache accounting attacks
KARMA [1]	✔	✔	✔	Bandwidth	✘	N/A
Floodgate [2]	✔	✔	✘	Monetary	✔	N/A
Dandelion [3]	✔	✔	✘	Hybrid	✔	N/A
Hincent [4]	✔	✘	✔	Hybrid	✔	N/A
Storj [5] & Filecoin [6]	✘	✔	✔	Monetary	✘	N/A
Torcoin [7] & Mysterium [8]	✘	✔	✔	Monetary	✘	N/A
Gringotts [9]	✔	Partial	✔	Monetary	✔	N/A
NIOS [10]	✔	Partial	✔	Monetary	✔	Requires a trusted CDN node
CacheCash	✔	✔	✔	***Monetary***	✔	✔

Thesis Statement

Distribute content without alteration.
Pay caches as promised.
Ensure that caches has earned their payments.

Optimize performance in terms of bandwidth and computation overhead.

Lower service cost than traditional solutions.

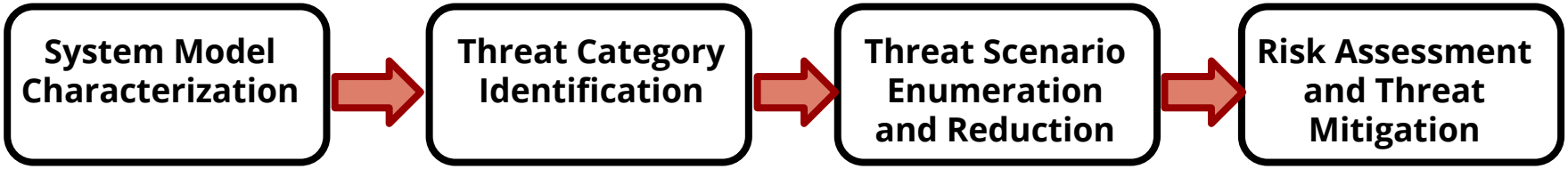
Design a secure, efficient, and low cost decentralized CDN service that addresses the previous challenges.

No centralized entity, and does not place trust in anyone.

Non-goal:
**Digital rights management.
**Preserving user's privacy.

More ABC

ABC Steps



Outline system use cases, modules, participant roles, its assets, etc.

Define broad threat categories that must be investigated.

Define all scenarios that attackers may follow to pursue their goals.

Prioritize threat cases and design mitigation techniques to secure the system.

Threat Category Identification Example

Asset	Security Threat Category
Service	Service corruption (provide corrupted service for clients).
	Denial of service (make the service unavailable to legitimate users).
	Information disclosure (service content/related data are public).
	Repudiation (the server can deny a service it delivered).
Service payments	Service slacking (a server collects payments without performing all the promised work).
	Service theft (a client obtains correct service for a lower payment than the agreed upon amount).
Blockchain	Inconsistency (honest miners hold copies of the blockchain that may differ beyond the unconfirmed blocks).
	Invalid blocks adoption (the blockchain contains invalid blocks that does not follow the system specifications).
	Biased mining (a miner pretends to expend the needed resources for mining to be elected to extend the blockchain).
Transactions	Repudiation (an attacker denies issuing transactions).
	Tampering (an attacker manipulates the transactions in the system).
	Deanonymization (an attacker exploits transaction linkability and violates users' anonymity).
Currency	Currency theft (an attacker steals currency from others in the system).
Communication network	Denial of service (interrupt the operation of the underlying network).

Step 2: Running Example Application

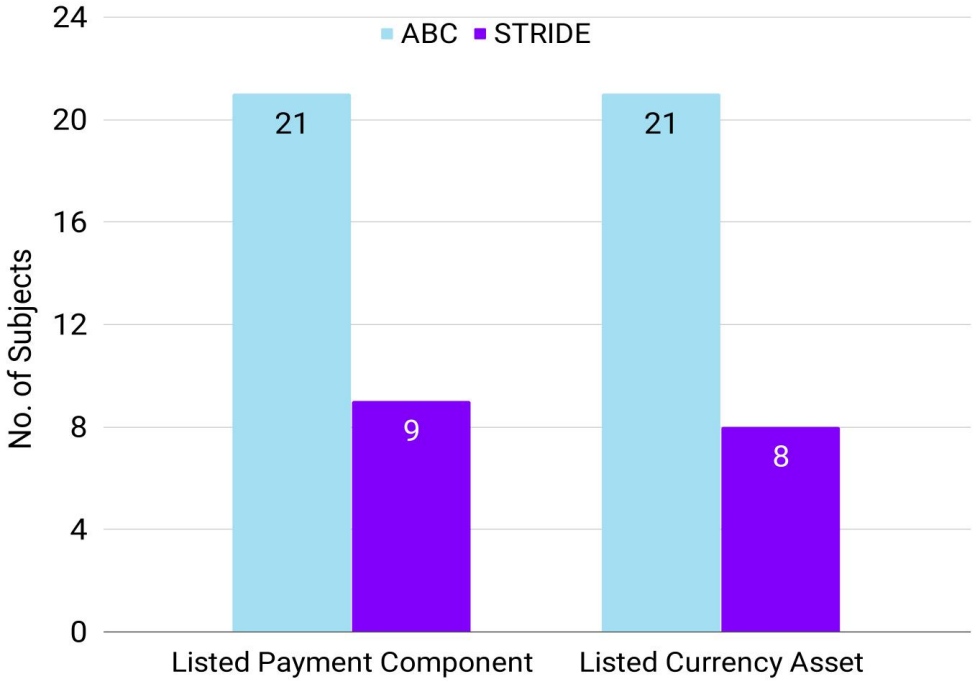
Service Theft Threat Collusion Matrix

Attacker \ Target	*Client*	*Server*	*Client and Server*
External	**Clients cannot be targets because they do not serve others.**	**Servers and external cannot attack because they do not ask/pay for service.**	**Reduced to the case of attacking servers only, clients do not serve others (cannot be targets).**
Server			
Server and External			
Client		**(1)** Refuse to pay after receiving the service. **(2)** Issue invalid payments.	
Client and External		**Reduced to the case of an attacker client. A client does not become stronger when colluding with other servers or external entities.**	
Server and Client			
Client, Server, and External			

User Study - ABC vs. STRIDE

- Recruited 53 participants (mainly security masters students).
  - 5 pilot run, two groups of 24 subjects (one tested STRIDE, one tested ABC).
- Asked to build a threat model for a cryptocurrency-based file storage and retrieval network called ArchiveCoin.
- Each session spanned 3 hours.
  - Overview of cryptocurrencies.
  - A tutorial for ABC or STRIDE.
  - Overview of ArchiveCoin.
  - Threat model building.

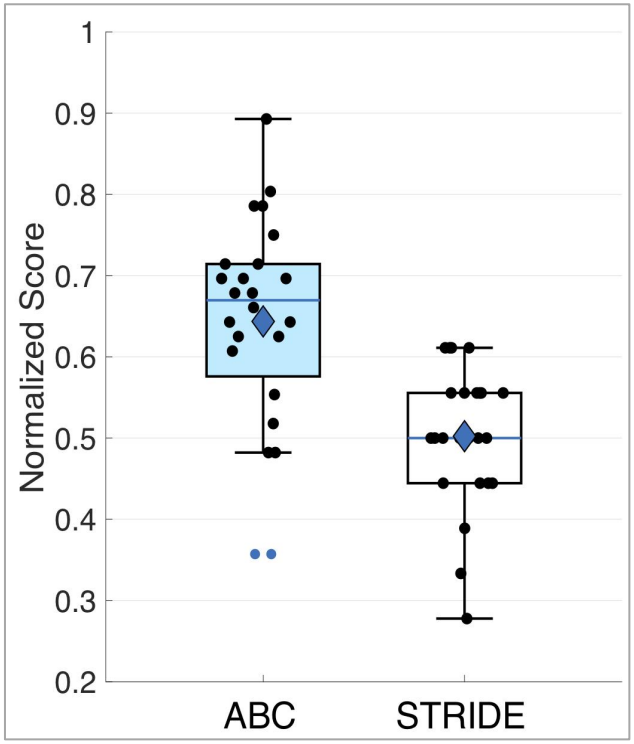
Results - Financial Aspects and Collusion



- For financial threat in question (service theft of file retrieval):
  - STRIDE 13%, ABC 71%.
- For collusion: none in STRIDE, while 45% in ABC.

Results - Accuracy

- Computed precision, recall, and total score.
  - Precision -- STRIDE 0.48, ABC 0.57
  - Recall -- STRIDE 0.4, ABC 0.48
  - Total scores (normalized).
    - STRIDE avg 0.5, ABC avg 0.64



Use Cases

- Applied ABC to three real world systems.
  - Bitcoin - well established system.
  - Filecoin - close to launch.
  - CacheCash - our system, under development.

- We developed ABC while working on CacheCash when we realized that none of traditional frameworks suited our needs.

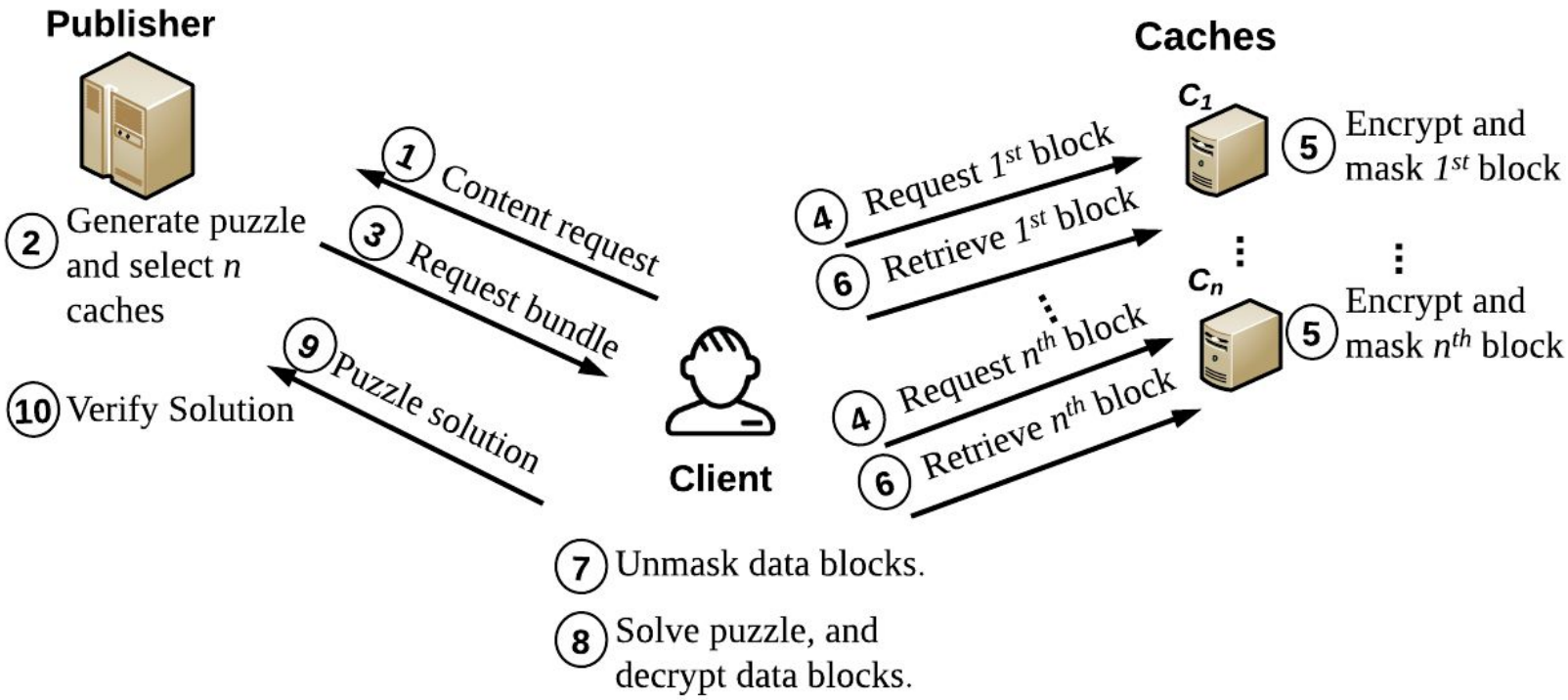
Use Cases - Outcome

Aspect	Bitcoin	Filecoin	CacheCash
ABC steps covered	Steps 1-3	Seps 1-3	Steps 1-4
Completion time (hr)	10	47	Not tracked
No. of collusion matrices	5	14	9
Threat cases total	105	882	525
Distilled threat cases	10	35	22

- All known threats to Bitcoin were mapped to the collusion matrices ABC produced.
- Revealed ***3*** *unaddressed threats* in the public design of Filecoin.
- ABC was useful for CacheCash in both pre-design threat modeling step, and after-design security analysis.

More CAPnet

CAPnet Work Model



Security Analysis

- Define a δ-bound, the ratio between the number of pieces a puzzle solver retrieves out of the total number of pieces in the requested chunks.
  - E.g., 0.9-bound means that 90% of the content will be retrieved in order to solve the puzzle.
- A publisher can set a specific bound by configuring the number of puzzle rounds.
  - Also, needs to configure the piece size.
- Simulation-based analysis while assuming a full knowledge of piece selection frequency.

Security Analysis II

- We assume a strong adversary that knows the frequency distribution of all pieces in all data chunks.
- A client is colluding with a set of malicious caches, Cm, of size m < n.
- One will be the puzzle solver and one will be the piece provider.
- A client always retrieve chunks from honest caches.
- Set piece size <= hash size/m
- Using simulation, we determine the number of puzzle rounds based on the desired δ-bound.

Parameter Setup - An Example

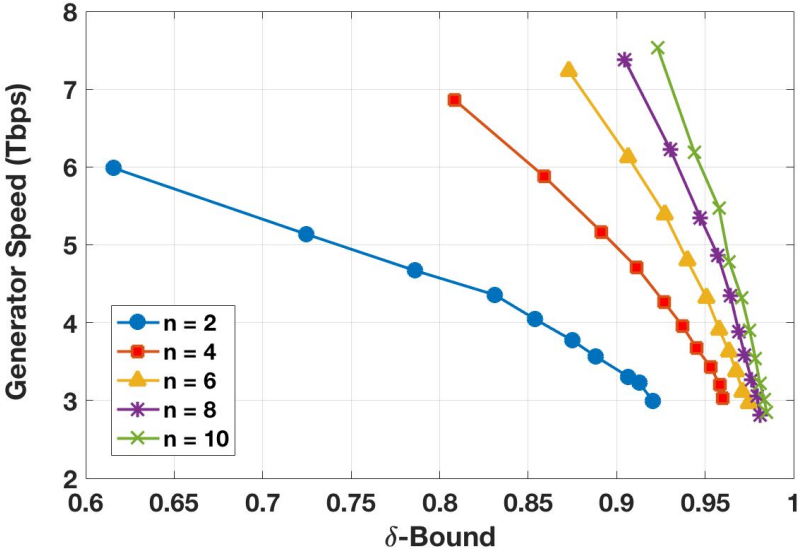
- 1 MB chunk size, 16-byte piece size, n = 6 caches.

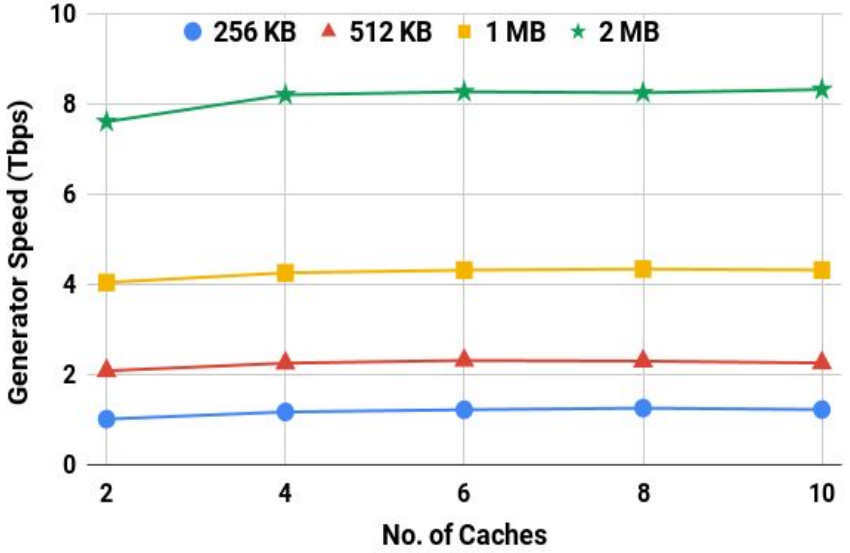
	Client as solver			Either	Cache as solver		
R \ m	**0**	**1**	**2**	**3**	**4**	**5**	**6**
1	1	0.87±0.03	0.78±0.06	0.71±0.08	0.45±0.06	0.21±0.03	0
2	1	0.91±0.04	0.86±0.06	0.82±0.08	0.52±0.06	0.24±0.04	0
3	1	0.93±0.04	0.9±0.05	0.87±0.07	0.57 ±0.05	0.26±0.04	0
4	1	0.94±0.03	0.92±0.05	0.91±0.06	0.59±0.05	0.28±0.03	0
5	1	0.95±0.03	0.94±0.04	0.93±0.04	0.6±0.05	0.29±0.03	0
6	1	0.96±0.03	0.95±0.04	0.94±0.04	0.61±0.04	0.29±0.03	0
7	1	0.96±0.02	0.95±0.02	0.95±0.04	0.62±0.04	0.3±0.03	0
8	1	0.97±0.02	0.96±0.03	0.95±0.03	0.63±0.03	0.3±0.02	0
9	1	0.97±0.02	0.97±0.03	0.96±0.03	0.63±0.03	0.3±0.02	0
10	1	0.97±0.02	0.97±0.03	0.97±0.03	0.64±0.03	0.31±0.02	0

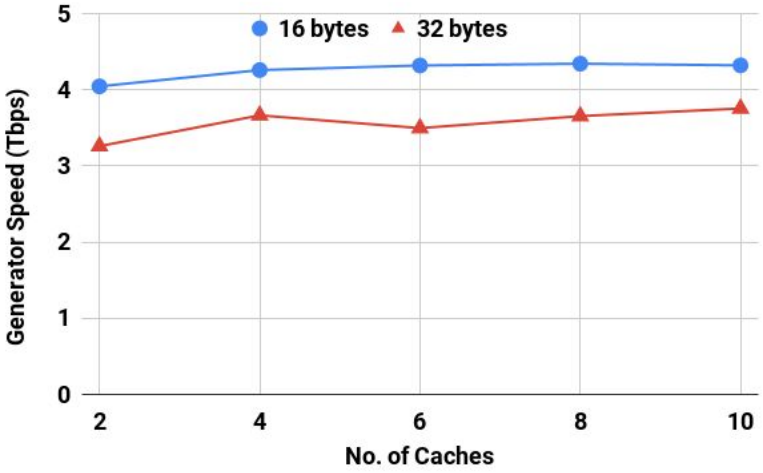
Performance Evaluation

- Benchmarks to evaluate puzzle generation and solving rate.
  - Represented in terms of content bitrate.
- Study the effect of puzzle rounds (or δ bound), chunk size, and piece size.

CAPnet Efficiency - Generator

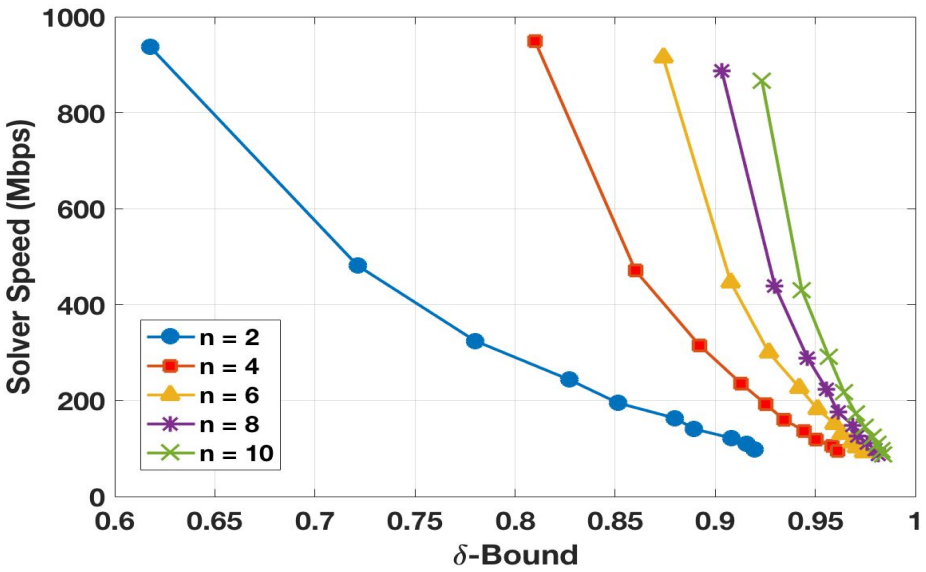


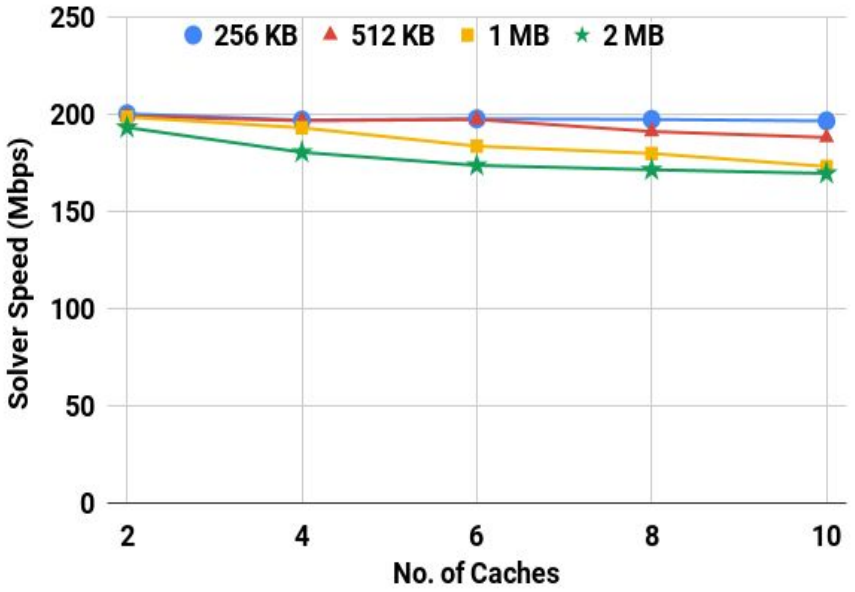


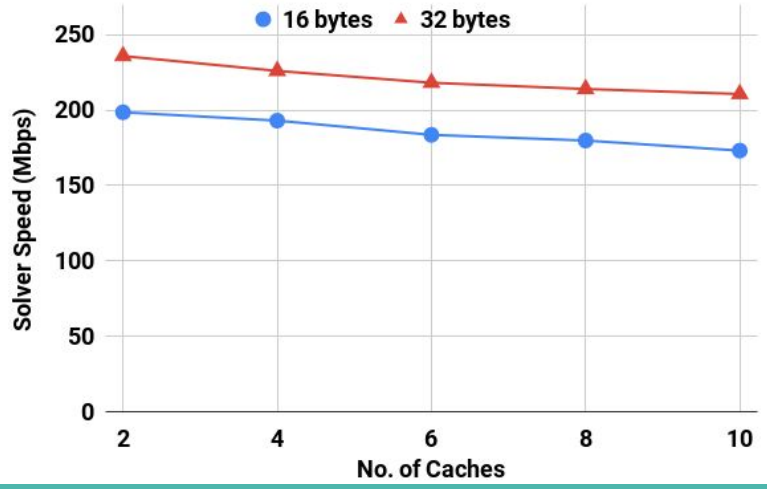


A publisher can generate puzzles sufficient to serve 870,000 clients watching the same 1080p video concurrently.

CAPnet Efficiency - Solver



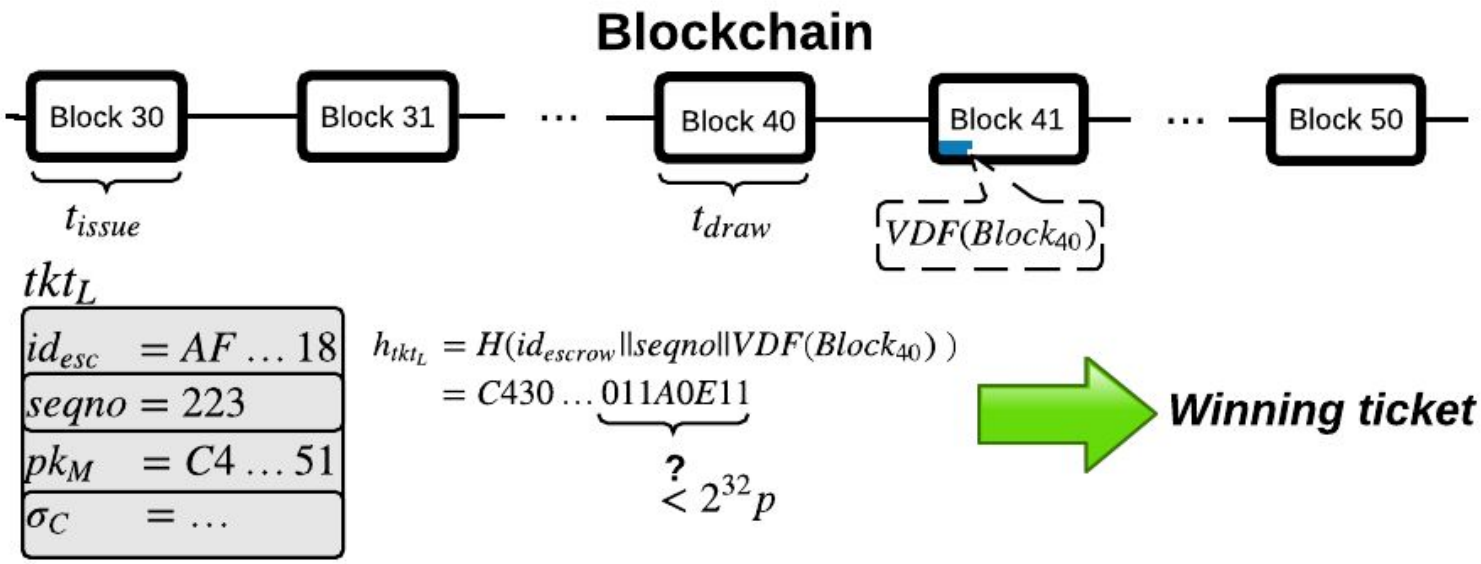




A client can solve puzzles sufficient to retrieve 34 1080p videos concurrently.

More MicroCash

The lottery Protocol

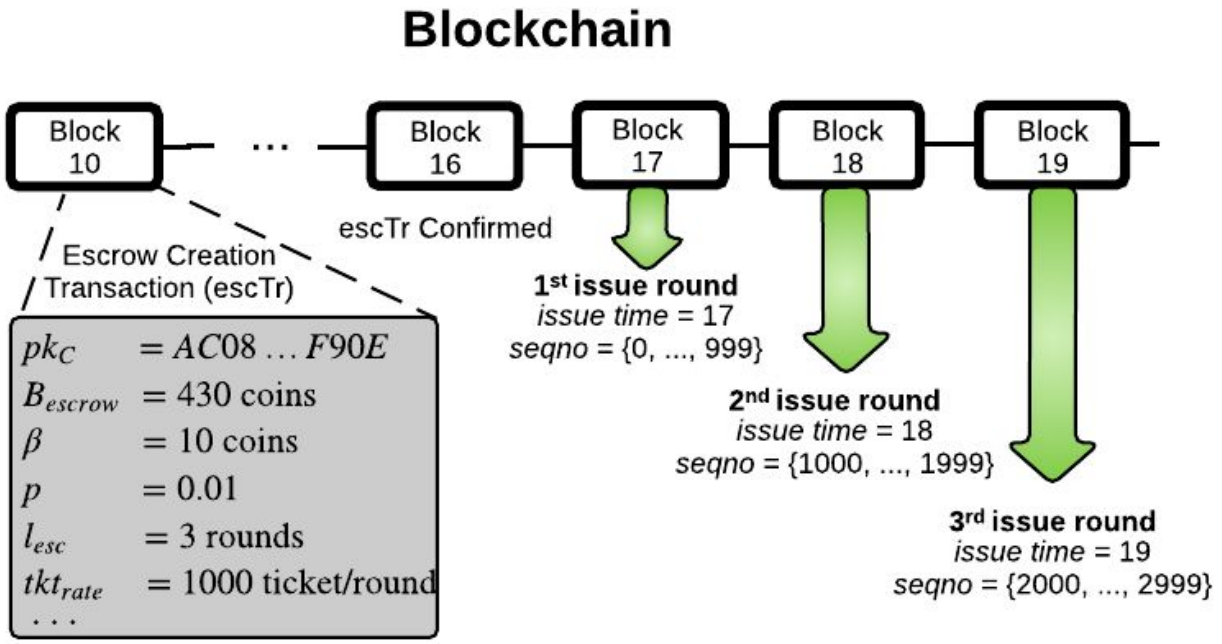


Lottery Ticket Issuance

- Each ticket is a simple structure consist of:

$$tkt_L = id_{esc} || pk_M || seqno || \sigma_C$$

- Ticket issuance must follow a ticket issuing schedule.



Escrow Balances - Example

$l = 200$ rounds, $\tau = 1000$ tickets, $p = 0.01$,
$\beta = 1$ coin, $m = 5$ merchants, $d = 6$ rounds,
$r = 6$ rounds, and $\epsilon = 0.01$.

$B_{escrow} = 2,104$ coins
(Note that the expected total payments, $\epsilon = 0.5$, is 2,000 coins)
$B_{penalty} \geq 480$ coins

MicroCash Security Properties

- Prevents **escrow overdraft**.
  - Front running attacks are not possible.
  - Ticket tracking prevent issuing more tickets than what can be covered.
- Prevents **escrow-withholding**.
  - An escrow will be refunded once all tickets expire.
- Prevents **manipulating the lottery** outcome.
  - Achieved by the use of VDF and ticket issuing schedule.
- Addresses **duplicated ticket issuance**.
  - Using detect-and-punish approach.

MicroCash Efficiency

- Compared with a sequential micropayment scheme, MICROPAY.
- Computational cost.
  - Increases ticket processing rate by **1.67 - 4.1x**.
- Effect of micropayment concurrency.
  - Amount of data logged on the blockchain: ~**50%** reduction.
- Bandwidth cost (in terms of lottery ticket size).
  - From customer to merchant: **48%** reduction.
  - From merchant to miner: **60%** reduction.

MicroCash Efficiency - MicroBenchmarks I

- **Ticket processing rate (**ticket / sec**):**

Scheme	ECDSA (secp256k1)	ECDSA (P-256)	EdDSA (Ed25519)
MICROPAY			
Customer	1891	32606	20884
Merchant	1353	2530	2509
Miner	1365	2591	2565
MicroCash			
Customer	1890	32978	20879
Merchant	2266	10463	7825
Miner	2266	10463	7825

Merchants and miners in MicroCash are **1.67x, 4.1x, and 3.1x** faster than in MICROPAY (for the three digital signature schemes shown above).

MicroCash Efficiency - MicroBenchmarks II

- **Bandwidth cost (in terms of ticket size):**
  - From customer to merchant; 274 bytes (MICROPAY), 142 byte (MicroCash, **48% reduction**).
  - From merchant to miner; 355 byte (MICROPAY), 142 bytes (MicroCash, **60% reduction**).
- **Number of escrows**:
  - MICROPAY needs **60, 1019, and 653 escrows** to support the rates reported previously.
  - MicroCash needs only ***one escrow***.

Real World Applications - Online Gaming

- **Bitcoin:** Average transaction fee is $0.068, and average transaction size is 250 bytes.
- **Minecraft:** 125 servers, each serving 8 players. Cost is $12 per 8 players per month.
  - With 2% overhead percentage, p = 0.00001
  - Each player pay one ticket per minute.
  - β = $3.472

Metric	Bitcoin	MICROPAY	MicroCash
Winning tickets / sec	N/A	0.000167	0.000167
Escrows / sec	N/A	0.000552	0.000386
Transactions /sec	16.67	0.000719	0.000552
Transaction fees / round	$680	$0.02934	$0.02254
Bandwidth between players and miners	3,333 bps	1.105 bps	1.0093 bps
Bandwidth between players and servers	N/A	36,533 bps	18,933 bps
Bandwidth between servers and miners	N/A	0.807 bps	0.523 bps
Delta blockchain size / round	2.38 MB	0.000137 MB	0.00011 MB

Real World Applications - Peer-assisted CDN

- **CDN:** one publisher serving 1 Gpb, cost is $0.0067, each cache gets a ticket per 1 MB it serves.
  - With 2% overhead percentage, p = 0.000015
  - Issues 128 tickets per second
  - β = $3.4

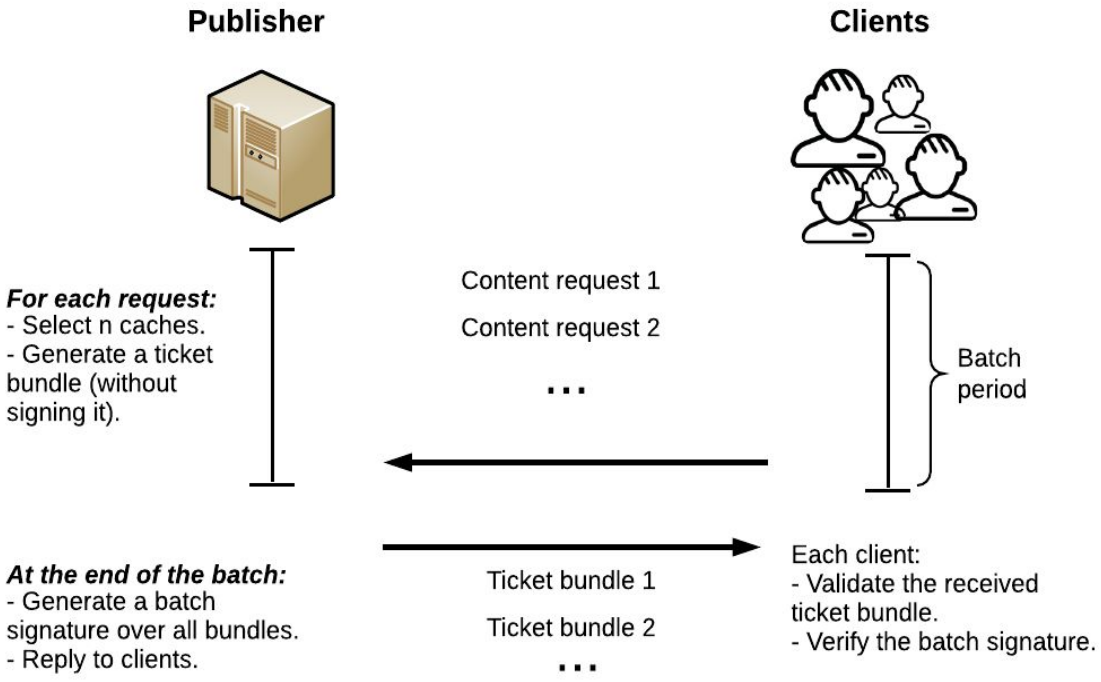
Metric	Bitcoin	MICROPAY	MicroCash
Winning tickets / sec	N/A	0.001964	0.001964
Escrows / sec	N/A	0.001976	0.00001157
Transactions /sec	128	0.00394	0.001976
Transaction fees / round	$5,222	$0.160769	$0.08062
Bandwidth between publisher and miners	256,000 bps	3.95 bps	0.165 bps
Bandwidth between publisher and caches	N/A	280,576 bps	145,408 bps
Bandwidth between caches and miners	N/A	9.508 bps	6.16 bps
Delta blockchain size / round	18.31 MB	0.000963 MB	0.000452 MB

More CacheCash

Bundle Signature

- Instead of signing each ticket individually, a publisher does the following:
  - Hash each ticket individually.
  - Hash all these hashes to produce a bundle hash.
  - Sign the bundle hash.
- A client can verify this signature because it receives the full ticket bundle all at once.
- A cache receives a copy of the signature and the bundle hash with tktr.
  - Verify signature over tickets by looking for an identical hash.

Batch Signature



- Compute a Merkle tree of all bundle hashes, sign the root, and provide a membership path for each bundle.

CacheCash Efficiency II

- Bundle/Batch signature boost publisher's speed by 7.2x and 22.7x, respectively, over individual ticket signing.

Signing Approach	Publisher (Tbps)	Cache (Gbps)	Client (Mbps)
Individual tickets	0.064	11.34	121.92
Individual bundles	0.46	23.55	122.56
Batch (64)	1.43	23.5	121.92
Batch (128)	1.51	23.24	122.24
Batch (256)	1.44	23.45	121.28
Batch (512)	1.48	23.35	120.64
Batch (1024)	1.45	23.47	121.28

CacheCash Efficiency - Comparison with MicroCash

Metric	MicroCash	CacheCash
Escrows / sec	0.00001157	0.00001157
Tickets / sec	128	160
Winning tickets / sec	0.001964	0.002456
Transactions / sec	0.001976	0.00394
Bandwidth between publisher and miners	0.16093 bps	0.1699 bps
Bandwidth between caches and miners	6 bps	27.3 bps
Delta blockchain size / round	0.00044 MB	0.00123 MB

- Bandwidth overhead cache/miner: CacheCash incurs 3.9x MicroCash's cost.
- Delta blockchain size: CacheCach adds 4.6x the amount of data MicroCash adds.
- Still the overall bandwidth cost is less than 0.1%