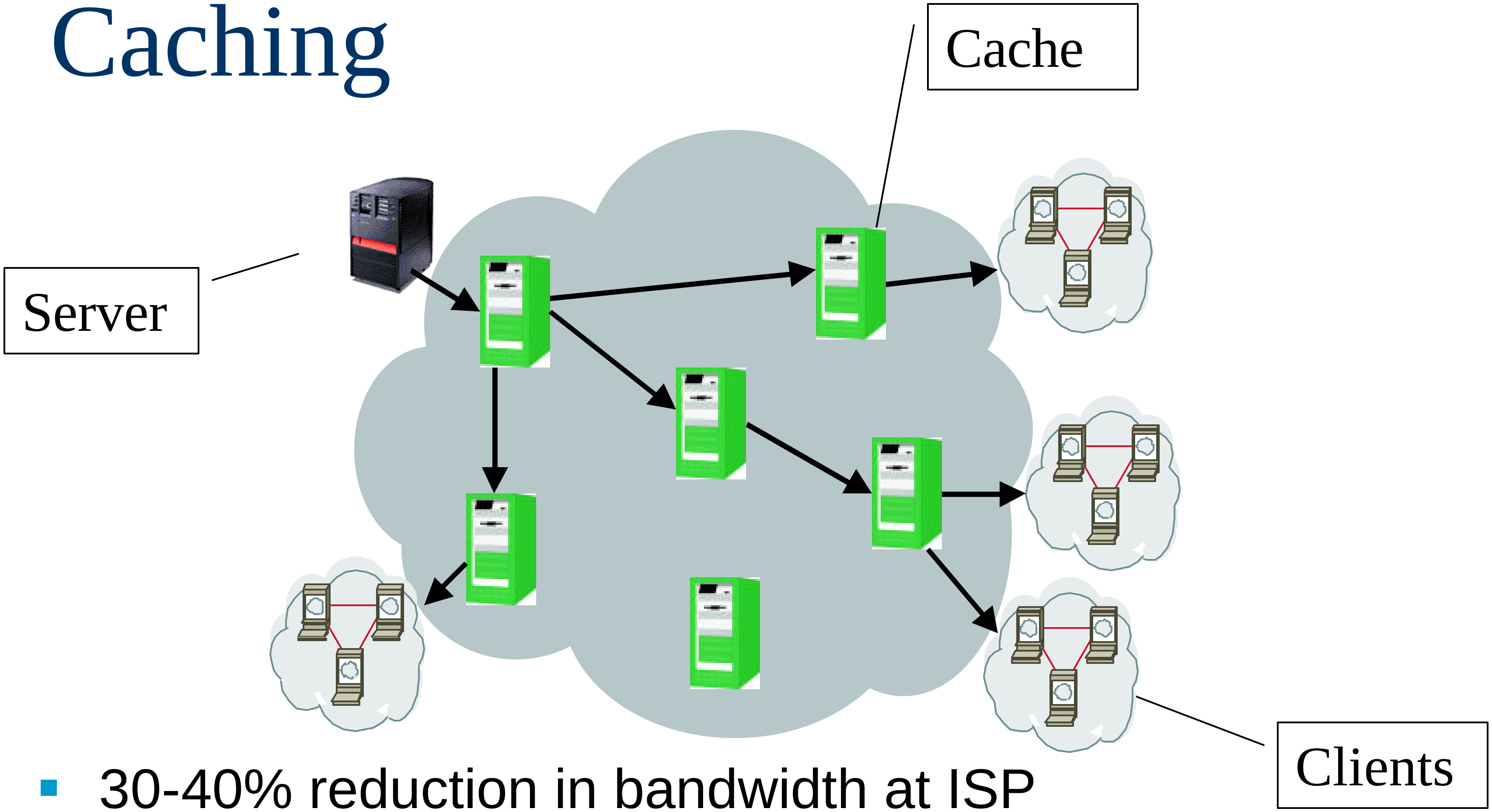
# A Secure, Publisher-Centric Web Caching Infrastructure

Andy Myers, John Chuang, Urs Hengartner, Yinglian Xie, Weiqiang Zhuang, Hui Zhang

Infocom 2001

# Caching



- 30-40% reduction in bandwidth at ISP
- Reduced waiting time for users
- Lower load on publisher’s server

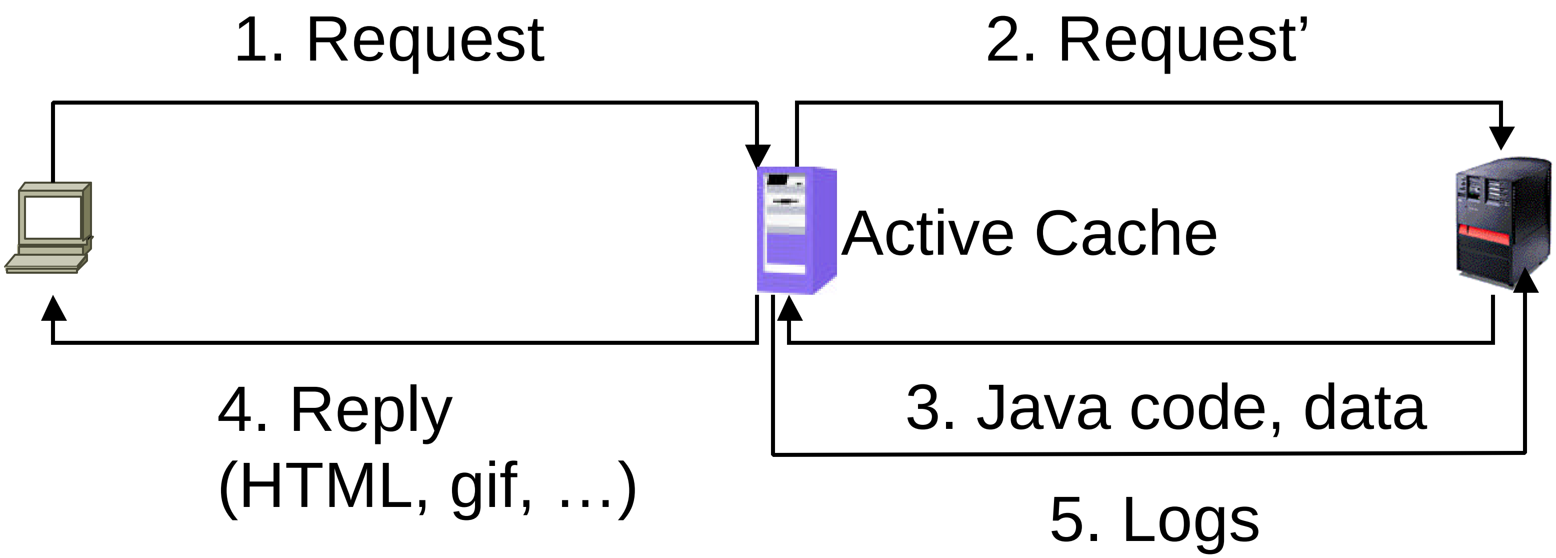
# But…

- Caches don’t meet publishers’ demands
- Logging of user accesses
  - Publishers routinely “cache bust” to get log information
- Generation of dynamic content
  - Lots of content uncacheable because it has a dynamic component
- Result: reduction in performance

# Make cache publisher-centric

- Do a bit for the publisher, get back a big performance increase
- Need to increase flexibility
- Solution: Java!
  - Publisher writes cache applets to generate content
  - Can perform custom logging

# Gemini



- Active cache generates reply for client based on code sent by publisher
- Later, cache returns access logs

# Example applications

- MyYahoo
  - Cache assembles preset components
  - Cache could act as front-end for publisher database
- AmIHotOrNot.com
  - Caches send ratings feedback in logs
- Content adaptation
  - 56K vs. DSL vs. WAP
  - Cache thins content for constrained client

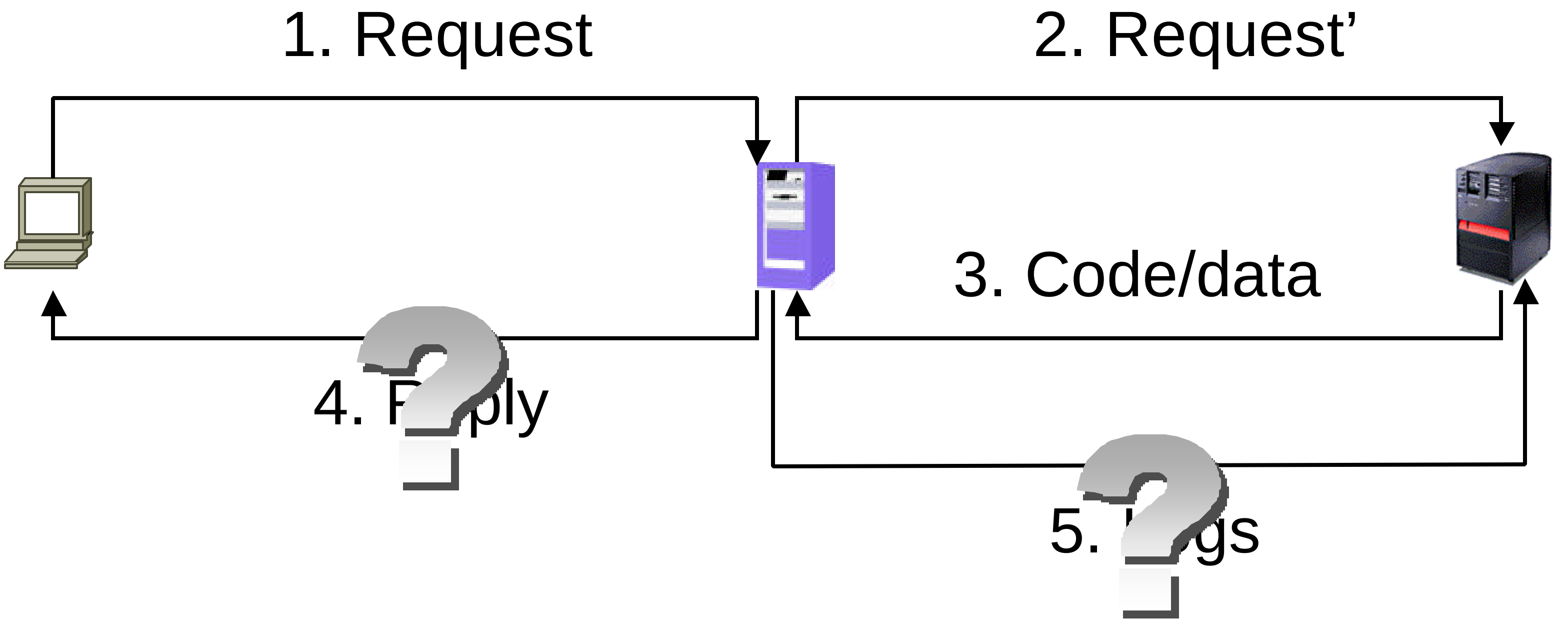
# Challenges

- Building an active cache
  - Addressed by previous work
- Incremental deployment
  - Some help from HTTP
- Security
  - Unaddressed until now

# Outline

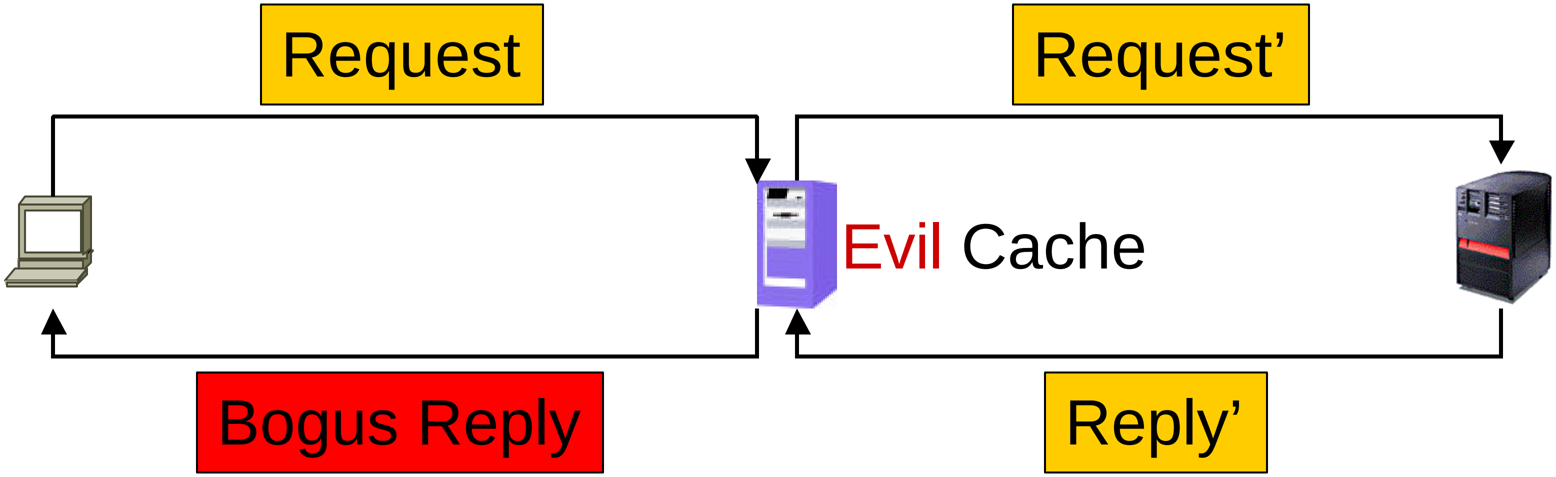
- The security problem
- Current solutions inadequate
- New approach to security
- Implementation
- Related work & conclusion

# New security problems



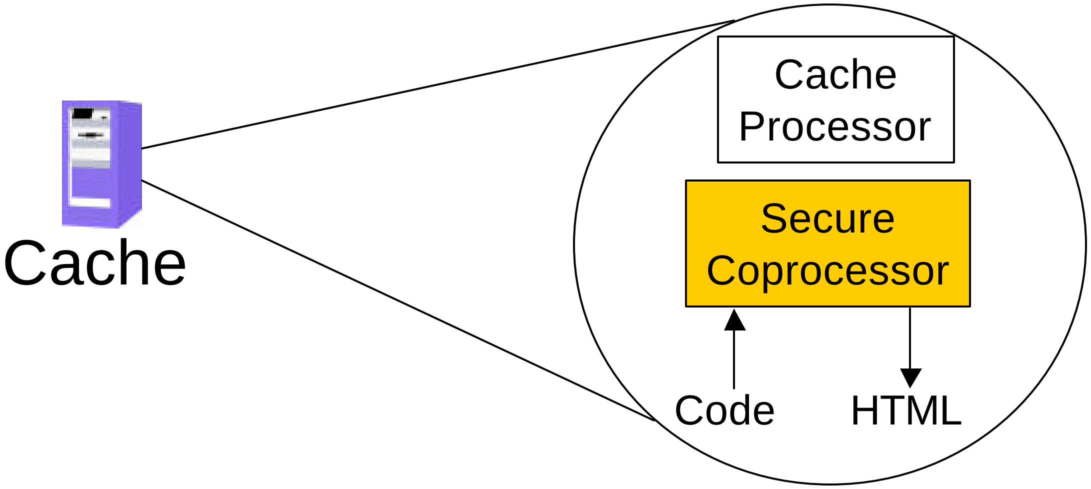
- Cache lies to client
- Cache lies to publisher
- (Malicious code sent to cache)

# Strawman: Public key signatures



- Cache supposed to alter content, so publisher signature meaningless to client
- Cache can still lie

# Strawman: Secure coprocessor



- Secure coprocessor is trusted by everyone
- Runs all publisher code
- Expensive and inflexible

# Outline

- The security problem
- Current solutions inadequate
- New approach to security
- Implementation
- Related work & conclusion

# Observations

- Securing individual request/reply pairs is expensive/difficult
- Publisher always knows what the right answer is
- Can we put publisher back into the loop?

# Solution architecture

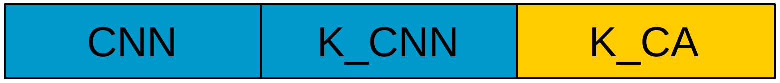
- Authorization
  - Publisher chooses caches to trust
- Authentication
  - Cache authenticates itself to client
  - Client can tell that a cache is authorized to serve a URL
  - Provides non-repudiation
- Verification
  - Client and publisher both verify that authorized caches are behaving

# Auth. basics

- Build on a Public Key Infrastructure (PKI)
- PKI provides a way to bind public keys to names
  - E.g. “CNN.com’s key is AD23428F989…”
  - Binding is in the form of a *certificate*
- We assume a Certificate Authority
  - Everyone trusts it
  - Everyone knows its public key, K_CA

# Meaning of a certificate

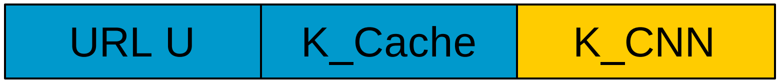
- Identity



| | | |
|---|---|---|
| CNN | K_CNN | K_CA |

  - E.g. CNN’s public key is K_CNN

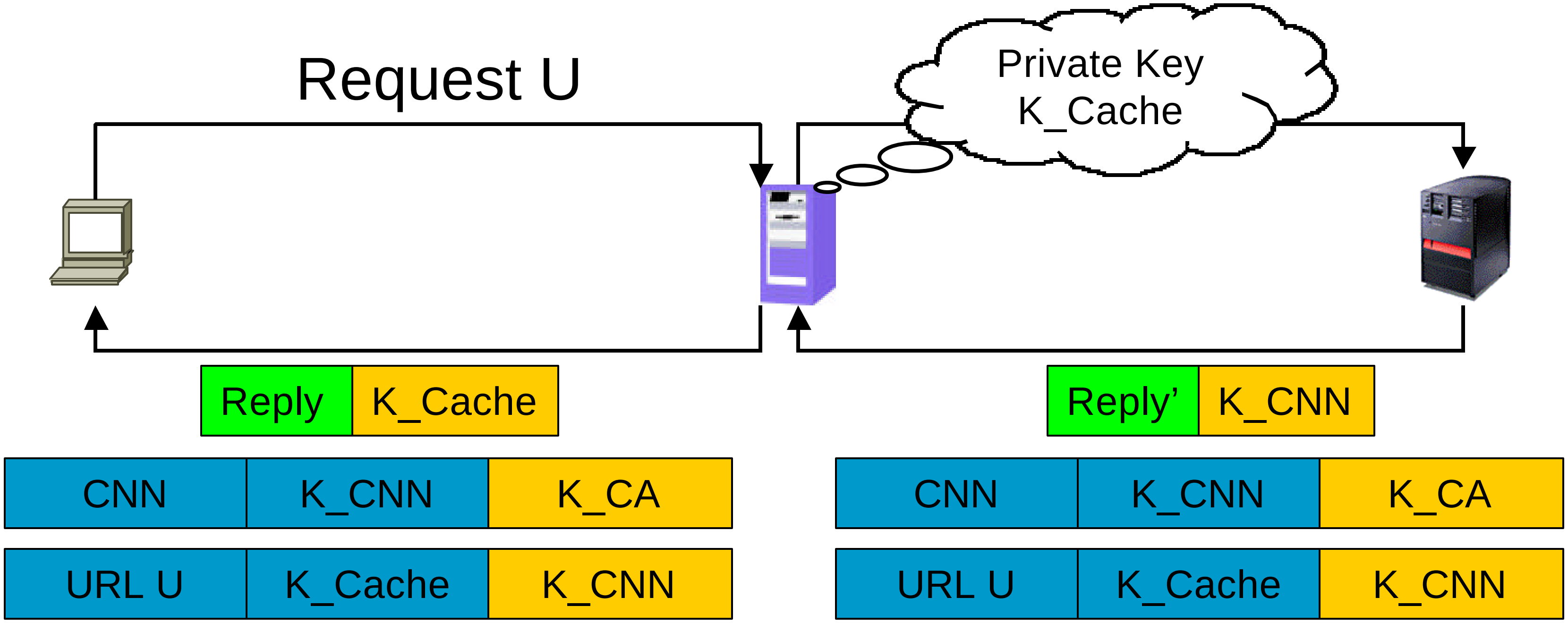
- Authorization



| | | |
|---|---|---|
| URL U | K_Cache | K_CNN |

  - E.g. CNN (the entity which knows K_CNN) authorizes the cache with key K_Cache to serve URL U

# Basic authorization



Request U

Private Key
K_Cache

| | |
|---|---|
| Reply | K_Cache |

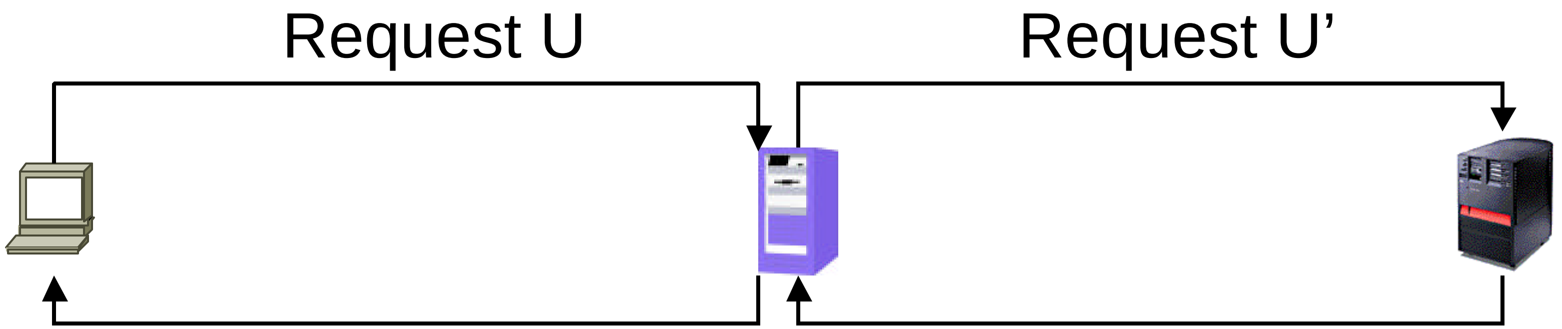
| | |
|---|---|
| Reply’ | K_CNN |

| | | |
|---|---|---|
| CNN | K_CNN | K_CA |
| URL U | K_Cache | K_CNN |

| | | |
|---|---|---|
| CNN | K_CNN | K_CA |
| URL U | K_Cache | K_CNN |

- CNN authorizes cache to serve U
- Cache signs its reply to client

# Authorization with delegation

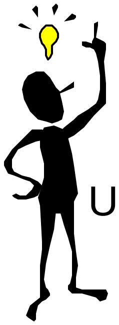


| | |
|---|---|
| Reply | K_Cache |

| | |
|---|---|
| Reply’ | K_CNN |

| | | |
|---|---|---|
| CNN | K_CNN | K_CA |
| URL U | K_UL | K_CNN |
| Honest | K_Cache | K_UL |

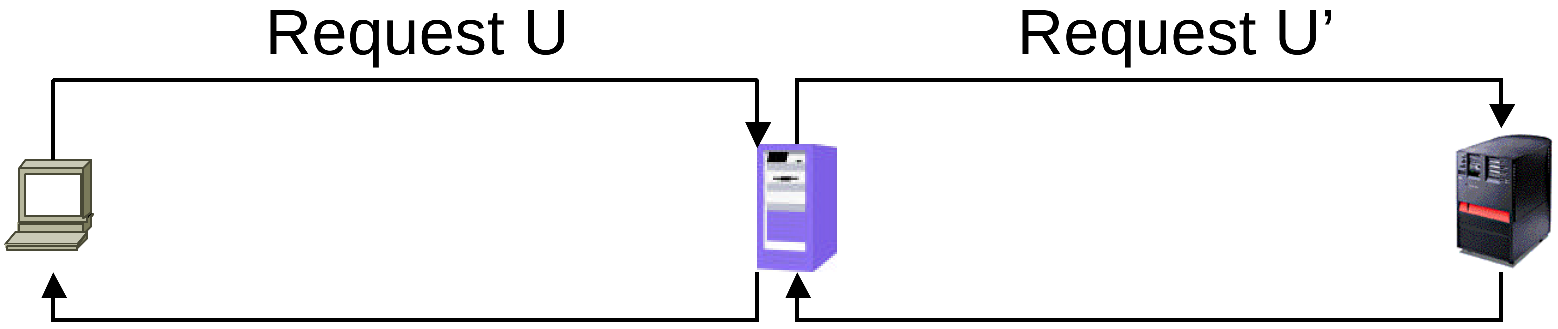
| | | |
|---|---|---|
| CNN | K_CNN | K_CA |
| URL U | K_UL | K_CNN |



| | | |
|---|---|---|
| Honest | K_Cache | K_UL |

Underwriters’ Laboratories

# Recursive delegation



| | |
|---|---|
| Reply | K_Cache |

| | |
|---|---|
| Reply’ | K_CNN |

| | | |
|---|---|---|
| CNN | K_CNN | K_CA |
| URL U | K_UL | K_CNN |
| Honest | K_AOL | K_UL |
| Cache | K_Cache | K_AOL |

| | | |
|---|---|---|
| CNN | K_CNN | K_CA |
| URL U | K_UL | K_CNN |

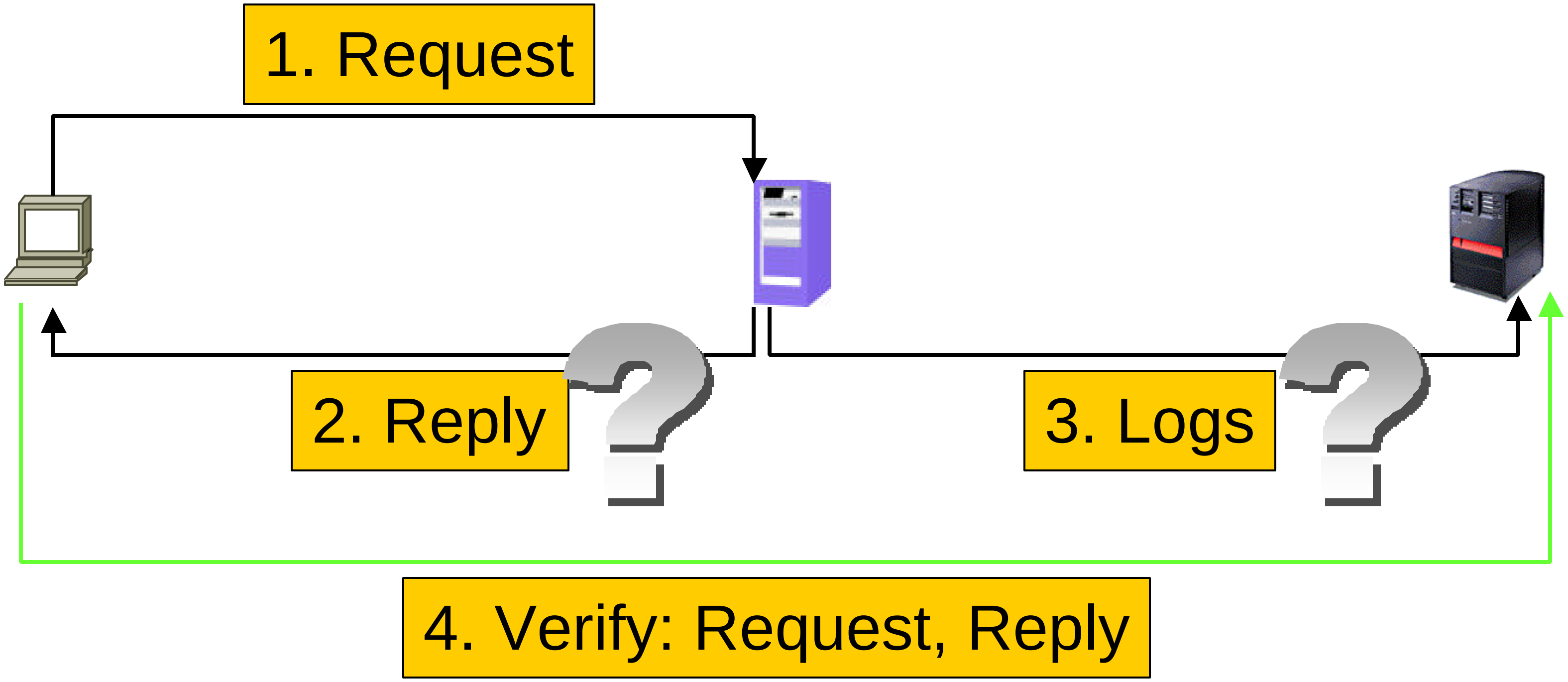
| | | |
|---|---|---|
| Honest | K_AOL | K_UL |

Underwriters’ Laboratories

# Verification

- Trusted cache can misbehave
  - Could be compromised
  - Administrator could be bribed
- Clients, publisher need to check cache’s output

# Verification design

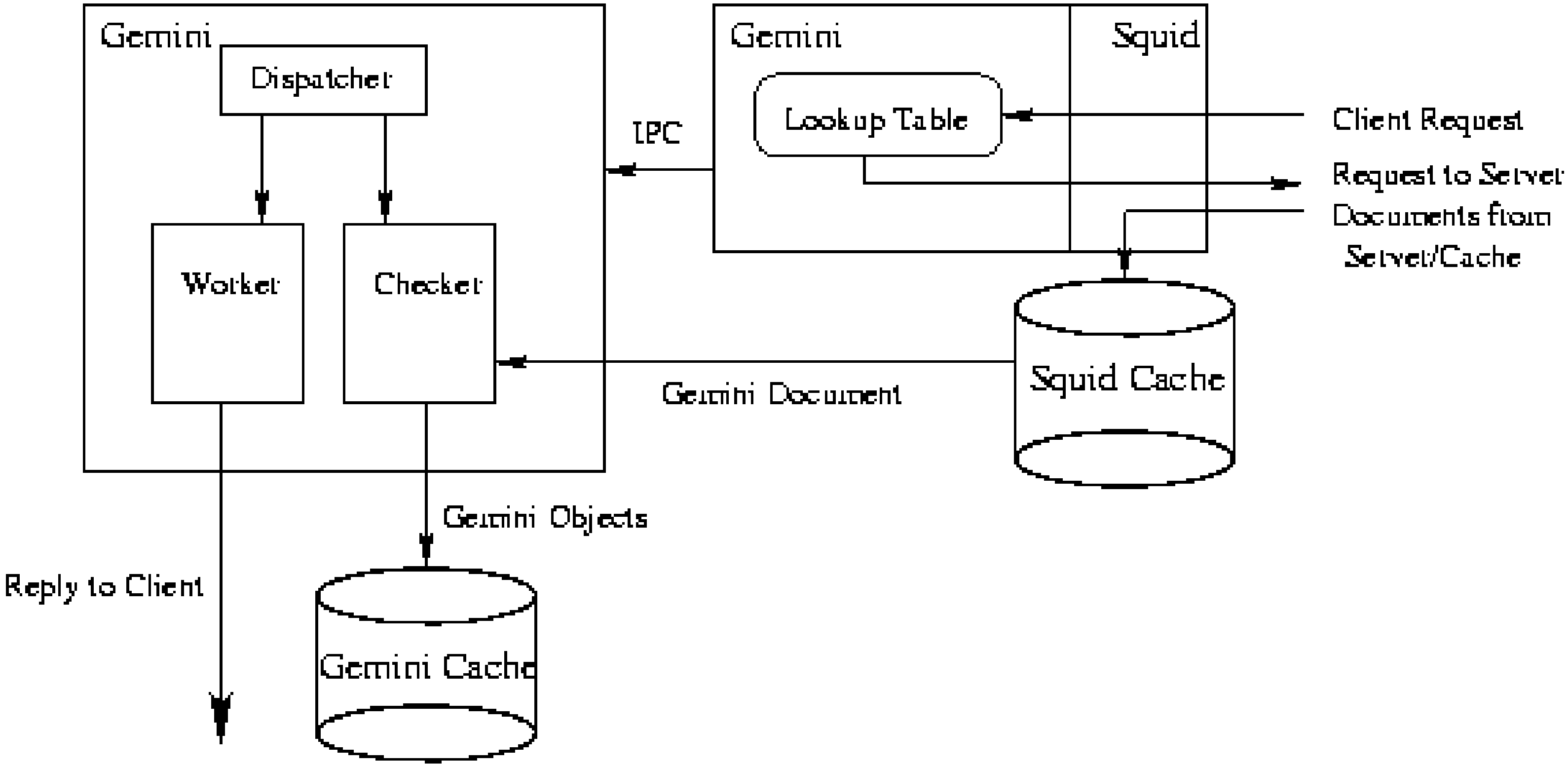


- Client sends verification request with some probability, *p*

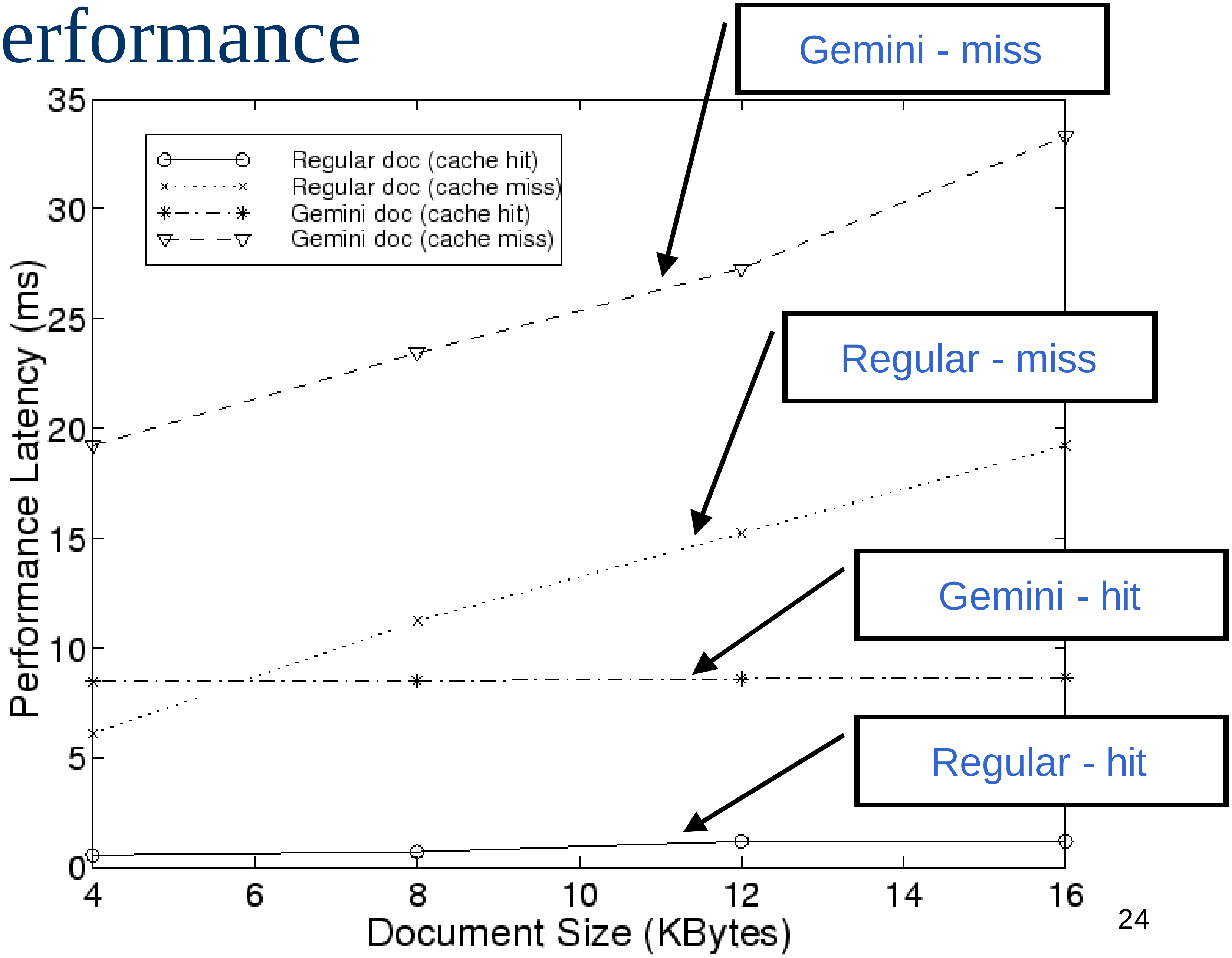
# Verification limitations

- Possible
  - Checking cache’s reply to client
  - Verifying that cache has not deleted logs
- Future work
  - Verifying that cache has not added bogus log entries

# System architecture



# Performance



Gemini - miss

Regular - miss

Gemini - hit

Regular - hit

# Related work

- Active proxies (Active Cache, HPP)
- WWW security (SSL, HTTPS, DSig, HTTP Digest Authentication)
- Mobile agents (e.g. Yee’s Sanctuary)
- Secure hardware (e.g. IBM’s coprocessor)

# Conclusion

- Caches need to become more publisher-centric
- We have addressed the security issues of publisher-centric caching
  - Authorization, Authentication, Verification
- We have implemented our ideas by adding a Java VM to Squid
  - Performance enhancement is future work