CDN Architecture Pain Points and ICN Cures?

Christopher A. Wood
UCI and PARC

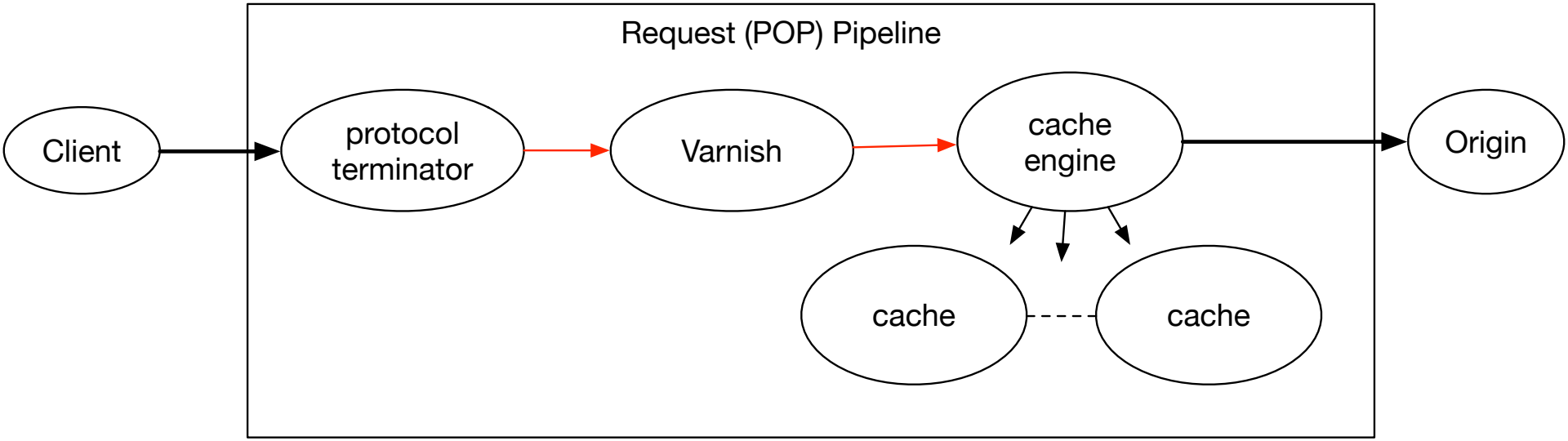
ICNRG Interim Meeting – IETF 96 – Berlin

July 17, 2016

Agenda

- CDN architectures and patterns
  - Fastly and CloudFlare
- TLS deployment concerns
- Major pain points

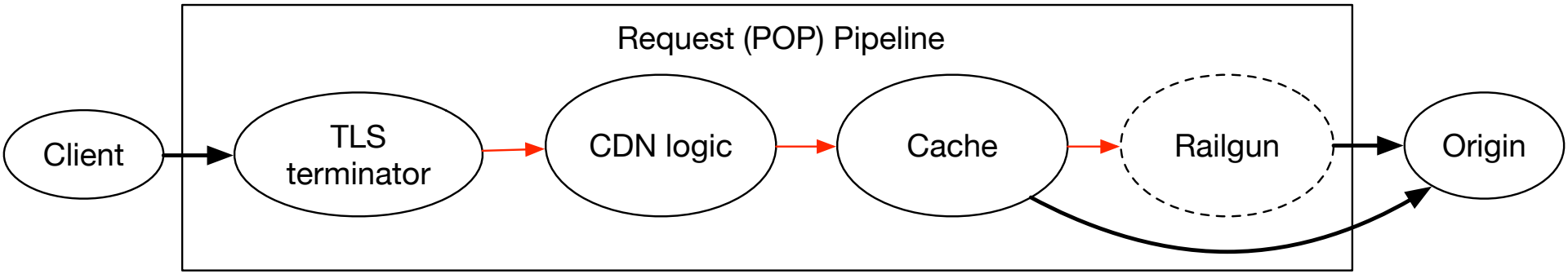
Fastly


Request (POP) Pipeline
Client
protocol terminator
Varnish
cache engine
Origin
cache
cache

Varnish Configuration Language

- Authentication
- Some rate limiting
- Personalized content as well as pretty sophisticated load-balancing
- Routing
- Failover

CloudFlare


Request (POP) Pipeline
Client
TLS terminator
CDN logic
Cache
Railgun
Origin

Architecture Patterns

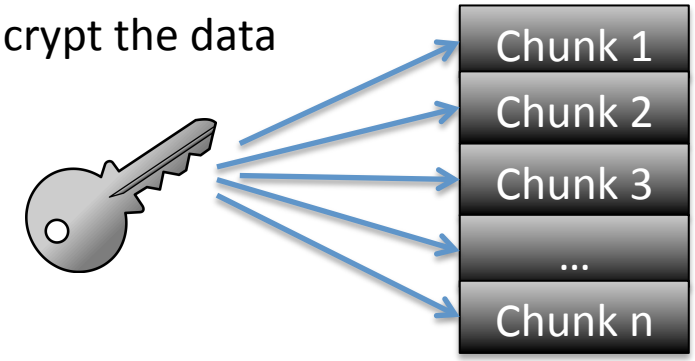
- DNS Anycast for routing-based load balancing
- Edge TLS termination, cleartext internal traffic
  - Keep an eye on LURK solutions to deal with private key relinquishment problem
- State synchronization or message passing within POPs
- Pushing application logic to the edge
  - Treating the origin as a data store or coordinator

TLS Deployment

- HTTPS everywhere: **best practice**
  - … but not needed everywhere?
- Is use context-sensitive?
  - EFF: HTTPS everywhere (obviously)
  - Netflix: HTTPS for PlayReady manifests and **HTTP(S?) for data**
  - Banks, e-commerce, etc.: HTTPS everywhere

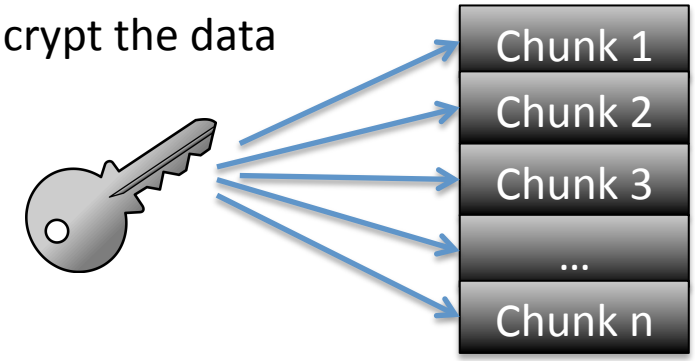
The Netflix Case

1) Encrypt the data

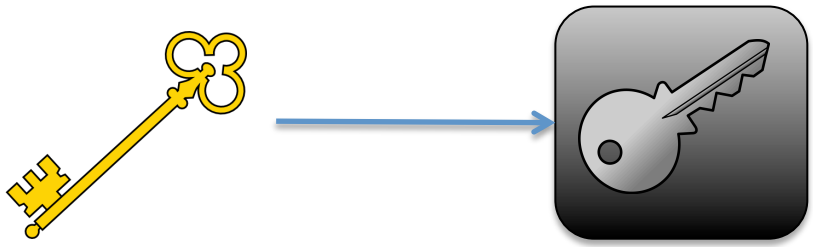


The Netflix Case

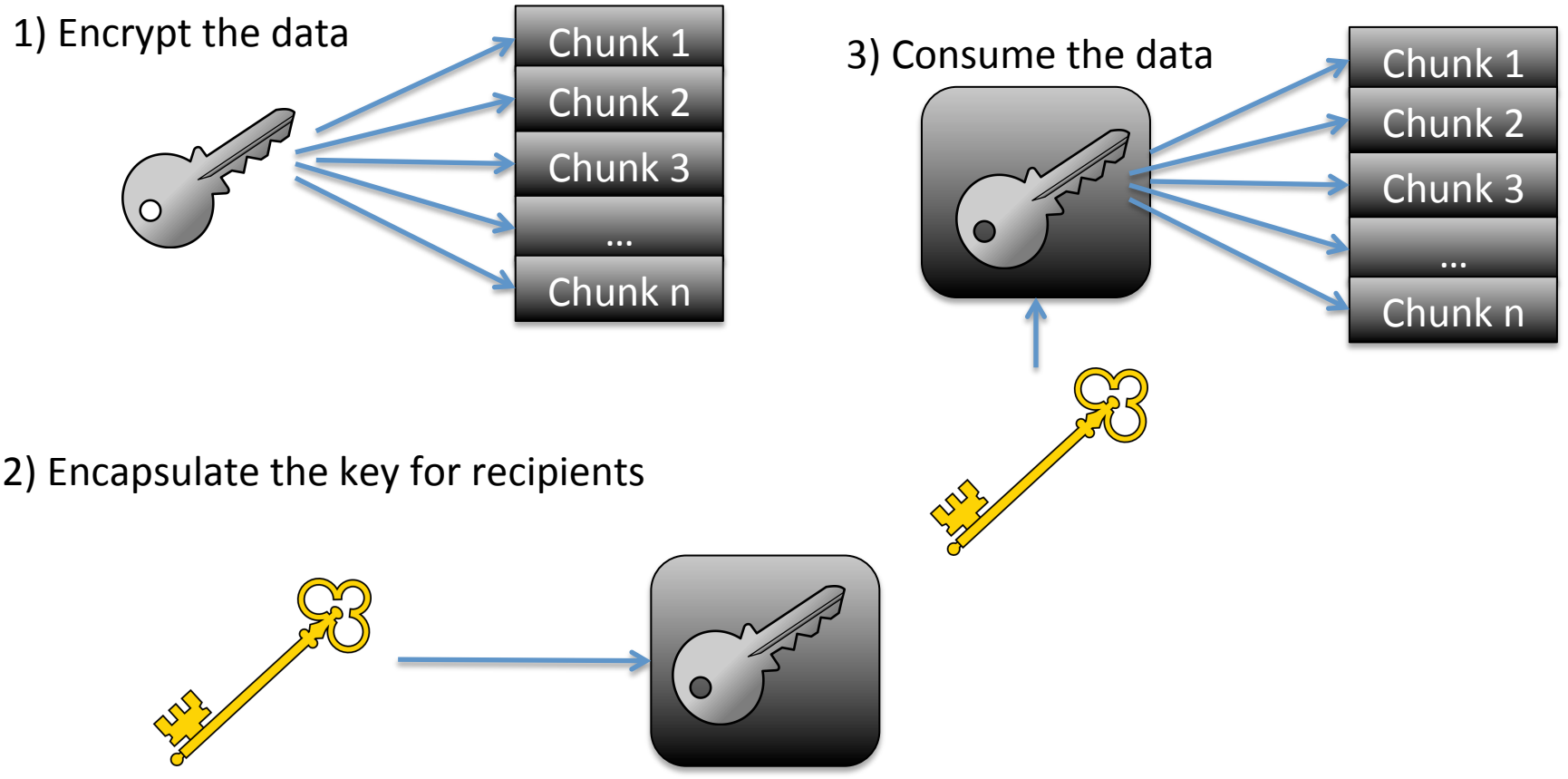
1) Encrypt the data



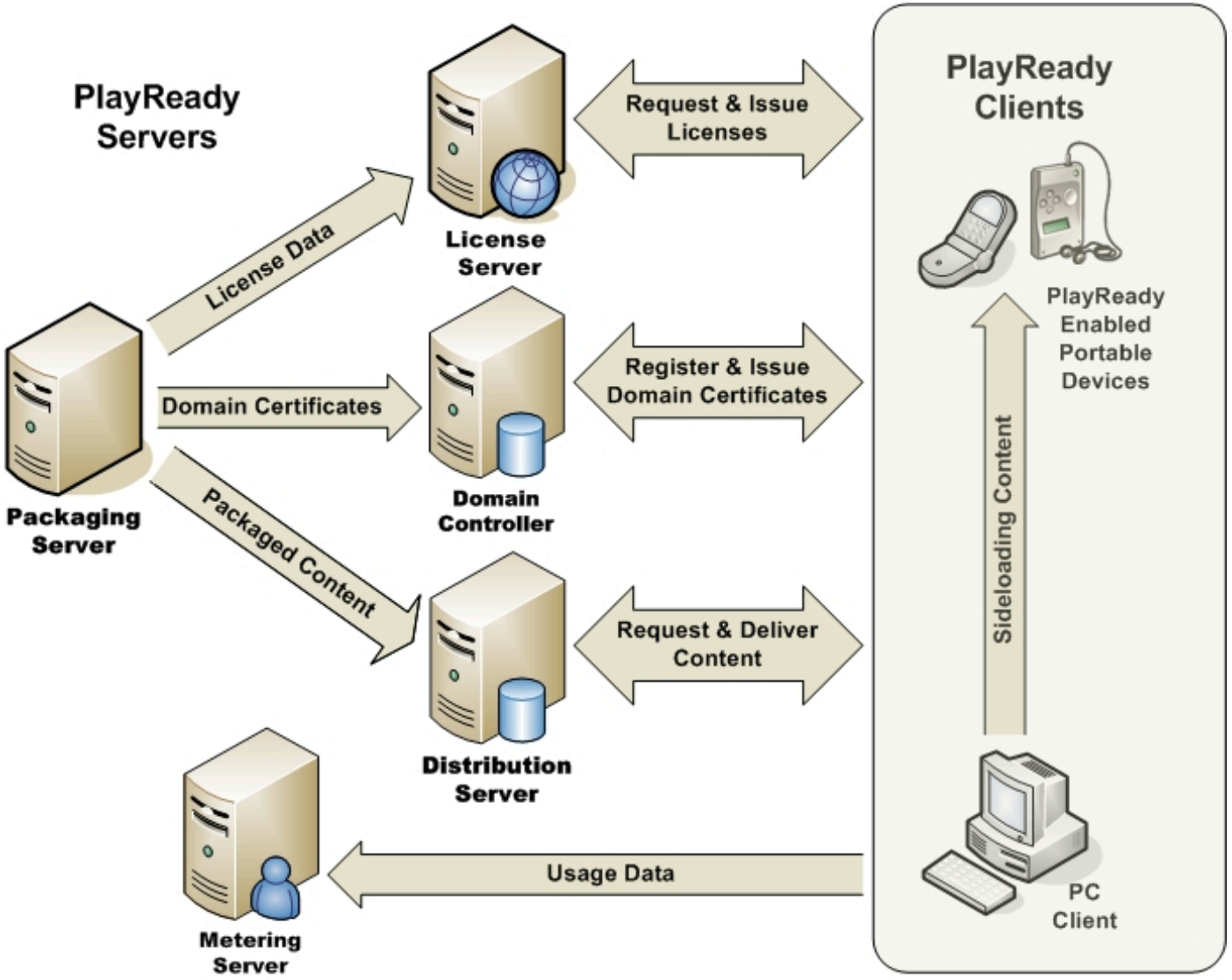
2) Encapsulate the key for recipients



The Netflix Case



Netflix: PlayReady



Protection Mechanisms

- Use AES-CBC to encrypt data chunks
- Only authorized consumers can decrypt the PlayReady manifest (license) and obtain the symmetric key
- Rationale?
  - AES-CBC Allows for random access and is not supported by TLS cipher suite
  - Exposure protected by client-specific license key encapsulation

Pain Point #1: DNS Anycast

- Problem: poor deployment or non-local resolver can result in suboptimal POP node.

Pain Point #2: Tracking State Changes

- Problem: What resources need to be changed when an object is modified?

Pain Point #3: Caching API Requests

- Problem: API requests may be dynamic and the responses typically contain “structured” JSON data

Pain Point #4: Mixed Content

- Problem: some applications serve HTTP content over HTTPS, or the other way around

Pain Point #5: Event-Driven Content

- Problem: how can we handle “event-driven” content?

Pain Point #6: Distributed Applications

- Problem: many applications, frameworks, etc. are not engineering with caching in mind

Pain Point #7: TLS Termination

- Problem: how do CDNs and origin servers coordinate to share private keys without causing long-term problems?