Generic Overlay Networks

(How to fetch your content)

The great thing about DNS

- It exists and is widely deployed
- It works to resolve names
  - mostly hosts/domains -> IPs.
  - but can resolve anything
- Has a secure option

The problems with DNS

- Name resolution requires DNS reachability
  ... and an additional round-trip in many cases for conn setup
- DNS TTL is commonly ignored
- There is no correct DNS TTL [Goldilock's problem]
- There is no correct number of IPs to return [Goldilocks]
- Interacts with same-origin in fragile ways
- Practical for hosts/domains, not resources.
  ... large blobs make having every resource everywhere impractical
- Replay attacks work, even with DNSSEC

The problems with DNS

Solved/Solvable with DoH

- Name resolution requires DNS reachability
  ... and an additional round-trip in many cases for conn setup
- DNS TTL is commonly ignored
- There is no correct DNS TTL [Goldilock's problem]
- There is no correct number of IPs to return [Goldilocks]
- Interacts with same-origin in fragile ways
- Practical for hosts/domains, not resources.
  ... large blobs make having every resource everywhere impractical
- Replay attacks work, even with DNSSEC

The problems with DNS

Unsolved with DoH

- Name resolution requires ~~DNS~~ **network** reachability
  ... and an additional round-trip in many cases for conn setup
- DNS TTL is commonly ignored
- There is no correct DNS TTL [Goldilock's problem]
- There is no correct number of IPs to return [Goldilocks]
- Interacts with same-origin in fragile ways
- Practical for hosts/domains, not resources.
  ... large blobs make having every resource everywhere impractical
- Replay attacks work, even with DNSSEC

What about this same-origin thing?

- Users trust your name, which effectively means they trust your cert (and the cert chain providers, yadda yadda)...

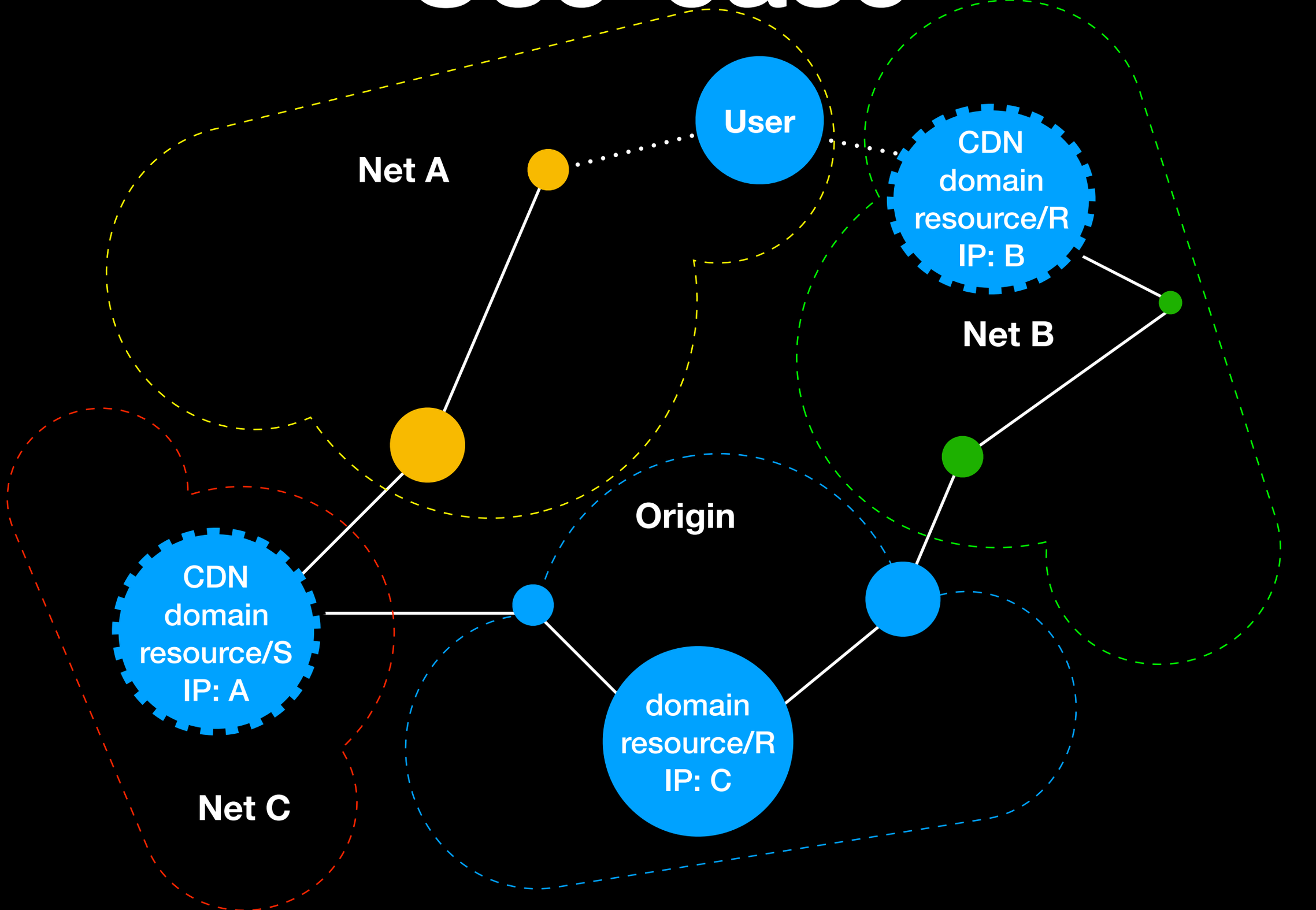
- So, how do you safely use a 3rd party cache (a.k.a. CDN?)

- I know, lets give some 3rd party our cert! That will be *Great* for security!

Use-case 1

- User wants resource http://domain/resource/R
- User also wants resource http://domain/resource/S
- Site wishes to use both CDN A **and** CDN B.

Use-case


User
Net A
CDN
domain
resource/R
IP: B
Net B
Origin
CDN
domain
resource/S
IP: A
domain
resource/R
IP: C
Net C

Policy

So.. which IP is the correct IP for this resource?

CDN A's nearest?
CDN B's nearest?
Origin's IP?
3rd party's IP?

Policy

There is no correct answer to that question.

The answer depends on:

- where the content lives
- where the user is
- cache sizes
- CDN node load
- CDN node reachability
- path load in the network
- regulatory issues
- time
- privacy/security issues
- etc.

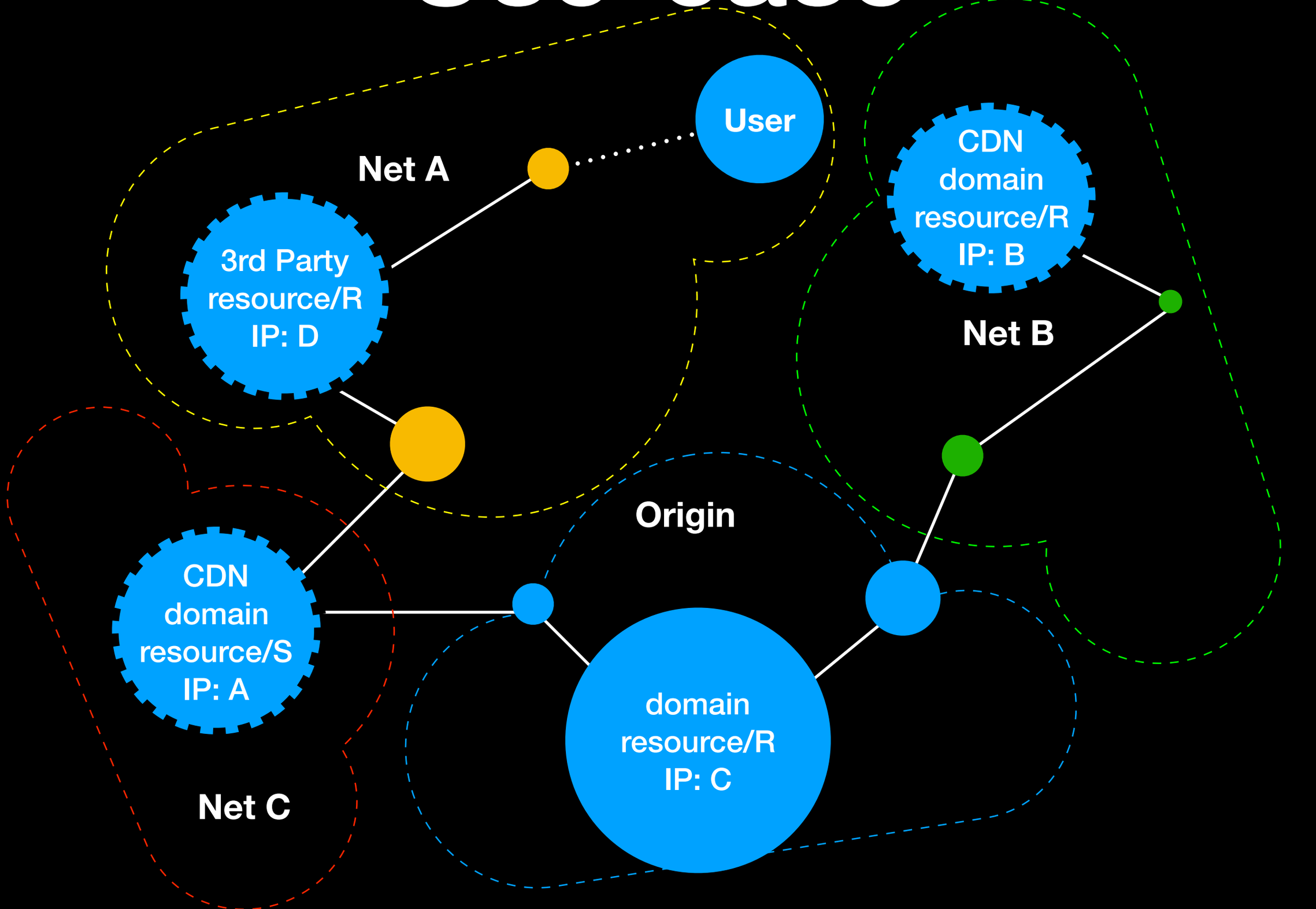
Policy

.. and yet our current resolution system effectively requires there to be a single answer.

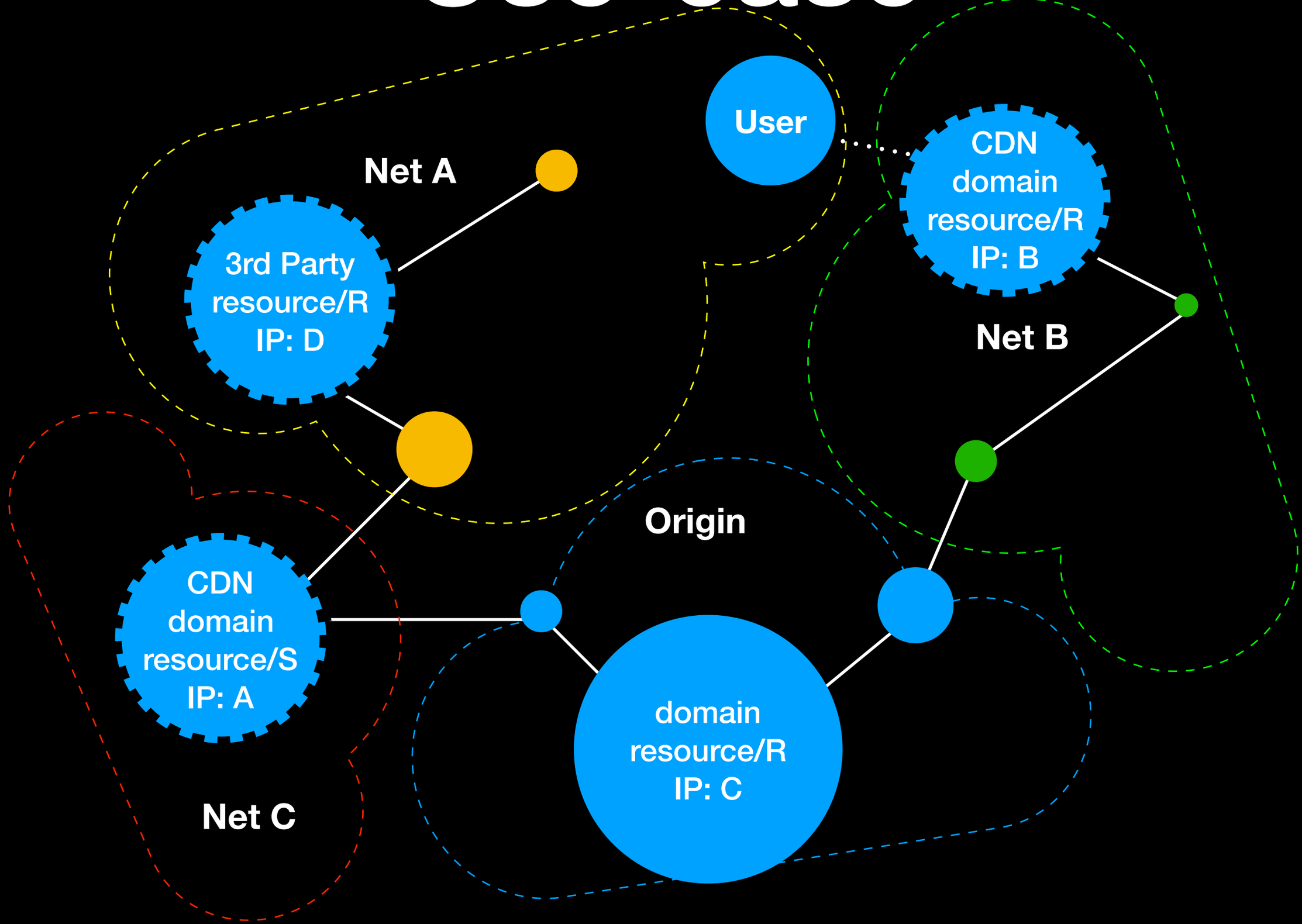
Use-case 2

- User wants resource http://domain/resource/R
- User regularly moves from network A to network B

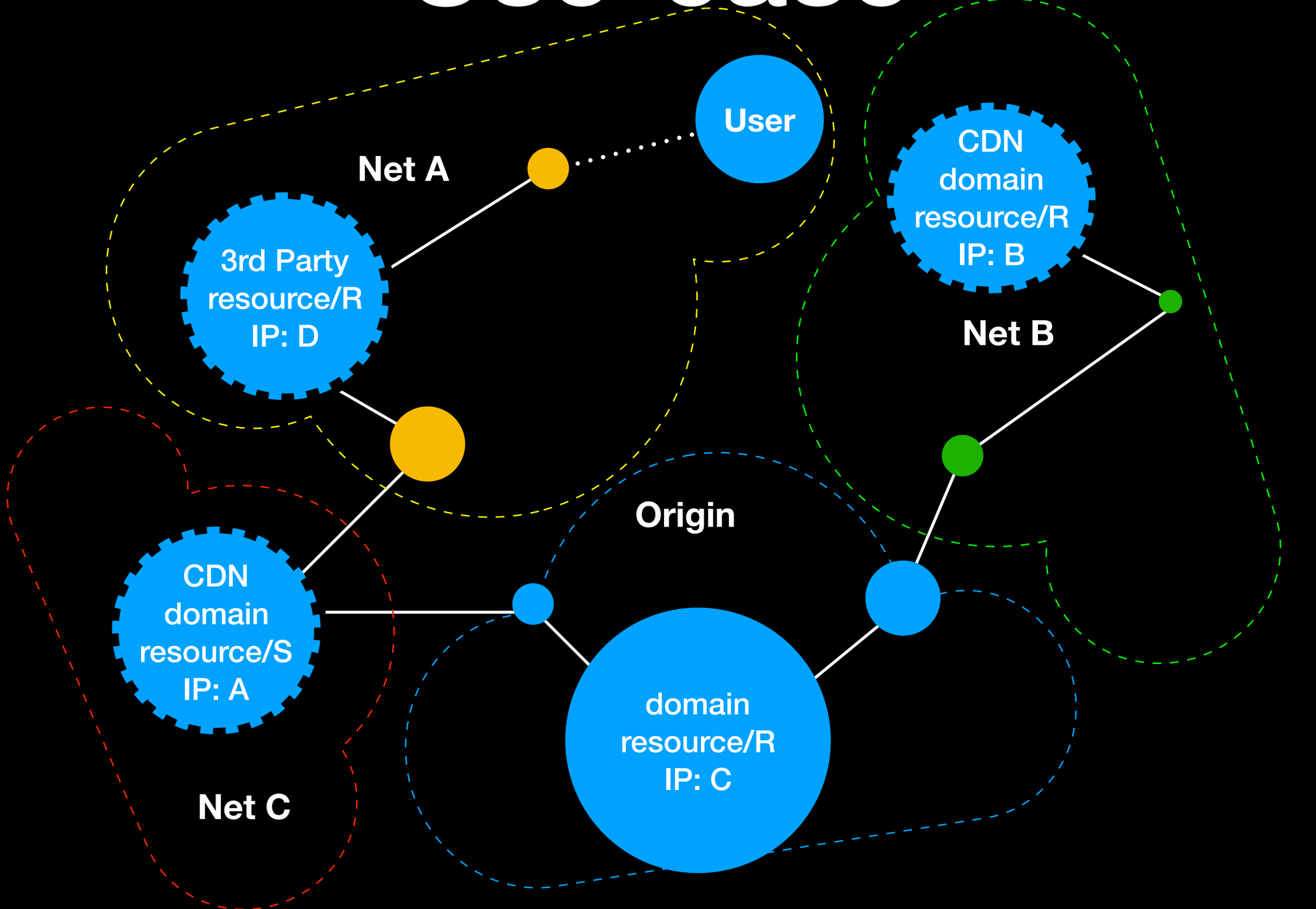
Use-case


User
Net A
3rd Party
resource/R
IP: D
CDN
domain
resource/R
IP: B
Net B
Origin
CDN
domain
resource/S
IP: A
Net C
domain
resource/R
IP: C

Use-case


User
Net A
3rd Party
resource/R
IP: D
CDN
domain
resource/R
IP: B
Net B
Origin
CDN
domain
resource/S
IP: A
Net C
domain
resource/R
IP: C

Use-case


User
Net A
3rd Party
resource/R
IP: D
CDN
domain
resource/R
IP: B
Net B
Origin
CDN
domain
resource/S
IP: A
Net C
domain
resource/R
IP: C

Policy

So.. which IP is the correct IP for this resource?

CDN A's nearest?
CDN B's nearest?
Origin's IP?
3rd party's IP?

Policy

There is no correct answer to that question.

The answer depends on:

- where the content lives
- where the user is
- cache sizes
- CDN node load
- CDN node reachability
- path load in the network
- regulatory issues
- time
- privacy/security issues
- etc.

Policy

There is no correct answer to that question.

The answer depends on:

A bunch of stuff.

Sensing a pattern here?

Sensing a pattern here?

Proposal:

Proposal

Provide for a means of safely delegating who can serve resource, when, and where.

Proposal

Provide for a means of safely delegating who can serve resource, when, and where.

... without being easily abused.

What about signed-exchange?

It's almost there-- but it is missing some policy bits that it should have.

Proposal

Define a policy that:

- is signed by the origin
- declares to whom it applies
- declares over what it applies
- provides for a 1:many mapping of resources:servers
- prevents bad actors from providing bad experiences

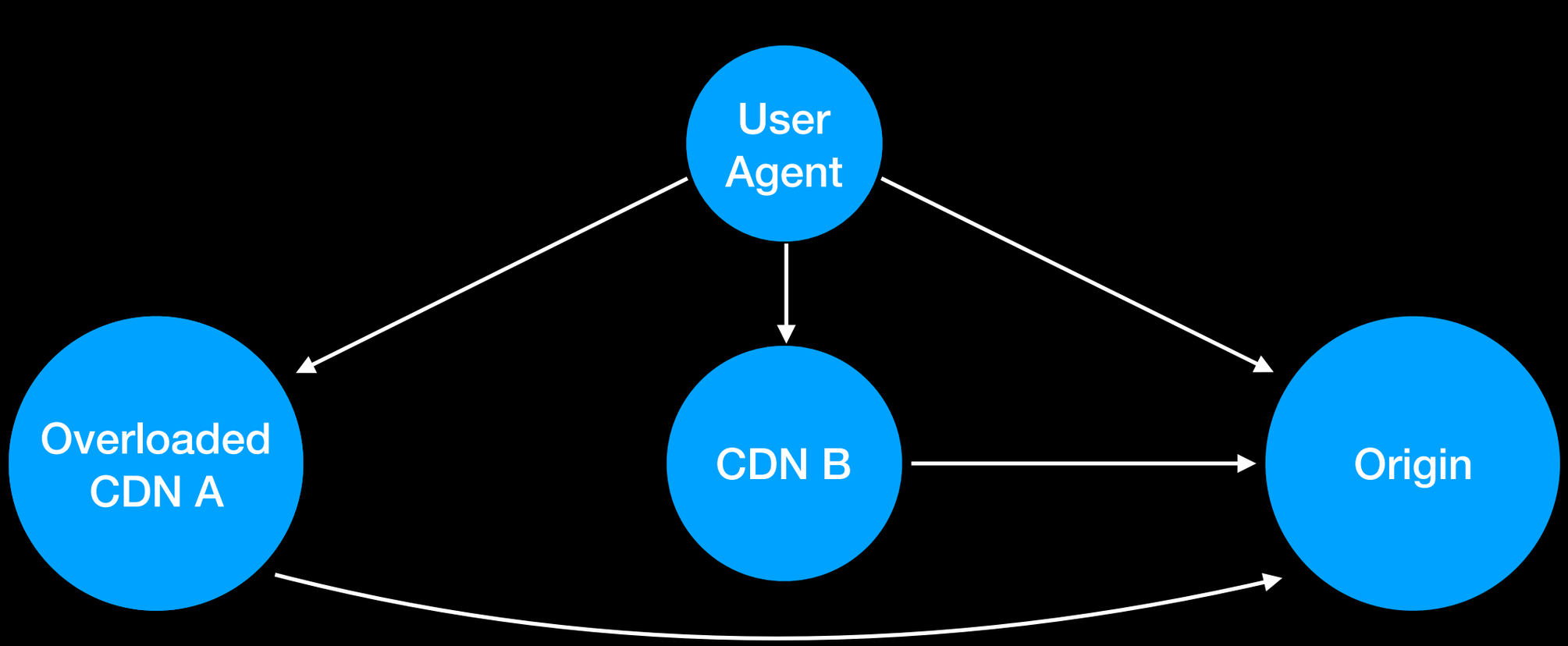
Proposal

Preventing bad actors from providing bad experiences

Primarily involves parallelism-- to prevent bad actors, at least one path must be controlled by origin.

Proposal

Primarily involves parallelism-- to prevent bad actors, at least one path must be controlled by origin.



Proposal

Preventing bad actors from providing bad experiences

An example (simple) policy language:

(race maxConcurrent=N ...)
(delay seconds ...)
(timeout seconds ...)
(https hostname ...)
(https ip:port ...)

Proposal

A policy to prevent bad actors:

```
(race maxConcurrent=3
      (timeout 0.5 (delay 0.00 (https CDN_A (https 0))))
      (timeout 0.5 (delay 0.25 (https CDN_B (https 0))))
      (delay 0.50  (https 1.2.3.4))
  )
```

Proposal

A policy to prevent bad actors:

```
(race maxConcurrent=3
      (timeout 0.5 (delay 0.00 (https CDN_A (https O))))
      (timeout 0.5 (delay 0.25 (https CDN_B (https O))))
      (delay 0.50  (https 1.2.3.4))
)
```

This tells the user-agent it can make up-to three concurrent connections.

The 1st happens immediately, and goes to CDN_A.
The 2nd happens after .25 seconds, and goes to CDN_B.
The 3rd happens after 0.5 seconds, and goes directly to the specified IP.

Both CDN_A and CDN_B are instructed to go to DNS O if they don't already have the resource.

Proposal

A policy to prevent bad actors:

```
(race maxConcurrent=3
      (timeout 0.5 (delay 0.00 (https CDN_A (https 0))))
      (timeout 0.5 (delay 0.25 (https CDN_B (https 0))))
      (delay 0.50  (https 1.2.3.4))
)
```

CDN_A's overload can only impact the user for 0.25 seconds.. because a second connection and request to CDN_B will happen at that time.

When both CDN_A and CDN_B are bad, this can only impact the user for 0.5 seconds, as then another connection and request will go to the appropriate origin IP.

Proposal

That isn't the whole policy, though:

(resourcePath /prefix/foo/*)
(domain com.foo.static.bar)
(origin com.foo)
(clientTarget ip/mask)
(refetchAfter secsFromEpoch)
(validUntil secsFromRefetchAfter)
(policy (race maxConcurrent=3
      (timeout 0.5 (delay 0.00 (https CDN_A (https 0))))
      (timeout 0.5 (delay 0.25 (https CDN_B (https 0))))
      (delay 0.50 (https 1.2.3.4))
))
<all above signed by com.foo>

Proposal

To compress things, we could also include names

(thisPolicyName name X)
(includePolicy otherName X-1)
<things below are now optional and override things imported from otherName>
(refetchAfter secsFromEpoch)
(validUntil secsFromRefetchAfter)
<all above signed by com.foo>

Proposal

How would we implement happy-eyeballs?

(thisPolicyName happyEyeballs)
(resourcePath /prefix/*)
(domain com.foo.static.bar)
(origin com.foo)
(clientTarget ip/mask)
(refetchAfter secsFromEpoch)
(validUntil secsFromRefetchAfter)

```
(policy (race maxConcurrent=2
        (timeout 0.5 (delay rand()*X (https CDN_A (https 0))))
        (timeout 0.5 (delay rand()*(X-1) (quic CDN_B (https 0))))
))
<all above signed by com.foo>
```

This is not complete

The idea is that we should start thinking about being explicit about policy.

Policy is a space where we need innovation.