The Case for Public Work

Wu-chang Feng, Ed Kaiser

Portland State UNIVERSITY

Supported by:

intel NSF

Motivation

- Unwanted traffic is uncontrollable
  - Spam
  - Viruses
  - Worms
  - Port scans
  - Denial of service
  - Phishing

Approaches

- Indirection
  - Hide or dynamically relocate to prevent indefinite access
- Filtering
  - Drop unwanted traffic upstream to save network resources
- Capabilities
  - Provide fine-grained control over who is given service
- Proof-of-Work
  - Make adversaries commit as many resources as they request

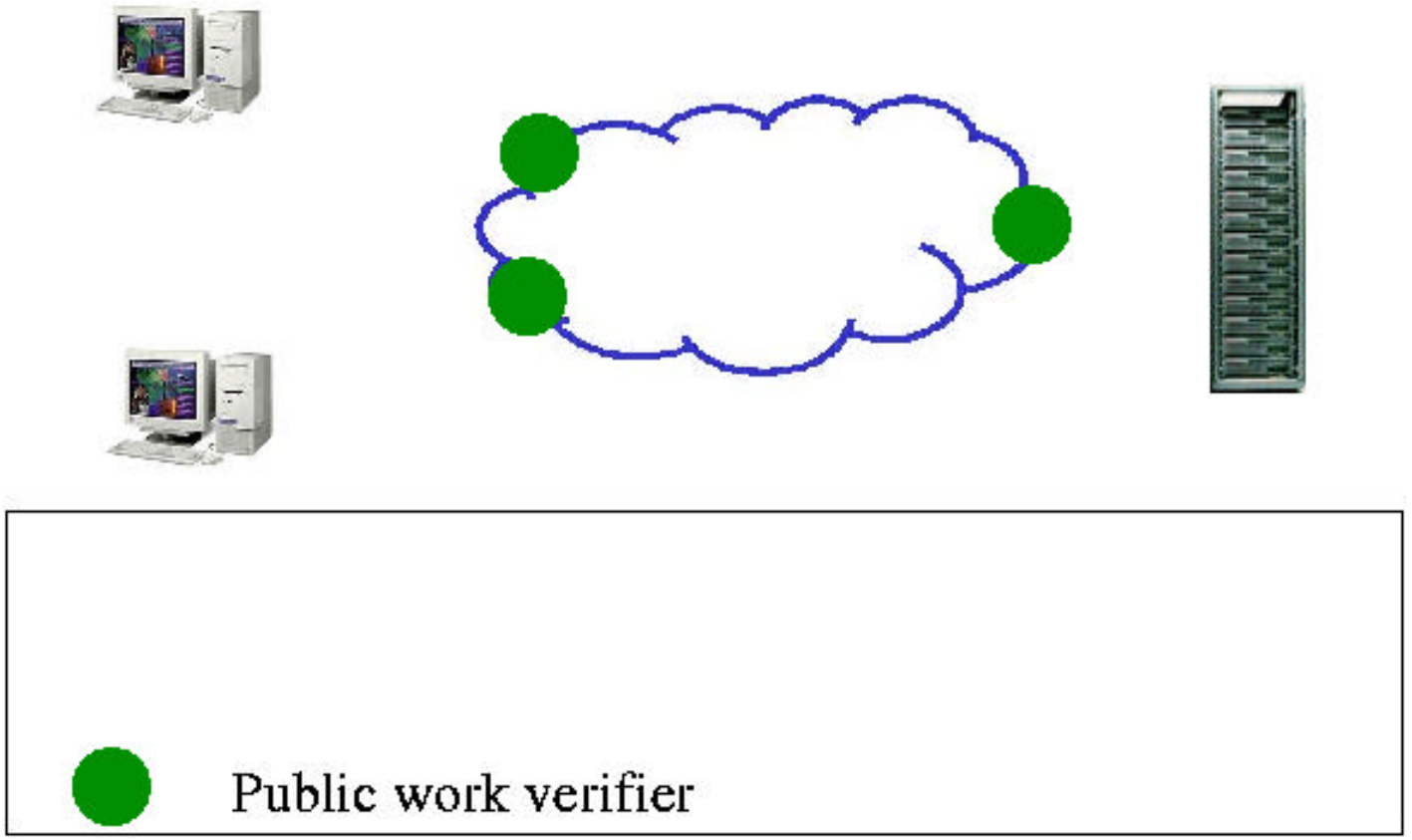
Public work approach

- Combine salient features of each approach into single mechanism based on “public work functions”
- What is a public work function?
  - A cryptographic puzzle issued by service whose answer can be verified by anyone in the network
  - Specifically, anyone can verify
    - Correctness
    - Freshness (work performed recently)
    - Amount of work (difficulty)

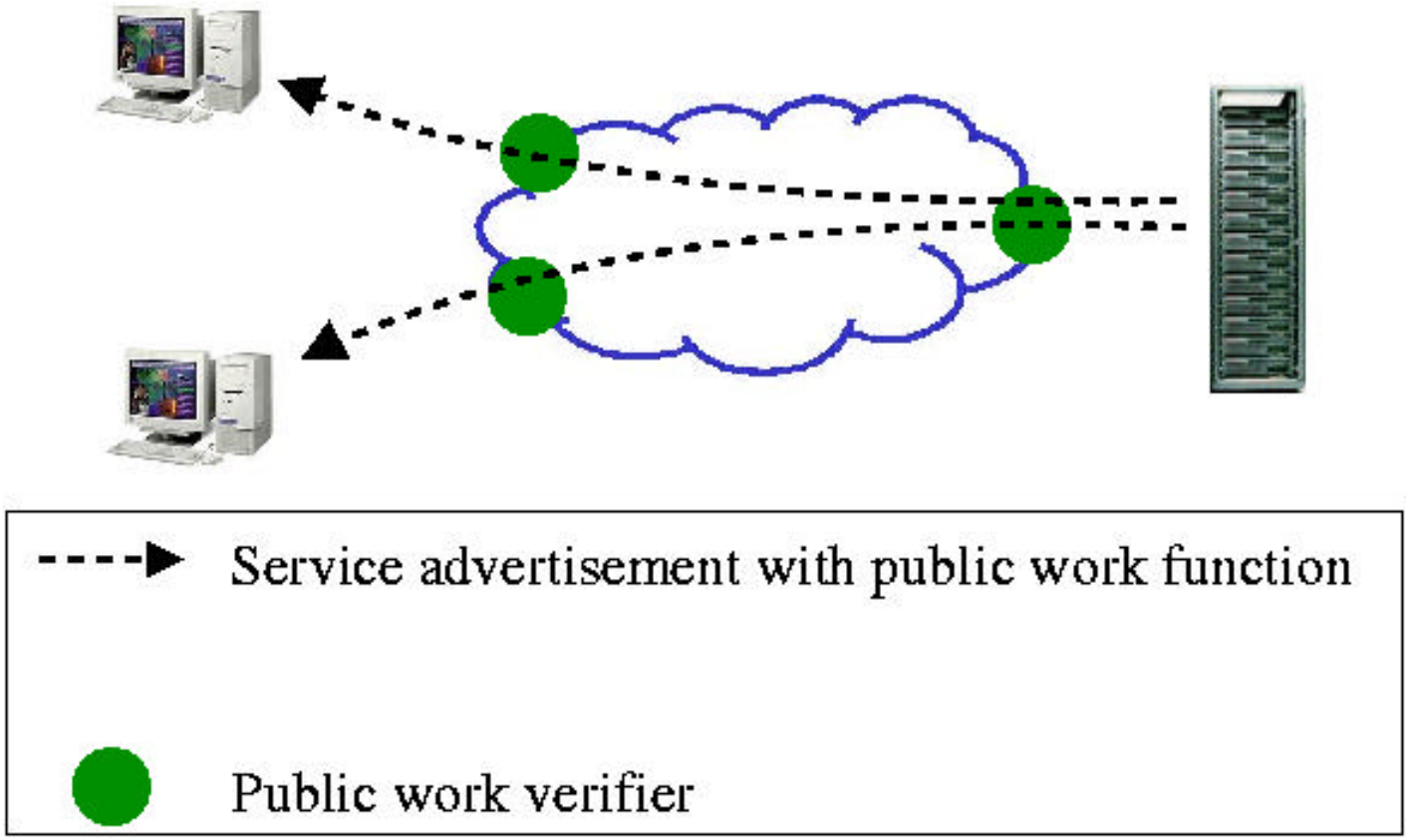
Basic operation

- Service advertises public work function with its location or resource
- Clients solve function and attach a valid answer on subsequent service requests
- Verifiers within network check for a valid answer before forwarding request

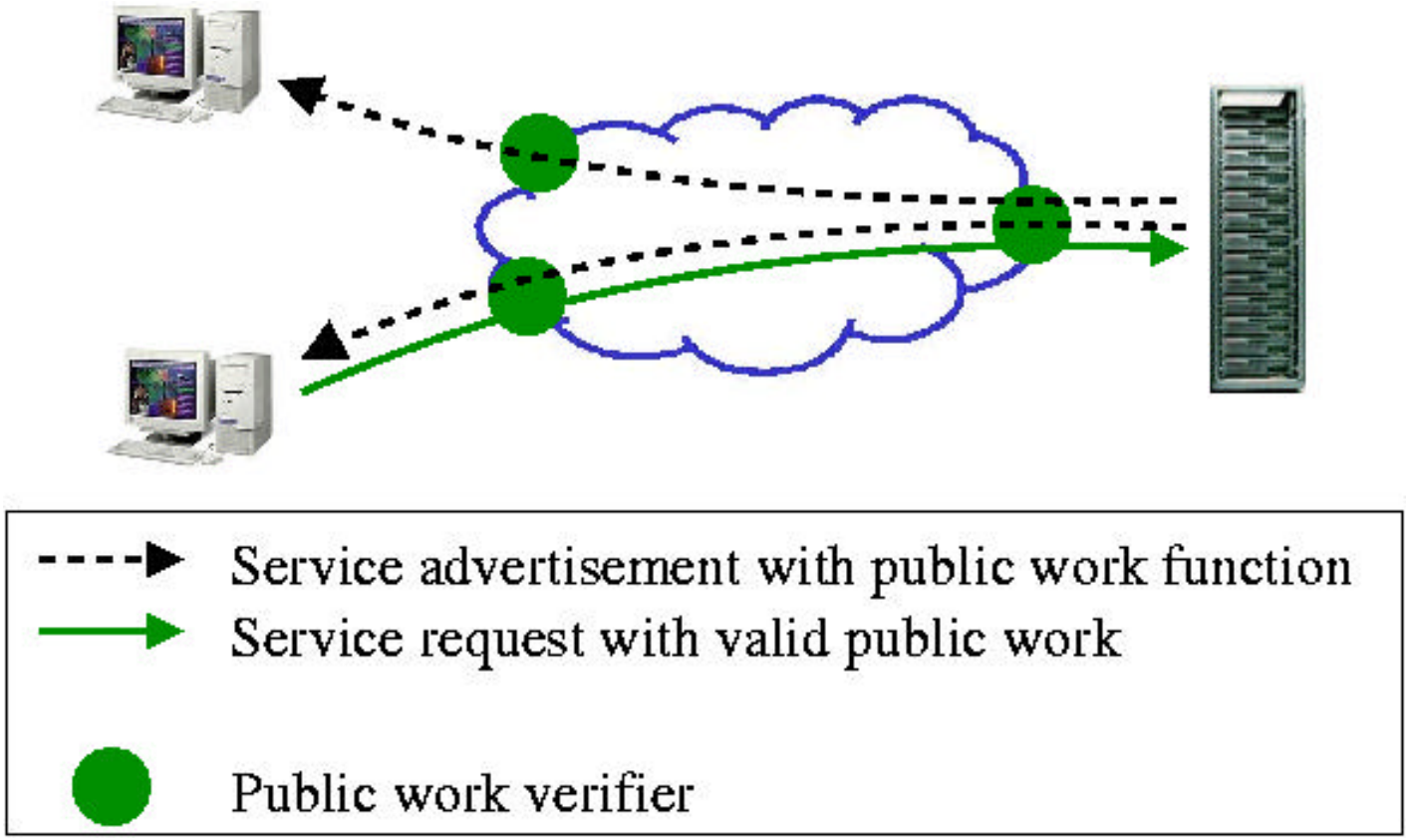
Basic operation



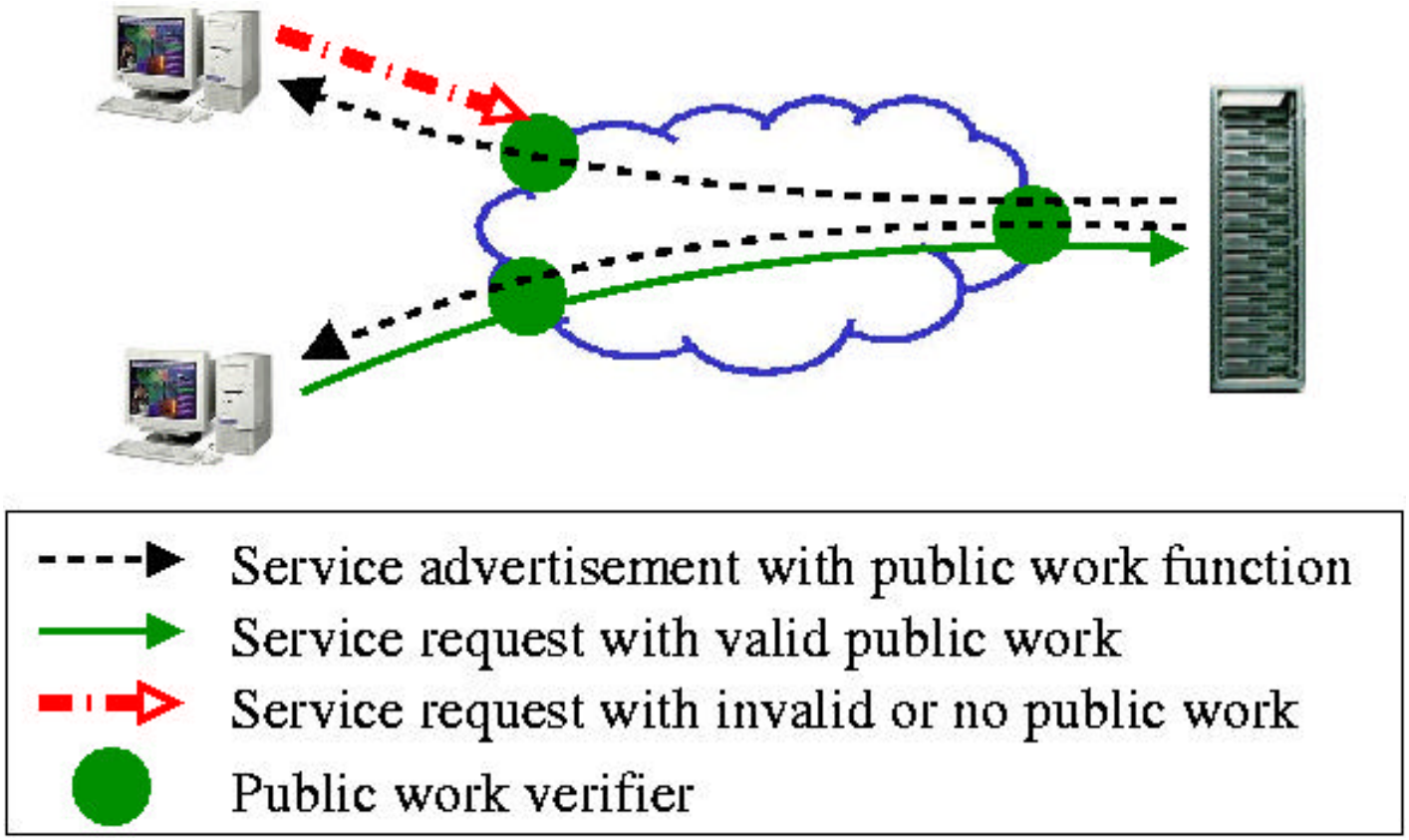
Basic operation



Basic operation



Basic operation



Revisiting approaches

- Indirection
  - Dynamically change reachable locations by changing public work function
- Filtering
  - Destination-controlled filtering at the client's network edge based on function difficulty
- Capabilities
  - Fine-grained control over access using source-specific public work function
- Proof-of-work
  - Public work function is a puzzle of a given difficulty

Public work functions

- Goals
  - Fast issuing
  - Fast verification
  - Flexible binding
  - Limited pre-computation and replay

A novel public work function

Targeted Hash Reversal

$$SHA1(A,F,N_c) \circ 0 \bmod D_c$$

- Server advertises nonce and difficulty N_c, D_c
- Client must find A for flow F that satisfies above equation
  - Desired output must land in ‘Bucket 0’

  - Relies on pre-image resistance of SHA1
  - Assumes SHA1 has uniformly distributed output

Public work functions

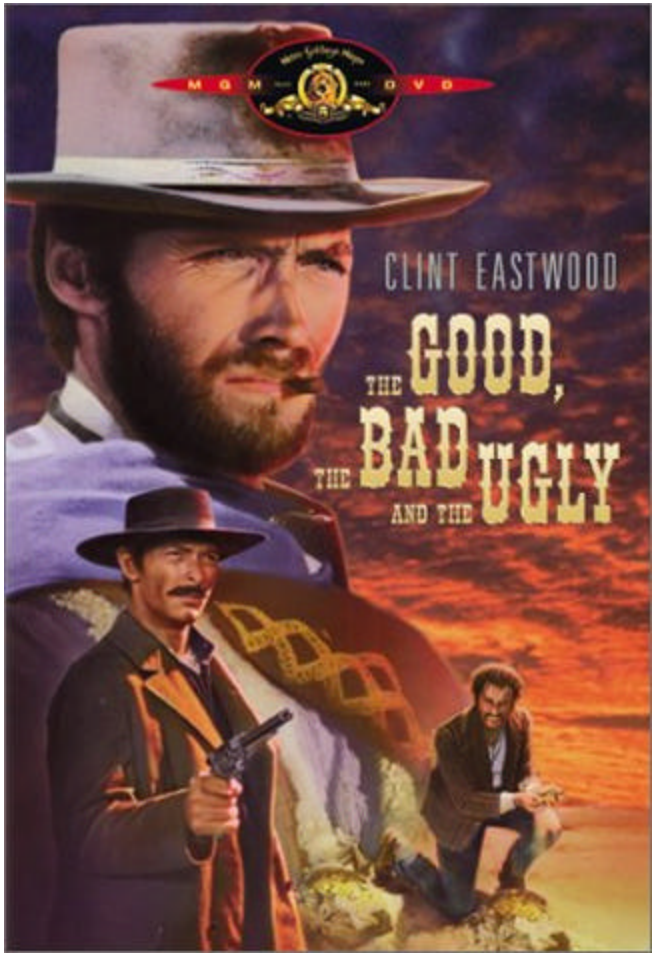
- Goals revisited
  - Fast issuing
    - Random number generation N_c and table lookup for D_c
  - Fast verification
    - Single `SHA1` hash (~ 1μs on commodity PC)
  - Flexible binding
    - `F` can be any property of request (IP addresses, ports, URIs)
  - Limited pre-computation and replay
    - Server updates N_c to invalidate previous answers

The End

- Unwanted traffic meets its match
- The Internet is saved
- All is good in the world again

Well, not really…

Well, not really…



The good

Problems made easier

Floods against issuer

- Public work function needs to be given once per client
  - N_c easy to generate
  - D_c easy to lookup
- Floods against issuer easily identified and dropped

Floods against verifier and network

- Verification is efficient
  - Look up public work function
  - Perform a single SHA1 hash on A, F, N_c
- Verification done near adversary
  - Unwanted traffic identified and dropped at source edge
  - Adversary cannot flood links to the issuer and verifier
- Adversary forced to expend arbitrary resources to attack system

The bad

Problems that still need work

Granularity

- What should public work functions be attached to?
  - F is unspecified
  - Attach to keys or files in DHTs and P2P networks?
  - Attach to DNS names?
  - Attach to HTTP URIs?
  - Attach to TCP/UDP 5-tuples?
  - Attach to IP source/destination addresses?

Delivery

- Mutual assured delivery of work functions
  - Spoofing work functions and requests for work functions
  - Client must know that the public work function is authentic
  - Server must know that the public work function has been delivered to the right client
- Approaches for addressing delivery problems
  - Strong: public-key certificates
    - SSL/X.509 certificates for TLS and DNSsec
  - Weak: three-way handshakes
    - TCP seq. #s
    - DNS request IDs

Difficulty

- Uniform difficulties are bound to fail
  - Adversaries can co-opt much more resources than individuals (i.e. Botnets with > 100k machines)
- Must give difficult functions to malicious users
  - Must have an accounting mechanism to track usage history (counting Bloom filters)
  - Must have a difficulty generation algorithm that can turn back targeted attacks

Spoofing

- Attributing activity to others
  - Spoofing requests with valid work from victim to increase its difficulty
  - Spoofing work function requests to disable a victim issuer
  - Spoofing requests from a targeted victim client to a large number of issuers (reflector)
- Must be reduced to make public work systems “work”

The ugly

Problems that are added

The ugly

Problems that are added

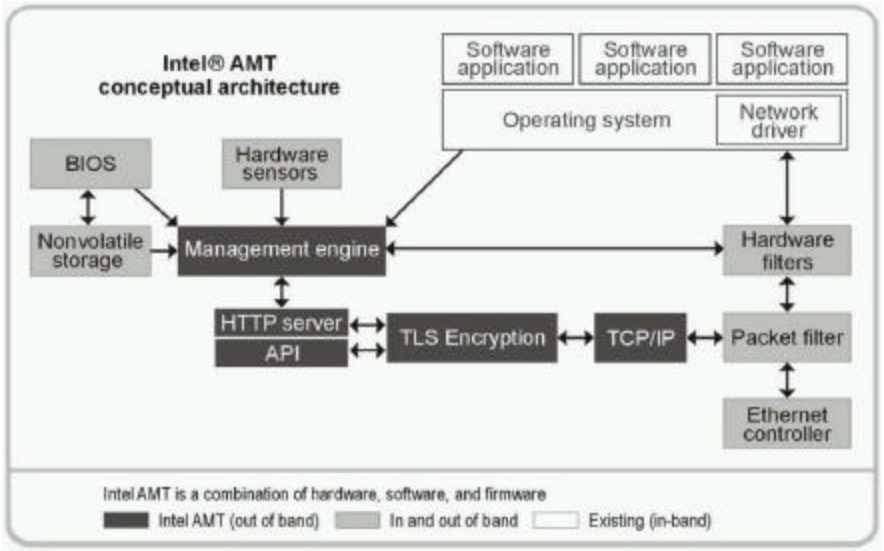
Time check?

Replay

- Preventing pre-computation and replay attacks
  - Adversary continuously re-uses a previously solved public work function
  - Adversary solves a bunch of work functions in advance and bombs service at selected time
- Must “freshen” public work function N_c, D_c
  - Pre-computation limited to time since last update
  - Replay limited to time until next update

Asymmetry

- Verifier must be along the path to and from service
  - Must observe, validate, and store public work function
  - Must verify subsequent answers attached to requests
  - Requires path symmetry that does not exist generally
- Addressing asymmetry
  - Secure sharing of public work for multi-homed client ASs
  - Verifiers at first-hop routers or on the client itself
    - A clean-slate proposal for using trusted hardware in networks



Statefulness

- Requires per-flow state to be kept at the verifier
  - Depends on F
  - Public work functions must be stored at the verifier for validating subsequent answers
  - Scalable only when verifiers are at the edges of the network
    - At client edge, per-flow state scales
    - At server edge, issuer can use keyed hashes to generate and validate N_c from a single secret N_i that is shared with server-side verifiers
      - e.g. $N_c = HMAC_{N_I}(IP_c, D_c)$

Status

- Implementations
  - DNS system
    - Via modifications to `bind` and `iptables`
    - Uses iterative DNS queries to deliver work function
  - HTTP system
    - Via new `apache` module and a client-side `javascript` solver
  - TCP system
    - Via modifications to `iptables`
- Simulator

  http://thefengs.com/wuchang/work/courses/cs592_spring2006

Conclusion

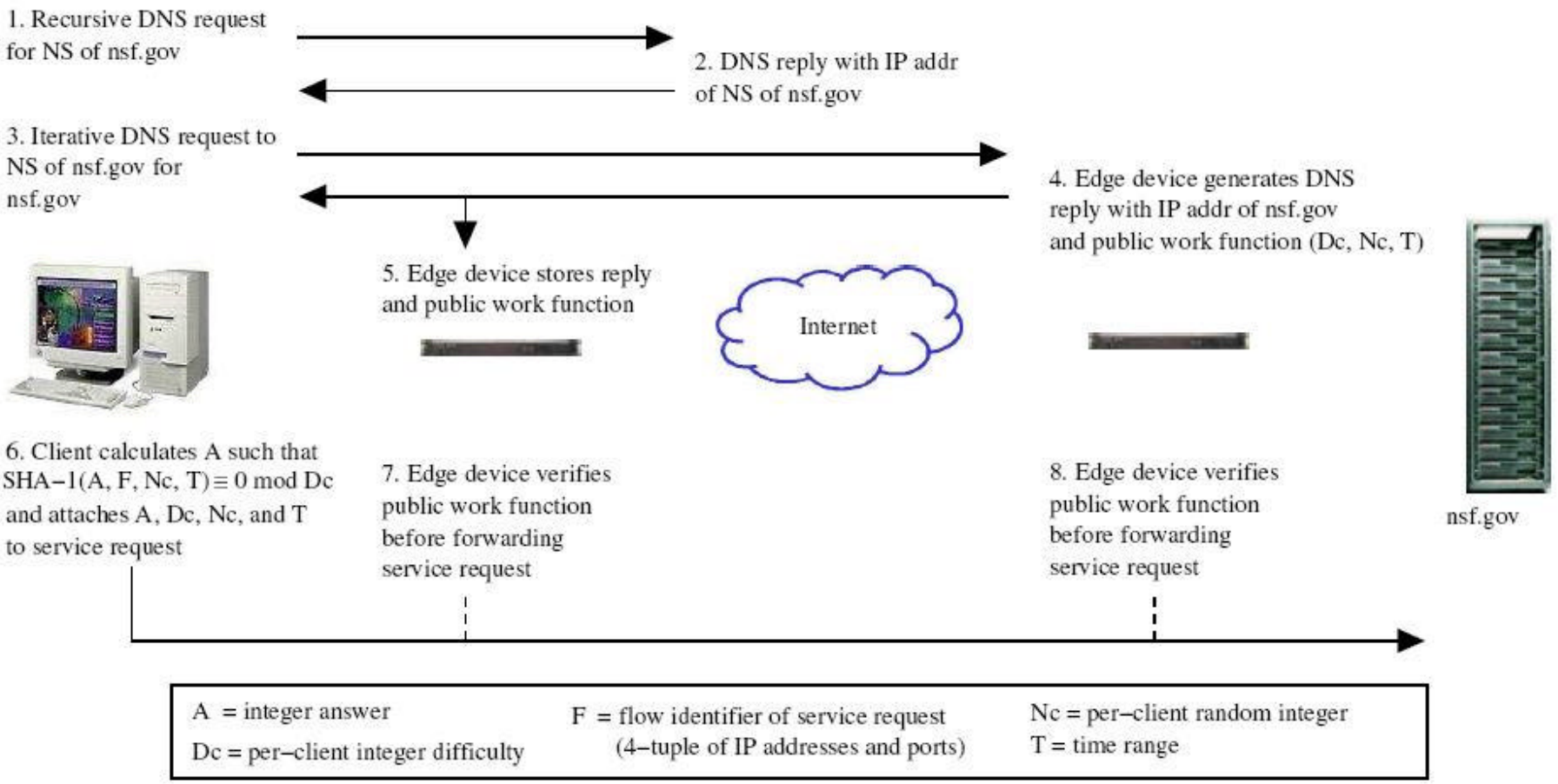
- Public work functions
  - Combine key aspects of indirection, filtering, capabilities, and proof-of-work
  - Single per-client work function prevents floods against the issuer
  - Public traffic validation prevents floods against the verifier
  - Lots of problems to be solved still!

Questions?

http://thefengs.com/wuchang/work/puzzles

Extra slides

DNS system


1. Recursive DNS request for NS of nsf.gov
2. DNS reply with IP addr of NS of nsf.gov
3. Iterative DNS request to NS of nsf.gov for nsf.gov
4. Edge device generates DNS reply with IP addr of nsf.gov and public work function (Dc, Nc, T)
5. Edge device stores reply and public work function
Internet
6. Client calculates A such that SHA−1(A, F, Nc, T) ≡ 0 mod Dc and attaches A, Dc, Nc, and T to service request
7. Edge device verifies public work function before forwarding service request
8. Edge device verifies public work function before forwarding service request
nsf.gov
A = integer answer
Dc = per−client integer difficulty
F = flow identifier of service request (4−tuple of IP addresses and ports)
Nc = per−client random integer
T = time range

Public work management

- Query bloom filter $Q_C(T)$
  - Keeps track of DNS requests per client
- Resource bloom filter $R_C(T)$
  - Keeps track of current resource consumption per client
  - Only update for requests with valid work
- Averaging bloom filter $M_C(T)$
  - Weighted smoothed average of resource bloom filter
- Generated difficulty $D_C(T)$
  - Derived as $g(M_C(T))$

Workload vs. Difficulty