DESIGN AND DEPLOYMENT OF COCA

F B Schneider

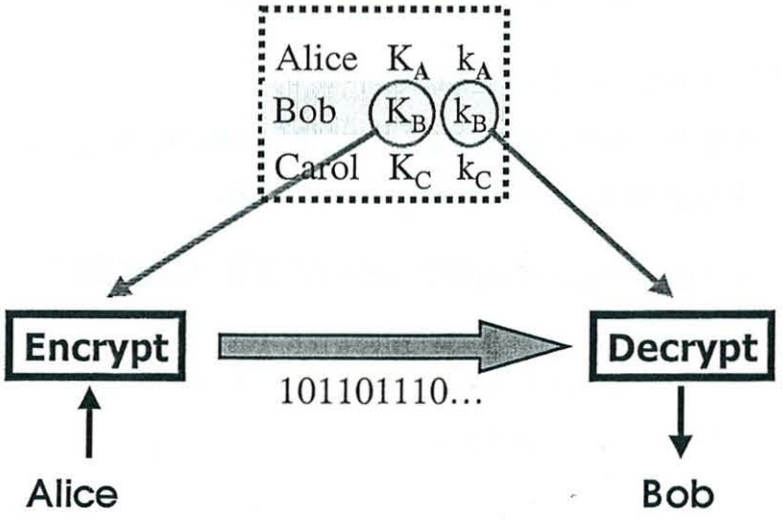
Rapporteur: I S Welch

Design and Deployment of COCA

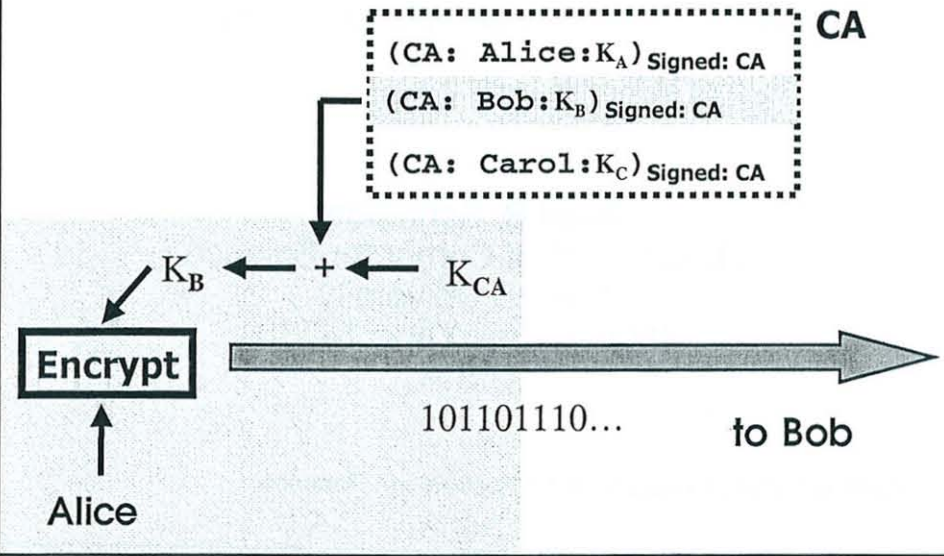
Fred B. Schneider
Department of Computer Science
Cornell University
Ithaca, New York 14853
U.S.A.

Joint work with Lidong Zhou and Robbert van Renesse.

Public Key Cryptography



Using a Certification Authority


CA
(CA: Alice:K_A) Signed: CA
(CA: Bob:K_B) Signed: CA
(CA: Carol:K_C) Signed: CA
K_B
+
K_{CA}
Encrypt
101101110...
to Bob
Alice

Certification Authority

- CA stores certificates.
  - Each certificate is a binding: ⟨name, K_{name}⟩
  - Each certificate is signed by CA.
- Clients know public key of CA. Clients issue requests:
  - Query to retrieve certificate for a name.
  - Update to change binding and invalidate certificate.

CA Security and Fault-tolerance

Fault-tolerance and security for a CA means

- CA service remains available.
- CA signing key remains secret.

despite

- failures (=independent events) and
- attacks (=correlated events).

COCA (Non)-Assumptions

- **Servers**: correct or compromised. At most t servers compromised during window of vulnerability, and 3t < n holds.
- **Fair Links**: A message sent enough times will be delivered.
- **Asynchrony**: No bound on message delivery delay or server speed.

Weaker assumptions are better.

Security and Fault-tolerance:

Query and Update

Dissemination Byzantine Quorum System:

- Intersection of any two quorums contains at least one correct server.
- A quorum comprising only correct servers always exists.

- Replicate certificates at servers.
- Each client request processed by all correct servers in some quorum.
- Use service (not server) signing key.

Security and Fault-tolerance:

Service Signing Key Secrecy

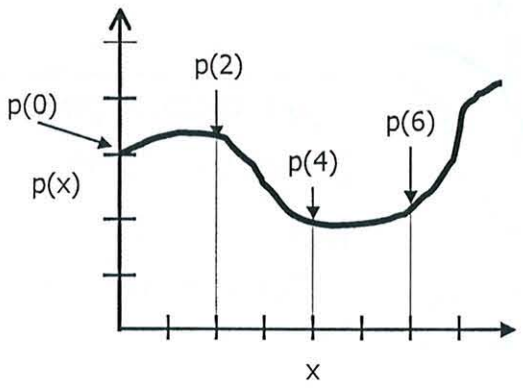
- Service signing key stored at each server.

versus

- Employ threshold signature protocol:
  - Store a share of signing key at each server.
  - Use (n, t+1) threshold cryptography to sign.

Security and Fault-tolerance:

Secret Sharing



p(0) is secret

p(i) is share at site i

m points determine an m-1 degree poly

(n,k) secret sharing: k-1 degree poly

8

Security and Fault-tolerance:

Mobile Virus Attacks

- Compromise server CA_1, detect, repair.
- Compromise server CA_2, detect, repair.

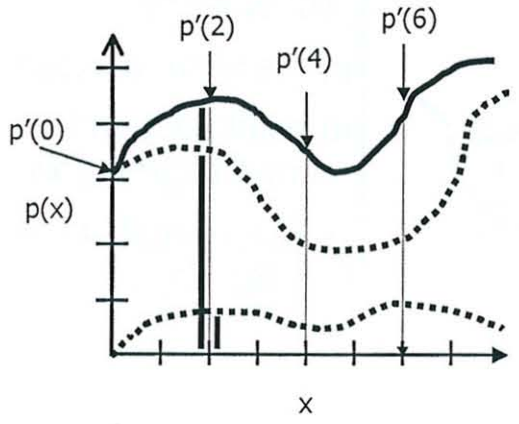
  ...
- Compromise server Ca_{t+1}, detect, repair.

t+1 secret shares revealed, even though at most 1 site ever compromised.

9

Security and Fault-tolerance—Mobile Virus Attacks:

Proactive Secret Sharing



q(x): random poly

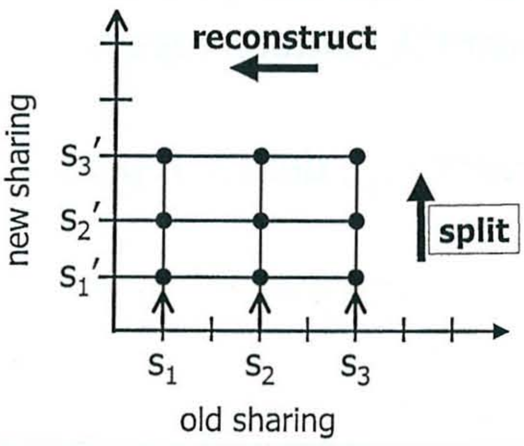
p'(x): p(x)+q(x)

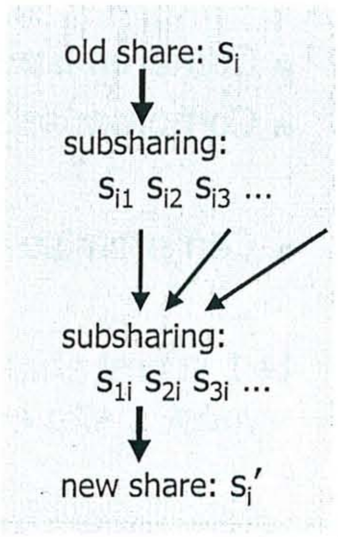
p'(0) = p(0)

p'(i) is share at site i

Proactive Secret Sharing:

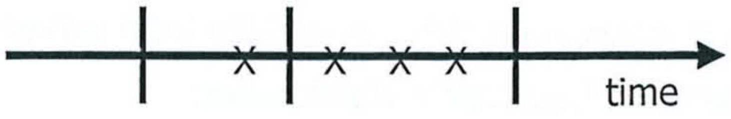
Computing New Shares




old share: s_i
subsharing:
s_{i1} s_{i2} s_{i3} ...
subsharing:
s_{1i} s_{2i} s_{3i} ...
new share: s_i'

Proactive Secret Sharing:

Windows of Vulnerability



| proactive refresh
X server compromise

- At most t servers compromised in a window.
- Shares, keys, state all refreshed.
- Local clock at some server initiates refresh.
- Denial of service increases window size.

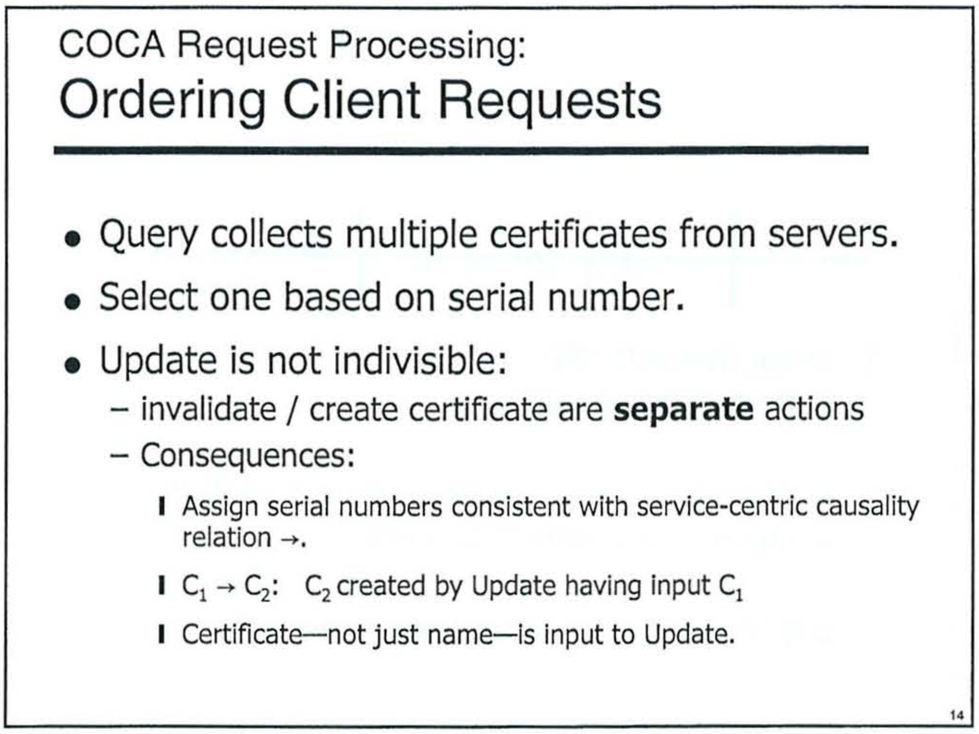
12

COCA Request Processing

- Client issues request and awaits response.
- COCA accepts request:
  - Some correct COCA server received request.
- COCA completes request:
  - Some correct COCA server constructs response.

Liveness: Every accepted request eventually is completed.

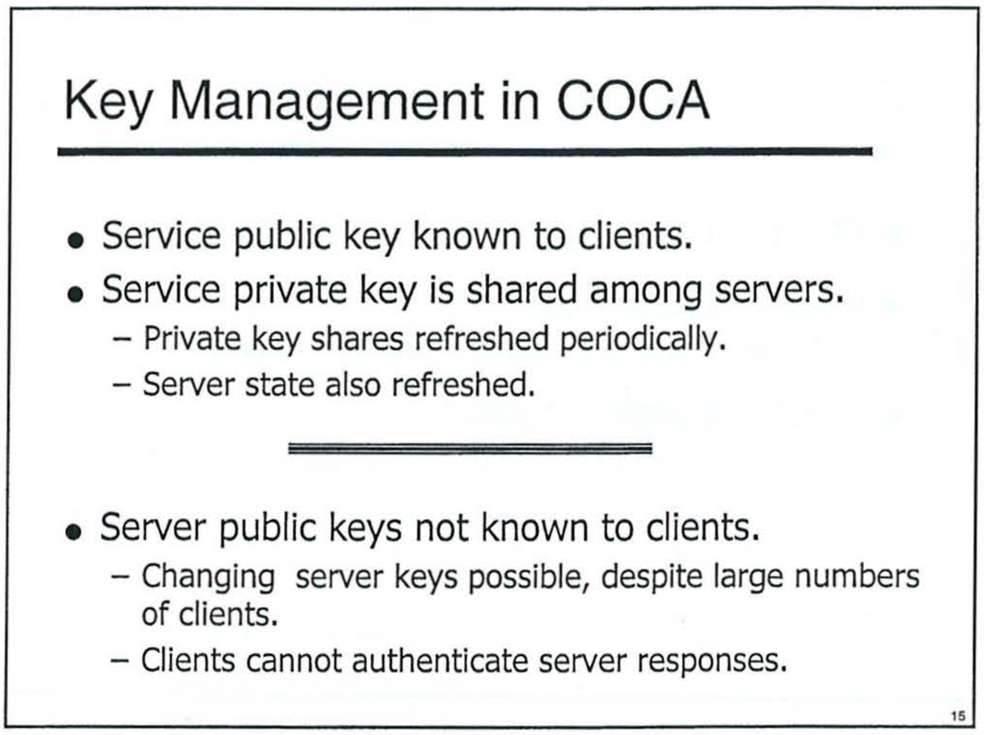
13



COCA Request Processing:
Ordering Client Requests

- Query collects multiple certificates from servers.
- Select one based on serial number.
- Update is not indivisible:
  - invalidate / create certificate are **separate** actions
  - Consequences:
    - Assign serial numbers consistent with service-centric causality relation $\rightarrow$.
    - $C_1 \rightarrow C_2$: C_2 created by Update having input C_1
    - Certificate—not just name—is input to Update.

14



Key Management in COCA

- Service public key known to clients.
- Service private key is shared among servers.
  - Private key shares refreshed periodically.
  - Server state also refreshed.

- Server public keys not known to clients.
  - Changing server keys possible, despite large numbers of clients.
  - Clients cannot authenticate server responses.

15

Role of Delegates

Problem: Without server public keys ...

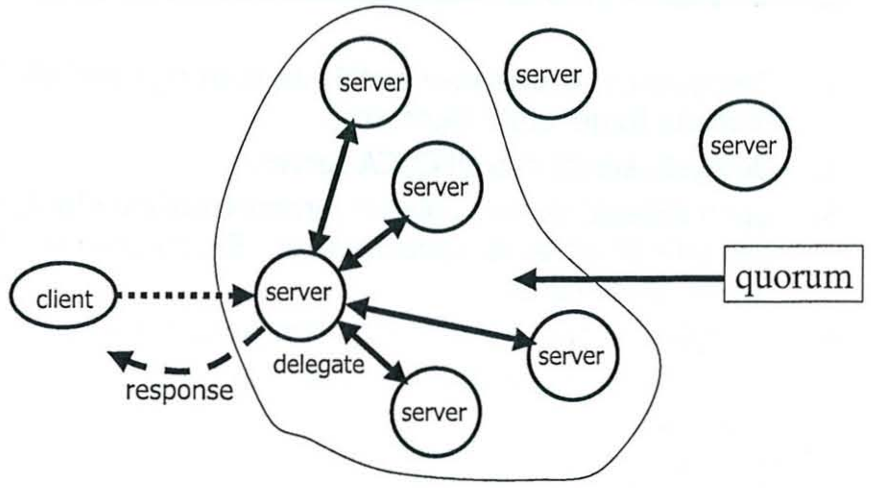
- Clients cannot authenticate messages from servers.
- Clients cannot determine whether a request has been processed by a quorum.

Solution: Delegate collects responses.

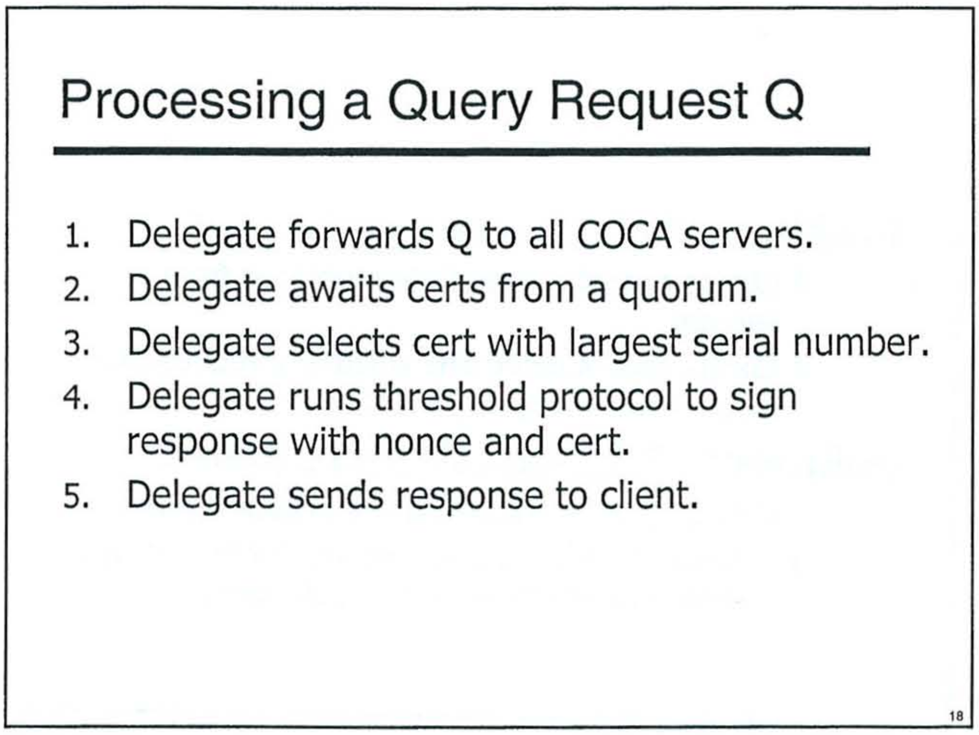
- Client requests are signed and include nonce.
- Delegate handles request on behalf of client. It is a server and it knows COCA public keys.

16

COCA Architecture



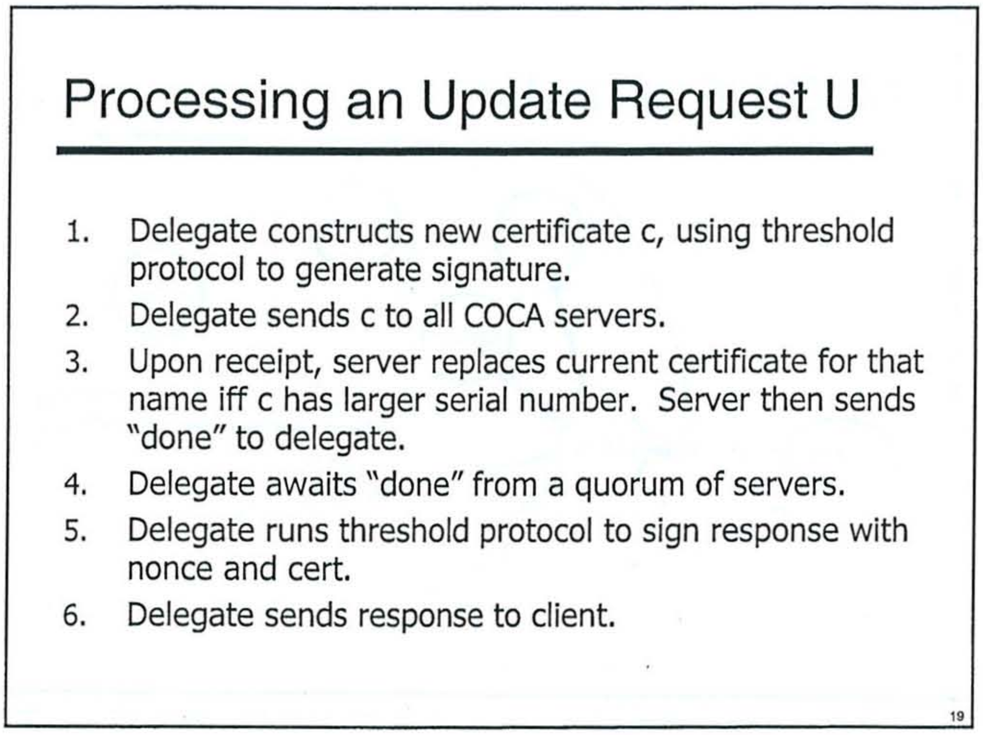
17



Processing a Query Request Q

1. Delegate forwards Q to all COCA servers.
2. Delegate awaits certs from a quorum.
3. Delegate selects cert with largest serial number.
4. Delegate runs threshold protocol to sign response with nonce and cert.
5. Delegate sends response to client.

18

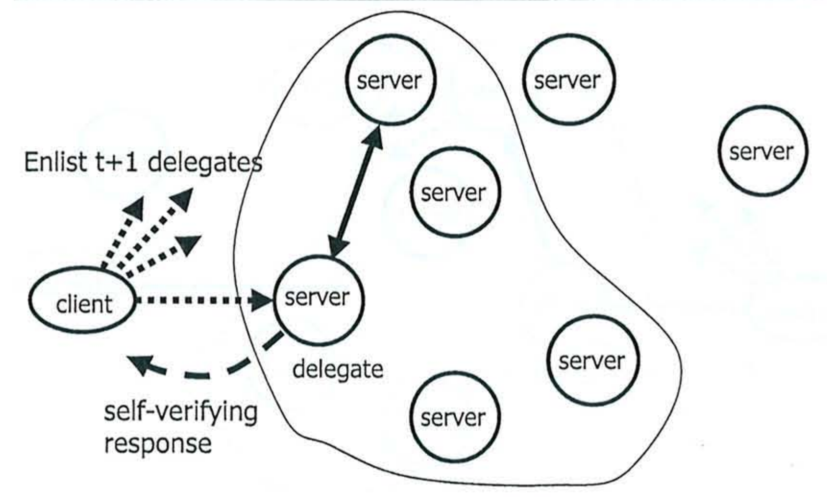


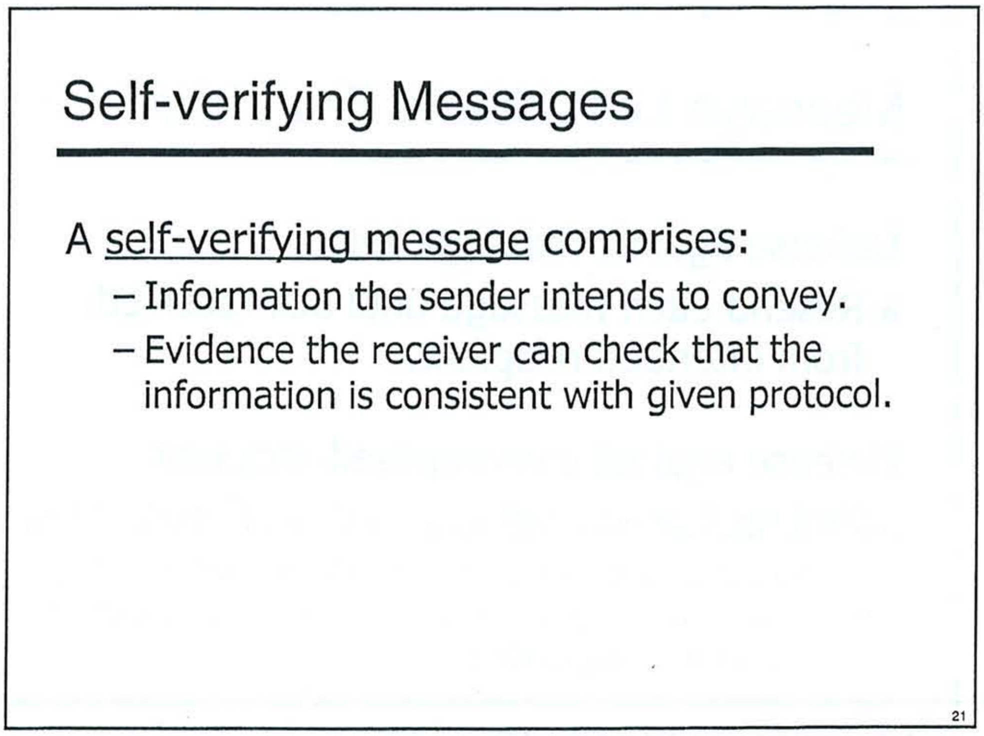
Processing an Update Request U

1. Delegate constructs new certificate c, using threshold protocol to generate signature.
2. Delegate sends c to all COCA servers.
3. Upon receipt, server replaces current certificate for that name iff c has larger serial number. Server then sends "done" to delegate.
4. Delegate awaits "done" from a quorum of servers.
5. Delegate runs threshold protocol to sign response with nonce and cert.
6. Delegate sends response to client.

19

Compromised Delegate



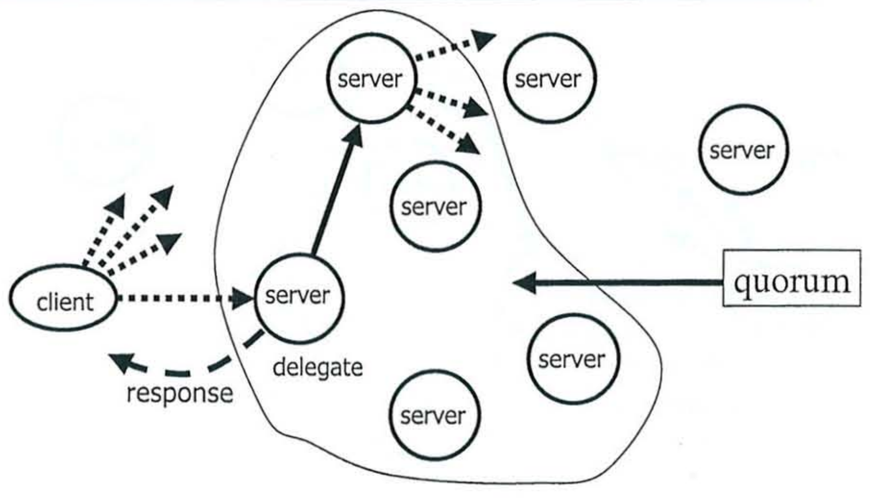


Self-verifying Messages

A <u>self-verifying message</u> comprises:

- Information the sender intends to convey.
- Evidence the receiver can check that the information is consistent with given protocol.

Compromised Client



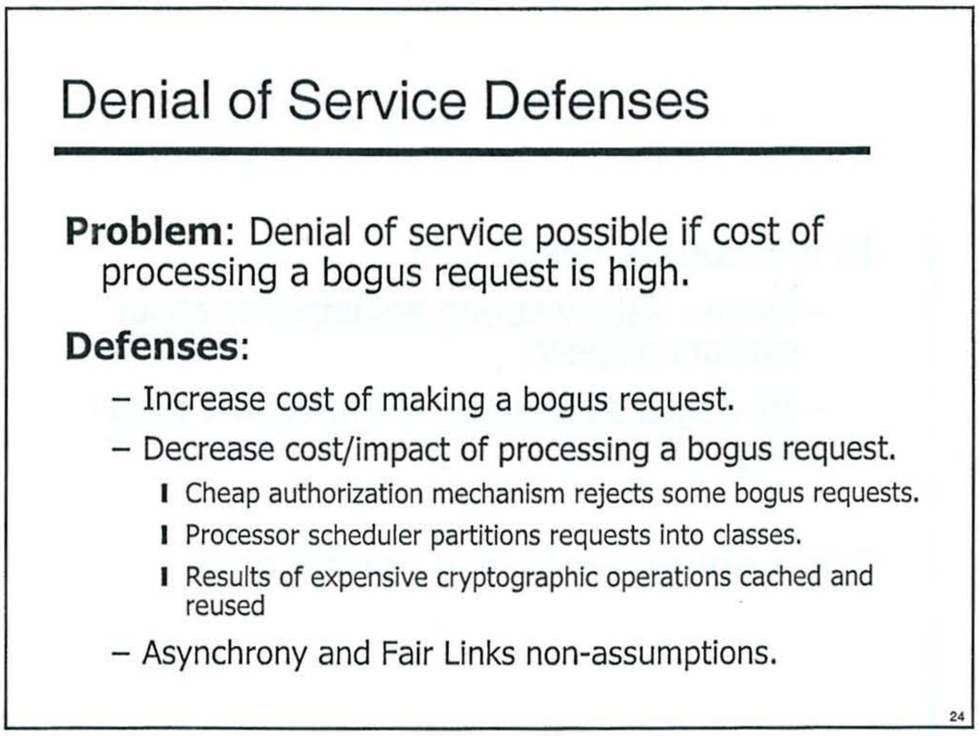
Message Loss due to Fair Links

Defense against message loss ...

- Resend each message until ack received from intended recipient.

Defense against compromised recipient ...

- Protocol structured as a series of multicasts.
  - If ack received from enough recipients, halt resending.
  - Ensure there are enough correct recipients even if t servers are compromised.



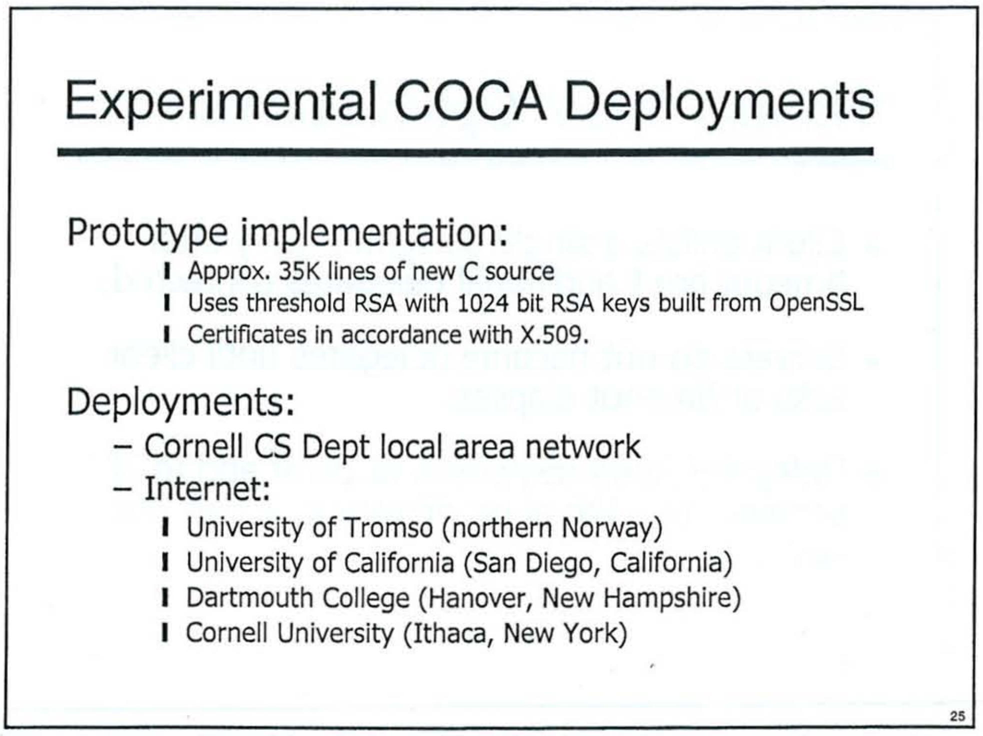
Denial of Service Defenses

Problem: Denial of service possible if cost of processing a bogus request is high.

Defenses:

- Increase cost of making a bogus request.
- Decrease cost/impact of processing a bogus request.
  - Cheap authorization mechanism rejects some bogus requests.
  - Processor scheduler partitions requests into classes.
  - Results of expensive cryptographic operations cached and reused
- Asynchrony and Fair Links non-assumptions.

24



Experimental COCA Deployments

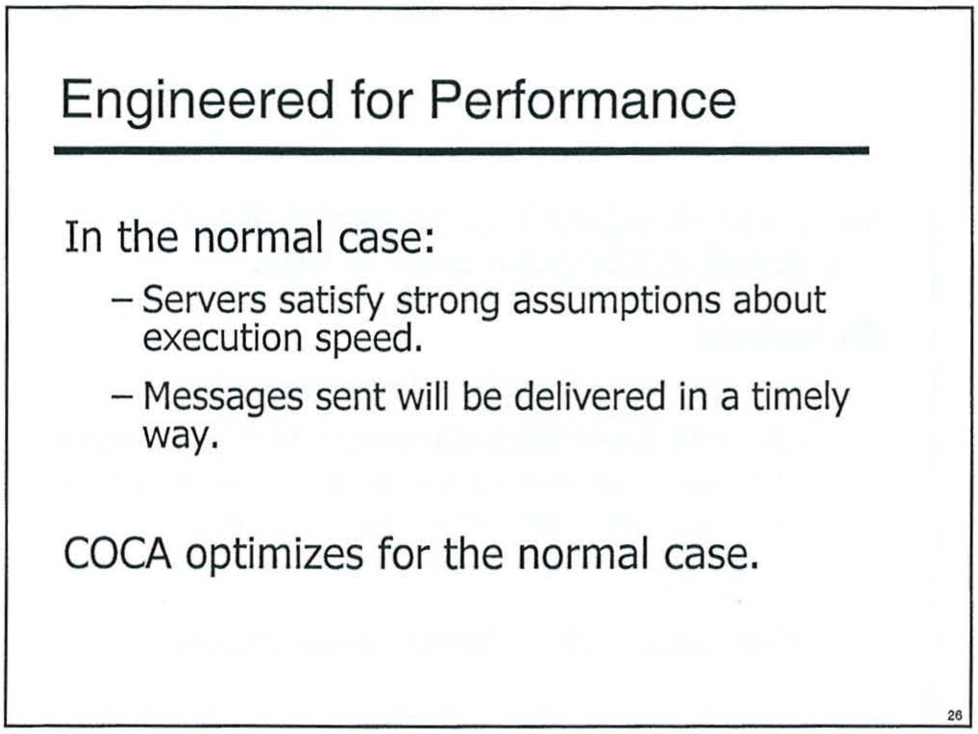
Prototype implementation:

- Approx. 35K lines of new C source
- Uses threshold RSA with 1024 bit RSA keys built from OpenSSL
- Certificates in accordance with X.509.

Deployments:

- Cornell CS Dept local area network
- Internet:
  - University of Tromso (northern Norway)
  - University of California (San Diego, California)
  - Dartmouth College (Hanover, New Hampshire)
  - Cornell University (Ithaca, New York)

25

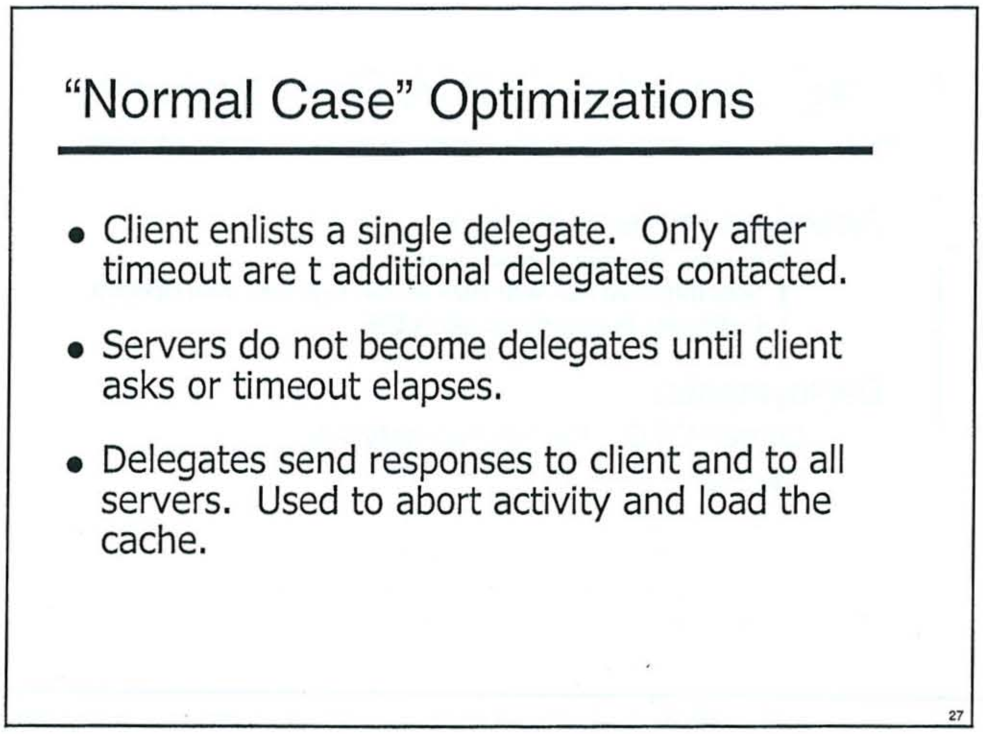


Engineered for Performance

In the normal case:

- Servers satisfy strong assumptions about execution speed.
- Messages sent will be delivered in a timely way.

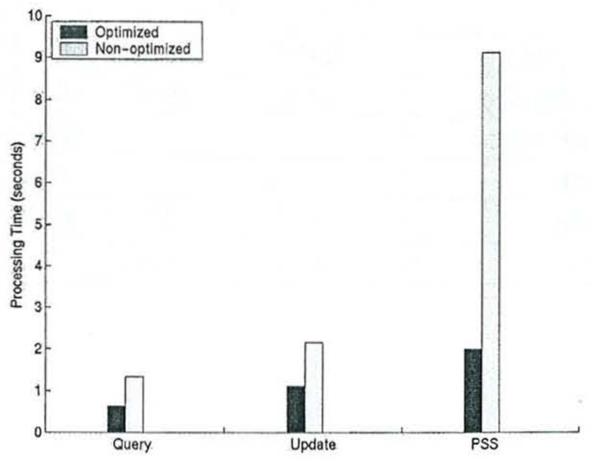
COCA optimizes for the normal case.



"Normal Case" Optimizations

- Client enlists a single delegate. Only after timeout are t additional delegates contacted.
- Servers do not become delegates until client asks or timeout elapses.
- Delegates send responses to client and to all servers. Used to abort activity and load the cache.

"Normal Case" Optimizations



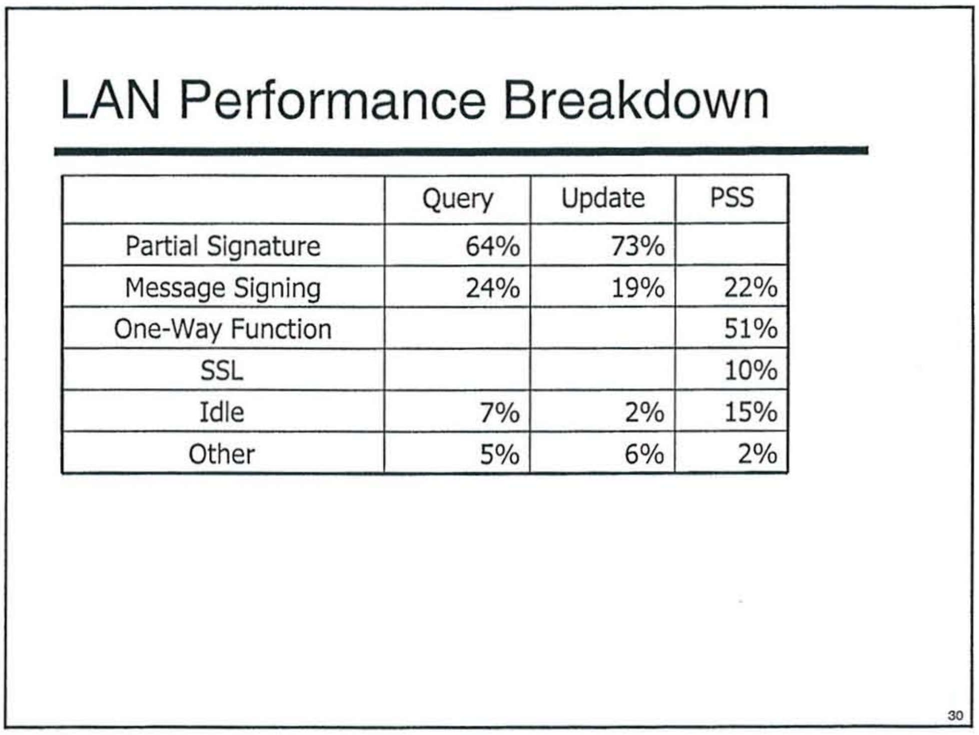
LAN Performance Data

COCA Operation	Mean *(msec)*	Std dev. *(msec)*
Query	629	16.7
Update	1109	9.0
PSS	1990	54.6

4 Sun E420R SPARC servers (4 450 Mhz processors. Solaris 2.6)

100 Mb Ethernet (Round trip delay for UDP packet: 300 micro secs)

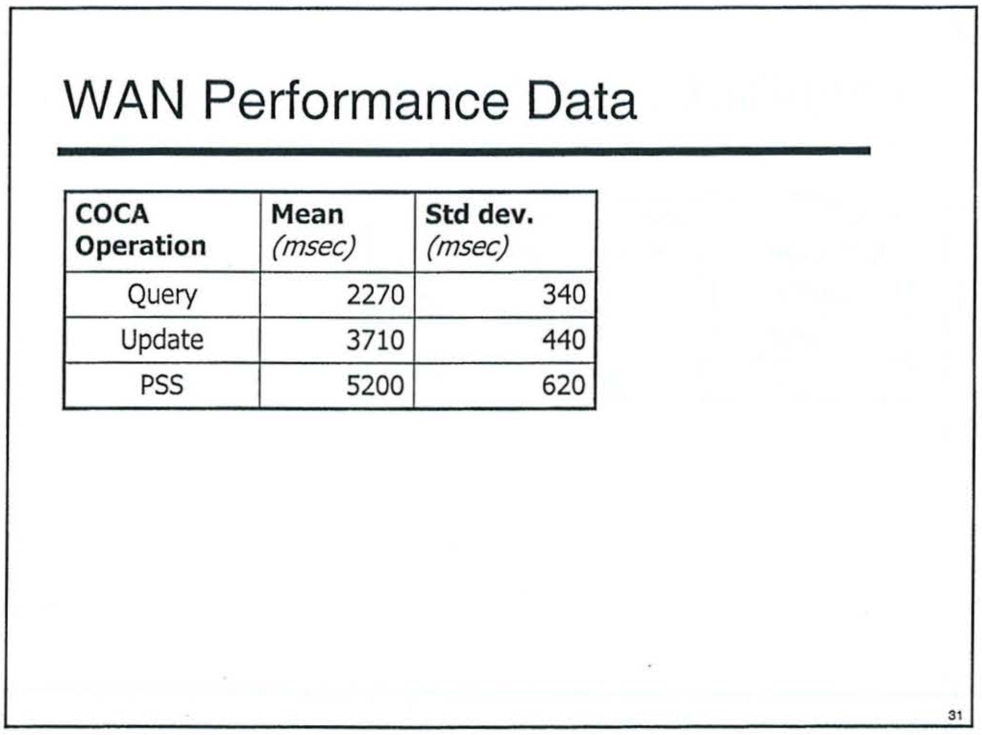
Sample means for 100 executions.



LAN Performance Breakdown

	Query	Update	PSS
Partial Signature	64%	73%	
Message Signing	24%	19%	22%
One-Way Function			51%
SSL			10%
Idle	7%	2%	15%
Other	5%	6%	2%

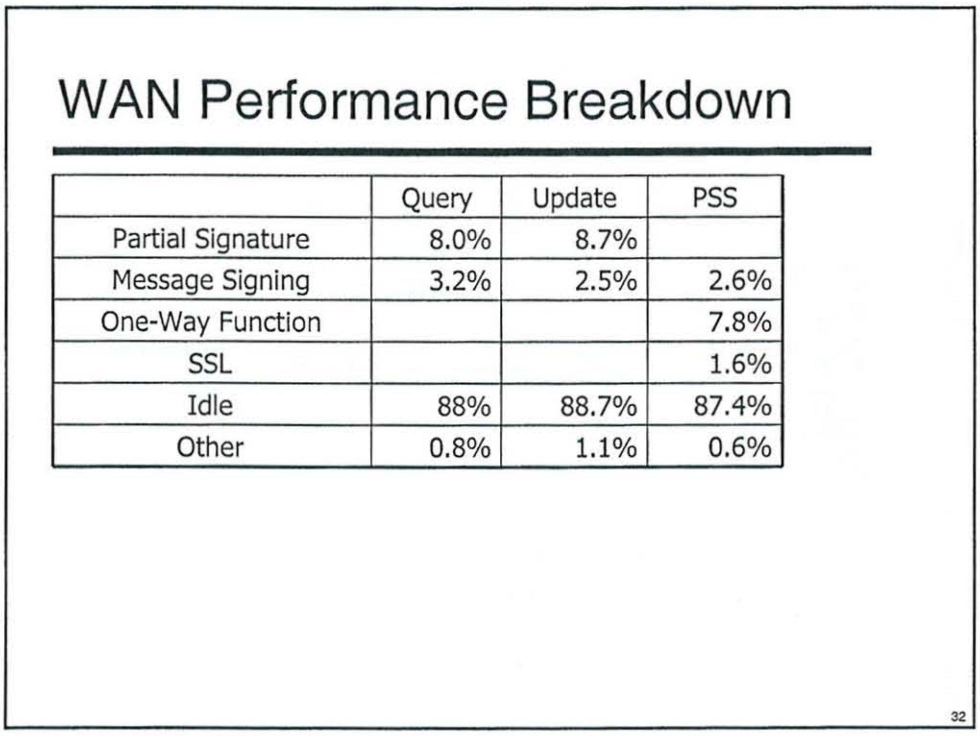
30



WAN Performance Data

COCA Operation	**Mean** *(msec)*	**Std dev.** *(msec)*
Query	2270	340
Update	3710	440
PSS	5200	620

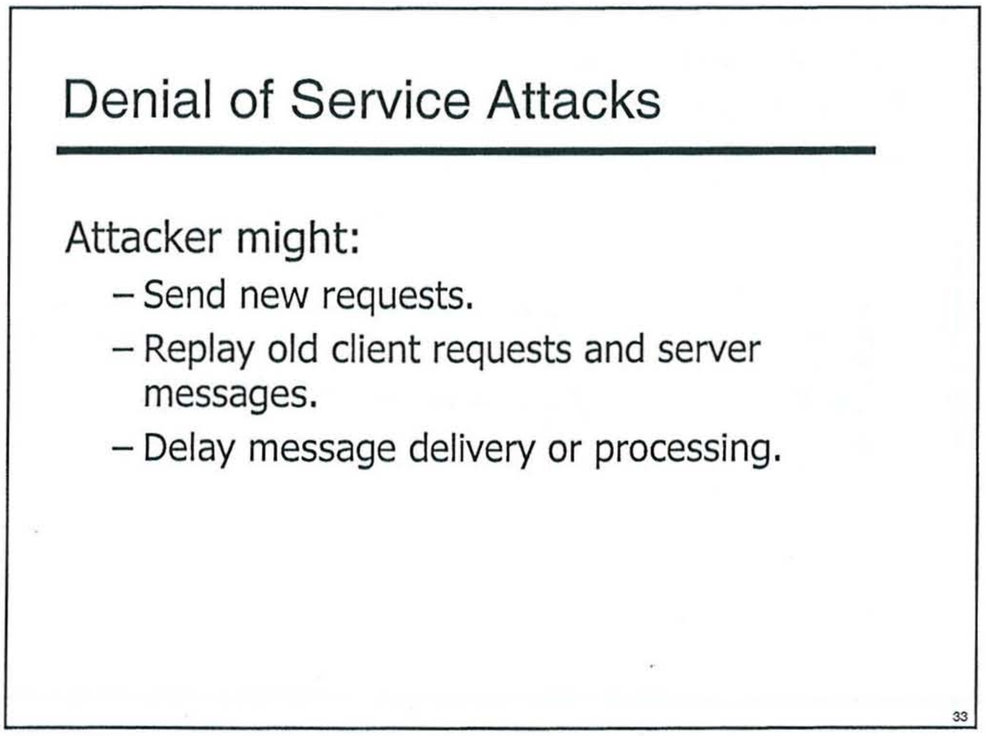
31



WAN Performance Breakdown

	Query	Update	PSS
Partial Signature	8.0%	8.7%	
Message Signing	3.2%	2.5%	2.6%
One-Way Function			7.8%
SSL			1.6%
Idle	88%	88.7%	87.4%
Other	0.8%	1.1%	0.6%

32

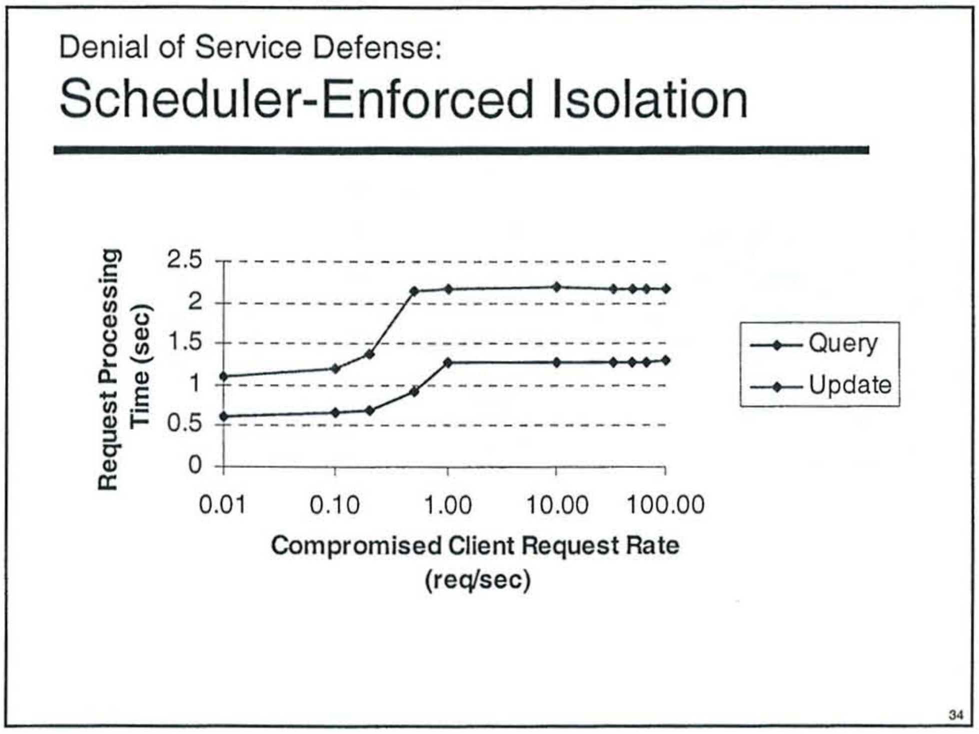


Denial of Service Attacks

Attacker might:

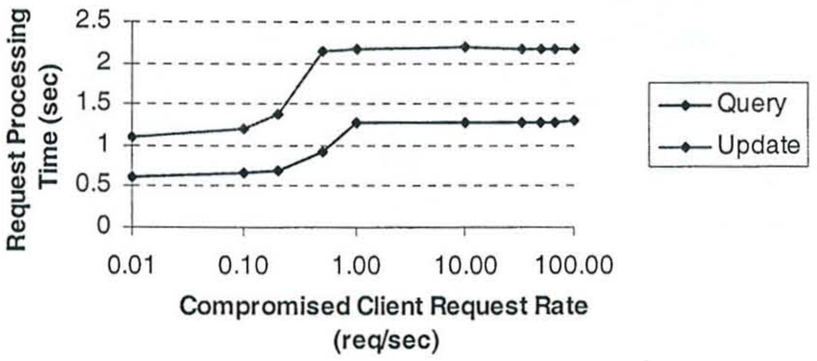
- Send new requests.
- Replay old client requests and server messages.
- Delay message delivery or processing.

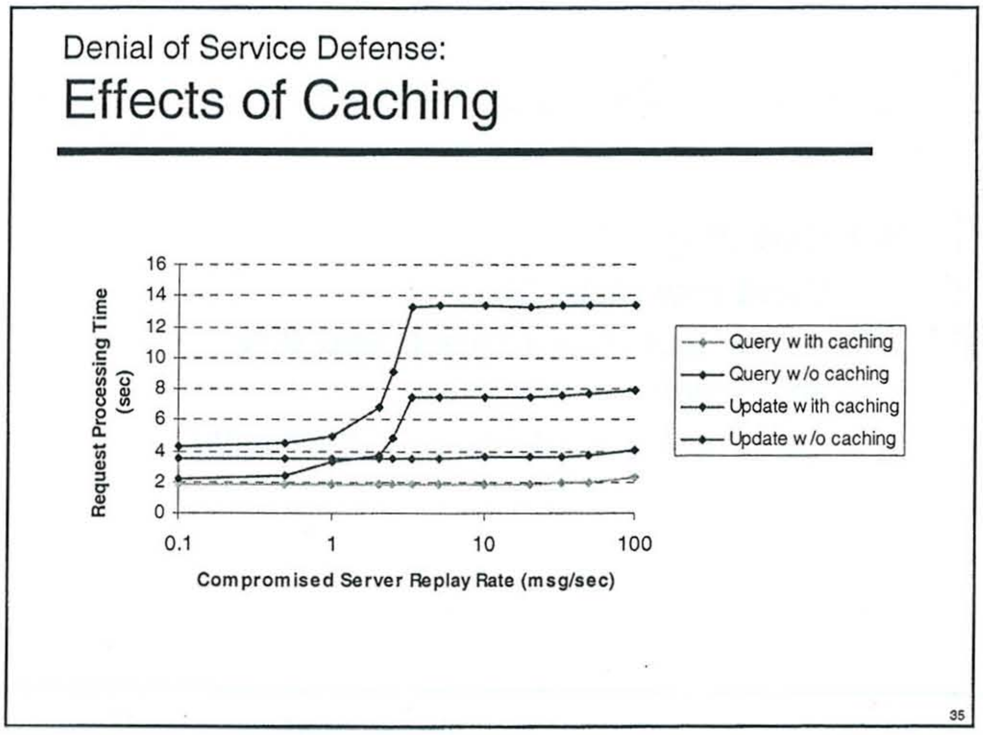
33



Denial of Service Defense:

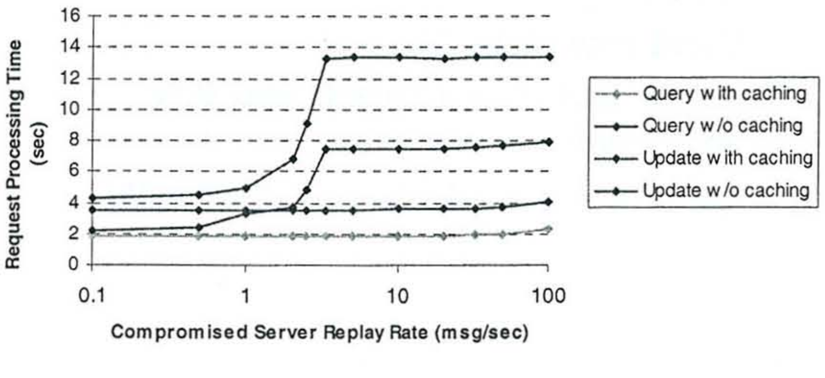
Scheduler-Enforced Isolation





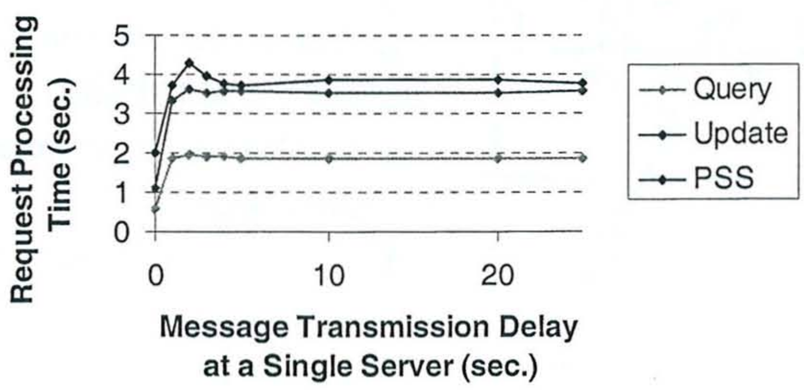
Denial of Service Defense:

Effects of Caching



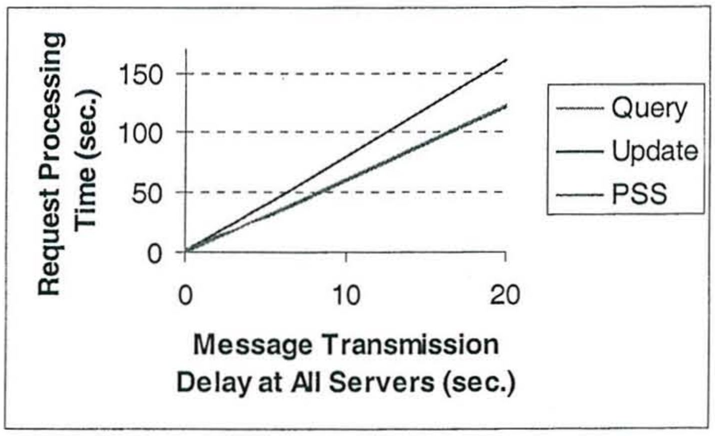
Denial of Service Defense:

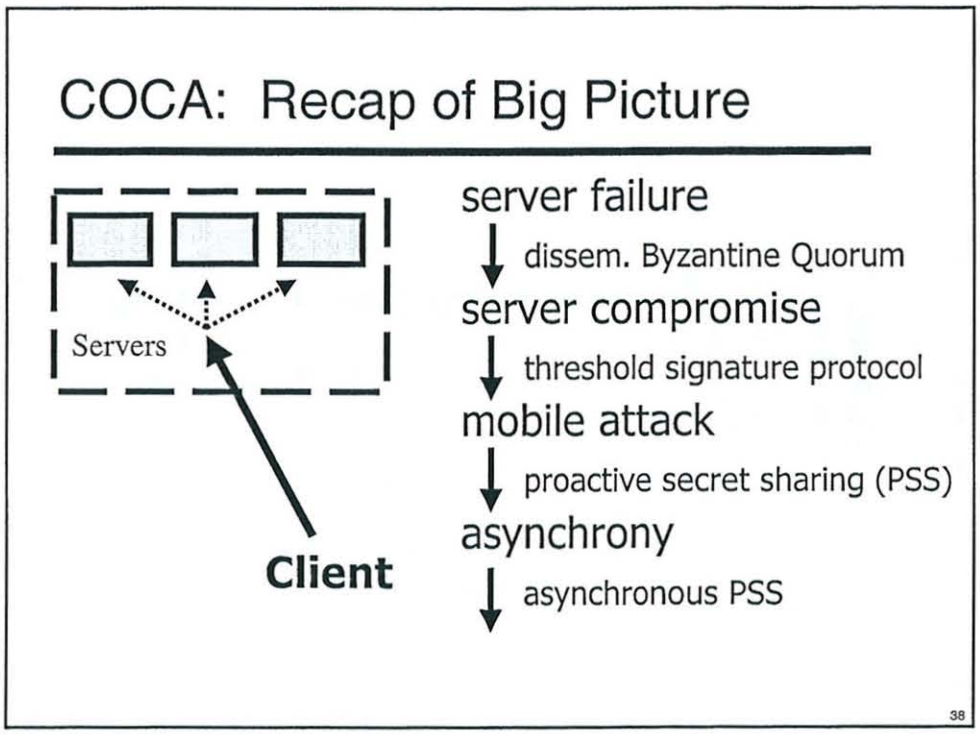
Effects of Message Delay



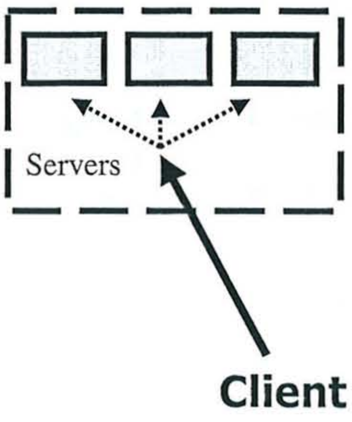
Denial of Service Defense:

Effects of Message Delay





COCA: Recap of Big Picture



server failure
↓ dissem. Byzantine Quorum
server compromise
↓ threshold signature protocol
mobile attack
↓ proactive secret sharing (PSS)
asynchrony
↓ asynchronous PSS

DISCUSSION

Rapporteur: I S Welch

Lecture One

Dr Xu asked Professor Schneider why he assumed 3t+1 replicated servers were required to tolerate t intrusions rather than 2t+1? Professor Schneider replied that the reason for this would fall out in subsequent discussion although by weakening the starting assumptions it may be possible to achieve a better bound.

Professor Randell asked if the worst case was a Byzantine fault model. A worse case might be where a Byzantine general is allowed to replicate itself and change the rules of the game. Professor Schneider answered that the original framing of the Byzantine general's problem assumed that the generals were hardware and therefore were not able to replicate themselves. If the Byzantine generals are processes then replication and voting more than once can be prevented by using cryptographic techniques to sign votes.

Professor Burns asked why not just define liveness to be with respect to a correct client? Professor Schneider replied that he suspected that not defining this way might lead to subtle compromises.

Dr Ezhilchelvan asked whether, in proactive secret sharing, the old key is destroyed? Professor Schneider replied that at the end of a window of possible compromise then the server discards the old shares. Dr Ezhilchelvan asked what would happen if we wanted to find out past behaviour - it may be that the old key is needed for this. Professor Schneider said that the question goes to the heart of the threshold signature scheme. The private key doesn't change from the old share to the new share, just the shares. The new shares are generated without revealing the private key.

Dr Rushby asked if Professor Schneider could contrast the approach described here with the Omega system. Professor Schneider replied that the main difference was that Omega assumed a synchronous network as opposed to an asynchronous network. Dr Rushby asked whether Omega couldn't also be used in the context of an asynchronous network if ISIS was used. Professor Schneider replied that the ISIS approach used failure detectors to mask the asynchronous nature of the network and the failure detectors were the problem. An attacker can attack the failure detectors and cause them to be always reconfiguring. A better approach (as taken in the work by Castro and Liskov) is to use algorithms that do not guarantee liveness but do guarantee safety properties.

There was some discussion about whether the approach taken here (that did not guarantee a timely response) was appropriate. Professor Schneider made the point that his approach was to work on the worst case but to look at optimisations. In the normal case messages will be delivered within a bound; only in extreme cases that this will not be satisfied.

Dr Xu asked if the 't's in the service and the 't' used when specifying the threshold scheme are related. Professor Schneider replied that they are related. The 3t+1 is a result of using the Byzantine quorum system.
Professor Shrivastava made the point that if an adversary manages to slow the network the service can be blocked.