

# Network Security

(assorted topics)

---

# Network threats and defenses

---

- Most traffic on the Internet is unencrypted and unauthenticated.
- This allows a range of attacks.  
(But adding cryptography is not always a satisfactory solution!)
- It also motivates a range of attack-detection techniques and defenses.

# Network mapping

---

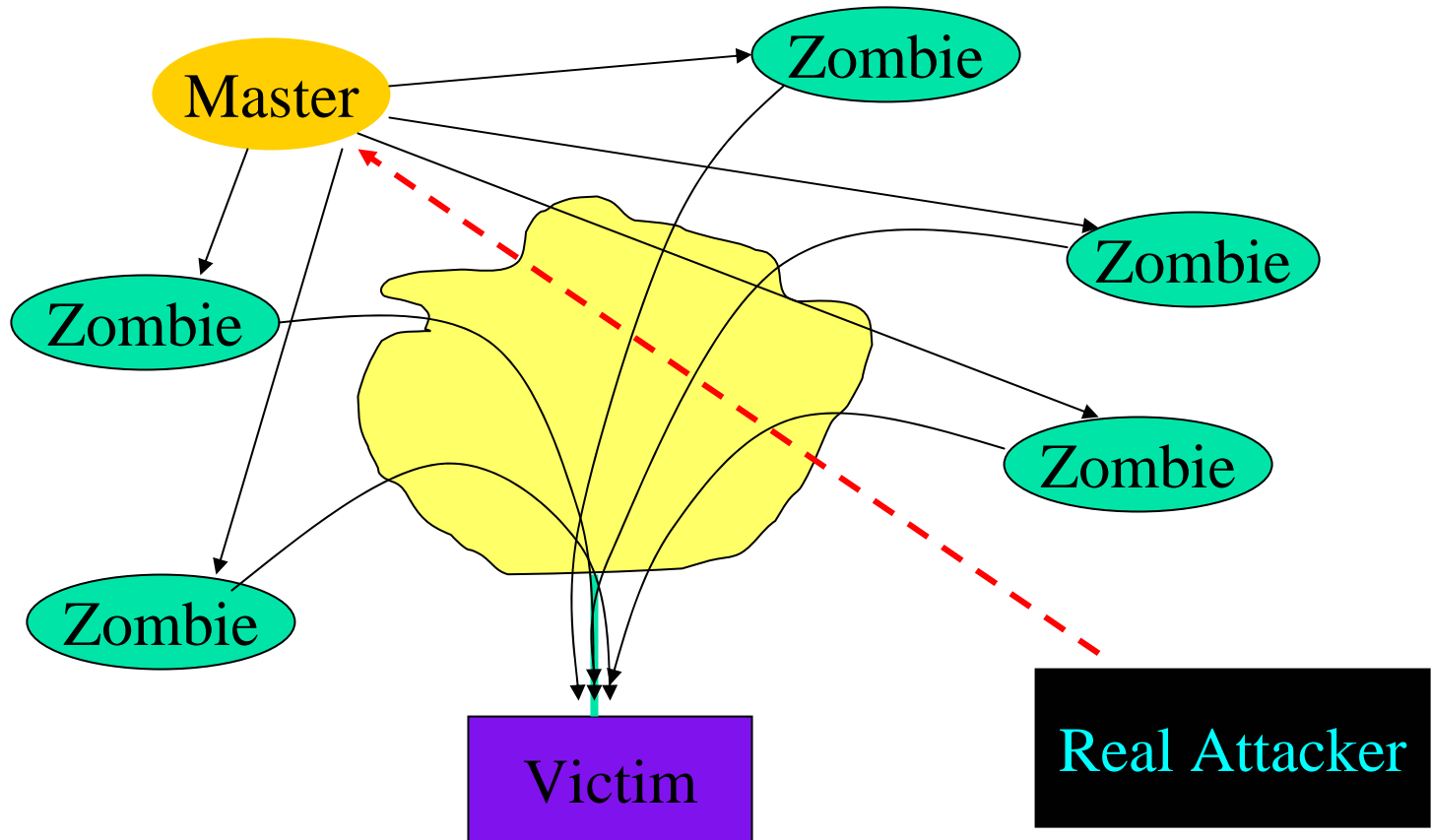
- Attacker probes a network to learn
  - topology/architecture
  - host information
  - services running on hosts
  - vulnerabilities
- Probing is allowed by open, low-level network protocols, e.g., pinging.
- Network mapping is automated in tools such as nmap, nessus, Saint, and several others. Such tools can also be used for testing security.

# Denial of service (DOS) attacks

---

- Attacks against availability of resources
- Sometimes accidental
- Sometimes distributed (DDOS), with “zombie” machines
- Easy to automate, and abundantly automated (see TFN, TFN2K, Trinoo, Stacheldraht)
- Difficult to prevent

# DDOS



# Botnets

[from honeynet.org, 2005]

---

- More than 100 botnets seen in 4 months.
- Ranging from 100s to up to 50,000 hosts each.
- Responsible for 100s of DDOS attacks (at least).
- For rent!

# Detecting attacks

---

- Most current intrusion-detection systems detect the current generation of attacks.
- They look for particular signatures. But these will change over time.
- Newer methods and tools try to find the origin of attacks (e.g., by IP traceback for DDOS).
- There are also methods that aim to detect anomalies, not just known attacks.

## Detecting attacks (cont.)

---

- Passive monitors are intrinsically limited.
- A monitor can be effective against an attack only if it is placed on the path of the attack.
  - So a monitor on the periphery of a network won't help much against an insider attack.
- Monitors can be attacked
  - by overloading them (possibly crashing them),
  - by discrediting them (making them generate many false alarms),
  - ...

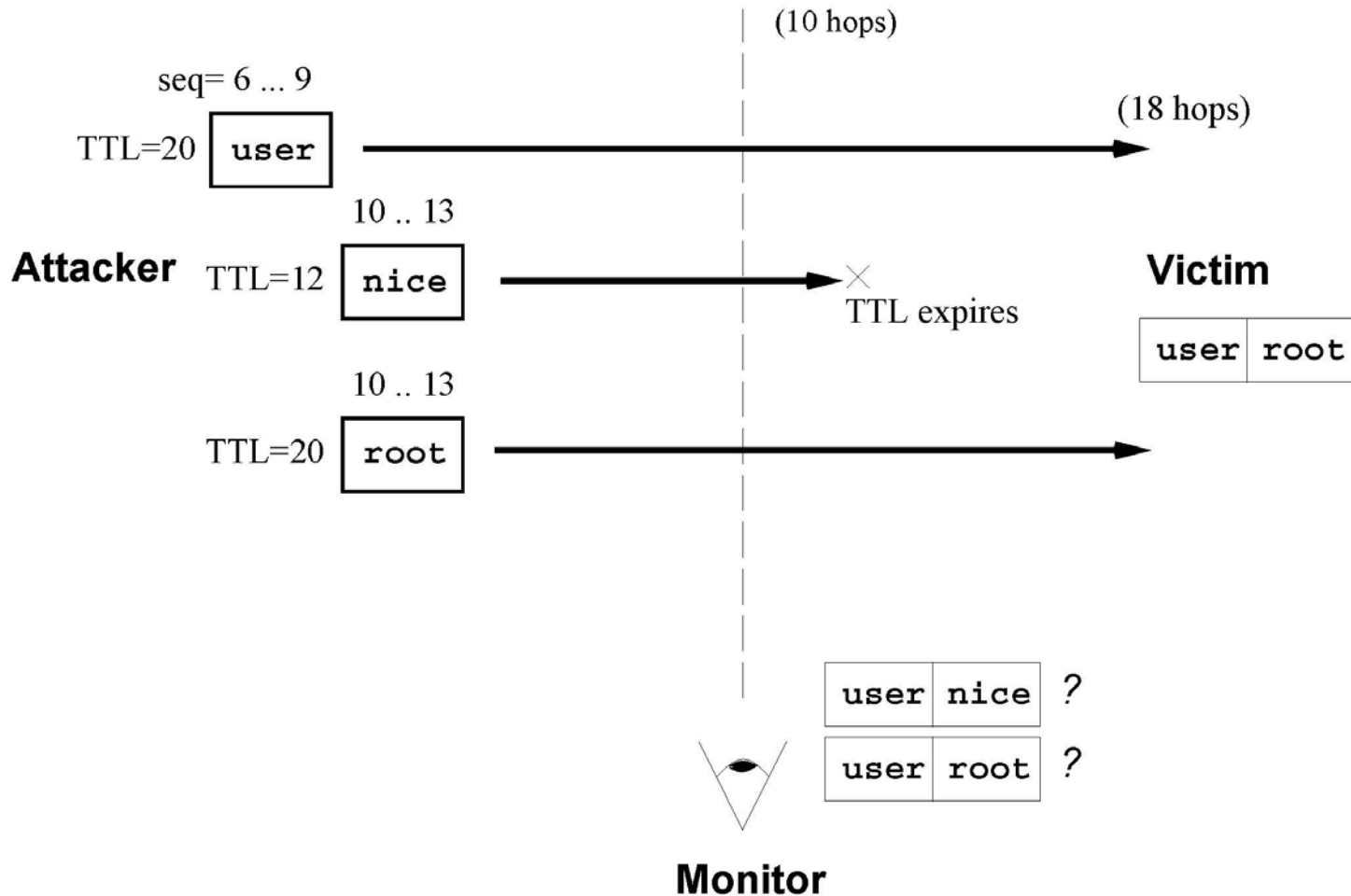
## Detecting attacks (cont.)

---

- A monitor may not see the same as the target of an attack.
  - It may be at a different level in the protocol stack (e.g., it may need to reassemble packets and interpret the results).
  - Even then, it may not do a perfect job (e.g., because it may not predict that a TTL will cause a packet to be dropped).
- Many evasions look anomalous, but so does a lot of legitimate traffic (see Paxson).

# Detecting attacks (cont.)

An example (from Paxson):



# What can ISPs do?

---

- Deploy source address anti-spoof filters.
- Develop security relationships with neighboring ISPs.
- Set up mechanism for handling security complaints.
- Develop traffic-volume monitoring techniques.
  - Look for too much traffic to a particular destination.
  - Learn to look for traffic to that destination at border routers (access routers, peers, exchange points, etc.).

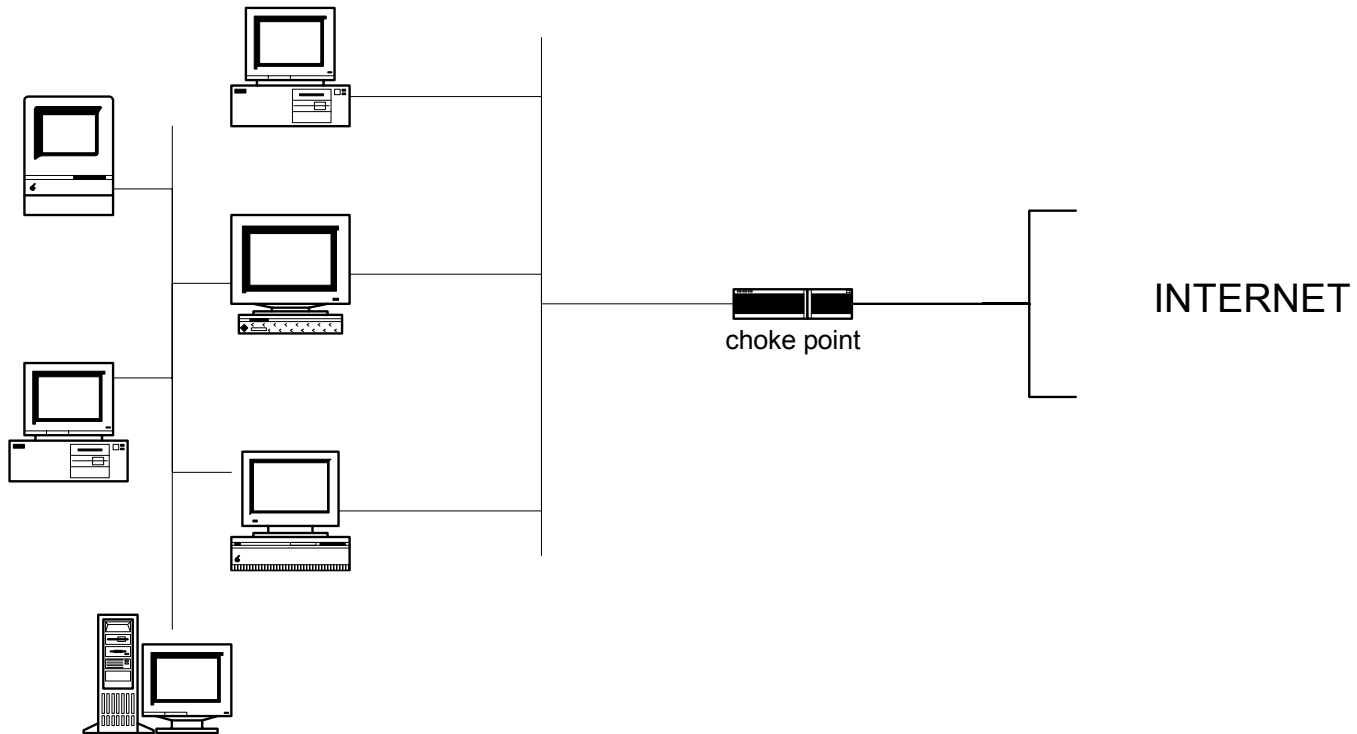
# Firewalls

---

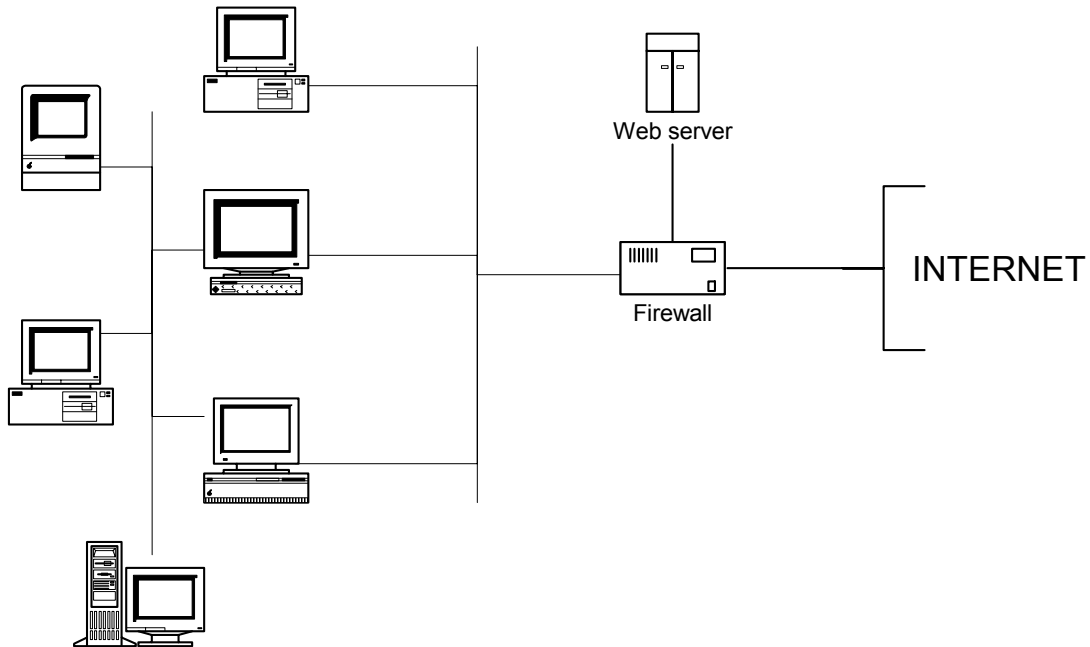
- Protecting a perimeter
  - A place for policy implementation and monitoring
  - Only coarse security (inside/outside)
- Hopefully reliable and testable
- Several kinds:
  - Packet filters (generically look at TCP and IP headers)
  - Application level gateways (proxies for particular services)

# A simple configuration

---



# Web server placement



# Firewall rules

Standard - VPN-1 & FireWall-1 Security Policy

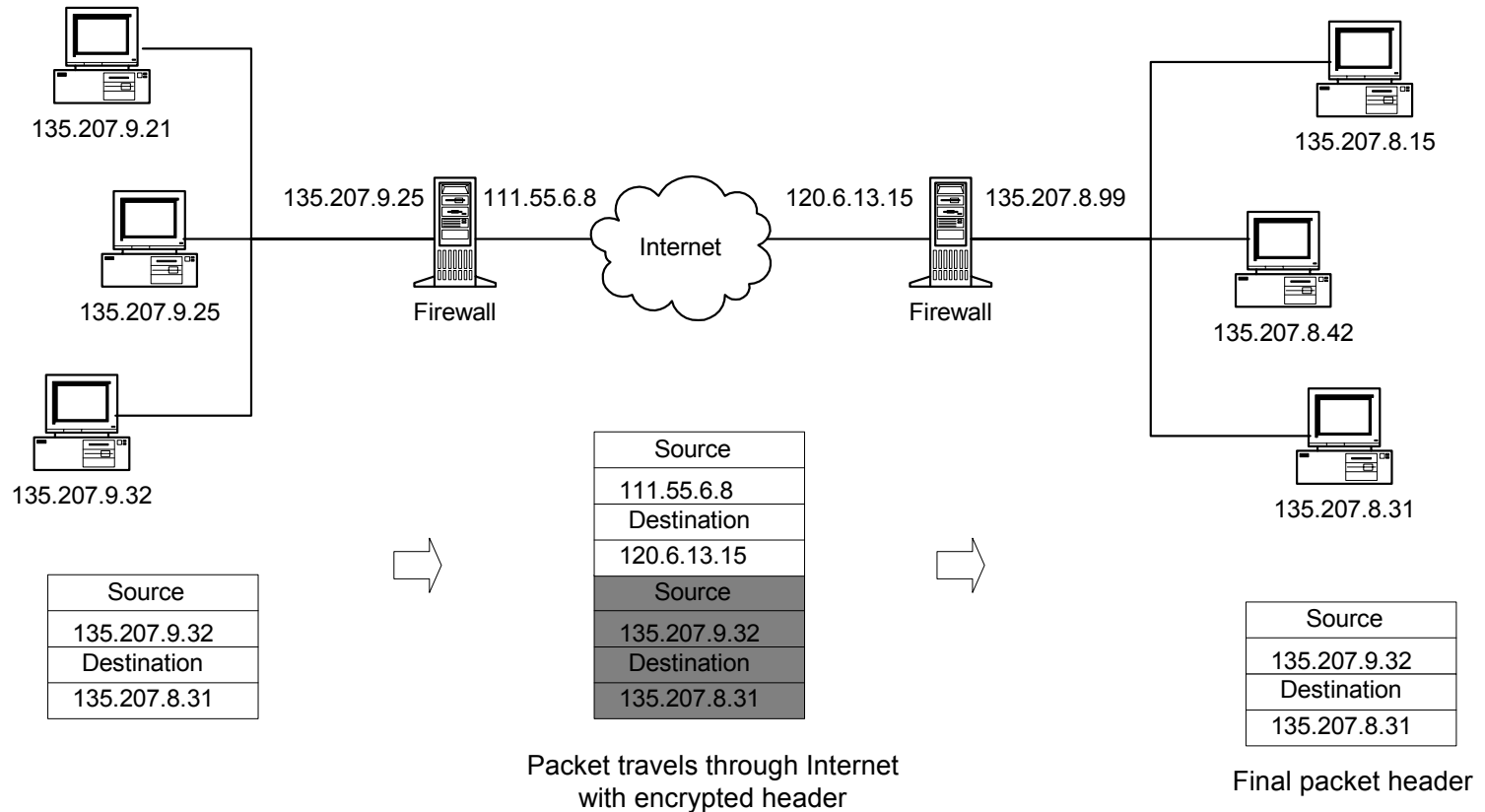
File Edit View Manage Policy Window Help

Security Policy | Address Translation

No.	Source	Destination	Service	Action	Track	Install On	T
1	fw-admin	firewall	FireWall1	accept	Long	Gateways	A
2	Any	firewall	NBT ident	reject		Gateways	A
3	Any	firewall	Any	drop	Long	Gateways	A
4	internal	dns-server	domain-udp	accept		Gateways	A
5	Any	mailserver	smtp	accept	Long	Gateways	A
6	Any	webserver	http	accept	Long	Gateways	A
7	internal	mailserver	pop-3	accept	Long	Gateways	A
8	internal	dmz	Any	accept	Long	Gateways	A
9	dmz	internal	Any	drop	Alert	Gateways	A
10	Any	Any	Any	drop	Long	Gateways	A

Save completed successfully! localhost Read/Write

# VPN (example)



# Fighting spam and other petty abuses

---

# Spam

---

- Spam is unsolicited, junk e-mail.  
(But it is hard to define exactly and universally.)
- Spam is a huge problem
  - for individuals: annoyance
  - for industry: costs in worker attention, infrastructure
  - for ISPs: huge storage costs (65-85% at Hotmail)
  - for the FTC and other agencies: fraud

Spam ruins e-mail and devalues the Internet.

# Why fight spam by technical means

---

- Non-technical measures may not suffice:
  - Laws and ISP rules can be helpful.
  - But they may never stop spam.
- Fighting spam presents interesting challenges.

# Spam as a security problem (and as a topic for us)

---

- Spam can be seen as an availability problem.
- Spammers behave much like other attackers.
- Some spammers seem to rely on other attacks, for enlisting zombies.
- Protection against spam can rely on
  - authentication: where does the email come from?
  - authorization: should the email be read?
- Systems against spam resemble those proposed in other security-related contexts, e.g., for electronic commerce.

# Spam and other petty abuses

---

- Other abuses are similar to spam:
  - Adding lots of URLs to Web indexes (e.g., AltaVista).
  - Creating lots of accounts (e.g., at Yahoo, PayPal).
  - Repackaging Web services (e.g., stock quotes).
  - Connection-depletion attacks on networks.
- Techniques against spam may be applicable against some of those as well.

# Goals

---

- Discourage or eliminate unwanted e-mail.
- Allow public e-mail addresses.
- Allow flow of legitimate e-mail.
- Minimize inconvenience to legitimate senders.

# Proposed techniques

---

- E-mail filtering:
  - by content
  - by sender
  - by collaborative rating
  - using e-mail addresses that get only spam

# Proposed techniques (cont.)

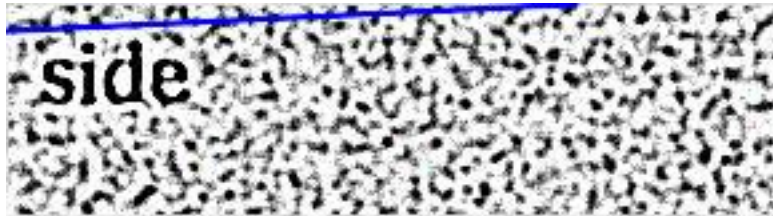
---

- Source-address confirmation
- Ephemeral e-mail addresses  
e.g., abadi-nyt-2lAbki34@cs.ucsc.edu

# Proposed techniques (cont.)

---

- Distinguishing humans from robots via Turing tests (aka CAPTCHAs)  
e.g., by ability at optical character recognition



body

# Proposed techniques (cont.)

---

- Requiring payment, somehow:
  - with money
  - with some computation

# Generic difficulties

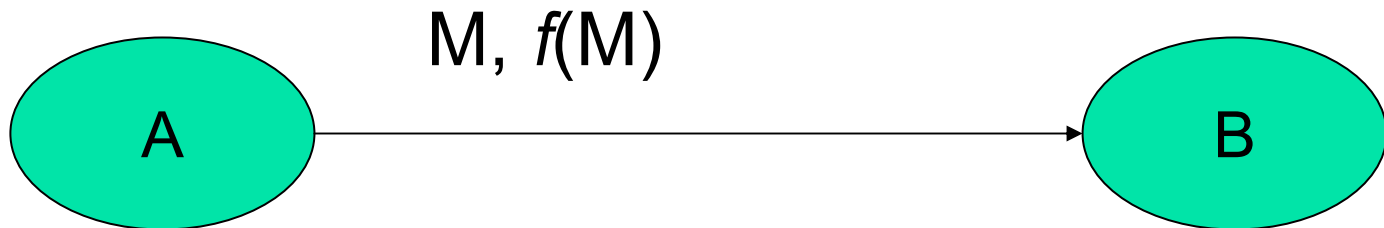
---

- Awkward introductory period
  - Old versions of mail programs
  - Lots of bounced messages
- Complications
  - Distribution lists
  - Whitelists
- An arms race

# Pricing via processing

[Dwork and Naor, and later Back]

- Pick a moderately hard function  $f$ .
  - 10 seconds or 10 minutes per application?
  - No amortization across applications.
- When A sends message  $M$  to B at time  $t$ , A tags  $M$  with  $f(M)$ .



- The tag should be quick for B to verify.

# Puzzles: a hurdle

---

- Historically, this work has relied on CPU-intensive computations.
  - E.g., breaking low-security Fiat-Shamir signatures.
  - E.g., finding hash collisions.
- Machines vary widely in CPU power.
  - 33MHz PDA vs. 2.5GHz PC.
- 25 billion CPU cycles may not discourage a spammer with high-end PCs or servers (who will compute 10 seconds, at leisure).
- 25 billion CPU cycles are a serious obstacle for users with low-end machines (who will wait minutes? hire help? give up?).

# Two recent efforts

---

- **Memory-bound computations**

(for payment)

[Martín Abadi, Mike Burrows, Mark Manasse, Ted Wobber;  
Cynthia Dwork, Andrew Goldberg, and Moni Naor]

- **A Ticket Server**

(similar to a post office and to a mint for electronic commerce)

[Martín Abadi, Andrew Birrell, Mike Burrows, Frank Dabek, Ted Wobber;  
see also e.g. SHRED, by Balachander Krishnamurthy and Ed Blackmond]

(Disclaimers: The generic difficulties still apply. If some of the ideas described are eventually adopted, it may be in the context of different systems.)

# Memory-bound functions

---

- The gap in memory-system performance is much smaller than that in CPU performance.
  - Memory latency is fairly uniform across machines built in the last 5-10 years (within a factor of 2-4).
  - Memory throughput varies a little more (with some expense).
- So memory-bound functions should be more egalitarian, and better for pricing via processing.

# The question

---

- We would like a family of problems that:
  - take many cache misses to solve (say,  $2^{20}$  or  $2^{25}$  cache misses),
  - are fast to set/check (1,000 or 10,000 times faster),
  - can be expressed and answered concisely (in at most a few kilobytes),
  - can be made harder by changing a parameter, and
  - where solving one problem does not help in solving another one.

# An idea

---

- For each e-mail, recipient B requires sender A to invert a well-chosen function  $g$  many times.
  - B specifies the function  $g$  (which may vary).
  - B specifies the values  $g(x)$  for which A should find  $x$ .
- A should build a large table and access it a lot.
  - Inverting  $g$  otherwise may be slow (100s of cycles or much more, each time).
  - The table should fit in a small memory (32MB) but not in a large cache (8MB).
  - Caches should not help much.
- B can easily check A's work by applying  $g$ .

# Flaws of a naïve embodiment

---

- Presenting many challenges at once allows a brute-force CPU-intensive search.
- Adding interaction is impractical.
- The ratio of work done at A and B is not high.
- Each challenge is voluminous.
- Optionally, we may want to avoid communication from B to A.

All of these are addressed in the actual scheme.

# Experiments: Machines

---

	model	processor
server	Dell PowerEdge 2650	Intel Pentium 4
desktop	Compaq DeskPro EN	Intel Pentium 3
laptop	Sony PictureBook C1VN	Transmeta Crusoe
settop	GCT AllWell STB3036N	NS Geode GX1
PDA	Sharp SL-5500	Intel SA-1110

# Machine characteristics

---

	CPU clock	Memory read
server	2.4GHz	0.19 $\mu$ s
desktop	1GHz	0.14 $\mu$ s
laptop	600MHz	0.25 $\mu$ s
settop	233MHz	0.23 $\mu$ s
PDA	206MHz	0.59 $\mu$ s

## Machine characteristics (cont.)

---

- No huge caches (up to 512KB).
- For the PDA, high TLB overhead in measurements of memory read times.
- Different compilers.

# Settings for HashCash

---

- Minting 100 20-bit HashCash tokens (that is, finding 100 independent 20-bit partial collisions in SHA-1).
- Using the HashCash software from the web.

# Timings

---

	HashCash		Memory-bound	
	times	ratios	times	ratios
server	110s	1.0	24s	1.1
desktop	140s	1.3	22s	1.0
laptop	330s	3.0	42s	1.9
settop	1430s	13.0	91s	4.1
PDA	1920s	17.5	100s	4.5

# Summary

---

- We have identified and studied some moderately hard, memory-bound computations.
- They seem to have good properties.
- Their timings are encouraging.
  - The timings correspond reasonably to memory latencies, and do not vary so much across machines.
  - These computations should be egalitarian enough as a basis for pricing via processing.

# Further work

---

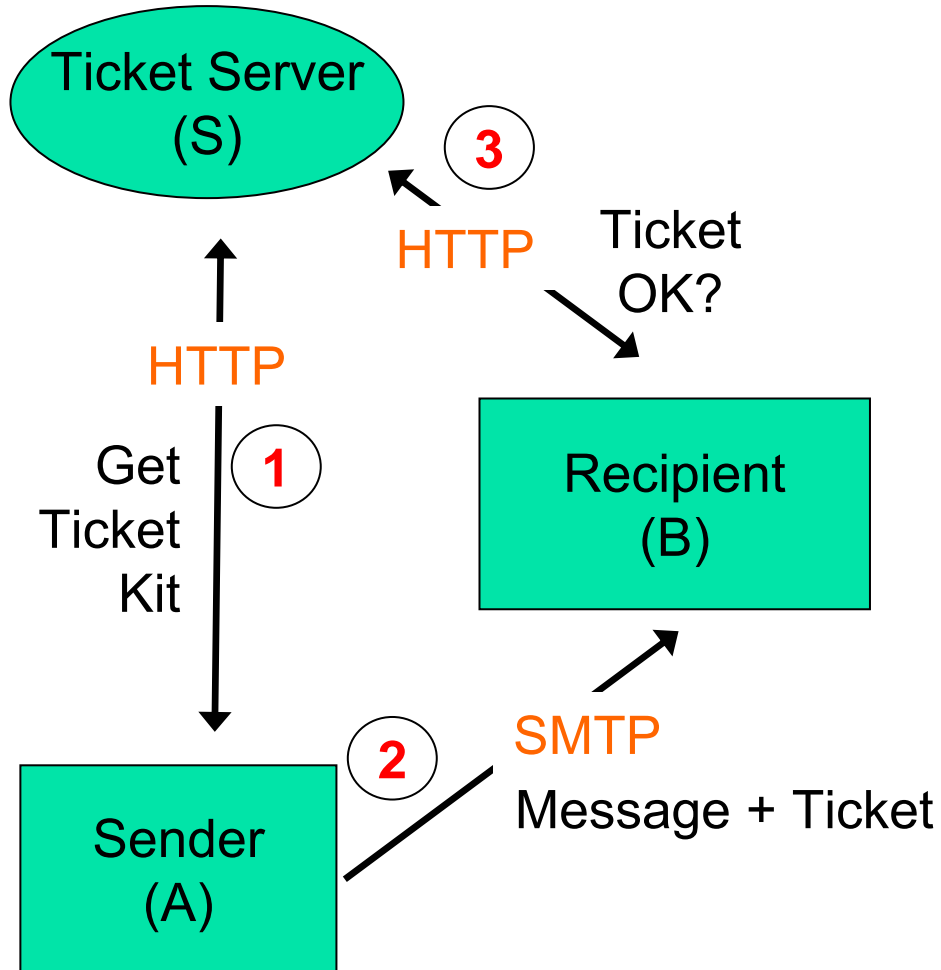
- Many twists and some calculations.
- Other good memory-bound functions  
[see Dwork, Goldberg, and Naor].
- Theory.

# The Ticket Server

---

- Account balances
  - refunds → more expensive tickets
  - out-of-band ticket acquisition
- Ticket acquisition independent of ticket use
  - pre-computation → more expensive tickets
  - useful in other applications
- Centralization – a mixed blessing
  - centralized management/distribution
  - trust: easier to trust known organizations
  - trust: who runs the ticket servers?
  - single point of failure/attack/defense

# Protocol overview



# Further work

---

- Complications
  - protection against message loss
  - whitelists
  - mailing lists
- Implementation
  - appropriate data structures
  - UIs
- Reasoning about the protocol
- Other applications