Sybil Attacks and Reputation Tracking

Ken Birman

Cornell University. CS5410 Fall 2008.

Background for today

- Consider a system like Astrolabe. Node p announces:
  - I've computed the aggregates for the set of leaf nodes to which I belong
  - It turns out that under the rules, I'm one regional contact to use, and my friend node q is the second contact
  - Nobody in our region has seen any signs of intrusion attempts.
- Should we trust any of this?
- Similar issues arise in many kinds of P2P and gossip-based systems

What could go wrong?

- Nodes p and q could be compromised
  - Perhaps they are lying about values other leaf nodes reported to them…
  - … and they could also have miscomputed the aggregates
  - … and they could have deliberately ignored values that they were sent, but felt were "inconvenient" ("oops, I thought that r had failed…")
  - Indeed, could assemble a "fake" snapshot of the region using a mixture of old and new values, and then computed a completely correct aggregate using this distorted and inaccurate raw data

Astrolabe can't tell

- … Even if we wanted to check, we have no easy way to fix Astrolabe to tolerate such attacks
  - We could assume a public key infrastructure and have nodes sign values, but doing so only secures raw data
  - Doesn't address the issue of who is up, who is down, or whether p was using correct, current data
  - And even if p says "the mean was 6.7" and signs this, how can we know if the computation was correct?

- Points to a basic security weakness in P2P settings

Today's topic

- We are given a system that uses a P2P or gossip protocol and does something important. Ask:

  Is there a way to strengthen it so that it will tolerate attackers (and tolerate faults, too)?

  - Ideally, we want our solution to also be a symmetric, P2P or gossip solution
  - We certainly don't want it to cost a fortune
    - For example, in Astrolabe, one could imagine sending raw data instead of aggregates: yes, this would work… but it would be far too costly and in fact would "break the gossip model"
  - And it needs to scale well

… leading to

- Concept of a Sybil attack

- Broadly:
  - Attacker has finite resources
  - Uses a technical trick to amplify them into a huge (virtual) army of zombies
  - These join the P2P system and then subvert it

Who was Sybil?

- Actual woman with a psychiatric problem
  - Termed “multiple personality disorder”
  - Unclear how real this is
- Sybil Attack: using small number of machines to mimic much larger set

Relevance to us?

- Early IPTPS paper suggested that P2P and gossip systems are particularly fragile in face of Sybil attacks
  - Researchers found that if one machine mimics many (successfully), the attackers can isolate healthy ones
  - Particularly serious if a machine has a way to pick its own hashed ID (as occurs in systems where one node inserts itself multiple times into a DHT)
- Having isolated healthy nodes, can create a "virtual" environment in which we manipulate outcome of queries and other actions

Real world scenarios

- Recording Industry of America (RIA) rumored to have used Sybil attacks to disrupt illegal file sharing

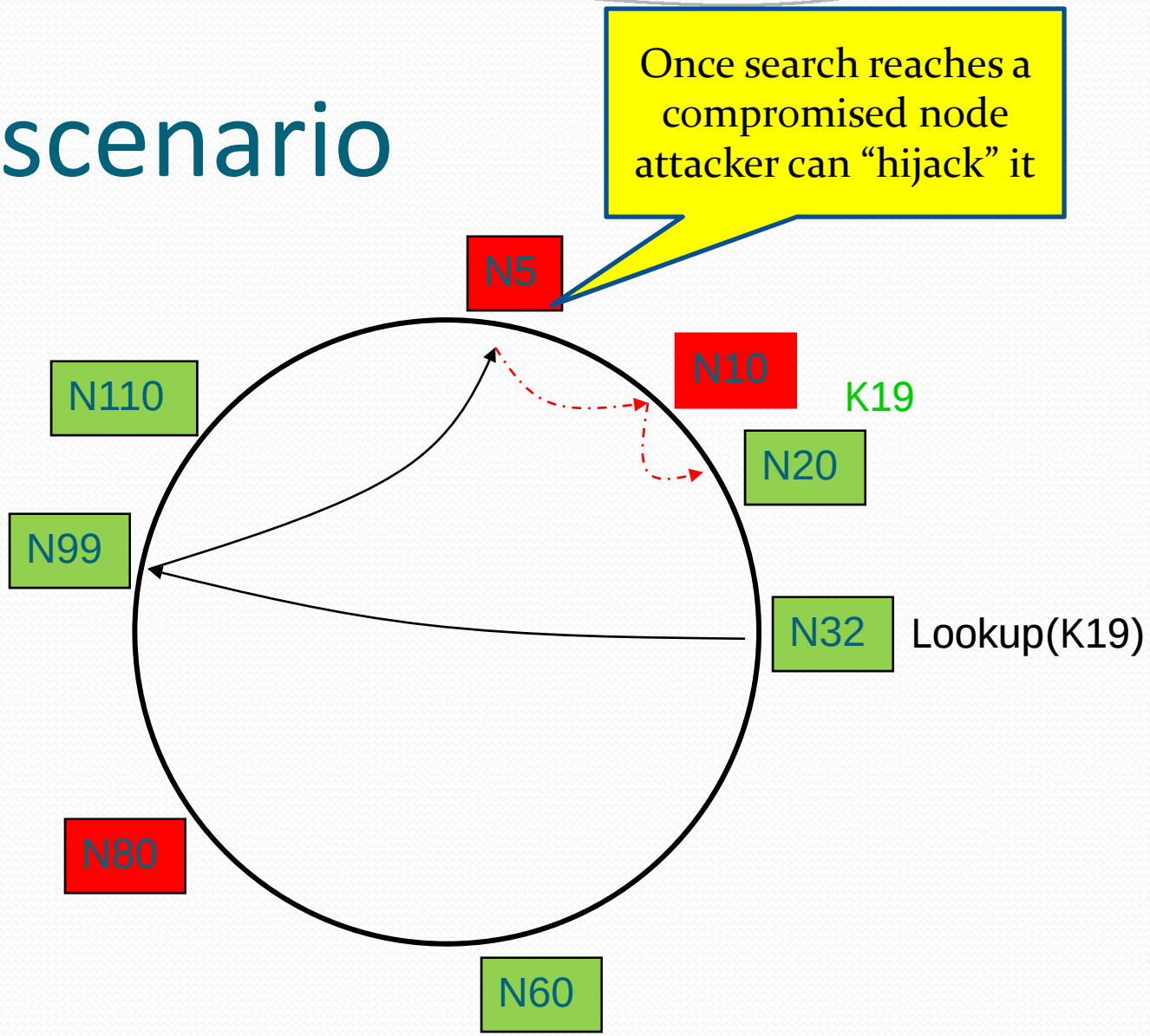
- So-called “Internet Honeypots” lure virus, worms, other malware (like insects to a pot of honey)

- Organizations like the NSA might use Sybil approach to evade onion-routing and other information hiding methods

Elements of a Sybil attack

- In a traditional attack, the intruder takes over some machines, perhaps by gaining root privilages
  - Once on board, intruder can access files and other data managed by the P2P system, maybe even modify them
  - Hence the node runs correct protocol but is controlled by the attacker
- In a Sybil attack, the intruder has similar goals, but seeks a *numerical* advantage.

Chord scenario



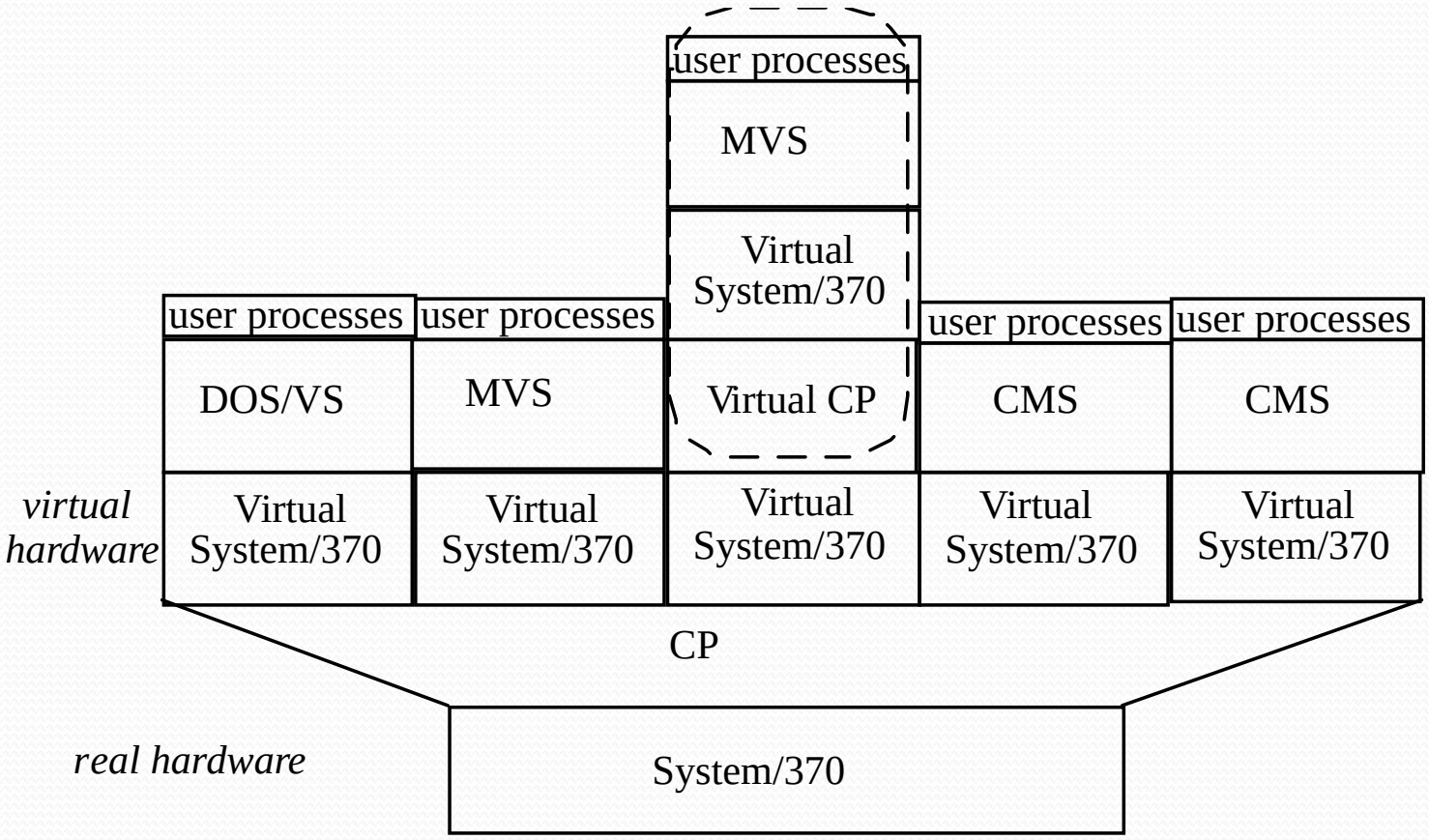
Challenge is numerical…

- In most P2P settings, there are LOTS of healthy clients
- Attack won't work unless the attacker has a huge number of machines at his disposal
  - Even a rich attacker is unlikely to have so much money
- Solution?
  - Attacker amplies his finite number of attack nodes by clever use of a kind of VMM

VMM technology

- Virtual machine technology dates to IBM in 1970's
  - Idea then was to host a clone of an outmoded machine or operating system on a more modern one
  - Very popular… reduced costs of migration
- Died back but then resurfaced during the OS wars between Unix-variants (Linux, FreeBSD, Mac-OS…) and the Windows platforms
  - Goal was to make Linux the obvious choice
  - Want Windows? Just run it in a VMM partition

Example: IBM VM/370

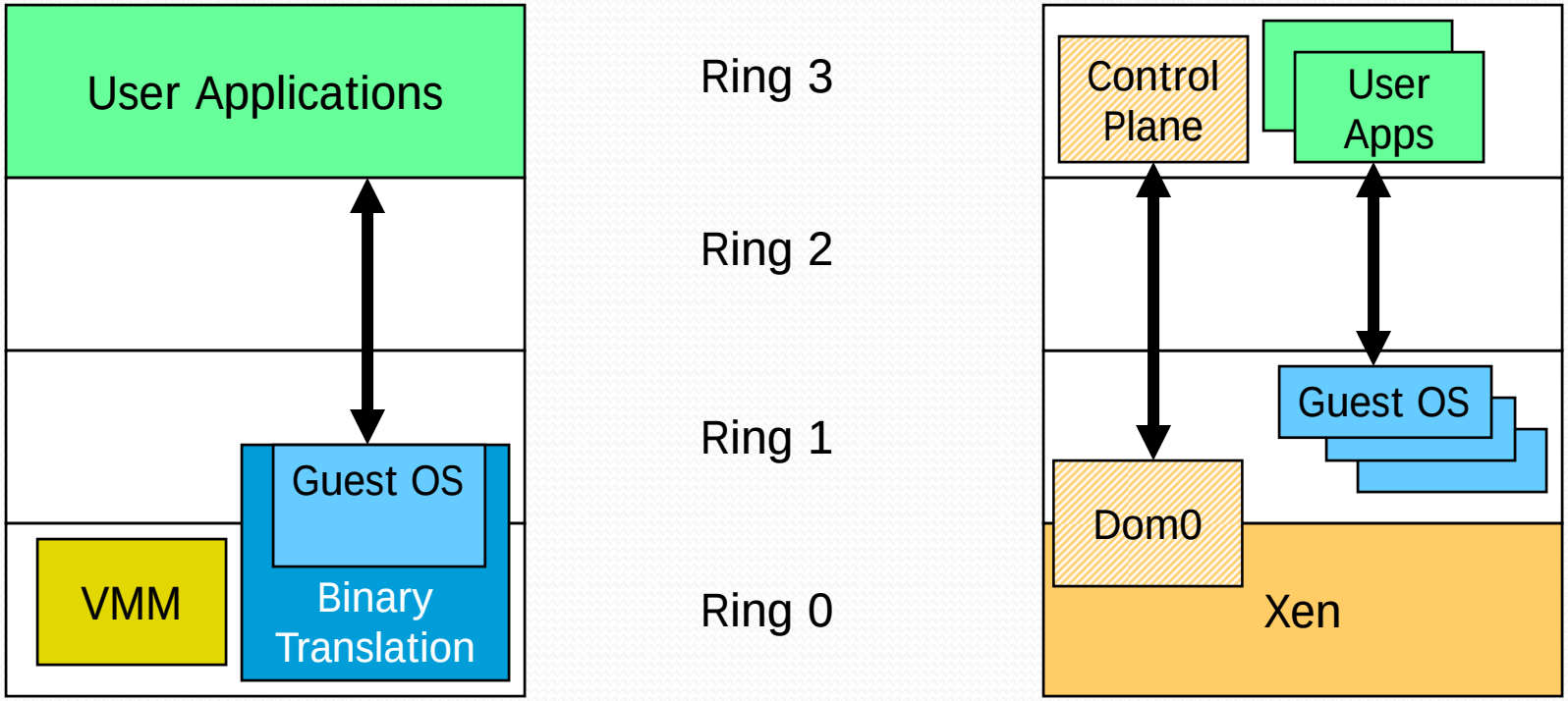


Adapted from Dietel, pp. 606–607

VMM technology took off

- Today VMWare is a huge company
  - Ironically, the actual VMM in widest use is Xen, from XenSource in Cambridge
  - Uses *paravirtualization*
- Main application areas?
  - Some "Windows on Linux"
  - But migration of VMM images has been very popular
  - Leads big corporations to think of thin clients that talk to VMs hosted on cloud computing platforms
  - Term is "consolidation"

Paravirtualization vs. Full Virtualization



Full Virtualization

Paravirtualization

VMMs and Sybil

- If one machine can host multiple VM images… then we have an ideal technology for Sybil attacks
  - Use one powerful machine, or a rack of them
  - Amplify them to look like thousands or hundreds of thousands of machines
  - Each of those machines offers to join, say, eMule
- Similar for honeypots
  - Our system tries to look like thousands of tempting, not very protected Internet nodes

Research issues

- If we plan to run huge numbers of instances of some OS on our VM, there will be a great deal of replication of pages
  - All are running identical code, configurations (or nearly identical)
- Hence want VMM to have a smart memory manager that has just one copy of any given page
  - Research on this has yielded some reasonable solutions
  - Copy-on-write quite successful as a quick hack and by itself gives a dramatic level of scalability

Other kinds of challenges

- One issue relates to IP addresses
  - Traditionally, most organizations have just one or two primary IP domain addresses
  - For example, Cornell has two "homes" that function as NAT boxes. All our machines have the same IP prefix
- This is an issue for the Sybil attacker
  - Systems like eMule have black lists
  - If they realize that one machine is compromised, it would be trivial to exclude others with the same prefix
  - But there may be a solution….

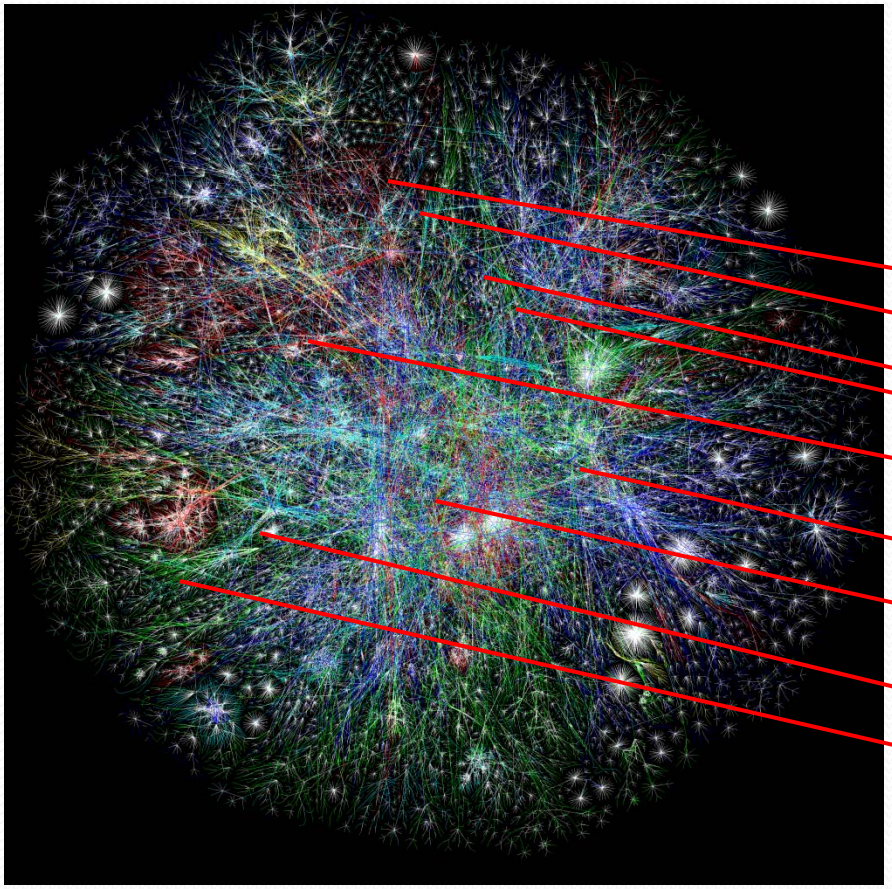
Attacker is the “good guy”

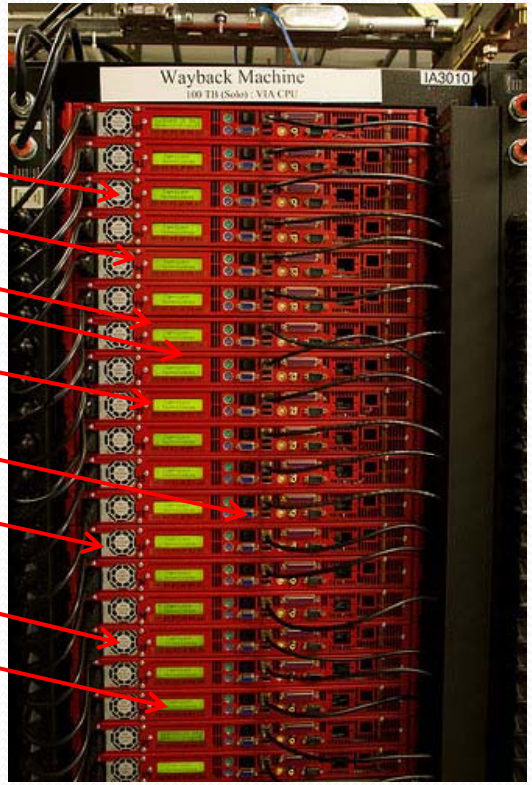
- In our examples, the attacker is doing something legal
- And has a lot of money

- Hence helping him is a legitimate line of business for ISPs

- So ISPs might offer the attacker a way to purchase lots and lots of seemingly random IP addresses
  - They just tunnel the traffic to the attack site

A very multi-homed Sybil attacker





Implications?

- Without "too much" expense, attacker is able to
  - Create a potentially huge number of attack points
  - Situate them all over the network (with a little help from AT&T or Verizon or some other widely diversified ISP)
  - Run whatever he would like on the nodes rather efficiently, gaining a 50x or even 100's scale-up factor!
- And this really works…
  - See, for example, the Honeypot work at UCSD
  - U. Michigan (Brian Ford, Peter Chen) another example

Defending against Sybil attacks

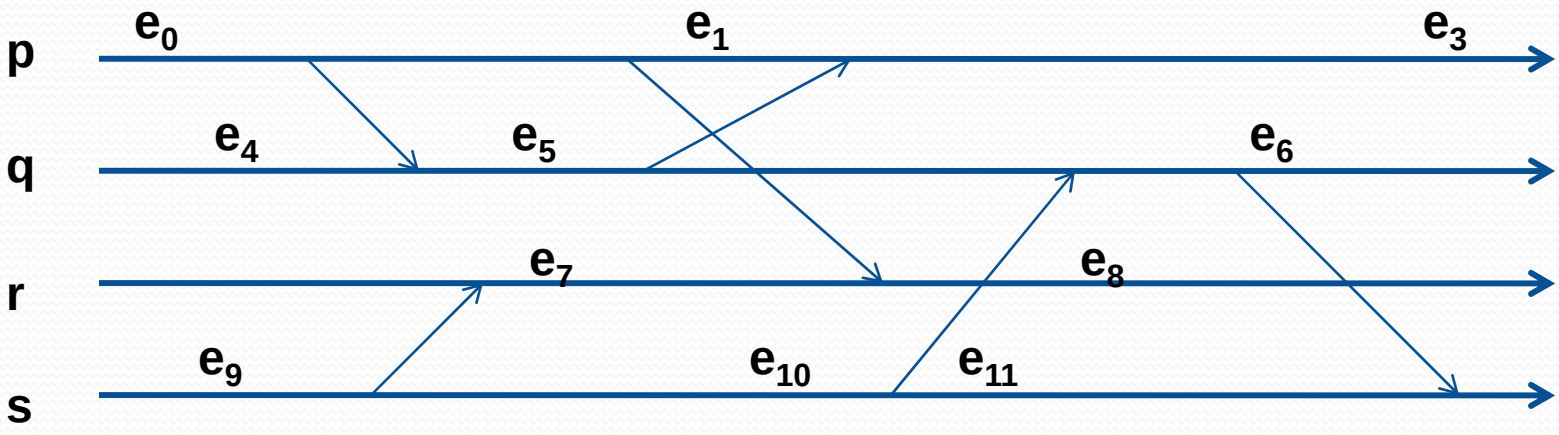
1. Often system maintains a black list
   - If nodes misbehave, add to black list
   - Need a robust way to share it around
   - Then can exclude the faulty nodes from the application
   - Issues? Attacker may try to hijack the black list itself
     - So black list is usually maintained by central service
2. Check joining nodes
   1. Make someone solve a puzzle (proof of human user)
   2. Perhaps require a voucher "from a friend"
3. Finally, some systems continuously track "reputation"

Reputation

- Basic idea:
  - Nodes track behavior of other nodes
  - Goal is to
    - Detect misbehavior
    - Be in a position to prove that it happened
  - Two versions of reputation tracking
    - Some systems assume that the healthy nodes outnumber the misbehaving ones (by a large margin)
      - In these, a majority can agree to shun a minority
    - Other systems want proof of misbehavior

Proof?

- Suppose that we model a system as a time-space diagram, with processes, events, messages



Options

- Node A to all:
  - Node B said “X” and I can prove it
  - Node B said “X” in state S and I can prove it
  - Node B said “X” when it was in state S after I reached state S’ and before I reached state S”

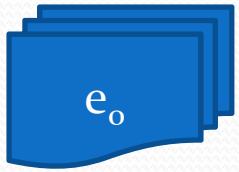
- First two are definitely achievable. Last one is trickier and comes down to cost we will pay
- Collusion attacks are also tricky

Collusion

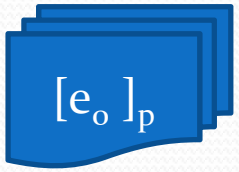
- Occurs when the attack compromises multiple nodes
- With collusion they can talk over their joint story and invent a plausible and mutually consistent one
- They can also share their private keys, gang up on a defenseless honest node, etc

An irrefutable log

- Look at an event sequence: e_0 e_1 e_2
- Suppose that we keep a log of these events



- If I'm shown a log, should I trust it?
  - Are the events legitimate?
  - We can assume public-key cryptography ("PKI")
  - Have the process that performed each event sign for it

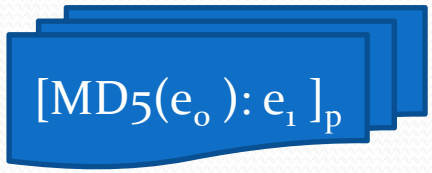


Use of a log?

- It lets a node prove that it was able to reach state S
- Once an honest third party has a copy of the node, the creator can't back out of the state it claimed to reach
- But until a third party looks at the log, logs are local and a dishonest node could have more than one…

An irrefutable log

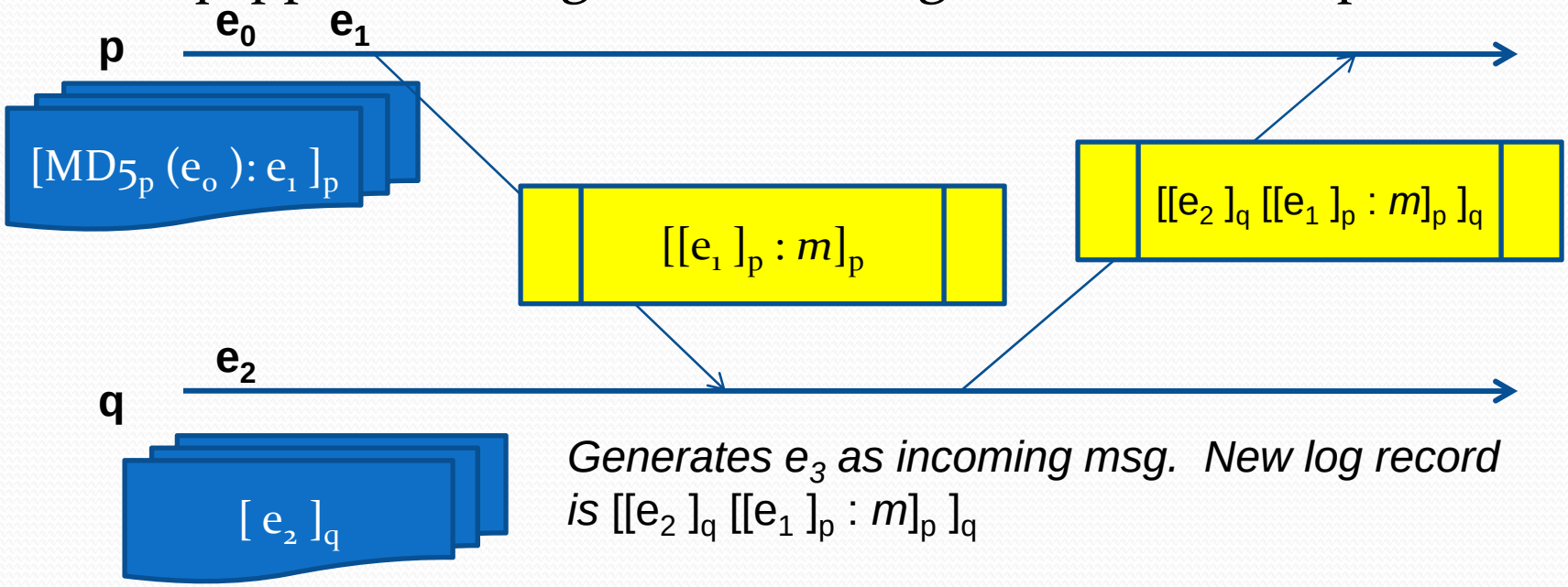
- But can I trust the sequence of events?
  - Each record can include a hash of the prior record



- Doesn't prevent a malicious process from maintaining multiple versions of the local log ("cooked books")
- But any given log has a robust record sequence now

An irrefutable log

- What if p talks to q?
  - p tells q the hash of its last log entry (and signs for it)
  - q appends to log and sends log record back to p



What does this let us prove?

- Node p can prove now that
  - When it was in state S
  - It sent message M to q
  - And node q received M in state S'

- Obviously, until p has that receipt in hand, though, it can't know (much less prove) that M was received

An irrefutable log

- q has freedom to decide when to receive the message from p… but once it accepts the message is compelled to add to its log and send proof back to p
- p can decide when to receive the proof, but then must log it

- Rule: must always log the outcome of the previous exchange before starting the next one

Logs can be audited

- Any third party can
  - Confirm that p's log is a well-formed log for p
  - Compare two logs and, if any disagreement is present, can see who lied
- Thus, given a system, we can (in general) create a consistent snapshot, examine the whole set of logs, and identify all the misbehaving nodes within the set
- Idea used in NightWatch (Haridisan, Van Renesse 07)

Costs?

- Runtime overhead is tolerable
  - Basically, must send extra signed hashes
  - These objects are probably 128 bits long
- Computing them is slow, however
  - Not extreme, but encrypting an MD5 hash isn't cheap
- Auditing a set of logs could be *very* costly
  - Study them to see if they embody a contradiction
  - Could even check that computation was done correctly

Methods of reducing costs

- One idea: don't audit in real-time
  - Run auditor as a background activity
  - Periodically, it collects some logs, verifies them individually, and verifies the cross-linked records too
- Might only check "now and then"
- For fairness: have everyone do some auditing work

- If a problem is discovered, broadcast the bad news with a proof (use gossip: very robust). Everyone checks the proof, then shuns the evil-doer

Limits of auditability

- Underlying assumption?
  - Event information captures everything needed to verify the log contents
- But is this assumption valid?
  - What if event says “process p detected a failure of process q”
    - Could be an excuse used by p for ignoring a message!
  - And we also saw that our message exchange protocol still left p and q some wiggle room (“it showed up late…”)

Apparent need?

- Synchronous network
- Accurate failure detection

- In effect: auditing is as hard as solving consensus

- But if so, FLP tells us that we can never guarantee that auditing will successfully reveal truth

How systems deal with this?

- Many don't: Most P2P systems can be disabled by Sybil attacks
- Some use human-in-the-loop solutions
  - Must prove human is using the system
  - And perhaps central control decides who to allow in
- Auditing is useful, but no panacea

Other similar scenarios

- Think of Astrolabe
  - If "bad data" is relayed, can contaminate the whole system (Amazon had such an issue in August 08)
- Seems like we could address this for leaf data with signature scheme… but what about aggregates
  - If node A tells B that "In region R, least loaded machine at time 10:21.376 was node C with load 5.1"
  - Was A using valid inputs? And was this correct at that specific time?
  - An evil-doer could delay data or detect failures to manipulate the values of aggregates!

Auditable time?

- Only way out of temporal issue is to move towards a state machine execution
  - Every event…
  - … eventually visible to every healthy node
  - … in identical order
  - … even if nodes fail during protocol, or act maliciously
- With this model, a faulty node is still forced to accept events in the agreed upon order

Summary?

- Sybil attacks: remarkably hard to stop
  - With small numbers of nodes: feasible
  - With large numbers: becomes very hard
- Range of options
  - Simple schemes like blacklists
  - Simple forms of reputation ("Jeff said that if I mentioned his name, I might be able to join…")
  - Fancy forms of state tracking and audit