CS514: Intermediate Course in Computer Systems

Lecture 23: March 14, 2003

Consistency issues in failure handling

TCP Streams

CS514

- One week ago we very briefly saw how TCP overcomes failures
- TCP is the workhorse of the Internet
  - This is because TCP is a “good citizen”
  - Until recently, every web operation used its own TCP connection!
- Today: look at one aspect of TCP failure handling in more detail
  - In what sense is TCP more reliable than UDP?

Dealing with failures

CS514

- Packets lost, duplicated, out of order: easy, just use sequence numbers (TCP calls these "segment" numbers)
- Sender or receiver fails, or line breaks:
  - After excessive retransmissions, or
  - After excessive wait for missing data, or
  - After not seeing "keepalives" for too long

  ... ***break the connection and report "end of file"***

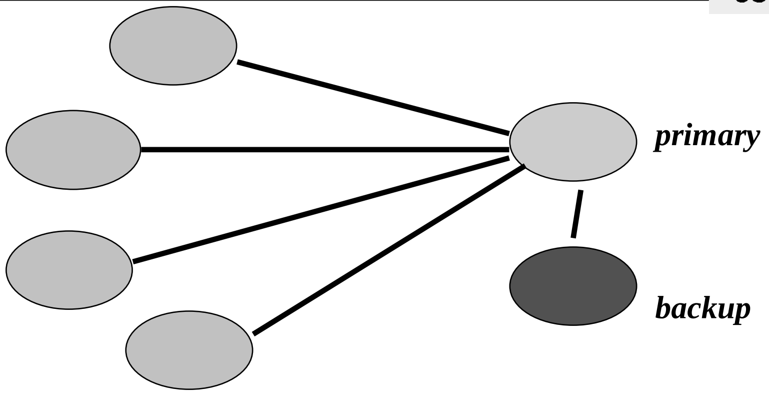
Problems with this approach?

CS514

- Channel can break because of a transient condition!
- Example: overloaded machine, connection that temporarily fails, router crashes and must reboot itself (all are relatively common conditions)
- Systems with many TCP channels: *some may break but others stay connected!*

Inconsistently broken TCP channels

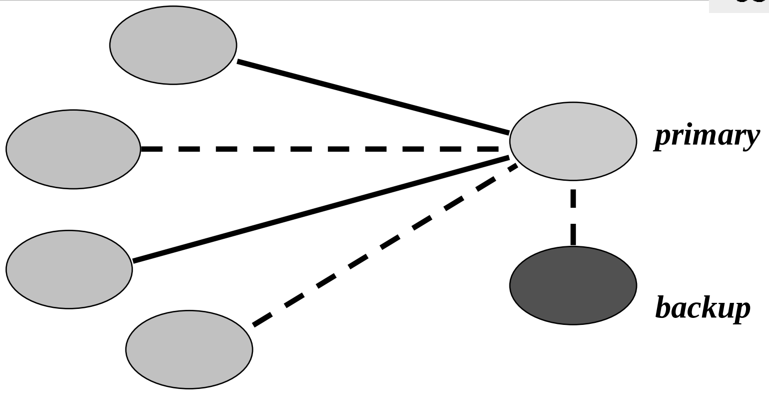
CS514



Clients initially connected to primary, which keeps backup up to date. (For example, in a database system)

Inconsistently broken TCP channels

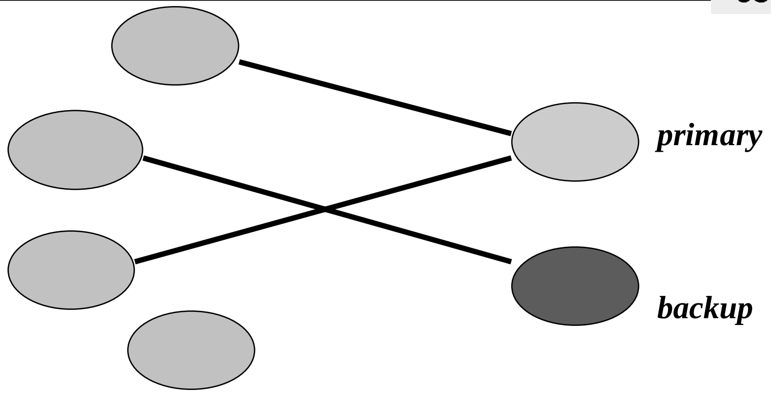
CS514



Transient problem causes some links to break but not all. Backup thinks it is now primary, primary thinks backup is down

Inconsistent outcome!

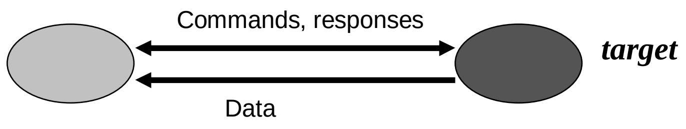
CS514



Some clients still connected to primary, but one has switched to backup and one is completely disconnected from both

A weird case seen in FTP

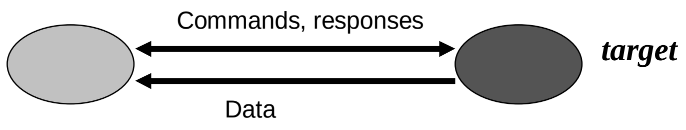
CS514



FTP (file transfer program) uses <u>two</u> TCP connections

A weird case seen in FTP

CS514



Suppose that for about 90 seconds, the target and end-user lose connectivity. One connection will break first!

Aside: Why don't they break at the exact same time?

CS514

- Each TCP connection is sending data, keep-alives
- So why don't they both break at the identical instant in time when problems arise?

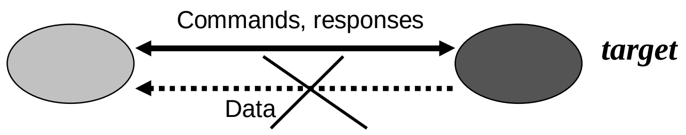
Aside: Why don't they break at the exact same time?

CS514

- Answer: *They aren't synchronized.*
  - If one channel just got some data through and the other has been idle for a while, the first may have 90 seconds or so before it breaks but the second could be much closer to needing a keepalive ping. If that fails, it will disconnect while the first one stays connected

A weird case seen in FTP

CS514



But if communication is now restored… the situation won't fix itself since the connection has already been lost

Why should this matter?

CS514

- Suppose that primary and backup are a service used for air traffic control
  - Service tells controllers which parts of airspace are “available” for routing flights towards airport
  - Primary and backup may try and give *different* controllers access to the *same* airspace! Each thinks it is “in charge” for the system as a whole!
- This is our old split-brain problem but now TCP is causing it

Subtle semantics questions

CS514

- Are the “reliability semantics” of TCP actually different from those of RPC?
  - In both cases, what you “know” is limited to what has been explicitly acknowledged
  - Both can report “failures” when none has occurred
  - Ultimately, TCP and RPC give same guarantees!
- Many systems run RPC over TCP as the “reliable” RPC option. Is this different from normal RPC?

And the answer

CS514

- In a simple sense, they have the same properties
  - Both try to overcome failure
  - Both can mistakenly break a connection
- We think of TCP as the “reliable protocol” but in reality, it is just more reliable than UDP!

What might a better world look like?

CS514

- Clearly, we need a notion of consistency
- What might be reasonable goals?

Possible consistency goals

CS514

- Goal #1:
  - *Connections should only break if one of the endpoints actually fails*
  - *If the network itself is having problems, the application should be warned but the decision to break the connection should be one it takes explicitly*

Is this goal achievable?

CS514

- At the core, becomes a question of how we detect failures
- Several cases
  - If the remote application crashes but the machine stays up, the remote machine can say “the app died”
  - If the machine crashed it just goes silent
  - If the network is hung, the machine *also* goes silent
- How can machine A learn that machine B has crashed?

Vogels: The world-wide failure detector concept

CS514

- Werner Vogels asked
  - Suppose your neighbor stopped collecting his mail
  - Would you call the police and report that he was dead?
- More likely you might knock on the door and see if he is ok. Perhaps would phone him, or his daughter…

Vogels: The world-wide failure detector concept

CS514

- Vogels points out that there are actually many ways to sense health in a modern network
  - We could ping the routers on the path and this would let us judge health of the network, to a degree
  - We could contact some related but different machine, like the email receiver (SMTP) contact for that machine
- He found a dozen or so of these…

Vogels' vision

CS514

- He proposed that we build a new kind of Internet service
- It would
  - Get told when an application (or a machine) starts
  - Monitor its health
  - Tell interested parties if something fails

Killing a sick machine

CS514

- Suppose an apparently sick machine recovers?
  - With Windows XP or Linux this is common
  - E.g. it was hung in a driver or printing error messages…
- If we already reported it dead, we might want to pull the plug and let it reboot! (Otherwise, we lied when we reported it dead, and Werner never lies…)

Consistency issue

CS514

- TCP and RPC could be rewired to talk to this health monitoring service
- Now, if the service avoids bad scenarios, TCP and RPC would behave more sensibly
  - E.g. client would only see a broken connection if the primary is *really* dead, and in this case backup would also know and can take over
  - Slow connection (but no failure notification) could be reported to application…

Turtles all the way down

CS514

- But how can we build our service?
  - Service itself can't be on just one machine or if that dies, it dies!
  - So we need to build a replicated service… how will it track health of its own members?

Membership service

CS514

- Leads to a new core goal
  - We want to build a “membership service”
  - It runs in the Internet
  - It has multiple members of its own
  - Applications can register with it
    - I’m “outlook contacts database” on “fafnir.cs.cornell.edu” and I’m alive!
    - Please monitor “oracle” for me on “…”
- And it does all the work of tracking its own health, monitoring them, reporting events

Consistency goals?

CS514

- Q: What is the simplest way to formalize exactly what consistency goals we seek for our membership service?

Consistency goals?

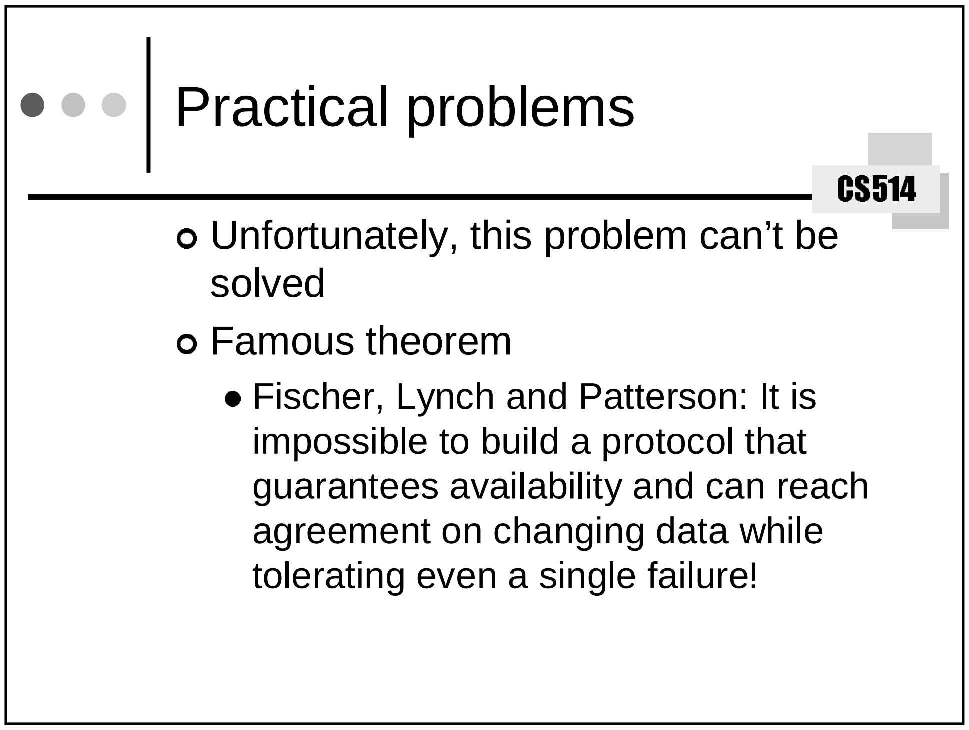
CS514

- Q: What is the simplest way to formalize exactly what consistency goals we seek for our membership service?
- A: It should act like a single server program on a single machine… but one that never fails or becomes unavailable

Implementing our server?

CS514

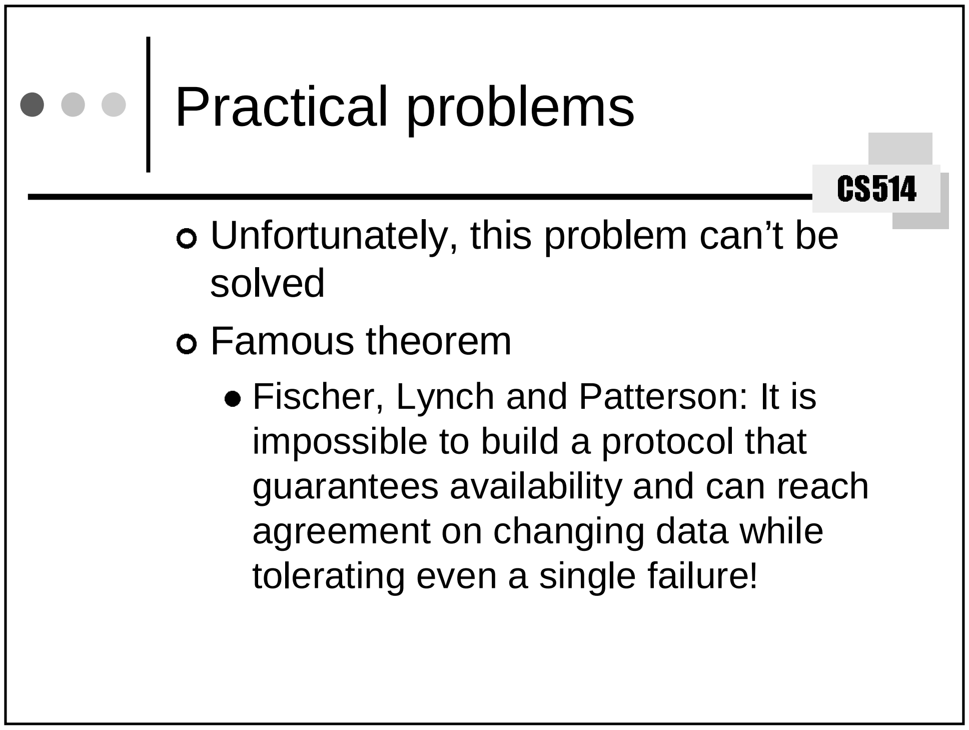
- In the “real world” it needs to have
  - Representatives at lots of places
    - Like the DNS
  - They somehow cooperate to pretend to be just a single program at a single place
  - And they use available monitoring interfaces to watch things
- Then would modify TCP, RPC to use it
  - Not a trivial thing since IETF would probably throw a conniption fit! (Not end-to-end…)



Practical problems

CS514

- Unfortunately, this problem can’t be solved
- Famous theorem
  - Fischer, Lynch and Patterson: It is impossible to build a protocol that guarantees availability and can reach agreement on changing data while tolerating even a single failure!



Practical problems

CS514

- Unfortunately, this problem can’t be solved
- Famous theorem
  - Fischer, Lynch and Patterson: It is impossible to build a protocol that guarantees availability and can reach agreement on changing data while tolerating even a single failure!

Practical problems

CS514

- Unfortunately, this problem can't be solved
- Famous theorem
  - Fischer, Lynch and Patterson: It is impossible to build a protocol that guarantees availability and can reach agreement on changing data while tolerating even a single failure!

Practical problems

CS514

- Unfortunately, this problem can't be solved
- Famous theorem
  - Fischer, Lynch and Patterson: It is impossible to build a protocol that guarantees availability and can reach agreement on changing data while tolerating even a single failure!

So who cares?

CS514

- This particular theorem limits what we can hope to achieve
- But there *is* a way to work around it
  - Basically, we won't guarantee availability
  - Our service will only promise to try to maintain availability
  - Alas it may make mistakes… but
    - If the service says "He's dead, Jim"
    - It simultaneously pulls a phaser from its pocket and fires!
    - "But Bones, he was alive!" "No, he's dead now"

Simplified problem

CS514

- We want to build a service that tracks its own membership
  - It tries to remain available during failures (but sometimes could hang if the failure sequence is awful enough)
  - It uses pings or other tools to monitor its own members
  - And it needs to avoid somehow partitioning into two services if the network itself splits (partitions), forming two disconnected subnets!

We'll return to this in 1 week

CS514

- After the break, we'll
  - Design a solution to this problem
  - Think about how it could solve the TCP and RPC consistency problem
  - Use it to build various forms of reliable multicast

…. And we'll see how such a service was used to replicate the state of the Swiss Stock Exchange!