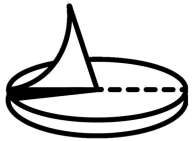


Sundial: Fault-tolerant Clock Synchronization for Datacenters

APNet 2022

Yuliang Li, Gautam Kumar, Hema Hariharan, Hassan Wassel, Peter Hochschild,
Dave Platt, Simon Sabato, Minlan Yu, Nandita Dukkipati, Prashant Chandra, Amin Vahdat

Google

Need for synchronized clocks in datacenters

- Simplify or improve existing applications
  - Distributed databases

Spanner

FaRMv2

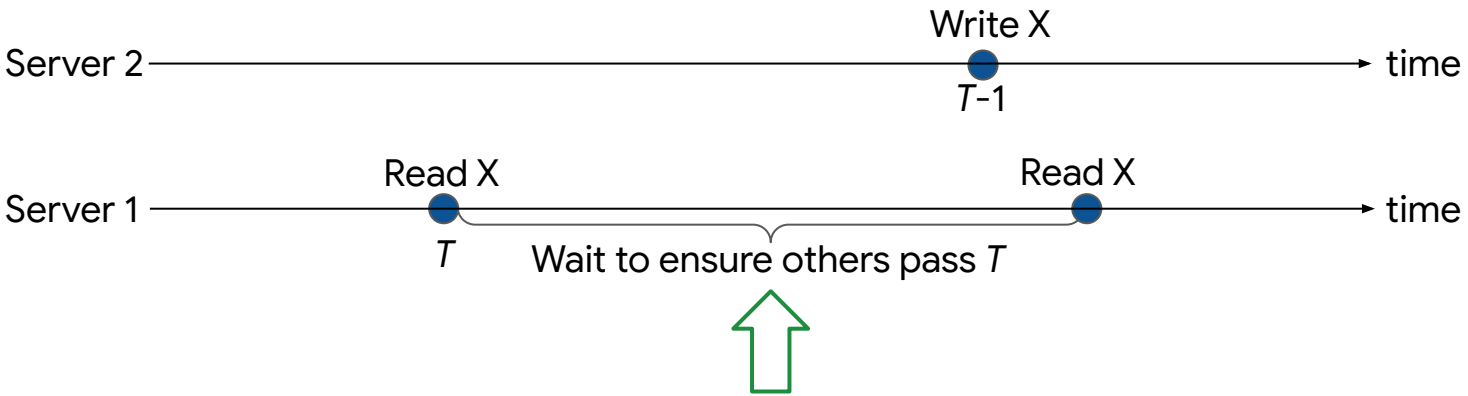
  - Consistent snapshots

- Enable new applications
  - Network telemetry, e.g., per-link loss/latency, network snapshot
  - One-way delay measurement for congestion-control
  - Distributed logging and debugging
  - Event ordering

- And more, if synchronized clocks with tight bound are available

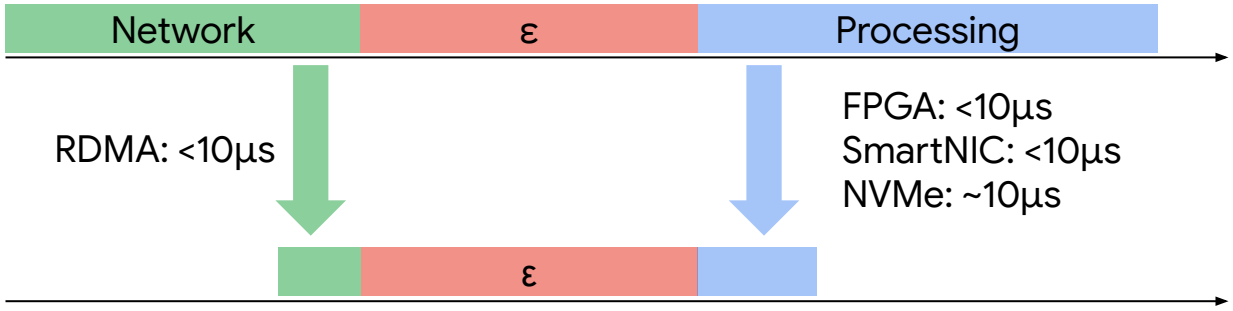
Need for time-uncertainty bound (ε)

Wait: a common op for ordering & consistency



Time-uncertainty bound (ε) decides how much to wait

Need for tighter time-uncertainty bound (ε)



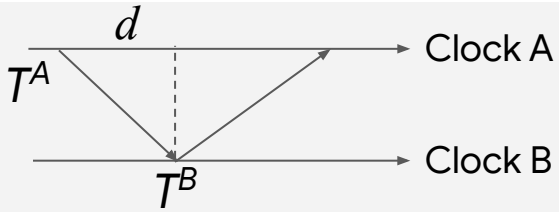
Even 10~20µs ε causes 25% extra median latency*!

Sundial: ~100ns time-uncertainty bound even under failures
2 to 3 orders of magnitude better than existing designs

*FaRMv2 [SIGMOD' 19]

State-of-the-art clock synchronization

Calculate *offset*
Between 2 clocks



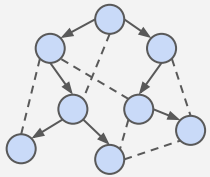
$$offset = T^A + d - T^B$$
$$\approx RTT/2$$

Path of messages

Variable and asymmetric delay ($d \neq RTT/2$):
1. Forward vs. Reverse paths
2. Queuing delay

Sync between neighboring devices
Fixed and symmetric delay ($d=RTT/2$)

Network-wide synchronization

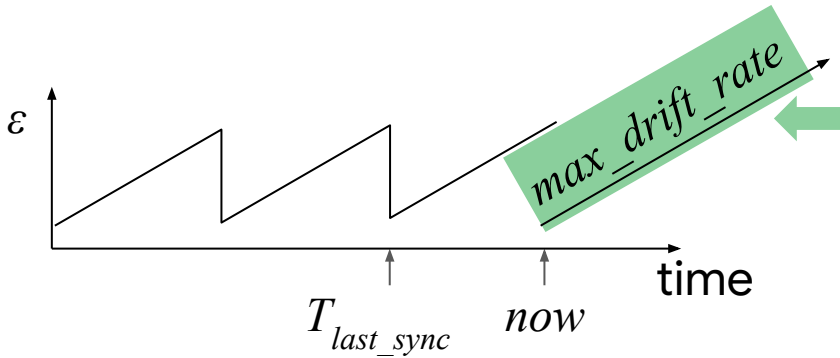


Spanning tree:
Clock values distributed along tree edges

Periodic synchronization

Clocks can drift apart over time, so periodic synchronization is needed

Calculation of time-uncertainty bound ε



Frequency-related failures:
- Cooling, voltage fluctuations

Connectivity failures:
- link/device failure that break the spanning tree

$$\varepsilon = (now - T_{last\_sync}) \times max\_drift\_rate + c$$

Impact of failures on max_drift_rate

- Clocks drift as oscillator frequencies vary with temperature, voltage, etc.
  - E.g., frequency ±100ppm between -40~80 °C from an oscillator specification.
  - Various failures cause frequency variations: cooling failure, fire, voltage fluctuations, etc.
- max_drift_rate is set conservatively in production (200ppm in Google TrueTime)
- Reason: must guarantee **correctness**
  - What if we set it more aggressively? A large number of clock-related errors (application consistency etc.) during cooling failures!

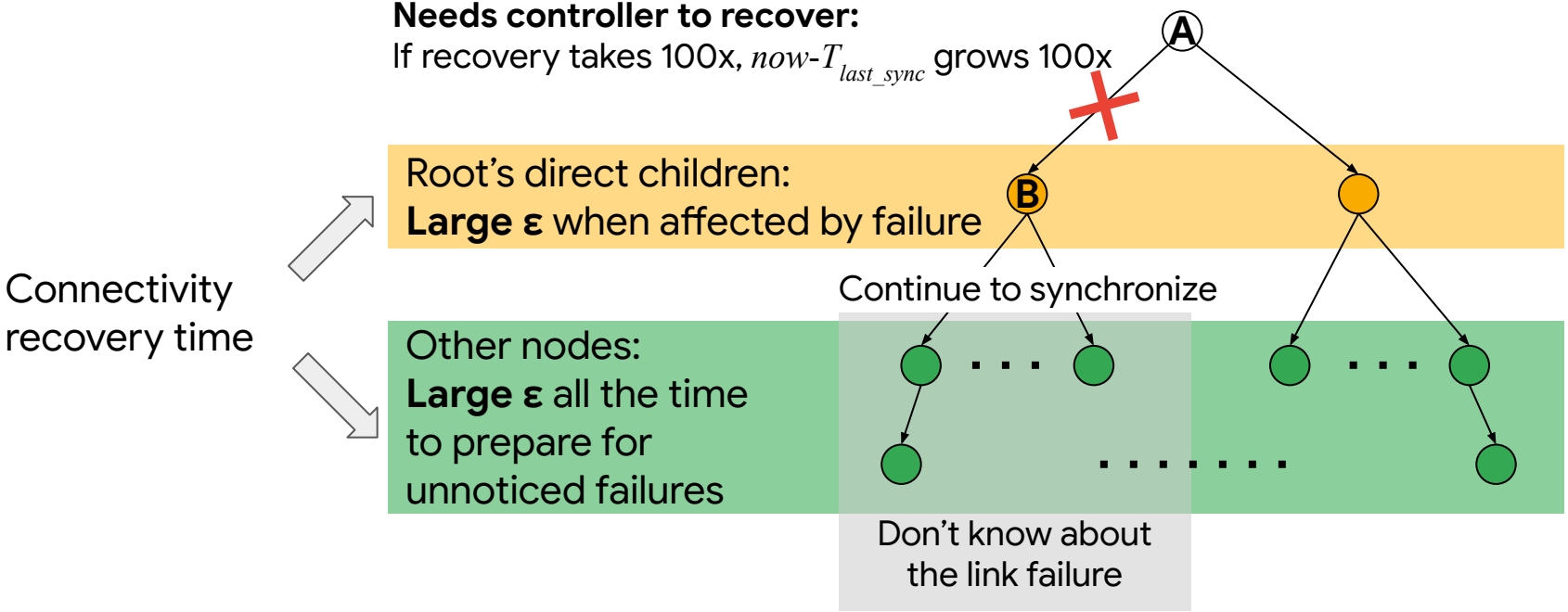
$$\varepsilon = (now - T_{last\_sync}) \times max\_drift\_rate + c$$

< 100ns | **< 500µs** | 200ppm

1. Need very **frequent synchronization**

Impact of failures on $now\text{-}T_{last\_sync}$

Needs controller to recover:
If recovery takes 100x, $now\text{-}T_{last\_sync}$ grows 100x



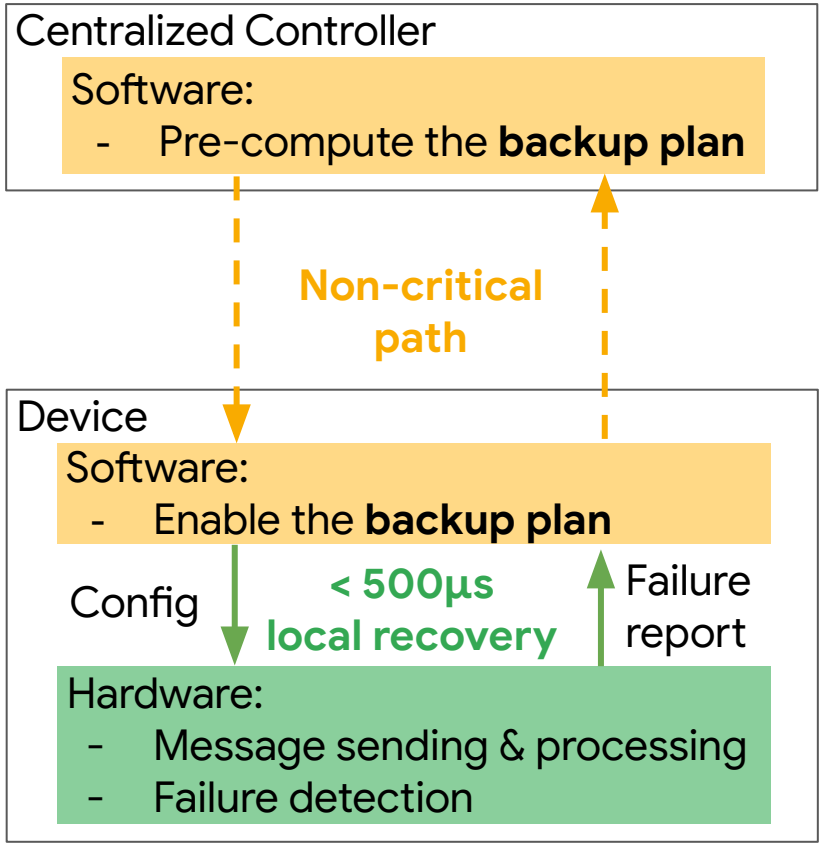
2. Need **fast recovery from connectivity failures**

Sundial design overview

Hardware-software codesign w/ two salient features:

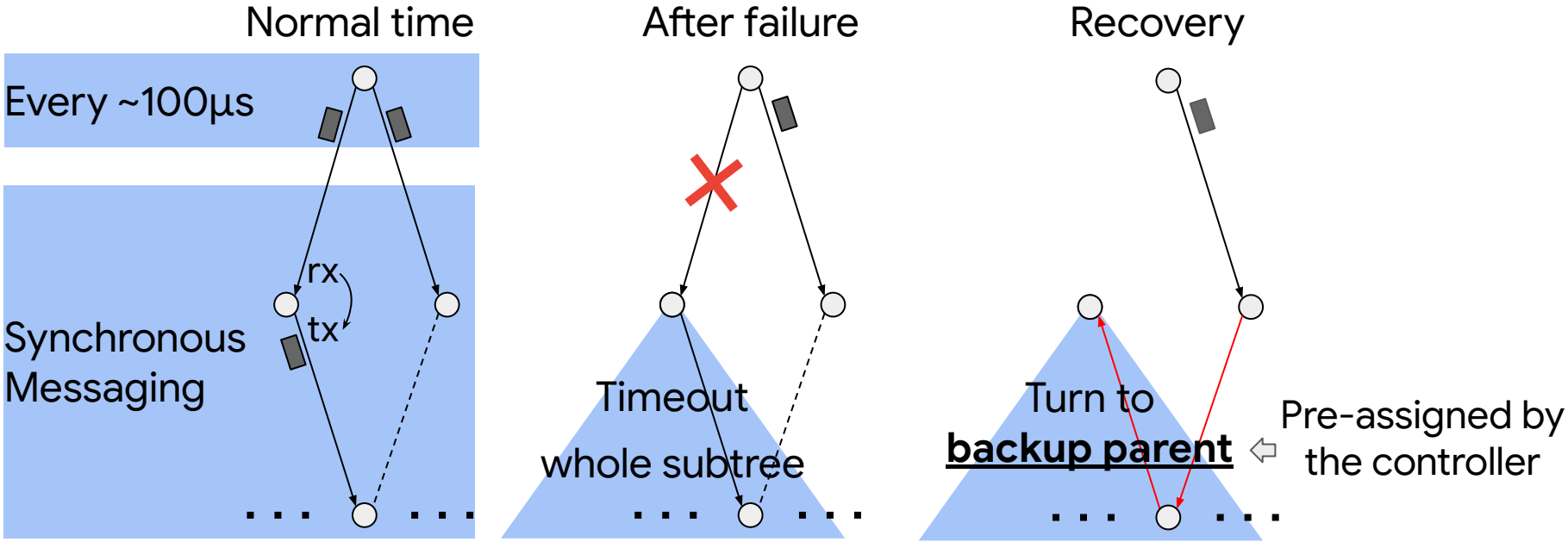
1. Frequent synchronization

2. Fast recovery from connectivity failures



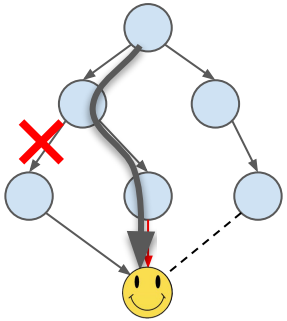
Sundial hardware design

3 key aspects	**Frequent messages** Every ~100µs	**Fast failure detection** Small timeout	**Remote failure detection** Synchronous messaging

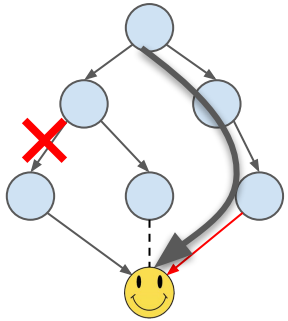


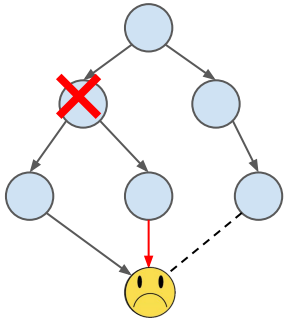
Sundial software design
Controller: pre-compute the backup plan

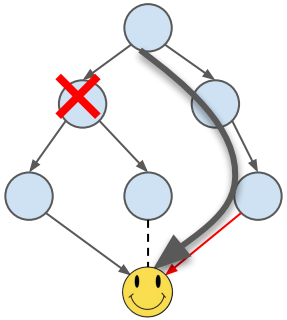
Option 1



Option 2







1 backup parent per device

Multiple options for the backup parent

Device can't distinguish different failures

Generic to different failures

Sundial software design
Controller: pre-compute the **generic** backup plan

- Any single link failure
- Any single device failure
- **Root device failure**
- **Any fault-domain (e.g., rack, pod, power) failure: multiple devices/links go down**

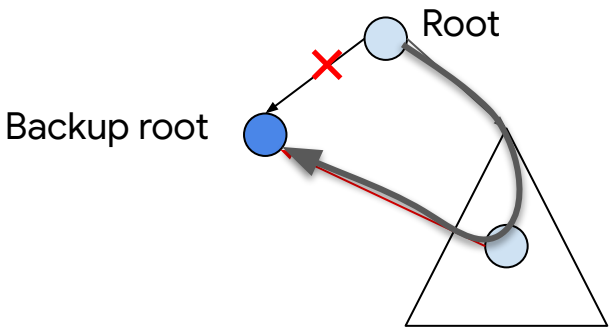
Backup plan
- 1 backup parent per device
- 1 backup root

Backup plan that handles root failure

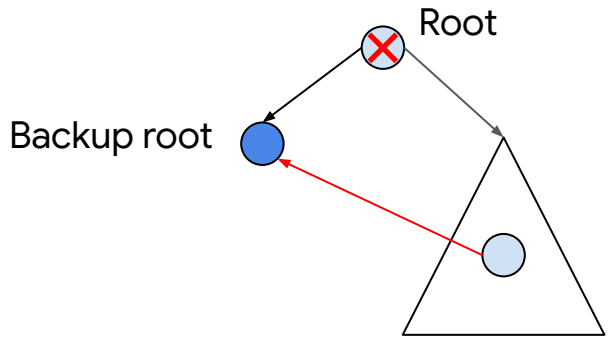
Backup root: elect itself as the new root when root fails (normal device otherwise)

? How to **distinguish root failure** from other failures?

! Get **independent observation** from other nodes



Non-root failure: continue receiving msg



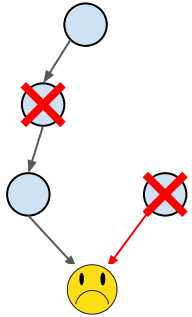
Root failure: no msg

[backup root only] **2nd timeout: elect itself as the new root**

Backup plan that handles fault-domain failures

If one domain failure:

1. Breaks connectivity
2. Takes down backup parent



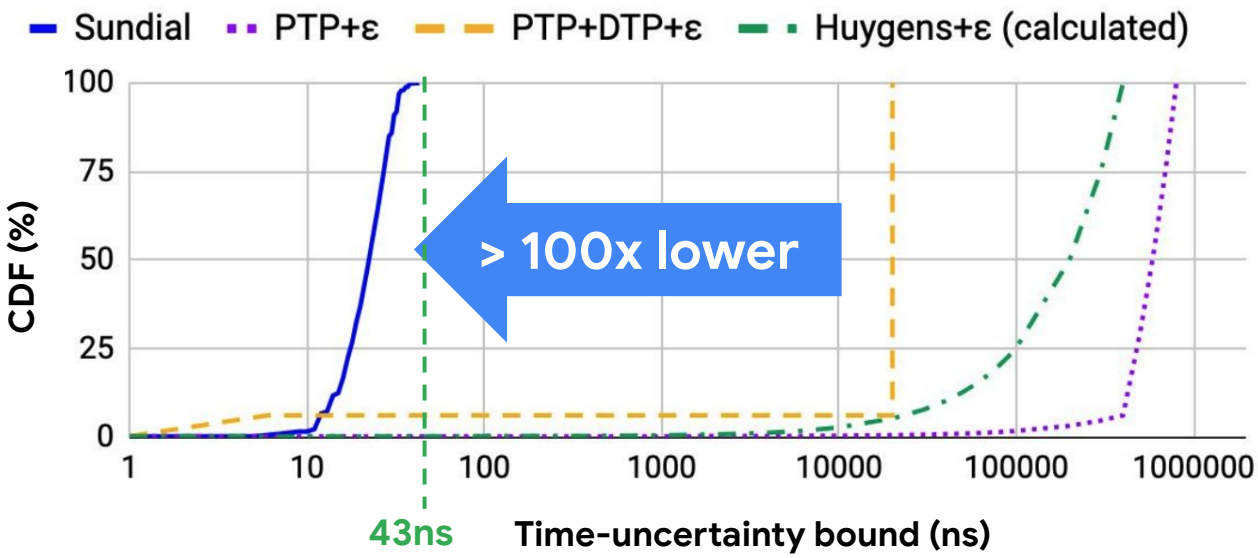
Avoid this case when computing the backup plan

Evaluation

- Testbed: 552 servers, 276 switches
- Compare with state-of-the-art **plus ε**
  - PTP+ε, PTP+DTP+ε, Huygens+ε
- Metrics: **ε**
- Scenarios:
  - Normal time (no failure)
  - Inject failure: link, device, domain

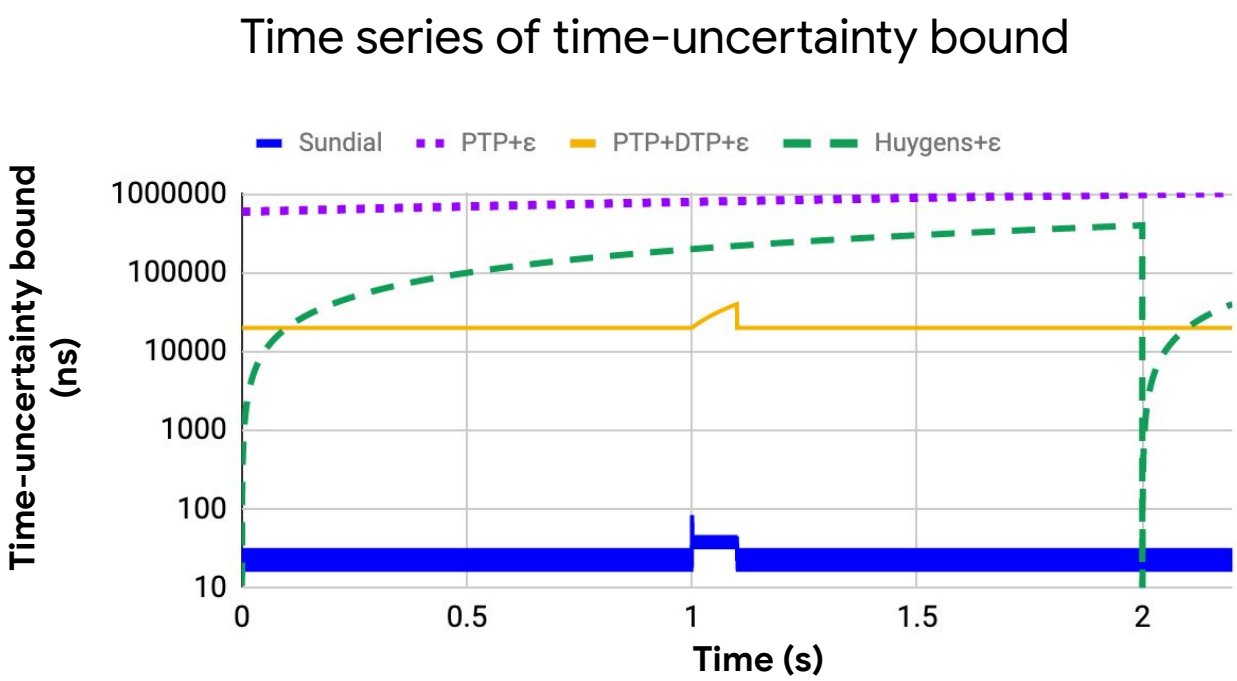
During normal time (w/o failures)

Time-uncertainty bound distribution over all devices



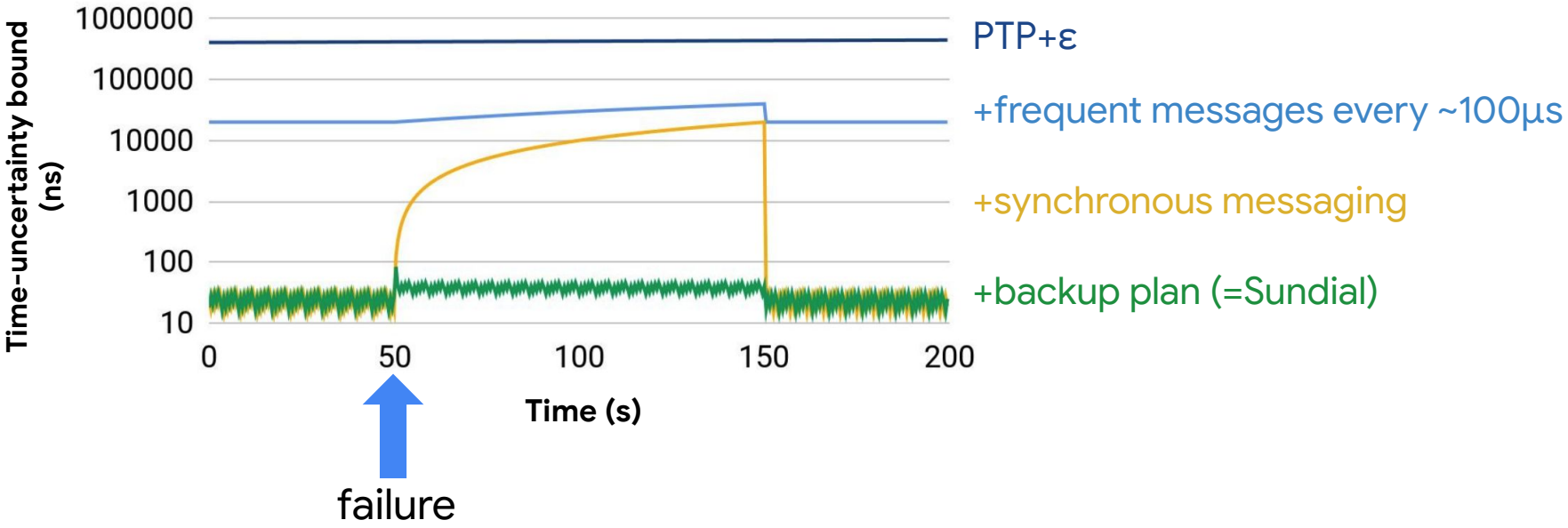
>2 orders of magnitudes lower during normal time

During failures



>2 orders of magnitudes lower during failures

How Sundial's different techniques help

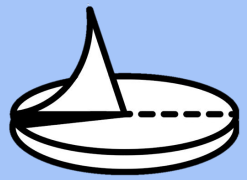


Sundial improves application performance

- Spanner: **3-4x** lower commit-wait latency
- Swift congestion control: with use of one-way-delays, **60%** higher throughput under reverse-path congestion
- Working on more applications using Sundial

Conclusion

- Time-uncertainty bound is the key metric
  - Existing sub-µs solutions fall short because of failures
- Sundial: hardware-software codesign
  - Device hardware: frequent message, synchronous messaging, fast failure detection
  - Device software: fast local recovery based on the backup plan
  - Controller: pre-compute the backup plan generic to different failures



First system: ~100ns time-uncertainty bound

Improvements on real applications

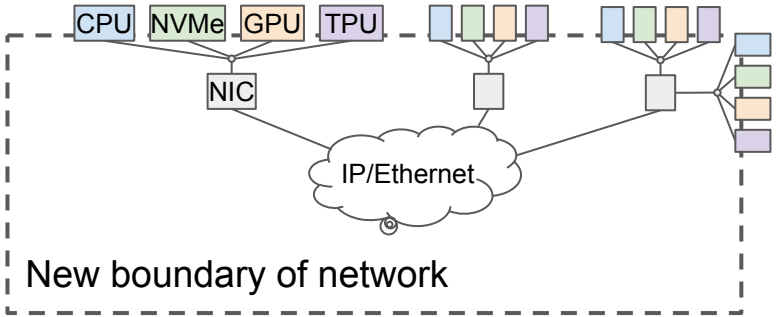
The story is just beginning

Access to NIC clock

- Packet timestamp becomes one of the major usage of NIC clock
- How can NIC efficiently deliver per-packet Tx & Rx timestamps?
  - Stored in metadata/descriptor, or embedded in packet? Cost of parsing, API, evolvability, etc.
- How to design line-rate Tx timestamp?
  - Challenge: Tx timestamp is at the very end of the egress pipeline.
  - Challenge: to get Tx timestamp, sender needs to process separate struct per-packet, doubling processing overhead
- How to get precise timestamp, with 1ns precision?
  - Every nanosecond counts as latency continue to decrease in the future
  - More industries moving to the cloud, with their own precision requirements.
  - Every nanosecond error accumulates hop-by-hop in the synchronization
- How to support more timestamping types/locations?

The boundary of network has expanded

- In ULL environment, in-host latency becomes a significant part.
- For 10s of years, most software didn’t pay attention to in-host latency.
- Face lots of classic problems in this new environment
  - In-host queuing delay: need congestion control
  - Diagnosis: need precise delay measurement
  - Consistency across devices.
- Need precise clock sync and timestamps over more heterogeneous locations



References

- Sundial [OSDI' 2020]
- Aquila [NSDI' 2022]
- International Timing and Sync Forum (ITSF) Webinar: Time & Sync in Data centers, Oct 5, 2021