Time, the Final Frontier…

Time Perimeters for Event Scheduling

Balaji Prabhakar
Stanford University

Time has no independent existence apart from the order of events by which we measure it.

Albert Einstein

The Role of Time in Distributed Systems

Early work of Leslie Lamport (1978)

- "Time, Clocks, and the Ordering of Events in a Distributed System" (11,954 citations on Google Scholar)

Basic question addressed

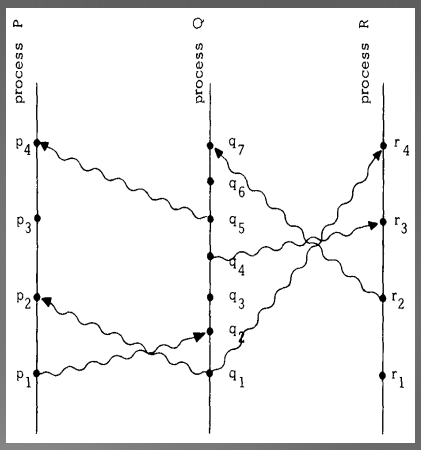
- Given N nodes, each of which has a monotonically increasing clock and are connected by a network, can events ***across*** the N nodes be ***totally ordered***?

Answer

- A **unique partial ordering is possible** using the following rules:
  (1) events at a single node ordered according to the local clock,
  (2) events at different notes ordered by causality and transitivity
- **Non-unique total orderings** are possible which are consistent with the above partial ordering

Conclusion

- A **unique total ordering possible** IFF clocks are synchronized to an **accuracy exceeding inter-event spacings**



The Role of Time in Networking

Circuit switched networks

- Have an intrinsic sense of time because all nodes are tightly synced

Packet switching

- Random delays at in-between nodes hamper clock synchronization

Conclusion

- **Accurately synchronizing** the N nodes ***requires syncing the in-between nodes*** (or have the in-between nodes declare the exact delays they impose)
- That is, end-to-end clock sync is not possible

Our starting point: The above conclusion is not true

- The Huygens algorithm shows how to **accurately sync just a subset of clocks** in a network, under load and without using special priorities

Factors Which Make Clock Sync Hard

Each clock is different

- Clocks have different resonant frequencies
- Clocks behave ***differently*** to the same frequency or offset control signal

Factors Which Make Clock Sync Hard

A clock's behavior varies over time

- Due to temperature
- Due to vibration noise (e.g., cooling fans), etc

To achieve good clock sync

- Need to solve **both** the **jittery network** and the **heterogenous clocks** problems
- **Continuously**

Living in a Clockless World

The difficulty of network clock synchronization at scale (size and distance) has caused distributed systems and network protocols to run under a "clockless assumption"

- In Distributed Systems: use "consensus protocols" to achieve ordering of events
- In Networking: handshakes for coordination, use of round-trip times not one-way delays
- In both areas: a central "coordinator" schedules events across nodes; e.g., snapshotting

Casualty of this approach: real-time control and decision, especially in multi-user settings

- Real-time control needs "deterministic or no-later-than" message delivery times
- Real-time decision needs information which has "freshness guarantees"

The Rise of Time-Sensitive Networks

New use cases have made "time-sensitive networking" a pressing need

Single node/user

- **Autonomous vehicles:** danger ahead, need to press brakes within the next 50 ms!
- **Automated manufacturing:** robot arm needs to move in sync with one or more conveyor belts

Multiple nodes/users

- **Financial trading:** If I spot an opportunity first, my trade needs to be executed first
- **First-person shooter games:** Two players fire at a target, who shot first?

The IEEE TSN collection of standards for bringing deterministic messaging to Ethernet

- Use the Precision Time Protocol (IEEE 802.1AS) to synchronize ***all nodes*** in a network
- IEEE 802.1Qbv is then used to deliver certain L2 packets on a schedule
- Finally, IEEE 802.1Qcc defines management interfaces for TSN administration

Our Approach: Time Perimeters

Synchronize clocks at **just the desired nodes**

- No need to sync all intermediate clocks → enables scaling in size and distance

In fact, sync clocks on a "perimeter" of the network and select internal or external nodes

- A perimeter is a set of nodes that cuts the network, dividing it into an "inside" and an "outside"

Timestamp packets as they pass through the perimeter

- This could be in either direction
- Use the timestamps to make scheduling decisions at internal or external nodes

Uses of Time Perimeters

Time perimeters enable powerful solutions in Distributed Systems and Networking

- Event ordering/scheduling (e.g., databases, distributed ledgers, snapshotting)
  → Lamport's total ordering of events at different nodes can be solved (up to clock fuzz)
- Building deterministic and **jitter-free networks** (e.g, **CloudEx**)
  → Resequencing and Hold/Release buffers can achieve this
- Large-scale **monitoring and control** (e.g., SIMON, **On-Ramp**)
- Take an action in a precise time window (e.g., sell stock for $X only in a specific time window)

Two major enhancements to scheduling enabled by Time Perimeters

- **Scheduling decisions** can be made based on **absolute values as opposed to relative values**
  - In Networking and Systems, scheduling decisions are based on comparisons (shortest, longest, oldest, etc)
- **Scheduling decisions** can be **based on non-local information** (e.g., timestamp taken elsewhere)
  - Typically, scheduling decisions are based on local state variables, not global variables