
André Schiper

Ecole Polytechnique Fédérale de Lausanne (EPFL)
Switzerland



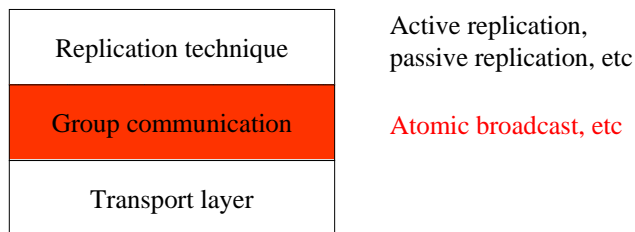
My research

- Fault tolerance in distributed systems, with focus on distributed algorithms for replication
- Key points:
 - Provide clear (and as simple as possible) **specifications** for the problems related to replication
 - Provide **algorithms** for solving these problems
 - Prove the **correctness** of these algorithms (proof of **safety** and proof of **liveness**)
 - **Quantitative performance evaluation**
 - **Test** of the system
- **All** these steps are fundamental to “trust” the system



Group communication (GC)

- GC provide 1-n (or n-m) communication primitives
- Why GC in the context of replication?



- Most of the hard problems of replication are encapsulated in GC



Specification issues

- Specifications define the problem to be solved
- Fundamental !!!
- What does the software implement when there is no clear and easily understandable specification?
- Opinion sometimes expressed:
 - What we want to implement is clear!
 - Why bother with painful specifications: **trust me** 😊
- No! I won't trust you if your are not able to clearly specify the problem that you are trying to solve



Specification issues (2)

Non satisfactory specifications in the context of group communication:

- Specification of the primary partition group membership problem
 - The current specifications do not capture the heart of the problem
 - They are polluted with unnecessary issues
- Specification of the partitionable membership problem
- Specification of dynamic group communication primitives
 - Why are they so different from the specification of static group communication primitives



Algorithmic issues: one example

-
- Two fundamental problems in the context of group communication:
 - Atomic broadcast
 - Group membership
 - **Option 1**: atomic broadcast on top of group membership
 - **Option 2**: group membership on top of atomic broadcast

 - **Option 1** is the standard solution (Isis, Ensemble, etc.)
 - **Option 2** is however a better solution



Correctness

- Tricky issue
- Algorithmic errors vs. coding errors
- Proofs address only algorithmic errors
- When is a proof *really* a proof?
 - Is a “mathematical” proof a satisfactory proof?
 - Some people don’t agree: a proof must be checkable by a computer
- Liveness issues are often ignored (e.g., termination)
 - Why is it so?



Quantitative evaluation

- What do user prefer?
 - A fast “incorrect” solution
 - A slow “correct” solution
- What is an “efficient” fault-tolerant algorithm?
 - An algorithm that is efficient when no crashes?
 - An algorithm that is also efficient with crashes?



Quantitative evaluation (2)

We have defined four faultloads:

- **Normal-steady**: no crashes, no wrong suspicions
- **Crash-transient**: crashes occur at the beginning of the experiment
- **Crash-steady**: one or several crashes occur long before the experiment (parameter = number of crashes); no wrong suspicions
- **Suspicion-steady**: no crashes, but wrong suspicions.
Parameters: frequency and duration of wrong suspicions



Conclusion

- Trust = specifications + proof of correctness + tests + performance figures + etc ?
- Trust = social issue ? (based on reputation)
- Specifications + proof of correctness + tests + performance figures + etc = way to acquire reputation?



Thank you

