An Ecosystem for Verifying Implementations of BFT protocols

Ivana Vukotic
SnT - Interdisciplinary Centre for Security, Reliability and Trust, University of Luxembourg
ivana.vukotic@uni.lu

Vincent Rahli
SnT - Interdisciplinary Centre for Security, Reliability and Trust, University of Luxembourg
vincent.rahli@uni.lu

Marcus Völp
SnT - Interdisciplinary Centre for Security, Reliability and Trust, University of Luxembourg
marcus.voelp@uni.lu

Paulo Esteves-Veríssimo
SnT - Interdisciplinary Centre for Security, Reliability and Trust, University of Luxembourg
paulo.verissimo@uni.lu

1 Research Context

Human society strongly depends on critical information infrastructures such as electrical grids, autonomous vehicles, blockchain applications, etc. Because of the complexity of these systems, proving that they operate in a correct and timely fashion is very hard to achieve. If we add on top of that an increasing number of sophisticated attacks (e.g. Stuxnet), securing correct behavior of these systems is even harder. Byzantine fault-tolerance state machine replication (BFT-SMR) is a technique that enables correct functioning of a system even when some parts of the system are not working correctly. BFT-SMR achieves this by masking the behavior of possibly faulty replicas behind the behavior of enough healthy replicas. Classical BFT protocols require $3f + 1$ replicas to mask the behavior of a minority of up to f faulty replicas. One major issue with these protocols is that they are very complex and therefore hard to get right. Another issue is that most of these protocols come without a formal specification, and some of them even without an implementation [18].

2 Research Goals and Achievements

To overcome these issues, we are developing an ecosystem of formal tools for verifying implementations of BFT protocols. In addition, this ecosystem will allow us to formally explore the breadth of possibilities for designing such protocols. As part of that endeavor, we have already built a generalized and extensible framework called Velisarios [29] for implementing and reasoning about the safety of BFT protocols. Our framework provides, among other things, a model that captures the idea of arbitrary/Byzantine faults; a collection of standard assumptions to reason about systems with faulty components; proof tactics that capture common reasoning patterns; as well as a general library of distributed knowledge. All these parts can be reused to reason about any BFT protocol. For example, most BFT protocols share the same high-level structure (they essentially disseminate knowledge and vote on the knowledge they gathered), which we capture in our knowledge theory. As a case study, we verified the agreement property of PBFT [12, 11], the reference protocol in the area. We chose PBFT because a significant number of protocols are based on it such as [21, 32, 8, 7, 19], to cite only a few. Therefore, a bug found in PBFT would probably result in the existence of bugs in some of those protocols. Verifying the correctness of PBFT is especially important because many variants of that protocol are now being developed and adopted in blockchain technology [16, 22, 6, 5, 27, 26, 3, 2, 1, 4]. Besides the normal-case operation, our PBFT implementation also includes reasoning about garbage collection, view changes and request batching, which are extremely important from a practical point of view. We extracted our verified PBFT implementation to OCaml, and showed that it performs well enough compared to the state of the art BFT-SMaRt [8] library.[1]

Velisarios extends the Logic of Events (LoE), as used in our previous EventML framework [9, 10, 28], in order to reason not only about crash faults, but also about arbitrary faults. In LoE, an event is an abstract entity that corresponds either (1) to the handling of a received message, or (2) to some arbitrary activity about which no information is provided. To prove properties about distributed systems, one only reasons about processes that have a correct behavior, i.e., about events that were triggered by some message. Processes react to the messages that triggered the events happening at their locations one at a time, by transitioning through their states and creating messages to send out, which in turn might trigger other events. When proving a property about a distributed system, one has to reason about its possible runs, which are sometimes modeled as execution traces [30], and which are captured in LoE using *event orderings*. An *event ordering* is an abstract representation of a run of a distributed system; it provides a formal definition of a *message sequence diagram* as used by system designers. As opposed to [30], a trace here is not just one sequence of events but instead can be seen as a collection of local traces (one local trace per sequential process), where a local trace is a collection of events all happening at the same location and ordered in time, and such that some events of different local traces are causally ordered. Some runs/event orderings are not possible and therefore excluded through our assumptions (see [29] for more details). For example, one of our assumptions excludes event orderings where more than f out of n nodes could be faulty.

Because many distributed systems work by exchanging messages to gain knowledge about the state of other participants in order to make progress consensually, our framework includes a library for reasoning about the way participants learn and disseminate information. In the presence

[1] BFT-SMaRt uses MAC vectors while our PBFT implementation uses digital signatures. Therefore, BFT-SMaRt is about one order of magnitude faster than our implementation. However, our implementation is only twice as slow as BFT-SMaRt when replacing digital signatures by MACs, without adapting the rest of the protocol.

of faulty nodes, one has to ensure that this knowledge is reliable. BFT-SMR protocols provide such guarantees using certificates, which ensure the existence of at least one correct node, which has reliable knowledge. Our framework includes abstractions and general lemmas for tracing back such reliable information.

As mentioned above, using Velisarios, we verified the agreement property of our implementation of PBFT. To prove this property, we have to prove that in any event ordering, if two outputs are sent by correct replicas for the same timestamp/client pair, then they have to contain the same reply. We essentially proved this by induction on the causal order of events, and by tracing back the two outputs to conflicting inputs and local states.

As shown in Fig. 1, ours is not the first framework for implementing and reasoning about distributed systems. However, to the best of our knowledge, Velisarios is the first theorem prover based framework for verifying the correctness of implementations of asynchronous BFT-SMR protocols. Velisarios is implemented within the Coq theorem prover [15], because when it comes to proving correctness of a system, theorem provers provide the highest guarantees known to mankind today.

3 Lines of Research for my PhD

Hybrid protocols. BFT-SMR is an extremely expensive technique. To enforce independence of failures between replicas one has to ensure that replicas have different: operating systems; implementations of the service; and administrators. On top of that, each replica should be stored in a different location. Hybrid protocols, such as MinBFT [32], significantly reduce these costs by reducing the number of replicas from $3f+1$ to $2f+1$. This is possible, because hybrid protocols assume the existence of trusted-trustworthy components, i.e., components that can only fail by crashing, and which otherwise always deliver correct results. As future work, we would like to extend our current framework to reason about systems where replicas consist of multiple components that can have different failure assumptions. We would like to formally study the classes of operations and interfaces between such trusted-trustworthy components and the main payload system to preserve their safety properties. Moreover, as future work we will study how the trusted-trustworthy components presented in the literature differ from each other and how each of them influences the design of distributed protocols.

Rejuvenation. In the core PBFT protocol, when the primary is suspected to be faulty, a new primary is elected, but the former primary is kept within the system. This is not practical because in case the former primary is faulty, it should be first repaired (i.e., brought back to a pristine state) before it is allowed to participate again in the system. Castro showed how to extend PBFT with a proactive recovery mechanism [13]. In the future, we plan to add support for reasoning about state transfer and rejuvenation to our ecosystem.

4 Further Lines of Research

Liveness/Timeliness. Although, proving safety of a distributed protocol (such as agreement or linearizability) is important, it is not enough. It might happen that the protocol does not run at all. Moreover, in order to apply BFT-SMR protocols to real-time critical infrastructures, we should show that such a replicated system will not only *eventually* reply, but that it will reply in a timely fashion. We leave reasoning about liveness/timeliness for future work.

Bridging the gap. As suggested in Fig. 1, there is still a gap between the verification and distributed systems worlds. On one hand, the distributed systems community would like to use mainstream programming languages to specify their protocols, such as C or Java. On the other hand, each new verification project tend to come with a new language, which implies that in order to specify the protocol, one has to first learn that language. In order to bridge this gap, as part of our future work, we plan to extend our ecosystem with transformers to turn C code into Velisarios code, and vice versa. We believe that such transformers would contribute to the popularization of formal verification ecosystems such as ours, as well as to make life easier for both, distributed systems as well as verification engineers/scientists.

Acknowledgments

This work is partially supported by the University of Luxembourg - SnT and by the Fonds National de la Recherche Luxembourg (FNR) through PEARL grant FNR/P14/8149128.

References

[1] URL: https://blog.cosmos.network/consensus-compare-casper-vs-tendermint-6df154ad56ae.
[2] URL: https://github.com/ethereum/cbc-casper/wiki/FAQ.
[3] URL: https://github.com/hyperledger-archives/fabric/wiki/Consensus.
[4] URL: https://steemit.com/cryptocurrency/@basiccrypto/almost-everything-you-wanted-to-know-about-neo-part-1-of-2.
[5] Ittai Abraham, Dahli Malkhi, Kartik Nayak, Ling Ren, and Alexander Spiegelman. "Solida: A Blockchain Protocol Based on Reconfigurable Byzantine Consensus". 2018.
[6] Ittai Abraham, Dahlia Malkhi, Kartik Nayak, Ling Ren, and Alexander Spiegelman. "Solidus: An Incentive-compatible Cryptocurrency Based on Permissionless Byzantine Consensus". In: *CoRR* abs/1612.02916 (2016). arXiv: 1612.02916.
[7] Pierre-Louis Aublin, Sonia Ben Mokhtar, and Vivien Quéma. "RBFT: Redundant Byzantine Fault Tolerance". In: *ICDCS 2013*. IEEE Computer Society, 2013, pp. 297–306.
[8] Alysson Neves Bessani, João Sousa, and Eduardo Adílio Pelinson Alchieri. "State Machine Replication for the Masses with BFT-SMART". In: *DSN 2014*. IEEE, 2014, pp. 355–362.
[9] Mark Bickford. "Component Specification Using Event Classes". In: *CBSE 2009*. Vol. 5582. LNCS. Springer, 2009, pp. 140–155.
[10] Mark Bickford, Robert L. Constable, and Vincent Rahli. "Logic of Events, a framework to reason about distributed systems". In: *Languages for Distributed Algorithms Workshop*. 2012.
[11] Miguel Castro. "Practical Byzantine Fault Tolerance". Also as Technical Report MIT-LCS-TR-817. Ph.D. MIT, Jan. 2001.
[12] Miguel Castro and Barbara Liskov. "Practical Byzantine Fault Tolerance". In: *OSDI 1999*. USENIX Association, 1999, pp. 173–186.

	Running code	Byz. (synch.)	Byz. (asynch.)
IronFleet [20]; EventML [28]; Verdi [33]; PSync [17]	✓	✗	✗
HO-model [14]; PVS [31]	✗	✓	✗
Event-B [24]	✓/✗	✓	✗
IOA [11]; TLA$^+$ [25]; ByMC [23]	✗	✓	✓
Velisarios [29]	✓	✓	✓

Figure 1. Comparison with related work

[13] Miguel Castro and Barbara Liskov. “Practical byzantine fault tolerance and proactive recovery”. In: *ACM Trans. Comput. Syst.* 20.4 (2002), pp. 398–461.

[14] Bernadette Charron-Bost, Henri Debrat, and Stephan Merz. “Formal Verification of Consensus Algorithms Tolerating Malicious Faults”. In: *SSS 2011*. Vol. 6976. LNCS. Springer, 2011, pp. 120–134.

[15] *The Coq Proof Assistant*. URL: http://coq.inria.fr/.

[16] Christian Decker, Jochen Seidel, and Roger Wattenhofer. “Bitcoin meets strong consistency”. In: *ICDCN 2016*. ACM, 2016, 13:1–13:10.

[17] Cezara Dragoi, Thomas A. Henzinger, and Damien Zufferey. “PSync: a partially synchronous language for fault-tolerant distributed algorithms”. In: *POPL 2016*. ACM, 2016, pp. 400–415.

[18] Cezara Dragoi, Thomas A. Henzinger, and Damien Zufferey. “The Need for Language Support for Fault-Tolerant Distributed Systems”. In: *SNAPL 2015*. Vol. 32. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2015, pp. 90–102.

[19] Rachid Guerraoui, Nikola Knezevic, Vivien Quéma, and Marko Vukolic. “The next 700 BFT protocols”. In: *European Conference on Computer Systems, Proceedings of the 5th European conference on Computer systems, EuroSys 2010, Paris, France, April 13-16, 2010*. ACM, 2010, pp. 363–376.

[20] Chris Hawblitzel, Jon Howell, Manos Kapritsos, Jacob R. Lorch, Bryan Parno, Michael L. Roberts, Srinath T. V. Setty, and Brian Zill. “IronFleet: proving practical distributed systems correct”. In: *SOSP 2015*. ACM, 2015, pp. 1–17.

[21] Rüdiger Kapitza, Johannes Behl, Christian Cachin, Tobias Distler, Simon Kuhnle, Seyed Vahid Mohammadi, Wolfgang Schröder-Preikschat, and Klaus Stengel. “CheapBFT: resource-efficient byzantine fault tolerance”. In: *EuroSys ’12*. ACM, 2012, pp. 295–308.

[22] Eleftherios Kokoris-Kogias, Philipp Jovanovic, Nicolas Gailly, Ismail Khoffi, Linus Gasser, and Bryan Ford. “Enhancing Bitcoin Security and Performance with Strong Consistency via Collective Signing”. In: *USENIX Security Symposium*. USENIX Association, 2016, pp. 279–296.

[23] Igor V. Konnov, Marijana Lazic, Helmut Veith, and Josef Widder. “A short counterexample property for safety and liveness verification of fault-tolerant distributed algorithms”. In: *POPL 2017*. ACM, 2017, pp. 719–734.

[24] Roman Krenický and Mattias Ulbrich. *Deductive Verification of a Byzantine Agreement Protocol*. Tech. rep. 2010-7. Karlsruhe Institute of Technology, Department of Computer Science, 2010.

[25] Leslie Lamport. “Byzantizing Paxos by Refinement”. In: *DISC 2011*. Vol. 6950. LNCS. Springer, 2011, pp. 211–224.

[26] Loi Luu, Viswesh Narayanan, Chaodong Zheng, Kunal Baweja, Seth Gilbert, and Prateek Saxena. “A Secure Sharding Protocol For Open Blockchains”. In: *CCS 2016*. ACM, 2016, pp. 17–30.

[27] Rafael Pass and Elaine Shi. “Hybrid Consensus: Efficient Consensus in the Permissionless Model”. In: *DISC 2017*. Vol. 91. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017, 39:1–39:16.

[28] Vincent Rahli, David Guaspari, Mark Bickford, and Robert L. Constable. “EventML: Specification, Verification, and Implementation of Crash-Tolerant State Machine Replication Systems”. In: *SCP* (2017).

[29] Vincent Rahli, Ivana Vukotic, Marcus Volp, and Paulo Verissiomo. “Velisarios: Byzantine Fault-Tolerant Protocols Powered by Coq”. ESOP. 2018.

[30] A. W. Roscoe, C. A. R. Hoare, and Richard Bird. *The Theory and Practice of Concurrency*. Upper Saddle River, NJ, USA: Prentice Hall PTR, 1997.

[31] Ulrich Schmid, Bettina Weiss, and John M. Rushby. “Formally Verified Byzantine Agreement in Presence of Link Faults”. In: *ICDCS*. 2002, pp. 608–616.

[32] Giuliana Santos Veronese, Miguel Correia, Alysson Neves Bessani, Lau Cheuk Lung, and Paulo Veríssimo. “Efficient Byzantine Fault-Tolerance”. In: *IEEE Trans. Computers* 62.1 (2013), pp. 16–30.

[33] James R. Wilcox, Doug Woos, Pavel Panchekha, Zachary Tatlock, Xi Wang, Michael D. Ernst, and Thomas E. Anderson. “Verdi: a framework for implementing and formally verifying distributed systems”. In: *PLDI 2015*. ACM, 2015, pp. 357–368.