Byzantine-Resistant Consensus based on a Novel Approach to Intrusion Tolerance *

Miguel Correia[1] Nuno Ferreira Neves[1] Lau Cheuk Lung[2] Paulo Veríssimo[1]

[1]*Faculdade de Ciências da Universidade de Lisboa*
Campo Grande, Bloco C5, 1749-016 Lisboa, Portugal
[2]*Pontifícia Universidade Católica do Paraná*
R. Imaculada Conceição, 1155, Prado Velho, Brasil, CEP: 80215-901
{*mpc,nuno,pjv*}*@di.fc.ul.pt* *lau@ppgia.pucpr.br*

1 Introduction

Intrusion tolerance has been raising a good deal of interest in the security and dependability communities [6]. It is a sad but daily fact that networked computer systems often have vulnerabilities that can be exploited by malicious hackers. The idea of intrusion tolerance is to acknowledge this fact and to build systems that tolerate a number of faults, including attacks and intrusions. In other words, an intrusion-tolerant (IT) system has to deliver its service according to its specification despite the failure of some of its components.

We have been exploring a new approach to build IT distributed systems. We consider a hybrid architecture: most of the systems is assumed to be time-free (asynchronous) and vulnerable to attacks and intrusions; however, there is a subsystem with stronger properties, which can be used to assist the execution of the applications and protocols. This subsystem is called the Trusted Timely Computing Base (TTCB). It has been designed to execute a small number of simple low-level services in a secure and timely way [3]. Therefore, we have a system that is basically asynchronous and insecure, with a 'small' subsystem that is synchronous (real-time) and secure. The TTCB is an example of a secure and timely *wormhole* [5].

The objective of this abstract is to show some of the benefits of our approach by presenting a simple IT consensus protocol. Consensus is known to be a fundamental problem in distributed systems. Using our approach we manage to implement a consensus protocol with interesting properties. The protocol is a simplified version of the protocol presented in a recent report [2].

*This work was partially supported by the FCT through the LASIGE Laboratory and projects POSI/1999/CHS/33996 (DEFEATS) and POSI/CHS/39815/2001 (COPE).

2 The Consensus Protocol

The consensus protocol is executed by a number of processes running in different machines. Processes communicate mostly using the 'normal' asynchronous and insecure network, called payload network. Processes can be attacked and fail, by stopping to interact or by trying to break the protocol, alone or in collusion with other failed processes. Nevertheless, we assume that at most one third of the processes less one can fail, i.e., $f = \lfloor \frac{n-1}{3} \rfloor$ (n is the total number of processes, f is the number of failed processes). The communication between correct (i.e., not failed) processes is assumed to be secure (this is simple to enforce with symmetric cryptography; see [2] for details).

Processes use only one of the TTCB services in runtime: the Trusted Block Agreement service (TBA). This service executes securely 'agreement' operations in a broad sense. In this paper, TBA makes a vote on the values given by the processes and returns three items: the value proposed by most processes, the list of the processes that proposed that value, and the list of the processes that proposed any value. The proposed values have a limited size of 160 bits.

The consensus protocol is defined in terms of three properties:

- *Validity.* If all correct processes propose the same value *v*, then any correct process that decides, decides *v*.
- *Agreement.* No two correct processes decide differently.
- *Termination.* Every correct process eventually decides.

The protocol is presented in Algorithm 1. A process is identified before the TTCB by an *eid* identifier. Each

process engages in an execution of the protocol by calling the function *consensus* with three parameters: the *value* it proposes, a list *elist* with the *eids* of the processes involved, and a *tstart* passed to the first execution of the TBA (line 7).

The protocol executes basically the following way (a more detailed description can be found in [2]). Each process starts by sending its value to all other processes (line 5). Then it enters a loop that calls TBA once per round (lines 6-15). Each (correct) process gives TBA a 'hash' of its value. A hash is the result of a cryptographic hash function, a one-way function that obtains a fixed-size digest of its input, with the property that it is computationally infeasible to find two inputs that give the same output.

The protocol can terminate in two situations. If $f + 1$ processes propose the same hash for the TBA (line 15) then the value decided is the value corresponding to this hash. This is the valued proposed by the correct processes in case all proposed the same. The second termination situation is when $2f+1$ processes managed to propose for the TBA but no $f + 1$ proposed the same (line 13). In this case the protocol can be sure that the correct processes did not propose the same so it decides on a default value (line 14).

Algorithm 1 Consensus protocol

```
1  function consensus(elist, tstart, value)
2  hv ←⊥;                                  {hash of the value decided}
3  bag ←∅;                                 {bag of received messages}
4  round ←0;                                          {round number}
5  multicast(elist, tstart, value) to processes in elist;

6  repeat
7    outp ←TTCB_propose(eid, elist, tstart,
     TBA_MAJORITY, Hash(value));
8    repeat
9      outd ←TTCB_decide(outp.tag);
10   until (outd.error ≠ TBA_RUNNING);
11   tstart ←tstart + T * func(α, round);
12   round ←round+1;
13   if (2f + 1 processes proposed) and (less than f + 1
     processes proposed the same hash) then
14     decide default-value;
15 until (f + 1 processes proposed the same hash);
16 hv ←outd.value;

17 when receive message M
18 bag ←bag ∪ {M};

19 when (hv ≠ ⊥) and (∃M∈bag : Hash(M.value) = hv)
20 decide M.value;
```

3 Benefits

This simple IT consensus protocol uses the TTCB TBA service to execute securely an agreement step of the protocol. Besides its simplicity, the first benefit of the protocol is to have low time and message complexities [2]. In the best case, the protocol runs in a single round. The number of messages sent is clearly low.

The second benefit of the protocol is not relying on public-key cryptography, usually three orders of magnitude slower than symmetric cryptography. This has been shown to be an important bottleneck in many IT protocols [1].

It has been shown that no deterministic algorithm can solve consensus in an asynchronous system if a single process can crash [4]. Our protocol is not bound by this result since our system is not fully asynchronous (the TTCB is synchronous). However, termination depends on a weak synchrony assumption on the local behavior of correct processes: we have to assume that eventually $2f + 1$ processes manage to propose to the TBA (lines 12-13).

These benefits of the protocol can give the reader an intuition about the practical interest of our approach to intrusion tolerance. Besides consensus, we have been designing a suite of efficient IT protocols, including reliable multicast, membership and atomic multicast.

References

[1] M. Castro and B. Liskov. Practical Byzantine fault tolerance. In *Proceedings of the Third Symposium on Operating Systems Design and Implementation*, pages 173–186, Feb. 1999.

[2] M. Correia, N. F. Neves, L. C. Lung, and P. Veríssimo. Low complexity Byzantine-resilient consensus. DI/FCUL TR 03–25, Department of Informatics, University of Lisbon, August 2003.

[3] M. Correia, P. Veríssimo, and N. F. Neves. The design of a COTS real-time distributed security kernel. In *Proceedings of the Fourth European Dependable Computing Conference*, pages 234–252, Oct. 2002.

[4] M. J. Fischer, N. A. Lynch, and M. S. Paterson. Impossibility of distributed consensus with one faulty process. *Journal of the ACM*, 32(2):374–382, Apr. 1985.

[5] P. Veríssimo. Uncertainty and predictability: Can they be reconciled? In *Future Directions in Distributed Computing*, volume 2584 of *Lecture Notes in Computer Science*, pages 108–113. Springer-Verlag, 2003.

[6] P. E. Veríssimo, N. F. Neves, and M. P. Correia. Intrusion-tolerant architectures: Concepts and design. In R. Lemos, C. Gacek, and A. Romanovsky, editors, *Architecting Dependable Systems*, volume 2677 of *Lecture Notes in Computer Science*, pages 3–36. Springer-Verlag, 2003.